MAC 412 – Organização de Computadores
Lógica Combinatória e Sequencial

Prof. Siang

(Nota de aula baseada parcialmente no livro de T.R. Blakeslee, *Digital Design with Standard MSI and LSI*, John-Wiley and Sons, 1975.)

1 Lógica Combinatória

Figura 1.1

Num sistema de lógica combinatória, as saídas são completamente definidas pelas entradas no presente momento e não dependem de entradas passadas nem de saídas computadas anteriormente. Não há portanto memória para armazenar valores anteriores.

1.1 Descrição de funções lógicas por meio de tabelas de verdade

Seja o problema de "display" digital que consiste em sete segmentos luminosos, como indica a Figura 1.2 seguinte.

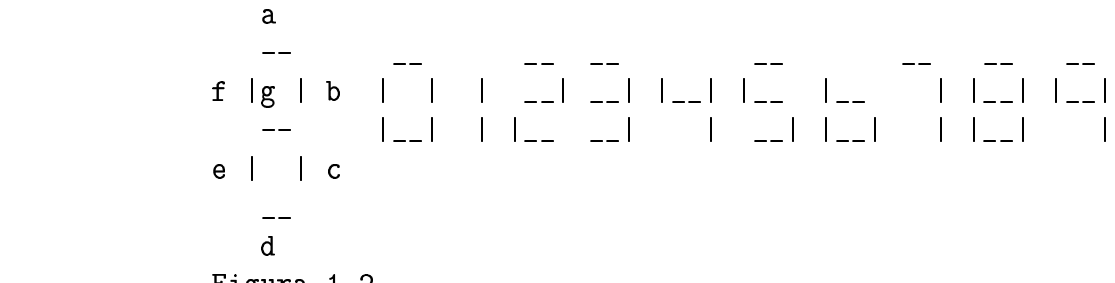


Figura 1.2

Segmentos apropriados devem ser ligados ou acesos de acordo com uma entrada que é um dígito decimal codificado em forma binária, por 4 linhas A, B, C e D. Isso pode ser representado pela tabela de verdade da Figura 1.3 Por exemplo, para as entradas $A = 0, B = 1, C = 0$ e $D = 0$ (correspondentes ao dígito decimal 4), os segmentos b, c, f e g devem ser ligados. Assim, na linha correspondente a $ABCD = 0100$, as saídas ou colunas b, c, f e g são iguais a 1.

Numero	Entradas				Saidas						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

Figura 1.3

Também podemos olhar a tabela de uma outra maneira. O segmento e, por exemplo, deve ser ligado quando a entrada é o número decimal 0, 2, 6 ou 8. Assim, a coluna correspondente a saída e tem 1 exatamente nas linhas correspondentes as entradas 0, 2, 6 e 8.

Observe-se que neste caso queremos o "display" de um dígito *decimal*. Portanto a entrada deve ser um número decimal de 0 a 9 (ou na forma binária, de 0000 a 1001). A combinação 1100, por exemplo, nunca ocorre e para essa combinação, tanto faz a saída assumir 0 ou 1. Uma tabela completa teria 16 linhas (isto é, 2^4 linhas), onde nas linhas 1010 até 1111 as saídas são geralmente marcadas como X (significando "tanto faz").

1.2 Descrição de funções lógicas por equações lógicas

Vimos no exemplo que a saída e deve ser igual a 1 para as entradas correspondentes a 0, 2, 6 e 8. Assim,

$e = 0$ ou 2 ou 6 ou 8

Indicando-se "ou lógico" por soma "+" e "e lógico" por produto ".",

- 0 por $A'B'C'D'$
- 2 por $A'B'CD'$
- 6 por $A'BCD'$
- 8 por $AB'C'D'$

temos a seguinte equação lógica

$$e = A'B'C'D' + A'B'CD' + A'BCD' + AB'C'D'$$

Certos problemas são especialmente bem expressos por equações lógicas. Por exemplo, um sistema de ignição deve permitir a partida do motor quando a chave de ignição está na posição de ligada e o cinto de segurança do motorista está apertado e, ainda, cada um dos demais cintos de segurança ou está apertado ou não há peso no referido assento. Supondo um total 4 passageiros mais o motorista, temos

partida = chave.cinto_1.(cinto_2 + peso_2')... (cinto_5 + peso_5')

Como temos 10 variáveis de entrada, uma descrição pela tabela de verdade teria 2^{10} ou 1024 linhas para cobrir todas as combinações possíveis. Nesse exemplo, a equação lógica é uma forma mais adequada para descrever o problema.

1.3 Descrição de funções lógicas por diagramas de Veitch

Consideremos uma função lógica de 4 variáveis A, B, C e D. A função fica dada se indicarmos todas aquelas combinações das 4 variáveis de entrada para as quais a saída é igual a 1. O diagrama de Veitch da Figura 1.4 apresenta 16 quadrados, cada um correspondendo a uma combinação das variáveis (no caso, 4 variáveis). Cada quadrado pode ser encarado como uma linha da tabela de verdade.

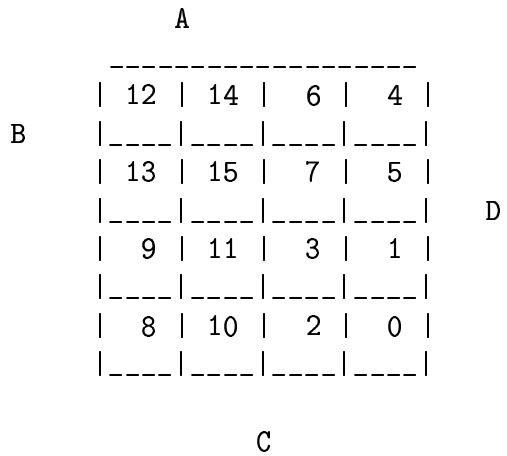


Figura 1.4

O quadrado 7 por exemplo, corresponde a $A = 0, B = 1, C = 1$ e $D = 1$. Uma função lógica pode então ser dada indicando quais os quadrados em que a função assume valor 1.

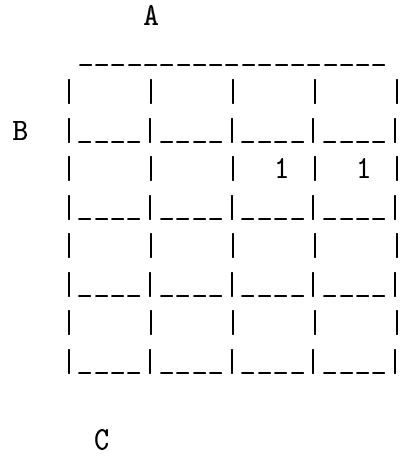
A função F da Figura 1.5 (a) indica que ela assume valor 1 quando as entradas valem 5 ou 7.

A função G da Figura 1.5 (b) assume 1 para as combinações 9, 11, 13 ou 15.

Visto desta forma, o diagrama de Veitch não é nada mais uma outra forma de descrever funções lógicas. Vejamos agora como ele pode útil para simplificar funções lógicas.

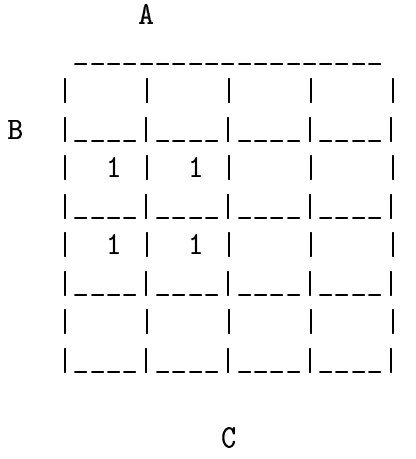
Tomemos novamente as funções F e G. Aplicando propriedades da Algebra Booleana, podemos escrever

$$F = A'BC'D + A'BCD = A'BD(C + C') = A'BD$$



F = A´B C´D + A´B C D

Figura 1.5 (a)



G = A B´C´D+A B´C D+A B C´D+ABCD

Figura 1.5 (b)

$$
\begin{aligned}
G &= AB'C'D + AB'CD + ABC'D + ABCD \\
&= AB'D(C' + C) + ABD(C' + C) \\
&= AB'D + ABD = AD(B' + B) = AD
\end{aligned}
$$

O diagarama de Veitch pode ser usado para derivar essas simplificações diretamente. Vejamos a Figura 1.6

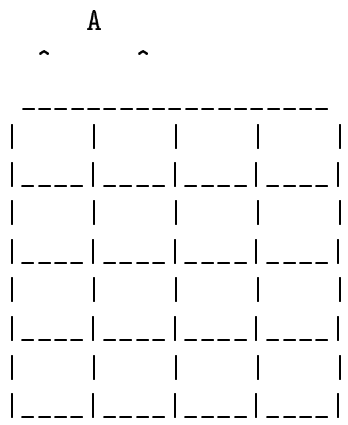


Figura 1.6

Em relação à variável A, os 16 quadrados podem ser divididos em duas partes: metade "iluminada" pelas "luzes" A e metade não iluminada (ficando na sombra). O mesmo pode ser pensado em relação às variáveis B, C e D. O que queremos é caracterizar ou "iluminar" uma região correspondente a determinada função de maneira mais simples.

A função G, por exemplo, pode ser descrita pela interseção das regiões iluminadas por A e D. Vejam a Figura 1.7 (a).

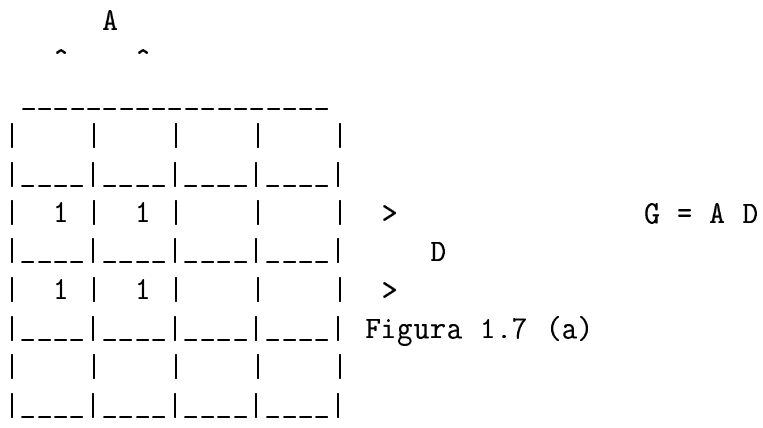
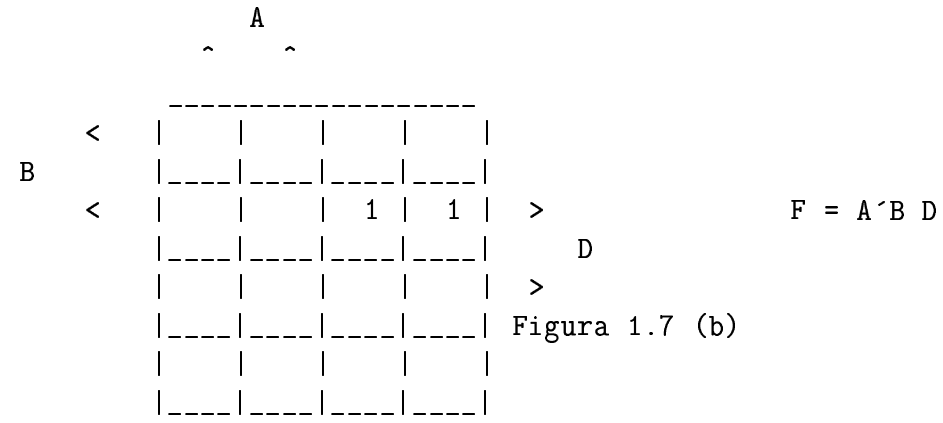


Figura 1.7 (a)

A função F, por sua vez, é a interseção das regiões iluminadas por B e D, e fica na sombra de A. Vejam a Figura 1.7 (b).



Pode-se verificar que 2 quadrados adjacentes no diagrama de Veitch correspondem a um produto de 3 variáveis (complementadas ou não); 4 quadrados adjacentes correspondem a um produto de 2 variáveis. Aqui o significado de adjacência deve ser estendida para incluir lados opostos do diagrama. A Figura 1.8 contém alguns exemplos.

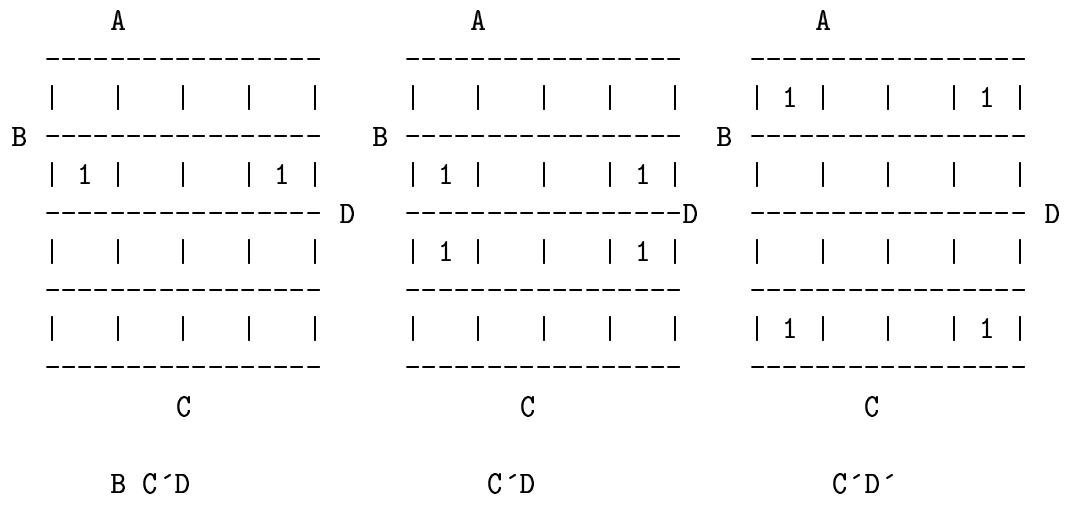
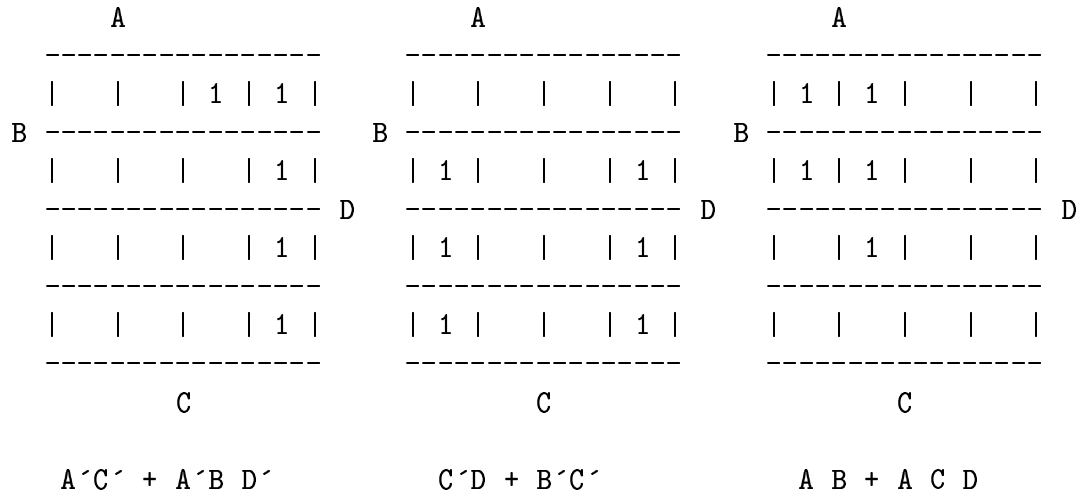


Figura 1.8

A Figura 1.9 contém mais exemplos sobre o uso do diagrama. Convém estudá-los com cuidado.

Voltando ao problema do "display" dos sete segmentos, vamos simplificar a equação para a saída d. Ela é igual a 1 para as entradas 0, 2, 3, 5, 6 e 8. Para as combinações 10 até 15, tanto faz a saída d ser 0 ou 1. Na Figura 1.10 isso é representado por um "X".

Na Figura 1.10, fizemos esses X's iguais a 0 ou 1, de maneira que mais convém à simplificação.


A
B
D
C
A´C´ + A´B D´
A
B
D
C
C´D + B´C´
A
B
D
C
A B + A C D

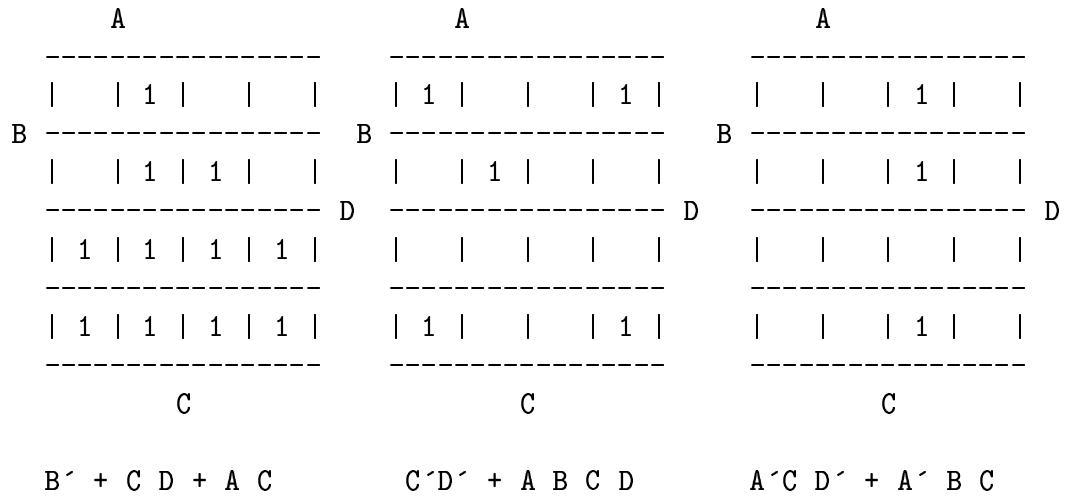

A
B
D
C
B´ + C D + A C
A
B
D
C
C´D´ + A B C D
A
B
D
C
A´C D´ + A´ B C

Figura 1.9

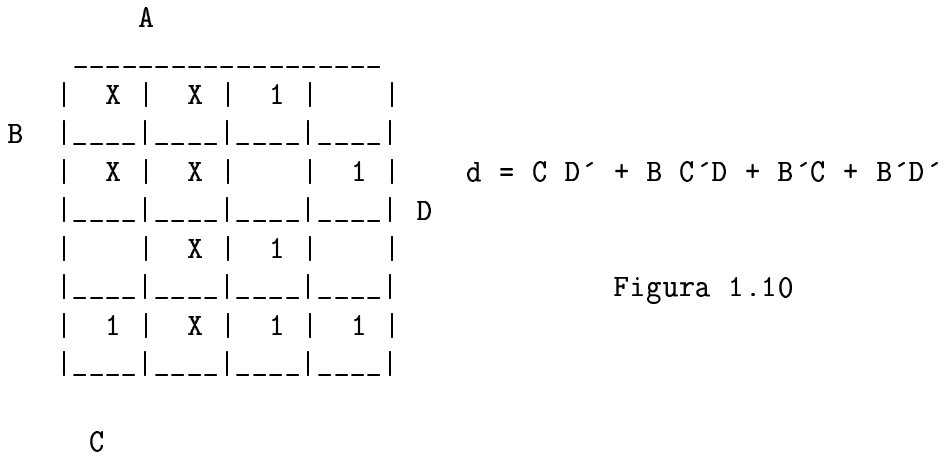


Figura 1.10

1.4 Projetos usando MSI e LSI

Projetistas de circuitos lógicos antigamente (por volta da década dos 60) preocupavam-se em simplificar equações lógicas, visando a minimização do número de portas lógicas ("gates") para a sua implementação. A Figura 1.11 mostra alguns tipos de portas lógicas:

Figura 1.11

A Algebra Booleana tem fornecido as ferramentas básicas para essa finalidade. Métodos usando diagramas foram desenvolvidos para facilitar a aplicação das regras de simplificação. Além do diagrama de Veitch, o de Karnaugh tem a mesma finalidade.

A tecnologia de circuitos integrados possibilita a integração de muitas portas lógicas num só circuito. Um circuito MSI ("Medium Scale Integration") contém tipicamente algumas centenas de portas. Um circuito LSI ("Large Scale Integration") contém milhares de portas. Hoje já temos circuitos integrados (VLSI ou "Very Large Scale Integration") com várias centenas de milhares de portas, numa pastilha medindo menos que 1 cm^2. A densidade de portas continua a crescer com o avanço da tecnologia, e pode atingir vários milhões de portas numa só pastilha.

O projetista deve portanto conhecer quais os circuitos "enlatados" que já existem e que ele pode adquirir no mercado. Toda a circuitaria para o "display" dos 7 segmentos, por exemplo, encontra-se num circuito integrado que pode ser facilmente encontrado no mercado.

A existência de pastilhas MSI e LSI já prontas não elimina ainda a necessidade dos

métodos de simplificação. No projeto de um sistema digital, uma parte ainda é feita com lógica aleatória, principalmente para a integração das pastilhas MSI e LSI. O projetista deve portanto conhecer as técnicas básicas de simplificação.

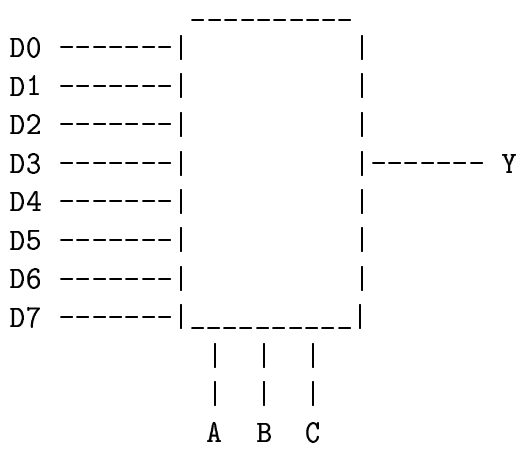


Figura 1.12

Multiplexador ou Seletor A Figura 1.12 ilustra um exemplo de um circuito MSI: um multiplexador ou seletor. Num multiplexador de 8 entradas $D_0, D_1, \ldots, D_7$, uma delas é selecionada para aparecer na saída Y, conforme o número de controle definido pelos 3 bits A, B e C. Temos

- $ABC = 000 \leftrightarrow Y = D_0$
- $ABC = 001 \leftrightarrow Y = D_1$
- $ABC = 010 \leftrightarrow Y = D_2$
- $ABC = 011 \leftrightarrow Y = D_3$
- $ABC = 100 \leftrightarrow Y = D_4$
- $ABC = 101 \leftrightarrow Y = D_5$
- $ABC = 110 \leftrightarrow Y = D_6$
- $ABC = 111 \leftrightarrow Y = D_7$

Em outras palavras, a saída Y é dada por

$$Y = A'B'C'D_0 + A'B'CD_1 + A'BC'D_2 + ... + ABCD_7$$

Fazendo-se ABC variar ciclicamente entre 000 a 111, a saída Y assume ciclicamente D_0 a D_7. O multiplexador pode assim ser usado para transmitir 8 sinais diferentes em uma única linha. Em cada intervalo de tempo, um determinado sinal é transmitido. Diz-se que o tempo é repartido.

Há ainda uma outra utilidade do multiplexador. O multiplexador de 8 entradas pode ser usado para implementar *qualquer* função de 4 variáveis. Vejamos como isso é feito. Seja uma função

$$F = A'B'C'D' + A'B'CD + A'BC'D + A'BC'D' + AB'C'D + AB'CD' + ABC'D' + ABC'D$$

No multiplexador, os sinais de controle A, B, C correspondem às variáveis A, B, C da função F. A saída Y deve dar origem ao valor de F. Precisamos saber quais os valores que devem ser usados como as entradas $D_0, D_1, \ldots, D_7$. Isso é facilmente obtido como se segue.

$A'B'C'$	$D_0 = D'$
$A'B'C$	$D_1 = D$
$A'BC'$	$D_2 = D + D' = 1$
$A'BC$	$D_3 = 0$
$AB'C'$	$D_4 = D$
$AB'C$	$D_5 = D'$
ABC'	$D_6 = D' + D = 1$
ABC	$D_7 = 0$

A Figura 1.13 mostra a realização da função desejada.

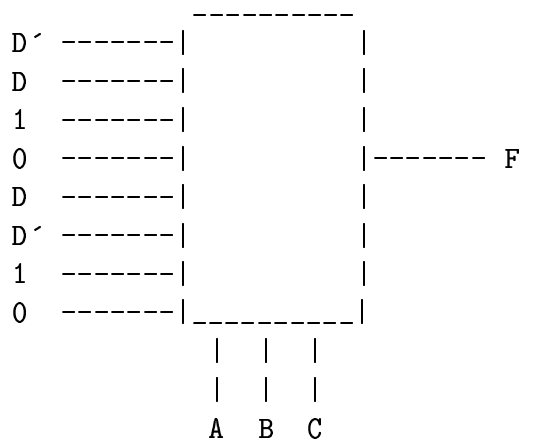


Figura 1.13

Decodificador ou demultiplexador Um decodificador "4-para-16" é mostrado na Figura 1.14. Ele tem 4 entradas A, B, C e D, um controle E (sinal "Enable" ou habilitado) e 16 saídas de 0 a 15. A "bolinha" na entrada do sinal de E significa que o sinal Enable ou habilitado é contrário ao usual (isto é, ao invés de 1 para habilitado e 0 para inibido, usamos 0 para habilitado e 1 para inibido). Assim, se $E = 1$, o circuito está inibido, e todas as saídas 0 a 15 valem 1. Se $E = 0$, o circuito está habilitado, então a entrada de 4 bits $ABCD$ determina qual das saídas valerá 0, enquanto que as demais saídas permanecem 1. Por exemplo, se $E = 0$ e $ABCD = 0100$, então a saída 4 será igual a 0 e todas as outras

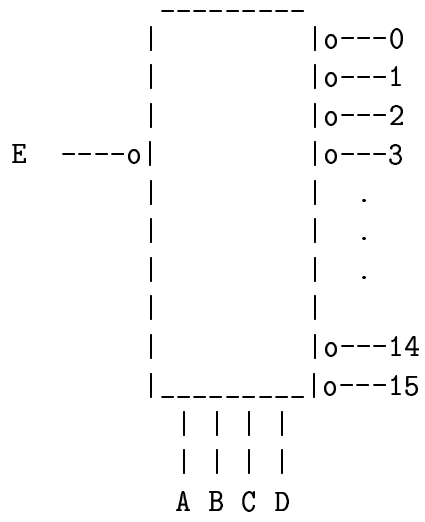


Figura 1.14

saídas são iguais a 1. O sinal E portanto habilita ($E = 0$) ou inibe ($E = 1$) o funcionamento do decodificador.

A decodificador pode ser acoplado a um multiplexador como indica a Figura 1.15.

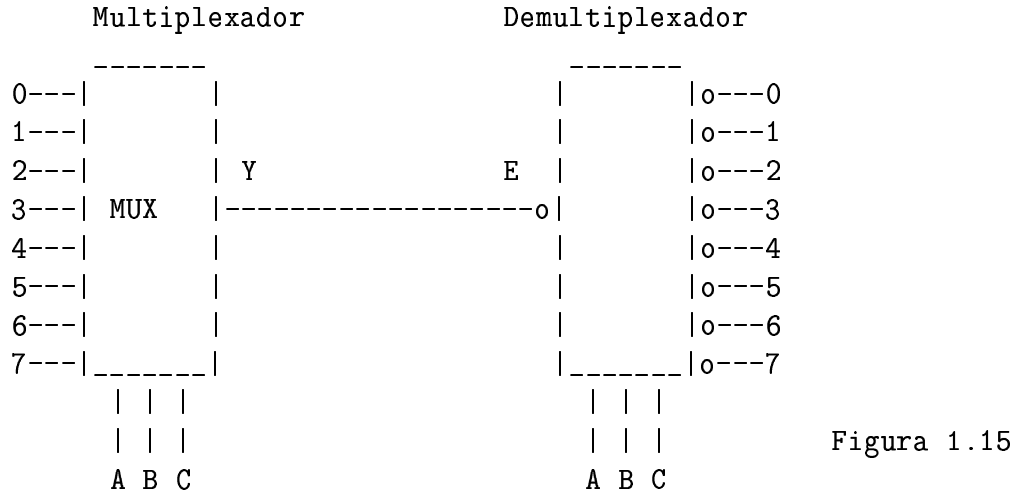


Figura 1.15

O decodificador da Figura 1.15 funciona como um demultiplexador assim: Quando ABC varia ciclicamente de 000 a 111, os sinais de entrada de 0 a 7 (do lado do multiplexador ou MUX) são transmitidos sucessivamente de Y para E, e o demultiplexador devoldve o sinal à linha de saída correspondente. Dessa maneira podemos transmitir 8 sinais usando poucas linhas. Com a repartição do tempo, ao invés de n linhas de transmissão, apenas $\log_2 n$ linhas são usadas.

ROM ou "Read-Only-Memory" e PROM ROM ("Read-Only-Memory") é uma memória cujo conteúdo é definido durante a fabricação conforme uma especificação pré-

determindada. Para uma ROM de 2^n entradas ou palavras (cada uma com k bits), n bits são usados para endereçamento. Temos então uma maneira regular de implementação de uma função combinatória de n entradas e k saídas. As n entradas formam um endereço para a ROM e as saídas são os k bits contidos na palavra correspondente àquele endereço.

O "display" dos 7 segmentos por exemplo pode ser construído de uma ROM de 10 palavras, de 7 bits cada uma. Vejam a Figura 1.16. Notem que a ROM implementa a tabela de verdade da Figura 1.3.

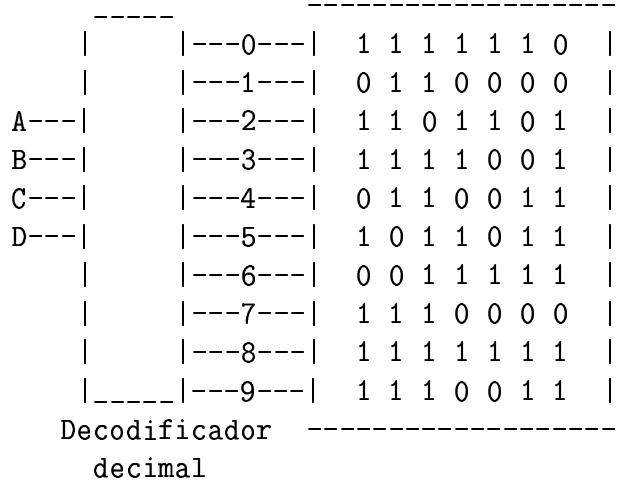


Figura 1.16

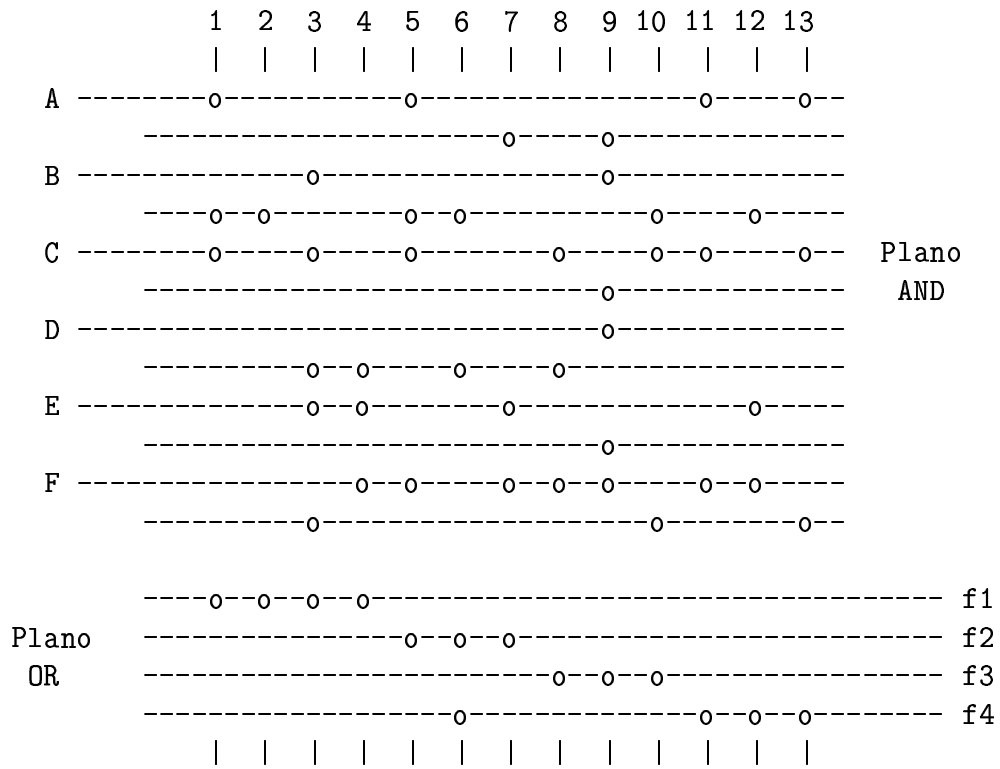
O uso principal de ROMs é nas unidades de controle de computadores. A execução de uma instrução simples, como ADD, envolve uma sequência de transferências de dados entre registradores, acesso à memória e operações lógicas. Sequências apropriadas podem ser armazenadas em ROMs. O computador neste caso diz-se *microprogramado*.

Uma PROM ("Programmable Read Only Memory") ou EPROM ("Eraseable Programmable Read Only Memory") é uma ROM programável. Seu conteúdo pode ser programado e apagado pelo usuário, através de dispositivos especiais para essa finalidade.

PLA ou "Programmable Logic Array" Um problema de usar ROM para implementar uma função lógica é que muitas vezes a maior parte das possíveis combinações de variáveis de entrada nunca ocorre. Muitas funções lógicas precisam apenas de uma pequena fração de todas as 2^n palavras, resultando assim em um desperdício de espaço.

A PLA ("Programmable Logic Array") é uma estrutura que possui a generalidade de uma memória para implementar funções lógicas, porém bem mais compacta que a ROM. A Figura 1.17 mostra um exemplo de uma PLA. Uma PLA consta de dois planos, chamados *plano AND* e *plano OR*. Numa linha vertical do plano AND, os pontos ou "bolinhas", especificados pelo cliente, indicam quais as variáveis (complementadas ou não) que devem participar de um termo de produto. O exemplo mostra uma PLA com 6 entradas e 13 termos de produto. As 13 linhas verticais formam os 13 termos de produto: $AB'C, B'D'$ etc.

Numa linha horizontal do plano OR, os pontos ou bolinhas indicam quais os produtos que devem participar numa soma (ou num circuito OR). Cada linha horizonta dá origem a uma saída. O exemplo produz assim 4 saídas.



As saídas f_1, f_2, f_3 e f_4 implementam as funções lógicas seguintes:

$$\begin{aligned} f_1 &= AB'C + B'D' + BCD'EF' + EF \\ f_2 &= AB'CF + B'D' + B'EF \\ f_3 &= CD'F + A'BC'DE'F + B'CF' \\ f_4 &= ACF + B'EF + B'D' + ACF' \end{aligned}$$

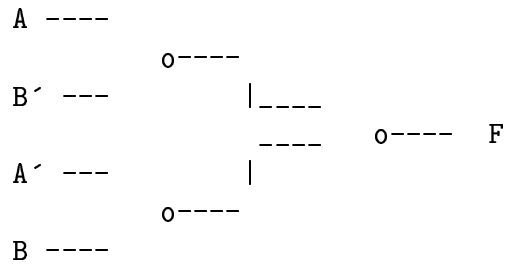
Notem que o mesmo produto $B'D'$ aparece nas funções f_1, f_2 e f_4. $B'D'$ aparece tanto na coluna 2 como na coluna 6, sendo que na coluna 6 ele é usado tanto para gerar f_2 como f_4. A PLA pode ser otimizada definindo $B'D'$ apenas uma vez.

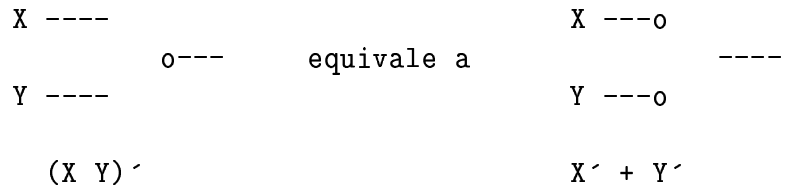
1.5 Exercícios de classe

1. Que função conhecida é implementada pelo circuito abaixo?

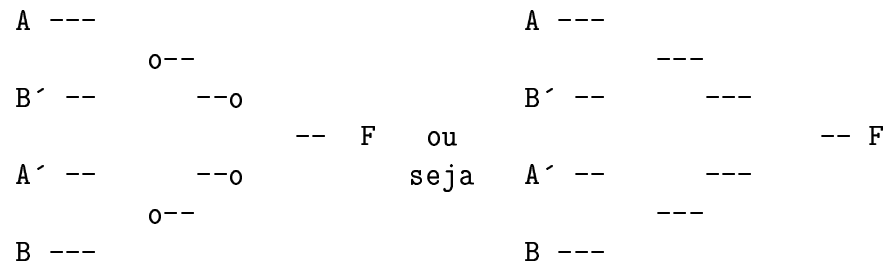
Solução :

Observe-se que pela lei de Morgan $(XY)' = X' + Y'$:





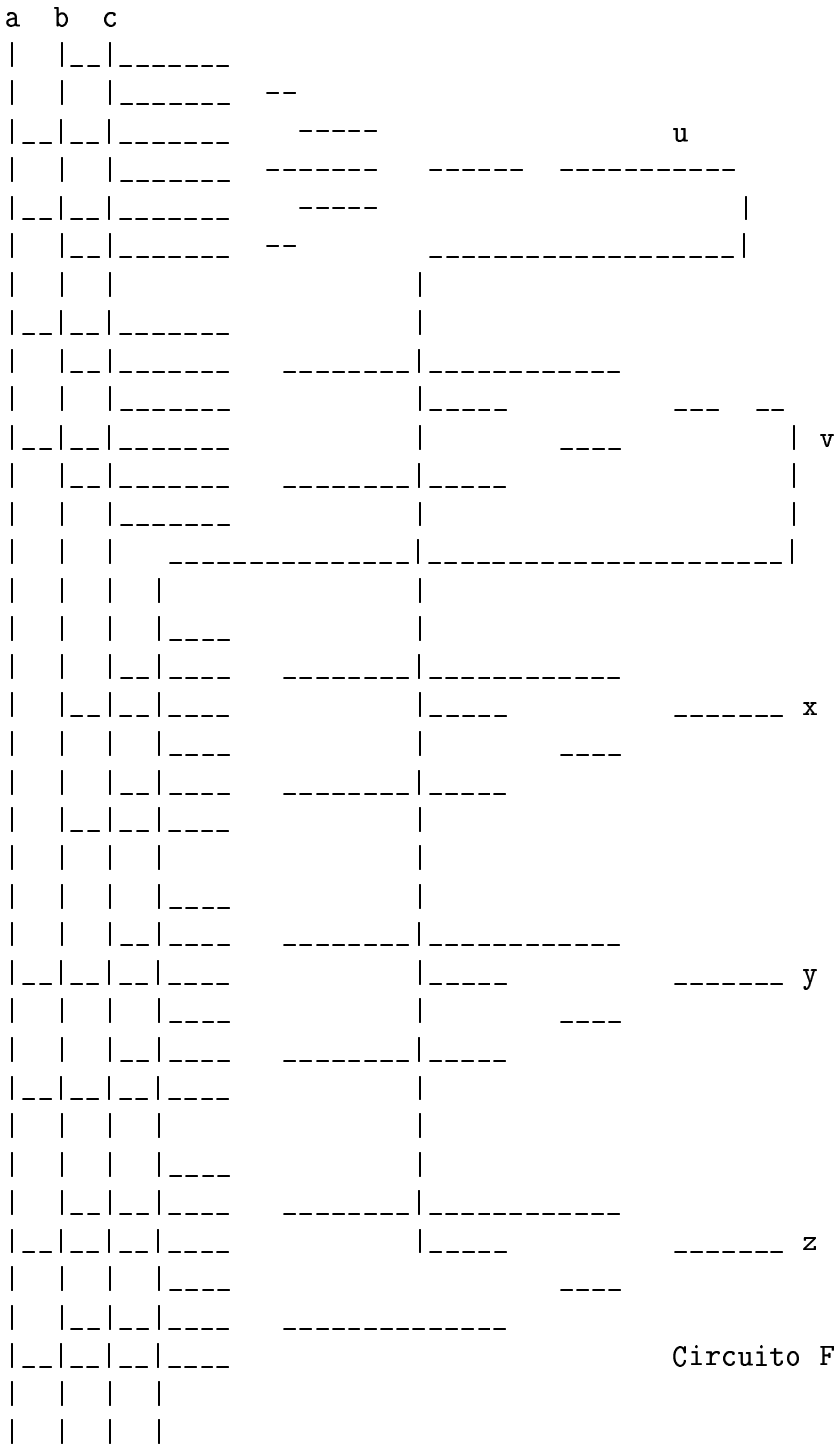
Podemos entao redesenhar o circuito somo sendo:



2. O circuito F a seguir tem 3 entradas a,b,c e 3 saídas x,y,z.

a) Descreva o comportamento de u e v como funções de a,b,c. (Sugestão: note que u e v são funções simétricas de a,b,c; isto é, elas dependem apenas de número de entradas iguais a 1).

b) Construa um circuito G que tem o mesmo comportamento de entrada-saí da como F.

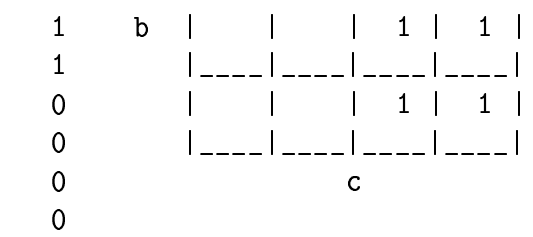

a b c
u
v
x
y
z
Circuito F

Solução a) Seja t o número de entradas iguais a 1. Podemos construir a seguinte tabela:

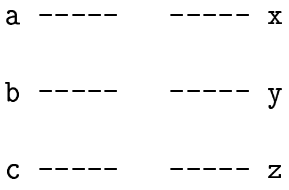
t	u	v
0	1	1
1	1	0
2	0	1
3	0	0

b) Podemos
$x = bcv + (b + c + v)u$

a	b	c	u	v	x
0	0	0	1	1	1
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	0	0

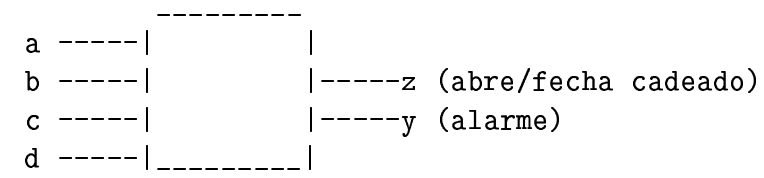


Temos portanto $x = a'$ e analogamente
$y = b', z = c'$
O circuito equivalente G é

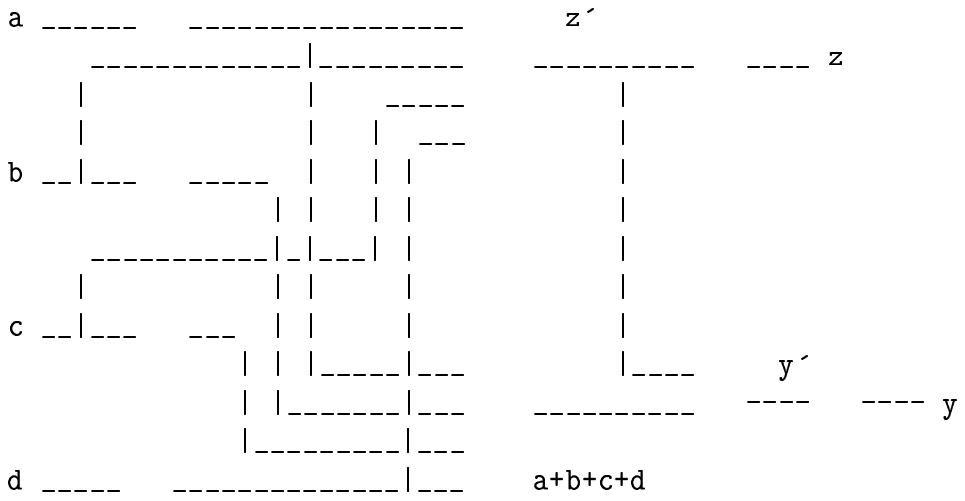


3. Projeto de um *cadeado digital*
Seja o estado de repouso $a = 0, b = 0, c = 0, d = 0$. Quando a entrada $abcd$ for igual à chave ou senha secreta (0110), então o cadeado abre (z fica igual a 1). O alarme deve ser ligado se a entrada for diferente do estado de repouso e diferente da chave secreta.
Projetar o circuito desejado, usando apenas portas NAND.

Solução Temos
chave secreta: $a = 0 \quad b = 1 \quad c = 1 \quad d = 0$



cadeado aberto se $abcd$ = chave secreta
alarme ligado se $abcd \neq$ chave secreta e $abcd \neq$ estado repouso
Assim
$z = a'bcd'$
$y = z'(a + b + c + d) = z'(a'b'c'd')'$



2 Lógica Sequencial

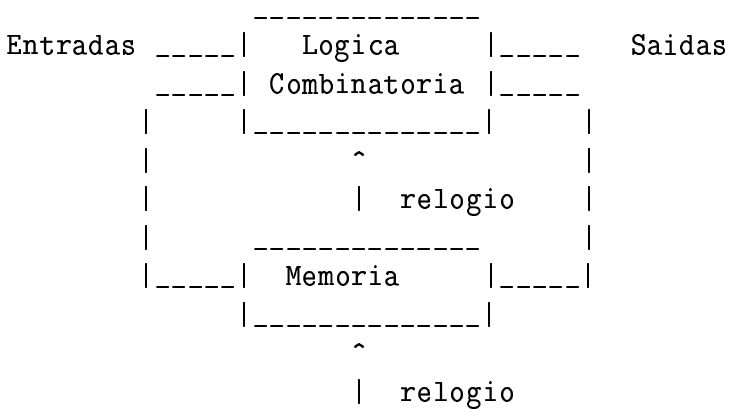
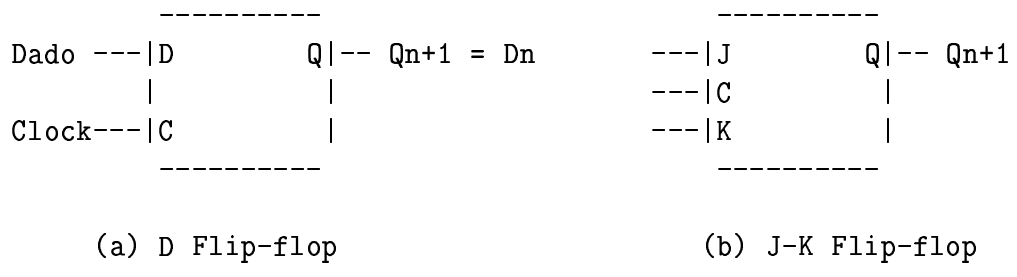


Figura 2.1

Um circuito de lógica sequencial possui elementos de armazenamento; as saídas dependem não somente das entradas presentes no momento atual mas também de valores armazenados. Ele pode ser visto como uma parte puramente combinatória juntamente com um memória, como mostra a Figura 2.1.

Vamos inicialmente examinar "flip-flops", registradores e contadores. Iremos depois apresentar um exemplo de síntese de circuitos sequenciais.

2.1 "Flip-flops"



(a) D Flip-flop

Dn	Qn+1
0	0
1	1

(b) J-K Flip-flop

Jn	Kn	Qn+1
0	0	Qn
0	1	0
1	0	1
1	1	(Qn)´

Figura 2.2

Um flip-flop é um circuito básico que armazena um bit de informação. A sáıda de um flip-flop só muda de estado durante a transição do sinal de relógio. A Figura 2.2 mostra flip-flops do tipo D e do tipo J-K.

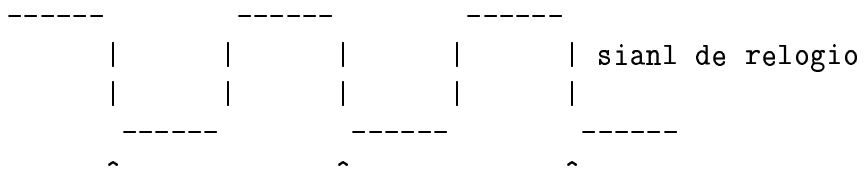


Figura 2.3

2.2 Registradores e contadores

Flip-flops podem ser agrupados para formar registradores. Registradores de deslocamento (ou "shift register") são simplesmente flip-flops ligados de tal maneira que dados são carregados de um flip-flop para o adjacente em cada ciclo do relógio. Alguns tipos comuns de registradores incluem:

- "serial-in serial-out"
- "serial-in parallel-out"
- "parallel-in parallel-out"

Vários tipos de contadores existem como circuitos MSI. A Figura 2.4 mostra um contador binário de 4 bits.

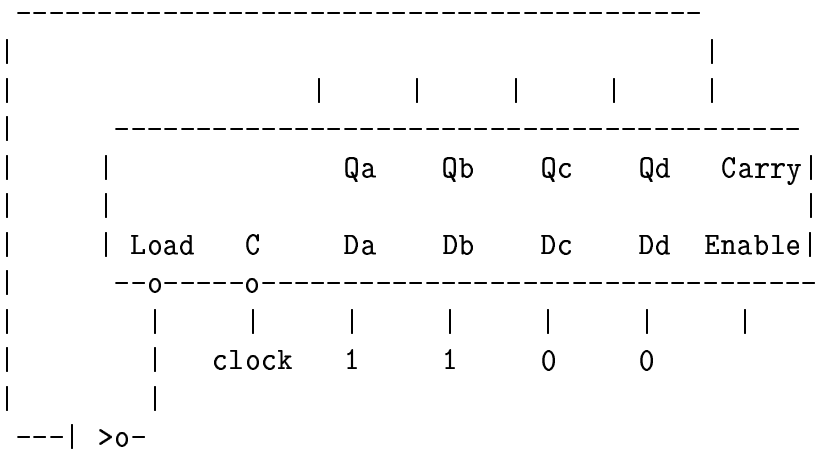


Figura 2.4

Quando Load é 0, o valor Dd Dc Db Da é carregado no contador, como seu valor inicial, no próximo ciclo do relógio. O sinal Enable, quando igual a 1, faz com que o contador seja acresentado de 1 no próximo ciclo de relógio. O sinal Carry é o sinal de "carrega" ou "vai-um" que permite construir contadores maiores. O sinal Carry fica igual a 1 quando $Qa = Qb = Qc = Qd = 1$. A Figura 2.4 mostra um contador que conta ciclicamente de 3 a 15, desde que Enable assim o comanda.

2.3 Síntese de um circuito sequencial por PLA

Vejamos um exemplo em que um circuito que controla o funcionamento de um semáforo num cruzamento. Conforme a Figura 2.5, uma rodovia expressa é intersectada por uma estradinha pouco movimentada. Um semáforo SemRod controla a rodovia e um semáforo SemEst controla a estradinha.

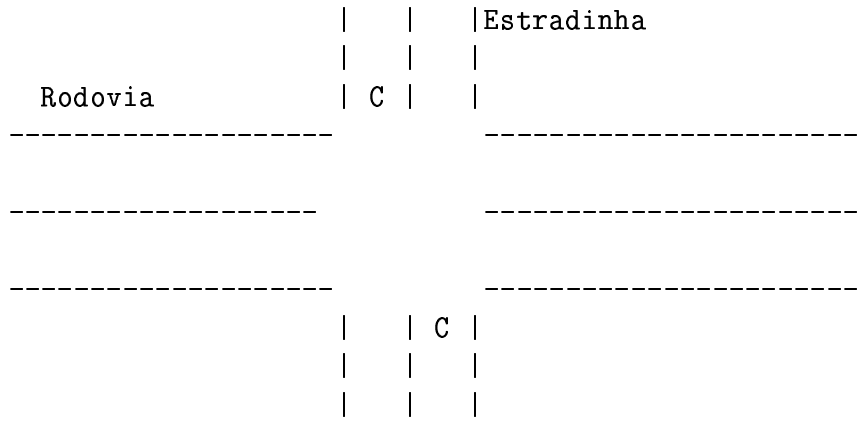


Figura 2.5

Sensores C são colocados na estradinha, em posições indicadas na figura, para detectar a presença de carros. Na ausência de carros detectados por C, o semáforo SemRod deve permanecer verde. Quando o sensor C for igual a 1 (indicando a presença de carros na estradinha), SemRod vai passar par amarelo, permanece amarelo por um tempo de duração Tcurto, e então passa para vermelho. É claro que nesse instante SemEst deve mudar para verde, abrindo o sinal para os carros da estradinha. SemEst deve permanecer verde apenas se C continua detectando passagem de carros na estradinha, mas nunca por um tempo superior a Tlongo. Passado este período Tlongo, ou C não detecta mais carros, SemEst deve mudar para amarelo (por uma duração Tcurto), depois para vermelho, quando então SemRod passa novamente para verde. SemRod não deve ser interrompido pelo tráfego na estradinha antes de ter decorrido um período de tempo igual a Tlongo.

No diagrama da Figura 2.6 temos 4 estados.

- RodVerde
- RodAmarelo
- EstVerde
- EstAmarelo

O sinal TL = 1 após a passagem de um tempo Tlongo; o sinal TC = 1 após a passagem de um tempo Tcurto. Um temporizador ou "timer" é usado para cronometrar o tempo decorrido. O "timer" é disparado pelo sinal DISPARA, começando a contar o tempo a partir de zero. O circuito a ser projetado tem as entradas C, TL e TC, e produz as

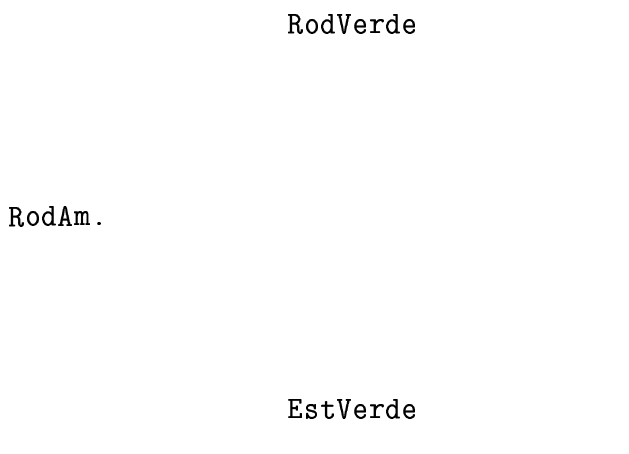


Figura 2.6

saídas DISPARA, SemRod e SemEst. Naturalmente, o estado presente e o estado próximo também participam no circuito.

A Figura 2.7 mostra a tabela de transição.

Estado presente	Entradas	Saidas SemRod	SemEst	DISPARA	Estado proximo
RodVerde	C.TL=0	Verde	Verm.	Nao	Rodverde
	C.TL=1	Verde	Verm.	Sim	RodAm
RodAm	TC=0	Am.	Verm.	Nao	RodAm
	TC=1	Am.	Verm.	Sim	EstVerde
EstVerde	C´+TL=0	Verm.	Verde	Nao	EstVerde
	C´+TL=1	Verm.	Verde	Sim	EstAm
EstAm	TC=0	Verm.	Am.	Nao	EstAm
	TC=1	Verm.	Am.	Sim	RodVerde

Figura 2.7

A Figura 2.8 mostra a mesma coisa, porém com os estados, entradas e saídas devidamente codificadas. As codificações usadas são:

Estados	RodVerde	00
	RodAm	01
	EstVerde	11
	EstAm	10

Cores	Verde	00
	Amarelo	01
	Vermelho	10

Temos ainda

- R0 R1 indicam a cor de SemRod
- E0 E1 indicam a cor de SemEst
- X0 X1 indicam o estado presente
- Y0 Y1 indicam o estado próximo

Estado presente		Entradas			Saidas					Estado proximo	
X0	X1	C	TL	TC	DISPARA	R0	R1	E0	E1	Y0	Y1
0	0	0	X	X	0	0	0	1	0	0	0
0	0	X	0	X	0	0	0	1	0	0	0
0	0	1	1	X	1	0	0	1	0	0	1
0	1	X	X	0	0	0	1	1	0	0	1
0	1	X	X	1	1	0	1	1	0	1	1
1	1	1	0	X	0	1	0	0	0	1	1
1	1	0	X	X	1	1	0	0	0	1	0
1	1	X	1	X	1	1	0	0	0	1	0
1	0	X	X	0	0	1	0	0	1	1	0
1	0	X	X	1	1	1	0	0	1	0	0

Figura 2.8

A Figura 2.9 mostra uma realização deste projeto por meio de uma PLA.

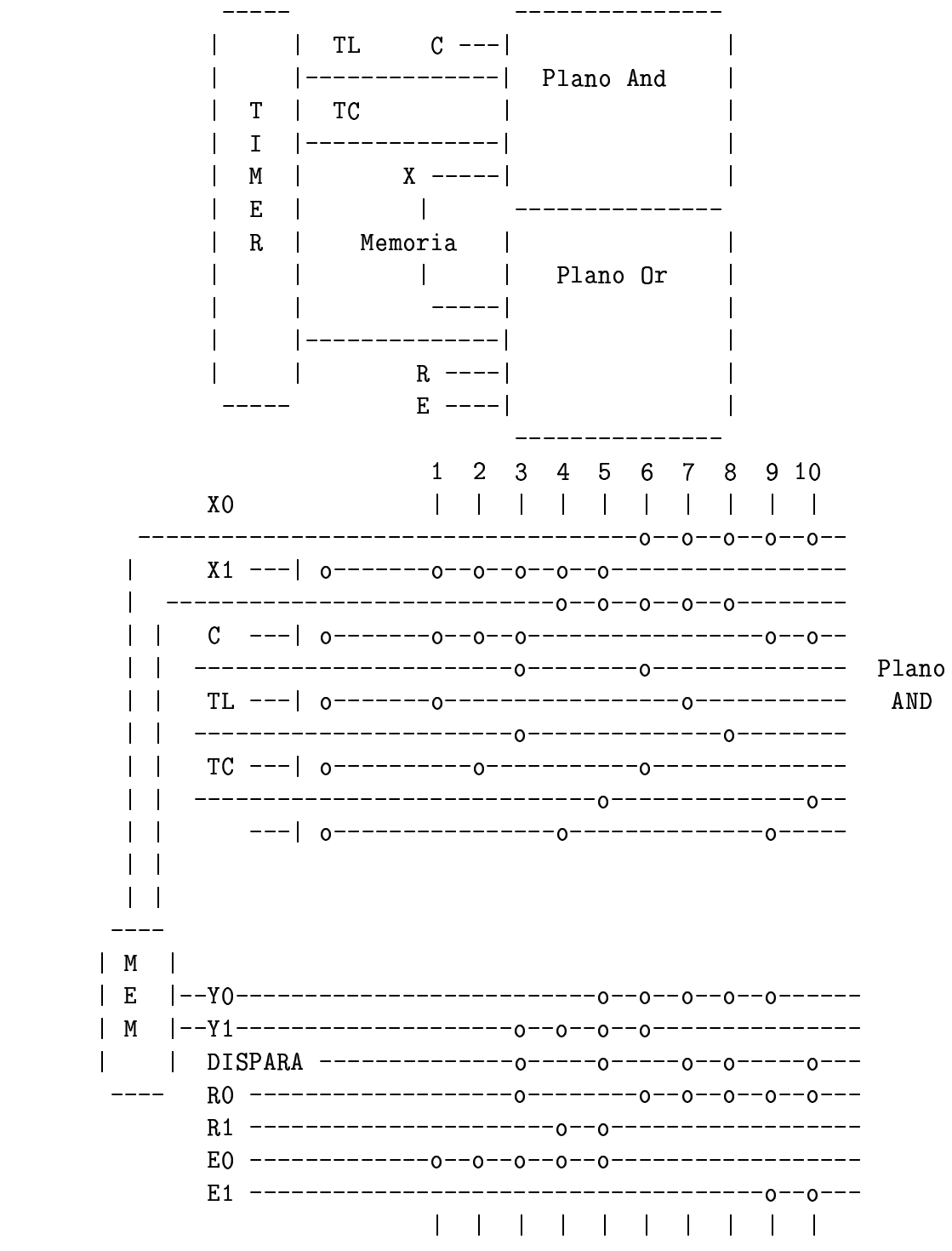

TL
C
TC
X
Memoria
R
E
Plano And
Plano Or
1 2 3 4 5 6 7 8 9 10
X0
X1
C
TL
TC
Plano
AND
M
E
M
Y0
Y1
DISPARA
R0
R1
E0
E1

Figura 2.9