UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA

Electrónica Digital

Engº. Albino B Cuinhane

AULA TEÓRICA 4
SUMÁRIO

- **Capítulo 4. Circuitos Combinatórios**
- 4.1 Expressões e circuitos a partir da tabela de verdade
- 4.2 Implementação de funções lógicas via Complemento
- **4.3** Circuitos de saídas múltiplas
- 4.4 Codificador e descodificador
- 4.5 Multiplexadores e demultiplexadores
- 4.6 Somadores
- 4.7 Comparadores

Capítulo 4

Circuitos Combinatórios

4.0. Introdução

- Foram apresentados até aqui algumas ferramentas para análise e síntese de funções lógicas. Apresentamos de seguida a primeira aplicação dos conceitos vistos até aqui: a implementação dos **Circuitos Combinatórios**
- Seja dado uma caixa em que entra algumas variáveis e saem outras como se mostra na figura 4.1

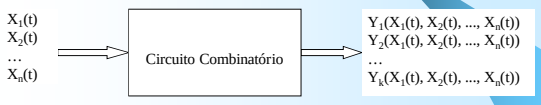


Fig.4.1 Modelo dum Circuito Combinatório.

- O circuito é denominado Circuito Combinatório se as variáveis de saída, Yi , no instante *t* dependem unicamente da combinação actual das variáveis de entrada, Xi, nesse instante.
- Desta definição infere que não importa o estado anterior tanto das variáveis de entrada como nas de saída

4.0. Introdução

- Normalmente no circuito combinatório a informação está distribuída espacialmente.
- A ***informação está distribuída espacialmente*** se uma variável lógica A só pode ser observada num dado ponto(fio) no espaço enquanto a outra B é observada noutro ponto.

Fig.4.2.a Distribuição espacial da informação.

- A ***informação está distribuída temporariamente*** se uma variável lógica A pode ser observada num dado ponto(fio) no espaço e a outra B pode ser observada no mesmo ponto, noutro instante.

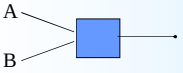
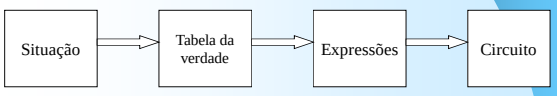


Fig.4.2.b Distribuição temporal da informação.

4.1 Expressões e Circuitos a Partir da Tabela de Verdade

4.1. Expressões e Circuito a partir da TBV

- É possível obter expressões a partir de circuitos, circuitos a partir de expressões e expressões a partir de tabela de verdade assim como circuitos a partir da tabela de verdade.
- No entanto a prática revela-nos uma dada situação e se coloca a necessidade de desenhar um circuito para resolvê-la. Então, o procedimento normal no projecto de circuitos combinatórios é esquematizado na Fig. 4.2.



- ***Fig.4.3 etapas de construção de circuitos digitais*** .

4.1. Expressões e Circuito da TBV

Duma forma geral as etapas para construir um Circuito Combinatório são:

a) Analisar a situação por forma a identificar as variáveis de entrada e as de saída

b) Em algumas situações podemos ter que dividir o problema em partes

c) Elaborar a(s) tabela(s) da verdade ilustrando todas as situações possíveis

d) Encontrar as expressões das funções de saída

e) Simplificar

f) Implementar

Consolidemos ideias com um exemplo

4.1. Expressões e Circuito da TBV

EXEMPLO.

Um circuito de alarme dum banco comporta um sensor de porta aberta que detecta a porta aberta, um botão de emergência para provocar alarme sonoro em caso de emergência e um botão de activação que arma ou desarma o sistema. Sempre que estiver armado, e for fora de expediente e houver violação da porta, o alarme sonoro soara. Sempre que pressionar o botão de emergência, o alarme soará. Desenhar o circuito que produz o alarme sonoro.

4.1. Expressões e Circuito da TBV

a) Pela exposição facilmente vemos que o sinal de saída é o alarme sonoro. Designemos por S a esta variável que acaba sendo a função lógica procurada. Por outro lado vemos que o sistema é excitado por:

· Um sensor de porta aberta que denominaremos por P. Consideremos que enquanto aberta a porta, a variável P assume o valor lógico 1.

· O botão de emergência que designaremos por E. Consideremos que enquanto pressionado o botão, a variável E assume o valor lógico 1

· O botão de armação que designaremos por A. Consideremos que enquanto armado o sistema, a variável A assume o valor lógico 1

b) O problema não nos parece tão complexo que mereça ser desdobrado em problemas pequenos.

4.1. Expressões e Circuito da TBV

c) Construamos a tabela de verdade (Fig. 4.3) onde vamos colocar todas as possibilidades e observarmos o comportamento da função. Lembramos que na linha correspondente à combinação onde a função existe colocamos 1 na coluna de S(A,E,P) .

AEP	S(A,E,P)
000	0
001	0
010	1
011	1
100	0
101	1
110	1
111	1

Tab. 4.1. tabela de verdade do circuito de alarme

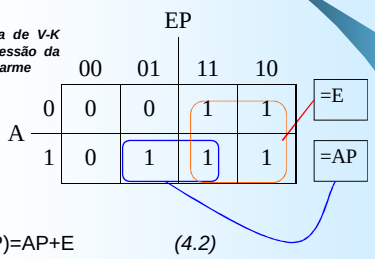
d) Da tabela de verdade obtemos a expressão de S(A,E,P) através da colecta dos termos produtos onde a função assume o valor lógico 1:

$$S(A,E,P) = \overline{A}E\overline{P} + \overline{A}EP + A\overline{E}P + AE\overline{P} + AEP \quad (4.1)$$

4.1. Expressões e Circuito da TBV

e) A expressão obtida da tabela de verdade pode não ser a mais simplifica. Então recorremos a qualquer dos métodos de simplificação para reduzi-la à forma mais simples.

Fig.4.4 Mapa de V-K para a expressão da função do alarme



S(A,E,P)=AP+E *(4.2)*

4.1. Expressões e Circuito da TBV

f) Vamos ao mercado para vermos quais as possibilidades que se nos oferecem para implementar o circuito da função(ou funções) obtida(s) em e). Se tivermos disponíveis todas as portas lógicas definidas pela função implementamo-la directamente. Caso contrário podemos ter que efectuar algumas modificações para ajustar às condições presentes. Uma destas técnicas é mostrada nos sub-capítulos 4.2 e 4.3 a seguir.

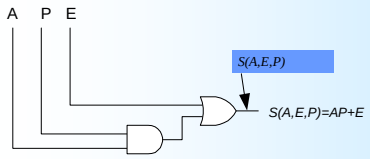


Fig.4.5. Implementação do circuito de alarme

4.2 Implementação de Funções Lógicas via Complemento

4.2 Implementação de Funções Lógicas via ~F

Em algumas ocasiões podemos depararmo-nos com uma função lógica cuja implementação consome muitos recursos. Suponha a função lógica representada pelo Mapa de Karnaugh seguinte:

AB \ CD	00	01	11	10
00	X	0	1	1
01	1	0	1	1
11	1	1	0	0
10	X	1	1	X

Fig.4.6 Mapa de V-K para uma função hipotética

Após a simplificação fica:

$$F(A,B,C,D) = \overline{A}C + A\overline{C} + A\overline{B} + \overline{C}\,\overline{D} \qquad (4.3)$$

4.2 Implementação de Funções Lógicas via ~F

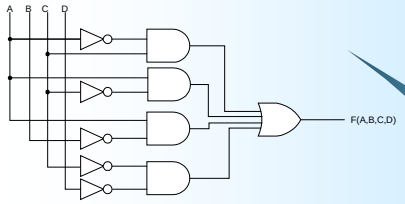


Fig.4.7 Circuito gerado através da expressão (4.3) .

Esta implementação da função F mesmo simplificada consome 27 pinos de circuitos integrados, onde estão encapsuladas as portas

4.2 Implementação de Funções Lógicas via ~F

Observando o mapa de V-K, donde tiramos a expressão final de F, vemos que a parte onde a função não existe ocupa menos espaço. E mais importante ainda, as células são adjacentes, o que nos permite efectuar simplificação.

Então encontremos a função que é dada por $\overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D} + ABCD + ABC\overline{D}$ que simplificada resulta na expressão:

$$\overline{F}(A,B,C,D) = \overline{A}\overline{C}D + ABC \qquad (4.4)$$

E que implementada resulta no circuito da Fig. 4.8

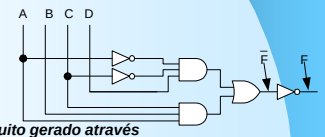


Fig.4.8 Circuito gerado através da expressão (4.4) .

4.2 Implementação de Funções Lógicas via ~F

A Fig. 4.8 mostra a implementação da função (4.3) através do seu complementar em (4.4). Esta implementação consome desta vez 17 pinos de circuitos integrados. Isto representa um ganho económico.

A implementação das funções via complemento é também útil nos casos em que não temos portas com saídas no modo afirmativo. Na prática a maior parte dos circuitos lógicos encapsulados nos circuitos integrados têm as saídas na forma negada em virtude de as portas básicas serem NAND e NOR

Seja dada a função (4.4) e se tivermos que implementá-la apenas com portas NAND e NOR devemos proceder do seguinte modo.

4.2 Implementação de Funções Lógicas via ~F

1. Assumindo que já não temos OR para tirar dela a nossa função, acabamos percebendo que ela sairá duma saída negada.
2. Como não queremos implementar a função onde ela não existe, então neguemos a saída negada para voltarmos à origem.
3. Depois aplicamos DeMorgan até vermos as portas NAND e NOR a produzirem a função.
4. As inversoras são facilmente produzidas pelas NAND e NOR, bastando ligar todas as entradas à variável a negar (com efeito se aplicar DeMorgam pode demostrar facilmente isto)

Acompanhemos a seguir as transformações indicadas:

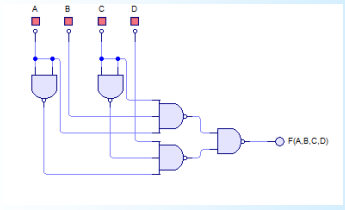
$$\overline{\overline{A}CD} + ABC = \overline{\overline{\overline{A}CD} + ABC} \qquad (4.5)$$

4.2 Implementação de Funções Lógicas via ~F

Aplicando DeMorgan ao membro direito da expressão (4.5) vem:

$$\overline{\overline{A}CD} + ABC = \overline{\overline{A}CD}.\overline{ABC} \qquad (4.6)$$

Donde se vê que há duas NAND que operam as variáveis na origem e outra NAND que opera as outras NAND.
Contudo é preciso produzir antes o complementar de A e de C. Finalmente temos:



4.3 Circuitos de Saídas Múltiplas

4.3 Circuitos de Saídas Múltiplas

Durante a implementação de projectos de sistemas electrónicos surgem muitos casos a necessidade de produzir um sinal que é usado em várias etapas do circuito.

Para ganhar tempo e recursos, e simplificar o circuito final, é prático produzir uma vez o sinal e derivá-lo para todos os locais onde é necessário.

Já imaginou se de cada vez que precisasse de 12V num ponto do circuito instalasse uma fonte de alimentação isolada!

Nos circuitos lógicos também pode-se aplicar este princípio salvaguardando no entanto algumas limitações, como fan-out (nivel de corrente à saída) e fan-in(quantidade de entradas), que serão estudados no tema sobre circuitos lógicos integrados.

A tabela de verdade a seguir sugere um projecto em que são produzidas 3 funções lógicas F1, F2 e F3.

ABCD	F1	F2	F3
0000	1		
0001	1		
0010	1		
0011	1		
0100		1	
0101		1	
0110			
0111		1	
1000	1		1
1001			
1010			1
1011			1
1100			1
1101			
1110			
1111			1

4.3 Circuitos de Saídas Múltiplas

$$F1(A,B,C,D) = \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + \bar{A}BCD + A\bar{B}C\bar{D}$$

$$F2(A,B,C,D) = \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD$$

$$F3(A,B,C,D) = \bar{A}B\bar{C}\bar{D} + A\bar{B}C\bar{D} + \bar{A}BCD + AB\bar{C}\bar{D} + ABCD$$

Fig.4.9 tabela de verdade de três funções hipotéticas.

4.3 Circuitos de Saídas Múltiplas

Simplificando e implementando separadamente as três funções custa 71 pinos de circuitos integrados(!), como pode ser visto na Fig. 4.10

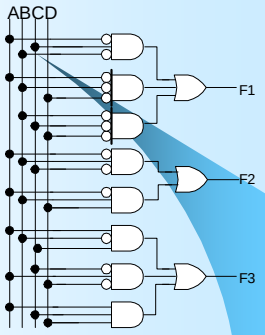


Fig.4.10 Implementação independente das funções F1, F2 e F3

4.3 Circuitos de Saídas Múltiplas

Porém se implementarmos primeiramente as partes comuns e re-aproveitar os sinais, conseguimos uma redução de pinos para 54! Como ilustra a Fig. 4.11.

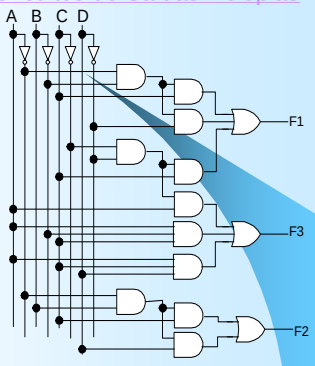


Fig.4.11 Implementação das funções F1, F2 e F3 com partilha de sinais

4.3 Circuitos de Saídas Múltiplas

Embora a Fig. 4.11 represente uma economia em recursos pode trazer desvantagens. A Fig. 4.10 mostra que para uma função ser produzida são necessários 3 níveis de decisão. O 1º corresponde às portas NOT, o 2º às portas AND e o 3º à porta OR

No entanto na Fig. 4.11 cada função F requer 4 níveis de decisão: As portas NOT, as primeiras AND´s, as segundas AND´s e finalmente a OR

O aumento dos níveis de decisão tem implicações no tempo de processamento. O tempo que cada porta leva a processar as entradas até apresentar o resultado (denominado tempo de atraso de propagação) influi no tempo final.

Esse atraso tem influência na frequência máxima com que o circuito pode trabalhar

4.4 Codificador e Descodificador

4.4 Codificador e descodificador

Observe o esquema da Fig. 4.12. Dois indivíduos falam linguas diferentes e comunicam-se com ajuda dum tradutor

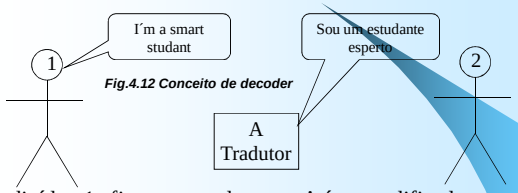


Fig.4.12 Conceito de decoder

O indivíduo 1 afirma que o elemento A é um codificador.

O indivíduo 2 afirma que o elemento A é um descodificador.

Os dois discutiram o dia todo sem chegar a um consenso

4.4 Codificador e descodificador

Mas no final constataram que os dois tinham razão!

É que o conceito de codificador/descodificador é referencial.

CODIFICADOR é o dispositivo que transforma um código conhecido num desconhecido

DESCODIFICADOR é o dispositivo que transforma um código desconhecido num conhecido

Nos circuitos digitais define-se **Codificador** um circuito que converte um conjunto de 2^n bits num código de n bits.

Nos circuitos digitais define-se **Descodificador** um circuito que converte um código de n bits em 2^n linhas activas. O decodificador gera minitermos. Mas atenção que o decodificador não precisa de produzir na saída todos os minitermos

4.4 Codificador e descodificador

No entanto é comum não distinguir os dois circuitos chamando-os apenas de descodificador como sendo um circuito que converte um código noutro

Etapas para o desenvolvimento dum descodificador

A construção do descodificador segue a seguintes etapas:

a) Determinar quantos bits são necessários para realizar o código original e o final

b) Elaborar uma tabela de correspondência, linha a linha, entre o código original e o final

c) Considerar cada bit do código final como uma função de todas as variáveis do código de entrada

d) Aplicar os conceito de circuitos de saídas multiplas e implementar cada uma das funções identificadas em c)

4.4 Codificador e descodificador

EXEMPLO 4.1

Projectar um descodificador que converte o código BCD8421 no codigo decimal

a) Na entrada temos 4 bit para codificação BCD8421. Na saída precisamos de 10 variáveis para representar cada um dígitos decimais

b) Elaborar a tbv

A	B	C	D	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0
1	0	1	0	X	X	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X

Tab.4.1. Correspon-dência linha a linha entre os dois códigos

4.4 Codificador e descodificador

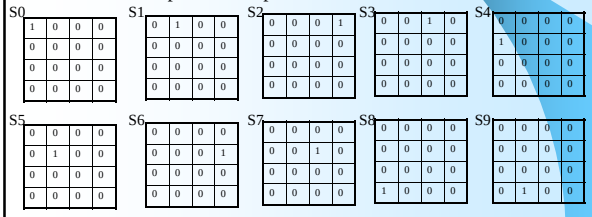
c) As expressões das funções S_i são:

$$S_0 = \overline{A}\overline{B}\overline{C}\overline{D} \quad S_1 = \overline{A}\overline{B}\overline{C}D \quad S_2 = \overline{A}\overline{B}C\overline{D} \quad S_3 = \overline{A}\overline{B}CD \quad S_4 = \overline{A}B\overline{C}\overline{D}$$

$$S_5 = \overline{A}B\overline{C}D \quad S_6 = \overline{A}BC\overline{D} \quad S_7 = \overline{A}BCD \quad S_8 = A\overline{B}\overline{C}\overline{D} \quad S_9 = A\overline{B}\overline{C}D \qquad (4.5)$$

d) Implemantação

Prenchemos os mapas de V-K para melhor visualizar



4.4 Codificador e descodificador

d1) Implementação directa das expressões das funções Si:

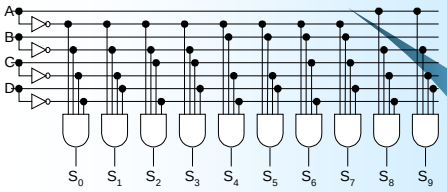


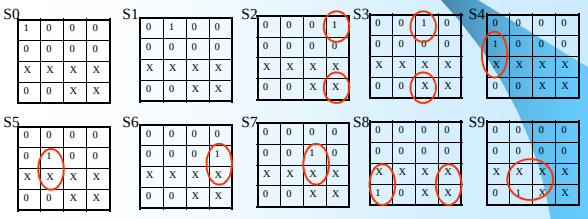
Fig.4.13. Descodificador BCD para decimal com implementação directa

4.4 Codificador e descodificador

d2) Implementação com *don´t care*

A implementação anterior pode não nos agradar por não ser a mais económica.

Considerando as combinações acima de 9 como x obtemos e aproveitando-as para a simplificação temos:



4.4 Codificador e descodificador

E as expressões das funções Si ficam:

$$S_0 = \overline{A}\overline{B}\overline{C}\overline{D} \quad S_1 = \overline{A}\overline{B}\overline{C}D \quad S_2 = \overline{B}\overline{C}\overline{D} \quad S_3 = \overline{B}CD \quad S_4 = B\overline{C}\overline{D}$$

$$S_5 = B\overline{C}D \quad S_6 = BC\overline{D} \quad S_7 = BCD \quad S_8 = A\overline{D} \quad S_9 = AD$$

Que implementadas ficam

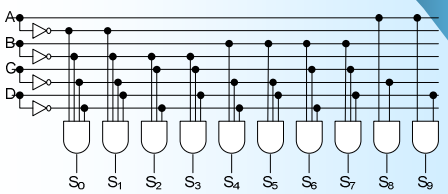


Fig.4.14. Descodificador BCD para decimal incorporando os X

4.4 Codificador e descodificador

PROBLEMAS CONSTATADOS

1º PROBLEMA: ESTADOS FALSOS

Embora o circuito da fig 4.14 seja mais económico tem um problema grave: É que basta que A=1 e D=1 para que S9 seja activo. Pois então, o S9 ficará activo mesmo que o código à entrada seja 1111. E esta combinação não corresponde a 9!

Para evitar os estados falsos deve-se incluir na simplificação apenas os termos em que a função realmente existe.

2º PROBLEMA: RUIDO

Na transição, por exemplo, de 0111 para 1000 sucede que as 4 variáveis nunca mudam ao mesmo tempo. Suponha que muda primeiro a D. Nessa altura teremos a combinação 0110 que é 6! O descodificador irá mostrar momentaneamente esse 6.

Para evitar o ruido introduz-se à entrada ou à saída um sinal chamado STROB como se mostra na Fig. 4.15. Este sinal inabilita as entradas ou saídas até que atinjam o equilíbrio.

4.4 Codificador e descodificador

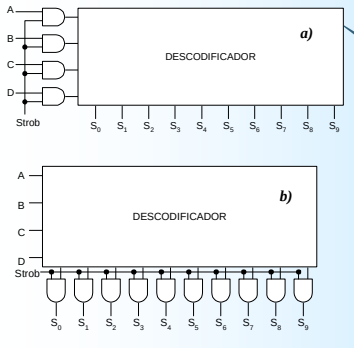


Fig. 4.15 colocação do Strob
a) À entrada
b) À saída

4.5 Multiplexadores e demultiplexadores

4.5 Multiplexadores e demultiplexadores

Multiplex é um circuito composto por várias entradas todas concorrentes para uma mesma saída.

Para que a informação das diversas entradas não entre em conflito na saída, o mux tem ainda as entradas de seleção, que escolhem qual entrada passa, pela única saída, em cada instante.

Conclui-se daqui que a informação é distribuida temporariamente na saída do mux

O elemento chave no mux é o gerador de produtos canónicos ao qual se associa uma porta OR para encaminhar as entradas numa única saída.

Seja dada uma porta AND de duas entradas, em que uma está ligada a C_0 e outra à P_0

4.5 Multiplexadores e demultiplexadores

Fig. 4.16

Se P_0 =0 a saída S será sempre 0 independente de C_0.

Se P_0 =1 a saída S depende do valor de C_0 . Deste modo conseguimos um circuito que pode vedar ou desvendar a passagem do sinal C_0. Ao sinal C_0 chama-se ***Canal*** e P_0 ***Linha de Seleção***

Suponha agora que tem 4 canais por controlar. Pretende que alcancem um a um a saída S. Então montaria o circuito da Fig. 4. 17.

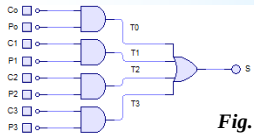


Fig. 4.17

Se colocarmos P0 em High enquanto P_1 , P_2 e P_3 em Low, observaremos Low em T_1 , T_2 e T_3 , de certeza.

4.5 Multiplexadores e demultiplexadores

Já em T_0 observaremos 0 se C_0 for Low e 1 se C_0 for High. Como o 0 é neutro na soma, o que observarmos em S saberemos que corresponde ao que existe em T_0

Do mesmo modo se activarmos em 1 apenas P_1 o que for observado em S corresponde à C_1

O perigo reside na possibilidade de activar mais do que um P_i, pois se tal for, não saberemos se o valor em S pertence à que porta aberta pela linha de seleção

A chave para resolver este problema é produzir as linhas de seleção através dum Gerador de Produtos Canônicos (GPC). A Fig. 4.18 mostra um gpc de 4 produtos com o qual pode-se construir um mux de 4 canais de entrada e 1 de saída: MUX 4x1

4.5 Multiplexadores e demultiplexadores

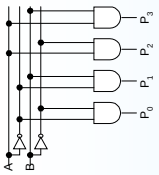


Fig. 4.18 Gerador de produtos canónicos

Para cada combinação das variáveis A e B apenas uma porta AND estará com a saída em 1.
Na verdade o gpc é um descodifcador n para 2^n

Quando A=B=0 as portas P_1 a P_3 terão pelo menos uma entrada em Low e consequentemente a saída irá para Low. Apenas P_0 vai ter à entrada a combinação que permite que vá ao estado 1.

Usemos o GPC para construir o circuito da Fig. 4.17. Os sinais C_i só serão repetidos na saída S quando o respectivo produto canônico for igual a 1. Desta forma construímos um Mux 4x1

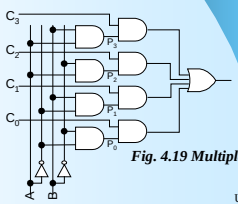


Fig. 4.19 Multiplex 4x1

4.5 Multiplexadores e demultiplexadores

O circuito Multiplex pode ser abreviado como mostra a figura 4.20 ao lado, representando uma economia de portas.

O símbolo do Mux está na Fig. 4.21. As entradas AB chamam-se entradas de Seleção.

As entradas C_i chamam-se canais

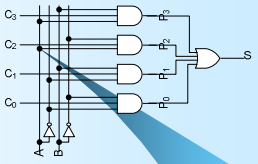


Fig. 4.20 Mux de 4 canais de 1 bit

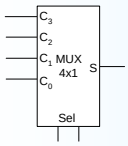
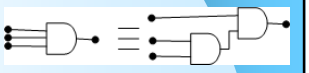


Fig. 4.21 Símbolo do MUX 4x1



4.5 Multiplexadores e demultiplexadores

DeMultiplex é um circuito composto por uma entrada que é orientada para várias saídas a que se chamam canais.

Este circuito faz exactamente o inverso do Mux. Uma vez os dados convertidos pelo Mux para caminharem numa linha, o DeMux faz a recuperação dos sinais e os distribui pelos canais respectivos.

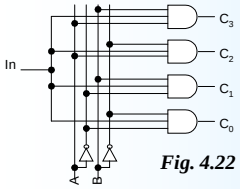


Fig. 4.22

O sinal em ***In*** será bloqueado em todos os canais cujo produto canônico seja nulo.

C_i imitará ***In*** se as respectivas entrada de seleção estiverem em 1.

4.6 Somadores e Subtractores

4.6 Somadores e subtractores

Um circuito muito utilizado nos circuito digitais é o somador e subtractor. O somador é usado para realizar a operação de adição de numero binários. O circuito mais simples para somar é o meio-somador

O meio somador apenas opera 2 bits e gera o bit da soma e o "vai-1". Assim a tabela de verdade do meio somador é:

AB	S	Co
00	0	0
01	1	0
10	1	0
11	0	1

Onde A é um dos bits a somar e B o outro; S é a soma dos dois bits e Co o "vai-1".

Da tabela vemos facilmente que S=A⊕B e Co=AB

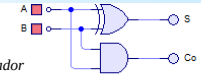


Fig. 4.23 Meio-somador

4.6 Somadores e subtractores

O meio somador não é prático quando se trata de somar números com mais de 1 bit cada, pela simples razão de que ao realizamos a soma de 2 bits temos que estar atentos ao Co da soma dos bits menos significativos.

O somador completo tem, pois, este cuidado e a sua tabela de verdade é:

C_iAB	S	Co
000	0	0
001	1	0
010	1	0
011	0	1
100	1	0
101	0	1
110	0	1
111	1	1

Onde C_i é o C_o vindo de fora ("vem-1"). Da tabela vem $S=C_i \oplus A \oplus B$ e $C_o=AB+C_iB+C_iA$

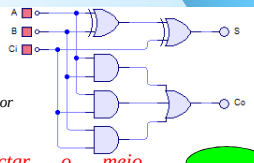


Fig. 4.24 Somador completo

TPC: projectar o meio subtrator e subtrator completo

4.7 Comparadores

4.7 Comparadores

Em diversas ocasiões há necessidade de comparar números para determinar qual deles é maior. Da mesma forma que nos somadores, nos comparadores podemos encontrar o meio-comparador e o comparador completo.

O primeiro compara dois bits e sinaliza qual deles é maior e quando são iguais. Sejam A e B os bits a comparar; M o sinal de saída que indica que A>B e I para indicar que A=B. A tbv fica:

AB	M	I
00	0	1
01	0	0
10	1	0
11	0	1

Donde tiramos as expressões de M e I assim:

$$M = A\overline{B}$$

$$I = A \otimes B$$

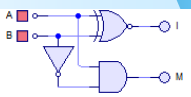


Fig. 4.25 Meio-comparador

4.7 Comparadores

Da mesma forma que no somador, em praticamente todas as ocasiões nos deparamos com a comparação de números com mais de 1 bit cada. Neste caso, ao fazer a comparação temos que observar primeiro se a comparação dos bits mais significativos não decidiu já qual dos números é maior.

Assim o comparador completo fica com a tbv assim:

M_iI_iAB	M	I
0000	0	0
0001	0	0
0010	0	0
0011	0	0
0100	0	1
0101	0	0
0110	1	0
0111	0	1

M_iI_iAB	M	I
1000	1	0
1001	1	0
1010	1	0
1011	1	0
1100	X	X
1101	X	X
1110	X	X
1111	X	X

Onde M_i e I_i é a indicação, respectivamente, de que a comparação anterior é **M**aior ou **I**gual

4.7 Comparadores

Donde extraímos as expressões de M e I:

$$M = M_i + I_i A\overline{B}$$

$$I = I_i A \otimes B$$

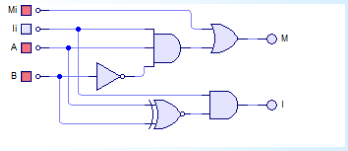


Fig. 4.26 comparador

TPC: porque X nas combinações finais na tbv?