15. DESENVOLVIMENTO DE APLICAÇÕES

O modo e forma como se desenvolvem as aplicações para sistemas baseados em micro processadores está dependente das ferramentas hardware e software que o fabricante do processador põe disponível ou que em alternativa venhamos a desenvolver.

É necessário dispor de um compilador que produza código máquina a partir de um ficheiro de texto escrito numa qualquer linguagem (*assembly*, C, etc.). Será conveniente que a aplicação se desenvolva modelarmente o que implica uma ferramenta de ligação dos vários módulos num único ficheiro e um localizador que estabeleça quais endereços a atribuir a cada um dos elementos que constituem a aplicação.

Produzido o ficheiro em Código Máquina da aplicação é necessário dispor de um sistema baseado no processador para o qual a aplicação foi desenvolvida e que permita carregar aplicação em memória e disponibilize meios para teste da aplicação (*debugger*).

Para o desenvolvimento de aplicações para o processador PDS16, estão disponíveis as seguintes ferramentas:

dasm	compilador (assemblador) que a partir de um ficheiro de texto escrito em linguagem assemblyPDS16 produz um ficheiro em linguagem máquina.
SDP16	Sistema Didáctico baseado no processador PDS16 com suporte para carregamento de aplicações e vários mecanismos de teste de programas.
PD_Debugger	Aplicação para Microsoft Windows com capacidade de carregamento e teste de aplicações escritas em linguagem assemblyPDS16 sobre o sistema SDP16.

15.1 dasm

O programa `dasm.exe` é um assemblador didáctico uni modular para o processador PDS16 que a partir de um ficheiro de texto em linguagem assemblyPDS16, produzido num qualquer editor de texto (NOTEPAD/++, UTRAEDIT; CODEWRITE etc.) produz dois ficheiros com o mesmo nome do ficheiro fonte mas com as extensões .LST e .HEX.

O ficheiro de extensão **`lst`** é um ficheiro de texto destinado a ser impresso por conter o texto original adicionado do código de cada instrução e respectivo endereço, entre outra informação. Os erros de compilação são assinalados neste ficheiro, na linha correspondente. Para cada erro é indicado o tipo e a possível causa.

O ficheiro de extensão **`hex`** é um ficheiro de texto com formato Intel HEX80 contendo o código máquina destinado a ser transmitido em série para um sistema alvo. O formato Intel HEX80 é reconhecido pela generalidade dos sistemas de desenvolvimento e pelos gravadores de PROMs (OTP, EPROM, E2PROM, FLASH). Este ficheiro é constituído por caracteres ASCII organizados em tramas, contendo cada trama uma marcas de sincronização, o endereço físico dos bytes contidos na trama e um código para detecção de erros de transmissão.

O dasm não permite o desenvolvimento modelar de aplicações, e por esta razão não é necessária a ferramenta de ligação. A localização é estática e estabelecida no ficheiro fonte.

O `dasm.exe` tem que ser evocado numa janela de DOS. O nome do ficheiro fonte a ser compilado não pode conter espaços e a extensão deverá ser **`.a`** para melhor ser identificada. A evocação do dasm será concretizada na linha de comando sob a seguinte forma:

```
C:>dasm file.asm
```

15.1.1 Linguagem assemblyPDS16

Um ficheiro em linguagem assemblyPDS16 é composto de instruções, sendo cada instrução confinada a uma linha de texto. O compilador não distingue letras maiúsculas de minúsculas. Cada instrução é composta por quatro campos ordenados, com a seguinte forma geral:

[Label:] Instrução [Operando][,Operando] [;comentário] <CrLf>

Label Serve para referir uma variável, uma constante ou um endereço. É uma palavra iniciada por uma letra seguida de mais letras ou dígitos, terminada pelo carácter '**:**'. Pode conter o carácter '**_**' tanto no início da palavra como entre caracteres. Uma *label* quando evocada por uma instrução não tem o carácter '**:**'.

```
main:   ldi     r0,#0xff    ; registo r0 = 0x00ff
_lab_1: jnz     _lab_1      ; utiliza o registo r7 com offset -1
```

Instrução Pode ser a mnemónica de uma instrução do PDS16 ou uma instrução da linguagem. No caso de ser uma instrução da linguagem, esta é precedida pelo carácter '**.**'. O compilador dasm, dispõe das instruções que constam da Tabela 15-1.

Directiva	Formato	Descrição
.ASCII	[label:] .ASCII "string ascii"	Define uma *string* ascii não terminada por zero.
.ASCIIZ	[label:] .ASCII "string ascii"	Define uma *string* ascii terminada por zero.
.BYTE	[label:] .BYTE expression_list	Gera uma lista de valores tipo byte (8 bits) separados por **,**.
.BSS	.BSS	Secção predefinida para dados não iniciados.
.DATA	.DATA	Secção predefinida para dados iniciados.
.END	.END	Determina o fim do módulo assembler
.EQU	.EQU symbol_name**,** expression	Define o valor a atribuir a um símbolo de forma permanente.
.ORG	.ORG expression	Estabelece o valor inicial para o contador de endereços da secção corrente.
.SECTION	.SECTION section_name	Define uma secção de dados ou código.
.SET	.SET symbol_name**,** expression	Define o valor a atribuir a um símbolo de forma temporária.
.SPACE	[label:] .SPACE expression [**,**init_val]	Reserva espaço em unidades de byte. O espaço reservado pode ser iniciado com o valor init_val.
.TEXT	.TEXT	Secção de código predefinida.
.WORD	[label:] .WORD expression_list	Gera uma lista de valores tipo *word* (16 bits) separados pelo carácter '**,**'.

Tabela 15-1

Para facilitar a escrita de programas a linguagem suporta as seguintes pseudo instruções:

```
mov/f   rd,rs   ;orl/f rd,rs,rs
inc/f   rd      ;add/f rd,rd,#1
dec/f   rd      ;sub/ rd,rd,#1
ret             ;orlf r7,r5,r5
```

Disponibiliza ainda as seguintes funções:

`high(operando a 16 bits)` retorna os oito bits de maior peso do parâmetro de 16 bits.
`low(operando a 16 bits)` retorna os oito bits de menor peso do parâmetro de 16 bits.

Operando São os elementos que constituem os parâmetros de uma instrução. O número e tipo de cada operando depende da instrução. Existem instruções que não têm operandos.

Comentário É iniciado pelo carácter '**;**' e só abrange a linha corrente. O compilador ignora os restantes caracteres até ao fim da linha. Para comentário abrangendo várias linhas pode ser utilizado o comentário JAVA ou C (/* comentário */).

O texto que se segue é o exemplo de um ficheiro escrito em linguagem assemblyPDS16.

```
; Entrada após reset
        .section start
        .org    0
        jmp     main
        ld      r7,[r7,#0] ;executa um jump incondicional para a ISR
        .word   isr     ;variável iniciada com o endereço da ISR

        .equ    port_addr,0xff00
; variáveis alocadas na zona de endereçamento directo a partir do endereço 6
var4x12:
        .byte   low(40*3+14/2),high(255 + 0x101),12,13,14,32,98,255 ; decimal
        .byte   014     ;octal
        .byte   0xC     ;hexadecimal
        .byte   -12     ;representação a oito bits em código dos complementos
        .byte   1100b   ;binario
varChar:
        .byte   'A','B','C',0x0a,0x0d,0 ;vários bytes com os códigos ascii das letras
varWord_1:
        .word   port_addr  ; dois bytes
varWord_2:
        .word   main,l1,rotina    ;inicia array com os endereços das várias referências
varArray:
        .space  5, 0xAB ;preenche 5 bytes com o valor 0xAB
        .space  3       ;reserva 3 bytes e inicia com zero

        .text
main:   ldi     R0,#123
        ld      r1,[r7,#2] ;coloca no registo r1 o valor do endereço de main
        jmp     l1
        .word   main       ;reserva espaço de uma Word e inicia com o endereço de main
l1:     ldi     R2,#low(str_1)    ;r2=0
```

```
        ldih    R3,#high(str_1)   ;r3=0x7f
        orl     r2,r2,r3   ; coloca em r2 o endereço de str_1
        jmpl    ROTINA     ; salto ligado para a rotina ROTINA
        jmp     $          ; salta para o endereço corrente

rotina: st      R1,[r2,#0] ; guarda endereço de main em varArray
        ld      r0,[r1,#4] ; coloca em r0 o código da instrução ldi r2,#low(varArray)
        ldb     r3,var4x12+2 ;coloca em r3 o byte lido do endereço directo varx12+2
        inc     r2
        movf    r0,r2
        ret

        .section indirectData
        .org    0x7f00
str_1:  .asciiz “string terminada pelo valor 0”

        .end
```

15.2 SDP16

O sistema didáctico SDP16, cujo diagrama de blocos é apresentado na Figura 15-1, é constituído pelos seguintes elementos:

- CPU PDS16
- memória de programa e dados;
- portos paralelos de entrada e saída;
- módulo *Single Step*;
- mostrador dos sinais dos barramentos de controlo, de endereços e de dados;
- Bus *Arbiter*;
- módulo DMA para acesso à memória e aos portos;
- canal USB/JTAG;
- filtro e regulador da tensão de alimentação.

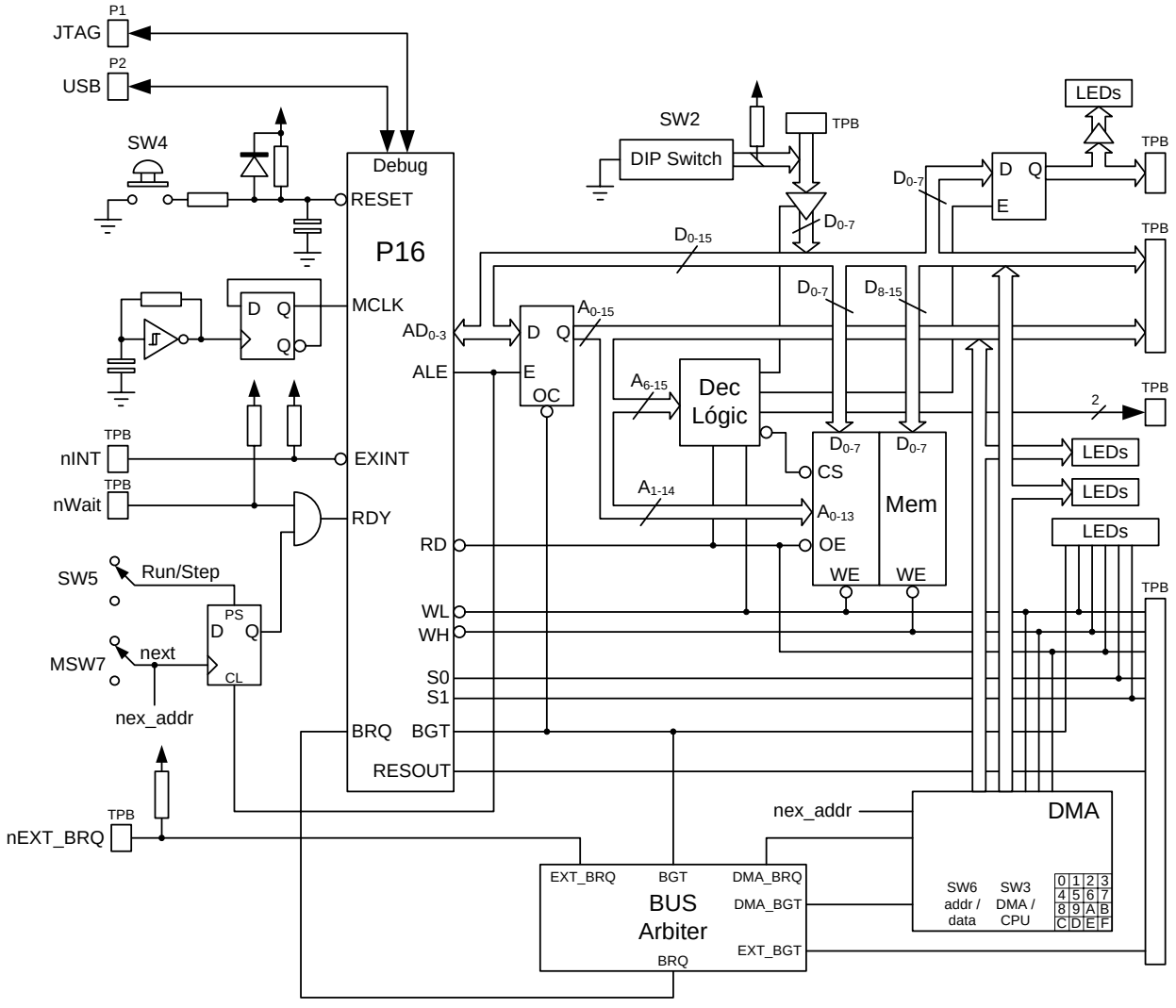


Figura 15-1 Diagrama de blocos do SDP16

15.2.1 PDS16

O processador PDS16 é interligado ao SDP16 através de duas fichas, uma de 34 pinos contendo todos os sinais de dados e controlo do CPU e outra de 10 pinos para ligação à máquina de *debug* através de um canal série assíncrono ou em alternativa um canal JTAG. Na ficha de 34 pinos estão disponíveis os seguintes sinais:

Oscilador 500Hz com *duty cycle* de 50%, determinado por malha de condensador e resistência;

RESETIN Uma malha não linear (resistências, condensador e díodo) assegura o reset automático, no estabelecimento e na falha/restabelecimento da alimentação (*power on* e *power fail*), permitindo também que a acção de reset seja provocada através de um botão de pressão;

Address/Data Bus de 16 bits para endereços multiplexado com dados, validados pelos seguintes sinais:

- ALE (*Address Latch Enable*) valida endereços;
- nRD (*ReaD*) valida dados de leitura de memória;
- nWRL, nWRH (*Write Low e High*) valida dados para escrita na memória. Permite acesso a byte.

BRQ (*Bus ReQuest*) pedido efectuado pelo bus *arbiter* para que o CPU liberte o bus;

BGT (*Bus GranT*) para indicar a disponibilização do bus;

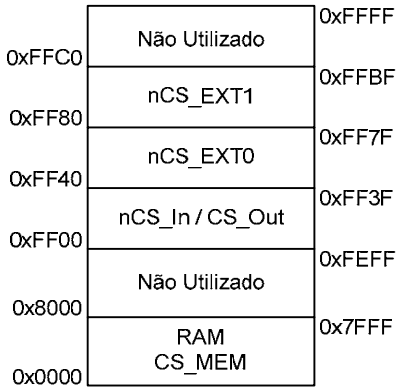
RDY (*ReaDY*) utilizado para realizar execução do CPU em *single step*. Disponível num TBP para levar a que o CPU entre em estado de espera;

EXTINT Entrada externa de interrupção.

15.2.2 Mapa de Memória

Memória instalada 32K*8, de 0x0000 a 0x7FFF, constituída por dois dispositivos RAM 32k*8 que ocupam os primeiros 32K endereços do mapa de memória.

A descodificação é implementada por uma PAL 22v10 com a programação em WinCupl descrita na Figura 15-2 e que disponibiliza para além do CS das RAMs, mais três espaços de 64 bytes, criando o mapeamento presente na Figura 15-2. Os sinais nCS_EXT0 e nCS_EXT1 estão disponíveis em *Tie Point Block*s, no sentido de simplificar a geração de CS, aquando da implementação de trabalhos de laboratório que necessitem inserir dispositivos no espaço de memória do CPU.




```
/* *************** INPUT PINS *********************/
PIN [2..11]  = [A6..15]; /* Endereços de A6 a A15 */
PIN 13  = !RD;
PIN  1  = !WRL;
/* *************** OUTPUT PINS *********************/
PIN 14 = !CS_MEM;
PIN 15 = CS_Out;
PIN 16 = !CS_In;
PIN [17..18] = ![CS_EX0..1];

CS_MEM =!A15;
CS_Out=A15&A14&A13&A12&A11&A10&A9&A8&!A7&!A6&WRL;
CS_In=A15&A14&A13&A12&A11&A10&A9&A8&!A7&!A6&RD;
CS_EX0=A15&A14&A13&A12&A11&A10&A9&A8&!A7&A6;
CS_EX1=A15&A14&A13&A12&A11&A10&A9&A8&A7&!A6;
```

Figura 15-2– Mapa de memória e descrição em CUPL do gerador de CS (PAL U12).

15.2.3 Porto Paralelo de entrada e de saída

O sistema inclui dois portos paralelos de oito bits – um de entrada, ao qual está associado um DIP de 8 interruptores e outro de saída, ligado a oito LEDs conforme a Figura 15-3 e a Figura 15-4. Deste modo é possível realizar alguns trabalhos de laboratório envolvendo I/O, sem necessidade de montagem externa. Os buffers U6A e U6B asseguram que os LEDs de visualização não carregam o porto de saída.

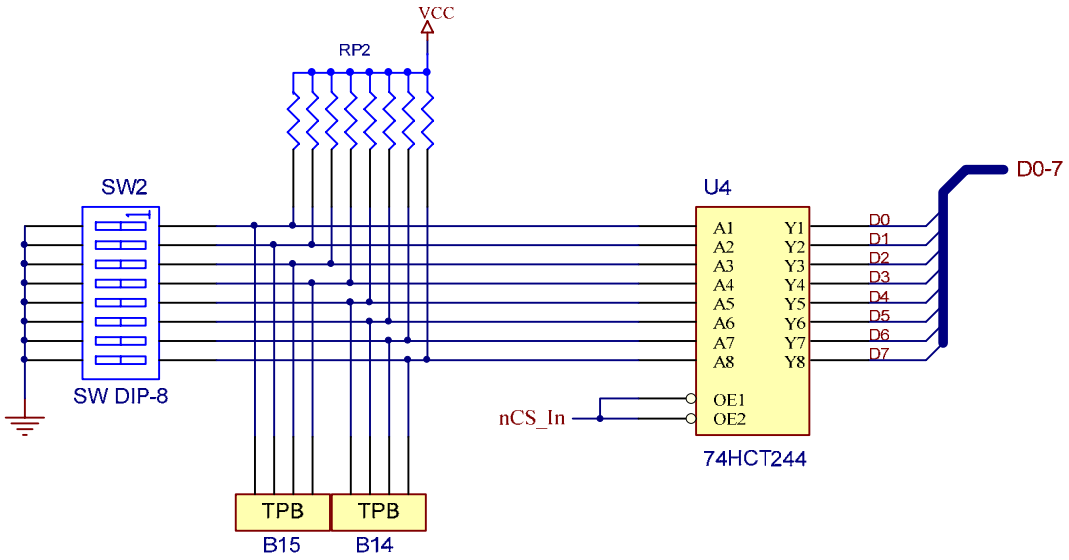


Figura 15-3 - Input Port

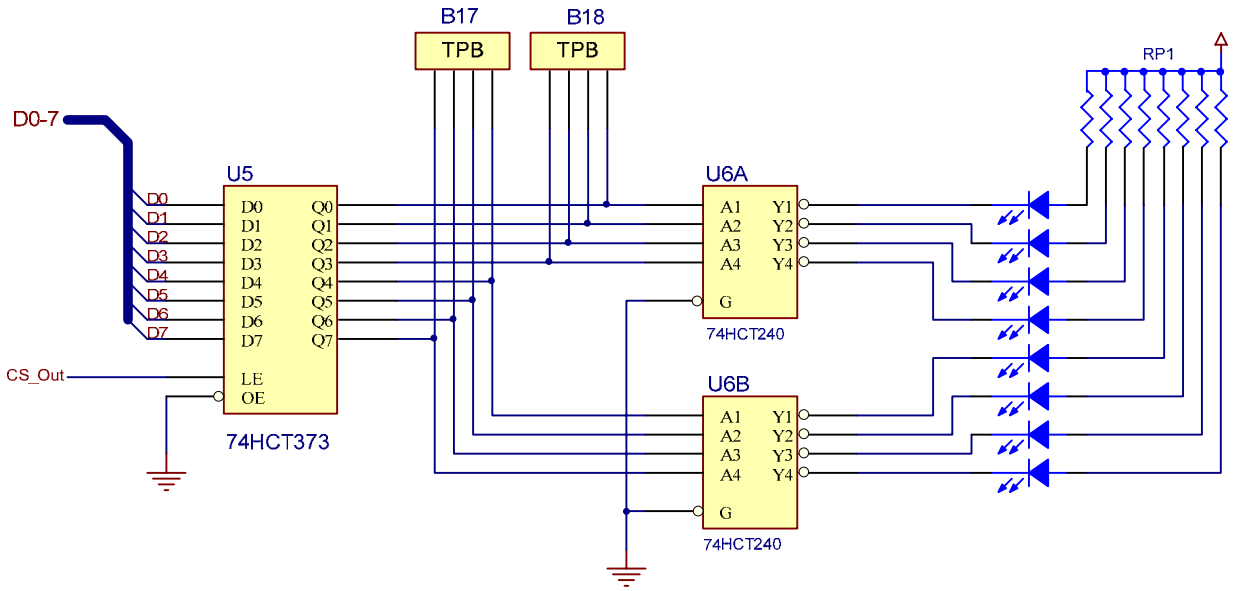


Figura 15-4 - Output Port

15.2.4 Módulo Single Step

O SDP16 dispõe de um módulo cujo esquema é mostrado na Figura 15-5 que, tirando partido do sinal RDY do CPU, permite observar numa fila de LEDs associados aos barramentos de dados, endereço e controlo, a informação neles presente, à medida que o CPU, na persecução das instruções de um programa, vai acedendo à memória para *fetch* de uma instrução ou execução das instruções LD e ST.

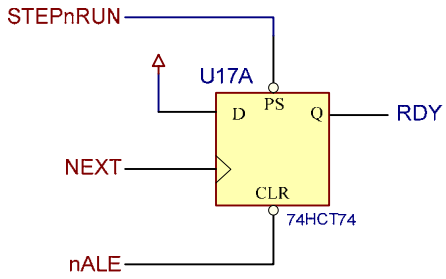


Figura 15-5

Esta funcionalidade permite o estudo dos ciclos do CPU e é particularmente útil no *debugging* do hardware envolvido em trabalhos de laboratório, permitindo, com o auxílio de pontas de prova, detectar erros de montagem ou de projecto.

15.2.5 Mostrador de sinais

Os sinais dos barramentos de controlo, de endereços e de dados são mostrados, através de *line drivers*, em trinta e oito LEDs, 16 de dados, 16 de endereço e 6 de controlo. O SDP16 coloca disponível em *Tie Point Block*s todos os sinais do PDS16 e mais uma série de outros sinais para darem suporte a exercícios de laboratório.

15.2.6 DMA (*Direct Memory Access*)

O SDP16, dispõe de um módulo denominado DMA (*Direct Memory Access*) que, através de um teclado hexadecimal e do mostrador de sinais, permite observar e alterar conteúdos da memória do sistema (ver apêndice A).

O DMA para aceder ao bus do sistema, activa um sinal requerendo o bus e espera que o bus seja libertado. Dado que o bus do sistema é partilhado pelo CPU, pelo DMA e a partir do exterior do SDP16, é necessário realizar a arbitragem dos pedidos de acesso ao bus do sistema. Com este objectivo o SDP16 dispõe de um módulo denominado *Bus Arbiter* integrado no módulo DMA.

Bus *Arbiter*

Dado que o CPU é o responsável pelo bus do sistema, o módulo de arbitragem só necessita gerir dois pedidos de acesso: um proveniente do DMA e outro vindo do exterior do sistema.

O *bus Arbiter* dispõe dos seguintes sinais de entrada e de saída:

nDMA_BRQ	Sinal de entrada proveniente do DMA e que é activado por este quando, através do comutador DMA/CPU, o utilizador requer o acesso para consulta ou escrita na memória;
nBRQ_EXT	Pedido estabelecido por um eventual dispositivo externo;
BGT	Sinal de entrada que é activado pelo CPU para informar o *bus Arbiter* que o bus já se encontra liberto;
DMA_BGT	Sinal de saída para informar o DMA que já pode utilizar o bus do CPU.
BGT_EXT	Sinal de saída para dar BGT ao pedido externo.
BRQ	Sinal utilizado pelo *bus Arbiter* para informar o CPU que um dispositivo pretende aceder ao bus.

Em caso de pedido simultâneo de acesso, o *bus Arbiter* dá prioridade ao pedido do DMA. Quando aceita um pedido, activa a entrada BRQ do CPU e espera que este active a saída BGT. Quando a linha BGT é activada o *bus arbiter* informa o dispositivo que requereu o bus que este já se encontra disponível.

Estrutura do DMA

O DMA do SPD16 é constituído por um teclado hexadecimal organizado em matriz espacial, um registo denominado MBR (*Memory Buffer Register*) de dezasseis bits (4 *nibbles*), um registo denominado MAR (*Memory Address Register*) de dezasseis bits (4 *nibbles*) e um comutador DATAnADDR que determina se quando uma tecla é premida o respectivo código deve ser escrito no MAR ou no MBR. Nestes registos a escrita é precedida de uma acção de deslocamento (*shift left*) a quatro bits da informação ai presente, para que o código da última tecla premida ocupe sempre o *nibble* de menor peso.

O MBR é responsável por manter estável a informação no Data Bus durante a acção de escrita que é promovida pelo controlo do DMA.

Actualiza-se no MAR o endereço da posição de memória a ser acedida para escrita ou leitura. Endereços sucessivos podem ser estabelecidos por acção de um comutador designado por NEXT,

pois o MAR, além da acção de deslocamento já descrito, suporta acção de incremento do conteúdo nele contido.

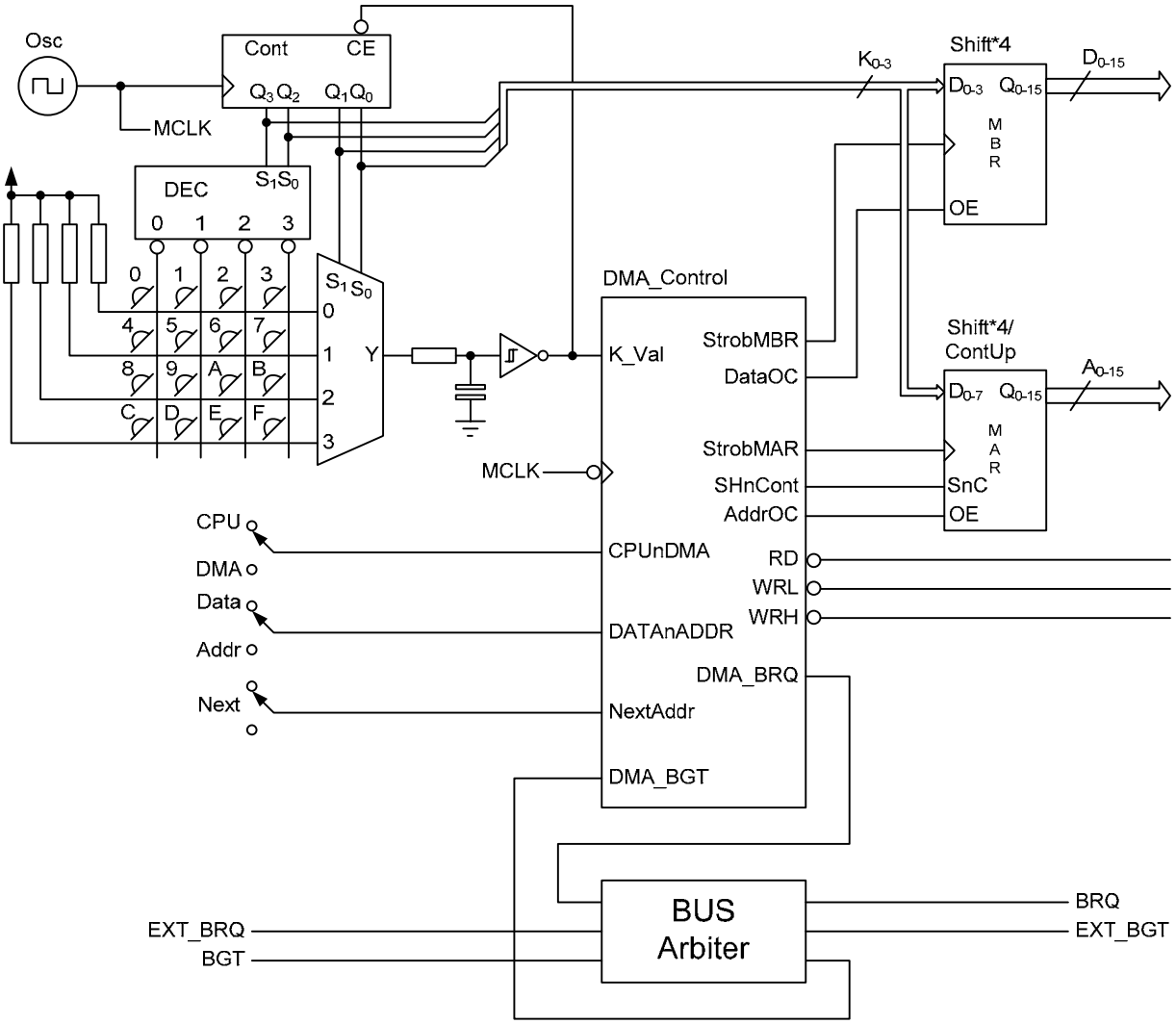


Figura 15-6

15.2.7 Canal de teste USB / JTAG

Para teste dos programas poderá ser utilizada a plataforma computador pessoal PC. Para utilização dessa plataforma é necessária a adaptação de um canal de comunicação do PC ao protocolo que o CPU em teste suportar. O SDP16 já põe disponível um conversor USB para protocolo série assíncrono full duplex START/STOP 9600. Quanto ao JTAG, o SDP16 põe disponível um fixa JTAG standard ligada a quatro pinos do CPU. Para teste dos programas utilizando como interface o computador pessoal, está disponível uma aplicação para Windows denominada PD4Debugger.