μDX®

Série 200

CONTROLADOR PROGRAMÁVEL

Manual de Utilização

μDX 200 & μDX201

REVISÃO 3.55

Nov/2024

DEXTER Indústria e Comércio de Equipamentos Eletrônicos Ltda.

Av. Pernambuco, 1328 Cjs.307/309 - Porto Alegre - RS - Fones: (51) 3208-0533, 99963-0370

Página Internet: www.dexter.ind.br - E-mail: dexter@dexter.ind.br

µDX 200 & µDX201

Este manual não foi revisado para se adequar a nova norma ortográfica, e nem existe previsão para tal. Com tantas questões relevantes neste País (como o desmatamento na Amazônia, que parece só será resolvido quando a última árvore tombar) há preocupação em modificar a língua pátria, sob justificativas claramente discutíveis. Portanto, espero que o leitor entenda e até aprecie os possíveis tremas que venha a encontrar. Variáveis usadas no programa PG não podem ser acentuadas (de forma similar a endereços de e-mail) e, por isso, os acentos foram suprimidos.

Publicação

DEXTER Indústria e Comércio de Equipamentos Eletrônicos Ltda

Revisão

3.55

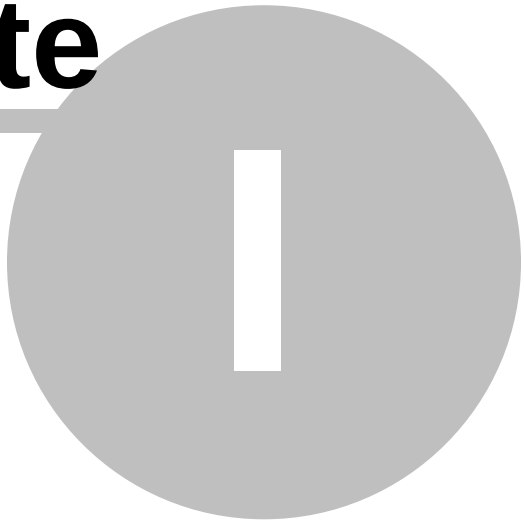
Contato

Claudio Richter

dexter@dexter.ind.br

Conteúdo

Parte



Controlador µDX200

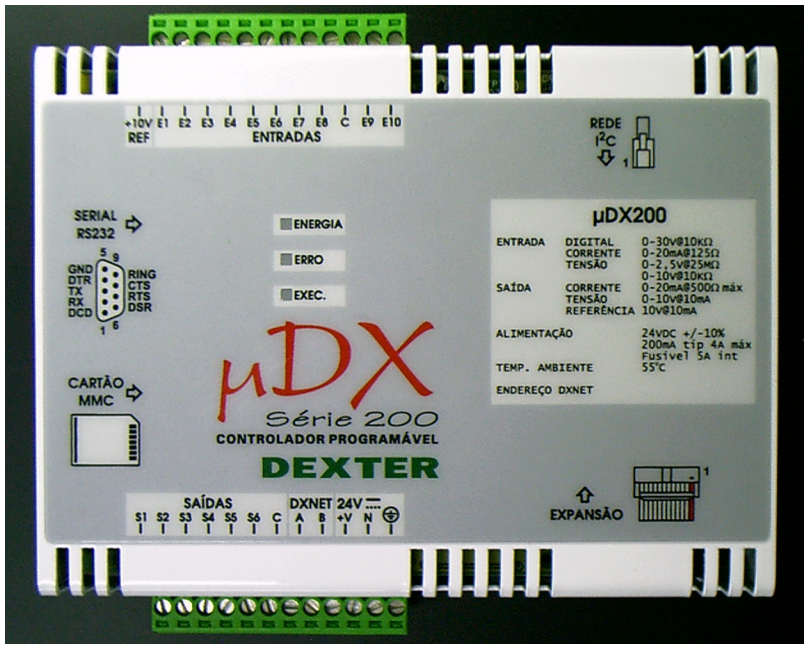
O Controlador Programável µDX200 é baseado em um microcontrolador de 16 bits com muitos periféricos incorporados, e possui uma série de proteções em suas entradas e saídas. Com isso, o µDX200 está preparado para enfrentar ambientes industriais, onde ruído elétrico e temperaturas extremas não são incomuns. Sua caixa metálica, além de conferir robustez ao equipamento, forma uma blindagem elétrica eficiente.

A rede de comunicação DXNET permite a comunicação entre controladores µDX200 até 1500 metros distantes. E permite conectar até 32 dispositivos sem a necessidade de amplificação. Note que existem dois jumpers para ligar uma terminação resistiva na rede DXNET. Estes jumpers devem ser ligados apenas nos dois dispositivos extremos da rede. Os demais dispositivos devem ficar sem terminação. A terminação resistiva é importante para melhorar a forma de onda na rede DXNET (evitando onda refletida), permitindo obter até 1500 metros de alcance.

A rede de comunicação I^2C possibilita a conexão de sensores de temperatura e umidade até 1000 metros distantes do Controlador µDX200.

As entradas analógicas são protegidas contra tensões superiores a 30V, assim como as saídas analógicas. Isso protege contra espúrios de alta tensão nestes pinos (embora não proteja estas entradas e saídas no caso de aplicação de tensões superiores a 30V continuamente). Também as entradas digitais E9 e E10 possuem esta proteção.

A saída de referência +10V, embora não seja precisa quanto ao seu valor absoluto (se admite uma variação de até ± 5%), é muito estável. Com isso se aconselha seu uso para, por exemplo, alimentar um potenciômetro ligado a uma das entradas analógicas. Lembre-se apenas de respeitar sua corrente máxima de saída de 10mA.



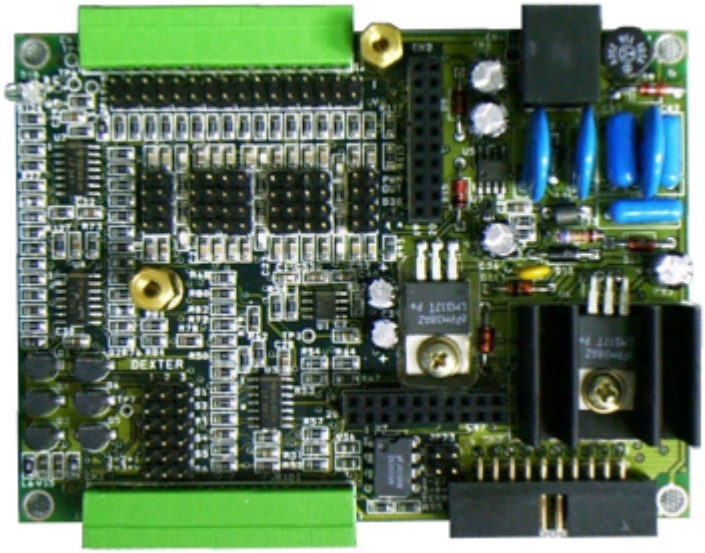
A foto mostra os conectores de entrada e saída do Controlador µDX200. Note que tanto as entradas quanto as saídas do controlador possuem conectores de engate rápido, o que facilita manutenções. No conector superior temos:

- **+10V REF → Saída de 10V de referência para circuitos externos.**
- **E1 a E8 → Entradas analógicas (0-2,5V, 0-10V, 0-20mA).**
- **C → Comum (GND) para entradas e +10V REF.**
- **E9 e E10 → Entradas digitais rápidas.**

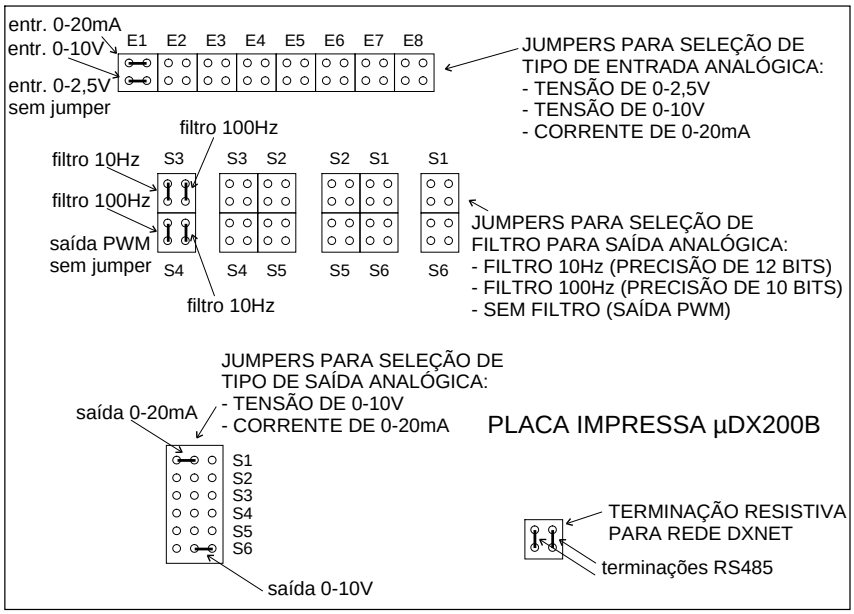
Já o conector inferior possui as seguintes conexões:

S1 a S6 → **Saídas analógicas (PWM, 0-10V, 0-20mA).**
C → **Comum (GND) para saídas analógicas.**
DXNET → **Conexão A e B para RS485 (rede DXNET).**
+V → **Entrada de alimentação elétrica positiva (+24V).**
N → **Entrada de neutro (GND) da alimentação elétrica.**
⏚ → **Aterramento.**

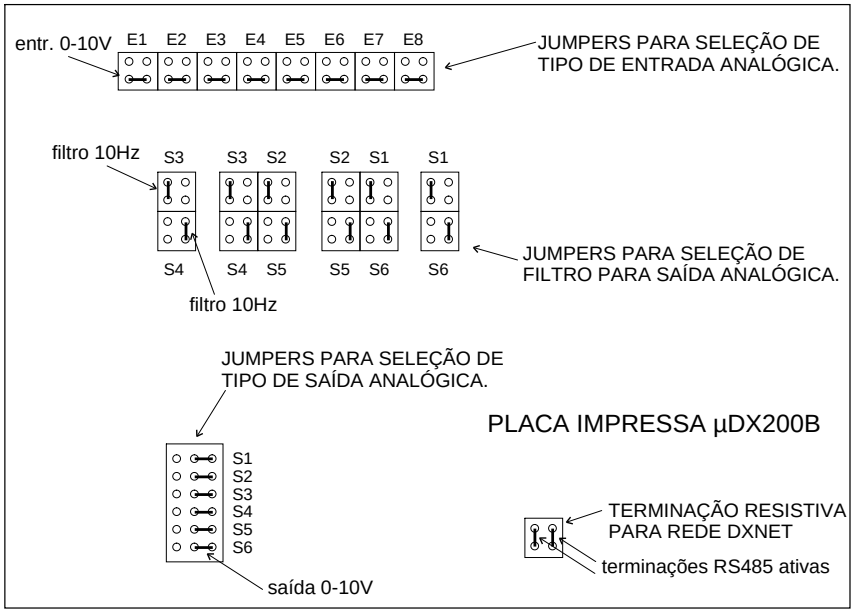
Seleção de Jumpers



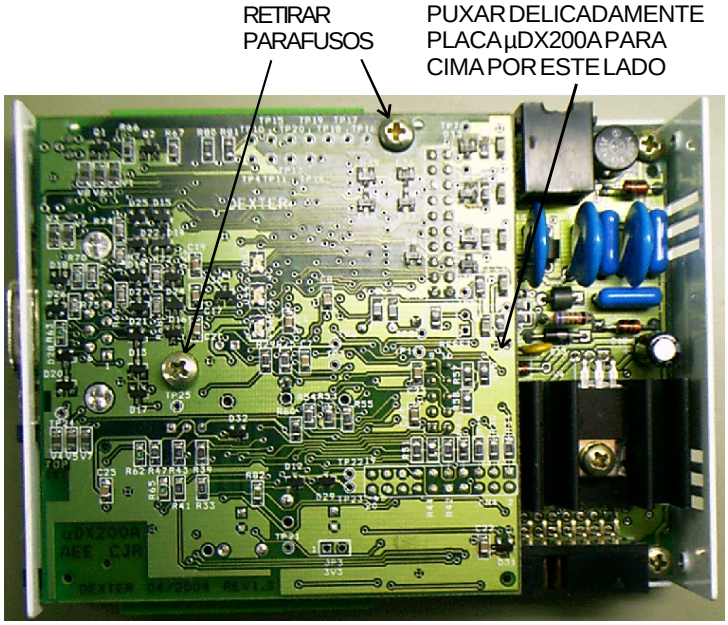
O Controlador Programável µDX200 é fornecido com as entradas e saídas analógicas calibradas para as diversas escalas possíveis, e selecionado para entradas e saídas analógicas em 0-10V, com resolução de 12 bits. Entretanto, é possível selecionar outras escalas para as entradas e saídas analógicas. Para isso, além de modificar a escala na janela de "Configurações de Hardware", é preciso posicionar os jumpers internos existentes na placa impressa µDX200B do controlador. A ilustração abaixo mostra a referida placa, e a posição dos jumpers:



O µDX200 é fornecido para entradas analógicas 0-10V, saídas analógicas 0-10V, filtro para saídas analógicas em 10Hz, e terminação para rede DXNET ativa. A configuração default de jumpers é a ilustrada abaixo:



Para abrir a caixa metálica do µDX200 retire os dois parafusos (fenda cruzada) existentes nas laterais da caixa, e force levemente as laterais para que se afastem dos encaixes que a prendem ao fundo da caixa. A seguir, é necessário retirar dois parafusos que prendem a placa µDX200A à placa µDX200B, que contém os jumpers. A placa µDX200A é encaixada sobre a placa µDX200B. Após a retirada dos parafusos (veja ilustração abaixo), puxe com delicadeza a placa µDX200A, de forma a ter acesso aos jumpers:



Para recolocar a placa µDX200A posicione-a usando a parede frontal da caixa metálica como guia e pressione-a para baixo com delicadeza. Cuidado para não forçar os conectores que interligam as placas impressas µDX200A e µDX200B.

Leds

O controlador µDX200 possui três leds (light emitting diode), ou indicadores visuais. São eles:

- **Led de Energia (verde).**
- **Led de Erro (vermelho).**
- **Led de Execução (vermelho).**

O led de **Energia** tem como finalidade apenas indicar que o controlador está energizado. Como este led está ligado diretamente a tensão de alimentação do µDX200, seu brilho varia levemente conforme a tensão utilizada (entre 11,0 e 26,4Vdc).

O led de **Erro** permite detectar diversos erros, conforme listado abaixo. Note que a indicação de erro é feita pelo número de piscadas deste led em um ciclo de aproximadamente 2,5 segundos (cada piscada dura cerca de 100ms). A única exceção é no caso de erro no cristal oscilador de alta freqüência (8 MHz). Neste caso o led de erro fica piscando constantemente em uma taxa elevada (cerca de 3 vezes por segundo). Neste caso o CLP deverá ser encaminhado à Dexter para manutenção.

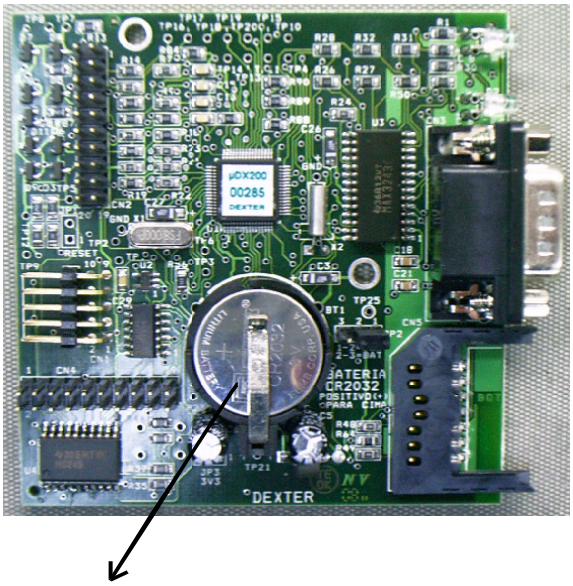
1 piscada	→	Erro de CRC no programa aplicativo. O programa aplicativo existente no controlador está corrompido.
2 piscadas	→	Erro de versão do programa aplicativo. O CLP não suporta a versão do programa aplicativo carregado.
3 piscadas	→	Não foi encontrado arquivo de log no cartão MMC, ou o arquivo está cheio (totalmente preenchido).
4 piscadas	→	Cartão MMC não suportado ou com formatação incorreta.

Por fim, o led de **Execução** indica se o programa aplicativo está sendo executado ou está parado, e ainda se o programa aplicativo está controlando a porta serial RS232 (modo normal) ou não (modo Programação). Seus possíveis estados são:

Led desligado	→	Programa parado.
Led ligado	→	Programa executando, modo Programação. (protocolo nativo do µDX200 controla porta RS232).
Led piscando	→	Programa executando, modo Normal. (programa aplicativo controla porta RS232).

Troca de bateria

A troca de bateria é feita na placa µDX200A (placa superior). A bateria do µDX200 tem uma durabilidade de 5 anos. Note que o próprio software PG indica o valor de tensão desta bateria na janela do Compilador. Valores abaixo de 2,5V indicam bateria descarregada. O µDX200 amostra o valor de bateria apenas quando não estiver energizado, e esta amostragem é feita a cada minuto. Assim, para obter uma leitura de bateria atualizada desenergize o controlador µDX200 por, pelo menos, dois minutos. A bateria é do tipo botão, não-recarregável, tensão de 3V, modelo CR2032. Para orientação a respeito de como abrir o controlador e ter acesso as placas impressas consulte o parágrafo anterior - Seleção de Jumpers.



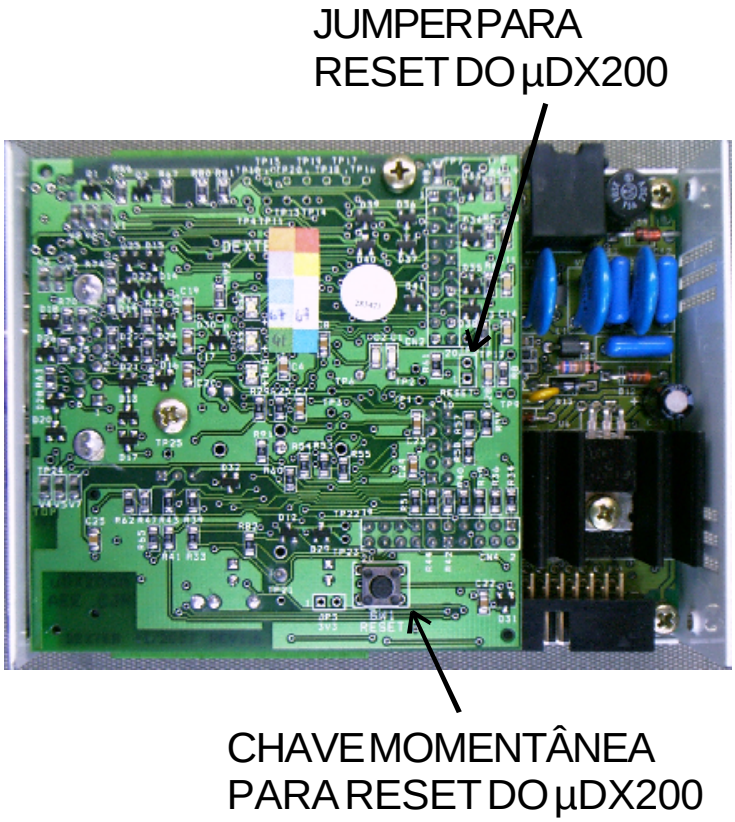
Bateria CR2032

A troca de bateria não implica em perda do programa aplicativo armazenado no controlador programável, pois o mesmo é armazenado em memória não-volátil (FLASH). Entretanto, os dados do programa (variáveis e nodos) e seu status (em execução ou parado) serão perdidos caso a operação de troca da bateria leve mais que aproximadamente um minuto (durante este tempo os dados são mantidos por capacitores).

Atenção: *Para retirar a bateria de seu soquete é necessário levantá-la suavemente (alguns soquetes possuem uma aba que deve ser baixada, permitindo a retirada da bateria). Evite utilizar ferramentas metálicas nesta operação para evitar curtos. Caso não esteja seguro quanto a esta operação envie o controlador para que a Dexter efetue a troca de bateria. Note que o terminal positivo da bateria é visível após a instalação da bateria (ou seja, o + é para cima).*

Reset Físico

Em algumas situações pode ocorrer do controlador µDX200 perder completamente a comunicação via porta serial RS232, impedindo assim qualquer acesso via software PG. Isso pode ocorrer, por exemplo, se o programa aplicativo utiliza a porta serial e inicializa a mesma com parâmetros errados. Uma vez que o controlador entre em modo **Normal** (e, portanto, o programa aplicativo passe a comandar a porta serial) não mais é possível retornar ao modo **Programação**, pois os parâmetros de configuração da porta serial não são consistentes. Neste caso, é possível efetuar-se um reset físico do controlador (hard reset). Para isso abra o controlador com cuidado (veja instruções em Seleção de Jumpers). Em controladores recentes existe uma pequena chave momentânea para efetuar este reset. Caso esta chave não esteja presente em seu equipamento é preciso efetuar um breve curto-circuito entre os terminais do conector mostrado na foto abaixo. Certifique-se que o CLP está desenergizado antes de abrí-lo, de forma a evitar acidentes. Ao reenergizar o µDX200 ele irá piscar momentaneamente os dois leds vermelhos em seu painel (Exec. e Erro), indicando hard reset, e irá comutar para modo Programação. Este estado irá perdurar por um minuto (caso o programa aplicativo esteja rodando), permitindo o acesso do PG neste período. Assim, é possível parar o programa aplicativo e substituí-lo por uma versão corrigida.



Especificações Técnicas

Características Gerais

- 341 instruções, incluindo lógica aritmética inteira de 16 e 32 bits.
- Aritmética em ponto flutuante.
- Mais de 1000 blocos de programação.
- Mais de 300 variáveis de 16 bits.
- Mais de 2000 nodos.
- Execução do programa em modo de paralelismo lógico.
- Ciclo de execução do programa aplicativo abaixo de 1ms.
- "Watch-Dog Timer" incorporado.
- 8 entradas analógicas, que podem ser usadas como entradas digitais.
- 6 saídas analógicas, que podem ser usadas como saídas digitais.
- 2 entradas de contagem rápida (até 8KHz).
- Até 512 I/Os via módulos de Expansão de Entradas/Saídas (µDX210).
- Relógio e calendário de tempo real (com previsão de ano bissexto).
- Dimensões reduzidas: 115 x 86 x 30 mm.
- Protegido contra transientes elétricos.
- Acondicionado em gabinete metálico, muito resistente.
- Bateria interna: pilha CR2032 com durabilidade de 5 anos.
- Temperatura de operação: 0°C até 55°C.
- Rede DXNET para 1500 metros.
- Rede I^2C para 1000 metros.
- Slot para cartão MMC (registro de eventos). Cartões de 64M a 2G.
- Comunicação serial de 110 a 115200bps (porta RS232C completa).

  Tempos de timeout default do µDX200 dependem do baud rate utilizado: (cerca de 5,5x o tempo de um caracter)

110 bps	450 ms
300 bps	165 ms
600 bps	87 ms
1200 bps	41 ms
2400 bps	22 ms
4800 bps	11 ms
9600 bps	5,5ms
19200 bps	2,8ms
38400 bps	1,7ms
57600 bps	1,2ms
115200 bps	0,8ms

Entradas Analógicas (E1 a E8)

Escala de 0-2,5V Resolução = 610,5µV (12 bits)
Resistência de entrada = 400KΩ
Precisão melhor que 0,15% do fundo de escala
Máxima Tensão de entrada = 30V

Escala de 0-10V Resolução = 2,442mV (12 bits)
Resistência de entrada = 10KΩ
Precisão melhor que 0,15% do fundo de escala
Máxima Tensão de entrada = 30V

Escala de 0-20mA Resolução = 4,884µA (12 bits)
Resistência de entrada = 125Ω
Precisão melhor que 0,15% do fundo de escala
Máxima Corrente de entrada = 30mA

Saídas Analógicas (S1 a S6)

Escala de 0-10V Resolução = 2,442mV (12 bits)
Corrente máxima de saída = 10mA
Precisão melhor que 0,3% do fundo de escala

Escala de 0-20mA Resolução = 4,884µA (12 bits)
Resistência de carga ≤ 500Ω
Precisão melhor que 0,3% do fundo de escala

Referência de Tensão (+10V REF)

Tensão nominal = 10V ± 5%
Estabilidade térmica típica = 100ppm/°C
Corrente de saída máxima = 10mA

Entradas Digitais Rápidas (E9 e E10)

Freqüência Máxima = 8KHz
Resistência de Entrada = 10KΩ
Mínima Tensão de entrada = 3V
Máxima Tensão de entrada = 30V

Alimentação Elétrica (+V e N)

Tensão de operação = 11,0 a 26,4Vdc *****
Corrente Típica (sem Expansões) = 150mA
Corrente Máxima (com 32 Expansões µDX210, em 24Vdc) = 4A

***** Equipamento projetado para operar em 24Vdc ±10%. Para funcionamento em 12V existem as seguintes restrições:

- É necessário empregar Expansões µDX210-12 (relés para 12Vdc).
- Limitação de 16 Expansões µDX210-12 (em vez de 32 µDX210).
- Saídas analógicas perdem sua função (não permitem saída em 0-10V ou 0-20mA) mas podem ser utilizadas como saídas digitais ou PWM.
- Saída de referência +10V REF perde estabilidade, ficando sujeita as flutuações da alimentação do µDX200. Também seu valor fica abaixo dos 10V nominais.

Atenção: *As entradas do Controlador Programável µDX200 não possuem isolação galvânica e, portanto, não podem ser ligadas em fontes de sinal com referências distintas, e muito menos a rede elétrica (127 ou 220 Vac).*

Nota: *Até a versão 2.08 de firmware do µDX200 não havia filtragem em freqüência nas entradas digitais E9 e E10. Isso poderia ocasionar problemas, caso fosse aplicado sinais com altas freqüências (>100KHz). Para evitar esta possibilidade foi incluída filtragem, limitando a freqüência máxima nestas entradas em 8KHz. Com isso, a especificação destas entradas foi modificada, já que antes eram admitidos sinais até 10KHz.*

Versões de Software

Firmware do µDX200

Trata-se do software interno do controlador µDX200. Note que este software pode ser atualizado, bastando para isso remeter a unidade para a Dexter. A única despesa será com o transporte, pois a atualização é gratuita (até o limite de duas atualizações por controlador).

V 1.00	15/10/2004	Versão inicial.
V 1.01	08/11/2004	Corrigida falha na int. externa.
V 1.10	08/11/2004	Implementado bloco conversor Longint->Int.
V 1.20	12/11/2004	Mudança do endereco de start para $8C00.
V 1.21	16/11/2004	Corrigido o problema de leitura de I/O quando ocorria interrupções justamente no momento após o clock do I/O. Bloco Delay_N_ciclos implementado.
V 1.22	05/12/2004	Alterada macro de I/O para fazer OR com o estado das entradas e Nodo_In em vez de MOV.
V 1.23	06/12/2004	Status retorna número e CRC do aplicativo (útil para comparar prog no PG com prog no uDX).
V 1.24	07/12/2004	Nodos sistema E9,E10 estavam invertidos.
V 1.25	09/12/2004	Pequenos ajustes na leitura de FLASH.
V 1.26	10/12/2004	Permite ler e forcar nodos de I/O diretamente usando números acima de 30000 (comandos DXNET 1 e 2). Retorna quantidade de I/O varridos pela CPU no status.
V 1.27	14/12/2004	Correções no comandos DXNET para leitura de nodos.
V 1.28	16/12/2004	Implementado bloco String POS.
V 1.29	17/12/2004	Bloco ASCII->Int/Lint.
V 1.30	21/12/2004	Nodo de erro aritmético (16), alteração no bloco Longint->Int.
V 1.31	23/12/2004	Início da implementação de MMC.
V 1.32	02/01/2005	Blocos MMC implementados: Int, CrLf, String, Longint. Leitura do MMC via RS232 apenas para endereço local (modo programação).
V 1.33	12/01/2005	Correção Sqrt longint->int.
V 1.34	15/01/2005	Correção na multiplicação de inteiros: saída de erro quando exceder limite 32767.
V 1.35	17/01/2005	Correção final da multiplicação de inteiros. Correção na divisão Longint->Int.
V 1.36	18/01/2005	Implementados a soma e subtração de word.
V 1.37	18/01/2005	Implementadas as comparações de word (> < =)
V 1.38	18/01/2005	Correção do bloco Delay N Ciclos.
V 1.39	19/01/2005	Correção bloco String POS.
V 1.40	19/01/2005	Correção no bloco aritmético: shifts estavam com erros graves de Cin/Cout.
V 1.41	20/01/2005	Correção comparação de word.
V 1.42	20/01/2005	Correção da leitura de VBAT.
V 1.43	21/01/2005	MMC128 ok. Controle de semáforo para otimizar uso da RAM de MMC.
V 1.44	24/01/2005	Corrigido blocos DXNET de consulta/forçamento de Longint.
V 1.45	25/01/2005	Comportamento do nodo de erro do MMC alterado. Incluído nodo de MMC Cheio. Bloco TX RS232 melhorado. Bloco MMC com funcao FLUSH. Bloco leitura de caracter->Inteiro.
V 1.46	26/01/2005	Tx RS232 Inteiro e Longint corrigidos para caso NumByte<tamanho do valor convertido.
V 1.47	26/01/2005	TX 232 com CRC, nodo de "fim de transmissão RS232", PG_configbits incrementado com controle de Half/Full-Duplex e controle de RTS para facilitar uso de adaptador RS485.

V 1.48	27/01/2005	Resolvido bug do TX RS232 Longint.
V 1.49	27/01/2005	Resolvido bug TX RS232 Longint.
V 1.50	27/01/2005	Bug TX CRC resolvido, Nodo TxEmpty corrigido.
V 1.51	27/01/2005	Número de bytes calculados no TX CRC corrigido.
V 1.52	27/01/2005	Loop do TX CRC corrigido.
V 1.53	28/01/2005	Bloco de cálculo de CRC/FCS em tabelas. Mudança do tipo de parâmetro do TX CRC. Implementado comando 16 tipo 7.
V 1.54	28/01/2005	Corrigido bloco TX RS232 devido ao uso de variável em TX CRC em vez de valor imediato.
V 1.55	28/01/2005	Corrigido bloco cálculo CRC.
V 1.56	28/01/2005	Corrigido nodo Zero no bloco de cálculo de CRC.
V 1.57	01/02/2005	Corrigido erro na inicializacao de timer.
V 1.58	10/03/2005	Recepção de comandos DXNET uDX100 agora reconhece grupo+endereço usando byte low de ENDEREÇODX.
V 1.59	15/04/2005	Correção do nodo RTS.
V 1.60	06/05/2005	Correção da leitura de tabela tipo byte, zerando o byte high do destino.
V 1.61	10/05/2005	Implementada a temporização de RTS, controle de TX por CTS e imagem do PG_CONFIGBITS em RAM.
V 1.62	11/05/2005	Implementado bloco SWAP (math inteiro) e blocos de TX RS232 enviando byte e word binários.
V 1.63	16/05/2005	Correção de erros da versão 1.62.
V 1.64	17/05/2005	Correção da DXNET quando ocorre múltiplos uDXs com mesmo endereço na rede.
V 1.65	18/05/2005	Removida a alteração na DXNET da versão 1.64.
V 1.66	19/05/2005	Inserido o teste de endereço válido para resultado de alguns blocos. Limite=0870h Utilizado o bloqueio dos pinos de RX para evitar entrada de espúrios.
V 1.67	21/05/2005	Assume configuração da RS232, indicada no header do PG, ao executar comando RESET ou após HARDRESET.
V 1.68	22/05/2005	Máquina de estados da recepção de senha resetada após entrar em modo programação.
V 1.69	23/05/2005	Não altera configuração da RS232 ao retornar de low-power.
V 1.70	01/06/2005	Implementado bloco RS232/uDX100 para DXNET+ (com byte de assinatura).
V 1.71	20/06/2005	Correção na execução da PresetRAM que poderia causar falhas graves. Tamanho do prog. limitado em 20.000 bytes (na rotina de teste do CRC).
V 1.72	21/06/2005	Recupera controle da RS232 caso envie comando p/ dxnet e o proprio endereço não esteja em tabscan. Timeout de comandos via RS232 extendido para 2 segundos.
V 1.73	03/08/2005	Correção do bloco de relógio.
V 1.74	01/11/2005	Continuação da implementação do bloco MODBUS-RTU. Implementado comando 01.
V 1.75	21/11/2005	Modbus 01,03 ok.
V 1.76	23/11/2005	Modbus 05,06,17 ok. Incluídos testes de validade das funções MODBUS para limites importantes como endereços de leitura e escrita de variáveis e nodos. No acesso das entradas e saídas (comandos 01 ou 05) pode-se usar a base 30000 para saídas e 32000 para entradas.
V 1.77	24/01/2006	Bloco RX232 Start+Nbytes corrigido para ligar saída OK após receber StartString e quando N=0.
V 1.78	28/03/2006	Aceita nodos OUT na escrita via MODBUS, comando 05. Nodos OUT a partir de 40000 com SAIDAS a partir de 60000 e ENTRADAS a partir de 62000.
V 1.79	28/03/2006	Base do firmware modificada.

V 1.80	29/03/2006	Início dos testes com bloco aritmética ponto flutuante.
V 1.81	08/04/2006	Continuação dos testes com blocos de PF.
V 1.82	11/04/2006	Intervenção para evitar que serial assuma novo valor antes de receber um RUN.
V 1.82	29/04/2006	Blocos PF Lint->Flt e Flt->Lint testados.
V 1.83	02/05/2006	Blocos de SOMA e SUBTRACAO em ponto flutuante. Inclui indicação de erro.
V 1.84	06/05/2006	Bloco de subtração corrigido. Nodo de erro na conversão ainda não corrigido.
V 1.85	06/05/2006	Bloco de mult e div em ponto flutuante estão implementados.
V 1.86	09/05/2006	Modificado para assumir conf. serial do Header quando receber comando RUN.
V 1.87	08/06/2006	Inclusão de funções PF: Ln, e^x, x^y, SQRT, CBRT.
V 1.88	20/06/2006	Corrigida a recuperação de UART0 ao retornar de low-power.
V 1.89	18/08/2006	Testa RUNKEY para saber se pode entrar em RUN após hard-reset.
V 1.90	22/08/2006	Copia leitura dos A/Ds para VarOut no fim do processo de ajuste da leitura A/D para garantir que leitura destas variáveis via DXNET retorne valor correto em vez de zero.
V 1.91	10/09/2006	Reseta temporizador de BACKTIME ao voltar de hard reset e caso entre em RUN.
V 1.92	10/09/2006	Novos blocos para Longint (AND,OR,XOR,SWAP,MONTA). Acionamento de nodos entrada_dx e entrada_out com numeração 32000.
V 1.93	14/09/2006	Leitura de MMC via RS232, re-inserção de MMC, blocos de comparação FLOAT.
V 1.94	31/10/2006	Alteração no processo de inicialização para tratar caso hard-reset em bateria.
V 1.95	11/11/2006	Blocos NODO->VAR e VAR->NODO.
V 1.96	24/01/2007	Incluídos testes de quant. de dados a serem lidos pela serial ou DXNET (máx. 29 words).
V 1.97	17/05/2007	Corrigido limite de escrita na RAM.
V 1.98	04/08/2007	Novas rotinas de MMC, tornando o µDX200 compatível com cartões de 64M a 2G.
V 1.99	05/08/2007	Melhorias nas rotinas de inicialização e append do MMC.
V 2.00	05/08/2007	Alterada base de tempo do I2C para 125µs (I2C com o dobro da velocidade).
V 2.01	06/08/2007	Incluída função DIMMER.
V 2.02	07/08/2007	Correção na função FLUSH do MMC, permitindo fazer FLUSH sem retirar o cartão MMC.
V 2.03	24/12/2007	Incluídas funções trigonométricas.
V 2.04	10/01/2008	Incluída função bit set/reset usando nodo CI para indicar SET ou RESET.
V 2.05	10/01/2008	Incluída a opção de escrita MMC com formato de tamanho fixo para INT, WORD, LINT e STRING.
V 2.06	19/09/2008	Correção no teste de bit para sensor de zero do dimmer via E9 ou E10. Dia da semana foi corrigido para ser incrementado também em meses com 30 dias. Corrigido congelamento de entradas ligadas dos µDX210s quando pára programa aplicativo.
V 2.07	14/01/2009	Várias correções para evitar reset no caso de retorno de low-power com programas aplicativos extensos.
V 2.08	20/01/2009	Aumento nos timeouts da rede DXNET (4,5ms e 3,5ms; em vez de 2,25ms e 1,75ms) (valores com folga de 0,25ms). Filtro em 8KHz para entradas de contagem rápida E9 e E10.
V 2.09	23/01/2009	Diversas otimizações no cálculo de CRC e INTs do µDX200.
V 2.10	19/06/2009	Correção no reset de bloco temporizador Atraso.

V 2.11	05/08/2009	Correção no cálculo de saídas analógicas (se resultado negativo força em zero)
V 2.12	17/08/2009	Implementada correção nas saídas analógicas, de forma a permitir melhora significativa na estabilidade térmica das mesmas.
V 2.13	25/08/2009	Correção na leitura de blocos de RAM e limites de validação dos comandos seriais.
V 2.14	20/11/2009	Correção no cálculo de número de dígitos de variável longint (usado para transmissão de longint via RS232).
V 2.15	14/12/2009	Correção no reset via rede DXNET após envio de programa aplicativo.
V 2.16	22/06/2010	Correção no controle de RTS caso esteja selecionada opção de espera de CTS.
V 2.17	09/08/2010	Previsão para baud-rate de 110 bps.
V 2.18	05/10/2010	Correção nos blocos TX-RS232 no que tange ao tamanho da transmissão.
V 2.19	21/12/2010	Inicialização de tempo para entrada em mono Normal ao resetar CLP.
V 2.20	21/02/2011	Correção no comando de modo normal quando sem bateria.
V 2.21	19/03/2013	Tentativa de correção na inicialização de RS232, evitando perda de dados a transmitir. De fato essa versão, além de não corrigir esse eventual problema, gerou problemas adicionais. Deve ser substituída por versão 2.22.
V 2.22	18/05/2013	Correção na inicialização de RS232, evitando perda de dados a transmitir.
V 2.23	28/08/2013	Precauções para evitar reset ao sair de low-power com comunicação serial RS232. Correção no zeramento de flags para comunicação serial. Correção no contador de tempo (60 segundos) para entrar em modo normal.
V 2.24	27/09/2018	Implementado cálculo e transmissão via serial de CRC CCITT.

Biblioteca de blocos para µDX200

A biblioteca de blocos contém todos os blocos de programação para o controlador µDX200, acessíveis via Editor PG. Abaixo as versões existentes até a data de confecção deste manual.

V 1.0	21/12/2004	Versão inicial com 178 blocos.
V 1.1	28/01/2005	243 blocos.
V 1.2	02/06/2005	254 blocos.
V 1.3	27/07/2005	256 blocos.
V 1.4	14/03/2006	257 blocos.
V 1.5	17/05/2006	286 blocos.
V 1.6	01/09/2006	304 blocos (blocos de ponto flutuante).
V 1.7	15/09/2006	304 blocos (pequena correção na versão 1.6).
V 1.8	25/10/2006	308 blocos (conversão Var-Nodo e Nodo-Var).
V 1.9	04/01/2007	308 blocos (pequena correção na versão 1.8).
V 2.0	01/07/2007	309 blocos (bloco de saída a relé via I2C).
V 2.1	19/11/2007	311 blocos (blocos de SLA Var e SRA Var Inteiro).
V 2.2	24/12/2007	323 blocos (blocos trigonométricos).
V 2.3	10/01/2008	333 blocos (BIT, SET, RESET, MMC tamanho fixo, DXNET Real).
V 2.4	08/07/2008	335 blocos (leitura e escrita em Tabela Real).
V 2.5	20/01/2009	336 blocos (leitura I^2C em 16 bits).
V 2.6	05/06/2009	336 blocos (correção em SQRT LongInt e Chave Inversora)
V 2.7	24/08/2009	340 blocos (quatro blocos adicionais para Dimmer via rede I2C).
V 2.8	20/04/2010	341 blocos (bloco de transmissão infravermelha via rede I2C).
V 2.9	03/08/2011	341 blocos (indicação de variável para diferenciar blocos de atribuição).
V 3.0	17/03/2015	342 blocos (bloco de µDX215 - 4 I/Os via rede I2C).

V 3.1	27/09/2018	345 blocos (transmissão RS232 e teste de CRC CCITT).

Editor e Compilador PG para µDX200

Ver lista de versões do PG em Versões de Software do Controlador µDX201.

Parte II

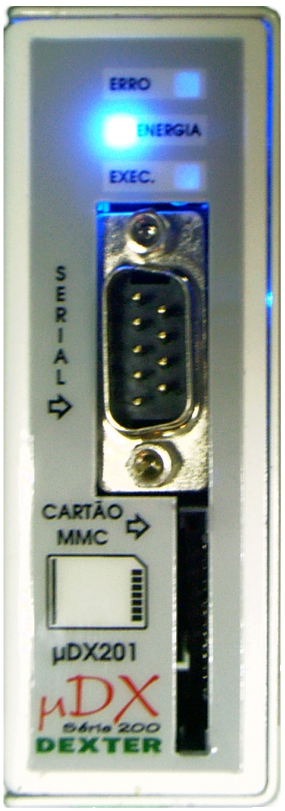
Controlador µDX201

O controlador µDX201 é idêntico, externamente, ao controlador µDX200, exceto pelo led de energia, que no caso do µDX201 é azul e não verde. Internamente também as diferenças são mínimas. O que os diferencia é a capacidade de processamento e memória, e também o software interno (firmware). Além disso, o controlador µDX201 permite conectar uma Interface Homem/ Máquina (IHM) via conector de expansão. Assim, todas as instruções para µDX200 referentes a seleção de jumpers, troca de bateria e reset físico são válidas para o µDX201.

As melhorias do µDX201 em relação ao µDX200, até o momento, são as seguintes:

- Dobro de blocos de instruções (cerca de 2000 blocos).
- Quádruplo de variáveis (cerca de 1200 variáveis 16 bits).
- Dobro de velocidade do processamento.
- Blocos para acionamento de IHM via conector de expansão.
- Blocos para cálculo de CRC utilizado em protocolo DNP-3.
- Quádruplo de área de buffer para comunicação serial (256 bytes).
- Quádruplo de área de buffer para tarefas de comunicação via rede DXNET.
- Diversos blocos adicionais, como MUX, DEMUX, FF tipo T, Limite, etc.

O µDX201 deve ser empregado sempre que for necessário utilizar a Interface Homem/Máquina para Série µDX200. Esta IHM possui display gráfico e tela sensível ao toque (touchscreen), permitindo grande flexibilidade na visualização e entrada de dados. A partir da versão 2.2.0.0 do software PG já é prevista a programação de controladores µDX201, assim como a utilização de IHM. Outro caso em que o µDX201 torna-se interessante é em programas aplicativos muito extensos, em que os recursos (blocos, variáveis, nodos) do controlador são excedidos.



Especificações Técnicas

Características Gerais

- 551 instruções, incluindo lógica aritmética inteira de 16 e 32 bits.
- Aritmética em ponto flutuante.
- Mais de 3000 blocos de programação.
- Mais de 1200 variáveis de 16 bits.
- Mais de 2000 nodos.
- Execução do programa em modo de paralelismo lógico.
- Ciclo de execução do programa aplicativo abaixo de 0,5ms.
- "Watch-Dog Timer" incorporado.
- 8 entradas analógicas, que podem ser usadas como entradas digitais.
- 6 saídas analógicas, que podem ser usadas como saídas digitais.
- 2 entradas de contagem rápida (até 8KHz).
- Até 512 I/Os via módulos de Expansão de Entradas/Saídas (µDX210).
- Interface Homem/Máquina (IHM) via módulo opcional.
- Relógio e calendário de tempo real (com previsão de ano bissexto).
- Dimensões reduzidas: 115 x 86 x 30 mm.
- Protegido contra transientes elétricos.
- Acondicionado em gabinete metálico, muito resistente.
- Bateria interna: pilha CR2032 com durabilidade de 5 anos.
- Temperatura de operação: 0°C até 55°C.
- Rede DXNET para 1500 metros.
- Rede I^2C para 1000 metros.
- Slot para cartão MMC, MMC Plus, SD ou microSD (registro de eventos). Cartões de 32MB a 2GB.

  Atenção: o controlador µDX201 não suporta cartões SDHC (high capacity - acima de 2GB)
- Comunicação serial de 110 a 115200bps (porta RS232C completa).

  Tempos de timeout default do µDX201 dependem do baud rate utilizado:
  (cerca de 5,5x o tempo de um caracter)

110 bps	450 ms
300 bps	165 ms
600 bps	87 ms
1200 bps	41 ms
2400 bps	22 ms
4800 bps	11 ms
9600 bps	5,5ms
19200 bps	2,8ms
38400 bps	1,7ms
57600 bps	1,2ms
115200 bps	0,8ms

- Duas portas seriais RS232C adicionais por software (110 a 9600bps).

  Ampla comunicação com outros dispositivos devido a grande quantidade de portas seriais:
  03 portas seriais RS232C; uma principal (115200bps) e duas auxiliares (9600bps).
  01 porta I^2C, capaz de comunicar-se com inúmeros periféricos.
  01 porta RS485 (rede DXNET), para comunicação entre CLPs.

Entradas Analógicas (E1 a E8)

Escala de 0-2,5V Resolução = 610,5µV (12 bits)
Resistência de entrada = 400KΩ
Precisão melhor que 0,15% do fundo de escala
Máxima Tensão de entrada = 30V

Escala de 0-10V Resolução = 2,442mV (12 bits)
Resistência de entrada = 10KΩ
Precisão melhor que 0,15% do fundo de escala
Máxima Tensão de entrada = 30V

Escala de 0-20mA Resolução = 4,884µA (12 bits)
Resistência de entrada = 125Ω
Precisão melhor que 0,15% do fundo de escala
Máxima Corrente de entrada = 30mA

*** Entrada E1 permite leitura em alta resolução (16 bits). Também é possível concatenar de 2 a 8 entradas analógicas em alta resolução (16 bits).**

Saídas Analógicas (S1 a S6)

Escala de 0-10V Resolução = 2,442mV (12 bits)
Corrente máxima de saída = 10mA
Precisão melhor que 0,3% do fundo de escala

Escala de 0-20mA Resolução = 4,884µA (12 bits)
Resistência de carga ≤ 500Ω
Precisão melhor que 0,3% do fundo de escala

Referência de Tensão (+10V REF)

Tensão nominal = 10V ± 5%
Estabilidade térmica típica = 100ppm/°C
Corrente de saída máxima = 10mA

Entradas Digitais Rápidas (E9 e E10)

Freqüência Máxima = 8KHz
Resistência de Entrada = 10KΩ
Mínima Tensão de entrada = 3V
Máxima Tensão de entrada = 30V

Alimentação Elétrica (+V e N)

Tensão de operação = 11,0 a 26,4Vdc *****
Corrente Típica (sem Expansões) = 150mA
Corrente Máxima (com 32 Expansões µDX210, em 24Vdc) = 4A

***** Equipamento projetado para operar em 24Vdc ±10%. Para funcionamento em 12V existem as seguintes restrições:

- É necessário empregar Expansões µDX210-12 (relés para 12Vdc).
- Limitação de 16 Expansões µDX210-12 (em vez de 32 µDX210).
- Saídas analógicas perdem sua função (não permitem saída em 0-10V ou 0-20mA) mas podem ser utilizadas como saídas digitais ou PWM.
- Saída de referência +10V REF perde estabilidade, ficando sujeita as flutuações da alimentação do µDX201. Também seu valor fica abaixo dos 10V nominais.

Atenção: *As entradas do Controlador Programável µDX201 não possuem isolação galvânica e, portanto, não podem ser ligadas em fontes de sinal com referências distintas, e muito menos a rede elétrica (127 ou 220 Vac).*

Versões de Software

Firmware do µDX201

Trata-se do software interno do controlador µDX201. Note que este software pode ser atualizado, bastando para isso remeter a unidade para a Dexter. A única despesa será com o transporte, pois a atualização é gratuita (até o limite de duas atualizações por controlador). Além de correções, o µDX201 é constantemente acrescido de novas facilidades e blocos de programação.

A partir da versão 3.61 o firmware do µDX201 pode ser atualizado via programa PG. Basta baixar a última versão disponível do PG no site da Dexter -www.dexter.ind.br, desinstalar a versão antiga do PG e instalar a nova. A seguir, estabeleça comunicação com o controlador via janela do Compilador no PG e selecione o menu pop-down **µDX** → **Atualizar Firmware para o µDX**. O PG mais atual sempre inclui a última versão de firmware para o µDX201.

V 3.18	13/10/2009	Versão inicial.
V 3.19	12/11/2009	Cálculo de CRC para DNP-3. Previsão de escrita tipo MOV, OR, XOR, AND no display da IHM. Blocos de IHM sem restrição no uso de variáveis nos parâmetros.
V 3.20	01/12/2009	Previsão para controle de contraste no display da IHM. Correção no uso de IHM com múltiplas expansões µDX210. Correção no reset via rede DXNET após envio de programa aplicativo.
V 3.21	22/06/2010	Correção no controle de RTS caso esteja selecionada a opção de espera de CTS.
V 3.22	13/08/2010	Previsão para baud-rate de 110 bps.
V 3.23	05/10/2010	Correção nos blocos TX-RS232 no que tange ao tamanho da transmissão.
V 3.24	26/10/2010	Previsão para tamanho=0 em impressão de string na IHM.
V 3.25	05/11/2010	Corrigido limite para escrita em tabela, permitindo acesso aos buffers de RS232.
V 3.26	21/12/2010	Inicialização de UART0 em blocos RS232 RX se não estiver transmitindo e vice-versa. Inicialização de tempo para entrada em mono Normal ao resetar CLP.
V 3.27	21/02/2011	Correção no comando de modo normal quando sem bateria.
V 3.28	22/02/2011	Status de run/stop salvo em FLASH, mantendo-o mesmo sem bateria.
V 3.29	11/07/2011	Correção no sinal de ATIVO de Expansões µDX210 com programas extensos.
V 3.30	19/03/2013	Tentativa de correção na inicialização de RS232, evitando perda de dados a transmitir. De fato essa versão, além de não corrigir esse eventual problema, gerou problemas adicionais. Deve ser substituída por versão 3.31.
V 3.31	18/05/2013	Correção na inicialização de RS232, evitando perda de dados a transmitir.
V 3.33	28/08/2013	Precauções para evitar reset ao sair de low-power com comunicação serial RS232. Correção no zeramento de flags para comunicação serial. Correção no contador de tempo (60 segundos) para entrar em modo normal.
V 3.34	07/09/2013	Verificaçao de falta de alimentação elétrica (24V) em até 5ms.
V 3.35	24/09/2013	Leitura de entrada analógica E1 em alta resolução (16 bits). Leitura de freqüência nas entradas digitais E9 e E10.

V 3.37	12/11/2013	Previstos novos blocos: - Blocos de comparação com teste invertido. - Flip-flop tipo T. - Blocos Limite Inteiro, LongInt, Word e Real. - Blocos MUX e DEMUX Inteiro e Word (8 e 16 bits). - Blocos Nodos->Var e Var->Nodos em 8 bits.
V 3.38	16/08/2014	Inseridos blocos DXNET para leitura e escrita com múltiplas variáveis.
V 3.39	24/11/2014	Corrigido erro introduzido na versão 3.37 em funções trigonométricas.
V 3.40	30/01/2015	Compatibilização com cartões de memória MMC Plus, SD e microSD. Atenção: não há compatibilidade com cartões SDHC (high capacity).
V 3.41	03/03/2015	Leitura de período nas entradas digitais E9 e E10 (resolução de 125µs).
V 3.42	25/03/2015	Corrigido bug na energização do µDX201. Caso entradas E9 e E10 estivessem ligadas ao energizar µDX201 ele não "acordava".
V 3.43	11/07/2016	Retirado bug de escrita em cartão MMC/SD. Ao completar 512 bytes do buffer de escrita os blocos MMC não aguardavam o final da gravação para iniciar nova gravação no buffer.
V 3.44	25/03/2017	Previsto cálculo de CRC-CCITT. Comutação automática de tipo de display usado na IHM.
V 3.45	02/05/2017	Correção na inicialização do novo display para IHM.
V 3.46	12/06/2017	Uso de CTS/RTS/DTR/DSR como portas seriais por software (de 110bps até 9600bps).
V 3.47	19/07/2017	Prevista programação de número de entradas analógicas lidas para conversão analógica/digital de alta resolução (palavra reservada _E1).
V 3.48	17/08/2017	Rede I2C tradicional aprimorada, com verificações mais consistentes.
V 3.49	21/01/2018	Maiores checagens nas comunicações via DXNET e RS232, verificando se o número de bytes da mensagem é coerente com o número de variáveis a ser lida ou escrita.
V 3.50	27/02/2018	Colocação de semáforo na rede DXNET, de forma a evitar problema de algum µDX201 decodificar a resposta de outro µDX201 como um comando a ser executado.
V 3.51	29/06/2018	Pequenas alterações transparentes ao usuário.
V 3.52	26/07/2018	ModBus RTU: inserido comando 02 similar ao comando 01 (ler nodos), e 04 similar ao comando 03 (ler variáveis). Suporte a blocos de Pointer (retornam endereço da variável). Suporte a blocos IHM X (parametrizáveis via variáveis).
V 3.53	14/08/2018	Blocos IHM X com uso de operandos do tipo pointers.
V 3.54	21/08/2018	Blocos I2C com transmissão de CRC-8 para checagem do dado.
V 3.55	10/09/2018	Antecipação de desligamento de dimmer de potência para evitar instabilidade com cargas capacitivas.
V 3.56	17/10/2018	Leitura de módulo µDX218 (leitura de múltiplos sensores) via conector de expansão.
V 3.57	24/10/2018	Bloco de CRC-16 para cartão MMC/SD.
V 3.58	11/11/2018	Previsão para gravar 4 variáveis Int ou 2 variáveis LInt no cartão SD.
V 3.59	13/02/2019	Correções nas rotinas de tratamento de cartão SD/MMC.
V 3.60	27/02/2019	Retirado problema catastrófico na recepção de dados via seriais por software (RS232-2 e RS232-3).
V 3.61	22/05/2019	Blocos Word para tratar múltiplas variáveis simultaneamente (8 words em cada bloco). Suporte para atualização de firmware do controlador µDX201 via PG.
V 3.62	12/08/2019	Blocos para inserção de dados e cálculo de somatório de tabelas.

V 3.63	26/09/2019	Mais melhorias na rotinas de tratamento de cartão SD/MMC.
V 3.64	20/01/2020	Inserido delay adicional na comutação de canais do µDX218, evitando influência entre os canais.
V 3.65	27/01/2020	Permite uso de múltiplos arquivos de dados no cartão MMC/SD. Padronização para leitura apenas em blocos de 512 bytes. Correção na rotina de APPEND prevendo leituras múltiplas de 512.
V 3.66	20/05/2020	Previsão para formatação FAT16X, usada em Windows 10.
V 3.67	25/05/2020	Correção em _RXQNT2 e _RXQNT3, tornando esses dados operacionais.
V 3.68	20/06/2020	Varredura de posições do diretório raiz e subdiretório UDX200 no cartão MMC/SD, permitindo múltiplos arquivos e subdiretórios no cartão. Mas arquivos de dados precisam ser em ordem crescente, adjacentes e contínuos (sem fragmentação).
V 3.69	27/08/2020	Permite editar setup de endereço de rede DXNET e parâmetros de porta serial RS232 (baud-rate, paridade, stop bits) via programa aplicativo. Suporta leitura pelo programa aplicativo da versão de firmware e do número de série do controlador µDX201.
V 3.70	07/10/2020	Correção na configuração de porta serial RS232 via programa aplicativo. Ao transmitir novo programa o µDX201 assume os dados da Configuração de Hardware associada ao programa, sem precisar de um hard reset.
V 3.71	16/05/2021	Correção no retorno de energia para evitar corrupção de dados gravados no cartão MMC/SD.
V 3.72	26/06/2021	Permite programar a profundidade de sobreamostras (oversampling) das entradas analógicas. Conforme o valor programado pode-se fazer oversampling em mais de uma entrada analógica.
V 3.73	09/09/2021	Implementado loop de execução de rotina em cartão MMC/SD, evitando falha no reconhecimento de cartão no caso de programas aplicativos extensos (TCiclo>5ms). Melhorias nas rotinas de cartão MMC/SD (inclusão de FFh antes e depois de comutação de CS).
V 3.74	27/10/2021	Inserido timeout em várias rotinas do cartão MMC/SD, evitando loops infinitos. Todas inicializações de watchdog timer em 4 segundos.
V 3.75	04/01/2022	Correção em temporizados de 1ms e 10ms de forma a evitar erros cumulativos na temporização. Agora o manual do uDX201 está correto ao afirmar existir apenas erro de jitter + erro devido ao tempo de execução do aplicativo. Limites de tempo de execução do programa aplicativo: Escala de 1ms = 64ms Escala de 10ms = 80ms Escala de 100ms = 100ms Escala de 1s = 1s
V 3.76	07/01/2022	Variáveis de oversampling para entradas E2 a E8 invadiam área de pilha, causando falha catastrófica. Usei area de buffer da RS232-3 para essas variáveis. Obviamente no caso de uso de oversampling com profundidade menor que 256 (o que habilita o oversampling de outras entradas) não é possível usar a porta serial RS232-3.
V 3.77	26/01/2022	Inibido reset no caso de detecção de falha de osciladores internos. Refresh de display e touchscreen da IHM a cada 10ms. Inserido nodo de sistema n25 para indicar falha nos osciladores.
V 3.78	23/02/2022	Corrigido algoritmo de detecção de múltiplos arquivos no cartão MMC/SD.

V 3.79	29/05/2022	Backup da RAM em FLASH via nodos de sistema: n26 = efetua backup de toda RAM na FLASH (salva RAM). n27 = leitura de backup da RAM em FLASH (recupera RAM). n28 = indica erro no backup de RAM em FLASH.
V 3.80	05/06/2022	Versão especial para Classificadora de Frutas.
V 3.81	21/08/2023	Correção em operação XOR INT e LINT (XOR de dois negativos acionava o nodo de erro aritmético). Retirada de comutação de RX RS232 para I/O normal. Correção nos blocos RS232 TX Int Mín e TX LInt Mín (número de caracteres). Inicialização de RS232-2 e RS232-3 no caso de reset ou programa parado.
V 3.82	19/09/2023	Correção de inicialização de baud-rate no caso de uso de blocos RS232-RX DXNET, DXNET+ e MODBUS. Além disso, incluí teste de validade da inicialização de RS232, permitindo inibir esse parâmetros nesses blocos (usar 0000h ou 0FFFFh).
V 3.83	04/09/2024	Geração de pulsos nas saídas analógicas S1,S3 e S5, com aceleração e desaceleração. Saídas S2,S4 e S6 indicam sentido de rotação de S1,S3 e S5, respectivamente. Versão abandonada porque a aceleração e desaceleração são muito toscas.
V 3.84	22/09/2024	Modo adicional para as saídas analógicas: FREQ (além dos já existentes V,mA,PWM). Esse modo gera uma freqüência na saída analógica de 977Hz a 10KHz, conforme o valor da variável correspondente à saída analógica (v8 a v13). Note que o valor mínimo da variável é 400, correspondendo a 10KHz, e o máximo é 4095, correspondendo a 977Hz: **FREQ = 4.000.000/Variável**. Valores abaixo de 400 força saída desligada, e valores acima de 4095 força saída ligada. Caso seja especificadas saídas analógicas em 10 bits os limites comutam para 400 (10KHz) a 1023 (3,9KHz).
V 3.85	02/10/2024	Resolução de 14 bits nas saídas analógicas. Com isso o modo FREQ pode ser estendido de 400 (10KHz) até 16383 (244Hz).
V 3.86	16/10/2024	Geração de pulsos nas saídas analógicas S1,S3 e S5, com aceleração e desaceleração. Saídas S2,S4 e S6 indicam sentido de rotação de S1,S3 e S5, respectivamente. Controle de aceleração e desaceleração precisos. Função de linearização da freqüência nas saídas analógicas. Possibilidade de saídas analógicas em 14 bits, além de 12 e 10 bits. Novo modo de operação das saídas analógicas (FREQ). Blocos para detecção de borda de subida, descida, ou ambas. Palavras reservadas para controle de motores.
V 3.87	24/10/2024	Recurso de programar profundidade e número de canais analógicos com oversampling, inserido a partir da versão 3.72, havia sido inibido para oversampling abaixo de 256.

Biblioteca de blocos para µDX201

A biblioteca de blocos contém todos os blocos de programação para o controlador µDX201, acessíveis via Editor PG. Abaixo as versões existentes até a data de confecção deste manual.

V 1.2	24/10/2009	Versão inicial com 359 blocos.
V 1.3	20/04/2010	360 blocos (bloco de transmissão infravermelha via rede I2C).
V 1.4	03/08/2011	360 blocos (indicação de variável para diferenciar blocos de atribuição).
V 1.6	12/11/2013	381 blocos (inseridos blocos MUX e DEMUX, FF tipo T, etc).
V 1.7	16/08/2014	409 blocos (inseridos blocos DXNET com múltiplas variáveis)
V 1.8	17/03/2015	410 blocos (bloco de µDX215 - 4 I/Os via rede I2C)
V 1.9	30/03/2017	413 blocos (blocos para CRC- CCITT)
V 2.0	12/06/2017	481 blocos (blocos para RS232-2 e RS232-3)
V 2.1	08/08/2018	497 blocos (blocos de Pointer e IHM X)
V 2.2	21/08/2018	505 blocos (blocos de rede I2C com transmissão de CRC-8)
V 2.3	17/10/2018	507 blocos (blocos para inicialização e leitura de µDX218)
V 2.4	11/11/2018	514 blocos (escrita de múltiplas variáveis e CRC em cartão SD).
V 2.5	22/05/2019	522 blocos (blocos Word para múltiplas variáveis).
V 2.6	27/06/2019	530 blocos (CRC CCITT nas portas seriais auxiliares RS232-2 e RS232-3)
V 2.7	12/08/2019	538 blocos (inserção de dados e cálculo de somatório em tabelas)
V 2.8	27/08/2020	540 blocos (blocos para setup de DXNET e RS232).
V 2.9	20/08/2023	540 blocos (correção nos índices para páginas do manual).
V 3.0	24/08/2023	540 blocos (ordem de parâmetros corrigida em RS232-2 e RS232-3).
V 3.1	04/09/2024	551 blocos (blocos para controle de motores e detecção de borda).

Editor PG para µDX200

O programa Editor PG permite gerar os arquivos fonte para os programas aplicativos do controlador µDX200.

1.0.0.0	Jul/2004	Primeira versão de publicação.
1.0.0.1	Ago/2004	Mudanças estéticas no selo de página. Proporção de pixels/mm - tamanho das páginas de edição - alterado de 106 dpi para 200 dpi. Para certas larguras da janela da biblioteca de componentes, ao iniciar o programa, uma coluna de componentes pode ficar escondida para além da borda direita. Moldura de seleção dos blocos não exibe corretamente quando a pele inclui retângulos e/ou bitmaps cobrindo toda a extensão do componente.
1.0.1.0	Set/2004	Diversas alterações na estrutura dos blocos. Alterações na forma de exibição dos blocos "como sombra" - quando em movimento - de forma a suavizar os efeitos de cor nos diversos elementos das peles. Janela de propriedades de blocos exibe página de variáveis e constantes mesmo quando o bloco possui apenas parâmetros locais. Modifica as janelas de propriedades de data, hora e dia da semana para permitir a determinação de valores "XX" (qualquer). Inclui imagem de abertura.
1.0.2.0	Set/2004	Alterações na estrutura dos blocos. Janela de propriedades de blocos não trata corretamente as datas e horas quando são definidas como referência. Incluído alerta de sistema operacional com linguagem diferente do português.

1.0.3.0	Out/2004	Implementada a versão Alfa 1 do pré-compilador.
1.0.3.1	Out/2004	Implementa compilacão de projetos (múltiplas páginas). As páginas do projeto podem estar no disco ou no editor e não precisam ser gravadas antes de compilar, caso modificadas. Implementa associação de help contextual para a lista de erros e avisos de compilação. Inclui item "Ajuda..." no menu Ajuda, para abertura do arquivo help na página de conteúdo. Conduites não conectados nos dois extremos são exibidos na cor cinza, quando não selecionados. Essa condição não significa que o conduite está ligado a um nodo específico, apenas que não tem as duas extremidades conectadas a algum bloco.
1.0.3.2	Out/2004	Caixa de texto não aciona janela de propriedades utilizando o botão direito do mouse. Editor de propriedades não permite utilização de números negativos. Editor de propriedades de dia-da-semana e hora atribui valor "qualquer", indevidamente, quando utilizada uma referência. Inclui botões rápidos para compilação. Modifica formato do aviso de nodos com tipos de dado conflitantes. Corrige exibição e posicionamento das janelas flutuantes durante a inicialização.
1.0.3.3	Out/2004	Implementa especialidades para seleção de módulo e entrada/saída.
1.0.3.4	Nov/2004	Em certas circunstâncias as linhas virtuais na criação de um conduite aparecem sólidas em vez de pontilhadas. Alguns blocos que se utilizam de linhas com espessura maior que um podem apresentar "pixels de arrasto" ao serem movidos.
1.0.4.0	Jan/2005	Pequenas correções ortográficas. Modifica lista de tipos para tabelas e listas, excluindo lógico, real e indefinido e incluindo longint. Reformata página de propriedades de listas, retirando o campo para definição do número de colunas. Reformatação da grade de edição de variáveis na janela de propriedades (a fim de usar todo o espaço disponível para a caixa de edição). Lista de variáveis, na janela de propriedades, não atualiza valor digitado em campo de variável quando um novo campo é selecionado usando apenas o click do mouse sobre outro item. Permite atribuir valores em notação hexa até o máximo suportado pelos bits da variável inteira (FF para BYTE; FFFF para INTEGER e FFFFFFFF para LONGINT). Permite atribuir valores "char" para variáveis numéricas (byte, word, int e longint) no formato 'A'. Implementado recurso de auto-abertura do programa de compilação externa ("Compilador"). Uma nova opção de configuração foi incluída para permitir ou impedir que o PG-Editor abra o compilador a cada processo de Compilação executado com sucesso. Incluído botão de acionamento rápido, opção de menu e "short-cut" (F5) para acionamento do compilador externo. Implementado recurso de auto-verificação de integridade do executável.
1.0.4.0a	Jan/2005	Janela de edição de propriedades não permite definir um longint com valor 0FFFFFFFFh.

1.0.5.0	Ago/2006	Correção das janelas de propriedades de página, que informavam de forma invertida a largura pela altura. A alteração pode acarretar carregamentos incorretos de páginas de versões antigas. Passa a não permitir a movimentação de blocos para fora da área editável das páginas. Avisa quando a página é reduzida e ficam elementos fora da área visível. Avisa mesmo que os blocos estejam em coordenadas negativas. Mas, nesse caso, o usuário não poderá alterar o formato de maneira que os blocos voltem para a área de edição. Será preciso copiar os blocos visíveis para outra página e refazer a parte do código de desapareceu da zona de edição. Ao executar cópia, mantém selecionado(s) o(s) bloco(s) de destino, permitindo cópias múltiplas. Permite a cópia múltipla entre páginas de fonte e usa a seleção do destino para que o processo seja natural ao usuário (vê qual a origem da cópia ao executá-la). Ao mover dois ou mais blocos ligados a um terceiro bloco (por exemplo) as ligações dos conduites podem ser desfeitas visualmente, mas continuam ativas para efeito de compilação. Essa versão corrige esse problema, incluindo tratamento completo de regeneração, reposicionamento e reconstrução automática das linhas. Atribui margem de um snap no contorno da página. Dessa forma, não permite a movimentação de blocos e segmentos de forma que encostem nos limites da página.
1.0.5.1	Out/2006	Manual incorporado ao help do software Editor PG. Help dos blocos sensível ao contexto.
1.0.5.2	Jan/2007	Inserida tecla de Ajuda na tela de edição dos blocos. Tecla [F1] do computador também aciona o help.
2.1.0.1	Jul/2008	Novo Compilador integrado ao software de Edição, com várias melhorias na visualização de nodos e variáveis. Monitoramento gráfico de variáveis, inclusive com leitura de arquivos. Macro-células (possibilidade de compactar todo programa aplicativo em um bloco de programação único, utilizável em outros programas). Melhorias na leitura e formatação de cartão MMC. Caixa de texto com maior número de caracteres, e alinhamentos de texto à esquerda e à direita. Janela com mensagens de compilação, e também listagem de variáveis e nodos usados no programa aplicativo. Importação e exportação de tabelas, e inserção de elementos em uma tabela.
2.1.0.5	Set/2008	Novo Compilador integrado ao software de Edição, com várias melhorias na visualização de nodos e variáveis. Monitoramento gráfico de variáveis, inclusive com leitura de arquivos. Macro-células (possibilidade de compactar todo programa aplicativo em um bloco de programação único, utilizável em outros programas). Melhorias na leitura e formatação de cartão MMC. Caixa de texto com maior número de caracteres, e alinhamentos de texto à esquerda e à direita. Janela com mensagens de compilação, e também listagem de variáveis e nodos usados no programa aplicativo. Importação e exportação de tabelas, e inserção de elementos em uma tabela.

2.1.0.6	Out/2008	Correção na geração de Macros, permitindo identificar constantes com formatos numéricos diferentes de números inteiros positivos. Ampliação da tabela de varredura TABSCAN da rede DXNET (de 32 endereços para 128 endereços).
2.1.2.0	Dez/2008	Melhorias na leitura de cartão MMC via comunicação serial com µDX200. Implementada leitura completa do cartão MMC via comunicação serial. Gerado layout de monitoração, permitindo escolher quais variáveis monitorar e fazer sort das mesmas. PG verifica se existe versão mais atualizada no site da Dexter: http://www.dexter.ind.br/pg1.htm Corrigida representação de números hexadecimais negativos. Edição de blocos com aba adicional com lista de conexões do bloco. Escrita em variáveis com parâmetros (equação de primeiro grau). Ordem dos pinos de Macro pode ser editada, ou ainda feito sort.
2.1.3.2	Jun/2009	Correção na exibição de parâmetro contendo caracter &. Inclusão de aviso de relevância de maiúsculas e minúsculas na senha.
2.1.3.6	Set/2009	Incluída opção "Produtos e acessórios..." no menu pop-down "Ajuda".
2.1.3.9	Set/2009	Ferramentas para calibração e compensação térmica do µDX200.
2.2.0.0	Nov/2009	Versão prevê comutação de dispositivo (µDX200 ou µDX201). Correção na comunicação serial via Ethernet. Representação no Compilador PG da IHM para µDX201.
2.2.0.1	Dez/2009	Leitura de tabelas em RAM.
2.2.0.4	Abr/2010	Tamanho máximo da página de programação aumentado para 900 x 900 milímetros.
2.2.0.7	Ago/2010	Previsão para baud-rate de 110 bps na Configuração de Hardware.
2.2.0.8	Mai/2011	Correção na alocação de tamanho automático para Macros.
2.2.0.10	Jun/2011	Falha de tratamento no início da monitoração automática, resultando em falhas consecutivas e ajuste para cálculo do timeout mínimo. Entretanto, esta versão possui problema na procura de µDXs e, portanto, não deve ser usada.
2.2.0.11	Jul/2011	Correção dos problemas de comunicação introduzidos na versão 2.2.0.10. Quantidade de Undo/Redo ampliado de 5 para 20. Diminuído tempo de apresentação da tela de abertura do PG.
2.2.0.13	Nov/2011	Compatibilização da área de IHM quando programa aplicativo utiliza cartão MMC.
2.2.0.15	Abr/2012	Compatibilização da leitura de versão do PG no novo site da Dexter. Aumento de constantes e blocos disponíveis no programa aplicativo, evitando erro 0112 no Compilador.

2.2.0.16	Jan/2013	Inserida informação de Tipo de Conexão nas conexões de Macro. Tipo de dispositivo (µDX200 ou µDX201) pode ser modificado em um projeto já elaborado. Inserida opção de "Salva Tudo" (salva projeto e todas as suas páginas). Inserida opção de "Cancela envio de programa", de forma a permitir cancelar o envio de programa aplicativo para controlador µDX200. PG permite "pular" para o bloco que originou o erro de compilação, bastando clicar sobre a descrição do erro nas mensagens existentes na janela do Compilador (aba Mensagens). Melhorias nos procedimentos de calibração do µDX200 e µDX201. Várias melhorias no monitoramento de IHM do µDX200 (µDX220), como visualização de nodos e variáveis analógicas da IHM, e possibilidade de inibir monitoração da IHM. Retirado bug ao pressionar Desfazer ou Refazer durante criação de conexão no editor PG. Inserida janela de estatísticas de comunicação.
2.2.0.17	Abr/2013	Falha introduzida na versão 2.2.0.16 impossibilita a leitura de páginas gravadas nesta versão. Problema corrigido na versão 2.2.0.17. A versão 2.2.0.16 não deve ser usada.
2.2.0.19	Jan/2014	A página em foco é marcada na lista de páginas do projeto. Corrigida advertência ao editar propriedades do projeto.
2.2.0.20	Jun/2014	Corrigido bug de rótulo interligando páginas de um projeto.
2.2.0.21	Mar/2015	Inseridas palavras reservadas _P9 e _P10, que retornam a leitura de período nas entradas E9 e E10 (em múltiplos de 125µs). Previsto módulo para rede I2C µDX215, com 4 entradas e 4 saídas digitais.
2.2.0.22	Jan/2017	Incluídas várias macros, como acionamento de tela de projeção e leitura de variáveis em controlador µDX101.
2.2.1.2	Jun/2017	Suporte à portas seriais auxiliares (RS232-2 e RS232-3).
2.2.1.3	Jun/2017	Corrigido bug de supressão de variável ao inserir variável absoluta.
2.2.2.0	Jul/2017	Prevista programação de número de entradas analógicas lidas para conversão analógica/digital de alta resolução (palavra reservada _E1).
2.2.2.1	Ago/2018	Inclusão de várias Macros para IHM. Blocos de IHM X (capazes de serem parametrizadas via variáveis). Blocos de Pointer (retorna endereço de variáveis).
2.2.2.2	Ago/2018	Blocos de I2C com transmissão de CRC-8.
2.2.2.3	Out/2018	Suporte à Expansão µDX218 (leitura de múltiplos sensores).
2.2.2.4	Jan/2019	Aumentado limite de blocos do µDX201 de 3500 para 4500 (condicionado ao tamanho dos blocos usados no programa aplicativo).
2.3.0.1	Mai/2019	Permite blocos Word para tratar múltiplas variáveis simultaneamente (8 words em cada bloco). Disponibilizada opção para atualização de firmware do controlador µDX201 no menu pop-down [µDX].
2.3.0.2	Mai/2019	Aumentado limite de variáveis monitoradas de 800 para 1600.
2.3.1.0	Set/2019	Blocos para inserção em tabela e somatório de tabela. Atualização de firmware do controlador µDX201 para versão 3.62.
2.3.1.1	Out/2019	Atualização de firmware do controlador µDX201 para versão 3.63.
2.3.2.0	Jan/2020	Permite uso de múltiplos arquivos de dados no cartão MMC/SD. Atualização de firmware do controlador µDX201 para versão 3.65.
2.3.2.1	Mai/2020	Atualização de firmware do controlador µDX201 para versão 3.66.
2.3.2.2	Mai/2020	Compilador v2.30, com indicação de erro por excesso de nodos. Atualização de firmware do controlador µDX201 para versão 3.67.
2.3.2.3	Jun/2020	Atualização de firmware do controlador µDX201 para versão 3.68.
2.3.2.4	Jul/2020	Correção no cálculo de nodos usados pelo programa aplicativo.

2.3.2.5	Ago/2020	Retorno ao cálculo original de nodos usados pelo programa aplicativo. O cálculo introduzido na versão 2.3.2.4 estava errado.
2.3.2.6	Ago/2020	Palavras reservadas _CFGDXNET e _CFGRS232 inseridas. Blocos para Setup DXNET e Setup RS232 suportados. Palavras reservadas _NUMSER e _FIRMWARE inseridas. Atualização de firmware do controlador µDX201 para versão 3.69.
2.3.2.7	Set/2020	Limite de bytes em tabelas RAM aumentado de 2048 para 4096.
2.3.2.8	Out/2020	Atualização de firmware do controlador µDX201 para versão 3.70.
2.3.2.9	Mai/2021	Atualização de firmware do controlador µDX201 para versão 3.71.
2.3.3.0	Jun/2021	Programa profundidade de sobreamostragem (oversampling) das entradas analógicas. Palavras reservadas _E2, _E3, _E4, _E5, _E6, _E7, _E8. Atualização de firmware do controlador µDX201 para versão 3.72.
2.3.3.1	Set/2021	Atualização de firmware do controlador µDX201 para versão 3.73.
2.3.4.0	Jan/2022	Palavras reservadas _E2, _E3, _E4, _E5, _E6, _E7, _E8 em outro endereço, de forma a não conflitar com pilha. Caso oversampling menor que 256 é inibida a porta serial RS232-3. Compilador v2.36, com novos endereços para _E2 a _E8.. Atualização de firmware do controlador µDX201 para versão 3.76.
2.3.4.1	Jan/2022	Atualização de firmware do controlador µDX201 para versão 3.77.
2.3.4.2	Fev/2022	Atualização de firmware do controlador µDX201 para versão 3.78.
2.3.4.3	Mai/2022	Atualização de firmware do controlador µDX201 para versão 3.79.
2.3.4.4	Ago/2023	Correção na compilação de blocos TX CRC DNP e TX CRC CCITT para porta serial RS232-3. Atualização de firmware do controlador µDX201 para versão 3.81.
2.3.4.5	Set/2023	Atualização de firmware do controlador µDX201 para versão 3.82.
2.3.5.0	Set/2024	Suporte para blocos antigos de controle de motores via saídas analógicas. Atualização de firmware do controlador µDX201 para versão 3.83. Versão abandonada porque a aceleração e desaceleração da versão 3.83 do µDX201 são muito toscas.
2.3.5.3	Out/2024	Suporte para novos blocos de controle de motores via saídas analógicas. Possibilidade de saídas analógicas em 14 bits, além de 12 e 10 bits. Novo modo de operação das saídas analógicas (FREQ). Bloco para linearização da freqüência nas saídas analógicas. Previsão de blocos para detecção de bordas. Atualização de firmware do controlador µDX201 para versão 3.85.
2.3.5.4	Out/2024	Palavras reservadas para controle de motores. Atualização de firmware do controlador µDX201 para versão 3.86.
2.3.5.5	Out/2024	Atualização de firmware do controlador µDX201 para versão 3.87.

Atenção: *É preciso utilizar Editor PG versão 2.2.0.0 ou mais recente com controlador µDX201. O Compilador PG deve ser versão 2.08 ou superior.*

Compilador PG para µDX200

O programa Compilador PG permite compilar os arquivos fonte gerados pelo Editor PG, assim como efetuar comunicação serial com o controlador µDX200.

0.65	Dez/2004	Versão inicial, ainda não operacional.
0.81	Jan/2005	Versão inicial, operacional.
0.984	Mai/2005	Blocos de SWAP (aritmética) e TX232 binário (word e byte) incluídos. Mensagem de confirmação quando programa tem controle de CTS.
0.985	Mai/2005	Transmissão do programa agora em blocos de 16-em-16 words para acelerar upload.
0.986	Mai/2005	Melhorado o controle que evita tentar duas comunicações com o uDX antes de receber a resposta da primeira. Resolvido o desaparecimento da janela tipo torta quando ocorria falha de transmissão de programa. Mudança de endereco DXNET de comunicação pode agora ser feita clicando com mouse na figura da CPU.
0.987	Mai/2005	Calcula corretamente o endereço de variável RESULTADO ou DESTINO quando o usuário emprega notação Mxxxx, evitando erro por causa do offset de Var_Out. Reduzida a transmissão de programa para porções de 8 em 8 words de forma a evitar falhas de comunicação remota.
0.988	Mai/2005	Ajusta timeout conforme baudrate da comunicação, permitindo comunicar em 300bps. Mostra baudrate no título da janela principal. Apaga área de status quando ocorre falha de comunicação.
0.989	Mai/2005	Corrigida a alocação de variáveis.
0.990	Mai/2005	Corrigido o tempo de wait após enviar senha primária. Corrigido a identificação de nodo default=0 (em alguns casos estava assumindo nodo 1).
0.991	Mai/2005	Ao enviar programa sai fora do loop de envio assim que detectar timeout.
0.992	Mai/2005	Força temporização no envio de senha com 500ms antes e 700ms depois. Testa consistência da configuração de USART0 ao enviar programa e solicita confirmação se necessário. Eliminada a opção de 150BPS na configuração de RS232.
0.993	Mai/2005	Melhorada a obtenção do nome do programa a partir do nome do arquivo.
0.994	Mai/2005	RTS e DTR ligados ao iniciar.
0.995	Jun/2005	Bloco uDX100 para DXNET+ reconhecido.
0.996	Jun/2005	Varredura de configuração da porta serial.
0.997	Jun/2005	Incluído um teste rigoroso para o limite de endereço de variáveis de resultado dos blocos.
0.998	Jul/2005	Integração de scripts de I2C conforme a ordem no arquivo I2C_DESC.TXT.
1.01	Set/2005	Leitura do setor TABSCAN. Envio de senha no menu GERAL.
1.02	Nov/2005	Envia senha escolhida pelo usuário ao executar comando de Envio de TABSCAN. Aparece uma janela para digitar a senha. Se o usuário nada escrever, apenas teclar ENTER, a senha enviada será a padrão ([uDX200). Modificado o comando de envio de senha secundária para "Altera Senha". Este comando altera a senha que será enviada pelo botão do painel principal (desenho de uma casa). Se não digitar nada a senha voltará a ser a padrão.
1.03	Nov/2005	Lê a senha secundária.

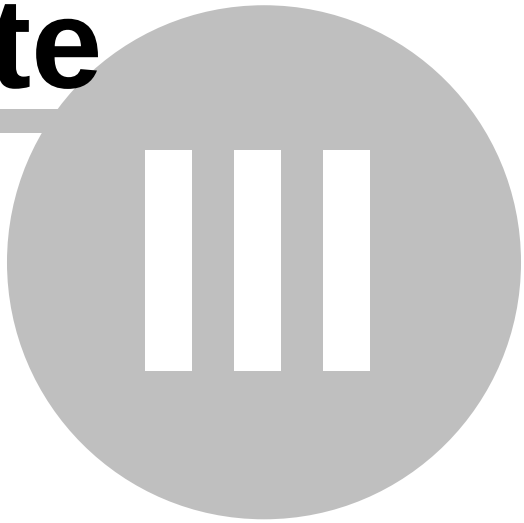
1.04	Jan/2006	Corrigido erro de reconhecer eco de modem como uDX200 conectado. Corrigida gravação do UDI antes dele ser propriamente criado. Implementada a compilação do bloco MODBUS. Implementado salvar configuração completa da serial em udx200.cf.
1.06	Mar/2006	Nova janela de codificação de senha secundária, eliminação da configuração serial para 7 bits (forçado sempre em 8 bits).
1.07	Mar/2006	Compilador aceita endereços absolutos de variáveis entre 0200h e 0980h.
1.08	Mar/2006	Corrigida falha de gravação do arquivo UDI.
1.09	Mar/2006	Mensagem de confirmação ao enviar Hard Reset. Mostra msg de erro caso não consiga enviar RESET após transmissão do programa.
1.10	Mar/2006	Suporte a palavras chaves iniciadas com _ (underline) incluindo leitura desde 0100h.
1.11	Mar/2006	Verifica versão do firmware (uDX200) para fazer a calibração, leitura de TABSCAN corretamente.
1.12	Mai/2006	Vários ajustes: limita em 80 variáveis para monitoração (apesar de mostrar todas que tiver); mostra estatística mais detalhada do programa compilado; testa corretamente o uso da RAM para limitar programas grandes.
1.13	Mai/2006	Mostra variáveis tipo Real (em hexa), reconhece constantes Real, duas novas palavras-chave adicionadas: _VARIN e _VAROUTPT.
1.14	Mai/2006	Mostra var REAL em formato científico com 6 casas após a vírgula.
1.15	Set/2006	VBAT com mais um dígito. Envio de senha e processamento de comunicação melhorados.
1.19	Nov/2006	Reconhece os blocos de NOVO-VAR e VAR-NODO.
1.20	Dez/2006	Incluída a opção OP para reconhecer parâmetro no bloco Lint->Int.
1.21	Jan/2007	Incluído o ajuste de offset para timeout de comunicação na janela Preferências. Este ajuste é memorizado no arquivo .CF para voltar com o mesmo valor ao re-entrar no Compilador. Com ajuste zero o tempo de timeout é identico às versões anteriores. Qualquer valor indicado será apenas adicionado ao valor base. O valor base pode ser de 500ms ou 2500ms, conforme a faixa de baud-rate (acima ou abaixo de 1200bps).
1.24	Set/2007	Correções no tratamento de cartão MMC, correções no tratamento de variáveis em caracteres maiúsculos e minúsculos.
1.27	Jan/2008	Correção para número elevado de I/Os (o número de I/Os só aumentava para determinado programa, independentemente da quantidade de entradas e saídas efetivamente utilizadas). Permite formatar cartões acima de 1GB.
1.28	Mai/2008	Corrigido o calculo do limite de variáveis.
1.29	Jul/2008	Corrigido o bloco de calendário, contudo não permite uso de variável para o parâmetro do ANO.
2.00	Jul/2008	A partir da versão 2.1.0.0 do Editor PG este engloba também o Compilador PG, formando um ambiente de desenvolvimento único. Melhorias significativas na visualização de nodos e variáveis monitoradas, e também de dados de status do CLP. Possibilidade de comunicação via Ethernet (endereço IP) e via Modem, além da comunicação serial. Lista de nodos e variáveis utilizados no programa aplicativo. Formatação de cartão MMC totalmente reformulada. Transmissão do programa pré-compilado (.UDP) automaticamente para o Compilador, sem a necessidade de abrir o arquivo manualmente.

2.01	Out/2008	Alocação de variáveis para blocos I^2C prioritária (já que estas variáveis devem estar abaixo de V256). Quando não usado, o parâmetro FAIXA dos blocos de Comparação não deve alocar variáveis auxiliares, e sim ser considerado igual a zero.
2.05	Jun/2009	Correção na interpretação de literal ´;´ no Compilador. Correção nos blocos CALENDÁRIO e RELÓGIO.
2.06	Jun/2009	Correção na alocação de nodos absolutos.
2.07	Set/2009	Resolvido erro devido a inexistência de especificação de parâmetro FAIXA em blocos de Comparação. Retirada a tentativa de correção introduzida na versão 2.01, e substituída por aviso na compilação (warning).
2.08	Nov/2009	Previsão de controlador µDX201, com blocos de IHM. Correção na alocação de tabelas RAM. Acréscimo de palavras reservadas para µDX200 e µDX201.
2.09	Dez/2009	Várias melhorias na compilação de blocos RS232 para µDX201.
2.10	Mar/2010	Correção em bug introduzido na versão 2.06 que não permitia a correta alocação de nodos absolutos abaixo de 32 (via rótulos).
2.11	Jul/2010	Correção em tabela RAM tipo byte.
2.12	Jul/2011	Correção em bloco Nodo sem conexão (nodo "voando").
2.13	Ago/2011	Melhoria no relato de erros de compilação.
2.14	Abr/2012	Aumento de constantes e blocos disponíveis no programa aplicativo, evitando erro 0112 no Compilador.
2.15	Jan/2013	Modificado Compilador para que o mesmo informe posição de erros na compilação, permitindo "pular" para o bloco que originou o erro.
2.16	Out/2013	Aumentado limite interno de memória, permitindo compilar programas muito extensos (acima de 2500 blocos) sem a geração do erro interno 0112.
2.18	Jan/2014	Previstos novos blocos para µDX201, como blocos de comparação com saída invertida, limite de variável, MUX e DEMUX, flip-flop tipo T, etc. Novas palavras reservadas para acessar leitura de entrada analógica E1 em alta resolução (16 bits), e para leitura de freqüência em E9 e E10.
2.19	Mar/2015	Inseridas palavras reservadas _P9 e _P10 para leitura de período nas entradas E9 e E10 (em múltiplos de 125µs).
2.20	Mar/2015	Aumentado limite de tabelas em RAM no µDX201 (2048 bytes em vez de apenas 384 bytes como no µDX200).
2.21	Abr/2015	Ampliado limite de blocos de 3000 para 3500 blocos.
2.22	Mar/2017	Blocos para µDX201 para transmissão e teste de CRC CCITT.
2.23	Jun/2017	Permite uso de portas seriais auxiliares RS232-2 e RS232-3.
2.24	Jun/2017	Corrigido bug de supressão de variável ao inserir variável absoluta.
2.25	Jul/2017	Prevista programação de número de entradas analógicas lidas para conversão analógica/digital de alta resolução (palavra reservada _E1).
2.26	Ago/2018	Suporte para blocos IHM X (parametrizáveis via variáveis).
2.27	Ago/2018	Suporte para blocos de I2C com transmissão de CRC-8.
2.28	Out/2018	Previsão de blocos de Expansão µDX218 - Inicialização e Leitura.
2.29	Jan/2019	Aumentado limite de blocos do µDX201 de 3500 para 4500 (condicionado ao tamanho dos blocos usados no programa aplicativo).
2.30	Mai/2020	Melhorada indicação de limite de blocos. Indicação de erro ao exceder limite de nodos.
2.31	Jul/2020	Correção no cálculo de nodos usados pelo programa aplicativo.
2.32	Ago/2020	Retorno ao cálculo original de nodos usados pelo programa aplicativo. O cálculo introduzido na versão 2.31 estava errado.

2.33	Ago/2020	Palavras reservadas _CFGDXNET e _CFGRS232 inseridas. Blocos para Setup DXNET e Setup RS232 suportados. Palavras reservadas _NUMSER e _FIRMWARE inseridas.
2.34	Out/2020	Limite de bytes em tabelas RAM aumentado de 2048 para 4096.
2.35	Jun/2021	Palavras reservadas _E2, _E3, _E4, _E5, _E6, _E7, _E8.
2.36	Jan/2022	Palavras reservadas _E2, _E3, _E4, _E5, _E6, _E7, _E8 em outro endereço, de forma a não conflitar com pilha.
2.37	Mai/2023	Palavras reservadas adicionais.
2.38	Ago/2023	Corrigida compilação de blocos CRC-DNP e CRC-CCITT para porta serial RS232-3 .
2.39	Out/2023	Várias correções referentes a endereçamento de dispositivos.
2.40	Set/2024	Suporte para blocos de controle de motores via saídas analógicas.
2.41	Out/2024	Inserida decodificação de bloco move motores.
2.42	Out/2024	Decodificação de blocos Borda.
2.43	Out/2024	Palavras reservadas para controle de motores.

Atenção: *É preciso utilizar Editor PG versão 2.2.0.0 ou mais recente com controlador µDX201. O Compilador PG deve ser versão 2.08 ou superior.*

Parte



Expansão de Entrada/Saída µDX210

A Expansão de Entradas/Saídas µDX210 está equipada com saídas à relés eletromecânicos, ou ainda com relés de estado sólido. As especificações das saídas (S1 a S8) para o caso de relés eletromecânicos são as seguintes:

µDX210

Saída Relé Eletromecânico	30Vdc @ 2A 250Vac @ 2A Fusível 5A Interno

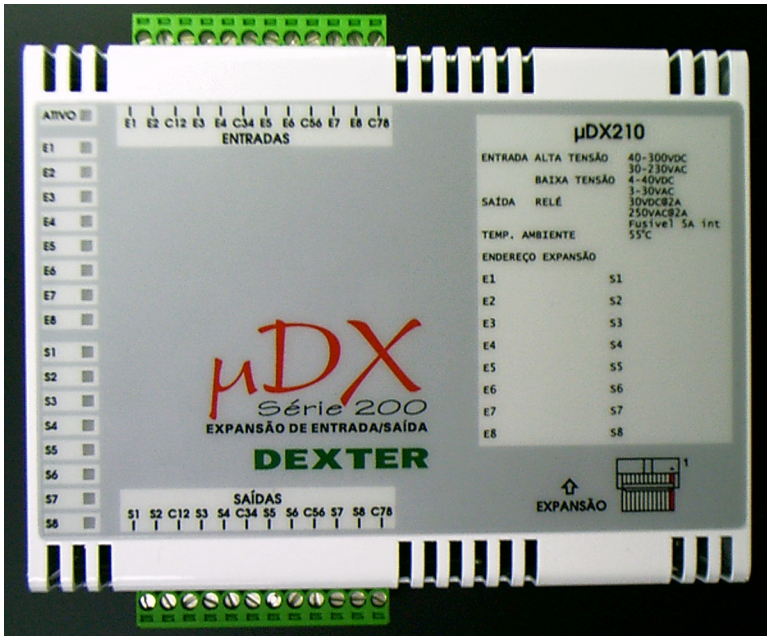
Já no caso de saídas a relés de estado sólido (SSR) as especificações das saídas S1 a S8 são (neste caso a Expansão de Entradas/Saídas é designada como µDX211):

µDX211

Saída Relé Estado Sólido	250Vdc @ 150mA 250Vac @ 150mA Fusível 315mA Interno

Note que o fusível, em ambos os casos, está ligado ao comum de duas saídas (C12, C34, C56, C78). Assim, seu valor está dimensionado para uma corrente um pouco superior à soma da corrente máxima em duas saídas.

É possível conectar até 32 Expansões de Entradas/Saídas µDX210 (ou µDX211) em um mesmo Controlador Programável µDX200. O cabo de Expansão pode ser fornecido pela Dexter para 1, 2, 4, 8, 16, ou 32 Expansões. Seu comprimento total máximo não deve exceder 2 metros.



A foto mostra os conectores de entrada e saída da Expansão µDX210. Note que tanto as entradas quanto as saídas da expansão possuem conectores de engate rápido, o que facilita manutenções. No conector superior temos:

E1 e E2 → Entradas digitais E1 e E2.
C12 → Comum das entradas E1 e E2.
E3 e E4 → Entradas digitais E3 e E4.

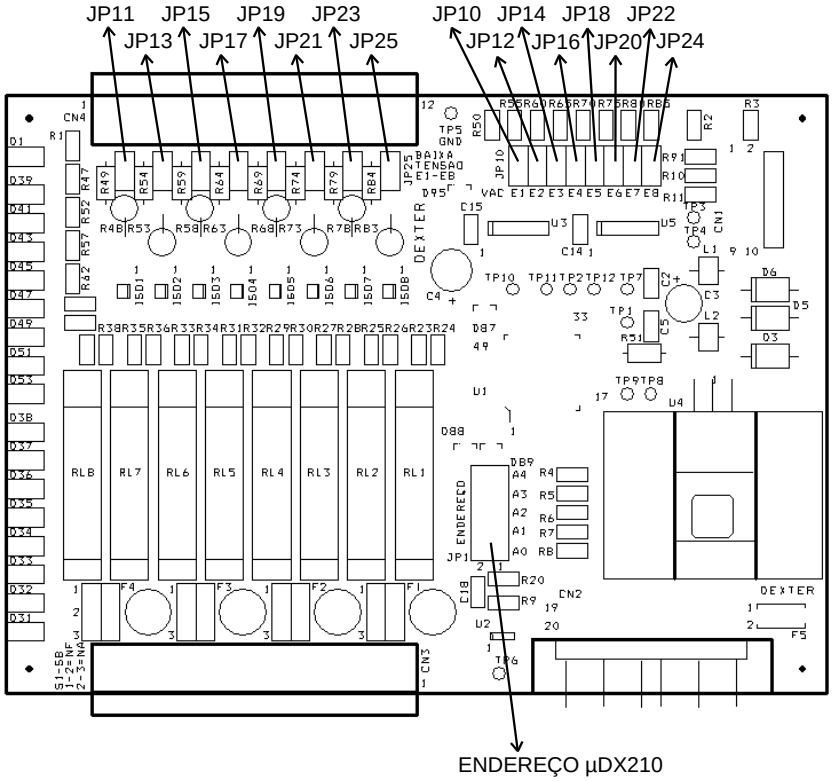
C34 → Comum das entradas E3 e E4.
E5 e E6 → Entradas digitais E5 e E6.
C56 → Comum das entradas E5 e E6.
E7 e E8 → Entradas digitais E7 e E8.
C78 → Comum das entradas E7 e E8.

Já o conector inferior possui as seguintes conexões:

S1 e S2 → Saídas digitais S1 e S2.
C12 → Comum para saídas S1 e S2.
S3 e S4 → Saídas digitais S3 e S4.
C34 → Comum para saídas S3 e S4.
S5 e S6 → Saídas digitais S5 e S6.
C56 → Comum para saídas S5 e S6.
S7 e S8 → Saídas digitais S7 e S8.
C78 → Comum para saídas S7 e S8.

Seleção de Jumpers

A especificação das entradas da Expansão µDX210 depende dos jumpers internos. Os jumpers da Expansão µDX210 têm a localização e função especificadas a seguir (no caso de módulo µDX211 - saídas em estado sólido - os jumpers são idênticos).



Note que os jumpers JP10 a JP25 se referem à programação de tensão das entradas do µDX210. Ainda existem cinco jumpers adicionais que permitem programar o endereço da Expansão (de módulo 1 até módulo 32). Para abrir a caixa metálica do µDX210 retire os dois parafusos (fenda cruzada) existentes nas laterais da caixa, e force levemente as laterais para que se afastem dos encaixes que a prendem ao fundo da caixa. Cuidado com os leds soldados a placa impressa, de

forma a não danificá-los.

Módulo	A5	A4	A3	A2	A1
1	0	0	0	0	0
2	0	0	0	0	1
3	0	0	0	1	0
4	0	0	0	1	1
5	0	0	1	0	0
6	0	0	1	0	1
7	0	0	1	1	0
8	0	0	1	1	1
9	0	1	0	0	0
10	0	1	0	0	1
11	0	1	0	1	0
12	0	1	0	1	1
13	0	1	1	0	0
14	0	1	1	0	1
15	0	1	1	1	0
16	0	1	1	1	1
17	1	0	0	0	0
18	1	0	0	0	1
19	1	0	0	1	0
20	1	0	0	1	1
21	1	0	1	0	0
22	1	0	1	0	1
23	1	0	1	1	0
24	1	0	1	1	1
25	1	1	0	0	0
26	1	1	0	0	1
27	1	1	0	1	0
28	1	1	0	1	1
29	1	1	1	0	0
30	1	1	1	0	1
31	1	1	1	1	0
32	1	1	1	1	1

O endereço do µDX210 é determinado pelos jumpers existentes no conector ENDEREÇO, sendo que jumper aberto significa nível 0 e jumper fechado nível 1. Note que o µDX200 é fornecido com os cinco jumpers de endereçamento abertos, ou seja, ocupando o endereço de módulo 1.

Lembre-se que todas as Expansões de Entrada/Saída µDX210 ligadas a um mesmo controlador µDX200 devem possuir endereços distintos.

Já os jumpers JP10 a JP25 programam a faixa de tensão das entradas do µDX210, além do tipo de sinal (corrente contínua ou alternada). Note que as entradas são todas optoisoladas, oferecendo isolação galvânica. Assim, é possível conectá-las diretamente à rede elétrica (127 ou 220Vac) ou usar várias fontes de sinal, sem conexão de referência (terra) entre elas.

Os jumpers internos permitem configurar as entradas do µDX210 para quatro tipos de sinal:

Alta tensão AC → 80 a 230 Vac (tensão alternada de 50 ou 60Hz)
Alta tensão DC → 80 a 230 Vdc (tensão contínua)
Baixa tensão AC → 4 a 30 Vac (tensão alternada de 50 ou 60Hz)
Baixa tensão DC → 4 a 30 Vdc (tensão contínua)

A configuração dos jumpers para cada uma destas configurações é dada abaixo, respeitando a convenção de 0 indicando jumper aberto e 1 indicando jumper fechado:

		Alta Tensão		Baixa Tensão	
Entrada	**Jumpers**	**DC**	**AC**	**DC**	**AC**
E1	JP10, JP11	00	10	01	11
E2	JP12, JP13	00	10	01	11
E3	JP14, JP15	00	10	01	11
E4	JP16, JP17	00	10	01	11
E5	JP18, JP19	00	10	01	11
E6	JP20, JP21	00	10	01	11
E7	JP22, JP23	00	10	01	11
E8	JP24, JP25	00	10	01	11

As Expansões de Entrada/Saída µDX210 são fornecidas de fábrica com as entradas programadas para alta tensão AC.

Atenção: *Caso alguma entrada esteja preparada para a opção de baixa tensão, a conexão direta à rede elétrica (127 ou 220Vac) provocará sua queima instantânea.*

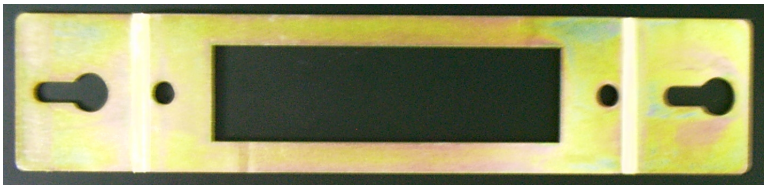
No caso de entrada em corrente contínua deve ser respeitada a seguinte polaridade: entradas E1, E2, E3, E4, E5, E6, E7 e E8 positivas; comuns C12, C34, C56 e C78 negativos.

Parte

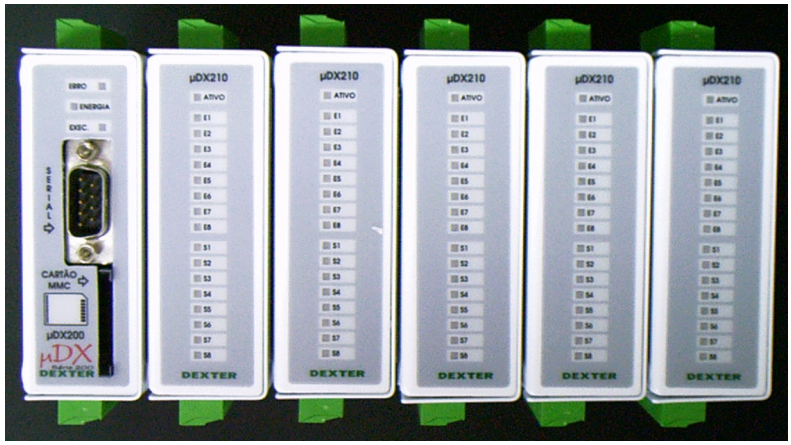
Fixação mecânica do µDX200 e µDX210

O Controlador Programável µDX200 e a Expansão de Entradas/Saídas µDX210 possuem exatamente as mesmas dimensões (115 x 86 x 30mm), e permitem montagem em duas posições. Isso porque todos os leds indicativos estão presentes tanto no painel frontal quanto na lateral dos equipamentos.

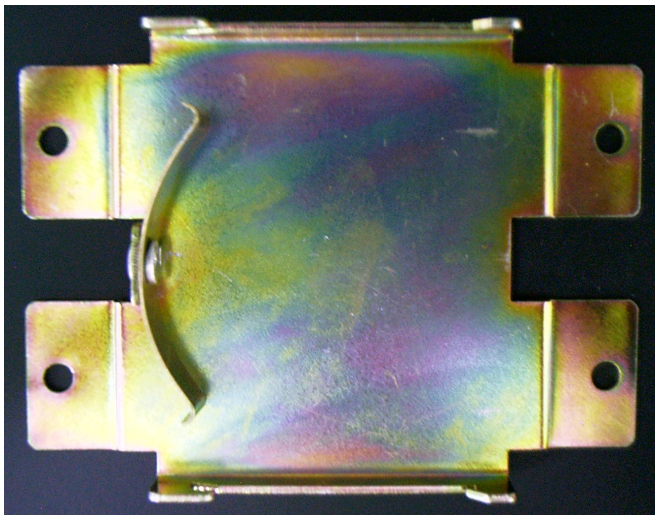
Para fixação acompanha tanto o µDX200 quanto o µDX210 um suporte para parafuso, como o visto abaixo:



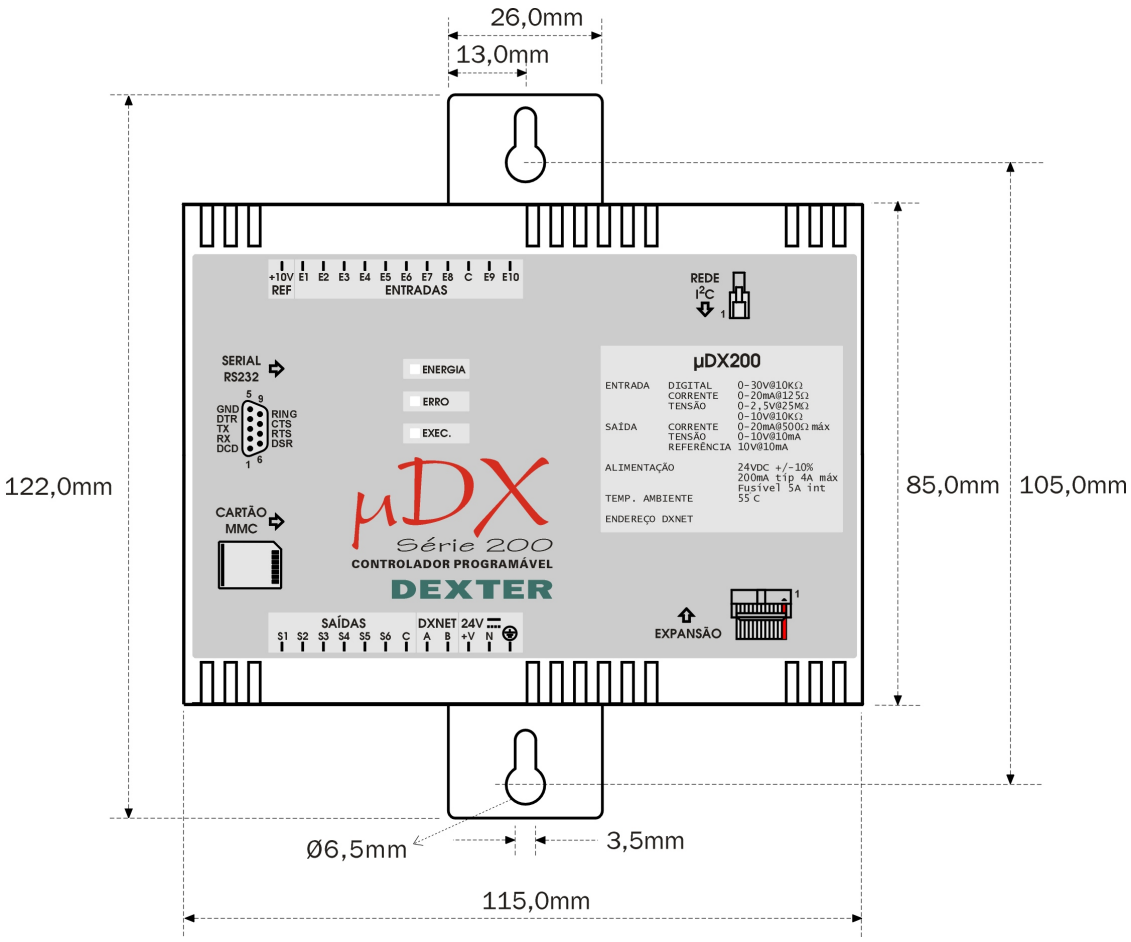
Este suporte pode ser fixado tanto no fundo dos equipamentos quanto na lateral desses, permitindo fixação em duas posições. No caso de projetos que envolvam grande quantidade de entradas e saídas (e, portanto, muitas Expansões µDX210), a montagem "em pé" oferece a vantagem de ocupar muito pouco espaço, como na foto a seguir, em que temos um Controlador µDX200 e cinco Expansões µDX210. Todo o conjunto, com 80 I/Os, ocupa uma área de apenas 185 x 86mm.



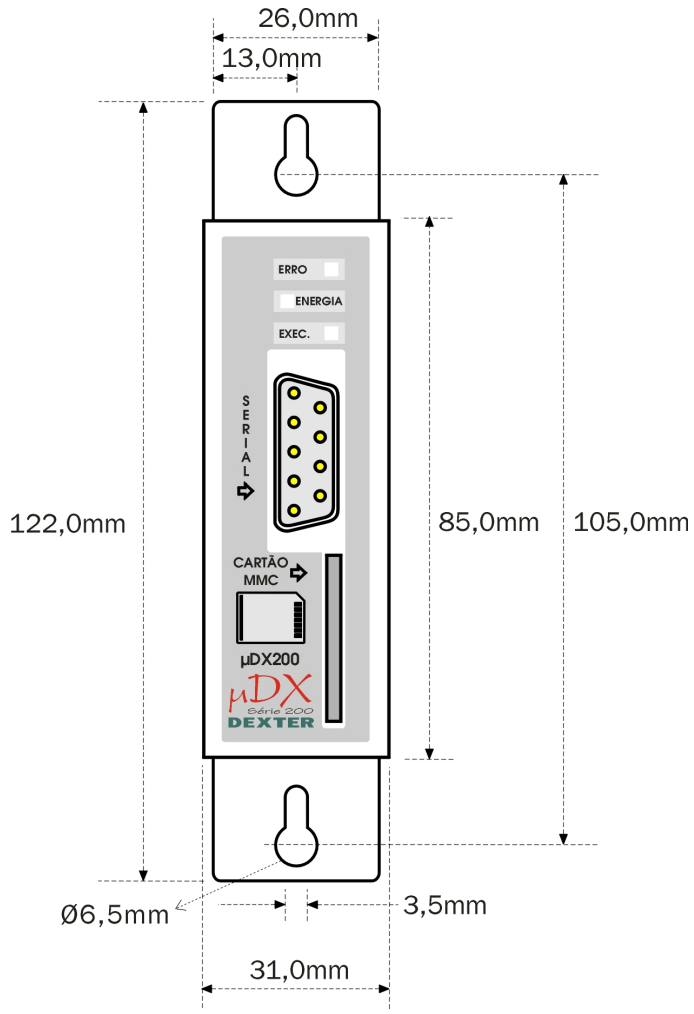
Para fixação em trilho DIN (35mm) a Dexter comercializa um acessório capaz de fixar dois dispositivos. Ou seja, um suporte para trilho DIN permite fixar, por exemplo, um controlador µDX200 e uma expansão µDX210.



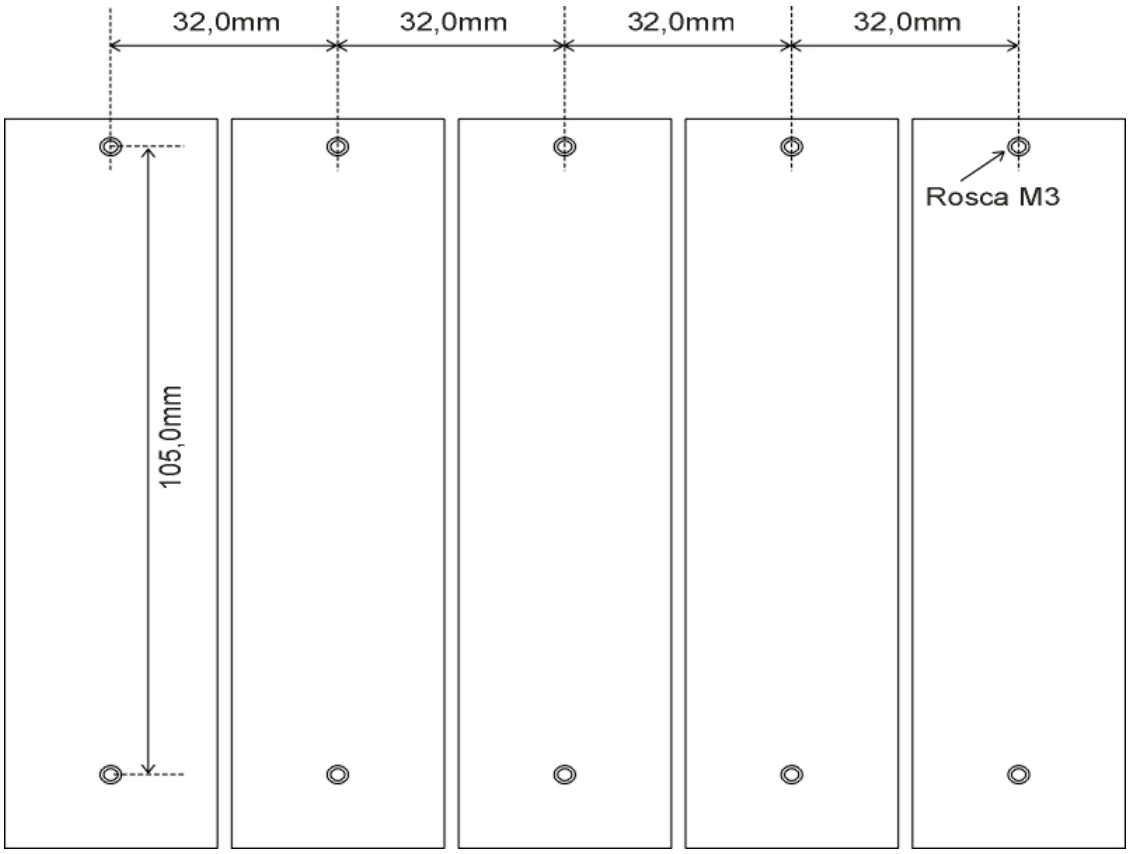
A seguir temos um desenho com as dimensões do suporte para parafuso do µDX200/µDX210, usando fixação pelo fundo da caixa:



Já nos desenhos a seguir os Controladores µDX200 e Expansões µDX210 são fixadas "em pé", de forma a economizar espaço no painel:



No caso de vários equipamentos montados nesta posição a furação do fundo deve ser a seguinte:



Parte

Instalações Industriais e Transitórios de Tensão

Em uma instalação industrial é comum a ocorrência de transitórios de alta tensão. Estes transitórios podem afetar o funcionamento de equipamentos eletrônicos, que normalmente operam em tensões baixas, como 3,3V ou 5 V. Os transitórios são originados da comutação de cargas indutivas, descargas atmosféricas e transitórios oriundos da rede elétrica, além de descargas eletrostáticas.

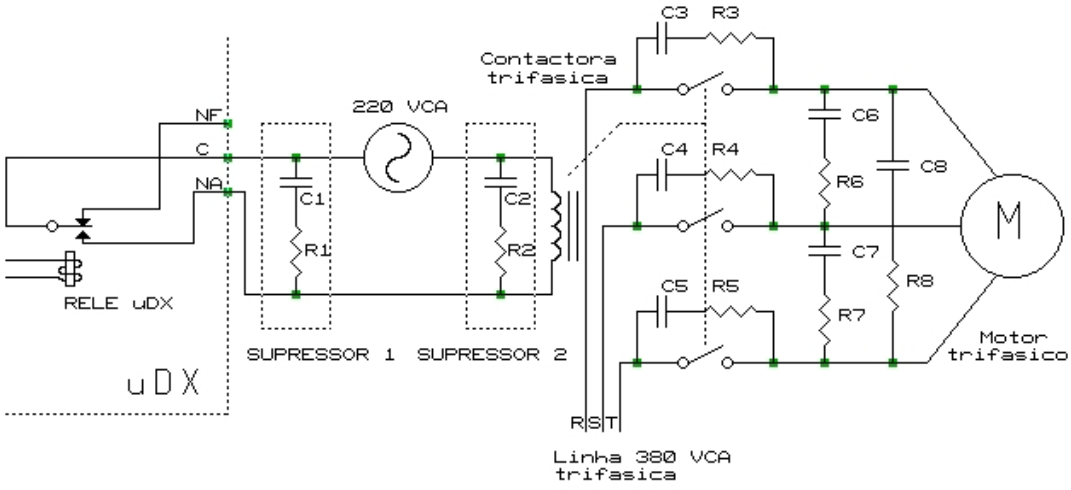
O controlador µDX200 possui vários componentes internos para protegê-lo de EMI (interferência eletromagnética), EMS (descargas eletrostáticas), e ainda descargas elétricas de alta energia. Ele foi testado com descargas de 1500V e energia de 3 joules em ciclos de 100 descargas, cada uma com duração total de cerca de 20µs (potência acima de 100KW). Isso garante um equipamento robusto e adequado a ambientes hostis, como comumente encontrados na indústria.

Cargas Indutivas

O controlador programável µDX Série 200 possui considerável proteção contra transitórios de alta tensão, como gerados na comutação de cargas indutivas (motores, solenóides, contactoras, etc.). Mas é aconselhável instalar supressores R-C (resistor + capacitor) para absorver e dissipar a força contraeletromotriz armazenada nas cargas indutivas, que é liberada na abertura dos contatos dos relés do µDX210. Estes supressores, além de evitar problemas no funcionamento do controlador programável, aumentam muito a vida útil dos contatos de relés, pois evitam o arco elétrico criado quando esses contatos abrem.

O supressor R-C é formado por um resistor e um capacitor em série, ligados em paralelo com o contato do relé e em paralelo com a carga indutiva. O capacitor absorve a energia liberada pela carga indutiva ao repentinamente ter a corrente interrompida e o resistor serve para dissipar esta energia (amortecer o circuito L-C formado).

Teoricamente, bastaria um supressor R-C em paralelo com a carga que gera o transitório. Entretanto, normalmente os cabos de conexão do µDX210 à carga são longos. Ora, cabos longos são indutivos. Assim, é aconselhável também um filtro R-C junto ao relé do µDX para suprimir o transitório gerado pelos cabos de conexão, no caso desses serem longos (maiores que 5 metros).



Muitas cargas possuem uma corrente de manutenção (que as mantêm acionadas) muito baixa. Por exemplo, um solenóide para uma válvula pneumática, pode acioná-la somente aplicando uma tensão próxima de 220VCA. Mas, uma vez acionada a válvula, tensões muito mais baixas aplicadas ao solenóide a mantém neste estado. Ou seja, o supressor R-C instalado em paralelo com os contatos do relé do µDX deve ter uma corrente quiescente baixa, sob risco de manter a carga indefinidamente energizada. Uma solução é fazer C2>>C1, de forma que a corrente do supressor 2 seja muito maior que a do supressor 1, garantindo a desenergização da carga.

No caso de acionamento de cargas não-indutivas (lâmpadas, resistências, etc.) basta a instalação do supressor em paralelo com os contatos do relé do µDX.

> ***ATENÇÃO:*** *Se o supressor estiver em uma instalação de corrente alternada existirá uma corrente quiescente devido ao capacitor. Isto pode acasionar choques elétricos ao manipular a carga, mesmo com os contatos do relé do µDX210 abertos.*

Se existirem contactoras acionando cargas indutivas (como motores, por exemplo), além dos supressores no contato do relé do µDX210 e em paralelo com a bobina da contactora, é interessante instalar supressores R-C em paralelo com a carga acionada pela contactora e em paralelo com os contatos dessa. Estes supressores em paralelo com o motor e em paralelo com os contatos da contactora evitam a formação de arco voltaico ao abrir a contactora, e portanto aumentam a vida útil dessa.

A seguir temos uma tabela com algumas cargas indutivas e os respectivos supressores R-C recomendados:

	Capacitor	**Resistor**	**Varistor**
Contactoras (bobina)	1µF/630V	100Ω,10W	-
Contactoras (contatos)	0,47µF/630V	100Ω, 5W	-
Relés µDX210 (contatos) (p/275V)	0,1µF/630V	100Ω,1/2W	S07K275
Solenóides (bobina) (válvulas pneumáticas, p/exemplo)	1µF/630V	100Ω,10W	-
Motores(bobina)	1µF/630V	100Ω,10W	-

Note que os capacitores devem ser adequados ao regime de corrente alternada com altas correntes de "ripple", como os capacitores de poliéster metalizado. Por exemplo, a série MKT da Siemens (MAC FITA - B32232).

A Dexter comercializa supressores de ruído com as especificações acima, sendo designados como:

Filtro pequeno	Capacitor 0,1µF + Resistor 100R 1/2W + Varistor 275V.
Filtro médio	Capacitor 0,47µF + Resistor 100R 5W.
Filtro grande	Capacitor 1µF + Resistor 100R 10W.

Gabinete

A caixa do controlador programável µDX200 e de seus acessórios não é adequada a uma instalação industrial, pois não tem proteção contra pó ou água, além das conexões elétricas ficarem expostas. Portanto, normalmente estes equipamentos são instalados dentro de outro gabinete metálico e aterrado (tanto o fundo quanto a tampa).

> ***ATENÇÃO:*** *O µDX200 e seus periféricos (Expansão de Entradas/Saídas µDX210, µDX211, etc.) não devem ser instalados no mesmo painel das contactoras. Além disso, os cabos das entradas e das saídas do µDX200 e da Expansão µDX210 devem ser mantidos o mais afastados possível. Cabos de alimentação elétrica para o equipamento e o cabo da rede DXNET também devem ser mantidos afastados dos cabos de saídas. Os cabos da rede DXNET, assim como os cabos das entradas do µDX200 e os de alimentação elétrica para os equipamentos não devem ser misturados a cabos de força.*

Utilize sempre duas calhas ou eletrodutos metálicos aterrados para agrupar os cabos de sinal. Onde for impossível evitar o cruzamento entre cabo de sinal e de força, este deve ser feito à 90, minimizando o acoplamento eletomagnético entre ambos. Nunca coloque cabos de sinal e de força em paralelo. Neste caso use um eletroduto metálico aterrado para isolar os cabos de sinal (entradas do µDX, DXNET, etc.).
Ainda referente ao gabinete para acondicionar µDX+periféricos, convém envolver os cabos de força que chegam as saídas da Expansão µDX210 com uma malha metálica, evitando irradiação eletromagnética dentro do gabinete. Outra providência interessante é utilizar um lado do gabinete para a fiação de entradas de baixa tensão, entradas e saídas analógicas, rede DXNET e sensores, e o outro lado (o mais afastado possível) para a cablagem de força (saídas de relé).

Alimentação elétrica

O µDX200 pode ser alimentado com uma fonte de 24Vdc regulada. O controlador consome cerca de 150mA em 24Vdc. Já cada Expansão µDX210 pode acrescer cerca de 100mA. Assim, um µDX200 com 2 µDX210, por exemplo, pode consumir 350mA. Evidente que este cálculo não leva em consideração sensores e outras cargas que por ventura sejam alimentados também pela fonte de 24V. Não utilize a fonte de alimentação do µDX200 para acionar cargas indutivas, como válvulas solenóides ou motores.

Parte

Programação em PDE - Utilização do PG

Acompanha o Controlador µDX201 um CD com o software de programação PG (Programador Gráfico), composto de Editor e Compilador, para computador IBM-PC compatível. Este software deve rodar sob plataforma Windows (Windows98, Windows Millenium, Windows NT, Windows 2000, Windows XP ou Windows Vista).

O Editor PG é uma ferramenta com a qual se pode elaborar ou modificar programas para o µDX (em linguagem PDE) e que, em conjunto com o programa Compilador PG (que acompanha o pacote de software) permite monitorar e interferir em qualquer µDX conectado na Rede Local DXNET. A partir da versão 2.1.0.0 o ambiente de desenvolvimento é integrado, de forma que o Editor e o Compilador PG compartilham a mesma interface de software. Assim, embora sejam gerados na instalação dois atalhos conforme figura acima, ambos rodam o mesmo programa. Apenas a diretiva em cada atalho é diferente, abrindo o PG em tela de Edição de Programas ou em tela de Compilação de Programas.

Atenção: Cuidado para não abrir duas vezes o PG inadvertidamente. Para passar do Editor para o Compilador e vice-versa use a tecla [Compilador/Fontes] (F5) existente no ambiente integrado. Caso seja chamado um atalho (por exemplo, µDX PG Editor) e depois o outro (µDX PG - Compilador) serão executados dois softwares PG simultaneamente no computador.

Esta linguagem PDE - Programação por Diagrama Esquemático - foi desenvolvida pela DEXTER como um meio de programação intuitiva, de fácil compreensão e que dispensa conhecimentos especializados. O novo PG mantêm a filosofia de programação inaugurada pelo programa PG para linha de controladores µDX100, mas expande bastante suas possibilidades, ao permitir múltiplas janelas, conexão com transporte de valor de variáveis, e muito mais blocos.

Para elaborar um programa basta "pegar" os blocos de instrução que sejam necessários (dispostos em forma de Menu), colocando-os depois na área de programação. Depois liga-se uns aos outros com fios e a representação artística final é a de um diagrama esquemático, com as chaves, relés, temporizadores, fios, fontes de energia e tudo o mais que o programa precisar. Textos podem ser inseridos no desenho para explicar alguma operação ou indicar a finalidade das entradas e saídas. E quando for necessário alterar o programa basta apontar o mouse e mover linhas e blocos que se desejar, apagar ou inserir novos blocos e linhas.

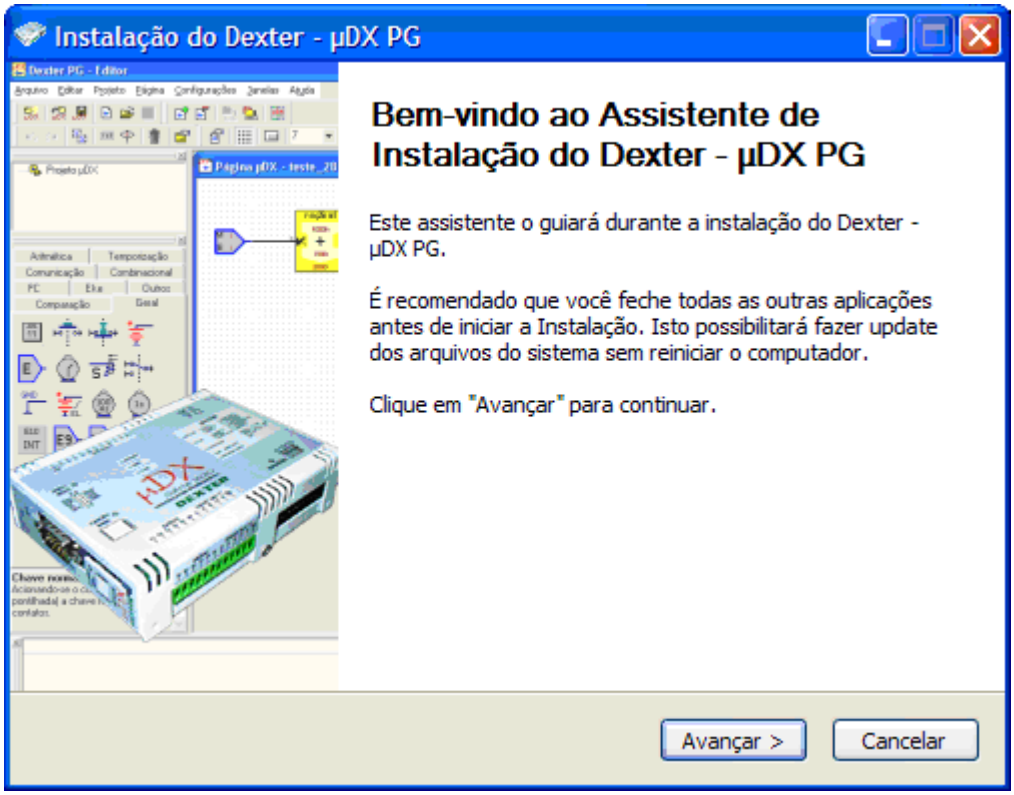
Finalmente, através de um único comando, o programa Editor PG pode chamar o módulo de Compilação, capaz de compilar e enviar o programa a qualquer µDX conectado a rede DXNET, sem perda de tempo.

Além disso, o Compilador permite monitorar valores de variáveis, forçar outros valores ou ainda ligar algum nodo de qualquer µDX. Por exemplo, através de um só comando pode-se ligar um relé de um µDX que está até 1500 metros distante do computador.

A partir da versão 2.2.0.0 o software PG prevê dois controladores distintos: µDX200 e µDX201.

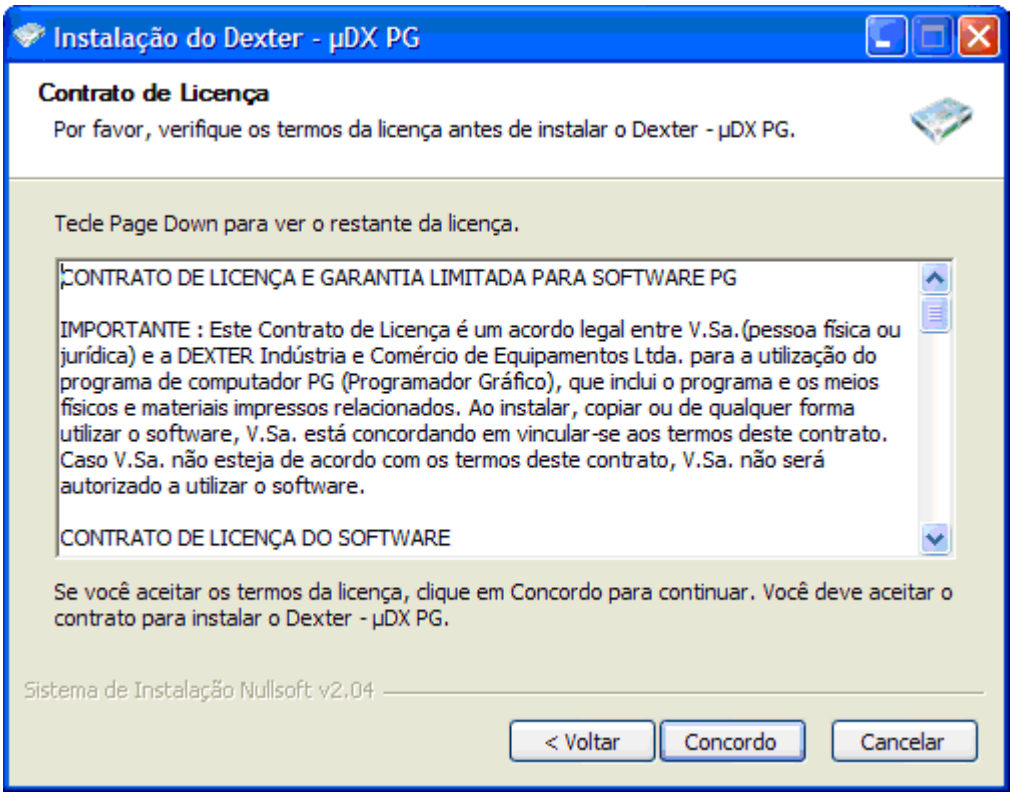
Instalação do software PG

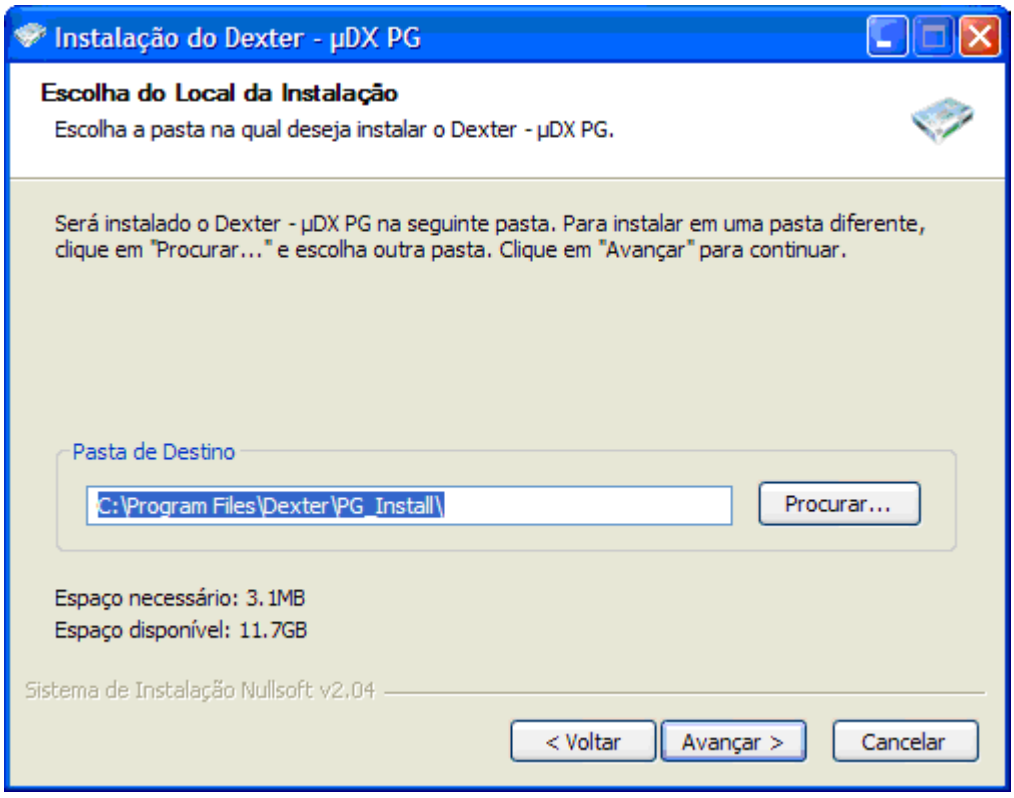
Para instalar o software PG (Programador Gráfico) basta inserir o CD que acompanha o Controlador Programável µDX200 no leitor de CD do computador IBM-PC compatível. É recomendável o uso de computador com, no mínimo, 100MHz de clock. O PG roda sob sistemas operacionais Windows 98, Windows Millenium, e Windows XP. Caso a tela de inicialização não surja abra o diretório do CD via Windows Explorer e rode o programa **Instalar.exe** existente no CD.



A seguir, é apresentado um contrato de licença de uso para o software PG. Caso haja alguma cláusula do referido contrato que não seja aceitável de seu ponto de vista clique em [Cancelar] e entre em contato com a Dexter.

Se o contrato for aceito o software PG será instalado, normalmente em c:\Arquivos de Programas\Dexter\PG.



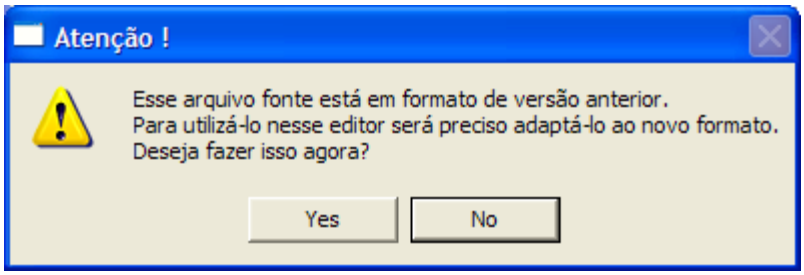


O instalador irá criar dois atalhos na tela do Windows, um para o Editor PG e outro para o Compilador PG:

Atenção: Estes dois atalhos chamam o mesmo programa - PGEditor.exe - a partir da versão 2.1.0.0 do software, apenas com diretivas diferentes, de forma a abrir na edição ou na compilação de programas aplicativos para o µDX200. Cuidado para não abrir duas vezes o PG inadvertidamente. Para passar do Editor para o Compilador e vice-versa use a tecla [Compilador/Fontes] (F5) existente no ambiente integrado. Caso seja chamado um atalho (por exemplo, µDX PG Editor) e depois o outro (µDX PG - Compilador) serão executados dois softwares PG simultaneamente no computador.

Compatibilidade com Versões Anteriores

O software PG, a partir da versão 2.1.0.0, incorpora o Compilador PG em uma única interface de software. Também disponibiliza a facilidade de gerar Macros, tornando possível a reutilização de programas aplicativos em novas aplicações. Com isso, as páginas de programação elaboradas em versões anteriores não são compatíveis com as novas versões do PG. Mas, no caso de páginas, o próprio PG possui dispositivos para convertê-las ao novo formato. Quando se tenta abrir uma página de programa aplicativo gerada em versões antigas do Editor PG na versão 2.1.0.0 ou superior do PG surge a seguinte janela:



Ao selecionar [**Sim**] ou [**Yes**], caso seu sistema operacional esteja em inglês, o PG converte a página para o novo formato. Entretanto, não é detectada nenhuma modificação na página devido a esta conversão. Então, é aconselhável fazer uma pequena alteração na página (por exemplo, arrastar um bloco uma posição e retorná-lo à posição original) para habilitar a opção (Salvar página ou macro) e clicar nesta opção para salvar a página neste novo formato (assim, a próxima vez que esta página for chamada já estará no novo formato).

Já projetos elaborados em versões anteriores do PG não são compatíveis com a versão 2.1.0.0 ou superior, e não existe um mecanismo de conversão como no caso de páginas de programação. Neste caso, é mandatório abrir um novo projeto no novo PG e incluir as páginas existentes no projeto original.

A partir da versão 2.2.0.0 o software PG prevê dois tipos de controladores programáveis: µDX200 e µDX201.

Atualização do PG

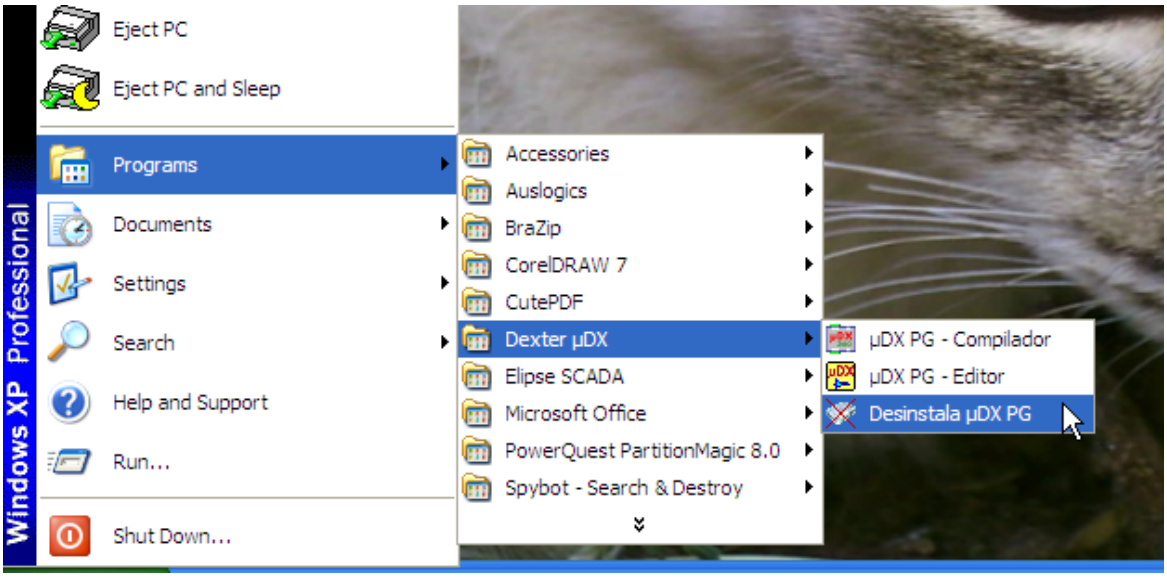
A partir da versão 2.1.2.0 o software PG verifica no site da Dexter - www.dexter.ind.br - se existem versões mais recentes para sua atualização. Neste caso se recomenda a desinstalação da versão atual do PG e instalação da versão atual. Para isso siga os procedimentos a seguir:

Passo 1

Feche o software PG, salvando os arquivos porventura criados no mesmo.

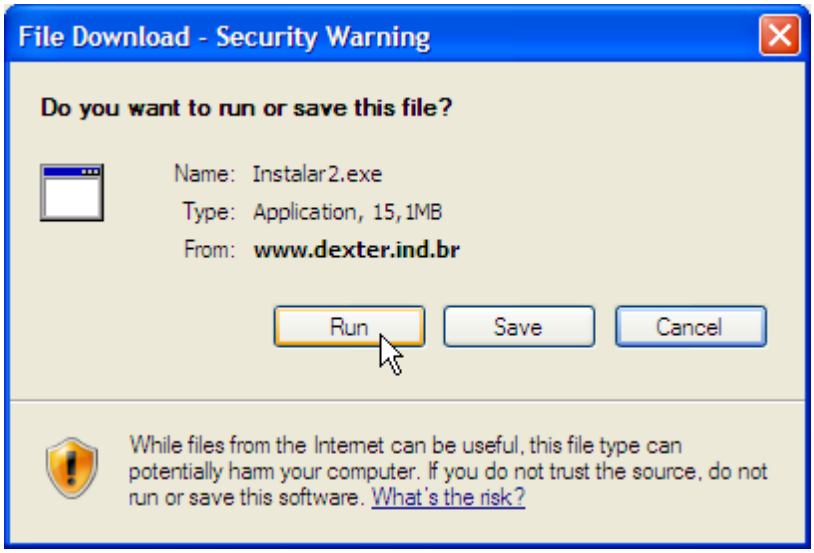
Passo 2

Utilize o ícone de desinstalação existente no menu **Iniciar** → **Programas** → **Dexter µDX** → **Desinstala µDX PG** do Windows:



Passo 3

Abra seu navegador de Internet (browser) e vá em http://www.dexter.ind.br/pg1.htm e faça o download da última versão do software PG. Pode-se optar por executar o programa:



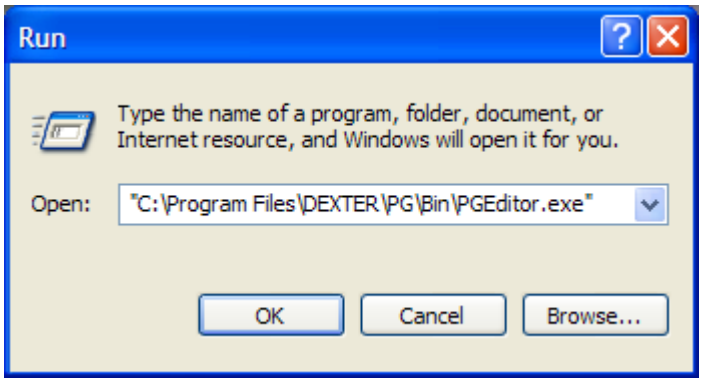
Passo 4

Siga os passos de instalação descritos no item Instalação do software PG. Está pronta sua atualização do PG! Note que todos os programas aplicativos para o PG que existiam nos diretórios de instalação (c:\Arquivos de Programas\Dexter) são preservados.

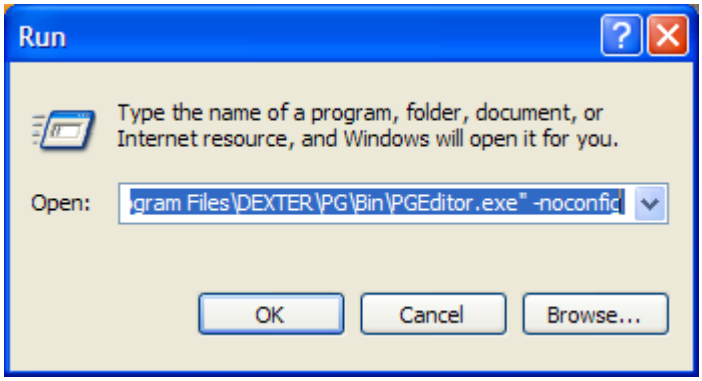
Diretivas de Linha de Comando

O programa **PG** (Programador Gráfico) permite uma série de diretivas a nível de linha de comando. Isso pode ser útil no caso de haver degradação em arquivos de configuração ou de bibliotecas, impedindo que o programa inicie.

Para rodar o programa via linha de comando vá em **Iniciar** → **Executar...** no Windows (**Start** → **Run**, caso seu sistema operacional esteja em inglês) e selecione o programa **PGEditor.exe**:



Digite após as aspas finais um espaço em branco e a diretiva desejada. Por exemplo, para que o PG ignore as configurações no Registro do Windows, agindo como se estivesse rodando pela primeira vez desde a sua instalação:



Esta diretiva pode resolver problemas oriundos de erros na instalação do PG, ou instalação de múltiplas versões do PG, restaurando o registro do PG no sistema operacional Windows.

Parâmetros na linha de comando

-newpack Faz com que o editor ignore a ausência do arquivo de lista de bibliotecas.

-noconfig Apaga as configurações existentes no registry, se houver, e as ignora, agindo como se estivesse iniciando pela primeira vez.

-odxp %1 Abre o projeto %1.

-odxg %1 Abre o arquivo de página %1.

-odxm %1 Abre o arquivo de macro %1.

-oudp %1 Abre o arquivo compilado %1.

-comp Abre a janela do compilador.

Tipos de Arquivo

O software PG gera uma série de arquivos com sufixos distintos. São eles:

Arquivos sufixo:	Função do Arquivo:
DXP	**Arquivo de projeto do PG (várias páginas DXG reunidas em um único projeto).**
DXG	**Página de programação do PG.**
UDP	**Página ou projeto pré-compilado pelo PG.**
UDI	**Arquivo com configurações de hardware do µDX200 (mesmo nome do arquivo UDP associado).**
UDG	**Layout de monitoração para programa UDP (grava CRC do programa aplicativo para consistência)**
DXM	**Página de Macro-célula do PG.**
DMB	**Macro-célula pré-compilada pelo PG.**
CSV	**Arquivo lido do cartão MMC ou gerado pela monitoração gráfica no PG (compatível com planilhas tipo Excel).**
TXT	**Arquivo texto gerado com dados do cartão MMC, pela janela de log do Compilador, ou ainda pela Lista de Variáveis e Nodos do Compilador.**
BMP	**Arquivo bit-map com imagem da página de programação, gerada pela opção Arquivo → Exportar...**
JPG	**Arquivo bit-map compactado com imagem da página de programação, gerada pela opção Arquivo → Exportar...**

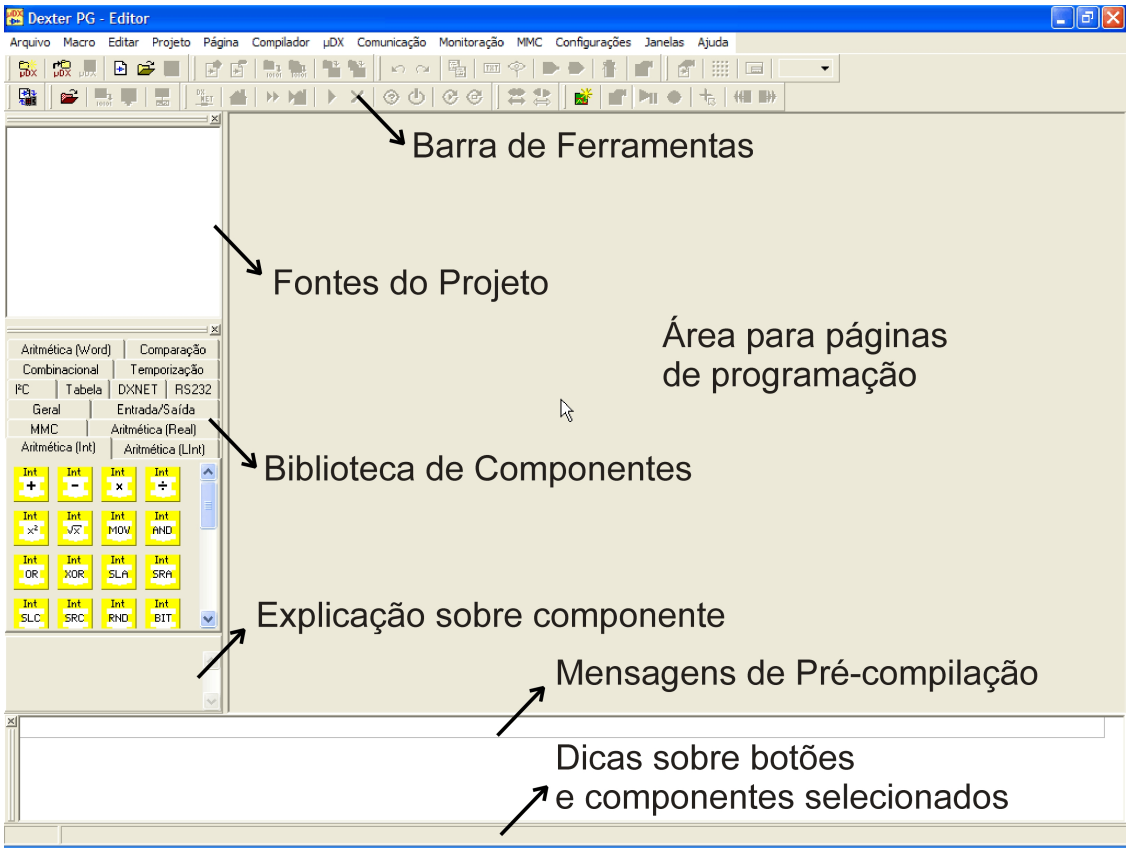
Teclas de Operação do Editor PG

Para utilizar o PG é necessário empregar um microcomputador compatível com IBM-PC com sistema operacional Windows 98 ou mais recente (Windows Millenium, NT, 2000, XP, Vista).

Caso ainda não tenha instalado o software PG siga as instruções constantes em capítulo anterior deste manual. No site da Dexter - www.dexter.ind.br - pode ser obtida a última versão do software Editor PG e do Compilador.

Ao rodar o Editor PG a primeira tela que aparece mostra uma foto do µDX200 e algumas informações, como a versão do software. Após alguns segundos surge a tela do PG.

A tela principal do Editor PG apresenta uma grande área retangular que é onde se localizam as páginas com os programas, uma área onde aparecem os escaninhos com os blocos utilizados para o desenho (Biblioteca de Componentes) e uma área reservada para mensagens de pré-compilação. Além disso, uma barra de ferramentas completa o conjunto, assim como teclas com menus pop-down para acesso as várias opções do software (muitas das quais estão disponíveis também na barra de ferramentas).



Os menus pop-down seguem a seguinte convenção:

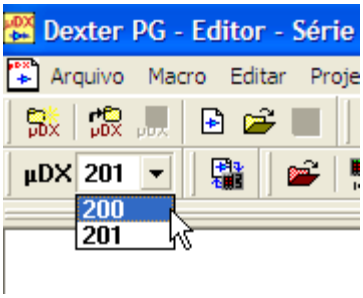
Funções imediatas: são designadas apenas pelo nome, sem terminação (exemplo: Arquivo → Salvar).

Funções que abrem janelas de opções: são terminadas por três pontos (exemplo: Arquivo → Salvar como...).

Funções que permitem selecionar um valor: são terminadas com seta ⇒ (exemplo: Página → Zoom).

Além disso, todas as funções que possuem teclas de atalho têm isto especificado após o nome da função (por exemplo, Página → Nova página Ctrl+N).

Atenção:
Note que, a partir da versão 2.2.0.0 do software PG, estão previstos múltiplos dispositivos, cada um dos quais com sua biblioteca de blocos e suas características próprias de compilação. Até o momento existem dois dispositivos: µDX200 e µDX201. Para determinar qual o dispositivo adotado na edição do programa aplicativo foi incluído um seletor na barra de ferramentas:



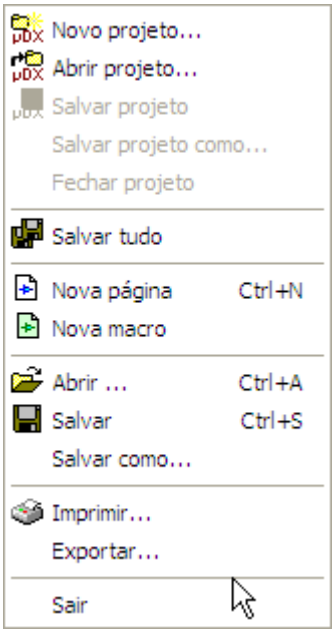
Este seletor especifica qual o dispositivo alvo na edição de programas aplicativos. Com isso, comuta a biblioteca de blocos para a biblioteca do equipamento escolhido. Entretanto, caso ao

abrir o Compilador o mesmo monitore equipamento diferente do especificado o programa aplicativo é recompilado, de forma a permitir sua transmissão. Se tiverem sido usados blocos que não sejam adequados ao dispositivo ocorrerá um erro de compilação (por exemplo, blocos de IHM em programa para µDX200).

No caso de projeto e não páginas de programação avulsas existe um campo próprio nas propriedades do projeto para especificar o dispositivo alvo (µDX200 ou µDX201). Assim, ao carregar um projeto automaticamente o PG se adapta ao dispositivo especificado pelo projeto.

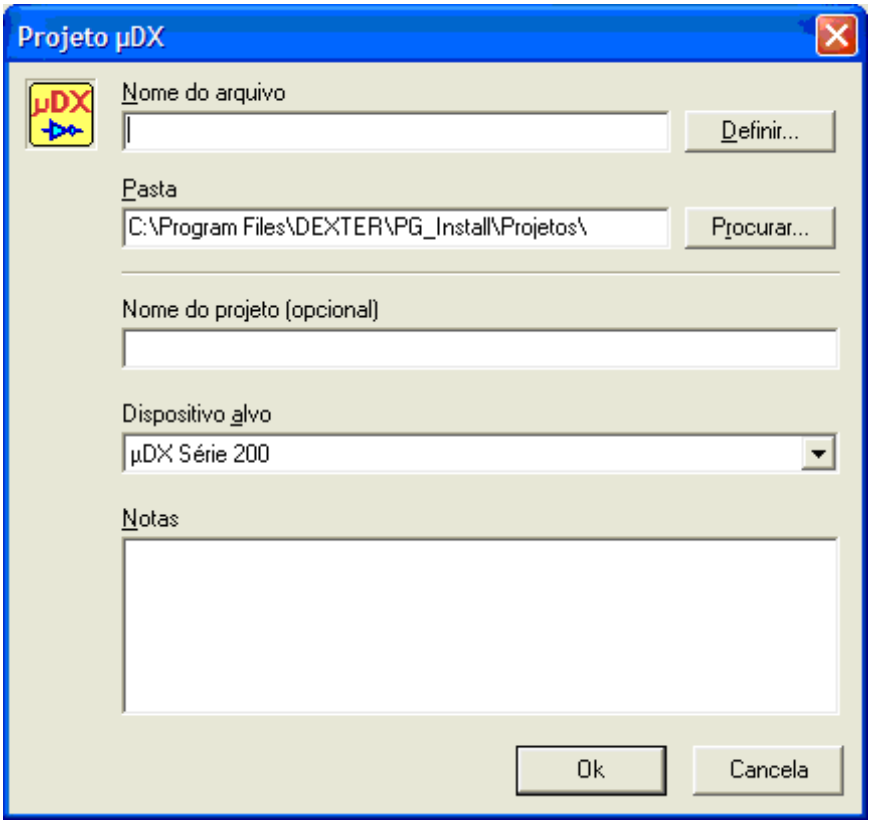
Menu Arquivo

Este menu permite gerenciar a leitura e salvamento de projetos ou páginas de programação, além de imprimir o programa e sair do Editor PG. Note que diversas funções estão disponíveis também na Barra de Ferramentas (as funções apresentam um ícone à esquerda que é usado também na barra de ferramentas), e três funções possuem teclas de atalho (Nova página - Ctrl+N, Abrir Página - Ctrl+A, e Salvar - Ctrl+S). As funções que não estão disponíveis aparecem em cinza.



Novo Projeto...

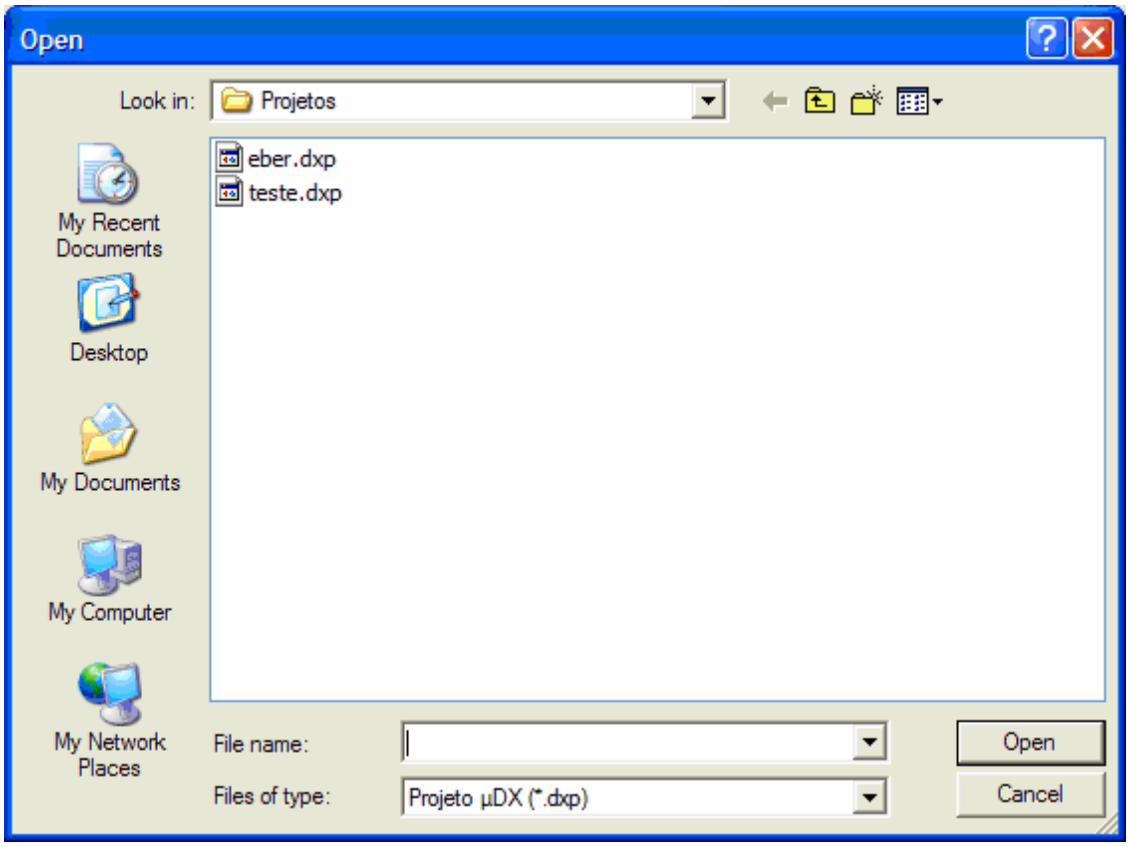
Esta tecla permite criar um novo projeto. Note que o Editor PG admite agrupar várias páginas de programação em um mesmo projeto. Com isso, ao se compilar o projeto todas as páginas serão compiladas e remetidas ao µDX200 como um programa único. Este recurso facilita sobremaneira a reutilização de partes de programas e a documentação de programas complexos. Ao selecionar [Novo Projeto...] surge a seguinte tela:



Esta tela permite determinar o nome do arquivo sufixo .dxp a ser criado, o diretório onde o mesmo será criado, um nome (opcional) para o projeto, o dispositivo alvo (que prevê µDX200 ou µDX201), e ainda inserir alguns comentários sobre o projeto (notas).

Abrir Projeto...

Esta tecla abre um projeto já salvo anteriormente. Ao pressionar esta opção surge uma janela para seleção do projeto a ser lido:



Salvar Projeto

Esta tecla salva no disco rígido do microcomputador o projeto corrente. Isso inclui o projeto propriamente dito (arquivo sufixo .dxp) e todas as páginas que compõem o projeto (arquivos sufixo .dxg).

Salvar Projeto como...

Esta tecla salva no disco rígido do microcomputador o projeto corrente, permitindo especificar um novo nome para o arquivo de projeto (arquivo sufixo .dxp). São salvas também todas as páginas que compõem o projeto (arquivos sufixo .dxg).

Fechar Projeto

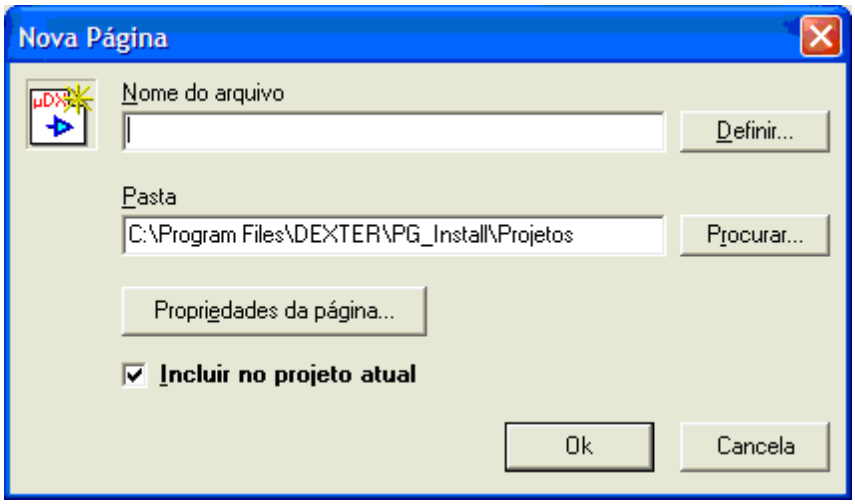
Permite encerrar o projeto corrente. Caso o projeto tenha sofrido modificações e não tenha sido salvo o PG irá alertar o usuário, de forma a evitar perda de informações.

Salvar Tudo

Salva o projeto e todas as páginas.

Nova Página (Ctrl+N)

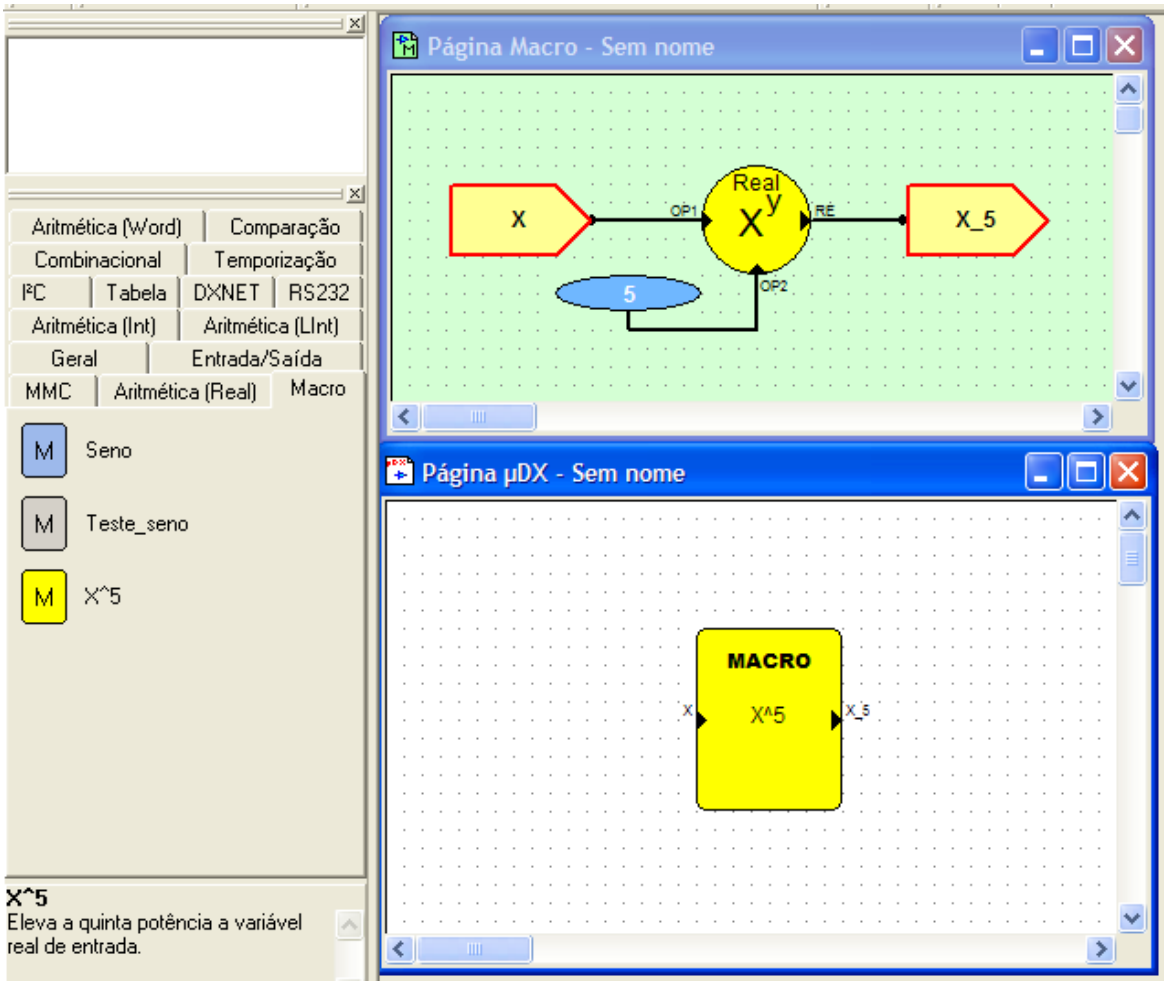
Abre uma nova página de programação na área de programação. Note que se houver um projeto aberto surgirá a tela abaixo:



Esta tela permite escolher um nome para a página, editar propriedades da página, e incluir a página no projeto corrente. Caso não exista projeto aberto a página é criada imediatamente, sem que surja a janela acima.

Nova Macro

Abre uma nova página de Macro na área de programação. Macros são programas aplicativos que serão compilados como um bloco único, permitindo sua utilização em outros programas. Note que a Macro se torna uma "caixa-preta", ou seja, nenhuma das variáveis e nodos internos estão disponíveis externamente, exceto se ligados às conexões de Macro (Entrada de Macro e Saída de Macro). A tela de programação de Macro (Página Macro) possui fundo verde, para diferenciar de tela de programação do µDX (Página µDX), que apresenta fundo branco. Note que é possível usar uma Macro dentro da definição de outra Macro. Com isso, é possível hierarquizar e organizar um programa, com Macros de crescente complexidade. Obviamente a Macro, apesar de visualmente se apresentar como apenas um bloco, ocupa na memória do controlador programável o número de blocos usados para sua criação. Abaixo um exemplo de Macro que eleva a quinta potência uma variável real. Veja que a Macro foi gerada na tela verde, e logo após ser compilada (Macro → Compilar e Salvar Macro) ela surge na Biblioteca de Componentes. Abaixo da janela de programação de Macro temos uma janela de programação µDX em que colocamos um bloco de Macro para elevar a quinta potência. Ou seja, criamos um bloco para que o µDX200 eleve um valor real à quinta potência.

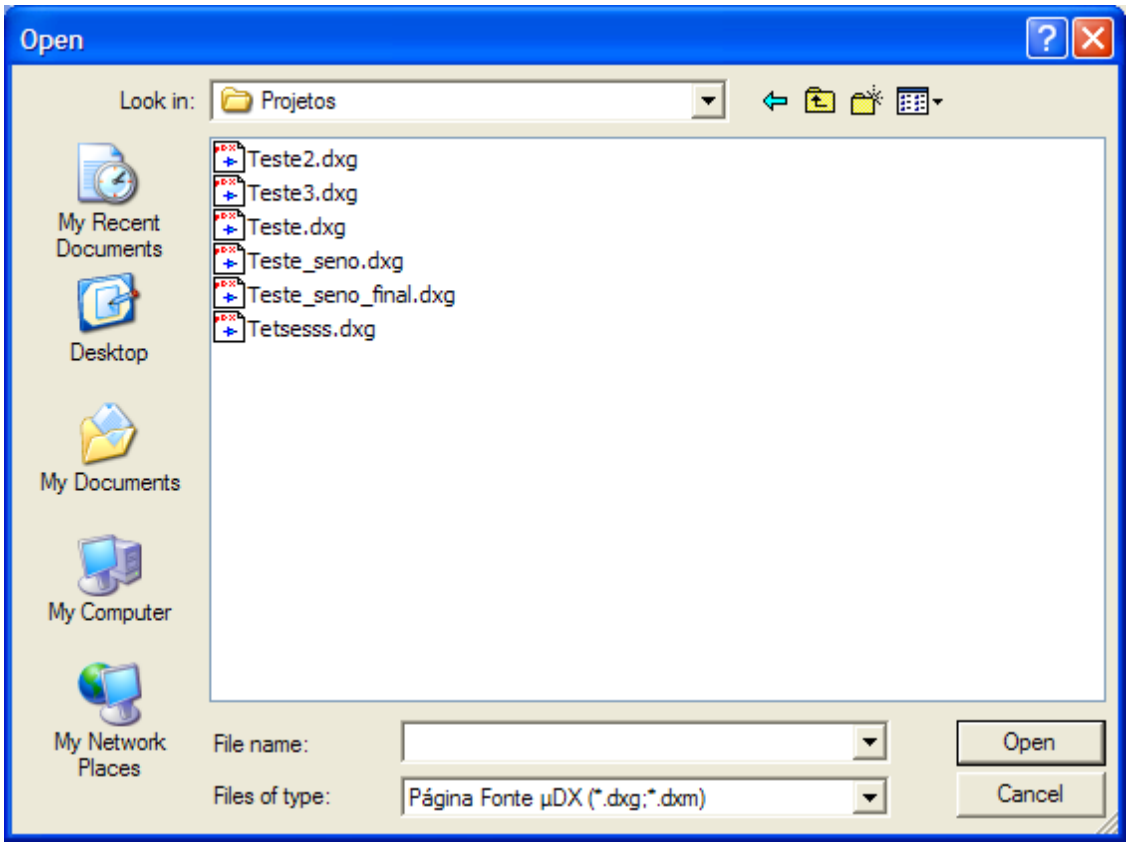


Exemplo de Macro: **X^5.dxm** (fonte); **X^5.dmb** (compilada).

Para salvar o programa que gerou uma Macro é necessário selecionar Salvar ou Salvar como... (veja as opções a seguir). Uma última observação: a função Nova Macro fica indisponível caso haja um projeto aberto no Editor PG. Para disponibilizar a função é necessário fechar o projeto.

Abrir... (Ctrl+A)

Abre uma página de programação previamente salva na área de programação. Note que, neste caso, a página aberta não é incluída no projeto (no caso de existir um projeto aberto). A janela de seleção da página tem o seguinte aspecto:



É possível também carregar uma página de programação de Macro. Para isso basta selecionar um arquivo fonte de Macro (sufixo .dxm).

Salvar (Ctrl+S)

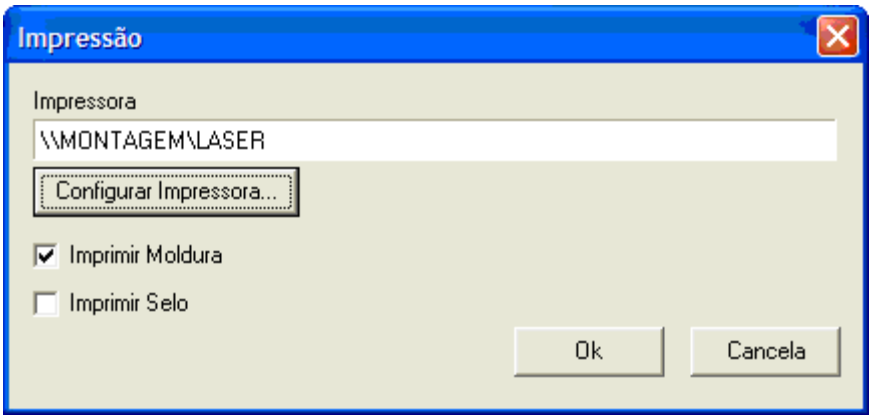
Salva no disco rígido do microcomputador a página µDX corrente (arquivo sufixo .dxg). Também permite salvar a página de Macro corrente (arquivo sufixo .dxm).

Salvar como...

Esta tecla salva no disco rígido do microcomputador a página µDX corrente, permitindo especificar um nome para o arquivo (arquivo sufixo .dxg). Também permite salvar a página de Macro corrente (arquivo sufixo .dxm).

Imprimir...

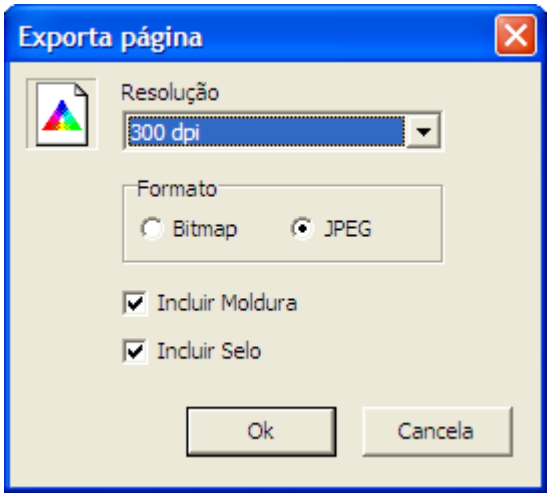
Imprime a página de programação corrente. Ao selecionar esta opção surge a tela:



É possível escolher a impressora (caso exista mais de uma impressora configurada no computador), configurá-la, e selecionar se devem ser impressas a moldura da página e o selo (quadro com informações no canto inferior direito da página).

Exportar...

Muitas vezes é necessário anexar a documentação de determinado projeto detalhes da programação implementada no controlador programável. Esta opção permite gerar um arquivo bit-map, compactado (JPG) ou não (BMP), com diversas resoluções. Normalmente uma resolução de 300 dpi (dots per inch) é suficiente, mas ela pode ser escolhida desde 150 dpi até 600 dpi. Como os arquivos JPG são cerca de 10 a 15 vezes menores que seus equivalentes em BMP aconselhamos seu uso sempre que possível:



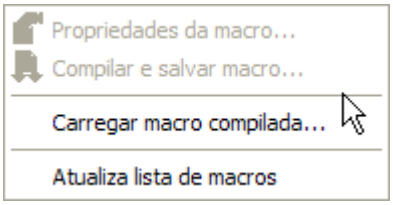
Note que para páginas de programação grandes a geração do arquivo BMP ou JPG pode levar vários segundos. Durante este período a janela acima ficará "congelada" no PG. Isso é normal. Aguarde a finalização do processo.

Sair

Permite encerrar o programa Editor PG.

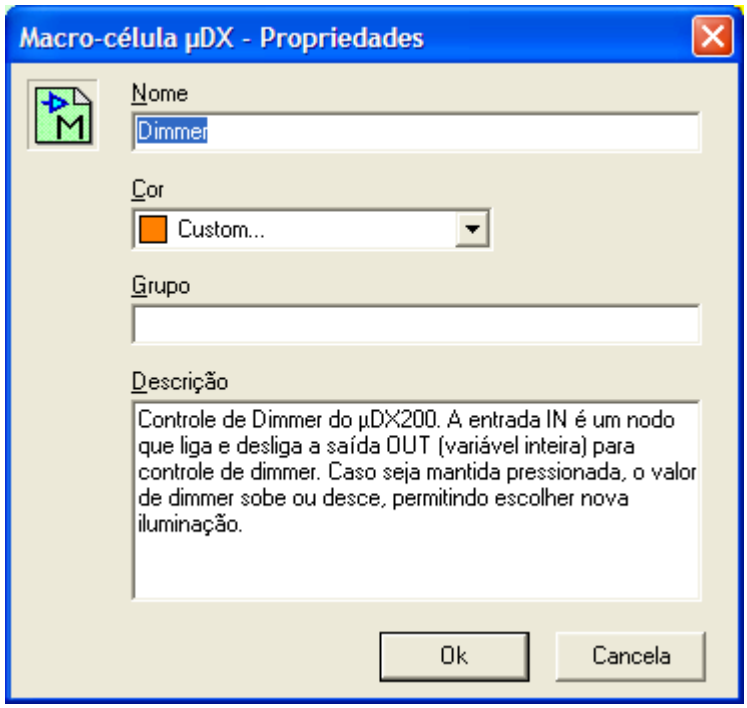
Menu Macro

Este menu permite editar propriedades de uma Macro, e também compilá-la, gerando um bloco utilizável nos programas aplicativos do µDX201. Além disso, é possível carregar macros compiladas por terceiros (arquivos sufixo .dmb) e atualizar a lista de Macros na Biblioteca de Componentes. As funções que não estão disponíveis aparecem em cinza.



Propriedades da Macro...

Esta tecla edita as propriedades de uma Macro. Ao selecionar esta função surge a seguinte tela (é necessário estar com uma página de Macro selecionada):



O campo **Nome** designa qual o nome da Macro. Este nome é que irá aparecer na Biblioteca de Componentes, assim como na própria representação como um bloco da Macro. Em **Cor** é possível especificar uma cor de fundo para a representação da Macro. Com isso, pode-se eleger determinadas cores para funções específicas (como é feito com os blocos do µDX201), facilitando a identificação das Macros. O campo **Grupo** permite agrupar as Macros na lista de Macros conforme sua funcionalidade. Caso seja mantido em branco as Macros não são agrupadas. Em **Descrição** pode-se incluir uma descrição sucinta da função da Macro.

Compilar e Salvar Macro...

Esta função compila a Macro, gerando um arquivo com sufixo .dmb. Note que o arquivo fonte de Macro possui sufixo .dxm. Ao compilar a Macro ela já é incluída na Biblioteca de Componentes (aba Macro). Note que esta opção é inibida no caso de existir um projeto aberto no PG. É necessário fechar o projeto antes de compilar Macro.

Carregar Macro Compilada...

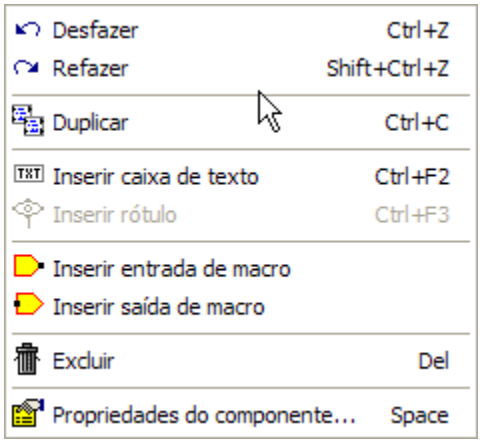
Pode-se carregar um arquivo .dmb de Macro compilada para a Biblioteca de Componentes do Editor PG. Com isso, é possível utilizar macros geradas por terceiros.

Atualiza Lista de Macros

Permite atualizar a lista de Macros existente na Biblioteca de Componentes do Editor PG.

Menu Editar

Este menu permite desfazer ou refazer operações, duplicar blocos ou áreas de programa, inserir caixas de texto ou rótulos, inserir entradas e saídas de Macros, excluir blocos ou áreas de programa, ou ainda visualizar propriedades de um componente selecionado. Note que todas funções estão disponíveis também na Barra de Ferramentas (com o mesmo ícone apresentado à esquerda), e também possuem teclas de atalho (representadas à direita). As funções que não estão disponíveis aparecem em cinza.



Desfazer (Ctrl+Z)

Desfaz a última operação efetuada no Editor PG. A tecla de Desfazer possui uma profundidade de cinco operações, ou seja, pode-se desfazer até as cinco últimas operações.

Refazer (Shift+Ctrl+Z)

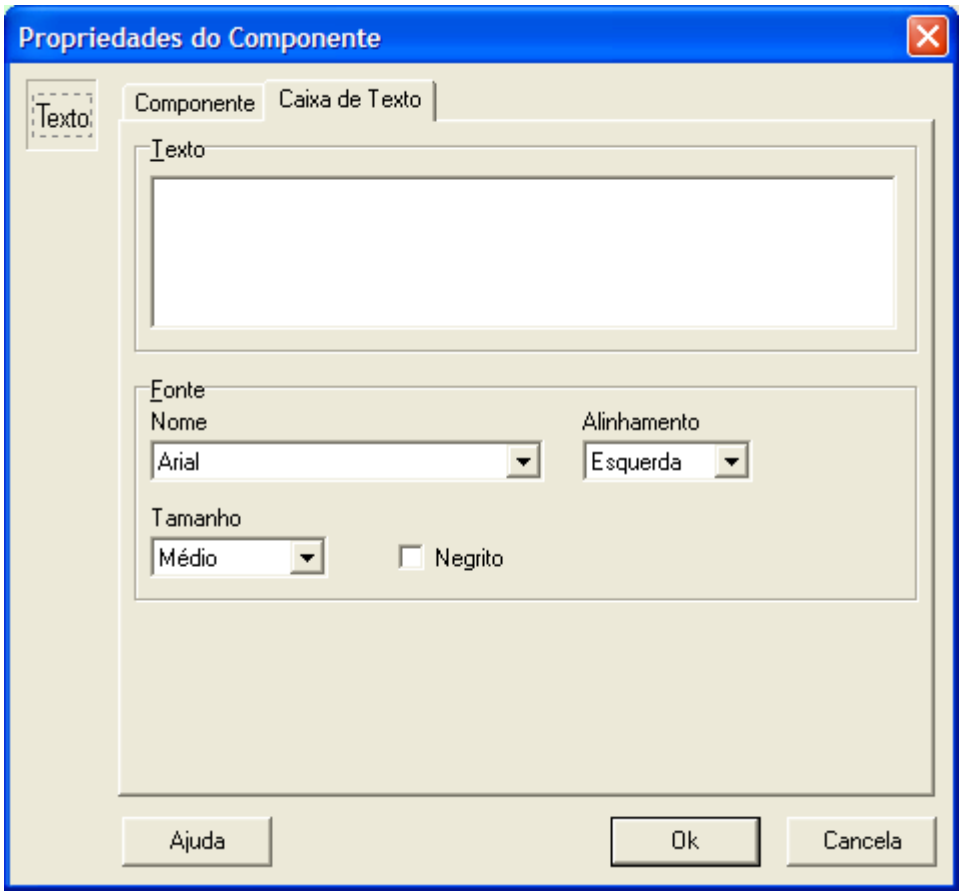
Refaz a última operação efetuada no Editor PG. A tecla de Refazer possui uma profundidade de cinco operações, ou seja, pode-se refazer até as cinco últimas operações

Duplicar (Ctrl+C)

Esta tecla permite duplicar um componente ou uma área selecionada do programa. Para selecionar um componente basta clicar sobre o componente com a tecla esquerda do mouse. Já para selecionar uma área mantenha pressionada a tecla esquerda do mouse e selecione um retângulo de seleção movimentando o mouse. Ao pressionar Duplicar os componentes presentes na área selecionada são duplicados, e podem ser incluídos tanto na própria página de programação que os originou quanto em outra página.

Inserir Caixa de Texto (Ctrl+F2)

Insere uma caixa de texto no programa. Esta caixa de texto não é utilizada pelo programa, servindo exclusivamente como comentário para o programa aplicativo do µDX200. Para "largar" a caixa de texto sobre o programa basta clicar com a tecla esquerda do mouse sobre o local desejado. Para editar a caixa de texto, permitindo inserir o texto, clique com a tecla direita do mouse apontando para a caixa de texto. Deve surgir uma janela que permite, além da digitação do texto, a escolha da fonte de caracteres a ser usada, o tamanho dos caracteres, e se em negrito ou não, além do alinhamento do texto.

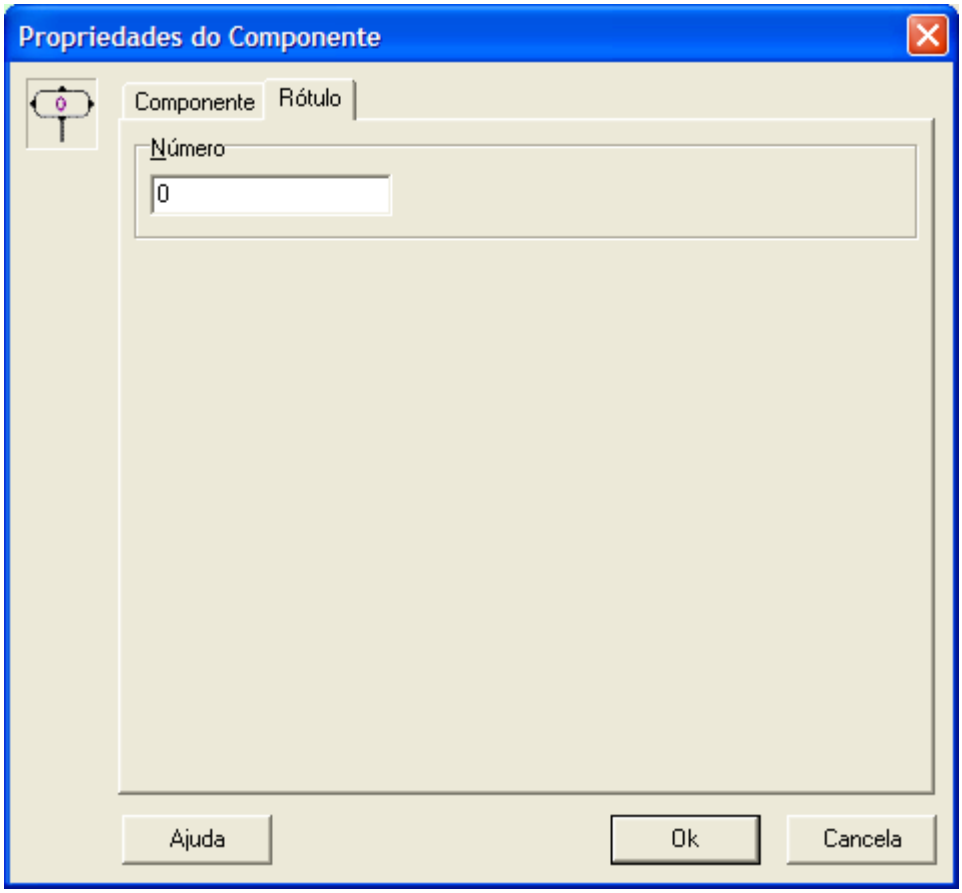


Para ajustar o tamanho da caixa de texto (de forma que o texto digitado caiba na caixa) basta selecionar a caixa e arrastar os cantos dessa, modificando suas dimensões.

Inserir Rótulo (Ctrl+F3)

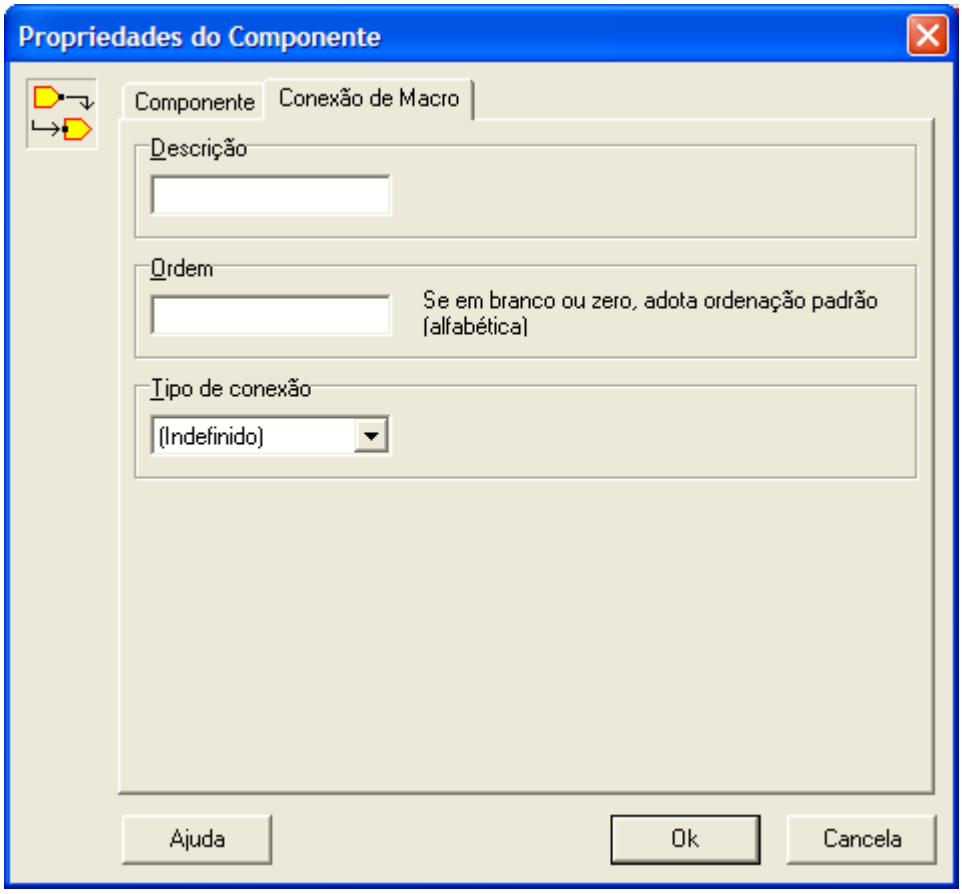
Insere um rótulo no programa. A função do rótulo é conectar diferentes pontos do programa entre si. O rótulo apenas conecta entre si todos os pontos com o mesmo número de rótulo (de 0 a 999), como se estivessem conectados por "fios", não atribuindo nenhum valor determinado ou nome para esta conexão. O rótulo pode conectar qualquer tipo de variável do µDX200 (nodo, inteiro, longint ou word). Ao editar o Rótulo surge a janela que permite atribuir um valor numérico ao mesmo. Todos os rótulos com mesmo valor numérico estarão conectados (mesmo em janelas de programação distintas de um mesmo projeto).

Note que rótulo não é permitido em Macros. Desta forma, esta função estará inibida quando estiver selecionada uma página de Macro.



Inserir Entrada de Macro

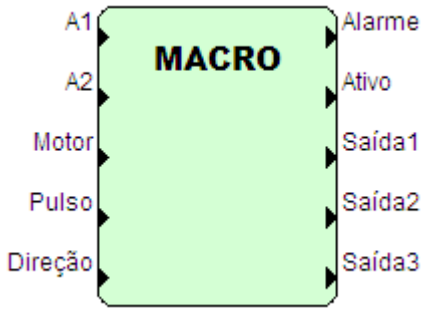
Insere um ponto de entrada para a Macro. Esta entrada pode tanto ser um nodo (variável binária), quanto uma variável (inteira, word, longint, real). Esta função somente fica ativa quando está selecionada uma página de Macro, já que não é possível usar **Entrada de Macro** em programas aplicativos do µDX200. Ao editar uma **Entrada de Macro** ou uma **Saída de Macro** surge a janela abaixo, que permite especificar um nome para a conexão da Macro, e escolher uma ordem para a colocação das conexões da Macro.



A **Ordem** em uma Entrada de Macro (ou Saída de Macro) especifica qual a ordem em que as conexões de entrada (ou saída) aparecerão no bloco de Macro. As entradas sempre são colocadas à esquerda do bloco de Macro, e as saídas à direita. Com isso, entradas e saídas são ordenadas independentemente. Caso o campo Ordem seja mantido em branco a ordenação será em ordem alfabética crescente. Caso seja colocado números no campo Ordem, os itens serão ordenados em ordem crescente destes números. É aceitável qualquer número inteiro (portanto, é possível usar-se números negativos). Se duas os mais entradas de Macro possuírem o mesmo número de ordem, elas serão ordenadas alfabeticamente. Por exemplo, digamos que foi gerada uma Macro com as seguintes conexões de entrada e saída:

Entradas de Macro		Saídas de Macro	
Descrição	**Ordem**	**Descrição**	**Ordem**
A1	1	Saída1	3
A2	1	Saída2	3
Motor	2	Saída3	3
Pulso	3	Alarme	1
Direção	4	Ativo	2

Neste caso o PG irá gerar um bloco de Macro como mostrado a seguir:



O **Tipo de Conexão** permite especificar qual tipo de dado é esperado para a conexão da Macro. No caso de **(Indefinido)** é aceito qualquer tipo de dado, sem que sejam geradas mensagens de alerta na compilação. Já nas outras opções é gerada uma mensagem de alerta caso seja conectado outro tipo de variável, diferente da especificada neste campo. Isso permite discernir o tipo de dado a ser conectado à Macro (Lógico, Byte, Inteiro, Word, Real, etc).

Ao clicar com botão direito sobre a Macro (Propriedades da Macro) surge uma lista de todas as conexões e seu tipo na aba **Conexões**. Como esta especificação de **Tipo de Conexão** foi implementada apenas na versão 2.2.0.16 do software PG as Macros que acompanham o PG em versões anteriores apresentam todas as conexões como indefinidas.

Inserir Saída de Macro

Insere um ponto de saída para a Macro. Esta saída pode tanto ser um nodo (variável binária), quanto uma variável (inteira, word, longint, real). Esta função somente fica ativa quando está selecionada uma página de Macro, já que não é possível usar **Saída de Macro** em programas aplicativos do µDX200.

Excluir (Del)

Exclui o componente ou a área selecionada do programa. Para selecionar um componente basta clicar sobre o componente com a tecla esquerda do mouse. Já para selecionar uma área mantenha pressionada a tecla esquerda do mouse e selecione um retângulo de seleção movimentando o mouse.

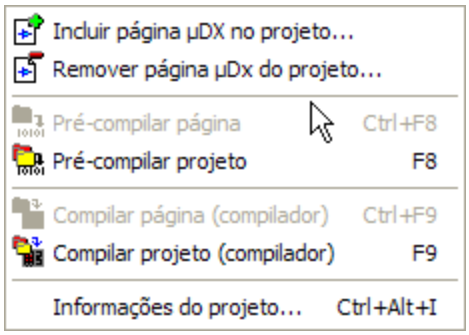
Propriedades do Componente (Space)

Apresenta as propriedades do componente selecionado. Note que as propriedades do componente também podem ser acessadas clicando sobre o mesmo com a tecla direita do mouse, exceto no caso das conexões ("fios"). A tecla direita, neste caso, gera uma conexão derivada da conexão existente. Já a tecla de Space (barra de espaço do teclado) sempre acessa

as propriedades do componente.

Menu Projeto

Este menu permite incluir ou remover páginas do projeto, compilar página de programa ou o projeto inteiro (note que o Editor PG, na verdade, efetua uma pré-compilação), compilar o programa pré-compilado e abrir a janela do Compilador, ou ainda visualizar informações sobre o projeto. Note que todas funções estão disponíveis também na Barra de Ferramentas, à exceção de Informações do projeto (com o mesmo ícone apresentado à esquerda), e também muitas delas possuem teclas de atalho (representadas à direita). As funções que não estão disponíveis aparecem em cinza.



Incluir página µDX no projeto...

Este item possibilita inserir páginas de programação µDX ao projeto corrente.

Remover página µDX no projeto...

Este item possibilita remover páginas de programação µDX ao projeto corrente.

Pré-compilar página (Ctrl+F8)

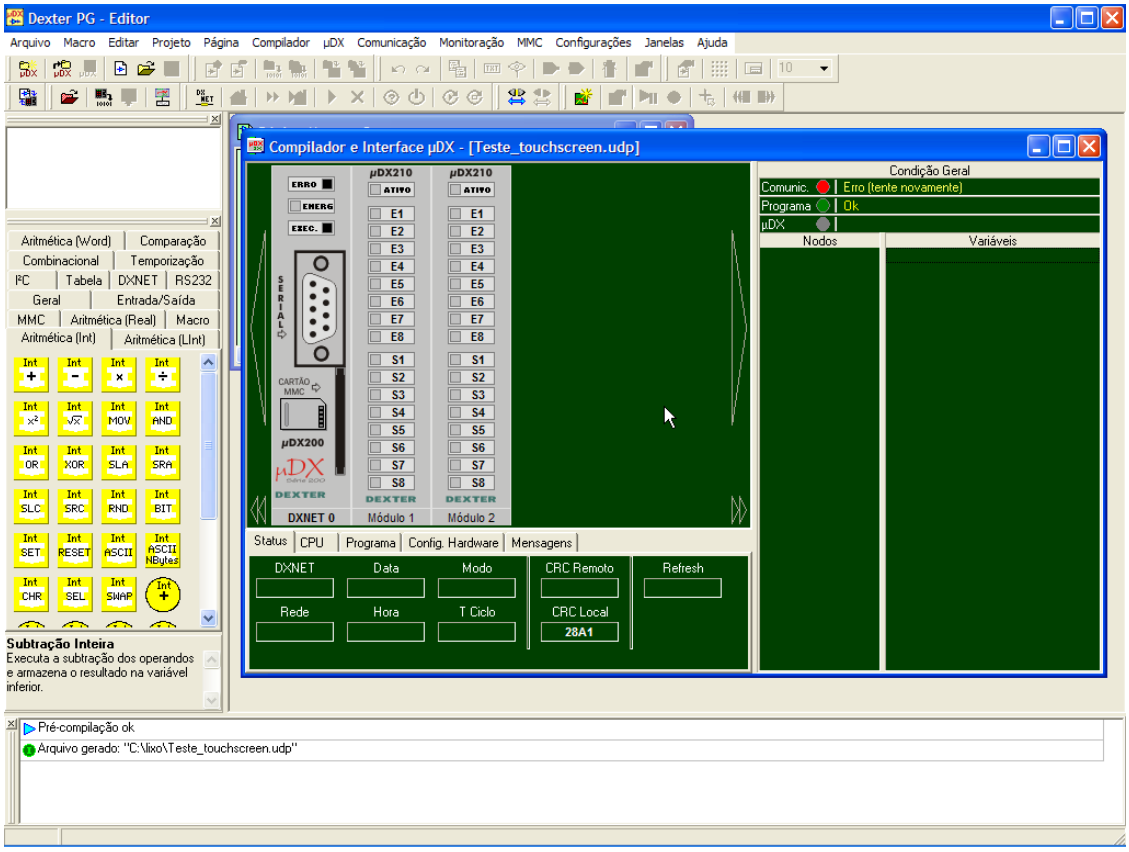
Efetua a pré-compilação apenas da página selecionada, gerando um arquivo de pré-compilação (arquivo sufixo .udp).

Pré-compilar projeto (F8)

Pré-compila todas as páginas do projeto, gerando um arquivo de pré-compilação (arquivo sufixo .udp) que inclui todas as páginas de programação µDX inclusas no projeto. As páginas podem compartilhar dados (basta que referências à mesma variável sejam feitas em diferentes páginas) ou conexões (usando rótulos com mesmo número em diferentes páginas).

Compilar Página (Compilador) (Ctrl+F9)

O Editor PG gera uma pré-compilação (arquivo sufixo .udp). Este arquivo deve ser lido no Compilador PG (janela que inclui monitoramento do µDX200 e ferramentas para sua programação) para que seja efetuada a compilação final do programa aplicativo (que será transmitida para o controlador). Para isso é necessário abrir a janela do Compilador, e isso pode ser feito a partir do Editor PG via esta opção. Ao clicar nesta função é feita a pré-compilação da página de programação µDX (arquivo sufixo .dxg gera arquivo sufixo .udp), a compilação propriamente dita, e é aberta a janela do Compilador e Interface µDX. Note que no cabeçalho desta janela aparece o nome do arquivo pré-compilado carregado:

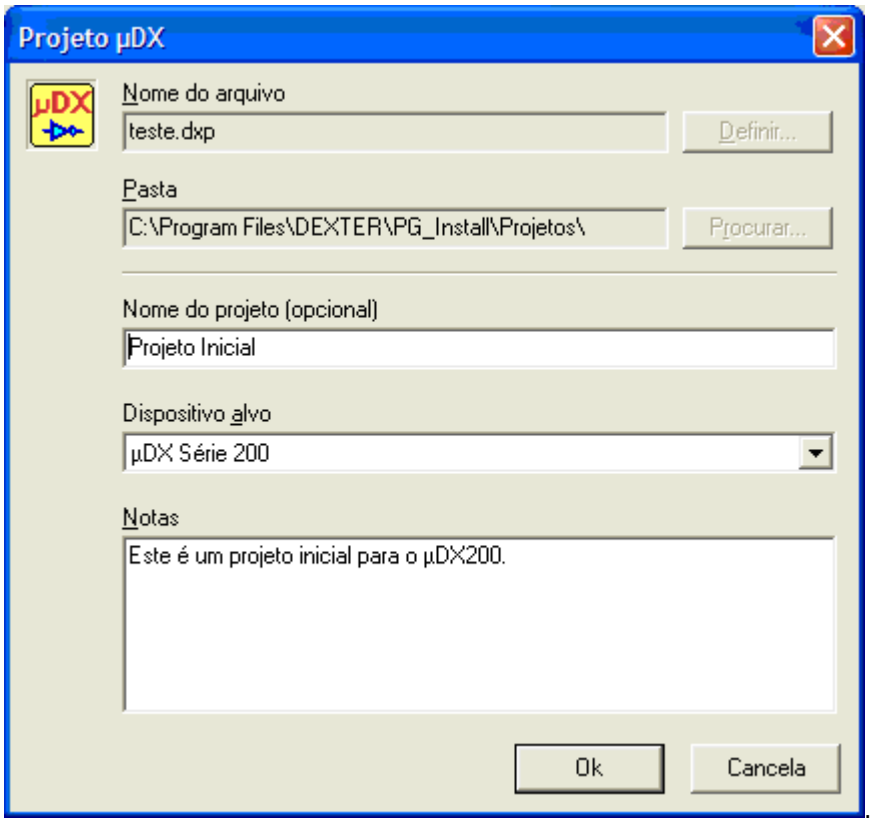


Compilar Projeto (Compilador) (F9)

O Editor PG gera uma pré-compilação (arquivo sufixo .udp). Este arquivo deve ser lido no Compilador PG (janela que inclui monitoramento do µDX200 e ferramentas para sua programação) para que seja efetuada a compilação final do programa aplicativo (que será transmitida para o controlador). Para isso é necessário abrir a janela do Compilador, e isso pode ser feito a partir do Editor PG via esta opção. Ao clicar nesta função é feita a pré-compilação do projeto de programação µDX (arquivo sufixo .dxp gera arquivo sufixo .udp), a compilação propriamente dita, e é aberta a janela do Compilador e Interface µDX. Note que no cabeçalho desta janela aparece o nome do arquivo pré-compilado carregado, como mostrado na figura anterior.

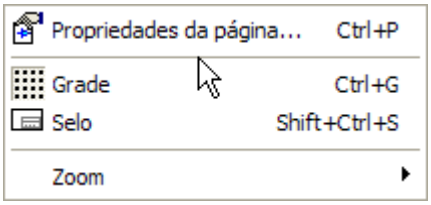
Informações do projeto... (Ctrl+Alt+I)

Retorna uma janela com informações sobre o projeto aberto no Editor PG, e permite editar estas informações.



Menu Página

Este menu permite editar propriedades da página, acionar ou inibir a grade na página, ativar ou desativar o selo da página (quadro no canto inferior direito com informações sobre a página de programação), e também selecionar o zoom de visualização. Note que todas funções estão disponíveis também na Barra de Ferramentas (com o mesmo ícone apresentado à esquerda), e também possuem teclas de atalho (representadas à direita). As funções que não estão disponíveis aparecem em cinza.



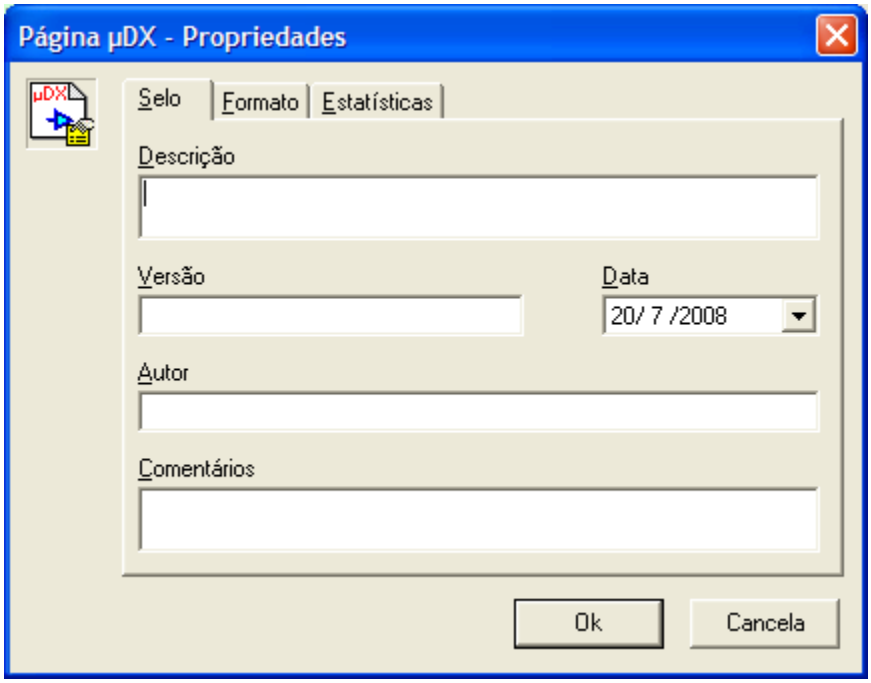
Propriedades da página... (Ctrl+P)

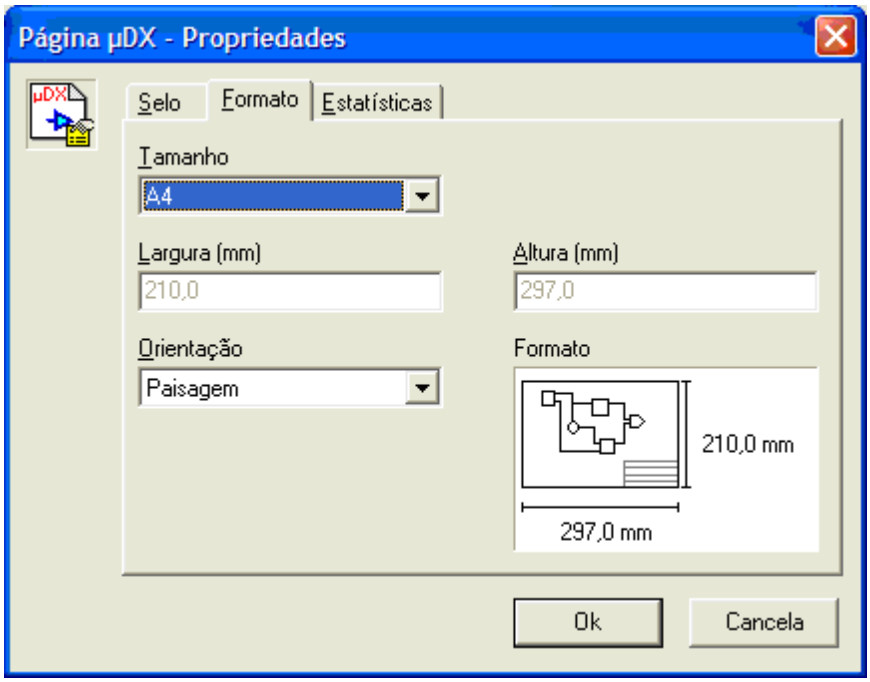
Retorna uma janela com propriedades da página selecionada.

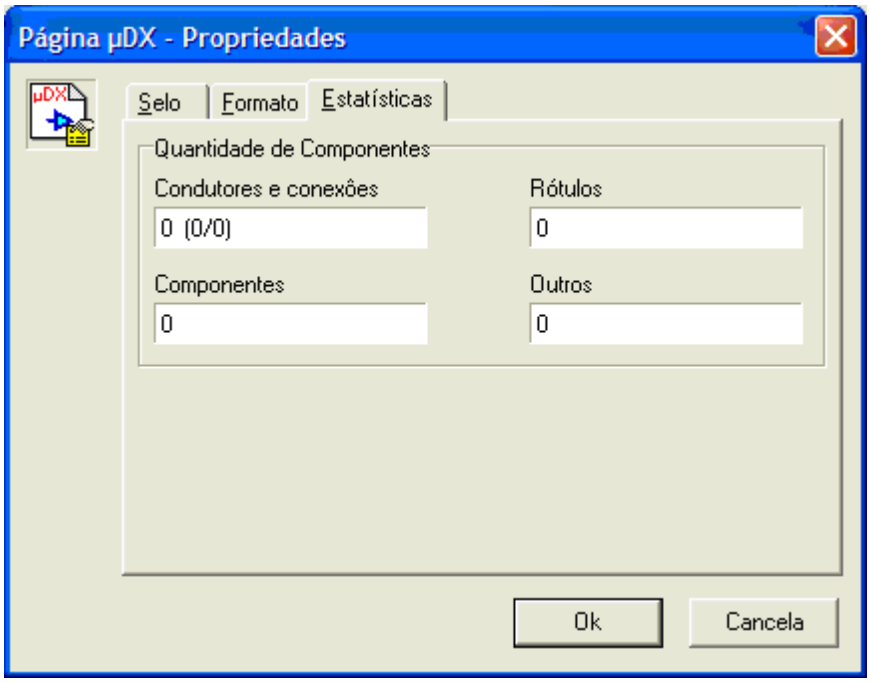
Na aba **Selo** pode-se inserir os dados a serem visualizados no Selo da página (quadro no canto inferior direito com informações sobre a página de programação).

Em **Formato** define-se o tamanho da página e sua orientação.

Por fim, em **Estatísticas** são apresentados alguns dados a respeito da página de programação, como o número de condutores, rótulos, etc., existentes na página.

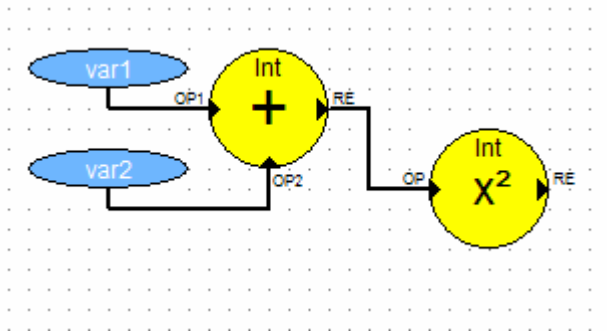



Página µDX - Propriedades
Selo
Formato
Estatísticas
Tamanho
A4
Largura (mm)
210,0
Altura (mm)
297,0
Orientação
Paisagem
Formato
210,0 mm
297,0 mm
Ok
Cancela


Página µDX - Propriedades
Selo
Formato
Estatísticas
Quantidade de Componentes
Condutores e conexões
0 (0/0)
Rótulos
0
Componentes
0
Outros
0
Ok
Cancela

Grade (Ctrl+G)

O posicionamento dos componentes no Editor PG obedecem a um alinhamento em relação a um quadriculado, que pode ser visível ou não, conforme a seleção deste item.



Selo (Shift+Ctrl+S)

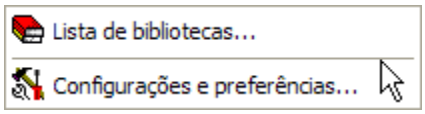
Ativa ou desativa a visualização de selo na página selecionada (página de programação µDX ou página de Macro).

Zoom

Permite selecionar entre sete níveis de ampliação (zoom): 2, 3, 5, 7, 10, 15 ou 20. O nível de zoom pode ser selecionado individualmente para cada página.

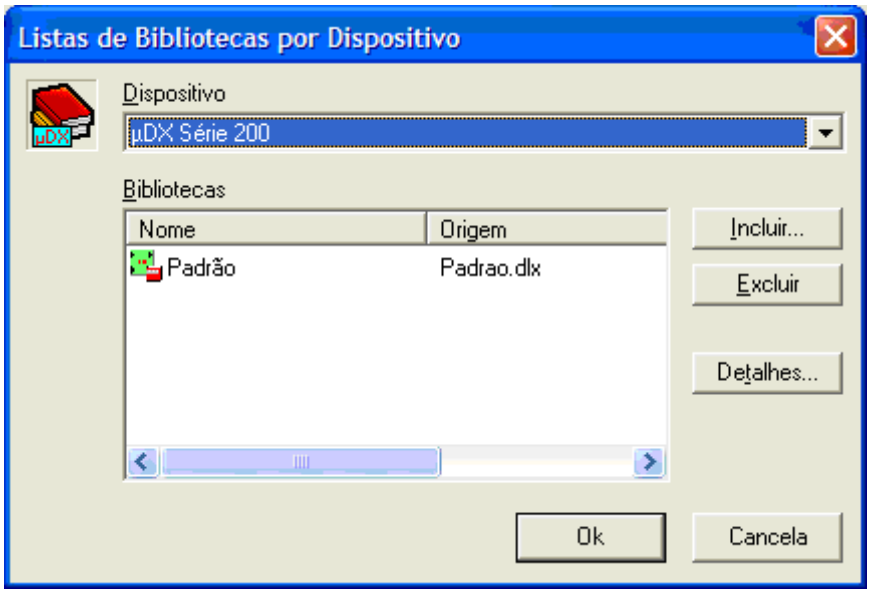
Menu Configurações

Este menu permite escolher as bibliotecas de blocos a serem utilizadas pelo Editor PG, e uma série de configurações para o software.



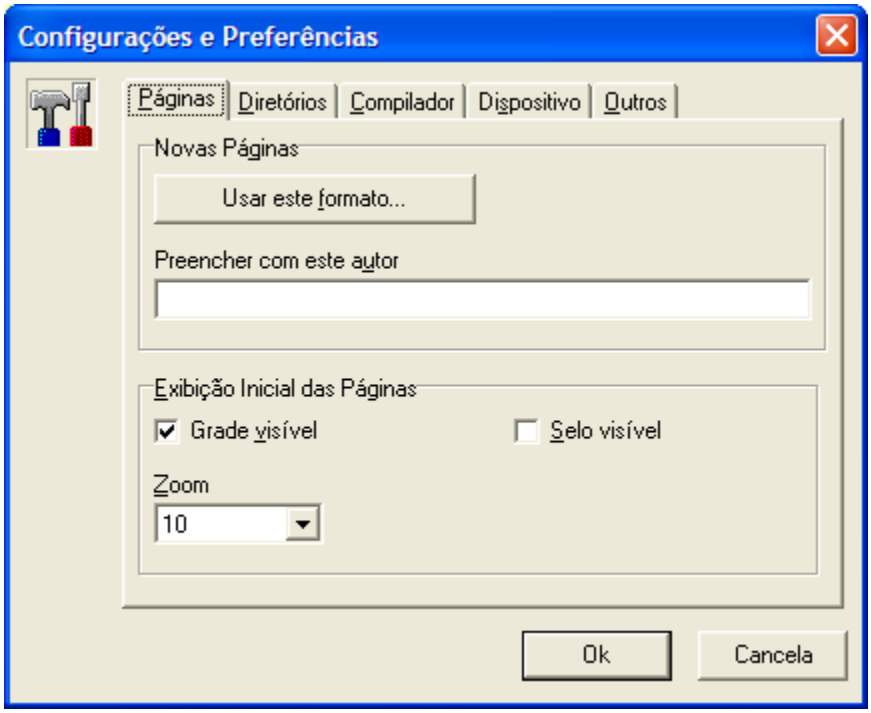
Lista de bibliotecas...

Inclui ou exclui bibliotecas de blocos para o Editor PG. As bibliotecas são arquivos com sufixo .dlx. Acompanha o PG a biblioteca padrão para µDX200 (Padrao.dlx), e a biblioteca padrão para µDX201 (Padrao_201.dlx).

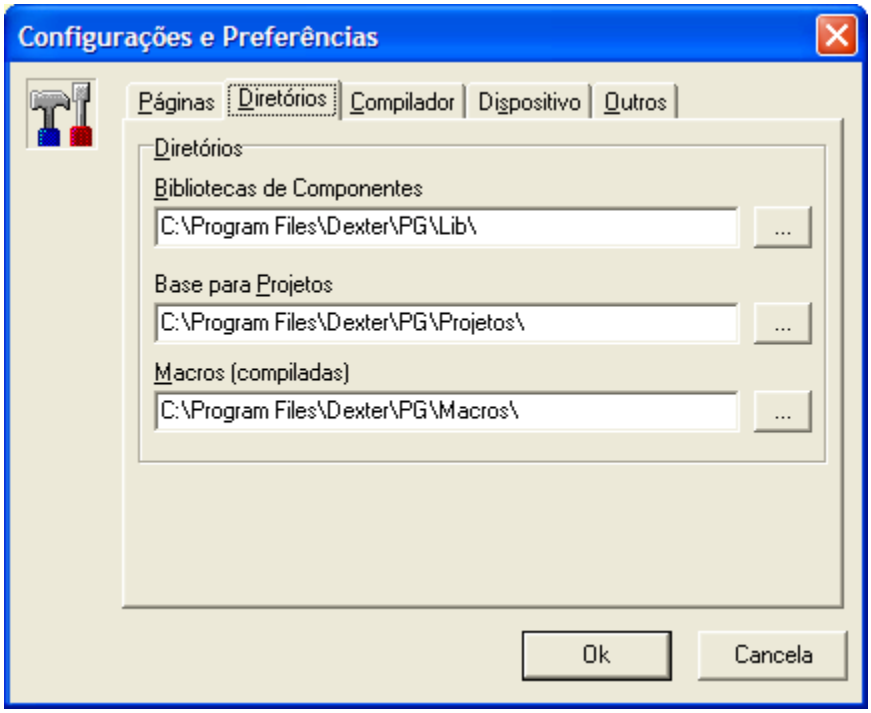


Configurações e preferências...

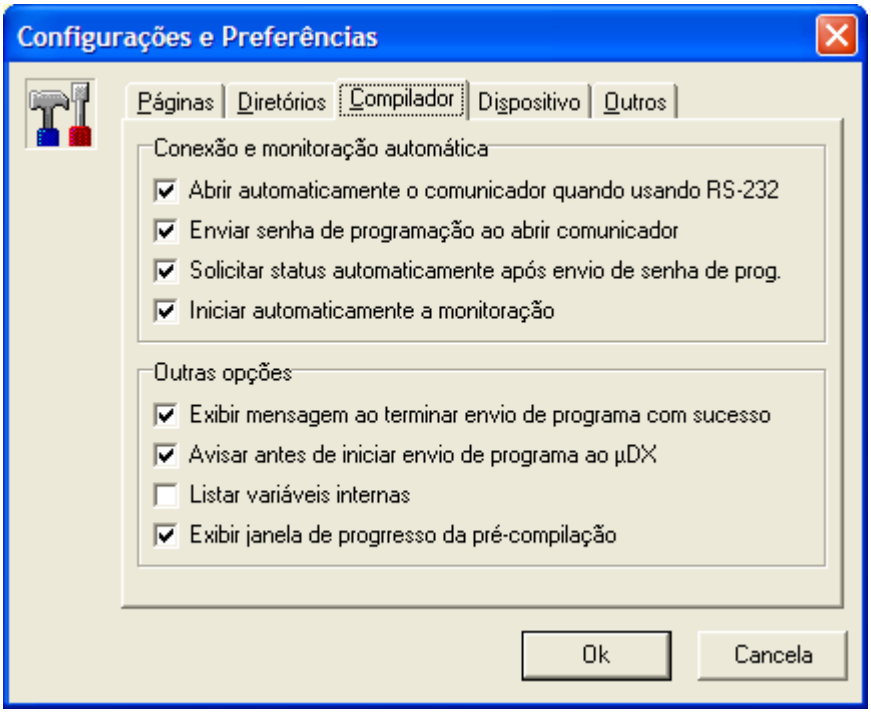
A aba de **Páginas** permite fixar vários parâmetros, como tamanho da página, autor, grade, selo e zoom, que serão usados sempre que forem criadas novas páginas de programação no software PG.



Já a aba de **Diretórios** indica o diretório padrão para as bibliotecas, macros e projetos do Editor PG.

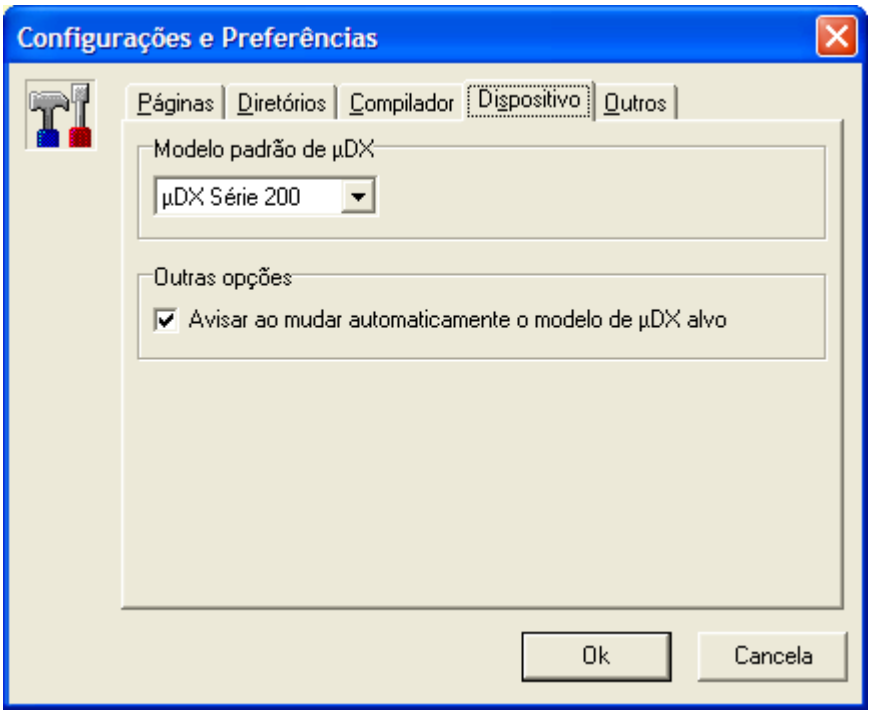


A aba **Compilador** possui diversas opções referentes a janela de compilação:

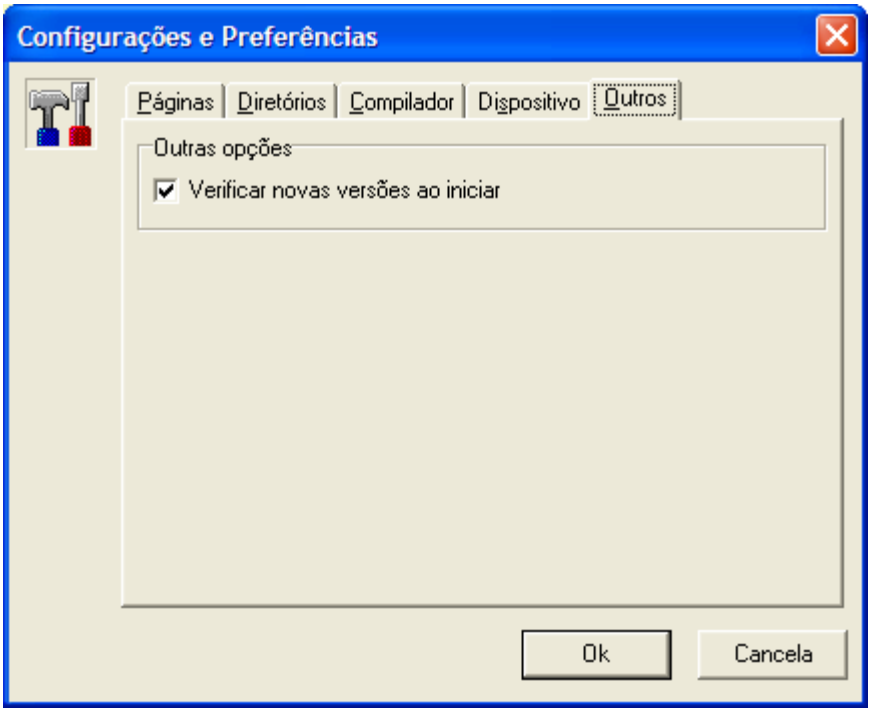


Convém manter marcadas as primeiras quatro opções, de forma que o PG inicie monitoramento automaticamente ao abrir comunicador.

A aba **Dispositivo** permite selecionar qual o dispositivo padrão ao iniciar o software PG. Atualmente é possível escolher entre **µDX200** e **µDX201**:

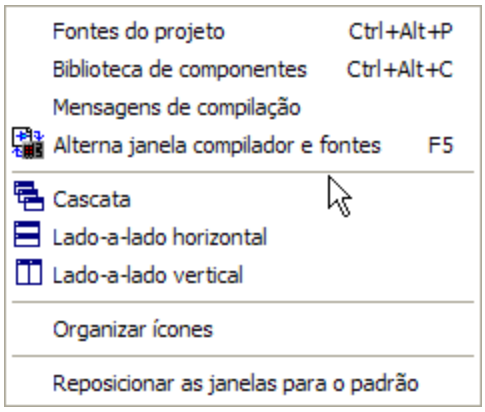


Por fim, a aba **Outros** possui, no momento, apenas a opção de permitir ao software PG verificar no site da Dexter se está disponível uma versão mais atualizada do software. Esta verificação ocorre apenas ao iniciar o software PG, ou selecionando esta verificação no menu Ajuda.



Menu Janelas

Este menu permite migrar entre as várias janelas do Editor PG (fontes, biblioteca, mensagens de compilação), e organizar as janelas de programação na área de programação. Além disso, uma última opção recoloca as janelas em uma posição padrão. As opções não disponíveis são apresentadas em cinza. Note que a opção de **Reposição das janelas para o padrão** não reativa as janelas que foram fechadas. Para isso clique na opção qua ativa a janela correspondente: **Fontes do projeto**, **Biblioteca de componentes**, ou **Mensagens de compilação**.



Fontes do projeto (Ctrl+Alt+P)

Exibe a janela de fontes do projeto, com todas as páginas de programação pertencentes ao projeto.

Biblioteca de componentes (Ctrl+Alt+C)

Exibe a janela de biblioteca de componentes do PG, com todas as famílias de blocos disponíveis.

Mensagens de compilação

Exibe a janela de mensagens de compilação do Editor PG. Note que se trata de uma pré-compilação, que indica variáveis de tipos diferentes interconectadas e outros erros no programa. A compilação propriamente dita ocorre no programa Compilador, que acompanha o pacote de software do controlador µDX200.

Alterna Janela Compilador e Fontes (F5)

Alterna entre janela de Compilador PG e janela com os programas aplicativos elaborados no Editor PG.

Cascata

Organiza as janelas de programação em cascata.

Lado-a-lado horizontal

Distribui as janelas de programação horizontalmente.

Lado-a-lado vertical

Distribui as janelas de programação verticalmente.

Organizar ícones

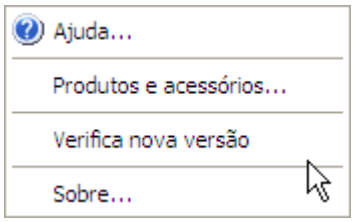
Realinha os ícones das janelas minimizadas no Editor PG.

Reposicionar as janelas para o padrão

Recoloca as janelas principais e flutuantes na configuração padrão do programa.

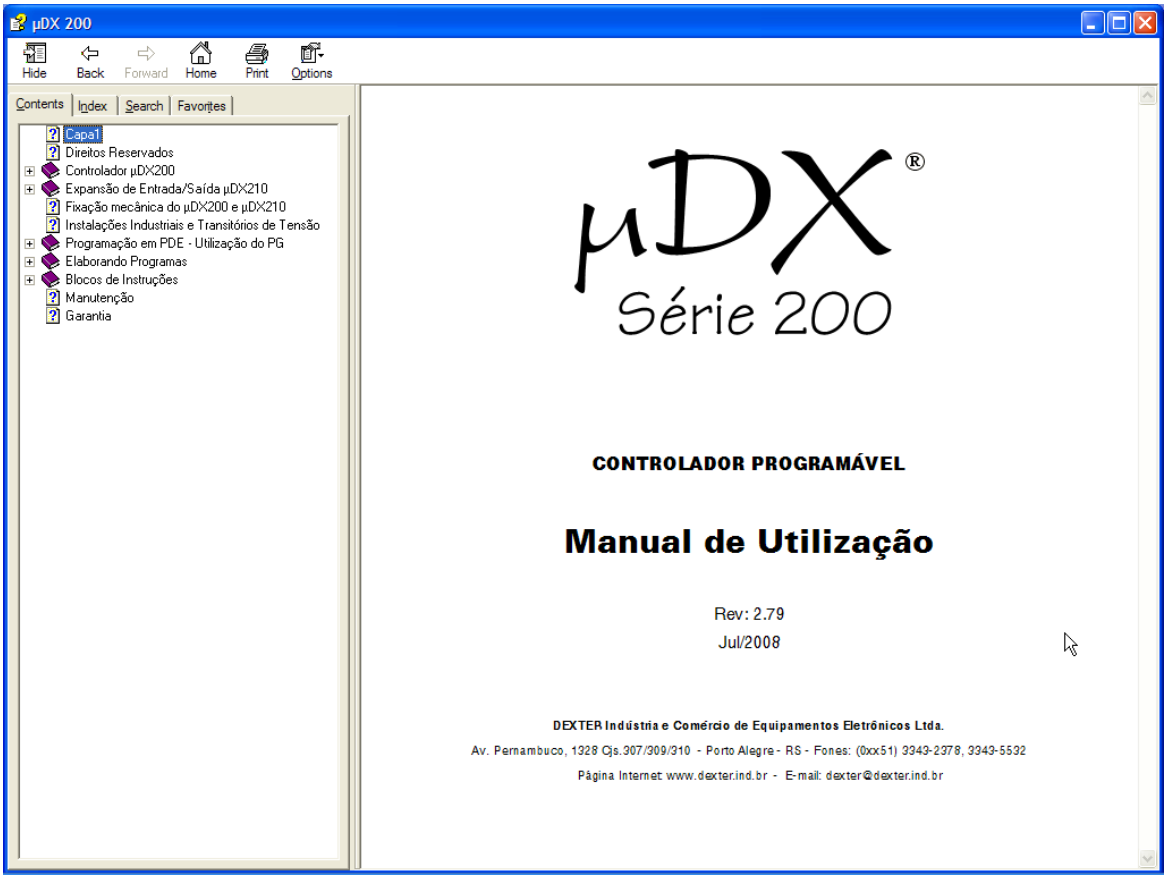
Menu Ajuda

Este menu permite o acesso ao manual (help) do Editor PG. Além disso, a tecla **Sobre...** informa versão do PG e algumas informações do software. Já a opção **Verifica nova versão** faz com quem o PG compare sua versão com a existente no site da Dexter - http://www.dexter.ind.br/pg1.htm - e informe se existe uma versão mais atual. A opção **Produtos e Acessórios...** abre um arquivo no formato PDF com descrição de todos os produtos Dexter para linha µDX200.



Ajuda...

Acessa o arquivo de ajuda (help) do Editor PG.

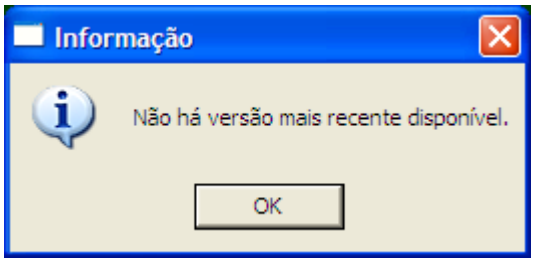


Produtos e Acessórios...

Abre catálogo em formato PDF com descrição de todos os produtos Dexter para linha µDX200.

Verifica nova versão

Verifica se existe uma versão mais atual do software PG no site da Dexter - http://www.dexter.ind.br/pg1.htm.



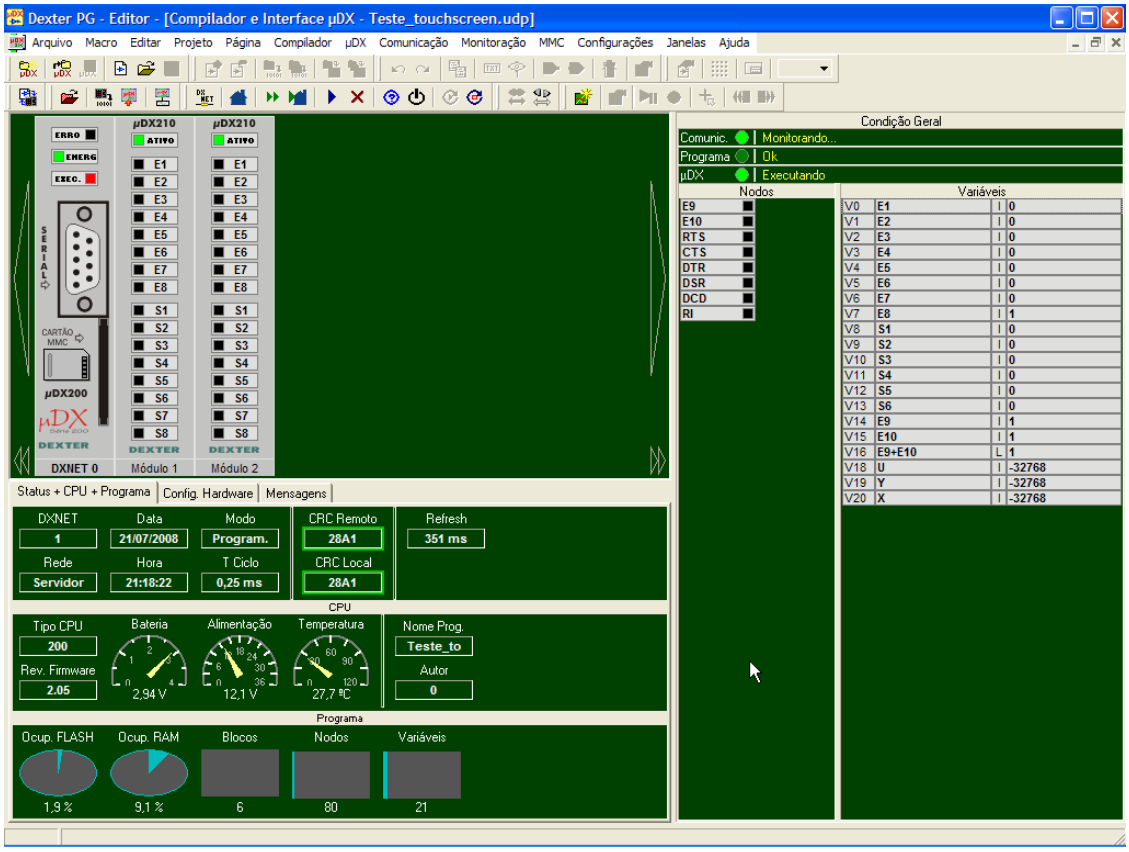
Sobre...

Informações sobre o software Editor PG.

Teclas de Operação do Compilador PG

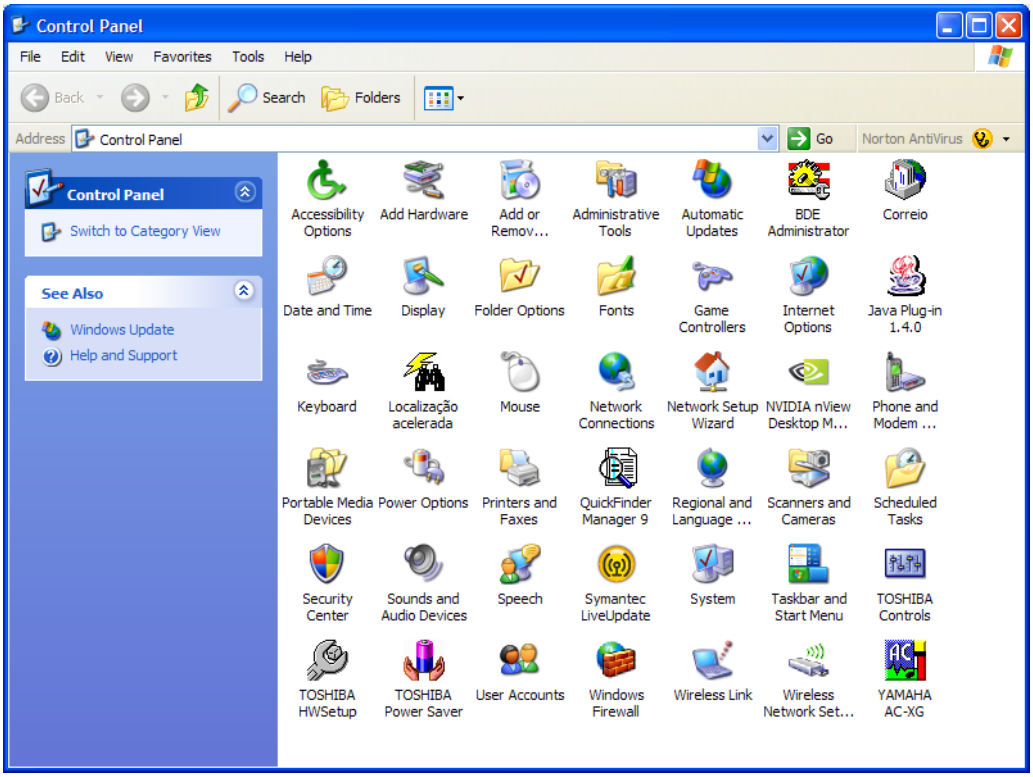
O Compilador PG (Programador Gráfico) não só permite compilar programas gerados no Editor PG, como monitorar status, nodos e variáveis do Controlador µDX200. A partir da versão 2.1.0.0 o ambiente de desenvolvimento é integrado, de forma que o Editor e o Compilador PG compartilham a mesma interface de software. Assim, embora sejam gerados na instalação dois atalhos na tela principal do computador, ambos rodam o mesmo programa. Apenas a diretiva em cada atalho é diferente, abrindo o PG em tela de Edição de Programas ou em tela de Compilação de Programas.

Atenção: Cuidado para não abrir duas vezes o PG inadvertidamente. Para passar do Editor para o Compilador e vice-versa use a tecla [Compilador/Fontes] (F5) existente no ambiente integrado. Caso seja chamado um atalho (por exemplo, µDX PG Editor) e depois o outro (µDX PG - Compilador) serão executados dois softwares PG simultaneamente no computador.

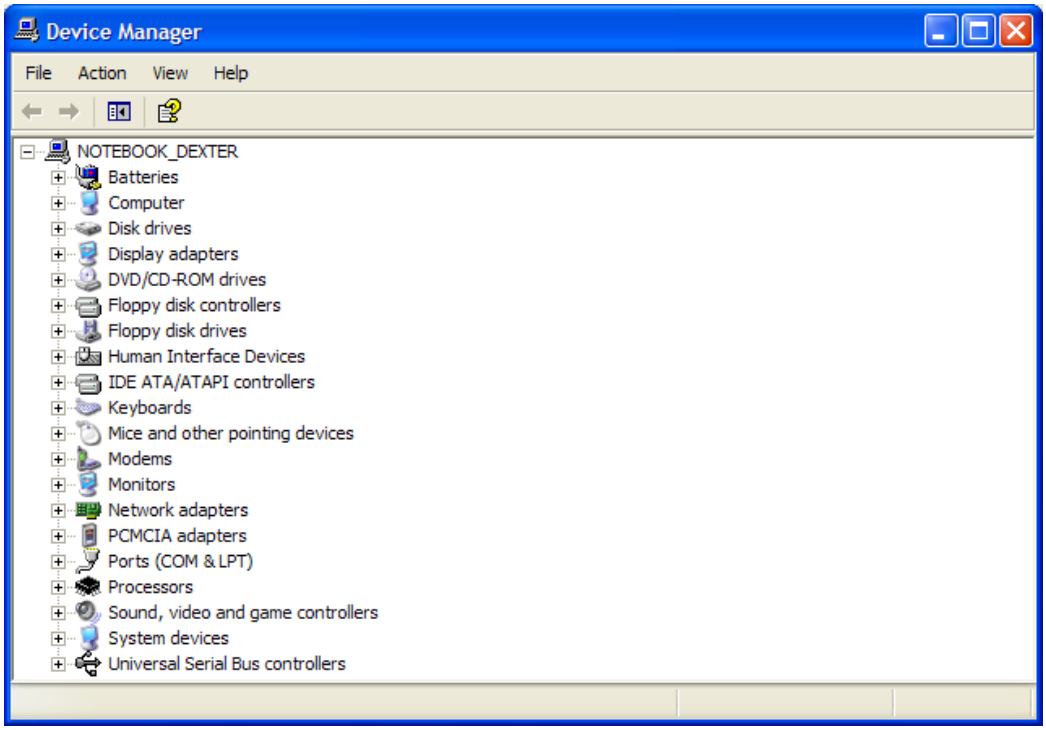


Caso seja estabelecida comunicação serial com o Controlador µDX200 a tela é preenchida com vários dados de status, e deve surgir o desenho do número de Expansões de Entrada/Saída (µDX210) utilizados (no mínimo duas Expansões).

Para estabelecer comunicação serial com o µDX200 ligue o cabo serial que acompanha o µDX200 à porta serial de seu computador (ou a um cabo adaptador USB ⇔ Serial caso seu computador não possua porta serial), e selecione o número de porta serial no Compilador. Para saber quais as portas seriais disponíveis vá no Windows em [Iniciar] → [Configurações] → [Painel de Controle]:



Selecione [Sistema] → [Hardware] → [Gerenciador de Dispositivos]:



A seguir, clique no símbolo [+] do item Portas (COM & LPT) para visualizar as portas seriais (COM) disponíveis. De posse dos endereços das portas seriais retorne ao Compilador PG, selecione o menu **Comunicação** → **Configurar Comunicador**:

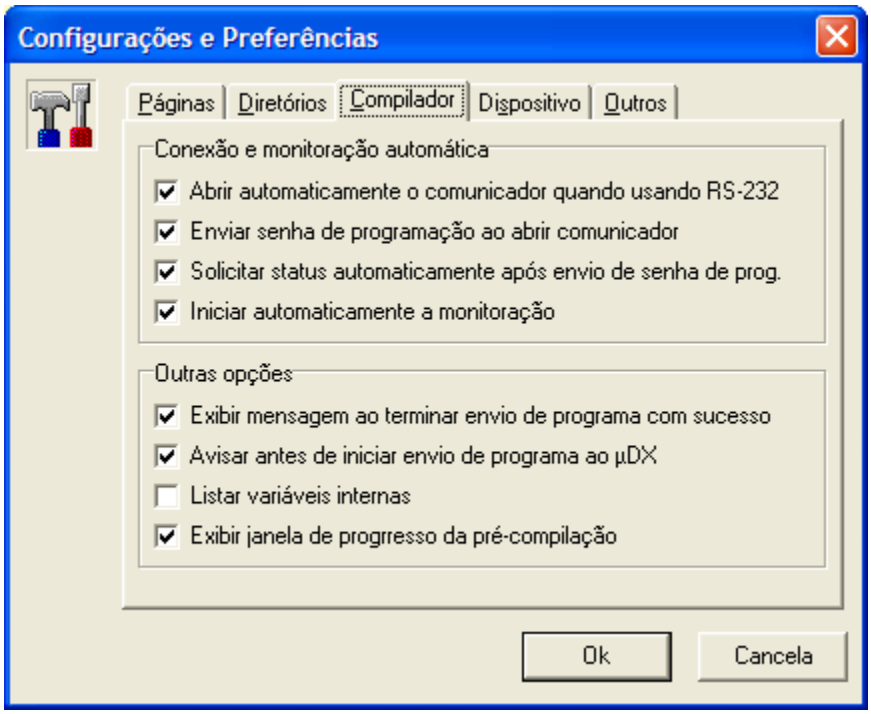
Selecione meio de comunicação RS-232, conforme a figura acima. A seguir, vá para a aba **RS-232**:

Selecione a porta de comunicação correta e pressione **Ok**. Note que os demais parâmetros não necessitam ser modificados. O µDX200 é fornecido programado para comunicação serial à 38400bps, 8 bits, sem paridade e 2 stop bits. Por fim, clique no ícone para abrir comunicador  , ou selecione **Comunicação** → **Abrir**.

Se for estabelecida comunicação serial deverão surgir diversos dados do controlador µDX200. Para isso devem estar selecionadas as opções:

[x] Enviar senha de programação ao abrir Comunicador.
[x] Solicitar status automaticamente após envio de senha de prog.
[x] Iniciar automaticamente a monitoração.

Estas opções estão disponíveis no menu **Configurações...** → **Configurações e Preferências** → **Compilador**.



Também é possível executar estes passos manualmente. Para isso clique nos seguintes ícones:

 Transmite senha de programação, permitindo o µDX200 entrar em modo de programação.

 Requisita status do controlador programável.

 Inicia monitoramento.

Note a existência de vários dados lidos do µDX200 na tela do Compilador. Assim, o Compilador lê do µDX200 as informações de:

- Nome do programa (em 8 caracteres)
- Autor (zero significa Dexter)
- Versão do firmware (software interno do µDX200)
- Tipo de operação na rede DXNET (Servidor ou Cliente)
- Modo da porta serial (Programação ou Normal)
- Tipo de CPU (no caso, µDX200)
- Tensão da bateria interna
- Tensão de alimentação elétrica
- Temperatura interna do µDX200 (deve estar abaixo dos 55°C)
- Tempo de ciclo de execução do programa aplicativo
- Data e hora do relógio de tempo real do µDX200
- CRC (Cyclic Redundance Check) do programa no Compilador e no µDX200

Além disso, estão disponibilizados os sinais de handshake da porta serial e as entradas digitais E9 e E10 sob a forma de leds (lâmpadas). Estes assumem a cor preta quando desligados, vermelho quando ligados, e vermelhos com moldura amarela quando estão ligados devido ao forçamento via Compilador. Para forçar um destes sinais a nível alto basta clicar duas vezes sobre o led correspondente com a tecla esquerda do mouse.

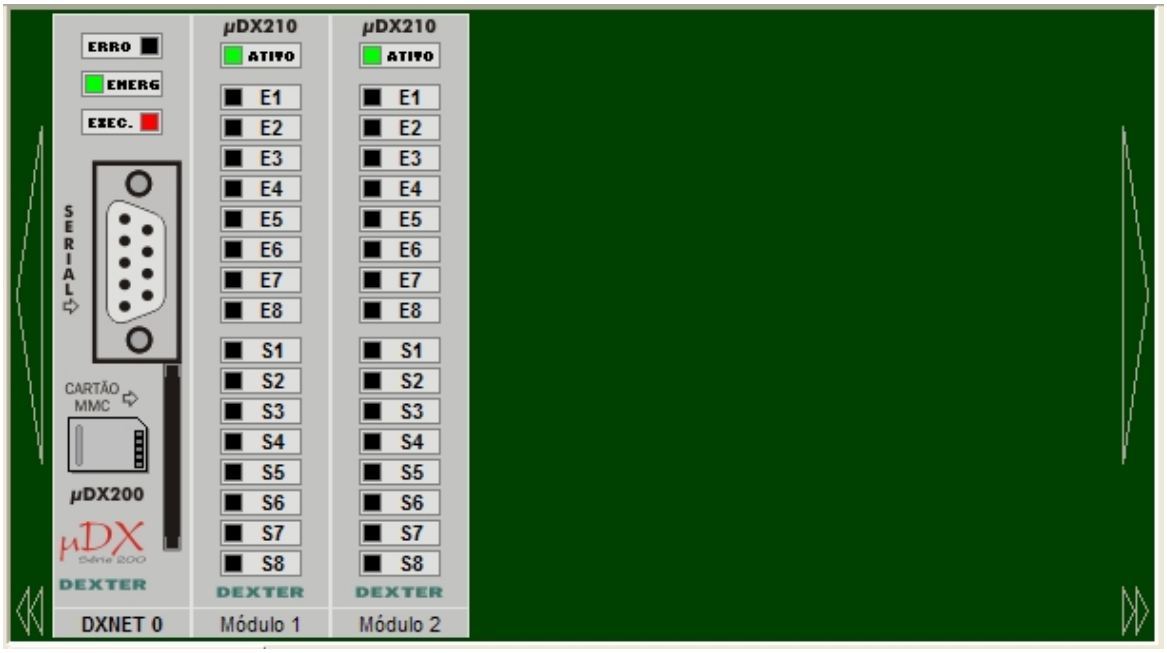
Por fim, ao lado aparecem todas as variáveis reservadas do µDX200 (de V0 a V16, sendo que V16 é uma variável longint e, portanto, ocupa V16 e V17). O tipo de variável é especificada pela

letra logo após a designação da variável. Assim, I significa variável inteira, W variável word, L variável longint, e R variável real.

Cabe aqui uma descrição das várias áreas existentes na janela do Compilador PG e suas funcionalidades:

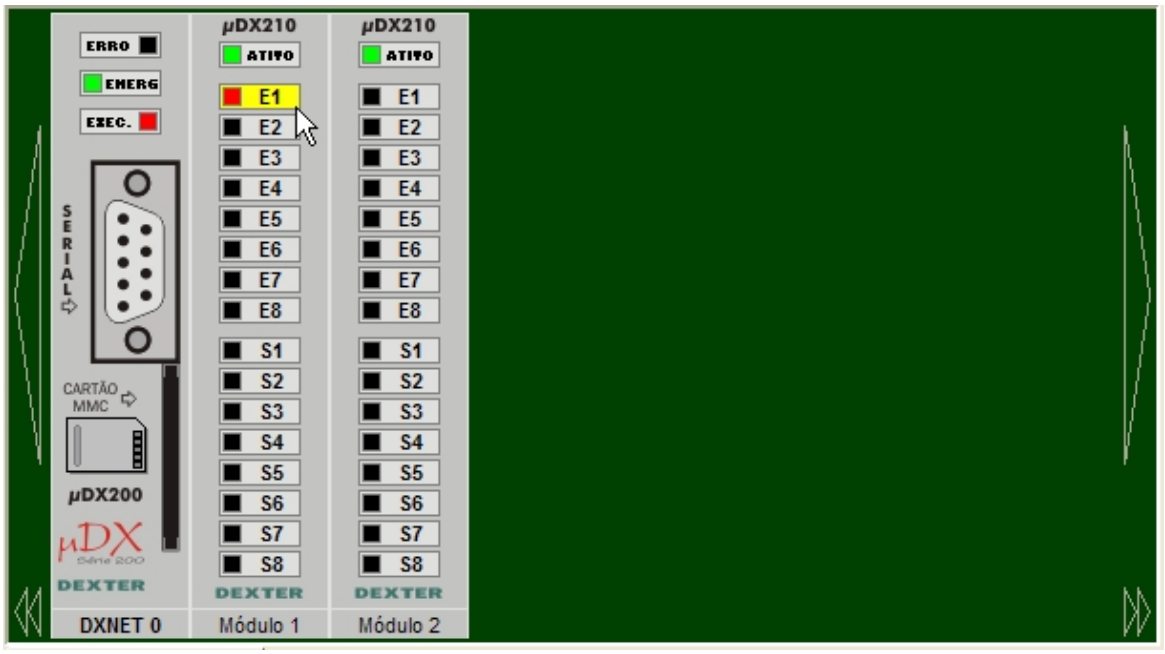
Área do Controlador µDX200 e Expansões µDX210

Esta área mostra uma vista do painel lateral do µDX200 e das Expansões µDX210 atualmente sendo varridas pelo controlador. Note que o mínimo são duas Expansões µDX210, e elas podem ser em número de 2, 4, 8, 16, ou 32.



As setas existentes à direita e à esquerda permitem acessar as demais Expansões µDX210, quando existirem. Neste caso as setas ficam ativas (com cor laranja). O cartão MMC representado no painel do µDX200 indica se há cartão instalado no µDX200 e sua capacidade (para isso é preciso instalar um cartão MMC formatado no µDX200 e programar o CLP com um programa que utilize, pelo menos, um bloco MMC). Os leds representam o estado do controlador e das expansões. Assim, eles ligam conforme o estado do CLP (programa aplicativo sendo executado, existência de erro) e conforme as entradas e saídas dos µDX210.

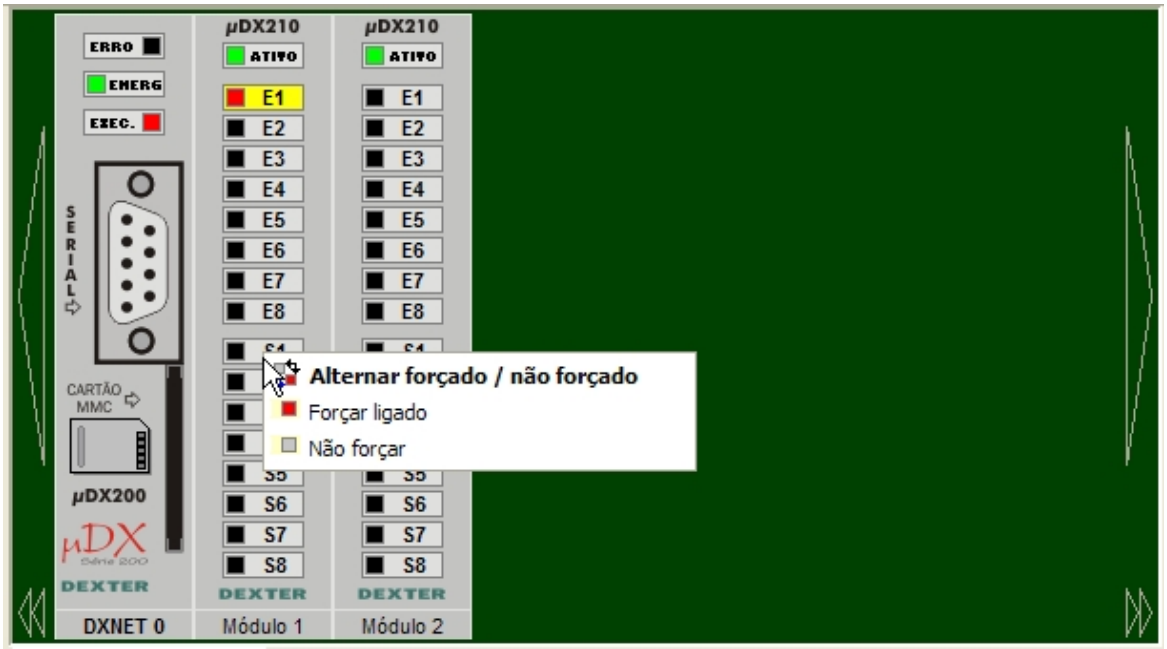
Durante monitoramento é possível forçar as entradas e saídas das Expansões µDX210, bastando apontar para a entrada ou saída e clicar duas vezes com o botão esquerdo do mouse (o led fica em vermelho com a moldura em amarelo para indicar que este nodo foi ligado forçadamente, e não como resultado de cálculo efetuado no programa aplicativo). Por exemplo, na figura a seguir foi forçado o nodo da entrada E1 da primeira Expansão µDX210:



Também é possível apontar para o nodo desejado e clicar a tecla esquerda do mouse. Surge uma janela com as seguintes opções:

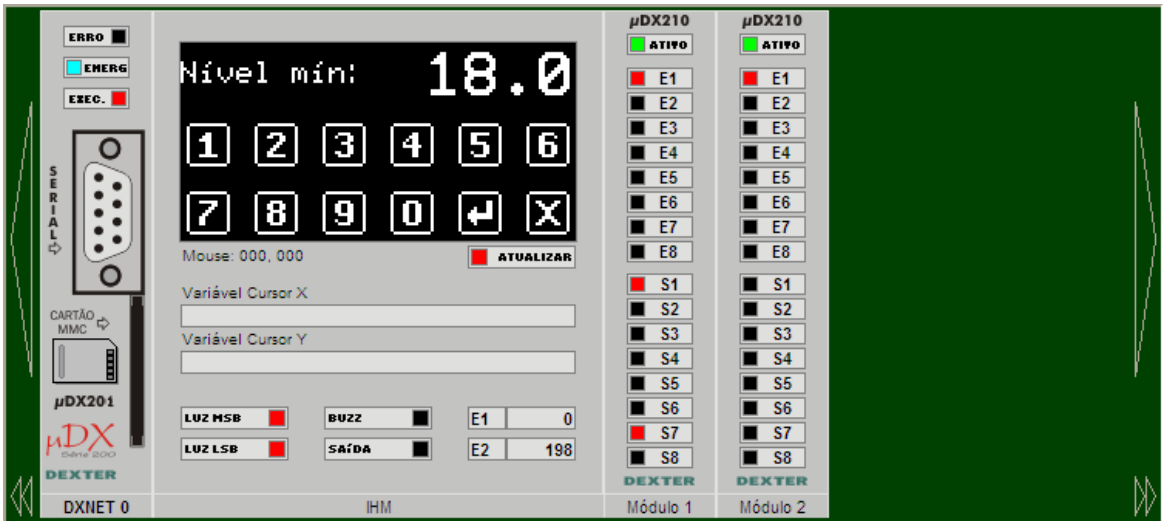
- **Alternar forçado / não forçado**
- **Forçar ligado**
- **Não forçar**

Note que esta operação acessa os nodos DXNET do Controlador µDX200. Como estes nodos fazem uma operação OR com os nodos calculados pelo programa aplicativo, sempre é possível forçar a energização de um nodo, mas não seu desligamento (caso o nodo esteja ligado devido ao programa aplicativo não é possível desligá-lo via nodo DXNET; por isso as opções são entre nodo forçado (ligado) ou não forçado (ligado ou não, conforme programa aplicativo). Abaixo foi usada a tecla direita do mouse sobre o nodo correspondente a saída S1 da primeira Expansão µDX210:

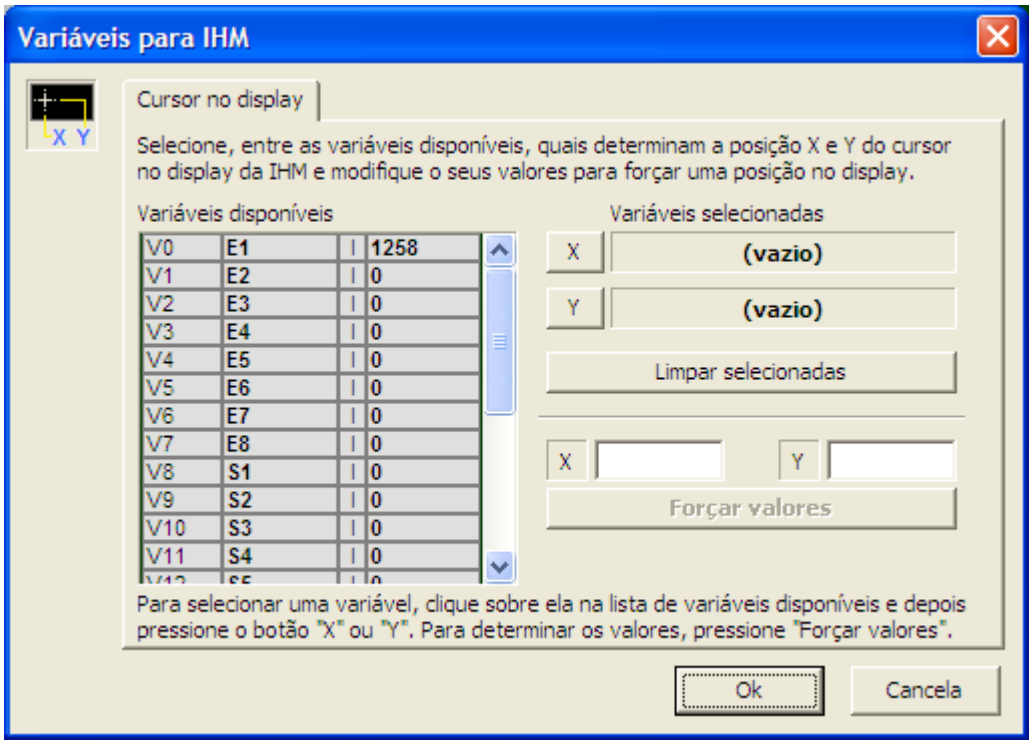


Ainda nesta área é possível editar o endereço DXNET acessado pelo Compilador PG. Note que o endereço 0 (zero) sempre acessa o µDX200 conectado ao microcomputador via porta serial, endereço TCP/IP, ou Modem. Ou seja, independentemente do endereço DXNET do µDX200, se for programado no Compilador PG para que ele utilize o endereço 0 (zero) a comunicação será feita com o dispositivo conectado ao computador, sem tentativa de comunicação via rede DXNET. O endereço 0 (zero) é o endereço padrão para o PG quando este é instalado, e só deve ser modificado caso existam mais controladores µDX200 ligados via rede DXNET. Para modificar este dado aponte com o mouse para o endereço DXNET (**DXNET 0**) que aparece abaixo da representação do µDX200 e clique duas vezes com a tecla esquerda do mouse. Irá surgir a seguinte janela:

No caso de controlador µDX201 as inscrições no painel lateral do CLP comutam para µDX201 e o led de energia é azul em vez de verde. Se existirem blocos de IHM no programa aplicativo do µDX201, e o CRC do programa compilado no PG conferir com o CRC do programa aplicativo carregado no µDX201, a IHM será apresentada ao lado da representação do µDX201:



Abaixo do display existe um campo com as coordenadas x e y do mouse. Ao clicar uma vez sobre o desenho do display estas coordenadas apresentam o valor apontado pelo mouse. Já a especificação das variáveis responsáveis pelas coordenadas (x,y) do display da IHM permite clicar duas vezes com o botão esquerdo sobre a representação do display no Compilador PG para forçar estas variáveis para o valor correspondente. Assim, pode-se facilmente simular o toque em determinado ponto do sensor touchscreen. Também é permitido digitar estes valores nos campos correspondentes e forçar as variáveis ao pressionar **[Forçar valores]**. A janela a seguir está disponível se for efetuado um duplo clique com o botão esquerdo do mouse sobre as variáveis de cursor X e Y existentes na representação de IHM do Compilador PG:



Selecione as variáveis pertinentes e pressione os botões [X] e [Y] correspondentes. É possível nesta janela também forçar valores para as variáveis selecionadas. Note que as coordenadas no display são crescentes da esquerda para a direita (eixo x) e de cima para baixo (eixo y). A resolução é de 128 pontos na vertical e 64 pontos na vertical:

Além disso, a partir da versão 2.2.0.16 do software PG o monitoramento da IHM do Controlador µDX201 (µDX220) inclui os nodos próprios deste dispositivo (Luz MSB, Luz LSB, Buzzer, Saída), e as duas entradas analógicas da IHM.

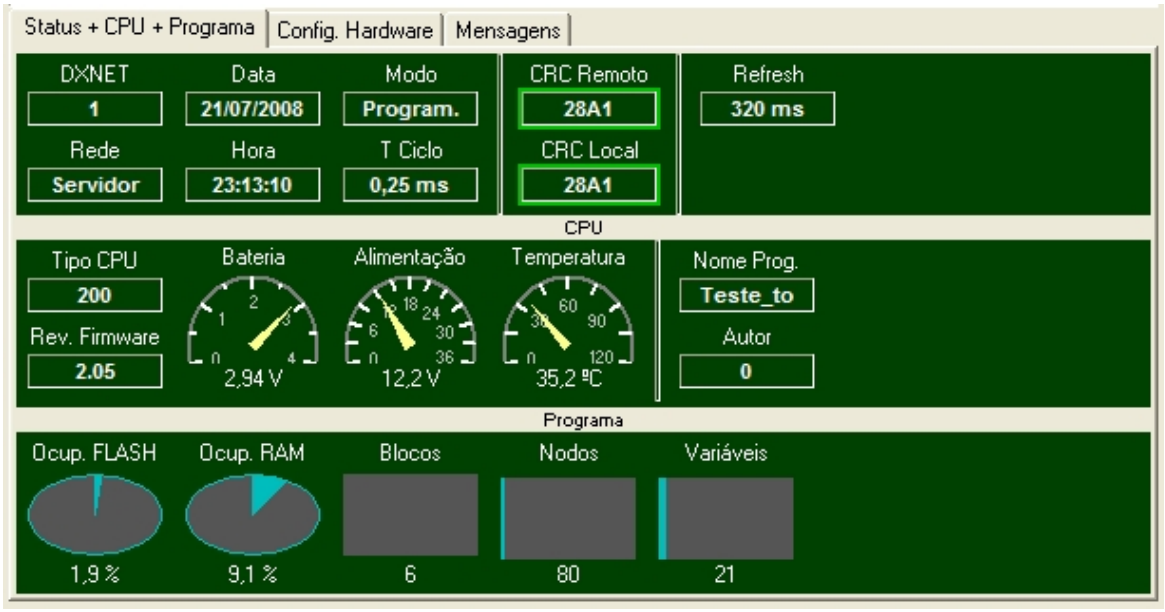
Note que a IHM do µDX201 disponibiliza duas entradas analógicas de baixa resolução (8 bits), para leitura de potenciômetros, por exemplo, e também uma saída digital, para acionamento de um buzzer externo ou outro dispositivo de sinalização. Além disso, os nodos **Luz MSB** e **Luz LSB** acionam a luz de backlight do display da IHM.

Por fim, um nodo adicional chamado **Atualizar** permite inibir o monitoramento da IHM por parte do PG. Isso é útil quando o tempo de refresh deste monitoramento fica excessivo, e não há necessidade de visualizar os dados do display na tela do PG.

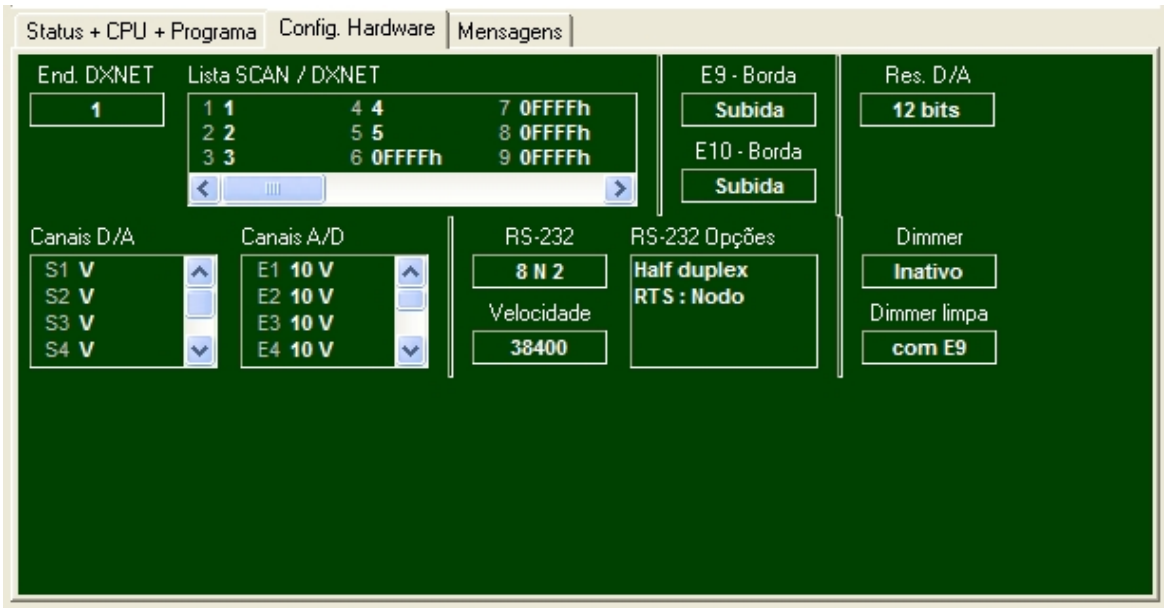
Área de Status do Controlador µDX200 e Programa Aplicativo

Na primeira aba existem os dados lidos constantemente do µDX200. São eles:

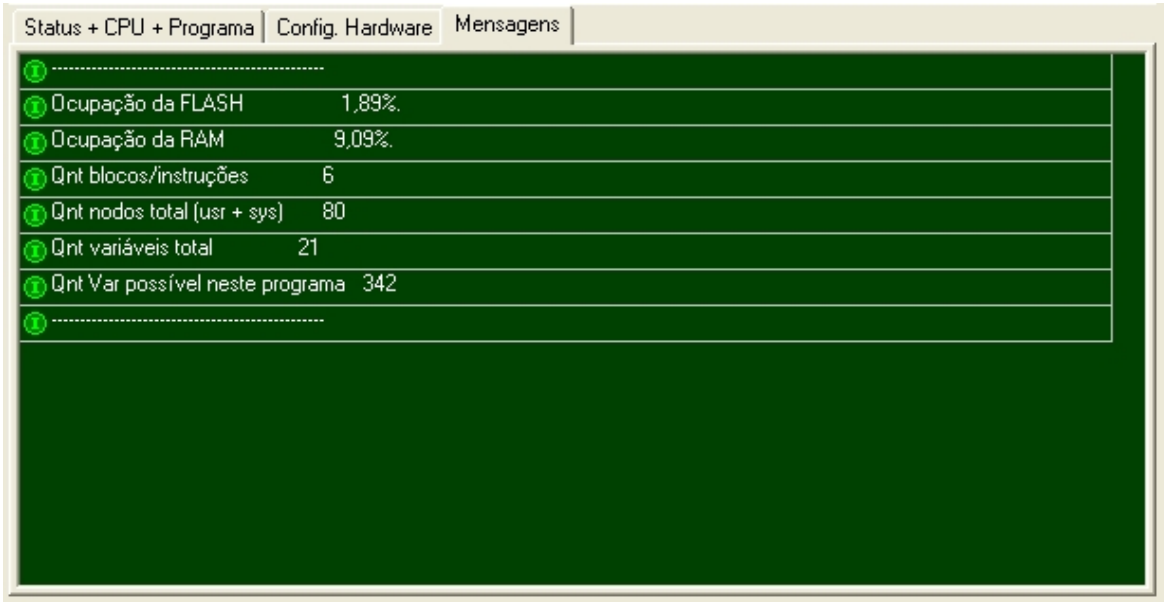
- Endereço DXNET do µDX200.
- Rede DXNET (µDX200 está na rede DXNET como Servidor ou Cliente - no caso de apenas um µDX200 na rede DXNET ele ficará comutando entre Servidor e Cliente).
- Data e Hora do relógio de tempo real do µDX200.
- Modo do Controlador µDX200 (Programação ou Normal).
- Tempo de ciclo de execução do programa aplicativo (em múltiplos de 250µs).
- CRC Remoto (do µDX200) e CRC Local (do programa aplicativo carregado no Compilador PG).
- Tempo de Refresh da tela do Compilador.
- Tipo de CPU (200 significa µDX200).
- Revisão de firmware do µDX200.
- Tensão de bateria interna (valores normais entre 2,5V e 3,5V).
- Tensão de alimentação elétrica (valores normais entre 11,0 e 27,0V).
- Temperatura interna do µDX200 (valores normais entre 0°C e 75°C).
- Nome do programa aplicativo existente no µDX200 (nome truncado em 7 caracteres).
- Autor (número entre 0 e 65535 - 0 representa a Dexter).
- Ocupação de memória FLASH do programa aplicativo carregado no Compilador PG, em percentual.
- Ocupação de memória RAM do programa aplicativo carregado no Compilador PG, em percentual.
- Número de blocos do programa aplicativo carregado no Compilador PG.
- Número de Nodos do programa aplicativo carregado no Compilador PG.
- Número de variáveis do programa aplicativo carregado no Compilador PG.



Já a segunda aba mostra as configurações de hardware (arquivo sufixo .udi) pertencentes ao programa aplicativo (arquivo sufixo .udp) carregado no Compilador PG. Estas configurações determinam que tipo de entrada e saída analógica será usada (0-10V, 4-20mA, etc), baud-rate de comunicação serial, endereço DXNET, etc.



A terceira e última aba apresenta mensagens de compilação do programa sufixo UDP carregado no Compilador PG:



Área de Condição Geral

Nesta área são apresentados três leds, que indicam o status da comunicação entre computador e µDX200, status do programa aplicativo carregado no Compilador PG, e status do próprio Controlador µDX200. Os leds assumem diferentes cores, conforme a seguinte convenção:

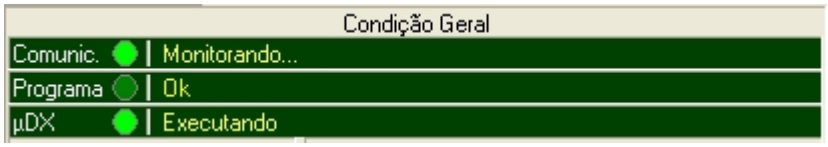
Comunicação:

Cinza → Comunicador Parado (sem comunicação).
Verde Escuro → Comunicador Aberto (sem comunicação).
Verde Claro → Monitoramento ativo (comunicação constante com µDX200).
Laranja → Comunicando (comunicação temporária de comando com µDX200).

Vermelho → Erro de comunicação (Timeout, erro de CRC).
Vermelho/Amarelo → Comunicação para carga de programa aplicativo ou leitura de cartão MMC.

Programa:
Cinza → Nenhum programa aplicativo carregado no Compilador PG (arquivo sufixo .udp).
Verde Escuro → Programa carregado no Compilador PG e compilado sem erros.
Verde Claro → Programa transmitido para µDX200 com sucesso.
Vermelho → Erro ao compilar programa aplicativo.

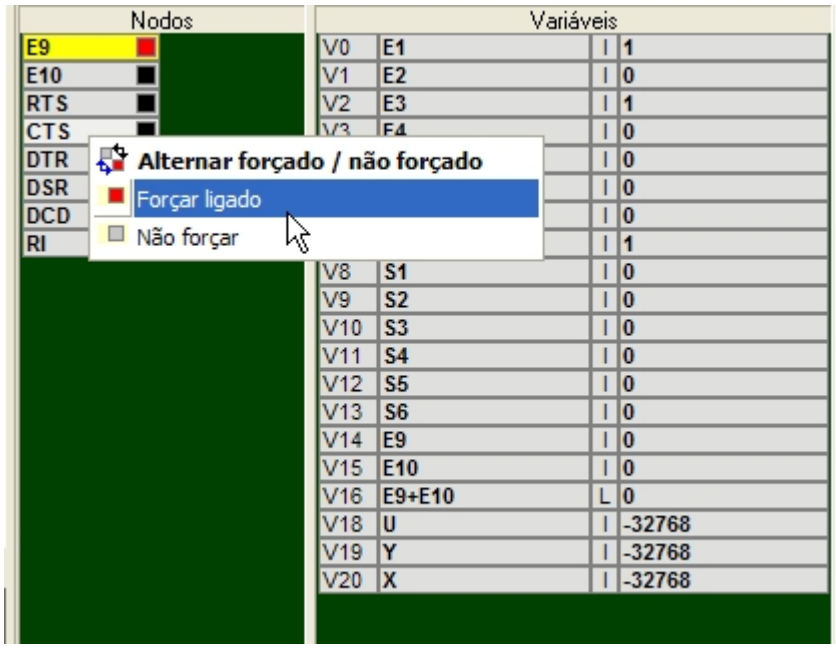
µDX:
Cinza → Não foi lido status do µDX200.
Verde Escuro → Programa aplicativo parado no µDX200.
Verde Claro → Programa aplicativo executando no µDX200.
Vermelho → Erro no µDX200.



Área de Nodos e Variáveis do Controlador µDX200

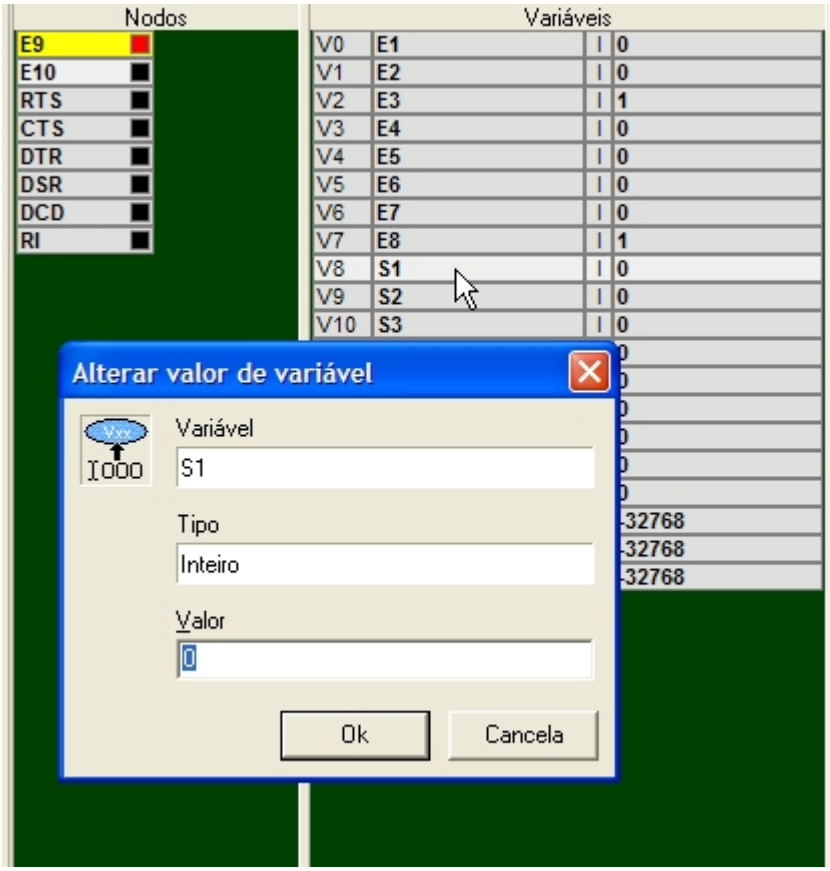
A área de nodos disponibiliza os nodos dos sinais da porta serial RS232, as entradas digitais E9 e E10, e também os nodos utilizados no programa aplicativo (para isso o programa aplicativo carregado no Compilador PG deve ser o mesmo que está no µDX200, ou seja, o CRC Local deve ser idêntico ao CRC Remoto).

Da mesma forma que na área de status do µDX200 e µDX210, aqui também é possível forçar nodos clicando duas vezes com a tecla esquerda do mouse com o cursor sobre o nodo desejado. Com a tecla direita se abre um menu de opções, como representado a seguir:



Já a área de variáveis apresenta as primeiras 18 variáveis fixas do µDX200 (v0 até V16, sendo que v16 é uma variável longint e, portanto, ocupa v16 e v17), e também as variáveis utilizadas no

programa aplicativo (desde que o programa aplicativo carregado no Compilador PG seja o mesmo que está no µDX200, ou seja, o CRC Local seja idêntico ao CRC Remoto). É possível forçar valores nestas variáveis. Para isso basta apontar com o cursor para a variável e clicar duas vezes com a tecla esquerda do mouse:

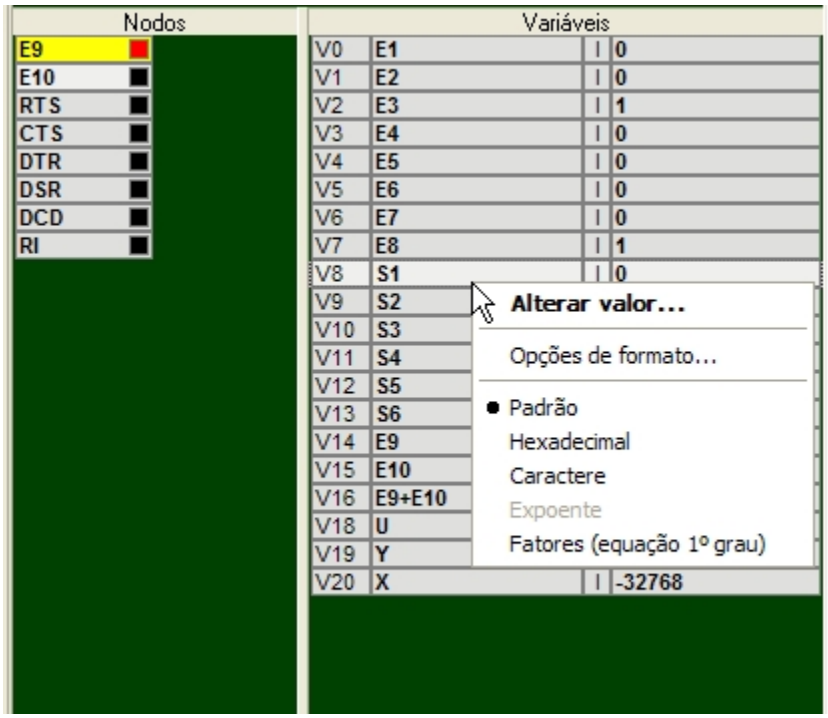


O exemplo anterior mostra o forçamento de valor na variável v8, responsável pela saída analógica S1. Se forçarmos, por exemplo, valor 2048 (metade da escala da saída analógica) em v8, e a saída analógica estiver em saída 0-10V, resultará em uma tensão de saída de 5V em S1.

Note que as variáveis possuem quatro colunas. A primeira indica seu endereço absoluto, a segunda indica o nome da variável, a terceira coluna indica o tipo de variável (I para inteira; W para word; L para longint; e R para real), e a quarta coluna mostra o valor da variável.

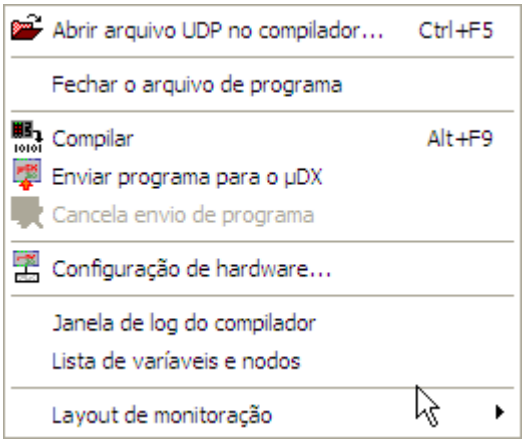
Clicando sobre a variável com a tecla direita do mouse surge uma janela de opções como mostrado a seguir. Nela existem as seguintes possibilidades:

- **Alterar valor:** exatamente a mesma funcionalidade já descrita obtida ao clicar duas vezes sobre a variável.
- **Opções de formato:** permite escolher entre notação Padrão, Hexadecimal, Caracter ASCII, Expoente (notação de engenharia), ou ainda com Fatores (equação de primeiro grau y=ax+b). Esta última opção permite converter a variável para outro valor. Por exemplo, se a variável representa uma temperatura pode-se converter o valor da variável para que apresente a temperatura diretamente em graus celsius. As opções de formato estão disponíveis também diretamente na janela, de forma a permitir comutar rapidamente entre as representações.


Nodos
E9
E10
RTS
CTS
DTR
DSR
DCD
RI
Variáveis
V0 E1 | 0
V1 E2 | 0
V2 E3 | 1
V3 E4 | 0
V4 E5 | 0
V5 E6 | 0
V6 E7 | 0
V7 E8 | 1
V8 S1 | 0
V9 S2
V10 S3
V11 S4
V12 S5
V13 S6
V14 E9
V15 E10
V16 E9+E10
V18 U
V19 Y
V20 X | -32768
Alterar valor...
Opções de formato...
Padrão
Hexadecimal
Caractere
Expoente
Fatores (equação 1º grau)

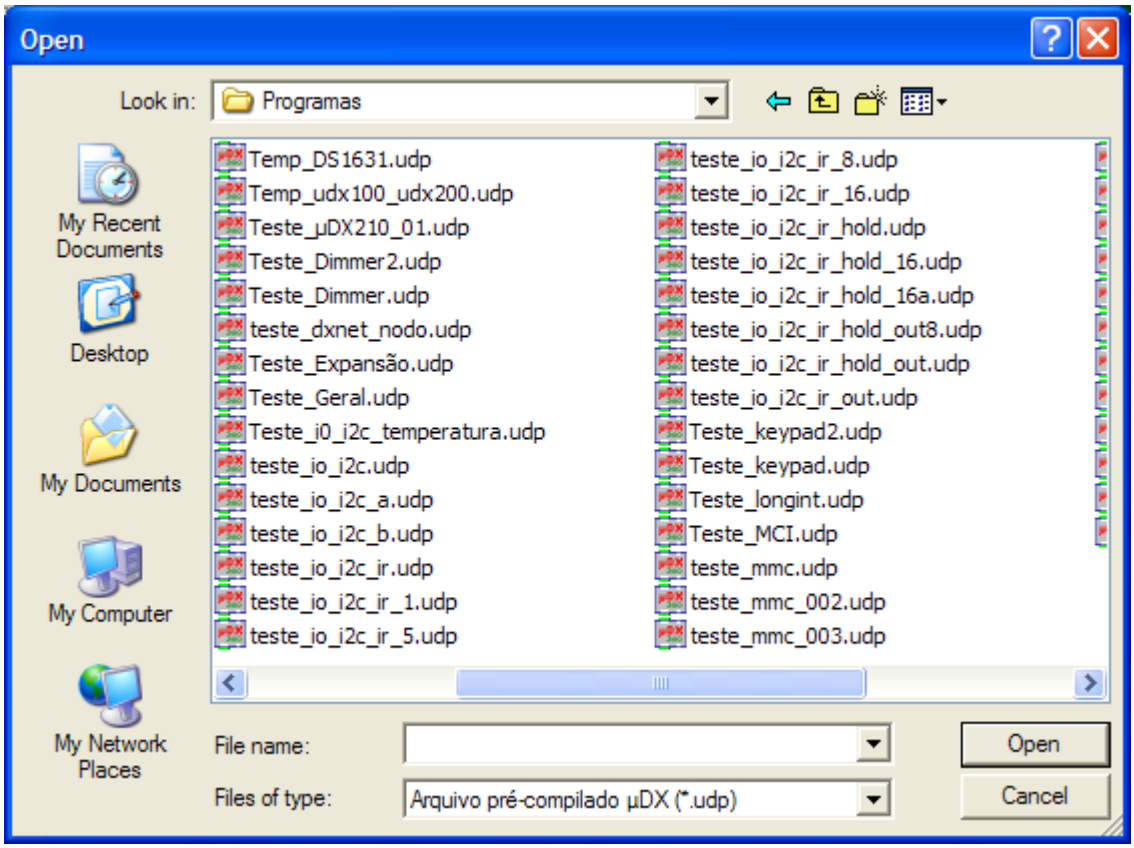
Menu Compilador

O menu Compilador possibilita carregar programas, compilá-los, configurar o hardware associando esta configuração ao programa caregado no Compilador, e abrir janela de log. Também é possível editar um layout de monitoração, escolhendo quais as variáveis e nodos do programa aplicativo devem ser monitorados.



Abrir Arquivo UDP no Compilador... (Ctrl+F5)

Permite abrir no Compilador PG o programa pré-compilado no Editor PG. Note que uma página de programação (sufixo .dxg) ou um projeto (sufixo .dxp) é pré-compilado no Editor PG, gerando um arquivo sufixo .udp, que pode ser lido no Compilador (que irá efetuar a compilação final, permitindo a transmissão do programa aplicativo para o µDX200). Ao selecionar esta opção surge a janela a seguir, que permite a escolha do programa a ser carregado:



Fechar o Arquivo de Programa

Fecha o programa aplicativo pré-compilado (sufixo .udp) previamente aberto no Compilador.

Compilar (Alt+F9)

Compila o programa previamente carregado no Compilador. Note que, ao carregar um programa sufixo .udp no Compilador PG ele já é compilado. Assim, esta tecla é útil apenas para efetuar nova compilação (por exemplo, para verificar as mensagens de compilação no log do compilador).

Enviar Programa para o µDX

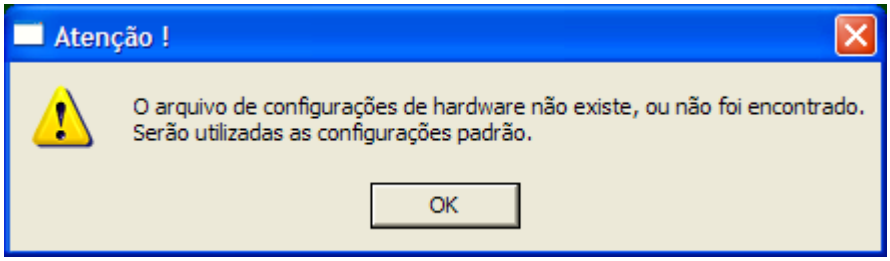
Envia o programa aplicativo compilado para o µDX200. Para isso é necessário que o canal de comunicação esteja aberto (Comunicador aberto). Ao enviar o programa existente no Compilador para o CLP os CRCs (cyclic redundance check) remoto e local devem ficar idênticos, indicando que o programa foi corretamente gravado no µDX200.

Cancela Envio de Programa

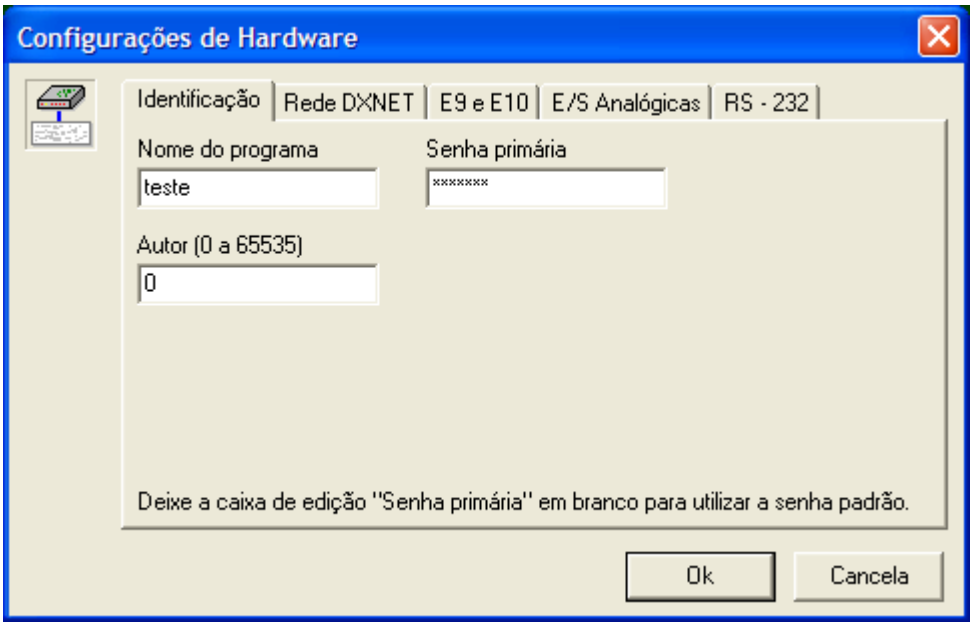
Permite cancelar o envio de programa aplicativo compilado para o µDX200. Esta tecla é útil no caso de programas extensos, que levam vários segundos para serem transmitidos, ou no caso de comunicação lenta com o µDX200 (baud rate baixo).

Configuração de Hardware

Ao carregar pela primeira vez um programa aplicativo pré-compilado (sufixo .udp) no Compilador PG aparece a seguinte mensagem de advertência:

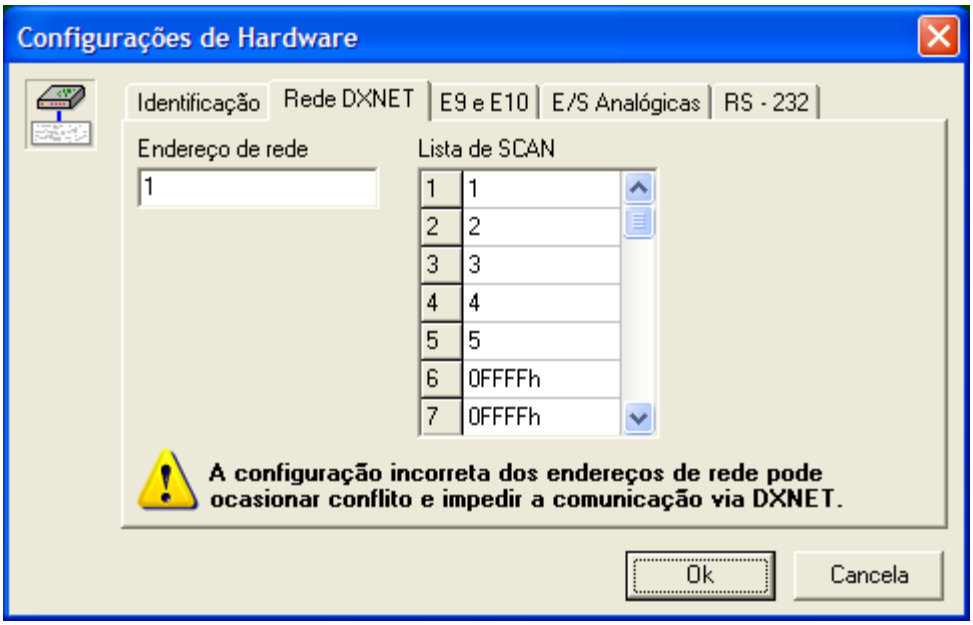


Isso porque não havia sido gerado ainda um arquivo de configuração de hardware (sufixo .udi) associado a este programa aplicativo. O arquivo de configuração de hardware possui o mesmo nome do programa aplicativo, mas com sufixo UDI. Estes parâmetros são associados ao programa aplicativo (arquivo sufixo .UDP) gerado pelo Editor PG. Assim, os parâmetros escolhidos para determinado programa aplicativo são salvos com o mesmo nome do programa (arquivo sufixo .UDI). O Compilador irá gerar um arquivo padrão de configuração. Esta opção permite editar este arquivo, de forma a modificar parâmetros de hardware associados a este programa, como taxa de comunicação usada (baud rate) ou escala das entradas analógicas (0-10V, 0-2,5V, 0-20mA). Os parâmetros disponíveis são (as figuras a seguir mostram a configuração padrão gerada para um programa aplicativo chamado **teste**):

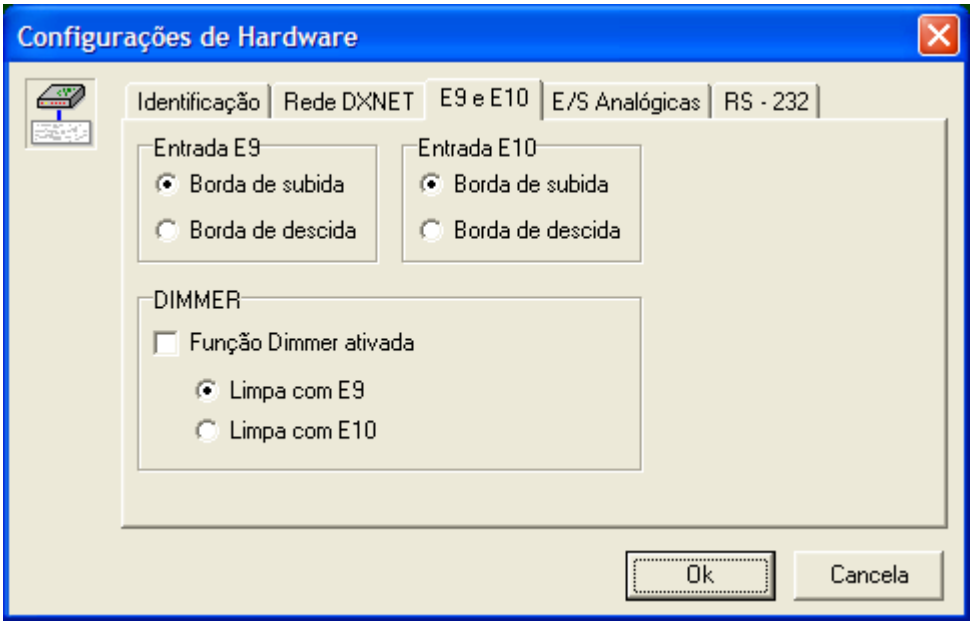


Na aba de **Identificação** existe o nome do programa (truncado em oito caracteres) que será gravado no µDX201, um numeral de 0 a 65535 para identificação do autor (zero significa Dexter), e a senha para entrar em modo de programação. A senha deve conter sete caracteres, e é sensível a caracteres maiúsculos e minúsculos (ou seja, a senha **TesteXX**, por exemplo, é diferente de **testexx** ou **TESTEXX**). A senha padrão é **[uDX200** e ela aparece representada pelos asteriscos no campo de senha. Se este campo for deixado em branco será assumida a senha padrão.

Atenção: Muito cuidado ao modificar esta senha! Se for transmitido um programa aplicativo para o µDX201 com outra senha que não a senha padrão ele passará a exigir a nova senha para permitir novas programações. Sem conhecê-la será impossível reprogramar o controlador programável! Trata-se de uma segurança, em especial quando o CLP está acessível remotamente (via endereço TCP/IP ou via Modem), mas deve-se tomar o devido cuidado para anotar a senha em local acessível.

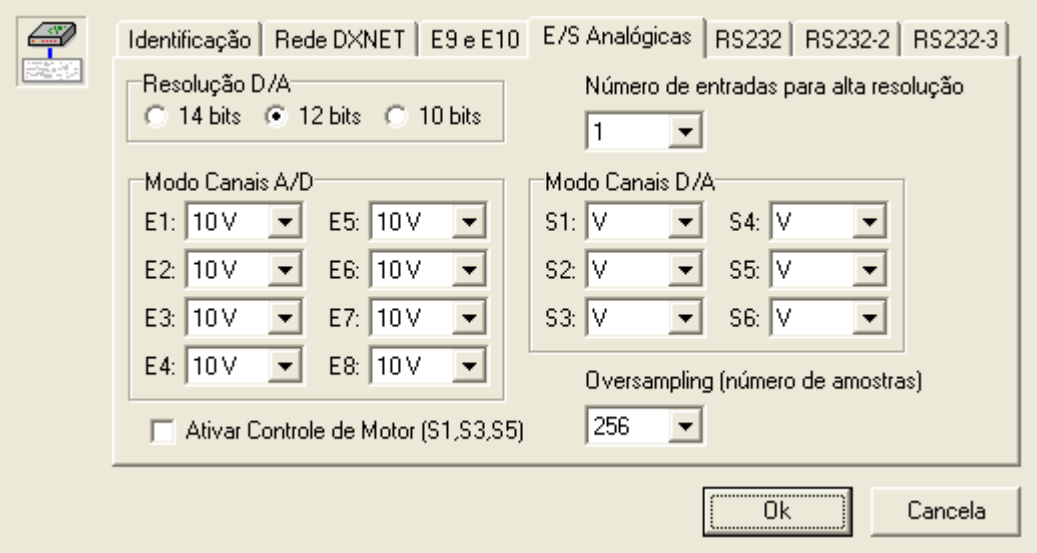


A aba **Rede DXNET** possui campo para selecionar endereço DXNET que o µDX201 irá assumir ao receber programa aplicativo, e também lista de scan para a rede DXNET. Note que apenas os endereços constantes nesta lista serão varridos na rede DXNET. Assim, é mandatório que todos os endereços de dispositivos na rede DXNET constem nesta lista de scan (e em todos os µDX200s existentes na rede DXNET, já que qualquer um deles pode assumir como mestre da rede). O endereço 0FFFFh (65535) é inválido e indica uma posição da lista de scan sem endereço a ser varrido. No caso são lidos os endereços de 1 a 5, e o µDX200 que receber o programa irá assumir endereço 1 na rede DXNET.



A aba **E9 e E10** parametriza as entradas digitais rápidas E9 e E10 do controlador µDX201. É possível programar se estas entradas irão gerar interrupção (veja ENTRADA/SAÍDA - Interrupção E9 e E10) na borda de subida ou na borda de descida do sinal de entrada, se a função DIMMER (que permite usar uma saída analógica no modo PWM para comandar dimmer), e qual entrada (E9 ou E10) será usada para limpar a contagem de tempo e sincronizar o dimmer com a rede elétrica (nesta entrada deve ser ligado o equipamento Sensor de Zero).

As entradas digitais E9 e E10 estão associadas à interrupções no controlador µDX201. Por isso permitem altas taxas de leitura (até 8KHz).



A aba **E/S Analógicas** modifica parâmetros das entradas analógicas (E1 a E8) e saídas analógicas (S1 a S6) do Controlador µDX201. As configurações disponíveis são:

- **Resolução D/A:** 14 bits, 12 bits, 10 bits (default = 12 bits).
- **Modo Canais D/A:** V, mA, PWM, FREQ (default = V).
- **Modo Canais A/D:** 2,5V, 10V, 20mA (default = 10V).

As saídas analógicas podem ser programadas para resolução de 10, 12 ou 14 bits (1024, 4096 ou 16384 passos). A vantagem da resolução de 10 bits é que permite saídas analógicas mais rápidas. Assim, em 10 bits pode-se gerar sinais analógicos de até 100Hz, contra apenas 10Hz no caso de saídas analógicas em 12 bits. Em 14 bits as saídas analógicas ficam ainda mais lentas que em 12 bits, mas com resolução 4x maior. As saídas analógicas também podem ser programadas para escala de 0-10V, 0-20mA (ou 4-20mA), ou ainda saída PWM ou FREQ. No caso de saída PWM (pulse width modulator) ou FREQ (freqüência), o sinal excursiona entre 0 e 10V, em onda quadrada, com a duração de pulso modulada, ou freqüência selecionada. Caso o modo DIMMER esteja selecionado e a entrada E9 ou E10 esteja ligada ao Sensor de Zero, é possível comandar módulos de Dimmer via saídas analógicas no modo PWM.

As entradas analógicas podem ser programadas para escala de 0-10V, 0-20mA (ou 4-20mA), ou ainda escala de 0-2,5V. Em todas estas escalas a resolução das entradas analógicas é de 12 bits (4096 passos).

As saídas analógicas no modo **PWM** geram pulsos de largura variável, em uma freqüência fixa, que depende da resolução das saídas analógicas selecionada:

Resolução 10 bits: Saída analógica **PWM** com onda quadrada de 7,81 KHz, modulada em largura de pulso (de 0 a 1024)

Resolução 12 bits: Saída analógica **PWM** com onda quadrada de 1,95 KHz, modulada em largura de pulso (de 0 a 4096)

Resolução 14 bits: Saída analógica **PWM** com onda quadrada de 488 Hz, modulada em largura de pulso (de 0 a 16384)

Note que valor 0 mantêm saída analógica sempre desligada, enquanto o valor máximo de cada resolução mantêm a saída analógica sempre ligada. Valores intermediários determinam uma relação proporcional entre o tempo ligado e desligado. Por exemplo, na resolução de 12 bits, um valor de 1024 determina que a onda quadrada na saída analógica permaneça 25% do tempo de ciclo ligada, e 75% desligada.

As saídas analógicas no modo **FREQ** geram ondas quadradas nas saídas analógicas, com um período determinado pelo valor da saída. A faixa de variação desse período depende da

resolução das saídas analógicas selecionada:

Resolução 10 bits: Saída analógica **FREQ** com onda quadrada de período entre 100µs e 255,8µs (de 400 a 1023)

Resolução 12 bits: Saída analógica **FREQ** com onda quadrada de período entre 100µs e 1,023ms (de 400 a 4093)

Resolução 14 bits: Saída analógica **FREQ** com onda quadrada de período entre 100µs e 4,093ms (de 400 a 16383)

Note que valor 0 mantêm saída analógica sempre desligada, enquanto valores maiores que o valor máximo de cada resolução mantêm a saída sempre ligada. O período é determinado proporcionalmente em relação ao valor máximo da resolução utilizada. Pode-se usar a seguinte fórmula para determinar o período da onda quadrada gerada na saída analógica:

Período = 125ns x Valor

Por exemplo, um valor de 1000 (que é válido em qualquer das três resoluções) gera um período de 125µs.
Veja que para determinar a freqüência gerada na saída analógica é preciso fazer o inverso do período:

Freqüência = 1 / Período

Freqüência = 4.000.000 / Valor

Portanto, a freqüência na saída analógica, ao contrário do período, e fortemente não-linear. Para linearizar a freqüência na saída analógica existe um bloco no conjunto de blocos **Entrada/Saída**, chamado **Lineariza Motor**. Esse bloco permite entrar com valor entre 10 e 400, correspondendo a valores linearizados de 25 em 25Hz. Portanto, valor 10 na entrada desse bloco irá gerar valor correspondente a freqüência de 250Hz (10 x 25Hz), e valor de entrada 400 irá gerar valor correspondente para gerar freqüência de 10KHz (400 x 25Hz). Ou seja, para entrada de 10 a saída desse bloco será 16000 (4.000.000/16.000 = 250Hz), enquanto que para entrada de 400 a saída desse bloco será 400 (4.000.000/400 = 10KHz). Obviamente, essa faixa entre 10 e 400 só é funcional para resolução de 14 bits nas saídas analógicas. No caso de resolução de 12 bits a faixa se reduz para valores entre 40 (1KHz) e 400 (10KHz). No caso de resolução de 10 bits a faixa se reduz ainda mais, sendo de 157 (3925Hz) a 400 (10KHz).

Note que os jumpers internos do µDX201 devem estar na posição **PWM** para uso no modo **FREQ** .

Outra opção disponível nessa aba da **Configuração de Hardware** é **Ativar Controle de Motor (S1,S3,S5)**. Essa opção permite usar blocos de MOVE e RUN para controlar até 3 motores independentes, com curva de aceleração e desaceleração, e posicionamento preciso. Para isso utiliza as saídas S1,S2 para pulsos e sentido de rotação do motor 1; S3,S4 para pulsos e sentido de rotação do motor 2; e S5,S6 para pulsos e sentido de rotação do motor 3. As saídas analógicas devem estar em modo FREQ e os jumpers internos em posição PWM. Além disso, é recomendável usar as saídas analógicas em resolução de 14 bits, de forma a permitir uma faixa de velocidades mais ampla para os motores (o que irá permitir acelerações e desacelerações mais suaves).

No caso de uso de controle de motor automaticamente a porta RS232-2 é inibida.

Atenção: Além de modificar nesta janela as opções de entradas e saídas analógicas é necessário modificar os jumpers internos correspondentes na placa impressa do controlador µDX200 (veja capítulo sobre jumpers do µDX200 - Seleção de Jumpers).

Atenção: No caso de controladores µDX201 versão 3.47 ou superior é possível programar o número de entradas analógicas que serão consideradas para a conversão analógica/digital de alta resolução (palavra reservada _E1). Com isso, é possível, por exemplo, conectar várias células de carga ao controlador µDX201 (via placas amplificadoras de célula de carga) e obter uma soma dos valores lidos em alta resolução em _E1 (16 bits). Além disso, via variáveis absolutas v0 a v7 pode-se ler cada célula de carga individualmente em resolução menor (12

bits), permitindo detectar problemas em alguma célula de carga.

No caso de controladores µDX201 versão 3.72 ou superior é possível programar a profundidade de sobre amostragem (oversampling) das entradas analógicas. Para oversampling menor que 256 é possível ler em alta resolução mais de uma entrada analógica. Para isso foram previstas as palavras reservadas _E2, _E3, _E4, _E5, _E6, _E7 e _E8. As características do oversampling para cada amostragem são as seguintes:

256 amostras	leitura em 16 bits Resposta até 7 Hz	Entrada E1	Atraso de 68ms
128 amostras	leitura em 15 bits Resposta até 15 Hz	Entradas E1,E5	Atraso de 34ms
64 amostras	leitura em 14 bits Resposta até 29 Hz	Entrada E1,E3,E5,E7	Atraso de 17ms
32 amostras	leitura em 13 bits Resposta até 59 Hz	Entrada E1,E2,E3,E4,E5,E6,E7,E8	Atraso de 8,5ms

A tabela abaixo explicita as formas de operação do oversampling das entradas analógicas do µDX201. Note que, no caso de 256 amostras, apenas a entrada E1 permite oversampling, mas é possível usar a soma de mais de uma entrada analógica (até o máximo das 8 entradas) para gerar a variável longint designada pela palavra reservada _E1. Já no caso de 128 amostras podemos ter duas variáveis de oversampling (_E1 e _E5), sendo que, novamente, é possível fazer a soma de várias entradas analógicas para obter _E1 e _E5. Por exemplo, poderíamos usar as entradas E1, E2, E3 e E4 para gerar _E1, e E5, E6, E7 e E8 para gerar _E5 (número de entradas=4). Com isso se obtêm uma resolução maior, pois se faz a média de 128 amostras de 4 entradas analógicas, em vez de apenas uma. E assim por diante, até que com 32 amostras é possível obter oversampling das 8 entradas analógicas independentemente.

Oversampling = 32 (13 bits - leitura x 2)

Número de entradas	_E1	_E2	_E3	_E4	_E5	_E6	_E7	_E8
1	E1	E2	E3	E4	E5	E6	E7	E8
2	E1+E2	E2+E3	E3+E4	E4+E5	E5+E6	E6+E7	E7+E8	E8+E1
3	E1+E2+E3	E2+E3+E4	E3+E4+E5	E4+E5+E6	E5+E6+E7	E6+E7+E8	E7+E8+E1	E8+E1+E2
4	E1+E2+E3+E4	E2+E3+E4+E5	E3+E4+E5+E6	E4+E5+E6+E7	E5+E6+E7+E8	E6+E7+E8+E1	E7+E8+E1+E2	E8+E1+E2+E3
5	E1+E2+E3+...+E5	E2+E3+E4+...+E6	E3+E4+E5+...+E7	E4+E5+E6+...+E8	E5+E6+E7+...+E1	E6+E7+E8+...+E2	E7+E8+E1+...+E3	E8+E1+E2+...+E4
6	E1+E2+E3+...+E6	E2+E3+E4+...+E7	E3+E4+E5+...+E8	E4+E5+E6+...+E1	E5+E6+E7+...+E2	E6+E7+E8+...+E3	E7+E8+E1+...+E4	E8+E1+E2+...+E5
7	E1+E2+E3+...+E7	E2+E3+E4+...+E8	E3+E4+E5+...+E1	E4+E5+E6+...+E2	E5+E6+E7+...+E3	E6+E7+E8+...+E4	E7+E8+E1+...+E5	E8+E1+E2+...+E6
8	E1+E2+E3+...+E8	E2+E3+E4+...+E1	E3+E4+E5+...+E2	E4+E5+E6+...+E3	E5+E6+E7+...+E4	E6+E7+E8+...+E5	E7+E8+E1+...+E6	E8+E1+E2+...+E7

Oversampling = 64 (14 bits - leitura x 4)

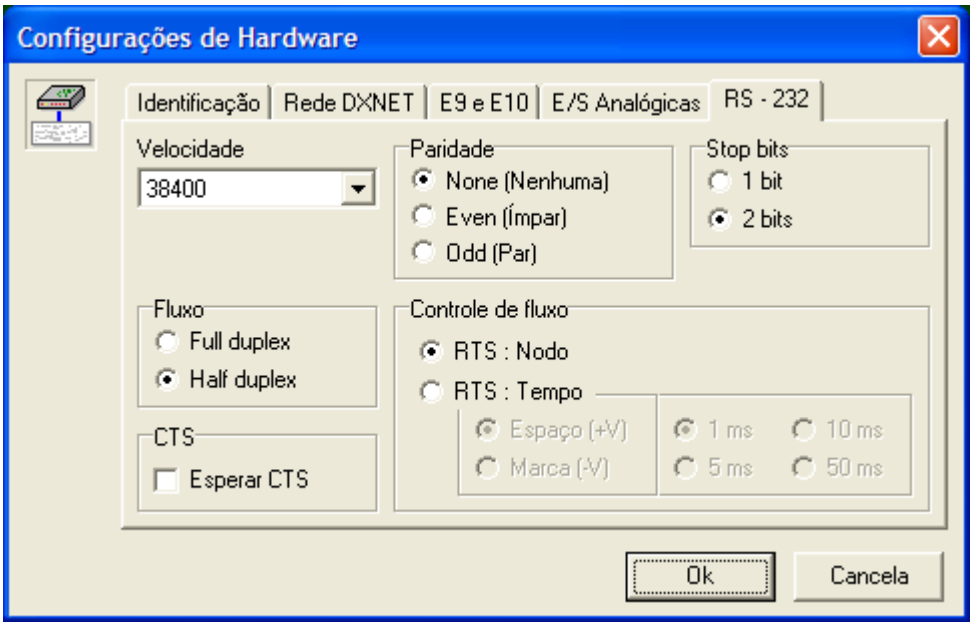
Número de entradas	_E1	_E2	_E3	_E4	_E5	_E6	_E7	_E8
1	E1		E3		E5		E7	
2	E1+E2		E3+E4		E5+E6		E7+E8	
3	E1+E2+E3		E3+E4+E5		E5+E6+E7		E7+E8+E1	
4	E1+E2+E3+E4		E3+E4+E5+E6		E5+E6+E7+E8		E7+E8+E1+E2	
5	E1+E2+E3+...+E5		E3+E4+E5+...+E7		E5+E6+E7+...+E1		E7+E8+E1+...+E3	
6	E1+E2+E3+...+E6		E3+E4+E5+...+E8		E5+E6+E7+...+E2		E7+E8+E1+...+E4	
7	E1+E2+E3+...+E7		E3+E4+E5+...+E1		E5+E6+E7+...+E3		E7+E8+E1+...+E5	
8	E1+E2+E3+...+E8		E3+E4+E5+...+E2		E5+E6+E7+...+E4		E7+E8+E1+...+E6	

Oversampling = 128 (15 bits - leitura x 8)

Número de entradas	_E1	_E2	_E3	_E4	_E5	_E6	_E7	_E8
1	E1				E5			
2	E1+E2				E5+E6			
3	E1+E2+E3				E5+E6+E7			
4	E1+E2+E3+E4				E5+E6+E7+E8			
5	E1+E2+E3+...+E5				E5+E6+E7+...+E1			
6	E1+E2+E3+...+E6				E5+E6+E7+...+E2			
7	E1+E2+E3+...+E7				E5+E6+E7+...+E3			
8	E1+E2+E3+...+E8				E5+E6+E7+...+E4			

Oversampling = 256 (16 bits - leitura x 16)

Número de entradas	_E1	_E2	_E3	_E4	_E5	_E6	_E7	_E8
1	E1							
2	E1+E2							
3	E1+E2+E3							
4	E1+E2+E3+E4							
5	E1+E2+E3+...+E5							
6	E1+E2+E3+...+E6							
7	E1+E2+E3+...+E7							
8	E1+E2+E3+...+E8							



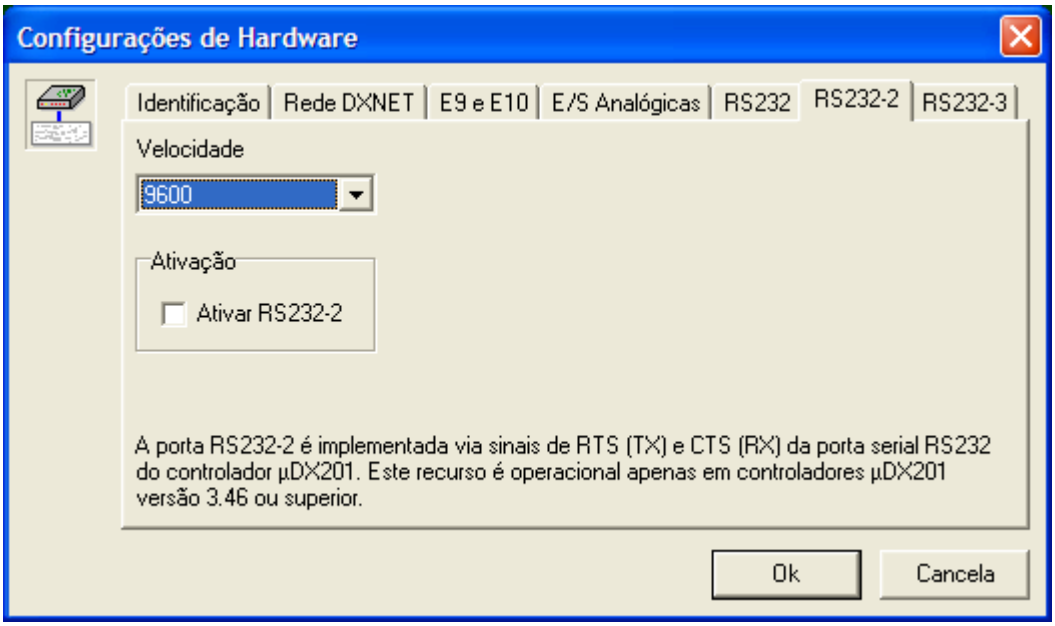
A aba **RS-232** se refere aos parâmetros de comunicação serial via porta RS232 do CLP. Estão disponíveis os seguintes parâmetros:

- **Velocidade (baud rate):** de 110 a 115200 bps (default = 38400).
- **Paridade:** par, ímpar, sem paridade (default = sem paridade).
- **Stop Bit:** 1, 2 (default = 2).
- **Tipo de comunicação:** full-duplex, half-duplex (default = half-duplex).
- **CTS:** aguardar CTS se marcado (default = não aguardar).
- **RTS:** nodo, tempo (default = nodo).

No caso de CTS, deve-se tomar cuidado ao selecionar esta opção. Se ela for selecionada e o programa for transmitido para o µDX201, este passa a testar o sinal de CTS e só irá responder a comunicações via serial RS232 quando este sinal estiver em nível alto. No caso de uso de cabos RS232 com apenas três fios (TX, RX e GND) a comunicação simplesmente cessa, a menos que no conector DB-9 seja ligado o RTS ao CTS (null modem).

O RTS pode ser comandado a partir do programa aplicativo (modo nodo), via bloco RTS, da família RS232. Outra opção é o próprio µDX200 acionar o RTS a cada transmissão via RS232. Neste caso deve-se marcar a opção Tempo, e escolher se ele deve ser acionado como espaço (+V) ou marca (-V) e o tempo de acionamento antes do início da transmissão (de 1 a 50ms).

No caso de uso de baud-rates muito lentos (abaixo de 1200 bps) recomenda-se inibir o monitoramento, pois neste caso ele irá tornar a comunicação com o controlador bastante morosa. Além disso, muitos cabos adaptadores USB-RS232 não funcionam em baud-rates extremamente lentos, como 110 bps. Neste caso é imperativo usar-se uma porta serial RS232 nativa no computador.



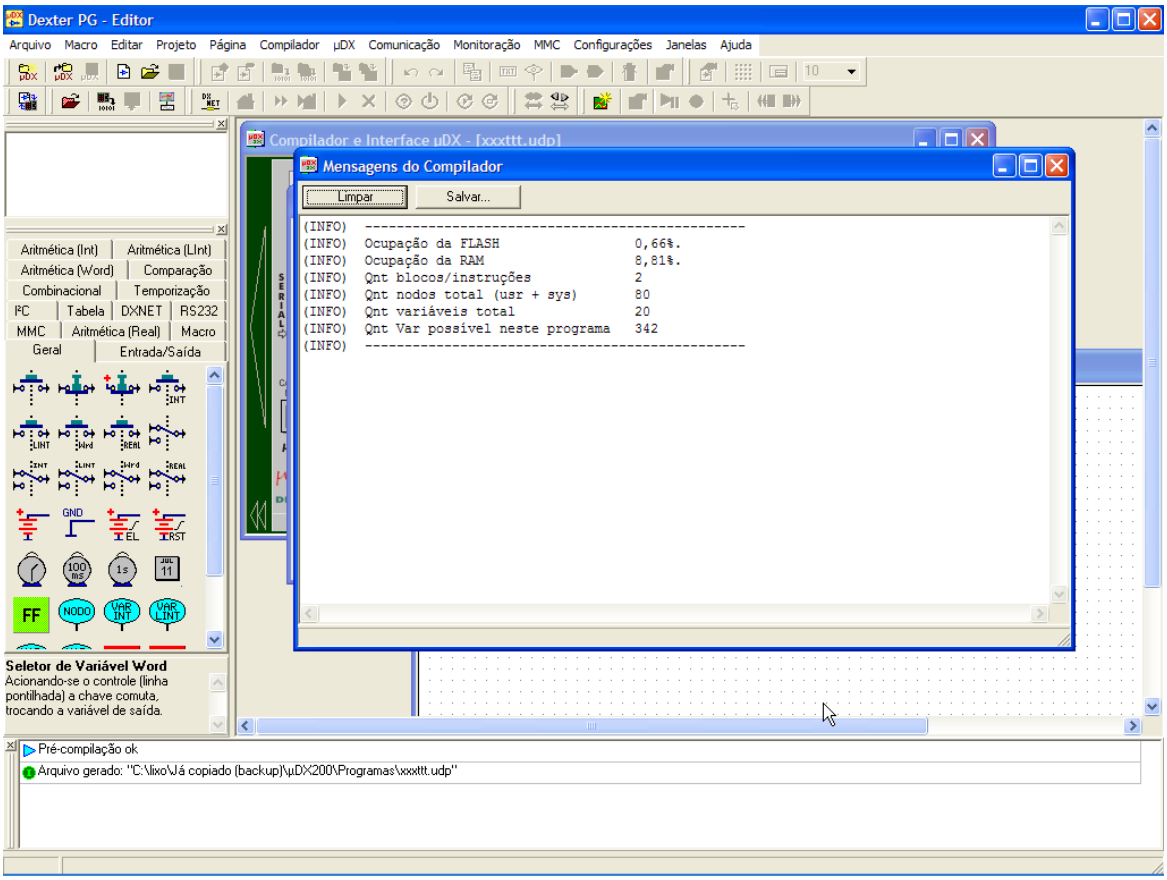
Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais DTR (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal.

Estas portas seriais auxiliares podem ser ativadas via Configuração de Hardware, como mostrado na figura anterior. Também é possível programar o baud-rate (entre 110 e 9600bps). Se as portas seriais são ativadas os sinais RTS e CTS (no caso de RS232-2) ou DTR e DSR (no caso de RS232-3) deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100.

No caso de uso de profundidade de sobreamostragem (oversampling) menor que 256 nas entradas analógicas é desabilitada a porta serial RS232-3. Isso porque os buffers de TX e RX dessa porta são usados para os cálculos de oversampling das entradas analógicas E2 a E8.
No caso de controle de motor a porta serial RS232-2 é inibida automaticamente.

Janela de Log do Compilador

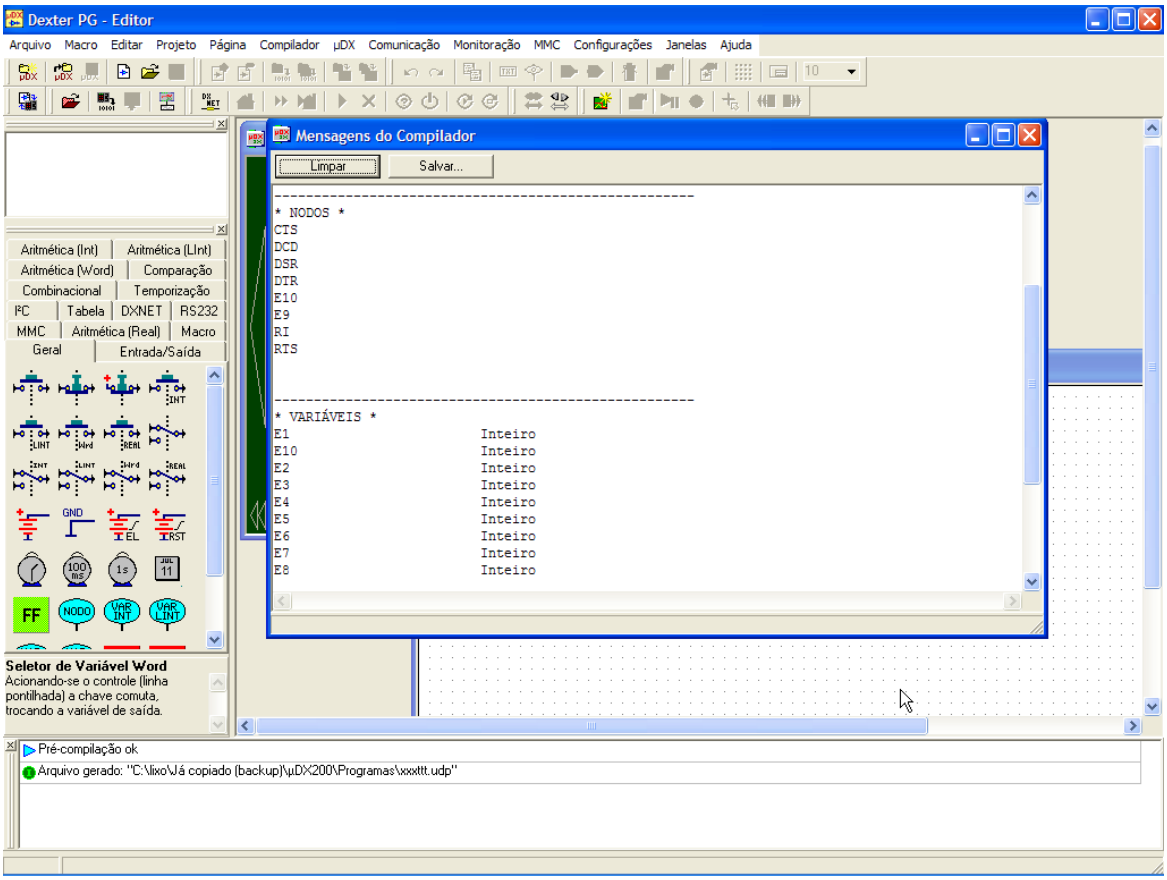
Esta opção permite habilitar a janela de mensagens (log) do Compilador PG. Ela serve para visualizar-se diversas mensagens geradas pelo Compilador durante sua operação, e permite exportar os resultados para um arquivo (permitindo, por exemplo, enviar estes resultados via e-mail para diagnóstico na Dexter).



A tecla **Limpar** zera toda a janela, e a tecla **Salvar...** permite salvar seu conteúdo em arquivo tipo texto (sufixo .txt).

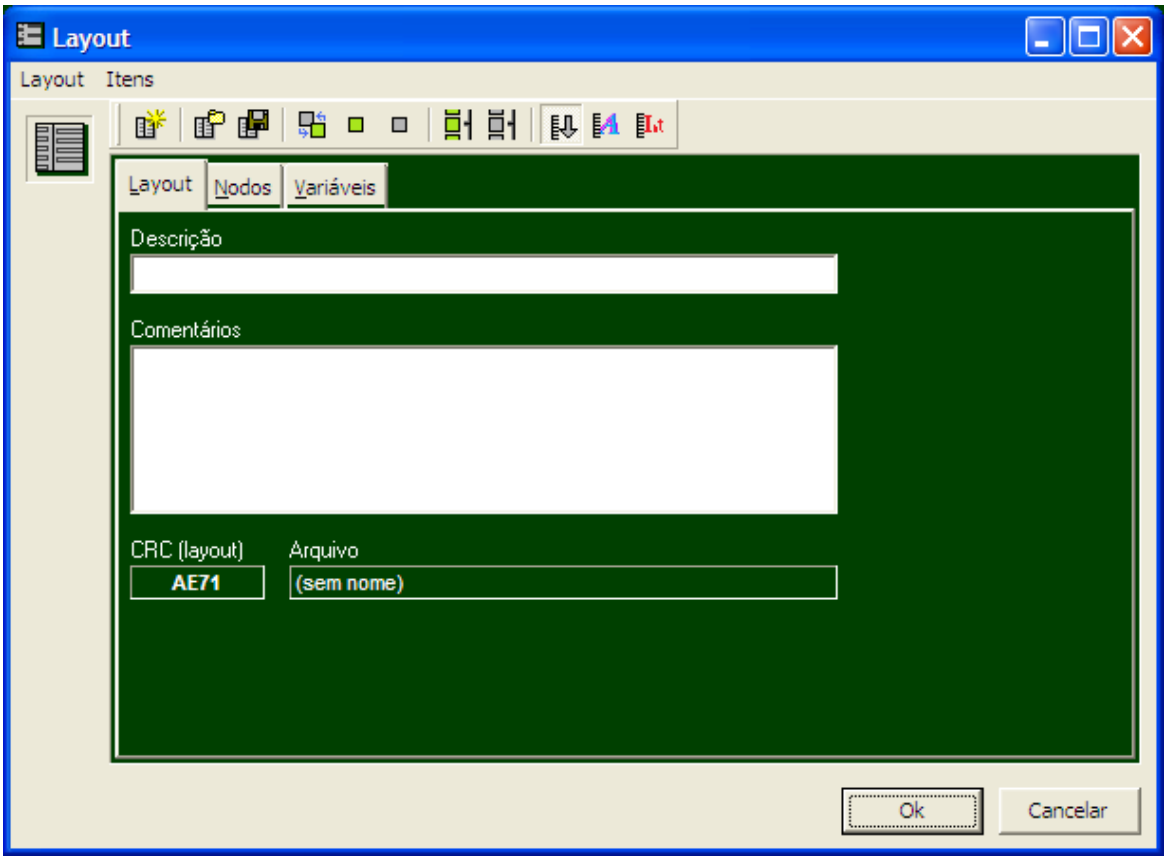
Lista de Variáveis e Nodos

Imprime lista de variáveis e nodos utilizados no programa aplicativo carregado no Compilador PG. É útil para depurar programas, principalmente no que se refere a nodos e variáveis com erros de grafia ou nomes repetidos.



Layout de Monitoração

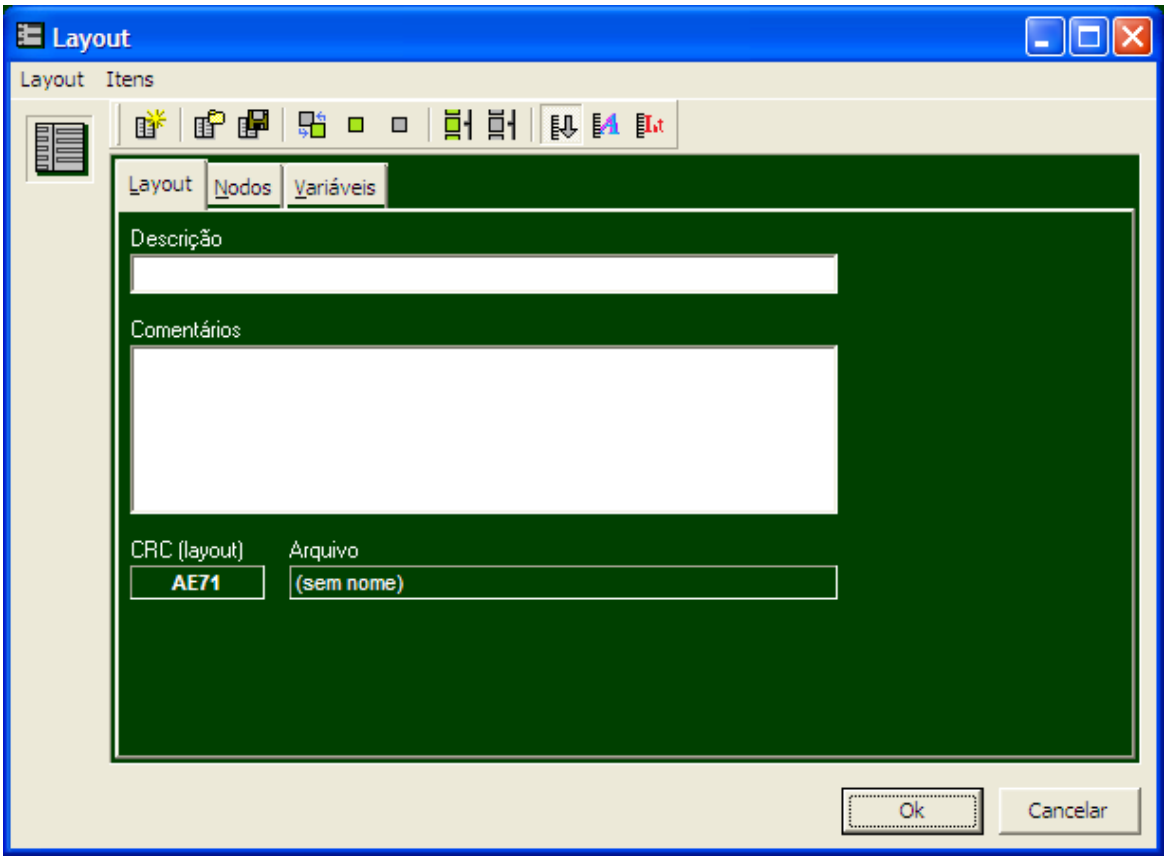
Esta opção permite criar e editar um layout de monitoração. O layout de monitoração é útil principalmente em programas aplicativos extensos, pois permite selecionar quais variáveis e nodos serão monitorados pelo PG, além de classificá-los por ordem alfabética ou por tipo de dado (inteiro, real, etc).



O layout gerado pode ser salvo (arquivo sufixo **.UDG**) para seu uso posterior. Note que o layout salva o CRC do programa aplicativo atualmente carregado no Compilador PG (arquivo **.UDP**). Com isso, é possível verificar se o layout de monitoração carregado é adequado ao programa aplicativo existente no Compilador PG. Existem as seguintes alternativas na opção de Layout de Monitoração:

Novo Layout

Permite criar um novo layout de monitoração. Ao selecionar surge a janela de layout de monitoração, conforme mostrado a seguir:



A primeira aba - **Layout** - possibilita inserir uma descrição para o layout de monitoração, e também comentários a respeito do mesmo. Além disso, mostra o CRC do programa aplicativo atualmente carregado no Compilador PG (e que será salvo junto com o layout) e o nome do arquivo em que o layout foi salvo.

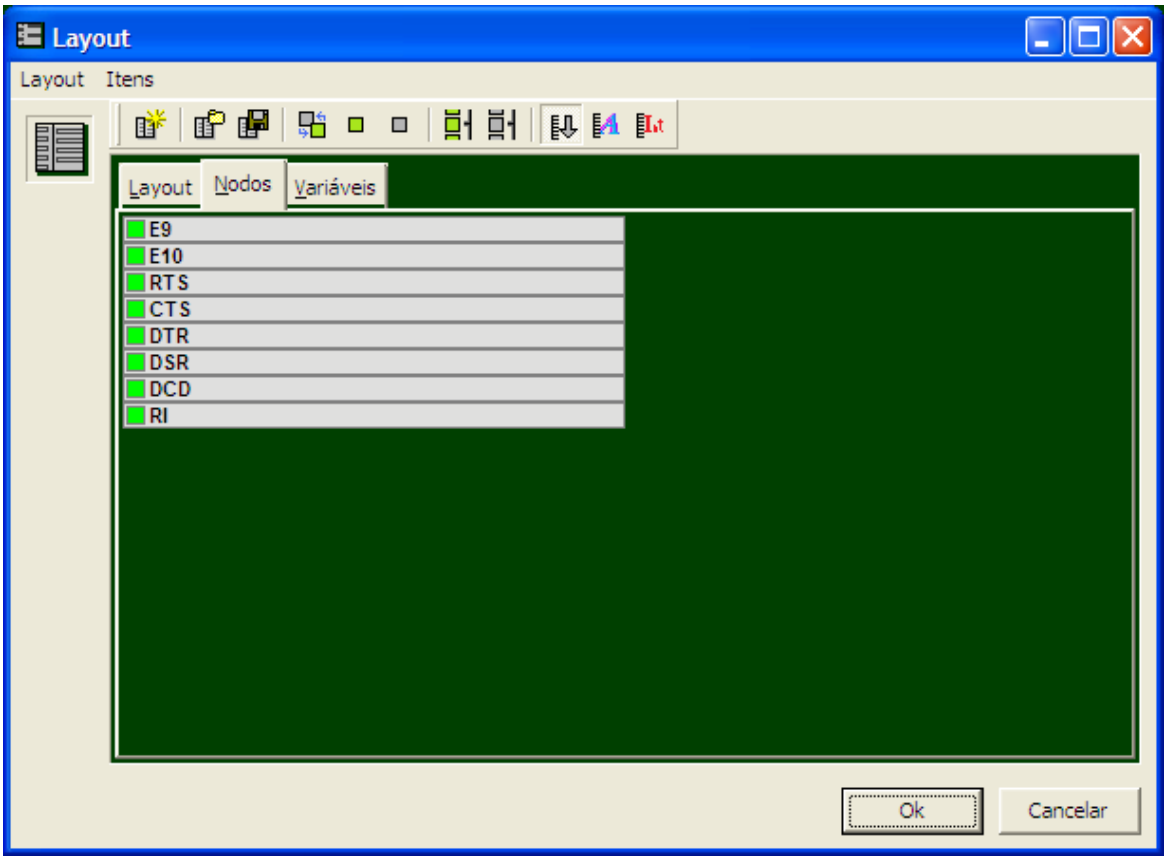
Já os ícones na parte superior desta janela permitem diversas operações. Neste momento vamos nos ater as três primeiras operações, ou seja:

Novo Layout: Zera todas as informações já inseridas no layout, gerando uma janela de layout de monitoração limpa.

Carrega Layout: Permite carregar um layout de monitoração previamente salvo em disco (arquivos sufixo **.UDG**).

Salva Layout: Salva o layout de monitoração em disco (arquivos sufixo **.UDG**).

A segunda aba - **Nodos** - possibilita selecionar quais nodos do programa aplicativo carregado no Compilador PG devem ser monitorados. Todos os itens marcados em verde serão monitorados. Já os itens em cinza serão excluídos, aliviando o processo de monitoração.



Os ícones pertinentes existentes na parte superior da janela são:

Alterna Estado do Item: alterna o item selecionado entre visível e invisível.

Item Visível: torna o item selecionado visível.

Item Invisível: torna o item selecionado invisível.

Todos Visíveis: torna todos os itens visíveis.

Todos Invisíveis: torna todos os itens invisíveis.

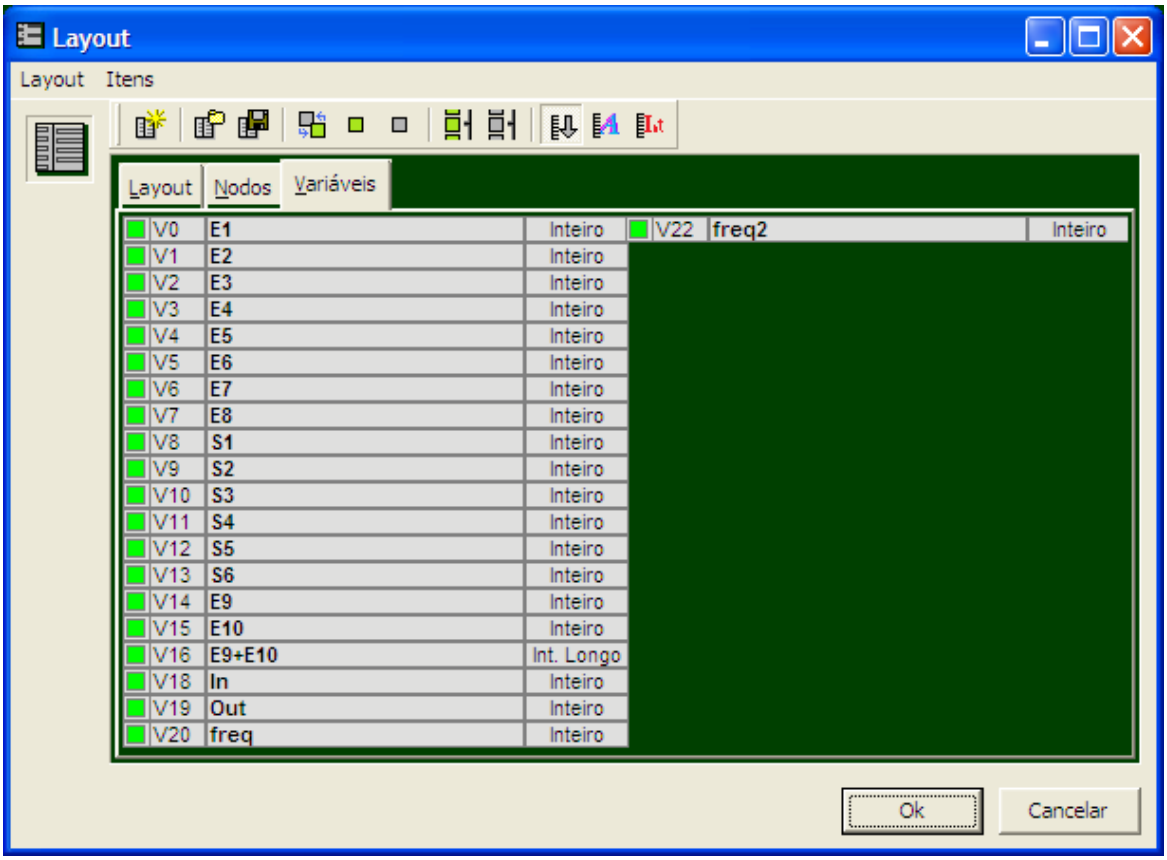
Ordem Padrão: ordena itens pela ordem padrão (variáveis absolutas em ordem crescente).

Ordem Alfabética: ordena itens por ordem alfabética crescente.

Ordem de Tipo: ordena itens por tipo (inteiro, inteiro longo, real, word)

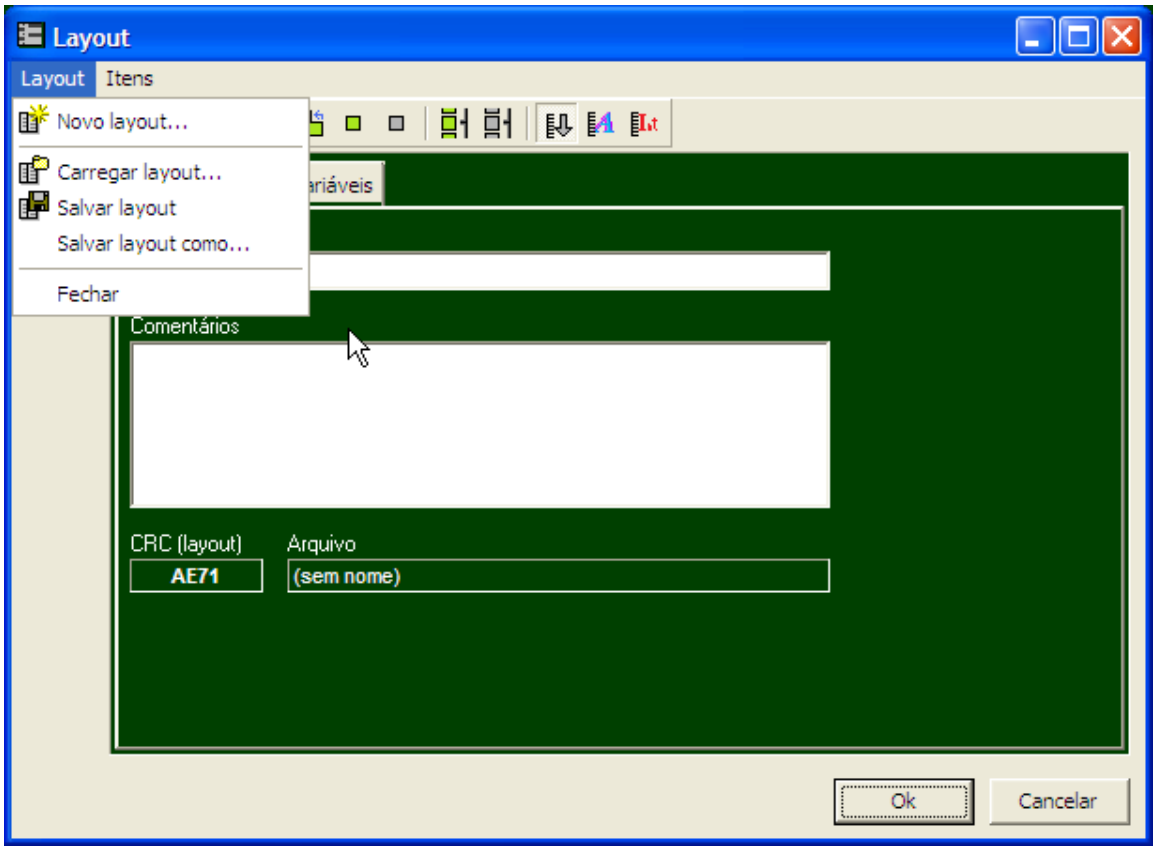
Note que a ordenação alfabética e por tipo é efetuada, separadamente, nas variáveis e nodos de sistema, e nas variáveis e nodos do programa aplicativo. Este procedimento garante que as variáveis e nodos de sistema estejam sempre agrupadas no início da lista.

A terceira e última aba - **Variáveis** - apresenta a lista de todas as variáveis utilizadas no programa aplicativo, assim como as variáveis de sistema do µDX200 (V0 até V17):



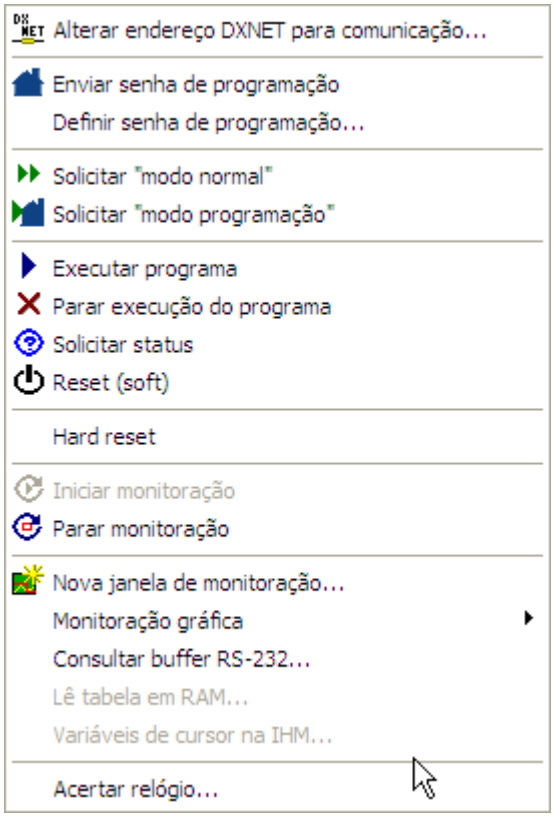
Dica: também é possível alternar entre visível e invisível qualquer item da lista de variáveis ou nodos simplesmente clicando duas vezes com o mouse sobre o item desejado.

Além dos ícones a janela de Layout de Monitoração apresenta dois menus pop-down - **Layout** e **Itens**, com funções idênticas às dos ícones e outras funções de funcionalidade óbvia, como Salvar como...:


Layout
Layout
Itens
Novo layout...
Carregar layout...
Salvar layout
Salvar layout como...
Fechar
ariáveis
Comentários
CRC (layout)
AE71
Arquivo
(sem nome)
Ok
Cancelar

Menu µDX

O Menu µDX permite interagir com o controlador programável, ordenando a execução do programa aplicativo, efetuando reset, solicitando status, etc. Também permite iniciar monitoração gráfica (existe um menu pop-down específico para monitoração gráfica que será descrito adiante, mas muitas de suas funcionalidades estão disponíveis aqui também).



Alterar endereço DXNET para comunicação...

Esta opção permite editar o endereço DXNET acessado pelo Compilador PG. Note que o endereço 0 (zero) sempre acessa o µDX200 conectado ao microcomputador via porta serial, endereço TCP/IP, ou Modem. Ou seja, independentemente do endereço DXNET do µDX200, se for programado no Compilador PG para que ele utilize o endereço 0 (zero) a comunicação será feita com o dispositivo conectado ao computador, sem tentativa de comunicação via rede DXNET. O endereço 0 (zero) é o endereço padrão para o PG quando este é instalado, e só deve ser modificado caso existam mais controladores µDX200 ligados via rede DXNET. Para modificar este dado é possível, além da escolha desta opção no menu µDX, apontar com o mouse para o endereço DXNET (**DXNET 0**) que aparece abaixo da representação do µDX200 e clicar duas vezes com a tecla esquerda do mouse. Irá surgir a seguinte janela:

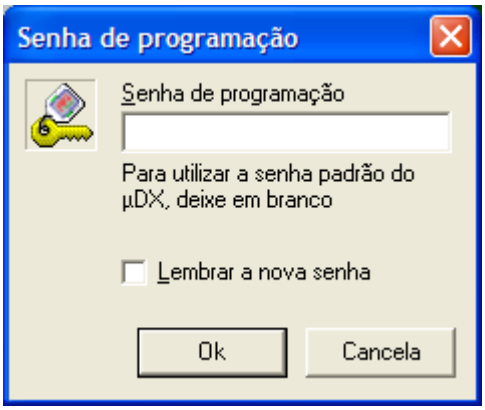
Digite o valor desejado e clique em **Ok**. Se o endereço especificado não coincidir com o do µDX200 conectado ao microcomputador via porta serial, endereço TCP/IP ou Modem, a mensagem será replicada para a rede DXNET a procura de um controlador com este endereço.

Enviar senha de programação

Transmite a senha de programação, forçando o controlador µDX200 a passar para o modo de comunicação **Programação**. Existem dois modos de comunicação para a porta serial: **Modo Normal** e **Modo Programação**. No **Modo Normal** o programa aplicativo assume o controle da porta serial, e fica inoperante o protocolo de comunicação nativo do µDX200. Já no **Modo Programação** o protocolo de comunicação nativo (usado pelo Compilador PG) é ativado, e o programa aplicativo perde o controle da porta serial. O controlador passa para o **Modo Normal** automaticamente, quando o programa aplicativo está sendo executado, após um minuto sem comunicação serial RS232. Já para retornar ao **Modo Comunicação** é preciso transmitir a senha de programação. O led de Execução existente no controlador programável fica constantemente ligado se o programa aplicativo está rodando e o CLP está em **Modo Programação**. Já se o programa aplicativo está rodando e o CLP está em **Modo Normal** este led fica piscando.

Definir senha de programação...

Permite definir a senha de programação a ser usada pelo Compilador PG. Note que, para programar uma senha no µDX200, é necessário digitar esta senha na Configuração de Hardware de um programa aplicativo e transmití-lo para o controlador programável. A senha padrão é **[uDX200**.



Repare na opção de lembrar a nova senha. Assim, é possível manter a nova senha de programação no Compilador PG, evitando ter que digitá-la a cada execução do PG. Se o campo de senha for deixado em branco será utilizada a senha padrão.

Atenção: Muito cuidado ao modificar esta senha! Se for transmitido um programa aplicativo para o µDX200 com outra senha que não a senha padrão ele passará a exigir a nova senha para permitir novas programações. Sem conhecê-la será impossível reprogramar o controlador programável! Trata-se de uma segurança, em especial quando o CLP está acessível remotamente (via endereço TCP/IP ou via Modem), mas deve-se tomar o devido cuidado para anotar a senha em local acessível.

Solicitar "Modo Normal"

Comuta o µDX200 para **Modo Normal**, transferindo o controle da porta serial do controlador para o programa aplicativo. Naturalmente, o Compilador não mais lê dados do µDX200.

Para retornar ao **Modo Programação** pressione a tecla  na tela principal do Compilador PG (Enviar Senha de Programação). Note que o µDX200 comuta automaticamente para **Modo Normal** caso não haja comunicação válida durante um minuto no modo Programação (desde que o programa aplicativo no µDX200 esteja sendo executado). Enquanto o Compilador estiver lendo dados do µDX200 isso não acontece, pois o software fica constantemente lendo dados, como status, nodos e variáveis. No µDX200, o estado de Normal ou Programação para a porta serial RS232 é indicado pelo led de Exec: ligado significa programa rodando e **Modo Programação**; piscando significa programa rodando e **Modo Normal**. No caso do led Exec estar desligado significa que o programa está parado e, neste caso, apenas o estado Programação é válido (já que o modo Normal é quando o programa aplicativo comanda a porta serial e, neste caso, o programa aplicativo está parado).

Solicitar "Modo Programação"

Permite comutar um µDX200 conectado via rede DXNET para o **Modo Programação**. Ou seja, sua porta serial comuta do **Modo Normal** para o **Modo Programação**.

Executar Programa

Ao transmitir o programa aplicativo para o controlador µDX200 este assume o estado parado, por segurança. Para executar o programa é necessário enviar este comando. Note que quando o programa está rodando o led **Exec.** existente no µDX200 é acionado. Caso ele esteja ligado constantemente significa que o programa aplicativo está rodando e a porta serial do µDX200 está no modo Programação. Já se este led estiver piscando significa que o programa aplicativo está rodando e a porta serial está no modo Normal. O modo Normal permite que o programa aplicativo utilize a porta serial. O µDX200 comuta automaticamente para o modo Normal caso não haja nenhuma comunicação válida no modo Programação durante um minuto.

Parar Execução do Programa

Esta opção permite parar o programa aplicativo do controlador µDX200. Note que o led **Exec.** existente no painel do controlador é desligado.

Solicitar Status

Este comando solicita envio do status (várias informações, como versão de firmware, temperatura interna, tensão de alimentação, etc.) do controlador µDX200. Esta informação é necessária para iniciar monitoramento do CLP.

Reset (soft)

Efetua um reset no µDX200. O programa aplicativo continua no estado que estava anteriormente ao reset (parado ou executando), o relógio de tempo real é preservado, mas as demais variáveis são zeradas.

Hard Reset

Efetua um reset geral no µDX200. O programa aplicativo não pára caso esteja sendo executado, mas todas as variáveis são zeradas. Já o relógio de tempo real não é afetado. Se o programa aplicativo estiver parado a porta serial retorna para a configuração padrão (38400bps, 8 bits, sem paridade, 2 stop bits).

Atualizar Firmware para o µDX

Esta opção foi incorporada ao PG versão 2.3.0.1 ou superior, e permite atualizar o firmware do controlador µDX201 conectado ao computador. Note que somente controladores com firmware igual ou superior a 3.61 possuem essa capacidade. A atualização de firmware não destrói o programa aplicativo existente no µDX201, e também preserva os dados de calibração das entradas e saídas analógicas. Entretanto, é um processo rápido (cerca de 40 segundos) mas que não pode ser interrompido, sob pena de tornar o controlador µDX201 inoperante e exigir seu envio para a Dexter para manutenção. Além disso, ao final do processo é efetuado um hard reset do controlador e, com isso, todas as variáveis e nodos são reinicializados em zero.

Atenção: O processo de atualização de firmware não pode ser interrompido, sob pena de tornar o controlador µDX201 inoperante e exigir seu envio para a Dexter para manutenção. Apenas controladores v3.61 ou superior suportam essa atualização.

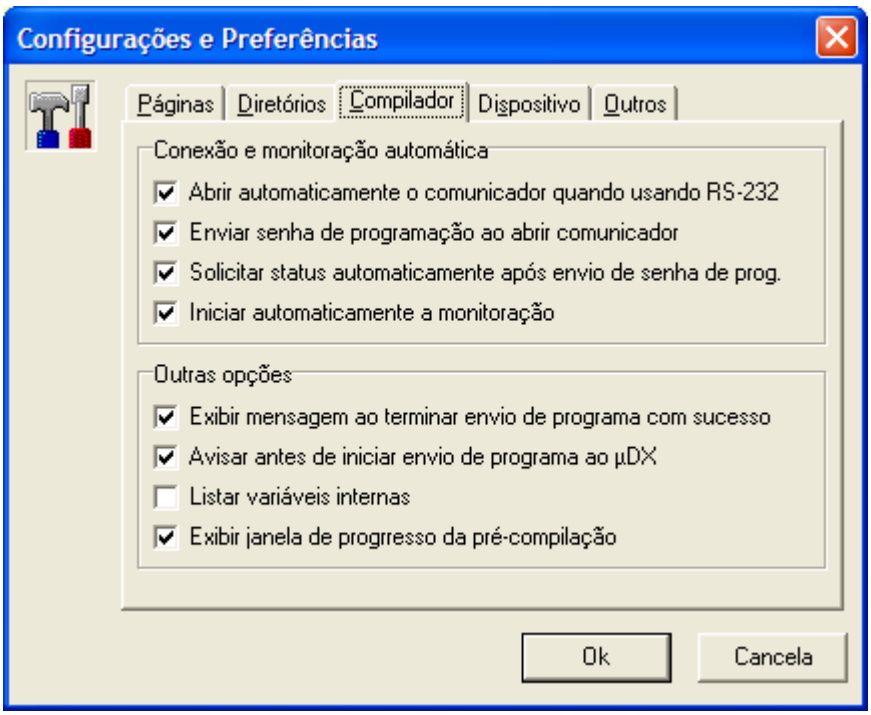
Iniciar Monitoração

Inicia a monitoração de status, nodos e variáveis do µDX200. Note que o dado **Refresh**, existente na tela do Compilador PG (aba Status+CPU+Programa), indica o tempo total para uma leitura completa de todos os dados. Evidentemente, quanto maior o número de variáveis e nodos a serem monitorados mais lenta fica a monitoração. Também influencia este tempo o meio de comunicação empregado (porta serial, endereço TCP/IP, ou ainda Modem). Caso exista programa aplicativo carregado no Compilador PG (sufixo .udp), e este seja idêntico ao carregado no µDX200 sob monitoração (CRC remoto coincide com CRC local), todas as variáveis e nodos empregados no programa aplicativo são monitorados.

Para iniciar monitoração é preciso enviar senha de programação (de forma que o µDX200 esteja em modo programação) e solicitar status. Caso em **Configurações** → **Configurações e Preferências** → **Compilador** estejam marcados os itens:

[x] Abrir automaticamente o comunicador quando usando RS-232
[x] Enviar senha de programação ao abrir comunicador.
[x] Solicitar status automaticamente após envio de senha de progr.
[x] Iniciar automaticamente a monitoração

todo o procedimento é automático (esta é a condição default ao instalar o PG).

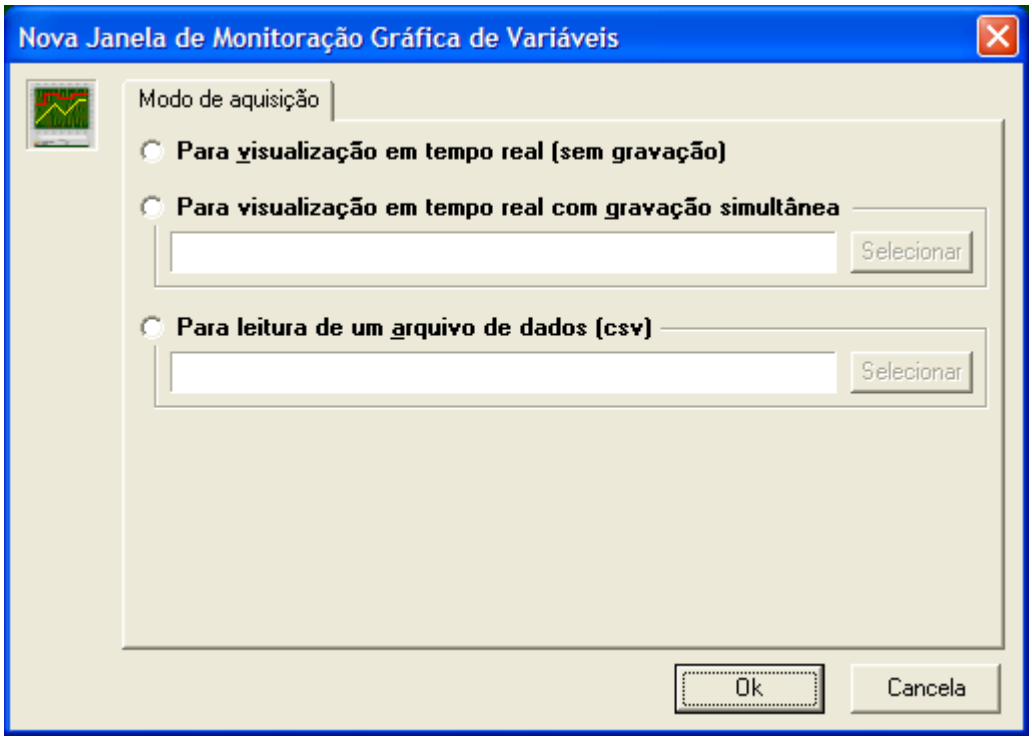


Parar Monitoração

Interrompe a monitoração. Isso pode ser importante para aliviar o processamento do computador, caso se queira usá-lo para outros fins.

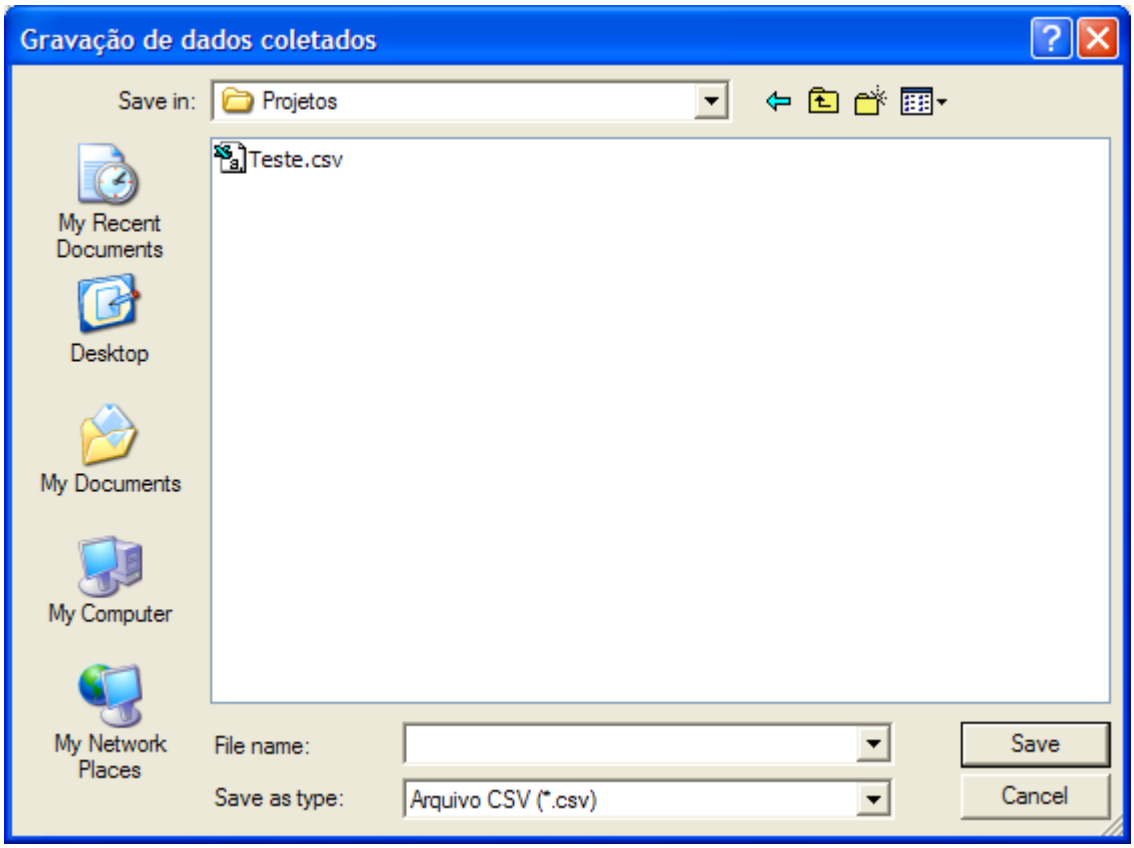
Nova Janela de Monitoração...

Esta opção cria uma nova janela de monitoração gráfica. Com isso, é possível monitorar graficamente variáveis do Controlador µDX200, e salvar os valores lidos em arquivos compatíveis com planilhas eletrônicas como Excel. Inicialmente é criada uma janela com as seguintes possibilidades:

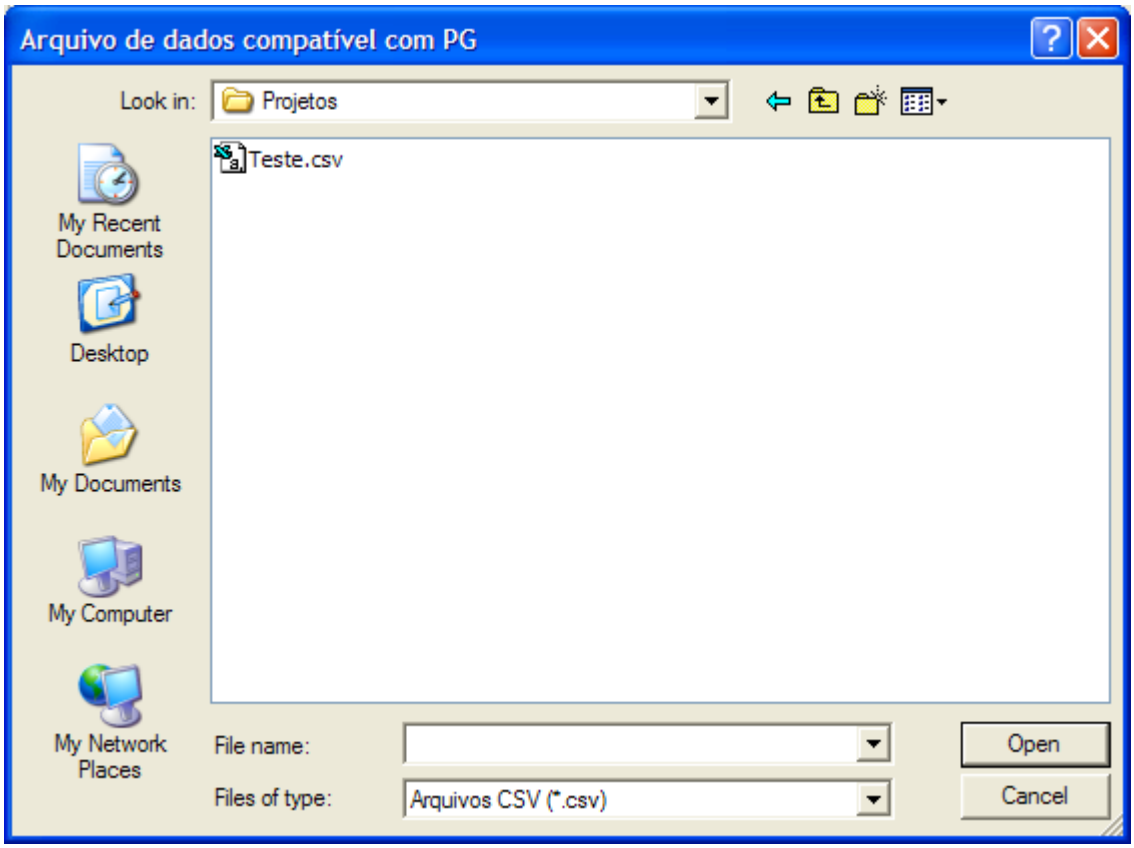


O **Modo de Aquisição** pode ser de três tipos:

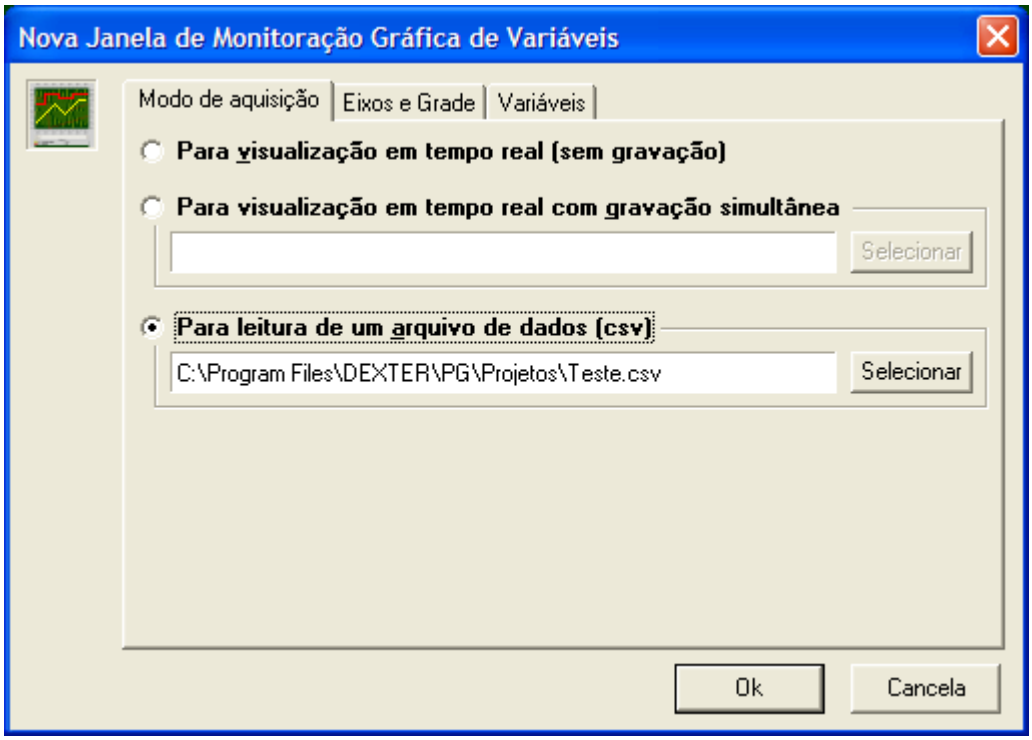
- **Para visualização em tempo real (sem gravação):** neste caso os dados irão aparecer na tela em tempo real, mas não serão gravados. Evidentemente, neste caso está descartado poder navegar para áreas anteriores do gráfico (ao completar a tela disponível todo o gráfico passa a rodar horizontalmente, descartando os dados mais antigos).
- **Para visualização em tempo real com gravação simultânea:** idêntico ao caso anterior, mas os dados lidos são também gravados em arquivo de tipo CSV, passível de ser lido posteriormente tanto no próprio PG como em uma planilha eletrônica como Excel, por exemplo. Ao selecionar esta opção surge uma janela para seleção do nome do arquivo a ser gerado com os dados:



- **Para leitura de um arquivo de dados (CSV):** permite ler um arquivo do tipo CSV gravado anteriormente via opção anterior, ou lido de um cartão MMC gravado com dados do µDX200. Neste caso surge uma janela para seleção do arquivo a ser lido:

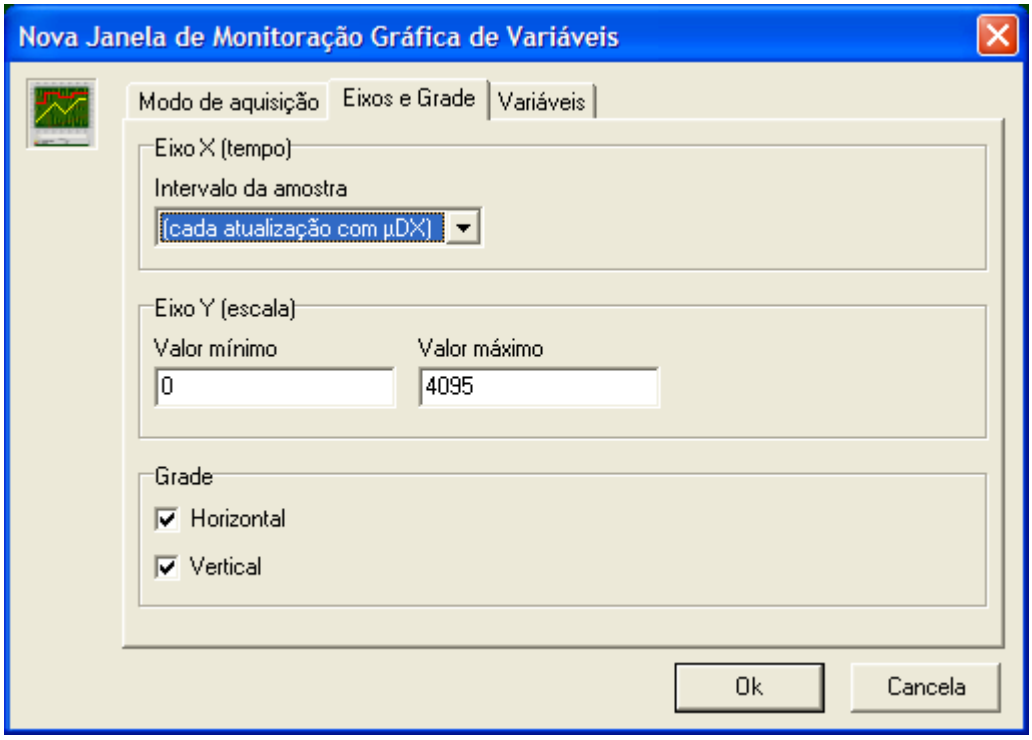


Ao selecionar uma das opções acima duas outras abas são ativadas na janela **Nova janela de Monitoração Gráfica de Variáveis**. São elas **Eixos e Grade** e **Variáveis**.

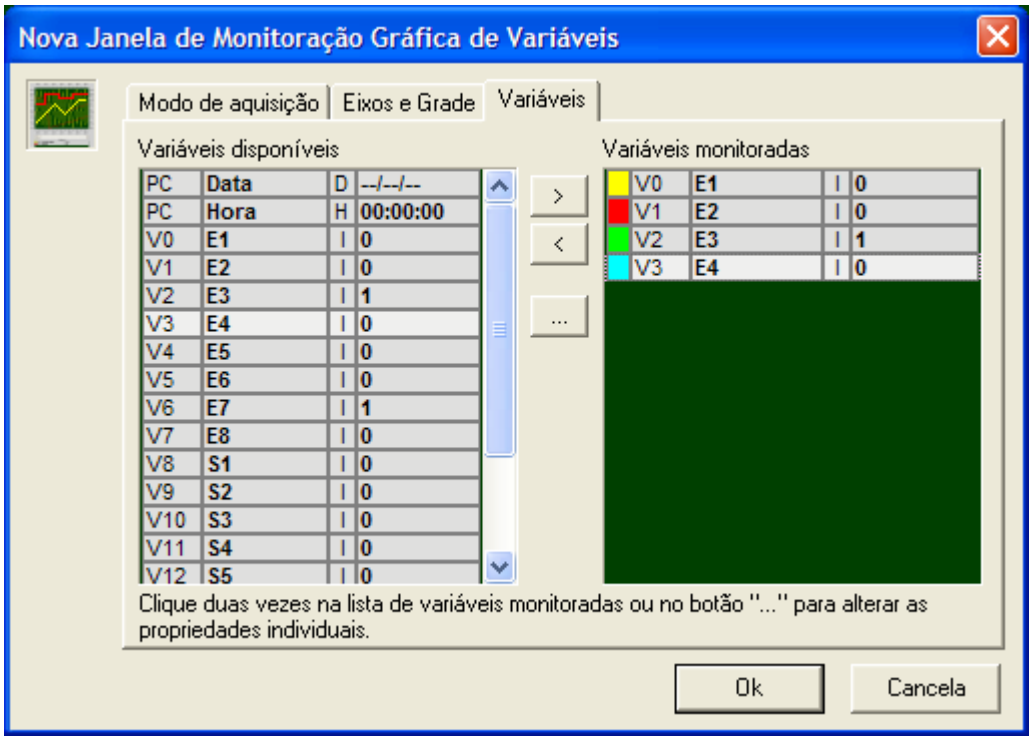


Na aba **Eixos e Grade** é possível selecionar o valor mínimo e máximo do eixo Y do gráfico (o

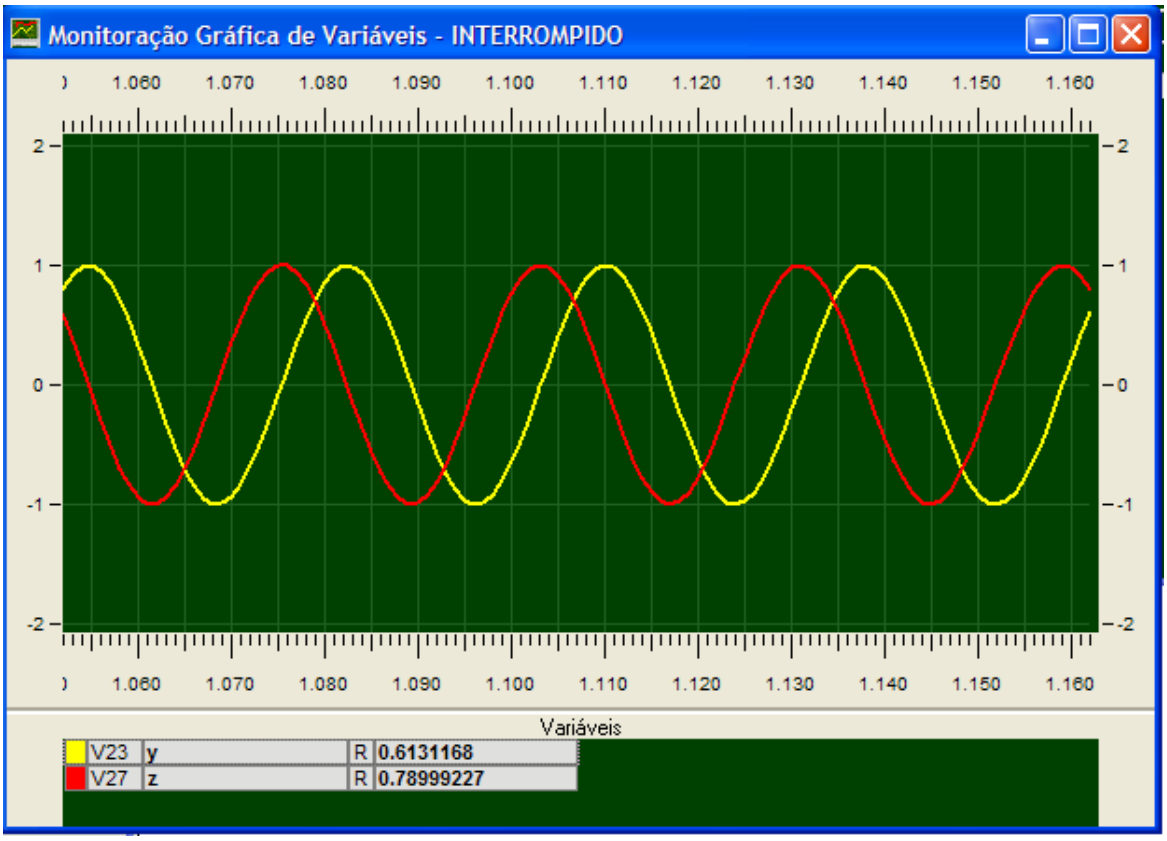
valor default é 0 e 4095), o intervalo de amostra do eixo X (o default é a cada atualização com µDX, ou seja, a velocidade máxima de monitoramento do PG). Por fim, também pode-se inibir a grade horizontal ou vertical do gráfico.



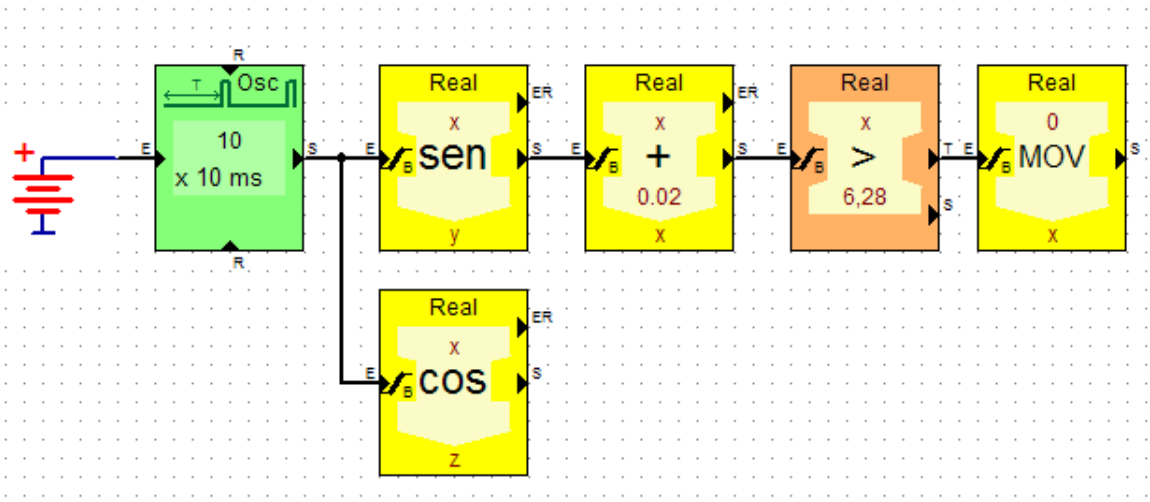
Na aba **Variáveis** seleciona-se quais variáveis serão plotadas no gráfico. Para que apareçam variáveis do µDX nas possibilidades de seleção desta janela é preciso que o Compilador PG esteja com a monitoração ativada (se não somente data e hora do microcomputador PC estarão disponíveis). Para selecionar as variáveis a serem plotadas basta clicar duas vezes sobre as mesmas, ou selecioná-las e clicar no botão ›. O botão ‹ permite eliminar variáveis da plotagem. O botão ... possibilita editar o nome da variável e sua cor.



Caso o programa aplicativo existente no µDX200 coincida com o programa aplicativo carregado no Compilador PG (os CRCs remoto e local devem ser idênticos) estarão disponíveis para plotagem todas as variáveis usadas no programa do CLP. Ao clicar em Ok é gerada a janela de monitoração gráfica. A janela de monitoração gráfica a seguir foi gerada com captura das variáveis y e z (seno e cosseno, respectivamente) a cada segundo:



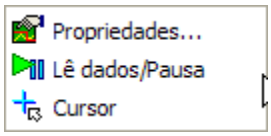
O programa aplicativo que gerou estas variáveis é:



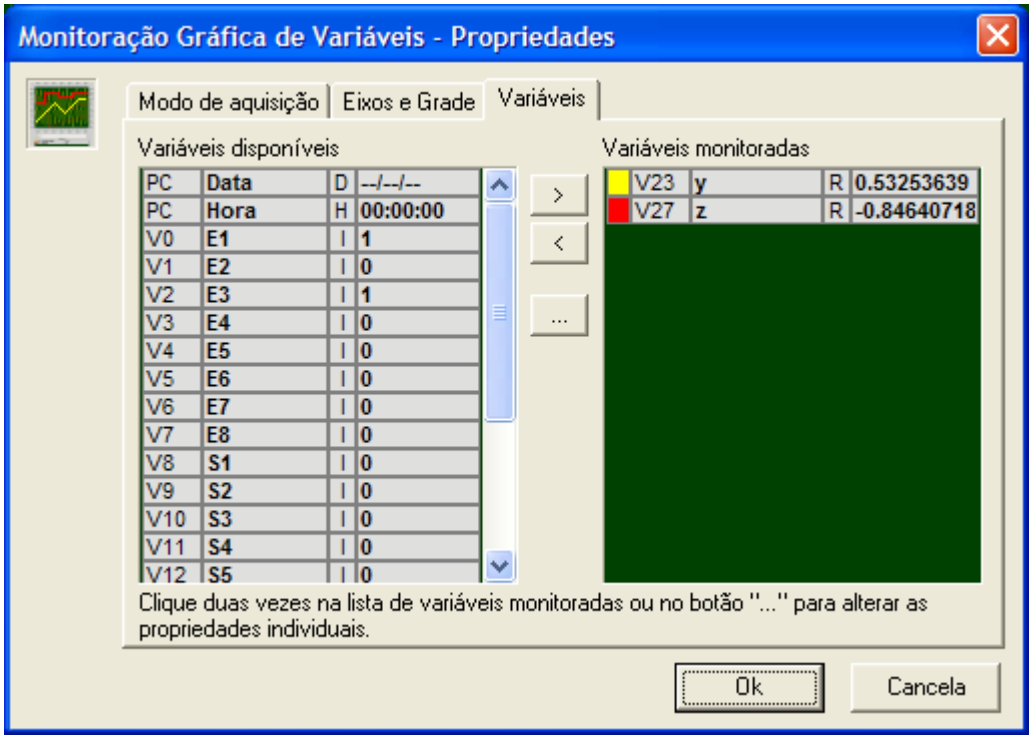
Exemplo de Programa Aplicativo: **Sen_Cos.dxg** (fonte); **Sen_Cos.udp** (compilado).

Monitoração Gráfica

Permite selecionar diversas funções para a monitoração gráfica de variáveis. São elas:

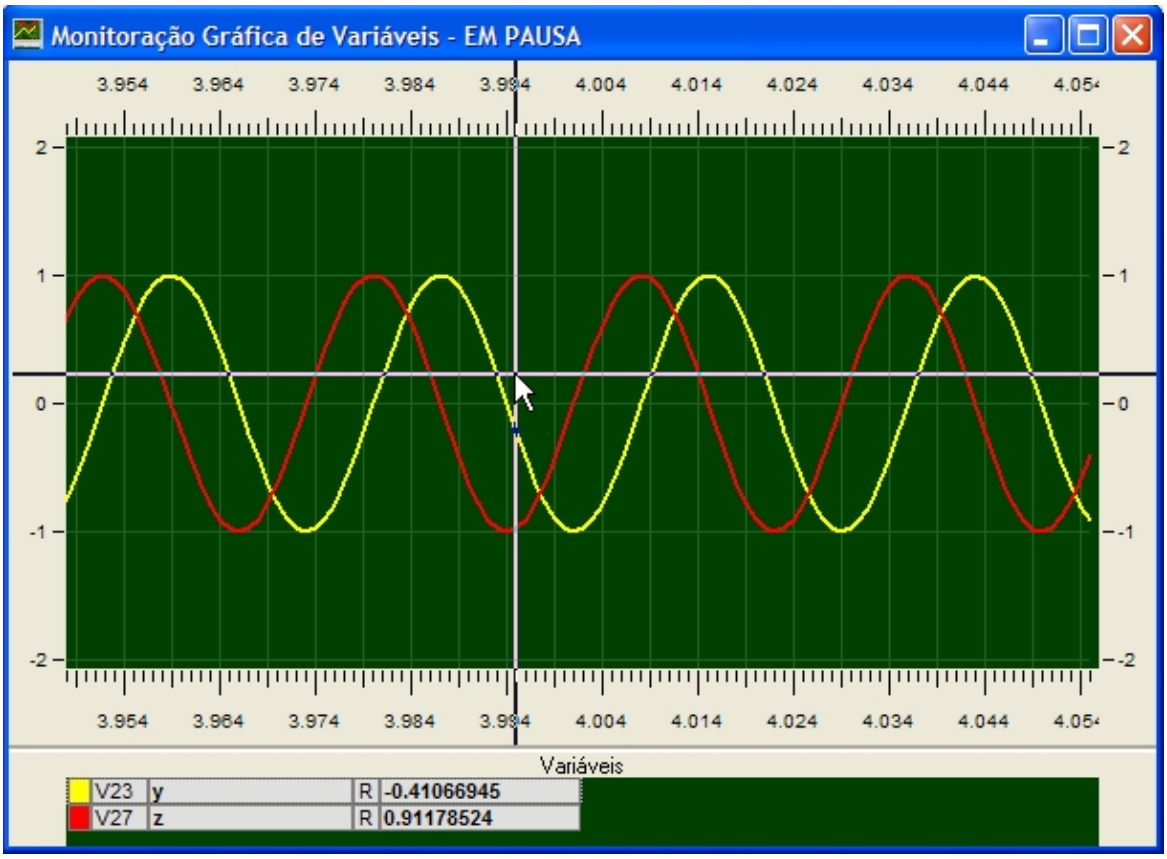


Propriedades: Acessa os parâmetros do gráfico, como escala X e Y, variáveis monitoradas, etc., permitindo modificá-los.



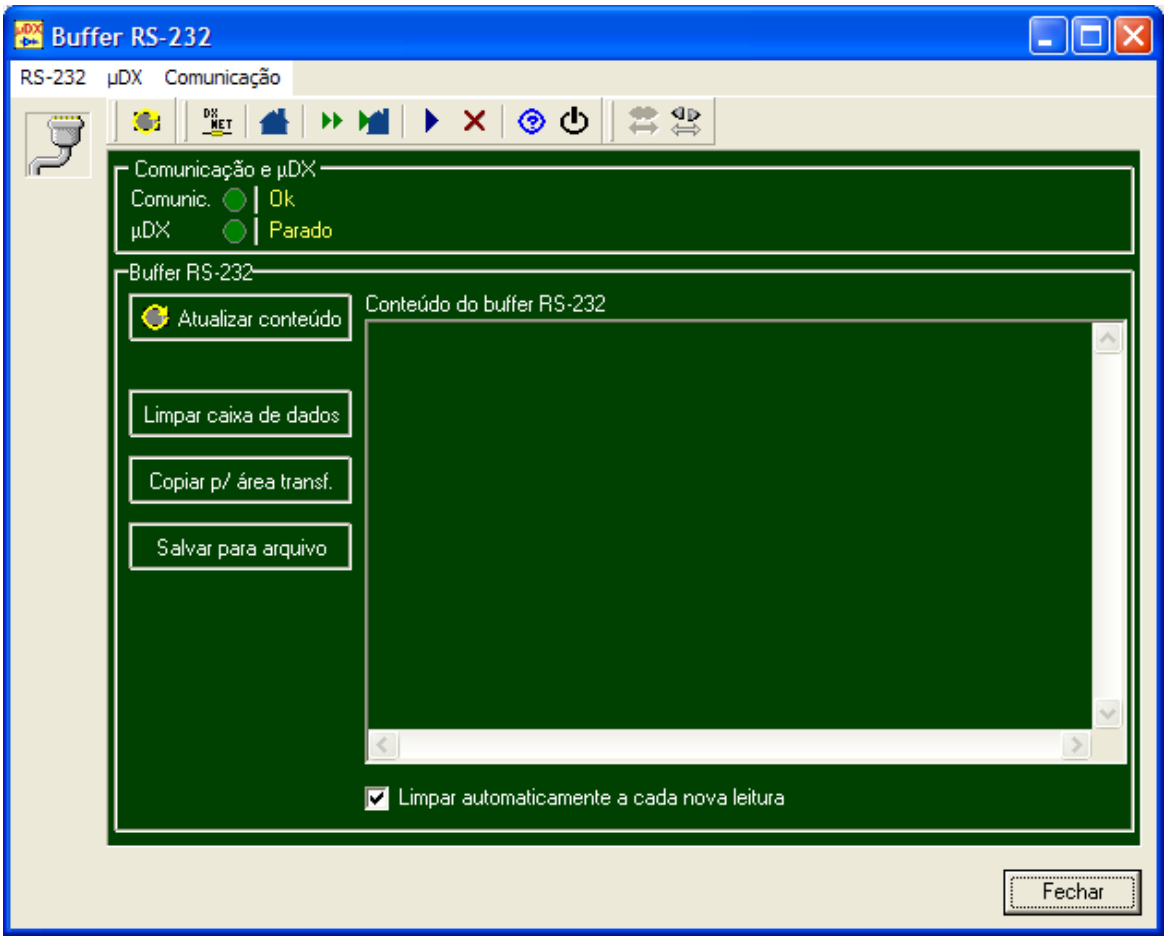
Lê dados/Pausa: interrompe ou reinicia a leitura de dados do gráfico.

Cursor: cria um cursor sobre o gráfico, permitindo a leitura dos valores das variáveis em qualquer ponto do mesmo.


Monitoração Gráfica de Variáveis - EM PAUSA
3.954
3.964
3.974
3.984
3.994
4.004
4.014
4.024
4.034
4.044
2
1
0
-1
-2
Variáveis
V23 y R -0.41066945
V27 z R 0.91178524

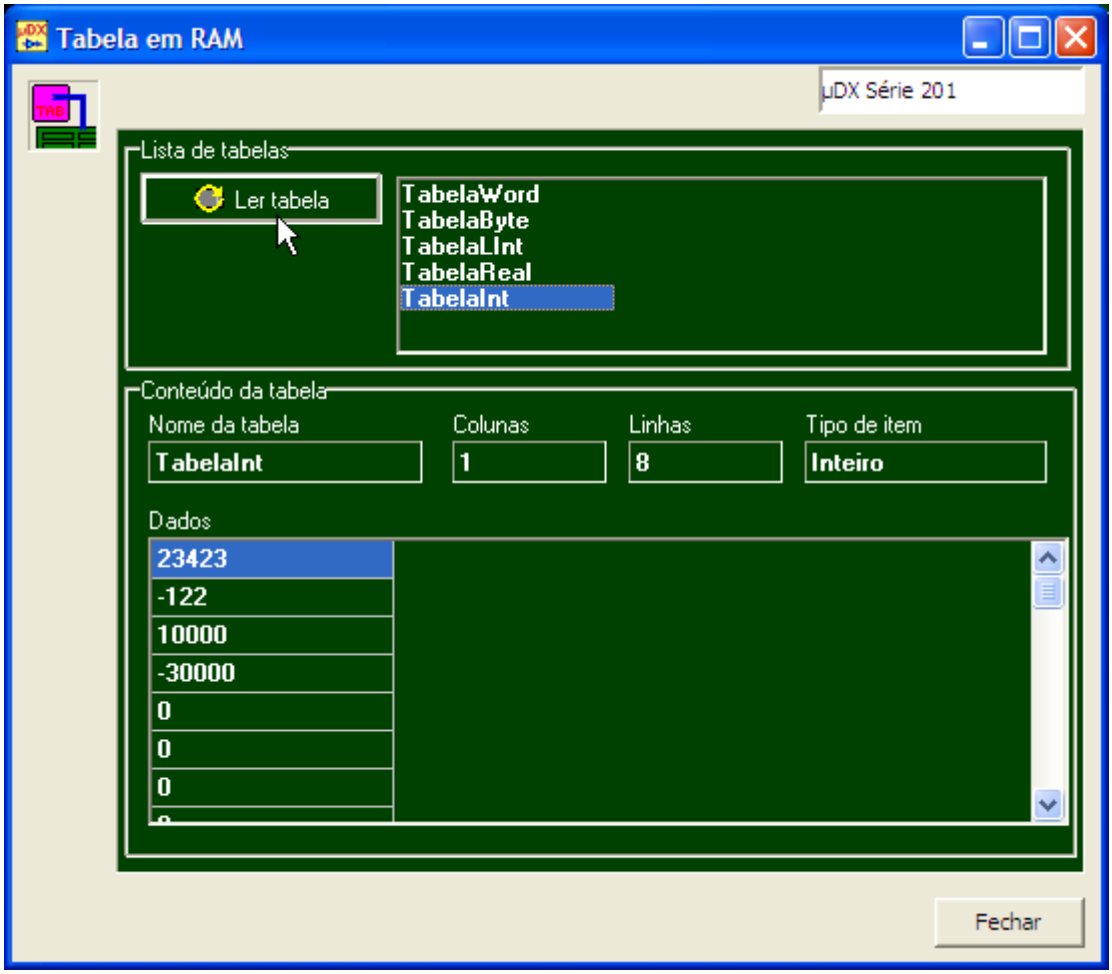
Consultar Buffer RS-232...

Abre uma janela para leitura do buffer de serial RS-232 do Controlador µDX200. Isso é útil em programas que utilizem a porta serial para comunicação com outros dispositivos. Evidentemente, neste caso o µDX200 deve estar sendo monitorado via rede DXNET (através de outro µDX200 ligado a rede DXNET).



Lê tabela em RAM...

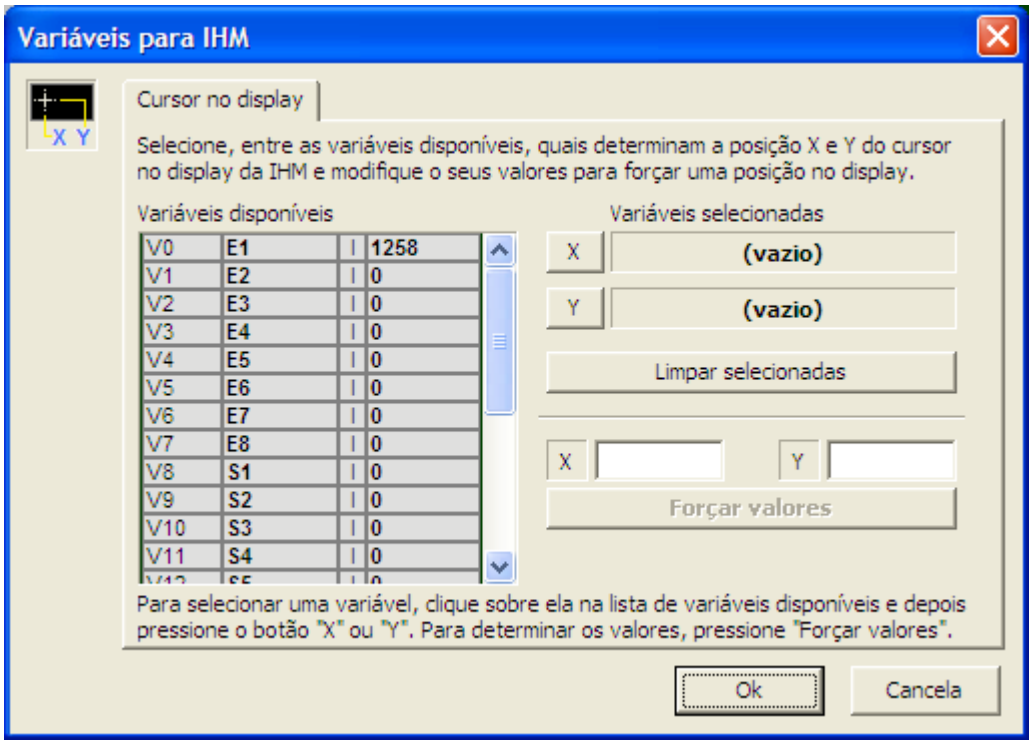
Abre uma janela para leitura do conteúdo de tabelas RAM (que permitem leitura e escrita) existentes no programa aplicativo. Para examinar o conteúdo de determinada tabela basta clicar com o botão esquerdo do mouse sobre o nome da tabela para selecionar a tabela a ser lida e pressionar o botão **[Ler tabela]**:



Note que no canto superior direito é especificado em qual dispositivo está sendo lida a tabela RAM. No caso se trata de µDX201.

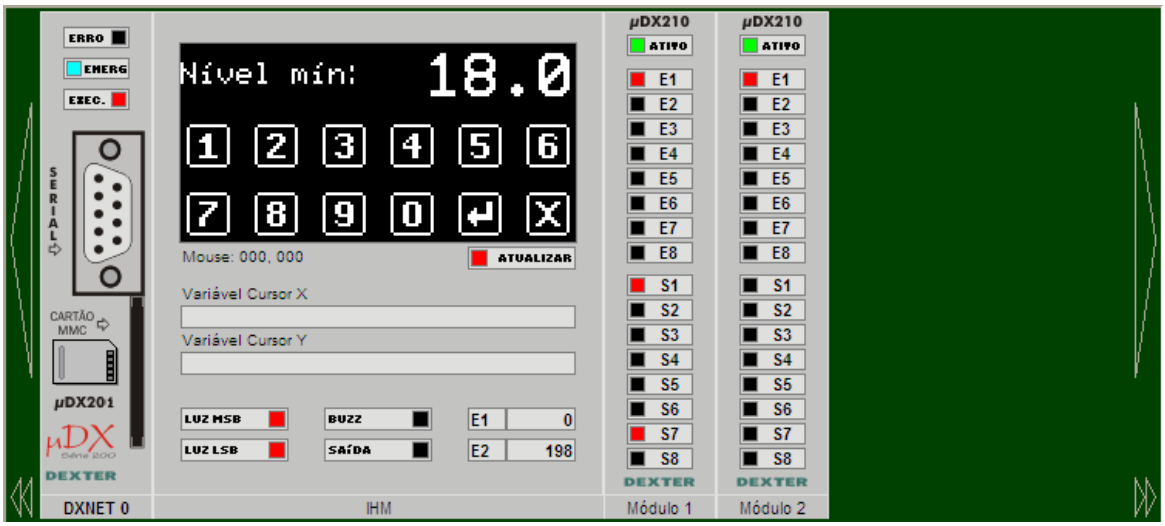
Variáveis de cursor na IHM...

Esta janela possibilita especificar quais variáveis do programa aplicativo são usadas para leitura da posição (x,y) do sensor touchscreen existente no display da Interface Homem/Máquina (IHM) para µDX201. Note que este dispositivo somente é funcional conectado a controlador µDX201. Ele não é permitido em controladores µDX200. A conexão é feita via conector de expansão do µDX201 (onde podem ser conectados até 32 Expansões de Entradas/Saídas µDX210 também). Em função de limitações deste tipo de conexão a IHM pode distar, no máximo, de 2 metros do controlador µDX201.



A especificação das variáveis responsáveis pelas coordenadas (x,y) do display da IHM permite clicar duas vezes com o botão esquerdo sobre a representação do display no Compilador PG para forçar estas variáveis para o valor correspondente. Assim, pode-se facilmente simular o toque em determinado ponto do sensor touchscreen. Também é permitido digitar estes valores nos campos correspondentes e forçar as variáveis ao pressionar **[Forçar valores]**.

A janela acima também está disponível se for efetuado um duplo clique com o botão esquerdo do mouse sobre as variáveis de cursor X e Y existentes na representação de IHM do Compilador PG (esta representação somente aparece se se tratar de µDX201 e o programa aplicativo do mesmo fizer uso de blocos de IHM):



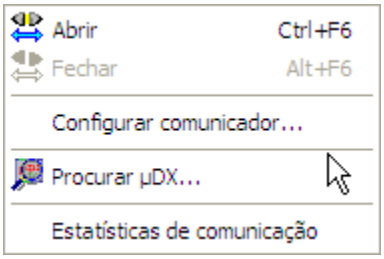
Note que as coordenadas no display são crescentes da esquerda para a direita (eixo x) e de cima para baixo (eixo y). A resolução é de 128 pontos na vertical e 64 pontos na vertical:

Acertar Relógio...

Permite acertar o relógio de tempo real do controlador µDX200. É possível atribuir a data e hora existentes no relógio interno do computador ligado ao µDX200 ao relógio de tempo real desse. Note que o µDX200 possui um calendário interno, inclusive com previsão para ano bissexto. Também permite introduzir data e hora manualmente no relógio interno do µDX200. Esta opção pode ser útil para teste de programas com comportamento dependente do relógio de tempo real do controlador.

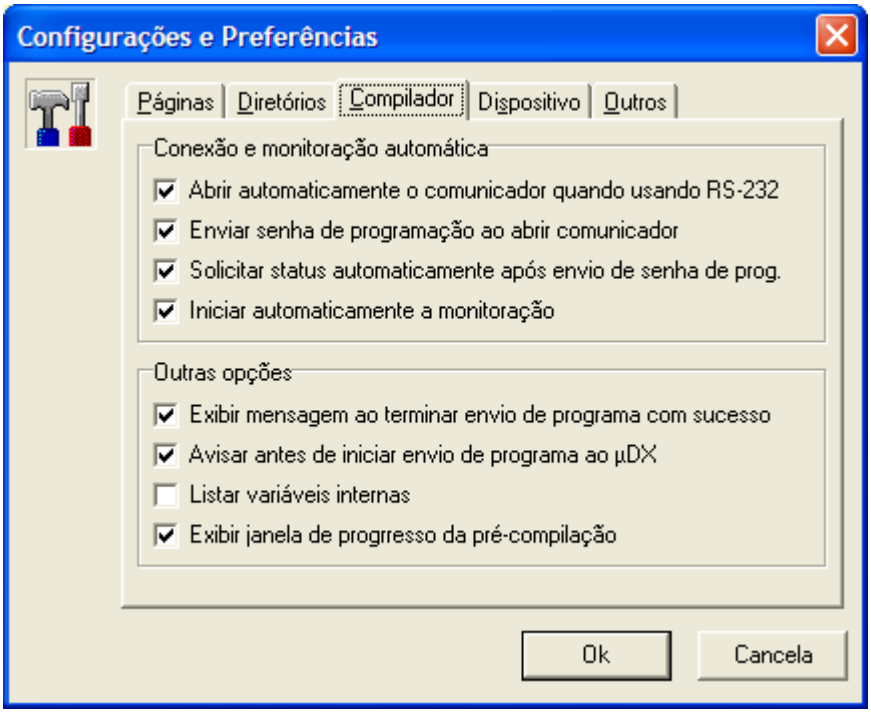
Menu Comunicação

O menu Comunicação concentra as opções de comunicação com o controlador µDX200:



Abrir (Ctrl+F6)

Abre o comunicador. Com isso, o Compilador PG aloca o recurso para si (porta serial no caso de comunicação serial; endereço TCP/IP no caso de comunicação via rede Ethernet; ou modem no caso de comunicação via linha telefônica). Caso o comunicador esteja utilizando comunicação serial e as opções automáticas estejam habilitadas no menu **Configurações → Configurações e Preferências → Compilador** é enviada a senha para o µDX200 (de forma a força-lo para o modo programação), é solicitado o status do CLP e é iniciada a monitoração.



Fechar (Alt+F6)

Fecha o comunicador, liberando o recurso para que outros programas possam utilizá-lo (porta serial no caso de comunicação serial; endereço TCP/IP no caso de comunicação via rede Ethernet; ou modem no caso de comunicação via linha telefônica).

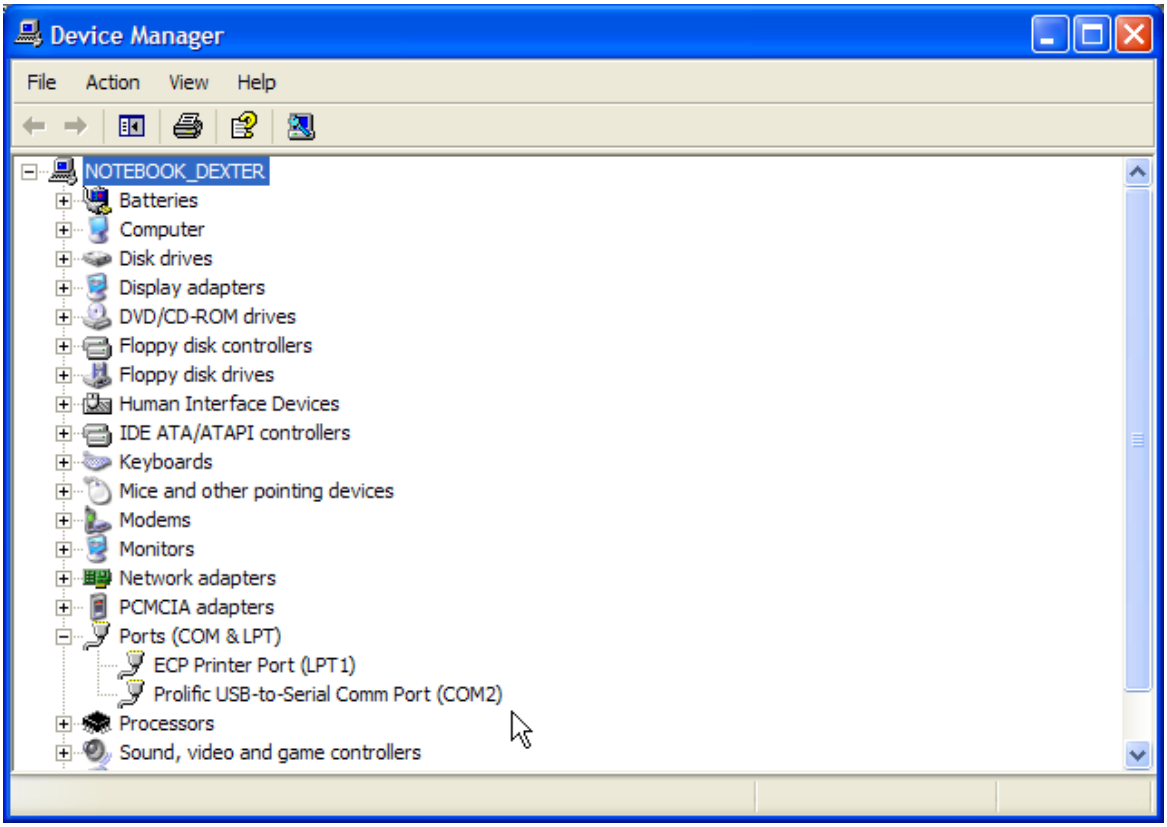
Configurar Comunicador...

A **Configuração do Comunicador** possui quatro abas. A primeira, **Comunicador**, permite escolher qual a forma de comunicação com o controlador µDX200. As opções são comunicação serial RS-232, comunicação via rede Ethernet, ou ainda via Modem. Nesta aba também existe um campo para determinar o timeout mínimo de comunicação (o default é 500ms). Este timeout é usado caso os timeouts programados para cada tipo de comunicação sejam menores, ou seja, determina o menor tempo de espera de respostas à perguntas feitas pelo PG.

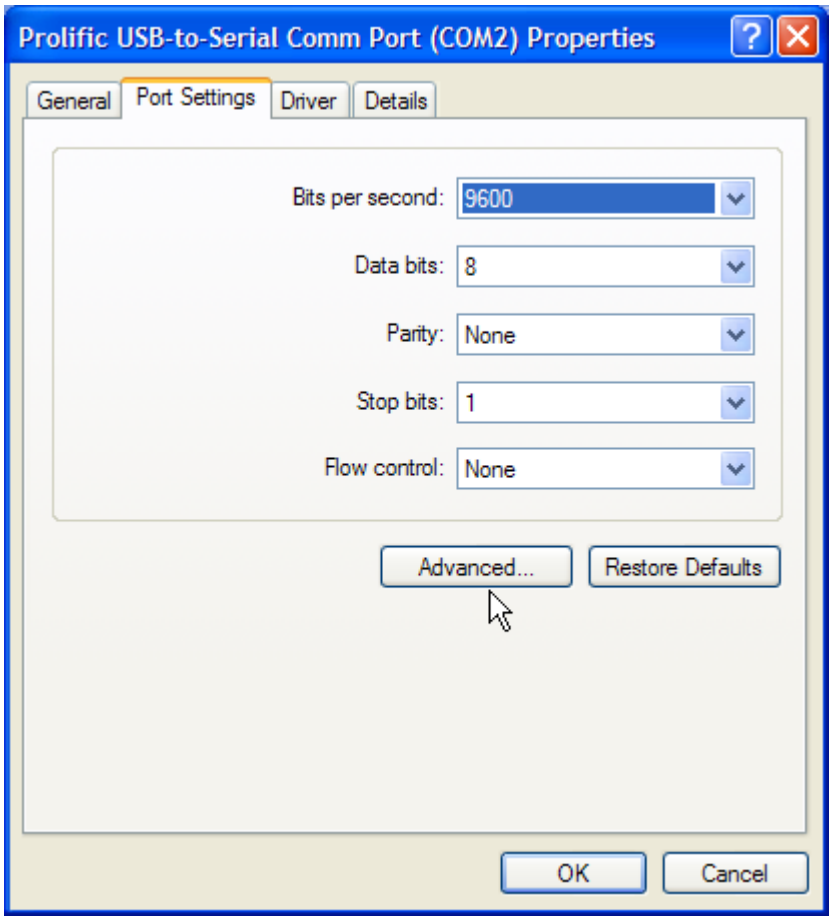
A aba **RS-232** configura a comunicação serial com o µDX200. O configuração default do controlador é 38400 bps, sem paridade, 8 bits de dados, 2 stop bits. Existe um campo para timeout da comunicação RS-232. Este timeout será usado para este tipo de comunicação caso ele seja superior ao timeout mínimo geral (programado na aba **Comunicador**).

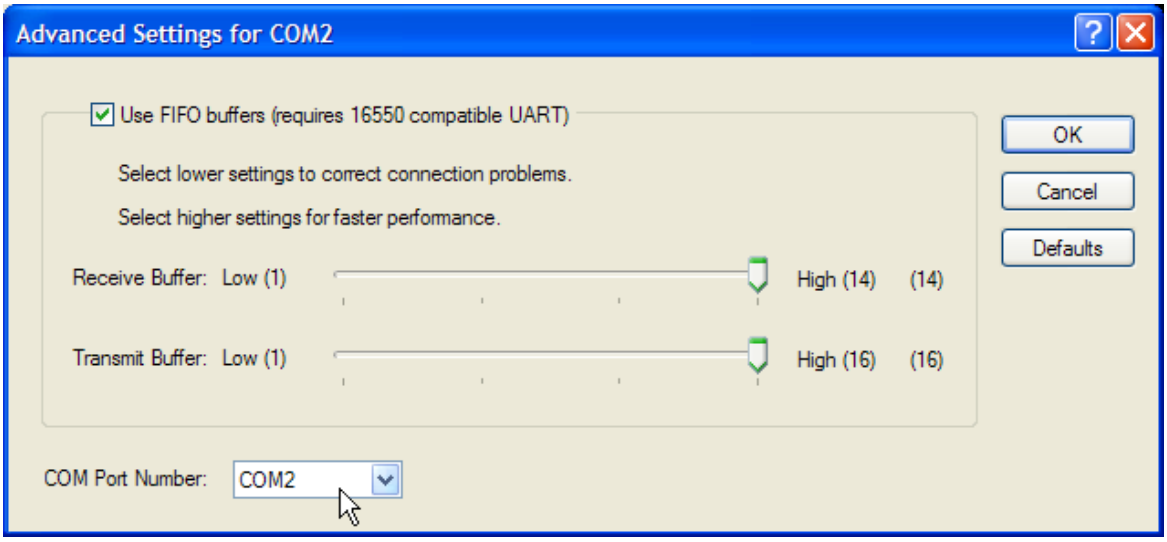
Atenção: caso seu computador não possua porta serial é possível utilizar um cabo adaptador USB↔RS-232. A Dexter comercializa este tipo de cabo, devidamente testado com o µDX200. Solicite um orçamento. De forma geral, todos os cabos adaptadores USB↔RS-232 do mercado devem funcionar sem problemas com o controlador µDX200. Entretanto, em velocidades extremamente baixas, como 110 bps, cabos adaptadores USB↔RS-232 normalmente não funcionam corretamente. Nestes casos é preciso usar uma porta serial RS232 nativa no computador.

Acompanha o cabo USB↔RS-232 um CD com o driver para Windows. Ele deve ser instalado para o correto funcionamento do dispositivo (com alguns cabos isso é desnecessário, e o Windows reconhece o cabo automaticamente). Para saber qual a porta de comunicação que foi gerada ao conectar o cabo adaptador USB↔RS-232 vá em **Painel de Controle** → **Sistema** → **Hardware** → **Gerenciador de Dispositivos** → **Portas (COM & LPT)** e verifique qual a COM gerada:



Caso se queira modificar o valor da COM clique duas vezes sobre o driver para porta serial no Gerenciador de Dispositivos (Device Manager) para abrir as propriedades da porta serial e selecione a opção **Avançado** na aba **Configurações de Porta**:



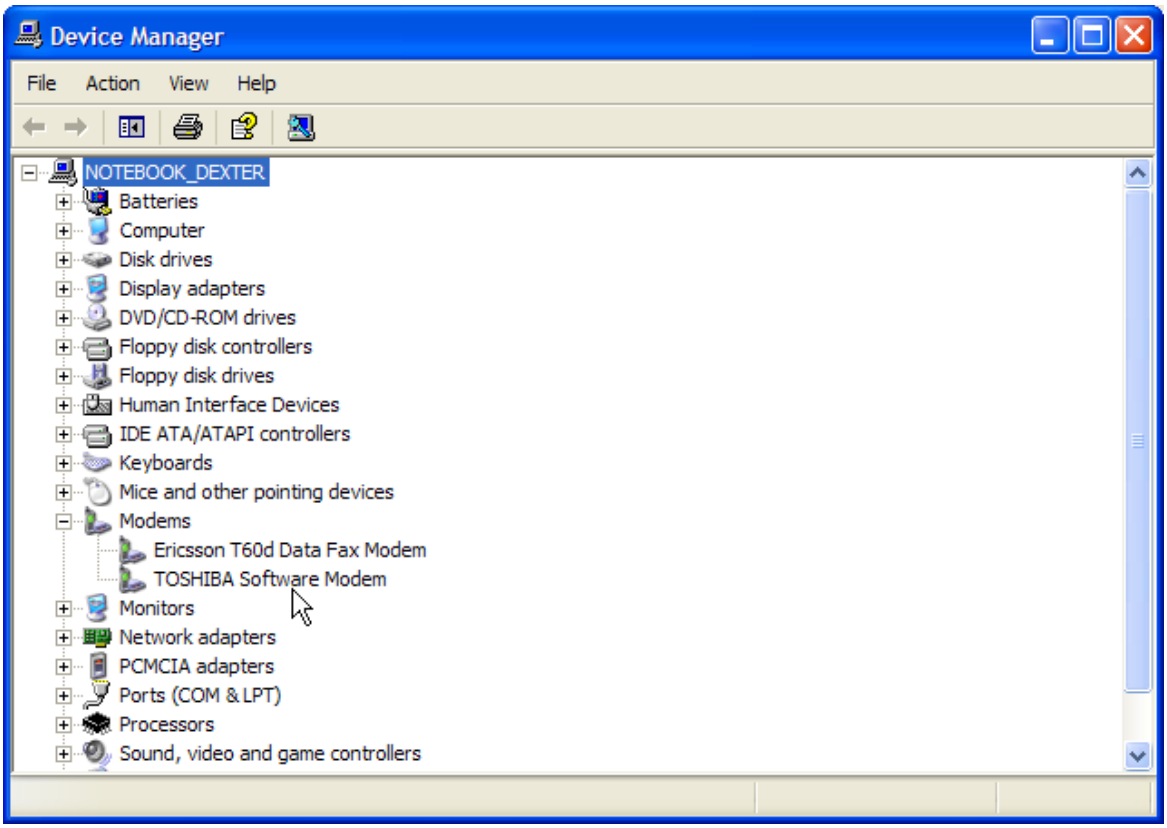


A aba TCP/IP permite selecionar o endereço do controlador µDX200 na rede Ethernet, o número da porta utilizado, e a velocidade de comunicação serial usada entre o CLP e o adaptador Ethernet↔RS-232. Note que é preciso intercalar entre o µDX200 e a rede Ethernet um adaptador Ethernet↔RS-232 (dispositivo facilmente obtenível no mercado. A Dexter pode fornecer tal equipamento se necessário, ou indicar fornecedores).

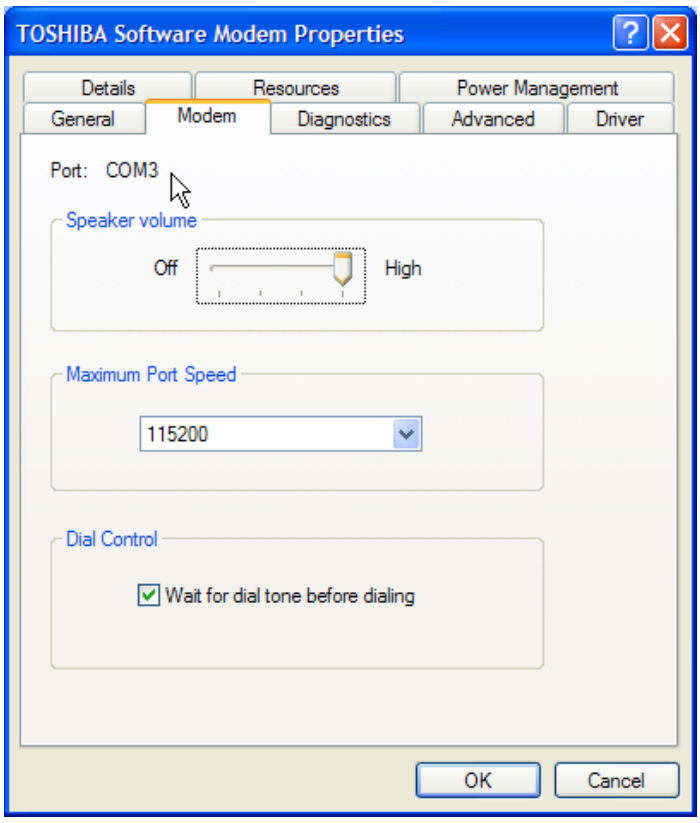
Por fim, a aba **Modem** configura a comunicação via rede telefônica fixa. Note que a comunicação é direcionada para uma porta serial, onde deve existir um modem (interno ou externo ao computador). Evidentemente, do outro lado da comunicação telefônica é mandatório existir um modem conectado via porta RS-232 ao µDX200.

Temos os campos para seleção da porta de comunicação serial ligada ao modem, velocidade de comunicação (em bits por segundo), paridade e número de stop bits. Também é possível programar um timeout para este tipo de comunicação (caso este dado seja menor que o timeout mínimo geral programado na aba **Comunicador** prevalecerá o timeout mínimo geral).

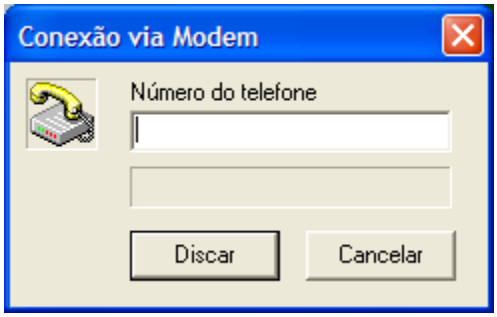
Existe um campo para programar string de inicialização do modem, cujo valor inicial é ATE0V1. Para descobrir qual a porta serial onde está o modem do computador usa-se, novamente, a janela **Gerenciador de Dispositivos** no sistema operacional Windows, selecionando o item **Modems**:



Clique duas vezes sobre o driver para modem de forma a abrir as propriedades do mesmo, e vá na aba **Modem** para obter a porta COM ocupa pelo modem do computador:

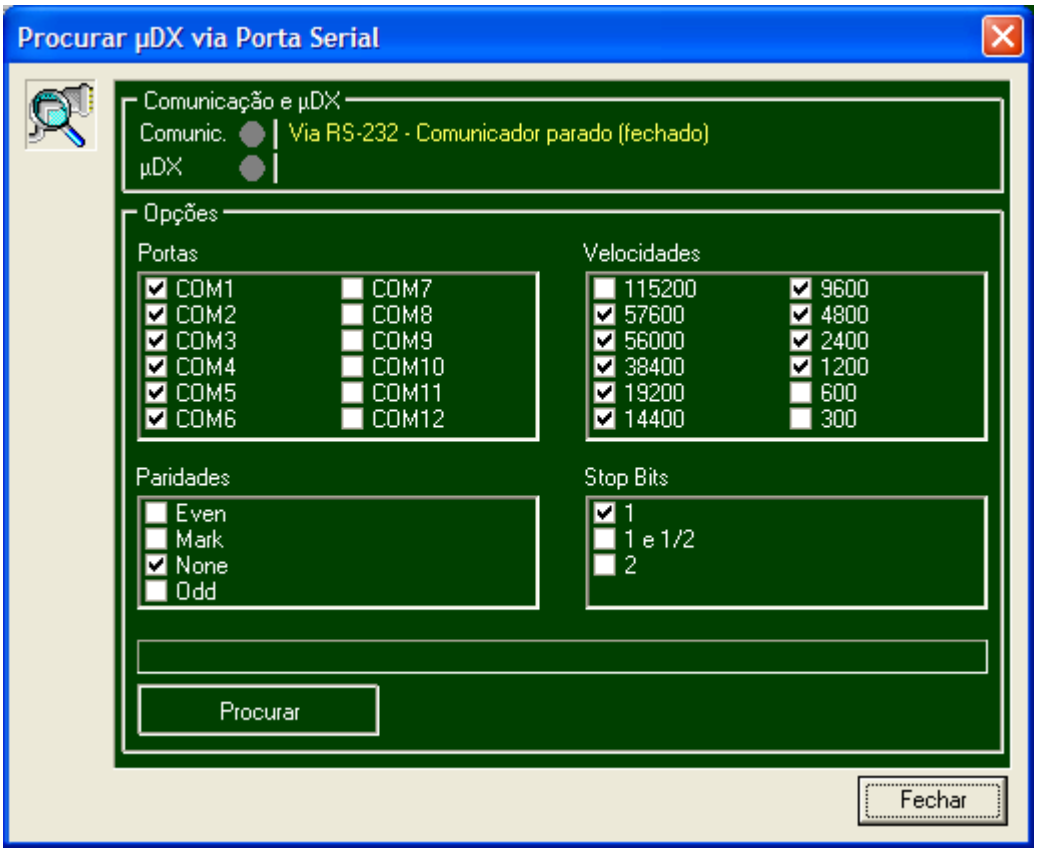


Caso esteja selecionada a opção Modem como meio de comunicação, ao abrir Comunicador do PG surgirá a tela seguinte, de forma a ser informado o número de telefone para discagem:



Procurar µDX...

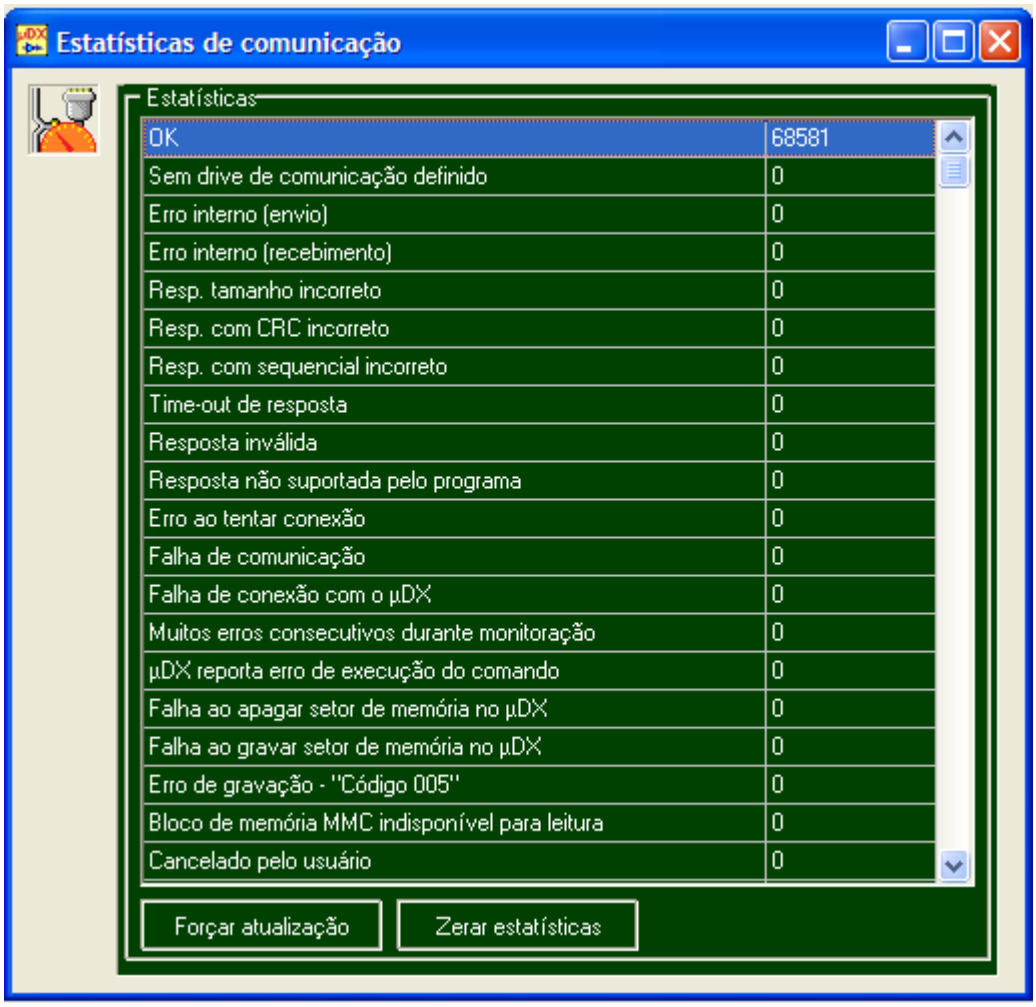
Devido a grande quantidade de opções de velocidade de comunicação, com ou sem paridade, e as diversas portas seriais que podem estar disponíveis no computador, pode ser bastante trabalhoso descobrir manualmente a configuração do µDX200 para comunicação RS-232. Neste caso esta ferramenta é bastante útil. Ela tenta estabelecer comunicação serial com o CLP em todas as portas COM selecionadas, e em todos os baud rates e configurações escolhidos:



Clique em Procurar para iniciar a varredura em todas as portas seriais (COM) selecionadas.

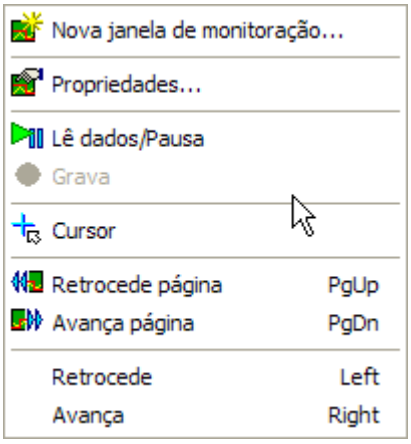
Estatísticas de comunicação

Esta opção permite verificar a qualidade da comunicação entre µDX200 e o software PG. A janela indica o total de comunicações, quais falharam e qual o motivo da falha. Uma taxa de falhas superior a 0,1% (uma falha em mil) é preocupante. A taxa de atualização é de cerca de 3 segundos, mas a tecla **[Forçar atualização]** permite atualizar a qualquer instante, enquanto a tecla **[Zerar estatísticas]** reinicia os dados.



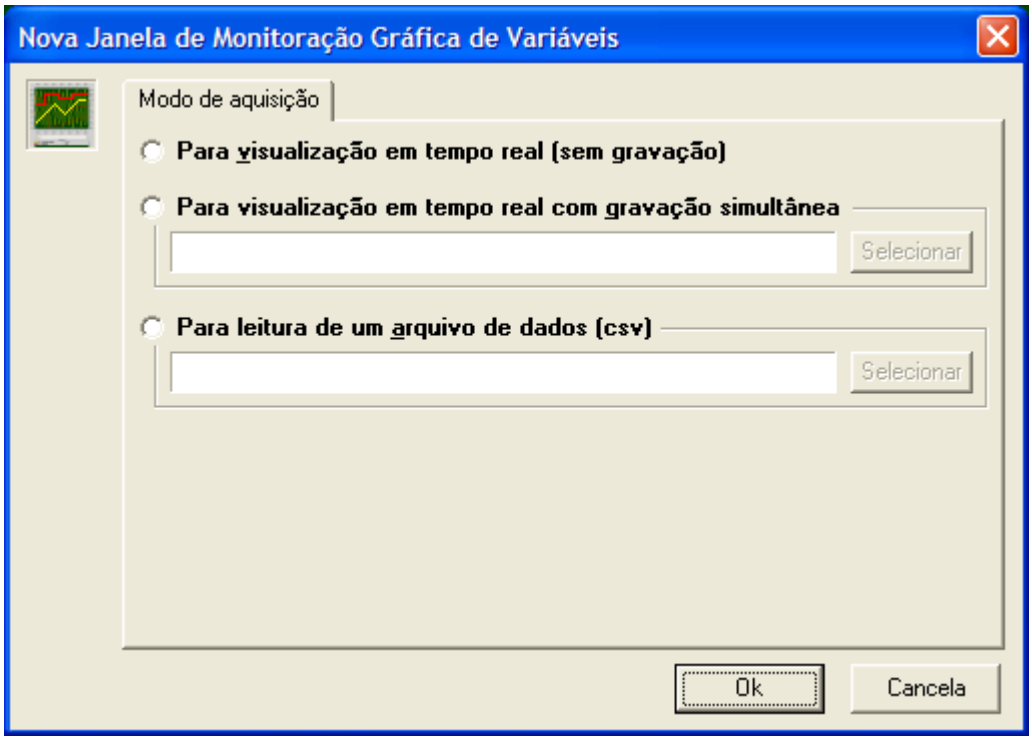
Menu Monitoração

Este menu controla as opções para monitoração gráfica de variáveis no Compilador PG. Com isso, é possível capturar valores de variáveis e plotá-las em um gráfico, facilitando a depuração de algoritmos de controle analógicos e leitura de sensores, por exemplo. É possível abrir diversas janelas de monitoração simultaneamente no PG.



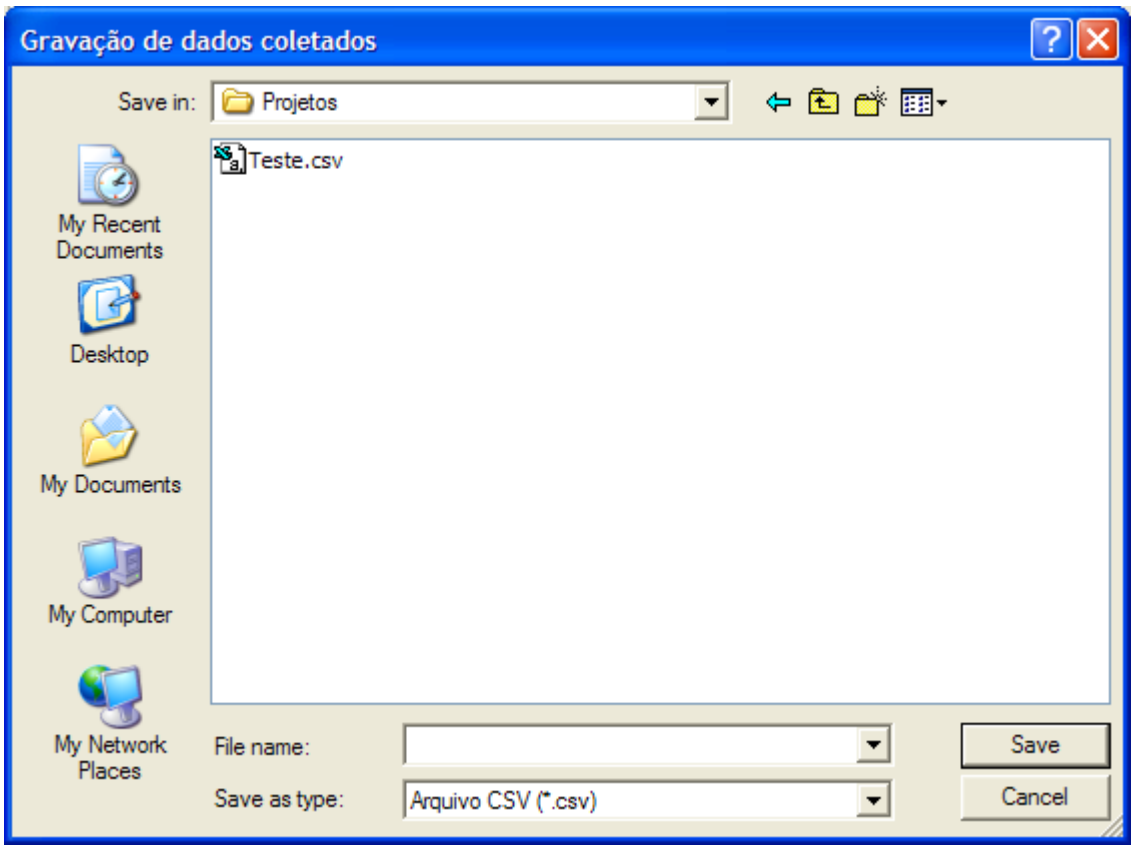
Nova Janela de Monitoração...

Esta opção cria uma nova janela de monitoração gráfica. Com isso, é possível monitorar graficamente variáveis do Controlador µDX200, e salvar os valores lidos em arquivos compatíveis com planilhas eletrônicas como Excel. Inicialmente é criada uma janela com as seguintes possibilidades:

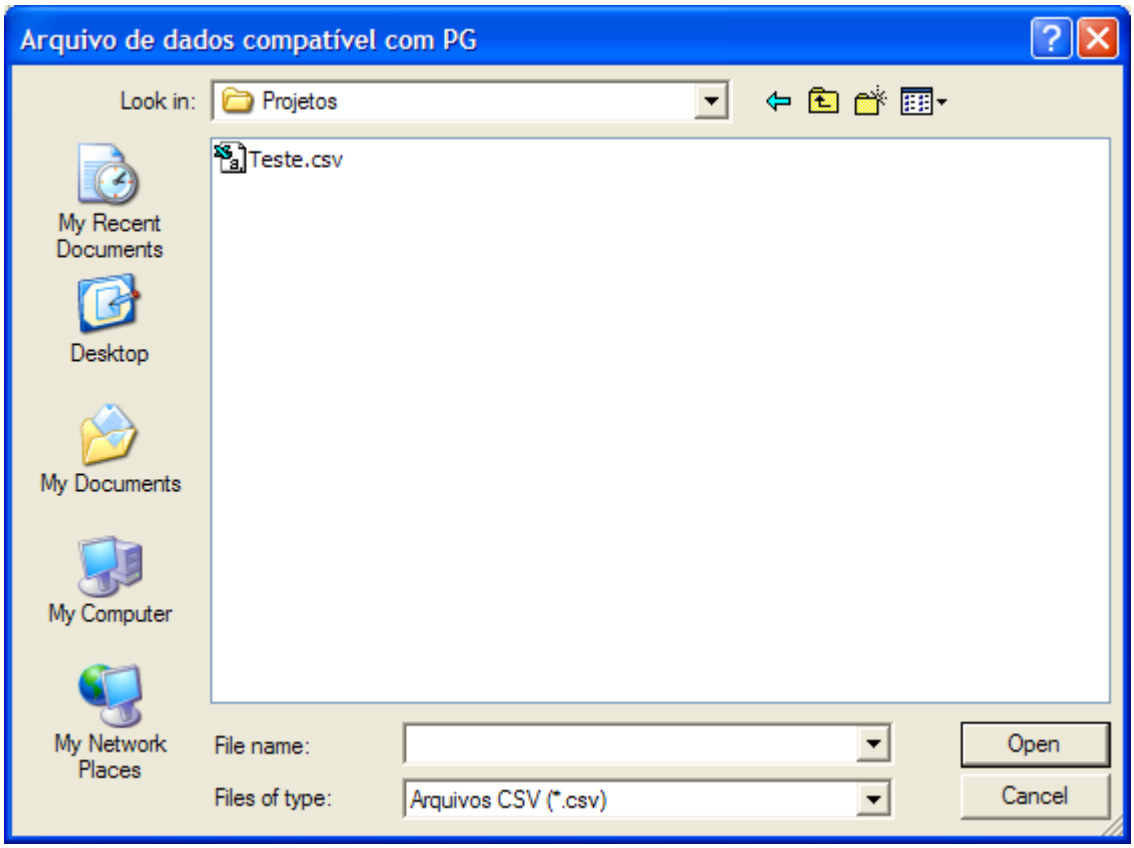


O **Modo de Aquisição** pode ser de três tipos:

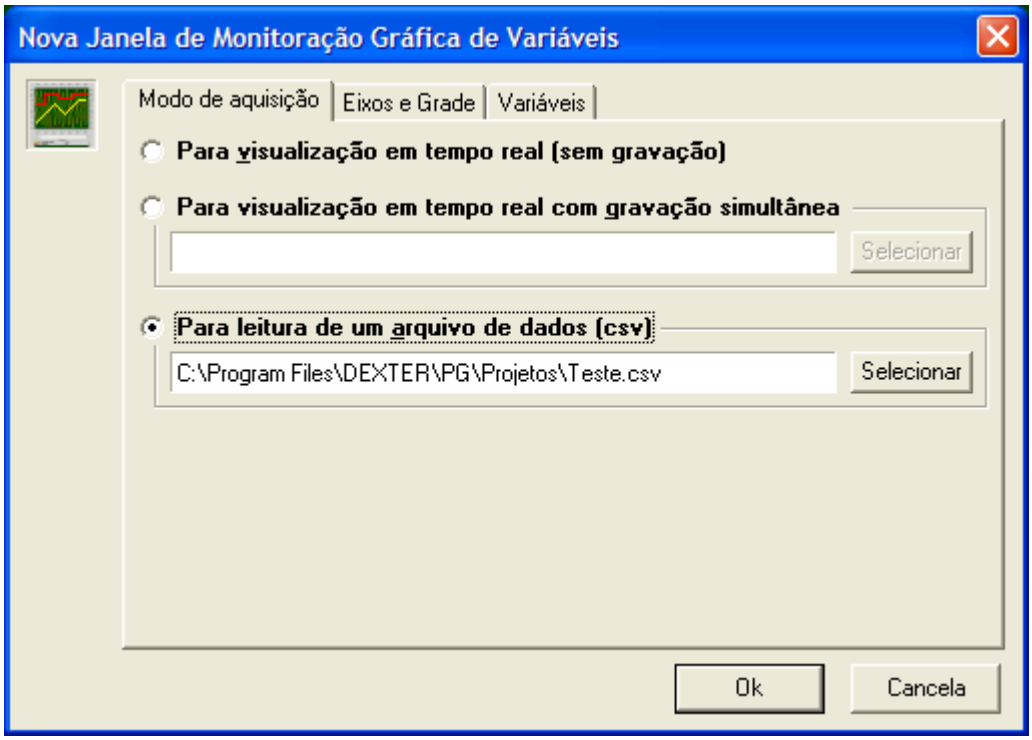
- **Para visualização em tempo real (sem gravação):** neste caso os dados irão aparecer na tela em tempo real, mas não serão gravados. Evidentemente, neste caso está descartado poder navegar para áreas anteriores do gráfico (ao completar a tela disponível todo o gráfico passa a rodar horizontalmente, descartando os dados mais antigos).
- **Para visualização em tempo real com gravação simultânea:** idêntico ao caso anterior, mas os dados lidos são também gravados em arquivo de tipo CSV, passível de ser lido posteriormente tanto no próprio PG como em uma planilha eletrônica como Excel, por exemplo. Ao selecionar esta opção surge uma janela para seleção do nome do arquivo a ser gerado com os dados:



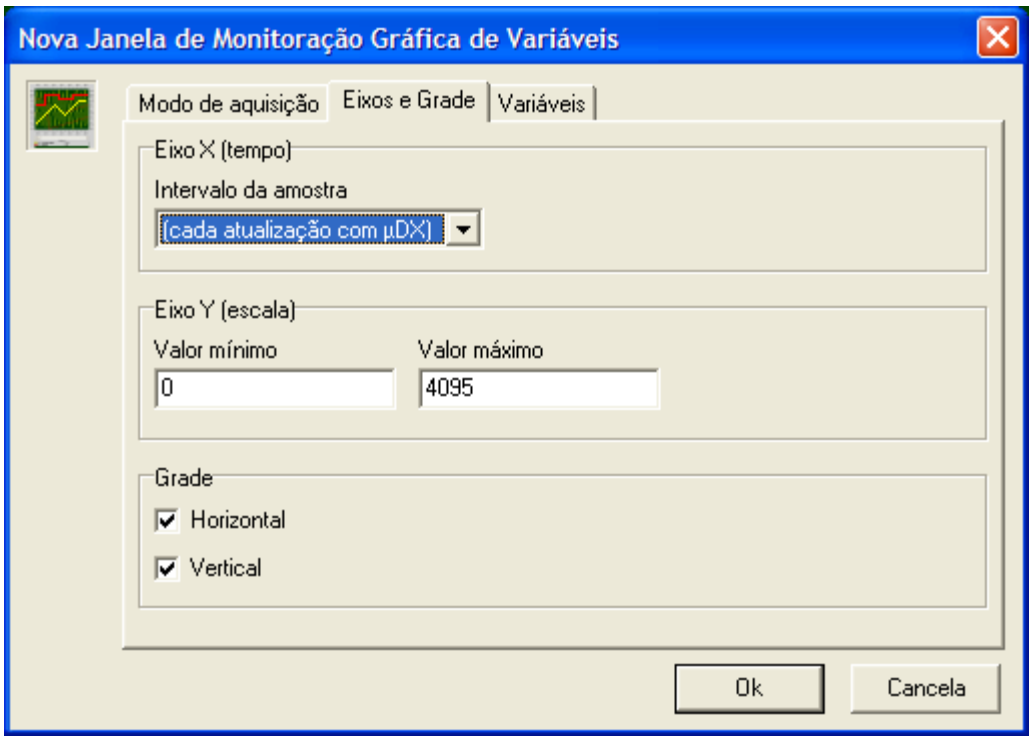
- **Para leitura de um arquivo de dados (CSV):** permite ler um arquivo do tipo CSV gravado anteriormente via opção anterior, ou lido de um cartão MMC gravado com dados do µDX200. Neste caso surge uma janela para seleção do arquivo a ser lido:



Ao selecionar uma das opções acima duas outras abas são ativadas na janela **Nova janela de Monitoração Gráfica de Variáveis**. São elas **Eixos e Grade** e **Variáveis**.

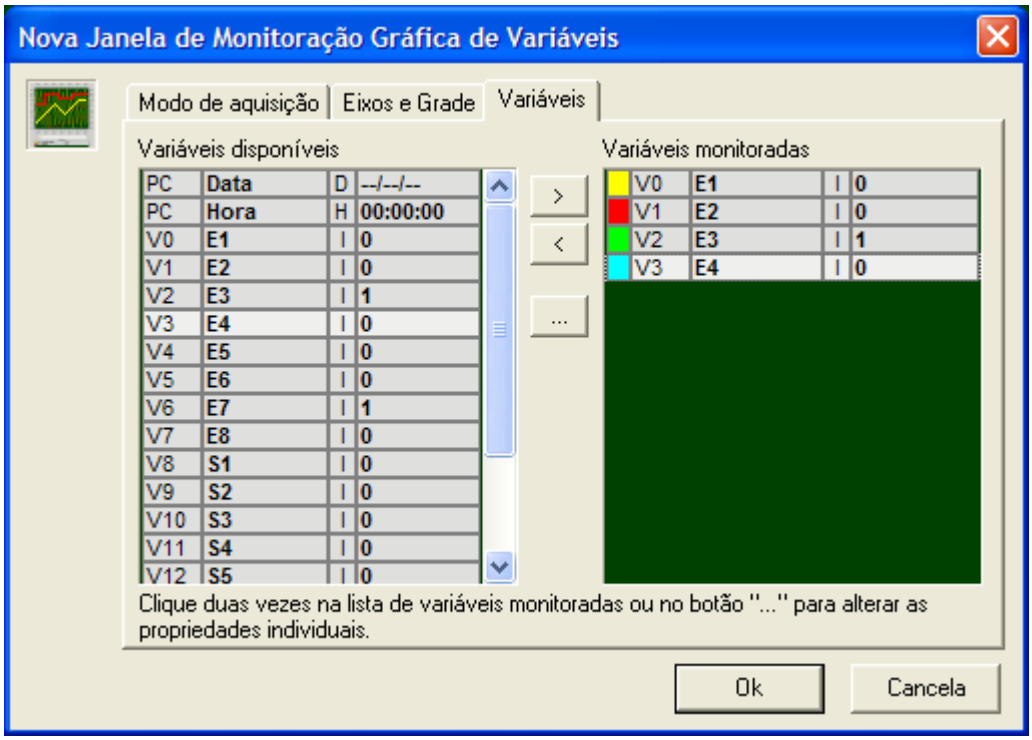


Na aba **Eixos e Grade** é possível selecionar o valor mínimo e máximo do eixo Y do gráfico (o valor default é 0 e 4095), o intervalo de amostra do eixo X (o default é a cada atualização com µDX, ou seja, a velocidade máxima de monitoramento do PG). Por fim, também pode-se inibir a grade horizontal ou vertical do gráfico.

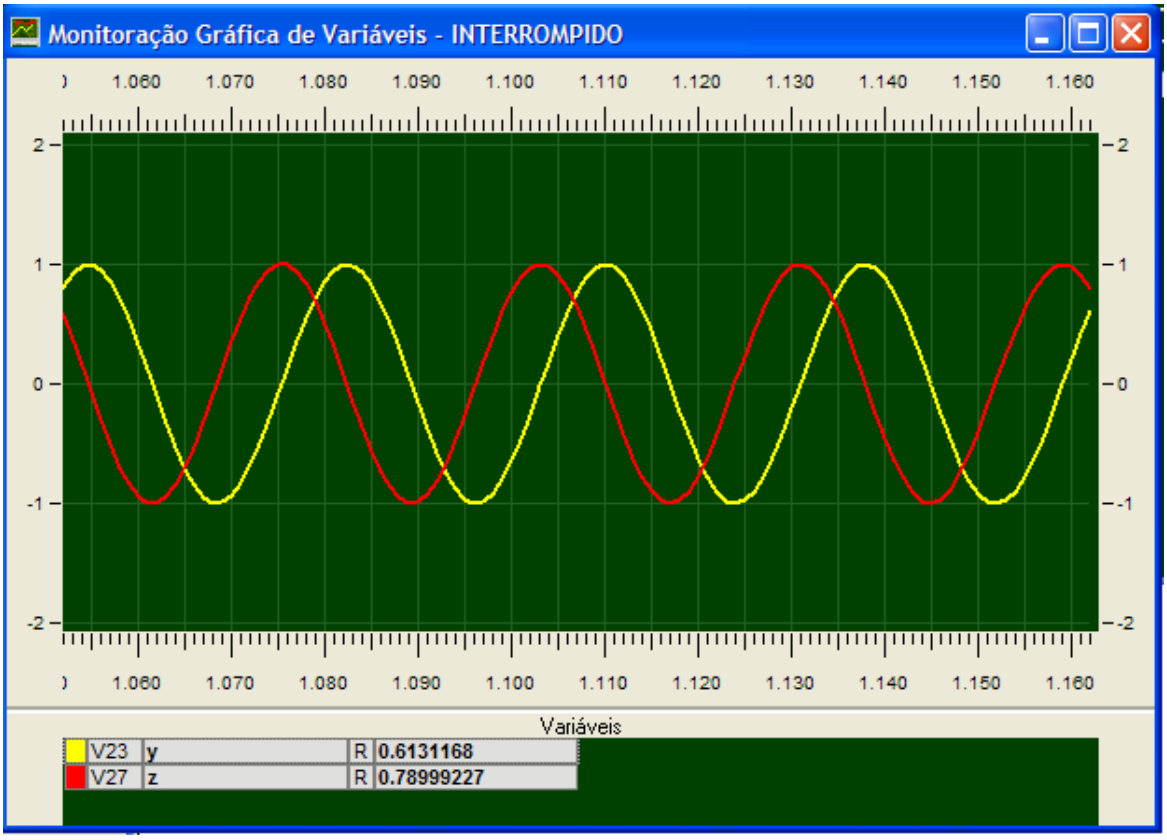


Na aba **Variáveis** seleciona-se quais variáveis serão plotadas no gráfico. Para que apareçam variáveis do µDX nas possibilidades de seleção desta janela é preciso que o Compilador PG esteja com a monitoração ativada (se não somente data e hora do microcomputador PC estarão disponíveis). Para selecionar as variáveis a serem plotadas basta clicar duas vezes sobre as mesmas, ou selecioná-las e clicar no botão  . O botão  permite eliminar variáveis da plotagem. O botão  possibilita editar o nome da variável e sua cor.

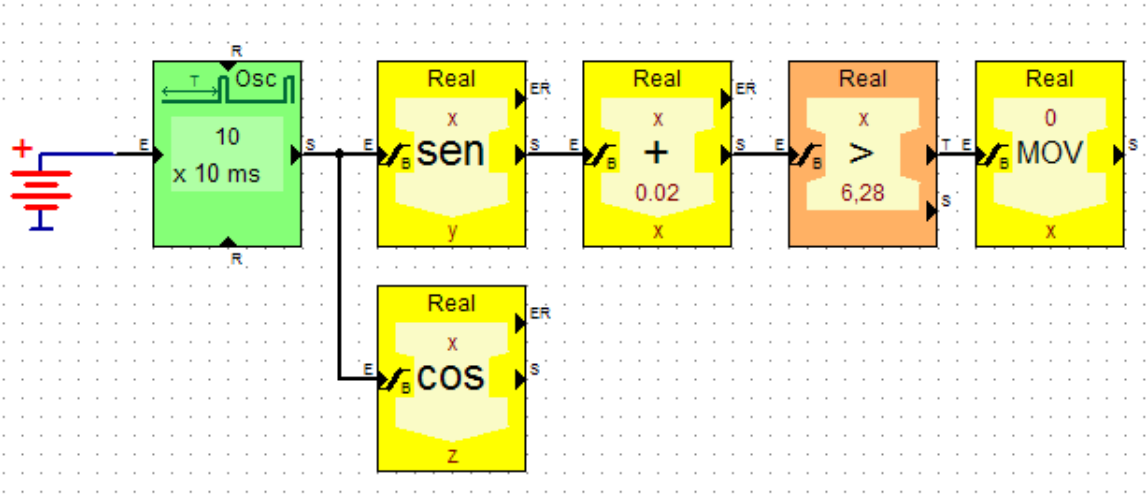
Caso o programa aplicativo existente no µDX200 coincida com o programa aplicativo carregado no Compilador PG (os CRCs remoto e local devem ser idênticos) estarão disponíveis para plotagem todas as variáveis usadas no programa do CLP. Ao clicar em Ok é gerada a janela de monitoração gráfica.



A janela de monitoração gráfica a seguir foi gerada com captura das variáveis y e z (seno e cosseno, respectivamente) a cada segundo:



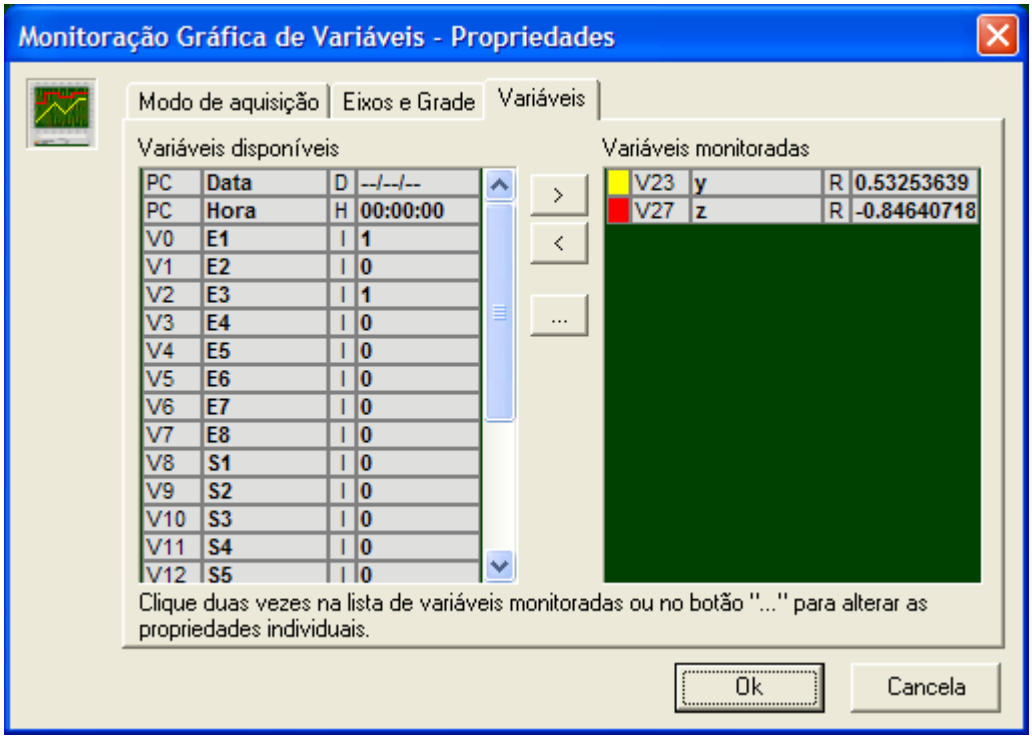
O programa aplicativo que gerou estas variáveis é:



Exemplo de Programa Aplicativo: **Sen_Cos.dxg** (fonte); **Sen_Cos.udp** (compilado).

Propriedades

Acessa os parâmetros do gráfico, como escala X e Y, variáveis monitoradas, etc., permitindo modificá-los.

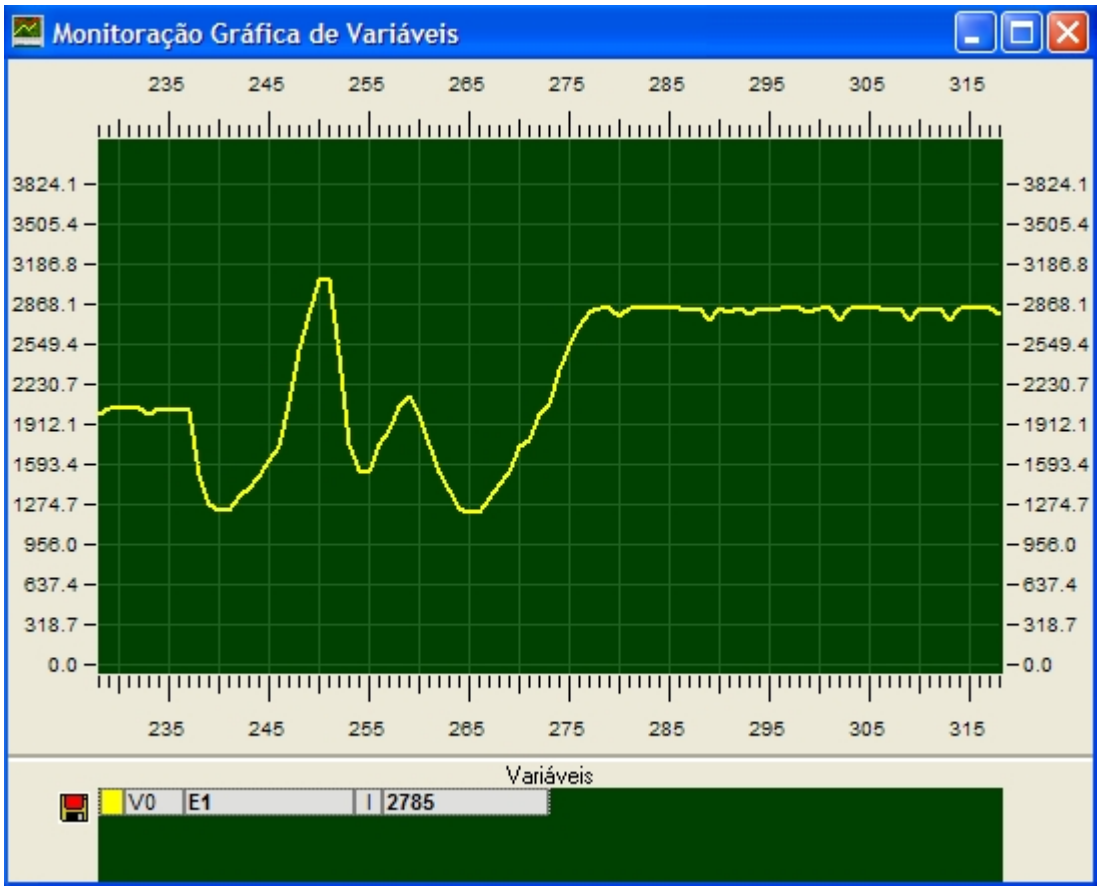


Lê Dados/Pausa

Interrompe ou reinicia a leitura de dados do gráfico.

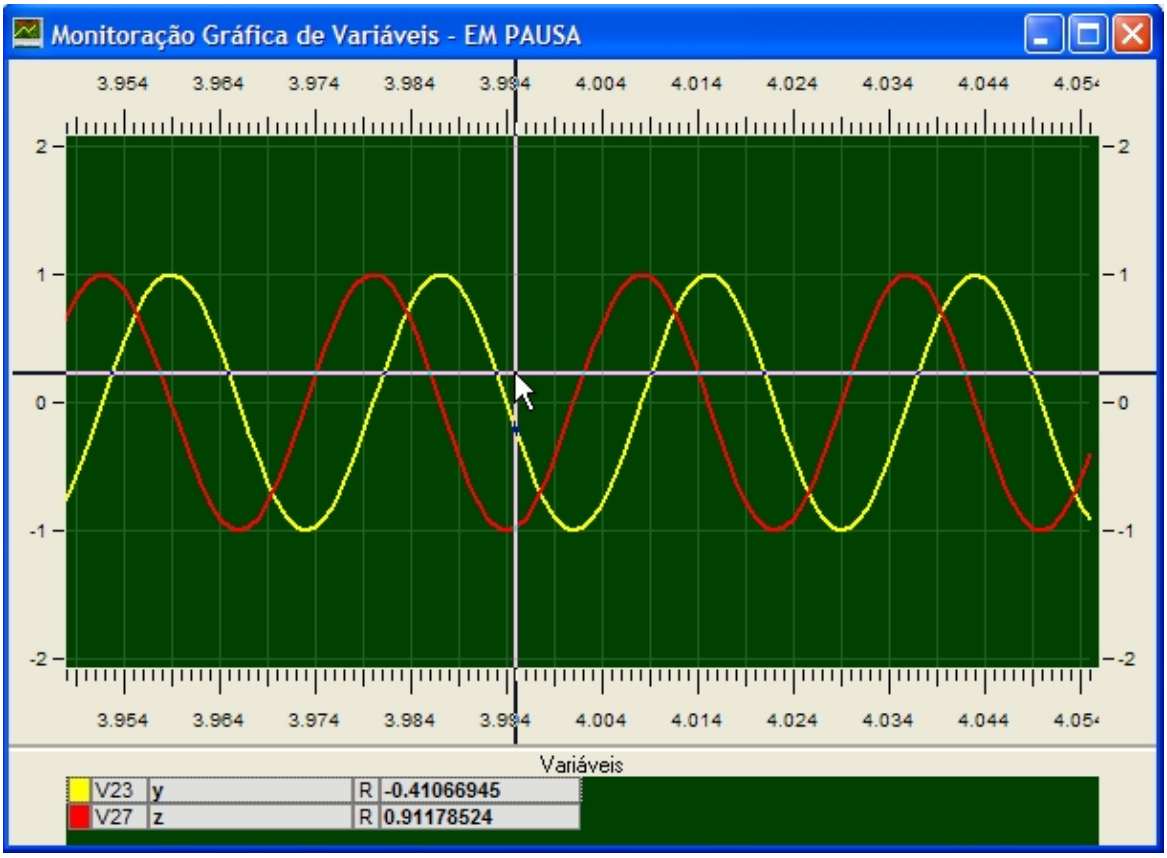
Grava

Interrompe ou reinicia a gravação dos dados lidos em arquivo. Note que a gravação dos dados é sinalizada por um símbolo de diskette em vermelho no canto inferior esquerdo da janela de monitoração.



Cursor

Cria um cursor sobre o gráfico, permitindo a leitura dos valores das variáveis em qualquer ponto do mesmo.



Retrocede Página (PgUp)

No caso de leitura de arquivo de dados (CSV) é permitido avançar e retroceder o gráfico. Esta opção retrocede em uma página completa o gráfico. Pode ser usada a tecla PgUp do teclado do computador. Note que o arquivo CSV pode ser bastante extenso (gerado via janela de monitoração gráfica com gravaçào ativada, ou via cartão MMC).

Avança Página (PgDn)

Esta opção avança em uma página completa o gráfico. Pode ser usada a tecla PgDn do teclado.

Retrocede (Left)

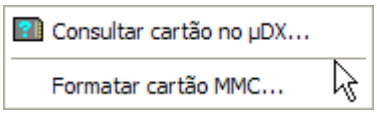
Retrocede em um registro o gráfico. Note que pode ser usada a seta para esquerda do teclado (Left).

Avança (Right)

Avança em um registro o gráfico. Note que pode ser usada a seta para direita do teclado (Right).

Menu MMC

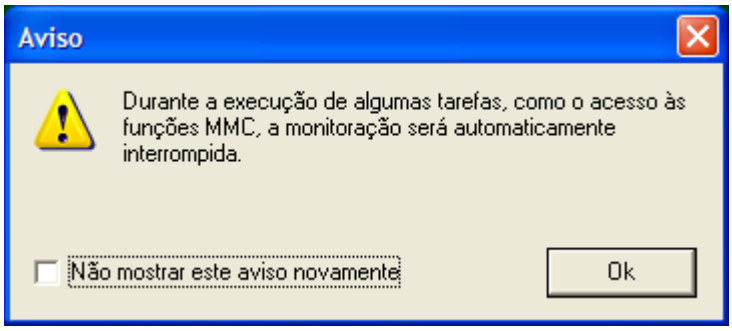
O menu MMC permite ler o cartão MMC (Multi Media Card) instalado no controlador µDX200, ou formatá-lo no computador para uso com o µDX200:

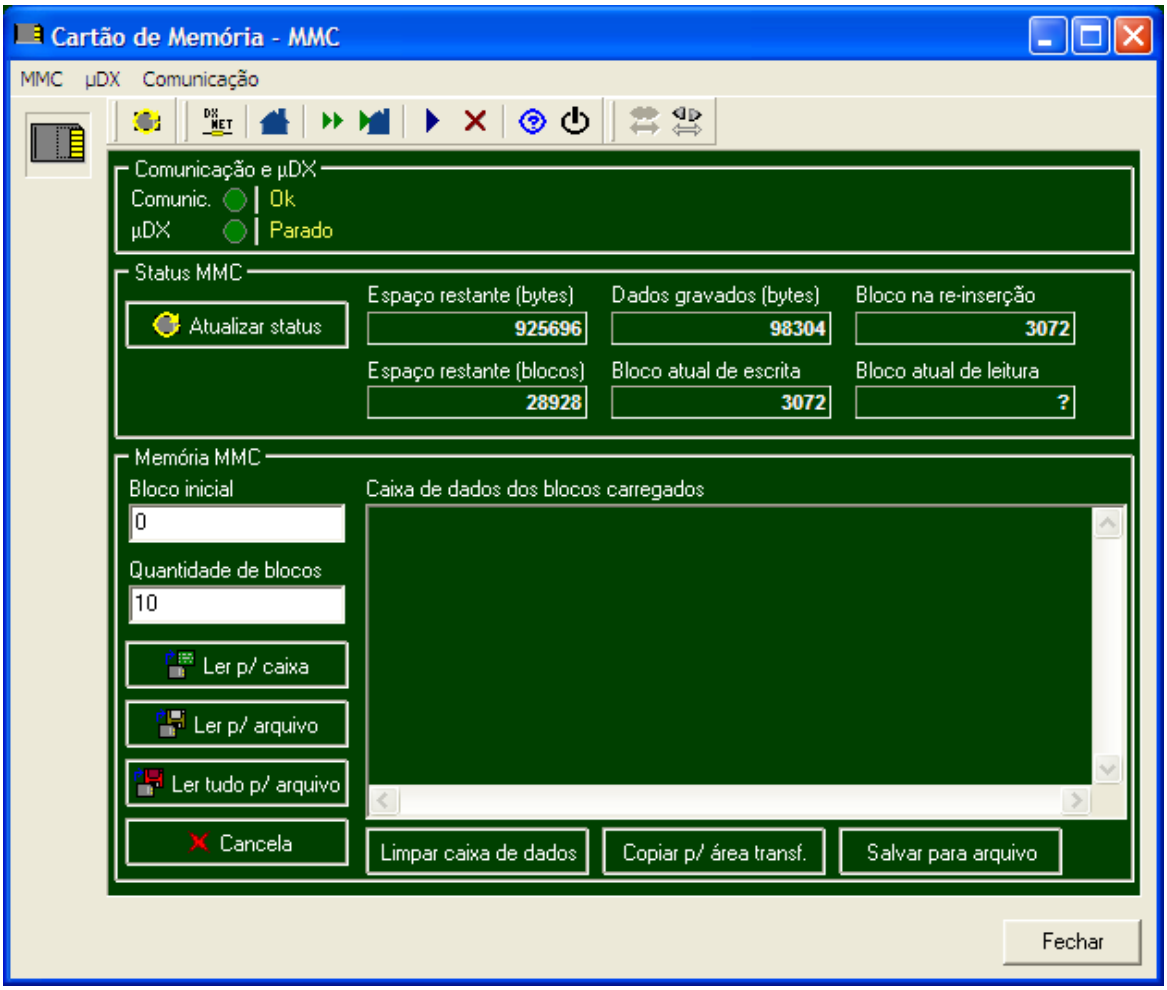


Atenção: Atualmente existem variantes do cartão padrão MMC, como MMC-RS (reduced size) ou MMC Plus (com maior velocidade de comunicação). Nenhuma destas variantes funciona corretamente no controlador µDX200. Aconselha-se, portanto, adquirir cartões MMC diretamente com a Dexter para garantia de compatibilidade. No caso de controladores µDX201 eles são, normalmente, compatíveis com essas variantes de cartão MMC. Além disso, permitem o uso de cartões SD, muito mais facéis de encontrar no mercado atualmente.

Consultar Cartão no µDX...

Esta opção permite a leitura dos dados gravados no cartão MMC instalado no controlador µDX200, via comunicação serial. Ao selecioná-la surge uma tela de advertência para alertar que a monitoração será interrompida (a leitura de cartão MMC, assim como a leitura de buffer serial, exige a interrupção da monitoração) e, a seguir, a tela de leitura de cartão MMC:

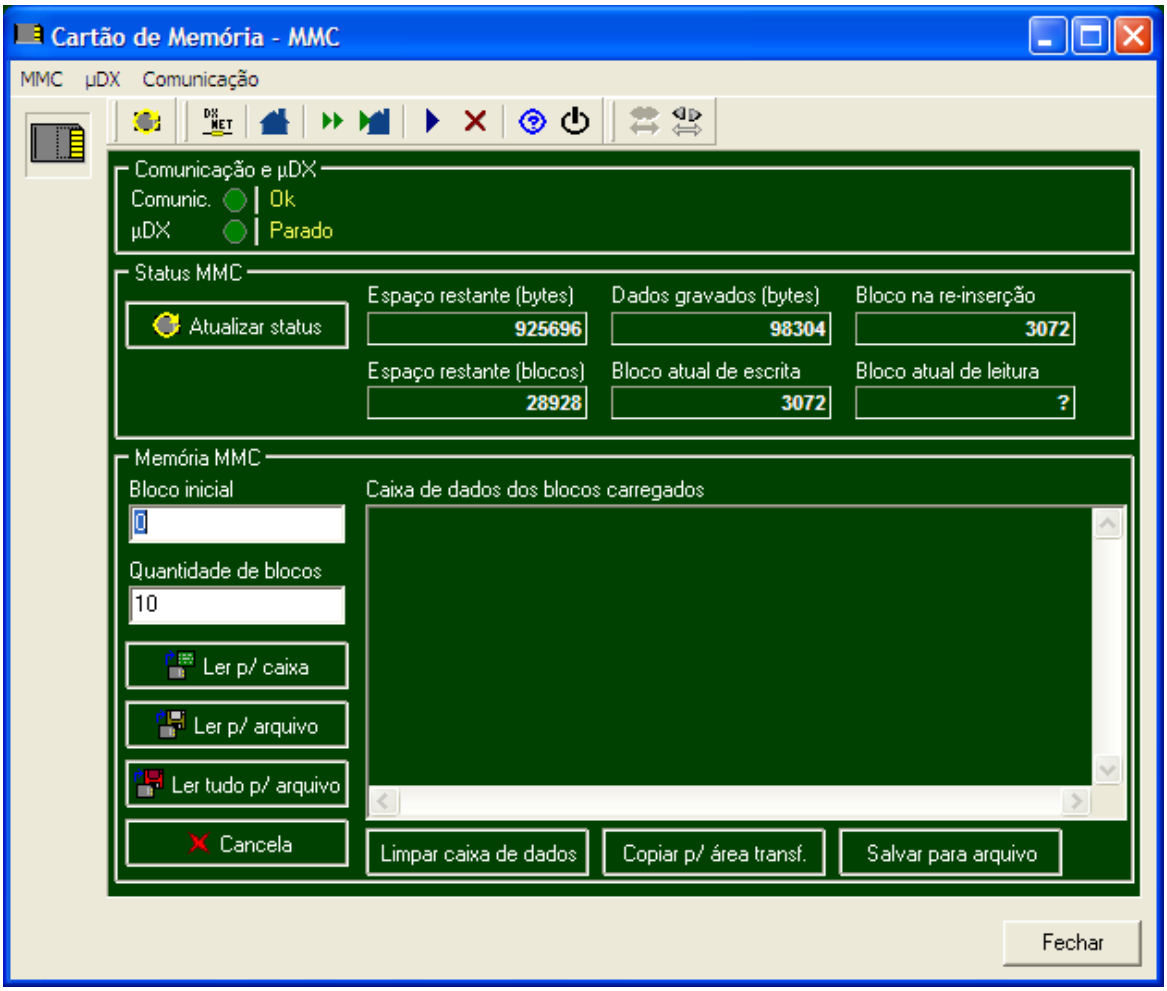




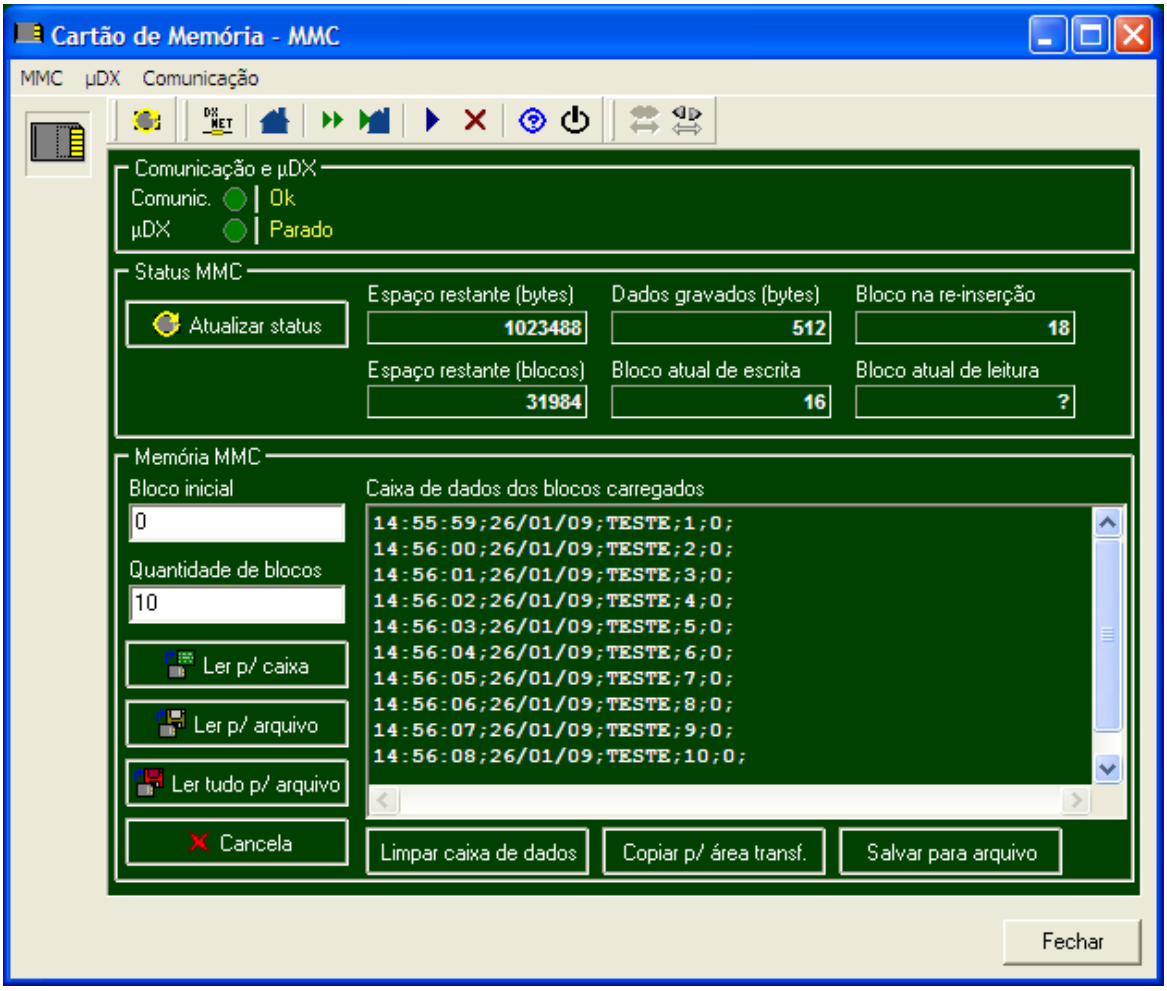
Esta janela replica vários comandos existentes no Compilador PG, já que a monitoração é interrompida durante a leitura de MMC. Assim, temos diversas opções nos menus pop-down desta janela mas, à exceção do menu **MMC**, que possui a opção **Atualizar status**, as demais opções já foram explicadas nos menus **µDX** e **Comunicação** do Compilador PG.

A opção de **Atualizar status** lê os dados de status do cartão MMC. Estão disponíveis os seguintes dados:

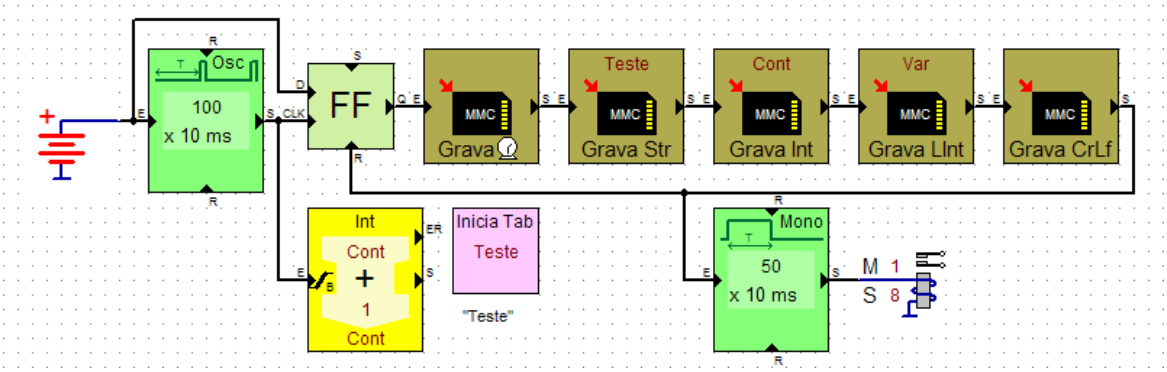
- **Espaço restante (bytes):** Indica quantos bytes restam no arquivo do cartão MMC (LOGDX200.CSV).
- **Espaço restante (blocos):** Cada bloco é constituído de 32 bytes. Portanto, é o valor anterior dividido por 32.
- **Dados gravados (bytes):** Número de bytes já gravado no arquivo do cartão MMC (LOGDX200.CSV).
- **Bloco atual de escrita:** Indica o bloco atual de escrita do cartão (número de bytes gravados ÷ 32).
- **Bloco na re-inserção:** Caso o cartão tenha sido retirado do µDX este dado indica o ponto de re-inserção.
- **Bloco atual de leitura:** No caso de leitura dos dados no PC este dado indica o primeiro bloco a ser lido.



Logo abaixo, é possível informar qual o bloco inicial e o número de blocos a serem lidos. Lembre-se que cada bloco é constituído por 32 bytes. Assim, se especificarmos 10 blocos serão lidos 320 bytes do arquivo LOGDX200.CSV existente no cartão MMC instalado no µDX200. Como os dados são gravados no cartão codificados em ASCII, cada byte irá corresponder a um caracter. É possível direcionar os dados para a caixa de dados existente na janela, ou diretamente para um arquivo. Também é possível limpar a caixa de dados, copiar seu conteúdo para a área de transferência (copiar e colar em outro aplicativo), ou ainda salvar o conteúdo da caixa de dados em um arquivo.



O programa que gerou os dados mostrados na caixa de dados é o seguinte:

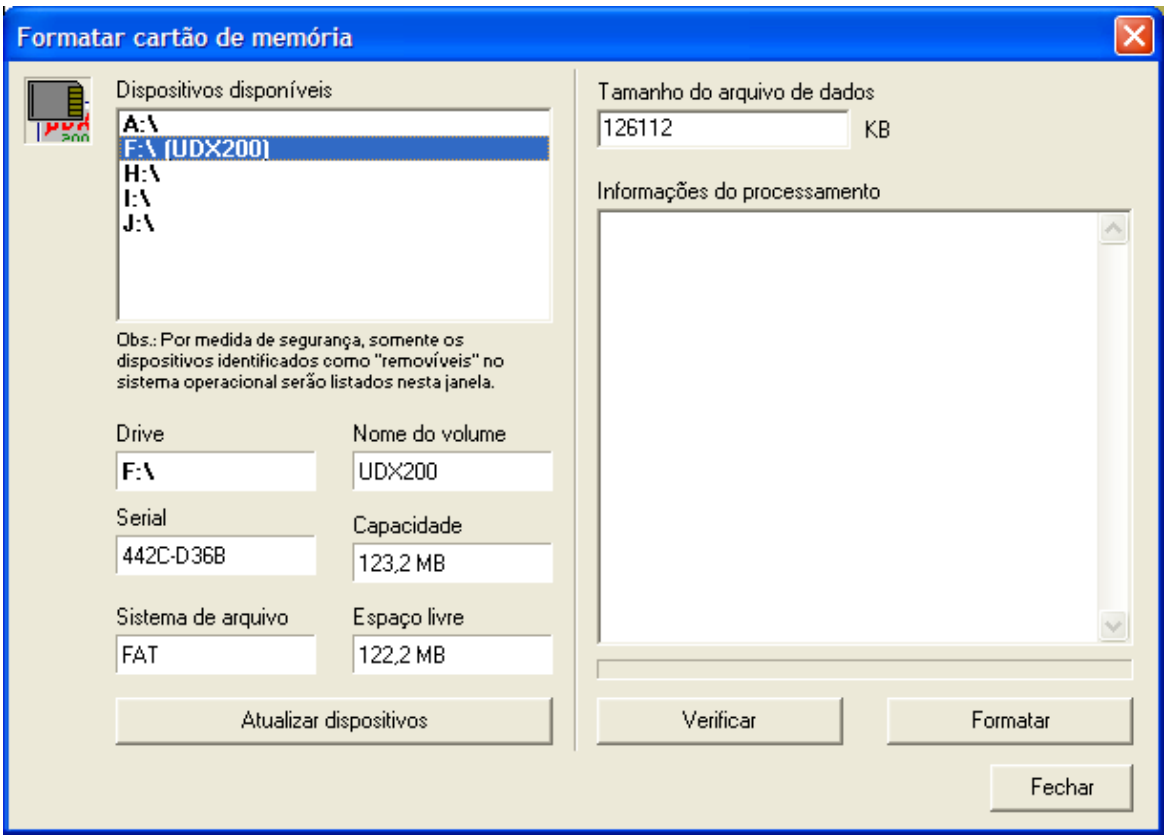


Exemplo de Programa Aplicativo: **Teste_MMC.dxg** (fonte); **Teste_MMC.udp** (compilado).

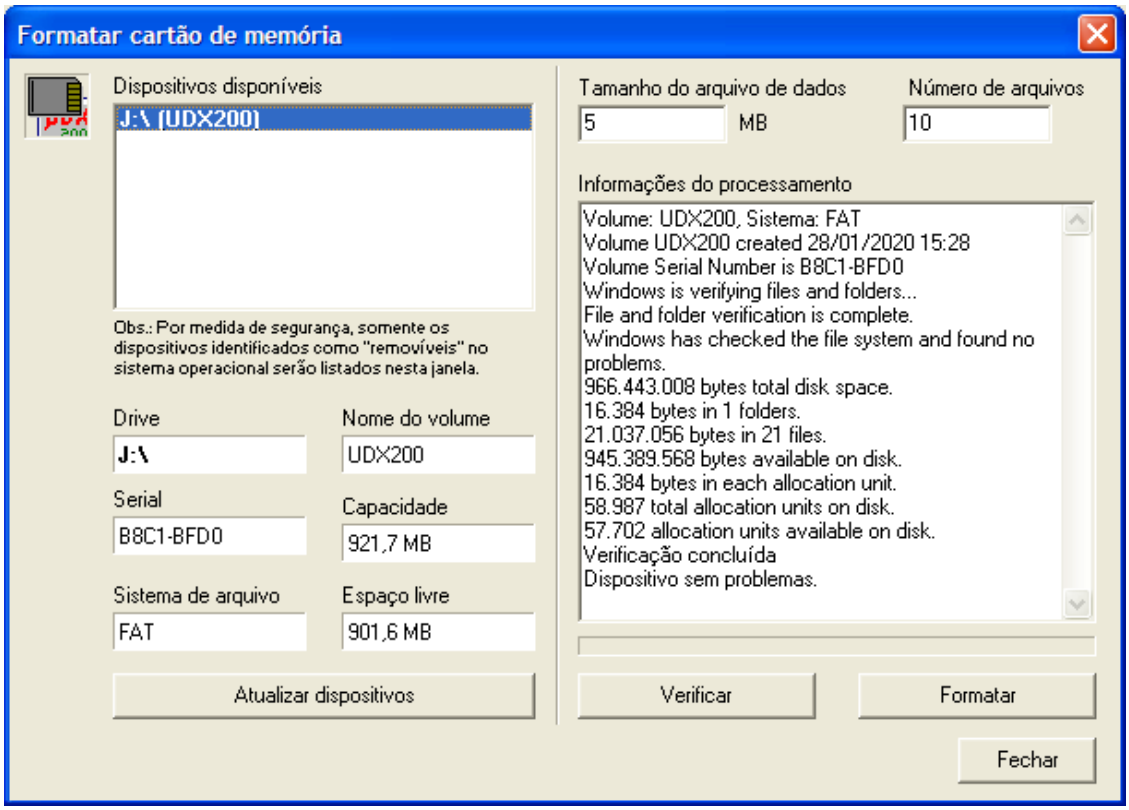
Note que, além da leitura para caixa de dados ou para arquivo, também existe a opção de leitura de todo cartão para arquivo (**Ler tudo p/arquivo**). Neste caso todo conteúdo do cartão MMC será lido e gravado em um arquivo sufixo **.CSV**. É preciso tomar certo cuidado com arquivos de registro muito extensos, pois a leitura via comunicação serial não é muito rápida e a operação pode demorar vários minutos.

Formatar Cartão MMC...

Esta janela permite formatar cartões MMC (Multi Media Card) para uso no controlador µDX200. Para isso é necessário conectar um leitor de cartão MMC (normalmente ligado a uma porta USB do computador) ao computador, se seu PC já não possui leitor de cartão embutido. O cartão será reconhecido pelo sistema operacional Windows como mais uma unidade de armazenamento (letra f:, por exemplo). Selecione a letra da unidade correspondente ao cartão MMC no campo "Dispositivos disponíveis". Normalmente o PG reconhece o endereço do cartão MMC e já o seleciona. Especifique um tamanho em Kbytes para o arquivo de registro (log) que será gerado no cartão MMC. O campo "Tamanho do arquivo de dados" inicializa com o valor máximo permitido pelo cartão MMC. Os cartões usados no µDX200 podem ter capacidade de 64 Mbytes a 2Gbytes. Assim, o arquivo de registro pode ser bastante extenso. Entretanto, quanto maior for este arquivo mais demorada será sua formatação. Então, aconselha-se um arquivo de registro não superior a 10000 Kbytes, já que a quantidade de dados a ser armazenada via programa aplicativo costuma ser pequena. Clique em [Formatar] para iniciar a formatação do cartão MMC. Uma vez completada a formatação o cartão está pronto para ser utilizado no µDX200.



A partir da versão 2.3.2.0 do PG é possível criar múltiplos arquivos de dados no cartão MMC/SD. Além disso foi comutado o tamanho do arquivo de KB para MB, uma vez que os cartões atuais permitem grande capacidade de armazenamento:


Formatar cartão de memória
Dispositivos disponíveis
J:\ (UDX200)
Obs.: Por medida de segurança, somente os dispositivos identificados como "removíveis" no sistema operacional serão listados nesta janela.
Drive
J:\
Nome do volume
UDX200
Serial
B8C1-BFD0
Capacidade
921,7 MB
Sistema de arquivo
FAT
Espaço livre
901,6 MB
Atualizar dispositivos
Tamanho do arquivo de dados
5
MB
Número de arquivos
10
Informações do processamento
Volume: UDX200, Sistema: FAT
Volume UDX200 created 28/01/2020 15:28
Volume Serial Number is B8C1-BFD0
Windows is verifying files and folders...
File and folder verification is complete.
Windows has checked the file system and found no problems.
966.443.008 bytes total disk space.
16.384 bytes in 1 folders.
21.037.056 bytes in 21 files.
945.389.568 bytes available on disk.
16.384 bytes in each allocation unit.
58.987 total allocation units on disk.
57.702 allocation units available on disk.
Verificação concluída
Dispositivo sem problemas.
Verificar
Formatar
Fechar

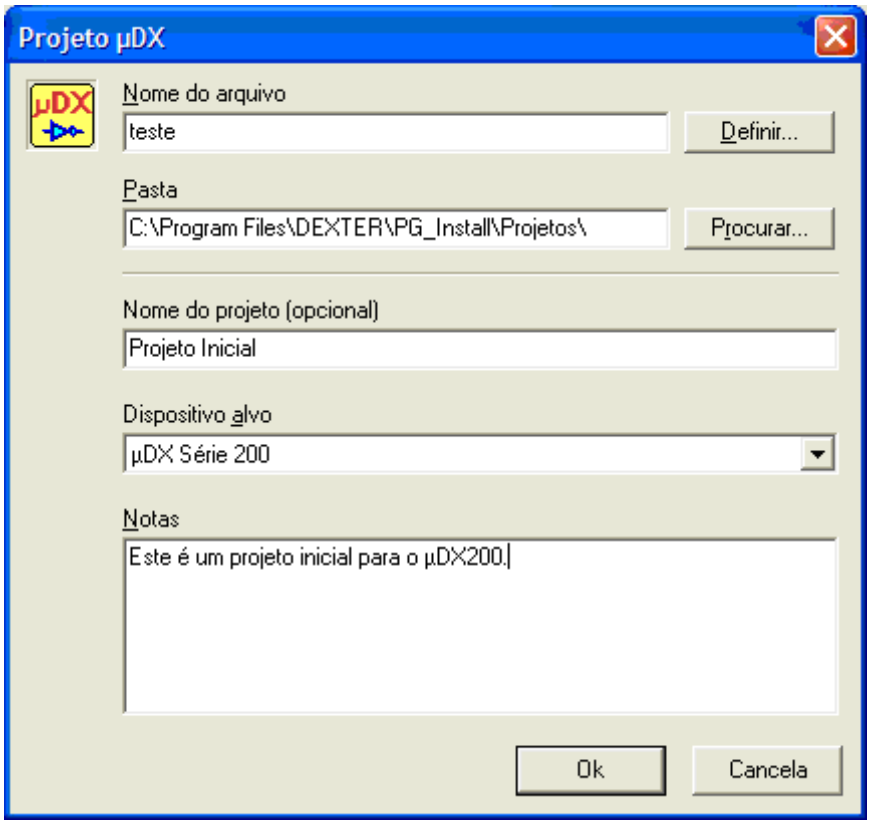
Parte VII

Elaborando Programas

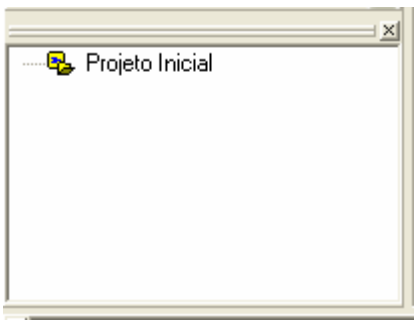
A criação de um programa para o µDX200 é muito fácil e pode ser dividida em quatro etapas:

- Abertura de um projeto para abrigar todas as páginas do programa.
- Confecção das páginas de programação.
- Pré-compilação do projeto.
- Compilação e envio do programa para o µDX200.

Então, acione o software Editor PG e pressione a tecla  existente na Barra de Ferramentas para gerar um novo projeto. Ou então abra o menu pop-down [**Arquivo**] → [**Novo projeto...**]. Irá surgir uma janela com campos de informação a serem preenchidos sobre o novo projeto. Digite as informações abaixo e pressione a tecla Ok.

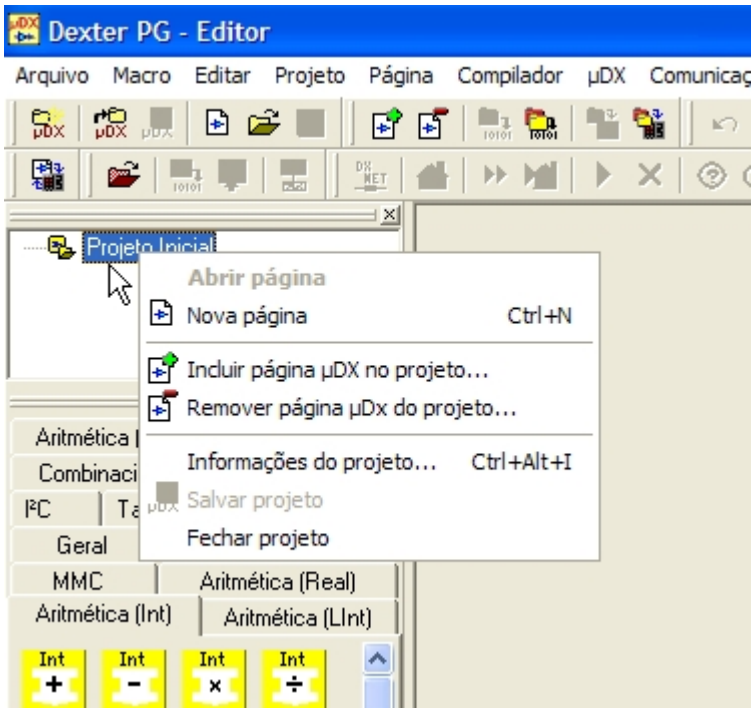


Surgirá uma pequena janela com o fontes do projeto criado (no caso, o projeto ainda não possui nenhuma página de programação):

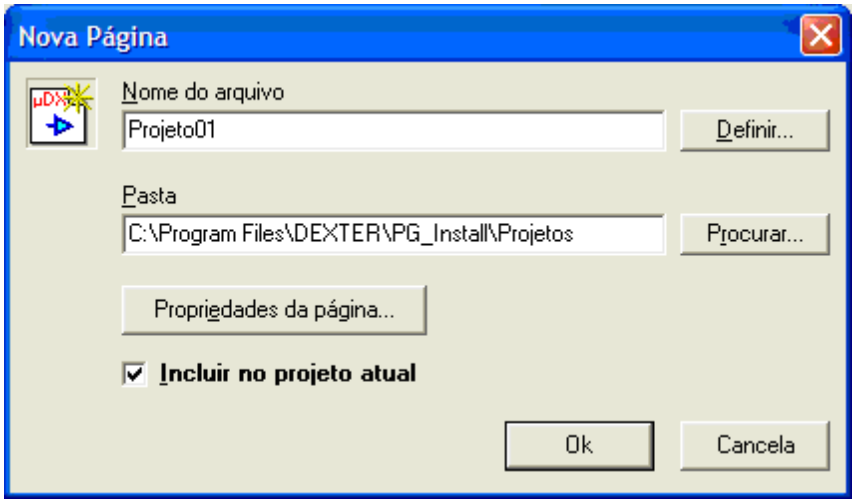


Pressionando a tecla direita do mouse sobre o projeto surgem várias opções, como inserção de

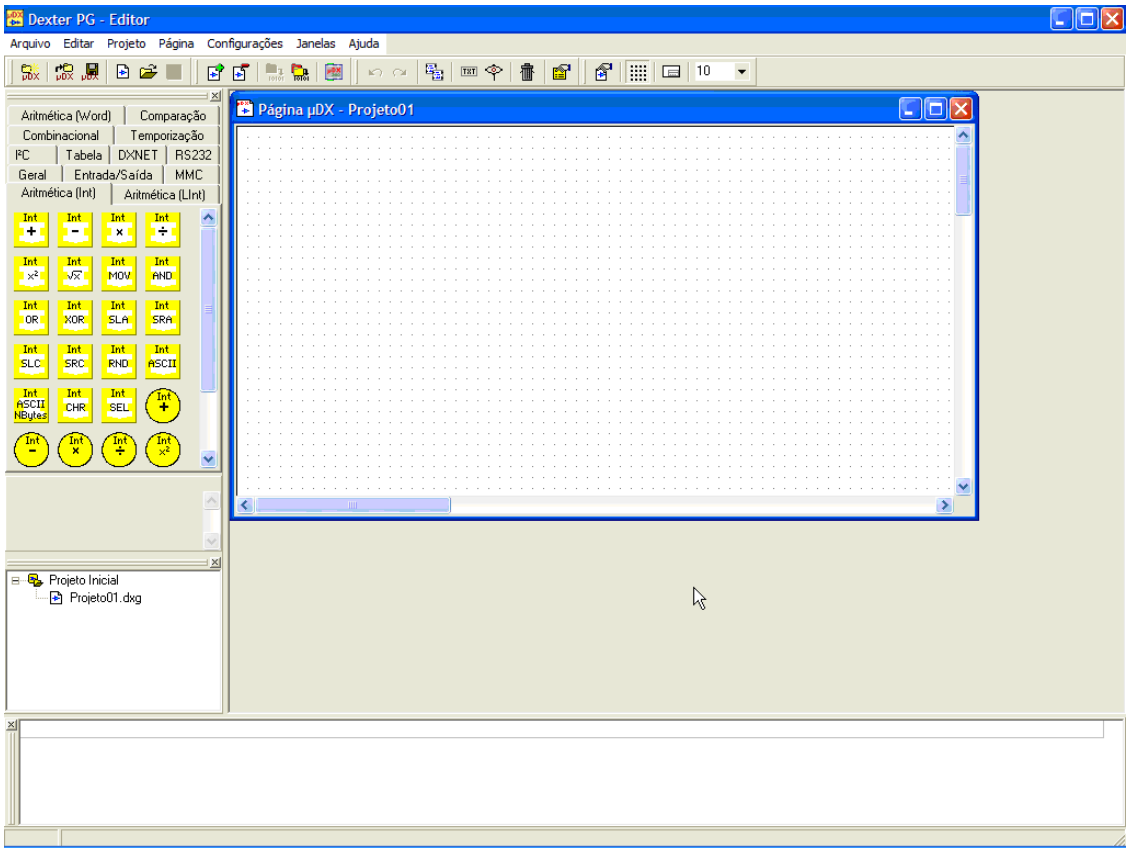
páginas, visualização de informações sobre o projeto, etc.



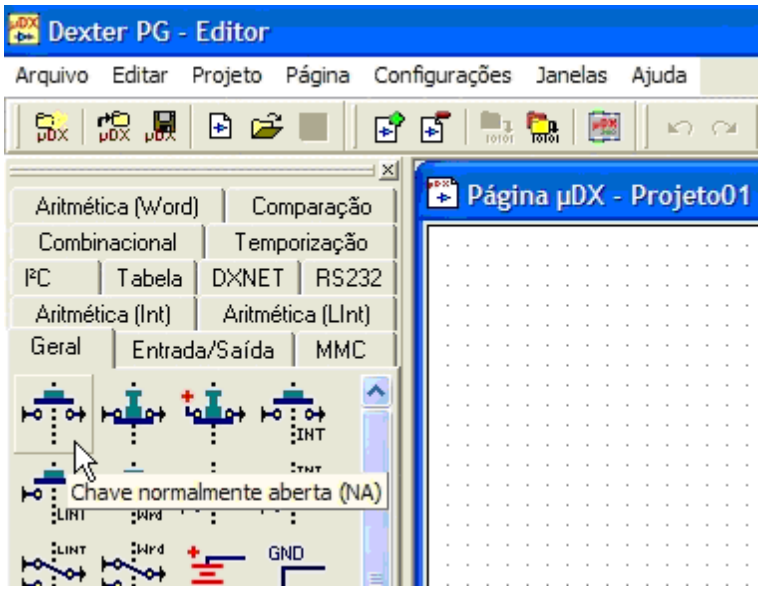
Clique em [Nova Página] para abrir uma página de programação vinculada ao projeto (ou utilize a tecla  existente na Barra de Ferramentas). Irá surgir uma janela com informações sobre a página. Preencha como abaixo:



Deverá surgir uma página de programação, chamada "Projeto01", na tela do Editor PG. Note que esta página está vinculada ao projeto chamado "Projeto Inicial". Caso não se queira gerar um projeto pode-se criar uma página avulsa apenas pressionando a tecla  existente na Barra de Ferramentas, sem a necessidade de primeiro gerar um projeto. O aspecto da tela do PG deve ser o seguinte:

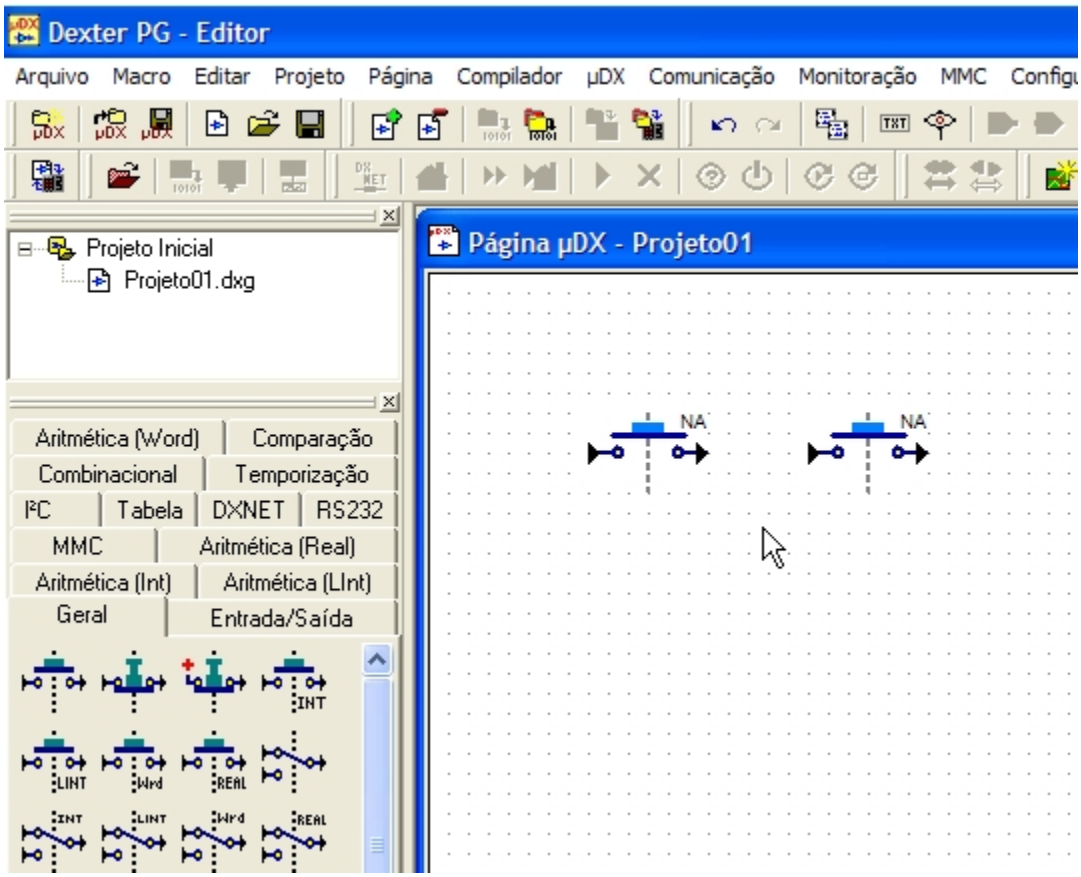


Podemos a partir deste ponto iniciar o desenvolvimento de um programa aplicativo. Digamos um programa bem simples, que ligue a saída S1 do módulo M1 de Expansão de Entradas/Saídas µDX210 sempre que as entradas E1 e E2 deste módulo forem energizadas simultaneamente. Para isso pode-se usar chaves NA (normalmente aberta), ou a função booleana AND. Elaborando o programa via chaves NA, selecione a família de blocos Geral e pressione a tecla esquerda do mouse sobre o bloco Chave NA:

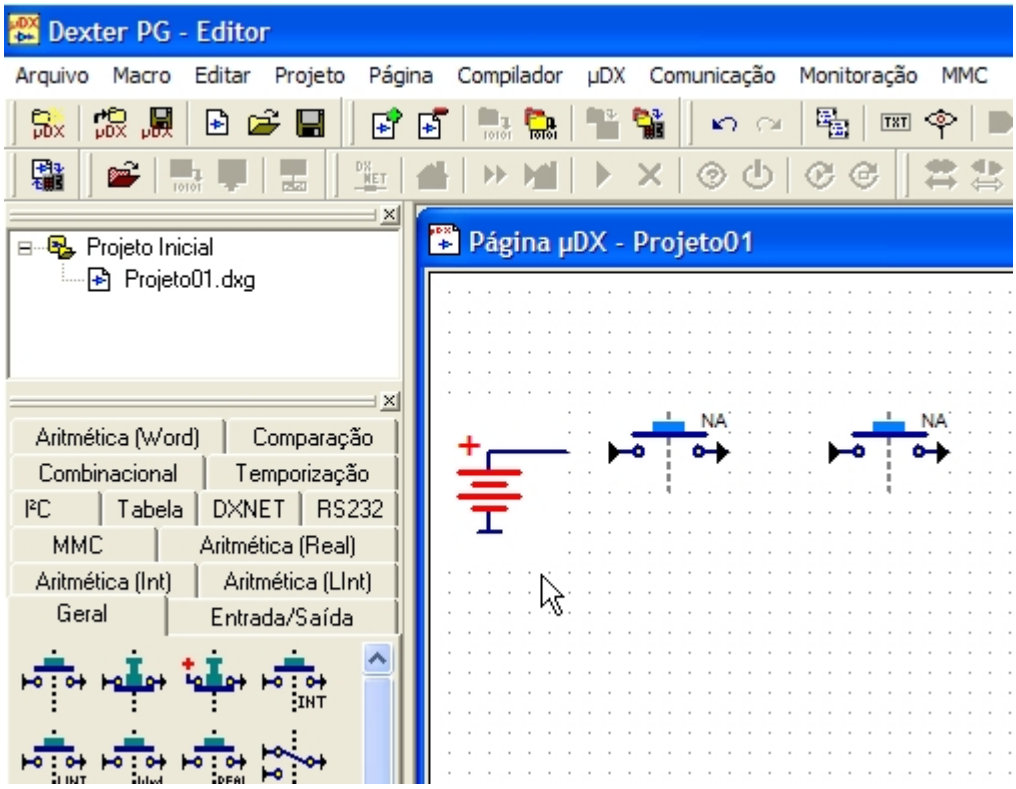


> ***Nota**: No software Editor PG deve-se pressionar a tecla esquerda do mouse uma vez sobre o bloco para capturá-lo e clicar novamente para largá-lo na janela de programação. Não arraste o bloco com a tecla esquerda do mouse pressionada.*

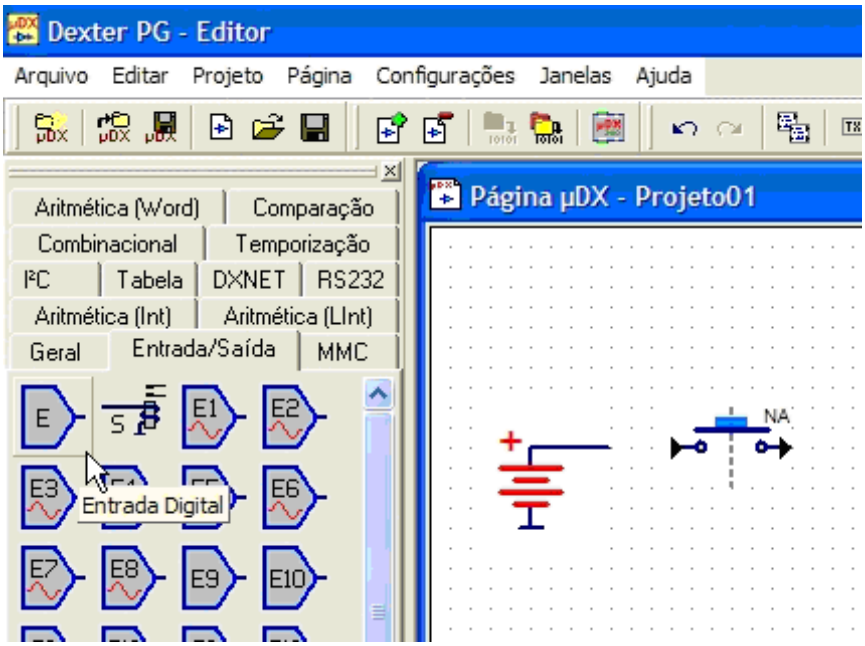
Monte desta forma a disposição mostrada abaixo:



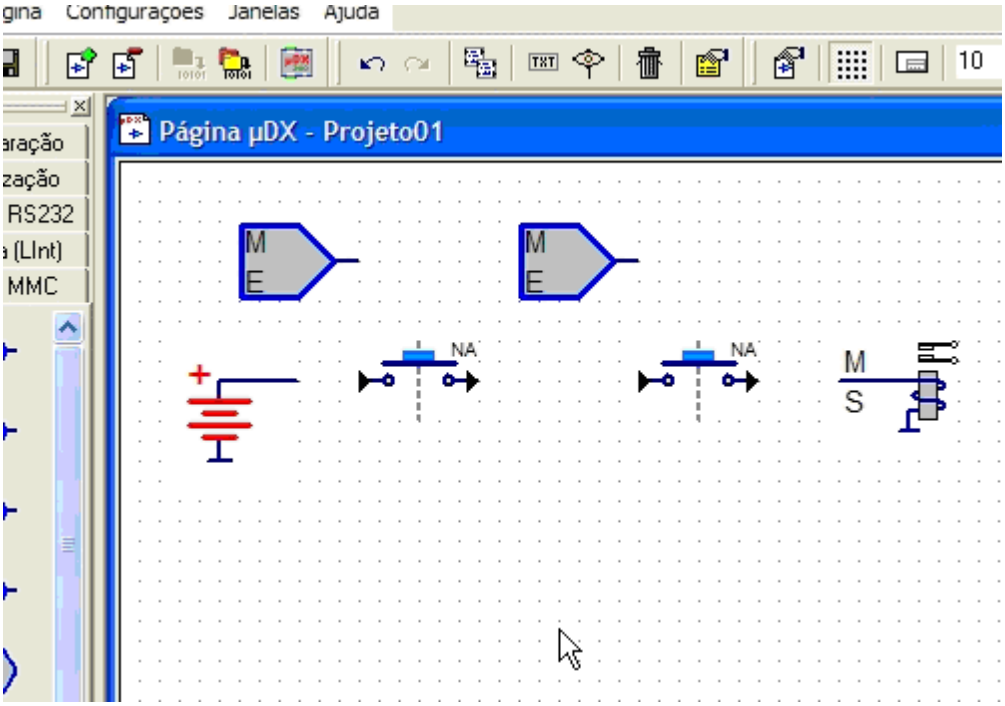
A seguir, coloque um bloco de Energia (também disponível na família Geral):



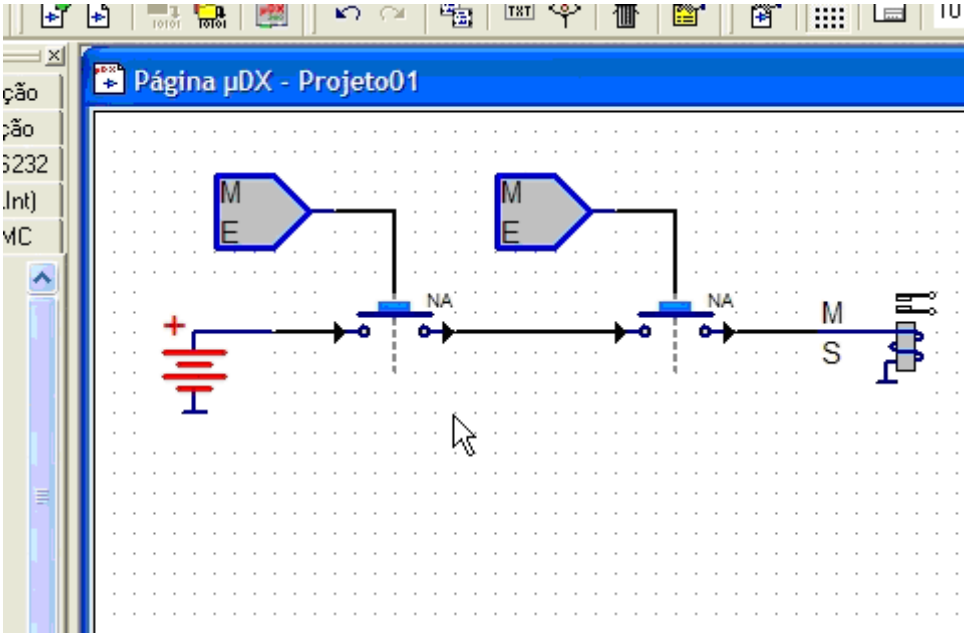
Por fim, vamos colocar os blocos de Entradas e Saídas Digitais, disponíveis na família Entrada/Saída:



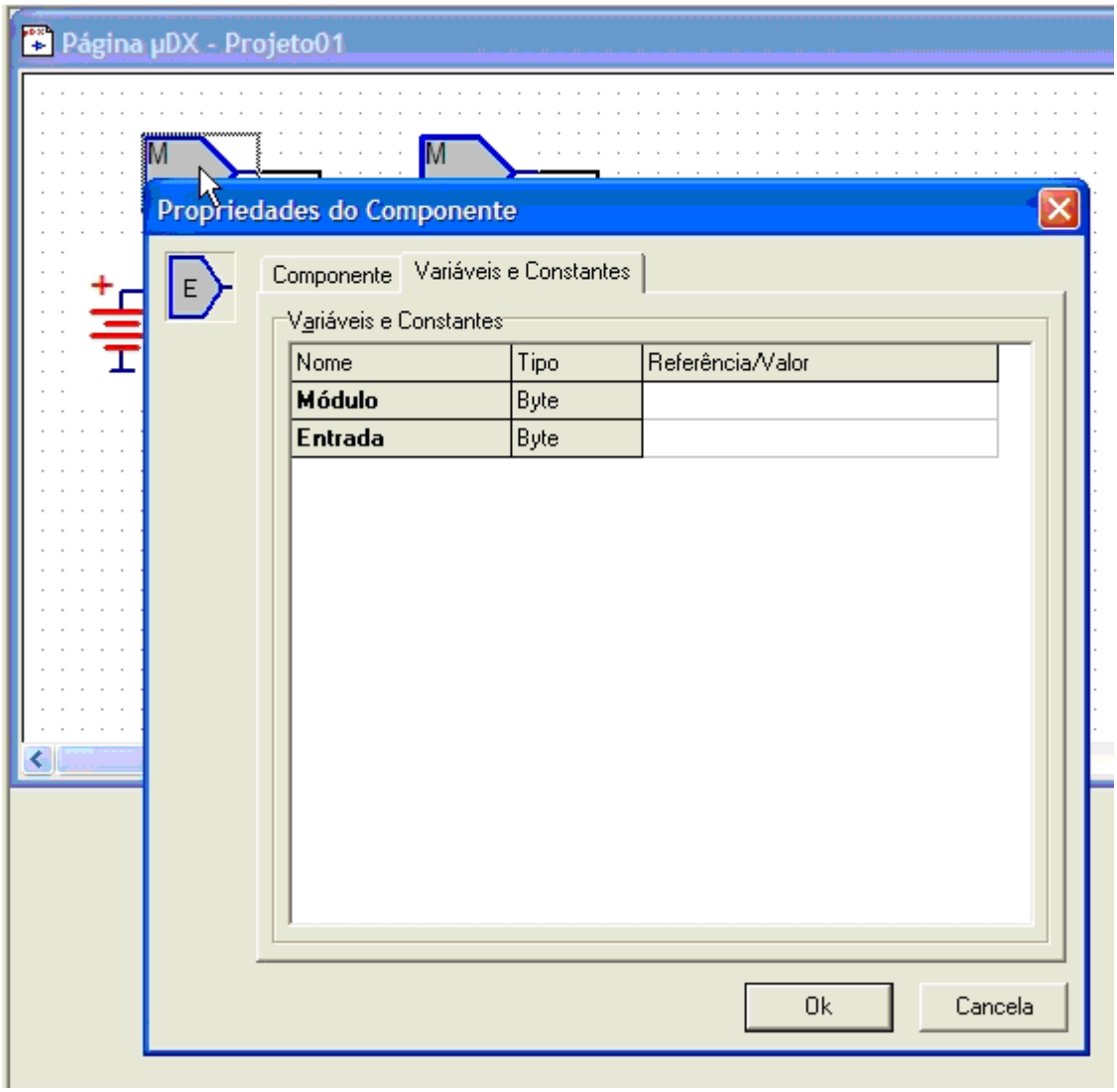
O aspecto final da disposição dos blocos na página de programação será:



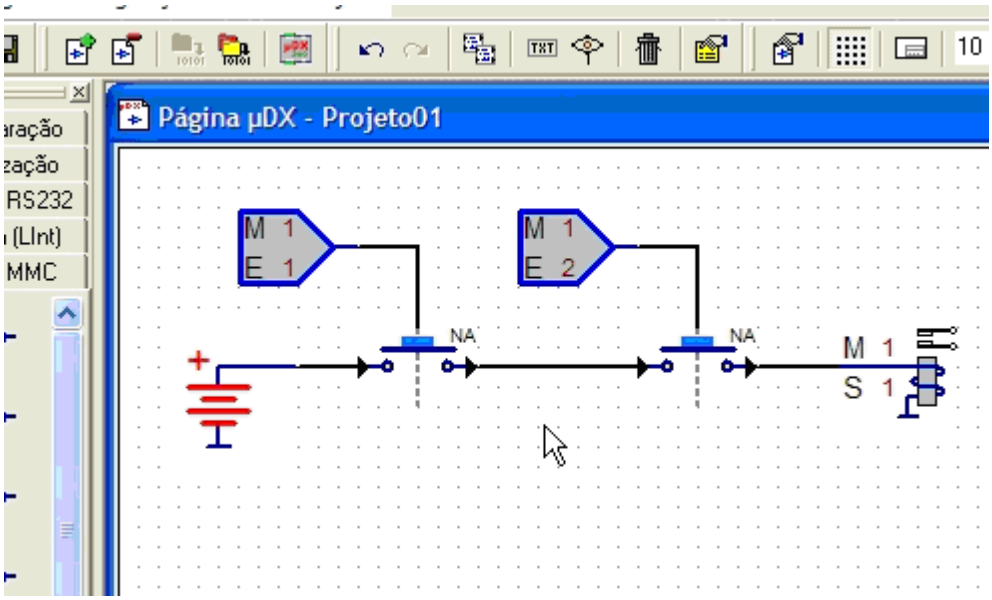
Agora basta conectar os blocos entre si e editar os blocos que possuem parâmetros. Para efetuar as conexões basta clicar com a tecla esquerda do mouse sobre as extremidades conectáveis dos blocos. Após todas as conexões o aspecto do programa é o seguinte:



Agora devemos editar os parâmetros dos blocos, no caso, os blocos de Entradas e Saídas Digitais. Para isso aponte para o bloco e pressione a tecla direita do mouse:



Edite os parâmetros destes blocos, selecionando todos com Módulo 1 (ou seja, iremos usar a primeira Expansão de Entradas/Saídas µDX210) e Entrada 1 e 2, e por fim Saída 1:



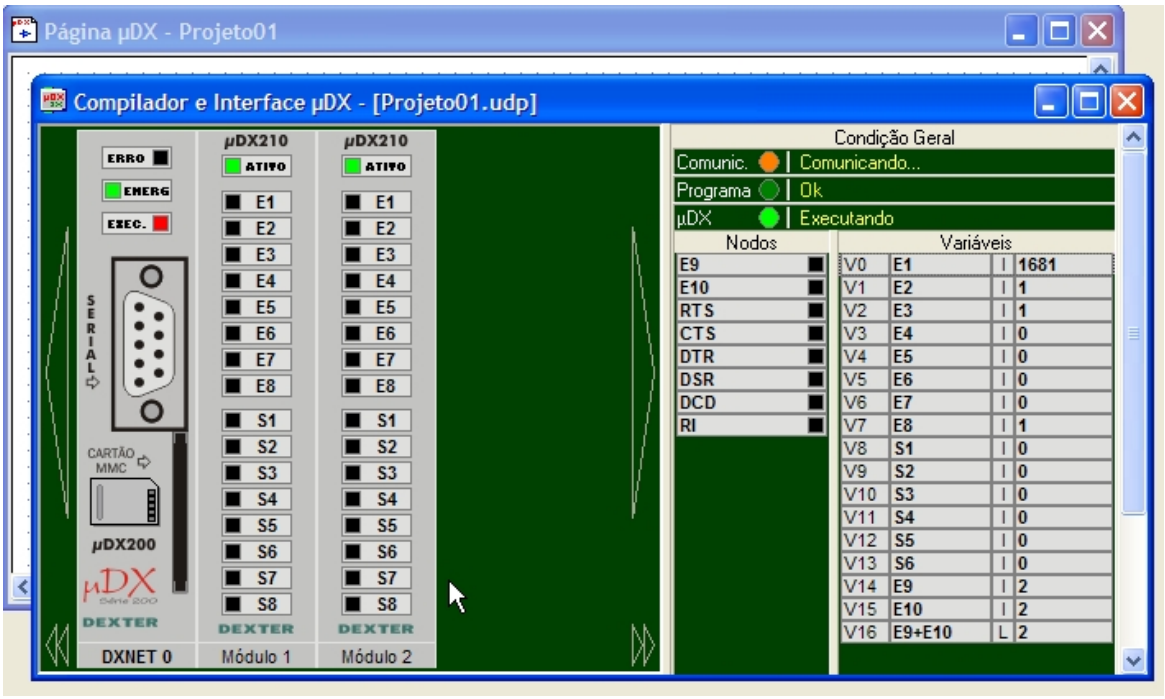
Exemplo de Projeto no PG: **Teste.dxp** (projeto); **Teste.udp** (compilado).

Está pronto o programa! Note que a Energia só alcançará a Saída S1 se as duas entradas, E1 e E2, forem energizadas simultaneamente (fechando as chaves NA).

Agora devemos pré-compilar o programa no Editor PG, gerando o arquivo sufixo UDP que será lido no Compilador PG. Para isso clique no botão (compila projeto) existente na barra de ferramentas para compilar o projeto, ou pressione a tecla **F9** do computador, ou ainda utilize o menu pop-down [**Projeto**] → [**Compilar Projeto (compilador)**]. Deve ser ativado automaticamente o Compilador PG. Para comutar entre Editor PG e Compilador PG pressione a tecla (compilador/fontes) existente na barra de ferramentas do PG, ou a tecla **F5** do computador.

Irá surgir o Compilador PG, conforme mostrado na figura a seguir. Caso o µDX200 já esteja conectado a porta serial do computador correta o Compilador PG irá mostrar os dados de status e as variáveis lidas do µDX200. Caso contrário conecte o µDX200 à porta serial do computador (ou a um cabo de adaptação USB - RS232) e siga os passos descritos no capítulo Teclas de Operação do Compilador.

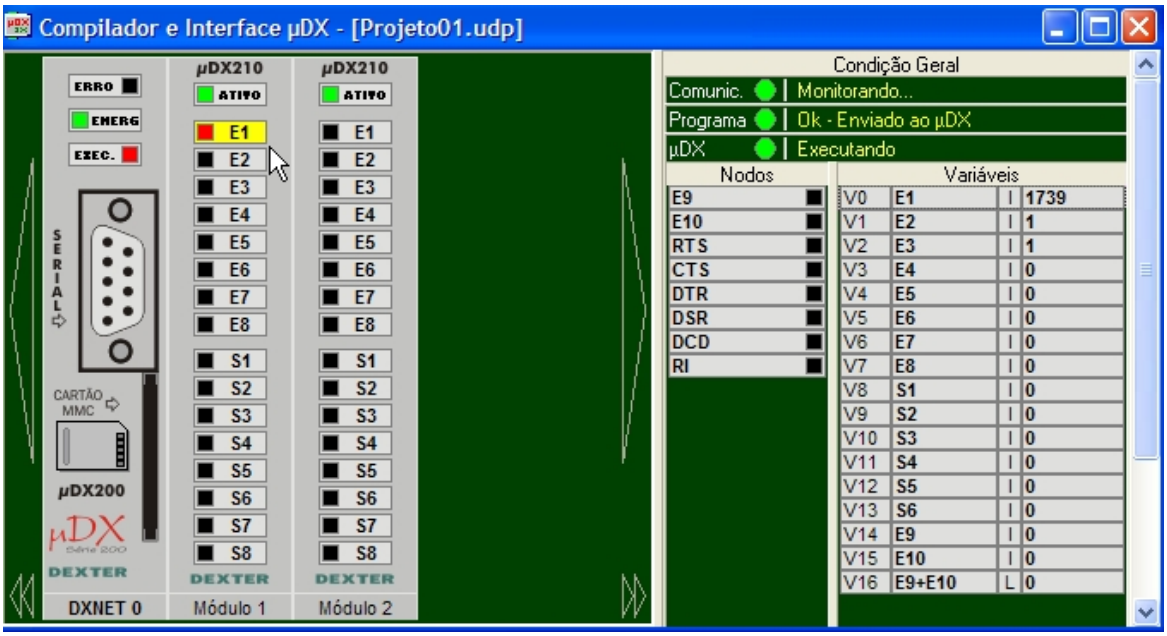
Se a comunicação foi estabelecida com o µDX200, podemos carregar o programa no controlador µDX200. Para isso pressione a tecla (enviar programa) existente no PG.



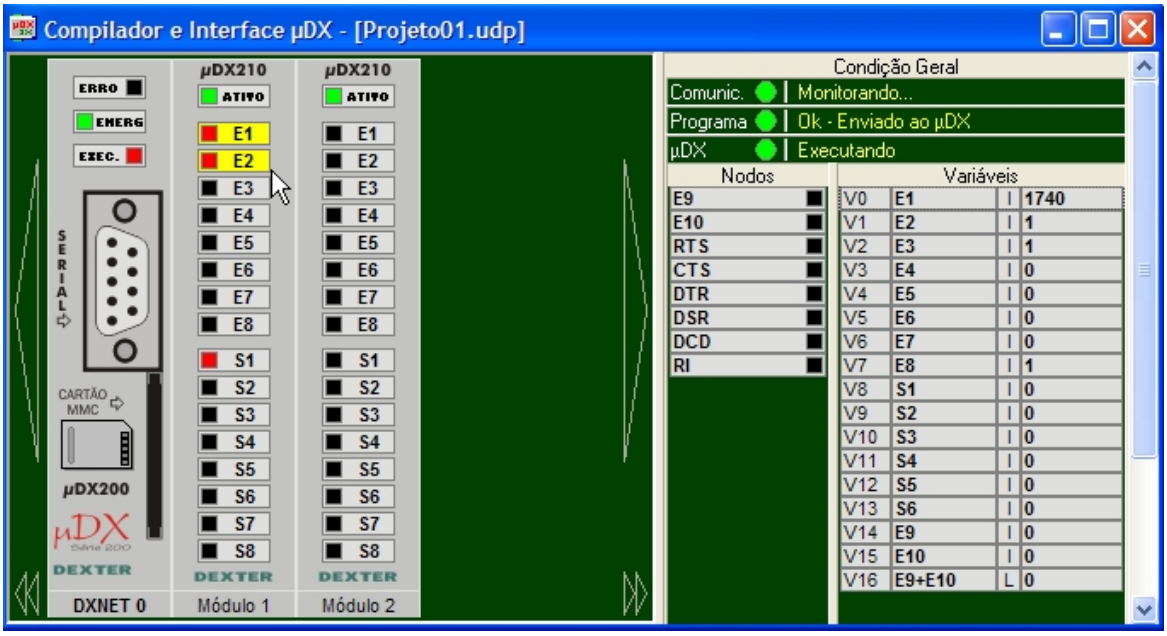
Por fim, pressione a tecla  para executar o programa transmitido. A tela do Compilador PG deve apresentar o led de Exec ligado, indicando que o programa aplicativo está sendo executado no µDX200.

Agora é possível testar o programa. Mesmo que não existam Expansões de Entradas/Saídas µDX210 instaladas, é possível acionar as entradas e ler as saídas das Expansões virtuais existentes na tela do Compilador PG.

Para isso pressione duas vezes a tecla esquerda do mouse sobre a entrada E1 do módulo 1 de Expansão µDX210. Esta entrada deve ser acionada e a moldura desta entrada ficará em amarelo. Isto sinaliza que esta entrada está ligada devido a um forçamento via Compilador:



Se fizermos o mesmo com a entrada E2 do mesmo módulo µDX210 deverá ser energizada a saída S1 do módulo, pois as duas chaves NA estarão fechadas, permitindo a energização da saída S1. Ou seja, fica bastante simples depurar o programa. Óbvio que se houver uma Expansão de Entradas/Saídas µDX210 acoplada ao controlador µDX200 é possível testar o programa efetivamente energizando as entradas E1 e E2 do µDX210.



Convenções do Compilador PG

O Compilador PG utiliza as seguintes convenções para acesso a memória, nodos e variáveis do controlador µDX200:

Mnnnn → Acessa endereço absoluto de memória do controlador.

Vnnnn → Acessa variável absoluta do controlador.

Nnnnn → Acessa nodo absoluto do controlador.

Além disso, palavras reservadas foram definidas para acessar endereços úteis do CLP. As palavras reservadas são sempre precedidas de **underline** (por exemplo, _**VCC**).

Para utilizar números hexadecimais basta incluir o sufixo **h**. Caso o número inicie por uma letra é preciso incluir um zero à esquerda para que o PG não confunda o valor literal com o nome de alguma variável. Assim, por exemplo, o valor inteiro -1365 (ou valor word 64171) em hexadecimal seria representado por **0FAABh**.

Para utilizar o valor ASCII equivalente a determinado caracter (por exemplo, nos blocos de TX-RS232 e RX-RS232 que usam stop bytes) basta indicá-lo entre **apóstrofes**. Por exemplo, para usar como stop byte o caracter A maiúsculo basta indicá-lo como **'A'** (corresponde ao valor 65 decimal, ou 41h em hexadecimal).

O controlador µDX200 utiliza os seguintes tipos de variáveis:

Nodo → Constantes e variáveis entre 0 e 1.

Byte → Apenas constantes entre 0 e 255.

Inteiro → Constantes e variáveis entre -32768 e 32767.

Word → Constantes e variáveis entre 0 e 65535.

LongInt → Constantes e variáveis entre -2.147.483.648 e 2.147.483.647.

Real → Constantes e variáveis entre -3,4E38 e 3,4E38.

String → Constantes e variáveis usadas em tabelas apenas.

Memória Absoluta (Mnnnn) e Palavras Reservadas

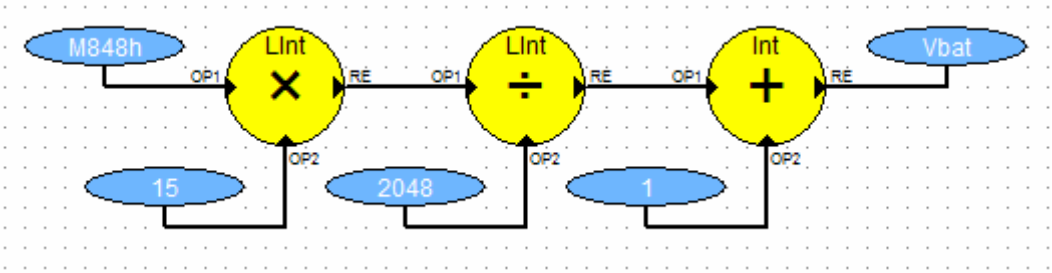
A letra M seguida de um valor numérico acessa um endereço absoluto da memória do CLP. Por exemplo, o endereço M848h (endereço hexadecimal 848h, ou 2120 em decimal) acessa a tensão de bateria interna do µDX200. A fórmula de conversão para a leitura de tensão de bateria é:

Vbat = (1,5 * Leitura) / 2048 + 0,1

Vou multiplicar todos os termos por 10, de forma a poder utilizar números inteiros, resultando em:

Vbat * 10 = (15 * Leitura) / 2048 + 1

O programa abaixo apresenta na variável Vbat a tensão de bateria interna do µDX200 (multiplicada por 10).



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - Mnnnn.dxg**

Note que foi utilizado um bloco de multiplicação longint, pois o conteúdo da posição de memória M848h multiplicado por 15 pode exceder a faixa representável pelos números inteiros (-32768 a 32767). A seguir um bloco de divisão longint reconverte o valor para inteiro (uma vez que este bloco utiliza como dividendo um longint, como divisor um inteiro, e resulta em um quociente inteiro).

Outra maneira de acessar endereços específicos de memória do controlador µDX200 é utilizando palavras reservadas. Note que as palavras reservadas ao final da lista a seguir são válidas apenas para µDX201, pois apenas este CLP permite conexão de Interface Homem/Máquina (IHM).

Também as portas seriais RS232-2 e RS232-3 estão disponíveis apenas em controladores µDX201 versão 3.46 ou superior.

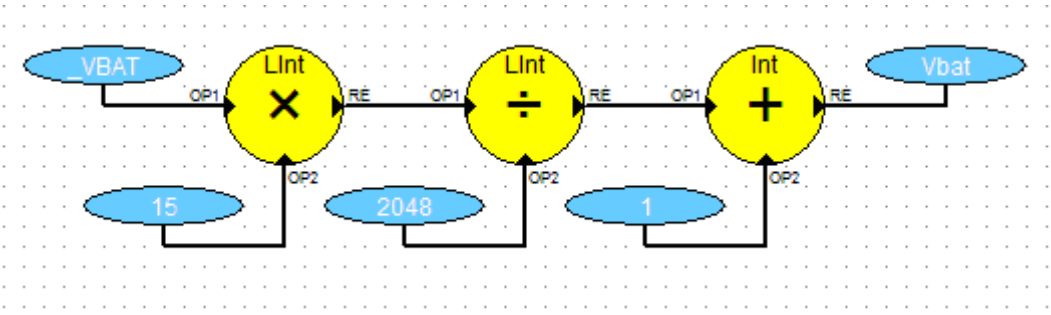
Elas sempre iniciam por _ (underline). As seguintes palavras reservadas estão disponíveis no Compilador PG:

_RXBUF	**Endereço do buffer de recepção serial (RS232).**
_TXBUF	**Endereço do buffer de transmissão serial (RS232).**
_DIAMES	**Dia e mês do relógio de tempo real (em BCD).**
_ANO	**Ano do relógio de tempo real (em BCD).**
_HORASEM	**Hora e dia da semana do relógio de tempo real (BCD).**
_SEGMIN	**Segundo e minuto do relógio de tempo real (BCD).**
_VBAT	**Tensão de bateria.**
_ERRODX	**Endereço DXNET sem resposta.**
_TCICLO	**Tempo de ciclo do CLP (em intervalos de 0,25ms).**
_RXQNT	**Quantidade de bytes recebidos pela serial (RS232).**
_ERROMMC	**Tipo de erro e tamanho do cartão MMC (hexa).**
_TXRDPT	**Pointer de leitura do buffer de TX serial (RS232).**
_TXWRPT	**Pointer de escrita do buffer de TX serial (RS232).**
_TEMP	**Temperatura interna do µDX200.**
_VCC	**Tensão de alimentação do µDX200.**

_VARIN	**Endereço de início da tabela de variáveis IN.**
_VAROUTPT	**Pointer para início da tabela de variáveis OUT.**
_DXNET	**Endereço DXNET do µDX200.**
_SENHA	**Temporização para exigir senha.**
_VINT	**Tensão de alimentação interna do CLP.**
_X	**Coordenada X de sensor touchscreen da IHM (apenas µDX201).**
_Y	**Coordenada Y de sensor touchscreen da IHM (apenas µDX201).**
_IN1	**Entrada analógica 1 da IHM (apenas µDX201).**
_IN2	**Entrada analógica 2 da IHM (apenas µDX201).**
_CONTRASTE	**Contraste do display da IHM (apenas µDX201).**
_E1	**Leitura entrada analógica E1 em alta resolução (apenas µDX201).**
_E9	**Leitura de freqüência na entrada digital E9 (apenas µDX201).**
_E10	**Leitura de freqüência na entrada digital E10 (apenas µDX201).**
_P9	**Leitura de período na entrada digital E9 em intervalos de 125µs (apenas µDX201).**
_P10	**Leitura de período na entrada digital E10 em intervalos de 125µs (apenas µDX201).**
_RXBUF2	**Endereço do buffer de recepção serial 2 (RS232-2) (apenas µDX201).**
_RXBUF3	**Endereço do buffer de recepção serial 3 (RS232-3) (apenas µDX201).**
_TXBUF2	**Endereço do buffer de transmissão serial 2 (RS232-2) (apenas µDX201).**
_TXBUF3	**Endereço do buffer de transmissão serial 3 (RS232-3) (apenas µDX201).**
_RXQNT2	**Quantidade de bytes recebidos pela serial 2 (RS232-2) (apenas µDX201).**
_RXQNT3	**Quantidade de bytes recebidos pela serial 3 (RS232-3) (apenas µDX201).**
_TXPT2	**Pointer de escrita (MSB) e leitura (LSB) do buffer de TX serial 2 (RS232-2) (apenas µDX201).**
_TXPT3	**Pointer de escrita (MSB) e leitura (LSB) do buffer de TX serial 3 (RS232-3) (apenas µDX201).**
_CFGRS232	**Leitura de configuração de porta serial RS232 (apenas µDX201).**
_CFGDXNET	**Leitura de endereço na rede DXNET (apenas µDX201).**
_NUMSER	**Leitura de número de série do controlador (apenas µDX201).**
_FIRMWARE	**Leitura de versão de firmware do controlador (apenas µDX201).**
_E2	**Leitura entrada analógica E2 em alta resolução (apenas µDX201).**
_E3	**Leitura entrada analógica E3 em alta resolução (apenas µDX201).**
_E4	**Leitura entrada analógica E4 em alta resolução (apenas µDX201).**
_E5	**Leitura entrada analógica E5 em alta resolução (apenas µDX201).**
_E6	**Leitura entrada analógica E6 em alta resolução (apenas µDX201).**
_E7	**Leitura entrada analógica E7 em alta resolução (apenas µDX201).**
_E8	**Leitura entrada analógica E8 em alta resolução (apenas µDX201).**
_PULSO1	**Número de pulsos a ser gerado na saída analógica S1 (apenas µDX201).**
_PULSO2	**Número de pulsos a ser gerado na saída analógica S3 (apenas µDX201).**
_PULSO3	**Número de pulsos a ser gerado na saída analógica S5 (apenas µDX201).**

_POSIC1	**Pulso atual (posição) do motor ligado à saída analógica S1 (apenas µDX201).**
_POSIC2	**Pulso atual (posição) do motor ligado à saída analógica S3 (apenas µDX201).**
_POSIC3	**Pulso atual (posição) do motor ligado à saída analógica S5 (apenas µDX201).**
_VELRG1	**Velocidade de regime para motor ligado à saída analógica S1 (apenas µDX201).**
_VELRG2	**Velocidade de regime para motor ligado à saída analógica S3 (apenas µDX201).**
_VELRG3	**Velocidade de regime para motor ligado à saída analógica S5 (apenas µDX201).**
_VELOC1	**Velocidade atual do motor ligado à saída analógica S1 (apenas µDX201).**
_VELOC2	**Velocidade atual do motor ligado à saída analógica S3 (apenas µDX201).**
_VELOC3	**Velocidade atual do motor ligado à saída analógica S5 (apenas µDX201).**
_MOTOR	**Status do controle de motores via saídas analógicas S1,S3 e S5.**

Assim, o programa anterior, que lia o valor de tensão de bateria e a convertia em tensão em volts multiplicado por 10, pode ser escrito usando uma palavra reservada, em vez do endereço absoluto M848h:



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - VBAT.dxg**

A seguir, vamos descrever cada um destes endereços com mais detalhes:

_RXBUF

Este endereço é o início do buffer de recepção serial (RS232). Equivale ao endereço absoluto M780h para o caso do µDX200 (para µDX201 equivale a M2900h). Este é o endereço inicial do buffer de recepção serial (RS232). Isto permite ler os dados recebidos via serial e analisá-los. Veja um exemplo de uso na descrição do bloco RS232 - RX N Bytes. O buffer de recepção serial não é circular, ou seja, a cada recepção o buffer é limpo e, portanto, o primeiro caracter recebido está sempre localizado em M780h. Note que se for usado o endereço absoluto o programa fica dependente da plataforma (µDX200 ou µDX201), enquanto se usarmos _RXBUF basta recompilar o programa para cada um dos tipos de CLP.

_TXBUF, _TXRDPT, _TXWRPT

_TXBUF é o endereço de início do buffer de transmissão serial (RS232). Equivale ao endereço absoluto M7C0h no µDX200 (M2A00h para µDX201). No caso do buffer de transmissão trata-se de buffer circular e, portanto, não necessariamente este endereço possui o primeiro caracter a ser transmitido, nem tampouco é o endereço inicial para escrita do buffer. Para fazer a leitura do buffer de transmissão serial devemos usar o apontador (pointer) de leitura do buffer de TX serial _TXRDPT (equivalente ao endereço absoluto M85Eh no µDX200, ou M2B5Eh no µDX201). Para calcular a primeira posição do buffer de transmissão serial devemos somar _TXBUF com _TXRDPT + 1. Ou seja:

Primeira posição do buffer TX = _TXBUF + _TXRDPT + 1

Para varremos o buffer de TX serial basta usar uma variável como indexador. Se for usada a variável N a equação fica:

Posição N do buffer TX = _TXBUF + [(_TXRDPT+ N + 1) AND 3Fh]

O programa a seguir exemplifica o uso destes endereços. A idéia é ler um sensor de temperatura ligado a rede I2C do µDX200, e montar uma mensagem a ser transmitida periodicamente via porta serial (a cada segundo). A mensagem deve ter o seguinte formato (7 bytes):

STX	**Temp.**	**Temp.**	**"."**	**Temp.**	**ETX**	**BCC**
02h	**(dezena)**	**(unidade)**	**2Eh**	**(décimos)**	**03h**	

O BCC é um byte de checagem da mensagem calculado fazendo-se XOR de cada byte da mensagem com o próprio BCC (que é inicializado com 00h) e shift left 1 posição do BCC. Por exemplo, se a mensagem for (para uma temperatura ambiente de 22,5°C):

02h 32h 32h 2Eh 35h 03h 42h

O BCC calculado foi:

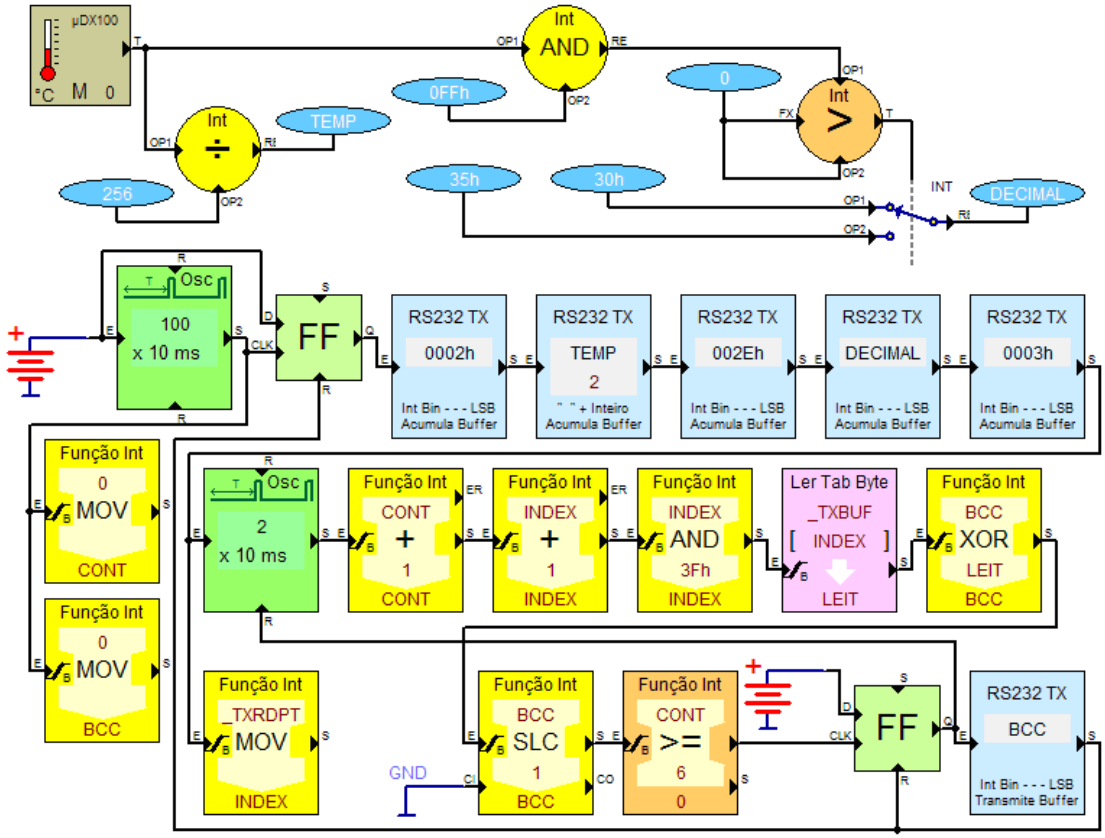
00h XOR 02h → 02h shift left → 04h
04h XOR 32h → 36h shift left → 6Ch
6Ch XOR 32h → 5Eh shift left → BCh
BCh XOR 2Eh → 92h shift left → 24h
24h XOR 35h → 11h shift left → 22h
22h XOR 03h → 21h shift left → 42h

O programa a seguir implementa esta funcionalidade no controlador µDX200. Este é um bom exemplo da flexibilidade do µDX200 para gerar qualquer protocolo de comunicação. Note que o buffer de transmissão serial é montado com a mensagem e, logo após, varrido para cálculo do BCC, que é incorporado à mensagem e esta é transmitida.

Note que a temperatura lida no sensor para µDX100 é composta de dois bytes, sendo o byte MSB os graus celsius, enquanto o byte LSB representa a parte fracionária (com resolução de 0,5°C). Por isso, esta informação foi separada em TEMP e DECIMAL. Caso a parte decimal seja diferente de zero já se atribui valor "5" (35h), caso contrário se atribui valor "0" (30h).

A cada segundo o oscilador gera um pulso que liga a saída do primeiro FF (flip-flop), que por sua vez dispara a montagem do buffer de transmissão (os 6 bytes da mensagem). Note que os blocos de TX RS232 são do tipo Acumula Buffer, ou seja, eles apenas colocam os dados no buffer de transmissão serial, mas não disparam a transmissão.

Feito isso, é atribuído a variável INDEX o valor do pointer de leitura de buffer de TX RS232, e o segundo oscilador (de 20ms) é habilitado. A cada pulso desse, CONT e INDEX são incrementados e é feita a leitura dos bytes contidos no buffer de TX. Estes bytes são usados em operação XOR com BCC, BCC é deslocado uma posição à esquerda (shift left), e é testado se chegou ao fim da mensagem (CONT≥6). Se sim, é acionado o segundo FF, inibe-se o oscilador de 20ms e é inserido o BCC na mensagem e esta é transmitida. Após transmissão os FFs são resetados.



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - TXBUF, TXRDPT, TXWRPT.dxg**

A palavra reservada _TXWRPT aponta para o ponto de escrita do buffer de transmissão serial. Equivale ao endereço absoluto M860h no µDX200, ou M2B60h no µDX201.

> ***ATENÇÃO:*** *Caso esse programa seja usado em Controlador µDX201 é necessário modificar o bloco (INDEX AND 3Fh) para (INDEX AND 0FFh). Isso porque o buffer de transmissão serial do µDX201 possui 256 bytes e não apenas 64, como no caso do µDX200.*

_DIAMES, _ANO, _HORASEM, _SEGMIN

Estas palavras reservadas permitem acessar os dados do relógio e calendário internos do controlador µDX200. Elas são representadas como valores BCD. Assim, _ANO irá assumir o valor 2006h se estivermos neste ano (em decimal 8198). Já se for dia 30 de abril a posição _DIAMES irá conter o valor 3004h. Estas palavras reservadas permitem efetuar operações com o conteúdo do relógio e calendário internos do µDX200. Os endereços absolutos correspondentes são:

_DIAMES	M840h (µDX200)	M2B40h (µDX201)
_ANO	M842h (µDX200)	M2B42h (µDX201)
_HORASEM	M844h (µDX200)	M2B44h (µDX201)
_SEGMIN	M846h (µDX200)	M2B46h (µDX201)

No exemplo a seguir se utiliza as variáveis absolutas V29 a V35 para especificar data e hora a ser escrita no relógio de tempo real (RTC) do µDX200. Foram usadas variáveis absolutas para

permitir que um software supervisório externo tenha acesso a elas e possa acertar o horário e calendário do controlador. Ao energizar a entrada digital E9 os valores gravados nas variáveis são transferidos para o relógio de tempo real do µDX200.

> ***ATENÇÃO:*** Esta rotina foi transformada em macro, facilitando sua aplicação. Veja macro **Grava Relógio**.

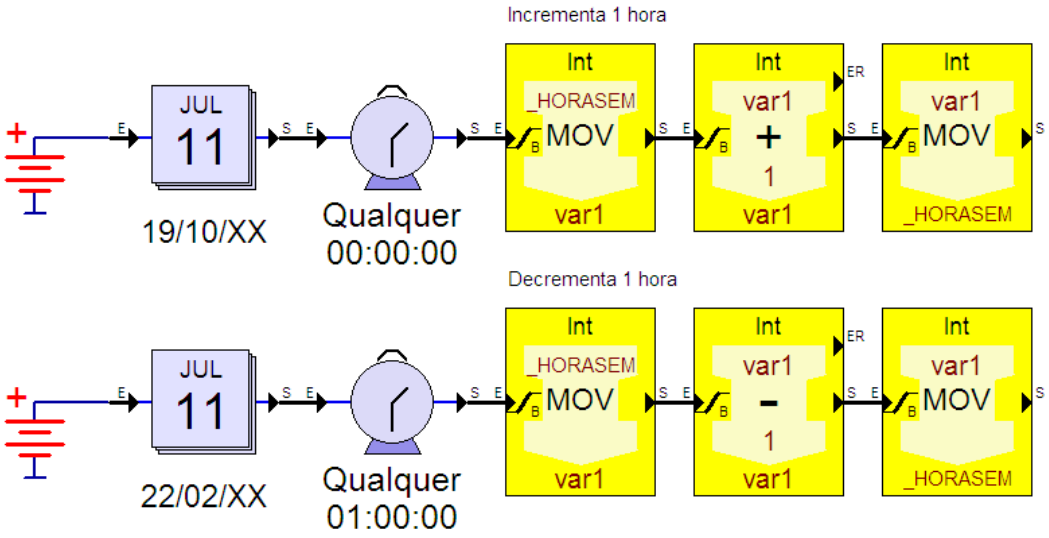
Exemplo de Programa Aplicativo: **Convenções do Compilador PG - RTC.dxg**

O exemplo anterior permite escrever um horário e data no RTC (relógio de tempo real) do controlador µDX200 ou µDX201. Já o exemplo a seguir lê o valor do relógio interno e transfere o valor lido a cada segundo para as variáveis DIA, MES, ANO, HORA, MIN, SEG. Como o RTC é codificado em BCD o programa necessita fazer alguns cálculos para converter em valores numéricos simples cada um dos dados.

ATENÇÃO: Esta rotina foi transformada em macro, facilitando sua aplicação. Veja macro **Lê Relógio**.

Exemplo de Programa Aplicativo: **Convenções do Compilador PG - Relogio_Leitura.dxg**

Por fim, um exemplo de como implementar horário de verão no µDX200. No dia 19 de outubro ele incrementa uma hora ao relógio interno do µDX200, e no dia 22 de fevereiro ele decrementa uma hora.



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - Horario_Verao.dxg**

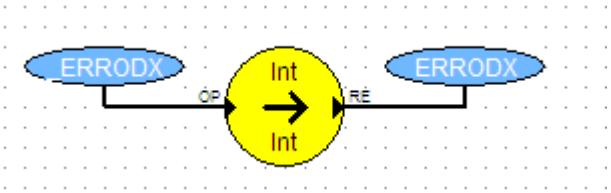
_VBAT

Contém a tensão da bateria interna do µDX200. Este valor deve estar entre 3,2V e 2,5V. Abaixo deste valor a bateria deve ser substituída (modelo CR2032). _VBAT corresponde ao endereço absoluto M848h para o µDX200, como já visto algumas páginas anteriores (ou M2B48h no µDX201). A fórmula para conversão do valor lido neste endereço para tensão de bateria em volts é:

Vbat = (1,5 * Leitura) /2048 + 0,1

_ERRODX

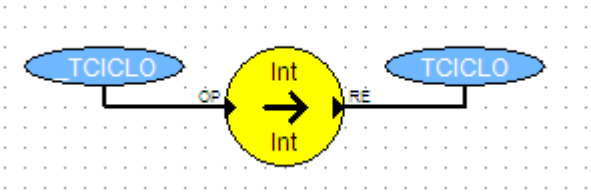
Indica um endereço DXNET que consta da tabela de varredura (TABSCAN), mas não está respondendo ao mestre da rede DXNET. Corresponde ao endereço absoluto M84Ah para o µDX200 (M2B4Ah para µDX201).



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - ERRODX.dxg**

_TCICLO

Retorna o tempo de execução de um ciclo completo do programa aplicativo, em intervalos de 250µs. Por exemplo, se o valor lido for 4 significa que o programa aplicativo está sendo executado em 1ms. Corresponde ao endereço absoluto M84Eh para µDX200 (M2B4Eh para µDX201).



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - TCICLO.dxg**

_RXQNT

Indica a quantidade de bytes existentes no buffer de recepção serial (RS232) e, portanto, o número de bytes recebidos pela porta RS232. Corresponde ao endereço absoluto M850h para µDX200 (M2B50h para µDX201).

_ERROMMC

Especifica qual o erro detectado no controlador µDX200 e o tamanho do cartão MMC (ou SD, no caso de µDX201) conectado ao controlador µDX200. Corresponde ao endereço absoluto M854h para µDX200, ou M2B54h para µDX201. Note que a informação de erro detectado no controlador é dada no byte LSB, enquanto o tamanho do cartão MMC é informado no byte MSB. Os códigos de erro são os seguintes:

00h → Nenhum erro detectado (led de erro apagado).
08h → Erro de CRC no programa aplicativo (1 piscada no led de erro).
0Ch → Erro de versão do programa aplicativo (2 piscadas no led de erro).

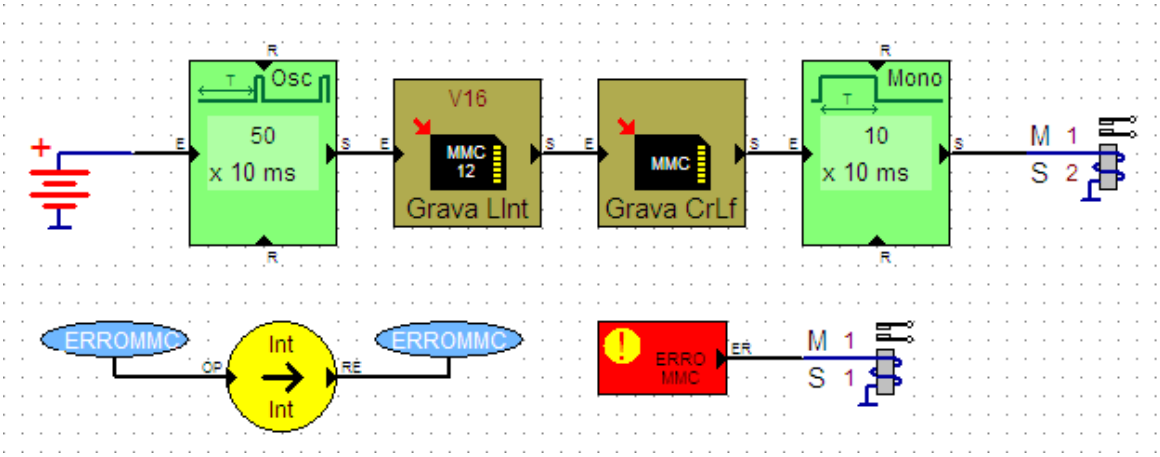
0Eh → Não foi encontrado arquivo no cartão MMC (3 piscadas no led de erro).
0Fh → Arquivo cheio no cartão MMC (3 piscadas no led de erro).
10h → Tipo ou tamanho de cartão MMC não suportado (4 piscadas no led de erro).
11h → Cartão MMC com formatação incorreta (4 piscadas no led de erro).

Conforme o código de erro o led de erro existente no controlador pisca um determinado número de vezes em um ciclo total de aproximadamente 2,5 segundos. Cada piscada dura cerca de 100ms. Com isso é possível ter-se uma indicação visual no próprio µDX200 de que existe erro, e qual sua natureza. Existe ainda uma indicação visual de erro de hardware via led de erro: caso o led pisque rapidamente de forma constante (cerca de 3 vezes por segundo, ou seja, 3 Hz) significa falha no oscilador interno de alta freqüência. Neste caso o CLP deverá ser encaminhado à Dexter para manutenção.

Já o byte MSB indica o tamanho do cartão MMC instalado no µDX200 via os seguintes códigos:

00h → Nenhum cartão MMC detectado.
01h → Cartão de 32 MBytes detectado
02h → Cartão de 64 MBytes detectado.
03h → Cartão de 128 MBytes detectado.
04h → Cartão de 256 MBytes detectado.
05h → Cartão de 512 MBytes detectado.
06h → Cartão de 1 GByte detectado.
07h → Cartão de 2 GBytes detectado.

Por exemplo, se a leitura da palavra reservada _ERROMMC resultou no valor hexadecimal 050Fh significa que foi detectado um cartão MMC de 512 MBytes instalado no controlador µDX200 (byte MSB = 05h), mas o cartão está com o arquivo de log cheio (byte LSB = 0Fh).

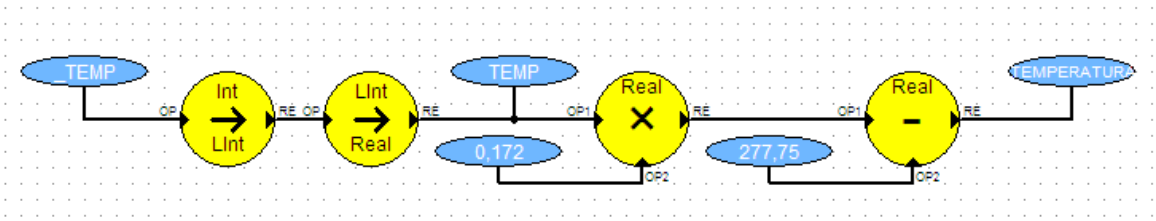


Exemplo de Programa Aplicativo: **Convenções do Compilador PG - ERROMMC.dxg**

_TEMP

Permite acessar a temperatura interna do controlador µDX200. Corresponde ao endereço M154h tanto no µDX200 como no µX201. Isso pode ser útil, por exemplo, para acionar uma saída caso esta temperatura ultrapasse determinados limites. Para converter o valor lido para a temperatura em graus celsius utilize a seguinte fórmula:

Temp = 0,172 * Leitura - 277,75

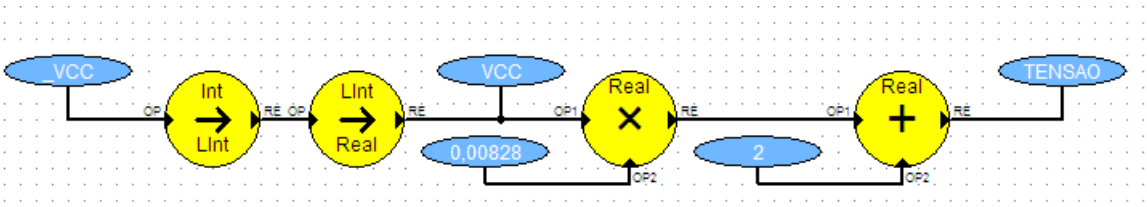


Exemplo de Programa Aplicativo: **Convenções do Compilador PG - TEMP.dxg**

_VCC

Acessa a tensão de alimentação do controlador µDX200. Corresponde ao endereço M150h tanto no µDX200 como no µX201. Use a fórmula abaixo para converter o valor lido em tensão de alimentação em volts:

V = 0,00828 * Leitura + 2



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - VCC.dxg**

_VARIN, _VAROUTPT

_VARIN é o endereço inicial da tabela de variáveis de entrada (IN) do programa aplicativo. Esta tabela é sempre escrita com os valores de variáveis calculados pelo programa aplicativo (que são colocados na tabela de variáveis de saída). Por isso, para modificar uma variável do controlador µDX200 é necessário acessar a tabela de variáveis de saída (OUT). A palavra reservada _VAROUTPT aponta para o início da tabela de variáveis OUT em relação ao início da tabela de variáveis IN.

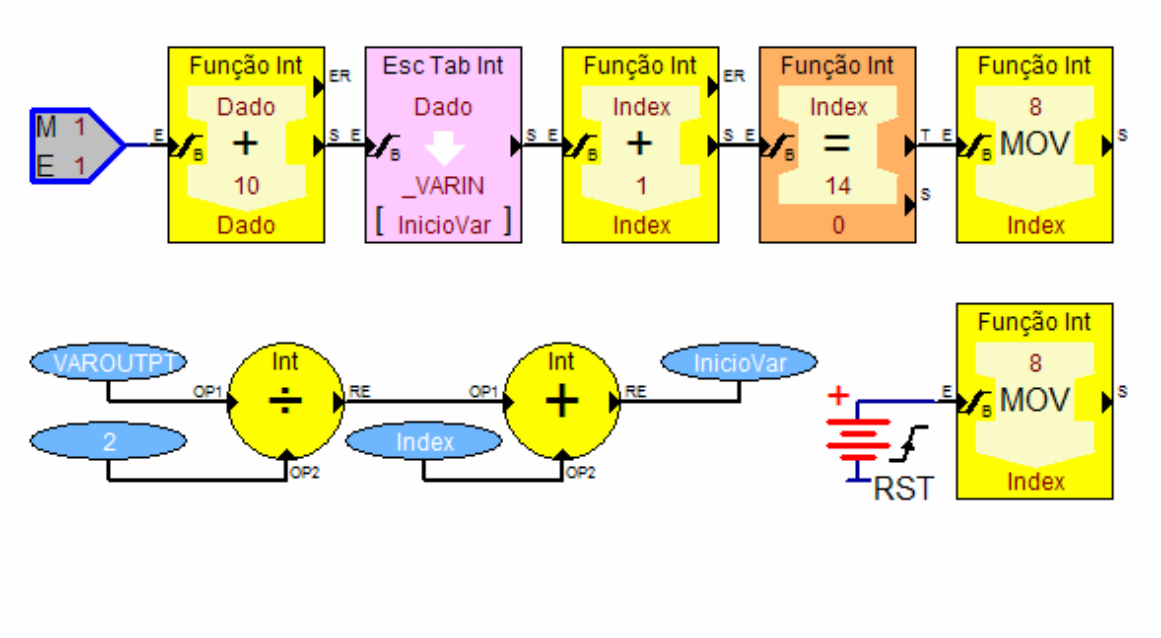
Para calcular o endereço de início da tabela de variáveis OUT use a fórmula:

Início das Variáveis OUT = _VARIN + (_VAROUTPT / 2)

Ou seja, uma tabela que inicie neste endereço irá acessar, na primeira posição a variável V0, na segunda posição V1, e assim por diante. Podemos, portanto, indexar variáveis por este método. O programa mostrado a seguir grava valores incrementados de 10 em 10 a partir da variável V8 (saída analógica S1), cada vez que a entrada digital E1 do primeiro módulo de Expansão é acionada. Ao chegar em V13 (saída analógica S6) o programa retorna o acesso à V8.

Os endereços correspondentes são os seguintes:

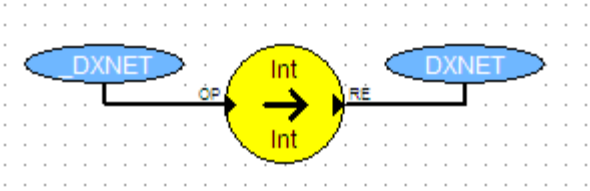
_VARIN	M200h (µDX200)	M1100h (µDX201)
_VAROUTPT	M1212h (µDX200)	M3112h (µDX201)



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - VARIN, VAROUTPT.dxg**

_DXNET

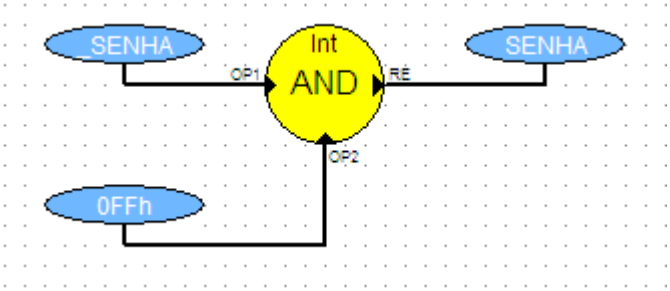
Retorna o endereço DXNET do µDX200. Corresponde ao endereço M8000h no µDX200, ou M0EA00h no caso do µDX201.



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - DXNET.dxg**

_SENHA

Permite acessar o contador decrescente para temporização de passagem do modo Programação para o modo Normal na comunicação serial. Note que, ao comutar para modo Normal, o programa aplicativo assume o controle da porta serial RS232 do µDX200, e para retornar ao modo Programação é necessário envio da senha (senha padrão **[uDX200**). A temporização é contada em segundos. Assim, se atribuirmos a _SENHA o valor 60 o µDX200 esperará um minuto sem comunicação serial para comutar automaticamente do modo Programação (led de EXEC sempre ativado) para o modo Normal (led de EXEC piscando). Este dado ocupa apenas o byte LSB da palavra reservada _SENHA. Assim, convém mascarar o byte MSB, evitando que venha valores indesejados no byte superior. Corresponde ao endereço M876h no µDX200, ou M2B76h no caso do µDX201.

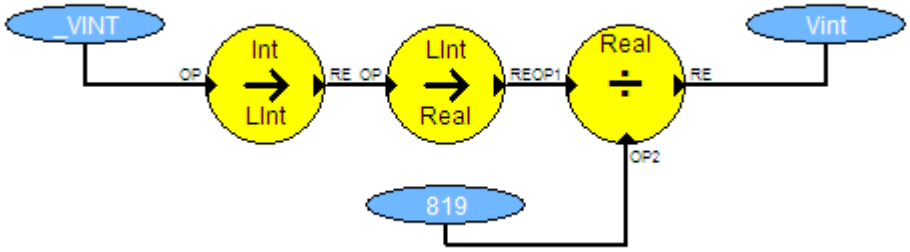


Exemplo de Programa Aplicativo: **Convenções do Compilador PG - SENHA.dxg**

_VINT

Retorna a tensão de alimentação interna do controlador µDX200 ou µDX201 (normalmente em cerca de 3,3V). Corresponde ao endereço M156h tanto no µDX200 como no µDX201. Use a fórmula abaixo para converter o valor lido em tensão de alimentação interna em volts:

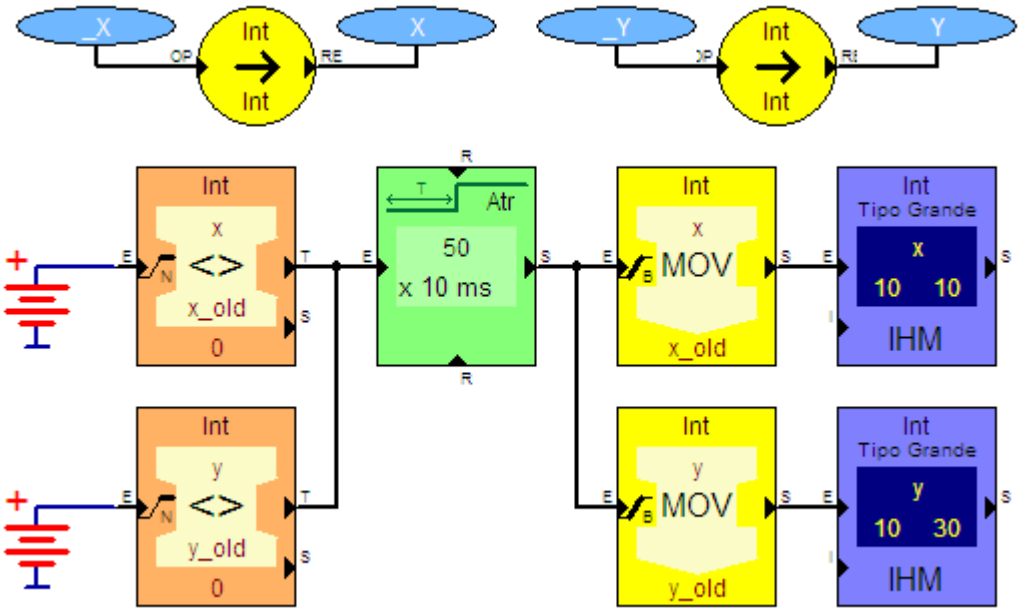
V = Leitura / 819



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - VINT.dxg**

_X, _Y

Indica o valor de coordenadas x e y lidas no sensor touchscreen da IHM para µDX201. Note que estas palavras reservadas estão implementadas apenas em controladores µDX201, uma vez que controladores µDX200 não permitem conexão de IHM. Além disso, é preciso empregar, pelo menos, um bloco de IHM no programa aplicativo, de forma que o acionamento de IHM esteja habilitado no µDX201.Os endereços são M2BACh para _X e M2BAEh para _Y.



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - X,Y.dxg**

_IN1, _IN2

Acessa a leitura das duas entradas analógicas existentes na IHM para µDX201. Note que estas entradas analógicas possuem fundo de escala em 5V e a resolução é baixa (8 bits, ou seja, retorna um valor entre 0 e 255). Mas as entradas podem ser muito úteis para incorporar potenciômetros à IHM, ou mesmo conectadas a botoeiras (on-off). São palavras reservadas válidas apenas para µDX201, e correspondem aos endereços M2BB0h (_IN1) e M2BB2h (_IN2).

_CONTRASTE

Permite modificar o contraste do display de uma IHM conectada ao controlador µDX201. O valor padrão é 39, sendo que o valor deste parâmetro pode ser entre 0 e 63. Em conjunto com o acionamento de luz de fundo (backlight) do display (via blocos de **LUZ MSB** e **LUZ LSB**) pode-se controlar um amplo espectro de brilho e contraste do display, permitindo adaptá-lo a várias situações. É uma palavra reservada válida apenas para µDX201, e corresponde ao endereço M2BB4h.

_E1

Leitura da entrada analógica E1 em alta resolução (16 bits). Note que esta leitura é direta, sem nenhuma calibração. Portanto, é preciso efetuar rotinas de calibração para adequar o valor lido à tensão ou corrente aplicada na entrada E1. É uma palavra reservada válida apenas para µDX201, e corresponde ao endereço M2E0Ch. A leitura de _E1 deve ser feita em longint (32 bits). No caso de controladores µDX201 v3.47 ou superior é possível programar-se o número de entradas analógicas que serão somadas para obter o valor de _E1. Com isso, é possível, por exemplo, conectar várias células de carga ao controlador µDX201 (via placas amplificadoras de célula de carga) e obter uma soma dos valores lidos em alta resolução em _E1 (16 bits). Além disso, via variáveis absolutas v0 a v7 pode-se ler cada célula de carga individualmente em resolução menor (12 bits), permitindo detectar problemas em alguma célula de carga.

_E2, _E3, _E4, _E5, _E6, _E7, _E8

A partir da versão 3.72 do µDX201 é possível programar a profundidade de oversampling, e no caso de oversampling com profundidades menores que 256 o µDX201 pode ler mais de uma entrada analógica com resolução maior que 12 bits:

256 amostras	leitura em 16 bits	Entrada E1	Atraso de 68ms	Resposta até 7 Hz
128 amostras	leitura em 15 bits	Entradas E1,E5	Atraso de 34ms	Resposta até 15 Hz
64 amostras	leitura em 14 bits	Entradas E1,E3,E5,E7	Atraso de 17ms	Resposta até 29 Hz
32 amostras	leitura em 13 bits	Entradas E1,E2,E3,E4,E5,E6,E7,E8	Atraso de 8,5ms	Resposta até 59 Hz

No caso de uso de profundidade de sobreamostragem (oversampling) menor que 256 é habilitado o oversampling de entradas analógicas adicionais (além de E1), e é desabilitada a porta serial RS232-3. Isso porque os buffers de TX e RX dessa porta são usados para os cálculos de oversampling das entradas analógicas E2 a E8.
A tabela abaixo explicita as formas de operação do oversampling das entradas analógicas do µDX201. Note que, no caso de 256 amostras, apenas a entrada E1 permite oversampling, mas é possível usar a soma de mais de uma entrada analógica (até o máximo das 8 entradas) para gerar a variável longint designada pela palavra reservada _E1. Já no caso de 128 amostras podemos ter duas variáveis de oversampling (_E1 e _E5), sendo que, novamente, é possível fazer a soma de várias entradas analógicas para obter _E1 e _E5. Por exemplo, poderíamos usar as entradas E1, E2, E3 e E4 para gerar _E1, e E5, E6, E7 e E8 para gerar _E5 (número de entradas=4). Com isso se obtêm uma resolução maior, pois se faz a média de 128 amostras de 4 entradas analógicas, em vez de apenas uma. E assim por diante, até que com 32 amostras é possível obter oversampling das 8 entradas analógicas independentemente.

Oversampling = 32 (13 bits - leitura x 2)								
Número de entradas	_E1	_E2	_E3	_E4	_E5	_E6	_E7	_E8
1	E1	E2	E3	E4	E5	E6	E7	E8
2	E1+E2	E2+E3	E3+E4	E4+E5	E5+E6	E6+E7	E7+E8	E8+E1
3	E1+E2+E3	E2+E3+E4	E3+E4+E5	E4+E5+E6	E5+E6+E7	E6+E7+E8	E7+E8+E1	E8+E1+E2
4	E1+E2+E3+E4	E2+E3+E4+E5	E3+E4+E5+E6	E4+E5+E6+E7	E5+E6+E7+E8	E6+E7+E8+E1	E7+E8+E1+E2	E8+E1+E2+E3
5	E1+E2+E3+...+E5	E2+E3+E4+...+E6	E3+E4+E5+...+E7	E4+E5+E6+...+E8	E5+E6+E7+...+E1	E6+E7+E8+...+E2	E7+E8+E1+...+E3	E8+E1+E2+...+E4
6	E1+E2+E3+...+E6	E2+E3+E4+...+E7	E3+E4+E5+...+E8	E4+E5+E6+...+E1	E5+E6+E7+...+E2	E6+E7+E8+...+E3	E7+E8+E1+...+E4	E8+E1+E2+...+E5
7	E1+E2+E3+...+E7	E2+E3+E4+...+E8	E3+E4+E5+...+E1	E4+E5+E6+...+E2	E5+E6+E7+...+E3	E6+E7+E8+...+E4	E7+E8+E1+...+E5	E8+E1+E2+...+E6
8	E1+E2+E3+...+E8	E2+E3+E4+...+E1	E3+E4+E5+...+E2	E4+E5+E6+...+E3	E5+E6+E7+...+E4	E6+E7+E8+...+E5	E7+E8+E1+...+E6	E8+E1+E2+...+E7

Oversampling = 64 (14 bits - leitura x 4)								
Número de entradas	_E1	_E2	_E3	_E4	_E5	_E6	_E7	_E8
1	E1		E3		E5		E7	
2	E1+E2		E3+E4		E5+E6		E7+E8	
3	E1+E2+E3		E3+E4+E5		E5+E6+E7		E7+E8+E1	
4	E1+E2+E3+E4		E3+E4+E5+E6		E5+E6+E7+E8		E7+E8+E1+E2	
5	E1+E2+E3+...+E5		E3+E4+E5+...+E7		E5+E6+E7+...+E1		E7+E8+E1+...+E3	
6	E1+E2+E3+...+E6		E3+E4+E5+...+E8		E5+E6+E7+...+E2		E7+E8+E1+...+E4	
7	E1+E2+E3+...+E7		E3+E4+E5+...+E1		E5+E6+E7+...+E3		E7+E8+E1+...+E5	
8	E1+E2+E3+...+E8		E3+E4+E5+...+E2		E5+E6+E7+...+E4		E7+E8+E1+...+E6	

Oversampling = 128 (15 bits - leitura x 8)								
Número de entradas	_E1	_E2	_E3	_E4	_E5	_E6	_E7	_E8
1	E1				E5			
2	E1+E2				E5+E6			
3	E1+E2+E3				E5+E6+E7			
4	E1+E2+E3+E4				E5+E6+E7+E8			
5	E1+E2+E3+...+E5				E5+E6+E7+...+E1			
6	E1+E2+E3+...+E6				E5+E6+E7+...+E2			
7	E1+E2+E3+...+E7				E5+E6+E7+...+E3			
8	E1+E2+E3+...+E8				E5+E6+E7+...+E4			

Oversampling = 256 (16 bits - leitura x 16)								
Número de entradas	_E1	_E2	_E3	_E4	_E5	_E6	_E7	_E8
1	E1							
2	E1+E2							
3	E1+E2+E3							
4	E1+E2+E3+E4							
5	E1+E2+E3+...+E5							
6	E1+E2+E3+...+E6							
7	E1+E2+E3+...+E7							
8	E1+E2+E3+...+E8							

_E9, _E10

Leitura da freqüência em Hz do sinal aplicado à entrada digital E9 ou E10. O limite deste sinal é de 8000 Hz. São palavras reservadas válidas apenas para µDX201, e correspondem aos endereços M2E16h (_E9) e M2E18h (_E10).

_P9, _P10

Leitura de período em intervalos de 125µs do sinal aplicado à entrada digital E9 ou E10. O limite mínimo deste sinal é de 125µs (8KHz), equivalente a leitura de valor 1, e o limite máximo seria de 1s (1Hz), equivalente as leitura de 8192 (8192 × 125µs = 1s). São palavras reservadas válidas apenas para µDX201, e correspondem aos endereços M2E1Eh (_P9) e M2E20h (_P10).

_RXBUF2, _RXBUF3

Estes endereços são o início do buffer de recepção serial 2 e 3 (RS232-2 e RS232-3). São palavras reservadas válidas apenas para µDX201, e correspondem aos endereços M2C60h (_RXBUF2) e M2CA0h (_RXBUF3). Isto permite ler os dados recebidos via serial por software 2 e 3 e analisá-los. O buffer de recepção serial não é circular, ou seja, a cada recepção o buffer é limpo e, portanto, o primeiro caracter recebido está sempre localizado em _RXBUF2 ou _RXBUF3.

Atenção: A porta RS232-2 é implementada com os sinais DTR (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas.

_TXBUF2, _TXPT2, _TXBUF3, _TXPT3

_TXBUF2 e _TXBUF3 são os endereços de início do buffer de transmissão serial 2 e 3 (RS232-2 e RS232-3). São palavras reservadas válidas apenas para µDX201, e correspondem aos endereços M2C80h (_TXBUF2) e M2CC0h (_TXBUF3). No caso do buffer de transmissão trata-se de buffer circular e, portanto, não necessariamente este endereço possui o primeiro caracter a ser transmitido, nem tampouco é o endereço inicial para escrita do buffer. Para fazer a leitura do buffer de transmissão serial devemos usar o byte LSB do apontador (pointer) de leitura do buffer de TX serial _TXPT2 ou _TXPT3 (equivalentes aos endereços absolutos M3028h e M302Ch, respectivamente). Para calcular a primeira posição do buffer de transmissão serial devemos somar _TXBUF2 com o byte MSB de _TXPT2 + 1 (ou _TXBUF3 com o byte MSB de _TXPT3 + 1). Ou seja:

Primeira posição do buffer TX 2 = _TXBUF2 + INT(_TXPT2 / 256) + 1

Primeira posição do buffer TX 3 = _TXBUF3 + INT(_TXPT3 / 256) + 1

Para varremos o buffer de TX serial basta usar uma variável como indexador. Se for usada a variável N a equação fica:

Posição N do buffer TX 2 = _TXBUF2 + [(INT(_TXPT2 / 256) + N + 1) AND 3Fh]

Posição N do buffer TX 3 = _TXBUF3 + [(INT(_TXPT3 / 256) + N + 1) AND 3Fh]

_RXQNT2, _RXQNT3

Indica a quantidade de bytes existentes no buffer de recepção serial 2 e 3 (RS232-2 e RS232-3) e, portanto, o número de bytes recebidos pela porta RS232-2 e RS232-3. Correspondem aos endereços absolutos M302Ah e M302Eh, respectivamente.

_CFGRS232

A partir da versão 3.69 do Controlador µDX201 é permitido modificar os parâmetros da porta serial RS232 (baud-rate, paridade, stop bits) via programa aplicativo. Os parâmetros especificados pelo programa aplicativo via bloco**Setup RS232** são usados prioritariamente em relação aos especificados na **Configuração de Hardware**.

_CFGDXNET

A partir da versão 3.69 do Controlador µDX201 é permitido modificar o endereço na rede DXNET do controlador µDX201 via programa aplicativo. O endereço especificado pelo programa aplicativo via bloco**Setup DXNET** é usado prioritariamente em relação ao especificado na **Configuração de Hardware**.

_NUMSER

A partir da versão 3.69 do Controlador µDX201 é possível ler o número de série do controlador µDX201 via programa aplicativo.

_FIRMWARE

A partir da versão 3.69 do Controlador µDX201 é possível ler a versão de firmware instalada no controlador µDX201 via programa aplicativo.

_PULSO1, _PULSO2, _PULSO3, _POSIC1, _POSIC2, _POSIC3

A versão 3.86 ou superior do Controlador µDX201 permite controle de motores via geração de pulsos nas saídas analógicas S1, S3 ou S5. As saídas S2, S4 e S6 indicam o sentido de rotação, respectivamente. Então, as palavras reservadas _PULSO1, _PULSO2 e _PULSO3 retornam um

valor longint correspondente ao total de pulsos a ser gerado em S1, S3 ou S5. Já as palavras reservadas _POSIC1, _POSIC2 e _POSIC3 retornam um valor longint com a posição atual do motor.

_VELRG1, _VELRG2, _VELRG3, _VELOC1, _VELOC2, _VELOC3

A versão 3.86 ou superior do Controlador µDX201 permite controle de motores via geração de pulsos nas saídas analógicas S1, S3 ou S5. As saídas S2, S4 e S6 indicam o sentido de rotação, respectivamente. Então, as palavras reservadas _VELRG1, _VELRG2 e _VELRG3 retornam um valor inteiro correspondente a velocidade (taxa de pulsos por segundo) a ser gerado em S1, S3 ou S5, quando o motor estiver em regime (após aceleração e antes da desaceleração). Já as palavras reservadas _VELOC1, _VELOC2 e _VELOC3 retornam um valor inteiro com a velocidade atual do motor.

_MOTOR

A versão 3.86 ou superior do Controlador µDX201 permite controle de motores via geração de pulsos nas saídas analógicas S1, S3 ou S5. As saídas S2, S4 e S6 indicam o sentido de rotação, respectivamente. A palavra reservada _MOTOR retorna uma variável inteira com o status das rotinas de geração de pulsos em S1, S3 e S5. Os bits da _MOTOR possuem o seguinte significado:

bit 0 → Saída S1 em free-run (sem limite de pulsos).
bit 1 → Saída S3 em free-run (sem limite de pulsos).
bit 2 → Saída S5 em free-run (sem limite de pulsos).
bit 3 → Motor ligado à saída S1 com sentido de rotação invertido (S2 acionado).
bit 4 → Motor ligado à saída S3 com sentido de rotação invertido (S4 acionado).
bit 5 → Motor ligado à saída S5 com sentido de rotação invertido (S6 acionado).
bit 6 → Geração de pulsos via S1 em andamento.
bit 7 → Geração de pulsos via S3 em andamento.
bit 8 → Geração de pulsos via S5 em andamento.
bit 9 → Status de aceleração/ desaceleração em S1.
bit 10 → Status de aceleração/ desaceleração em S3.
bit 11 → Status de aceleração/ desaceleração em S5.
bit 12 → Status de operação em regime na saída S1.
bit 13 → Status de operação em regime na saída S1.
bit 14 → Status de operação em regime na saída S1.
bit 15 → Não usado.

Note que os bits de aceleração e de regime especificam se o motor está acelerando, em velocidade de regime, ou desacelerando. No caso da saída S1 trata-se dos bits 9 e 12 de _MOTOR. Eles indicam as seguintes situações, enquanto o bit 6 estiver acionado (geração de pulsos em andamento). Quando o bit 6 desligar o motor terá parado.

	ACELERAÇÃO	REGIME	DESACELERAÇÃO	PARADO
Bit 6 (Geração de pulsos ligado)	1	1	1	0
Bit 9 (Status de Acel/Desac)	0	0	1	x
Bit 12 (Status de Regime)	0	1	1	x

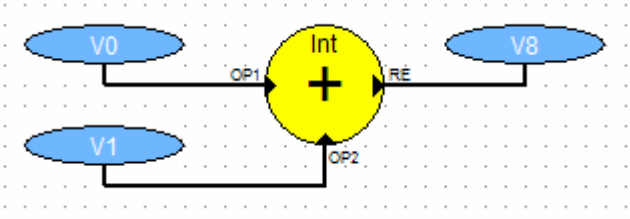
Variável Absoluta (Vnnnn)

A letra V seguida de um valor numérico acessa uma variável absoluta do CLP. Note que o µDX200 utiliza as variáveis V0 até V17 para guardar dados das entradas e saídas do CLP. Ou seja, as variáveis livres para uso no programa aplicativo iniciam em V18. As variáveis V0 até V17 são:

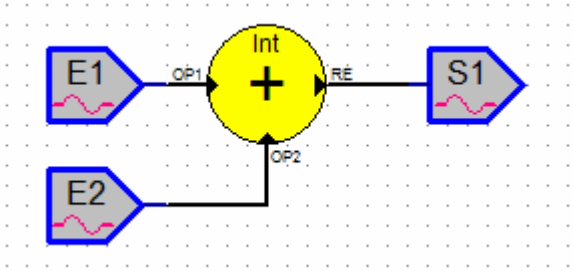
V0 Entrada analógica E1.
V1 Entrada analógica E2.
V2 Entrada analógica E3.
V3 Entrada analógica E4.
V4 Entrada analógica E5.
V5 Entrada analógica E6.
V6 Entrada analógica E7.
V7 Entrada analógica E8.
V8 Saída analógica S1.
V9 Saída analógica S2.
V10 Saída analógica S3.
V11 Saída analógica S4.
V12 Saída analógica S5.
V13 Saída analógica S6.
V14 Contador incremental para entrada digital E9.
V15 Contador incremental para entrada digital E10.
V16-V17 Contador incrementa/decrementa para E9 e E10 (longint).

Note que as variáveis do µDX200 são sempre 16 bits (inteiro ou word). Caso seja utilizada uma variável longint (32 bits) ela ocupa duas posições absolutas (como no caso de V16 e V17, ocupadas por variável longint para contador up/down das entradas rápidas). O acesso direto ao número de uma variável é útil no caso desta variável ser acessada por outro µDX200 (via rede DXNET), ou via software supervisório. Neste caso é necessário acessar a variável absoluta, uma vez que um nome atribuído no programa aplicativo do µDX200 fica restrito ao controlador, não sendo disponibilizado para consulta via rede DXNET ou comunicação serial. Por exemplo, se quisermos ler a entrada analógica E1 via software supervisório ligado a porta serial RS232, precisamos ler V0.

Por exemplo, as duas implementações a seguir somam as entradas analógicas E1 e E2 e colocam o resultado na saída S1. Ou seja, elas são absolutamente equivalentes.



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - Vnnnn.dxg**



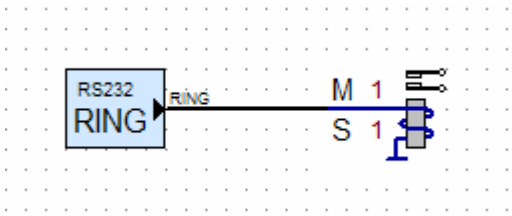
Exemplo de Programa Aplicativo: **Convenções do Compilador PG - I_O.dxg**

Nodo Absoluto (Nnnnn)

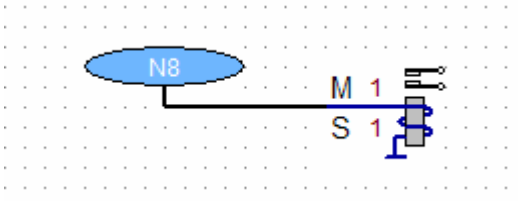
A letra N seguida de um valor numérico acessa um nodo absoluto do CLP. Novamente a maior utilidade de acesso a um determinado nodo específico é para disponibilizá-lo para um dispositivo externo (seja um outro µDX200 ligado a rede DXNET, seja um sistema supervisório ligado a porta serial). Os nodos N0 até N31 são reservados e possuem atribuições específicas. Ou seja, os nodos livres para uso no programa aplicativo iniciam em N32. Os nodos N0 até N31 são:

N0	**Sempre em zero.**
N1	**Sempre em um.**
N2	**Pulso de Energia Liga ao retornar alimentação (nodo EL).**
N3	**Ocorreu Reset ou comando Run (nodo Reset).**
N4	**Ocorreu interrupção externa 1 (entrada E9).**
N5	**Ocorreu interrupção externa 1 (entrada E10).**
N6	**Estado da entrada de int. ext. 1 (entrada E9).**
N7	**Estado da entrada de int. ext. 2 (entrada E10).**
N8	**Sinal de entrada RING (RS232).**
N9	**Sinal de entrada DCD (RS232).**
N10	**Sinal de entrada DSR (RS232).**
N11	**Sinal de saída DTR (RS232).**
N12	**Sinal de saída RTS (RS232).**
N13	**Sinal de entrada CTS (RS232).**
N14	**Pulso a cada 1 segundo (síncrono com relógio).**
N15	**Pulso a cada 100ms (síncrono com relógio).**
N16	**Erro aritmético.**
N17	**Erro de MMC.**
N18	**MMC ausente.**
N19	**MMC cheio.**
N20	**Buffer de transmissão serial RS232 vazio.**
N21	**Acionamento de buzzer da IHM (apenas µDX201).**
N22	**Brilho de backlight do display da IHM - bit LSB (apenas µDX201).**
N23	**Brilho de backlight do display da IHM - bit MSB (apenas µDX201).**
N24	**Saída comandada da IHM (apenas µDX201).**
N25	**Erro nos osciladores internos do controlador (apenas µDX201).**
N26	**Efetua backup de toda RAM na FLASH (salva RAM).**
N27	**Leitura de backup da RAM em FLASH (recupera RAM).**
N28	**Indica erro no backup de RAM em FLASH.**
N29 a N31	**Reservados.**

Note que nodos são variáveis binárias, ou seja, assumem valor 1 ou 0 (verdadeiro ou falso). Todos estes nodos especiais estão disponíveis através de blocos específicos. Por exemplo, podemos ler o estado da entrada RING disponível no conector RS232 do CLP tanto usando o bloco existente para este fim nos blocos de RS232, quanto acessando diretamente o nodo N8:



Exemplo de Programa Aplicativo: **RS232 - RING.dxg**



Exemplo de Programa Aplicativo: **Convenções do Compilador PG - Nnnnn.dxg**

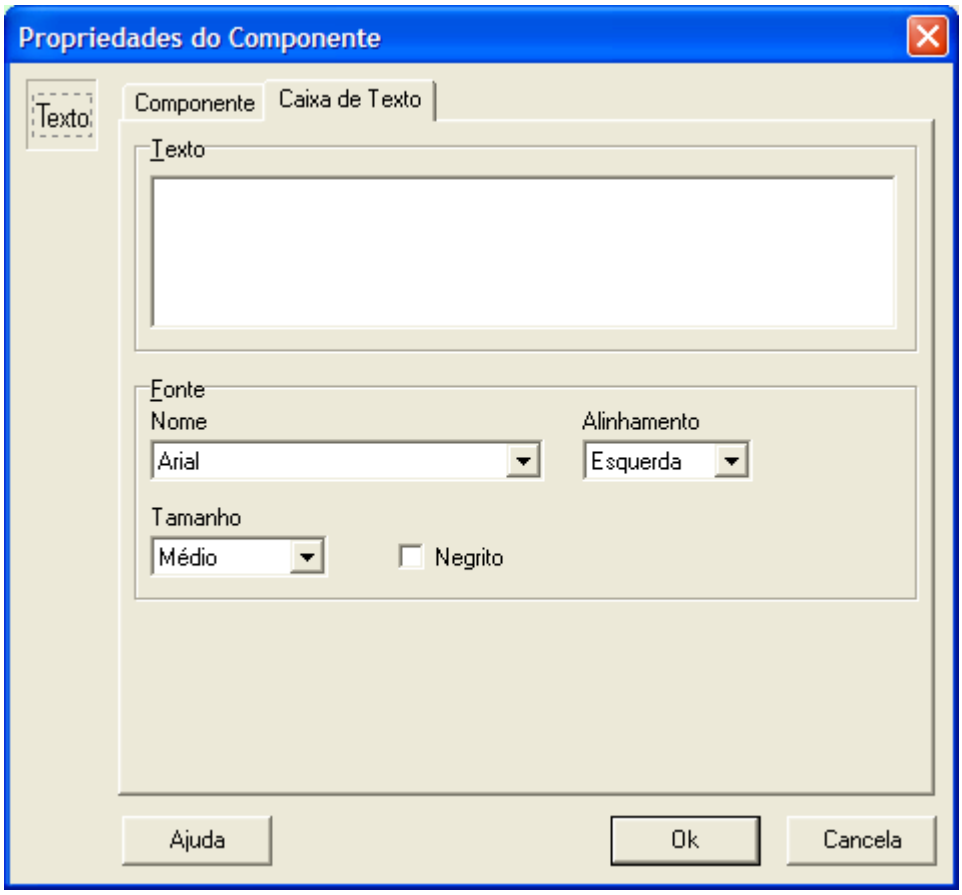
Os nodos N21 a N24 se referem a IHM para µDX201 e, portanto, não possuem funcionalidade em controladores µDX200. Os nodos N22 e N23 determinam o brilho da luz de fundo (backlight) do display da IHM. Assim, a combinação destes dois nodos é a seguinte:

Brilho	**N23**	**N22**
Desligado	0	0
Baixo	0	1
Médio	1	0
Alto	1	1

Já o nodo N24 comanda uma saída tipo NPN na IHM, capaz de acionar relés eletromecânicos ou pequenos solenóides (até 100mA em 24V).

Texto

É possível inserir uma caixa de texto no programa. Esta caixa de texto não é utilizada pelo programa, servindo exclusivamente como comentário para o programa aplicativo do µDX200. Para "largar" a caixa de texto sobre o programa basta clicar com a tecla esquerda do mouse sobre o local desejado. Para editar a caixa de texto, permitindo inserir o texto, clique com a tecla direita do mouse apontando para a caixa de texto. Deve surgir uma janela que permite, além da digitação do texto, a escolha da fonte de caracteres a ser usada, o tamanho dos caracteres, e se em negrito ou não, além do alinhamento do texto.



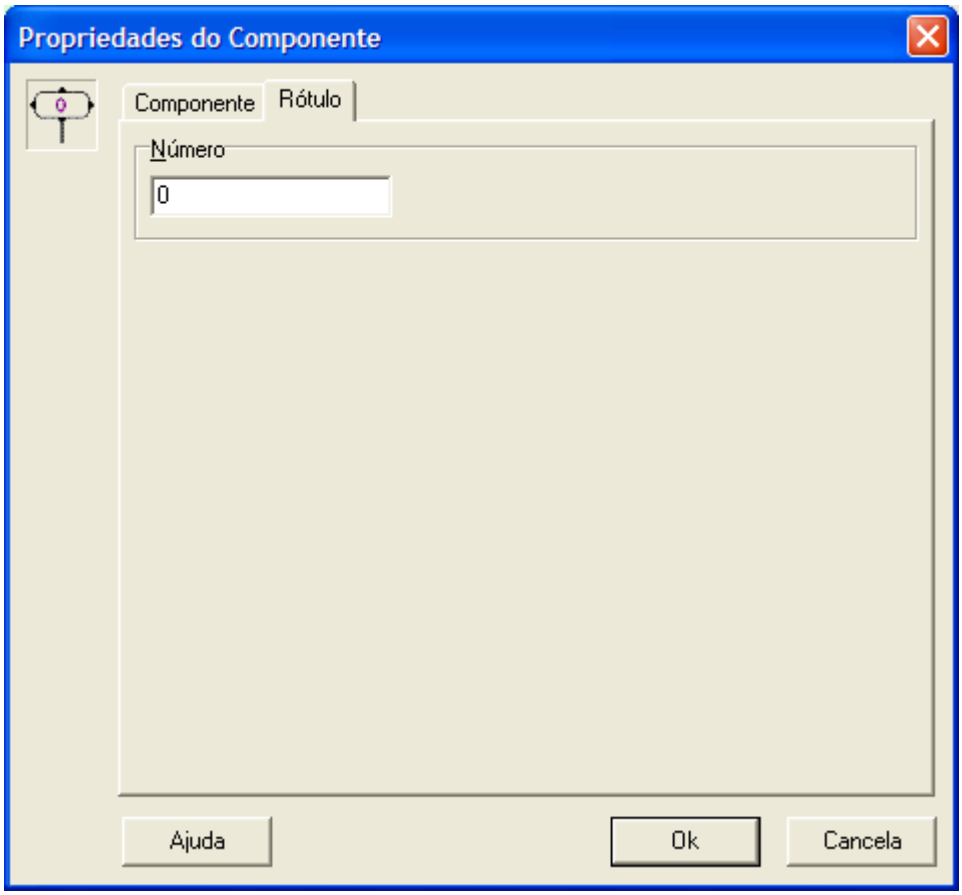
Para ajustar o tamanho da caixa de texto (de forma que o texto digitado caiba na caixa) basta selecionar a caixa e arrastar os cantos dessa, modificando suas dimensões.

Observação: É uma boa prática de programação incluir caixas de texto com comentários pertinentes sobre o funcionamento do programa aplicativo (tipo função das entradas e saídas, função de determinados blocos, etc.). Lembre-se que isso irá facilitar muito o entendimento do programa no futuro.

Rótulo

A função do rótulo é conectar diferentes pontos do programa entre si. O rótulo apenas conecta entre si todos os pontos com o mesmo número de rótulo (de 0 a 999), como se estivessem conectados por "fios", não atribuindo nenhum valor determinado ou nome para esta conexão. O rótulo pode conectar qualquer tipo de variável do µDX200 (nodo, inteiro, longint ou word). Ao editar o Rótulo surge a janela que permite atribuir um valor numérico ao mesmo. Todos os rótulos com mesmo valor numérico estarão conectados (mesmo em janelas de programação distintas de um mesmo projeto).

Note que rótulo não é permitido em Macros. Desta forma, esta função estará inibida quando estiver selecionada uma página de Macro.

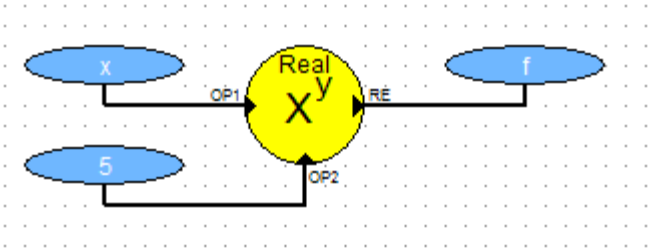


Perceba a diferença sutil entre **Rótulo** e os blocos **Nodo**, **Variável Inteira**, **Variável LongInt**, **Variável Word** e **Variável Real**, existentes na aba **Geral** da Biblioteca de Componentes. O Rótulo apenas conecta dois pontos como se fossem utilizados "fios" de conexão. Ele não determina o tipo de conexão (nodo, variável inteira, etc.) nem tampouco o nome da conexão. Já os blocos citados fazem exatamente isso. Eles atribuem um nome e um tipo para a conexão ligada a eles.

Macro

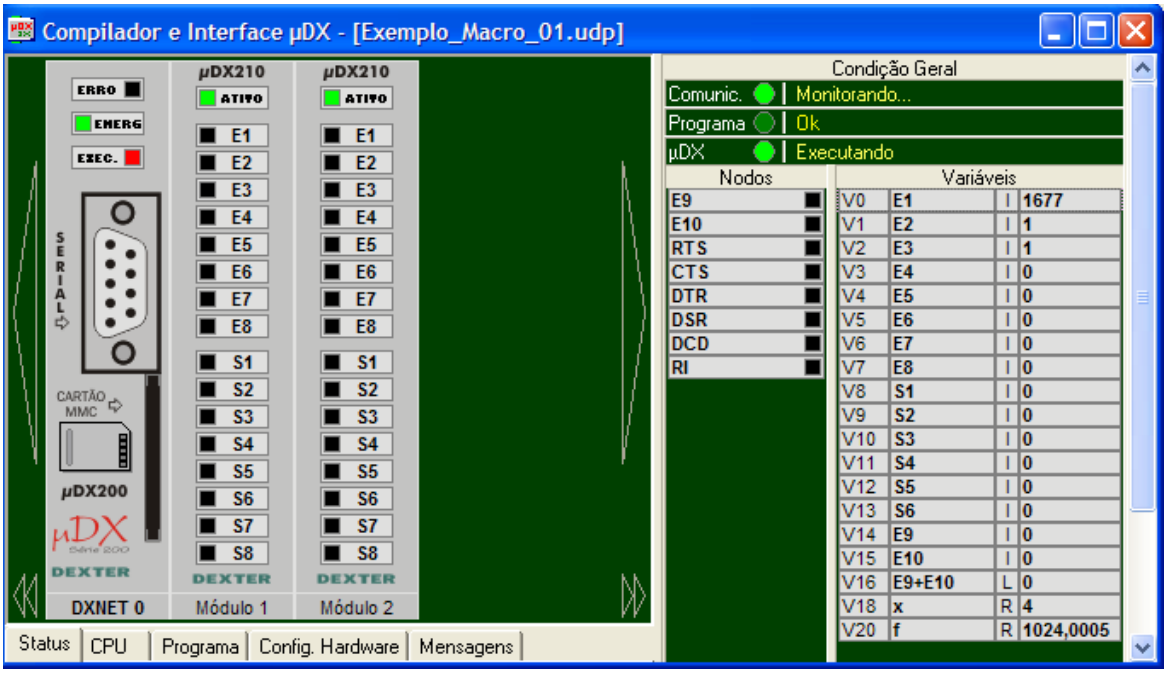
A Macro (macro-célula, ou macro-bloco) permite condensar todo um programa aplicativo em apenas um bloco, a ser utilizado em outros programas. Este recurso é muito poderoso, pois permite abordar uma determinada aplicação complexa a partir de blocos menores. Note que pode-se gerar uma nova Macro a partir de outras Macros pré-existentes. Vamos exemplificar isso.

Digamos que seja necessária a função matemática **$f(x)=x^5$.** Esta função não está disponível nos blocos aritméticos do µDX200, mas existe a função **$f(x)=x^y$.** Então, a função desejada pode ser facilmente implementada (vamos trabalhar neste exemplo com números reais):



Exemplo de Programa Aplicativo: **Exemplo_Macro_01.dxg**

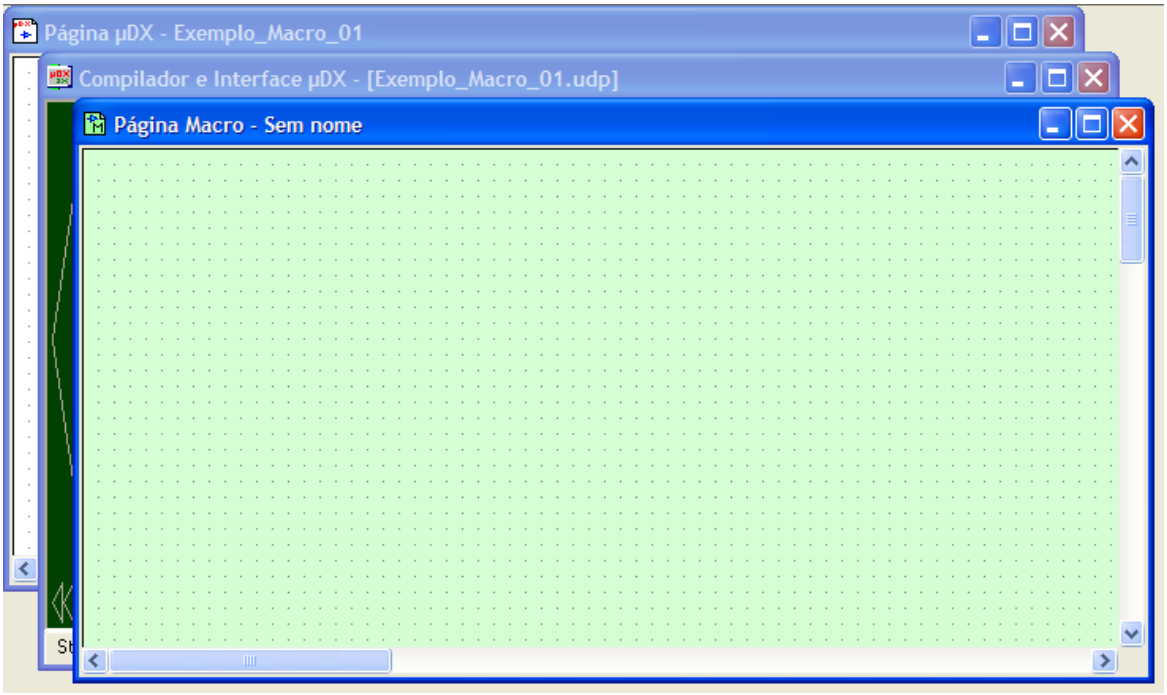
O programa aplicativo acima facilmente gera a função desejada. Podemos testar seu funcionamento via Compilador PG. Por exemplo, se forçarmos a variável **x** com valor 4:



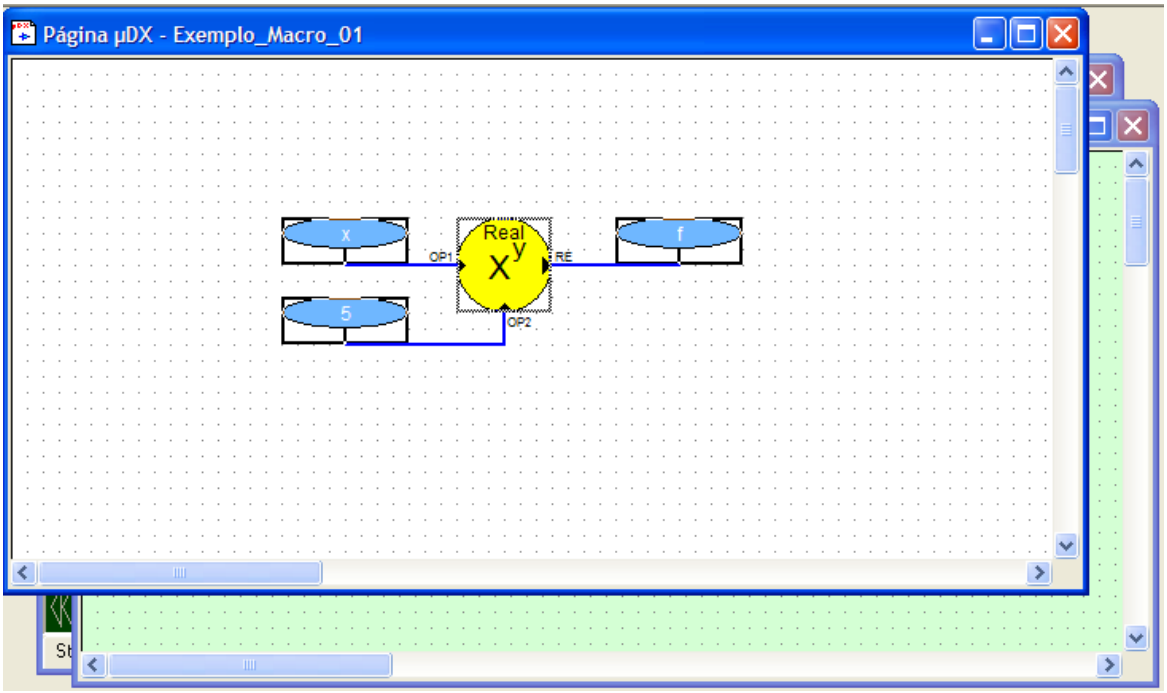
Note que a variável **f** assumiu o valor 1024,0005. O valor exato de 4^5 é 1024, obviamente. O erro de 0,0005 é devido a representação de números reais no µDX200, que utiliza apenas 32 bits para armazenar mantissa e expoente do número real. Isso limita a precisão dos cálculos (gerando este erro de 0,00005%), mas por outro lado permite realizar cálculos em ponto flutuante com relativa rapidez.

Vamos transformar este pequeno programa em uma Macro, capaz de ser utilizada em outros programas para o µDX200. Para isso vá em [**Arquivo**] → [**Nova Macro**]. Deve surgir uma nova janela (Página Macro). As janelas de programação de Macros possuem o fundo verde, para

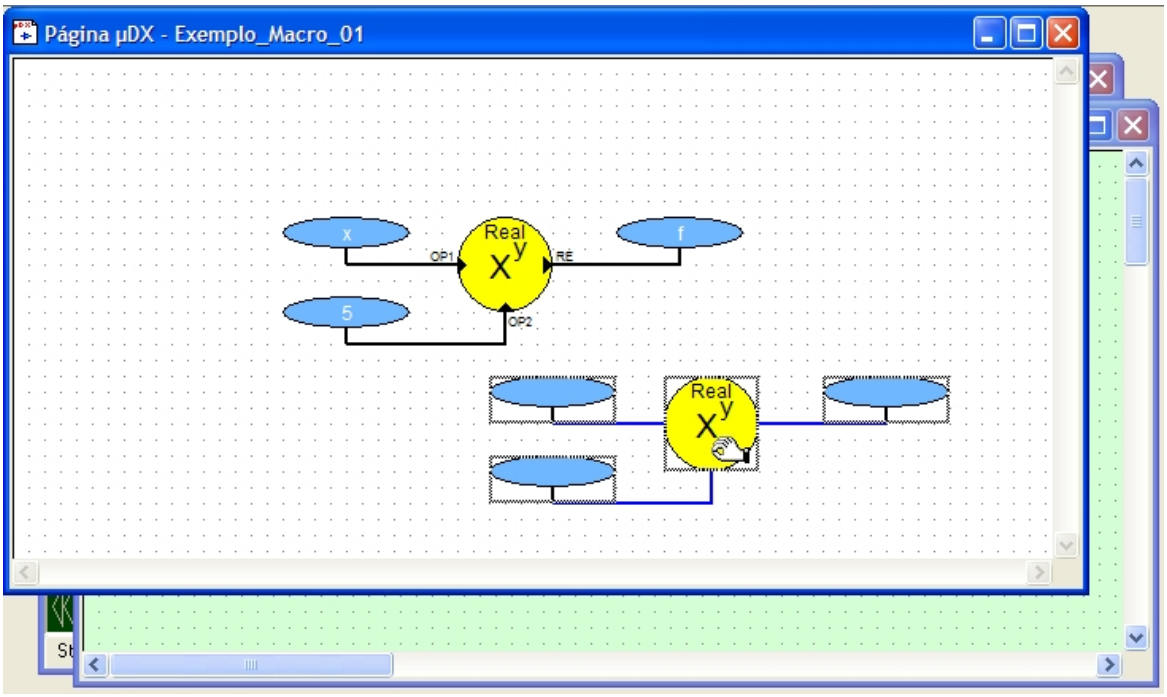
distinguí-las das janelas de programação do µDX, cujo fundo é branco.



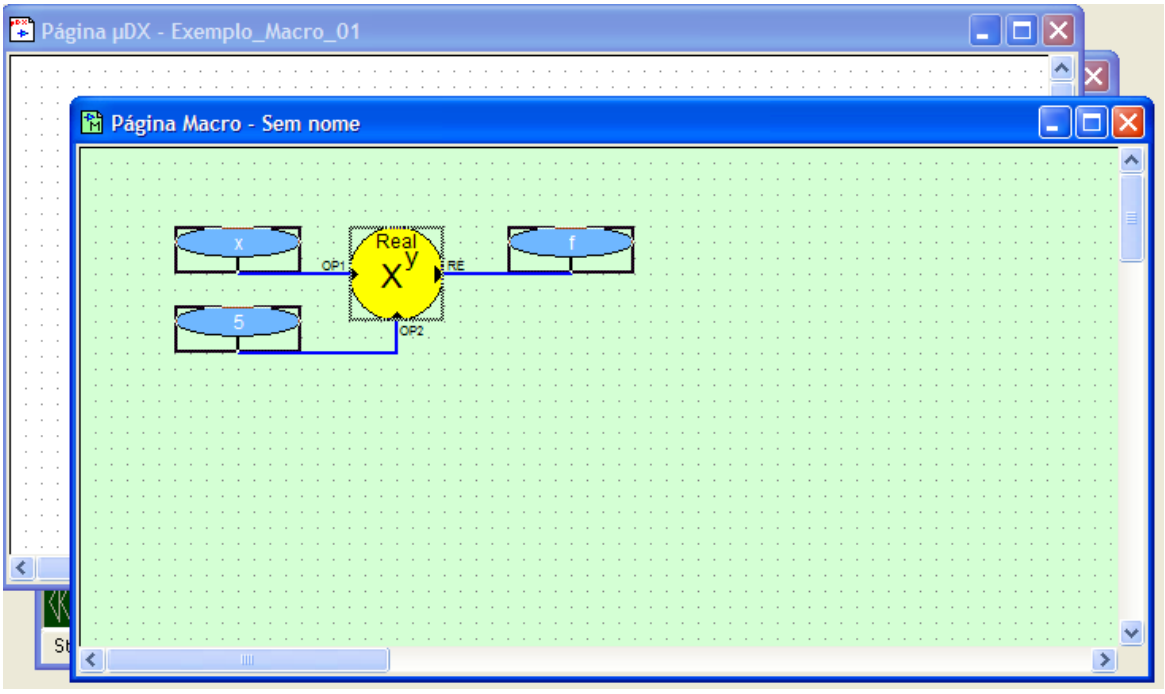
Retorne à Página µDX onde foi desenhado o programa e o selecione (para isso basta criar um retângulo de seleção mantendo pressionado o botão esquerdo do mouse e deslocar o cursor até abarcar todo o programa):



Agora pressione **Ctrl+C** no teclado do computador, ou vá no menu pop-down [**Editar**] → [**Duplicar**]. Com isso duplicamos o programa selecionado:

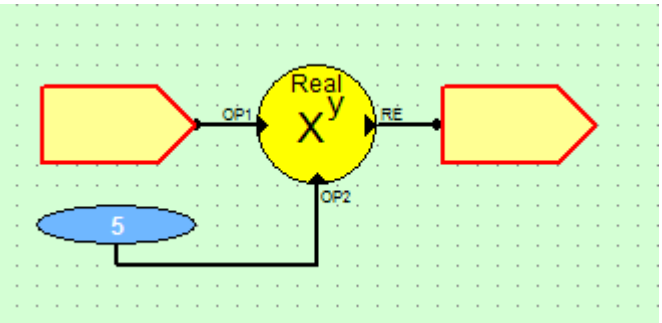


Agora clique com o botão esquerdo do mouse sobre a janela de programação de Macro (tela verde). Com isso o programa duplicado poderá ser "largado" nesta janela. Escolha um local adequado e pressione novamente o botão esquerdo do mouse:

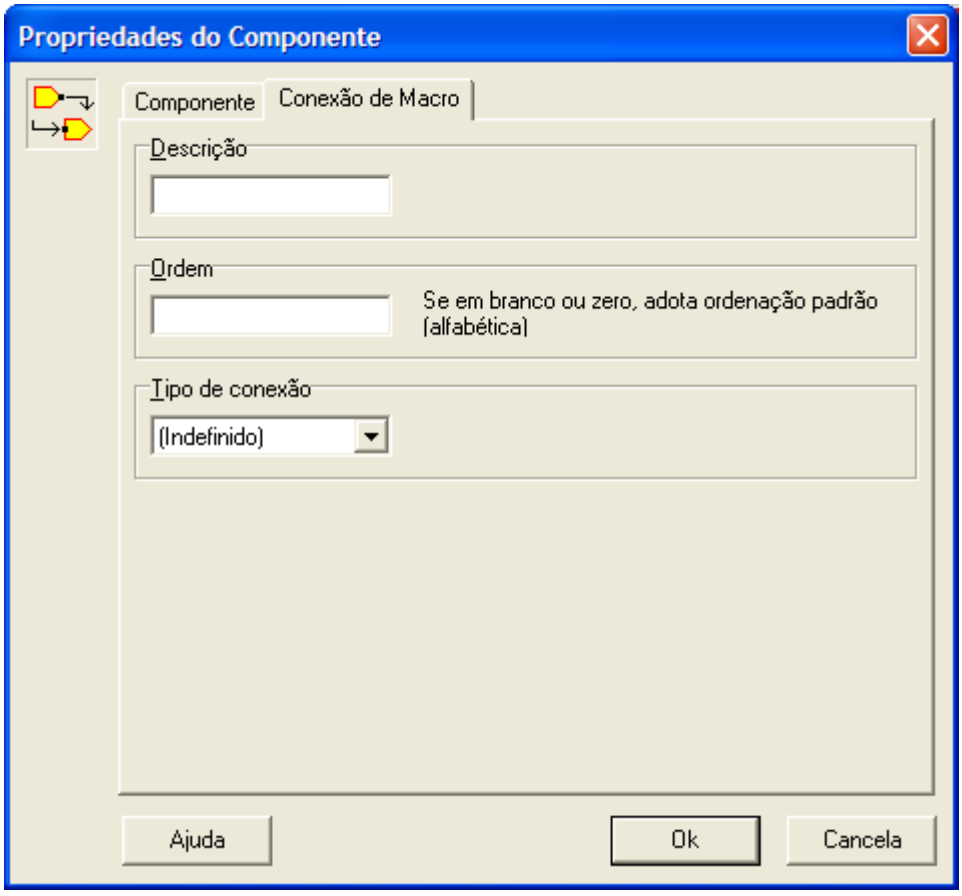


Com isso o programa foi copiado para a página de Macro. Mas todas as variáveis utilizadas em uma Macro não são acessíveis externamente. Portanto, uma Macro como a representada anteriormente não teria nenhuma serventia, pois não teria nenhum contato com o mundo exterior a ela. Vamos, então, utilizar as entidades **Entrada de Macro** e **Saída de Macro** para estabelecer pontos de entrada e saída para a Macro. Neste caso a única entrada seria para a variável **x**, e a única saída seria para a variável **f**. Substituindo os blocos **Variável Real** utilizados por 

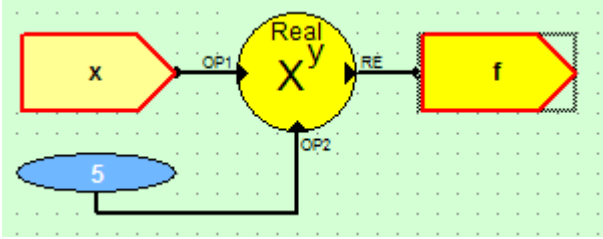
(Entrada de Macro) e (Saída de Macro) existentes na Barra de Ferramentas do PG (ou vá em [**Editar**] → [**Inserir Entrada de Macro**] e [**Editar**] → [**Inserir Saída de Macro**]), teremos:



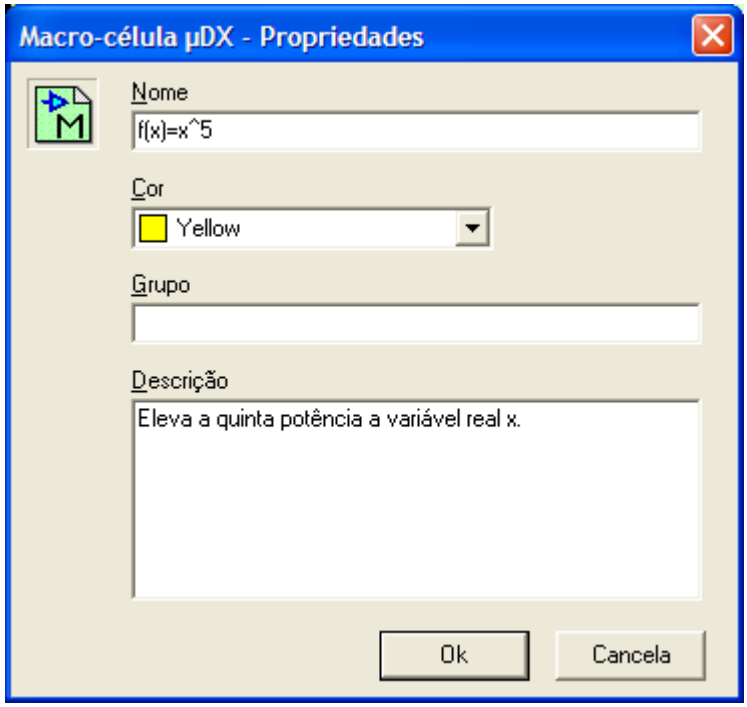
Agora edite a **Entrada de Macro** e **Saída de Macro**, atribuindo nomes a elas. clicando com a tecla direita do mouse sobre as mesmas:



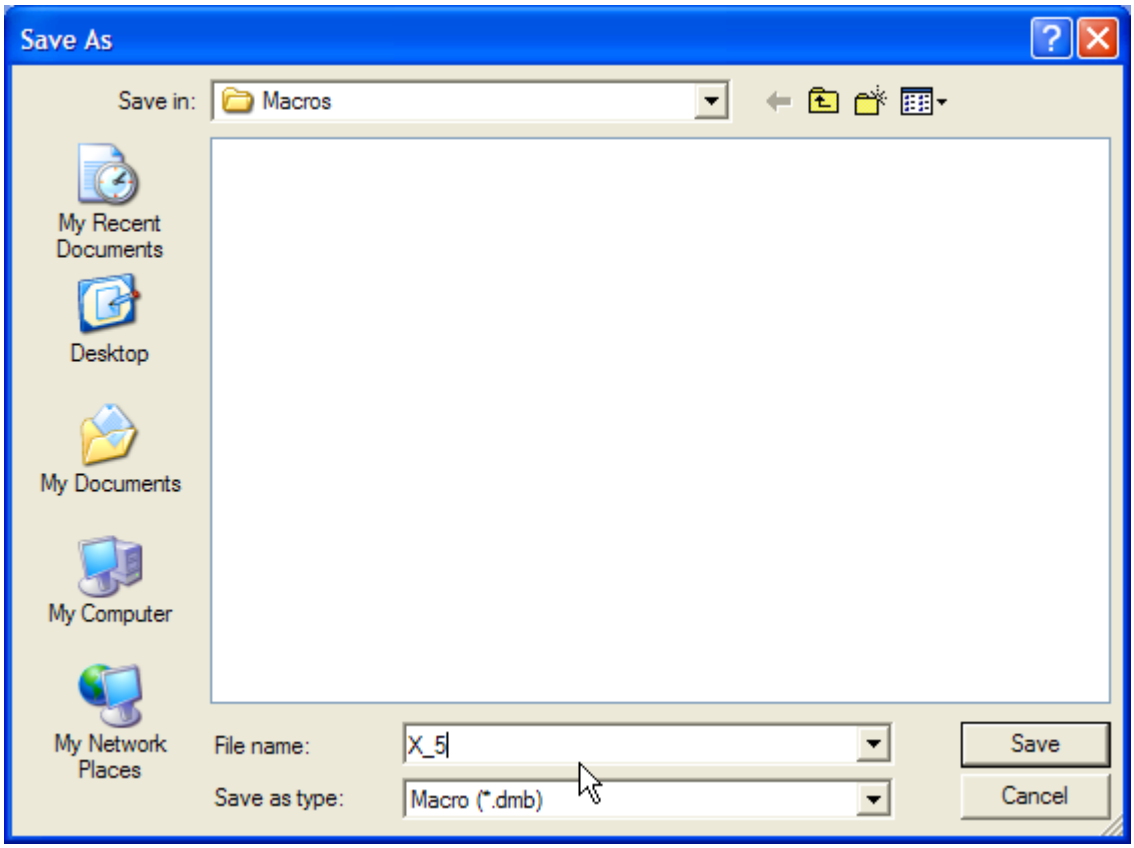
Digamos que se atribua para a **Entrada de Macro** o nome **x**, e para a **Saída de Macro** o nome **f**, de forma a manter a nomenclatura empregada no programa:



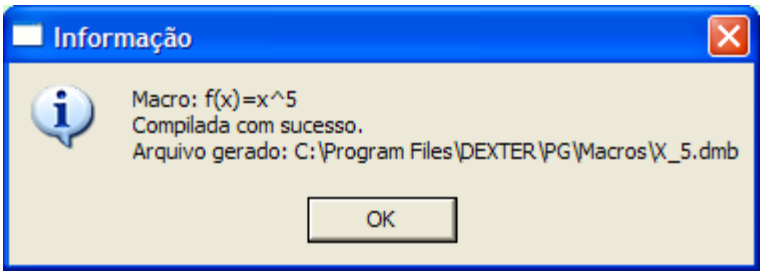
Está pronta a Macro! Falta apenas selecionar as propriedades da Macro e compilá-la. Para isso vá no menu pop-down [**Macro**] → [**Propriedades da Macro...**]. Digite os seguintes dados (note que foi escolhida a cor amarela para esta macro, mas pode ser escolhida qualquer cor desejada):



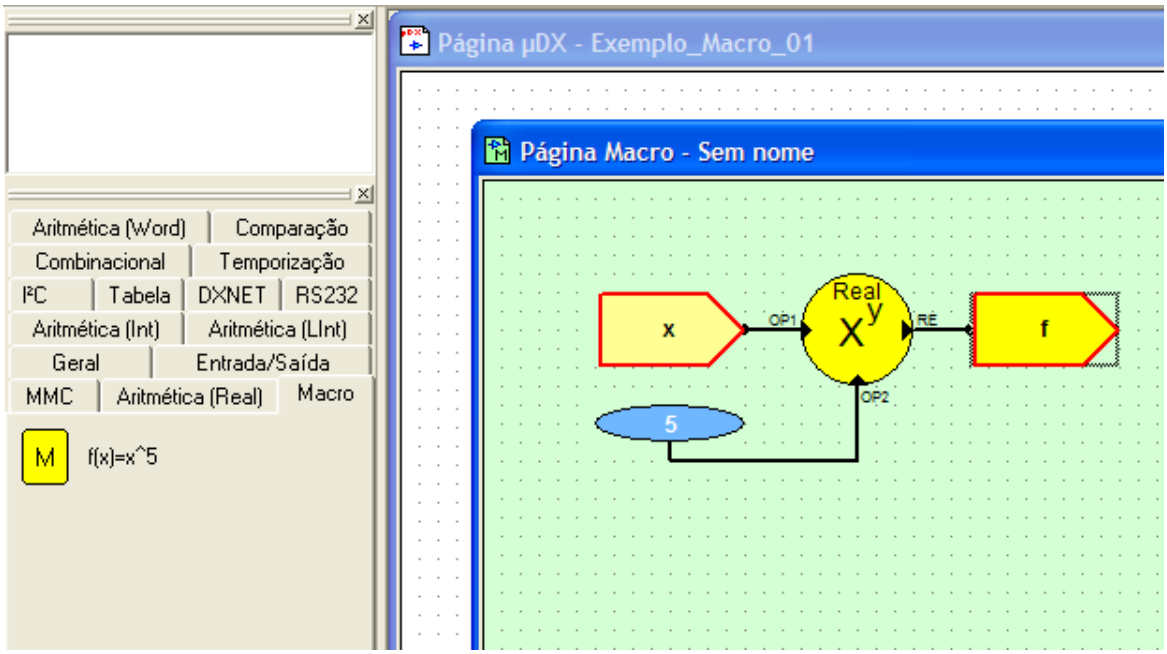
Agora vá em [**Macro**] → [**Compilar e Salvar Macro...**]. O PG irá perguntar o nome da Macro a ser salva. No caso foi chamada de **X_5**. Será gerado um arquivo sufixo **DMB** que será o resultado da compilação desta Macro. Estes arquivos devem ser salvos no diretório **Macros** existente no diretório de instalação do PG (normalmente em c:\ Arquivos de Programas \ Dexter \ PG \ Macros).



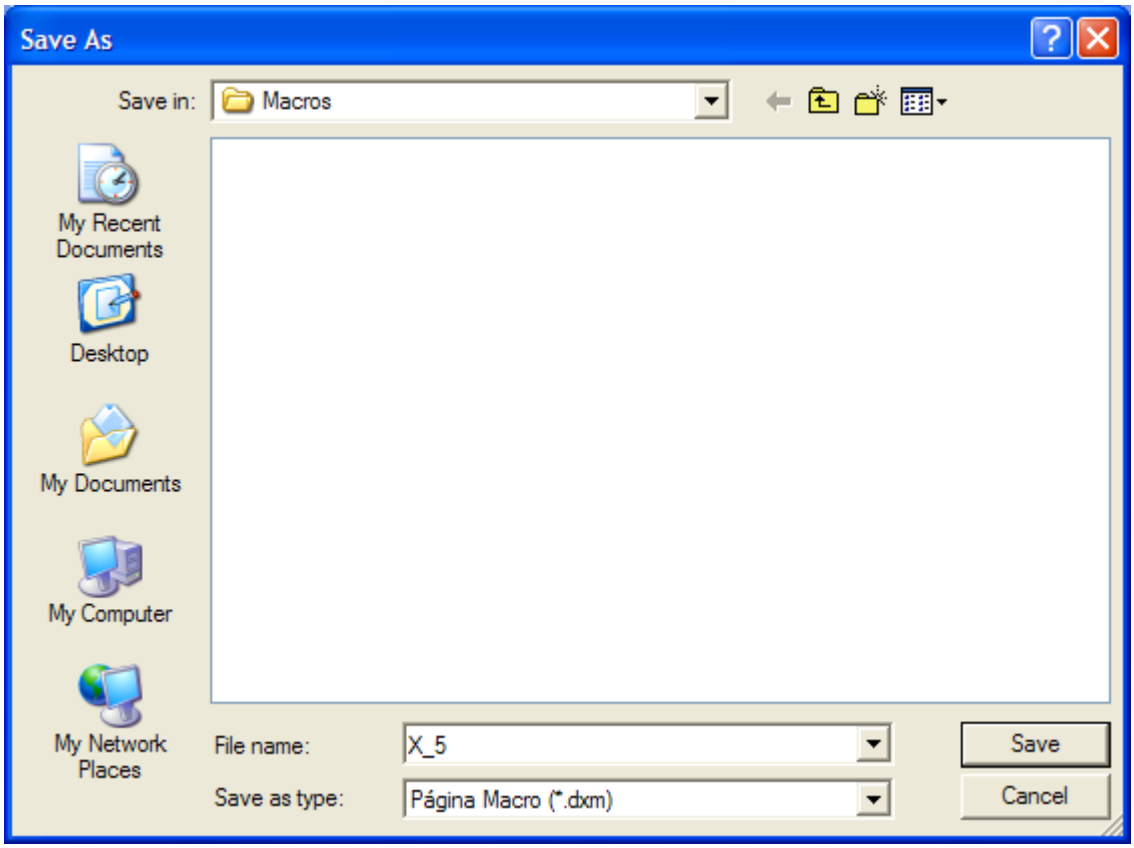
Caso a compilação tenha ocorrido sem falhas deve surgir a seguinte janela de informação:



Note que é gerada uma aba adicional na **Biblioteca de Componentes**, com designação **Macro**, onde já deve constar a macro gerada:



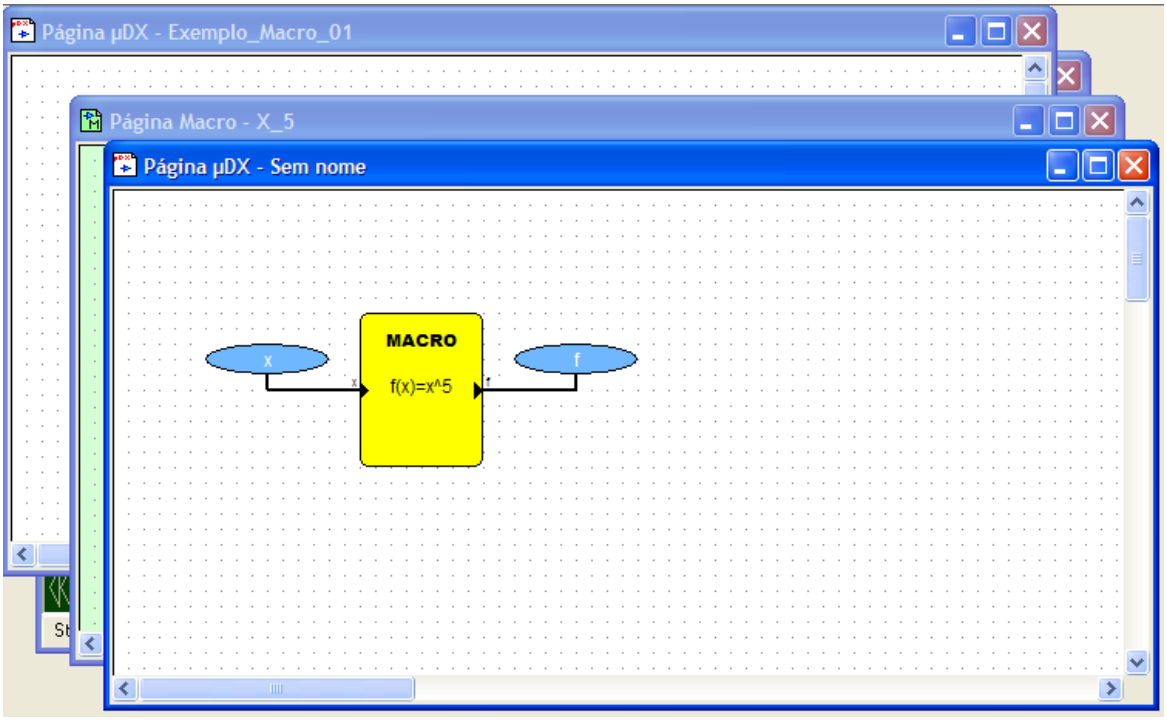
O arquivo **X_5.dmb** gerado corresponde a macro compilada. Entretanto, o arquivo fonte desta macro ainda não foi salvo. Para isso clique em na Barra de Ferramentas do PG, ou selecione [**Arquivo**] → [**Salvar**] nos menus.



Vamos chamar do mesmo nome (X_5). Mas agora será gerado um arquivo sufixo **DXM**, correspondendo ao arquivo fonte da Macro. Com isso, no futuro, será possível implementar

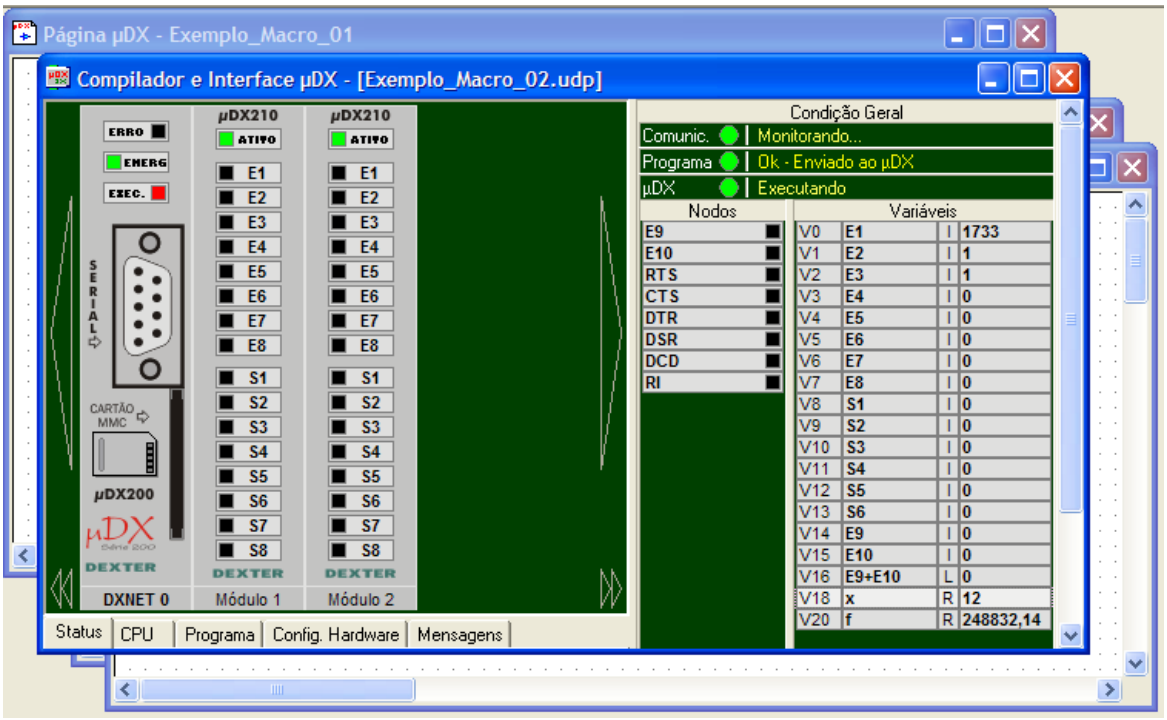
modificações nesta Macro. Normalmente se salva fontes de macros no mesmo diretório **c:\ Arquivos de Programas \ dexter \ PG \Macros**.

Bem, agora podemos utilizar a Macro criada. Para isso vamos abrir uma nova página µDX clicando em (Nova Página) na barra de ferramentas do PG, ou em [**Arquivo**] → [**Nova Página**]. A seguir, utilize a Macro como se fosse um bloco normal do µDX200 e monte o programa como a seguir:



Salve a nova página µDX e compile este novo programa, agora usando Macro (use a tecla da barra de ferramentas). Veja que, ao compilar a Macro, o PG percebeu que os "fios" de conexão dos componentes **Entrada de Macro** e **Saída de Macro** transportavam variáveis reais e, com isso, gerou pontos de conexão para a Macro com variáveis reais.

No Compilador PG transmita o programa para o µDX200 e teste seu resultado. Abaixo está o resultado para x=12.

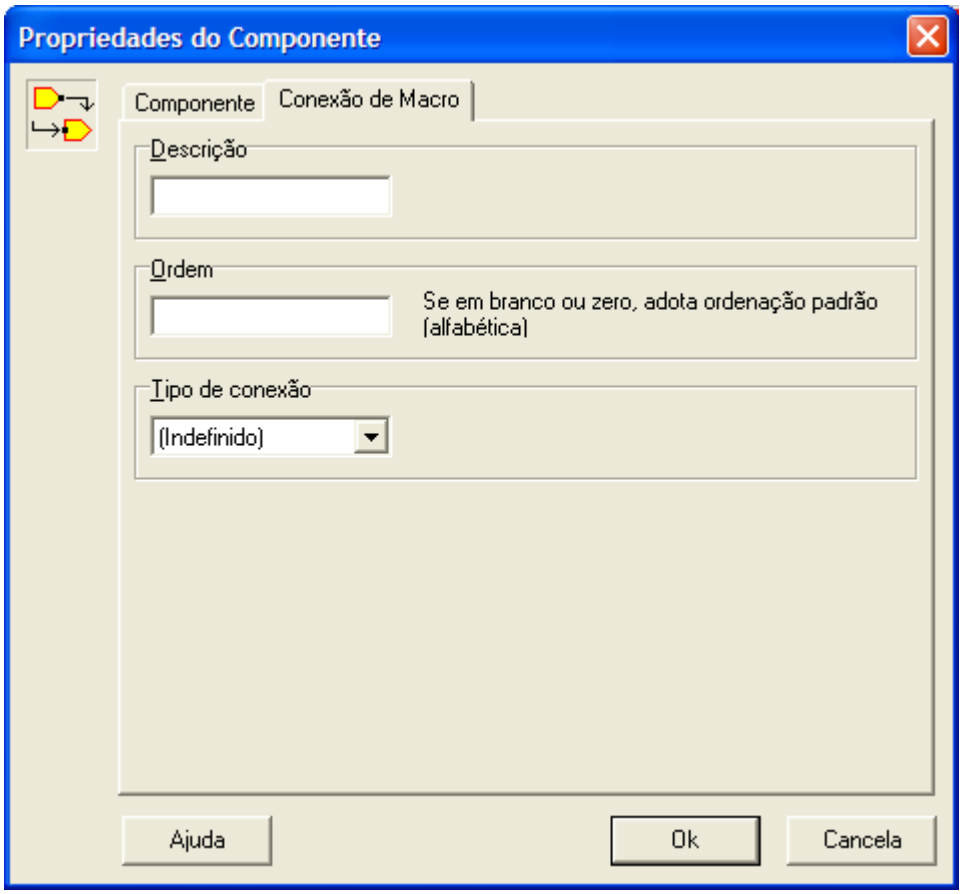


Veja que é perfeitamente possível utilizar a Macro criada na elaboração de outras macros, aumentando gradativamente a complexidade do programa.

Atenção: Apesar da Macro ser representada apenas como um bloco, ela ocupa o número de blocos, variáveis e nodos utilizados para sua elaboração. Assim, o uso massivo de Macros complexas em um programa aplicativo pode, rapidamente, esgotar estes recursos.

Entrada e Saída de Macro

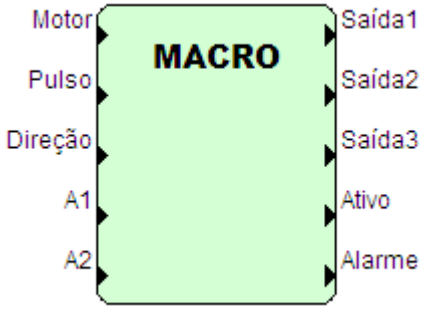
Entrada de Macro e **Saída de Macro** são objetos capazes de inserir pontos de conexão da Macro com o exterior. Estas entradas ou saídas podem tanto ser um nodo (variável binária), quanto uma variável (inteira, word, longint, real). As opções para inserir estes objetos somente ficam ativas quando está selecionada uma página de Macro, já que não é possível usar **Entrada de Macro** ou **Saída de Macro** em programas aplicativos do µDX200. Ao editar uma **Entrada de Macro** ou uma **Saída de Macro** surge a janela abaixo, que permite especificar um nome para a conexão da Macro.



A **Ordem** em uma Entrada de Macro (ou Saída de Macro) especifica qual a ordem em que as conexões de entrada (ou saída) aparecerão no bloco de Macro. As entradas sempre são colocadas à esquerda do bloco de Macro, e as saídas à direita. Com isso, entradas e saídas são ordenadas independentemente. Caso o campo Ordem seja mantido em branco a ordenação será em ordem alfabética crescente. Caso seja colocado números no campo Ordem, os itens serão ordenados em ordem crescente destes números. É aceitável qualquer número inteiro (portanto, é possível usar-se números negativos). Se duas os mais entradas de Macro possuírem o mesmo número de ordem, elas serão ordenadas alfabeticamente. Por exemplo, digamos que foi gerada uma Macro com as seguintes conexões de entrada e saída:

Entradas de Macro		Saídas de Macro	
Descrição	**Ordem**	**Descrição**	**Ordem**
A1		Saída1	-3
A2		Saída2	-3
Motor	-1	Saída3	-3
Pulso	3	Alarme	10
Direção	4	Ativo	2

Neste caso o PG irá gerar um bloco de Macro como mostrado a seguir:



O **Tipo de Conexão** permite especificar qual tipo de dado é esperado para a conexão da Macro. No caso de **(Indefinido)** é aceito qualquer tipo de dado, sem que sejam geradas mensagens de alerta na compilação. Já nas outras opções é gerada uma mensagem de alerta caso seja conectado outro tipo de variável, diferente da especificada neste campo. Isso permite discernir o tipo de dado a ser conectado à Macro (Lógico, Byte, Inteiro, Word, Real, etc).

Ao clicar com botão direito sobre a Macro (Propriedades da Macro) surge uma lista de todas as conexões e seu tipo na aba **Conexões**. Como esta especificação de **Tipo de Conexão** foi implementada apenas na versão 2.2.0.16 do software PG as Macros que acompanham o PG em versões anteriores apresentam todas as conexões como indefinidas.

Macros que acompanham o PG

O poder de síntese das Macros é enorme, e para exemplificar seu uso algumas foram incluídas na instalação do PG. Foram geradas até o momento doze Macros, oito delas referentes à automação residencial, duas referentes à automação industrial, e duas para uso com Interface Homem/Máquina (IHM µDX220). Note que estas últimas só funcionam com controlador µDX201. As Macros que acompanham o software PG são:

Dimmer → Permite controle de iluminação por Dimmer via saídas analógicas.

Dimmer 50Hz → Idêntica à macro anterior, mas adaptada para rede elétrica em 50Hz.

Dimmer I2C → Permite controle de iluminação por Dimmer via rede I^2C (mini-dimmer).

Dimmer I2C ON-OFF → Permite controle de iluminação por Dimmer via rede I^2C (mini-dimmer), comutando diretamente entre 0 e 100%.

IR → Decodifica todos os botões do controle remoto para Keypad da Dexter, exceto os números, ponto e enter (↵).

IR_2 → Decodifica os demais botões do controle remoto para Keypad da Dexter - números, ponto e enter (↵).

KEYPAD → Permite ler o Keypad da Dexter, fazendo consistência dos dados.

MUX → Decodifica unidades de MUX ligadas às entradas analógicas do µDX201.

MUX2 → Decodifica unidades de MUX2 (novo modelo de Multiplexador imune a mau contato nos contatos de pulsadores).

MUX2P → Decodifica unidades de MUX2 usando média de 32 amostras da entrada analógica.

PID → Controle Proporcional - Integral - Derivativo para µDX201.

uDX212 → Acionamento com retenção para as saídas de módulos µDX212.

uDX212 Latch → Acionamento com retenção para as saídas de módulos µDX212 com entradas para cenários.

Cenário → Armazena até 4 variáveis para montagem de cenários em automação residencial.

Calibração Touch → Possibilita a calibração do sensor touchscreen existente na IHM para µDX201.

Entrada (Int) → Desenha teclado numérico na IHM para µDX201, permitindo entrada de valores inteiros.

Senha (Int) → Permite digitar senha com 4 dígitos numéricos.

DNP3 → Decodifica vários comandos do protocolo DNP3 recebidos via porta serial RS232.

Step → Gera acionamento para motor de passo.

Motor → Gera pulsos para driver de motor de passo, com aceleração e desaceleração.

Persiana (1 botão) → Controle de persiana motorizada via um único controle.

Persiana (2 botões) → Controle de persiana motorizada via dois controles (sobe e desce).

Confirma → Permite seleção entre duas opções (SIM ou NÃO) na IHM para µDX201.

IR-TX → Captura e transmite comandos infravermelhos via módulo IR-TX.

Lê Relógio → Lê relógio interno (RTC) do controlador e disponibiliza data e hora.

Grava Relógio → Gava relógio interno (RTC) do controlador com os dados especificados para data e hora.

µDX215 → Decodifica as entradas digitais de dois módulos µDX215.

Le_Var_uDX101 → Leitura de variáveis tipo byte de µDX101 conectado à porta serial RS232 do µDX201.

Le_Var_Word_uDX101 → Leitura de variáveis tipo word de µDX101 conectado à porta serial RS232 do µDX201.

Tela_Projecao → Acionamento de tela de projeção com apenas um botão.

LEDs → Desenha LED de vários formatos e tamanhos na posição especificada da IHM, representando o estado do nodo de entrada.

Texto → Permite escrever string na posição especificada da IHM.

Variável Inteira → Permite escrever variável inteira na posição especificada da IHM.

ADC (DIF) → Leitura de entrada analógica diferencial na Expansão µDX218 (até +/-1,25V com 24 bits de resolução)

ADC (SGL) → Leitura de entrada analógica referenciada ao GND (single-ended) na Expansão µDX218 (até +/-1,25V com 24 bits de resolução)

Diodo → Leitura de temperatura via diodo conectado à entrada da Expansão µDX218

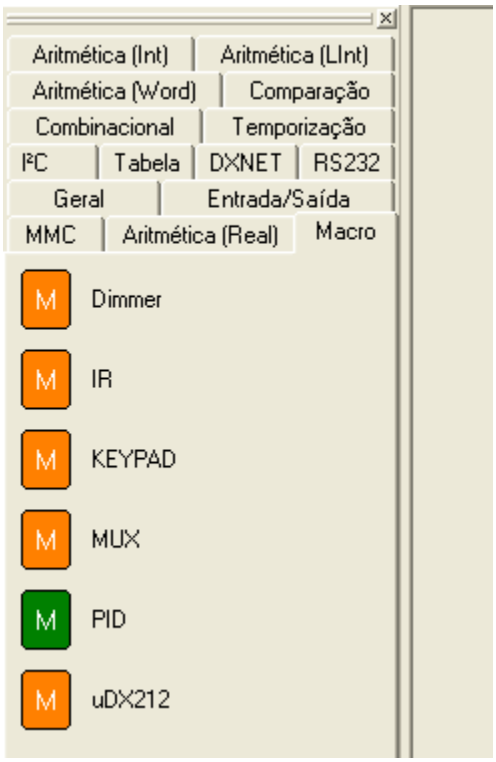
RTD → Leitura de temperatura via sensor RTD conectado à entrada da Expansão µDX218 (tipos PT100, PT200, PT500, PT1000)

Sense Resistor → Resistor interno à Expansão µDX218 usado para leitura de sensores tipo RTD

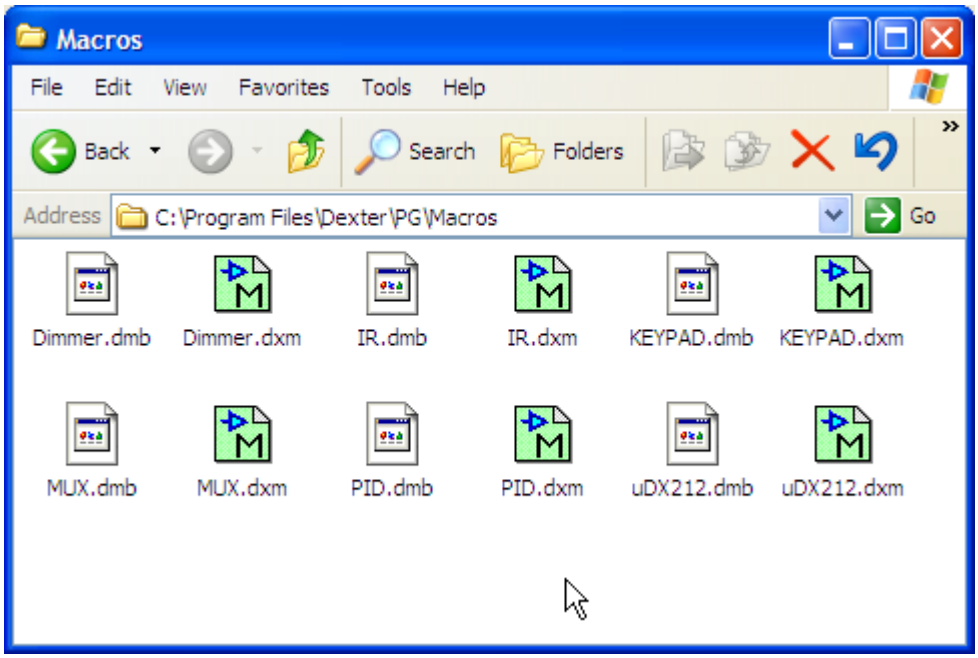
Termopar → Leitura de temperatura via sensor Termopar conectado à entrada da Expansão µDX218 (tipos J, K, E, N, R, S, T, B)

µDX218 → Inicializa e lê Expansão µDX218 (leitura de múltiplos sensores)

Maiores detalhes a respeito dos equipamentos citados acima (Dimmer, Dimmer I^2C, Keypad, MUX, uDX212) podem ser obtidos na documentação que acompanha o CD do µDX200 (veja diretório Manuais). Note que foram incluídos na instalação do PG tanto as Macros compiladas quanto os arquivos que as originaram. Assim, é possível modificá-las para que se adaptem melhor as necessidades de cada aplicação. A lista de Macros é apresentada em uma aba da Biblioteca de Componentes:



As Macros constantes desta aba devem estar no diretório Macros criado na instalação do PG (normalmente em **c:\Arquivos de Programas\Dexter\PG\Macros**):



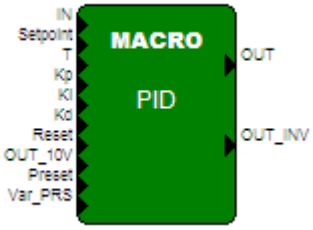
Nas instalações mais recentes do PG os fontes das Macros (sufixo **.dxm**) não são instalados neste diretório, mas apenas as Macros já compiladas (sufixo **.dmb**). Os programas fonte para as Macros, neste caso, podem ser obtidos em **C:\Arquivos de Programas\Dexter\PG\Bin \Exemplos Manual\Macros**.
Atualmente as Macros estão classificadas em grupos, de forma a facilitar sua utilização. Os grupos atuais são os seguintes:

Autom. Industrial	→	Várias funcionalidades usadas na indústria, como controle PID e acionamento de motor de passo.
Autom. Residencial	→	Macros utilizadas em automação residencial e predial, como controle de iluminação e cenários.
Autom. Residencial X	→	Macros para automação residencial e predial otimizadas com uso de novos blocos criados para µDX201 v3.37 ou posterior.
Comunicação	→	Macros para comunicação com controlador µDX101.
IHM	→	Várias rotinas prontas para IHM do µDX201, como entrada de valores e calibração do touchscreen.
IHM X	→	Rotinas otimizadas para IHM do µDX201, mas que necessitam de µDX201 versão 3.52 ou superior.
Relógio	→	Rotinas para uso com relógio interno do controlador (RTC).

Automação Industrial

PID

O controle PID é muito usado para controle de processos em que um simples ON-OFF é inadequado. Assim, para controle de temperatura, por exemplo, em processos de baixa inércia térmica ou temperaturas muito elevadas, pode-se empregar este bloco para gerar um sinal proporcional. Este sinal poderá ser aplicado a uma saída analógica para acionamento das resistências de aquecimento. Sugiro consultar a Internet para maiores detalhes sobre controle PID (a rede possui bastante material a este respeito). A macro **PID** possui dez entradas e duas saídas, como mostrado a seguir:



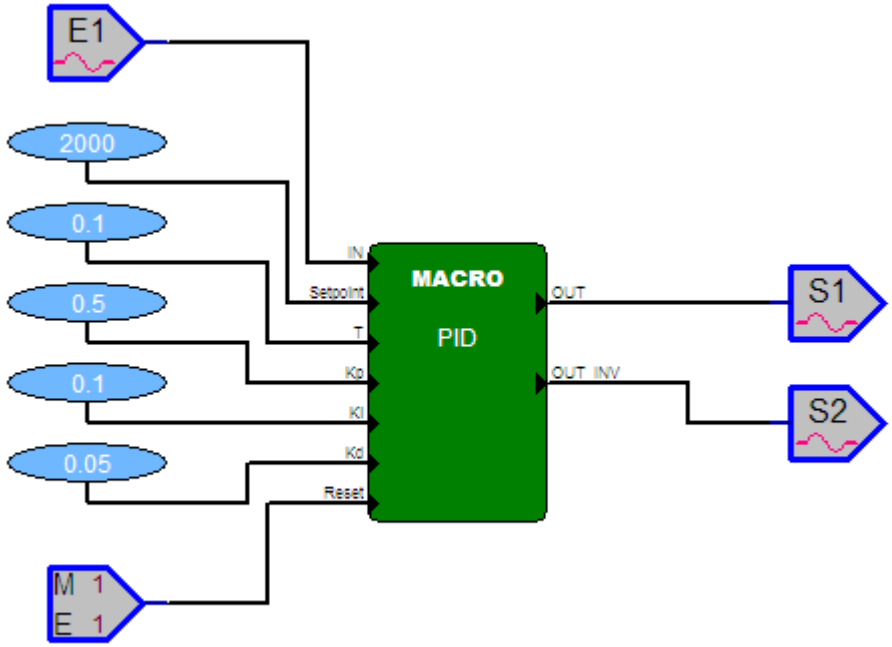
Exemplo de Macro: **PID.dxm**

A entrada **IN** é uma entrada inteira correspondente a variável a ser controlada. Já **Setpoint** é uma variável inteira que indica o valor desejado a ser atingido pela variável em **IN**. **T** é uma entrada real que especifica o tempo de ciclo (em segundos) do controlador PID. **Kp**, **Kd** e **Ki** são os coeficientes reais que fixam o comportamento do controlador. **Reset** é um nodo capaz de reinicializar o controle PID (zerando a componente integral). O nodo de entrada **OUT_10V** comuta as saídas da macro PID para saída em 0-10V, em vez de 4-20mA. Já o nodo de entrada **Preset** permite forçar as saídas para o valor inicial determinado pela entrada inteira **Var_PRS**. Esse recurso permite iniciar o controle PID em qualquer ponto da escala de 4-20mA (ou 0-10V), e não somente a partir do início da escala (4mA ou 0V).

Já as saídas são variáveis inteiras adequadas para conexão à saídas analógicas em corrente. Assim, elas irão gerar um sinal entre 4 e 20mA (as saídas podem assumir valores entre 819 e 4095). Caso o nodo **OUT_10V** esteja energizado as saídas variam de 0 a 4095, adequadas para saída 0-10V. Note a existência de **OUT** e **OUT_INV**. A saída **OUT** deve ser usada quando a saída influencia diretamente a variável controlada. Por exemplo, em um sistema de aquecimento. O aumento de potência de saída aumenta a temperatura. Já a saída **OUT_INV** é usada em processos em que a variável controlada é inversamente proporcional à saída. Por exemplo, em um sistema de resfriamento, no qual quanto maior a saída mais o sistema é resfriado e, portanto, menor é a variável controlada (temperatura). O algoritmo de controle implementado na macro **PID** é dado pela fórmula:

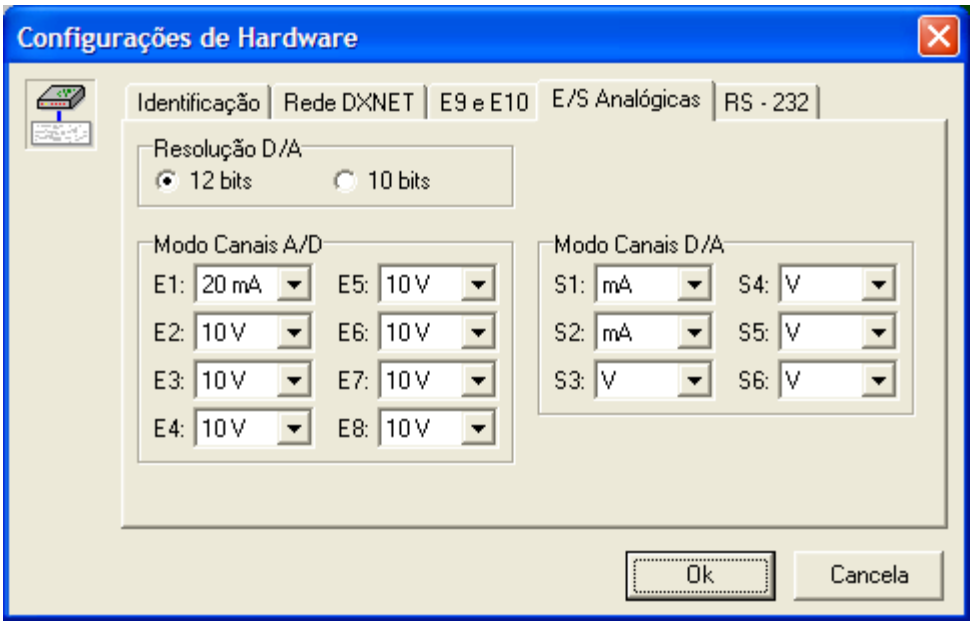
Saída(n) = Saída(n-1) + (Kp+Ki*T+Kd/T) * Erro(n-1) – (2*Kd/T+Kp) * Erro(n-2) + Kd/T * Erro(n-3)

O exemplo a seguir aciona a saída analógica S1 de forma a manter a variável controlada lida na entrada analógica E1 o mais próxima possível do valor de Setpoint (no caso, constante igual a 2000). O período é de 100ms (0,1s) e os coeficientes usados são Kp=0,5, Ki=0,1 e Kd=0,05. A entrada E1 do módulo de Expansão µDX210 permite reinicializar o controle PID. A saída analógica S2 é conectada à saída invertida da macro. Caso S1 = X (sendo $819 \leq X \leq 4095$), então S2 = 4914 - X.



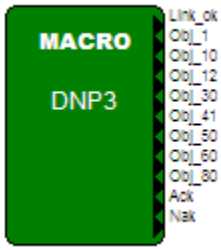
Exemplo de Programa Aplicativo: **Macro - PID.dxg**

As **Configurações de Hardware** devem ser editadas de forma a prever entrada 0-20mA na entrada analógica E1 e nas saídas analógicas S1 e S2, assim como os jumpers destas entradas e saídas devem ser posicionados para corrente (ver Seleção de Jumpers).



DNP3

A implementação do protocolo DNP3 foi efetuada via Macro DNP3 no controlador µDX201. Esta solução permite grande flexibilidade no acréscimo futuro de novas funcionalidades, como mensagens não-solicitadas. As únicas facilidades implementadas em firmware foram referentes ao cálculo de CRC (já que o CRC usado em DNP3 difere do usado pelo protocolo nativo do µDX201 ou do ModBus). Apesar da implementação por blocos, todo programa consumiu cerca de 30% dos recursos do controlador, permitindo acrescer um comportamento relativamente complexo ao CLP. Foi implementada classe 0 (zero) nesta versão, e também classes 1,2,3. A classe 0 retorna todos os dados, independentemente de os mesmos terem variado ou não. As classes 1,2,3 retornam exatamente os mesmos dados da classe 0, ou seja, não existe distinção entre classes na resposta – todas retornam a integralidade dos dados. O endereço do µDX201 a nível de protocolo DNP3 é determinado pelo endereço DXNET do CLP.



Exemplo de Macro: **DNP3.dxm**

Ao inserir esta Macro em um programa aplicativo o µDX201 passa a responder ao protocolo DNP3 tão logo passe para o modo Normal (led de Exec piscando). A Macro possui apenas nodos de saída (variáveis binárias). O nodo **Link_OK**, como o nome já indica, é acionado sempre que há estabelecimento de comunicação via protocolo DNP3. Os nodos **Obj_1** a **Obj_80** geram pulsos sempre que comandos usando os respectivos objetos são recebidos pelo µDX201. Já os nodos **Ack** e **Nak** são acionados momentaneamente sempre que o µDX201 transmite um acknoledge ou no acknoledge a determinado comando recebido via serial RS232.

O protocolo de comunicação, da forma como está implementado no µDX201, disponibiliza 64 nodos de entradas (entradas binárias), 8 nodos de saídas (relés de saída, com comportamento pulsado ou com retenção), e 8 variáveis de entrada e saída (entradas e saídas analógicas de 16 bits). Os nodos de saída estão organizados em pares trip/close, para o caso de ser utilizada esta facilidade. Assim, para cada endereço de saída digital DNP3 existem dois nodos associados, um para trip e outro para close. Estes nodos e variáveis podem ser usados para implementar leituras e comandar saídas do controlador. São eles:

Entradas/Saídas Analógicas (16 bits): variáveis v18 a v25. Note que as variáveis v0 a v17 possuem funcões específicas no µDX201, e por isso não foram usadas. Assim, v0 a v7 corespondem as 8 entradas analógicas, v8 a v13 as 6 saídas analógicas, e v14 a v17 as entradas de contagem rápida. Já as variáveis v18 a v25 disponibilizadas são genéricas, e podem ser usadas para qualquer funcionalidade do programa aplicativo. O acesso via DNP3 é a partir do endereço 0, ou seja, a variável v18 corresponde ao endereço DNP3 zero, v19 ao endereço um, e assim sucessivamente.

Entradas Digitais: nodos 32 a 95. Note que os nodos de 0 a 31 são de sistema e, por isso, não foram utilizados. Os nodos 32 a 95 são genéricos e podem ser usados para qualquer funcionalidade do programa aplicativo. Os endereços de entradas digitais são acessados no protocolo DNP3 a partir do endereço zero.

Saídas Digitais: nodos 96 a 111. Estes nodos podem ser comandados pelo protocolo DNP3 para acionamento momentâneo (pulse on ou pulse off), com tempo programável (entre 10ms a 30s), ou acionamento com retenção (latch on ou latch off). Qualquer destes nodos pode ser usado pelo programa aplicativo para acionar saídas ou determinar pontos de decisão do programa. As saídas digitais são acessadas pelo protocolo DNP3 a partir do endereço zero.

Abaixo temos uma correlação entre as posições vistas pelo protocolo DNP3 e seu correspondente dado (variável ou nodo) no controlador µDX201:

Entradas/Saídas Analógicas:

Endereço DNP3 = 00 → Variável V18
Endereço DNP3 = 01 → Variável V19
Endereço DNP3 = 02 → Variável V20
Endereço DNP3 = 03 → Variável V21
Endereço DNP3 = 04 → Variável V22
Endereço DNP3 = 05 → Variável V23
Endereço DNP3 = 06 → Variável V24
Endereço DNP3 = 07 → Variável V25

Saídas Digitais:

Endereço DNP3 = 00 → Nodo N96 (trip)
Endereço DNP3 = 00 → Nodo N97 (close)
Endereço DNP3 = 01 → Nodo N98 (trip)
Endereço DNP3 = 01 → Nodo N99 (close)
Endereço DNP3 = 02 → Nodo N100 (trip)
Endereço DNP3 = 02 → Nodo N101 (close)
Endereço DNP3 = 03 → Nodo N102 (trip)
Endereço DNP3 = 03 → Nodo N103 (close)
Endereço DNP3 = 04 → Nodo N104 (trip)
Endereço DNP3 = 04 → Nodo N105 (close)
Endereço DNP3 = 05 → Nodo N106 (trip)
Endereço DNP3 = 05 → Nodo N107 (close)
Endereço DNP3 = 06 → Nodo N108 (trip)
Endereço DNP3 = 06 → Nodo N109 close)
Endereço DNP3 = 07 → Nodo N110 (trip)
Endereço DNP3 = 07 → Nodo N111 (close)

Entradas Digitais:

Endereço DNP3 = 00 → Nodo N32
Endereço DNP3 = 01 → Nodo N33
Endereço DNP3 = 02 → Nodo N34
Endereço DNP3 = 03 → Nodo N35
Endereço DNP3 = 04 → Nodo N36
Endereço DNP3 = 05 → Nodo N37
Endereço DNP3 = 06 → Nodo N38
Endereço DNP3 = 07 → Nodo N39
Endereço DNP3 = 08 → Nodo N40
Endereço DNP3 = 09 → Nodo N41
Endereço DNP3 = 10 → Nodo N42
Endereço DNP3 = 11 → Nodo N43
Endereço DNP3 = 12 → Nodo N44
Endereço DNP3 = 13 → Nodo N45
Endereço DNP3 = 14 → Nodo N46
Endereço DNP3 = 15 → Nodo N47
Endereço DNP3 = 16 → Nodo N48
Endereço DNP3 = 17 → Nodo N49
Endereço DNP3 = 18 → Nodo N50
Endereço DNP3 = 19 → Nodo N51
Endereço DNP3 = 20 → Nodo N52
Endereço DNP3 = 21 → Nodo N53
Endereço DNP3 = 22 → Nodo N54
Endereço DNP3 = 23 → Nodo N55
Endereço DNP3 = 24 → Nodo N56
Endereço DNP3 = 25 → Nodo N57
Endereço DNP3 = 26 → Nodo N58
Endereço DNP3 = 27 → Nodo N59
Endereço DNP3 = 28 → Nodo N60

Endereço DNP3 = 32 → Nodo N64
Endereço DNP3 = 33 → Nodo N65
Endereço DNP3 = 34 → Nodo N66
Endereço DNP3 = 35 → Nodo N67
Endereço DNP3 = 36 → Nodo N68
Endereço DNP3 = 37 → Nodo N69
Endereço DNP3 = 38 → Nodo N70
Endereço DNP3 = 39 → Nodo N71
Endereço DNP3 = 40 → Nodo N72
Endereço DNP3 = 41 → Nodo N73
Endereço DNP3 = 42 → Nodo N74
Endereço DNP3 = 43 → Nodo N75
Endereço DNP3 = 44 → Nodo N76
Endereço DNP3 = 45 → Nodo N77
Endereço DNP3 = 46 → Nodo N78
Endereço DNP3 = 47 → Nodo N79
Endereço DNP3 = 48 → Nodo N80
Endereço DNP3 = 49 → Nodo N81
Endereço DNP3 = 50 → Nodo N82
Endereço DNP3 = 51 → Nodo N83
Endereço DNP3 = 52 → Nodo N84
Endereço DNP3 = 53 → Nodo N85
Endereço DNP3 = 54 → Nodo N86
Endereço DNP3 = 55 → Nodo N87
Endereço DNP3 = 56 → Nodo N88
Endereço DNP3 = 57 → Nodo N89
Endereço DNP3 = 58 → Nodo N90
Endereço DNP3 = 59 → Nodo N91
Endereço DNP3 = 60 → Nodo N92

Endereço DNP3 = 29 → Nodo N61
Endereço DNP3 = 30 → Nodo N62
Endereço DNP3 = 31 → Nodo N63
Endereço DNP3 = 61 → Nodo N93
Endereço DNP3 = 62 → Nodo N94
Endereço DNP3 = 63 → Nodo N95

Os objetos implementados são listados a seguir. A comunicação é serial RS232, com número de stop bits, baud-rate e paridade determinados pela configuração de hardware do controlador µDX201. O valor padrão é 38400bps, 8 bits, sem paridade, 2 stop bits. Note que existe uma série de restrições no uso de DNP3 com µDX201, a saber:

- Só é permitido um comando por frame de comunicação (embora este comando possa envolver vários nodos ou variáveis, caso seja leitura. No caso de escrita é necessário comandar individualmente o nodo ou escrever em uma única variável). No caso de questionamento de classe (objeto 60) é permitido que a pergunta contenha mais de um objeto, uma vez que todas as classes respondem com a mesma resposta (todos os dados).
- Não é permitido o uso de múltiplos segmentos nas mensagens (single transport frame).
- Convém especificar um delay de 200ms entre mensagens no software supervisório, de forma a permitir tempo suficiente para que o µDX201 processe os CRCs e elabore a resposta.
- Timeout de resposta é fixo em 3 segundos. Em função disso recomenda-se um timeout de 3,5 ou 4 segundos para o software supervisório.

Objetos implementados:

Objetos implementados			Pergunta (request)		Resposta (response)	
Objeto	**Variação**	**Descrição**	**Função**	**Qualificador**	**Função**	**Qualificador**
1	1	Binary Input	01	00h,01h,06h	129	00h
10	2	Binary Output Status	01	00h,01h,06h	129	00h
12	1	Control Relay Output	03,04,05,06	28h	129	28h
30	4	16-Bit Analog Input	01	00h,01h,06h	129	00h
41	2	16-Bit Analog Output	05, 06	28h	129	28h
50	1	Date & Time	02	07h	129	-
60	1	Classe 0	01	06h	129	00h
60	2	Classe 1	01	06h	129	00h
60	3	Classe 2	01	06h	129	00h
60	4	Classe 3	01	06h	129	00h
80	1	Internal Indications	02	00h	129	-
-	-	Delay Measurement	23	-	-	-

Note que a leitura de nodos de entrada é efetuada pelo objeto 1, variação 1. Já para leitura dos nodos de saída deve-se usar objeto 10, variação 2. Os nodos de saída possuem comportamento especial, uma vez que podem ser pulsados durante determinado tempo, ou se manterem ligados ou desligados. Então, necessariamente, saídas digitais devem ser implementadas com os nodos 96 a 111, e entradas digitais devem comandar nodos de 32 a 95. O acionamento de nodos de saída é feita pelo objeto 12, variação 1.
Já no caso das entradas e saídas analógicas, como o comportamento de todas é idêntico, e quaisquer variáveis de 18 a 25 podem ser usadas tanto como entradas analógicas ou saídas analógicas. Para leitura de variáveis (16 bits) usa-se o objeto 30, variação 4, e para escrita em variáveis analógicas (16 bits) emprega-se o objeto 41, variação 2.
Foi implementada leitura de classe 0, que retorna o conteúdo de todas as variáveis (v18 a v25), o estado de todos os nodos de entrada (n32 a n95), e o estado dos nodos de saída (n96 a n103). Também a leitura de classe 1/2/3 retorna todas variáveis e nodos.

Exemplos de comunicação (µDX200 em endereço DXNET 1)

Estabelecimento de link de comunicação (link reset)
TX: 05 64 05 C0 01 00 0A 00 E0 8C
RX: 05 64 05 00 0A 00 01 00 7F FD

Classe 0 (object 60, variation 1, qualifier 6)
TX: 05 64 0B C4 01 00 0A 00 FD F1 C0 C0 01 3C 01 06 FF 50
RX: 05 64 5E 44 0A 00 01 00 ED 9E EE C0 81 80 00 1E 04 00 12 31 00 00 00 00 00 00 4B D5 00 00 00 00 00 00 00 00 FE FF FF FF 00 00 00 00 6C 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 00 00 00 00 FD FF 00 00 00 00 00 00 00 00 00 00 CB 71 00 00 00 00 00 00 00 00 00 00 01 01 00 20 6F 01 3C 0F 00 80 00 00 00 00 00 01 00 00 7B

Leitura de nodo 55 (object 1, variation 1, qualifier 1)
TX: 05 64 0F C4 01 00 0A 00 93 BC C0 C1 01 01 01 01 37 00 37 00 B6 22
RX: 05 64 10 44 0A 00 01 00 95 C1 EF C1 81 80 00 01 01 00 37 37 01 6E 73

Leitura de nodos 33 a 111 (object 1, variation 1, qualifier 1)
TX: 05 64 0F C4 01 00 0A 00 93 BC C0 C2 01 01 01 01 21 00 6F 00 BE A4
RX: 05 64 19 44 0A 00 01 00 AE EE F1 C2 81 80 00 01 01 00 21 6F 00 00 40 00 00 00 70 A7 00 80 00 00 58 AA

Leitura de variáveis 20 a 39 (object 30, variation 4, qualifier 1)
TX: 05 64 0F C4 01 00 0A 00 93 BC C0 C6 01 1E 04 01 14 00 27 00 A5 FD
RX: 05 64 37 44 0A 00 01 00 29 EE F5 C6 81 80 00 1E 04 00 14 27 00 00 00 00 00 00 A4 45 00 00 00 00 FE FF FF FF 00 00 00 00 00 00 00 00 4B 45 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FD FF DF 82

Leitura de variável 18 (object 30, variation 4, qualifier 1)
TX: 05 64 0F C4 01 00 0A 00 93 BC C0 C7 01 1E 04 01 12 00 12 00 63 87
RX: 05 64 11 44 0A 00 01 00 72 74 F6 C7 81 80 00 1E 04 00 12 12 00 00 A3 F1

Desacionamento de nodo 96 (latch off) (object 12, variation 1, qualifier 28h)
TX: 05 64 1A C4 01 00 0A 00 DB 3C C0 CA 05 0C 01 28 01 00 60 00 04 01 D0 07 00 00 81 99 D0 07 00 00 00 C9 D2
RX: 05 64 1C 44 0A 00 01 00 27 16 F9 CA 81 80 00 0C 01 28 01 00 60 00 04 01 D0 07 4D C1 00 00 D0 07 00 00 00 C9 D2

Acionamento de nodo 97 (pulse on) durante 2 segundos (object 12, variation 1, qualifier 28h)
TX: 05 64 1A C4 01 00 0A 00 DB 3C C0 CC 05 0C 01 28 01 00 61 00 01 01 D0 07 00 00 33 5E D0 07 00 00 00 C9 D2
RX: 05 64 1C 44 0A 00 01 00 27 16 FB CC 81 80 00 0C 01 28 01 00 61 00 01 01 D0 07 58 D8 00 00 D0 07 00 00 00 C9 D2

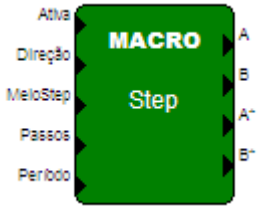
Atualização de data e hora do CLP, com confirmação (object 50, variation 1, qualifier 7)
TX: 05 64 12 F3 01 00 03 00 00 39 C2 C3 02 32 01 07 01 98 8E 83 F2 27 01 EF 92
RX: 05 64 05 00 03 00 01 00 B8 CA
RX: 05 64 0A 44 03 00 01 00 F8 32 C1 C3 81 80 00 F4 DA

Classe 3/2/1/0, com confirmação (object 60, variation 4,3,2,1, qualifier 6)
TX: 05 64 14 F3 01 00 03 00 D9 52 C0 C1 01 3C 04 06 3C 03 06 3C 02 06 3C 01 06 4A D2
RX: 05 64 05 00 03 00 01 00 B8 CA
RX: 05 64 5E 44 03 00 01 00 2A A9 C3 C1 81 80 00 1E 04 00 12 31 00 00 00 00 00 00 C7 76 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 00 00 00 00 00 00 00 00 00 00 01 01 00 20 6F 00 62 39 00 00 00 00 00 00 00 00 00 FF FF

Delay Measurement (no-object, function 23)
TX: 05 64 08 C4 01 00 0A 00 AD 62 C0 C0 17 C2 A7
RX: 05 64 10 44 0A 00 01 00 95 C1 C2 C0 81 80 00 34 02 07 01 46 00 8A B8

Step

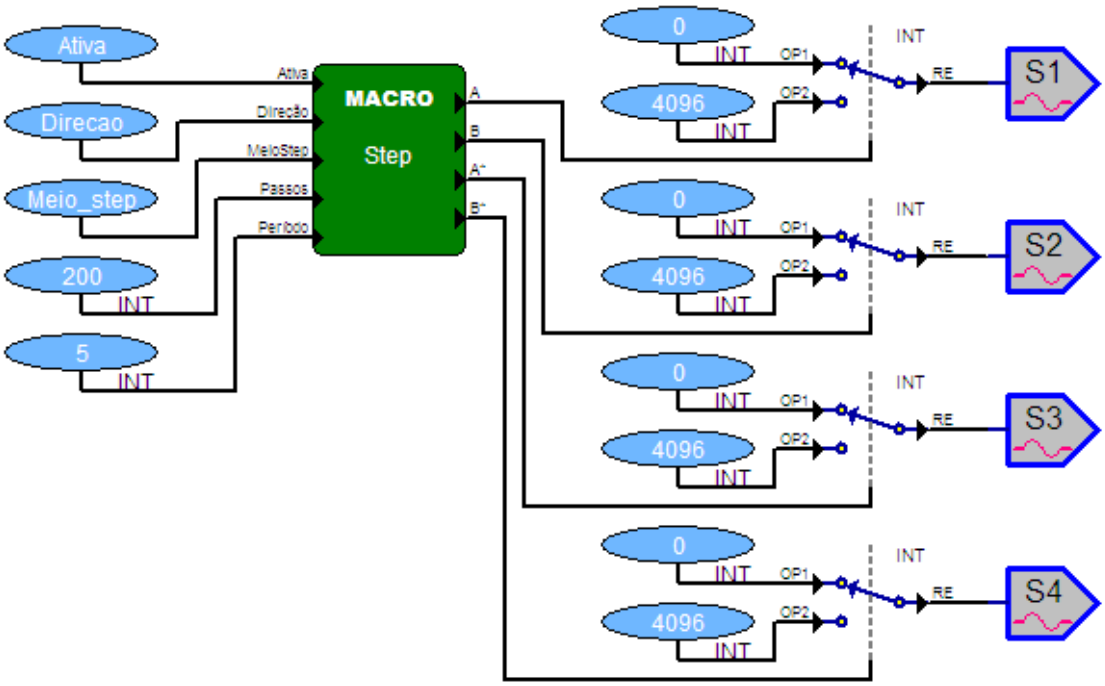
Esta macro gera acionamento para motor de passo unipolar (5 ou 6 fios). Abaixo sua representação esquemática:



Exemplo de Macro: **Step.dxm**

As entradas Ativa, Direção e MeioStep são binárias, e permitem acionar a macro, especificar a direção do giro do motor, e se ele irá se movimentar em meio passo ou passo completo. Já as entradas Passos e Período são variáveis inteiras. A primeira entrada especifica quantos passos serão executados a cada energização da entrada Ativa. Já a entrada Periodo especifica quantos milissegundos de pausa entre cada passo. Já as saídas A, B, A* e B* são os acionamentos de cada enrolamento do motor.

Para testar a macro foi elaborado o programa a seguir. Note que foram usadas as saídas analógicas para acionamento das fases do motor. Para tornar as saídas analógicas rápidas elas foram comutadas para saídas PWM (tanto na Configuração de Hardware do programa aplicativo quanto nos jumpers internos do µDX200). Estas saídas acionam transistores adequados à corrente de atuação do motor de passo usado.



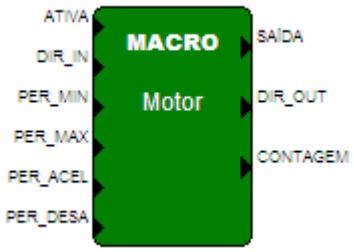
Exemplo de Programa Aplicativo: **Macro - Step.dxg**

No caso foram usados 200 passos a cada acionamento do nodo Ativa, com um intervalo de 5ms

entre eles. Como o motor usado possua 200 passos por volta (passo de 1,8°) resulta em um volta completa em 1 segundo. Caso seja acionado o nodo Meio_step o motor irá efetuar meia volta em 1 segundo, já que irá se deslocar de meio em meio passo (0,9°).

Motor

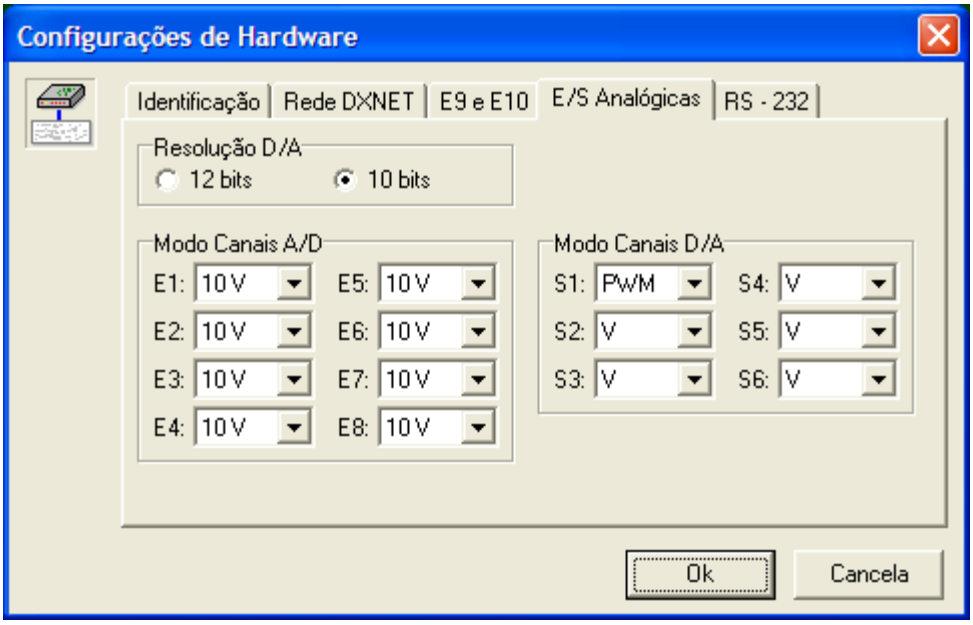
Esta macro gera pulsos para controlador (driver) de motor de passo. Ela permite programar velocidade máxima e mínima, tempo de aceleração e desaceleração, além de possuir entrada e saída para direção:



Exemplo de Macro: **Motor.dxm**

As entradas ATIVA e DIR_IN são binárias, e permitem acionar o motor e especificar a direção do giro do motor, Já PER_MIN, PER_MAX, PER_ACEL e PER_DESA são entradas inteiras que especificam, em milissegundos, o período mínimo e máximo, e o período de aceleração e desaceleração. Já a saída inteira SAIDA deve ser ligada a uma saída analógica do µDX201 programada para saída PWM. A saída DIR_OUT e binária e informa a direção de giro para o driver de controle do motor de passo, e pode ser usada uma saída digital (saída do µDX210), ou uma saída analógica comutando entre dois valores, como no exemplo a seguir. A saída CONTAGEM é uma saída longint e conta o número de passos efetuados pelo motor.

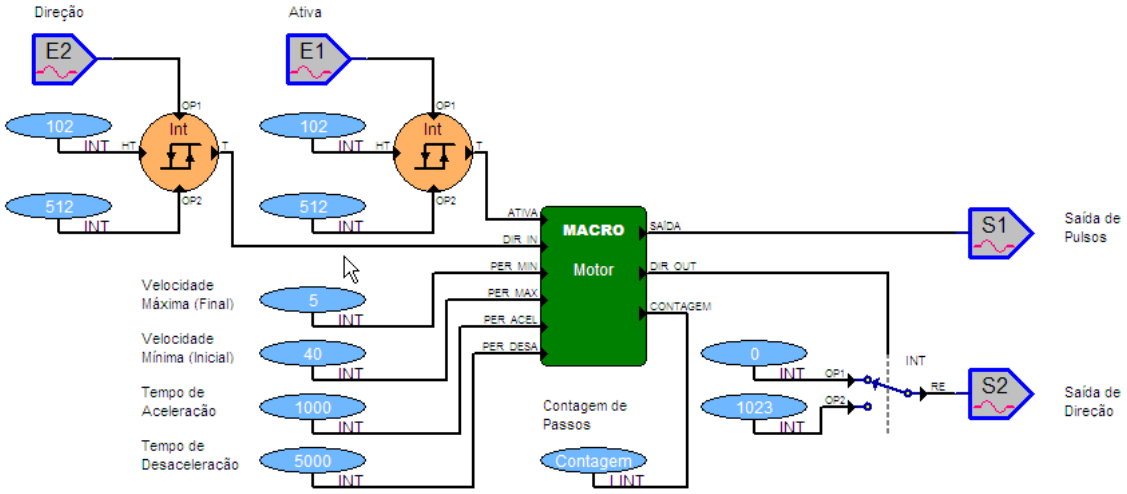
Para usarmos toda a velocidade possível (máximo de 1000 pulsos/segundo na saída) é imprescindível usar um controlador µDX201 (o µDX200 não tem velocidade suficiente para esta velocidade, ficando restrito a cerca de 250 pulsos/segundo). Além disso, as saídas analógicas devem ser programadas para resolução de 10 bits na Configuração de Hardware:



O exemplo a seguir exemplifica o uso desta macro. Note que foram usadas duas entradas analógicas para ativar o motor e para especificar direção. Como foi usado um bloco de histerese com ponto de comparação em 512 e histerese de 102 teremos o ponto de decisão em 512*10V/4095 = 1,25V, e histerese de 102*10V/4095 = 0,25V. Significa que a entrada será considerada ligada quando atingir 1,25V+0,25V = 1,5V, e será considerada desligada ao cair abaixo de 1,25V-0,25V = 1V.

O período mínimo é de 5ms, o que resulta em uma taxa de pulsos máxima de 1/0,005 = 200 pulsos/segundo. Já o período máximo é de 40ms, resultando em 1/0,04 = 25 pulsos/segundo. Portanto, ao iniciar o movimento o motor irá fazer 40 passos/s, e irá acelerando até 200 passos/s, e vice-versa na desaceleração. O período de aceleração é de 1000 ms, ou seja, 1 segundo. Já o período de desaceleração é de 5 segundos.

A saída de direção foi ligada a uma chave inteira, que comuta a saída S2 entre 0V e 10V (note que as saídas analógicas estão programadas para resolução de 10 bits - 1024 steps).

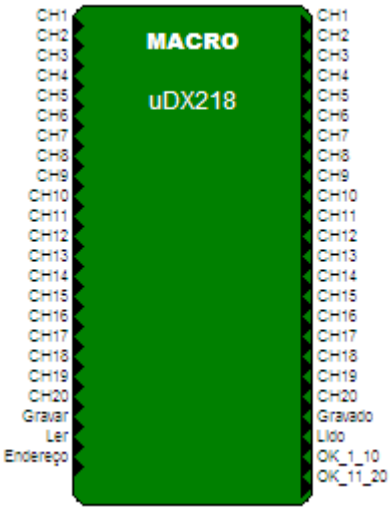


Exemplo de Programa Aplicativo: **Teste_Motor.dxg**

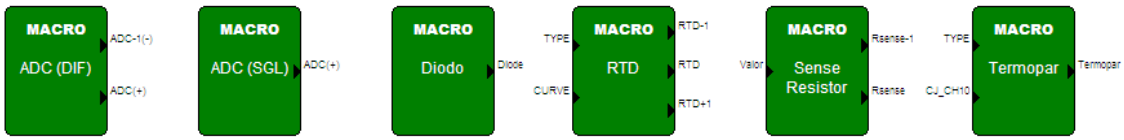
uDX218

Esta macro inicializa e lê Expansões µDX218 ligada ao controlador µDX218 via conector de expansão. A Expansão µDX218 possui 20 entradas analógicas de alta resolução (24 bits), permitindo a leitura de múltiplos sensores, como RTDs (Resistance Temperature Detector ou Termorresistência), Termopares, strain-gauges (células de carga), etc.

Esta macro permite programar a funcionalidade de cada entrada (CH1 a CH20), indicar o endereço do µDX218 (é possível conectar até 4 µDX218s a um único µDX201), e ler os valores digitalizados em variáveis LongInt (32 bits).



As entradas **CH1** a **CH20** existentes à esquerda da macro **uDX218** programam a funcionalidade das correspondentes entradas da Expansão µDX218. Para isso existe uma série de outras macros, cada uma permitindo especificar um tipo de sensor. Seriam elas:



Vamos descrever cada uma dessas macros:

ADC (DIF) → Leitura de entrada analógica diferencial na Expansão µDX218 (até +/-1,25V com 24 bits de resolução)

Com esta macro podemos alocar duas entradas analógicas do µDX218 de forma a obter uma leitura diferencial dessas entradas. Note que esse seria o caso para, por exemplo, a leitura de células de carga, usadas para medição de peso. É preciso determinar a diferença de tensão presente entre dois braços de uma ponte de Wheatstone. Note que o próprio µDX218 possui uma fonte de tensão programável para alimentar a célula de carga. É possível especificar a tensão de saída para 2,5V, 5V, 7,5V e 10V (corrente máxima de 100mA). Esta fonte é muito estável e isolada da alimentação elétrica do controlador µDX201. A tensão diferencial nas entradas pode variar entre +/-1,25V e é avaliada com resolução de 24 bits (step de 477nV).

ADC (SGL) → Leitura de entrada analógica referenciada ao GND (single-ended) na Expansão µDX218 (até +/-1,25V com 24 bits de resolução)

Com esta macro podemos alocar uma entrada analógicas do µDX218 de forma a obter uma leitura da tensão dessa entrada. Nesse caso apenas uma entrada analógica é usada e a tensão nela é referenciada ao comum da alimentação (GND). A tensão diferencial nas entradas pode variar entre +/-1,25V e é avaliada com resolução de 24 bits (step de 477nV).

Diode → Leitura de temperatura via diodo conectado à entrada da Expansão µDX218

Esta macro indica que um diodo está conectado à entrada analógica do µDX218. Normalmente esse diodo é usado para determinar a temperatura de junta fria quando são usados termopares. O µDX218 possui dois sensores desse tipo internamente, que podem ser habilitados via estrapes (jumpers). Eles podem ser ligados as entradas CH10 e CH20.

RTD → Leitura de temperatura via sensor RTD conectado à entrada da Expansão µDX218 (tipos PT100, PT200, PT500, PT1000)

A macro **RTD** usa três entradas analógicas do µDX218 para medir um sensor RTD (Resistance Temperature Detector ou Termorresistência). A entrada **Type** permite escolher o tipo de sensor:

PT100, PT200, PT500 ou PT1000. Já a entrada **CURVE** determina o tipo de curva a ser usada: Americana, Européia, Japonesa ou ITS-90. Note que para uso desse tipo de sensor é mandatório usar um resistor de comparação (Sense Resistor), determinado pela próxima macro.

Sense Resistor → Resistor interno à Expansão µDX218 usado para leitura de sensores tipo RTD

A macro **Sense Resistor** especifica em quais entradas está conectado o resistor de comparação usado para leitura de RTDs, e também seu valor. A Expansão µDX218 possui um Sense Resistor interno ligado as entradas CH1 e CH2 que pode ser habilitado via estrapes (jumpers). Seu valor também pode ser selecionado entre 5,00K ou 10,0K.

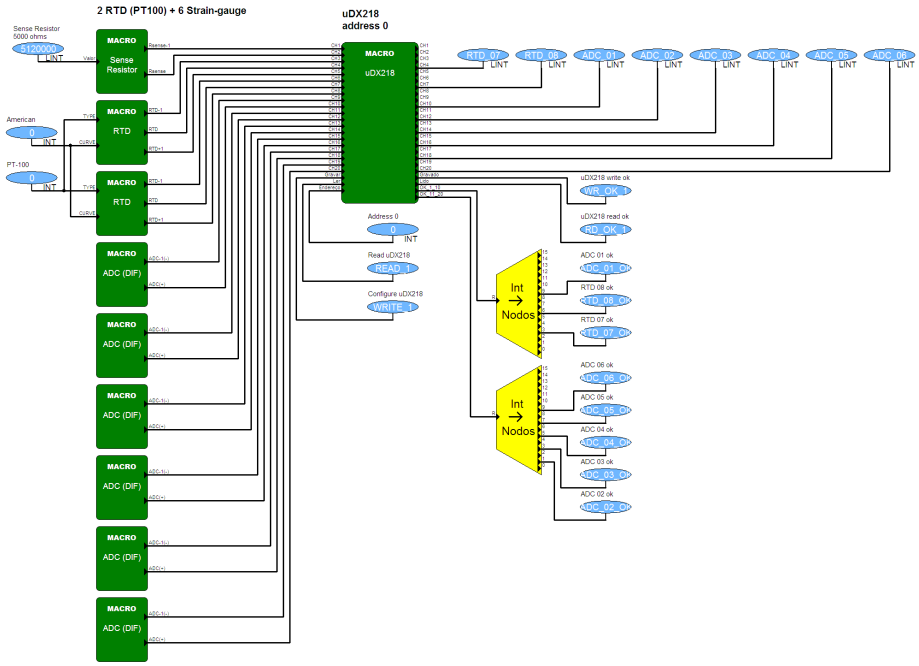
Termopar → Leitura de temperatura via sensor Termopar conectado à entrada da Expansão µDX218 (tipos J, K, E, N, R, S, T, B)

A macro **Termopar** aloca uma entrada analógica do µDX218 para leitura de Termopar. A entrada **TYPE** especifica o tipo de termopar usado: J, K, E, N, R, S, T, B. Já a entrada **CJ_CH10** é uma entrada binária que, quando energizada, permite especificar que o diodo de compensação de junta fria está ligado à entrada CH10 e não na entrada CH20. Note que esse tipo de sensor sempre necessita ler a temperatura ambiente, uma vez que a tensão em seus terminais é função da diferença de temperatura entre o sensor e o ambiente.

As entradas **Gravar** e **Ler** são entradas binárias que disparam a programação das entradas do µDX218 ou sua leitura. A entrada **Endereço** é uma entrada inteira que permite endereçar a expansão µDX218 (endereços de 0 a 3).

As saídas **CH1** a **CH20** da Macro uDX218 são variáveis LongInt com o valor da conversão analógica/digital de cada entrada. As saídas binárias **Gravado** e **Lido** indicam que as respectivas operações foram executadas. As saídas inteiras **OK_1_10** e **OK_11_20** indicam o status de cada entrada analógica do µDX218. Cada bit ligado nessas variáveis indica que a entrada analógica correspondente está ok, não apresentando erro.

Para tornar mais claro o uso dessas macros vamos examinar um exemplo, no qual são lidos dois sensores RTD e 6 strain-gauges. Note que as entradas analógicas CH1 e CH2 são usadas para o sensor de comparação (sense resistor) para os RTDs (no caso foram usados sensores PT100 e resistor de comparação de 5K). Para programar o µDX218 é preciso gerar um pulso momentâneo no nodo WRITE_1, e para leitura das entradas é preciso gerar um pulso momentâneo no nodo READ_1.

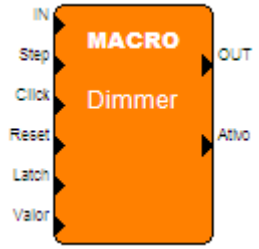


Exemplo de Programa Aplicativo: **uDX218_Exemplo.dxg**

Automação Residencial

Dimmer

Esta Macro permite acionar uma unidade de Dimmer ligada a uma das seis saídas analógicas do controlador µDX200. O Dimmer permite modular o acionamento de motores, resistências de aquecimento ou lâmpadas incandescentes em até 8000 passos entre 0 e 100%. A potência controlada pode atingir 2200W em 220Vac. Esta macro deve ser usada com rede elétrica de 60Hz. No caso de rede de 50Hz usar a macro **Dimmer 50Hz**. O bloco desta Macro tem o seguinte aspecto:



Exemplo de Macro: **Dimmer.dxm**

Existem seis entradas e duas saídas. A entrada IN é do tipo lógico ou binária (nodo), ou seja, está ligada ou desligada, e serve para comandar o Dimmer. Um pulso breve nesta entrada comuta o Dimmer entre acionamento no último valor programado e desligado. Já retendo esta entrada acionada o Dimmer incrementa ou decrementa seu ponto de operação.

A entrada Step é uma entrada inteira, e indica de quantos em quantos passos (steps) é feito o incremento ou decremento do ponto de operação do Dimmer. Como ele possui 8000 passos entre acionamento desligado e máxima potência a passagem de 0 a 100% pode levar tempo considerável se for feita passo a passo. Isso não é prático, e também a diferença entre um passo e outro é tão sutil que não é perceptível em um sistema de iluminação. Esta entrada permite fixar um passo para cada incremento ou decremento do Dimmer. Por exemplo, se fixarmos a esta entrada uma constante de valor 100 reduziremos o Dimmer a 80 passos.

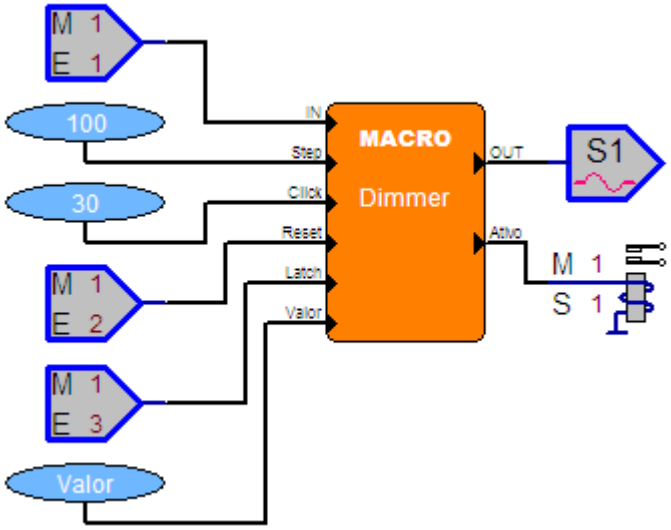
A entrada Click determina qual o tempo em dezenas de milisegundos (x 10ms) que será considerado como pulso breve na entrada IN. Assim, se o pulso demorar mais do que o especificado na entrada Click ele será considerado como mudança de ponto de operação e não como comutação entre ligar e desligar Dimmer. Esta entrada é uma entrada inteira. Por exemplo, se colocarmos uma constante inteira de valor 30 nesta entrada significa que pulsos em IN mais breves que 300ms irão comutar o Dimmer entre ligado e desligado. Já pulsos maiores que 300ms irão modificar seu ponto de operação (incrementando ou decrementando segundo a entrada Step).

A entrada binária Reset desliga o Dimmer, restabelecendo as condições iniciais da Macro. Já a entrada binária Latch, quando acionada, força que o Dimmer assuma o valor (de 0 a 100%) especificado na entrada inteira Valor. Com isso, fica fácil atuar o Dimmer segundo determinado cenário. Basta especificar na entrada Valor qual a iluminação requerida (variável entre 0 e 100, correspondendo a 0 a 100% de atuação do Dimmer) e energizar a entrada Latch.

A saída OUT é uma variável inteira a ser conectada a uma das seis saídas analógicas do controlador µDX200. Note que é preciso comutar as saídas analógicas para saída PWM, e usar o Sensor de Zero conectado a uma das entradas de contagem rápida (E9 ou E10). Maiores detalhes podem ser obtidos no manual do Dimmer.

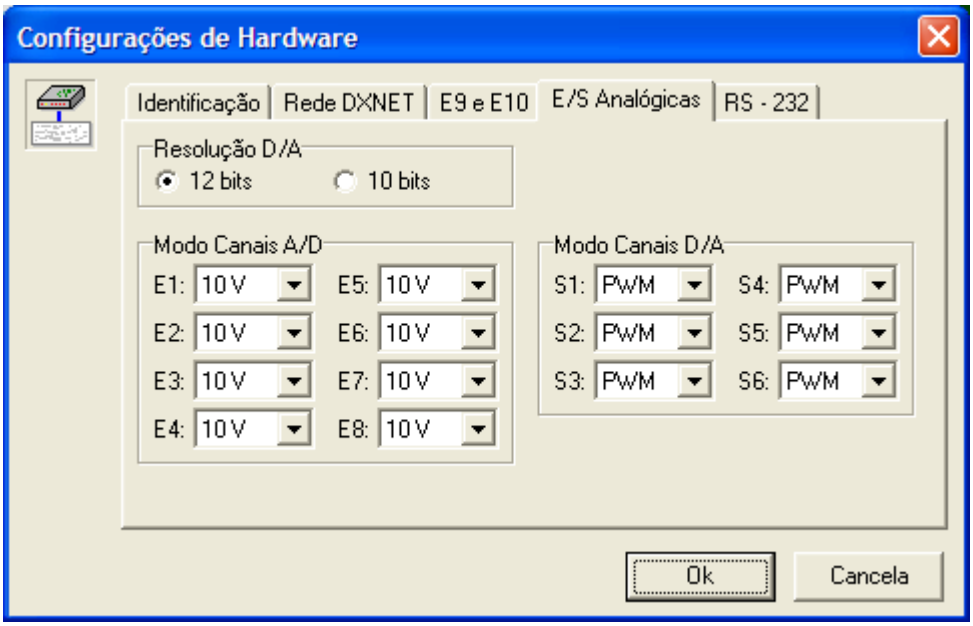
A saída lógica ou binária Ativo indica que o Dimmer não está desligado. Esta saída é útil para gerar um feedback de que o Dimmer está ligado. Por exemplo, atuando sobre o led do Keypad que controla este Dimmer.

Um possível programa de exemplo do uso desta Macro é mostrado a seguir. Note que usamos a saída analógica S1 para acionar o Dimmer, e a entrada E1 do módulo de Expansão µDX210 para comandá-la, a entrada E2 para reset, e a entrada E3 para transferir o dado na variável Valor para a saída do Dimmer:



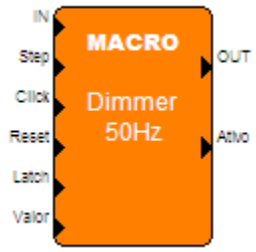
Exemplo de Programa Aplicativo: **Macro - Dimmer.dxg**

Note que não é possível utilizar apenas uma saída analógica do µDX200 para acionamento de Dimmer. Uma vez selecionada a **Função Dimmer Ativada** nas **Configurações de Hardware** todas as saídas analógicas serão sincronizadas pela rede elétrica, ficando aptas a acionar unidades de Dimmer. Devemos usar as **Configurações de Hardware** como mostrado a seguir:



Dimmer 50Hz

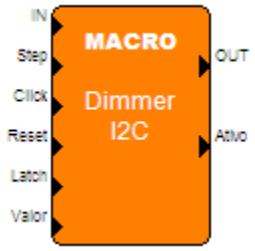
Esta Macro é idêntica à anterior, mas adaptada para redes elétricas de 50Hz. No caso desta macro o Dimmer permite modular o acionamento de motores, resistências de aquecimento ou lâmpadas incandescentes em até 10000 passos entre 0 e 100%. A potência controlada pode atingir 2200W em 220Vac. O bloco desta Macro tem o seguinte aspecto:



Exemplo de Macro: **Dimmer_50Hz.dxm**

Dimmer I²C

Esta Macro permite acionar uma unidade de Dimmer ligada a rede I²C do controlador µDX200. O Dimmer permite modular o acionamento de lâmpadas incandescentes em 100 passos (de 0 e 100%). A potência controlada pode atingir 260W em 220Vac. Esta macro está adaptada para uso tanto em rede elétrica de 60Hz (como no Brasil) quanto em 50Hz (Uruguai). Entretanto, para isso é preciso que o mini-dimmer usado seja de versão 2.2 ou superior. O bloco desta Macro tem o seguinte aspecto:



Exemplo de Macro: **Dimmer_I2C.dxm**

Existem seis entradas e duas saídas. A entrada IN é binária (nodo), ou seja, está ligada ou desligada, e serve para comandar o Dimmer I^2C. Um pulso breve nesta entrada comuta o Dimmer entre acionamento no último valor programado e desligado. Já retendo esta entrada acionada o Dimmer incrementa ou decrementa seu ponto de operação.

A entrada Step é uma entrada inteira, e indica de quantos em quantos passos (steps) é feito o incremento ou decremento do ponto de operação do Dimmer. Normalmente, se utiliza incremento unitário ou valores pequenos, como de 5 em 5, já que o número de passos disponíveis no Dimmer I^2C não é elevado (100 passos).

A entrada Click determina qual o tempo em dezenas de milisegundos (x 10ms) que será considerado como pulso breve na entrada IN. Assim, se o pulso demorar mais do que o especificado na entrada Click ele será considerado como mudança de ponto de operação e não como comutação entre ligar e desligar Dimmer. Esta entrada é uma entrada inteira. Por exemplo, se colocarmos uma constante inteira de valor 30 nesta entrada significa que pulsos em IN mais breves que 300ms irão comutar o Dimmer entre ligado e desligado. Já pulsos maiores que 300ms irão modificar seu ponto de operação (incrementando ou decrementando segundo a entrada Step).

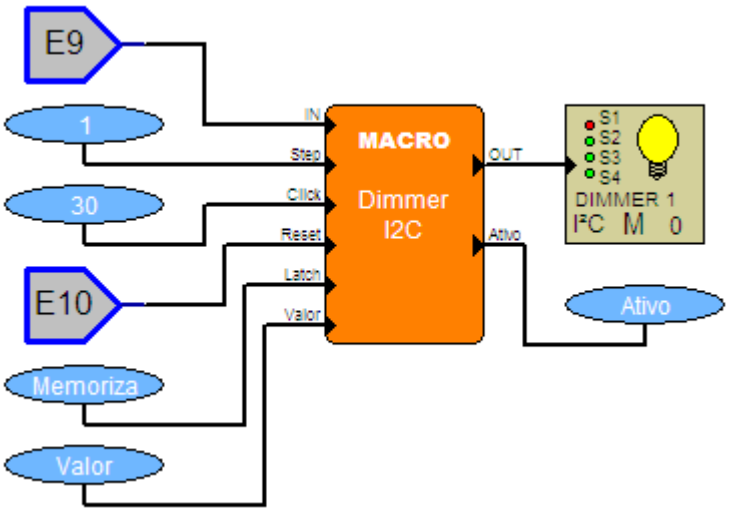
A entrada binária Reset reinicializa toda a macro de Dimmer I^2C, desligando o canal controlado pelo Dimmer I^2C.

A entrada binária Latch permite atualizar o valor do Dimmer instantaneamente, fazendo com que o mesmo assuma o valor especificado na entrada inteira Valor (de 0 a 100, correspondendo a 0 a 100% de iluminação). Este recurso é útil para acionar cenários. Basta colocar o valor de dimmerização desejado na entrada Valor e acionar momentaneamente a entrada Latch.

A saída OUT é uma variável inteira a ser conectada a um bloco de controle de Dimmer I^2C. Note que existem 4 blocos diferentes, correspondentes aos quatro canais existentes em cada Dimmer I^2C. O Dimmer I^2C já possui sensor de zero incorporado, de forma que, ao contrário do Dimmer tradicional, não requer o uso de Sensor de Zero.

A saída binária Ativo indica que o Dimmer I^2C não está desligado. Esta saída é útil para gerar um feedback de que o Dimmer está ligado. Por exemplo, atuando sobre o led do Keypad que controla este Dimmer.

Um possível programa de exemplo do uso desta Macro é mostrado a seguir. Note que usamos a saída S1 do Dimmer I^2C endereço 0, a entrada E9 do controlador µDX200 para comandá-la, a entrada E10 para reset, e o nodo Memoriza para transferir o dado na variável Valor para a saída do Dimmer I^2C:



Exemplo de Programa Aplicativo: **Macro - Dimmer I2C.dxg**

O step de incremento/decremento foi fixado em unitário, e o tempo de detecção de pulso em E9 em 300ms. A entrada E10 permite resetar o Dimmer. Já o nodo Memoriza permite transferir o valor especificado na variável Valor (de 0 a 100%) para a saída dimmerizada.

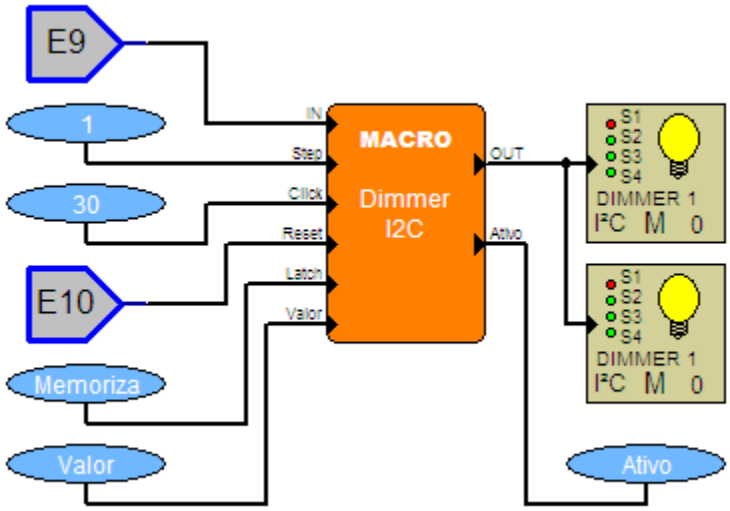
Consideração importante sobre a rede I2C

Como a rede I^2C não possui uma checagem de erros ela é passível de falhas e má interpretação das mensagens. Para evitar isso tanto o dimmer I^2C (min-dimmer) quanto os módulos µDX212 exigem duas mensagens idênticas e válidas para executarem uma operação. Isto foi inserido como forma de validação dos dados recebidos. Ocorre que no incremento ou decremento do nível de dimmerização o dado transmitido é modificado a cada 100ms na Macro **Dimmer I2C**. Ora, cada canal do dimmer I^2C acrescenta um atraso de 8,5ms à rede I^2C. Então, até cerca de 5 canais o atraso na rede I^2C é metade de 100ms, o que garante que sempre os valores são escritos pelo menos duas vezes na rede e o dimmer o aceita. Para mais de 5 canais até 11 canais o atraso da rede é inferior a 100ms e, neste caso, conforme o sincronismo entre a escrita do novo valor e a varredura da rede I^2C, o valor é aceito ou não pelo mini-dimmer. Assim mesmo ainda ocorre atualização eventualmente. Mas acima de 11 canais a rede I^2C possui um atraso superior a 100ms e, portanto, nunca o dado será repetido (já que o mesmo é incrementado a cada 100ms). Com isso o mini-dimmer não o aceita, e só irá aceitar o dado quando a tecla for solta, pois aí o dado estabiliza em um valor fixo. Evidentemente a presença de outros dispositivos I^2C na rede irá acrescer mais atrasos, o que irá agravar o problema. Abaixo temos uma lista dos atrasos inseridos por cada dispositivo:

Δt Keypad = 15,5ms
Δt Keypad (16 bits) = 18,9ms
Δt µDX212 = 15,5ms
Δt IR-TX = 15,5ms
Δt Dimmer = 34,0ms (4 canais)
Δt Umidade = 15,5ms
Δt Temperatura = 23,3ms
Δt Temperatura (µDX100) = 26,7ms
Δt Temperatura (µDX200) = 26,7ms
Δt µDX215 = 18,9ms (2 módulos)

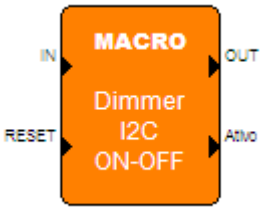
O atraso máximo recomendável na rede I^2C é de 275 ms. Então, para garantir uma atualização a cada incremento ou decremento, devemos efetuar este incremento/decremento em intervalos superiores a 2 x 275ms (de forma a garantir duas leituras do mini-dimmer). Mas tempos de

550ms entre cada incremento/decremento gera uma rampa irregular, com degraus visíveis de luminosidade. Então, a solução é colocar dois blocos **Dimmer** idênticos na saída da Macro **Dimmer I2C**. Isso força o Compilador PG a incluir duas vezes esta saída dimmer na lista de varredura I²C, garantindo que haja duas escritas idênticas e, portanto, elas sejam aceitas pelo mini-dimmer.



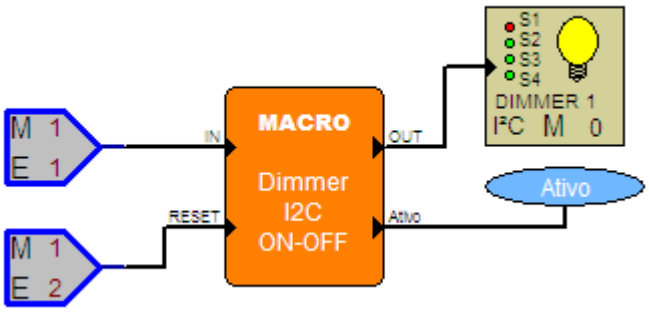
Dimmer I²C ON-OFF

O módulo Mini-Dimmer permite modular entre 0 e 100% de iluminação, mas isso exige que as lâmpadas permitam controle dimmer. No caso de lâmpadas incandescentes ou halógenas todas permitem este controle. Já no caso de lâmpadas LED apenas alguns modelos disponibilizam tal funcionalidade. No caso de lâmpadas LED ou lâmpadas fluorescentes que não admitem controle dimmer é preciso usar esta macro para fazer um controle liga-desliga (on-off), inibindo a rampa de subida e descida de iluminação que o Mini-Dimmer normalmente utiliza. Com esta macro a iluminação comuta entre 0 e 100% diretamente, sem valores intermediários.



Exemplo de Macro: **Dimmer_I2C_onoff.dxm**

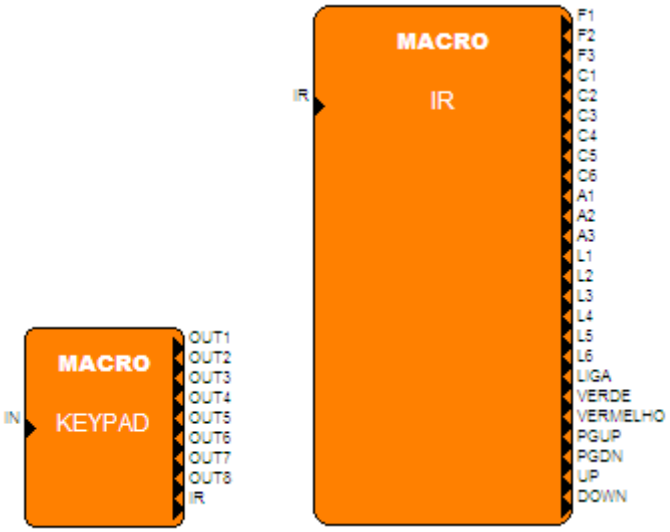
A entrada **IN** é binária (ligada ou desligada) e comuta a saída **OUT** (saída inteira) entre 0 e 100% de iluminação no canal do Mini-Dimmer. A entrada binária **RESET** permite reinicializar a macro, forçando o desligamento do canal do mini-dimmer (0% de iluminação). Por fim, a saída **Ativo** (saída binária) indica que o canal do mini-dimmer está ativado (100% de iluminação). O programa a seguir exemplifica o uso desta macro:



Exemplo de Programa Aplicativo: **Teste_Dimmer_onoff.dxg**

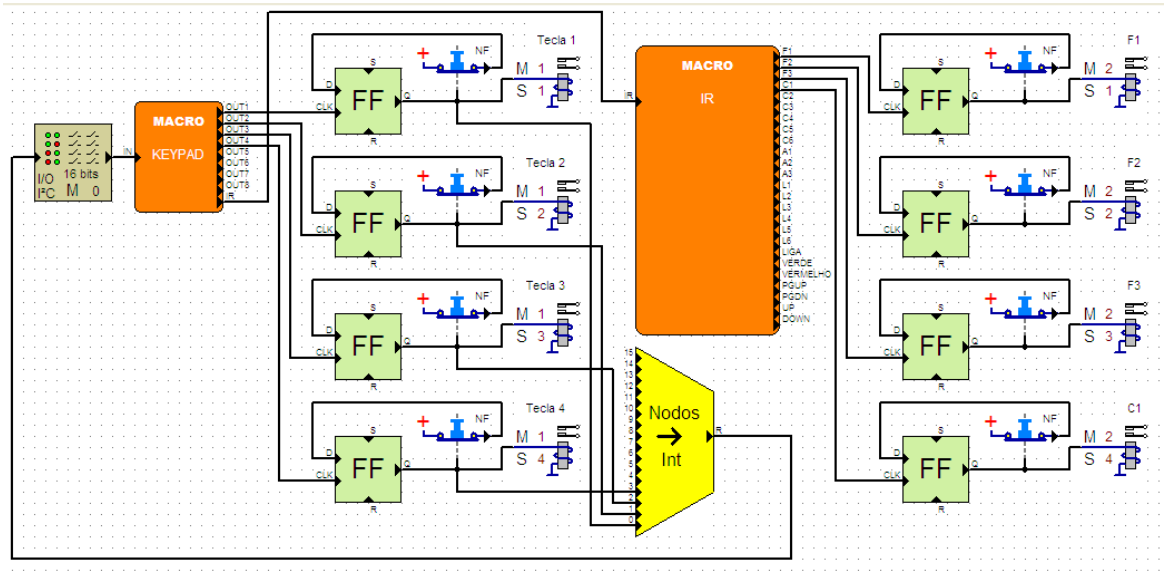
KEYPAD e IR

Estas Macros permitem ler um Keypad instalado na rede I²C do controlador µDX200, e decodificar os comandos recebidos. O aspecto destas Macros é mostrado adiante. Note que as Macros KEYPAD e IR possuem saídas correspondentes aos botões do keypad para µDX200 e do controle remoto, respectivamente. Estas saídas são todas nodos, e são acionados durante o tempo em que o botão está sendo pressionado.



Exemplos de Macros: **Keypad.dxm**, **IR.dxm**

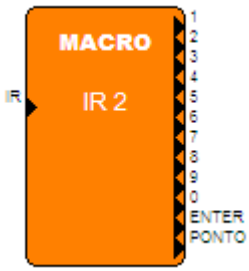
A entrada da macro IR deve ser ligada à saída IR da macro KEYPAD. Já a entrada IN da macro KEYPAD deve ser ligada à saída de um bloco de leitura de Keypad I²C. Abaixo temos um exemplo de uso destas Macros. Maiores detalhes a respeito do Keypad para µDX200 podem ser obtidas no manual do mesmo (ver diretório Manuais no CD do µDX200).



Exemplo de Programa Aplicativo: **Macro - Keypad e IR.dxg**

Note que a macro KEYPAD deve ser excitada por um bloco de I/O (16 bits). No exemplo foi usado o endereço 0 para leitura do keypad. As quatro primeiras teclas momentâneas do keypad acionam as quatro primeiras saídas (S1 a S4) do primeiro módulo de Expansão µDX210. A retenção é feita com flip-flops, e o estado destes é reportado nos leds do keypad. Já o controle remoto aciona as quatro primeiras saídas (S1 a S4) do segundo módulo de Expansão µDX210. Para acionar estas saídas foram usadas as teclas F1, F2, F3 e C1 do controle remoto (note que estas designações correspondem ao policarbonato original fornecido pela Dexter no controle remoto e, portanto, podem variar no caso de policarbonatos específicos). A retenção novamente é feita com flip-flops tipo D com a saída invertida ligada a entrada D (a cada pulso em CLK mudam de estado).

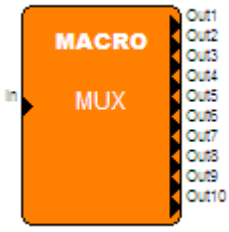
Caso seja necessário detectar as teclas numéricas do controle remoto, a tecla ponto (.) ou a tecla enter (↵) deve-se usar a macro IR_2. Note que ela pode ser usada em conjunto com a macro IR, de forma a decodificar todas as teclas do controle remoto.



Exemplos de Macros: **IR_2.dxm**

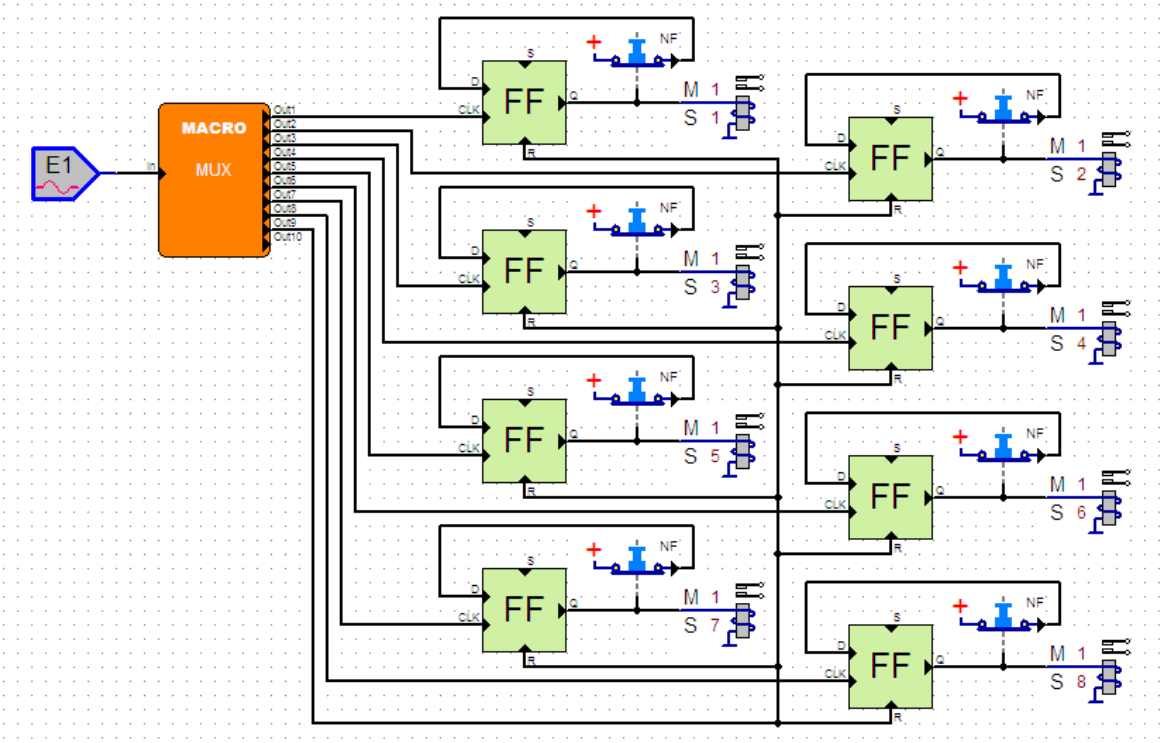
MUX

O MUX permite, de forma econômica, usar cada entrada analógica do controlador para conexão de até 10 botoeiras. Com isso, é possível ler 80 botoeiras (ou pulsadores) em um único µDX200. Maiores informações a respeito do MUX podem ser obtidas em seu manual (ver diretório Manuais no CD do µDX200).



Exemplo de Macro: **MUX.dxm**

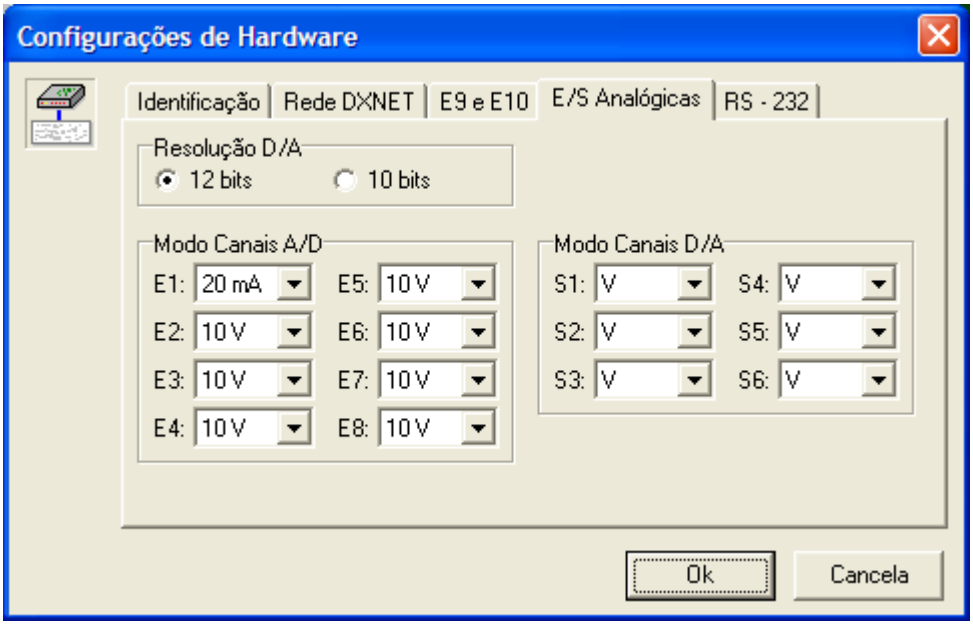
Esta macro consta de uma entrada inteira (a ser ligada a uma das entradas analógicas) e dez saídas binárias (nodos). Conforme o pulsador acionado no MUX a saída correspondente é acionada. Um exemplo é visto a seguir:



Exemplo de Programa Aplicativo: **Macro - MUX.dxg**

As entradas 1 a 8 do MUX ligado à entrada analógica E1 do µDX200 ligam e desligam as saídas S1 a S8 do módulo de Expansão µDX210. A retenção é obtida com os flip-flops. Note que a entrada 9 do MUX permite desacionar todas as saídas.

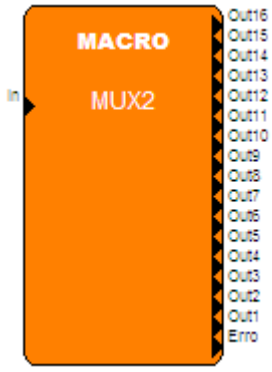
As **Configurações de Hardware** devem ser editadas de forma a prever entrada 0-20mA na entrada analógica E1, assim como os jumpers desta entrada devem ser posicionados para entrada em corrente (ver Seleção de Jumpers).



MUX2

A macro **MUX2** permite ler o novo modelo de Multiplexador da Dexter. Este novo modelo resolve o problema no uso de Multiplexador com botoeiras comuns, feitas para uso em tensão de rede elétrica (127 ou 220Vac) e não baixa tensão. Com o tempo os contatos dessas botoeiras oxidam e passam a apresentar resistências de contato elevadas. Com isso o Multiplexador passa a decodificar a tecla errada. Basta uma limpeza nos contatos para tudo se normalizar, mas é um transtorno nada trivial fazer isso em uma instalação com muitas botoeiras.

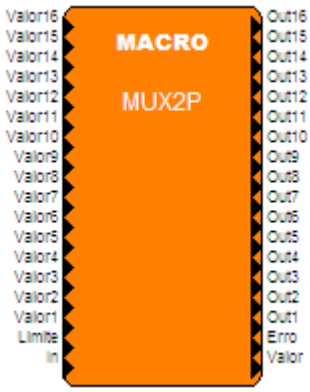
O novo Multiplexador é muito mais tolerante à botoeiras com resistência de contato elevadas (permite até 100 ohms de resistência de contato), virtualmente eliminando o problema sem a necessidade do uso de botoeiras próprias para automação (contatos dourados). Além disso, possui 16 entradas e é conectado ao µDX201 através de apenas 2 fios (o Multiplexador original usava 3 fios). Maiores detalhes podem ser obtidos no manual do novo Multiplexador (ver diretório Manuais no CD do µDX200).



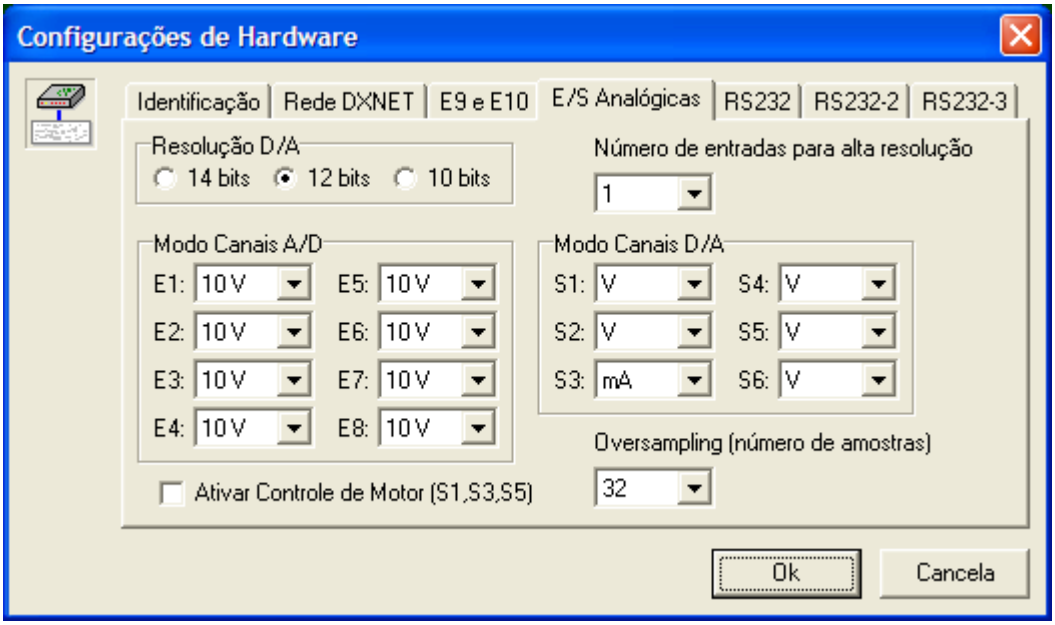
A entrada **In** deve ser ligada a uma entrada analógica na escala de 0-20mA ligada ao novo Multiplexador. As saídas **Out1** a **Out16** são acionadas conforme qual entrada do MUX2 é ativada. A saída **Erro** é ativada caso a corrente na entrada analógica esteja fora dos limites estabelecidos para o MUX2 (de 4 a 20mA).

MUX2P

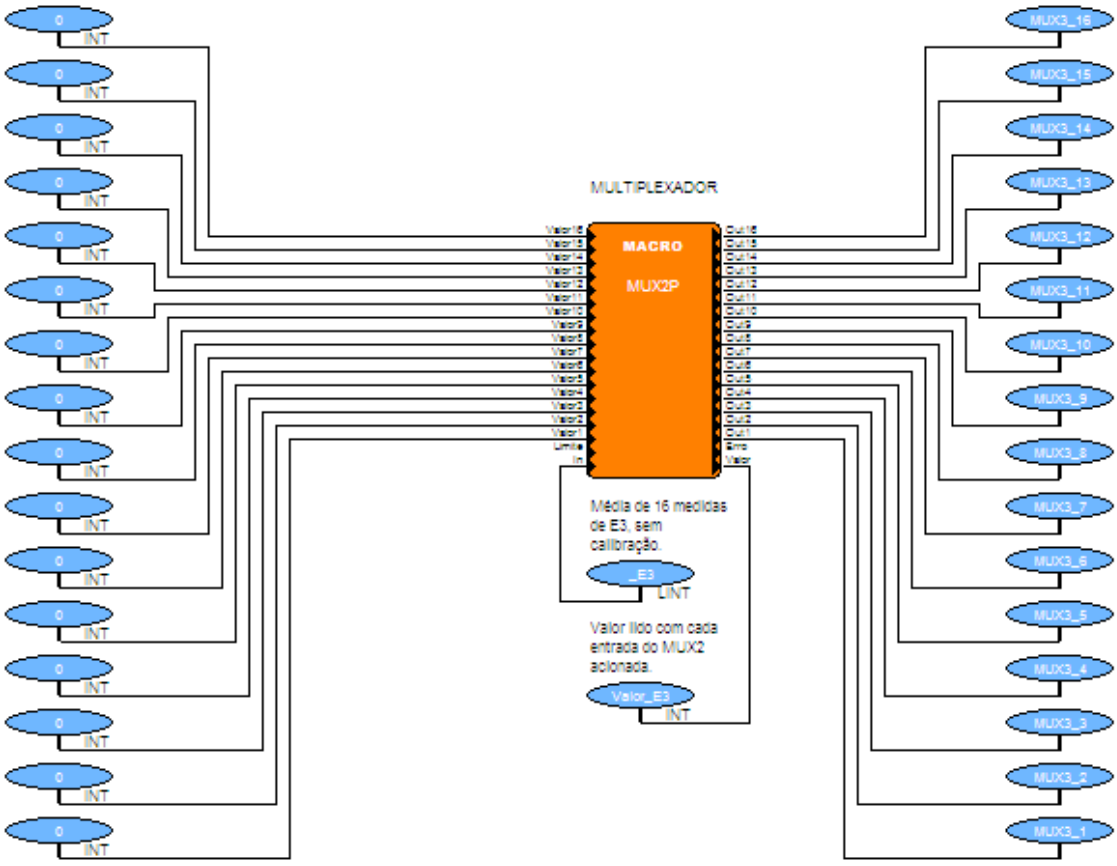
A macro **MUX2P** permite ler o novo modelo de Multiplexador da Dexter, fazendo uma média de 32 medidas do MUX2. Apesar do novo modelo de multiplexador resolver o problema com pulsadores comuns, devido à resistência de contato, ele é sensível ao ripple (60Hz) da rede elétrica induzido nas conexões entre os pulsadores e o MUX2. Esse problema ocorre quando se usa pulsadores distantes do MUX2 (acima de 2 metros). A macro **MUX2P** usa a variável de oversampling para leitura analógica (_E1 a _E8). E permitye programar os pontos de decisão de cada pulsador, e também a variação máxima permitida entre as leituras. É bem mais trabalhosa, pois exige que se feche pulsador a pulsador e se anote o valor médio lido, que depois será inserido na entrada correspondente (Valor 1 a Valor 16). A entrada Limite determina a máxima oscilação na leitura permitida. Esse Limite pode ser de 10 (valor padrão caso não seja especificado) até o valor máximo de 100. A saída Valor permite monitorar o valor lido, de forma a obter os valores a serem inseridos em Valor1 a Valor16.



Para permitir o uso da macro MUX2P é preciso programar a Configuração de Hardware do µDX201 com Número de entradas para alta resolução = 1, e Oversampling = 32. Além disso, a entrada usada deve estar para a escala de mA. Abaixo um exemplo usando a entrada E3:

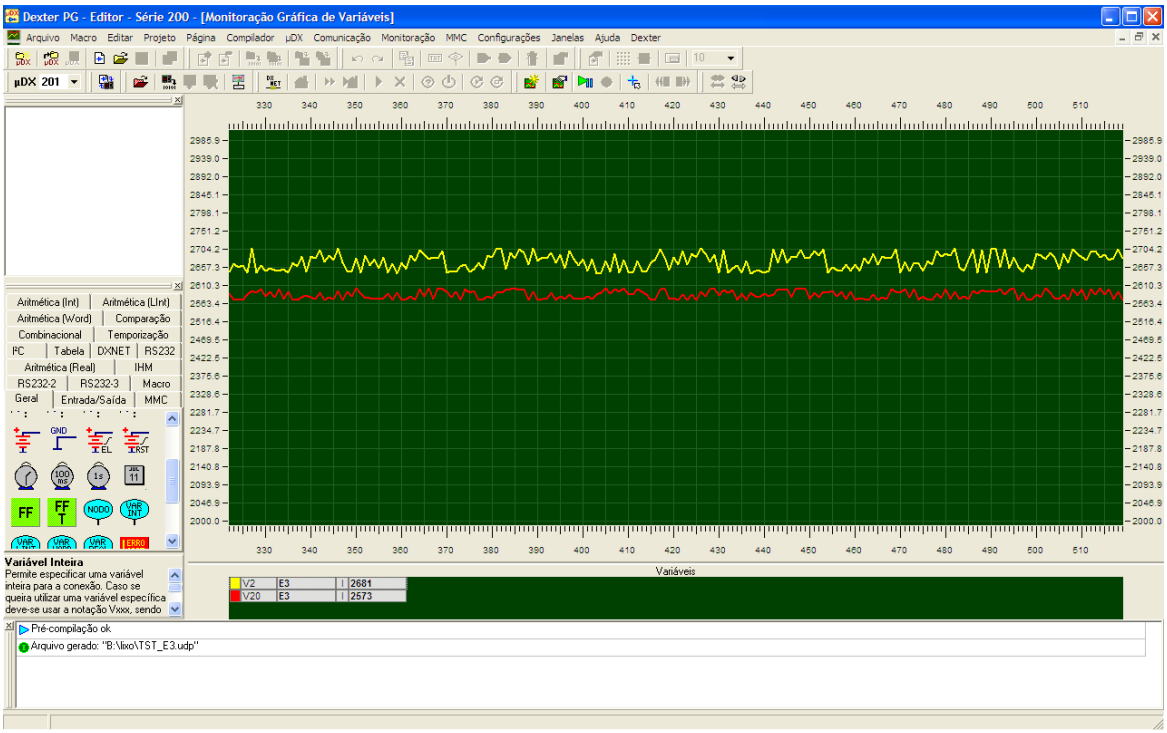


Um possível programa usando essa macro seria:



Note que podemos monitorar a variável Valor_E3, de forma a obtermos o valor médio para cada pulsador pressionado, e também verificar se a variação de medidas não ultrapassa 10 unidades. Se ultrapassar será necessário atribuir um valor maior a Limite (até o máximo de 100).

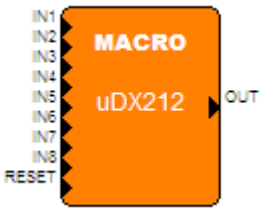
Um exemplo de monitoração da variável Valor_E3 seria o seguinte:



Em amarelo temos a entrada E3 diretamente, e em vermelho temos Valor_E3 (média das últimas 32 medidas). Enquanto E3 varia cerca de 75 divisões em torno de 2668, Valor_E3 varia cerca de 25 divisões em torno de 2586. Portanto, teríamos de atribuir o valor 2586 para a entrada Valor3 da macro, e atribuirmos algo como 35 ou 40 para o limite de variação na leitura.

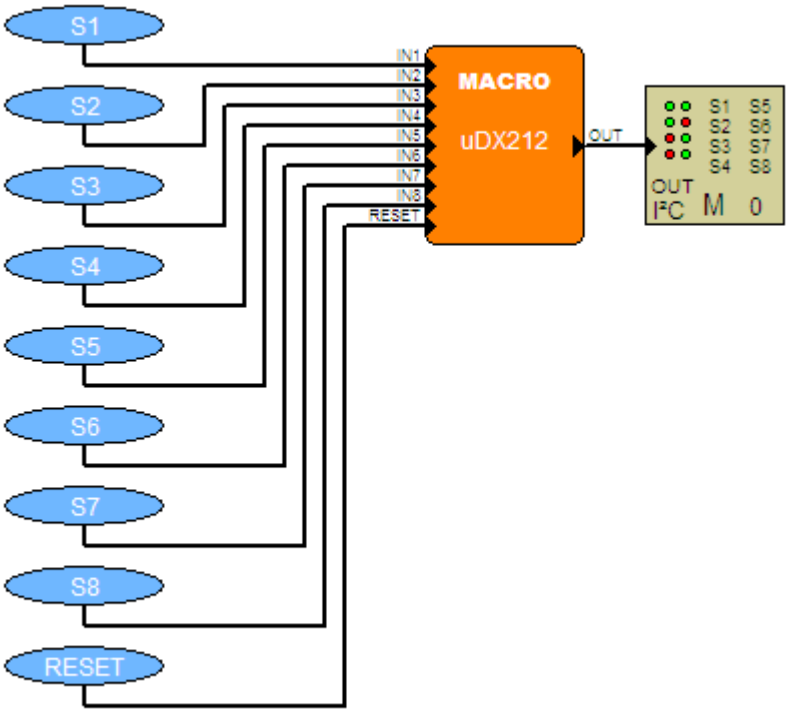
uDX212

A macro **uDX212** possibilita acionar as 8 saídas à relé de um módulo µDX212 via 8 nodos na entrada desta macro-célula. Note que o acionamento possui retenção, ou seja, um pulso no nodo liga a correspondente saída, e outro pulso a desliga. As entradas são todas nodos e a saída é uma variável inteira para comandar um bloco I²C do tipo **Saídas I2C**.



Exemplo de Macro: **uDX212.dxm**

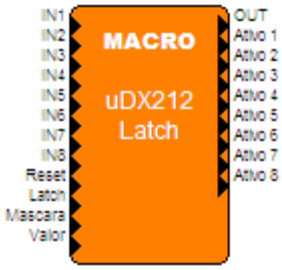
O exemplo a seguir usa oito nodos (designados de S1 a S8) para acionar cada uma das saídas a relé de um módulo µDX212 conectado à rede I²C do µDX200, e ocupando endereço zero. Maiores informações a respeito do µDX212 podem ser obtidas em seu manual (ver diretório Manuais no CD do µDX200). Um nono nodo permite resetar todas as saídas do µDX212.



Exemplo de Programa Aplicativo: **Macro - uDX212.dxg**

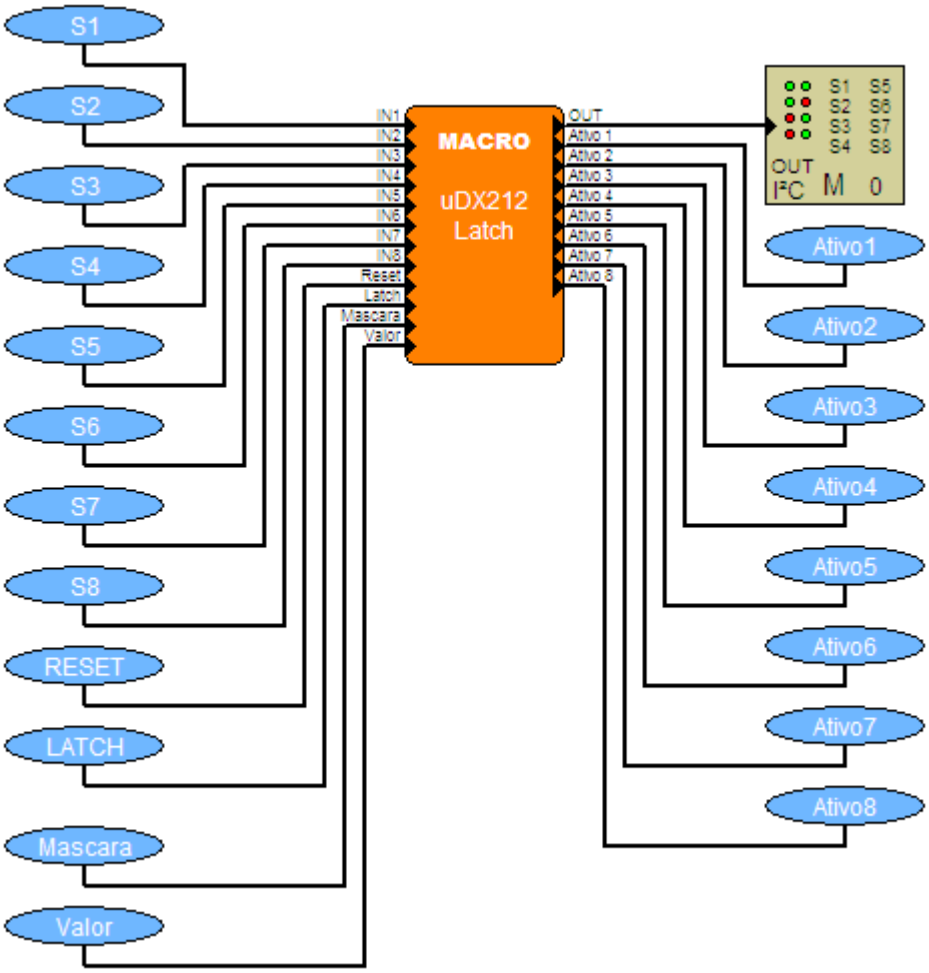
uDX212 Latch

Trata-se de uma versão mais sofisticada da macro-célula descrita anteriormente. Além das 8 entradas para acionamento e desacionamento das saídas e a entrada de reset, esta Macro possui três entradas adicionais: Latch, Mascara e Valor. Estas entradas permitem implementar cenários envolvendo saídas on-off com facilidade. A entrada Latch é uma entrada binária. Já as entradas Mascara e Valor são variáveis inteiras. Sempre que o nodo Latch é energizado, o µDX212 assume na saídas os valores dos bits correspondentes em Valor. Mas apenas as saídas cujos bits correspondentes em Mascara estão ligados. Por exemplo, se Mascara = 0C3h (dois bits mais significativos e dois bits menos significativos ligados), ao energizar Latch apenas estes quatro bits serão afetados pelos bits correspondentes em Valor. Se Valor = 0F1h as saídas S1, S7 e S8 serão acionadas, e a saída S2 será desligada. As demais saídas não serão afetadas.



Exemplo de Macro: **uDX212_latch.dxm**

Além disso, esta Macro possui 8 saídas binárias que indicam o estado de cada saída do µDX212 comandado por ela. Isso facilita implementar a indicação de quais saídas estão acionadas, por exemplo, via leds do Keypad. A seguir temos um exemplo de utilização desta Macro:



Exemplo de Programa Aplicativo: **Macro - uDX212 Latch.dxg**

CENÁRIO

É muito comum o uso de cenários em ambientes residenciais, em que uma série de valores pré-programados para iluminação são ativados simultaneamente. A macro **Cenário** possibilita armazenar até 4 variáveis para uso com Dimmer I^2C. Caso o tempo de acionamento do nodo IN seja maior que o tempo especificado em Click (em múltiplos de 10ms) as variáveis são armazenadas. Se o tempo de acionamento de IN for inferior ao especificado em Click o nodo Ativa é ligado durante um segundo, para permitir que o cenário seja ativado. O nodo Reinicia permite restabelecer o cenário com valores pré-definidos nas entradas inteiras VarInic1 a VarInic4.

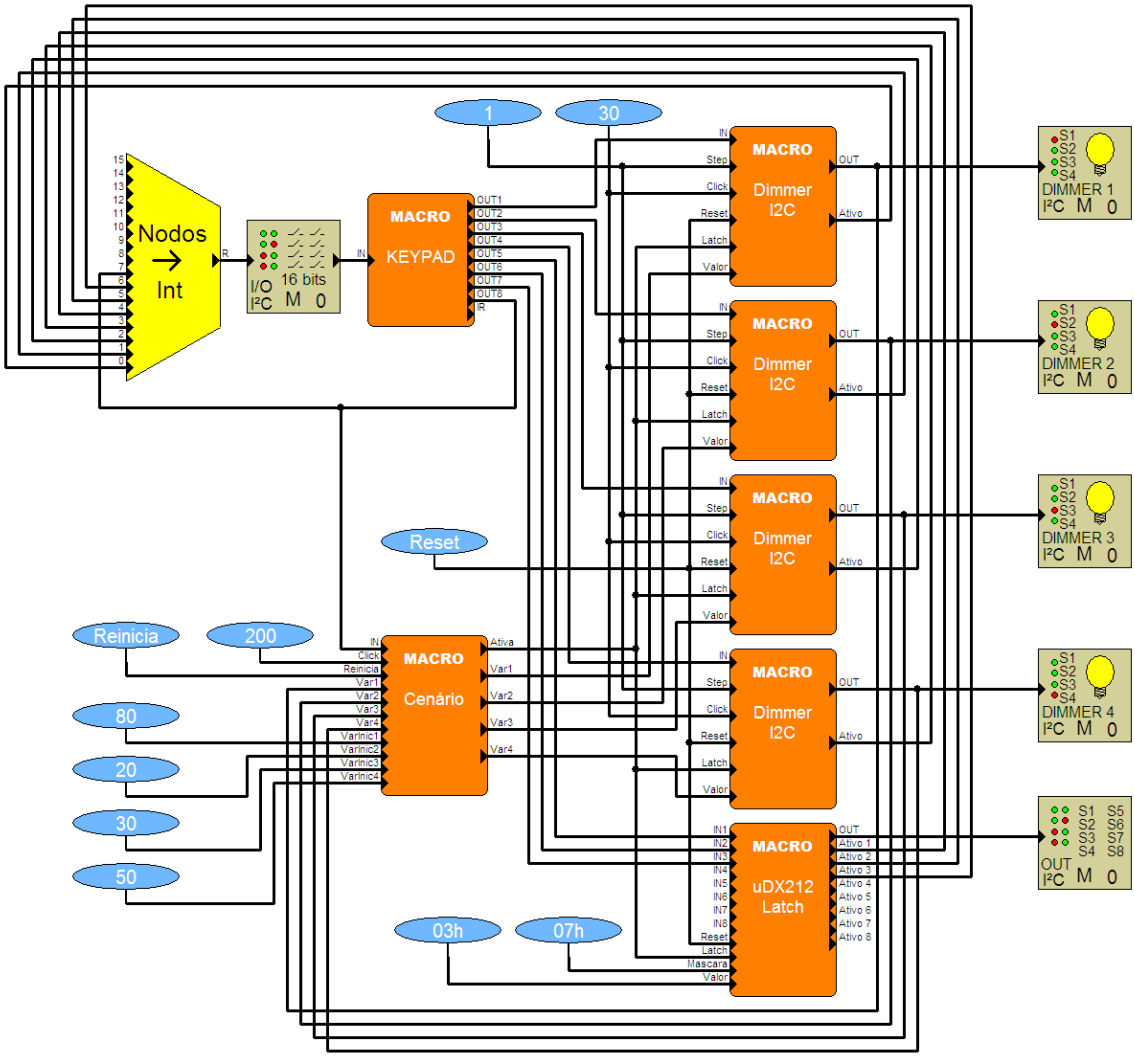
Exemplo de Macro: **Cena.dxm**

O exemplo a seguir utiliza várias macros para gerar um programa de iluminação residencial relativamente complexo. As teclas 1 a 4 do Keypad acionam os dimmers correspondentes. Caso a tecla seja pressionada por menos de 300ms ela alterna o dimmer correspondente entre o último valor de iluminação e desligado. Já se a tecla for mantida pressionada este valor de iluminação pode ser modificado via rampa crescente ou decrescente. As teclas 5 a 7 controlam saídas liga/desliga (on/off) na Expansão µDX212. Note que todas estas saídas, tanto com controle dimmer quanto on/off, indicam seu estado através dos leds do Keypad.

Já a tecla 8 do Keypad, ao ser pressionada rapidamente (menos de 2 segundos), aciona cenário (ou seja, coloca valores pré-programados nas quatro saídas do dimmer I^2C, e também liga ou desliga saídas on/off do µDX212). Já ao manter pressionada a tecla 8 do Keypad por mais de 2 segundos os valores atuais de iluminação dos quatro canais do Dimmer são armazenados como um cenário. É claro que se esta funcionalidade não for adequada, sendo preferível manter os valores dos dimmers em valores constantes para o cenário, basta ligar as entradas Var1 a Var4 da macro Cenário a valores inteiros constantes, em vez de ligá-las às saídas dos blocos Dimmer I2C. Ou então ligá-las as entradas VarInic1 a VarInic4. Estas entradas restabelecem valores padrão para este cenário sempre que o nodo Reinicia é ativado. No exemplo foram usados valores padrão de 80%, 20%, 30% e 50% de iluminação.

A ativação do cenário também aciona ou desaciona saídas on/off do módulo µDX212. No caso, este cenário aciona as saídas S1 e S2 e desaciona a saída S3. As demais saídas permanecem inalteradas. Isso porque foi colocado valor 7 na entrada Máscara da Macro µDX212 Latch. Esta entrada indica quais saídas serão afetadas caso seja acionado o nodo Latch desta Macro. Com o valor 7 (0000 0111 em binário) indica-se que apenas as três saídas menos significativas serão afetadas. Já em Valor é especificado que estado cada uma destas saídas irá assumir ao energizar o nodo Latch. Com o valor 3 (0000 0011 em binário) indica-se que a saída S1 e S2 devem ser ligadas, e a saída S3 deve ser desligada. Foi ligado ao led correspondente à tecla 8 do Keypad o sinal de leitura da tecla 8 do Keypad. Com isso este led será ativado sempre que esta tecla for pressionada.

O nodo de Reset desliga toda iluminação, zerando as saídas das macros Dimmer I2C e µDX212 Latch. Note a diferença em relação ao nodo Reinicia existente na macro Cenário. Ao acionar Reinicia o cenário assume o valor indicado por VarInic1 a VarInic4. Claro que se estas entradas inteiras não forem usadas ao acionar Reinicia o cenário será zerado, já que entradas sem conexão assumem valor zero.



Exemplo de Programa Aplicativo: **Macro - Iluminação.dxg**

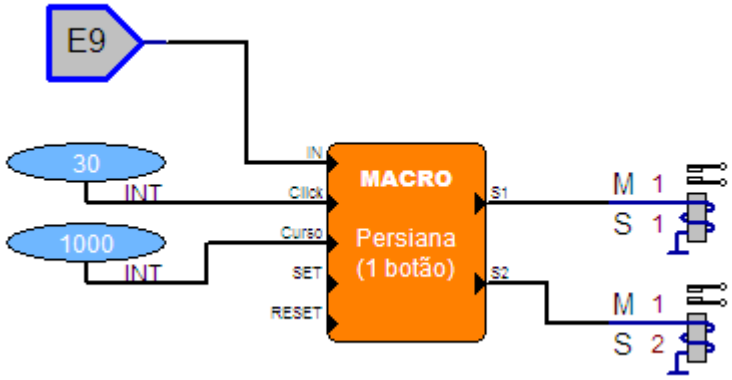
Persiana (1 botão)

Para controle de persianas motorizadas foi criada esta macro. O controle da persiana é feito através de apenas uma entrada, que pode ser uma chave momentânea de um Keypad, ou um pulsador ligado à uma das entradas de um Multiplexador, ou ainda uma entrada digital de um módulo µDX210 ou µDX214.

Exemplo de Macro: **Persiana_1.dxm**

A entrada IN é binária e controla o acionamento da persiana. Com um toque (energização por

tempo inferior ao especificado em Click) ela sobe ou desce até o fim de curso (tempo de acionamento determinado por Curso). O sentido do movimento (subir ou descer) é alternado a cada energização de IN. Caso seja mantida pressionada (energização por tempo superior ao especificado em Click) a persiana sobe ou desce enquanto a entrada IN permanecer energizada, permitindo escolher a posição. As entradas Click e Cursor são inteiras e permitem ajustar o tempo de click e fim de curso (em múltiplos de 10ms). As saídas S1 e S2 são binárias e acionam a persiana para subir ou descer. Por fim, as entradas SET e RESET possibilitam abrir ou fechar compulsoriamente a persiana. Isso é útil no caso dessa fazer parte de um cenário (por exemplo, ao assistir um filme a iluminação decresce e a persiana é fechada). O exemplo abaixo exemplifica o uso desta macro:

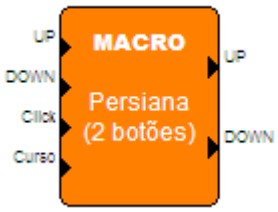


Exemplo de Programa Aplicativo: **Teste - Persiana1.dxg**

Foi usada a entrada E9 do controlador para acionar a persiana. Qualquer pulso menor que 30 x 10ms = 300ms é considerado um toque e abre ou fecha a persiana completamente. Para isso ela é acionada pelo tempo de 1000 x 10ms = 10 segundos, especificado em Curso.

Persiana (2 botões)

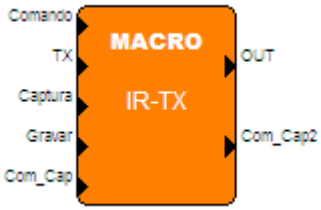
Esta outra macro controla persiana via dois acionamentos, um para subir e outro para descer.Estas entradas podem estar ligadas a saída de um Keypad, ou uma saída de um Multiplexador, ou ainda uma entrada digital de um módulo µDX210 ou µDX214. Fora o detalhe de existirem duas entradas distintas para subir ou descer a persiana, esta macro possui comportamento idêntico à macro anterior, com apenas um botão.



As entradas UP e DOWN são binárias e controlam o acionamento da persiana. Com um toque (energização por tempo inferior ao especificado em Click) ela sobe ou desce até o fim de curso (tempo de acionamento determinado por Curso). O sentido do movimento (subir ou descer) é determinado por qual entrada foi energizada (UP ou DOWN). Caso seja mantida pressionada (energização por tempo superior ao especificado em Click) a persiana sobe ou desce enquanto a entrada permancer energizada, permitindo escolher a posição. As entradas Click e Cursor são inteiras e permitem ajustar o tempo de click e fim de curso (em múltiplos de 10ms). As saídas UP e DOWN são binárias e acionam a persiana para subír ou descer.

IR-TX

A macro **IR-TX** aciona um módulo de transmissão de comandos por infravermelho. É possível tanto transmitir um comando especificado (cada módulo IR-TX permite armazenar até 57 comandos - comando 0 até comando 56) como fazer com que o módulo armazene o comando. No caso de aprendizado de comandos a macro incrementa automaticamente o número do comando a ser armazenado a cada novo comando "aprendido".



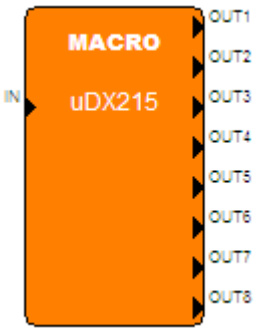
A entrada **Comando** é uma variável inteira que especifica o comando a ser transmitido (de comando 0 a comando 56) pelo módulo IR-TX quando a entrada binária **TX** é energizada. Note que a saída **OUT** deve ser ligada a um bloco I²C de IR TX (transmissor de infravermelho). Para transmitir comandos previamente armazenados no módulo IR TX bastam estes dois sinais - **Comando** e **TX**. Convém manter o pulso em **TX** durante cerca de 300 ms para garantir que o módulo IR TX irá transmitir o comando.

Já no caso de armazenar os comandos no módulo IR-TX para posterior uso (aprendizagem) usa-se as entradas **Captura**, **Gravar** e **Com_Cap**. Para iniciar o processo se atribui a **Com_Cap** (entrada inteira) o valor do primeiro comando a ser capturado (comando 0 a comando 56). A seguir se aciona a entrada binária **Gravar**, indicando para a macro que se inicia um ciclo de gravação de comandos. Por fim, um acionamento momentâneo na entrada binária **Captura** aciona o led de captura do módulo IR TX. Aí é só apontar o controle remoto a ser capturado para o módulo IR TX e pressionar a tecla a ser capturada. Caso o comando seja capturado corretamente o led de captura do módulo IR TX se apaga. Para capturar o próximo comando em ordem crescente basta acionar novamente a entrada **Captura**. A saída **Com_Cap2** indica o número do próximo comando a ser capturado. Convém manter o pulso em **Captura** durante cerca de 300 ms para garantir que o módulo IR TX irá entrar em modo de aprendizado. Para encerrar a captura de comandos basta desenergizar a entrada **Gravar**.

---

uDX215

A macro **uDX215** decodifica as entradas de dois módulos µDX215 (veja exemplo em **I²C - µDX215**):



As saídas OUT1 a OUT4 correspondem as entradas de um µDX215 ligado à rede I²C, e as saídas OUT5 a OUT8 correspondem as entradas do segundo µDX215. Já o acionamento das saídas dos µDX215s pode ser feito usando-se a macro µDX212 Latch. Note que esta macro também pode ser usada com µDX216 (4 saídas com relés de estado sólido).

Tela Projeção (1 botão)

A macro **Tela_Projeção** foi concebida para baixar e levantar uma tela de projeção com apenas um acionamento (1 botão):

A entrada binária **IN** permite subir ou descer a tela alternadamente. As entradas **UP**, **DOWN** e **STOP** fornecem estas funcionalidades individualmente, o que facilita o controle da tela no caso do uso de software supervisório ou aplicativo. O tempo de acionamento é determinado pela entrada inteira **Curso** (em múltiplos de 10ms). As saídas **S1** e **S2** são binárias e acionam a tela de projeção, fazendo-a subir ou descer.

Automação Residencial X

As Macros pertencentes ao grupo **Automação Residencial X** possuem comportamento idêntico às similares existentes no grupo **Automação Residencial**. A única diferença é que essas macros foram otimizadas usando os novos blocos desenvolvidos para µDX201 versão 3.37 ou posterior. Isso possibilitou uma economia de blocos internos razoável, o que permite criar programas com mais funcionalidades. Mas todas as macros deste grupo só funcionam em µDX201 versão 3.37 ou mais recente. A seguir temos uma tabela com o número de blocos de cada macro X e de sua similar para comparação:

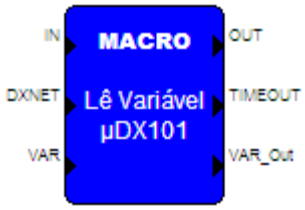
Nome da Macro	Automação Residencial	Automação Residencial X
Dimmer I2C	68 blocos	49 blocos
IR 2	26 blocos	7 blocos
IR	52 blocos	17 blocos
KEYPAD	42 blocos	28 blocos
Persiana (1 botão)	46 blocos	33 blocos
Persiana (2 botões)	49 blocos	36 blocos
µDX212 Latch	29 blocos	21 blocos

Atenção: Macros operacionais apenas em controlador µDX201 versão 3.37 ou posterior.

Comunicação

Lê Variável µDX101

A macro descrita a seguir permite especificar uma variável de µDX101 tipo byte (8 bits) a ser lida em um endereço DXNET determinado, via comunicação serial RS232. Evidentemente, como o µDX101 possui o mesmo protocolo de comunicação do Modem para µDX100, pode-se ler variáveis do controlador µDX100 através do Modem para µDX100 (com protocolo DXNET+).



Exemplo de Macro: **Le_Var_uDX101.dxm**

A macro possui as seguintes entradas:

IN (entrada binária) - permite ativar sua execução. Basta uma borda de subida nesta entrada digital para ativar a macro.

DXNET (entrada inteira) - especifica o endereço DXNET (0 a 15) de onde será lida a variável do µDX101.

VAR (entrada inteira) - variável a ser lida (0 a 255).

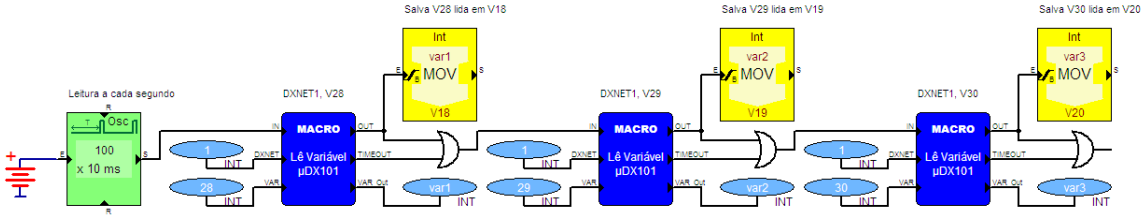
Já as saídas são as seguintes:

OUT (saída binária) - gera um pulso ao ler a variável do µDX101.

TIMEOUT (saída binária) - gera um pulso caso não tenha sido possível ler a variável do µDX101 (timeout de leitura).

VAR_Out (saída inteira) - valor lido da variável de µDX101 (valor de 0 a 255).

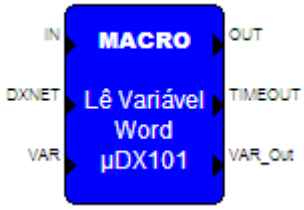
Abaixo temos um exemplo de uso desta macro para leitura de três variáveis do µDX101:



Exemplo de Programa Aplicativo: **Leitura_vars_uDX101.dxg**

Lê Variável Word µDX101

A macro descrita a seguir permite especificar uma variável de µDX101 tipo word (16 bits) a ser lida em um endereço DXNET determinado, via comunicação serial RS232. Evidentemente, como o µDX101 possui o mesmo protocolo de comunicação do Modem para µDX100, pode-se ler variáveis do controlador µDX100 através do Modem para µDX100 (com protocolo DXNET+).



Exemplo de Macro: **Le_Var_Word_uDX101.dxm**

A macro possui as seguintes entradas:

IN (entrada binária) - permite ativar sua execução. Basta uma borda de subida nesta entrada digital para ativar a macro.

DXNET (entrada inteira) - especifica o endereço DXNET (0 a 15) de onde será lida a variável word do µDX101.

VAR (entrada inteira) - variável word a ser lida (0 a 255).

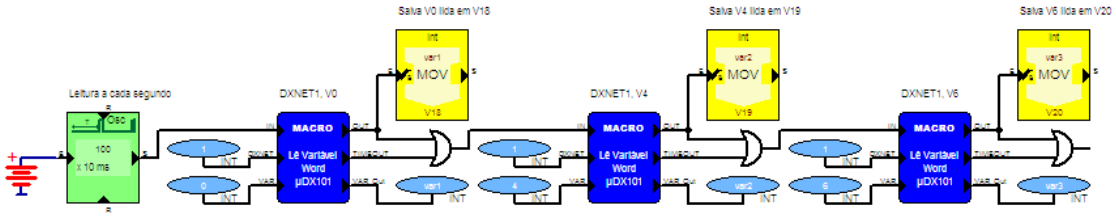
Já as saídas são as seguintes:

OUT (saída binária) - gera um pulso ao ler a variável do µDX101.

TIMEOUT (saída binária) - gera um pulso caso não tenha sido possível ler a variável do µDX101 (timeout de leitura).

VAR_Out (saída inteira) - valor lido da variável de µDX101 (valor de 0 a 255).

Abaixo temos um exemplo de uso desta macro para leitura de três variáveis word do µDX101:

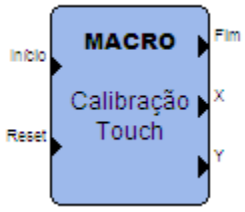


Exemplo de Programa Aplicativo: **Leitura_vars_word_uDX101.dxg**

IHM

Calibração TouchScreen

A Interface Homem/Máquina (IHM µDX220) para controlador µDX201 possui sensor touchscreen na área de display, de forma que é possível desenhar teclas e símbolos sensíveis ao toque do usuário. Mas o sensor sofre variações de unidade para unidade, intrínsecas a própria natureza de seu funcionamento (malha resistiva). Então, para garantir que as coordenadas lidas pelo sensor touchscreen correspondam as coordenadas do display é preciso efetuar a calibração do sensor. As coordenadas do display abrangem valores de 0 a 127 para o eixo horizontal (x), e 0 a 63 para o eixo vertical (y). A macro de calibração pede ao usuário que pressione dois pontos situados em extremos opostos do display e, com base nos valores lidos nestes pontos, calcula os fatores de conversão de forma a calibrar o sensor touchscreen. Na calibração é mandatório utilizar a caneta que acompanha a IHM para pressionar estes pontos, de forma a obter precisão no processo. Uma vez pressionados os dois pontos, é possível pressionar qualquer ponto do display para testar a precisão obtida, ou pressionar as palavras **[OK]** para terminar a calibração ou **[NOK]** para reiniciá-la.



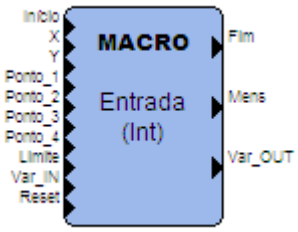
Exemplo de Macro: **Calibra.dxm**

A Macro possui um nodo de entrada (nodo **Início**), que ao ser energizado inicia o processo de calibração. Ao encerrar a calibração é gerado um pulso no nodo de saída da Macro (nodo **Fim**). As conexões **X** e **Y** fornecem valores inteiros correspondentes ao ponto pressionado na display. **X** varia entre 0 e 127, e **Y** de 0 a 63. Caso o display não esteja sendo pressionado tanto **X** quanto **Y** retornam valor zero. A entrada **Reset** permite abortar a operação de calibração do touchscreen da IHM.

Atenção: Macro operacional apenas em controlador µDX201.

Entrada (Int)

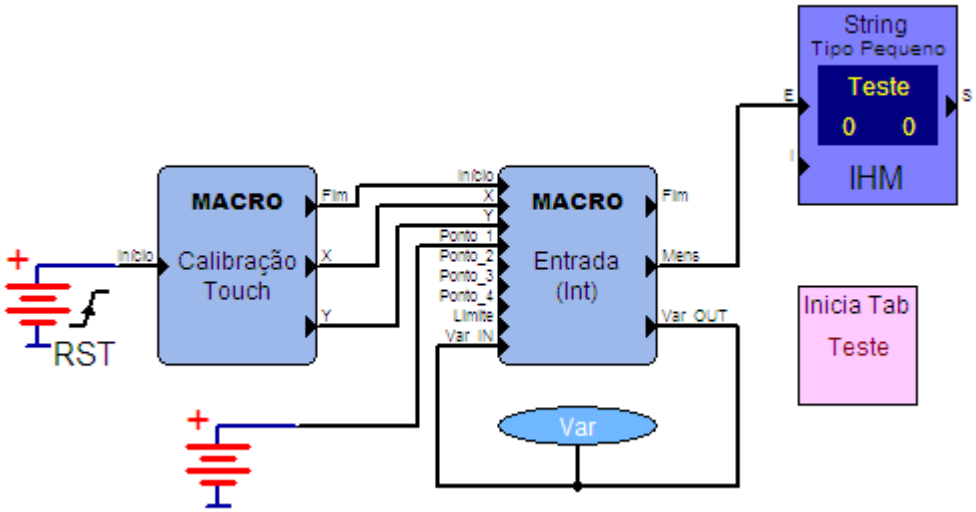
Esta Macro para IHM do controlador µDX201 desenha teclado numérico no display, permitindo entrada de números inteiros em variáveis do controlador. A Macro possui um nodo de entrada (nodo **Início**) e um nodo de saída da Macro (nodo **Fim**). Ao energizar o nodo **Início** o display apresenta o teclado numérico, e ao pressionar a tecla de ↵ (Enter) a Macro encerra, gerando um pulso no nodo **Fim**.



Exemplo de Macro: **Entr_Int.dxm**

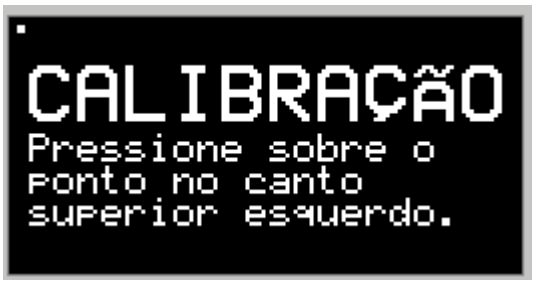
As conexões de entrada **X** e **Y** são variáveis inteiras utilizadas para determinar a posição pressionada no sensor touchscreen. Estes valores, normalmente, são fornecidos pela Macro **Calibração Touch** descrita anteriormente. Também podem ser usadas as palavras reservadas **_X** e **_Y**, que retornam os valores absolutos (sem calibração) do touchscreen. As entradas **Ponto_1** a **Ponto_4** são nodos (entradas binárias) que determinam a posição do ponto decimal. Caso nenhum destes nodos seja energizado não é usado ponto decimal. **Limite** é uma entrada de valor inteiro que determina qual o máximo valor admissível na digitação. Como a variável que irá receber o valor digitado é inteira, este valor máximo deve ser menor ou igual a 32767. **Var_IN** é uma entrada de valor inteiro que determina o valor inicial que irá aparecer no display ao entrar na Macro. Normalmente se utiliza a própria variável que irá receber o valor digitado. O nodo **Mens** gera um pulso para ativar um bloco de escrita de string, de forma a se poder escrever no display um nome para a variável a ser digitada. Devido a área utilizada para a digitação, é possível escrever apenas 5 caracteres grandes a partir da posição (0,0) do display, ou 10 caracteres pequenos a partir da mesma posição. **Var_OUT** é uma saída de variável inteira com o valor digitado. A entrada **Reset** permite abortar a operação de entrada de dados.

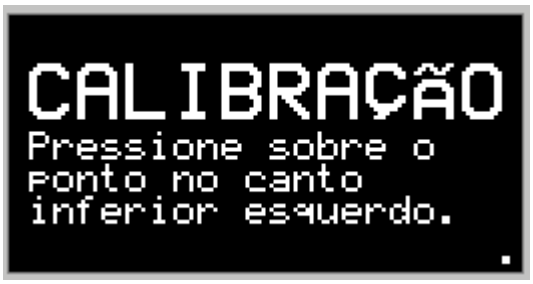
O exemplo a seguir efetua uma calibração do sensor touchscreen, e a seguir aguarda a digitação da variável Var com uma casa decimal:

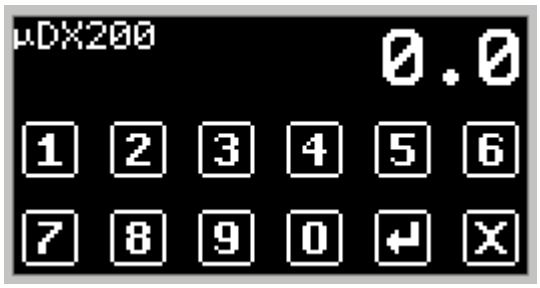


Exemplo de Programa Aplicativo: **Macro - IHM.dxg**

Atenção: Macro operacional apenas em controlador µDX201.

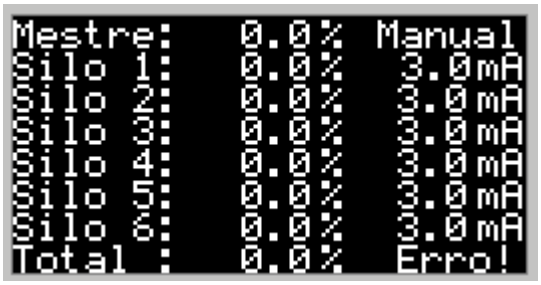


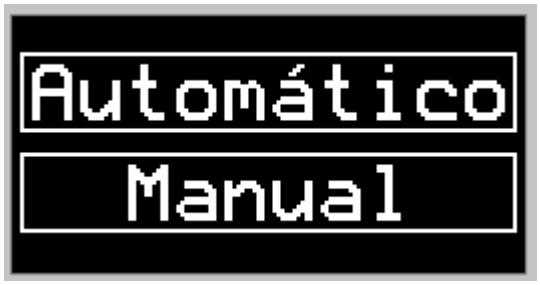


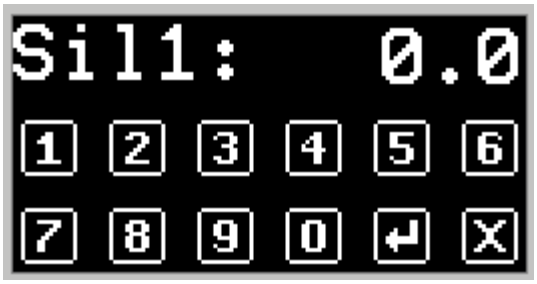


Telas do programa: **Macro - IHM.dxg**

Para um exemplo mais completo de uso de Macros para IHM veja o projeto **IHM - Controle Silos.dxp**. Este projeto utiliza várias páginas de programação para permitir visualizar no display da IHM seis percentuais de produção (um para cada silo), sendo que a soma dos seis deve ser sempre 100,0%. Além disso, é monitorada a corrente na entrada analógica E1 para determinar a produção total de 0,0% (correspondendo a 4mA) até 100,0% (correspondendo a 20mA). Conforme o percentual do silo e da produção total requerida é fixada uma corrente de saída em cada uma das saídas analógicas. Também é possível fixar a produção total manualmente. Para isso o programa permite comutar entre operação Manual ou Automática. Para comutar o display para entrada de dados basta pressionar qualquer ponto do mesmo quando o programa estiver na tela principal.

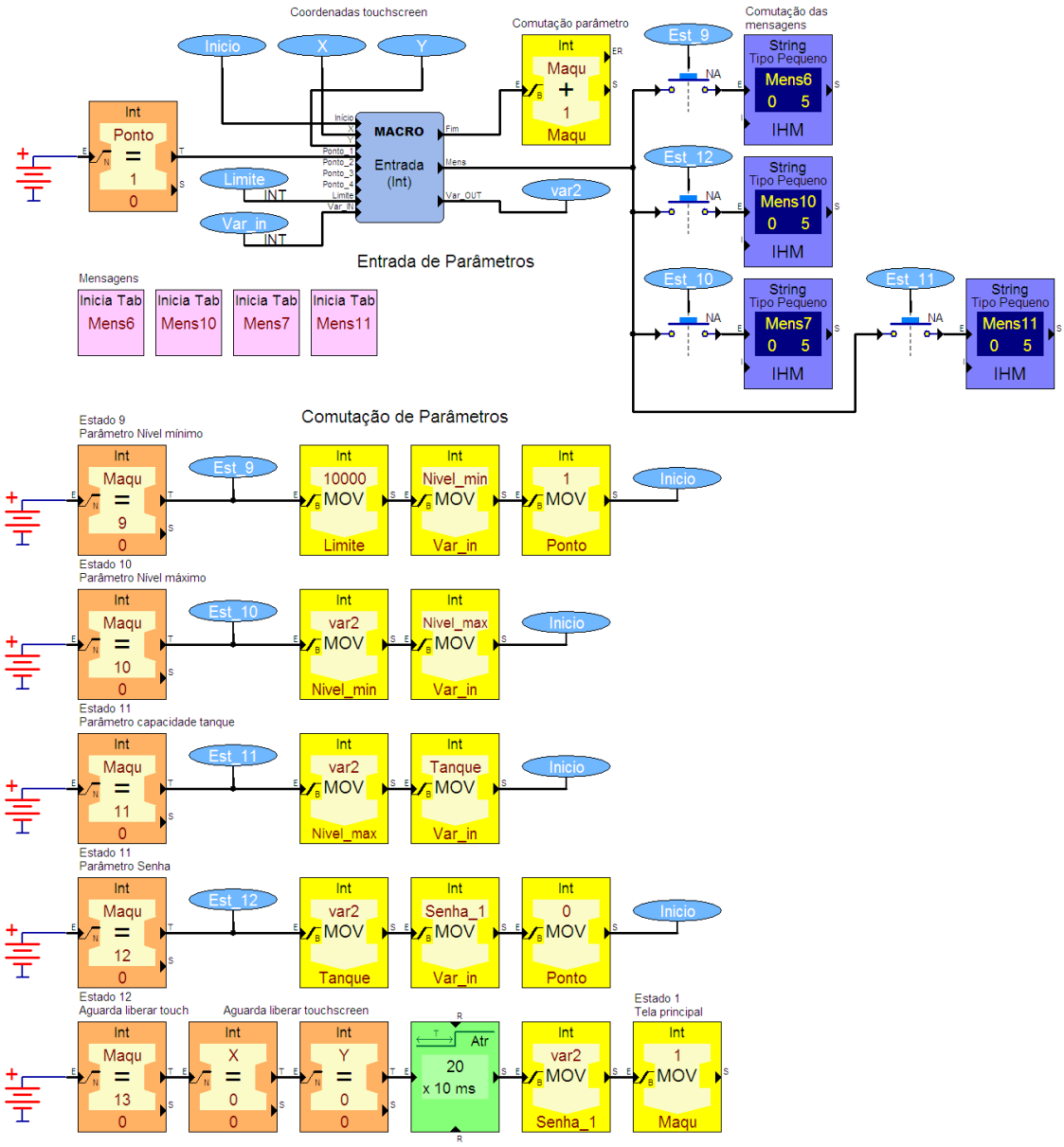






Telas do projeto: **IHM - Controle Silos.dxp**

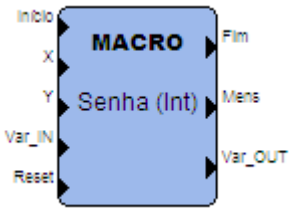
A macro **Entrada (Int)** consome um número razoável de blocos do controlador µDX201. Assim, no caso de entrada de múltiplos parâmetros é aconselhável o uso da mesma macro, apenas variando as entradas e saídas da mesma. O programa a seguir exemplifica isso via máquina de estados. A partir do estado 9 são inseridos os parâmetros nível mínimo, nível máximo, tanque e senha, sendo que os três primeiros parâmetros com uma casa decimal. Convém estudar este exemplo no caso da necessidade de inserção de múltiplos parâmetros via IHM do µDX201.



Exemplo de Programa Aplicativo: **Macro - Entrada Parâmetros.dxg**

Senha (Int)

Esta Macro para IHM do controlador µDX201 desenha teclado numérico no display, permitindo entrada de quatro dígitos de senha. Esta macro é muito similar a anterior - **Entrada (Int)** - mas, em vez de apresentar o valor digitado preenche os dígitos com o caracter asterisco (*). A Macro possui um nodo de entrada (nodo **Início**) e um nodo de saída da Macro (nodo **Fim**). Ao energizar o nodo **Início** o display apresenta o teclado numérico, e ao pressionar a tecla de ↵ (Enter) a Macro encerra, gerando um pulso no nodo **Fim**.



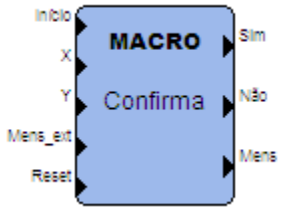
Exemplo de Macro: **Senha.dxm**

As conexões de entrada **X** e **Y** são variáveis inteiras utilizadas para determinar a posição pressionada no sensor touchscreen. Estes valores, normalmente, são fornecidos pela Macro **Calibração Touch** descrita anteriormente. Também podem ser usadas as palavras reservadas **_X** e **_Y**, que retornam os valores absolutos (sem calibração) do touchscreen. **Var_IN** é uma entrada de valor inteiro que determina o valor inicial que irá aparecer no display ao entrar na Macro. Normalmente se mantém esta entrada sem conexão, fazendo com que a senha inicial seja nula. O nodo **Mens** gera um pulso para ativar um bloco de escrita de string, de forma a se poder escrever no display um nome para a senha a ser digitada. Devido a área utilizada para a digitação, é possível escrever apenas 5 caracteres grandes a partir da posição (0,0) do display, ou 10 caracteres pequenos a partir da mesma posição. **Var_OUT** é uma saída de variável inteira com o valor digitado. A entrada **Reset** permite abortar a operação de entrada de senha.

Atenção: Macro operacional apenas em controlador µDX201.

Confirma

Esta Macro para IHM do controlador µDX201 desenha duas teclas no display, permitindo seleção entre duas opções (SIM e NÃO). Caso a entrada digital **Mens_ext** esteja em zero ao habilitar esta Macro (pulso na entrada digital **Inicio**) é impressa a mensagem "Confirma?" acima das teclas de sim e não. Já se a entrada digital **Mens_ext** estiver em um a Macro não imprime mensagem e gera um pulso na saída digital **Mens**, permitindo acionar uma mensagem do usuário. A Macro possui um nodo de entrada (nodo **Início**) e dois nodos de saída da Macro (nodos **Sim** e **Não**). Ao energizar o nodo **Início** o display apresenta as teclas SIM e NÃO, e ao pressionar uma delas a Macro encerra, gerando um pulso na saída correspondente.



Exemplo de Macro: **Confirma.dxm**

As conexões de entrada **X** e **Y** são variáveis inteiras utilizadas para determinar a posição pressionada no sensor touchscreen. Estes valores, normalmente, são fornecidos pela Macro **Calibração Touch** descrita anteriormente. Também podem ser usadas as palavras reservadas **_X** e **_Y**, que retornam os valores absolutos (sem calibração) do touchscreen. A entrada digital **Reset** permite abortar a operação da macro.

Atenção: Macro operacional apenas em controlador µDX201.

LED, LED Grande, LED Quadrado, LED Quadrado Grande

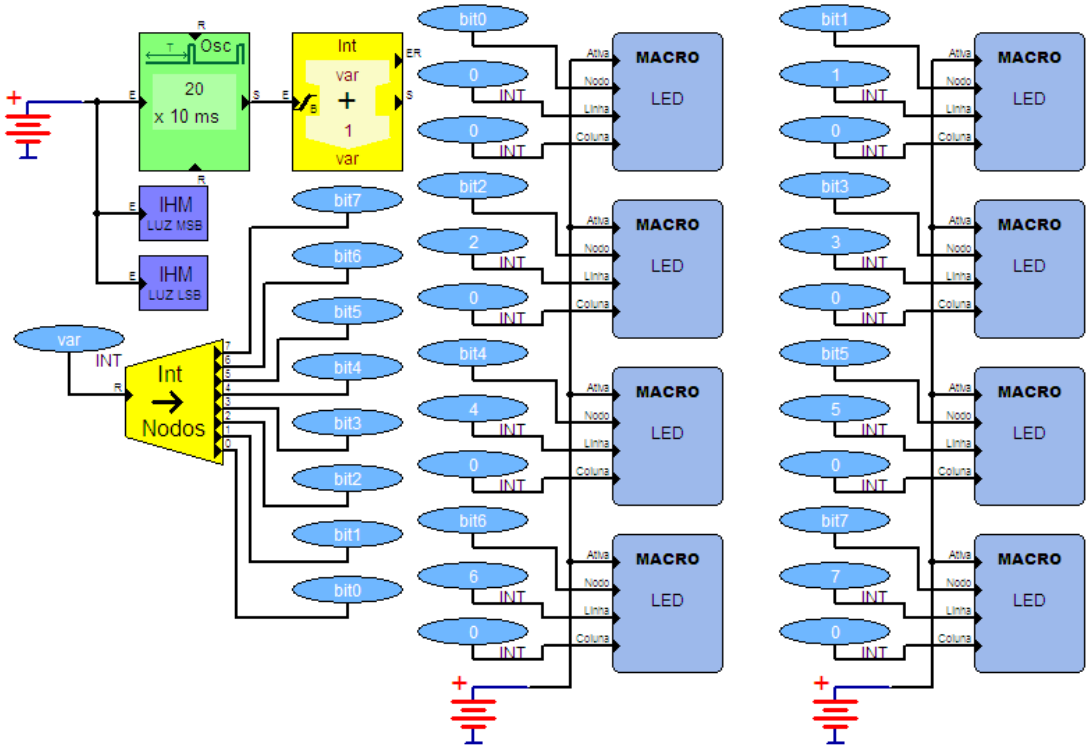
Estas Macros para IHM do controlador µDX201 desenham LEDs no display, representando o estado do nodo de entrada.

Exemplos de Macro: **LED.dxm** **LED_Grande.dxm** **LED_Quadr.dxm** **LED_Quadr_Grande.dxm**

As conexões de entrada **Linha** e **Coluna** são variáveis inteiras utilizadas para determinar a posição a ser colocado o LED no display. No caso de **LED** e **LED Quadrado** é desenhado um LED pequeno, e é permitido **Linha** de 0 a 7, e **Coluna** de 0 a 20. Já no caso de **LED Grande** e **LED Quadrado Grande** a representação do LED é bem maior, e é permitido **Linha** de 0 a 3, e **Coluna** de 0 a 9. Note que essas Macros repartem o display em linhas e colunas fixas, de forma a facilitar o posicionamento na tela da IHM. As Macros **Texto** e **Variável Inteira** utilizam as mesmas posições. Cada posição, no caso de representação normal (**LED** ou **LED Quadrado**) ocupa 8 linhas x 6 colunas. Já no caso de representação grande (**LED Grande** ou **LED Quadrado Grande**) cada posição ocupa 16 linhas x 12 colunas. A conexão **Ativa** aciona a Macro. A seguir temos uma representação das posições no display da IHM (128 x 64 pixels):

O exemplo a seguir exemplifica o uso da macro LED para representar o estado dos primeiros 8

bits de uma variável inteira, que é incrementada a cada 200ms:



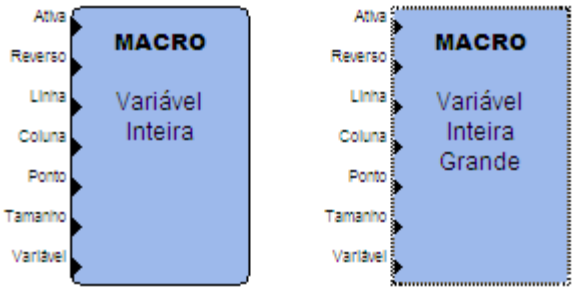
Exemplo de Programa Aplicativo: **Macro - LED.dxg**

O resultado no display da IHM é o seguinte:

Atenção: Macro operacional apenas em controlador µDX201.

Variável Inteira, Variável Inteira Grande

Estas Macros para IHM do controlador µDX201 escrevem o valor de uma variável inteira no display.

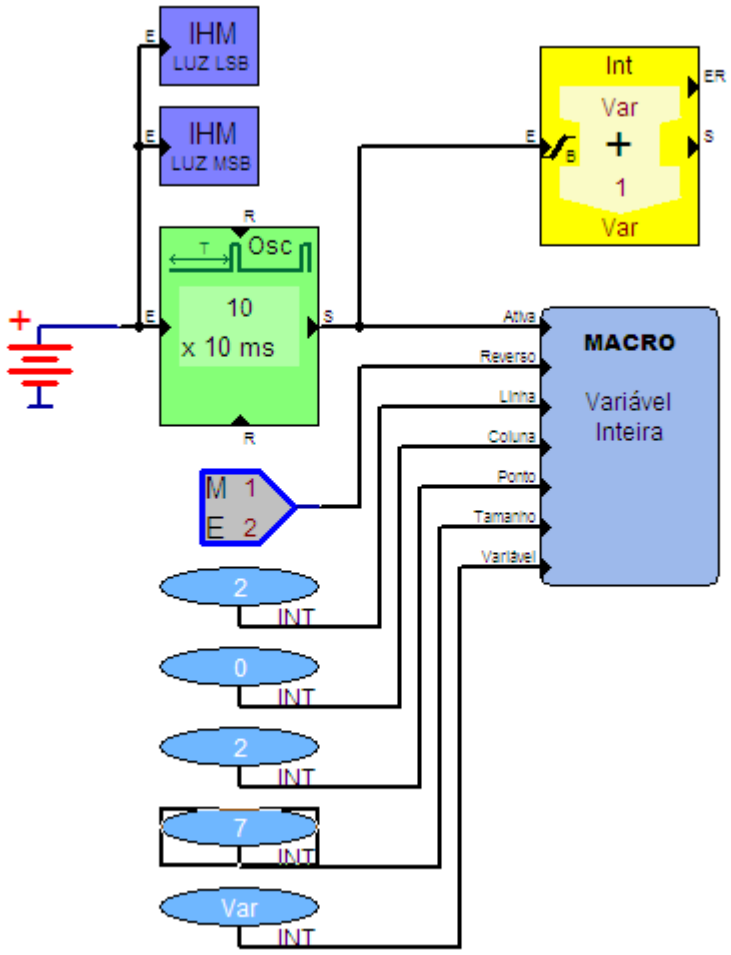


Exemplos de Macro: **Var_Int.dxm** **Var_Int_Grande.dxm**

As conexões de entrada **Linha** e **Coluna** são variáveis inteiras utilizadas para determinar a posição a ser colocado o valor da variável no display. No caso de **Variável Inteira** são usados dígitos pequenos, e é permitido **Linha** de 0 a 7, e **Coluna** de 0 a 20. Já no caso de **Variável Inteira Grande** a representação dos dígitos é bem maior, e é permitido **Linha** de 0 a 3, e **Coluna** de 0 a 9. Note que essas Macros repartem o display em linhas e colunas fixas, de forma a facilitar o posicionamento na tela da IHM. As Macros **Texto** e **LED** utilizam as mesmas posições. Cada posição, no caso de representação normal (**Variável Inteira**) ocupa 8 linhas x 6 colunas. Já no caso de representação grande (**Variável Inteira Grande**) cada posição ocupa 16 linhas x 12 colunas. A seguir temos uma representação das posições no display da IHM (128 x 64 pixels):

O exemplo a seguir exemplifica o uso da macro **Variável Inteira** para mostrar o valor da variável

Var, que é incrementada a cada 100ms:



Exemplo de Programa Aplicativo: **Inteira.dxg**

O resultado no display da IHM é o seguinte:

Atenção: Macro operacional apenas em controlador µDX201.

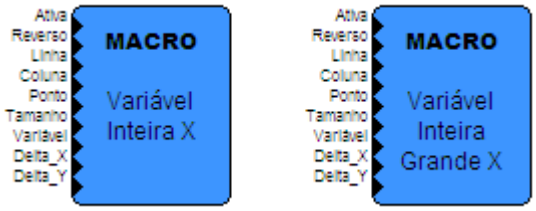
IHM X

As Macros do grupo **IHM X** possuem a mesma funcionalidade de suas homólogas no grupo **IHM**, quando houverem, mas consomem muito menos blocos de programação. Mas necessitam de controlador µDX201 versão 3.53 ou superior para funcionarem. Abaixo a comparação entre as Macros nos dois grupos:

Variável Inteira & Variável Inteira Grande	**103 blocos**	**Variável Inteira X & Variável Inteira Grande X**	**35 blocos**

Variável Inteira X, Variável Inteira Grande X

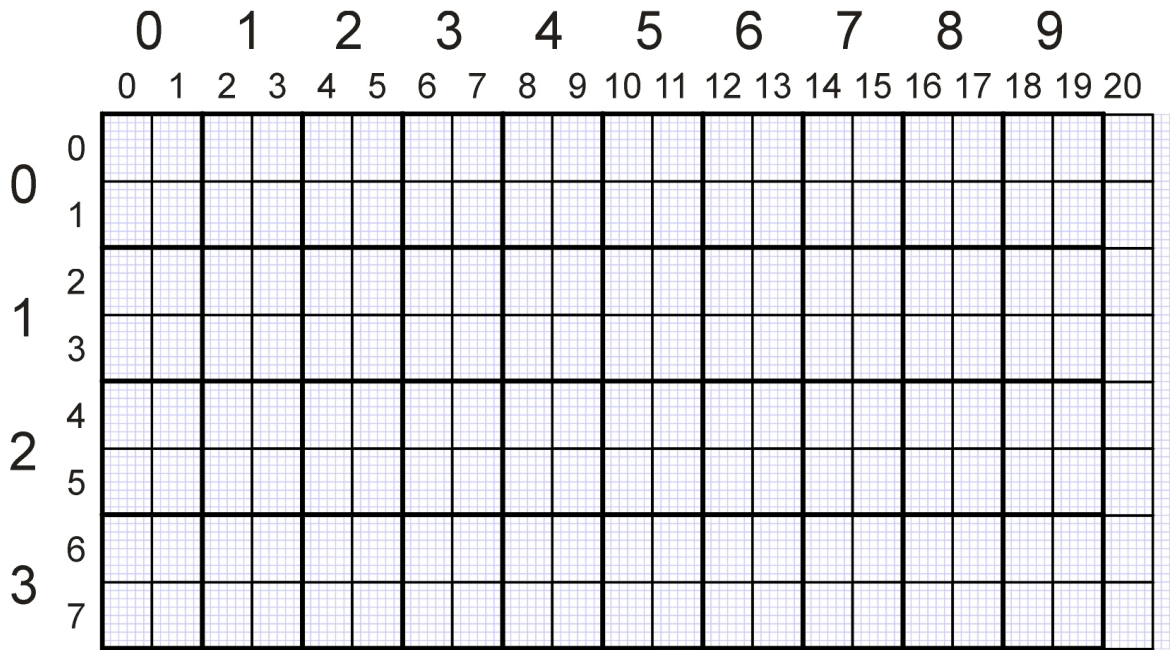
Estas Macros para IHM do controlador µDX201 escrevem o valor de uma variável inteira no display.



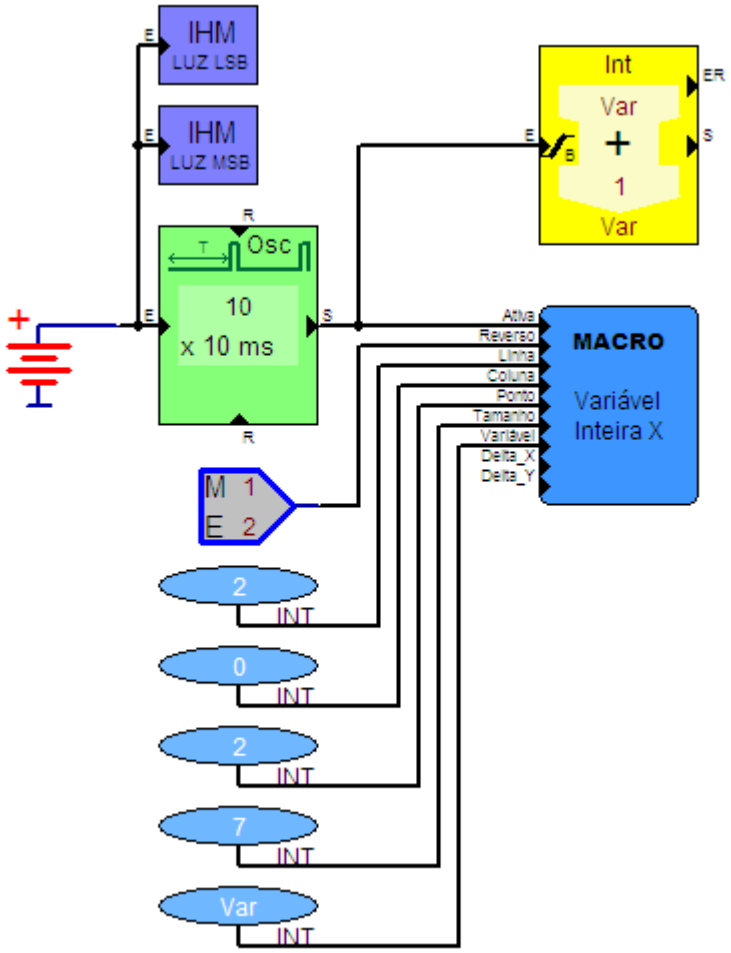
Exemplos de Macro: **Var_IntX.dxm** **Var_Int_GrandeX.dxm**

As conexões de entrada **Linha** e **Coluna** são variáveis inteiras utilizadas para determinar a posição a ser colocado o valor da variável no display. No caso de **Variável Inteira** são usados dígitos pequenos, e é permitido **Linha** de 0 a 7, e **Coluna** de 0 a 20. Já no caso de **Variável Inteira Grande** a representação dos dígitos é bem maior, e é permitido **Linha** de 0 a 3, e **Coluna** de 0 a 9. Note que essas Macros repartem o display em linhas e colunas fixas, de forma a facilitar o posicionamento na tela da IHM. As Macros **Texto** e **LED** utilizam as mesmas posições. Cada posição, no caso de representação normal (**Variável Inteira**) ocupa 8 linhas x 6 colunas. Já no caso de representação grande (**Variável Inteira Grande**) cada posição ocupa 16 linhas x 12 colunas. As conexões **Delta_X** e **Delta_Y** permitem especificar um pequeno deslocamento no eixo X e Y, de forma a possibilitar centralizar a impressão. A seguir temos uma representação das posições no display da IHM (128 x 64 pixels):

POSIÇÕES NO DISPLAY DE IHM PARA µDX201



O exemplo a seguir exemplifica o uso da macro **Variável Inteira** para mostrar o valor da variável **Var**, que é incrementada a cada 100ms:



Exemplo de Programa Aplicativo: **InteiraX.dxg**

O resultado no display da IHM é o seguinte:

Atenção: Macro operacional apenas em controlador µDX201 versão 3.53 ou superior.

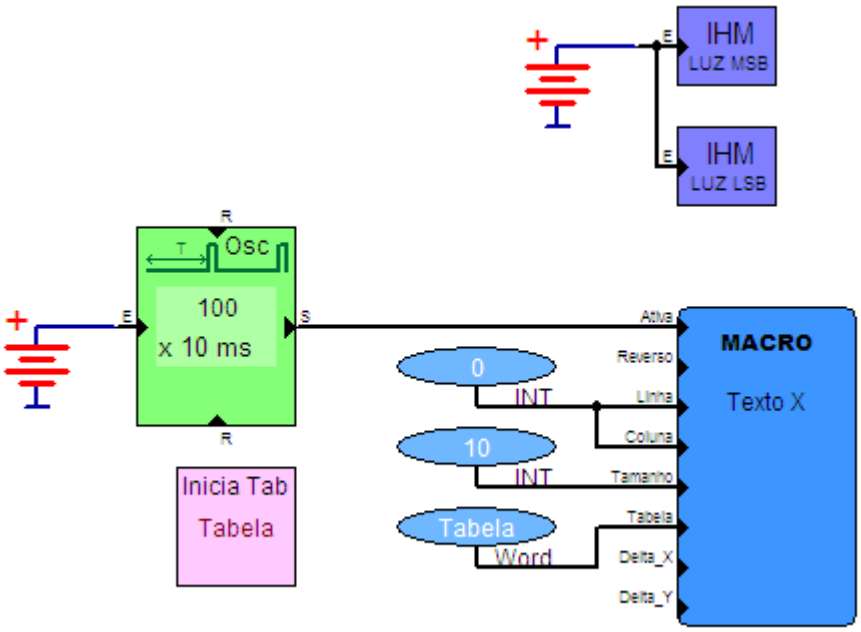
Texto X, Texto Grande X

Estas Macros para IHM do controlador µDX201 permitem escrever um string no display, definido em uma tabela no programa aplicativo.

Exemplos de Macro: **TextoX.dxm** **Texto_GrandeX.dxm**

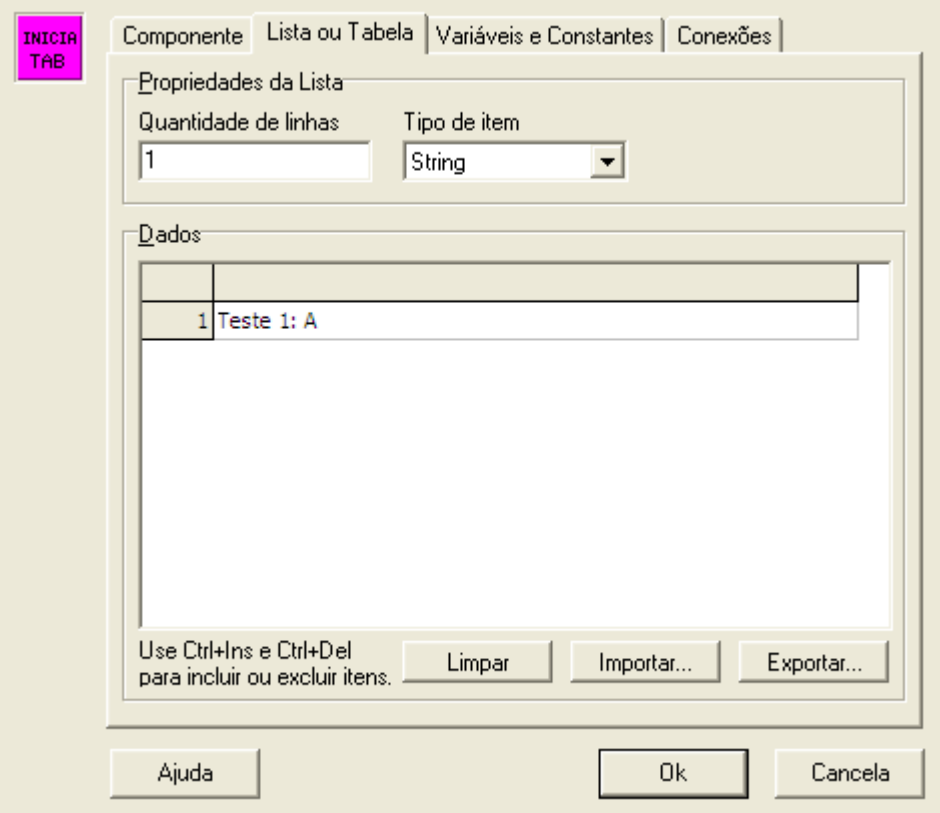
As conexões de entrada **Linha** e **Coluna** são variáveis inteiras utilizadas para determinar a posição a ser colocado o valor da variável no display. No caso de **Texto** são usados caracteres pequenos, e é permitido **Linha** de 0 a 7, e **Coluna** de 0 a 20. Já no caso de **Texto Grande** a representação dos caracteres é bem maior, e é permitido **Linha** de 0 a 3, e **Coluna** de 0 a 9. Note que essas Macros repartem o display em linhas e colunas fixas, de forma a facilitar o posicionamento na tela da IHM. As Macros **Variável Inteira** e **LED** utilizam as mesmas posições. Cada posição, no caso de representação normal (**Texto**) ocupa 8 linhas x 6 colunas. Já no caso de representação grande (**Texto Grande**) cada posição ocupa 16 linhas x 12 colunas. As conexões **Delta_X** e **Delta_Y** permitem especificar um pequeno deslocamento no eixo X e Y, de forma a possibilitar centralizar a impressão. A seguir temos uma representação das posições no display da IHM (128 x 64 pixels):

O exemplo a seguir exemplifica o uso desta Macro:



Exemplo de Programa Aplicativo: **Macro - Texto.dxg**

O conteúdo da tabela é o seguinte, nesse exemplo:

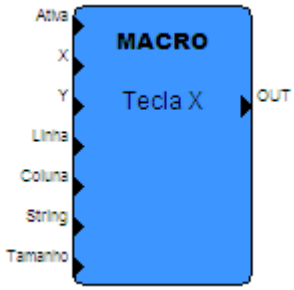


O resultado no display da IHM é mostrado a seguir:

Atenção: Macro operacional apenas em controlador µDX201 versão 3.53 ou superior.

Tecla X

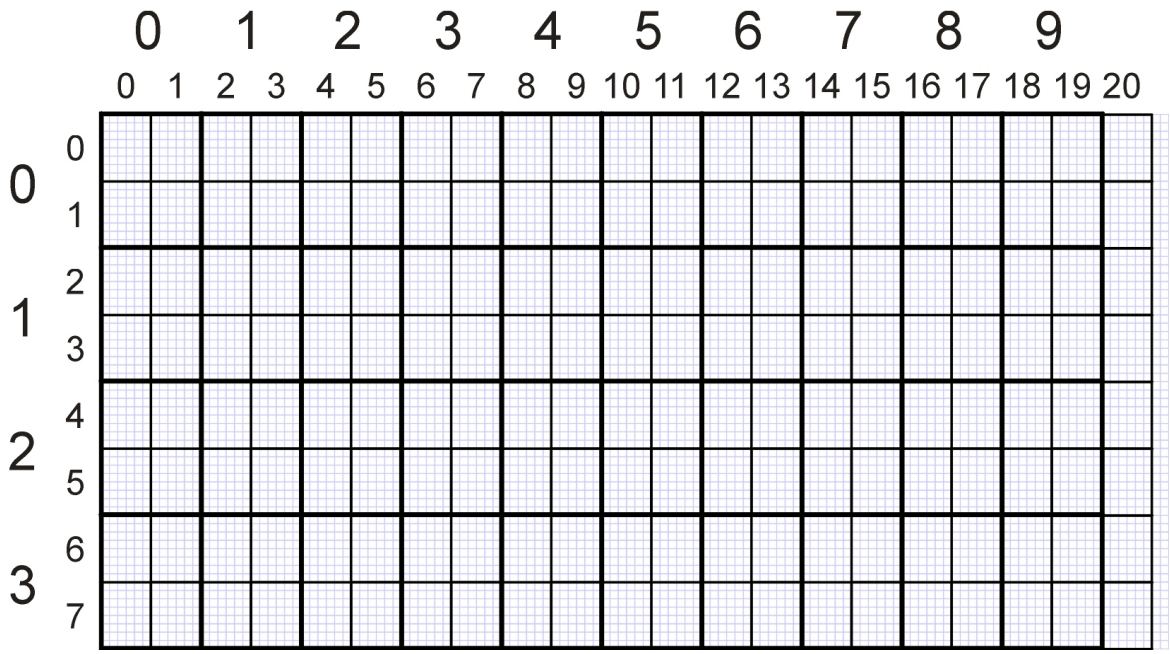
Esta Macro para IHM do controlador µDX201 permite desenhar uma tecla no display, com uma inscrição interna de até 3 caracteres definida em uma tabela no programa aplicativo.



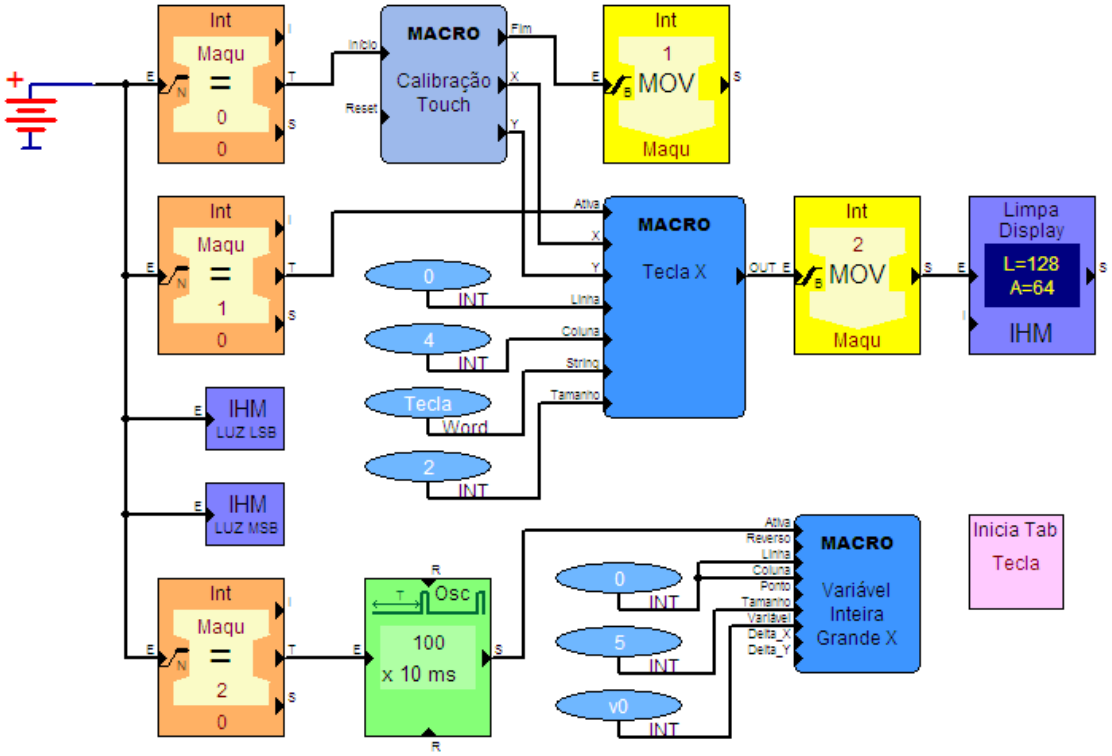
Exemplos de Macro: **TeclaX.dxm**

As conexões de entrada **Linha** e **Coluna** são variáveis inteiras utilizadas para determinar a posição a ser colocado o valor da variável no display. No caso de **Tecla** é permitido **Linha** de 0 a 3, e **Coluna** de 0 a 4. Note que essa Macro repartem o display em linhas e colunas fixas, de forma a facilitar o posicionamento na tela da IHM. As Macros **Variável Inteira** e **LED** utilizam as mesmas posições. Cada tecla ocupa 16 linhas x 24 colunas. As conexões de entrada **X** e **Y** são variáveis inteiras utilizadas para determinar a posição pressionada no sensor touchscreen. Estes valores, normalmente, são fornecidos pela Macro **Calibração Touch** descrita anteriormente. Também podem ser usadas as palavras reservadas **_X** e **_Y**, que retornam os valores absolutos (sem calibração) do touchscreen. A conexão **String** deve apontar para uma tabela string com a inscrição a ser colocada na tecla, e a conexão **Tamanho** indica o número de acracteres (0 a 3). A seguir temos uma representação das posições no display da IHM (128 x 64 pixels):

POSIÇÕES NO DISPLAY DE IHM PARA µDX201

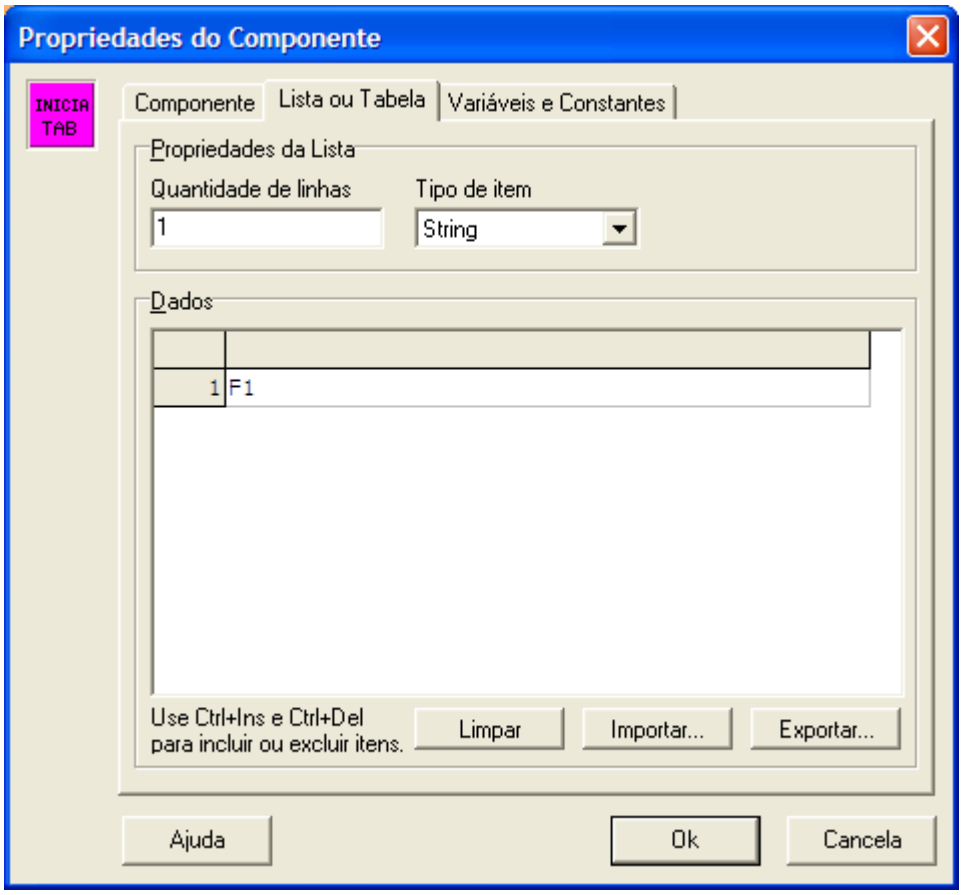


O exemplo a seguir exemplifica o uso desta Macro:

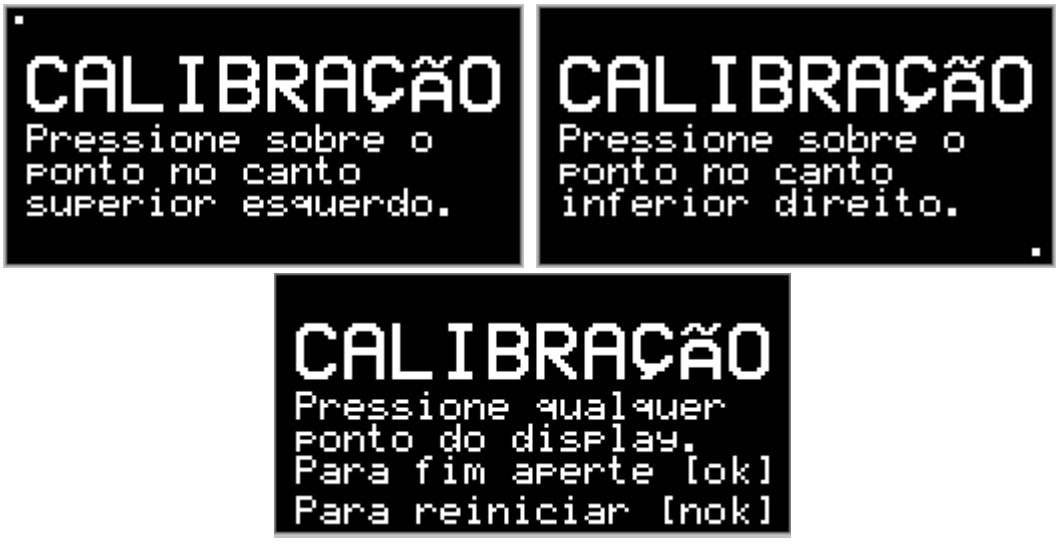


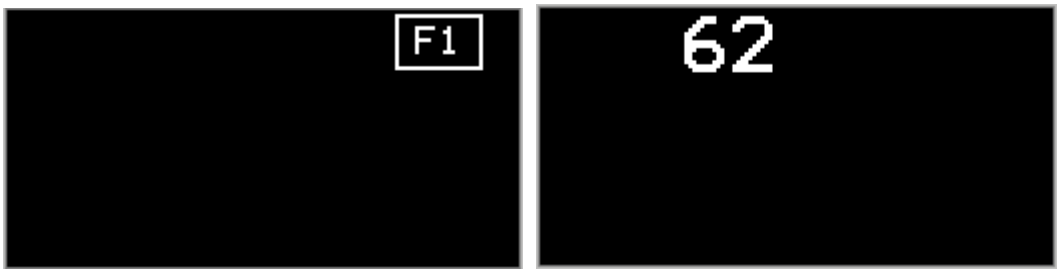
Exemplo de Programa Aplicativo: **Macro - TeclaX.dxg**

O conteúdo da tabela é o seguinte, nesse exemplo:



O resultado no display da IHM é mostrado a seguir, em seqüência. Note que, após a calibração do touchscreen surge a tecla **F1** e, ao pressioná-la, o display apresenta o valor da variável v0 (entrada analógica E1) em caracteres grandes, com o valor atualizado a cada segundo.



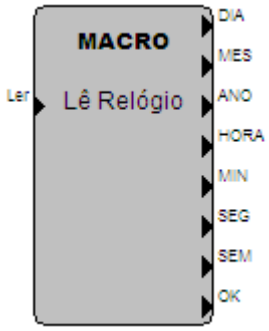


Atenção: Macro operacional apenas em controlador µDX201 versão 3.53 ou superior.

Relógio

Lê Relógio

Esta macro disponibiliza os dados do relógio de tempo real do controlador µDX200 em variáveis, facilitando seu uso. Note que o relógio interno é codificado em BCD e usa a mesma variável para especificar mais de um dado. A macro separa esses dados, além de transformar a codificação BCD em numeração decimal usual.



Exemplo de Macro: **Le_Relógio.dxm**

A Macro possui as saídas inteiras DIA, MES, ANO, HORA, MIN, SEG, SEM. As variáveis correspondem aos dados lidos do relógio interno. A variável SEM indica o dia da semana (de 1 a 7 correspondendo a segunda até domingo). A entrada binária Ler atualiza os dados com o valor do relógio interno do CLP. A saída binária OK indica que a operação foi completada (é gerado um pulso nesta saída ao final da operação).

SEM Dia da Semana (1 a 7 para segunda a domingo)

HORA Hora (0 a 23)

MIN Minuto (0 a 59)

SEG Segundo (0 a 59)

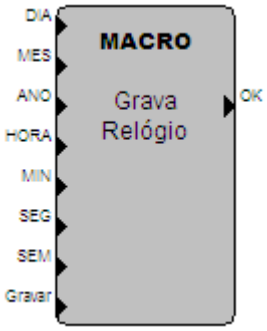
DIA Dia (1 a 31)

MES Mês (1 a 12)

ANO Ano (por exemplo, 2015)

Grava Relógio

Esta macro permite gravar uma nova data e hora no relógio de tempo real (RTC) do controlador µDX200.



Exemplo de Macro: **Grava_Relógio.dxm**

A Macro possui as entradas inteiras DIA, MES, ANO, HORA, MIN, SEG, SEM. As variáveis correspondem aos dados a serem gravados no relógio interno. A variável SEM indica o dia da semana (de 1 a 7 correspondendo a segunda até domingo): A entrada binária Gravar atualiza o relógio interno com os dados especificados nas demais entradas. A saída binária OK indica que a operação foi completada (é gerado um pulso nesta saída ao final da operação).

SEM Dia da Semana (1 a 7 para segunda a domingo)

HORA Hora (0 a 23)

MIN Minuto (0 a 59)

SEG Segundo (0 a 59)

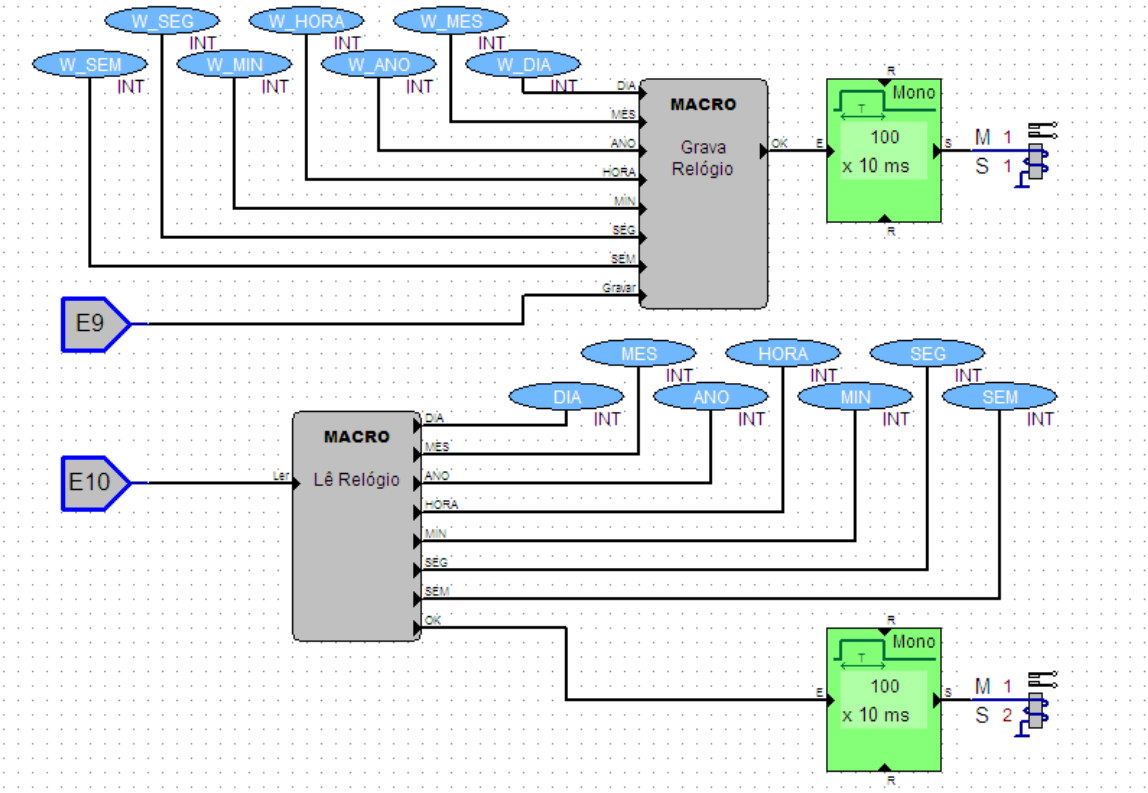
DIA Dia (1 a 31)

MES Mês (1 a 12)

ANO Ano (por exemplo, 2015)

O programa a seguir exemplifica o uso destas macros. As variáveis DIA, MES, ANO, HORA, MIN, SEG, SEM recebem o valor lido do relógio interno a cada energização da entrada digital E10. Na saída digital S2 da primeira expansão µDX210 é gerado um pulso de 1 segundo para indicar que o comando foi executado.

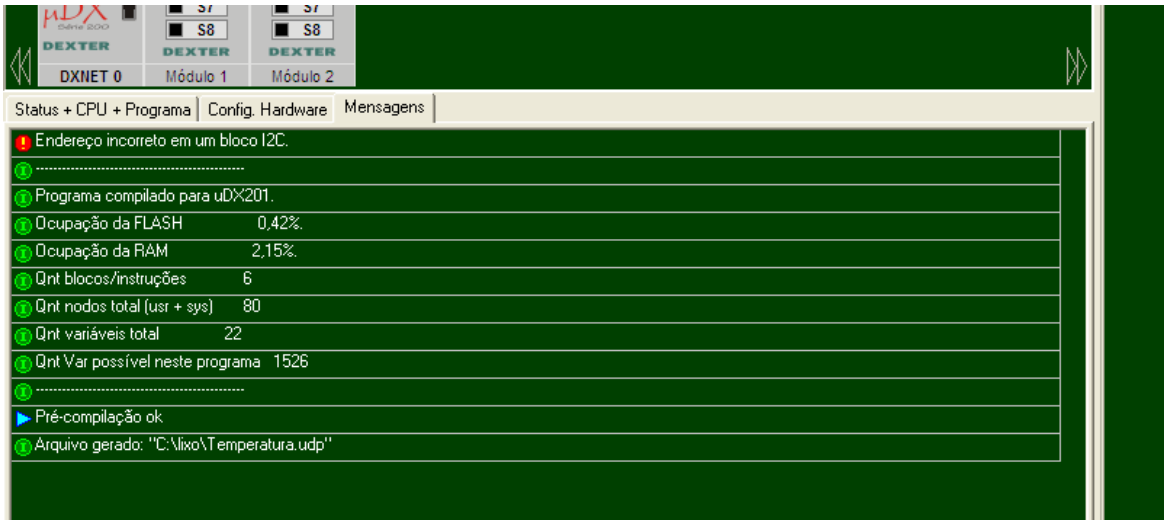
Já as variáveis W_DIA, W_MES, W_ANO, W_HORA, W_MIN, W_SEG, W_SEM permitem atualizar o relógio interno com o valor destas variáveis ao ser energizada a entrada digital E9 do controlador. Na saída digital S1 da primeira expansão µDX210 é gerado um pulso de 1 segundo para indicar que o comando foi executado.



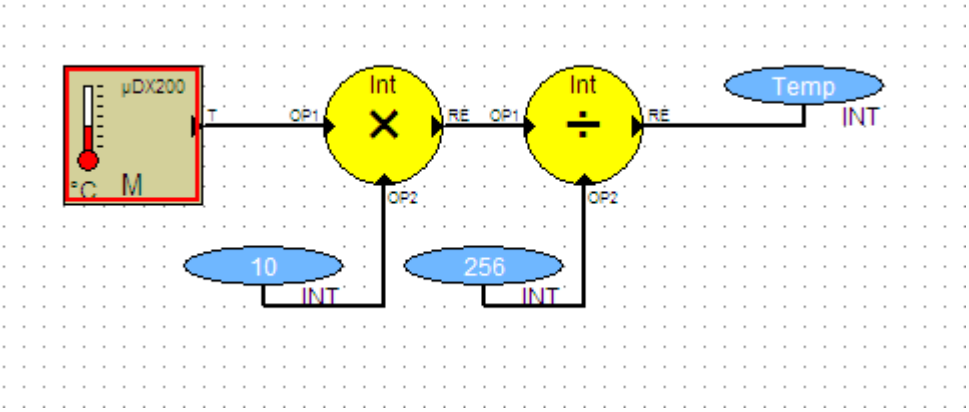
Exemplo de Programa Aplicativo: **Teste_Relógio.dxg**

Depuração de Erros

A partir da versão 2.2.0.16 do software PG foi introduzida uma ferramenta poderosa de depuração dos programas aplicativos. As mensagens de erro e aviso geradas pelo Pré-compilador e Compilador são apresentadas na aba **Mensagens** existente na janela Compilador e Interface µDX, como mostrado a seguir:



Note a existência de um erro no programa compilado. Para localizar o erro no programa aplicativo basta clicar duas vezes com a tecla esquerda do mouse sobre a linha que descreve o erro. Automaticamente o PG irá "pular" para o bloco que originou esse erro! Além disso, irá desenhar uma moldura vermelha ao redor do bloco com erro:



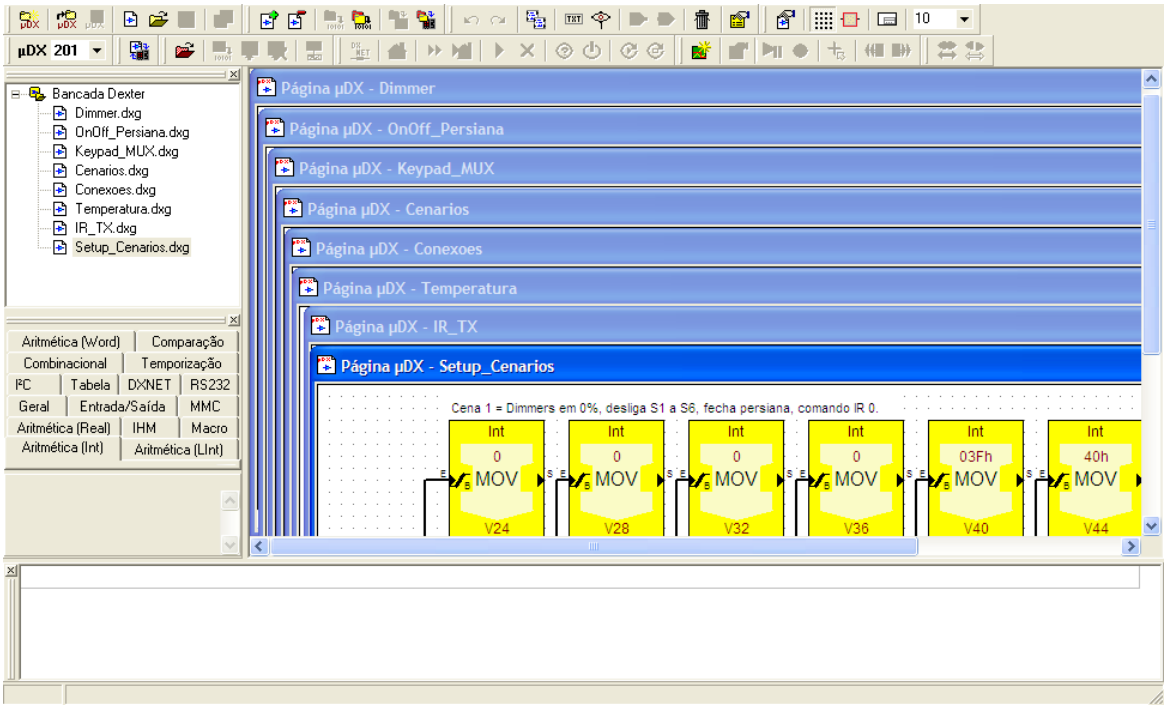
No caso, o erro foi originado por um bloco de rede I^2C sem indicação de endereço (0 a 7). Para desmarcar o bloco basta pressionar a tecla  (Limpa marcas de erro) existente na barra de ferramentas do PG.

Esta nova facilidade do PG deve auxiliar sobremaneira a depuração, principalmente em projetos extensos, com várias páginas.

Automação Residencial

Existem várias macros no PG para uso em automação residencial. O uso destas macros facilita muito a implementação de recursos normalmente encontrados em automação residencial, como controle de luminosidade (dimmer), cenários e controle via infravermelho. Para exemplificar o uso destas macros em um programa real foi incluído o projeto **Bancada Dexter** a este manual. Trata-se de uma aplicação desenvolvida em uma bancada para teste em automação residencial, e que usa praticamente todos os principais recursos comuns a uma automação residencial. Convém estudar este projeto, e em muitos casos usá-lo como programa inicial para gerar programas na área.

O projeto **Bancada Dexter** é composto de 8 páginas de programação: **Dimmer.dxg**, **OnOff_Persiana.dxg**, **Keypad_MUX.dxg**, **Cenarios.dxg**, **Conexoes.dxg**, **Temperatura.dxg**, **IR_TX.dxg** e **Setup_Cenarios.dxg**. Note que foram utilizadas macros de Automação Residencial X (otimizadas para controlador µDX201 versão 3.37 ou posterior).



Exemplo de Projeto: **Bancada.dxp**

Dimmer.dxg

Esta página aciona os quatro canais de um módulo de Mini-Dimmer. Para permitir o comando via aplicativo externo (aplicativo em tablet ou software supervisório) foram usadas variáveis absolutas (V19 a V22) nas entradas de **Valor** das macros. Além disso, sempre que estas variáveis são modificadas é gerado um pulso de **Latch** na macro correspondente. Assim, basta o aplicativo escrever na variável para que o canal dimmer seja modificado. Para ler o valor atual do dimmer o aplicativo pode ler a mesma variável, já que sempre que o valor do dimmer é modificado localmente (por exemplo, via tecla de Keypad) o novo valor é sobreescrito na variável absoluta correspondente. Os nodos N116 a N119 permitem acionar os canais do MIni-Dimmer localmente ou via aplicativo, pois se tratam de nodos absolutos. Já os nodos N120 a N123 indicam os canais ativos (valor de saída > 0).

OnOff_Persiana.dxg

Aqui temos um módulo de 8 saídas digitais (µDX212) sendo usado para controlar uma persiana (saídas S7 e S8), enquanto as demais saídas controlam cargas do tipo liga-desliga (por exemplo, iluminação sem controle de luminosidade). Note que, devido ao uso de duas saídas do µDX212 para controle de persiana, foi necessário usar um bloco **Nodos→Int** para gerar a variável de controle para o bloco **OUT I²C**. Se não houvesse o controle de persianas poderíamos ter usado a saída **OUT** da macro **µDX212** diretamente ligada a entrada do bloco **OUT I²C**.

Keypad_MUX.dxg

Esta página lê um Keypad com infravermelho, disponibilizando tanto as 8 teclas do Keypad quanto a decodificação de comandos via controle remoto. A página também inclui a leitura de dois Multiplexadores.

Cenarios.dxg

Quatro cenários foram criados na página, acionados pelos nodos absolutos N112 a N115. As variáveis V24 a V39 permitem especificar o valor de dimmer para os quatro canais do Mini-Diimmer em cada cenário. Como estes valores são armazenados em variáveis permitem que os valores em cada cenário sejam facilmente modificáveis via aplicativo em tablet ou software supervisório. Já as variáveis V40 a V43 indicam a máscara para o µDX212 referente a cada cenário. Já as variáveis V44 a V47 indicam o valor para o µDX212 referente a cada cenário. Por fim, as variáveis V48 a V51 indicam qual o comando infravermelho a ser transmitido no cenário.

Conexoes.dxg

Esta página conecta entradas e saídas, permitindo modificar quais entradas irão acionar cada recurso do programa. Por exemplo, KeyOut1_1 está ligado ao nodo N116. Com isso, a primeira tecla do Keypad controla o canal 1 do Mini-dimmer. Já as quatro cenas são acionadas pelas teclas C1 a C4 do controle remoto, por exemplo.

Temperatura.dxg

A variável absoluta V18 recebe o valor de temperatura lido em um sensor de temperatura lido ao controlador µDX201 via rede I²C. Já a variável V23 recebe o valor lido de luminosidade através de um sensor de luminosidade ligado à entrada analógica E8 do CLP.

IR-TX.dxg

Esta página controla o módulo de Transmissor de Infravermelho. Para transmitir um comando infravermelho previamente gravado no módulo IR-TX basta colocar o número do comando (comando 0 a comando 56) na variável **Comando**, e acionar o nodo **Transmite**. Isso é feito nos cenários, por exemplo. Para garantir que o módulo IR TX transmita o comando se incluiu um monoestável de 300 ms.

Já para gravar comandos no módulo IR TX devemos atribuir a variável **Com_Cap** o número do primeiro comando a ser capturado e acionar o nodo **Gravar**. A seguir, deve ser acionado momentaneamente o nodo **Captura**. Isso aciona o led de captura do módulo IR TX. Aí é só apontar o controle remoto a ser capturado para o módulo IR TX e pressionar a tecla a ser capturada. Caso o comando seja capturado corretamente o led de captura do módulo IR TX se apaga. Para capturar o próximo comando em ordem crescente basta acionar novamente a entrada **Captura**. A saída **Com_Cap2** indica o número do próximo comando a ser capturado. Convém manter o pulso em **Captura** durante cerca de 300 ms para garantir que o módulo IR TX irá entrar em modo de aprendizado. Para encerrar a captura de comandos basta desenergizar a entrada **Gravar**.

Setup_Cenarios.dxg

Esta última página atribui valores iniciais para os cenários. Assim, a cena 1 é tudo desligado, por exemplo. Já a cena 2 é tudo ligado.

Parte VIII

Blocos de Instruções

O Controlador µDX Série 200 (µDX200) reconhece e interpreta 341 blocos de instruções diferentes (biblioteca padrão para µDX200 versão 2.9), reunidos em 14 famílias:

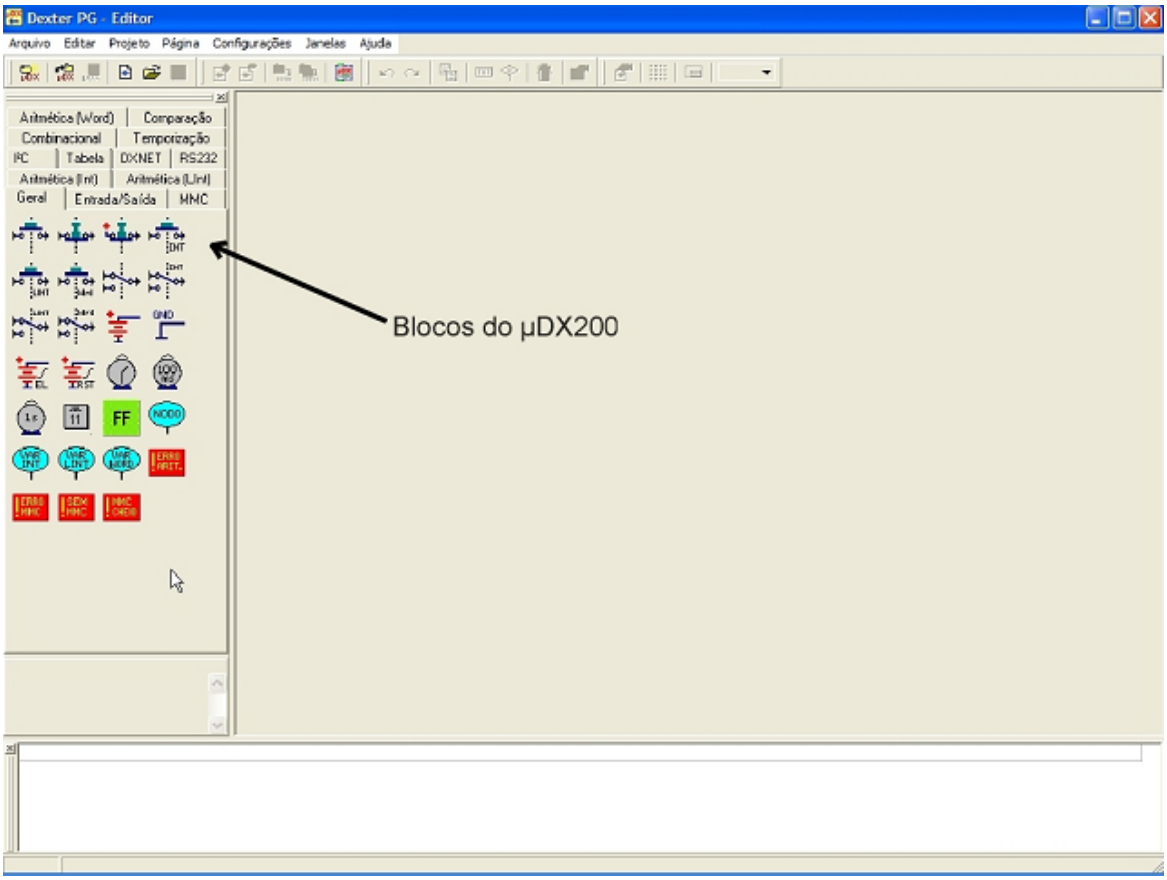
- **Aritmética Inteira**
- **Aritmética LongInt**
- **Aritmética Word**
- **Aritmética Real**
- **Comparação**
- **Combinacional**
- **Temporização**
- **Geral**
- **Entrada/Saída**
- **MMC**
- **I2C**
- **Tabela**
- **DXNET**
- **RS232**

No caso de controlador µDX201 existe uma família adicional para blocos de IHM, perfazendo 381 blocos (biblioteca padrão para µDX201 versão 1.6):

- **IHM**

O controlador µDX201 permite duas portas seriais auxiliares (RS232-2 e RS232-3). É preciso que a versão do µDX201 seja 3.46 ou superior:

- **RS232-2**
- **RS232-3**



O µDX200 trabalha com números inteiros, longint, word ou reais. Além disso, permite acessar tabelas em formato byte (8 bits) ou em formato string (seqüência de caracteres representados em ASCII e acessados como bytes).

Os blocos de aritmética inteira trabalham com números de 16 bits e faixa entre -32.768 e 32.767. Já os de aritmética utilizando inteiros longos (longint) trabalham com números de 32 bits e faixa entre -2.147.483.648 e 2.147.483.647. Os blocos de aritmética word trabalham com números de 16 bits, como os de aritmética inteira, mas a faixa abrangida por eles é de 0 a 65.535 (não existem números negativos). Por fim, os blocos de aritmética real operam na faixa entre -3,4E38 e 3,4E38. O menor valor absoluto real representável é 3,0E-39.

Os blocos de comparação permitem comparar variáveis entre si ou com constantes. Existem blocos de comparação para trabalhar com três formatos suportados pelo controlador: inteiro, longint e word.

Existem dois tipos de blocos de aritmética e comparação: Blocos quadrados e blocos circulares. Os blocos em formato quadrado utilizam variáveis binárias nas conexões ("fios" que interligam os blocos). Ou seja, estas conexões assumem apenas estado ligado ou desligado, e são utilizadas para ativar o bloco (como a conexão de entrada E, sensível a borda ou nível).

Já os blocos circulares utilizam variáveis nas conexões (variáveis essas que podem ser inteiras, longint ou word, conforme o bloco utilizado), e estão sempre ativos (não possuem conexão de entrada para ativar o bloco).

Os blocos combinacionais permitem efetuar operações de lógica booleana, como AND, OR, XOR, NOT, e estas funções invertidas.

Os blocos de temporização disponibilizam várias opções de temporizadores, como monoestáveis e atrasos.

No grupo Geral estão reunidos vários blocos, como chaves NA (normalmente abertas), chaves NF (normalmente fechadas), nodos de sistema (energia, reset, erros), FF (flip-flop), relógio e calendário.

Em Entrada/Saída encontram-se os blocos tanto para acesso as entradas e saídas digitais existentes nos módulos de Expansão de Entrada/Saída µDX210 quanto as entradas e saídas analógicas do Controlador µDX200, e ainda as entradas de contagem rápida E9 e E10.

Os blocos reunidos sob a designação MMC acessam o cartão MMC (multimedia card) que pode ser inserido no Controlador µDX200, permitindo gravar dados nesse (por exemplo, valores de variáveis, data e hora) em formato que permita a leitura posterior dos dados em um computador IBM-PC compatível.

Os blocos I^2C possibilitam o uso da rede de comunicação I^2C para leitura de sensores de temperatura e umidade ambientes.

Já os blocos Tabela criam tabelas tanto em memória não-volátil (memória do programa aplicativo) quanto em memória volátil (memória RAM). As tabelas podem ser declaradas, além dos formatos usuais suportados pelo µDX200 (inteiro, longint ou word), também no formato byte (8 bits) ou como um string (seqüência de bytes).

Os blocos DXNET permitem a comunicação entre controladores µDX200 ligados em rede. É possível transferir o estado de nodos (variáveis binárias) ou de variáveis de um µDX200 para outro, de forma a interligar programas aplicativos.

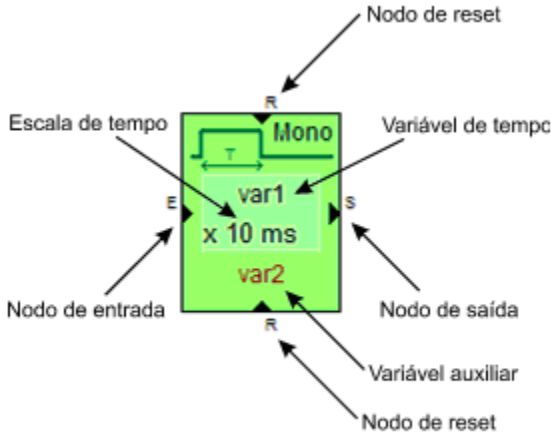
Os blocos RS232 tanto disponibilizam ler dados via a porta serial RS232, quanto transmitir strings ou o valor de variáveis por esta porta. Note que o µDX201 permite duas portas seriais auxiliares (RS232-2 e RS232-3).

Funcionamento dos Blocos de Tempo

No µDX200 os blocos de temporização (Monoestável, Pulso, Atraso e Oscilador) funcionam da seguinte maneira:

A constante ou a variável especificadas para programar o tempo escolhido são reconhecidas apenas uma vez a cada temporização pelo programa que controla estes blocos. Assim, quando o nodo de entrada de um bloco de monoestável é ligado o programa do µDX reconhece este estado e lê o valor da constante ou o valor atual da variável que indica o tempo programado para o bloco (Tempo).

O valor obtido é então memorizado numa variável auxiliar (Var.Auxiliar) para ser decrementado a cada intervalo de tempo (o intervalo de tempo pode ser programado pelo usuário em 1 ms, 10 ms, 100 ms ou ainda 1 s).



Bloco de Temporização
(Monoestável)

Note que tanto a variável de tempo quanto a variável auxiliar são variáveis do tipo word e, portanto, podem assumir valores entre 0 e 65535. Com isso, conforme a escala de tempo escolhida pode-se programar os seguintes tempos:

Escala de Tempo	Tempo Mínimo	Tempo Máximo
x 1 ms	1 ms	65,535 s = 1 min 5 s
x 10 ms	10 ms	655,35 s = 10 min 55 s
x 100 ms	100 ms	6553,5 s = 1 h 49 min 13 s
x 1 s	1 s	65535 s = 18 h 12 min 15 s

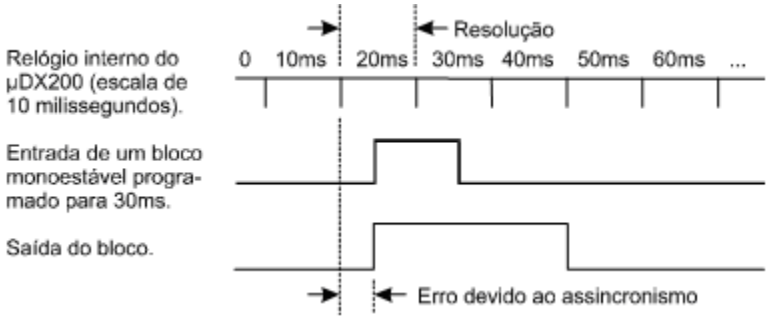
Quando o valor nesta variável auxiliar chegar a zero o programa do µDX vai executar a segunda parte da função do bloco. No caso do monoestável isto significa desligar o nodo de saída e para os outros tipos (como o blocos de atraso) seria o momento de ligar o nodo de saída.

Como o µDX não dispõe de espaço de RAM interna para variáveis auxiliares, estas são obtidas pela utilização das variáveis de cálculo disponíveis. Caso não seja especificada a variável auxiliar no bloco de temporização o compilador irá alocar um espaço vago na memória RAM disponível.

Note que qualquer modificação no valor de uma variável que se usa para programar o tempo de um destes blocos não será notada senão quando ele for ativado. Por outro lado, se o programa feito pelo usuário interferir no valor da variável utilizada como auxiliar, isto vai resultar em alteração do período de tempo de atividade do bloco.

AT*ENÇÃO: No caso de blocos de temporização utilizando escala de tempo de 1 ms é necessário que o tempo de ciclo do programa aplicativo seja menor ou igual a 1 ms, ou o bloco de temporização sofrerá um atraso na temporização, devido ao tempo de ciclo do programa ser superior à resolução da temporização. Por exemplo, um bloco de monoestável de 10ms (com escala de 1ms e variável de tempo igual a 10), se utilizado em um programa com tempo de ciclo de 2 ms, pode apresentar uma temporização de até 13 ms (2 ms adicionais devido ao tempo de ciclo do programa aplicativo e 1 ms adicional devido a imprecisão("jitter") dos blocos de temporização).*

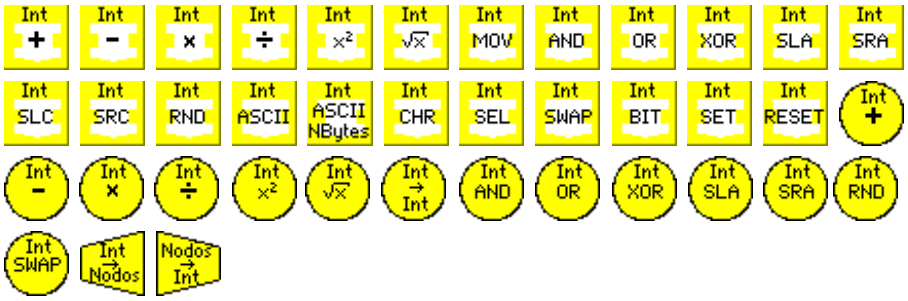
Observação: Para cada escala de tempo escolhida existe uma imprecisão ("jitter") para a duração ajustada. Assim, por exemplo, a escala de 10ms tem resolução de 10ms, o que quer dizer que a duração programada é sempre um intervalo múltiplo de 10ms. Além disso, esta resolução torna o intervalo total programado dependente do momento em que o temporizador é disparado. Isto ocorre porque para cada escala os sinais internos que fazem a contagem do tempo programado ocorrem em momentos fixos. Veja a figura:



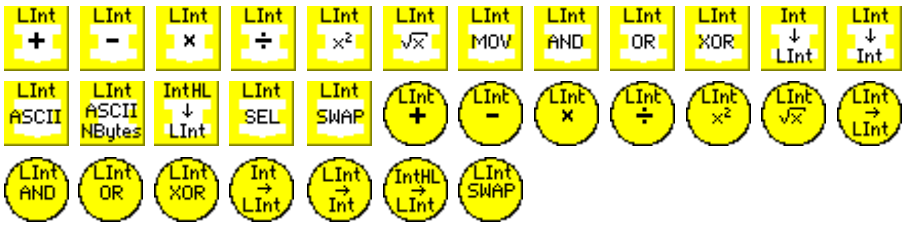
Assim, para temporizadores selecionados para a escala de 10ms existe uma incerteza de 10ms. Já para temporizadores selecionados para a escala de segundos esta incerteza é de 1 segundo, e assim por diante. Entretanto, em escalas de tempo muito rápidas (escala de 1ms) e programas aplicativos grandes podem ocorrer atrasos adicionais devido ao processamento do programa, como já frisado.

Famílias de Blocos

Aritmética Inteira

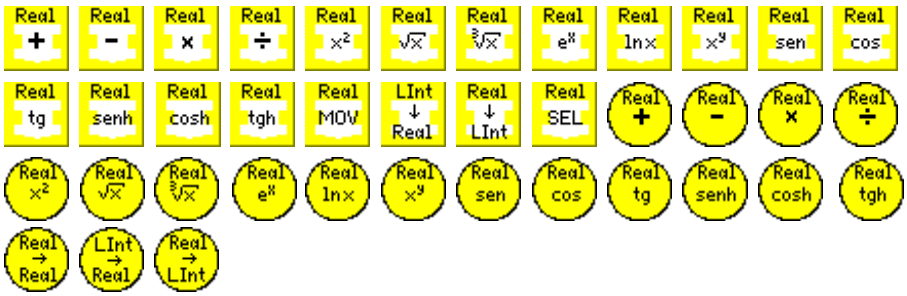


Aritmética LongInt

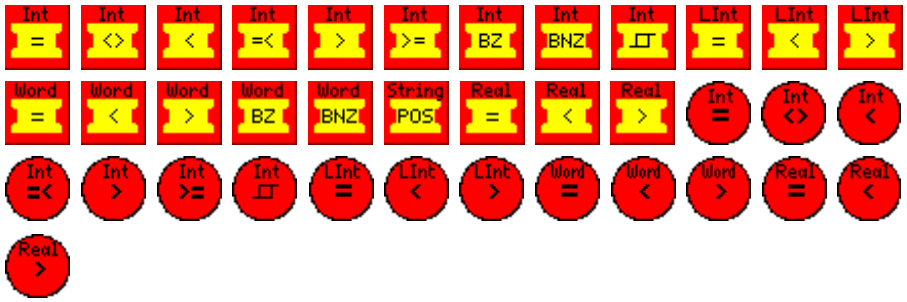


Aritmética Word

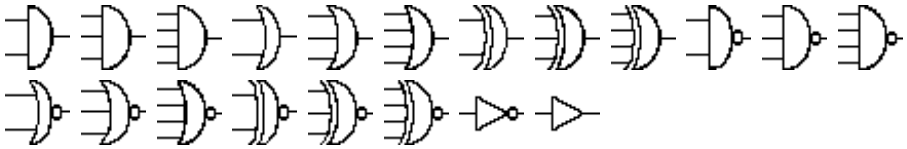
Aritmética Real



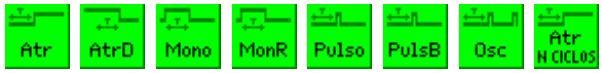
Comparação



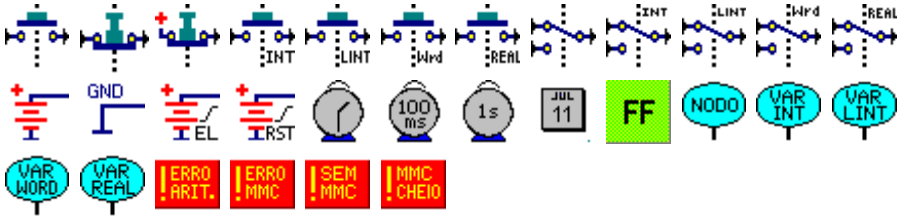
Combinacional



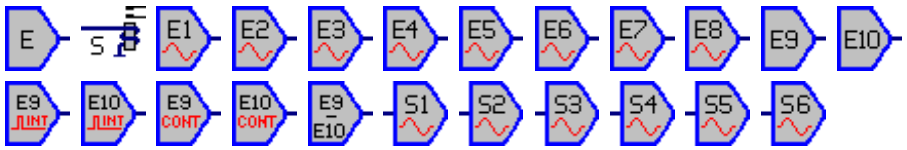
Temporização



Geral



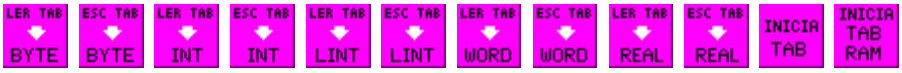
Entrada/Saída



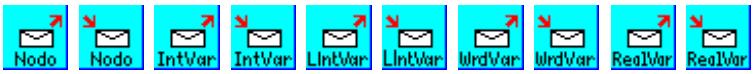
MMC

I2C

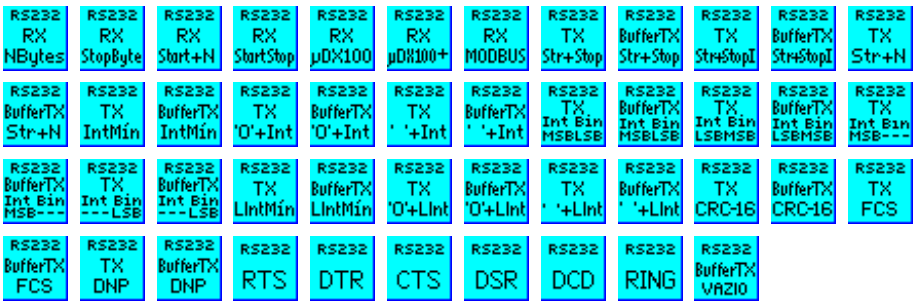
Tabela



DXNET



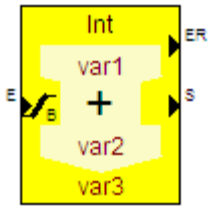
RS232



IHM

ARITMÉTICA INTEIRA - Adição

Este bloco efetua a adição de dois operandos inteiros, colocando o resultado em um terceiro operando inteiro.



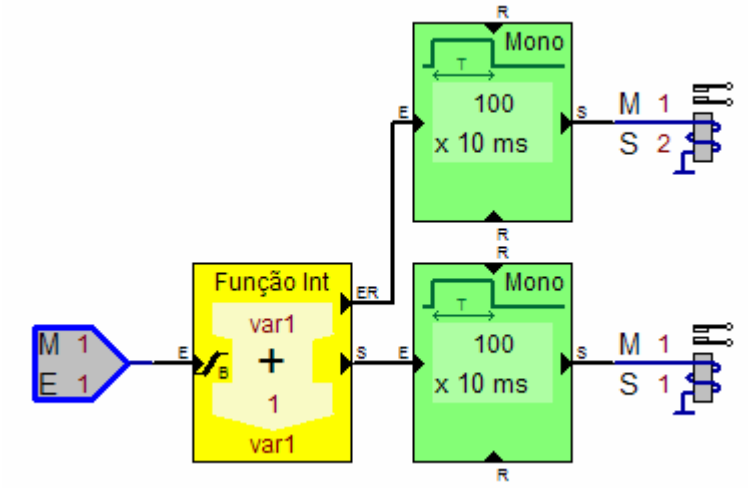
Operando1 e Operando2 são os operandos a serem adicionados e Resultado recebe o resultado da adição.

Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow ou underflow) na operação. Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Assim, por exemplo, se somarmos duas variáveis, ambas com valor 30000 o resultado será excessivo (60000) para representação via inteiros e o nodo de erro será ativado. Neste caso deverá ser usado um bloco de adição longint.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Adição.dxg**

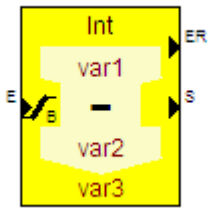
Acima temos um pequeno exemplo de uso do bloco de adição inteira. No caso, a variável var1 é incrementada cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de adição está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de adição inteira. Note que foram colocados monoestáveis nas saídas S e ER do bloco. Isso é necessário para visualizar

o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

Veja também:
ARITMÉTICA INTEIRA - Adição Var
ARITMÉTICA LONGINT - Adição
ARITMÉTICA LONGINT - Adição Var
ARITMÉTICA WORD - Adição
ARITMÉTICA WORD - Adição Var
ARITMÉTICA REAL - Adição
ARITMÉTICA REAL - Adição Var

ARITMÉTICA INTEIRA - Subtração

Este bloco efetua a subtração de dois operandos inteiros, colocando o resultado em um terceiro operando inteiro.



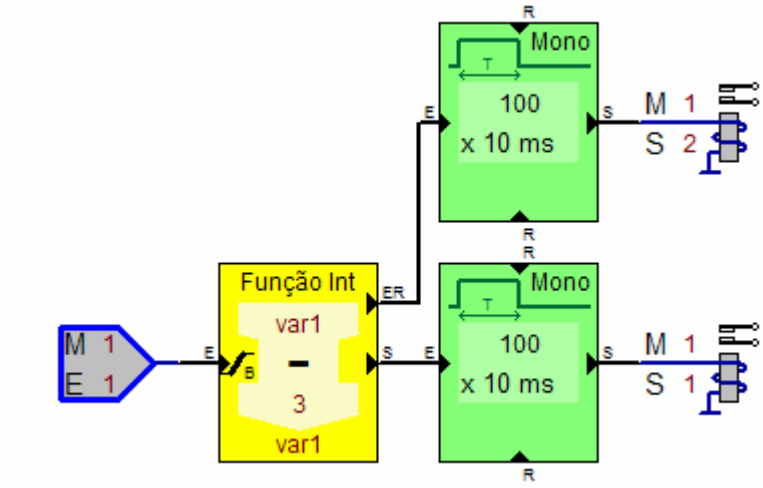
Operando1 e Operando2 são os operandos a serem subtraídos e Resultado recebe o resultado da subtração.

Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow ou underflow) na operação. Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Assim, por exemplo, se subtrairmos duas variáveis, a primeira com valor 30000 e a segunda com valor -30000 o resultado será excessivo (60000) para representação via inteiros e o nodo de erro será ativado. Neste caso deverá ser usado um bloco de subtração longint.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



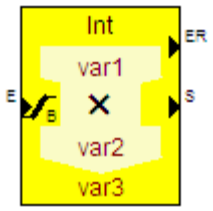
Exemplo de Programa Aplicativo: **Aritmética Inteira - Subtração.dxg**

Acima temos um pequeno exemplo de uso do bloco de subtração inteira. No caso, a variável var1 é decrementada em 3 unidades cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de subtração está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de subtração inteira. Note que foram colocados monoestáveis nas saídas S e ER do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

Veja também:
ARITMÉTICA INTEIRA - Subtração Var
ARITMÉTICA LONGINT - Subtração
ARITMÉTICA LONGINT - Subtração Var
ARITMÉTICA WORD - Subtração
ARITMÉTICA WORD - Subtração Var
ARITMÉTICA REAL - Subtração
ARITMÉTICA REAL - Subtração Var

ARITMÉTICA INTEIRA - Multiplicação

Este bloco efetua a multiplicação de dois operandos inteiros, colocando o resultado em um terceiro operando inteiro.



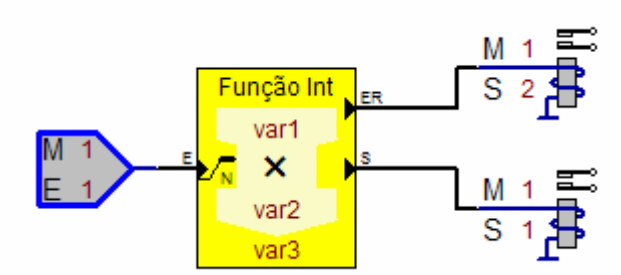
Operando1 e Operando2 são os operandos a serem multiplicados e Resultado recebe o resultado da multiplicação.

Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow ou underflow) na operação. Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Assim, por exemplo, se multiplicarmos duas variáveis, ambas com valor 200 o resultado será excessivo (40000) para representação via inteiros e o nodo de erro será ativado. Neste caso deverá ser usado um bloco de multiplicação longint.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



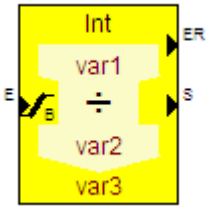
Exemplo de Programa Aplicativo: **Aritmética Inteira - Multiplicação.dxg**

Acima temos um pequeno exemplo de uso do bloco de multiplicação inteira. No caso, a variável var1 é multiplicada pela variável var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de multiplicação está selecionada por nível). O resultado da operação é armazenado na variável var3. Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de multiplicação inteira. Note que neste caso não foram colocados monoestáveis nas saídas S e ER do bloco, ao contrário dos exemplos anteriores. Isso porque a entrada E do bloco de multiplicação está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA INTEIRA - Multiplicação Var
ARITMÉTICA LONGINT - Multiplicação
ARITMÉTICA LONGINT - Multiplicação Var
ARITMÉTICA REAL - Multiplicação
ARITMÉTICA REAL - Multiplicação Var

ARITMÉTICA INTEIRA - Divisão

Este bloco efetua a divisão de dois operandos inteiros, colocando o resultado em um terceiro operando inteiro.



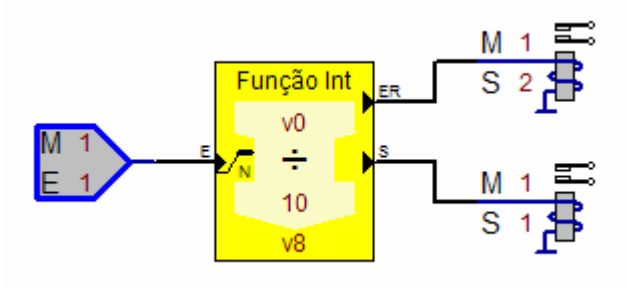
Operando1 e Operando2 são os operandos a serem divididos e Resultado recebe o resultado da divisão.

Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Assim, a divisão sempre resultará em um valor válido, exceto no caso de divisão por zero. Ou seja, o nodo ER só será ativado se o operando 2 assumir valor zero.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Divisão.dxg**

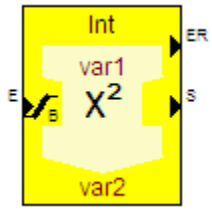
Acima temos um pequeno exemplo de uso do bloco de divisão inteira. No caso, a variável v0 (que corresponde a entrada analógica E1 do µDX200) é dividida pela constante 10 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de divisão está selecionada por nível). O resultado da operação é armazenado na variável v8 (que corresponde a saída analógica S1 do µDX200). Ou seja, a saída analógica S1 irá rastrear a entrada analógica E1 dividida por 10 do µDX200. Se tanto a entrada quanto a saída analógica estiver para a escala de 0-10V, se aplicarmos 6V na entrada E1 irá resultar em 0,6V na saída S1.

Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de divisão inteira. Note que neste caso não foram colocados monoestáveis nas saídas S e ER do bloco, ao contrário de exemplos anteriores. Isso porque a entrada E do bloco de divisão está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA INTEIRA - Divisão Var
ARITMÉTICA LONGINT - Divisão
ARITMÉTICA LONGINT - Divisão Var
ARITMÉTICA REAL - Divisão
ARITMÉTICA REAL - Divisão Var

ARITMÉTICA INTEIRA - Quadrado

Este bloco eleva ao quadrado um operando inteiro, colocando o resultado em um segundo operando inteiro.



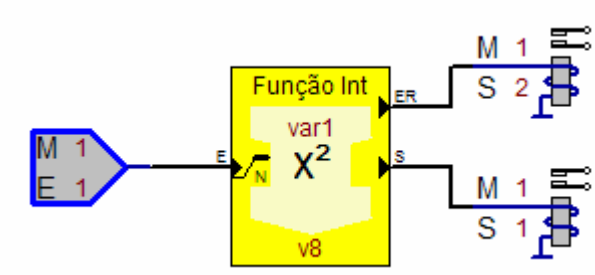
Operando é o operando a ser elevado ao quadrado e Resultado recebe o resultado da operação.

Note que Operando pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow ou underflow) na operação. Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Assim, por exemplo, se elevarmos ao quadrado um valor superior a 181 ou inferior a -181 o resultado será excessivo para representação via inteiros e o nodo de erro será ativado. Neste caso deverá ser usado um bloco de quadrado longint.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Quadrado.dxg**

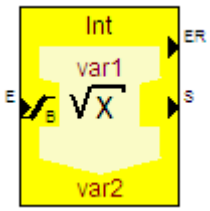
Acima temos um pequeno exemplo de uso do bloco de quadrado inteiro. No caso, a variável var1 é elevada ao quadrado sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de quadrado está selecionada por nível). O resultado da operação é armazenado na variável v8 (que corresponde a saída analógica S1 do µDX200). Ou seja, a saída analógica S1 irá assumir o valor do quadrado de var1. Se var1 for igual a 50 v8 será 2500. Se esta saída analógica estiver selecionada para escala de 0-10V, irá assumir tensão de saída de 10*2500/4095 = 6,105V.

Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de quadrado inteiro. A entrada E do bloco de quadrado está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA INTEIRA - Quadrado Var
ARITMÉTICA LONGINT - Quadrado
ARITMÉTICA LONGINT - Quadrado Var
ARITMÉTICA REAL - Quadrado
ARITMÉTICA REAL - Quadrado Var

ARITMÉTICA INTEIRA - Raiz Quadrada

Este bloco calcula a raiz quadrada de um operando inteiro, colocando o resultado em um segundo operando inteiro.



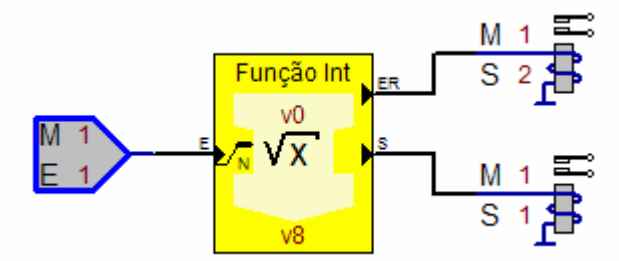
Operando é o operando a ser efetuada a raiz quadrada e Resultado recebe o resultado da operação.

Note que Operando pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. No caso deste bloco o erro só ocorre se for tentado retirar a raiz quadrada de um número negativo.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Raiz Quadrada.dxg**

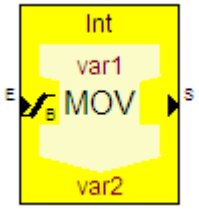
Acima temos um pequeno exemplo de uso do bloco de raiz quadrada inteira. No caso, é retirada a raiz quadrada da variável v0 (que corresponde a entrada analógica E1 do µDX200) sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de raiz quadrada está selecionada por nível). O resultado da operação é armazenado na variável v8 (que corresponde a saída analógica S1 do µDX200). Ou seja, a saída analógica S1 irá rastrear a entrada analógica raiz quadrada de E1 do µDX200. Se tanto a entrada quanto a saída analógica estiver para a escala de 0-10V, se aplicarmos 9V na entrada E1 irá resultar em 3V na saída S1.

Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de raiz quadrada inteira. A entrada E do bloco de raiz quadrada está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA INTEIRA - Raiz Quadrada Var
ARITMÉTICA LONGINT - Raiz Quadrada
ARITMÉTICA LONGINT - Raiz Quadrada Var
ARITMÉTICA REAL - Raiz Quadrada
ARITMÉTICA REAL - Raiz Quadrada Var

ARITMÉTICA INTEIRA - Atribuição

Este bloco transfere o valor de um operando inteiro para um segundo operando inteiro.



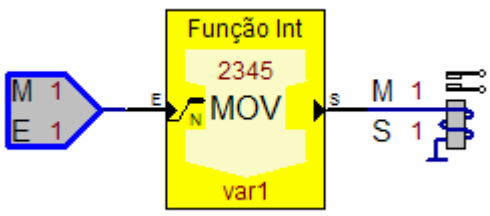
Operando é o operando cujo valor será transferido para Resultado.

Note que Operando pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Atribuição.dxg**

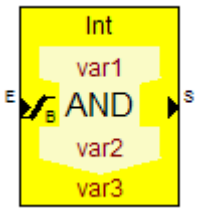
Acima temos um pequeno exemplo de uso do bloco de atribuição inteira. No caso, é atribuído o valor 2345 na variável inteira var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível).

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição inteira. A entrada E do bloco de atribuição está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA INTEIRA - Atribuição Var
ARITMÉTICA LONGINT - Atribuição
ARITMÉTICA LONGINT - Atribuição Var
ARITMÉTICA WORD - Atribuição
ARITMÉTICA WORD - Atribuição Var
ARITMÉTICA REAL - Atribuição
ARITMÉTICA REAL - Atribuição Var

ARITMÉTICA INTEIRA - Operação AND

Este bloco efetua a operação booleana AND entre dois operandos inteiros, e Resultado recebe o resultado da operação. Observe que a operação AND é feita bit a bit com os 16 bits dos operandos.

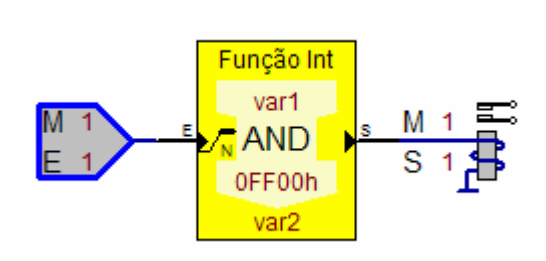


Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 0000h a 0FFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A30h e não A30h (pois o µDX200 entenderia que se trata da variável chamada A30H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação AND.dxg**

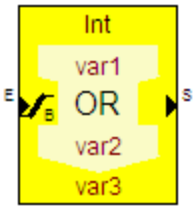
Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana AND inteira. No caso, é efetuada a operação AND entre a variável var1 e a constante hexadecimal 0FF00h sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível). O resultado da operação é guardado em var2. Note que esta operação zera o byte LSB de var1 (já que qualquer valor AND com 00h resulta em zero). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação AND com 0FF00h irá resultar no valor 3328 (0D00h) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição inteira. A entrada E do bloco de AND está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA INTEIRA - Operação AND Var
ARITMÉTICA LONGINT - Operação AND
ARITMÉTICA LONGINT - Operação AND Var
ARITMÉTICA WORD - Operação AND
ARITMÉTICA WORD - Operação AND Var

ARITMÉTICA INTEIRA - Operação OR

Este bloco efetua a operação booleana OR entre dois operandos inteiros, e Resultado recebe o resultado da operação. Observe que a operação OR é feita bit a bit com os 16 bits dos operandos.

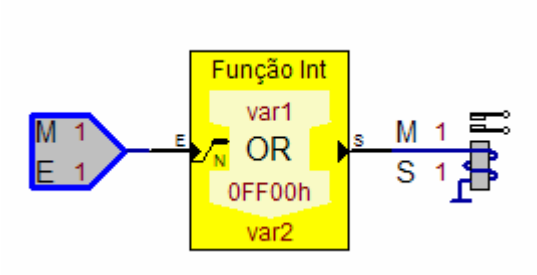


Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 0000h a 0FFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A30h e não A30h (pois o µDX200 entenderia que se trata da variável chamada A30H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação OR.dxg**

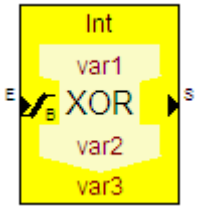
Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana OR inteira. No caso, é efetuada a operação OR entre a variável var1 e a constante hexadecimal 0FF00h sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível). O resultado da operação é guardado em var2. Note que esta operação liga todos os bits do byte MSB de var1 (já que qualquer valor OR com 0FFh resulta em um). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação OR com 0FF00h irá resultar no valor -85 (0FFABh) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição inteira. A entrada E do bloco de OR está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA INTEIRA - Operação OR Var
ARITMÉTICA LONGINT - Operação OR
ARITMÉTICA LONGINT - Operação OR Var
ARITMÉTICA WORD - Operação OR
ARITMÉTICA WORD - Operação OR Var

ARITMÉTICA INTEIRA - Operação XOR

Este bloco efetua a operação booleana XOR entre dois operandos inteiros, e Resultado recebe o resultado da operação. Observe que a operação XOR é feita bit a bit com os 16 bits dos operandos.



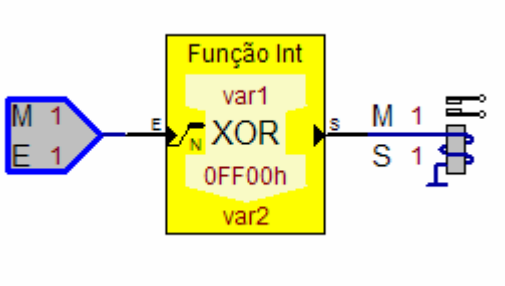
Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado

deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 0000h a 0FFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A30h e não A30h (pois o µDX200 entenderia que se trata da variável chamada A30H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação XOR.dxg**

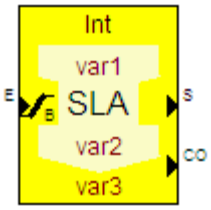
Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana XOR inteira. No caso, é efetuada a operação XOR entre a variável var1 e a constante hexadecimal 0FF00h sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível). O resultado da operação é guardado em var2. Note que esta operação inverte todos os bits do byte MSB de var1 (já que qualquer valor XOR com 0FFh resulta em inversão bit a bit). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação XOR com 0FF00h irá resultar no valor -3413 (0F2ABh) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição inteira. A entrada E do bloco de XOR está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA INTEIRA - Operação XOR Var
ARITMÉTICA LONGINT - Operação XOR
ARITMÉTICA LONGINT - Operação XOR Var
ARITMÉTICA WORD - Operação XOR
ARITMÉTICA WORD - Operação XOR Var

ARITMÉTICA INTEIRA - Operação SLA

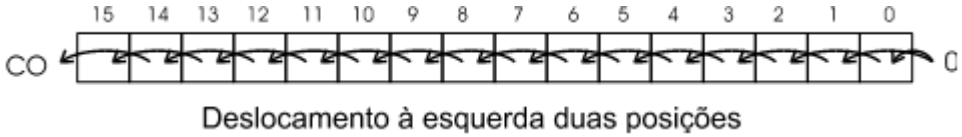
Este bloco efetua a operação de deslocamento aritmético à esquerda do Operando1 pelo número de deslocamentos indicado pelo Operando2, e Resultado recebe o resultado da operação.



Observe que a operação SLA (shift left arithmetical) é feita bit a bit com os 16 bits do Operando1, segundo o número de deslocamentos especificado por Operando2. Assim, se Operando2 igual a um o Operando1 será deslocado em uma posição para a esquerda (equivalente a multiplicar por dois). Já se Operando2 igual a dois o Operando1 será deslocado em duas posições para a esquerda (equivalente a multiplicar por quatro), e assim sucessivamente.

Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Note que os bits LSB são preenchidos com zeros, enquanto o bit MSB é disponibilizado no nodo de saída Carry Out (CO).

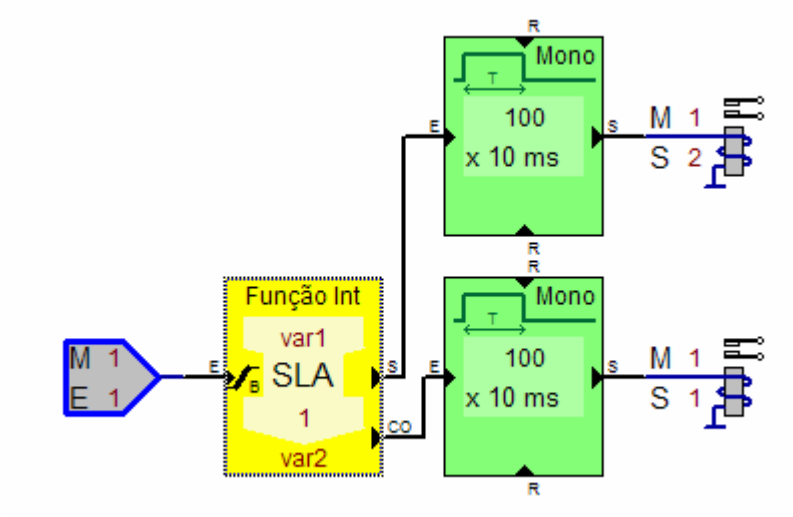
Deslocamento à esquerda uma posição



Deslocamento à esquerda duas posições

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação SLA.dxg**

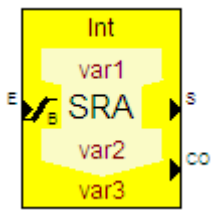
Na figura acima está representado um pequeno exemplo de uso do bloco de deslocamento aritmético à esquerda. No caso, é efetuada a operação SLA em uma posição na variável var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for energizada (já que a entrada E do bloco de atribuição está selecionada por borda). O resultado da operação é guardado em var2. Note que esta operação equivale a multiplicar por dois o valor da variável var1. Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação SLA com deslocamento unitário irá resultar no valor 6998 (1B56h) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco, assim como o nodo CO. A entrada E do bloco SLA está selecionada para acionamento por borda e, portanto, a saída S e CO ficarão ativas durante apenas um ciclo de execução do programa aplicativo (<500µs). Então, foram inseridos monoestáveis para aumentar os pulsos de saída para um segundo, de forma a serem visíveis.

Veja também:
ARITMÉTICA INTEIRA - Operação SRA
ARITMÉTICA INTEIRA - Operação SLC
ARITMÉTICA INTEIRA - Operação SRC
ARITMÉTICA WORD - Operação SLC
ARITMÉTICA WORD - Operação SRC
ARITMÉTICA INTEIRA - Operação SLA Var
ARITMÉTICA INTEIRA - Operação SRA Var

ARITMÉTICA INTEIRA - Operação SRA

Este bloco efetua a operação de deslocamento aritmético à direita do Operando1 pelo número de deslocamentos indicado pelo Operando2, e Resultado recebe o resultado da operação.

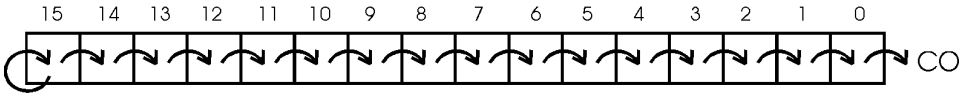


Observe que a operação SRA (shift right arithmetical) é feita bit a bit com os 16 bits do

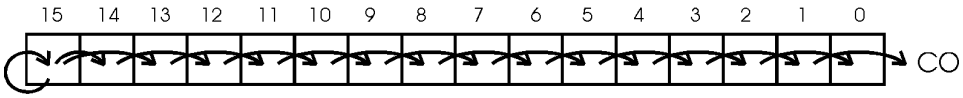
Operando1, segundo o número de deslocamentos especificado por Operando2. Assim, se Operando2 igual a um o Operando1 será deslocado em uma posição para a direita (equivalente a dividir por dois). Já se Operando2 igual a dois o Operando1 será deslocado em duas posições para a direita (equivalente a dividir por quatro), e assim sucessivamente.

Como se trata de deslocamento aritmético o bit mais significativo (bit 15) é preservado, de forma a manter o sinal do valor deslocado. Por exemplo, -32768 (8000h) deslocado uma posição à direita equivale a dividir por dois o valor original, ou seja, -16384 (C000h).

Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Note que os bits MSB são preenchidos com o valor do bit 15, enquanto o bit LSB é disponibilizado no nodo de saída Carry Out (CO).



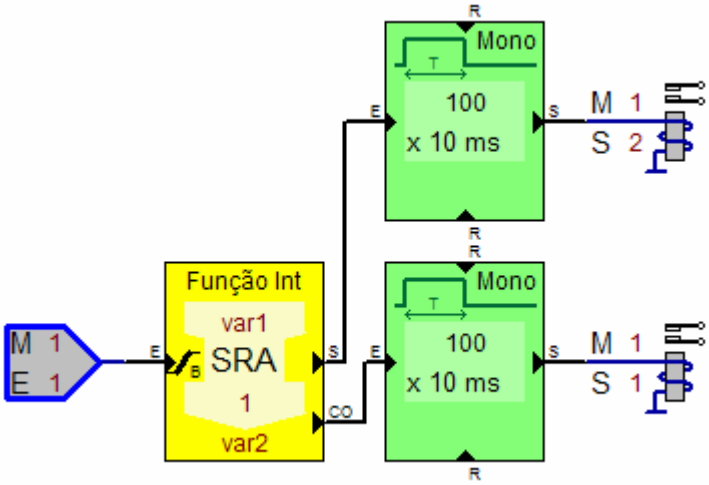
Deslocamento à direita uma posição



Deslocamento à direita duas posições

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação SRA.dxg**

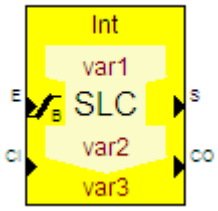
Na figura acima está representado um pequeno exemplo de uso do bloco de deslocamento aritmético à direita. No caso, é efetuada a operação SRA em uma posição na variável var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for energizada (já que a entrada E do bloco de atribuição está selecionada por borda). O resultado da operação é guardado em var2. Note que esta operação equivale a dividir por dois o valor da variável var1. Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação SRA com deslocamento unitário irá resultar no valor 1749 (6D5h) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco, assim como o nodo CO. A entrada E do bloco SRA está selecionada para acionamento por borda e, portanto, a saída S e CO ficarão ativas durante apenas um ciclo de execução do programa aplicativo (<500µs). Então, foram inseridos monoestáveis para aumentar os pulsos de saída para um segundo, de forma a serem visíveis.

Veja também:
ARITMÉTICA INTEIRA - Operação SLA
ARITMÉTICA INTEIRA - Operação SLC
ARITMÉTICA INTEIRA - Operação SRC
ARITMÉTICA WORD - Operação SLC
ARITMÉTICA WORD - Operação SRC
ARITMÉTICA INTEIRA - Operação SLA Var
ARITMÉTICA INTEIRA - Operação SRA Var

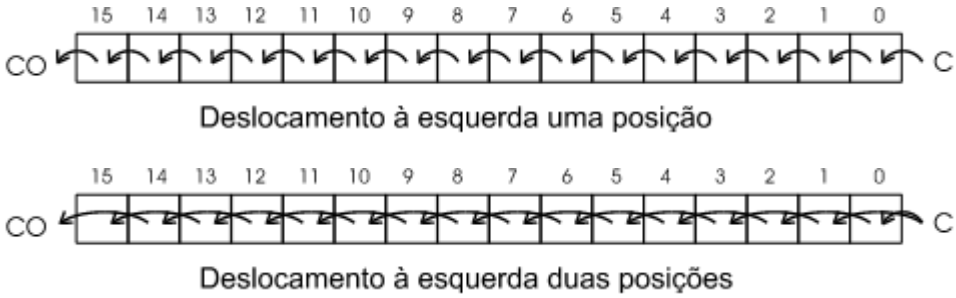
ARITMÉTICA INTEIRA - Operação SLC

Este bloco efetua a operação de deslocamento com carry à esquerda do Operando1 pelo número de deslocamentos indicado pelo Operando2, e Resultado recebe o resultado da operação.



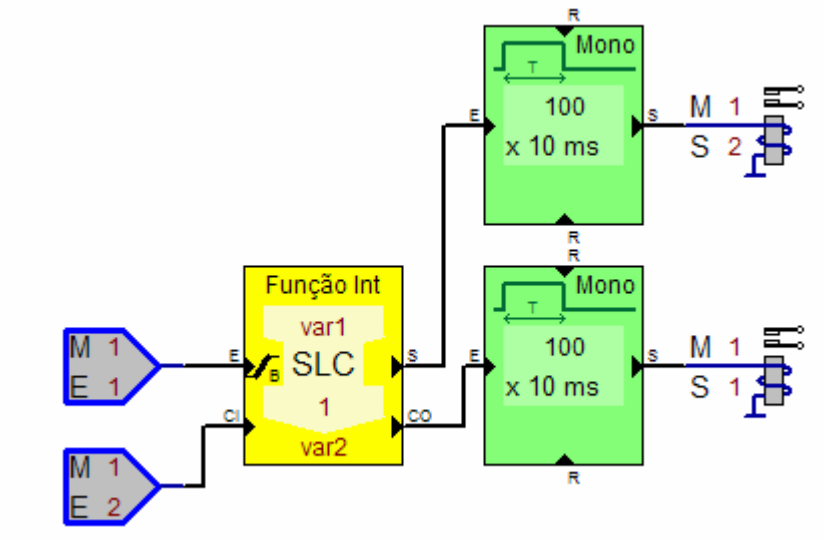
Observe que a operação SLC (shift left carry) é feita bit a bit com os 16 bits do Operando1, segundo o número de deslocamentos especificado por Operando2. Assim, se Operando2 igual a um o Operando1 será deslocado em uma posição para a esquerda. Já se Operando2 igual a dois o Operando1 será deslocado em duas posições para a esquerda, e assim sucessivamente.

Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Note que os bits LSB são preenchidos com o valor do nodo de entrada Carry In (CI), enquanto o bit MSB é disponibilizado no nodo de saída Carry Out (CO).



O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação SLC.dxg**

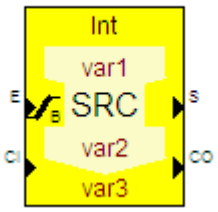
Na figura acima está representado um pequeno exemplo de uso do bloco de deslocamento à esquerda com carry. No caso, é efetuada a operação SLC em uma posição na variável var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for energizada (já que a entrada E do bloco de atribuição está selecionada por borda). O resultado da operação é guardado em var2. Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação SLC com deslocamento unitário irá resultar no valor 6998 (1B56h) na variável var2, se o nodo CI for mantido desligado, ou irá resultar no valor 6999 (1B57h) se este nodo estiver ligado.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco, assim como o nodo CO. A entrada E do bloco SLC está selecionada para acionamento por borda e, portanto, a saída S e CO ficarão ativas durante apenas um ciclo de execução do programa aplicativo (<500µs). Então, foram inseridos monoestáveis para aumentar os pulsos de saída para um segundo, de forma a serem visíveis.

Veja também:
ARITMÉTICA INTEIRA - Operação SRC
ARITMÉTICA INTEIRA - Operação SLA
ARITMÉTICA INTEIRA - Operação SRA
ARITMÉTICA WORD - Operação SLC
ARITMÉTICA WORD - Operação SRC

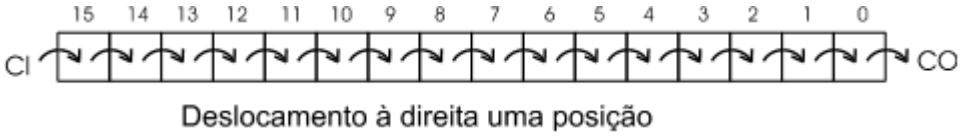
ARITMÉTICA INTEIRA - Operação SRC

Este bloco efetua a operação de deslocamento com carry à direita do Operando1 pelo número de deslocamentos indicado pelo Operando2, e Resultado recebe o resultado da operação.

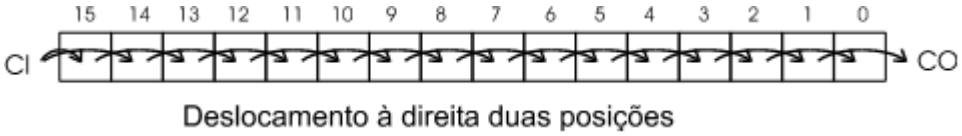


Observe que a operação SRC (shift right carry) é feita bit a bit com os 16 bits do Operando1, segundo o número de deslocamentos especificado por Operando2. Assim, se Operando2 igual a um o Operando1 será deslocado em uma posição para a direita. Já se Operando2 igual a dois o Operando1 será deslocado em duas posições para a direita, e assim sucessivamente.

Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Note que os bits MSB são preenchidos com o estado do nodo de entrada Carry In (CI), enquanto o bit LSB é disponibilizado no nodo de saída Carry Out (CO).



Deslocamento à direita uma posição

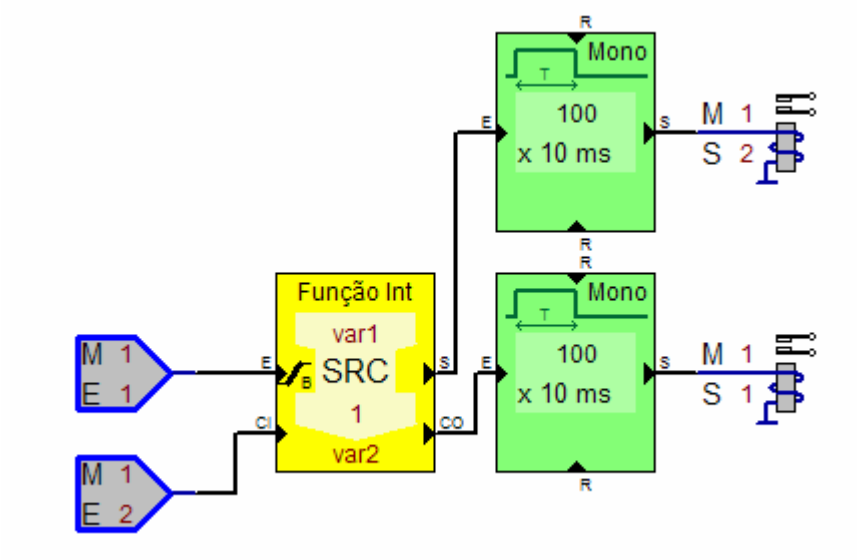


Deslocamento à direita duas posições

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo).

Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação SRC.dxg**

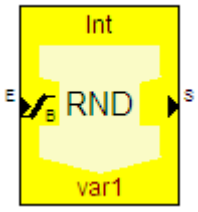
Na figura acima está representado um pequeno exemplo de uso do bloco de deslocamento com carry à direita. No caso, é efetuada a operação SRC em uma posição na variável var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for energizada (já que a entrada E do bloco de atribuição está selecionada por borda). O resultado da operação é guardado em var2. Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação SRC com deslocamento unitário irá resultar no valor 1749 (6D5h) na variável var2, caso o nodo de Carry In seja mantido desligado. Já se ele estiver acionado irá entrar um bit MSB igual a um, e o resultado será -31019 (86D5h).

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco, assim como o nodo CO. A entrada E do bloco SRA está selecionada para acionamento por borda e, portanto, a saída S e CO ficarão ativas durante apenas um ciclo de execução do programa aplicativo (<500µs). Então, foram inseridos monoestáveis para aumentar os pulsos de saída para um segundo, de forma a serem visíveis.

Veja também:
ARITMÉTICA INTEIRA - Operação SLC
ARITMÉTICA INTEIRA - Operação SLA
ARITMÉTICA INTEIRA - Operação SRA
ARITMÉTICA WORD - Operação SLC
ARITMÉTICA WORD - Operação SRC

ARITMÉTICA INTEIRA - Operação Randômica

Este bloco gera um valor randômico inteiro no operando especificado.



Resultado é o operando cujo valor será modificado de forma randômica pelo Controlador

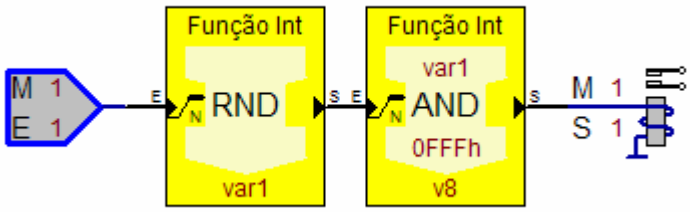
µDX200.

Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Caso o nodo de Entrada (E) esteja sensível ao nível, ao energizá-lo o µDX200 irá gerar um novo número randômico a cada ciclo de execução do programa aplicativo (o que pode atingir milhares de vezes por segundo). Se o nodo de Entrada (E) estiver selecionado para ser sensível a borda um novo número aleatório será gerado a cada borda de subida do nodo E.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

Este bloco pode ser útil para introduzir um componente de aleatoriedade em um programa, evitando assim laços infinitos (dead-lock) em programas realimentados.



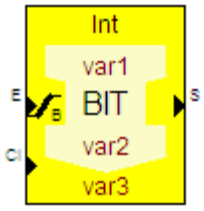
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação Randômica.dxg**

O programa acima gera números inteiros aleatórios entre 0 e 4095 (limites de operação das saídas analógicas de 12 bits) na saída analógica S1. Com isso obtêm-se um sinal de ruído branco nesta saída sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver energizada (já que a entrada E do bloco de atribuição está selecionada por nível). Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco.

Veja também:
ARITMÉTICA INTEIRA - Operação Randômica Var
ARITMÉTICA WORD - Operação Randômica
ARITMÉTICA WORD - Operação Randômica Var

ARITMÉTICA INTEIRA - Operação BIT

Este bloco acessa o bit do Operando1 apontado por Operando2, faz com que ele assuma o estado do nodo de entrada CI, e Resultado recebe o resultado da operação. Observe que os demais bits que compõem Operando1 não são afetados pela operação BIT. Ou seja, este bloco permite transferir o estado de um nodo (CI) para determinado bit de uma variável inteira.



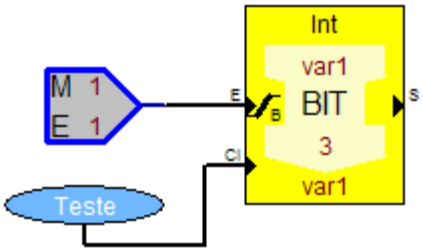
Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos

em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com os operandos expressos em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 0000h a 0FFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A30h e não A30h (pois o µDX200 entenderia que se trata da variável chamada A30H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. O nodo CI é o nodo que irá determinar o estado do bit apontado em Operando1 (ligado ou desligado).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



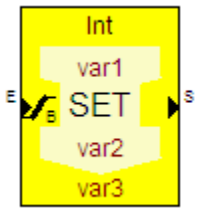
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação BIT.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação BIT inteira. No caso, é transferido o valor do nodo Teste para o bit 3 de var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for acionada (já que a entrada E do bloco de BIT está selecionada por borda). O resultado da operação é guardado na própria variável var1. Por exemplo, se var1 possui valor inicial zero e o nodo Teste está ligado, ao efetuar a operação BIT irá resultar no valor 8 (0000000000001000b em binário) na variável var1.

Veja também:
ARITMÉTICA INTEIRA - Operação SET
ARITMÉTICA INTEIRA - Operação RESET

ARITMÉTICA INTEIRA - Operação SET

Este bloco acessa o bit do Operando1 apontado por Operando2, faz com que ele assuma o estado 1 (ligado), e Resultado recebe o resultado da operação. Observe que os demais bits que compõem Operando1 não são afetados pela operação SET. Ou seja, este bloco permite ligar determinado bit de uma variável inteira. O bloco é similar ao bloco de Operação BIT, só que em vez de transferir o estado de um nodo para o bit apontado na variável, este bloco força que o bit apontado assuma valor 1.

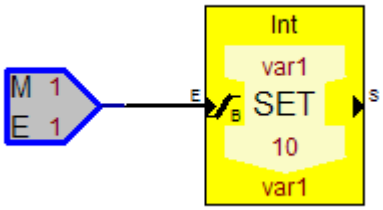


Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com os operandos expressos em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 0000h a 0FFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A30h e não A30h (pois o µDX200 entenderia que se trata da variável chamada A30H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



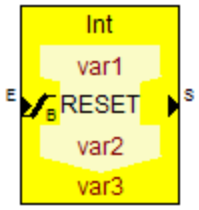
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação SET.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação SET inteira. No caso, é ligado o bit 10 de var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for acionada (já que a entrada E do bloco de SET está selecionada por borda). O resultado da operação é guardado na própria variável var1. Por exemplo, se var1 possui valor inicial zero ao efetuar a operação SET irá resultar no valor 1024 (0400h em hexadecimal; ou 0000010000000000b em binário) na variável var1.

Veja também:
ARITMÉTICA INTEIRA - Operação BIT
ARITMÉTICA INTEIRA - Operação RESET

ARITMÉTICA INTEIRA - Operação RESET

Este bloco acessa o bit do Operando1 apontado por Operando2, faz com que ele assuma o estado 0 (desligado), e Resultado recebe o resultado da operação. Observe que os demais bits que compõem Operando1 não são afetados pela operação RESET. Ou seja, este bloco permite desligar determinado bit de uma variável inteira. O bloco é similar ao bloco de Operação BIT, só que em vez de transferir o estado de um nodo para o bit apontado na variável, este bloco força que o bit apontado assuma valor 0.

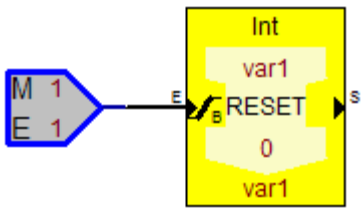


Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com os operandos expressos em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 0000h a 0FFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A30h e não A30h (pois o µDX200 entenderia que se trata da variável chamada A30H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação RESET.dxg**

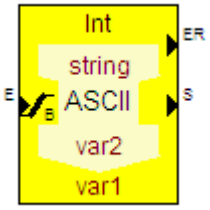
Na figura acima está representado um pequeno exemplo de uso do bloco de operação RESET inteira. No caso, é desligado o bit 0 de var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for acionada (já que a entrada E do bloco de RESET está selecionada por borda). O resultado da operação é guardado na própria variável var1. Por exemplo, se var1 possui valor inicial 7 (0000000000000111b em binário) ao efetuar a operação RESET irá resultar no valor 6 (0000000000000110b em binário) na variável var1.

Veja também:
ARITMÉTICA INTEIRA - Operação BIT

ARITMÉTICA INTEIRA - Operação SET

ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro

Este bloco converte um string apontado pelo bloco em um valor inteiro. É possível especificar um offset inicial para o string, e um caracter de finalização do string.



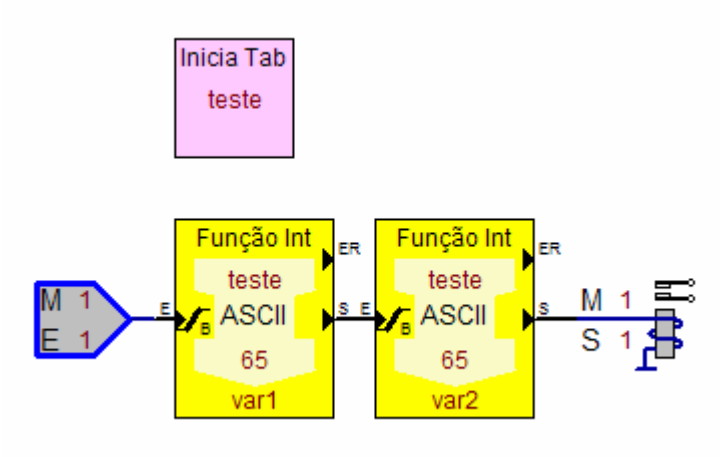
String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Offset indica a partir de qual posição do string é iniciada a conversão. Já Resultado recebe o valor convertido, e Caracter especifica um caracter de finalização para a conversão do string.

Note que String e Offset podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Já Caracter deve ser necessariamente uma constante, pois trata-se de tipo byte (que não é tratado pelo µDX200, sendo admitido apenas como constante).

Caso seja especificado Offset = -1 o bloco inicia a conversão a partir do ponto do string em que o bloco anterior de Conversão ASCIIàInteiro parou. Isso é útil para analisar strings com vários valores numéricos encadeados.

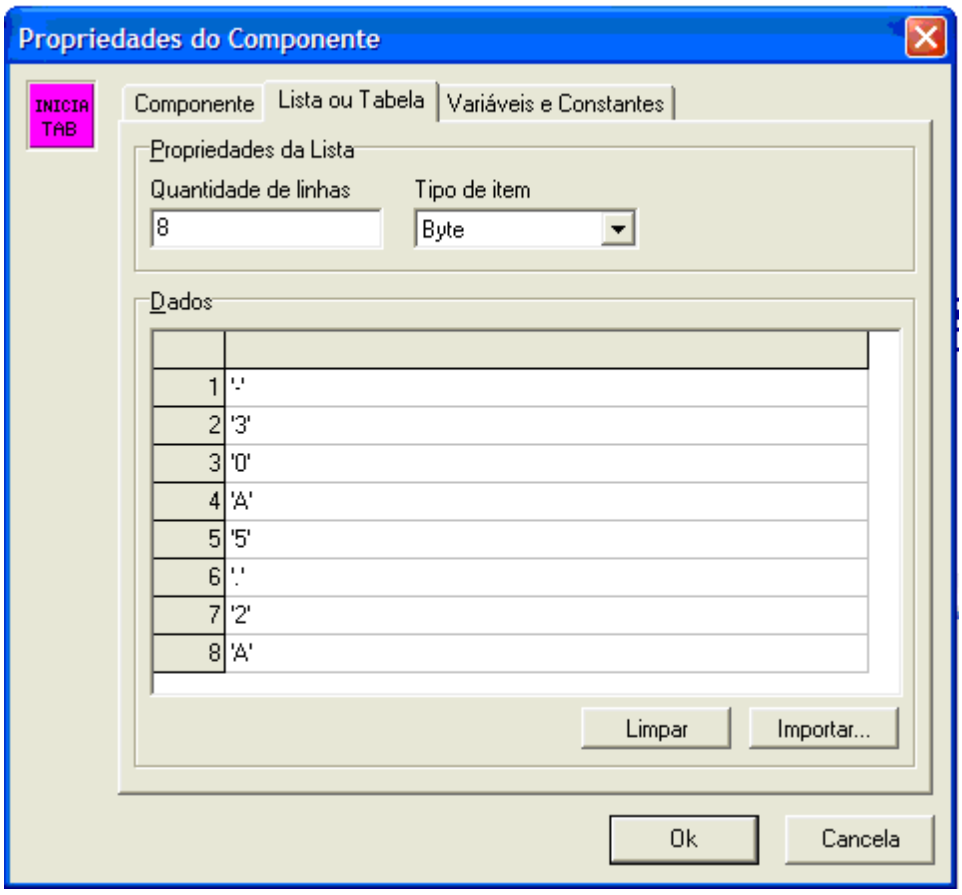
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

O nodo de Erro (ER) indica ter havido problemas na conversão. Por exemplo, foi encontrado um caracter não válido no string. Os caracteres válidos são os numerais (de '0' a '9') representados em ASCII (30h a 39h), vírgula ou ponto (',' ou '.'), que são desconsiderados já que é feita a conversão para inteiro (sem ponto decimal), ou o caracter de terminação especificado. Ainda é permitido o uso de sinal de menos ('-') para indicar números negativos.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Conversão ASCII - Inteiro.dxg**

O programa de exemplo acima converte dois valores numéricos representados no string teste para as variáveis inteiras var1 e var2. Note que utilizou-se Offset =0 para o bloco à esquerda (de forma que ele inicie a conversão a partir da primeira posição do string teste), e Offset = -1 para o bloco à direita (de forma que este bloco reinicie a conversão onde o primeiro bloco parou). Os valores inicializados no string teste via bloco de Inicia Tab são:



Note que colocando caracteres entre apóstrofes o µDX200 entende que deve armazenar a representação ASCII do caracter no byte. Assim, foram usados caracteres 'A' para caracter de terminação (cujo valor em ASCII é 65). Portanto, o string teste é inicializado com teste = "-30A5.2A".

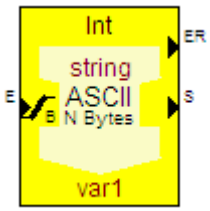
Ao rodar este programa e energizar a entrada E1 do módulo de Expansão M1 resulta em var1 =

-30 e var2 = 52.

Veja também:
ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro N Bytes
ARITMÉTICA INTEIRA - Conversão Caracter -> Inteiro
ARITMÉTICA LONGINT - Conversão ASCII -> LongInt
ARITMÉTICA LONGINT - Conversão ASCII -> LongInt N Bytes

ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro N Bytes

Este bloco converte um string apontado pelo bloco em um valor inteiro. É possível especificar um offset inicial para o string, e em vez do caracter de terminação especificado no bloco anterior, neste bloco é especificado o número de caracteres a serem analisados.



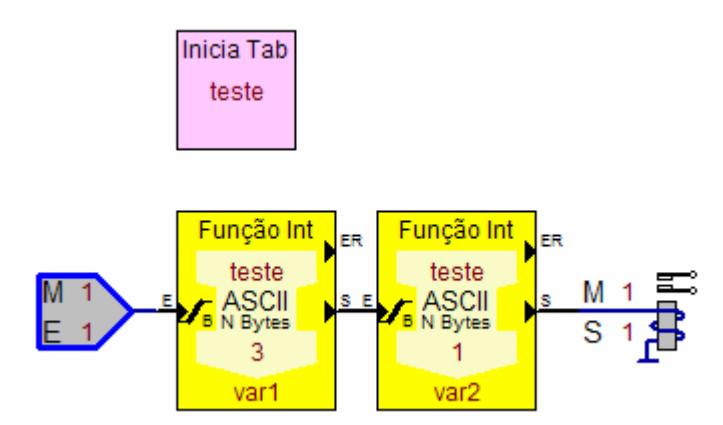
String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Offset indica a partir de qual posição do string é iniciada a conversão. Já Resultado recebe o valor convertido, e Núm.Bytes especifica quantos caracteres a partir de Offset serão analisados.

Note que String, Offset e Núm.Bytes podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Caso seja especificado Offset = -1 o bloco inicia a conversão a partir do ponto do string em que o bloco anterior de Conversão ASCIIàInteiro parou. Isso é útil para analisar strings com vários valores numéricos encadeados.

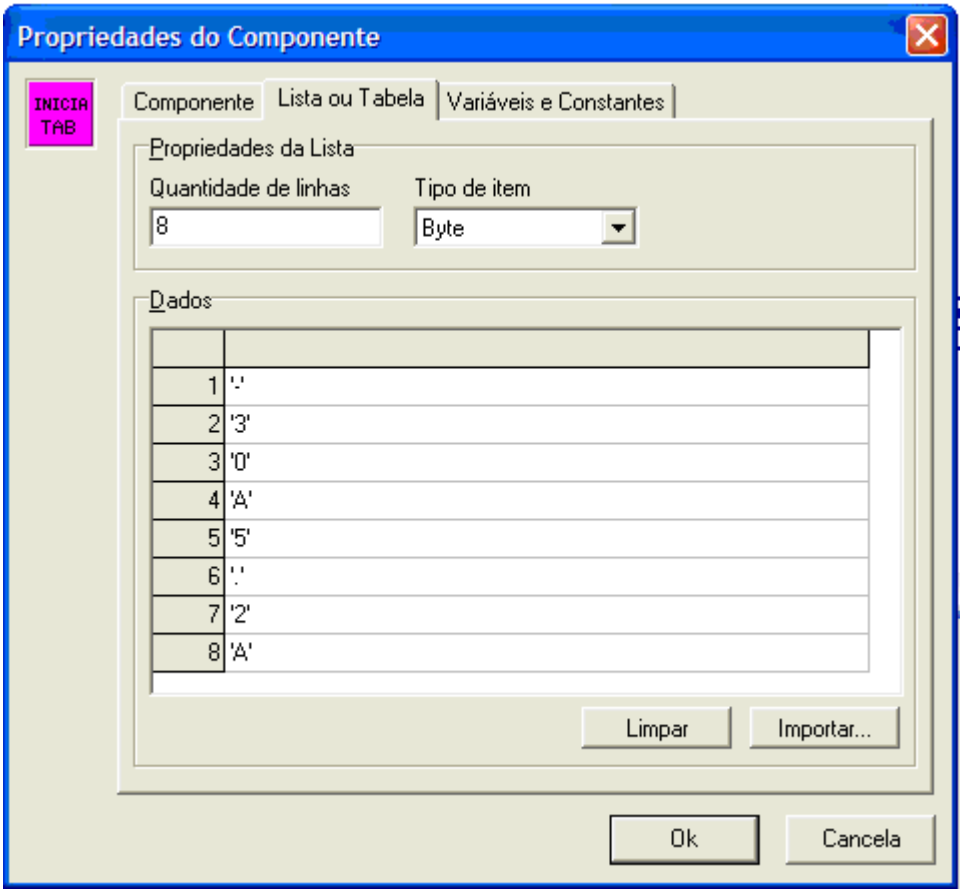
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

O nodo de Erro (ER) indica ter havido problemas na conversão. Por exemplo, foi encontrado um caracter não válido no string. Os caracteres válidos são os numerais (de '0' a '9') representados em ASCII (30h a 39h), vírgula ou ponto (',' ou '.'), que são desconsiderados já que é feita a conversão para inteiro (sem ponto decimal). Ainda é permitido o uso de sinal de menos ('-') para indicar números negativos.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Conversão ASCII - Inteiro N Bytes.dxg**

O programa de exemplo acima converte dois valores numéricos representados no string teste para as variáveis inteiras var1 e var2. Note que utilizou-se Offset =0 para o bloco à esquerda (de forma que ele inicie a conversão a partir da primeira posição do string teste), e Offset = 6 para o bloco à direita (de forma que este bloco inicie a conversão a partir da sétima posição do string teste). Os valores inicializados no string teste via bloco de Inicia Tab são:



Note que colocando caracteres entre apóstrofes o µDX200 entende que deve armazenar a representação ASCII do caracter no byte. Foram mantidos caracteres 'A' no string teste apenas para mantê-lo igual ao do exemplo anterior, mas agora tais caracteres não têm significado, já que o bloco analisa um número de caracteres do string e não até um carcter de terminação. Portanto, o string teste é inicializado com teste = "-30A5.2A".

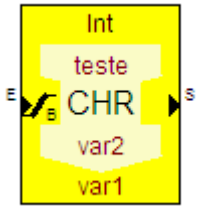
Ao rodar este programa e energizar a entrada E1 do módulo de Expansão M1 resulta em var1 =

-30 e var2 = 2.

Veja também:
ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro
ARITMÉTICA INTEIRA - Conversão Caracter -> Inteiro
ARITMÉTICA LONGINT - Conversão ASCII -> LongInt
ARITMÉTICA LONGINT - Conversão ASCII -> LongInt N Bytes

ARITMÉTICA INTEIRA - Conversão Caracter -> Inteiro

Os blocos anteriores convertiam uma representação ASCII de um número existente em um string em um valor inteiro. Este bloco converte um caracter ASCII em um número inteiro de valor igual ao valor ASCII correspondente a sua representação. Por exemplo, o caracter 'B' possui representação na tabela ASCII igual a 66. Então, se for apontado um caracter 'B' no string o bloco irá retornar o valor 66 em Resultado.

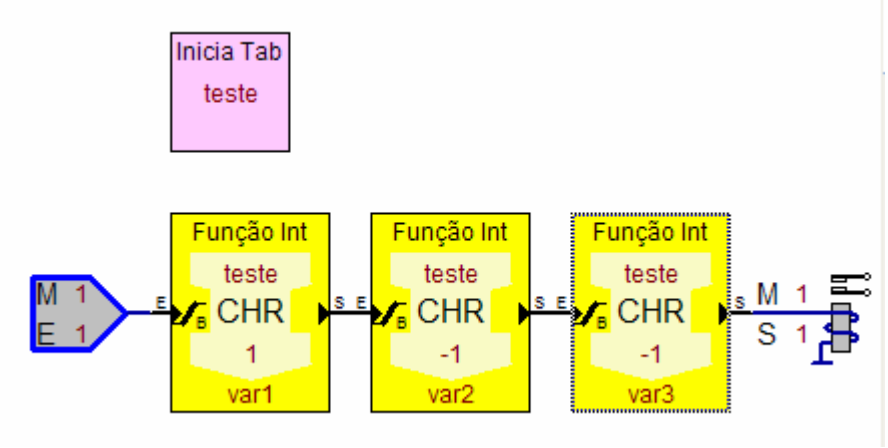


String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Offset indica a partir de qual posição do string é feita a conversão. Já Resultado recebe o valor convertido.

Note que String e Offset podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

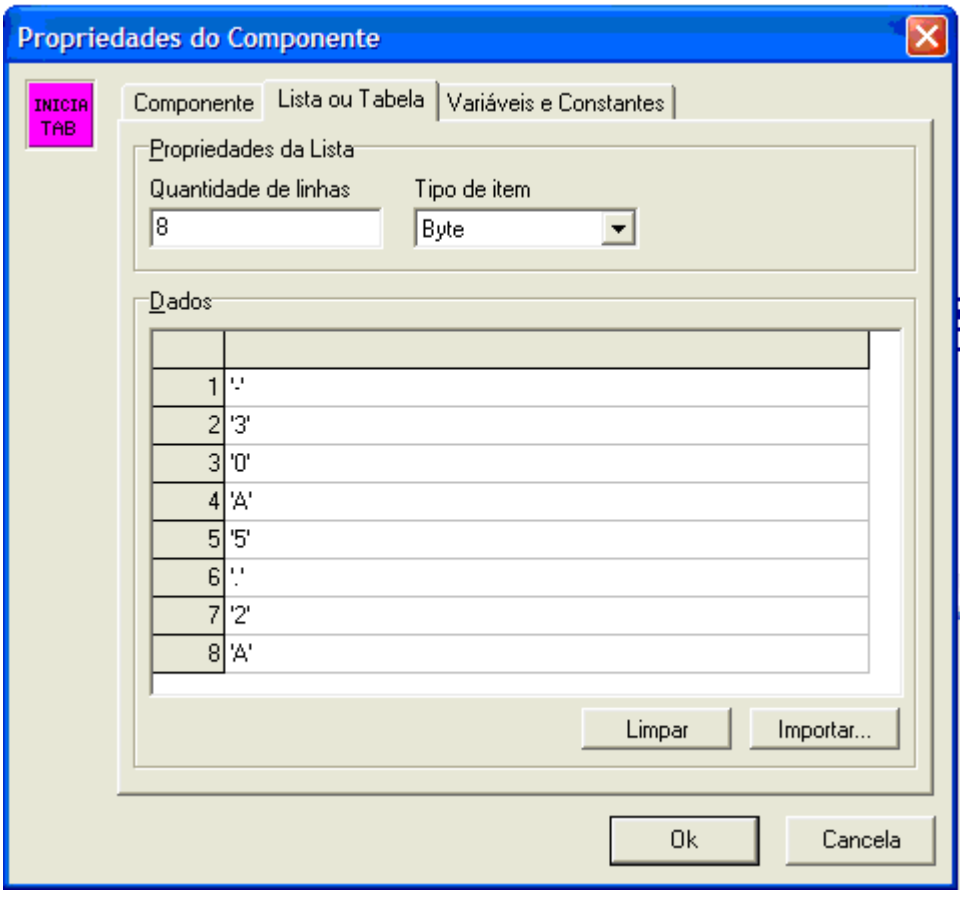
Caso seja especificado Offset = -1 o bloco inicia a conversão a partir do ponto do string em que o bloco anterior de Conversão CaracteràInteiro parou. Isso é útil para analisar vários caracteres encadeados no string.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Conversão Caracter - Inteiro.dxg**

O programa de exemplo acima converte os três primeiros caracteres do string teste para suas representações numéricas em ASCII. Note que utilizou-se Offset =0 para o bloco à esquerda (de forma que ele inicie a conversão a partir da primeira posição do string teste), e Offset = -1 para os demais blocos à direita (de forma que estes blocos iniciem a conversão a partir da posição do bloco anterior). Os valores inicializados no string teste via bloco de Inicia Tab são:



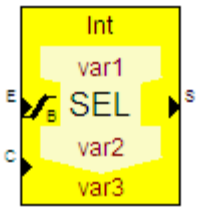
Note que colocando caracteres entre apóstrofes o µDX200 entende que deve armazenar a representação ASCII do caracter no byte. Portanto, o string teste é inicializado com teste = "-30A5.2A".

Ao rodar este programa e energizar a entrada E1 do módulo de Expansão M1 resulta em var1 = 51, var2 = 48, e var3 = 65, o que corresponde aos valores numéricos em ASCII dos caracteres '3', '0' e 'A'.

Veja também:
ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro
ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro N Bytes
ARITMÉTICA LONGINT - Conversão ASCII -> LongInt
ARITMÉTICA LONGINT - Conversão ASCII -> LongInt N Bytes

ARITMÉTICA INTEIRA - Seletor

Este bloco transfere o valor de um ou outro operando inteiro para um terceiro operando inteiro.



Operando1 e Operando2 são os operandos que serão transferidos para Resultado, conforme o estado do nodo de Controle (C).

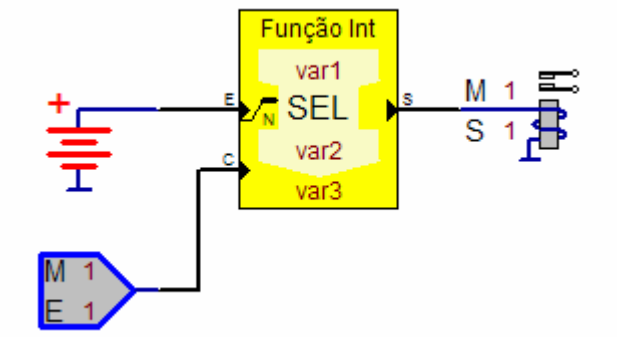
Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

O nodo de Controle (C) especifica qual dos operandos será transferido para Resultado. Se C=0 o Operando1 é transferido para Resultado. Já se C=1 o Operando2 é transferido para Resultado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Seletor.dxg**

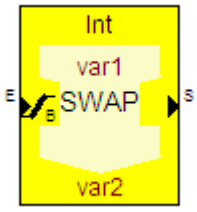
Acima temos um pequeno exemplo de uso do bloco de seleção inteira. No caso, é atribuído o valor da variável var1 ou da variável var2 na variável inteira var3, conforme o estado do nodo de Controle (C). O nodo de Entrada (E) está sempre ativado, pois está selecionado para acionamento por nível e está ligado constantemente pelo bloco de energia. O nodo de Controle (C) é comando pela entrada E1 do módulo 1 de Expansão de Entradas/Saídas (µDX210).

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição inteira. A entrada E do bloco de atribuição está selecionada para acionamento por nível e sempre ligada. Então, a saída S ficará ativa sempre.

Veja também:
ARITMÉTICA LONGINT - Seletor
ARITMÉTICA WORD - Seletor
ARITMÉTICA REAL - Seletor
GERAL - Seletor de Variável Inteira
GERAL - Seletor de Variável LongInt
GERAL - Seletor de Variável Word
GERAL - Seletor de Variável Real

ARITMÉTICA INTEIRA - Operação SWAP

Este bloco permuta o byte mais significativo e o byte menos significativo de um operando inteiro, e coloca o resultado em um segundo operando inteiro.



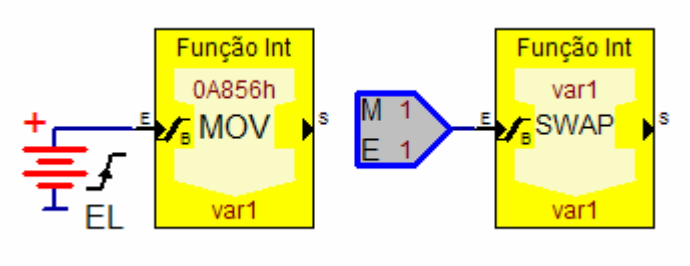
Operando é o operando cujo valor será permutado e transferido para Resultado.

Note que Operando pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



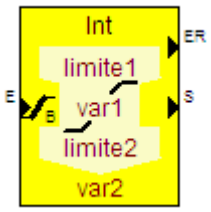
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação SWAP.dxg**

Acima temos um pequeno exemplo de uso do bloco de SWAP inteiro. No caso, é atribuído o valor A856h na variável inteira var1 ao energizar o controlador programável. Sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for energizada a variável var1 troca o byte MSB pelo LSB e vice-versa. Ou seja, na primeira energização de E1 a variável var1 irá assumir valor 56A8h. Na próxima energização de E1 irá retornar ao valor A856h, e assim sucessivamente. Como foi usado um bloco de Nodo EL para acionar a inicialização da variável var1, é preciso desenergizar momentaneamente o controlador para que a variável inicialize com valor A856h. Caso se queira que a variável inicialize ao rodar o programa aplicativo substitua o bloco de Nodo EL por um bloco de Nodo Reset.

Veja também:
ARITMÉTICA INTEIRA - Operação SWAP Var
ARITMÉTICA LONGINT - Operação SWAP
ARITMÉTICA LONGINT - Operação SWAP Var

ARITMÉTICA INTEIRA - Operação Limite

Permite limitar uma variável inteira entre dois valores especificados. Assim, se garante que a variável assumirá apenas valores entre os dois limites. Este bloco somente é operacional em controladores µDX201 versão 3.37 ou superior.



Operando é a variável cujo valor será limitado entre os limites estabelecidos por Lim.Superior e Lim.Inferior e transferido para Resultado.

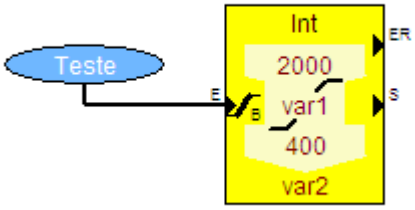
Note que Operando, Lim.Superior e Lim.Inferior podem ser variáveis inteiras, constantes (valor numérico em decimal ou hexadecimal), ou ainda referências a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Variáveis inteiras podem assumir valores entre -32768 e 32767.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que Operando possui valor fora dos limites impostos por Lim.Superior e Lim.Inferior.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

O exemplo abaixo transfere o valor de Var1 para Var2, limitando-o entre 400 e 2000, sempre que o nodo Teste é acionado:



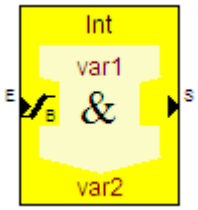
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação Limite.dxg**

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Limite Var
ARITMÉTICA LONGINT - Operação Limite
ARITMÉTICA LONGINT - Operação Limite Var
ARITMÉTICA WORD - Operação Limite
ARITMÉTICA WORD - Operação Limite Var
ARITMÉTICA REAL - Operação Limite
ARITMÉTICA REAL - Operação Limite Var

ARITMÉTICA INTEIRA - Operação Pointer

Este bloco retorna na variável inferior o endereço de memória da variável especificada. Este bloco somente é operacional em controladores µDX201 versão 3.52 ou superior.



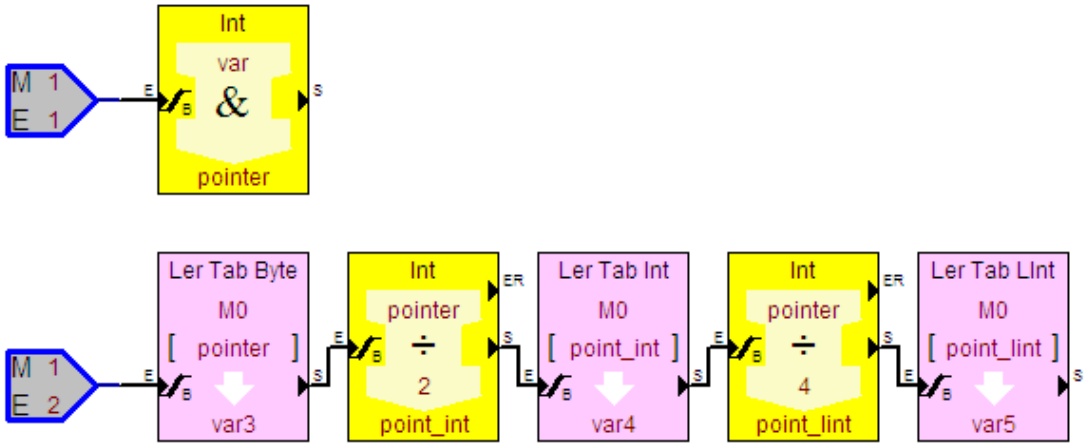
Operando1 é a variável cujo endereço será transferido para Resultado.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco

só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

O exemplo abaixo transfere o endereço da variável inteira **var** para a variável inteira **pointer** quando a entrada **E1** da Expansão 1 é acionada. Ao acionar a entrada **E2** dessa expansão é lido o endereço de memória **M0** (endereço 0 de memória) indexado por **pointer**, que aponta para o endereço de **var**. O byte de resultado é transferido para a variável inteira var3. Ou seja. o byte **LSB** de **var** é transferido para **var3**. A seguir divide-se **pointer** por 2, já que a leitura de tabela inteira pula de 2 em 2 bytes, e se transfere a posição de memória **M0** (endereço 0 de memória) indexado por **pointer / 2** para a variável **var4**. Nesse caso, todo o valor de **var** é transferido para **var4**. Por fim, divide-se pointer por 4 porque a leitura de tabela longint pula de 4 em 4 bytes, e se transfere a posição de memória **M0** (endereço 0 de memória) indexado por **pointer / 4** para a variável **var5**. O valor de **var** e da variável inteira subseqüente (4 bytes) é transferido para a variável **var5**.



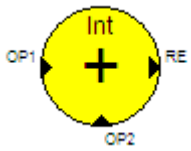
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação Pointer.dxg**

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.52 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Pointer Var
ARITMÉTICA WORD - Operação Pointer
ARITMÉTICA WORD - Operação Pointer Var

ARITMÉTICA INTEIRA - Adição Var

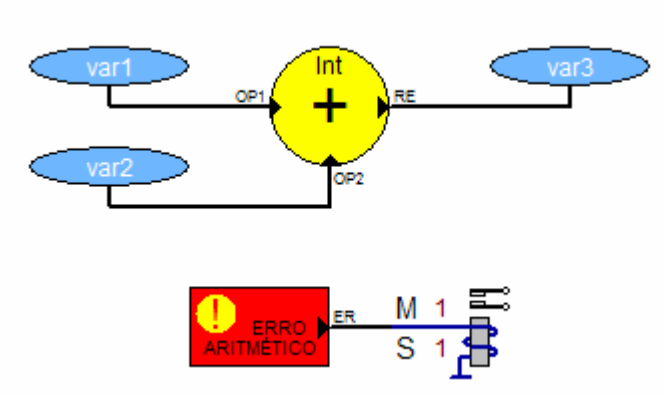
Este bloco efetua a adição de dois operandos inteiros, colocando o resultado em um terceiro operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos inteiros a serem adicionados e RE recebe o resultado da adição.

Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis inteiras podem assumir valores entre -32768 e 32767. Assim, por exemplo, se somarmos duas variáveis, ambas com valor 30000 o resultado será excessivo (60000) para representação via inteiros e ocorrerá um erro. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética Inteira - Adição Var.dxg**

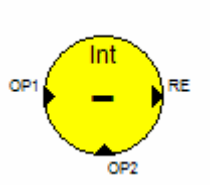
Acima temos um pequeno exemplo de uso do bloco de adição var inteira. No caso, a variável var1 é somada a variável var2 constantemente, e o resultado da operação é colocado em var3. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de adição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA INTEIRA - Adição
ARITMÉTICA LONGINT - Adição
ARITMÉTICA LONGINT - Adição Var
ARITMÉTICA WORD - Adição
ARITMÉTICA WORD - Adição Var
ARITMÉTICA REAL - Adição
ARITMÉTICA REAL - Adição Var

ARITMÉTICA INTEIRA - Subtração Var

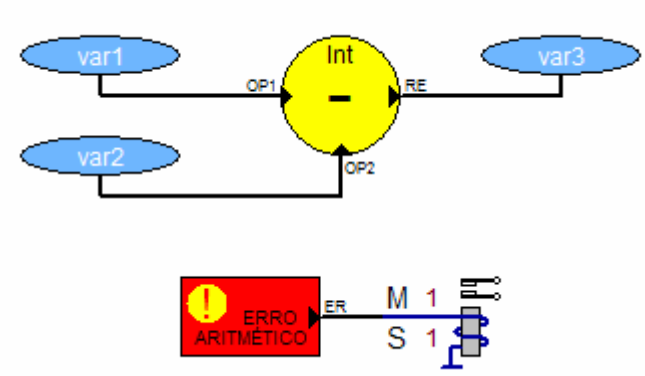
Este bloco efetua a subtração de dois operandos inteiros, colocando o resultado em um terceiro operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos inteiros a serem subtraídos e RE recebe o resultado da subtração.

Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis inteiras podem assumir valores entre -32768 e 32767. Assim, por exemplo, se subtrairmos duas variáveis, uma de valor -30000 e outr de valor 30000 o resultado será excessivo (-60000) para representação via inteiros e ocorrerá um erro. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética Inteira - Subtração Var.dxg**

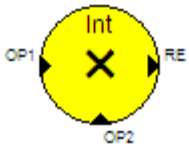
Acima temos um pequeno exemplo de uso do bloco de subtração var inteira. No caso, a variável var2 é subtraída da variável var1 constantemente, e o resultado da operação é colocado em var3. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de subtração pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA INTEIRA - Subtração
ARITMÉTICA LONGINT - Subtração
ARITMÉTICA LONGINT - Subtração Var
ARITMÉTICA WORD - Subtração
ARITMÉTICA WORD - Subtração Var
ARITMÉTICA REAL - Subtração
ARITMÉTICA REAL - Subtração Var

ARITMÉTICA INTEIRA - Multiplicação Var

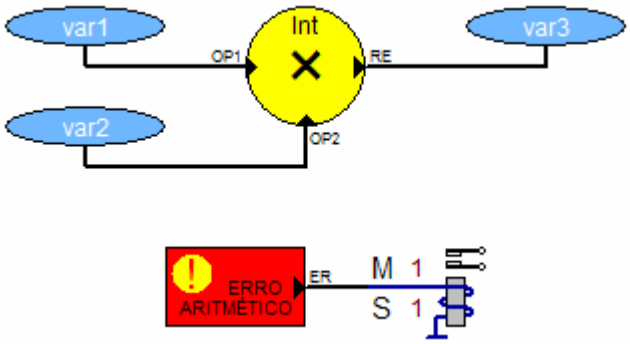
Este bloco efetua a multiplicação de dois operandos inteiros, colocando o resultado em um terceiro operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos inteiros a serem multiplicados e RE recebe o resultado da multiplicação.

Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis inteiras podem assumir valores entre -32768 e 32767. Assim, por exemplo, se multiplicarmos duas variáveis, ambas de valor 200 o resultado será excessivo (40000) para representação via inteiros e ocorrerá um erro. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética Inteira - Multiplicação Var.dxg**

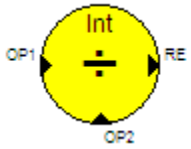
Acima temos um pequeno exemplo de uso do bloco de multiplicação var inteira. No caso, a variável var1 é multiplicada com a variável var2 constantemente, e o resultado da operação é colocado em var3. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de multiplicação pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA INTEIRA - Multiplicação
ARITMÉTICA LONGINT - Multiplicação
ARITMÉTICA LONGINT - Multiplicação Var
ARITMÉTICA REAL - Multiplicação
ARITMÉTICA REAL - Multiplicação Var

ARITMÉTICA INTEIRA - Divisão Var

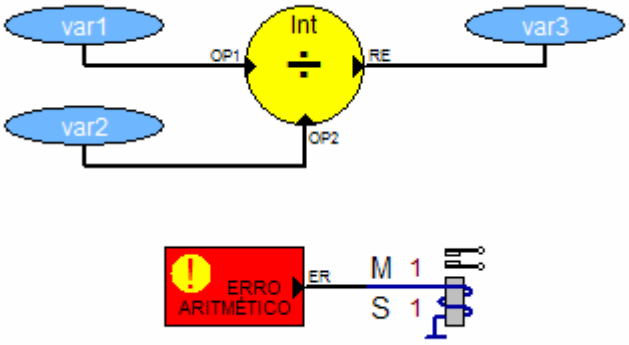
Este bloco efetua a divisão de dois operandos inteiros, colocando o resultado em um terceiro operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos inteiros a serem divididos e RE recebe o resultado da divisão.

Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Este bloco pode ocasionar um erro aritmético se OP2 for igual a zero. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética Inteira - Divisão Var.dxg**

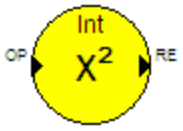
Acima temos um pequeno exemplo de uso do bloco de divisão var inteira. No caso, a variável var1 é dividida pela variável var2 constantemente, e o resultado da operação é colocado em var3. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de divisão pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA INTEIRA - Divisão
ARITMÉTICA LONGINT - Divisão
ARITMÉTICA LONGINT - Divisão Var
ARITMÉTICA REAL - Divisão
ARITMÉTICA REAL - Divisão Var

ARITMÉTICA INTEIRA - Quadrado Var

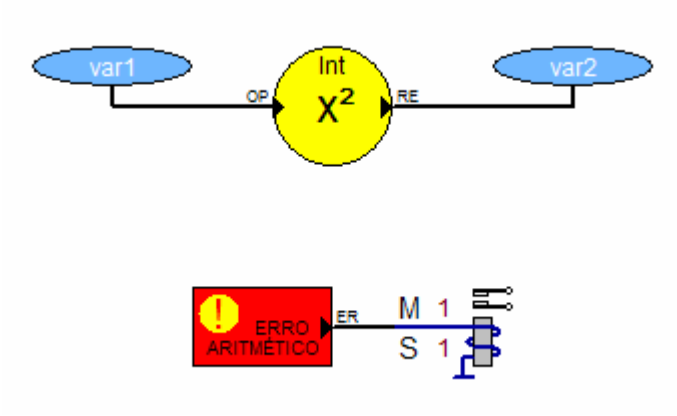
Este bloco eleva ao quadrado um operando inteiro, colocando o resultado em um segundo operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP é os operando inteiro a ser elevado ao quadrado e RE recebe o resultado da operação.

Note que OP pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis inteiras podem assumir valores entre -32768 e 32767. Assim, por exemplo, se elevarmos ao quadrado um valor superior a 181 ou inferior a -181 o resultado será excessivo para representação via inteiros e ocorrerá um erro. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética Inteira - Quadrado Var.dxg**

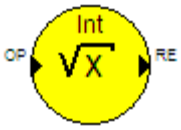
Acima temos um pequeno exemplo de uso do bloco de quadrado var inteiro. No caso, a variável var1 é elevada ao quadrado constantemente, e o resultado da operação é colocado em var2. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de quadrado pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA INTEIRA - Quadrado
ARITMÉTICA LONGINT - Quadrado
ARITMÉTICA LONGINT - Quadrado Var
ARITMÉTICA REAL - Quadrado
ARITMÉTICA REAL - Quadrado Var

ARITMÉTICA INTEIRA - Raiz Quadrada Var

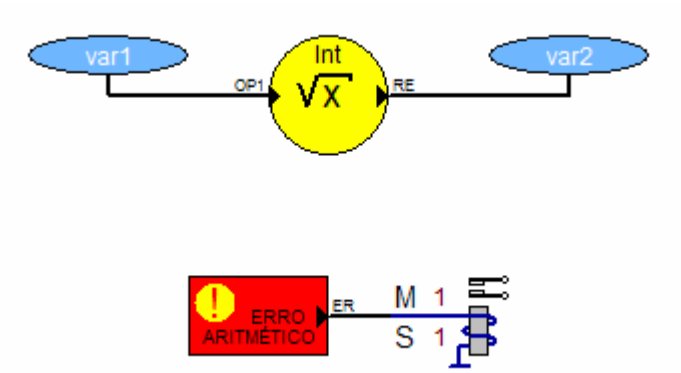
Este bloco calcula a raiz quadrada de um operando inteiro, colocando o resultado em um segundo operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é o operando inteiro em que será efetuada a operação de raiz quadrada e RE recebe o resultado da operação.

Note que OP1 pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis inteiras podem assumir valores entre -32768 e 32767. Mas tentar retirar uma raiz quadrada de número negativo irá gerar um erro aritmético. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética Inteira - Raiz Quadrada Var.dxg**

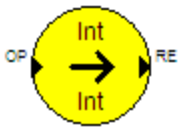
Acima temos um pequeno exemplo de uso do bloco de raiz quadrada var inteira. No caso, é retirada a raiz quadrada da variável var1 constantemente, e o resultado da operação é colocado em var2. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de raiz quadrada pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA INTEIRA - Raiz Quadrada
ARITMÉTICA LONGINT - Raiz Quadrada
ARITMÉTICA LONGINT - Raiz Quadrada Var
ARITMÉTICA REAL - Raiz Quadrada
ARITMÉTICA REAL - Raiz Quadrada Var

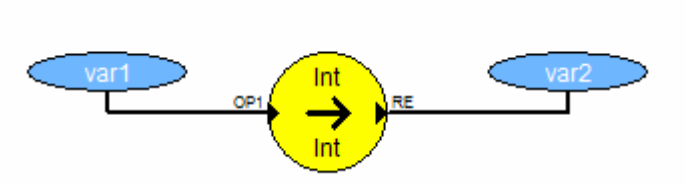
ARITMÉTICA INTEIRA - Atribuição Var

Este bloco transfere o valor de um operando inteiro para um segundo operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é o operando inteiro a transferido para RE.

Note que OP1 pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



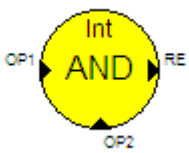
Exemplo de Programa Aplicativo: **Aritmética Inteira - Atribuição Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de atribuição var inteira. No caso, é transferido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

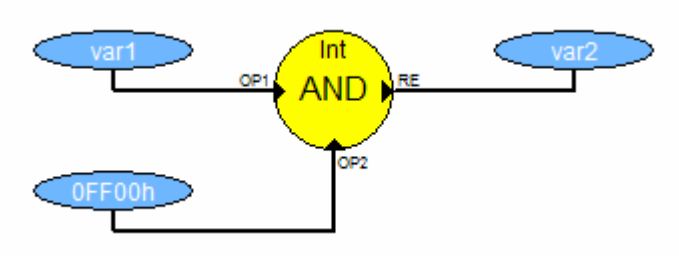
Veja também:
ARITMÉTICA INTEIRA - Atribuição
ARITMÉTICA LONGINT - Atribuição
ARITMÉTICA LONGINT - Atribuição Var
ARITMÉTICA WORD - Atribuição
ARITMÉTICA WORD - Atribuição Var
ARITMÉTICA REAL - Atribuição
ARITMÉTICA REAL - Atribuição Var

ARITMÉTICA INTEIRA - Operação AND Var

Este bloco efetua a operação booleana AND entre dois operandos inteiros, e RE recebe o resultado da operação. Observe que a operação AND é feita bit a bit com os 16 bits dos operandos. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



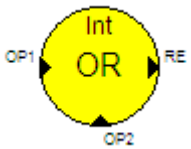
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação AND Var.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana AND inteira. No caso, é efetuada a operação AND entre a variável var1 e a constante hexadecimal 0FF00h constantemente. O resultado da operação é guardado em var2. Note que esta operação zera o byte LSB de var1 (já que qualquer valor AND com 00h resulta em zero). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação AND com 0FF00h irá resultar no valor 3328 (0D00h) na variável var2.

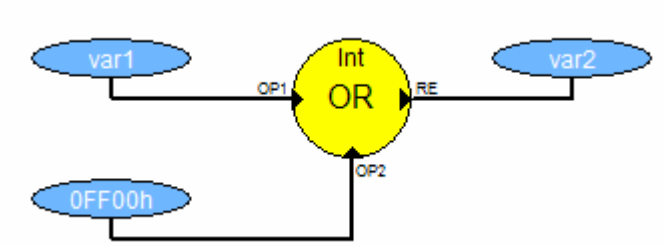
Veja também:
ARITMÉTICA INTEIRA - Operação AND
ARITMÉTICA LONGINT - Operação AND
ARITMÉTICA LONGINT - Operação AND Var
ARITMÉTICA WORD - Operação AND
ARITMÉTICA WORD - Operação AND Var

ARITMÉTICA INTEIRA - Operação OR Var

Este bloco efetua a operação booleana OR entre dois operandos inteiros, e RE recebe o resultado da operação. Observe que a operação OR é feita bit a bit com os 16 bits dos operandos. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação OR Var.dxg**

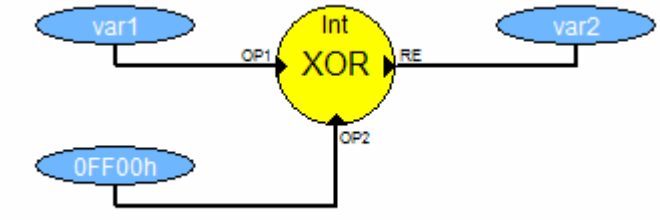
Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana OR inteira. No caso, é efetuada a operação OR entre a variável var1 e a constante hexadecimal 0FF00h constantemente. O resultado da operação é guardado em var2. Note que esta operação liga todos os bits do byte MSB de var1 (já que qualquer valor OR com 0FFh resulta em um). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação OR com 0FF00h irá resultar no valor -85 (0FFABh) na variável var2.

Veja também:
ARITMÉTICA INTEIRA - Operação OR
ARITMÉTICA LONGINT - Operação OR
ARITMÉTICA LONGINT - Operação OR Var
ARITMÉTICA WORD - Operação OR
ARITMÉTICA WORD - Operação OR Var

ARITMÉTICA INTEIRA - Operação XOR Var

Este bloco efetua a operação booleana XOR entre dois operandos inteiros, e RE recebe o resultado da operação. Observe que a operação XOR é feita bit a bit com os 16 bits dos operandos. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação XOR Var.dxg**

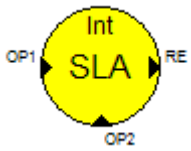
Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana

XOR inteira. No caso, é efetuada a operação XOR entre a variável var1 e a constante hexadecimal 0FF00h constantemente. O resultado da operação é guardado em var2. Note que esta operação inverte todos os bits do byte MSB de var1 (já que qualquer valor XOR com 0FFh resulta em inversão bit a bit). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação XOR com 0FF00h irá resultar no valor de -3413 (0F2ABh) na variável var2.

Veja também:
ARITMÉTICA INTEIRA - Operação XOR
ARITMÉTICA LONGINT - Operação XOR
ARITMÉTICA LONGINT - Operação XOR Var
ARITMÉTICA WORD - Operação XOR
ARITMÉTICA WORD - Operação XOR Var

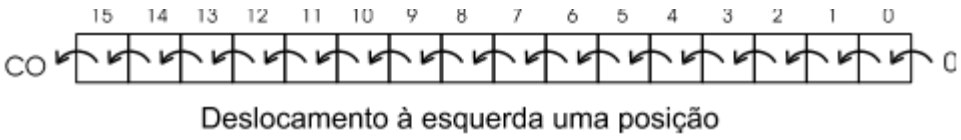
ARITMÉTICA INTEIRA - Operação SLA Var

Este bloco efetua a operação de deslocamento aritmético à esquerda do operando OP1 pelo número de deslocamentos indicado pelo operando OP2, e a conexão RE recebe o resultado da operação.

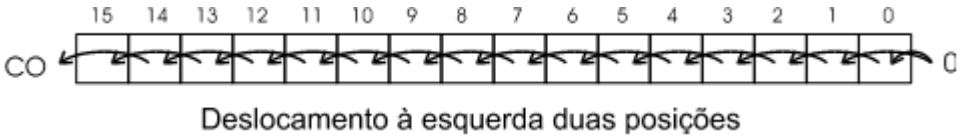


Observe que a operação SLA (shift left arithmetical) é feita bit a bit com os 16 bits do operando OP1, segundo o número de deslocamentos especificado pelo operando OP2. Assim, se OP2 é igual a um o operando OP1 será deslocado em uma posição para a esquerda (equivalente a multiplicar por dois). Já se OP2 é igual a dois o operando OP1 será deslocado em duas posições para a esquerda (equivalente a multiplicar por quatro), e assim sucessivamente.

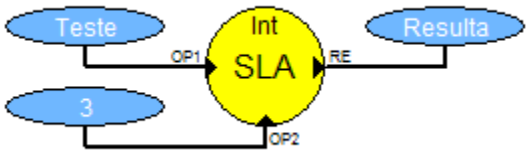
Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Note que os bits LSB são preenchidos com zeros, enquanto o bit MSB é perdido (uma vez que este bloco não disponibiliza o nodo CO, como faz o bloco retangular equivalente).



Deslocamento à esquerda uma posição



Deslocamento à esquerda duas posições



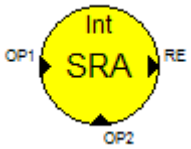
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação SLA Var.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de deslocamento aritmético à esquerda Var. No caso, é efetuada a operação SLA em três posições na variável Teste, e o resultado é colocado na variável Resulta. Note que esta operação equivale a multiplicar por oito o valor da variável Teste.

Veja também:
ARITMÉTICA INTEIRA - Operação SRA Var
ARITMÉTICA INTEIRA - Operação SLA
ARITMÉTICA INTEIRA - Operação SRA

ARITMÉTICA INTEIRA - Operação SRA Var

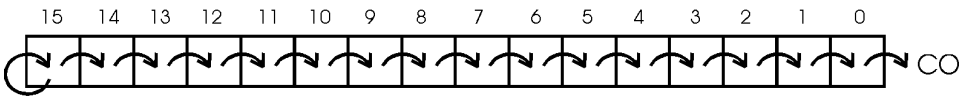
Este bloco efetua a operação de deslocamento aritmético à direita do operando OP1 pelo número de deslocamentos indicado pelo operando OP2, e a conexão RE recebe o resultado da operação.



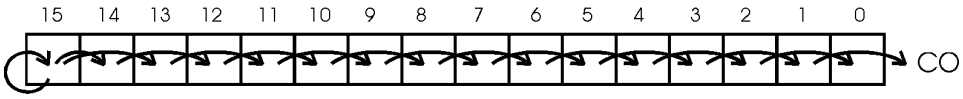
Observe que a operação SRA (shift right arithmetical) é feita bit a bit com os 16 bits do operando OP1, segundo o número de deslocamentos especificado pelo operando OP2. Assim, se OP2 é igual a um o operando OP1 será deslocado em uma posição para a direita (equivalente a dividir por dois). Já se OP2 é igual a dois o operando OP1 será deslocado em duas posições para a direita (equivalente a dividir por quatro), e assim sucessivamente.

Como se trata de deslocamento aritmético o bit mais significativo (bit 15) é preservado, de forma a manter o sinal do valor deslocado. Por exemplo, -32768 (8000h) deslocado uma posição à direita equivale a dividir por dois o valor original, ou seja, -16384 (C000h).

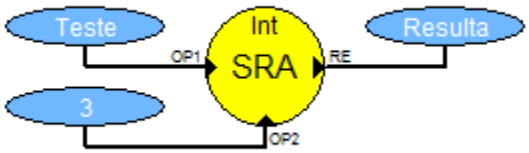
Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Note que os bits MSB são preenchidos com o valor do bit 15, enquanto o bit LSB é perdido (uma vez que este bloco não disponibiliza o nodo CO, como faz o bloco retangular equivalente).



Deslocamento à direita uma posição



Deslocamento à direita duas posições



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação SRA Var.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de deslocamento aritmético à direita Var. No caso, é efetuada a operação SRA em três posições na variável Teste e o resultado é fornecido na variável Resulta. Note que esta operação equivale a dividir por oito o valor da variável var1.

Veja também:
ARITMÉTICA INTEIRA - Operação SLA Var
ARITMÉTICA INTEIRA - Operação SLA
ARITMÉTICA INTEIRA - Operação SRA

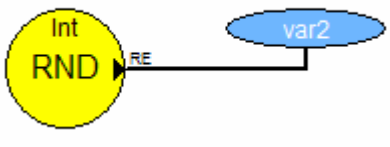
ARITMÉTICA INTEIRA - Operação Randômica Var

Este bloco gera um valor randômico inteiro no operando especificado.

RE é o operando cujo valor será modificado de forma randômica pelo Controlador µDX200. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Este bloco pode ser útil para introduzir um componente de aleatoriedade em um programa, evitando assim laços infinitos (dead-lock) em programas realimentados.



Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação Randômica Var.dxg**

O programa de exemplo acima gera números aleatórios continuamente e os transfere para a

variável var2. Note que estes números são gerados dentro da faixa admitida pelos números inteiros (-32768 a 32767).

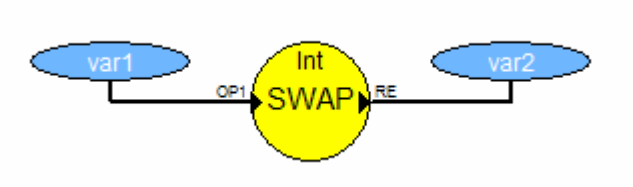
Veja também:
ARITMÉTICA INTEIRA - Operação Randômica
ARITMÉTICA WORD - Operação Randômica
ARITMÉTICA WORD - Operação Randômica Var

ARITMÉTICA INTEIRA - Operação SWAP Var

Este bloco permuta o byte mais significativo e o byte menos significativo de um operando inteiro para um segundo operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

OP1 é o operando inteiro a transferido para RE.

Note que OP1 pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



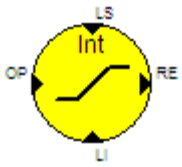
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação SWAP Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de SWAP inteiro var. No caso, é atribuído à variável var2 o valor de var1 com o byte MSB trocado pelo byte LSB e vice-versa. Por exemplo, se atribuirmos a var1 o valor hexadecimal 7C4Ah a variável var2 irá assumir o valor hexadecimal 4A7Ch.

Veja também:
ARITMÉTICA INTEIRA - Operação SWAP
ARITMÉTICA LONGINT - Operação SWAP
ARITMÉTICA LONGINT - Operação SWAP Var

ARITMÉTICA INTEIRA - Operação Limite Var

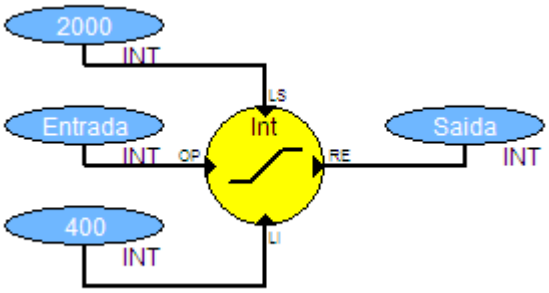
Permite limitar uma variável inteira entre dois valores especificados. Assim, se garante que a variável assumirá apenas valores entre os dois limites. Este bloco somente é operacional em controladores µDX201 versão 3.37 ou superior. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP (operando) é a variável cujo valor será limitado entre os limites estabelecidos por LS (limite superior) e LI (limite inferior) e transferido para RE (resultado).

Note que OP, LS e LI podem ser variáveis inteiras, constantes (valor numérico em decimal ou hexadecimal), ou ainda referências a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Variáveis inteiras podem assumir valores entre -32768 e 32767.

O exemplo a seguir limita o valor da variável Entrada entre os valores 400 e 2000, transferindo-o para a variável Saida:



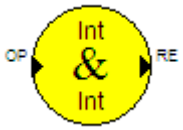
Exemplo de Programa Aplicativo: **Aritmética Inteira - Operação Limite Var.dxg**

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Limite
ARITMÉTICA LONGINT - Operação Limite
ARITMÉTICA LONGINT - Operação Limite Var
ARITMÉTICA WORD - Operação Limite
ARITMÉTICA WORD - Operação Limite Var
ARITMÉTICA REAL - Operação Limite
ARITMÉTICA REAL - Operação Limite Var

ARITMÉTICA INTEIRA - Operação Pointer Var

Permite retornar na conexão de saída o endereço na memória da variável especificada na conexão de entrada. Este bloco somente é operacional em controladores µDX201 versão 3.52 ou superior.



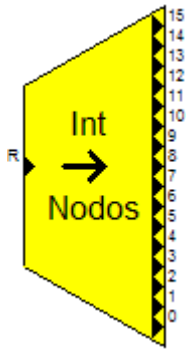
OP é a variável cujo endereço será transferido para RE.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.52 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Pointer
ARITMÉTICA WORD - Operação Pointer
ARITMÉTICA WORD - Operação Pointer Var

ARITMÉTICA INTEIRA - Conversão Int -> Nodos

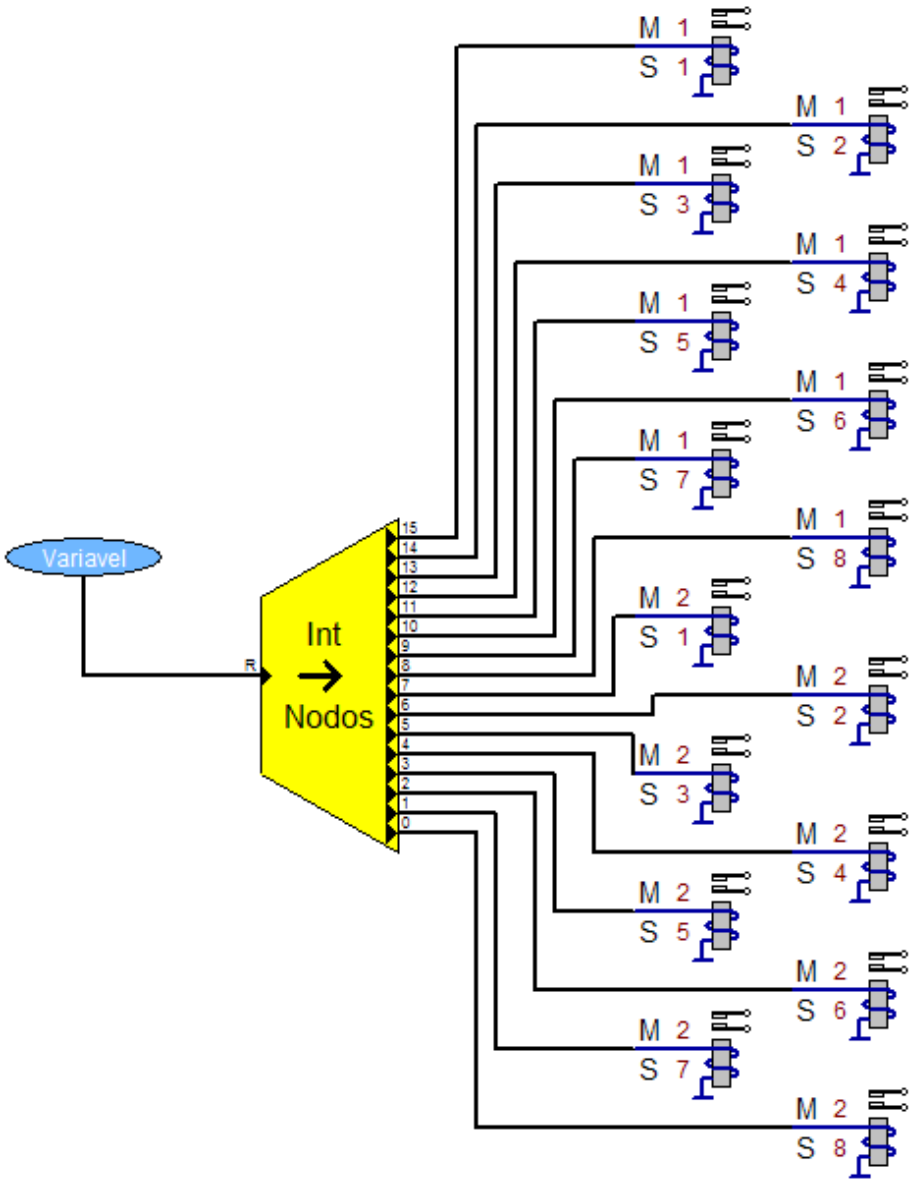
Este bloco permite separar os 16 bits de uma variável inteira em 16 nodos distintos. Com isso, conforme os bits da variável inteira estejam ligados ou desligados os nodos correspondentes serão acionados ou não.



No caso deste tipo de bloco a variável inteira é especificada na conexão R, ou seja, o fio de conexão transporta a variável (em vez de ser especificada literalmente via edição do bloco).

Note que R pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já as conexão de 0 a 15 à esquerda do bloco são nodos (ou seja, variáveis binárias - assumem apenas dois estados, ligado ou desligado).

A seguir um pequeno exemplo do uso deste bloco. Conforme o valor atribuído a variável Variavel os nodos são acionados ou não. Por exemplo, para Variavel = 10 apenas S5 e S7 da segunda Expansão µDX210 serão acionados, pois 10 em decimal é 0000 0000 0000 1010 em binário.

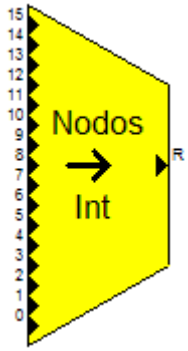


Exemplo de Programa Aplicativo: **Aritmética Inteira - Conversão Inteiro - Nodos.dxg**

Veja também:
ARITMÉTICA INTEIRA - Conversão Nodos -> Int
ARITMÉTICA WORD - Conversão Int -> Word
ARITMÉTICA WORD - Conversão Word -> Int

ARITMÉTICA INTEIRA - Conversão Nodos -> Int

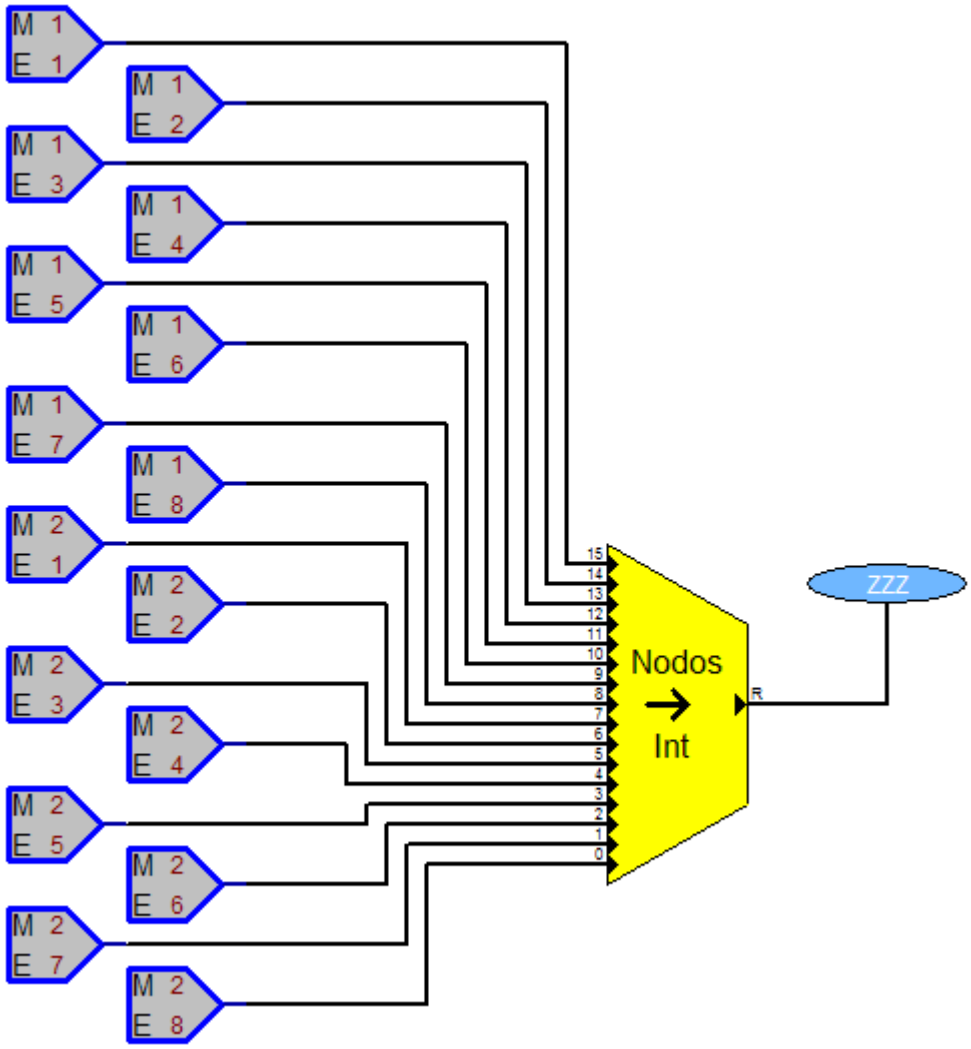
Este bloco efetua a operação contrária ao bloco descrito anteriormente, ou seja, permite condensar até 16 nodos em uma variável inteira. Conforme os nodos de 0 a 15 são acionados ou não os correspondentes bits da variável inteira são ligados ou não.



No caso deste tipo de bloco a variável é especificada na conexão, ou seja, o fio de conexão transporta a variável (em vez de serem especificadas literalmente via edição do bloco).

Note que as conexões de 0 a 15 são nodos, ou seja, variáveis binárias (assumem apenas dois valores, ligado ou desligado). Já R deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O exemplo a seguir lê as 16 entradas digitais das Expansões µDX210 e condensa estes dados na variável ZZZ. Por exemplo, se apenas a entrada E8 da segunda Expansão µDX210 estiver energizada ZZZ irá assumir valor 1. Já se ligarmos todas as entradas deste µDX210 teremos em ZZZ o valor 255 (já que 0000 0000 1111 1111 em binário corresponde a 255 em decimal).

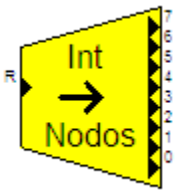


Exemplo de Programa Aplicativo: **Aritmética Inteira - Conversão Nodos - Inteiro.dxg**

Veja também:
ARITMÉTICA INTEIRA - Conversão Int -> Nodos
ARITMÉTICA WORD - Conversão Int -> Word
ARITMÉTICA WORD - Conversão Word -> Int

ARITMÉTICA INTEIRA - Conversão Int -> Nodos (8 bits)

Este bloco permite separar os 8 bits inferiores de uma variável inteira em 8 nodos distintos. Com isso, conforme os bits da variável inteira estejam ligados ou desligados os nodos correspondentes serão acionados ou não.



No caso deste tipo de bloco a variável inteira é especificada na conexão R, ou seja, o fio de conexão transporta a variável (em vez de ser especificada literalmente via edição do bloco).

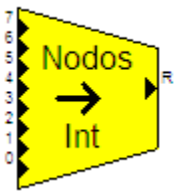
Note que R pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já as conexão de 0 a 7 à esquerda do bloco são nodos (ou seja, variáveis binárias - assumem apenas dois estados, ligado ou desligado).

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Conversão Nodos -> Int (8 bits)
ARITMÉTICA WORD - Conversão Word -> Nodos (8 bits)
ARITMÉTICA WORD - Conversão Nodos -> Word (8bits)

ARITMÉTICA INTEIRA - Conversão Nodos -> Int (8 bits)

Este bloco efetua a operação contrária ao bloco descrito anteriormente, ou seja, permite condensar até 8 nodos em uma variável inteira (utilizando apenas os bits menos significativos da variável). Conforme os nodos de 0 a 7 são acionados ou não os correspondentes bits da variável inteira são ligados ou não.



No caso deste tipo de bloco a variável é especificada na conexão, ou seja, o fio de conexão transporta a variável (em vez de serem especificadas literalmente via edição do bloco).

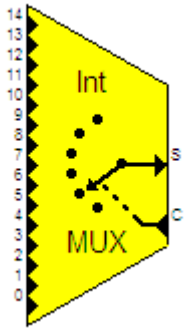
Note que as conexões de 0 a 7 são nodos, ou seja, variáveis binárias (assumem apenas dois valores, ligado ou desligado). Já R deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Conversão Int -> Nodos (8 bits)
ARITMÉTICA WORD - Conversão Word -> Nodos (8 bits)
ARITMÉTICA WORD - Conversão Nodos -> Word (8bits)

ARITMÉTICA INTEIRA - Multiplexador

Este bloco permite selecionar um dos 15 nodos de entrada (0 a 14) a ser conectado ao nodo de saída (S). A seleção é feita conforme o valor da variável inteira conectada ao controle (C). É uma chave selecionadora com 15 posições, sendo que a posição da chave é determinada por uma variável inteira. Caso esta variável assuma valor maior que 14 ou menor que zero nenhum nodo é conectado à saída S.

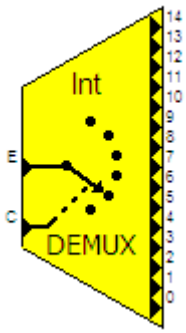


Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Demultiplexador
ARITMÉTICA INTEIRA - Multiplexador (8 bits)
ARITMÉTICA INTEIRA - Demultiplexador (8 bits)
ARITMÉTICA WORD - Multiplexador
ARITMÉTICA WORD - Demultiplexador
ARITMÉTICA WORD - Multiplexador (8 bits)
ARITMÉTICA WORD - Demultiplexador (8 bits)

ARITMÉTICA INTEIRA - Demultiplexador

Este bloco permite ligar o nodo de entrada E a um dos 15 nodos de saída (0 a 14) A seleção é feita conforme o valor da variável inteira conectada ao controle (C). É uma chave comutadora com 15 posições, sendo que a posição da chave é determinada por uma variável inteira. Caso esta variável assuma valor maior que 14 ou menor que zero a entrada E não é conectada a nenhuma saída.

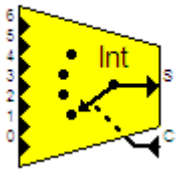


Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Multiplexador
ARITMÉTICA INTEIRA - Multiplexador (8 bits)
ARITMÉTICA INTEIRA - Demultiplexador (8 bits)
ARITMÉTICA WORD - Multiplexador
ARITMÉTICA WORD - Demultiplexador
ARITMÉTICA WORD - Multiplexador (8 bits)
ARITMÉTICA WORD - Demultiplexador (8 bits)

ARITMÉTICA INTEIRA - Multiplexador (8 bits)

Este bloco permite selecionar um dos 7 nodos de entrada (0 a 6) a ser conectado ao nodo de saída (S). A seleção é feita conforme o valor da variável inteira conectada ao controle (C). É uma chave selecionadora com 7 posições, sendo que a posição da chave é determinada por uma variável inteira. Caso esta variável assuma valor maior que 6 ou menor que zero nenhum nodo é conectado à saída S.

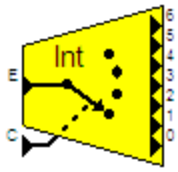


Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Multiplexador
ARITMÉTICA INTEIRA - Demultiplexador
ARITMÉTICA INTEIRA - Demultiplexador (8 bits)
ARITMÉTICA WORD - Multiplexador
ARITMÉTICA WORD - Demultiplexador
ARITMÉTICA WORD - Multiplexador (8 bits)
ARITMÉTICA WORD - Demultiplexador (8 bits)

ARITMÉTICA INTEIRA - Demultiplexador (8 bits)

Este bloco permite ligar o nodo de entrada E a um dos 7 nodos de saída (0 a 6) A seleção é feita conforme o valor da variável inteira conectada ao controle (C). É uma chave comutadora com 7 posições, sendo que a posição da chave é determinada por uma variável inteira. Caso esta variável assuma valor maior que 6 ou menor que zero a entrada E não é conectada a nenhuma saída.

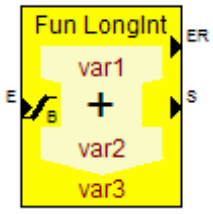


Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Multiplexador
ARITMÉTICA INTEIRA - Demultiplexador
ARITMÉTICA INTEIRA - Multiplexador (8 bits)
ARITMÉTICA WORD - Multiplexador
ARITMÉTICA WORD - Demultiplexador
ARITMÉTICA WORD - Multiplexador (8 bits)
ARITMÉTICA WORD - Demultiplexador (8 bits)

ARITMÉTICA LONGINT - Adição

Este bloco efetua a adição de dois operandos longint, colocando o resultado em um terceiro operando longint.



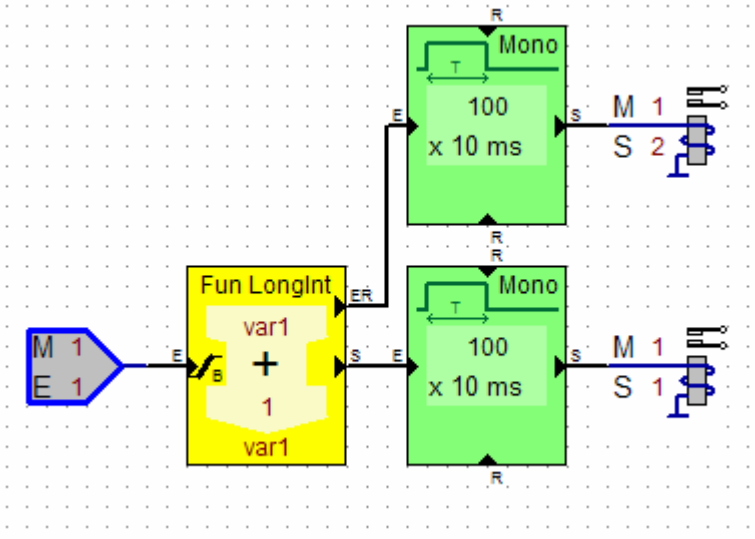
Operando1 e Operando2 são os operandos a serem adicionados e Resultado recebe o resultado da adição.

Note que Operando1 e Operando2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow ou underflow) na operação. Note que variáveis longint podem assumir valores entre os seguintes limites: -2.147.483.648 e 2.147.483.647. Assim, por exemplo, se somarmos duas variáveis, ambas com valor 2.000.000.000 (2 bilhões) o resultado será excessivo (4.000.000.000) para representação via longint e o nodo de erro será ativado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



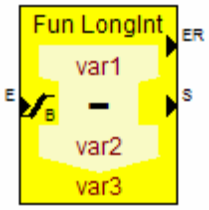
Exemplo de Programa Aplicativo: **Aritmética LongInt - Adição.dxg**

Acima temos um pequeno exemplo de uso do bloco de adição longint. No caso, a variável var1 é incrementada cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de adição está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de adição longint. Note que foram colocados monoestáveis nas saídas S e ER do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

Veja também:
ARITMÉTICA LONGINT - Adição Var
ARITMÉTICA INTEIRA - Adição
ARITMÉTICA INTEIRA - Adição Var
ARITMÉTICA WORD - Adição
ARITMÉTICA WORD - Adição Var
ARITMÉTICA REAL - Adição
ARITMÉTICA REAL - Adição Var

ARITMÉTICA LONGINT - Subtração

Este bloco efetua a subtração de dois operandos longint, colocando o resultado em um terceiro operando longint.



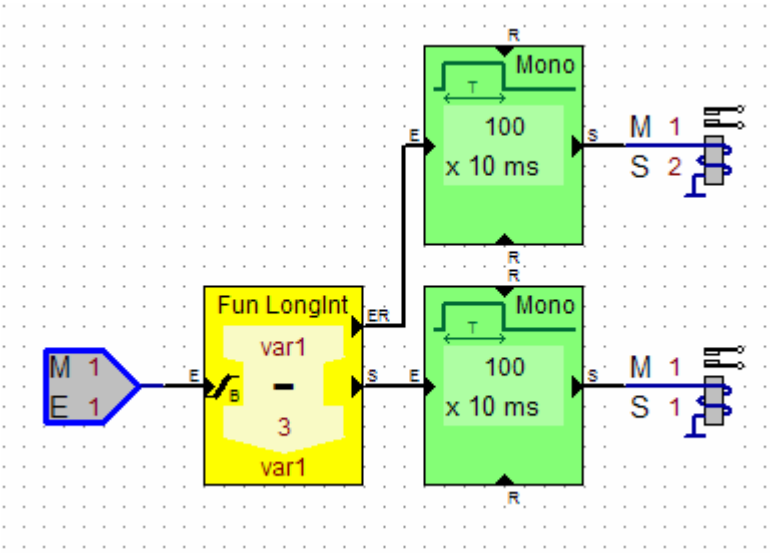
Operando1 e Operando2 são os operandos a serem subtraídos e Resultado recebe o resultado da subtração.

Note que Operando1 e Operando2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow ou underflow) na operação. Note que variáveis longint podem assumir valores entre os seguintes limites: -2.147.483.648 e 2.147.483.647. Assim, por exemplo, se subtrairmos duas variáveis, a primeira com valor 2.000.000.000 (2 bilhões) e a segunda com valor -2.000.000.000 (-2 bilhões) o resultado será excessivo (4.000.000.000) para representação via longint e o nodo de erro será ativado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Subtração.dxg**

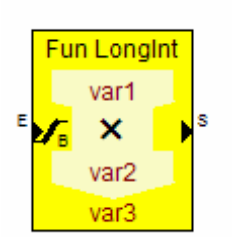
Acima temos um pequeno exemplo de uso do bloco de subtração longint. No caso, a variável var1 é decrementada em 3 unidades cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de subtração está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de

subtração longint. Note que foram colocados monoestáveis nas saídas S e ER do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

Veja também:
ARITMÉTICA LONGINT - Subtração Var
ARITMÉTICA INTEIRA - Subtração
ARITMÉTICA INTEIRA - Subtração Var
ARITMÉTICA WORD - Subtração
ARITMÉTICA WORD - Subtração Var
ARITMÉTICA REAL - Subtração
ARITMÉTICA REAL - Subtração Var

ARITMÉTICA LONGINT - Multiplicação

Este bloco efetua a multiplicação de dois operandos inteiros, colocando o resultado em um terceiro operando longint.



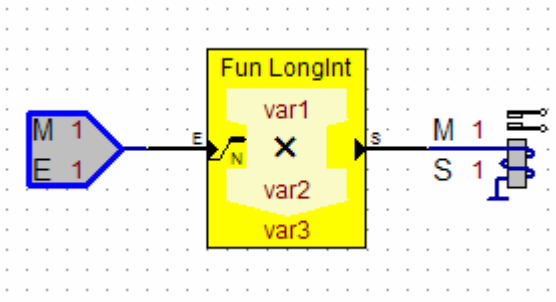
Operando1 e Operando2 são os operandos inteiros a serem multiplicados e Resultado recebe o resultado da multiplicação (longint).

Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Já o resultado da operação é um longint, e com isso pode assumir valores entre -2.147.483.648 e 2.147.483.647. Com isso não há como a operação gerar erro, pois o maior valor de entrada (-32768 x -32768) ainda resulta em um valor representável por longint (1.073.741.824).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



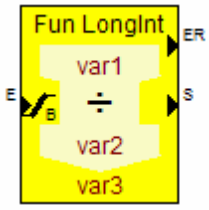
Exemplo de Programa Aplicativo: **Aritmética LongInt - Multiplicação.dxg**

Acima temos um pequeno exemplo de uso do bloco de multiplicação longint. No caso, a variável var1 é multiplicada pela variável var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de multiplicação está selecionada por nível). O resultado da operação é armazenado na variável var3. Já a saída S1 do mesmo módulo é acionada conforme o nodo Saída (S) do bloco de multiplicação longint. Note que neste caso não foi colocado monoestável na saída S do bloco, ao contrário dos exemplos anteriores. Isso porque a entrada E do bloco de multiplicação está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA LONGINT - Multiplicação Var
ARITMÉTICA INTEIRA - Multiplicação
ARITMÉTICA INTEIRA - Multiplicação Var
ARITMÉTICA REAL - Multiplicação
ARITMÉTICA REAL - Multiplicação Var

ARITMÉTICA LONGINT - Divisão

Este bloco efetua a divisão entre um operando longint (dividendo) por um operando inteiro (divisor), colocando o resultado em um terceiro operando inteiro (quociente).



Operando1 (longint) e Operando2 (inteiro) são os operandos a serem divididos e Resultado (inteiro) recebe o resultado da divisão.

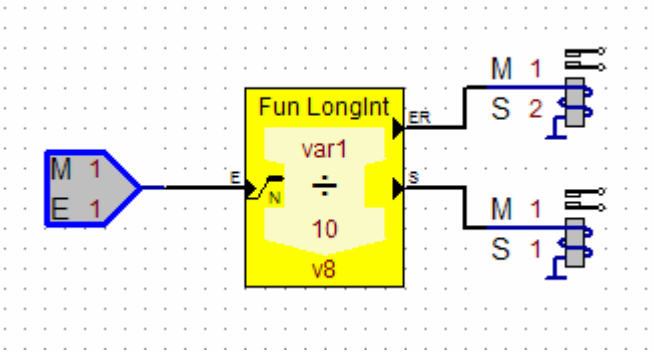
Note que Operando1 pode ser uma variável longint, constante ou uma referência a uma posição de memória. Já Operando2 pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Já as variáveis longint podem assumir valores entre -2.147.483.648 e 2.147.483.647. Com isso, se o resultado da operação resultar em valor menor

que -32768 ou maior que 32767, ou ainda o divisor for zero, o nodo de Erro (ER) será acionado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Divisão.dxg**

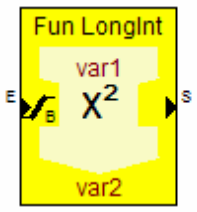
Acima temos um pequeno exemplo de uso do bloco de divisão longint. No caso, a variável var1 é dividida pela constante 10 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de divisão está selecionada por nível). O resultado da operação é armazenado na variável v8 (que corresponde a saída analógica S1 do µDX200). Ou seja, a saída analógica S1 irá rastrear a variável var1 dividida por 10 do µDX200.

Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de divisão longint. Note que neste caso não foram colocados monoestáveis nas saídas S e ER do bloco, ao contrário de exemplos anteriores. Isso porque a entrada E do bloco de divisão está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA LONGINT - Divisão Var
ARITMÉTICA INTEIRA - Divisão
ARITMÉTICA INTEIRA - Divisão Var
ARITMÉTICA REAL - Divisão
ARITMÉTICA REAL - Divisão Var

ARITMÉTICA LONGINT - Quadrado

Este bloco eleva ao quadrado um operando inteiro, colocando o resultado em um segundo operando longint.



Operando é o operando a ser elevado ao quadrado e Resultado recebe o resultado da operação.

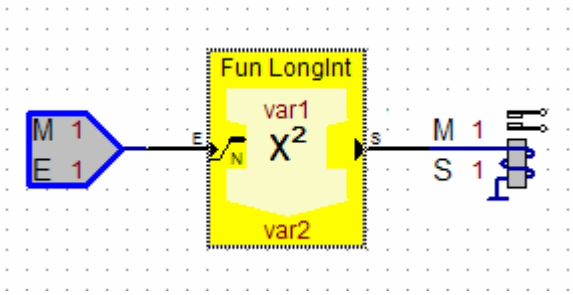
Note que Operando pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser

necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Já o resultado da operação é um longint, e com isso pode assumir valores entre -2.147.483.648 e 2.147.483.647. Com isso não há como a operação gerar erro, pois o maior valor de entrada (-32768 x -32768) ainda resulta em um valor representável por longint (1.073.741.824).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Quadrado.dxg**

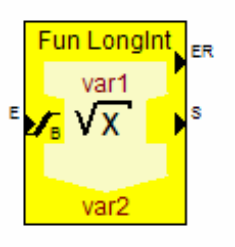
Acima temos um pequeno exemplo de uso do bloco de quadrado longint. No caso, a variável var1 é elevada ao quadrado sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de quadrado está selecionada por nível). O resultado da operação é armazenado na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de quadrado longint. A entrada E do bloco de quadrado está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA LONGINT - Quadrado Var
ARITMÉTICA INTEIRA - Quadrado
ARITMÉTICA INTEIRA - Quadrado Var
ARITMÉTICA REAL - Quadrado
ARITMÉTICA REAL - Quadrado Var

ARITMÉTICA LONGINT - Raiz Quadrada

Este bloco calcula a raiz quadrada de um operando longint, colocando o resultado em um segundo operando inteiro.



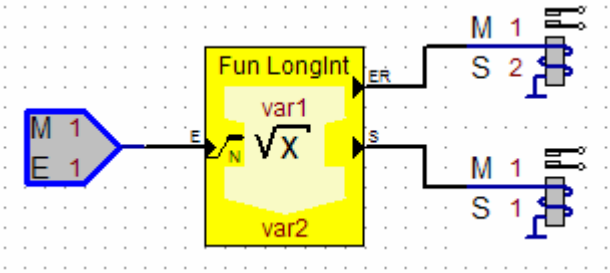
Operando é o operando a ser efetuada a raiz quadrada e Resultado recebe o resultado da operação.

Note que Operando pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. No caso deste bloco o erro ocorre se for tentado retirar a raiz quadrada de um número negativo, ou se o Operando assumir valor maior que 1.073.676.289.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Raiz Quadrada.dxg**

Acima temos um pequeno exemplo de uso do bloco de raiz quadrada longint. No caso, é retirada a raiz quadrada da variável var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de raiz quadrada está selecionada por nível). O resultado da operação é armazenado na variável var2.

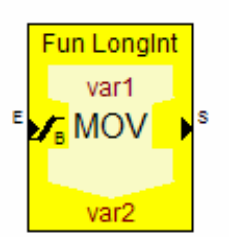
Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de raiz quadrada longint. A entrada E do bloco de raiz quadrada está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA LONGINT - Raiz Quadrada Var
ARITMÉTICA INTEIRA - Raiz Quadrada
ARITMÉTICA INTEIRA - Raiz Quadrada Var
ARITMÉTICA REAL - Raiz Quadrada

ARITMÉTICA REAL - Raiz Quadrada Var

ARITMÉTICA LONGINT - Atribuição

Este bloco transfere o valor de um operando longint para um segundo operando longint.



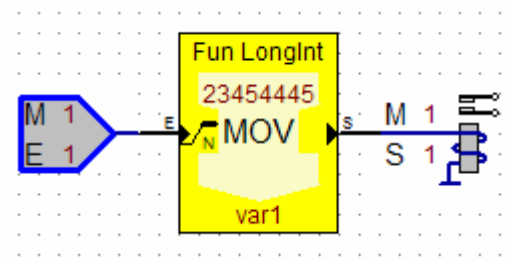
Operando é o operando cujo valor será transferido para Resultado.

Note que Operando pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Atribuição.dxg**

Acima temos um pequeno exemplo de uso do bloco de atribuição longint. No caso, é atribuído o valor 23.454.445 na variável longint var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível).

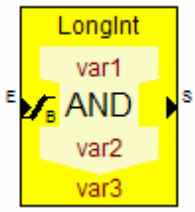
Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição longint. A entrada E do bloco de atribuição está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA LONGINT - Atribuição Var
ARITMÉTICA INTEIRA - Atribuição
ARITMÉTICA INTEIRA - Atribuição Var
ARITMÉTICA WORD - Atribuição

ARITMÉTICA WORD - Atribuição Var
ARITMÉTICA REAL - Atribuição
ARITMÉTICA REAL - Atribuição Var

ARITMÉTICA LONGINT - Operação AND

Este bloco efetua a operação booleana AND entre dois operandos longint, e Resultado recebe o resultado da operação. Observe que a operação AND é feita bit a bit com os 32 bits dos operandos.

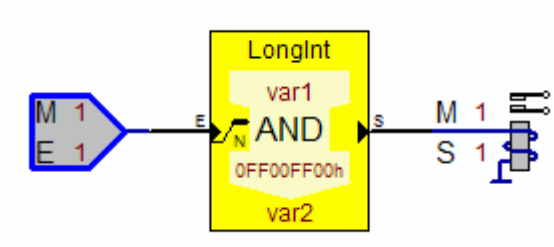


Note que Operando1 e Operando2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 00000000h a 0FFFFFFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A304D11h e não A304D11h (pois o µDX200 entenderia que se trata da variável chamada A304D11H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Operação AND.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana AND longint. No caso, é efetuada a operação AND entre a variável var1 e a constante hexadecimal 0FF00FF00h sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível). O resultado da operação é guardado em var2. Note que esta operação zera dois bytes de var1 (já que qualquer valor AND com 00h resulta em zero). Por exemplo, se var1 possui valor de 34992010 (215EF8Ah) ao efetuar a operação AND com 0FF00FF00h irá resultar no valor 33615616

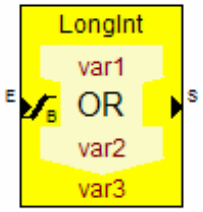
(200EF00h) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição longint. A entrada E do bloco de AND está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA LONGINT - Operação AND Var
ARITMÉTICA INTEIRA - Operação AND
ARITMÉTICA INTEIRA - Operação AND Var
ARITMÉTICA WORD - Operação AND
ARITMÉTICA WORD - Operação AND Var

ARITMÉTICA LONGINT - Operação OR

Este bloco efetua a operação booleana OR entre dois operandos longint, e Resultado recebe o resultado da operação. Observe que a operação OR é feita bit a bit com os 32 bits dos operandos.

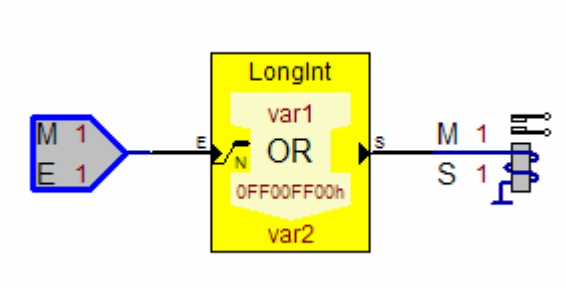


Note que Operando1 e Operando2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 00000000h a 0FFFFFFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A304D11h e não A304D11h (pois o µDX200 entenderia que se trata da variável chamada A304D11H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Operação OR.dxg**

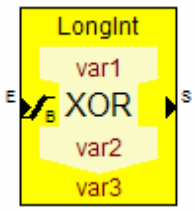
Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana OR inteira. No caso, é efetuada a operação OR entre a variável var1 e a constante hexadecimal 0FF00FF00h sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível). O resultado da operação é guardado em var2. Note que esta operação liga todos os bits do byte MSB de var1 (já que qualquer valor OR com 0FFh resulta em um). Por exemplo, se var1 possui valor de 34992010 (215EF8Ah) ao efetuar a operação OR com 0FF00FF00h irá resultar no valor -15335542 (0FF15FF8Ah) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição longint. A entrada E do bloco de OR está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA LONGINT - Operação OR Var
ARITMÉTICA INTEIRA - Operação OR
ARITMÉTICA INTEIRA - Operação OR Var
ARITMÉTICA WORD - Operação OR
ARITMÉTICA WORD - Operação OR Var

ARITMÉTICA LONGINT - Operação XOR

Este bloco efetua a operação booleana XOR entre dois operandos longint, e Resultado recebe o resultado da operação. Observe que a operação XOR é feita bit a bit com os 32 bits dos operandos.



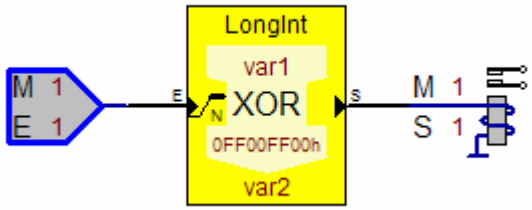
Note que Operando1 e Operando2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 00000000h a 0FFFFFFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não

uma variável. Por exemplo, deve-se usar 0A304D11h e não A304D11h (pois o µDX200 entenderia que se trata da variável chamada A304D11H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Operação XOR.dxg**

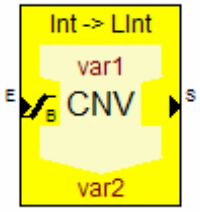
Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana XOR longint. No caso, é efetuada a operação XOR entre a variável var1 e a constante hexadecimal 0FF00FF00h sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível). O resultado da operação é guardado em var2. Note que esta operação inverte todos os bits de dois bytes MSB de var1 (já que qualquer valor XOR com 0FFh resulta em inversão bit a bit). Por exemplo, se var1 possui valor de 34992010 (215EF8Ah) ao efetuar a operação XOR com 0FF00FF00h irá resultar no valor -48951158 (0FD15108Ah) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição longint. A entrada E do bloco de XOR está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA LONGINT - Operação XOR Var
ARITMÉTICA INTEIRA - Operação XOR
ARITMÉTICA INTEIRA - Operação XOR Var
ARITMÉTICA WORD - Operação XOR
ARITMÉTICA WORD - Operação XOR Var

ARITMÉTICA LONGINT - Conversão Inteiro -> LongInt

Este bloco transfere o valor de um operando inteiro para um segundo operando longint.



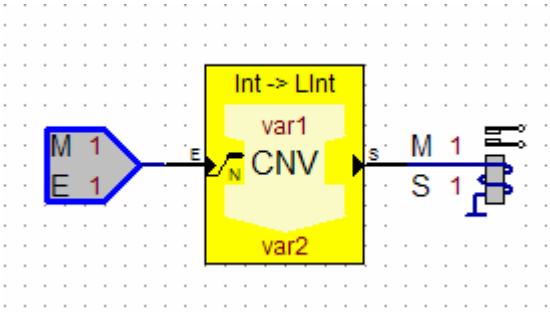
Operando é o operando cujo valor será transferido para Resultado.

Note que Operando pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Conversão Inteiro - LongInt.dxg**

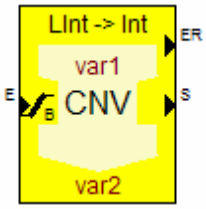
O exemplo acima transfere o valor da variável inteira var1 para a variável longint var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível).

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de conversão inteiro à longint. A entrada E do bloco de atribuição está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA LONGINT - Conversão LongInt -> Inteiro
ARITMÉTICA LONGINT - Conversão Int -> LInt
ARITMÉTICA LONGINT - Conversão LInt -> Int

ARITMÉTICA LONGINT - Conversão LongInt -> Inteiro

Este bloco transfere o valor de um operando longint para um segundo operando inteiro.



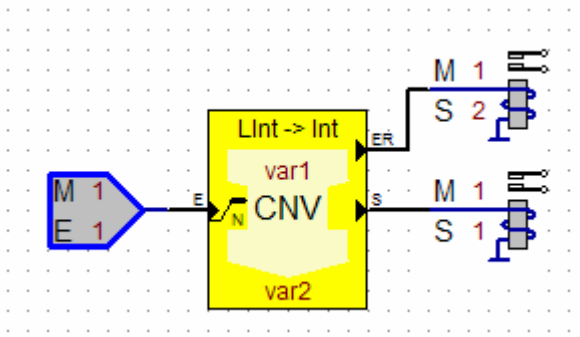
Operando é o operando cujo valor será transferido para Resultado.

Note que Operando pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco possui nodo de Erro (ER), pois se Operando assumir valores maiores que os limites representáveis pelos números inteiros (-32768 a 32767) a conversão não será possível.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Conversão LongInt - Inteiro.dxg**

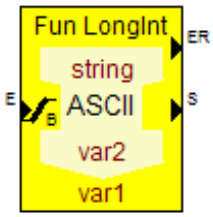
O exemplo acima transfere o valor da variável longint var1 para a variável inteira var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível).

Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de conversão longint à Inteiro. A entrada E do bloco está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver ligada.

Veja também:
ARITMÉTICA LONGINT - Conversão Inteiro -> LongInt
ARITMÉTICA LONGINT - Conversão Int -> LInt
ARITMÉTICA LONGINT - Conversão LInt -> Int

ARITMÉTICA LONGINT - Conversão ASCII -> LongInt

Este bloco converte um string apontado pelo bloco em um valor longint. É possível especificar um offset inicial para o string, e um caracter de finalização do string.



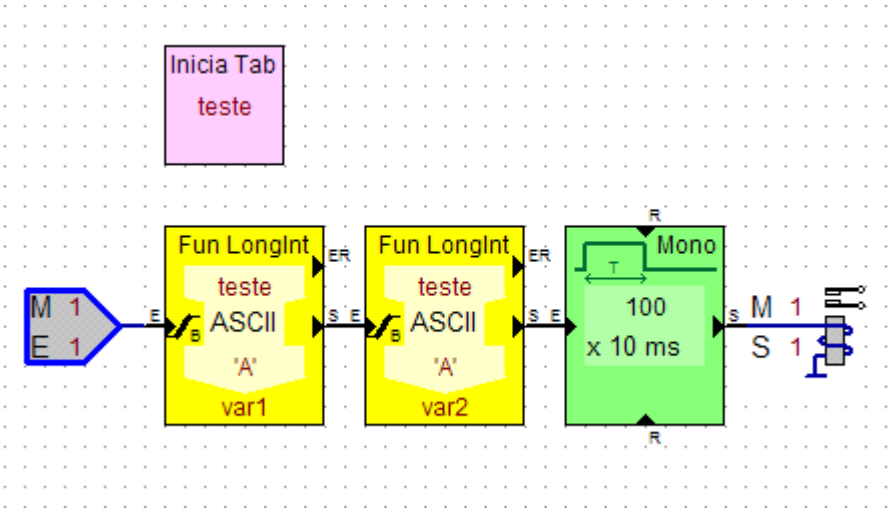
String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Offset indica a partir de qual posição do string é iniciada a conversão. Já Resultado recebe o valor convertido, e Caracter especifica um caracter de finalização para a conversão do string.

Note que String e Offset podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Já Caracter deve ser necessariamente uma constante, pois trata-se de tipo byte (que não é tratado pelo µDX200, sendo admitido apenas como constante).

Caso seja especificado Offset = -1 o bloco inicia a conversão a partir do ponto do string em que o bloco anterior de Conversão ASCIIàLongint parou. Isso é útil para analisar strings com vários valores numéricos encadeados.

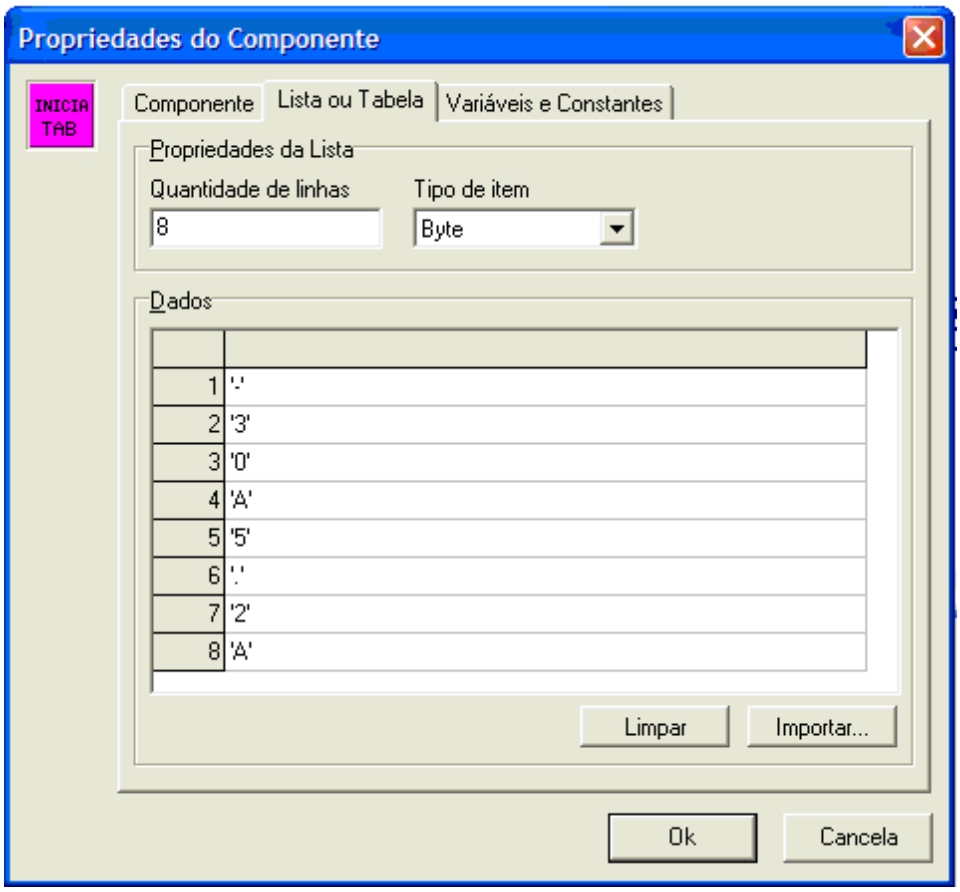
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

O nodo de Erro (ER) indica ter havido problemas na conversão. Por exemplo, foi encontrado um caracter não válido no string. Os caracteres válidos são os numerais (de '0' a '9') representados em ASCII (30h a 39h), vírgula ou ponto (',' ou '.'), que são desconsiderados já que é feita a conversão para longint (sem ponto decimal), ou o caracter de terminação especificado. Ainda é permitido o uso de sinal de menos ('-') para indicar números negativos.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Conversão ASCII - LongInt.dxg**

O programa de exemplo acima converte dois valores numéricos representados no string teste para as variáveis longint var1 e var2. Note que utilizou-se Offset =0 para o bloco à esquerda (de forma que ele inicie a conversão a partir da primeira posição do string teste), e Offset = -1 para o bloco à direita (de forma que este bloco reinicie a conversão onde o primeiro bloco parou). Os valores inicializados no string teste via bloco de Inicia Tab são:



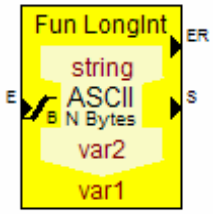
Note que colocando caracteres entre apóstrofes o µDX200 entende que deve armazenar a representação ASCII do caracter no byte. Assim, foram usados caracteres 'A' para caracter de terminação (cujo valor em ASCII é 65). Portanto, o string teste é inicializado com teste = "-30A5.2A".

Ao rodar este programa e energizar a entrada E1 do módulo de Expansão M1 resulta em var1 = -30 e var2 = 52.

Veja também:
ARITMÉTICA LONGINT - Conversão ASCII -> LongInt N Bytes
ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro
ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro N Bytes

ARITMÉTICA LONGINT - Conversão ASCII -> LongInt N Bytes

Este bloco converte um string apontado pelo bloco em um valor longint. É possível especificar um offset inicial para o string, e em vez do caracter de terminação especificado no bloco anterior, neste bloco é especificado o número de caracteres a serem analisados.



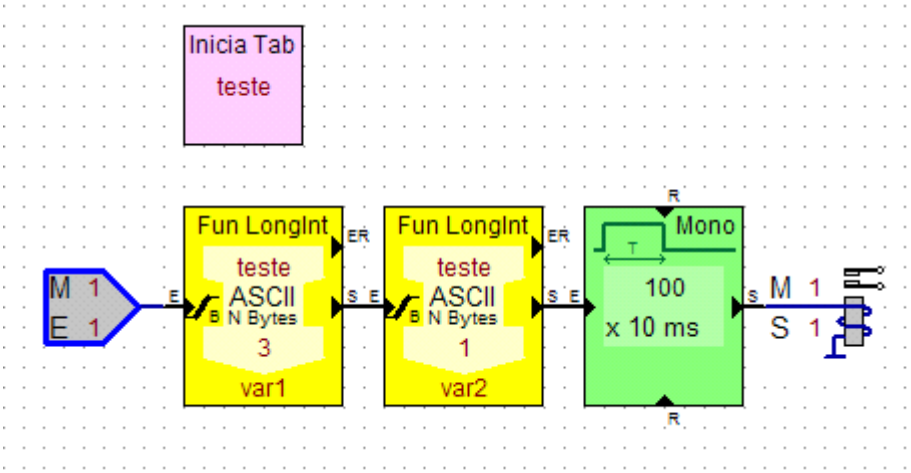
String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Offset indica a partir de qual posição do string é iniciada a conversão. Já Resultado recebe o valor convertido, e Núm.Bytes especifica quantos caracteres a partir de Offset serão analisados.

Note que String, Offset e Núm.Bytes podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Caso seja especificado Offset = -1 o bloco inicia a conversão a partir do ponto do string em que o bloco anterior de Conversão ASCIIàLongInt parou. Isso é útil para analisar strings com vários valores numéricos encadeados.

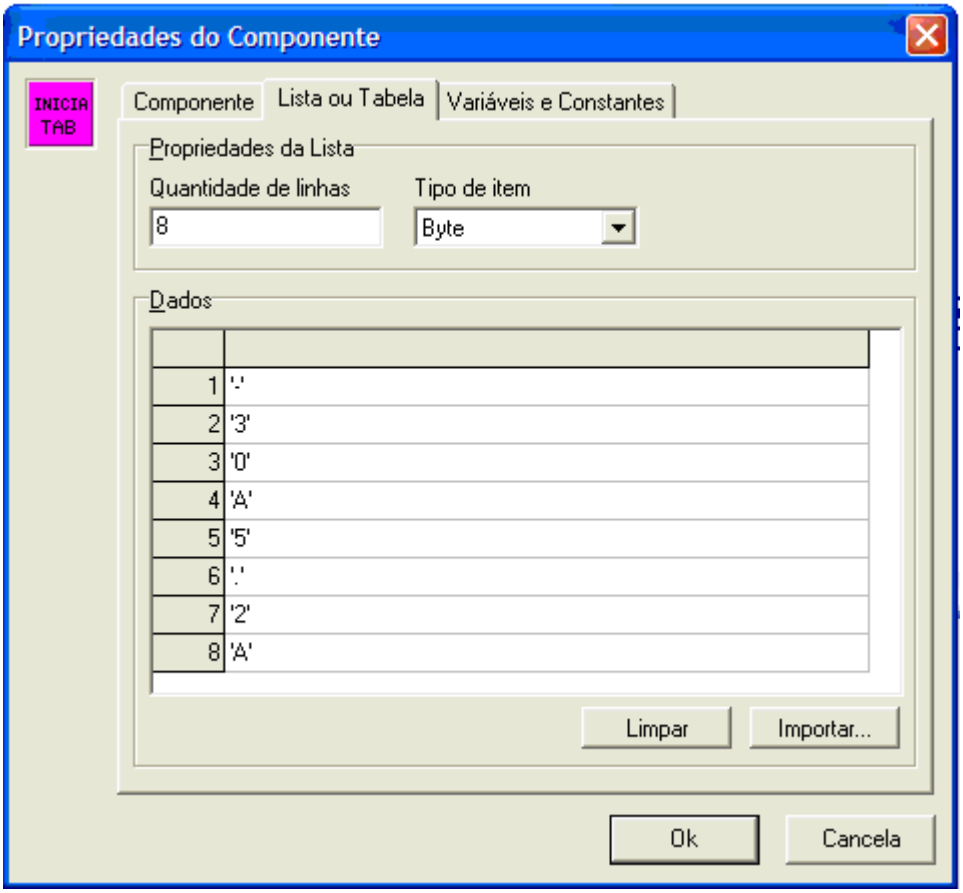
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

O nodo de Erro (ER) indica ter havido problemas na conversão. Por exemplo, foi encontrado um caracter não válido no string. Os caracteres válidos são os numerais (de '0' a '9') representados em ASCII (30h a 39h), vírgula ou ponto (',' ou '.'), que são desconsiderados já que é feita a conversão para inteiro (sem ponto decimal). Ainda é permitido o uso de sinal de menos ('-') para indicar números negativos.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Conversão ASCII - LongInt N Bytes.dxg**

O programa de exemplo acima converte dois valores numéricos representados no string teste para as variáveis inteiras var1 e var2. Note que utilizou-se Offset =0 para o bloco à esquerda (de forma que ele inicie a conversão a partir da primeira posição do string teste), e Offset = 6 para o bloco à direita (de forma que este bloco inicie a conversão a partir da sétima posição do string teste). Os valores inicializados no string teste via bloco de Inicia Tab são:



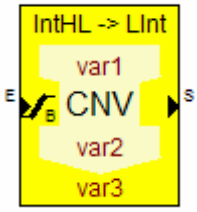
Note que colocando caracteres entre apóstrofes o µDX200 entende que deve armazenar a representação ASCII do caracter no byte. Foram mantidos caracteres 'A' no string teste apenas para mantê-lo igual ao do exemplo anterior, mas agora tais caracteres não têm significado, já que o bloco analisa um número de caracteres do string e não até um caracter de terminação. Portanto, o string teste é inicializado com teste = "-30A5.2A".

Ao rodar este programa e energizar a entrada E1 do módulo de Expansão M1 resulta em var1 = -30 e var2 = 2.

Veja também:
ARITMÉTICA LONGINT - Conversão ASCII -> LongInt
ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro
ARITMÉTICA INTEIRA - Conversão ASCII -> Inteiro N Bytes

ARITMÉTICA LONGINT - Conversão IntH,IntL -> LongInt

Este bloco transfere o primeiro operando inteiro para o word superior, e o segundo operando inteiro para o word inferior de uma variável longint.



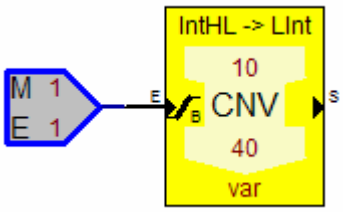
Ou seja, ele irá "montar" uma variável longint a partir de dois valores inteiros. Isso equivale a operação matemática de multiplicar o primeiro operando por 65536 e somar o resultado com o segundo operando.

Note que Operando pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

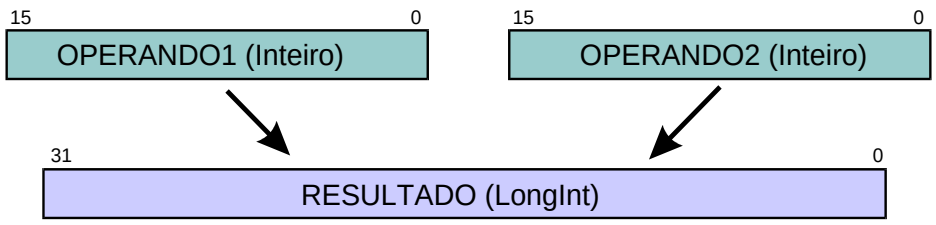
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Conversão IntH, IntL - LongInt.dxg**

O exemplo acima transfere a constante 10 para o word superior da variável longint var, e tranasfere a constante 40 para o word inferior desta variável., sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for ligada (já que a entrada E do bloco de atribuição está selecionada por borda).

Isso equivale a atribuir a variável var o valor 10*65536+40 = 655400. Note que transferir o primeiro operando para o word superior equivale a multiplicá-lo por 216 = 65536.

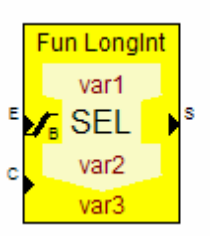


Variáveis inteiras do µDX200 são representadas por dois bytes (um word), enquanto variáveis longint são representadas por quatro bytes (dois words).

Veja também:
ARITMÉTICA LONGINT - Conversão IntH,IntL -> LongInt Var

ARITMÉTICA LONGINT - Seletor

Este bloco transfere o valor de um ou outro operando longint para um terceiro operando longint.



Operando1 e Operando2 são os operandos que serão transferidos para Resultado, conforme o estado do nodo de Controle (C).

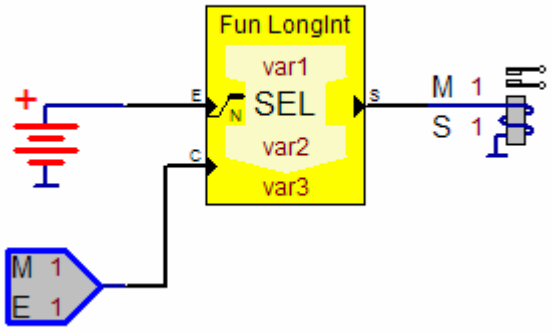
Note que Operando1 e Operando2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

O nodo de Controle (C) especifica qual dos operandos será transferido para Resultado. Se C=0 o Operando1 é transferido para Resultado. Já se C=1 o Operando2 é transferido para Resultado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Seletor.dxg**

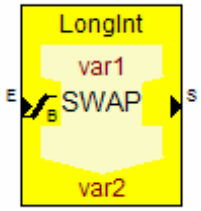
Acima temos um pequeno exemplo de uso do bloco de seleção longint. No caso, é atribuído o valor da variável var1 ou da variável var2 na variável longint var3, conforme o estado do nodo de Controle (C). O nodo de Entrada (E) está sempre ativado, pois está selecionado para acionamento por nível e está ligado constantemente pelo bloco de energia. O nodo de Controle (C) é comando pela entrada E1 do módulo 1 de Expansão de Entradas/Saídas (µDX210).

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição inteira. A entrada E do bloco de atribuição está selecionada para acionamento por nível e sempre ligada. Então, a saída S ficará ativa sempre.

Veja também:
ARITMÉTICA INTEIRA - Seletor
ARITMÉTICA WORD - Seletor
ARITMÉTICA REAL - Seletor
GERAL - Seletor de Variável Inteira
GERAL - Seletor de Variável LongInt
GERAL - Seletor de Variável Word
GERAL - Seletor de Variável Real

ARITMÉTICA LONGINT - Operação SWAP

Este bloco permuta o word mais significativo e o word menos significativo de um operando longint, e coloca o resultado em um segundo operando longint.



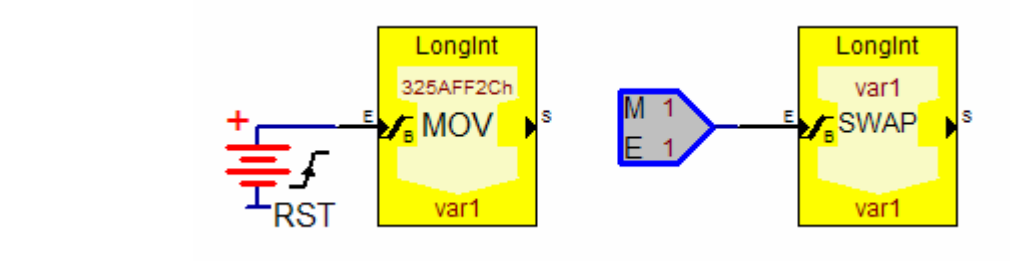
Operando é o operando cujo valor será permutado e transferido para Resultado.

Note que Operando pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



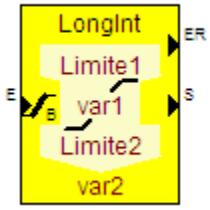
Exemplo de Programa Aplicativo: **Aritmética LongInt - Operação SWAP.dxg**

Acima temos um pequeno exemplo de uso do bloco de SWAP longint. No caso, é atribuído o valor 325AFF2Ch na variável longint var1 ao resetar o controlador programável. Sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for energizada a variável var1 troca o word MSB pelo LSB e vice-versa. Ou seja, na primeira energização de E1 a variável var1 irá assumir valor 0FF2C325Ah. Na próxima energização de E1 irá retornar ao valor 325AFF2Ch, e assim sucessivamente.

Veja também:
ARITMÉTICA LONGINT - Operação SWAP Var

ARITMÉTICA LONGINT - Operação Limite

Permite limitar uma variável longint entre dois valores especificados. Assim, se garante que a variável assumirá apenas valores entre os dois limites. Este bloco somente é operacional em controladores µDX201 versão 3.37 ou superior.



Operando é a variável cujo valor será limitado entre os limites estabelecidos por Lim.Superior e Lim.Inferior e transferido para Resultado.

Note que Operando, Lim.Superior e Lim.Inferior podem ser variáveis longint, constantes (valor numérico em decimal ou hexadecimal), ou ainda referências a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Variáveis longint podem assumir valores entre os seguintes limites: -2.147.483.648 e 2.147.483.647.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que Operando possui valor fora dos limites impostos por Lim.Superior e Lim.Inferior.

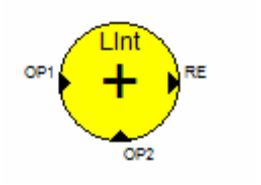
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Limite
ARITMÉTICA INTEIRA - Operação Limite Var
ARITMÉTICA LONGINT - Operação Limite Var
ARITMÉTICA WORD - Operação Limite
ARITMÉTICA WORD - Operação Limite Var
ARITMÉTICA REAL - Operação Limite
ARITMÉTICA REAL - Operação Limite Var

ARITMÉTICA LONGINT - Adição Var

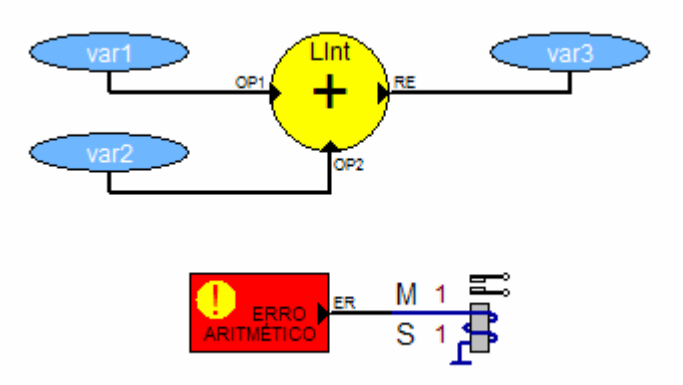
Este bloco efetua a adição de dois operandos longint, colocando o resultado em um terceiro operando longint. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos longint a serem adicionados e RE recebe o resultado da adição.

Note que OP1 e OP2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis longint podem assumir valores entre -2.147.483.648 e 2.147.483.647. Assim, por exemplo, se somarmos duas variáveis, ambas com valor 2.000.000.000 (2 bilhões) o resultado será excessivo (4.000.000.000) para representação via longint e ocorrerá um erro. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética LongInt - Adição Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de adição var longint. No caso, a variável

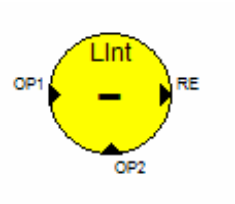
var1 é somada a variável var2 constantemente, e o resultado da operação é colocado em var3. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de adição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis longint (entre -2.147.483.648 e 2.147.483.647) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA LONGINT - Adição
ARITMÉTICA INTEIRA - Adição
ARITMÉTICA INTEIRA - Adição Var
ARITMÉTICA WORD - Adição
ARITMÉTICA WORD - Adição Var
ARITMÉTICA REAL - Adição
ARITMÉTICA REAL - Adição Var

ARITMÉTICA LONGINT - Subtração Var

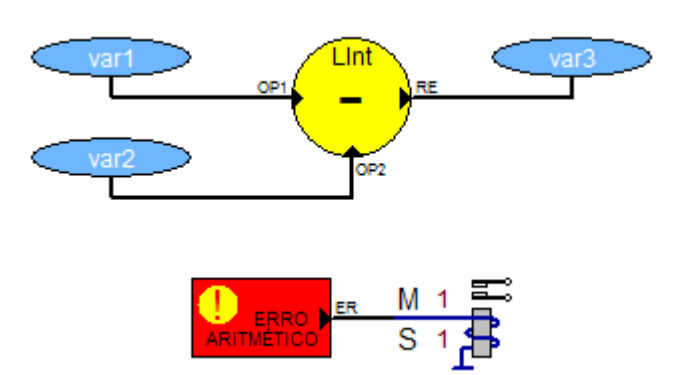
Este bloco efetua a subtração de dois operandos longint, colocando o resultado em um terceiro operando longint. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos longint a serem subtraídos e RE recebe o resultado da subtração.

Note que OP1 e OP2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis longint podem assumir valores entre -2.147.483.648 e 2.147.483.647. Assim, por exemplo, se subtrairmos duas variáveis, uma com valor positivo de 2.000.000.000 (2 bilhões) e a outra com valor negativo de -2.000.000.000 (-2 bilhões) o resultado será excessivo (4.000.000.000) para representação via longint e ocorrerá um erro. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética LongInt - Subtração Var.dxg**

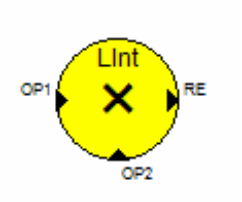
Acima temos um pequeno exemplo de uso do bloco de subtração var longint. No caso, a variável var2 é subtraída da variável var1 constantemente, e o resultado da operação é colocado em var3. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de subtração pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (entre -2.147.483.648 e 2.147.483.647) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA LONGINT - Subtração
ARITMÉTICA INTEIRA - Subtração
ARITMÉTICA INTEIRA - Subtração Var
ARITMÉTICA WORD - Subtração
ARITMÉTICA WORD - Subtração Var
ARITMÉTICA REAL - Subtração
ARITMÉTICA REAL - Subtração Var

ARITMÉTICA LONGINT - Multiplicação Var

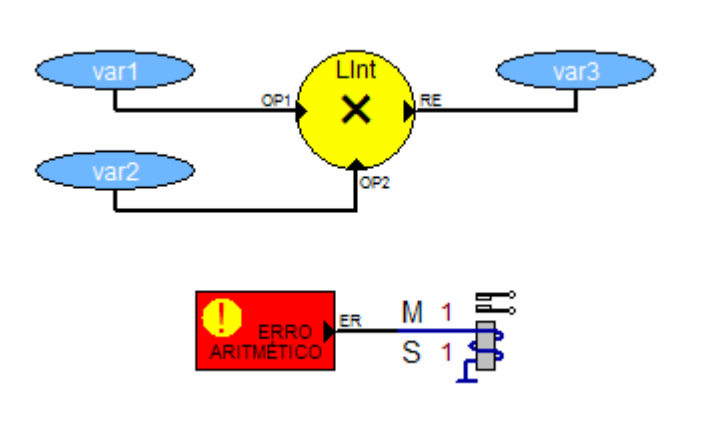
Este bloco efetua a multiplicação de dois operandos inteiros, colocando o resultado em um terceiro operando longint. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos inteiros a serem multiplicados e RE recebe o resultado da multiplicação (longint).

Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Já as variáveis longint podem assumir valores entre -2.147.483.648 e 2.147.483.647. Com isso não há como a operação gerar erro, pois o maior valor de entrada (-32768 x -32768) ainda resulta em um valor representável por longint (1.073.741.824).



Exemplo de Programa Aplicativo: **Aritmética LongInt - Multiplicação Var.dxg**

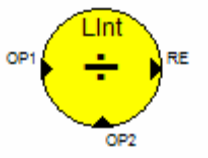
Acima temos um pequeno exemplo de uso do bloco de multiplicação var longint. No caso, a variável var1 é multiplicada com a variável var2 constantemente, e o resultado da operação é colocado em var3. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210). Só que neste tipo de operação (multiplicação de dois números inteiros resultando em um número longint) nunca haverá erro aritmético.

Note que as variáveis são transportadas para o bloco de multiplicação pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) e longint (entre -2.147.483.648 e 2.147.483.647) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA LONGINT - Multiplicação
ARITMÉTICA INTEIRA - Multiplicação
ARITMÉTICA INTEIRA - Multiplicação Var
ARITMÉTICA REAL - Multiplicação
ARITMÉTICA REAL - Multiplicação Var

ARITMÉTICA LONGINT - Divisão Var

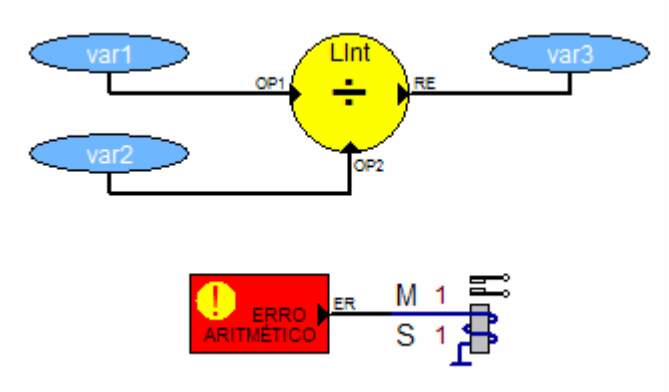
Este bloco efetua a divisão de um operando longint (longint) por um operando inteiro (divisor), colocando o resultado em um terceiro operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é um operando longint (dividendo), OP2 é um operando inteiro (divisor), e RE recebe o resultado da divisão (quociente).

Note que OP1 e OP2 podem ser variáveis longint ou inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Já as variáveis longint podem assumir valores entre -2.147.483.648 e 2.147.483.647. Com isso, se o resultado da operação resultar em valor menor que -32768 ou maior que 32767, ou ainda o divisor for zero, será gerado um erro aritmético. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética LongInt - Divisão Var.dxg**

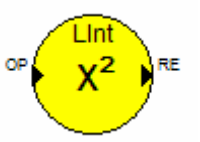
Acima temos um pequeno exemplo de uso do bloco de divisão var longint. No caso, a variável var1 é dividida pela variável var2 constantemente, e o resultado da operação é colocado em var3. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de divisão pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis longint (entre -2.147.483.648 e 2.147.483.647) ou inteiras (entre -32768 e 32767) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA LONGINT - Divisão
ARITMÉTICA INTEIRA - Divisão
ARITMÉTICA INTEIRA - Divisão Var
ARITMÉTICA REAL - Divisão
ARITMÉTICA REAL - Divisão Var

ARITMÉTICA LONGINT - Quadrado Var

Este bloco eleva ao quadrado um operando inteiro, colocando o resultado em um segundo operando longint. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

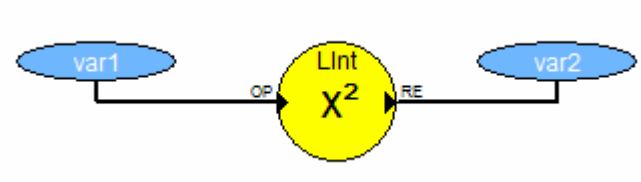


OP é os operando inteiro a ser elevado ao quadrado e RE recebe o resultado da operação (longint).

Note que OP pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado

da operação.

Variáveis inteiras podem assumir valores entre -32768 e 32767. Já as variáveis longint podem assumir valores entre -2.147.483.648 e 2.147.483.647. Com isso não há como a operação gerar erro, pois o maior valor absoluto de entrada (-32768) ainda resulta em um valor representável por longint (1.073.741.824).



Exemplo de Programa Aplicativo: **Aritmética LongInt - Quadrado Var.dxg**

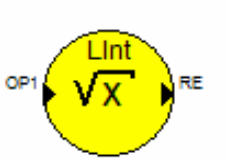
Acima temos um pequeno exemplo de uso do bloco de quadrado var longint. No caso, a variável var1 é elevada ao quadrado constantemente, e o resultado da operação é colocado em var2.

Note que as variáveis são transportadas para o bloco de quadrado pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) ou longint (entre -2.147.483.648 e 2.147.483.647) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA LONGINT - Quadrado
ARITMÉTICA INTEIRA - Quadrado
ARITMÉTICA INTEIRA - Quadrado Var
ARITMÉTICA REAL - Quadrado
ARITMÉTICA REAL - Quadrado Var

ARITMÉTICA LONGINT - Raiz Quadrada Var

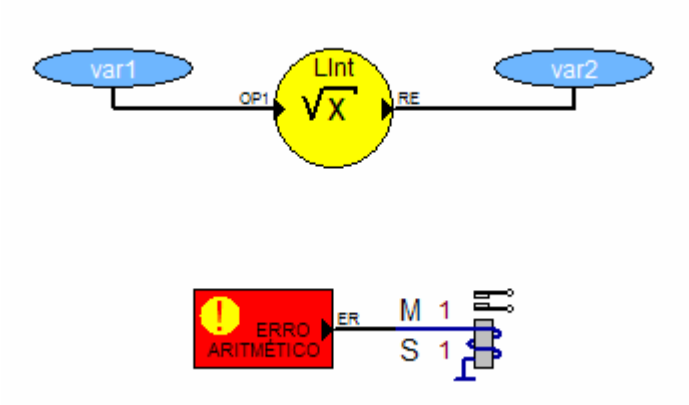
Este bloco calcula a raiz quadrada de um operando longint, colocando o resultado em um segundo operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é o operando longint do qual será extraída a raiz quadrada e RE recebe o resultado da operação (inteiro).

Note que OP1 pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis inteiras podem assumir valores entre -32768 e 32767. Já as variáveis longint podem assumir valores entre -2.147.483.648 e 2.147.483.647. Caso a variável de entrada assuma valor maior que 1.073.741.823 irá resultar em erro aritmético, pois o resultado irá exceder o máximo valor representável via variável inteira. Também se tentarmos retirar uma raiz quadrada de número negativo irá gerar um erro aritmético. Como o bloco não possui nodo de erro só é possível detectar o erro via bloco de Erro Aritmético (família Geral).



Exemplo de Programa Aplicativo: **Aritmética LongInt - Raiz Quadrada Var.dxg**

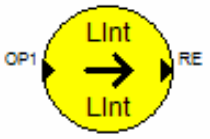
Acima temos um pequeno exemplo de uso do bloco de raiz quadrada var longint. No caso, é retirada a raiz quadrada da variável var1 constantemente, e o resultado da operação é colocado em var2. Para indicar erro na operação foi incluído um bloco de Erro Aritmético que liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210).

Note que as variáveis são transportadas para o bloco de raiz quadrada pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-32768 a 32767) ou longint (entre -2.147.483.648 e 2.147.483.647) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA LONGINT - Raiz Quadrada
ARITMÉTICA INTEIRA - Raiz Quadrada
ARITMÉTICA INTEIRA - Raiz Quadrada Var
ARITMÉTICA REAL - Raiz Quadrada
ARITMÉTICA REAL - Raiz Quadrada Var

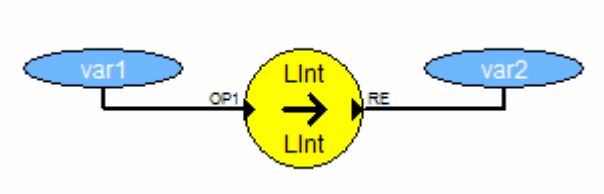
ARITMÉTICA LONGINT - Atribuição Var

Este bloco transfere o valor de um operando longint para um segundo operando longint. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é o operando longint a ser transferido para o operando longint RE.

Note que OP1 pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



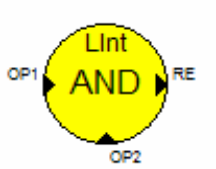
Exemplo de Programa Aplicativo: **Aritmética LongInt - Atribuição Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de atribuição var longint. No caso, é transferido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis longint (entre -2.147.483.648 e 2.147.483.647) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA LONGINT - Atribuição
ARITMÉTICA INTEIRA - Atribuição
ARITMÉTICA INTEIRA - Atribuição Var
ARITMÉTICA WORD - Atribuição
ARITMÉTICA WORD - Atribuição Var
ARITMÉTICA REAL - Atribuição
ARITMÉTICA REAL - Atribuição Var

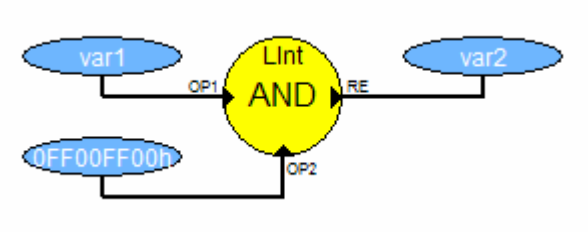
ARITMÉTICA LONGINT - Operação AND Var

Este bloco efetua a operação booleana AND entre dois operandos longint, e um terceiro operando longint recebe o resultado da operação . Observe que a operação AND é feita bit a bit com os 32 bits dos operandos. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



Note que OP1 e OP2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 00000000h a 0FFFFFFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A304D11h e não A304D11h (pois o µDX200 entenderia que se trata da variável chamada A304D11H).



Exemplo de Programa Aplicativo: **Aritmética LongInt - Operação AND Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de operação booleana AND longint. No caso, é efetuada a operação AND entre a variável var1 e a constante hexadecimal 0FF00FF00h constantemente.

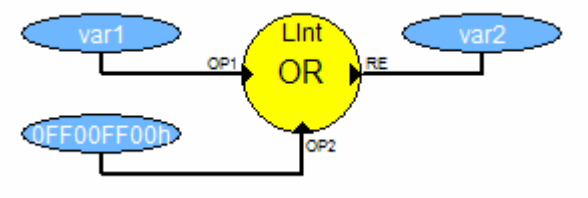
Veja também:
ARITMÉTICA LONGINT - Operação AND
ARITMÉTICA INTEIRA - Operação AND
ARITMÉTICA INTEIRA - Operação AND Var
ARITMÉTICA WORD - Operação AND
ARITMÉTICA WORD - Operação AND Var

ARITMÉTICA LONGINT - Operação OR Var

Este bloco efetua a operação booleana OR entre dois operandos longint, e um terceiro operando recebe o resultado da operação. Observe que a operação OR é feita bit a bit com os 32 bits dos operandos. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

Note que OP1 e OP2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 00000000h a 0FFFFFFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A304D11h e não A304D11h (pois o µDX200 entenderia que se trata da variável chamada A304D11H).



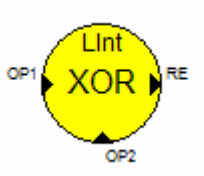
Exemplo de Programa Aplicativo: **Aritmética LongInt - Operação OR Var.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana OR inteira. No caso, é efetuada a operação OR entre a variável var1 e a constante hexadecimal 0FF00FF00h constantemente.

Veja também:
ARITMÉTICA LONGINT - Operação OR
ARITMÉTICA INTEIRA - Operação OR
ARITMÉTICA INTEIRA - Operação OR Var
ARITMÉTICA WORD - Operação OR
ARITMÉTICA WORD - Operação OR Var

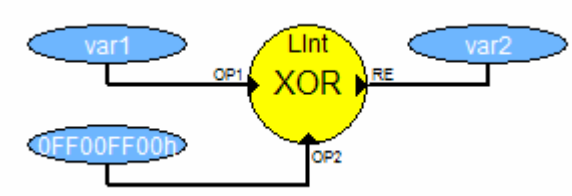
ARITMÉTICA LONGINT - Operação XOR Var

Este bloco efetua a operação booleana XOR entre dois operandos longint, e Resultado recebe o resultado da operação. Observe que a operação XOR é feita bit a bit com os 32 bits dos operandos. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



Note que OP1 e OP2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 00000000h a 0FFFFFFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A304D11h e não A304D11h (pois o µDX200 entenderia que se trata da variável chamada A304D11H).



Exemplo de Programa Aplicativo: **Aritmética LongInt - Operação XOR Var.dxg**

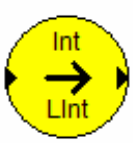
Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana XOR longint. No caso, é efetuada a operação XOR entre a variável var1 e a constante hexadecimal 0FF00FF00h constantemente.

Veja também:
ARITMÉTICA LONGINT - Operação XOR

ARITMÉTICA INTEIRA - Operação XOR
ARITMÉTICA INTEIRA - Operação XOR Var
ARITMÉTICA WORD - Operação XOR
ARITMÉTICA WORD - Operação XOR Var

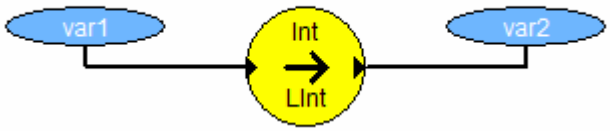
ARITMÉTICA LONGINT - Conversão Int -> LInt Var

Este bloco converte o valor de um operando inteiro para um segundo operando longint. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é o operando inteiro a ser transferido para o operando longint RE.

Note que OP1 pode ser variável inteiro, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



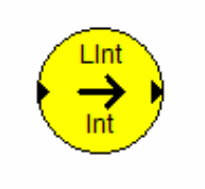
Exemplo de Programa Aplicativo: **Aritmética LongInt - Conversão Inteiro - LongInt Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de conversão var inteiro à longint. No caso, é transferido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (entre -32.768 e 32.767) e longint (entre -2.147.483.648 e 2.147.483.647) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA LONGINT - Conversão LInt -> Int
ARITMÉTICA LONGINT - Conversão Inteiro -> LongInt
ARITMÉTICA LONGINT - Conversão LongInt -> Inteiro

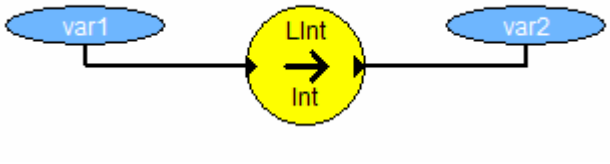
ARITMÉTICA LONGINT - Conversão LInt -> Int Var

Este bloco converte o valor de um operando longint para um segundo operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é o operando longint a ser transferido para o operando inteiro RE.

Note que OP1 pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



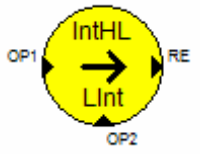
Exemplo de Programa Aplicativo: **Aritmética LongInt - Conversão LongInt - Inteiro Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de conversão var longint à inteiro. No caso, é transferido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (entre -32.768 e 32.767) e longint (entre -2.147.483.648 e 2.147.483.647) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

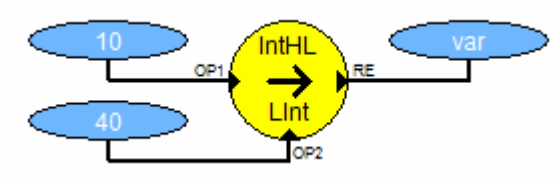
Veja também:
ARITMÉTICA LONGINT - Conversão Int -> LInt
ARITMÉTICA LONGINT - Conversão Inteiro -> LongInt
ARITMÉTICA LONGINT - Conversão LongInt -> Inteiro

ARITMÉTICA LONGINT - Conversão IntH,IntL -> LongInt Var

Este bloco transfere o primeiro operando inteiro para o word superior, e o segundo operando inteiro para o word inferior de uma variável longint. Ou seja, ele irá "montar" uma variável longint a partir de dois valores inteiros. Isso equivale a operação matemática de multiplicar o primeiro operando por 65536 e somar o resultado com o segundo operando. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

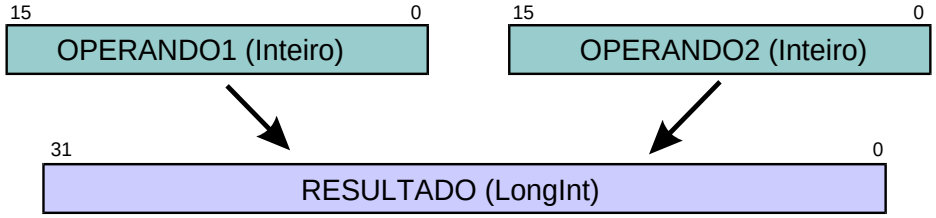


Note que OP1 e OP2 podem ser variáveis inteiras, constantes (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Conversão IntH, IntL - LongInt Var.dxg**

O exemplo acima transfere a constante 10 para o word superior da variável longint var, e transfere a constante 40 para o word inferior desta variável. Isso equivale a atribuir a variável var o valor 10*65536+40 = 655400. Note que transferir o primeiro operando para o word superior equivale a multiplicá-lo por 216 = 65536.Variáveis inteiras do µDX200 são representadas por dois bytes (um word), enquanto variáveis longint são representadas por quatro bytes (dois words).



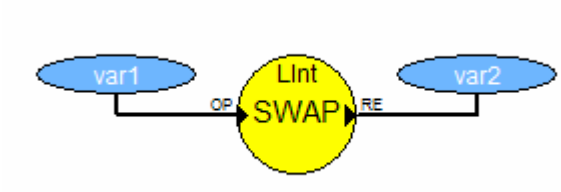
Veja também:
ARITMÉTICA LONGINT - Conversão IntH,IntL -> LongInt

ARITMÉTICA LONGINT - Operação SWAP Var

Este bloco permuta o word mais significativo e o word menos significativo de um operando longint para um segundo operando longint. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

OP1 é o operando longint a ser transferido para RE.

Note que OP1 pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



Exemplo de Programa Aplicativo: **Aritmética LongInt - Operação SWAP Var.dxg**

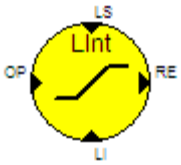
Acima temos um pequeno exemplo de uso do bloco de SWAP longint var. No caso, é atribuído à

variável var2 o valor de var1 com o word MSB trocado pelo word LSB e vice-versa. Por exemplo, se atribuirmos a var1 o valor hexadecimal 11307C4Ah a variável var2 irá assumir o valor hexadecimal 7C4A1130h.

Veja também:
ARITMÉTICA LONGINT - Operação SWAP

ARITMÉTICA LONGINT - Operação Limite Var

Permite limitar uma variável longint entre dois valores especificados. Assim, se garante que a variável assumirá apenas valores entre os dois limites. Este bloco somente é operacional em controladores µDX201 versão 3.37 ou superior. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP (operando) é a variável cujo valor será limitado entre os limites estabelecidos por LS (limite superior) e LI (limite inferior) e transferido para RE (resultado).

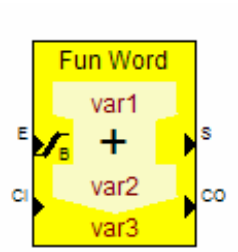
Note que OP, LS e LI podem ser variáveis inteiras, constantes (valor numérico em decimal ou hexadecimal), ou ainda referências a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Variáveis longint podem assumir valores entre os seguintes limites: -2.147.483.648 e 2.147.483.647.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Limite
ARITMÉTICA INTEIRA - Operação Limite Var
ARITMÉTICA LONGINT - Operação Limite
ARITMÉTICA WORD - Operação Limite
ARITMÉTICA WORD - Operação Limite Var
ARITMÉTICA REAL - Operação Limite
ARITMÉTICA REAL - Operação Limite Var

ARITMÉTICA WORD - Adição

Este bloco efetua a adição de dois operandos word, colocando o resultado em um terceiro operando word.



Operando1 e Operando2 são os operandos a serem adicionados e Resultado recebe o resultado da adição.

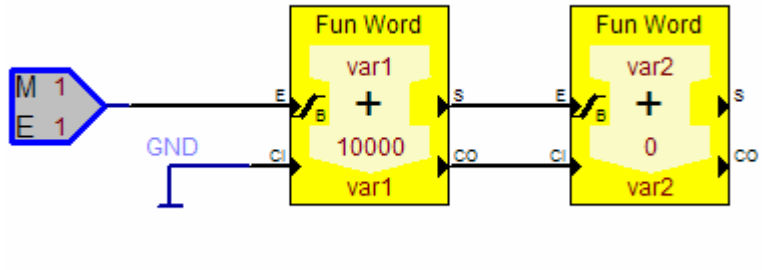
Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Carry Out (CO) indica que houve excesso na operação. Note que variáveis word podem assumir valores entre 0 e 65535. Assim, por exemplo, se somarmos duas variáveis, ambas com valor 40000 o resultado será excessivo (80000) para representação via word e o nodo de CO será ativado. Neste caso pode-se ligar este nodo de saída em um nodo de entrada Carry In (CI) de outro bloco de adição word. Isso permite concatenar blocos de adição. Assim, o primeiro bloco resultaria em 80000-65536 = 14464, e o segundo bloco resultaria em 1 (devido ao nodo de CI ligado), o que representa 65536*1 = 65536. Portanto, o resultado final seria 14464 + 65536 = 80000.

O nodo de Carry In (CI) adiciona uma unidade ao resultado da adição word, e serve para concatenar mais de um bloco de adição, permitindo com isso efetuar operações com números maiores que 65535.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Adição.dxg**

Acima temos um pequeno exemplo de uso do bloco de adição word. No caso, a variável var1 é incrementada cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de adição está selecionada por borda) em 10000. Quando esta variável

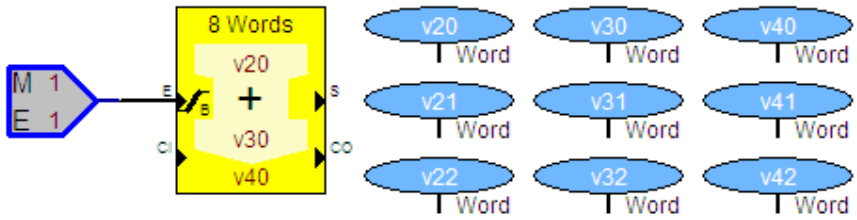
exceder 65535 será acionado o nodo de Carry In (CI) do próximo bloco, incrementando var2 (que representa, portanto, var1*65536). A variável var1 assume valores 0, 10000, 20000, 30000, 40000, 50000, 60000 e, a seguir, 70000-65536 = 4464. Neste momento var2 assume valor 1 (que representa 65536). A soma total será dada por:

Total = var2 * 65536 + var1

Note que podemos associar quantos blocos forem necessários. O programa acima poderá contar até 65535*65536+65535 = 4.294.967.295. Se usarmos três blocos de adição word o limite será de mais de 281 trilhões (281.474.976.710.655)!

Obs.: foi ligado um bloco de GND (terra) para forçar a zero o nodo de CI do primeiro bloco de adição word. Poderia também simplesmente ser deixado desconectado o nodo de CI (Carry In).

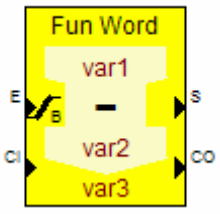
Atenção: A partir da versão 3.61 do controlador µDX201 existe suporte para blocos Word com múltiplas variáveis (8 words). Nesses blocos se usa uma variável absoluta para apontar o word inicial, e o bloco efetuada a operação nessa variável e nas 7 subseqüentes. Por exemplo, ao efetuar uma soma de v20 com v30 usando o bloco de **Adição 8 Words**, ao ativar o bloco será feita a soma de v20,v21,v22,v23,v24,v25,v26,v27 com v30,v31,v32,v33,v34,v35,v36,v37. Note que é considerado como words LSB v20 e v30, e como words MSB v27 e V37. Abaixo um exemplo:



Veja também:
ARITMÉTICA WORD - Adição Var
ARITMÉTICA INTEIRA - Adição
ARITMÉTICA INTEIRA - Adição Var
ARITMÉTICA LONGINT - Adição
ARITMÉTICA LONGINT - Adição Var
ARITMÉTICA REAL - Adição
ARITMÉTICA REAL - Adição Var

ARITMÉTICA WORD - Subtração

Este bloco efetua a subtração de dois operandos word, colocando o resultado em um terceiro operando word.



Operando1 e Operando2 são os operandos a serem subtraídos e Resultado recebe o resultado da subtração.

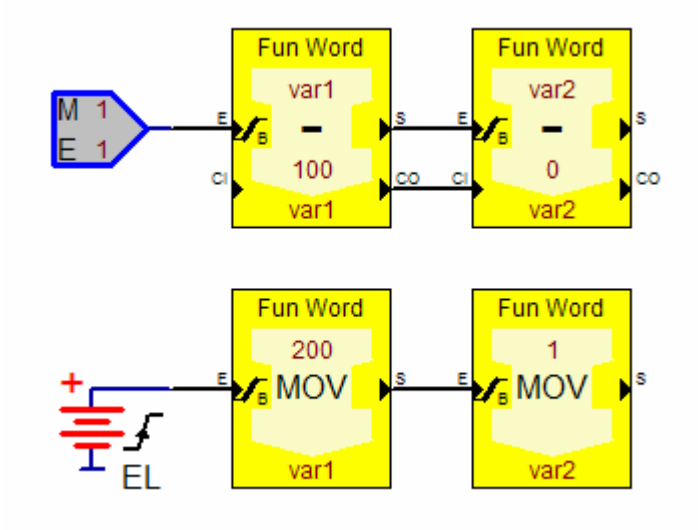
Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Carry Out (CO) indica que houve falta (borrow) na operação. Note que variáveis word podem assumir valores entre 0 e 65535. Assim, por exemplo, se subtrairmos duas variáveis, sendo que a primeira seja menor que a segunda, o resultado seria um número negativo, sem representação via word e o nodo de CO será ativado. Neste caso pode-se ligar este nodo de saída em um nodo de entrada Carry In (CI) de outro bloco de subtração word. Isso permite concatenar blocos de subtração.

O nodo de Carry In (CI) subtrai uma unidade ao resultado da subtração word, e serve para concatenar mais de um bloco de subtração, permitindo com isso efetuar operações com números maiores que 65535.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Subtração.dxg**

Acima temos um pequeno exemplo de uso do bloco de subtração word. No caso, a variável var1 é decrementada em 100 unidades cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de subtração está selecionada por borda). Como a variável var1 é inicializada (via bloco de Energia Liga - EL) com valor 200 e var2 é inicializada com valor 1, as variáveis irão assumir os seguintes valores a cada energização da entrada E1: Como foi usado um bloco de Nodo EL para acionar a inicialização das variáveis var1 e var2, é preciso desenergizar momentaneamente o controlador para que as variáveis inicializem. Caso se queira que as variáveis inicializem ao rodar o programa aplicativo substitua o bloco de Nodo EL por um bloco de Nodo Reset.

Var1=200 var2=1
Var1=100 var2=1
Var1=0 var2=1
Var1=65436 var2=0

Var1=65336 var2=0 .....

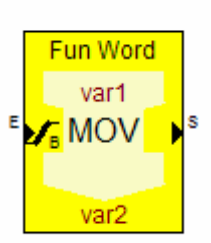
Ou seja, o valor total inicial é 65536*var2+var1 = 65736.

> **Atenção:** A partir da versão 3.61 do controlador µDX201 existe suporte para blocos Word com múltiplas variáveis (8 words). Nesses blocos se usa uma variável absoluta para apontar o word inicial, e o bloco efetuada a operação nessa variável e nas 7 subseqüentes. Por exemplo, ao efetuar uma subtração de v20 com v30 usando o bloco de **Subtração 8 Words**, ao ativar o bloco será feita a subtração de v20,v21,v22,v23,v24,v25,v26,v27 com v30,v31,v32,v33,v34,v35,v36,v37. Note que é considerado como words LSB v20 e v30, e como words MSB v27 e V37.

Veja também:
ARITMÉTICA WORD - Subtração Var
ARITMÉTICA INTEIRA - Subtração
ARITMÉTICA INTEIRA - Subtração Var
ARITMÉTICA LONGINT - Subtração
ARITMÉTICA LONGINT - Subtração Var
ARITMÉTICA REAL - Subtração
ARITMÉTICA REAL - Subtração Var

ARITMÉTICA WORD - Atribuição

Este bloco transfere o valor de um operando word para um segundo operando word.



Operando é o operando cujo valor será transferido para Resultado.

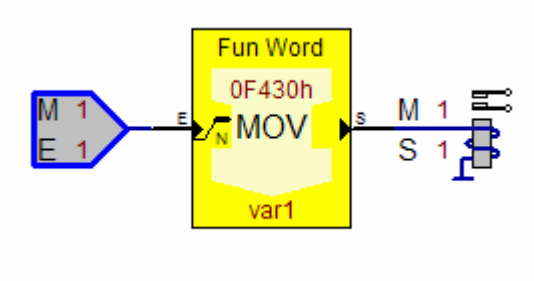
Note que Operando pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

Note que variáveis word podem assumir valores entre 0 e 65535 (0 e 0FFFFh em hexadecimal).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Atribuição.dxg**

Acima temos um pequeno exemplo de uso do bloco de atribuição word. No caso, é atribuído o valor 0F430h (em decimal 62512) na variável word var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível).

Note que para especificar números hexadecimais é necessário que o número inicie por um numeral (por isso o número zero no início do valor 0F430h) e terminar com a letra "h".

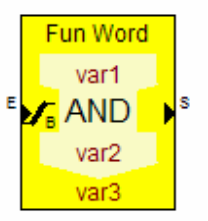
Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição word. A entrada E do bloco de atribuição está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Atenção: A partir da versão 3.61 do controlador µDX201 existe suporte para blocos Word com múltiplas variáveis (8 words). Nesses blocos se usa uma variável absoluta para apontar o word inicial, e o bloco efetuada a operação nessa variável e nas 7 subseqüentes. Por exemplo, ao efetuar uma atribuição de v20 em v30 usando o bloco de **Atribuição 8 Words**, ao ativar o bloco será feita a transferência do valor de v20,v21,v22,v23,v24,v25,v26,v27 para v30,v31,v32,v33,v34,v35,v36,v37.

Veja também:
ARITMÉTICA WORD - Atribuição Var
ARITMÉTICA INTEIRA - Atribuição
ARITMÉTICA INTEIRA - Atribuição Var
ARITMÉTICA LONGINT - Atribuição
ARITMÉTICA LONGINT - Atribuição Var
ARITMÉTICA REAL - Atribuição
ARITMÉTICA REAL - Atribuição Var

ARITMÉTICA WORD - Operação AND

Este bloco efetua a operação booleana AND entre dois operandos word, e Resultado recebe o resultado da operação. Observe que a operação AND é feita bit a bit com os 16 bits dos operandos.

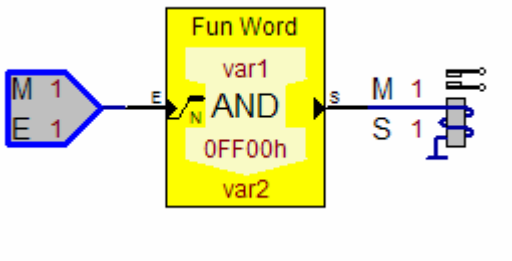


Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 0000h a 0FFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A30h e não A30h (pois o µDX200 entenderia que se trata da variável chamada A30H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Operação AND.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana AND word. No caso, é efetuada a operação AND entre a variável var1 e a constante hexadecimal 0FF00h sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível). O resultado da operação é guardado em var2. Note que esta operação zera o byte LSB de var1 (já que qualquer valor AND com 00h resulta em zero). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação AND com 0FF00h irá resultar no valor 3328 (0D00h) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição inteira. A entrada E do bloco de AND está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

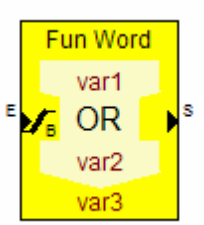
Atenção: A partir da versão 3.61 do controlador µDX201 existe suporte para blocos Word com múltiplas variáveis (8 words). Nesses blocos se usa uma variável absoluta para apontar o word inicial, e o bloco efetuada a operação nessa variável e nas 7 subseqüentes. Por exemplo, ao efetuar uma operação AND de v20 e v30 usando o bloco de **AND 8 Words**, ao ativar o bloco será feita a operação AND word a word entre v20,v21,v22,v23,v24,v25,v26,v27 e v30,v31,v32,v33,v34,v35,v36,v37.

Veja também:
ARITMÉTICA WORD - Operação AND Var
ARITMÉTICA INTEIRA - Operação AND
ARITMÉTICA INTEIRA - Operação AND Var
ARITMÉTICA LONGINT - Operação AND

ARITMÉTICA LONGINT - Operação AND Var

ARITMÉTICA WORD - Operação OR

Este bloco efetua a operação booleana OR entre dois operandos word, e Resultado recebe o resultado da operação. Observe que a operação OR é feita bit a bit com os 16 bits dos operandos.

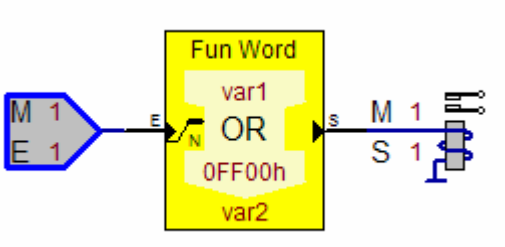


Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 0000h a 0FFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A30h e não A30h (pois o µDX200 entenderia que se trata da variável chamada A30H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Operação OR.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana OR word. No caso, é efetuada a operação OR entre a variável var1 e a constante hexadecimal 0FF00h sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível). O resultado da operação é guardado em var2. Note que esta operação liga todos os bits do byte MSB de var1 (já que qualquer valor OR com 0FFh resulta em um). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação OR com 0FF00h irá resultar no valor 65451 (0FFABh) na variável var2.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo

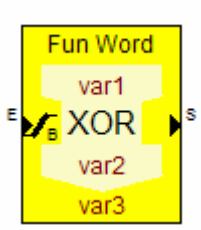
Saída (S) do bloco de atribuição inteira. A entrada E do bloco de OR está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

> **Atenção:** A partir da versão 3.61 do controlador µDX201 existe suporte para blocos Word com múltiplas variáveis (8 words). Nesses blocos se usa uma variável absoluta para apontar o word inicial, e o bloco efetuada a operação nessa variável e nas 7 subseqüentes. Por exemplo, ao efetuar uma operação OR de v20 e v30 usando o bloco de **OR 8 Words**, ao ativar o bloco será feita a operação OR word a word entre v20,v21,v22,v23,v24,v25,v26,v27 e v30,v31,v32,v33,v34,v35,v36,v37.

Veja também:
ARITMÉTICA WORD - Operação OR Var
ARITMÉTICA INTEIRA - Operação OR
ARITMÉTICA INTEIRA - Operação OR Var
ARITMÉTICA LONGINT - Operação OR
ARITMÉTICA LONGINT - Operação OR Var

ARITMÉTICA WORD - Operação XOR

Este bloco efetua a operação booleana XOR entre dois operandos word, e Resultado recebe o resultado da operação. Observe que a operação XOR é feita bit a bit com os 16 bits dos operandos.

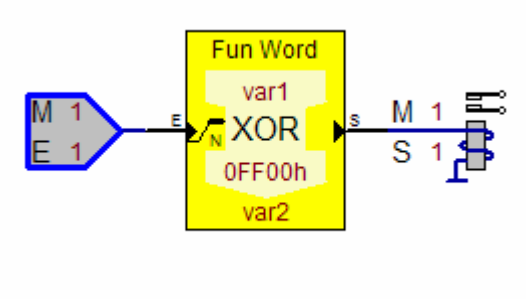


Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Muitas vezes este bloco é usado com o segundo operando igual a uma constante em hexadecimal. Neste caso, pode-se especificar um valor hexadecimal de 0000h a 0FFFFh (valores hexadecimais são especificados com numerais terminados pela letra h ou H). No caso de valores hexadecimais iniciados por uma letra (A,B,C,D,E ou F) é preciso incluir um zero à esquerda, de forma que o Compilador PG entenda que se trata de uma constante e não uma variável. Por exemplo, deve-se usar 0A30h e não A30h (pois o µDX200 entenderia que se trata da variável chamada A30H).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Operação XOR.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana XOR word. No caso, é efetuada a operação XOR entre a variável var1 e a constante hexadecimal 0FF00h sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível). O resultado da operação é guardado em var2. Note que esta operação inverte todos os bits do byte MSB de var1 (já que qualquer valor XOR com 0FFh resulta em inversão bit a bit). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação XOR com 0FF00h irá resultar no valor 62123 (0F2ABh) na variável var2.

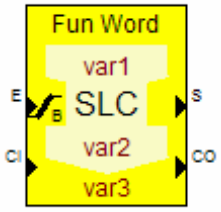
Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição inteira. A entrada E do bloco de XOR está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Atenção: A partir da versão 3.61 do controlador µDX201 existe suporte para blocos Word com múltiplas variáveis (8 words). Nesses blocos se usa uma variável absoluta para apontar o word inicial, e o bloco efetuada a operação nessa variável e nas 7 subseqüentes. Por exemplo, ao efetuar uma operação XOR de v20 e v30 usando o bloco de **XOR 8 Words**, ao ativar o bloco será feita a operação XOR word a word entre v20,v21,v22,v23,v24,v25,v26,v27 e v30,v31,v32,v33,v34,v35,v36,v37.

Veja também:
ARITMÉTICA WORD - Operação XOR Var
ARITMÉTICA INTEIRA - Operação XOR
ARITMÉTICA INTEIRA - Operação XOR Var
ARITMÉTICA LONGINT - Operação XOR
ARITMÉTICA LONGINT - Operação XOR Var

ARITMÉTICA WORD - Operação SLC

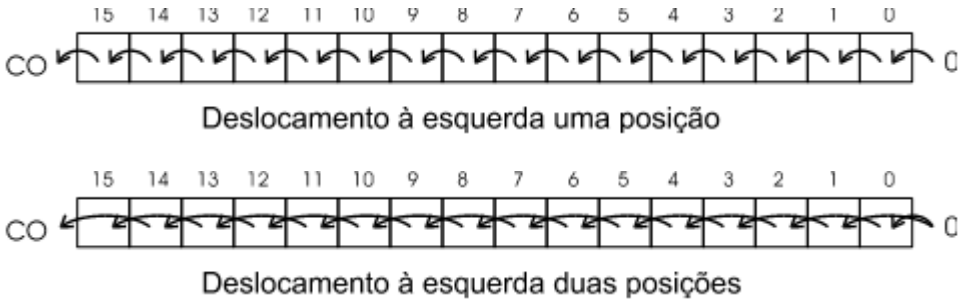
Este bloco efetua a operação de deslocamento com carry à esquerda do Operando1 pelo número de deslocamentos indicado pelo Operando2, e Resultado recebe o resultado da operação.



Observe que a operação SLC (shift left carry) é feita bit a bit com os 16 bits do Operando1, segundo o número de deslocamentos especificado por Operando2. Assim, se Operando2 igual a

um o Operando1 será deslocado em uma posição para a esquerda. Já se Operando2 igual a dois o Operando1 será deslocado em duas posições para a esquerda, e assim sucessivamente.

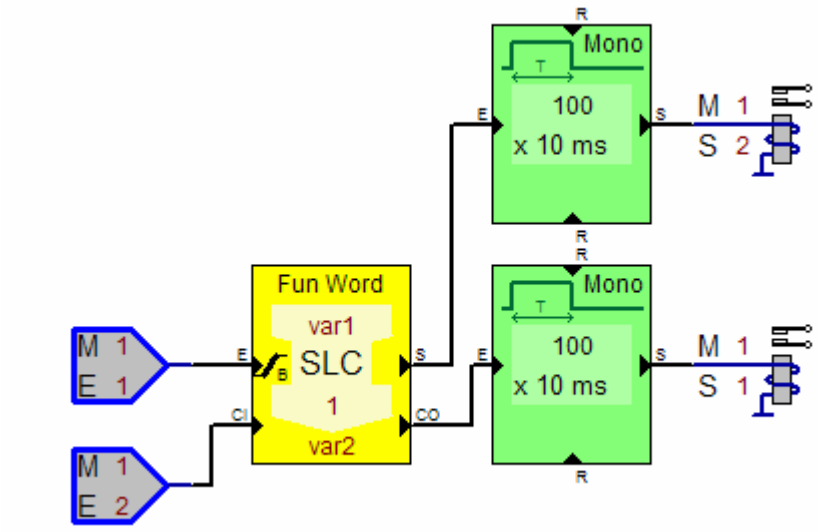
Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Note que os bits LSB são preenchidos com o valor do nodo de entrada Carry In (CI), enquanto o bit MSB é disponibilizado no nodo de saída Carry Out (CO).



O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo).

Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Operação SLC.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de deslocamento à esquerda com carry. No caso, é efetuada a operação SLC em uma posição na variável var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for energizada (já que a entrada E do bloco de atribuição está selecionada por borda). O resultado da operação é guardado em var2. Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação SLC com deslocamento unitário irá resultar no valor 6998 (1B56h) na variável var2, se o nodo CI for

mantido desligado, ou irá resultar no valor 6999 (1B57h) se este nodo estiver ligado.

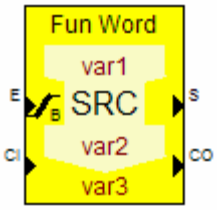
Já a saída S2 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco, assim como o nodo CO. A entrada E do bloco SLC está selecionada para acionamento por borda e, portanto, a saída S e CO ficarão ativas durante apenas um ciclo de execução do programa aplicativo (<500µs). Então, foram inseridos monoestáveis para aumentar os pulsos de saída para um segundo, de forma a serem visíveis.

> **Atenção:** A partir da versão 3.61 do controlador µDX201 existe suporte para blocos Word com múltiplas variáveis (8 words). Nesses blocos se usa uma variável absoluta para apontar o word inicial, e o bloco efetuada a operação nessa variável e nas 7 subseqüentes. Por exemplo, ao efetuar uma operação SLC de v20 usando o bloco de **Deslocamento com Carry à Esquerda 8 Words**, ao ativar o bloco será feita a operação SLC word a word de v20,v21,v22,v23,v24,v25,v26,v27.

Veja também:
ARITMÉTICA WORD - Operação SRC
ARITMÉTICA INTEIRA - Operação SLC
ARITMÉTICA INTEIRA - Operação SRC
ARITMÉTICA INTEIRA - Operação SLA
ARITMÉTICA INTEIRA - Operação SRA

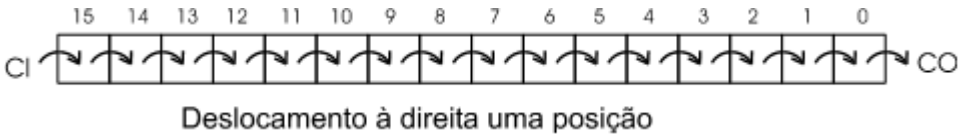
ARITMÉTICA WORD - Operação SRC

Este bloco efetua a operação de deslocamento com carry à direita do Operando1 pelo número de deslocamentos indicado pelo Operando2, e Resultado recebe o resultado da operação.

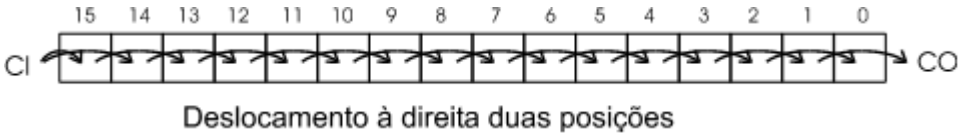


Observe que a operação SRC (shift right carry) é feita bit a bit com os 16 bits do Operando1, segundo o número de deslocamentos especificado por Operando2. Assim, se Operando2 igual a um o Operando1 será deslocado em uma posição para a direita. Já se Operando2 igual a dois o Operando1 será deslocado em duas posições para a direita, e assim sucessivamente.

Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Note que os bits MSB são preenchidos com o estado do nodo de entrada Carry In (CI), enquanto o bit LSB é disponibilizado no nodo de saída Carry Out (CO).



Deslocamento à direita uma posição

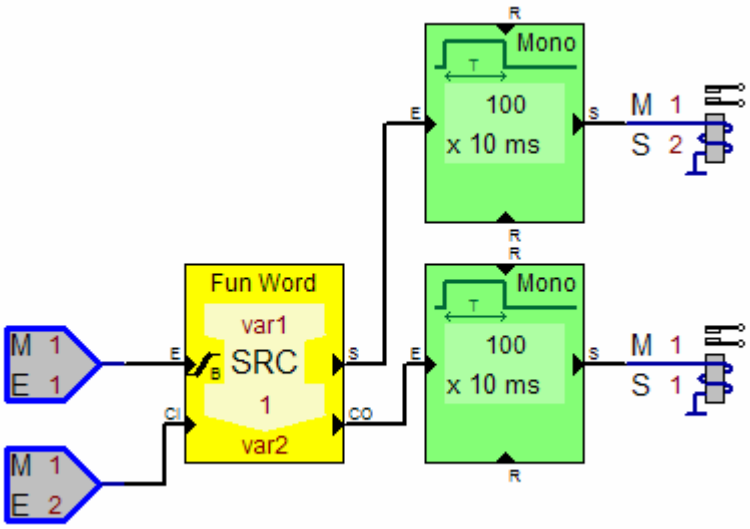


Deslocamento à direita duas posições

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo).

Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Operação SRC.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de deslocamento com carry à direita. No caso, é efetuada a operação SRC em uma posição na variável var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for energizada (já que a entrada E do bloco de atribuição está selecionada por borda). O resultado da operação é guardado em var2. Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação SRC com deslocamento unitário irá resultar no valor 1749 (6D5h) na variável var2, caso o nodo de Carry In seja mantido desligado. Já se ele estiver acionado irá entrar um bit MSB igual a um, e o resultado será 34517 (86D5h).

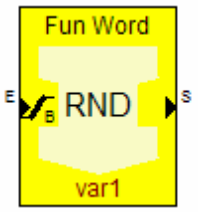
Já a saída S2 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco, assim como o nodo CO. A entrada E do bloco SRC está selecionada para acionamento por borda e, portanto, a saída S e CO ficarão ativas durante apenas um ciclo de execução do programa aplicativo (<500µs). Então, foram inseridos monoestáveis para aumentar os pulsos de saída para um segundo, de forma a serem visíveis.

Atenção: A partir da versão 3.61 do controlador µDX201 existe suporte para blocos Word com múltiplas variáveis (8 words). Nesses blocos se usa uma variável absoluta para apontar o word inicial, e o bloco efetuada a operação nessa variável e nas 7 subseqüentes. Por exemplo, ao efetuar uma operação SRC de v20 usando o bloco de **Deslocamento com Carry à Direita 8 Words**, ao ativar o bloco será feita a operação SRC word a word de v20,v21,v22,v23,v24,v25,v26,v27.

Veja também:
ARITMÉTICA WORD - Operação SLC
ARITMÉTICA INTEIRA - Operação SLC
ARITMÉTICA INTEIRA - Operação SRC
ARITMÉTICA INTEIRA - Operação SLA
ARITMÉTICA INTEIRA - Operação SRA

ARITMÉTICA WORD - Operação Randômica

Este bloco gera um valor randômico word no operando especificado.



Resultado é o operando cujo valor será modificado de forma randômica pelo Controlador µDX200.

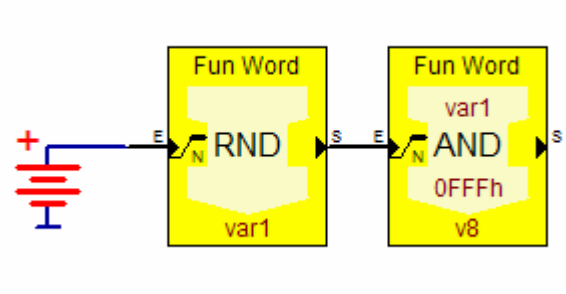
Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Caso o nodo de Entrada (E) esteja sensível ao nível, ao energizá-lo o µDX200 irá gerar um novo número randômico a cada ciclo de execução do programa aplicativo (o que pode atingir milhares de vezes por segundo). Se o nodo de Entrada (E) estiver selecionado para ser sensível a borda um novo número aleatório será gerado a cada borda de subida do nodo E.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

Este bloco pode ser útil para introduzir um componente de aleatoriedade em um programa, evitando assim laços infinitos (dead-lock) em programas realimentados.

Note que os valores gerados, por serem variável word, estão contidos no intervalo de números entre 0 e 65535.



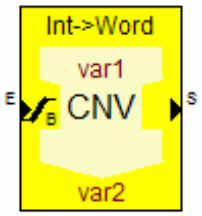
Exemplo de Programa Aplicativo: **Aritmética Word - Operação Randômica.dxg**

O programa acima gera números inteiros aleatórios entre 0 e 4095 (limites de operação das saídas analógicas de 12 bits) na saída analógica S1. Com isso obtêm-se um sinal de ruído branco nesta saída. Note que para manter o valor aleatório entre 0 e 4095 (e não entre 0 e 65535, como é gerado pelo bloco RND) foi utilizada uma operação AND com 0FFFh (4095), de forma a zerar os 4 bits MSB do número randômico gerado.

Veja também:
ARITMÉTICA WORD - Operação Randômica Var
ARITMÉTICA INTEIRA - Operação Randômica
ARITMÉTICA INTEIRA - Operação Randômica Var

ARITMÉTICA WORD - Conversão Inteiro -> Word

Este bloco transfere o valor de um operando inteiro para um segundo operando word.



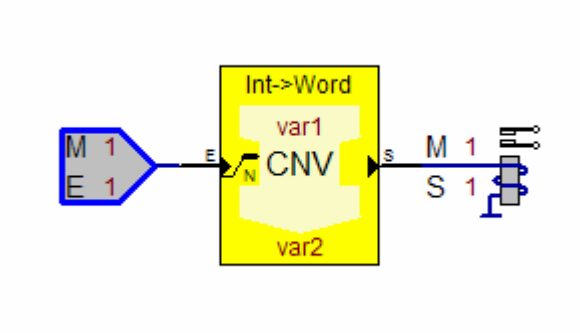
Operando é o operando cujo valor será transferido para Resultado.

Note que Operando pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso. Como os números inteiros no µDX200 são representados em complemento de dois, números inteiros negativos geram conversões para números word com valor superior a 7FFFh (32767).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Conversão Inteiro - Word.dxg**

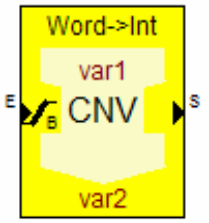
O exemplo acima transfere o valor da variável inteira var1 para a variável word var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de conversão está selecionada por nível).

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de conversão inteiro à word. A entrada E do bloco de conversão está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA WORD - Conversão Word -> Inteiro
ARITMÉTICA WORD - Conversão Int -> Word
ARITMÉTICA WORD - Conversão Word -> Int

ARITMÉTICA WORD - Conversão Word -> Inteiro

Este bloco transfere o valor de um operando word para um segundo operando inteiro.



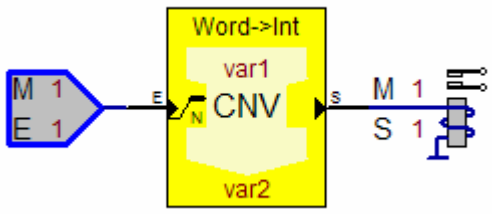
Operando é o operando cujo valor será transferido para Resultado.

Note que Operando pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre pode ser concluída com sucesso. Como os números inteiros no µDX200 são representados em complemento de dois, caso o valor em word seja superior a 7FFFh (32767) o valor convertido para inteiro será negativo.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Conversão Word - Inteiro.dxg**

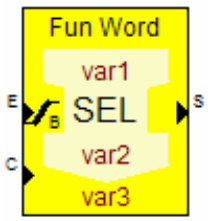
O exemplo acima transfere o valor da variável word var1 para a variável inteira var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de conversão está selecionada por nível).

Já a saídas S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de conversão word à Inteiro. A entrada E do bloco está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA WORD - Conversão Inteiro -> Word
ARITMÉTICA WORD - Conversão Int -> Word
ARITMÉTICA WORD - Conversão Word -> Int

ARITMÉTICA WORD - Seletor

Este bloco transfere o valor de um ou outro operando word para um terceiro operando word.



Operando1 e Operando2 são os operandos que serão transferidos para Resultado, conforme o estado do nodo de Controle (C).

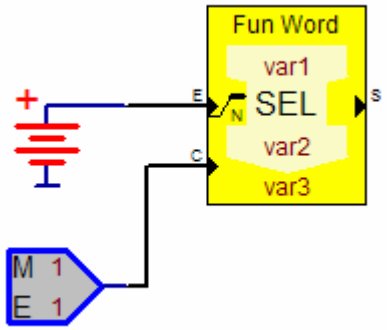
Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

O nodo de Controle (C) especifica qual dos operandos será transferido para Resultado. Se C=0 o Operando1 é transferido para Resultado. Já se C=1 o Operando2 é transferido para Resultado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



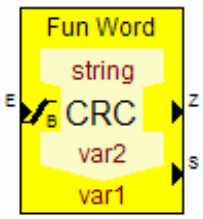
Exemplo de Programa Aplicativo: **Aritmética Word - Seletor.dxg**

Acima temos um pequeno exemplo de uso do bloco de seleção word. No caso, é atribuído o valor da variável var1 ou da variável var2 na variável word var3, conforme o estado do nodo de Controle (C). Se a entrada E1 estiver desenergizada var3 irá assumir o valor de var1. Já se a entrada E1 estiver energizada var3 irá assumir o valor de var2. O nodo de Entrada (E) está sempre ativado, pois está selecionado para acionamento por nível e está ligado constantemente pelo bloco de energia. O nodo de Controle (C) é comando pela entrada E1 do módulo 1 de Expansão de Entradas/Saídas (µDX210).

Veja também:
ARITMÉTICA INTEIRA - Seletor
ARITMÉTICA LONGINT - Seletor
ARITMÉTICA REAL - Seletor
GERAL - Seletor de Variável Inteira
GERAL - Seletor de Variável LongInt
GERAL - Seletor de Variável Word
GERAL - Seletor de Variável Real

ARITMÉTICA WORD - Verifica CRC-16

Calcula o CRC-16 (cyclic redundancy check) sobre o string especificado. Offset indica quantos bytes a partir do início do buffer devem ser desconsiderados, e Núm.Bytes indica quantos bytes devem ser analisados. O nodo Z liga quando o resultado for zero (note que uma mensagem com CRC-16 correto resulta em resultado zero). O valor calculado de CRC é transferido para Resultado.



String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Offset indica a partir de qual posição do string é iniciada a conversão.

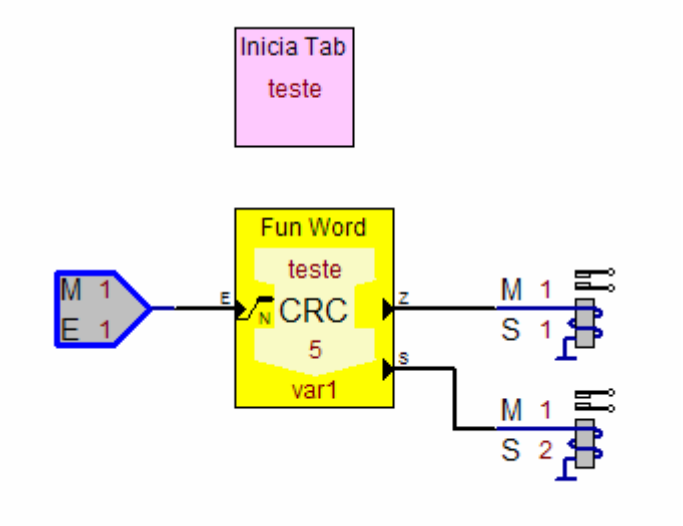
Note que String, Offset e Núm.Bytes podem ser variáveis word ou inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

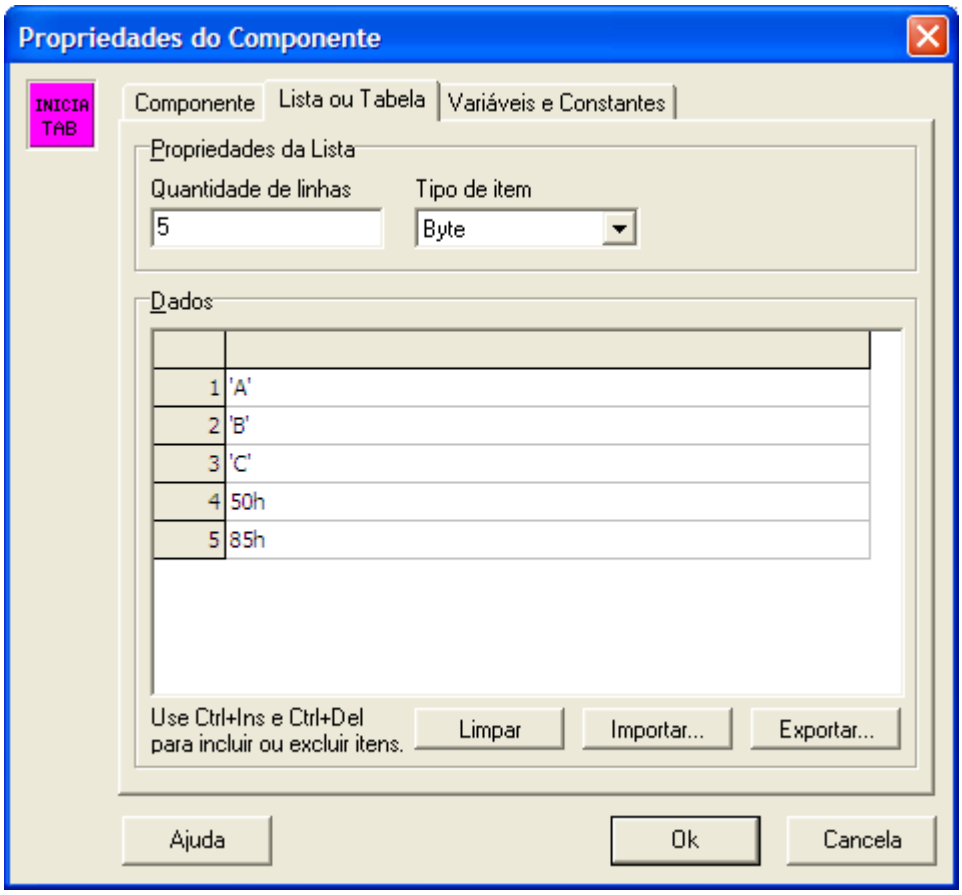
O nodo de Zero (Z) indica que o CRC calculado resultou em zero. Note que uma mensagem com CRC incluído nela sempre resulta em CRC nulo, ou seja, este nodo é ligado caso a mensagem analisada em String esteja correta.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Verifica CRC-16.dxg**

Acima temos um pequeno exemplo de uso do bloco de verificação de CRC. O programa analisa o string teste ("ABC"+CRC16) e liga o nodo de saída Zero (Z), já que o CRC confere com o conteúdo do string. Note que utilizou-se Offset =0 para o bloco (de forma que ele inicie a verificação a partir da primeira posição do string teste. Os valores inicializados no string teste via bloco de Inicia Tab são:

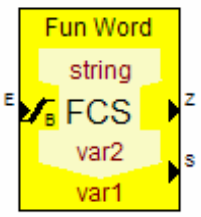


Note que o valor 8550h corresponde ao CRC-16 do string "ABC". Note que o µDX200 pressupõe sempre o byte menos significativo do CRC-16 antes do byte mais significativo. Por isso, o CRC-16 incluso ao final do string "ABC" é 5085h e não 8550h. Esta característica foi implementada porque tanto o protocolo nativo do µDX200 quanto o protocolo ModBus-RTU usam esta ordem para transmissão de CRC-16 (byte LSB, byte MSB).

Veja também:
ARITMÉTICA WORD - Verifica FCS
ARITMÉTICA WORD - Verifica CRC-DNP

ARITMÉTICA WORD - Verifica FCS

Calcula o FCS (frame check sequence) sobre o string especificado. Offset indica quantos bytes a partir do início do buffer devem ser desconsiderados, e Núm.Bytes indica quantos bytes devem ser analisados. O nodo Z liga quando o resultado for zero (note que uma mensagem com FCS correto resulta em resultado zero). O valor calculado de FCS é transferido para Resultado.



String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Offset indica a partir de qual posição do string é iniciada a conversão.

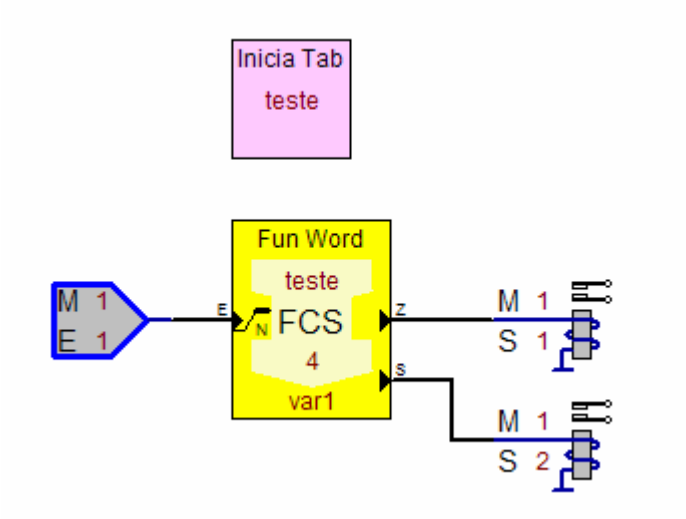
Note que String, Offset e Núm.Bytes podem ser variáveis word ou inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

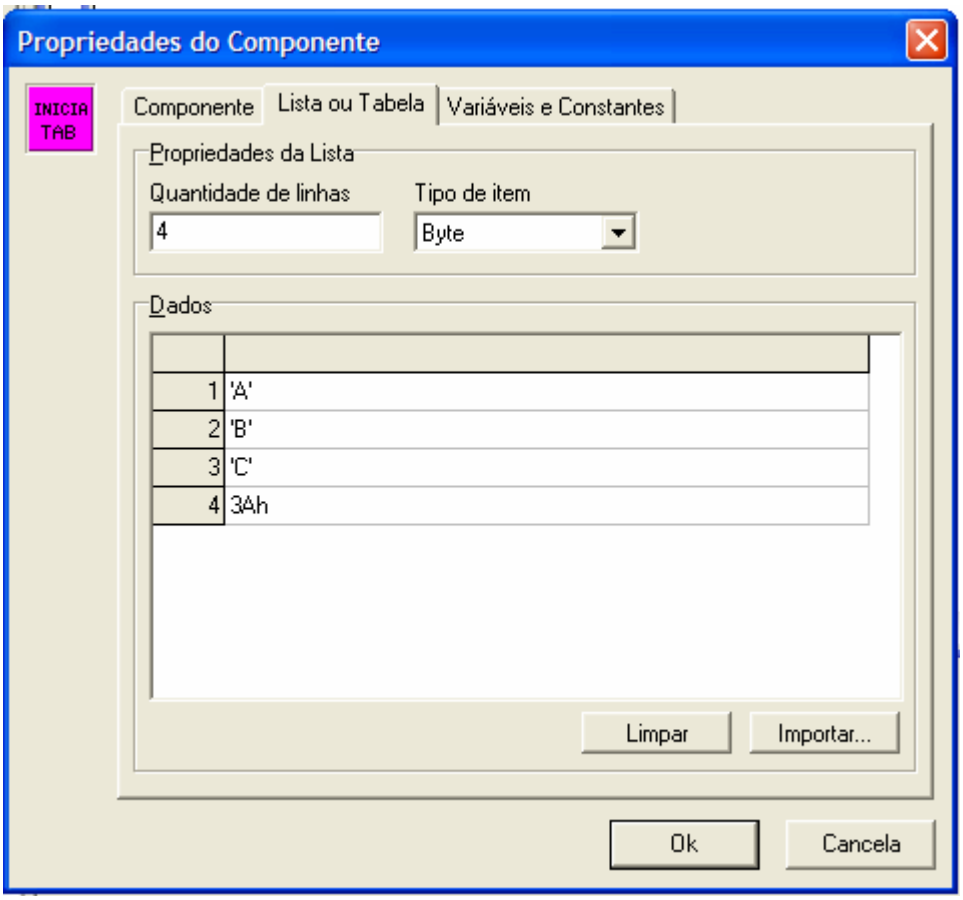
O nodo de Zero (Z) indica que o FCS calculado resultou em zero. Note que uma mensagem com FCS incluído nela sempre resulta em FCS nulo, ou seja, este nodo é ligado caso a mensagem analisada em String esteja correta.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Word - Verifica FCS.dxg**

Acima temos um pequeno exemplo de uso do bloco de verificação de FCS. O programa analisa o string teste ("ABC"+FCS) e liga o nodo de saída Zero (Z), já que o FCS confere com o conteúdo do string. Note que utilizou-se Offset =0 para o bloco (de forma que ele inicie a verificação a partir da primeira posição do string teste. Os valores inicializados no string teste via bloco de Inicia Tab são:

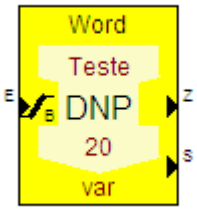


Note que a representação ASCII de 'A' é 65 (41h), de 'B' é 66 (42h), e de 'C' é 67 (43h). Então, a soma de todos os bytes do string resultam no FCS, considerando-se apenas o byte LSB da soma (até 255, portanto). Ora, 41h+42h+43h+3Ah = 100h. Desprezando o byte MSB, resulta em soma igual a zero. Ou seja, o FCS do string resulta em zero, o que indica que 3Ah é o FCS correto para o string "ABC".

Veja também:
ARITMÉTICA WORD - Verifica CRC-16
ARITMÉTICA WORD - Verifica CRC-DNP

ARITMÉTICA WORD - Verifica CRC-DNP

Calcula o CRC-16 (cyclic redundancy check) sobre o string especificado, conforme especificação do protocolo DNP-3. Offset indica quantos bytes a partir do início do buffer devem ser desconsiderados, e Núm.Bytes indica quantos bytes devem ser analisados. O nodo Z liga quando o resultado for zero (note que uma mensagem com CRC-16 correto resulta em resultado zero). O valor calculado de CRC é transferido para Resultado.



String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Offset indica a partir de qual posição do string é iniciada a conversão.

Note que String, Offset e Núm.Bytes podem ser variáveis word ou inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

O nodo de Zero (Z) indica que o CRC calculado resultou em zero. Note que uma mensagem com CRC incluído nela sempre resulta em CRC nulo, ou seja, este nodo é ligado caso a mensagem analisada em String esteja correta.

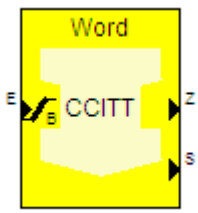
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX201 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Atenção: Bloco disponível apenas em controlador µDX201.

Veja também:
ARITMÉTICA WORD - Verifica CRC-16
ARITMÉTICA WORD - Verifica FCS

ARITMÉTICA WORD - Verifica CRC-CCITT

Calcula o CRC-16 (cyclic redundancy check) sobre o string especificado, conforme especificação CCITT. Offset indica quantos bytes a partir do início do buffer devem ser desconsiderados, e Núm.Bytes indica quantos bytes devem ser analisados. O nodo Z liga quando o resultado for zero (note que uma mensagem com CRC-16 correto resulta em resultado zero). O valor calculado de CRC é transferido para Resultado.



String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Offset indica a partir de qual posição do string é iniciada a conversão.

Note que String, Offset e Núm.Bytes podem ser variáveis word ou inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

O nodo de Zero (Z) indica que o CRC calculado resultou em zero. Note que uma mensagem com CRC incluído nela sempre resulta em CRC nulo, ou seja, este nodo é ligado caso a mensagem analisada em String esteja correta.

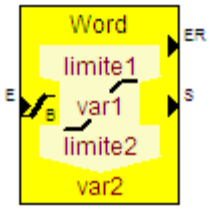
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX201 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Atenção: Bloco disponível apenas em controlador µDX201 (µDX200 somente v2.24 ou superior).

Veja também:
ARITMÉTICA WORD - Verifica CRC-16
ARITMÉTICA WORD - Verifica FCS

ARITMÉTICA WORD - Operação Limite

Permite limitar uma variável word entre dois valores especificados. Assim, se garante que a variável assumirá apenas valores entre os dois limites. Este bloco somente é operacional em controladores µDX201 versão 3.37 ou superior.



Operando é a variável cujo valor será limitado entre os limites estabelecidos por Lim.Superior e Lim.Inferior e transferido para Resultado.

Note que Operando, Lim.Superior e Lim.Inferior podem ser variáveis word, constantes (valor numérico em decimal ou hexadecimal), ou ainda referências a uma posição de memória. Já Resultado deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Variáveis word podem assumir valores entre 0 e 65535.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que Operando possui valor fora dos limites impostos por Lim.Superior e Lim.Inferior.

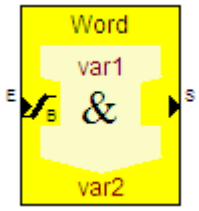
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Limite
ARITMÉTICA INTEIRA - Operação Limite Var
ARITMÉTICA LONGINT - Operação Limite
ARITMÉTICA LONGINT - Operação Limite Var
ARITMÉTICA WORD - Operação Limite Var
ARITMÉTICA REAL - Operação Limite
ARITMÉTICA REAL - Operação Limite Var

ARITMÉTICA WORD - Operação Pointer

Este bloco retorna na variável inferior o endereço de memória da variável especificada. Este bloco somente é operacional em controladores µDX201 versão 3.52 ou superior.



Operando1 é a variável cujo endereço será transferido para Resultado.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

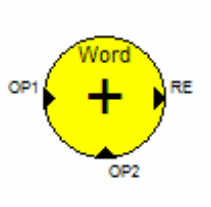
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.52 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Pointer Var
ARITMÉTICA INTEIRA - Operação Pointer
ARITMÉTICA WORD - Operação Pointer Var

ARITMÉTICA WORD - Adição Var

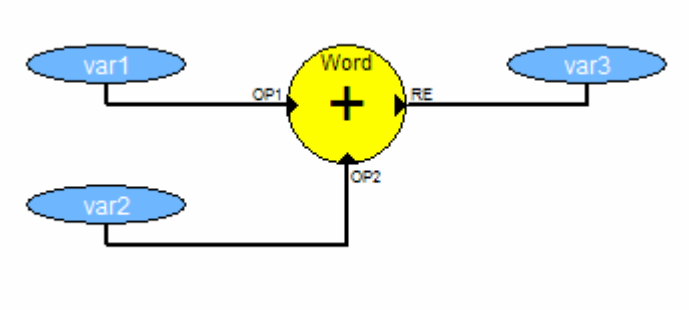
Este bloco efetua a adição de dois operandos word, colocando o resultado em um terceiro operando word. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos word a serem adicionados e RE recebe o resultado da adição.

Note que OP1 e OP2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis word podem assumir valores entre 0 e 65535. Assim, por exemplo, se somarmos duas variáveis, ambas com valor 40000 o resultado será excessivo (80000) para representação via word e ocorrerá carry. Só que o sinal de Carry Out (CO), ao contrário do bloco de adição word retangular, não está disponível neste bloco. O resultado da adição acima será 80000-65536 = 14464.



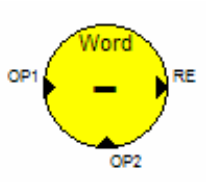
Exemplo de Programa Aplicativo: **Aritmética Word - Adição Var.dxg**

No exemplo de uso do bloco de adição var inteira a variável var1 é somada a variável var2 constantemente, e o resultado da operação é colocado em var3. Note que as variáveis são transportadas para o bloco de adição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis word (0 a 65535) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA WORD - Adição
ARITMÉTICA INTEIRA - Adição
ARITMÉTICA INTEIRA - Adição Var
ARITMÉTICA LONGINT - Adição
ARITMÉTICA LONGINT - Adição Var
ARITMÉTICA REAL - Adição
ARITMÉTICA REAL - Adição Var

ARITMÉTICA WORD - Subtração Var

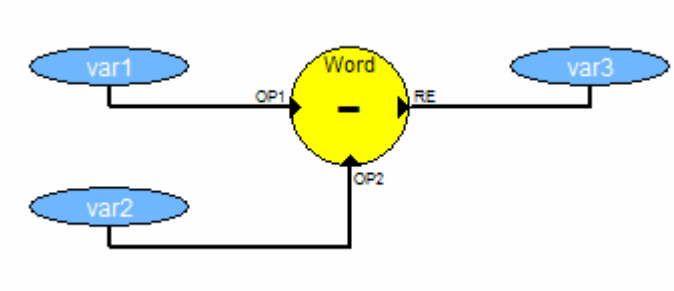
Este bloco efetua a subtração de dois operandos word, colocando o resultado em um terceiro operando word. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos word a serem subtraídos e RE recebe o resultado da subtração.

Note que OP1 e OP2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis word podem assumir valores entre 0 e 65535. Assim, por exemplo, se subtrairmos duas variáveis, uma de valor 20000 e outra de valor 30000 o resultado será negativo (-10000), sem representação em word. Neste caso o resultado será 65536-10000= 55536.



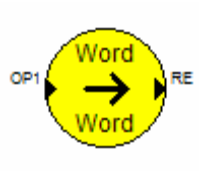
Exemplo de Programa Aplicativo: **Aritmética Word - Subtração Var.dxg**

No pequeno exemplo de uso do bloco de subtração var inteira a variável var2 é subtraída da variável var1 constantemente, e o resultado da operação é colocado em var3. Note que as variáveis são transportadas para o bloco de subtração pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis word (0 a 65535) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA WORD - Subtração
ARITMÉTICA INTEIRA - Subtração
ARITMÉTICA INTEIRA - Subtração Var
ARITMÉTICA LONGINT - Subtração
ARITMÉTICA LONGINT - Subtração Var
ARITMÉTICA REAL - Subtração
ARITMÉTICA REAL - Subtração Var

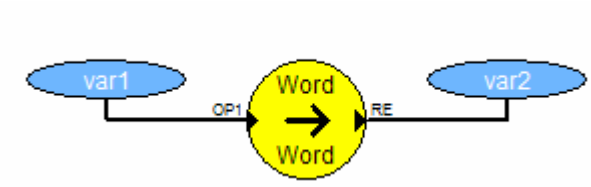
ARITMÉTICA WORD - Atribuição Var

Este bloco transfere o valor de um operando word para um segundo operando word. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é o operando word a transferido para RE.

Note que OP1 pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



Exemplo de Programa Aplicativo: **Aritmética Word - Atribuição Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de atribuição var word. No caso, é transferido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis word (0 a 65535) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

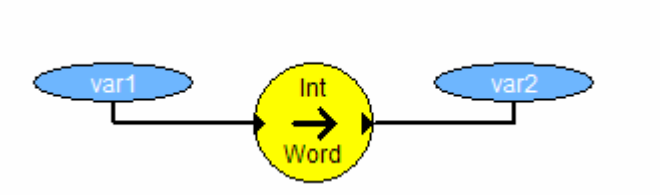
Veja também:
ARITMÉTICA WORD - Atribuição
ARITMÉTICA INTEIRA - Atribuição
ARITMÉTICA INTEIRA - Atribuição Var
ARITMÉTICA LONGINT - Atribuição
ARITMÉTICA LONGINT - Atribuição Var
ARITMÉTICA REAL - Atribuição
ARITMÉTICA REAL - Atribuição Var

ARITMÉTICA WORD - Conversão Int -> Word Var

Este bloco converte o valor de um operando inteiro para um segundo operando word. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

OP1 é o operando inteiro a ser transferido para o operando word RE.

Note que OP1 pode ser variável inteiro, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



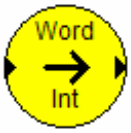
Exemplo de Programa Aplicativo: **Aritmética Word - Conversão Inteiro - Word Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de conversão var inteiro à word. No caso, é transferido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (entre -32.768 e 32.767) e word (entre 0 e 65.535) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA WORD - Conversão Word -> Int
ARITMÉTICA WORD - Conversão Inteiro -> Word
ARITMÉTICA WORD - Conversão Word -> Inteiro

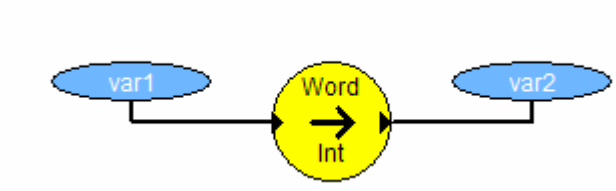
ARITMÉTICA WORD - Conversão Word -> Int Var

Este bloco converte o valor de um operando word para um segundo operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é o operando word a ser transferido para o operando inteiro RE.

Note que OP1 pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



Exemplo de Programa Aplicativo: **Aritmética Word - Conversão Word - Inteiro Var.dxg**

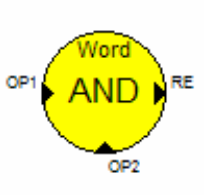
Acima temos um pequeno exemplo de uso do bloco de conversão var word à inteiro. No caso, é transferido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (entre -32.768 e 32.767) e word (entre 0 e 65.535) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA WORD - Conversão Int -> Word

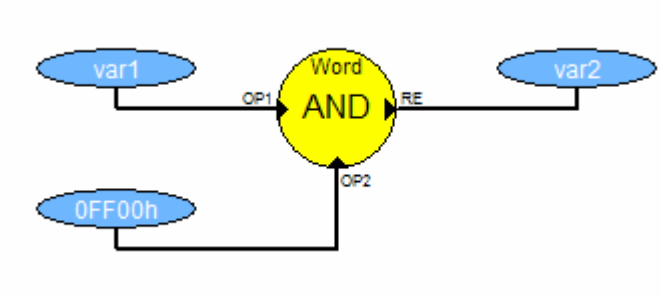
ARITMÉTICA WORD - Conversão Inteiro -> Word
ARITMÉTICA WORD - Conversão Word -> Inteiro

ARITMÉTICA WORD - Operação AND Var

Este bloco efetua a operação booleana AND entre dois operandos word, e RE recebe o resultado da operação. Observe que a operação AND é feita bit a bit com os 16 bits dos operandos. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



Note que OP1 e OP2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



Exemplo de Programa Aplicativo: **Aritmética Word - Operação AND Var.dxg**

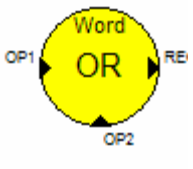
Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana AND word. No caso, é efetuada a operação AND entre a variável var1 e a constante hexadecimal 0FF00h constantemente. O resultado da operação é guardado em var2. Note que esta operação zera o byte LSB de var1 (já que qualquer valor AND com 00h resulta em zero). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação AND com 0FF00h irá resultar no valor 3328 (0D00h) na variável var2.

Veja também:
ARITMÉTICA WORD - Operação AND
ARITMÉTICA INTEIRA - Operação AND
ARITMÉTICA INTEIRA - Operação AND Var
ARITMÉTICA LONGINT - Operação AND
ARITMÉTICA LONGINT - Operação AND Var

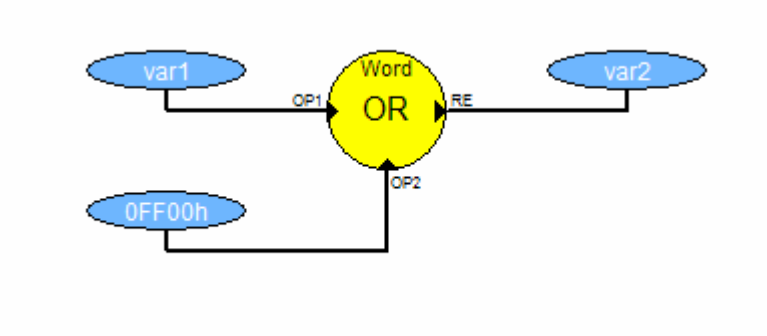
ARITMÉTICA WORD - Operação OR Var

Este bloco efetua a operação booleana OR entre dois operandos word, e RE recebe o resultado da operação. Observe que a operação OR é feita bit a bit com os 16 bits dos operandos. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente

via edição do bloco, como no caso de blocos retangulares).



Note que OP1 e OP2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



Exemplo de Programa Aplicativo: **Aritmética Word - Operação OR Var.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana OR word. No caso, é efetuada a operação OR entre a variável var1 e a constante hexadecimal 0FF00h constantemente. O resultado da operação é guardado em var2. Note que esta operação liga todos os bits do byte MSB de var1 (já que qualquer valor OR com 0FFh resulta em um). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação OR com 0FF00h irá resultar no valor 65451 (0FFABh) na variável var2.

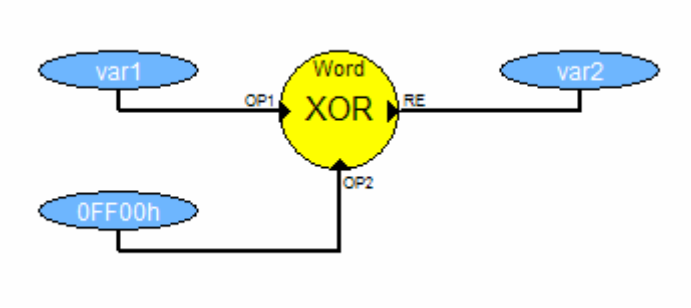
Veja também:
ARITMÉTICA WORD - Operação OR
ARITMÉTICA INTEIRA - Operação OR
ARITMÉTICA INTEIRA - Operação OR Var
ARITMÉTICA LONGINT - Operação OR
ARITMÉTICA LONGINT - Operação OR Var

ARITMÉTICA WORD - Operação XOR Var

Este bloco efetua a operação booleana XOR entre dois operandos word, e RE recebe o resultado da operação. Observe que a operação XOR é feita bit a bit com os 16 bits dos operandos. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

Note que OP1 e OP2 podem ser variáveis word, constantes (valores numéricos em decimal ou

hexadecimal), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



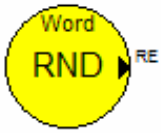
Exemplo de Programa Aplicativo: **Aritmética Word - Operação XOR Var.dxg**

Na figura acima está representado um pequeno exemplo de uso do bloco de operação booleana XOR word. No caso, é efetuada a operação XOR entre a variável var1 e a constante hexadecimal 0FF00h constantemente. O resultado da operação é guardado em var2. Note que esta operação inverte todos os bits do byte MSB de var1 (já que qualquer valor XOR com 0FFh resulta em inversão bit a bit). Por exemplo, se var1 possui valor de 3499 (0DABh) ao efetuar a operação XOR com 0FF00h irá resultar no valor de 62123 (0F2ABh) na variável var2.

Veja também:
ARITMÉTICA WORD - Operação XOR
ARITMÉTICA INTEIRA - Operação XOR
ARITMÉTICA INTEIRA - Operação XOR Var
ARITMÉTICA LONGINT - Operação XOR
ARITMÉTICA LONGINT - Operação XOR Var

ARITMÉTICA WORD - Operação Randômica Var

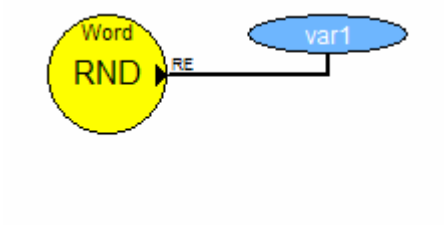
Este bloco gera um valor randômico word no operando especificado.



RE é o operando cujo valor será modificado de forma randômica pelo Controlador µDX200. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

RE deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Este bloco pode ser útil para introduzir um componente de aleatoriedade em um programa, evitando assim laços infinitos (dead-lock) em programas realimentados.



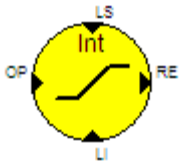
Exemplo de Programa Aplicativo: **Aritmética Word - Operação Randômica Var.dxg**

O programa de exemplo acima gera números aleatórios continuamente e os transfere para a variável var1. Note que estes números são gerados dentro da faixa admitida pelos números word (entre 0 e 65535).

Veja também:
ARITMÉTICA WORD - Operação Randômica
ARITMÉTICA INTEIRA - Operação Randômica
ARITMÉTICA INTEIRA - Operação Randômica Var

ARITMÉTICA WORD - Operação Limite Var

Permite limitar uma variável word entre dois valores especificados. Assim, se garante que a variável assumirá apenas valores entre os dois limites. Este bloco somente é operacional em controladores µDX201 versão 3.37 ou superior. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP (operando) é a variável cujo valor será limitado entre os limites estabelecidos por LS (limite superior) e LI (limite inferior) e transferido para RE (resultado).

Note que OP, LS e LI podem ser variáveis word, constantes (valor numérico em decimal ou hexadecimal), ou ainda referências a uma posição de memória. Já RE deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Variáveis word podem assumir valores entre 0 e 65535.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Limite
ARITMÉTICA INTEIRA - Operação Limite Var
ARITMÉTICA LONGINT - Operação Limite
ARITMÉTICA LONGINT - Operação Limite Var
ARITMÉTICA WORD - Operação Limite
ARITMÉTICA REAL - Operação Limite
ARITMÉTICA REAL - Operação Limite Var

ARITMÉTICA WORD - Operação Pointer Var

Permite retornar na conexão de saída o endereço na memória da variável especificada na conexão de entrada. Este bloco somente é operacional em controladores µDX201 versão 3.52 ou superior.

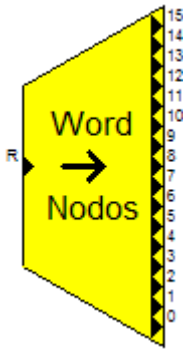
OP é a variável cujo endereço será transferido para RE.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.52 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Pointer
ARITMÉTICA WORD - Operação Pointer
ARITMÉTICA INTEIRA - Operação Pointer Var

ARITMÉTICA WORD - Conversão Word -> Nodos

Este bloco permite separar os 16 bits de uma variável word em 16 nodos distintos. Com isso, conforme os bits da variável word estejam ligados ou desligados os nodos correspondentes serão acionados ou não.



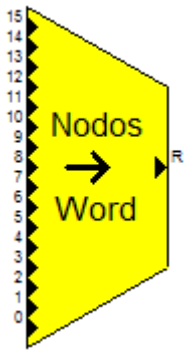
No caso deste tipo de bloco a variável word é especificada na conexão R, ou seja, o fio de conexão transporta a variável (em vez de ser especificada literalmente via edição do bloco).

Note que R pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já as conexão de 0 a 15 à esquerda do bloco são nodos (ou seja, variáveis binárias - assumem apenas dois estados, ligado ou desligado).

Para exemplo de aplicação veja bloco similar: Conversão Int -> Nodos.

ARITMÉTICA WORD - Conversão Nodos -> Word

Este bloco efetua a operação contrária ao bloco descrito anteriormente, ou seja, permite condensar até 16 nodos em uma variável word. Conforme os nodos de 0 a 15 são acionados ou não os correspondentes bits da variável word são ligados ou não.



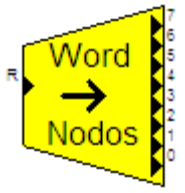
No caso deste tipo de bloco a variável é especificada na conexão, ou seja, o fio de conexão transporta a variável (em vez de serem especificadas literalmente via edição do bloco).

Note que as conexões de 0 a 15 são nodos, ou seja, variáveis binárias (assumem apenas dois valores, ligado ou desligado). Já R deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Para exemplo de aplicação veja bloco similar: Conversão Nodos -> Int.

ARITMÉTICA WORD - Conversão Word -> Nodos (8 bits)

Este bloco permite separar os 8 bits inferiores de uma variável word em 8 nodos distintos. Com isso, conforme os bits da variável word estejam ligados ou desligados os nodos correspondentes serão acionados ou não.



No caso deste tipo de bloco a variável word é especificada na conexão R, ou seja, o fio de conexão transporta a variável (em vez de ser especificada literalmente via edição do bloco).

Note que R pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou

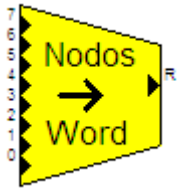
ainda uma referência a uma posição de memória. Já as conexão de 0 a 7 à esquerda do bloco são nodos (ou seja, variáveis binárias - assumem apenas dois estados, ligado ou desligado).

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Conversão Int -> Nodos (8 bits)
ARITMÉTICA INTEIRA - Conversão Nodos -> Int (8 bits)
ARITMÉTICA WORD - Conversão Nodos -> Word (8bits)

ARITMÉTICA WORD - Conversão Nodos -> Word (8 bits)

Este bloco efetua a operação contrária ao bloco descrito anteriormente, ou seja, permite condensar até 8 nodos em uma variável word (utilizando apenas os bits menos significativos da variável). Conforme os nodos de 0 a 7 são acionados ou não os correspondentes bits da variável word são ligados ou não.



No caso deste tipo de bloco a variável é especificada na conexão, ou seja, o fio de conexão transporta a variável (em vez de serem especificadas literalmente via edição do bloco).

Note que as conexões de 0 a 7 são nodos, ou seja, variáveis binárias (assumem apenas dois valores, ligado ou desligado). Já R deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

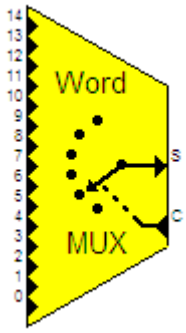
Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Conversão Int -> Nodos (8 bits)
ARITMÉTICA INTEIRA - Conversão Nodos -> Int (8 bits)
ARITMÉTICA WORD - Conversão Word -> Nodos (8 bits)

ARITMÉTICA WORD - Multiplexador

Este bloco permite selecionar um dos 15 nodos de entrada (0 a 14) a ser conectado ao nodo de saída (S). A seleção é feita conforme o valor da variável word conectada ao controle (C). É uma chave selecionadora com 15 posições, sendo que a posição da chave é determinada por uma variável word. Caso esta variável assuma valor maior que 14 ou menor que zero nenhum nodo é

conectado à saída S.

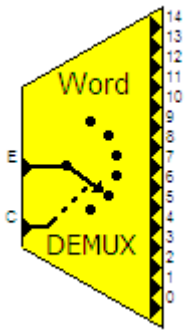


Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Multiplexador
ARITMÉTICA INTEIRA - Demultiplexador
ARITMÉTICA INTEIRA - Multiplexador (8 bits)
ARITMÉTICA INTEIRA - Demultiplexador (8 bits)
ARITMÉTICA WORD - Demultiplexador
ARITMÉTICA WORD - Multiplexador (8 bits)
ARITMÉTICA WORD - Demultiplexador (8 bits)

ARITMÉTICA WORD - Demultiplexador

Este bloco permite ligar o nodo de entrada E a um dos 15 nodos de saída (0 a 14) A seleção é feita conforme o valor da variável word conectada ao controle (C). É uma chave comutadora com 15 posições, sendo que a posição da chave é determinada por uma variável word. Caso esta variável assuma valor maior que 14 ou menor que zero a entrada E não é conectada a nenhuma saída.

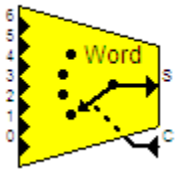


Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Multiplexador
ARITMÉTICA INTEIRA - Demultiplexador

ARITMÉTICA WORD - Multiplexador (8 bits)

Este bloco permite selecionar um dos 7 nodos de entrada (0 a 6) a ser conectado ao nodo de saída (S). A seleção é feita conforme o valor da variável word conectada ao controle (C). É uma chave selecionadora com 7 posições, sendo que a posição da chave é determinada por uma variável word. Caso esta variável assuma valor maior que 6 ou menor que zero nenhum nodo é conectado à saída S.

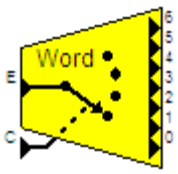


Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Multiplexador
ARITMÉTICA INTEIRA - Demultiplexador
ARITMÉTICA INTEIRA - Multiplexador (8 bits)
ARITMÉTICA INTEIRA - Demultiplexador (8 bits)
ARITMÉTICA WORD - Multiplexador
ARITMÉTICA WORD - Demultiplexador
ARITMÉTICA WORD - Demultiplexador (8 bits)

ARITMÉTICA WORD - Demultiplexador (8 bits)

Este bloco permite ligar o nodo de entrada E a um dos 7 nodos de saída (0 a 6) A seleção é feita conforme o valor da variável word conectada ao controle (C). É uma chave comutadora com 7 posições, sendo que a posição da chave é determinada por uma variável word. Caso esta variável assuma valor maior que 6 ou menor que zero a entrada E não é conectada a nenhuma saída.

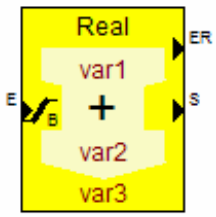


Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Multiplexador
ARITMÉTICA INTEIRA - Demultiplexador
ARITMÉTICA INTEIRA - Multiplexador (8 bits)
ARITMÉTICA INTEIRA - Demultiplexador (8 bits)
ARITMÉTICA WORD - Multiplexador
ARITMÉTICA WORD - Demultiplexador
ARITMÉTICA WORD - Multiplexador (8 bits)

ARITMÉTICA REAL - Adição

Este bloco efetua a adição de dois operandos reais, colocando o resultado em um terceiro operando real.



Operando1 e Operando2 são os operandos a serem adicionados e Resultado recebe o resultado da adição.

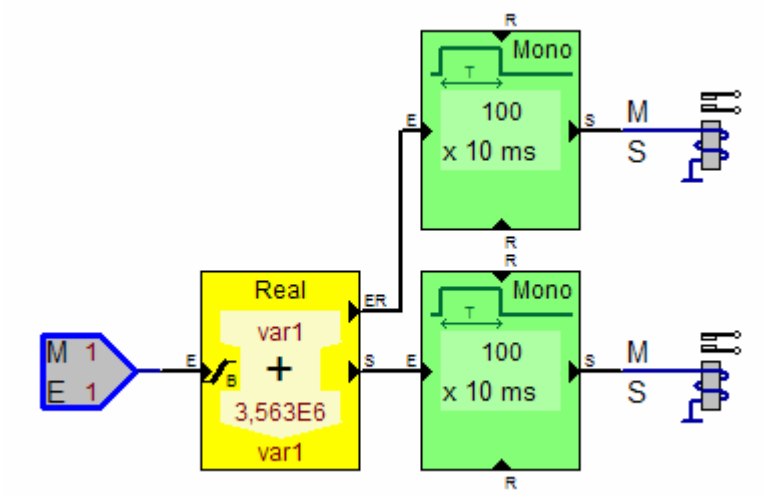
Note que Operando1 e Operando2 podem ser variáveis reais, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow ou underflow) na operação. Note que variáveis reais podem assumir valores entre -3,4E38 e 3,4E38 (note a convenção para especificação do expoente: 1E5=100000, 1E-3=0,001). Assim, por exemplo, se somarmos duas variáveis, ambas com valor 3E38 o resultado será excessivo (6E38) para representação via reais do µDX200 e o nodo de erro será ativado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

> *Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.*



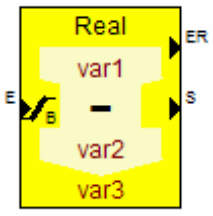
Exemplo de Programa Aplicativo: **Aritmética Real - Adição.dxg**

Acima temos um pequeno exemplo de uso do bloco de adição real. No caso, a variável var1 é incrementada de 3563000 unidades cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de adição está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de adição real. Note que foram colocados monoestáveis nas saídas S e ER do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

Veja também:
ARITMÉTICA REAL - Adição Var
ARITMÉTICA INTEIRA - Adição
ARITMÉTICA INTEIRA - Adição Var
ARITMÉTICA LONGINT - Adição
ARITMÉTICA LONGINT - Adição Var
ARITMÉTICA WORD - Adição
ARITMÉTICA WORD - Adição Var

ARITMÉTICA REAL - Subtração

Este bloco efetua a subtração de dois operandos reais, colocando o resultado em um terceiro operando real.



Operando1 e Operando2 são os operandos a serem subtraídos e Resultado recebe o resultado da subtração.

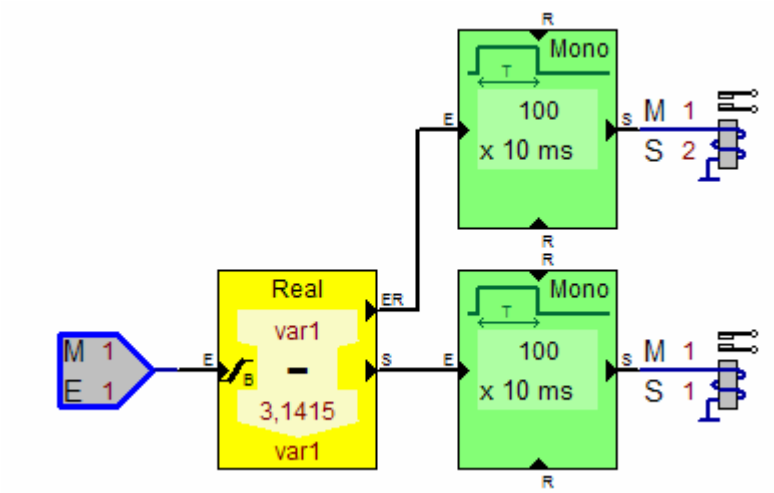
Note que Operando1 e Operando2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow ou underflow) na operação. Note que variáveis reais podem assumir valores entre -3,4E38 e 3,4E38. Assim, por exemplo, se subtrairmos duas variáveis, a primeira com valor 1E38 e a segunda com valor -3E38 o resultado será excessivo (4E38) para representação via reais do µDX200 e o nodo de erro será ativado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

> *Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.*



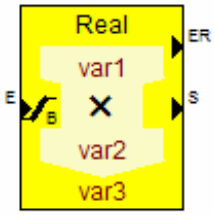
Exemplo de Programa Aplicativo: **Aritmética Real - Subtração.dxg**

Acima temos um pequeno exemplo de uso do bloco de subtração real. No caso, a variável var1 é decrementada do valor 3,1415 (aproximadamente p) unidades cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de subtração está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de subtração real. Note que foram colocados monoestáveis nas saídas S e ER do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

Veja também:
ARITMÉTICA REAL - Subtração Var
ARITMÉTICA INTEIRA - Subtração
ARITMÉTICA INTEIRA - Subtração Var
ARITMÉTICA LONGINT - Subtração
ARITMÉTICA LONGINT - Subtração Var
ARITMÉTICA WORD - Subtração
ARITMÉTICA WORD - Subtração Var

ARITMÉTICA REAL - Multiplicação

Este bloco efetua a multiplicação de dois operandos reais, colocando o resultado em um terceiro operando real.



Operando1 e Operando2 são os operandos a serem multiplicados e Resultado recebe o resultado da multiplicação.

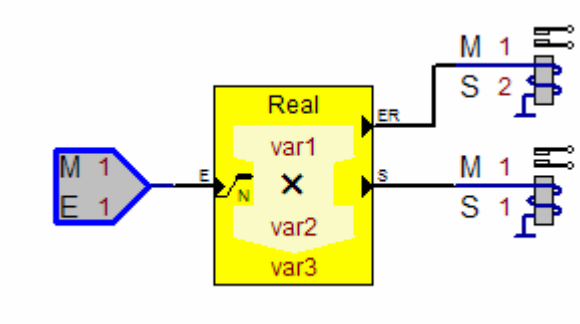
Note que Operando1 e Operando2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow ou underflow) na operação. Note que variáveis inteiras podem assumir valores entre -3,4E38 e 3,4E38. Assim, por exemplo, se multiplicarmos duas variáveis, ambas com valor 2,0E20 o resultado será excessivo (4,0E40) para representação via reais no µDX200 e o nodo de erro será ativado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

> *Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.*



Exemplo de Programa Aplicativo: **Aritmética Real - Multiplicação.dxg**

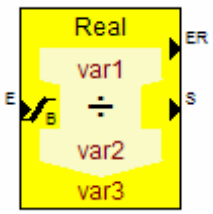
Acima temos um pequeno exemplo de uso do bloco de multiplicação real. No caso, a variável var1 é multiplicada pela variável var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de multiplicação está selecionada por nível). O resultado da operação é armazenado na variável var3. Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de multiplicação real. Note que neste caso não foram colocados monoestáveis nas saídas S e ER do bloco, ao contrário dos

exemplos anteriores. Isso porque a entrada E do bloco de multiplicação está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Multiplicação Var
ARITMÉTICA INTEIRA - Multiplicação
ARITMÉTICA INTEIRA - Multiplicação Var
ARITMÉTICA LONGINT - Multiplicação
ARITMÉTICA LONGINT - Multiplicação Var

ARITMÉTICA REAL - Divisão

Este bloco efetua a divisão de dois operandos reais, colocando o resultado em um terceiro operando real.



Operando1 e Operando2 são os operandos a serem divididos e Resultado recebe o resultado da divisão.

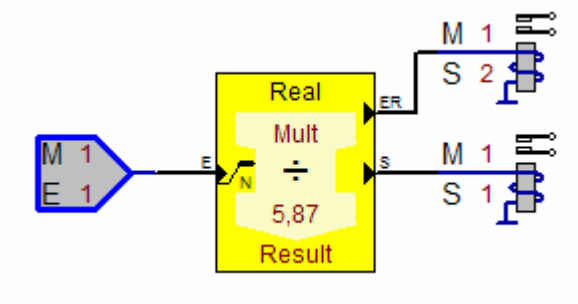
Note que Operando1 e Operando2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. Note que variáveis reais podem assumir valores entre -3,4E38 e 3,4E38. Assim, a divisão sempre resultará em um valor inválido quando o resultado da operação exceder estes limites.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

> *Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.*



Exemplo de Programa Aplicativo: **Aritmética Real - Divisão.dxg**

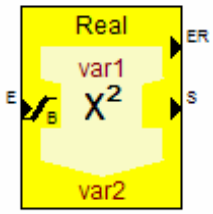
Acima temos um pequeno exemplo de uso do bloco de divisão real. No caso, a variável Mult é dividida pela constante 5,87 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de divisão está selecionada por nível). O resultado da operação é armazenado na variável Result.

Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de divisão real. Note que neste caso não foram colocados monoestáveis nas saídas S e ER do bloco, ao contrário de exemplos anteriores. Isso porque a entrada E do bloco de divisão está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Divisão Var
ARITMÉTICA INTEIRA - Divisão
ARITMÉTICA INTEIRA - Divisão Var
ARITMÉTICA LONGINT - Divisão
ARITMÉTICA LONGINT - Divisão Var

ARITMÉTICA REAL - Quadrado

Este bloco eleva ao quadrado um operando real, colocando o resultado em um segundo operando real.



Operando é o operando a ser elevado ao quadrado e Resultado recebe o resultado da operação.

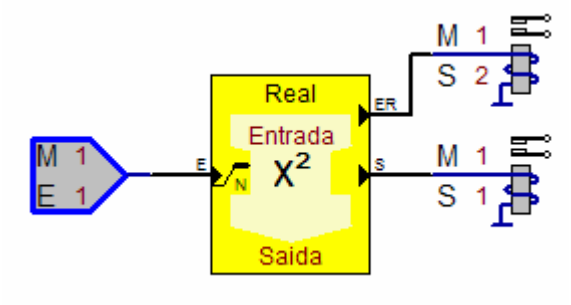
Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow) na operação. Note que variáveis reais podem assumir valores entre -3,4E38 e 3,4E38. Assim, por exemplo, se elevarmos ao quadrado um valor superior a 18,44E18 ou inferior a -18,44E18 o resultado será excessivo para representação via reais no µDX200 e o nodo de erro será ativado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

> *Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.*



Exemplo de Programa Aplicativo: **Aritmética Real - Quadrado.dxg**

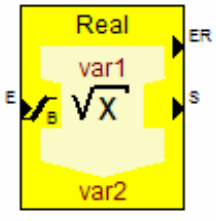
Acima temos um pequeno exemplo de uso do bloco de quadrado real. No caso, a variável Entrada é elevada ao quadrado sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de quadrado está selecionada por nível). O resultado da operação é armazenado na variável Saida (note que foi omitido o acento no nome da palavra; acentuação não é permitida no nome de variáveis no Editor PG).

Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de quadrado real. A entrada E do bloco de quadrado está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Quadrado Var
ARITMÉTICA INTEIRA - Quadrado
ARITMÉTICA INTEIRA - Quadrado Var
ARITMÉTICA LONGINT - Quadrado
ARITMÉTICA LONGINT - Quadrado Var

ARITMÉTICA REAL - Raiz Quadrada

Este bloco calcula a raiz quadrada de um operando real, colocando o resultado em um segundo operando real.



Operando é o operando a ser efetuada a raiz quadrada e Resultado recebe o resultado da

operação.

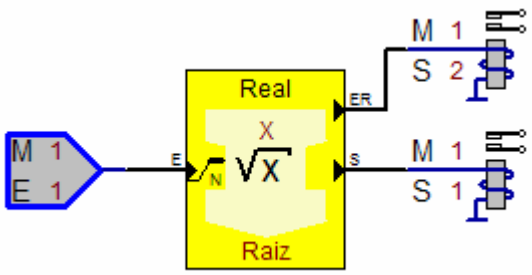
Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. No caso deste bloco o erro só ocorre se for tentado retirar a raiz quadrada de um número negativo.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.



Exemplo de Programa Aplicativo: **Aritmética Real - Raiz Quadrada.dxg**

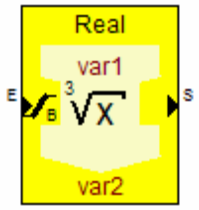
Acima temos um pequeno exemplo de uso do bloco de raiz quadrada real. No caso, é retirada a raiz quadrada da variável X sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de raiz quadrada está selecionada por nível). O resultado da operação é armazenado na variável Raiz.

Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de raiz quadrada real. A entrada E do bloco de raiz quadrada está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Raiz Quadrada Var
ARITMÉTICA INTEIRA - Raiz Quadrada
ARITMÉTICA INTEIRA - Raiz Quadrada Var
ARITMÉTICA LONGINT - Raiz Quadrada
ARITMÉTICA LONGINT - Raiz Quadrada Var

ARITMÉTICA REAL - Raiz Cúbica

Este bloco calcula a raiz cúbica de um operando real, colocando o resultado em um segundo operando real.



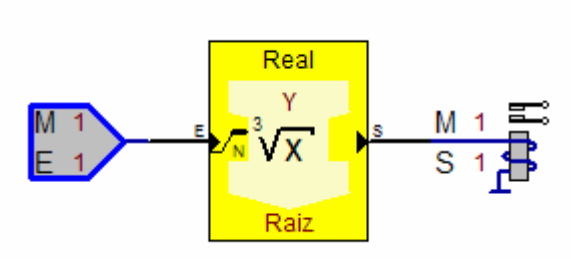
Operando é o operando a ser efetuada a raiz cúbica e Resultado recebe o resultado da operação.

Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Não existe nodo de erro neste bloco, uma vez que a raiz cúbica tanto de números positivos quanto negativos sempre resulta em um valor real.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Real - Raiz Cúbica.dxg**

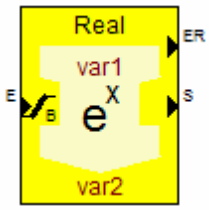
Acima temos um pequeno exemplo de uso do bloco de raiz cúbica real. No caso, é retirada a raiz cúbica da variável Y sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de raiz cúbica está selecionada por nível). O resultado da operação é armazenado na variável Raiz.

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de raiz cúbica real. A entrada E do bloco de raiz cúbica está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Raiz Cúbica Var

ARITMÉTICA REAL - Exponencial Natural

Este bloco calcula a exponencial natural (base = 2,71828182...) de um operando real, colocando o resultado em um segundo operando real.



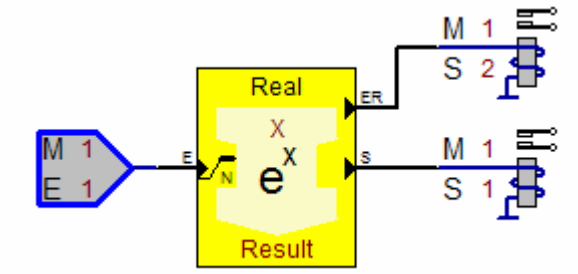
Operando é o operando que serve de expoente e Resultado recebe o resultado da operação.

Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow) na operação. Note que variáveis reais podem assumir valores entre -3,4E38 e 3,4E38. Assim, por exemplo, se elevarmos e=2,7172... a um expoente superior a 88,7 o resultado será excessivo para representação via reais no µDX200 e o nodo de erro será ativado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Real - Exponencial Natural.dxg**

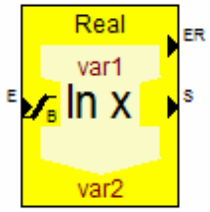
Acima temos um pequeno exemplo de uso do bloco de exponencial natural real. No caso, a base e é elevada ao expoente X sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de exponencial natural está selecionada por nível). O resultado da operação é armazenado na variável Result.

Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de exponencial natural real. A entrada E do bloco de exponencial natural está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Exponencial Natural Var

ARITMÉTICA REAL - Logaritmo Natural

Este bloco calcula o logaritmo natural (base = 2,71828182...) de um operando real, colocando o resultado em um segundo operando real.



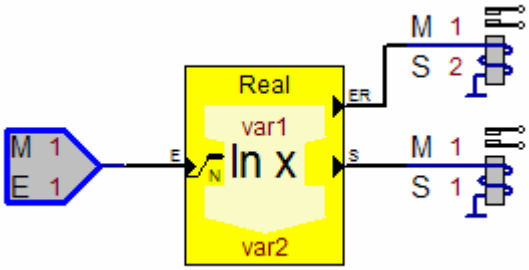
Operando é o operando que serve de dado de entrada e Resultado recebe o resultado da operação. Lembre-se que o ln(x) = y significa que $e^y = x$.

Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. Isso irá ocorrer caso o operando seja menor ou igual a 1 (pois não existe ln(x) se x≤0, pois nenhum número real y faria $e^y = x$).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Real - Logaritmo Natural.dxg**

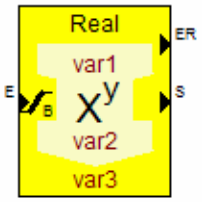
Acima temos um pequeno exemplo de uso do bloco de logaritmo natural real. No caso, é calculado o logaritmo natural da variável var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de logaritmo natural está selecionada por nível). O resultado da operação é armazenado na variável var2.

Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de logaritmo natural real. A entrada E do bloco de logaritmo natural está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Logaritmo Natural Var

ARITMÉTICA REAL - Exponencial

Este bloco calcula um operando real elevado a outro operando real, colocando o resultado em um terceiro operando real.



Operando1 é o operando que serve de base, Operando2 é o operando que serve de expoente e Resultado recebe o resultado da operação.

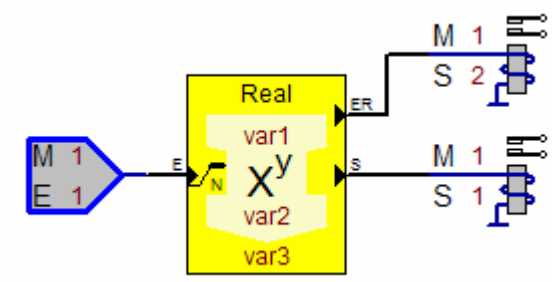
Note que Operando1 e Operando2 podem ser variáveis reais, constantes (valores numéricos), ou ainda referências a posições de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve excesso (overflow) na operação. Note que variáveis reais podem assumir valores entre -3,4E38 e 3,4E38. Assim, por exemplo, se elevarmos 10 a um expoente superior a 38 o resultado será excessivo para representação via reais no µDX200 e o nodo de erro será ativado. Também para bases negativos o bloco sempre retorna erro. Isso porque, neste caso, dependendo do expoente, o resultado pode ser um número real positivo, negativo, ou ainda resultar em um número complexo (e números complexos não são tratados pelo µDX200). Por exemplo:

$(-2)^2 = 4$ $(-2)^3 = -8$ $(-2)^{2,5} = (-2)^{5/2} = 4 \cdot (-2)^{1/2}$

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Real - Exponencial.dxg**

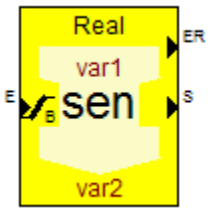
Acima temos um pequeno exemplo de uso do bloco de exponencial real. No caso, a base var1 é elevada ao expoente var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de exponencial natural está selecionada por nível). O resultado da operação é armazenado na variável var3.

Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de exponencial real. A entrada E do bloco de exponencial está selecionada para acionamento por nível e, portanto, as saídas S e ER ficarão ativas durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Exponencial Var

ARITMÉTICA REAL - Seno

Este bloco calcula o seno de um operando real, representando um ângulo em radianos, e coloca o resultado em um segundo operando real.



Operando é o operando a ser efetuada a operação trigonométrica seno e Resultado recebe o resultado da operação.

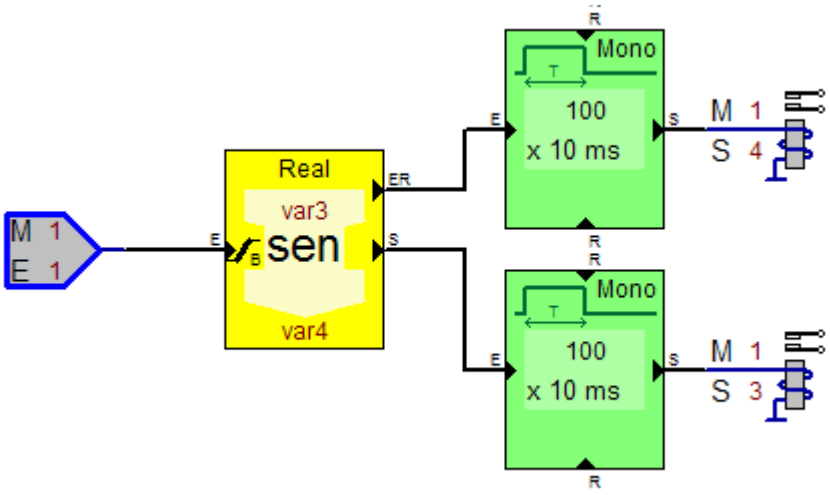
Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. No caso deste bloco o erro só ocorre se for tentado retirar o seno de um ângulo muito grande (maior que 200 radianos).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.



Exemplo de Programa Aplicativo: **Aritmética Real - Seno.dxg**

Acima temos um pequeno exemplo de uso do bloco de seno real. No caso, é retirado o seno da variável var3 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for ligada (já que a entrada E do bloco de seno está selecionada por borda). O resultado da operação é armazenado na variável var4.

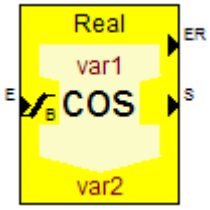
Já as saídas S3 e S4 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de seno real. A entrada E do bloco de seno está selecionada para acionamento por borda e, portanto, as saídas S e ER ficarão ativas durante apenas um ciclo em que a entrada E for energizada. Por isso a inclusão de monoestáveis nas saídas, de forma a permitir a visualização do pulso de saída.

Por exemplo, se fizermos var3 = 2 radianos (114,6 graus) e acionarmos a entrada E1 obteremos var4 = 0,909297, já que sen(2)=0,909297.

Veja também:
ARITMÉTICA REAL - Seno Var

ARITMÉTICA REAL - Cosseno

Este bloco calcula o cosseno de um operando real, representando um ângulo em radianos, e coloca o resultado em um segundo operando real.



Operando é o operando a ser efetuada a operação trigonométrica cosseno e Resultado recebe o resultado da operação.

Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

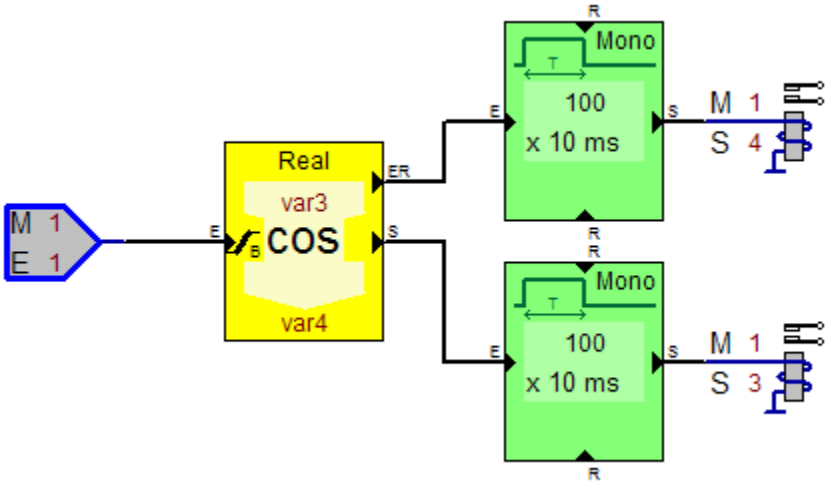
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e

pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. No caso deste bloco o erro só ocorre se for tentado retirar o cosseno de um ângulo muito grande (maior que 200 radianos).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.



Exemplo de Programa Aplicativo: **Aritmética Real - Cosseno.dxg**

Acima temos um pequeno exemplo de uso do bloco de cosseno real. No caso, é retirado o cosseno da variável var3 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for ligada (já que a entrada E do bloco de cosseno está selecionada por borda). O resultado da operação é armazenado na variável var4.

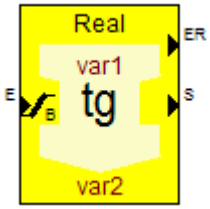
Já as saídas S3 e S4 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de cosseno real. A entrada E do bloco de cosseno está selecionada para acionamento por borda e, portanto, as saídas S e ER ficarão ativas durante apenas um ciclo em que a entrada E for energizada. Por isso a inclusão de monoestáveis nas saídas, de forma a permitir a visualização do pulso de saída.

Por exemplo, se fizermos var3 = 2 radianos (114,6 graus) e acionarmos a entrada E1 obteremos var4 = -0,416147, já que cos(2)=-0,416147.

Veja também:
ARITMÉTICA REAL - Cosseno Var

ARITMÉTICA REAL - Tangente

Este bloco calcula a tangente de um operando real, representando um ângulo em radianos, e coloca o resultado em um segundo operando real. Esta função é definida como tg(x) = sen(x) / cos(x).



Operando é o operando a ser efetuada a operação trigonométrica tangente e Resultado recebe o resultado da operação.

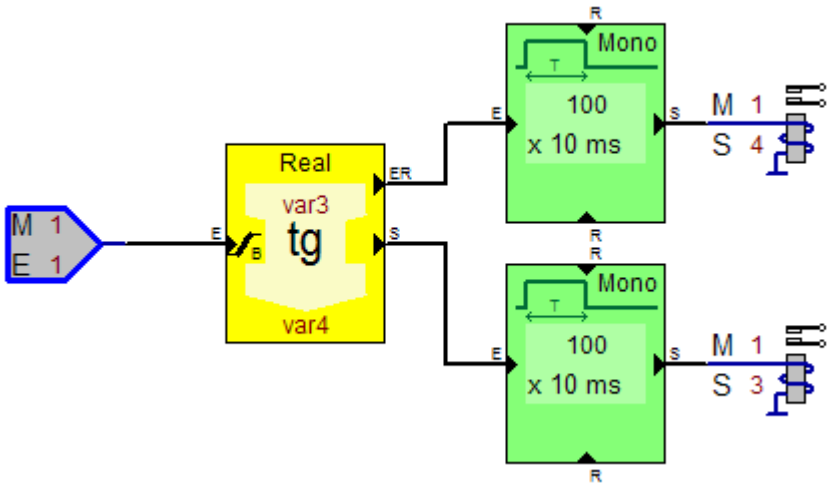
Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. No caso deste bloco o erro só ocorre se for tentado retirar a tangente de um ângulo muito grande (maior que 200 radianos). Embora tg(π /2+nπ) também devesse retornar erro (uma vez que a tangente tende ao infinito), limitações na precisão do cálculo restringem a faixa do resultado entre aproximadamente -1E9 a 1E9.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.



Exemplo de Programa Aplicativo: **Aritmética Real - Tangente.dxg**

Acima temos um pequeno exemplo de uso do bloco de tangente real. No caso, é retirada a tangente da variável var3 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for ligada (já que a entrada E do bloco de tangente está selecionada por borda). O resultado da operação é armazenado na variável var4.

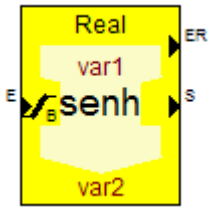
Já as saídas S3 e S4 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de tangente real. A entrada E do bloco de tangente está selecionada para acionamento por borda e, portanto, as saídas S e ER ficarão ativas durante apenas um ciclo em que a entrada E for energizada. Por isso a inclusão de monoestáveis nas saídas, de forma a permitir a visualização do pulso de saída.

Por exemplo, se fizermos var3 = 2 radianos (114,6 graus) e acionarmos a entrada E1 obteremos var4 = -2,18504, já que tg(2)=-2,18504.

Veja também:
ARITMÉTICA REAL - Tangente Var

ARITMÉTICA REAL - Seno Hiperbólico

Este bloco calcula o seno hiperbólico de um operando real, e coloca o resultado em um segundo operando real. Esta função é definida como senh(x) = $(e^{x} - e^{-x}) / 2$.



Operando é o operando a ser efetuada a operação seno hiperbólico e Resultado recebe o resultado da operação.

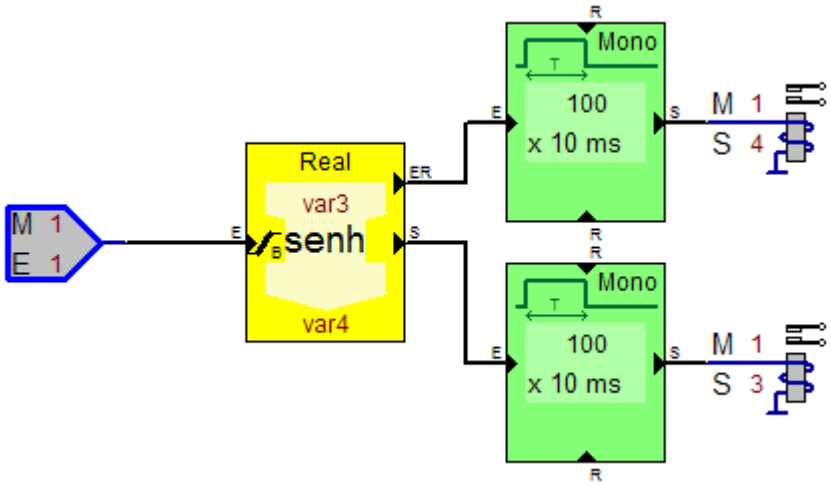
Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. No caso deste bloco o erro só ocorre se for tentado retirar o seno hiperbólico de um operando muito elevado (maior que 200 ou menor que -200).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.



Exemplo de Programa Aplicativo: **Aritmética Real - Seno Hiperbólico.dxg**

Acima temos um pequeno exemplo de uso do bloco de seno hiperbólico real. No caso, é retirado o seno hiperbólico da variável var3 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for ligada (já que a entrada E do bloco de seno hiperbólico está selecionada por borda). O resultado da operação é armazenado na variável var4.

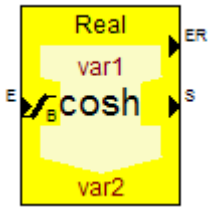
Já as saídas S3 e S4 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de seno hiperbólico real. A entrada E do bloco de seno hiperbólico está selecionada para acionamento por borda e, portanto, as saídas S e ER ficarão ativas durante apenas um ciclo em que a entrada E for energizada. Por isso a inclusão de monoestáveis nas saídas, de forma a permitir a visualização do pulso de saída.

Por exemplo, se fizermos var3 = 2 e acionarmos a entrada E1 obteremos var4 = 3,62686, já que senh(2)=3,62686. Note que se calcularmos senh(2) = $(e^2 - e^{-2}) / 2$ também resultará em 3,62686.

Veja também:
ARITMÉTICA REAL - Seno Hiperbólico Var

ARITMÉTICA REAL - Cosseno Hiperbólico

Este bloco calcula o cosseno hiperbólico de um operando real, e coloca o resultado em um segundo operando real. Esta função é definida como cosh(x) = $(e^x + e^{-x}) / 2$.



Operando é o operando a ser efetuada a operação cosseno hiperbólico e Resultado recebe o resultado da operação.

Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

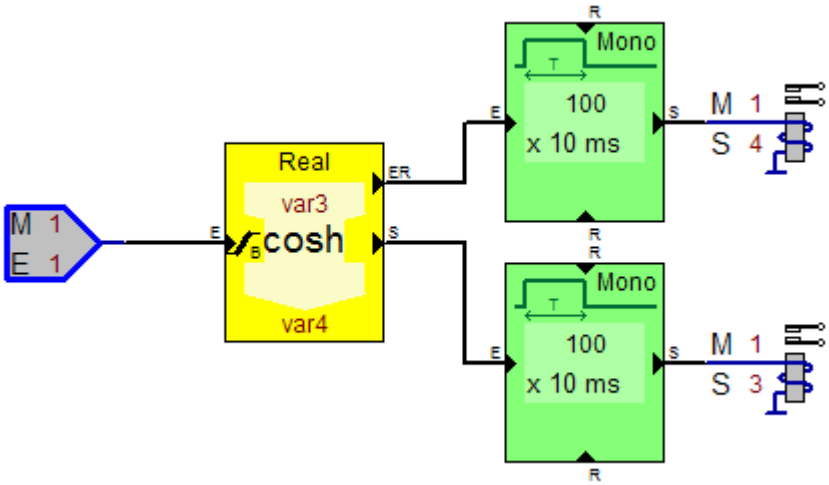
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e

pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. No caso deste bloco o erro só ocorre se for tentado retirar o cosseno hiperbólico de um operando muito elevado (maior que 200 ou menor que -200).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.



Exemplo de Programa Aplicativo: **Aritmética Real - Cosseno Hiperbólico.dxg**

Acima temos um pequeno exemplo de uso do bloco de cosseno hiperbólico real. No caso, é retirado o cosseno hiperbólico da variável var3 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for ligada (já que a entrada E do bloco de cosseno hiperbólico está selecionada por borda). O resultado da operação é armazenado na variável var4.

Já as saídas S3 e S4 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de cosseno hiperbólico real. A entrada E do bloco de cosseno hiperbólico está selecionada para acionamento por borda e, portanto, as saídas S e ER ficarão ativas durante apenas um ciclo em que a entrada E for energizada. Por isso a inclusão de monoestáveis nas saídas, de forma a permitir a visualização do pulso de saída.

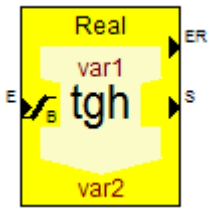
Por exemplo, se fizermos var3 = 2 e acionarmos a entrada E1 obteremos var4 = 3,7622, já que cosh(2)=3,7622. Note que se calcularmos cosh(2) = $(e^2 + e^{-2}) / 2$ também resultará em 3,7622.

Nota: Ao observar um fio preso em dois postes notamos que seu peso faz com que ele assuma uma forma arredondada, dando a impressão de descrever uma parábola. Na verdade, tal curva é o gráfico da função cosseno hiperbólico, conhecida como catenária.

Veja também:
ARITMÉTICA REAL - Cosseno Hiperbólico Var

ARITMÉTICA REAL - Tangente Hiperbólica

Este bloco calcula a tangente hiperbólica de um operando real, e coloca o resultado em um segundo operando real. Esta função é definida como tgh(x) = senh(x) / cosh(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x}).



Operando é o operando a ser efetuada a operação tangente hiperbólica e Resultado recebe o resultado da operação.

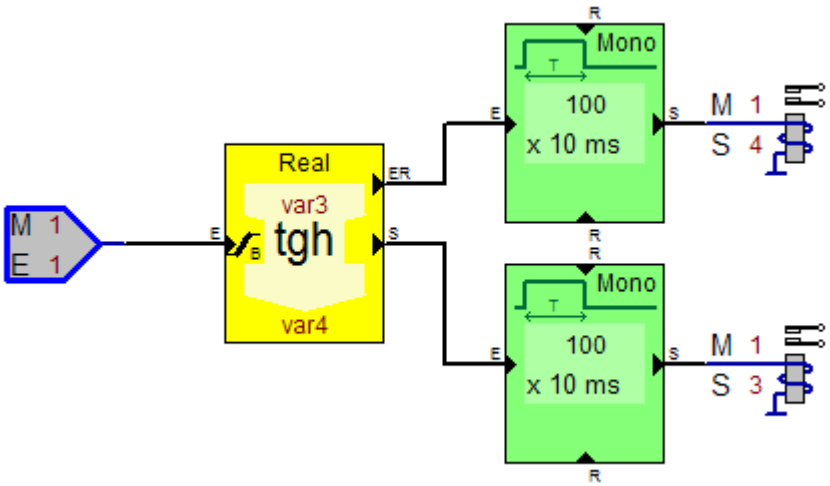
Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que houve erro na operação. No caso deste bloco o erro só ocorre se for tentado retirar a tangente hiperbólica de um operando muito elevado (maior que 200 ou menor que -200).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir centenas de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Observação: as operações com números reais são muito mais lentas do que as com números inteiros. Assim, sempre que possível deve-se usar operações inteiras para acelerar o programa aplicativo.



Exemplo de Programa Aplicativo: **Aritmética Real - Tangente Hiperbólica.dxg**

Acima temos um pequeno exemplo de uso do bloco de tangente hiperbólica real. No caso, é

retirado a tangente hiperbólica da variável var3 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) for ligada (já que a entrada E do bloco de tangente hiperbólica está selecionada por borda). O resultado da operação é armazenado na variável var4.

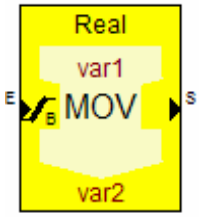
Já as saídas S3 e S4 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de tangente hiperbólica real. A entrada E do bloco de tangente hiperbólica está selecionada para acionamento por borda e, portanto, as saídas S e ER ficarão ativas durante apenas um ciclo em que a entrada E for energizada. Por isso a inclusão de monoestáveis nas saídas, de forma a permitir a visualização do pulso de saída.

Por exemplo, se fizermos var3 = 2 e acionarmos a entrada E1 obteremos var4 = 0,964028, já que tgh(2)=0,964028. Note que se calcularmos $tgh(2) = (e^{2} - e^{-2}) / (e^{2} + e^{-2})$ também resultará em 0,964028.

Veja também:
ARITMÉTICA REAL - Tangente Hiperbólica Var

ARITMÉTICA REAL - Atribuição

Este bloco transfere o valor de um operando real para um segundo operando real.



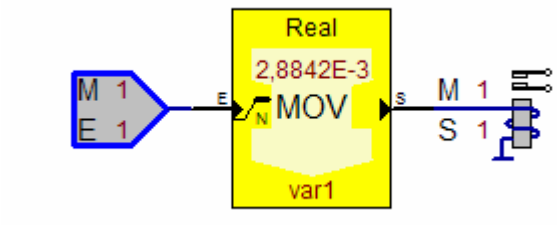
Operando é o operando cujo valor será transferido para Resultado.

Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Real - Atribuição.dxg**

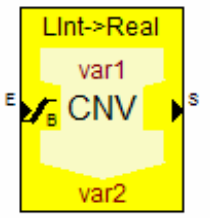
Acima temos um pequeno exemplo de uso do bloco de atribuição real. No caso, é atribuído o valor 2,8842E-3 (0,0028842) na variável real var1 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível).

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição real. A entrada E do bloco de atribuição está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Atribuição Var
ARITMÉTICA INTEIRA - Atribuição
ARITMÉTICA INTEIRA - Atribuição Var
ARITMÉTICA LONGINT - Atribuição
ARITMÉTICA LONGINT - Atribuição Var
ARITMÉTICA WORD - Atribuição
ARITMÉTICA WORD - Atribuição Var

ARITMÉTICA REAL - Conversão LongInt -> Real

Este bloco transfere o valor de um operando longint para um segundo operando real. Ou seja, converte um valor longint em um valor real.



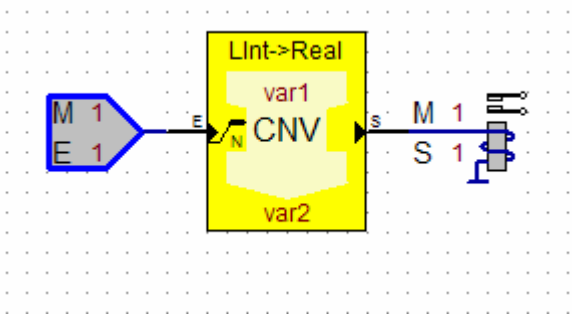
Operando é o operando cujo valor será transferido para Resultado.

Note que Operando pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso (uma vez que o conjunto de números reais abrange os inteiros).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Real - Conversão LongInt - Real.dxg**

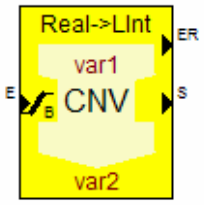
O exemplo acima transfere o valor da variável longint var1 para a variável real var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível).

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de conversão longint à real. A entrada E do bloco de atribuição está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver energizada.

Veja também:
ARITMÉTICA REAL - Conversão Real -> LongInt
ARITMÉTICA REAL - Conversão LongInt -> Real Var
ARITMÉTICA REAL - Conversão Real -> LongInt Var

ARITMÉTICA REAL - Conversão Real -> LongInt

Este bloco transfere o valor de um operando real para um segundo operando longint. Ou seja, converte um valor real em um valor longint.



Operando é o operando cujo valor será transferido para Resultado.

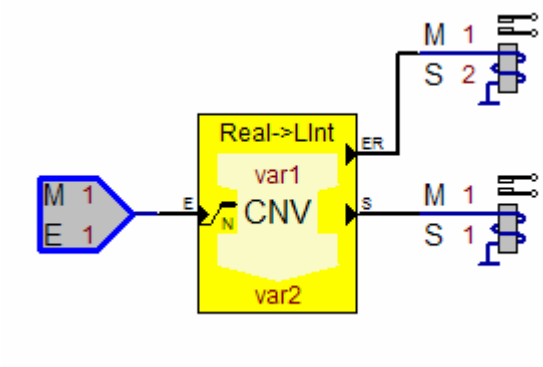
Note que Operando pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco possui nodo de Erro (ER), pois se Operando assumir valores maiores que os limites representáveis pelos números longint (que representam valores de -2.147.483.648 a 2.147.483.647) a conversão não será possível.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco

só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Real - Conversão Real - LongInt.dxg**

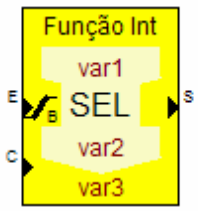
O exemplo acima transfere o valor da variável real var1 para a variável longint var2 sempre que a entrada E1 do módulo de Expansão M1 (µDX210) estiver ligada (já que a entrada E do bloco de atribuição está selecionada por nível).

Já as saídas S1 e S2 do mesmo módulo de Entradas/Saídas (µDX210) são acionadas conforme os nodos Saída (S) e Erro (ER) do bloco de conversão real à LongInt. A entrada E do bloco está selecionada para acionamento por nível e, portanto, a saída S ficará ativa durante todo o tempo em que a entrada E estiver ligada.

Veja também:
ARITMÉTICA REAL - Conversão LongInt -> Real
ARITMÉTICA REAL - Conversão LongInt -> Real Var
ARITMÉTICA REAL - Conversão Real -> LongInt Var

ARITMÉTICA REAL - Seletor

Este bloco transfere o valor de um ou outro operando real para um terceiro operando real.



Operando1 e Operando2 são os operandos que serão transferidos para Resultado, conforme o estado do nodo de Controle (C).

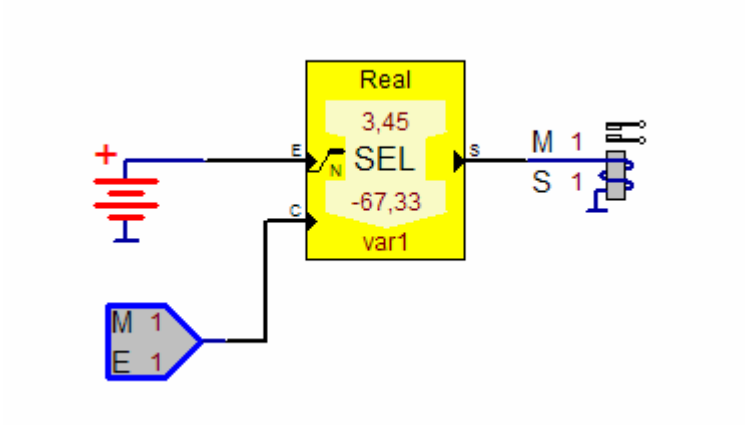
Note que Operando1 e Operando2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Este bloco não possui nodo de Erro (ER), pois a operação sempre será completada com sucesso.

O nodo de Controle (C) especifica qual dos operandos será transferido para Resultado. Se C=0 o Operando1 é transferido para Resultado. Já se C=1 o Operando2 é transferido para Resultado.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Aritmética Real - Seletor.dxg**

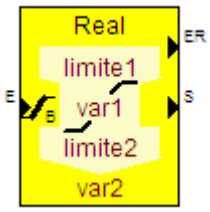
Acima temos um pequeno exemplo de uso do bloco de seleção real. No caso, é atribuído o valor de 3,45 ou -67,33 na variável real var1, conforme o estado do nodo de Controle (C). O nodo de Entrada (E) está sempre ativado, pois está selecionado para acionamento por nível e está ligado constantemente pelo bloco de energia. O nodo de Controle (C) é comando pela entrada E1 do módulo 1 de Expansão de Entradas/Saídas (µDX210).

Já a saída S1 do mesmo módulo de Entradas/Saídas (µDX210) é acionada conforme o nodo Saída (S) do bloco de atribuição inteira. A entrada E do bloco de atribuição está selecionada para acionamento por nível e sempre ligada. Então, a saída S ficará ativa sempre.

Veja também:
ARITMÉTICA INTEIRA - Seletor
ARITMÉTICA LONGINT - Seletor
ARITMÉTICA WORD - Seletor
GERAL - Seletor de Variável Inteira
GERAL - Seletor de Variável LongInt
GERAL - Seletor de Variável Word
GERAL - Seletor de Variável Real

ARITMÉTICA REAL - Operação Limite

Permite limitar uma variável real entre dois valores especificados. Assim, se garante que a variável assumirá apenas valores entre os dois limites. Este bloco somente é operacional em controladores µDX201 versão 3.37 ou superior.



Operando é a variável cujo valor será limitado entre os limites estabelecidos por Lim.Superior e Lim.Inferior e transferido para Resultado.

Note que Operando, Lim.Superior e Lim.Inferior podem ser variáveis reais, constantes (valor numérico em decimal ou hexadecimal), ou ainda referências a uma posição de memória. Já Resultado deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Variáveis reais podem assumir valores entre -3,4E38 e 3,4E38 (note a convenção para especificação do expoente: 1E5=100000, 1E-3=0,001).

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Erro (ER) indica que Operando possui valor fora dos limites impostos por Lim.Superior e Lim.Inferior.

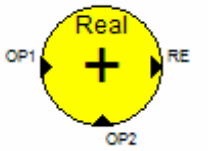
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Limite
ARITMÉTICA INTEIRA - Operação Limite Var
ARITMÉTICA LONGINT - Operação Limite
ARITMÉTICA LONGINT - Operação Limite Var
ARITMÉTICA WORD - Operação Limite
ARITMÉTICA WORD - Operação Limite Var
ARITMÉTICA REAL - Operação Limite Var

ARITMÉTICA REAL - Adição Var

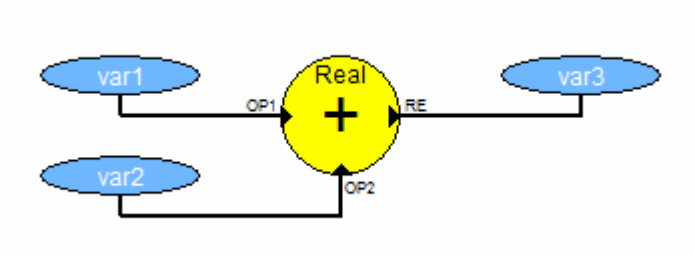
Este bloco efetua a adição de dois operandos reais, colocando o resultado em um terceiro operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos reais a serem adicionados e RE recebe o resultado da adição.

Note que OP1 e OP2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

As variáveis reais podem assumir valores entre -3,4E38 e 3,4E38.



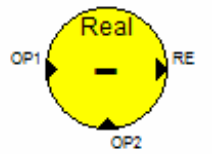
Exemplo de Programa Aplicativo: **Aritmética Real - Adição Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de adição var real. No caso, a variável var1 é somada a variável var2 constantemente, e o resultado da operação é colocado em var3. Note que as variáveis são transportadas para o bloco de adição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis reais (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Adição
ARITMÉTICA INTEIRA - Adição
ARITMÉTICA INTEIRA - Adição Var
ARITMÉTICA LONGINT - Adição
ARITMÉTICA LONGINT - Adição Var
ARITMÉTICA WORD - Adição
ARITMÉTICA WORD - Adição Var

ARITMÉTICA REAL - Subtração Var

Este bloco efetua a subtração de dois operandos reais, colocando o resultado em um terceiro operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

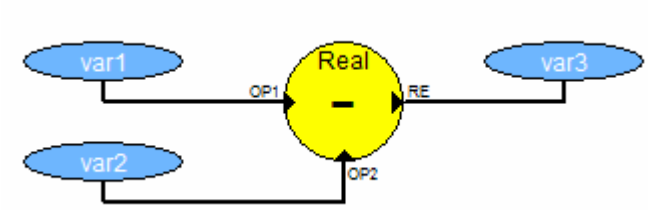


OP1 e OP2 são os operandos reais a serem subtraídos e RE recebe o resultado da subtração.

Note que OP1 e OP2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma

referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



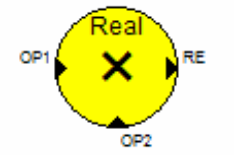
Exemplo de Programa Aplicativo: **Aritmética Real - Subtração Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de subtração var real. No caso, a variável var2 é subtraída da variável var1 constantemente, e o resultado da operação é colocado em var3. Note que as variáveis são transportadas para o bloco de subtração pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis reais (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Subtração
ARITMÉTICA INTEIRA - Subtração
ARITMÉTICA INTEIRA - Subtração Var
ARITMÉTICA LONGINT - Subtração
ARITMÉTICA LONGINT - Subtração Var
ARITMÉTICA WORD - Subtração
ARITMÉTICA WORD - Subtração Var

ARITMÉTICA REAL - Multiplicação Var

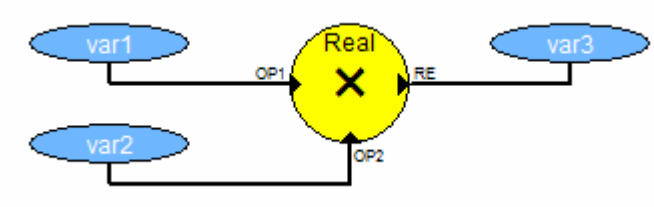
Este bloco efetua a multiplicação de dois operandos reais, colocando o resultado em um terceiro operando inteiro. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos reais a serem multiplicados e RE recebe o resultado da multiplicação.

Note que OP1 e OP2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



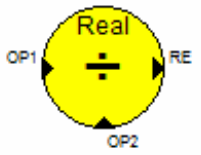
Exemplo de Programa Aplicativo: **Aritmética Real - Multiplicação Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de multiplicação var real. No caso, a variável var1 é multiplicada com a variável var2 constantemente, e o resultado da operação é colocado em var3. Note que as variáveis são transportadas para o bloco de multiplicação pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis reais (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Multiplicação
ARITMÉTICA INTEIRA - Multiplicação
ARITMÉTICA INTEIRA - Multiplicação Var
ARITMÉTICA LONGINT - Multiplicação
ARITMÉTICA LONGINT - Multiplicação Var

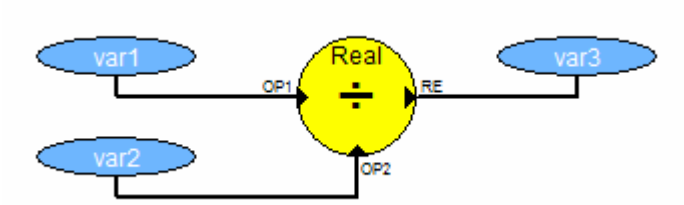
ARITMÉTICA REAL - Divisão Var

Este bloco efetua a divisão de dois operandos reais, colocando o resultado em um terceiro operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 e OP2 são os operandos reais a serem divididos e RE recebe o resultado da divisão.

Note que OP1 e OP2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



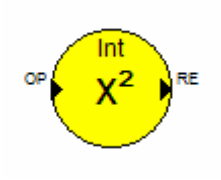
Exemplo de Programa Aplicativo: **Aritmética Real - Divisão Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de divisão var real. No caso, a variável var1 é dividida pela variável var2 constantemente, e o resultado da operação é colocado em var3. Note que as variáveis são transportadas para o bloco de divisão pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis reais (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Divisão
ARITMÉTICA INTEIRA - Divisão
ARITMÉTICA INTEIRA - Divisão Var
ARITMÉTICA LONGINT - Divisão
ARITMÉTICA LONGINT - Divisão Var

ARITMÉTICA REAL - Quadrado Var

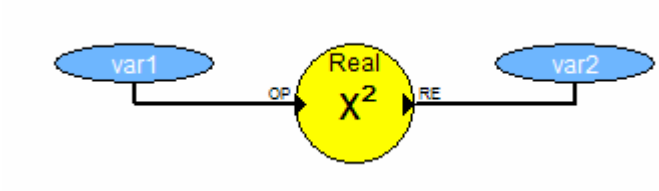
Este bloco eleva ao quadrado um operando real, colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP é o operando real a ser elevado ao quadrado e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



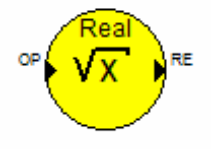
Exemplo de Programa Aplicativo: **Aritmética Real - Quadrado Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de quadrado var real. No caso, a variável var1 é elevada ao quadrado constantemente, e o resultado da operação é colocado em var2. Note que as variáveis são transportadas para o bloco de quadrado pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis reais (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Quadrado
ARITMÉTICA INTEIRA - Quadrado
ARITMÉTICA INTEIRA - Quadrado Var
ARITMÉTICA LONGINT - Quadrado
ARITMÉTICA LONGINT - Quadrado Var

ARITMÉTICA REAL - Raiz Quadrada Var

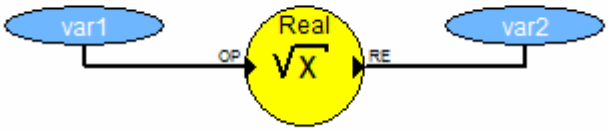
Este bloco calcula a raiz quadrada de um operando real, colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP é o operando real do qual será extraída a raiz quadrada e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



Exemplo de Programa Aplicativo: **Aritmética Real - Raiz Quadrada Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de raiz quadrada var real. No caso, é retirada a raiz quadrada da variável var1 constantemente, e o resultado da operação é colocado em var2. Note que as variáveis são transportadas para o bloco de raiz quadrada pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Raiz Quadrada
ARITMÉTICA INTEIRA - Raiz Quadrada
ARITMÉTICA INTEIRA - Raiz Quadrada Var
ARITMÉTICA LONGINT - Raiz Quadrada
ARITMÉTICA LONGINT - Raiz Quadrada Var

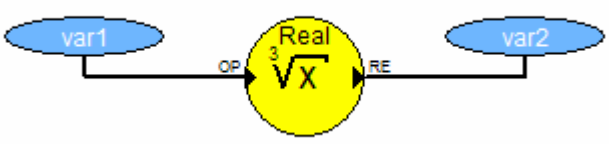
ARITMÉTICA REAL - Raiz Cúbica Var

Este bloco calcula a raiz cúbica de um operando real, colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

OP é o operando real do qual será extraída a raiz cúbica e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



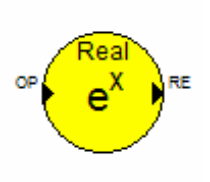
Exemplo de Programa Aplicativo: **Aritmética Real - Raiz Cúbica Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de raiz cúbica var real. No caso, é retirada a raiz cúbica da variável var1 constantemente, e o resultado da operação é colocado em var2. Note que as variáveis são transportadas para o bloco de raiz cúbica pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Raiz Cúbica

ARITMÉTICA REAL - Exponencial Natural Var

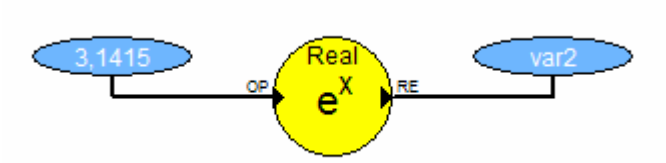
Este bloco eleva a constante e (2,718281...) a um expoente real, colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP é o operando real que servirá de expoente para a operação e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



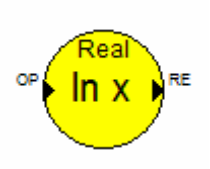
Exemplo de Programa Aplicativo: **Aritmética Real - Exponencial Natural Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de exponencial natural var real. No caso, a base 2,718281...(e) é elevada a 3,1415 (aproximadamente π) constantemente, e o resultado da operação é colocado em var2 (resultando em aproximadamente 23,13855). Note que as variáveis são transportadas para o bloco de exponencial natural pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis reais (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Exponencial Natural

ARITMÉTICA REAL - Logaritmo Natural Var

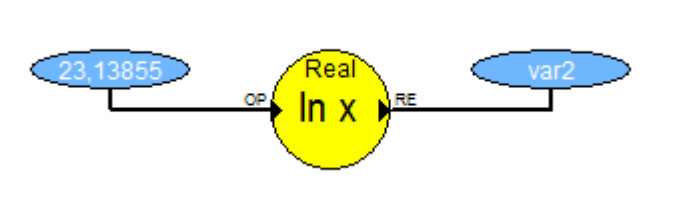
Este bloco calcula o logaritmo neperiano ou natural de um operando real, colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP é o operando real a ser calculado o logaritmo natural e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



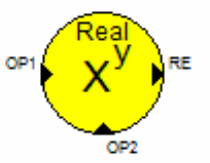
Exemplo de Programa Aplicativo: **Aritmética Real - Logaritmo Natural Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de logaritmo natural var real. No caso, é retirado o logaritmo natural de 23,13855 constantemente, e o resultado da operação é colocado em var2 (resulta em aproximadamente 3,1415 - ou seja, aproximadamente π). Note que as variáveis são transportadas para o bloco de logaritmo natural pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis reais (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Logaritmo Natural

ARITMÉTICA REAL - Exponencial Var

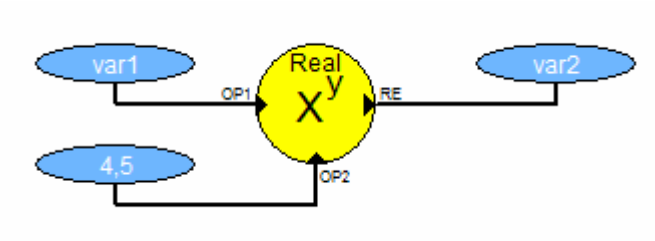
Este bloco eleva a base especifica por um operando real no expoente especificado por um segundo operando real, colocando o resultado em um terceiro operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP1 é o operando real que especifica a base da exponenciação, OP1 especifica o expoente e RE recebe o resultado da operação.

Note que OP1 e OP2 podem ser variáveis reais, constantes (valores numéricos), ou ainda referências a posições de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



Exemplo de Programa Aplicativo: **Aritmética Real - Exponencial Var.dxg**

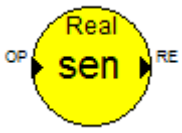
Acima temos um pequeno exemplo de uso do bloco de exponencial var real. No caso, a variável var1 é elevada a 4,5 constantemente, e o resultado da operação é colocado em var2. Note que as variáveis são transportadas para o bloco de quadrado pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis reais (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja que: $(var1)^{4,5} = (var1)^{9/2} = (var1)^{4}.(var1)^{1/2}$

Veja também:
ARITMÉTICA REAL - Exponencial

ARITMÉTICA REAL - Seno Var

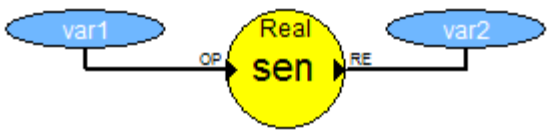
Este bloco calcula a função trigonométrica seno de um operando real (representando um ângulo em radianos), colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP é o operando real no qual será efetuada a operação seno e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



Exemplo de Programa Aplicativo: **Aritmética Real - Seno Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de seno var real. No caso, é retirada o seno da variável var1 constantemente, e o resultado da operação é colocado em var2. Note que as variáveis são transportadas para o bloco de seno pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Seno

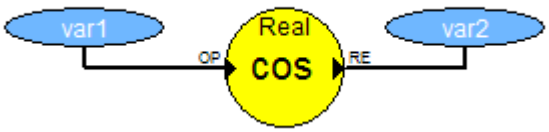
ARITMÉTICA REAL - Cosseno Var

Este bloco calcula a função trigonométrica cosseno de um operando real (representando um ângulo em radianos), colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

OP é o operando real no qual será efetuada a operação cosseno e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



Exemplo de Programa Aplicativo: **Aritmética Real - Cosseno Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de cosseno var real. No caso, é retirada o cosseno da variável var1 constantemente, e o resultado da operação é colocado em var2. Note que as variáveis são transportadas para o bloco de cosseno pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Cosseno

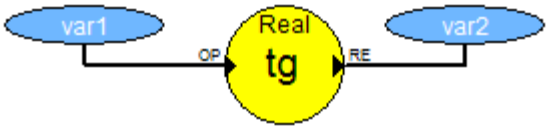
ARITMÉTICA REAL - Tangente Var

Este bloco calcula a função trigonométrica tangente de um operando real (representando um ângulo em radianos), colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).

OP é o operando real no qual será efetuada a operação tangente e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



Exemplo de Programa Aplicativo: **Aritmética Real - Tangente Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de tangente var real. No caso, é retirada a tangente da variável var1 constantemente, e o resultado da operação é colocado em var2. Note que as variáveis são transportadas para o bloco de tangente pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Tangente

ARITMÉTICA REAL - Seno Hiperbólico Var

Este bloco calcula a função seno hiperbólico de um operando real, colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). A função seno hiperbólico é definida como senh(x) = $(e^x - e^{-x}) / 2$.

OP é o operando real no qual será efetuada a operação seno hiperbólico e RE recebe o resultado

da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.

Exemplo de Programa Aplicativo: **Aritmética Real - Seno Hiperbólico Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de seno hiperbólico var. No caso, é retirada o seno hiperbólico da variável var1 constantemente, e o resultado da operação é colocado em var2. Note que as variáveis são transportadas para o bloco de seno hiperbólico pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Seno Hiperbólico

ARITMÉTICA REAL - Cosseno Hiperbólico Var

Este bloco calcula a função cosseno hiperbólico de um operando real, colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). A função cosseno hiperbólico é definida como $\cosh(x) = (e^x + e^{-x}) / 2$.

OP é o operando real no qual será efetuada a operação cosseno hiperbólico e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.

Exemplo de Programa Aplicativo: **Aritmética Real - Cosseno Hiperbólico Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de cosseno hiperbólico var. No caso, é retirada o cosseno hiperbólico da variável var1 constantemente, e o resultado da operação é

colocado em var2. Note que as variáveis são transportadas para o bloco de cosseno hiperbólico pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Cosseno Hiperbólico

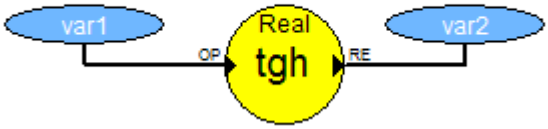
ARITMÉTICA REAL - Tangente Hiperbólica Var

Este bloco calcula a função tangente hiperbólica de um operando real, colocando o resultado em um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). A função tangente hiperbólica é definida como tgh(x) = senh(x) / cosh(x) = $(e^x - e^{-x}) / (e^x + e^{-x})$.

OP é o operando real no qual será efetuada a operação tangente hiperbólica e RE recebe o resultado da operação.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

Variáveis reais no µDX200 podem assumir valores entre -3,4E38 e 3,4E38.



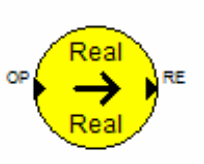
Exemplo de Programa Aplicativo: **Aritmética Real - Tangente Hiperbólica Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de tangente hiperbólica var. No caso, é retirada o tangente hiperbólica da variável var1 constantemente, e o resultado da operação é colocado em var2. Note que as variáveis são transportadas para o bloco de tangente hiperbólica pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis inteiras (-3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Tangente Hiperbólica

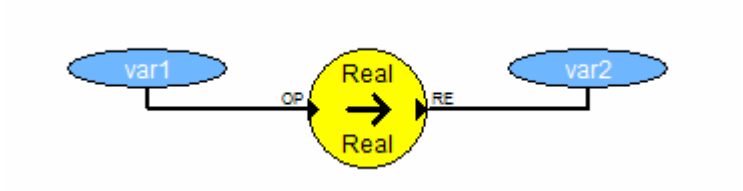
ARITMÉTICA REAL - Atribuição Var

Este bloco transfere o valor de um operando real para um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP é o operando real a ser transferido para RE.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



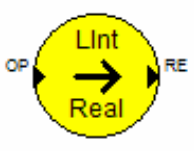
Exemplo de Programa Aplicativo: **Aritmética Real - Atribuição Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de atribuição var real. No caso, é transferido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis reais (de -3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Atribuição
ARITMÉTICA INTEIRA - Atribuição
ARITMÉTICA INTEIRA - Atribuição Var
ARITMÉTICA LONGINT - Atribuição
ARITMÉTICA LONGINT - Atribuição Var
ARITMÉTICA WORD - Atribuição
ARITMÉTICA WORD - Atribuição Var

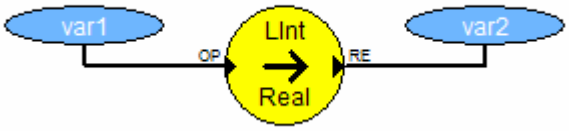
ARITMÉTICA REAL - Conversão LongInt -> Real Var

Este bloco converte o valor de um operando longint para um segundo operando real. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP é o operando longint a ser convertido para o operando real RE.

Note que OP pode ser variável longint, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



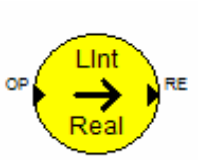
Exemplo de Programa Aplicativo: **Aritmética Real - Conversão LongInt - Real Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de conversão longint à real var. No caso, é convertido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis longint (de -2.147.483.648 a 2.147.483.647) e variáveis reais (de -3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Conversão Real -> LongInt Var
ARITMÉTICA REAL - Conversão Real -> LongInt
ARITMÉTICA REAL - Conversão LongInt -> Real

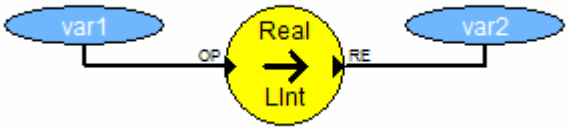
ARITMÉTICA REAL - Conversão Real -> LongInt Var

Este bloco converte o valor de um operando real para um segundo operando longint. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP é o operando real a ser convertido para o operando longint RE.

Note que OP pode ser variável real, constante (valor numérico), ou ainda uma referência a uma posição de memória. Já RE deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.



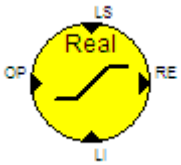
Exemplo de Programa Aplicativo: **Aritmética Real - Conversão Real - LongInt Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de conversão real à longint var. No caso, é convertido o valor de var1 para var2 constantemente. Note que as variáveis são transportadas para o bloco de atribuição pelos "fios" de conexão. Ou seja, neste caso estas conexões transportam variáveis longint (de -2.147.483.648 a 2.147.483.647) e variáveis reais (de -3,4E38 a 3,4E38) e não binárias (nodos), com apenas dois estados (ligado ou desligado).

Veja também:
ARITMÉTICA REAL - Conversão LongInt -> Real Var
ARITMÉTICA REAL - Conversão Real -> LongInt
ARITMÉTICA REAL - Conversão LongInt -> Real

ARITMÉTICA REAL - Operação Limite Var

Permite limitar uma variável real entre dois valores especificados. Assim, se garante que a variável assumirá apenas valores entre os dois limites. Este bloco somente é operacional em controladores µDX201 versão 3.37 ou superior. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares).



OP (operando) é a variável cujo valor será limitado entre os limites estabelecidos por LS (limite superior) e LI (limite inferior) e transferido para RE (resultado).

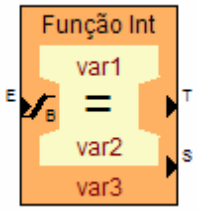
Note que OP, LS e LI podem ser variáveis reais, constantes (valor numérico em decimal ou hexadecimal), ou ainda referências a uma posição de memória. Já RE deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação. Variáveis reais podem assumir valores entre -3,4E38 e 3,4E38 (note a convenção para especificação do expoente: 1E5=100000, 1E-3=0,001).

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
ARITMÉTICA INTEIRA - Operação Limite
ARITMÉTICA INTEIRA - Operação Limite Var
ARITMÉTICA LONGINT - Operação Limite
ARITMÉTICA LONGINT - Operação Limite Var
ARITMÉTICA WORD - Operação Limite
ARITMÉTICA WORD - Operação Limite Var
ARITMÉTICA REAL - Operação Limite

COMPARAÇÃO - Igual Inteiro

Este bloco efetua a comparação entre dois operandos inteiros, ligando o nodo de Teste (T) caso os dois operandos sejam iguais. Note que um terceiro operando inteiro permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais.



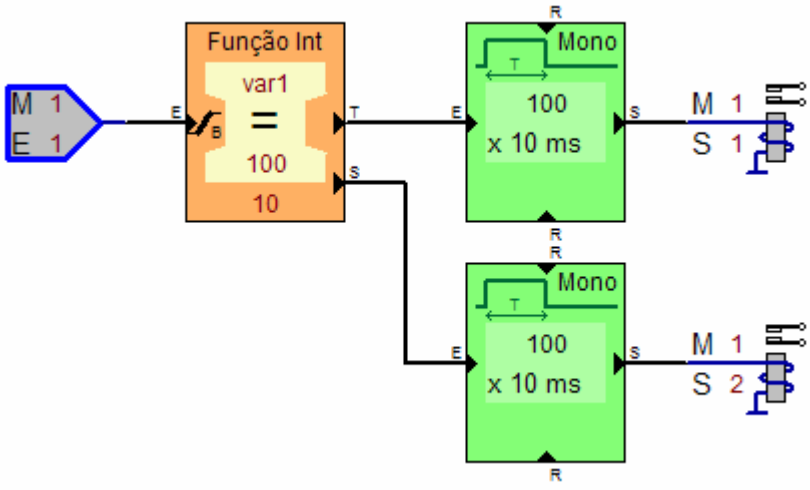
Operando1 e Operando2 são os operandos a serem comparados e Faixa é o valor dentro do qual os dois operandos são considerados iguais. Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, os dois operandos devem ter valor idêntico para haver o acionamento do nodo de Teste (T).

Note que Operando1, Operando2 e Faixa podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 = Operando2 dentro da Faixa especificada.

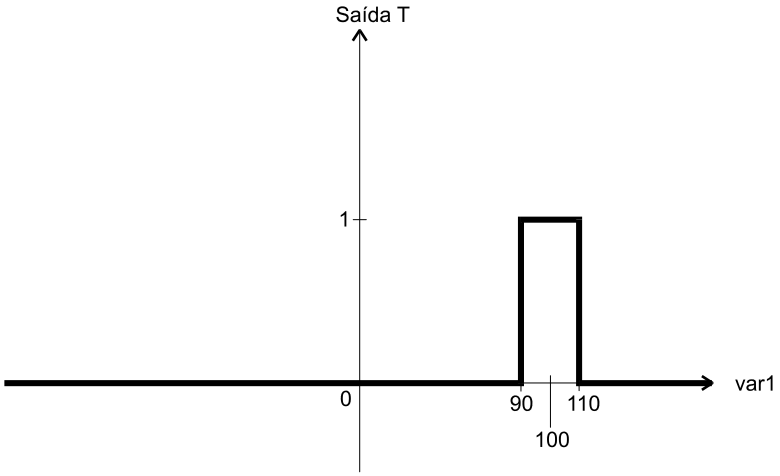
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Igual Inteiro.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Igual inteira. No caso, a variável var1 é comparada com a constante 100 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 igual a 100, dentro da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores entre 90 e 110 (incluindo os valores 90 e 110). O teste será válido quando var1 for 90, 91, 92, ..., 108, 109, ou 110. Caso o bloco fosse editado e o valor de Faixa fosse trocado para zero, o bloco só ligaria o nodo T caso var1 assumisse o valor 100.



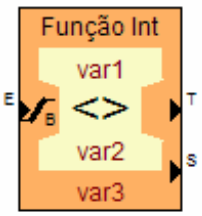
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação inteira Igual. O valor de comparação é 100 e a faixa é 10.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Igual Inteiro Var
COMPARAÇÃO - Igual LongInt
COMPARAÇÃO - Igual LongInt Var
COMPARAÇÃO - Igual Word
COMPARAÇÃO - Igual Word Var
COMPARAÇÃO - Igual Real
COMPARAÇÃO - Igual Real Var

COMPARAÇÃO - Diferente Inteiro

Este bloco efetua a comparação entre dois operandos inteiros, ligando o nodo de Teste (T) caso os dois operandos sejam diferentes. Note que um terceiro operando inteiro permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) não é acionado.



Operando1 e Operando2 são os operandos a serem comparados e Faixa é o valor dentro do qual

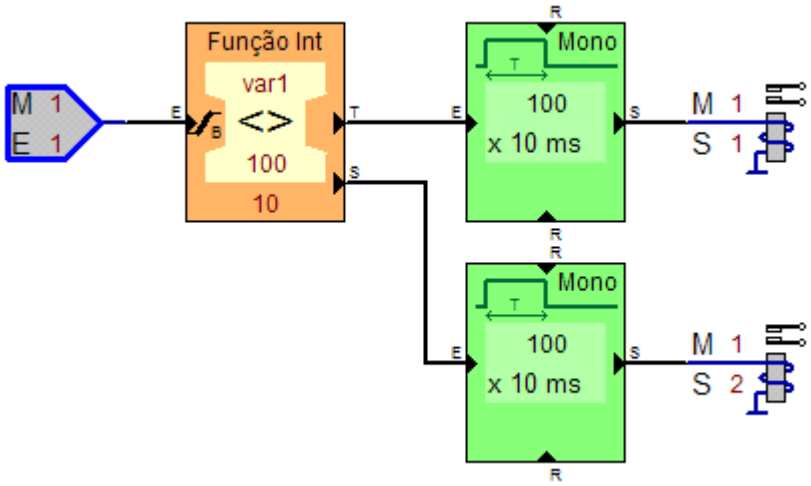
os dois operandos são considerados iguais. Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, os dois operandos devem ter valor idêntico para não haver o acionamento do nodo de Teste (T). Em todos os outros casos o nodo T é acionado.

Note que Operando1, Operando2 e Faixa podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 ≠ Operando2 fora da Faixa especificada.

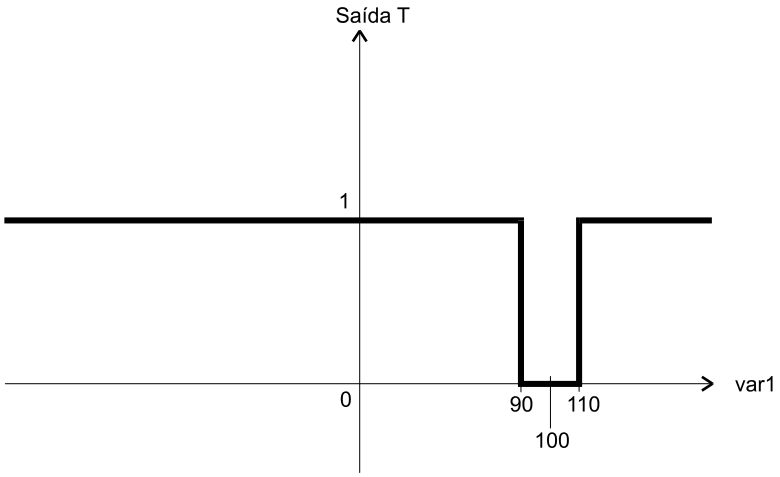
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Diferente Inteiro.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Diferente inteira. No caso, a variável var1 é comparada com a constante 100 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 diferente de 100, fora da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores menores que 90 ou maiores que 110 (excluindo os valores 90 e 110). O teste será válido quando var1 for ...87, 88, 89 ou 111, 112, 113.... Caso o bloco fosse editado e o valor de Faixa fosse trocado para zero, o bloco só desligaria o nodo T caso var1 assumisse o valor 100.



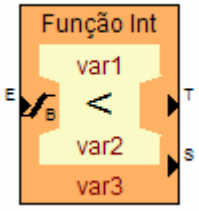
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação inteira Diferente. O valor de comparação é 100 e a faixa é 10.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Diferente Inteiro Var

COMPARAÇÃO - Menor Inteiro

Este bloco efetua a comparação entre dois operandos inteiros, ligando o nodo de Teste (T) caso o primeiro operando seja menor que o segundo operando. Note que um terceiro operando inteiro permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) não é acionado.



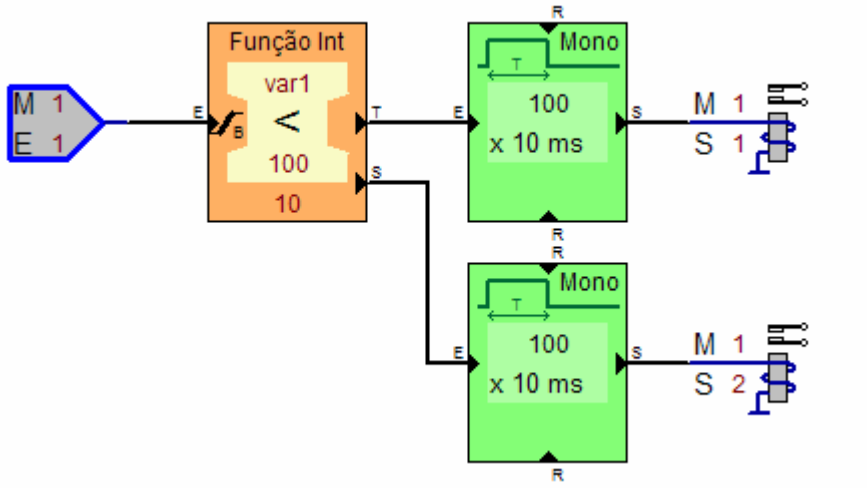
Operando1 e Operando2 são os operandos a serem comparados e Faixa é o valor dentro do qual os dois operandos são considerados iguais. Assim, o nodo de Teste (T) é acionado se Operando1 for menor que (Operando2 - Faixa). Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, o Operando1 deve ser menor que Operando2 para haver o acionamento do nodo de Teste (T).

Note que Operando1, Operando2 e Faixa podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 < Operando2 fora da Faixa especificada.

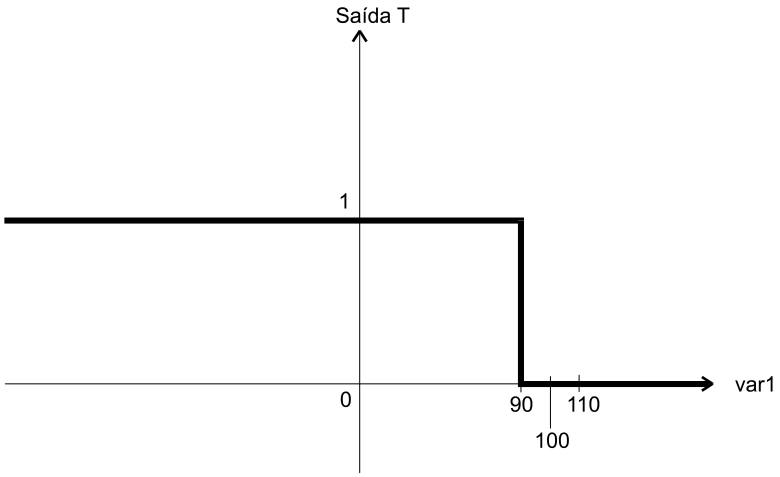
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Menor Inteiro.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Menor inteira. No caso, a variável var1 é comparada com a constante 100 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 menor que 100, fora da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores menores que 90 (excluindo o valor 90). O teste será válido quando var1 for ...87, 88, 89. Caso o bloco fosse editado e o valor de Faixa fosse trocado para zero, o bloco ligaria o nodo T caso var1 assumisse valores menores que 100 (...97, 98, 99).



Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação inteira Menor. O valor de comparação é 100 e a faixa é

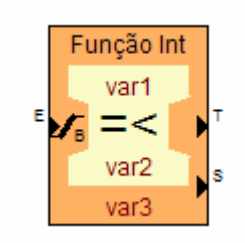
10.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Menor Inteiro Var
COMPARAÇÃO - Menor LongInt
COMPARAÇÃO - Menor LongInt Var
COMPARAÇÃO - Menor Word
COMPARAÇÃO - Menor Word Var
COMPARAÇÃO - Menor Real
COMPARAÇÃO - Menor Real Var

COMPARAÇÃO - Menor Igual Inteiro

Este bloco efetua a comparação entre dois operandos inteiros, ligando o nodo de Teste (T) caso o primeiro operando seja menor ou igual ao segundo operando. Note que um terceiro operando inteiro permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) é acionado.



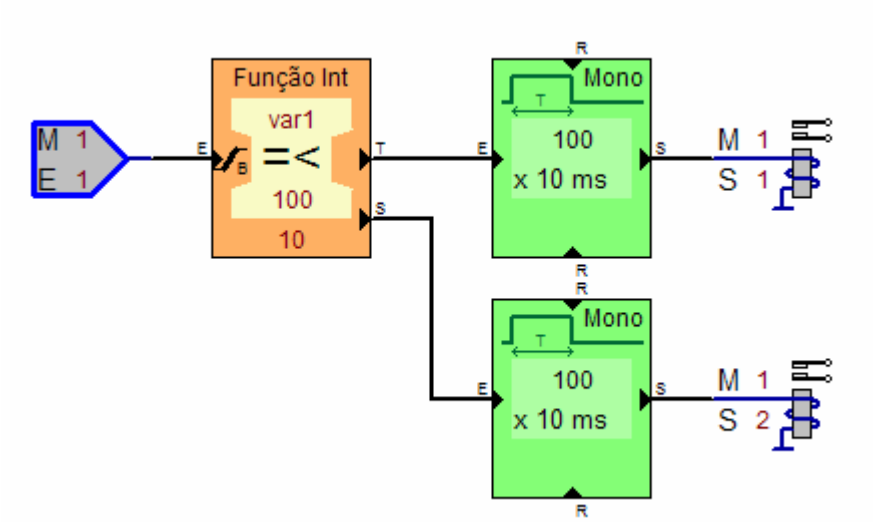
Operando1 e Operando2 são os operandos a serem comparados e Faixa é o valor dentro do qual os dois operandos são considerados iguais. Assim, o nodo de Teste (T) é acionado se Operando1 for menor ou igual que (Operando2 + Faixa). Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, o Operando1 deve ser menor ou igual que Operando2 para haver o acionamento do nodo de Teste (T).

Note que Operando1, Operando2 e Faixa podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 ≤ Operando2 dentro da Faixa especificada.

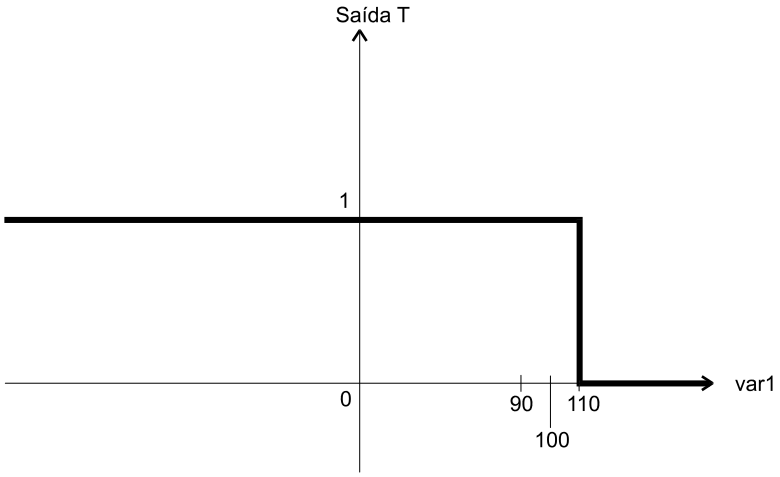
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Menor Igual Inteiro.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Menor Igual inteira. No caso, a variável var1 é comparada com a constante 100 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 menor ou igual a 100, dentro da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores menores ou iguais a 110 (incluindo o valor 110). O teste será válido quando var1 for ...108, 109, 110. Caso o bloco fosse editado e o valor de Faixa fosse trocado para zero, o bloco ligaria o nodo T caso var1 assumisse valores menores ou iguais a 100 (...97, 98, 99, 100).



Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação inteira Menor Igual. O valor de comparação é 100 e a faixa é 10.

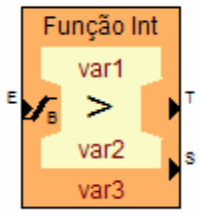
Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o

teste resulte falso.

Veja também:
COMPARAÇÃO - Menor Igual Inteiro Var

COMPARAÇÃO - Maior Inteiro

Este bloco efetua a comparação entre dois operandos inteiros, ligando o nodo de Teste (T) caso o primeiro operando seja maior que o segundo operando. Note que um terceiro operando inteiro permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) não é acionado.



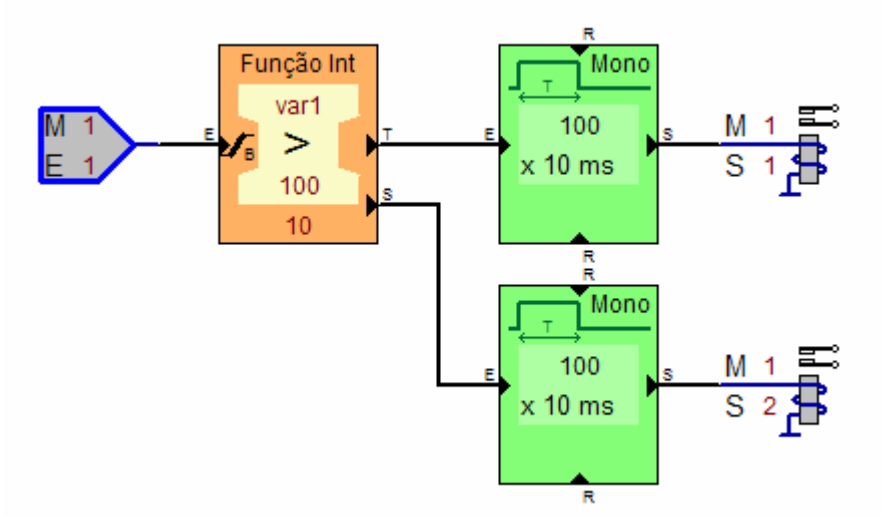
Operando1 e Operando2 são os operandos a serem comparados e Faixa é o valor dentro do qual os dois operandos são considerados iguais. Assim, o nodo de Teste (T) é acionado se Operando1 for maior que (Operando2 + Faixa). Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, o Operando1 deve ser maior que Operando2 para haver o acionamento do nodo de Teste (T).

Note que Operando1, Operando2 e Faixa podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 > Operando2 fora da Faixa especificada.

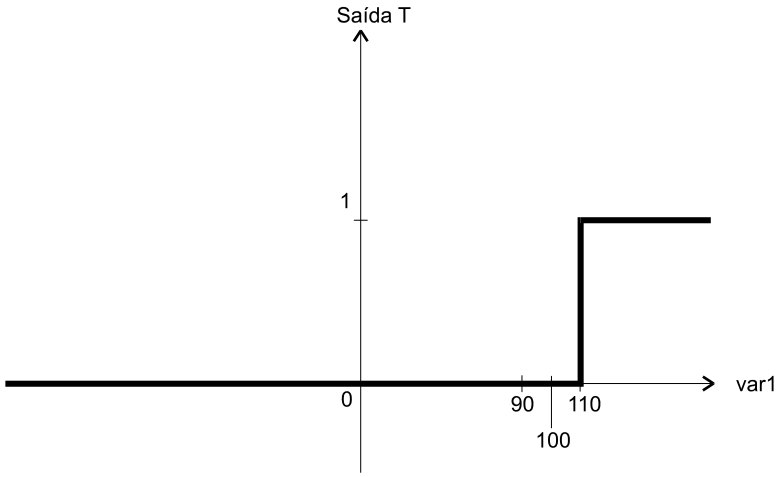
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Maior Inteiro.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Maior inteira. No caso, a variável var1 é comparada com a constante de valor 100 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 maior que 100, fora da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores maiores que 110 (excluindo o valor 110). O teste será válido quando var1 for 111, 112, 113.... Caso o bloco fosse editado e o valor de Faixa fosse trocado para zero, o bloco ligaria o nodo T caso var1 assumisse valores maiores que 100 (111, 112, 113...).



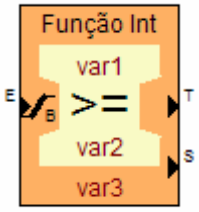
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação inteira Maior. O valor de comparação é 100 e a faixa é 10.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Maior Inteiro Var
COMPARAÇÃO - Maior LongInt
COMPARAÇÃO - Maior LongInt Var
COMPARAÇÃO - Maior Word
COMPARAÇÃO - Maior Word Var
COMPARAÇÃO - Maior Real
COMPARAÇÃO - Maior Real Var

COMPARAÇÃO - Maior Igual Inteiro

Este bloco efetua a comparação entre dois operandos inteiros, ligando o nodo de Teste (T) caso o primeiro operando seja maior ou igual ao segundo operando. Note que um terceiro operando inteiro permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) é acionado.



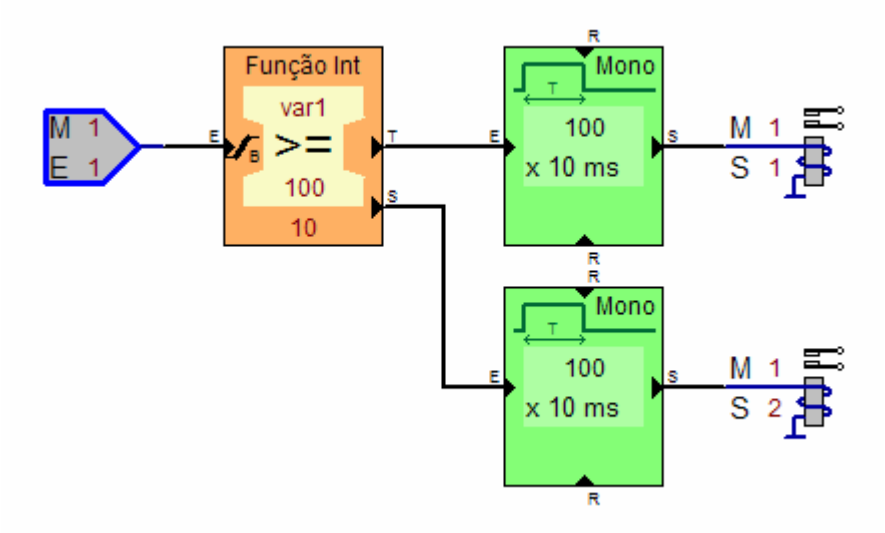
Operando1 e Operando2 são os operandos a serem comparados e Faixa é o valor dentro do qual os dois operandos são considerados iguais. Assim, o nodo de Teste (T) é acionado se Operando1 for maior ou igual que (Operando2 - Faixa). Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, o Operando1 deve ser maior ou igual que Operando2 para haver o acionamento do nodo de Teste (T).

Note que Operando1, Operando2 e Faixa podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 ≥ Operando2 dentro da Faixa especificada.

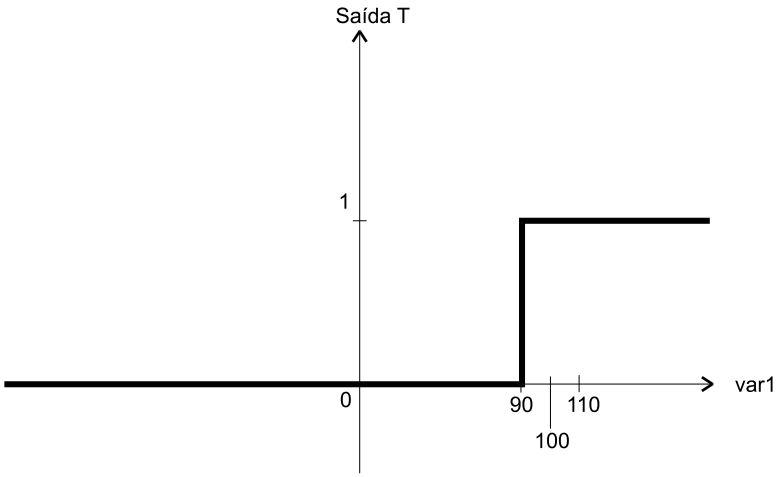
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Maior Igual Inteiro.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Maior Igual inteira. No caso, a variável var1 é comparada com a constante de valor 100 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 maior ou igual a 100, dentro da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores maiores ou iguais a 90 (incluindo o valor 90). O teste será válido quando var1 for 90, 91, 92.... Caso o bloco fosse editado e o valor de Faixa fosse trocado para zero, o bloco ligaria o nodo T caso var1 assumisse valores maiores ou iguais a 100 (100, 101, 102...).



Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação inteira Maior Igual. O valor de comparação é 100 e a faixa é 10.

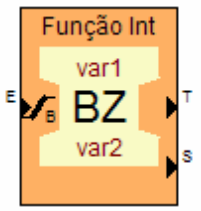
Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o

teste resulte falso.

Veja também:
COMPARAÇÃO - Maior Igual Inteiro Var

COMPARAÇÃO - Bit Zero Inteiro

Este bloco efetua a verificação se determinado bit do Operando1 está desligado. Note que o bit é indicado pelo Operando2 (de bit 0 a bit 15). O nodo de Teste (T) é acionado caso o bit especificado do Operando1 seja zero.

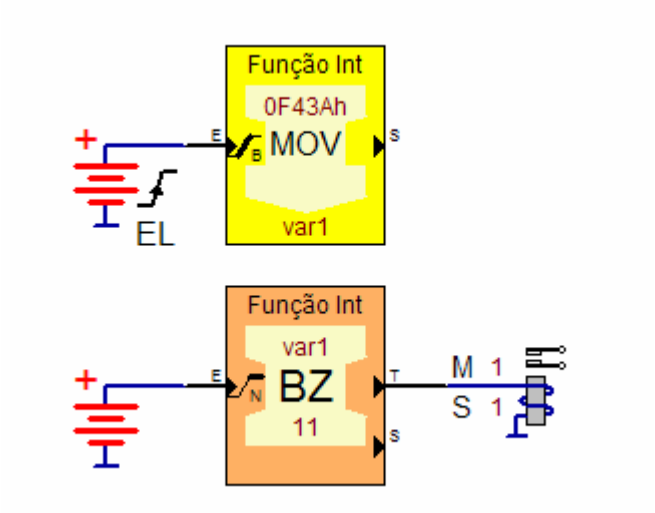


Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, caso o bit selecionado seja zero.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Bit Zero Inteiro.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Bit Zero inteira. No caso, o bit 11 da variável var1 é testado e, caso seja zero, o nodo de Teste (T) do bloco é acionado, ligando a saída S1 do módulo de Expansão de Entradas/Saídas (µDX210).

Note que var1 é inicializada com valor 0F43Ah em hexadecimal, o que corresponde a 1111010000111010b em binário. O bit 11 de var1, portanto, está desligado e isso implica que o teste é verdadeiro (BZ) e o nodo de Teste (T) é acionado. Como foi usado um bloco de Nodo EL para acionar a inicialização da variável var1, é preciso desenergizar momentaneamente o controlador para que a variável inicialize com valor F43Ah. Caso se queira que a variável inicialize ao rodar o programa aplicativo substitua o bloco de Nodo EL por um bloco de Nodo Reset.

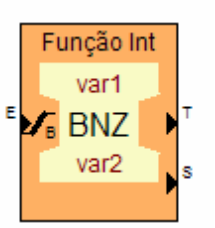
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	0	0	1	1	1	0	1	0

Valor binário correspondente a 0F43Ah

Veja também:
COMPARAÇÃO - Bit Zero Word

COMPARAÇÃO - Bit Not Zero Inteiro

Este bloco efetua a verificação se determinado bit do Operando1 está ligado. Note que o bit é indicado pelo Operando2 (de bit 0 a bit 15). O nodo de Teste (T) é acionado caso o bit especificado do Operando1 seja um.

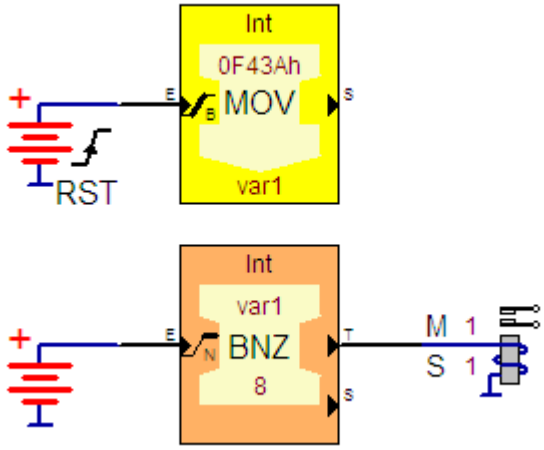


Note que Operando1 e Operando2 podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, caso o bit selecionado seja um.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Bit Not Zero Inteiro.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Bit Not Zero inteira. No caso, o bit 8 da variável var1 é testado e, caso seja um, o nodo de Teste (T) do bloco é acionado, ligando a saída S1 do módulo de Expansão de Entradas/Saídas (µDX210).

Note que var1 é inicializada com valor 0F43Ah em hexadecimal, o que corresponde a 1111010000111010b em binário. O bit 8 de var1, portanto, está desligado e isso implica que o teste é falso (BNZ) e o nodo de Teste (T) não é acionado. Neste programa foi usado o bloco de Nodo Reset em vez do bloco de Nodo EL (Energia Liga) para inicializar a variável var1. Com isso, não é preciso desligar momentaneamente a energia elétrica do controlador para inicializar a variável. O bloco de Nodo Reset gera um pulso sempre que ocorre um reset (e isso acontece quando é enviada a ordem para executar um programa aplicativo). Já o bloco de Nodo EL somente gera um pulso no retorno da energia elétrica e, portanto, exige uma desenergização momentânea.

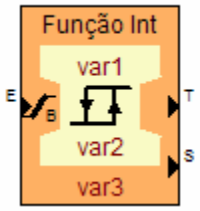
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	0	0	1	1	1	0	1	0

Valor binário correspondente a 0F43Ah

Veja também:
COMPARAÇÃO - Bit Not Zero Word

COMPARAÇÃO - Histerese Inteiro

Este bloco efetua a comparação com histerese entre dois operandos inteiros, ligando o nodo de Teste (T) quando Operando1 for maior ou igual a (Operando2 + Histerese), e desligando o nodo de Teste (T) quando Operando1 for menor ou igual a (Operando2 - Histerese). Note que um terceiro operando inteiro permite especificar a histerese.



Note que Operando1, Operando2 e Histerese podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Este bloco é muito útil, por exemplo, para acionar uma determinada saída conforme dois valores de comparação. Em um controle de compressor de ar-condicionado não devemos acionar e desacionar o compressor conforme a temperatura suba de determinado ponto, pois neste caso ficaríamos constantemente acionando e desacionando o compressor. Se a temperatura desejada é 21,0°C e colocássemos um teste simples o compressor acionaria em 21,0°C e desacionaria em 20,9°C (considerando uma resolução de 0,1°C para o sensor de temperatura). Como a diferença de temperatura é muito pequena, logo estaríamos novamente em 21,0°C e ele seria novamente acionado e assim por diante, diminuindo a vida útil do equipamento.

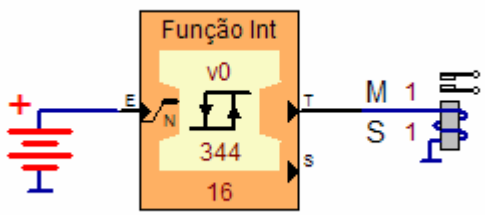
Então, é mais conveniente fixar duas temperaturas distintas, uma para ligar o compressor e outra para desligá-lo. Digamos que em 22,0°C ele será acionado e em 20,0°C desacionado. Podemos fazer isso com um simples bloco de comparação com histerese. Para isso bastaria especificar o valor correspondente a 21,0°C no Operando2, enquanto o Operando1 aponta para o sensor de temperatura. Já na Histerese especifica-se um valor correspondente a 1,0°C (histerese desejada de ± 1,0°C). Se o sensor de temperatura estiver ligado a uma das entradas analógicas, programada para escala de 0-2,5V, e fornecer 10mV/°C (é o caso, por exemplo, do sensor LM35, da National Semiconductors), teremos:

2,5V / 4095 = 610,5µV de resolução.

Logo, cada 1°C corresponde a 10mV / 610,5µV = 16,38 passos.

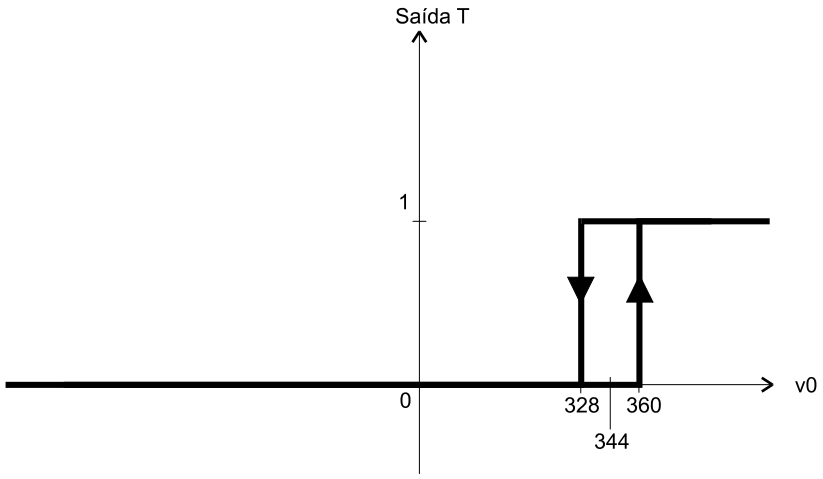
Assim, em 21,0°C teremos 210mV na entrada analógica, o que corresponde a um valor de 210mV / 610,35µV = 343,98 passos @ 344 passos.

Então, devemos comparar a temperatura com o valor 344, e inserir uma histerese de 16 no bloco. Com isso, a saída será acionada quando o valor da entrada analógica corresponder a 344 + 16 = 360 (o que corresponde a 360 * 610,5µV = 21,978mV @ 22,0mV, ou seja, 22,0°C). Já a saída será desligada quando o valor da entrada analógica corresponder a 344 - 16 = 328 (o que corresponde a 328 * 610,5µV = 20,024mV @ 20,0mV, ou seja, 20,0°C). A seguir temos este exemplo implementado no programa PG.



Exemplo de Programa Aplicativo: **Comparação - Histerese Inteiro.dxg**

Note que foi utilizada a entrada analógica E1 (que corresponde a v0). O nodo de saída Teste (T) é acionado sempre que o bloco detecta que v0 assumiu valor maior ou igual a 360, e desliga este nodo sempre que v0 assumir valores menores ou iguais a 328.

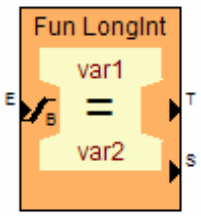


Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável v0, para o bloco de comparação inteira Histerese. O valor de comparação é 344 e a histerese é 16.

Veja também:
COMPARAÇÃO - Histerese Inteiro Var

COMPARAÇÃO - Igual LongInt

Este bloco efetua a comparação entre dois operandos longint, ligando o nodo de Teste (T) caso os dois operandos sejam iguais. Note que este bloco, ao contrário do bloco de comparação Igual Inteiro, não possui um terceiro operando para especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais.



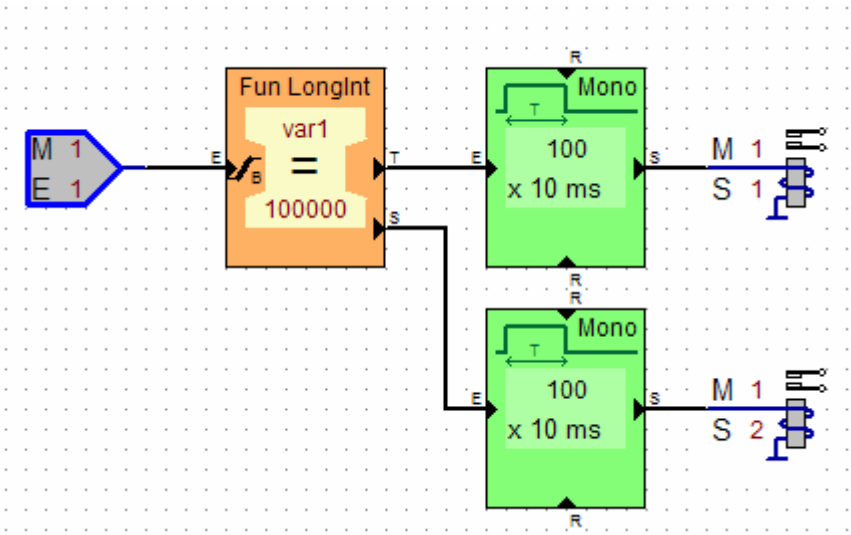
Operando1 e Operando2 são os operandos a serem comparados. Note que Operando1 e Operando2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 =

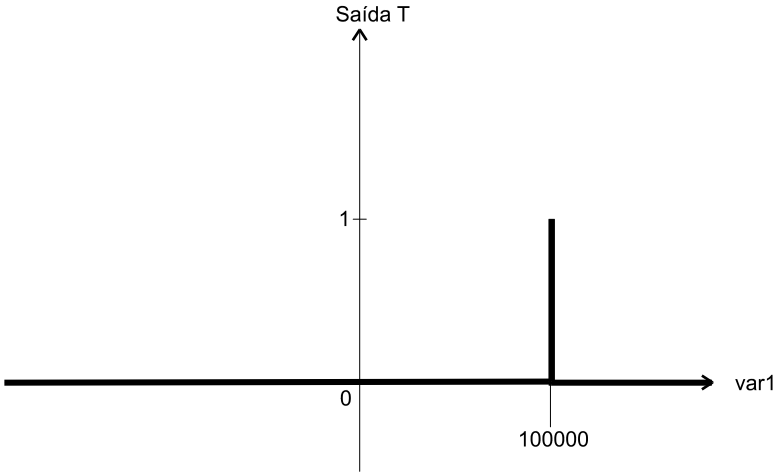
Operando2.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Igual LongInt.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Igual longint. No caso, a variável var1 é comparada com a constante 100.000 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).



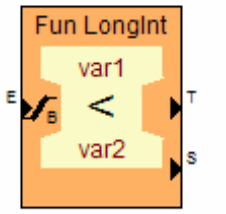
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação longint Igual. O nodo T só é ligado quando o valor de var1 é 100.000.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Igual LongInt Var
COMPARAÇÃO - Igual Inteiro
COMPARAÇÃO - Igual Inteiro Var
COMPARAÇÃO - Igual Word
COMPARAÇÃO - Igual Word Var
COMPARAÇÃO - Igual Real
COMPARAÇÃO - Igual Real Var

COMPARAÇÃO - Menor LongInt

Este bloco efetua a comparação entre dois operandos longint, ligando o nodo de Teste (T) caso o primeiro operando seja menor que o segundo operando.

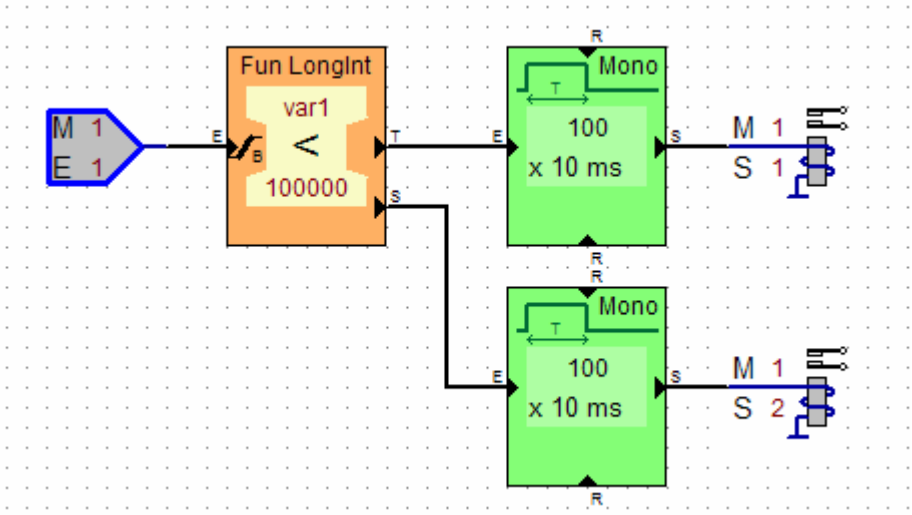


Operando1 e Operando2 são os operandos a serem comparados. Note que Operando1 e Operando2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 < Operando2.

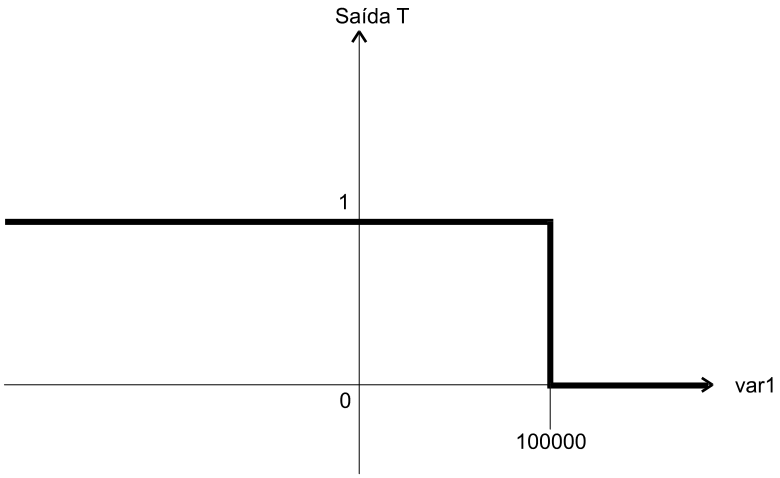
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Menor LongInt.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Menor longint. No caso, a variável var1 é comparada com a constante 100.000 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 menor que 100.000. Ou seja, o nodo T será acionado quando a variável var1 assumir valores ...99.997, 99.998, 99.999.



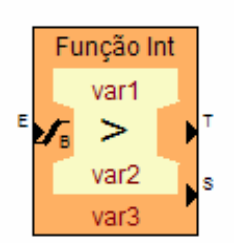
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação longint Menor. O nodo T é acionado sempre que var1 assume valor menor que 100.000.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:

COMPARAÇÃO - Maior LongInt

Este bloco efetua a comparação entre dois operandos longint, ligando o nodo de Teste (T) caso o primeiro operando seja maior que o segundo operando.

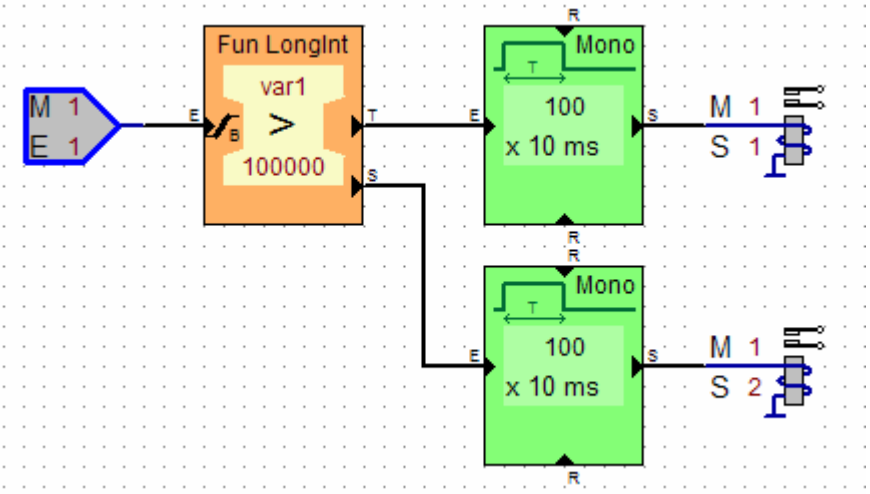


Operando1 e Operando2 são os operandos a serem comparados. Note que Operando1 e Operando2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 > Operando2.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

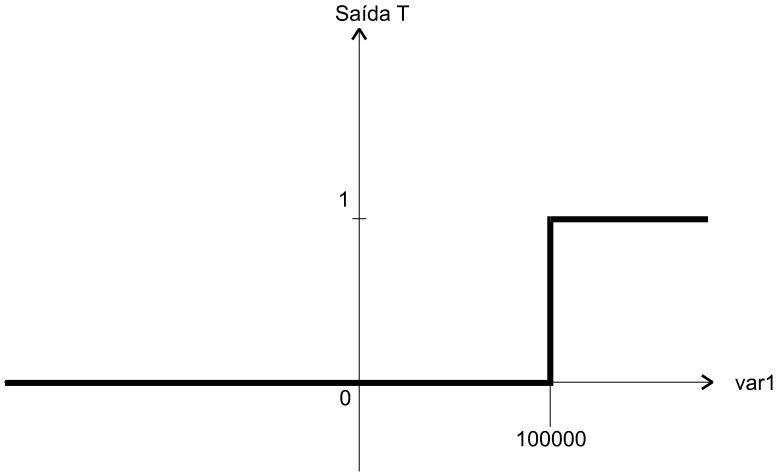


Exemplo de Programa Aplicativo: **Comparação - Maior LongInt.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Maior longint. No caso, a variável var1 é comparada com a constante 100.000 cada vez que a entrada E1 do módulo de

Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 maior que 100.000. Ou seja, o nodo T será acionado quando a variável var1 assumir valores de 100.001, 100.002, 100.003,....



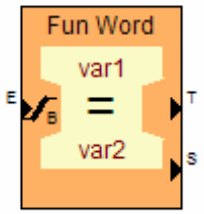
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação longint Maior. O nodo T é acionado sempre que var1 for maior que 100.000.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Maior LongInt Var
COMPARAÇÃO - Maior Inteiro
COMPARAÇÃO - Maior Inteiro Var
COMPARAÇÃO - Maior Word
COMPARAÇÃO - Maior Word Var
COMPARAÇÃO - Maior Real
COMPARAÇÃO - Maior Real Var

COMPARAÇÃO - Igual Word

Este bloco efetua a comparação entre dois operandos word, ligando o nodo de Teste (T) caso os dois operandos sejam iguais. Note que este bloco, ao contrário do bloco de comparação Igual Inteiro, não possui um terceiro operando para especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais.

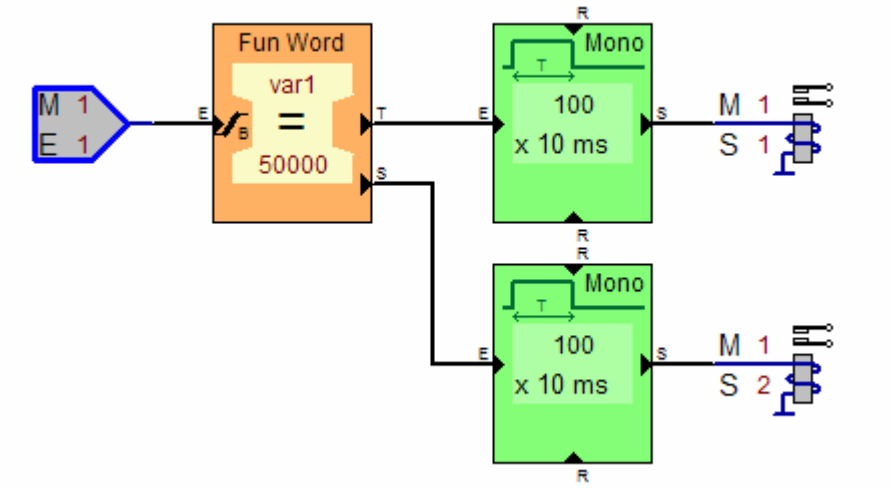


Operando1 e Operando2 são os operandos a serem comparados. Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

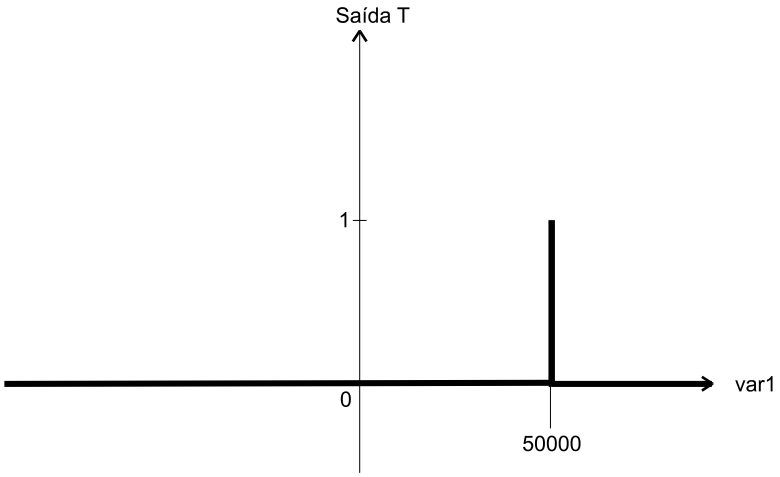
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 = Operando2.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Igual Word.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Igual word. No caso, a variável var1 é comparada com a constante 50.000 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).



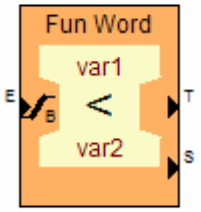
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação word Igual. O nodo T só é ligado quando o valor de var1 é 50.000. Note que o limite para números word é 65535.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Igual Word Var
COMPARAÇÃO - Igual Inteiro
COMPARAÇÃO - Igual Inteiro Var
COMPARAÇÃO - Igual LongInt
COMPARAÇÃO - Igual LongInt Var
COMPARAÇÃO - Igual Real
COMPARAÇÃO - Igual Real Var

COMPARAÇÃO - Menor Word

Este bloco efetua a comparação entre dois operandos word, ligando o nodo de Teste (T) caso o primeiro operando seja menor que o segundo operando.



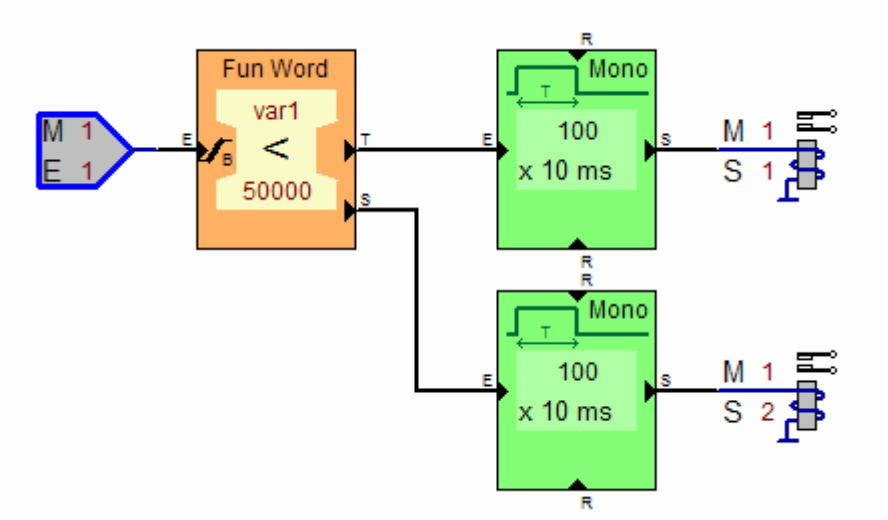
Operando1 e Operando2 são os operandos a serem comparados. Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 <

Operando2.

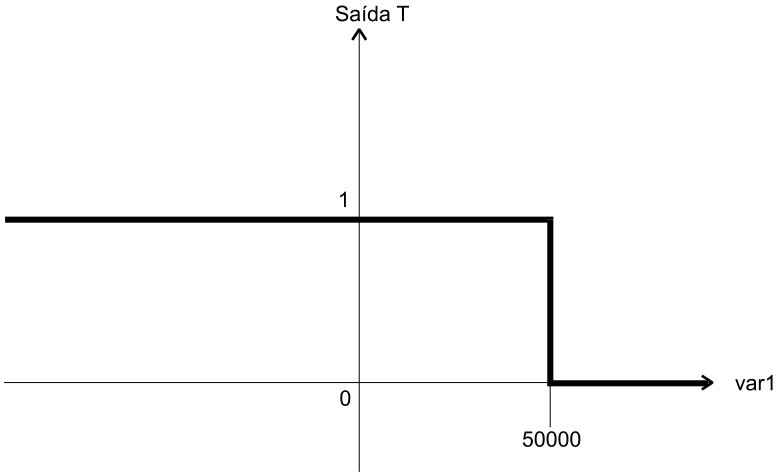
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Menor Word.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Menor word. No caso, a variável var1 é comparada com a constante 50.000 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 menor que 50.000. Ou seja, o nodo T será acionado quando a variável var1 assumir valores ...49.997, 49.998, 49.999.



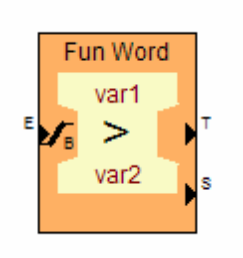
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação word Menor. O nodo T é acionado sempre que var1 assume valor menor que 50.000. Note que o limite de representação de variáveis word é 65.535.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Menor Word Var
COMPARAÇÃO - Menor Inteiro
COMPARAÇÃO - Menor Inteiro Var
COMPARAÇÃO - Menor LongInt
COMPARAÇÃO - Menor LongInt Var
COMPARAÇÃO - Menor Real
COMPARAÇÃO - Menor Real Var

COMPARAÇÃO - Maior Word

Este bloco efetua a comparação entre dois operandos word, ligando o nodo de Teste (T) caso o primeiro operando seja maior que o segundo operando.

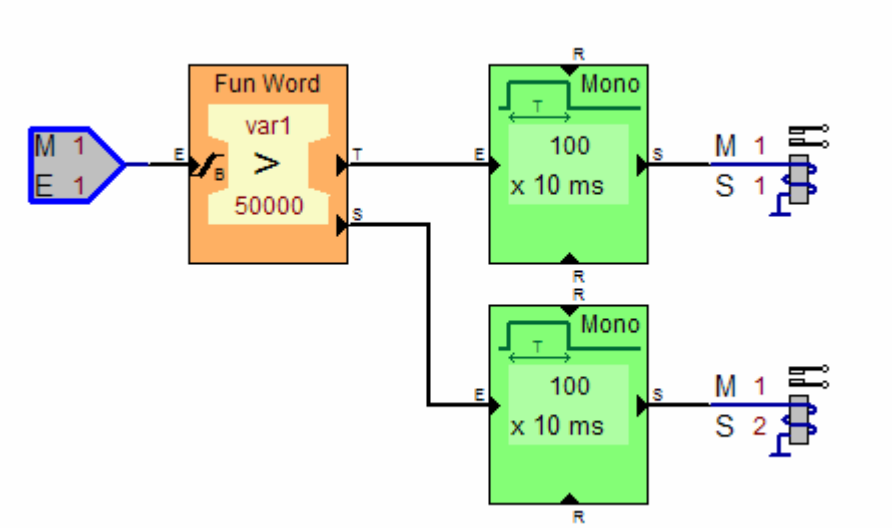


Operando1 e Operando2 são os operandos a serem comparados. Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 > Operando2.

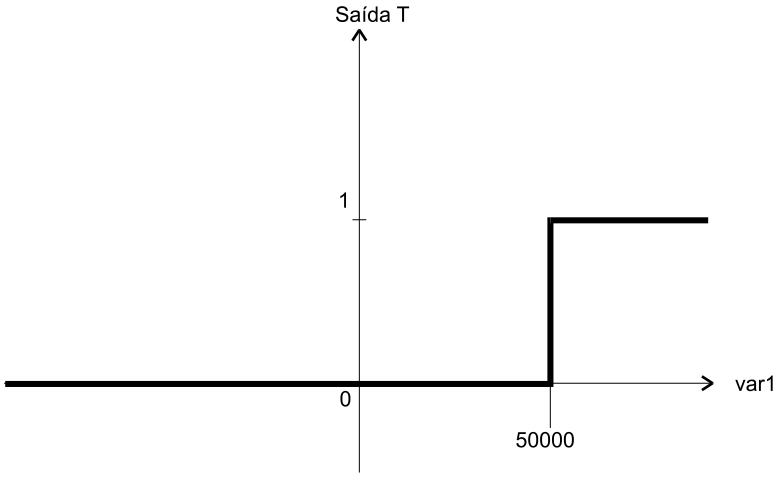
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Maior Word.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Maior word. No caso, a variável var1 é comparada com a constante 50.000 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 maior que 50.000. Ou seja, o nodo T será acionado quando a variável var1 assumir valores de 50.001, 50.002, 50.003,....



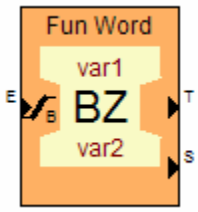
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação word Maior. O nodo T é acionado sempre que var1 for maior que 50.000. Note que o limite de representação para números word é de 65.535.

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Maior Word Var

COMPARAÇÃO - Bit Zero Word

Este bloco efetua a verificação se determinado bit do Operando1 está desligado. Note que o bit é indicado pelo Operando2 (de bit 0 a bit 15).



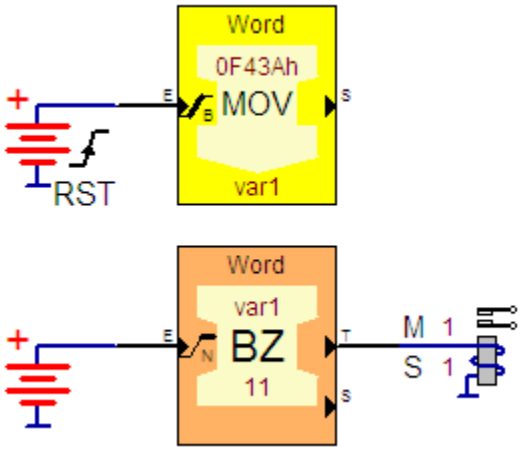
O nodo de Teste (T) é acionado caso o bit especificado do Operando1 seja zero.

Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, caso o bit selecionado seja zero.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Bit Zero Word.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Bit Zero word. No caso, o bit 11 da variável var1 é testado e, caso seja zero, o nodo de Teste (T) do bloco é acionado, ligando

a saída S1 do módulo de Expansão de Entradas/Saídas (µDX210).

Note que var1 é inicializada com valor 0F43Ah em hexadecimal, o que corresponde a 1111010000111010b em binário. O bit 11 de var1, portanto, está desligado e isso implica que o teste é verdadeiro (BZ) e o nodo de Teste (T) é acionado. Neste programa foi usado o bloco de Nodo Reset em vez do bloco de Nodo EL (Energia Liga) para inicializar a variável var1. Com isso, não é preciso desligar momentaneamente a energia elétrica do controlador para inicializar a variável. O bloco de Nodo Reset gera um pulso sempre que ocorre um reset (e isso acontece quando é enviada a ordem para executar um programa aplicativo). Já o bloco de Nodo EL somente gera um pulso no retorno da energia elétrica e, portanto, exige uma desenergização momentânea.

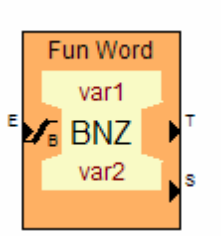
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	0	0	1	1	1	0	1	0

Valor binário correspondente a 0F43Ah

Veja também:
COMPARAÇÃO - Bit Zero Inteiro

COMPARAÇÃO - Bit Not Zero Word

Este bloco efetua a verificação se determinado bit do Operando1 está ligado. Note que o bit é indicado pelo Operando2 (de bit 0 a bit 15).



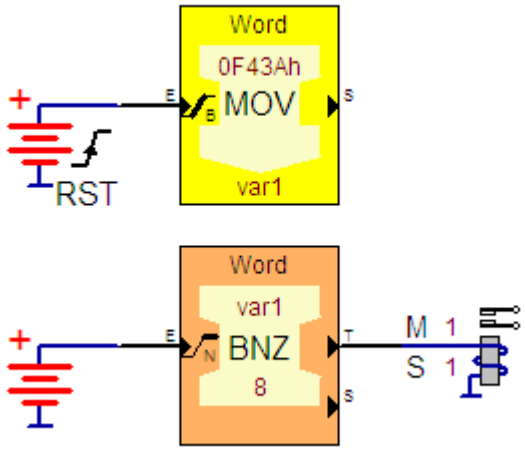
O nodo de Teste (T) é acionado caso o bit especificado do Operando1 seja um.

Note que Operando1 e Operando2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, caso o bit selecionado seja um.

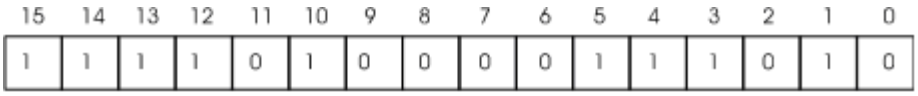
Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Bit Not Zero Word.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Bit Not Zero word. No caso, o bit 8 da variável var1 é testado e, caso seja um, o nodo de Teste (T) do bloco é acionado, ligando a saída S1 do módulo de Expansão de Entradas/Saídas (µDX210).

Note que var1 é inicializada com valor 0F43Ah em hexadecimal, o que corresponde a 1111010000111010b em binário. O bit 8 de var1, portanto, está desligado e isso implica que o teste é falso (BNZ) e o nodo de Teste (T) não é acionado. Neste programa foi usado o bloco de Nodo Reset em vez do bloco de Nodo EL (Energia Liga) para inicializar a variável var1. Com isso, não é preciso desligar momentaneamente a energia elétrica do controlador para inicializar a variável. O bloco de Nodo Reset gera um pulso sempre que ocorre um reset (e isso acontece quando é enviada a ordem para executar um programa aplicativo). Já o bloco de Nodo EL somente gera um pulso no retorno da energia elétrica e, portanto, exige uma desenergização momentânea.

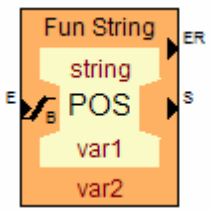


Valor binário correspondente a 0F43Ah

Veja também:
COMPARAÇÃO - Bit Not Zero Inteiro

COMPARAÇÃO - Posição no String

Este bloco testa se o caracter especificado (Caracter) ocorre no string apontado (String). O bloco retorna em Resultado a posição no string em que foi encontrada a enésima ocorrência do caracter, sendo que n é especificado em Núm.Ocorrências.



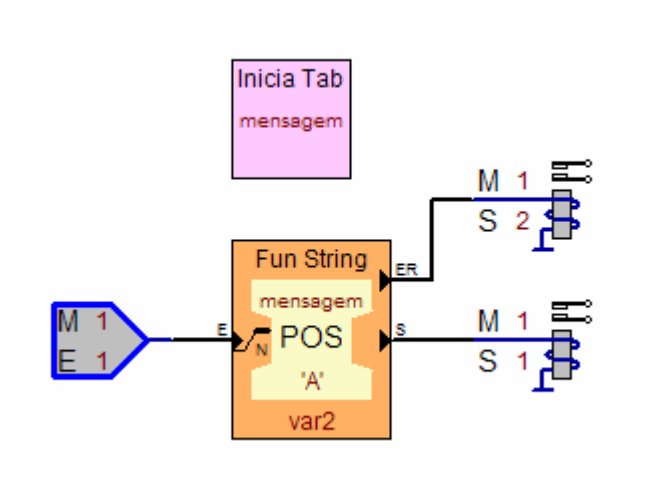
Note que Caracter, Tam.String e Núm.Ocorrências podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Resultado deve ser necessariamente uma variável inteira, pois irá receber o resultado da operação. Além disso, String é uma variável word que aponta para o string a ser analisado.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

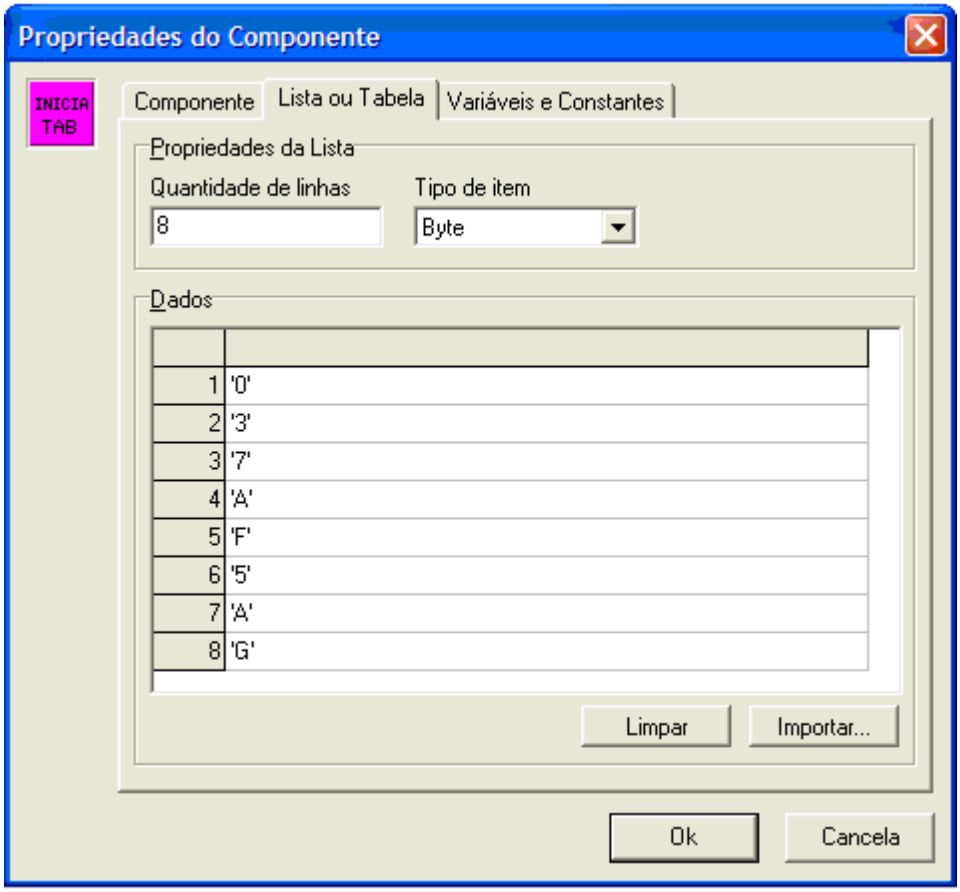
O nodo de erro (ER) só é acionado caso o Número de Ocorrências programado no bloco seja menor ou igual a zero.

Este bloco é útil, por exemplo, para verificar a posição de determinado caracter de separação entre dados em um string. Por exemplo, se tivermos um determinado dado iniciando com caracter "A" (65 ou 41h), podemos usar este bloco para localizá-lo em determinado string. Veja o exemplo a seguir:



Exemplo de Programa Aplicativo: **Comparação - Posição no String.dxg**

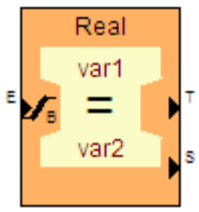
O string Mensagem foi inicializado com os seguintes valores:



Caso se especifique Tam.String (tamanho do string) de 8 posições e Núm.Ocorrências = 1 o programa irá retornar o valor 4 em Resultado (var2). Já se especificarmos Núm.Ocorrências =2 o programa irá retornar o valor 7 em Resultado (var2). Por fim, ao especificar Núm.Ocorrências = 3 o Resultado será zero (var2=0), já que não existem três caracteres "A" no string.

COMPARAÇÃO - Igual Real

Este bloco efetua a comparação entre dois operandos reais, ligando o nodo de Teste (T) caso os dois operandos sejam iguais.



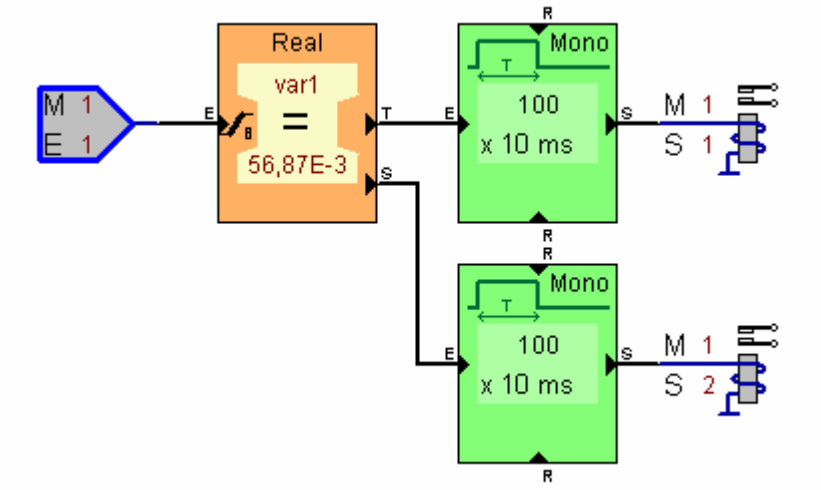
Note que este bloco, ao contrário do bloco de comparação Igual Inteiro, não possui um terceiro operando para especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais.

Operando1 e Operando2 são os operandos a serem comparados. Note que Operando1 e Operando2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 = Operando2.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Igual Real.dxg**

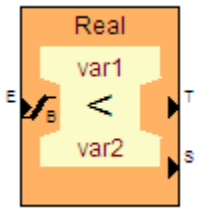
Acima temos um pequeno exemplo de uso do bloco de comparação Igual real. No caso, a variável var1 é comparada com a constante 56,87E-3 (0,05687) cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Igual Real Var
COMPARAÇÃO - Igual Inteiro
COMPARAÇÃO - Igual Inteiro Var
COMPARAÇÃO - Igual LongInt
COMPARAÇÃO - Igual LongInt Var
COMPARAÇÃO - Igual Word
COMPARAÇÃO - Igual Word Var

COMPARAÇÃO - Menor Real

Este bloco efetua a comparação entre dois operandos reais, ligando o nodo de Teste (T) caso o primeiro operando seja menor que o segundo operando.

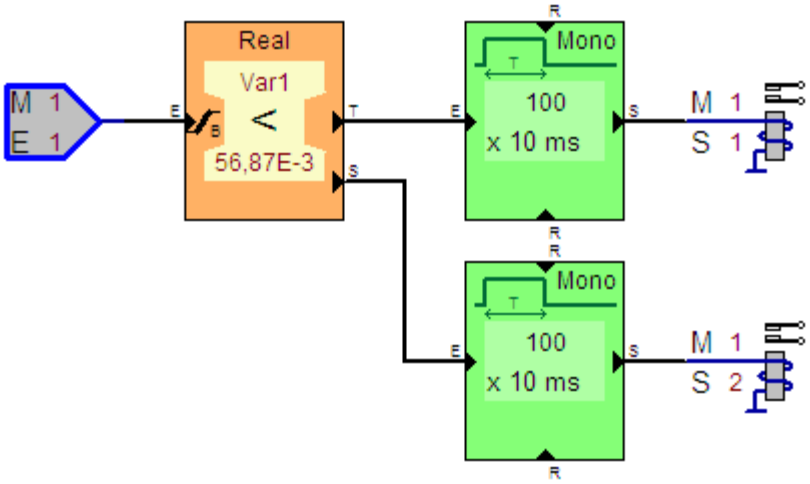


Operando1 e Operando2 são os operandos a serem comparados. Note que Operando1 e Operando2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 < Operando2.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Menor Real.dxg**

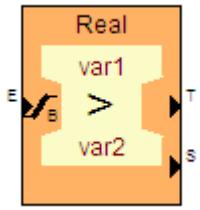
Acima temos um pequeno exemplo de uso do bloco de comparação Menor real. No caso, a variável var1 é comparada com a constante 56,87E-3 (0,05687) cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Menor Real Var
COMPARAÇÃO - Menor Inteiro
COMPARAÇÃO - Menor Inteiro Var
COMPARAÇÃO - Menor LongInt
COMPARAÇÃO - Menor LongInt Var
COMPARAÇÃO - Menor Word
COMPARAÇÃO - Menor Word Var

COMPARAÇÃO - Maior Real

Este bloco efetua a comparação entre dois operandos reais, ligando o nodo de Teste (T) caso o primeiro operando seja maior que o segundo operando.

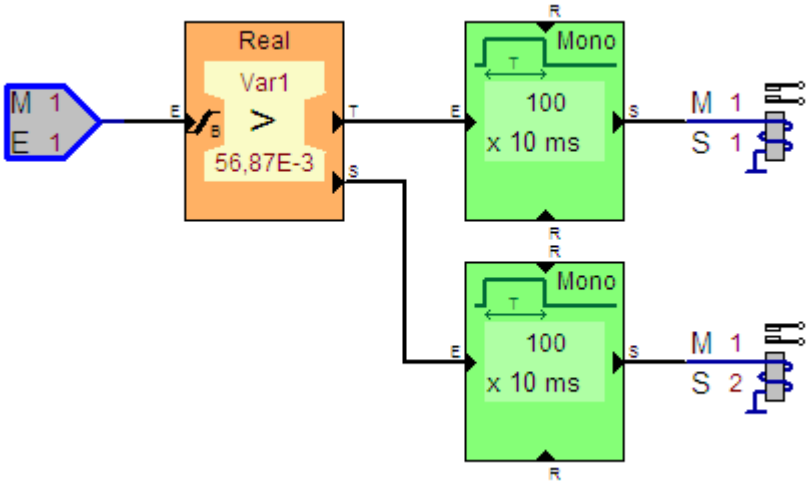


Operando1 e Operando2 são os operandos a serem comparados. Note que Operando1 e Operando2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso Operando1 > Operando2.

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Comparação - Maior Real.dxg**

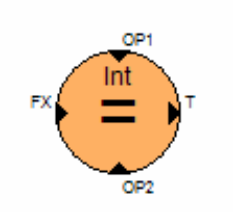
Acima temos um pequeno exemplo de uso do bloco de comparação Maior real. No caso, a variável var1 é comparada com a constante 56,87E-3 (0,05687) cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é acionada (já que a entrada E do bloco de Igual está selecionada por borda). Já as saídas S1 e S2 do mesmo módulo são acionadas conforme os nodos Teste (T) e Saída (S) do bloco. Note que foram colocados monoestáveis nas saídas T e S do bloco. Isso é necessário para visualizar o acionamento destas saídas, uma vez que elas ficam ativam durante apenas um ciclo de CLP no caso de entrada E por borda (se E estivesse por nível estas saídas seriam acionadas enquanto a entrada E ficasse energizada).

Atenção: Em controlador µDX201 versão 3.37 ou posterior está disponível nodo de saída adicional (nodo I) neste bloco. O nodo de teste invertido (I) liga caso o teste resulte falso.

Veja também:
COMPARAÇÃO - Maior Real Var
COMPARAÇÃO - Maior Inteiro
COMPARAÇÃO - Maior Inteiro Var
COMPARAÇÃO - Maior LongInt
COMPARAÇÃO - Maior LongInt Var
COMPARAÇÃO - Maior Word
COMPARAÇÃO - Maior Word Var

COMPARAÇÃO - Igual Inteiro Var

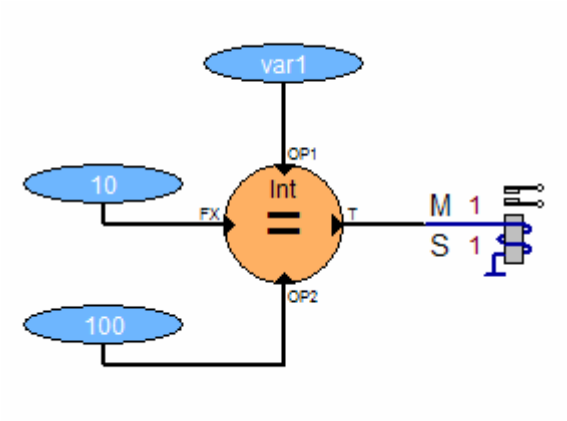
Este bloco efetua a comparação entre dois operandos inteiros (OP1 e OP2), ligando o nodo de Teste (T) caso os dois operandos sejam iguais. Note que um terceiro operando inteiro (FX) permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados e FX é o valor dentro do qual os dois operandos são considerados iguais. Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, os dois operandos devem ter valor idêntico para haver o acionamento do nodo de Teste (T).

Note que OP1, OP2 e FX podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

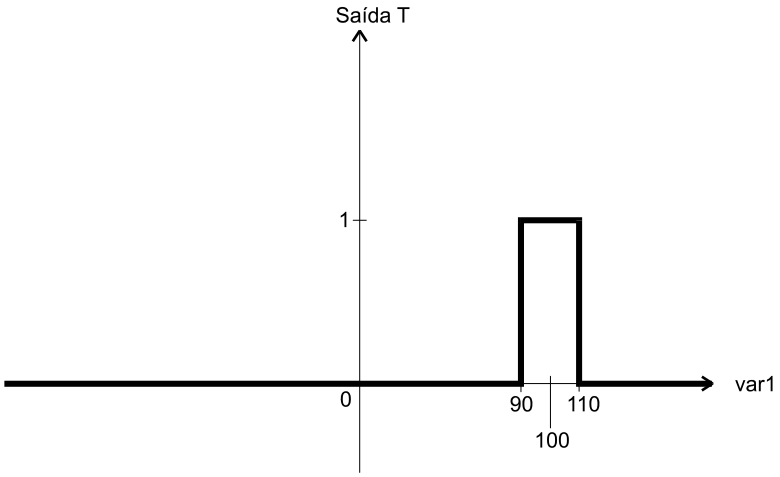
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 = OP2 dentro da faixa FX especificada.



Exemplo de Programa Aplicativo: **Comparação - Igual Inteiro Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Igual inteira Var. No caso, a variável var1 é comparada com a constante de valor 100 constantemente.

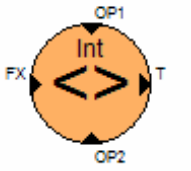
O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 igual a 100, dentro da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores entre 90 e 110 (incluindo os valores 90 e 110). O teste será válido quando var1 for 90, 91, 92, ..., 108, 109, ou 110. Caso o programa fosse editado e o valor de FX fosse trocado para zero, o bloco só ligaria o nodo T caso var1 assumisse o valor 100.



Veja também:

COMPARAÇÃO - Diferente Inteiro Var

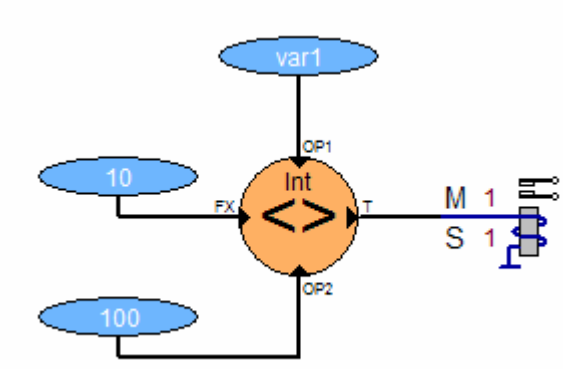
Este bloco efetua a comparação entre dois operandos inteiros (OP1 e OP2), ligando o nodo de Teste (T) caso os dois operandos sejam diferentes. Note que um terceiro operando inteiro (FX) permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) não é acionado. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados e FX é o valor dentro do qual os dois operandos são considerados iguais. Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, os dois operandos devem ter valor idêntico para não haver o acionamento do nodo de Teste (T). Em todos os outros casos o nodo T é acionado.

Note que OP1, OP2 e FX podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 ≠ OP2 fora da faixa FX especificada.

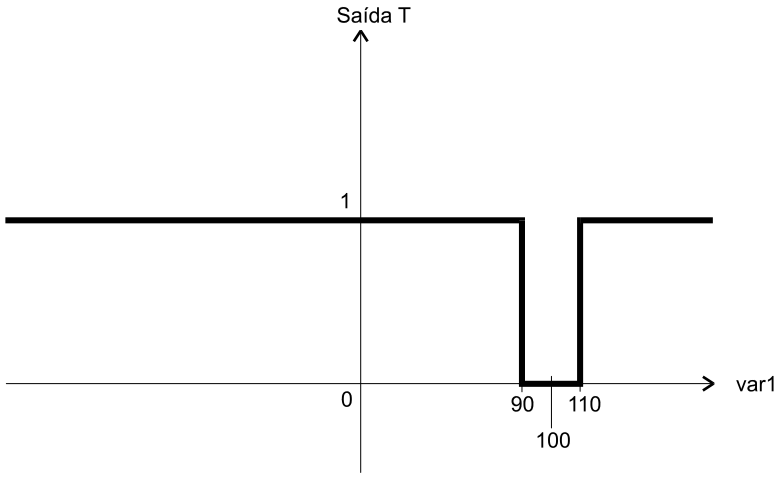


Exemplo de Programa Aplicativo: **Comparação - Diferente Inteiro Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Diferente inteira Var. No caso, a variável var1 é comparada com a constante 100.

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 diferente de 100, fora da faixa FX especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores menores que 90 ou maiores que 110 (excluindo os valores 90 e 110). O teste será válido quando var1 for ...87, 88, 89 ou 111, 112, 113.... Caso o programa fosse

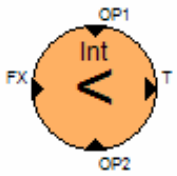
editado e o valor de faixa FX fosse trocado para zero, o bloco só desligaria o nodo T caso var1 assumisse o valor 100.



Veja também:
COMPARAÇÃO - Diferente Inteiro

COMPARAÇÃO - Menor Inteiro Var

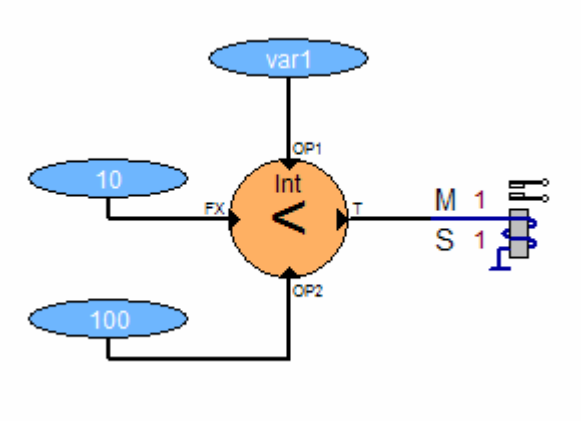
Este bloco efetua a comparação entre dois operandos inteiros (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja menor que o segundo operando. Note que um terceiro operando inteiro (FX) permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) não é acionado. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados e FX é o valor dentro do qual os dois operandos são considerados iguais. Assim, o nodo de Teste (T) é acionado se OP1 for menor que (OP2 - FX). Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, o OP1 deve ser menor que OP2 para haver o acionamento do nodo de Teste (T).

Note que OP1, OP2 e FX podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

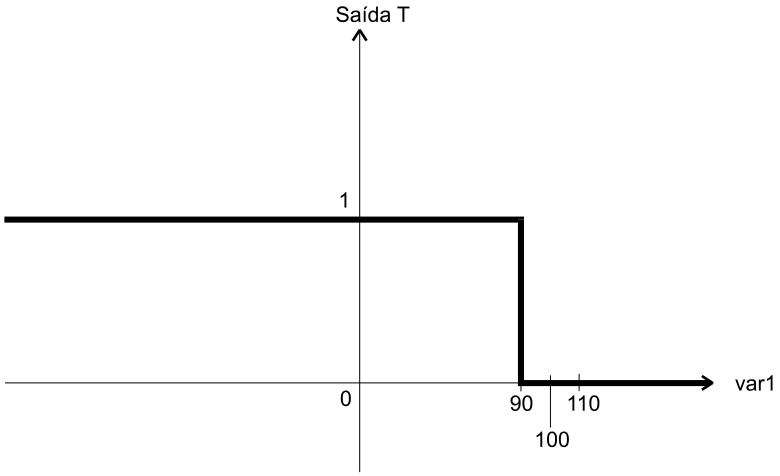
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 < OP2 fora da faixa FX especificada.



Exemplo de Programa Aplicativo: **Comparação - Menor Inteiro Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Menor inteira Var. No caso, a variável var1 é comparada com a constante de valor 100.

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 menor que 100, fora da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores menores que 90 (excluindo o valor 90). O teste será válido quando var1 for ...87, 88, 89. Caso o programa fosse editado e o valor de FX fosse trocado para zero, o bloco ligaria o nodo T caso var1 assumisse valores menores que 100 (...97, 98, 99).

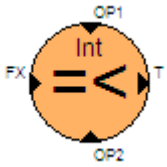


Veja também:
COMPARAÇÃO - Menor Inteiro
COMPARAÇÃO - Menor LongInt
COMPARAÇÃO - Menor LongInt Var
COMPARAÇÃO - Menor Word
COMPARAÇÃO - Menor Word Var
COMPARAÇÃO - Menor Real
COMPARAÇÃO - Menor Real Var

COMPARAÇÃO - Menor Igual Inteiro Var

Este bloco efetua a comparação entre dois operandos inteiros (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja menor ou igual ao segundo operando. Note que um terceiro operando inteiro (FX) permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) é acionado.

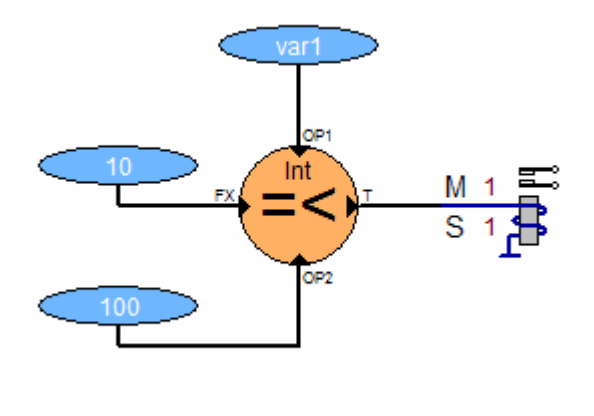
No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados e FX é o valor dentro do qual os dois operandos são considerados iguais. Assim, o nodo de Teste (T) é acionado se OP1 for menor ou igual que (OP2 + FX). Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, o operando OP1 deve ser menor ou igual que operando OP2 para haver o acionamento do nodo de Teste (T).

Note que OP1, OP2 e FX podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

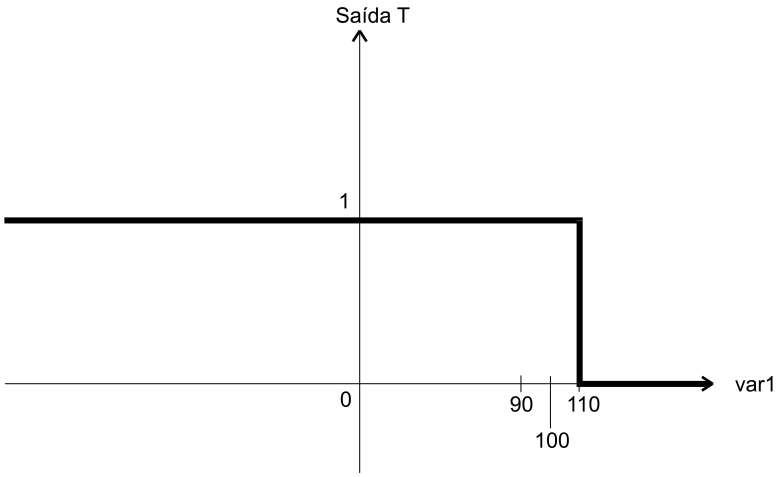
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 ≤ OP2 dentro da faixa FX especificada.



Exemplo de Programa Aplicativo: **Comparação - Menor Igual Inteiro Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Menor Igual inteira Var. No caso, a variável var1 é comparada com a constante de valor 100.

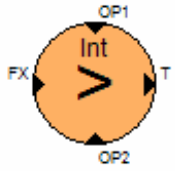
O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 menor ou igual a 100, dentro da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores menores ou iguais a 110 (incluindo o valor 110). O teste será válido quando var1 for ...108, 109, 110. Caso o programa fosse editado e o valor de faixa FX fosse trocado para zero, o bloco ligaria o nodo T caso var1 assumisse valores menores ou iguais a 100 (...97, 98, 99, 100).



Veja também:
COMPARAÇÃO - Menor Igual Inteiro

COMPARAÇÃO - Maior Inteiro Var

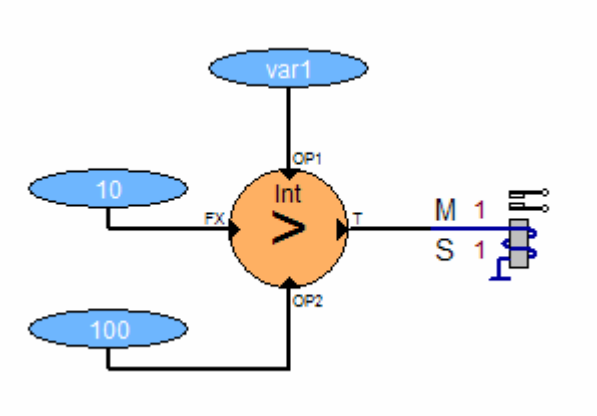
Este bloco efetua a comparação entre dois operandos inteiros (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja maior que o segundo operando. Note que um terceiro operando inteiro (FX) permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) não é acionado. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados e FX é o valor dentro do qual os dois operandos são considerados iguais. Assim, o nodo de Teste (T) é acionado se OP1 for maior que (OP2 + FX). Caso não seja especificada a faixa FX, esta é considerada igual a zero e, neste caso, o operando OP1 deve ser maior que operando OP2 para haver o acionamento do nodo de Teste (T).

Note que OP1, OP2 e FX podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

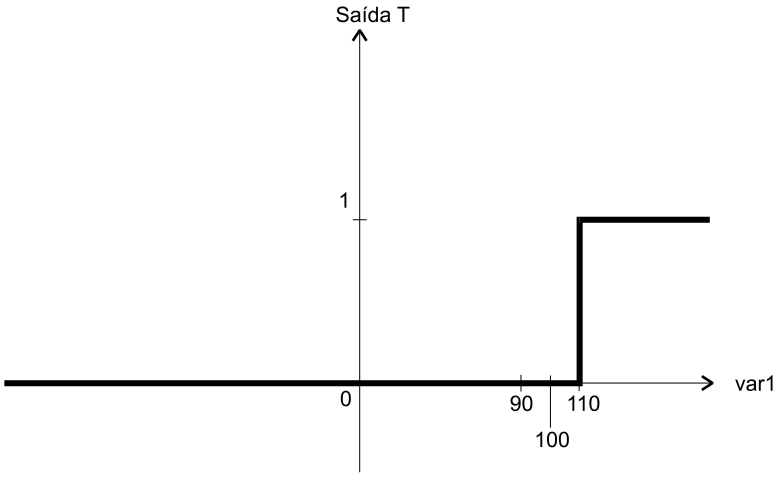
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 > OP2 fora da faixa FX especificada.



Exemplo de Programa Aplicativo: **Comparação - Maior Inteiro Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Maior inteira Var. No caso, a variável var1 é comparada com a constante de valor 100.

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 maior que 100, fora da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores maiores que 110 (excluindo o valor 110). O teste será válido quando var1 for 111, 112, 113.... Caso o programa fosse editado e o valor de faixa FX fosse trocado para zero, o bloco ligaria o nodo T caso var1 assumisse valores maiores que 100 (111, 112, 113...).

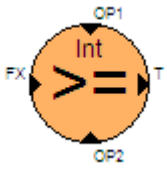


Veja também:
COMPARAÇÃO - Maior Inteiro
COMPARAÇÃO - Maior LongInt
COMPARAÇÃO - Maior LongInt Var
COMPARAÇÃO - Maior Word
COMPARAÇÃO - Maior Word Var
COMPARAÇÃO - Maior Real
COMPARAÇÃO - Maior Real Var

COMPARAÇÃO - Maior Igual Inteiro Var

Este bloco efetua a comparação entre dois operandos inteiros (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja maior ou igual ao segundo operando. Note que um terceiro operando inteiro (FX) permite especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais e, portanto, o nodo de Teste (T) é acionado. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas

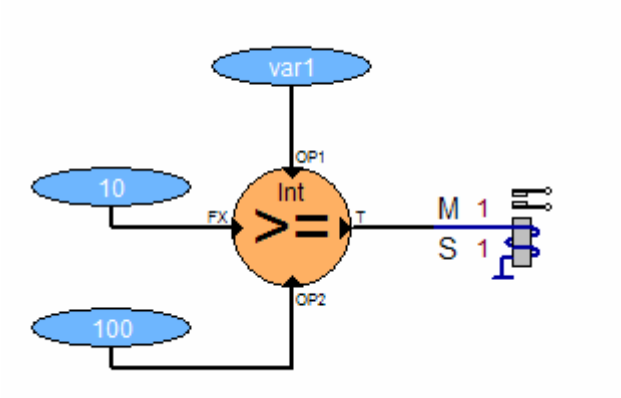
conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados e FX é o valor dentro do qual os dois operandos são considerados iguais. Assim, o nodo de Teste (T) é acionado se OP1 for maior ou igual que (OP2 - FX). Caso não seja especificada a faixa, esta é considerada igual a zero e, neste caso, o operando OP1 deve ser maior ou igual que operando OP2 para haver o acionamento do nodo de Teste (T).

Note que OP1, OP2 e FX podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

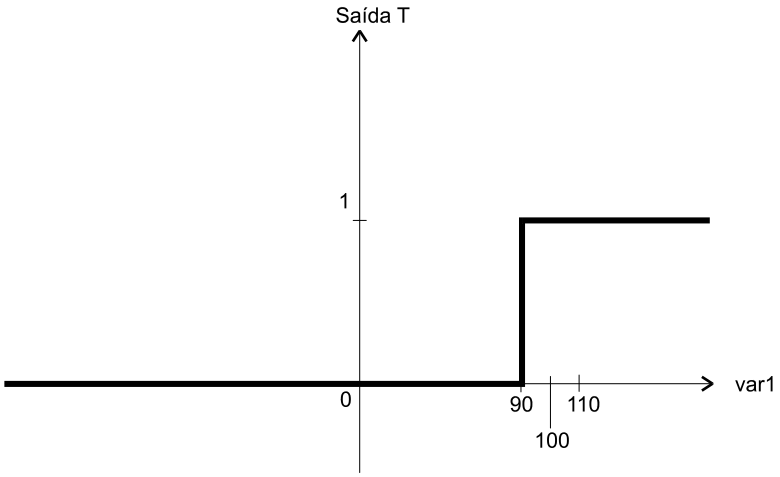
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 ≥ OP2 dentro da faixa FX especificada.



Exemplo de Programa Aplicativo: **Comparação - Maior Igual Inteiro Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Maior Igual inteira Var. No caso, a variável var1 é comparada com a constante de valor 100.

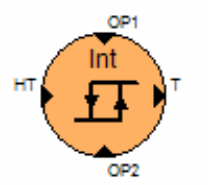
O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 maior ou igual a 100, dentro da faixa especificada (no caso, 10 unidades). Ou seja, o nodo T será acionado quando a variável var1 assumir valores maiores ou iguais a 90 (incluindo o valor 90). O teste será válido quando var1 for 90, 91, 92.... Caso o programa fosse editado e o valor de faixa FX fosse trocado para zero, o bloco ligaria o nodo T caso var1 assumisse valores maiores ou iguais a 100 (100, 101, 102...).



Veja também:
COMPARAÇÃO - Maior Igual Inteiro

COMPARAÇÃO - Histerese Inteiro Var

Este bloco efetua a comparação com histerese entre dois operandos inteiros (OP1 e OP2), ligando o nodo de Teste (T) quando OP1 for maior ou igual a (OP2 + HT), e desligando o nodo de Teste (T) quando OP1 for menor ou igual a (OP2 - HT). Note que um terceiro operando inteiro (HT) permite especificar a histerese. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



Note que OP1, OP2 e HT podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

Este bloco é muito útil, por exemplo, para acionar uma determinada saída conforme dois valores de comparação. Em um controle de compressor de ar-condicionado não devemos acionar e desacionar o compressor conforme a temperatura suba de determinado ponto, pois neste caso ficaríamos constantemente acionando e desacionando o compressor. Se a temperatura desejada é 21,0°C e colocássemos um teste simples o compressor acionaria em 21,0°C e desacionaria em 20,9°C (considerando uma resolução de 0,1°C para o sensor de temperatura). Como a diferença de temperatura é muito pequena, logo estaríamos novamente em 21,0°C e ele seria novamente acionado e assim por diante, diminuindo a vida útil do equipamento.

Então, é mais conveniente fixar duas temperaturas distintas, uma para ligar o compressor e outra para desligá-lo. Digamos que em 22,0°C ele será acionado e em 20,0°C desacionado. Podemos fazer isso com um simples bloco de comparação com histerese. Para isso bastaria especificar o valor correspondente a 21,0°C no operando OP2, enquanto o operando OP1 aponta para o sensor de temperatura. Já na histerese HT especifica-se um valor correspondente a 1,0°C (histerese desejada de ± 1,0°C). Se o sensor de temperatura estiver ligado a uma das entradas

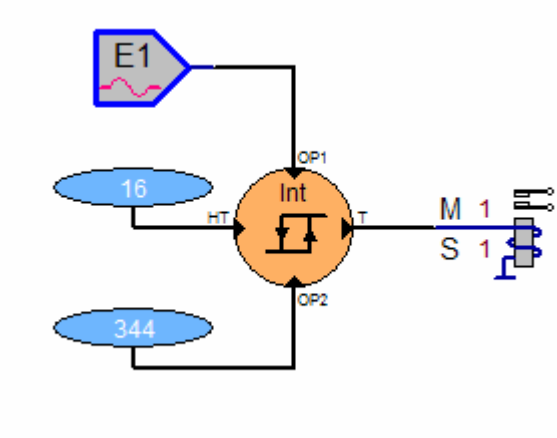
analógicas, programada para escala de 0-2,5V, e fornecer 10mV/°C (é o caso, por exemplo, do sensor LM35, da National Semiconductors), teremos:

2,5V / 4095 = 610,5µV de resolução.

Logo, cada 1°C corresponde a 10mV / 610,5µV = 16,38 passos.

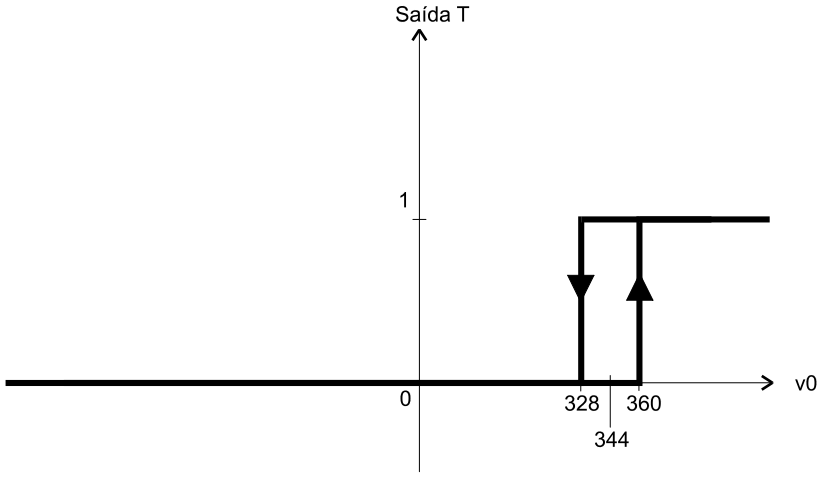
Assim, em 21,0°C teremos 210mV na entrada analógica, o que corresponde a um valor de 210mV / 610,35µV = 343,98 passos @ 344 passos.

Então, devemos comparar a temperatura com o valor 344, e inserir uma histerese de 16 no bloco. Com isso, a saída será acionada quando o valor da entrada analógica corresponder a 344 + 16 = 360 (o que corresponde a 360 * 610,5µV = 21,978mV @ 22,0mV, ou seja, 22,0°C). Já a saída será desligada quando o valor da entrada analógica corresponder a 344 - 16 = 328 (o que corresponde a 328 * 610,5µV = 20,024mV @ 20,0mV, ou seja, 20,0°C). A seguir temos este exemplo implementado no programa PG.



Exemplo de Programa Aplicativo: **Comparação - Histerese Inteiro Var.dxg**

Note que foi utilizada a entrada analógica E1 (que corresponde a v0). O nodo de saída Teste (T) é acionado sempre que o bloco detecta que a entrada analógica E1 assumiu valor maior ou igual a 360, e desliga este nodo sempre que E1 assumir valores menores ou iguais a 328.

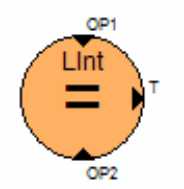


Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável v0 (que corresponde a entrada analógica E1), para o bloco de comparação inteira Histerese Var. O valor de comparação é 344 e a histerese é 16.

Veja também:
COMPARAÇÃO - Histerese Inteiro

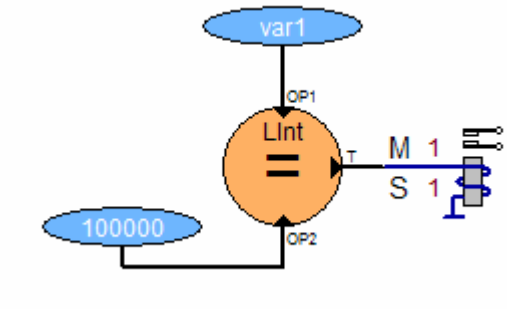
COMPARAÇÃO - Igual LongInt Var

Este bloco efetua a comparação entre dois operandos longint (OP1 e OP2), ligando o nodo de Teste (T) caso os dois operandos sejam iguais. Note que este bloco, ao contrário do bloco de comparação Igual Inteiro, não possui um terceiro operando para especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



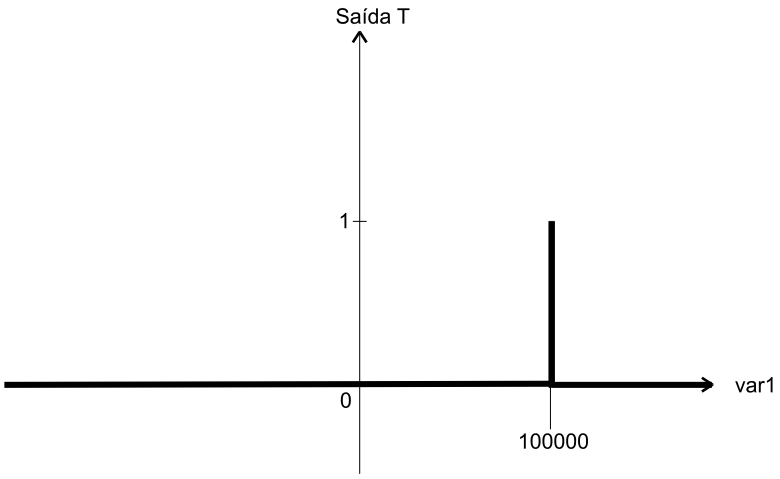
OP1 e OP2 são os operandos a serem comparados. Note que OP1 e OP2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 = OP2.



Exemplo de Programa Aplicativo: **Comparação - Igual LongInt Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Igual longint Var. No caso, a variável var1 é comparada com a constante 100.000.

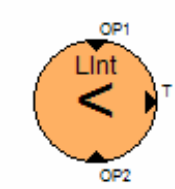


Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação longint Igual Var. O nodo T só é ligado quando o valor de var1 é 100.000.

Veja também:
COMPARAÇÃO - Igual LongInt
COMPARAÇÃO - Igual Inteiro
COMPARAÇÃO - Igual Inteiro Var
COMPARAÇÃO - Igual Word
COMPARAÇÃO - Igual Word Var
COMPARAÇÃO - Igual Real
COMPARAÇÃO - Igual Real Var

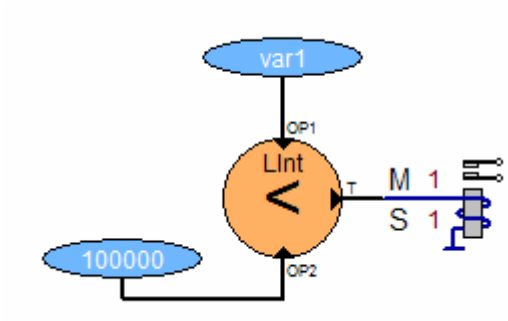
COMPARAÇÃO - Menor LongInt Var

Este bloco efetua a comparação entre dois operandos longint (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja menor que o segundo operando. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados. Note que OP1 e OP2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

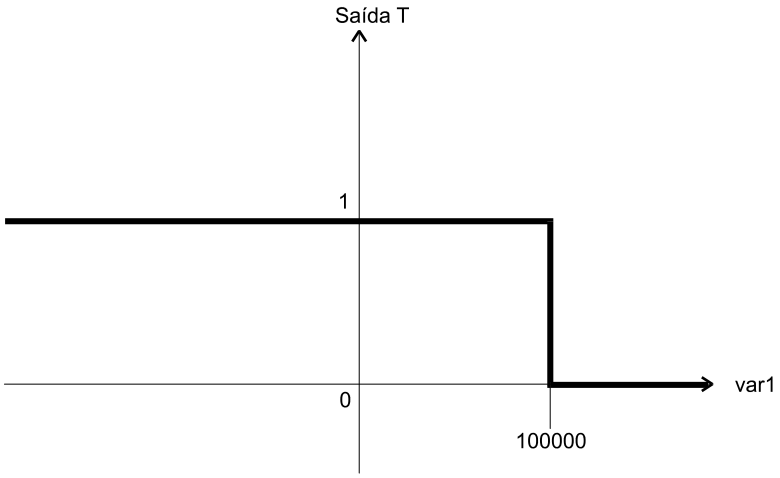
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 < OP2.



Exemplo de Programa Aplicativo: **Comparação - Menor LongInt Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Menor longint Var. No caso, a variável var1 é comparada com a constante 100.000.

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 menor que 100.000. Ou seja, o nodo T será acionado quando a variável var1 assumir valores ...99.997, 99.998, 99.999.

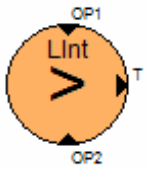


Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação longint Menor Var. O nodo T é acionado sempre que var1 assume valor menor que 100.000.

Veja também:
COMPARAÇÃO - Menor LongInt
COMPARAÇÃO - Menor Inteiro
COMPARAÇÃO - Menor Inteiro Var
COMPARAÇÃO - Menor Word
COMPARAÇÃO - Menor Word Var
COMPARAÇÃO - Menor Real
COMPARAÇÃO - Menor Real Var

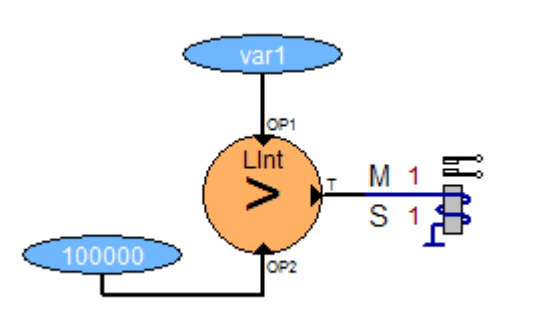
COMPARAÇÃO - Maior LongInt Var

Este bloco efetua a comparação entre dois operandos longint (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja maior que o segundo operando. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados. Note que OP1 e OP2 podem ser variáveis longint, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

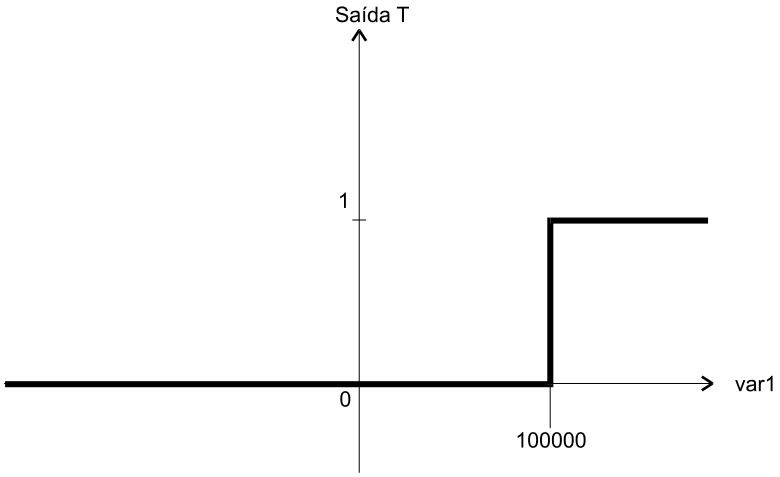
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 > OP2.



Exemplo de Programa Aplicativo: **Comparação - Maior LongInt Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Maior longint Var. No caso, a variável var1 é comparada com a constante 100.000.

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 maior que 100.000. Ou seja, o nodo T será acionado quando a variável var1 assumir valores de 100.001, 100.002, 100.003,....



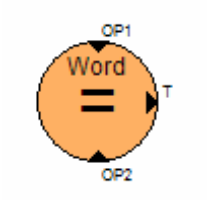
Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação longint Maior Var. O nodo T é acionado sempre que var1 for maior que 100.000.

Veja também:
COMPARAÇÃO - Maior LongInt
COMPARAÇÃO - Maior Inteiro
COMPARAÇÃO - Maior Inteiro Var
COMPARAÇÃO - Maior Word
COMPARAÇÃO - Maior Word Var
COMPARAÇÃO - Maior Real
COMPARAÇÃO - Maior Real Var

COMPARAÇÃO - Igual Word Var

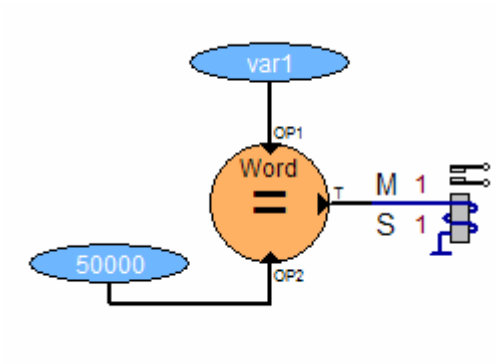
Este bloco efetua a comparação entre dois operandos word (OP1 e OP2), ligando o nodo de Teste (T) caso os dois operandos sejam iguais. Note que este bloco, ao contrário do bloco de comparação Igual Inteiro, não possui um terceiro operando para especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais. No caso deste tipo

de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



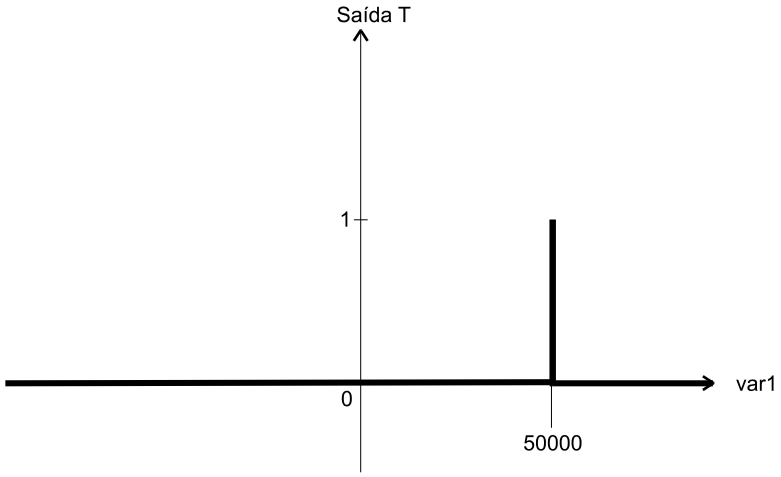
OP1 e OP2 são os operandos a serem comparados. Note que OP1 e OP2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 = OP2.



Exemplo de Programa Aplicativo: **Comparação - Igual Word Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Igual word Var. No caso, a variável var1 é comparada com a constante 50.000.

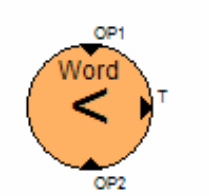


Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação word Igual Var. O nodo T só é ligado quando o valor de var1 é 50.000. Note que o limite para números word é 65535.

Veja também:
COMPARAÇÃO - Igual Word
COMPARAÇÃO - Igual Inteiro
COMPARAÇÃO - Igual Inteiro Var

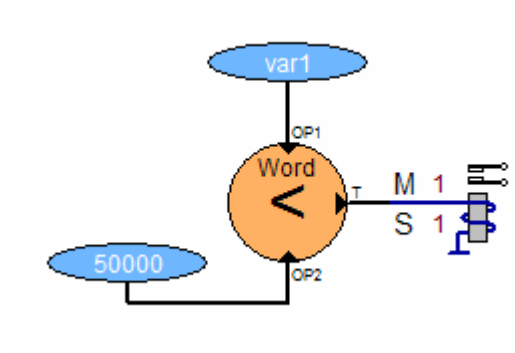
COMPARAÇÃO - Menor Word Var

Este bloco efetua a comparação entre dois operandos word (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja menor que o segundo operando. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados. Note que OP1 e OP2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

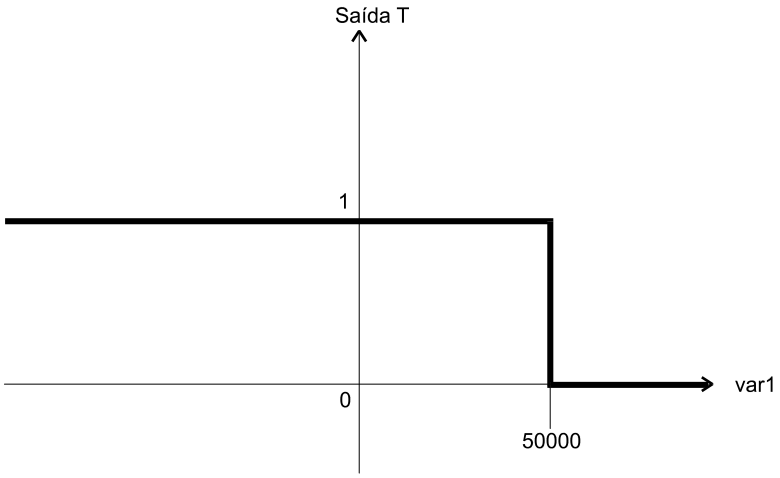
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 < OP2.



Exemplo de Programa Aplicativo: **Comparação - Menor Word Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Menor word Var. No caso, a variável var1 é comparada com a constante 50.000.

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 menor que 50.000. Ou seja, o nodo T será acionado quando a variável var1 assumir valores ...49.997, 49.998, 49.999.

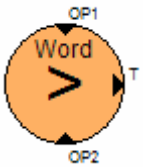


Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação word Menor Var. O nodo T é acionado sempre que var1 assume valor menor que 50.000. Note que o limite de representação de variáveis word é 65.535.

Veja também:
COMPARAÇÃO - Menor Word
COMPARAÇÃO - Menor Inteiro
COMPARAÇÃO - Menor Inteiro Var
COMPARAÇÃO - Menor LongInt
COMPARAÇÃO - Menor LongInt Var
COMPARAÇÃO - Menor Real
COMPARAÇÃO - Menor Real Var

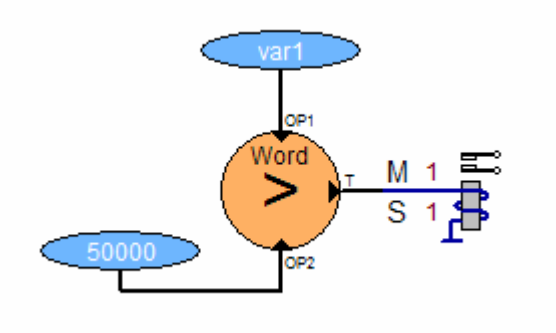
COMPARAÇÃO - Maior Word Var

Este bloco efetua a comparação entre dois operandos word (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja maior que o segundo operando. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados. Note que OP1 e OP2 podem ser variáveis word, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

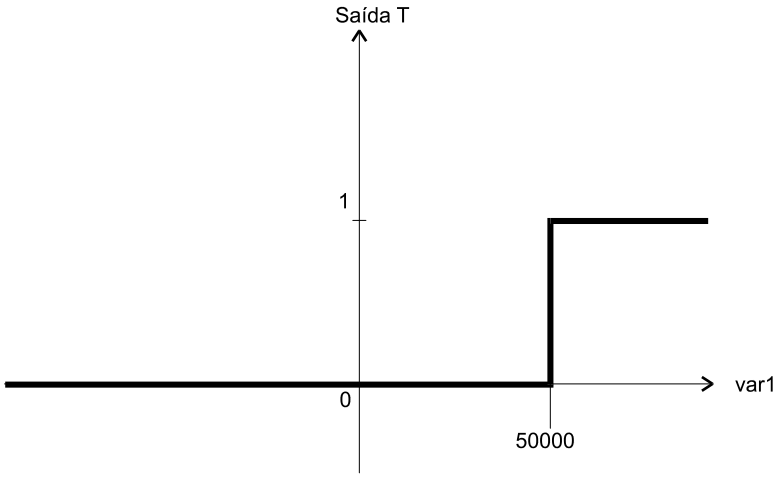
O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 > OP2.



Exemplo de Programa Aplicativo: **Comparação - Maior Word Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Maior word Var. No caso, a variável var1 é comparada com a constante 50.000.

O nodo de saída Teste (T) é acionado sempre que o bloco considera var1 maior que 50.000. Ou seja, o nodo T será acionado quando a variável var1 assumir valores de 50.001, 50.002, 50.003,....

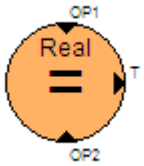


Acima temos uma representação gráfica do nodo de Teste (T) do bloco, conforme o valor da variável var1, para o bloco de comparação word Maior Var. O nodo T é acionado sempre que var1 for maior que 50.000. Note que o limite de representação para números word é de 65.535.

Veja também:
COMPARAÇÃO - Maior Word
COMPARAÇÃO - Maior Inteiro
COMPARAÇÃO - Maior Inteiro Var
COMPARAÇÃO - Maior LongInt
COMPARAÇÃO - Maior LongInt Var
COMPARAÇÃO - Maior Real
COMPARAÇÃO - Maior Real Var

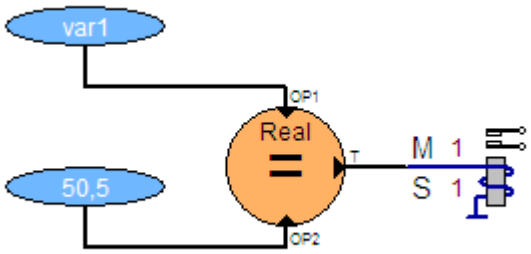
COMPARAÇÃO - Igual Real Var

Este bloco efetua a comparação entre dois operandos reais (OP1 e OP2), ligando o nodo de Teste (T) caso os dois operandos sejam iguais. Note que este bloco, ao contrário do bloco de comparação Igual Inteiro, não possui um terceiro operando para especificar uma faixa dentro da qual os dois operandos a serem comparados podem ser considerados iguais. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados. Note que OP1 e OP2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 = OP2.



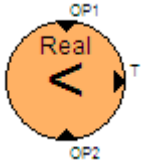
Exemplo de Programa Aplicativo: **Comparação - Igual Real Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Igual Real Var. No caso, a variável var1 é comparada com a constante 50,5.

Veja também:
COMPARAÇÃO - Igual Real
COMPARAÇÃO - Igual Inteiro
COMPARAÇÃO - Igual Inteiro Var
COMPARAÇÃO - Igual LongInt
COMPARAÇÃO - Igual LongInt Var
COMPARAÇÃO - Igual Word
COMPARAÇÃO - Igual Word Var

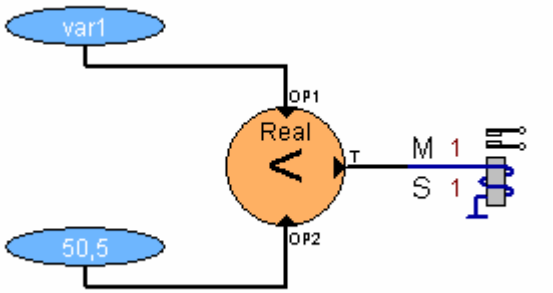
COMPARAÇÃO - Menor Real Var

Este bloco efetua a comparação entre dois operandos reais (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja menor que o segundo operando. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados. Note que OP1 e OP2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 < OP2.



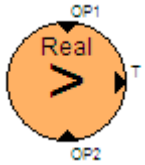
Exemplo de Programa Aplicativo: **Comparação - Menor Real Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Menor Real Var. No caso, a variável var1 é comparada com a constante 50,5.

Veja também:
COMPARAÇÃO - Menor Real
COMPARAÇÃO - Menor Inteiro
COMPARAÇÃO - Menor Inteiro Var
COMPARAÇÃO - Menor LongInt
COMPARAÇÃO - Menor LongInt Var
COMPARAÇÃO - Menor Word
COMPARAÇÃO - Menor Word Var

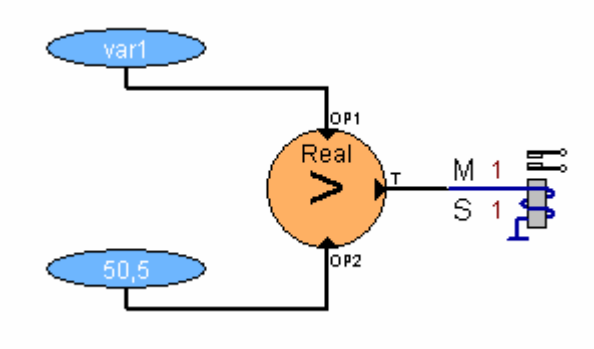
COMPARAÇÃO - Maior Real Var

Este bloco efetua a comparação entre dois operandos reais (OP1 e OP2), ligando o nodo de Teste (T) caso o primeiro operando seja maior que o segundo operando. No caso deste tipo de bloco (circular em vez de retangular) as variáveis são especificadas nas conexões, ou seja, os fios de conexão transportam as variáveis (em vez de serem especificadas literalmente via edição do bloco, como no caso de blocos retangulares). Este tipo de bloco não possui nodo de entrada, e está constantemente habilitado.



OP1 e OP2 são os operandos a serem comparados. Note que OP1 e OP2 podem ser variáveis reais, constantes (valores numéricos), ou ainda uma referência a uma posição de memória.

O nodo de Teste (T) é acionado caso o teste resulte verdadeiro, ou seja, neste caso OP1 > OP2.



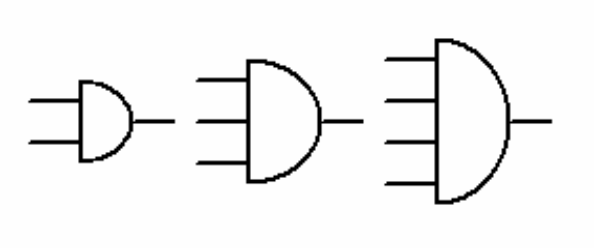
Exemplo de Programa Aplicativo: **Comparação - Maior Real Var.dxg**

Acima temos um pequeno exemplo de uso do bloco de comparação Maior Real Var. No caso, a variável var1 é comparada com a constante 50,5.

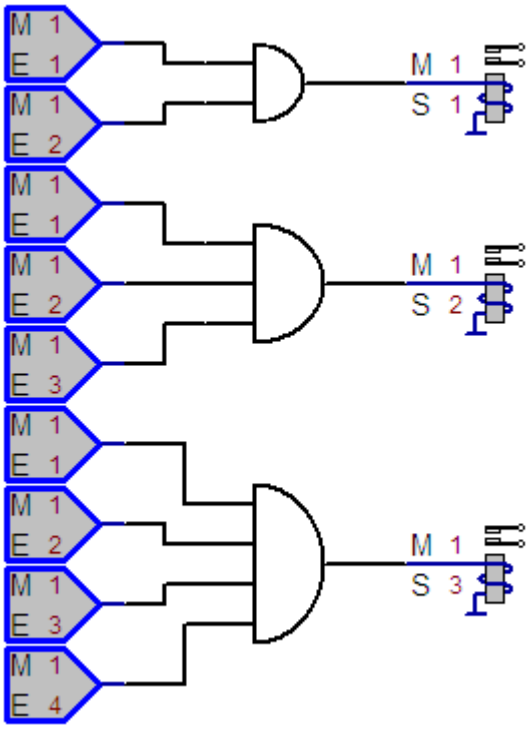
Veja também:
COMPARAÇÃO - Maior Real
COMPARAÇÃO - Maior Inteiro
COMPARAÇÃO - Maior Inteiro Var
COMPARAÇÃO - Maior LongInt
COMPARAÇÃO - Maior LongInt Var
COMPARAÇÃO - Maior Word
COMPARAÇÃO - Maior Word Var

COMBINACIONAL - Porta AND

Estes blocos efetuam a operação booleana AND (E) entre dois, três ou quatro operandos binários. Note que as conexões são nodos, ou seja, variáveis binárias (apenas dois estados: ligado ou desligado). O nodo de saída é acionado apenas quando todos os nodos de entrada estiverem ligados.



No exemplo abaixo, apenas quando as três entradas E1, E2 e E3 do módulo de Expansão M1 (µDX210) são acionadas simultaneamente, a saída S1 do mesmo módulo é ligada. Ou seja, para a saída ligar é necessário que todas as entradas estejam ligadas.



Exemplo de Programa Aplicativo: **Combinacional - Porta AND.dxg**

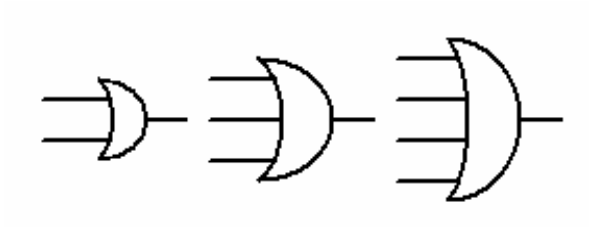
A tabela de estados das entradas e respectivo valor de saída é dada a seguir (note que 0 representa nodo desligado e 1 representa nodo ligado):

Porta AND

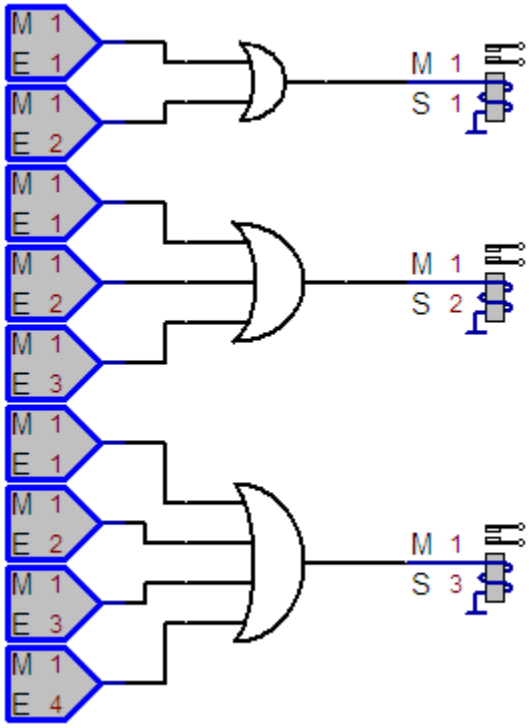
Entrada 4	Entrada 3	Entrada 2	Entrada 1	AND	AND 3	AND 4
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	1	0	0
0	1	0	0		0	0
0	1	0	1		0	0
0	1	1	0		0	0
0	1	1	1		1	0
1	0	0	0			0
1	0	0	1			0
1	0	1	0			0
1	0	1	1			0
1	1	0	0			0
1	1	0	1			0
1	1	1	0			0
1	1	1	1			1

COMBINACIONAL - Porta OR

Estes blocos efetuam a operação booleana OR (OU) entre dois, três ou quatro operandos binários. Note que as conexões são nodos, ou seja, variáveis binárias (apenas dois estados: ligado ou desligado). O nodo de saída é acionado sempre que pelo menos um dos nodos de entrada esteja ligado.



No exemplo abaixo, quando uma das entradas E1, E2 e E3 do módulo de Expansão M1 (µDX210) é acionada, a saída S1 do mesmo módulo é ligada. Ou, se tivermos uma ou mais entradas ligadas a saída será ligada.



Exemplo de Programa Aplicativo: **Combinacional - Porta OR.dxg**

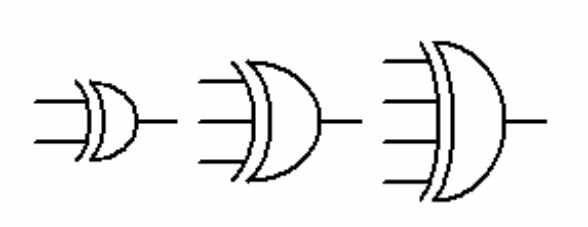
A tabela de estados das entradas e respectivo valor de saída é dada a seguir (note que 0 representa nodo desligado e 1 representa nodo ligado):

Porta OR

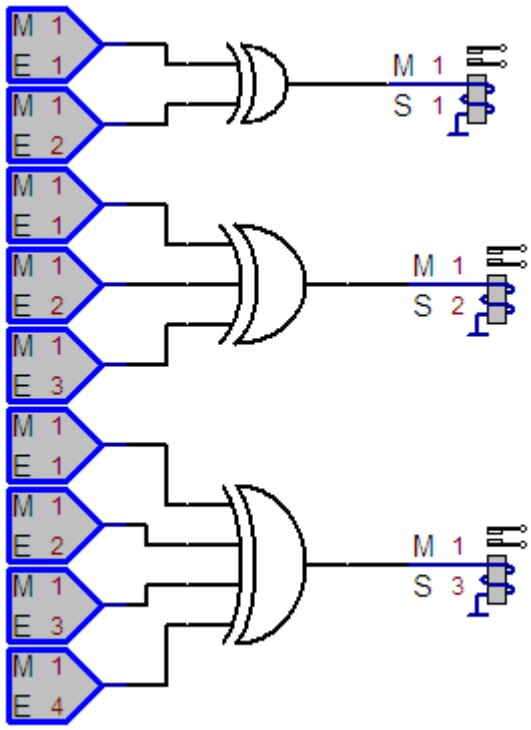
Entrada 4	Entrada 3	Entrada 2	Entrada 1	OR	OR 3	OR 4
0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	0	1	0	1	1	1
0	0	1	1	1	1	1
0	1	0	0		1	1
0	1	0	1		1	1
0	1	1	0		1	1
0	1	1	1		1	1
1	0	0	0			1
1	0	0	1			1
1	0	1	0			1
1	0	1	1			1
1	1	0	0			1
1	1	0	1			1
1	1	1	0			1
1	1	1	1			1

COMBINACIONAL - Porta XOR

Estes blocos efetuam a operação booleana XOR (OU EXCLUSIVO) entre dois, três ou quatro operandos binários. Note que as conexões são nodos, ou seja, variáveis binárias (apenas dois estados: ligado ou desligado). O nodo de saída é acionado apenas quando um número ímpar de nodos de entrada estiverem ligados.



No exemplo abaixo, apenas quando uma ou as três entradas E1, E2 e E3 do módulo de Expansão M1 (µDX210) são acionadas simultaneamente, a saída S1 do mesmo módulo é ligada. Ou seja, para a saída ligar é necessário que um número ímpar de entradas estejam ligadas.



Exemplo de Programa Aplicativo: **Combinacional - Porta XOR.dxg**

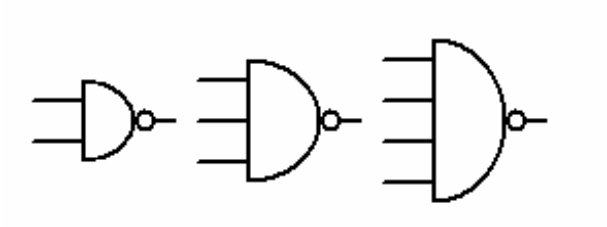
A tabela de estados das entradas e respectivo valor de saída é dada a seguir (note que 0 representa nodo desligado e 1 representa nodo ligado):

Porta XOR

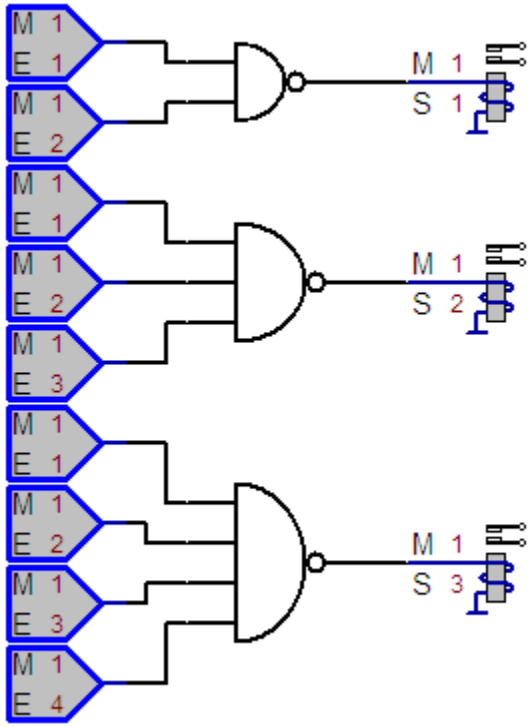
Entrada 4	Entrada 3	Entrada 2	Entrada 1	XOR	XOR 3	XOR 4
0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	0	1	0	1	1	1
0	0	1	1	0	0	0
0	1	0	0		1	1
0	1	0	1		0	0
0	1	1	0		0	0
0	1	1	1		1	1
1	0	0	0			1
1	0	0	1			0
1	0	1	0			0
1	0	1	1			1
1	1	0	0			0
1	1	0	1			1
1	1	1	0			1
1	1	1	1			0

COMBINACIONAL - Porta NAND

Estes blocos efetuam a operação booleana NAND (E negado) entre dois, três ou quatro operandos binários. Note que as conexões são nodos, ou seja, variáveis binárias (apenas dois estados: ligado ou desligado). O nodo de saída é desacionado apenas quando todos os nodos de entrada estiverem ligados.



No exemplo abaixo, apenas quando as três entradas E1, E2 e E3 do módulo de Expansão M1 (µDX210) são acionadas simultaneamente, a saída S1 do mesmo módulo é desligada. Ou seja, para a saída desligar é necessário que todas as entradas estejam ligadas.



Exemplo de Programa Aplicativo: **Combinacional - Porta NAND.dxg**

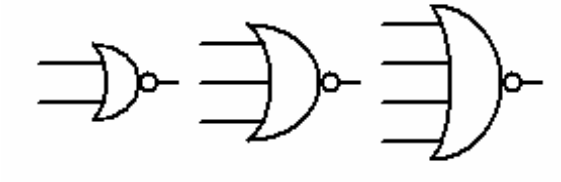
A tabela de estados das entradas e respectivo valor de saída é dada a seguir (note que 0 representa nodo desligado e 1 representa nodo ligado):

Porta NAND

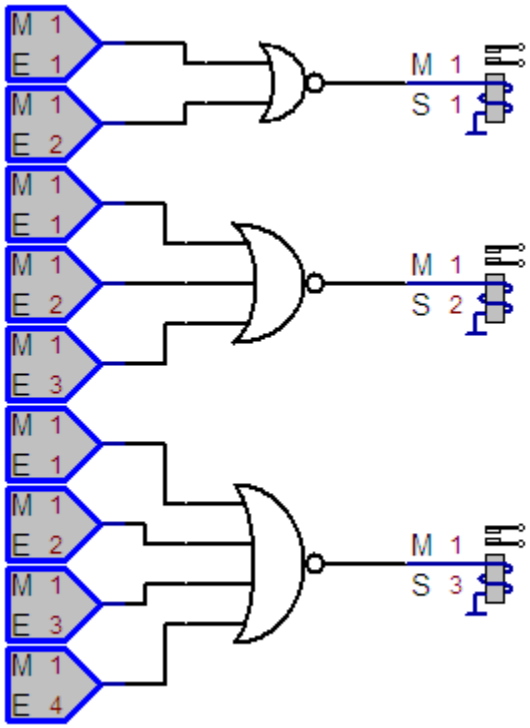
Entrada 4	Entrada 3	Entrada 2	Entrada 1	NAND	NAND 3	NAND 4
0	0	0	0	1	1	1
0	0	0	1	1	1	1
0	0	1	0	1	1	1
0	0	1	1	0	1	1
0	1	0	0		1	1
0	1	0	1		1	1
0	1	1	0		1	1
0	1	1	1		0	1
1	0	0	0			1
1	0	0	1			1
1	0	1	0			1
1	0	1	1			1
1	1	0	0			1
1	1	0	1			1
1	1	1	0			1
1	1	1	1			0

COMBINACIONAL - Porta NOR

Estes blocos efetuam a operação booleana NOR (OU negado) entre dois, três ou quatro operandos binários. Note que as conexões são nodos, ou seja, variáveis binárias (apenas dois estados: ligado ou desligado). O nodo de saída é desacionado sempre que pelo menos um dos nodos de entrada esteja ligado.



No exemplo abaixo, quando uma das entradas E1, E2 e E3 do módulo de Expansão M1 (µDX210) é acionada, a saída S1 do mesmo módulo é desligada. Ou, se tivermos uma ou mais entradas ligadas a saída será desligada.



Exemplo de Programa Aplicativo: **Combinacional - Porta NOR.dxg**

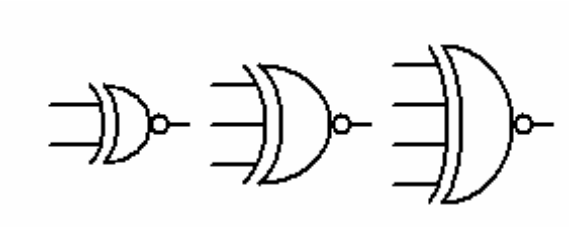
A tabela de estados das entradas e respectivo valor de saída é dada a seguir (note que 0 representa nodo desligado e 1 representa nodo ligado):

Porta NOR

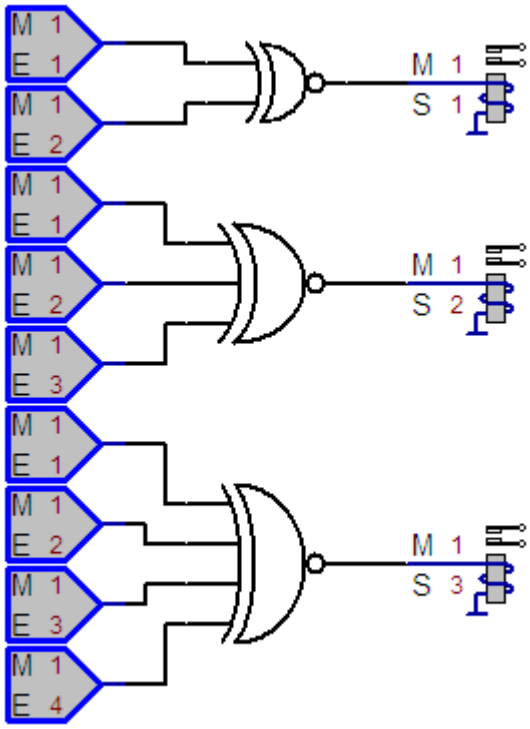
Entrada 4	Entrada 3	Entrada 2	Entrada 1	NOR	NOR 3	NOR 4
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0		0	0
0	1	0	1		0	0
0	1	1	0		0	0
0	1	1	1		0	0
1	0	0	0			0
1	0	0	1			0
1	0	1	0			0
1	0	1	1			0
1	1	0	0			0
1	1	0	1			0
1	1	1	0			0
1	1	1	1			0

COMBINACIONAL - Porta NXOR

Estes blocos efetuam a operação booleana NXOR (OU EXCLUSIVO negado) entre dois, três ou quatro operandos binários. Note que as conexões são nodos, ou seja, variáveis binárias (apenas dois estados: ligado ou desligado). O nodo de saída é acionado apenas quando um número par de nodos de entrada estiverem ligados, ou nenhuma entrada estiver acionada.



No exemplo abaixo, apenas quando nenhuma ou duas das três entradas E1, E2 e E3 do módulo de Expansão M1 (µDX210) são acionadas simultaneamente, a saída S1 do mesmo módulo é ligada. Ou seja, para a saída ligar é necessário que um número par de entradas estejam ligadas, ou nenhuma entrada esteja ligada.



Exemplo de Programa Aplicativo: **Combinacional - Porta NXOR.dxg**

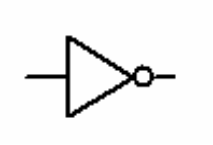
A tabela de estados das entradas e respectivo valor de saída é dada a seguir (note que 0 representa nodo desligado e 1 representa nodo ligado):

Porta NXOR

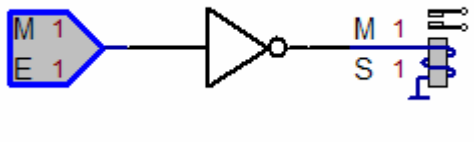
Entrada 4	Entrada 3	Entrada 2	Entrada 1	NXOR	NXOR 3	NXOR 4
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	1	1	1
0	1	0	0		0	0
0	1	0	1		1	1
0	1	1	0		1	1
0	1	1	1		0	0
1	0	0	0			0
1	0	0	1			1
1	0	1	0			1
1	0	1	1			0
1	1	0	0			1
1	1	0	1			0
1	1	1	0			0
1	1	1	1			1

COMBINACIONAL - Porta NOT

Este bloco efetua a operação booleana NOT (NÃO ou negação) de um operando binário. Note que as conexões são nodos, ou seja, variáveis binárias (apenas dois estados: ligado ou desligado). O nodo de saída é acionado sempre que o nodo de entrada esteja desligado, e vice-versa.



No exemplo abaixo, quando a entrada E1 do módulo de Expansão M1 (µDX210) é acionada, a saída S1 do mesmo módulo é desligada. Já quando E1 é desligada a saída S1 é ligada.



Exemplo de Programa Aplicativo: **Combinacional - Porta NOT.dxg**

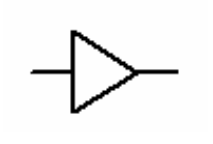
A tabela de estados da entrada e respectivo valor de saída é dada a seguir (note que 0 representa nodo desligado e 1 representa nodo ligado):

Porta NOT

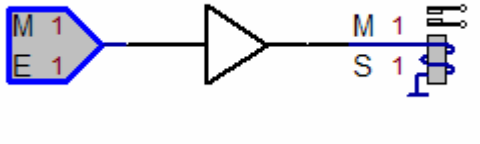
Entrada	NOT
0	1
1	0

COMBINACIONAL - Porta Buffer

Este bloco copia o estado de um operando de entrada binário para outro operando binário de saída. Note que as conexões são nodos, ou seja, variáveis binárias (apenas dois estados: ligado ou desligado). O nodo de saída é acionado sempre que o nodo de entrada esteja ligado, e vice-versa.

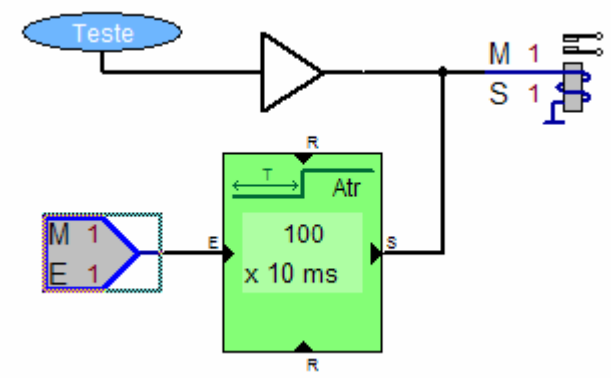


No exemplo abaixo, quando a entrada E1 do módulo de Expansão M1 (µDX210) é acionada, a saída S1 do mesmo módulo é ligada. Já quando E1 é desligada a saída S1 é desligada.



Exemplo de Programa Aplicativo: **Combinacional - Porta Buffer.dxg**

Este bloco Buffer (ou porta não-inversora) é usada para isolar nodos. Por exemplo, no exemplo abaixo é usada uma porta não-inversora para isolar o nodo Teste da saída S1, que é acionada tanto pelo nodo Teste quanto pelo bloco de temporização de atraso:



Exemplo de Programa Aplicativo: **Combinacional - Porta Buffer 2.dxg**

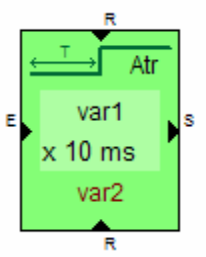
A tabela de estados da entrada e respectivo valor de saída é dada a seguir (note que 0 representa nodo desligado e 1 representa nodo ligado):

Porta Buffer

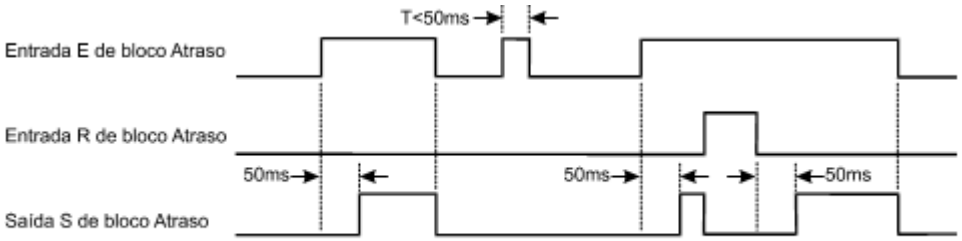
Entrada	Buffer
0	0
1	1

TEMPORIZAÇÃO - Atraso

Este bloco gera um atraso entre a energização do nodo de Entrada (E) e a energização do nodo de Saída (S). Caso o nodo de entrada retorne a zero antes do tempo de atraso programado o nodo de saída não é acionado. Quando o nodo de entrada é desenergizado a saída imediatamente também é desligada. Os nodos de Reset (R) estão internamente interligados e permitem desativar o bloco imediatamente. Note que qualquer sinal ligado a um dos nodos R está automaticamente ligado ao outro nodo R, já que se trata de dois nodos ligados internamente pelo bloco.



Abaixo temos uma representação gráfica do comportamento do bloco de Atraso:



BLOCO ATRASO PROGRAMADO PARA 50ms

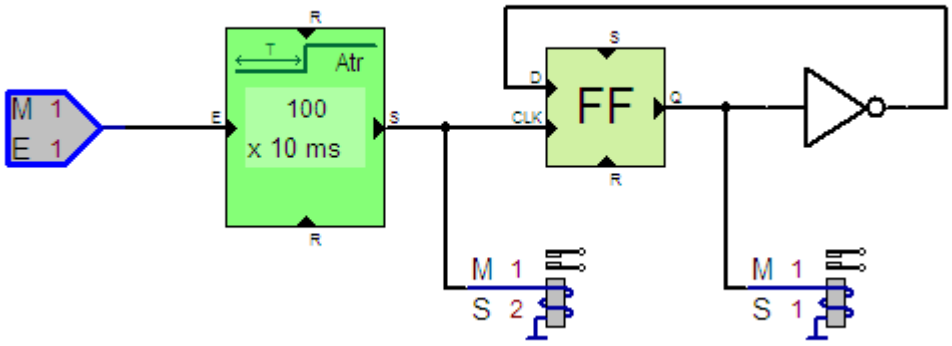
Tempo é o operando que determina o tempo do bloco de Temporização. Já a Var.Auxiliar é a variável usada para contar este tempo. Note que Tempo pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Já Var.Auxiliar não pode ser uma constante, já que deverá ser decrementada durante a operação do bloco. Caso não seja especificada a variável auxiliar no bloco de temporização o compilador irá alocar um espaço vago na memória RAM disponível.

A Escala de tempo determina qual a unidade de temporização a que se refere o operando Tempo. Assim, é possível escolher entre 1ms, 10ms, 100ms, ou ainda 1s (para editar o bloco selecionando a Escala de tempo e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Como operandos word podem assumir valores entre 0 e 65535 resulta que podemos programar o bloco dentro dos seguintes intervalos:

Escala de Tempo	Tempo Mínimo	Tempo Máximo
x 1 ms	1 ms	65,535 s = 1 min 5 s
x 10 ms	10 ms	655,35 s = 10 min 55 s
x 100 ms	100 ms	6553,5 s = 1 h 49 min 13 s
x 1 s	1 s	65535 s = 18 h 12 min 15 s

Já o nodo de Reset (R) permite zerar a contagem de tempo do bloco, e ele passa a temporizar novamente tão logo o nodo de Reset seja desativado (desde que, evidentemente, o nodo de Entrada E esteja ativado).

***ATENÇÃO**: No caso de blocos de temporização utilizando escala de tempo de 1 ms é necessário que o tempo de ciclo do programa aplicativo seja menor ou igual a 1 ms, ou o bloco de temporização sofrerá um atraso na temporização, devido ao tempo de ciclo do programa ser superior à resolução da temporização. Por exemplo, um bloco de atraso de 10ms (com escala de 1ms e variável de tempo igual a 10), se utilizado em um programa com tempo de ciclo de 2 ms, pode apresentar uma temporização de até 13 ms (2 ms adicionais devido ao tempo de ciclo do programa aplicativo e 1 ms adicional devido a imprecisão("jitter") dos blocos de temporização).*



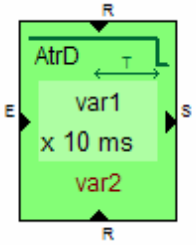
Exemplo de Programa Aplicativo: **Temporização - Atraso.dxg**

Acima temos um pequeno exemplo de uso do bloco de temporização Atraso. No caso, este bloco está sendo usado para filtrar o sinal da entrada E1 do módulo de Expansão M1 (µDX210). Ele só permite que pulsos com duração superior a 100ms passem para a saída S. Com isso, evita-se que sinais espúrios venham a trocar o estado da saída S1 do mesmo módulo de Expansão (esta saída troca de estado a cada borda de subida da saída S do bloco de Atraso graças ao flip-flop tipo D realimentado com a saída invertida). Ou seja, é necessário que a entrada E1 fique energizada por um mínimo de 100ms para ocasionar a troca de estado na saída S1. Foi incluída uma saída S2 entre o bloco de atraso e o flip-flop para observação do nodo de saída do bloco Atraso.

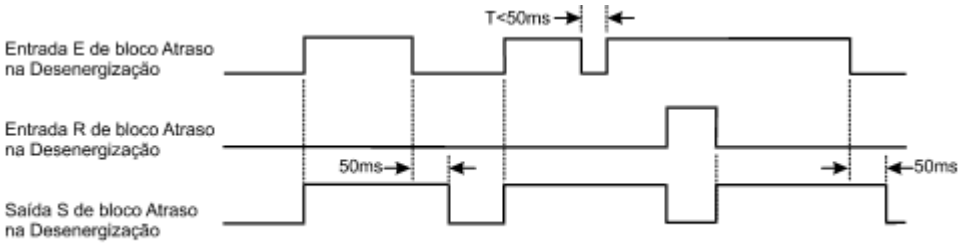
Veja também:
TEMPORIZAÇÃO - Atraso na Desenergização
TEMPORIZAÇÃO - Atraso N Ciclos

TEMPORIZAÇÃO - Atraso na Desenergização

Este bloco gera um atraso entre a desenergização do nodo de Entrada (E) e a desenergização do nodo de Saída (S). Quando o nodo de entrada é acionado o nodo de saída é também acionado imediatamente. Quando o nodo de entrada é desenergizado a saída é mantida energizada durante o tempo programado, antes de ser desenergizada. Caso neste ínterim a entrada seja novamente energizada o bloco de Atraso na Desenergização mantém a saída acionada e volta a temporizar a desenergização da saída S quando o nodo de entrada E voltar a zero. Os nodos de Reset (R) estão internamente interligados e permitem desativar o bloco imediatamente. Note que qualquer sinal ligado a um dos nodos R está automaticamente ligado ao outro nodo R, já que se trata de dois nodos ligados internamente pelo bloco.



Abaixo temos uma representação gráfica do comportamento do bloco de Atraso na Desenergização:



BLOCO ATRASO NA DESENERGIZAÇÃO PROGRAMADO PARA 50ms

Tempo é o operando que determina o tempo do bloco de Temporização. Já a Var.Auxiliar é a variável usada para contar este tempo. Note que Tempo pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

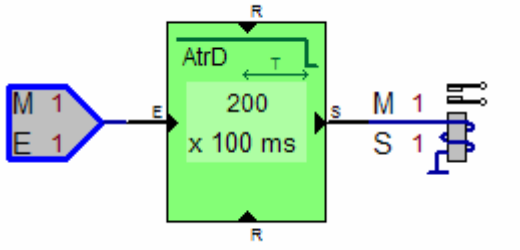
Já Var.Auxiliar não pode ser uma constante, já que deverá ser decrementada durante a operação do bloco. Caso não seja especificada a variável auxiliar no bloco de temporização o compilador irá alocar um espaço vago na memória RAM disponível.

A Escala de tempo determina qual a unidade de temporização a que se refere o operando Tempo. Assim, é possível escolher entre 1ms, 10ms, 100ms, ou ainda 1s (para editar o bloco selecionando a Escala de tempo e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Como operandos word podem assumir valores entre 0 e 65535 resulta que podemos programar o bloco dentro dos seguintes intervalos:

Escala de Tempo	Tempo Mínimo	Tempo Máximo
x 1 ms	1 ms	65,535 s = 1 min 5 s
x 10 ms	10 ms	655,35 s = 10 min 55 s
x 100 ms	100 ms	6553,5 s = 1 h 49 min 13 s
x 1 s	1 s	65535 s = 18 h 12 min 15 s

Já o nodo de Reset (R) permite zerar a contagem de tempo do bloco, e ele passa a temporizar novamente tão logo o nodo de Reset seja desativado (desde que, evidentemente, o nodo de Entrada E esteja ativado).

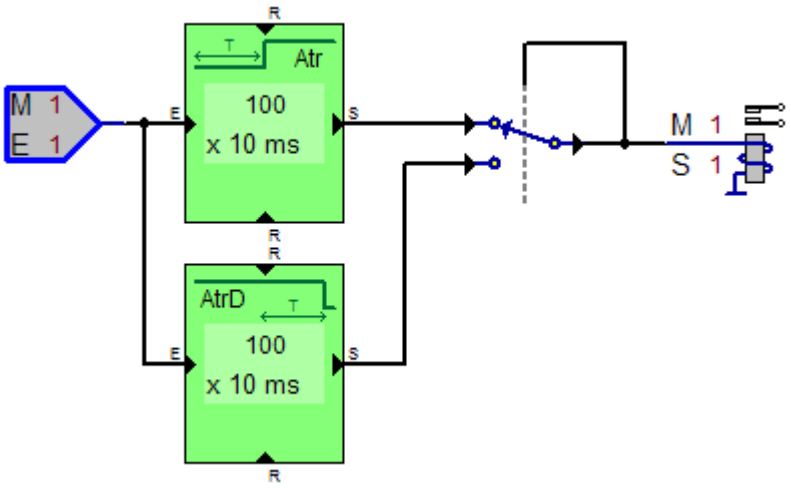
ATENÇÃO: *No caso de blocos de temporização utilizando escala de tempo de 1 ms é necessário que o tempo de ciclo do programa aplicativo seja menor ou igual a 1 ms, ou o bloco de temporização sofrerá um atraso na temporização, devido ao tempo de ciclo do programa ser superior à resolução da temporização. Por exemplo, um bloco de atraso na desenergização de 10ms (com escala de 1ms e variável de tempo igual a 10), se utilizado em um programa com tempo de ciclo de 2 ms, pode apresentar uma temporização de até 13 ms (2 ms adicionais devido ao tempo de ciclo do programa aplicativo e 1 ms adicional devido a imprecisão("jitter") dos blocos de temporização).*



Exemplo de Programa Aplicativo: **Temporização - Atraso na Desenergização.dxg**

Acima temos um pequeno exemplo de uso do bloco de temporização Atraso na Desenergização. No caso, este bloco está sendo usado para aumentar o pulso recebido na entrada E1 do módulo de Expansão M1 (µDX210) para um mínimo de 20 segundos (200 x 100ms) na saída S1 do mesmo módulo. Isso pode ser utilizado em uma minuteria, ou seja, para temporizar o tempo que a iluminação em um corredor de um edifício fica acionada. Assim, quando o botão ligado a E1 é acionado as luzes (comandadas por S1) são mantidas ligadas por 20 segundos. Caso o usuário detecte que levará mais tempo para abrir a fechadura de sua porta ou carregar sua bagagem para dentro do apartamento, basta pressionar novamente o botão ligado a E1 para ganhar mais 20 segundos (mesmo que os primeiros 20 segundos ainda não tenham sido esgotados).

Um exemplo interessante de uso tanto do bloco de Atraso quanto o de Atraso na Desenergização é mostrado a seguir. O programa executa um atraso de 0,1s tanto na energização quanto na desenergização da entrada E1 do primeiro módulo de Expansão µDX210. Com isso, qualquer sinal espúrio que apareça nesta entrada com duração inferior a 100ms será desprezado.
Note que a saída S1 da Expansão µDX210 somente será acionada se a entrada E1 permanecer em um estado estável por mais de 100ms. Quando a saída está desenergizada ela é ligada a saída do bloco de Atraso. Já quando a saída S1 está energizada ela é ligada a saída do bloco de Atraso na Desenergização.

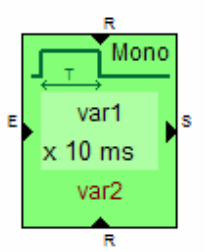


Exemplo de Programa Aplicativo: **Temporização - Atraso Energização - Desenergização.dxg**

Veja também:
TEMPORIZAÇÃO - Atraso
TEMPORIZAÇÃO - Atraso N Ciclos

TEMPORIZAÇÃO - Monoestável

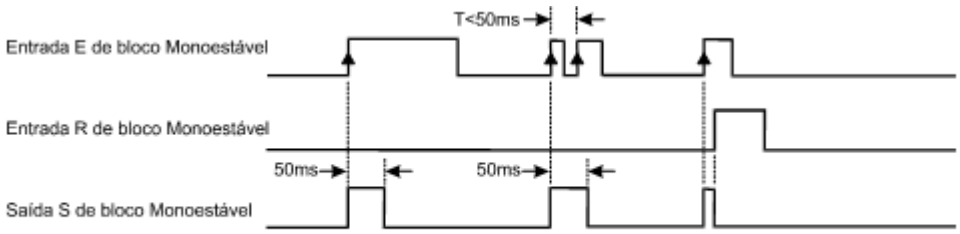
Este bloco gera um pulso de largura programável no nodo de Saída S a cada borda de subida no nodo de Entrada E. Quando o nodo de entrada (E) é acionado o nodo de saída é também acionado imediatamente, mantendo-se energizado durante o tempo programado no operando Tempo. Ao contrário do bloco anterior, o nodo de saída (S) retorna a zero após este tempo, independentemente da entrada (E) ter retornado ou não a zero. Para um novo disparo é preciso que o nodo de entrada (E) retorne a zero e volte a subir. Durante o tempo de Monoestável (saída S ativa) o bloco se torna insensível a entrada E, ou seja, se trata de Monoestável não-retrigável.



Note que, ao contrário dos blocos de Atraso, os blocos de Monoestável são sensíveis a borda de subida do nodo de Entrada E. Já os blocos de Atraso são sensíveis ao nível do nodo de Entrada E.

Os nodos de Reset (R) estão internamente interligados e permitem desativar o bloco imediatamente. Note que qualquer sinal ligado a um dos nodos R está automaticamente ligado ao outro nodo R, já que se trata de dois nodos ligados internamente pelo bloco.

Abaixo temos uma representação gráfica do comportamento do bloco de Monoestável:



BLOCO MONOESTÁVEL PROGRAMADO PARA 50ms

Tempo é o operando que determina o tempo do bloco de Temporização. Já a Var.Auxiliar é a variável usada para contar este tempo. Note que Tempo pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

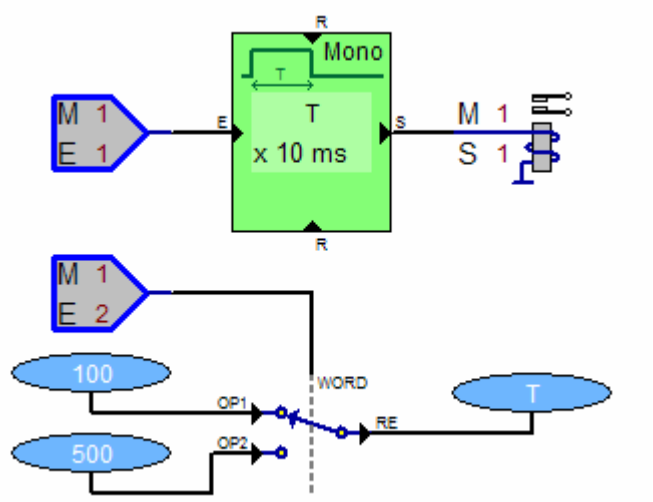
Já Var.Auxiliar não pode ser uma constante, já que deverá ser decrementada durante a operação do bloco. Caso não seja especificada a variável auxiliar no bloco de temporização o compilador irá alocar um espaço vago na memória RAM disponível.

A Escala de tempo determina qual a unidade de temporização a que se refere o operando Tempo. Assim, é possível escolher entre 1ms, 10ms, 100ms, ou ainda 1s (para editar o bloco selecionando a Escala de tempo e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Como operandos word podem assumir valores entre 0 e 65535 resulta que podemos programar o bloco dentro dos seguintes intervalos:

Escala de Tempo	Tempo Mínimo	Tempo Máximo
x 1 ms	1 ms	65,535 s = 1 min 5 s
x 10 ms	10 ms	655,35 s = 10 min 55 s
x 100 ms	100 ms	6553,5 s = 1 h 49 min 13 s
x 1 s	1 s	65535 s = 18 h 12 min 15 s

Já o nodo de Reset (R) permite zerar a contagem de tempo do bloco, e ele passa a temporizar novamente tão logo o nodo de Reset seja desativado (desde que, evidentemente, o nodo de Entrada E esteja ativado).

ATENÇÃO: *No caso de blocos de temporização utilizando escala de tempo de 1 ms é necessário que o tempo de ciclo do programa aplicativo seja menor ou igual a 1 ms, ou o bloco de temporização sofrerá um atraso na temporização, devido ao tempo de ciclo do programa ser superior à resolução da temporização. Por exemplo, um bloco de monoestável de 10ms (com escala de 1ms e variável de tempo igual a 10), se utilizado em um programa com tempo de ciclo de 2 ms, pode apresentar uma temporização de até 13 ms (2 ms adicionais devido ao tempo de ciclo do programa aplicativo e 1 ms adicional devido a imprecisão("jitter") dos blocos de temporização).*



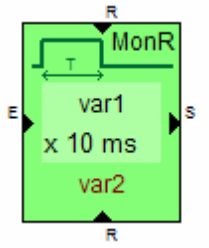
Exemplo de Programa Aplicativo: **Temporização - Monoestável.dxg**

Acima temos um pequeno exemplo de uso do bloco de temporização Monoestável. No caso, este bloco está sendo usado para gerar um pulso de 1 segundo ou 5 segundos na saída S1 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é energizada. A seleção do tempo é feita mudando-se a atribuição de valor para a variável T via entrada E2 do mesmo módulo. Caso a entrada E1 seja desacionada e novamente acionada durante o tempo do Monoestável, este pulso na entrada será ignorado, já que se trata de Monoestável não-retrigável.

Veja também:
TEMPORIZAÇÃO - Monoestável Retrigável

TEMPORIZAÇÃO - Monoestável Retrigável

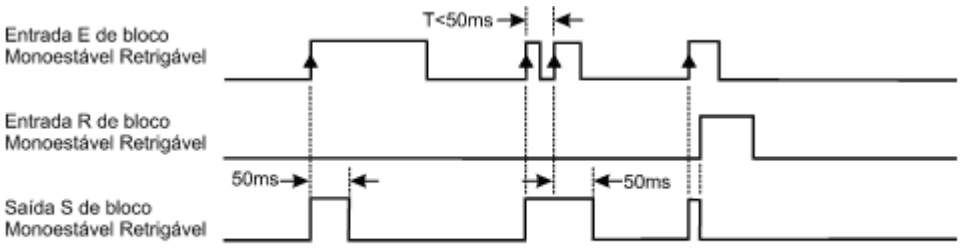
Este bloco gera um pulso de largura programável no nodo de Saída S a cada borda de subida no nodo de Entrada E. Quando o nodo de entrada (E) é acionado o nodo de saída é também acionado imediatamente, mantendo-se energizado durante o tempo programado no operando Tempo. Ao contrário do bloco anterior, o nodo de saída (S) retorna a zero após este tempo, independentemente da entrada (E) ter retornado ou não a zero. Para um novo disparo é preciso que o nodo de entrada (E) retorne a zero e volte a subir. Durante o tempo de Monoestável (saída S ativa) o bloco se mantêm sensível a entrada E, ou seja, se trata de Monoestável Retrigável. Caso durante a temporização do bloco ocorra uma nova borda de subida no nodo E a temporização do bloco é reiniciada.



Note que, ao contrário dos blocos de Atraso, os blocos de Monoestável são sensíveis a borda de subida do nodo de Entrada E. Já os blocos de Atraso são sensíveis ao nível do nodo de Entrada E.

Os nodos de Reset (R) estão internamente interligados e permitem desativar o bloco imediatamente. Note que qualquer sinal ligado a um dos nodos R está automaticamente ligado ao outro nodo R, já que se trata de dois nodos ligados internamente pelo bloco.

Abaixo temos uma representação gráfica do comportamento do bloco de Monoestável Retrigável:



BLOCO MONOESTÁVEL RETRIGÁVEL PROGRAMADO PARA 50ms

Tempo é o operando que determina o tempo do bloco de Temporização. Já a Var.Auxiliar é a variável usada para contar este tempo. Note que Tempo pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

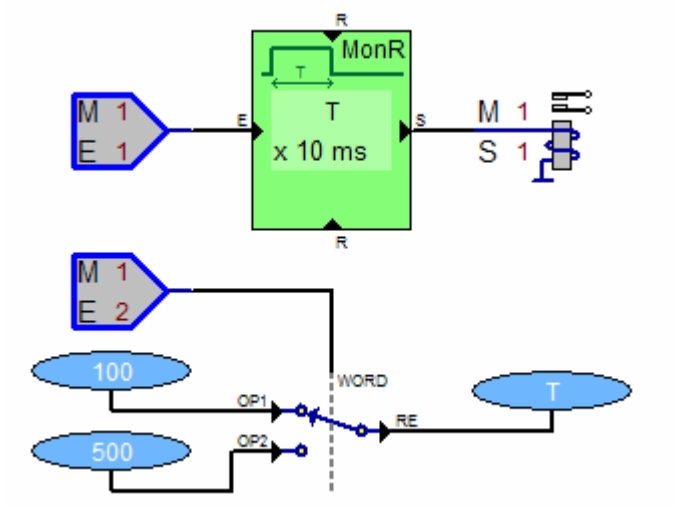
Já Var.Auxiliar não pode ser uma constante, já que deverá ser decrementada durante a operação do bloco. Caso não seja especificada a variável auxiliar no bloco de temporização o compilador irá alocar um espaço vago na memória RAM disponível.

A Escala de tempo determina qual a unidade de temporização a que se refere o operando Tempo. Assim, é possível escolher entre 1ms, 10ms, 100ms, ou ainda 1s (para editar o bloco selecionando a Escala de tempo e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Como operandos word podem assumir valores entre 0 e 65535 resulta que podemos programar o bloco dentro dos seguintes intervalos:

Escala de Tempo	Tempo Mínimo	Tempo Máximo
x 1 ms	1 ms	65,535 s = 1 min 5 s
x 10 ms	10 ms	655,35 s = 10 min 55 s
x 100 ms	100 ms	6553,5 s = 1 h 49 min 13 s
x 1 s	1 s	65535 s = 18 h 12 min 15 s

Já o nodo de Reset (R) permite zerar a contagem de tempo do bloco, e ele passa a temporizar novamente tão logo o nodo de Reset seja desativado (desde que, evidentemente, o nodo de Entrada E esteja ativado).

ATENÇÃO: *No caso de blocos de temporização utilizando escala de tempo de 1 ms é necessário que o tempo de ciclo do programa aplicativo seja menor ou igual a 1 ms, ou o bloco de temporização sofrerá um atraso na temporização, devido ao tempo de ciclo do programa ser superior à resolução da temporização. Por exemplo, um bloco de monoestável retrigável de 10ms (com escala de 1ms e variável de tempo igual a 10), se utilizado em um programa com tempo de ciclo de 2 ms, pode apresentar uma temporização de até 13 ms (2 ms adicionais devido ao tempo de ciclo do programa aplicativo e 1 ms adicional devido a imprecisão("jitter") dos blocos de temporização).*



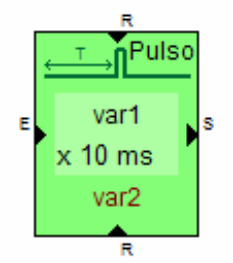
Exemplo de Programa Aplicativo: **Temporização - Monoestável Retrigável.dxg**

Acima temos um exemplo de uso do bloco de temporização Monoestável Retrigável. O programa gera um pulso de 1 ou 5 segundos na saída S1 cada vez que a entrada E1 do módulo de Expansão M1 (µDX210) é energizada. A seleção do tempo é feita mudando-se a atribuição de valor para a variável T via entrada E2 do mesmo módulo. Caso a entrada E1 seja desacionada e novamente acionada durante o tempo do Monoestável Retrigável, este pulso na entrada irá reiniciar a contagem de tempo do bloco, já que se trata de Monoestável Retrigável. Novamente pode ser usado para gerar uma minuteria, como no exemplo de bloco de Atraso na Desenergização, com a vantagem de que, caso seja pressionada constantemente a botoeira ligada a entrada E1 o programa não irá manter energizada a saída S1.

Veja também:
TEMPORIZAÇÃO - Monoestável

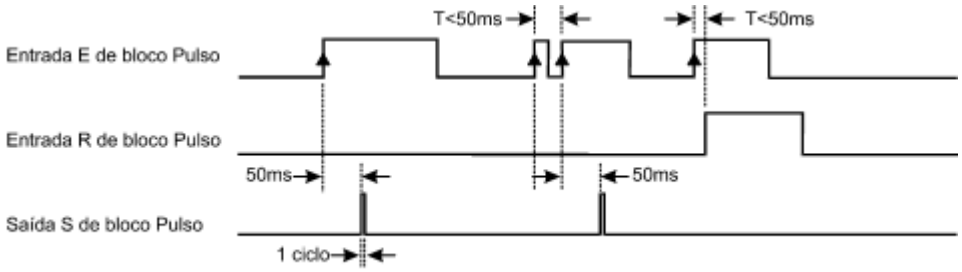
TEMPORIZAÇÃO - Pulso

Este bloco retarda a energização do nodo de saída S durante o tempo programado, quando detecta energização do nodo de entrada E, e mantêm a saída ligada somente durante um ciclo do CLP. Note que somente irá gerar o pulso na saída S se a entrada E se mantiver ligada por um tempo superior ao tempo programado. Para um novo disparo é preciso que o nodo de entrada (E) retorne a zero e volte a subir.



Os nodos de Reset (R) estão internamente interligados e permitem desativar o bloco imediatamente. Note que qualquer sinal ligado a um dos nodos R está automaticamente ligado ao outro nodo R, já que se trata de dois nodos ligados internamente pelo bloco.

Abaixo temos uma representação gráfica do comportamento do bloco de Pulso:



BLOCO PULSO PROGRAMADO PARA 50ms

Tempo é o operando que determina o tempo do bloco de Temporização. Já a Var.Auxiliar é a variável usada para contar este tempo. Note que Tempo pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

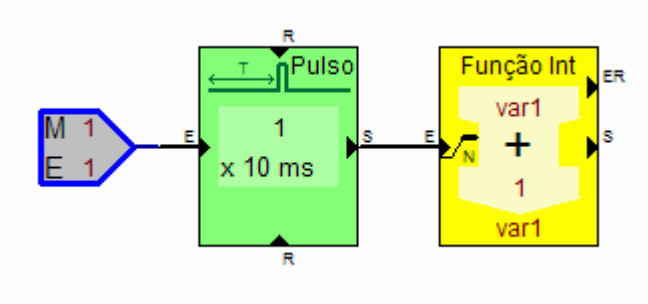
Já Var.Auxiliar não pode ser uma constante, já que deverá ser decrementada durante a operação do bloco. Caso não seja especificada a variável auxiliar no bloco de temporização o compilador irá alocar um espaço vago na memória RAM disponível.

A Escala de tempo determina qual a unidade de temporização a que se refere o operando Tempo. Assim, é possível escolher entre 1ms, 10ms, 100ms, ou ainda 1s (para editar o bloco selecionando a Escala de tempo e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Como operandos word podem assumir valores entre 0 e 65535 resulta que podemos programar o bloco dentro dos seguintes intervalos:

Escala de Tempo	Tempo Mínimo	Tempo Máximo
x 1 ms	1 ms	65,535 s = 1 min 5 s
x 10 ms	10 ms	655,35 s = 10 min 55 s
x 100 ms	100 ms	6553,5 s = 1 h 49 min 13 s
x 1 s	1 s	65535 s = 18 h 12 min 15 s

Já o nodo de Reset (R) permite zerar a contagem de tempo do bloco, e ele passa a temporizar novamente tão logo o nodo de Reset seja desativado (desde que, evidentemente, o nodo de Entrada E esteja ativado).

ATENÇÃO: *No caso de blocos de temporização utilizando escala de tempo de 1 ms é necessário que o tempo de ciclo do programa aplicativo seja menor ou igual a 1 ms, ou o bloco de temporização sofrerá um atraso na temporização, devido ao tempo de ciclo do programa ser superior à resolução da temporização. Por exemplo, um bloco de Pulso de 10ms (com escala de 1ms e variável de tempo igual a 10), se utilizado em um programa com tempo de ciclo de 2 ms, pode apresentar uma temporização de até 13 ms (2 ms adicionais devido ao tempo de ciclo do programa aplicativo e 1 ms adicional devido a imprecisão("jitter") dos blocos de temporização).*



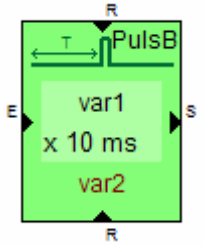
Exemplo de Programa Aplicativo: **Temporização - Pulso.dxg**

Acima temos um exemplo de uso do bloco de temporização Pulso. O programa gera um pulso de apenas um ciclo a cada energização da entrada E1 do módulo de Expansão M1 (µDX210). Este pulso é usado para incrementar a variável var1 a cada nova energização de E1. Note que o bloco de aritmética inteira foi programado para entrada sensível ao nível e, portanto, caso o nodo de entrada fique mais de um ciclo energizado ocorreria incrementos múltiplos em var1. Óbvio que poderíamos usar este bloco com entrada sensível à borda, dispensando o bloco de Pulso. Mas com este bloco a entrada E1 somente incrementa a variável var1 se a energização em E1 durar um mínimo de 10ms. Pulsos mais curtos são ignorados. Isso pode ser útil para filtrar ruídos existentes no sinal de entrada.

Veja também:
TEMPORIZAÇÃO - Pulso Borda

TEMPORIZAÇÃO - Pulso Borda

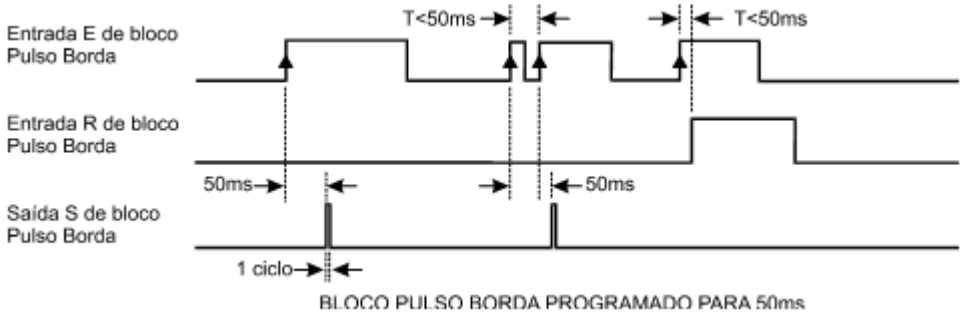
Este bloco retarda a energização do nodo de saída S durante o tempo programado, quando detecta energização do nodo de entrada E, e mantêm a saída ligada somente durante um ciclo do CLP. Note que este bloco, ao contrário do anterior, irá gerar o pulso na saída S mesmo que a entrada E retorne a zero antes de decorrido o tempo programado. Para um novo disparo é preciso que o nodo de entrada (E) retorne a zero e volte a subir.



Os nodos de Reset (R) estão internamente interligados e permitem desativar o bloco

imediatamente. Note que qualquer sinal ligado a um dos nodos R está automaticamente ligado ao outro nodo R, já que se trata de dois nodos ligados internamente pelo bloco.

Abaixo temos uma representação gráfica do comportamento do bloco de Pulso Borda:



BLOCO PULSO BORDA PROGRAMADO PARA 50ms

Tempo é o operando que determina o tempo do bloco de Temporização. Já a Var.Auxiliar é a variável usada para contar este tempo. Note que Tempo pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

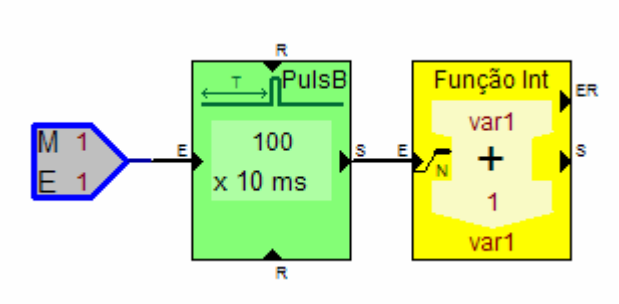
Já Var.Auxiliar não pode ser uma constante, já que deverá ser decrementada durante a operação do bloco. Caso não seja especificada a variável auxiliar no bloco de temporização o compilador irá alocar um espaço vago na memória RAM disponível.

A Escala de tempo determina qual a unidade de temporização a que se refere o operando Tempo. Assim, é possível escolher entre 1ms, 10ms, 100ms, ou ainda 1s (para editar o bloco selecionando a Escala de tempo e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Como operandos word podem assumir valores entre 0 e 65535 resulta que podemos programar o bloco dentro dos seguintes intervalos:

Escala de Tempo	Tempo Mínimo	Tempo Máximo
x 1 ms	1 ms	65,535 s = 1 min 5 s
x 10 ms	10 ms	655,35 s = 10 min 55 s
x 100 ms	100 ms	6553,5 s = 1 h 49 min 13 s
x 1 s	1 s	65535 s = 18 h 12 min 15 s

Já o nodo de Reset (R) permite zerar a contagem de tempo do bloco, e ele passa a temporizar novamente tão logo o nodo de Reset seja desativado (desde que, evidentemente, o nodo de Entrada E esteja ativado).

AT*ENÇÃO: No caso de blocos de temporização utilizando escala de tempo de 1 ms é necessário que o tempo de ciclo do programa aplicativo seja menor ou igual a 1 ms, ou o bloco de temporização sofrerá um atraso na temporização, devido ao tempo de ciclo do programa ser superior à resolução da temporização. Por exemplo, um bloco de Pulso Borda de 10ms (com escala de 1ms e variável de tempo igual a 10), se utilizado em um programa com tempo de ciclo de 2 ms, pode apresentar uma temporização de até 13 ms (2 ms adicionais devido ao tempo de ciclo do programa aplicativo e 1 ms adicional devido a imprecisão("jitter") dos blocos de temporização).*



Exemplo de Programa Aplicativo: **Temporização - Pulso Borda.dxg**

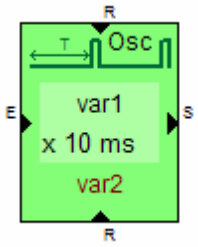
Acima temos um exemplo de uso do bloco de temporização Pulso Borda. O programa gera um pulso de apenas um ciclo a cada energização da entrada E1 do módulo de Expansão M1 (µDX210). Este pulso é usado para incrementar a variável var1 a cada nova energização de E1. Note que o bloco de aritmética inteira foi programado para entrada sensível ao nível e, portanto, caso o nodo de entrada fique mais de um ciclo energizado ocorreria incrementos múltiplos em var1. Óbvio que poderíamos usar este bloco com entrada sensível à borda, dispensando o bloco de Pulso. Mas com este bloco a entrada E1 somente incrementa a variável var1 a cada segundo, independentemente do número de energizações em E1. Ou seja, o µDX200 irá ficar sensível a entrada E1 apenas para um pulso por segundo, sendo que os demais serão desprezados.

Veja também:
TEMPORIZAÇÃO - Pulso

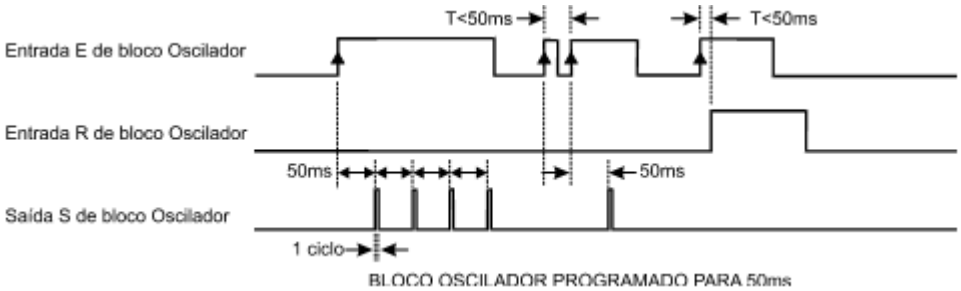
TEMPORIZAÇÃO - Oscilador

Este bloco gera pulsos no nodo de Saída (S) com largura de um ciclo de execução do programa aplicativo, a cada tempo programado pelo operando Tempo. Para habilitá-lo é necessário energizar o nodo de Entrada (E).

Os nodos de Reset (R) estão internamente interligados e permitem desativar o bloco imediatamente. Note que qualquer sinal ligado a um dos nodos R está automaticamente ligado ao outro nodo R, já que se trata de dois nodos ligados internamente pelo bloco.



Abaixo temos uma representação gráfica do comportamento do bloco de Oscilador:



BLOCO OSCILADOR PROGRAMADO PARA 50ms

Tempo é o operando que determina o tempo do bloco de Temporização. Já a Var.Auxiliar é a variável usada para contar este tempo. Note que Tempo pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

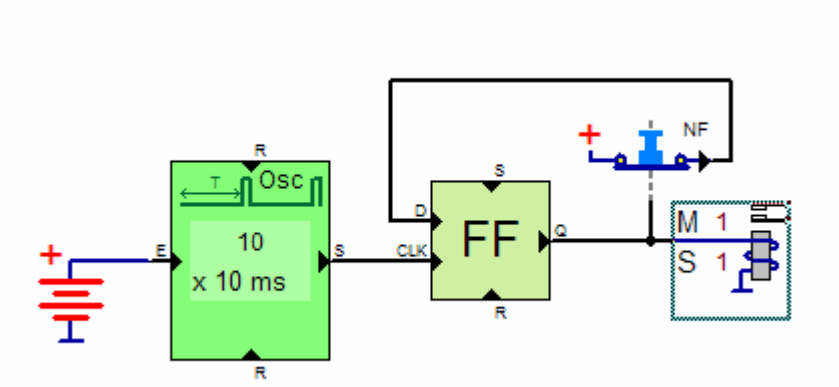
Já Var.Auxiliar não pode ser uma constante, já que deverá ser decrementada durante a operação do bloco. Caso não seja especificada a variável auxiliar no bloco de temporização o compilador irá alocar um espaço vago na memória RAM disponível.

A Escala de tempo determina qual a unidade de temporização a que se refere o operando Tempo. Assim, é possível escolher entre 1ms, 10ms, 100ms, ou ainda 1s (para editar o bloco selecionando a Escala de tempo e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Como operandos word podem assumir valores entre 0 e 65535 resulta que podemos programar o bloco dentro dos seguintes intervalos:

Escala de Tempo	Tempo Mínimo	Tempo Máximo
x 1 ms	1 ms	65,535 s = 1 min 5 s
x 10 ms	10 ms	655,35 s = 10 min 55 s
x 100 ms	100 ms	6553,5 s = 1 h 49 min 13 s
x 1 s	1 s	65535 s = 18 h 12 min 15 s

Já o nodo de Reset (R) permite zerar a contagem de tempo do bloco, e ele passa a temporizar novamente tão logo o nodo de Reset seja desativado (desde que, evidentemente, o nodo de Entrada E esteja ativado).

AT*ENÇÃO*: *No caso de blocos de temporização utilizando escala de tempo de 1 ms é necessário que o tempo de ciclo do programa aplicativo seja menor ou igual a 1 ms, ou o bloco de temporização sofrerá um atraso na temporização, devido ao tempo de ciclo do programa ser superior à resolução da temporização. Por exemplo, um bloco de Oscilador de 10ms (com escala de 1ms e variável de tempo igual a 10), se utilizado em um programa com tempo de ciclo de 2 ms, pode apresentar uma temporização de até 13 ms (2 ms adicionais devido ao tempo de ciclo do programa aplicativo e 1 ms adicional devido a imprecisão("jitter") dos blocos de temporização). É claro que esta imprecisão, no caso do bloco Oscilador, só poderá ocorrer no tempo para o primeiro pulso do bloco. Os demais pulsos gerados terão o período exato programado.*

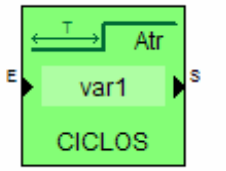


Exemplo de Programa Aplicativo: **Temporização - Oscilador.dxg**

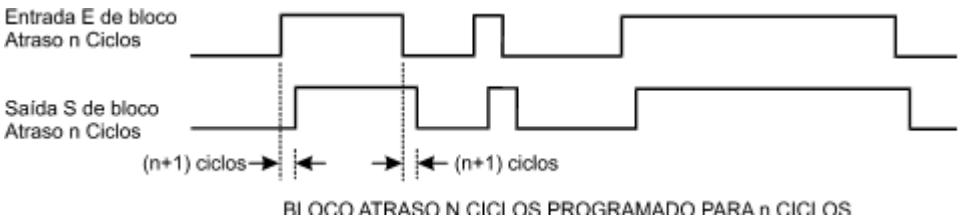
Acima temos um exemplo de uso do bloco de temporização Oscilador. O bloco Oscilador gera pulsos constantemente na saída S (já que a entrada E do bloco está sempre ativada). Estes pulsos trocam de estado o bloco de Flip-flop. A cada pulso a saída Q do flip-flop (FF) é invertida. Com isso, surge uma onda quadrada de 5Hz na saída S1 do módulo de Expansão M1 (µDX210).

TEMPORIZAÇÃO - Atraso N Ciclos

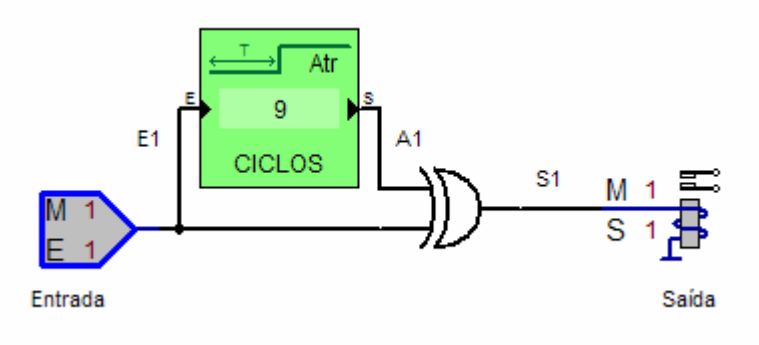
Este bloco gera um atraso entre o sinal presente na Entrada (E) e o sinal na Saída (S) de n ciclos de execução do programa aplicativo. O valor de n pode variar entre 0 e 15, correspondendo a um atraso entre 1 e 16 ciclos. Note que este bloco não possui nodos de Reset. O bloco pode ser usado para compensar atrasos entre ramos diferentes do programa aplicativo, fazendo com que os sinais estejam presentes em determinado ponto simultaneamente.



Abaixo temos uma representação gráfica do comportamento do bloco de Atraso de n Ciclos:

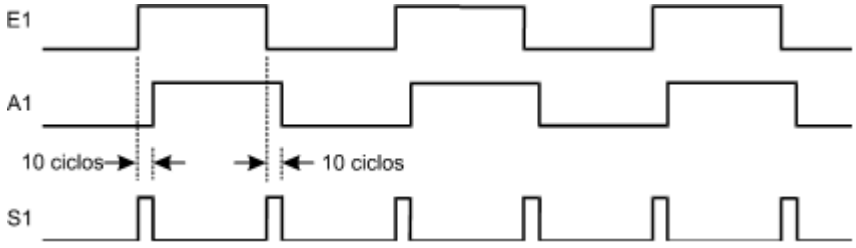


Ciclos é o operando que determina o número de ciclos de execução do programa aplicativo a serem introduzidos de atraso. Como cada bloco do µDX200 introduz um atraso de um ciclo, o valor inserido em Ciclos é acrescido em um ciclo de atraso (já que existe o ciclo de atraso inerente ao bloco de temporização). Assim, se especificarmos Ciclos = 10 serão introduzidos 11 ciclos de atraso. Note que Ciclos deve ser uma constante byte (valor numérico em decimal ou hexadecimal), já que o µDX200 não trata variáveis tipo byte.

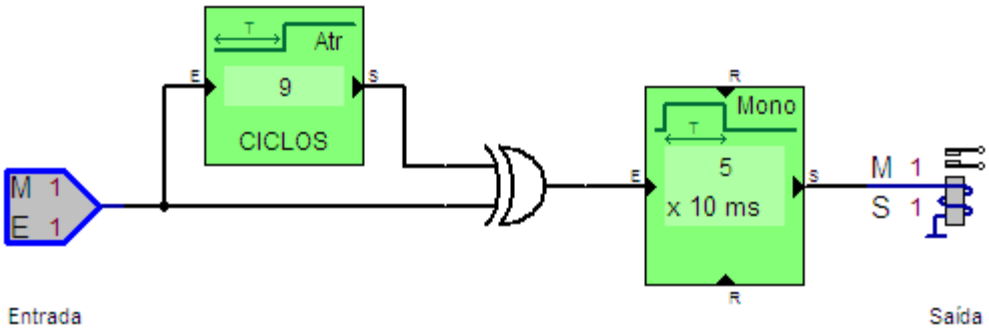


Exemplo de Programa Aplicativo: **Temporização - Atraso N Ciclos.dxg**

Acima temos um exemplo de uso do bloco de temporização Atraso N Ciclos. No caso, este bloco está sendo usado para introduzir um atraso de 10 ciclos (n+1) no sinal E1. A seguir, tanto E1 quanto E1 atrasado (A1) entram em uma porta XOR.. Com isso, o sinal dobra em freqüência (já que é gerado um pulso em S1 a cada transição do sinal E1). Portanto, na saída S1 do módulo de Expansão M1 (µDX210) irá surgir uma onda com o dobro da freqüência do sinal aplicado na entrada E1. Temos um dobrador de freqüência. As formas de onda são:



Os pulsos na saida S1 terão duração de apenas 10 ciclos do µDX200, o que no caso de programas tão pequenos como este exemplo é da ordem de 250µs. Ou seja, os pulsos terão duração de cerca de 2,5ms, insuficientes para acionar o relé de saída de uma Expansão µDX210 (no caso de uma Expansão µDX211, com saídas de estado sólido, deve ser possível observar o acionamento das saídas durante este breve período com instrumento adequado, como osciloscópio). Então, podemos aumentar o pulso de saída para 50ms através de um bloco de Monoestável:



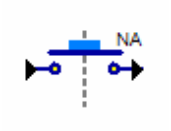
Exemplo de Programa Aplicativo: **Temporização - Atraso N Ciclos 2.dxg**

Veja também:
TEMPORIZAÇÃO - Atraso
TEMPORIZAÇÃO - Atraso na Desenergização

GERAL - Chave NA

A chave NA (Normalmente Aberta) é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica o botão que aparece no centro tem dois contatos por onde "passará" a corrente em determinadas condições.

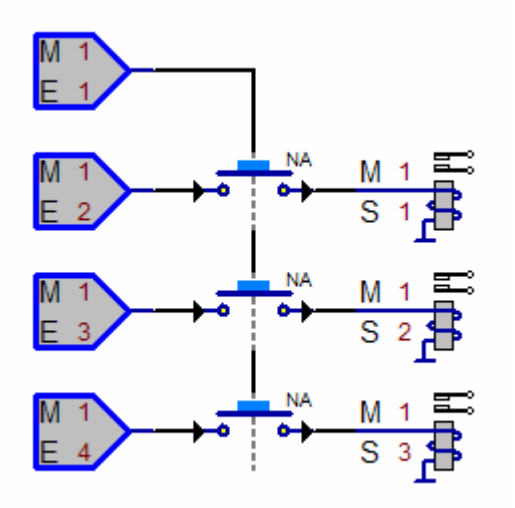
A chave Normalmente Aberta (NA) precisa que o nodo de controle (linha pontilhada) seja ativado para que ela permita passar o estado do nodo de entrada para o nodo de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.



Um exemplo de uso de chave NA é o interruptor de campainha de residências. Ao acioná-lo ele permite a passagem de corrente elétrica, acionando a campainha.

Note que a chave NA, ao contrário de seu similar físico (interruptor elétrico), permite a transmissão de sinal apenas do nodo de entrada (nodo à esquerda) para o nodo de saída (nodo à direita), como indicado pelas setas. Caso outro bloco ative o nodo de saída de uma chave NA, o nodo de entrada da mesma não será acionado, mesmo que a chave esteja fechada (nodo de controle ligado).

No exemplo a seguir, quando a entrada E1 do módulo de Expansão M1 (µDX210) é acionada, permite que os sinais presentes nas entradas E2, E3 e E4 do mesmo módulo sejam replicados nas saídas S1, S2, e S3, respectivamente. Quando E1 está desativado as chaves NA abrem e nenhuma saída é ativada, independentemente do estado de E2, E3 e E4. Note que E1 controla as três chaves NA, já que os nodos de controle delas estão interligados.



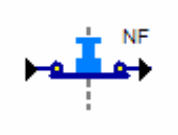
Exemplo de Programa Aplicativo: **Geral - Chave NA.dxg**

Veja também:
GERAL - Chave NF
GERAL - Chave Inversora
GERAL - Seletor de Nodo

GERAL - Chave NF

A chave NF (Normalmente Fechada) é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica o botão que aparece no centro tem dois contatos por onde "passará" a corrente em determinadas condições.

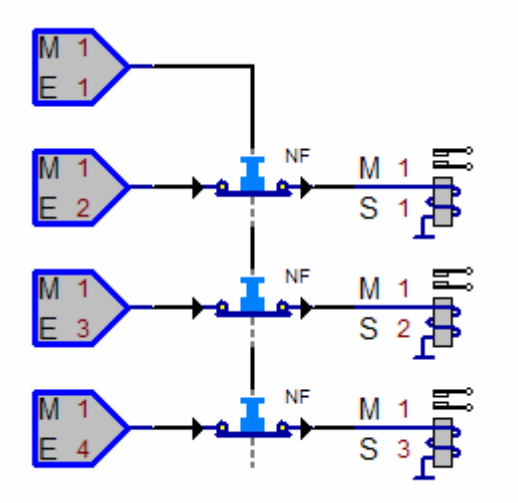
A chave Normalmente Fechada (NF) precisa que o nodo de controle (linha pontilhada) seja desativado para que ela permita passar o estado do nodo de entrada para o nodo de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.



Um exemplo de chave NF usada comumente é a chave de iluminação interna de refrigeradores. Quando pressionada a lâmpada permanece desligada (porta fechada), e ao desacionar a chave (porta aberta) a lâmpada acende.

Note que a chave NF, ao contrário de seu similar físico (interruptor elétrico), permite a transmissão de sinal apenas do nodo de entrada (nodo à esquerda) para o nodo de saída (nodo à direita), como indicado pelas setas. Caso outro bloco ative o nodo de saída de uma chave NF, o nodo de entrada da mesma não será acionado, mesmo que a chave esteja fechada (nodo de controle desligado).

No exemplo a seguir, quando a entrada E1 do módulo de Expansão M1 (µDX210) é desacionada, permite que os sinais presentes nas entradas E2, E3 e E4 do mesmo módulo sejam replicados nas saídas S1, S2, e S3, respectivamente. Quando E1 está ativado as chaves NF abrem e nenhuma saída é ativada, independentemente do estado de E2, E3 e E4. Note que E1 controla as três chaves NF, já que os nodos de controle delas estão interligados.

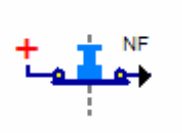


Exemplo de Programa Aplicativo: **Geral - Chave NF.dxg**

Veja também:
GERAL - Chave NA
GERAL - Chave Inversora
GERAL - Seletor de Nodo

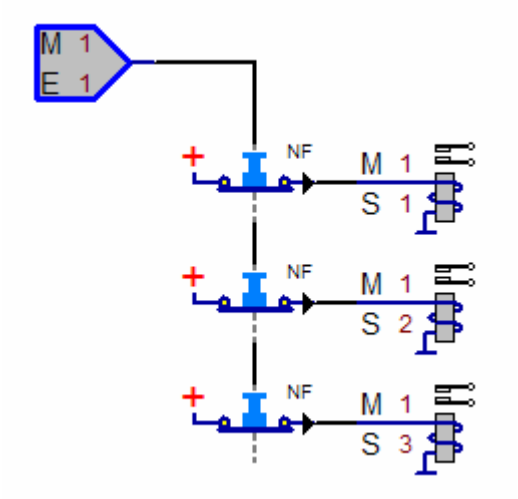
GERAL - Chave Inversora

A chave Inversora é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica o botão que aparece no centro tem dois contatos por onde "passará" a corrente em determinadas condições. Na verdade, se trata de uma chave NF (Normalmente Fechada) com o nodo de entrada constantemente ligado. Com isso, caso se ative o nodo de controle a chave abre e o nodo de saída é desligado. Já se for desligado o nodo de controle a chave fecha e o nodo de saída será ativado. Com isso, o estado do nodo de saída é invertido em relação ao estado do nodo de controle.



Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.

No exemplo a seguir, quando a entrada E1 do módulo de Expansão M1 (µDX210) é desacionada, ativa as três saídas S1, S2 e S3 do mesmo módulo. Já quando esta entrada é acionada desativa as três saídas. Note que E1 controla as três chaves NF, já que os nodos de controle delas estão interligados.



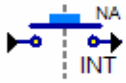
Exemplo de Programa Aplicativo: **Geral - Chave Inversora.dxg**

Veja também:
GERAL - Chave NA
GERAL - Chave NF
GERAL - Seletor de Nodo

GERAL - Chave NA Inteira

A chave NA (Normalmente Aberta) é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica o botão que aparece no centro tem dois contatos por onde "passará" a corrente em determinadas condições.

A chave Normalmente Aberta Inteira (NA INT) precisa que o nodo de controle (linha pontilhada) seja ativado para que ela permita passar o valor da variável inteira de entrada para a variável inteira de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.

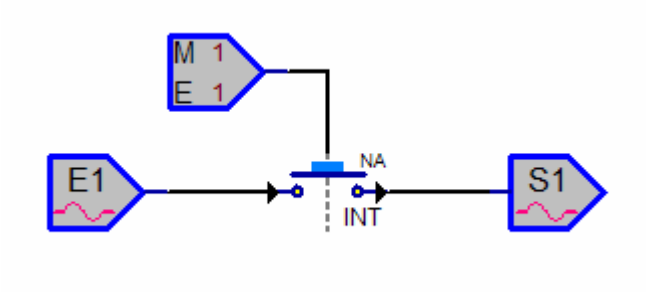


Ou seja, esta chave transmite uma variável inteira em vez de um nodo (variável binária), como no caso da chave NA comum.

Note que a chave NA INT, ao contrário de seu similar físico (interruptor elétrico), permite a transmissão de sinal apenas da variável de entrada (conexão à esquerda) para a variável de saída (conexão à direita), como indicado pelas setas. Caso outro bloco atribua determinado valor à variável de saída de uma chave NA INT, a variável de entrada da mesma não será afetada, mesmo que a chave esteja fechada (nodo de controle ligado).

No exemplo a seguir, quando a entrada digital E1 do módulo de Expansão M1 (µDX210) é acionada, permite que o valor de conversão presente na entrada analógica E1 venha a ser atribuído a saída analógica S1 do Controlador µDX200. Portanto, enquanto a entrada E1 do µDX210 estiver acionada a saída analógica S1 do µDX200 irá seguir o valor de tensão aplicado na entrada analógica E1 do µDX200. Se tanto a entrada E1 quanto a saída S1 do µDX200 estiverem selecionadas para escala de 0-10V, por exemplo, ao aplicarmos 5V em E1 deverá surgir 5V em S1, e assim por diante. Já se desligarmos a entrada E1 do µDX210 a chave NA INT irá abrir, e o valor da saída analógica S1 ficará congelado no último valor lido na entrada analógica E1 do µDX200 imediatamente antes da abertura da chave. Este bloco permite, portanto, implementar um "sample and hold" com facilidade.

Os números inteiros permitem representar valores de -32768 a 32767. Como a entrada analógica pode assumir valores entre 0 e 4095 (as vezes assume valores levemente superiores, como 4096, devido a calibração por software feita durante a fabricação das entradas e saídas analógicas do µDX200), os valores estão dentro da faixa de representatividade da chave NA INT.



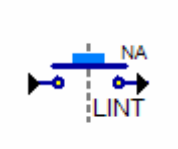
Exemplo de Programa Aplicativo: **Geral - Chave NA Inteira.dxg**

Veja também:
GERAL - Seletor de Variável Inteira
GERAL - Chave NA LongInt
GERAL - Chave NA Word
GERAL - Chave NA Real

GERAL - Chave NA LongInt

A chave NA (Normalmente Aberta) é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica o botão que aparece no centro tem dois contatos por onde "passará" a corrente em determinadas condições.

A chave Normalmente Aberta LongInt (NA LINT) precisa que o nodo de controle (linha pontilhada) seja ativado para que ela permita passar o valor da variável longint de entrada para a variável longint de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.

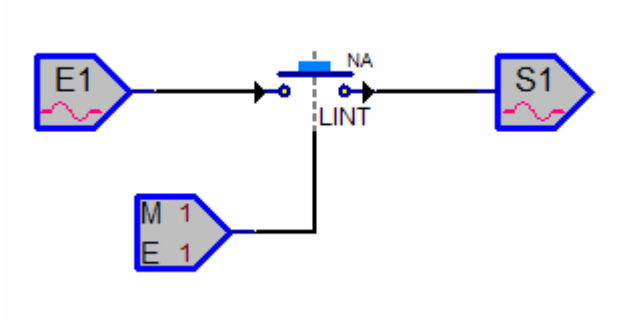


Ou seja, esta chave transmite uma variável longint em vez de um nodo (variável binária), como no caso da chave NA comum.

Note que a chave NA LINT, ao contrário de seu similar físico (interruptor elétrico), permite a transmissão de sinal apenas da variável de entrada (conexão à esquerda) para a variável de saída (conexão à direita), como indicado pelas setas. Caso outro bloco atribua determinado valor à variável de saída de uma chave NA LINT, a variável de entrada da mesma não será afetada, mesmo que a chave esteja fechada (nodo de controle ligado).

No exemplo a seguir, quando a entrada digital E1 do módulo de Expansão M1 (µDX210) é acionada, permite que o valor de conversão presente na entrada analógica E1 venha a ser atribuído a saída analógica S1 do Controlador µDX200. Portanto, enquanto a entrada E1 do µDX210 estiver acionada a saída analógica S1 do µDX200 irá seguir o valor de tensão aplicado na entrada analógica E1 do µDX200. Se tanto a entrada E1 quanto a saída S1 do µDX200 estiverem selecionadas para escala de 0-10V, por exemplo, ao aplicarmos 5V em E1 deverá surgir 5V em S1, e assim por diante. Já se desligarmos a entrada E1 do µDX210 a chave NA LINT irá abrir, e o valor da saída analógica S1 ficará congelado no último valor lido na entrada analógica E1 do µDX200 imediatamente antes da abertura da chave. Este bloco permite, portanto, implementar um "sample and hold" com facilidade.

Os números longint permitem representar valores entre -2.147.483.648 e 2.147.483.647. Como a entrada analógica pode assumir valores entre 0 e 4095 (as vezes assume valores levemente superiores, como 4096, devido a calibração por software feita durante a fabricação das entradas e saídas analógicas do µDX200), os valores estão dentro da faixa de representatividade da chave NA LINT. Como as entradas e saídas analógicas do µDX200 são representadas por variáveis inteiras a compilação do programa de exemplo gera duas advertências devido a conexão de variável inteira com variável longint.



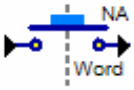
Exemplo de Programa Aplicativo: **Geral - Chave NA LongInt.dxg**

Veja também:
GERAL - Seletor de Variável LongInt
GERAL - Chave NA Inteira
GERAL - Chave NA Word
GERAL - Chave NA Real

GERAL - Chave NA Word

A chave NA (Normalmente Aberta) é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica o botão que aparece no centro tem dois contatos por onde "passará" a corrente em determinadas condições.

A chave Normalmente Aberta Word (NA Word) precisa que o nodo de controle (linha pontilhada) seja ativado para que ela permita passar o valor da variável word de entrada para a variável word de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.

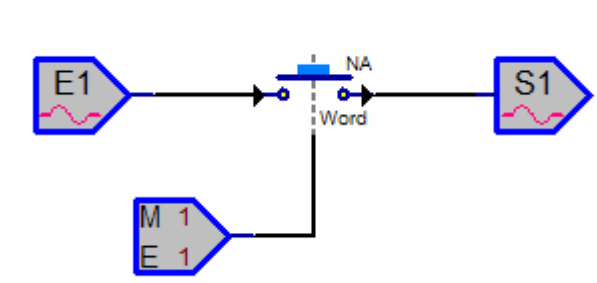


Ou seja, esta chave transmite uma variável word em vez de um nodo (variável binária), como no caso da chave NA comum.

Note que a chave NA Word, ao contrário de seu similar físico (interruptor elétrico), permite a transmissão de sinal apenas da variável de entrada (conexão à esquerda) para a variável de saída (conexão à direita), como indicado pelas setas. Caso outro bloco atribua determinado valor à variável de saída de uma chave NA Word, a variável de entrada da mesma não será afetada, mesmo que a chave esteja fechada (nodo de controle ligado).

No exemplo a seguir, quando a entrada digital E1 do módulo de Expansão M1 (µDX210) é acionada, permite que o valor de conversão presente na entrada analógica E1 venha a ser atribuído a saída analógica S1 do Controlador µDX200. Portanto, enquanto a entrada E1 do µDX210 estiver acionada a saída analógica S1 do µDX200 irá seguir o valor de tensão aplicado na entrada analógica E1 do µDX200. Se tanto a entrada E1 quanto a saída S1 do µDX200 estiverem selecionadas para escala de 0-10V, por exemplo, ao aplicarmos 5V em E1 deverá surgir 5V em S1, e assim por diante. Já se desligarmos a entrada E1 do µDX210 a chave NA Word irá abrir, e o valor da saída analógica S1 ficará congelado no último valor lido na entrada analógica E1 do µDX200 imediatamente antes da abertura da chave. Este bloco permite, portanto, implementar um "sample and hold" com facilidade.

Os números word permitem representar valores de 0 a 65535. Como a entrada analógica pode assumir valores entre 0 e 4095 (as vezes assume valores levemente superiores, como 4096, devido a calibração por software feita durante a fabricação das entradas e saídas analógicas do µDX200), os valores estão dentro da faixa de representatividade da chave NA Word. Como as entradas e saídas analógicas do µDX200 são representadas por variáveis inteiras a compilação do programa de exemplo gera duas advertências devido a conexão de variável inteira com variável word.



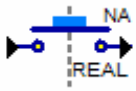
Exemplo de Programa Aplicativo: **Geral - Chave NA Word.dxg**

Veja também:
GERAL - Seletor de Variável Word
GERAL - Chave NA Inteira
GERAL - Chave NA LongInt
GERAL - Chave NA Real

GERAL - Chave NA Real

A chave NA (Normalmente Aberta) é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica o botão que aparece no centro tem dois contatos por onde "passará" a corrente em determinadas condições.

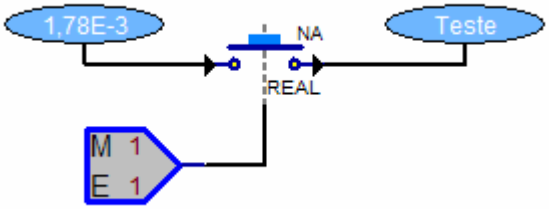
A chave Normalmente Aberta Real (NA Real) precisa que o nodo de controle (linha pontilhada) seja ativado para que ela permita passar o valor da variável real de entrada para a variável real de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.



Ou seja, esta chave transmite uma variável real em vez de um nodo (variável binária), como no caso da chave NA comum.

Note que a chave NA Real, ao contrário de seu similar físico (interruptor elétrico), permite a transmissão de sinal apenas da variável de entrada (conexão à esquerda) para a variável de saída (conexão à direita), como indicado pelas setas. Caso outro bloco atribua determinado valor à variável de saída de uma chave NA Real, a variável de entrada da mesma não será afetada, mesmo que a chave esteja fechada (nodo de controle ligado).

No exemplo a seguir, quando a entrada digital E1 do módulo de Expansão M1 (µDX210) é acionada, permite que o valor constante 1,78E-3 (0,00178) venha a ser atribuído a variável real Teste do Controlador µDX200.

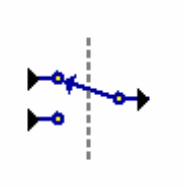


Exemplo de Programa Aplicativo: **Geral - Chave NA Real.dxg**

Veja também:
GERAL - Seletor de Variável Real
GERAL - Chave NA Inteira
GERAL - Chave NA LongInt
GERAL - Chave NA Word

GERAL - Seletor de Nodo

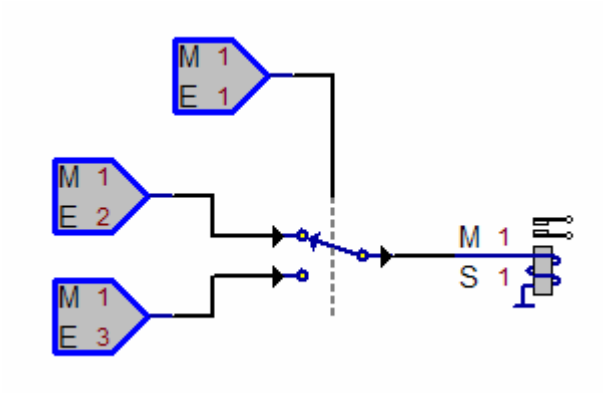
O Seletor de Nodo é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica a seta que aparece no centro tem dois contatos por onde "passará" a corrente em determinadas condições.



O Seletor de Nodo permite que o nodo de entrada superior seja transmitido ao nodo de saída quando o nodo de controle (linha pontilhada) estiver desativado. Já quando este nodo de controle for ativado o nodo de entrada inferior é transmitido ao nodo de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.

Note que o Seletor de Nodo, ao contrário de seu similar físico (interruptor elétrico com um pólo e duas posições), permite a transmissão de sinal apenas dos nodos de entrada (nodos à esquerda) para o nodo de saída (nodo à direita), como indicado pelas setas. Caso outro bloco ative o nodo de saída de um Seletor de Nodo, os nodos de entrada do bloco não serão acionados.

No exemplo a seguir, quando a entrada E1 do módulo de Expansão M1 (µDX210) é acionada, permite que o sinal presente na entrada E3 do mesmo módulo seja replicada na saída S1. Quando E1 está desativado o sinal presente na entrada E2 é replicado na mesma saída S1.

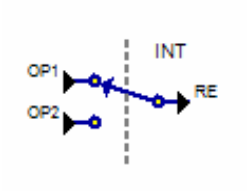


Exemplo de Programa Aplicativo: **Geral - Seletor de Nodo.dxg**

Veja também:
GERAL - Chave NA
GERAL - Chave NF
GERAL - Chave Inversora

GERAL - Seletor de Variável Inteira

O Seletor de Variável Inteira é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica a seta que aparece no centro tem dois contatos por onde "passará" a corrente (no caso, o valor de uma variável inteira) em determinadas condições.

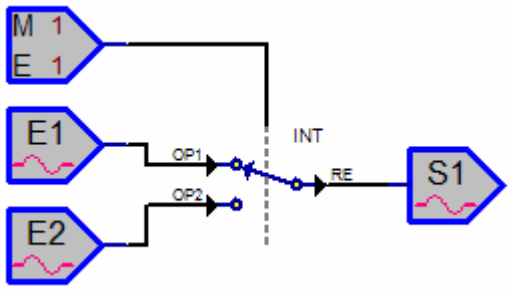


O Seletor de Variável Inteira permite que a variável inteira superior (OP1) seja transmitida à variável inteira de saída (RE) quando o nodo de controle (linha pontilhada) estiver desativado. Já quando este nodo de controle for ativado a variável de entrada inferior é transmitida à variável de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.

Note que o Seletor de Variável Inteira, ao contrário de seu similar físico (interruptor elétrico com um pólo e duas posições), permite a transmissão de sinal apenas das variáveis de entrada (conexões à esquerda) para a variável de saída (conexão à direita), como indicado pelas setas. Caso outro bloco modifique o valor da variável de saída de um Seletor de Variável Inteira, as variáveis de entrada do bloco não serão afetadas.

Note que variáveis inteiras podem assumir valores entre os seguintes limites: -32.768 e 32.767.

No exemplo a seguir, quando a entrada E1 do módulo de Expansão M1 (µDX210) é acionada, permite que o valor analógico presente na entrada analógica E2 do controlador µDX200 seja transmitido para a saída analógica S1. Quando E1 está desativado o sinal analógico presente na entrada E1 é replicado na mesma saída analógica S1.

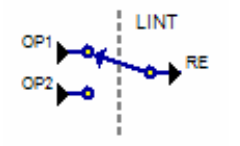


Exemplo de Programa Aplicativo: **Geral - Seletor de Variável Inteira.dxg**

Veja também:
GERAL - Seletor de Variável LongInt
GERAL - Seletor de Variável Word
GERAL - Seletor de Variável Real

GERAL - Seletor de Variável LongInt

O Seletor de Variável LongInt é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica a seta que aparece no centro tem dois contatos por onde "passará" a corrente (no caso, o valor de uma variável longint) em determinadas condições.

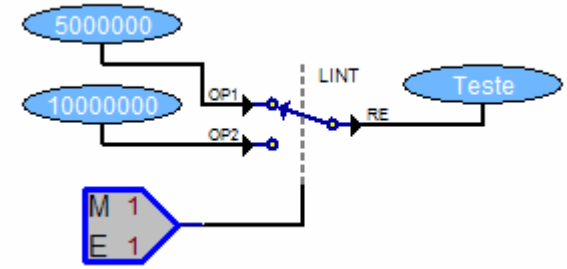


O Seletor de Variável LongInt permite que a variável longint superior (OP1) seja transmitida à variável longint de saída (RE) quando o nodo de controle (linha pontilhada) estiver desativado. Já quando este nodo de controle for ativado a variável de entrada inferior é transmitida à variável de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.

Note que o Seletor de Variável LongInt, ao contrário de seu similar físico (interruptor elétrico com um pólo e duas posições), permite a transmissão de sinal apenas das variáveis de entrada (conexões à esquerda) para a variável de saída (conexão à direita), como indicado pelas setas. Caso outro bloco modifique o valor da variável de saída de um Seletor de Variável LongInt, as variáveis de entrada do bloco não serão afetadas.

Note que variáveis longint podem assumir valores entre os seguintes limites: -2.147.483.648 e 2.147.483.647.

No exemplo a seguir, quando a entrada E1 do módulo de Expansão M1 (µDX210) é acionada, permite que a constante 10.000.000 seja transmitida para a variável longint Teste. Quando E1 está desativado a constante 5.000.000 é replicada na mesma variável Teste.

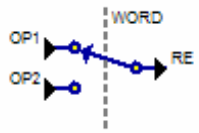


Exemplo de Programa Aplicativo: **Geral - Seletor de Variável LongInt.dxg**

Veja também:
GERAL - Seletor de Variável Inteira
GERAL - Seletor de Variável Word
GERAL - Seletor de Variável Real

GERAL - Seletor de Variável Word

O Seletor de Variável Word é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica a seta que aparece no centro tem dois contatos por onde "passará" a corrente (no caso, o valor de uma variável word) em determinadas condições.



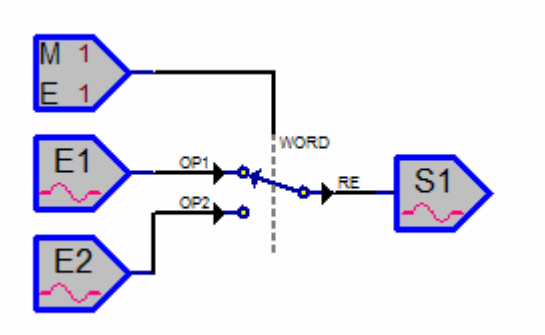
O Seletor de Variável Word permite que a variável word superior (OP1) seja transmitida à variável word de saída (RE) quando o nodo de controle (linha pontilhada) estiver desativado. Já quando este nodo de controle for ativado a variável de entrada inferior é transmitida à variável de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.

Note que o Seletor de Variável Word, ao contrário de seu similar físico (interruptor elétrico com um pólo e duas posições), permite a transmissão de sinal apenas das variáveis de entrada

(conexões à esquerda) para a variável de saída (conexão à direita), como indicado pelas setas. Caso outro bloco modifique o valor da variável de saída de um Seletor de Variável Word, as variáveis de entrada do bloco não serão afetadas.

Note que variáveis word podem assumir valores entre os seguintes limites: 0 e 65.535.

No exemplo a seguir, quando a entrada E1 do módulo de Expansão M1 (µDX210) é acionada, permite que o valor analógico presente na entrada analógica E2 do controlador µDX200 seja transmitido para a saída analógica S1. Quando E1 está desativado o sinal analógico presente na entrada E1 é replicado na mesma saída analógica S1. Como as entradas e saídas analógicas do µDX200 são representadas por variáveis inteiras a compilação do programa de exemplo gera três advertências devido a conexão de variável inteira com variável word.

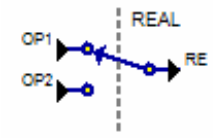


Exemplo de Programa Aplicativo: **Geral - Seletor de Variável Word.dxg**

Veja também:
GERAL - Seletor de Variável Inteira
GERAL - Seletor de Variável LongInt
GERAL - Seletor de Variável Real

GERAL - Seletor de Variável Real

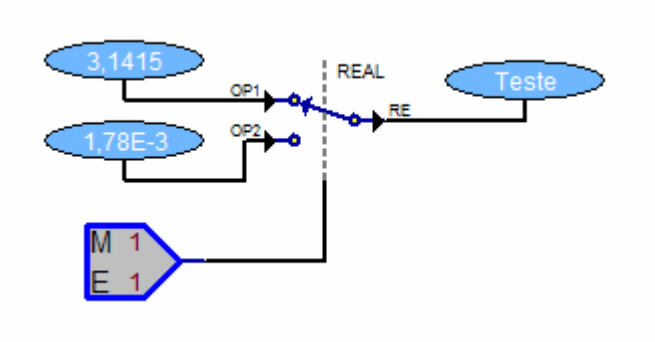
O Seletor de Variável Real é um bloco cuja função está descrita pela própria representação gráfica: como um interruptor de corrente elétrica a seta que aparece no centro tem dois contatos por onde "passará" a corrente (no caso, o valor de uma variável real) em determinadas condições.



O Seletor de Variável Real permite que a variável real superior (OP1) seja transmitida à variável real de saída (RE) quando o nodo de controle (linha pontilhada) estiver desativado. Já quando este nodo de controle for ativado a variável de entrada inferior é transmitida à variável de saída. Note que o nodo de controle possui dois pontos de conexão: um na parte superior e outro na parte inferior do bloco.

Note que o Seletor de Variável Real, ao contrário de seu similar físico (interruptor elétrico com um pólo e duas posições), permite a transmissão de sinal apenas das variáveis de entrada (conexões à esquerda) para a variável de saída (conexão à direita), como indicado pelas setas. Caso outro bloco modifique o valor da variável de saída de um Seletor de Variável Real, as variáveis de entrada do bloco não serão afetadas.

Note que variáveis Real podem assumir valores entre os seguintes limites: de -3,4E-38 a 3,4E38.



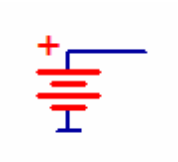
Exemplo de Programa Aplicativo: **Geral - Seletor de Variável Real.dxg**

No exemplo acima, conforme a entrada E1 do módulo de Expansão M1 (µDX210) é acionada ou não, a variável real Teste assume o valor 1,78E-3 ou o valor 3,1415.

Veja também:
GERAL - Seletor de Variável Inteira
GERAL - Seletor de Variável LongInt
GERAL - Seletor de Variável Word

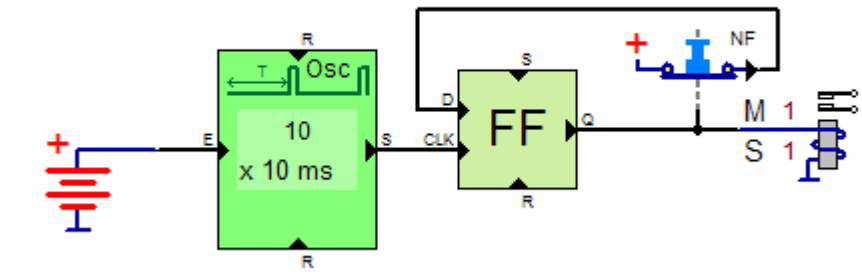
GERAL - Energia

Este bloco serve para indicar quais nodos devem ser forçadamente energizados (ligados). Este bloco é interpretado pelo µDX200 como sendo um forçamento do nodo para o estado ligado constantemente. Internamente, este nodo será o nodo número 1, que é mantido sempre ligado.



Assim, sempre que um bloco de instrução ou uma ligação que una um ou mais blocos de instrução precisar ficar continuamente no estado ligado, basta conectar a este bloco de energia.

Abaixo temos um exemplo de uso deste bloco para ativar constantemente o bloco de Oscilador, que gera uma onda quadrada (via FF) na saída S1 do módulo de Entradas/Saídas M1 (µDX210).



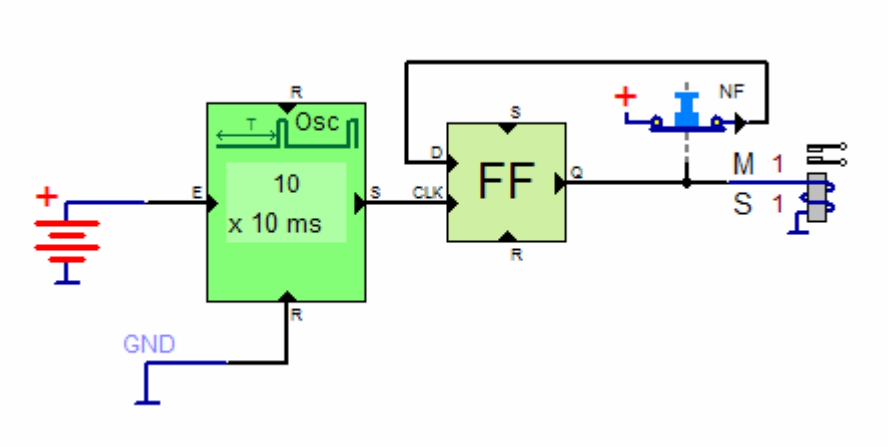
Exemplo de Programa Aplicativo: **Geral - Energia.dxg**

GERAL - Terra

Este bloco serve para indicar quais nodos devem ser forçadamente desenergizados (desligados). Este bloco é interpretado pelo µDX200 como sendo um forçamento do nodo para o estado desligado constantemente. Internamente, este nodo será o nodo número 0, que é mantido sempre desligado.

Assim, sempre que um bloco de instrução ou uma ligação que una um ou mais blocos de instrução precisar ficar continuamente no estado desligado, basta conectar a este bloco de energia.

Abaixo temos um exemplo de uso deste bloco para desativar constantemente o nodo de reset do bloco de Oscilador do exemplo anterior, que gera uma onda quadrada (via FF) na saída S1 do módulo de Entradas/Saídas M1 (µDX210). Note que poderíamos, simplesmente, deixar esta entrada de reset sem nenhuma conexão (qualquer nodo de entrada sem nenhuma conexão é mantido no estado zero).



Exemplo de Programa Aplicativo: **Geral - Terra.dxg**

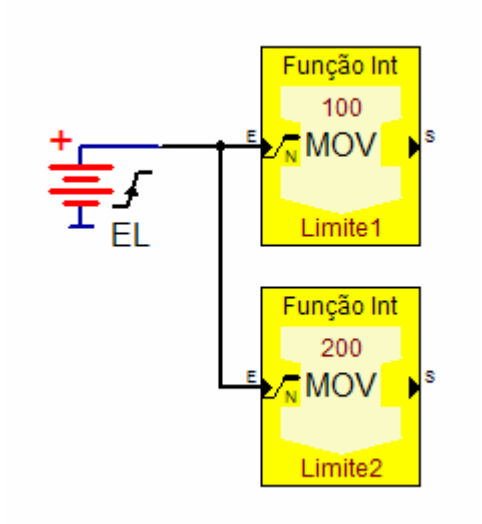
GERAL - Energia Liga

Este bloco produz um único pulso no nodo de saída, com duração de um ciclo de execução do programa aplicativo, quando o controlador programável µDX200 é energizado.

O Nodo EL produz o pulso no momento em que o suprimento de energia (fonte de alimentação) for ligado ao µDX200.

O bloco Nodo EL pode ser utilizado para inicializar variáveis no programa do µDX200. Note que o µDX200 inicia todas as variáveis com valor zero, mas muitas vezes isso não é conveniente. Neste caso podemos usar este bloco para gerar um pulso quando o CLP for energizado, e este pulso acionar blocos de atribuição de variáveis. Veja o exemplo abaixo, que inicializa as variáveis Limite1 e Limite2 com os valores 100 e 200, respectivamente. Note que o bloco EL inicializa as

variáveis sempre que o controlador é energizado. Caso se queira que isso ocorra quando o controlador iniciar a execução de um programa aplicativo e quando sofrer reset deve-se usar o bloco Reset.



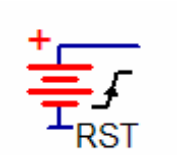
Exemplo de Programa Aplicativo: **Geral - Energia Liga.dxg**

Sempre que a energia elétrica do µDX200 for restabelecida o bloco Nodo Energia Liga (EL) irá gerar um pulso na entrada dos blocos MOV.

Veja também:
GERAL - Reset

GERAL - Reset

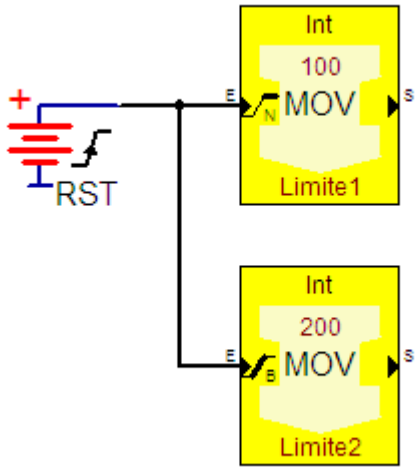
Este bloco produz um único pulso no nodo de saída, com duração de um ciclo de execução do programa aplicativo, quando o controlador programável µDX200 é reinicializado (reset). Isso ocorre, por exemplo, ao remeter um programa aplicativo para o controlador programável. Portanto, este bloco serve para inicializar variáveis do programa aplicativo.



Normalmente o µDX200 não deve reinicializar, pois com isso todas as variáveis são reinicializadas para zero, e o programa aplicativo é reiniciado. Mesmo com falta de alimentação elétrica o µDX200 não reinicializa, pois possui bateria interna que mantêm os dados do programa aplicativo e o relógio de tempo real operando. O programa aplicativo "congela" onde estiver e, quando a alimentação elétrica for restabelecida, ele retoma o processo.

O bloco Reset é muito utilizado para inicializar variáveis no programa do µDX200. Note que o µDX200 inicia todas as variáveis com valor zero, mas muitas vezes isso não é conveniente. Neste caso podemos usar este bloco para gerar um pulso quando o CLP for resetado, e este pulso acionar blocos de atribuição de variáveis. O reset ocorre ao mandar executar o programa aplicativo, de forma que as variáveis são inicializadas neste momento. Veja o exemplo abaixo, que inicializa as variáveis Limite1 e Limite2 com os valores 100 e 200, respectivamente. Note que o bloco Reset inicializa as variáveis sempre que o controlador sofre um reset, e isso é feito ao mandar executar um programa aplicativo. Existem dois tipos de reset do µDX200: o reset a que

nos referimos até o momento e o Hard Reset. O hard reset provoca uma reinicialização geral do µDX200. Ambos os tipos de reset estão disponíveis nos menus pop-down do Compilador para µDX200.

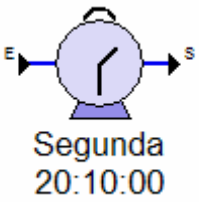


Exemplo de Programa Aplicativo: **Geral - Reset.dxg**

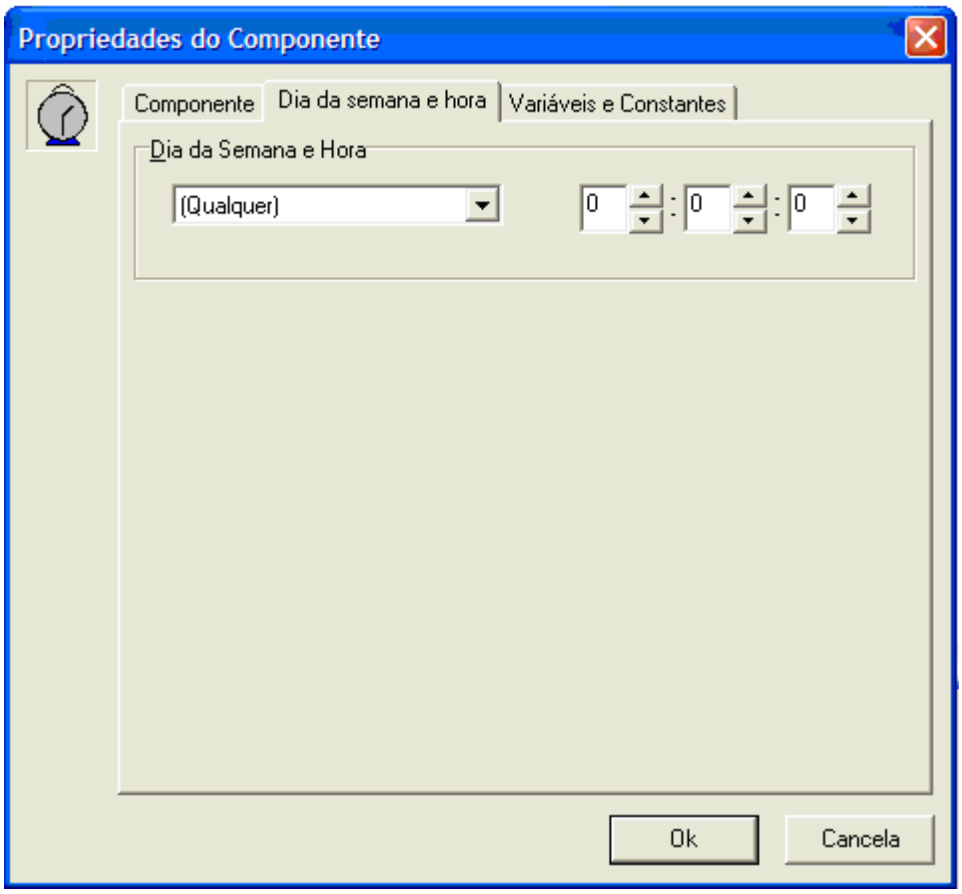
Veja também:
GERAL - Energia Liga

GERAL - Relógio

Este bloco liga o nodo de saída (S) quando o horário do relógio de tempo real do controlador programável µDX200 coincidir com o horário especificado no bloco. No desenho acima, por exemplo, isso acontece toda segunda-feira, às 20 horas e 10 minutos. Para que o nodo de saída (S) gere o pulso é necessário que o nodo de entrada (E) esteja energizado durante o horário programado.

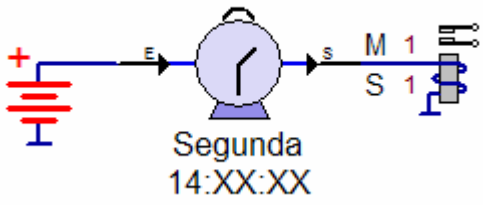


Note que o bloco permite tanto especificar um horário específico, como usar uma variável para especificar este horário. No caso de horários específicos, que é o caso mais comum, podemos selecionar o dia da semana (entre Segunda, Terça, Quarta, Quinta, Sexta, Sábado, ou Qualquer), a hora (entre 0 e 23, ou ainda XX), minuto (entre 0 e 59, ou ainda XX), e segundo (entre 0 e 59, ou ainda XX).



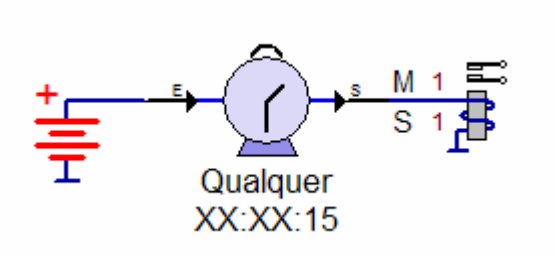
Caso seja especificado dia da semana qualquer, significa que todos os dias no horário especificado o bloco será acionado (desde que o nodo de entrada esteja energizado). Já se especificarmos hora = XX significa que qualquer hora é válida. O mesmo vale para os minutos e segundos. Por exemplo, um bloco de Relógio especificado para Dia da Semana = Qualquer, Hora = XX, Minuto = 10 e Segundo = 30 irá ligar todos os dias, a cada hora, quando forem 10 minutos e 30 segundos. Note que, neste caso, o bloco ficará ligado durante 1 segundo. Já se modificarmos este bloco especificando Segundo = XX o bloco irá ligar todos os dias, a cada hora, quando forem 10 minutos, e manterá o nodo de saída ligado durante 1 minuto (já que os segundos podem ser quaisquer - XX).

A seguir temos um exemplo que liga S1 do módulo de Entradas/Saídas M1 (µDX210) às 14 horas de Segunda-feira. Como tanto minuto quanto segundo podem ser quaisquer, o bloco mantêm S1 ligada durante 1 hora (até às 15 horas de Segunda).



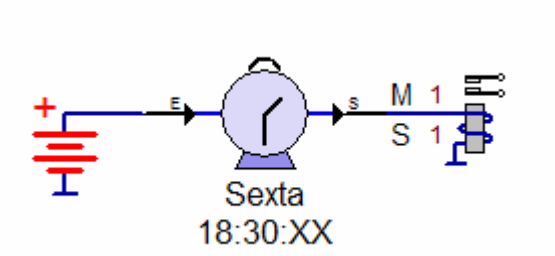
Exemplo de Programa Aplicativo: **Geral - Relógio.dxg**

Já no exemplo abaixo, a saída S1 é acionada a cada minuto, quando o relógio de tempo real do µDX200 marca 15 segundos, durante 1 segundo. Como Dia da Semana, Hora e Minuto podem ser quaisquer, o bloco energiza a cada minuto:



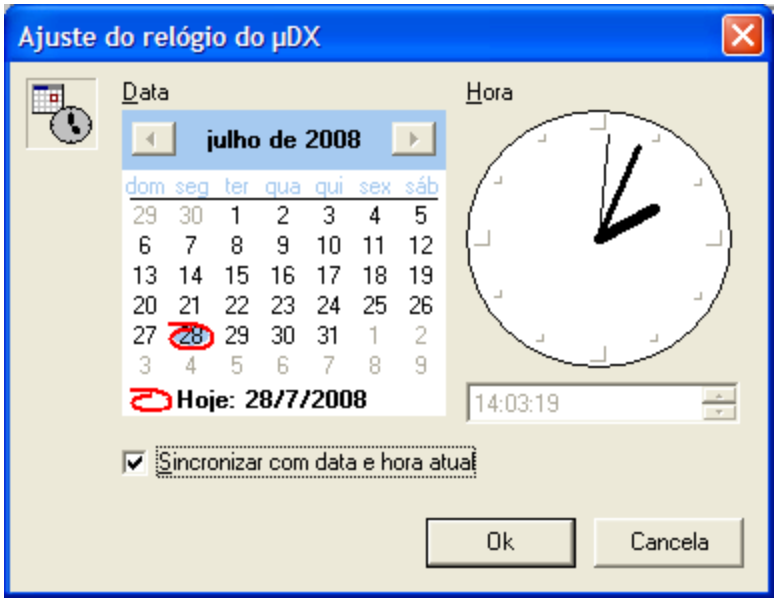
Exemplo de Programa Aplicativo: **Geral - Relógio 2.dxg**

Por fim, o próximo exemplo liga a saída S1 durante 1 minuto (já que Segundo = XX) apenas às Sextas-feiras, 18:30:

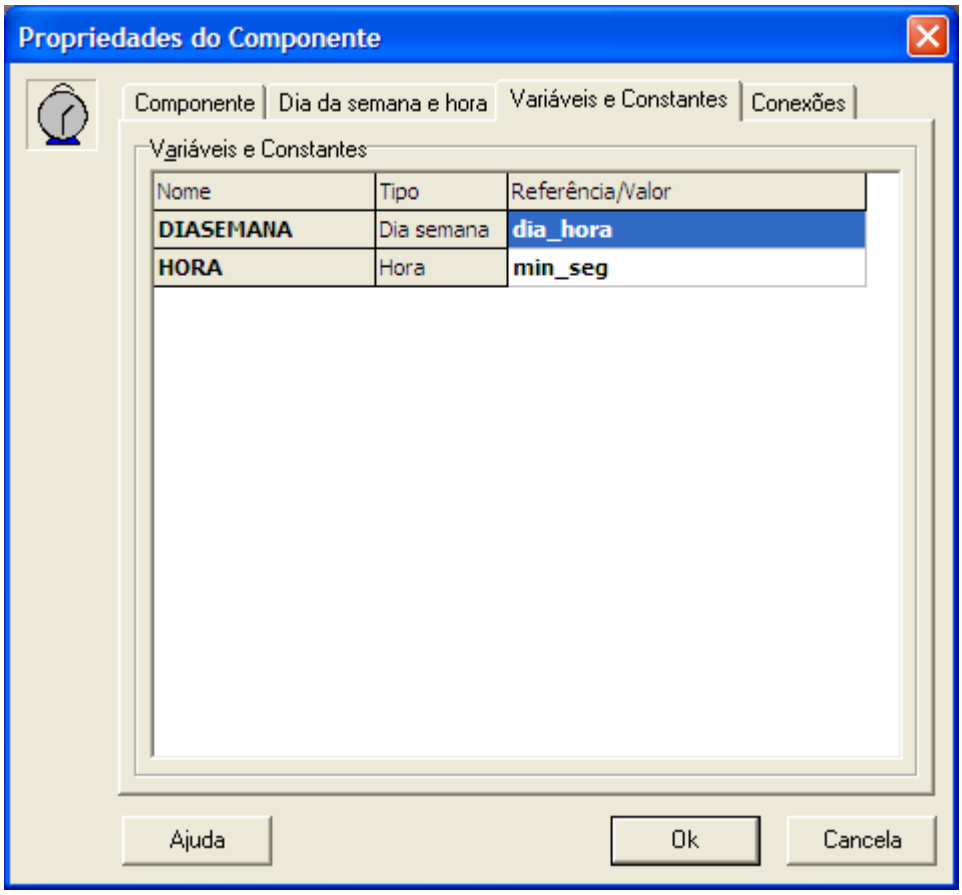


Exemplo de Programa Aplicativo: **Geral - Relógio 3.dxg**

Note que, evidentemente, o relógio de tempo real do controlador µDX200 deve estar inicializado com o horário e data correta para o funcionamento adequado deste bloco. Para acertar o relógio do µDX200 basta acessar o comando de **Acertar Relógio...** existente no menu pop-down **µDX** do Compilador PG.



Já no caso de usar-se variáveis para especificar o horário do bloco Relógio basta indicar duas variáveis, uma para dia da semana e hora, e outra para minuto e segundo, na aba **Variáveis e Constantes**:



No caso foi usada a variável **dia_hora** e a variável **min_seg** para os parâmetros **DIASEMANA** e **HORA**, respectivamente. O formato destes parâmetros é o seguinte:

	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIASEMANA	x1	d	d	d	d	d	d	d	x2	h	h	h	h	h	h	h
HORA	x3	m	m	m	m	m	m	m	x4	s	s	s	s	s	s	s

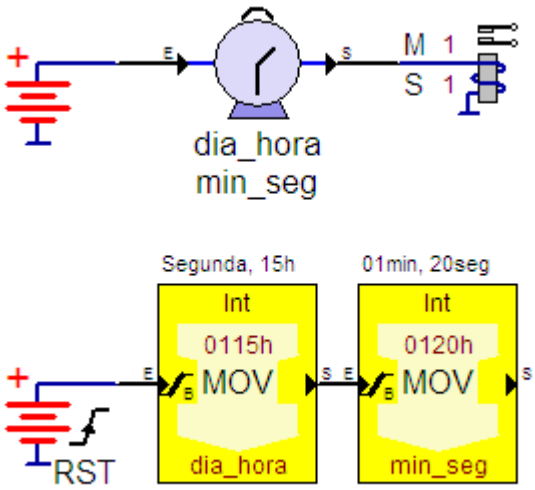
sendo:

x1 = despreza informação de dia da semana (qualquer dia da semana ativa o bloco Relógio)
ddddddd = dia da semana em BCD (1 = segunda, 2 = terça, 3 = quarta, 4 = quinta, 5 = sexta, 6 = sábado, 7 = domingo)
x2 = despreza informação de hora (qualquer hora ativa o bloco Relógio)
hhhhhhh = hora em BCD (00h a 23h)
x3 = despreza informação de minuto (qualquer minuto ativa o bloco Relógio)
mmmmmmm = minuto em BCD (00h a 59h)
x4 = despreza informação de segundo (qualquer segundo ativa o bloco Relógio)
sssssss = segundo em BCD (00h a 59h)

Os parâmetros são variáveis inteiras (16 bits), sendo que cada byte especifica um parâmetro. Caso o bit superior de cada byte seja ligado então aquele parâmetro é desprezado. **DIASEMANA** especifica o dia da semana e a hora, em formato BCD (binary-coded decimal). Por exemplo, ao fazer **DIASEMANA** = 0120h iremos especificar segunda-feira às 20 horas para o bloco Relógio. Já **DIASEMANA** = 8007h significa qualquer dia da semana às 7 horas. **HORA** especifica o minuto

e segundo em que o bloco irá ligar seu nodo de saída, em formato BCD. Portanto com **HORA** = 5240h, por exemplo, significa 52 minutos e 40 segundos.

Os parâmetros são especificados em BCD (binary-coded decimal). Assim, se escrevermos os valores em hexadecimal pode-se ler diretamente o valor das horas, minutos e segundos. A seguir temos um exemplo em que as variáveis do bloco **Relógio** são inicializadas de forma a especificar segunda-feira, 15h01m20s.

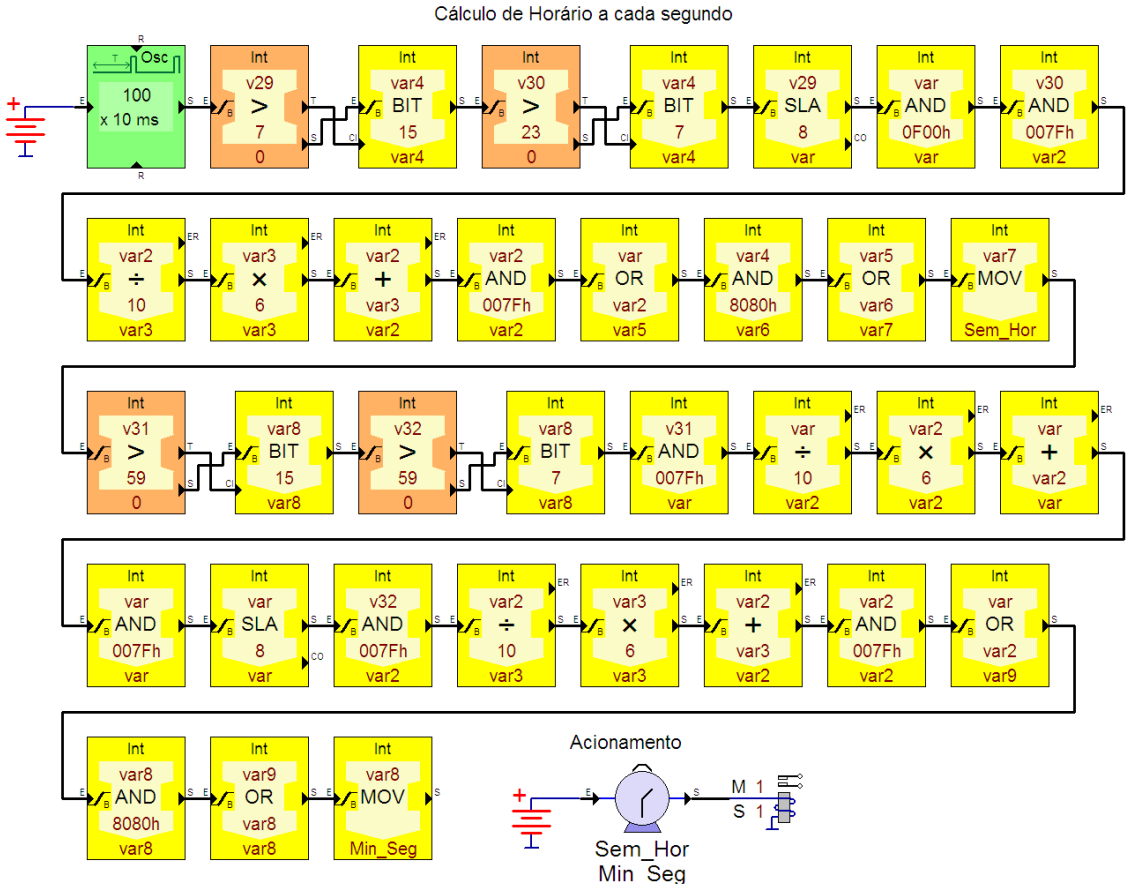


Exemplo de Programa Aplicativo: **Geral - Relógio 4.dxg**

Um outro exemplo é dado a seguir. Neste caso foram usadas variáveis para indicar o dia da semana e o horário em que o bloco relógio irá disparar. Foram usadas variáveis absolutas, de forma que um software supervisório externo possa programar o horário de disparo do bloco. Note que foram usadas as seguintes variáveis:

V29 = Dia da Semana (1 a 7 para segunda a domingo; maior que 7 qualquer)
V30 = Hora (0 a 23; maior que 23 qualquer)
V31 = Minuto (0 a 59; maior que 59 qualquer)
V32 = Segundo (0 a 59; maior que 59 qualquer)

Ou seja, se programarmos V29 com valor 2, V30 com valor 10, V31 com valor 15, e V32 com valor 60, o bloco Relógio irá disparar às 10:15 de toda terça-feira, durante um minuto (já que a informação de segundos foi especificada como qualquer).



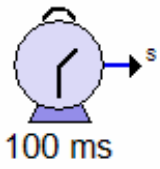
V29 = Dia da Semana (1 a 7 para segunda a domingo; maior que 7 qualquer)
V30 = Hora (0 a 23; maior que 23 qualquer)
V31 = Minuto (0 a 59; maior que 59 qualquer)
V32 = Segundo (0 a 59; maior que 59 qualquer)

Exemplo de Programa Aplicativo: **Geral - Relógio 5.dxg**

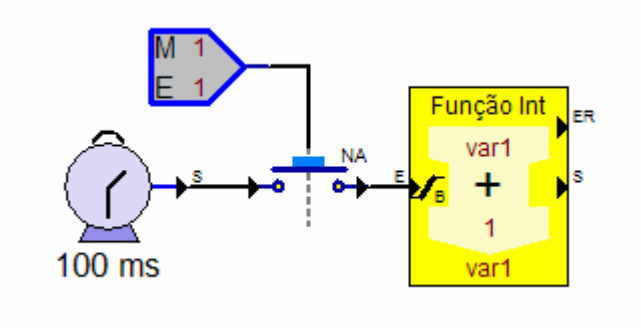
Veja também:
GERAL - Calendário
GERAL - Relógio 1 s
GERAL - Relógio 100 ms

GERAL - Relógio 100 ms

Este bloco liga o nodo de saída (S) a cada 100 ms (0,1 segundo), durante um ciclo de execução do programa aplicativo, síncrono com o relógio de tempo real do controlador µDX200. Note que este bloco não possui nodo de entrada e está sempre ativo.



Este bloco pode ser usado para gerar um contador de tempo sincronizado com o relógio de tempo real do µDX200. No exemplo abaixo, a variável inteira var1 é incrementada a cada 0,1s, desde que a entrada E1 do módulo M1 de Entradas/Saídas (µDX210) esteja ligada.

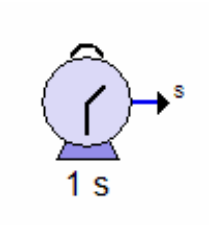


Exemplo de Programa Aplicativo: **Geral - Relógio 100 ms.dxg**

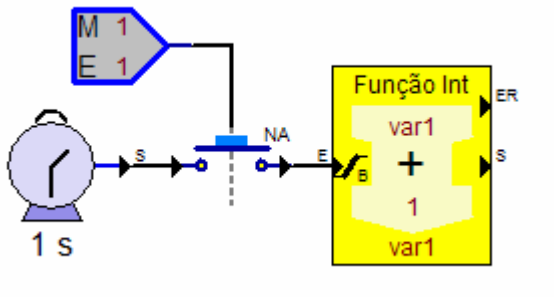
Veja também:
GERAL - Relógio 1 s
GERAL - Relógio

GERAL - Relógio 1 s

Este bloco liga o nodo de saída (S) a cada 1 s (1 segundo), durante um ciclo de execução do programa aplicativo, síncrono com o relógio de tempo real do controlador µDX200. Note que este bloco não possui nodo de entrada e está sempre ativo.



Este bloco pode ser usado para gerar um contador de tempo sincronizado com o relógio de tempo real do µDX200. No exemplo abaixo, a variável inteira var1 é incrementada a cada 1 segundo, desde que a entrada E1 do módulo M1 de Entradas/Saídas (µDX210) esteja ligada.

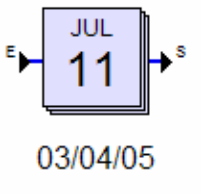


Exemplo de Programa Aplicativo: **Geral - Relógio 1 s.dxg**

Veja também:
GERAL - Relógio 100 ms
GERAL - Relógio

GERAL - Calendário

Este bloco liga o nodo de saída (S) quando a data do relógio de tempo real do controlador programável µDX200 coincidir com a data especificada no bloco. No desenho acima, por exemplo, isso acontece dia 3 de abril de 2005. Para que o nodo de saída (S) gere o pulso é necessário que o nodo de entrada (E) esteja energizado durante a data programada.

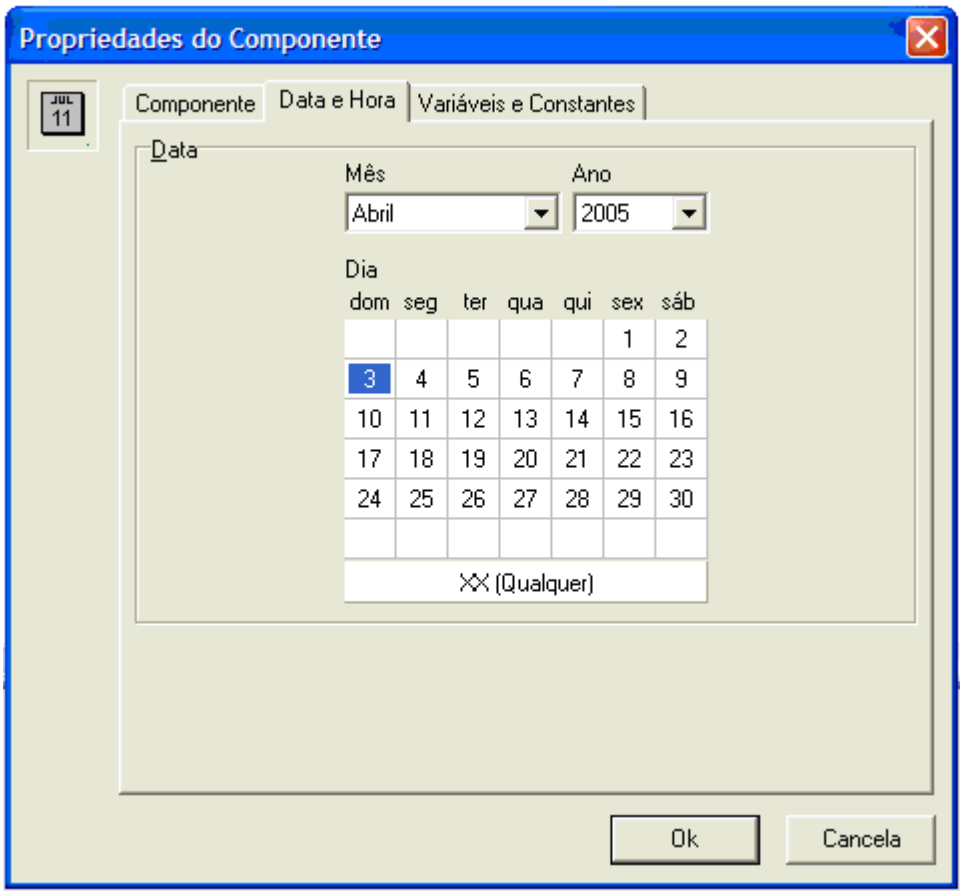


03/04/05

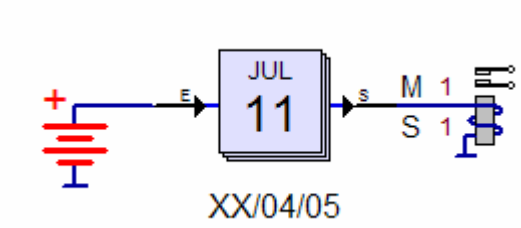
Obs.: Note que a data de 11 de julho que aparece na representação gráfica do bloco é apenas uma data genérica para representar o bloco. A data real do bloco é especificada abaixo desta representação gráfica (no caso acima, 03/04/05).

Note que o bloco permite tanto especificar uma data específica, como usar uma variável para especificar esta data. No caso de datas específicas, que é o caso mais comum, podemos selecionar o dia (entre 1 e 31, ou ainda XX), o mês (entre Janeiro e Dezembro, ou ainda XX), e ano (entre 2004 e 2020, ou ainda XX). Note que XX significa qualquer, ou seja, neste caso o dado não é relevante.

Caso seja especificado dia qualquer, significa que todos os dias no mês e ano especificados o bloco será acionado (desde que o nodo de entrada esteja energizado). Já se especificarmos mês = XX significa que qualquer mês é válido. O mesmo vale para o ano. Por exemplo, um bloco de Calendário especificado para Mês = XX, Dia = 15, Ano = XX irá ligar todos os meses no dia 15. Note que, neste caso, o bloco ficará ligado durante todo o dia. Já se modificarmos este bloco especificando Mês = Abril o bloco irá ligar apenas no dia 15 de abril a cada ano (já que Ano = XX), e manterá o nodo de saída ligado durante todo dia.

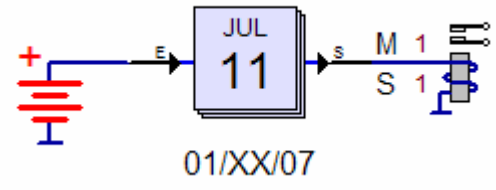


Abaixo temos um exemplo que liga S1 do módulo de Entradas/Saídas M1 (µDX210) em abril de 2005. Como dia pode ser qualquer, o bloco mantêm S1 ligada durante o mês de abril inteiro.



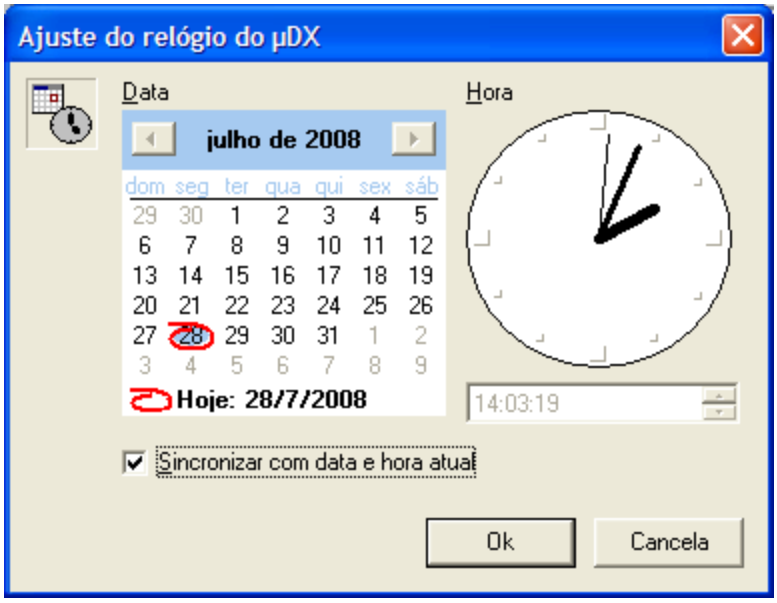
Exemplo de Programa Aplicativo: **Geral - Calendário.dxg**

Já no exemplo a seguir, a saída S1 é acionada a cada dia primeiro de cada mês, durante todo o ano de 2007. O bloco energiza o nodo de saída durante todo o dia primeiro de cada mês de 2007:



Exemplo de Programa Aplicativo: **Geral - Calendário 2.dxg**

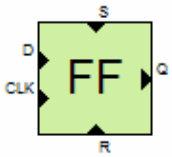
Note que, evidentemente, o relógio de tempo real do controlador µDX200 deve estar inicializado com o horário e data correta para o funcionamento adequado deste bloco. Para acertar o relógio do µDX200 basta acessar o comando de **Acertar Relógio** existente no menu pop-down **µDX** do Compilador PG.



Veja também:
GERAL - Relógio

GERAL - Flip-Flop

Este bloco implementa um flip-flop (biestável) tipo D (data). Este tipo de biestável transfere o estado do nodo de entrada (D) para o nodo de saída (Q) a cada borda de subida do nodo de clock (CLK).



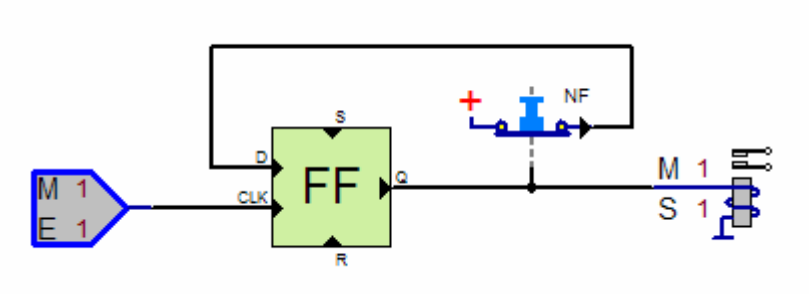
Além disso, existem nodos de Set (S) e Reset (R), que forçam a saída a nível 1 ou 0, respectivamente. Abaixo temos uma tabela de estados para o flip-flop tipo D:

D	CLK	S	R	Q
x	0	0	0	Q
x	1	0	0	Q
0	^	0	0	0
1	^	0	0	1
x	x	1	0	1
x	x	0	1	0
x	x	1	1	?

Note que foi usada a seguinte simbologia:

x Estado irrelevante
0 Estado zero
1 Estado um
^ Borda de subida
? Estado indeterminado
Q Estado inalterado

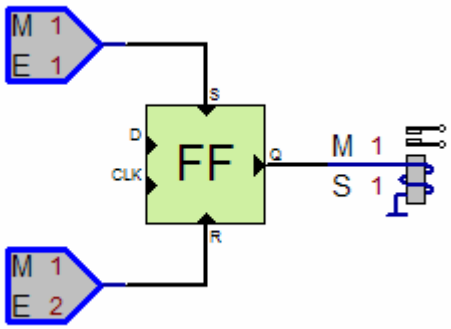
Este bloco pode ser usado tanto para gerar um comutador de estado (a cada pulso na entrada CLK troca o estado da saída), como para gerar uma botoeira liga-desliga (usando as entradas S e R). Veja os exemplos a seguir:



Exemplo de Programa Aplicativo: **Geral - Flip-Flop.dxg**

Neste exemplo a entrada D é ligada a saída Q invertida. Com isso, se a saída Q estiver desligada a entrada D estará ligada e vice-versa. Como a cada borda de subida de CLK a entrada D é transferida para a saída Q, resulta que a cada energização da entrada E1 do módulo de Expansão M1 a saída S1 deste módulo troca de estado.

No próximo exemplo não são usadas as entradas D e CLK, mas apenas S e R (ou seja, usamos um flip-flop RS). A entrada E1 do módulo de Expansão de Entradas/Saídas M1 (µDX210) liga a saída S1, enquanto a entrada E2 do mesmo módulo desliga esta saída:

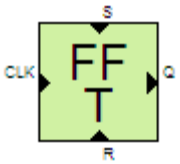


Exemplo de Programa Aplicativo: **Geral - Flip-Flop 2.dxg**

Veja também:
GERAL - Flip-Flop T

GERAL - Flip-Flop T

Este bloco implementa um flip-flop (biestável) tipo T (toggle). Este tipo de biestável troca o estado do nodo de saída (Q) a cada borda de subida do nodo de clock (CLK).



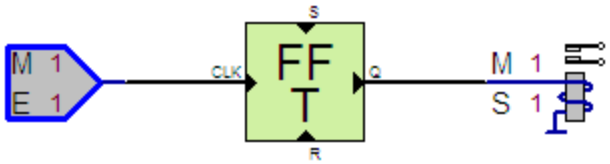
Além disso, existem nodos de Set (S) e Reset (R), que forçam a saída a nível 1 ou 0, respectivamente. Abaixo temos uma tabela de estados para o flip-flop tipo T:

CLK	S	R	Q
0	0	0	Q
1	0	0	Q
^	0	0	/Q
x	1	0	1
x	0	1	0
x	1	1	?

Note que foi usada a seguinte simbologia:

x Estado irrelevante
0 Estado zero
1 Estado um
^ Borda de subida
? Estado indeterminado
Q Estado inalterado
/Q Estado invertido

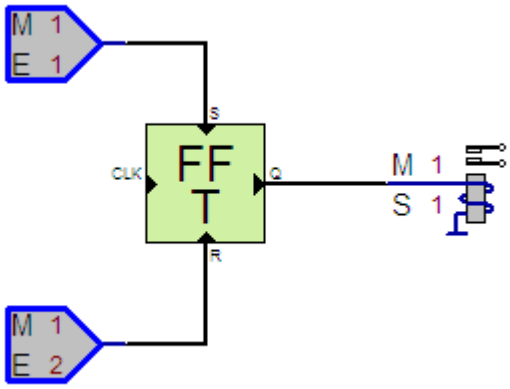
Este bloco pode ser usado tanto para gerar um comutador de estado (a cada pulso na entrada CLK troca o estado da saída), como para gerar uma botoeira liga-desliga (usando as entradas S e R). Veja os exemplos a seguir:



Exemplo de Programa Aplicativo: **Geral - Flip-Flop T.dxg**

Neste exemplo a cada energização da entrada E1 do módulo de Expansão M1 a saída S1 deste módulo troca de estado.

No próximo exemplo não é usada a entrada CLK, mas apenas S e R (ou seja, usamos um flip-flop RS). A entrada E1 do módulo de Expansão de Entradas/Saídas M1 (µDX210) liga a saída S1, enquanto a entrada E2 do mesmo módulo desliga esta saída:



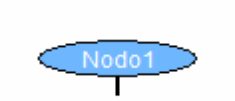
Exemplo de Programa Aplicativo: **Geral - Flip-Flop 2 T.dxg**

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.37 ou superior.

Veja também:
GERAL - Flip-Flop

GERAL - Nodo

Este bloco permite nomear um nodo, ou seja, uma variável binária (apenas dois estados, ligado ou desligado), ou atribuir o valor de uma constante a este nodo (1 para ligado ou 0 para desligado), ou ainda especificar um número de nodo específico para uma conexão binária.



Com isso, é possível conectar pontos distantes do programa, ou vários programas escritos em diferentes páginas do projeto. Além disso, usando-se a nomenclatura Nxxx, sendo xxx o número do nodo, é possível especificar um número de nodo determinado para a conexão. Isso é importante se esta conexão deve ser acessível para outros controladores µDX200 em rede DXNET.

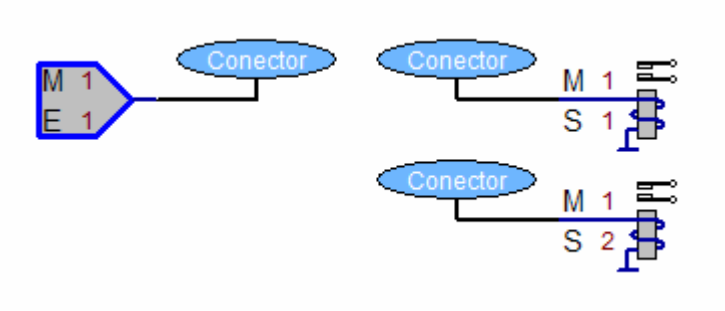
Lembre-se que os nodos de 0 a 31 (N0 a N31) são nodos de sistema no µDX200 e, portanto, possuem aplicações específicas, como detecção de erros. Assim, não podem ser usados como nodos de conexão no programa aplicativo.

Note que no menu principal do Editor PG existe uma entidade chamada Rótulo, cuja função também é conectar diferentes pontos do programa entre si.

Entretanto, o rótulo apenas conecta entre si todos os pontos com o mesmo número de rótulo (de 0 a 999), não atribuindo nenhum valor determinado ou nome para esta conexão. Além disso, o Rótulo pode conectar qualquer tipo de variável do µDX200 (nodo, inteiro, longint ou word), enquanto o bloco Nodo conecta apenas nodos.

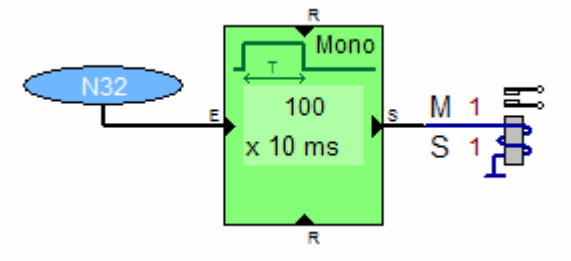
Os exemplos a seguir mostram possíveis aplicações para este bloco. O primeiro exemplo conecta todos os nodos chamados de "Conector" entre si. Note que ao acionar a entrada E1 do módulo

M1 de Expansão de Entradas/Saídas (µDX210) as saídas S1 e S2 deste mesmo módulo são acionadas.



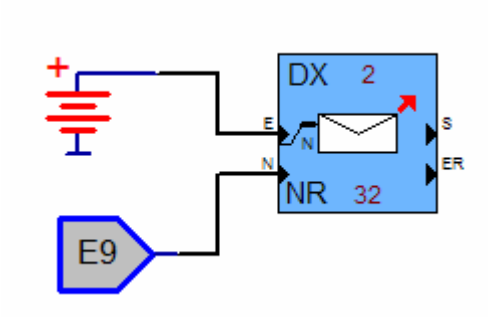
Exemplo de Programa Aplicativo: **Geral - Nodo.dxg**

O próximo exemplo especifica que a entrada do monoestável deve ser o nodo número 32 (primeiro nodo livre no µDX200, já que os nodos de 0 a 31 são nodos de sistema). Isso permite que outro µDX200 ligado a rede DXNET acione este nodo, simplesmente utilizando um bloco de comunicação DXNET Escreve Nodo endereçado para o nodo 32. Assim, o primeiro programa (com o monoestável) pode estar em um µDX200, com endereço DXNET 2, enquanto que o outro programa (com a entrada E9) pode estar em outro µDX200. Ao energizar a entrada E9 deste µDX200 acionamos por 1 segundo a saída S1 do módulo M1 (µDX210) ligado ao outro µDX200 (endereço DXNET 2).



µDX endereço DXNET 2

Exemplo de Programa Aplicativo: **Geral - Nodo 2.dxg**

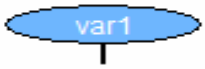


µDX endereço DXNET 1

Exemplo de Programa Aplicativo: **Geral - Nodo 3.dxg**

GERAL - Variável Inteira

Este bloco permite nomear uma conexão que transmite uma variável inteira (valor entre -32768 e 32767), ou atribuir o valor de uma constante a esta conexão, ou ainda especificar um número de variável específica para uma conexão inteira.



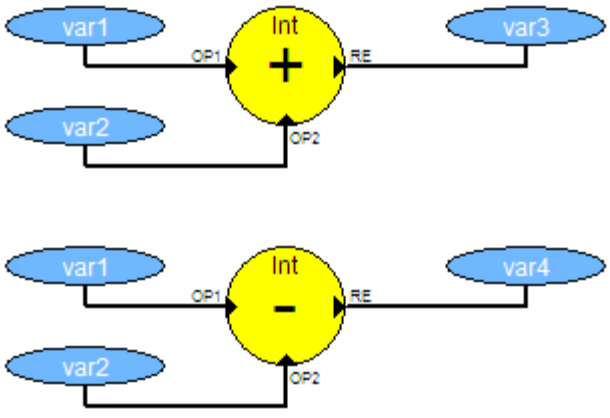
Com isso, é possível conectar pontos distantes do programa, ou vários programas escritos em diferentes páginas do projeto. Além disso, usando-se a nomenclatura Vxxx, sendo xxx o número da variável, é possível especificar um número de variável determinado para a conexão. Isso é importante se esta conexão inteira deve ser acessível para outros controladores µDX200 em rede DXNET.

Lembre-se que as variáveis de 0 a 17 (V0 a V17) são variáveis de sistema no µDX200 e, portanto, possuem aplicações específicas, como valor das entradas analógicas. Assim, não podem ser usados como variáveis do programa aplicativo.

Note que no menu principal do Editor PG existe uma entidade chamada Rótulo, cuja função também é conectar diferentes pontos do programa entre si.

Entretanto, o rótulo apenas conecta entre si todos os pontos com o mesmo número de rótulo (de 0 a 999), não atribuindo nenhum valor determinado ou nome para esta conexão. Além disso, o Rótulo pode conectar qualquer tipo de variável do µDX200 (nodo, inteiro, longint ou word), enquanto o bloco Variável Inteira conecta apenas variáveis inteiras.

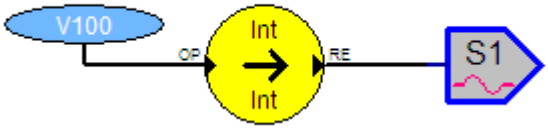
Os exemplos a seguir mostram possíveis aplicações para este bloco. O primeiro exemplo liga as variáveis var1 e var2 as entradas de um bloco de Soma Var, cujo resultado é atribuído à variável inteira var3. Também as mesmas variáveis var1 e var2 são utilizadas em um segundo bloco de Subtração Var, cujo resultado é atribuído a variável inteira var4.



Exemplo de Programa Aplicativo: **Geral - Variável Inteira.dxg**

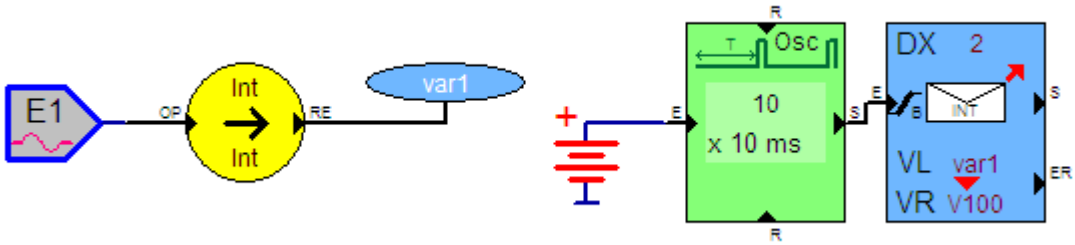
O próximo exemplo liga a variável V100 à saída analógica S1 do µDX200 com endereço DXNET 2. Já no µDX200 com endereço DXNET 1 ligamos um bloco de transmissão DXNET Escreve Variável transmitindo o valor da entrada analógica E1 do µDX200 endereço DXNET 1 para a variável V100 do µDX200 no endereço DXNET 2. Com isso, conectamos a entrada analógica E1 do µDX200 endereço 1 à saída analógica S1 do µDX200 endereço 2. A transmissão é feita a cada 100ms. Note que foi usado um bloco de Atribuição Inteira Var para conectar a variável V100 à

saída analógica S1 (não é possível conectar diretamente variáveis distintas no controlador). Da mesma forma, para conectar a entrada analógica E1 à variável var1 é preciso usar um bloco de Atribuição Inteira Var.



µDX endereço DXNET 2

Exemplo de Programa Aplicativo: **Geral - Variável Inteira 2.dxg**



µDX endereço DXNET 1

Exemplo de Programa Aplicativo: **Geral - Variável Inteira 3.dxg**

Veja também:
GERAL - Variável LongInt
GERAL - Variável Word
GERAL - Variável Real

GERAL - Variável LongInt

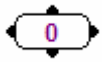
Este bloco permite nomear uma conexão que transmite uma variável longint (valor entre -2.147.483.648 e 2.147.483.647), ou atribuir o valor de uma constante a esta conexão, ou ainda especificar um número de variável específica para uma conexão longint.

Com isso, é possível conectar pontos distantes do programa, ou vários programas escritos em diferentes páginas do projeto. Além disso, usando-se a nomenclatura Vxxx, sendo xxx o número da variável, é possível especificar um número de variável determinado para a conexão. Isso é importante se esta conexão longint deve ser acessível para outros controladores µDX200 em rede DXNET.

Lembre-se que as variáveis de 0 a 17 (V0 a V17) são variáveis de sistema no µDX200 e, portanto, possuem aplicações específicas, como valor das entradas analógicas. Assim, não podem ser usados como variáveis do programa aplicativo. Além disso, variáveis longint são variáveis com 32 bits, que utilizam duas variáveis inteiras (16bits). Com isso, ao especificar em um bloco, por

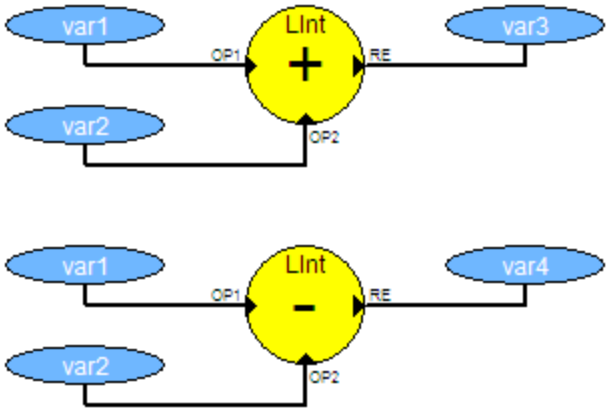
exemplo, a variável V18 como longint, na verdade o bloco está endereçando V18 e V19 (de forma a totalizar os 16 bits da variável longint).

Note que no menu principal do Editor PG existe uma entidade chamada Rótulo, cuja função também é conectar diferentes pontos do programa entre si.



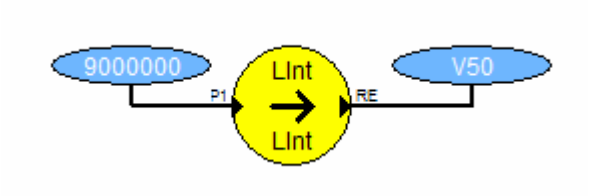
Entretanto, o rótulo apenas conecta entre si todos os pontos com o mesmo número de rótulo (de 0 a 999), não atribuindo nenhum valor determinado ou nome para esta conexão. Além disso, o Rótulo pode conectar qualquer tipo de variável do µDX200 (nodo, inteiro, longint ou word), enquanto o bloco Variável LongInt conecta apenas variáveis longint.

Os exemplos a seguir mostram possíveis aplicações para este bloco. O primeiro exemplo liga as variáveis var1 e var2 as entradas de um bloco de Soma Var, cujo resultado é atribuído à variável longint var3. Também as mesmas variáveis var1 e var2 são utilizadas em um segundo bloco de Subtração Var, cujo resultado é atribuído a variável longint var4.



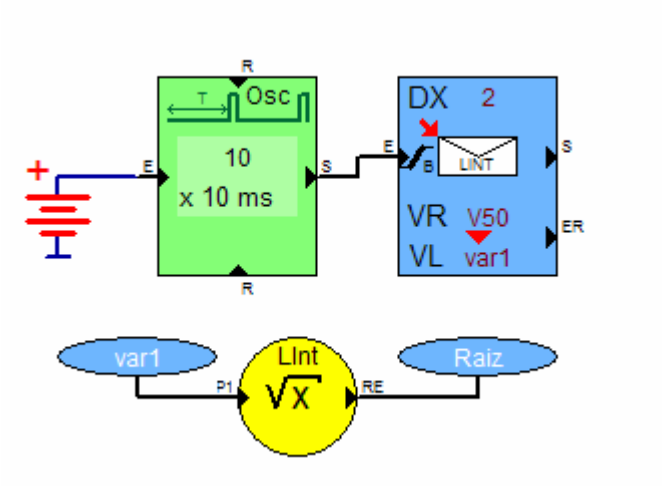
Exemplo de Programa Aplicativo: **Geral - Variável LongInt.dxg**

O próximo exemplo utiliza dois µDX200 ligados em rede DXNET. O µDX200 que ocupa o endereço DXNET 1 lê o valor da variável longint V50 do µDX200 endereço DXNET 2 e calcula a raiz quadrada desta variável, armazenando o resultado na variável inteira Raiz. No µDX200 endereço 2 atribuímos o valor 9.000.000 à variável V50 (resultando em uma raiz quadrada de 3000 em Raiz). A leitura de V50 é feita a cada 100ms.



µDX endereço DXNET 2

Exemplo de Programa Aplicativo: **Geral - Variável LongInt 2.dxg**



µDX endereço DXNET 1

Exemplo de Programa Aplicativo: **Geral - Variável LongInt 3.dxg**

Note que a saída do bloco de Raiz Quadrada LongInt Var resulta em variável inteira. Então, a variável Raiz é inteira e o bloco usado é de Variável Inteira, ao contrário dos demais blocos usados nos programas (que utilizam sempre o bloco de Variável LongInt).

Veja também:
GERAL - Variável Inteira
GERAL - Variável Word
GERAL - Variável Real

GERAL - Variável Word

Este bloco permite nomear uma conexão que transmite uma variável word (valor entre 0 e 65535), ou atribuir o valor de uma constante a esta conexão, ou ainda especificar um número de variável específica para uma conexão word.

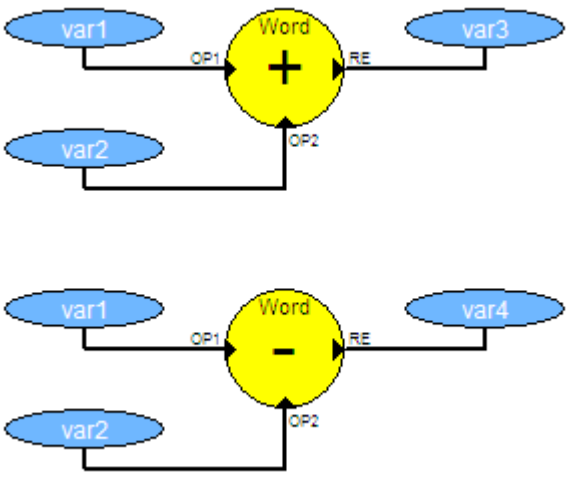
Com isso, é possível conectar pontos distantes do programa, ou vários programas escritos em diferentes páginas do projeto. Além disso, usando-se a nomenclatura Vxxx, sendo xxx o número da variável, é possível especificar um número de variável determinado para a conexão. Isso é importante se esta conexão word deve ser acessível para outros controladores µDX200 em rede DXNET.

Lembre-se que as variáveis de 0 a 17 (V0 a V17) são variáveis de sistema no µDX200 e, portanto, possuem aplicações específicas, como valor das entradas analógicas. Assim, não podem ser usados como variáveis do programa aplicativo.

Note que no menu principal do Editor PG existe uma entidade chamada Rótulo, cuja função também é conectar diferentes pontos do programa entre si.

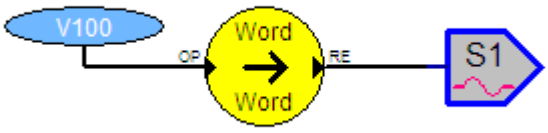
Entretanto, o rótulo apenas conecta entre si todos os pontos com o mesmo número de rótulo (de 0 a 999), não atribuindo nenhum valor determinado ou nome para esta conexão. Além disso, o Rótulo pode conectar qualquer tipo de variável do µDX200 (nodo, inteiro, longint ou word), enquanto o bloco Variável Word conecta apenas variáveis word.

Os exemplos a seguir mostram possíveis aplicações para este bloco. O primeiro exemplo liga as variáveis var1 e var2 as entradas de um bloco de Soma Var, cujo resultado é atribuído à variável word var3. Também as mesmas variáveis var1 e var2 são utilizadas em um segundo bloco de Subtração Var, cujo resultado é atribuído a variável word var4.



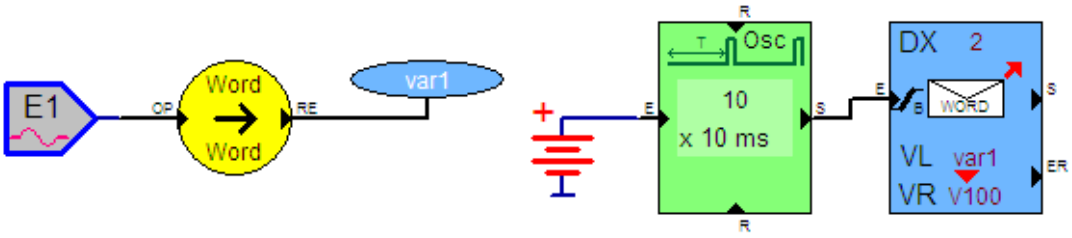
Exemplo de Programa Aplicativo: **Geral - Variável Word.dxg**

O próximo exemplo liga a variável V100 à saída analógica S1 do µDX200 com endereço DXNET 2. Já no µDX200 com endereço DXNET 1 ligamos um bloco de transmissão DXNET Escreve Variável transmitindo o valor da entrada analógica E1 do µDX200 endereço DXNET 1 para a variável V100 do µDX200 no endereço DXNET 2. Com isso, conectamos a entrada analógica E1 do µDX200 endereço 1 à saída analógica S1 do µDX200 endereço 2. Como as entradas e saídas analógicas do controlador µDX200 resultam em variáveis inteiras, ao conectá-las à variáveis word ocorre um aviso (warning) na compilação do programa. Este aviso tem o intuito de alertar para essa mistura de tipos de variáveis, e pode ser ignorado.



µDX endereço DXNET 2

Exemplo de Programa Aplicativo: **Geral - Variável Word 2.dxg**



µDX endereço DXNET 1

Exemplo de Programa Aplicativo: **Geral - Variável Word 3.dxg**

Veja também:
GERAL - Variável Inteira
GERAL - Variável LongInt
GERAL - Variável Real

GERAL - Variável Real

Este bloco permite nomear uma conexão que transmite uma variável real (valor entre -3,4E-38 e 3,4E38), ou atribuir o valor de uma constante a esta conexão, ou ainda especificar um número de variável específica para uma conexão real.

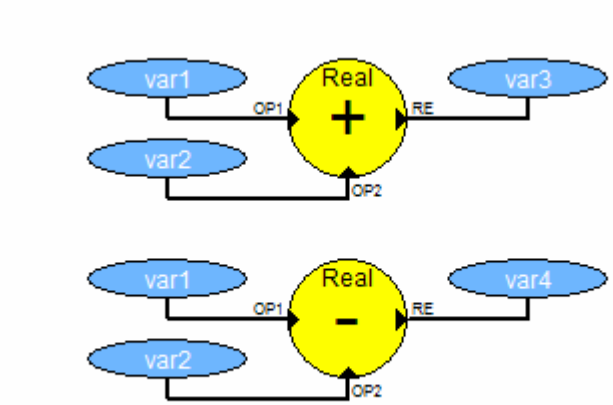
Com isso, é possível conectar pontos distantes do programa, ou vários programas escritos em diferentes páginas do projeto. Além disso, usando-se a nomenclatura Vxxx, sendo xxx o número da variável, é possível especificar um número de variável determinado para a conexão. Isso é importante se esta conexão real deve ser acessível para outros controladores µDX200 em rede DXNET.

Lembre-se que as variáveis de 0 a 17 (V0 a V17) são variáveis de sistema no µDX200 e, portanto, possuem aplicações específicas, como valor das entradas analógicas. Assim, não podem ser usados como variáveis do programa aplicativo.

Note que no menu principal do Editor PG existe uma entidade chamada Rótulo, cuja função também é conectar diferentes pontos do programa entre si.

Entretanto, o rótulo apenas conecta entre si todos os pontos com o mesmo número de rótulo (de 0 a 999), não atribuindo nenhum valor determinado ou nome para esta conexão. Além disso, o Rótulo pode conectar qualquer tipo de variável do µDX200 (nodo, inteiro, longint, word ou real), enquanto o bloco Variável Real conecta apenas variáveis real.

O exemplo a seguir mostra uma possível aplicação para este bloco. Ele liga as variáveis var1 e var2 as entradas de um bloco de Soma Var, cujo resultado é atribuído à variável real var3. Também as mesmas variáveis var1 e var2 são utilizadas em um segundo bloco de Subtração Var, cujo resultado é atribuído a variável real var4.



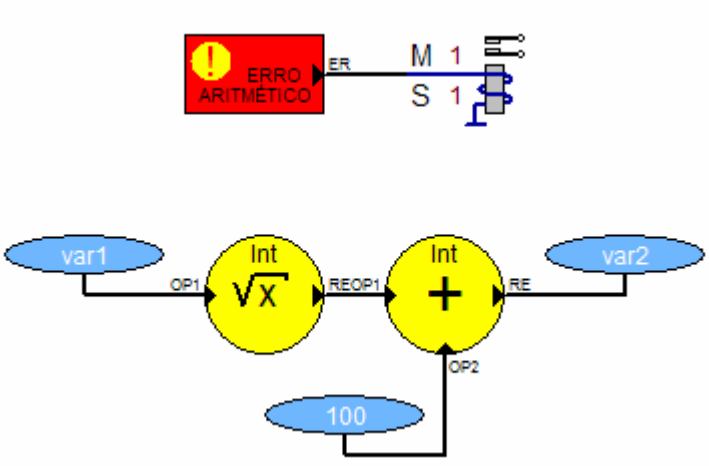
Exemplo de Programa Aplicativo: **Geral - Variável Real.dxg**

Veja também:
GERAL - Variável Inteira
GERAL - Variável LongInt
GERAL - Variável Word

GERAL - Erro Aritmético

Este bloco liga o nodo de saída (ER) sempre que for detectado um erro em um dos blocos de operações aritméticas utilizados no programa aplicativo. Embora os blocos aritméticos retangulares possuam nodos de saída para indicação de erro, isso não é verdadeiro para os blocos aritméticos circulares (que utilizam os operandos nas próprias conexões). Assim, este bloco pode indicar a ocorrência de erro aritmético em um destes blocos.

O exemplo a seguir liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro em um dos blocos aritméticos usados no programa. Isso pode ocorrer ao atribuir valores negativos para var1 (raiz quadrada de número negativo).



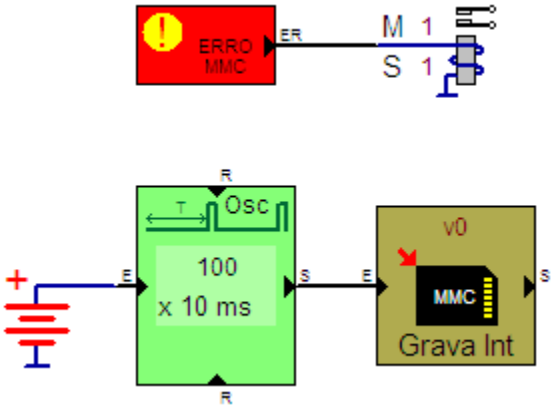
Exemplo de Programa Aplicativo: **Geral - Erro Aritmético.dxg**

Note que o programa calcula: var2 = sqrt(var1) + 100.

GERAL - Erro MMC

Este bloco liga o nodo de saída (ER) sempre que for detectado um erro em um dos blocos de tratamento de cartão MMC utilizados no programa aplicativo. Este bloco relata erro devido ao cartão MMC não ter sido previamente formatado no Compilador, não existir arquivo, ou o arquivo estar corrompido. Note que a detecção de cartão MMC não instalado é feita pelo bloco descrito a seguir - bloco MMC Ausente.

O exemplo a seguir liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, ou ainda arquivo corrompido). Note que o programa salva o valor lido na entrada analógica E1 do µDX200 (que corresponde a variável v0) a cada segundo.



Exemplo de Programa Aplicativo: **Geral - Erro MMC.dxg**

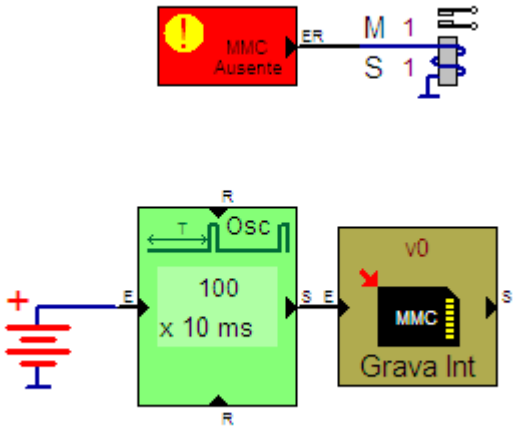
Veja também:
GERAL - MMC Ausente
GERAL - MMC Cheio

GERAL - MMC Ausente

Este bloco liga o nodo de saída (ER) sempre que for detectada a ausência de cartão MMC ou SD. Ele só é ativado se forem utilizados blocos com cartão MMC no programa aplicativo.

O exemplo a seguir liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando não existir cartão MMC instalado no µDX200. Note que o programa salva o valor lido na entrada analógica E1 do µDX200 (que corresponde a variável v0) a cada segundo.

Este bloco aciona o nodo de saída sempre que for detectada a ausência do cartão MMC (Multi Media Card) ou SD (Secure Digital).



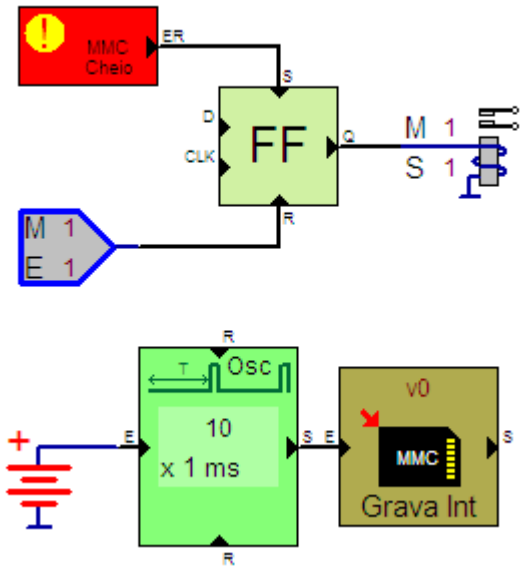
Exemplo de Programa Aplicativo: **Geral - MMC Ausente.dxg**

Veja também:
GERAL - Erro MMC
GERAL - MMC Cheio

GERAL - MMC Cheio

Este bloco liga o nodo de saída (ER) durante apenas um ciclo de execução do programa aplicativo quando for detectado que o cartão MMC (Multi Media Card) esgotou sua capacidade de armazenamento. Note que a capacidade de armazenamento do cartão MMC, além de depender da memória disponível no próprio cartão, Também depende da capacidade em Kbytes especificada na formatação do cartão via Compilador.

O exemplo a seguir liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando houver estouro de capacidade do cartão MMC instalado no µDX200. Note que o programa salva o valor lido na entrada analógica E1 do µDX200 (que corresponde a variável v0) a 10ms e, portanto, consome 200 bytes do cartão MMC a cada segundo. Como o bloco de MMC Cheio gera apenas um pulso quando o limite de capacidade do cartão é atingido, a sua saída foi ligada a entrada Set (S) do Flip-flop, de forma a memorizar a condição de erro. A entrada E1 do µDX210 é usada para desligar a saída S1, indicativa de MMC cheio.

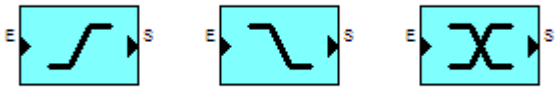


Exemplo de Programa Aplicativo: **Geral - MMC Cheio.dxg**

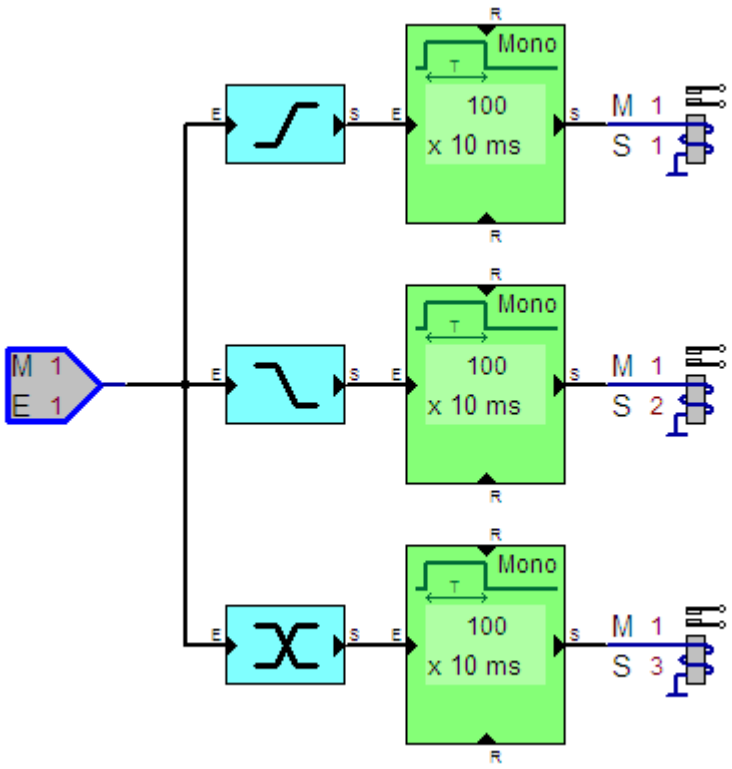
Veja também:
GERAL - Erro MMC
GERAL - MMC Ausente

GERAL - Borda

Estes blocos ligam o nodo de saída (S) durante apenas um ciclo de execução do programa aplicativo quando for detectado uma borda de subida no nodo de entrada (E), uma borda de descida no nodo de entrada(E), ou ainda tanto na borda de subida quanto na borda de descida.



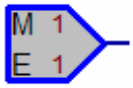
O programa a seguir exemplifica o uso desses blocos. Note que foram utilizados monoestáveis de 1 segundo nas saídas dos blocos, de forma a ser possível a visualização do acionamento das saídas, uma vez que esse acionamento é muito rápido (apenas um ciclo de execução do programa aplicativo):



Exemplo de Programa Aplicativo: **Teste_Borda.dxg**

ENTRADA/SAÍDA - Entrada Digital

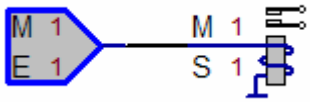
Este bloco acessa as entradas digitais dos módulos de Expansão de Entradas/Saídas do controlador µDX200. Estes módulos são designados µDX210 e cada módulo possui 8 entradas digitais e 8 saídas digitais. Como cada µDX200 permite acoplar até 32 módulos, pode-se chegar a 256 entradas digitais e 256 saídas digitais em um único controlador µDX200. Portanto, o operando Módulo pode ser uma constante de 1 a 32, enquanto o operando Entrada pode ser uma constante de 1 a 8.



A conexão de saída deste bloco é um nodo, ou seja, uma variável binária (apenas dois estados, ligado ou desligado).

As entradas digitais do módulo de Expansão µDX210 são optoacopladas, podendo ser ligadas diretamente a rede elétrica, desde que estejam selecionadas para entrada em alta tensão. Existem estrapes (jumpers) internos para selecionar o tipo de entrada (alta ou baixa tensão, DC ou AC) no µDX210.

O exemplo a seguir liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando a entrada E1 do mesmo módulo for energizada.

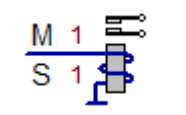


Exemplo de Programa Aplicativo: **Entrada_Saída - Entrada Digital.dxg**

Veja também:
ENTRADA/SAÍDA - Saída Digital

ENTRADA/SAÍDA - Saída Digital

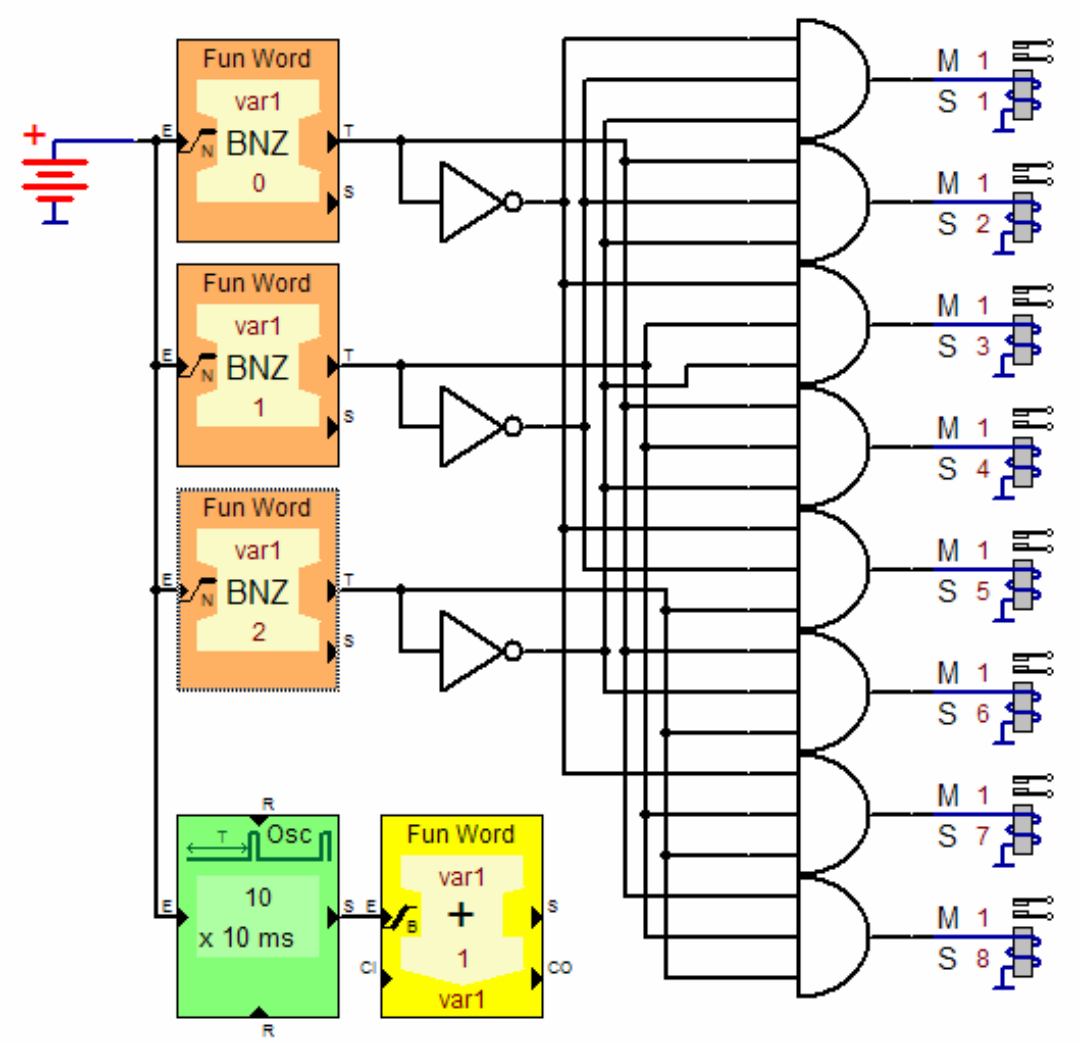
Este bloco acessa as saídas digitais dos módulos de Expansão de Entradas/Saídas do controlador µDX200. Estes módulos são designados µDX210 e cada módulo possui 8 entradas digitais e 8 saídas digitais. Como cada µDX200 permite acoplar até 32 módulos, pode-se chegar a 256 entradas digitais e 256 saídas digitais em um único controlador µDX200. Portanto, o operando Módulo pode ser uma constante de 1 a 32, enquanto o operando Saída pode ser uma constante de 1 a 8.



A conexão de entrada deste bloco é um nodo, ou seja, uma variável binária (apenas dois estados, ligado ou desligado).

As saídas digitais do módulo de Expansão µDX210 são implementadas com relés (contato seco), podendo ser ligadas diretamente a rede elétrica, desde que seja respeitado o limite de corrente nos contatos do relé (2A).

O exemplo a seguir liga as saídas S1 a S8 do módulo M1 de Expansão de Entradas/Saídas (µDX210) em seqüência. Note que a variável word var1 é incrementada a cada 100ms. Os blocos de comparação BNZ testam os três primeiros bits de var1, e as portas AND decodificam os 8 valores binários resultantes. Portanto, S1 é acionado quando estes bits são 000, S2 é acionado quando estes bits resultarem em 001, S3 quando for 010, e assim por diante, até S8 (111).

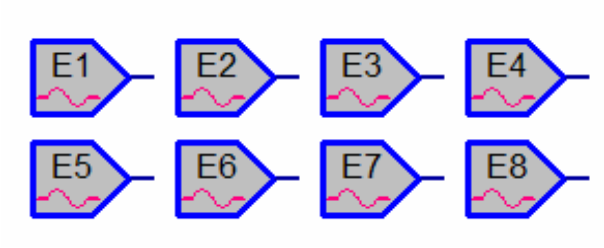


Exemplo de Programa Aplicativo: **Entrada_Saída - Saída Digital.dxg**

Veja também:
ENTRADA/SAÍDA - Entrada Digital

ENTRADA/SAÍDA - Entrada Analógica

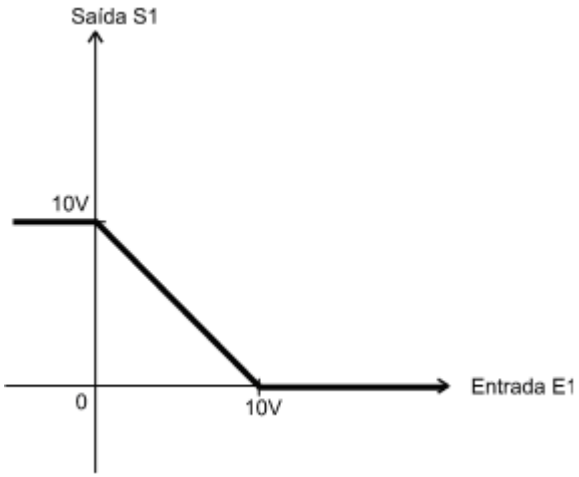
Este bloco acessa as entradas analógicas do Controlador Programável µDX200. Todas as 8 entradas disponíveis possuem resolução de 12 bits (4096 passos) e podem ser configuradas para sinais de 0-2,5V, 0-10V, ou ainda 0-20mA. Para diferenciar estes blocos dos referentes às entradas digitais foi incluído um ciclo de senóide na sua representação.



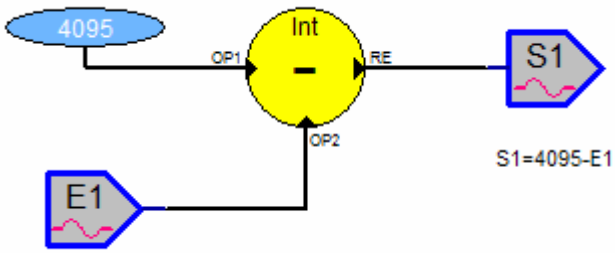
A conexão de saída deste bloco é uma variável inteira que assume valores entre 0 e 4095, conforme o valor do sinal aplicado na entrada analógica correspondente. As variáveis v0 a v7 estão associadas as entradas analógicas E1 a E8.

A resistência interna da entrada analógica depende da escala selecionada. Note que, além de existir uma seleção de escala no software Compilador, também é necessário selecionar corretamente os estrapes (jumpers) internos ao µDX200. Na escala de 0-2,5V a resistência interna de entrada é elevada (25MW), o que permite acoplar fontes de sinal de elevada impedância. Já na escala de 0-10V a resistência interna é 10KW. Por fim, na escala de 0-20mA a resistência interna é muito baixa (125W), o que facilita a fonte de corrente externa ligada a entrada analógica atingir o fundo de escala (20mA).

A seguir o exemplo mostra uma aplicação em que necessita-se que o sinal na entrada analógica (0-10V) seja invertido, ou seja, quando for aplicado 0V resulte em um sinal de saída de 10V. Já quando aplicados 10V resulta em sinal de saída de 0V. Se aplicados 5V na entrada analógica o sinal de saída será também de 5V. O gráfico da tensão de saída em função da tensão de entrada é o seguinte:



O programa para implementar este comportamento pode ser o abaixo. Note que 10V na entrada corresponde ao fundo de escala, ou seja, valor 4095.



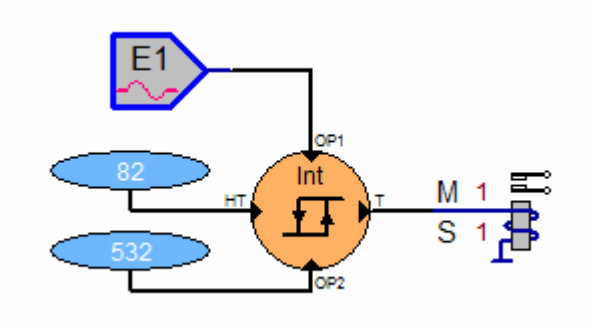
Exemplo de Programa Aplicativo: **Entrada_Saída - Entrada Analógica.dxg**

As entradas analógicas do µDX200 podem ser usadas como entradas digitais. Neste caso estipulamos limites para os quais o sinal de entrada é interpretado como nível 0, e limites para os quais ele é interpretado como nível 1. Por exemplo, níveis TTL possuem os seguintes limites:

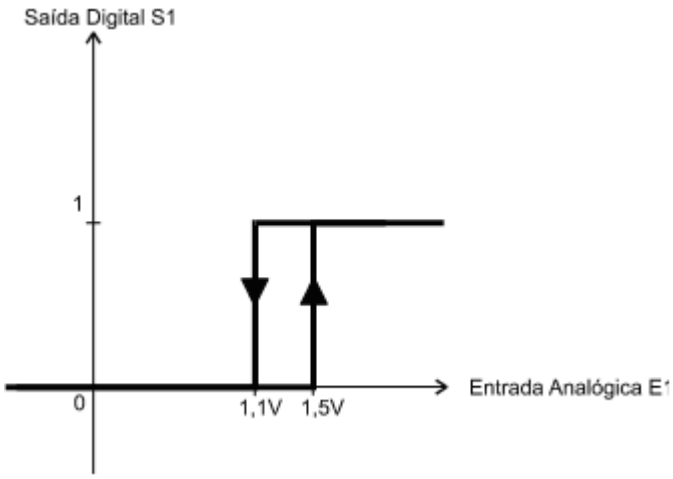
nível 0 de 0,0 a 0,8V, e nível 1 de 2,4 a 5,0V. Então, podemos estabelecer que nível 0 será quando o sinal de entrada for abaixo de 1,1V e nível 1 quando este sinal assumir valores acima de 1,5V. Ora, 1,1V na entrada analógica selecionada para escala de 0-10V corresponde a um valor de (4095*1,1)/10 = 450. Já 1,5V corresponde a (4095*1,5)/10 = 614.

O ponto médio de decisão será (450+614)/2 = 532, com uma histerese de 82 divisões para cada lado.

O programa aplicativo a seguir liga a saída S1 do módulo M1 de Expansão (µDX210) sempre que a tensão na entrada analógica atingir 1,5V ou mais, e desliga esta saída quando a tensão na entrada analógica atingir 1,1V ou menos.



Exemplo de Programa Aplicativo: **Entrada Saída - Entrada Analógica 2.dxg**



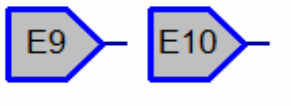
A conexão de saída deste bloco é um nodo, ou seja, uma variável binária (apenas dois estados, ligado ou desligado).

> ***ATENÇÃO**: As entradas analógicas do µDX200, fisicamente, **não são isoladas galvanicamente** (ao contrário das entradas digitais da Expansão µDX210). Portanto, todas as entradas analógicas estão referenciadas ao terminal Comum (C), não admitindo sinais com referências diferentes.*

Veja também:
ENTRADA/SAÍDA - Saída Analógica

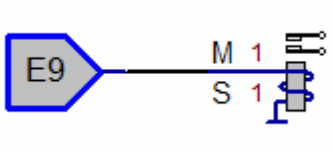
ENTRADA/SAÍDA - Entrada Digital E9 e E10

Estes blocos acessam as entradas digitais E9 e E10 do Controlador Programável µDX200. Note que estas entradas são de contagem rápida (até 8KHz), mas podem ser usadas também como entradas digitais comuns. A conexão de saída deste bloco é um nodo, ou seja, uma variável binária (apenas dois estados, ligado ou desligado).



As entradas digitais E9 e E10 não são optoacopladas, e portanto não podem ser ligadas diretamente a rede elétrica, até porque seu limite de tensão de entrada é 30V.

O exemplo a seguir liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando a entrada E9 do controlador µDX200 for energizada.



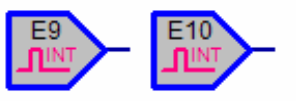
Exemplo de Programa Aplicativo: **Entrada_Saída - Entrada Digital E9 e E10.dxg**

***ATENÇÃO**: As entradas E9 e E10 do µDX200, fisicamente, **não são isoladas galvanicamente** (ao contrário das entradas da Expansão µDX210, que são isoladas). Portanto não conecte os fios da energia elétrica domiciliar diretamente em qualquer das entradas do µDX200.*

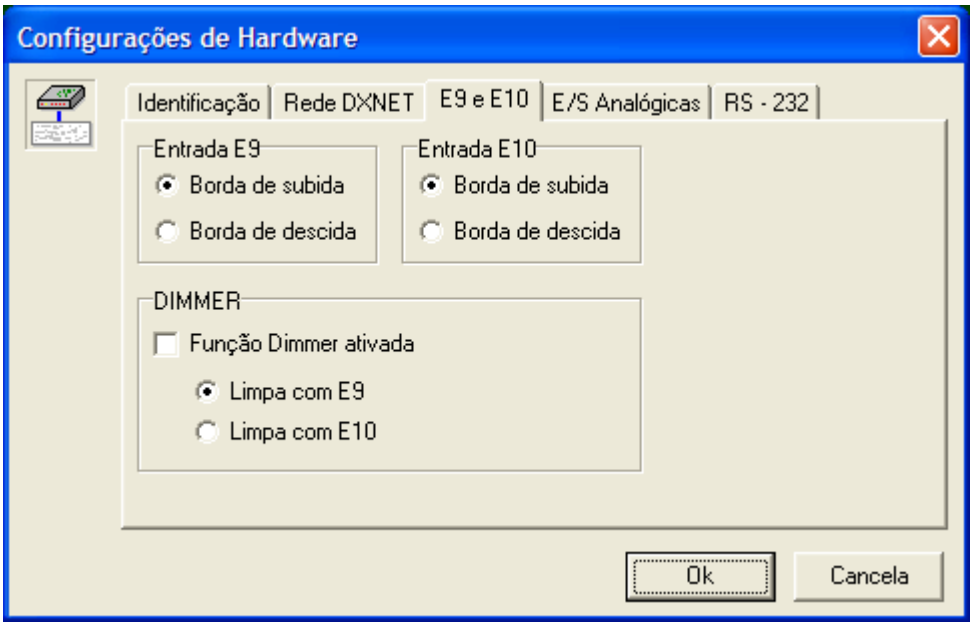
Veja também:
ENTRADA/SAÍDA - Interrupção E9 e E10

ENTRADA/SAÍDA - Interrupção E9 e E10

Estes blocos geram pulsos quando as entradas E9 ou E10 do controlador µDX200 trocam de estado. No Compilador é possível escolher se o pulso será gerado pela borda de subida ou pela borda de descida do sinal presente nas entradas E9 e E10 (via Configurações de Hardware do programa aplicativo). O pulso gerado tem duração de um ciclo de execução do programa aplicativo.

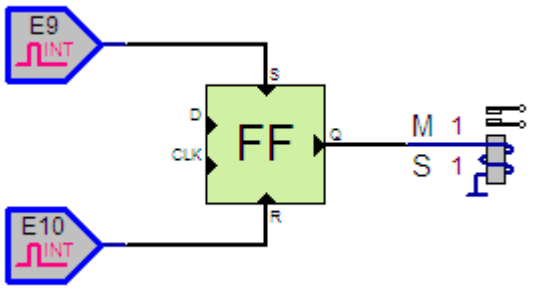


Estes blocos são usados para detecção rápida de eventos ou situações de alarme. Como estas entradas geram interrupção no CLP, elas não estão sujeitas aos tempos de varredura do programa aplicativo e, com isso, permitem detectar pulsos mais curtos que o tempo de ciclo do programa. A conexão de saída deste bloco é um nodo, ou seja, uma variável binária (apenas dois estados, ligado ou desligado).



As entradas digitais E9 e E10 não são optoacopladas, e portanto não podem ser ligadas diretamente a rede elétrica, até porque seu limite de tensão de entrada é 30V.

O exemplo a seguir liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando a entrada E9 do controlador µDX200 detectar um pulso, e desliga a saída S1 quando a entrada E10 detectar um pulso.



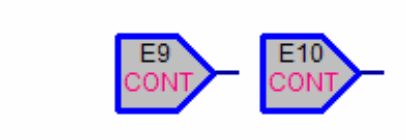
Exemplo de Programa Aplicativo: **Entrada Saída - Interrupção E9 e E10.dxg**

***ATENÇÃO**: As entradas E9 e E10 do µDX200, fisicamente, **não são isoladas galvanicamente** (ao contrário das entradas da Expansão µDX210, que são isoladas). Portanto não conecte os fios da energia elétrica domiciliar diretamente em qualquer das entradas do µDX200. Mesmo porque o limite de tensão destas entradas é 30V.*

Veja também:
ENTRADA/SAÍDA - Entrada Digital E9 e E10

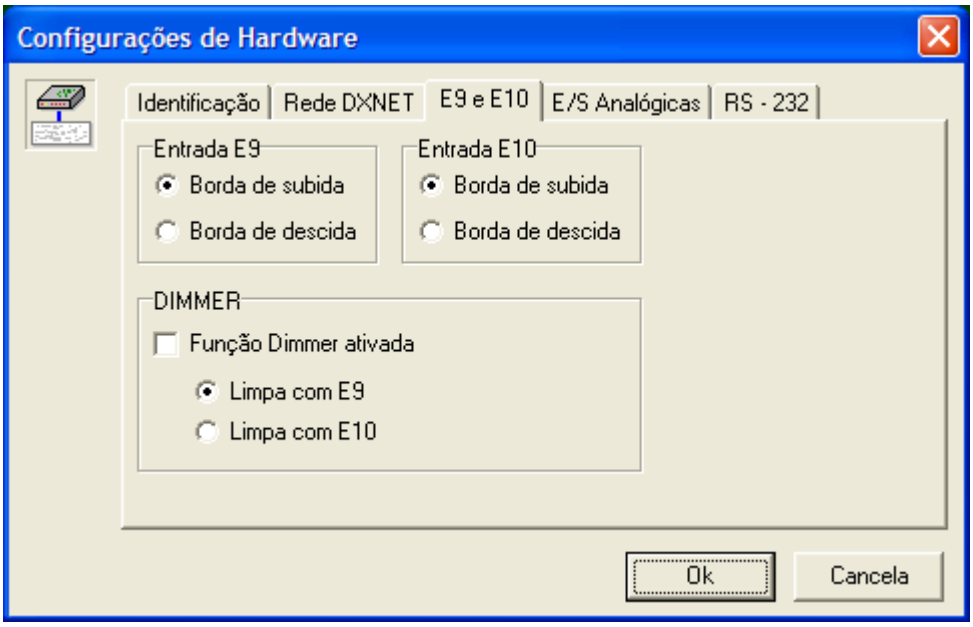
ENTRADA/SAÍDA - Contador E9 e E10

Estes blocos acessam variáveis inteiras que são incrementadas a cada borda de transição do sinal aplicado nas entradas E9 e E10 do controlador µDX200. Note que a variável v14 é usada como contador para a entrada E9 e a variável v15 é usada como contador para a entrada E10. Como estas entradas são entradas rápidas, estes contadores podem contabilizar pulsos a uma taxa de até 8.000 pulsos por segundo (8KHz).



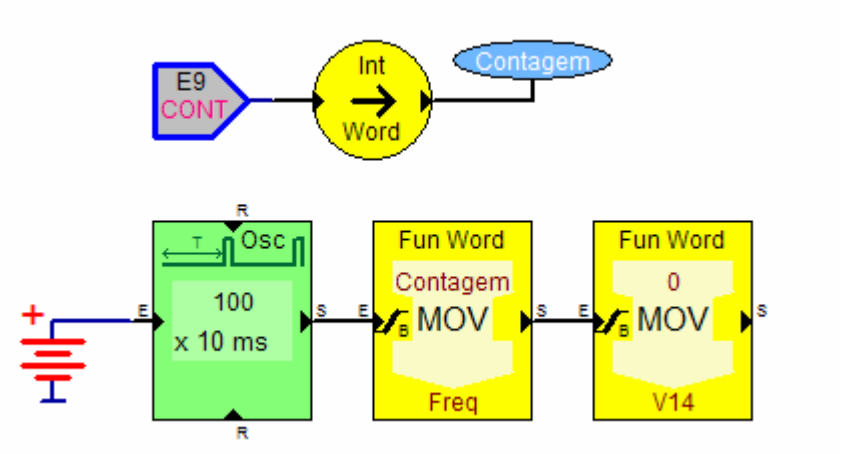
Estas entradas geram interrupção no CLP e não estão sujeitas aos tempos de varredura do programa aplicativo. Com isso, permitem detectar pulsos mais curtos que o tempo de ciclo do programa.

A conexão de saída deste bloco é uma variável inteira que assume valores entre -32768 e 32767, conforme o número de pulsos aplicado na entrada correspondente (E9 ou E10). Para aumentar esta faixa para 0 a 65535 (já que não faz sentido em uma contagem negativa, já que os contadores associados as entradas E9 e E10 são apenas incrementais) basta converter esta variável de saída do bloco Contador para variável word (ver bloco de Conversão Inteiro à Word Var).



No Compilador PG é possível escolher se o pulso será gerado pela borda de subida ou pela borda de descida do sinal presente nas entradas E9 e E10 (via Configuração de Hardware do programa aplicativo).

A seguir o exemplo mostra uma aplicação em que foi implementado um freqüencímetro na entrada E9. O freqüencímetro permite ler freqüências de 0 a 8KHz. A freqüência é atualizada a cada segundo na variável Freq. Note que logo após o valor de Contagem ser transferido para Freq o valor do contador associado a entrada E9 (v14) é zerado, iniciando nova contagem.

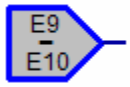


Exemplo de Programa Aplicativo: **Entrada Saída - Contador E9 e E10.dxg**

Veja também:
ENTRADA/SAÍDA - Contador E9-E10 (Quadratura)

ENTRADA/SAÍDA - Contador E9-E10 (Quadratura)

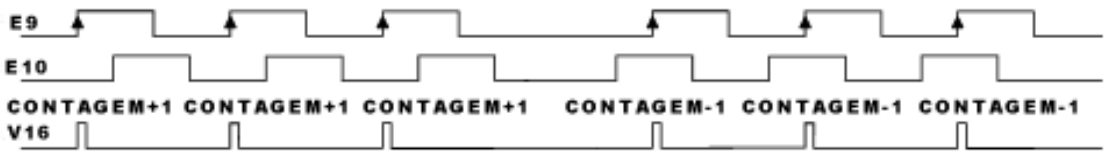
Este bloco acessa variável longint que é incrementada ou decrementada a cada transição do sinal na entrada E9 (borda de subida ou borda de descida, conforme o programado via Compilador). A variável longint é incrementada caso a entrada E10 esteja em nível lógico zero. Já se a entrada E10 estiver em nível lógico 1 esta variável é decrementada a cada transição de E9. Note que as variáveis v16 e v17 são usadas como contador incremental/decremental.



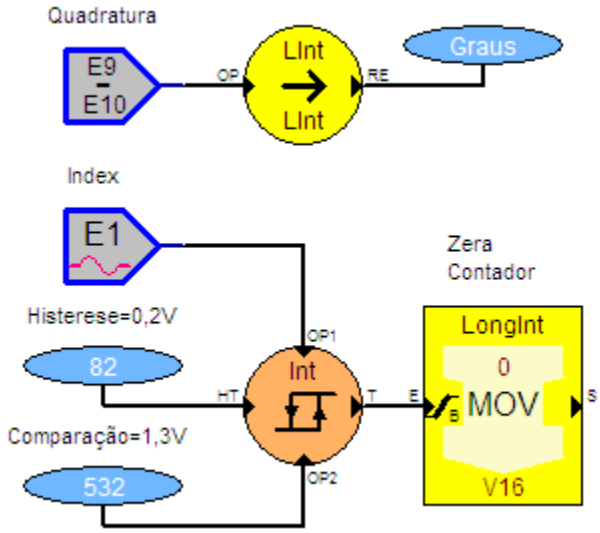
Como estas entradas são entradas rápidas, este contador pode contabilizar pulsos, no mínimo, a uma taxa de 8.000 pulsos por segundo (8KHz). Estas entradas geram interrupção no CLP e não estão sujeitas aos tempos de varredura do programa aplicativo. Com isso, permitem detectar pulsos mais curtos que o tempo de ciclo do programa.

A conexão de saída deste bloco é uma variável longint que assume valores entre -2.147.483.648 e 2.147.483.647, conforme o número de pulsos aplicado na entrada E9 e a operação especificada por E10 (incrementar/decrementar).

Este bloco pode ser usado para contabilizar os pulsos de um encoder por quadratura (de forma a poder discernir o sentido de rotação). Este tipo de sensor gera dois sinais quadrados defasados de 90°. Conforme qual dos sinais está adiantado em relação ao outro sinal a rotação é em um sentido ou em outro sentido, e a contagem deve ser incrementada ou decrementada para se saber qual a posição angular atual.



A seguir o exemplo mostra uma aplicação em que foi conectado um encoder por quadratura nas entradas E9 e E10, sendo ainda utilizada a entrada analógica E1 do µDX200 como uma entrada digital para sinais TTL (pontos de decisão entre 1,1V e 1,5V). A entrada E1 é conectada ao sinal de index do encoder. Este sinal indica que o encoder está na posição angular zero, evitando que haja acúmulo de erro na contabilização dos pulsos. Considerando que o encoder gere 360 pulsos por volta, a variável Graus indica qual a posição angular do eixo do encoder em graus. Note que Graus assume valores entre -359 e 359, conforme o sentido de rotação.

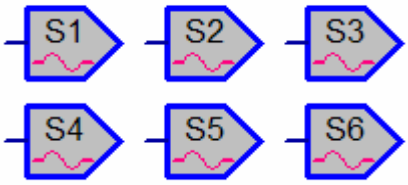


Exemplo de Programa Aplicativo: **Entrada_Saída - Contador E9-E10 (Quadratura).dxg**

Veja também:
ENTRADA/SAÍDA - Contador E9 e E10

ENTRADA/SAÍDA - Saída Analógica

Este bloco acessa as saídas analógicas do Controlador Programável µDX200. Todas as 6 saídas disponíveis possuem resolução de 12 bits (4096 passos) e podem ser configuradas para sinais de saída de 0-10V, 0-20mA, ou ainda modulação por largura de pulso (PWM). Para diferenciar estes blocos dos referentes às saídas digitais foi incluído um ciclo de senóide na sua representação.



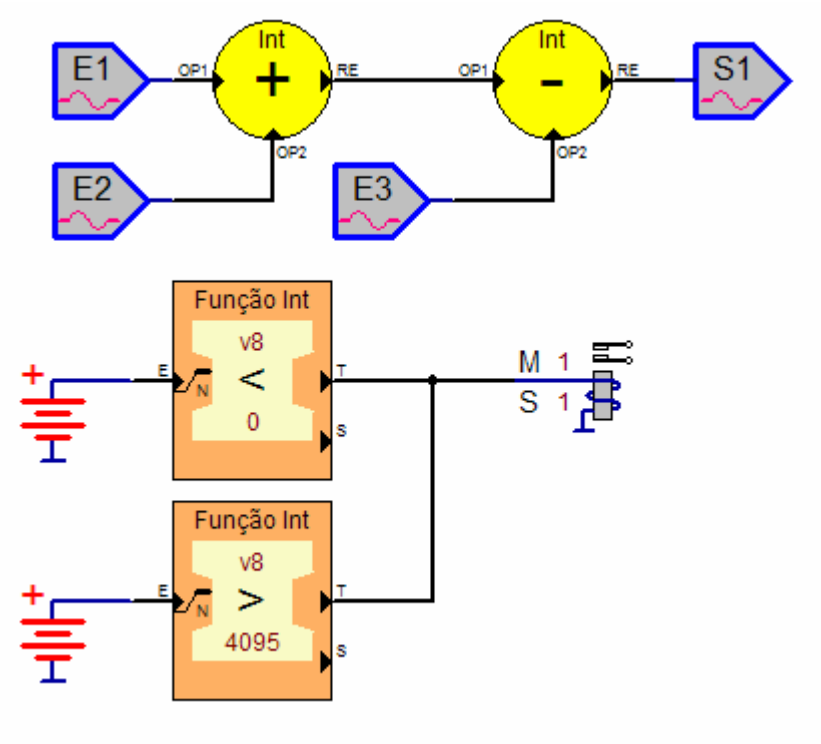
A conexão de entrada deste bloco é uma variável inteira que deve assumir valores entre 0 e 4096 (note que existe um passo a mais que nas entradas analógicas, que indicam valores de 0 a 4095), conforme o valor do sinal que se quer obter na saída analógica correspondente. As variáveis v8 a v13 estão associadas as saídas analógicas S1 a S8.

Note que, além de existir uma seleção de escala no software Compilador, também é necessário selecionar corretamente os estrapes (jumpers) internos ao µDX200. Note que a escala de 0-10V se comporta como uma fonte de tensão analógica dentro destes limites. Já na escala de 0-20mA a saída analógica se comporta como uma fonte de corrente. Se quisermos gerar um sinal entre 4

a 20mA, em vez de 0 a 20mA, basta excursionar a variável inteira de entrada do bloco de Saída Analógica entre 819 e 4096 (em vez de entre 0 e 4096).

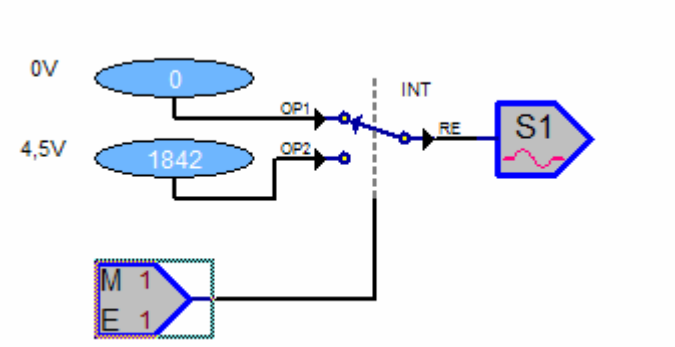
A seguir o exemplo mostra uma aplicação em que o sinal na saída analógica S1 seja o resultado da soma dos sinais analógicos aplicados nas entradas analógicas E1 e E2, subtraído do sinal analógico presente na entrada E3. Ou seja, o µDX200 irá efetuar uma operação de soma de dois sinais analógicos e, a seguir, uma subtração de outro sinal analógico. Obviamente o resultado deve estar dentro da faixa de 0 e 4095 (poderia ser até 4096, para ser exato) para resultar em um valor válido na saída analógica S1. Para indicar que o resultado excedeu estes limites foi acrescido ao programa os testes, que ligam a saída digital S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando isso ocorre. Lembre-se que v8 é a variável associada a saída analógica S1 do µDX200.

Estamos pressupondo que tanto as entradas analógicas E1, E2 e E3 quanto a saída analógica S1 estão programadas para escala de 0-10V. Com isso, se aplicarmos, por exemplo, 5V em E1, 2V em E2 e 3V em E3 resultará em S1 = E1+E2-E3 = 5+2-3 = 4V.



Exemplo de Programa Aplicativo: **Entrada_Saída - Saída Analógica.dxg**

As saídas analógicas do µDX200 podem ser usadas como saídas digitais. Neste caso estipulamos limites para os quais o sinal de saída é interpretado como nível 0, e limites para os quais ele é interpretado como nível 1. Por exemplo, níveis TTL possuem os seguintes limites: nível 0 de 0,0 a 0,8V, e nível 1 de 2,4 a 5,0V. Então, podemos estabelecer que nível 0 será quando o sinal de saída for 0V e nível 1 quando este sinal assumir valor de 4,5V. Ora, 0V na saída analógica selecionada para escala de 0-10V corresponde a um valor de zero. Já 4,5V corresponde a (4096*4,5)/10 = 1842. Então, basta colocar a variável de entrada do bloco nestes valores para fazer com que a saída analógica assuma valor 0 ou 1.



Exemplo de Programa Aplicativo: **Entrada_Saída - Saída Analógica 2.dxg**

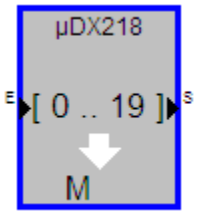
O programa aplicativo aciona a saída analógica S1 dentro dos limites de 0 e 4,5V conforme a entrada do módulo M1 de Expansão (µDX210).

> ***ATENÇÃO**: As saídas analógicas do µDX200, fisicamente, **não são isoladas galvanicamente** (ao contrário das saídas digitais da Expansão µDX210). Portanto, todas as saídas analógicas estão referenciadas ao terminal Comum (C), gerando todos os sinais com a mesma referência.*

Veja também:
ENTRADA/SAÍDA - Entrada Analógica

ENTRADA/SAÍDA - Inicializar µDX218

Transfere a tabela de inicialização longint apontada por **Tabela** para a Expansão Multi-sensor µDX218. O parâmetro **Endereço** indica qual das Expansões µDX218 será inicializada (endereço de 0 a 3).

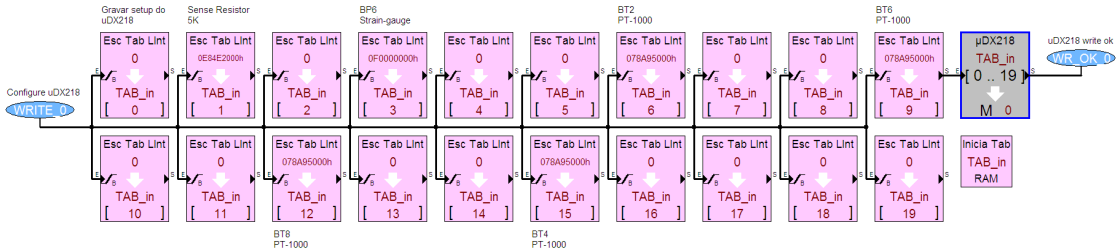


Cada posição da tabela corresponde à inicialização de uma das entradas analógicas da Expansão µDX218. Os bits desse valor longint determinam o tipo de sensor a ser lido na entrada, corrente de excitação (no caso de RTDs), linearização, etc. Os seguintes valores correspondem aos principais sensores lidos no µDX218:

Termopar tipo J (cold junction=CH20)	**00D300000h**	**Termopar tipo J (cold junction=CH10)**	**00AB00000h**
Termopar tipo K (cold junction=CH20)	**015300000h**	**Termopar tipo K (cold junction=CH10)**	**012B00000h**

Termopar tipo E (cold junction=CH20)	**01D300000h**	**Termopar tipo E (cold junction=CH10)**	**01AB00000h**
Termopar tipo N (cold junction=CH20)	**025300000h**	**Termopar tipo N (cold junction=CH10)**	**022B00000h**
Termopar tipo R (cold junction=CH20)	**02D300000h**	**Termopar tipo R (cold junction=CH10)**	**02AB00000h**
Termopar tipo S (cold junction=CH20)	**035300000h**	**Termopar tipo S (cold junction=CH10)**	**032B00000h**
Termopar tipo T (cold junction=CH20)	**03D300000h**	**Termopar tipo T (cold junction=CH10)**	**03AB00000h**
Termopar tipo B (cold junction=CH20)	**045300000h**	**Termopar tipo B (cold junction=CH10)**	**042B00000h**
RTD PT100 (american curve)	**060A95000h**	**RTD PT500 (american curve)**	**070A95000h**
RTD PT100 (european curve)	**060A94000h**	**RTD PT500 (european curve)**	**070A94000h**
RTD PT100 (japanese curve)	**060A96000h**	**RTD PT500 (japanese curve)**	**070A96000h**
RTD PT100 (ITS-90 curve)	**060A97000h**	**RTD PT500 (ITS-90 curve)**	**070A97000h**
RTD PT200 (american curve)	**068A95000h**	**RTD PT1000 (american curve)**	**078A95000h**
RTD PT200 (european curve)	**068A94000h**	**RTD PT1000 (european curve)**	**078A94000h**
RTD PT200 (japanese curve)	**068A96000h**	**RTD PT1000 (japanese curve)**	**078A96000h**
RTD PT200 (ITS-90 curve)	**068A97000h**	**RTD PT1000 (ITS-90 curve)**	**078A97000h**
ADC (differential)	**0F0000000h**	**ADC (singled-ended)**	**0F4000000h**
Sense Resistor (5,00K)	**0E84E2000h**	**Sense Resistor (10,0K)**	**0E89C4000h**
Diode (Ideality factor = 1,004)	**0E7501062h**		

Um exemplo tornará mais clara a aplicação desse bloco. Abaixo temos um programa que inicializa a Expansão µDX218 localizada no endereço 0 para que leia um sense resistor (interno ao µDX218) no canal CH1 e CH2, logo após uma entrada diferencial para strain-gauge no canal CH3 e CH4, e por fim leia 4 sensores RTD tipo PT1000 nas entradas CH5,6,7, CH8.9.10, CH11,12,13 e CH14,15,16. Para inicializar o µDX218 basta aplicar um pulso momentâneo no nodo WRITE_0.



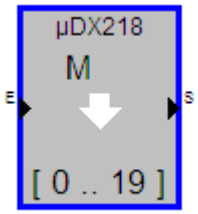
Note que foi usada uma tabela em RAM (TAB_In), inicializada com os parâmetros corretos para cada funcionalidade desejada para as entradas analógicas do µDX218. Por fim, vale comentar que é mais fácil usar as macros para µDX218, que facilitam bastante o uso do µDX218 e evitam a necessidade de iniciar uma tabela com valores hexadecimais (as macros fazem isso internamente, tornando o processo mais amigável para o usuário). Veja o capítulo Elaborando Programas - Macro - Macros que acompanham o PG - Automação Industrial.

> ***ATENÇÃO**: A expansão µDX218 não funciona em 12V. A tensão mínima exigida para seu funcionamento é 18V. Assim, para seu uso (uma vez que ela é alimentada pelo controlador µDX201) é preciso usar o µDX201 em 24V.*

Veja também:
ENTRADA/SAÍDA - Ler µDX218

ENTRADA/SAÍDA - Ler µDX218

Lê as entradas analógicas do µDX218 e transfere para a tabela longint apontada por **Tabela**. O parâmetro **Endereço** indica qual das Expansões µDX218 será inicializada (endereço de 0 a 3).



Cada posição da tabela irá corresponder a uma das 20 entradas analógicas da Expansão µDX218. Os valores lidos são longint (32 bits), sendo que os 24 bits menos significativos são o valor lido em si, enquanto os 8 bits mais significativos indicam erros:

D31 = Sensor Hard Fault - Erro no sensor. Por exemplo, rompimento de um sensor tipo RTD.

D30 = ADC Hard Fault - Erro na conversão A/D. Por exemplo, excesso de ruído na leitura A/D ou falha de leitura.

D29 = CJ Hard Fault - Junção Fria (Cold Junction) usada com termopares apresenta falha de operação.

D28 = CJ Soft Fault - Junção Fria (Cold Junction) usada com termopares apresenta leitura fora da faixa normal de operação.

D27 = Sensor Over Range - Leitura do sensor é maior que o normal.

D26 = Sensor Under Range - Leitura do sensor é menor que o normal.

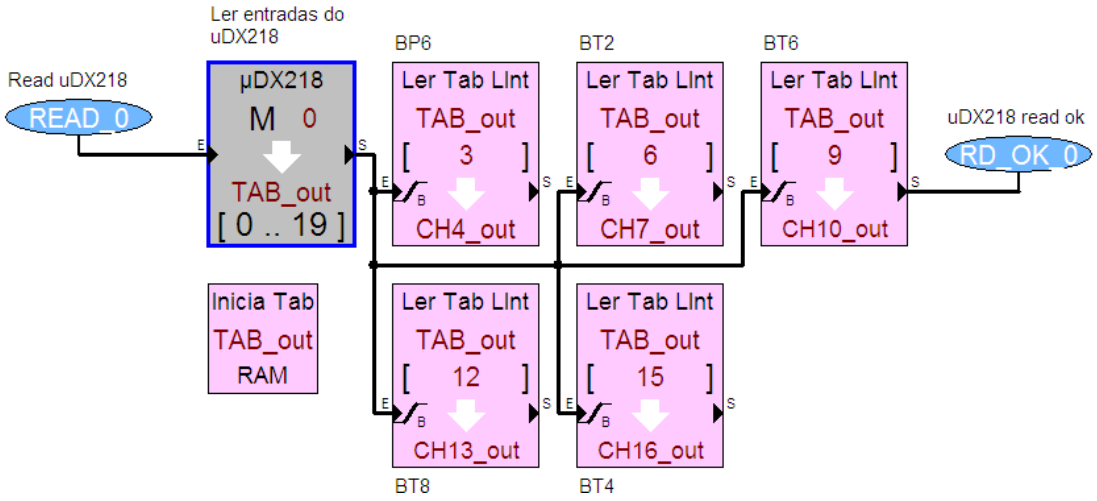
D25 = ADC Out of Range - Leitura A/D fora da faixa normal de operação (cerca de +/-1,25V).

D24 = Valid - Caso esse bit esteja ligado significa que a leitura é válida.

D23 = Signal - Sinal da leitura (para leituras menores que zero esse bit é ligado).

D22-D0 = Value - Leitura. Para temperatura a resolução (bit D0) é de 1/1024°C. Para tensão é de 477nV.

O exemplo a seguir mostra como ler as entradas que foram programadas pelo exemplo do bloco anterior (Inicializar µDX218) para leitura de um strain-gauge e quatro sensores RTD tipo PT1000:



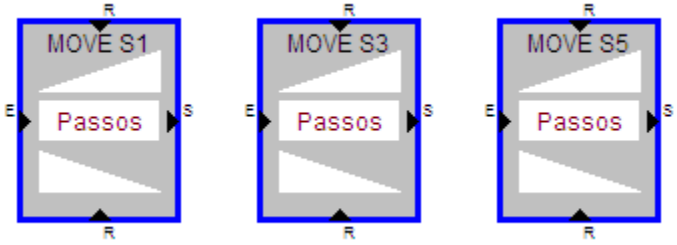
Note que é mais fácil usar as macros para µDX218, que facilitam bastante o uso do µDX218 e evitam a necessidade de usar uma tabela. Essas macros já disponibilizam o valor nas entradas analógicas do µDX218 em conexões longint. Veja o capítulo Elaborando Programas - Macro - Macros que acompanham o PG - Automação Industrial.

Veja também:
ENTRADA/SAÍDA - Inicializar µDX218

ENTRADA/SAÍDA - MOVE & RUN

Os blocos a seguir permitem controle de motor via geração de pulsos nas saídas analógicas.

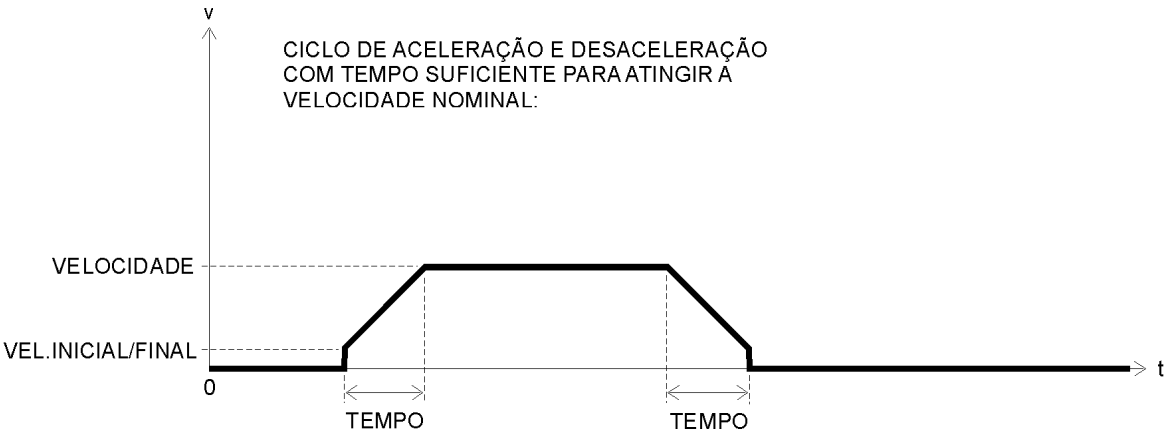
A função **MOVE** gera um número específico de pulsos na saída S1, S3 ou S5. Conforme o sinal da variável que indica o número de pulsos as saídas S2, S4 ou S6 são acionadas para reverter a rotação. O número de passos é especificado por uma variável longint, o que permite gerar pulsos nos dois sentidos de rotação, até valores acima de 2 bilhões (+/- 2.147.483.647 pulsos). O bloco especifica uma velocidade inicial/final e uma velocidade de regime (de 10 a 400, correspondendo a 250 Hz até 10 KHz, de 25 em 25 Hz, caso seja usada a resolução de 14 bits nas saídas analógicas), e o tempo de aceleração/desaceleração em ms. Esse tempo em ms é o tempo entre cada incremento na velocidade quando acelerando, e o tempo entre cada decremento de velocidade na desaceleração. As saídas responsáveis pela geração de pulsos (S1,S3 e S5) devem estar em modo FREQ. Mais detalhes da programação de resolução e modo de saídas analógicas veja **Configuração de Hardware**.

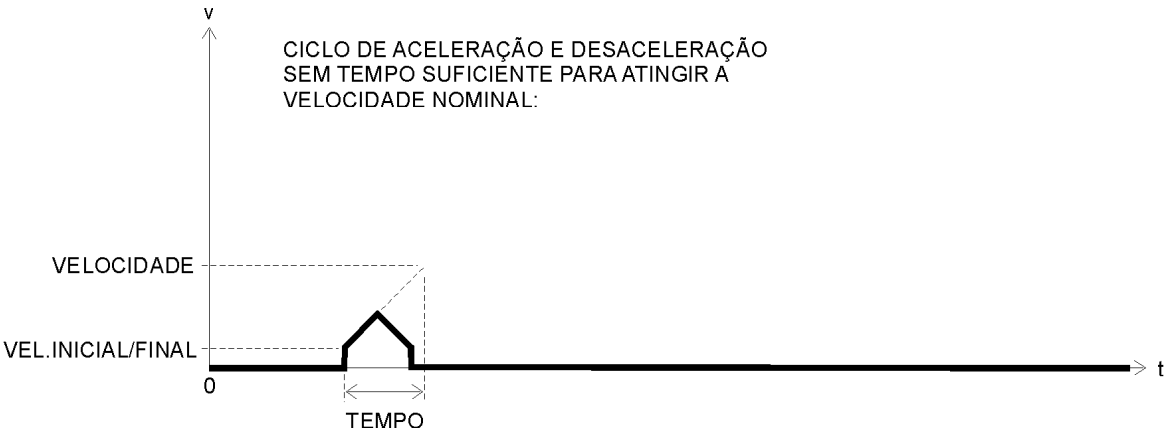


O nodo de Entrada (E) habilita o bloco por borda de subida (para editar o bloco selecionando os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Ou seja, sempre que o nodo E é energizado o controlador gera o número de pulsos especificado em Passos, com aceleração e desaceleração determinadas pela Vel.Inicial/Final e TempoAcel/Desac. A tempo total de aceleração e desaceleração pode ser calculado pela fórmula:

Tempo = (Velocidade - Vel.Inicial/Final) x TempoAcel/Desac

Esse tempo total é dado em milisegundos (ms). Caso o número de passos não permita chegar a velocidade de regime antes de ser necessário iniciar desaceleração, o controlador antecipa a desaceleração. Os gráficos abaixo exemplificam essa situação:





Os parâmetros **Velocidade** e **Vel.Inicial/Final** possuem valores mínimos e máximos que dependem da resolução nas saídas analógicas utilizada:

Resolução 10 bits: **Velocidade** e **Vel.Inicial/Final** entre 157 e 400.

Resolução 12 bits: **Velocidade** e **Vel.Inicial/Final** entre 40 e 400.

Resolução 14 bits: **Velocidade** e **Vel.Inicial/Final** entre 10 e 400.

Para calcular a velocidade real de rotação do motor primeiro calculamos o número de pulsos por segundo na saída analógica. Ele é determinado multiplicando o parâmetro por 25 (resolução de 25Hz):

Freqüência = 25 x Velocidade

Tendo a quantidade de pulsos necessária para uma volta completa do motor podemos calcular a velocidade rotacional em rpm (rotações por minuto):

Veloc(rpm) = 1500 x Velocidade / n

Sendo n o número de pulsos para completar uma volta completa no motor. Portanto, as velocidades podem variar, conforme a resolução das saídas analógicas, entre:

Resolução 10 bits: **Velocidade** e **Vel.Inicial/Final** entre 157 e 400 → Veloc entre 235500/n

até 600000/n

Resolução 12 bits: **Velocidade** e **Vel.Inicial/Final** entre 40 e 400 → Veloc entre 60000/n até 600000/n

Resolução 14 bits: **Velocidade** e **Vel.Inicial/Final** entre 10 e 400 → Veloc entre 15000/n até 600000/n

Se o motor necessita de 400 pulsos para completar uma volta completa, por exemplo, teremos os seguintes limites de velocidade:

Resolução 10 bits: **Velocidade** e **Vel.Inicial/Final** entre 157 e 400 → Veloc entre 588,75 rpm até 1500 rpm

Resolução 12 bits: **Velocidade** e **Vel.Inicial/Final** entre 40 e 400 → Veloc entre 150 rpm até 1500 rpm

Resolução 14 bits: **Velocidade** e **Vel.Inicial/Final** entre 10 e 400 → Veloc entre 37,5 rpm até 1500 rpm

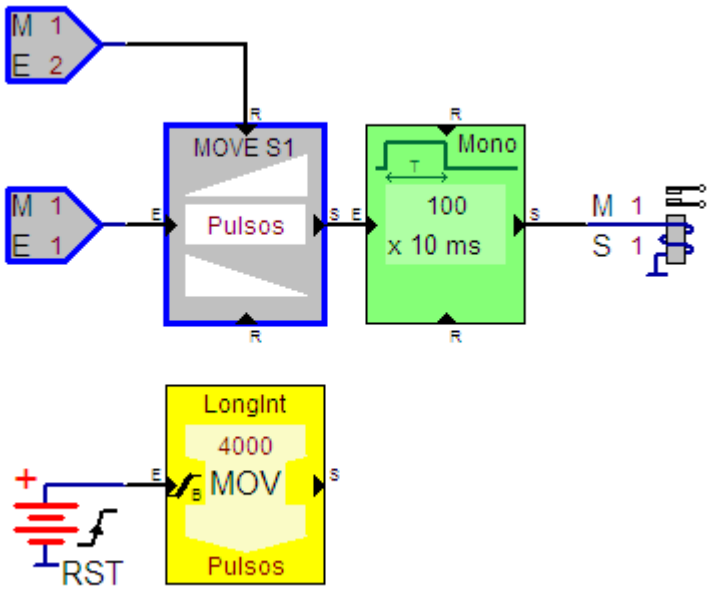
O parâmetro **TempoAcel/Desac** permite programar o tempo, em ms, entre um incremento ou decremento de velocidade e outro. Esse parâmetro pode variar de 1 a 1000, ou seja, de 1ms a 1s. Para calcular a aceleração ou desaceleração em rpm/s usamos a fórmula:

Acel (rpm/s) = 1.500.000 / (TempoAcel/Desac x n)

Por exemplo, se usarmos **TempoAcel/Desac** = 10 (10ms), e o motor for de 400 pulsos por volta, teremos uma aceleração e desaceleração de 375 rpm/s. Ou seja, considerando ainda motor de 400 passos por volta, se partirmos de 37,5 rpm (velocidade mínima em resolução de 14 bits) atingiremos 1500 rpm em 3,9 segundos. Portanto, a aceleração e desaceleração pode ser programada entre os limites de 1.500/n a 1.500.000/n. No caso de motor de 400 pulsos por volta esses limites são de 3,75 rpm/s a 3750 rpm/s.

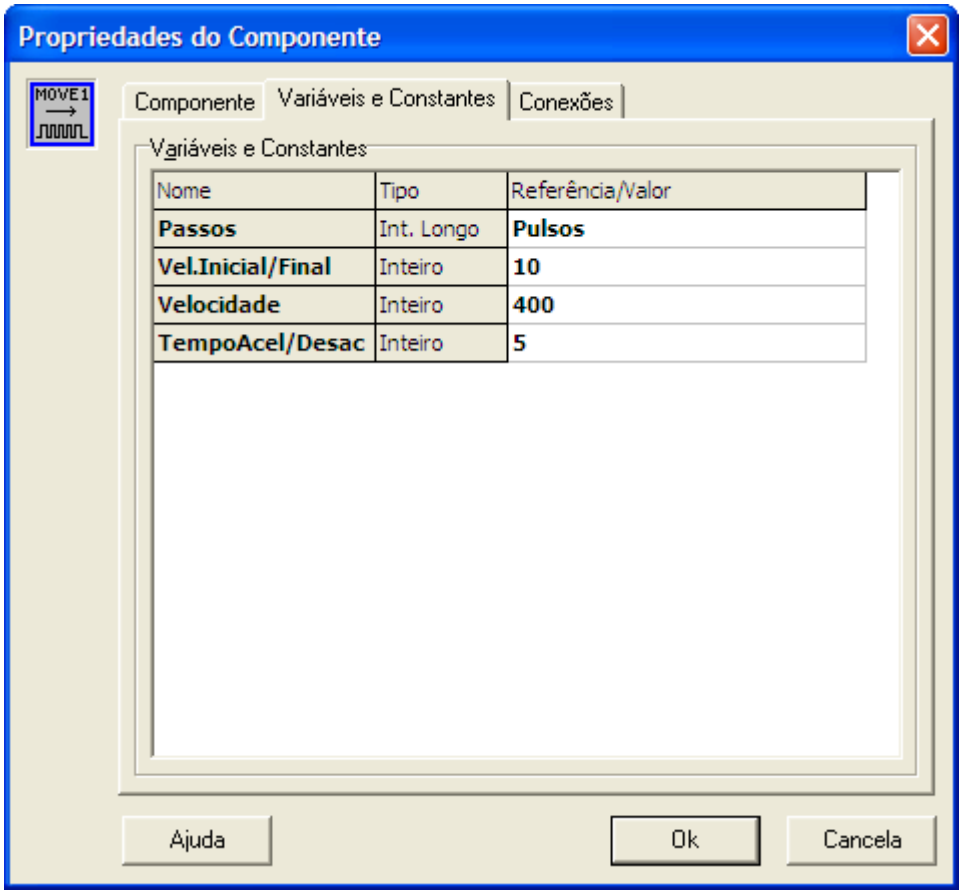
Já o nodo de Saída (S) é ativado ao iniciar a operação.Os nodos de Reset (R) estão internamente interligados e permitem desativar o bloco imediatamente. Note que qualquer sinal ligado a um dos nodos R está automaticamente ligado ao outro nodo R, já que se trata de dois nodos ligados internamente pelo bloco.

A seguir temos um exemplo de uso dos blocos Move S1:

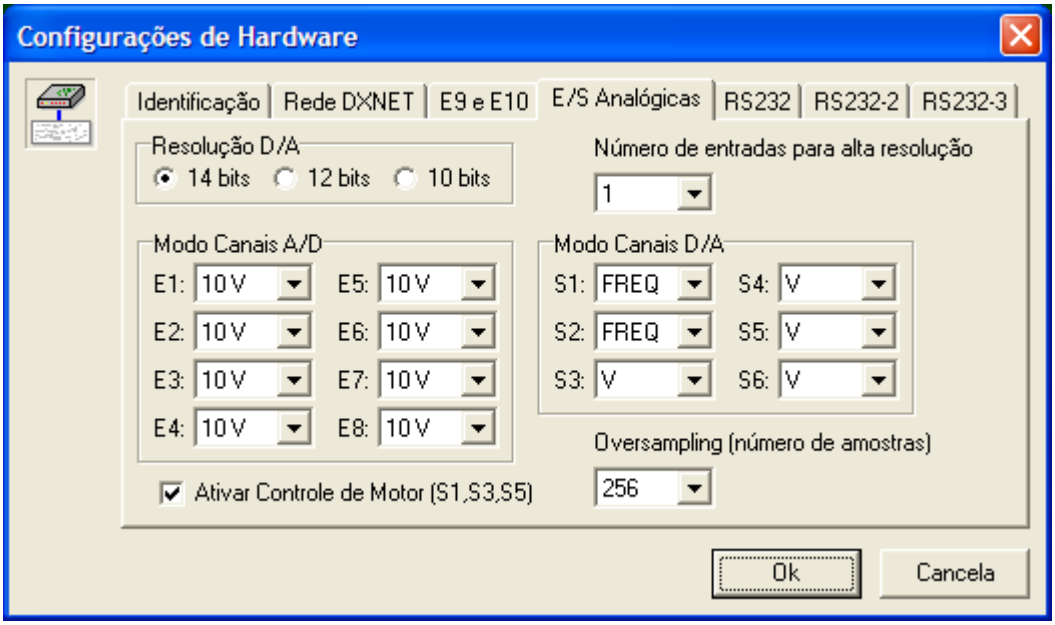


Exemplo de Programa Aplicativo: **Move_Motor.dxg**

Nesse exemplo usamos os seguintes parâmetros no bloco Move_S1:



E a **Configuração de Hardware** usada foi a seguinte:



Com esses parâmetros e essa configuração de hardware (resolução de 14 bits), considerando-se o uso de um motor de 400 passos por volta, teremos:

Veloc.Inicial/Final = 10 → Velocidade inicial e final em rpm = 1500 x 10 / 400 = 37,5 rpm

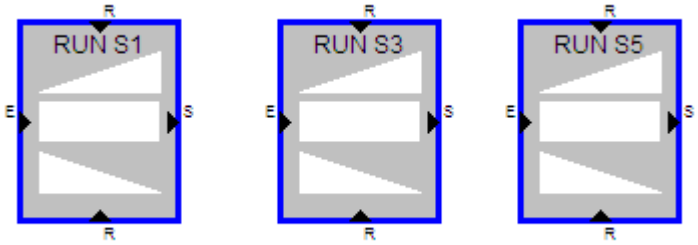
Velocidade = 400 → Velocidade de regime em rpm = 1500 x 400 / 400 = 1500 rpm

TempoAcel/Desac = 5 → Aceleração e desaceleração em rpm/s = 1.500.000 / (5 x 400) = 750 rpm/s

Ou seja, o motor irá partir de 37,5 rpm, e incrementar 750 rpm a cada segundo, até atingir 1500 rpm (em cerca de 2 segundos). Coloquei um valor inicial de **Passos** = 4000 (10 voltas completas do motor), que será insuficiente para o motor atingir a velocidade de regime (1500 rpm) com essa aceleração, e já iniciará a desaceleração em velocidade bem inferior. Se o número de passos for ampliado para, por exemplo, 40.000 (100 voltas completas do motor) será possível ao motor atingir a velocidade de regime (1500 rpm) antes de iniciar a desaceleração.

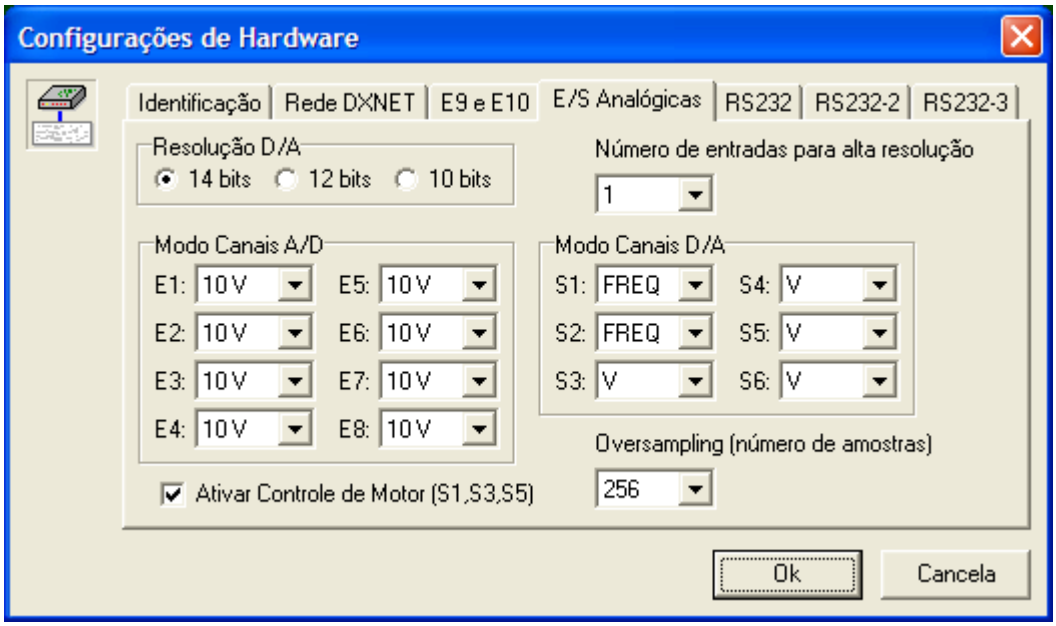
O motor, no exemplo, gira 10 voltas completas a cada energização da entrada E1 da Expansão µDX210 n°1. Caso a entrada E2 seja energizada o movimento cessa imediatamente, sem curva de desaceleração.

A função **RUN** é muito parecida com a função **MOVE**, mas mantêm a geração de pulsos enquanto a entrada E do bloco estiver energizada. Também utiliza o parâmetro **TempoAcel/Desac** para efetuar um início e final de movimento suave. Mas o parâmetro **Passos** é usado apenas para indicar o sentido de rotação do motor, e não o número de passos (esses são gerados enquanto o nodo de entrada E do bloco estiver ligado). Note que associada a saída S1, S3 e S5 temos as saídas S2, S4 e S6 para indicar o sentido de rotação, respectivamente. Se Passos for negativo o sentido de rotação é invertido, ao energizar uma dessas saídas que indicam o sentido de rotação.

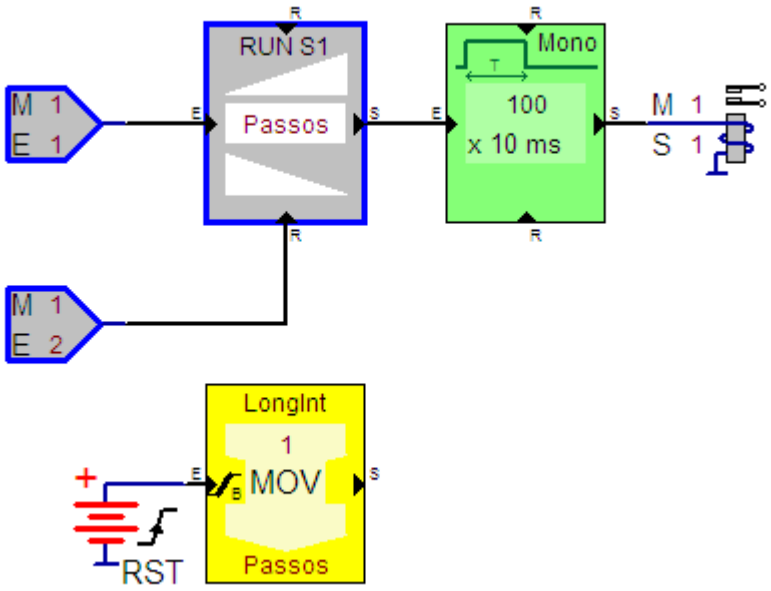


O nodo de Entrada (E) habilita o bloco por nível (para editar o bloco selecionando os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Ou seja, sempre que o nodo E está energizado o controlador gera pulsos, com aceleração. Ao desenergizar o nodo de entrada E o controlador desacelera até cessar a geração de pulsos. Já o nodo de Saída (S) é ativado ao iniciar a operação.Os nodos de Reset (R) estão internamente interligados e permitem desativar o bloco imediatamente. Note que qualquer sinal ligado a um dos nodos R está automaticamente ligado ao outro nodo R, já que se trata de dois nodos ligados internamente pelo bloco.

A conexão de até três drivers para controle de motores exige que, na **Configuração de Hardware** do programa aplicativo, seja marcada a opção **Ativar Controle de Motor (S1,S3,S5)**. Também as saídas analógicas a serem usadas para controle de motores devem ser comutadas para PWM, tanto na Configuração de Hardware quanto nos jumpers internos (ver seleção de jumpers). No exemplo abaixo são usadas as saídas analógicas S1 (pulsos) e S2 (sentido de rotação) para controle de um motor. Nesse caso os jumpers de seleção de filtro para as saídas S1 e S2 devem ser retirados (PWM), e os jumpers de seleção do tipo de saída analógica de S1 e S2 comutados para saída 0-10V.



O programa **Free_Run_S1.dxg** gera pulsos na saída analógica S1 enquanto a entrada E1 da Expansão µDX210 n°1 estiver energizada. Caso a variável **Passos** seja positiva a saída analógica S2 é mantida em zero durante a geração dos pulsos. Caso a variável **Passos** tenha um valor negativo a saída S2 é mantida energizada (10V), de forma a reverter a rotação do motor. O **TempoAcel/Desac** é especificado em 5, ou seja, de 5 em 5 ms a velocidade é incrementada ou decrementada. Cada vez que a entrada E1 da Expansão µDX210 n°1 é energizada o motor é acionado, só desacelerando ao desenergizar E1. Caso a entrada E2 seja acionada durante o movimento o motor pára imediatamente, sem desaceleração.



Exemplo de Programa Aplicativo: **Free_Run_S1.dxg**

Veja também:
ENTRADA/SAÍDA - Lineariza Motor

ENTRADA/SAÍDA - Lineariza Motor

Esse bloco permite linearizar a freqüência programada nas saídas analógicas (e, portanto, a velocidade de motores ligados a essas saídas analógicas).

Note que o valor da variável que determina a saída analógica no modo **FREQ** especifica o período da onda quadrada gerada, e não a freqüência. Assim, no caso da saída analógica S1 a variável v8 especifica o período conforme a fórmula abaixo:

Período = 125ns x Valor

Conforme a resolução usada para as saídas analógicas o valor máximo da variável se modifica:

Resolução 10 bits: Saída analógica **FREQ** com onda quadrada de período entre 100µs e 255,8µs (de 400 a 1023)
Resolução 12 bits: Saída analógica **FREQ** com onda quadrada de período entre 100µs e 1,023ms (de 400 a 4093)
Resolução 14 bits: Saída analógica **FREQ** com onda quadrada de período entre 100µs e 4,093ms (de 400 a 16383)

Note que valor 0 mantêm saída analógica sempre desligada, enquanto valores maiores que o valor máximo de cada resolução mantêm a saída sempre ligada. Por exemplo, um valor de 1000 (que é válido em qualquer das três resoluções) gera um período de 125µs.

Veja que para determinar a freqüência gerada na saída analógica é preciso fazer o inverso do período:

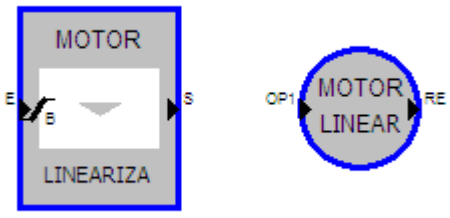
Freqüência = 1 / Período

Freqüência = 4.000.000 / Valor

Portanto, a freqüência na saída analógica, ao contrário do período, e fortemente não-linear. Para linearizar a freqüência na saída analógica existem os blocos no conjunto de blocos **Entrada/Saída** , chamado **Lineariza Motor** e **Lineriza Motor Var**. Esses blocos permitem entrar com valor entre 10 e 400, correspondendo a valores linearizados de 25 em 25Hz. Portanto, valor 10 na entrada desse bloco irá gerar valor correspondente a freqüência de 250Hz (10 x 25Hz), e valor de entrada 400 irá gerar valor correspondente para gerar freqüência de 10KHz (400 x 25Hz). Ou seja, para entrada de 10 a saída desse bloco será 16000 (4.000.000/16.000 = 250Hz), enquanto que para entrada de 400 a saída desse bloco será 400 (4.000.000/400 = 10KHz). Obviamente, essa faixa entre 10 e 400 só é funcional para resolução de 14 bits nas saídas analógicas. No caso de resolução de 12 bits a faixa se reduz para valores entre 40 (1KHz) e 400 (10KHz). No caso de resolução de 10 bits a faixa se reduz ainda mais, sendo de 157 (3925Hz) a 400 (10KHz). Assim, o valor de freqüência gerado conforme o valor da variável de entrada Operando do bloco é dado pela fórmula:

Freqüência = 25 x Operando

Note que os jumpers internos do µDX201 devem estar na posição **PWM** para uso no modo **FREQ** .



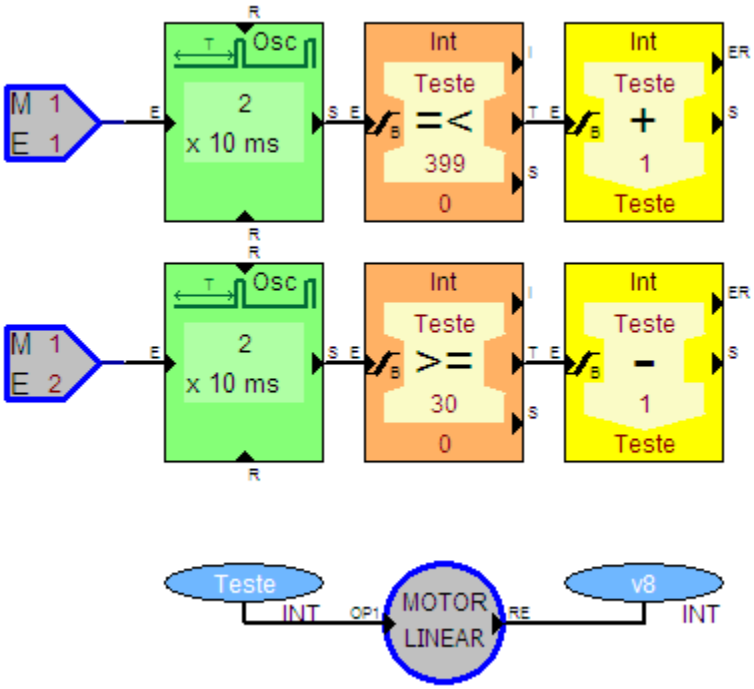
No caso do bloco **Lineariza Motor** o nodo de Entrada (E) habilita o bloco por nível ou por borda

de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Note que no caso do nodo de entrada E estar no modo por nível é possível modificar uma variável que especifica o setup e, instantaneamente, mudar a variável de saída. Já se a entrada E estiver por borda de subida apenas na energização desse nodo haverá a mudança de valor da variável de saída.

Já no caso de bloco **Lineariza Motor Var** a variável de entrada (valor a ser linearizado) e a de saída (valor linearizado) são conectados as conexões OP1 e RE, respectivamente, e a linearização ocorre constantemente.

O exemplo abaixo demonstra o uso desse bloco. A freqüência gerada na saída S1 inicia em zero (Teste=0). Ao acionar a entrada E1 da Expansão µDX210 n°1 a cada 20ms a variável Teste é incrementada. Ao atingir 10 começa a ser gerada uma freqüência de 250Hz na saída analógica S1, que é incrementada de 25 em 25 Hz a cada 20ms enquanto E1 for mantida energizada, até o limite de Teste=400 (10KHz). Ao energizar E2 a variável Teste é decrementada até o limite de Teste=29 (725 Hz).

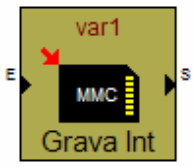


Exemplo de Programa Aplicativo: **Teste_Linear2.dxg**

Veja também:
ENTRADA/SAÍDA - MOVE & RUN

MMC - Gravar Inteiro

Este bloco permite gravar uma variável inteira no cartão MMC (Multi Media Card), em formato de comprimento variável. Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a captura do valor da variável inteira a ser gravada no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada.

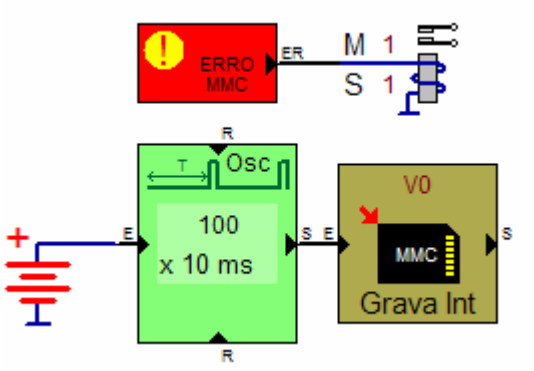


O operando Dado indica a variável inteira a ser gravada no cartão. Note que Dado pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

Note que este bloco grava uma variável inteira em representação ASCII no cartão MMC com o mínimo de caracteres possíveis. Por exemplo, se a variável a ser gravada estiver com valor 2 serão gravados apenas dois caracteres: 32h + 3Bh (2;). Já se a mesma variável assumir valor -12450 serão gravados sete caracteres: 2Dh + 31h + 32h + 34h + 35h + 30h + 3Bh (-12450;). Note que variáveis inteiras podem assumir valores entre -32768 e 32767.

Este bloco, na verdade, coloca o dado a ser gravado no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

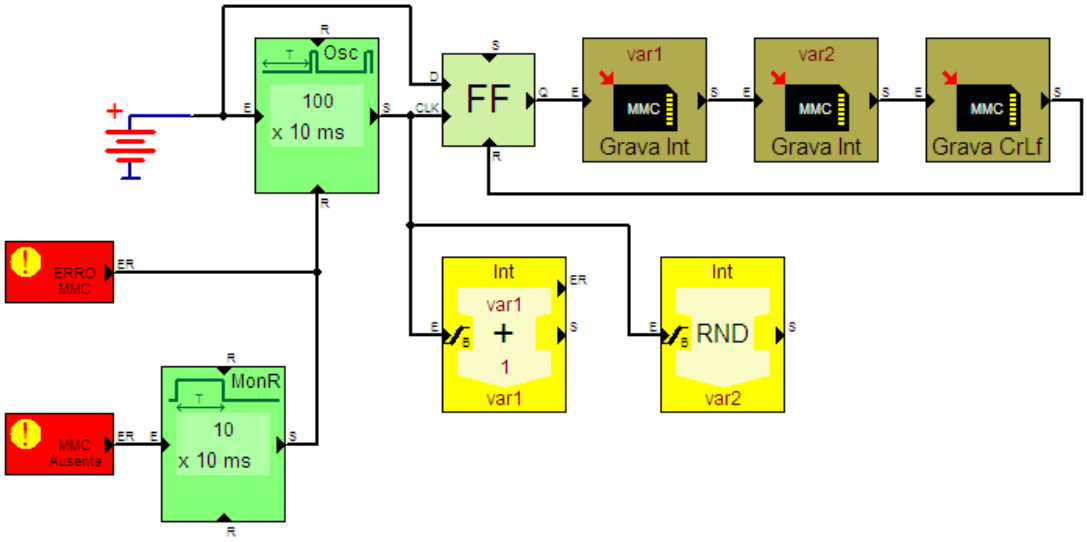
Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.



Exemplo de Programa Aplicativo: **MMC - Gravar Inteiro.dxg**

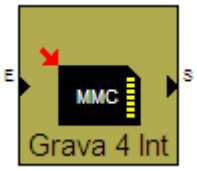
O exemplo salva o valor lido na entrada analógica E1 do µDX200 (que corresponde a variável v0) a cada segundo. Além disso, o programa liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, ou ainda arquivo corrompido).

O próximo exemplo grava duas variáveis inteiras (var1 e var2) e finaliza a aquisição com CrLf (carriage return e line feed). Note que foi usado um bloco de flip-flop (FF) para garantir que os blocos ficassem energizados até a operação ser concluída. Os dados são salvos a cada 100ms. Note que var1 é incrementada a cada aquisição, enquanto var2 é um número randômico.



Exemplo de Programa Aplicativo: **MMC - Gravar Inteiro 2.dxg**

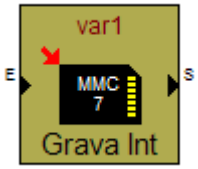
A partir da versão 3.58 do controlador µDX201 existe a possibilidade de gravar 4 variáveis inteiras com apenas um bloco MMC. Nesse caso as variáveis são gravadas em seqüência e, portanto, deve-se indicar uma variável absoluta (Vnn) e o bloco irá gravá-la e as três subseqüentes (Vnn+1, Vnn+2, Vnn+3).



Veja também:
MMC - Gravar Inteiro (7 caracteres)

MMC - Gravar Inteiro (7 caracteres)

Este bloco permite gravar uma variável inteira no cartão MMC (Multi Media Card), sempre com formato de 7 caracteres. Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a captura do valor da variável inteira a ser gravada no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada.

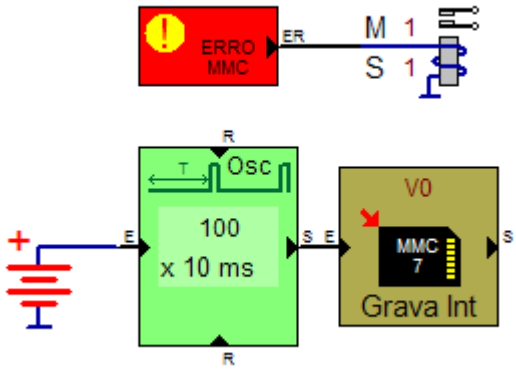


O operando Dado indica a variável inteira a ser gravada no cartão. Note que Dado pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

Note que este bloco grava uma variável inteira em representação ASCII no cartão MMC sempre com sete caracteres. Por exemplo, se a variável a ser gravada estiver com valor 2 serão gravados sete caracteres: 20h + 20h + 20h + 20h + 20h + 32h + 3Bh (2;). Já se a mesma variável assumir valor -12450 serão gravados também sete caracteres: 2Dh + 31h + 32h + 34h + 35h + 30h + 3Bh (-12450;). Note que variáveis inteiras podem assumir valores entre -32768 e 32767. Este tamanho fixo de caracteres a serem gravados é útil para manter alinhamento nos campos do arquivo gerado no cartão MMC, permitindo acessar registros de forma aleatória.

Este bloco, na verdade, coloca o dado a ser gravado no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

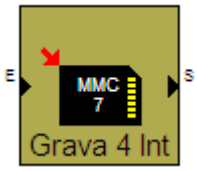
Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.



Exemplo de Programa Aplicativo: **MMC - Gravar Inteiro (7 caracteres).dxg**

O exemplo salva o valor lido na entrada analógica E1 do µDX200 (que corresponde a variável v0) a cada segundo. Além disso, o programa liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, ou ainda arquivo corrompido).

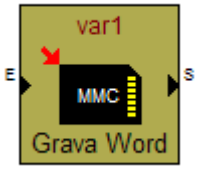
A partir da versão 3.58 do controlador µDX201 existe a possibilidade de gravar 4 variáveis inteiras com apenas um bloco MMC. Nesse caso as variáveis são gravadas em seqüência e, portanto, deve-se indicar uma variável absoluta (Vnn) e o bloco irá gravá-la e as três subseqüentes (Vnn+1, Vnn+2, Vnn+3). Cada variável utiliza 7 caracteres.



Veja também:
MMC - Gravar Inteiro

MMC - Gravar Word

Este bloco permite gravar uma variável word no cartão MMC (Multi Media Card), em formato de comprimento variável. Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a captura do valor da variável word a ser gravada no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada.

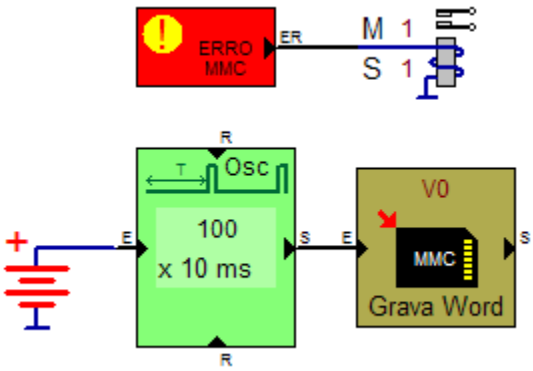


O operando Dado indica a variável word a ser gravada no cartão. Note que Dado pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

Note que este bloco grava uma variável word em representação ASCII no cartão MMC com o mínimo de caracteres possíveis. Por exemplo, se a variável a ser gravada estiver com valor 2 serão gravados dois caracteres: 32h + 3Bh (2;). Já se a mesma variável assumir valor 56300 serão gravados seis caracteres: 35h + 36h + 33h + 30h + 30h + 3Bh (56300;). Note que variáveis word podem assumir valores entre 0 e 65535.

Este bloco, na verdade, coloca o dado a ser gravado no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

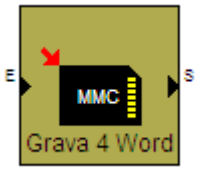
Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.



Exemplo de Programa Aplicativo: **MMC - Gravar Word.dxg**

O exemplo salva o valor lido na entrada analógica E1 do µDX200 (que corresponde a variável v0) a cada segundo. Além disso, o programa liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, ou ainda arquivo corrompido).

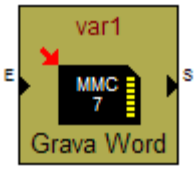
A partir da versão 3.58 do controlador µDX201 existe a possibilidade de gravar 4 variáveis word com apenas um bloco MMC. Nesse caso as variáveis são gravadas em seqüência e, portanto, deve-se indicar uma variável absoluta (Vnn) e o bloco irá gravá-la e as três subseqüentes (Vnn+1, Vnn+2, Vnn+3).



Veja também:
MMC - Gravar Word (7 caracteres)

MMC - Gravar Word (7 caracteres)

Este bloco permite gravar uma variável word no cartão MMC (Multi Media Card), sempre com formato de 7 caracteres. Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a captura do valor da variável word a ser gravada no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada.

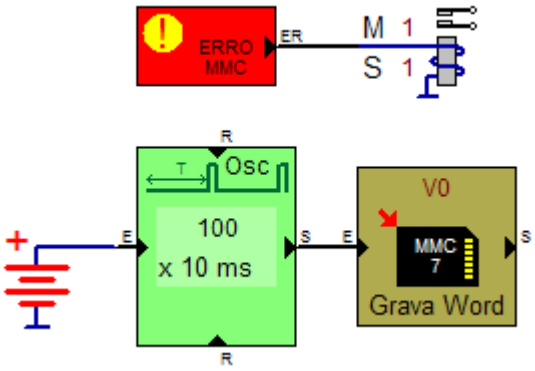


O operando Dado indica a variável word a ser gravada no cartão. Note que Dado pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

Note que este bloco grava uma variável word em representação ASCII no cartão MMC sempre com sete caracteres. Por exemplo, se a variável a ser gravada estiver com valor 2 serão gravados sete caracteres: 20h + 20h + 20h + 20h + 20h + 32h + 3Bh (2;). Já se a mesma variável assumir valor 56300 serão gravados também sete caracteres: 20h + 35h + 36h + 33h + 30h + 30h + 3Bh (56300;). Note que variáveis word podem assumir valores entre 0 e 65535. Este tamanho fixo de caracteres a serem gravados é útil para manter alinhamento nos campos do arquivo gerado no cartão MMC, permitindo acessar registros de forma aleatória.

Este bloco, na verdade, coloca o dado a ser gravado no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

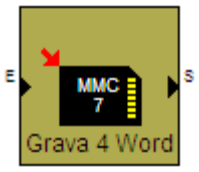
Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.



Exemplo de Programa Aplicativo: **MMC - Gravar Word (7 caracteres).dxg**

O exemplo salva o valor lido na entrada analógica E1 do µDX200 (que corresponde a variável v0) a cada segundo. Além disso, o programa liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, ou ainda arquivo corrompido).

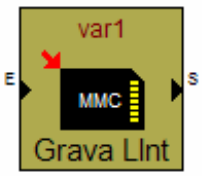
A partir da versão 3.58 do controlador µDX201 existe a possibilidade de gravar 4 variáveis word com apenas um bloco MMC. Nesse caso as variáveis são gravadas em seqüência e, portanto, deve-se indicar uma variável absoluta (Vnn) e o bloco irá gravá-la e as três subseqüentes (Vnn+1, Vnn+2, Vnn+3). Cada variável utiliza 7 caracteres.



Veja também:
MMC - Gravar Word

MMC - Gravar LongInt

Este bloco permite gravar uma variável longint no cartão MMC (Multi Media Card), em formato de comprimento variável. Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a captura do valor da variável longint a ser gravada no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada.



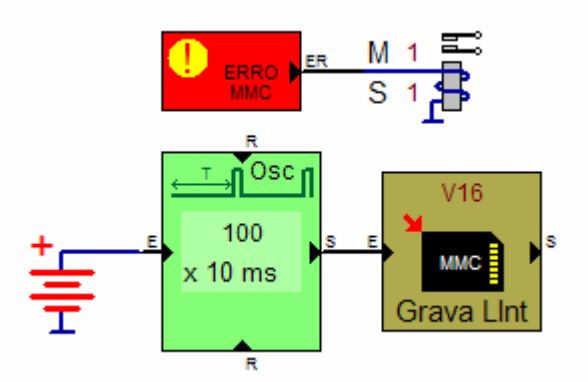
O operando Dado indica a variável longint a ser gravada no cartão. Note que Dado pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

Note que este bloco grava uma variável longint em representação ASCII no cartão MMC com o mínimo de caracteres possíveis. Por exemplo, se a variável a ser gravada estiver com valor 2 serão gravados apenas dois caracteres: 32h + 3Bh (2;). Já se a mesma variável assumir valor

-1234567 serão gravados nove caracteres: 2Dh + 31h + 32h + 33h + 34h + 35h + 36h + 37h + 3Bh (-1234567;). Note que variáveis longint podem assumir valores entre os seguintes limites: -2.147.483.648 e 2.147.483.647.

Este bloco, na verdade, coloca o dado a ser gravado no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

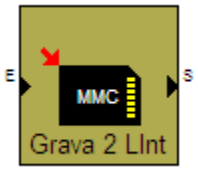
Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.



Exemplo de Programa Aplicativo: **MMC - Gravar LongInt.dxg**

O exemplo salva o valor do contador incremental/decremental longint associado as entradas E9 e E10 do µDX200 (que corresponde a variável v16) a cada segundo. Além disso, o programa liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, ou ainda arquivo corrompido).

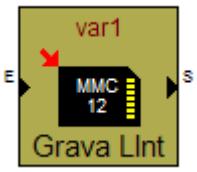
A partir da versão 3.58 do controlador µDX201 existe a possibilidade de gravar 2 variáveis longint com apenas um bloco MMC. Nesse caso as variáveis são gravadas em seqüência e, portanto, deve-se indicar uma variável absoluta (Vnn) e o bloco irá gravá-la e a subseqüente (Vnn+2).



Veja também:
MMC - Gravar LongInt (12 caracteres)

MMC - Gravar LongInt (12 caracteres)

Este bloco permite gravar uma variável longint no cartão MMC (Multi Media Card), sempre com formato de 12 caracteres. Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a captura do valor da variável longint a ser gravada no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada.

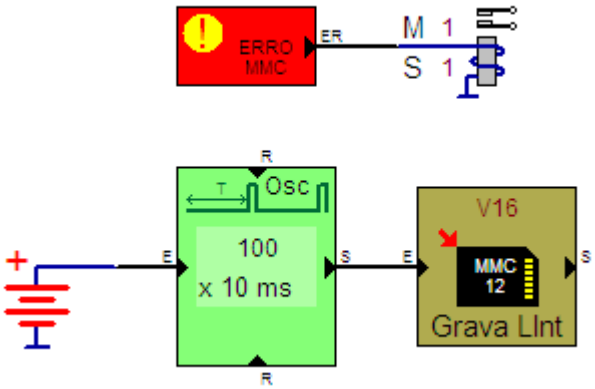


O operando Dado indica a variável longint a ser gravada no cartão. Note que Dado pode ser variável longint, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

Note que este bloco grava uma variável longint em representação ASCII no cartão MMC sempre com 12 caracteres. Por exemplo, se a variável a ser gravada estiver com valor 2 serão gravados doze caracteres: 20h + 20h + 20h + 20h + 20h + 20h + 20h + 20h + 20h + 20h + 32h + 3Bh (2;). Já se a mesma variável assumir valor -1234567 serão gravados também doze caracteres: 20h + 20h + 20h + 2Dh + 31h + 32h + 33h + 34h + 35h + 36h + 37h + 3Bh (-1234567;). Note que variáveis longint podem assumir valores entre os seguintes limites: -2.147.483.648 e 2.147.483.647.

Este bloco, na verdade, coloca o dado a ser gravado no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

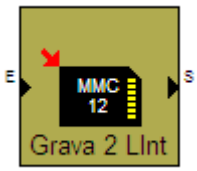
Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.



Exemplo de Programa Aplicativo: **MMC - Gravar LongInt (12 caracteres).dxg**

O exemplo salva o valor do contador incremental/decremental longint associado as entradas E9 e E10 do µDX200 (que corresponde a variável v16) a cada segundo. Além disso, o programa liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, ou ainda arquivo corrompido).

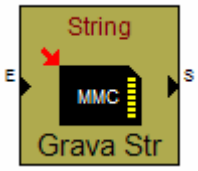
A partir da versão 3.58 do controlador µDX201 existe a possibilidade de gravar 2 variáveis longint com apenas um bloco MMC. Nesse caso as variáveis são gravadas em seqüência e, portanto, deve-se indicar uma variável absoluta (Vnn) e o bloco irá gravá-la e a subseqüente (Vnn+2). Cada variável utiliza 12 caracteres.



Veja também:
MMC - Gravar LongInt

MMC - Gravar String

Este bloco permite gravar um string no cartão MMC (Multi Media Card), em formato de comprimento variável. O string pode ter, no máximo, comprimento de 30 caracteres. Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a captura do string a ser gravado no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada.



String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Note que String pode ser variável word ou inteira, ou ainda uma referência a uma posição de memória (Mxxxx).

Note que para inicializar strings em memória de programa (Flash) ou em memória volátil (RAM) deve-se usar os blocos de Inicia Tabela ou Inicia Tabela em RAM.

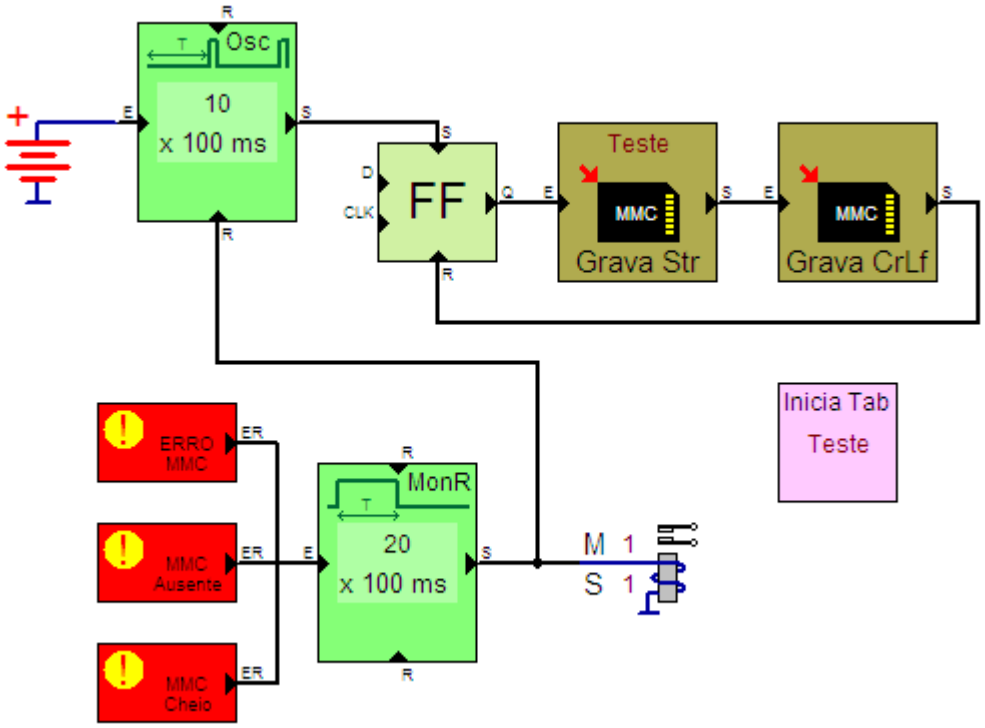
Este bloco, na verdade, coloca os dados a serem gravados no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.

O exemplo salva o valor do string Teste ("TesteMMC") a cada segundo. Além disso, o programa liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, arquivo corrompido, cartão não-instalado, ou ainda cartão cheio).

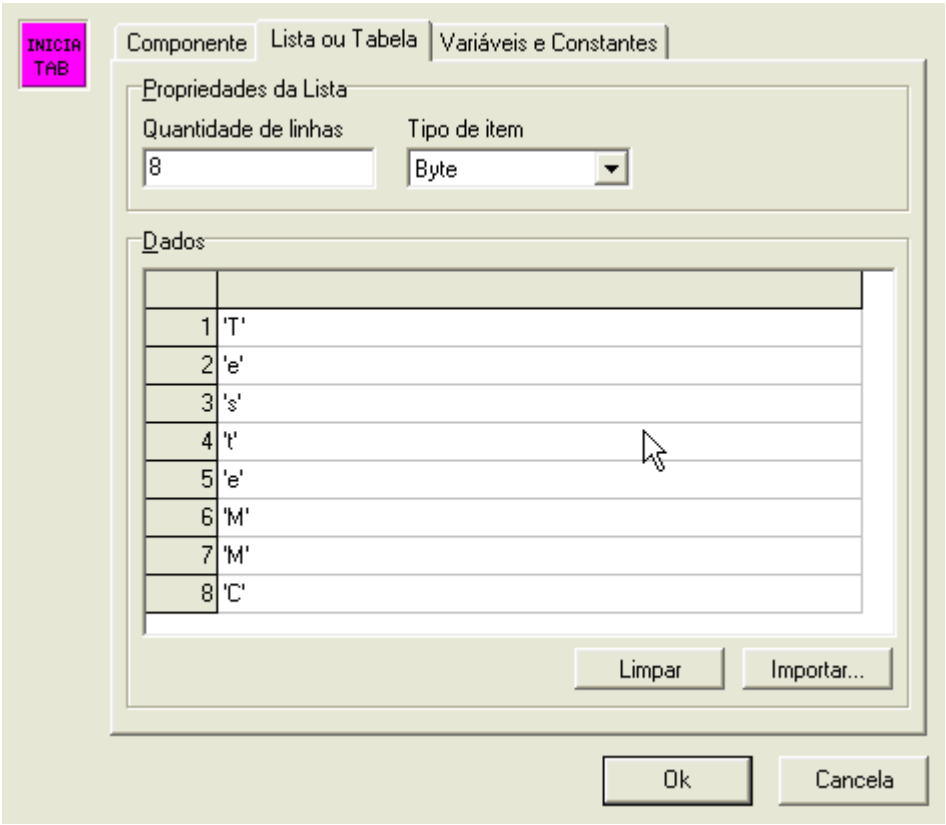
Repare que é utilizado um flip-flop (FF) para garantir que os blocos de MMC serão energizados até a conclusão de sua operação. O bloco Oscilador gera um pulso a cada segundo na entrada S (Set) do FF, o que faz este ligar a saída Q, energizando os blocos de MMC, que gravam o string Teste e uma finalização CrLf (carriage return e line feed).

Após a conclusão da gravação o nodo de saída do bloco MMC Grava CrLf liga, o que desliga o FF via nodo R (reset). Caso ocorra um erro a saída S1 é acionada por pelo menos 2 segundos (monoestável) e o oscilador de gravação é inibido.



Exemplo de Programa Aplicativo: **MMC - Gravar String.dxg**

O string Teste foi inicializado com os seguintes valores:

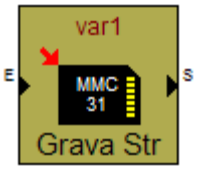


Veja também:

MMC - Gravar String (31 caracteres)

MMC - Gravar String (31 caracteres)

Este bloco permite gravar um string no cartão MMC (Multi Media Card), sempre com formato de 31 caracteres. O string pode ter, no máximo, comprimento de 30 caracteres. Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a captura do string a ser gravado no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada.



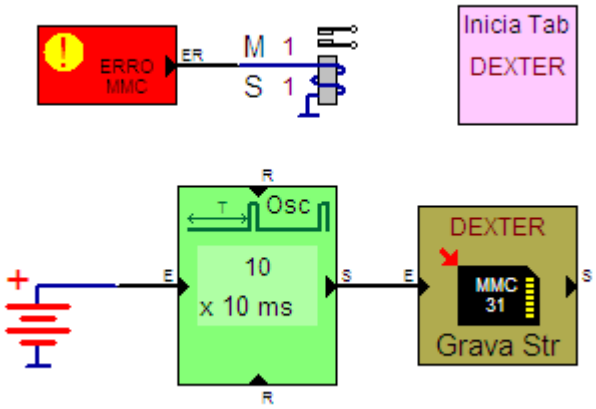
String é a variável tipo word (0 a 65535) que aponta para o string a ser convertido. Note que String pode ser variável word ou inteira, ou ainda uma referência a uma posição de memória (Mxxxx).

Note que para inicializar strings em memória de programa (Flash) ou em memória volátil (RAM) deve-se usar os blocos de Inicia Tabela ou Inicia Tabela em RAM.

Este bloco, na verdade, coloca os dados a serem gravados no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.

Este bloco grava o string, independentemente de seu comprimento, com 31 caracteres. Por exemplo, se o string a ser gravado for "Teste" serão gravados os seguintes caracteres no cartão MMC: 54h + 65h + 73h + 74h + 65h + 20h + 3Bh (Teste ;).

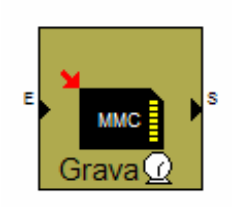


Exemplo de Programa Aplicativo: **MMC - Gravar String (31 caracteres).dxg**

Veja também:
MMC - Gravar String

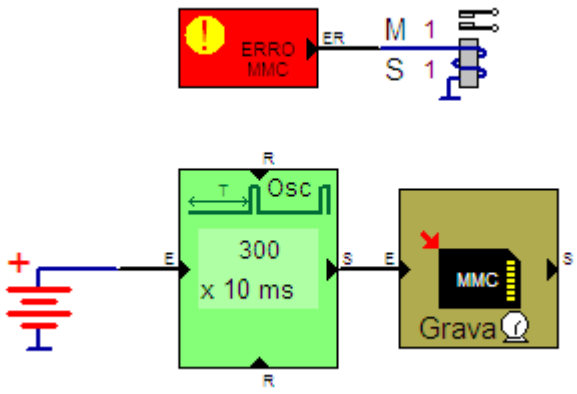
MMC - Gravar Relógio

Este bloco grava o horário (hora, minuto e segundo) e a data (dia, mês, ano) constantes no relógio de tempo real do µDX200 no cartão MMC (Multi Media Card). Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a captura do horário e data a serem gravados no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada. O formato é fixo, com 18 caracteres: hh:mm:ss;dd/mm/aa;.



Este bloco, na verdade, coloca o dado a ser gravado no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.

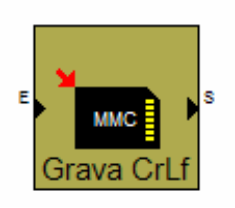


Exemplo de Programa Aplicativo: **MMC - Gravar Relógio.dxg**

O exemplo salva o valor do relógio de tempo real a cada 3 segundos. Além disso, o programa liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, ou ainda arquivo corrompido).

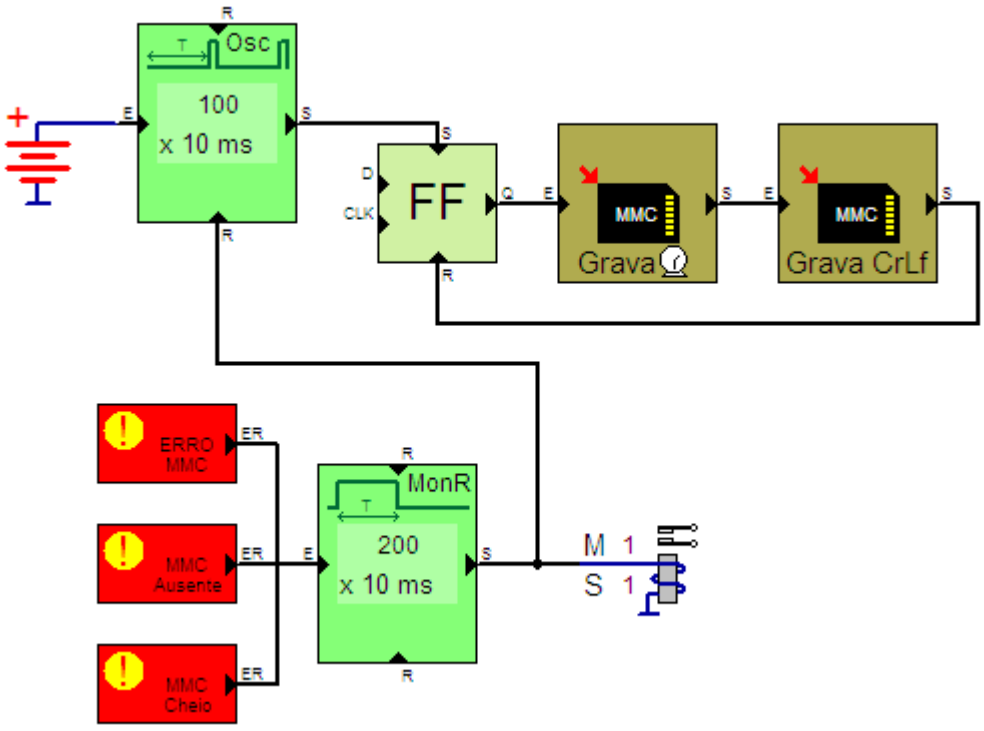
MMC - Gravar CrLf

Este bloco grava uma finalização de linha (carriage return e line feed) no cartão MMC (Multi Media Card). Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a finalização de linha no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada. O formato é fixo com dois caracteres: Cr + Lf (0Dh + 0Ah).



Este bloco, na verdade, coloca o dado a ser gravado no buffer de gravação do cartão MMC. A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado um bloco MMC - Salvar Buffer.

Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.



Exemplo de Programa Aplicativo: **MMC - Gravar CrLf.dxg**

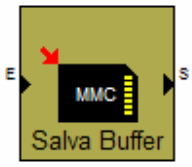
O exemplo salva o valor do relógio de tempo real do µDX200, seguido de uma finalização de linha (CrLf) a cada segundo. Além disso, o programa liga a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) quando ocorrer um erro no cartão MMC (cartão não-formatado, arquivo inexistente no cartão MMC, arquivo corrompido, cartão não-instalado, ou ainda cartão cheio).

Repare que é utilizado um flip-flop (FF) para garantir que os blocos de MMC serão energizados até a conclusão de sua operação. O bloco Oscilador gera um pulso a cada segundo na entrada S (set) do FF, o que faz este ligar a saída Q, energizando os blocos de MMC, que gravam o relógio de tempo real e uma finalização CrLf (carriage return e line feed).

Após a conclusão da gravação o nodo de saída do bloco MMC Grava CrLf liga, o que desliga o FF via nodo R (reset). Caso ocorra um erro a saída S1 é acionada por pelo menos 2 segundos (monoestável) e o oscilador de gravação é inibido.

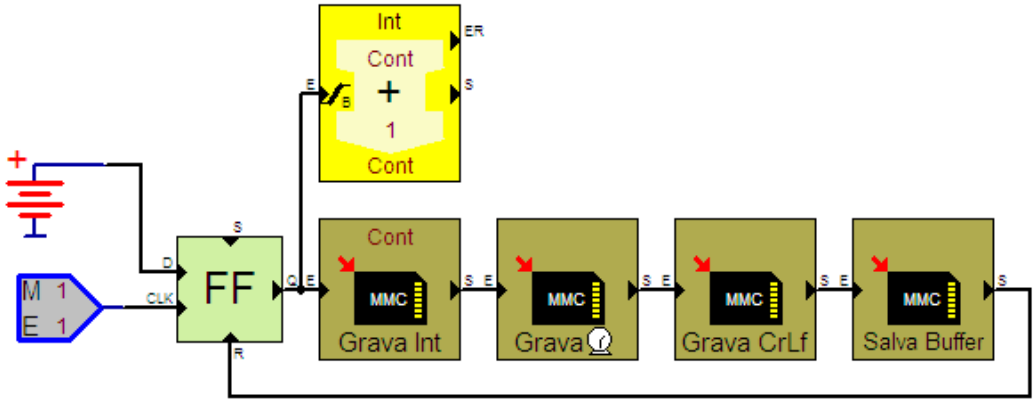
MMC - Salvar Buffer

Todos os blocos anteriores para gravação de dados no cartão MMC, na verdade, salvam estes dados em um buffer em memória volátil (RAM). A gravação do cartão propriamente dita é feita sempre que este buffer atingir 512 bytes, ou quando for executado o bloco de Salvar Buffer.



Note que este bloco possui um nodo de entrada (E) que, ao ser energizado, sinaliza para o bloco o momento em que deve ser feita a gravação do buffer no cartão MMC, e um nodo de saída (S) que é acionado quando a operação for completada.

Os dados gravados no cartão MMC podem ser lidos posteriormente em um computador IBM-PC compatível, em formato compatível com planilhas como o Excel da Microsoft. Caso o cartão MMC seja retirado do controlador µDX200, ao reinserí-lo o CLP irá efetuar novas gravações a partir do final do arquivo já existente. Ou seja, não há problema em inserir ou retirar o cartão do µDX200, mas deve-se antes efetuar uma execução do bloco MMC - Salvar Buffer para que os dados existentes no buffer de 512 bytes não sejam perdidos.

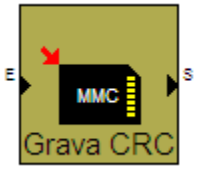


Exemplo de Programa Aplicativo: **MMC - Salvar Buffer.dxg**

O exemplo grava um número seqüencial (Cont), a data e hora, e uma terminação de linha (CrLf) cada vez que a entrada E1 do módulo M1 de Entradas/Saídas (µDX210) for energizada. Para garantir que os dados sejam efetivamente gravados no cartão MMC cada vez que E1 é energizada foi utilizado o bloco de Salvar Buffer.

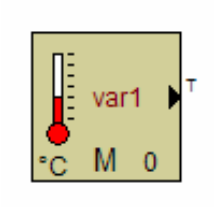
MMC - Gravar CRC

Este bloco calcula o CRC-16 dos dados gravados no cartão MMC/SD desde o último carriage return / line feed (CrLf). Com isso é possível verificar se os dados da linha de dados gravados está idônea, sem erros. Note que o CRC-16 gerado é do mesmo padrão usado no protocolo nativo do µDX201 ou do protocolo ModBus- RTU. Este bloco utiliza 5 caracteres para gravar o CRC.



I²C - Temperatura

Estes blocos acessam a rede I²C existente no controlador µDX200, permitindo a leitura de sensores de temperatura e umidade. No caso, este bloco lê sensores de temperatura instalados na rede I²C (até 8 sensores). A rede I²C possibilita a conexão de sensores distantes, até 1 Km do controlador µDX200 (desde que a fiação esteja instalada em eletroduto metálico exclusivo para conexão de dados).

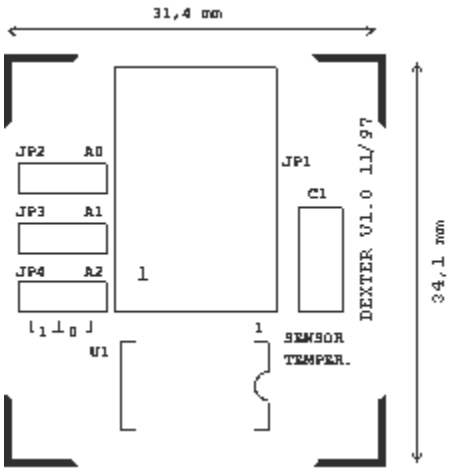


Note que este bloco possui uma conexão inteira de saída (T) que disponibiliza o valor de temperatura. Caso não seja especificada nenhuma variável no operando Temperatura é preciso usar esta conexão para acessar o valor de temperatura do sensor. Se for associada uma variável ao operando a temperatura do sensor é constantemente transferida para esta variável, além da variável estar disponibilizada na conexão de saída do bloco.

O operando Temperatura indica a variável inteira que irá receber o valor lido de temperatura no sensor especificado no operando Endereço. Note que Temperatura pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Também pode ser omitida e, neste caso, o valor de temperatura estará disponível na conexão de saída do bloco.

Já o operando Endereço é um dado com formato byte e, portanto, deve necessariamente ser uma constante (o µDX200 trata variáveis apenas de 16 bits - inteiro ou word - e 32 bits - longint ou real). Endereço pode assumir valores de 0 a 7, correspondendo aos endereços dos 8 sensores de temperatura passíveis de serem instalados na rede I²C do µDX200.

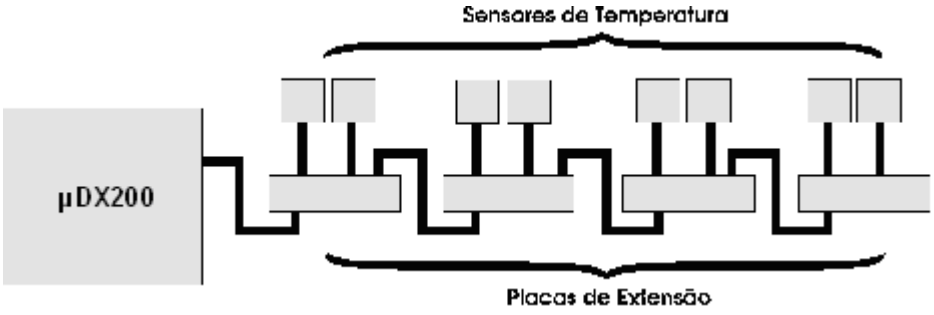
Para conexão dos sensores de temperatura existe um conector tipo RJ11 no Controlador Programável µDX200, o que permite conectar um sensor. Para conectar sensores adicionais a DEXTER comercializa uma placa de extensão com 3 derivações. Também pode ser utilizado qualquer derivador para linha telefônica, como os comumente encontrados em lojas de material de informática. Note que todos os sensores utilizam a mesma linha de comunicação a 4 fios. O que os distingue é o endereço programado via estrapes (jumpers) na placa de cada sensor - 3 jumpers, permitindo endereço de 0 a 7.



Acima temos o layout da placa de sensor de temperatura . Note o conector RJ11 central (JP1), que conecta a placa ao controlador µDX200. Existem 3 jumpers à esquerda (JP2, JP3, JP4), responsáveis pelo endereço do sensor de temperatura. JP2 é o bit menos significativo deste endereço e JP4 o mais significativo. Se o jumper estiver ligado entre o pino à esquerda e o pino central o jumper estará em 1 (um). Já se o jumper estiver entre o pino central e o pino à direita o jumper estará em 0 (zero). A tabela a seguir indica o valor de JP2, JP3 e JP4 para os 8 endereços possíveis para o sensor de temperatura:

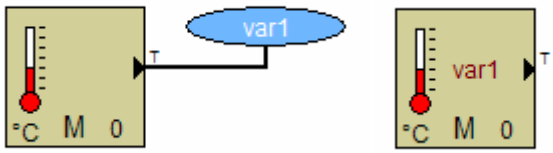
	JP4	JP3	JP2
Sensor de Temperatura 0	0	0	0
Sensor de Temperatura 1	0	0	1
Sensor de Temperatura 2	0	1	0
Sensor de Temperatura 3	0	1	1
Sensor de Temperatura 4	1	0	0
Sensor de Temperatura 5	1	0	1
Sensor de Temperatura 6	1	1	0
Sensor de Temperatura 7	1	1	1

A placa de extensão opcional permite derivar 3 ligações para sensores de temperatura a partir do cabo conectado ao µDX200. Assim, para conectar todos os 8 sensores de temperatura, por exemplo, são necessárias 4 placas de extensão (sendo que uma saída de uma das placas fica vazia):



A Dexter comercializa o sensor de temperatura, próprio para mensurar temperatura ambiente, em dois modelos. O modelo existente para a linha de controladores µDX100 é compatível com o µDX200, e permite leitura de temperatura em toda faixa do sensor (-55°C a 125°C), com resolução de 0,5°C ou de 1°C. Já o novo modelo de sensor de temperatura também permite leitura em toda faixa do sensor (-55°C a 125°C), mas com uma resolução de 0,0625°C ou de 1°C. O bloco mostrado até o momento permite a leitura dos dois tipos de sensor com resolução de

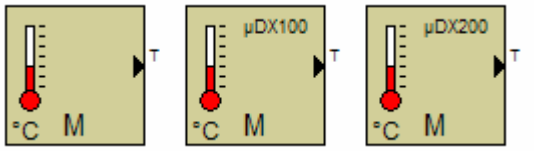
1°C. Neste caso a variável de saída do bloco indica diretamente a temperatura ambiente, não sendo necessário efetuar nenhuma conversão. Por exemplo, se a variável var1 dos exemplos abaixo assumir valor 25 significa que a temperatura ambiente está em 25°C.



Exemplo de Programa Aplicativo: **I2C - Temperatura.dxg**

Os dois exemplos acima têm efeitos idênticos, ou seja, transferem o valor lido de temperatura do sensor zero para a variável var1.

Mas a partir da versão 1.3 da Biblioteca Padrão para µDX200 foram introduzidos outros dois blocos para leitura de sensores de temperatura. Assim, atualmente existem três blocos para leitura de temperatura:

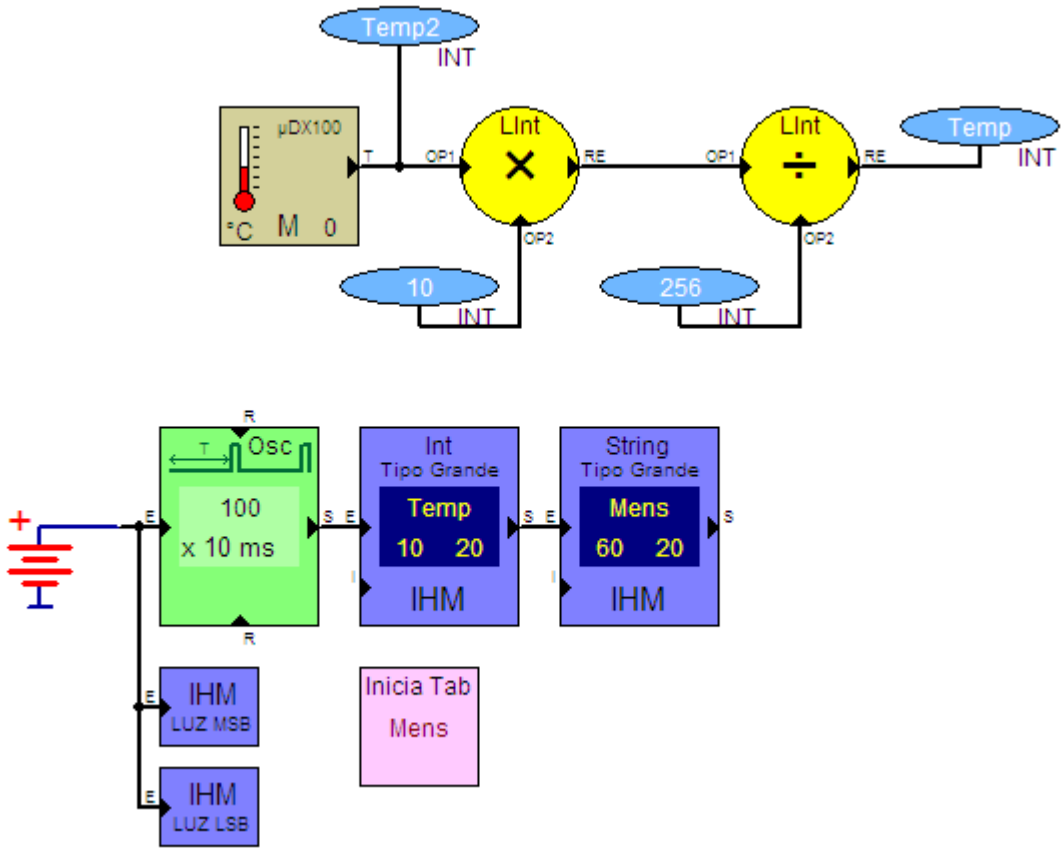


O primeiro bloco é o já citado, que permite leitura dos dois tipos de sensor de temperatura, com resolução de 1°C e variável de saída direta, ou seja, não há necessidade de conversão dos dados.

Já o segundo bloco lê os dois tipos de sensor (para µDX100 e para µDX200) com resolução de 0,5°C. Neste caso a variável de saída deve ser dividida por 256 para se obter o valor de temperatura em °C. Por exemplo, se a variável de saída assumir valor 6528 isso correspoderá a temperatura ambiente de 6528/256 = 25,5°C.

Por fim, o terceiro bloco permite leitura apenas do novo sensor (para µDX200), com uma resolução bastante elevada (0,0625°C). Também é preciso dividir a variável de saída por 256. A variável inteira resultante do sensor de temperatura representa a temperatura em graus celsius, sendo que o byte superior da variável indica os graus, e o byte inferior os décimos de grau. Assim, para obtermos o valor de temperatura em °C basta dividir a variável por 256. Por exemplo, digamos que a variável de saída do bloco Temperatura assumiu valor 6256. Neste caso, a temperatura é 6256/256 = 24,4375°C.

A seguir temos um exemplo de uso do sensor de temperatura lendo temperatura de 0,5 em 0,5°C e indicando a temperatura em IHM para µDX201:



Exemplo de Programa Aplicativo: **I2C - Temperatura2.dxg**

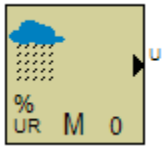
Atenção: *Atualmente a Dexter possui um novo modelo de sensor que incorpora o sensor de temperatura e do de umidade em um único dispositivo. Esse equipamento é designado como* ***Sensor Temp & Umid*** *para µDX201. Cada controlador µDX201 pode ler até 8 sensores* ***Temp & Umid****, e as características do novo sensor são similares aos modelos em separado. Ele possui 3 estrapes (jumpers) para determinar seu endereço na rede I²C:*

	A2	A1	A0
Sensor de Temp & Umid 0	0	0	0
Sensor de Temp & Umid 1	0	0	1
Sensor de Temp & Umid 2	0	1	0
Sensor de Temp & Umid 3	0	1	1
Sensor de Temp & Umid 4	1	0	0
Sensor de Temp & Umid 5	1	0	1
Sensor de Temp & Umid 6	1	1	0
Sensor de Temp & Umid 7	1	1	1

Veja também:
I²C - Umidade

I²C - Umidade

Estes blocos acessam a rede I²C existente no controlador µDX200, permitindo a leitura de sensores de temperatura e umidade. No caso, este bloco lê sensores de umidade instalados na rede I²C (até 8 sensores). A rede I²C possibilita a conexão de sensores distantes, até 1 Km do controlador µDX200 (desde que a fiação esteja instalada em eletroduto metálico exclusivo para conexão de dados).

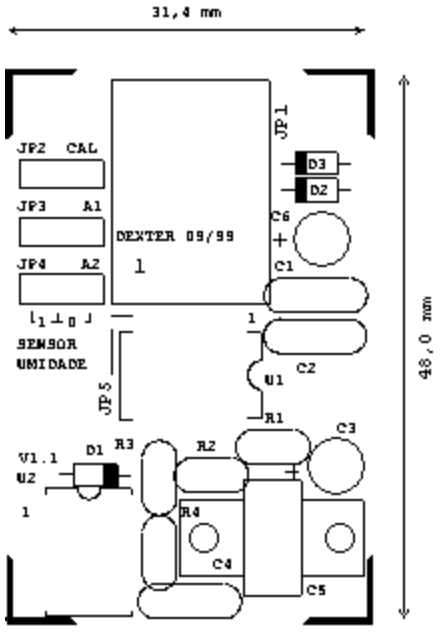


Note que este bloco possui uma conexão inteira de saída (U) que disponibiliza o valor de umidade. Caso não seja especificada nenhuma variável no operando Umidade é preciso usar esta conexão para acessar o valor de umidade do sensor. Se for associada uma variável ao operando a umidade do sensor é constantemente transferida para esta variável, além da variável estar disponibilizada na conexão de saída do bloco.

O operando Umidade indica a variável inteira que irá receber o valor lido de umidade no sensor especificado no operando Endereço. Note que Umidade pode ser variável inteira, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Também pode ser omitida e, neste caso, o valor de umidade estará disponível na conexão de saída do bloco.

Já o operando Endereço é um dado com formato byte e, portanto, deve necessariamente ser uma constante (o µDX200 trata variáveis apenas de 16 bits - inteiro ou word - e 32 bits - longint ou real). Endereço pode assumir valores de 0 a 7, correspondendo aos endereços dos 8 sensores de umidade passíveis de serem instalados na rede I²C do µDX200.

Os sensores de umidade são conectados no mesmo conector RJ11 dos sensores de temperatura. Para conectar vários sensores a DEXTER comercializa uma placa de extensão com 3 derivações. Note que todos os sensores utilizam a mesma linha de comunicação a 4 fios. O que os distingue é o endereço programado via estrapes (jumpers) na placa de cada sensor - 3 jumpers, como no caso do sensor de temperatura, permitindo endereço de 0 a 7. Mas no caso do sensor de umidade os conectores JP2, JP3 e JP4, que possuem estrapes para programação do endereço do sensor, possuem uma ordem contrária de bits em relação ao sensor de temperatura.



Acima temos o layout da placa de sensor de umidade. Note o conector RJ11 central (JP1), que conecta a placa ao controlador µDX200.

Existem 3 jumpers à esquerda (JP2, JP3, JP4), responsáveis pelo endereço do sensor de umidade. JP2 é o bit mais significativo deste endereço e JP4 o menos significativo.

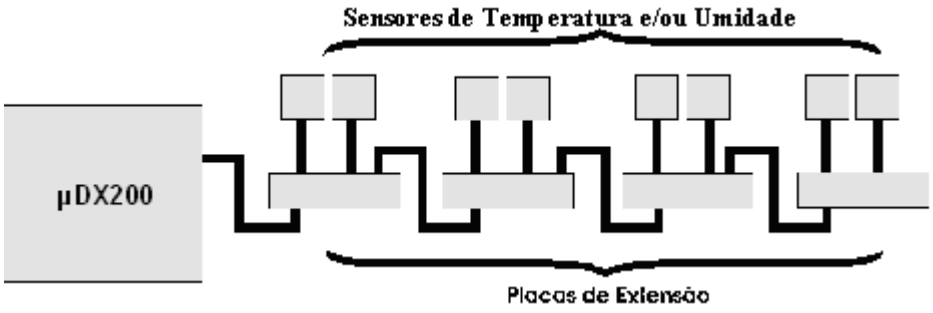
Se o jumper estiver ligado entre o pino à esquerda e o pino central o jumper estará em 1 (um). Já se o jumper estiver entre o pino central e o pino à direita o jumper estará em 0 (zero). A tabela a seguir indica o valor de JP2, JP3 e JP4 para os 8 endereços possíveis do sensor de umidade:

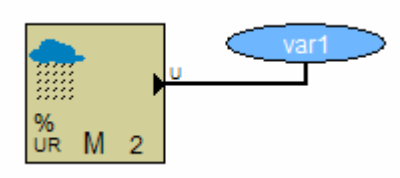
	JP2	JP3	JP4
Sensor de Umidade 0	0	0	0
Sensor de Umidade 1	0	0	1
Sensor de Umidade 2	0	1	0
Sensor de Umidade 3	0	1	1
Sensor de Umidade 4	1	0	0
Sensor de Umidade 5	1	0	1
Sensor de Umidade 6	1	1	0
Sensor de Umidade 7	1	1	1

Note que um endereço usado como sensor de umidade pode possuir sensor de temperatura e vice-versa, ou seja, eles não são mutuamente excludentes. Ou seja, podemos conectar, no máximo, 8 sensores de temperatura e 8 sensores de umidade no mesmo µDX200.

> ***Importante:*** *o sensor de umidade para µDX200 é específico para esta linha de controladores programáveis. Não pode ser usado o sensor de umidade da linha de controladores µDX100.*

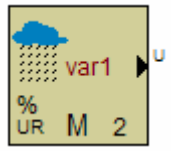
A placa de extensão opcional permite derivar 3 ligações para sensores de temperatura e/ou umidade a partir do cabo conectado ao µDX200. Assim, para conectar 4 sensores de temperatura + 4 sensores de umidade, por exemplo, são necessárias 4 placas de extensão (sendo que uma saída de uma das placas fica vazia):





Exemplo de Programa Aplicativo: **I2C - Umidade.dxg**

Os sensores de umidade permitem leitura de umidade relativa ambiente com faixa de 10 a 90% de UR, e resolução de 0,5% UR. A variável inteira resultante do sensor de umidade representa a umidade ambiente em % UR, multiplicada por 2. Assim, para obtermos o valor de umidade em %UR basta dividir a variável por 2. Por exemplo, digamos que a variável de saída do bloco Umidade assumiu valor 93. Neste caso, a umidade relativa é 93/2 = 46,5% UR.



Os dois exemplos têm efeitos idênticos, ou seja, transferem o valor lido de umidade do sensor dois para a variável var1. Para converter o valor da variável em %UR basta dividir por 2.

Atenção: *o sensor de umidade possui resolução de 0,5% UR, mas sua precisão é de cerca de 5% UR. Para uma calibração mais precisa pode-se somar ou subtrair uma constante (offset) do valor lido via programa aplicativo.*

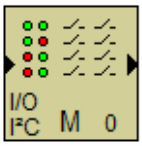
Atenção: *Atualmente a Dexter possui um novo modelo de sensor que incorpora o sensor de temperatura e do de umidade em um único dispositivo. Esse equipamento é designado como* ***Sensor Temp & Umid*** *para µDX201. Cada controlador µDX201 pode ler até 8 sensores* ***Temp & Umid****, e as características do novo sensor são similares aos modelos em separado. Ele possui 3 estrapes (jumpers) para determinar seu endereço na rede I²C:*

	A2	A1	A0
Sensor de Temp & Umid 0	0	0	0
Sensor de Temp & Umid 1	0	0	1
Sensor de Temp & Umid 2	0	1	0
Sensor de Temp & Umid 3	0	1	1
Sensor de Temp & Umid 4	1	0	0
Sensor de Temp & Umid 5	1	0	1
Sensor de Temp & Umid 6	1	1	0
Sensor de Temp & Umid 7	1	1	1

Veja também:
I²C - Temperatura

I²C - I/O

Este bloco acessa a rede I²C existente no controlador µDX200, permitindo a leitura de 8 entradas digitais e acionamento de 8 saídas digitais. Com isso, é disponibilizado um I/O (input/output) remoto, uma vez que a rede I²C permite conectar dispositivos a longa distância. Além disso, esta conexão exige apenas dois pares de fios (um par para alimentação elétrica e outro par para os dados).



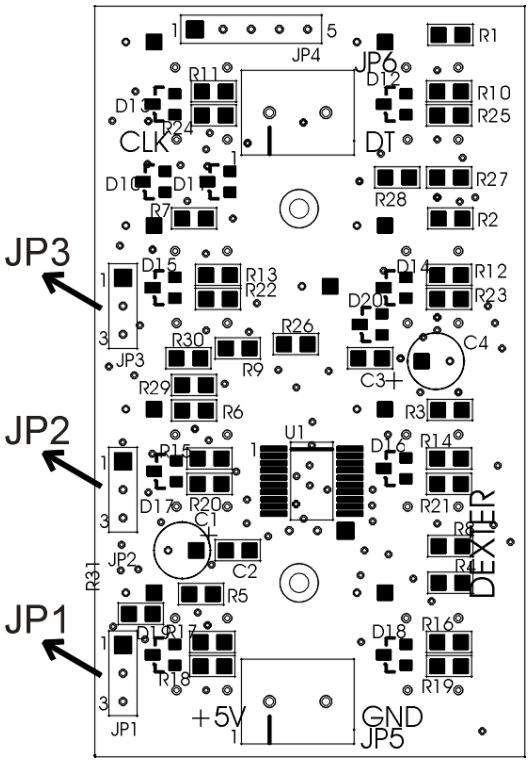
Este bloco é utilizado para leitura de keypad ativo do µDX200. Trata-se de dispositivo com 8 teclas momentâneas e 8 leds indicativos, muito usado em automações residenciais e prediais. Ele substitui as tradicionais chaves liga/desliga para iluminação residencial por teclas momentâneas. A grande vantagem é em termos de fiação, pois com apenas dois pares de fios (rede I^2C) é possível acessar até 8 keypads ativos, perfazendo 64 teclas. Além disso, o keypad funciona com apenas 5V, eliminando o risco de choques elétricos.

Atenção: *Recomenda-se o uso do bloco* ***I/O (16bits)*** *em vez do bloco* ***I/O*** *em novas aplicações, devido a possibilidade de checar as leituras comparando o valor do byte MSB e LSB da variável inteira de saída do bloco. Isso permite maior segurança no uso de Keypads em aplicações residenciais e prediais.*

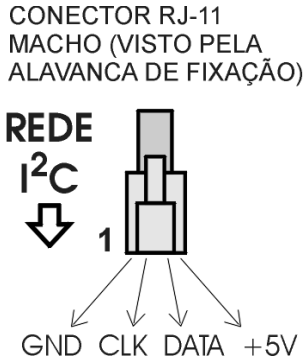
No caso do keypad, o nodo de entrada aciona os leds (conforme o valor dos 8 bits menos significativos da variável utilizada), enquanto o nodo de saída disponibiliza uma variável inteira cujos 8 bits menos significativos refletem o estado das teclas.

O operando Endereço é um dado com formato byte e, portanto, deve necessariamente ser uma constante (o µDX200 trata variáveis apenas de 16 bits - inteiro ou word - e 32 bits - longint ou real). Endereço pode assumir valores de 0 a 7, correspondendo aos endereços dos 8 keypads passíveis de serem instalados na rede I^2C do µDX200.

Para conexão dos keypads existe um conector tipo RJ11 no Controlador Programável µDX200, o que permite conectar um keypad. Para conectar keypads adicionais a DEXTER comercializa uma placa de extensão com 3 derivações. Também pode ser utilizado qualquer derivador para linha telefônica, como os comumente encontrados em lojas de material de informática. Note que todos os keypads utilizam a mesma linha de comunicação a 4 fios. O que os distingue é o endereço programado via estrapes (jumpers) na placa de cada keypad - 3 jumpers, permitindo endereço de 0 a 7.



O Keypad para µDX200 prevê dois conectores para fio, em vez do conector RJ-11 tradicional utilizado nos demais equipamentos Dexter para conexão à rede I^2C. Isso porque o cabo usado para conexão do Keypad ao µDX200 em instalações residenciais e prediais muitas vezes deve ser embutido em eletrodutos, e o conector seria um empecilho. Os conectores para fio permitem parafusar diretamente o cabo (normalmente dois pares de condutores de um cabo UTP cat.5), além de permitir parafusar junto cabos adicionais para os demais Keypads da rede I^2C. A pinagem do conector RJ-11 a ser ligado ao µDX200 é a seguinte:



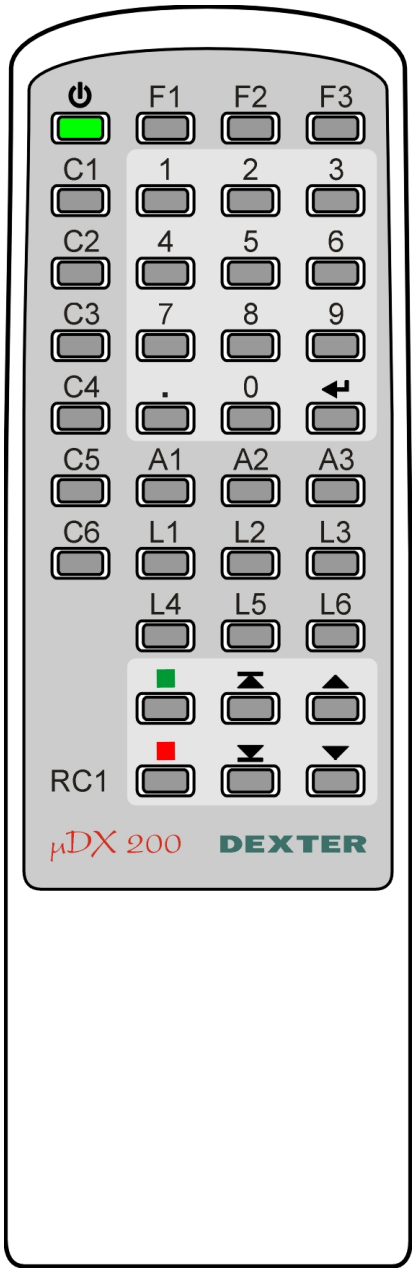
Os jumpers na placa do keypad indicam o endereço do mesmo. Se o jumper estiver ligado entre o pino central (pino 2) e o pino inferior do conector (pino 3) o jumper estará em nível 0 (zero). Já se o jumper estiver conectado entre o pino central (pino 2) e o pino superior do conector (pino 1) o jumper estará em nível 1 (um). Os 8 endereços possíveis para o keypad são determinados pela combinação de JP1, JP2 e JP3:

	JP3	JP2	JP1
Keypad 0	0	0	0
Keypad 1	0	0	1
Keypad 2	0	1	0

Keypad 3	*0*	1	1
Keypad 4	1	*0*	*0*
Keypad 5	1	*0*	1
Keypad 6	1	1	*0*
Keypad 7	1	1	1

Controle Remoto via Infravermelho

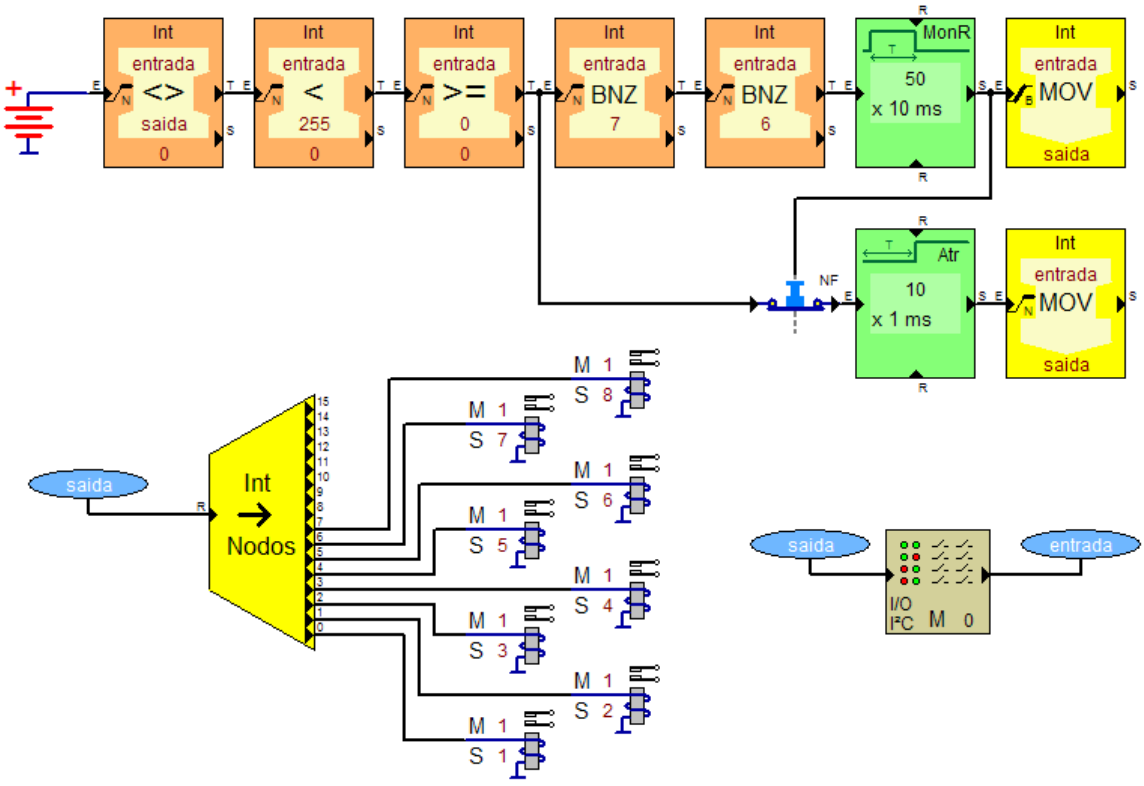
O Keypad Ativo permite leitura de controle remoto por infravermelho. Normalmente apenas um bit de saída está acionado no keypad, correspondendo a tecla que está sendo pressionada. No caso dos comandos por infravermelho os bits 7 e 6 da variável de saída do bloco de I/O I^2C estão sempre ativos. Assim, restam 6 bits para indicação de qual tecla foi pressionada no controle remoto. O controle remoto possui as teclas a seguir:



O controle remoto por infravermelho possui um alcance de cerca de 10 metros. Os códigos correspondentes as teclas do controle remoto são:

Tecla	Valor decimal	Valor Hexadecimal	Valor Binário
⏻	204	0CC	1100 1100
F1	243	0F3	1111 0011
F2	244	0F4	1111 0100
F3	242	0F2	1111 0010
C1	205	0CD	1100 1101
1	193	0C1	1100 0001
2	194	0C2	1100 0010
3	195	0C3	1100 0011
C2	212	0D4	1101 0100
4	196	0C4	1100 0100
5	197	0C5	1100 0101
6	198	0C6	1100 0110
C3	213	0D5	1101 0101
7	199	0C7	1100 0111
8	200	0C8	1100 1000
9	201	0C9	1100 1001
C4	232	0E8	1110 1000
.	206	0CE	1100 1110
0	192	0C0	1100 0000
↵	202	0CA	1100 1010
C5	249	0F9	1111 1001
A1	247	0F7	1111 0111
A2	238	0EE	1110 1110
A3	248	0F8	1111 1000
C6	239	0EF	1110 1111
L1	220	0DC	1101 1100
L2	235	0EB	1110 1011
L3	221	0DD	1101 1101
L4	207	0CF	1100 1111
L5	228	0E4	1110 0100
L6	229	0E5	1110 0101
■	246	0F6	1111 0110
▲	245	0F5	1111 0101
▲	208	0D0	1101 0000
■	225	0E1	1110 0001
▼	224	0E0	1110 0000
▼	209	0D1	1101 0001

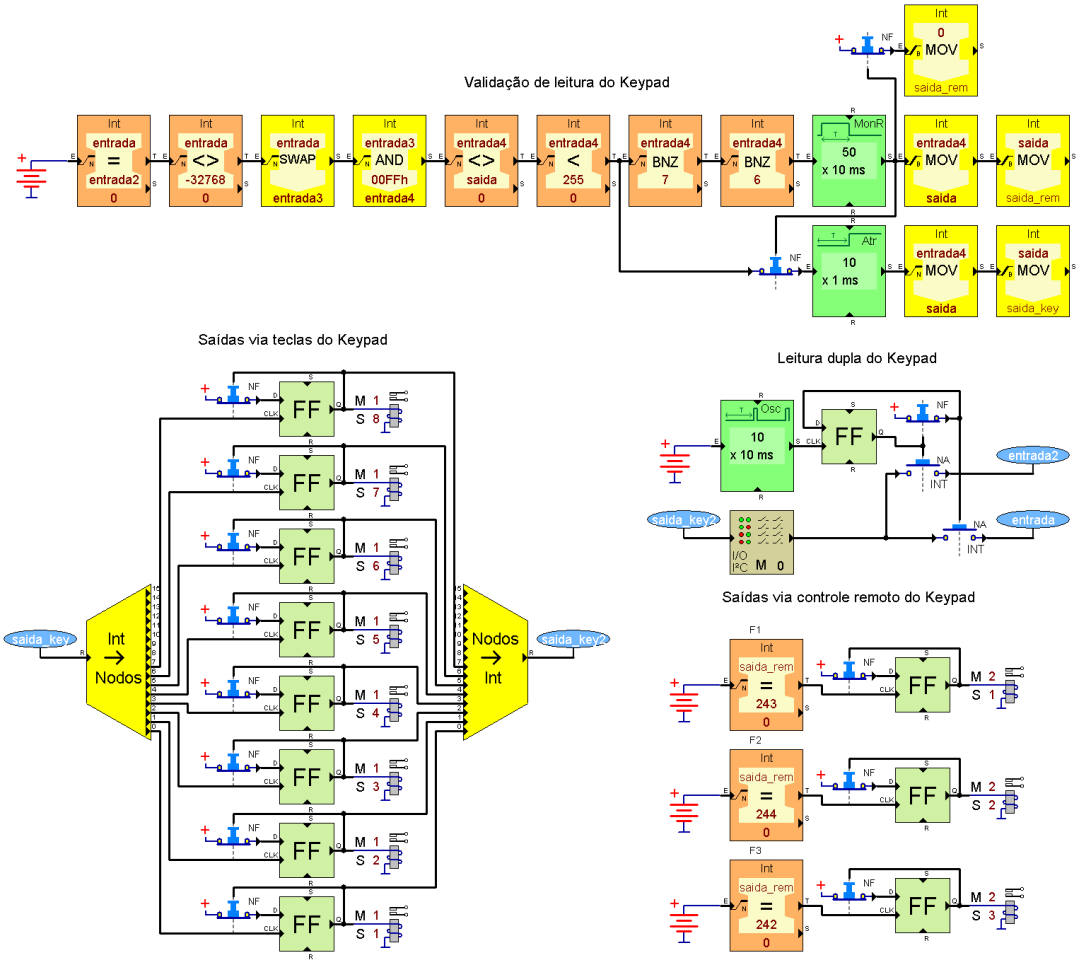
O programa a seguir replica os valores lidos no keypad endereço 0 para a Expansão µDX210 módulo 1. Note que, caso se trate de comando via controle remoto (bits 7 e 6 ligados) o programa retêm os dados durante 0,5 segundo, de forma que as saídas do µDX210 não fiquem piscando a cada comando infravermelho recebido. Já no caso de detecção de teclas pressionadas existe um atraso de 10ms para transmitir o valor detectado para a variável "saída", que irá acionar as saídas da Expansão µDX210 e também os leds do Keypad.



Exemplo de Programa Aplicativo: **I2C - I_O.dxg**

Um problema que pode ocorrer neste tipo de dispositivo é a falsa leitura do Keypad devido a ruídos induzidos na rede I²C. Como a rede pode ser bastante extensa (até 1000 metros) e os comandos não possuem checagem de erros a possibilidade de leituras errôneas é razoável, principalmente em redes de grande comprimento e sujeitas a distúrbios elétricos. Para resolver isso foi gerado um outro bloco de leitura de I/O I²C, chamado I/O (16 bits). Este bloco lê o Keypad em 16 bits em vez de apenas 8 bits. Keypads fabricados a partir de 2009 permitem leitura em 16 bits, sendo que o valor lido nas teclas é replicado tanto no byte superior quanto inferior da variável. Com isso tem-se uma forma eficiente de checagem dos dados, bastando comparar o byte MSB com o byte LSB. Caso sejam iguais o dado é válido; se forem diferentes o dado é inválido. No caso de Keypads antigos, cuja leitura só pode ser efetuada em 8 bits, o uso destes keypads com bloco I²C (16 bits) resulta que o dado fica no byte MSB, e o byte LSB fica sempre com FFh (255). Pode-se remeter estes para que a Dexter efetue uma atualização nos mesmos (este procedimento é gratuito, implicando apenas em custos de transporte para o cliente).

Outra opção para controle de erros, no caso de leitura dos keypads em 8 bits, é exigir via programa aplicativo que duas leituras consecutivas possuam o mesmo valor. O programa abaixo demonstra esta técnica. Note que a cada 100ms a leitura do keypad é comutada entre as variáveis entrada e entrada2. E somente quando entrada=entrada2 o dado é validado (além das demais condições, como dado≥0 e dado<255). As saídas das Expansões µDX210 estão associadas a flip-flops (FF), de forma que a cada pressionar da tecla do keypad elas ligam e desligam alternadamente.

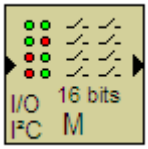


Exemplo de Programa Aplicativo: **I2C - I_O 2.dxg**

Veja também:
I²C - I/O (16 bits)
I²C - OUT
Macros que acompanham o PG

I²C - I/O (16 bits)

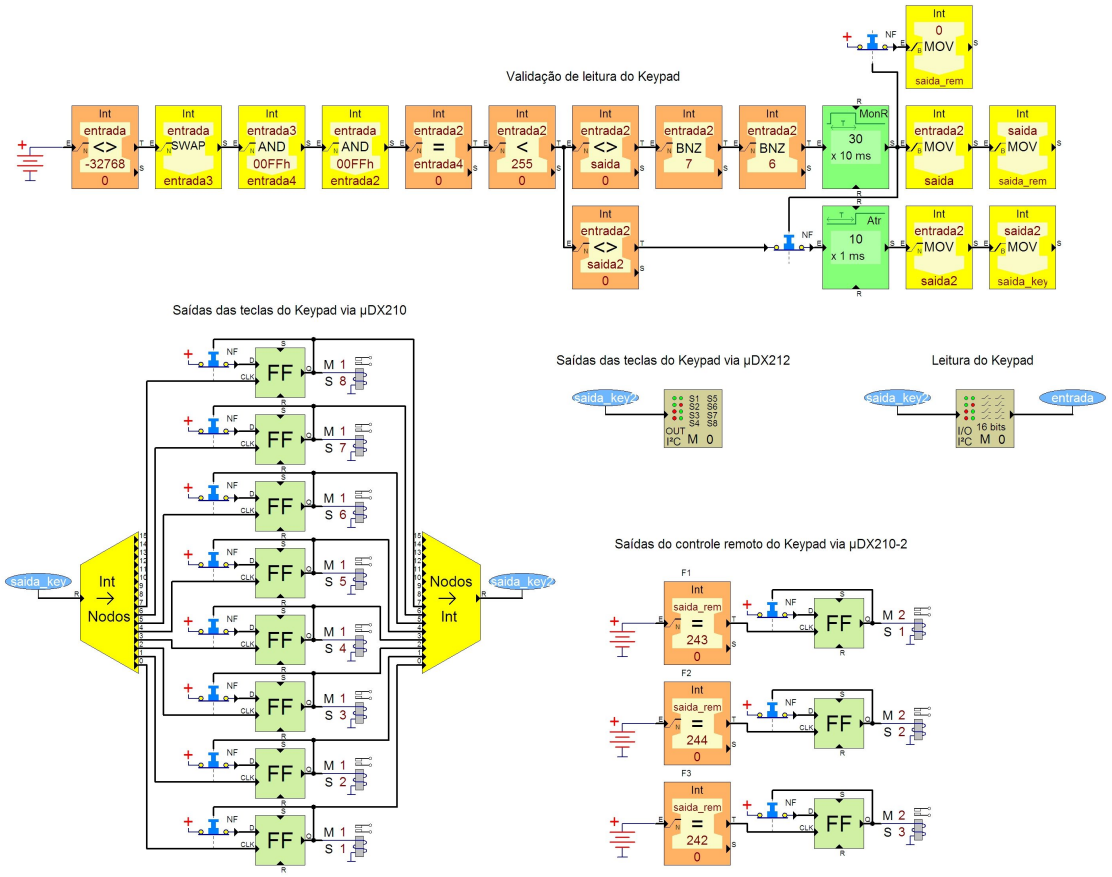
Este bloco acessa a rede I²C existente no controlador µDX200, permitindo a leitura de 8 entradas digitais e acionamento de 8 saídas digitais. Com isso, é disponibilizado um I/O (input/output) remoto, uma vez que a rede I²C permite conectar dispositivos a longa distância. Além disso, esta conexão exige apenas dois pares de fios (um par para alimentação elétrica e outro par para os dados). A única diferença entre este bloco e o bloco anterior (I/O) é que a leitura das 8 entradas digitais é feita em dois bytes replicados. Com isso, é possível checar se o dado lido é consistente, bastando para isso verificar se o byte MSB é idêntico ao byte LSB da leitura.



Este bloco é utilizado para leitura de keypad ativo do µDX200. Para detalhes de programação de endereço do Keypad e teclas do Controle Remoto ver bloco I/O. Recomenda-se enfaticamente o uso deste bloco em vez do bloco anterior para evitar a possibilidade de erros de leitura, no caso de redes I²C extensas. Todos os Keypads fabricados a partir de 2009 permitem a leitura tanto pelo bloco I/O quanto por este bloco I/O (16 bits). No caso de Keypads antigos, cuja leitura só pode ser efetuada em 8 bits, o uso destes keypads com bloco I²C (16 bits) resulta que o dado fica no byte MSB, e o byte LSB fica sempre com FFh (255). Pode-se remeter estes para que a Dexter efetue uma atualização nos mesmos (este procedimento é gratuito, implicando apenas em custos de transporte para o cliente).

Atenção: *Recomenda-se o uso do bloco* ***I/O (16bits)*** *em vez do bloco* ***I/O*** *em novas aplicações, devido a possibilidade de checar as leituras comparando o valor do byte MSB e LSB da variável inteira de saída do bloco. Isso permite maior segurança no uso de Keypads em aplicações residenciais e prediais.*

O programa a seguir ilustra o uso deste bloco. Como o byte MSB também é usado, pode ocorrer do bit mais significativo ser ligado e, portanto, a variável inteira lida assumir valor negativo. Por isso, não é possível fazer um teste se entrada<0 para detectar que keypad foi desconectado da rede I²C (quando ele assume valor -32768, ou FFFFh). É testado explicitamente se entrada<>-32768 para validar o dado. A seguir, é feito um SWAP (transferência do byte MSB para o LSB e vice-versa) de entrada e o resulatado armazenado em entrada3. A seguir, se mascara entrada e entrada3 com 00FFh (ou seja, se mantêm apenas o byte LSB destas variáveis) e se transfere para entrada2 e entrada4. Por fim, se compara entrada2 (que possui o byte LSB lido no keypad) com entrada4 (que possui o byte MSB lido no keypad) para validar o dado. Caso sejam idênticos, se verifica se o dado é diferente do validado anteriormente e se ele é menor que 255. Caso se trate de comando via controle remoto (bits 7 e 6 ligados) o programa retêm os dados durante 0,5 segundo, de forma que a variável saida_rem não fique comutando a cada comando infravermelho recebido. Já no caso de detecção de teclas pressionadas existe um atraso de 10ms para transmitir o valor detectado para a variável saída_key, que irá acionar as saídas da Expansão µDX212 e também as saídas do µDX210 (via flip-flops, de forma que elas comutem de estado a cada comando recebido).



Exemplo de Programa Aplicativo: **I2C - I O (16 bits).dxg**

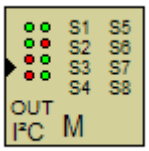
Bloco I²C - I/O (16 bits) CRC

A partir da versão **2.2.2.2** do software Programador Gráfico **PG** para µDX200 estão disponíveis os blocos **I²C CRC**, que possuem a mesma funcionalidade de seus equivalentes blocos de **I²C**, mas transmitem um byte de checagem de erros após o dado, em formato CRC-8. Para usar os blocos **I²C CRC** é necessário controlador programável **µDX201** com firmware versão **3.54** ou superior, e também é mandatório que os periféricos comandados (Mini-Dimmer, µDX212, µDX215, IR-TX, Keypad) sejam de versão recente. Note que o uso de checagem de erros via CRC na rede I²C é aconselhável, em especial quando os cabos de rede passam próximos a cabos de força e outras fontes de ruído elétrico.

Veja também:
I²C - I/O
I²C - OUT
Macros que acompanham o PG

I²C - OUT

Este bloco acessa a rede I²C existente no controlador µDX200, permitindo o acionamento de 8 saídas digitais. Com isso, são disponibilizadas saídas digitais remotas, uma vez que a rede I²C permite conectar dispositivos a longa distância. Além disso, esta conexão exige apenas dois pares de fios (um par para alimentação elétrica e outro par para os dados) e alimentação de potência (12V ou 24V, conforme modelo da placa de saídas digitais via I²C - µDX212).

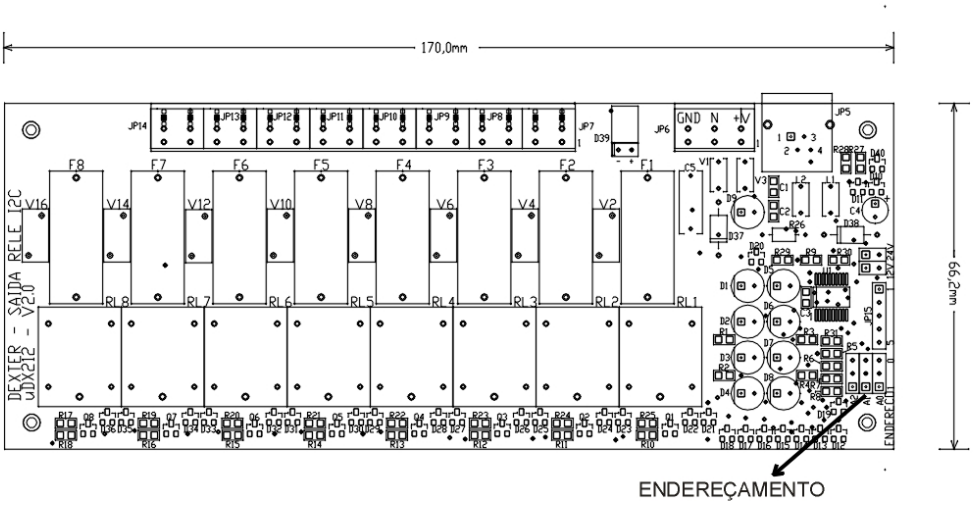


Este bloco é utilizado para acionamento das saídas a relé via I²C - µDX212. Trata-se de dispositivo com 8 relés de potência (10A), utilizado mais comumente em automações residenciais e prediais. Ele substitui as Expansões µDX210 com a vantagem de permitir maior potência de comutação. Além disso, seu perfil esguio permite a instalação em painéis de pouca profundidade (comuns em instalações residenciais). Por segurança, o µDX212 apenas aciona saídas caso receba dois comandos sucessivos com dados idênticos. Isso é feito para evitar que algum dado degradado devido a ruído elétrico venha a acionar indevidamente as saídas.

O nodo de entrada aciona os relés conforme o valor dos 8 bits menos significativos da variável utilizada.

O operando Endereço é um dado com formato byte e, portanto, deve necessariamente ser uma constante (o µDX200 trata variáveis apenas de 16 bits - inteiro ou word - e 32 bits - longint ou real). Endereço pode assumir valores de 0 a 7, correspondendo aos endereços dos 8 µDX212 passíveis de serem instalados na rede I²C do µDX200.

Para conexão dos módulos µDX212 existe um conector tipo RJ11 no Controlador Programável µDX200, o que permite conectar um módulo. Para conectar módulos adicionais a DEXTER comercializa uma placa de extensão com 3 derivações. Também pode ser utilizado qualquer derivador para linha telefônica, como os comumente encontrados em lojas de material de informática. Note que todos os módulos utilizam a mesma linha de comunicação a 4 fios. O que os distingue é o endereço programado via estrapes (jumpers) na placa de cada µDX212 - 3 jumpers, permitindo endereço de 0 a 7. Para abrir a caixa metálica do µDX212 retire os dois parafusos (fenda cruzada) existentes nas laterais da caixa, e force levemente as laterais para que se afastem dos encaixes que a prendem ao fundo da caixa. Cuidado com os leds soldados a placa impressa, de forma a não danificá-los.



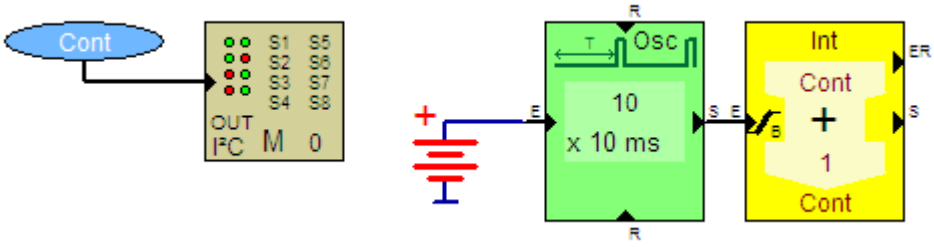
Os jumpers na placa do µDX212 indicam o endereço do mesmo. Se o jumper estiver ligado entre o pino central (pino 2) e o pino inferior do conector (pino 1) o jumper estará em nível 1 (um). Já se o jumper estiver conectado entre o pino central (pino 2) e o pino superior do conector (pino 3) o jumper estará em nível 0 (zero). Os 8 endereços possíveis para o µDX212 são determinados pela combinação de A2, A1 e A0:

	A2	A1	A0
µDX212 0	0	0	0
µDX212 1	0	0	1
µDX212 2	0	1	0
µDX212 3	0	1	1
µDX212 4	1	0	0
µDX212 5	1	0	1
µDX212 6	1	1	0
µDX212 7	1	1	1

A Expansão de Entradas/Saídas µDX212 está equipada com saídas a relés eletromecânicos para alta corrente. Cada saída NA (normalmente aberta) está disponível em conector de engate rápido capaz de suportar estas correntes. As especificações das saídas (S1 a S8) são as seguintes:

Saída Relé	30Vdc @ 10A
	250Vac @ 10A
	Fusível 10A Interno

Os fusíveis utilizados são do tipo com corpo de vidro, de 20mm de comprimento por 5 mm de diâmetro.



Exemplo de Programa Aplicativo: **I2C - Out.dxg**

O exemplo acima simplesmente incrementa a variável Cont a cada 0,1 segundos, e aciona as saídas do módulo µDX212 conforme o valor dos 8 bits inferiores desta variável. Portanto, nas saídas do µDX212 teremos uma contagem binária a uma taxa de 10 Hz.

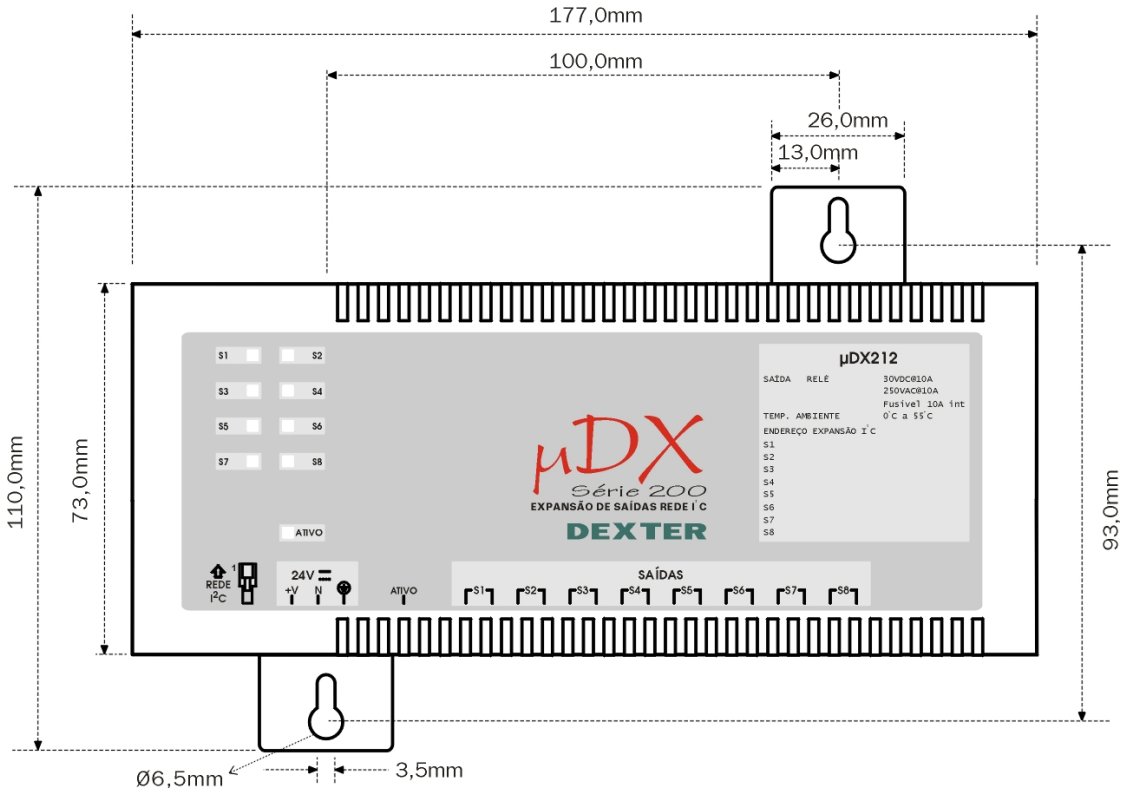
Atenção: *A Expansão de Entradas/Saídas* ***µDX212*** *é fornecida com duas tensões de alimentação elétrica: 12V e 24V. Jumpers internos determinam esta tensão no que se refere ao circuito de controle, mas os relés da unidade são fixos em uma ou outra tensão. Assim, não é possível modificar a tensão de alimentação do equipamento apenas modificando estes jumpers. Especifique a tensão de alimentação ao efetuar o pedido:* ***µDX212*** *(tensão de 24Vdc ± 10%) ou* ***µDX212-12*** *(tensão de 12Vdc ± 10%).*

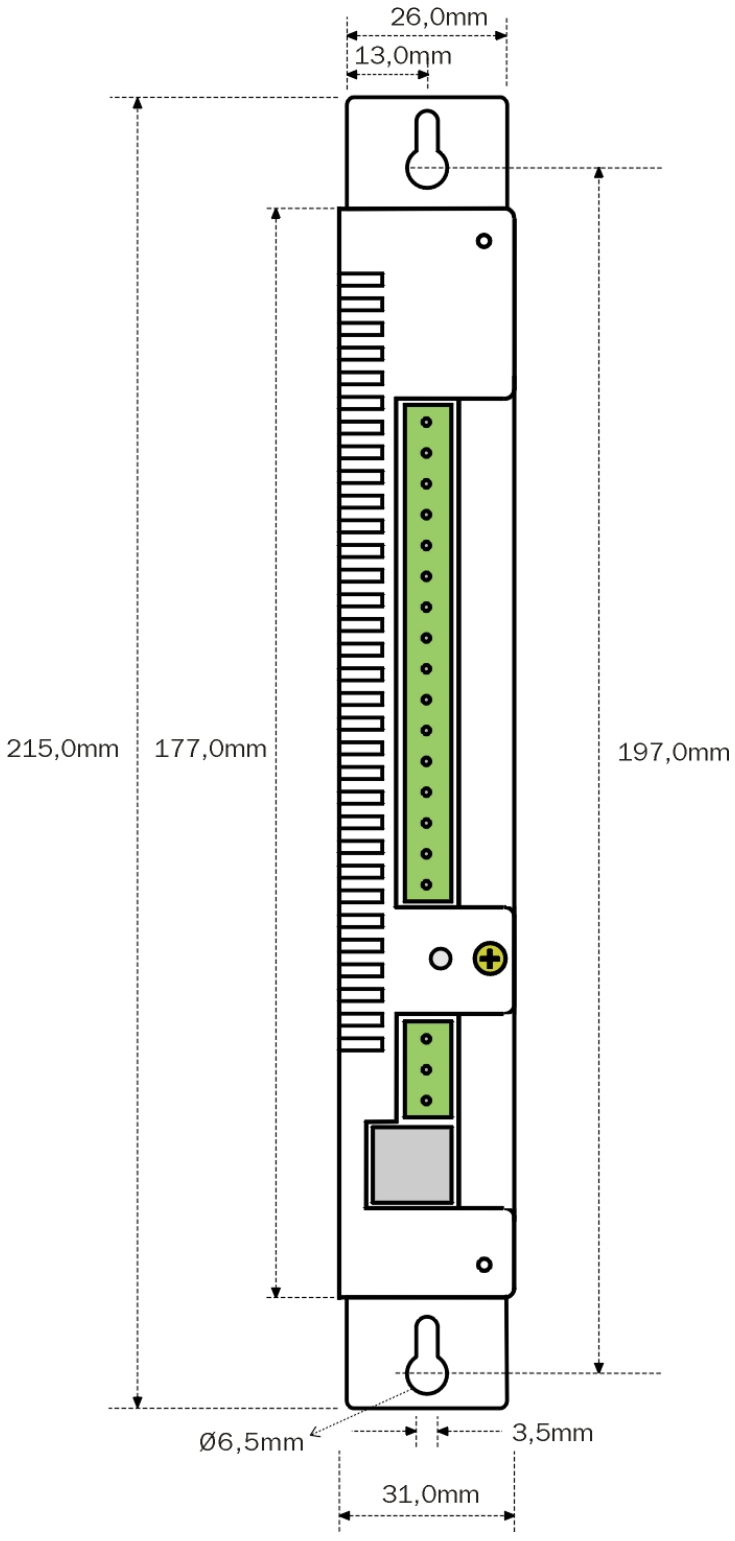
Fixação mecânica da Expansão µDX212 e µDX212-12

- Fixação pelo fundo do µDX212
  Perfil baixo, próprio para gabinetes de pouca profundidade.
  Profundidade necessária = 35mm

- Fixação pela lateral do µDX212
  Pouca área ocupada, mas necessita gabinetes com certa profundidade.
  Profundidade necessária = 100mm

  Pouca área ocupada, mas necessita gabinetes com certa profundidade.
  Profundidade necessária = 100mm





Bloco I²C - OUT CRC

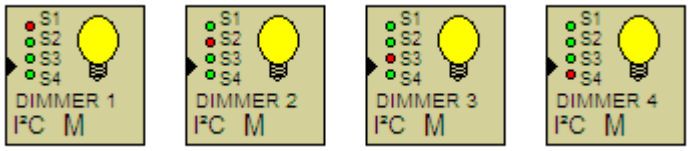
A partir da versão **2.2.2.2** do software Programador Gráfico **PG** para µDX200 estão disponíveis os blocos **I²C CRC**, que possuem a mesma funcionalidade de seus equivalentes blocos de **I²C**, mas transmitem um byte de checagem de erros após o dado, em formato CRC-8. Para usar os blocos **I²C CRC** é necessário controlador programável **µDX201** com firmware versão **3.54** ou superior, e também é mandatório que os periféricos comandados (Mini-Dimmer, µDX212, µDX215, IR-TX,

Keypad) sejam de versão recente. Note que o uso de checagem de erros via CRC na rede I²C é aconselhável, em especial quando os cabos de rede passam próximos a cabos de força e outras fontes de ruído elétrico.

Veja também:
I²C - I/O
I²C - I/O (16 bits)
Macros que acompanham o PG

I²C - Dimmer

Estes blocos acessam saídas moduladas por fase da rede elétrica (dimmer) via rede I²C. Cada Dimmer I²C possui 4 saídas, acessíveis pelos blocos abaixo:



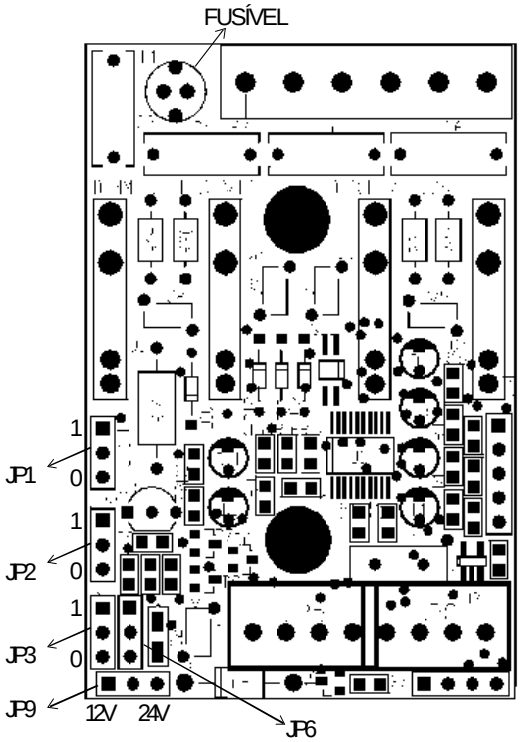
Note que, neste caso, um mesmo Dimmer I²C é acessado por 4 blocos no programa aplicativo; um bloco para cada canal. Como são permitidos até 8 Dimmers na rede I²C simultaneamente, é possível acionar até 32 canais através de um único Controlador µDX200. Com isso, são disponibilizadas saídas com dimmer remotas, uma vez que a rede I²C permite conectar dispositivos a longa distância. Além disso, esta conexão exige apenas dois pares de fios (um par para alimentação elétrica e outro par para os dados). A alimentação elétrica pode ser selecionada entre 12Vdc e 24Vdc via estrape (jumper) interno no Dimmer I²C. No caso de 12V são permitidas tensões de 9V até 15V, enquanto que em 24V os limites se situam entre 21V e 27V. Portanto, mesmo quedas de tensão substanciais na cablagem são admissíveis. Além disso, seu consumo é baixo (cerca de 30mA).

O Dimmer I²C possui 4 relés de estado sólido, capazes de comandar até 220Vac @ 1,2A em cada canal (260W por canal), e é utilizado mais comumente em automações residenciais e prediais. Ele substitui o Dimmer comandado pelas saídas analógicas do µDX200, desde que a potência seja baixa. Seu tamanho reduzido permite a instalação em caixas para alvenaria 4x2 de boa qualidade (Pial-Legrand ou Tigre, por exemplo). Por segurança, o Dimmer I²C apenas aciona saídas caso receba dois comandos sucessivos com dados idênticos. Isso é feito para evitar que algum dado degradado devido a ruído elétrico venha a acionar indevidamente as saídas.

O nodo de entrada dos blocos aciona os relés de estado sólido conforme o valor dos 8 bits menos significativos da variável utilizada.

O operando Endereço é um dado com formato byte e, portanto, deve necessariamente ser uma constante (o µDX200 trata variáveis apenas de 16 bits - inteiro ou word - e 32 bits - longint ou real). Endereço pode assumir valores de 0 a 7, correspondendo aos endereços dos 8 Dimmers passíveis de serem instalados na rede I²C do µDX200.

Para conexão dos módulos Dimmer existe um conector tipo RJ11 no Controlador Programável µDX200, o que permite conectar um módulo. Para conectar módulos adicionais a DEXTER comercializa uma placa de adaptação que transforma a conexão RJ11 em conexão por parafuso. Note que todos os módulos utilizam a mesma linha de comunicação a 2 fios. O que os distingue é o endereço programado via estrapes (jumpers) na placa de cada Dimmer - 3 jumpers, permitindo endereço de 0 a 7. Para abrir a caixa metálica do Dimmer I²C force levemente as laterais para que se afastem dos encaixes que a prendem ao fundo da caixa. Cuidado com os leds soldados a placa impressa, de forma a não danificá-los.



Os jumpers JP1, JP2 e JP3 da placa do Dimmer indicam o endereço do mesmo. Se o jumper estiver ligado entre o pino central (pino 2) e o pino superior do conector (pino 1) o jumper estará em nível 1 (um). Já se o jumper estiver conectado entre o pino central (pino 2) e o pino inferior do conector (pino 3) o jumper estará em nível 0 (zero). Os 8 endereços possíveis para o Dimmer I^2C são determinados pela combinação de JP1, JP2 e JP3:

	JP3	JP2	JP1
Dimmer 0	0	0	0
Dimmer 1	0	0	1
Dimmer 2	0	1	0
Dimmer 3	0	1	1
Dimmer 4	1	0	0
Dimmer 5	1	0	1
Dimmer 6	1	1	0
Dimmer 7	1	1	1

O jumper JP9 permite selecionar a tensão de alimentação: com o jumper colocado entre o pino à esquerda (pino 1) e o pino central (pino 2) a alimentação é 12V (9 a 15Vdc). Já com o jumper entre o pino central.(pino 2) e o pino à direita (pino 3) a alimentação elétrica é em 24V (21 a 27Vdc). Por segurança os Dimmers são fornecidos de fábrica sempre com o jumper para 24V.

O jumper JP6 permite comutar os canais 1 e 2 do Dimmer com os canais 3 e 4, e vice-versa. Isso permite, em uma emergência (devido a queima de um canal, por exemplo) simplesmente substituir as saídas por outras duas ociosas ou menos importantes.

O fusível é comum para todas as saídas e tem capacidade para 5A. Cada saída pode suprir até 1,2A. Isso permite potências por canal de até 150W em 127Vac, e de 260W em 220Vac.

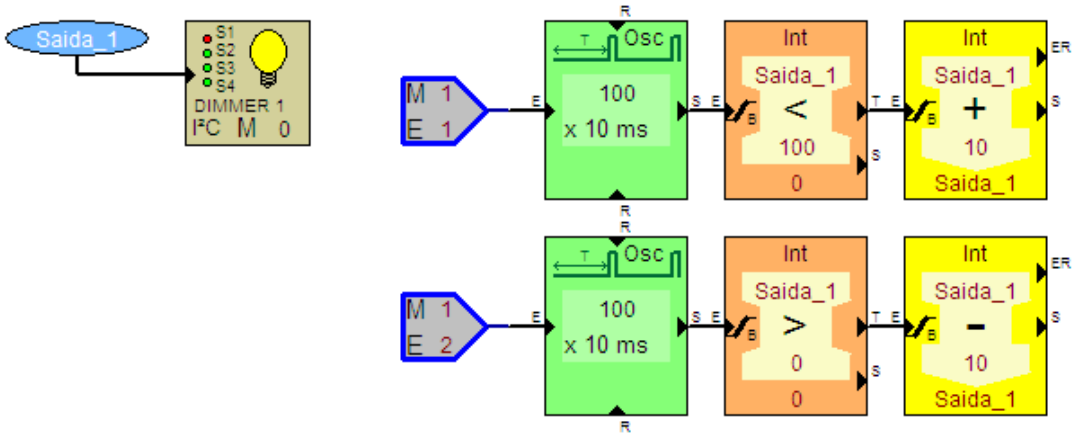
O conector de Saídas disponibiliza a conexão dos quatro canais, e também a conexão de Fase (F) e Neutro (N). Caso seja instalação em 220V normalmente esta tensão é entre fases. Neste caso uma das fases é ligada ao borne F e a outra ao borne N. Quando a alimentação de 127Vac ou 220Vac está presente um led é acionado no Dimmer (led designado como AC).

O conector de Entradas possui bornes para sinais I^2C (dados e clock), alimentação elétrica (12 ou

24Vdc), e também 4 bornes para acionamento de Dimmers de potência, caso necessário. Assim, se a potência disponibilizada pelo Dimmer I²C seja insuficiente, pode-se conectar Dimmers de potência (ver lista de produtos Dexter para linha µDX200) comandados pelos canais do Dimmer I²C.

O Dimmer I²C aceita valores na variável de entrada dos blocos **I²C - Dimmer** entre 0 e 100, correspondendo a iluminação entre 0 e 100%. O próprio Dimmer possui uma tabela de compensação interna, de forma que o valor de percentual especificado na variável corresponda a taxa de iluminação percentual em lux (considerando resposta espectral do olho humano e acionamento de lâmpadas incandescentes). Qualquer transição de luminosidade obedece a uma rampa, de forma a evitar bruscas mudanças de luminosidade, aumentando o conforto visual.

O exemplo abaixo utiliza as entradas E1 e E2 da Expansão de Entradas/Saídas µDX210 para incrementar ou decrementar em 10% a luminosidade do canal 1 do Dimmer I²C:



Exemplo de Programa Aplicativo: **I2C - Dimmer.dxg**

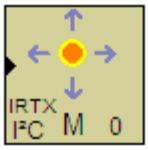
Bloco I²C - Dimmer CRC

A partir da versão **2.2.2.2** do software Programador Gráfico **PG** para µDX200 estão disponíveis os blocos **I²C CRC**, que possuem a mesma funcionalidade de seus equivalentes blocos de **I²C**, mas transmitem um byte de checagem de erros após o dado, em formato CRC-8. Para usar os blocos **I²C CRC** é necessário controlador programável **µDX201** com firmware versão **3.54** ou superior, e também é mandatório que os periféricos comandados (Mini-Dimmer, µDX212, µDX215, IR-TX, Keypad) sejam de versão recente. Note que o uso de checagem de erros via CRC na rede I²C é aconselhável, em especial quando os cabos de rede passam próximos a cabos de força e outras fontes de ruído elétrico.

Veja também:
Macros que acompanham o PG

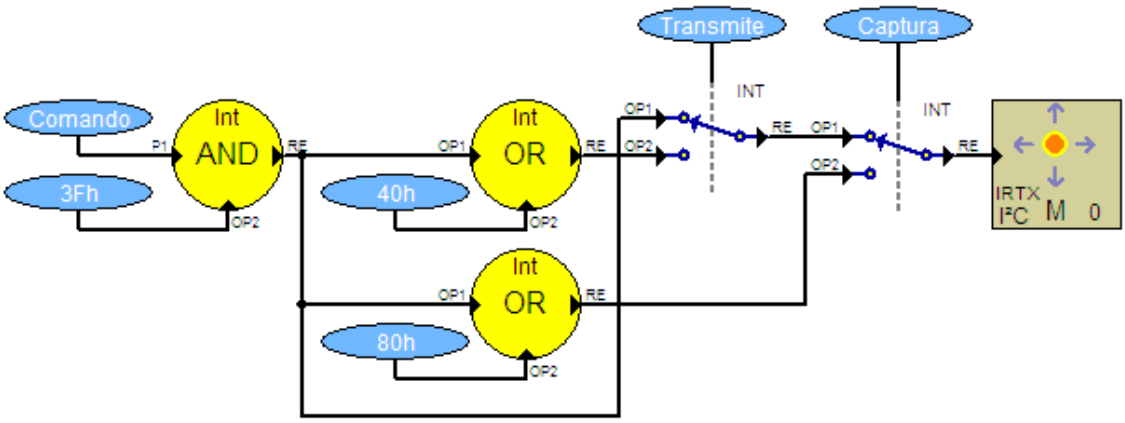
I²C - IR TX

O bloco IR-TX I²C permite acessar módulos de Transmissores Infravermelhos conectados à rede I²C do controlador µDX200 (ou µDX201). O Transmissor de Infravermelho (IR-TX) permite o aprendizado de até 57 comandos infravermelhos de vários equipamentos, e sua reprodução comandada pelo controlador µDX200. Assim, é possível comandar sistemas de refrigeração, áudio e vídeo via programa aplicativo do CLP. A aplicação mais comum é em automações residenciais e prediais.



A conexão de entrada do bloco deve ser ligada a uma variável inteira. O byte superior desta variável é irrelevante. Os bits 0 a 5 permitem escolher qual dos comandos deve ser selecionado, o bit 6 aciona a transmissão, e o bit 7 aciona a captura de comandos infravermelhos.

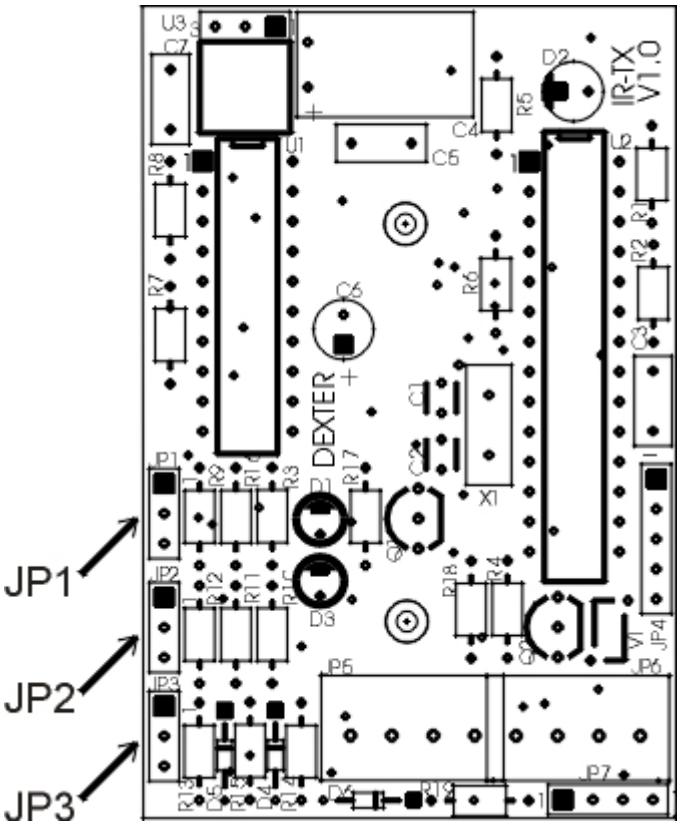
Para selecionar o comando infravermelho desejado utiliza-se os 6 bits menos significativos (bit 0 a bit 5) da variável usada na entrada do bloco IR-TX I²C. É possível selecionar comando 0 (000000 em binário) até comando 56 (111000 em binário). Já o bit 6 aciona a transmissão do comando selecionado pelos 6 bits menos significativos. Por fim, o bit 7 aciona a captura de uma transmissão infravermelha e a armazena no endereço especificado nos 6 bits menos significativos. Note que os comandos infravermelhos são armazenados em memória não-volátil, de forma que não são perdidos no caso de interrupção no fornecimento de energia elétrica.
Para transmitirmos o comando 2, por exemplo, é preciso atribuir valor 66 (ou seja, 42 em hexadecimal, ou 01000010 em binário) a variável de entrada do bloco IR-TX. Já para armazenarmos um comando na posição 12 devemos fazer esta variável assumir valor 140 (ou seja, 8C em hexadecimal, ou 10001100 em binário).
O programa a seguir exemplifica o uso do bloco IR-TX I²C:



Exemplo de Programa Aplicativo: **I2C - IRTX.dxg**

O endereço do comando no Transmissor Infravermelho é especificado pela variável Comando. Note que é efetuada uma operação AND desta variável com 3Fh, de forma a manter apenas os 6 bits menos significativos. A seguir, gera-se outros dois valores, um com o bit 6 da variável ligado (para transmitir os comandos infravermelhos) e outro com o bit 7 ligado (para capturar comandos infravermelhos). Ao acionar o nodo Transmite o comando infravermelho previamente armazenado na posição de memória especificada pela variável Comando é transmitido. Já ao acionar o nodo Captura o Transmissor Infravermelho aguarda a recepção de um comando infravermelho e o armazena na posição de memória determinada pela variável Comando.
Ao editar o bloco IR-TX I²C é possível indicar o endereço do transmissor na rede I²C (até 8

transmissores; de endereço 0 até endereço 7). Entretanto, o endereço no módulo deve coincidir com o utilizado no bloco do programa aplicativo do µDX200. Para modificar o endereço do módulo é preciso acessar sua placa impressa e modificar estrapes (jumpers) internos. Para abrir a caixa metálica do Transmissor Infravermelho I²C force levemente as laterais para que se afastem dos encaixes que a prendem ao fundo da caixa. Cuidado com os leds soldados a placa impressa, de forma a não danificá-los.



Os jumpers JP1, JP2 e JP3 da placa do Transmissor Infravermelho indicam o endereço do mesmo. Se o jumper estiver ligado entre o pino central (pino 2) e o pino superior do conector (pino 1) o jumper estará em nível 1 (um). Já se o jumper estiver conectado entre o pino central (pino 2) e o pino inferior do conector (pino 3) o jumper estará em nível 0 (zero). Os 8 endereços possíveis para o Transmissor Infravermelho I²C são determinados pela combinação de JP1, JP2 e JP3:

	JP3	**JP2**	**JP1**
IR-TX 0	0	0	0
IR-TX 1	0	0	1
IR-TX 2	0	1	0
IR-TX 3	0	1	1
IR-TX 4	1	0	0
IR-TX 5	1	0	1
IR-TX 6	1	1	0
IR-TX 7	1	1	1

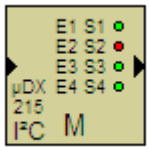
Bloco I²C - IR TX CRC

A partir da versão **2.2.2.2** do software Programador Gráfico **PG** para µDX200 estão disponíveis os blocos **I²C CRC**, que possuem a mesma funcionalidade de seus equivalentes blocos de **I²C**, mas

transmitem um byte de checagem de erros após o dado, em formato CRC-8. Para usar os blocos **I²C CRC** é necessário controlador programável **µDX201** com firmware versão **3.54** ou superior, e também é mandatório que os periféricos comandados (Mini-Dimmer, µDX212, µDX215, IR-TX, Keypad) sejam de versão recente. Note que o uso de checagem de erros via CRC na rede I²C é aconselhável, em especial quando os cabos de rede passam próximos a cabos de força e outras fontes de ruído elétrico.

I²C - µDX215

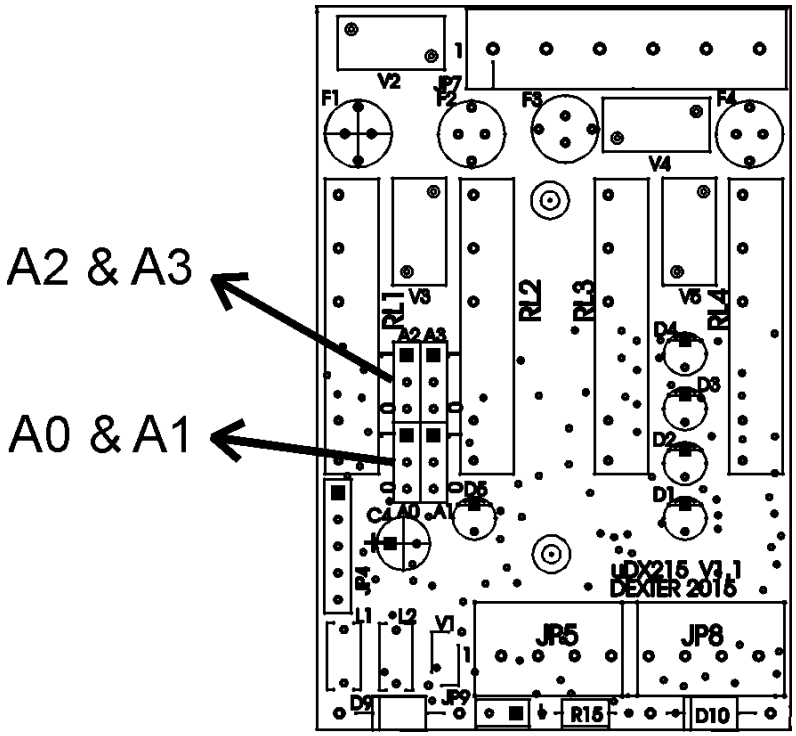
O bloco µDX215 I²C acessa dois módulos µDX215 ligados a rede I²C do controlador µDX200. Cada µDX215 possui 4 entradas digitais (para leitura de pulsadores) e 4 saídas à relé (capacidade máxima de 5A).



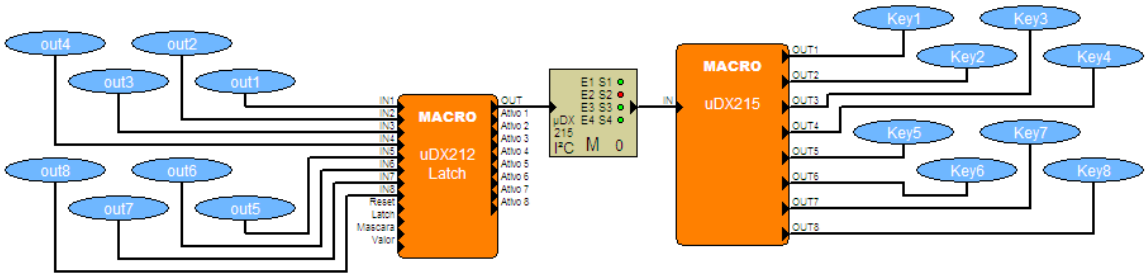
Note que o µDX215 possui quatro jumpers para endereçamento, permitindo até 16 módulos ligados a um único controlador µDX200. Os jumpers A2,A1,A0 indicam o endereço do µDX215 na rede I²C, enquanto o jumper A3 indica se o µDX215 ocupa os quatro bits inferiores (A3=0) ou os quatro bits superiores (A3=1), como especificado na tabela abaixo:

	A3	A2	A1	A0
µDX215 0 (LSB)	0	0	0	0
µDX215 0 (MSB)	1	0	0	0
µDX215 1 (LSB)	0	0	0	1
µDX215 1 (MSB)	1	0	0	1
µDX215 2 (LSB)	0	0	1	0
µDX215 2 (MSB)	1	0	1	0
µDX215 3 (LSB)	0	0	1	1
µDX215 3 (MSB)	1	0	1	1
µDX215 4 (LSB)	0	1	0	0
µDX215 4 (MSB)	1	1	0	0
µDX215 5 (LSB)	0	1	0	1
µDX215 5 (MSB)	1	1	0	1
µDX215 6 (LSB)	0	1	1	0
µDX215 6 (MSB)	1	1	1	0
µDX215 7 (LSB)	0	1	1	1
µDX215 7 (MSB)	1	1	1	1

A posição dos jumpers na placa impressa é a seguinte:



O programa abaixo exemplifica o uso de dois µDX215 no endereço 0 da rede I²C. Observe que para decodificar as entradas digitais do µDX215 usamos a macro µDX215. Já para o acionamento das saídas do módulo µDX215 usamos uma macro µDX212 Latch, como é feito no caso de uso de módulos µDX212. As entradas do µDX215 devem ser comutadas para zero ao pressionar-se o pulsador. Ao contrário do módulo Multiplexador, não há restrição quanto ao uso de pulsadores para alta tensão, pois é admitida uma resistência de contato bastante elevada (mais de 2000Ω).



Exemplo de Programa Aplicativo: **Teste_uDX215.dxg**

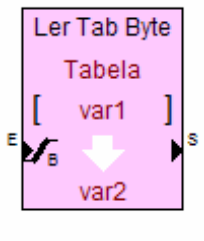
Atenção: As mesmas orientações acima são aplicadas para o módulo µDX216. A única diferença em relação à Expansão µDX215 é quanto a natureza das saídas: enquanto o µDX215 utiliza relés eletromecânicos, o µDX216 utiliza relés de estado sólido com comutação pela passagem do zero da rede elétrica. No caso de acionamento de cargas capacitivas, como luminárias LED, é mandatório o uso de Expansão µDX216, pois os relés eletromecânicos usados no µDX215 não suportam as altas correntes de partida destes dispositivos.

Bloco I²C - µDX215 CRC

A partir da versão **2.2.2.2** do software Programador Gráfico **PG** para µDX200 estão disponíveis os blocos **I²C CRC**, que possuem a mesma funcionalidade de seus equivalentes blocos de **I²C**, mas transmitem um byte de checagem de erros após o dado, em formato CRC-8. Para usar os blocos **I²C CRC** é necessário controlador programável **µDX201** com firmware versão **3.54** ou superior, e também é mandatório que os periféricos comandados (Mini-Dimmer, µDX212, µDX215, IR-TX, Keypad) sejam de versão recente. Note que o uso de checagem de erros via CRC na rede I²C é aconselhável, em especial quando os cabos de rede passam próximos a cabos de força e outras fontes de ruído elétrico.

TABELA - Ler Tabela Byte

Este bloco permite ler posições de uma tabela de bytes (0 a 255).

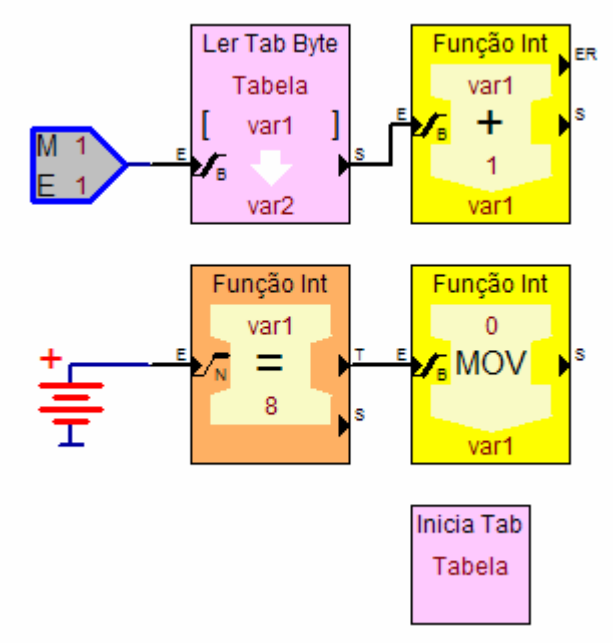


Como o µDX200 não trabalha com bytes, o valor lido é colocado no byte inferior (LSB) de uma variável inteira.

Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser lida na memória de programa (flash) ou na memória volátil (RAM). Índice indica a posição a ser lida da tabela (de zero à posição final da tabela menos um). Já Destino recebe o valor lido na posição de tabela especificado.

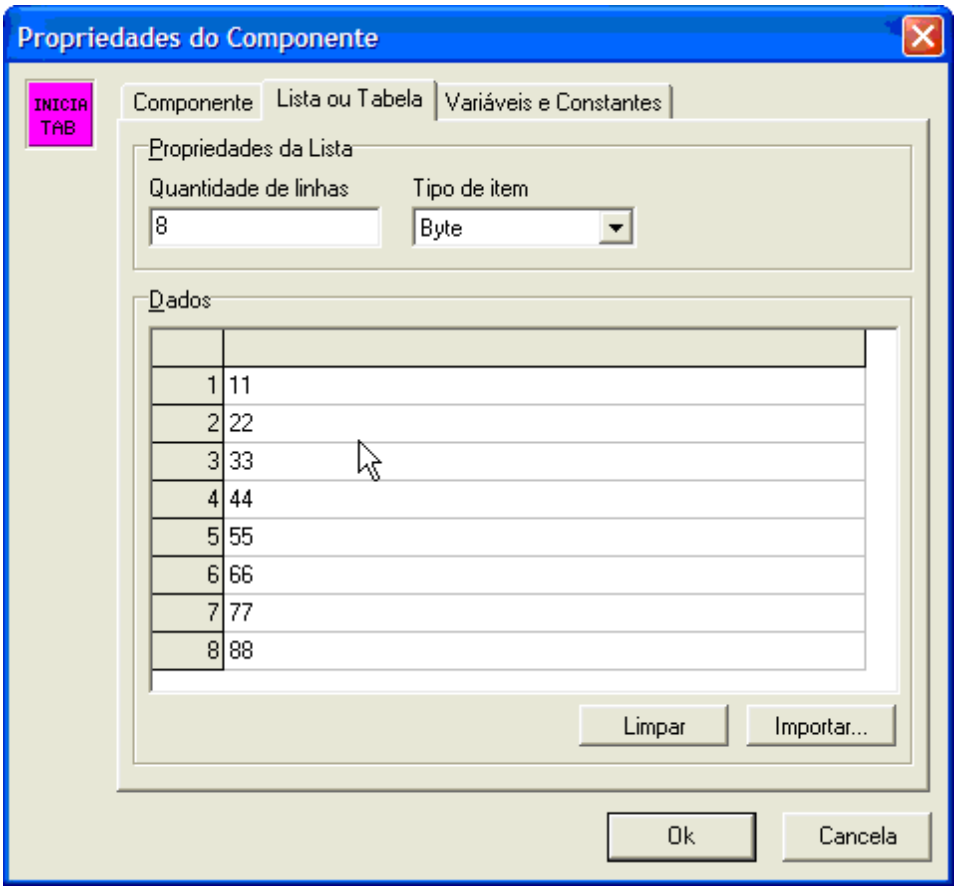
Note que Tabela e Índice podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Destino deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Tabela - Ler Tabela Byte.dxg**

O programa de exemplo acima lê as 8 posições da tabela byte e as coloca em var2. Cada vez que a entrada E1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) é energizada é lida a posição da tabela apontada por var1 e o resultado é colocado em var2. A seguir é incrementado o índice (var1). Note que este índice inicia no valor 0 e é incrementado até 7. Ao atingir valor 8 ele é novamente inicializado com valor 0. Os valores inicializados na tabela via bloco de Inicia Tab são:



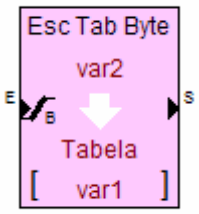
Importante: *A primeira posição de tabelas no µDX200 é acessada com variável Índice no valor zero, ou seja, o Índice indica a posição da tabela a ser lida a partir da primeira posição. Portanto, o Índice deverá assumir valores entre zero e tamanho da tabela menos um. No caso de tabelas de 8 posições, como no exemplo, Índice deve ser restrito a valores entre 0 e 7.*

Atenção: *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Veja também:
TABELA - Ler Tabela Inteira
TABELA - Ler Tabela LongInt
TABELA - Ler Tabela Word
TABELA - Ler Tabela Real

TABELA - Escrever Tabela Byte

Este bloco permite escrever em posições de uma tabela de bytes (0 a 255).

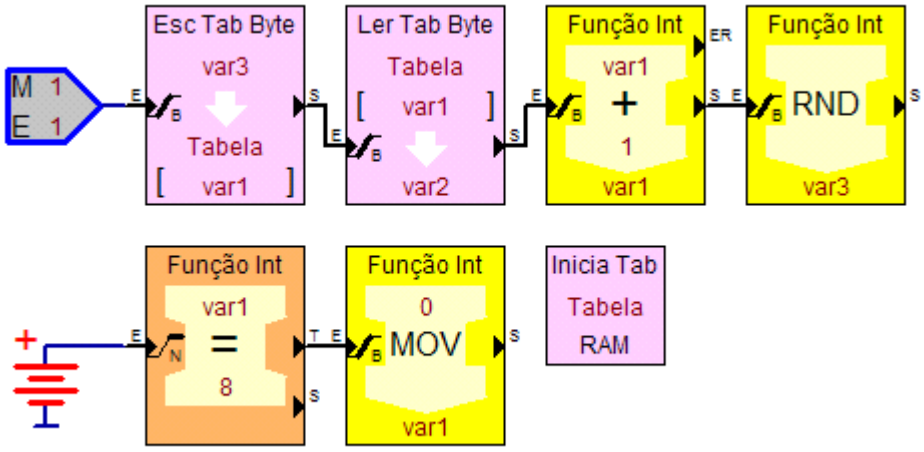


Naturalmente esta tabela deve estar em memória volátil (RAM), de forma a permitir escrita. Como o µDX200 não trabalha com bytes, o valor a ser escrito é retirado do byte inferior (LSB) de uma variável inteira.

Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser escrita na memória volátil (RAM). Índice indica a posição a ser escrita na tabela (de zero à posição final da tabela menos um). Já Origem é o valor a ser escrito na posição de tabela especificado.

Note que Origem e Índice podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Tabela deve ser necessariamente um valor word que aponta para uma tabela em RAM, uma vez que irá receber o resultado da operação.

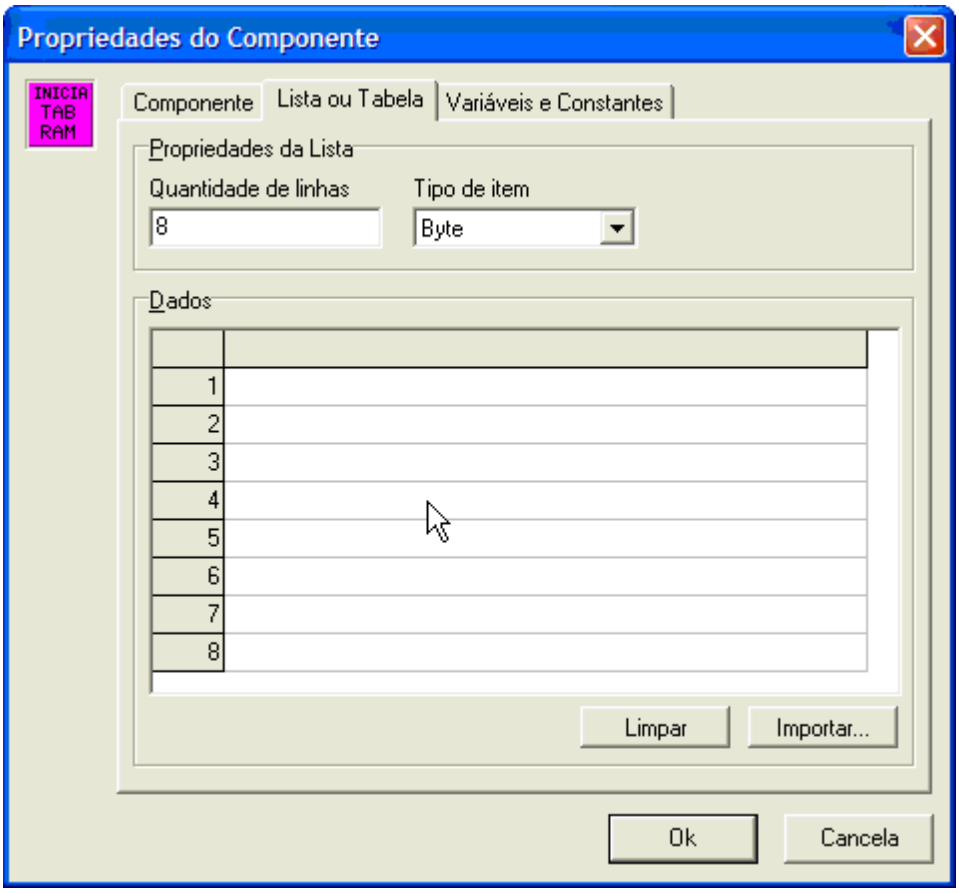
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Tabela - Escrever Tabela Byte.dxg**

O programa de exemplo escreve a variável var3 em posições sucessivas da tabela (apontada por var1). A seguir, esta mesma posição é lida e colocada em var2. Por fim, o índice var1 é incrementado e em var3 é colocado um número randômico inteiro. Note que como se trata de uma tabela byte, apenas o byte LSB de var3 é salvo. Assim, não necessariamente o valor salvo na tabela irá coincidir com o valor de var3. Note que o índice var1 é inicializado com valor zero (para apontar o primeiro elemento da tabela), e é incrementado até 7. Quando assume valor 8 é reinicializado para valor zero (já que a tabela possui apenas 8 posições).

Como este programa escreve na tabela é necessário que se trate de uma tabela em RAM, pois tabelas em memória de programa (flash) não permitem gravação de novos valores. No caso de tabelas em RAM o µDX200 não admite inicialização de valores via bloco de Inicia Tab RAM, como no caso de tabelas em memória de programa (flash). Neste caso todas as posições da tabela são inicializadas com valor zero (como as variáveis do µDX200), independentemente dos valores especificados para cada posição da tabela no bloco Inicia Tab RAM. A edição do bloco INICIA Tab RAM é a seguinte:



Importante: *A primeira posição de tabelas no µDX200 é acessada com variável Índice no valor zero, ou seja, o Índice indica a posição da tabela a ser lida a partir da primeira posição. Portanto, o Índice deverá assumir valores entre zero e tamanho da tabela menos um. No caso de tabelas de 8 posições, como no exemplo, Índice deve ser restrito a valores entre 0 e 7.*

Atenção: *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Veja também:
TABELA - Escrever Tabela Inteira
TABELA - Escrever Tabela LongInt
TABELA - Escrever Tabela Word
TABELA - Escrever Tabela Real

TABELA - Ler Tabela Inteira

Este bloco permite ler posições de uma tabela de números inteiros (-32768 a 32767).

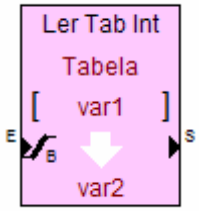
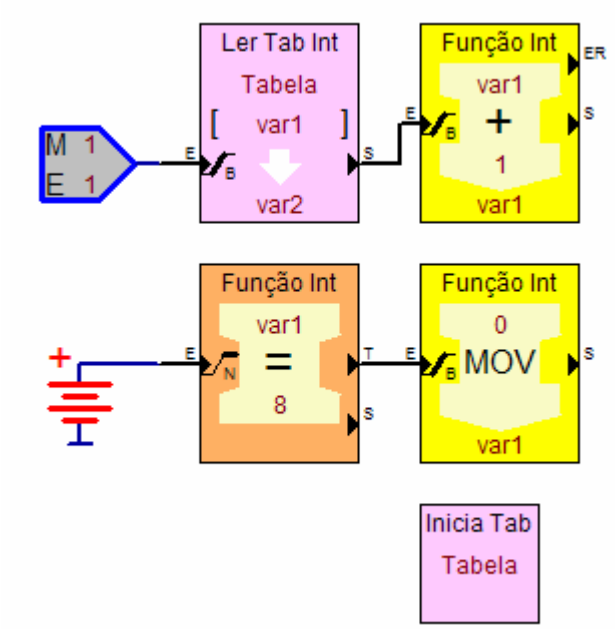


Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser lida na memória de programa (flash) ou na memória volátil (RAM). Índice indica a posição a ser lida da tabela (de zero à posição final da tabela menos um). Já Destino recebe o valor lido na posição de tabela especificado.

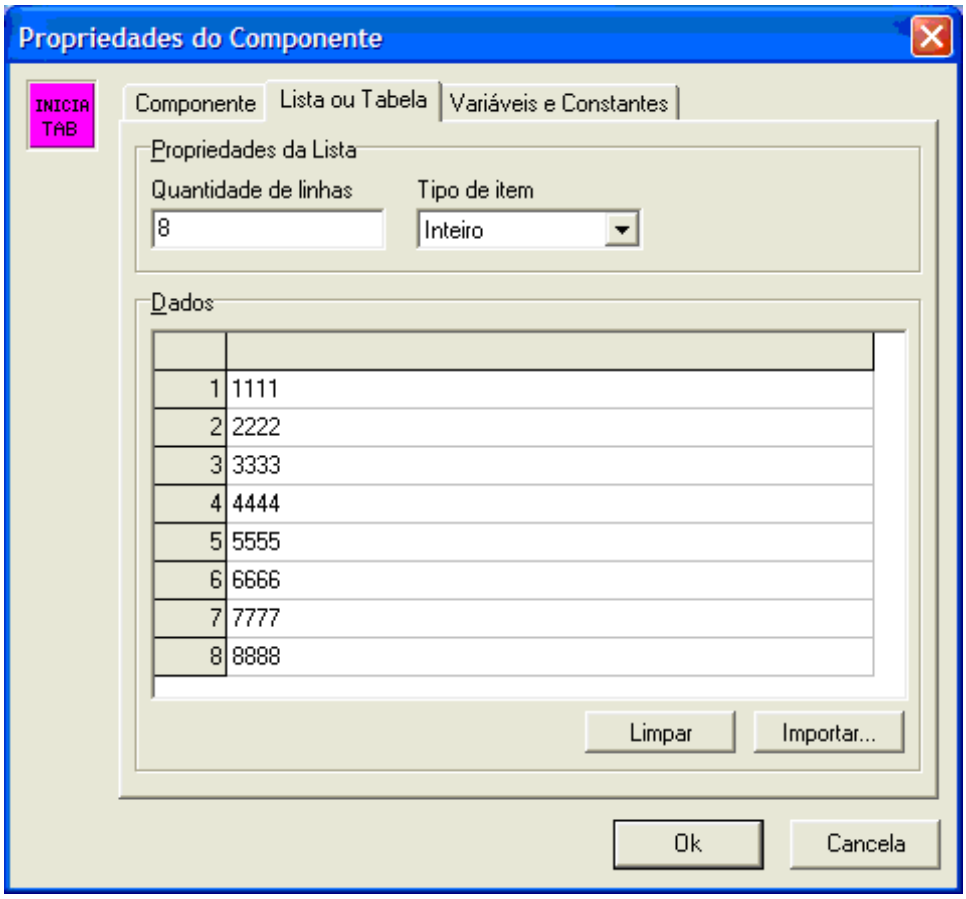
Note que Tabela e Índice podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Destino deve ser necessariamente uma variável inteira ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Tabela - Ler Tabela Inteira.dxg**

O programa de exemplo acima lê as 8 posições da tabela inteira e as coloca em var2. Cada vez que a entrada E1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) é energizada é lida a posição da tabela apontada por var1 e o resultado é colocado em var2. A seguir é incrementado o índice (var1). Note que este índice inicia no valor 0 e é incrementado até 7. Ao atingir valor 8 ele é novamente inicializado com valor 0. Os valores inicializados na tabela via bloco de Inicia Tab são:



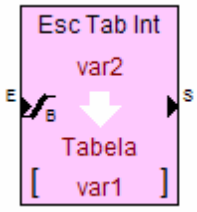
Importante: *A primeira posição de tabelas no µDX200 é acessada com variável Índice no valor zero, ou seja, o Índice indica a posição da tabela a ser lida a partir da primeira posição. Portanto, o Índice deverá assumir valores entre zero e tamanho da tabela menos um. No caso de tabelas de 8 posições, como no exemplo, Índice deve ser restrito a valores entre 0 e 7.*

Atenção: *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Veja também:
TABELA - Ler Tabela Byte
TABELA - Ler Tabela LongInt
TABELA - Ler Tabela Word
TABELA - Ler Tabela Real

TABELA - Escrever Tabela Inteira

Este bloco permite escrever em posições de uma tabela de números inteiros (-32768 a 32767).

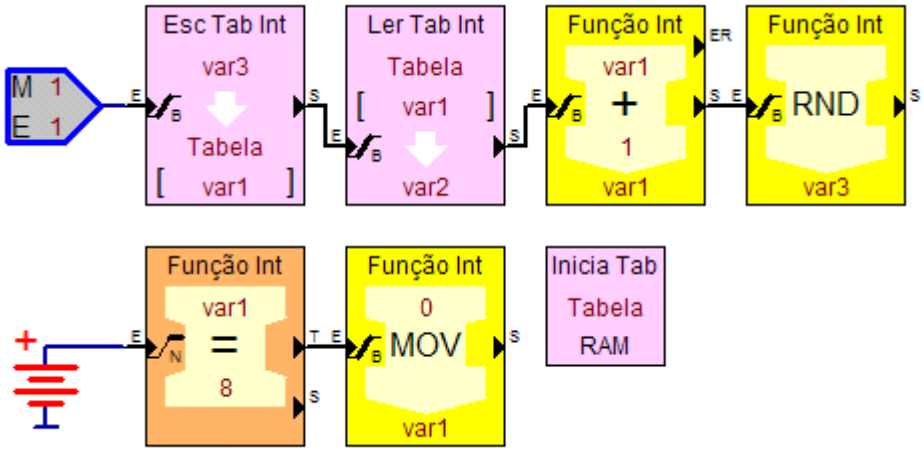


Naturalmente esta tabela deve estar em memória volátil (RAM), de forma a permitir escrita.

Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser escrita na memória volátil (RAM). Índice indica a posição a ser escrita na tabela (de zero à posição final da tabela menos um). Já Origem é o valor a ser escrito na posição de tabela especificado.

Note que Origem e Índice podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Tabela deve ser necessariamente um valor word que aponta para uma tabela em RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

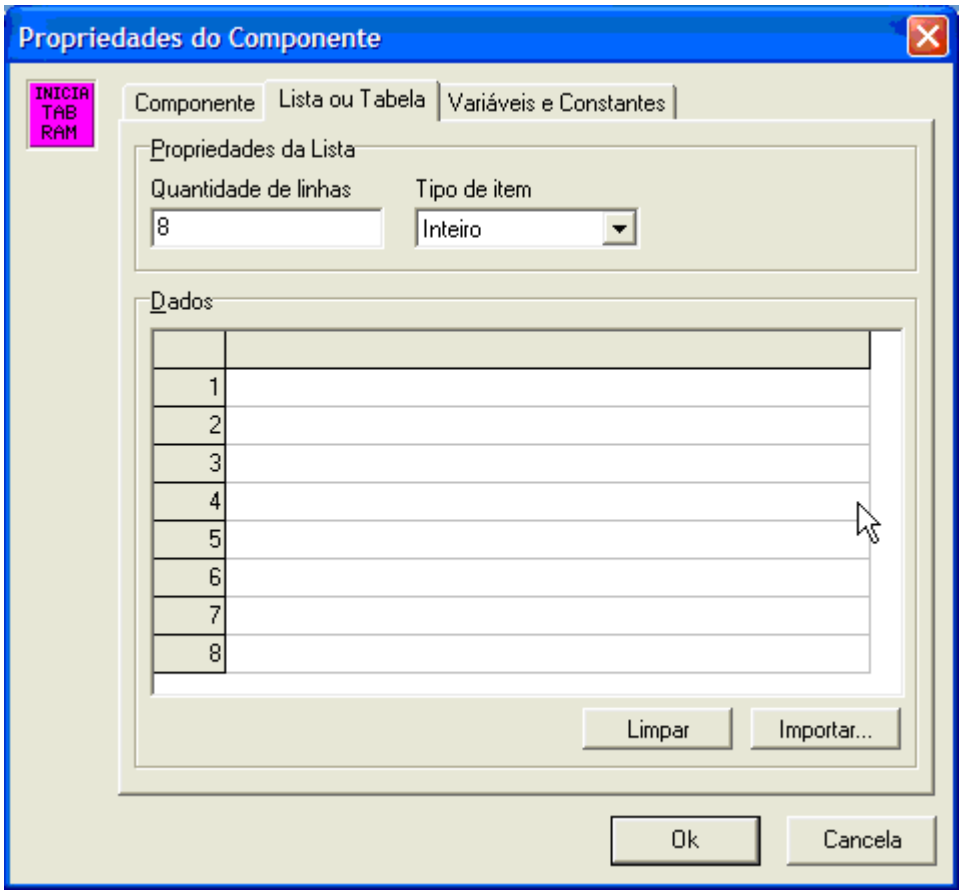


Exemplo de Programa Aplicativo: **Tabela - Escrever Tabela Inteira.dxg**

O programa de exemplo escreve a variável var3 em posições sucessivas da tabela (apontada por var1). A seguir, esta mesma posição é lida e colocada em var2. Por fim, o índice var1 é incrementado e em var3 é colocado um número randômico inteiro. Note que o índice var1 é inicializado com valor zero (para apontar o primeiro elemento da tabela), e é incrementado até 7. Quando assume valor 8 é reinicializado para valor zero (já que a tabela possui apenas 8 posições).

Como este programa escreve na tabela é necessário que se trate de uma tabela em RAM, pois tabelas em memória de programa (flash) não permitem gravação de novos valores. No caso de tabelas em RAM o µDX200 não admite inicialização de valores via bloco de Inicia Tab RAM, como no caso de tabelas em memória de programa (flash). Neste caso todas as posições da tabela são

inicializadas com valor zero (como as variáveis do µDX200), independentemente dos valores especificados para cada posição da tabela no bloco Inicia Tab RAM. A edição do bloco INICIA Tab RAM é a seguinte:



Importante: *A primeira posição de tabelas no µDX200 é acessada com variável Índice no valor zero, ou seja, o Índice indica a posição da tabela a ser lida a partir da primeira posição. Portanto, o Índice deverá assumir valores entre zero e tamanho da tabela menos um. No caso de tabelas de 8 posições, como no exemplo, Índice deve ser restrito a valores entre 0 e 7.*

Atenção: *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Veja também:
TABELA - Escrever Tabela Byte
TABELA - Escrever Tabela LongInt
TABELA - Escrever Tabela Word
TABELA - Escrever Tabela Real

TABELA - Ler Tabela LongInt

Este bloco permite ler posições de uma tabela de números longint (-2.147.483.648 a 2.147.483.647).

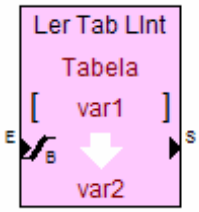
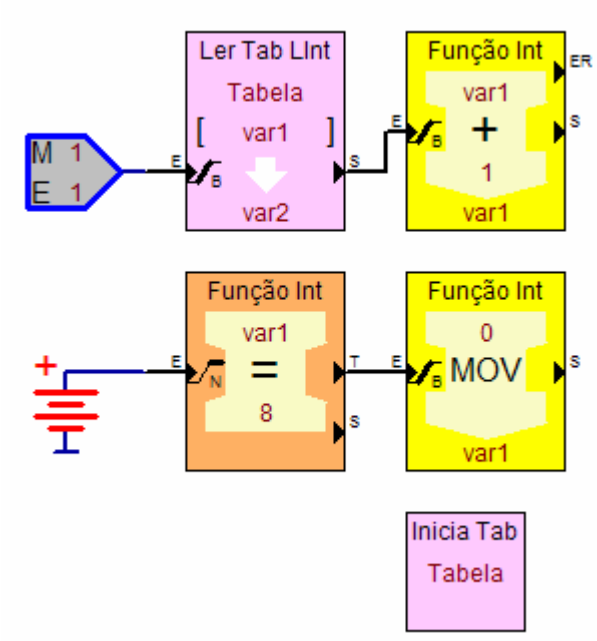


Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser lida na memória de programa (flash) ou na memória volátil (RAM). Índice indica a posição a ser lida da tabela (de zero à posição final da tabela menos um). Já Destino recebe o valor lido na posição de tabela especificado.

Note que Tabela e Índice podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Destino deve ser necessariamente uma variável longint ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

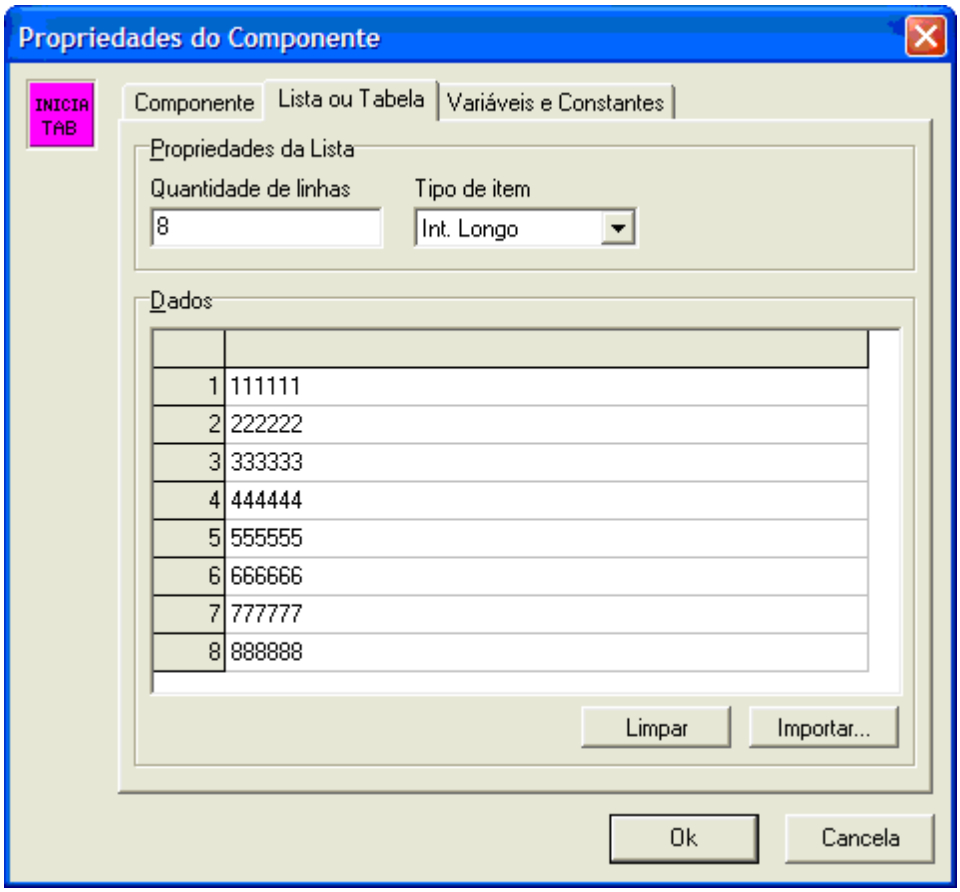
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Tabela - Ler Tabela LongInt.dxg**

O programa de exemplo acima lê as 8 posições da tabela longint e as coloca em var2. Cada vez que a entrada E1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) é energizada é lida a posição da tabela apontada por var1 e o resultado é colocado em var2. A seguir é incrementado o índice (var1). Note que este índice inicia no valor 0 e é incrementado até 7. Ao atingir valor 8 ele é

novamente inicializado com valor zero. Os valores inicializados na tabela via bloco de Inicia Tab são:



Importante: *A primeira posição de tabelas no µDX200 é acessada com variável Índice no valor zero, ou seja, o Índice indica a posição da tabela a ser lida a partir da primeira posição. Portanto, o Índice deverá assumir valores entre zero e tamanho da tabela menos um. No caso de tabelas de 8 posições, como no exemplo, Índice deve ser restrito a valores entre 0 e 7.*

Atenção: *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Veja também:
TABELA - Ler Tabela Byte
TABELA - Ler Tabela Inteira
TABELA - Ler Tabela Word
TABELA - Ler Tabela Real

TABELA - Escrever Tabela LongInt

Este bloco permite escrever em posições de uma tabela de números longint (-2.147.483.648 a 2.147.483.647).

Naturalmente esta tabela deve estar em memória volátil (RAM), de forma a permitir escrita.

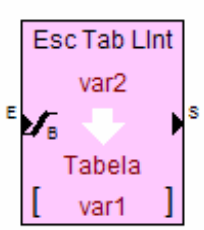
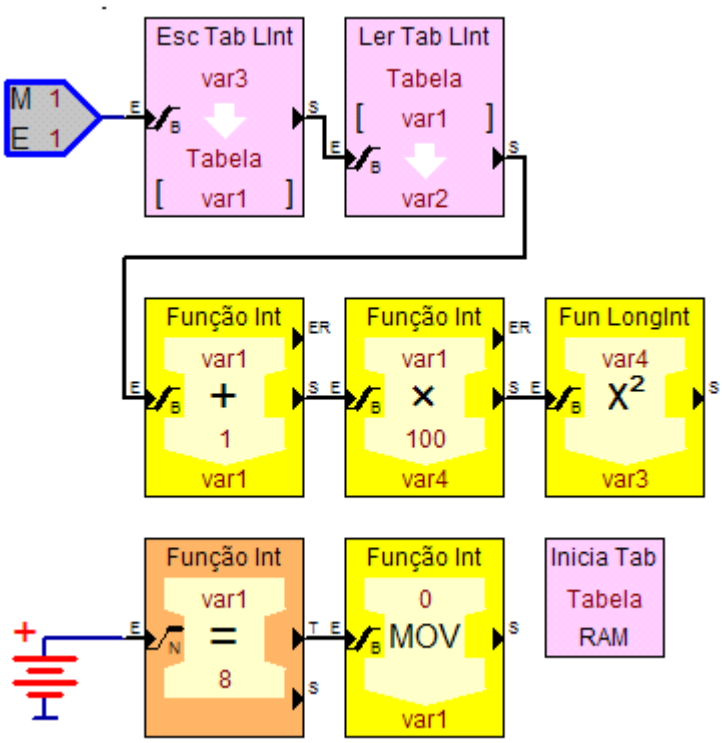


Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser escrita na memória volátil (RAM). Índice indica a posição a ser escrita na tabela (de zero à posição final da tabela menos um). Já Origem é o valor a ser escrito na posição de tabela especificado.

Note que Origem e Índice podem ser variáveis longint e inteira, respectivamente, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Tabela deve ser necessariamente um valor word que aponta para uma tabela em RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



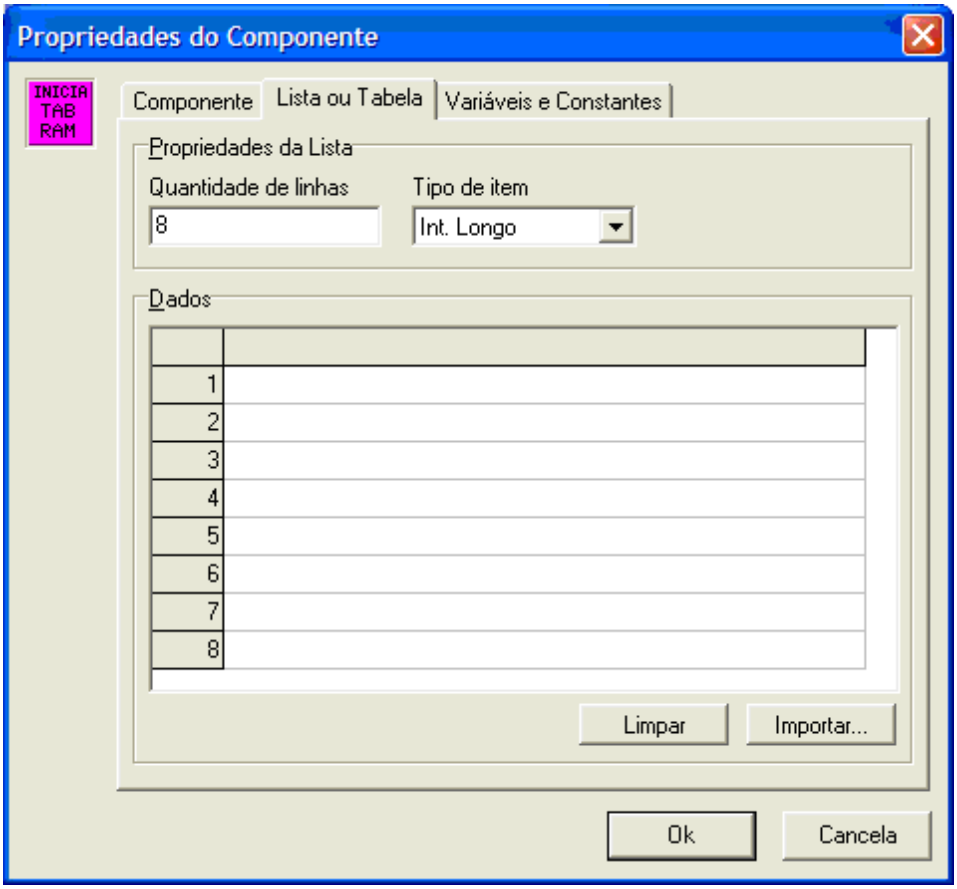
Exemplo de Programa Aplicativo: **Tabela - Escrever Tabela LongInt.dxg**

O programa de exemplo escreve a variável var3 em posições sucessivas da tabela (apontada por var1). A seguir, esta mesma posição é lida e colocada em var2. Por fim, o índice var1 é incrementado e em var3 é colocado o valor do índice multiplicado por 100 e elevado ao quadrado. Note que o índice var1 é inicializado com valor zero (para apontar o primeiro elemento da tabela), e é incrementado até 7. Quando assume valor 8 é reinicializado para valor zero (já que a tabela possui apenas 8 posições).

As posições da tabela longint é preenchida com os seguintes valores:

var1 = 0 → var3 = ((var1+1) x 100)^2 → var3 = 10000
var1 = 1 → var3 = ((var1+1) x 100)^2 → var3 = 40000
var1 = 2 → var3 = ((var1+1) x 100)^2 → var3 = 90000
var1 = 3 → var3 = ((var1+1) x 100)^2 → var3 = 160000
var1 = 4 → var3 = ((var1+1) x 100)^2 → var3 = 250000
var1 = 5 → var3 = ((var1+1) x 100)^2 → var3 = 360000
var1 = 6 → var3 = ((var1+1) x 100)^2 → var3 = 490000
var1 = 7 → var3 = ((var1+1) x 100)^2 → var3 = 640000

Como este programa escreve na tabela é necessário que se trate de uma tabela em RAM, pois tabelas em memória de programa (flash) não permitem gravação de novos valores. No caso de tabelas em RAM o µDX200 não admite inicialização de valores via bloco de Inicia Tab RAM, como no caso de tabelas em memória de programa (flash). Neste caso todas as posições da tabela são inicializadas com valor zero (como as variáveis do µDX200), independentemente dos valores especificados para cada posição da tabela no bloco Inicia Tab RAM. A edição do bloco INICIA Tab RAM é a seguinte:



Importante: *A primeira posição de tabelas no µDX200 é acessada com variável Índice no valor zero, ou seja, o Índice indica a posição da tabela a ser lida a partir da primeira posição. Portanto, o Índice deverá assumir valores entre zero e tamanho da tabela menos um. No caso de tabelas de 8 posições, como no exemplo, Índice deve ser restrito a valores entre 0 e 7.*

Atenção: *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Veja também:
TABELA - Escrever Tabela Byte
TABELA - Escrever Tabela Inteira
TABELA - Escrever Tabela Word
TABELA - Escrever Tabela Real

TABELA - Ler Tabela Word

Este bloco permite ler posições de uma tabela de números word (0 a 65535).

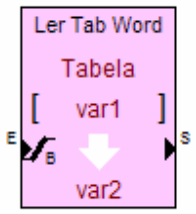
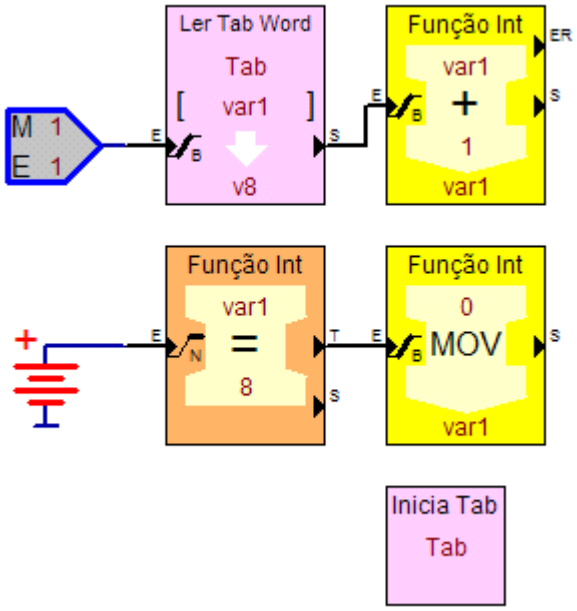


Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser lida na memória de programa (flash) ou na memória volátil (RAM). Índice indica a posição a ser lida da tabela (de zero à posição final da tabela menos um). Já Destino recebe o valor lido na posição de tabela especificado.

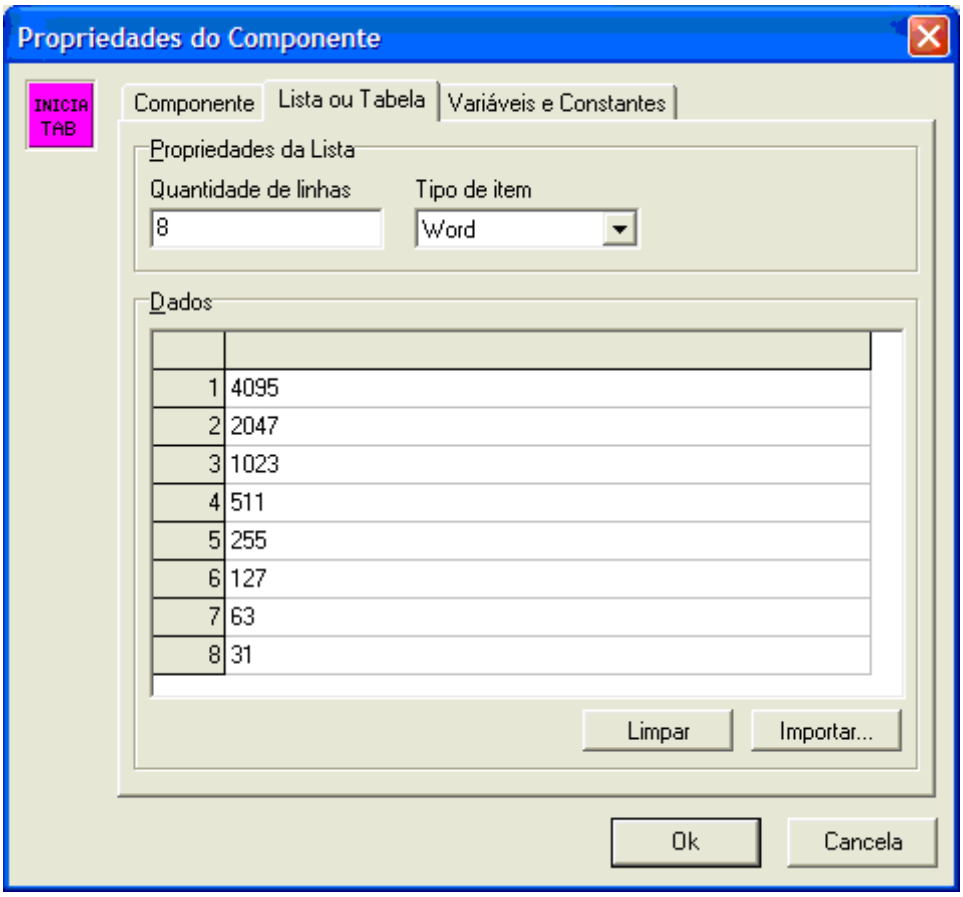
Note que Tabela e Índice podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Destino deve ser necessariamente uma variável word ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Tabela - Ler Tabela Word.dxg**

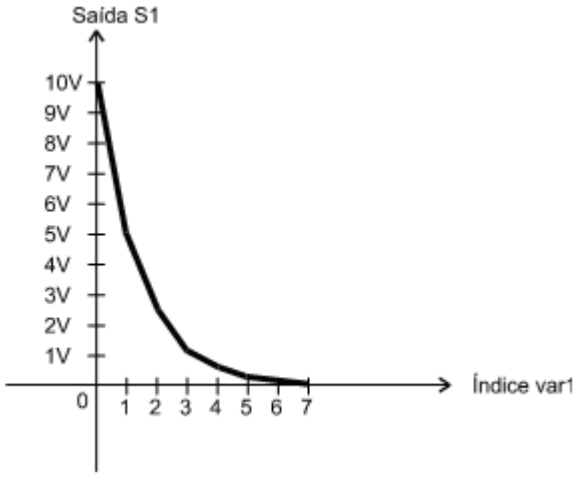
O programa de exemplo acima lê as 8 posições da tabela word e as coloca em v8. Mas note que v8 é a variável associada a saída analógica S1 do controlador µDX200. Cada vez que a entrada E1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) é energizada é lida a posição da tabela apontada por var1 e o resultado é colocado em v8. A seguir é incrementado o índice (var1). Note que este índice inicia no valor zero e é incrementado até 7. Ao atingir valor 8 ele é novamente inicializado com valor zero. Os valores inicializados na tabela via bloco de Inicia Tab são:



Estes valores correspondem ao fundo de escala da saída analógica (4095) dividido por 2 elevado ao (índice-1). Isso irá gerar a seguinte seqüência de valores analógicos na saída analógica E1 (pressupondo saída analógica programada para escala 0-10V):

Índice = 0 → Saída S1 = 10,00V
Índice = 1 → Saída S1 = 5,00V
Índice = 2 → Saída S1 = 2,50V
Índice = 3 → Saída S1 = 1,25V
Índice = 4 → Saída S1 = 0,63V
Índice = 5 → Saída S1 = 0,31V
Índice = 6 → Saída S1 = 0,16V
Índice = 7 → Saída S1 = 0,08V

Note que tabelas podem ser usadas, por exemplo, para linearizar a entrada de determinado sensor de díficil descrição matemática.



Importante: *A primeira posição de tabelas no µDX200 é acessada com variável Índice no valor zero, ou seja, o Índice indica a posição da tabela a ser lida a partir da primeira posição. Portanto, o Índice deverá assumir valores entre zero e tamanho da tabela menos um. No caso de tabelas de 8 posições, como no exemplo, Índice deve ser restrito a valores entre 0 e 7.*

Atenção: *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Veja também:
TABELA - Ler Tabela Byte
TABELA - Ler Tabela Inteira
TABELA - Ler Tabela LongInt
TABELA - Ler Tabela Real

TABELA - Escrever Tabela Word

Este bloco permite escrever em posições de uma tabela de números word (0 a 65535).

Naturalmente esta tabela deve estar em memória volátil (RAM), de forma a permitir escrita.

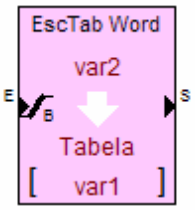
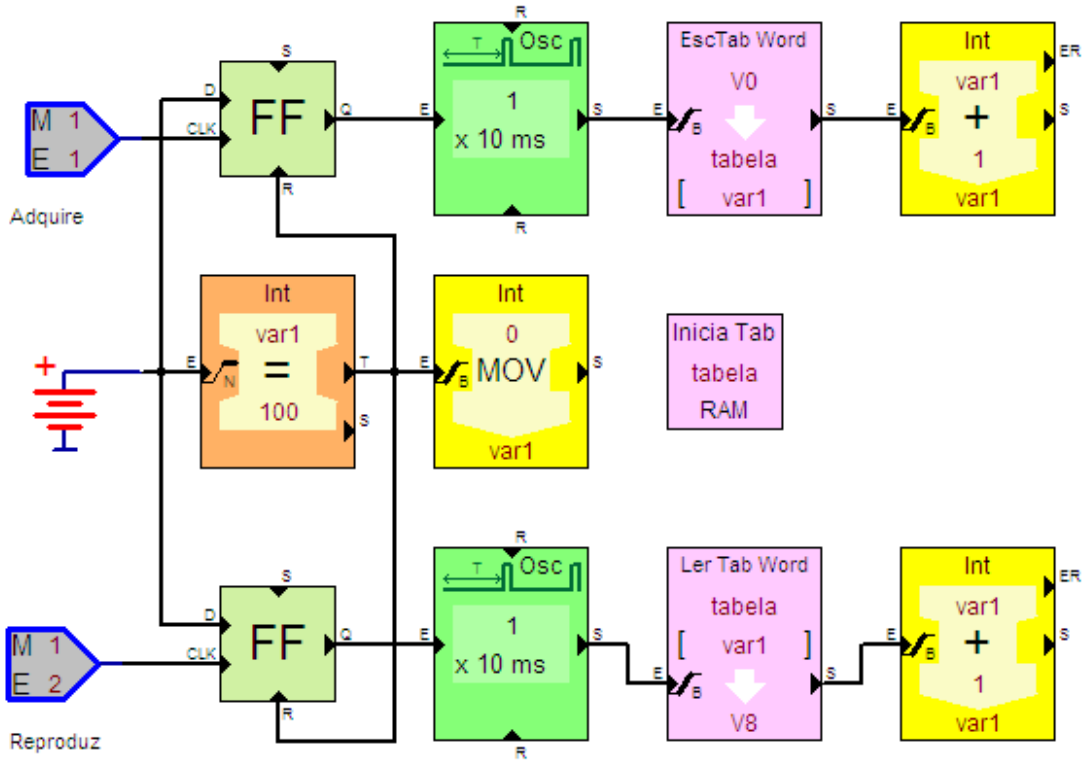


Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser escrita na memória volátil (RAM). Índice indica a posição a ser escrita na tabela (de zero à posição final da tabela menos um). Já Origem é o valor a ser escrito na posição de tabela especificado.

Note que Origem e Índice podem ser variáveis word e inteiras, respectivamente, constantes

(valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Tabela deve ser necessariamente um valor word que aponta para uma tabela em RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

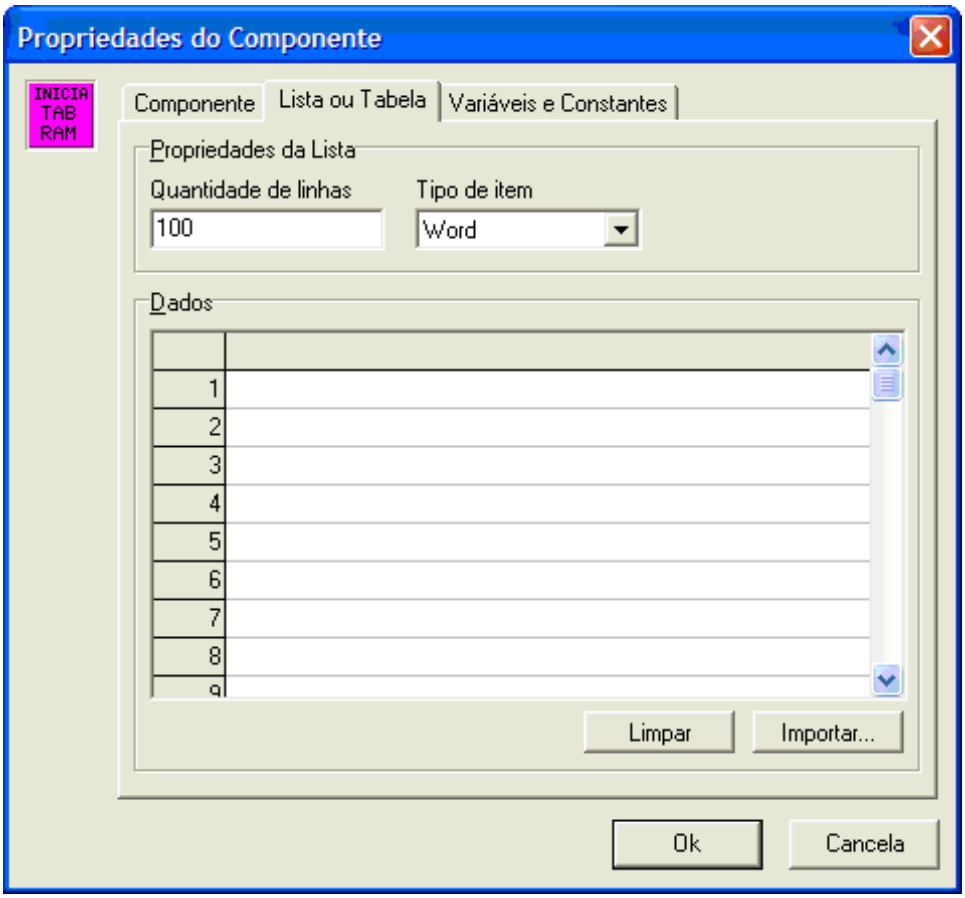


Exemplo de Programa Aplicativo: **Tabela - Escrever Tabela Word.dxg**

O interessante programa de exemplo acima amostra o sinal analógico presente na entrada analógica E1 do µDX200 durante 1 segundo, ao energizar a entrada digital E1 do módulo M1 de Entradas/Saídas (µDX210). A taxa de amostragem é de 10ms, perfazendo 100 amostras durante o 1 segundo de amostragem. Ao pressionar a entrada E2 do mesmo módulo M1 de Entradas/Saídas (µDX210) o sinal capturado é reproduzido na saída analógica S1 do µDX200. Se tanto a entrada analógica E1 quanto a saída analógica S1 do µDX200 estiverem programadas para a mesma escala (ambas como 0-10V ou ambas para 0-20mA) o sinal será reproduzido idêntico.

Como este programa escreve na tabela é necessário que se trate de uma tabela em RAM, pois tabelas em memória de programa (flash) não permitem gravação de novos valores. No caso de tabelas em RAM o µDX200 não admite inicialização de valores via bloco de Inicia Tab RAM, como no caso de tabelas em memória de programa (flash).

Neste caso todas as posições da tabela são inicializadas com valor zero (como as variáveis do µDX200), independentemente dos valores especificados para cada posição da tabela no bloco Inicia Tab RAM. A edição do bloco INICIA Tab RAM é a seguinte:



Importante: *A primeira posição de tabelas no µDX200 é acessada com variável Índice no valor zero, ou seja, o Índice indica a posição da tabela a ser lida a partir da primeira posição. Portanto, o Índice deverá assumir valores entre zero e tamanho da tabela menos um. No caso de tabelas de 8 posições, como no exemplo, Índice deve ser restrito a valores entre 0 e 7.*

Atenção: *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Veja também:
TABELA - Escrever Tabela Byte
TABELA - Escrever Tabela Inteira
TABELA - Escrever Tabela LongInt
TABELA - Escrever Tabela Real

TABELA - Ler Tabela Real

Este bloco permite ler posições de uma tabela de números reais (-3,4E-38 a 3,4E38).

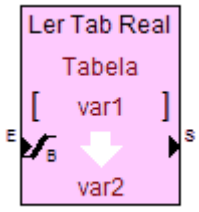
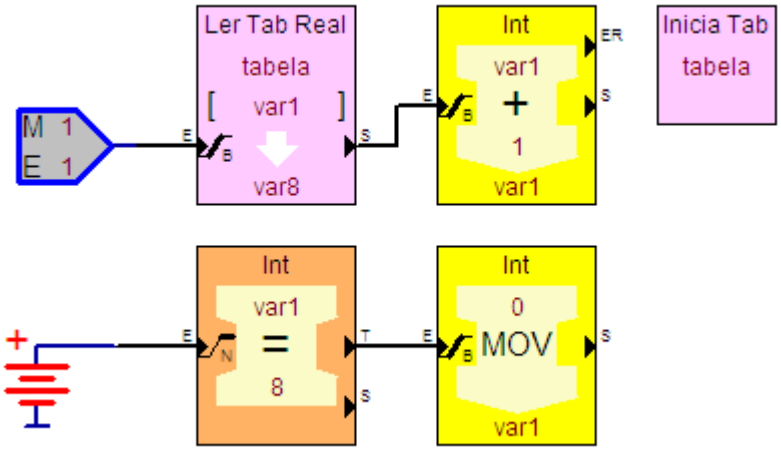


Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser lida na memória de programa (flash) ou na memória volátil (RAM). Índice indica a posição a ser lida da tabela (de zero à posição final da tabela menos um). Já Destino recebe o valor lido na posição de tabela especificado.

Note que Tabela e Índice podem ser variáveis word ou inteira, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Destino deve ser necessariamente uma variável real ou uma referência a uma posição de memória RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **Tabela - Ler Tabela Real.dxg**

Veja também:
TABELA - Ler Tabela Byte
TABELA - Ler Tabela Inteira
TABELA - Ler Tabela LongInt
TABELA - Ler Tabela Word

TABELA - Escrever Tabela Real

Este bloco permite escrever em posições de uma tabela de números reais (-3,4E-38 a 3,4E38).

Naturalmente esta tabela deve estar em memória volátil (RAM), de forma a permitir escrita.

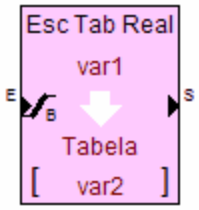
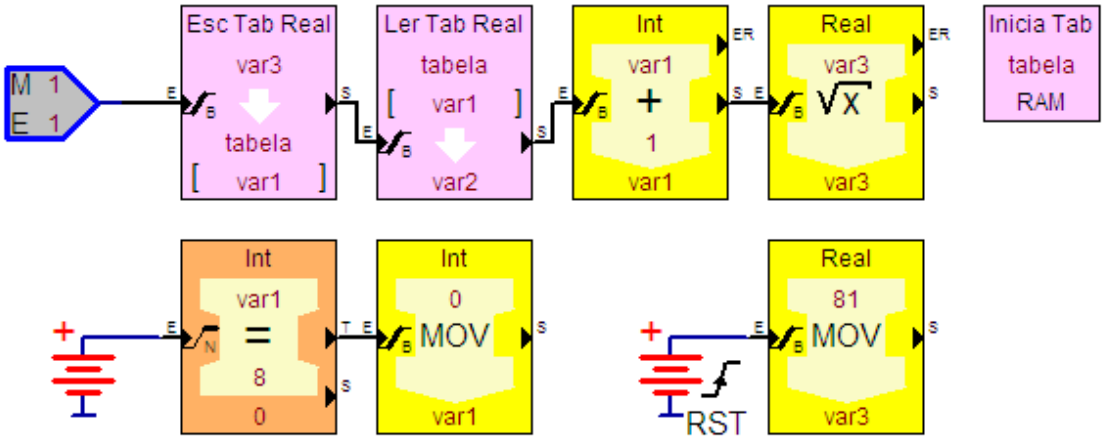


Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser escrita na memória volátil (RAM). Índice indica a posição a ser escrita na tabela (de zero à posição final da tabela menos um). Já Origem é o valor a ser escrito na posição de tabela especificado.

Note que Origem e Índice podem ser variáveis real e inteira, respectivamente, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). Já Tabela deve ser necessariamente um valor word que aponta para uma tabela em RAM, uma vez que irá receber o resultado da operação.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



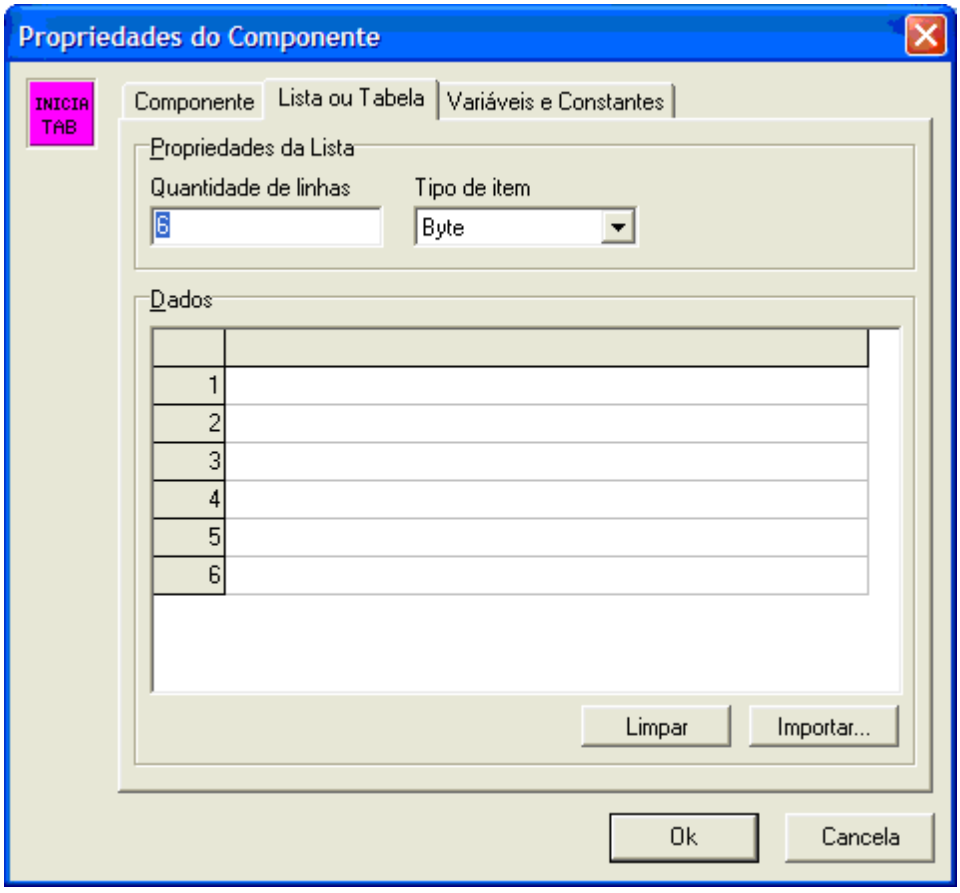
Exemplo de Programa Aplicativo: **Tabela - Escrever Tabela Real.dxg**

Veja também:
TABELA - Escrever Tabela Byte
TABELA - Escrever Tabela Inteira
TABELA - Escrever Tabela LongInt
TABELA - Escrever Tabela Word

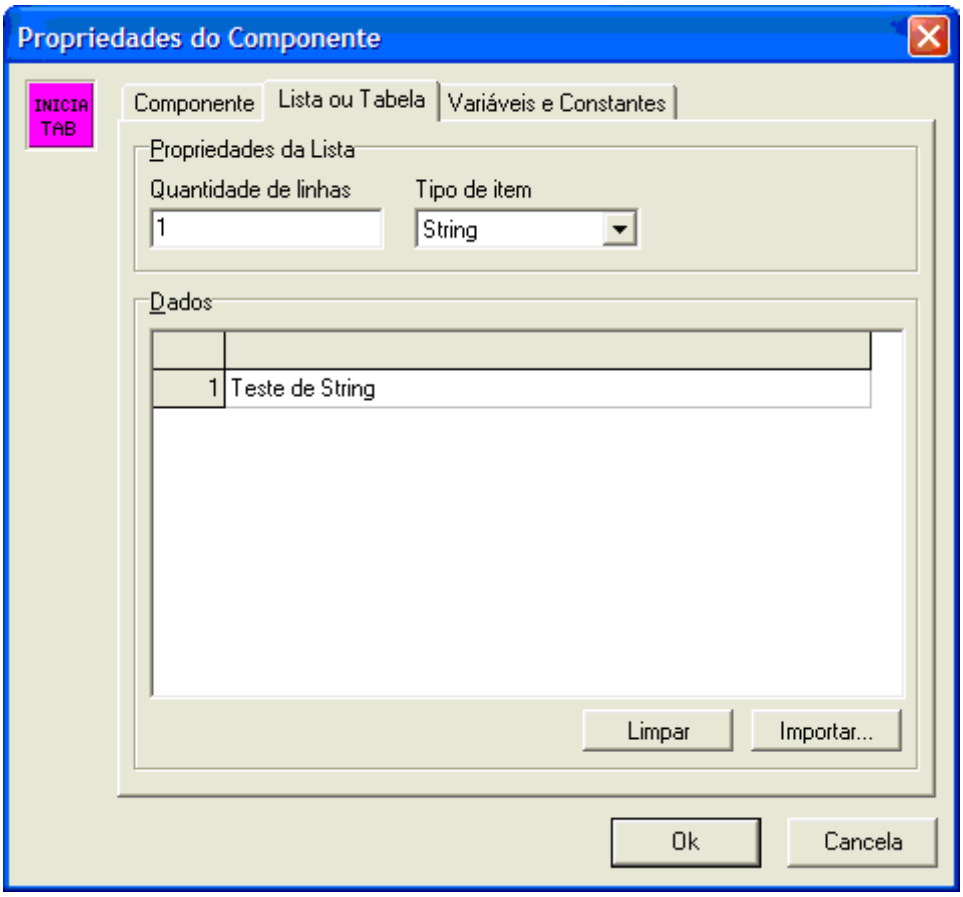
TABELA - Inicializa Tabela

Este bloco permite declarar uma tabela na memória não-volátil (Flash). Este tipo de tabela permite apenas leitura, e os valores de cada posição da tabela é especificado neste bloco. Ou seja, as posições da tabela são constantes definidas na própria declaração da tabela.

Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser lida na memória de programa (flash). Quantidade de linhas é uma constante que indica a quantidade de posições da tabela (de 1 à capacidade máxima de memória livre de programa, que depende do tamanho do programa aplicativo). Já Tipo de Item indica qual o tipo de dado será usado na tabela. Pode-se escolher entre Byte (8 bits), Word (16 bits), Inteiro (16 bits), String (número variável de caracteres), ou ainda Int. Longo (LongInt - 32 bits). Note que Tabela pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).



No caso específico de tabela String, esta deve ter apenas uma linha. Ou seja, este bloco é usado também para declarar um string. No exemplo abaixo declara-se um string com conteúdo "Teste de String":



Este bloco não possui nodos de entrada ou saída, e a declaração de tabela está sempre ativa no programa aplicativo.

Existem duas teclas na janela de edição do bloco de Inicializa Tabela. Uma delas permite limpar todas as posições da tabela. Outra tecla permite importar os valores de uma tabela em arquivo para as posições da tabela definida no bloco. Note que são aceitos arquivos com terminação .CSV (arquivo de valores separados por ";" do Microsoft Excel) ou com terminação .TXT (arquivos texto). Este recurso é muito útil para importar tabelas geradas por um programa de cálculo matemático, por exemplo. Ou tabelas de linearização de um sensor.

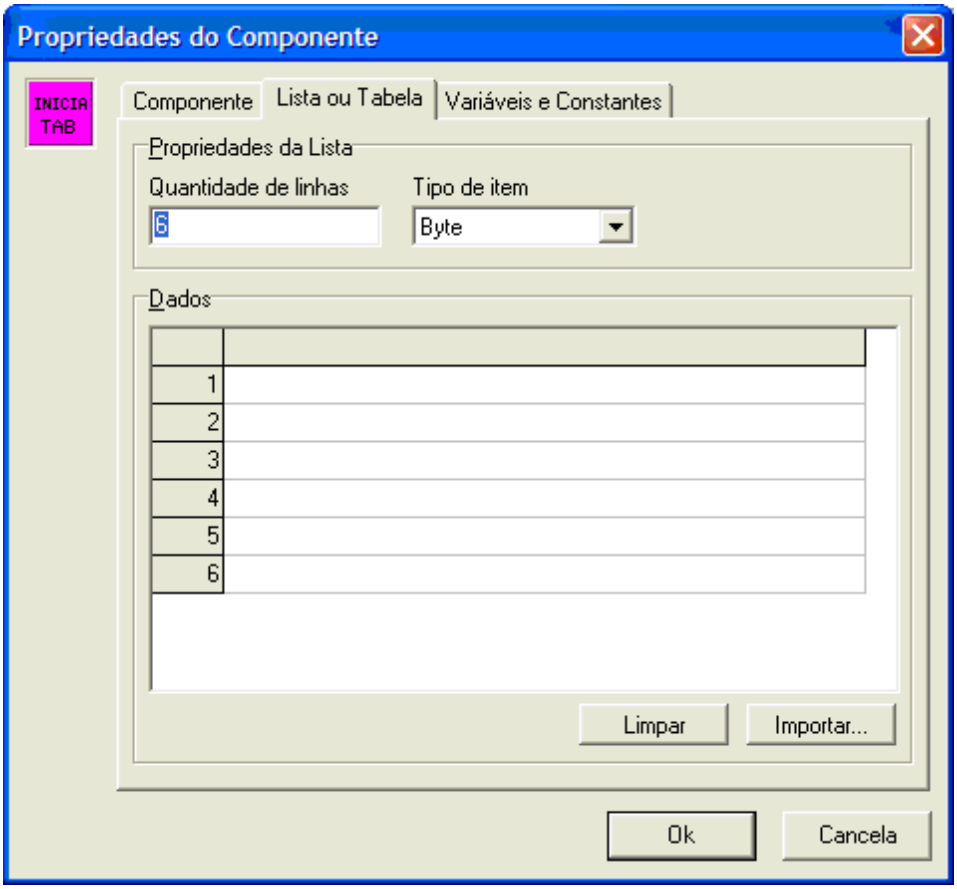
> ***Atenção:*** *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Veja também:
TABELA - Inicializa Tabela RAM

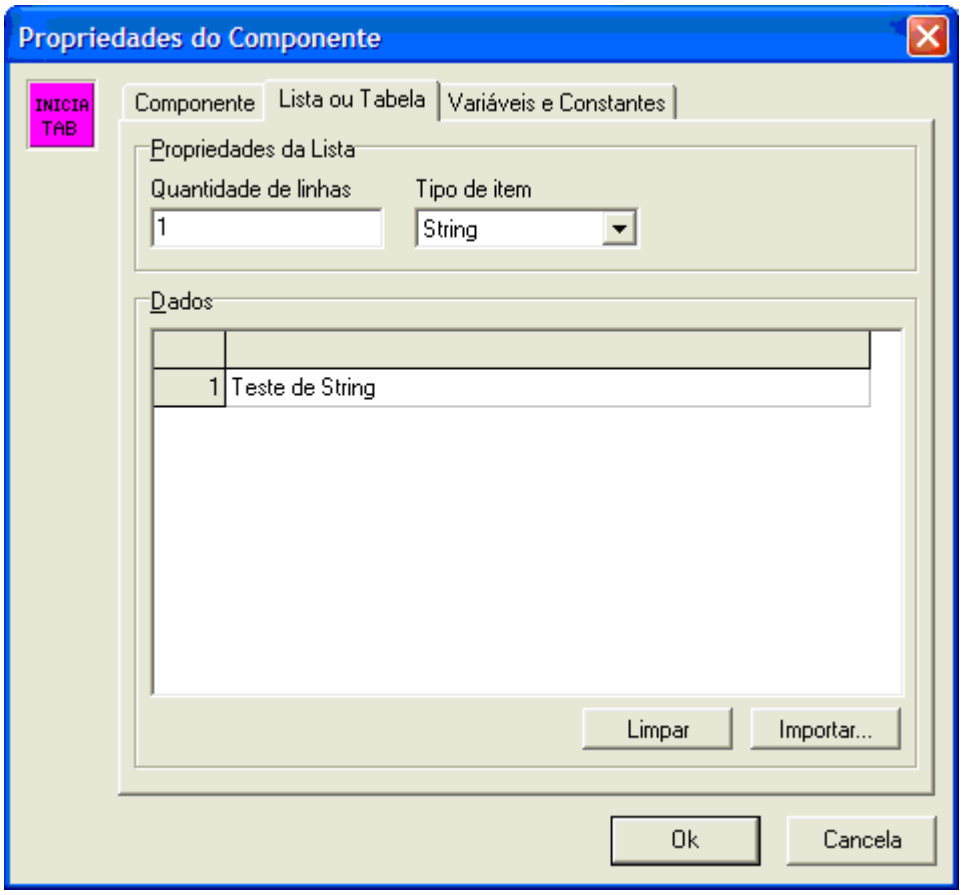
TABELA - Inicializa Tabela RAM

Este bloco permite declarar uma tabela na memória volátil (RAM). Este tipo de tabela permite leitura e escrita, e os valores de cada posição da tabela são sempre inicializados com valor zero (exceto no caso de tabela string). Ou seja, as posições da tabela são variáveis inicializadas com valor nulo na declaração da tabela (assim como as demais variáveis do µDX200).

Tabela é a variável tipo word (0 a 65535) que aponta para a posição da tabela a ser lida na memória volátil (RAM). Quantidade de linhas é uma constante que indica a quantidade de posições da tabela (de 1 à capacidade máxima de memória RAM livre, que depende da quantidade de nodos e variáveis usados no programa aplicativo). Já Tipo de Item indica qual o tipo de dado será usado na tabela. Pode-se escolher entre Byte (8 bits), Word (16 bits), Inteiro (16 bits), String (número variável de caracteres), ou ainda Int. Longo (LongInt - 32 bits). Note que Tabela pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).



No caso específico de tabela String, esta deve ter apenas uma linha. Ou seja, este bloco é usado também para declarar um string. No exemplo abaixo declara-se um string em RAM com conteúdo "Teste de String":

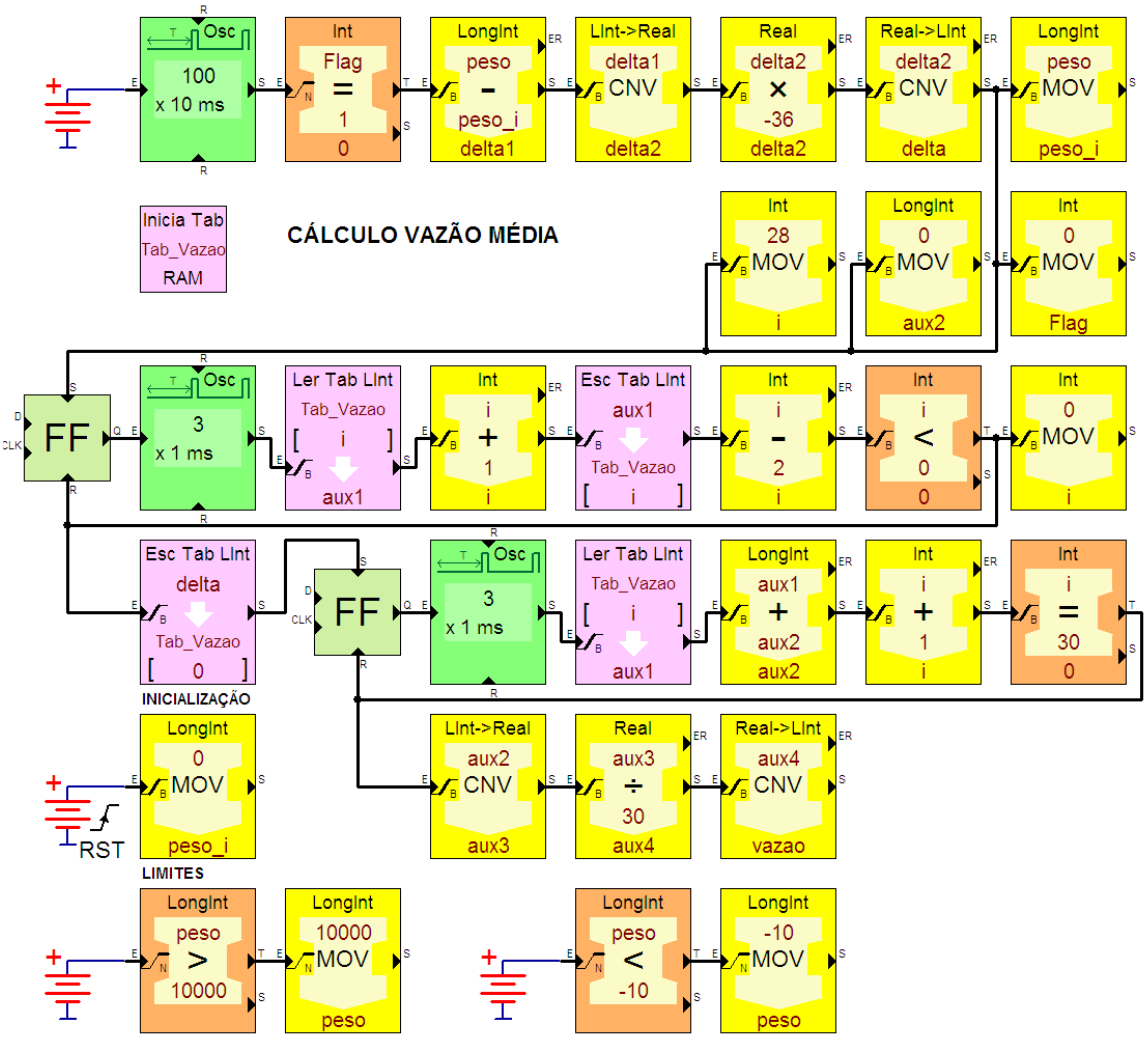


Este bloco não possui nodos de entrada ou saída, e a declaração de tabela está sempre ativa no programa aplicativo.

Existem duas teclas na janela de edição do bloco de Inicializa Tabela RAM. Uma delas permite limpar todas as posições da tabela. Outra tecla permite importar os valores de uma tabela em arquivo para as posições da tabela definida no bloco. Note que são aceitos arquivos com terminação .CSV (arquivo de valores separados por ";" do Microsoft Excel) ou com terminação .TXT (arquivos texto). Este recurso, no caso de tabelas em RAM, é inútil, já que os valores das posições de tabela em RAM são sempre inicializados com zero.

> ***Atenção:*** *Nas versões mais recentes do PG estão disponíveis as funções* ***Ctrl+Ins*** *e* ***Ctrl+Del*** *para incluir ou excluir itens da tabela. Também é possível exportar a tabela para um arquivo.*

Uma aplicação interessante da Tabela RAM é para armazenar os últimos dados lidos de um processo, por exemplo, para obtermos o valor médio. Se forem poucos dados é possível simplesmente usar uma variável para cada dado obtido, mas se o número de dados for grande este método se mostra ineficaz. Por exemplo, digamos que é medido o peso em um silo, e pela diferença entre o peso atual e o anterior se calcule a vazão de descarga do silo. Se o peso for dado em Kg e a vazão em ton/h, resulta que uma diferença de n Kg em um segundo implica em vazão de 3,6n ton/h (já que cada hora possui 3600 segundos). O programa Tabela - Vazão Média.dxg executa a armazenagem das últimas 30 vazões calculadas, e retorna na variável vazao a média aritmética das mesmas.

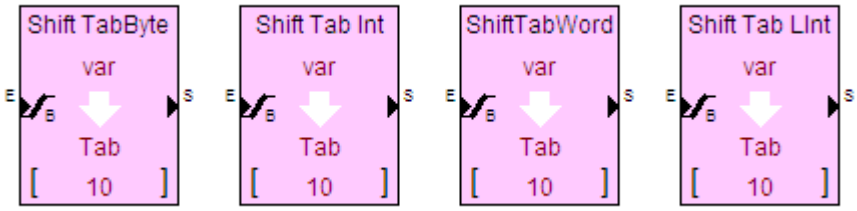


Exemplo de Programa Aplicativo: **Tabela - Vazão Média.dxg**

Veja também:
TABELA - Inicializa Tabela

TABELA - Shift Tabela

Os blocos Shift Tabela permitem abrir espaço em uma tabela RAM e incluir um dado nessa tabela. O último dado da tabela é perdido. O penúltimo dado da tabela é transferido para a última posição da tabela, e assim por diante, até o dado na posição 1 que é transferido para a posição 2. Já a posição 1 é preenchido pela variável especificada pelo bloco. Existe um bloco para cada tipo de tabela RAM: byte, inteira, word ou longint.



A variável **Tabela** deve apontar para uma tabela compatível com o bloco de shift usado (byte, inteira, word ou longint). Já a variável inteira **Índice** indica quantos elementos dessa tabela serão deslocados. Por fim, a variável **Origem** irá ser transferida para a posição 1 da tabela, que ficou vaga pelo deslocamento (shift).

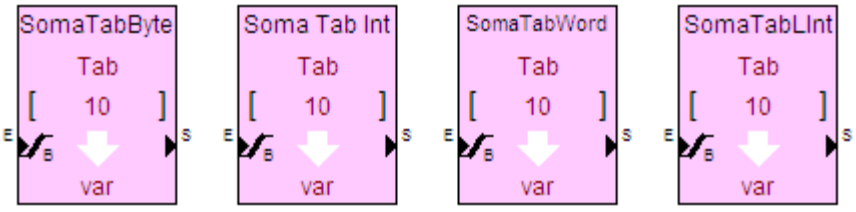
O nodo de **Entrada** (**E**) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de **Saída** (**S**) é ativado ao completar a operação. Se o nodo de **Entrada** (**E**) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX201 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de **Entrada** (**E**) ser ativado por borda o bloco só é executado uma vez na borda de subida de **E**, aguardando este nodo desligar para permitir novo acionamento.

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.62 ou superior.

Veja também:
TABELA - Somatório Tabela

TABELA - Somatório Tabela

Estes blocos permitem calcular o somatório de uma tabela. Isso é útil para gerar a média aritmética de valores adquiridos, por exemplo. Existe um bloco para cada tipo de tabela: byte, inteira, word ou longint. Note que o resultado do somatório sempre está disponível em uma variável longint. No caso de tabelas inteiras e longint existem valores negativos e isso é levado em consideração na soma dos elementos da tabela. Já em tabelas byte e word os elementos são sempre positivos.

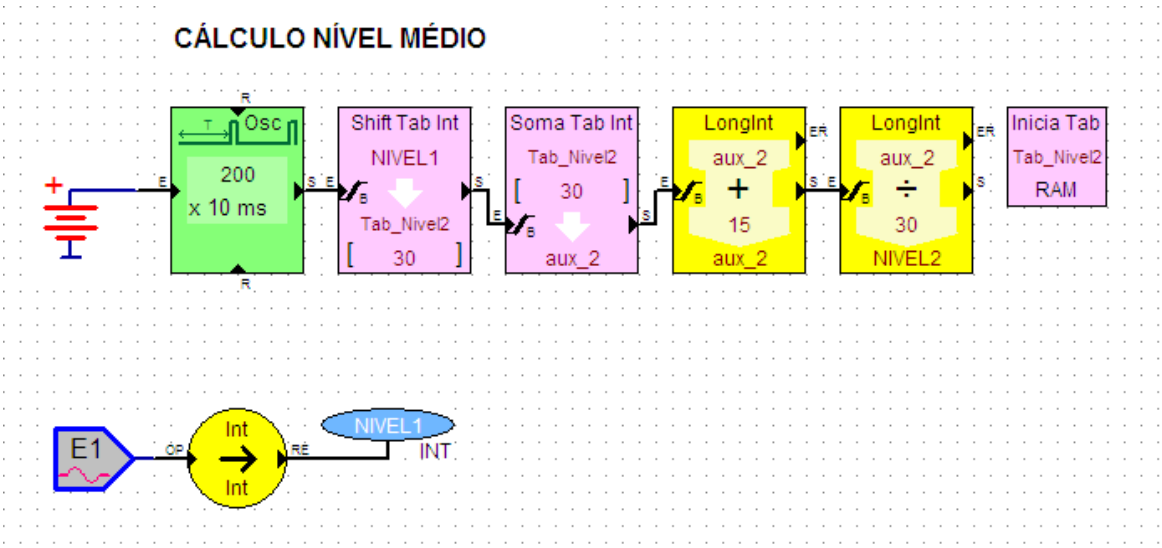


A variável **Tabela** deve apontar para uma tabela compatível com o bloco de somatório usado (byte, inteira, word ou longint). Já a variável inteira **Índice** indica quantos elementos dessa tabela serão usados na soma. O mínimo é um elemento. Por fim, a variável longint **Destino** irá receber o valor do somatório.

O nodo de **Entrada** (**E**) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de **Saída** (**S**) é ativado ao completar a operação. Se o nodo de **Entrada** (**E**) estiver selecionado para acionamento por nível o bloco será executado

sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX201 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de **Entrada** (**E**) ser ativado por borda o bloco só é executado uma vez na borda de subida de **E**, aguardando este nodo desligar para permitir novo acionamento.

Abaixo temos um exemplo de uso deste bloco, em conjunto com o bloco de Shift Tabela, para o cálculo de média de valores de uma entrada analógica:



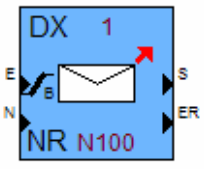
Exemplo de Programa Aplicativo: **Tabela - Somatório Tabela.dxg**

Atenção: Bloco disponível apenas em controlador µDX201 versão 3.62 ou superior..

Veja também:
TABELA - Shift Tabela

DXNET - Escreve Nodo

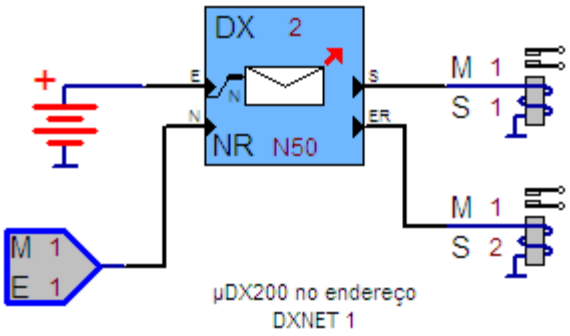
Este bloco permite transferir o estado de um nodo local para um nodo remoto em outro µDX200 ligado à rede DXNET. O nodo transmitido via rede DXNET é a conexão Nodo (N). Quando a conexão Entrada (E) é energizada o estado do nodo N é transferido para o endereço e nodo remoto especificados no bloco. Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).



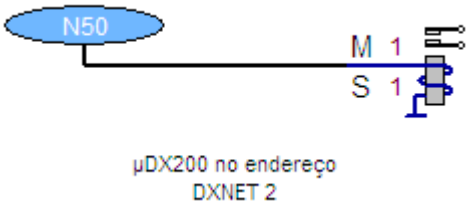
DXNET e NodoRemoto são variáveis tipo word (0 a 65535) que indicam o endereço do µDX200 destino e o número absoluto do nodo, respectivamente. Como um µDX200 não pode saber quais os nomes foram atribuídos a nodos em outro µDX200 pelo programa aplicativo, é necessário apontar para endereços absolutos de nodos. Para isso basta atribuir a estes nodos valores absolutos usando a designação Nxxx (letra N ou n seguida do número do nodo). Os nodos de 0 a

15 são nodos de sistema, e não devem ser usados como nodos de uso geral no programa aplicativo.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **DXNET - Escreve Nodo.dxg**

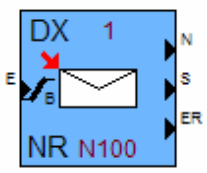


Exemplo de Programa Aplicativo: **DXNET - Escreve Nodo 2.dxg**

Os programas de exemplo acima transmitem o estado da entrada digital E1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) ligado ao µDX200 endereço DXNET 1 para a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) ligado ao µDX200 com endereço DXNET 2.

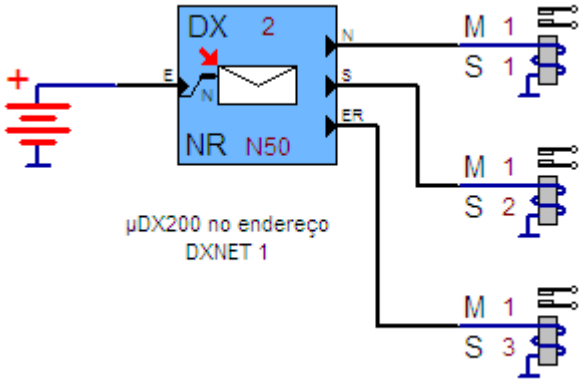
DXNET - Consulta Nodo

Este bloco permite consultar o estado de um nodo remoto em outro µDX200 ligado à rede DXNET, transferindo seu estado para um nodo local. O estado do nodo consultado via rede DXNET é colocado na conexão Nodo (N) quando o nodo Entrada (E) é energizado. Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).

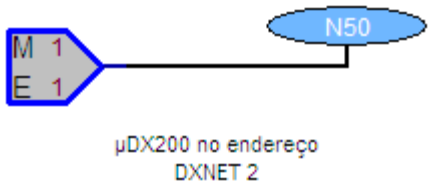


DXNET e NodoRemoto são variáveis tipo word (0 a 65535) que indicam o endereço do µDX200 e o número absoluto do nodo a ser consultado, respectivamente. Como um µDX200 não pode saber quais os nomes foram atribuídos a nodos em outro µDX200 pelo programa aplicativo, é necessário especificar endereços absolutos de nodos. Para isso basta atribuir a estes nodos valores absolutos usando a designação Nxxx (letra N ou n seguida do número do nodo). Os nodos de 0 a 15 são nodos de sistema, e não devem ser usados como nodos de uso geral no programa aplicativo.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **DXNET - Consulta Nodo.dxg**



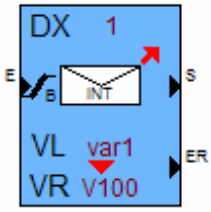
Exemplo de Programa Aplicativo: **DXNET - Consulta Nodo 2.dxg**

O primeiro programa de exemplo, instalado no µDX200 endereço DXNET 1, lê o estado do nodo 50 do µDX200 endereço DXNET 2. Como este nodo está ligado à entrada digital E1 do módulo M1 de Expansão de Entradas/Saídas (µDX210), o estado desta entrada E1 no µDX200 DXNET 2 se reflete na saída S1 do µDX200 DXNET 1.

DXNET - Escreve Variável Inteira

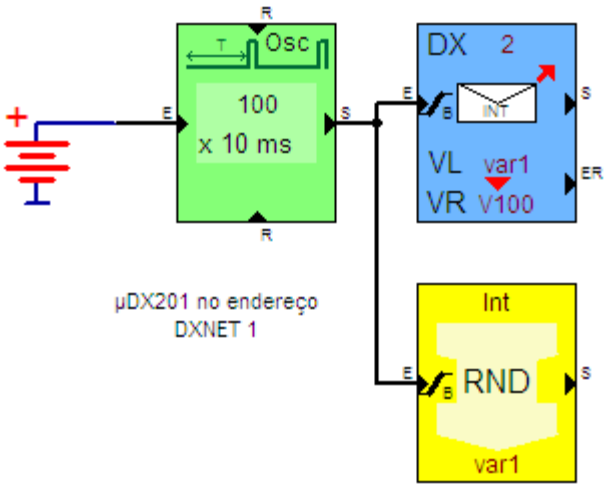
Este bloco permite transferir o valor de uma variável inteira local para uma variável inteira remota em outro µDX200 ligado à rede DXNET.

Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).

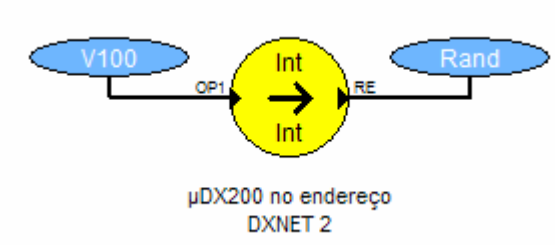


DXNET é uma variável tipo word (0 a 65535) que indica o endereço do µDX200 destino. Já VarLocal e VarRemoto são, respectivamente, a variável local a ser transmitida e a variável remota que irá receber o valor de VarLocal. Como um µDX200 não pode saber quais os nomes foram atribuídos à variáveis em outro µDX200 pelo programa aplicativo, é necessário apontar para endereços absolutos de variáveis. Para isso basta atribuir a estas variáveis valores absolutos usando a designação Vxxx (letra V ou v seguida do número da variável). As variáveis de v0 a v17 são variáveis de sistema, e não devem ser usadas como variáveis de uso geral no programa aplicativo. Já a variável local (VarLocal) pode ser um nome de variável usado no programa, pois o bloco DXNET pode reconhecer as variáveis locais pelo nome atribuído a elas.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



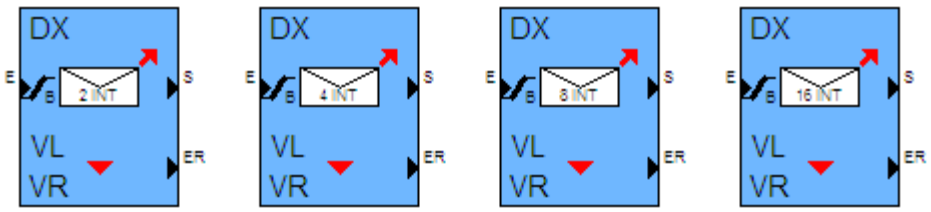
Exemplo de Programa Aplicativo: **DXNET - Escreve Variável Inteira.dxg**



Exemplo de Programa Aplicativo: **DXNET - Escreve Variável Inteira 2.dxg**

O programa de exemplo acima transmite o valor da variável inteira var1, a cada segundo, para a variável v100 do µDX200 endereço DXNET 2. Note que neste µDX200 a variável v100 é transferida para a variável Rand. Já a valor da variável var1 transmitida a cada segundo é constantemente modificada pelo bloco de RND (número randômico).

No caso de versões iguais ou superiores a 3.38 do Controlador µDX201 é possível transmitir mais de uma variável simultaneamente pela rede DXNET. Com os blocos abaixo pode-se transmitir 2, 4, 8 ou 16 variáveis inteiras. Note que o bloco indica apenas a variável inicial a ser transmitida e a variável inicial para receber os dados. Ou seja, as variáveis são transmitidas em seqüência.

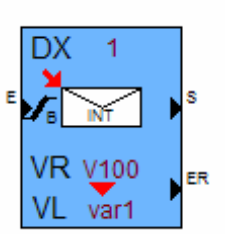


Veja também:
DXNET - Escreve Variável LongInt
DXNET - Escreve Variável Word
DXNET - Escreve Variável Real

DXNET - Consulta Variável Inteira

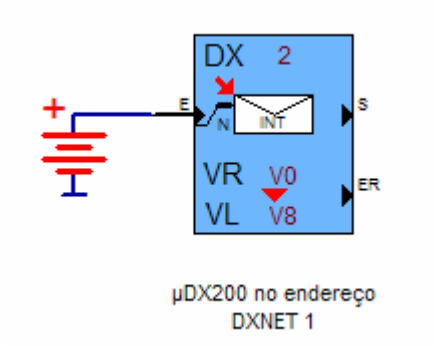
Este bloco permite consultar o valor de uma variável inteira remota em outro µDX200 ligado à rede DXNET e transferir o valor lido para uma variável inteira local.

Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).



DXNET é uma variável tipo word (0 a 65535) que indica o endereço do µDX200 remoto. Já VarLocal e VarRemoto são, respectivamente, a variável local que irá receber o valor lido e a variável remota que irá ser consultada. Como um µDX200 não pode saber quais os nomes foram atribuídos à variáveis em outro µDX200 pelo programa aplicativo, é necessário apontar para endereços absolutos de variáveis. Para isso basta atribuir a estas variáveis valores absolutos usando a designação Vxxx (letra V ou v seguida do número da variável). As variáveis de v0 a v17 são variáveis de sistema, e não devem ser usadas como variáveis de uso geral no programa aplicativo. Já a variável local (VarLocal) pode ser um nome de variável usado no programa, pois o bloco DXNET pode reconhecer as variáveis locais pelo nome atribuído a elas.

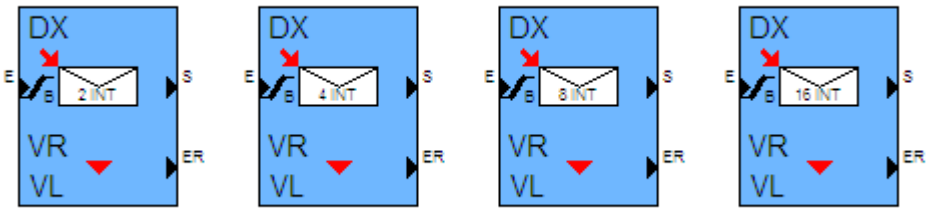
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **DXNET - Consulta Variável Inteira.dxg**

O programa de exemplo acima, instalado no µDX200 endereço DXNET 1, lê constantemente o valor da variável v0 no µDX200 endereço DXNET 2, e transfere o valor lido para a variável local v8. Como a variável v0 está associada a entrada analógica E1, e a variável v8 está associada a saída analógica S1, resulta que qualquer sinal aplicado a entrada analógica E1 do µDX200 DXNET 2 é replicado na saída analógica S1 do µDX200 DXNET 1.

No caso de versões iguais ou superiores a 3.38 do Controlador µDX201 é possível ler mais de uma variável simultaneamente pela rede DXNET. Com os blocos abaixo pode-se ler 2, 4, 8 ou 16 variáveis inteiras. Note que o bloco indica apenas a variável inicial a ser lida remotamente e a variável inicial para receber os dados. Ou seja, as variáveis são lidas em seqüência.

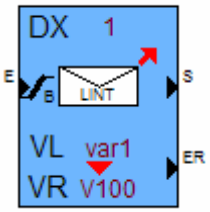


Veja também:
DXNET - Consulta Variável LongInt
DXNET - Consulta Variável Word
DXNET - Consulta Variável Real

DXNET - Escreve Variável LongInt

Este bloco permite transferir o valor de uma variável longint local para uma variável longint remota em outro µDX200 ligado à rede DXNET.

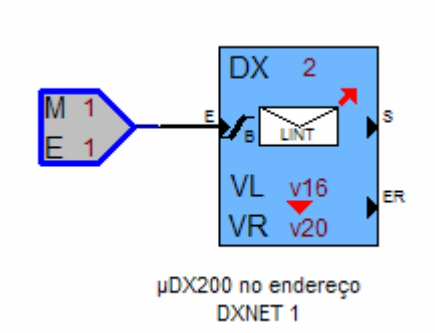
Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).



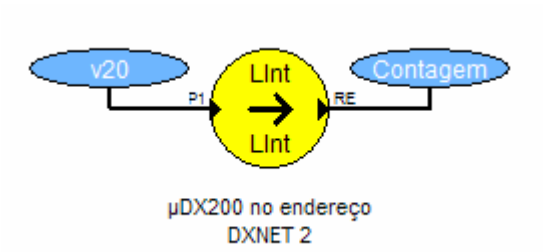
DXNET é uma variável tipo word (0 a 65535) que indica o endereço do µDX200 destino. Já

VarLocal e VarRemoto são, respectivamente, a variável local a ser transmitida e a variável remota que irá receber o valor de VarLocal. Como um µDX200 não pode saber quais os nomes foram atribuídos à variáveis em outro µDX200 pelo programa aplicativo, é necessário apontar para endereços absolutos de variáveis. Para isso basta atribuir a estas variáveis valores absolutos usando a designação Vxxx (letra V ou v seguida do número da variável). As variáveis de v0 a v17 são variáveis de sistema, e não devem ser usadas como variáveis de uso geral no programa aplicativo. Já a variável local (VarLocal) pode ser um nome de variável usado no programa, pois o bloco DXNET pode reconhecer as variáveis locais pelo nome atribuído a elas.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



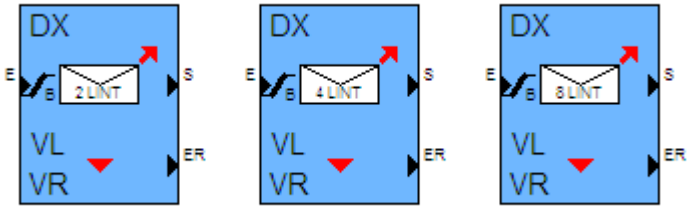
Exemplo de Programa Aplicativo: **DXNET - Escreve Variável LongInt.dxg**



Exemplo de Programa Aplicativo: **DXNET - Escreve Variável LongInt 2.dxg**

O programa de exemplo acima transmite o valor da variável longint v16, sempre que a entrada E1 do módulo M1 de Expansão for energizada, para a variável v20 do µDX200 endereço DXNET 2. Note que neste µDX200 a variável v20 é transferida para a variável Contagem. Como a variável v16 no controlador µDX200 está associada ao contador rápido por quadratura, pulsos nas entradas digitais E9 e E10 incrementam ou decrementam o valor da variável.

No caso de versões iguais ou superiores a 3.38 do Controlador µDX201 é possível transmitir mais de uma variável simultaneamente pela rede DXNET. Com os blocos abaixo pode-se transmitir 2, 4 ou 8 variáveis longint. Note que o bloco indica apenas a variável inicial a ser transmitida e a variável inicial para receber os dados. Ou seja, as variáveis são transmitidas em seqüência.

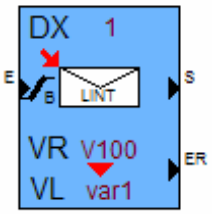


Veja também:
DXNET - Escreve Variável Inteira
DXNET - Escreve Variável Word
DXNET - Escreve Variável Real

DXNET - Consulta Variável LongInt

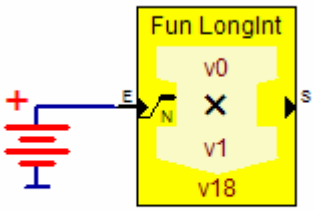
Este bloco permite consultar o valor de uma variável longint remota em outro µDX200 ligado à rede DXNET e transferir o valor lido para uma variável longint local.

Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).

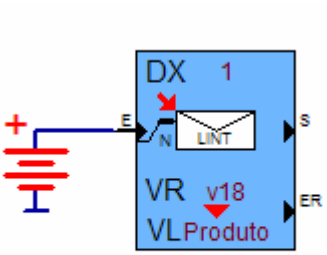


DXNET é uma variável tipo word (0 a 65535) que indica o endereço do µDX200 remoto. Já VarLocal e VarRemoto são, respectivamente, a variável local que irá receber o valor lido e a variável remota que irá ser consultada. Como um µDX200 não pode saber quais os nomes foram atribuídos à variáveis em outro µDX200 pelo programa aplicativo, é necessário apontar para endereços absolutos de variáveis. Para isso basta atribuir a estas variáveis valores absolutos usando a designação Vxxx (letra V ou v seguida do número da variável). As variáveis de v0 a v17 são variáveis de sistema, e não devem ser usadas como variáveis de uso geral no programa aplicativo. Já a variável local (VarLocal) pode ser um nome de variável usado no programa, pois o bloco DXNET pode reconhecer as variáveis locais pelo nome atribuído a elas.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



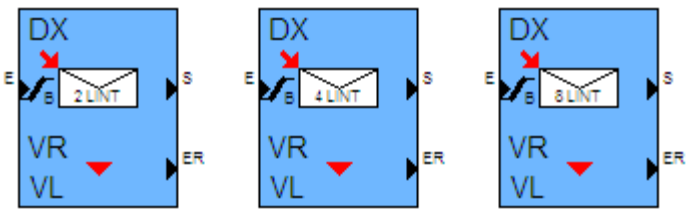
Exemplo de Programa Aplicativo: **DXNET - Consulta Variável LongInt.dxg**



Exemplo de Programa Aplicativo: **DXNET - Consulta Variável LongInt 2.dxg**

O programa de exemplo acima, instalado no µDX200 endereço DXNET 1, multiplica os valores lidos nas entradas analógicas E1 e E2 (respectivamente associadas às variáveis v0 e v1), colocando o resultado na variável longint v18 (por ser uma variável longint, é usada v18 e v19). Já o programa instalado no µDX200 endereço DXNET 2 lê constantemente o valor da variável longint v18 do µDX200 DXNET 1 e coloca o valor lido na variável longint Produto.

No caso de versões iguais ou superiores a 3.38 do Controlador µDX201 é possível ler mais de uma variável simultaneamente pela rede DXNET. Com os blocos abaixo pode-se ler 2, 4 ou 8 variáveis longint. Note que o bloco indica apenas a variável inicial a ser lida remotamente e a variável inicial para receber os dados. Ou seja, as variáveis são lidas em seqüência.

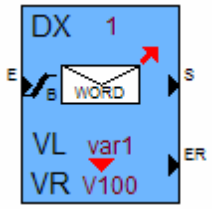


Veja também:
DXNET - Consulta Variável Inteira
DXNET - Consulta Variável Word
DXNET - Consulta Variável Real

DXNET - Escreve Variável Word

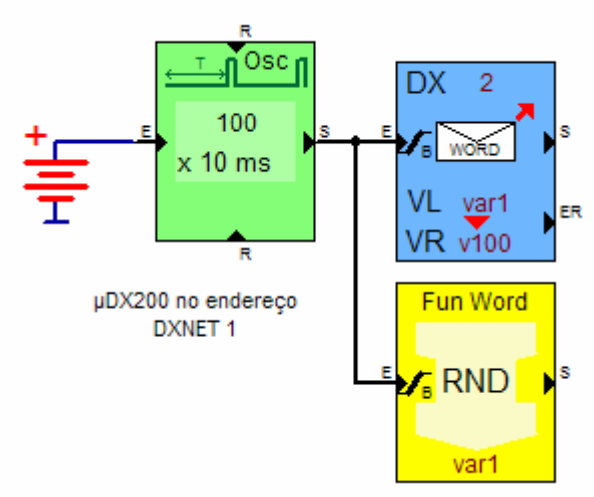
Este bloco permite transferir o valor de uma variável word local para uma variável inteira remota em outro µDX200 ligado à rede DXNET.

Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).

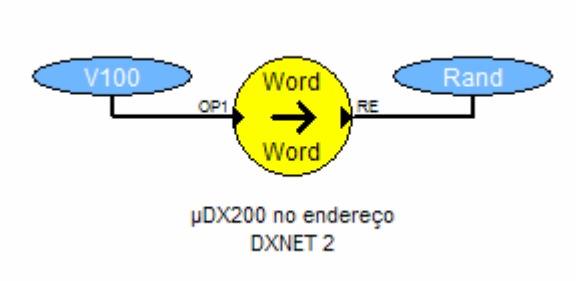


DXNET é uma variável tipo word (0 a 65535) que indica o endereço do µDX200 destino. Já VarLocal e VarRemoto são, respectivamente, a variável local a ser transmitida e a variável remota que irá receber o valor de VarLocal. Como um µDX200 não pode saber quais os nomes foram atribuídos à variáveis em outro µDX200 pelo programa aplicativo, é necessário apontar para endereços absolutos de variáveis. Para isso basta atribuir a estas variáveis valores absolutos usando a designação Vxxx (letra V ou v seguida do número da variável). As variáveis de v0 a v17 são variáveis de sistema, e não devem ser usadas como variáveis de uso geral no programa aplicativo. Já a variável local (VarLocal) pode ser um nome de variável usado no programa, pois o bloco DXNET pode reconhecer as variáveis locais pelo nome atribuído a elas.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



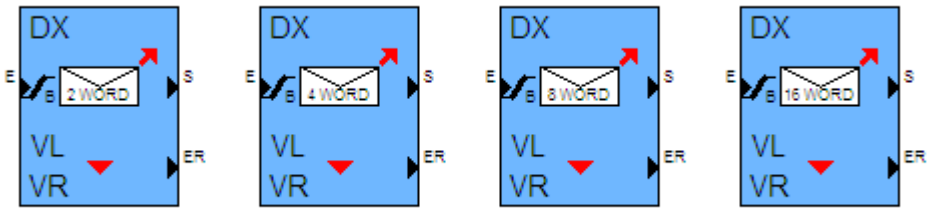
Exemplo de Programa Aplicativo: **DXNET - Escreve Variável Word.dxg**



Exemplo de Programa Aplicativo: **DXNET - Escreve Variável Word 2.dxg**

O programa de exemplo acima transmite o valor da variável word var1, a cada segundo, para a variável v100 do µDX200 endereço DXNET 2. Note que neste µDX200 a variável v100 é transferida para a variável Rand. Já a valor da variável var1 transmitida a cada segundo é constantemente modificada pelo bloco de RND (número randômico).

No caso de versões iguais ou superiores a 3.38 do Controlador µDX201 é possível transmitir mais de uma variável simultaneamente pela rede DXNET. Com os blocos abaixo pode-se transmitir 2, 4, 8 ou 16 variáveis word. Note que o bloco indica apenas a variável inicial a ser transmitida e a variável inicial para receber os dados. Ou seja, as variáveis são transmitidas em seqüência.

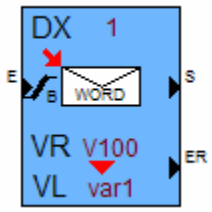


Veja também:
DXNET - Escreve Variável Inteira
DXNET - Escreve Variável LongInt
DXNET - Escreve Variável Real

DXNET - Consulta Variável Word

Este bloco permite consultar o valor de uma variável word remota em outro µDX200 ligado à rede DXNET e transferir o valor lido para uma variável word local.

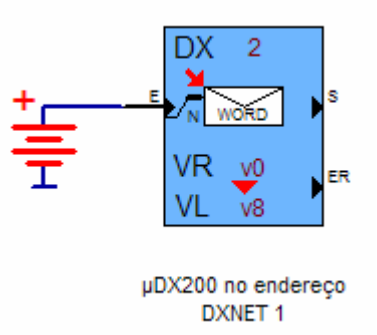
Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).



DXNET é uma variável tipo word (0 a 65535) que indica o endereço do µDX200 remoto. Já VarLocal e VarRemoto são, respectivamente, a variável local que irá receber o valor lido e a variável remota que irá ser consultada. Como um µDX200 não pode saber quais os nomes foram atribuídos à variáveis em outro µDX200 pelo programa aplicativo, é necessário apontar para endereços absolutos de variáveis. Para isso basta atribuir a estas variáveis valores absolutos

usando a designação Vxxx (letra V ou v seguida do número da variável). As variáveis de v0 a v17 são variáveis de sistema, e não devem ser usadas como variáveis de uso geral no programa aplicativo. Já a variável local (VarLocal) pode ser um nome de variável usado no programa, pois o bloco DXNET pode reconhecer as variáveis locais pelo nome atribuído a elas.

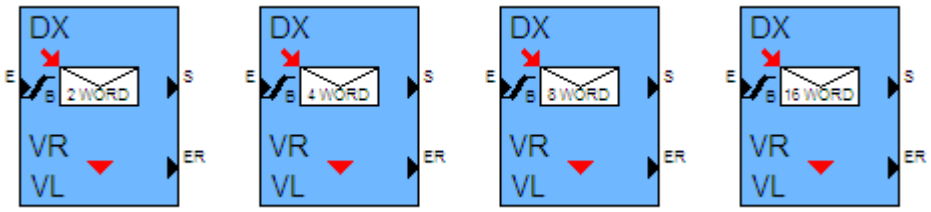
O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



Exemplo de Programa Aplicativo: **DXNET - Consulta Variável Word.dxg**

O programa de exemplo acima, instalado no µDX200 endereço DXNET 1, lê constantemente o valor da variável v0 no µDX200 endereço DXNET 2, e transfere o valor lido para a variável local v8. Como a variável v0 está associada a entrada analógica E1, e a variável v8 está associada a saída analógica S1, resulta que qualquer sinal aplicado a entrada analógica E1 do µDX200 DXNET 2 é replicado na saída analógica S1 do µDX200 DXNET 1.

No caso de versões iguais ou superiores a 3.38 do Controlador µDX201 é possível ler mais de uma variável simultaneamente pela rede DXNET. Com os blocos abaixo pode-se ler 2, 4, 8 ou 16 variáveis word. Note que o bloco indica apenas a variável inicial a ser lida remotamente e a variável inicial para receber os dados. Ou seja, as variáveis são lidas em seqüência.

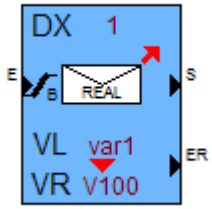


Veja também:
DXNET - Consulta Variável Inteira
DXNET - Consulta Variável LongInt
DXNET - Consulta Variável Real

DXNET - Escreve Variável Real

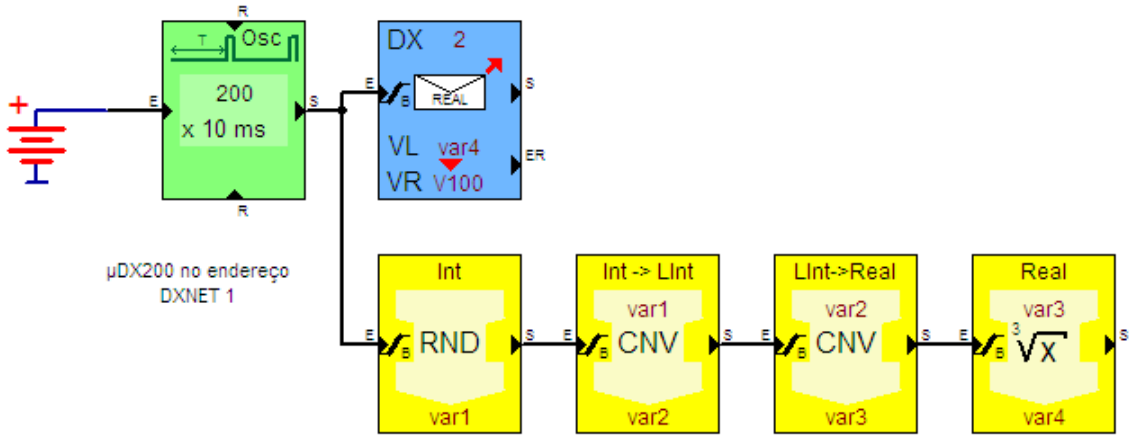
Este bloco permite transferir o valor de uma variável real local para uma variável real remota em outro µDX200 ligado à rede DXNET.

Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).

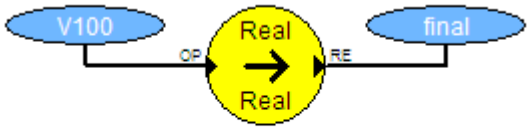


DXNET é uma variável tipo word (0 a 65535) que indica o endereço do µDX200 destino. Já VarLocal e VarRemoto são, respectivamente, a variável local a ser transmitida e a variável remota que irá receber o valor de VarLocal. Como um µDX200 não pode saber quais os nomes foram atribuídos à variáveis em outro µDX200 pelo programa aplicativo, é necessário apontar para endereços absolutos de variáveis. Para isso basta atribuir a estas variáveis valores absolutos usando a designação Vxxx (letra V ou v seguida do número da variável). As variáveis de v0 a v17 são variáveis de sistema, e não devem ser usadas como variáveis de uso geral no programa aplicativo. Já a variável local (VarLocal) pode ser um nome de variável usado no programa, pois o bloco DXNET pode reconhecer as variáveis locais pelo nome atribuído a elas.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.



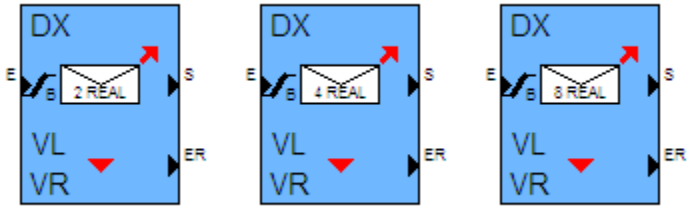
Exemplo de Programa Aplicativo: **DXNET - Escreve Variável Real.dxg**



µDX200 no endereço
DXNET 2

Exemplo de Programa Aplicativo: **DXNET - Escreve Variável Real 2.dxg**

No caso de versões iguais ou superiores a 3.38 do Controlador µDX201 é possível transmitir mais de uma variável simultaneamente pela rede DXNET. Com os blocos abaixo pode-se transmitir 2, 4 ou 8 variáveis reais. Note que o bloco indica apenas a variável inicial a ser transmitida e a variável inicial para receber os dados. Ou seja, as variáveis são transmitidas em seqüência.

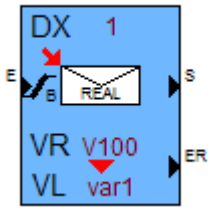


Veja também:
DXNET - Escreve Variável Inteira
DXNET - Escreve Variável LongInt
DXNET - Escreve Variável Word

DXNET - Consulta Variável Real

Este bloco permite consultar o valor de uma variável real remota em outro µDX200 ligado à rede DXNET e transferir o valor lido para uma variável real local.

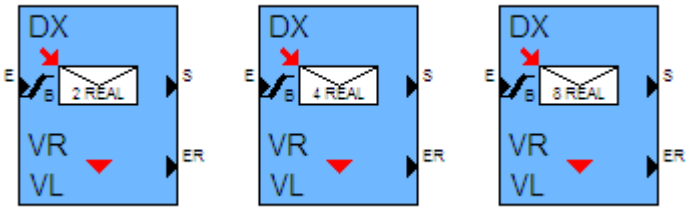
Caso a transmissão não obtenha sucesso é acionado o nodo de Erro (ER).



DXNET é uma variável tipo word (0 a 65535) que indica o endereço do µDX200 remoto. Já VarLocal e VarRemoto são, respectivamente, a variável local que irá receber o valor lido e a variável remota que irá ser consultada. Como um µDX200 não pode saber quais os nomes foram atribuídos à variáveis em outro µDX200 pelo programa aplicativo, é necessário apontar para endereços absolutos de variáveis. Para isso basta atribuir a estas variáveis valores absolutos usando a designação Vxxx (letra V ou v seguida do número da variável). As variáveis de v0 a v17 são variáveis de sistema, e não devem ser usadas como variáveis de uso geral no programa aplicativo. Já a variável local (VarLocal) pode ser um nome de variável usado no programa, pois o bloco DXNET pode reconhecer as variáveis locais pelo nome atribuído a elas.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação. Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX200 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

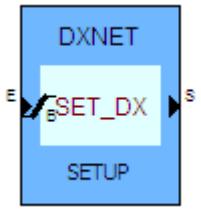
No caso de versões iguais ou superiores a 3.38 do Controlador µDX201 é possível ler mais de uma variável simultaneamente pela rede DXNET. Com os blocos abaixo pode-se ler 2, 4 ou 8 variáveis reais. Note que o bloco indica apenas a variável inicial a ser lida remotamente e a variável inicial para receber os dados. Ou seja, as variáveis são lidas em seqüência.



Veja também:
DXNET - Consulta Variável Inteira
DXNET - Consulta Variável LongInt
DXNET - Consulta Variável Word

DXNET - Setup

O bloco **Setup DXNET** permite especificar o endereço DXNET do controlador µDX201. Cuidado ao especificar esse endereço, pois valores fora da tabela de scan não serão varridos na rede DXNET e, portanto, não permitirão comunicação com outros controladores µDX201. O valor especificado através desse bloco é usado prioritariamente em relação ao especificado na **Configuração de Hardware**. No caso de um hard reset do CLP o valor da **Configuração de Hardware** é restabelecido.



Operando é o operando cujo valor será transferido para o endereço de DXNET.

Note que Operando pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

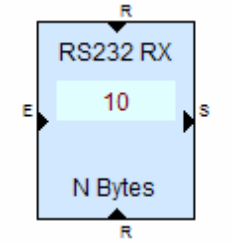
Note que variáveis word podem assumir valores entre 0 e 65535 (0 e 0FFFFh em hexadecimal).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX201 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco

só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

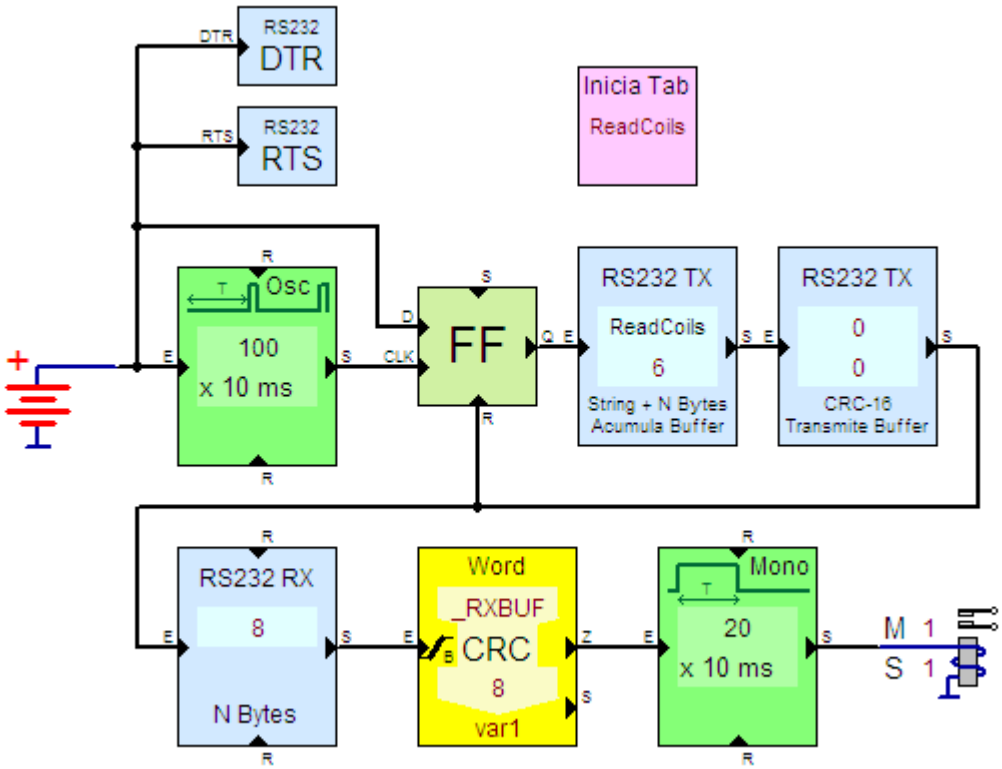
RS232 - RX N Bytes

Este bloco permite receber um número fixo de bytes pela porta serial RS232 do µDX200.



Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até receber via RS232 o número de bytes programado, ou ser inibido pela energização dos nodos de Reset (R). Quando for recebido o número de bytes programado no bloco RX-N bytes será gerado um pulso no nodo de Saída (S) do bloco. Caso o nodo de Entrada (E) permaneça ligado o bloco é novamente acionado, passando a aguardar mais um conjunto de N bytes via serial.

Núm.Bytes é uma variável tipo inteira que indica o número de bytes a receber pela porta serial RS232 do µDX200.



Exemplo de Programa Aplicativo: **RS232 - RX N Bytes.dxg**

O programa de exemplo demonstra a facilidade de implementação de protocolos no µDX200. No caso, o µDX200 transmite um comando ModBus via serial a cada segundo e aguarda a resposta do dispositivo ModBus conectado a serial. Caso a resposta seja recebida corretamente (CRC-16

correto) a saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210) é acionada por 0,2 segundos. Os pacotes de dados são os seguintes:

µDX200 → Dispositivo ModBus

Endereço	Comando	Endereço Inicial (MSB)	Endereço Inicial (LSB)	Número de bits (MSB)	Número de bits (LSB)	CRC-16 (LSB)	CRC-16 (MSB)
01h	01h	00h	00h	00h	15h	FDh	C5h

Dispositivo ModBus → µDX200

Endereço	Comando	Número de bytes	Dados	Dados	Dados	CRC-16 (LSB)	CRC-16 (MSB)
01h	01h	03h	Dado	Dado	Dado	CRC	CRC

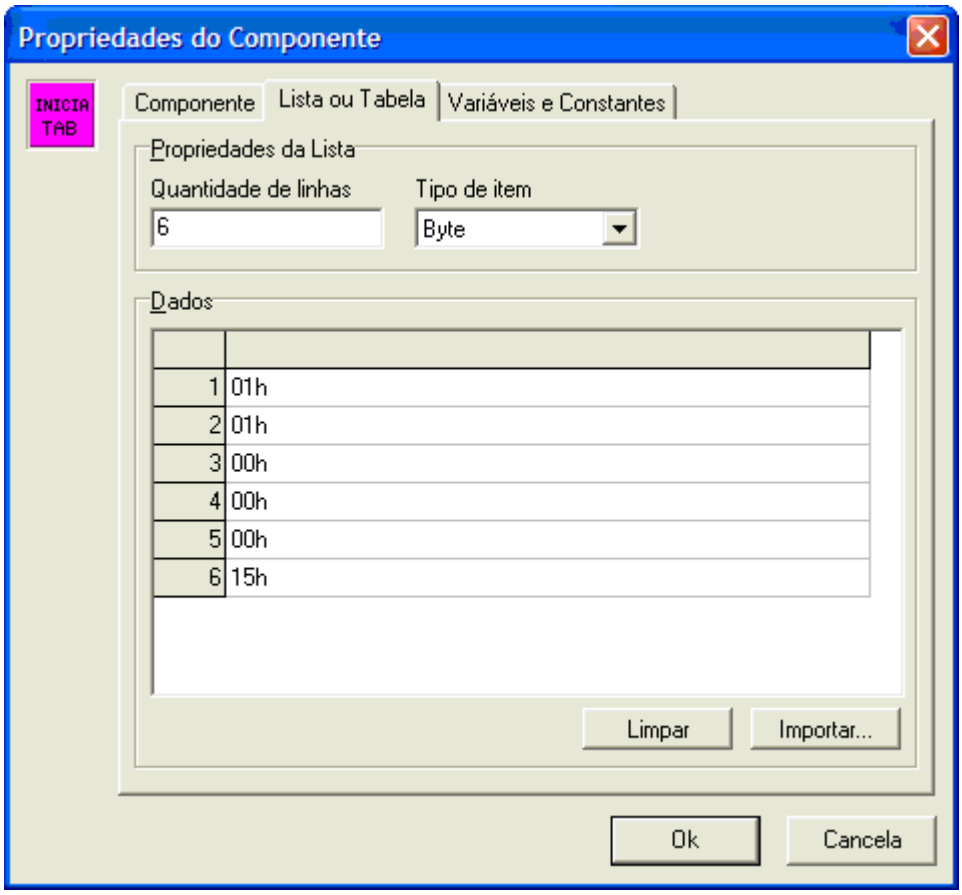
Note que o dispositivo ModBus está no endereço 01, e o comando utilizado é o comando 01 - Read Coil Status. Como foi requisitada a leitura de 15h bits (ou seja, 21 bits), a resposta é composta de 3 bytes (24 bits). Especificações completas a respeito do protocolo ModBus podem ser obtidas junto a empresa Modicon - www.modicon.com.

São utilizados vários blocos de comunicação serial neste exemplo. Note que o oscilador, a cada segundo, liga a saída Q do flip-flop. Este, por sua vez, dispara a escrita no buffer de transmissão serial do string de comando ModBus 01h 01h 00h 00h 00h 15h. A seguir, é calculado o CRC-16 da mensagem ModBus (C5FDh) e a mensagem + CRC é transmitida pela RS232. A transmissão gera um pulso na saída S do bloco de transmissão de CRC, o que reinicializa o flip-flop e dispara a leitura dos 8 bytes de resposta do dispositivo.

Uma vez recebidos os 8 bytes, é gerado um pulso no nodo de saída S do bloco de RX N bytes que dispara a checagem de CRC-16 da resposta recebida. Note que este bloco aponta para o endereço absoluto de memória do µDX200 _RXBUF. Este é o endereço inicial do buffer de recepção serial RS232. Caso o CRC-16 corresponda à mensagem recebida será gerado um pulso na saída Z do bloco. Este pulso é alargado pelo monoestável de 200ms e aplicado na saída S1 do módulo M1 de Expansão (µDX210). Uma possível mensagem recebida pelo µDX200 seria (com todos os nodos desligados):

01h 01h 03h 00h 00h 00h 3Ch 4Eh

Os nodos RTS e DTR (que correspondem a sinais de handshake da porta serial) são forçados a nível 1 (sempre ligados). A tabela ReadCoils é inicializada com os seguintes valores:



Atenção: *É necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Os blocos RS232-RX são ativados pela detecção de nível alto no nodo de entrada (E) do bloco. Uma vez ativados só são desativados pela recepção via serial do string programado, ou energizando o nodo de reset (R) do bloco. E um bloco de RS232-RX ativo irá bloquear todos os demais blocos RS232-RX existentes no programa aplicativo. Assim, para comutar entre blocos RS232-RX no programa é necessário resetar todos os demais blocos deste tipo existentes no programa para µDX200.*

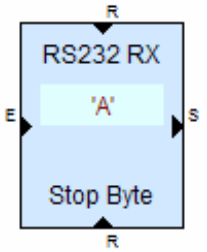
Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial*

principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.

Veja também:
RS232 - RX Stop Byte
RS232 - RX Start String + N Bytes
RS232 - RX Start String + Stop Byte

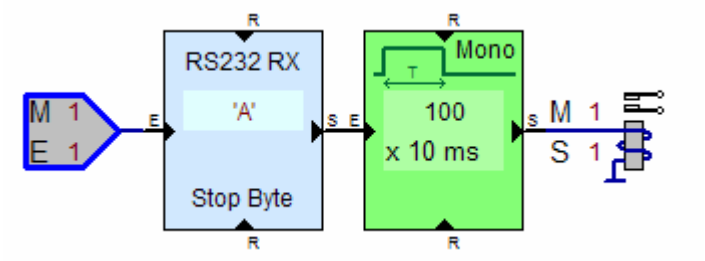
RS232 - RX Stop Byte

Este bloco permite receber bytes pela porta serial RS232 do µDX200 até que é detectado o recebimento do stop byte especificado.



Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até receber via RS232 o byte especificado como o de finalização (stop byte), ou ser inibido pela energização dos nodos de Reset (R). Quando for recebido o stop byte programado no bloco RX-Stop Byte será gerado um pulso no nodo de Saída (S) do bloco. Caso o nodo de Entrada (E) permaneça ligado o bloco é novamente acionado, passando a aguardar mais um stop byte via serial. O stop byte não é incluído no buffer de recepção serial (endereço M780h).

StopByte é uma constante do tipo byte que indica qual o stop byte a receber pela porta serial RS232 do µDX200 de forma a encerrar a recepção. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. Assim, podemos colocar, por exemplo, o valor 65 para designar o caracter 'A' a ser recebido como stop byte (já que a representação ASCII de 'A' é 65). Podemos também representar diretamente o caracter ASCII a ser utilizado como stop byte representando-o entre apóstrofes no operando StopByte.



Exemplo de Programa Aplicativo: **RS232 - RX Stop Byte.dxg**

O exemplo aguarda mensagens terminadas pelo caracter 'A' via serial do µDX200. Sempre que a entrada digital E1 da Expansão M1 é energizada o µDX200 captura os bytes recebidos pela serial até receber o byte 65 ('A'), quando aciona momentaneamente o nodo S que liga a saída S1 por 1 s.

Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

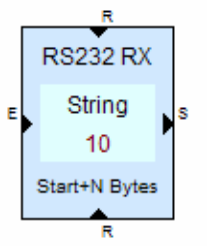
Atenção: *Os blocos RS232-RX são ativados pela detecção de nível alto no nodo de entrada (E) do bloco. Uma vez ativados só são desativados pela recepção via serial do string programado, ou energizando o nodo de reset (R) do bloco. E um bloco de RS232-RX ativo irá bloquear todos os demais blocos RS232-RX existentes no programa aplicativo. Assim, para comutar entre blocos RS232-RX no programa é necessário resetar todos os demais blocos deste tipo existentes no programa para µDX200.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - RX N Bytes
RS232 - RX Start String + N Bytes
RS232 - RX Start String + Stop Byte

RS232 - RX Start String + N Bytes

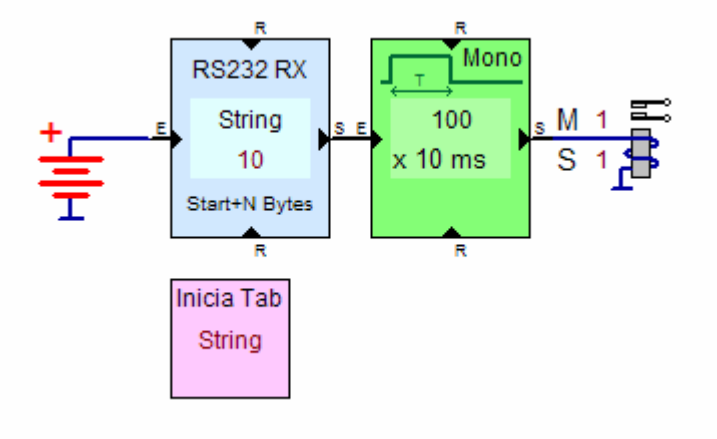
Este bloco permite receber bytes pela porta serial RS232 do µDX200 a partir da detecção de um string. A seguir, são capturados o número de bytes especificado.



Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, aguardando a detecção do string especificado. Uma vez detectado o string o bloco passa a receber o número de bytes programado. O bloco é desativado ao receber o número de bytes selecionado ou ao ser inibido pela energização dos nodos de Reset (R). Quando forem recebidos os bytes programados no bloco RX-Start+NBytes será gerado um pulso no nodo de Saída (S) do bloco. Caso o nodo de Entrada (E) permaneça ligado o bloco é novamente acionado, passando a aguardar mais uma vez

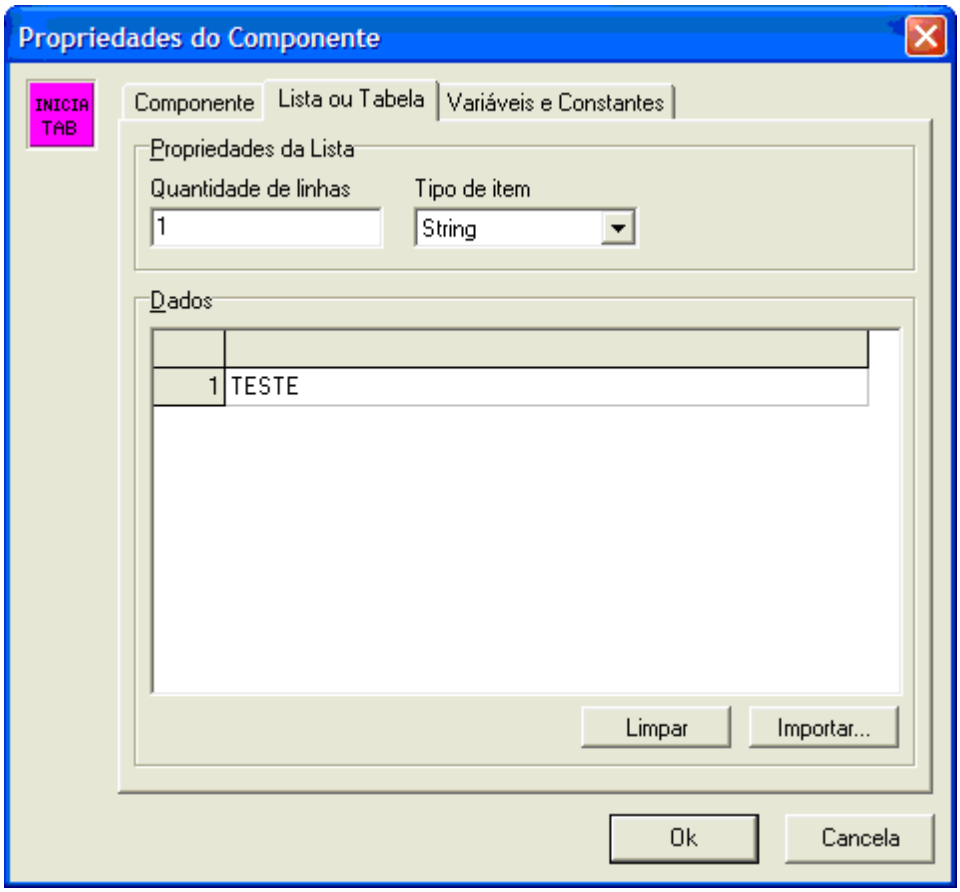
um start string via serial. Note que o start string não é incluído na mensagem salva no buffer de recepção serial, ou seja, ele é perdido.

Note que StartString, Núm.Bytes e TamStart podem ser variáveis word ou inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). StartString deve apontar para um string a ser esperado como inicializador da mensagem via serial, Núm.Bytes especifica o número de bytes da mensagem após o string inicial, e TamStart indica qual o tamanho do string inicial.



Exemplo de Programa Aplicativo: **RS232 - RX Start String + N Bytes.dxg**

O exemplo aguarda mensagens iniciadas pelo string "TESTE". A seguir, captura 10 caracteres recebidos pela serial e liga a saída S1 do módulo de Expansão M1 (µDX210) por 1 segundo. Note que, como o nodo de entrada (E) do bloco de RX-Start+NBytes está sempre ligado, o bloco está constantemente esperando novo string de inicialização. A tabela é inicializada com:



Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Os blocos RS232-RX são ativados pela detecção de nível alto no nodo de entrada (E) do bloco. Uma vez ativados só são desativados pela recepção via serial do string programado, ou energizando o nodo de reset (R) do bloco. E um bloco de RS232-RX ativo irá bloquear todos os demais blocos RS232-RX existentes no programa aplicativo. Assim, para comutar entre blocos RS232-RX no programa é necessário resetar todos os demais blocos deste tipo existentes no programa para µDX200.*

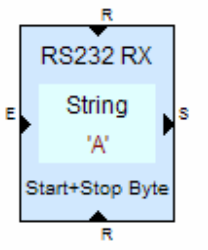
Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - RX N Bytes
RS232 - RX Stop Byte
RS232 - RX Start String + Stop Byte

RS232 - RX Start String + Stop Byte

Este bloco permite receber bytes pela porta serial RS232 do µDX200 a partir da detecção de um string.

A seguir, são capturados os bytes até a detecção de um stop byte.

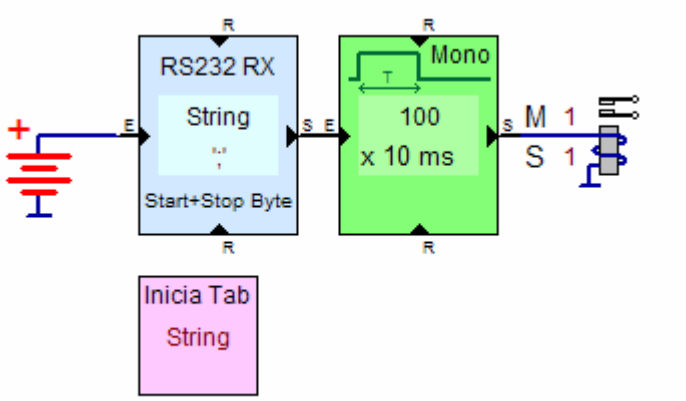


Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, aguardando a detecção do string especificado. Uma vez detectado o string o bloco passa a receber caracteres, até que seja recebido o stop byte programado para finalização da mensagem. O bloco é desativado ao receber o stop byte selecionado ou ao ser inibido pela energização dos nodos de

Reset (R). Quando for recebido o stop byte programado no bloco RX-Start+StopByte será gerado um pulso no nodo de Saída (S) do bloco. Caso o nodo de Entrada (E) permaneça ligado o bloco é novamente acionado, passando a aguardar mais uma vez um start string via serial. Note que o start string e stop byte não são incluídos na mensagem salva no buffer de recepção serial, ou seja, eles são perdidos.

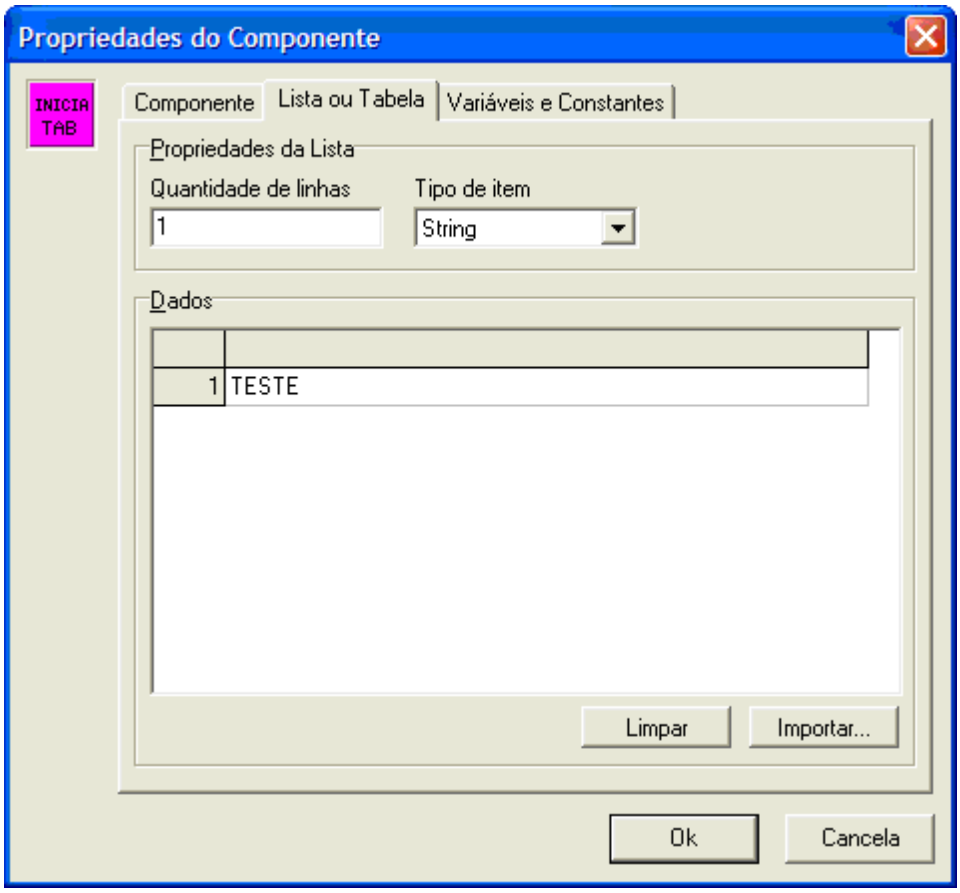
Note que StartString pode ser uma variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). StartString deve apontar para um string a ser esperado como inicializador da mensagem via serial, e TamStart indica qual o tamanho do string inicial. TamStart deve ser uma constante do tipo byte (já que o µDX200 não aceita variáveis do tipo byte este operando deve ser necessariamente uma constante).

StopByte é uma constante do tipo byte que indica qual o stop byte a receber pela porta serial RS232 do µDX200 de forma a encerrar a recepção. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. Assim, podemos colocar, por exemplo, o valor 65 para designar o caracter 'A' a ser recebido como stop byte (já que a representação ASCII de 'A' é 65). Podemos também representar diretamente o caracter ASCII a ser utilizado como stop byte representando-o entre apóstrofes no operando StopByte.



Exemplo de Programa Aplicativo: **RS232 - RX Start String + Stop Byte.dxg**

O exemplo aguarda mensagens iniciadas pelo string "TESTE". A seguir, captura caracteres recebidos pela serial até receber o stop byte (no caso, o caracter ';' - ponto e vírgula), quando liga a saída S1 do módulo de Expansão M1 (µDX210) por 1 segundo. Note que, como o nodo de entrada (E) do bloco de RX-Start+stopByte está sempre ligado, o bloco está constantemente esperando novo string de inicialização. A tabela é inicializada com:



Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Os blocos RS232-RX são ativados pela detecção de nível alto no nodo de entrada (E) do bloco. Uma vez ativados só são desativados pela recepção via serial do string programado, ou energizando o nodo de reset (R) do bloco. E um bloco de RS232-RX ativo irá bloquear todos os demais blocos RS232-RX existentes no programa aplicativo. Assim, para comutar entre blocos RS232-RX no programa é necessário resetar todos os demais blocos deste tipo existentes no programa para µDX200.*

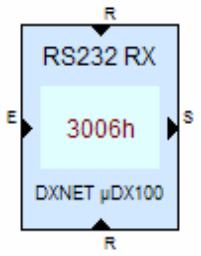
Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial*

principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.

Veja também:
RS232 - RX N Bytes
RS232 - RX Stop Byte
RS232 - RX Start String + N Bytes

RS232 - RX DXNET µDX100

Este bloco permite ao µDX200 receber comandos via porta serial utilizando o protocolo da linha de controladores µDX100 (protocolo DXNET).



Na verdade, apenas o comando 5 (ler variável) e o comando 7 (escrever em variável) foram implementados. Sempre que este bloco for ativado no programa aplicativo o µDX200 pode responder a estes dois comandos vindos pela porta serial. Os formatos dos comandos são os seguintes:

Comando 5 - Lê variável

Dispositivo serial → µDX200

Byte 1	=	1	1	1	1	Z3	Z2	Z1	Z0	Start+Cj
Byte 2	=	0	1	0	1	E3	E2	E1	E0	Cmd+DX
Byte 3	=	V7	V6	V5	V4	V3	V2	V1	V0	Var
Byte 4	=	B7	B6	B5	B4	B3	B2	B1	B0	FCS

µDX200 → Dispositivo serial

Byte 1	=	X7	X6	X5	X4	X3	X2	X1	X0	Valor
Byte 2	=	B7	B6	B5	B4	B3	B2	B1	B0	FCS

Comando 7 - Força variável

Dispositivo serial → µDX

Byte 1	=	1	1	1	1	Z3	Z2	Z1	Z0	Start+Cj
Byte 2	=	0	1	1	1	E3	E2	E1	E0	Cmd+DX
Byte 3	=	V7	V6	V5	V4	V3	V2	V1	V0	Var
Byte 4	=	D7	D6	D5	D4	D3	D2	D1	D0	Valor
Byte 5	=	B7	B6	B5	B4	B3	B2	B1	B0	FCS

µDX → Dispositivo serial

Byte 1	=	0	0	0	0	0	1	1	0	Ack
Byte 2	=	B7	B6	B5	B4	B3	B2	B1	B0	FCS

Sendo:

Z3Z2Z1Z0	→	Endereço de conjunto do µDX200.
E3E2E1E0	→	Endereço DXNET do µDX200.
V7V6V5V4V3V2V1V0	→	Número da variável.
B7B6B5B4B3B2B1B0	→	Checksum dos bytes.
X7X6X5X4X3X2X1X0	→	Valor da variável lida.
D7D6D5D4D3D2D1D0	→	Valor a ser escrito na variável.

Maiores informações sobre o protocolo do Controlador µDX100 pode ser obtido no documento "Comunicação Serial para µDX Plus", disponível no site da Dexter - www.dexter.ind.br.

Alguns esclarecimentos, entretanto, são necessários. O checksum é o valor que somado aos bytes da mensagem resulta em 00h (desprezando o carry, ou seja, pode resultar em 100h, 200h, 300h, etc.). Neste cálculo é desprezado o nibble superior (Start - Fh) do primeiro byte de comando.

Como o µDX200 possui 16 bits de endereçamento e o protocolo do µDX100 prevê apenas 8 bits para isso (4 bits de conjunto e 4 bits de endereço DXNET), o protocolo utiliza apenas os 8 bits LSB do endereçamento do µDX200 para este bloco.

Além disso, as variáveis do µDX200 são em 16 bits, enquanto as variáveis do µDX100 são 8 bits. Então, para ler a variável v0 (entrada analógica E1) do µDX200, por exemplo, é necessário ler a variável v0 (byte LSB) e v1 (byte MSB) via protocolo do µDX100. Para ler a variável v1 (entrada analógica E2) do µDX200 é necessário ler v2 (byte LSB) e v3(byte MSB) via protocolo do µDX100, e assim por diante.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, aguardando a detecção de um dos comandos DXNET µDX100 especificados. Uma vez detectado o comando o bloco o executa e o bloco é desativado até detectar novamente nível alto no nodo de Entrada (E). O bloco pode ser inibido pela energização dos nodos de Reset (R). Quando for executado um comando de µDX100 será gerado um pulso no nodo de Saída (S) do bloco. Caso o nodo de Entrada (E) permaneça ligado o bloco é novamente acionado, passando a aguardar mais uma vez comandos DXNET µDX100.

Note que Configuração pode ser uma variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). O operando Configuração permite especificar os parâmetros de comunicação a serem adotados pela porta serial do µDX200 quando o bloco de Recebe DXNET µDX100 for ativado. Estes parâmetros são os seguintes:

Configuração de porta serial

Permite configurar a porta serial (RS232) de acordo com a necessidade do bloco de RX-DXNET µDX100. O byte baixo (bits 7-0) indicam a velocidade e o byte alto (bits 15-8) indicam os detalhes da configuração conforme indicado:

Bit 15 – 0 = No parity, 1 = Parity enabled
Bit 14 – 0 = odd, 1 = even
Bit 13 – 0 = 1 stop bit, 1 = 2 stop bits
Bit 12 – 0 = 7bits , 1 = 8 bits (opção inibida, esse bit deve estar sempre ligado - 8 bits)
Bit 11 – 0 = loopback off, 1 = loopback on (opção inibida, esse bit deve estar sempre desligado - loopback off)
Bit 10 – 0 = bit deve estar desligado
Bit 9 – 0 = bit deve estar desligado
Bit 8 – 0 = bit deve estar desligado

A velocidade indicada no byte low segue a seguinte tabela:

Valor	bps	Valor	bps	Valor	bps
0	110	1	300	2	600
3	1200	4	2400	5	4800
6	9600	7	19200	8	38400
9	57600	A	115200	B	250000

Normalmente no modo programação esta porta está configurada como 38400, N, 8, 2 (Configuração = 3008h).

Abaixo algumas configurações usuais:

Sem paridade, 8 bits, 1 stop bit.

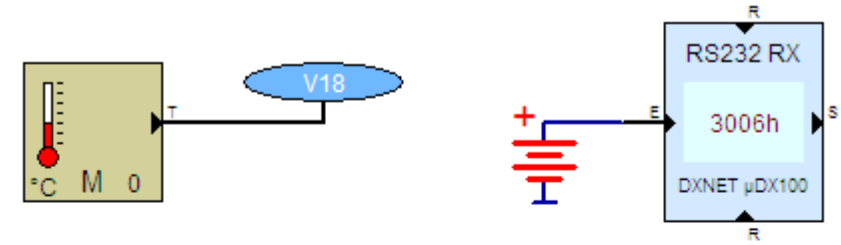
110,n,8,1 → 1000h
300,n,8,1 → 1001h
600,n,8,1 → 1002h
1200,n,8,1 → 1003h
2400,n,8,1 → 1004h
4800,n,8,1 → 1005h
9600,n,8,1 → 1006h
19200,n,8,1 → 1007h
38400,n,8,1 → 1008h
57600,n,8,1 → 1009h
115200,n,8,1 → 100Ah
250000,n,8,1 → 100Bh

Sem paridade, 8 bits, 2 stop bits.

110,n,8,2 → 3000h
300,n,8,2 → 3001h
600,n,8,2 → 3002h
1200,n,8,2 → 3003h
2400,n,8,2 → 3004h
4800,n,8,2 → 3005h
9600,n,8,2 → 3006h
19200,n,8,2 → 3007h
38400,n,8,2 → 3008h
57600,n,8,2 → 3009h
115200,n,8,2 → 300Ah
250000,n,8,2 → 300Bh

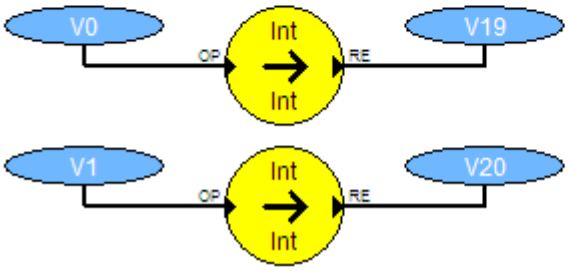
Atenção: *é possível inibir a Configuração de porta RS232 do bloco simplesmente utilizando o valor* ***0000h*** *ou* ***0FFFFh*** *(valores inválidos). Com isso o bloco não irá alterar os parâmetros de porta serial RS232, mantendo os definidos na Configuração de Hardware ou em outros blocos em que existam esse parâmetro.*

O exemplo a seguir utiliza um µDX200 para leitura de duas entradas analógicas (E1 e E2), e de um sensor de temperatura. Os dados são transferidos para as variáveis absolutas v18, v19 e v20 do µDX200, e disponibilizadas para leitura via supervisório Elipse (utilizando o protocolo DXNET do µDX100). Note que para leitura de v18 devemos ler v36 e v37 no protocolo DXNET µDX100. Já para ler v19 do µDX200 devemos ler v38 e v39 no protocolo DXNET µDX100, e assim por diante.



Leitura de temperatura
(resolução de 0,5°)

Protocolo µDX100 em 9600bps,
sem paridade, 8 bits, 2 stop bits.



Tranferência dos valores de
E1 e E2 (entradas analógicas)

Exemplo de Programa Aplicativo: **RS232 - RX DXNET µDX100.dxg**

Note que no bloco de RX DXNET µDX100 foi usado o operando Configuração com o valor 1001h, o que corresponde a 300bps, 8 bits, 1 stop, sem paridade.

Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

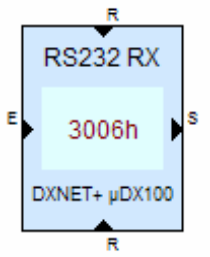
Atenção: *Os blocos RS232-RX são ativados pela detecção de nível alto no nodo de entrada (E) do bloco. Uma vez ativados só são desativados pela recepção via serial do string programado, ou energizando o nodo de reset (R) do bloco. E um bloco de RS232-RX ativo irá bloquear todos os demais blocos RS232-RX existentes no programa aplicativo. Assim, para comutar entre blocos RS232-RX no programa é necessário resetar todos os demais blocos deste tipo existentes no programa para µDX200.*

Atenção: *o protocolo DXNET µDX100 somente permite a leitura de dados do controlador µDX200 conectado via porta serial ao equipamento que utiliza este protocolo. Não é possível ler dados de outros µDX200s conectados via rede DXNET.*

Veja também:
RS232 - RX DXNET+ µDX100
RS232 - RX MODBUS

RS232 - RX DXNET+ µDX100

Este bloco permite ao µDX200 receber comandos via porta serial utilizando o protocolo da linha de controladores µDX100 (protocolo DXNET+).



Na verdade, apenas o comando 5 (ler variável) e o comando 7 (escrever em variável) foram implementados. Sempre que este bloco for ativado no programa aplicativo o µDX200 pode responder a estes dois comandos vindos pela porta serial. Os formatos dos comandos são os seguintes:

Comando 5 - Lê variável

Dispositivo serial → µDX200

Byte 1	=	1	1	1	1	Z3	Z2	Z1	Z0	Start+Cj
Byte 2	=	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Assinat.
Byte 3	=	0	1	0	1	E3	E2	E1	E0	Cmd+DX
Byte 4	=	V7	V6	V5	V4	V3	V2	V1	V0	Var
Byte 5	=	B7	B6	B5	B4	B3	B2	B1	B0	FCS

µDX200 → Dispositivo serial

Byte 1	=	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Assinat.
Byte 2	=	X7	X6	X5	X4	X3	X2	X1	X0	Valor
Byte 3	=	B7	B6	B5	B4	B3	B2	B1	B0	FCS

Comando 7 - Força variável

Dispositivo serial → µDX

Byte 1	=	1	1	1	1	Z3	Z2	Z1	Z0	Start+Cj
Byte 2	=	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Assinat.
Byte 3	=	V7	V6	V5	V4	V3	V2	V1	V0	Var
Byte 4	=	D7	D6	D5	D4	D3	D2	D1	D0	Valor
Byte 5	=	B7	B6	B5	B4	B3	B2	B1	B0	FCS

µDX → Dispositivo serial

Byte 1	=	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Assinat.
Byte 2	=	0	0	0	0	0	1	1	0	Ack
Byte 3	=	B7	B6	B5	B4	B3	B2	B1	B0	FCS

Sendo:

Z3Z2Z1Z0	→	Endereço de conjunto do µDX200.
E3E2E1E0	→	Endereço DXNET do µDX200.
V7V6V5V4V3V2V1V0	→	Número da variável.
B7B6B5B4B3B2B1B0	→	Checksum dos bytes.
X7X6X5X4X3X2X1X0	→	Valor da variável lida.
D7D6D5D4D3D2D1D0	→	Valor a ser escrito na variável.
Y7Y6Y5Y4Y3Y2Y1Y0	→	Assinatura do pacote.

Maiores informações sobre o protocolo do Controlador µDX100 pode ser obtido no documento "Comunicação Serial para µDX Plus", disponível no site da Dexter - www.dexter.ind.br. Note que a única diferença entre o protocolo DXNET e DXNET+ para µDX100 é a presença do byte de assinatura tanto na pergunta quanto na resposta da comunicação. Isso permite identificar que a resposta recebida realmente pertença a determinada pergunta (comparando-se as assinaturas). Assim, a cada nova comunicação o byte de assinatura deve ser modificado (incremental ou randomicamente). O byte de assinatura não entra no cálculo de FCS.

Alguns esclarecimentos, entretanto, são necessários. O checksum é o valor que somado aos bytes da mensagem resulta em 00h (desprezando o carry, ou seja, pode resultar em 100h, 200h, 300h, etc.). Neste cálculo é desprezado o nibble superior (Start - Fh) do primeiro byte de comando.

Como o µDX200 possui 16 bits de endereçamento e o protocolo do µDX100 prevê apenas 8 bits para isso (4 bits de conjunto e 4 bits de endereço DXNET), o protocolo utiliza apenas os 8 bits LSB do endereçamento do µDX200 para este bloco.

Além disso, as variáveis do µDX200 são em 16 bits, enquanto as variáveis do µDX100 são 8 bits. Então, para ler a variável v0 (entrada analógica E1) do µDX200, por exemplo, é necessário ler a variável v0 (byte LSB) e v1 (byte MSB) via protocolo do µDX100. Para ler a variável v1 (entrada analógica E2) do µDX200 é necessário ler v2 (byte LSB) e v3(byte MSB) via protocolo do µDX100, e assim por diante.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, aguardando a detecção de um dos comandos DXNET µDX100 especificados. Uma vez detectado o comando o bloco o executa e o bloco é desativado até detectar novamente nível alto no nodo de Entrada (E). O bloco pode ser inibido pela energização dos nodos de Reset (R). Quando for executado um comando de µDX100 será gerado um pulso no nodo de Saída (S) do bloco. Caso o nodo de Entrada (E) permaneça ligado o bloco é novamente acionado, passando a aguardar mais uma vez comandos DXNET µDX100.

Note que Configuração pode ser uma variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). O operando Configuração permite especificar os parâmetros de comunicação a serem adotados pela porta serial do µDX200 quando o bloco de Recebe DXNET µDX100 for ativado. Estes parâmetros são os seguintes:

Configuração de porta serial

Permite configurar a porta serial (RS232) de acordo com a necessidade do bloco de RX-DXNET+ µDX100. O byte baixo (bits 7-0) indicam a velocidade e o byte alto (bits 15-8) indicam os detalhes da configuração conforme indicado:

Bit 15 – 0 = No parity, 1 = Parity enabled
Bit 14 – 0 = odd, 1 = even
Bit 13 – 0 = 1 stop bit, 1 = 2 stop bits
Bit 12 – 0 = 7bits , 1 = 8 bits (opção inibida, esse bit deve estar sempre ligado - 8 bits)
Bit 11 – 0 = loopback off, 1 = loopback on (opção inibida, esse bit deve estar sempre desligado - loopback off)
Bit 10 – 0 = bit deve estar desligado
Bit 9 – 0 = bit deve estar desligado
Bit 8 – 0 = bit deve estar desligado

A velocidade indicada no byte low segue a seguinte tabela:

Valor	bps	Valor	bps	Valor	bps
0	110	1	300	2	600
3	1200	4	2400	5	4800
6	9600	7	19200	8	38400
9	57600	A	115200	B	250000

Normalmente no modo programação esta porta está configurada como 38400, N, 8, 2 (Configuração = 3008h).

Abaixo algumas configurações usuais:

Sem paridade, 8 bits, 1 stop bit.

110,n,8,1 → 1000h
300,n,8,1 → 1001h
600,n,8,1 → 1002h
1200,n,8,1 → 1003h
2400,n,8,1 → 1004h
4800,n,8,1 → 1005h
9600,n,8,1 → 1006h
19200,n,8,1 → 1007h
38400,n,8,1 → 1008h
57600,n,8,1 → 1009h
115200,n,8,1 → 100Ah
250000,n,8,1 → 100Bh

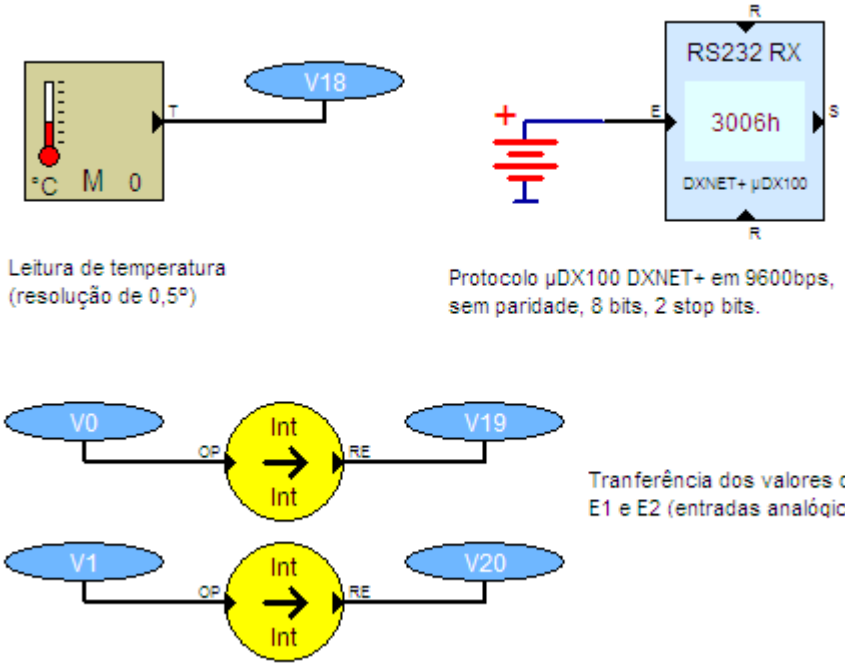
Sem paridade, 8 bits, 2 stop bits.

110,n,8,2 → 3000h
300,n,8,2 → 3001h
600,n,8,2 → 3002h
1200,n,8,2 → 3003h
2400,n,8,2 → 3004h
4800,n,8,2 → 3005h
9600,n,8,2 → 3006h
19200,n,8,2 → 3007h
38400,n,8,2 → 3008h
57600,n,8,2 → 3009h
115200,n,8,2 → 300Ah
250000,n,8,2 → 300Bh

Atenção: *é possível inibir a Configuração de porta RS232 do bloco simplesmente utilizando o valor* ***0000h*** *ou* ***0FFFFh*** *(valores inválidos). Com isso o bloco não irá alterar os parâmetros de porta serial RS232, mantendo os definidos na Configuração de Hardware ou em outros blocos em que existam esse parâmetro.*

O exemplo a seguir utiliza um µDX200 para leitura de duas entradas analógicas (E1 e E2), e de um sensor de temperatura. Os dados são transferidos para as variáveis absolutas v18, v19 e v20 do µDX200, e disponibilizadas para leitura via supervisório Elipse (utilizando o protocolo DXNET+ do µDX100). Note que para leitura de v18 devemos ler v36 e v37 no protocolo DXNET+ µDX100. Já para ler v19 do µDX200 devemos ler v38 e v39 no protocolo DXNET+ µDX100, e assim por

diante.



Exemplo de Programa Aplicativo: **RS232 - RX DXNET+ µDX100.dxg**

Note que no bloco de RX DXNET+ µDX100 foi usado o operando Configuração com o valor 1001h, o que corresponde a 300bps, 8 bits, 1 stop, sem paridade.

> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

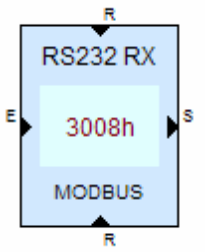
> ***Atenção:*** *Os blocos RS232-RX são ativados pela detecção de nível alto no nodo de entrada (E) do bloco. Uma vez ativados só são desativados pela recepção via serial do string programado, ou energizando o nodo de reset (R) do bloco. E um bloco de RS232-RX ativo irá bloquear todos os demais blocos RS232-RX existentes no programa aplicativo. Assim, para comutar entre blocos RS232-RX no programa é necessário resetar todos os demais blocos deste tipo existentes no programa para µDX200.*

> ***Atenção:*** *o protocolo DXNET+ µDX100 somente permite a leitura de dados do controlador µDX200 conectado via porta serial ao equipamento que utiliza este protocolo. Não é possível ler dados de outros µDX200s conectados via rede DXNET.*

Veja também:
RS232 - RX DXNET µDX100
RS232 - RX MODBUS

RS232 - RX MODBUS

Este bloco permite ao µDX200 receber comandos via porta serial utilizando o protocolo ModBus RTU.



Estão implementados o comando 1 e 2 (ler nodos), comando 3 e 4 (ler variáveis), comando 5 (força nodo), comando 6 (força variável), e o comando 17 (lê status). Sempre que este bloco for ativado no programa aplicativo o µDX200 pode responder a estes comandos ModBus RTU vindos pela porta serial. Os formatos dos comandos são os seguintes:

Comando 1,2 - Lê nodos

Dispositivo serial → µDX200

Byte 1	=	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2	=	0	0	0	0	0	0	0	1	Função 1
Byte 3	=	E15	E14	E13	E12	E11	E10	E9	E8	End. Hi
Byte 4	=	E7	E6	E5	E4	E3	E2	E1	E0	End. Lo
Byte 5	=	N15	N14	N13	N12	N11	N10	N9	N8	Núm. Nod. Hi
Byte 6	=	N7	N6	N5	N4	N3	N2	N1	N0	Núm. Nod. Lo
Byte 7	=	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte 8	=	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

µDX200 → Dispositivo serial

Byte 1	=	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2	=	0	0	0	0	0	0	0	1	Função 1
Byte 3	=	B7	B6	B5	B4	B3	B2	B1	B0	Byte count
Byte 4	=	D7	D6	D5	D4	D3	D2	D1	D0	Dados
. . .										
Byte n	=	D7	D6	D5	D4	D3	D2	D1	D0	Dados
Byte n+1	=	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte n+2	=	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

Note que o primeiro byte é o endereço do µDX200 (o mesmo que foi especificado para a rede DXNET); a seguir vem o comando, o número do primeiro nodo a ser lido (por exemplo, se for N100 deve-se especificar 0064h neste campo); e por fim o número de nodos a serem lidos.

Veja que na resposta do comando ModBus os dados são agrupados de 8 em 8, de forma que se o número de nodos a serem lidos não for múltiplo de 8 os bits inúteis da resposta são preenchidos com zeros.

Por questões de tamanho do buffer de transmissão serial do µDX200, o máximo número de nodos que podem ser lidos em um único comando 01 são 456 nodos (o que resulta em 456/8 = 57 bytes de resposta).

Existe um mapeamento das entradas e saídas digitais do controlador µDX200 nos endereços 32000 e 30000, respectivamente. Assim, para ler a entrada E1 da primeira Expansão µDX210 ligada ao CLP basta ler o nodo 32000. Já para ler a entrada E1 da segunda Expansão µDX210 devemos acessar o endereço 32008. Para as saídas dos µDX210 o processo é similar, ou seja, a saída S1 da primeira Expansão está localizada no endereço 30000; a saída S1 da segunda

Expansão está no endereço 30008; etc.

Todos os nodos referenciados até agora são os nodos_dx, ou seja, os nodos acessíveis via comunicação serial ou via rede DXNET. Este conjunto de nodos se sobreescreve ao conjunto de nodos calculado pelo programa aplicativo (nodos_out). Assim, se um nodo, segundo o programa aplicativo estiver em estado zero, ao ligarmos o nodo_dx correspondente este nodo será ligado, independentemente do programa aplicativo, e só se desligará ao desligarmos o nodo_dx. Já se o nodo estava ligado originalmente devido ao programa aplicativo (nodo_out), ao acionarmos o nodo_dx correspondente este nodo ficará definitivamente em estado alto, pois o nodo_dx em 1 o forçará a este estado.

Ou seja, os nodos_dx permitem acionar nodos do programa aplicativo de forma forçada, independentemente do programa aplicativo. E estes nodos ficam ligados até que sejam zerados os nodos_dx correspondentes.

Entretanto, também é possível acessar diretamente os nodos calculados pelo programa aplicativo (nodos_out). Neste caso, se o nodo estiver desligado ele será acionado apenas por um ciclo de execução do programa aplicativo. Isso é útil para que se efetue apenas um acionamento momentâneo do nodo. A seguir temos a tabela de endereçamento de nodos_dx e nodos_out do µDX200 acessíveis pelo protocolo ModBus RTU.

00000 → nodos_dx	**40000 → nodos_out**
. . .	**. . .**
30000 → saídas_dx	**60000 → saídas_out**
. . .	**. . .**
32000 → entradas_dx	**62000 → entradas_out**

Ou seja, o nodo 30000 aciona o nodo_dx associado a saída S1 do primeiro módulo de Expansão µDX210. Já o nodo 60000 aciona o nodo_out associado a esta mesma saída.

> ***Observação:*** *o comando 2 em Modbus RTU funciona da mesma forma que o comando 1 descrito acima, ou seja, lê nodos do controlador µDX200.*

Comando 3,4 - Lê variáveis

Dispositivo serial → µDX200

Byte 1	=	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2	=	0	0	0	0	0	0	1	1	Função 3
Byte 3	=	E15	E14	E13	E12	E11	E10	E9	E8	End. Hi
Byte 4	=	E7	E6	E5	E4	E3	E2	E1	E0	End. Lo
Byte 5	=	N15	N14	N13	N12	N11	N10	N9	N8	Núm. Var. Hi
Byte 6	=	N7	N6	N5	N4	N3	N2	N1	N0	Núm. Var. Lo
Byte 7	=	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte 8	=	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

µDX200 → Dispositivo serial

Byte 1	=	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2	=	0	0	0	0	0	0	1	1	Função 3
Byte 3	=	B7	B6	B5	B4	B3	B2	B1	B0	Byte count
Byte 4	=	D15	D14	D13	D12	D11	D10	D9	D8	Var. Hi
Byte 5	=	D7	B6	B5	B4	B3	B2	B1	B0	Var. Lo
. . .										
Byte n	=	D15	D14	D13	D12	D11	D10	D9	D8	Var. Hi
Byte n+1	=	D7	D6	D5	D4	D3	D2	D1	D0	Var. Lo

Byte n+2 =	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte n+2 =	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

Note que o primeiro byte é o endereço do µDX200 (o mesmo que foi especificado para a rede DXNET); a seguir vem o comando, o número da primeira variável a ser lida (por exemplo, se for V30 deve-se especificar 001Eh neste campo); e por fim o número de variáveis a serem lidas.

As variáveis são 16 bits e, portanto, cada variável lida ocupa dois bytes na resposta enviada pelo µDX200.

Por questões de tamanho do buffer de transmissão serial do µDX200, o máximo número de variáveis que pode ser lidas em um único comando 03 são 29 variáveis.

> ***Observação:*** *o comando 4 em Modbus RTU funciona da mesma forma que o comando 3 descrito acima, ou seja, lê variáveis do controlador µDX200.*

Comando 5 - Força nodo

Dispositivo serial → µDX200

Byte 1 =	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2 =	0	0	0	0	0	1	0	1	Função 5
Byte 3 =	E15	E14	E13	E12	E11	E10	E9	E8	End. Hi
Byte 4 =	E7	E6	E5	E4	E3	E2	E1	E0	End. Lo
Byte 5 =	F15	F14	F13	F12	F11	F10	F9	F8	Valor Hi
Byte 6 =	F7	F6	F5	F4	F3	F2	F1	F0	Valor Lo
Byte 7 =	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte 8 =	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

µDX200 → Dispositivo serial

Byte 1 =	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2 =	0	0	0	0	0	1	0	1	Função 5
Byte 3 =	E15	E14	E13	E12	E11	E10	E9	E8	End. Hi
Byte 4 =	E7	E6	E5	E4	E3	E2	E1	E0	End. Lo
Byte 5 =	F15	F14	F13	F12	F11	F10	F9	F8	Valor Hi
Byte 6 =	F7	F6	F5	F4	F3	F2	F1	F0	Valor Lo
Byte 7 =	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte 8 =	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

Para forçar o nodo especificado para o valor um (ligado) deve-se usar o campo Valor = FF00h. Já para forçar o nodo para o valor zero (desligado) deve-se usar o campo Valor = 0000h. Por exemplo, para ligar o nodo N56 deve-se remeter o seguinte comando para o µDX200 no endereço DXNET 1:

01h 05h 00h 38h FFh 00h CRC

Existe um mapeamento das entradas e saídas digitais do controlador µDX200 nos endereços 32000 e 30000, respectivamente. Assim, para escrever na entrada E1 da primeira Expansão µDX210 ligada ao CLP basta escrever no nodo 32000. Já para escrever na entrada E1 da segunda Expansão µDX210 devemos acessar o endereço 32008. Para as saídas dos µDX210 o processo é similar, ou seja, a saída S1 da primeira Expansão está localizada no endereço 30000; a saída S1 da segunda Expansão está no endereço 30008; etc.

Comando 6 - Força variável

Dispositivo serial → µDX200

Byte 1 =	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2 =	0	0	0	0	0	1	1	0	Função 6
Byte 3 =	E15	E14	E13	E12	E11	E10	E9	E8	End. Hi
Byte 4 =	E7	E6	E5	E4	E3	E2	E1	E0	End. Lo
Byte 5 =	F15	F14	F13	F12	F11	F10	F9	F8	Valor Hi
Byte 6 =	F7	F6	F5	F4	F3	F2	F1	F0	Valor Lo
Byte 7 =	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte 8 =	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

µDX200 → Dispositivo serial

Byte 1 =	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2 =	0	0	0	0	0	1	1	0	Função 6
Byte 3 =	E15	E14	E13	E12	E11	E10	E9	E8	End. Hi
Byte 4 =	E7	E6	E5	E4	E3	E2	E1	E0	End. Lo
Byte 5 =	F15	F14	F13	F12	F11	F10	F9	F8	Valor Hi
Byte 6 =	F7	F6	F5	F4	F3	F2	F1	F0	Valor Lo
Byte 7 =	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte 8 =	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

Por exemplo, para forçar a variável V50 para o valor 1000 deve-se remeter o seguinte comando para o µDX200 no endereço DXNET 1:

01h 06h 00h 32h 03h E8h CRC

Comando 17 - Status

Dispositivo serial → µDX200

Byte 1 =	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2 =	0	0	0	1	0	0	0	1	Função 17
Byte 7 =	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte 8 =	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

µDX200 → Dispositivo serial

Byte 1 =	S7	S6	S5	S4	S3	S2	S1	S0	Endereço
Byte 2 =	0	0	0	1	0	0	0	1	Função 17
Byte 3 =	0	0	0	1	1	1	0	1	Byte count
Byte 4 =	F15	F14	F13	F12	F11	F10	F9	F8	Equip. Hi
Byte 5 =	F7	F6	F5	F4	F3	F2	F1	F0	Equip. Lo
Byte 6 =	1	1	1	1	1	1	1	1	Run/stop
Byte 7 =	S7	S6	S5	S4	S3	S2	S1	S0	Status
Byte 8 =	V7	V6	V5	V4	V3	V2	V1	V0	Versão
Byte 9 =	W7	W6	W5	W4	W3	W2	W1	W0	Revisão
Byte 10 =	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Sub-rev.
Byte 11 =	D7	D6	D5	D4	D3	D2	D1	D0	Descrição
Byte 12 =	D7	D6	D5	D4	D3	D2	D1	D0	Descrição
Byte 13 =	D7	D6	D5	D4	D3	D2	D1	D0	Descrição
Byte 14 =	D7	D6	D5	D4	D3	D2	D1	D0	Descrição
Byte 15 =	D7	D6	D5	D4	D3	D2	D1	D0	Descrição
Byte 16 =	D7	D6	D5	D4	D3	D2	D1	D0	Descrição
Byte 17 =	D7	D6	D5	D4	D3	D2	D1	D0	Descrição
Byte 18 =	D7	D6	D5	D4	D3	D2	D1	D0	Descrição
Byte 19 =	B15	B14	B13	B12	B11	B10	B9	B8	Bateria Hi

Byte 20 =	B7	B6	B5	B4	B3	B2	B1	B0	Bateria Lo
Byte 21 =	I7	I6	I5	I4	I3	I2	I1	I0	Dia
Byte 22 =	M7	M6	M5	M4	M3	M2	M1	M0	Mês
Byte 23 =	A15	A14	A13	A12	A11	A10	A9	A8	Ano Hi
Byte 24 =	A7	A6	A5	A4	A3	A2	A1	A0	Ano Lo
Byte 25 =	H7	H6	H5	H4	H3	H2	H1	H0	Hora
Byte 26 =	S7	S6	S5	S4	S3	S2	S1	S0	Dia semana
Byte 27 =	G7	G6	G5	G4	G3	G2	G1	G0	Segundo
Byte 28 =	U7	U6	U5	U4	U3	U2	U1	U0	Minuto
Byte 29 =	T15	T14	T13	T12	T11	T10	T9	T8	Temp. Hi
Byte 30 =	T7	T6	T5	T4	T3	T2	T1	T0	Temp. Lo
Byte 31 =	V15	V14	V13	V12	V11	V10	V9	V8	Tensão Hi
Byte 32 =	V7	V6	V5	V4	V3	V2	V1	V0	Tensão Lo
Byte 33 =	C15	C14	C13	C12	C11	C10	C9	C8	CRC Lo
Byte 34 =	C7	C6	C5	C4	C3	C2	C1	C0	CRC Hi

A resposta contempla os seguintes bytes:

Byte 1 → Endereço do µDX200 consultado.
Byte 2 → A própria função 17 é replicada na resposta.
Byte 3 → Número de bytes de resposta (valor fixo de 29 bytes).
Byte 4,5 → Tipo de equipamento (200 = µDX200).
Byte 6 → Indica programa aplicativo em Run ou Stop.
Se este byte retorna FFh o programa está rodando, se retornar 00h está parado.
Note que ele deve sempre retornar FFh, uma vez que com o programa aplicativo parado o µDX200 não responde ao protoclo ModBus RTU.
Byte 7 → Status. Os bits deste byte possuem os seguintes dados:
Bit 0 : Programa aplicativo; Run=1, Stop=0
Bit 1 : CRC do programa no µDX200; NOK=1, OK=0
Bit 2 : Rede DXNET; É o mestre = 1, Não é mestre = 0
Bit 3 : Modo de comunicação; Programação=1, Normal=0
Bit 4 : Cartão MMC; Detectado=1, Ausente=0
Bit 5 : Tamanho do cartão MMC (bits 7,6,5).
Bit 6 : 001=32MB, 010=64MB, 011=128MB
Bit 7 :
Byte 8 → Versão do firmware.
Byte 9 → Revisão do firmware.
Byte 10 → Sub-revisão do firmware.
Byte 11 a 18 → Descrição do programa aplicativo (nome em ASCII).
Byte 19, 20 → Tensão de bateria interna do µDX200 (em Volts).
Vbat = (1,5 * Leitura) / 2048 + 0,1
Byte 21 → Dia conforme relógio de tempo real do µDX200.
Byte 22 → Mês conforme relógio de tempo real do µDX200.
Byte 23, 24 → Ano conforme relógio de tempo real do µDX200.
Byte 25 → Hora conforme relógio de tempo real do µDX200.
Byte 26 → Dia da semana conforme relógio de tempo real do µDX200.
(1=Segunda, 2=Terça, ... , Domingo=7)
Byte 27 → Segundo conforme relógio de tempo real do µDX200.
Byte 28 → Minuto conforme relógio de tempo real do µDX200.
Byte 29, 30 → Temperatura interna do µDX200 (em graus celsius).
Temp = 0,172 * Leitura - 277,75
Byte 31, 32 → Tensão de alimentação do µDX200 (em Volts).
V = 0,00828 * Leitura + 2

Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Os blocos RS232-RX são ativados pela detecção de nível alto no nodo de entrada (E) do bloco. Uma vez ativados só são desativados pela recepção via serial do string programado, ou energizando o nodo de reset (R) do bloco. E um bloco de RS232-RX ativo irá bloquear todos os demais blocos RS232-RX existentes no programa aplicativo. Assim, para comutar entre blocos RS232-RX no programa é necessário resetar todos os demais blocos deste tipo existentes no programa para µDX200.*

Atenção: *o protocolo ModBus somente permite a leitura de dados do controlador µDX200 conectado via porta serial ao equipamento que utiliza este protocolo. Não é possível ler dados de outros µDX200s conectados via rede DXNET.*

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, aguardando a detecção de um dos comandos ModBus RTU especificados. Uma vez detectado o comando o bloco o executa e o bloco é desativado até detectar novamente nível alto no nodo de Entrada (E). O bloco pode ser inibido pela energização dos nodos de Reset (R). Quando for executado um comando de ModBus será gerado um pulso no nodo de Saída (S) do bloco. Caso o nodo de Entrada (E) permaneça ligado o bloco é novamente acionado, passando a aguardar mais uma vez comandos ModBus RTU.

Note que Configuração pode ser uma variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx). O operando Configuração permite especificar os parâmetros de comunicação a serem adotados pela porta serial do µDX200 quando o bloco de Recebe ModBus for ativado. Estes parâmetros são os seguintes:

Configuração de porta serial

Permite configurar a porta serial (RS232) de acordo com a necessidade do bloco de RX-MODBUS. O byte baixo (bits 7-0) indicam a velocidade e o byte alto (bits 15-8) indicam os detalhes da configuração conforme indicado:

Bit 15 – 0 = No parity, 1 = Parity enabled
Bit 14 – 0 = odd, 1 = even
Bit 13 – 0 = 1 stop bit, 1 = 2 stop bits
Bit 12 – 0 = 7bits , 1 = 8 bits (opção inibida, esse bit deve estar sempre ligado - 8 bits)
Bit 11 – 0 = loopback off, 1 = loopback on (opção inibida, esse bit deve estar sempre desligado - loopback off)
Bit 10 – 0 = bit deve estar desligado
Bit 9 – 0 = bit deve estar desligado
Bit 8 – 0 = bit deve estar desligado

A velocidade indicada no byte low segue a seguinte tabela:

Valor	bps	Valor	bps	Valor	bps
0	110	1	300	2	600
3	1200	4	2400	5	4800
6	9600	7	19200	8	38400
9	57600	A	115200	B	250000

Normalmente no modo programação esta porta está configurada como 38400, N, 8, 2 (Configuração = 3008h).

Abaixo algumas configurações usuais:

Sem paridade, 8 bits, 1 stop bit.

110,n,8,1	→	1000h
300,n,8,1	→	1001h
600,n,8,1	→	1002h
1200,n,8,1	→	1003h
2400,n,8,1	→	1004h
4800,n,8,1	→	1005h
9600,n,8,1	→	1006h
19200,n,8,1	→	1007h
38400,n,8,1	→	1008h
57600,n,8,1	→	1009h
115200,n,8,1	→	100Ah
250000,n,8,1	→	100Bh

Sem paridade, 8 bits, 2 stop bits.

110,n,8,2	→	3000h
300,n,8,2	→	3001h
600,n,8,2	→	3002h
1200,n,8,2	→	3003h
2400,n,8,2	→	3004h
4800,n,8,2	→	3005h
9600,n,8,2	→	3006h
19200,n,8,2	→	3007h
38400,n,8,2	→	3008h
57600,n,8,2	→	3009h
115200,n,8,2	→	300Ah
250000,n,8,2	→	300Bh

***Atenção:** é possível inibir a Configuração de porta RS232 do bloco simplesmente utilizando o valor **0000h** ou **0FFFFh** (valores inválidos). Com isso o bloco não irá alterar os parâmetros de porta serial RS232, mantendo os definidos na Configuração de Hardware ou em outros blocos em que existam esse parâmetro.*

Códigos de Erro em ModBus RTU

Caso seja detectado pelo µDX200 erro de CRC, mensagem truncada ou ainda timeout ele simplesmente não responde à comunicação. Mas se for erro de função não-suportada, endereçamento de variável ou nodo fora dos limites, ou ainda quantidade de variáveis excessiva (devido ao limite de buffer serial) ele irá responder da seguinte forma:

Byte 1	→	Endereço do µDX200 consultado.
Byte 2	→	Função requerida com bit MSB ligado (indicando erro).
Byte 3	→	Código do erro.
Byte 4	→	CRC Lo.
Byte 5	→	CRC Hi.

Os códigos de erro são:

01 = Função ilegal (a função usada não é suportada pelo µDX200).

02 = Endereçamento de dados ilegal (endereço de variável ou nodo fora dos limites).

03 = Quantidade de variáveis excessiva (devido ao limite de buffer serial).

Assim, se for usada a função 7, que não está implementada no µDX200, será respondida com a seguinte resposta:

Byte 1 → Endereço do µDX200 consultado.
Byte 2 → 84h (função 4 com bit MSB ligado para indicar erro).
Byte 3 → 01h (código do erro para função não-suportada).
Byte 4 → CRC Lo.
Byte 5 → CRC Hi.

Exemplos de Comunicação ModBus RTU com µDX200

Liga Saída S1
Em todos os exemplos se pressupõe endereço DXNET 1 para o µDX200. Note que a saída S1 da primeira Expansão µDX210 é acessível no endereço de nodo 30000 (em hexa 7530h). Assim, se ligarmos o nodo 30000 será acionada a saída S1 do primeiro módulo de expansão µDX210. Repare que o µDX200 replica o comando 05 (força nodo) na resposta (caso o comando tenha sido executado com sucesso). Para acionar um nodo devemos colocar FF00h no campo de dados. Um último detalhe: o CRC-16 resultante de 01h 05h 75h 30h FFh 00h é 3996h.

PC à µDX200: 01h 05h 75h 30h FFh 00h 96h 39h
µDX200 à PC: 01h 05h 75h 30h FFh 00h 96h 39h

Desliga Saída S1
Note que a saída S1 da primeira Expansão µDX210 é acessível no endereço de nodo 30000 (em hexa 7530h). Assim, se desligarmos o nodo 30000 será desacionada a saída S1 do primeiro módulo de expansão µDX210. Repare que o µDX200 replica o comando 05 (força nodo) na resposta (caso o comando tenha sido executado com sucesso). Para desacionar um nodo devemos colocar 0000h no campo de dados.

PC à µDX200: 01h 05h 75h 30h 00h 00h D7h C9h
µDX200 à PC: 01h 05h 75h 30h 00h 00h D7h C9h

Liga Saída S2
Em todos os exemplos se pressupõe endereço DXNET 1 para o µDX200. Note que a saída S2 da primeira Expansão µDX210 é acessível no endereço de nodo 30001 (em hexa 7531h). Assim, se ligarmos o nodo 30001 será acionada a saída S2 do primeiro módulo de expansão µDX210. Repare que o µDX200 replica o comando 05 (força nodo) na resposta (caso o comando tenha sido executado com sucesso). Para acionar um nodo devemos colocar FF00h no campo de dados. Um último detalhe: o CRC-16 resultante de 01h 05h 75h 31h FFh 00h é F9C7h.

PC à µDX200: 01h 05h 75h 31h FFh 00h C7h F9h
µDX200 à PC: 01h 05h 75h 31h FFh 00h C7h F9h

Desliga Saída S2
Note que a saída S2 da primeira Expansão µDX210 é acessível no endereço de nodo 30001 (em hexa 7531h). Assim, se desligarmos o nodo 30001 será desacionada a saída S2 do primeiro módulo de expansão µDX210. Repare que o µDX200 replica o comando 05 (força nodo) na resposta (caso o comando tenha sido executado com sucesso). Para desacionar um nodo devemos colocar 0000h no campo de dados.

PC à µDX200: 01h 05h 75h 31h 00h 00h 86h 09h
µDX200 à PC: 01h 05h 75h 31h 00h 00h 86h 09h

Lê Variável V18
Novamente se pressupõe endereço DXNET 1 para o µDX200. Note que a variável V18 do µ DX200 é inicializada com um número de série (entre 0 e 65535), de forma a identificar o CLP (inicializei V18 com valor 455 neste exemplo). A pergunta é feita com comando 3, e se questiona uma variável apenas. O CRC-16 resultante de 01h 03h 00h 12h 00h 01h é 0F24h. A resposta do µ DX200 inclui o endereço (01h), o próprio comando (03h), o número de bytes de resposta (02h), e a resposta em si (01C7h, correspondendo a V18=455).

PC à µDX200: 01h 03h 00h 12h 00h 01h 24h 0Fh

µDX200 à PC: 01h 03h 02h 01h C7h F8h 46h

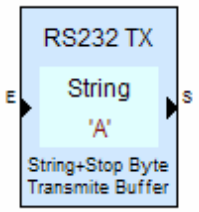
Lê Variável V0 à V7 (entradas analógicas E1 à E8)
O endereço DXNET 1 é ocupado pelo µDX200. Note que as variáveis V0 até V7 correspondem as entradas analógicas do CLP (resolução de 12 bits; portanto variam de 0 a 4095). A pergunta é feita com comando 3, e se questiona 8 variáveis. O CRC-16 resultante de 01h 03h 00h 00h 00h 08h é 0C44h. A resposta do µDX200 inclui o endereço (01h), o próprio comando (03h), o número de bytes de resposta (10h), e a resposta em si (V0=1803, V1=1, V2=1, V3=0, V4=0, V5=0, V6=1, V7=1).

PC à µDX200: 01h 03h 00h 00h 00h 08h 44h 0Ch
µDX200 à PC: 01h 03h 10h 07h 0Bh 00h 01h 00h 01h 00h 00h 00h 00h 00h 00h 00h 01h 00h 01h 7Bh 11h

Veja também:
RS232 - RX DXNET µDX100
RS232 - RX DXNET+ µDX100

RS232 - TX String + Stop Byte

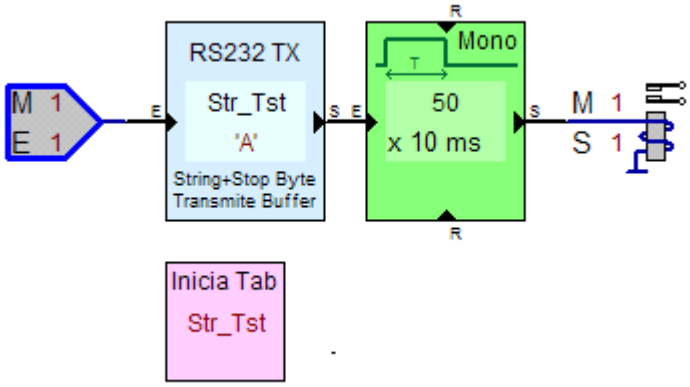
Este bloco permite transmitir um string pela porta serial RS232 do µDX200. O string deve ser finalizado pelo stop byte especificado.



Note que o caracter de finalização do string (StopByte) não é incluído na transmissão serial. Ele é usado para finalizar a transmissão do string e, portanto, deve necessariamente ser incluído no final do string. Este bloco inclui o string especificado no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200.

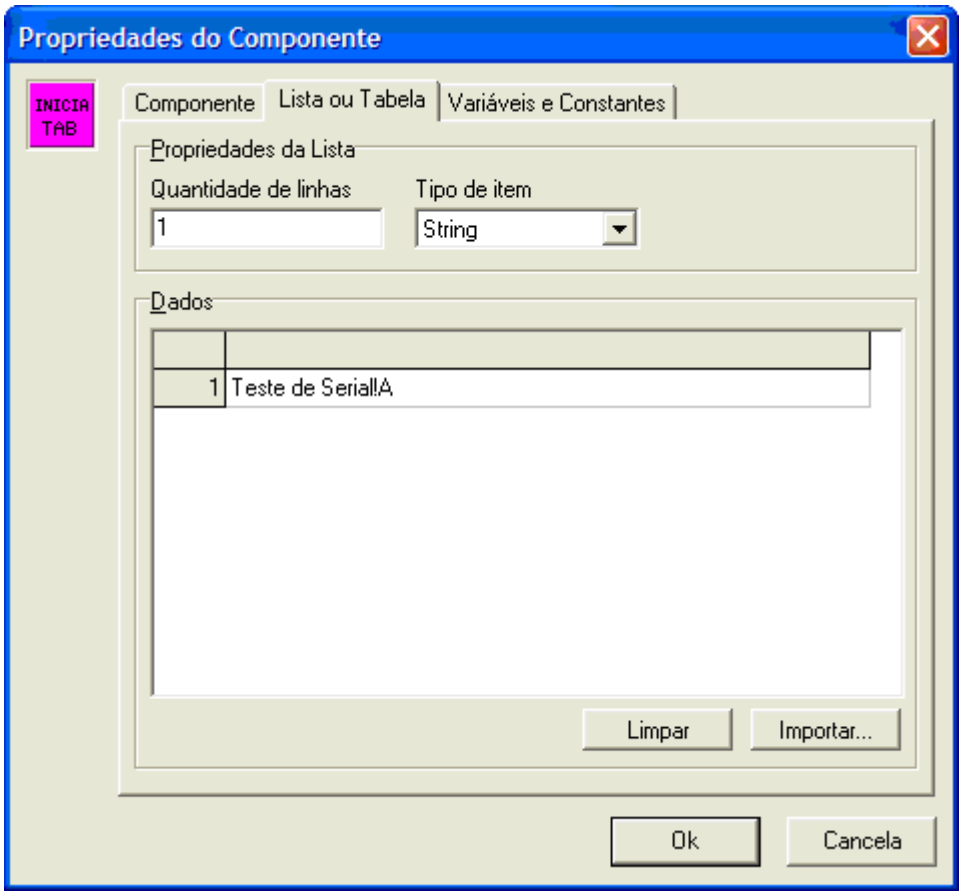
Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 o string especificado com o caracter de finalização (StopByte). Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.

StopByte é uma constante do tipo byte que indica qual o stop byte a ser detectado no string para encerrar a transmissão pela porta serial RS232 do µDX200. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. Assim, podemos colocar, por exemplo, o valor 65 para designar o caracter 'A' a ser detectado como stop byte (já que a representação ASCII de 'A' é 65). Podemos também representar diretamente o caracter ASCII a ser utilizado como stop byte representando-o entre apóstrofes no operando StopByte.



Exemplo de Programa Aplicativo: **RS232 - TX String + Stop Byte.dxg**

O exemplo transmite o string Str_Tst via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. A tabela Str_Tst foi inicializada como string, e finalizada pelo caracter 'A':



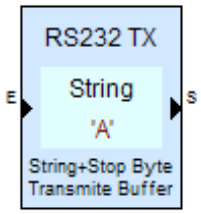
Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial*

principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.

Veja também:
RS232 - TXBuffer String + Stop Byte
RS232 - TX String + Stop Byte Incluso
RS232 - TXBuffer String + Stop Byte Incluso
RS232 - TX String + N Bytes
RS232 - TXBuffer String + N Bytes

RS232 - TXBuffer String + Stop Byte

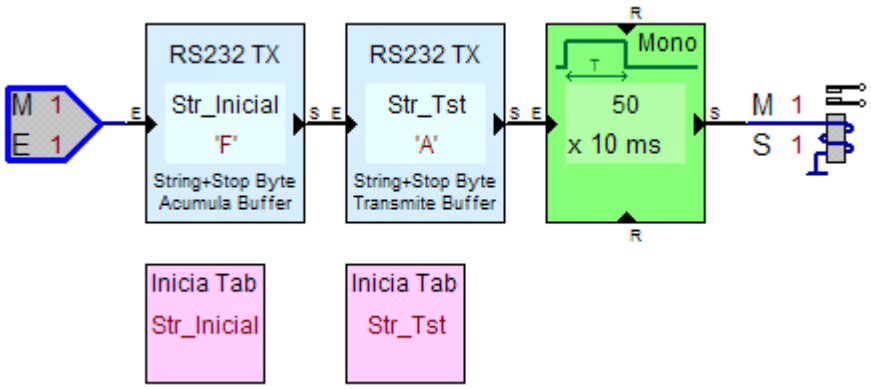
Este bloco permite incluir um string no buffer de transmissão serial RS232 do µDX200. O string deve ser finalizado pelo stop byte especificado.



Note que o caracter de finalização do string (StopByte) não é incluído no buffer de transmissão serial. Ele é usado para finalizar a transmissão do string e, portanto, deve necessariamente ser incluído no final do string. Este bloco inclui o string especificado no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois transmití-la.

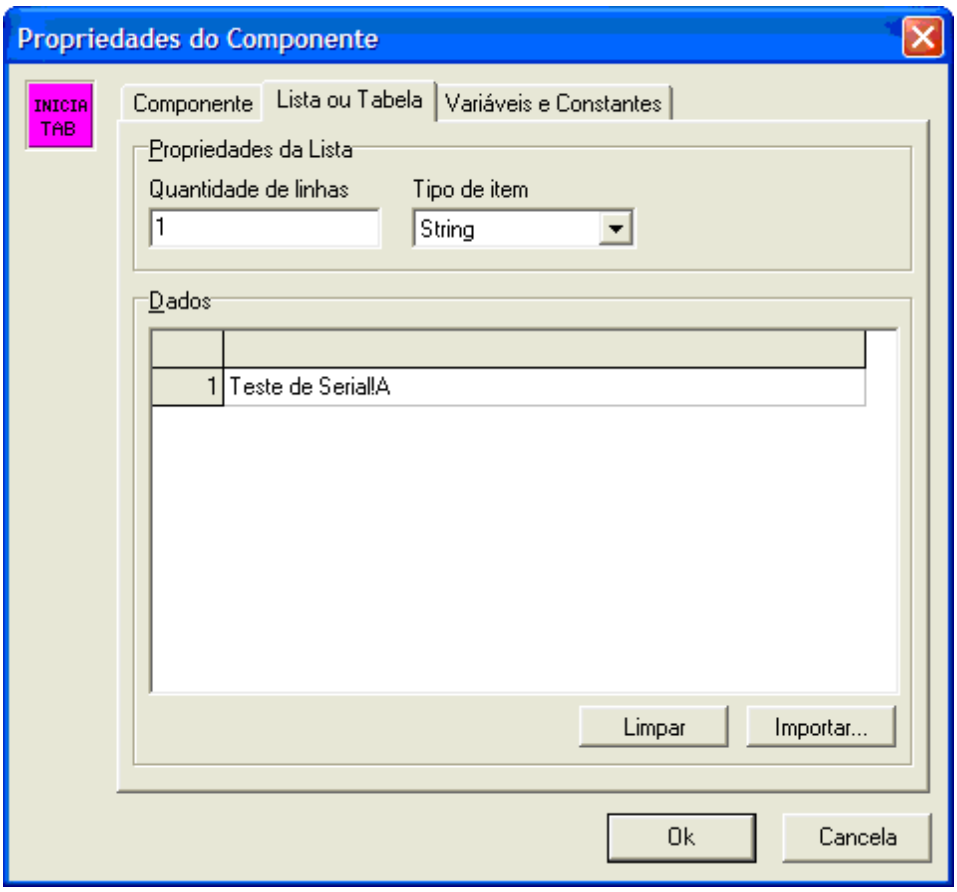
Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até o string especificado ser incluído no buffer de transmissão serial. A seguir será gerado um pulso no nodo de Saída (S) do bloco.

StopByte é uma constante do tipo byte que indica qual o stop byte a ser detectado no string para encerrar a leitura do string e sua inclusão no buffer de transmissão serial. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. Assim, podemos colocar, por exemplo, o valor 65 para designar o caracter 'A' a ser detectado como stop byte (já que a representação ASCII de 'A' é 65). Podemos também representar diretamente o caracter ASCII a ser utilizado como stop byte representando-o entre apóstrofes no operando StopByte.

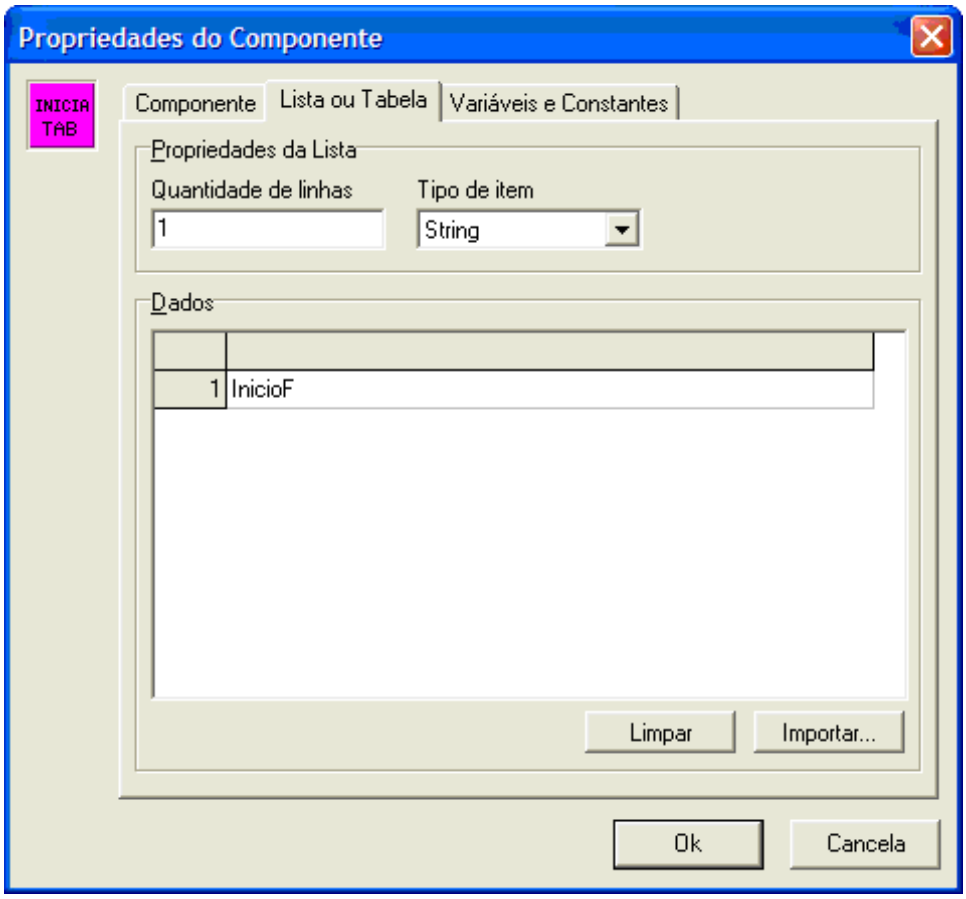


Exemplo de Programa Aplicativo: **RS232 - TXBuffer String + Stop Byte.dxg**

O exemplo transmite o string Str_Inicial + Str_Tst via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. A tabela Str_Tst foi inicializada como string, e finalizada pelo caracter 'A':



Já a tabela Str_Inicial também foi inicializada como string, mas foi finalizada com 'F':



Com isso, é transmitido pela serial o seguinte string: "InicioTeste de Serial!". Note que o primeiro bloco de transmissão serial apenas acumula o string Str_Inicial no buffer. Já o segundo bloco inclui o string Str_Tst no buffer e transmite todo buffer via serial. Note que após a transmissão o buffer é automaticamente zerado.

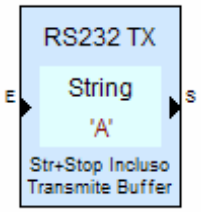
Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX String + Stop Byte

RS232 - TX String + Stop Byte Incluso

Este bloco permite transmitir um string pela porta serial RS232 do µDX200. O string deve ser finalizado pelo stop byte especificado.

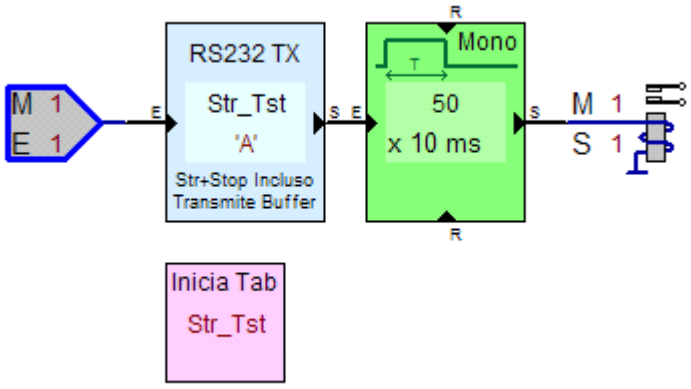


Note que o caracter de finalização do string (StopByte), neste caso, é incluído na transmissão serial. Ele é usado para finalizar a transmissão do string e, portanto, deve necessariamente ser incluído no final do string. Este bloco inclui o string especificado no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 o string especificado com o caracter de finalização (StopByte). Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.

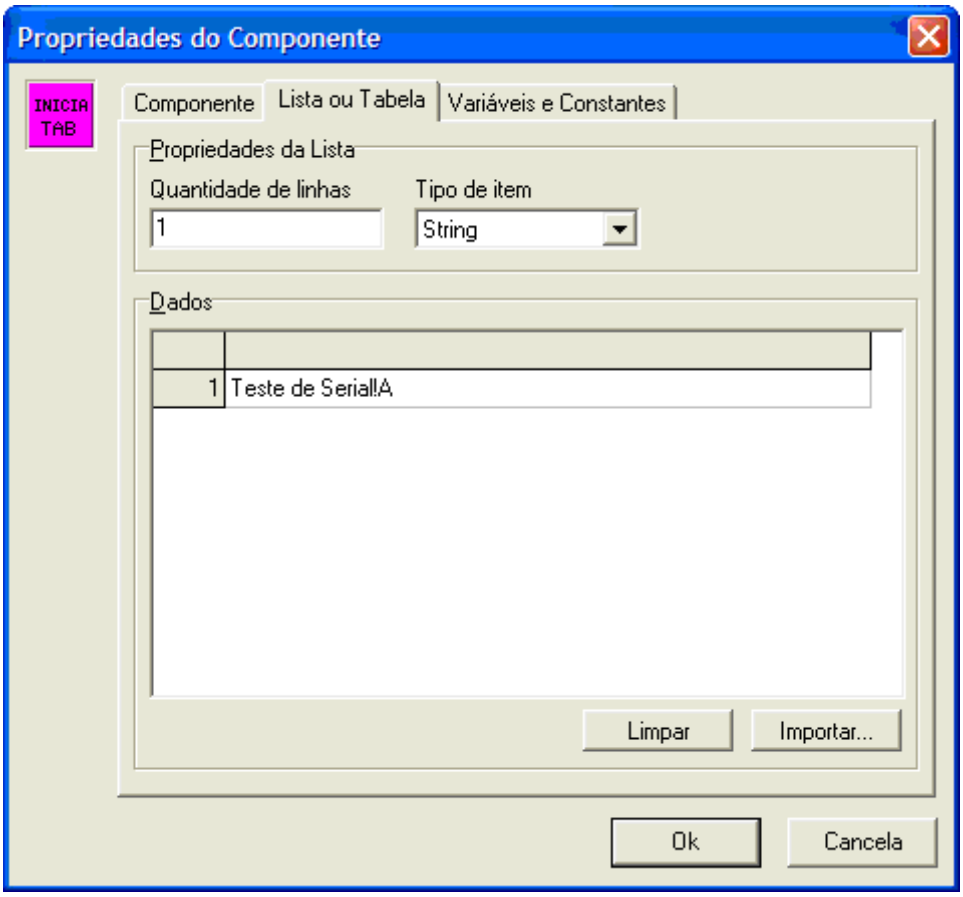
StopByte é uma constante do tipo byte que indica qual o stop byte a ser detectado no string para encerrar a transmissão pela porta serial RS232 do µDX200. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. Assim, podemos colocar, por exemplo, o valor 65 para designar o caracter 'A' a ser detectado como stop byte (já que a representação ASCII de 'A' é 65). Podemos também representar diretamente o caracter ASCII a ser utilizado como stop byte representando-o entre apóstrofes no operando StopByte.

O exemplo transmite o string Str_Tst via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s.



Exemplo de Programa Aplicativo: **RS232 - TX String + Stop Byte Incluso.dxg**

A tabela Str_Tst foi inicializada como string, e finalizada pelo caracter 'A':

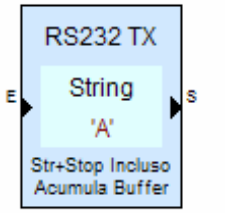


Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX String + Stop Byte
RS232 - TXBuffer String + Stop Byte
RS232 - TXBuffer String + Stop Byte Incluso
RS232 - TX String + N Bytes
RS232 - TXBuffer String + N Bytes

RS232 - TXBuffer String + Stop Byte Incluso

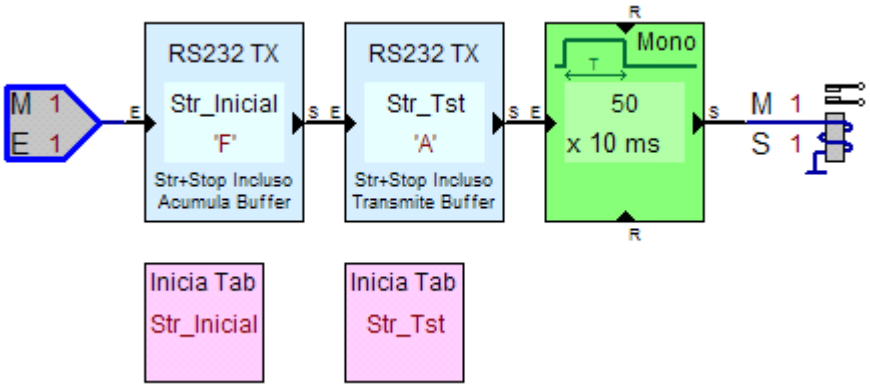
Este bloco permite incluir um string no buffer de transmissão serial RS232 do µDX200. O string deve ser finalizado pelo stop byte especificado.



Note que o caracter de finalização do string (StopByte), neste caso, é incluído no buffer de transmissão serial. Ele é usado para finalizar a transmissão do string e, portanto, deve necessariamente ser incluído no final do string. Este bloco inclui o string especificado no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois transmití-la.

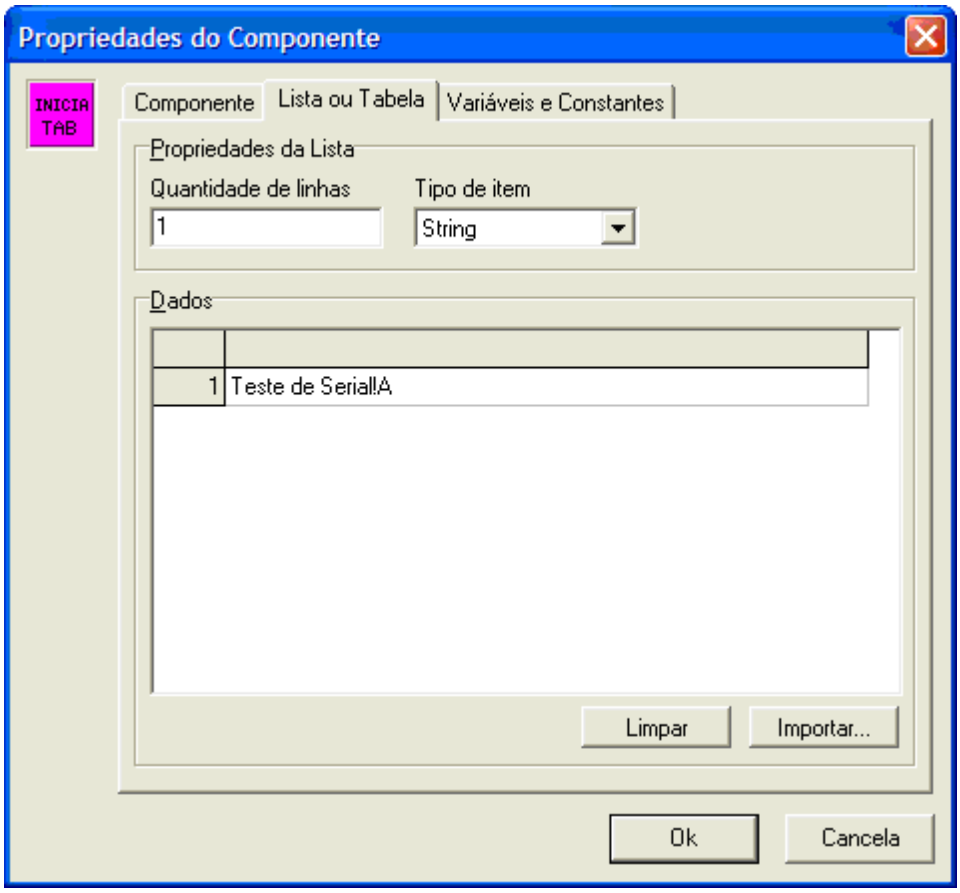
Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até o string especificado ser incluído no buffer de transmissão serial. A seguir será gerado um pulso no nodo de Saída (S) do bloco.

StopByte é uma constante do tipo byte que indica qual o stop byte a ser detectado no string para encerrar a leitura do string e sua inclusão no buffer de transmissão serial. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. Assim, podemos colocar, por exemplo, o valor 65 para designar o caracter 'A' a ser detectado como stop byte (já que a representação ASCII de 'A' é 65). Podemos também representar diretamente o caracter ASCII a ser utilizado como stop byte representando-o entre apóstrofes no operando StopByte.

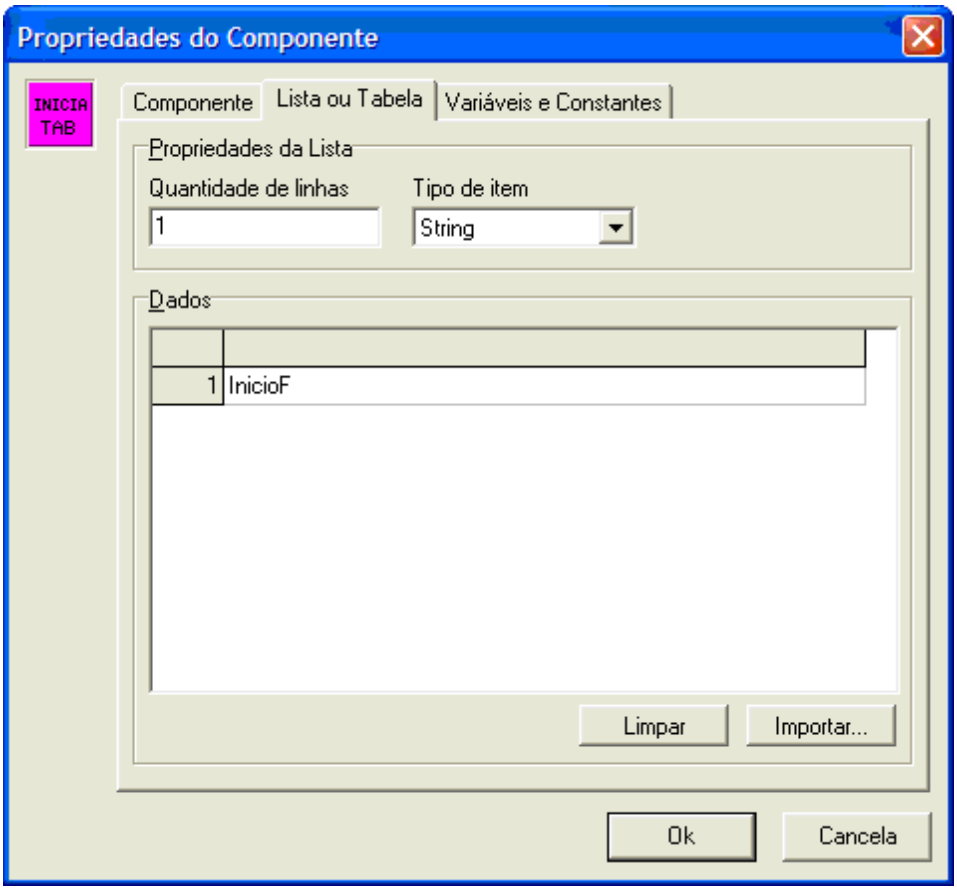


Exemplo de Programa Aplicativo: **RS232 - TXBuffer String + Stop Byte Incluso.dxg**

O exemplo transmite o string Str_Inicial + Str_Tst via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. A tabela Str_Tst foi inicializada como string, e finalizada pelo caracter 'A':



Já a tabela Str_Inicial também foi inicializada como string, mas foi finalizada com 'F':



Com isso, é transmitido pela serial o seguinte string: "InicioFTeste de Serial!A". Note que o primeiro bloco de transmissão serial apenas acumula o string Str_Inicial no buffer. Já o segundo bloco inclui o string Str_Tst no buffer e transmite todo buffer via serial. Note que após a transmissão o buffer é automaticamente zerado. Como foram usados blocos que incluem os caracteres de terminação (StopByte) na transmissão, os caracteres "A" e "F" são transmitidos via serial.

Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

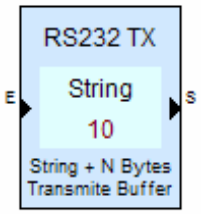
Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX String + Stop Byte
RS232 - TXBuffer String + Stop Byte
RS232 - TX String + Stop Byte Incluso
RS232 - TX String + N Bytes
RS232 - TXBuffer String + N Bytes

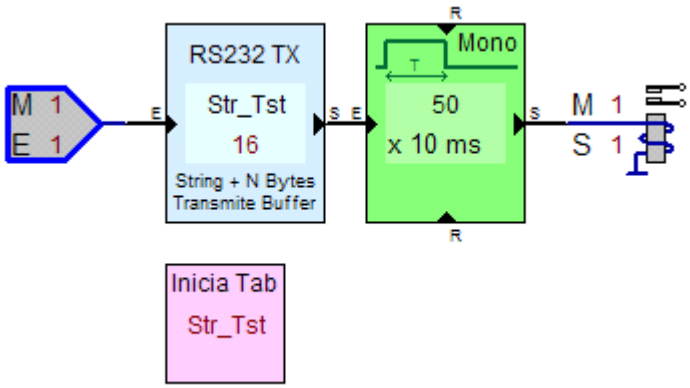
RS232 - TX String + N Bytes

Este bloco permite transmitir um string de tamanho especificado pela porta serial RS232 do µDX200.

Este bloco inclui o string especificado no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200.

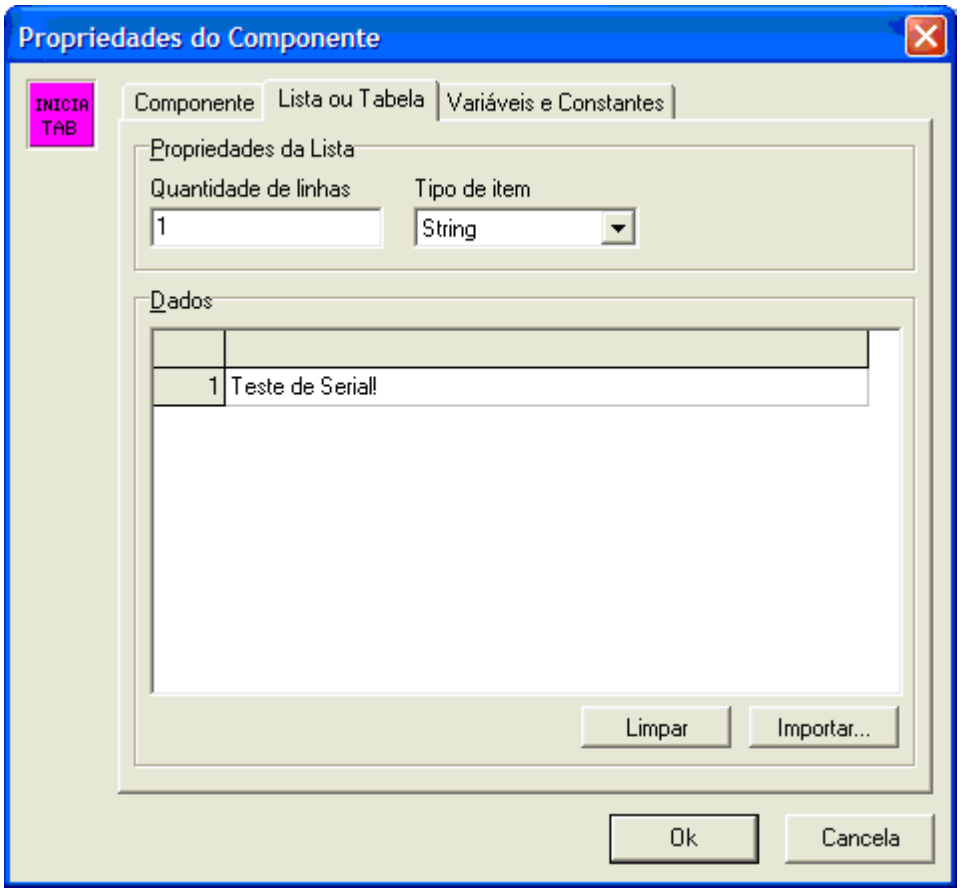


Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 o string especificado com o número de caracteres programado (Núm.Bytes). Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX String + N Bytes.dxg**

O exemplo transmite o string Str_Tst via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. A tabela Str_Tst foi inicializada como string, com o seguinte conteúdo:



Ou seja, a cada energização da entrada digital E1 é transmitida a mensagem de 16 caracteres "Teste de Serial!" pela porta RS232 do µDX200.

Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

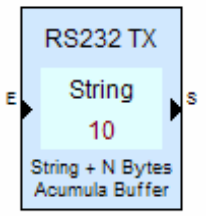
Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX String + Stop Byte
RS232 - TXBuffer String + Stop Byte
RS232 - TX String + Stop Byte Incluso

RS232 - TXBuffer String + Stop Byte Incluso
RS232 - TXBuffer String + N Bytes

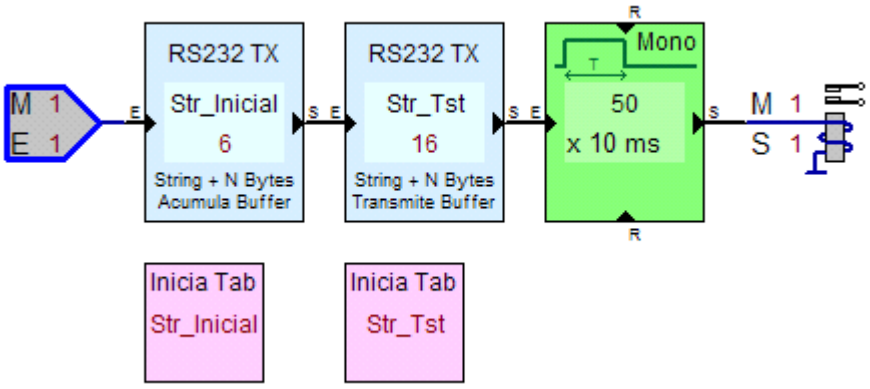
RS232 - TXBuffer String + N Bytes

Este bloco permite incluir um string de tamanho especificado no buffer de porta serial RS232 do µDX200.



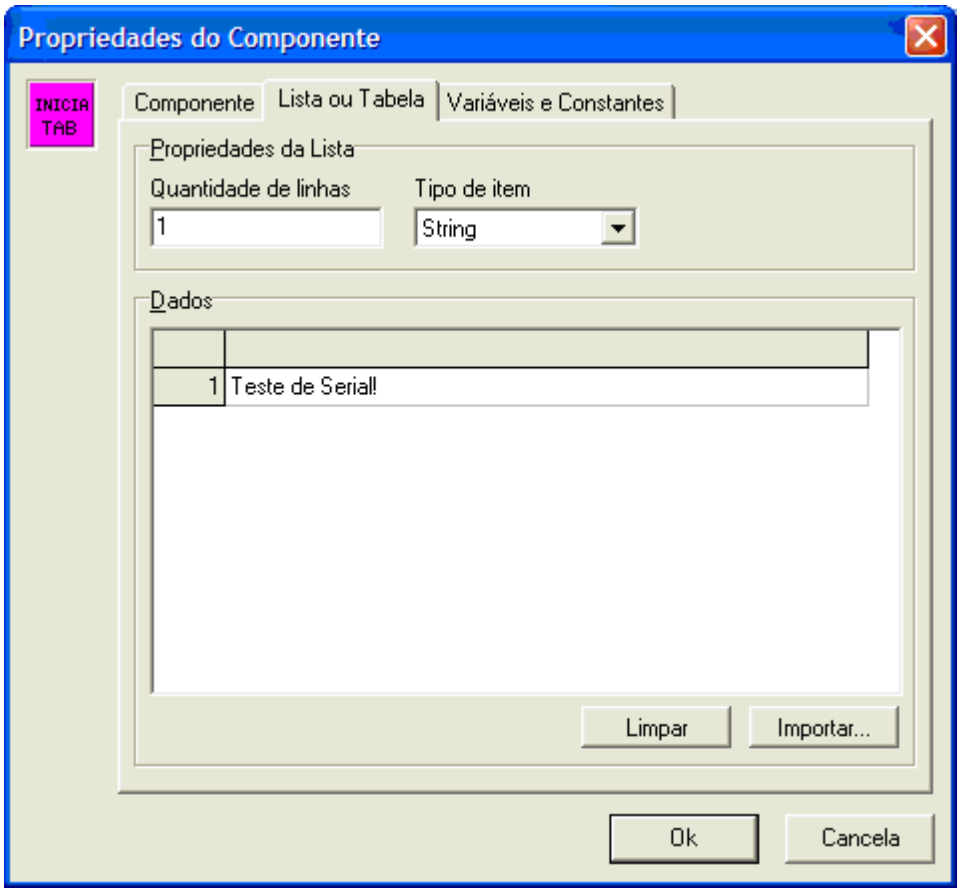
Este bloco inclui o string especificado no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois transmití-la.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transferir para o buffer de serial o string especificado com o número de caracteres programado (Núm.Bytes). Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco.

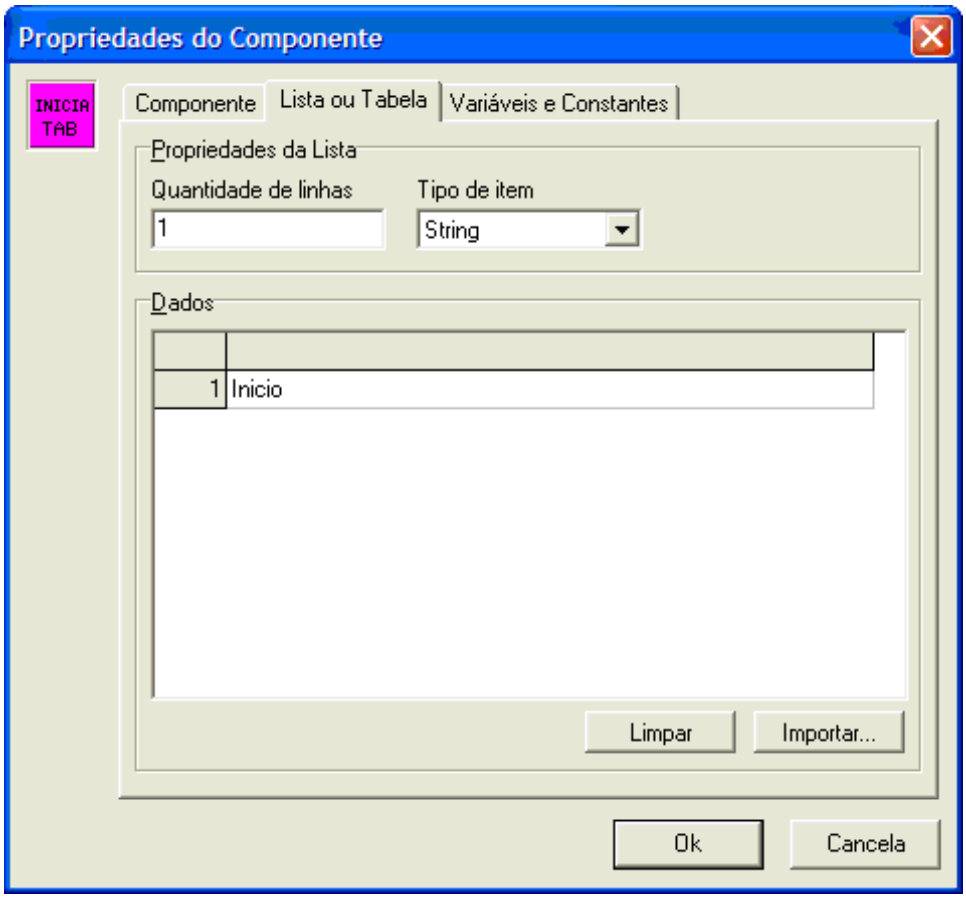


Exemplo de Programa Aplicativo: **RS232 - TXBuffer String + N Bytes.dxg**

O exemplo transmite o string Str_Inicial + Str_Tst via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. A tabela Str_Tst foi inicializada como string, com o seguinte conteúdo:



Já a tabela Str_Inicial também foi inicializada como string com a mensagem a seguir:



Com isso, é transmitido pela serial o seguinte string: "InicioTeste de Serial!". Note que o primeiro bloco de transmissão serial apenas acumula o string Str_Inicial no buffer. Já o segundo bloco inclui o string Str_Tst no buffer e transmite todo buffer via serial. Note que após a transmissão o buffer é automaticamente zerado.

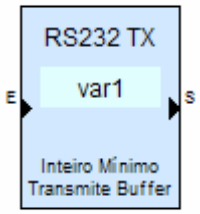
Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX String + Stop Byte

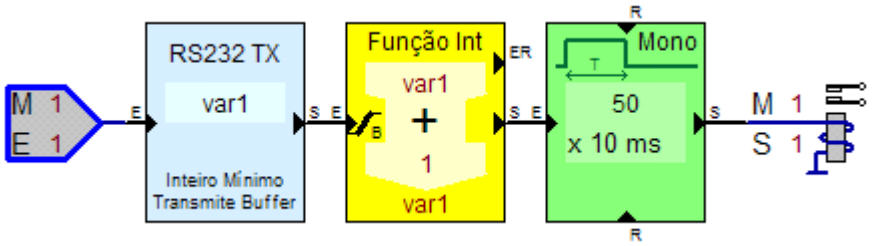
RS232 - TX Inteiro Mínimo

Este bloco permite transmitir uma variável inteira, no formato ASCII, pela porta serial RS232 do µDX200.



Este bloco inclui a variável inteira especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor da variável em ASCII, com o mínimo de caracteres possíveis. Assim, uma variável de valor 25 será representada por apenas dois caracteres ASCII. Já uma variável com valor -30000 será representada por seis caracteres ASCII.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável inteira especificada com o mínimo de caracteres possível. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX Inteiro Mínimo.dxg**

O exemplo transmite o valor da variável var1 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também incrementa a variável var1 a cada transmissão.

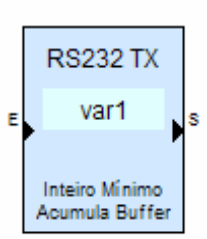
> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TXBuffer Inteiro Mínimo
RS232 - TX N Zeros + Inteiro
RS232 - TXBuffer N Zeros + Inteiro
RS232 - TX N Espaços + Inteiro
RS232 - TXBuffer N Espaços + Inteiro

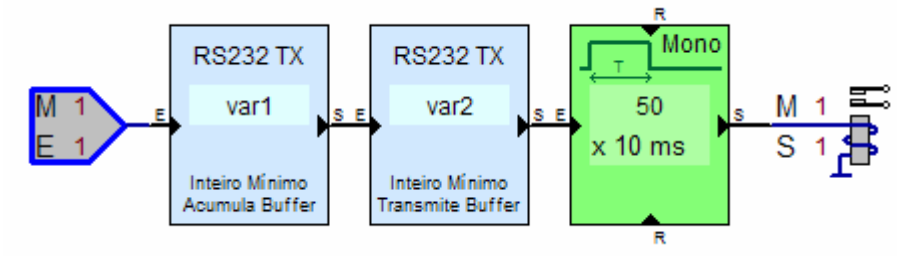
RS232 - TXBuffer Inteiro Mínimo

Este bloco permite incluir a representação do valor de uma variável inteira no buffer de porta serial RS232 do controlador µDX200.



Este bloco inclui a variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir a variável inteira especificada com o mínimo de caracteres possível no buffer de transmissão serial. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TXBuffer Inteiro Mínimo.dxg**

O exemplo transmite o valor da variável var1 e da variável var2 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Note que as duas variáveis são transmitidas em representação ASCII com o mínimo de caracteres possível.

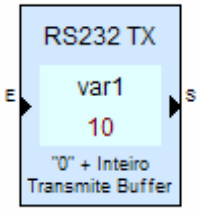
> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX Inteiro Mínimo
RS232 - TX N Zeros + Inteiro
RS232 - TXBuffer N Zeros + Inteiro
RS232 - TX N Espaços + Inteiro
RS232 - TXBuffer N Espaços + Inteiro

RS232 - TX N Zeros + Inteiro

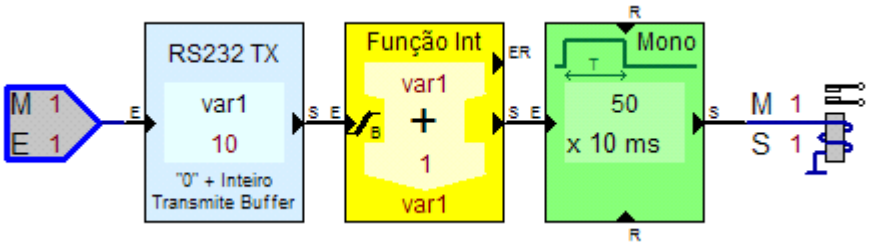
Este bloco permite transmitir uma variável inteira, no formato ASCII, preenchendo o campo com zeros à esquerda, pela porta serial RS232 do µDX200.



Este bloco inclui a variável inteira especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor da variável em ASCII, com o número de caracteres especificado em Núm.Bytes. Os caracteres à esquerda do valor são preenchidos com "0" (zero, 30h em ASCII). Assim, se o bloco especificar 6 caracteres em Núm.Bytes, uma variável de valor 25 será representada por "000025". Já uma variável com valor -30000 será representada por "-30000".

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável inteira especificada com o número de caracteres especificado. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do

bloco.



Exemplo de Programa Aplicativo: **RS232 - TX N Zeros + Inteiro.dxg**

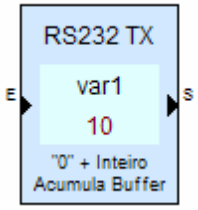
O exemplo transmite o valor da variável var1 com 10 caracteres via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também incrementa a variável var1 a cada transmissão.

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX Inteiro Mínimo
RS232 - TXBuffer Inteiro Mínimo
RS232 - TXBuffer N Zeros + Inteiro
RS232 - TX N Espaços + Inteiro
RS232 - TXBuffer N Espaços + Inteiro

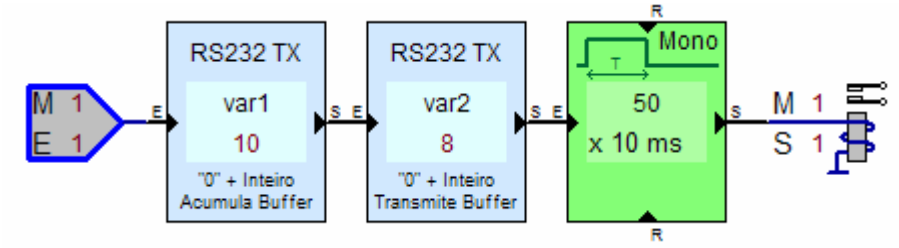
RS232 - TXBuffer N Zeros + Inteiro

Este bloco permite incluir a representação do valor de uma variável inteira, preenchendo o campo com zeros à esquerda, no buffer de porta serial RS232 do controlador µDX200.



Este bloco inclui a variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir a variável inteira especificada com o número de caracteres especificado no buffer de transmissão serial. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TXBuffer N Zeros + Inteiro.dxg**

O exemplo transmite o valor da variável var1, com 10 caracteres, e da variável var2, com 8 caracteres, via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s.

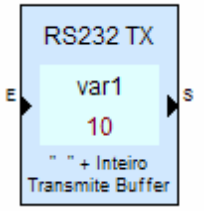
> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX Inteiro Mínimo
RS232 - TXBuffer Inteiro Mínimo
RS232 - TX N Zeros + Inteiro
RS232 - TX N Espaços + Inteiro
RS232 - TXBuffer N Espaços + Inteiro

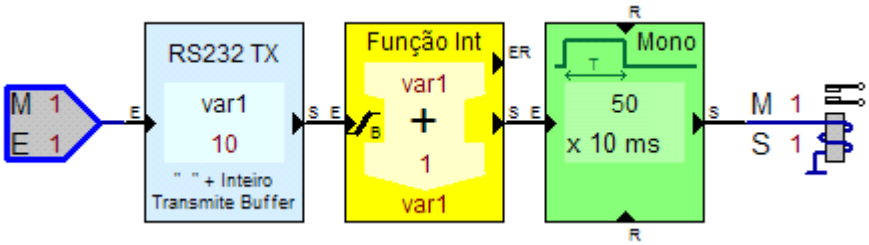
RS232 - TX N Espaços + Inteiro

Este bloco permite transmitir uma variável inteira, no formato ASCII, preenchendo o campo com espaços à esquerda, pela porta serial RS232 do µDX200.



Este bloco inclui a variável inteira especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor da variável em ASCII, com o número de caracteres especificado em Núm.Bytes. Os caracteres à esquerda do valor são preenchidos com " " (espaço, 20h em ASCII). Assim, se o bloco especificar 6 caracteres em Núm.Bytes, uma variável de valor 25 será representada por " 25". Já uma variável com valor -30000 será representada por "-30000".

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável inteira especificada com o número de caracteres especificado. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX N Espaços + Inteiro.dxg**

O exemplo transmite o valor da variável var1 com 10 caracteres via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também incrementa a variável var1 a cada transmissão.

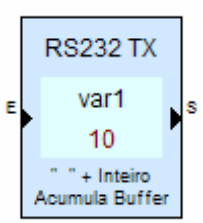
> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX Inteiro Mínimo
RS232 - TXBuffer Inteiro Mínimo
RS232 - TX N Zeros + Inteiro

RS232 - TXBuffer N Zeros + Inteiro
RS232 - TXBuffer N Espaços + Inteiro

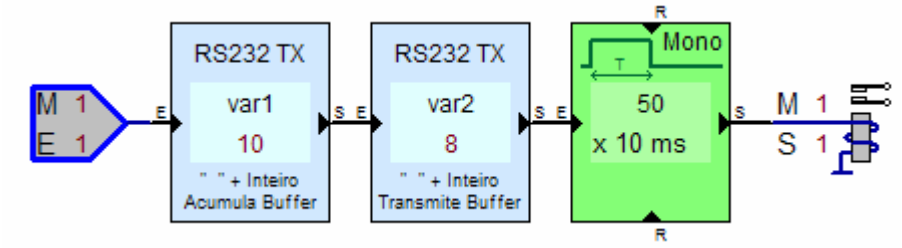
RS232 - TXBuffer N Espaços + Inteiro

Este bloco permite incluir a representação do valor de uma variável inteira, preenchendo o campo com espaços à esquerda, no buffer de porta serial RS232 do controlador µDX200.



Este bloco inclui a variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir a variável inteira especificada com o número de caracteres especificado no buffer de transmissão serial. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TXBuffer N Espaços + Inteiro.dxg**

O exemplo transmite o valor da variável var1, com 10 caracteres, e da variável var2, com 8 caracteres, via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s.

> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

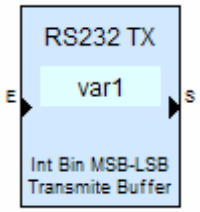
Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX Inteiro Mínimo
RS232 - TXBuffer Inteiro Mínimo
RS232 - TX N Zeros + Inteiro
RS232 - TXBuffer N Zeros + Inteiro
RS232 - TX N Espaços + Inteiro

RS232 - TX Inteiro Binário MSB-LSB

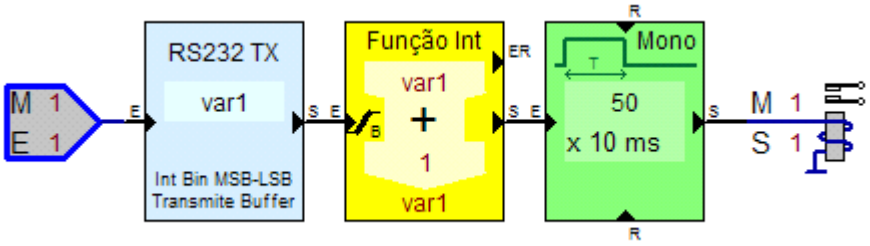
Este bloco permite transmitir uma variável inteira, no formato binário, pela porta serial RS232 do µDX200.

É transmitido primeiro o byte MSB e depois o byte LSB da variável inteira.



Este bloco inclui a variável inteira especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor da variável em binário, em dois bytes, sendo transmitido primeiro o byte mais significativo (MSB) e depois o byte menos significativo (LSB).

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável inteira especificada. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX Inteiro Binário MSB-LSB.dxg**

O exemplo transmite o valor da variável var1 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também incrementa a variável var1 a cada transmissão.

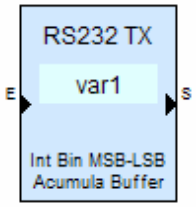
> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TXBuffer Inteiro Binário MSB-LSB
RS232 - TX Inteiro Binário LSB-MSB
RS232 - TXBuffer Inteiro Binário LSB-MSB
RS232 - TX Inteiro Binário MSB
RS232 - TXBuffer Inteiro Binário MSB
RS232 - TX Inteiro Binário LSB
RS232 - TXBuffer Inteiro Binário LSB

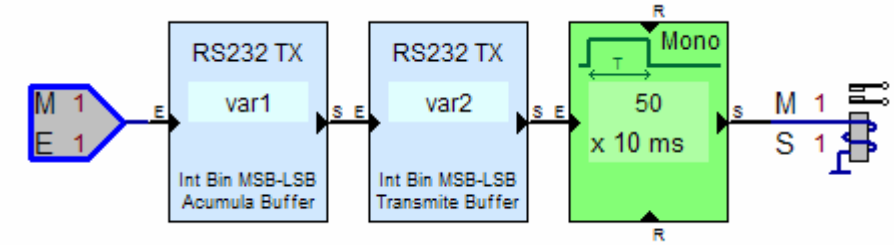
RS232 - TXBuffer Inteiro Binário MSB-LSB

Este bloco permite incluir o valor binário de uma variável inteira no buffer de porta serial RS232 do controlador µDX200. É incluído no buffer primeiro o byte MSB e depois o byte LSB da variável inteira especificada.



Este bloco inclui a variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir a variável inteira especificada no buffer de transmissão serial. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco. Note que é incluída no buffer a representação do valor da variável em binário, em dois bytes, sendo incluído primeiro o byte mais significativo (MSB) e depois o byte menos significativo (LSB).



Exemplo de Programa Aplicativo: **RS232 - TXBuffer Inteiro Binário MSB-LSB.dxg**

O exemplo transmite o valor da variável var1 e da variável var2 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s.

> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial*

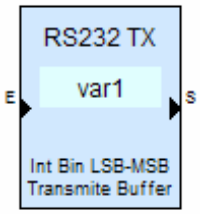
principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.

Veja também:
RS232 - TX Inteiro Binário MSB-LSB
RS232 - TX Inteiro Binário LSB-MSB
RS232 - TXBuffer Inteiro Binário LSB-MSB
RS232 - TX Inteiro Binário MSB
RS232 - TXBuffer Inteiro Binário MSB
RS232 - TX Inteiro Binário LSB
RS232 - TXBuffer Inteiro Binário LSB

RS232 - TX Inteiro Binário LSB-MSB

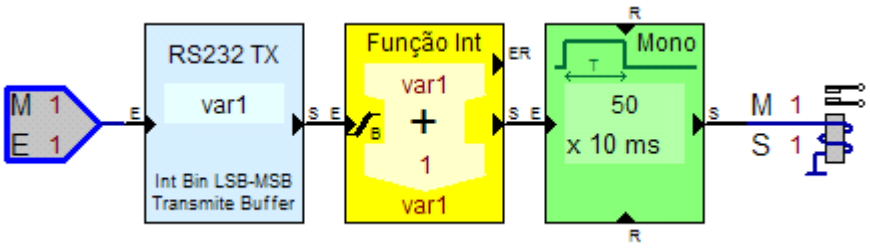
Este bloco permite transmitir uma variável inteira, no formato binário, pela porta serial RS232 do µDX200.

É transmitido primeiro o byte LSB e depois o byte MSB da variável inteira.



Este bloco inclui a variável inteira especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor da variável em binário, em dois bytes, sendo transmitido primeiro o byte menos significativo (LSB) e depois o byte mais significativo (MSB).

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável inteira especificada. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX Inteiro Binário LSB-MSB.dxg**

O exemplo transmite o valor da variável var1 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também incrementa a variável var1 a cada transmissão.

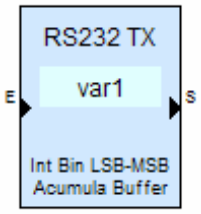
Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX Inteiro Binário MSB-LSB
RS232 - TXBuffer Inteiro Binário MSB-LSB
RS232 - TXBuffer Inteiro Binário LSB-MSB
RS232 - TX Inteiro Binário MSB
RS232 - TXBuffer Inteiro Binário MSB
RS232 - TX Inteiro Binário LSB
RS232 - TXBuffer Inteiro Binário LSB

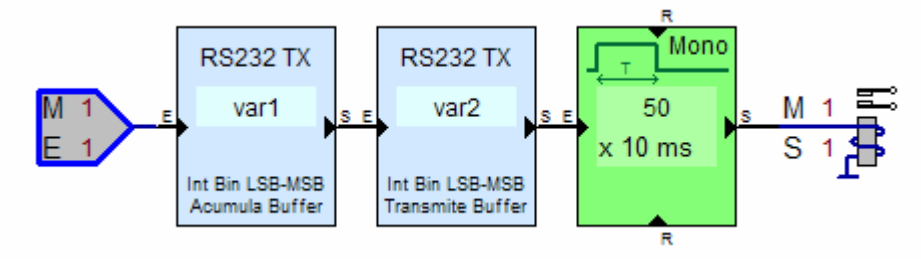
RS232 - TXBuffer Inteiro Binário LSB-MSB

Este bloco permite incluir o valor binário de uma variável inteira no buffer de porta serial RS232 do controlador µDX200. É incluído no buffer primeiro o byte LSB e depois o byte MSB da variável inteira especificada.



Este bloco inclui a variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir a variável inteira especificada no buffer de transmissão serial. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco. Note que é incluída no buffer a representação do valor da variável em binário, em dois bytes, sendo incluído primeiro o byte menos significativo (LSB) e depois o byte mais significativo (MSB).



Exemplo de Programa Aplicativo: **RS232 - TXBuffer Inteiro Binário LSB-MSB.dxg**

O exemplo transmite o valor da variável var1 e da variável var2 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s.

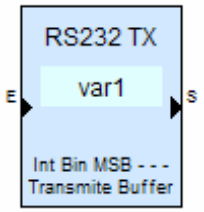
Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX Inteiro Binário MSB-LSB
RS232 - TXBuffer Inteiro Binário MSB-LSB
RS232 - TX Inteiro Binário LSB-MSB
RS232 - TX Inteiro Binário MSB
RS232 - TXBuffer Inteiro Binário MSB
RS232 - TX Inteiro Binário LSB
RS232 - TXBuffer Inteiro Binário LSB

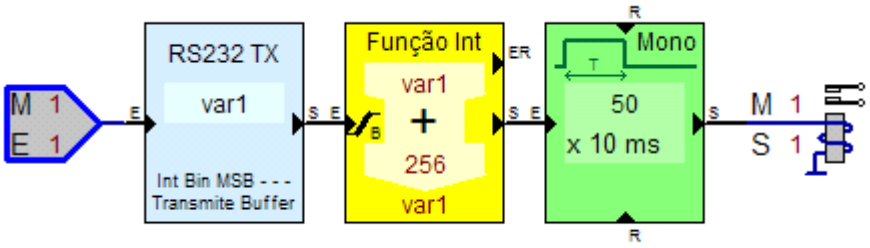
RS232 - TX Inteiro Binário MSB

Este bloco permite transmitir o byte MSB de uma variável inteira, no formato binário, pela porta serial RS232 do µDX200. É transmitido apenas o byte MSB da variável.



Este bloco inclui o byte mais significativo (MSB) da variável inteira especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor do byte MSB da variável em binário.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável inteira especificada. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX Inteiro Binário MSB.dxg**

O exemplo transmite o valor do byte MSB da variável var1 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também soma 256 à variável var1 a cada transmissão, de forma a incrementar o byte MSB da variável.

> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

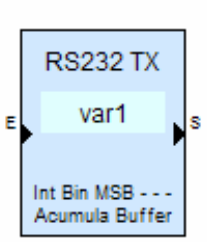
> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é*

preciso habilitá-las via Configuração de Hardware no programa aplicativo.

Veja também:
RS232 - TX Inteiro Binário MSB-LSB
RS232 - TXBuffer Inteiro Binário MSB-LSB
RS232 - TX Inteiro Binário LSB-MSB
RS232 - TXBuffer Inteiro Binário LSB-MSB
RS232 - TXBuffer Inteiro Binário MSB
RS232 - TX Inteiro Binário LSB
RS232 - TXBuffer Inteiro Binário LSB

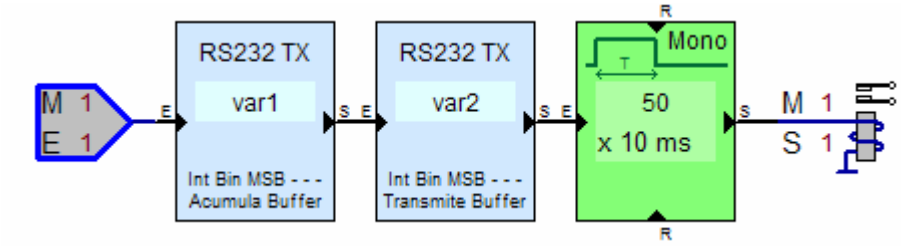
RS232 - TXBuffer Inteiro Binário MSB

Este bloco permite incluir o valor binário do byte MSB de uma variável inteira no buffer de porta serial RS232 do controlador µDX200. É incluído apenas o byte MSB da variável no buffer.



Este bloco inclui o byte mais significativo (MSB) da variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir o byte MSB da variável inteira especificada no buffer de transmissão serial. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco. Note que é incluída no buffer a representação do valor do byte MSB da variável em binário.



Exemplo de Programa Aplicativo: **RS232 - TXBuffer Inteiro Binário MSB.dxg**

O exemplo transmite o valor do bytes MSB das variáveis var1 e var2 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s.

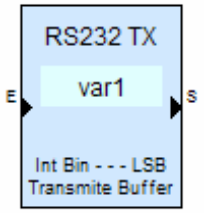
***Atenção:** é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX Inteiro Binário MSB-LSB
RS232 - TXBuffer Inteiro Binário MSB-LSB
RS232 - TX Inteiro Binário LSB-MSB
RS232 - TXBuffer Inteiro Binário LSB-MSB
RS232 - TX Inteiro Binário MSB
RS232 - TX Inteiro Binário LSB
RS232 - TXBuffer Inteiro Binário LSB

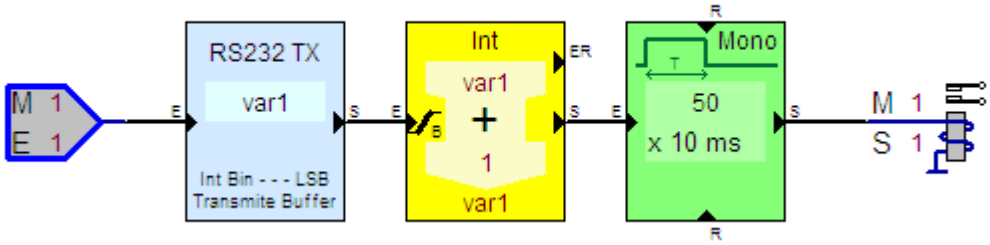
RS232 - TX Inteiro Binário LSB

Este bloco permite transmitir o byte LSB de uma variável inteira, no formato binário, pela porta serial RS232 do µDX200. É transmitido apenas o byte LSB da variável.



Este bloco inclui o byte menos significativo (LSB) da variável inteira especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor do byte LSB da variável em binário.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável inteira especificada. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX Inteiro Binário LSB.dxg**

O exemplo transmite o valor do byte LSB da variável var1 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também incrementa var1 a cada transmissão.

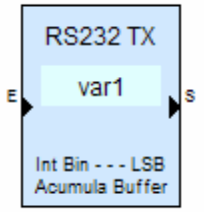
Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX Inteiro Binário MSB-LSB
RS232 - TXBuffer Inteiro Binário MSB-LSB
RS232 - TX Inteiro Binário LSB-MSB
RS232 - TXBuffer Inteiro Binário LSB-MSB
RS232 - TX Inteiro Binário MSB
RS232 - TXBuffer Inteiro Binário MSB
RS232 - TXBuffer Inteiro Binário LSB

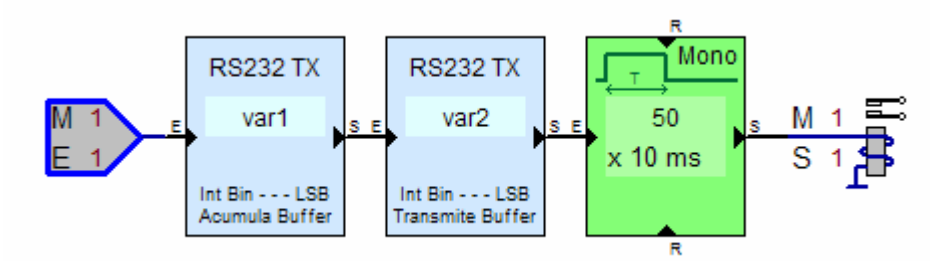
RS232 - TXBuffer Inteiro Binário LSB

Este bloco permite incluir o valor binário do byte LSB de uma variável inteira no buffer de porta serial RS232 do controlador µDX200. É incluído apenas o byte LSB da variável no buffer.



Este bloco inclui o byte menos significativo (LSB) da variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir o byte LSB da variável inteira especificada no buffer de transmissão serial. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco. Note que é incluída no buffer a representação do valor do byte LSB da variável em binário.



Exemplo de Programa Aplicativo: **RS232 - TXBuffer Inteiro Binário LSB.dxg**

O exemplo transmite o valor do bytes LSB das variáveis var1 e var2 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s.

Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

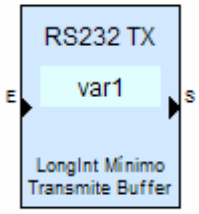
Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam*

protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.

Veja também:
RS232 - TX Inteiro Binário MSB-LSB
RS232 - TXBuffer Inteiro Binário MSB-LSB
RS232 - TX Inteiro Binário LSB-MSB
RS232 - TXBuffer Inteiro Binário LSB-MSB
RS232 - TX Inteiro Binário MSB
RS232 - TXBuffer Inteiro Binário MSB
RS232 - TX Inteiro Binário LSB

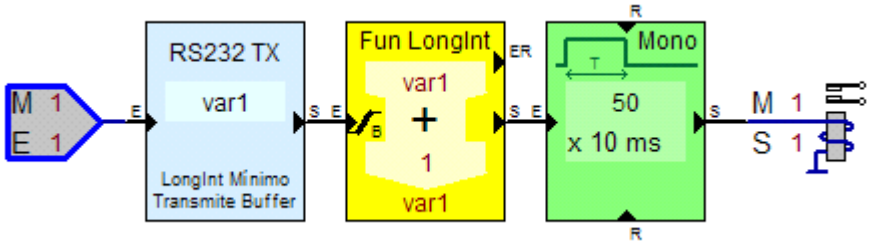
RS232 - TX LongInt Mínimo

Este bloco permite transmitir uma variável longint, no formato ASCII, pela porta serial RS232 do µDX200.



Este bloco inclui a variável longint especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor da variável em ASCII, com o mínimo de caracteres possíveis. Assim, uma variável de valor 25 será representada por apenas dois caracteres ASCII. Já uma variável com valor -300000 será representada por sete caracteres ASCII.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável longint especificada com o mínimo de caracteres possível. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX LongInt Mínimo.dxg**

O exemplo transmite o valor da variável var1 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também incrementa a variável var1 a cada transmissão.

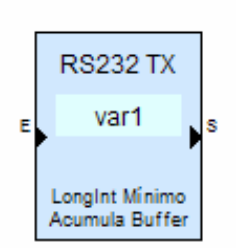
Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TXBuffer LongInt Mínimo
RS232 - TX N Zeros + LongInt
RS232 - TXBuffer N Zeros + LongInt
RS232 - TX N Espaços + LongInt
RS232 - TXBuffer N Espaços + LongInt

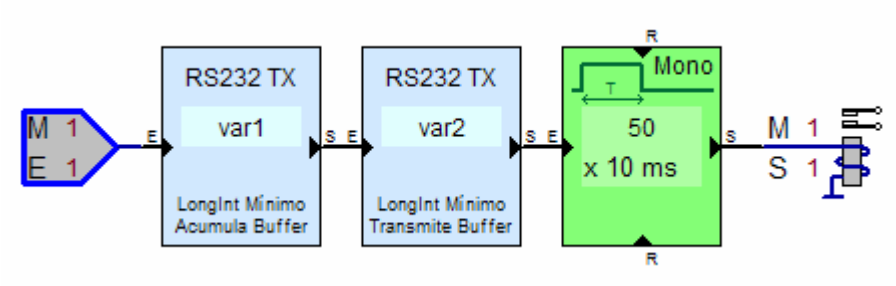
RS232 - TXBuffer LongInt Mínimo

Este bloco permite incluir a representação do valor de uma variável longint no buffer de porta serial RS232 do controlador µDX200.



Este bloco inclui a variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir a variável longint especificada com o mínimo de caracteres possível no buffer de transmissão serial. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TXBuffer LongInt Mínimo.dxg**

O exemplo transmite o valor da variável var1 e da variável var2 via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Note que as duas variáveis são transmitidas em representação ASCII com o mínimo de caracteres possível.

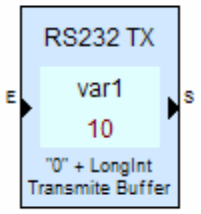
> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX LongInt Mínimo
RS232 - TX N Zeros + LongInt
RS232 - TXBuffer N Zeros + LongInt
RS232 - TX N Espaços + LongInt
RS232 - TXBuffer N Espaços + LongInt

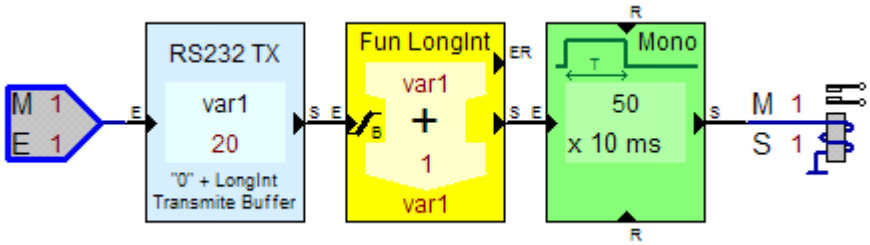
RS232 - TX N Zeros + LongInt

Este bloco permite transmitir uma variável longint, no formato ASCII, preenchendo o campo com zeros à esquerda, pela porta serial RS232 do µDX200.



Este bloco inclui a variável longint especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor da variável em ASCII, com o número de caracteres especificado em Núm.Bytes. Os caracteres à esquerda do valor são preenchidos com "0" (zero, 30h em ASCII). Assim, se o bloco especificar 7 caracteres em Núm.Bytes, uma variável de valor 25 será representada por "0000025". Já uma variável com valor -300000 será representada por "-300000".

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável longint especificada com o número de caracteres especificado. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX N Zeros + LongInt.dxg**

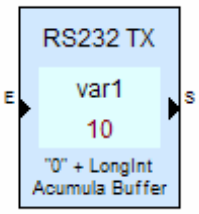
O exemplo transmite o valor da variável var1 com 20 caracteres via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também incrementa a variável var1 a cada transmissão.

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:

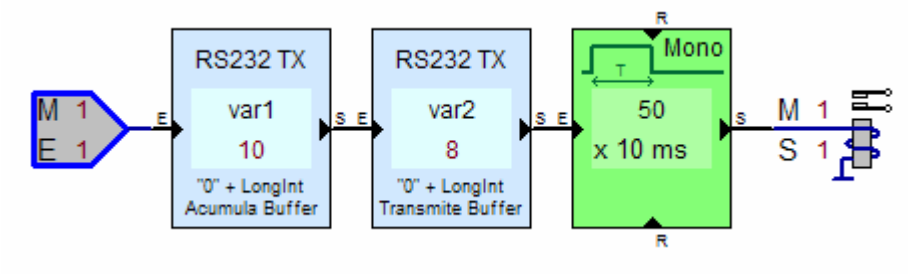
RS232 - TXBuffer N Zeros + LongInt

Este bloco permite incluir a representação do valor de uma variável longint, preenchendo o campo com zeros à esquerda, no buffer de porta serial RS232 do controlador µDX200.



Este bloco inclui a variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir a variável longint especificada com o número de caracteres especificado no buffer de transmissão serial. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TXBuffer N Zeros + LongInt.dxg**

O exemplo transmite o valor da variável var1, com 10 caracteres, e da variável var2, com 8 caracteres, via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s.

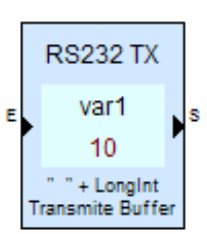
> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX LongInt Mínimo
RS232 - TXBuffer LongInt Mínimo
RS232 - TX N Zeros + LongInt
RS232 - TX N Espaços + LongInt
RS232 - TXBuffer N Espaços + LongInt

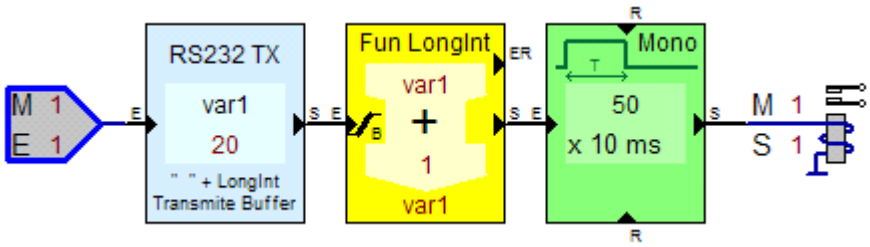
RS232 - TX N Espaços + LongInt

Este bloco permite transmitir uma variável longint, no formato ASCII, preenchendo o campo com espaços à esquerda, pela porta serial RS232 do µDX200.



Este bloco inclui a variável longint especificada no buffer de transmissão serial e imediatamente transmite o buffer pela porta serial do µDX200. Note que é transmitida a representação do valor da variável em ASCII, com o número de caracteres especificado em Núm.Bytes. Os caracteres à esquerda do valor são preenchidos com " " (espaço, 20h em ASCII). Assim, se o bloco especificar 7 caracteres em Núm.Bytes, uma variável de valor 25 será representada por " 25". Já uma variável com valor -300000 será representada por "-300000".

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 a variável longint especificada com o número de caracteres especificado. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TX N Espaços + LongInt.dxg**

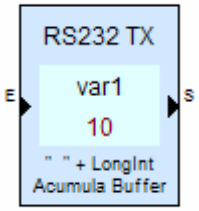
O exemplo transmite o valor da variável var1 com 20 caracteres via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s. Também incrementa a variável var1 a cada transmissão.

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX LongInt Mínimo
RS232 - TXBuffer LongInt Mínimo
RS232 - TX N Zeros + LongInt
RS232 - TXBuffer N Zeros + LongInt
RS232 - TXBuffer N Espaços + LongInt

RS232 - TXBuffer N Espaços + LongInt

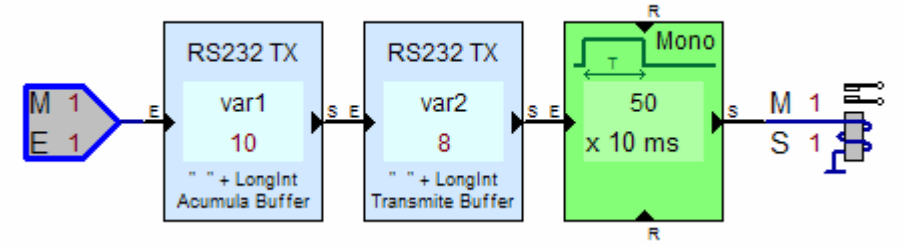
Este bloco permite incluir a representação do valor de uma variável longint, preenchendo o campo com espaços à esquerda, no buffer de porta serial RS232 do controlador µDX200.



Este bloco inclui a variável especificada no buffer de transmissão serial mas, ao contrário do bloco anterior, não transmite o buffer pela porta serial do µDX200. Com isso é possível montar toda mensagem a ser transmitida pela serial no buffer, e depois a transmitir.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo

ativo até transmitir via porta RS232 a variável longint especificada com o número de caracteres especificado. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco.



Exemplo de Programa Aplicativo: **RS232 - TXBuffer N Espaços + LongInt.dxg**

O exemplo transmite o valor da variável var1, com 10 caracteres, e da variável var2, com 8 caracteres, via porta serial cada vez que a entrada digital E1 da Expansão M1 é energizada, e aciona momentaneamente o nodo S que liga a saída S1 por 0,5 s.

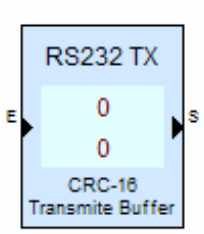
Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX LongInt Mínimo
RS232 - TXBuffer LongInt Mínimo
RS232 - TX N Zeros + LongInt
RS232 - TXBuffer N Zeros + LongInt
RS232 - TX N Espaços + LongInt

RS232 - TX CRC-16

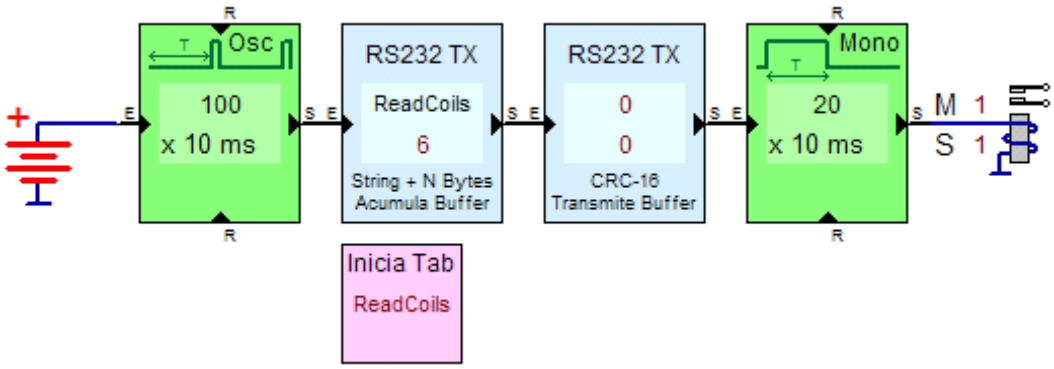
Calcula o CRC-16 (cyclic redundancy check) do buffer de transmissão de serial, inclui este valor ao final do buffer de serial, e transmite todo buffer serial. ByteInicial indica quantos bytes a partir do início do buffer devem ser desconsiderados no cálculo de CRC, e ByteFinal indica quantos bytes devem ser desconsiderados ao final do buffer. Se ByteInicial = ByteFinal = 0 (zero) então todo buffer de transmissão serial será considerado no cálculo de CRC.



Note que ByteInicial e ByteFinal podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 o buffer acrescido do CRC calculado. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.

Este bloco deve ser usado para gerar mensagens a serem transmitidas via serial que utilizem a checagem de erros por CRC-16. Por exemplo, no caso de protocolo ModBus-RTU. Note que o CRC é incluído no buffer de transmissão serial em formato binário (2 bytes), sendo que o byte LSB (menos significativo) é transmitido antes do byte MSB (mais significativo).



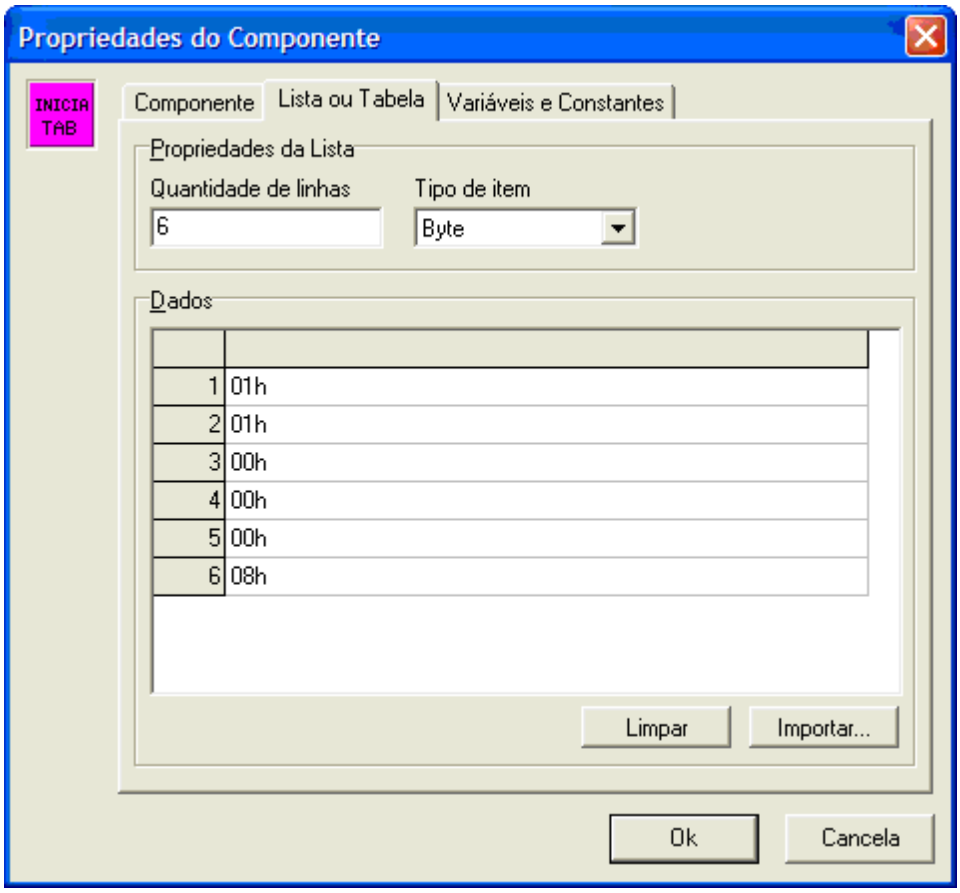
Exemplo de Programa Aplicativo: **RS232 - TX CRC-16.dxg**

Por exemplo, o programa acima permite transmitir o comando Read Coils (comando 01h) em ModBus RTU. O formato do comando é:

µDX200 → Dispositivo ModBus

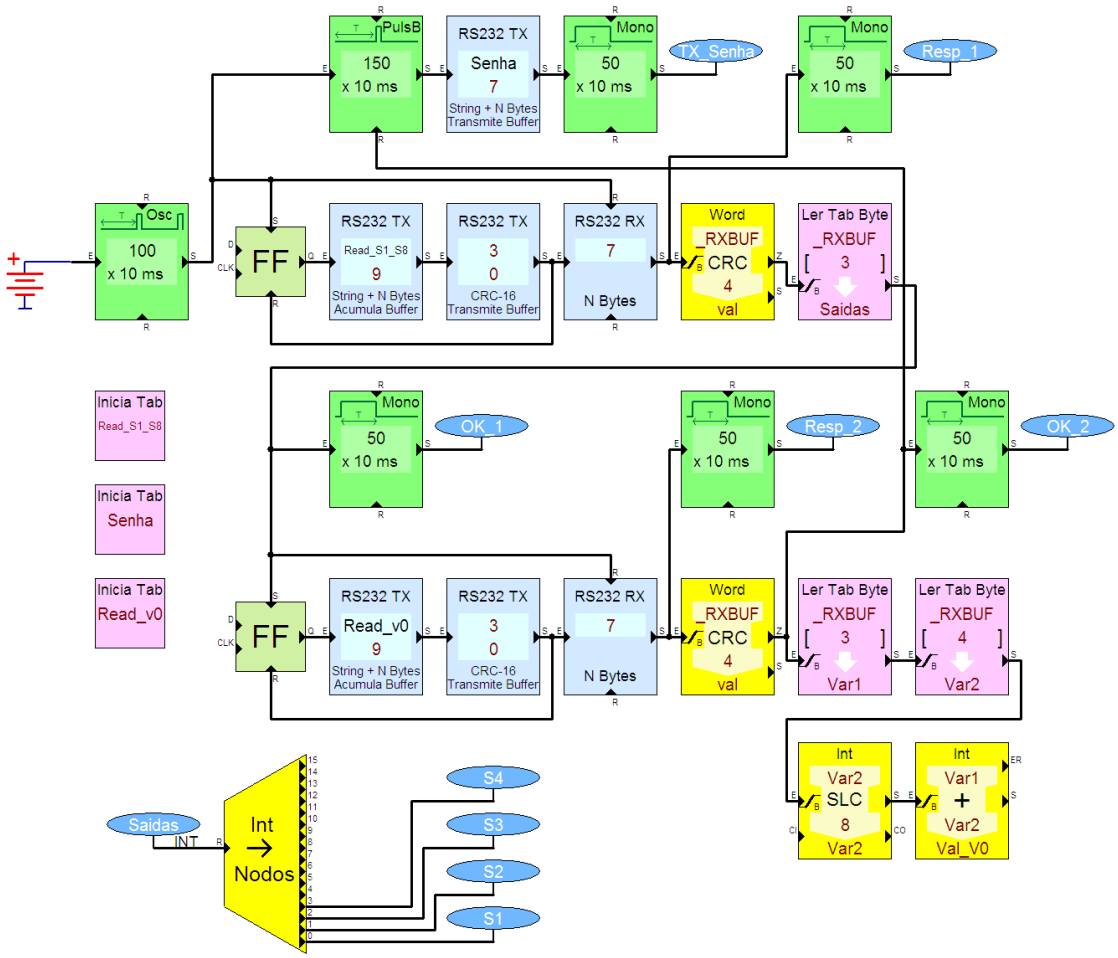
Endereço	Comando	Ender. Inicial (MSB)	Ender. Inicial (LSB)	Número de bits (MSB)	Número de bits (LSB)	CRC-16 (LSB)	CRC-16 (MSB)
01h	01h	00h	00h	00h	08h	CRC	CRC

Note que o µDX200 está consultando os primeiros 8 nodos ("coils") do dispositivo ModBus de endereço 01h. A tabela ReadCoils é inicializada com os seguintes valores:



Atenção: *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

O mesmo cálculo de CRC é usado no protocolo nativo do µDX200. O exemplo a seguir transmite duas perguntas em protocolo nativo para um segundo CLP conectado via porta serial RS232. Além disso, caso não receba resposta é transmitida a senha padrão **[uDX200** para forçar o µDX200 a retornar ao modo programação.



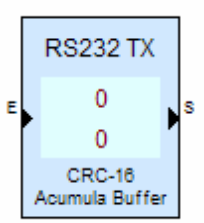
Exemplo de Programa Aplicativo: **RS232 - TX CRC-16 Nativo.dxg**

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TXBuffer CRC-16
RS232 - TX FCS
RS232 - TXBuffer FCS
RS232 - TX CRC-DNP
RS232 - TXBuffer CRC-DNP

RS232 - TXBuffer CRC-16

Calcula o CRC-16 (cyclic redundancy check) do buffer de transmissão de serial e inclui este valor ao final do buffer de serial. Note que, ao contrário do bloco anterior, este bloco não transmite o buffer via serial. ByteInicial indica quantos bytes a partir do início do buffer devem ser desconsiderados no cálculo de CRC, e ByteFinal indica quantos bytes devem ser desconsiderados ao final do buffer. Se ByteInicial = ByteFinal = 0 (zero) então todo buffer de transmissão serial será considerado no cálculo de CRC. Note que o CRC é incluído no buffer de transmissão serial em formato binário (2 bytes). Note que os bytes são incluídos no buffer de transmissão serial na ordem LSB (menos significativo) e MSB (mais significativo).



Note que ByteInicial e ByteFinal podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir o CRC calculado no buffer de transmissão serial. Quando for concluída a inserção será gerado um pulso no nodo de Saída (S) do bloco.

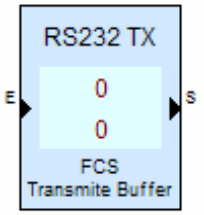
> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX CRC-16
RS232 - TX FCS
RS232 - TXBuffer FCS
RS232 - TX CRC-DNP
RS232 - TXBuffer CRC-DNP

RS232 - TX FCS

Calcula o FCS (frame check sequence) do buffer de transmissão de serial, inclui este valor ao final do buffer de serial, e transmite todo buffer serial. ByteInicial indica quantos bytes a partir do início do buffer devem ser desconsiderados, e ByteFinal indica quantos bytes devem ser desconsiderados ao final do buffer. Se ByteInicial = ByteFinal = 0 (zero) então todo buffer de transmissão serial será considerado no cálculo de FCS.



Note que ByteInicial e ByteFinal podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).

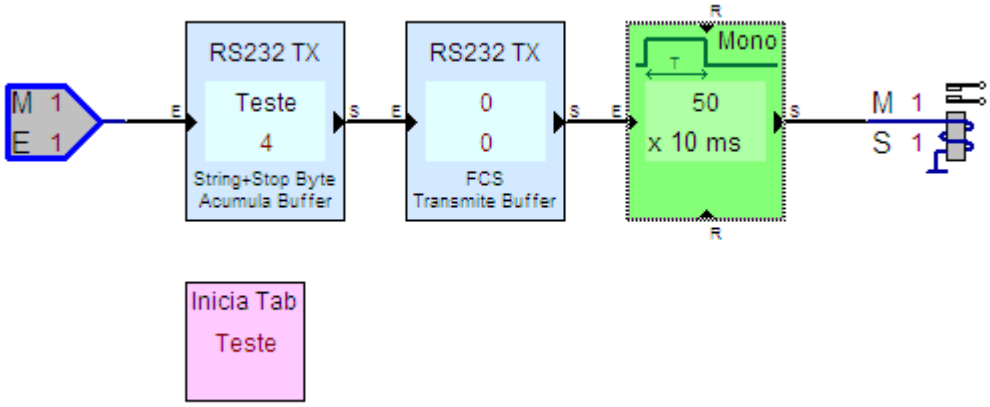
Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 o buffer acrescido do FCS calculado. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.

Este bloco deve ser usado para gerar mensagens a serem transmitidas via serial que utilizem a checagem de erros por FCS. É o caso, por exemplo, do protocolo utilizado no µDX100. O FCS é um byte calculado pelo complemento de dois da soma de todos os bytes da mensagem, desprezando-se o estouro (carry). Se a mensagem for 56h F0h 50h 04h a soma será: 56h+F0h+50h+04h = 19Ah. Desprezando o byte MSB resulta em 9Ah. Portanto, o FCS será o complemento de dois de 9Ah, ou seja: 100h-9Ah = 66h.

Para verificar se uma mensagem possui um FCS correto (ou seja, checar se a mensagem não possui erros) basta somar todos os bytes da mensagem, inclusive o FCS. O resultado, desprezando-se o byte superior da soma, deve ser zero. No exemplo anterior a mensagem completa, com FCS, seria:

56h F0h 50h 04h 66h

A soma destes valores resulta em 200h. Desprezando o byte MSB da soma resulta em 00h → mensagem ok!



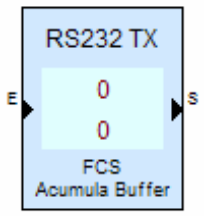
Exemplo de Programa Aplicativo: **RS232 - TX FCS.dxg**

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX CRC-16
RS232 - TXBuffer CRC-16
RS232 - TXBuffer FCS
RS232 - TX CRC-DNP
RS232 - TXBuffer CRC-DNP

RS232 - TXBuffer FCS

Calcula o FCS (frame check sequence) do buffer de transmissão de serial, e inclui este valor ao final do buffer de serial. Ao contrário do bloco anterior, este bloco não transmite o buffer serial. ByteInicial indica quantos bytes a partir do início do buffer devem ser desconsiderados, e ByteFinal indica quantos bytes devem ser desconsiderados ao final do buffer. Se ByteInicial = ByteFinal = 0 (zero) então todo buffer de transmissão serial será considerado no cálculo de FCS.



Note que ByteInicial e ByteFinal podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir no buffer de transmissão serial o FCS calculado. Quando for concluída a inclusão será gerado um pulso no nodo de Saída (S) do bloco.

Este bloco deve ser usado para gerar mensagens a serem transmitidas via serial que utilizem a checagem de erros por FCS. É o caso, por exemplo, do protocolo utilizado no µDX100. O FCS é um byte calculado pelo complemento de dois da soma de todos os bytes da mensagem, desprezando-se o estouro (carry). Se a mensagem for 56h F0h 50h 04h a soma será: 56h+F0h+50h+04h = 19Ah. Desprezando o byte MSB resulta em 9Ah. Portanto, o FCS será o complemento de dois de 9Ah, ou seja: 100h-9Ah = 66h.

Para verificar se uma mensagem possui um FCS correto (ou seja, checar se a mensagem não possui erros) basta somar todos os bytes da mensagem, inclusive o FCS. O resultado, desprezando-se o byte superior da soma, deve ser zero. No exemplo anterior a mensagem completa, com FCS, seria:

56h F0h 50h 04h 66h

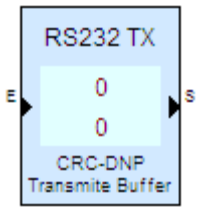
A soma destes valores resulta em 200h. Desprezando o byte MSB da soma resulta em 00h → mensagem ok!

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
RS232 - TX CRC-16
RS232 - TXBuffer CRC-16
RS232 - TX FCS
RS232 - TX CRC-DNP
RS232 - TXBuffer CRC-DNP

RS232 - TX CRC-DNP

Calcula o CRC-16 (cyclic redundancy check) do buffer de transmissão de serial, inclui este valor ao final do buffer de serial, e transmite todo buffer serial. O cálculo de CRC é baseado nas especificações do protocolo DNP-3. ByteInicial indica quantos bytes a partir do início do buffer devem ser desconsiderados no cálculo de CRC, e ByteFinal indica quantos bytes devem ser desconsiderados ao final do buffer. Se ByteInicial = ByteFinal = 0 (zero) então todo buffer de transmissão serial será considerado no cálculo de CRC.



Note que ByteInicial e ByteFinal podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 o buffer acrescido do CRC calculado. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.

Este bloco deve ser usado para gerar mensagens a serem transmitidas via serial que utilizem a checagem de erros por CRC-16 padrão DNP-3. Note que o CRC é incluído no buffer de transmissão serial em formato binário (2 bytes), sendo que o byte MSB (mais significativo) é transmitido antes do byte LSB (menos significativo).

Atenção: *é necessário que o µDX201 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX201 não habilita os blocos de comunicação serial do programa aplicativo.*

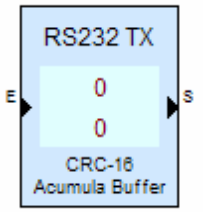
Atenção: Bloco disponível apenas em controlador µDX201.

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
Macros que acompanham o PG
RS232 - TX CRC-16
RS232 - TXBuffer CRC-16
RS232 - TX FCS
RS232 - TXBuffer FCS
RS232 - TXBuffer CRC-DNP

RS232 - TXBuffer CRC-DNP

Calcula o CRC-16 (cyclic redundancy check) do buffer de transmissão de serial e inclui este valor ao final do buffer de serial. O cálculo de CRC é baseado nas especificações do protocolo DNP-3. Note que, ao contrário do bloco anterior, este bloco não transmite o buffer via serial. ByteInicial indica quantos bytes a partir do início do buffer devem ser desconsiderados no cálculo de CRC, e ByteFinal indica quantos bytes devem ser desconsiderados ao final do buffer. Se ByteInicial = ByteFinal = 0 (zero) então todo buffer de transmissão serial será considerado no cálculo de CRC. Note que o CRC é incluído no buffer de transmissão serial em formato binário (2 bytes). Note que os bytes são incluídos no buffer de transmissão serial na ordem MSB (mais significativo) e LSB (menos significativo).



Note que ByteInicial e ByteFinal podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir o CRC calculado no buffer de transmissão serial. Quando for concluída a inserção será gerado um pulso no nodo de Saída (S) do bloco.

Atenção: *é necessário que o µDX201 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX201 não habilita os blocos de comunicação serial do programa aplicativo.*

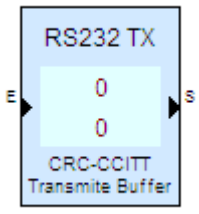
Atenção: Bloco disponível apenas em controlador µDX201.

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
Macros que acompanham o PG
RS232 - TX CRC-16
RS232 - TXBuffer CRC-16
RS232 - TX FCS
RS232 - TXBuffer FCS
RS232 - TX CRC-DNP

RS232 - TX CRC-CCITT

Calcula o CRC-16 (cyclic redundancy check) do buffer de transmissão de serial, inclui este valor ao final do buffer de serial, e transmite todo buffer serial. O cálculo de CRC é baseado nas especificações CCITT. ByteInicial indica quantos bytes a partir do início do buffer devem ser desconsiderados no cálculo de CRC, e ByteFinal indica quantos bytes devem ser desconsiderados ao final do buffer. Se ByteInicial = ByteFinal = 0 (zero) então todo buffer de transmissão serial será considerado no cálculo de CRC.



Note que ByteInicial e ByteFinal podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até transmitir via porta RS232 o buffer acrescido do CRC calculado. Quando for concluída a transmissão será gerado um pulso no nodo de Saída (S) do bloco.

Este bloco deve ser usado para gerar mensagens a serem transmitidas via serial que utilizem a checagem de erros por CRC-16 padrão CCITT. Note que o CRC é incluído no buffer de transmissão serial em formato binário (2 bytes), sendo que o byte MSB (mais significativo) é transmitido antes do byte LSB (menos significativo).

Atenção: *é necessário que o µDX201 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX201 não habilita os blocos de comunicação serial do programa aplicativo.*

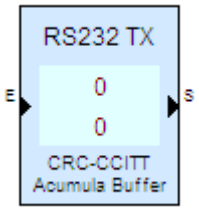
Atenção: Bloco disponível apenas em controlador µDX201 (µDX200 somente v2.24 ou superior).

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

Veja também:
Macros que acompanham o PG
RS232 - TX CRC-16
RS232 - TXBuffer CRC-16
RS232 - TX FCS
RS232 - TXBuffer FCS
RS232 - TXBuffer CRC-DNP

RS232 - TXBuffer CRC-CCITT

Calcula o CRC-16 (cyclic redundancy check) do buffer de transmissão de serial e inclui este valor ao final do buffer de serial. O cálculo de CRC é baseado nas especificações CCITT. Note que, ao contrário do bloco anterior, este bloco não transmite o buffer via serial. ByteInicial indica quantos bytes a partir do início do buffer devem ser desconsiderados no cálculo de CRC, e ByteFinal indica quantos bytes devem ser desconsiderados ao final do buffer. Se ByteInicial = ByteFinal = 0 (zero) então todo buffer de transmissão serial será considerado no cálculo de CRC. Note que o CRC é incluído no buffer de transmissão serial em formato binário (2 bytes). Note que os bytes são incluídos no buffer de transmissão serial na ordem MSB (mais significativo) e LSB (menos significativo).



Note que ByteInicial e ByteFinal podem ser variáveis inteiras, constantes (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória (Mxxxx).

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, permanecendo ativo até incluir o CRC calculado no buffer de transmissão serial. Quando for concluída a inserção será gerado um pulso no nodo de Saída (S) do bloco.

Atenção: *é necessário que o µDX201 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX201 não habilita os blocos de comunicação serial do programa aplicativo.*

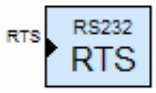
Atenção: Bloco disponível apenas em controlador µDX201 (µDX200 somente v2.24 ou superior).

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas. Para uso das portas seriais auxiliares existe a família de blocos RS232-2 e RS232-3, com blocos similares aos usados pela porta serial principal. Note que, no caso das portas seriais auxiliares, elas não suportam protocolo nativo, ModBus ou do µDX100. Para ativar as portas seriais auxiliares é preciso habilitá-las via Configuração de Hardware no programa aplicativo.*

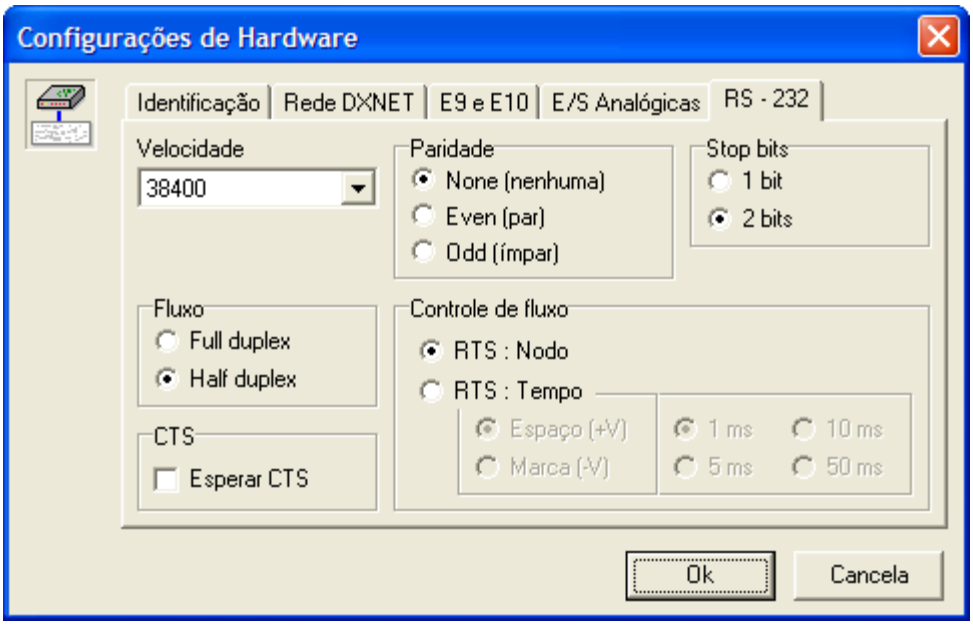
Veja também:
Macros que acompanham o PG
RS232 - TX CRC-16
RS232 - TXBuffer CRC-16
RS232 - TX FCS
RS232 - TXBuffer FCS
RS232 - TX CRC-DNP

RS232 - RTS

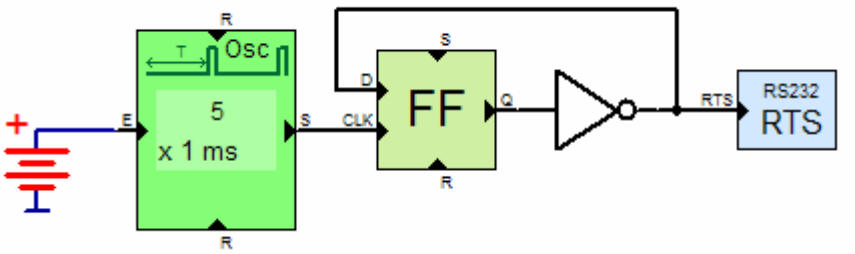
Este bloco comanda o sinal de RTS presente na porta serial RS232 (pino 7 do conector DB-9) conforme o nodo de entrada. RTS é o sinal conhecido como Request to Send da porta serial. Note que o RTS pode ser comandado por este bloco, ou ser automaticamente ativado por temporização no µDX200.



Para que ele seja comandado pelo bloco RTS é necessário escolher modo **RTS: Nodo** na janela de **Configuração de Hardware**, existente no Compilador para µDX200:



O exemplo a seguir aplica uma onda quadrada de 100Hz a saída RTS da porta serial do µDX200:



Exemplo de Programa Aplicativo: **RS232 - RTS.dxg**

Note que o oscilador gera um pulso a cada 5ms, ou seja, em uma freqüência de 200Hz. Estes pulsos são divididos por dois pelo FF (que troca de estado a saída Q a cada pulso em CLK graças a realimentação de Q invertido para a entrada D) e aplicados a saída RTS.

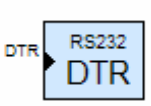
Note que podemos usar este pino como uma saída digital adicional para o µDX200. Ele excurciona entre -5V e 5V, podendo ativar relés de estado sólido, por exemplo.

> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas.*

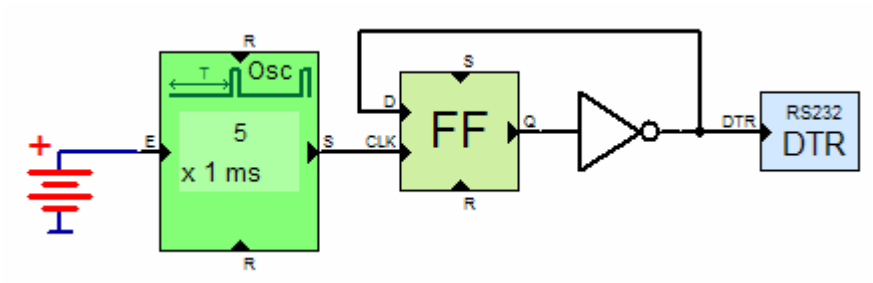
Veja também:
RS232 - DTR
RS232 - CTS
RS232 - DSR
RS232 - DCD
RS232 - RING

RS232 - DTR

Este bloco comanda o sinal de DTR presente na porta serial RS232 (pino 4 do conector DB-9) conforme o nodo de entrada. O DTR é o sinal de Data Terminal Ready da porta serial.



O exemplo a seguir aplica uma onda quadrada de 100Hz a saída DTR da porta serial do µDX200:



Exemplo de Programa Aplicativo: **RS232 - DTR.dxg**

Note que o oscilador gera um pulso a cada 5ms, ou seja, em uma freqüência de 200Hz. Estes pulsos são divididos por dois pelo FF (que troca de estado a saída Q a cada pulso em CLK graças a realimentação de Q invertido para a entrada D) e aplicados a saída DTR.

Note que podemos usar este pino como uma saída digital adicional para o µDX200. Ele excurciona entre -5V e 5V, podendo ativar relés de estado sólido, por exemplo.

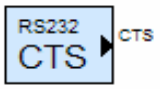
> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial*

principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas.

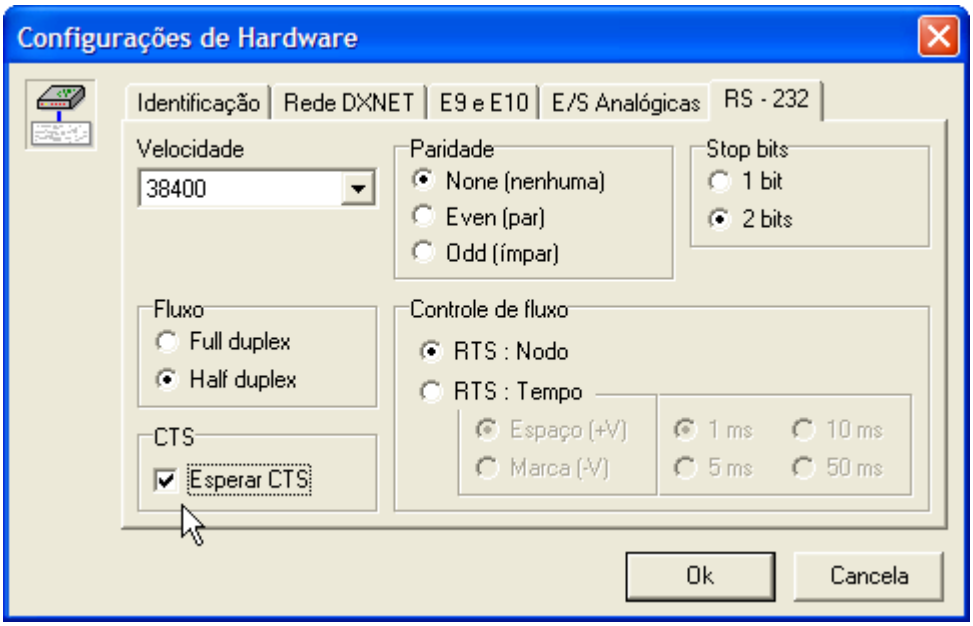
Veja também:
RS232 - RTS
RS232 - CTS
RS232 - DSR
RS232 - DCD
RS232 - RING

RS232 - CTS

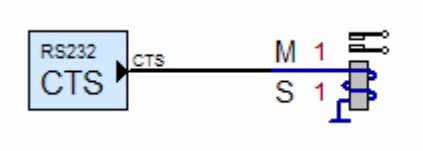
Este bloco lê o sinal de CTS presente na porta serial RS232 (pino 8 do conector DB-9), disponibilizando-o no nodo de saída do bloco. O CTS é o sinal de Clear to Send da porta serial. O µDX200 pode ser programado para que só transmita pela serial caso o CTS esteja habilitado, ou pode desconsiderar este sinal (este é o estado padrão).



Note que se o µDX200 for programado para habilitar a verificação de CTS e o cabo serial utilizado não possuir conexão das linhas de handshake da porta serial simplesmente a comunicação será interrompida. Abaixo a figura mostra a janela de configuração de hardware do Compilador com a verificação de CTS habilitada:



O exemplo lê o sinal de CTS da porta serial do µDX200 e o replica na saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210):



Exemplo de Programa Aplicativo: **RS232 - CTS.dxg**

Note que podemos usar este pino como uma entrada digital adicional para o µDX200. Ele permite sinais entre -12V e 12V.

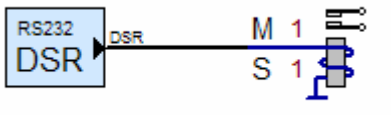
> ***Atenção:*** *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas.*

Veja também:
RS232 - RTS
RS232 - DTR
RS232 - DSR
RS232 - DCD
RS232 - RING

RS232 - DSR

Este bloco lê o sinal de DSR presente na porta serial RS232 (pino 6 do conector DB-9), disponibilizando-o no nodo de saída do bloco. O DSR é o sinal de Data Set Ready da porta serial.

O exemplo lê o sinal de DSR da porta serial do µDX200 e o replica na saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210):



Exemplo de Programa Aplicativo: **RS232 - DSR.dxg**

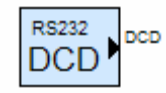
Note que podemos usar este pino como uma entrada digital adicional para o µDX200. Ele permite sinais entre -12V e 12V.

Atenção: *Controladores µDX201 versão 3.46 ou superior permitem o uso dos sinais de controle da porta serial (RTS,CTS,DTR e DSR) como portas seriais auxiliares. Estas portas seriais auxiliares são designadas como RS232-2 e RS232-3, e permitem taxas de transmissão de 110 bps a 9600bps. A porta RS232-2 é implementada com os sinais RTS (TX) e CTS (RX) da porta serial principal. Já a porta serial RS232-3 é implementada com os sinais DTR (TX) e DSR (RX) da porta serial principal. No caso destas portas seriais auxiliares estarem ativadas via Configuração de Hardware estes sinais deixam de ser comandados pelos respectivos nodos absolutos, e passam a serem usados como TX e RX destas portas.*

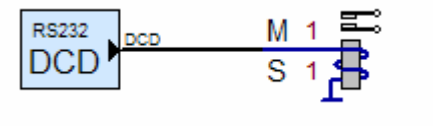
Veja também:
RS232 - RTS
RS232 - DTR
RS232 - CTS
RS232 - DCD
RS232 - RING

RS232 - DCD

Este bloco lê o sinal de DCD presente na porta serial RS232 (pino 1 do conector DB-9), disponibilizando-o no nodo de saída do bloco. O DCD é o sinal de Data Carrier Detect da porta serial.



O exemplo lê o sinal de DCD da porta serial do µDX200 e o replica na saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210):



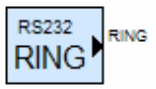
Exemplo de Programa Aplicativo: **RS232 - DCD.dxg**

Note que podemos usar este pino como uma entrada digital adicional para o µDX200. Ele permite sinais entre -12V e 12V.

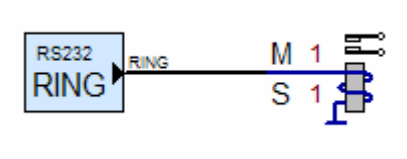
Veja também:
RS232 - RTS
RS232 - DTR
RS232 - CTS
RS232 - DSR
RS232 - RING

RS232 - RING

Este bloco lê o sinal de RING presente na porta serial RS232 (pino 9 do conector DB-9), disponibilizando-o no nodo de saída do bloco. O RING é o sinal de Ring Indicator da porta serial.



O exemplo lê o sinal de RING da porta serial do µDX200 e o replica na saída S1 do módulo M1 de Expansão de Entradas/Saídas (µDX210):



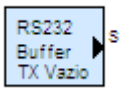
Exemplo de Programa Aplicativo: **RS232 - RING.dxg**

Note que podemos usar este pino como uma entrada digital adicional para o µDX200. Ele permite sinais entre -12V e 12V.

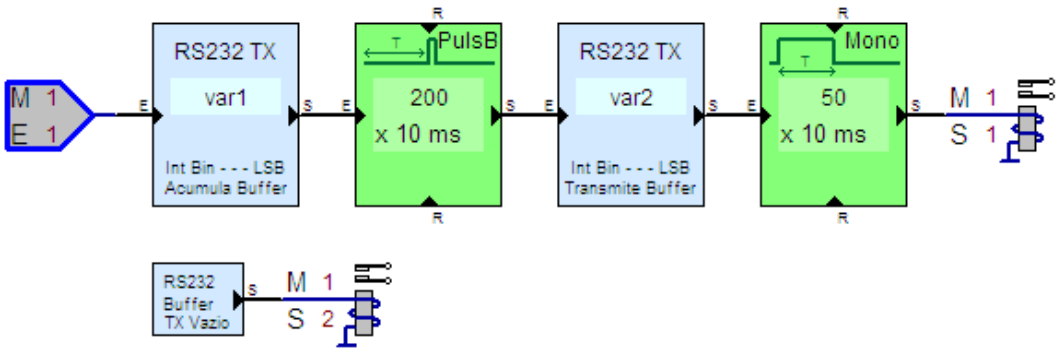
Veja também:
RS232 - RTS
RS232 - DTR
RS232 - CTS
RS232 - DSR
RS232 - DCD

RS232 - Buffer TX Vazio

Este bloco indica que o buffer de transmissão serial está vazio.



Abaixo se reproduz um pequeno programa para teste deste bloco. Note que o µDX200 deve estar em modo Normal.



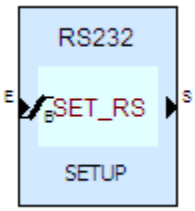
Exemplo de Programa Aplicativo: **RS232 - Buffer TX Vazio.dxg**

Este sinal é importante para testar se já foi completada a transmissão serial, de forma a permitir o uso da porta serial novamente. No exemplo acima foi inserido propositadamente um atraso de 2 segundos para transmissão dos dados do buffer de transmissão serial. Durante este período a saída S2 é desacionada pelo bloco de Buffer TX Vazio, indicando que o buffer neste período não está vazio.

> ***Atenção:*** *é necessário que o µDX200 esteja no Modo Normal para que o programa aplicativo tenha acesso a porta serial. No modo Programação o µDX200 não habilita os blocos de comunicação serial do programa aplicativo.*

RS232 - Setup

O bloco **Setup RS232** permite especificar os parâmetros da porta serial RS232 (baud-rate, paridade, stop bits) do controlador µDX201. Cuidado ao especificar esses parâmetros, pois o µDX201 irá assumir a nova configuração imediatamente. Se o programa PG estiver monitorando o CLP e a nova configuração de RS232 for diferente ele irá perder comunicação e terá de ser feita a correção dos parâmetros no **Comunicador** do PG. O valor especificado através desse bloco é usado prioritariamente em relação ao especificado na **Configuração de Hardware**. No caso de um hard reset do CLP o valor da **Configuração de Hardware** é restabelecido.



Operando é o operando cujo valor será transferido para os parâmetros de porta serial RS232.

Note que Operando pode ser variável word, constante (valor numérico em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

O nodo de Entrada (E) habilita o bloco por nível ou por borda de subida (para editar o bloco selecionando o modo de operação e os parâmetros aponte para o bloco com o mouse e pressione a tecla direita desse). Já o nodo de Saída (S) é ativado ao completar a operação.

Note que variáveis word podem assumir valores entre 0 e 65535 (0 e 0FFFFh em hexadecimal).

Se o nodo de Entrada (E) estiver selecionado para acionamento por nível o bloco será executado sempre que este nodo estiver ligado, e na velocidade de ciclo do µDX201 (o que pode atingir milhares de vezes por segundo). Já no caso do nodo de Entrada (E) ser ativado por borda o bloco só é executado uma vez na borda de subida de E, aguardando este nodo desligar para permitir novo acionamento.

Configuração de porta serial

Permite configurar a porta serial (RS232) de acordo com a necessidade do bloco de RX-DXNET µDX100. O byte baixo (bits 7-0) indicam a velocidade e o byte alto (bits 15-8) indicam os detalhes da configuração conforme indicado:

Bit 15 – 0 = No parity, 1 = Parity enabled
Bit 14 – 0 = odd, 1 = even
Bit 13 – 0 = 1 stop bit, 1 = 2 stop bits

Bit 12 – 1 = 8 bits
Bit 11 – 0
Bit 10 – 0
Bit 9 – 0
Bit 8 – 0

Note que o bit 12 deve ser, necessariamente, ligado (8 bits). O bloco não aceita o valor de configuração caso esse bit esteja desligado (7 bits) porque isso poderia causar trancamento na comunicação com o software PG. Os bits 11 a 8 devem estar, necessariamente, em zero, para que o parâmetro seja considerado válido.

A velocidade indicada no byte low segue a seguinte tabela:

Valor	bps	Valor	bps	Valor	bps
0	110	1	300	2	600
3	1200	4	2400	5	4800
6	9600	7	19200	8	38400
9	57600	A	115200	B	250000

Normalmente no modo programação esta porta está configurada como 38400, N, 8, 2 (Configuração = 3008h).

Abaixo algumas configurações usuais:

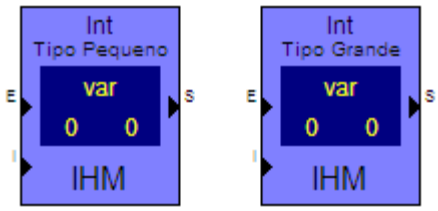
Sem paridade, 8 bits, 1 stop bit.		
110,n,8,1	→	1000h
300,n,8,1	→	1001h
600,n,8,1	→	1002h
1200,n,8,1	→	1003h
2400,n,8,1	→	1004h
4800,n,8,1	→	1005h
9600,n,8,1	→	1006h
19200,n,8,1	→	1007h
38400,n,8,1	→	1008h
57600,n,8,1	→	1009h
115200,n,8,1	→	100Ah
250000,n,8,1	→	100Bh

Sem paridade, 8 bits, 2 stop bits.		
110,n,8,2	→	3000h
300,n,8,2	→	3001h
600,n,8,2	→	3002h
1200,n,8,2	→	3003h
2400,n,8,2	→	3004h
4800,n,8,2	→	3005h
9600,n,8,2	→	3006h
19200,n,8,2	→	3007h
38400,n,8,2	→	3008h
57600,n,8,2	→	3009h
115200,n,8,2	→	300Ah
250000,n,8,2	→	300Bh

Atenção: *é possível inibir a Configuração de porta RS232 do bloco simplesmente utilizando o valor* ***0000h*** *ou* ***0FFFFh*** *(valores inválidos). Com isso o bloco não irá alterar os parâmetros de porta serial RS232, mantendo os definidos na Configuração de Hardware ou em outros blocos em que existam esse parâmetro.*

IHM - Escreve Inteiro

Permite escrever valores de variáveis inteiras no display da Interface Homem/Máquina (IHM) para µDX201. Existem dois blocos, um que escreve em caracteres pequenos (7x5), e outro que escreve em caracteres grandes (14x10).



Os parâmetros destes blocos são os seguintes:

Operando → Trata-se da variável cujo valor será escrito no display. Pode ser uma variável inteira

ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

Tamanho → Número de dígitos utilizados. Note que o ponto decimal e o sinal de número negativo são contados como dígitos. Este parâmetro é expresso em um byte. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante.

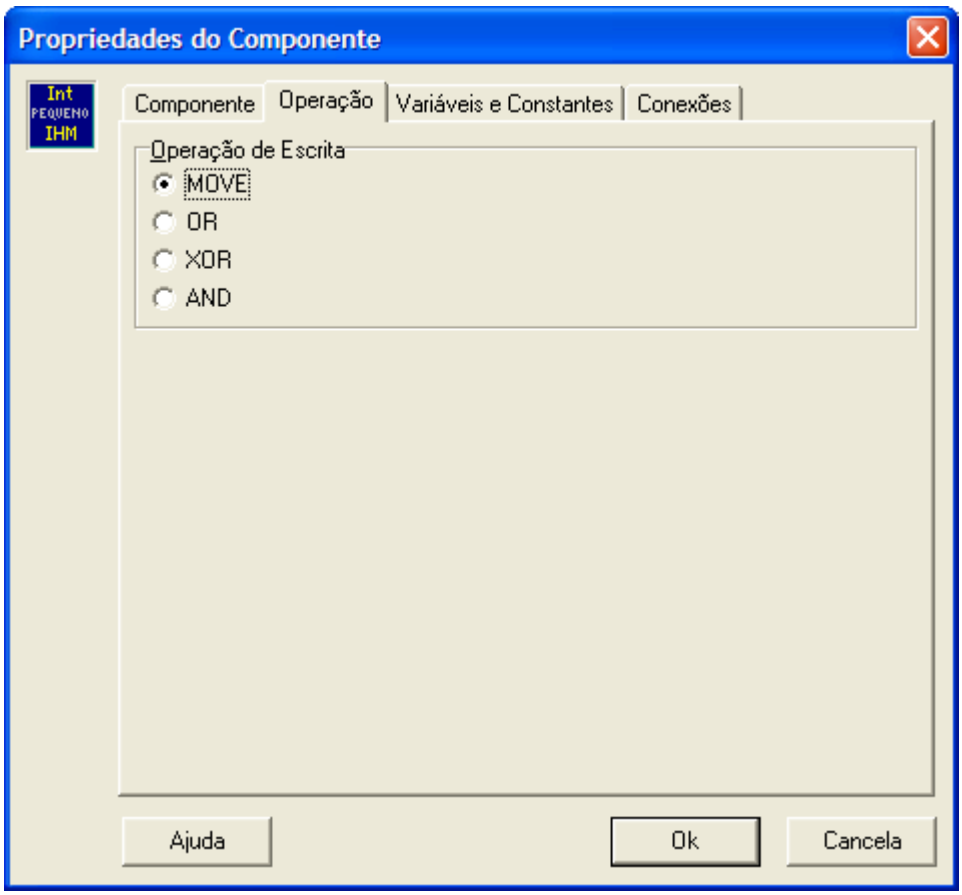
X → Posição X para início da escrita no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 128 linhas, este valor pode variar entre 0 e 127.

Y → Posição Y para início da escrita no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 64 colunas, este valor pode variar entre 0 e 63.

Ponto → Posição do ponto decimal, se necessário. Este parâmetro é expresso em um byte. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. Caso este parâmetro seja zero não é empregado ponto decimal na representação de **Operando**.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, escrevendo o valor de **Operando** no display da IHM. Quando for concluída a escrita será gerado um pulso no nodo de Saída (S) do bloco. O nodo Inverte (I) permite reverter a escrita, tornando todos os pixels ligados em desligados e vice-versa.

Na edição deste bloco, além dos parâmetros acima, também é possível definir o tipo de operação a ser efetuada no display da IHM. Assim, existem as seguintes opções:



A operação **MOVE** é a padrão, e permite sobreescrever a área acessada pelo bloco no display da

IHM, apagando os dados anteriores. Ou seja, independentemente de quais pixels estivessem ligados na área acessada pelo bloco, todos eles serão substituídos pelos novos dados.

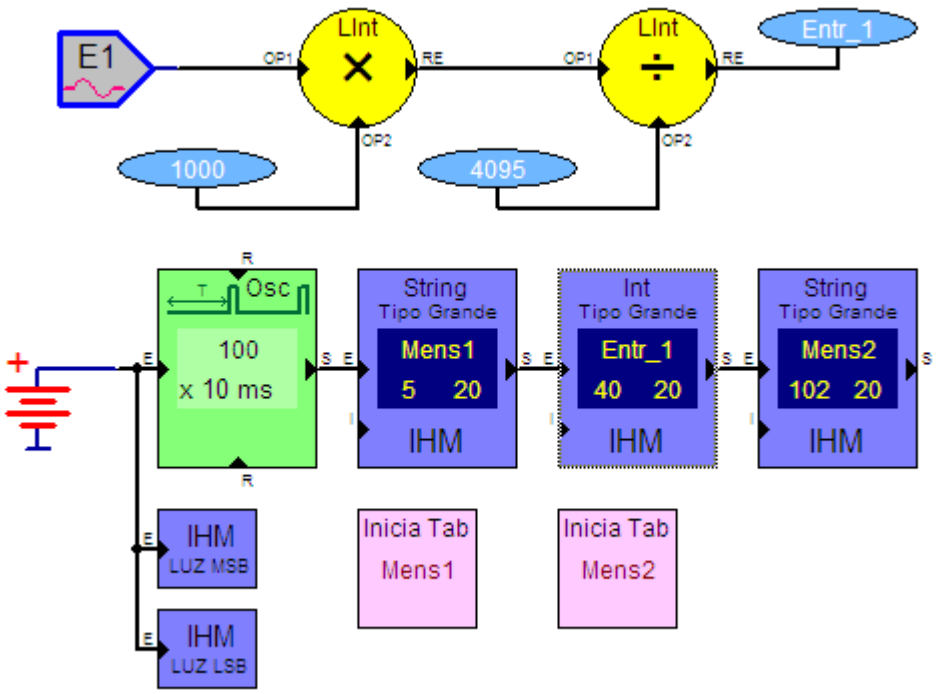
A operação **OR** escreve preservando todos os dados previamente existentes no display. Portanto, todos os pixels que já estivessem ligados na área acessada pelo bloco serão mantidos ligados. Com isso os dados já existentes no display são mantidos.

A operação **XOR** reverte os dados a serem escritos sempre que eles forem ocupar uma posição já ativa do display. Então, todos os pixels ligados a serem escritos sobre pixels já acionados serão invertidos, ou seja, os pixels destas posições serão desligados. Isso permite identificar dados sobrepostos através da imagem invertida formada.

A operação **AND** apenas aciona os novos dados a serem escritos em regiões já ativas do display. Na verdade, esta opção é útil para mascarar certas regiões do display, permitindo dados apenas em áreas cujos pixels já estejam ligados.

Atenção: Bloco disponível apenas em controlador µDX201.

O exemplo a seguir lê a entrada analógica E1 na escala 0-10V e apresenta o valor no display da IHM para µDX201 (módulo µDX220), com duas casas decimais.



Exemplo de Programa Aplicativo: **IHM - Escreve Inteiro.dxg**

Tela de programa aplicativo: **IHM - Escreve Inteiro.dxg**

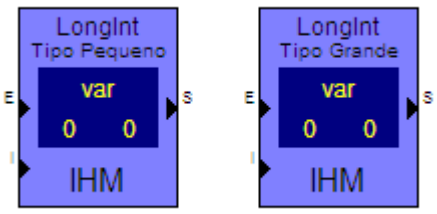
Blocos IHM - Escreve Inteiro X

A partir da versão **2.2.2.1** do software Programador Gráfico **PG** para µDX200 estão disponíveis os blocos **IHM X**, que possuem a mesma funcionalidade de seus equivalentes blocos de **IHM**, mas permitem que todos os parâmetros do bloco sejam informados através de variáveis. Os blocos de **IHM** originais possuem alguns parâmetros em formato byte, o que não permitia o uso de uma variável para modificar seus valores. No caso de escrita inteira, o parâmetro **Ponto** (que determina a posição do ponto decimal) e o parâmetro **Tamanho** (que determina o número total de dígitos - note que o sinal e o ponto decimal contam como uma posição) eram tipo byte nos blocos de **IHM**. Já nos blocos **IHM X** esses dois parâmetros estão reunidos em um único campo inteiro, chamado **Ponto/Tamanho**, que pode receber uma constante ou uma variável. Note que o **byte MSB** determina o ponto decimal, enquanto o **byte LSB** determina o tamanho do campo a ser escrito. Para usar os blocos **IHM X** é necessário controlador programável **µDX201** com firmware versão **3.53** ou superior.

Veja também:
IHM - Escreve LongInt
IHM - Escreve String
IHM - Escreve Bit-Map
Macros que acompanham o PG

IHM - Escreve LongInt

Permite escrever valores de variáveis longint no display da Interface Homem/Máquina (IHM) para µDX201. Existem dois blocos, um que escreve em caracteres pequenos (7x5), e outro que escreve em caracteres grandes (14x10).



Os parâmetros destes blocos são os seguintes:

Operando → Trata-se da variável cujo valor será escrito no display. Pode ser uma variável longint, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória.

Tamanho → Número de dígitos utilizados. Note que o ponto decimal e o sinal de número negativo

são contados como dígitos. Este parâmetro é expresso em um byte. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante.

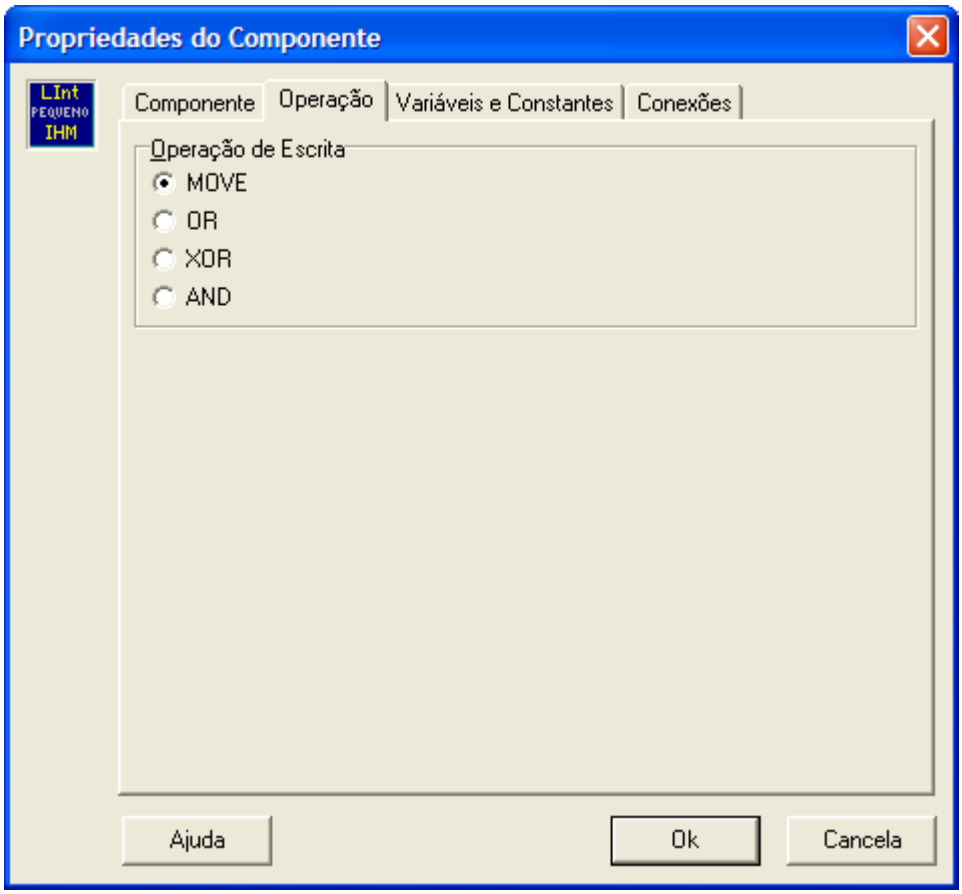
X → Posição X para início da escrita no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 128 linhas, este valor pode variar entre 0 e 127.

Y → Posição Y para início da escrita no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 64 colunas, este valor pode variar entre 0 e 63.

Ponto → Posição do ponto decimal, se necessário. Este parâmetro é expresso em um byte. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. Caso este parâmetro seja zero não é empregado ponto decimal na representação de **Operando**.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, escrevendo o valor de **Operando** no display da IHM. Quando for concluída a escrita será gerado um pulso no nodo de Saída (S) do bloco. O nodo Inverte (I) permite reverter a escrita, tornando todos os pixels ligados em desligados e vice-versa.

Na edição deste bloco, além dos parâmetros acima, também é possível definir o tipo de operação a ser efetuada no display da IHM. Assim, existem as seguintes opções:



A operação **MOVE** é a padrão, e permite sobreescrever a área acessada pelo bloco no display da IHM, apagando os dados anteriores. Ou seja, independentemente de quais pixels estivessem ligados na área acessada pelo bloco, todos eles serão substituídos pelos novos dados.

A operação **OR** escreve preservando todos os dados previamente existentes no display. Portanto,

todos os pixels que já estivessem ligados na área acessada pelo bloco serão mantidos ligados. Com isso os dados já existentes no display são mantidos.

A operação **XOR** reverte os dados a serem escritos sempre que eles forem ocupar uma posição já ativa do display. Então, todos os pixels ligados a serem escritos sobre pixels já acionados serão invertidos, ou seja, os pixels destas posições serão desligados. Isso permite identificar dados sobrepostos através da imagem invertida formada.

A operação **AND** apenas aciona os novos dados a serem escritos em regiões já ativas do display. Na verdade, esta opção é útil para mascarar certas regiões do display, permitindo dados apenas em áreas cujos pixels já estejam ligados.

Atenção: Bloco disponível apenas em controlador µDX201.

Blocos IHM - Escreve LongInt X

A partir da versão **2.2.2.1** do software Programador Gráfico **PG** para µDX200 estão disponíveis os blocos **IHM X**, que possuem a mesma funcionalidade de seus equivalentes blocos de **IHM**, mas permitem que todos os parâmetros do bloco sejam informados através de variáveis. Os blocos de **IHM** originais possuem alguns parâmetros em formato byte, o que não permitia o uso de uma variável para modificar seus valores. No caso de escrita longint, o parâmetro **Ponto** (que determina a posição do ponto decimal) e o parâmetro **Tamanho** (que determina o número total de dígitos - note que o sinal e o ponto decimal contam como uma posição) eram tipo byte nos blocos de **IHM**. Já nos blocos **IHM X** esses dois parâmetros estão reunidos em um único campo inteiro, chamado **Ponto/Tamanho**, que pode receber uma constante ou uma variável. Note que o **byte MSB** determina o ponto decimal, enquanto o **byte LSB** determina o tamanho do campo a ser escrito. Para usar os blocos **IHM X** é necessário controlador programável **µDX201** com firmware versão **3.53** ou superior.

Veja também:
IHM - Escreve Inteiro
IHM - Escreve String
IHM - Escreve Bit-Map
Macros que acompanham o PG

IHM - Escreve String

Permite escrever strings no display da Interface Homem/Máquina (IHM) para µDX201. Existem dois blocos, um que escreve em caracteres pequenos (7x5), e outro que escreve em caracteres grandes (14x10). Note que para declarar strings no µDX201 é preciso recorrer a tabelas.

Os parâmetros destes blocos são os seguintes:

Operando → Trata-se do nome da tabela que contém o string a ser escrito no display. A tabela pode ser composta diretamente por um string, ou ser composta por bytes (que serão impressos

segundo sua representação em código ASCII). É possível representar todas as acentuações mais comuns, assim como vários carateres especiais.

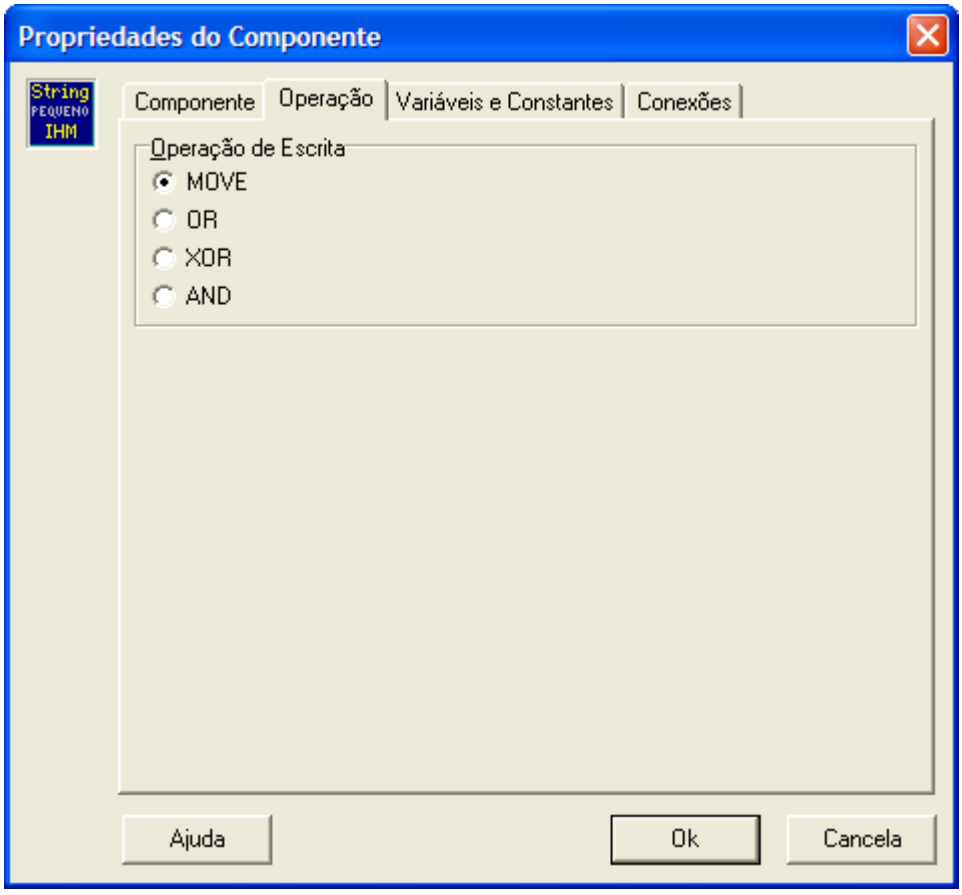
Tamanho → Número de caracteres do string. Este parâmetro é expresso em um byte. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante.

X → Posição X para início da escrita no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 128 linhas, este valor pode variar entre 0 e 127.

Y → Posição Y para início da escrita no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 64 colunas, este valor pode variar entre 0 e 63.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, escrevendo o valor de **Operando** no display da IHM. Quando for concluída a escrita será gerado um pulso no nodo de Saída (S) do bloco. O nodo Inverte (I) permite reverter a escrita, tornando todos os pixels ligados em desligados e vice-versa.

Na edição deste bloco, além dos parâmetros acima, também é possível definir o tipo de operação a ser efetuada no display da IHM. Assim, existem as seguintes opções:



A operação **MOVE** é a padrão, e permite sobreescrever a área acessada pelo bloco no display da IHM, apagando os dados anteriores. Ou seja, independentemente de quais pixels estivessem ligados na área acessada pelo bloco, todos eles serão substituídos pelos novos dados.

A operação **OR** escreve preservando todos os dados previamente existentes no display. Portanto, todos os pixels que já estivessem ligados na área acessada pelo bloco serão mantidos ligados. Com isso os dados já existentes no display são mantidos.

A operação **XOR** reverte os dados a serem escritos sempre que eles forem ocupar uma posição já ativa do display. Então, todos os pixels ligados a serem escritos sobre pixels já acionados serão invertidos, ou seja, os pixels destas posições serão desligados. Isso permite identificar dados sobrepostos através da imagem invertida formada.

A operação **AND** apenas aciona os novos dados a serem escritos em regiões já ativas do display. Na verdade, esta opção é útil para mascarar certas regiões do display, permitindo dados apenas em áreas cujos pixels já estejam ligados.

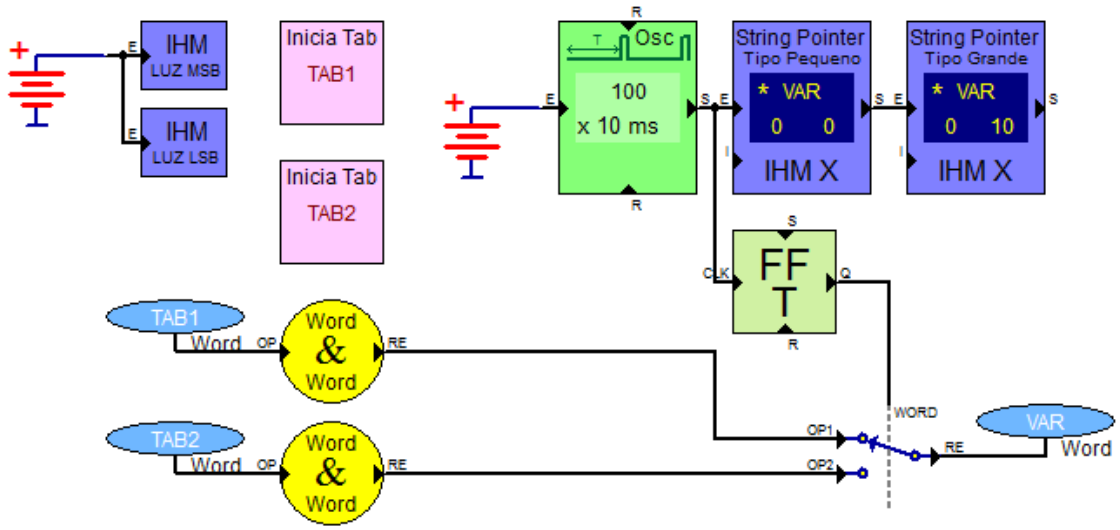
Atenção: Bloco disponível apenas em controlador µDX201.

Blocos IHM - Escreve String X

A partir da versão **2.2.2.1** do software Programador Gráfico **PG** para µDX200 estão disponíveis os blocos **IHM X**, que possuem a mesma funcionalidade de seus equivalentes blocos de **IHM**, mas permitem que todos os parâmetros do bloco sejam informados através de variáveis. Os blocos de **IHM** originais possuem alguns parâmetros em formato byte, o que não permitia o uso de uma variável para modificar seus valores. No caso de escrita string, o parâmetro **Tamanho** (que determina o total de caracteres do string a serem escritos no display) era tipo byte nos blocos de **IHM**. Já nos blocos **IHM X** esse parâmetro é do tipo inteiro e é chamado **Comprimento**, podendo receber uma constante ou uma variável. Para usar os blocos **IHM X** é necessário controlador programável **µDX201** com firmware versão **3.53** ou superior.

Blocos IHM - Escreve String Pointer X

Este bloco é idêntico ao anterior, exceto que **Operando** é um pointer, ou seja, uma variável cujo conteúdo é o endereço do string a ser escrito. Com isso, pode-se mudar o string simplesmente modificando para qual tabela o pointer está apontando. O exemplo a seguir exemplifica essa facilidade. Ele escreve alternadamente na IHM, nos dois tamanhos de caracteres disponíveis, os strings armazenados nas tabelas **TAB1** e **TAB2**. Para isso os endereços de cada tabela são transferidos para a variável **VAR**, sendo que a cada 1 segundo a variável **VAR** aponta para a outra tabela. Com isso, a mensagem na tela da IHM fica alternando, já que os blocos de IHM usam **Operando** como um pointer para obter o endereço do string a ser escrito.

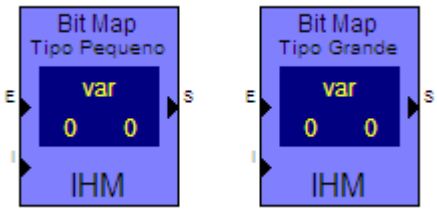


Exemplo de Programa Aplicativo: **Teste_IHM_Pointer.dxg**

Veja também:
IHM - Escreve Inteiro
IHM - Escreve LongInt
IHM - Escreve Bit-Map
Macros que acompanham o PG

IHM - Escreve Bit Map

Permite escrever bit maps no display da Interface Homem/Máquina (IHM) para µDX201. Existem dois blocos, um que escreve bit maps pequenos (8 pixels de altura), e outro que escreve em bit maps grandes (16 pixels de altura). Note que para declarar bit maps no µDX201 é preciso recorrer a tabelas.



Os parâmetros destes blocos são os seguintes:

Operando → Trata-se do nome da tabela que contém o bit map a ser escrito no display. A tabela deve ser composta por bytes (que serão impressos segundo seus bits acionados ou não). Note que o bit menos significativo do byte é impresso na parte superior do display, enquanto o bit mais significativo é impresso abaixo. Cada byte ocupa uma coluna do display.

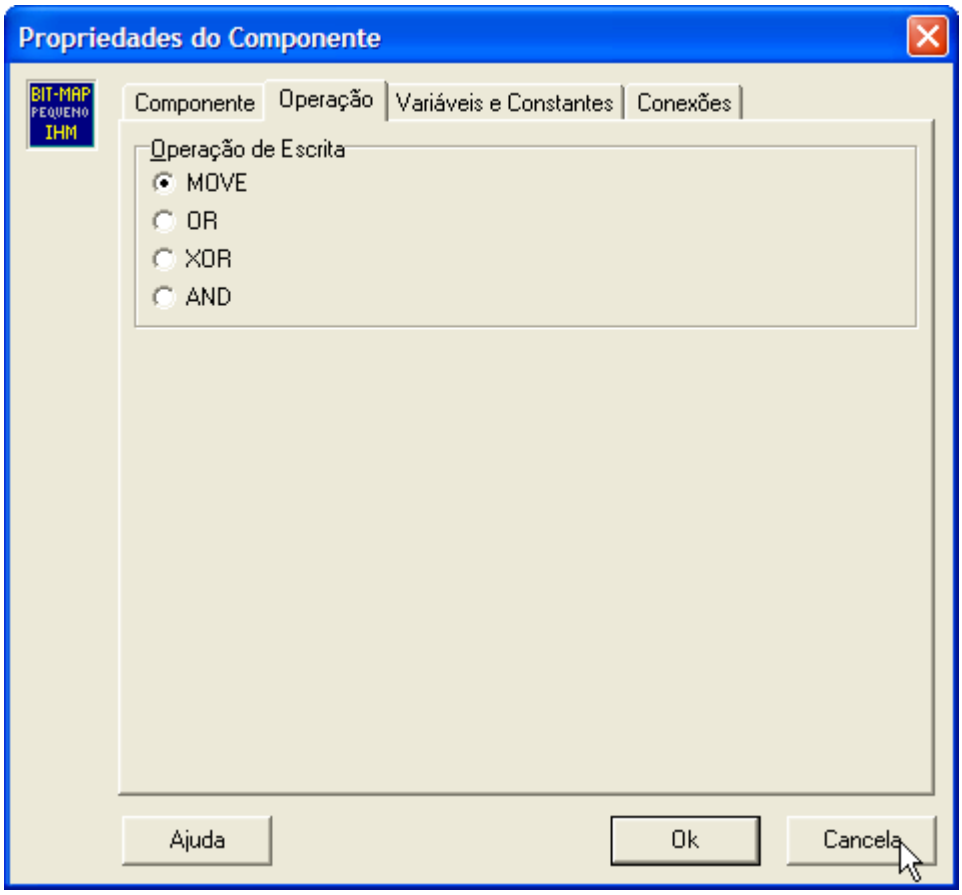
Tamanho → Número de bytes do bit map. Este parâmetro é expresso em um byte. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante.

X → Posição X para início da escrita no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 128 linhas, este valor pode variar entre 0 e 127.

Y → Posição Y para início da escrita no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 64 colunas, este valor pode variar entre 0 e 63.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, escrevendo o valor de **Operando** no display da IHM. Quando for concluída a escrita será gerado um pulso no nodo de Saída (S) do bloco. O nodo Inverte (I) permite reverter a escrita, tornando todos os pixels ligados em desligados e vice-versa.

Na edição deste bloco, além dos parâmetros acima, também é possível definir o tipo de operação a ser efetuada no display da IHM. Assim, existem as seguintes opções:



A operação **MOVE** é a padrão, e permite sobreescrever a área acessada pelo bloco no display da IHM, apagando os dados anteriores. Ou seja, independentemente de quais pixels estivessem ligados na área acessada pelo bloco, todos eles serão substituídos pelos novos dados.

A operação **OR** escreve preservando todos os dados previamente existentes no display. Portanto, todos os pixels que já estivessem ligados na área acessada pelo bloco serão mantidos ligados. Com isso os dados já existentes no display são mantidos.

A operação **XOR** reverte os dados a serem escritos sempre que eles forem ocupar uma posição já ativa do display. Então, todos os pixels ligados a serem escritos sobre pixels já acionados serão invertidos, ou seja, os pixels destas posições serão desligados. Isso permite identificar dados sobrepostos através da imagem invertida formada.

A operação **AND** apenas aciona os novos dados a serem escritos em regiões já ativas do display. Na verdade, esta opção é útil para mascarar certas regiões do display, permitindo dados apenas em áreas cujos pixels já estejam ligados.

Atenção: Bloco disponível apenas em controlador µDX201.

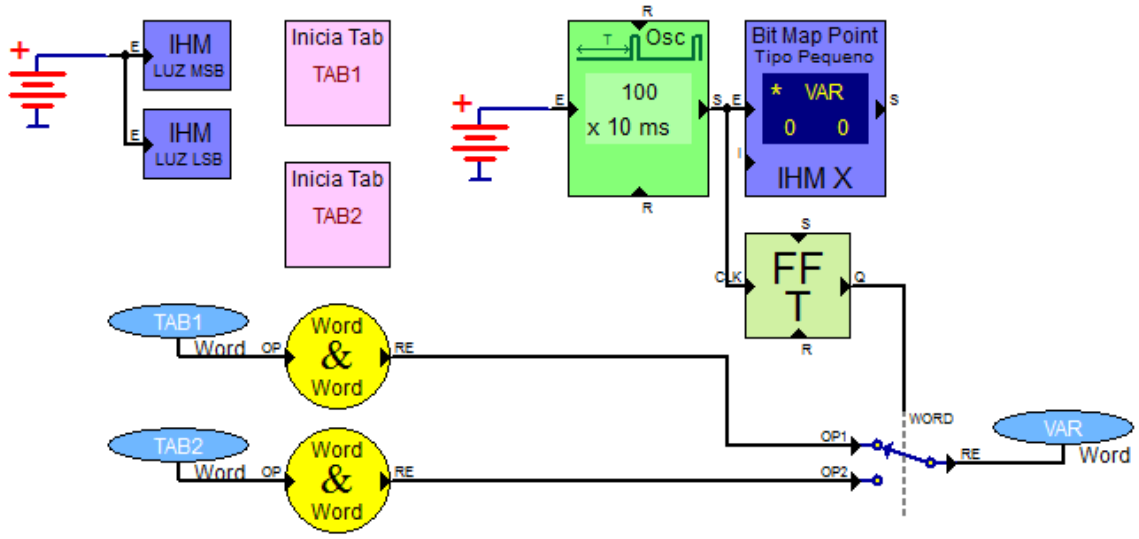
Blocos IHM - Escreve Bit Map X

A partir da versão **2.2.2.1** do software Programador Gráfico **PG** para µDX200 estão disponíveis os blocos **IHM X**, que possuem a mesma funcionalidade de seus equivalentes blocos de **IHM**, mas permitem que todos os parâmetros do bloco sejam informados através de variáveis. Os blocos de **IHM** originais possuem alguns parâmetros em formato byte, o que não permitia o uso de uma variável para modificar seus valores. No caso de escrita de bit map, o parâmetro **Tamanho** (que determina o total de bytes ou words do bit map a serem escritos no display) era tipo byte nos

blocos de **IHM**. Já nos blocos **IHM X** esse parâmetro é do tipo inteiro e é chamado **Comprimento**, podendo receber uma constante ou uma variável. Para usar os blocos **IHM X** é necessário controlador programável **µDX201** com firmware versão **3.53** ou superior.

Blocos IHM - Escreve Bit Map Pointer X

Este bloco é idêntico ao anterior, exceto que **Operando** é um pointer, ou seja, uma variável cujo conteúdo é o endereço do bit map a ser escrito. Com isso, pode-se mudar o bit map simplesmente modificando para qual tabela o pointer está apontando. O exemplo a seguir exemplifica essa facilidade. Ele escreve alternadamente na IHM, os bit maps armazenados nas tabelas **TAB1** e **TAB2**. Para isso os endereços de cada tabela são transferidos para a variável **VAR**, sendo que a cada 1 segundo a variável **VAR** aponta para a outra tabela. Com isso, 0 bit map na tela da IHM fica alternando, já que os blocos de IHM usam **Operando** como um pointer para obter o endereço do bit map a ser escrito.

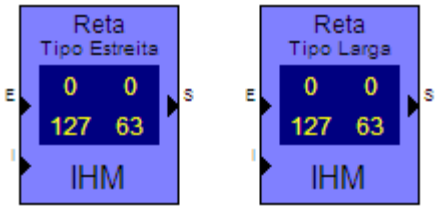


Exemplo de Programa Aplicativo: **Teste_IHM_Pointer2.dxg**

Veja também:
IHM - Escreve Inteiro
IHM - Escreve LongInt
IHM - Escreve String
Macros que acompanham o PG

IHM - Desenha Reta

Permite desenhar segmentos de reta no display da Interface Homem/Máquina (IHM) para µDX201. Existem dois blocos, um que desenha segmentos estreitos (1 pixel de largura), e outro que desenha segmentos largos (2 pixels de largura). Caso a coordenada inicial seja igual a coordenada final este bloco desenha apenas um ponto.



Os parâmetros destes blocos são os seguintes:

X1 → Posição X para início do segmento de reta a ser escrito no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 128 linhas, este valor pode variar entre 0 e 127.

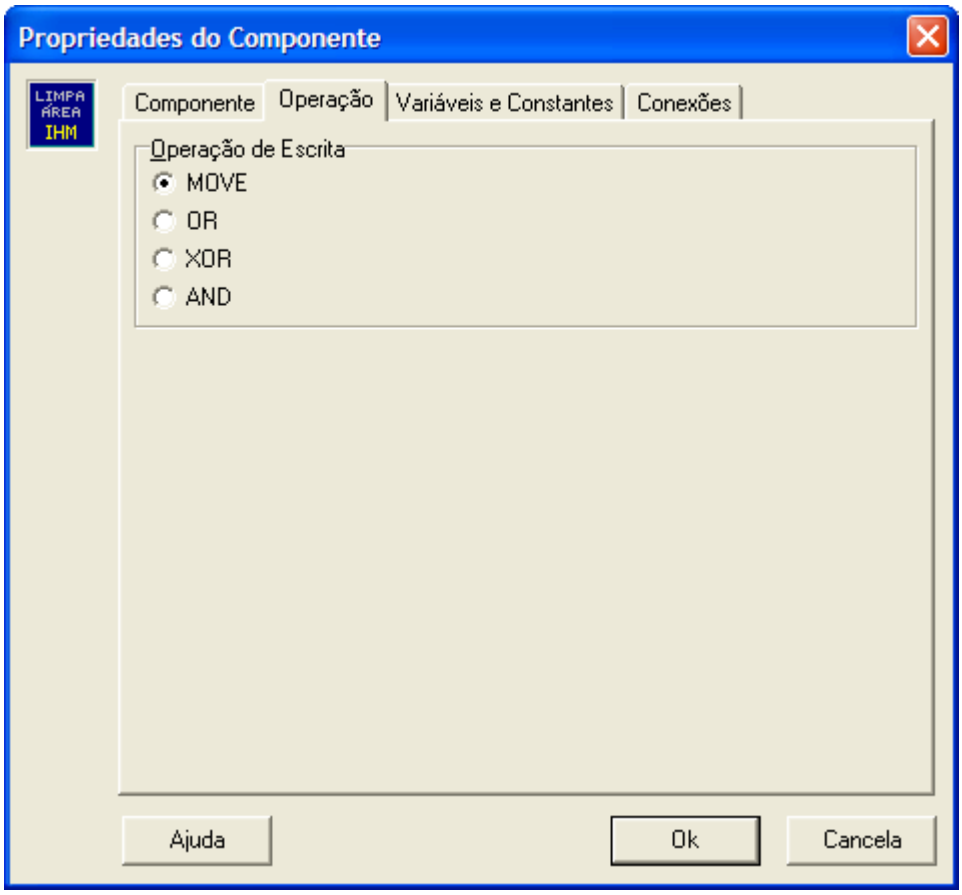
Y1 → Posição Y para início do segmento de reta a ser escrito no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 64 colunas, este valor pode variar entre 0 e 63.

X2 → Posição X para fim do segmento de reta a ser escrito no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 128 linhas, este valor pode variar entre 0 e 127.

Y2 → Posição Y para fim do segmento de reta a ser escrito no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 64 colunas, este valor pode variar entre 0 e 63.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, desenhando o segmento de reta entre (x1,y1) e (x2,y2) no display da IHM. Quando for concluída a escrita será gerado um pulso no nodo de Saída (S) do bloco. O nodo Inverte (I) permite reverter a escrita, tornando todos os pixels ligados em desligados e vice-versa.

Na edição deste bloco, além dos parâmetros acima, também é possível definir o tipo de operação a ser efetuada no display da IHM. Assim, existem as seguintes opções:



A operação **MOVE** é a padrão, e permite sobreescrever a área acessada pelo bloco no display da IHM, apagando os dados anteriores. Ou seja, independentemente de quais pixels estivessem ligados na área acessada pelo bloco, todos eles serão substituídos pelos novos dados.

A operação **OR** escreve preservando todos os dados previamente existentes no display. Portanto, todos os pixels que já estivessem ligados na área acessada pelo bloco serão mantidos ligados. Com isso os dados já existentes no display são mantidos.

A operação **XOR** reverte os dados a serem escritos sempre que eles forem ocupar uma posição já ativa do display. Então, todos os pixels ligados a serem escritos sobre pixels já acionados serão invertidos, ou seja, os pixels destas posições serão desligados. Isso permite identificar dados sobrepostos através da imagem invertida formada.

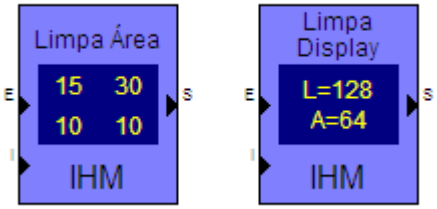
A operação **AND** apenas aciona os novos dados a serem escritos em regiões já ativas do display. Na verdade, esta opção é útil para mascarar certas regiões do display, permitindo dados apenas em áreas cujos pixels já estejam ligados.

Atenção: Bloco disponível apenas em controlador µDX201.

Veja também:
Macros que acompanham o PG

IHM - Limpa Área

Permite limpar áreas do display da Interface Homem/Máquina (IHM) para µDX201. Existem dois blocos, um que limpa uma área definida por certa largura e altura, e outro que limpa todo o display.



Os parâmetros do bloco de **Limpa Área** são os seguintes:

Largura → Largura em número de pixels a serem limpos. Este parâmetro é expresso em um byte. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. O valor deve estar entre 1 e 128.

Altura → Altura em número de pixels a serem limpos. Este parâmetro é expresso em um byte. Como o µDX200 não trata com variáveis do tipo byte o valor especificado neste campo deve ser, necessariamente, uma constante. O valor deve ser entre 1 e 64.

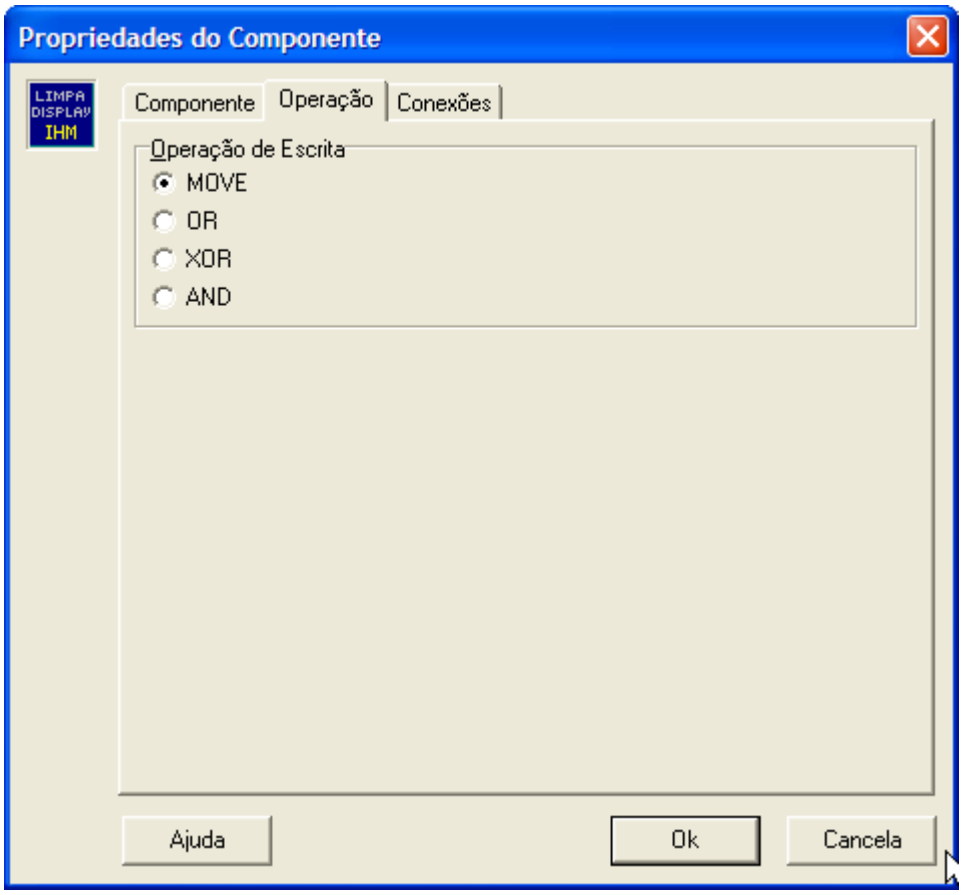
X → Posição X para início da área a ser limpa no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 128 linhas, este valor pode variar entre 0 e 127.

Y → Posição Y para início da área a ser limpa no display. Pode ser uma variável inteira ou word, uma constante (valores numéricos em decimal ou hexadecimal), ou ainda uma referência a uma posição de memória. Como o display possui 64 colunas, este valor pode variar entre 0 e 63.

O bloco de **Limpa Display** nada mais é do que o bloco de **Limpa Área** com largura=128, altura=64, X=0, Y=0.

Quando o nodo de Entrada (E) é energizado momentaneamente o bloco é ativado, limpando a área no display da IHM. Quando for concluída a escrita será gerado um pulso no nodo de Saída (S) do bloco. O nodo Inverte (I) permite reverter a escrita, tornando todos os pixels ligados em desligados e vice-versa.

Na edição deste bloco, além dos parâmetros acima, também é possível definir o tipo de operação a ser efetuada no display da IHM. Assim, existem as seguintes opções:



A operação **MOVE** é a padrão, e permite sobreescrever a área acessada pelo bloco no display da IHM, apagando os dados anteriores. Ou seja, independentemente de quais pixels estivessem ligados na área acessada pelo bloco, todos eles serão substituídos pelos novos dados.

A operação **OR** escreve preservando todos os dados previamente existentes no display. Portanto, todos os pixels que já estivessem ligados na área acessada pelo bloco serão mantidos ligados. Com isso os dados já existentes no display são mantidos.

A operação **XOR** reverte os dados a serem escritos sempre que eles forem ocupar uma posição já ativa do display. Então, todos os pixels ligados a serem escritos sobre pixels já acionados serão invertidos, ou seja, os pixels destas posições serão desligados. Isso permite identificar dados sobrepostos através da imagem invertida formada.

A operação **AND** apenas aciona os novos dados a serem escritos em regiões já ativas do display. Na verdade, esta opção é útil para mascarar certas regiões do display, permitindo dados apenas em áreas cujos pixels já estejam ligados.

Atenção: Bloco disponível apenas em controlador µDX201.

Veja também:
Macros que acompanham o PG

IHM - Buzzer

Este bloco permite acionar o buzzer existente na Interface Homem/Máquina (IHM) para µDX201, bastando para isso energizar o nodo de entrada do mesmo.

Esta funcionalidade é importante para fornecer uma realimentação (feedback) ao usuário ao pressionar o sensor touchscreen existente no display da IHM. Isso porque este tipo de sensor não gera uma sensação táctil ao ser pressionado, como um teclado eletromecânico, por exemplo. Este bloco equivale a acionar o nodo absoluto N21.

Atenção: Bloco disponível apenas em controlador µDX201.

Veja também:
Macros que acompanham o PG

IHM - Saída

Este bloco aciona a saída existente na Interface Homem/Máquina (IHM) para µDX201, bastando para isso energizar o nodo de entrada do mesmo.

Esta saída, normalmente, está em aberto. Ao energizar o bloco a saída é ligada ao terra lógico (GND), permitindo acionar pequenas cargas como lâmpadas ou solenóides (saída NPN). O limite de corrente é de 100mA, e a tensão que pode ser comutada deve ser, no máximo, 30V. Este bloco equivale a acionar o nodo absoluto N24.

Atenção: Bloco disponível apenas em controlador µDX201.

Veja também:
Macros que acompanham o PG

IHM - Backlight

Estes blocos acionam a luz de fundo (backlight) do display da Interface Homem/Máquina (IHM) para µDX201, bastando para isso energizar os nodos de entrada dos blocos. O brilho é determinado pela combinação binária dos blocos, ou seja, o valor 00 corresponde ao display com luz desativada, enquanto o valor 11 corresponde ao brilho máximo.

A combinação destes dois blocos é a seguinte:

Brilho	Luz MSB	Luz LSB
Desligado	0	0
Baixo	0	1
Médio	1	0
Alto	1	1

Estes blocos equivalem a acionar os nodos absolutos N23 (Luz MSB) e N22 (Luz LSB). Só são operacionais em controladores µDX201.

Atenção: Bloco disponível apenas em controlador µDX201.

Veja também:
Macros que acompanham o PG

Parte

Manutenção

O não funcionamento correto de qualquer uma das partes do Controlador Programável µDX Série 200 (seja o próprio µDX200, o cabo de comunicação ou o programa) deverá ser comunicado diretamente à DEXTER.

Evite qualquer tentativa de conserto, adaptação ou configuração que não tenha sido cuidadosamente abordada neste manual.

A DEXTER não se responsabiliza pelo uso indevido ou incorreto do µDX200 ou das partes que o acompanham.

Leia este manual com atenção antes de energizar o µDX200.

Parte

Garantia

A DEXTER oferece uma garantia de 1 (um) ano, a contar da data da compra, para reposição ou conserto do todo ou das partes do Controlador Programável µDX200 no caso de mau funcionamento ou defeitos originários na fábrica.

Esta garantia deixa de vigorar caso o defeito apresentado for resultante do uso indevido ou incorreto do todo ou das partes do µDX200, assim como no caso de serem feitas alterações de qualquer espécie em qualquer das partes do µDX200, sem autorização por escrito da DEXTER.

A DEXTER garante que o CD que acompanha o µDX200 está isento de contaminação pelos vírus de computador conhecidos até a data de fabricação.

Não estão incluídos nesta garantia os custos com transporte do µDX200 ou de suas partes, tanto para recebimento como para devolução.

Esta garantia se restringe ao controlador programável µDX200, não se estendendo ao processo controlado, nem a sensores e/ou acionamentos ligados ao controlador. O bom funcionamento do µDX200 pressupõe uma linha de alimentação sem ruídos e, no caso de acionamento de cargas indutivas, a instalação de supressores de ruído.

A DEXTER não se responsabiliza pela aplicação do µDX200 em processos perigosos ou de risco de vida.

DEXTER Indústria e Comércio de Equipamentos Eletrônicos Ltda.
Av. Pernambuco, 1328, cjs. 307/309 - CEP:90240-001 - Porto Alegre - RS
Fone: (51) 3208-0533 - Celular: (51) 99963-0370
Página Internet: http://www.dexter.ind.br
E-mail: dexter@dexter.ind.br

Índice

- _ -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- K -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- X -

- Z -