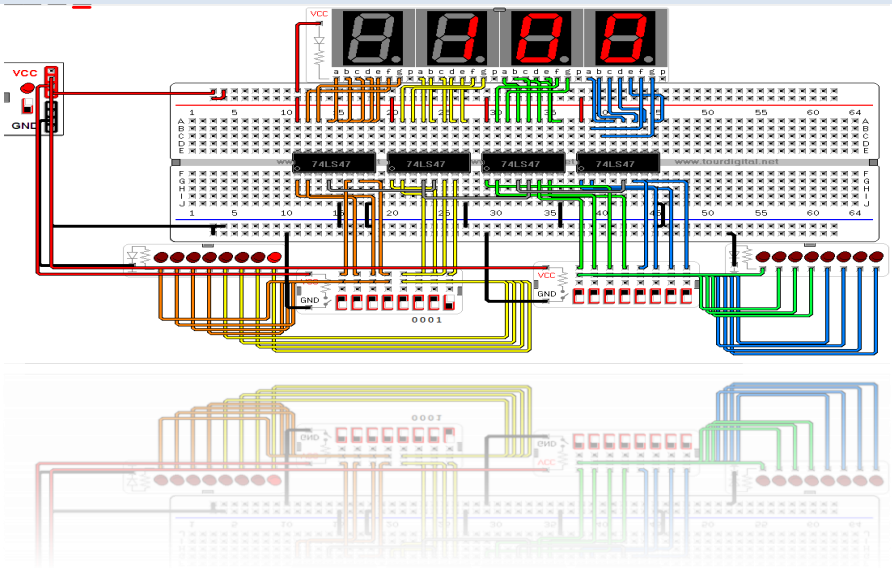
CIRCUITOS LÓGICOS COMBINACIONAIS MULTITERMINAIS



Docente: Engº Ernesto M.B. António

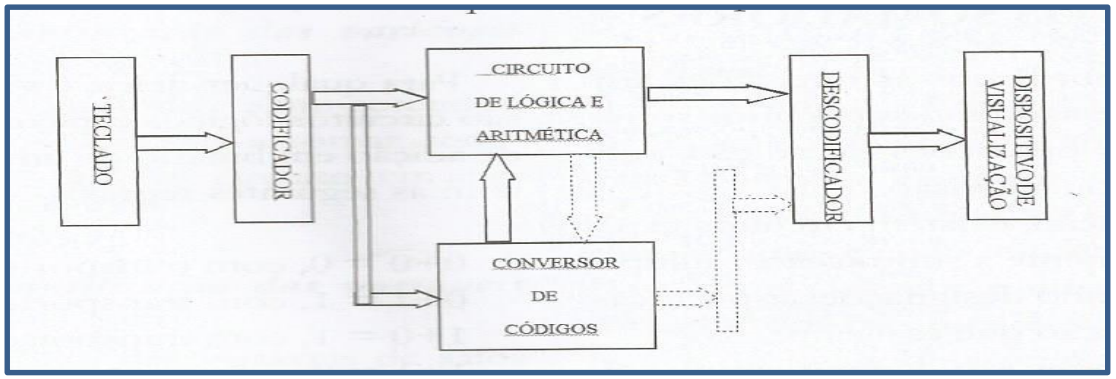
INTRODUÇÃO

Circuitos Lógicos Combinacionais Multiterminais – são circuitos lógicos combinacionais, dotados de várias saídas que, apresentam em cada momento, valores relacionados entre si para formarem uma entidade. São mais complexos, relativamente aos uniterminais, apresentando um modo de funcionameto mais estruturado.

Classificação:

- Circuitos somadores
- Circuitos codificadores
- Circuitos conversores de códigos
- Circuitos descodificadores

Estes 4 tipos de circuitos podem construir um sistema digital interessante, cujo diagrama em blocos apresentamos a seguir:



-O teclado – é um dispositivo periférico, que nos permite introduzir os dados alfa-numéricos a serem processados pelo sistema lógico.

-O codificador – converte a representação alfa-numérica introduzida através do teclado para o código binário.

-O circuito de lógica e aritmética (que pode comportar os circuitos somadores) – processa esses dados.

-O circuito descodificador – transforma o código binário em caracteres alfa-numéricos ou grupo de padrões reconhecíveis pelo operador humano.

-O dispositivo de visualização – permite que o operador humano vizualize o resultado do processamento elaborado.

-Conversor de códigos – transforma sequências de códigos, num outro código em representação binária, quando necessário.

1.1. OS CIRCUITOS SOMADORES

Circuitos somadores – são circuitos lógicos capazes de realizar a operação de adição em binário de um ou mais bits de acordo com as seguintes regras:

- 0+0=0, com transporte 0;
- 0+1=1, com transporte 0;
- 1+0=1, com transporte 0;
- 1+1=0, com transporte 1;
- 1+1+1=1, com transporte 1;

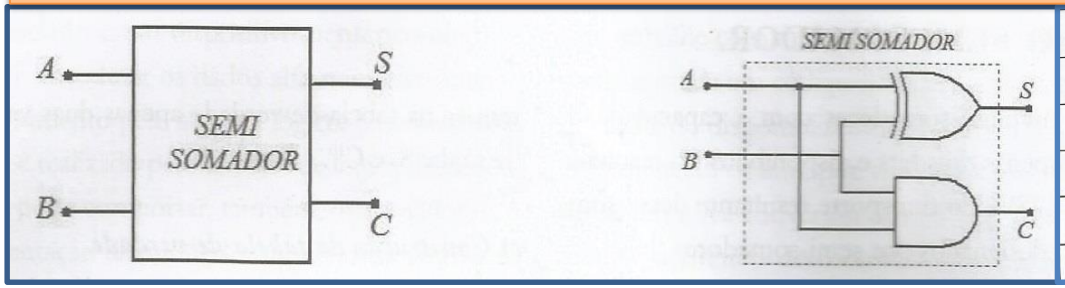
Classificação:

- ❑Semi-somador
- ❑Somador completo
- ❑Somador-paralelo
- ❑Somador-subtrator
- ❑Somador de DCB-8421

I.1.1. O SEMI-SOMADOR

Semi-somadores – são circuitos somadores com a capacidade de operar apenas dois bits e disponibilizar o resultado da soma (S) e o transporte resultante dessa soma (C).

Símbolo, Esquema e Tabela-de-verdade do semi-somador



Entradas		Saídas	
A_0	B_0	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Expressão booleana do semi-somador

$$S = A_0 \oplus B_0;$$
$$C = A_0 \cdot B_0.$$

Por exemplo:

a) $(1+1)_{10}$:

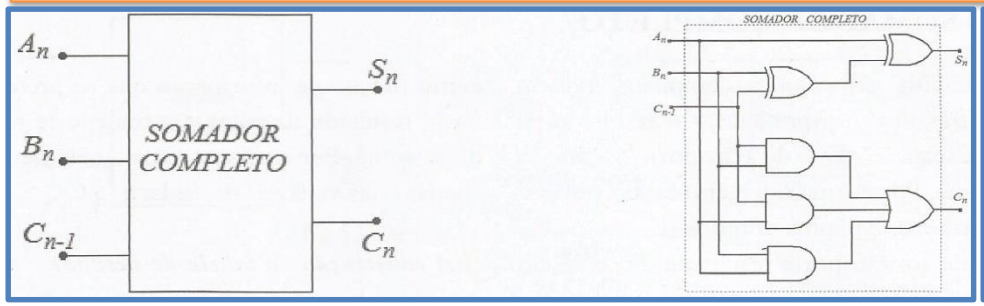
$(1)_{10} = (1)_2$; Logo: $S = 1 \oplus 1 = 0$ e $C = 1.1 = 1$.

Assim o resultado C=1 e S=0 (nesta ordem) corresponde a 2, em decimal.

I.1.2. O SOMADOR COMPLETO

Somadores completos – são circuitos somadores com a capacidade de operar três bits e disponibilizar o resultado da soma (S_n) e o transporte resultante dessa soma (C_n). Por norma o terceiro bit corresponde ao transporte da soma anterior (C_{n-1}).

Símbolo, Esquema e Tabela-de-verdade do somador completo

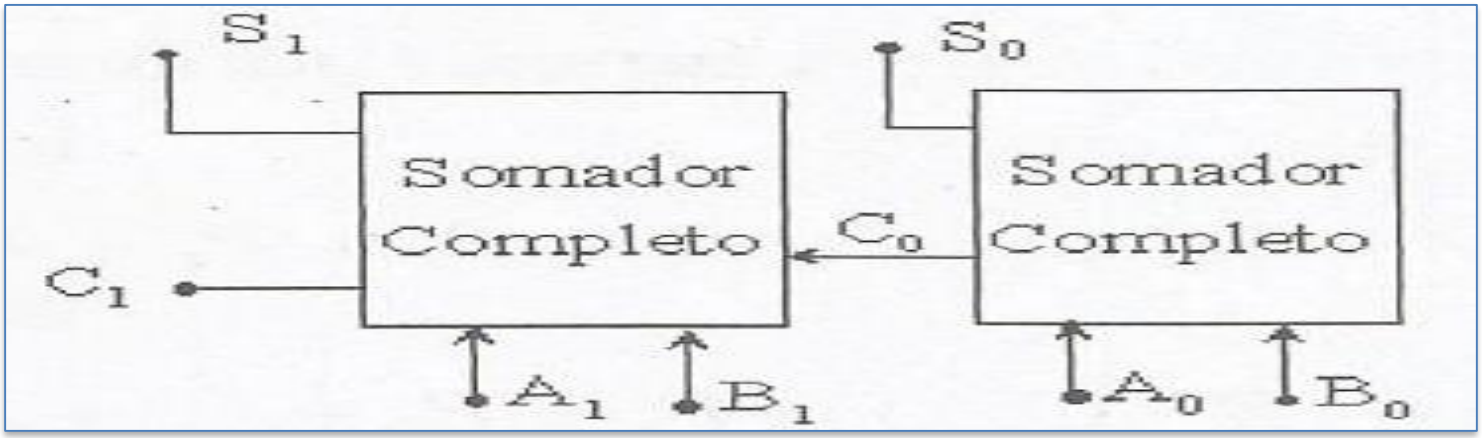


Entradas			Saídas	
A_n	B_n	C_{n-1}	S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Expressão booleana do somador completo

$$S_n = A_n \oplus B_n \oplus C_{n-1}$$
$$(C_n) = A_n.B_n + A_n.C_{n-1} + B_n.C_{n-1}$$

Sistema com somadores completos ligados em cascata

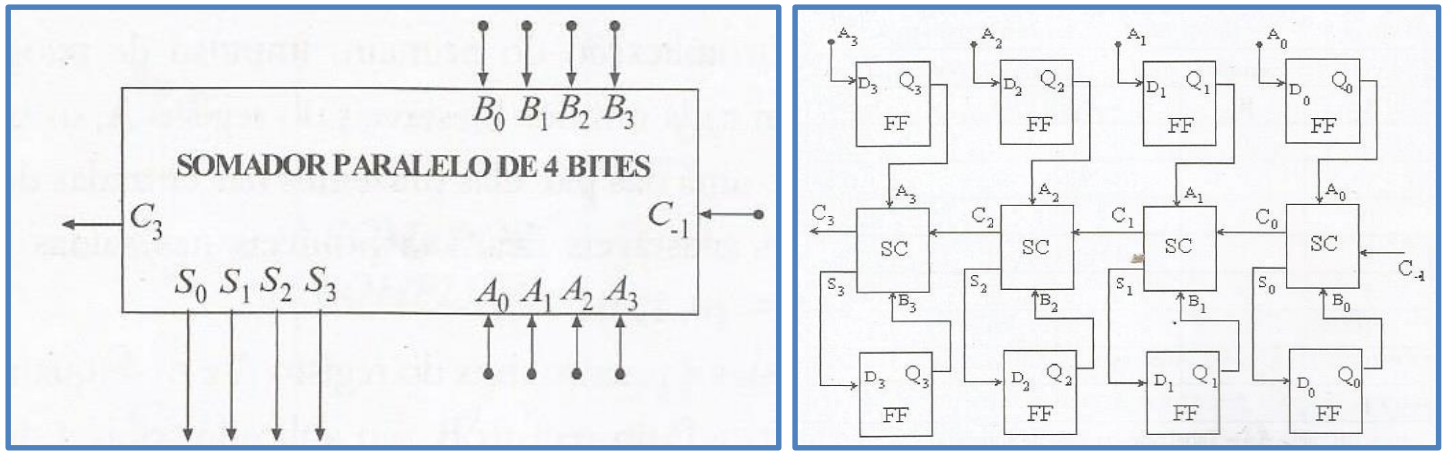


Com um sistema em cascata, é possível adicionar números com mais de 1 bit.

I.1.3. O SOMADOR – PARALELO

Somadores-paralelos – são circuitos com capacidade de adicionar números com 4 bits cada. É constituído por 4 somadores completos ligados em cascata e dois grupos de 4 biestáveis do tipo D, que funcionam como células de armazenamento dos números.

Símbolo e Esquema do somador-paralelo



Etapas para a realização da operação no somador-paralelo

a) Iniciam-se os 8 flip-flops a 0, fazendo-se também $C_{-1} = 0$;

b) Transferem-se os bits de uma das parcelas para o registo A, perfazendo-se $A = \{ A_0, A_1, A_2, A_3 \}$;

c) Transferem-se os bits existentes nos biestáveis para os somadores e adicionam-se os seus conteúdos: A_0+0, A_1+0, A_2+0 e A_3+0.

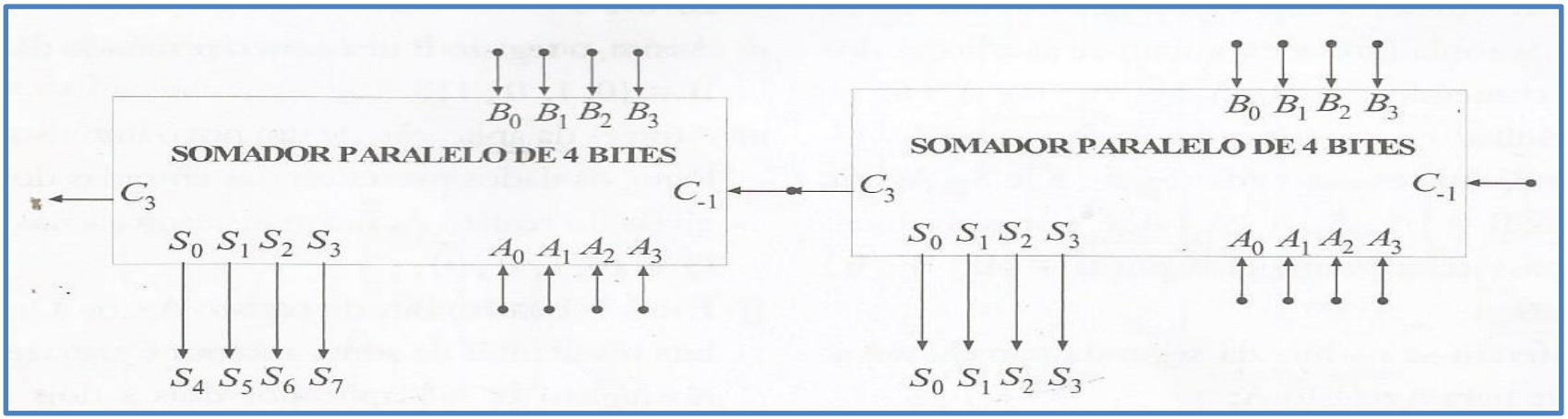
d) O resultado desta soma é transferido para o registo B, através das saídas S_0, S_1, S_2 e S_3. Assim perfaz: $B = \{ A_0, A_1, A_2, A_3 \}$, que por conveniência passaremos a designar por $B = \{ B_0, B_1, B_2, B_3 \}$;

e) Transferem-se os bits da segunda parcela novamente para o registo A.

f) Transferem-se os bits existentes nos biestáveis A e B e adicionam-se os seus conteúdos: A_0+B_0, A_1+B_1, A_2+B_2, A_3+B_3;

g) O resultado desta soma que é o valor pretendido, é transferido para o registo B, através das saídas S_0, S_1, S_2 e S_3.

Associação de somadores-paralelos a fim de se adicionar números com 8 bits.

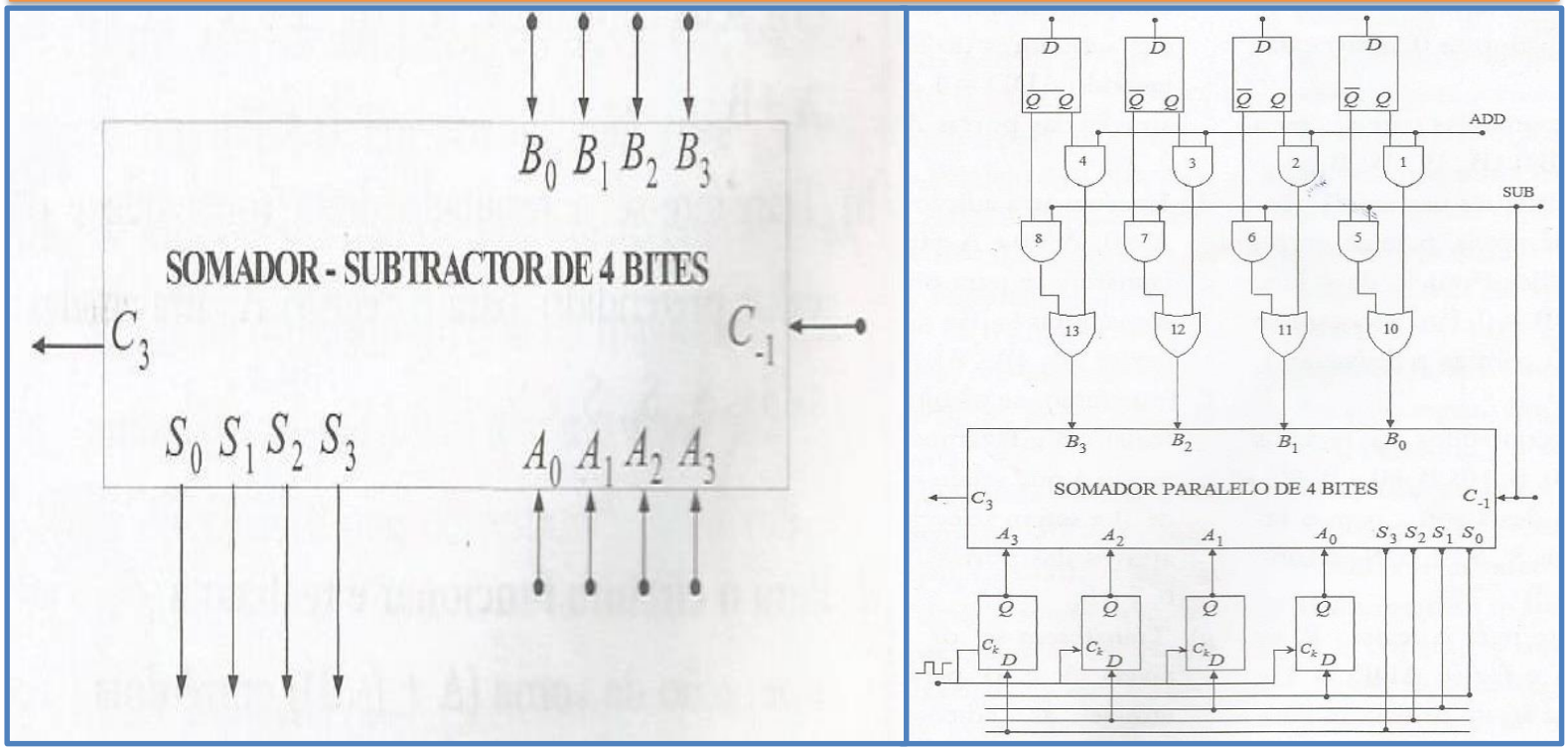


Estes podem ser acoplados de modo a adicionarem números com 8, 12 ou mais bits

I.1.4. O SOMADOR – SUBTRACTOR

Somador-subtractor – é um circuito capaz de realizar a soma e a subtracção de números binários, usando a forma de 2º complemento. É composto por um somador paralelo de 4 bits e uma lógica para seleccionar os valores adequados à realização da operação.

Símbolo e Esquema do somador-subtractor



Etapas para a realização da operação no somador-subtractor

1. Para a operação de soma (A+B) entre dois números binários de quatro bits cada:

a) Iniciam-se os 8 flip-flops a 0, através das entradas assíncronas;
b) Transferem-se os dados de uma das parcelas para o registo B, perfazendo B={ B_0, B_1 , B_2, B_3 };
c) Transferem-se os bits existentes nas saídas não complementadas dos biestáveis, para as entradas superiores do somador-paralelo de 4 bits, fazendo ADD=1 e SUB=0. Isto processa-se através das portas AND 1, 2, 3, e 4;
d) Executa-se a adição dos conteúdos dos registos de cima e de baixo B_0+0, B_1+0, B_2+0 e B_3+0;
e) Transfere-se o resultado desta soma para o registo A, através das saídas S_0, S_1 , S_2 e S_3 . Assim perfaz A={ B_0, B_1 , B_2, B_3 };
f) Transferem-se novamente para o registo B, os bits da segunda parcela e faz-se ADD=1 e SUB=0, de modo a que sejam os valores não-complementados a serem seleccionados para os somadores, através das portas AND 1, 2, 3 e 4;
g) Transferem-se os bits existentes nos biestáveis A e B para os somadores e executam-se as adições dos seus conteúdos: A_0+B_0, A_1+B_1, A_2+B_2, A_3+B_3;

h) Transfere-se o resultado desta soma que é o valor pretendido, para o registo A, através das saídas S_0, S_1 , S_2 e S_3 .

2. Para realizar a soma (A+ (-B)):

a) Iniciam-se os 8 flip-flops a 0, através das entradas assíncronas;
b) Transferem-se os dados de uma das parcelas para o registo B, perfazendo B={ B_0, B_1 , B_2, B_3 };
c) Transferem-se os bits existentes nas saídas não complementadas dos biestáveis, para as entradas superiores do somador-paralelo de 4 bits, fazendo ADD=1 e SUB=0. Isto processa-se através das portas AND 1, 2, 3, e 4;
d) Executa-se a adição dos conteúdos dos registos de cima e de baixo A_0+0, A_1+0, A_2+0 e A_3+0;
e) Transfere-se o resultado desta soma para o registo A, através das saídas S_0, S_1 , S_2 e S_3 . Assim perfaz A={ A_0, A_1 , A_2, A_3 };
f) Transferem-se novamente para o registo B, os bits da segunda parcela e faz-se ADD=0 e SUB=1, de modo a que sejam os valores complementados a serem seleccionados para os somadores, através das portas AND 5, 6, 7 e 8;
g) Transferem-se os bits existentes nos biestáveis A e B para os somadores e executam-se as adições dos seus conteúdos: $A_0+\overline{B}_0+1$, $A_1+\overline{B}_1+1$, $A_2+\overline{B}_2+1$ e $A3+\overline{B}_3+1$.

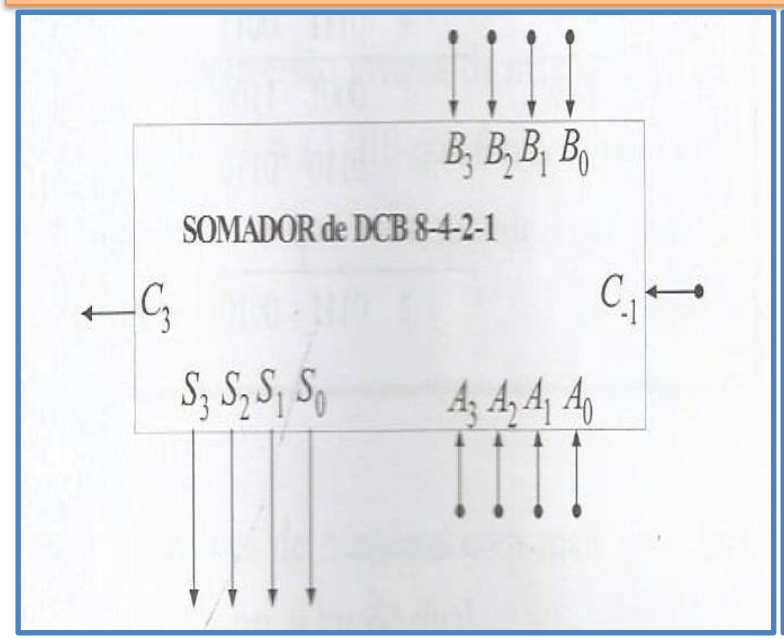
h) Transfere-se o resultado desta soma que é o valor pretendido, para o registo A, através das saídas S_0, S_1 , S_2 e S_3

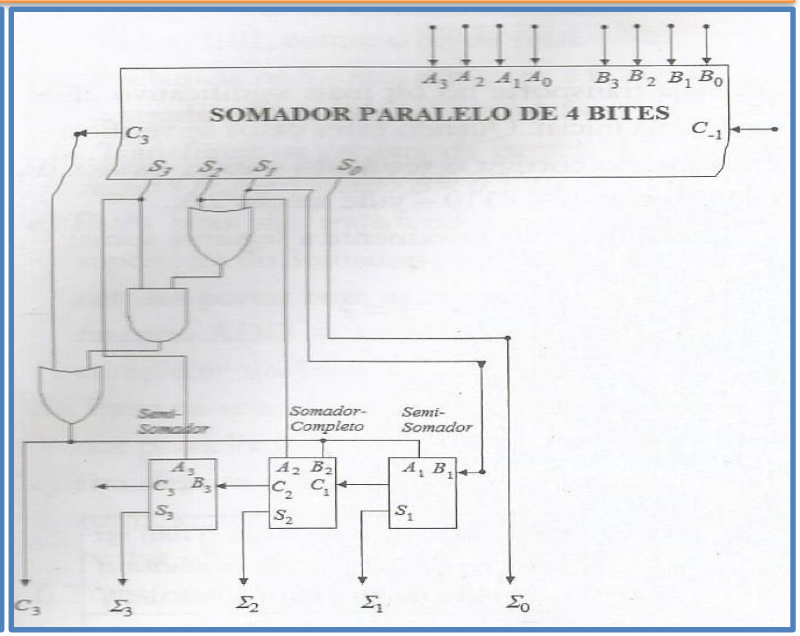
3. A+(+B) ou –A + (-B) são realizadas com as mesmas etapas mas, fazendo-se inicialmente ADD=0 e SUB=1 ou ADD=1 e SUB=0, em momentos apropriados ou em mais que uma vez.

I.1.5. O SOMADOR DE DCB

Somador de DCB – é um circuito capaz de realizar a soma de números na representação de Decimal Codificado em Binário. É composto por um somador-paralelo de 4 bits e uma lógica para detectar a ocorrência de somas iniciais maiores que nove ou em casos que haja transporte no bit mais significativo após uma soma inicial.

Símbolo e Esquema do somador de DCB



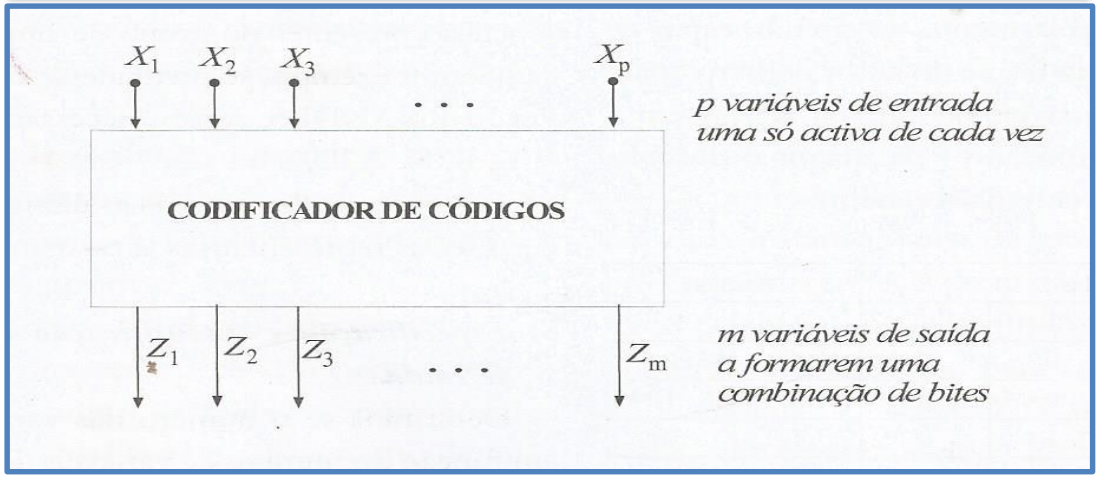


1.2. OS CIRCUITOS CODIFICADORES

Circuitos codificadores – são partes dos sistemas digitais que surgem da necessidade de transformar a informação do tipo alfa-numérico, fornecida pelos dispositivos periféricos do sistema digital , em binário.

Eis abaixo o esquema simbólico de um codificador:

Símbolo



Esquema

O esquema lógico de cada codificador, é diferente dependendo de vários factores, entre os quais o tipo de transformações que se pretende implementar.

Classificação:

- Circuitos codificadores comuns (sem-prioridade)
- Circuitos codificadores de prioridade

I.2.1. OS CIRCUITOS CODIFICADORES COMUNS (Sem-prioridade)

Etapas para a sua implementação

1. Especificação e quantificação das variáveis de entrada
2. Especificação e quantificação das variáveis de saída
3. Codificação das variáveis
4. Construção da tabela-de-verdade
5. Explicitação das expressões booleanas
6. Criação do circuito de implementação

1. Especificação e quantificação das variáveis de entrada

O número de variáveis de entrada (p), coincide com o número de símbolos que se pretende codificar .

Exemplo 1: Circuito codificador capaz de codificar a palavra IGOR.
Temos 4 símbolos diferentes (I,G,O e R), logo p=4.

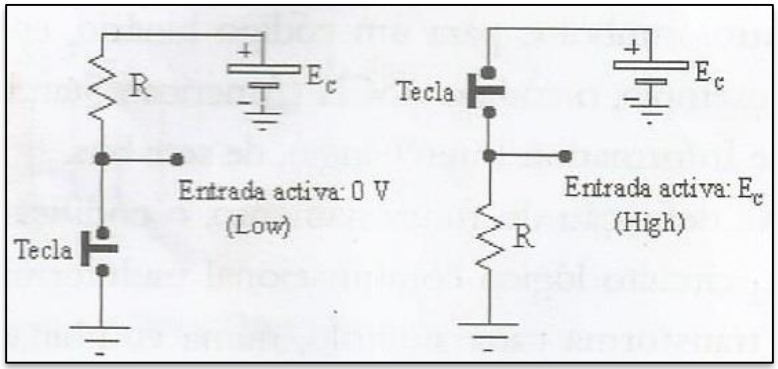
Exemplo 2: Circuito codificador capaz de codificar a palavra DIAMANTE.
Temos 8 símbolos (D,I,A,M,A,N,T e E), dos quais o símbolo A foi repetido.
Porém só precisamos de uma variável para cada tipo de símbolo. Logo consideraremos como variáveis de entrada os 7 símbolos diferentes (D,I,A,M,N,T e E), isto é p=7.

2. Especificação e quantificação das variáveis de saída

Determina-se o número de variáveis de saídas (m) em função do número de variáveis de entrada (p), pela fórmula: $p <= 2^m$.
Assim para o exemplo 1 (IGOR): $4 <= 2^2$, logo: m=2.

3. Codificação das variáveis

As variáveis de entrada podem ser activas em High (1) ou Low (0), dependendo do tipo de circuito associado às teclas - Fig. abaixo.



As variáveis de saída, podemos escolhê-las também activas em High (1) ou Low (0).
Assim para o nosso exemplo (IGOR), podemos escolher os códigos: I=00; G=01; O=10 e R=11.

4. Construção da tabela-de-verdade

Tabela-de-verdade 1: Entradas activas em High (1).

Entradas				Saídas	
I	G	O	R	Z_0	Z_1
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Em cada instante (linha da tabela-de-verdade), se pretende uma só tecla premida, por isso tem-se uma só tecla activa em High (1), estando as outras em Low (0).

Por seu turno, às variáveis de saída associa-se, a cada activação de uma variável de entrada, uma combinação de bits do código escolhido. Escolheu-se Z_0 =0 e Z_1 =0 para I, Z_0 =0 e Z_1 =1 para G, Z_0 =1 e Z_1 =0 para O e Z_0 =1 e Z_1 =1 para R .

Tabela-de-verdade 2: Entradas activas em Low (0).

Entradas				Saídas	
I	G	O	R	Z_0	Z_1
0	1	1	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	0	1	1

Em cada instante (linha da tabela-de-verdade), se pretende uma só tecla premida, por isso tem-se uma só tecla activa em Low (0), estando as outras em Low (1).

A codificação das variáveis de saída pode ser igual: $Z_0 = 0$ e $Z_1 = 0$ para I, $Z_0 = 0$ e $Z_1 = 1$ para G, $Z_0 = 1$ e $Z_1 = 0$ para O e $Z_0 = 1$ e $Z_1 = 1$ para R .

5. Explicitação das expressões booleanas

- Quando as variáveis de entrada e de saída são activas em High (High-High) as expressões finais das variáveis de saída são disjunções das variáveis de entrada activadas e não-complementadas.
- Quando as entradas são activas em High e as saídas activas em Low (High-Low) essas expressões são conjunções de variáveis activadas e complementadas.
- Quando as entradas são activas em Low e as saídas em Low (Low-Low) as expressões são conjunções das variáveis activadas e não-complementadas.
- Quando as entradas são activas em Low e as saídas activas em High (Low-High) as expressões finais são disjunções de variáveis complementadas.

Voltando ao nosso exemplo (IGOR), teremos as seguintes expressões para os seguintes casos:

a) High-High: $Z_0 = O+R$ e $Z_1 = G+R$

b) Low-Low: $Z_0 = I.G$ e $Z_1 = I.O$

c) High-Low: $Z_0 = \overline{I}.\overline{G}$ e $Z_1 = \overline{I}.\overline{O}$

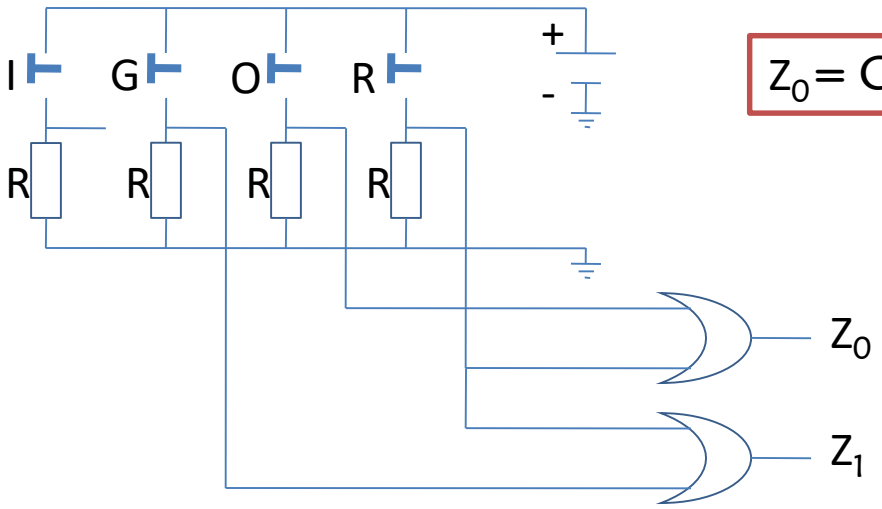
d) Low-High: $Z_0 = \overline{O}+\overline{R}$ e $Z_1 = \overline{I}+\overline{R}$

6. Criação do circuito de implementação

Para a criação do circuito de implementação, vale converter as expressões boolenas para a forma de esquema lógico.

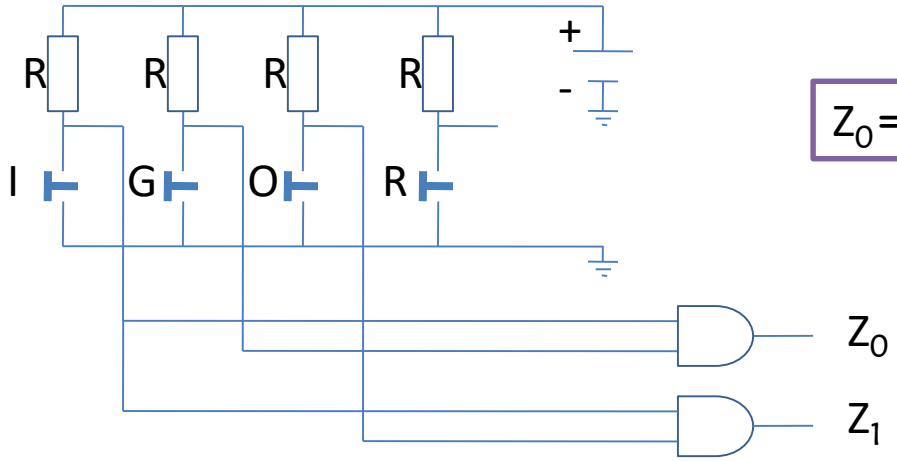
Façamos então os esquemas do nosso codificador exemplo, nas variantes High-High, Low-Low e High-Low :

Esquema do codificador , High-High:



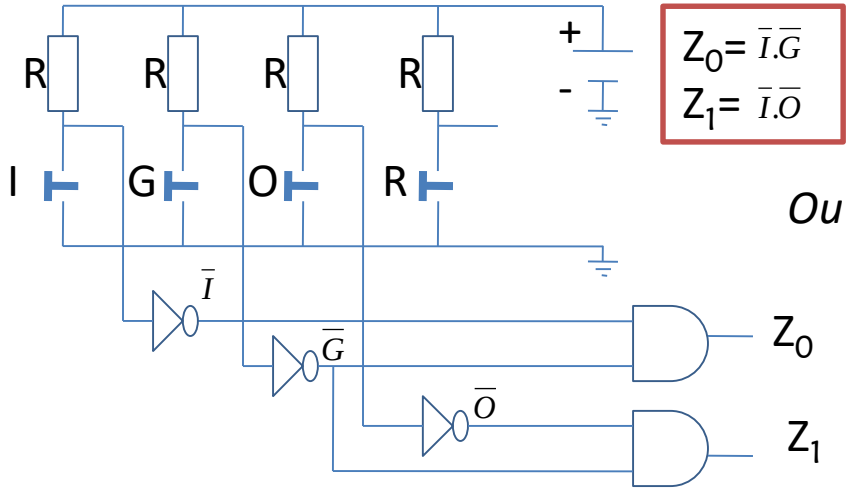
$Z_0 = O+R$ e $Z_1 = G+R$

Esquema do codificador , Low-Low:



$Z_0 = I.G$ e $Z_1 = I.O$

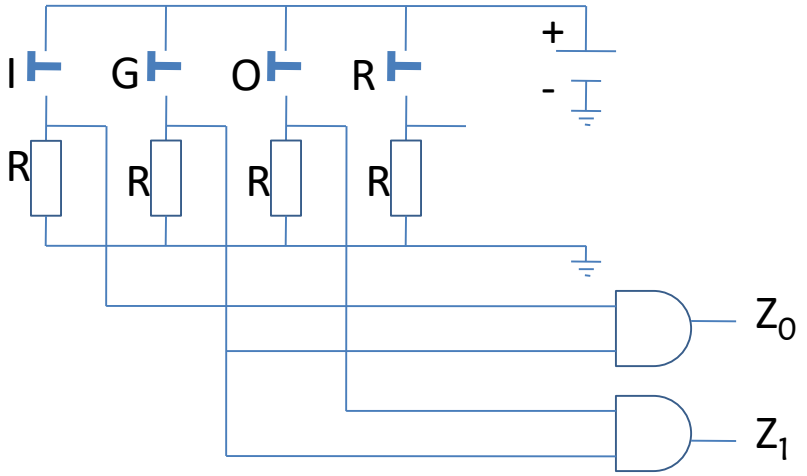
Esquema do codificador , High-Low:



$Z_0 = \overline{I}.\overline{G}$

$Z_1 = \overline{I}.\overline{O}$

Ou



1.2.1. OS CIRCUITOS CODIFICADORES DE PRIORIDADE

Etapas para a sua implementação

Os codificadores de prioridade são implementados diferentemente dos anteriores.
Estes codificadores surgem para minimizar a ocorrência de erros de codificação, resultantes na sequência dos accionamentos duplos, que motivam que mais do que uma tecla, seja premida em simultâneo.
Para tal, estabelecem-se níveis de prioridades para alguns dos símbolos e caso ocorram simultaneamente dois ou mais símbolos, dá-se prevalência de apresentação de código à saída, ao símbolo com maior prioridade.

1. Especificação e quantificação das variáveis de entrada

O número de variáveis de entrada (p) estabelece-se, basicamente, como no caso do codificador sem-prioridade.
Exemplo: Circuito codificador capaz de codificar a palavra IGOR.
Temos 4 símbolos diferentes (I,G,O e R), logo p=4.

I.2.1. OS CIRCUITOS CODIFICADORES DE PRIORIDADE

Etapas para a sua implementação

Os codificadores de prioridade são implementados diferentemente dos anteriores.
Estes codificadores surgem para minimizar a ocorrência de erros de codificação, resultantes na sequência dos accionamentos duplos, que motivam que mais do que uma tecla, seja premida em simultâneo.
Para tal, estabelecem-se níveis de prioridades para alguns dos símbolos e caso ocorram simultaneamente dois ou mais símbolos, dá-se prevalência de apresentação de código à saída, ao símbolo com maior prioridade.

1. Especificação e quantificação das variáveis de entrada

O número de variáveis de entrada (p) estabelece-se, basicamente, como no caso do codificador sem-prioridade; considerando as entradas e saídas activas em High.
Exemplo: Circuito codificador capaz de codificar a palavra IGOR, em que as letras mais a esquerda têm maior prioridade relativamente às situadas à sua direita.
Temos 4 símbolos diferentes (I,G,O e R), logo p=4.

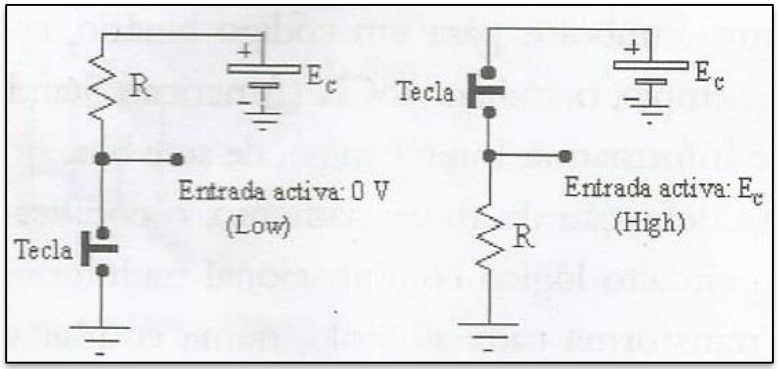
2. Especificação e quantificação das variáveis de saída

$p <= 2^m$.

$4 <= 2^2$, logo: m=2.

3. Codificação das variáveis

Tal como no caso do codificador sem-prioridade, as variáveis de entrada podem ser activas em High (1) ou Low (0),
dependendo do tipo de circuito associado às teclas - Fig. abaixo.



As variáveis de saída, podemos escolhê-las também activas em High (1) ou Low (0).
Assim para o nosso exemplo (IGOR), podemos escolher os códigos: I=00; G=01; O=10 e R=11.

4. Construção da tabela-de-verdade

Tabela-de-verdade :

Entradas				Saídas		
I	G	O	R	Z_0	Z_1	V
1	X	X	X	0	0	1
0	1	X	X	0	1	1
0	0	1	X	1	0	1
0	0	0	1	1	1	1
0	0	0	0	X	X	0

i. Quando I está activo (2ª linha), o código (00) a si associado aparece À saída, indiferentemente do estado das outras variáveis (indiferentemente significa X).

ii. Quando G está activo, o código (01) a si associado aparece à saída, apenas se I estiver desactivado (0), não sendo relevante o estado das outras variáveis de menor prioridade.

iii. Quando O está activo, o código (10) a si associado aparece à saída, se I e G estiverem desactivados (0), não sendo relevante o estado das outras variáveis de menor prioridade.

iv. R activo, não havendo outras variáveis de menor prioridade, o código a si associado (11) aparece à saída, quando todas as outras variáveis estiverem desactivadas (0).

v. No caso de nenhuma das variáveis estar activada, o código a associar pode ser qualquer (isto é X). Neste caso, o código deve ser anotado como não-válido, atribuindo-se 0 à variável V.

5. Explicitação das expressões booleanas

Pela forma como a tabela-de-verdade construída se apresenta, as saídas devem ser explicitadas pelo mapa de Karnaugh. Repare-se que muitos dos termos estão apresentados implicitamente. Apenas os termos de ordem 0 e 1 estão explícitos.

Z_0 (I,G,O,R)

OR \ I G	00	01	11	10
00	X	0	0	0
01	1	0	0	0
11	1	0	0	0
10	1	0	0	0

$Z_0 = \overline{I}.\overline{G}$

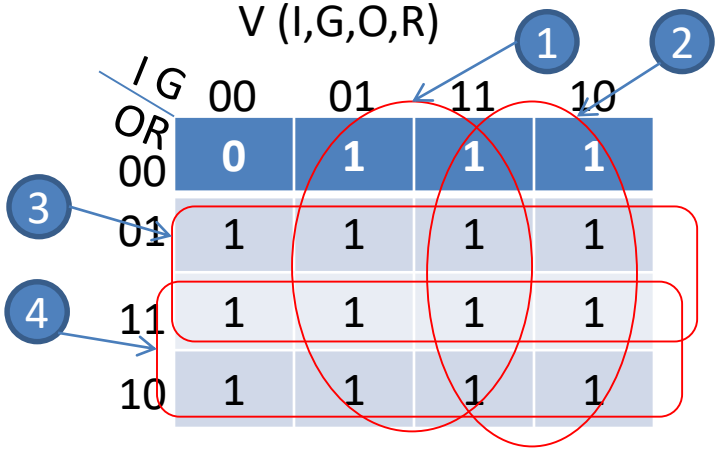
Z_1 (I,G,O,R)

I G / OR	00	01	11	10
00	X	1	0	0
01	1	1	0	0
11	0	1	0	0
10	0	1	0	0

$Z_1 = \bar{I}.G + \bar{I}.\bar{O}$

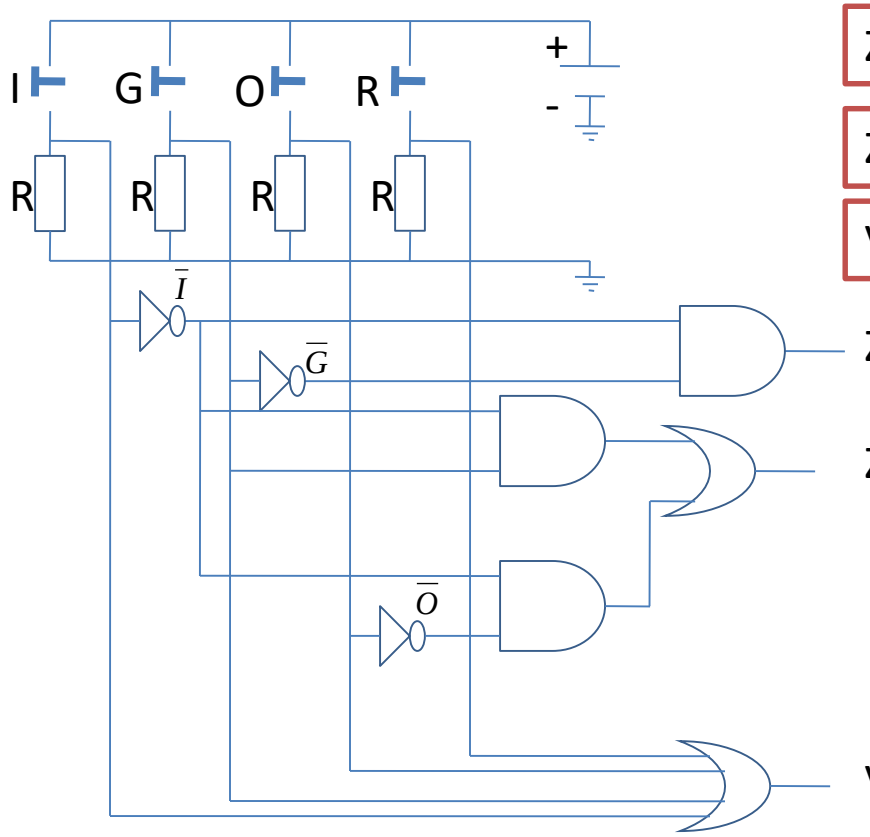
V (I,G,O,R)

I G / OR	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1



V= G+I+O+R = I+G+O+R

6. Criação do circuito de implementação



$Z_0 = \bar{I}.\bar{G}$

$Z_1 = \bar{I}.G + \bar{I}.\bar{O}$

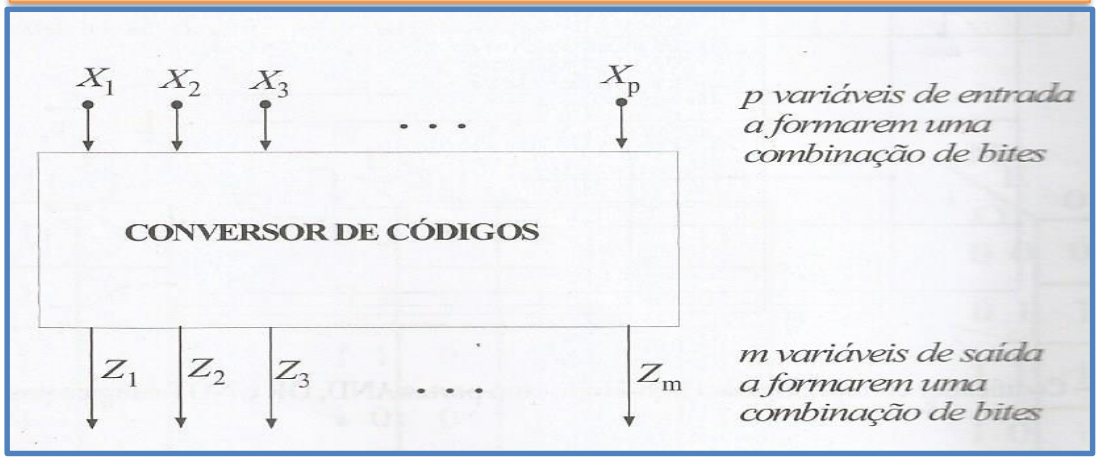
V= G+I+O+R = I+G+O+R

1.3. O CIRCUITO CONVERSOR DE CÓDIGO

Conversores de Códigos – são partes dos sistemas digitais que transformam a informação existente num código para outro código em representação binária.

Exemplo de códigos que podem transformar-se entre si: Gray, binário-ordinário, DCB-8421, DCB-EXC 3, ASCII de sete bits, etc.

Símbolo



Esquema

O esquema lógico de cada conversor de códigos, é diferente dependendo de vários factores, entre os quais o tipo de transformações que se pretende implementar.

Etapas para a sua implementação

1. Especificação e quantificação das variáveis de entrada
2. Especificação e quantificação das variáveis de saída
3. Codificação das variáveis
4. Construção da tabela-de-verdade
5. Explicitação das expressões booleanas
6. Criação do circuito de implementação

1. Especificação e quantificação das variáveis de entrada

O número de variáveis de entrada (p), coincide com o número de bits do código de origem.

Exemplo: Implementemos um circuito conversor do **Gray para o DCB-8-4-2-1.**

O código de origem (Gray) tem 4 bits. Logo: $p=4$.

2. Especificação e quantificação das variáveis de saída

O número de variáveis de saída (m), coincide com o número de bits do código de destino.

O código de destino (DCB-8-4-2-1) tem sempre 10 combinações com 4 bits, então o código de entrada convém que se considere também, com 4 bits de modo a se considerarem todas as combinações do código de saída (destino).

$$m=4.$$

Obs: Casos haverá, em que o número de bits de entrada ou saída depende do número de bits usados pelo outro lado (porto) do sistema.

3. Codificação das variáveis

As variáveis de entrada podem ser activas em High (1) ou Low (0). Esta consideração é irrelevante, porque a estas variáveis associamos uma combinação de bits correspondente a um código.

As variáveis de saída, podemos escolhê-las também activas em High (1) ou Low (0).

Assim para o nosso caso, teremos por exemplo: 1100 (Gray) - > 1000 (DCB), só para avançar uma das conversões.

4. Construção da tabela-de-verdade

Tabela-de-verdade para o Conversor de Código Gray-DCB-8-4-2-1:

Entradas: Gray				Saídas: DCB			
A	B	C	D	Z_0	Z_1	Z_2	Z_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	X	X	X	X
1	1	1	0	X	X	X	X

5. Explicitação das expressões booleanas

- Se se pretender implementar com **portas lógicas**, deve transcrever-se as funções de saída para o diagrama de Veitch-Karnaugh e proceder a simplificação, para obter as expressões sob a forma **elementar conjuntiva** ou **disjuntiva**. Depois transcreve-se as funções para as **formas não-elementares** em conformidade com o tipo de portas lógicas que se pretenda usar, na lógica positiva ou negativa.
- Se se pretender implementar com **MUX** ou **DMUX**, transcreve-se as expressões de saída para as **formas canónicas disjuntivas**.

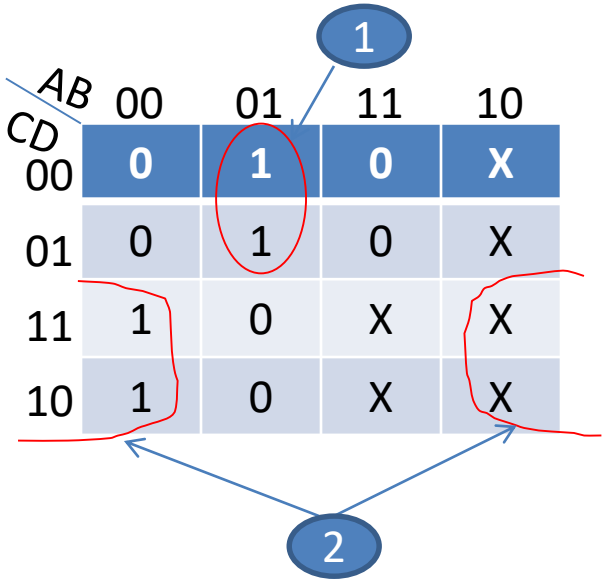
Implementemos nos dois métodos: primeiro com portas lógicas NAND e depois com DMUX e portas NAND e por fim com MUX, na lógica positiva.

AB / CD	00	01	11	10
00	**0**	**0**	**1**	**X**
01	0	0	1	X
11	0	0	X	X
10	0	0	X	X

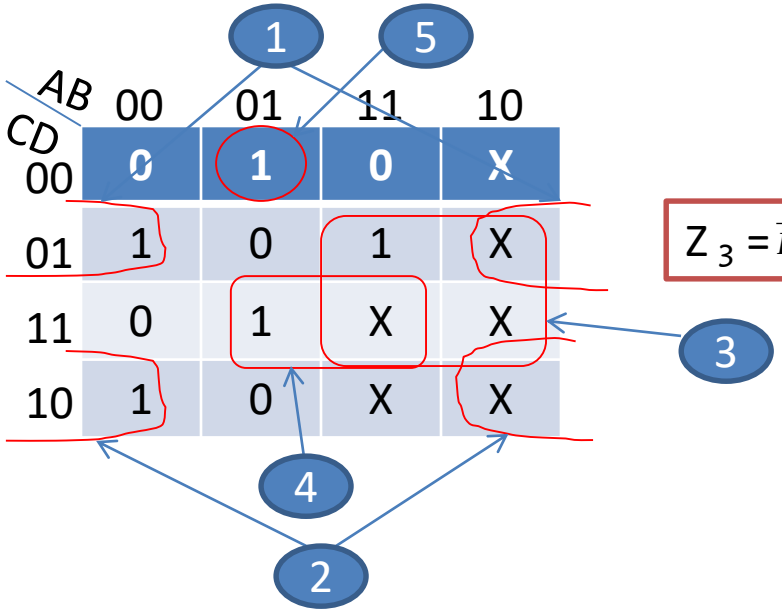
$Z_0 = A$

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

$Z_1 = \overline{A}.B$



$Z_2 = \overline{A}.B.\overline{C} + \overline{B}.C$



$Z_3 = \overline{B}.\overline{C}.D + \overline{B}.C.\overline{D} + A.D + B.C.D + \overline{A}.B.\overline{C}.\overline{D}$

1- Implementação com portas NAND:

$Z_0 = A$

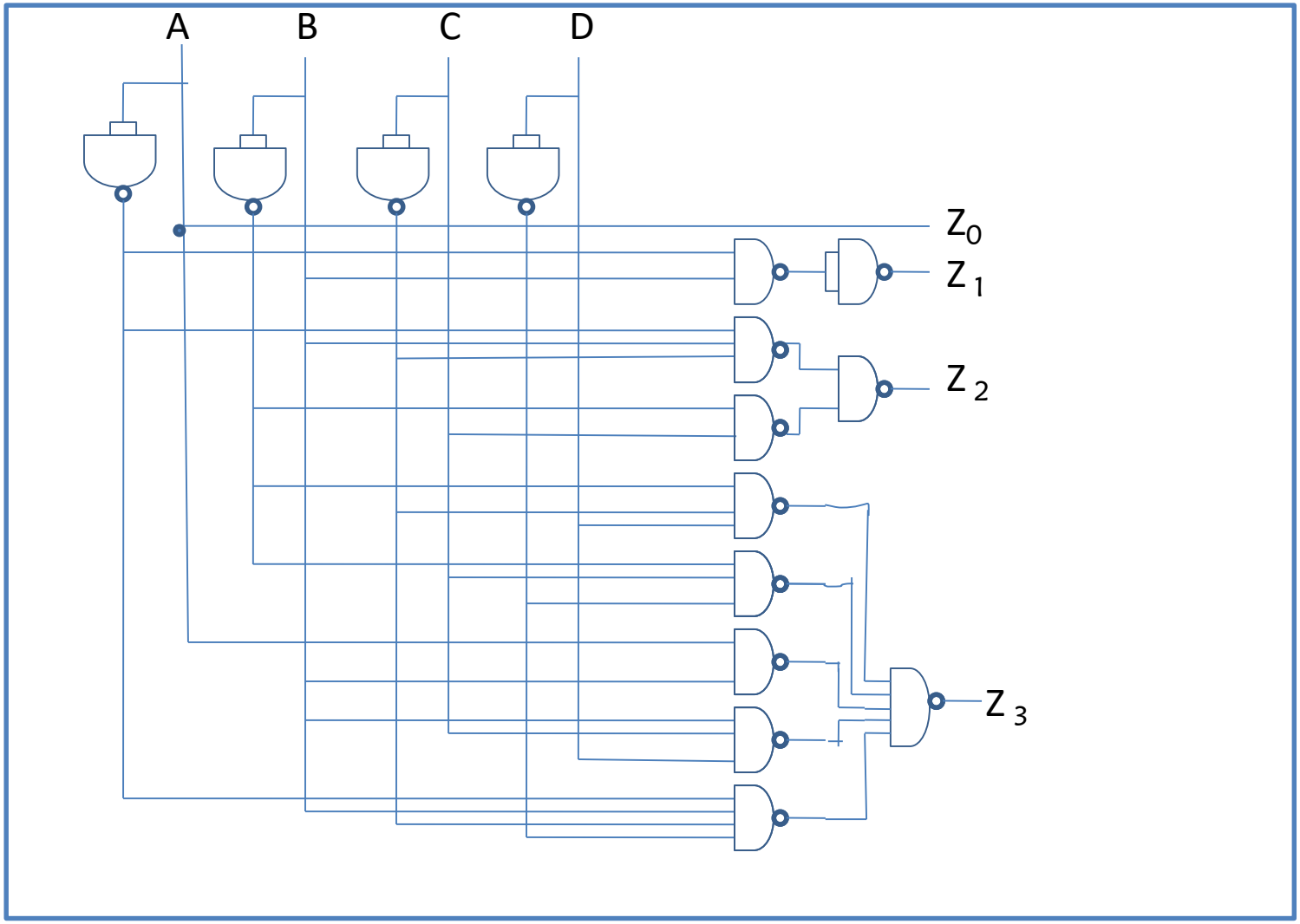
$Z_1 = \overline{\overline{(\overline{A}.B)}}$

$Z_2 = \overline{\overline{\overline{A}.B.\overline{C}}.\overline{\overline{B}.C}} = \overline{\overline{\overline{A}.B.\overline{C}}.\overline{B}.C}$

$Z_3 = \overline{\overline{\overline{B}.\overline{C}.D + \overline{B}.C.\overline{D} + A.D + B.C.D + \overline{A}.B.\overline{C}.\overline{D}}}$

$= \overline{\overline{B}.\overline{C}.D}.\overline{\overline{B}.C.\overline{D}}.\overline{A.D}.\overline{B.C.D}.\overline{\overline{A}.B.\overline{C}.\overline{D}}$

6. Circuito de implementação



2- Implementação com DMUX e portas NAND:

a) Transformemos as expressões para a forma canónica disjuntiva, a partir do mapa de Karnaugh ou da tabela-de-verdade.

$Z_0 = P_{12} + P_{13}$

$Z_1 = P_4 + P_5 + P_6 + P_7$

$Z_2 = P_2 + P_3 + P_4 + P_5$

$Z_3 = P_1 + P_2 + P_4 + P_7 + P_{13}$

b) Transformemos pela operação primitiva NAND:

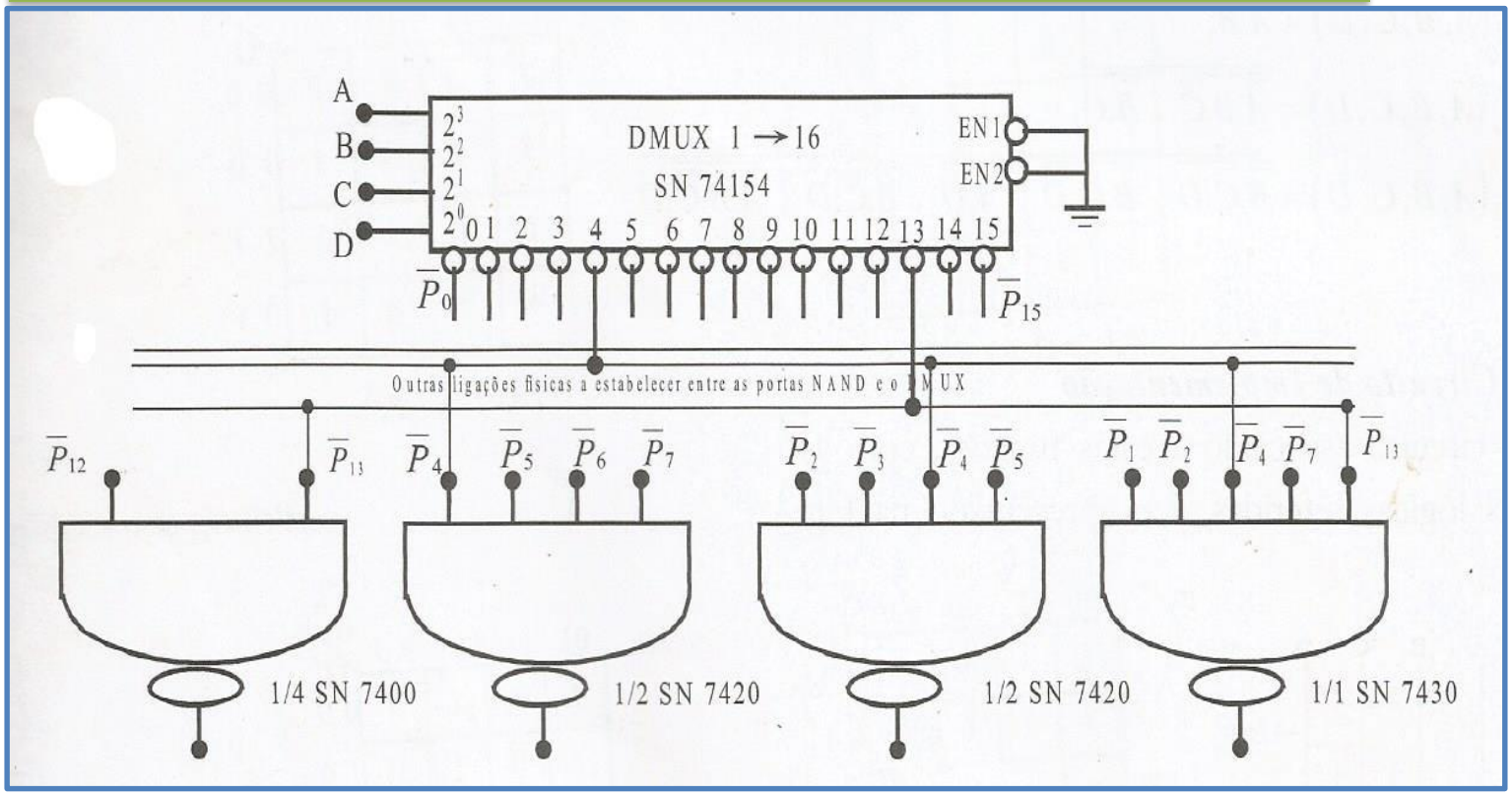
$Z_0 = \overline{\overline{P12} \cdot \overline{P13}}$

$Z_1 = \overline{\overline{P4} \cdot \overline{P5} \cdot \overline{P6} \cdot \overline{P7}}$

$Z_2 = \overline{\overline{P2} \cdot \overline{P3} \cdot \overline{P4} \cdot \overline{P5}}$

$Z_3 = \overline{\overline{P1} \cdot \overline{P2} \cdot \overline{P4} \cdot \overline{P7} \cdot \overline{P13}}$

Circuito do Conversor de Código Gray-DCB-8421, com DMUX e portas NAND, na variante convencional e na lógica positiva

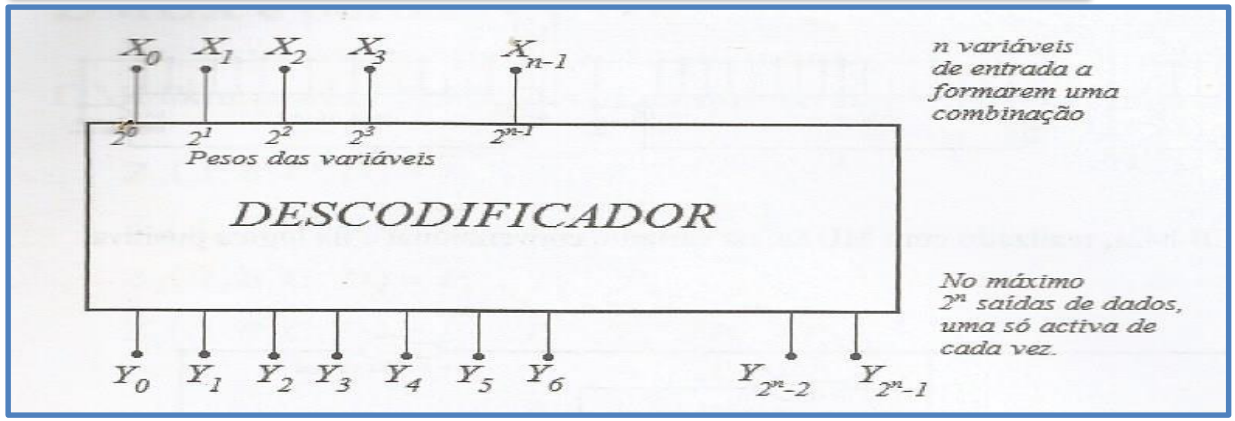


1.4. O CIRCUITO DESCODIFICADOR

Circuitos Descodificadores – são partes dos sistemas digitais que transformam a informação representada num código em representação binária, em informação do tipo alfa-numérica perceptível pelo utilizador humano.

Exemplo de algumas transformações: binário-decimal, DCB-decimal, DCB-7-segmentos, binário-ASCII.

Símbolo



Esquema

O esquema lógico de cada descodificador, é diferente dependendo de vários factores, entre os quais o tipo de transformações que se pretende implementar.

DESCODIFICADOR BINÁRIO-QUATERNÁRIO

1. Especificação e quantificação das variáveis de saída

O número de variáveis de saída (m), coincide com o número de símbolos a descodificar.

Pelo facto de se pretender representar a informação do sistema quaternário, ***m=4: S_0, S_1, S_2 e S_3.***

2. Especificação e quantificação das variáveis de entrada

O número de variáveis de entrada (p), determina-se em função do número das variáveis de saída, de acordo com a expressão: $2^p >= m$.

Resulta p=2: ***E_0, E_1.***

3. Codificação das variáveis

As variáveis de saída podem ser activas em High (1) ou Low (0), dependendo do tipo do circuito associado aos dispositivos de visualização.

As variáveis de entrada, podemos escolhê-las também activas em High (1) ou em Low (0). Nesta fase, esta consideração é irrelevante porque, a cada variável é associada uma combinação de bits.

4. Construção da tabela-de-verdade

Entradas		Saídas				Display (ou Mostrador)
E_0	E_1	S_0	S_1	S_2	S_3	
0	0	1	0	0	0	0
0	1	0	1	0	0	1
1	0	0	0	1	0	2
1	1	0	0	0	1	3

5. Explicitação das expressões booleanas

A partir da tabela-de-verdade obtêm-se as expressões das variáveis de saída que se seguem:

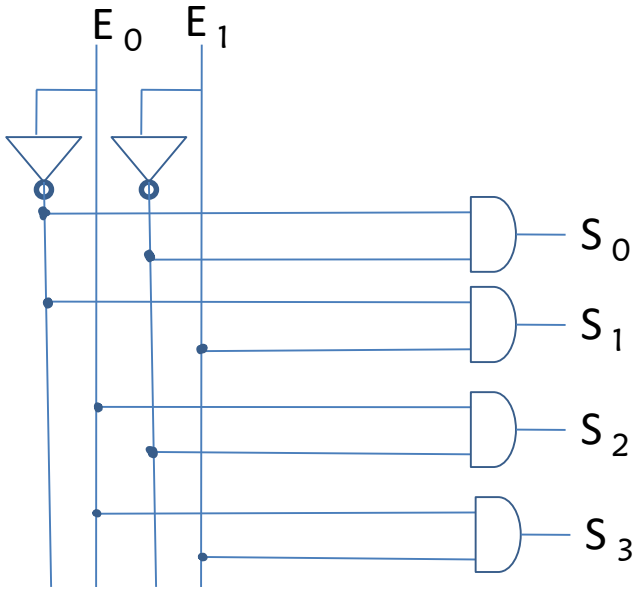
$S_0 = \overline{E0}.\overline{E1}$

$S_1 = \overline{E0}.E1$

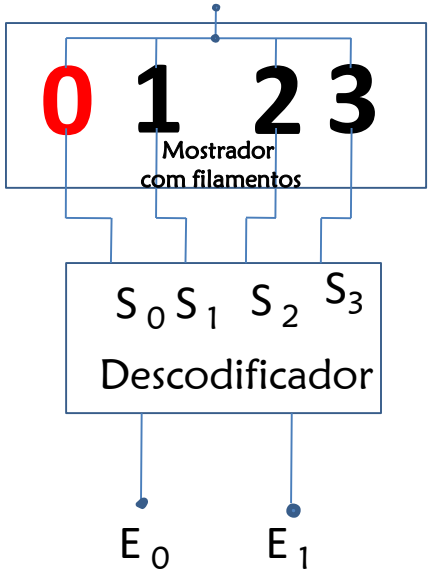
$S_2 = E0.\overline{E1}$

$S_3 = E0.E1$

6. Circuito de Implementação



Esquema Lógico

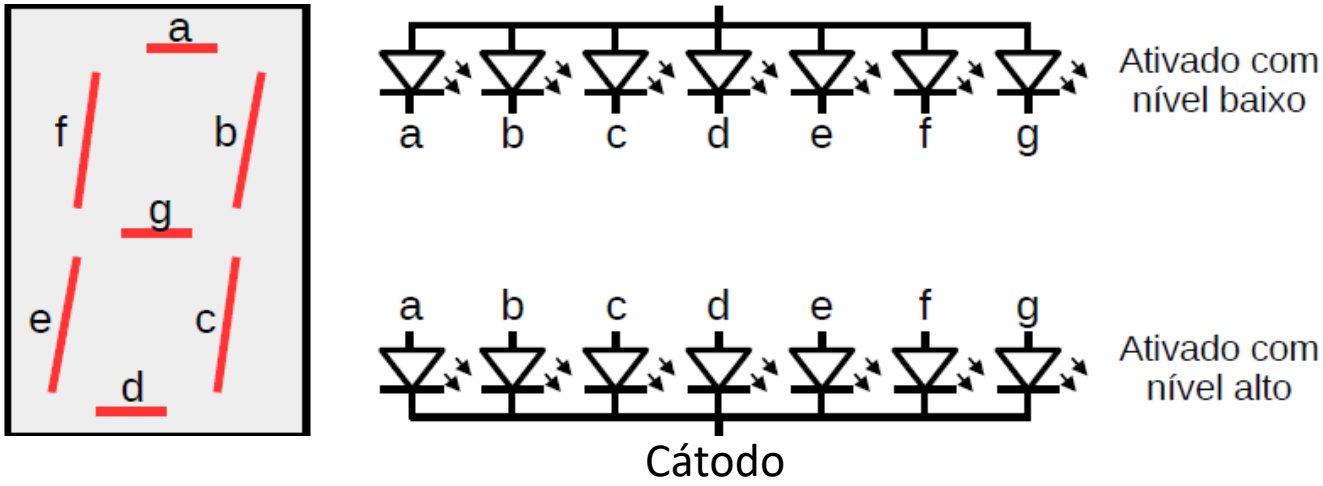


Representação simbólica

DESCODIFICADOR DCB-7-SEGMENTOS

Mostrador de 7-segmentos

- Formado por 7 LEDs
  - Dois tipos: Ânodo Comum e Cátodo Comum.



DESCODIFICADOR DCB-7-SEGMENTOS

1. Especificação e quantificação das variáveis de saída

Pelo facto de o mostrador dispor de ***7 (sete) segmentos*** , o número de variáveis de saída é ***m=7***: ***a, b, c, d, e, f, e g.***

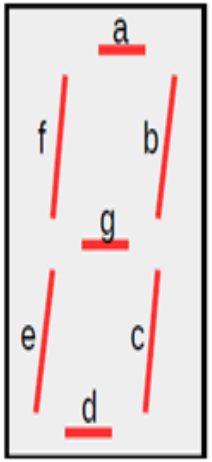
2. Especificação e quantificação das variáveis de entrada

No caso do DCB-7-segmentos, o número de variáveis de entrada não depende do número de variáveis de saída, mas sim do número de símbolos que se pretende visualizar no mostrador. O código DCB é apropriado para representar 10 símbolos do sistema decimal.
Portanto, ***p=4 : E_0, E_1, E_2 e E_3.***

3. Codificação das variáveis

As variáveis de saída podem ser activas em High (1) ou Low (0), dependendo do tipo de mostrador disponível.
Para as variáveis de entrada a codificação é irrelevante, sendo necessário apenas escrever-se o código DCB-8-4-2-1.

4. Construção da tabela-de-verdade



Entradas				Saídas							Mostrador tipo cátodo comum
E_0	E_1	E_2	E_3	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9

5. Explicitação das expressões booleanas

Transcrevemos a função booleana da forma de tabela-de-verdade para a forma canónica disjuntiva:

a $(E_0, E_1, E_2, E_3) = P_0 + P_2 + P_3 + P_5 + P_6 + P_7 + P_8 + P_9$;

b $(E_0, E_1, E_2, E_3) = P_0 + P_1 + P_2 + P_3 + P_4 + P_7 + P_8 + P_9$;

c $(E_0, E_1, E_2, E_3) = P_0 + P_1 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8 + P_9$;

d $(E_0, E_1, E_2, E_3) = P_0 + P_2 + P_3 + P_5 + P_6 + P_8$;

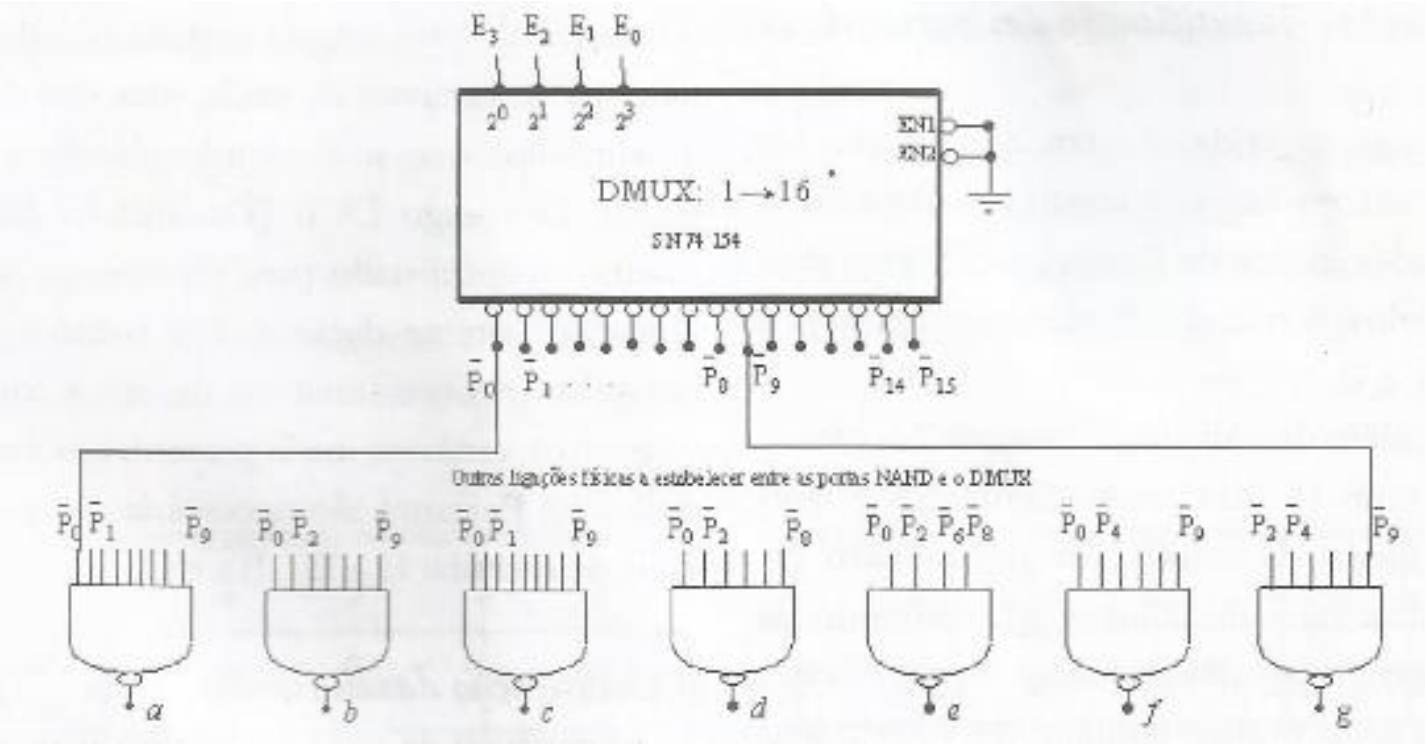
e $(E_0, E_1, E_2, E_3) = P_0 + P_2 + P_6 + P_8$;

f $(E_0, E_1, E_2, E_3) = P_0 + P_4 + P_5 + P_6 + P_8 + P_9$;

g $(E_0, E_1, E_2, E_3) = P_2 + P_3 + P_4 + P_5 + P_6 + P_8 + P_9$;

6. Circuito de implementação

O desenho do circuito descodificador DCB-7-segmentos com DMUX e portas NAND está apresentado abaixo:



Esquema com Circuitos Integrados

1- Implemente um codificador capaz de codificar os dígitos de 0 a 9, considerando as entradas e saídas activas em Low.