

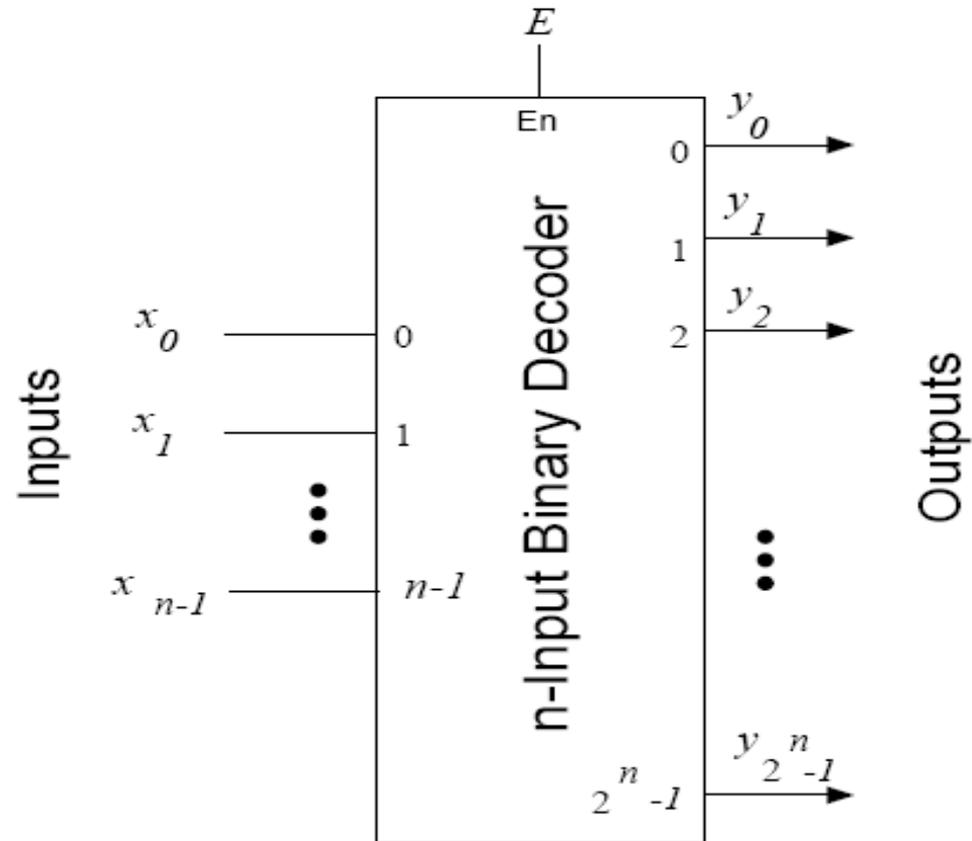
Circuitos Combinacionais Complexos

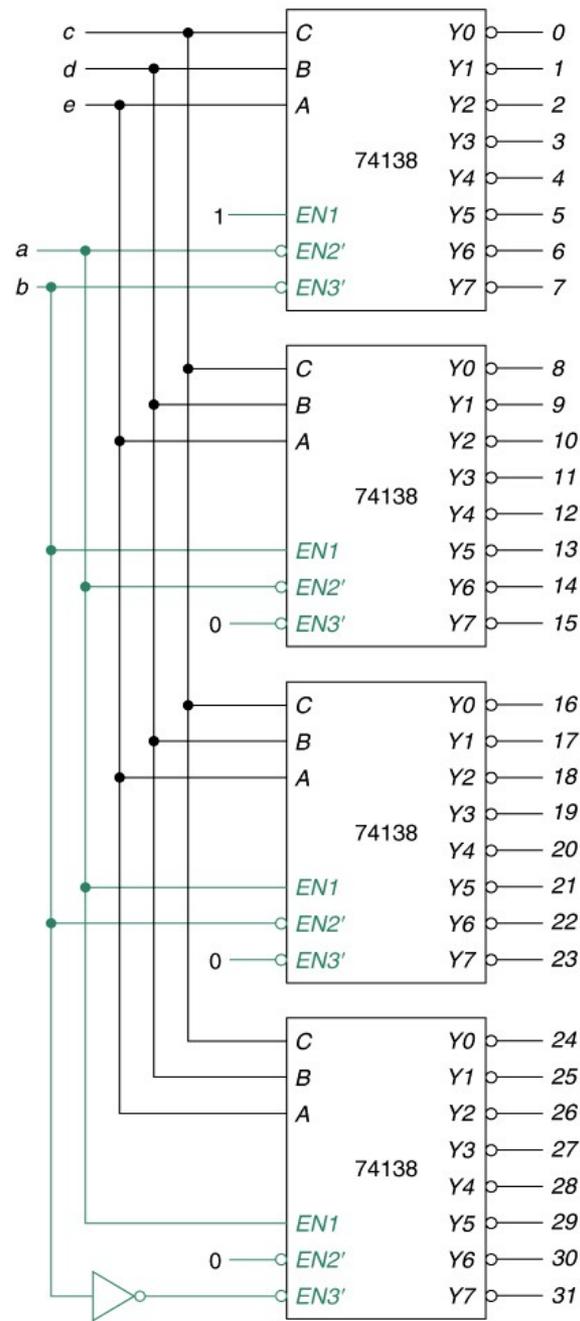
Circuitos Lógicos

DCC-IM/UFRJ

Prof. Gabriel P. Silva

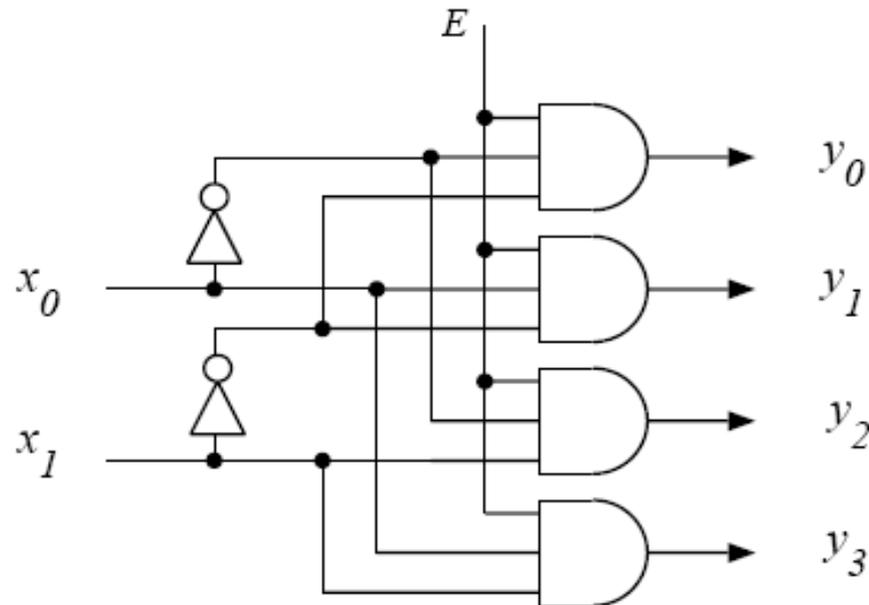
Decodificador



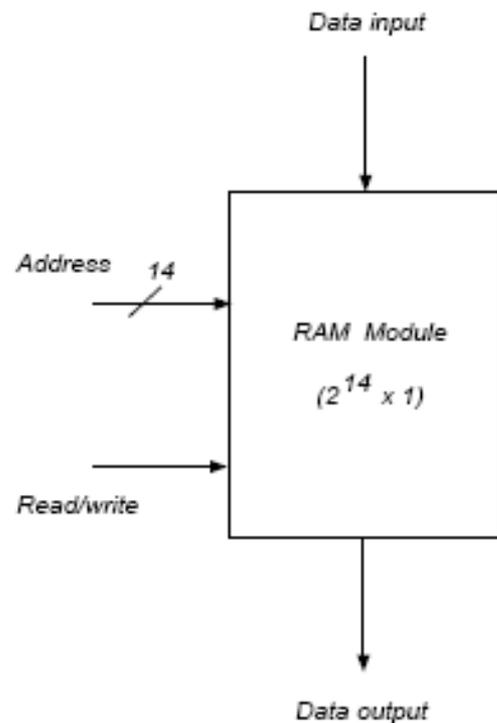


Decodificador

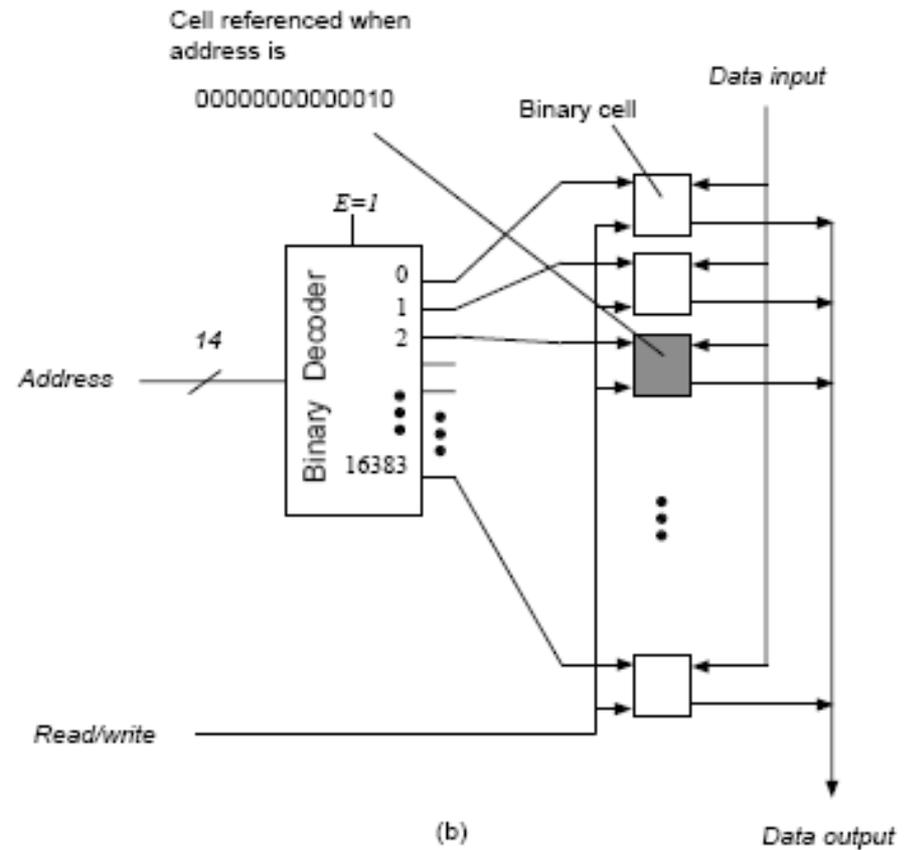
- Implementação de um decodificador com duas entradas



Decodificador

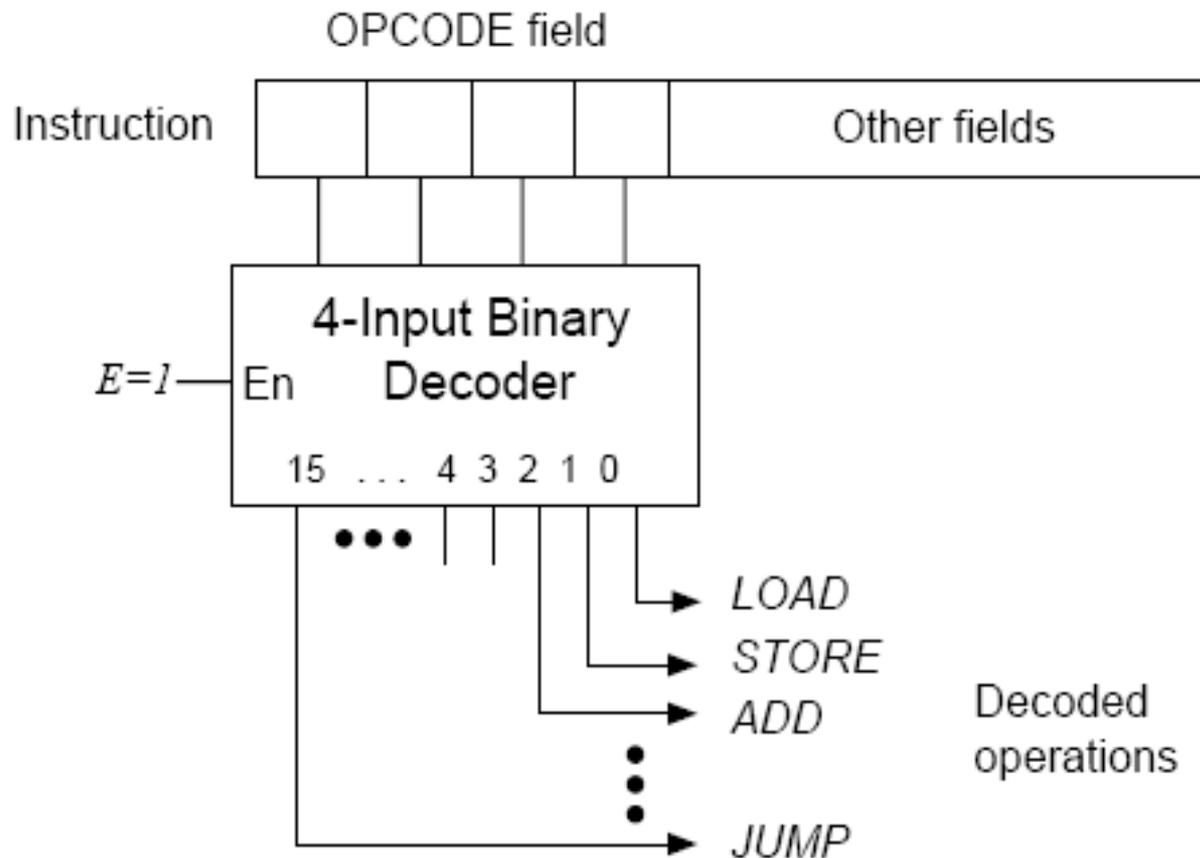


(a)



(b)

Decodificador



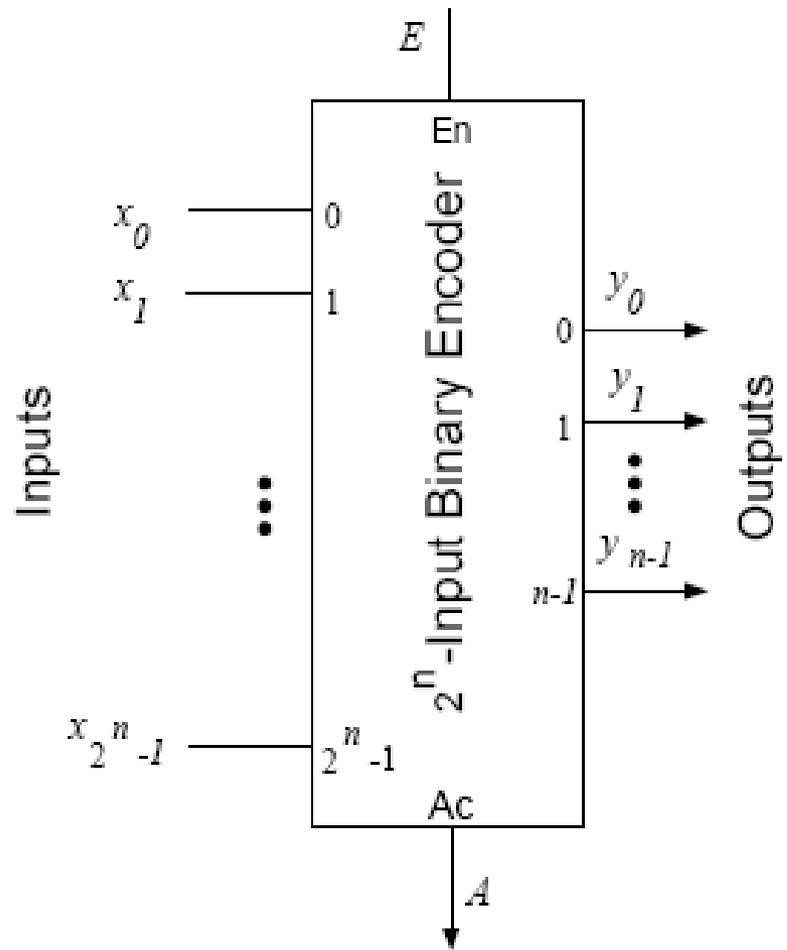
Decodificador

```
1  ENTITY fig9_54 IS
2  PORT(
3      a                :IN BIT_VECTOR (2 DOWNTO 0);
4      e3, e2bar, e1bar :IN BIT;
5      y                :OUT BIT_VECTOR (7 DOWNTO 0)
6  );
7  END fig9_54 ;
8  ARCHITECTURE truth OF fig9_54 IS
9  SIGNAL inputs: BIT_VECTOR (5 DOWNTO 0); --combina habilitações c/ entrada binária
10 BEGIN
11     inputs <= e3 & e2bar & e1bar & a;
12     WITH inputs SELECT
13         y <= "11111110" WHEN "100000", --Y0 ativa
14             "11111101" WHEN "100001", --Y1 ativa
15             "11111011" WHEN "100010", --Y2 ativa
16             "11110111" WHEN "100011", --Y3 ativa
17             "11101111" WHEN "100100", --Y4 ativa
18             "11011111" WHEN "100101", --Y5 ativa
19             "10111111" WHEN "100110", --Y6 ativa
20             "01111111" WHEN "100111", --Y7 ativa
21             "11111111" WHEN OTHERS; --desabilitadas
22     END truth;
```

FIGURA 9.54

Equivalente em VHDL ao decodificador 74138.

Codificador

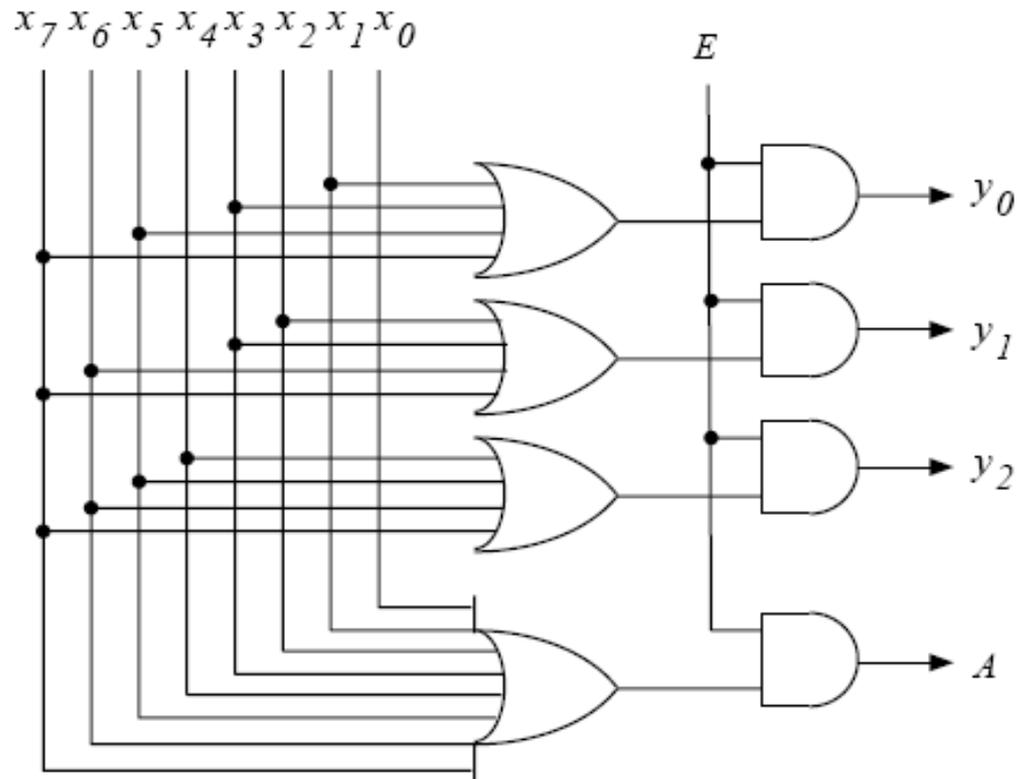


Codificador

E	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	y	y_2	y_1	y_0	A
1	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	0	0	1	0	1	0	0	1	1
1	0	0	0	0	0	1	0	0	2	0	1	0	1
1	0	0	0	0	1	0	0	0	3	0	1	1	1
1	0	0	0	1	0	0	0	0	4	1	0	0	1
1	0	0	1	0	0	0	0	0	5	1	0	1	1
1	0	1	0	0	0	0	0	0	6	1	1	0	1
1	1	0	0	0	0	0	0	0	7	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	-	-	-	-	-	-	-	-	0	0	0	0	0

Codificador

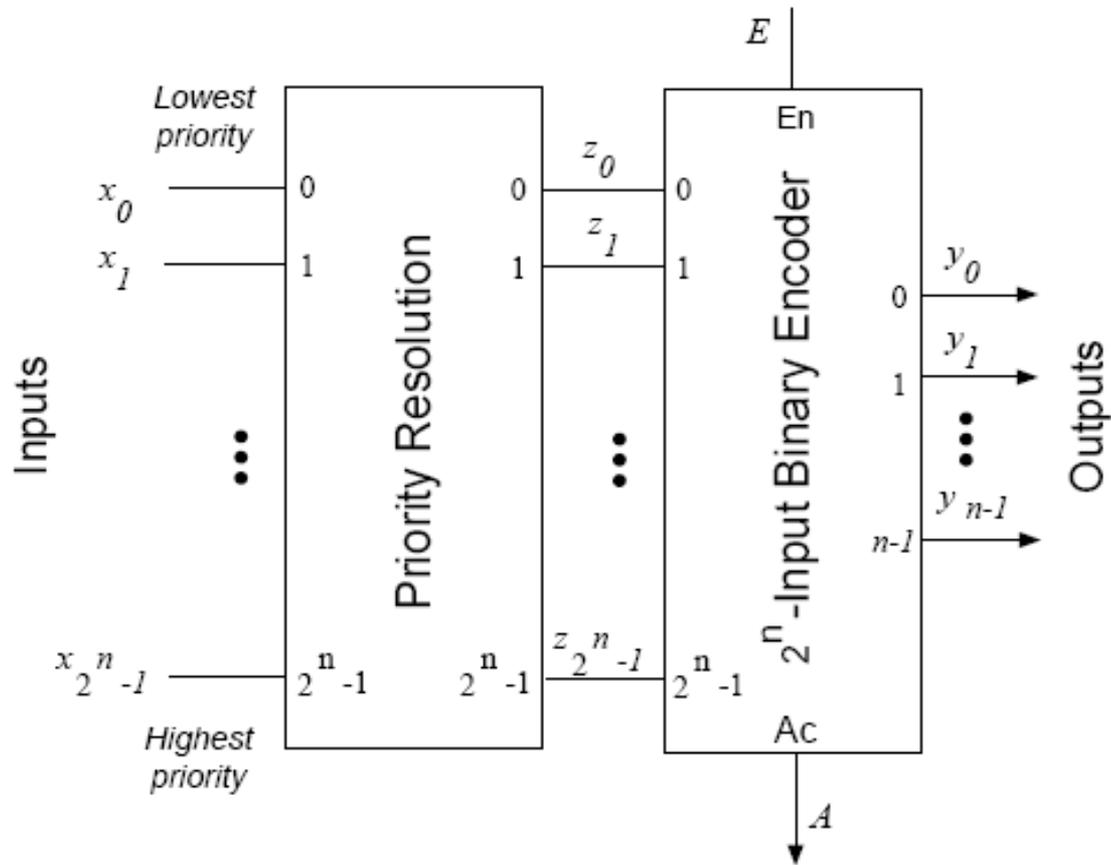
- Implementação de um codificador binário de 8 entradas



Codificador de Prioridade

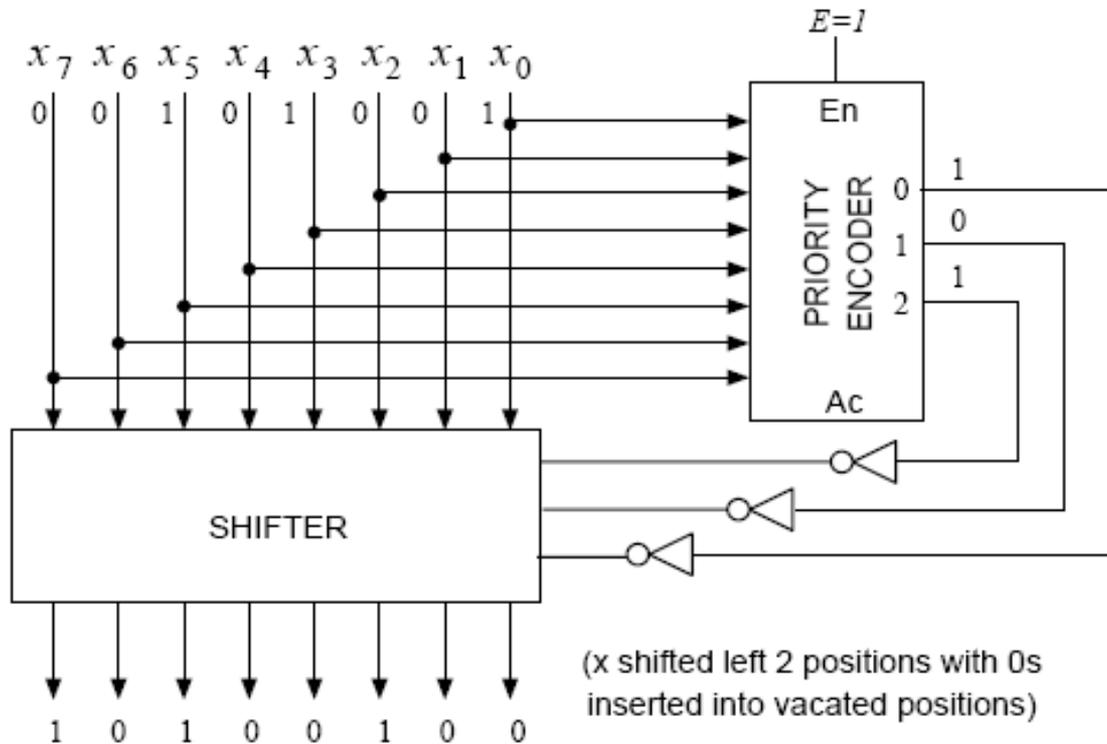
E	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	y_2	y_1	y_0	A
1	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	0	1	-	0	0	1	1
1	0	0	0	0	0	1	-	-	0	1	0	1
1	0	0	0	0	1	-	-	-	0	1	1	1
1	0	0	0	1	-	-	-	-	1	0	0	1
1	0	0	1	-	-	-	-	-	1	0	1	1
1	0	1	-	-	-	-	-	-	1	1	0	1
1	1	-	-	-	-	-	-	-	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0
0	-	-	-	-	-	-	-	-	0	0	0	0

Codificador de Prioridade



Codificador de Prioridade

- Detecção do bit mais à esquerda em '1' e remoção dos zeros precedentes



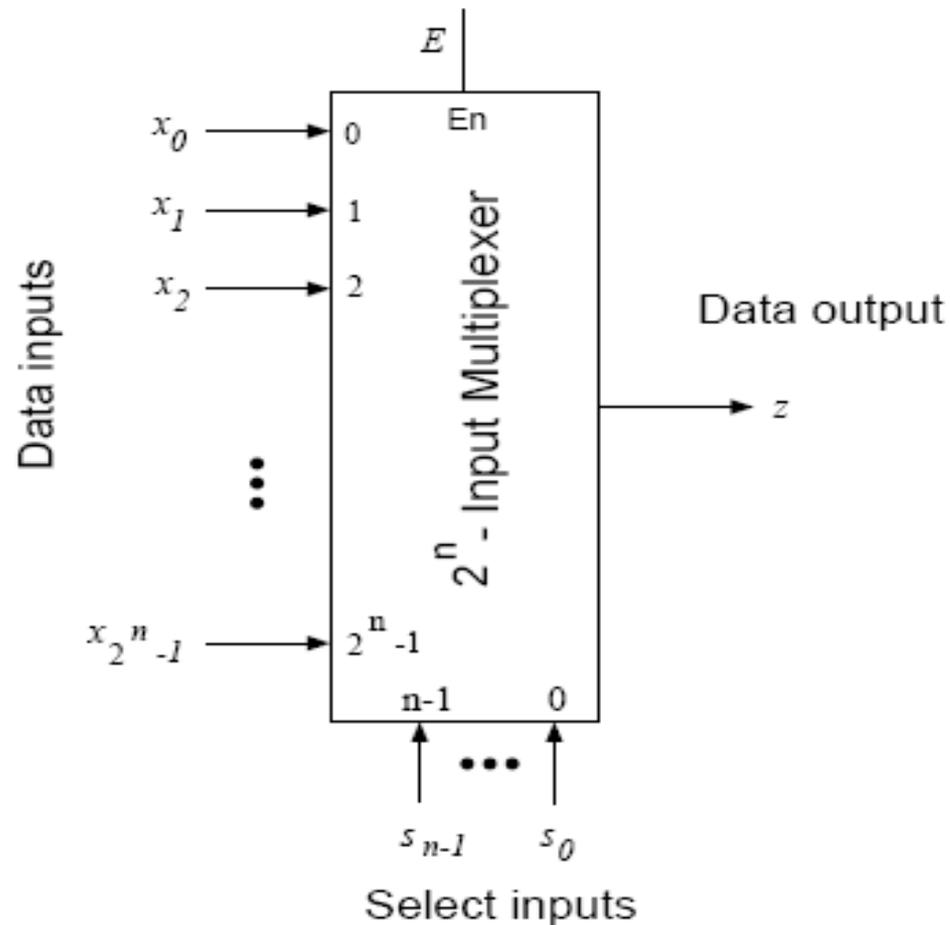
Codificador de Prioridade

```
1  LIBRARY  ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY fig9_60 IS
5  PORT(
6      sw    :IN BIT_VECTOR (9 DOWNT0 0);  -- lógica padrão não é necessária
7      oe    :IN BIT;                       -- lógica padrão não é necessária
8      d     :OUT STD_LOGIC_VECTOR (3 DOWNT0 0)  -- lógica padrão para alta impedância
9  );
10 END fig9_60;
11
12 ARCHITECTURE a OF fig9_60 IS
13 BEGIN
14     d <= "ZZZZ"  WHEN ((oe = '0') OR (sw = "1111111111")) ELSE
15         "1001"  WHEN sw(9) = '0' ELSE
16         "1000"  WHEN sw(8) = '0' ELSE
17         "0111"  WHEN sw(7) = '0' ELSE
18         "0110"  WHEN sw(6) = '0' ELSE
19         "0101"  WHEN sw(5) = '0' ELSE
20         "0100"  WHEN sw(4) = '0' ELSE
21         "0011"  WHEN sw(3) = '0' ELSE
22         "0010"  WHEN sw(2) = '0' ELSE
23         "0001"  WHEN sw(1) = '0' ELSE
24         "0000"  WHEN sw(0) = '0';
25 END a;
```

FIGURA 9.60

Codificador de prioridade usando atribuição de sinal condicional em VHDL.

Multiplexador



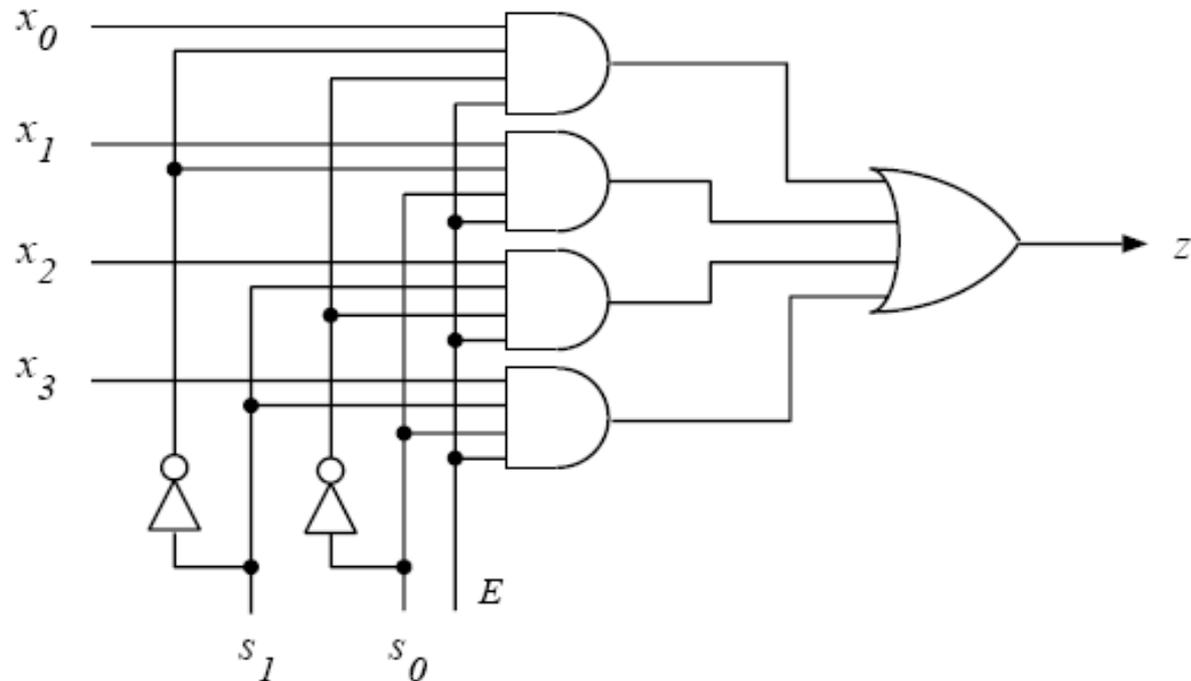
Multiplexador

- Multiplexador com 4 entradas

E	s_1	s_0	z
1	0	0	x_0
1	0	1	x_1
1	1	0	x_2
1	1	1	x_3
0	-	-	0

Multiplexador

- Implementação do multiplexador com 4 entradas

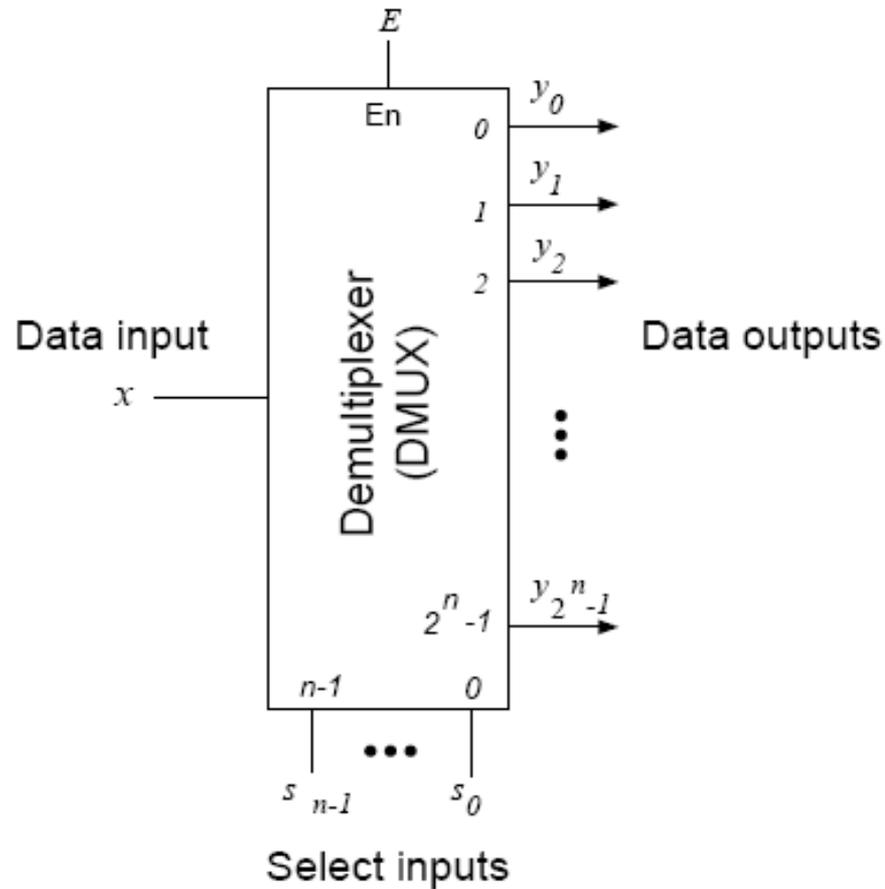


Multiplexador

```
1  ENTITY fig9_64 IS
2  PORT    (
3          ch0, ch1, ch2, ch3  :IN BIT_VECTOR (3 DOWNTO 0);
4          s                    :IN INTEGER RANGE 0 TO 3;
5          dout                 :OUT BIT_VECTOR (3 DOWNTO 0)
6      );
7  END fig9_64;
8
9  ARCHITECTURE selector OF fig9_64 IS
10     BEGIN
11         WITH s SELECT
12             dout <= ch0 WHEN 0,  -- seleciona canal 0 para a saída
13                  ch1 WHEN 1,  -- seleciona canal 1 para a saída
14                  ch2 WHEN 2,  -- seleciona canal 2 para a saída
15                  ch3 WHEN 3;  -- seleciona canal 3 para a saída
16     END selector;
```

FIGURA 9.64
MUX de quatro bits × quatro canais em VHDL.

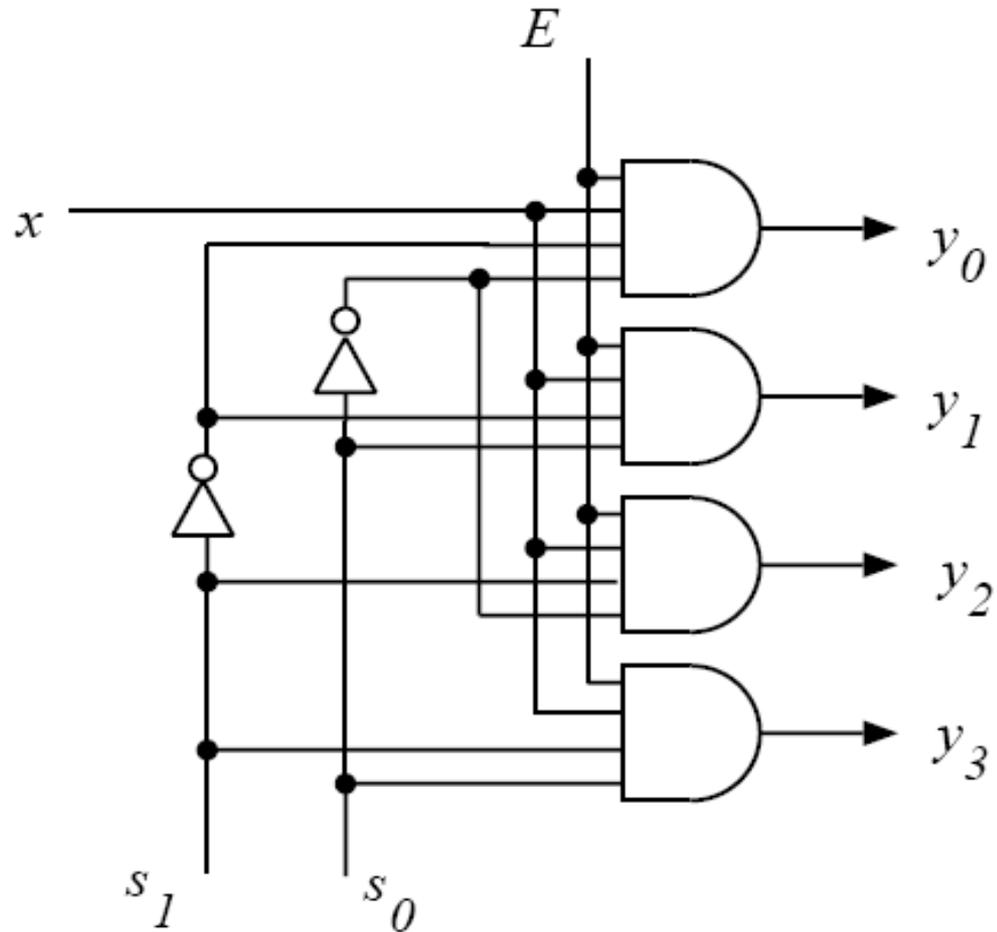
Demultiplexador



Demultiplexador

E	s_1	s_0	s	y_3	y_2	y_1	y_0
1	0	0	0	0	0	0	x
1	0	1	1	0	0	x	0
1	1	0	2	0	x	0	0
1	1	1	3	x	0	0	0
0	-	-	-	0	0	0	0

Demultiplexador

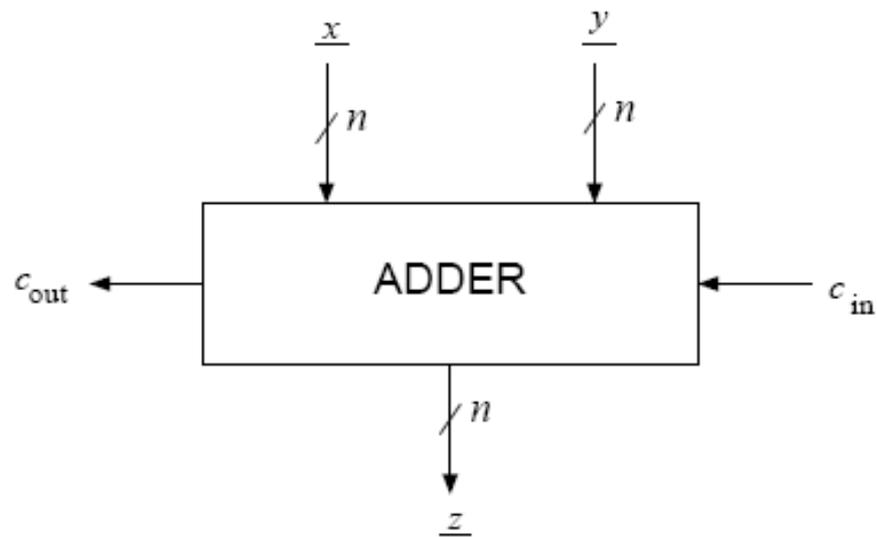


Demultiplexador

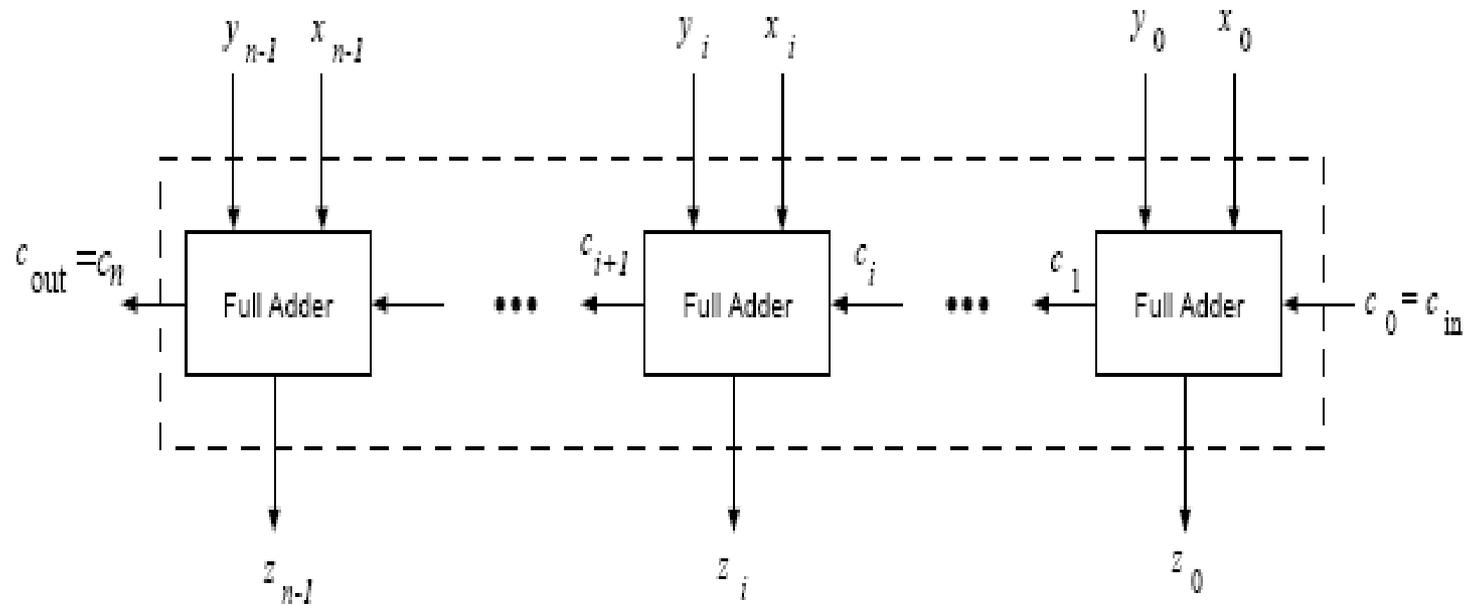
```
1  ENTITY fig9_65 IS
2  PORT (
3      s          :IN INTEGER RANGE 0 TO 3;
4      din        :IN BIT_VECTOR (3 DOWNTO 0);
5      ch0, ch1, ch2, ch3 :OUT BIT_VECTOR(3 DOWNTO 0)
6  );
7  END fig9_65;
8
9  ARCHITECTURE selector OF fig9_65 IS
10 BEGIN
11     ch0 <= din WHEN s = 0 ELSE "1111";
12     ch1 <= din WHEN s = 1 ELSE "1111";
13     ch2 <= din WHEN s = 2 ELSE "1111";
14     ch3 <= din WHEN s = 3 ELSE "1111";
15 END selector;
```

FIGURA 9.65
DEMUX de quatro bits × quatro canais em VHDL.

Somador



Somador

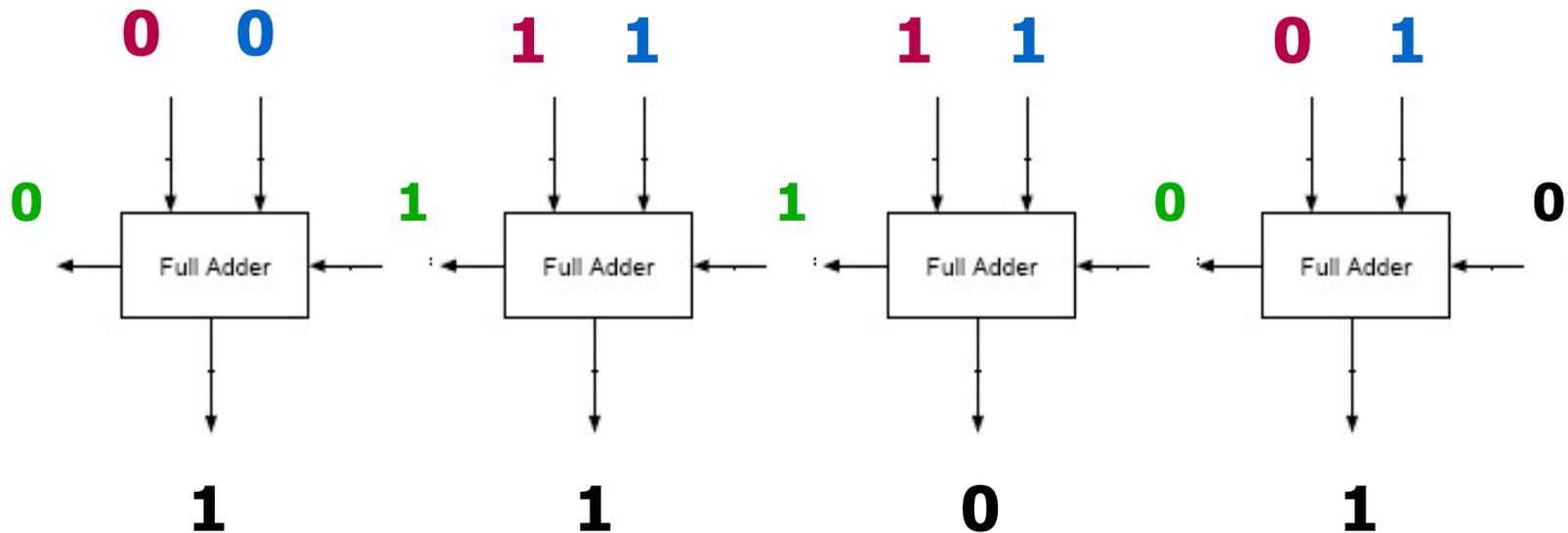


Somador

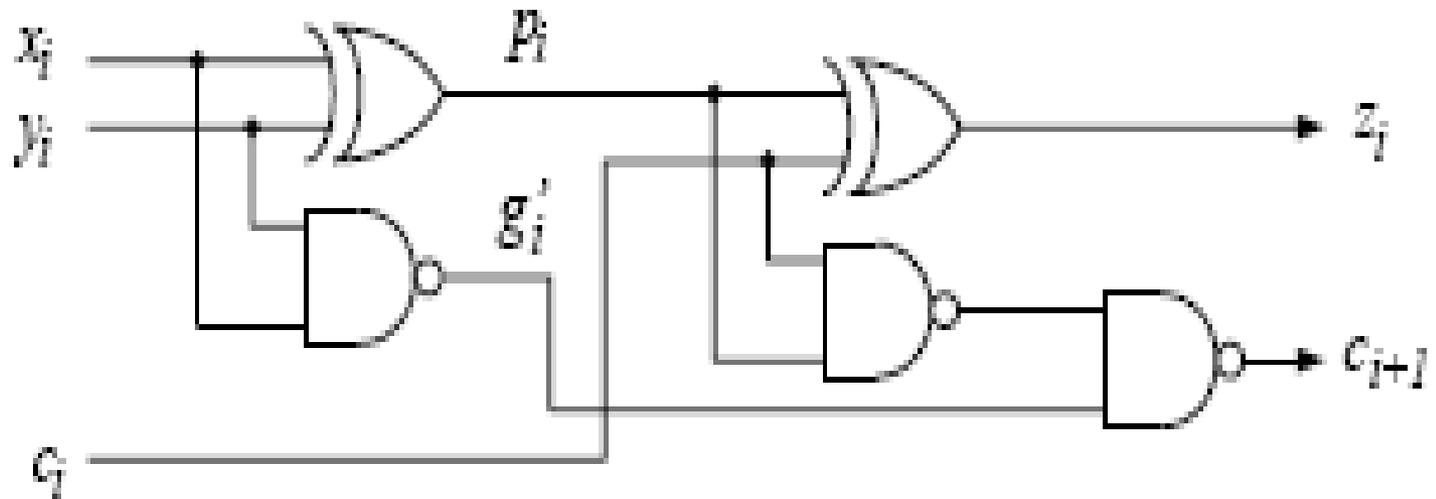
1 1 0
0 1 1 0
0 1 1 1

1 1 0 1

Somador



Somador



Somador

- Tabela da verdade

C_i	X_i	Y_i	Z_i	C_{i+1}
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

Somador

- Tabela da verdade

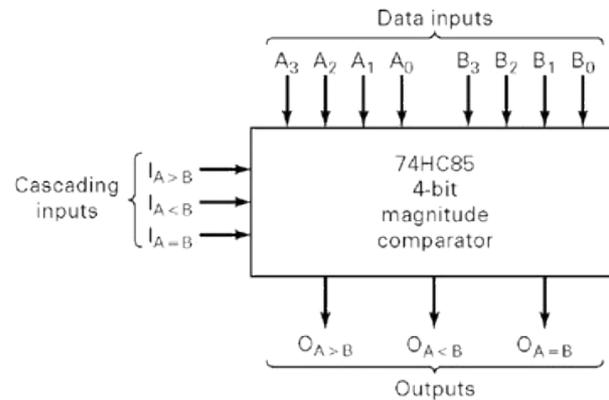
C_i	X_i	Y_i	Z_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Somador

```
1  ENTITY fig6_22 IS
2  PORT(
3      cin    :IN BIT;
4      a      :IN BIT_VECTOR(3 DOWNTO 0);
5      b      :IN BIT_VECTOR(3 DOWNTO 0);
6      s      :OUT BIT_VECTOR(3 DOWNTO 0);
7      cout   :OUT BIT);
8  END fig6_22;
9
10 ARCHITECTURE a OF fig6_22 IS
11     SIGNAL c :BIT_VECTOR (4 DOWNTO 0); -- carries exigem matrizes de 5 bits
12
13 BEGIN
14     c(0) <= cin;           -- Lê carry de entrada na matriz de bits
15     s <= a XOR b XOR c(3 DOWNTO 0); -- Gera soma dos bits
16     c(4 DOWNTO 1) <=      (a AND b)
17                          OR   (a AND c(3 DOWNTO 0))
18                          OR   (b AND c(3 DOWNTO 0));
19     cout <= c(4);        -- leva para a saída o carry do MSB.
20 END a;
```

FIGURA 6.22
Somador em VHDL.

Comparador de Magnitude

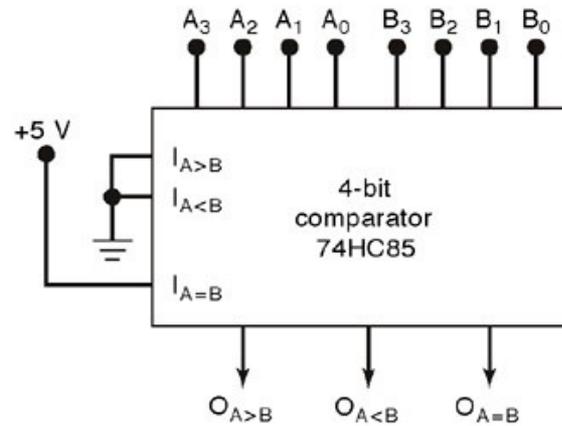


TRUTH TABLE

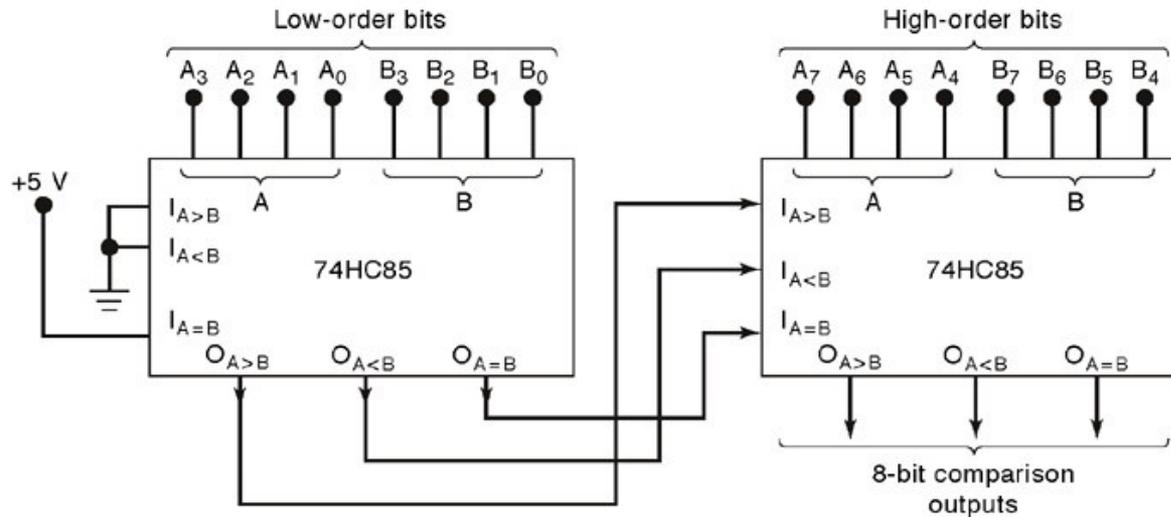
COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	X	X	H	L	L	H
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	L	H	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	H	L	L	L	L

H = HIGH Voltage Level
 L = LOW Voltage Level
 X = Immaterial

Comparador de Magnitude



(a)



(b)

Comparador de Magnitude

```
1  ENTITY fig9_67 IS
2  PORT ( a, b           : IN INTEGER RANGE 0 TO 15;
3         agtb, altb, aeqb : OUT BIT);
4  END fig9_67;
5
6  ARCHITECTURE vhd1 OF fig9_67 IS
7  BEGIN
8      PROCESS (a, b)
9      BEGIN
10         IF a < b THEN      altb <= '1'; agtb <= '0'; aeqb <= '0';
11         ELSIF a > b THEN   altb <= '0'; agtb <= '1'; aeqb <= '0';
12         ELSE               altb <= '0'; agtb <= '0'; aeqb <= '1';
13         END IF;
14     END PROCESS;
15 END vhd1;
```

FIGURA 9.67

Comparador de magnitude em VHDL.

Conversor BCD em Binário

```
1  ENTITY fig9_69 IS
2  PORT (  ones, tens  :IN INTEGER RANGE 0 TO 9;
3         binary       :OUT INTEGER RANGE 0 TO 99);
4  END fig9_69;
5
6  ARCHITECTURE vhdl OF fig9_69 IS
7  SIGNAL times10      :INTEGER RANGE 0 TO 90;
8  BEGIN
9      times10 <= tens * 10;
10     binary <= times10 + ones;
11 END vhdl;
```

FIGURA 9.69

Conversor de código BCD em binário em VHDL.

Decodificador/Driver BCD para 7-Segmentos

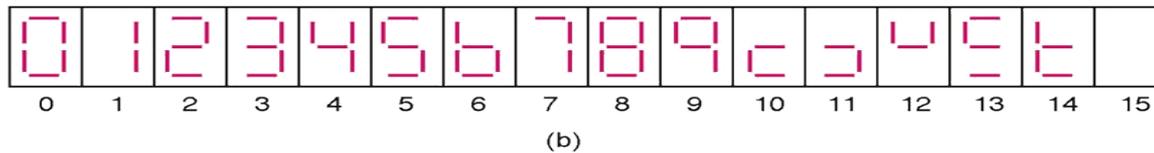
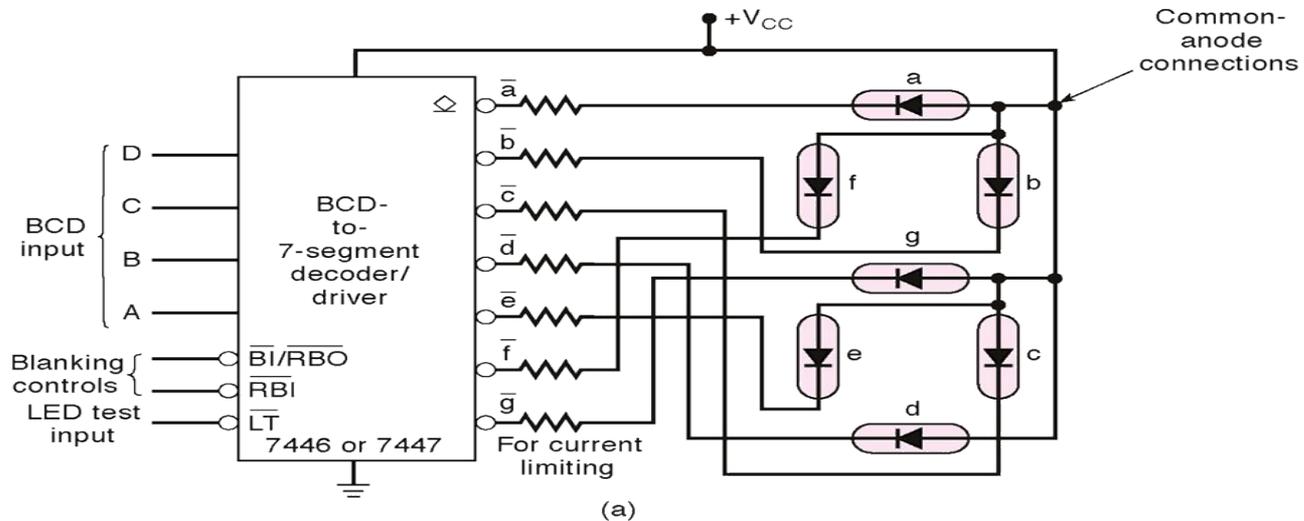
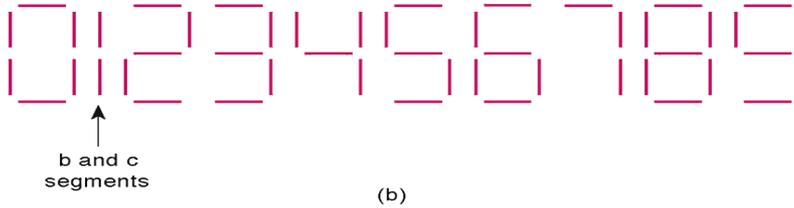
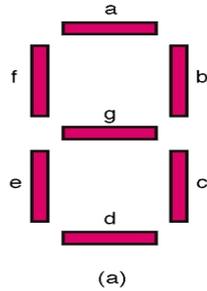
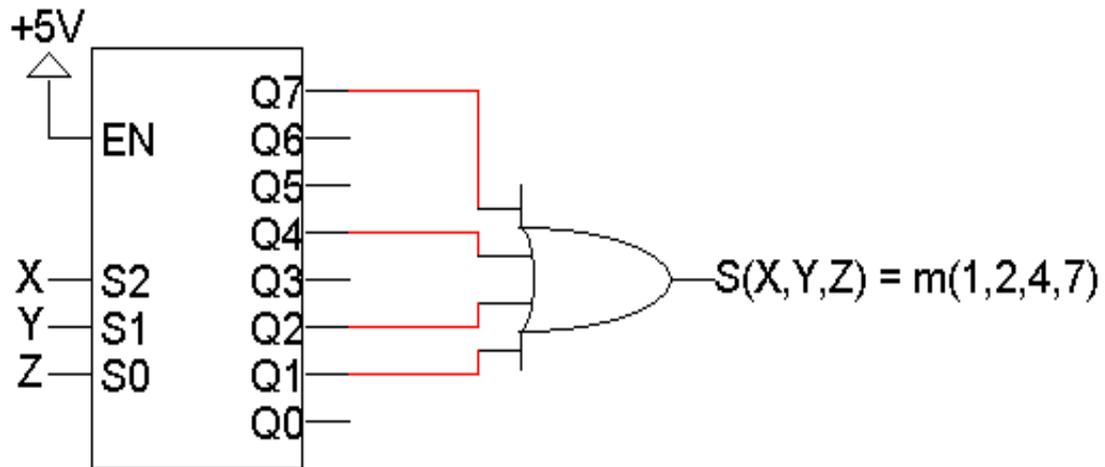
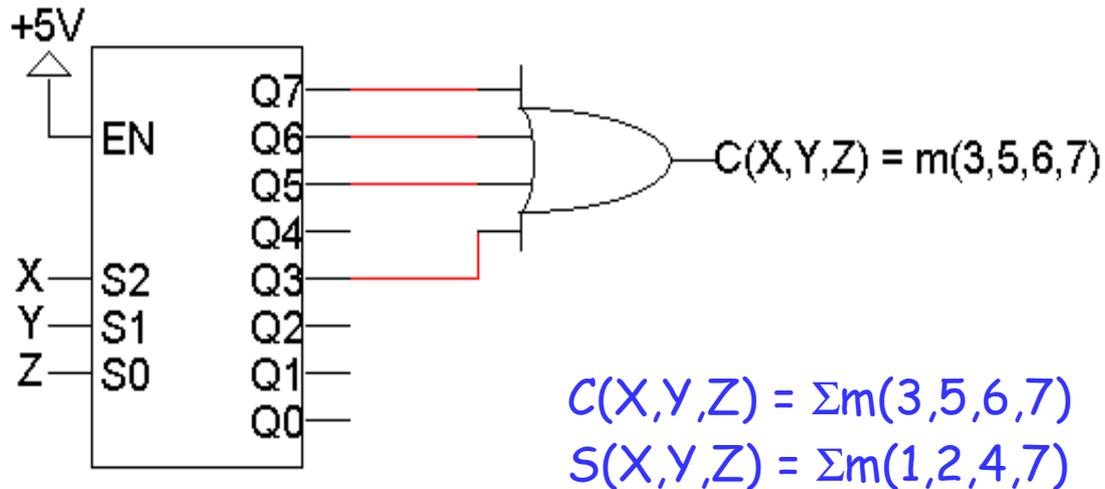


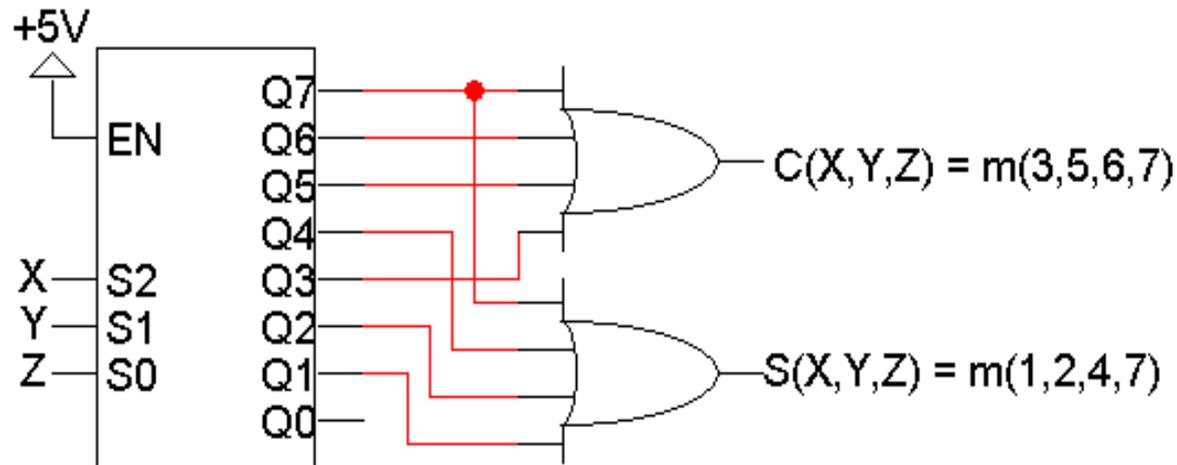
FIGURA 9.56
Decodificador-display BCD
para 7 segmentos em VHDL.

```
1 ENTITY fig9_56 IS
2 PORT (
3     bcd           :IN INTEGER RANGE 0 TO 15;
4     lt, bi, rbi   :IN BIT;
5     a,b,c,d,e,f,g,rbo :OUT BIT
6 );
7 END fig9_56 ;
8
9 ARCHITECTURE vhdl OF fig9_56 IS
10 BEGIN
11 PROCESS (bcd, lt, bi, rbi)
12 VARIABLE segments :BIT_VECTOR (0 TO 6);
13 BEGIN
14     IF bi = '0' THEN
15         segments := "1111111"; rbo <= '0'; -- apaga tudo
16     ELSIF lt = '0' THEN
17         segments := "0000000"; rbo <= '1'; -- testa segmentos
18     ELSIF (rbi = '0' AND bcd = 0) THEN
19         segments := "1111111"; rbo <= '0'; -- apaga 0s iniciais
20     ELSE
21         rbo <= '1';
22         CASE bcd IS -- display padrão Anodo Comum para 7 segmentos
23             WHEN 0 => segments := "0000001";
24             WHEN 1 => segments := "1001111";
25             WHEN 2 => segments := "0010010";
26             WHEN 3 => segments := "0000110";
27             WHEN 4 => segments := "1001100";
28             WHEN 5 => segments := "0100100";
29             WHEN 6 => segments := "1100000";
30             WHEN 7 => segments := "0001111";
31             WHEN 8 => segments := "0000000";
32             WHEN 9 => segments := "0001100";
33             WHEN OTHERS => segments := "1111111";
34         END CASE;
35     END IF;
36     a <= segments(0); --atribui bits de matriz a pinos de saída
37     b <= segments(1);
38     c <= segments(2);
39     d <= segments(3);
40     e <= segments(4);
41     f <= segments(5);
42     g <= segments(6);
43 END PROCESS;
44 END vhdl;
```

Funções Lógicas com Decodificador



Usando só um Decodificador



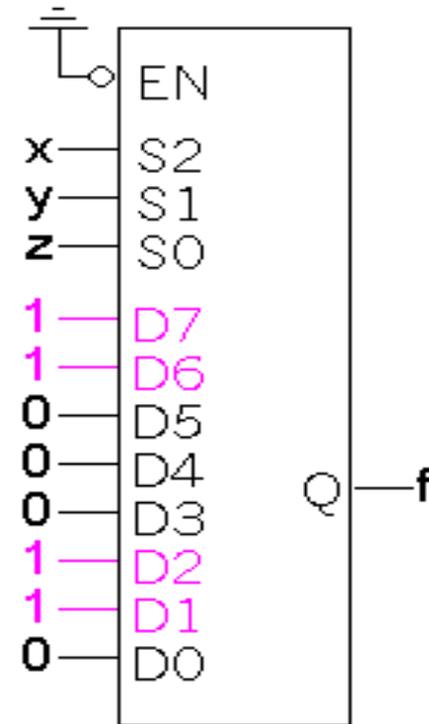
$$C(X,Y,Z) = \sum m(3,5,6,7)$$

$$S(X,Y,Z) = \sum m(1,2,4,7)$$

Funções Lógicas com Multiplexadores

- Uma maneira de implementar uma função com n variáveis é usar um multiplexador n -to-1
- Por exemplo, se $f(x,y,z) = \Sigma m(1,2,6,7)$.

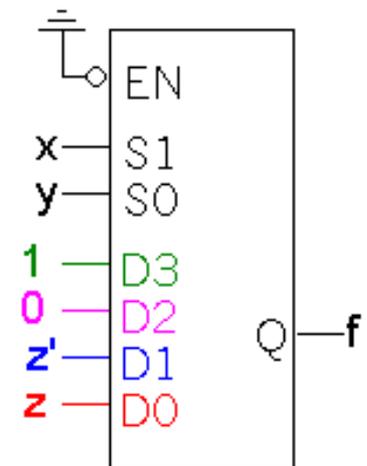
x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Economizando

- A função $f(x,y,z) = \sum m(1,2,6,7)$ pode ser implementada com um mux 4-to-1, ao invés de 8-to-1.
- Passo 1: Agrupe as linhas em pares com x e y com os mesmos valores, de modo que f é uma função de z apenas.
 - Quando $xy=00$, $f=z$
 - Quando $xy=01$, $f=z'$
 - Quando $xy=10$, $f=0$
 - Quando $xy=11$, $f=1$
- Passo 2: Conecte x e y às linhas de seleção e a variável z, 0 ou 1 às entradas, de acordo com a tabela da verdade.

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Funções Lógicas com Multiplexador

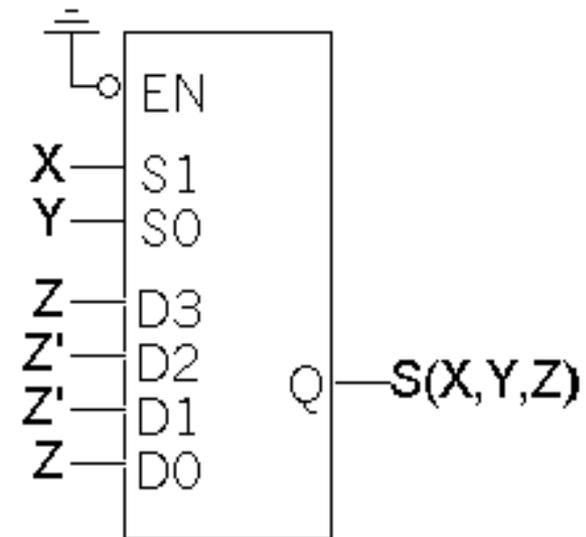
X	Y	Z	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Quando XY=00, S=Z

Quando XY=01, S=Z'

Quando XY=10, S=Z'

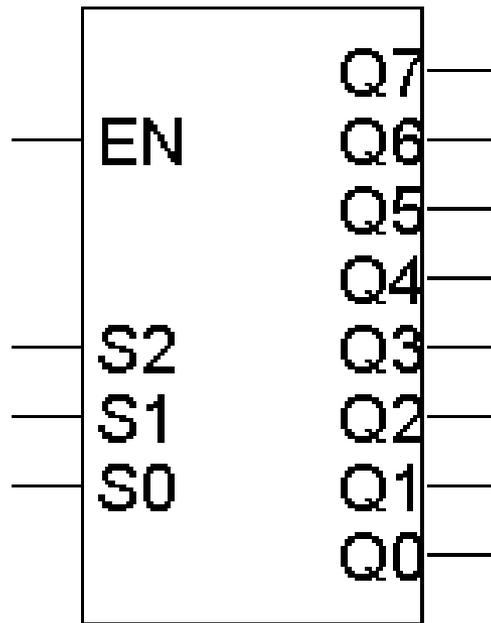
Quando XY=11, S=Z



$$\begin{aligned}
 S &= X' Y' D0 + X' Y D1 + X Y' D2 + X Y D3 \\
 &= X' Y' Z + X' Y Z' + X Y' Z' + X Y Z \\
 &= \Sigma m(1,2,4,7)
 \end{aligned}$$

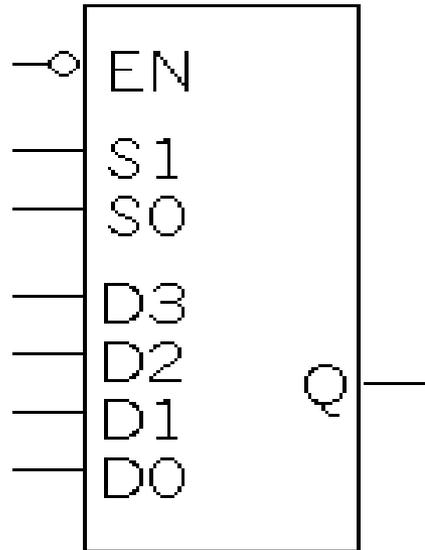
Exercício

- Implemente $F(x,y,z) = \sum m(1,3,4,5)$ usando um decodificador.



Exercício

- Implementar $F(x,y,z) = \sum m(1,3,4,5)$ usando um multiplexador.



Deslocador

- O deslocador serve para mover um conjunto de bits de uma ou mais posições para a esquerda ou direita.
- Dependendo do tipo de deslocamento, podem ser inseridos '0's para as posições que ficam vagas à medida que os bits correspondentes vão sendo deslocados.
- Para os números cuja representação coloca o bit de sinal no bit mais à esquerda, normalmente esse bit é replicado quando os bits são deslocados para a direita.

Deslocador

- Deslocamento para a direita de 2 bits:

00001111 --> 00000011

- Deslocamento para esquerda de 3 bits

00001111 --> 01111000

- Deslocamento para a direita de 2 bits de valor negativo em complemento a dois:

10001111 --> 11100011

Deslocador

- Note que para cada bit deslocado para a direita, corresponde a uma divisão inteira por 2:

$$\begin{array}{rcl} 00011100 & \text{-->} & 00000111 \\ 28 & \text{-->} & 7 \end{array}$$

- E para cada bit deslocado para a esquerda, corresponde a uma multiplicação por 2:

$$\begin{array}{rcl} 00001111 & \text{-->} & 01111000 \\ 15 & \text{-->} & 120 \end{array}$$

Deslocador

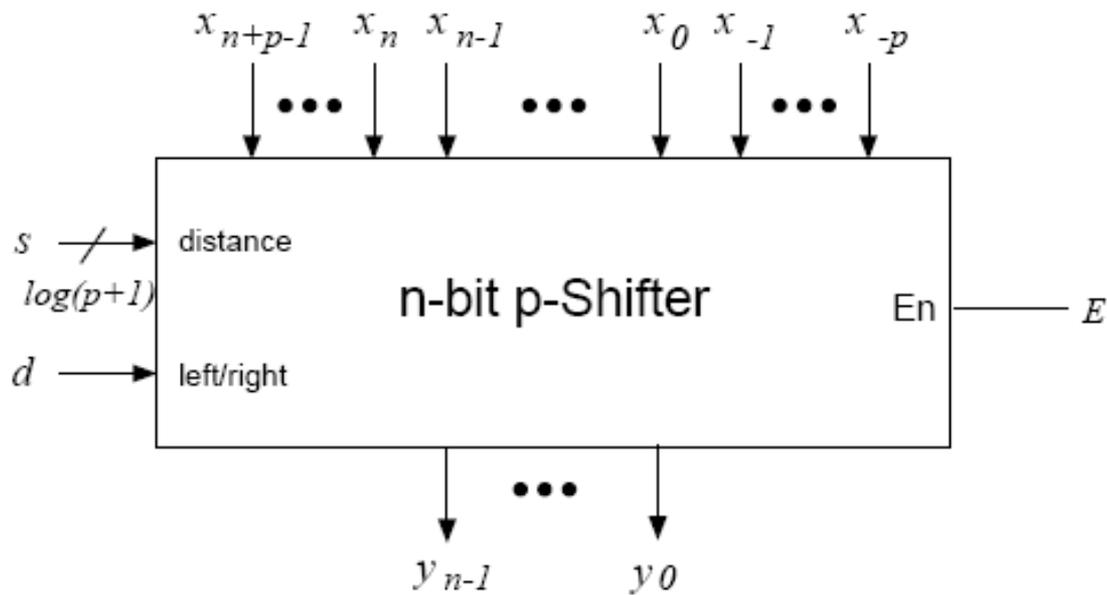


Figure 9.30: n-BIT *p*-SHIFTER.