ISCTE IUL
Instituto Universitário de Lisboa
Departamento de Ciências e Tecnologias da Informação

Arquitectura de Computadores (I)

Textos de apoio

– Circuitos Sequenciais –

Draft – Versão 0.02b – Novembro de 2009 – Tomás Brandão.

Índice

Circuitos sequenciais

1. Latches

Nesta secção são abordados os circuitos sequenciais mais elementares, a partir dos quais podem ser construídos circuitos mais complexos e com mais funcionalidades. Estes circuitos elementares designam-se por *latches* [1].

A principal função de um *latch* resume-se a possibilitar o armazenamento de um determinado valor lógico – um '1' ou um '0' – desde que se forneça energia eléctrica ao circuito. Pode por isso ser visto como um elemento de memória que permite armazenar um bit de informação.

Para controlar o modo como o armazenamento de informação é feita, os *latches* possuem algumas linhas de entrada. Existem diferentes tipos de *latches*, que diferem na sua topologia e na forma como a informação é guardada.

1.1. Latch SR

O *latch* cujo funcionamento é o mais simples de compreender é provavelmente o ***latch* SR** [2], que se encontra representado na Figura 1.

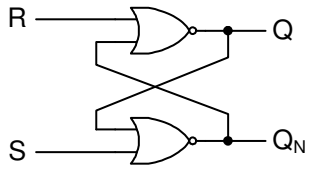


Figura 1 – Latch SR.

Para perceber como funciona este *latch*, considere que é aplicada às entradas S e R a sequência de valores lógicos que se segue:

Inicialmente, admita que R=0 e S=1 – nesta situação o valor da saída Q_N vai ser inevitavelmente '0', independente do valor lógico na outra entrada da porta NOR de baixo. Como R=0 e Q_N=0 então a saída Q assumirá o valor lógico 1. Suponha que depois se coloca a '0' o valor lógico de S – nada se irá alterar no circuito: Q=1 e Q_N=0.

Subindo depois o valor de R para '1', a saída Q irá a '0', e consequentemente Q_N irá subir para o nível lógico '1'. Baixando novamente R para '0', nada se altera no circuito: Q=0 e Q_N=1.

Repare agora nos resultados obtidos nos dois parágrafos anteriores – em ambos os casos, as entradas R e S terminam a '0', mas as saídas terminam com valores diferentes – no primeiro caso Q termina a '1' e no segundo caso Q termina a '0'. Pode-se concluir imediatamente que este circuito é sequencial, pois para a mesma combinação de entradas, as saídas apresentam valores diferentes... Pode-se também observar que este circuito permite a memorização de um bit, desde que se actue de forma adequada nas entradas S e R.

[1] Em português, uma possível tradução seria *báscula* ou *trinco*.
[2] Também é conhecido por *latch* RS.

A sequência de operações que foi seguida encontra-se sintetizada na Tabela 1. Note que esta não é uma tabela de verdade, mas sim uma tabela onde se encontra a sequência dos valores lógicos de R e S que foram sendo aplicados à entrada do *latch*.

R	S	Q	Q_N	Observações
0	1	1	0	*Set* (Guardar um '1')
0	0	1	0	Mantém o estado
1	0	0	1	*Reset* (Guardar um '0')
0	0	0	1	Mantém o estado

Tabela 1 – Sequência de operações no latch SR.

Então e para o caso em que S=R=1? Nesse caso, ambas as saídas (Q e Q_N) são forçadas ao nível lógico '0'. Porém, após ser aplicado '1' em S e em R, pode ocorrer uma possibilidade curiosa se tanto R como S forem simultaneamente a '0': nessa situação é impossível de prever teoricamente quais os valores resultantes para Q e Q_N (pelo menos com o que se sabe à partida). Tudo depende dos tempos de propagação das portas lógicas: se a porta de cima reagir mais depressa, então o circuito tenderá a estabilizar com Q=1 e Q_N=0; se, pelo contrário, se for a porta de baixo a reagir mais depressa, então o circuito estabilizará em Q=0 e Q_N=1. Este é o chamado *estado indefinido* do *latch*.

Note também que, na prática, dificilmente S e R desceriam em simultâneo para o valor '0' – na realidade existe sempre uma diferença temporal entre as duas descidas, que por mais insignificante que seja, influencia os valores resultantes para Q e Q_N.

Todas estas situações podem ser observadas na Figura 2, onde se encontram representadas as formas de onda resultantes para as saídas Q e Q_N em função dos sinais aplicados às entradas.

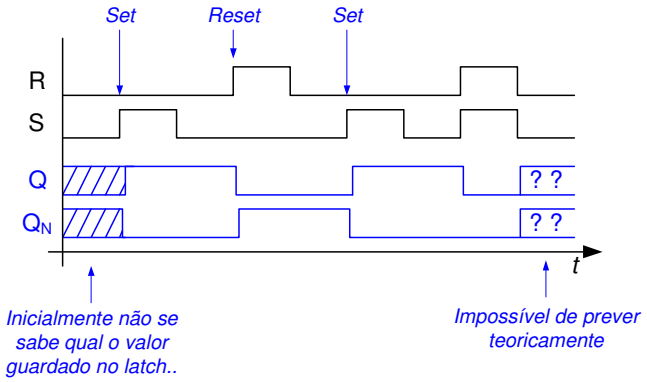


Figura 2 – Formas de onda típicas para o latch RS.

Para resumir, encontram-se sintetizadas na Tabela 2 as operações que podem ser efectuadas com um *latch* SR.

R	S	Operação
0	0	Manter o estado anterior
0	1	*Set* (Q=1)
1	0	*Reset* (Q=0)
1	1	Não se utiliza

Tabela 2 – Latch SR – Síntese das operações.

1.2. Latch $\overline{S}\overline{R}$

Outro *latch* que é facilmente construído com duas portas lógicas é o ***latch*** $\overline{\mathbf{S}}\,\overline{\mathbf{R}}$ (lê-se *não-S não-R*), representado na Figura 3. O seu funcionamento é semelhante ao do *latch* SR descrito na secção anterior, com a diferença das suas entradas responderem ao nível lógico '0' (como se R e S estivessem complementados).

Deste modo, para efectuar uma operação de *Set* faz-se S=0 e R=1; *Reset* faz-se com R=0 e S=1 e para manter o estado colocam-se ambas as entradas a '1'. Em suma, faz-se o contrário do que foi visto para o *latch* SR. A combinação de entradas que não é utilizada (e que pode conduzir à indefinição) é S=R=0.

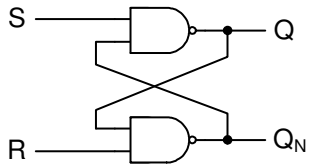


Figura 3 – Latch $\overline{S}\overline{R}$.

As operações que se podem realizar com este *latch* (em função de S e R) são portanto as seguintes:

R	S	Operação
0	0	Não se utiliza
0	1	*Reset* (Q=0)
1	0	*Set* (Q=1)
1	1	Manter o estado anterior

Tabela 3 – Latch $\overline{S}\overline{R}$ – Síntese das operações.

1.3. Latches com controlo

Pode-se acrescentar lógica combinatória aos *latches* descritos anteriormente, de modo a que seja introduzida uma variável de controlo C que funciona da seguinte maneira: quando desactivada, "obriga" o *latch* a manter o estado; quando activada permite alterações ao estado do *latch*. Desta forma consegue-se um maior controlo no modo como se introduz a informação a armazenar no *latch*. Um possível circuito para um ***latch SR com controlo*** é o seguinte:

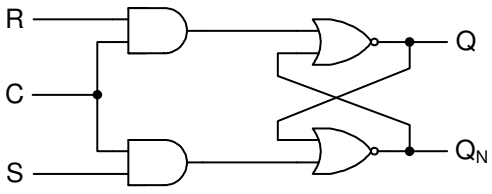


Figura 4 – Latch SR com controlo.

As duas portas AND acrescentadas às entradas do *latch* SR fazem com que os valores lógicos que passam para as linhas de entrada nas portas NOR sejam R e S, quando C está a '1', ou

sejam ‘0’ quando C está a ‘0’. Pode-se portanto concluir que quando C está a ‘0’, o *latch* mantém o estado, independentemente dos valores lógicos nas linhas R e S.

Na Figura 5 encontra-se esquematizado outro circuito correspondente a um *latch* SR com controlo. A vantagem deste circuito relativamente ao anterior é o facto de apenas serem utilizadas portas lógicas NAND. Este circuito é construído com base no *latch* $\overline{S}\overline{R}$, acrescentando-se mais um par de portas NAND para controlo.

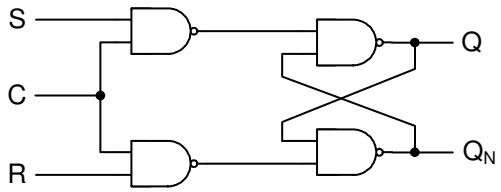


Figura 5 – Latch SR com controlo (construído com NANDs).

As operações (válidas para ambos os circuitos) encontram-se descritas na Tabela 4.

C	R	S	Operação
0	x	x	Manter o estado anterior
1	0	0	Manter o estado anterior
1	0	1	*Set* (Q=1)
1	1	0	*Reset* (Q=0)
1	1	1	Não se utiliza

Tabela 4 - Latch RS com controlo – síntese das operações.

O *latch* SR com controlo é, de certo modo, redundante, pois é possível manter o estado fazendo C=0, ou então C=1 com S=R=0. Para além disso existe a combinação que não se usa quando C=R=S=1...

É neste contexto que surge o ***latch D***. Este é construído com base num *latch* SR com controlo, fazendo-se com que S e R sejam sempre o complemento (negação) um do outro. Para tal, basta acrescentar um inversor entre S e R, como se pode observar na Figura 6, originando uma única entrada que se designa por D.

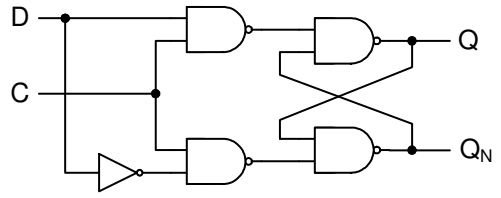


Figura 6 – Latch D.

Como S e R ficam sempre com valores lógicos complementares, as situações em que ambos estão a ‘1’ ou ambos estão a ‘0’ deixam de ser possíveis. Sendo assim, para manter o estado faz-se apenas C=0; para guardar um ‘1’ activa-se C e faz-se D=1; e para guardar um ‘0’ activa-se C e faz-se D=0.

O *latch* D com controlo é portanto capaz de efectuar todas as operações com interesse no armazenamento de um bit informação. Tem ainda a vantagem de ser completamente determinístico, isto é, não ocorrem estados que não se conseguem prever teoricamente.

C	D	Operação
0	X	Manter o estado
1	0	*Reset* (Q=0)
1	1	*Set* (Q=1)

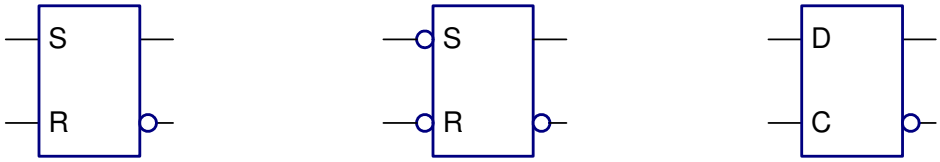
Tabela 5 – Latch D – síntese das operações.

Devido à facilidade com que é feito o controlo, os *latches* D são muito utilizados no fabrico de memórias (obviamente são utilizados muitos *latches*, pois cada um só consegue guardar um bit...).

Repare também que não faz sentido a existência de um *latch* D sem entrada de controlo, pois nesse caso não seria possível manter o estado, i.e., não seria possível deixar guardado um bit de informação – não seria um circuito sequencial...

1.4. Símbolos

Os *latches* são representados por rectângulos, nos quais se assinalam todas as entradas e saídas do *latch*. Na Figura 7 encontram-se representados alguns desses símbolos.



(a) Latch SR (b) Latch $\overline{S}\overline{R}$ (c) Latch D

Figura 7 – Exemplos de símbolos utilizados para representar *latches*.

Relembre que a existência de círculos (as "bolinhas") nas entradas indica que estas são activas a '0'. É comum existir a saída correspondente a Q_N – esta saída é sempre o complemento de Q (a negação de Q), excepto nos casos especiais que foram referidos anteriormente (e que não se usam) para os quais Q e Q_N assumem o mesmo valor lógico.

2. Flip-flops

Na secção anterior foram introduzidos os *latches*. Ao serem efectuadas alterações aos valores lógicos das suas entradas, pode observar-se um efeito imediato ao estado do *latch*. Nos *latches* com controlo é possível escolher os momentos em que se pretende efectuar modificações ao estado do *latch*: quando a variável de controlo está activada podem-se efectuar modificações ao estado do *latch*; quando esta está desactivada, o estado mantém-se. Esta possibilidade, apesar de desejável, é muitas vezes insuficiente em sistema mais complexos, onde se pretende que os elementos sequenciais apenas possam modificar o seu estado em instantes temporais específicos.

É neste contexto que surge um elemento sequencial designado por *flip-flop*. Qualquer *flip-flop*, independentemente do seu tipo, possui a característica de só alterar o seu estado em instantes temporais específico, que correspondem a variações no nível lógico de um dado sinal de referência – o sinal de relógio.

2.1. Flip-flops *Master-Slave*

Na Figura 8 encontra-se esquematizado um ***flip-flop SR Master-Slave***. Este flip-flop é construído com base em dois *latches* SR com controlo. O *latch* ligado directamente às entradas é o ***master*** e o *latch* ligado directamente às saídas é o ***slave*** [3], daí a designação atribuída a este tipo de flip-flops.

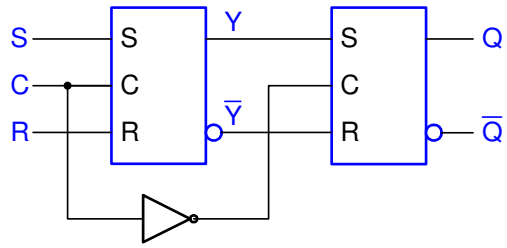


Figura 8 – Flip-flop SR Master-Slave.

Considere agora que é aplicado um sinal de relógio na entrada do *latch master*. Enquanto o sinal de relógio está a '1', o *latch master* está activado, podendo-se efectuar *set*, *reset* ou manter o estado anterior deste *latch*. Repare também que enquanto o sinal C está a '1', o sinal de controlo que chega ao *latch slave* é '0' – este encontra-se desactivado, mantendo por isso o valor lógico que se encontrava anteriormente armazenado. Quando o sinal C desce de '1' para '0', é desactivado o *master* e é activado o *slave*. O *slave* efectua então a operação que ficou definida nas saídas do *master*, enquanto o sinal C estava '1'.

O flip-flop SR *master-slave* pode no entanto conduzir a uma situação indesejável: quando ambas as entradas S e R estão a '1' e o sinal de relógio desce de '1' para '0', o *latch master* vai ficar no estado indefinido, sendo por isso impossível de prever o estado em que depois ficará o *slave*...

[3] É comum aparecer a designação *Master-slave* em circuitos digitais (mesmo em contextos diferentes dos *flip-flops*). Esta designação aparece normalmente quando existe uma dependência entre dois elementos, onde as acções de um elemento B dependem do que faz um outro elemento A – A é *master*, B é *slave*.

O seguinte diagrama temporal ilustra as situações descritas:

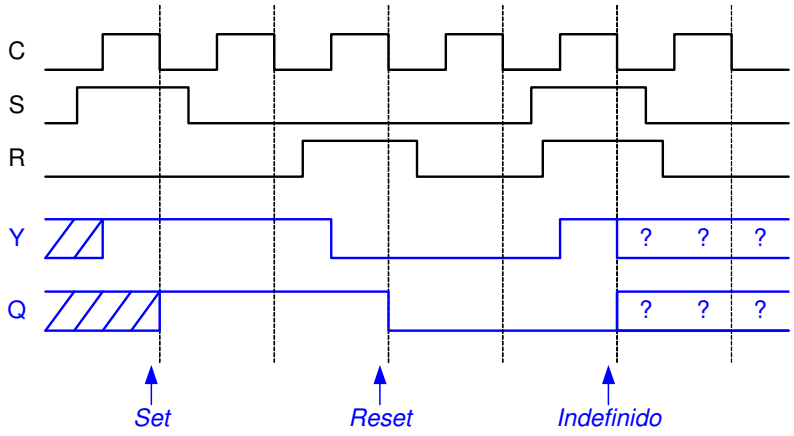


Figura 9 – Exemplo de formas de onda aplicadas a um FF SR Master-Slave.

Apresentam-se na Tabela 6 as operações que podem ser efectuadas com um flip-flop SR *master-slave*. As designações Q_t e Q_{t+1} designam o estado actual e o próximo estado do flip-flop, respectivamente. O desenho associado ao sinal C indica que o flip-flop só reage quando o relógio está a '1' e depois vem a '0'.

C	S	R	Operação
⌐¬	0	0	Manter o estado anterior ($Q_t = Q_{t+1}$)
⌐¬	0	1	*Reset* ($Q_{t+1} = 0$)
⌐¬	1	0	*Set* ($Q_{t+1} = 1$)
⌐¬	1	1	Indefinido ($Q_{t+1} = ?$)

Tabela 6 – Flip-flop SR Master-Slave – síntese das operações.

Mais uma vez salienta-se o facto deste flip-flop ter um comportamento difícil de prever quando S=R=1 na altura em que o relógio desce de '1' para '0'.

Um flip-flop que elimina esse problema é o ***flip-flop JK master-slave***, representado na Figura 10. Este é construído com base no flip-flop SR anterior, acrescentando-se um par de ligações entre as saídas do FF e as entradas.

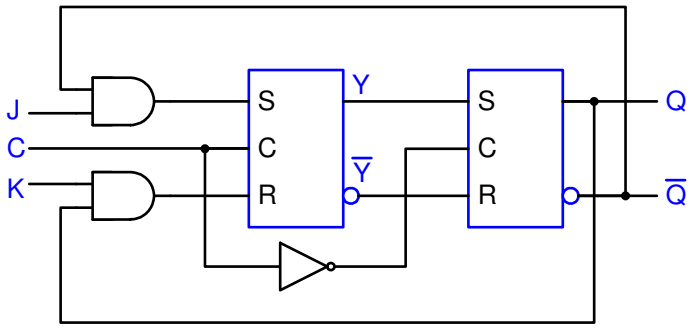


Figura 10 – Flip-flop JK Master-Slave.

Admitindo que as saídas Q e $\overline{Q}$ têm sempre valores lógicos complementares, quando J=K=1, e C se encontra a '1', acontece uma das seguintes situações:

- se Q=0 e $\overline{Q}$=1, então à entrada do *master* tem-se S=1 e R=0 – é feito um *Set* (Y=1 e $\overline{Y}$=0). Quando relógio passar para '0', o *slave* faz *Set* e portanto as saídas do FF ficarão com os valores Q=1 e $\overline{Q}$=0;
- se Q=1 e $\overline{Q}$=0, então à entrada do *master* tem-se S=0 e R=1 – é feito um *Reset* (Y=0 e $\overline{Y}$=1). Quando relógio passar para '0', as saídas do FF ficarão com os valores Q=0 e $\overline{Q}$=1.

Em ambos os casos, os valores das saídas são complementados face aos valores que estavam armazenados no estado anterior. Pode portanto concluir-se que, quando J=K=1 (e o relógio transita de '1' para '0'), este flip-flop complementa o estado anterior.

Para ilustrar as várias situações que podem acontecer num flip-flop JK *master-slave*, representam-se na Figura 9 formas de onda para Y e Q em função dos valores de J, K e do sinal de relógio C.

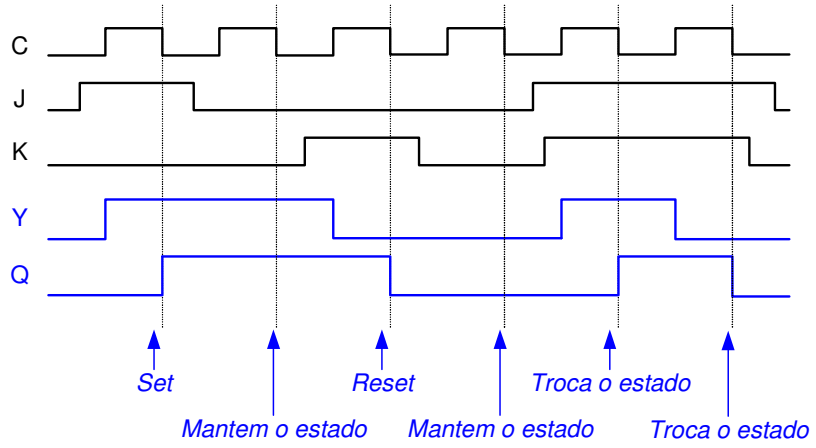


Figura 11 – Exemplo de formas de onda num flip-flop JK Master-Slave.

Para completar esta secção, na Tabela 7 resumem-se as operações que podem ser efectuadas com um flip-flop JK *master-slave*.

C	J	K	Operação
⎍	0	0	Manter o estado anterior ($Q_{t+1} = Q_t$)
⎍	0	1	*Reset* (Q_{t+1}=0)
⎍	1	0	*Set* (Q_{t+1}=1)
⎍	1	1	Complementar o estado anterior ($Q_{t+1} = \overline{Q}_t$)

Tabela 7 – Flip-flop JK Master-Slave – síntese das operações.

2.2. Flip-flops *Edge-Triggered*

Os flip-flops do tipo *master-slave* possuem a particularidade de, enquanto o relógio está activado, poderem ser efectuadas operações no *latch master* (*set*, *reset* e manter o estado). Pode-se então afirmar que durante o intervalo de tempo em que o relógio está activado, tudo o que é feito no *master* ou seja, nas entradas do flip-flop, pode influenciar o próximo estado.

Existem outros FFs em que tal não acontece – só têm interesse os níveis lógicos apresentados às entradas do flip-flop quando o relógio efectua uma transição do seu nível lógico (de '0' para '1'

ou de ‘1’ para ‘0’). Este tipo de FFs designam-se por *Edge-triggered* [4]. Nos flip-flops *edge-triggered*, o *master* é um *latch* do tipo D.

Dentro dos FFs *edge-triggered*, aquele cujo funcionamento é o mais simples é o flip-flop D, representado na Figura 12. Neste flip-flop, quando o sinal de relógio está a ‘0’, o latch D encontra-se activado, mas o SR não. Quando se dá uma transição de ‘0’ para ‘1’ em C, é desactivado o *latch* D e é activado o *latch* SR, que guarda o valor que estava no *latch* D nesse momento.

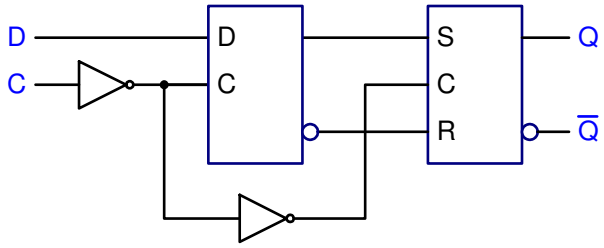


Figura 12 – Flip-flop D Edge-triggered.

Repare que interessa apenas o valor que se encontra na entrada D no momento em que se dá a transição de ‘0’ para ‘1’ no sinal de relógio. Como os valores são passados para o segundo *latch* na transição de ‘0’ para ‘1’, diz-se que este flip-flop é *positive edge-triggered* (reage no flanco ascendente). Se fosse retirada a porta NOT aplicada na entrada C, então teríamos um FF com funcionamento idêntico, mas com reacção no flanco negativo do sinal de relógio, de ‘1’ para ‘0’, ou seja, um FF D *negative edge-triggered*.

C	D	Operação
↓	0	*Reset* (Q=0)
↑	1	*Set* (Q=1)

Tabela 8 – Flip-flop D Edge-Triggered (positivo) – síntese das operações.

A Tabela 8 sintetiza as operações realizadas com um flip-flop D *edge-triggered*. As operações que podem ser feitas (*Set* ou *Reset*) ficam sincronizadas com o sinal de relógio.

Com base no FF D pode-se construir um flip-flop JK *edge-triggered*, que funciona de modo semelhante ao JK *master-slave*, mas para o qual interessam apenas os instantes de tempo em que o sinal de relógio transita de ‘1’ para ‘0’ (*negative*) ou de ‘0’ para ‘1’ (*positive*).

O esquema deste flip-flop é apresentado na Figura 13 e uma descrição do seu comportamento pode ser observado na Tabela 9.

[4] Em português, *com reacção no flanco*.

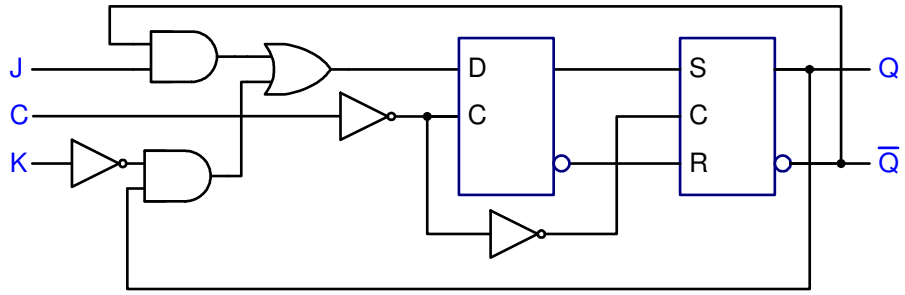


Figura 13 – Flip-flop JK Edge-triggered.

C	J	K	Operação
⌐↑	0	0	Manter o estado anterior ($Q_{t+1} = Q_t$)
⌐↑	0	1	*Reset* (Q_{t+1}=0)
⌐↑	1	0	*Set* (Q_{t+1}=1)
⌐↑	1	1	Complementar o estado anterior ($Q_{t+1} = \overline{Q}_t$)

Tabela 9 – Operações num flip-flop JK edge-triggered (positivo).

2.3. Símbolos

A simbologia associada aos *flip-flops* é semelhante à utilizada nos *latches*, diferenciando-se desta por introdução de símbolos que indicam uma resposta a variações no sinal de relógio. Alguns destes símbolos podem ser observados na Figura 14.

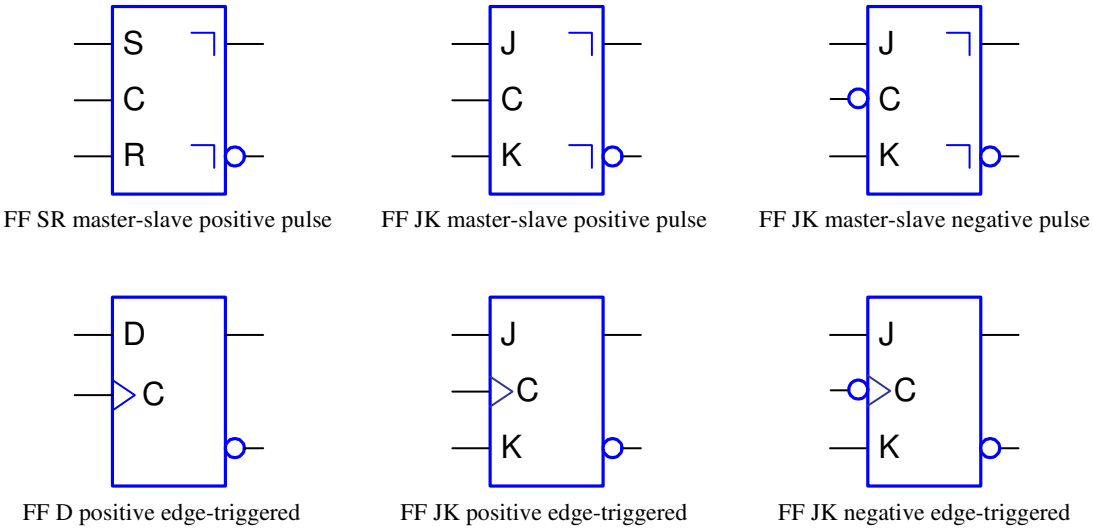


Figura 14 – Exemplos de símbolos utilizados em flip-flops.

Os flip-flops do tipo *edge-triggered* apresentam um triângulo na entrada do sinal de relógio e os flip-flops do tipo *master-slave* apresentam o símbolo '⌐' junto às suas saídas (representa a variação do sinal de relógio após ter estado activo). A designação de negativo ou positivo é indicada pela presença (ou não) do símbolo da negação junto à entrada do sinal de relógio.

2.4. Temporizações do flip-flops

À semelhança do que acontece com os circuitos combinatórios, os tempos de propagação associados às portas lógicas introduzem também algumas restrições ao funcionamento dos flip-

flops. Para além do tempo de propagação, nos flip-flops aparecem também novas temporizações que têm a ver com o tempo em que os sinais de entrada (S e R, J e K, ou D) devem permanecer estáveis quando se aplica um estímulo de relógio que faz reagir o flip-flop:

- Tempo de preparação (*setup time*) – é o intervalo de tempo de guarda, antes de um impulso do sinal de relógio, durante o qual as entradas do flip-flop devem permanecer estáveis.
- Tempo de manutenção (*hold time*) – intervalo de tempo de guarda, após um impulso do sinal de relógio, durante o qual é necessário manter estáveis as entradas.
- Tempo de propagação (*propagation delay time*) – tempo que decorre entre uma transição do sinal de relógio até que o FF mude o valor da saída.

Na Figura 15 representa-se um pequeno exemplo referente a estas temporizações aplicadas a um flip-flop D. Pretende-se colocar o estado do flip-flop a '1' – para que tal aconteça há que respeitar os tempos de preparação (t_S) e de manutenção (t_H). Para tal coloca-se a '1' a entrada D pelo menos t_S antes do momento em que se dá a transição do sinal de relógio e mantém-se a entrada a '1' pelo menos t_H após a transição. A saída Q mudará para o valor '1' t_{PD} depois do impulso de relógio. Para flip-flops JK ou SR seriam situações semelhantes – a única diferença seria o facto de serem duas entradas.

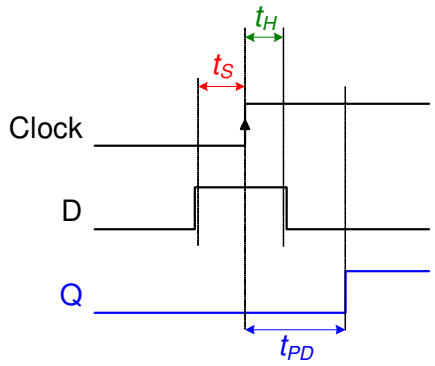


Figura 15 – Temporizações num flip-flop D.

Estas características temporais não ideais tem uma importância fundamental no cálculo da frequência máxima a que um circuito sequencial síncrono pode funcionar correctamente, como será visto mais adiante.

2.5. Entradas assíncronas

É habitual um FF possuir outras entradas (para além das já referidas) que "forçam" um determinado valor lógico no estado do FF, independentemente do sinal de relógio. Este tipo de entradas funcionam por isso de uma forma assíncrona e permitem inicializar o estado do flip-flop com um determinado valor lógico. Este tipo de inicialização é bastante útil em circuitos mais complexos.

Cada uma destas entradas assíncronas ou responde ao nível lógico '0' ou ao nível lógico '1', i.e., podem ser activadas ou a '1 ou a '0' – uma forma de distinguir se são activadas a '1' ou a '0' é a presença no esquema do símbolo da negação (a "bolinha"). Quando aparece o símbolo da negação, significa que essa entrada é activada com '0'; caso contrário, é activada a '1'.

Na Figura 16 representa-se o esquema de um flip-flop JK *edge-triggered* típico, contendo entradas assíncronas que permitem inicializar o estado do flip-flop a '1' (*Set*) ou a '0' (*Reset*). Neste caso ambas as entradas são activadas a '0'.

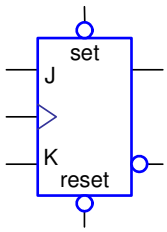


Figura 16 – Flip-flop com entradas assíncronas.

Uma possível descrição para este circuito seria a que é dada na Tabela 10 (presume-se que se *reset* e *set* estiverem ambos activados prevalece o *reset*).

Reset	Set	C	J	K	Operação
0	x	x	x	X	*Reset* assíncrono (Q=0)
1	0	x	x	X	*Set* assíncrono (Q=1)
1	1	↑	0	0	Manter o estado anterior ($Q_{t+1} = Q_t$)
1	1	↑	0	1	*Reset* (Q_{t+1}=0)
1	1	↑	1	0	*Set* (Q_{t+1}=1)
1	1	↑	1	1	Complementar o estado anterior ($Q_{t+1} = \overline{Q_t}$)

Tabela 10 – Descrição do funcionamento de um flip-flop JK *edge-triggered* com entradas assíncronas.

3. Análise e projecto de circuitos sequenciais

Como já foi referido anteriormente, um circuito sequencial distingue-se de um circuito combinatório pelo facto de, para a mesma combinação de valores lógicos nas entradas, as saídas do circuito poderem apresentar valores lógicos diferentes em instantes temporais diferentes.

De uma forma geral, um circuito sequencial pode ser visto como um circuito composto por dois blocos distintos (ver Figura 17):

- *Elementos de memória* – que guardam conjunto de bits que definem o estado do circuito (normalmente utilizam-se flip-flops).
- *Bloco de lógica combinatória* – determina quais serão os valores da saída – em função do estado e, possivelmente, das entradas – e determina também qual será o próximo estado do circuito.

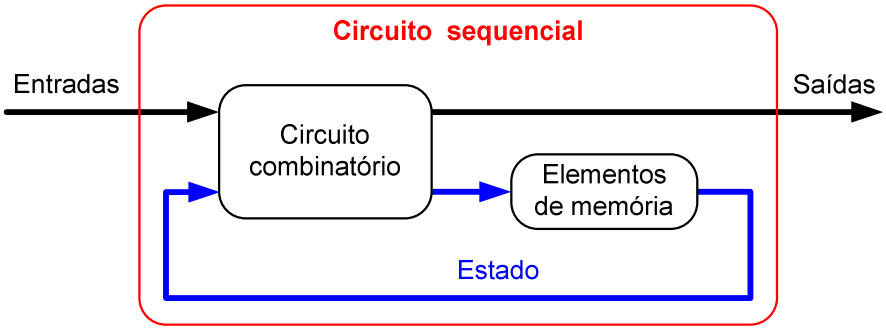


Figura 17 – Diagrama de blocos de um circuito sequencial.

Os elementos de memória que fazem parte do circuito sequencial são geralmente flip-flops. Quando o circuito sequencial muda de estado em instantes definidos por um dado sinal de referência (e.g. um sinal de relógio) diz-se que o circuito é ***síncrono***; no caso de não existir nenhum sinal de referência, o circuito diz-se ***assíncrono***.

Nesta secção serão abordados os circuitos sequenciais síncronos, dado que tanto o seu projecto como o seu funcionamento são substancialmente mais simples do que o respeitante a circuitos assíncronos.

3.1. Análise de circuitos sequenciais

Basicamente, a análise de um circuito sequencial consiste em determinar qual a sequência de estados seguida por um dado. Essa sequência pode variar em função dos valores que vão sendo apresentados às entradas em cada ciclo do sinal de relógio.

Considere o circuito representado na Figura 18, constituído por um flip-flop D *edge-triggered* e lógica combinatória adicional. O circuito possui uma entrada *X*, uma saída Z e é síncrono com o sinal de relógio *Clock*.

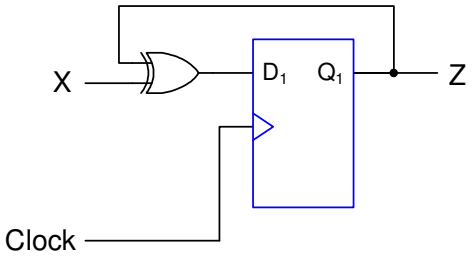


Figura 18 – Circuito sequencial em análise.

Uma das formas de descrever um circuito sequencial consiste em elaborar uma ***tabela de transição de estados***. Esta tabela indica qual será o próximo estado do circuito com base no estado actual e nos valores das entradas.

Para perceber melhor, considere que o circuito se encontra no estado '0', isto é, $Q_1 = 0$. Com a entrada X também a '0', o valor de D_1 será '0' ($D_1=X \oplus Q_1$). Quando ocorrer um impulso de relógio, o próximo estado do circuito será '0'. Sintetizando:

$Q_1 = 0$ e $X = 0 \Rightarrow Q_1' = 0$,

Em que Q_1' representa o próximo estado de Q_1.

Continuando no estado '0', mas supondo agora que X=1, então $D_1=1$. Quando ocorrer um impulso de relógio o valor de Q_1 passará para '1', ou seja, o próximo estado será o estado '1'. Tem-se então:

$Q_1 = 0$ e $X = 1 \Rightarrow Q_1' = 1$.

Seguindo um raciocínio semelhante para o caso em que o circuito se encontra no estado '1' é fácil de concluir que:

$Q_1 = 1$ e $X = 0 \Rightarrow Q_1' = 1$,

$Q_1 = 1$ e $X = 1 \Rightarrow Q_1' = 0$.

Utilizando todas as conclusões anteriores elabora-se a tabela de transição de estados que descreve o comportamento deste circuito:

Estado actual (Q_1)	Entrada (X)	Estado seguinte (Q_1')
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 11 – Tabela de transição de estados para o circuito em análise.

No caso de existirem mais flip-flops ter-se-ão mais variáveis de estado, o que originará mais combinações possíveis de estados e entrada (mais colunas e mais linhas na tabela). O mesmo se passaria no caso de existirem mais entradas.

Outra forma de descrever um circuito sequencial síncrono é feita através de um ***diagrama de estados*.** Um diagrama de estados consiste num esquema onde aparecem círculos, que representam os estados, e setas, que representam as transições entre estados. O diagrama de estados pode ser obtido com base numa tabela de transição de estados e vice-versa. O diagrama de estados que corresponde ao circuito anterior é representado na Figura 19.

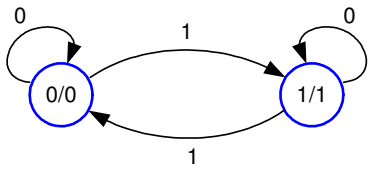


Figura 19 – Diagrama de estados do circuito em análise.

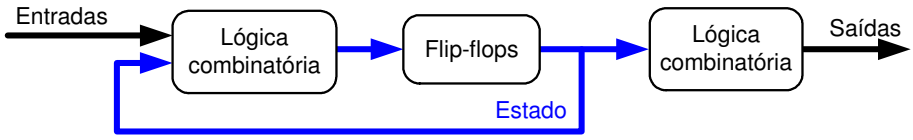
Os valores lógicos que aparecem nas setas são as várias combinações possíveis de valores lógicos das entradas (neste caso são apenas os valores possíveis de X). Quanto aos dígitos que aparecem dentro dos círculos são o estado e o valor de saída a este associado (estado/saída). Por exemplo, a designação '0/0' significa que no estado '0' o valor da saída é '0'. Como se verá na próxima secção, nem sempre a saída do circuito está associada apenas ao estado do circuito.

3.2. Modelos de *Mealy* e de *Moore*

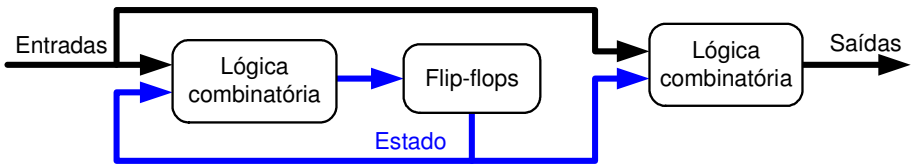
Quando se elabora o projecto de um circuito sequencial, é habitual começar-se por decidir se um circuito sequencial irá seguir o modelo de *Mealy* ou o modelo de *Moore*:

- Num circuito que segue o modelo de *Mealy*, o valor lógico das saídas, em qualquer instante temporal, depende do valor do estado e das entradas do circuito. As saídas são portanto funções lógicas das entradas e do estado.
- Nos circuitos que seguem o modelo de *Moore*, os valores lógicos das saídas dependem apenas do estado do circuito. São portanto funções do estado. Os valores das entradas influenciam o qual será o próximo estado do circuito, mas não influenciam de forma directa as saídas.

A Figura 20 pretende ilustrar a diferença entre estes dois modelos.



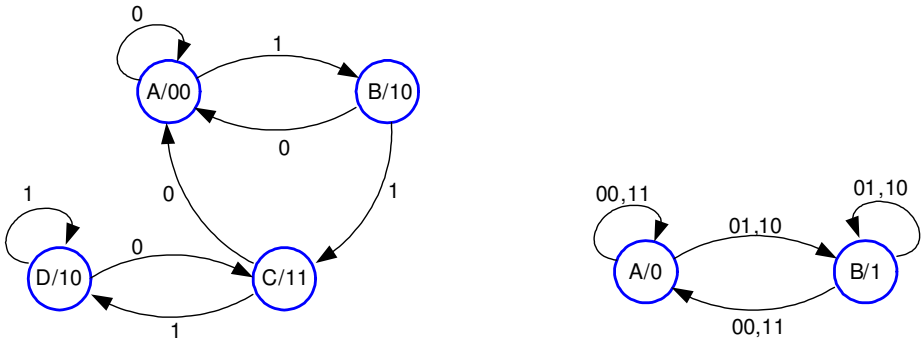
(a) Modelo de Moore.



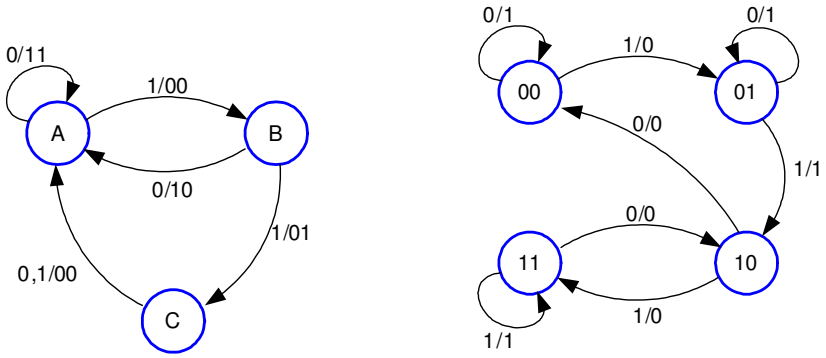
(b) Modelo de Mealy.

Figura 20 – Diferenças entre os modelos de Moore e de Mealy.

Nos diagramas de estados também se distingue qual o modelo seguido pelo circuito. Ao passo que num circuito que segue o modelo de *Moore*, os valores das saídas aparecem junto do estado (dentro dos círculos, à semelhança do que acontecia com o circuito da secção anterior), num circuito que segue o modelo de *Mealy* isso já não pode acontecer... para representar as saídas em função do estado e das entradas utilizam-se as setas – deste modo associam-se as saídas às entradas (de uma forma directa, uma vez que as entradas já estavam sobre a seta) e aos estados (de uma forma indirecta – as setas saem dos estados). Na Figura 21 apresentam-se alguns diagramas de circuitos que seguem ambos os modelos.



(a) Diagramas que seguem o modelo de Moore.



(b) Diagramas que seguem o modelo de Mealy.

Figura 21 – Exemplos de diagramas de transição de estados.

A escolha do modelo a utilizar depende da especificidade do problema – alguns problemas podem ser resolvidos utilizando o modelo de *Moore*, noutros será mais adequado o modelo de *Mealy*, que é o mais genérico.

Com base nas características dos dois modelos, também se podem tirar algumas conclusões adicionais. No modelo de *Moore*, como as saídas dependem apenas do estado do circuito, os seus valores só podem mudar quando ocorrem transições de estados, ou seja, quando ocorrem impulsos de relógio. Pode-se por isso concluir que as saídas estão também sincronizadas com um sinal de referência.

No modelo de *Mealy* essa conclusão não é válida. Como os valores das saídas dependem também dos valores das entradas, podem ocorrer variações nas saídas que não estão

sincronizadas com o sinal de relógio – se os valores das entradas variarem, os valores de saída poderão variar também, sem que o circuito tenha mudado de estado.

3.3. Projecto de circuitos sequenciais

Para projectar um circuito sequencial são geralmente seguidos os seguintes passos:

- ***Diagrama de transição de estados*** – a partir de um problema, elabora-se o diagrama de estados, efectuando-se logo a escolha do modelo (*Mealy*/*Moore*) a utilizar.

- ***Codificação de estados*** – atribuir uma combinação binária a cada um dos estados do circuito.

- ***Tabela de transição de estados*** – elaborar a tabela de transição de estados correspondente ao obtido nos pontos anteriores.

- ***Escolher os flip-flops a utilizar*** – optar pela utilização de flip-flops do tipo D ou de flip-flops do tipo JK. A utilização de FFs do tipo D conduz a uma realização mais rápida do projecto, mas o circuito resultante é, em geral, mais complicado (i.e., com maior número de portas lógicas) do que o obtido utilizando FFs JK.

- ***Obter as equações das entradas nos flip-flops*** – obter as expressões lógicas para as variáveis de entrada de cada FF (Ds ou JKs). As expressões são obtidas com base na tabela de transições de estados, usando-se os métodos já conhecidos para obter expressões simplificadas (mapas de Karnaugh, por exemplo).

- ***Obter as equações de saída*** – obter as expressões lógicas para as saídas do circuito. No caso do modelo de *Moore*, estas são uma função do estado; no caso do modelo de *Mealy* são uma função do estado e das entradas.

- ***Desenhar o circuito resultante***.

A título de exemplo, nas próximas páginas apresenta-se o desenvolvimento de um projecto cujo objectivo é chegar ao esquema de um circuito que corresponda ao diagrama de transição de estados representado na Figura 22.

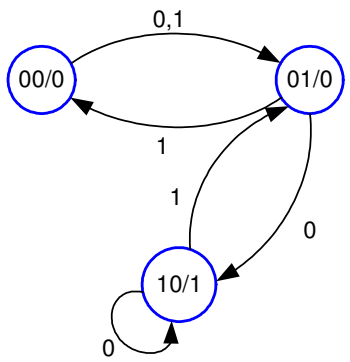


Figura 22 – Diagrama de transição de estados do circuito a projectar.

Por observação do diagrama anterior, pode-se concluir que:

- este circuito sequencial possui uma entrada (que será designada por X) e uma saída (que será designada por Z);
- o circuito segue o modelo de *Moore*. Quer isto dizer que o valor lógico da saída Z só depende do valor das variáveis de estado;
- transita por 3 estados distintos. Como para representar três combinações diferentes são necessários 2 bits, então existem 2 variáveis de estado e, consequentemente, 2 flip-flops para armazenar os seus valores.

Neste caso, já foram atribuídas combinações binárias a cada estado no diagrama – tem-se os estados '00', '01' e '10'. O próximo passo é obter a tabela de transições de estados. Com base no diagrama, pode-se observar que quando o circuito está no estado '00', independentemente da variável de entrada X estar a '0' ou a '1', o estado que seguinte é sempre o estado '01'.

Quando o circuito está no estado '01', o estado seguinte depende do valor de X: se X estiver a '0', então o próximo estado será '10', se X estiver a '1' então o próximo estado será '00'.

Utilizando o mesmo raciocínio para o caso em que o circuito está no estado '10', chega-se então à seguinte tabela de transições de estados:

Estado actual			Estado seguinte	
Q_1	Q_0	X	Q_1'	Q_0'
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	x	x
1	1	1	x	x

Tabela 12 – Tabela de transição de estados do circuito a projectar.

Repare que, como o estado '11' não existe no diagrama, considerou-se que para esse caso é indiferente o estado seguinte do circuito.

Depois de obtida a tabela de transição de estados, pode-se optar por implementar o circuito com flip-flops D, ou então com flip-flops JK. A escolha dos flip-flops a usar é livre, mas obviamente tem implicações no circuito resultante. Neste exemplo serão obtidos ambos os circuitos – o projectado com FFs D e o projectado com FFs JK.

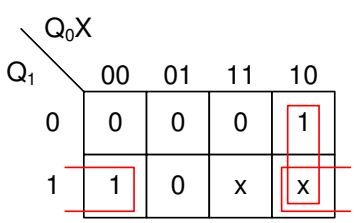
3.3.1. Projecto com flip-flops D

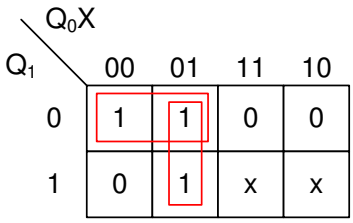
Num flip-flop D, o valor lógico que se quer armazenar no FF é o valor que se coloca à entrada do flip-flop. Por outras palavras, quando se quer guardar um '1', coloca-se a entrada D a '1'; quando se quer guardar um '0' coloca-se a entrada D a '0'. Sendo assim, pode-se afirmar que o

valor do próximo estado num flip-flop D é o valor lógico que se coloca na sua entrada antes de chegar o impulso de relógio.

Extrapolando esta abordagem para o circuito sequencial que se pretende projectar, pode-se concluir que o próximo estado do circuito é definido através dos valores lógicos que se apresentam à entrada de cada um dos flip-flop D.

Em equações, o raciocínio do parágrafo anterior traduz-se por $D_1 = Q_1$' e $D_0 = Q_0$'. Podem-se então obter as equações de entrada nos FFs D, a partir da tabela de transição de estados.





$$D_1 = Q_1' = Q_0\overline{X} + Q_1\overline{X} \qquad D_0 = Q_0' = \overline{Q}_1\overline{Q}_0 + \overline{Q}_0 X$$

Quanto ao valor da saída, como esta é uma função do estado, pode-se escrever uma tabela de verdade que relaciona o estado actual com a saída. Com base no diagrama de transição de estados da Figura 22, constrói-se a seguinte tabela:

Estado actual		Saída
Q_1	Q_0	Z
0	0	0
0	1	0
1	0	1
1	1	x

Tabela 13 –Relação entre a saída e o estado actual do circuito a projectar.

Considerando a indiferença a '1', pode-se escrever que $Z = Q_1$.

Finalmente, e com base nas várias expressões que foram sendo obtidas, chega-se ao seguinte circuito:

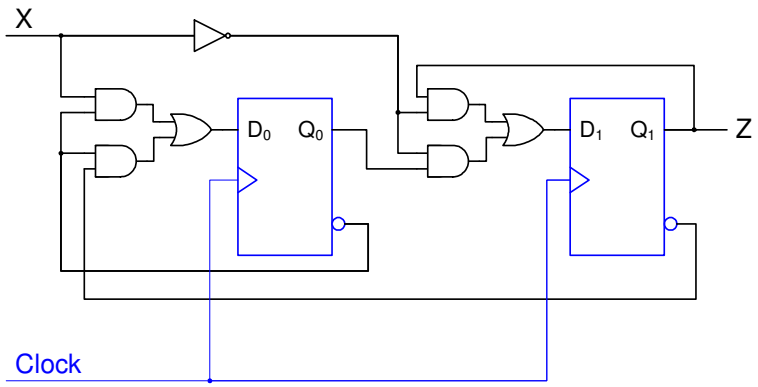


Figura 23 – Circuito obtido utilizando flip-flops D.

3.3.2. Projecto com flip-flops JK

Outra hipótese seria projectar o circuito utilizando flip-flops JK. Para além de se poderem efectuar as operações de *set* e *reset*, com um flip-flop JK é também possível manter ou complementar o estado anterior.

Suponha, por exemplo, que um flip-flop JK se encontra no estado '0' e se pretende que o próximo estado também seja '0'. Com um FF JK é possível conseguir essa situação de duas maneiras diferentes: ou se coloca J=0 e K=1, efectuando-se um *reset*, ou então coloca-se J=K=0, mantendo-se o estado anterior (que já era '0').

Para ilustrar todas as transições de estados possíveis num flip-flop JK, com base no funcionamento do FF obtém-se uma tabela que se designa por *tabela de excitação*. Esta tabela indica os valores que são necessários aplicar em J e em K de modo a que o flip-flop faça a transição de estados desejada.

Estado actual	Estado seguinte	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Tabela 14 – Tabela de excitação de um flip-flop JK.

Esta tabela diz-nos, por exemplo, que para o flip-flop transitar do estado '0' para o estado '1' (2ª linha da tabela), basta fazer J=1. Com J=1 e K=0 seria feito um *set* e portanto o próximo estado é '1'; com J=1 e K=1, troca-se o estado anterior, passando de '0' para '1'.

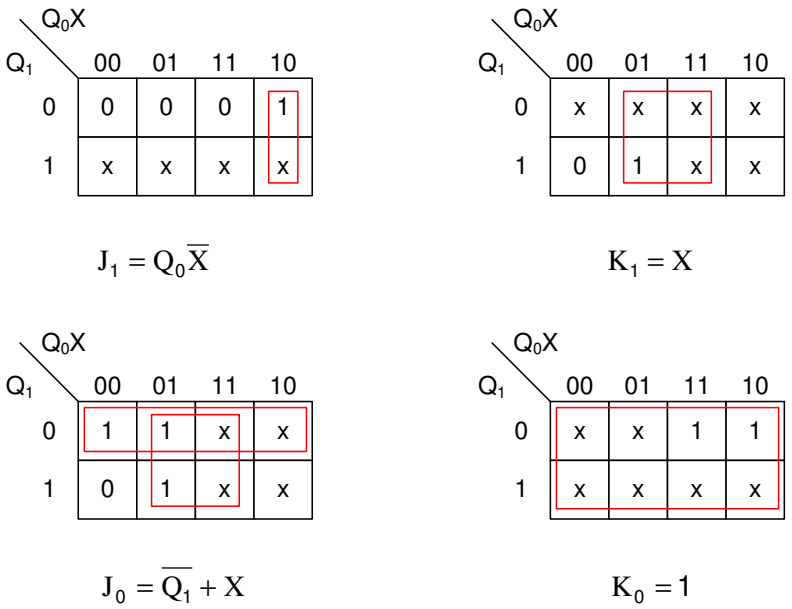
Considere de novo o projecto do circuito cujo diagrama e tabela de transição de estados se representaram na Figura 22 e na Tabela 12. Escolhendo a opção de o implementar com base em flip-flops JK, é necessário acrescentar novas colunas à tabela de transição de estados, de modo a aparecerem os valores de J e K que devem ser colocados nas entradas dos FFs. Os valores de J e K que se colocam em cada linha da tabela são os que provocam a transição de estados correspondente.

Com base na tabela de excitação do FF JK, acrescentam-se então à tabela de transição de estados os valores de J e K que fazem com o circuito siga a transição de estados desejada, resultando deste modo a Tabela 15.

Q_1	Q_0	X	Q_1'	Q_0'	J_1 K_1	J_0 K_0
0	0	0	0	1	0 x	1 x
0	0	1	0	1	0 x	1 x
0	1	0	1	0	1 x	x 1
0	1	1	0	0	0 x	x 1
1	0	0	1	0	x 0	0 x
1	0	1	0	1	x 1	1 x
1	1	0	x	x	x x	x x
1	1	1	x	x	x x	x x

Tabela 15 – Tabela de transição de estados (com os valores de J e K pretendidos).

Depois basta tirar as expressões correspondentes às entradas J e K de cada flip-flop. Para tal podem-se utilizar mapas de Karnaugh.



Com base nas expressões resultantes, obtém-se o circuito representado na Figura 24.

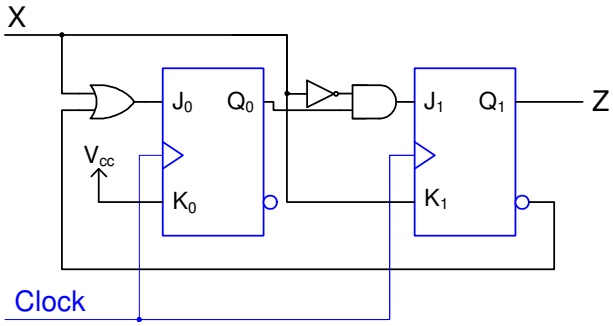


Figura 24 – Circuito obtido utilizando flip-flops JK.

A utilização de flip-flops JK, embora com um projecto mais trabalhoso, geralmente conduz a circuitos bem mais simples. No exemplo que foi apresentado, a solução obtida com os FFs D conduziu a um circuito com 7 portas lógicas (para além dos FFs) ao passo que o circuito obtido com FFs JK contém apenas 3 portas lógicas para além dos FFs.

3.4. Exemplos de projecto

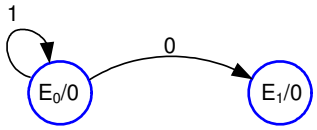
Nesta secção apresenta-se um problema que é resolvido através de um circuito sequencial. Todos os passos para projecto estão presentes neste exemplo, incluindo a construção de um diagrama de estados.

3.4.1. Detector de sequências de '0's

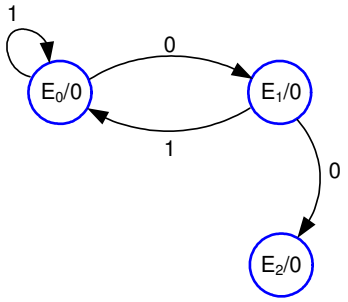
Pretende-se projectar um circuito sequencial, com uma entrada X, pela qual entram bits ao mesmo ritmo que o sinal de relógio. A saída Y do circuito deverá ir a '1' sempre que o valor '0' é apresentado à entrada durante três ou mais ciclos do sinal de relógio. Noutras circunstâncias a saída deverá ser sempre '0'.

Em primeiro lugar começa-se por desenhar um digrama de estados. Neste problema interessa que no caso da entrada estar três vezes (ou mais) consecutivas a '0', a saída dê o valor lógico '1'. Começa-se então por desenhar um estado inicial – E_0 –, a partir do qual se desenha o resto do diagrama. Quando o circuito está neste estado, o valor lógico da saída deverá ser '0' uma vez que ainda não chegou nenhum '0' à entrada.

Quando o circuito está neste estado inicial, pode acontecer uma de duas coisas: chegar um '1' à entrada, ou chegar um '0'. Se chegar um '1', então o circuito pode (e deve) permanecer no estado inicial, pois não há nenhum '0' a contabilizar. Mas, caso chegue um '0', então está-se na situação em que apareceu um 1º zero na entrada, o que deverá fazer com que o circuito passe para um novo estado em que já foi contabilizada a ocorrência de um '0' na entrada. Desenha-se por isso um novo estado – E_1 –, que corresponde a essa situação:



Agora suponha que, estando o circuito no estado E_1, chegar um novo '0'. Nesse caso o circuito deverá transitar para um novo estado – E_2 – no qual já foi contabilizado o aparecimento de 2 zeros. Caso contrário chegasse um '1', volta-se ao estado inicial – apareceu um '1' que "estragou" a sequência de 3 zeros e por isso o circuito volta ao estado E_0.



Seguindo esta linha de raciocínio chega-se ao diagrama de estados da Figura 25.

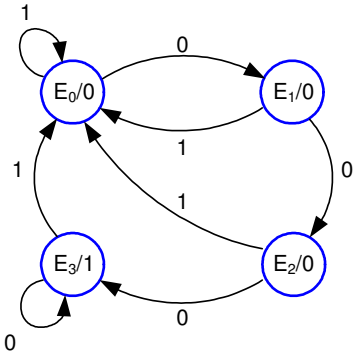


Figura 25 – Diagrama de estados para o detector de sequências.

Apenas há a salientar que o circuito permanece no estado E_3 desde que continuem a aparecer '0's na entrada do circuito – e como este estado corresponde à ocorrência de 3 ou mais '0's, então a saída do circuito deverá ser '1'.

Após a elaboração do diagrama de estados, é necessário proceder à codificação de estados. No diagrama foi utilizada a simbologia E_0...E_3 para representar os vários estados – é agora necessário codificá-los. Como são quatro estados diferentes, podem ser codificados utilizando dois bits, fazendo-se a codificação representada na Tabela 16.

Estado	Q_1Q_0
E_0	00
E_1	01
E_2	10
E_3	11

Tabela 16 – Codificação de estados.

Após serem codificados os estados, e antes de se obter a tabela de transições de estados, o diagrama de estados ficará mais claro se forem incluídas as combinações binárias correspondentes aos estados codificados, como representado na Figura 26.

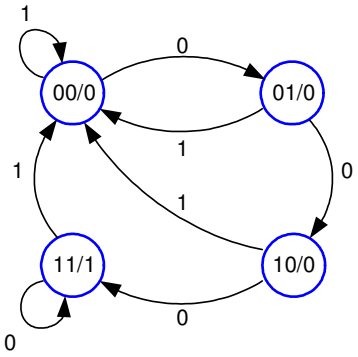


Figura 26 – Diagrama de estados para o detector de sequências (com os estados codificados).

Com base no diagrama, constrói-se a tabela de transições de estados. Na tabela, a primeira linha corresponde à situação em que o circuito está no estado '00' com a entrada a '0'. Nesse caso o próximo estado seria '01'. Se a entrada estivesse a '1', continuava no estado '00'. O resto da tabela é construído olhando para os círculos (o estado em que o circuito está) e para as setas (o valor da entrada).

Q_1	Q_0	X	Q_1'	Q_0'	J_1K_1	J_0K_0
0	0	0	0	1	0 x	1 x
0	0	1	0	0	0 x	0 x
0	1	0	1	0	1 x	x 1
0	1	1	0	0	0 x	x 1
1	0	0	1	1	x 0	1 x
1	0	1	0	0	x 1	0 x
1	1	0	1	1	x 0	x 0
1	1	1	0	0	x 1	x 1

Tabela 17 – Tabela de transição de estados.

Concluída a tabela, há que escolher o tipo de FFs a utilizar. Supondo que foram escolhidos FFs do tipo JK, acrescentam-se os valores lógicos para as entradas J e K de cada flip-flop (com base nas tabelas de excitação) e extraem-se as equações de entrada dos flip-flops.

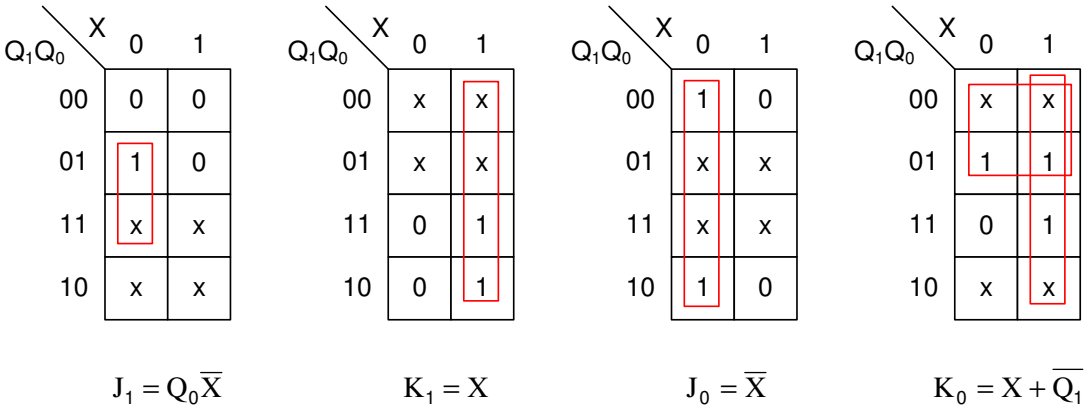


Figura 27 – Equações de entrada nos flip-flops.

Para além das equações de entrada nos flip-flops, é também necessário obter a equação de saída. Como neste caso o valor de saída só depende do estado em que o circuito se encontra (recorde que segue o modelo de *Moore*), será obtida uma equação para Y em função das variáveis de estado Q_1 e Q_0. Como a saída só é '1' no estado '11' (i.e., quando Q_1 e Q_0 estão ambos a '1'), é fácil de concluir que $Y=Q_1Q_0$.

Uma vez definidas todas as equações, pode-se então proceder ao desenho do circuito. Para resolver este problema foi obtido o circuito representado na Figura 28.

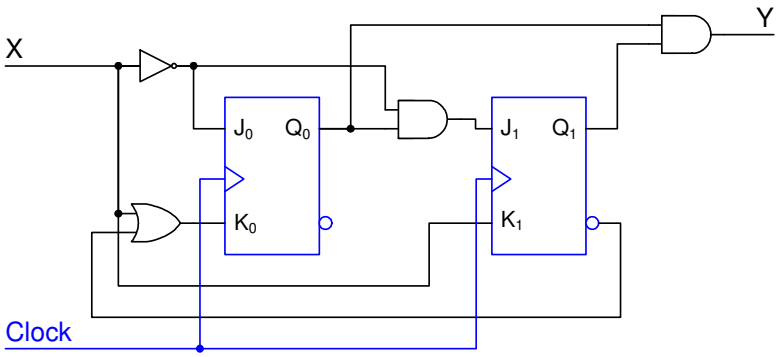


Figura 28 – Circuito resultante para o detector de sequências.

4. Registos

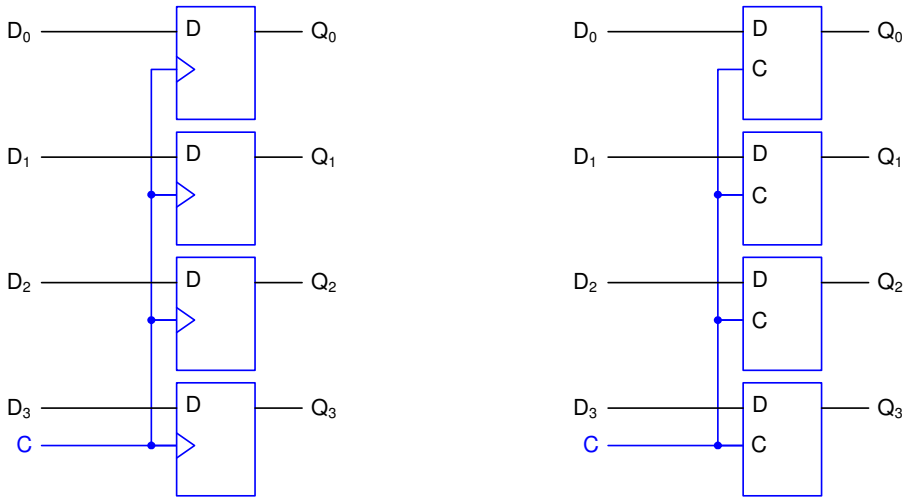
Existem inúmeras aplicações em que há interesse em armazenar informação organizada de um conjunto de bits. Um **registo** é circuito sequencial que permite um armazenamento básico, mas organizado, de um conjunto de vários bits de informação.

Num sistema digital, os registos são tipicamente utilizados para armazenar dados resultantes de operações aritméticas ou de transferências de informação (por exemplo, o processador guarda em registos dados que lê da memória). Muitas vezes os dados estão organizados segundo conjuntos de bits que se designam por *palavras*. Uma *palavra* é basicamente uma combinação de *n* bits que representa informação: um número, um carácter, uma instrução.

Faz muitas vezes sentido trabalhar com palavras como um todo e não com cada um dos bits individualmente. Os registos possibilitam um conjunto de funcionalidades básicas, entre as quais se destacam:

- ***Load*** ou ***Carregamento*** – guardar uma nova palavra no registo.
- ***Clear*** ou ***Reset*** – inicializar o registo com '0's.
- ***Shift*** ou ***Deslocamento*** – deslocar os bits dentro do registo (este tipo de operação é muito importante em multiplicações e divisões).

Um registo é construído com base em *flip-flops* ou em *latches*. Como foi visto anteriormente, cada flip-flop (ou latch) é capaz de armazenar um bit de informação. Sendo assim, faz sentido que um registo que lida com o armazenamento de *n* bits de informação (palavras de *n* bits) seja composto por *n flip-flops* ou *latches*, cada um responsável por guardar um bit de informação.



(a) Registo com flip-flops D (b) Registo com latches D

Figura 29 – Registos de 4 bits.

Considere os circuitos esquematizados na Figura 29. Ambos representam registos básicos que armazenam a informação fornecida através das linhas D_0 a D_3 – seriam registos que lidam com palavras de 4 bits e portanto seriam registos de 4 bits. A diferença entre os dois esquemas

consiste nos instantes de tempo em que novos valores podem ser carregados no registo: o registo construído com base em flip-flops D carrega os dados no instante em que C transita de '0' para '1', ao passo que o registo construído com base em *latches* responde durante todo o intervalo de tempo em que C se encontra a '1'.

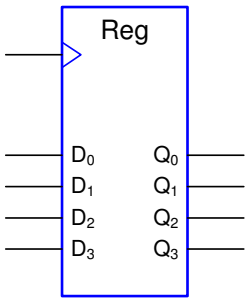


Figura 30 – Símbolo de um registo.

O símbolo de um registo consiste num rectângulo sobre o qual se representam as suas entradas e as suas saídas (ver Figura 30). O triângulo associado ao sinal de relógio indica que o registo é implementado com base em flip-flops *edge-triggered* – o carregamento de dados para o registo ocorre quando o relógio transita (de '0 para '1', neste caso).

4.1. Registos com carregamento e *reset*

Os registos apresentados na introdução, apesar de permitirem o armazenamento da informação, não servirão para muito num sistema mais complexo sincronizado por um sinal de relógio. Isto porque sempre que o relógio fosse de '0' para '1' seriam carregados novos dados no registo, o que poderia não ser desejável. No entanto, a partir destes registos básicos podem ser construídos outros registos que permitem maior controlo sobre a informação armazenada.

Uma forma de se poder controlar o carregamento (ou não) de novos dados no registo seria acrescentar uma variável de controlo *Load* que funcionaria do seguinte modo: se *Load* estiver a '1' o registo carrega novos dados, caso contrário o registo mantém os dados que já se encontravam armazenados.

Intuitivamente, uma forma de implementar este esquema seria acrescentar uma porta AND ao sinal de relógio, tal como esquematizado na Figura 31.

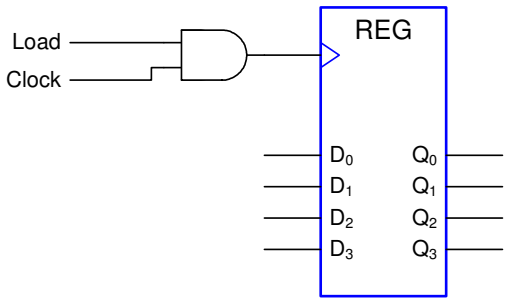
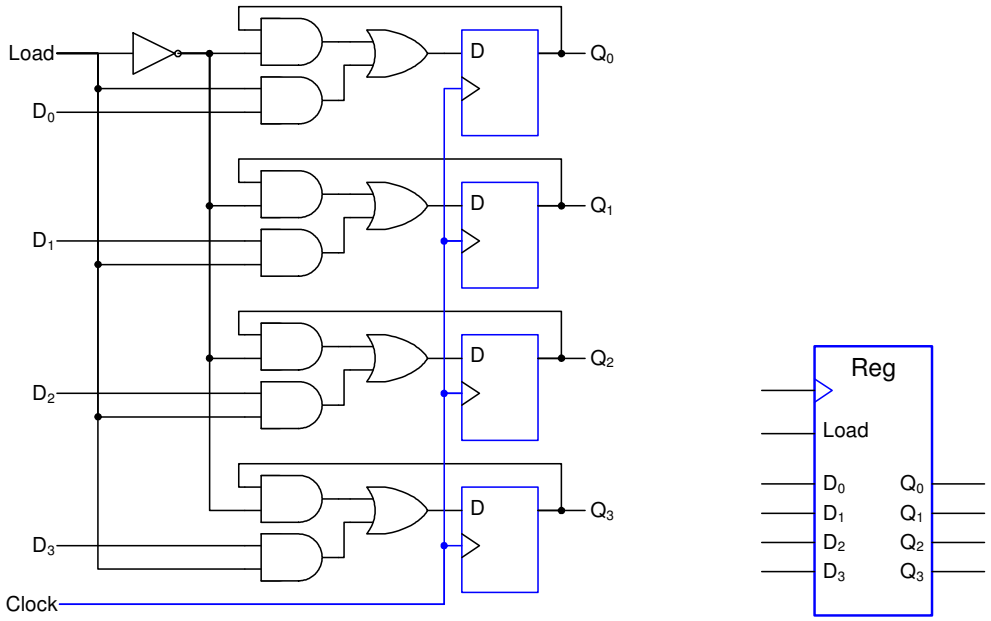


Figura 31 – Registo de 4 bits com controlo de carregamento.

Com *Load* a ‘0’, o sinal que chega à entrada de relógio do registo é ‘0’, logo não há transições e não se carregam novos dados. Com *Load* a ‘1’ o relógio passa normalmente e portanto carregam-se dados.

Esta técnica, na qual são utilizadas portas lógicas de forma a manipular o sinal de relógio, designa-se por *clock gating*. Pode no entanto trazer inconvenientes quando os registos fazem parte de um sistema mais complexo, no qual seria desejável que todos os elementos reagissem ao mesmo tempo. Ao serem acrescentadas portas lógicas no caminho do sinal de relógio, é introduzido um atraso devido aos tempos de propagação dessas portas lógicas. Quer isto dizer que as transições do sinal de relógio chegam atrasadas aos elementos com *clock gating* e portanto esses elementos reagem mais tarde do que outros. Tal facto designa-se por *escorregamento do sinal de relógio* (*clock skew*), e pode conduzir a problemas de sincronização.

Em alternativa às técnicas de *clock gating*, é preferível usar um método que se designa por *carregamento paralelo síncrono*. Há semelhança do que foi visto, existe uma variável de controlo *Load*, que, quando activada, faz com que sejam carregados novos dados no registo, mas quando desactivada, o registo é “forçado” a recarregar os dados que já lá se encontravam armazenados. Uma forma simples de implementar esta ideia encontra-se ilustrada na Figura 32.



(a) Esquema de ligações. (b) Símbolo.

Figura 32 – Registo de 4 bits com carregamento paralelo síncrono.

Para compreender o que é feito, considere o esquema de um “andar” deste registo, representado na Figura 33. De um modo geral, para se perceber como funciona um registo basta analisar o comportamento de um dos andares que o compõem – os restantes têm um comportamento semelhante, diferenciando-se apenas pela ordem do bit informação que tratam.

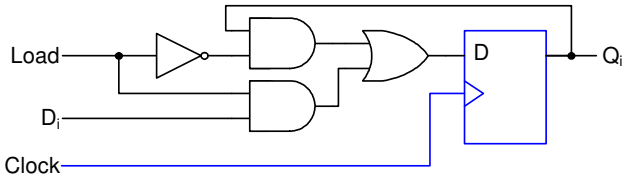


Figura 33 – Andar de um registo com carregamento paralelo síncrono.

Com a variável *Load* a '0' a saída da porta AND de baixo será '0', ao passo que a saída da porta de cima será Q_i. Deste modo, na entrada D do flip-flop estará o valor de Q_i. Quando chegar um impulso de relógio o flip-flop carregará Q_i – e portanto não altera o seu estado.

Quando *Load* está a '1', a saída da porta AND de baixo será D_i, ao passo que a saída da porta de cima será '0'. Sendo assim, a entrada D do flip-flop terá o valor de D_i. Quando chegar o impulso de relógio, a saída do flip-flop assumirá o valor de D_i – carrega-se portanto o valor de D_i.

Na Tabela 18 sintetizam-se estas operações. Os símbolos Q_i e Q_i' designam o estado actual e o próximo estado do i-ésimo flip-flop do registo, respectivamente.

Clock	Load	Operação
↑	0	$Q_i' = Q_i$ (manter o valor de Q_i)
↑	1	$Q_i' = D_i$ (carregar o valor de D_i)

Tabela 18 – Operações no registo com carregamento paralelo síncrono.

Pode portanto concluir-se que, estando *Load* a '1', o registo carrega os valores de D_0 a D_{n-1} nos vários flip-flops quando chegar um impulso de relógio. Com *Load* a '0' o registo mantém os valores $Q_0 ... Q_{n-1}$ que se encontravam armazenados anteriormente.

Outra forma de implementar um andar de um registo com carregamento paralelo síncrono seria utilizando multiplexers. *Load* seria ligado ao selector, D_i e Q_i seriam ligados às entradas de dados do multiplexer, como o esquematizado na Figura 34. Quando *Load* está activado selecciona-se D_i; com *Load* desactivado selecciona-se Q_i.

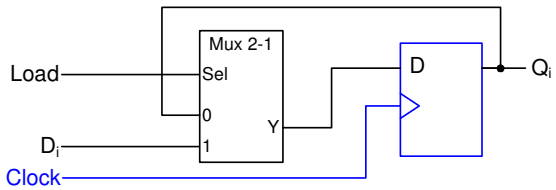
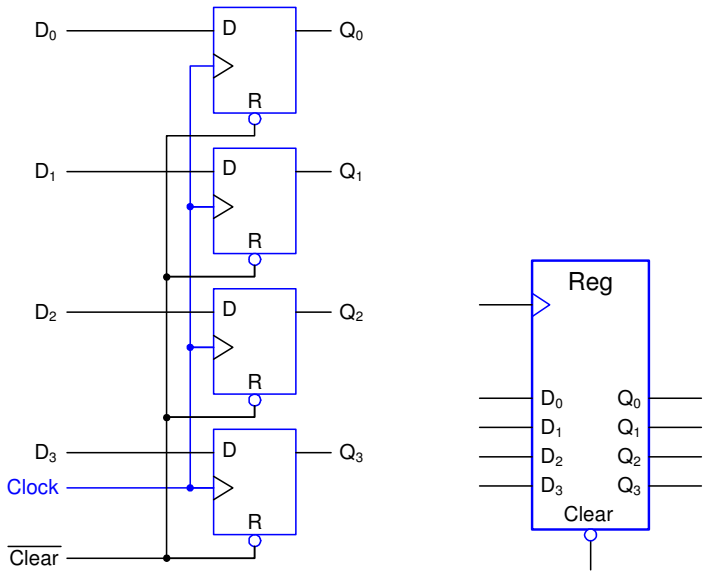


Figura 34 – Andar de um registo com carregamento paralelo síncrono (utilizando um multiplexer 2-1).

Outra operação que é importante nos registos é a sua inicialização a '0's, extremamente útil numa grande variedade de aplicações, com destaque para as operações aritméticas. A inicialização dos registos pode ser feita de forma assíncrona com o sinal de relógio ou de uma forma síncrona. Efectuar uma operação de *Clear* (*reset*) assíncrono é em geral simples, pois muitas vezes os próprios FFs que fazem parte do registo já possuem uma entrada de *Reset* assíncrono. Deste modo basta existir um sinal que actua em simultâneo sobre todas as entradas de *reset* dos flip-flops, tal como o esquematizado na Figura 35.



(a) Esquema de ligações (b) Símbolo

Figura 35 – Registo de 4 bits com *reset* assíncrono.

A elaboração de *clear* síncrono é semelhante ao carregamento paralelo síncrono, mas em vez de se fazer o carregamento com um valor D_i, faz-se com que seja carregado o valor lógico '0' em todos os flip-flops do registo.

É ainda possível um registo possuir ambas as possibilidades: carregamento paralelo e *clear* síncronos. Nestes casos é habitual o *clear* prevalecer sobre o carregamento, i.e., quando *Clear* está activado, os flip-flops são forçados a carregar '0's, independente do nível lógico de *Load*. Um possível esquema para um andar de um registo com carregamento paralelo e *clear* síncronos encontra-se esquematizado na Figura 36.

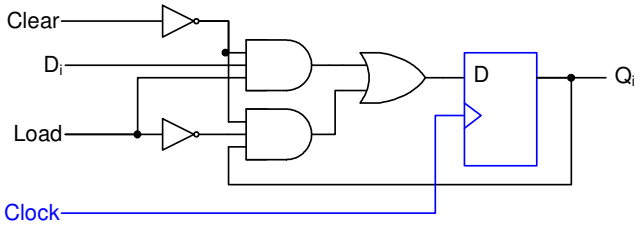
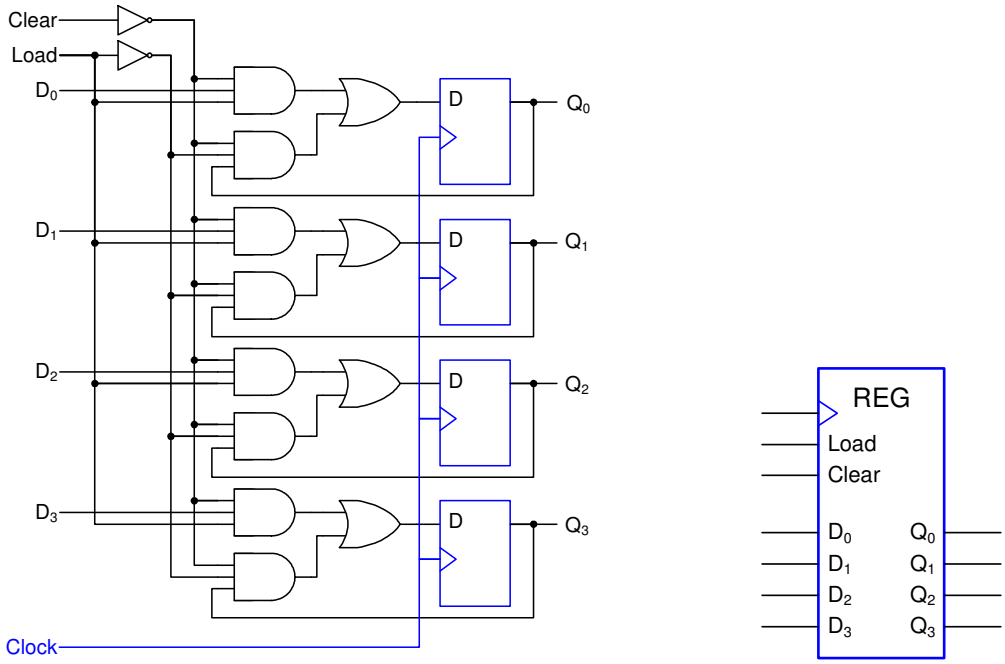


Figura 36 – Andar de um registo com carregamento paralelo e *clear* síncronos.

Repare que com quando a variável *Clear* está a '1', as saídas das portas AND são '0' e consequentemente, a entrada do FF D também é '0'. Quando ocorrer um impulso de relógio, a saída Q_i ficará a '0'. Com *Clear*=0, o circuito fica com um comportamento semelhante ao que já foi analisado na Figura 33. As operações possíveis encontram-se resumidas na Tabela 19.

Clock	Clear	Load	Operação
↑	0	0	$Q_i' = Q_i$ (manter o valor de Q_i)
↑	0	1	$Q_i' = D_i$ (carregar o valor de D_i)
↑	1	X	$Q_i' = 0$ (*Reset*)

Tabela 19 – Operações num registo com carregamento paralelo e *clear* síncronos.



(a) Esquema interno (b) Símbolo

Figura 37 – Registo de 4 bits com carregamento paralelo e *clear* síncronos

Com base em *n* andares idênticos pode ser construído um registo de *n* bits com carregamento paralelo e *clear* síncronos. A título de exemplo, representa-se na Figura 37 um registo de 4 bits com estas duas possibilidades.

4.2. Registos de deslocamento

Outras operações igualmente importantes são as chamadas operações de deslocamento. Uma operação de deslocamento consiste em fazer deslocar os bits de uma palavra guardada dentro de um registo. O deslocamento pode ser feito para a esquerda ou para a direita. Para compreender melhor uma operação de deslocamento, considere um registo R, de 8 bits, que contém armazenado o valor 0100 1011 (ver Figura 38).

Deslocar para a direita o conteúdo deste registo corresponderia a deixar o registo com o valor *b*010 0101, em que *b* é um bit novo que entra no registo, que tanto poderia valer '1' como '0'. De modo análogo, deslocar para a esquerda corresponderia a deixar o registo com o valor 1001 011*b*. Repare também que quando se desloca o conteúdo de um registo para a direita, o bit mais à direita sai do registo (e quando se desloca para a esquerda, sai o bit mais à esquerda).

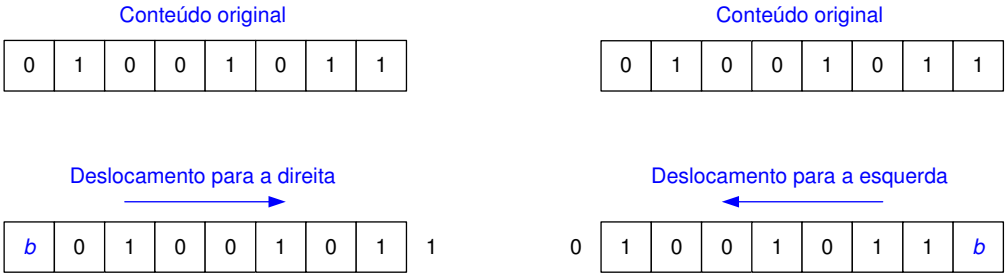


Figura 38 – Deslocamento do conteúdo de um registo.

Repare também que o deslocamento à direita e o deslocamento à esquerda podem ser vistos também como deslocamentos no sentido do bit menos significativo, ou no sentido do bit mais significativo, respectivamente.

As operações de deslocamento têm um papel muito importante na multiplicação e divisão binárias. Servem também para paralelizar dados que chegam numa linha série e vice-versa.

Consoante o bit que entra e o que é feito ao bit que sai, existem algumas designações específicas para alguns tipos de deslocamento:

- ***deslocamento lógico*** – neste tipo de deslocamento, o bit que entra é sempre '0';
- ***deslocamento aritmético*** – neste tipo de deslocamento, que só se faz para a direita, o bit que entra pela esquerda é igual ao bit mais significativo. Desta forma preserva-se o bit de sinal do número armazenado no registo, daí a designação *aritmético*.
- ***deslocamento rotativo*** – neste tipo de deslocamento o bit que sai pela esquerda entra pela direita (ou vice-versa, consoante a direcção do deslocamento).

Um registo básico que permite o deslocamento de bits encontra-se esquematizado na Figura 39. Repare que este registo possui uma única entrada – a entrada série *SI* – por onde vão entrando os bits a armazenar/deslocar. Quando ocorre um impulso de relógio, cada um dos flip-flops vai ser carregado com o conteúdo do flip-flop anterior (excepto claro está o primeiro FF, que é carregado com o valor de *SI*). Por outras palavras, à medida que chegam impulsos de relógio, o registo vai sendo preenchido com os bits que vão sendo apresentados em SI. Cada impulso de relógio origina um deslocamento para a esquerda.

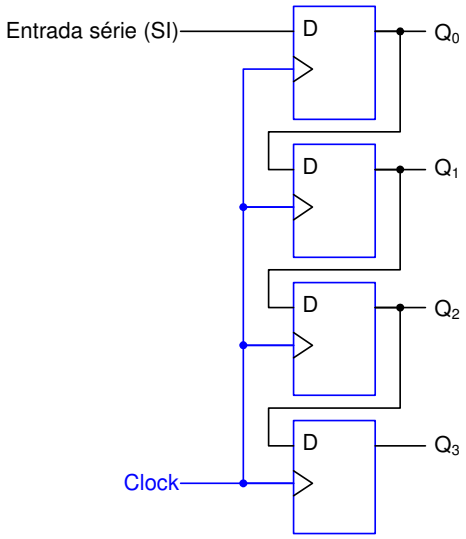


Figura 39 – Registo de deslocamento para a esquerda.

No fundo, o que acontece em cada impulso de relógio pode ser descrito do seguinte modo:

$$\begin{cases} Q_0 = SI \\ Q_i = Q_{i-1} \end{cases}$$

Este tipo de registo, embora muito simples, poderia ser utilizado para distribuir os bits que vão chegando à entrada série pelas linhas paralelas Q_0 a Q_3. Poderia ser por isso designado por *registo de deslocamento série-paralelo*.

Suponha agora que se pretende construir um registo de deslocamento que, para além de deslocar o conteúdo, possui também a hipótese de serem carregados novos valores. Tal registo seria um *registo de deslocamento com carregamento paralelo*.

Tal como foi visto anteriormente, para projectar um registo de *n* bits que permita efectuar estas operações, basta projectar um andar do registo e depois repetir esse andar *n* vezes. Um registo de deslocamento com carregamento paralelo deverá então permitir as operações sintetizadas na seguinte tabela:

Clock	*Shift*	*Load*	Operação
↑	0	0	$Q_i' = Q_i$ (manter o valor de Q_i)
↑	0	1	$Q_i' = D_i$ (carregar o valor de D_i)
↑	1	X	$Q_i' = Q_{i-1}$(deslocamento para a esquerda)

É fácil de verificar que um circuito que efectua estas operações é, por exemplo, o circuito representado na Figura 40 (Q_{i-1} representa a saída do FF anterior).

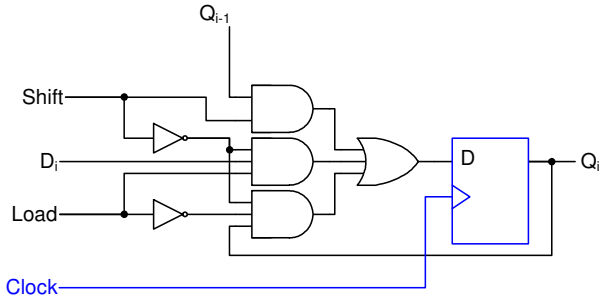


Figura 40 – Andar de um registo de deslocamento com carregamento paralelo síncrono.

Com base em 4 "andares" semelhantes pode-se construir um registo de deslocamento de 4 bits com carregamento paralelo.

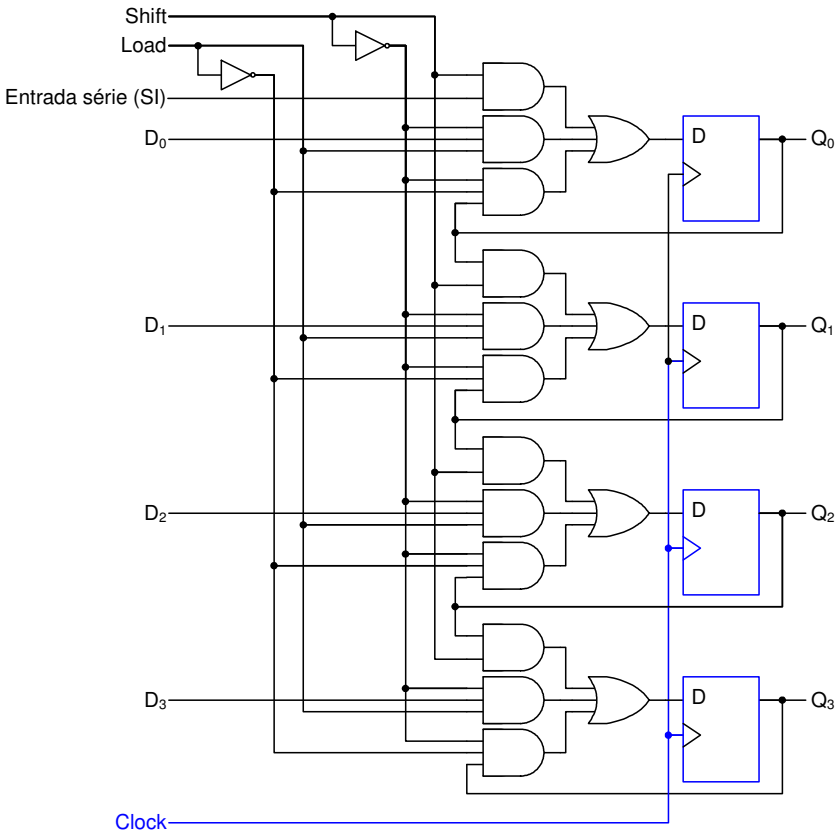


Figura 41 – Registo de deslocamento 4 bits com carregamento paralelo.

O registo projectado permite deslocamento para esquerda (na direcção do bit mais significativo). Existem registos que permitem deslocamento nos dois sentidos. Recorrendo a multiplexers é relativamente fácil projectar um registo que possibilite o deslocamento da informação em ambas as direcções. Assim, utilizando um multiplexer 4-1, um andar de um registo com estas funcionalidades poderia ser o representado na Figura 42, onde Q_{i-1} e Q_{i+1} representam as saídas do flip-flop anterior e do flip-flop seguinte, respectivamente.

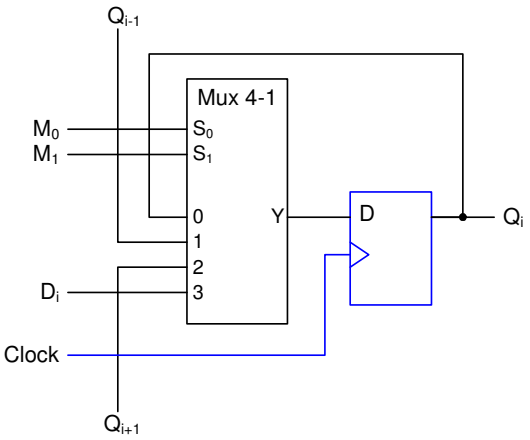


Figura 42 – Andar de um registo de deslocamento em ambas as direcções e com carregamento paralelo.

Na Tabela 20 encontram-se sintetizadas as operações que seriam possíveis de efectuar com este registo:

Clock	M_1	M_0	Operação
↑	0	0	$Q_i' = Q_i$ (manter o valor de Q_i)
↑	0	1	$Q_i' = Q_{i+1}$ (deslocamento para a direita)
↑	1	0	$Q_i' = Q_{i-1}$ (deslocamento para a esquerda)
↑	1	1	$Q_i' = D_i$ (carregar o valor de D_i)

Tabela 20 – Operações no registo com deslocamento bidireccional.

Se fosse projectado um registo de 4 bits, com base em 4 "andares" idênticos ao da Figura 42, o símbolo para o circuito resultante seria o representado na Figura 43. Repare que possui duas entradas adicionais – RSI (*Right serial input*) e LSI (*left serial input*) que seriam as entradas série para os novos bits que entram quando é feito um deslocamento para a esquerda ou para direita, respectivamente.

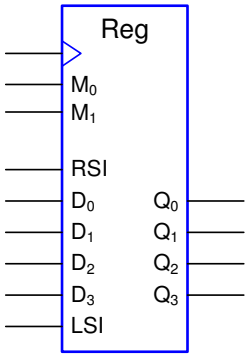


Figura 43 – Símbolo de um registo com deslocamento bidireccional.

5. Contadores

Um **contador** consiste num circuito sequencial que segue uma sequência de estados pré-determinada. A designação contador justifica-se pois essa sequência de estados corresponde normalmente a uma sequência de contagem numérica. Por outro lado, é comum estabelecer uma analogia entre contadores e registos, uma vez que a maioria dos contadores possuem características semelhantes às dos registos (tipicamente permitem o carregamento e armazenamento de informação), como será visto mais adiante.

5.1. Contadores binários

A classe de contadores mais simples são os **contadores binários**. Um contador binário segue uma sequência de estados que corresponde à sequência do código binário natural. Esta sequência de estados corresponde a sequência de contagem do contador. Num contador binário de *n* bits, a sequência de contagem corresponde aos números entre 0 e 2^n-1 (e.g., um contador binário de 4 bits efectua uma contagem entre 0 e 15).

À semelhança dos registos, um contador binário de *n* bits é construído com base em *n* flip-flops. Consoante o modo como os flip-flops têm ligado o sinal de relógio, os contadores binários podem ser divididos em dois grupos distintos:

- Contadores binários ***ripple*** ou ***assíncronos*** – neste tipo de contadores, existe apenas um flip-flop está ligado ao sinal de relógio. Cada um dos restantes flip-flops reage com base no sinal de saída de outro flip-flop.

- Contadores binários ***síncronos*** – num contador síncrono, todos os flip-flops reagem em simultâneo, por possuírem um sinal de relógio comum a todos eles.

Um contador binário *ripple* de 4 bits pode ser observado na Figura 44. Repare na sua arquitectura: cada um dos flip-flops possui as entradas J e K ligadas ao nível lógico '1', o que significa que ao ocorrer uma transição de '1' para '0' na sua entrada de relógio, o flip-flop complementa o estado anterior.

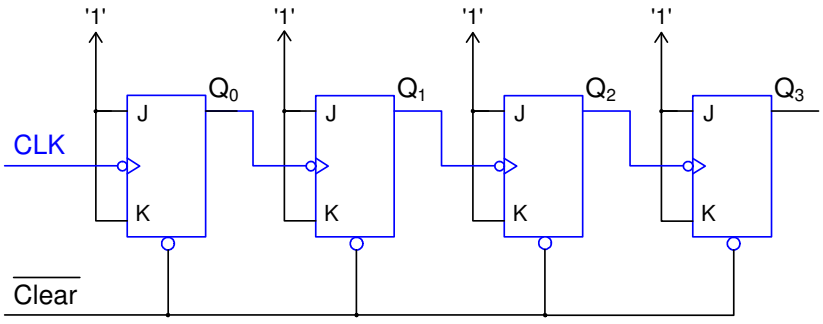


Figura 44 – Contador binário *ripple* de 4 bits.

Continuando a seguir o raciocínio, pode-se afirmar que, para o i-ésimo flip-flop trocar o seu estado, é necessário que o flip-flop anterior troque o seu estado duas vezes – de '0' para '1' e depois de '1' para '0'.

Considerando que inicialmente os flip-flops estão todos a '0' (poder-se-ia actuar sobre a entrada de *clear* para garantir essa situação), pode-se escrever que o estado do circuito é:

$$Q_3Q_2Q_1Q_0 = 0000$$

Após a ocorrência da primeira transição de '1' para '0' no sinal de relógio, o primeiro flip-flop complementa o seu estado, i.e., Q_0 passa de '0' para '1'. Quanto aos restantes flip-flops permanecem no mesmo estado, uma vez que nenhum dos sinais de relógio aos quais estão ligados transitou de '1' para '0'. Logo o 2º estado do circuito será:

$$Q_3Q_2Q_1Q_0 = 0001$$

Ao chegar a 2ª transição de '1' para '0' no sinal de relógio, o primeiro flip-flop complementa novamente o seu estado ($Q_0 = 0$). O segundo FF complementa-o também, pois a transição de Q_0 de '1 para '0' faz com que reaja, ou seja Q_1 vai passar de '0' para '1'. O circuito irá então para o estado

$$Q_3Q_2Q_1Q_0 = 0010$$

Continuando com esta abordagem poder-se-ia verificar com facilidade que o circuito segue uma sequência de estados que corresponde à sequência binária de 0000 a 1111, isto é, os números inteiros entre 0 e 15. Pode-se também observar o comportamento do circuito no diagrama temporal representado na Figura 45.

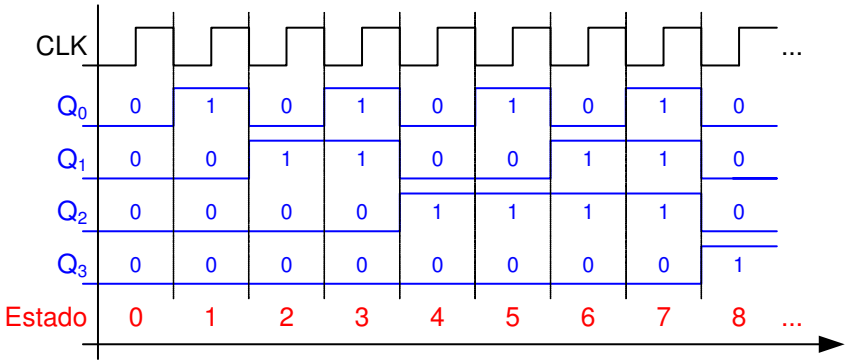


Figura 45 – Evolução temporal da sequência de estados do contador *ripple*.

Apesar do seu princípio de funcionamento ser muito simples, a utilização de um contador *ripple* deve ser evitada em sistemas mais complexos. Como já foi referido anteriormente (ver a secção sobre flip-flops), o tempo de propagação de um flip-flop não é nulo. Se considerarmos que cada flip-flop tem um tempo de propagação T_{PD}, até que o i-ésimo flip-flop mude o valor da saída decorrerá um tempo igual a i vezes T_{PD}. Por exemplo, na transição do estado '7' para o estado '8', o 4º flip-flop só irá reagir após todos os anteriores terem reagido – mas como cada um destes também só reage quando os anteriores reagirem, serão necessários 3 T_{PD} até que o 4º FF comece a reagir, mais T_{PD} até o valor da sua saída se alterar. Entre a transição do estado '7' para o estado '8' decorre então um intervalo de tempo correspondente a $4T_{PD}$.

Para além do atraso, o circuito passa momentaneamente por outros estados, permanecendo um intervalo de tempo aproximadamente igual a T_{PD} em cada um deles. No diagrama temporal da Figura 46 pretende-se ilustrar a situação descrita no parágrafo anterior.

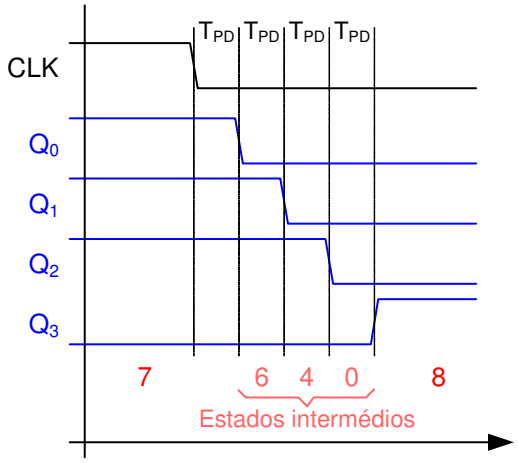


Figura 46 – Estados intermédios num contador *ripple*.

Estes estados intermédios são os chamados *estados instáveis*, e devem ser evitados em muitas situações.

Os contadores síncronos não possuem este problema, pois todos os flip-flops reagem ao mesmo tempo. Se todos os flip-flops tiverem tempos de propagação idênticos, não há passagens por estados instáveis, e as várias saídas do contador mudam os seus valores praticamente em simultâneo após ter decorrido um intervalo de tempo igual ao tempo de propagação do flip-flop.

Na Figura 47 pode ser observado um contador binário síncrono de 4 bits, construído com base em flip-flops JK *edge-triggered*. O funcionamento deste tipo de contadores é simples: as entradas J e K de cada flip-flop só ficam com o valor lógico '1' quando as saídas dos flip-flops anteriores estão todas a '1'. Por outras palavras, cada flip-flop só muda de estado quando todos os que o precedem estão no estado '1'. A única excepção é o flip-flop que corresponde ao bit menos significativo do estado (o 1º flip-flop), cujas entradas J e K estão sempre '1', para que mude o seu estado em todos os impulsos do sinal de relógio.

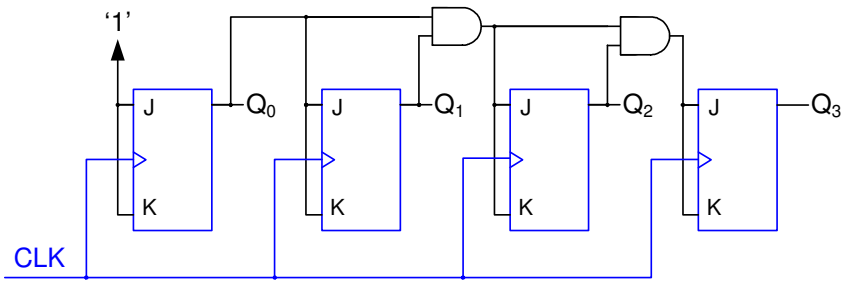


Figura 47 – Contador binário síncrono de 4 bits.

Existem situações em que interessa assinalar o último estado da sequência de contagem. Num contador binário, o último estado da sequência de contagem corresponde ao estado em que as saídas do contador estão todas a '1', i.e., Q_1=1, Q_2=1 ... Q_{n-1} = 1. Para assinalar o último estado

é habitual os contadores possuírem uma saída extra designada por *fim de contagem* (*terminal count*). No circuito anterior é relativamente simples implementar uma saída deste género – bastaria uma porta lógica AND entre as várias saídas Q_i.

Para além do fim de contagem, um contador costuma também possuir uma entrada que, quando activada, deixa o contador seguir a sequência de contagem, mas que quando está desactivada, pára a contagem, fazendo com que o contador a permaneça no estado em que estava no momento da paragem. Tal entrada costuma ser designada por *enable* de contagem (*count enable*). Para conseguir que um contador páre a contagem, basta que se coloquem a '0' as entradas J e K dos flip-flops, fazendo com que estes mantenham o estado actual.

Estes sinais extra podem ser implementados com facilidade acrescentando material ao contador síncrono anterior. O resultado é o contador representado na Figura 48, onde a entrada *CE* e a saída *TC* são os sinais de *enable* de contagem e fim de contagem, respectivamente.

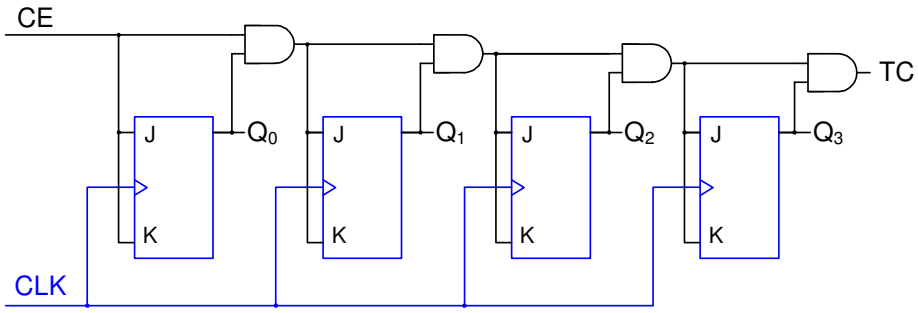


Figura 48 – Contador binário síncrono de 4 bits com *enable* e fim de contagem.

5.2. Contadores especiais

Para além dos contadores binários, existem também outros contadores que seguem sequências diferentes do código binário natural, ou então que permitem contagens descendentes.

Entre os contadores mais comuns, destacam-se:

- *Contadores BCD* – contadores cuja sequência de contagem corresponde à representação em binários dos números entre 0 e 9 (de 0000 a 1001);
- *Contadores Up/Down* – contadores (binários ou não) que possuem uma variável de entrada que permite controlar a direcção da contagem (contagem crescente ou decrescente);
- *Contadores módulo-N* – contadores que seguem uma sequência de N estados, que habitualmente correspondem a números consecutivos – entre os mais comuns encontram-se o contador módulo-6 e o módulo-12, que contam de 0 a 5 e de 0 a 11, respectivamente [5];
- *Contadores com carregamento paralelo* – possuem uma entrada de *load* que, quando activada permite carregar um dado valor no contador. Esse valor é especificado em entradas paralelas de dados, à semelhança de um registo.

[5] Ambos costumam ser utilizados em relógios digitais: o módulo-6 conta o algarismo das dezenas para os minutos/segundos e o módulo-12 conta as horas nos relógios que distinguem AM/PM.

5.3. Projecto de contadores

Qualquer contador pode ser projectado com base nos procedimentos de projecto para circuitos sequenciais. Como as saídas do contador são dadas directamente pelos estados em que os flip-flops se encontram, os contadores seguem o modelo de *Moore*.

Tal como já foi feito em secções anteriores, nada melhor do que um exemplo para ajudar a perceber como se projecta a base de um contador. Suponha que se pretende projectar um contador *up/down* de 3 bits. Recorde que um contador *up/down* possui uma entrada que permite controlar a direcção da contagem – essa variável de controlo será designada por $U/\overline{D}$. O contador efectua contagem crescente ou decrescente consoante essa variável esteja a '1 ou a '0', respectivamente. Como o contador é de 3 bits, a sequência de contagem será de 0 a 7 (ou de 7 a 0, se estiver em modo decrescente).

Em primeiro lugar traduz-se o problema por um diagrama de estados. O diagrama resultante encontra-se representado na Figura 49.

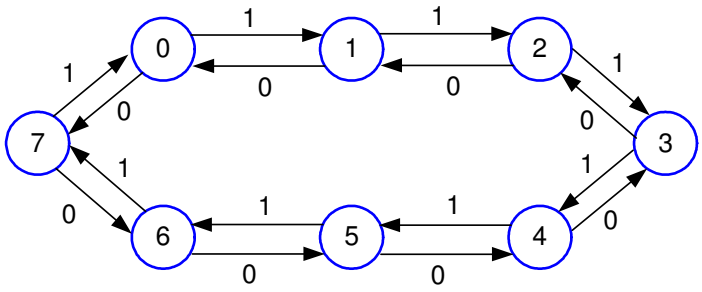


Figura 49 – Diagrama de estados do contador *up/down* de 3 bits.

Como se trata de um contador, é conveniente que a codificação de estados siga a sequência pretendida. Assim, o estado '7' corresponde a '111', o 6 a '110', o 5 a '101' e assim sucessivamente. Com base no diagrama pode-se construir a tabela de transições de estados:

Q_2	Q_1	Q_0	$U/\overline{D}$	Q_2'	Q_1'	Q_0'
0	0	0	0	1	1	1
0	0	0	1	0	0	1
0	0	1	0	0	0	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	1	1
0	1	1	0	0	1	0
0	1	1	1	1	0	0
1	0	0	0	0	1	1
1	0	0	1	1	0	1
1	0	1	0	1	0	0
1	0	1	1	1	1	0
1	1	0	0	1	0	1
1	1	0	1	1	1	1
1	1	1	0	1	1	0
1	1	1	1	0	0	0

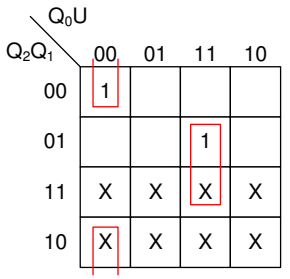
Tabela 21 – Tabela de transição de estados do contador *up/down* de 3 bits.

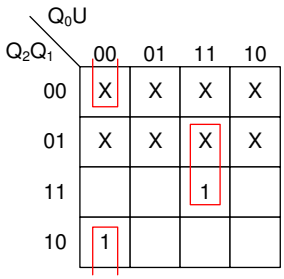
Neste ponto pode-se optar pela utilização de flip-flops JK ou flip-flops D. Como a utilização de flip-flops JK conduz geralmente a equações mais simples, optou-se pela utilização de FFs deste tipo. É portanto necessário acrescentar à tabela os valores de J e K de modo a que sejam efectuadas as transições de estados pretendidas. O resultado pode ser observado na Tabela 22.

Q_2	Q_1	Q_0	$U / \overline{D}$	Q_2'	Q_1'	Q_0'	J_2K_2	J_1K_1	J_0K_0
0	0	0	0	1	1	1	1X	1X	1X
0	0	0	1	0	0	1	0X	0X	1X
0	0	1	0	0	0	0	0X	0X	X1
0	0	1	1	0	1	0	0X	1X	X1
0	1	0	0	0	0	1	0X	X1	1X
0	1	0	1	0	1	1	0X	X0	1X
0	1	1	0	0	1	0	0X	X0	X1
0	1	1	1	1	0	0	1X	X1	X1
1	0	0	0	0	1	1	X1	1X	1X
1	0	0	1	1	0	1	X0	0X	1X
1	0	1	0	1	0	0	X0	0X	X1
1	0	1	1	1	1	0	X0	1X	X1
1	1	0	0	1	0	1	X0	X1	1X
1	1	0	1	1	1	1	X0	X0	1X
1	1	1	0	1	1	0	X0	X0	X1
1	1	1	1	0	0	0	X1	1X	X1

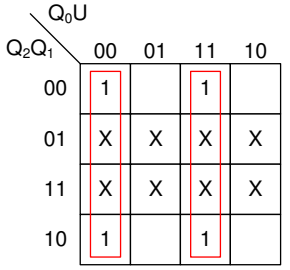
Tabela 22 –Valores a aplicar aos Js e aos Ks.

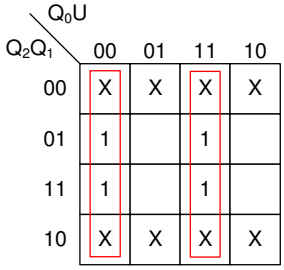
Com base na tabela, utilizando simplificação por mapas de Karnaugh (e também algumas propriedades da álgebra de Boole) chegam-se às seguintes expressões (por uma questão de simplicidade, U designa a variável de entrada $U / \overline{D}$):





$$J_2 = K_2 = \overline{Q_1}\,\overline{Q_0}\,\overline{U} + Q_1 Q_0 U = \overline{Q_1 \oplus U} \cdot \overline{Q_0 \oplus U}$$





$$J_1 = K_1 = \overline{Q_0}\,\overline{U} + Q_0 U = \overline{Q_0 \oplus U}$$

Quanto a J_0 e K_0, olhando para a tabela é fácil de concluir que J_0=K_0=1. Têm-se portanto as seguintes equações:

$$J_2 = K_2 = \overline{Q_1 \oplus U} \cdot \overline{Q_0 \oplus U}$$

$$J_1 = K_1 = \overline{Q_0 \oplus U}$$

$$J_0 = K_0 = 1$$

Com base nas equações desenha-se o circuito resultante:

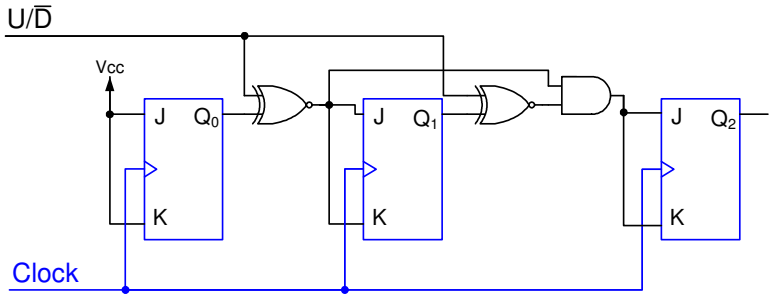


Figura 50 – Contador *up/down* de 3 bits.

Um possível símbolo para este contador seria:

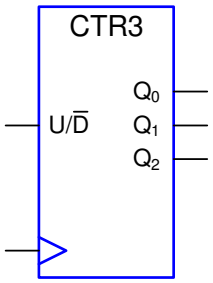


Figura 51 – Símbolo para o contador *up/down* de 3 bits.

Lista de revisões

Versão	Autor	Data	Comentários
0.01	TB	Jan./2005	Versão *draft* inicial para publicação antecipada.
0.01a	TB	Jan./2005	Correcção de gralha na Figura 22
0.02	TB	Out./2009	Correcção de diversas gralhas, reformulação de texto, reformatação do documento
0.02a	TB, JPO	Nov./2009	Correcção de algumas gralhas no texto (até à secção 4)
0.02b	TB, JPO	Nov./2009	Correcção de algumas gralhas no texto (da secção 4 até ao fim)