9. SÍNTESE DE SISTEMAS SEQUENCIAIS

Consoante a complexidade do sistema a projectar, assim os níveis de abordagem em que este terá de se desenvolver. O nível que iremos agora introduzir, é referido como *random logic*, e consiste em interligar módulos combinatórios e sequenciais de comportamento conhecido, constituindo um ***módulo funcional***, cujo comportamento lhe é determinado por um ***módulo de controlo***. O módulo de controlo recebe para além dos sinais externos ao sistema, sinais provenientes do módulo funcional para assim poder determinar a sequência de acções a realizar sobre o módulo funcional. Cada elemento do módulo funcional pode *de per si* constituir um sistema que foi ou será desenvolvido utilizando este mesmo nível de abordagem, ou seja, o projecto será desenvolvido numa perspectiva *top-down* dividindo sucessivamente e de uma forma hierárquica o projecto em módulos cada vez menos complexos.

Sempre que pretendemos resolver um problema, deparamos com a necessidade de encontrar um algoritmo, ou seja um conjunto ordenado de acções e decisões que nos permita obter uma solução. Embora o algoritmo possa ser encontrado mentalmente, a passagem a uma forma escrita, facilita a estruturação das ideias e a análise da solução encontrada.

A forma encontrada para exprimir o algoritmo tem que ser inteligível, ou seja, não pode conter ambiguidades, susceptível de várias interpretações.

Se pretendermos sintetizar uma implementação hardware a escrita do algoritmo deverá levar em linha de conta o tipo de implementação, no sentido de se encontrar uma solução fazível e ponderada.

Uma das dificuldades que desde sempre se tem posto ao projectista de sistemas, quando se está a conceber um sistema, é a forma como o pode representar. Ficaremos sempre entre a escolha de uma representação gráfica ou escrita. Qualquer das formas que adoptemos deve, como anteriormente foi referido, garantir uma única leitura pelos restantes membros da equipa.

A forma escrita tem a vantagem de poder vir a ser facilmente compilável e não estar sujeita à maior ou menor habilidade gráfica do projectista, mas tem como grande desvantagem o facto de não se poder ter uma ideia do global da solução. São exemplo destes formalismos descritivos, RTL (*Register Transfer Level*), VHDL (*Verbose Hardware Description Language*) e o Verilog. O humano é mais sensível ao grafismo do que à forma escrita, pelo que, a descrição gráfica permite ter uma visão global simples "*The Big Picture*" o que em projecto de sistemas é de extrema importância. Mas na descrição gráfica põem-se dois problemas, se utilizarmos os vários componentes na sua forma esquemática: ou se é demasiado pormenorizado o que a este nível de abordagem é indesejável, ou se não formos pormenorizados o esquema fica ambíguo.

Neste sentido J. B. Dennis e S. S. Patil propuseram um formalismo gráfico que denominaram por *Basic Schemata*, e que se insere como passo intermédio entre a definição do algoritmo genérico e o desenho do diagrama de blocos e do ASM do módulo de controlo. Este formalismo também se mostra muito útil quando necessitamos analisar o comportamento de um determinado circuito sequencial.

9.1 Basic schemata

O *basic schemata* comporta dois diagramas, o EFI (Esquema de Fluência de Informação), e o ESA (Esquema de Sequencia de Acções).

EFI

O EFI corresponde à descrição do bloco funcional também designado por *data path* (caminho dos dados), podendo nele observar-se o fluxo de informação, as interacções entre os vários objectos e a sua relação com os operadores. No EFI os vários operadores podem ser entendidos como os métodos exportados pelos vários objectos que constituem o módulo funcional, como acontece nas linguagens POO.

Registos

Os registos são elementos com memória, permitindo neles armazenar informação. Um facto que é importante realçar é que no EFI não é especificado se o registo é síncrono ou assíncrono. O símbolo gráfico de um registo é um rectângulo inscrevendo-se no seu interior o nome ou referência. A informação contida num registo pode ser intrínseca (constante), ou ser-lhe atribuída por um operador. Os registos podem ser só de leitura, só de escrita ou de leitura e escrita. Na Figura 9-1 está representado um registo, e onde podemos identificar as entradas e saídas de dados.

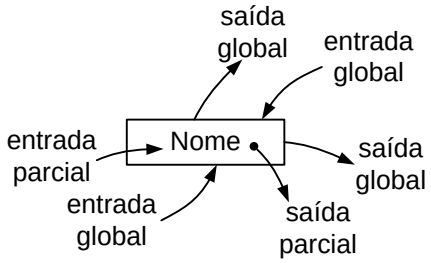


Figura 9-1 - Registo

Operadores

Os operadores são entidades, que quando evocados pelo ESA, realizam a função que lhes está atribuída. A evocação de um operador pode ser feita pela máquina de controlo ou por um agente externo ao sistema. O símbolo gráfico do operador é um circulo, podendo-se inscrever ou não no seu interior qual a função que este realiza. Na parte exterior do círculo inscreve-se o nome pelo qual é evocado. Os operadores podem ser unários, binários ou múltiplos, ou seja podem ter uma, duas ou mais entradas podendo estas ser vectorizadas como mostra a Figura 9-2.

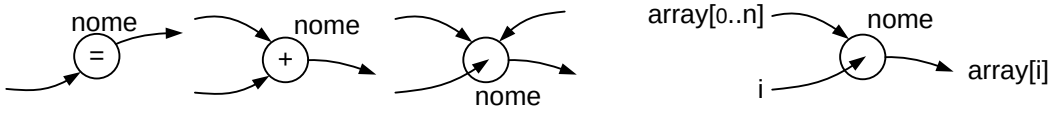


Figura 9-2 - Operador

Se o operador A tem como entrada a saída do operador B, então a evocação de A tem implícita a evocação de B.

Os operadores dividem-se em dois tipos:

Acção — Quando evocados/activados realizam a função que lhes está atribuída afectando um ou mais registos com o valor daí resultante.

Controlo — Quando evocados/activados fazem presente ao módulo de controlo o valor resultante da avaliação da expressão que lhes está atribuída, não sendo por isso, representada a saída.

Os operadores evocados por agentes externos são representados por dois círculos concêntricos. Na Figura 9-3 está representado como exemplo um comutador de nome **com**, e que é um registo que pode tomar o valor lógico 0 ou 1. O seu valor pode ser testado pelo módulo de controlo (ESA), por evocação do operador **test**, e que devolve TRUE se o valor registado é igual a 1. O valor lógico a que o comutador se encontra pode servir por exemplo para ser entrada de selecção de um operador multiplexer de duas entradas.

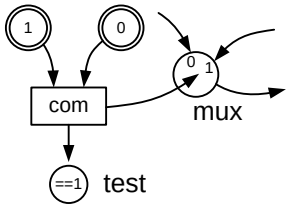


Figura 9-3- EFI de um comutador

ESA

O ESA é o grafo de acções a exercer sobre o EFI, nele podemos observar a sequência pela qual os vários operadores do EFI são activados/evocados, conferindo desta forma um comportamento e uma funcionalidade ao sistema. O ESA corresponde assim ao comportamento do bloco de controlo.

O grafismo utilizado no ESA revela as tendências das linguagens modernas de programação (exemplo Pascal, C, etc..), pois utiliza as primitivas de controlo de fluxo *if then else*, *while do* e *switch case*, e permite a evocação simultânea de vários operadores de acção.

Na Figura 9-4 estão representados os símbolos gráficos utilizados.

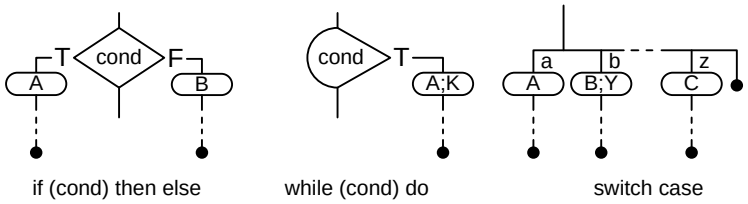


Figura 9-4

9.2 Exemplos

9.2.1 Multiplicador

Tomemos como exemplo o projecto de um multiplicador de dois números de 4 bits pelo algoritmo das somas sucessivas.

$$Mxm=\underbrace{M+M+\ldots+M}_{m}$$

O sistema tem como entradas, dois operandos **M** e **m** de 4 bits, um sinal **S** para dar inicio à operação de multiplicação, uma saída **P** de 8 bits para apresentar o produto e um sinal de saída **RDY** para indicar que a operação foi concluída, ficando assim pronto para efectuar outra multiplicação. Na Figura 9-5 está representada o multiplicador, bem como a relação temporal entre S e RDY.

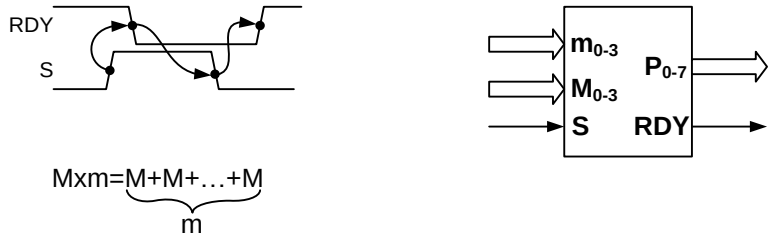


Figura 9-5

A relação entre os sinais S e RDY garante a sincronização entre o sistema multiplicador e o sistema que integra/utiliza o multiplicador. A relação entre cada uma das transições assegura a sincronização independentemente do ritmo de cada um deles.

Poderemos descrever o algoritmo como se segue:

```
boolean RDY,S,OV;
nible M,m,P,i;
while (1) {
  RDY=TRUE;
  while (!S);
  RDY=FALSE;
  P=0;
  for (i=m; i>0; i--) {
    P=P+M;
  }
  While (S);
}
```

Como se pode observar no algoritmo descrito, existem 3 registos de um bit e 4 registo de 4 bits obtendo-se assim o *Basic Schemata* da Figura 9-6. Como o sinal RDY é saída função de estado podemos omitir o registo a ele associado.

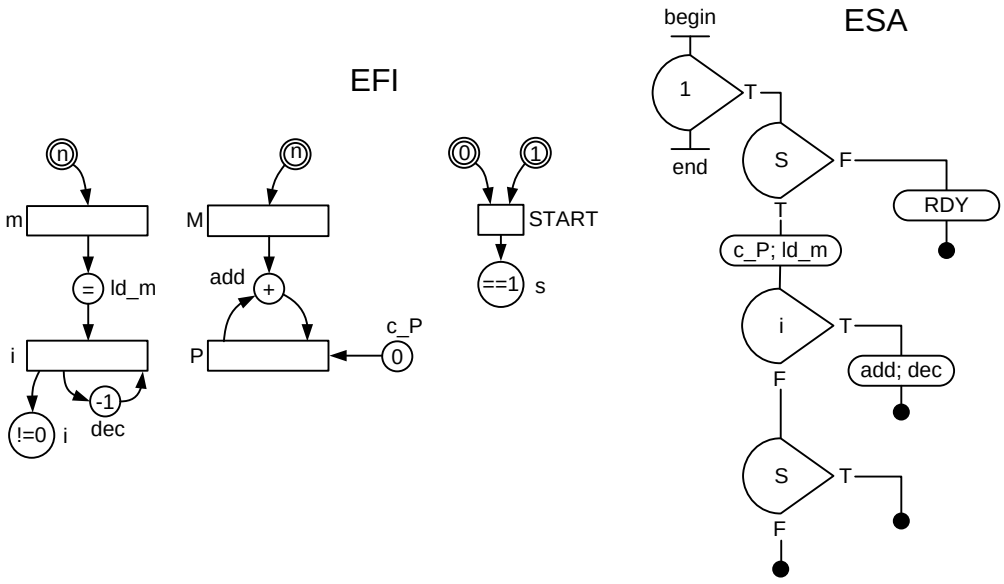


Figura 9-6 – *Basic Schemata* do Multiplicador

Os vários registos que compõem o bloco funcional, suportam diferentes tipos de acções. Por exemplo, o registo associado à variável **i**, suporta que nele registemos um valor, e que o valor nele registado vá sendo decrementado de uma unidade, implicando a utilização de um contador com entrada de PL. Este mesmo registo necessita por disponível informação de *borrow*. O registo associado à variável **P**, para além de receber o resultado da soma suporta a acção de CLEAR.

A Figura 9-7 apresenta uma possível implementação do multiplicador por somas sucessivas. O módulo de controlo obedece ao ASM descrito na Figura 9-7 b).

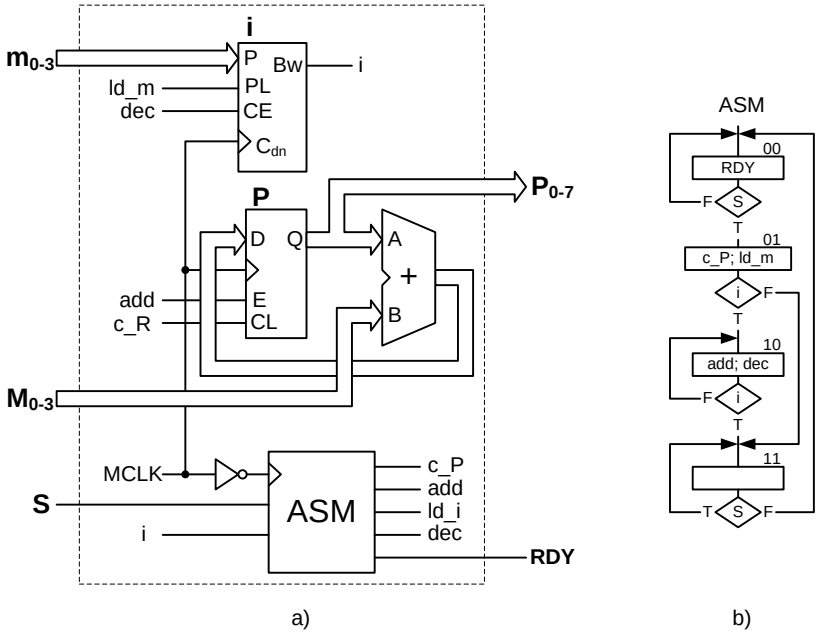


Figura 9-7

Na Figura 9-8 podemos ver o desenho esquemático utilizando um contador com **PL** assíncrono e controlo de contagem **CE**, um somador de operandos de oito bits e um registador *Edge Trigger* com controlo de *Enable* e entrada assíncrona de *Clear*. É necessário que o registo **P** seja *Edge Trigger* por existir realimentação, ou seja, o registo **P** é simultaneamente operando e resultado. Se o registo **P** fosse do tipo *Transparent Latch,* ao activarmos **E,** o resultado da soma transparecia de imediato para a saída do registo **P**, levando a que se realizasse uma nova soma e por conseguinte o registo **P** transparecia para a saída um novo resultado. Esta realimentação ocorreria durante todo o tempo que o sinal **E** se mantivesse activo, tornando indeterminado o resultado.

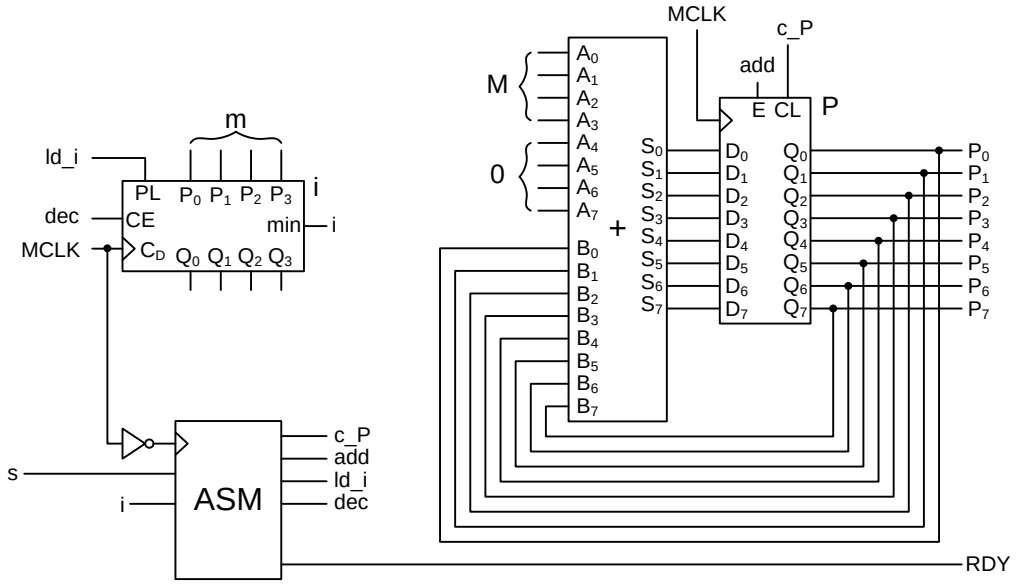


Figura 9-8

9.2.2 Divisor

Um idêntico algoritmo poderá ser usado na divisão, ou seja, dividir, é contabilizar quantas vezes podemos subtrair ao dividendo **D** o divisor **d**. Como estamos a falar da divisão inteira então como resultado será produzido um quociente **Q** e um resto **r**.

Poderemos então descrever o algoritmo como se segue:

```
boolean RDY,S,OV;
nible D,d,r,Q;

RDY=TRUE;
while (!S);
RDY=FALSE;
Q=0;
r=D;
if (d!=0){
  While (r >= d) {
    r=r-d;
    Q++;
  }
  OV=FALSE;
}
else
  OV=TRUE;
While (S);
```

Dado o algoritmo descrito, podemos descrever o módulo divisor através do *Basic Schemata* da Figura 9-9.

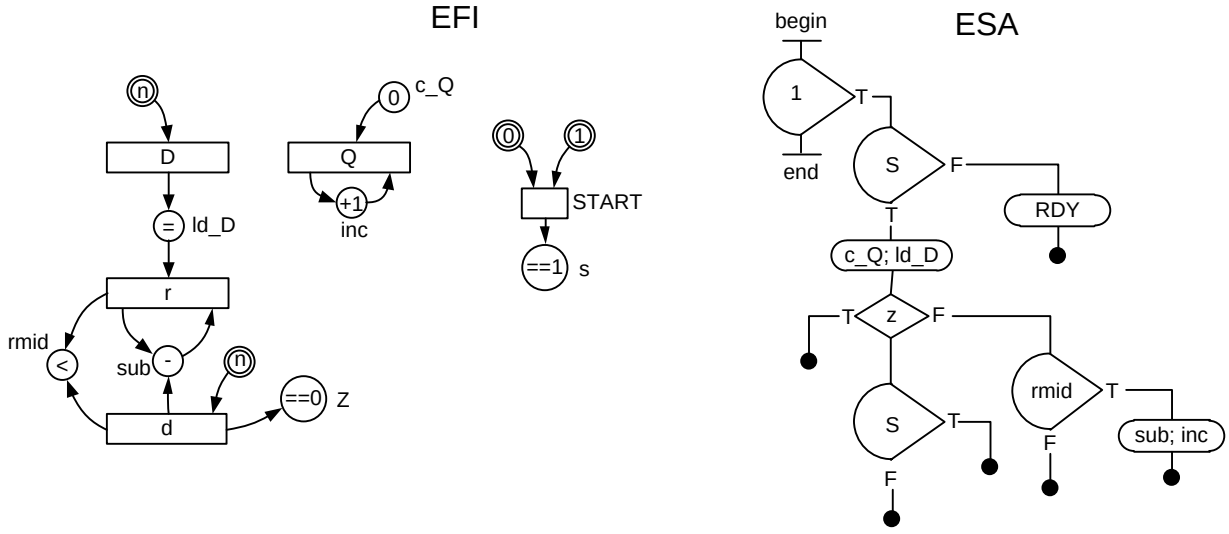


Figura 9-9

A Figura 9-10 apresenta uma possível implementação de um divisor por subtracções sucessivas. O sinal BW (*Borrow*) do subtractor, fica activo quando r < d, ou seja, quando !(r >= d) levando a que não seja necessário utilizar um comparador para relacionar r com d.

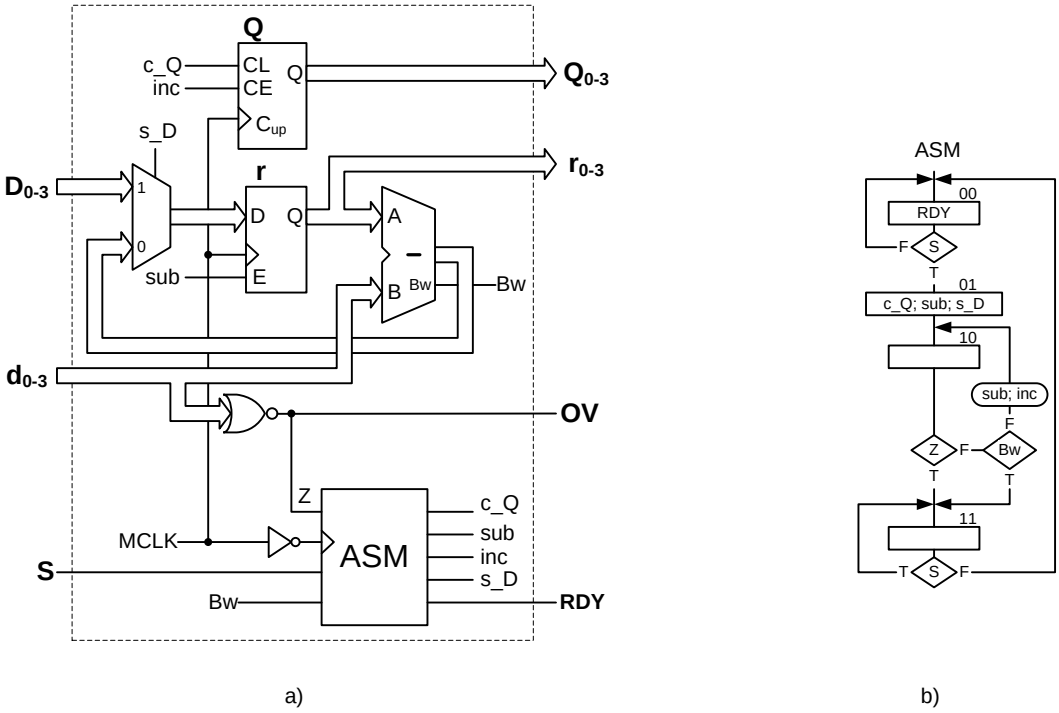


Figura 9-10

9.2.3 Multiplicador/Divisor

Dada a semelhança das estruturas poderemos pensar numa máquina que realize uma multiplicação ou uma divisão, em função de um sinal de entrada MnD (multiplica/divide). Na Figura 9-11 está representado *Basic Schemata* do módulo multiplicador/divisor.

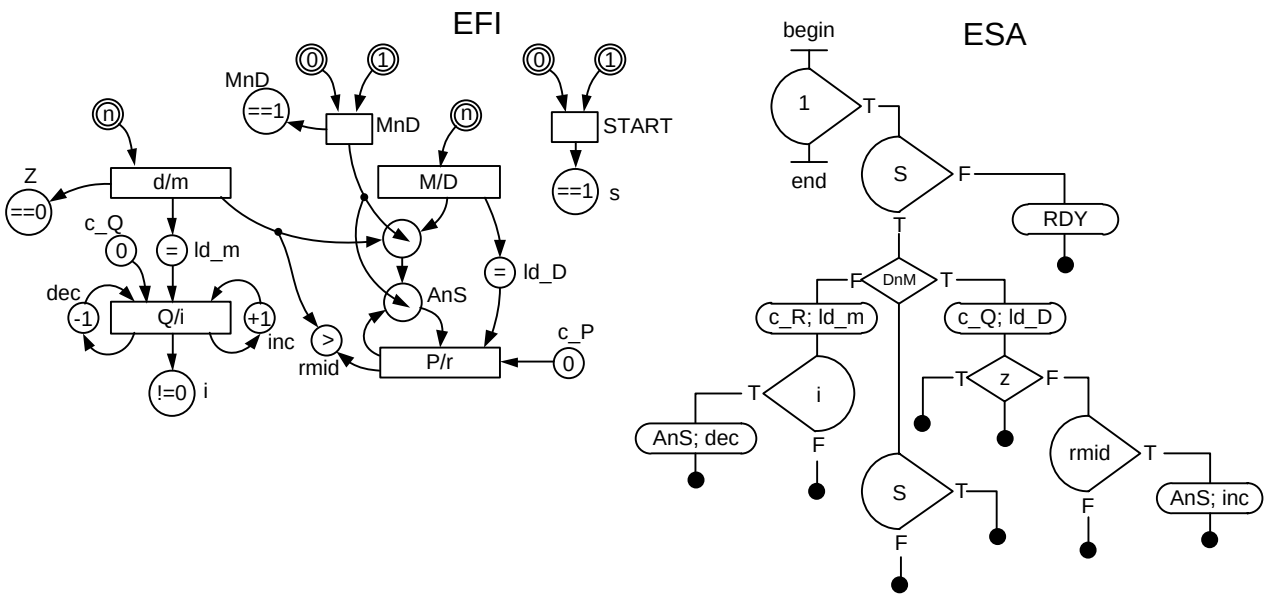


Figura 9-11

Dado o EFI e o ESA da Figura 9-11 obtemos o diagrama de blocos e o ASM da Figura 9-12.

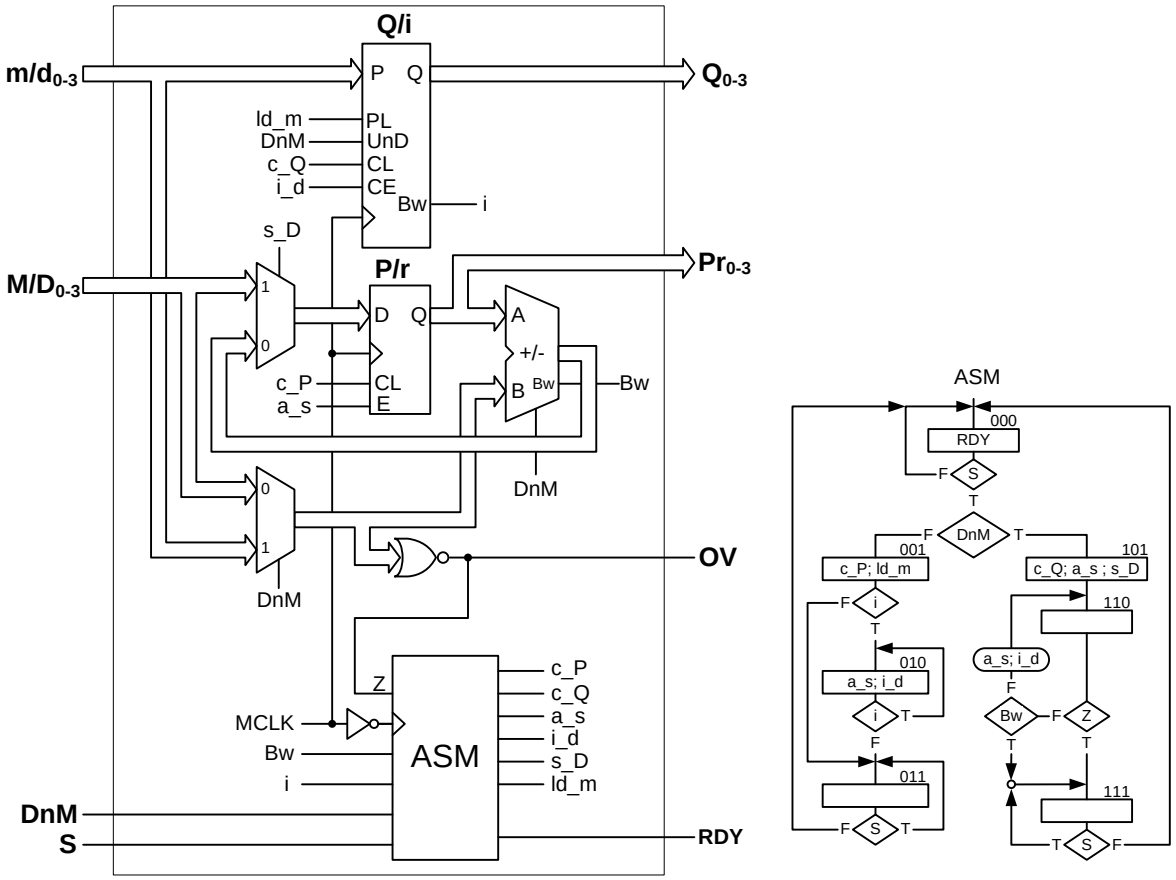


Figura 9-12

9.3 Micro Programação

Existe uma grande variedade de técnicas para a implementação de uma máquina de estados. A utilização de cada uma dessas técnicas depende de vários factores, nomeadamente a tecnologia que estivermos a utilizar, a característica do ASM, etc. Como temos visto até aqui, as expressões lógicas dos sinais de controlo são implementadas por portas lógicas ou mesmo por PLDs, diz-se neste caso que o controlo tem uma implementação *hardwire*d. Em estruturas mais complexas esta tarefa pode ser difícil de implementar e testar para as possíveis combinações de entrada e estado. Em 1953 o cientista Maurice Wilkes propôs uma outra técnica de implementação denominada por controlo micro programado, e que consiste em guardar em memória uma sequência de palavras binárias que se denominam por micro-instruções. Cada um dos bits da micro-instrução especifica uma micro-operação a ser realizada sobre o bloco funcional. A sequência das micro-instruções é denominada por micro-programa.
Este modelo de implementação é sugerido.
O micro-programa é normalmente armazenado numa ROM. O conteúdo de uma palavra armazenada num determinado endereço da ROM, especifica as micro-operações a serem realizadas nesse estado sobre bloco funcional e determina também o comportamento do bloco de controlo função do estado corrente e das entradas. Se admitirmos como já anteriormente foi referido que com uma ROM podemos implementar qualquer função, o modelo geral do controlo microprogramado é idêntico ao modelo geral de uma máquina de estados como mostra a Figura 9-13.

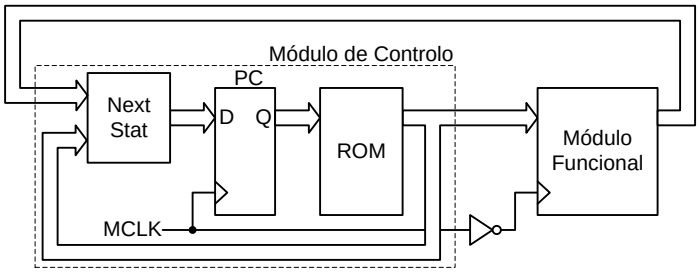


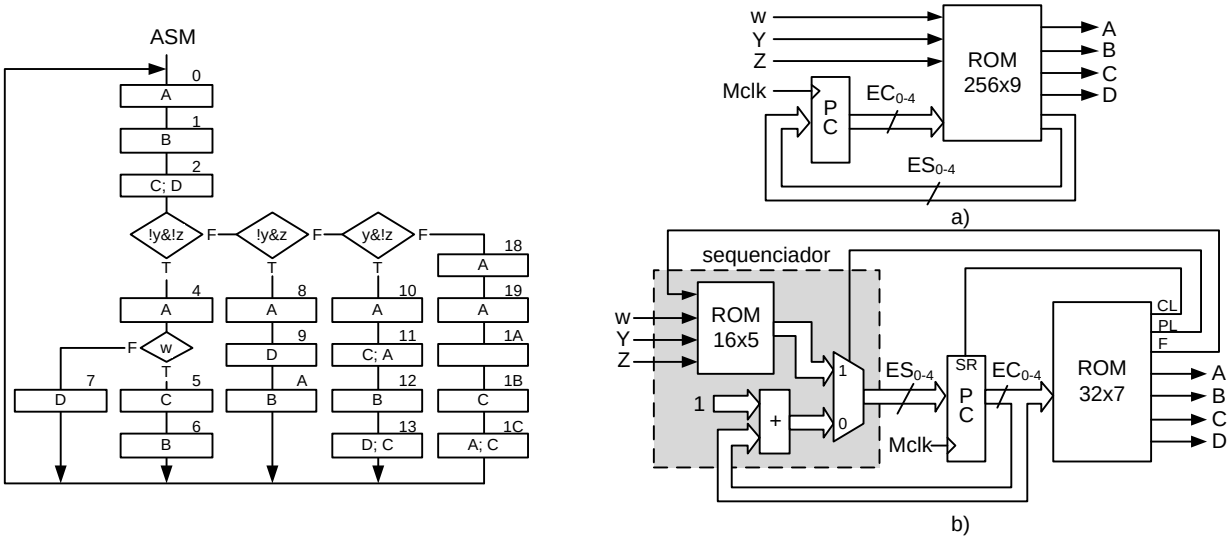
Figura 9-13 – Modelo geral de uma Máquina de estados Micro-programada

É evidente que esta solução pode implicar uma dimensão de ROM muito grande, com muitos dos termos produto não utilizados. No sentido de diminuir a dimensão da ROM, e quando estamos perante um ASM com sequência de estados não condicionados a variáveis de entrada, é possível optimizar o controlo, substituindo o registo de endereço por um contador com entrada de *Parallel Load* PL a que denominaremos por **PC** (*Program Counter*). Com esta solução, a ROM em vez disponibilizar todos os estados, passa a disponibilizar um bit de saída que indica ao contador se existe sequência ou transição de estado (*Switch Case*).
Esta implementação é denominada por controlo mico-programado com sequenciador, e tem sido muito utilizada na síntese do módulo de controlo dos processadores.

9.3.1 Sequenciador

O sequenciador tem o seguinte comportamento: o contador selecciona o endereço da ROM de onde vai ser lida a instrução a ser executada, cada um dos bits que constituem a instrução, determina se uma determinada micro-operação é executada, o condicionador de estado seguinte controla a entrada **PL** do contador, determinando se a máquina se encontra em sequência ou comutação de estado.

Tomemos como exemplo o ASM apresentado na Figura 9-14 que representa o comportamento de um sistema com três entradas y, w, z e quatro saídas A, B, C, D. Este ASM tem um comportamento que se adequa a este tipo de implementação, ou seja, uma sequência de acções, uma decisão múltipla (*Switch Case*) para quatro diferentes sequências e uma decisão simples entre dois estados. A implementação microprogramada utilizando uma ROM que gerasse as quatro saídas A, B, C e D mais o código do estado seguinte X_{0-4}, implica uma ROM de dimensão 256x9 como mostra a Figura 9-14 a). A implementação microprogramada com um circuito sequenciador como é apresentado na Figura 9-14 b) embora utilize duas ROMs representa uma significativa economia em dimensão, pois uma das ROMs, tem dimensão 32x7 e a outra 16x5. A acção CL (*clear*) no registo PC é síncrona para garantir um tempo de estado.



A4	A3	A2	A1	A0	D6	D5	D4	D3	D2	D1	D0
EC4	EC3	EC2	EC1	EC0	D	C	B	A	F	PL	CL
0	0	0	0	0	0	0	0	1	-	0	0
0	0	0	0	1	0	0	1	0	-	0	0
0	0	0	1	0	1	1	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1	1	0
0	0	1	0	1	0	1	0	0	-	0	0
0	0	1	1	0	0	0	1	0	-	0	1
0	0	1	1	1	1	0	0	0	-	0	1
0	1	0	0	0	0	0	0	1	-	0	0
0	1	0	0	1	1	0	0	0	-	0	0
0	1	0	1	0	0	0	1	0	-	0	1
1	0	0	0	0	0	0	0	1	-	0	0
1	0	0	0	1	0	1	0	1	-	0	0
1	0	0	1	0	0	0	1	0	-	0	0
1	0	0	1	1	1	1	0	0	-	0	1
1	1	0	0	0	0	0	0	1	-	0	0
1	1	0	0	1	0	0	0	1	-	0	0
1	1	0	1	0	0	0	0	0	-	0	0
1	1	0	1	1	0	1	0	0	-	0	0
1	1	1	0	0	0	1	0	1	-	0	1

	ES
y,z	4,8,10,18
w	5,7

Figura 9-14 – Implementação com sequênciador

EC_{0-4} Cinco bits provenientes do PC (*Program Counter*)

ES_{0-4} Cinco bits que indicam qual o próximo estado.

9.4 Frequência máxima

Um factor importante a ter em conta no projecto de um circuito sequencial é a frequência máxima a que o circuito pode funcionar, ou seja, qual a frequência máxima do sinal de *clock*. Como podemos observar na Figura 9-15, no modelo geral do circuito sequencial que temos vindo a estudar, após uma transição ascendente do sinal de *clock* no registo de estado, desencadeia-se a seguinte sequência de acontecimentos: propagação no registo de estado, descodificação das saídas pela ROM, reacção do módulo funcional aos novos valores de entrada e geração do estado seguinte função dos sinais gerados pelo módulo funcional.

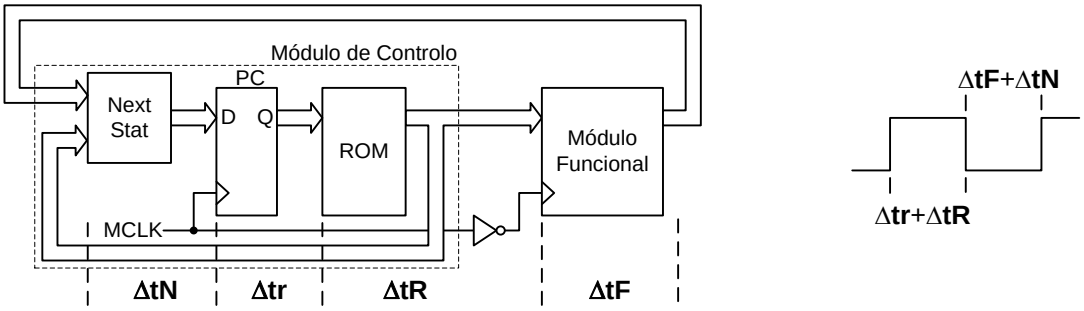


Figura 9-15

Alguns destes elementos têm tempos de reacção muito grandes, por exemplo a ROM e o módulo funcional caso contenha somadores e outros elementos iterativos, podem ter uma relação de duas a quatro vezes o tempo de propagação de um registador, o que leva a que a frequência máxima de funcionamento não possa ser muito elevada. A forma como até aqui temos implementado o esquema de sincronização implica um desfasamento de 180º entre o módulo de controlo e módulo funcional, o que implica um período T do sinal MCLK igual ao maior dos dois valores $T = 2 * (\Delta tr + \Delta tR)$ ou $T = 2 * (\Delta tF + \Delta tN)$. A multiplicação por 2 é admitindo um *duty cycle* de 50%, ou seja o patamar a 1 tem uma duração igual ao patamar a 0.

Se colocarmos um registo à saída da ROM do micro-code como mostra a Figura 9-16, poderemos diminuir o tempo de realimentação de informação permitindo desta forma, aumentar a frequência máxima de funcionamento.

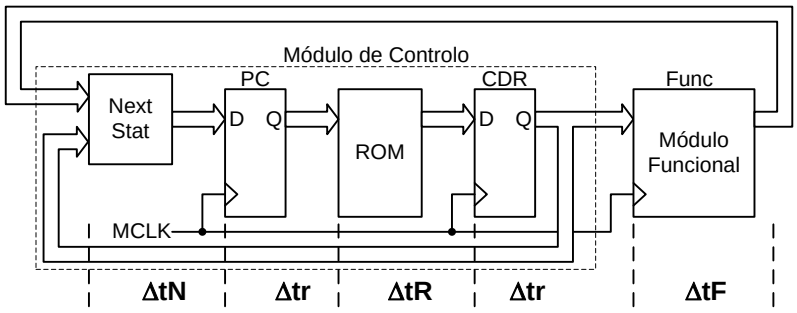


Figura 9-16

Com a introdução deste registo passaremos a ter uma propagação $\Delta tr + \Delta tF + \Delta tN$ ou $\Delta tr + \Delta tR$. Para podermos comparar o desempenho obtido consideremos $\Delta tr = 20, \Delta tR = 100, \Delta tF = 80, \Delta tN = 50$. No primeiro caso teríamos um período $T = 2 * (80 + 50) = 260$, enquanto que na nova solução teríamos $T = 20 + 80 + 50 = 150$, o que permite uma frequência $(260/150)$ vezes mais alta. A introdução deste registo embora torne a máquina mais rápida, tem como desvantagem a complexidade do desenho do ASM, ou seja,

a sequência das micro-instruções tem que levar em linha de conta o desfasamento de 3 ciclos de *clock* entre o registo de estado e os registos do módulo funcional, dado que o conjunto dos três elementos pode ser visto como um *shift register* de três estágios. Este facto é mais relevante, quando uma decisão no ASM depende de sinais oriundos do módulo funcional. Para tornar este facto mais evidente, é apresentado na Figura 9-17 o ASM do módulo de controlo do multiplicador por somas sucessivas anteriormente estudado. Como se pode observar a acção **ld**; **cl** só chegam ao módulo funcional no estado 3. Por esta razão, o teste para verificar se o multiplicando m é igual a zero, só pode ser efectuado na transição do estado 3 para o estado 4. A razão da saída RDY ser função de estado e entrada prende-se com o facto de o protocolo implicar a desactivação imediata do sinal RDY com a activação do sinal S. Se RDY fosse saída função de estado teria que ser activada no estado 5, e por outro lado esta só seria desactivada no estado 2.

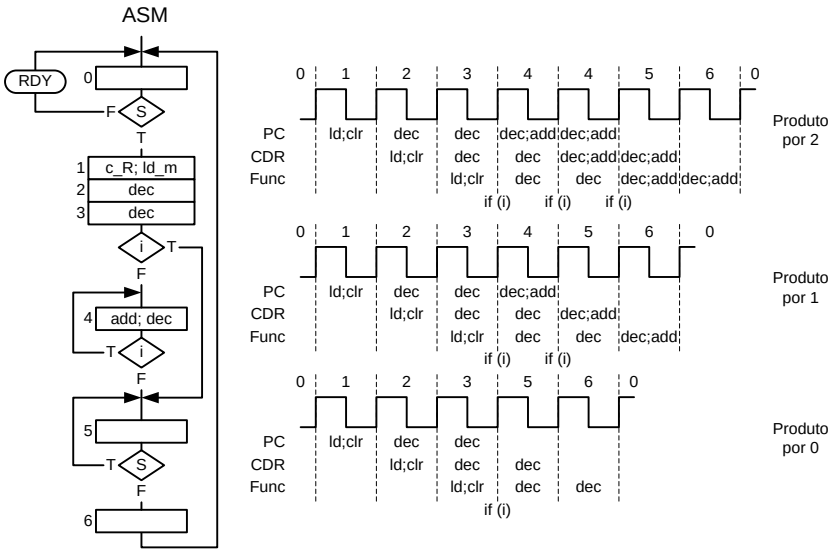


Figura 9-17

Este modelo, tem um comportamento conhecido como *pipeline*, pela semelhança que o seu comportamento apresenta relativamente aos tubos de transporte de líquidos. Quando num tubo que se encontra completamente cheio, se insere uma qualquer quantidade de líquido num dos extremos, implica a saída dessa mesma quantidade no outro extremo do tubo. Como podemos observar no ASM da Figura 9-17 os primeiros estados são para encher o tubo, os últimos estados são para vazar o tubo.

Este modelo só surte um efeito apreciável, quando o algoritmo que estamos a implementar é iterativo, ou seja, quando existe a repetição em grande número de uma mesma acção, de outro modo o modelo é contra producente, pois pode tornar o processo mais lento do que na arquitectura anteriormente estudada.

9.5 Memória RAM

Se pretendêssemos implementar uma máquina capaz de realizar o cálculo de uma expressão envolvendo multiplicações e divisões entre vários operandos seria necessário dispor de um memória que suporta-se escrita e leitura e onde se armazenariam os operandos e os resultados intermédios, tornando o sistema mais flexível.

Os dispositivos de memória que iremos abordar neste capítulo constituem actualmente a memória central dos computadores, suportam leitura e escrita e têm como principal característica a sua organização sob a forma de um vector (matriz unidimensional) suportando acesso aleatório a cada um dos seus conteúdos como acontece com as ROMs. Este tipo de memória é denominado por RAM (*Random Access Memory*) por contraste com as memórias de acesso sequencial como é o caso dos discos de memória (CD e DVD) e a fita magnética, em que para se aceder a um dado é necessário percorrer os dados anteriores.

A memória RAM é constituída por 2^n registos de m bits cada, onde são escritos dados para posterior consulta. A cada registo corresponde um endereço que permite a invocação do respectivo conteúdo, para actualização ou consulta. O nome de um registo equivale ao seu "endereço" que pode ser estabelecido de forma aleatória através de um número codificado em binário.

Existem memórias RAM de várias tecnologias, sendo as mais vulgares as RAMs dinâmicas (DRAM) que baseiam o seu funcionamento na carga acumulada por elementos capacitivos, e a RAM estáticas (SRAM) constituídas por *flip-flops*. No âmbito deste documento iremos apenas abordar as SRAMs, por serem de mais simples funcionamento e mais comuns nos sistemas baseados em microprocessadores de pequena complexidade.

Como mostra a Figura 9-18, a SRAM tem um barramento de endereço (*Address*) unidireccional, constituído por n bits que codificam em binário o endereço. Quanto ao barramento de dados (*Data*), pode ser constituído por dois barramentos unidireccionais, sendo um para leitura e outro para escrita, ou simplesmente um barramento bidireccional para leitura e escrita. O número de linhas **m** que constituem o barramento de dados é igual ao número de bits de cada registo, permitindo desta forma numa escrita ou leitura ter acesso em simultâneo a todos os bits de um registo.

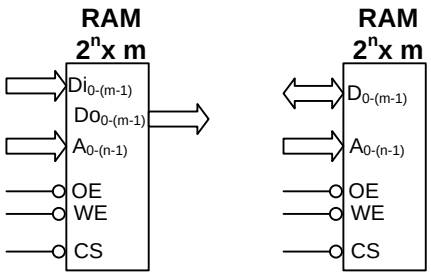


Figura 9-18

Os dispositivos SRAM postos disponíveis no mercado, podem apresentar para o controlo da leitura e da escrita, um único sinal RnW que quando a 1 determina leitura e quando a zero escrita, ou dois sinais independentes, um que determina leitura OE (***O**utput **E**nable*) e outro a escrita WE (***W**rite **E**nable*) com prioridade sobre a leitura. Com o objectivo de diminuir o consumo e permitir a expansão da memória de um sistema, através da concatenação de vários destes dispositivos, é usual estes dispositivos disponibilizarem um sinal de entrada para inibição/desinibição CS (***C**hip **S**elect*), que quando desactivo torna a SRAM insensível aos sinais de leitura e escrita e coloca o barramento de dados em alta impedância.

Tal como acontece com a ROM, a sua referência é dada pelo número de registos que possui e pelo número de bits de cada registo. Assim uma SRAM 2048x8 tem 2048 registos, contendo cada registo 8 bits. Dado que a dimensão da RAM pode ser elevada, é usual utilizar-se na designação, múltiplos de bit 2^{10} K (*Kilobit*), 2^{20} M (*Megabit*), 2^{30} G (*Gigabit*), etc. No caso da SRAM anteriormente referida receberia a designação 2Kx8.
Com o objectivo de diminuir a complexidade o consumo e os tempos de acesso existem uma inúmera variedade de arquitecturas. Na Figura 9-19 podemos ver o esquema de uma SRAM de 16x4, com sinais de leitura e escrita independentes. Os registos que constituem a SRAM são usualmente de natureza *latch*. A arquitectura apresentada na Figura 9-19 é meramente conceptual não corresponde a uma implementação real, principalmente no concerne à descodificação de endereços e à organização das várias células de memória.

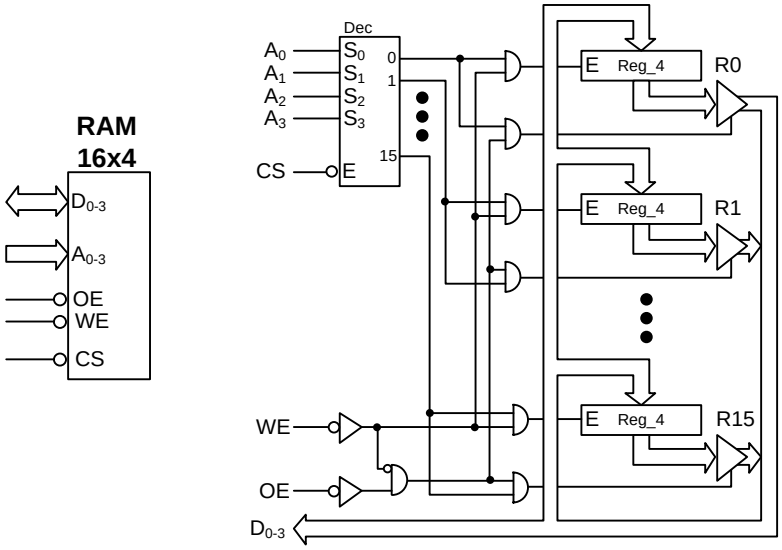


Figura 9-19

No sentido de diminuir a dimensão do descodificador, que no caso de uma memória de 32Kx8 corresponderia a um descodificador com 32K saídas, é utilizada uma matriz espacial, para o controlo da inibição/desinibição de cada registo como mostra a Figura 9-20, que em vez de utilizar um descodificador com 64 saídas, utiliza dois descodificadores de 8 saídas. Esta solução torna-se mais relevante, quanto maior for a dimensão da memória.

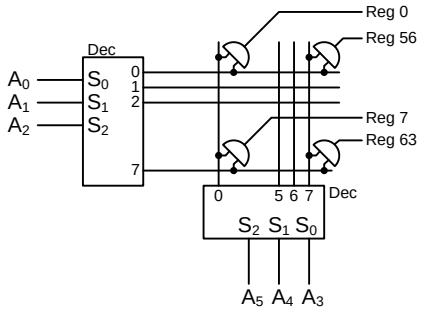


Figura 9-20 – Descodificação para 64 registos

As implementações comerciais, no sentido de diminuir o número de transístores, utilizam uma estrutura designada por *bit slice* (fatias de bits). Nesta estrutura as várias células de memória são organizadas numa

matriz quadrada dividida em fatias (*slice*) e é utilizada na multiplexagem do bus de dados, em vez de *buffers tri-state*, uma ligação em *wired or*. Para melhor se entender a estrutura, vamos primeiramente analisa-la do ponto de vista lógico. Comecemos por observar uma dessas fatias como mostra a Figura 9-21 a). Cada célula de memória é constituída por um *flip-flop* SR controlado por uma das linhas de descodificação. A linha de descodificação controla não só a escrita através das duas intercepções de S e R, como também a intercepção de saída. A saída é controlada por uma porta em *open drain* para poder constituir um bus multiplexado em *wired or* (ver Capitulo 2). São estas três intercepções que representam a intercepção entre uma linha e uma coluna da descodificação mostrada na Figura 9-20. O descodificador das colunas associa *slices*, numa quantidade igual ao número de linhas que constituem o data bus como mostra a Figura 9-21 b).

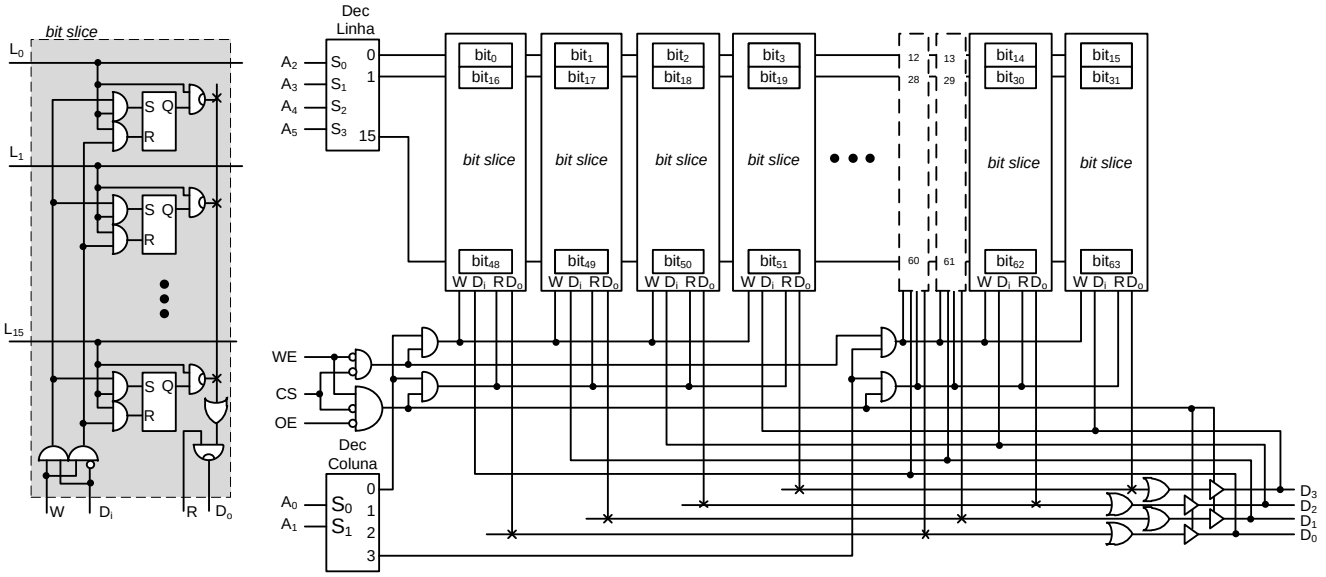


Figura 9-21

A capacidade deste tipo de memória aumentou drasticamente a partir de 1991 com o aparecimento de uma estrutura conhecida por T6 da autoria de *Frank Wanlass* cuja patente pode ser consultada em www.patentgenius.com/patent/5020028.html#show-last-page. Esta designação deve-se ao facto da célula ser formada por seis transístores como mostra a Figura 9-22. Para poder utilizar o mesmo bus para escrita e leitura e devido à dinâmica associada aos transístores CMOS estes dispositivos não são totalmente estáticos, ou seja, utilizam um esquema de refrescamento em que a leitura é momentânea para não destruir a informação armazenada na célula e o valor lido é armazenado numa célula de saída. Os transístores T5 e T6, que são responsáveis pela escrita e leitura, são transístores de passagem, ou seja, permitem a condução nos dois sentidos.

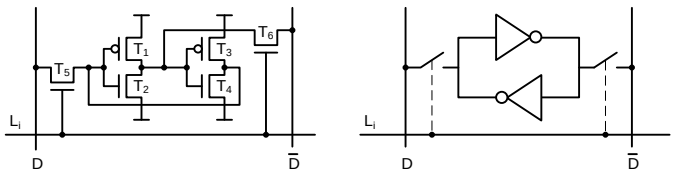


Figura 9-22

Dada a estrutura da memória SRAM, são apresentados na Figura 9-23 e Figura 9-24 os diagramas temporais de um ciclo de leitura e escrita. As siglas utilizadas para referir os tempos (t_{xxx}) são as normalmente utilizadas pelos fabricantes deste tipo de dispositivo.

9.5.1 Ciclo de leitura

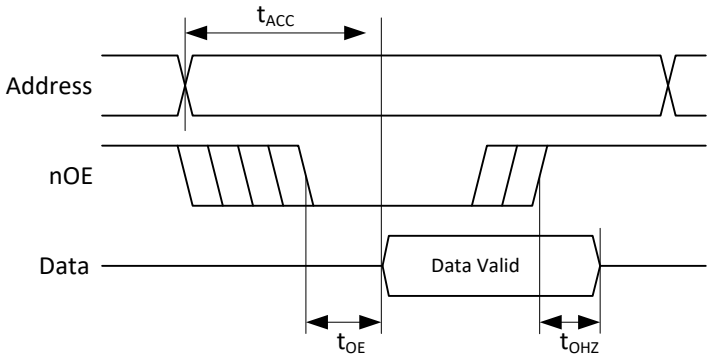


Figura 9-23 – Ciclo de leitura

t_{ACC} (*time ACCess*) tempo mínimo de acesso à informação.

t_{OE} (OE to *Output valid*) tempo que media entre a activação do sinal nOE e a presença no data bus, de informação estável em baixa impedância posta disponível pelo dispositivo.

t_{OHZ} (*Output Disable to Output High Z*) tempo que media entre a desactivação do sinal nOE e libertação do data bus por parte do dispositivo.

9.5.2 Ciclo de escrita

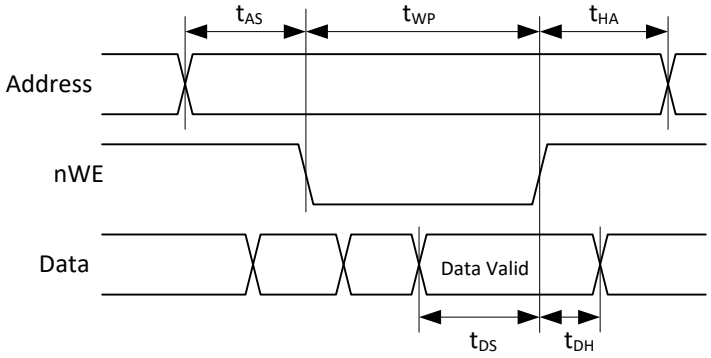


Figura 9-24 – Ciclo de Escrita

t_{AS} (*Address Setup Time*) intervalo de tempo mínimo a respeitar entre o estabelecimento de um endereço e a activação do sinal nWE.

t_{WP} (*Write Pulse Width*) duração mínima do sinal nWE.

t_{DS} (*Data Setup Time*) intervalo de tempo mínimo a respeitar entre o estabelecimento de informação valida no data bus e a desactivação do sinal nWE.

t_{DH} (*Data Hold Time*) tempo mínimo durante o qual ainda se torna necessário manter os dados estáveis no bus após ter terminado o sinal de nWE.

t_{HA} (*Hold Address*) tempo mínimo durante o qual ainda se torna necessário manter o endereço estável após ter terminado o sinal de nWE.

9.5.3 DMA (*Direct Memory Access*)

Tomemos como exemplo o projecto de um sistema que denominamos por DMA (*Direct Memory Access*), que através de um teclado de 16 teclas (0 a F) e uma fila de 8 LEDs, realiza o acesso a uma memória suportando escrita e leitura de dados numa RAM de 16x8. O sistema permite estabelecer através do teclado o endereço que se pretende aceder para escrita ou leitura, servindo o mesmo teclado para estabelecer os dados a serem escritos na RAM. O sistema disponibiliza uma entrada AnD (ADDRnDATA), que determina se os dados produzidos pelo teclado se destinam a estabelecer um endereço cujo conteúdo de memória vai ser lido, ou constituem dados a serem escritos na RAM no endereço previamente estabelecido. Sempre que se estabelece um endereço ficam automaticamente disponíveis na fila de LEDs o valor binário armazenado nesse endereço. Para facilitar a operação sobre o sistema, fica também disponível uma entrada denominada NEXT, que a cada activação incrementa o valor do endereço anteriormente estabelecido.
Dado que durante um ciclo de escrita ou leitura na RAM o endereço tem que permanecer estável, é necessário adicionar à estrutura um registo que denominaremos por MAR (*Memory Address Register*).
Para registar a informação que se pretende escrever na memória e a informação que se lê da memória, adicionamos um registo de 8 bits e que denominaremos por MBR (*Memory Buffer Register*). Dada a funcionalidade pretendida para o DMA, poderemos descrever o seu comportamento através do seguinte troço de programa:

```
boolean k_val,next,add;
nible MAR,key;
byte MBR,mem[16];

while (true) {
   while (!k_val){
     if (next){
       MAR++;
       while (next);
     }
   }
   if (addr){
     MAR=key;
     MBR=mem[MAR];
   }
   else {
     MBR=(MBR << 4) | key;
     mem[MAR]=MBR;
   }
   While (k_val);
}
```

Os dados a serem escritos são construídos por concatenação e deslocamento das várias teclas que vão sendo premidas, ou seja, a inserção da nova tecla faz-se nos 4 bits de menor peso depois de se ter deslocado 4 bits para a esquerda o byte que estava no registo MBR.
Dado o algoritmo anteriormente descrito, podemos inferir o *Basic schemata* descrito na Figura 9-25. Baseado no algoritmo anteriormente descrito, começamos por colocar no EFI os diferentes registos declarados, e de seguida os operadores que sobre eles actuam.

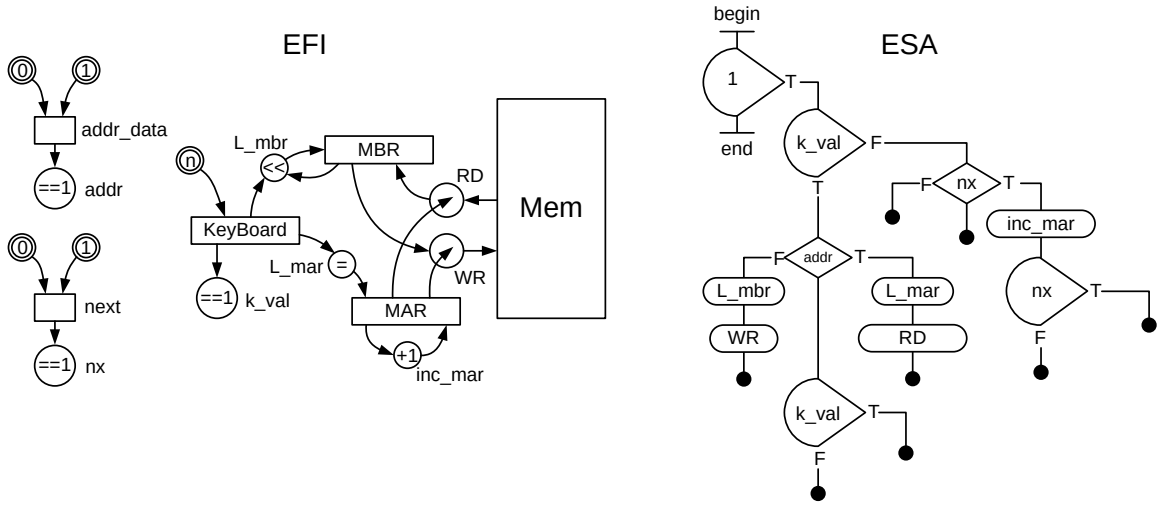


Figura 9-25 – *Basic schemata* do DMA

Quanto à implementação teremos que considerar primeiramente qual a tecnologia a utilizar (*random* ou CPLD), e função da tecnologia serão adoptadas diferentes estratégias. No caso de adoptarmos *random* teremos que ter um conhecimento exacto de qual a funcionalidade dos vários dispositivos MSI disponíveis no mercado, se adoptarmos por CPLD necessitamos saber qual o tipo de *flip-flops* que temos disponíveis e que acções podemos realizar sobre eles. Para mais simples compreensão, a implementação que em seguida se descreve tende a ser independente da tecnologia que venhamos a adoptar, ou seja, introduzimos nos vários elementos as entradas que necessitarmos. Porque os registos disponíveis em alguns dispositivos CPLD, não suportam acções assíncronas de SET, optaremos por acções exclusivamente síncronas, com recurso ao *clock* desfasado de 180º (*clock* invertido).
A Figura 9-26 representa uma possível implementação do DMA. Como se pode observar existe uma enorme abstracção quando se representa em *Basic schemata* relativamente ao esquema de implementação, tornando evidente a utilidade deste tipo de descrição. A implementação sugerida pressupõe a sintetização em PAL.

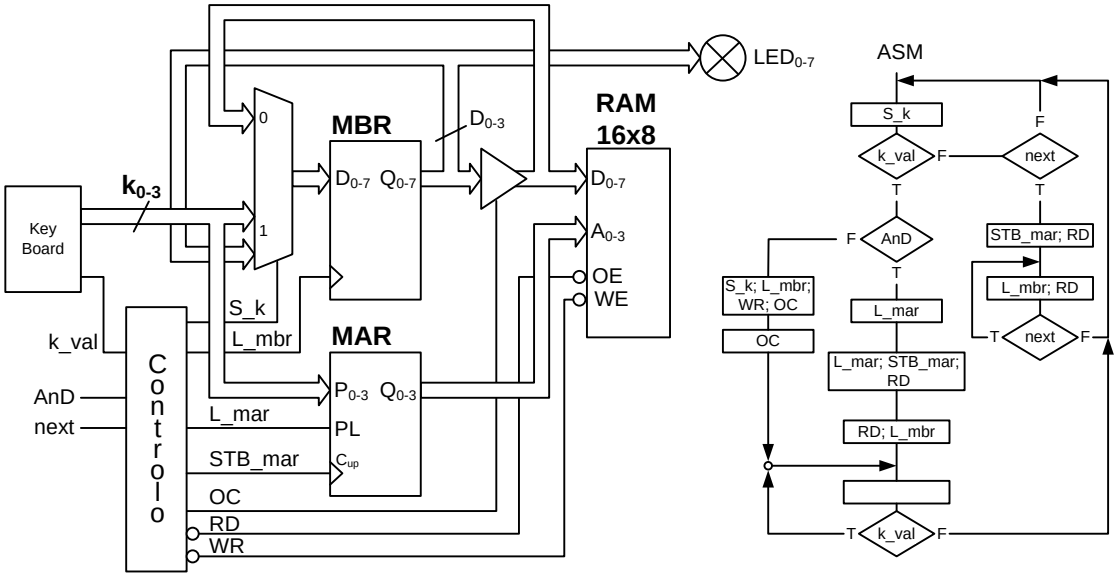


Figura 9-26 – Diagrama de blocos e ASM do módulo de controlo do DMA

Os dados a serem escritos são construídos por concatenação com deslocamento das várias teclas que vão sendo premidas. O código de uma tecla premida é inserido no *nibble* de menor peso do byte a ser escrito, ficando como *nibble* de maior peso o anterior *nibble* de menor peso, ou seja, a inserção da nova tecla faz-se com o deslocamento de 4 bits para a esquerda do byte acumulado no registo MBR.

9.6 Exercícios do capítulo 8

Exercício 1:

Pretende-se implementar o sistema de controlo para abertura de uma fechadura electromecânica F accionada por três botões de pressão **B1**, **B2** e **B3**. A abertura da fechadura depende de uma sequência introduzida através dos botões (**B1** a **B3**). A sequência é composta por quatro dígitos D0 a D3 estabelecidos através de um DIP *Switch* existentes no interior do sistema conforme a Figura 9-27. Cada um dos dígitos (D0 a D3) que estabelece a sequência é codificado por dois bits.

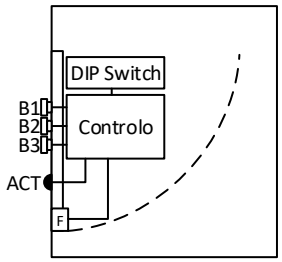


Figura 9-27

O sistema é constituído pelos seguintes elementos:

- Três botões B1 a B3, para que o utilizador insira a sequência de tentativa de abertura da fechadura.
- Um DIP *Switch* de 8 interruptores, oculto, para introduzir os quatro dígitos que estabelecem a sequência de abertura.
- Uma saída de sinal luminoso L para indicar que o sistema está activo.
- Uma saída F para activar a fechadura electromecânica.

Operação:

A abertura da fechadura consiste numa sequência de actuações dos botões B1 a B3. O código é constituído por quatro actuações. Por se tratar de uma sequência de quatro dígitos, a sequência que constitui o código de abertura contém obrigatoriamente mais que uma actuação num dos botões B1 a B3.

Após ter actuado o primeiro botão da sequência, a lâmpada L acende-se e dispõe de 6 (+-1) segundos para finalizar a sequência. A título de exemplo, podemos ver na Figura 9-28 o DIP *Switch* para a sequência de abertura 3,2,1,2 sendo o botão B2 o primeiro da sequência.

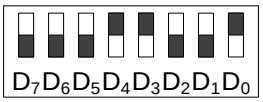
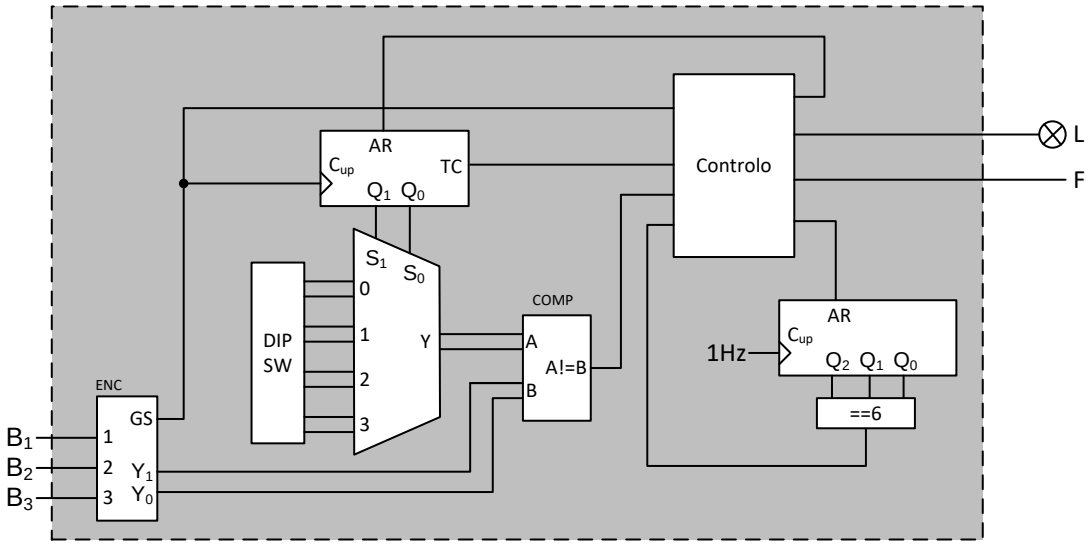
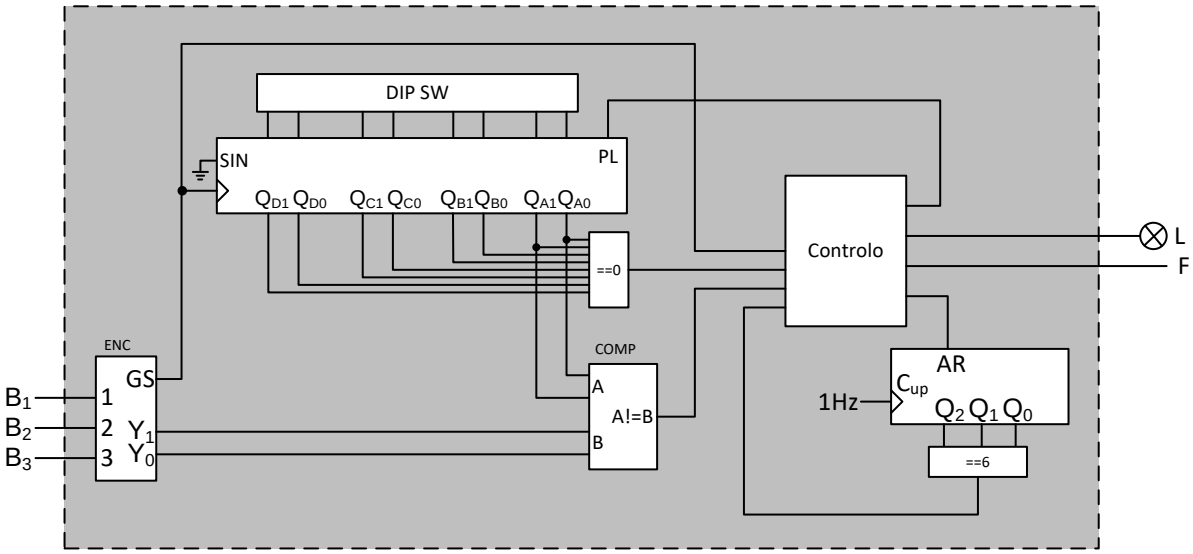


Figura 9-28

Possíveis arquitecturas:





Exercício 2:

Pretende-se implementar o sistema de controlo para abertura de uma fechadura electromecânica F accionada por dois botões de pressão B1 e B2. A abertura da fechadura depende de um código composto por dois dígitos decimais D1 e D2 e que são estabelecidos através de dois DIP *Switches* (SW1 e SW2) existentes no interior do sistema conforme a Figura 9-29. Cada um dos códigos D1 e D2 é composto por dois bits.

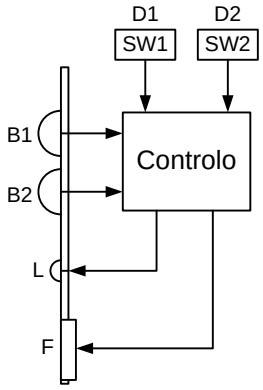
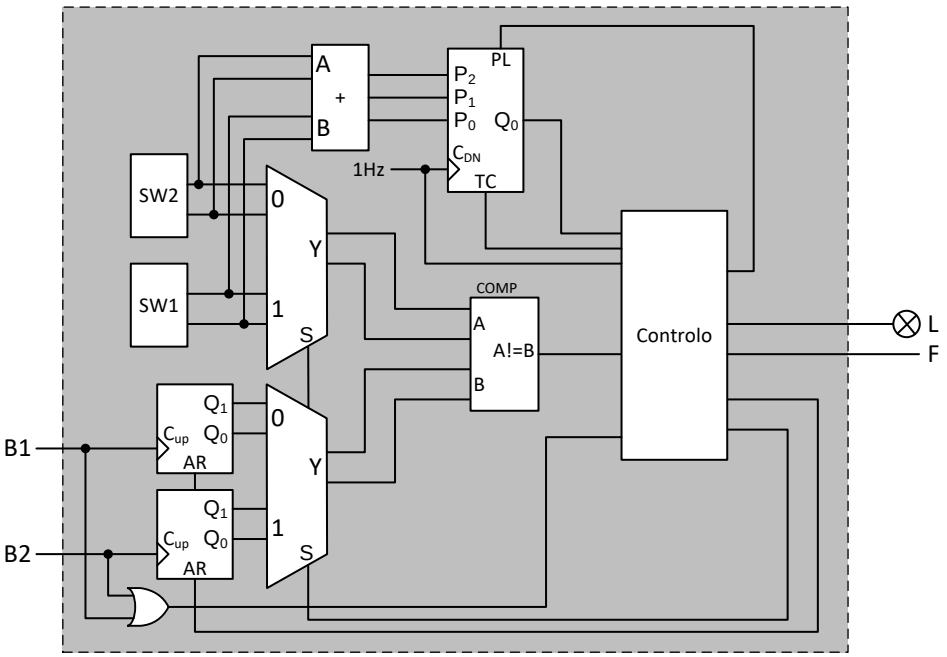


Figura 9-29

A tentativa de abertura da fechadura inicia-se premindo o botão B1 ou B2; após premir um dos botões B1 ou B2 acende-se uma lâmpada L durante um tempo em segundos, igual à soma dos dois códigos D1 e D2; durante este tempo actua-se B1 e B2 tantas vezes quantas as estabelecidas por D1 e D2 respectivamente; findo este tempo se os códigos introduzidos através de B1 e B2 coincidirem com os valores D1 e D2 estabelecidos pelos dois DIP *Switches*, a fechadura F é accionada aproximadamente durante 1 segundo.

Possível arquitectura:



Exercício 3:

Pretende-se implementar o sistema de controlo de um elevador de quatro pisos. O mecanismo do elevador é composto por um motor com dois sentidos de rotação, dois sensores nos extremos da coluna do elevador para indicar fim de curso, e um sensor que indica presença num piso. Em cada piso existe um botão de chamada (C0 a C3). A chamada realizada pelo piso zero é a mais prioritária e a do piso 3 a menos prioritária.

Possível arquitectura: