Capítulo 8

Circuitos Sequenciais Síncronos

Cuatro son las historias. Durante el tiempo que nos queda seguiremos narrándolas, transformadas.
Jorge Luis Borges, El oro de los tigres.

Um circuito combinacional é um circuito com entradas e saídas digitais, cujo comportamento funcional é definido por uma função que especifica os valores das saídas para cada uma das combinações de entradas. Seu comportamento temporal é determinado pelo tempo de propagação, que determina um limite superior para o intervalo entre o instante em que as entradas estabilizaram e as saídas estabilizem nos valores determinados pela especificação funcional. Circuitos combinacionais não podem conter ciclos, curto-circuitos entre saídas e tampouco entradas desconectadas.

Um *circuito sequencial* produz saídas que dependem da sequência com que os valores são apresentados às suas entradas. Num dado instante, a saída de um circuito sequencial depende da sequência de todos os valores apresentados às suas entradas. Posto de outra forma, o *estado atual* de um circuito sequencial depende das entradas e da sequência de todos os estados anteriores. Circuitos sequenciais empregam, necessariamente, alguma forma de memória para armazenar o estado atual.

Circuitos sequenciais são extremamente úteis porque seu comportamento pode ser definido como uma *sequência de estados*, que é determinada pela sequência de entradas. Um computador nada mais é do que um circuito sequencial de grande complexidade, cuja sequência de estados é determinada pelo programa que está sendo executado e pelos dados de entrada. Este capítulo[1] define o que se entende por um *circuito sequencial síncrono* (CSS) e apresenta um conjunto de técnicas de projeto destes circuitos.

Dependendo de como são interligados os *flip-flops* de um registrador, sua sequência de estados pode representar um subconjunto dos números naturais, conforme mostra a Seção 8.2, que trata de *contadores*. Contadores podem produzir sinais de sincronização particularmente úteis e estes são discutidos na Seção 8.2.10.

A Seção 8.3 apresenta uma outra maneira de interligar os *flip-flops* para formar *registradores de deslocamento*, que permitem deslocar um certo número de bits, de uma ou mais posições para a direita, ou para a esquerda. Estes registradores são os componentes básicos de qualquer circuito de comunicação serial, nos quais os bits são enviados ou recebidos um a um através

[1] © Roberto André Hexsel, 2012-2021. Versão de 15 de março de 2021.

de um fio ou através de um par de antenas.

A Seção 8.4 discute a metodologia para o projeto de uma classe particularmente útil de CSSs, que são as *máquinas de estados finitas*. Estas máquinas de estados (MEs) podem ser consideradas uma generalização de circuitos contadores.

A Seção 8.5 introduz uma metodologia de projeto para máquinas de estados complexas. Esta metodologia permite o projeto rápido e eficaz de máquinas complexas, chamadas de *sequenciadores* ou de *controladores*. Os *microcontroladores* são máquinas de estado cujo comportamento é determinado por uma forma rudimentar de programa.

Quando há abundância de recursos, especialmente *flip-flops*, podemos implementar máquinas de estado através de uma 'tradução' direta do *diagrama de estados*. Esta forma de implementação de MEs, com *um flip-flop por estado* é discutida na Seção 8.6.

A Seção 8.7 apresenta exemplos de circuitos sequenciais relativamente complexos, obtidos da composição de blocos simples como registradores, memórias, e circuitos combinacionais como somadores e multiplicadores. A operação de tais circuitos pode ser coordenada por máquinas de estado ou por microcontroladores.

As quatro 'histórias' referidas na citação a Borges são (i) os contadores, (ii) os registradores de deslocamento, (iii) as máquinas de estado finitas, especialmente os microcontroladores, e (iv) os processadores, que são uma classe de microcontroladores cuja sequência de controle é o programa executado no processador. A quarta 'história' merece um capítulo inteiro, que é o Capítulo 10.

8.1 Uma Rápida Olhada no Relógio

Os circuitos que são o objeto deste capítulo são qualificados como "síncronos" porque as mudanças de estado acontecem sincronicamente a um sinal que define os instantes em que as transições podem ocorrer. Estes sinais são chamados de *relógio* por conta da periodicidade e regularidade das transições.

Um sinal contínuo no tempo e que apresenta comportamento cíclico pode ser descrito pelas suas *amplitude* e *frequência*. A amplitude é a distância entre seus valores máximos e mínimos, e em circuitos digitais a amplitude dos sinais é tal que varia entre valores modelados como 0 e 1 – isto é uma idealização, embora na prática a amplitude dos sinais permaneça (quase) sempre entre os limites de tensão das faixas aceitáveis como nível lógico 0 e o nível lógico 1 para uma dada tecnologia de circuitos integrados.

A frequência f de um sinal periódico é o número de vezes por segundo em que o ciclo do sinal se repete, e a unidade da frequência é Hertz, com unidade [Hz]. O período T de um sinal é a duração de um ciclo, e é medido em segundos [s]. Frequência e período são relacionados pela Equação 8.1.

$$T = 1/f \tag{8.1}$$

A Figura 8.1 mostra um período de uma *onda quadrada* que poderia ser usada como sinal de relógio. Esta forma de onda é chamada de 'quadrada' porque o semiciclo em 0 tem a mesma duração do semiciclo em 1.

A faixa de frequências empregadas em circuitos digitais é ampla, desde frequências abaixo de um Hertz ($f \propto 0,25Hz$), em sinais de trânsito, até uma dezena de gigahertz, em circuitos de

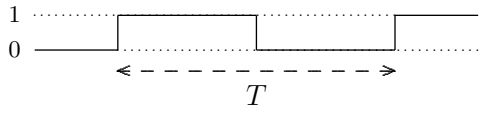


Figura 8.1: Período de um sinal periódico.

alto desempenho ($f \propto 10^{10} Hz$). Analogamente, a faixa dos períodos destes sinais também é ampla, desde uns poucos segundos até picossegundos ($T \propto 10^{-12} s$). A Tabela 8.1 relaciona as potências de 10 e os nomes usados para qualificar as frequências e períodos. Em se tratando de tempo ou frequência, são usadas potências de 10 e não potências de 2. Isto se deve ao desenvolvimento das telecomunicações ter ocorrido antes do desenvolvimento da computação automática, e a nomenclatura empregada pelos engenheiros de telecomunicações se manteve em uso.

Tabela 8.1: Tabela de potências de 10.

potência	nome	símbolo
-12	pico	p
-9	nano	n
-6	micro	μ
-3	mili	m
3	kilo	k
6	mega	M
9	giga	G
12	tera	T

8.2 Contadores

Contadores são circuitos sequenciais síncronos com um comportamento cíclico, e cuja sequência de estados pouco depende de outros estímulos externos além do sinal de relógio – em sua forma mais simples, um contador é um circuito que *conta pulsos do relógio*. Esta seção apresenta vários exemplos de contadores sequenciais síncronos. A Figura 8.2 mostra o modelo para um circuito sequencial sem nenhuma entrada. O sinal de relógio deve estar sempre presente mas não é contado como sendo uma entrada do CSS.

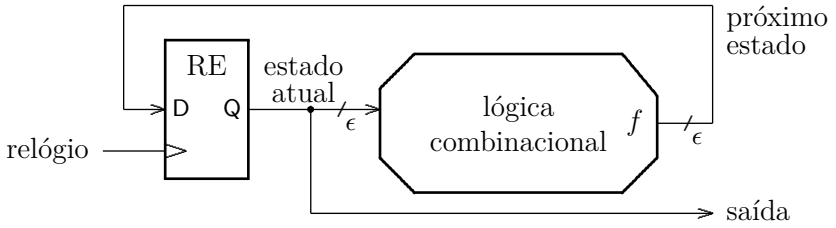


Figura 8.2: Modelo de contador como um circuito sequencial síncrono.

Quando falamos de contadores, usamos tuplas de bits para representar números naturais ou inteiros, como vimos na Seção 6.1.

8.2.1 Contador Módulo-2

Nosso primeiro circuito contador segue uma sequência de contagem de tamanho dois, e é chamado de *contador módulo-2*:

$$\text{módulo-2:} \quad 0 \mapsto 1 \mapsto 0 \mapsto 1 \cdots .$$

e sua tabela verdade é mostrada na Figura 8.2. O lado esquerdo da tabela mostra o estado atual (EA), que é o valor da contagem no ciclo c, e sua representação como um número natural. O lado direito da tabela mostra o próximo estado (PE), que é o valor da contagem no ciclo $c+1$, e sua representação como um número natural. O sinal d_0 é a entrada do *flip-flop*, e o sinal q_0 é sua saída.

Tabela 8.2: Sequência de contagem módulo-2.

	EA	PE	
$\mathbb{N}$	q_0	d_0	$\mathbb{N}$
0	0	1	1
1	1	0	0

Ao inspecionar a Tabela 8.2, percebe-se que a função que determina o próximo estado, em função do estado atual, é um inversor. O circuito do contador módulo-2 é mostrado na Figura 8.3.

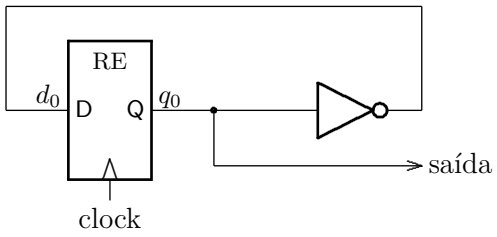


Figura 8.3: Contador módulo-2.

8.2.2 Contador Módulo-4

Um *contador módulo-4* conta numa sequência com quatro itens:

$$\text{módulo-4:} \quad 00 \mapsto 01 \mapsto 10 \mapsto 11 \mapsto 00 \mapsto 01 \mapsto 10 \cdots$$

e a Tabela 8.3 define esta sequência para um circuito com dois *flip-flops*.

Tabela 8.3: Sequência de contagem módulo-4.

	EA	PE	
$\mathbb{N}$	$q_1\, q_0$	$d_1\, d_0$	$\mathbb{N}$
0	0 0	0 1	1
1	0 1	1 0	2
2	1 0	1 1	3
3	1 1	0 0	0

Inspecionando a tabela, percebe-se que o próximo estado de q_0 é sempre o seu complemento, e a função de próximo estado para este *flip-flop* é $d_0 = \overline{q_0}$. O próximo estado de q_1 é 1 somente quando os dois bits do estado atual $\langle q_1, q_0 \rangle$ são distintos e portanto a função de próximo estado do *flip-flop* q_1 é $d_1 = q_1 \oplus q_0$. O circuito do contador módulo-4 é mostrada na Figura 8.4.

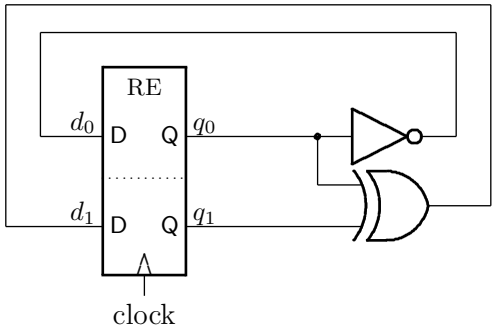


Figura 8.4: Contador módulo-4.

Embora pareçam um tanto inúteis, estes circuitos executam uma função importante: eles dividem a frequência do relógio por 2^n. A cada borda ascendente, a saída q_0 troca de estado, e o período do sinal em q_0 é o dobro daquele de *clock*. A cada segunda borda ascendente, a saída q_1 troca de estado, e o período do sinal em q_1 é o quádruplo daquele de *clock*. A frequência de q_0 é 1/2 da de *clock*, e a de q_1 é 1/4 da de *clock*.

8.2.3 Contador Módulo-8

Um *contador módulo-8* conta numa sequência com oito itens:

$$\text{módulo-8:} \quad 000 \mapsto 001 \mapsto 010 \mapsto \cdots \mapsto 110 \mapsto 111 \mapsto 000 \mapsto 001 \cdots$$

e a Tabela 8.4 define a função de próximo estado deste contador. Da mesma forma que nos outros contadores, a função de próximo estado para o *flip-flop* q_0 é $d_0 = \overline{q_0}$ porque o bit menos significativo sempre inverte a cada novo elemento da sequência: par $\mapsto$ ímpar $\mapsto$ par $\mapsto$ ímpar...

Tabela 8.4: Sequência de contagem módulo-8.

	EA	PE	
$\mathbb{N}$	$q_2\, q_1\, q_0$	$d_2\, d_1\, d_0$	$\mathbb{N}$
0	0 0 0	0 0 1	1
1	0 0 1	0 1 0	2
2	0 1 0	0 1 1	3
3	0 1 1	1 0 0	4
4	1 0 0	1 0 1	5
5	1 0 1	1 1 0	6
6	1 1 0	1 1 1	7
7	1 1 1	0 0 0	0

A metade de cima da tabela, ignorando-se a metade com $q_2 = 1$, é exatamente a mesma função do contador módulo-4 e portanto $d_1 = q_1 \oplus q_0$. O mesmo vale para a metade de baixo da tabela, com $q_2 = 1$.

A função de próximo estado d_2 pode ser obtida da inspeção da tabela verdade: q_2 se altera, e inverte, somente quando q_1 e q_0 são 1. Assim, $d_2 = q_2 \oplus (q_1 \wedge q_0)$. A Figura 8.5 mostra o circuito do contador módulo-8.

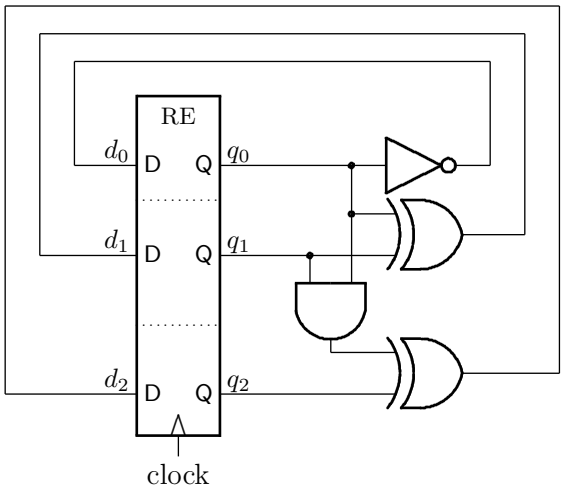


Figura 8.5: Contador módulo-8.

Intuitivamente, o bit n do contador inverte somente quando

$$1 + \langle q_{n-1} q_{n-2} \ldots q_1 q_0 \rangle$$

produz vai-um, e isso ocorre somente se os bits $q_{n-1} \ldots q_0$ são todos 1.

O diagrama de tempo do contador módulo-8 é mostrado na Figura 8.6. A cada borda ascendente do sinal de relógio, o sinal q_0 troca de valor, e seu período é o dobro daquele do sinal de relógio. Em todos as bordas em que $q_0 = 1$, q_1 troca de valor, e portanto seu período é o quádruplo daquele do relógio. Quando ambos, q_0 e q_1 são 1, q_2 troca de estado na próxima borda.

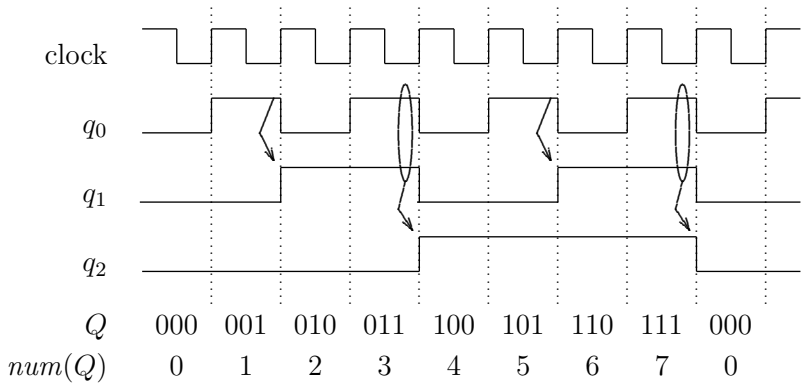


Figura 8.6: Comportamento do contador módulo-8.

A Figura 8.6 mostra também a sequência da contagem como uma sequência de números naturais. A cada borda ascendente do sinal de relógio, o estado da máquina muda para o estado que representa o próximo número na sequência de contagem módulo-8.

8.2.4 Contador Módulo-16

Um contador que conta na sequência de 0 a 15 necessita de $\log_2 16 = 4$ bits para armazenar o estado atual da contagem, e sua função de próximo estado é mostrada na Tabela 8.5.

Tabela 8.5: Sequência de contagem módulo-16.

	EA	PE	
$\mathbb{N}$	$q_3\, q_2\, q_1\, q_0$	$d_3\, d_2\, d_1\, d_0$	$\mathbb{N}$
0	0 0 0 0	0 0 0 1	1
1	0 0 0 1	0 0 1 0	2
2	0 0 1 0	0 0 1 1	3
3	0 0 1 1	0 1 0 0	4
4	0 1 0 0	0 1 0 1	5
5	0 1 0 1	0 1 1 0	6
6	0 1 1 0	0 1 1 1	7
7	0 1 1 1	1 0 0 0	8
8	1 0 0 0	1 0 0 1	9
9	1 0 0 1	1 0 1 0	10
10	1 0 1 0	1 0 1 1	11
11	1 0 1 1	1 1 0 0	12
12	1 1 0 0	1 1 0 1	13
13	1 1 0 1	1 1 1 0	14
14	1 1 1 0	1 1 1 1	15
15	1 1 1 1	0 0 0 0	0

As funções d_0, d_1 *e* d_2 são aquelas derivadas para os contadores de módulo menor que 16. A função de próximo estado d_3 também pode ser obtida ao inspecionar a tabela ou através de simplificação – quais são as condições para que q_3 troque de estado?

A Equação 8.2 mostra as funções de próximo estado para os quatro bits do contador módulo-16.

$$\begin{aligned} d_0 &= q_0 \oplus 1 = \overline{q_0} \\ d_1 &= q_1 \oplus q_0 \\ d_2 &= q_2 \oplus (q_1 \wedge q_0) \\ d_3 &= q_3 \oplus (q_2 \wedge q_1 \wedge q_0) \end{aligned} \tag{8.2}$$

A Figura 8.7 mostra a implementação do contador módulo-16. As funções de próximo estado para os bits d_2 e d_3 são parecidas e isso indica uma forma de implementar contadores para qualquer módulo que seja uma potência de dois.

Como uma alternativa ao circuito com portas lógicas, a função de próximo estado poderia ser implementada com uma memória ROM com 16 linhas de 4 bits. O conteúdo da ROM deve ser a coluna de próximo estado da Tabela 8.5, e os quatro bits do estado atual são ligados nas entradas de endereço. O conteúdo da linha indexada é entrada de cada um dos bits do registrador de estado.

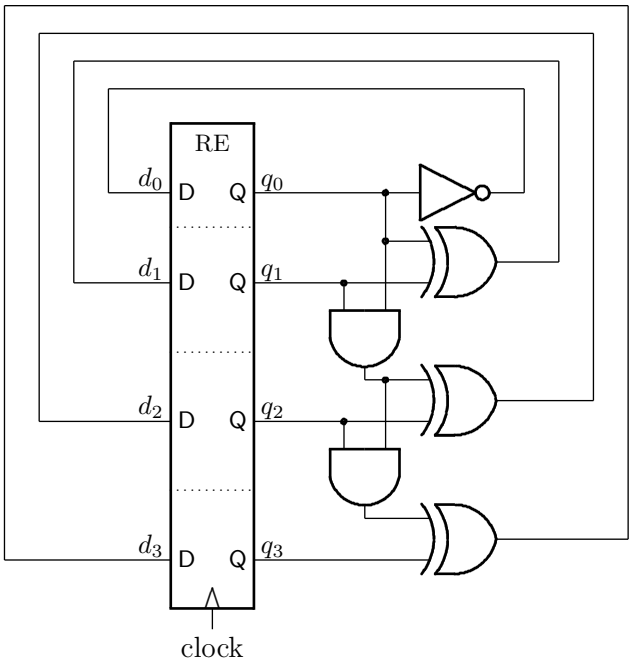


Figura 8.7: Contador módulo-16.

8.2.5 Divisão de Frequência

Um circuito contador pode efetuar a *divisão de frequência* do sinal de relógio. Considerando a frequência f do sinal de relógio, a saída do primeiro *flip-flop* do contador (q_0) produz um sinal com frequência $f/2$, a saída do segundo (q_1) tem frequência $f/4$, e assim sucessivamente. A cada divisão na frequência corresponde uma duplicação no período, como mostra a Figura 8.8.

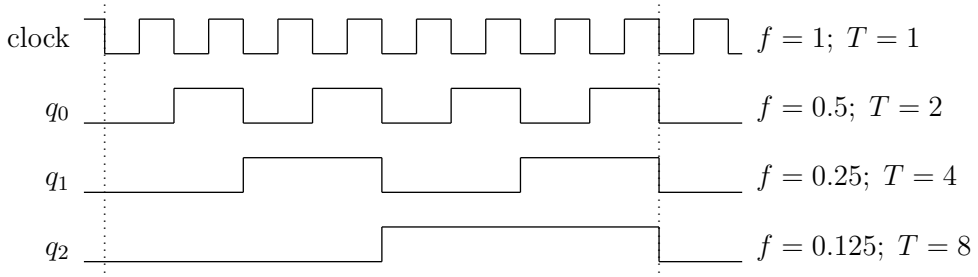


Figura 8.8: Divisão de frequência.

Exemplo 8.1 Vejamos como implementar um contador de três bits que conta na sequência de um Código de Gray, mostrada na Tabela 8.6.

Os Mapas de Karnaugh para as funções d_0, d_1, e d_2 são mostrados na Figura 8.9. As funções de próximo estado são definidas na Equação 8.3. O segundo termo de d_1 pode ser aproveitado em d_2. ◁

Tabela 8.6: Sequência de contagem Gray com três bits.

EA	PE
$q_2\ q_1\ q_0$	$d_2\ d_1\ d_0$
0 0 0	0 0 1
0 0 1	0 1 1
0 1 1	0 1 0
0 1 0	1 1 0
1 1 0	1 1 1
1 1 1	1 0 1
1 0 1	1 0 0
1 0 0	0 0 0

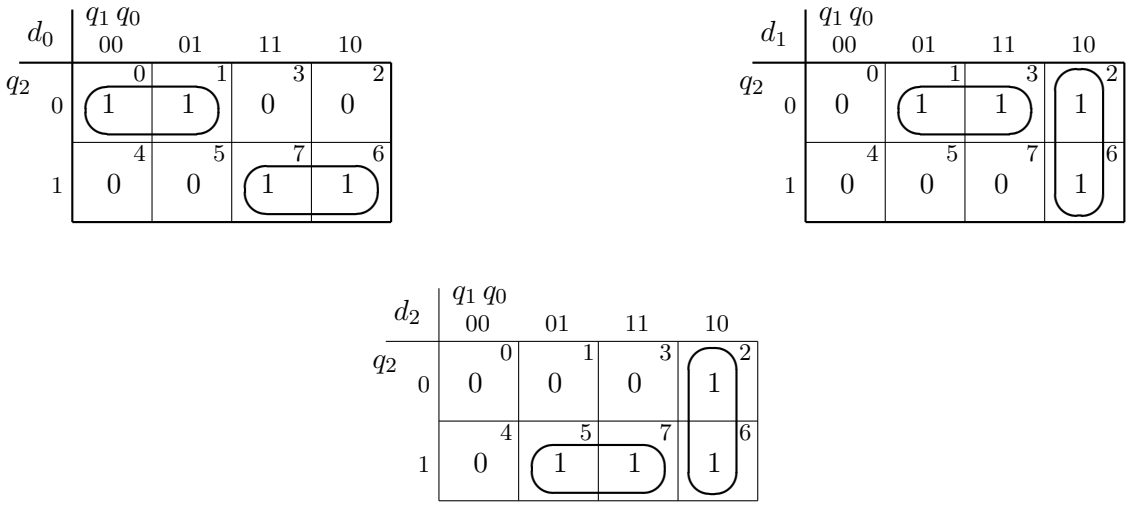


Figura 8.9: Mapas de Karnaugh para o contador Gray de três bits.

$$\begin{array}{lcl} d_0 & = & \overline{q_2} \wedge \overline{q_1} \vee q_2 \wedge q_1 \ = \ q_2 \oplus \overline{q_1} \\ d_1 & = & \overline{q_2} \wedge q_0 \vee q_1 \wedge \overline{q_0} \\ d_2 & = & q_2 \wedge q_0 \vee q_1 \wedge \overline{q_0} \end{array} \tag{8.3}$$

Os contadores que vimos até aqui são CSS sem entradas, e as saídas são o próprio estado do contador – sua função de saída é a função identidade. Vejamos alguns exemplos de contadores com sinais de controle que aumentam a sua funcionalidade.

Exemplo 8.2 O contador 74163, da família TTL, é um contador módulo-16, síncrono, com entradas de carga (*ld*) e inicialização (*clr*) síncronas. Se as entradas *p* e *t* estão ativas (*p = t = 1*), e *ld* e *clr* estão inativas (*ld = clr = 1*) então a contagem é incrementada na borda do relógio. Quando a contagem chega em quinze, o sinal *rco* fica em 1. '*rco*' é a abreviatura do Inglês para *ripple carry out*, que é o vai-um do incremento da contagem. A Figura 8.10 mostra o símbolo desse contador, e a Equação 8.4 define seu comportamento. O operador *mod* é o resto da divisão inteira. ◁

$$
\begin{array}{l}
p, pt, ld, clr, rco : \mathbb{B} \\
E, Q : \mathbb{B}^4 \\
163 : (\mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B}^4) \mapsto [(\mathbb{B}^4 \mapsto \mathbb{B}^4) \times \mathbb{B}]
\end{array}
$$

$$
\begin{array}{rcl}
(p \wedge t \wedge ld \wedge clr) & \Rightarrow & Q \leftarrow num^{-1}[(num(Q) + 1) \bmod 16] \\
\overline{clr} & \Rightarrow & Q \leftarrow 0 \\
(p \wedge t \wedge \overline{ld} \wedge clr) & \Rightarrow & Q \leftarrow E \\
(\overline{p} \vee \overline{t}) \wedge ld \wedge clr & \Rightarrow & Q \leftarrow Q \\
num(Q) = 15 & \Rightarrow & rco = 1
\end{array}
\tag{8.4}
$$

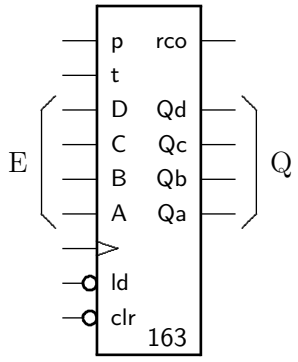


Figura 8.10: Símbolo e sinais do 74163.

Espaço em branco proposital.

Exemplo 8.3 A Figura 8.11 mostra um contador de 12 bits implementado com três 74163, e o diagrama de tempo desse contador indicando as mudanças de estado que causam incremento em mais de um dos contadores é mostrado na Figura 8.12. Note que a saída *rco* é combinacional e portanto esse sinal é um pouco atrasado com relação às mudanças de estado do contador. Quando a contagem do primeiro contador passa pelo estado 15, sua saída *rcoA* fica em 1 durante o tempo suficiente para habilitar o incremento do segundo contador, de 0 para 1. Após 255 pulsos de relógio, quando o primeiro e o segundo contadores estão ambos em 15, os sinais *rcoA* e *rcoB* ficam em 1, habilitando o incremento do terceiro contador.

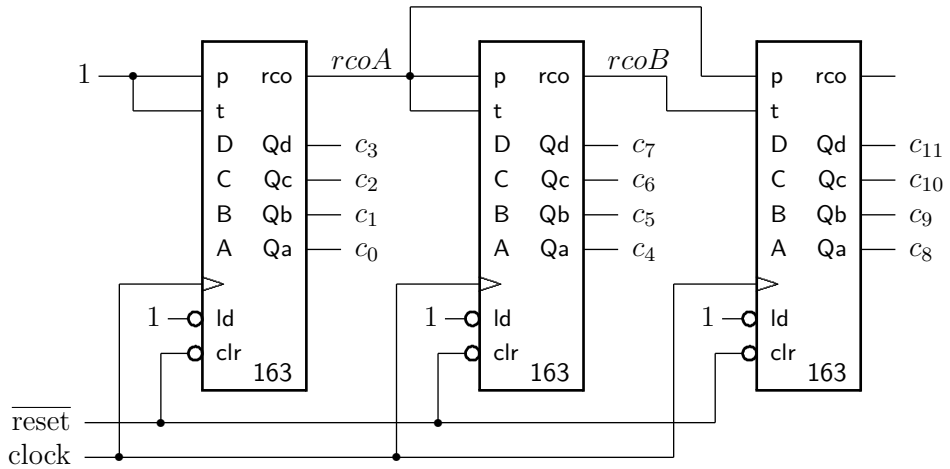


Figura 8.11: Contador síncrono de 12 bits com 74163.

A Figura 8.12 mostra o diagrama de tempo do contador de 12 bits, para as transições na contagem de 15 para 16, e de 255 para 256. Na transição de 15 para 16, o sinal *rcoA* fica ativo quando a saída do contador menos significativo $\langle c_3, c_2, c_1, c_0\rangle$ atinge 15 e isso permite que o segundo contador $\langle c_7, c_6, c_5, c_4\rangle$ seja incrementado. Quando a contagem atinge 255, os sinais *rcoA* e *rcoB* estão ambos ativos e isso permite que o contador mais significativo $\langle c_{11}, c_{10}, c_9, c_8\rangle$ seja incrementado. Note que é o sinal *rcoA* quem define o instante em que o terceiro contador é incrementado, e que o pulso em *rcoA* dura aproximadamente um ciclo de relógio. ◁

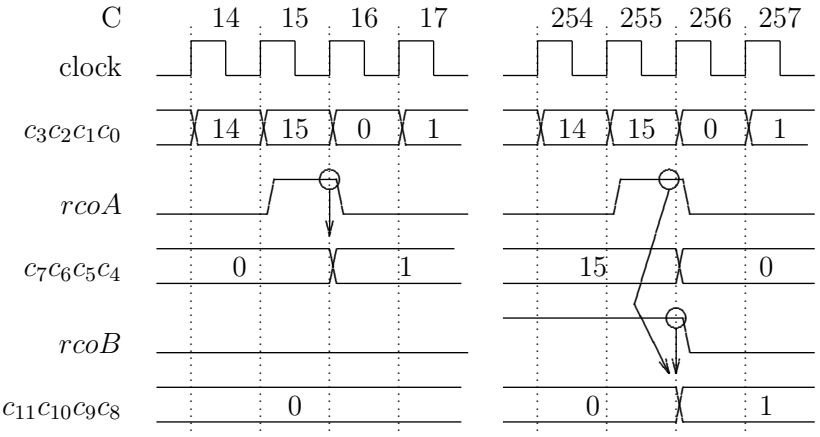
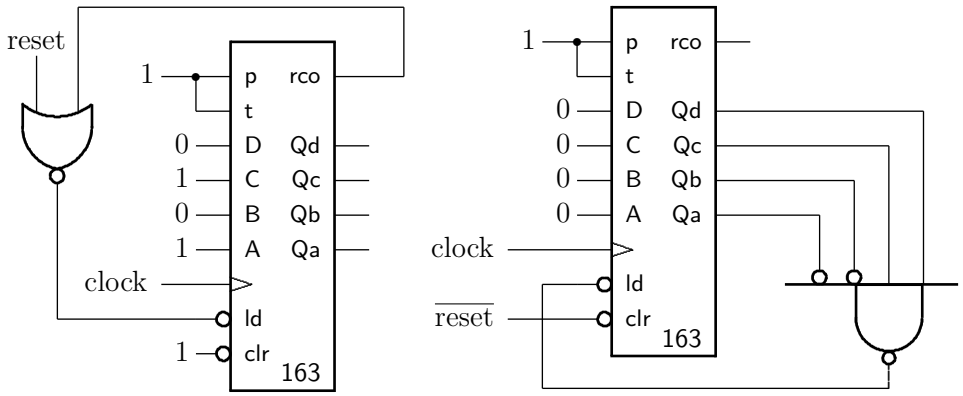


Figura 8.12: Diagrama de tempo do contador de 12 bits com 74163.

Exemplo 8.4 A Figura 8.13 mostra mais duas aplicações do 74163. O circuito (a) conta na sequência entre 5 e 15: quando a contagem chega em 15, o sinal *rco* faz com que a tupla que representa o número 5, ligada às entradas *D-A*, seja carregado sincronicamente. Nos ciclos subsequentes do relógio a contagem prossegue de 6 até 15. O sinal *reset* inicializa o contador em 5 para garantir que a primeira sequência seja correta. O circuito (b) é um contador módulo-13: toda vez que a contagem chegar em 12, a saída da porta *and* reinicializa a sequência sincronicamente. O desenho da porta foi ajustado para acomodar as quatro entradas, duas delas com inversões. ◁



(a) contagem entre 5 e 15; (b) contador módulo-13.

Figura 8.13: Aplicações do 74163.

8.2.6 Contador Assíncrono – *Ripple Counter*

Se observarmos o diagrama de tempos da Figura 8.6, veremos que o estado do *flip-flop* q_{n+1} se inverte quando há uma transição de 1 para 0 no *flip-flop* q_n. Podemos tirar proveito deste fato para simplificar o projeto do contador módulo-8, e para tanto necessitamos de *flip-flops* com as saídas q e $\overline{q}$. Tal circuito é mostrado na Figura 8.14, e emprega três *flip-flops* tipo D. A saída complementada é ligada à entrada de relógio do próximo *flip-flop* porque, quando há uma borda de descida em q_n, há uma borda de subida em $\overline{q_n}$.

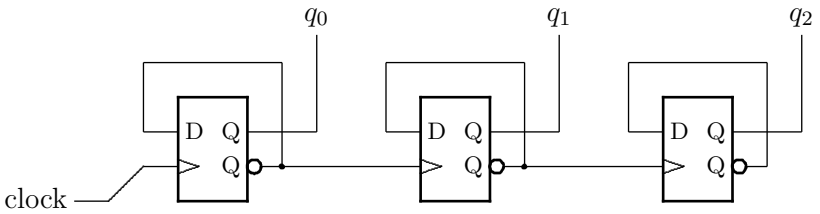


Figura 8.14: *Ripple counter* implementado com FFs tipo D.

O valor de contagem C para um contador módulo-8 é obtido do estado dos *flip-flops*, e é expresso na Equação 8.5. Nesta equação os valores dos bits q_0, q_1 e q_2 são interpretados como números naturais 0 ou 1. A Equação 8.5 indica que C não é exatamente a sequência desejada; as razões para tal são discutidas adiante.

$$\begin{array}{l} Q : \mathbb{B}^3 \\ C : \mathbb{N} \\ \mathit{ripple} : \mathbb{B}^3 \mapsto \mathbb{B}^3 \\ C = \mathit{num}(Q) \approx q_2 \cdot 2^2 + q_1 \cdot 2^1 + q_0 \cdot 2^0 \end{array} \tag{8.5}$$

Este contador é chamado de *ripple* porque os sinais de relógio dos vários *flip-flops* se propagam como uma onda, da saída $\overline{q}$ de um *flip-flop* para a entrada de relógio do próximo *flip-flop*.

As mudanças de estado nos *flip-flops* ocorrem na borda ascendente do sinal de relógio, e a sequência de contagem é mostrada na Figura 8.15. Suponha que inicialmente os três *flip-flops* estão com a saída $\langle q_2, q_1, q_0 \rangle = 000$ e portanto $C = 0$. Na primeira borda ascendente do sinal de relógio, o *flip-flop* q_0 troca de estado e $C = 1$ (001). Na primeira borda descendente em q_0 ($\overline{q_0}$ $\nearrow$) o *flip-flop* q_1 troca de estado e $C = 2$ (010). Quando a contagem atinge sete, os três *flip-flops* trocam de estado, e a sequência reinicia de zero. Esse o comportamento esperado de um contador módulo-8.

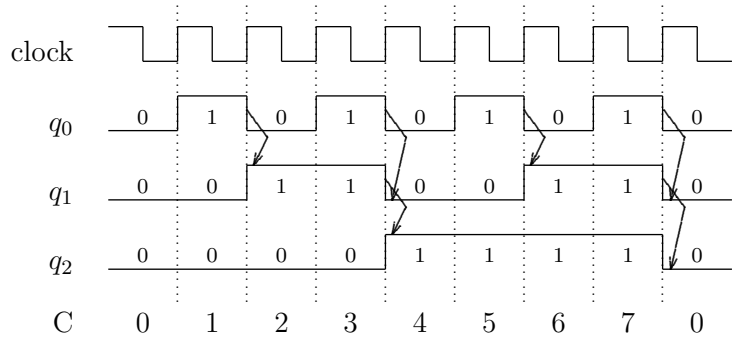


Figura 8.15: Comportamento idealizado do *ripple counter*.

Os sinais não se propagam instantaneamente através de *flip-flops*, e isso implica em que a sequência de contagem desse contador não é exatamente aquela mostrada na Figura 8.15. Dependendo do período do sinal de relógio, é mais provável que a sequência de estados se pareça com a mostrada na Figura 8.16. Neste diagrama, a escala de tempo é tal que o tempo de propagação dos *flip-flops* é próximo da duração de um ciclo do relógio.

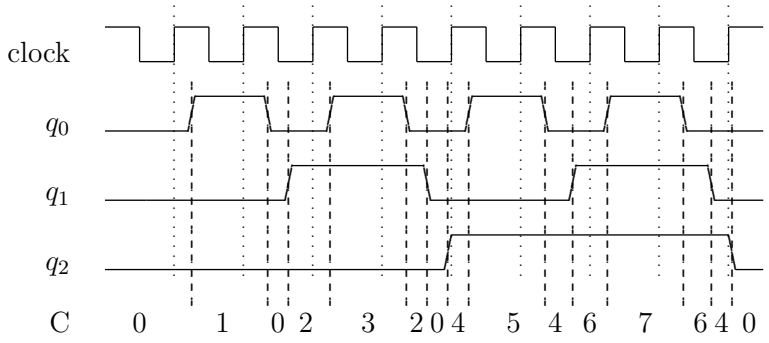


Figura 8.16: Sequência de contagem do *ripple counter*.

Esse contador é dito de *assíncrono* porque a saída de um *flip-flop* é usada como sinal de relógio do *flip-flop* seguinte, e o tempo de propagação dos sinais faz com que os *flip-flops* mudem de estado em instantes diferentes, e essas mudanças de estado se propaguem feito uma onda

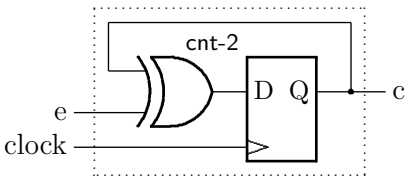
(*ripple*). Tal comportamento é altamente indesejável e frequentemente perigoso, dependendo da aplicação[2]. Existem técnicas de projeto que permitem tolerar a assincronia entre sinais, mas o projeto de circuitos assíncronos está fora do escopo deste texto.

É impossível garantir a perfeita sincronia nos eventos num circuito que não seja trivial por causa dos tempos de propagação dos diversos tipos de portas lógicas e da propagação de sinais através de fios com comprimentos diferentes. Contudo, as técnicas de projeto de circuitos síncronos reduzem os problemas potenciais causados pela assincronia porque tais técnicas garantem que todas as trocas de estado ocorrem durante janelas de tempo muito estreitos, e na vizinhança das transições no sinal de relógio. Se as condições definidas nas Seções 7.6.1 e 7.6.2 para *setup*, *hold* e *skew* forem respeitadas, então circuitos sequencias síncronos podem ser projetados com relativa facilidade por causa da abstração de tempo discretizado.

8.2.7 Contadores Síncronos

Os contadores vistos nas Seções 8.2.1 a 8.2.4 são ditos *síncronos* porque as transições em todos os *flip-flops* são disparadas por eventos simultâneos no sinal de relógio, que são as bordas ascendentes. Ao contrário do especificado na Equação 8.5, para esses contadores a igualdade é válida, e portanto o '$\approx$' deve ser substituído por um '$=$' quando se emprega a abstração para tempo discreto.

Exemplo 8.5 Vejamos como implementar um contador síncrono de 5 bits, projetado para operar na máxima frequência possível. Você dispõe de um contador módulo-2 (*cnt-2*) e das portas lógicas especificadas na Figura 8.17. Os tempos estão em nanosegundos, e são dados os tempos de propagação (T_{prop}), de contaminação (T_{cont}), de estabilização (T_{su}) e de manutenção (T_h). O circuito completo é mostrado na Figura 8.18. O bit q_{i+1} inverte quando $\langle q_i..q_0\rangle$ são todos 1, o que é garantido pela conjunção $q_i \wedge \ldots \wedge q_0$. No circuito da Figura 8.18, emprega-se a associatividade do *and* para computar a conjunção das saídas dos *cnt-2* menos significativos. ◁



	T_{prop}	T_{cont}	T_{su}	T_h
FF	6	2	4	2
xor-2	3	1		
and-2	3	1		
and-3	4	2		
and-4	5	3		

Figura 8.17: Especificação dos componentes do contador de 5 bits.

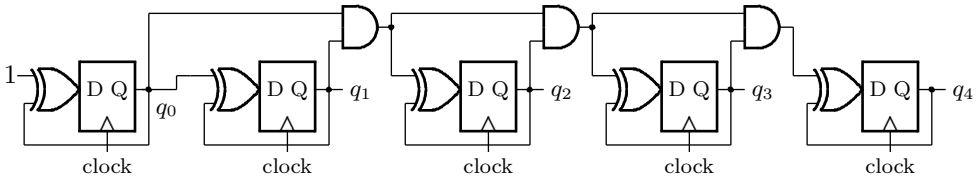


Figura 8.18: Primeira versão do contador de 5 bits.

[2]O Ministério dos Projetos adverte: circuitos assíncronos fazem mal à saúde e à nota.

Exemplo 8.6 Iniciemos pelo comportamento temporal do contador de 1 bit da Figura 8.17. O circuito combinacional é somente a porta *xor*; o período mínimo e a restrição de *hold* são:

$$\text{cnt-2}: \; T_{min} = T_{FF} + T_x + T_{su} = 6 + 3 + 4 = 13\,\text{ns},$$

$$\text{cnt-2}: \; T_{C,F} + T_{C,x} = 2 + 1 = 3 > 2 = T_h,$$

e portanto este circuito opera apropriadamente. ◁

Exemplo 8.7 Vejamos agora o comportamento temporal do contador de 5 bits da Figura 8.18. O caminho combinacional mais longo é aquele através das três portas *and* mais um *xor*, da saída q_0 até a entrada d_4. Esse caminho atravessa 3 portas *and* e o *xor* na entrada d_4.

$$\text{cnt-32 v1}: \; T_{min} = T_{FF} + 3T_a + T_x + T_{su} = 6 + 3 \times 3 + 3 + 4 = 22\,\text{ns}.$$

O tempo de manutenção de *cnt-32* é o mesmo de *cnt-2* porque o caminho de contaminação mais curto é o laço de realimentação $q_i \rightsquigarrow xor \rightsquigarrow d_i$. ◁

Exemplo 8.8 Se trocarmos a cadeia de *and*s, por portas com maior número de entradas, mas com tempo de propagação menor do que a cadeia de *and*s, o período mínimo pode ser reduzido. Este circuito é mostrado na Figura 8.19.

O período mínimo de relógio é determinado pelo caminho mais longo, e este é na entrada do *flip-flop* Q_4 por causa da porta *and*-4.

$$\text{cnt-32 v2}: \; T_{min} = T_{FF} + T_{a4} + T_x + T_{su} = 6 + 5 + 3 + 4 = 18\,\text{ns}.$$

A segunda versão é $22ns/18ns = 1,22$ vezes mais rápida do que a primeira. ◁

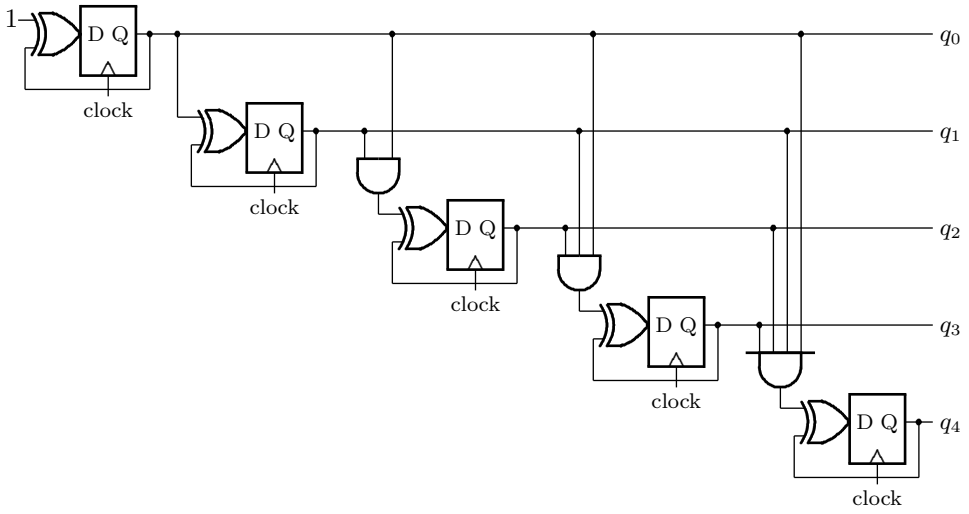


Figura 8.19: Segunda versão do contador de 5 bits.

Exercícios

Ex. 8.1 Qual seria a sequência de contagem do contador da Figura 8.14 se os FFs forem sensíveis à borda ascendente do relógio?

Ex. 8.2 O que se pode dizer quanto ao comportamento temporal de um contador *ripple* com 16 FFs? E quanto a um contador com 64 FFs?

Ex. 8.3 Re-compute o período mínimo de relógio para os circuitos dos Exemplos 8.5 e 8.8 considerando que o *skew* de pior caso é -3ns. Defina com cuidado o que seja o *skew de pior caso*. Qual dos dois circuitos é o mais rápido com *skew* negativo?

Ex. 8.4 O que se pode dizer quanto ao comportamento temporal de um contador síncrono com 16 FFs? Idem para 64 FFs?

Ex. 8.5 Especifique um contador de três bits que incrementa, ou decrementa, dependendo do estado da entrada Id: se $Id = 1$, a contagem incrementa (módulo-8); se $Id = 0$, a contagem decrementa (módulo-8).

Ex. 8.6 Implemente a especificação do Exercício 8.5, que é o circuito *inc-dec*.

Ex. 8.7 Estenda a especificação do Exercício 8.5 para 4 bits e a implemente.

8.2.8 Contador em Anel

Um *contador em anel* se comporta como um contador de N bits cujo conjunto de estados é menor que 2^N estados. Um contador em anel pode ser implementado como aquele mostrado na Figura 8.20. A saída de cada um dos FFs é ligada à entrada do FF seguinte, de forma tal que os valores iniciais dos FFs circulem pelo anel. A sequência de contagem depende do valor inicial dos FFs, e a contagem nunca se alteraria nos casos triviais com $\langle q_3 q_2 q_1 q_0 \rangle = \{0000, 1111\}$. O circuito da Figura 8.20 é inicializado em $\langle q_3 q_2 q_1 q_0 \rangle = 0001$.

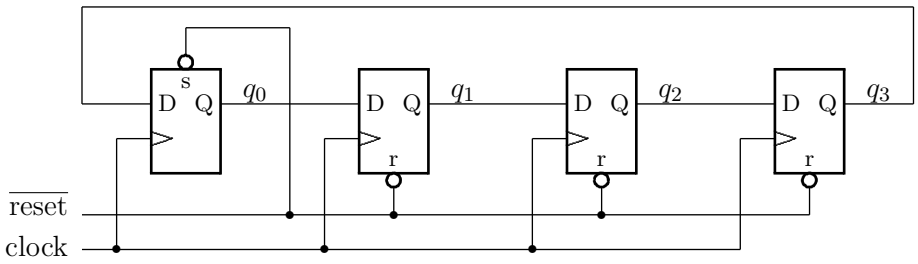


Figura 8.20: Contador em anel.

O contador em anel da Figura 8.20 tem N=4 bits e possui um conjunto de estados E com $n < 2^N$ estados, que são $E = \{0001, 0010, 0100, 1000\}$, e a sequência de contagem é $1, 2, 4, 8, 1, 2, 4, \cdots$, como mostra a Figura 8.21. No diagrama, os q_i são mostrados como ϕ_i, que significa *fase-i*, e cuja utilidade descobriremos em breve.

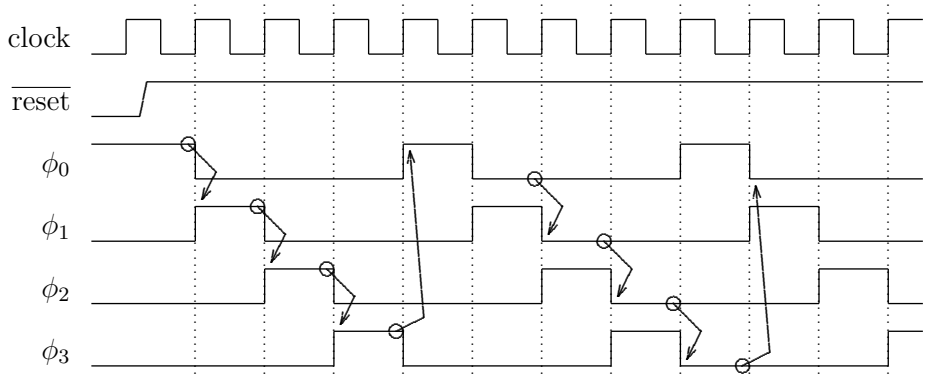


Figura 8.21: Comportamento do contador em anel.

8.2.9 Ciclo de Trabalho

A saída de um dos FFs do contador em anel produz um sinal periódico *assimétrico* porque os intervalos em que o sinal permanece em 1 e em 0 são diferentes. Por exemplo, o diagrama de tempo mostrado na Figura 8.21 mostra que os sinais ϕ_i permanecem em 1 durante 1/4 do período e em 0 por 3/4 do período. O *ciclo de trabalho* (*duty cycle*) de um sinal periódico é a relação entre os intervalos em que o sinal permanece em 1 e o período – tempo em 1 mais o tempo em 0. A Figura 8.22 mostra sinais com ciclos de trabalho de 25, 50 e 75%.

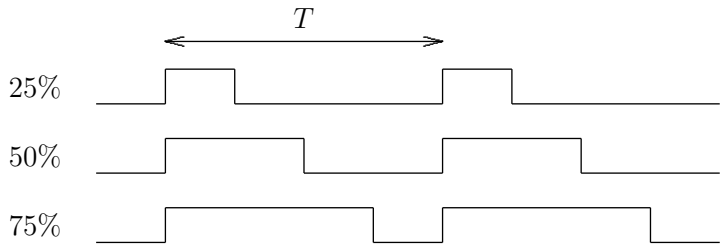


Figura 8.22: Ciclo de trabalho.

8.2.10 Relógio Multi-fase

Uma das aplicações para contadores em anel é a geração de sinal de *relógio multi-fase* de tal forma que cada um dos FFs gera uma dentre as quatro fases do sinal de relógio. Geralmente, as fases são chamadas de $\phi_0, \phi_1, \phi_2, \phi_3$, e a cada instante somente uma das quatro fases está ativa, como mostra a Figura 8.21.

Suponha que seja necessário reduzir o custo de um circuito, e para tal são empregados básculos simples, ao invés de *flip-flops* mestre-escravo. A Figura 8.23 mostra um circuito com três blocos combinacionais separados por quatro registradores compostos por básculos simples. Esta implementação é problemática porque, quando o sinal de relógio estiver em 1, o sinal d_1 contamina todo o circuito pois os quatro básculos ficam transparentes.

Se somente um par de saídas do contador em anel for usado para gerar os sinais de relógio dos básculos, *viz.* $\langle \phi_0, \phi_2 \rangle$, o comportamento do circuito é aquele mostrado na Figura 8.24.

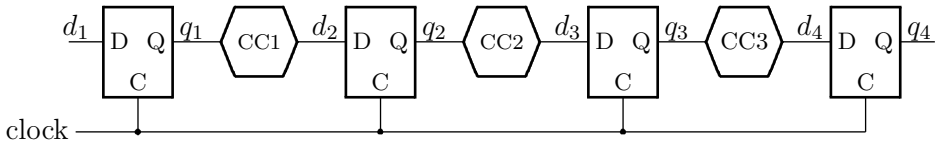


Figura 8.23: Básculos simples com contaminação de todas as entradas.

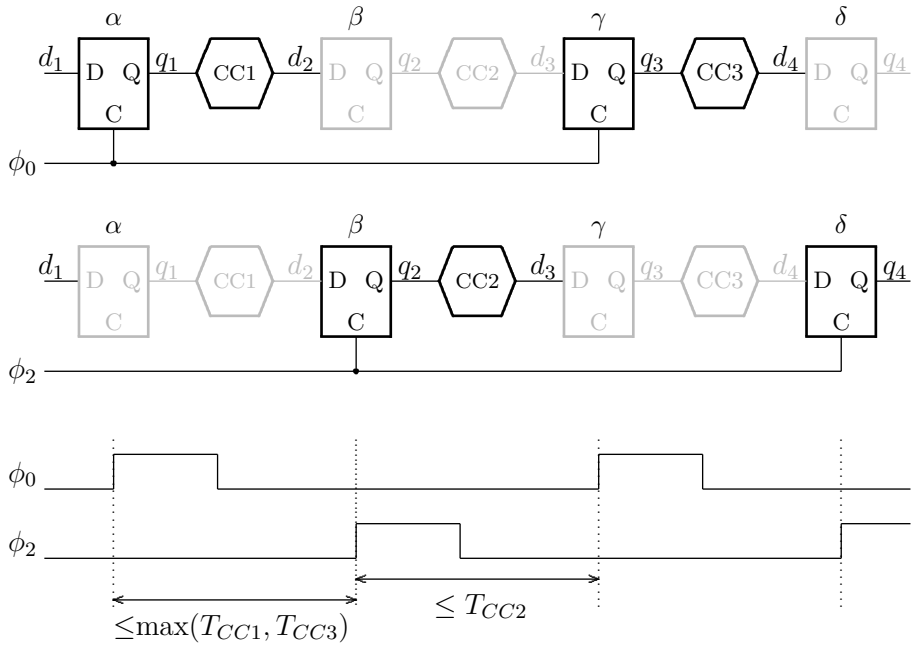


Figura 8.24: Relógio com duas fases elimina a contaminação.

Quando ϕ_0 está ativo, os básculos α e γ ficam transparentes, enquanto que os básculos β e δ mantém seus valores q_2 e q_4 inalterados. Quando ϕ_2 está ativo, a situação se inverte e os básculos β e δ ficam transparentes, enquanto que α e γ mantém seus valores. Os básculos que mantém seus valores são mostrados em cinza porque estes, por estar inativos, impedem a contaminação de suas saídas pelos sinais dos básculos adjacentes ligados as suas entradas.

A Figura 8.24 indica que os intervalos entre ϕ_0 e ϕ_2 ativos devem ser mais longos que os tempos de propagação dos circuitos combinacionais, mais a propagação através dos básculos fonte e o *setup* dos básculos destino. No diagrama estão indicados somente os tempos de propagação dos circuitos combinacionais.

Se o tempo de propagação dos circuitos combinacionais entre os básculos for mais longo do que o intervalo nos quais os sinais de relógio ficam inativos, – ϕ_1 a ϕ_3 na Figura 8.21 – estes intervalos inativos podem ser elongados. Para tanto basta acrescentar *flip-flops* ao contador em anel para afastar as bordas entre as fases ativas dos sinais de relógio.

Relógios multi-fase são usados em circuitos sequenciais implementados com básculos simples para garantir que seu estado seja alterado somente após os sinais nas suas entradas estabilizarem. Normalmente, uma fase é usada para a alteração do estado dos básculos (ϕ_i), e a fase seguinte (ϕ_{i+1}) é usada para garantir que os sinais se propaguem através da parte combinacional do circuito e estabilize imediatamente antes da próxima fase de alteração de estado (ϕ_{i+2}).

8.2.11 Contador Johnson

Um *contador Johnson* é similar a um contador em anel exceto que a saída do último FF é complementada. Um contador Johnson de N bits possui um conjunto de estados E cujo tamanho é $|\,\mathrm{E}\,| = 2N$ estados. A Figura 8.25 mostra o diagrama de tempos de um contador Johnson com 4 FFs.

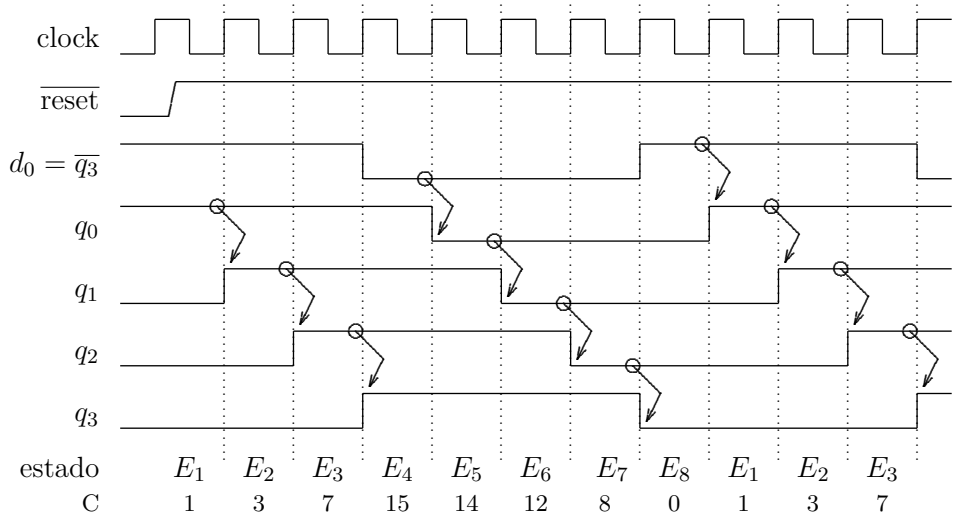


Figura 8.25: Comportamento do contador Johnson.

Note que na sequência de contagem de um contador Johnson apenas uma das saídas se altera a cada ciclo, o que torna esta sequência de estados um código de Gray. Veja o Exercício 8.20 para outra implementação de um contador Gray.

Espaço em branco proposital.

8.3 Registradores de Deslocamento

Um *registrador de deslocamento* permite deslocar seu conteúdo de tal forma que, a cada borda do relógio, todos os bits armazenados nos *flip-flops* deslocam-se simultaneamente de uma posição. Os registradores de deslocamento básicos são de dois tipos: carga em série e leitura em paralelo, ou carga em paralelo e leitura em série.

8.3.1 Registrador de Deslocamento Série-Paralelo

A Figura 8.26 mostra um *registrador de deslocamento série-paralelo* que é carregado com um bit através da entrada d_3 a cada ciclo do relógio. As saídas $\langle q_3, q_2, q_1, q_0 \rangle$ de todos os *flip-flops* podem ser observadas ao mesmo tempo.

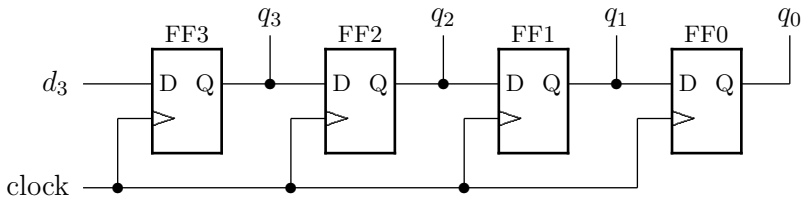


Figura 8.26: Registrador de deslocamento série-paralelo.

Na borda ascendente do relógio a entrada de cada *flip-flop* é copiada para sua saída, deslocando assim o conteúdo do registrador de uma posição. A Tabela 8.7 mostra uma sequência de estados deste registrador. Inicialmente, os *flip-flops* $\langle q_3, q_2, q_1, q_0 \rangle$ contêm os bits $\langle x, y, z, w \rangle$, e à entrada d_3 são apresentados os bits $\langle a, b, c, d \rangle$, um a cada ciclo. Após 4 ciclos, os *flip-flops* estão no estado $\langle q_3, q_2, q_1, q_0 \rangle = \langle d, c, b, a \rangle$.

Tabela 8.7: Sequência de deslocamento de estados.

estado	d_3	q_3	q_2	q_1	q_0
0	–	x	y	z	w
1	a	a	x	y	z
2	b	b	a	x	y
3	c	c	b	a	x
4	d	d	c	b	a

O registrador da Figura 8.26 é chamado de *conversor série-paralelo* porque este circuito converte entradas que são amostradas em série – um bit a cada ciclo – em saída que é apresentada em paralelo – todos os bits ao mesmo tempo.

O diagrama da Figura 8.26 identifica cada um dos *flip-flops*, de FF3 a FF0. Esta convenção é usada ao longo do texto, e para simplificar os diagramas, o nome do *flip-flop* é omitido, mas o seu número/nome é o mesmo do índice dos sinais ligados à saída Q. Assim, a saída do FF3 é chamada de q_3.

8.3.2 Registrador de Deslocamento Paralelo-Série

Todos os *flip-flops* de um *registrador de deslocamento paralelo-série* podem ser carregados simultaneamente, para que então seus conteúdos sejam deslocados, um bit a cada ciclo de relógio. Os quatro *flip-flops* do circuito da Figura 8.27 são carregados em paralelo se o sinal *carga=1* numa borda ascendente do relógio. Do contrário, *carga=0*, os bits armazenados nos *flip-flops* são deslocados a cada borda do sinal de relógio.

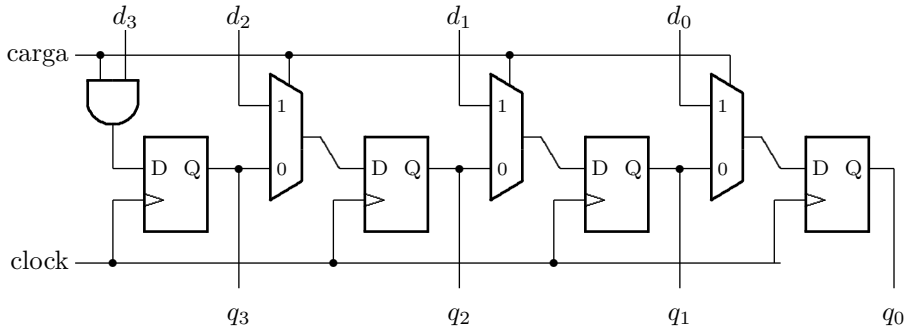


Figura 8.27: Registrador de deslocamento paralelo-série.

Um registrador que permite a carga dos valores nas entradas em paralelo, e apresente na saída um bit de cada vez, é chamado de *conversor paralelo-série*.

Exemplo 8.9 Considere o registrador de deslocamento da Figura 8.27, sendo as entradas carregadas com $\langle d_3, d_2, d_1, d_0\rangle = \langle 1, 0, 1, 0\rangle$. O diagrama de tempos da Figura 8.28 mostra a sequência de estados nas saídas do registrador.

No ciclo α, os quatro *flip-flops* são carregados com os valores em $d_{3..0}$, e no ciclo β os deslocamentos se iniciam. Por causa da porta *and*, a partir do ciclo β, $d_3 = 0$ e este 0 se propaga ao longo do registrador, um bit a cada borda do relógio. No ciclo δ, o bit inicialmente carregado no *flip-flop* mais à esquerda é transferido para q_0, e a partir de então, a saída q_0 é sempre 0. ◁

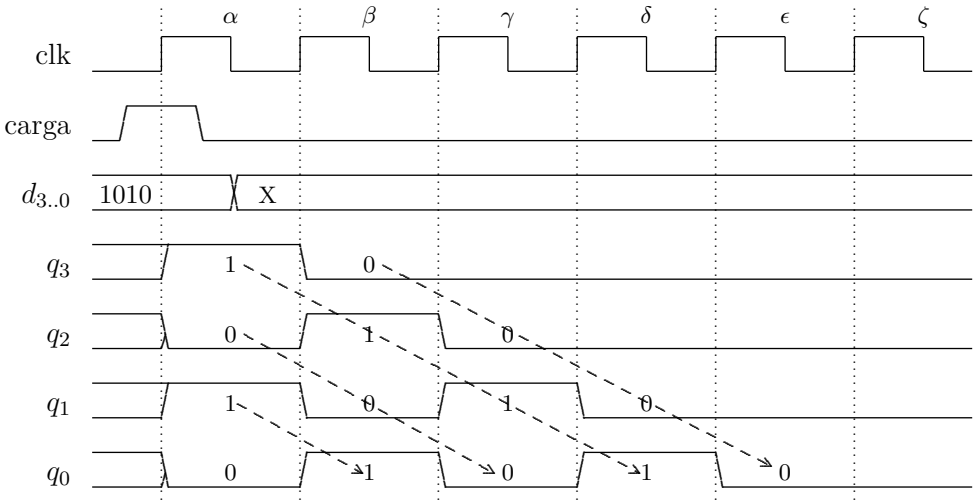


Figura 8.28: Diagrama de tempos do registrador paralelo-série.

A Figura 8.29 mostra os símbolos dos dois tipos de conversores, um registrador série-paralelo e um registrador paralelo-série.

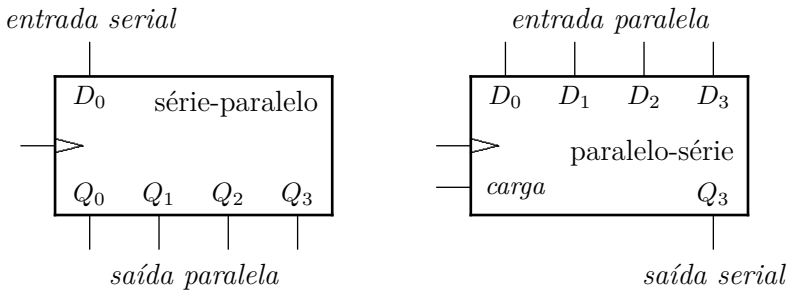


Figura 8.29: Conversores série-paralelo e paralelo-série.

Exemplo 8.10 Vejamos uma primeira versão, assaz primitiva, de um circuito de transmissão serial. Este circuito recebe uma sequência de quatro bits, os repassa sem alteração da entrada para a saída, e então insere um quinto bit, este com a paridade do quarteto recém enviado.

O valor do bit de "paridade par" para 4 bits deve ser tal que o número de bits em 1, considerando os 4 bits de dados mais o bit de paridade, seja par. Para os bits `0000`, o bit de paridade par é `0`. Para `1111`, o bit de paridade também é `0`. Para `0001`, o bit de paridade é `1`, pois o bit de dados em `1` mais o bit de paridade em `1`, garantem que o número de bits em `1` seja par.

A solução emprega um *flip-flop* T para computar a paridade, e é mostrada na Figura 8.30. A cada bit transmitido, a paridade é computada no *flip-flop* T. No intervalo correspondente ao quinto bit (sel=1), o valor armazenado no *flip-flop* T é a paridade dos quatro bits anteriores. Quando se inicia um novo quarteto, o *flip-flop* T deve ser reinicializado em 0, fazendo-se *novo=1*.

O diagrama de tempo da operação do circuito é mostrado na Figura 8.31. As quatro entradas são copiadas da entrada para a saída, e então o bit de paridade π é inserido na sequência. ◁

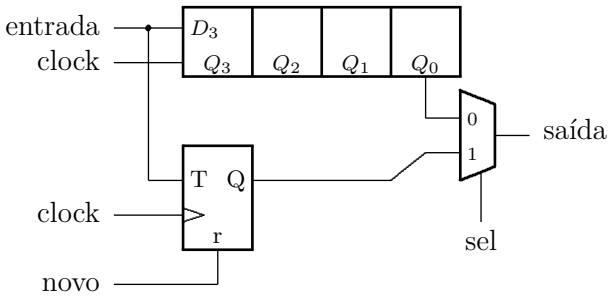


Figura 8.30: Paridade 'sequencial' para 4 bits.

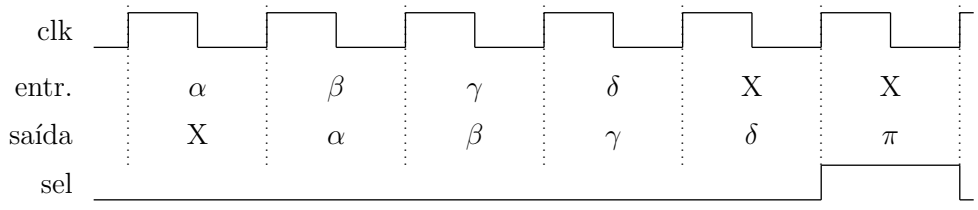


Figura 8.31: Temporização do circuito da paridade para 4 bits.

8.3.3 Registrador de Deslocamento Universal

Um *registrador de deslocamento universal* é um registrador de deslocamento que efetua as duas formas de conversão (série-paralelo e paralelo-série), e permite deslocamentos para a esquerda e para a direita, carga paralela, e manutenção dos bits armazenados. O comportamento deste registrador é especificado na Tabela 8.8, e a Figura 8.32 mostra uma parte do seu circuito.

Tabela 8.8: Especificação do registrador de deslocamento universal.

	$s_1 s_0$	d_3	d_2	d_1	d_0	q_3	q_2	q_1	q_0
carga	1 1	a	b	c	d	a	b	c	d
$\gg 1$	1 0	v	X	X	X	v	q_3	q_2	q_1
$\ll 1$	0 1	X	X	X	v	q_2	q_1	q_0	v
mantém	0 0	X	X	X	X	q_3	q_2	q_1	q_0

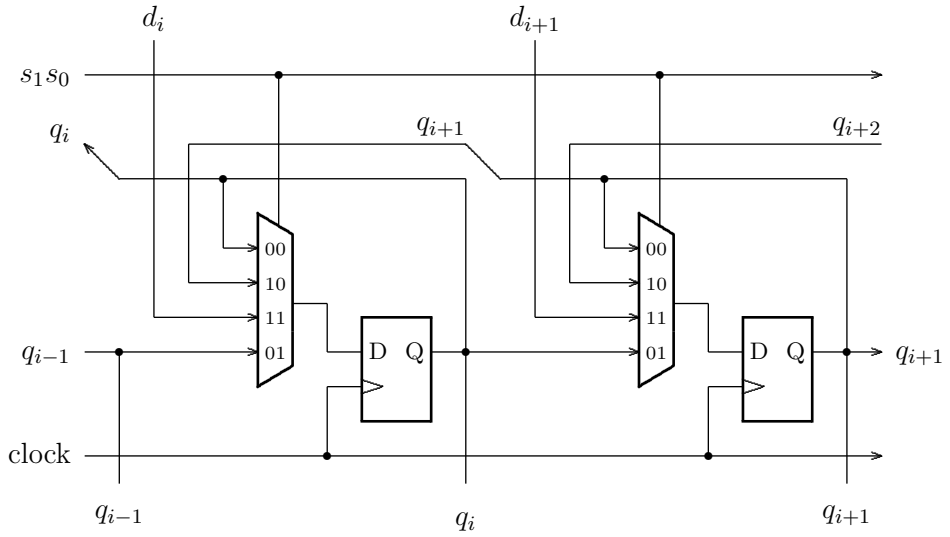


Figura 8.32: Registrador de deslocamento universal.

8.3.4 Somador Serial

Existem aplicações nas quais consumo de energia reduzido é mais importante do que velocidade de execução. Nestas aplicações pode ser conveniente empregar circuitos que operam serialmente, ao invés de paralelamente. Por exemplo, um somador serial para números inteiros representados em n bits pode ser implementado com três registradores de deslocamento para conter os operandos e o resultado, e um somador completo para efetuar a soma de um par de bits a cada ciclo do relógio, como mostrado na Figura 8.33. O circuito opera de forma similar ao cálculo manual da soma de dois números com muitos dígitos: efetua-se a soma da direita para a esquerda, considerando-se um par de dígitos mais o vem-um do par anterior, e lembrando do vai-um para o próximo par de dígitos. Este algoritmo está formalizado na Equação 8.6, na qual são computados os dois bits $\langle c_i s_i \rangle$ que resultam da soma dos dois operandos (a_i, b_i) e do vem-um (c_{i-1}).

$$\begin{aligned} c_{-1} &= 0 \\ c_i s_i &= a_i + b_i + c_{i-1} \end{aligned} \tag{8.6}$$

O bloco que efetua a soma dos três bits contém um somador completo e um *flip-flop*, para armazenar o valor do vai-um para a soma do próximo par de bits.

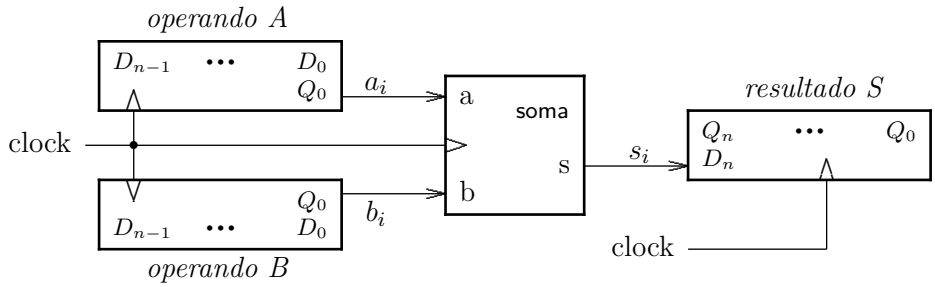


Figura 8.33: Somador serial.

Exercícios

Ex. 8.8 Complemente o diagrama de blocos da Figura 8.33, e explicite como devem ser inicializados o registradores com os operandos, o de resultado, e o *flip-flop* do vai-um.

Ex. 8.9 Tomando o circuito mostrado na Figura 8.18 como base, projete um contador de 32 bits. Quantas portas lógicas *and* são ligadas em série num contador com N bits?

Ex. 8.10 Escreva uma expressão para o limite de velocidade do circuito do Ex. 8.9 como uma função do número de bits do contador.

Ex. 8.11 Usando uma técnica similar a do somador com adiantamento de vai-um (*carry look-ahead*), modifique o contador de 32 bits do Ex. 8.10 para melhorar seu desempenho e reescreva a equação do limite de velocidade para refletir a modificação. Qual o ganho em desempenho? *Pista*: veja a Seção 6.6.

Ex. 8.12 O fabricante do registrador 74374, com 8 FFs tipo D, informa que o tempo de propagação médio é de 20ns, e o máximo de 30ns, o *setup time* é no mínimo 20ns, e o *hold time* de 3ns. Suponha que este registrador é usado em um circuito no qual o tempo de propagação da parte combinacional é de 130ns no caminho crítico. Qual a frequência máxima do relógio para este circuito?

Ex. 8.13 Mostre como um contador em anel (Seção 8.2.8) pode ser usado para reduzir o impacto na frequência do relógio de um circuito com um componente combinacional cujo tempo de propagação é muito mais longo do que qualquer outra parte do circuito.

Ex. 8.14 Adapte a Equação 7.3 para uso num circuito com relógio de duas fases, gerado por um contador em anel, como aquele mostrado na Figura 8.21.

8.4 Máquinas de Estados Finitas

Um *algoritmo* pode ser definido como uma sequência de operações que são aplicadas sobre um conjunto de dados para produzir um determinado resultado. No nosso contexto, uma das possíveis implementações de um algoritmo consiste de: (i) um certo número de circuitos combinacionais que efetuem as operações requeridas; (ii) registradores (ou memória) para manter os resultados intermediários; e (iii) um circuito de controle para gerenciar o fluxo de dados através dos circuitos combinacionais e registradores. Veremos adiante que o circuito de controle é geralmente implementado como uma *máquina de estados finita*.

As seções anteriores tratam de contadores e registradores de deslocamento, que são exemplos relativamente simples de máquinas de estados finitas. Esta seção formaliza a definição destas máquinas e discute técnicas de projeto que se aplicam a uma classe mais ampla de circuitos sequenciais.

Formalmente, uma *máquina de estados finita* M é definida pela sêxtupla da Equação 8.7. Q é um conjunto finito de estados, q_0 é o estado inicial pós-*reset*, I é um conjunto finito de símbolos de entrada, f é a função de transição ou *função de próximo estado*, S é um conjunto finito de símbolos de saída, e g é a função de saída.

$$\begin{array}{ll}
Q : \mathcal{Q} & \text{conjunto de estados, de tipo } \mathcal{Q} \\
I : \mathcal{I} & \text{alfabeto de entrada, de tipo } \mathcal{I} \\
S : \mathcal{S} & \text{alfabeto de saída, de tipo } \mathcal{S} \\
q_0 \in Q & \text{estado inicial} \\
f : (\mathcal{Q} \times \mathcal{I}) \mapsto \mathcal{Q} & \text{função de próximo estado} \\
g : (\mathcal{Q} \times \mathcal{I}) \mapsto \mathcal{S} & \text{função de saída}
\end{array} \tag{8.7}$$

$$M = (Q, I, f, q_0, g, S)$$

A entrada para uma máquina de estados finita é uma sequência de símbolos em I e quando ocorre uma mudança de estado o símbolo associado àquela transição é consumido da sequência de entradas.

Exemplo 8.11 A especificação do contador módulo-4 pode ser formalizada pela máquina de estados definida na Equação 8.8.

$$\begin{aligned}
Q &= \{00, 01, 10, 11\} \\
I &= \emptyset \\
f &= \{(00, 01), (01, 10), (10, 11), (11, 00)\} \\
q_0 &= 00 \\
g &= \text{identidade} \\
S &= Q
\end{aligned} \tag{8.8}$$

$$M_{\text{mod}-4} = (Q, \emptyset, f, 00, =, Q)$$

O conjunto de quatro estados é representado em binário, o conjunto de entrada é vazio porque as transições ocorrem independentemente de qualquer sinal externo, a função de próximo estado é definida como um conjunto de pares ordenados, o estado inicial é o estado 00, a função de saída é a identidade porque a saída do contador é o próprio estado, e o conjunto de saída é o conjunto de estados. ◁

8.4.1 Diagrama de Estados

Usamos *diagramas de estado* para representar os estados de uma máquina de estado (ME), as transições entre os estados em função das entradas, e as saídas produzidas pela ME.

Um estado é representado por um círculo, e as transições de estado por setas, e a cada seta é associada à condição que possibilita a mudança de estado. O nome do estado é indicado na parte superior do círculo, e a saída produzida no estado é indicada na parte inferior. O estado inicial é indicado por uma seta que não provém de nenhum outro estado, e este é o estado para o qual a máquina vai quando o sinal de *reset* é ativado. A Figura 8.34 mostra estas convenções.

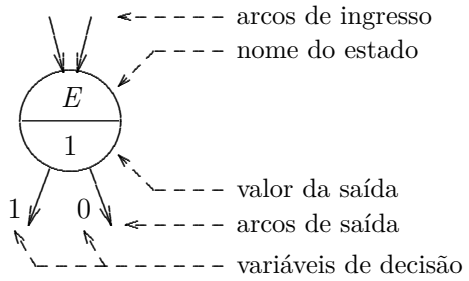


Figura 8.34: Convenção para desenhar diagramas de estados.

Exemplo 8.12 O comportamento de um contador módulo-4 pode ser descrito pela sequência de estados que é percorrida ao longo do tempo:

$$\text{reset} \to 0 \to 1 \to 2 \to 3 \to 0 \to 1 \cdots .$$

Esse comportamento é representado por um diagrama de estados, como o mostrado na Figura 8.35. No diagrama, não são indicadas as saídas porque o estado dos *flip-flops* é a própria saída. As condições para as transições de estado não estão marcadas porque num contador como este *sempre* ocorre uma mudança de estado a cada ciclo do relógio. ◁

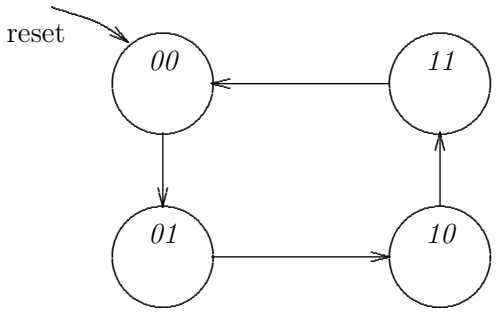


Figura 8.35: Diagrama de estados de um contador módulo-4.

Exemplo 8.13 Considere uma máquina de estados que produz um pulso na sua saída toda vez que sua entrada amostrar a sequência 01110. Dito de outra forma, esta máquina é capaz de reconhecer a sequência 01110. No modelo da Figura 7.28, a entrada e a saída têm largura de um bit ($\lambda = 1, \sigma = 1$).

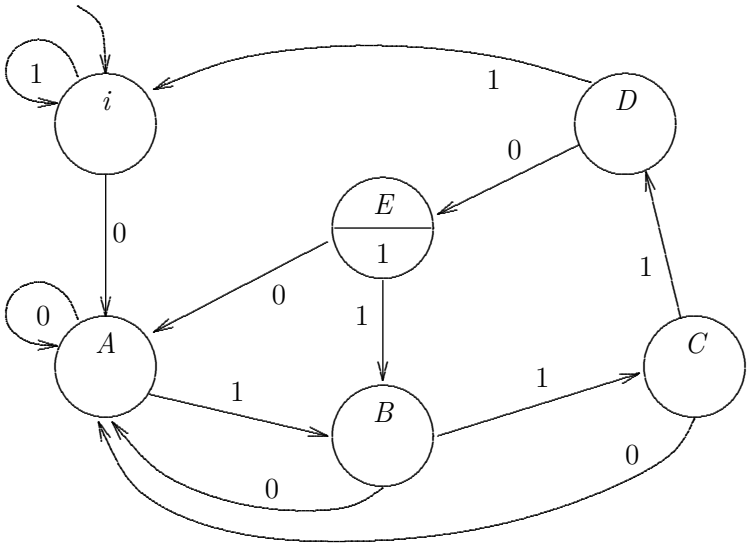


Figura 8.36: Diagrama de estados do reconhecedor de 01110.

As mudanças de estado ocorrem a cada ciclo do relógio, caso a sequência de entradas assim o permita. Como há somente uma entrada binária, os estados apresentam duas possíveis transições, uma para entrada 0 e outra para entrada 1.

Uma decisão de projeto pode determinar que, para a sequência 011101110, a máquina produza dois pulsos em sua saída. A Figura 8.36 mostra o diagrama de estados desta ME.

Após a inicialização, enquanto a entrada for 1, a máquina permanece no estado i. A sequência de estados que produz um pulso, com a duração de um período do relógio, é $i\,A\,B\,C\,D\,E$ para entradas 01110. Para entradas 011101110, a sequência de estados é $i\,A\,B\,C\,D\,E\,B\,C\,D\,E$. Para simplificar o diagrama, somente no estado E a saída é indicada como 1, e subentende-se que os demais produzem saída 0. ◁

Exemplo 8.14 O comportamento da ME que detecta a sequência 01110, do Exemplo 8.13, é formalizado na Equação 8.9.

$$\begin{aligned} Q &= \{i, A, B, C, D, E\} \\ I &= \{0, 1\} \\ f &= \{(i,0,i), (i,1,A), (A,0,A), (A,1,B), \\ &\quad\ (B,0,A), \ldots, (E,0,A), (E,1,B)\} \\ q_0 &= i \\ g &= \{(i,0), (A,0), \ldots, (D,0), (E,1)\} \\ S &= I \end{aligned} \tag{8.9}$$

$$M_{01110} = (Q, I, f, i, g, I)$$

A função de próximo estado é definida por triplas ⟨estado atual, entrada, próximo estado⟩ porque, em todos os estados, a decisão de qual é o próximo estado depende do valor da entrada naquele estado. ◁

Dos dois exemplos de diagramas de estado discutidos até agora, o contador não depende de nenhuma entrada externa porque as transições ocorrem a cada ciclo do relógio, e o reconhecedor da sequência 01110 depende de uma única entrada binária. Em geral, para uma máquina de estado com n entradas, em cada estado podem ocorrer até 2^n transições distintas para próximo(s) estado(s).

No projeto de MEs há três restrições importantes que não podem ser ignoradas:

1. os diagramas de estado devem permitir a implementação de máquinas de estado *determinísticas* e portanto não pode haver mais de um próximo estado para cada um dos possíveis valores das entradas;
2. dentre todas as transições para outros estados, *ao menos uma, e somente uma*, deve ser possível; e
3. todos os estados devem ser o destino de ao menos uma transição.

Para garantir que todos os estados sejam alcançáveis, alguma transição para cada um dos estados deve ser possível, para ao menos uma dentre todas as combinações possíveis das entradas e estados.

Os diagramas de estados são uma representação para *máquinas de estado finitas*, que por sua vez pertencem ao conjunto dos *autômatos finitos*. Estes últimos são estudados na disciplina Teoria da Computação [Koh78, HU79].

Exercícios

Ex. 8.15 Desenhe os diagramas de estados completos de um contador em anel (Seção 8.2.8) e de um contador Johnson (Seção 8.2.11), ambos com 4 FFs.

Ex. 8.16 Formalize o projeto do contador em anel da Seção 8.2.8. Especifique a função de próximo estado através de uma tabela com os campos (i) *estado atual* e (ii) *próximo estado*.

Ex. 8.17 Especifique formalmente o contador Johnson da Seção 8.2.11. Especifique a função de próximo estado através de uma tabela.

Ex. 8.18 Especifique a função de próximo estado do detector de sequências da Figura 8.36 através de uma tabela com três colunas, (i) *estado atual*, (ii) *entrada*, e (iii) *próximo estado*.

Ex. 8.19 Modifique o diagrama de estados do detector de sequências de Figura 8.36 para evitar que a sequência 011101110 produza saída no terceiro 0. O circuito deve reconhecer somente a primeira sequência em **01110**1110 e ignorar a segunda. A sequência 0111001110 produz dois pulsos na saída.

Ex. 8.20 Projete e implemente um contador Gray, de 3 bits cuja sequência de contagem é aquela definida no Exercício 4.4.

Ex. 8.21 Projete um circuito sequencial síncrono que produz em sua saída a sequência mostrada abaixo. Seu projeto deve empregar quatro FFs tipo D.
`0000` $\mapsto$ `0011` $\mapsto$ `0110` $\mapsto$ `1001` $\mapsto$ `1100` $\mapsto$ `1111` $\mapsto$ `0000`, e repete.

8.4.2 Máquinas de Mealy e Máquinas de Moore

Dependendo da função de saída, podem ser definidos dois tipos de máquinas de estado finitas, as Máquinas de Moore e as Máquinas de Mealy.

Máquina de Moore

Numa Máquina de Moore a saída depende apenas do estado atual, como definido na Equação 8.10.

$$g : \mathcal{Q} \mapsto \mathcal{S} \qquad \text{Máquina de Moore} \tag{8.10}$$

A saída produzida por uma Máquina de Moore, em resposta a uma sequência de entrada $e_1, e_2 \cdots e_n,\ n \geqslant 0$ é

$$g(q_0), g(q_1) \cdots g(q_n)$$

quando $q_0, q_1 \cdots q_n$ é a sequência de estados tal que $f(q_{i-1}, e_i) = q_i\ ,\ 1 \leqslant i \leqslant n$.

A Figura 8.37 mostra um circuito genérico que pode ser usado para implementar Máquinas de Moore. Estas máquinas consistem de um *registrador de estado*, de uma *função de próximo estado*, e de uma *função de saída*. O próximo estado (PE) da máquina depende do estado atual (EA) e das entradas (E): $PE = f(E, EA)$.

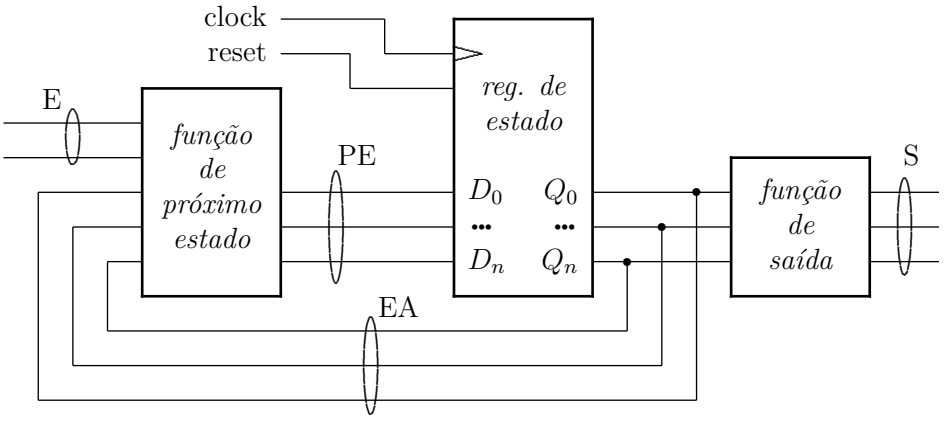


Figura 8.37: Máquina de Moore.

Espaço em branco proposital.

Exemplo 8.15 Vejamos um exemplo de uma *Máquina de Moore* que detecta sequências de dois ou mais **1**s em sua entrada. O diagrama de estados da ME é mostrado no lado esquerdo da Figura 8.38, e um diagrama de tempo para a sequência de entrada **0010111** é mostrado à direita.

O sinal *reset* coloca a ME no estado A. Enquanto a entrada for **0**, a ME permanece no estado A. Na recepção do primeiro **1** a ME muda para o estado B, e retorna para A na recepção do terceiro **0**. No segundo **1** a máquina muda para o estado B, e na recepção do terceiro **1** a máquina vai para o estado C, quando então a saída é ativada porque uma sequência de dois **1**s foi detectada.

Note que a mudança de estado ocorre *depois* da borda do relógio. Evidentemente, os parâmetros de temporização da entrada devem ser atendidos. ◁

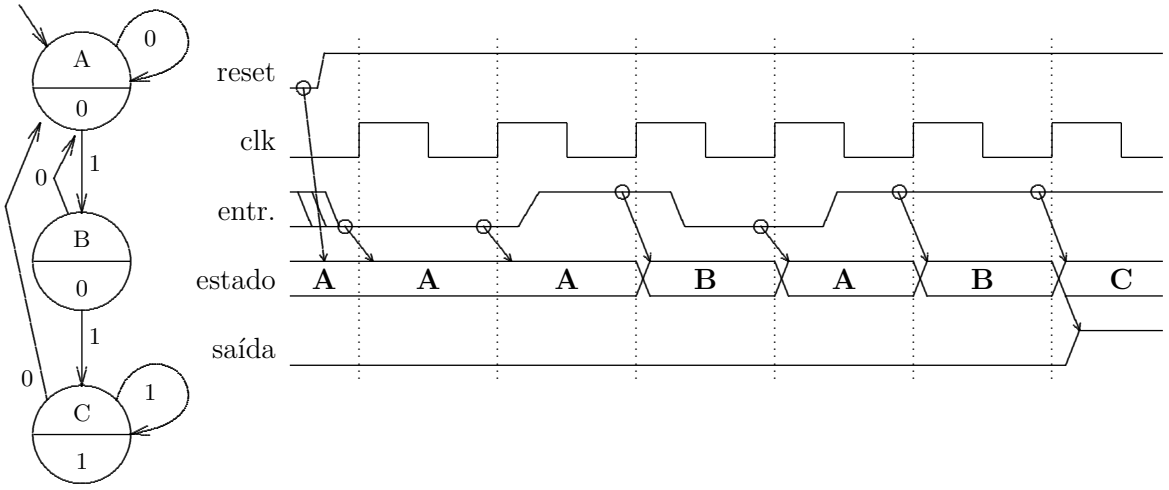


Figura 8.38: Máquina de Moore que reconhece a sequência 011^+.

Máquina de Mealy

Numa máquina de Mealy a saída depende do estado atual e das entradas, como definido na Equação 8.11.

$$g : (\mathcal{Q} \times \mathcal{I}) \mapsto \mathcal{S} \qquad \text{Máquina de Mealy} \tag{8.11}$$

Em resposta a uma sequência de entrada $e_1, e_2 \cdots e_n,\ n \geq 0$, a saída produzida por uma Máquina de Mealy é

$$g(q_0, e_1), g(q_1, e_2) \cdots g(q_{n-1}, e_n)$$

quando $q_0, q_1 \cdots q_n$ é a sequência de estados tal que $f(q_{i-1}, e_i) = q_i\,,\ 1 \leqslant i \leqslant n$. Note que esta sequência de estados tem um estado a menos que a sequência correspondente produzida por uma Máquina de Moore.

Numa Máquina de Mealy, como aquela mostrada Figura 8.39, as saídas dependem do estado atual e das entradas: $S = g(E, EA)$.

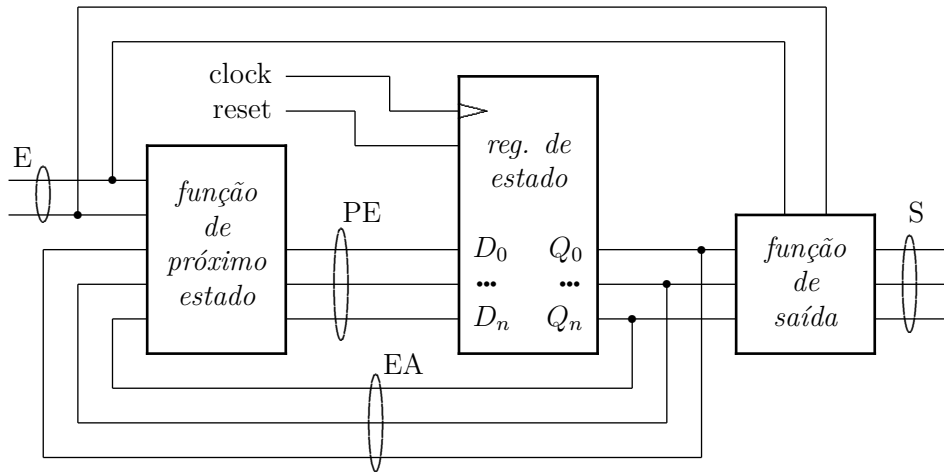


Figura 8.39: Máquina de Mealy.

Exemplo 8.16 Vejamos a *Máquina de Mealy* que detecta sequências de dois ou mais **1**s em sua entrada. O diagrama de estados da ME é mostrado no lado esquerdo da Figura 8.40, e um diagrama de tempo para a sequência de entrada **0010111** é mostrado à direita.

O sinal *reset* coloca a ME no estado A. Enquanto a entrada for **0**, a ME permanece no estado A. Na recepção do primeiro **1** a ME muda para o estado B, e retorna para A na recepção do terceiro **0**.

No segundo **1** a máquina muda para o estado B, e a saída é ativada porque uma sequência de dois **1**s foi detectada. Enquanto a ME estiver no estado B, a saída segue a entrada, e nesse exemplo ocorre um pulso espúrio na saída quando o primeiro **1** é amostrado. Contudo, imediatamente antes da borda do relógio, a saída é **0**, como desejado.

Esta é uma diferença importante entre os dois modelos de ME: numa Máquina de Mealy, a saída geralmente é "mais suja", ou mais ruidosa, do que numa Máquina de Moore. ◁

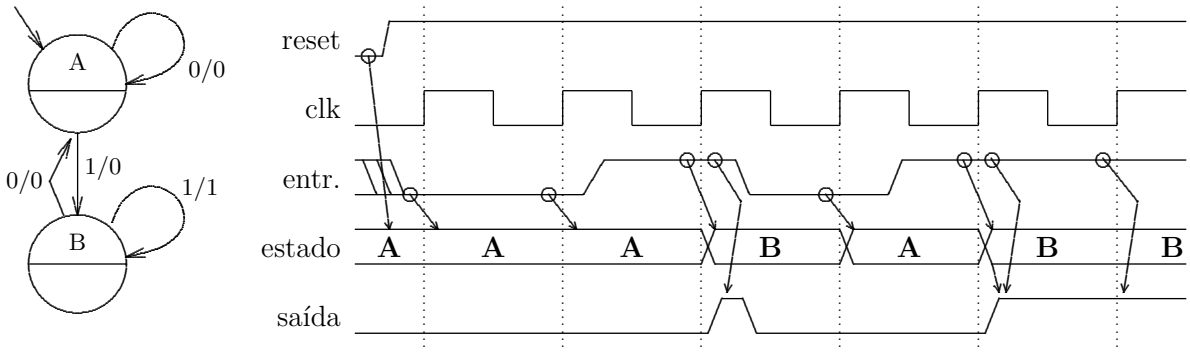


Figura 8.40: Máquina de Mealy que reconhece a sequência 011^+.

Espaço em branco proposital.

8.4.3 Implementação de Máquinas de Estado

A metodologia de projeto de máquinas de estado compreende seis passos:

1. compreensão e formalização do problema;
2. atribuição de estados;
3. cálculo da função de próximo estado;
4. cálculo da função de saída;
5. implementação; e
6. projeto dos testes.

Exemplo 8.17 Implementemos a ME do Exemplo 8.13, que reconhece a sequência 01110.

O passo de compreensão e formalização está descrito nos Exemplos 8.13 e 8.14.

São seis os estados e portanto são necessários $\lceil \log_2 6 \rceil = 3$ FFs. A atribuição dos estados pode ser, iniciando-se do estado de *reset*: $i \mapsto 0$, $A \mapsto 1$, $B \mapsto 2$, $C \mapsto 3$, $D \mapsto 4$, $E \mapsto 7$. A atribuição de 7 ao estado E simplifica a função de saída.

A função de próximo estado é mostrada na Tabela 8.9. Para os dois estados sem atribuição, estados 5 e 6, a escolha tenta ser segura: se a ME acidentalmente entrar num destes estados, no próximo ciclo a ME volta ao estado inicial.

A função de saída é a conjunção das saídas dos FFs porque $num(\langle q_2, q_1, q_0 \rangle) = 7$.

Para implementar as funções de próximo estado pode-se empregar três multiplexadores, como mostra a Figura 8.41. Como temos uma entrada e três FFs, são necessárias quatro variáveis de controle para selecionar o próximo estado. As funções de próximo estado foram simplificadas – conforme a coluna '8' da Tabela 8.9 – para a implementação com três mux-8 ao invés da implementação direta com três mux-16.

Tabela 8.9: Tabela de PE da máquina que reconhece a sequência 01110.

EA	*8*	$q_2q_1q_0$	e	PE	$d_2d_1d_0$
i	*0*	0 0 0	0	A	0 0 1
		0 0 0	1	i	0 0 0
A	*1*	0 0 1	0	A	0 0 1
		0 0 1	1	B	0 1 0
B	*2*	0 1 0	0	A	0 0 1
		0 1 0	1	C	0 1 1
C	*3*	0 1 1	0	A	0 0 1
		0 1 1	1	D	1 0 0
D	*4*	1 0 0	0	E	1 1 1
		1 0 0	1	i	0 0 0
ϕ	*5*	1 0 1	X	i	0 0 0
ψ	*6*	1 1 0	X	i	0 0 0
E	*7*	1 1 1	0	A	0 0 1
		1 1 1	1	B	0 1 0

Quanto aos testes, obviamente deve-se empregar sequências adequadas para garantir que a saída é aquela esperada quando a sequência 01110 é observada na entrada. A sequência 011101110 também deve fazer parte do conjunto de vetores de teste. Sequências de 0s (indicadas como 0^+) e de 1s (1^+) devem ser testadas, assim como $\{0,1\}^+01110\{0,1\}^+$. ◁

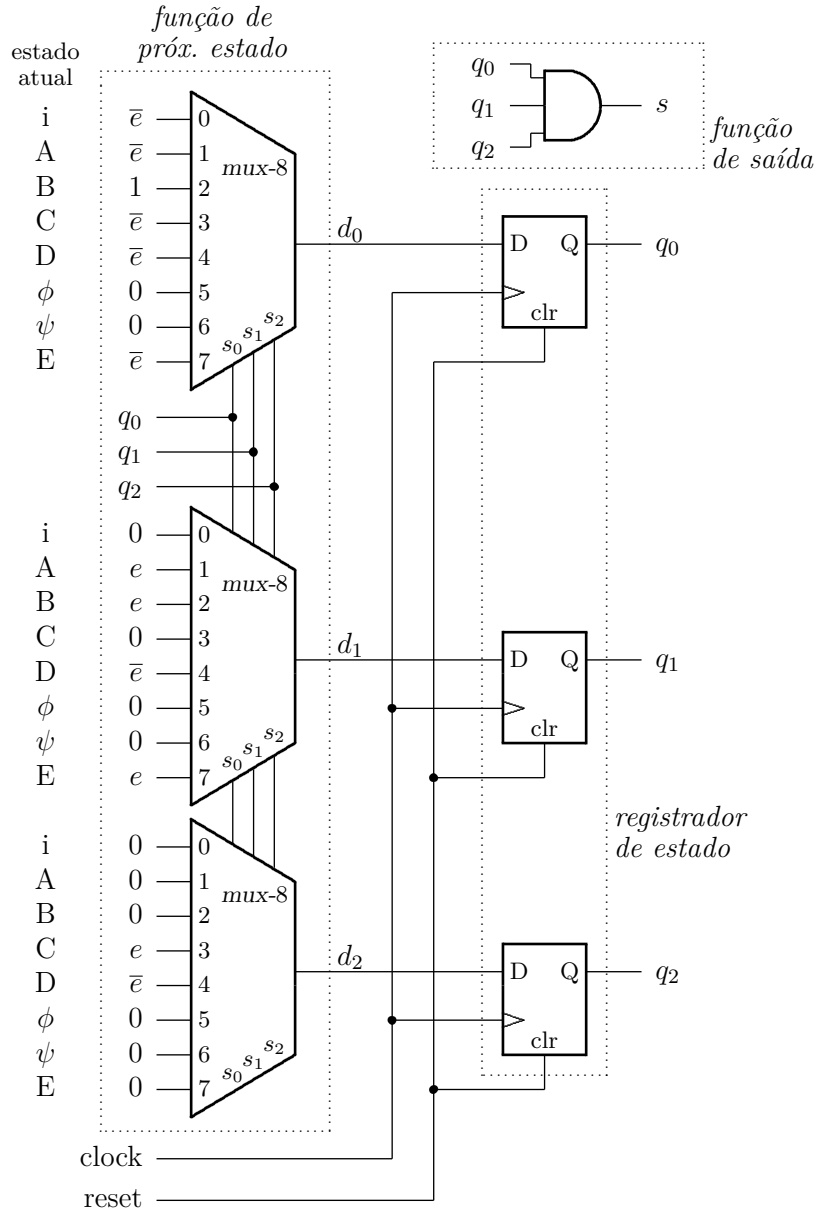


Figura 8.41: Implementação da ME que reconhece a sequência 01110.

Exemplo 8.18 Considere o projeto de um contador módulo-10: para implementar este contador são necessários, no mínimo, quatro *flip-flops*: $\lceil \log_2 10 \rceil = 4$. A atribuição de estados mais simples é associar a cada estado de um contador binário o número que lhe corresponde. A função de próximo estado deve garantir que a sequência normal de contagem seja obedecida, e ainda que quando o contador atinja o estado correspondente ao número 9 a sequência reinicie de 0. A função de saída é trivial porque a própria atribuição de estados a definiu implicitamente. Quanto aos testes, a sequência de

contagem deve ser observada ao longo de, pelo menos, dois ciclos completos ($0 \mapsto 9, 0 \mapsto 9$). ◁

8.4.4 Máquina de Vender Chocolates

Suponha que o controlador de uma máquina de vender chocolates deva ser implementado com uma máquina de estados. A máquina possui uma entrada para a inserção de moedas, uma saída para a devolução de moedas, e uma saída para entregar os chocolates vendidos. Para simplificar o problema, suponha que a máquina não devolve troco – se as moedas recebidas excedem o preço do chocolate, todas as moedas são devolvidas.

Cada chocolate custa $\$1,00$, e a máquina deve aceitar apenas moedas de $\$0,50$ e de $\$0,25$. O detector de moedas é muito bem construído e aceita e informa o valor de uma moeda de cada vez, e é capaz de rejeitar moedas de valores fora da faixa aceitável – moedas rejeitadas são desviadas automaticamente para a saída de moedas. O controlador digital possui duas entradas correspondentes a moedas de $\$0,50$ e de $\$0,25$, e duas saídas, choc que aciona o dispensador de chocolates, e dev que aciona a devolução de moedas. A Figura 8.42 mostra um diagrama de blocos com os componentes da máquina de vender chocolates.

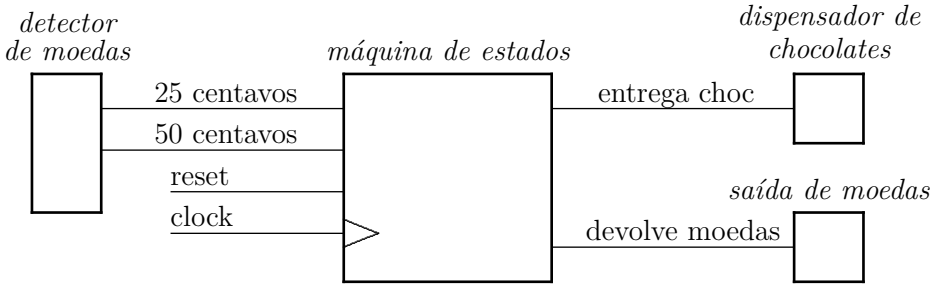


Figura 8.42: Diagrama de blocos da máquina de vender chocolates.

Formalização do problema A Figura 8.43 contém o diagrama de estados com a especificação do comportamento da máquina de vender chocolates. Os nomes dos estados correspondem ao valor acumulado até aquele estado. Se as moedas inseridas ultrapassam o valor de um chocolate, no estado *erro* todas as moedas são devolvidas. O cliente recebe seu chocolate após inserir $1,00.

Atribuição de estados O diagrama de estados contém seis estados, sendo portanto necessários $\lceil \log_2 6 \rceil = 3$ FFs para implementá-lo. Uma das possíveis atribuições de estados é mostrada na Tabela 8.10.

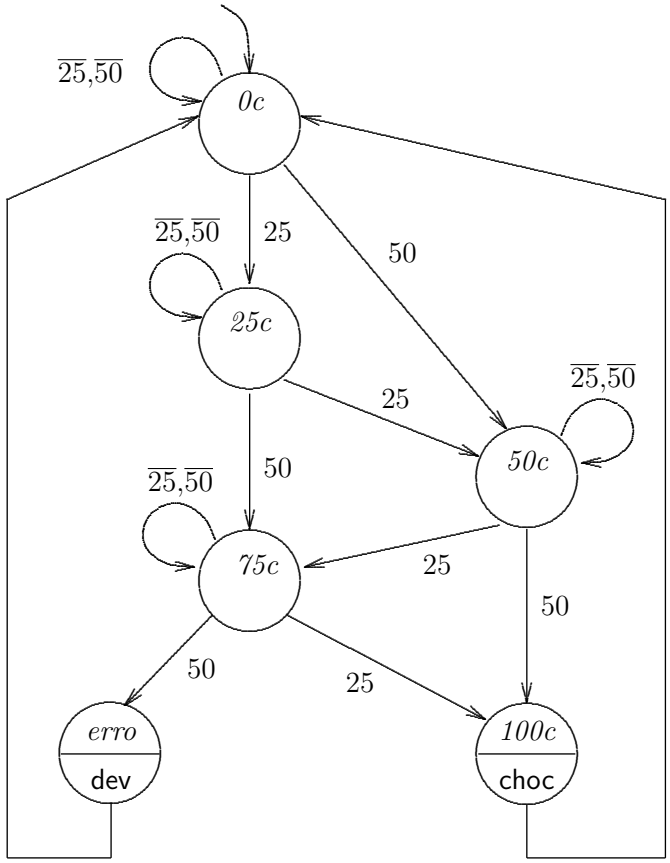


Figura 8.43: Especificação da máquina de vender chocolates.

Tabela 8.10: Atribuição de estados de máquina de vender chocolates.

estado	$q_2 q_1 q_0$
$0c$	0 0 0
$25c$	0 0 1
$50c$	0 1 0
$75c$	0 1 1
$100c$	1 0 0
erro	1 1 1

Função de próximo estado Tomando por base a atribuição de estados da Tabela 8.10 e a especificação do comportamento do controlador da Figura 8.43, a função de próximo estado pode ser projetada, e a tabela de próximo estado é mostrada na Tabela 8.11. Para reduzir o tamanho da tabela, as colunas marcadas com X representam o par de valores $\{0, 1\}$ porque as saídas dos estados *devem* estar definidas para as quatro possibilidades das entradas: $\langle 0, 0\rangle$, $\langle 0, 1\rangle$, $\langle 1, 0\rangle$ e $\langle 1, 1\rangle$.

Duas decisões de projeto estão implícitas na tabela de próximo estado. A primeira, sinalizada por '†', indica uma preferência por moedas de $\$0,25$ – se existem duas moedas disponíveis, o controlador consume a moeda de $\$0,25$. A segunda decisão de projeto, sinalizada por '‡', define o comportamento do controlador se estados inválidos são atingidos pelo controlador – as moedas são todas devolvidas e a máquina fica a esperar por uma nova transação.

Tabela 8.11: Tabela de próximo estado de máquina de vender chocolates.

	EA	entr	PE	
estado	$q_2q_1q_0$	25 50	$d_2d_1d_0$	
0c	0 0 0	0 0	0 0 0	
		0 1	0 1 0	
		1 0	0 0 1	†
		1 1	0 0 1	†
25c	0 0 1	0 0	0 0 1	
		0 1	0 1 1	
		1 0	0 1 0	†
		1 1	0 1 0	†
50c	0 1 0	0 0	0 1 0	
		0 1	1 0 0	
		1 0	0 1 1	†
		1 1	0 1 1	†
75c	0 1 1	0 0	0 1 1	
		0 1	1 1 1	
		1 0	1 0 0	†
		1 1	1 0 0	†
100*c*	1 0 0	X X	0 0 0	
–	1 0 1	X X	1 1 1	‡
–	1 1 0	X X	1 1 1	‡
erro	1 1 1	X X	0 0 0	

Função de saída A função de saída para uma Máquina de Moore é mostrada na Tabela 8.12. A saída choc é ativada no estado *100c* para entregar um chocolate ao cliente, e a saída dev é ativada no estado *erro* para devolver as moedas em caso de erro ou excesso de moedas.

Espaço em branco proposital.

Tabela 8.12: Função de saída de máquina de vender chocolates.

estado	EA	saída
$100c$	100	choc
$erro$	111	dev

Implementação A função de próximo estado mostrada na Tabela 8.11 deve ser desmembrada em três funções, uma para cada um dos FFs (d_2, d_1, d_0). Cada função tem 5 variáveis – três FFs de estado (q_2, q_1, q_0) e duas entradas (25,50). As saídas dependem da mera decodificação dos estados: $\mathsf{choc} = q_2 \wedge \overline{q_1} \wedge \overline{q_0}$, $\mathsf{dev} = q_2 \wedge q_1 \wedge q_0$.

Exercícios

Ex. 8.22 Desenhe outro diagrama de estados para a máquina de vender chocolates (MdVC) considerando que a implementação será com uma Máquina de Mealy.

Ex. 8.23 Estenda o projeto da MdVC adicionando um botão que permita ao comprador abortar a operação a qualquer instante, causando a devolução das moedas.

Ex. 8.24 Estenda o projeto da MdVC adicionando um sinal para que o controlador solicite uma nova moeda ao detector de moedas.

8.5 Microcontroladores

Vejamos uma técnica mais simples e eficiente para o projeto de máquinas de estados finitas. Esta técnica se baseia na implementação das funções de próximo estado e de saída com memórias ROM. Considere a Máquina de Moore da Figura 8.44. A função de próximo estado e a função de saída podem ser implementadas como uma memória ROM indexada pelos bits de estado atual (EA) e pelas entradas (E). Algumas das saídas da ROM contém a função de próximo estado (PE) e outras a função de saída (S). A codificação das tabelas de próximo estado e de saída são gravadas diretamente na ROM, de tal forma que o conteúdo da n-ésima linha da tabela é gravado na n-ésima posição da ROM.

A utilidade desta técnica fica óbvia quando se considera o projeto de máquinas de estado complexas. Considere uma ME com 18 estados e 4 entradas; para codificar os 18 estados são necessários $\lceil log_2 18 \rceil = 5$ *flip-flops*. A tabela de próximo estado possui $2^4 \times 2^5 = 2^9 = 512$ linhas. A simplificação das funções de PE dos 5 *flip-flops* é uma tarefa tediosa e que exige paciência e obstinação digna de orientais.

Se a tabela de próximo estado for copiada direta e simplesmente para uma memória ROM, com 512 linhas, o esforço de projeto é dramaticamente reduzido, diminui a possibilidade de introdução de erros, bem como o esforço para verificar a corretude da ME resultante. A geração do conteúdo da ROM pode ser facilmente automatizada, o que elimina quase todo o trabalho tedioso do projeto de MEs.

Na Figura 8.44, as entradas são sincronizadas no registrador de estado. Isso é necessário se as entradas são geradas por circuitos que operam de forma assíncrona ao relógio da ME. Se as

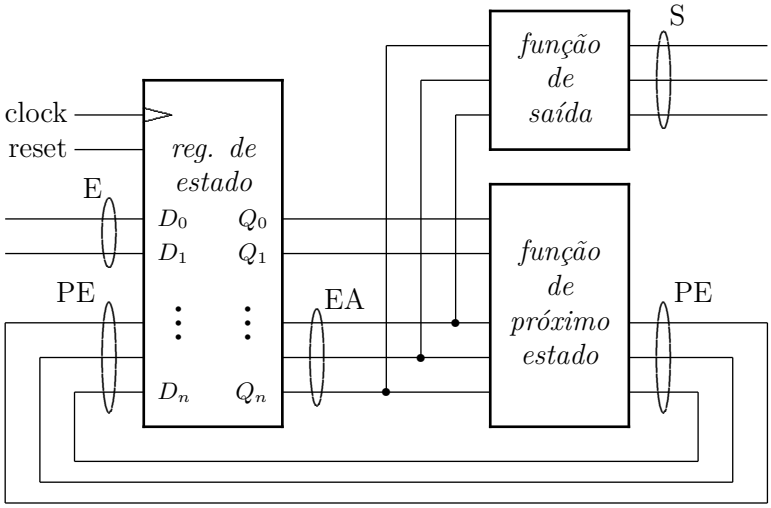


Figura 8.44: Máquina de Moore.

entradas forem geradas externamente, mas de forma a garantir os requisitos de *setup* e *hold* do registrador de estado, então não é necessário sincronizar as entradas.

Algumas máquinas de estado complexas são chamadas de *microcontroladores*. O nome *microcontrolador*, como usado aqui, refere-se à técnica de implementação de máquinas de estado com dezenas ou centenas de estados e com fluxos de controle não triviais. Tais máquinas de estado são empregadas para controlar sistemas digitais complexos, tipicamente com milhares de portas lógicas e centenas de registradores.

Exemplo 8.19 Considere uma ME que detecta uma sequência de dois ou mais 1s, seguidos de 0. Um diagrama de blocos dessa ME é mostrado na Figura 8.45, com uma entrada e e uma saída s, que fica em 1 somente quando a sequência é observada na entrada e. A figura também mostra o diagrama de estados do detector de sequência.

A Tabela 8.13 contém a codificação do diagrama de estados bem como a codificação da ROM que controla a ME. O lado esquerdo da tabela com a codificação da ROM mostra os bits de endereço da ROM, que são os dois bits do registrador de estado $\langle q_1 q_0 \rangle$ mais o bit da entrada e. Note que o sinal de entrada e é ligado ao endereço menos significativo da ROM.

O lado direito da tabela de codificação mostra o conteúdo de cada linha da ROM, com os dois bits de próximo estado $\langle d_1 d_0 \rangle$ e o bit de saída s. ◁

Exemplo 8.20 Considere um microcontrolador que pode seguir uma de várias sequências distintas. Uma entrada de 3 bits de largura determina qual das 8 possíveis sequências será seguida. Suponha que cada sequência possui no máximo 32 microinstruções. Ao final de cada sequência, o estado atual é reinicializado com o endereço da primeira microinstrução. O controlador possui 4 saídas.

A Figura 8.46 mostra um diagrama de blocos do controlador. As 3 entradas do controlador são ligadas, através do registrador, aos 3 bits mais significativos do endereço da memória ROM, $\langle e_7, e_6, e_5 \rangle$. Note que, ao contrário do circuito do Exemplo 8.19, as entradas ligadas aos bits mais significativos dividem a ROM em oito faixas de endereços, e cada uma delas corresponde a uma sequência de $2^5 = 32$ microinstruções.

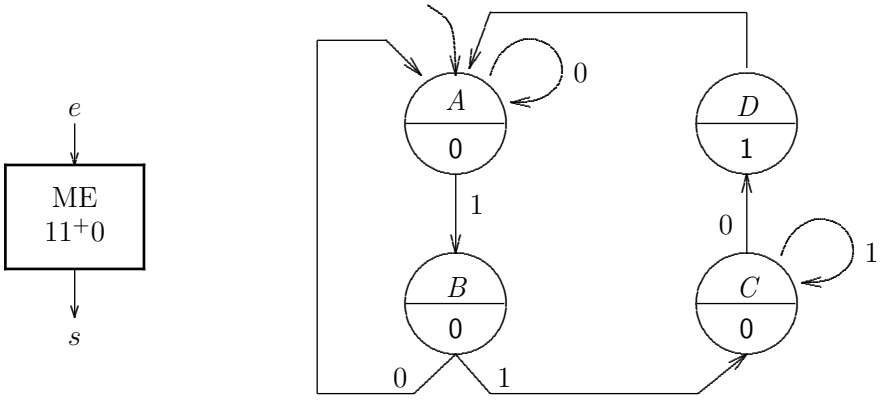


Figura 8.45: Máquina de estados que detecta a sequência $11^{+}0$.

Tabela 8.13: Codificação da ROM da ME que detecta a sequência $11^{+}0$.

Tabela de próximo estado

EA	q_1q_0	e	PE	d_1d_0	s
A	00	0	A	00	0
		1	B	01	
B	01	0	A	00	0
		1	C	10	
C	10	0	D	11	0
		1	C	10	
D	11	0	A	00	1
		1	A	00	

Codificação da ROM

$\langle$EA,$e\rangle$	$\langle$PE,s$\rangle$
$q_1q_0\,e$	$d_1d_0\,s$
0 0 0	0 0 0
0 0 1	0 1 0
0 1 0	0 0 0
0 1 1	1 0 0
1 0 0	1 1 0
1 0 1	1 0 0
1 1 0	0 0 1
1 1 1	0 0 1

Os cinco bits menos significativos do endereço da ROM $\langle e_4 \ldots e_0\rangle$ são ligados ao registrador de estado, e o conteúdo de cada posição da ROM aponta o próximo endereço na sequência de microinstruções. Este projeto não usa um contador para percorrer as sequências, mas o próprio conteúdo da ROM determina qual a próxima microinstrução será executada. Cada palavra da ROM é dividida em 2 campos, o primeiro com os bits d_0 a d_4 aponta para a próxima microinstrução $\langle p_4, p_3, p_2, p_1, p_0\rangle$, e o segundo campo, com os bits d_5 a d_8, com as quatro saídas $\langle s_0, s_1, s_2, s_3\rangle$. ◁

Exemplo 8.21 Suponha que você foi contratada para projetar a ME que controla uma bomba de (posto de) gasolina. A bomba fornece diesel, gasolina, etanol e querosene. O processo de venda para os quatro combustíveis é o mesmo, o que muda é o preço, que é definido pelo volume de líquido para cada Real de pagamento.

Uma possibilidade para o projeto da ME seria usar o mesmo sensor de volume, e fazer a multiplicação do preço com uma micro-rotina diferente para cada combustível. Os quatro combustíveis são coisas parecidas, mas não iguais. Portanto, sua ME emprega uma rotina (similar a uma função em Pascal) diferente para cada combustível.

Na implementação da tabela de próximo estado, cada micro-rotina corresponde a uma faixa de endereços, que contém as operações – sequência de sinais de controle – na ordem certa para cada rotina (diesel, etanol, etc.). Para simplificar o projeto, as faixas de endereço das microrotinas são do mesmo tamanho (2^n), mesmo que alguns endereços não sejam usados, e assim desperdiçados. Em geral, uns

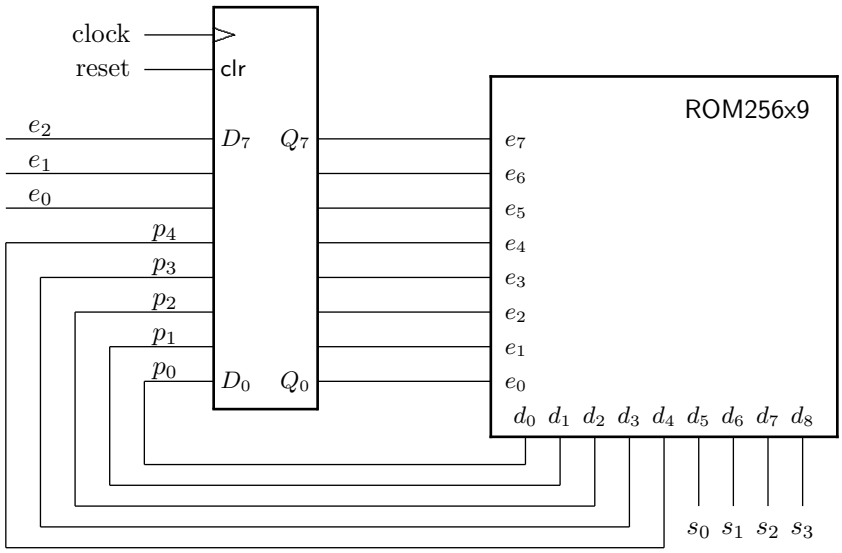


Figura 8.46: Microcontrolador com três entradas.

tantos bits de memória são muito menos custosos do que o tempo de projeto, implementação e testes.

Necessariamente, um sinal de dois bits identifica qual dos quatro combustíveis está sendo vendido, e este número é usado para escolher a microrotina apropriada. ◁

A época de glória dos microcontroladores foi nas décadas de 1960-80, quando eram usados como unidades de controle dos computadores de médio e grande porte da Digital (VAX11-780) e IBM (360, 370), por exemplo [BJ97]. Unidades de controle com microcontroladores também foram empregadas em microprocessadores das décadas de 1980-90, como a família 68000 da Motorola e 80x86 da Intel, e são usadas correntemente em microprocessadores da AMD e da Intel para a interpretação das instruções complexas da arquitetura 80x86 [HP12]. Microcontroladores são também usados em multiprocessadores, para controlar as transações de memória, que é fisicamente distribuída na máquina [JNW08]. Em geral, empregam-se microcontroladores em sistemas cuja velocidade de operação é tal que as funções de controle não podem ser desempenhadas por um microprocessador executando um programa de controle.

8.5.1 Máquina de Vender Chocolates – Versão 2

Considere uma segunda implementação do circuito de controle da máquina de vender chocolates. A função de próximo estado é repetida na Tabela 8.14, com as quatro combinações possíveis das entradas 25 e 50.

Espaço em branco proposital.

Tabela 8.14: Função de próximo estado da máquina de vender chocolates.

	EA	entr	PE
estado	$q_2q_1q_0$	25 50	$d_2d_1d_0$
0c	0 0 0	0 0	0 0 0
		0 1	0 1 0
		1 0	0 0 1
		1 1	0 0 1
25c	0 0 1	0 0	0 0 1
		0 1	0 1 1
		1 0	0 1 0
		1 1	0 1 0
50c	0 1 0	0 0	0 1 0
		0 1	1 0 0
		1 0	0 1 1
		1 1	0 1 1
75c	0 1 1	0 0	0 1 1
		0 1	1 1 1
		1 0	1 0 0
		1 1	1 0 0
100c	1 0 0	X X	0 0 0
–	1 0 1	X X	1 1 1
–	1 1 0	X X	1 1 1
erro	1 1 1	X X	0 0 0

Esta implementação do controlador da máquina de vender chocolates emprega uma memória ROM para implementar a função de próximo estado. A memória ROM é preenchida diretamente com a função de próximo estado de tal forma que o sequenciamento é gravado na ROM, conforme mostra a Tabela 8.15. A coluna EA corresponde ao estado atual, a coluna '25 50' corresponde às duas entradas, a coluna PE corresponde ao próximo estado, e as saídas são mostradas nas duas colunas à direita.

A ROM é logicamente dividida em oito faixas de endereços, com quatro posições em cada faixa. O estado atual é determinado pelos 3 bits mais significativos do endereço $\langle e_4, e_3, e_2\rangle$. As entradas são ligadas aos 2 bits menos significativos do endereço ($50 \leftrightarrow e_0$ e $25 \leftrightarrow e_1$). Dependendo das combinações de estado atual e entradas, o próximo estado é determinado pelos bits $\langle r_4, r_3, r_2\rangle$ da ROM.

Tabela 8.15: Codificação da ROM da máquina de vender chocolates.

	endereço		conteúdo		
	EA	25 50	PE	choc	dev
estado	$e_4e_3e_2$	e_1e_0	$r_4r_3r_2$	r_1	r_0
$0c$	0 0 0	0 0	0 0 0	0	0
	0 0 0	0 1	0 1 0	0	0
	0 0 0	1 0	0 0 1	0	0
	0 0 0	1 1	0 0 1	0	0
$25c$	0 0 1	0 0	0 0 1	0	0
	0 0 1	0 1	0 1 1	0	0
	0 0 1	1 0	0 1 0	0	0
	0 0 1	1 1	0 1 0	0	0
$50c$	0 1 0	0 0	0 1 0	0	0
	0 1 0	0 1	1 0 0	0	0
	0 1 0	1 0	0 1 1	0	0
	0 1 0	1 1	0 1 1	0	0
$75c$	0 1 1	0 0	0 1 1	0	0
	0 1 1	0 1	1 1 1	0	0
	0 1 1	1 0	1 0 0	0	0
	0 1 1	1 1	1 0 0	0	0
$100c$	1 0 0	0 0	0 0 0	1	0
	1 0 0	0 1	0 0 0	1	0
	1 0 0	1 0	0 0 0	1	0
	1 0 0	1 1	0 0 0	1	0
$inv1$	1 0 1	0 0	1 1 1	0	0
	1 0 1	0 1	1 1 1	0	0
	1 0 1	1 0	1 1 1	0	0
	1 0 1	1 1	1 1 1	0	0
$inv2$	1 1 0	0 0	1 1 1	0	0
	1 1 0	0 1	1 1 1	0	0
	1 1 0	1 0	1 1 1	0	0
	1 1 0	1 1	1 1 1	0	0
$erro$	1 1 1	0 0	0 0 0	0	1
	1 1 1	0 1	0 0 0	0	1
	1 1 1	1 0	0 0 0	0	1
	1 1 1	1 1	0 0 0	0	1

A Figura 8.47 mostra a implementação do controlador. A cada ciclo do relógio o registrador é atualizado em função do estado atual e das entradas. Note que as entradas são sincronizadas pelo relógio.

As duas principais vantagens desta implementação com relação àquela discutida na Seção 8.4.4 são a simplicidade do projeto e a possibilidade de se alterar a lógica do controlador sem grandes mudanças no circuito – basta ajustar o conteúdo da ROM para que este corresponda ao novo diagrama de estados. O circuito de controle é composto por apenas dois blocos, o registrador de estado e a memória ROM, o que simplifica muito a sua depuração e testes.

Espaço em branco proposital.

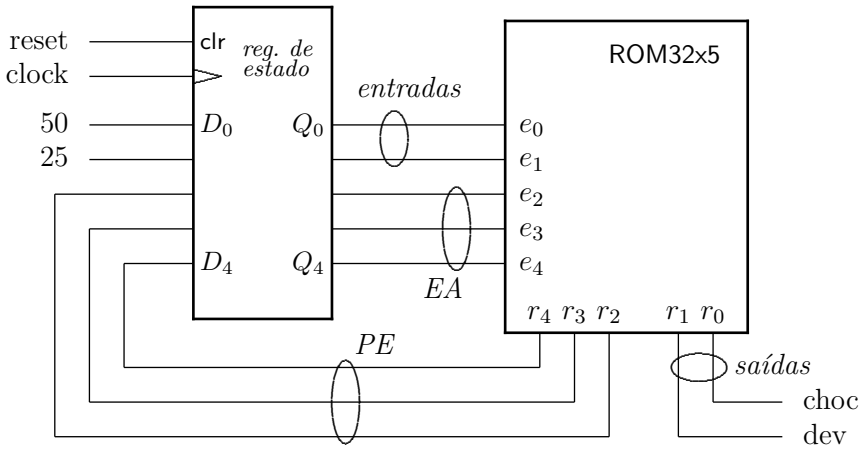


Figura 8.47: Controlador da máquina de vender chocolates.

8.5.2 Otimização da ROM com o Microprograma

Considere um microcontrolador com E entradas e Q bits de estado: a altura da sua ROM é de 2^{E+Q} linhas. Suponha que dentre as entradas, os valores de e_i e e_j nunca são amostrados simultaneamente num mesmo estado. Se esse é o caso, então a altura da ROM pode ser reduzida à metade se um multiplexador for usado para escolher uma dentre e_i ou e_j, e neste caso, $E' = E/2$. Isso é obtido à custa de um bit de saída adicional, que é usado para escolher qual das entradas é amostrada em cada estado. Em geral, essa otimização é vantajosa se, (1) é fácil acrescentar um multiplexador ao projeto, e (2) o número de bits da ROM diminui. O número de bits da ROM é $A \times L$ para altura A e largura L. A otimização é vantajosa se $A/2 \times (L+1) < A \times L$. Geralmente, uma ROM "mais quadrada" ocupa menos área do que uma ROM "mais retangular" e é mais rápida. Essa otimização é mostrada na Figura 8.48.

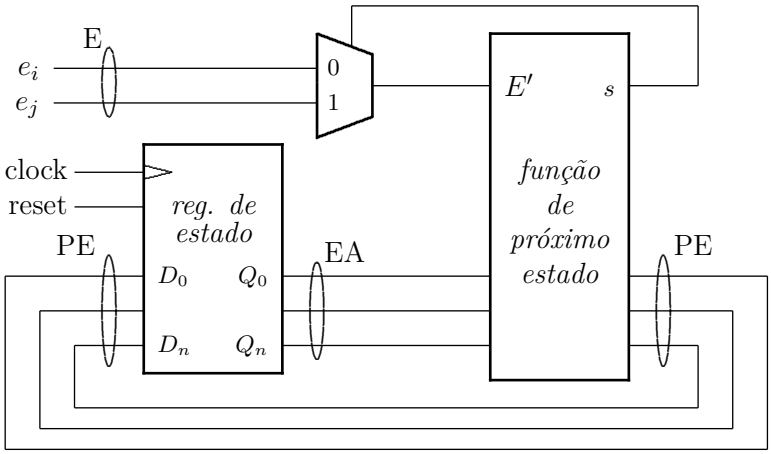


Figura 8.48: Microcontrolador com seleção de entradas.

Caso seja possível aumentar a largura da ROM, pode-se duplicar o campo de próximo estado – todo ele ou somente alguns dos bits. O próximo estado pode ser selecionado diretamente por uma ou mais entradas. Essa otimização reduz a altura da ROM porque uma das suas entradas é eliminada – e_k no diagrama da Figura 8.49 – enquanto que aumenta a largura porque alguns bits são acrescentados para a escolha do próximo estado.

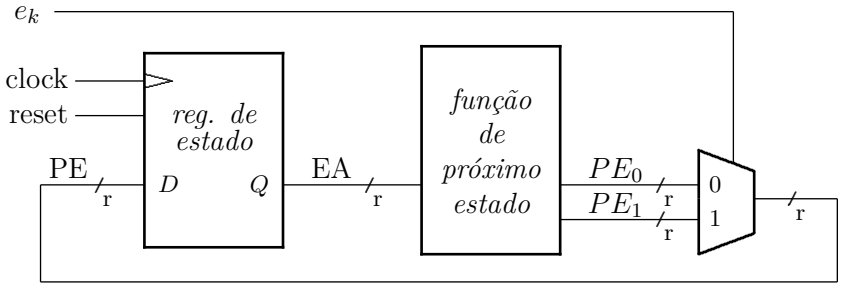


Figura 8.49: Microcontrolador com seleção do próximo estado.

Exercícios

Ex. 8.25 No circuito do Exemplo 8.20 podem ocorrer algumas sequências nas quais a porção final é comum a todas elas. Esta porção comum é chamada de *microrrotina*. Projete o circuito que permite desvios para a microrrotina. Suponha que uma microrrotina possui no máximo 32 microinstruções.

Ex. 8.26 Estenda o projeto do exercício anterior para permitir o desvio condicional para uma microrrotina.

Ex. 8.27 Estenda o projeto do exercício anterior para permitir o desvio condicional para uma de várias microrrotinas. Suponha que existem até oito microrrotinas distintas.

Ex. 8.28 Modifique a Tabela 8.15 associando as entradas 25 e 50 aos bits mais significativos $\langle e_3, e_4 \rangle$. Esta troca é vantajosa? Justifique.

Ex. 8.29 Considere a possibilidade de usar um contador no lugar do registrador de estado. Em que isso altera o nível de complexidade do projeto do microcontrolador?

8.6 Um *Flip-flop* por Estado

As formas de implementação de máquinas de estados que vimos até agora tentam minimizar o número de *flip-flops* que mantêm o estado da ME. Uma alternativa mais perdulária é usar um *flip-flop* para cada estado, e nesse caso, o que se tenta otimizar é o tempo do projetista, ou a facilidade de gerar automaticamente uma implementação da ME. Essa implementação emprega um *flip-flop* por estado, e o projeto deve garantir que, a qualquer instante, o estado da ME é representado por exatamente *um flip-flop* com sua saída em 1, enquanto todos os demais estados/*flip-flops* mantêm suas saídas em 0.

Usaremos como exemplo a ME que detecta sequências de dois ou mais 1s seguidos de 0, descrita no Exemplo 8.19. O diagrama de estados desta ME é mostrado na Figura 8.50, juntamente com os *circuitos de entrada* dos estados à esquerda do diagrama, e os *circuitos de saída* dos estados, à direita. A flecha '$\rightsquigarrow$' indica o estado associado à saída de cada circuito de entrada ou de saída.

O circuito de entrada do estado A é uma porta *or* de três entradas porque são três os estados que antecedem o estado A: o próprio estado A ou os estados B ou D. O circuito de entrada do estado C é uma porta *or* de duas entradas porque os estados que antecedem aquele estado são os estados B ou C.

Espaço em branco proposital.

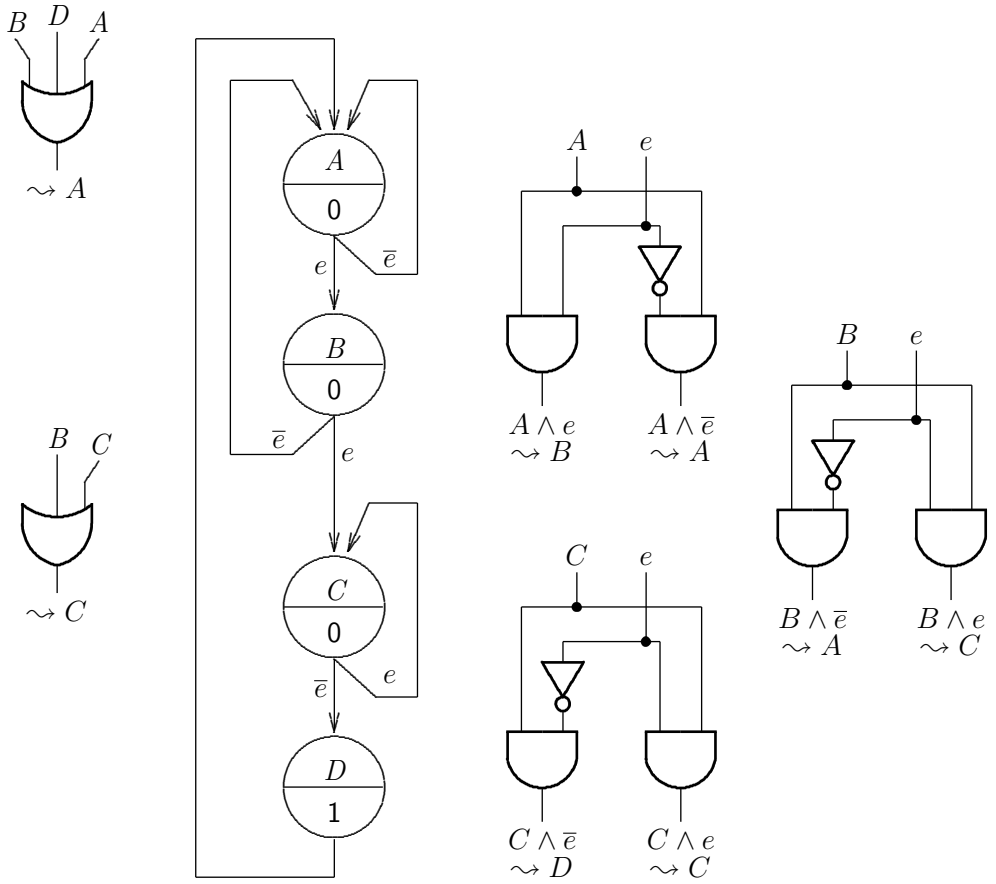


Figura 8.50: Transformação da ME que detecta a sequência $11^{+}0$.

O circuito de saída do estado A depende da entrada e: se $e = 0$ então a ME permanece a esperar por uma entrada 1, senão, $e = 1$ e o próximo estado é B. Os circuitos de saída dos estados B e C são similares. Os circuitos de saída são demultiplexadores cuja saída depende da combinação das entradas da ME.

Espaço em branco propositado.

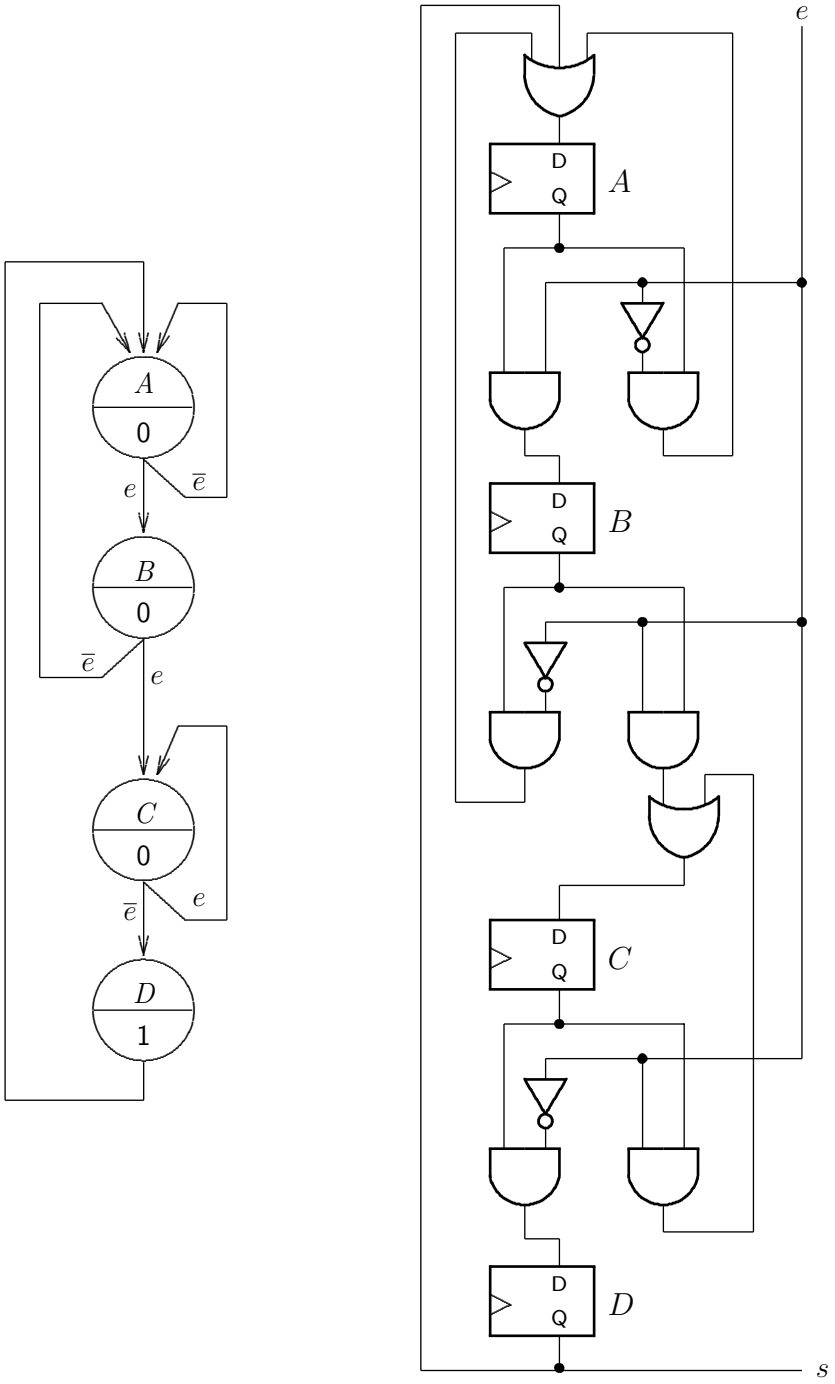


Figura 8.51: Implementação da ME que detecta a sequência $11^{+}0$.

Vejamos a implementação completa da ME com um *flip-flop* por estado, que é mostrada na Figura 8.51. O estado que corresponde ao *flip-flop* está indicado junto ao *flip-flop*. O circuito completo é a junção dos circuitos de entrada e de saída indicados na Figuras 8.50, mais os quatro *flip-flops* para os quatro estados.

No diagrama da figura não está indicado o circuito de inicialização da ME. Se os *flip-flops* não possuem entrada para o sinal de *reset* e nem de *set*, as portas lógicas dos circuitos de entrada

e de saída devem ser usadas para garantir que a ME é inicializada corretamente. No nosso exemplo, a porta *or* na entrada do *flip-flop-A* deve ter uma entrada adicional para que este *flip-flop* seja inicializado em 1. Os circuitos de entrada dos demais *flip-flops* devem empregar portas *and* de três entradas, com o sinal de *reset* ligado a todas as portas *and*, assim garantindo que todos os demais *flip-flops* sejam inicializados em 0.

Como dito inicialmente, essa implementação pode ser perdulária, por geralmente empregar um número maior de *flip-flops* e de portas lógicas do que uma implementação que minimiza o número de *flip-flops*. Simplificar a função de próximo estado de MEs pequenas como as que vimos até agora, com uns poucos estados e umas poucas variáveis, é factível. Esse não é o caso para MEs realistas com mais de 10 estados e/ou de 5 variáveis.

Uma vez compreendida a técnica de projeto com um *flip-flop* por estado, a implementação pode ser derivada pela mera inspeção do diagrama de estados da ME. A entrada de cada *flip-flop*/estado é uma porta *or*, e a saída de cada estado é um demultiplexador, que escolhe, em função das variáveis de entrada relevantes, qual será o próximo estado. Esses componentes são mostrados na Figura 8.52.

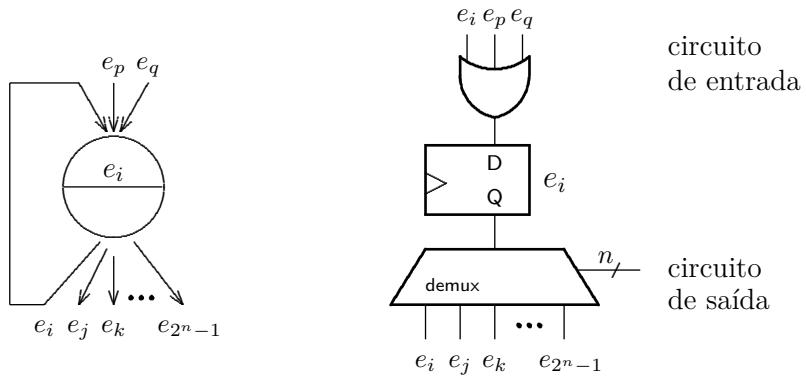


Figura 8.52: Modelo para implementação com um *flip-flop* por estado.

Nos falta implementar a função de saída da ME. No exemplo da ME que identifica a sequência 11^+0, a saída que indica que a sequência foi encontrada é emitida no estado D, uma vez que esta é uma Máquina de Moore. Logo, a saída s é a própria saída Q do *flip-flop*. Numa Máquina de Mealy, cada saída é a conjunção do sinal de estado com a(s) entrada(s) apropriada(s).

8.7 Circuitos Complexos

Esta seção contém exemplos de circuitos sequenciais relativamente complexos que podem ser implementados pela composição de componentes simples tais como memória, contadores, registradores e máquinas de estado. Os controladores dos circuitos desta seção podem ser implementados como Máquinas de Mealy ou de Moore, com um *flip-flop* por estado, ou com microcontroladores. A escolha depende dos requisitos de projeto da aplicação, tais como custo, tempo de projeto, tempo de depuração, e velocidade mínima de operação. Os circuitos foram escolhidos por serem componentes de dispositivos sofisticados e para exemplificar as técnicas de projeto apresentadas nas seções anteriores.

8.7.1 Circuito de Dados Mais Controlador

Os circuitos nos exemplos desta seção consistem de duas partes. Um *circuito de dados* (*data-path*), com somadores, multiplexadores, deslocadores, unidades de lógica e aritmética e registradores, e um *circuito de controle*.

O circuito de controle é uma máquina de estados que ativa os sinais de controle dos circuitos combinacionais nos momentos apropriados. Os dados são processados repetidamente nos circuitos combinacionais, e os resultados parciais são mantidos nos registradores. Ao final de uns tantos ciclos, obtém-se o(s) resultado(s) desejado(s).

A Figura 8.53 mostra um modelo destes circuitos. A linha pontilhada indica a fronteira entre um circuito externo ao projeto (mestre), que fornece ao circuito interno (escravo) as entradas (ε_i e ε_j). Quando disponibilizadas, o mestre captura as saídas produzidas (σ_m e σ_n). No diagrama, as entradas são sinais com larguras i e j, e os sinais de saída têm largura m e n.

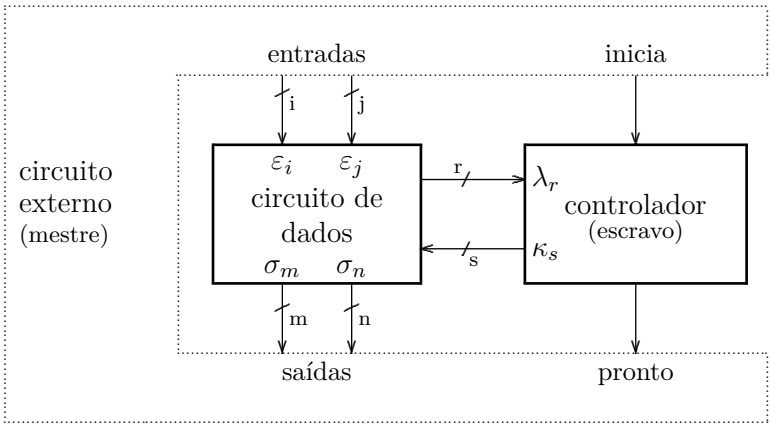


Figura 8.53: Modelo de circuito de dados e controlador.

Quando necessita de uma computação, o circuito externo apresenta as entradas e ativa o sinal *inicia*, indicando que os sinais de entrada estão válidos e estáveis. O controlador observa o estado do circuito de dados (λ_r), e atua sobre os componentes do circuito de dados ativando um ou mais dos sinais de controle (κ_s). Uma vez que os resultados estejam disponíveis, o controlador ativa o sinal *pronto*, indicando essa condição ao circuito externo. Uma vez que detecta o sinal *pronto=1*, o circuito externo faz *inicia=0*.

Considere que, inicialmente, o circuito está em repouso e os sinais *inicia* e *pronto* estão inativos. A sequência de sinalização

$$\mathit{inicia} \rightsquigarrow \text{espera} \rightsquigarrow \mathit{pronto} \rightsquigarrow \overline{\mathit{inicia}} \rightsquigarrow \overline{\mathit{pronto}}$$

sincroniza o mestre com o escravo, e ainda garante que o circuito interno somente observe entradas válidas, e que o circuito externo somente observe respostas válidas. Este é um *protocolo de sincronização* simples e extremamente útil. O termo em Inglês para esse tipo de sincronização é *handshake*, que é uma forma de interação com regras bem definidas.

A Figura 8.54 mostra as máquinas de estado do mestre e do escravo que efetuam a sincronização. O mestre inicia no estado Γ e nele permanece até que necessite que o escravo efetue uma computação. Quando necessita de atenção, o mestre transiciona para o estado α e verifica se o escravo está livre, ao testar o sinal *pronto*. Se o escravo está livre (*pronto* inativo), então o mestre transiciona para o estado β e nele permanece até que o escravo complete sua tarefa.

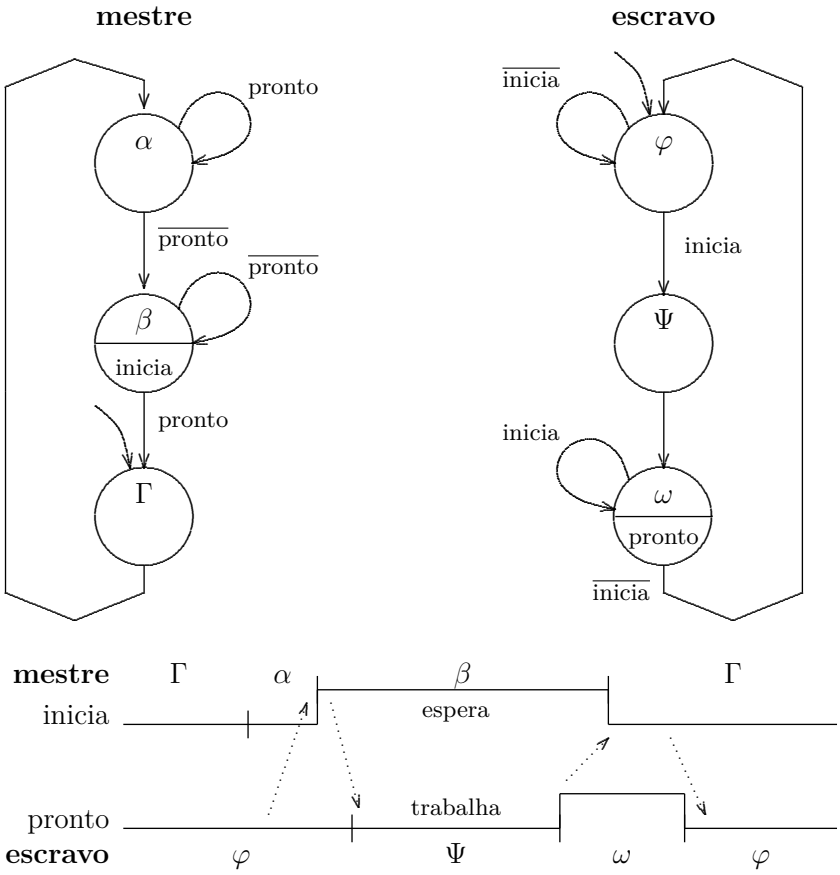


Figura 8.54: *Handshake* entre mestre e escravo.

O escravo inicia no estado φ, e quando o sinal *inicia* é ativado pelo mestre, o escravo transiciona para o estado Ψ e nele permanece até completar a tarefa. Quando isso ocorre, o escravo muda para o estado ω e ativa o sinal *pronto*. O mestre informa ao escravo que observou o resultado desativando o sinal *inicia*. A Figura 8.54 mostra também um diagrama de tempo com a interação entre as duas máquinas de estado. As quatro interações indicadas pelas linhas pontilhadas são bastantes e suficientes para garantir a sincronização entre mestre e escravo.

Para simplificar o diagrama, os estados Γ no mestre, e Ψ no escravo, são indicados como um único estado. Na prática, estes 'estados' podem demorar de dezenas a milhares de ciclos do relógio, dependendo da complexidade dos circuitos do mestre e do escravo.

Falta vermos uma questão importante quanto ao modelo da Figura 8.53, que é a sincronização das transferências de dados do mestre para o escravo. Dependendo dos detalhes de cada projeto, é necessário ter em mente a temporização das trocas de dados entre mestre e escravo. Supondo que estes operam com um mesmo relógio, ou o mestre armazena os dados em registradores e os mantêm estáveis até que *pronto* seja ativado, ou o escravo armazena suas entradas em registradores assim que *inicia* for ativado. O mesmo vale para as saídas: o escravo pode manter os resultados num registrador até que *inicia* seja ativado novamente, ou o mestre registra os resultados assim que *pronto* seja ativado. Seria assaz problemático se nem mestre nem escravo registrassem entradas e/ou saídas.

Exemplo 8.22 Vejamos como projetar a máquina de estados que controla a operação do somador serial da Seção 8.3.4, considerando operandos de 16 bits.

O circuito de dados é mostrado na Figura 8.55. Os dois registradores do lado esquerdo são registradores paralelo-série (com carga em paralelo); o registrador da direita é um registrador série-paralelo – desloca se o sinal de controle *desloc*=1.

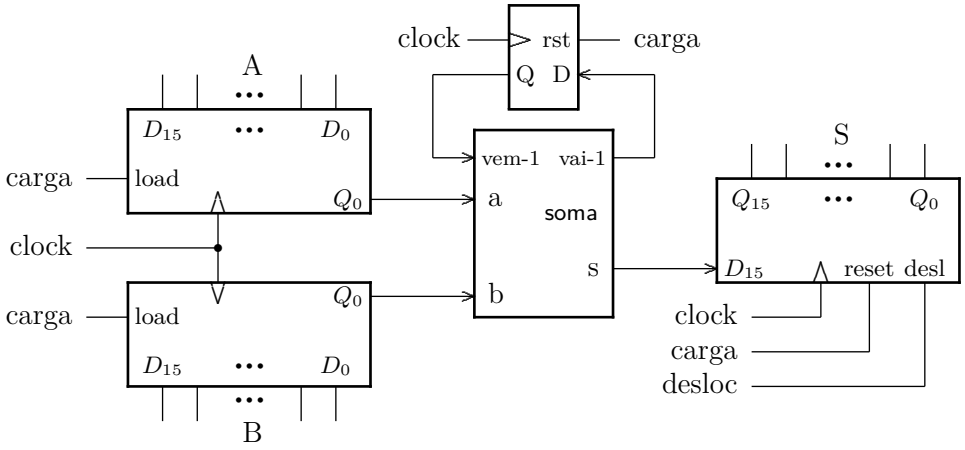
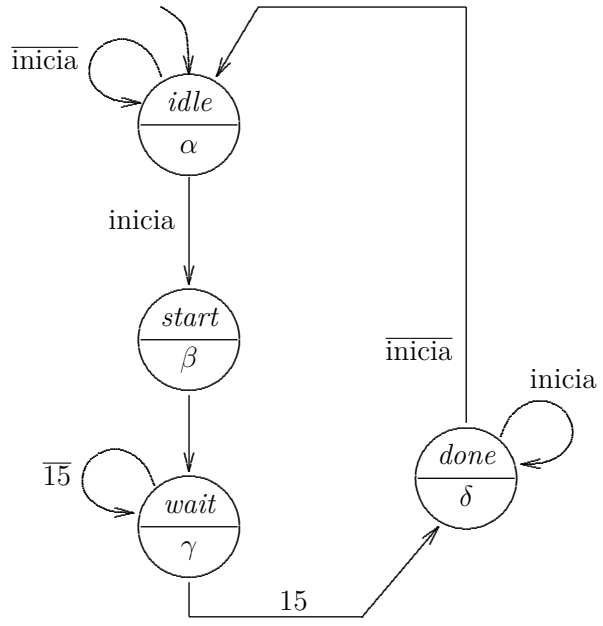


Figura 8.55: Somador serial de 16 bits – circuito de dados.

Depois que os operandos A e B estejam estáveis nas entradas dos registradores, o circuito externo dispara uma operação de soma ativando o sinal *inicia*. Quando a operação completar, a máquina de estados que controla o somador deve ativar o sinal de *pronto* e manter a saída S estável até que *inicia* seja ativado novamente.

O diagrama de estados do controlador é mostrado na Figura 8.56. Para simplificar o desenho, as saídas ativas em cada estado são mostradas à direita do diagrama. Os estados *start* e *wait* correspondem ao estado Ψ da Figura 8.54.



α: carga $\leftarrow 0$
desloc $\leftarrow 0$
pronto $\leftarrow 0$

β: carga $\leftarrow 1$
desloc $\leftarrow 0$
pronto $\leftarrow 0$

γ: carga $\leftarrow 0$
desloc $\leftarrow 1$
pronto $\leftarrow 0$

δ: carga $\leftarrow 0$
desloc $\leftarrow 0$
pronto $\leftarrow 1$

Figura 8.56: Somador serial de 16 bits – controlador.

O projeto do controlador pressupõe a existência de um contador módulo-16, que gera o sinal 15 quando a contagem chega em 15. Tanto os operandos quanto o resultados são mantidos nos registradores do escravo.

No estado *idle*, a ME espera pelo sinal externo *inicia*, e nenhum dos sinais de controle está ativo.

No estado *start*, os registradores da entrada são carregados com os operandos A e B, o *flip-flop* do vai-um é inicializado em zero, e o registrador da direita é inicializado em zero ($S = 0$). O sinal *inicia* pode ser removido a qualquer tempo, desde que o seja antes da contagem chegar em 15.

No estado *wait*, a ME espera durante 16 ciclos – até que o sinal *15* seja ativado – a cada ciclo deslocando o resultado parcial no registrador S.

No estado *done*, a ME avisa ao circuito externo que o registrador S contém a soma $A + B$, fazendo *pronto*=1, e este registrador não é mais deslocado até que *inicia*=1.

Esta ME pode ser implementada com um *flip-flop* por estado, com dois *flip-flops* mais multiplexadores ou uma memória ROM. Uma alternativa ao contador seria incluir a contagem das 16 somas na própria ME, ao invés de usar um contador externo.

O diagrama de tempo da Figura 8.57 mostra o desenrolar da computação de uma soma. ◁

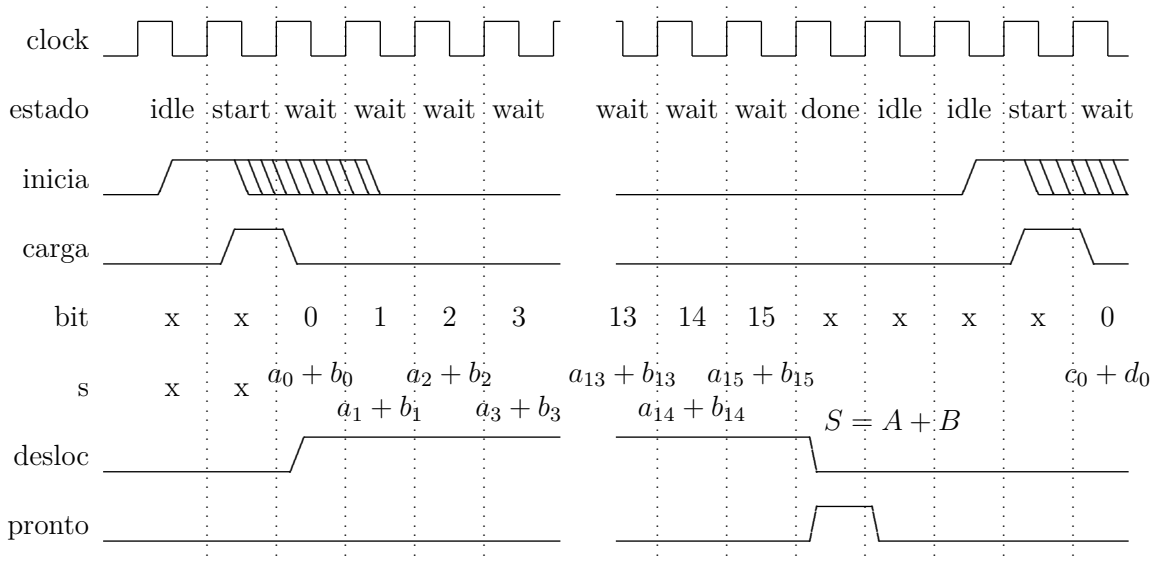


Figura 8.57: Somador serial de 16 bits – diagrama de tempos.

Espaço em branco proposital.

Exemplo 8.23 Você deve projetar um circuito sequencial síncrono que aceita, em sua entrada, uma sequência E de números positivos de 8 bits. A cada pulso do relógio, um novo valor E_i deve ser amostrado, e então as duas saídas *MAX* e *min* mostram os valores máximos e mínimos observados até o i-ésimo ciclo na sequência de entrada. O circuito é especificado pela Equação 8.12. A função com tipo $\mathbb{N} \mapsto \mathbb{B}^8$ corresponde a uma sequência de bytes; a cada ciclo do relógio $(i, i+1, i+2, \cdots)$, um novo valor é apresentado à entrada do circuito $(E(i), E(i+1), E(i+2), \cdots)$. Considere que os elementos da sequência são números naturais.

$$
\begin{array}{l}
inicia : \mathbb{B} \\
E : \mathbb{N} \mapsto \mathbb{B}^8 \\
MAX, min : \mathbb{B}^8 \\
MAXmin : \mathbb{B} \times (\mathbb{N} \mapsto \mathbb{B}^8) \mapsto (\mathbb{B}^8 \times \mathbb{B}^8) \\
MAXmin(inicia, E, MAX, min) \ \equiv \\
\qquad inicia \Rightarrow \forall\, t \ \bullet \ [\, num(MAX) \geqslant num(E(t)) \ \wedge \\
\qquad\qquad\qquad\qquad num(min) \leqslant num(E(t)) \,]
\end{array}
\tag{8.12}
$$

Uma possível implementação para o circuito de dados é mostrada na Figura 8.58.

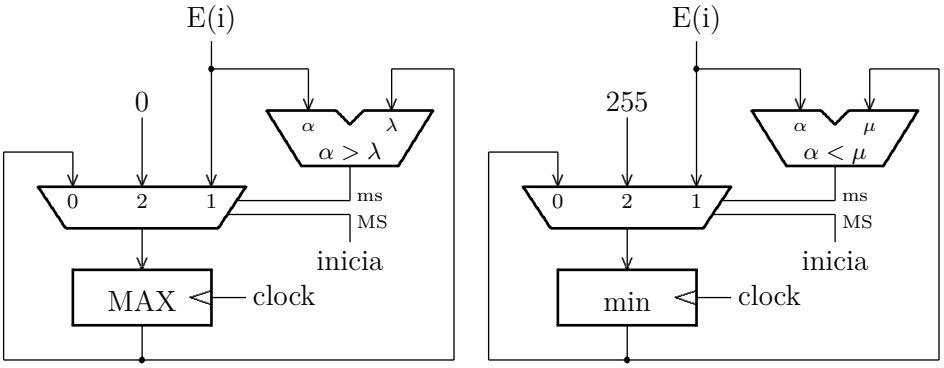


Figura 8.58: Circuito de dados do *MAXmin*.

O sinal *inicia* é ligado ao bit mais significativo da seleção dos multiplexadores e, assim, inicializa o registrador *MAX* com o menor valor possível e o registrador *min* com o maior valor possível. O resultado das comparações é ligado ao bit menos significativo da seleção dos multiplexadores, e atualiza, se for o caso, o conteúdo dos registradores. O circuito de controle, neste exemplo, é trivial. ◁

Para computar o período mínimo do relógio de nossos projetos, usaremos a especificação de temporização da Tabela 8.16.

Tabela 8.16: Especificação temporal dos componentes dos exemplos.

Componente *[ns]*	T_{prop}	T_{cont}	T_{setup}	T_{hold}
registrador	6	1	4	2
somador 8 bits	13	3	–	–
somador 16 bits	25	3	–	–
comparador de magnitude 8 bits	15	4	–	–
multiplexador	5	1	–	–
comparador com zero	8	1	–	–

Exemplo 8.24 Considerando a especificação temporal dos componentes da Tabela 8.16, qual é o período mínimo do relógio do circuito *MAXmin*?

O caminho crítico nos dois circuitos é composto do comparador de magnitude, do multiplexador e do registrador *MAX/min*, e o caminho mais longo é através do comparador de magnitude.

$$T_{min} \geqslant T_{reg} + T_{cMag} + T_{mux} + T_{su} = 6 + 15 + 5 + 4 = 30\,\text{ns}$$

$$T_h = 2ns \geqslant T_{C,reg} + T_{C,mux} = 1 + 1 = 2\,\text{ns}$$

Na Figura 8.58, o caminho mais curto para o T_h é mostrado à esquerda dos registradores. O resultado para T_h é marginal, mas ainda dentro do especificado. ◁

Exemplo 8.25 Vejamos o projeto de um circuito que efetua a multiplicação de dois números positivos de 8 bits, através de adições repetidas. Seu comportamento é especificado pela Equação 8.13. Se os sinais *inicia* e *pronto* estão ambos ativos, então o registrador *prod* contém o resultado da multiplicação.

$$\begin{aligned}
&inicia, pronto : \mathbb{B} \\
&mndo, mdor : \mathbb{B}^8 \\
&prod : \mathbb{B}^{16} \\
&prod8 : \mathbb{B} \times (\mathbb{B}^8 \times \mathbb{B}^8) \mapsto \mathbb{B}^{16} \times \mathbb{B} \\
&prod8(inicia, mndo, mdor, prod, pronto) \;\equiv \\
&\qquad inicia \Rightarrow pronto \Rightarrow \\
&\qquad\qquad [\, num(prod) = num(mndo) \cdot num(mdor) \,]
\end{aligned} \tag{8.13}$$

A implementação do multiplicador é separada em duas partes, um circuito de dados, e a máquina de estados que o controla. A ME que controla o multiplicador é mostrada na Figura 8.59. A interface externa do multiplicador (*prod8*) aceita os operandos multiplicando (*mndo*) e multiplicador (*mdor*). O sinal de controle *inicia* faz com que os operandos sejam carregados nos registradores *mndo* e *mdor*, e dispara a computação do produto. Quando o resultado estiver disponível no registrador *prod*, o sinal *pronto* é ativado. Os estados *load*, *start* e *wait* correspondem ao estado Ψ da Figura 8.54.

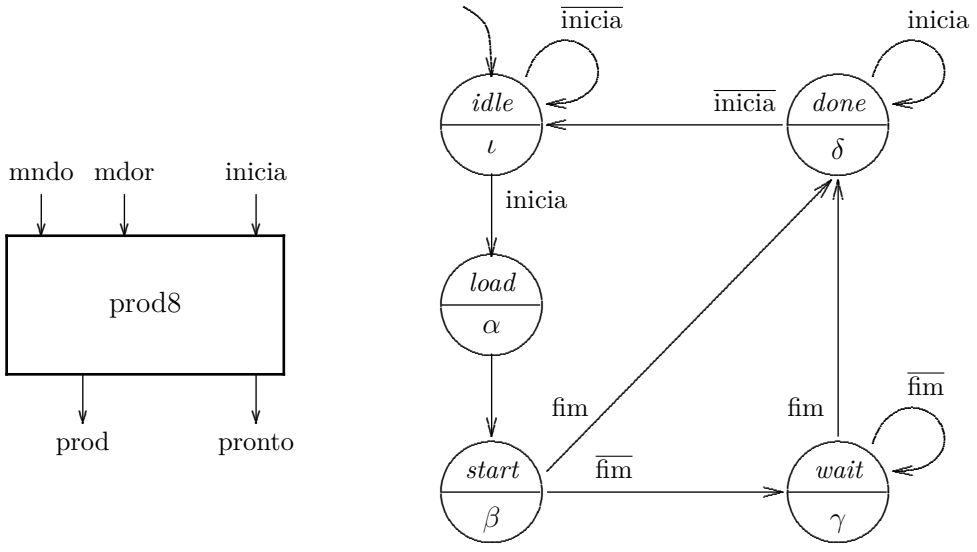


Figura 8.59: Interface e ME que controla o multiplicador.

Numa multiplicação por zero ($N \times 0$), o sinal *pronto* deve ser ativado sem que nenhuma parcela seja adicionada ao registrador *prod*. Se o multiplicador for diferente de zero, então a ME permanecerá no estado *wait* até que todas as parcelas sejam adicionadas.

Tabela 8.17: Sinais de controle do multiplicador.

estado	carM	carC	selC	pronto
idle (ι)	0	0	0	0
load (α)	1	0	0	0
start (β)	0	1	1	0
wait (γ)	0	1	0	0
done (δ)	0	0	0	1

O circuito de dados é mostrado na Figura 8.60 e a Tabela 8.17 contém os sinais ativos em cada estado.

Os dois operandos, multiplicando e multiplicador são carregados nos registradores *mndo* e *mdor* no estado *load*, em resposta a um comando pelo sinal *inicia*.

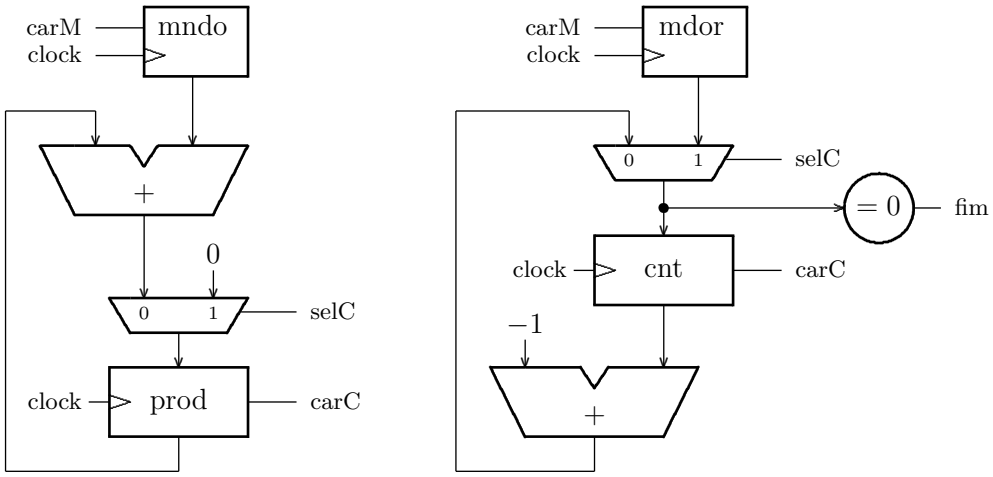


Figura 8.60: Circuito de dados do multiplicador.

O registrador *cnt* atua como o contador de parcelas por adicionar ao produto. A cada parcela adicionada, *cnt* é decrementado; quando *cnt* for zero, o registrador *prod* contém o produto.

No estado *start*, o registrador *cnt* é inicializado com o multiplicado, e o registrador *prod* é inicializado com zero. A cada ciclo, uma nova parcela – o multiplicando – é adicionada ao resultado parcial.

Se, no estado *start* o sinal *fim* está ativo porque o multiplicador é zero, então a computação pode ser encerrada já no estado *start*. Do contrário, A cada ciclo dispendido no estado *wait*, *cnt* é decrementado e *prod* é acrescido de uma parcela do multiplicando.

A última parcela do produto é adicionada quando $cnt = 1$, e o sinal *fim* é ativado ao final desse ciclo, causando a transição para o estado *done*. O sinal *fim* é amostrado na entrada do registrador *cnt* e por isso *fim* é ativado quando $cnt = 1$, ou quando $mdor = 0$.

A Figura 8.61 mostra um diagrama de tempo com duas operações; a primeira é uma multiplicação $N \times 0$, e a segunda uma multiplicação $M \times 2$. Nesse diagrama, os sinais combinacionais são ligeiramente atrasados com relação à borda do relógio e é por isso que os registradores são carregados na borda do relógio do final do estado – o sinal de controle é ativado no estado X_i mas o registrador só é atualizado na borda da transição para o estado X_{i+1}.

O diagrama mostra um produto $N \times 0$ para $N \neq 0$; certifique-se que o circuito opera corretamente com o produto $0 \times N$. ◁

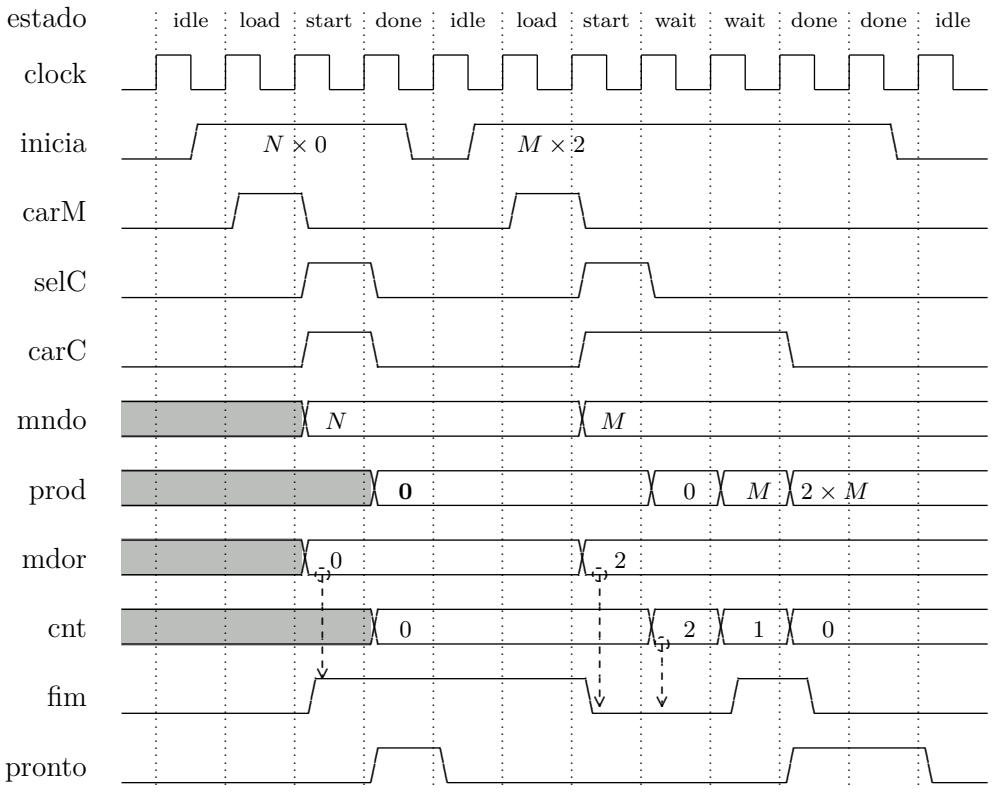


Figura 8.61: Diagrama de tempos de N×0 e N×1.

Exemplo 8.26 Considerando a especificação temporal dos componentes da Tabela 8.16, qual é o período mínimo do relógio do multiplicador?

O circuito com maior tempo de propagação é o que computa o produto, porque o tempo de propagação do somador de 16 bits (25ns) é maior do que aquele do circuito com o somador de 8 bits mais o comparador (13+8ns).

$$T_{min} \geqslant T_{prod} + T_{som16} + T_{mux} + T_{su} = 6 + 25 + 5 + 4 = 40\,\text{ns}$$

O tempo de propagação dos circuitos nas duas entradas do somador é idêntico porque as duas são providas pelos registradores *mndo* e *prod*. O *setup* no caminho crítico é o do registrador com o produto.

$$T_h = 2ns \leqslant T_{C,prod} + T_{C,som16} + T_{C,mux} = 1 + 3 + 1 = 5\,\text{ns}$$

A contaminação no circuito é maior do que T_h. ◁

Exemplo 8.27 Uma pequena alteração no circuito do Ex. 8.25 nos permite computar a exponenciação de inteiros. Se o somador que adiciona as parcelas do multiplicando for substituído por um multiplicador combinacional, o conteúdo do registrador *prod* será $\text{prod} = \text{mndo}^{\text{mdor}}$. O circuito da exponenciação é especificado pela Equação 8.14.

$$
\begin{array}{l}
inicia, pronto : \mathbb{B} \\
valor, exp : \mathbb{B}^8 \\
result : \mathbb{B}^{32} \\
expon8 : \mathbb{B} \times (\mathbb{B}^8 \times \mathbb{B}^8) \mapsto \mathbb{B}^{32} \times \mathbb{B} \\
expon8(inicia, valor, exp, result, pronto) \ \equiv \\
\qquad inicia \Rightarrow pronto \Rightarrow \\
\qquad\quad [\, num(result) = num(valor)^{num(exp)} \,]
\end{array}
\tag{8.14}
$$

Os dois operandos de 8 bits devem ser carregados nos registradores *valor (mndo)* e *exp (mdor)*. O sinal de entrada da interface externa ao multiplicador *inicia* carrega os operandos e dispara a operação. O resultado, representado em 32 bits, é disponibilizado no registrador *result (prod)* quando o sinal *pronto* for ativado pela máquina de estados que controla o circuito. ◁

8.7.2 Bloco de Registradores

O *bloco de registradores* é um dos componentes mais importantes do circuito de dados de um processador. Os *registradores de uso geral* do processador são o nível mais elevado da hierarquia de armazenamento de dados de um computador – os demais níveis são memória cache, memória principal (RAM) e memória secundária (SSD ou disco). Na sua versão mais simples, um bloco de registradores contém duas portas de saída ou de leitura (A e B) e uma porta de escrita, ou de *destino* do resultado (D), além de sinais de controle.

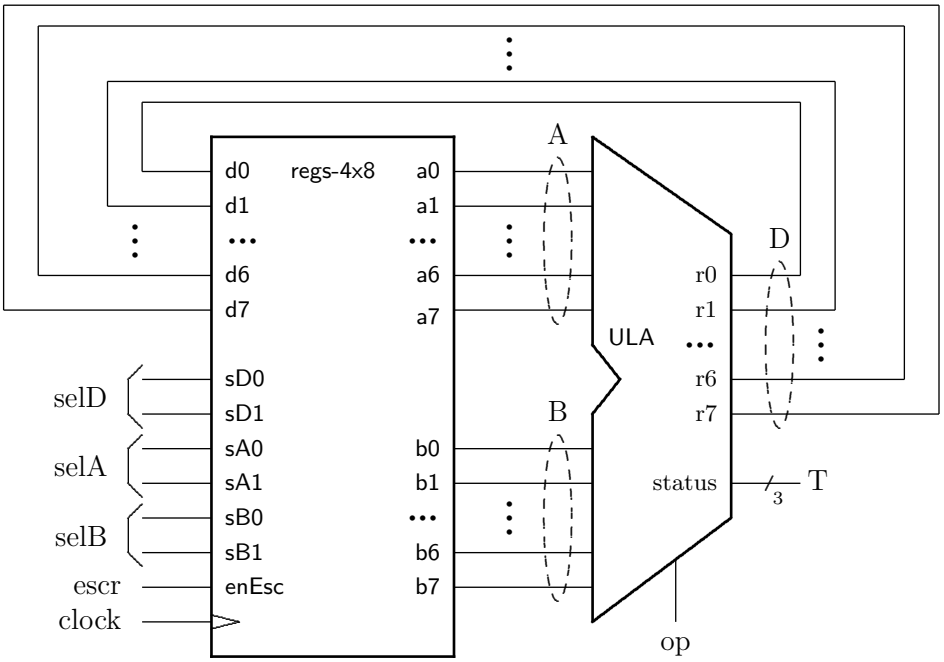


Figura 8.62: Bloco de registradores e Unidade de Lógica e Aritmética.

A Figura 8.62 mostra uma parte do circuito de dados de um processador com a Unidade de Lógica e Aritmética (ULA) e um bloco de registradores com quatro registradores de 8 bits cada. As duas portas de saída A e B permitem a leitura do conteúdo dos registradores apontados por $selA$ e $selB$ respectivamente, enquanto a porta D permite a escrita síncrona do registrador

apontado por $selD$, quando o sinal $escr$ está ativo. Note que a leitura dos registradores apontados por $selA$ e $selB$ depende da operação de um circuito puramente combinacional – os multiplexadores que selecionam os conteúdos que serão exibidos nas *portas de leitura* A e B. A atualização do registrador apontado por $selD$ é síncrona e só ocorre na borda ascendente do sinal de relógio. A conjunto de sinais $selD, escr$ e D é chamado de *porta de escrita*.

Os sinais de controle $escr$, $selA$, $selB$ e $selD$ são gerados pelo controlador do processador, e os valores armazenados nos registradores são computados na medida em que as instruções do programa são executadas pelo processador. A operação a ser efetuada pela ULA é determinada pelo circuito de controle, quando aquele decodifica as instruções de lógica e aritmética.

Um diagrama de blocos do bloco de registradores é mostrado na Figura 8.63.

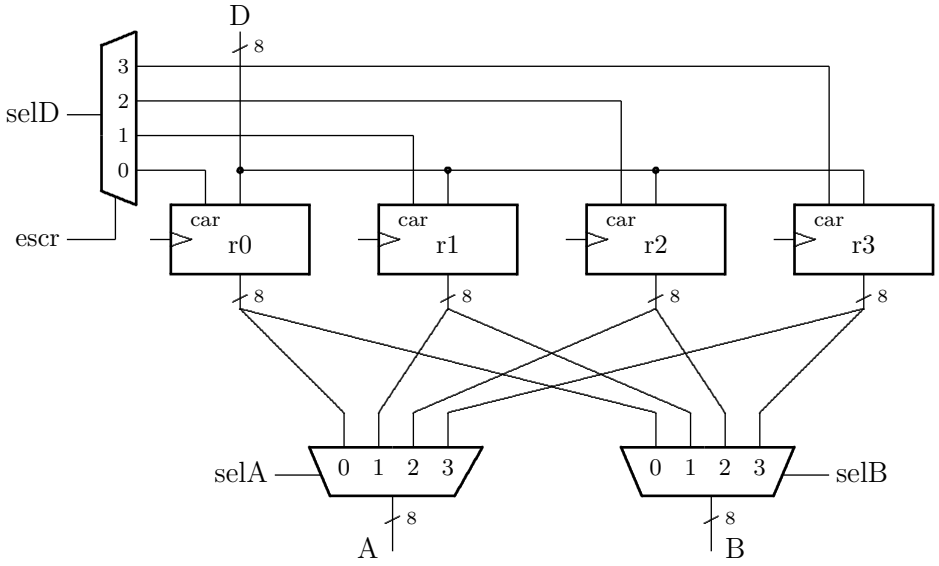


Figura 8.63: Bloco de registradores.

O bloco de registradores é um vetor de quatro elementos de 8 bits cada, e os sinais de seleção têm 2 bits para escolher um dentre os quatro registradores. Quando um par de endereços é aplicado às entradas de seleção, o par de valores armazenados nos registradores selecionados é apresentado nas saídas A e B, base do diagrama. Quando o sinal de escrita está ativo, na borda ascendente do relógio, o conteúdo do registrador apontado por $selD$ é atualizado com o valor na entrada D, no topo do diagrama.

O bloco de registradores implementa uma função memória que mapeia endereços em valores: endereço $\mapsto$ valor. A atualização dos registradores corresponde a uma modificação da imagem da função memória, que é atualizada com o novo valor (valor$'$).

A Equação 8.15 especifica o bloco de registradores da Figura 8.63. As funções mem e mem' são os vetores de 4 bytes; a função mem' é a memória imediatamente após uma atualização.

A função LER especifica as portas de leitura. Esta operação é combinacional: uma vez que os endereços dos registradores estabilizem em $selA$ e $selB$, e decorrido o tempo de propagação através dos registradores e multiplexadores, os conteúdos dos registradores endereçados são exibidos nas portas de leitura A e B.

A função ESC especifica a porta de escrita. Esta operação é sequencial: a função memória é atualizada na borda do relógio se o sinal *escr* estiver ativo. O conteúdo do endereço apontado por $selD$ é sobrescrito com o valor na entrada D.

$$
\begin{array}{l}
mem, mem' : \mathbb{B}^2 \mapsto \mathbb{B}^8 \\
A, B, D : \mathbb{B}^8 \\
selA, selB, selD : \mathbb{B}^2 \\
escr : \mathbb{B} \\
\\
LER : \overbrace{(\mathbb{B}^2 \times \mathbb{B}^2)}^{selA,selB} \mapsto \overbrace{(\mathbb{B}^8 \times \mathbb{B}^8)}^{A,B} \\
\\
ESC : (\overbrace{\mathbb{B}^8}^{D} \times \overbrace{\mathbb{B}}^{escr} \times \overbrace{\mathbb{B}^2}^{selD} \times \overbrace{(\mathbb{B}^2 \mapsto \mathbb{B}^8)}^{\text{mem}}) \mapsto \overbrace{(\mathbb{B}^2 \mapsto \mathbb{B}^8)}^{\text{mem}'} \\
\\
\mathcal{R} : (LER \times ESC) \\
\\
\mathcal{R}(selA, selB, A, B, escr, selD, D) \equiv \left\{ \begin{array}{l} (A, B) = LER(selA, selB) \\ mem' = ESC(D, escr, selD, mem) \end{array} \right.
\end{array}
\tag{8.15}
$$

A Equação 8.15 especifica os tipos das operações do bloco de registradores mas não define, concretamente, o comportamento dos circuitos. Este forma de especificação abstrata deixa em aberto os detalhes da implementação, que devem ser fornecidos pelo projetista do bloco de registradores. O circuito da Figura 8.63 é uma dentre as possíveis implementações para o bloco de registradores.

A Figura 8.64 mostra parte da implementação do bloco de registradores. Para simplificar o diagrama, é mostrado somente o plano correspondente ao bit 0 do bloco de registradores. Os outros sete planos, correspondentes aos bits 1 a 7, são idênticos ao plano do bit 0.

A cada ciclo do relógio os valores dos registradores são atualizados. Se o sinal *escr* estiver inativo, todos os seletores nas entradas dos FFs escolhem a saída do FF e o conteúdo dos registradores não se altera. Se o sinal *escr* estiver ativo, o registrador selecionado por $selD$ será atualizado com o valor presente na porta D. Os sinais $selA$ e $selB$ selecionam os registradores cujos conteúdos serão apresentados nas portas A e B, respectivamente. O sinal de *status* não é armazenado no bloco de registradores.

Espaço em branco proposital.

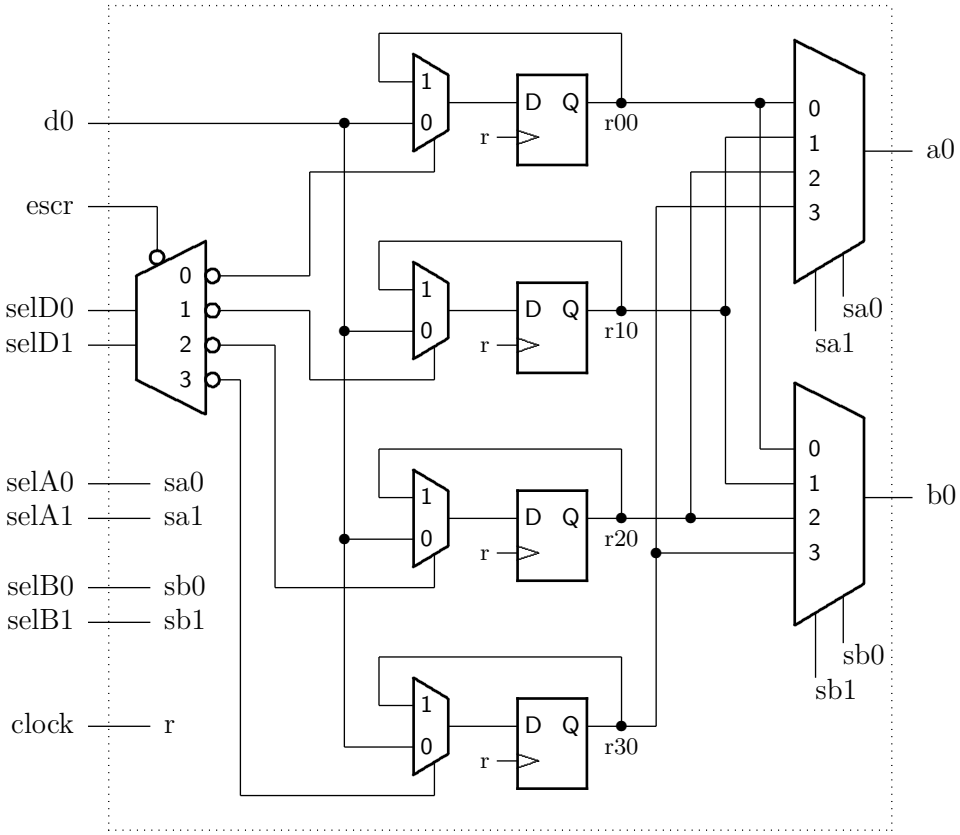


Figura 8.64: Plano do bit 0 do bloco de registradores.

8.7.3 Circuito Integrado com Memória RAM

Esta seção contém uma brevíssima descrição da interface de circuitos integrados de memória RAM porque esses componentes são usados nos exemplos discutidos a seguir.

O comportamento de um circuito integrado de memória RAM é similar ao da memória ROM mas existe a possibilidade de que o conteúdo de cada posição endereçável seja modificado, além de ser examinado. A Figura 8.65 mostra o símbolo para uma memória RAM de 16 palavras, cada palavra com 8 bits de largura. Esta memória é chamada de RAM16x8: 16 palavras (de altura) com 8 bits (de largura).

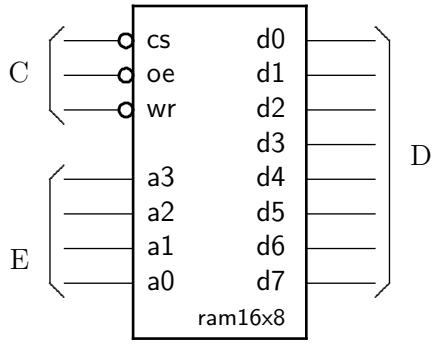


Figura 8.65: Símbolo da memória RAM 16x8.

Memórias RAM geralmente possuem três sinais de controle: cs, oe e wr. O sinal cs, ou *chip select*, habilita a operação do circuito – se inativo, o CI de memória não reage a nenhum dos outros sinais de controle. O sinal oe, ou *output enable*, habilita a saída de dados – se oe está inativo, a saída fica em *three-state* e o conteúdo da posição endereçada não é acessível para leitura. O sinal wr, *write*, define o tipo da referência: wr=0 indica escrita e a posição endereçada é atualizada com o conteúdo das linhas de dados, e wr=1 indica leitura e a posição endereçada é exibida nas linhas de dados quando oe=0. As relações entre os sinais de controle são definidas na Equação 8.16.

$$\begin{aligned} &D : \mathbb{B}^8 \\ &E : \mathbb{B}^4 \\ &cs, oe, wr : \mathbb{B} \\ &ram16x8 : (\mathbb{B} \times \mathbb{B} \times \mathbb{B}) \times \mathbb{B}^4 \times \mathbb{B}^8 \\ &\qquad \mapsto [(\mathbb{B}^4 \mapsto \mathbb{B}^8) \mapsto (\mathbb{B}^4 \mapsto \mathbb{B}^8)] \end{aligned} \tag{8.16}$$

$$\begin{aligned} \overline{cs} \wedge \overline{oe} \wedge wr \Rightarrow &\quad D = ram16x8(E) \\ \overline{cs} \wedge oe \wedge \overline{wr} \Rightarrow &\quad ram16x8(E) \leftarrow D \\ cs : &\quad \text{inativo} \end{aligned}$$

8.7.4 Pilha

Uma *pilha* é uma estrutura de dados que suporta, como um mínimo, três operações: (i) inicialização, (ii) inserção, e (iii) retirada. Elementos são inseridos na pilha através da operação *push*, e removidos da pilha através da operação *pop*. A inicialização deixa a pilha no estado *vazio*. Novos elementos são inseridos no topo da pilha, e o elemento inserido com último *push* é removido pelo próximo *pop*. Se a pilha está vazia um elemento não pode ser retirado, e se a pilha está cheia um novo elemento não pode ser inserido. Esta disciplina é conhecida como *LIFO* por conta do nome em Inglês: *Last In, First Out* – o último que entra é o primeiro a sair, que é o que ocorre numa pilha de cartas ou de pratos.

A Figura 8.66 mostra uma pilha com capacidade para 16 elementos, e com 14 posições já preenchidas. O *topo* da pilha é o endereço da última posição preenchida. O *apontador de pilha* aponta para a posição do topo. Na pilha da Figura 8.66, após a próxima inserção o topo estará na posição 14 e somente mais uma inserção é possível. Se o topo da pilha está na posição 1, somente uma retirada é possível porque esta última esvaziará a pilha. A operação *push* armazena o novo valor no endereço imediatamente acima do topo e então incrementa o apontador. A operação *pop* decrementa o apontador e então mostra o valor que é o novo topo da pilha.

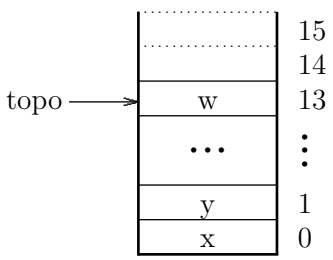


Figura 8.66: Pilha com capacidade para 16 elementos.

Uma pilha como essa pode ser implementada com um circuito sequencial. Os valores mantidos na pilha devem ser armazenados em uma memória que permita tanto a leitura como a atualização de seu conteúdo, como a apresentada na Seção 8.7.3. A implementação da pilha descrita a seguir possui três sinais de controle: *reset*, *push*, *pop*, quatro sinais de *status vazia*, *cheia*, *válido* e *erro*, e um barramento de dados D através do qual os elementos da pilha são inseridos e removidos.

As operações *push* e *pop* são mutuamente exclusivas e sua ativação simultânea é sinalizada como *erro*. O controlador da pilha não responde até que um dos dois sinais seja desativado, resolvendo assim a ambiguidade.

O comportamento da pilha é especificado pela Equação 8.17.

$$
\begin{array}{l}
D : \mathbb{B}^8 \\
topo : \mathbb{B}^4 \\
reset, push, pop, vazia, cheia, erro : \mathbb{B} \\
pilha16x8 : (\mathbb{B} \times \mathbb{B} \times \mathbb{B}) \times \mathbb{B}^8 \\
\qquad\qquad \mapsto \{\, [\, (\mathbb{B}^4 \mapsto \mathbb{B}^8) \mapsto (\mathbb{B}^4 \mapsto \mathbb{B}^8)\,] \times (\mathbb{B} \times \mathbb{B} \times \mathbb{B}) \,\} \\
\\
\qquad \overline{reset} \Rightarrow \quad vazia = 1,\ cheia = 0,\ topo = 0,\ erro = 0 \\
push \wedge \overline{pop} \Rightarrow \quad pilha16x8(topo) \leftarrow D\,;\ topo \leftarrow topo + 1 \\
pop \wedge \overline{push} \Rightarrow \quad topo \leftarrow topo - 1\,;\ D = pilha16x8(topo) \\
push \wedge pop \Rightarrow \quad erro = 1 \\
\text{repouso} : [cheia \triangleleft topo = 15 \triangleright \overline{cheia}] \wedge [vazia \triangleleft topo = 0 \triangleright \overline{vazia}]
\end{array}
\tag{8.17}
$$

A Figura 8.67 mostra os componentes principais do circuito de dados da pilha. O contador *inc-dec* é o apontador da pilha, e aponta sempre para a próxima posição disponível. Quando *reset* é ativado, o valor na sua entrada é carregado e aquele endereça a primeira posição da memória, inicializando a pilha, vazia.

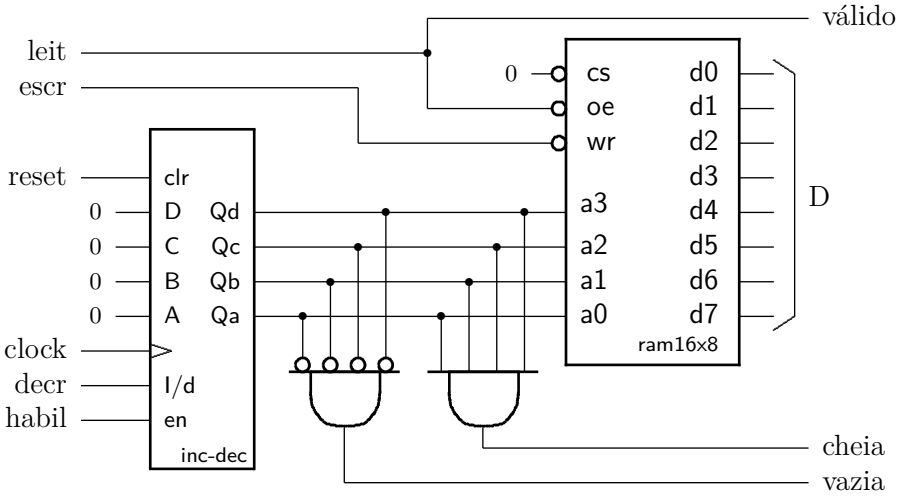


Figura 8.67: Circuito de dados da pilha.

Numa inserção, o valor a ser inserido na pilha é gravado na memória e então o contador é incrementado, apontando a próxima posição vazia. Numa remoção, o contador é decrementado, apontando a última posição que foi preenchida, e então o conteúdo daquele endereço é exibido

no barramento de dados. Os sinais *cheia* e *vazia* informam ao circuito externo sobre o estado da pilha. O sinal *válido* indica que o valor no barramento é válido e pode ser lido.

A Figura 8.68 mostra a máquina de Moore que determina o comportamento da pilha conforme especificado acima. Os sinais ativos em cada estado são mostrados à direita. Normalmente o barramento D está em *three-state* (*leit=1*) e o contador está desabilitado. Uma operação de remoção (*pop=1*) faz com que o contador seja decrementado no estado *rem* (*habil=0,decr=1*) e então o valor no topo da pilha é exibido no estado *val* (*leit=0*). Uma operação de inserção (*push=1*) causa uma escrita na memória (*escr=0*) e então o incremento no contador (*habil=0, decr=0*). Note que a escrita na posição vazia ocorre ao longo de todo o ciclo de relógio do estado *ins* e que o contador somente é incrementado na borda do relógio ao final do ciclo/estado.

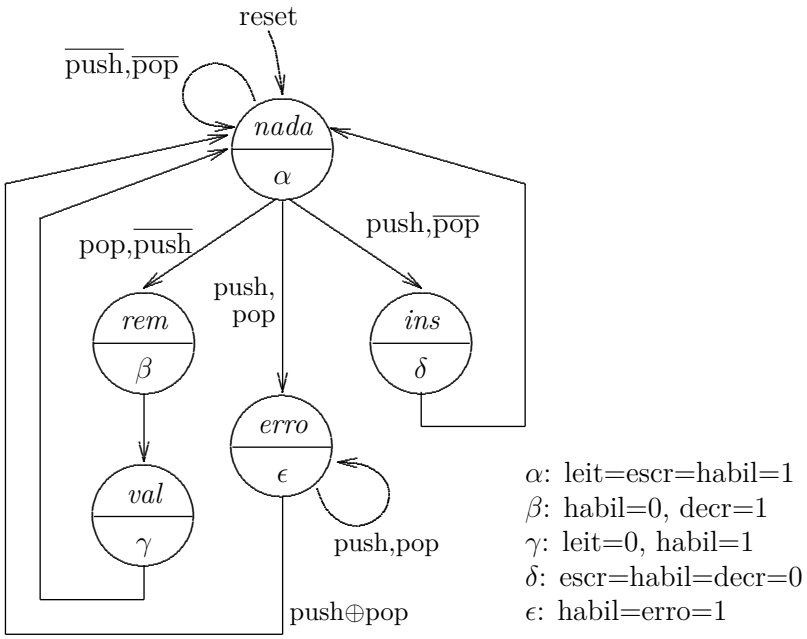


Figura 8.68: Máquina de estados da pilha.

O projetista circuito que faz uso desta pilha deve garantir que: (i) os sinais da interface são acionados/observados corretamente; (ii) o barramento contém dados válidos durante uma inserção; (iii) o valor do topo da pilha é copiado enquanto o sinal *válido* está ativo; e (iv) o período do relógio é suficientemente longo para a correta operação do contador e da memória.

8.7.5 Fila Circular

Uma fila circular pode ser implementada com um circuito sequencial similar ao da pilha vista na Seção 8.7.4. Ao contrário da pilha, inserções ocorrem no *fim* da fila enquanto que remoções ocorrem no *início* da fila, ou na sua *cabeça*. A fila de que trata esta seção é chamada de circular porque os apontadores de início e de fim da fila são implementados com contadores módulo N. Quando a contagem ultrapassa o módulo, a posição apontada pelo contador "fecha o círculo", como indica a Figura 8.69. A disciplina de fila é chamada também de FIFO, de *First In, First Out* – o primeiro a entrar é o primeiro a sair, como numa fila de supermercado.

Essas filas são geralmente empregadas para amortecer diferenças entre as velocidades de dois componentes, um produtor de dados e um consumidor de dados. O tamanho da fila deve ser

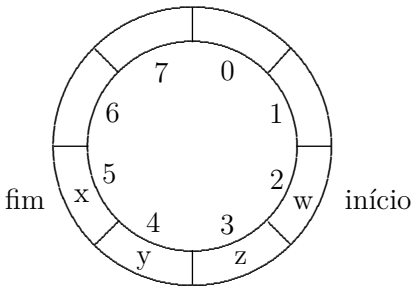


Figura 8.69: Fila circular com capacidade para oito elementos.

tal que o produtor possa produzir dados a uma taxa momentaneamente diferente daquela em que eles são consumidos, embora, no longo prazo, as taxas de produção e consumo devam ser iguais. A fila deve suportar, no mínimo, três operações: (i) inicialização; (ii) inserção; e (iii) retirada. Elementos são inseridos no final da fila com a operação *insere*, e removidos da cabeça da fila através da operação *remove*. A inicialização deixa a fila *vazia*. Se a fila está vazia a retirada de um elemento não é possível, e se a fila está cheia um novo elemento não pode ser inserido. Estes dois casos devem ser sinalizados ao circuito do qual a fila é um componente.

Para implementar uma fila são necessários uma memória RAM e dois contadores, um que aponta para o endereço do início da fila, e outro que aponta para o final da fila. A diferença entre os dois endereços é o tamanho da fila – o cálculo do endereço pode ser problemático porque os dois contadores têm módulo finito. Assim, um terceiro contador pode ser usado para manter o número de elementos na fila. A contagem é incrementada na inserção e decrementada na remoção. O comportamento da fila é especificado pelas Equações 8.18. Os incrementos nos apontadores são módulo-16.

$$
\begin{array}{l}
D : \mathbb{B}^8 \\
ini, fim, tam : \mathbb{B}^4 \\
reset, ins, rem, vazia, cheia : \mathbb{B} \\
fila16x8 : (\mathbb{B} \times \mathbb{B} \times \mathbb{B}) \times \mathbb{B}^8 \\
\qquad\qquad \mapsto \{\, [\, (\mathbb{B}^4 \mapsto \mathbb{B}^8) \mapsto (\mathbb{B}^4 \mapsto \mathbb{B}^8)\,] \times (\mathbb{B} \times \mathbb{B} \times \mathbb{B}^4)\, \} \\
\\
\overline{reset} \Rightarrow \quad vazia = 1,\ cheia = 0,\ tam \leftarrow 0 \\
\quad ins \Rightarrow \quad fila(fim) \leftarrow D;\ fim \leftarrow fim + 1 \\
\quad rem \Rightarrow \quad D = fila(ini);\ ini \leftarrow ini + 1 \\
\text{repouso} : [cheia \triangleleft tam = 15 \triangleright \overline{cheia}] \wedge [vazia \triangleleft tam = 0 \triangleright \overline{vazia}]
\end{array}
\tag{8.18}
$$

A Figura 8.70 mostra os componentes principais do circuito de dados da fila. A fila possui três sinais de controle *reset*, *ins*, e *rem*, dois sinais de *status* *vazia* e *cheia* e um barramento de dados D através do qual novos valores são inseridos e removidos. O sinal *válido* indica que o valor no barramento é válido e pode ser capturado pelo circuito externo. O contador *inc-dec*, – veja o Exercício 8.7 – mantém o número de elementos na fila. Os dois 74163 apontam um para o início e o outro para o fim da fila, e o seletor de endereços determina qual dos dois apontadores é usado para indexar a memória. Quando *reset* é ativado, os três contadores são inicializados em zero, esvaziando assim a fila.

Numa inserção, o valor a ser inserido é gravado no endereço apontado pelo contador de fim da fila, e este é incrementado para que aponte para uma posição vazia. Numa remoção, o

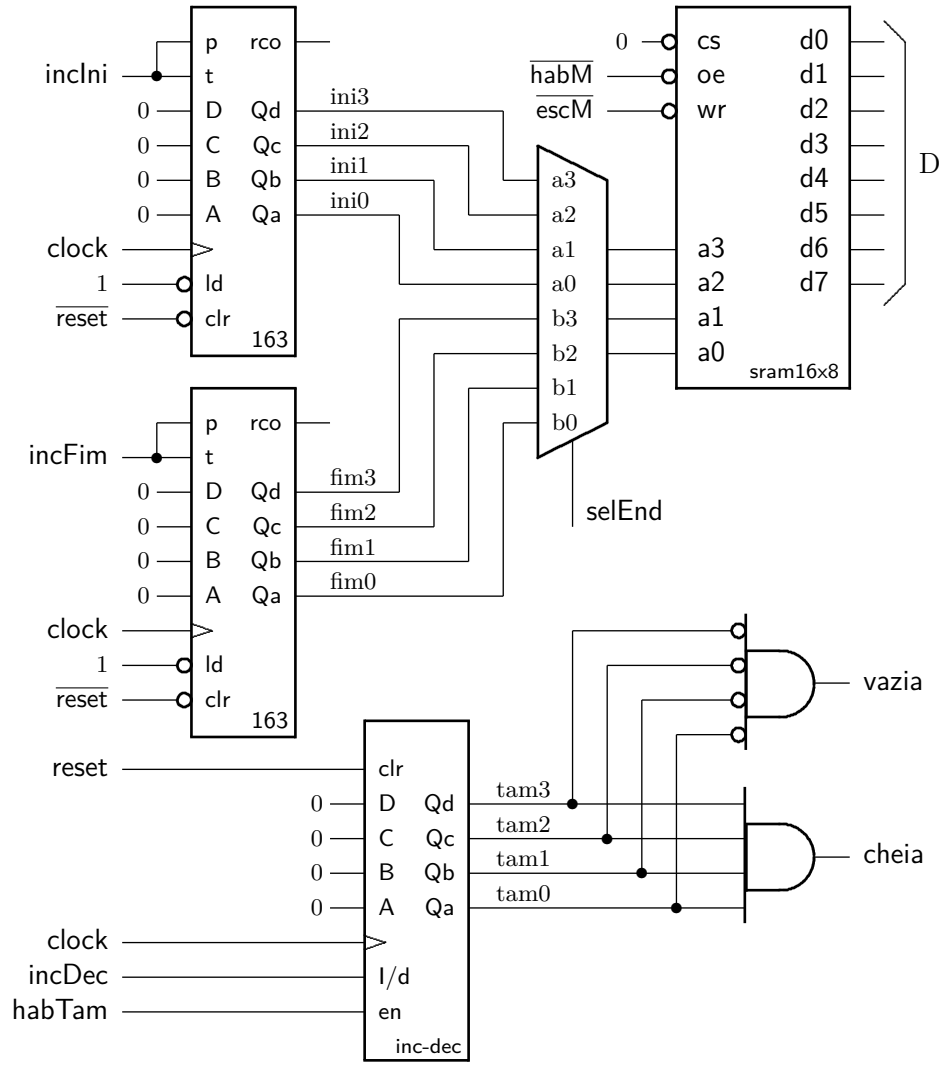


Figura 8.70: Circuito de dados da fila.

conteúdo do endereço apontado pelo contador de início de fila é exibido no barramento, e então este contador é incrementado para que aponte para nova cabeça da fila. Os sinais *cheia* e *vazia* informam ao circuito externo sobre o estado da fila.

Da mesma forma que no projeto da pilha, o usuário desse circuito deve garantir que: (i) o barramento contém dados válidos durante uma inserção; (ii) que o valor da cabeça da fila é copiado enquanto o sinal *válido* está ativo; e (iii) o período do relógio é tão longo quanto o necessário para a correta operação dos contadores e da memória.

Exercícios

Ex. 8.30 Implemente o controlador da máquina de vender chocolates com um FF por estado. *Pista*: percorra o diagrama de estados de trás para a frente para definir as funções de próximo estado de cada FF.

Ex. 8.31 Desenhe os diagramas de tempo para o controlador da pilha, com os sinais de controle do contador e da memória, e certifique-se de que os sinais gerados pela máquina de estados da Figura 8.68 produzem os efeitos esperados.

Ex. 8.32 É possível simplificar a máquina de estados da pilha? *Pista*: projete uma Máquina de Mealy. Repita o Exercício 8.31 para seu novo controlador.

Ex. 8.33 Desenhe diagramas de tempo com os sinais de controle dos três contadores e da memória mostrados na Figura 8.70. Com base nos diagramas de tempo projete a máquina de estados que controla as operações da fila.

Ex. 8.34 Altere a especificação da fila para que ela se comporte como uma fila com capacidade para até 15 elementos. Se a capacidade for excedida, o sinal *cheia* é ativado; se a fila estiver vazia, o sinal *vazia* é ativado.

Ex. 8.35 A máquina de estados que controla o multiplicador do Exemplo 8.25 é uma Máquina de Moore. Projete uma Máquina de Mealy com funcionalidade equivalente. Talvez seja necessário alterar o circuito de dados.

Ex. 8.36 Indique no diagrama de tempos da Figura 8.61 todos os eventos do protocolo de sincronização entre mestre e escravo.

Ex. 8.37 Altere o circuito do Exemplo 8.25 para que o resultado seja o fatorial da entrada.

Ex. 8.38 Projete um circuito que efetua a divisão inteira de dois números positivos de 8 bits, como especificado na Equação 8.19. O circuito deve implementar a divisão por subtrações repetidas. Seu projeto deve conter o circuito de dados e a máquina de estados que o controla. Os dois operandos são carregados nos registradores *ddndo* e *dvsor* quando o sinal de entrada da interface externa *inicia* é ativado. Os valores finais do quociente e do resto são carregados nos registradores *quoc* e *resto* quando o sinal *pronto* for ativado pela máquina de estados do controlador.

$$
\begin{array}{l}
inicia, pronto : \mathbb{B} \\
ddndo, dvsor : \mathbb{B}^8 \\
quoc, resto : \mathbb{B}^8 \\
div8 : \mathbb{B} \times (\mathbb{B}^8 \times \mathbb{B}^8) \mapsto (\mathbb{B}^8 \times \mathbb{B}^8) \times \mathbb{B} \\
div8(inicia, ddndo, dvsor, quoc, resto, pronto) \ \equiv \\
\qquad inicia \Rightarrow pronto \Rightarrow \\
\qquad\quad [\, num(ddndo) = num(dvsor) \cdot num(quoc) + num(resto) \,]
\end{array}
\tag{8.19}
$$

Ex. 8.39 Altere o sua resposta ao Ex. 8.38 para que ele compute uma aproximação para a raiz quadrada. Este exercício é similar ao Exemplo 8.27.

Espaço em branco proposital.

Ex. 8.40 Projete um circuito combinacional que efetua a comparação de magnitude de dois números positivos de 16 bits. Este circuito tem duas entradas P e Q, de 16 bits cada, e três saídas que são *menor*, *igual* e *maior*, definidas na Equação 8.20.

$$\begin{array}{l} P, Q : \mathbb{B}^{16} \\ menor, igual, maior : \mathbb{B} \\ menor = num(P) < num(Q) \\ igual = num(P) = num(Q) \\ maior = num(P) > num(Q) \end{array} \tag{8.20}$$

Ex. 8.41 Um circuito captura dados emitidos por um sensor e os armazena numa memória com 1024 palavras de 16 bits. Os dados são gravados e mantidos em ordem crescente. Especifique e projete um circuito que encontra a posição na memória na qual um valor desejado se encontra, ou indica que o valor não existe. A implementação do circuito deve ser eficiente do ponto de vista do tempo de operação. Seu projeto deve conter o circuito de dados e a máquina de estados que o controla. *Pista*: por 'eficiente' entenda-se um algoritmo de busca com tempo de execução melhor do que "proporcional a N".

Índice Remissivo