# Exemplos de Aplicações do Controlador Programável µDX200

## Comunicação entre µDX100 e µDX200

Pode ser necessário comunicar controladores da linha µDX100 com o controlador µDX200. No caso de ser necessária apenas uma comunicação unidirecional no sentido µDX100 para µDX200 é possível usar-se os blocos **RS232-RX DXNET µDX100** e **RS232-RX DXNET+ µDX100** para isso. Estes blocos assumem que o µDX200 irá comunicar-se via porta serial com protocolo DXNET ou DXNET+ do µDX100. Como o µDX100 possui apenas blocos de comunicação para escrita de nodos e variáveis em outros dispositivos, mas não leitura de dados destes dispositivos, a comunicação é limitada a transmissão de dados do µDX100 para o µDX200. Ou seja, o µDX100 é o mestre (quem origina as comunicações) e o µDX200 recebe os dados (no caso, variáveis de 8 bits). O protocolos DXNET ou DXNET+ implementados nestes blocos do µDX200 comportam apenas comandos de escrita e leitura de variáveis. Maiores detalhes podem ser obtidos nas páginas do Manual de Utilização do µDX200 que descrevem estes blocos.
A figura 1 mostra o programa **Figura01_comunicação_µDX100_µDX200.dxg**, que utiliza esta técnica. Note que o parâmetro 3006h atribuído ao bloco de comunicação RS232 indica 9600bps, sem paridade, 8 bits e 2 stops bits. Como as variáveis no µDX200 são 16 bits e as do µDX100 são de 8 bits, é necessário que o µDX100 sempre escreva na variável*2 do µDX200. Por exemplo, para escrever na variável v20 do µDX200 o µDX100 deve transmitir para a variável v40 no endereço DXNET do µDX200. Caso ele transmita para a variável v41 irá acessar o byte mais significativo (MSB) da variável v20 do µDX200. Também o endereço do µDX200 é composto de 2 bytes (16 bits), mas para fins de endereçamento via protocolo DXNET ou DXNET+ apenas o byte menos significativo (LSB) é relevante. Na aplicação da figura 1 o µDX100 pode comandar as 6 saídas analógicas do µDX200 escrevendo nas variáveis correspondentes (v20 até v25) do CLP. Como o µDX100 transmite apenas 8 bits e as saídas analógicas possuem resolução de 12 bits, o valor recebido é multiplicado por 16, de forma a atingir toda a faixa de atuação da saída analógica (0-10V ou 0-20mA).

Muitas vezes é necessário não somente escrever variáveis do µDX100 para o µDX200, mas também o contrário, ou seja, transferir o valor de variáveis do µDX200 para o µDX100. Neste caso, o controlador µDX200 é que deve assumir o papel de mestre (quem origina as comunicações). Para isso devemos fazer uso da flexibilidade do µDX200 em gerar protocolos pela porta serial. Na verdade, não é difícil implementar um protocolo qualquer na porta serial via programa aplicativo do CLP.
A figura 2 mostra o programa **Figura02_comunicação_µDX100_µDX200.dxg**, que demonstra esta possibilidade implementando duas instruções do protocolo DXNET+ via blocos RS232. Estas instruções são escrita de variável (comando 7) e leitura de variável (comando 5). Para complicar um pouco, o µDX200 acessa o µDX100 no endereço dxnet 1, conjunto 1 (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). O uso de conjunto atrapalha um pouco a geração do checksum final, já que o start byte do protocolo DXNET+ possui o nibble superior igual a Fh (15), e o nibble inferior é o conjunto do µDX100. Ocorre que o conjunto deve entrar no cálculo de checksum (FCS - Frame Check Sequence), mas a parte superior (Fh) não. Isso obrigou calcular o FCS via programa aplicativo, em vez de usar o bloco **RS232-TX FCS**. O programa, para ler a variável v10 do µDX100, coloca no buffer serial o start byte (F1h, já que acessa o conjunto 1), um byte de contagem, o comando 51h (comando 5 para endereço DXNET 1) e, por fim, o número da variável a ser lida (0Ah=10). Para terminar a mensagem falta apenas o checksum. Então, se faz a variável Index assumir o endereço de início de leitura do buffer de transmissão serial (_TXRDPT). A seguir, se incrementa Index (já que _TXRDPT aponta para o endereço imediatamente anterior ao primeiro byte a ser transmitido - veja maiores detalhes no Manual de Utilização do µDX200, capítulo Elaborando Programas - Convenções do Compilador PG) e se faz um AND com 3Fh (lembre-se que o buffer serial é circular e possui 64 bytes; logo Index deve retornar ao mesmo valor após 64 incrementos).

A próxima etapa é ler o valor armazenado no buffer serial e fazer um AND com 0Fh para desprezar o nibble superior do start byte (só levar em consideração o conjunto). Se faz o mesmo com os demais bytes armazenados no buffer serial, mas sem o AND com 0Fh (já que os demais bytes devem entrar no checksum por inteiro). Repare que Index é incrementado por dois logo após a leitura do start byte para não ler o contador, já que este não entra no cálculo de checksum. Por fim, se faz checksum = 100h - (checksum AND FFh) (veja detalhes do protocolo no documento "Comunicação Serial para µDX Plus"). Uma vez transmitido o comando o µDX200 aguarda 3 bytes de resposta, verifica o FCS (checksum da resposta), verifica se o byte de contagem confere com o transmitido (garantindo que a resposta se refere a esta pergunta - assinatura da mensagem) e, se tudo estiver ok, o dado recebido é armazenado em v18 do µDX200. A transmissão de variável do µDX200 para o µDX100 ao acionar a entrada E1 da Expansão µDX210 funciona de forma similar, e liga a saída S2 desta Expansão momentaneamente caso a operação seja concluída com sucesso.

O programa **Figura03_comunicação_µDX100_µDX200.dxg** demonstra como ficaria o programa caso o µDX100 estivesse no conjunto 0 e, portanto, fosse possível usar o cálculo automático de checksum. Note que há uma considerável simplificação na transmissão dos comandos. Já a recepção dos dados é idêntica, ou seja, é verificado o checksum e a assinatura da mensagem.

Obviamente, como o µDX100 não possui porta serial, o controlador µDX200 deve ser conectado a porta serial de um Modem para µDX100, e este conectado ao controlador µDX100 via rede DXNET. O baud rate do Modem deve ser compatível com o utilizado no µDX200, e o protocolo programado no Modem para µDX100 deve ser DXNET+. Um exemplo final de comunicação é o programa **Figura04_comunicação_µDX100_µDX200.dxg**. Este programa transmite para o µDX100 o valor de 8 sensores de temperatura e 2 sensores de umidade. O µDX200 transmite os dados para os seguintes endereços:

- Sensor de Temperatura 1: Variável v20, µDX100 endereço DXNET 2.
- Sensor de Temperatura 2: Variável v21, µDX100 endereço DXNET 2.
- Sensor de Temperatura 3: Variável v20, µDX100 endereço DXNET 3.
- Sensor de Temperatura 4: Variável v22, µDX100 endereço DXNET 2.
- Sensor de Temperatura 5: Variável v21, µDX100 endereço DXNET 3.
- Sensor de Temperatura 6: Variável v20, µDX100 endereço DXNET 4.
- Sensor de Temperatura 7: Variável v20, µDX100 endereço DXNET 5.
- Sensor de Temperatura 8: Variável v21, µDX100 endereço DXNET 5.
- Sensor de Umidade 1: Variável v21, µDX100 endereço DXNET 4.
- Sensor de Umidade 2: Variável v22, µDX100 endereço DXNET 5.

Cada dado é transmitido seqüencialmente, a cada 200ms, graças a máquina de estados comandada pela variável var. Uma última observação: note que as temperaturas são deslocadas em 7 bits (shift right) porque a leitura de temperatura, com o bloco usado no programa, deve ser dividido por 128 para se obter a temperatura em graus celsius, com resolução de 0,5°C. Já a umidade não deve ser deslocada.

## Teste de Expansão µDX210

O programa **Figura05_teste_µDX210.dxg** simplesmente replica o que houver nas entradas digitais da Expansão µDX210 para as respectivas saídas desta Expansão. Note que as entradas do µDX210 são fornecidas de fábrica com os jumpers internos para alta tensão (>40V). Também é possível acionar as entradas clicando com o mouse sobre os "leds" do µDX210 representado no Compilador PG.

Outro exemplo é o programa **Figura06_varre_µDX210.dxg**, que liga em seqüência as saídas da Expansão µDX210 endereço 1. Além disso, todas as saídas analógicas (S1 a S6) são

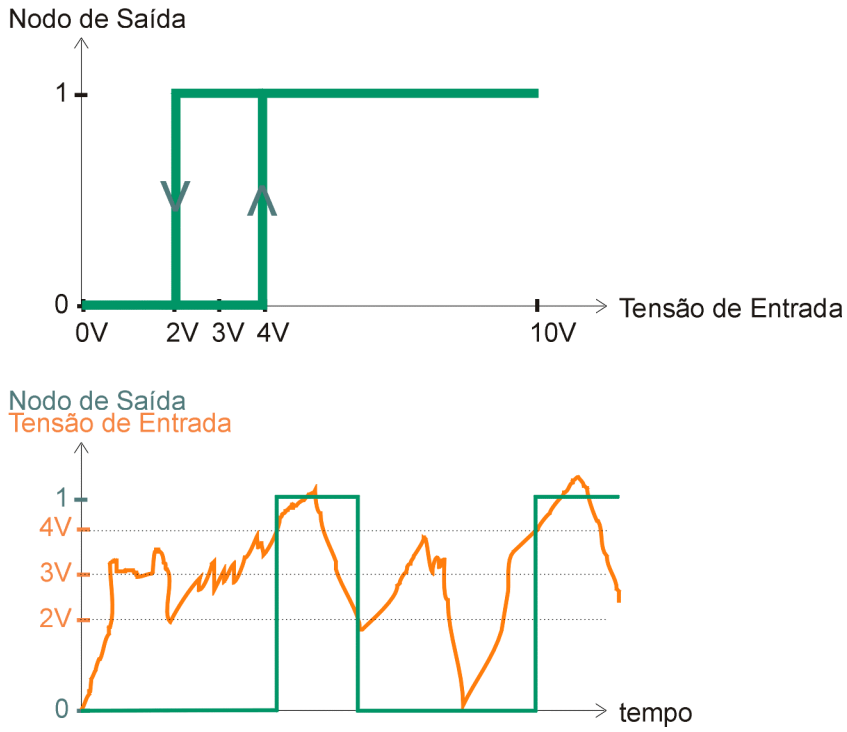
colocadas no máximo valor (4095). Ao ligar o nodo INIBE todas as saídas analógicas retornam ao mínimo valor (0). Por fim, o µDX200 lê um sensor de temperatura na rede $I^2C$ e coloca o valor lido na variável Temperatura.

## Máquina de Estados

A técnica de programação via máquina de estados é muito poderosa para implementar programas complexos de maneira organizada e clara.
O exemplo **Figura07_maquina_estados.dxg** demonstra o uso desta abordagem para implementar um pequeno programa fictício. Ao acionar a entrada E1 da Expansão µDX210 o programa pula para o estado 1 (atribuindo valor 1 para a variável Maquina). A seguir, cada estado é mantido por 3 segundos até retornar ao estado 0. Se a entrada E1 ainda estiver acionada um novo ciclo é iniciado. Já a entrada E2 permite congelar a máquina de estados no estado atual (funciona como uma entrada de pausa) e a entrada E3 reinicializa a máquina de estados.

## Uso de Entradas e Saídas Analógicas como Entradas e Saídas Digitais

As vezes pode ser necessário usar as entradas e saídas analógicas do µDX200 como entradas e saídas digitais, evitando o uso de uma Expansão µDX210.
O exemplo **Figura08_IO_digital.dxg** ilustra como isso pode ser feito. Para a entrada analógica foi utilizado um bloco de histerese com ponto de comparação em 1229 e histerese de 410. Ora, se considerarmos a entrada analógica na escala de 0-10V (os µDX200 são fornecidos de fábrica com todas as entradas e saídas analógicas na escala 0-10V) este ponto de decisão corresponde a 1229*10V/4095 = 3V. Já a histerese é de 410*10V/4095 = ±1V. Portanto, o nodo de saída do bloco de comparação com histerese irá ligar sempre que a entrada analógica for excitada com tensão igual ou superior a 4V, e irá desligar sempre que na entrada analógica houver 2V ou menos. A histerese evita que surjam comutações indesejadas quando o sinal de entrada estiver em valor próximo ao ponto de decisão e tiver associado a ele algum nível de ruído. A figura abaixo esclarece melhor este aspecto:




Já o acionamento da saída analógica é feito comutando-se a mesma entre seus valores máximo (4095) e mínimo (0), o que faz com que a saída analógica assuma 10V ou 0V, respectivamente. Note que foi utilizado um seletor de variável inteira para efetuar esta comutação.

## Multiplexador para Pulsadores via Entradas Analógicas

O programa a seguir exemplifica o uso do Multiplexador no programa aplicativo do µDX200. Note que é feita a comparação do valor na entrada analógica E1, de forma a discernir qual pulsador foi ativado (lembre-se que a variável v0 está sempre associada a entrada analógica E1 no µDX200). Os pontos de comparação possuem uma margem de ±150 divisões. Isso corresponde a ±150/4095*20mA = ±0,73mA de tolerância na medida de corrente. Os pontos de decisão são calculados da mesma forma:

| | | |
|---|---|---|
| 2mA | → | 02/20*4095 = 409,5 ≈ 410 |
| 4mA | → | 04/20*4095 = 819 |
| 6mA | → | 06/20*4095 = 1228,5 ≈ 1229 |
| 8mA | → | 08/20*4095 = 1638 |
| 10mA | → | 10/20*4095 = 2047,5 ≈ 2048 |
| 12mA | → | 12/20*4095 = 2457 |
| 14mA | → | 14/20*4095 = 2866,5 ≈ 2867 |
| 16mA | → | 16/20*4095 = 3276 |
| 18mA | → | 18/20*4095 = 3685,5 ≈ 3686 |
| 20mA | → | 20/20*4095 = 4095 |

Um último comentário a respeito deste programa é referente aos blocos de atraso e comparação existentes para energizar a discriminação do pulsador. Note que, constantemente, a variável v0 é transferida para a variável temp, e o valor desta variável é comparada com v0 dentro de uma margem de ±10 divisões. Isso serve para garantir que a corrente na entrada E1 estabilizou em um valor fixo (dentro de ±50µA) e, com isso, energizar a entrada do bloco de Atraso. Este bloco exige que a entrada permaneça estável por 100ms antes de iniciar a análise da corrente de entrada. Isso evita que haja transitórios ao pressionar algum pulsador. O programa liga as saídas da Expansão de Entrada/Saída µDX210 para sinalizar o pulsador pressionado. Lembre-se que a entrada analógica do µDX200 utilizada para leitura do Multiplexador deve estar na escala 0-20mA. Para isso é necessário comutar os jumpers internos do CLP, e também especificar escala 0-20mA na Janela de Configuração de Hardware do Compilador PG. O programa está salvo como **Figura09_Multiplexador.dxg**.

## Keypad para Controlador µDX200 via rede $I^2C$

O programa **Figura10_Keypad.dxg** 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 100ms para transmitir o valor detectado para a variável "saída", que irá acionar as saídas da Expansão µDX210 e também os leds do Keypad.

## Expansão de Saídas µDX212 via rede $I^2C$

A Expansão µDX212 permite acionar cargas de alta potência à distância, já que é conectada ao µDX200 via rede $I^2C$. Para sua utilização existe um bloco chamado **Saídas I2C** no Editor PG. Este bloco permite endereçar até 8 Expansões µDX212 (endereços de 0 a 7) em um único Controlador µDX200. O programa **Figura11_µDX212.dxg** disponibiliza 8 nodos (de Saida1 até Saida8) que acionam as saídas da Expansão µDX212.

## Dimmer via Saídas Analógicas do µDX200

Para utilizar as saídas analógicas do µDX200 como controle de Dimmers é necessário comutar os jumpers internos do µDX200 para saídas analógicas sem filtro (saídas PWM), e também colocar os jumpers de tipo de saída analógica em tensão de 0-10V. Veja o manual do Controlador µDX200 para detalhes do posicionamento destes jumpers.

Além disso, no programa aplicativo que será transmitido para o CLP deve constar que as saídas analógicas estão sendo usadas para controle de dimmer. Repare que não é possível usar algumas saídas analógicas como controle para dimmer e outras como saídas analógicas convencionais. Para especificar controle dimmer no programa aplicativo basta selecionar esta opção na janela de Configuração de Hardware existente no Compilador PG. Também deve-se escolher qual entrada rápida do µDX200 (E9 ou E10) será utilizada para conexão do Sensor de Zero.


Para acionamento do Dimmer basta modificar a saída correspondente. Para plena potência coloque valor 100 nesta saída, e para completo desligamento coloque valor 8400 em 60Hz, ou 10100 em 50Hz. Temos, portanto, 8300 divisões se a rede elétrica for com freqüência de 60Hz, e 10000 divisões caso a rede elétrica seja em 50Hz. O valor de plena carga deve ser 100 e não zero para que haja uma pequena margem no início do ciclo, pois pode existir uma diferença de fase entre tensão da rede elétrica e corrente na carga (potência reativa).
O programa de exemplo **Figura12_Dimmer.dxg** aciona uma carga via Dimmer, aumentando e diminuindo sua potência de forma cíclica. Note que v8 corresponde a saída S1 no µDX200. Os limites de variação de v8 foram fixados entre 1000 e 7000. Isso porque para valores abaixo de 1000 cargas como lâmpadas não apresentam diferenças visíveis de brilho (ou seja, já estão a pleno brilho), e com v8=7000 já apresentam o filamento totalmente sem brilho.
Além disso, o incremento é de 100 em 100 divisões, ou seja, foram fixadas 60 graduações de brilho. Como no incremento da variável (e. portanto, diminuição de brilho da lâmpada) o oscilador é de 2ms significa que o ciclo de brilho máximo até total desligamento é feito em 120ms. Já o ciclo de lâmpada desligada até máximo brilho é efetuado em 1,2s, já que o oscilador responsável pelo decremento da variável v8 possui período de 20ms.

## Dimmer via Módulos MCI conectados à rede DXNET

Outra possibilidade de controle proporcional de iluminação no µDX200 é através dos módulos de dimmer MCI da empresa RKM. Os módulos utilizam a rede DXNET do µDX200 para conexão ao controlador, e ocupam endereçamento na rede a partir do endereço 0800h. O programa **Figura13_MCI.dxg** utiliza dois nodos, um para escrever a variável teste no MCI (controlando assim o nível de iluminação) - nodo Envia; e outro para ler o valor da variável do MCI e colocar o valor lido em Valor. Note que a variável teste é inicializada com 100 ao iniciar o programa aplicativo do µDX200, que corresponde a plena potência no canal dimmerizado. Não esqueça de incluir o endereço 2048 (0800h) na lista de scan da rede DXNET (janela de configuração de hardware no Compilador PG), e transmitir esta tabela para o µDX200 (vá no menu Geral → Configura ScanTab ).


## Escrita de Dados no Cartão MMC

O programa **Figura14_MMC.dxg** exemplifica o salvamento de dados no cartão MMC. Foi utilizado um oscilador com período de 1 segundo, de forma que os dados sejam gravados a cada segundo. O oscilador aciona um flip-flop tipo D que mantêm sua saída ligada até que todo ciclo de gravação de dados no cartão MMC seja completado. Os dados gravados são: data e hora; string definido na tabela Teste; variável inteira Cont (incrementada a cada gravação); variável longint Var; e finalização de linha (Carriage Return + Line Feed). A saída S8 da Expansão µDX210 é acionada durante 0,5 segundo para indicar que os dados foram gravados. A entrada E9 do µDX200, quando acionada, inibe o oscilador, de forma a evitar novos ciclos de gravação, aguarda 200ms para garantir que o ciclo de gravação corrente se complete, e aciona um bloco de Salva Buffer. Lembre-se que os dados são gravados no cartão MMC somente após o buffer possuir 512 bytes. Portanto, se o cartão for retirado antes de completar o buffer simplesmente os dados serão perdidos. Para evitar isso deve-se acionar E9 antes da retirada do cartão MMC, garantindo que os dados armazenados no buffer de MMC serão salvos. A saída S7 indica que os dados foram salvos e o cartão pode ser retirado. As saídas S1, S2 e S3 indicam erros do cartão MMC (cartão ausente, cartão cheio, ou erro de cartão).

## Acionamento de Taps de Transformadores

Este exemplo demonstra como gerar uma máquina de estados um pouco mais complexa. Existem dois transformadores com derivações (taps) que devem ser mantidos sempre com a mesma posição de tap. Além disso, o µDX200 deve comandar ambos os transformadores para que subam ou desçam os taps. Para indicar qual o tap em que cada transformador se encontra existe um potenciômetro de 180 Ω. Cada transformador pode assumir 19 posições diferentes e, portanto, o potenciômetro irá apresentar uma diferença de 10 Ω entre cada tap.

A baixa resistência de cada tap mais a precisão requerida tornam soluções corriqueiras, como a colocação de fontes de tensão para excitar o circuito perigosa. Seria necessário acrescer uma resistência série para evitar o curto-circuito da fonte quando a resistência se aproximasse de 0 Ω. Ocorre que esta resistência implicaria em escala não-linear para a distribuição dos taps, uma vez que a tensão de saída seria dada por Vo = (Rtap*V) / (Rtap+Rs).

Assim, se obtivermos Vo para o tap 1 (180R), e considerando Rs=180R, para o tap 9 (90R), obteremos 0,67*Vo em vez de 0,5*Vo. Embora isso possa ser considerado no programa aplicativo, a distribuição não-linear dos intervalos dos taps torna o sistema menos tolerante a imprecisões (já que os valores dos taps irão ficar muito próximos para os taps de alta resistência). Uma solução interessante para isso é usar as saídas analógicas do µDX200 como fontes de corrente para excitar a leitura dos taps. O uso de fonte de corrente torna o sistema seguro, pois eventuais curtos não causarão dano. Além disso, permite suprimir a resistência série e, com isso, torna a distribuição de leitura de taps linear. Por fim, esta solução evita a necessidade de uma fonte de tensão externa de boa precisão.

Como as fontes de corrente do µDX200 possuem limite superior de 20mA, elas podem gerar 3,6V em uma resistência de 180R. Então, vou programar as saídasanalógicas para uma corrente inferior, de forma que gere 2,5V em 180R (assim posso usar as entradas analógicas do µDX200 na escala de 0-2,5V). Para isso basta que a fonte de corrente gere 13,89mA. Note que uma vantagem adicional desta solução é que ela permite, facilmente, adaptar-se a outras resistências, bastando modificar a corrente da saída analógica do µDX200.

Uma questão é referente ao início do sistema. E se ao ligar o CLP os transformadores já estiverem em posições distintas? Neste caso considerei que a posição do transformador ligado a entrada analógica E1 é o correto, e farei com que o ligado a entrada E2 se adapte a ele. Existem 19 posições possíveis para os taps. Então, considerando que 0V corresponde a 0 na variável associada a entrada analógica, e 2,5V corresponde a 4095 nesta variável, se fizermos 4095/18 = 227,5. Logo, os valores de comparação devem ser:

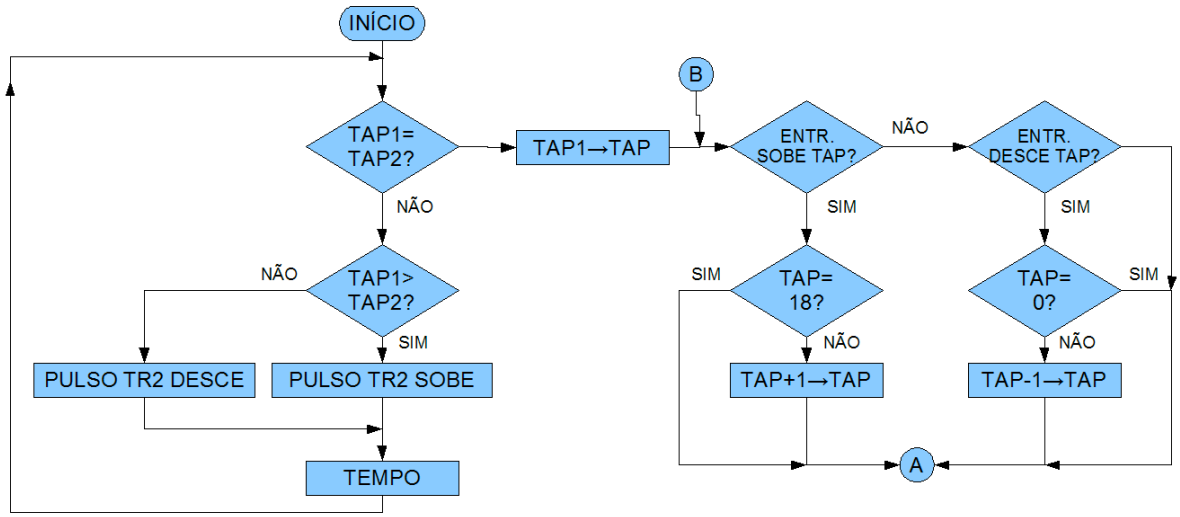
| **Tap** | **R** | **valor** |
|---|---|---|
| 19 | 0Ω | 0 |
| 18 | 10Ω | 228 |
| 17 | 20Ω | 455 |
| 16 | 30Ω | 683 |
| 15 | 40Ω | 910 |
| 14 | 50Ω | 1138 |
| 13 | 60Ω | 1365 |
| 12 | 70Ω | 1593 |
| 11 | 80Ω | 1820 |
| 10 | 90Ω | 2048 |
| 9 | 100Ω | 2275 |
| 8 | 110Ω | 2503 |
| 7 | 120Ω | 2730 |
| 6 | 130Ω | 2958 |
| 5 | 140Ω | 3185 |
| 4 | 150Ω | 3413 |
| 3 | 160Ω | 3640 |
| 2 | 170Ω | 3868 |
| 1 | 180Ω | 4095 |

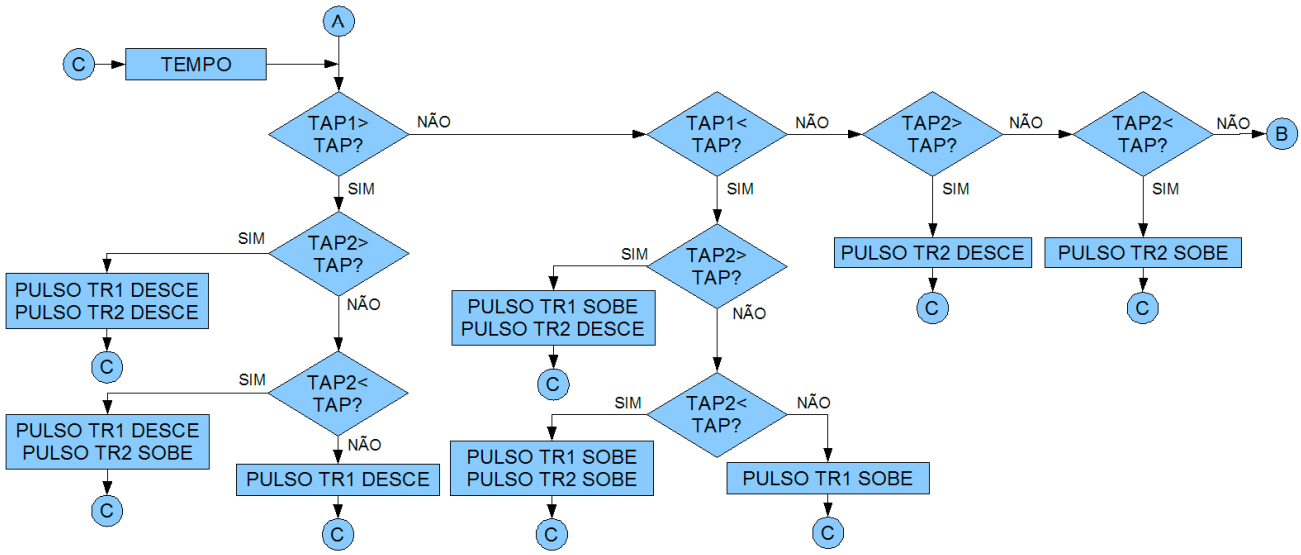
Cada intervalo é de 227,5. Portanto, posso colocar uma folga de 113 para cada lado da

comparação. Além disso, as saídas analógicas, em escala de 0-20mA, geram 0mA com valor 0 na variável associada a saída analógica, e 20mA com esta variável com valor 4095. Logo, para gerar 13,89mA preciso atribuir a esta variável o valor 2844. As entradas e saídas usadas são:

**µDX200**
E1: entrada analógica do tap1 (deve estar em escala de 0-2,5V).
E2: entrada analógica do tap2 (deve estar em escala de 0-2,5V).
S1: saída analógica para excitar tap1 (deve estar em escala 0-20mA).
S2: saída analógica para excitar tap2 (deve estar em escala 0-20mA).
**µDX210**
E1: sobe tap (deve estar em entrada digital para alta tensão AC).
E2: desce tap (deve estar em entrada digital para alta tensão AC).
S1: sobe tap1.
S2: desce tap1.
S3: sobe tap2.
S4: desce tap2.
S5: alarme de sincronismo.

O mais importante neste programa é a descrição da máquina de estados. Uma vez esta definida, seu translado para o Editor PG é muito simples e evidente. Na figura seguinte é reproduzido o fluxograma do projeto **Figura15_Taps.dxp**, que é formado pelas páginas de programação **Figura15_Taps1.dxg**, **Figura15_Taps2.dxg** e **Figura15_Taps3.dxg**.






As conexões dos sensores de taps dos transformadores ao µDX200 são descritas na figura seguinte. Note que usa-se uma fonte de corrente de 13,39mA (suprida pelas saídas analógicas do µDX200) para alimentar cada um dos potenciômetros dos transformadores.

## Decodificação do Relógio de Tempo Real

O programa **Figura16_RTC.dxg** demonstra como é possível ler o relógio de tempo real do µDX200 e colocar os dados em variáveis distintas. A cada segundo as variáveis internas _HORASEM, _SEGMIN, _DIAMES e _ANO são lidas. O formato destas variáveis é o seguinte (lembre-se que todas estas variáveis são inteiras, ou seja, possuem 16 bits):

| | |
|---|---|
| **_HORASEM** | Dia da semana (01h para Segunda, 07h para Domingo).<br>Hora em codificação BCD (00h até 23h).<br>Exemplo: 0115h para Segunda, 15 horas. |
| **_SEGMIN** | Minutos em codificação BCD (00h até 59h).<br>Segundos em codificação BCD (00h até 59h).<br>Exemplo: 2732h para 27 minutos e 32 segundos. |
| **_DIAMES** | Mês em codificação BCD (01h até 12h).<br>Dia em codificação BCD (01h até 31h).<br>Exemplo: 1206h para dezembro, dia 6. |
| **_ANO** | Ano em codificação BCD (0000h até 9999h).<br>Exemplo: 2008h para o ano de 2008. |

Ou seja, é preciso separar os dados das variáveis e também convertê-los de BCD para binário. A separação é facilmente implementada com funções AND (para isolar apenas os bits pertinentes para cada dado) e funções SRA (para deslocar 8 posições os dados que estão no byte MSB da variável). Mas ainda é preciso converter os dados de BCD para binário. A codificação BCD (binary coded decimal, ou seja, decimal representado em binário) representa o valor decimal diretamente em cada nibble (4 bits). Assim, por exemplo, o valor 78 é representado por 78h, ou seja, 0111 1000 em binário. Já 78 em binário seria 01001110 em binário (4Eh). Para converter de BCD para binário basta subtrair do valor de cada byte da variável inteira o valor inteiro da divisão deste byte por 16 multiplicado por 6. Em notação matemática:

$$\mathbf{V_{BIN} = V_{BCD} - INT(V_{BCD}/16)*6}$$

Um exemplo deve esclarecer o procedimento. Digamos que tenhamos na variável _ANO o valor hexadecimal 2008h (representando o ano de 2008). Vamos convertê-lo para binário. Note que é necessário converter cada um dos bytes da variável inteira separadamente:

$$V_{BIN\_MSB} = 20h - INT(20h/16)*6 = 32 - INT(2)*6 = 32 - 12 = 20$$
$$V_{BIN\_LSB} = 08h - INT(08h/16)*6 = 8 - INT(0)*6 = 8 - 0 = 8$$

O valor binário total será:

$$V_{BIN} = V_{BIN\_MSB} * 100 + V_{BIN\_LSB} = 20*100 + 8 = 2008 = 07D8h$$

O programa faz todas estas conversões a cada segundo e gera as variáveis HORA, MINUTO, SEGUNDO, DIA, MES e ANO com os valores do RTC (Real Time Clock) interno do µDX200. Foram colocados 6 testes, que comparam estas variáveis com as variáveis SETUP_HORA, SETUP_MINUTO, SETUP_SEGUNDO, SETUP_DIA, SETUP_MES e SETUP_ANO. Caso os valores coincidam são ligados os nodos N_HORA, N_MINUTO, N_SEGUNDO, N_DIA, N_MES e N_ANO.

## Controle PID

Acompanha estes exemplos uma macrocélula para controle **PID** (proporcional-integral-derivativo). Não vamos descrever a teoria envolvida neste tipo de controle, e junto ao desenho da macrocélula existem comentários a respeito do cálculo envolvido. O algoritmo é:

| 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) |
|---|

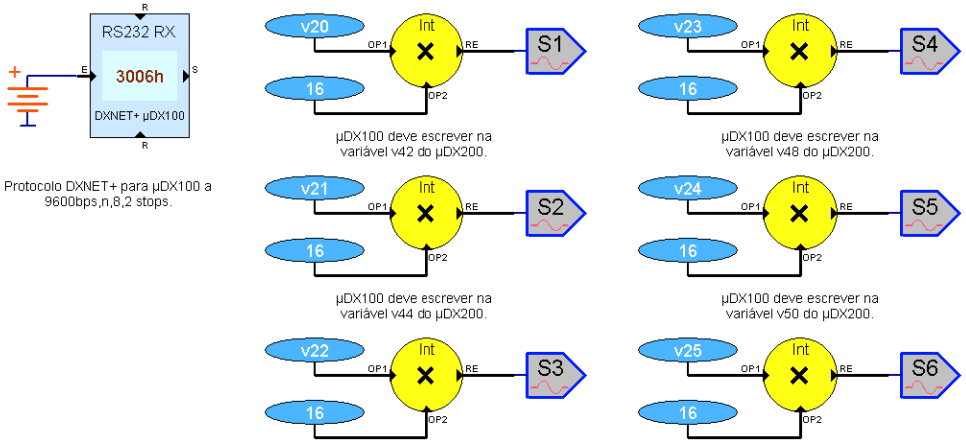
Note que os coeficientes Kp, Ki e Kd, assim como o período de cálculo T são determinados por parâmetros nas conexões de entrada da macrocélula PID. Também o valor desejado (setpoint) é determinado desta forma. No exemplo usou-se a entrada analógica E1 do µDX200 como parâmetro a ser controlado, e as saídas S1 e S2 como acionadores.
O programa **Teste_PID.dxg** utiliza esta macrocélula para exemplificar seu uso. Note que fixamos o período em 100 milisegundos, e os fatores Kp, Ki e Kd. O setpoint desejado é 2000, e a variável mensurada é obtida pela entrada analógica E1. As saídas do bloco são ligadas às saídas S1 e S2.
Para incluir a macrocélula na lista de blocos do programa PG basta copiar a macrocélula compilada (arquivo sufixo .DMB) no diretório correspondente a macros do PG (normalmente, c:\Arquivos de Programas\Dexter\PG\Macro). Esta macrocélula foi elaborada para permitir uma saída entre 4 e 20mA. Por isso, a variável de saída assume valores entre 819 e 4095. Evidentemente, a saída analógica utilizada deve ser programada para a escala 0-20mA (via Configuração de Hardware), e os jumpers correspondentes comutados no interior do µDX200.

## Automação Residencial e Predial

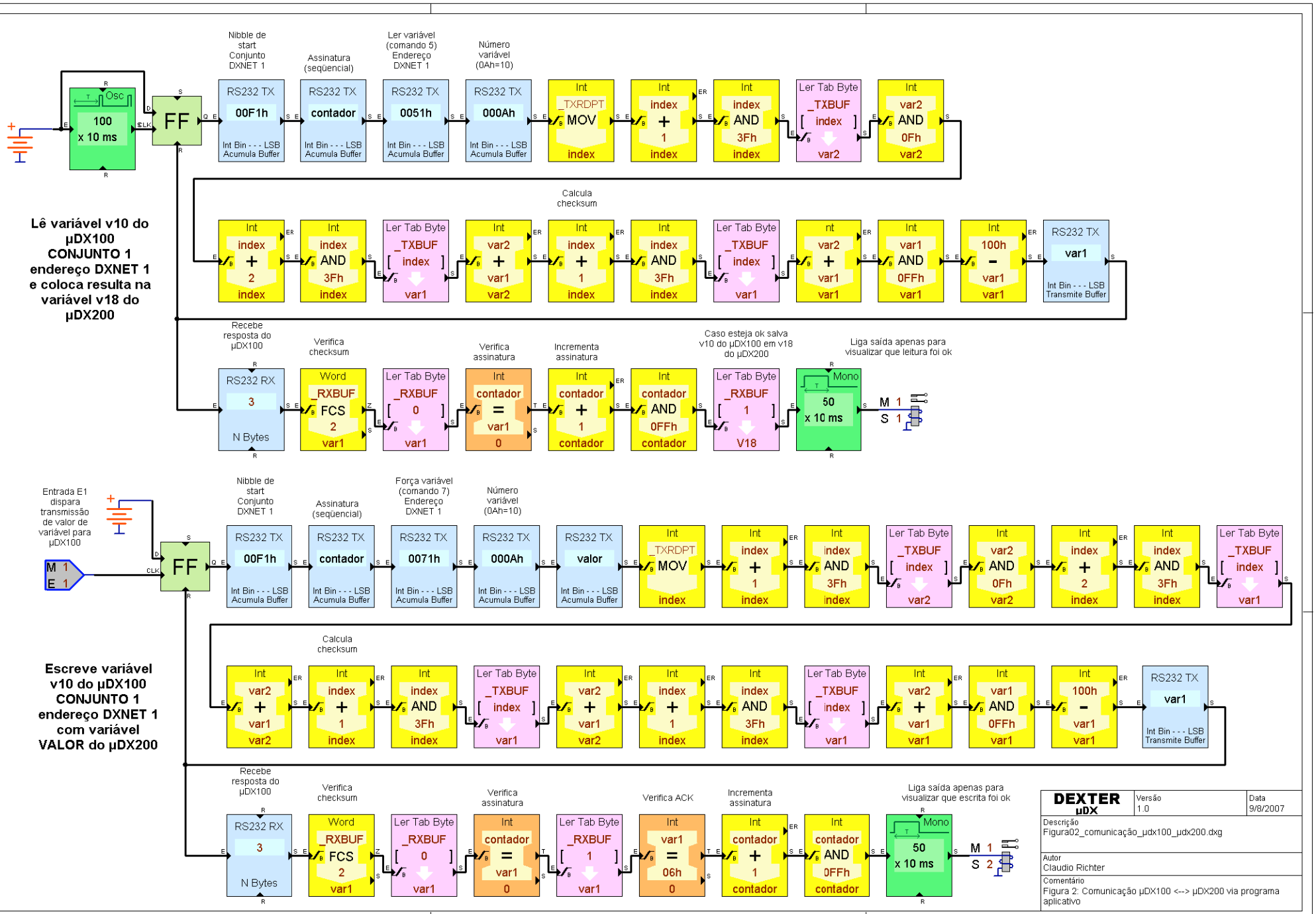
Foram geradas algumas macrocélulas que facilitam muito o uso dos módulos elaborados para o controlador µDX200 em aplicações residenciais e prediais. São elas:
**MUX:** permite a decodificação de Multiplexador para Pulsadores. A entrada desta macro deve ser ligada à entrada analógica conectada ao Multiplexador, e as saídas são nodos correspondentes aos valores decodificados. Na verdade, esta Macro corresponde a funcionalidade do programa **Figura09_Multiplexador.dxg**. Mas foi feita uma filtragem melhor dos valores lidos, de forma a evitar sinais espúrios nas saídas.
**Keypad:** esta macrocélula decodifica as 8 teclas de um Keypad. Deve-se usar na entrada um bloco de **I/O I2C (16 bits)**. Ela possui 8 saídas de nodos correspondendo as 8 teclas, e também uma saída de variável inteira com o valor lido via controle remoto infravermelho.
**IR:** possui uma entrada para variável inteira que, ao ser ligada à saída de leitura de controle remoto da macrocélula anterior, decodifica várias teclas do controle remoto em nodos.
**Dimmer:** esta macrocélula possui uma máquina de estados que permite controlar uma saída à dimmer do µDX200. Caso se energize o nodo IN por um tempo inferior ao especificado na entrada inteira Click (tempo este dado de 10 em 10ms) a saída comuta entre desligado e o último valor de dimmer. Caso se mantenha o nodo IN energizado tempo superior ao especificado na entrada inteira Click, o dimmer sobe ou desce a iluminação em passos determinados pela entrada inteira Step.
**uDX212:** possibilita acionar as 8 saídas à relé de um µDX212 via 8 nodos na entrada desta macrocélula. Note que o acionamento possui retenção, ou seja, um pulso no nodo liga a correspondente saída, e outro pulso a desliga.
Os programas **Teste_MUX.dxg**, **Teste_Keypad.dxg**, **Teste_Dimmer.dxg** e **Teste_uDX212.dxg** exercitam estas macrocélulas. Convém examiná-los com cuidado para obter maior familiaridade com as macrocélulas. Como os arquivos-fonte das mesmas também é fornecido (arquivos sufixos .DXM) é muito fácil modificá-las para se adaptarem melhor as necessidades de determinada aplicação.
Note que para incluir a macrocélula na lista de blocos do programa PG basta copiar a macrocélula compilada (arquivo sufixo .DMB) no diretório correspondente a macros do PG (normalmente, c:\Arquivos de Programas\Dexter\PG\Macro). As macrocélulas incluídas no

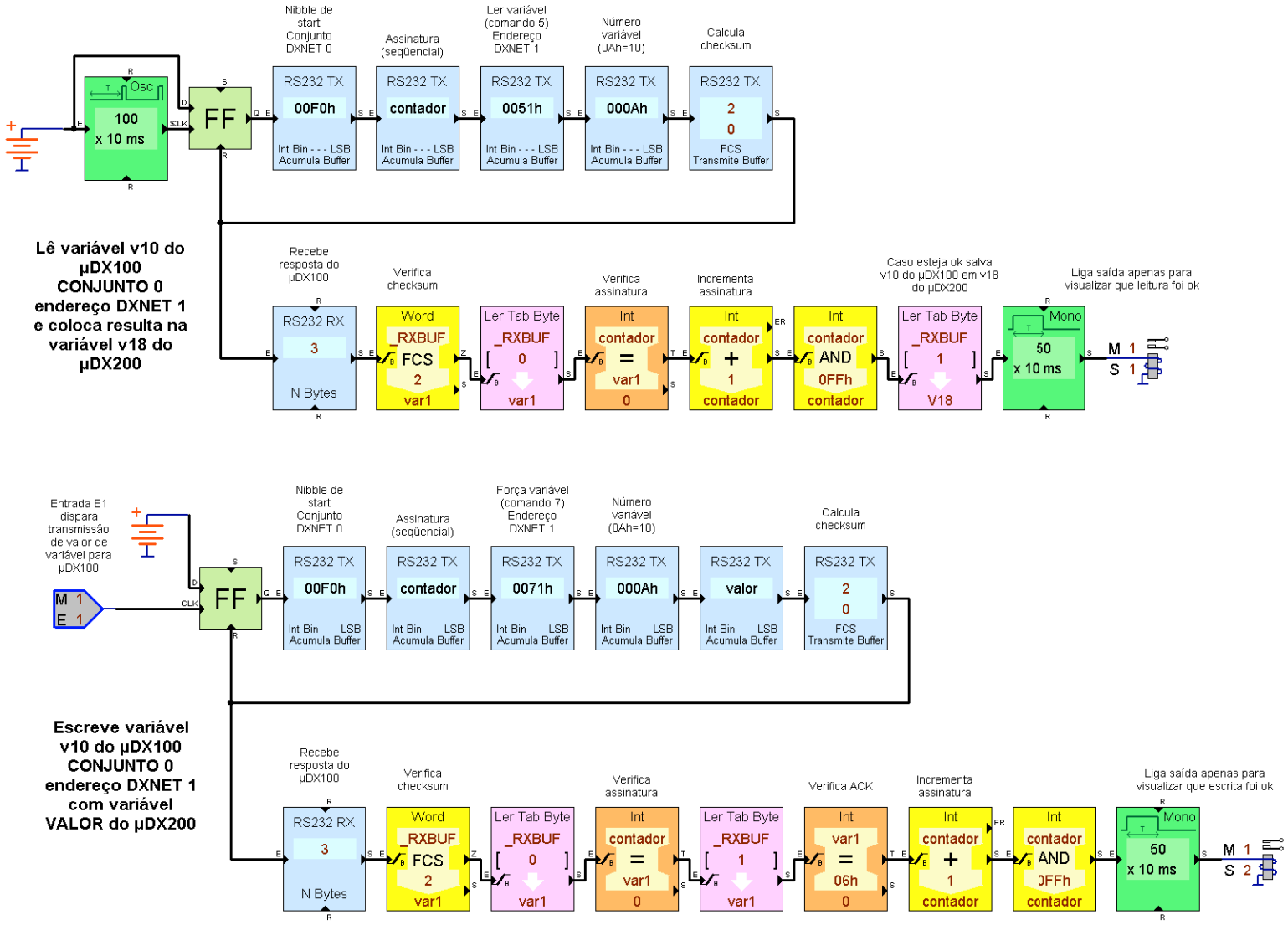
diretório de Macros (este diretório pode ser especificado no PG em Configurações → Configurações e Preferências... → Diretórios) aparecerão na Biblioteca de Componentes (aba Macros):


**DEXTER Indústria e Comércio de Equipamentos Eletrônicos Ltda.**
Av. Pernambuco, 1328 Cjs.307/309/310 - Porto Alegre - RS
Fones: (0xx51) 3343-2378, 3343-5532
Página Internet: www.dexter.ind.br
E-mail: dexter@dexter.ind.br

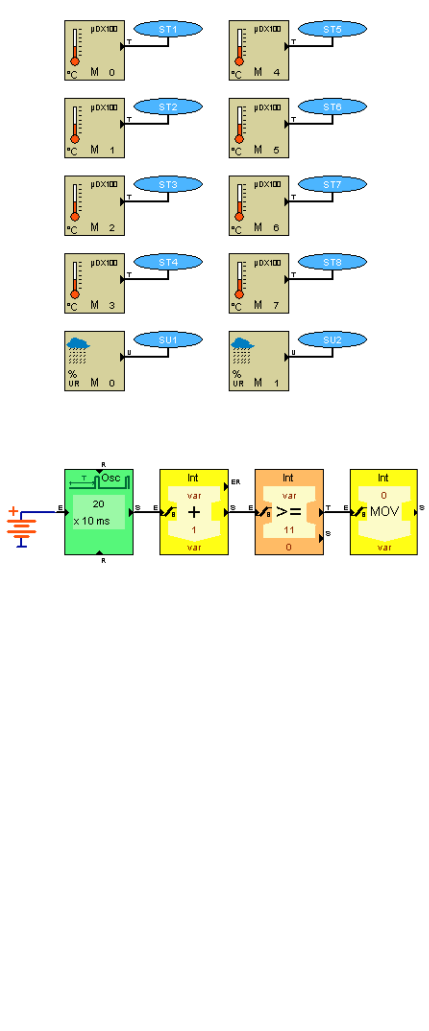



| DEXTER µDX | Versão 1.0 | Data 5/3/2007 |
|---|---|---|
| Descrição: Figura01_comunicação_µDX100_µDX200.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 1: Comunicação µDX100 -> µDX200 via bloco RS232-RX DXNET+ µDX100 | | |


Lê variável v10 do µDX100 CONJUNTO 1 endereço DXNET 1 e coloca resulta na variável v18 do µDX200
Escreve variável v10 do µDX100 CONJUNTO 1 endereço DXNET 1 com variável VALOR do µDX200
Entrada E1 dispara transmissão de valor de variável para µDX100
Nibble de start Conjunto DXNET 1
Assinatura (seqüencial)
Ler variável (comando 5) Endereço DXNET 1
Número variável (0Ah=10)
Força variável (comando 7) Endereço DXNET 1
Calcula checksum
Recebe resposta do µDX100
Verifica checksum
Verifica assinatura
Incrementa assinatura
Verifica ACK
Caso esteja ok salva v10 do µDX100 em v18 do µDX200
Liga saída apenas para visualizar que leitura foi ok
Liga saída apenas para visualizar que escrita foi ok
DEXTER µDX
Versão 1.0
Data 9/8/2007
Descrição Figura02_comunicação_µdx100_µdx200.dxg
Autor Claudio Richter
Comentário Figura 2: Comunicação µDX100 <--> µDX200 via programa aplicativo

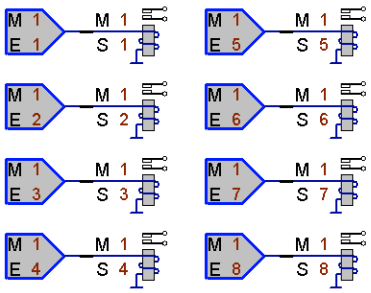



| DEXTER µDX | Versão 1.0 | Data 20/4/2008 |
|---|---|---|
| Descrição: Figura03_comunicação_µdx100_µdx200.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 3: Comunicação µDX100 <--> µDX200 via programa aplicativo | | |

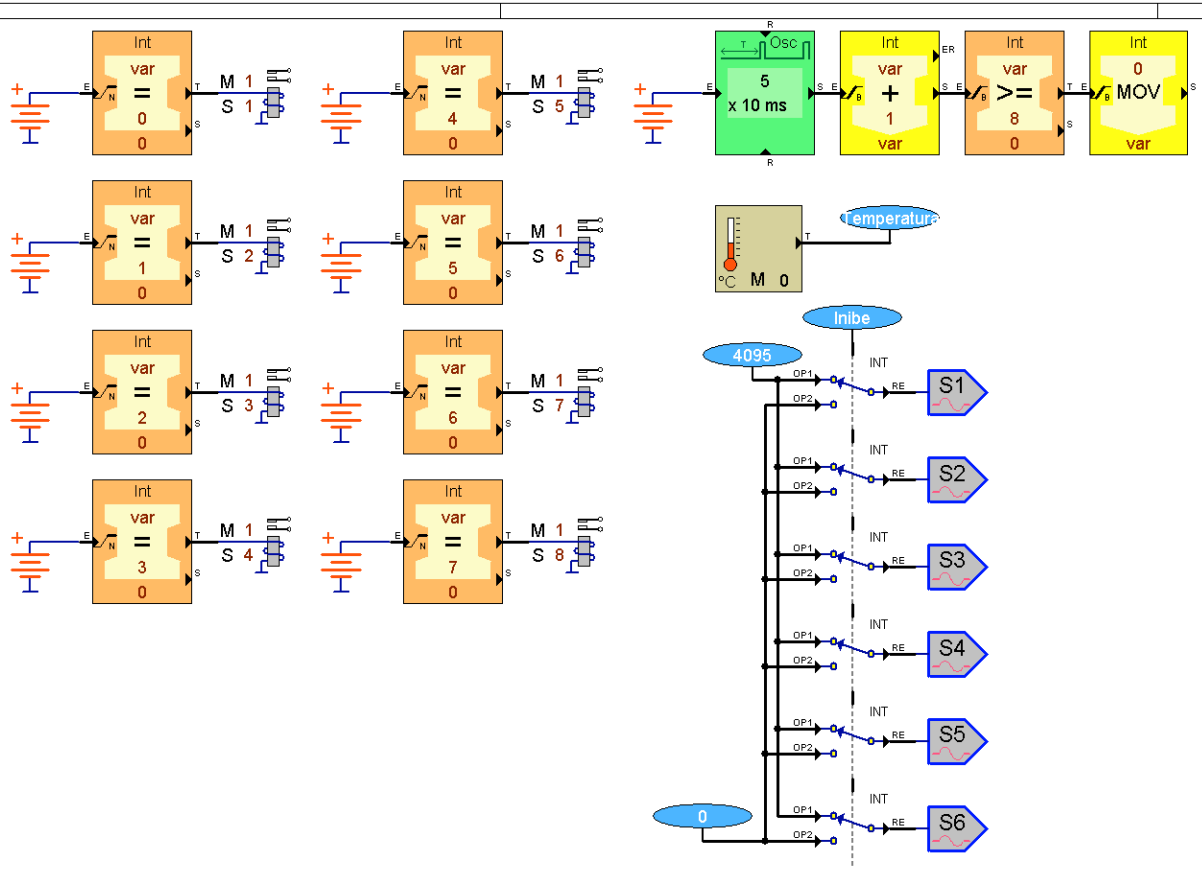




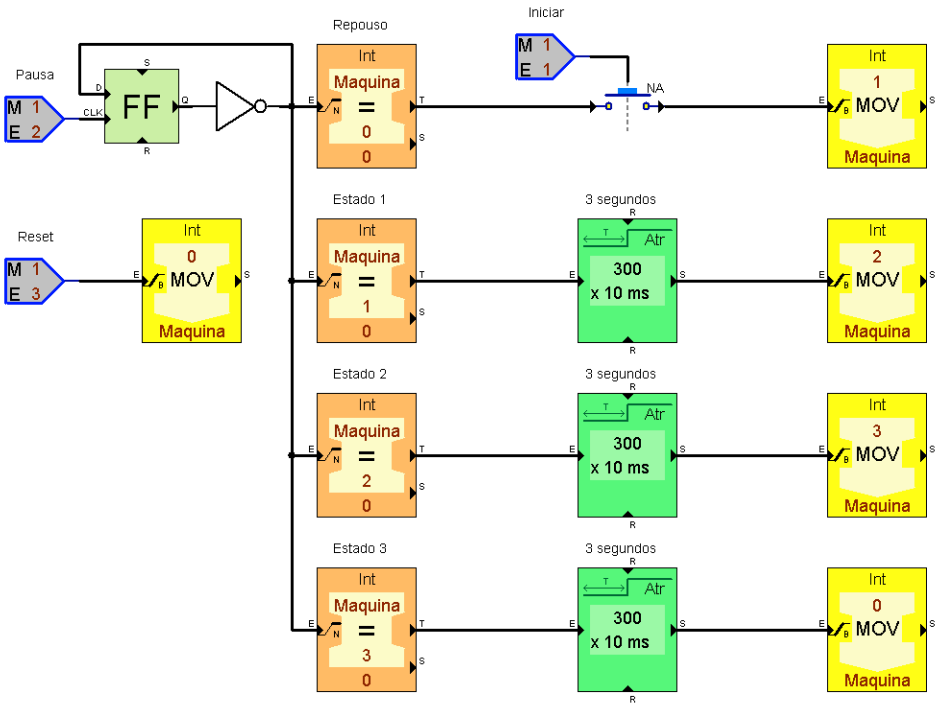
| DEXTER µDX | Versão 1.0 | Data 21/5/2007 |
|---|---|---|
| Descrição: Figura04_comunicação_µDX100_µDX200.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 4: Comunicação µDX200 -> µDX100 | | |



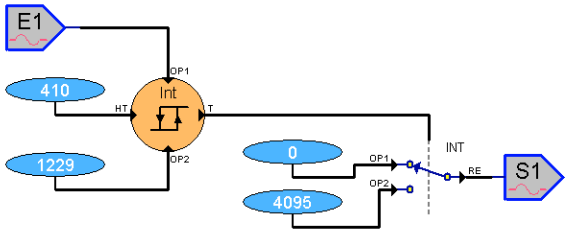

| DEXTER µDX | Versão 1.0 | Data 18/5/2005 |
|---|---|---|
| Descrição: Figura05_teste_µDX210.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 5: Teste de Expansão µDX210 | | |



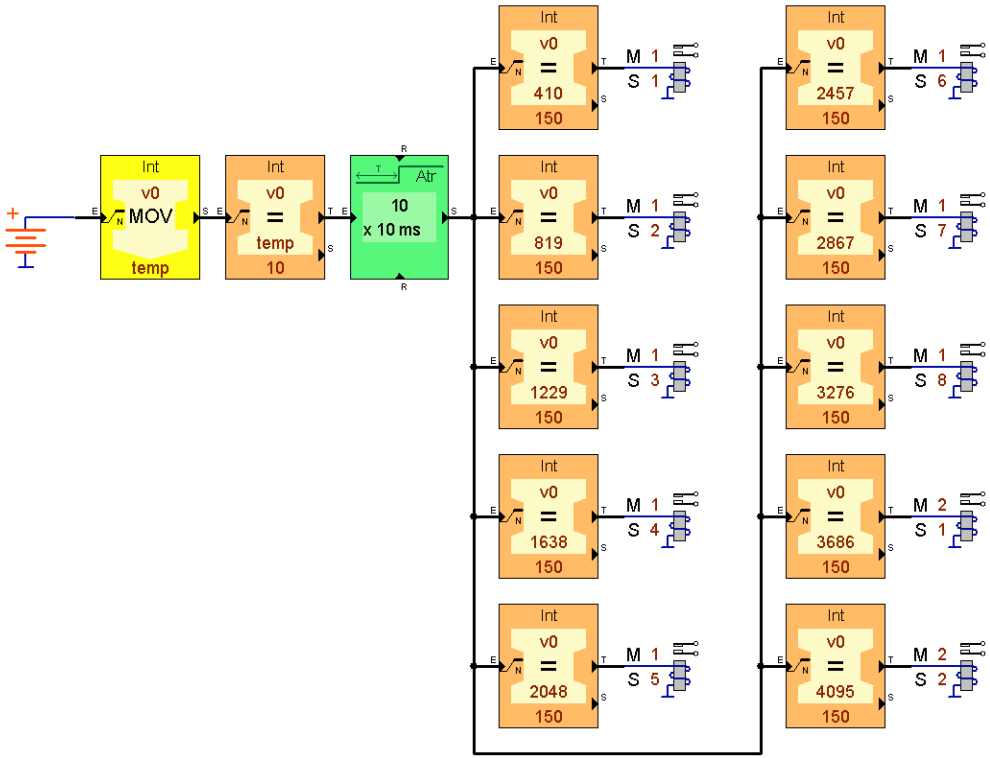

| DEXTER µDX | Versão 1.0 | Data 10/7/2006 |
|---|---|---|
| Descrição: Figura06_varre_µDX210.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 6: Varre as saídas da Expansão µDX210. | | |



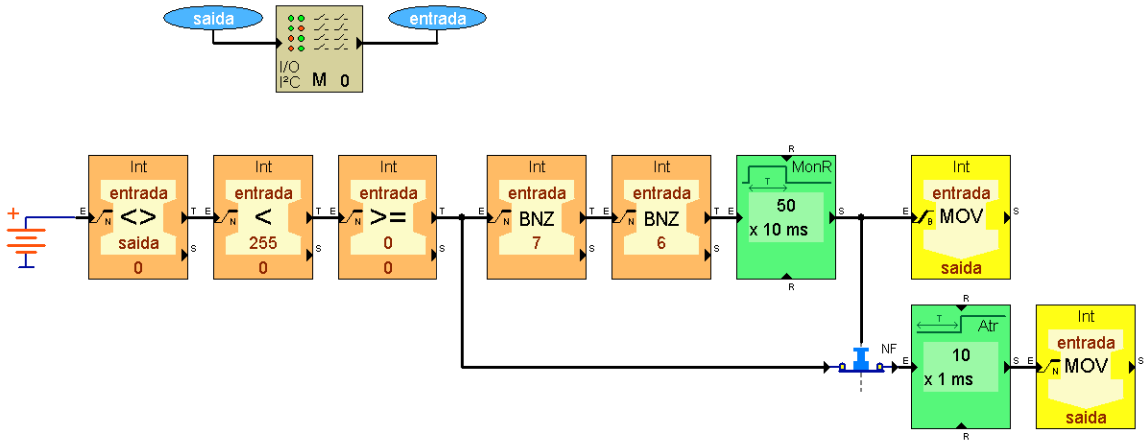

| DEXTER µDX | Versão 1.0 | Data 10/6/2007 |
|---|---|---|
| Descrição: Figura07_maquina_estados.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 7: Exemplo de Máquina de Estados | | |

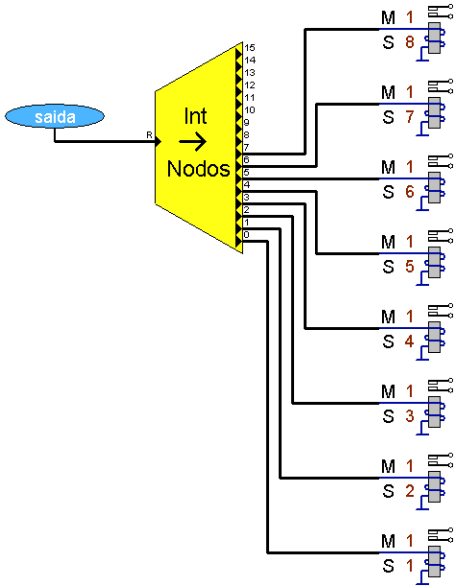



| | | |
|---|---|---|
| DEXTER µDX | Versão 1.0 | Data 9/11/2006 |
| Descrição Figura08_IO_digital.dxg | | |
| Autor Claudio Richter | | |
| Comentário Figura 8: Exemplo de uso de I/O analógico como I/O digital | | |

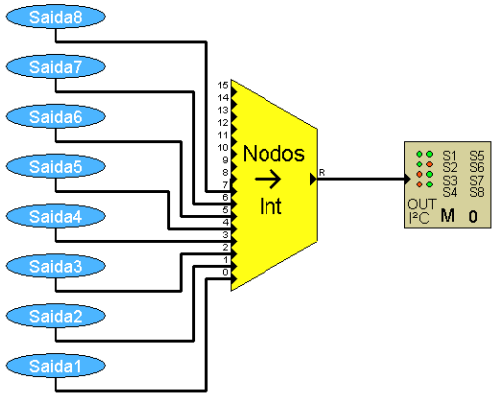



| DEXTER µDX | Versão 1.0 | Data 24/4/2007 |
|---|---|---|
| Descrição: Figura09_Multiplexador.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 9: Uso de Multiplexador de Pulsadores | | |

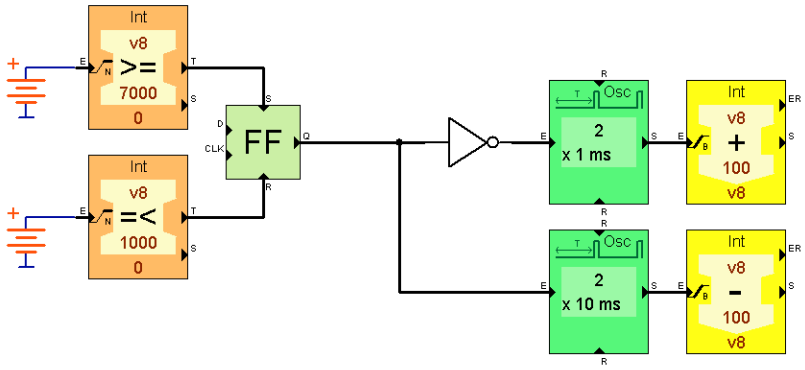


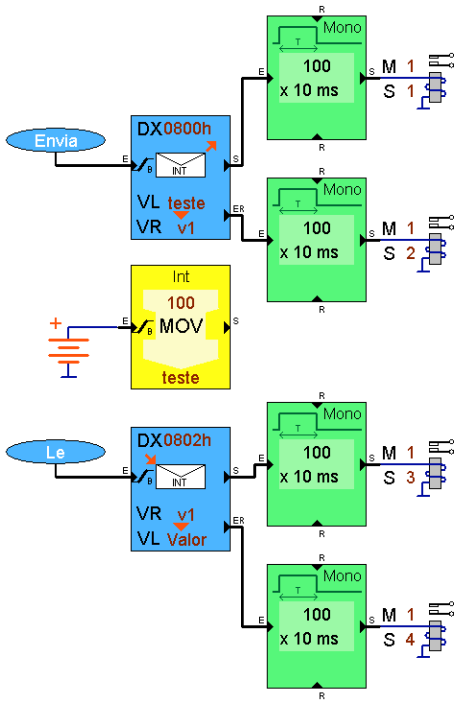

| DEXTER µDX | Versão 1.0 | Data 11/10/2006 |
|---|---|---|
| Descrição: Figura10_Keypad.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 10: Uso de Keypad do µDX200 | | |



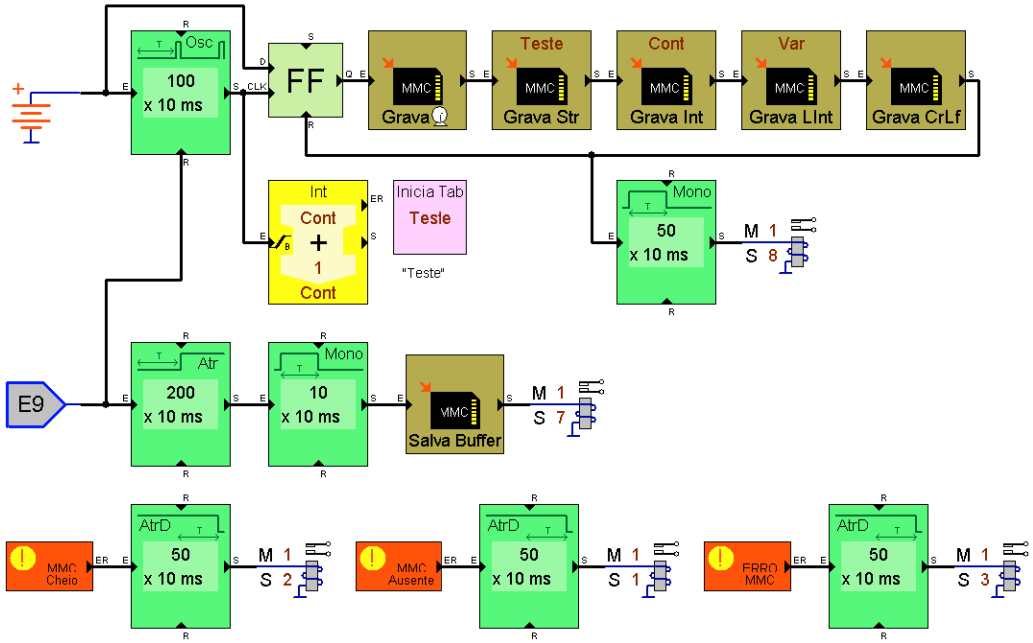

| DEXTER µDX | Versão 1.0 | Data 21/4/2008 |
|---|---|---|
| Descrição<br>Figura11_µDX212.dxg | | |
| Autor<br>Claudio Richter | | |
| Comentário<br>Figura 11: Acionamento de Expansão µDX212 | | |



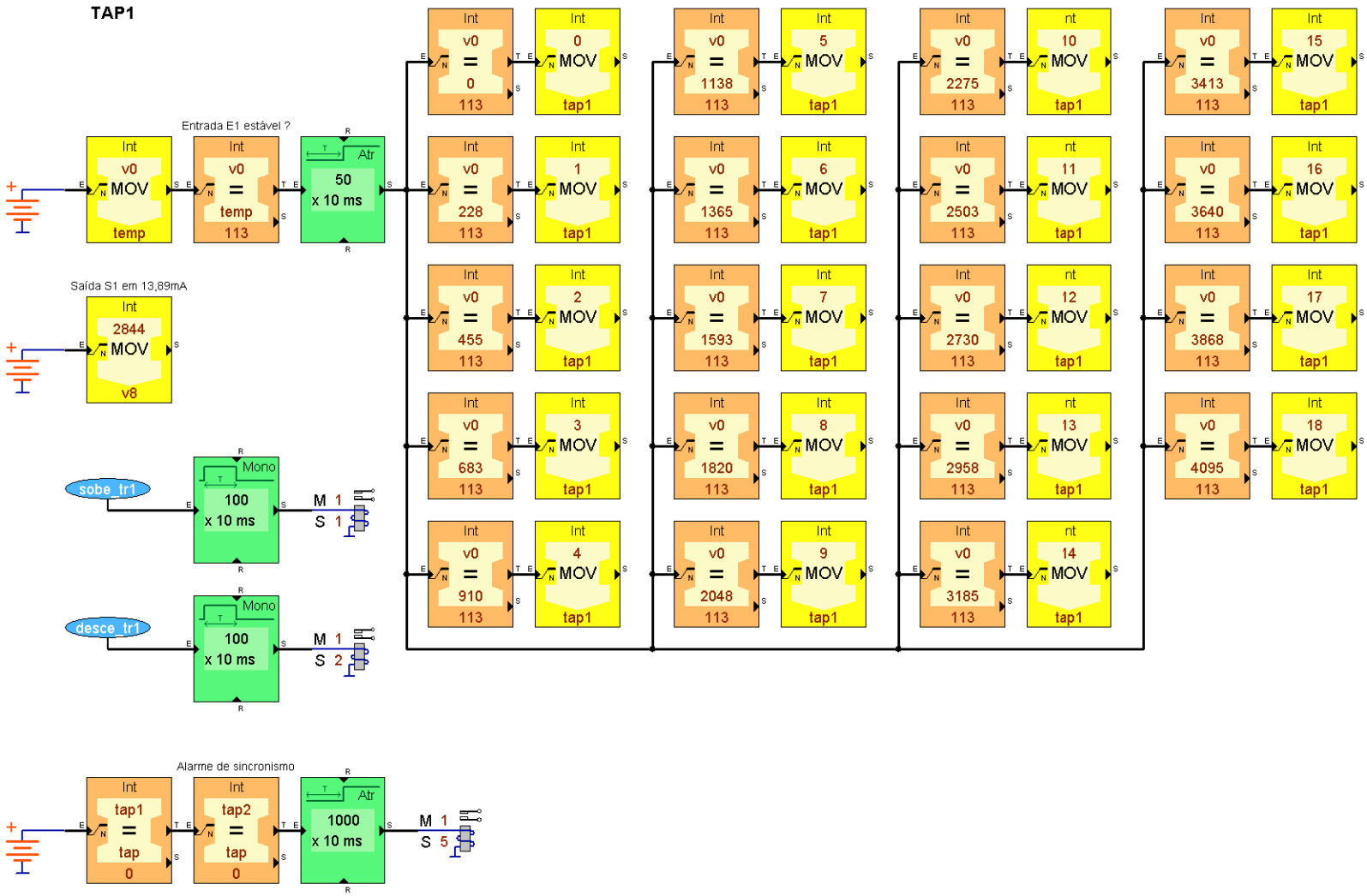

| DEXTER µDX | Versão 1.0 | Data 31/8/2007 |
|---|---|---|
| Descrição: Figura12_Dimmer.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 12: Acionamento de Dimmer | | |



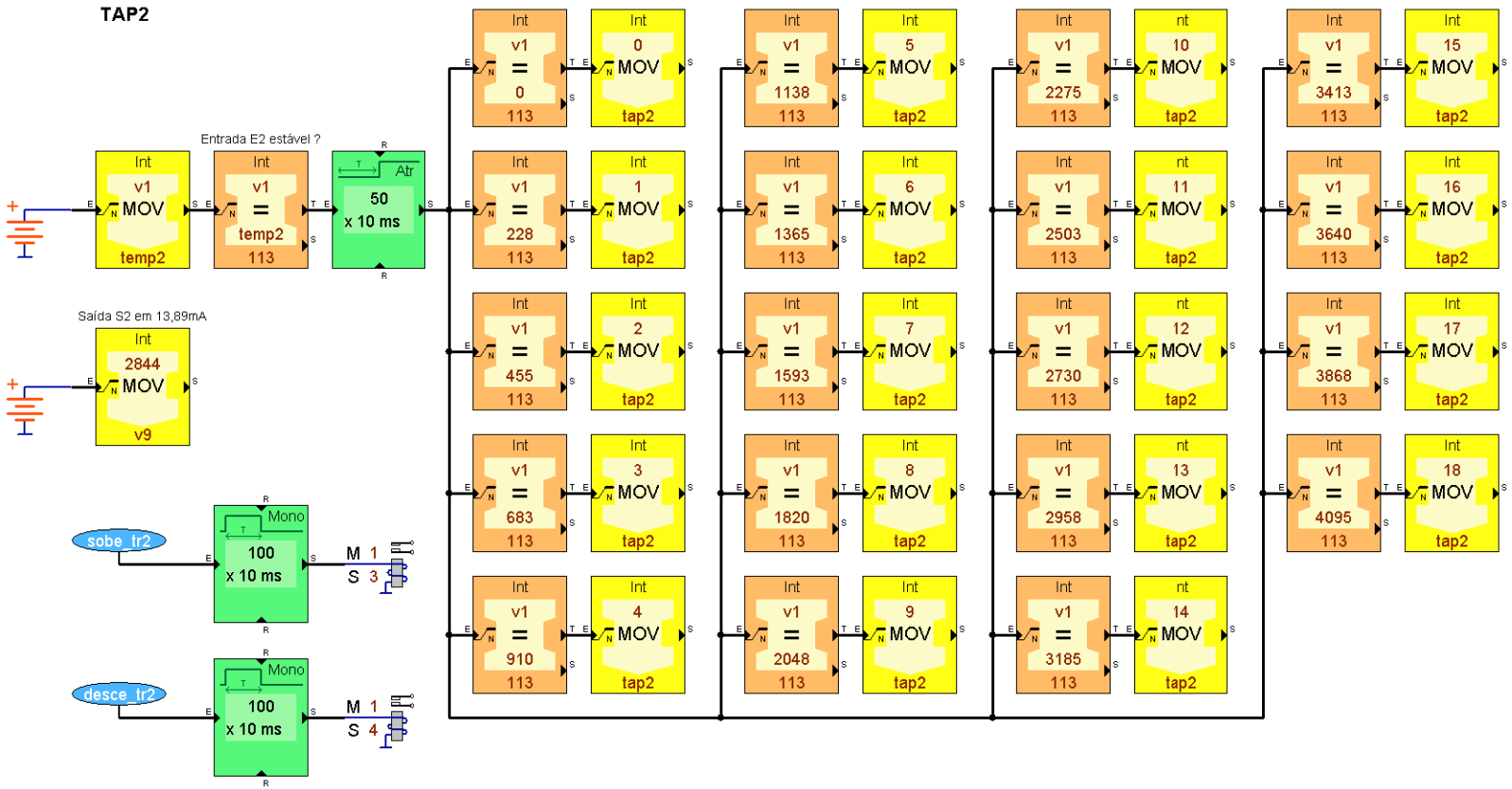

| DEXTER µDX | Versão 1.0 | Data 19/9/2007 |
|---|---|---|
| Descrição: Figura13_MCI.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 13: Dimmer via MCI | | |



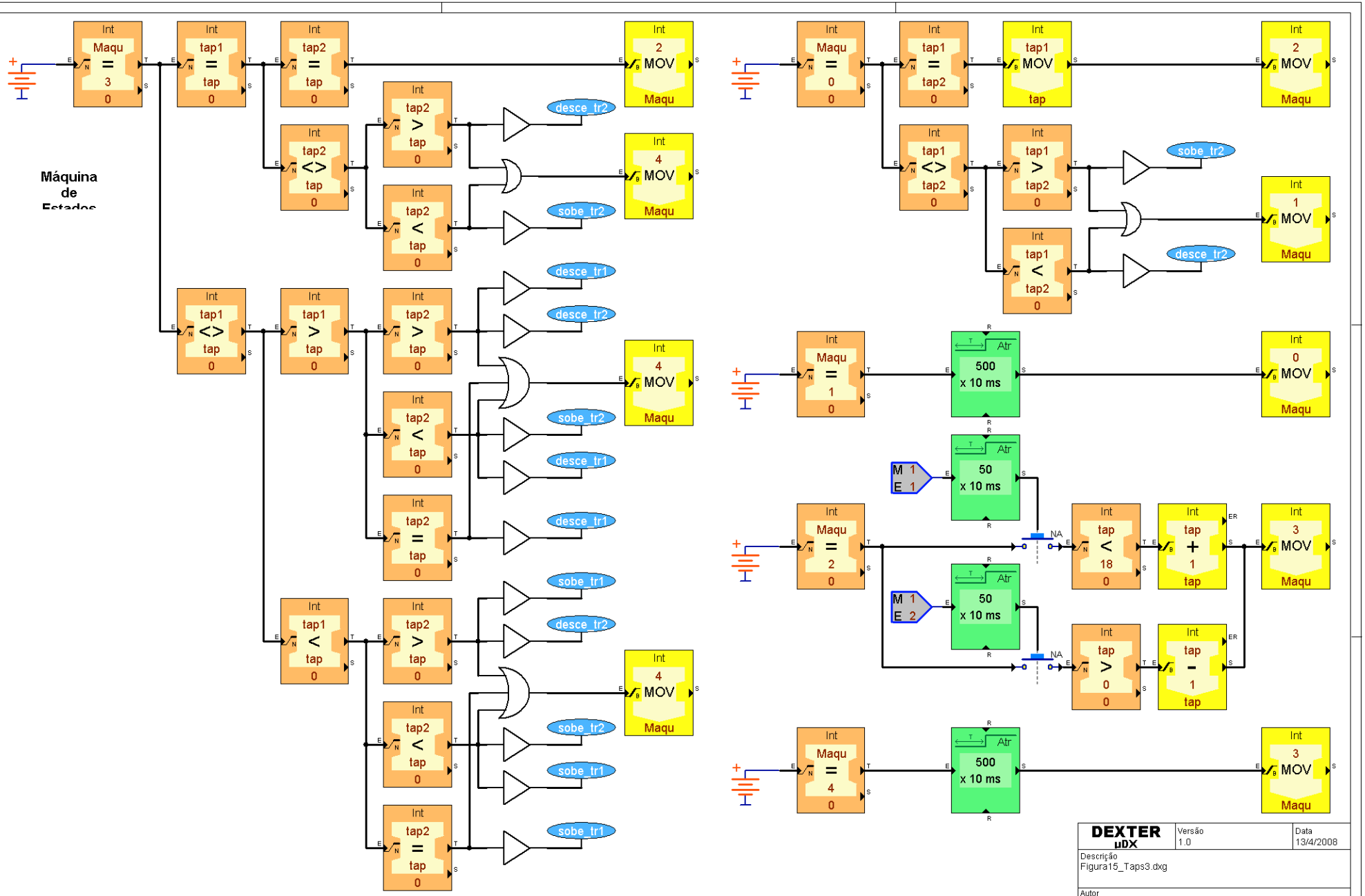

| DEXTER µDX | Versão 1.0 | Data 14/9/2006 |
|---|---|---|
| Descrição: Figura14_MMC.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 14: Uso de cartão MMC | | |

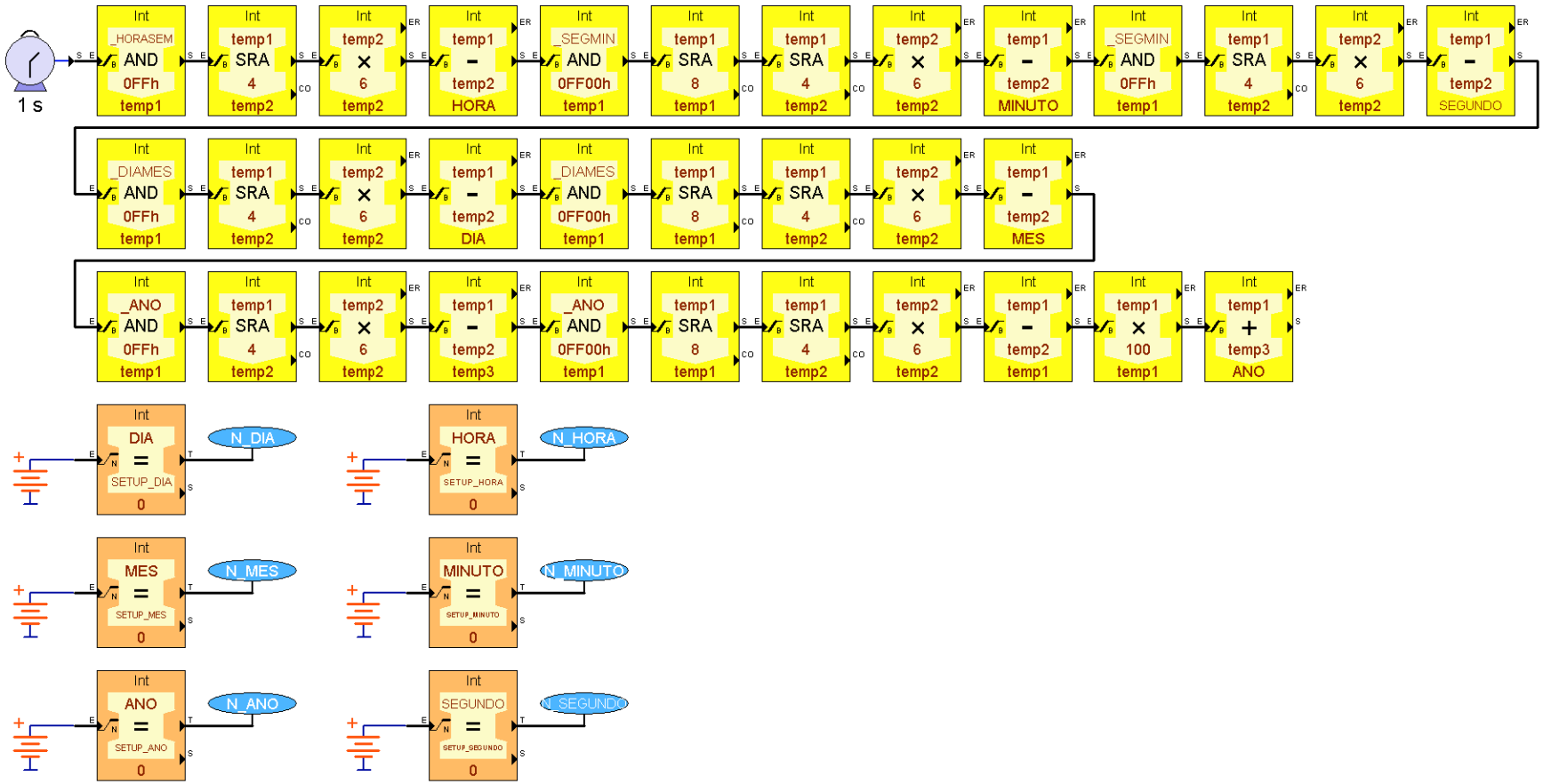



| DEXTER µDX | Versão 1.0 | Data 24/4/2007 |
|---|---|---|
| Descrição: Figura15_Taps1.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 15: Leitura de TAP1 | | |

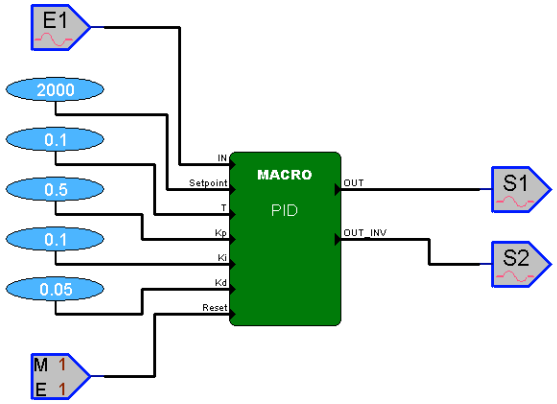



| DEXTER µDX | Versão 1.0 | Data 24/4/2007 |
|---|---|---|
| Descrição: Figura15_Taps2.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 15: Leitura de TAP2 | | |

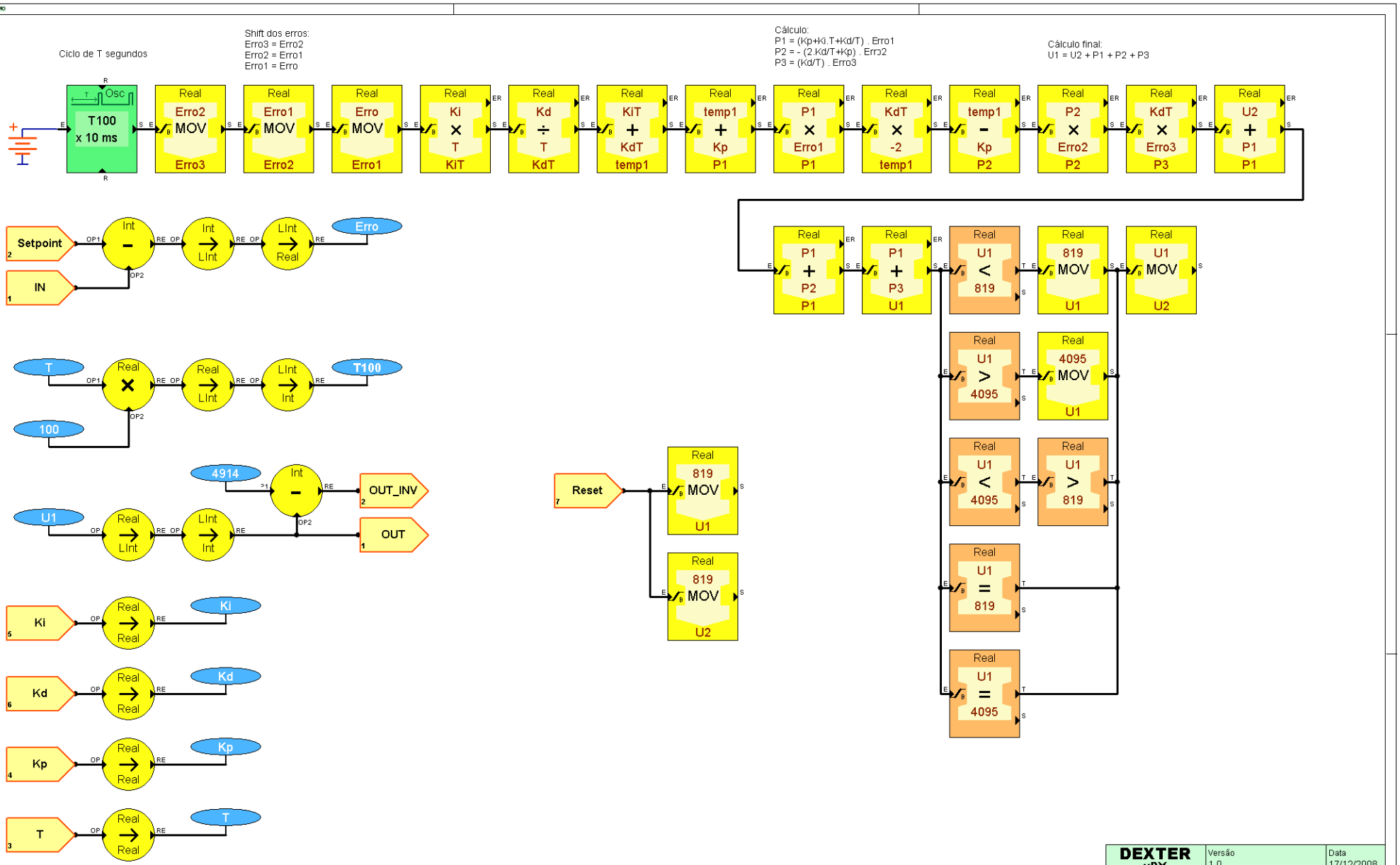

Máquina
de
Estados
Int Maqu = 3 0
Int tap1 = tap 0
Int tap2 = tap 0
Int tap2 <> tap 0
Int tap2 > tap 0
Int tap2 < tap 0
Int 2 MOV Maqu
Int 4 MOV Maqu
desce_tr2
sobe_tr2
Int tap1 <> tap 0
Int tap1 > tap 0
Int tap2 > tap 0
Int tap2 < tap 0
Int tap2 = tap 0
Int 4 MOV Maqu
desce_tr1
desce_tr2
sobe_tr2
desce_tr1
desce_tr1
Int tap1 < tap 0
Int tap2 > tap 0
Int tap2 < tap 0
Int tap2 = tap 0
Int 4 MOV Maqu
sobe_tr1
desce_tr2
sobe_tr2
sobe_tr1
sobe_tr1
Int Maqu = 0 0
Int tap1 = tap2 0
Int tap1 MOV tap
Int 2 MOV Maqu
Int tap1 <> tap2 0
Int tap1 > tap2 0
Int tap1 < tap2 0
Int 1 MOV Maqu
sobe_tr2
desce_tr2
Int Maqu = 1 0
Atr 500 x 10 ms
Int 0 MOV Maqu
M 1
E 1
Atr 50 x 10 ms
Int Maqu = 2 0
NA
Int tap < 18 0
Int tap + 1 tap
Int 3 MOV Maqu
ER
M 1
E 2
Atr 50 x 10 ms
NA
Int tap > 0 0
Int tap - 1 tap
ER
Int Maqu = 4 0
Atr 500 x 10 ms
Int 3 MOV Maqu
DEXTER
µDX
Versão
1.0
Data
13/4/2008
Descrição
Figura15_Taps3.dxg
Autor
Claudio Richter
Comentário
Figura 15: Máquina de Estados



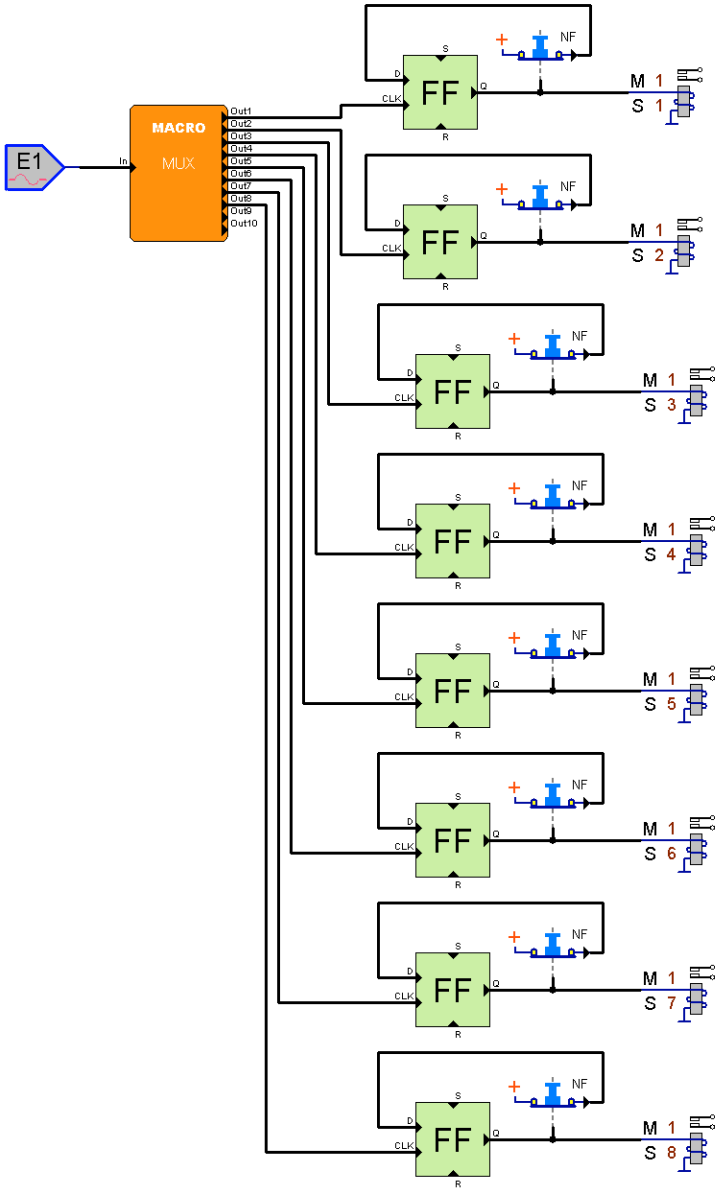

| DEXTER µDX | Versão 1.0 | Data 4/6/2008 |
|---|---|---|
| Descrição: Figura16_RTC.dxg | | |
| Autor: Claudio Richter | | |
| Comentário: Figura 16: Leitura de dados do relógio de tempo real do µDX200 | | |




| DEXTER µDX | Versão 1.0 | Data 17/12/2008 |
|---|---|---|
| Descrição<br>Teste de Controle PID | | |
| Autor<br>Claudio Richter | | |
| Comentário | | |

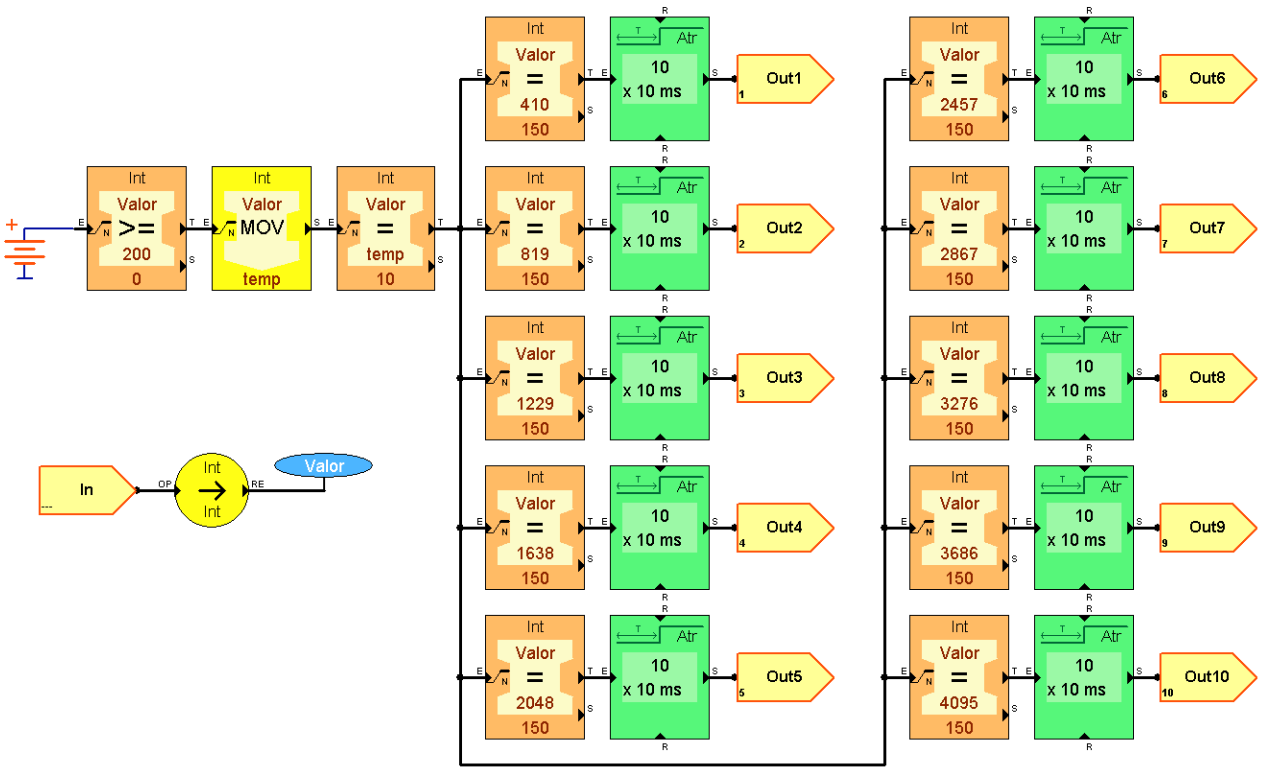



| DEXTER uDX | Versão 1.0 | Data 17/12/2008 |
|---|---|---|
| Descrição: Macro para Controle PID | | |
| Autor: Claudio Richter | | |
| Comentário: Saídas complementares (4-20mA). | | |

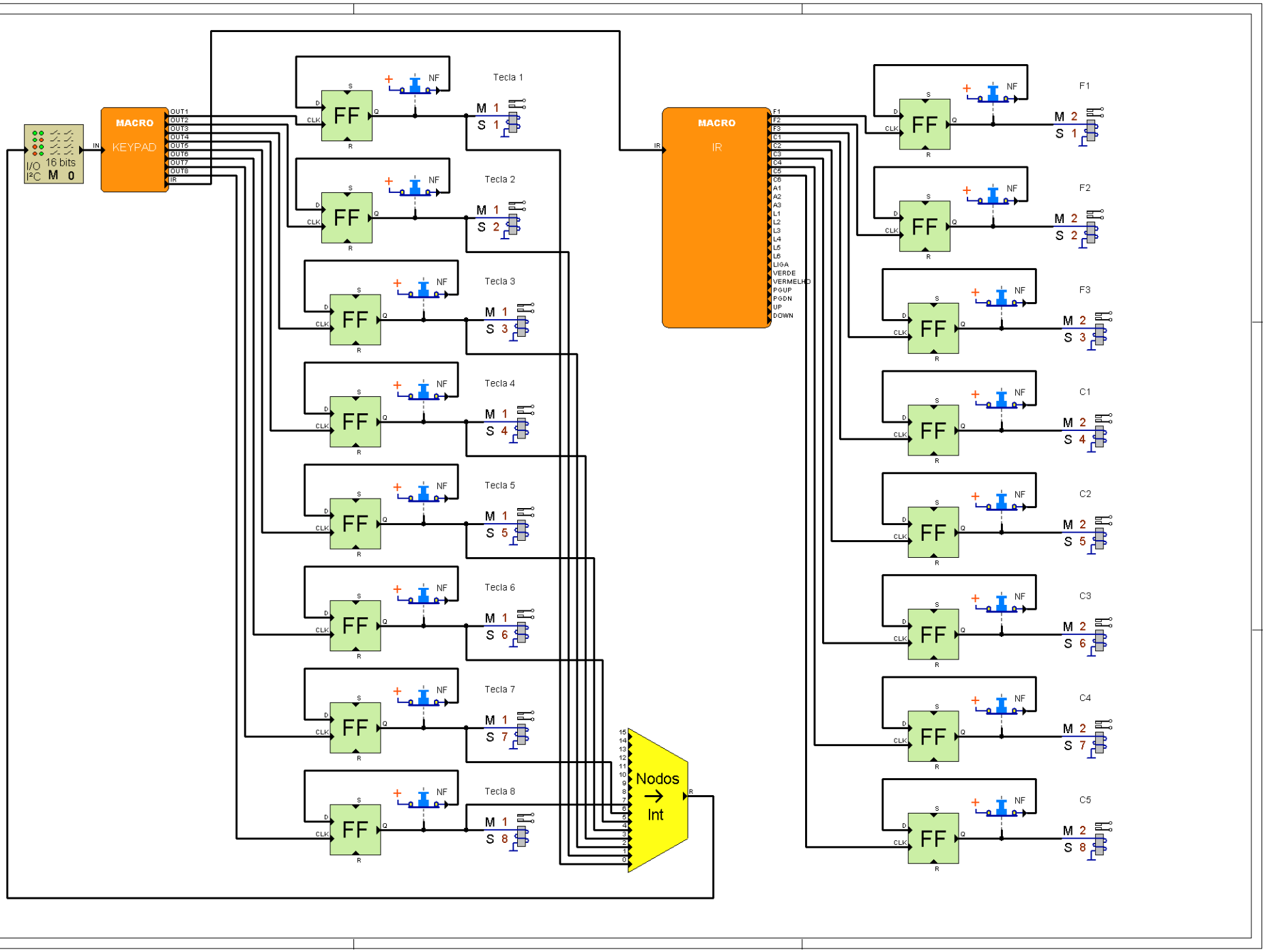


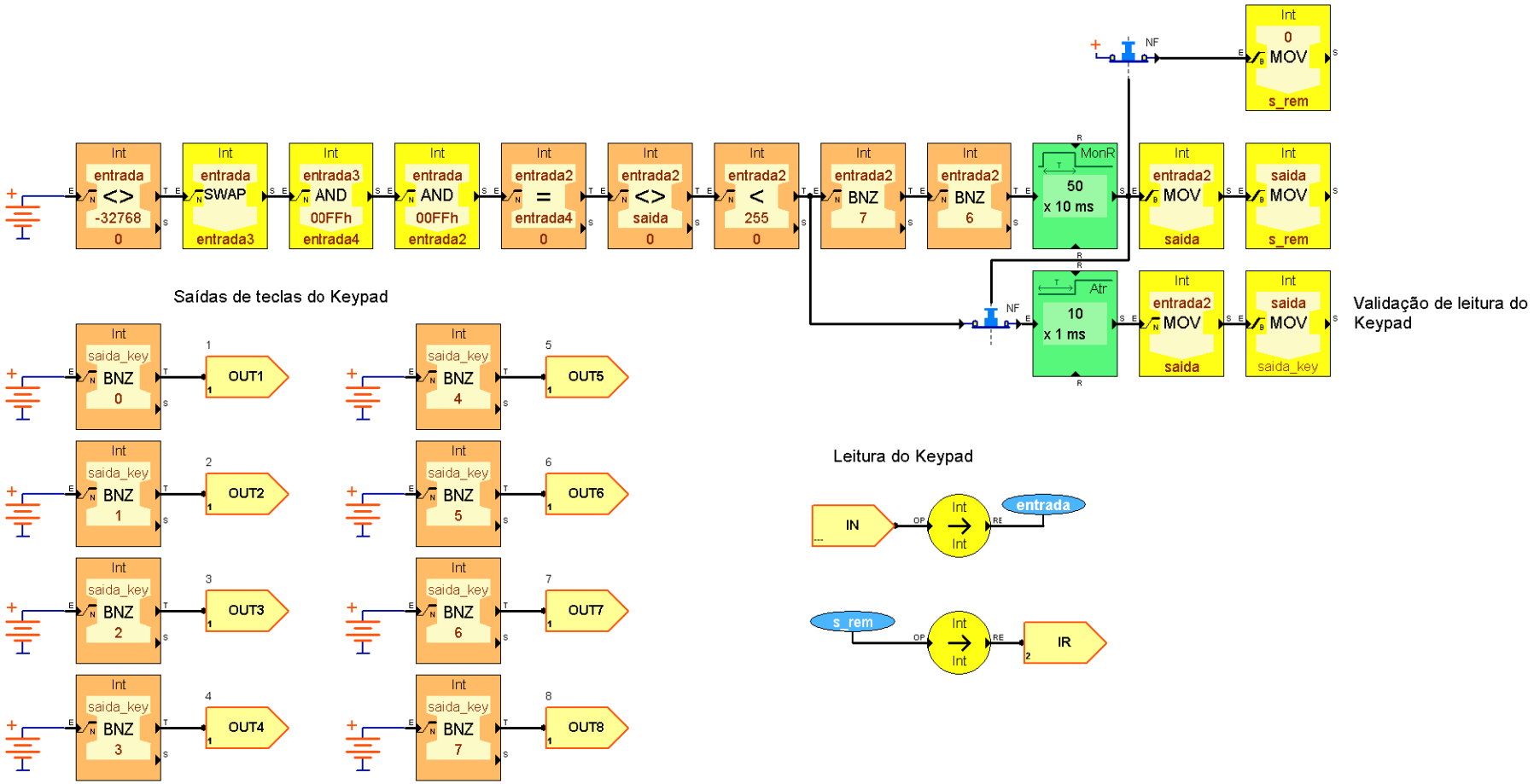

| DEXTER µDX | Versão 1.0 | Data 1/1/2009 |
|---|---|---|
| Descrição<br>Exemplo de uso da macrocélula MUX | | |
| Autor<br>Claudio Richter | | |
| Comentário | | |

MACRO



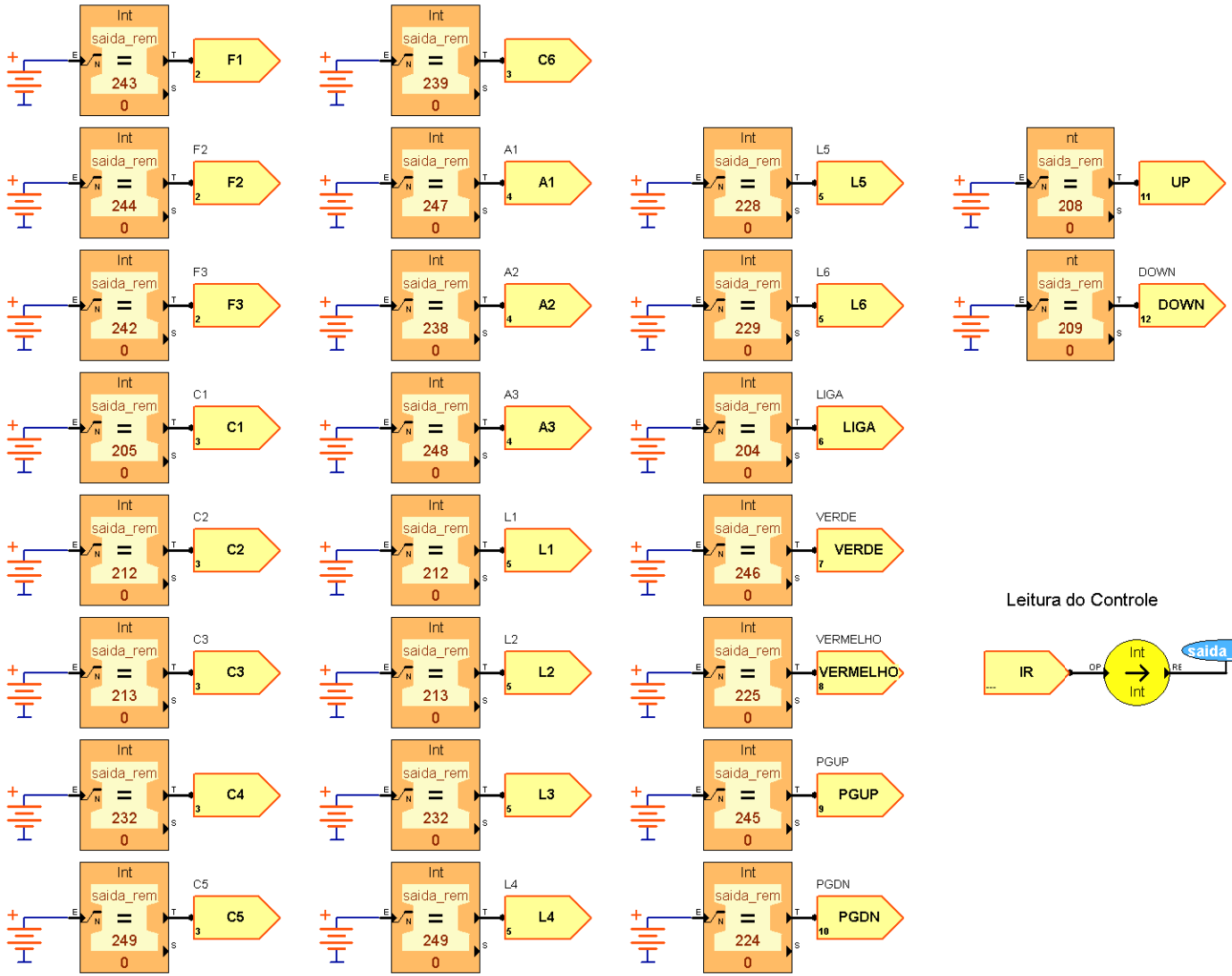

| DEXTER µDX | Versão 1.0 | Data 29/1/2009 |
|---|---|---|
| Descrição<br>Macro para decodificação de Multiplexador de Pulsadores | | |
| Autor<br>Claudio Richter | | |
| Comentário | | |


I/O
I²C
16 bits
M 0
MACRO
KEYPAD
IN
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
OUT8
IR
FF
NF
Tecla 1
M 1
S 1
Tecla 2
S 2
Tecla 3
S 3
Tecla 4
S 4
Tecla 5
S 5
Tecla 6
S 6
Tecla 7
S 7
Tecla 8
S 8
Nodos
Int
MACRO
IR
F1
F2
F3
C1
C2
C3
C4
C5
C6
A1
A2
A3
L1
L2
L3
L4
L5
L6
LIGA
VERDE
VERMELHO
PGUP
PGDN
UP
DOWN
M 2
S 1
S 2
S 3
S 4
S 5
S 6
S 7
S 8

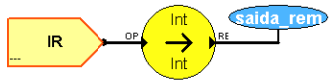

Int
entrada
<>
-32768
0
Int
entrada
SWAP
entrada3
Int
entrada3
AND
00FFh
entrada4
Int
entrada
AND
00FFh
entrada2
Int
entrada2
=
entrada4
0
Int
entrada2
<>
saida
0
Int
entrada2
<
255
0
Int
entrada2
BNZ
7
Int
entrada2
BNZ
6
MonR
50
x 10 ms
Int
entrada2
MOV
saida
Int
saida
MOV
s_rem
NF
Int
0
MOV
s_rem
Atr
10
x 1 ms
Int
entrada2
MOV
saida
Int
saida
MOV
saida_key
Validação de leitura do Keypad
Saídas de teclas do Keypad
Int
saida_key
BNZ
0
1
OUT1
Int
saida_key
BNZ
1
2
OUT2
Int
saida_key
BNZ
2
3
OUT3
Int
saida_key
BNZ
3
4
OUT4
Int
saida_key
BNZ
4
5
OUT5
Int
saida_key
BNZ
5
6
OUT6
Int
saida_key
BNZ
6
7
OUT7
Int
saida_key
BNZ
7
8
OUT8
Leitura do Keypad
IN
Int
Int
entrada
s_rem
Int
Int
IR

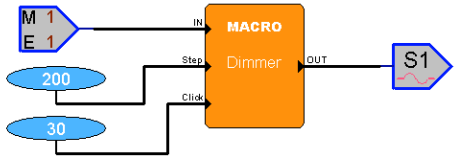
MACRO

Saídas de controle remoto do Keypad

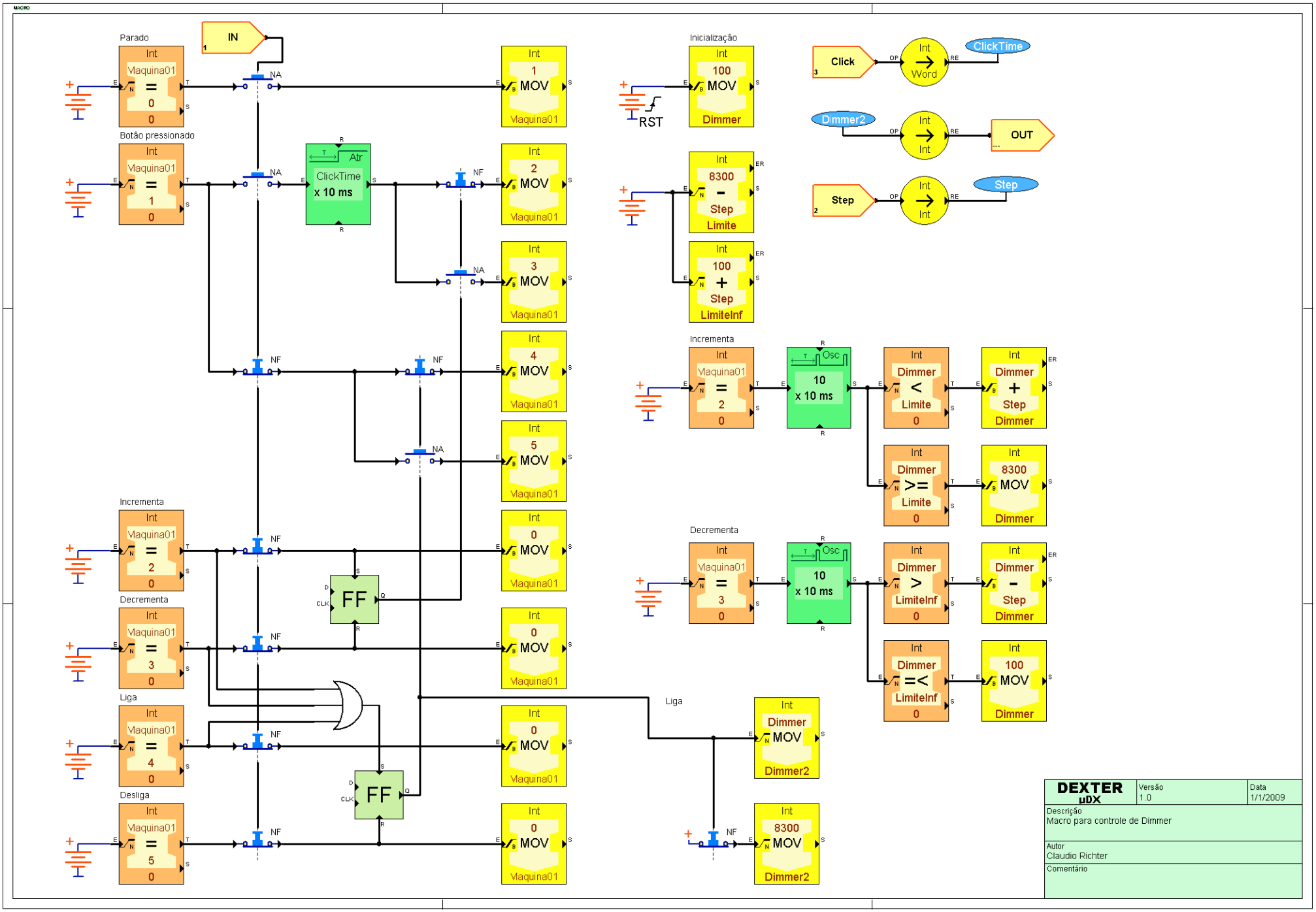


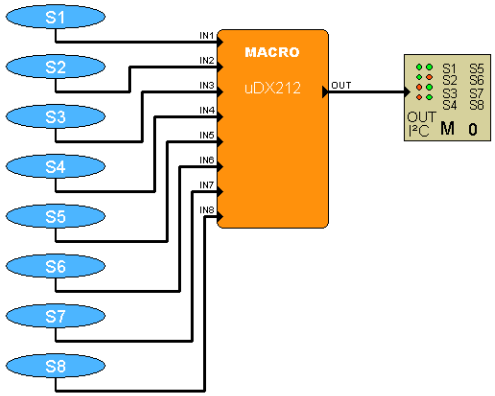

Leitura do Controle





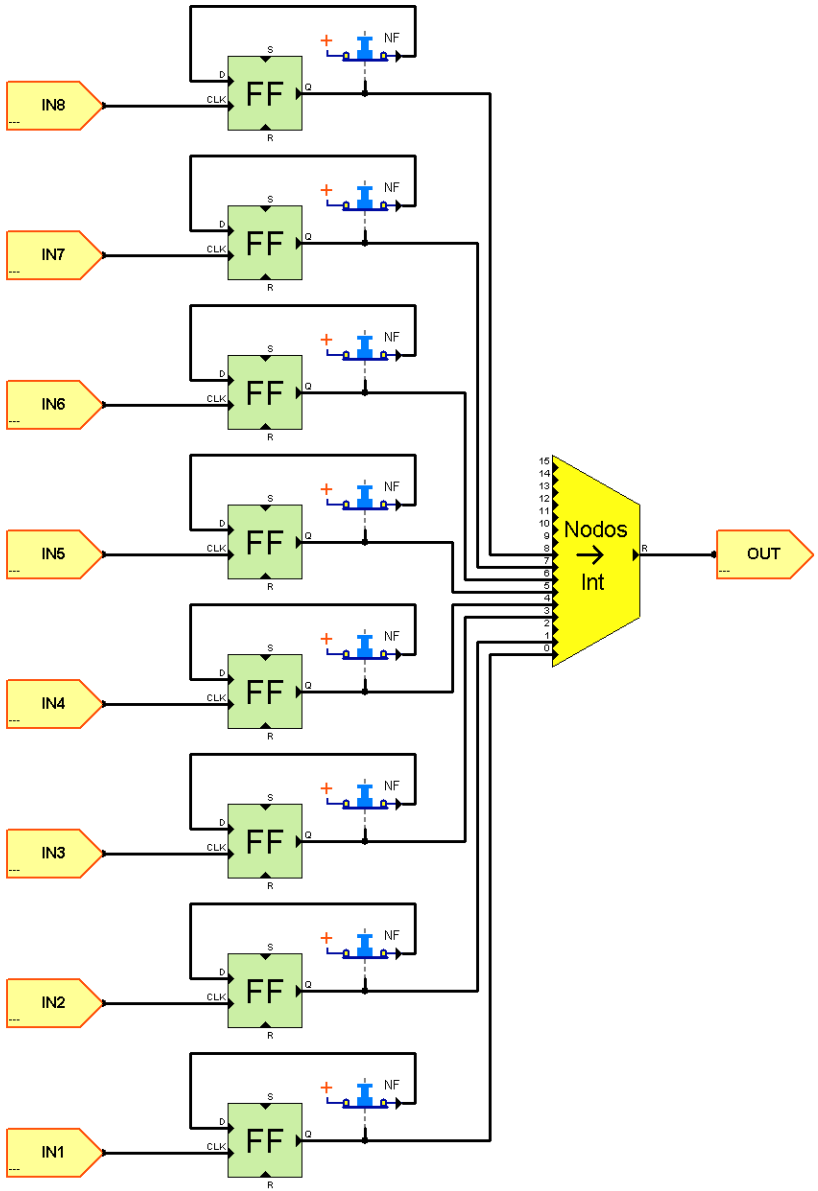

| DEXTER µDX | Versão 1.0 | Data 1/1/2009 |
|---|---|---|
| Descrição<br>Teste_Dimmer | | |
| Autor<br>Claudio Richter | | |
| Comentário | | |


MACRO
Parado
IN
Int
Vlaquina01
= 0 0
NA
Int 1 MOV Vlaquina01
Botão pressionado
Int
Vlaquina01
= 1 0
NA
R
T Atr
ClickTime
x 10 ms
R
NF
Int 2 MOV Vlaquina01
NA
Int 3 MOV Vlaquina01
NF
NF
Int 4 MOV Vlaquina01
NA
Int 5 MOV Vlaquina01
Incrementa
Int
Vlaquina01
= 2 0
NF
FF
D CLK S Q R
Int 0 MOV Vlaquina01
Decrementa
Int
Vlaquina01
= 3 0
NF
Int 0 MOV Vlaquina01
Liga
Int
Vlaquina01
= 4 0
NF
FF
D CLK S Q R
Int 0 MOV Vlaquina01
Desliga
Int
Vlaquina01
= 5 0
NF
Int 0 MOV Vlaquina01
Inicialização
Int 100 MOV Dimmer
RST
Int 8300 - Step Limite ER
Int 100 + Step LimiteInf ER
Click
OP
Int → Word
RE
ClickTime
Dimmer2
Int → Int
OUT
Step
Step
Incrementa
Int
Vlaquina01
= 2 0
R
T Osc
10
x 10 ms
R
Int Dimmer < Limite 0
Int Dimmer + Step Dimmer ER
Int Dimmer >= Limite 0
Int 8300 MOV Dimmer
Decrementa
Int
Vlaquina01
= 3 0
R
T Osc
10
x 10 ms
R
Int Dimmer > LimiteInf 0
Int Dimmer - Step Dimmer ER
Int Dimmer =< LimiteInf 0
Int 100 MOV Dimmer
Liga
Int Dimmer MOV Dimmer2
NF
Int 8300 MOV Dimmer2
DEXTER
µDX
Versão
1.0
Data
1/1/2009
Descrição
Macro para controle de Dimmer
Autor
Claudio Richter
Comentário




| DEXTER µDX | Versão 1.0 | Data 1/1/2009 |
|---|---|---|
| Descrição<br>Teste de macro para acionamento de µDX212 | | |
| Autor<br>Claudio Richter | | |
| Comentário | | |

MACRO




| DEXTER µDX | Versão 1.0 | Data 1/1/2009 |
|---|---|---|
| Descrição<br>Macro para acionamento com retenção de µDX212 | | |
| Autor<br>Claudio Richter | | |
| Comentário | | |