Circuitos Lógicos e Organização de Computadores

Capítulo 6 – Blocos com Circuitos Combinacionais

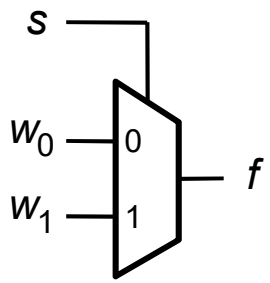
Ricardo Pannain

pannain@unicamp.br

Blocos Básicos com Circuitos Combinacionais

- Projetar:
  - Multiplexador 2-para-1
  - Multiplexador 4-para-1
  - Multiplexador 4-para-1construído a partir de multiplexadores 2-para-1
  - Multiplexador 16-para-1construído a partir de multiplexadores 4-para-1

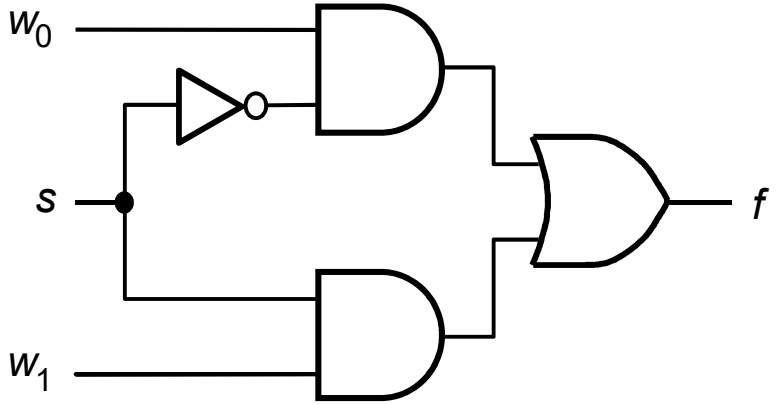
Multiplexador 2-para-1



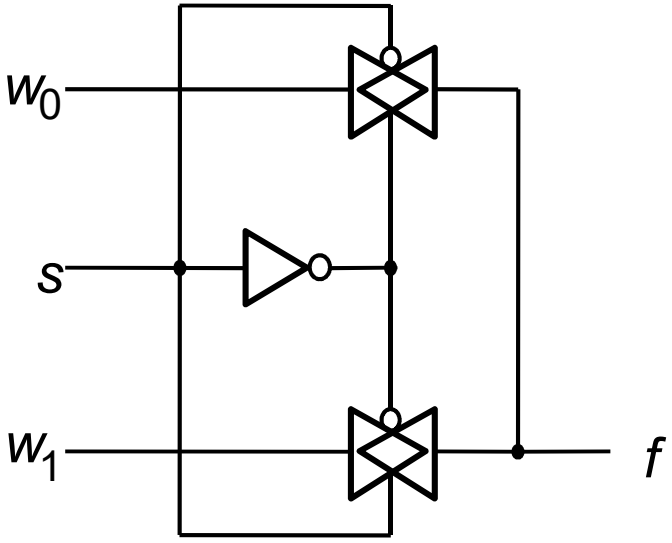
(a) Símbolo Gráfico

s	f
0	w_0
1	w_1

(b) Tabela Verdade

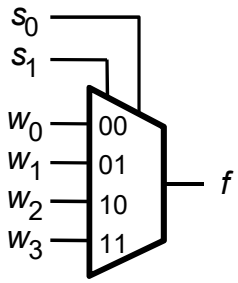


(c) Circuito SOP



(d) Circuito com Transmission Gate

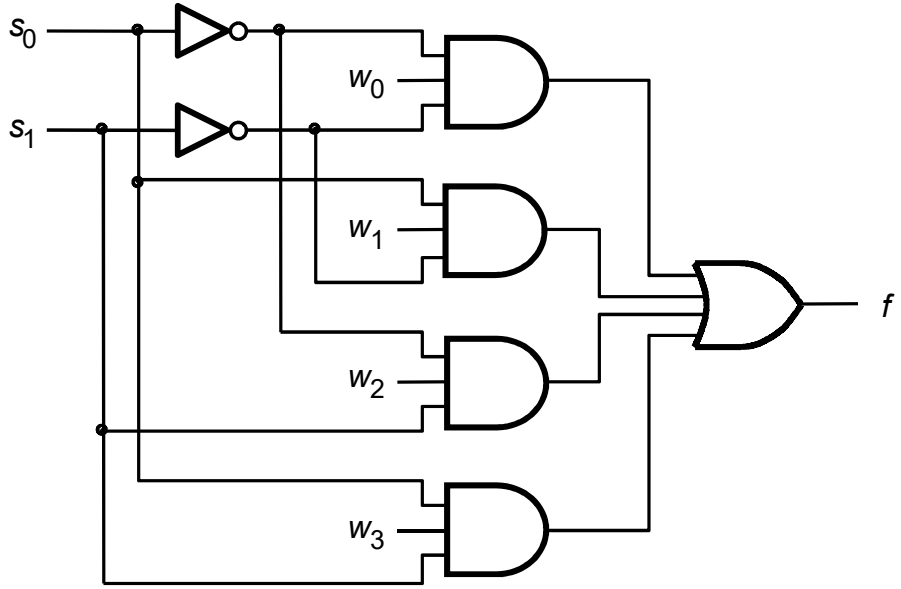
Multiplexador 4-para-1



(a) Símbolo Gráfico

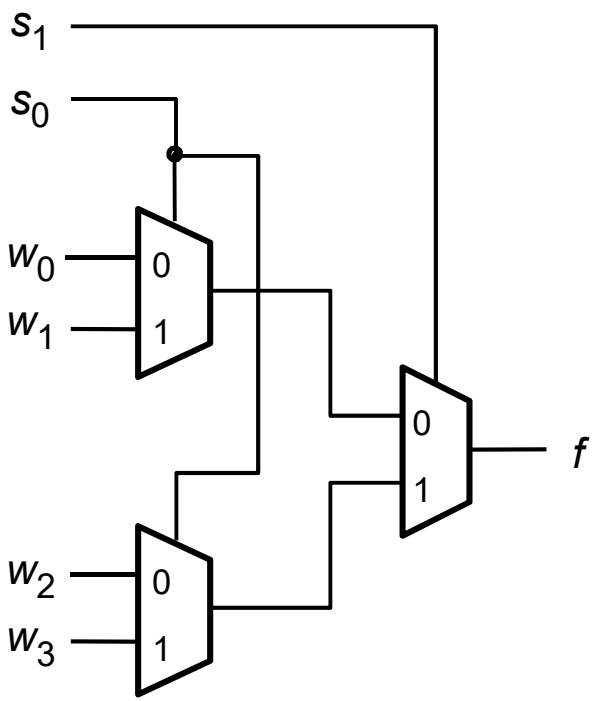
s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Tabela Verdade



(c) Circuito

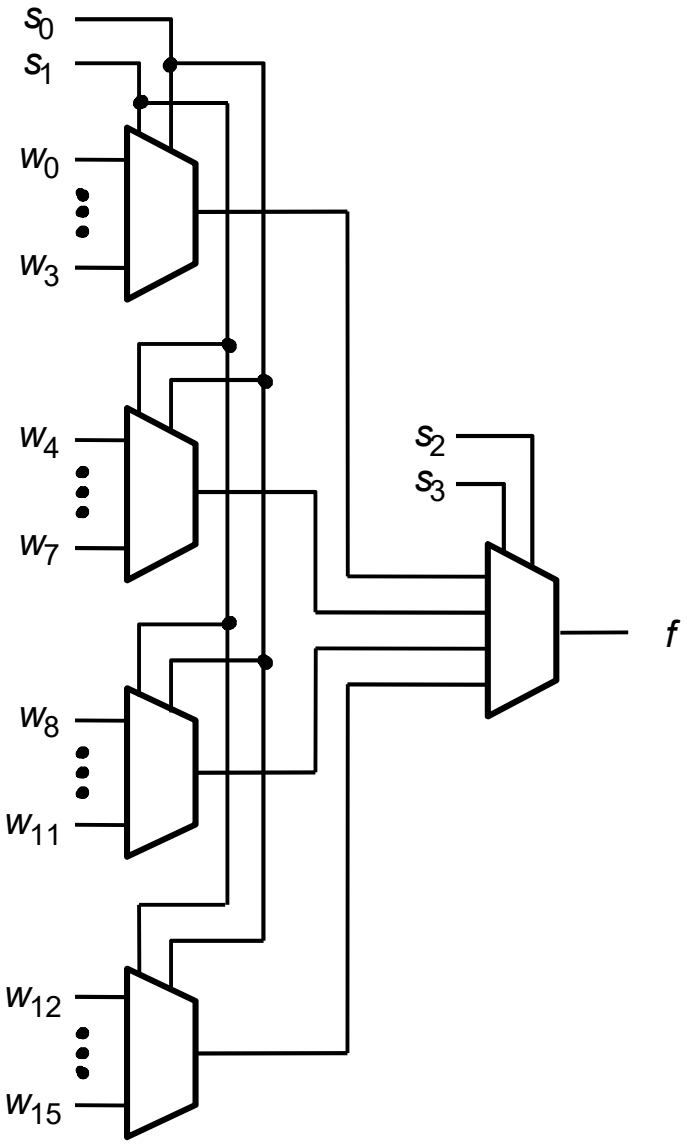
Multiplexador 4-para-1construído a partir de multiplexadores 2-para-1



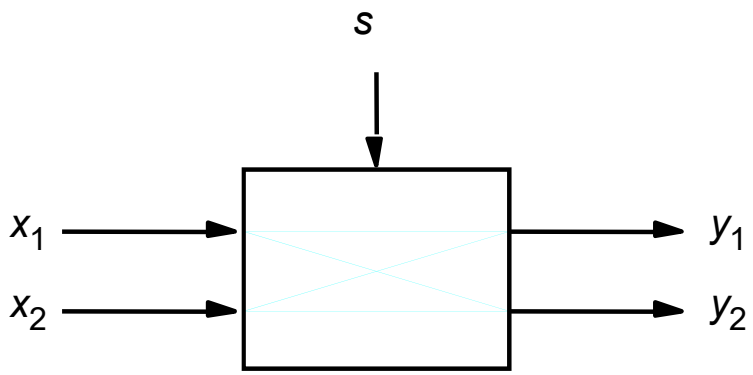
s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Tabela Verdade

Multiplexador 16-para-1construído a partir de multiplexadores 4-para-1

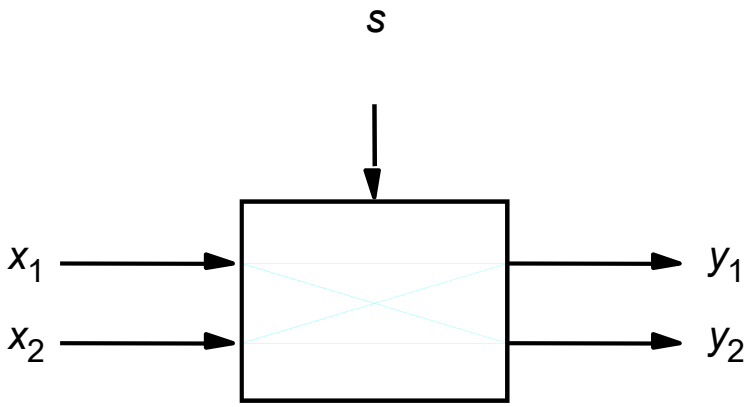


Aplicação prática de multiplexadores

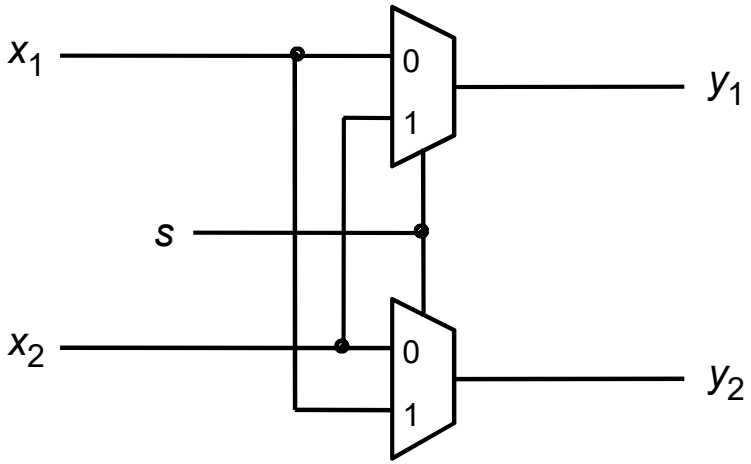


(a) Uma chave crossbar 2x2

Aplicação prática de multiplexadores



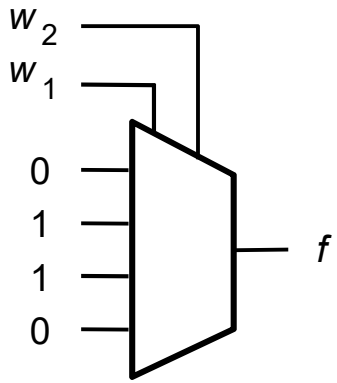
(a) Uma chave crossbar 2x2



(b) Implementação com multiplexadores

Síntese de uma função lógica usando multiplexadores

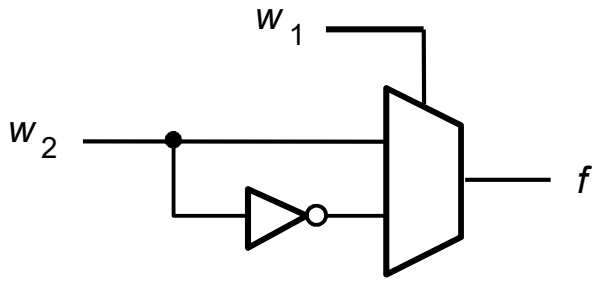
w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0



(a) Implementação usando um multiplexador 4-para-1

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0

w_1	f
0	w_2
1	$\overline{w}_2$



(b) Tabela verdade modificada

(c) Circuito

Síntese de uma função lógica de 3 entradas usando multiplexadores

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

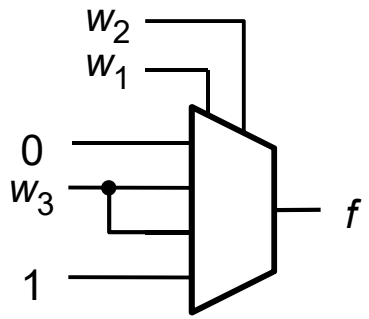
(a) Tabela verdade modificada

Síntese de uma função lógica de 3 entradas usando multiplexadores

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1

(a) Tabela verdade modificada



(b) Circuito

Função XOR de 3 entradas

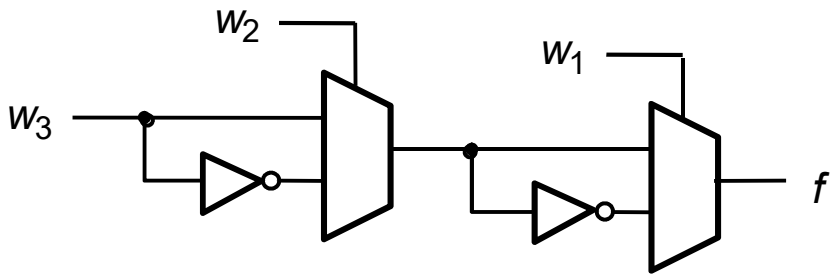
w_1	w_2	w_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(a) Tabela Verdade

Função XOR de 3 entradas

w_1	w_2	w_3	f	
0	0	0	0	$w_2 \oplus w_3$
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	$\overline{w_2 \oplus w_3}$
1	0	1	0	
1	1	0	0	
1	1	1	1	

(a) Tabela Verdade



(b) Circuito

x = w

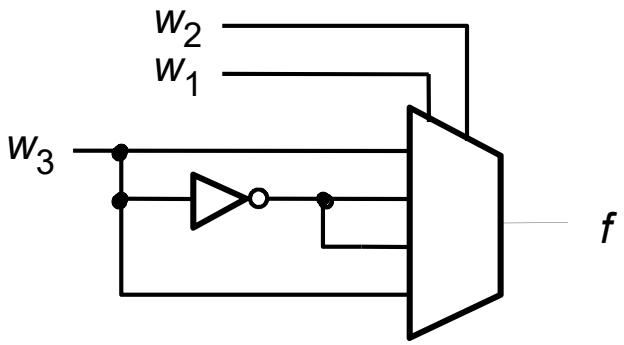
f = x1’(x2’x3 + x2x3’) + x1 (x2x3 + x2’x3’) = x1 XOR x2 XOR x3 =

= (x1 XOR x2) XOR x3

Função XOR de 3 entradas

w_1	w_2	w_3	f	
0	0	0	0	w_3
0	0	1	1	
0	1	0	1	$\bar{w}_3$
0	1	1	0	
1	0	0	1	$\bar{w}_3$
1	0	1	0	
1	1	0	0	w_3
1	1	1	1	

(a) Tabela Verdade



(b) Circuito

Teorema de Shannon

$$f(w1,w2,...,wn) = \overline{w1} \cdot \underbrace{f(0,w2,...wn)}_{\text{co-fator}} + w1 \cdot f(1,w2,...wn)$$

$$f(w1,w2,...,wn) = \overline{w}_i \, f_{\overline{w}i} + w_i \, f_{wi}$$

Exercícios:

1) f(w1,w2,w3) = w1w2 + w1w3 + w2w3

2)Para xor de 3 entradas: f = w1 xor (w2 xor w3)

3) f = w1’w3’+w1w2+w1w3

Síntese de uma função lógica de 3 entradas usando multiplexadores

Exercícios:

1)

f(w1,w2,w3) = w1w2 + w1w3 + w2w3

f = w1' (w2w3) + w1(w2+w3)

Obs:

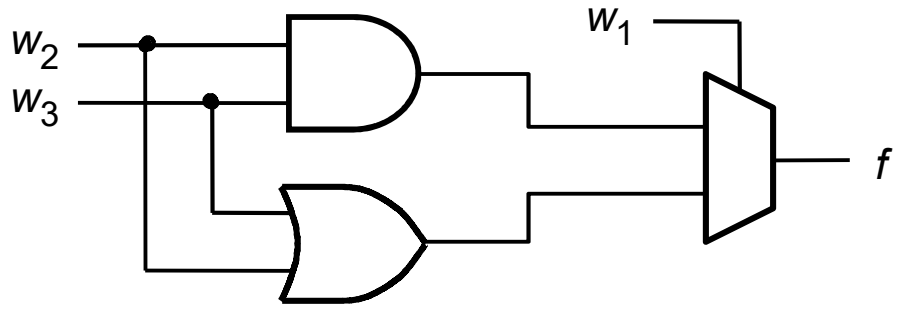
fw1= w2+w3+w2w3 = w2 (1+w3) + w3 (1+ w2) =

= w2 + w3

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	w_2w_3
1	$w_2 + w_3$

(b) Tabela Verdade



(b) Circuito

1) f(w1,w2,w3) = w1w2 + w1w3 + w2w3

fw1' = w2w3
fw1 = w2 + w3 + w2w3 = w2 + w3

f = w1' (w2w3) + w1 (w2 +w3)

fw1'w2' = 0
fw1'w2 = w3
fw1w2' = w3
fw1w2 = 1

Exercícios:

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	w_2w_3
1	$w_2 + w_3$

1) f(w1,w2,w3) = w1w2 + w1w3 + w2w3

Exercícios:

1) f(w1,w2,w3) = w1w2 + w1w3 + w2w3

fw1' = w2w3
fw1 = w2 + w3 + w2w3 = w2 + w3

f = w1' (w2w3) + w1 (w2 +w3)

fw1'w2' = 0
fw1'w2 = w3
fw1w2' = w3
fw1w2 = 1

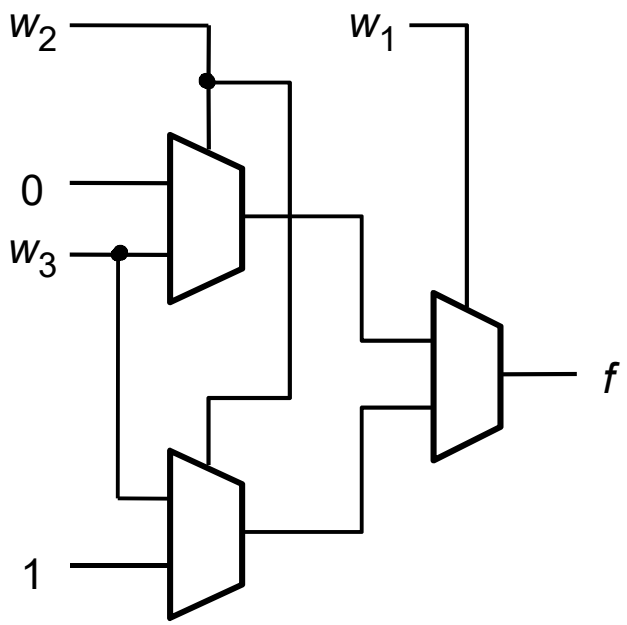
fw1'w2' w3'= 0
fw1'w2'w3 = 0
fw1'w2w3' = 0
fw1'w2 w3= 1
fw1w2'w3' = 0
fw1w2'w3 = 1
fw1w2w3'= 1
fw1w2w3 = 1

Exercícios:

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	w_2w_3
1	$w_2 + w_3$

Exemplos de circuitos com multiplexadores



2) Para xor de 3 entradas:

f = w1 xor (w2 xor w3)

$f = w1' (w2 \text{ xor } w3) + w1 \overline{(w2 \text{ xor } w3)}$

$f = w1' (w2'w3 + w2w3') + w1 (w2'w3' + w2w3)$

$f = w1'w2'(g) + w1' w2 (h) + w1w2' (i) + w1w2(j)$

$g = w3 = fw1'w2'$

$h = w3' = fw1'w2$

$i = w3' = fw1w2'$

$j = w3 = fw1w2$

3) f = w1’w3’+w1w2+w1w3

Em função de w1

$f_{w1’}$= w3’

f_{w1}= w2 + w3

Em função de w1 e w2

$f_{w1’w2’}$= w3’

$f_{w1’w2}$= w3’

$f_{w1w2’}$= w3

f_{w1w2}= 1

Em função de w1 e w2 e w3

$f_{w1’w2’w3’}$= 1

$f_{w1’w2’w3}$= 0

$f_{w1’w2w3’}$= 1

$f_{w1’w2w3}$= 0

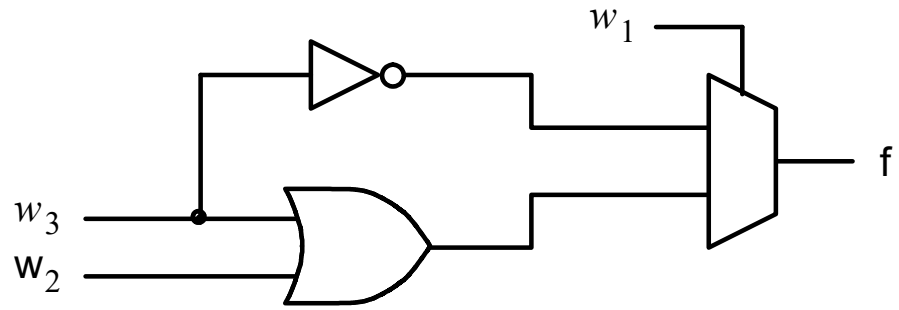
$f_{w1w2’w3’}$= 0

$f_{w1w2’w3}$= 1

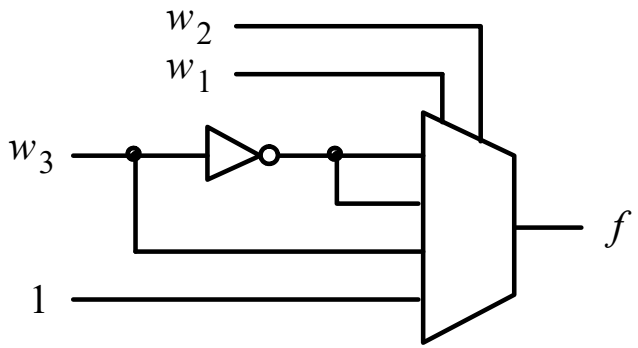
$f_{w1w2w3’}$= 1

f_{w1w2w3}= 1

Exemplos de circuitos com multiplexadores



(a) Using a 2-to-1 multiplexer



(b) Using a 4-to-1 multiplexer

Exemplo

Em função de w1 e w2

f = w1w2+w1w3+w2w3

$f_{w1'} = w2w3$

$f_{w1} = w2 + w3$

OBS: W2 + W3 + W2W3 = W2(1+W3) +
+ W3(1+W2) = W2 +W3

$f_{w1'w2'} = 0$

$f_{w1'w2} = w3$

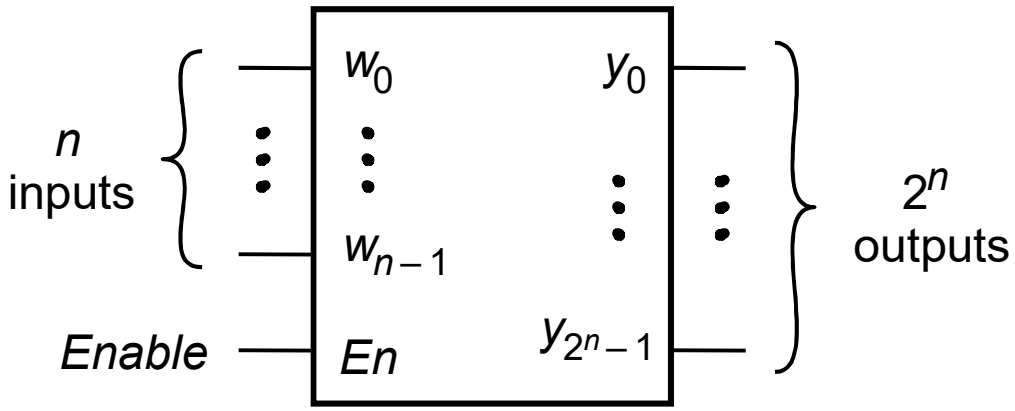
$f_{w1w2'} = w3$

$f_{w1w2} = 1$

Decodificador n-para-2^n

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Tabela Verdade



Se Enable = 0 → yi = 0; para qualquer w1w0

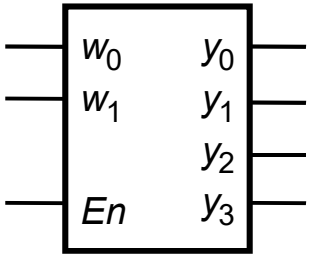
Se Enable = 1 → w1w0 = 00 → só y0 = 1; 01 → só y1 = 1;

w1w0 = 10 → só y2 = 1 e 11 → só y3 = 1

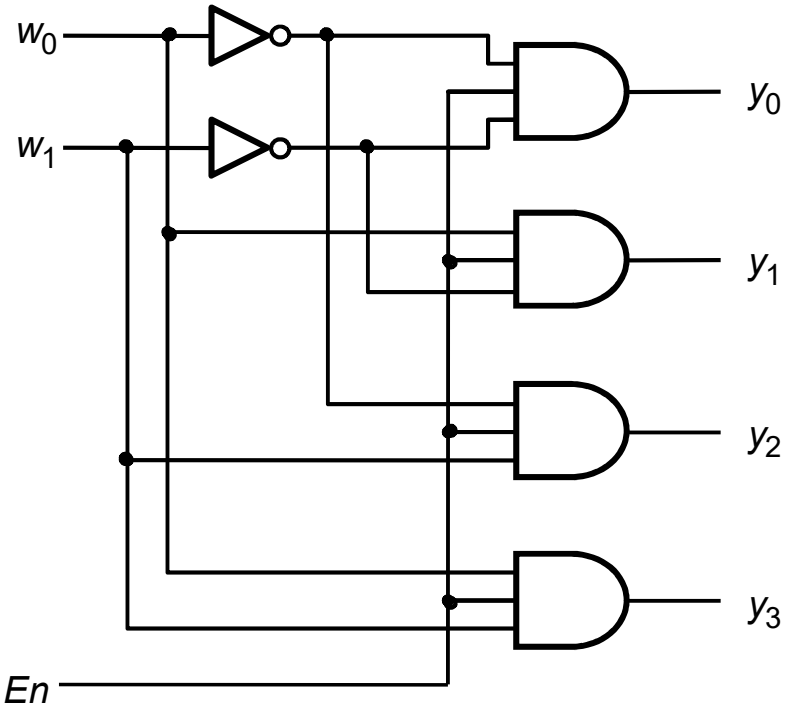
Decodificador 2-para-4

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Tabela Verdade



(b) Símbolo Gráfico

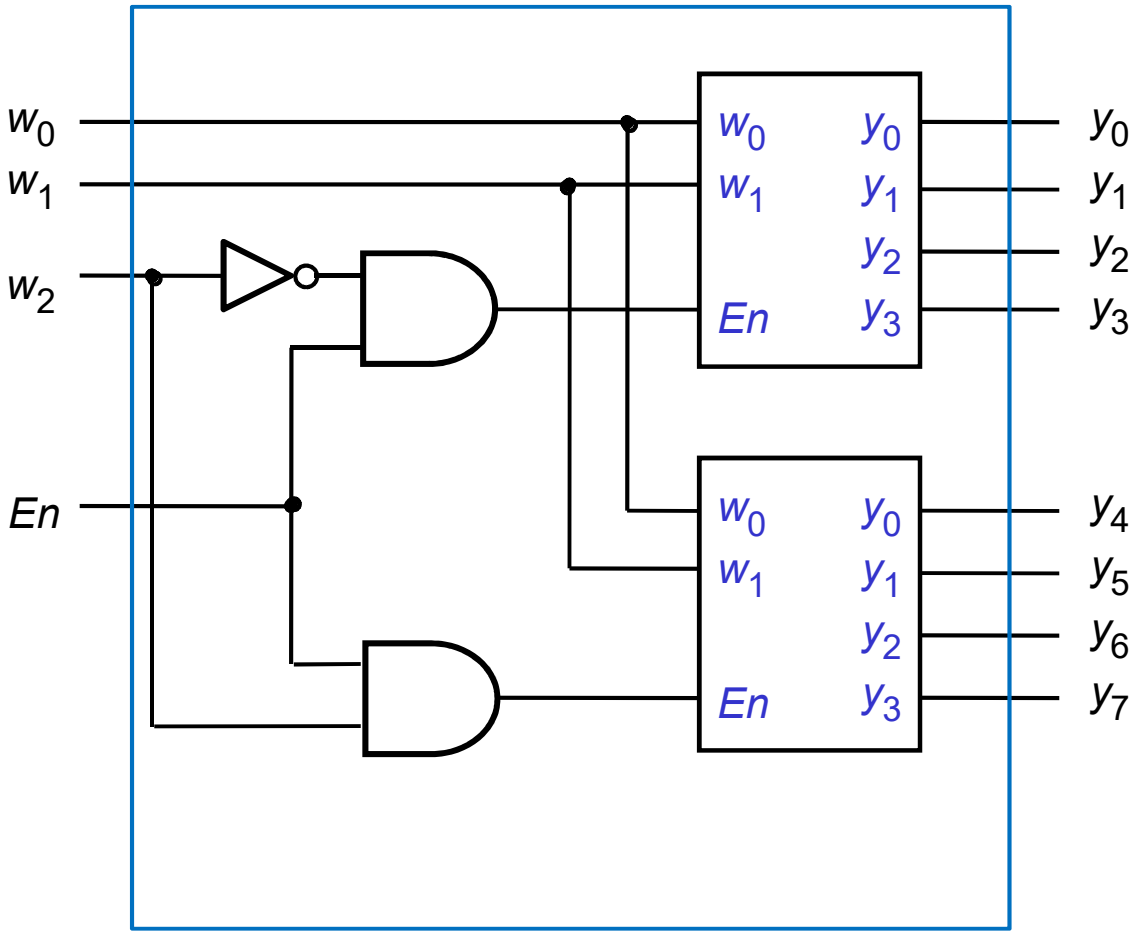


(c) Circuito Lógico

1-Decodificador 3 para 8, utilizando decodificadores 2 para 4

w2
0 decod 1 - entradas w1 w0
1 decod 2 - entradas w1 w0

Decodificador 3-para-8 usando dois decodificadores 2-para-4



2 – Decodificador 4 para 16, com decodificadores 2 para 4

w3 w2

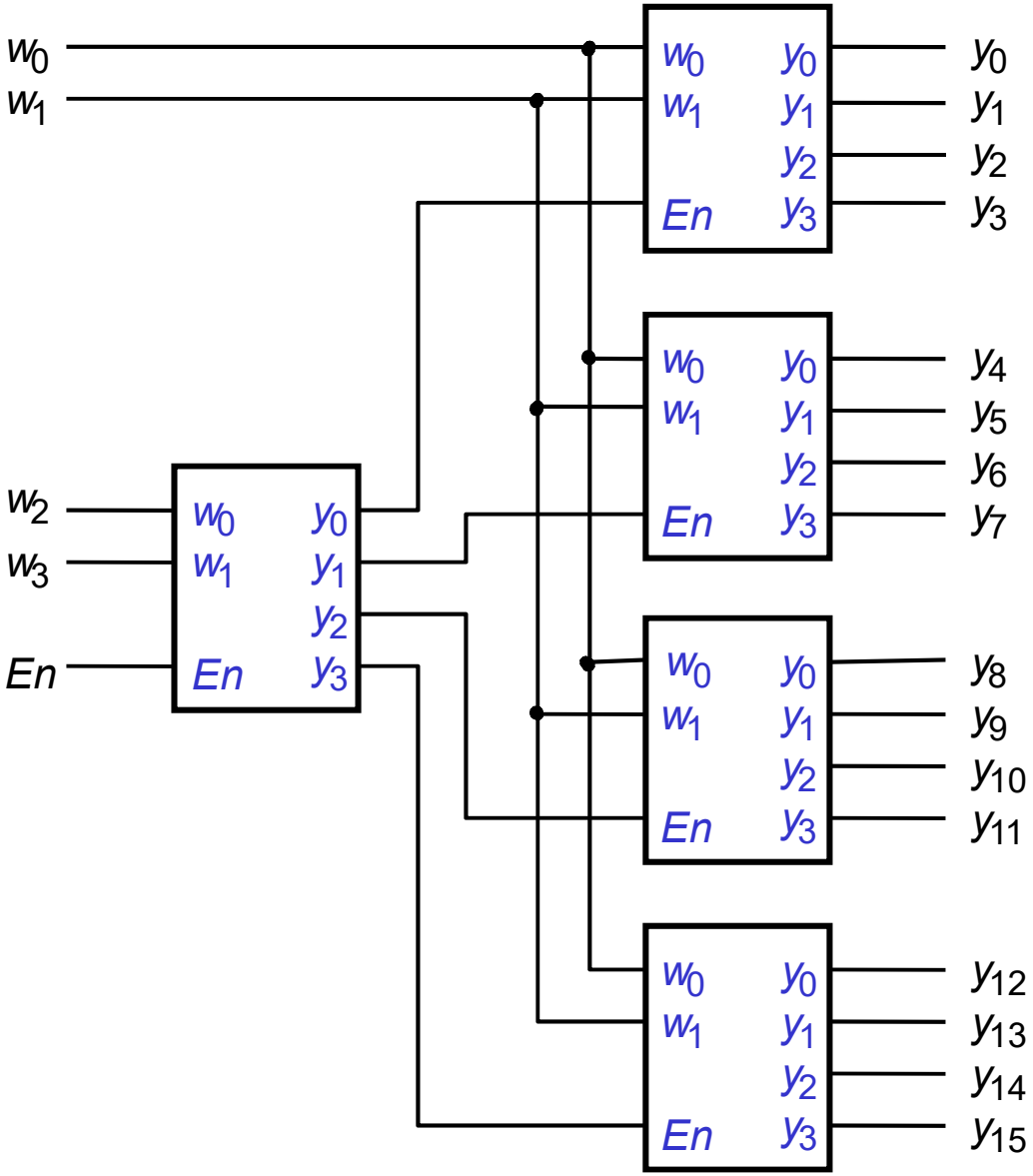
0 0 decod1 - entradas w1 w0

0 1 decod2 - entradas w1 w0

1 0 decod3 - entradas w1 w0

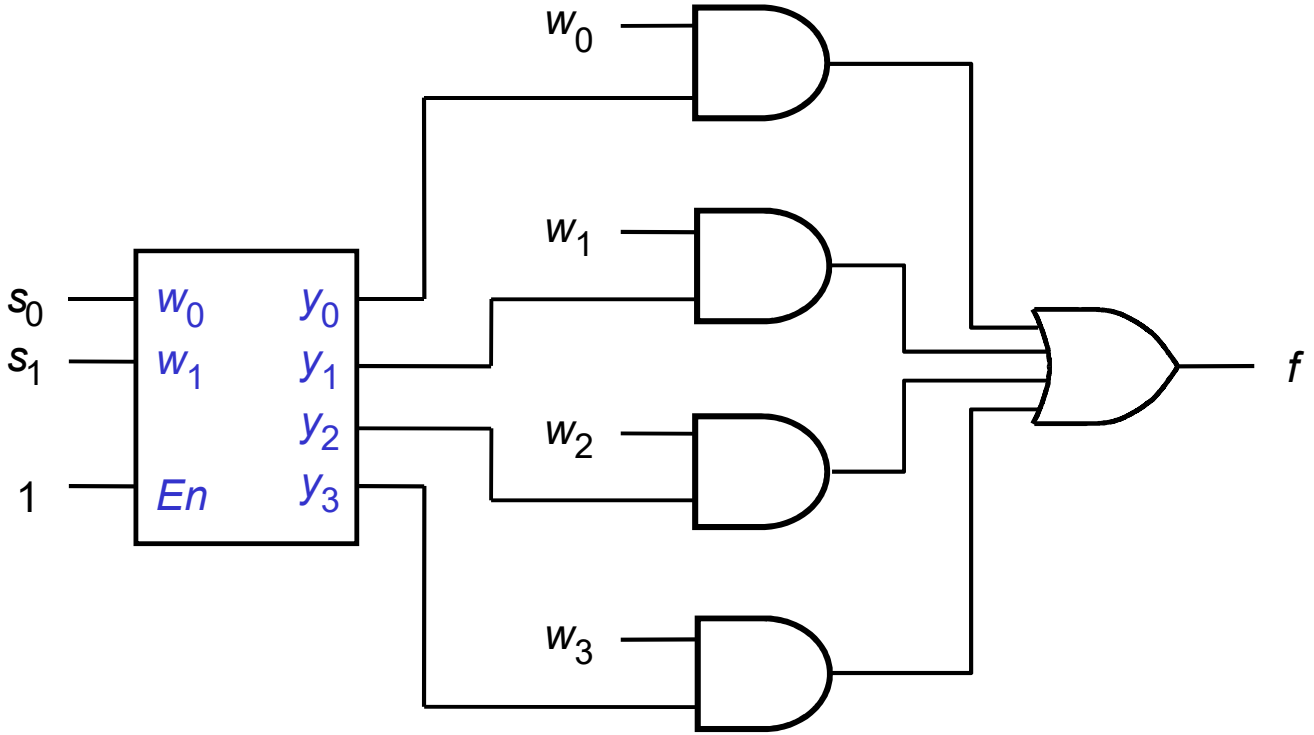
1 1 decod4 - entradas w1 w0

Decodificador 4-to-16 usando decodificadores 2-para-4

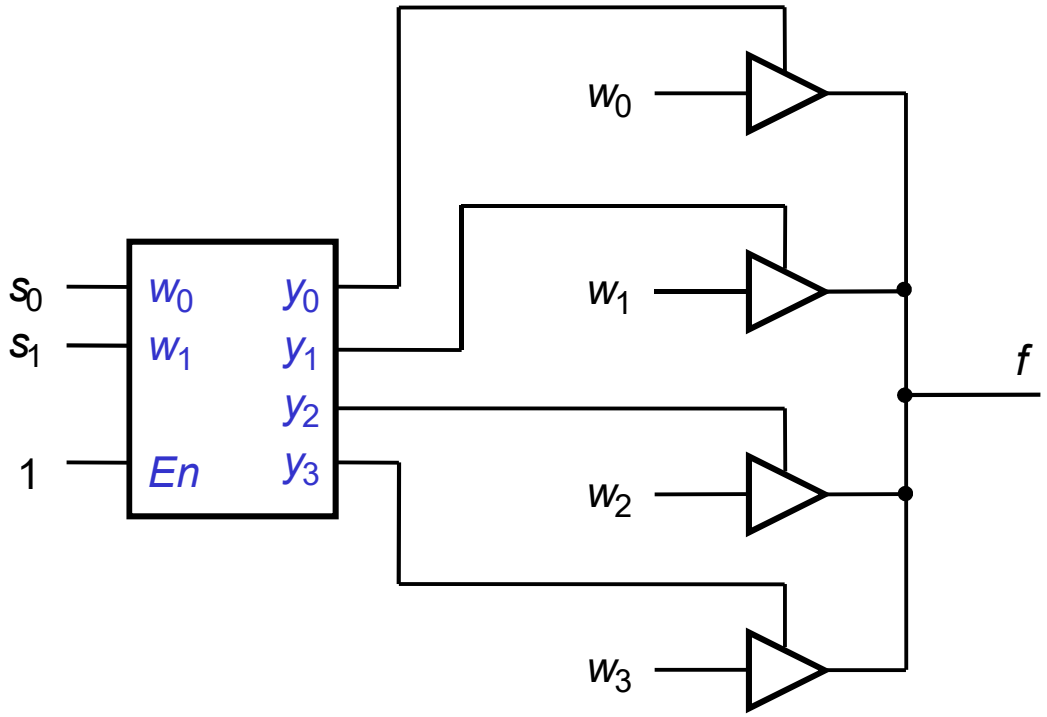


3 – MUX 4:1 com decodificador (2:4)

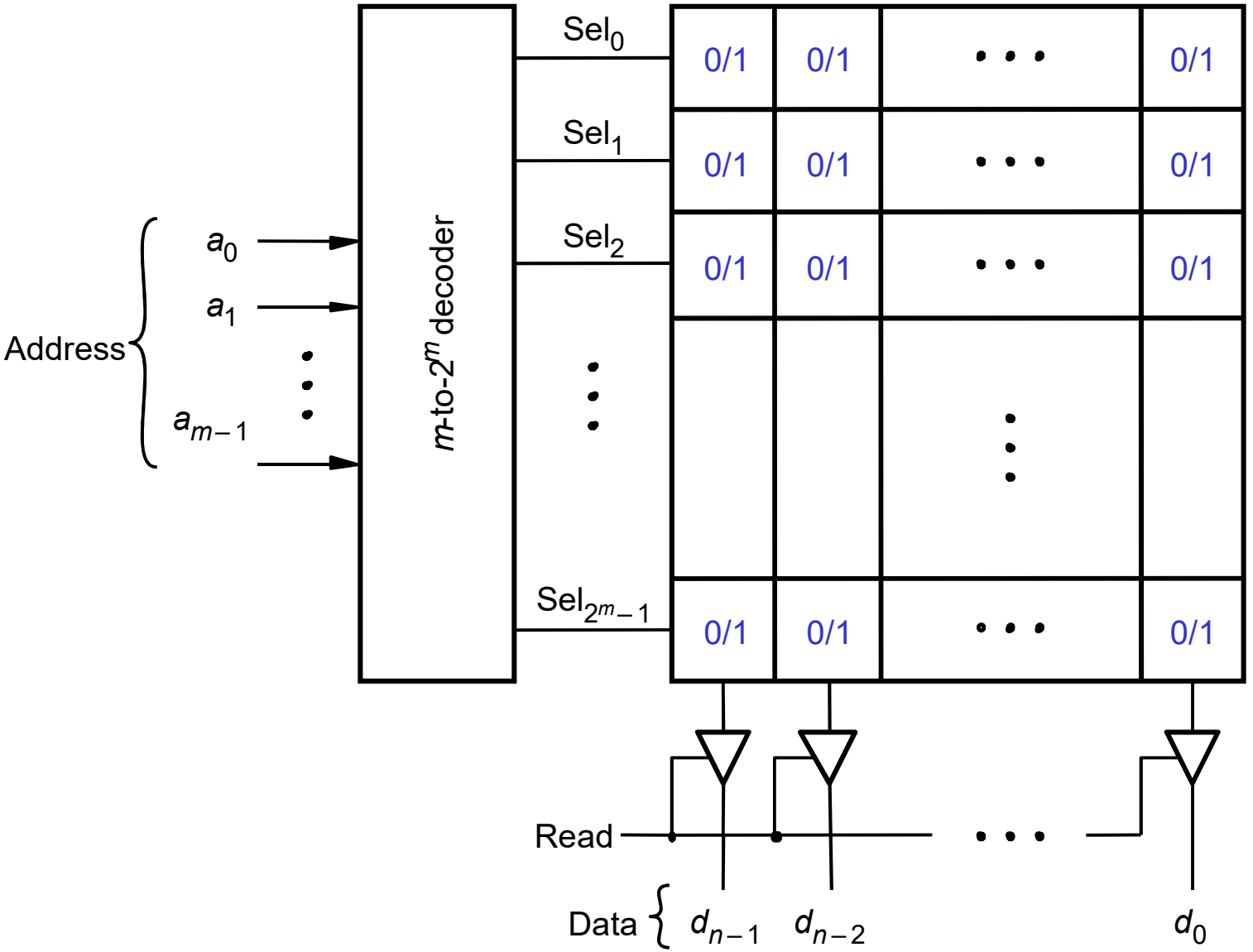
Multipexador 4-para-1 usando um decodificador



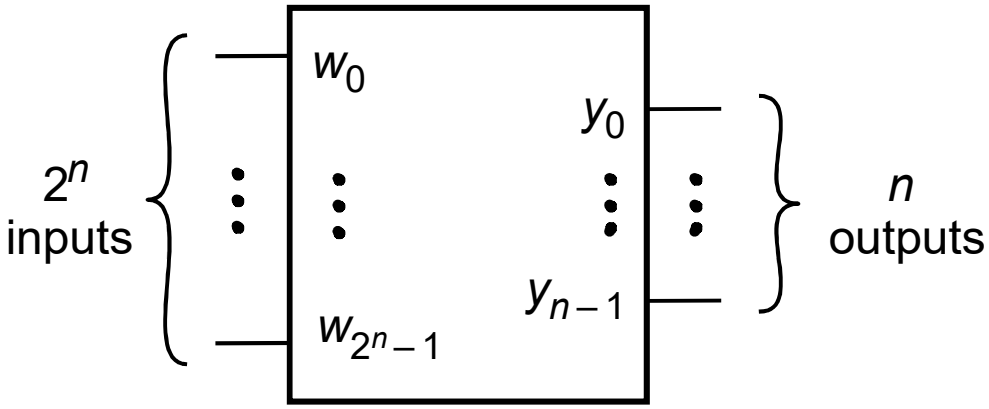
Multiplexador 4-para-1 usando um decodificador e buffers tri-state



Bloco de memória ***read only*** (ROM) 2^m x n



Codificador binário 2^n-para-n



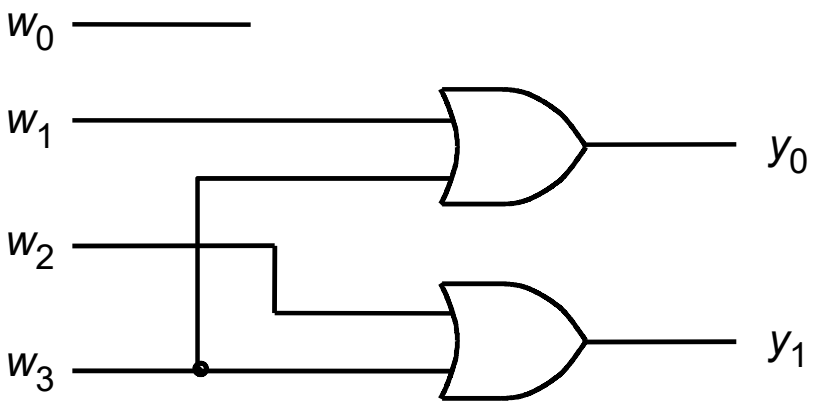
Que gere o par y1y0, que identifique qual de 4 entradas (w0,w1,w2,w3) está ativa, ou seja,
00 se w0 = 1 e restante = 0, 01 se w1 = 1 e restante = 0,
10 se w2 = 1 e restante = 0,
11 se w3 = 1 e restante = 0,

Codificador binário 4-para-2

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

(a) Tabela Verdade

w3	w2	w1	w0	y1	y0
0	0	0	0	d	d
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	d	d
0	1	0	0	1	0
0	1	0	1	d	d
0	1	1	0	d	d
0	1	1	1	d	d
1	0	0	0	1	1
1	0	0	1	d	d
1	0	1	0	d	d
1	0	1	1	d	d
1	1	0	0	d	d
1	1	0	1	d	d
1	1	1	0	d	d
1	1	1	1	d	d



(b) Circuito

Tabela Verdade para um codificador de prioridade 4-para-2

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Tabela Verdade para um codificador de prioridade 4-para-2

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

$$z = w3 + w2 + w1 + w0$$

w3	w2	w1	w0	y1	y0
0	0	0	0	d	d
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

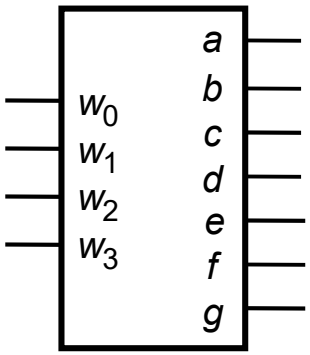
	00	01	11	10
00	d	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

$$y1 = w2 + w3$$

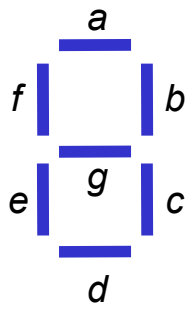
00	00	01	11	10
00	d	0	1	1
01	0	0	1	1
11	1	0	1	1
10	1	0	1	1

$$y0 = w3 + w2'w1$$

Conversor BCD para display de 7 segmentos



(a) Code converter



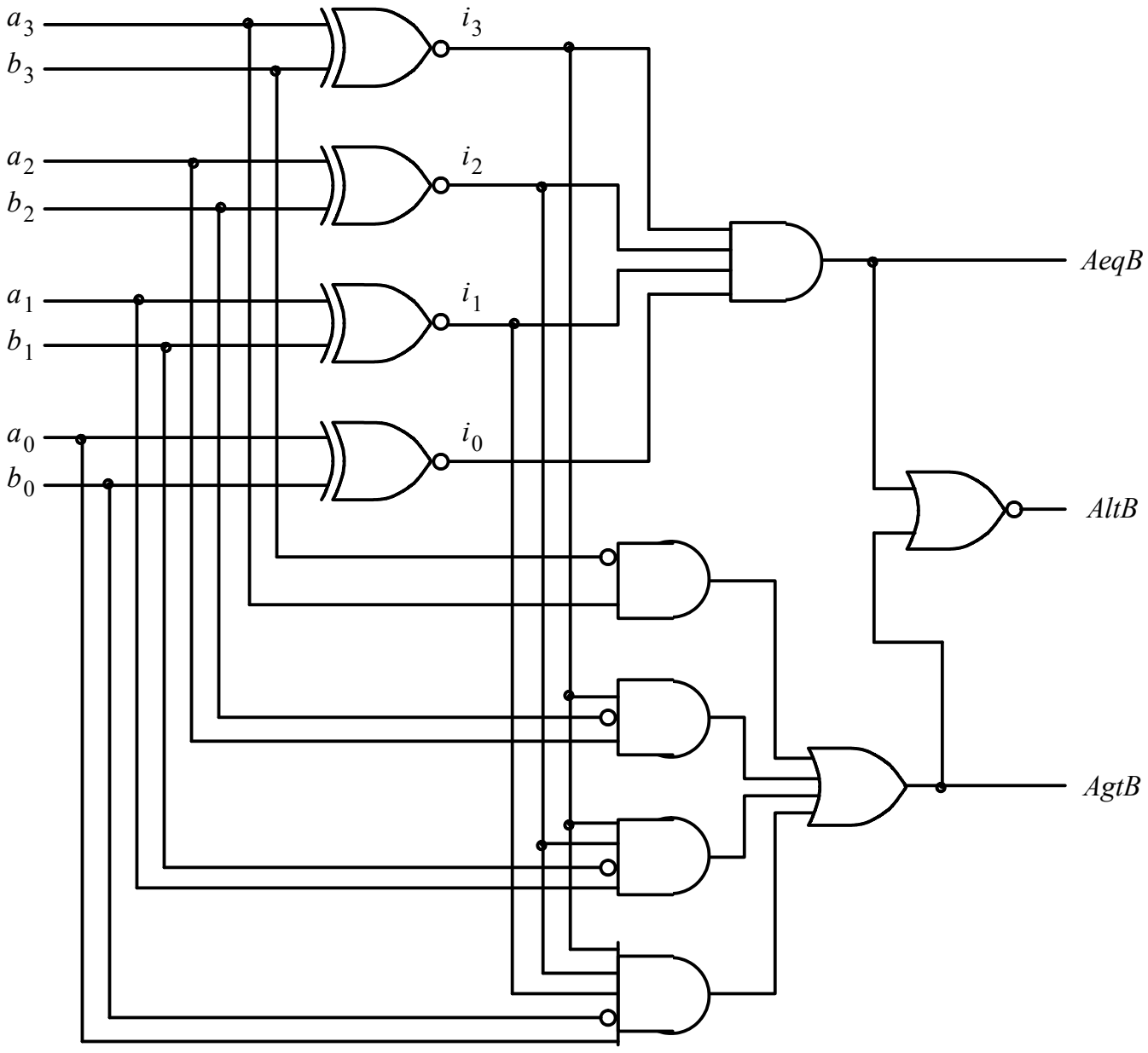
(b) 7-segment display

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

(c) Truth table

Exercício – projetar um comparador de 2 números de 4 bits (AeqB, AgtB, AltB)

Circuito Comparador de quatro bits



Código VHDL para um multiplexador 2-para-1

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT (  w0, w1, s     : IN     STD_LOGIC ;
            f             : OUT    STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    WITH s SELECT
        f <= w0 WHEN '0',
             w1 WHEN OTHERS ;
END Behavior ;
```

Código VHDL para um multiplexador 4-para-1

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux4to1 IS
    PORT (  w0, w1, w2, w3      : IN    STD_LOGIC ;
            s                   : IN    STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            f                   : OUT   STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE Behavior OF mux4to1 IS
BEGIN
    WITH s SELECT
        f <= w0 WHEN "00",
             w1 WHEN "01",
             w2 WHEN "10",
             w3 WHEN OTHERS ;
END Behavior ;
```

Declaração de componente para multiplexador 4-para-1

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
PACKAGE mux4to1_package IS
    COMPONENT mux4to1
        PORT (  w0, w1, w2, w3    : IN     STD_LOGIC ;
                s                 : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                f                 : OUT    STD_LOGIC ) ;
    END COMPONENT ;
END mux4to1_package ;
```

Código hierárquico para multiplexador 16-para-1

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
LIBRARY work ;
USE work.mux4to1_package.all ;

ENTITY mux16to1 IS
    PORT (  w   : IN    STD_LOGIC_VECTOR(0 TO 15) ;
            s   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            f   : OUT   STD_LOGIC ) ;
END mux16to1 ;

ARCHITECTURE Structure OF mux16to1 IS
    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
BEGIN
    Mux1: mux4to1 PORT MAP ( w(0), w(1), w(2), w(3), s(1 DOWNTO 0), m(0) ) ;
    Mux2: mux4to1 PORT MAP ( w(4), w(5), w(6), w(7), s(1 DOWNTO 0), m(1) ) ;
    Mux3: mux4to1 PORT MAP ( w(8), w(9), w(10), w(11), s(1 DOWNTO 0), m(2) ) ;
    Mux4: mux4to1 PORT MAP ( w(12), w(13), w(14), w(15), s(1 DOWNTO 0), m(3) ) ;
    Mux5: mux4to1 PORT MAP
            ( m(0), m(1), m(2), m(3), s(3 DOWNTO 2), f ) ;
END Structure ;
```

Código VHDL para um decodificador binário 2-para-4

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY dec2to4 IS
    PORT (  w   : IN    STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            En  : IN    STD_LOGIC ;
            y   : OUT   STD_LOGIC_VECTOR(0 TO 3) ) ;
END dec2to4 ;

ARCHITECTURE Behavior OF dec2to4 IS
    SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0) ;
BEGIN
    Enw <= En & w ;
    WITH Enw SELECT
        y <= "1000" WHEN "100",
             "0100" WHEN "101",
             "0010" WHEN "110",
             "0001" WHEN "111",
             "0000" WHEN OTHERS ;
END Behavior ;
```

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT (  w0, w1, s       : IN    STD_LOGIC ;
            f               : OUT   STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    f <= w0 WHEN s = '0' ELSE w1 ;
END Behavior ;
```

Figure 6.31 A 2-to-1 multiplexer using a conditional signal assignment

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY priority IS
    PORT (  w   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            y   : OUT   STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            z   : OUT   STD_LOGIC ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
    y <=    "11" WHEN w(3) = '1' ELSE
            "10" WHEN w(2) = '1' ELSE
            "01" WHEN w(1) = '1' ELSE
            "00" ;
    z <= '0' WHEN w = "0000" ELSE '1' ;
END Behavior ;
```

Figure 6.32 VHDL code for a priority encoder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY priority IS
    PORT (  w : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            y : OUT     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            z : OUT     STD_LOGIC ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
    WITH w SELECT
        y <=    "00" WHEN "0001",
                "01" WHEN "0010",
                "01" WHEN "0011",
                "10" WHEN "0100",
                "10" WHEN "0101",
                "10" WHEN "0110",
                "10" WHEN "0111",
                "11" WHEN OTHERS ;
    WITH w SELECT
        z <=    '0' WHEN "0000",
                '1' WHEN OTHERS ;
END Behavior ;
```

Figure 6.33 Less efficient code for a priority encoder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY compare IS
    PORT (  A, B                : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            AeqB, AgtB, AltB    : OUT   STD_LOGIC ) ;
END compare ;

ARCHITECTURE Behavior OF compare IS
BEGIN
    AeqB <= '1' WHEN A = B ELSE '0' ;
    AgtB <= '1' WHEN A > B ELSE '0' ;
    AltB <= '1' WHEN A < B ELSE '0' ;
END Behavior ;
```

Figure 6.34 VHDL code for a four-bit comparator

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.all ;

ENTITY compare IS
      PORT (   A, B                  : IN      SIGNED(3 DOWNTO 0) ;
               AeqB, AgtB, AltB      : OUT     STD_LOGIC ) ;
END compare ;

ARCHITECTURE Behavior OF compare IS
BEGIN
      AeqB <= '1' WHEN A = B ELSE '0' ;
      AgtB <= '1' WHEN A > B ELSE '0' ;
      AltB <= '1' WHEN A < B ELSE '0' ;
END Behavior ;
```

Figure 6.35 A four-bit comparator using signed numbers

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.mux4to1_package.all ;

ENTITY mux16to1 IS
        PORT (  w     : IN      STD_LOGIC_VECTOR(0 TO 15) ;
                s     : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
                f     : OUT     STD_LOGIC ) ;
END mux16to1 ;

ARCHITECTURE Structure OF mux16to1 IS
        SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
BEGIN
        G1: FOR i IN 0 TO 3 GENERATE
                Muxes: mux4to1 PORT MAP (
                        w(4*i), w(4*i+1), w(4*i+2), w(4*i+3), s(1 DOWNTO 0), m(i) ) ;
        END GENERATE ;
        Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNTO 2), f ) ;
END Structure ;
```

Figure 6.36 Code for a 16-to-1 multiplexer using a generate statement

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY dec4to16 IS
    PORT (   w    : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
             En   : IN     STD_LOGIC ;
             y    : OUT    STD_LOGIC_VECTOR(0 TO 15) ) ;
END dec4to16 ;

ARCHITECTURE Structure OF dec4to16 IS
    COMPONENT dec2to4
        PORT (   w    : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                 En   : IN     STD_LOGIC ;
                 y    : OUT    STD_LOGIC_VECTOR(0 TO 3) ) ;
    END COMPONENT ;
    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
BEGIN
    G1: FOR i IN 0 TO 3 GENERATE
        Dec_ri: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(i), y(4*i TO 4*i+3) );
        G2: IF i=3 GENERATE
            Dec_left: dec2to4 PORT MAP ( w(i DOWNTO i-1), En, m ) ;
        END GENERATE ;
    END GENERATE ;
END Structure ;
```

Figure 6.37 Hierarchical code for a 4-to-16 binary decoder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT (   w0, w1, s      : IN     STD_LOGIC ;
             f              : OUT    STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    PROCESS ( w0, w1, s )
    BEGIN
        IF s = '0' THEN
            f <= w0 ;
        ELSE
            f <= w1 ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figure 6.38 A 2-to-1 multiplexer specified using an if-then-else statement

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT (   w0, w1, s      : IN     STD_LOGIC ;
             f              : OUT    STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    PROCESS ( w0, w1, s )
    BEGIN
        f <= w0 ;
        IF s = '1' THEN
            f <= w1 ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figure 6.39 Alternative code for a 2-to-1 multiplexer

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY priority IS
    PORT (  w   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            y   : OUT   STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            z   : OUT   STD_LOGIC ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
    PROCESS ( w )
    BEGIN
        IF w(3) = '1' THEN
            y <= "11" ;
        ELSIF w(2) = '1' THEN
            y <= "10" ;
        ELSIF w(1) = '1' THEN
            y <= "01" ;
        ELSE
            y <= "00" ;
        END IF ;
    END PROCESS ;
    z <= '0' WHEN w = "0000" ELSE '1' ;
END Behavior ;
```

Figure 6.40 A priority encoder specified using if-then-else

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY priority IS
    PORT (  w   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            y   : OUT   STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            z   : OUT   STD_LOGIC ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
    PROCESS ( w )
    BEGIN
        y <= "00" ;
        IF w(1) = '1' THEN y <= "01" ; END IF ;
        IF w(2) = '1' THEN y <= "10" ; END IF ;
        IF w(3) = '1' THEN y <= "11" ; END IF ;

        z <= '1' ;
        IF w = "0000" THEN z <= '0' ; END IF ;
    END PROCESS ;
END Behavior ;
```

Figure 6.41 Alternative code for the priority encoder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY compare1 IS
    PORT (  A, B      : IN      STD_LOGIC ;
            AeqB      : OUT     STD_LOGIC ) ;
END compare1 ;

ARCHITECTURE Behavior OF compare1 IS
BEGIN
    PROCESS ( A, B )
    BEGIN
        AeqB <= '0' ;
        IF A = B THEN
            AeqB <= '1' ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figure 6.42 Code for a one-bit equality comparator

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY implied IS
        PORT (   A, B       : IN        STD_LOGIC ;
                 AeqB       : OUT       STD_LOGIC ) ;
END implied ;

ARCHITECTURE Behavior OF implied IS
BEGIN
        PROCESS ( A, B )
        BEGIN
            IF A = B THEN
                AeqB <= '1' ;
            END IF ;
        END PROCESS ;
END Behavior ;
```

Figure 6.43 An example of code that results in implied memory

```
...
PROCESS ( A, B )
BEGIN
      IF A = B THEN
            AeqB <= '1' ;
      END IF ;
END PROCESS ;
 ...
```

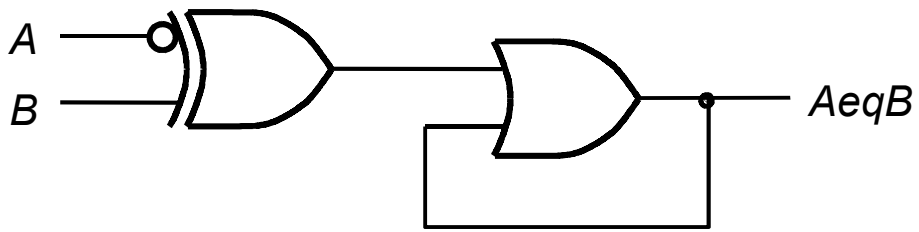


Figure 6.44 Circuit generated due to implied memory

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT (  w0, w1, s   : IN    STD_LOGIC ;
            f           : OUT   STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    PROCESS ( w0, w1, s )
    BEGIN
        CASE s IS
            WHEN '0' =>
                f <= w0 ;
            WHEN OTHERS =>
                f <= w1 ;
        END CASE ;
    END PROCESS ;
END Behavior ;
```

Figure 6.45 A CASE statement that represents a 2-to-1 multiplexer

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY dec2to4 IS
    PORT (  w    : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            En   : IN     STD_LOGIC ;
            y    : OUT    STD_LOGIC_VECTOR(0 TO 3) ) ;
END dec2to4 ;

ARCHITECTURE Behavior OF dec2to4 IS
BEGIN
    PROCESS ( w, En )
    BEGIN
        IF En = '1' THEN
            CASE w IS
                WHEN "00" =>        y <= "1000" ;
                WHEN "01" =>        y <= "0100" ;
                WHEN "10" =>        y <= "0010" ;
                WHEN OTHERS =>  y <= "0001" ;
            END CASE ;
        ELSE
            y <= "0000" ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figure 6.46 A 2-to-4 binary decoder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY seg7 IS
        PORT (      bcd     : IN        STD_LOGIC_VECTOR(3 DOWNTO 0) ;
                    leds    : OUT       STD_LOGIC_VECTOR(1 TO 7) ) ;
END seg7 ;
ARCHITECTURE Behavior OF seg7 IS
BEGIN
        PROCESS ( bcd )
        BEGIN
                CASE bcd IS             --              abcdefg
                        WHEN "0000"     => leds <=    "1111110" ;
                        WHEN "0001"     => leds <=    "0110000" ;
                        WHEN "0010"     => leds <=    "1101101" ;
                        WHEN "0011"     => leds <=    "1111001" ;
                        WHEN "0100"     => leds <=    "0110011" ;
                        WHEN "0101"     => leds <=    "1011011" ;
                        WHEN "0110"     => leds <=    "1011111" ;
                        WHEN "0111"     => leds <=    "1110000" ;
                        WHEN "1000"     => leds <=     "1111111" ;
                        WHEN "1001"     => leds <=     "1110011" ;
                        WHEN OTHERS     => leds <=     "-------" ;
                END CASE ;
        END PROCESS ;
END Behavior ;
```

Figure 6.47 A BCD-to-7-segment decoder

Operation	Inputs s_2 s_1 s_0	Outputs F
Clear	0 0 0	0 0 0 0
B−A	0 0 1	$B - A$
A−B	0 1 0	$A - B$
ADD	0 1 1	$A + B$
XOR	1 0 0	A XOR B
OR	1 0 1	A OR B
AND	1 1 0	A AND B
Preset	1 1 1	1 1 1 1

Table 6.1 The functionality of the 74381 ALU

Please see “portrait orientation” PowerPoint file for Chapter 6

Figure 6.48 Code that represents the functionality of the 74381 ALU

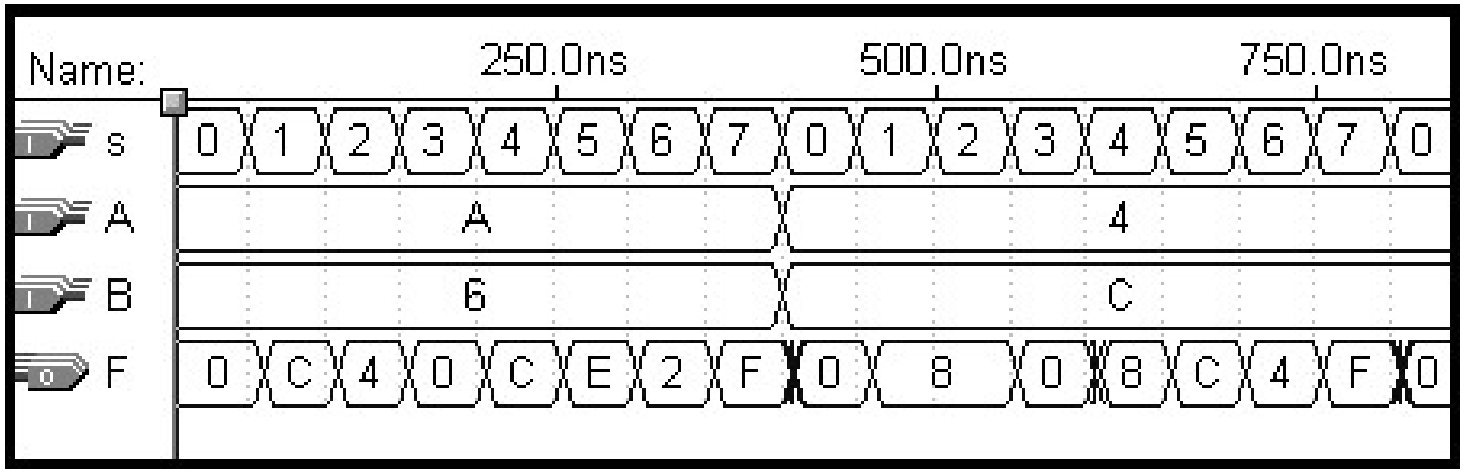


Figure 6.49 Timing simulation for the 74381 ALU code