IC-UNICAMP

MC 613

IC/Unicamp

Prof Guido Araújo
Prof Mario Côrtes

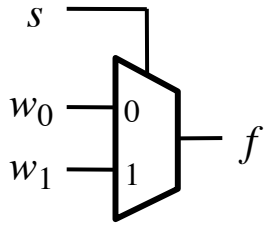
Circuitos Combinacionais Típicos

IC-UNICAMP

Tópicos

- Multiplexadores
- Decodificadores
- Decodificadores de prioridade
- Conversores de código
- Conversão bin-> 7 segmentos

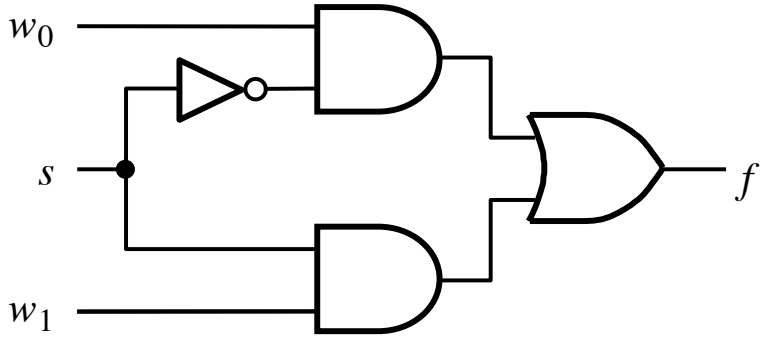
Mux 2:1



(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table



(c) Sum-of-products circuit

IC-UNICAMP

MUX 2:1 com atribuição selecionada de sinal– VHDL

```
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 ;
```

IC-UNICAMP

MUX 2:1 com atribuição condicional – VHDL

```
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 ;
```

IC-UNICAMP

VHDL: Selected Signal Assignment (Atribuição selecionada de sinal)

- A atribuição a um sinal pode ter vários valores em função de um sinal de "seleção"
  - IMPORTANTE: **todas as combinações(*)** de valores do sinal de seleção têm que ser explicitamente listadas (como um MUX)
  - Variante: uso do OTHERS
  - Exemplo: sinal de seleção = ctl de 2 bits

```
WITH ctl SELECT
   f <=  w0 WHEN "00",
         w1 WHEN "01",
         w1 WHEN "10",
         w1 WHEN "11";
```

```
WITH ctl SELECT
  f <=w0 WHEN "00",
       w1 WHEN OTHERS ;
```

(*) Atenção com o tipo do sinal

IC-UNICAMP

VHDL: Atribuição condicional

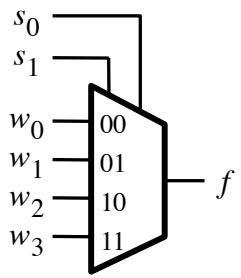
- Ao contrário do que parece, não é equivalente a “Selected Signal Assignment”
  - As condições listadas após o WHEN não precisam ser mutuamente exclusivas (elas têm prioridade da esquerda para a direita)
- Exemplo com uma condição

```
f <= w0 WHEN ctl = "00" ELSE w1;
```

- Exemplo com 3 condições

```
f <=     w0 WHEN ctl = "00" ELSE

         w1 WHEN ctl = "01" ELSE w3;
```

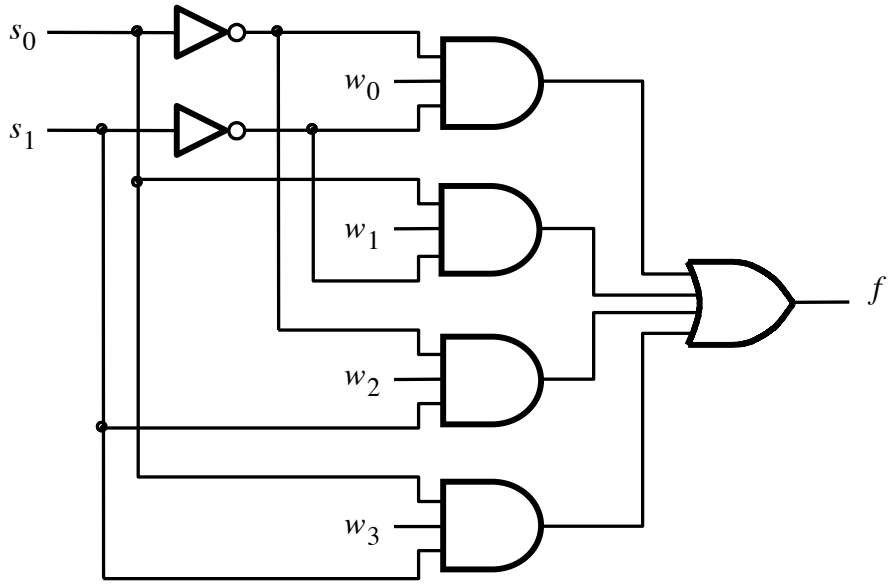
Mux 4:1



(a) Graphic symbol

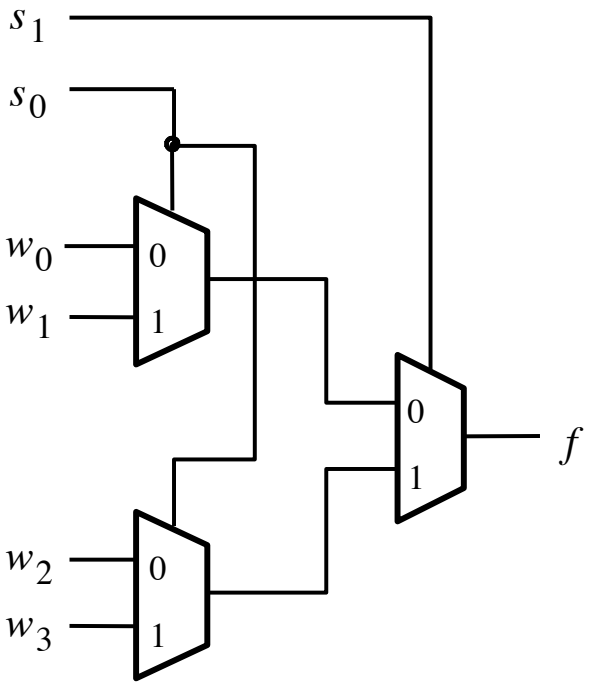
s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table



(c) Circuit

Mux 4:1 construído com Mux 2:1

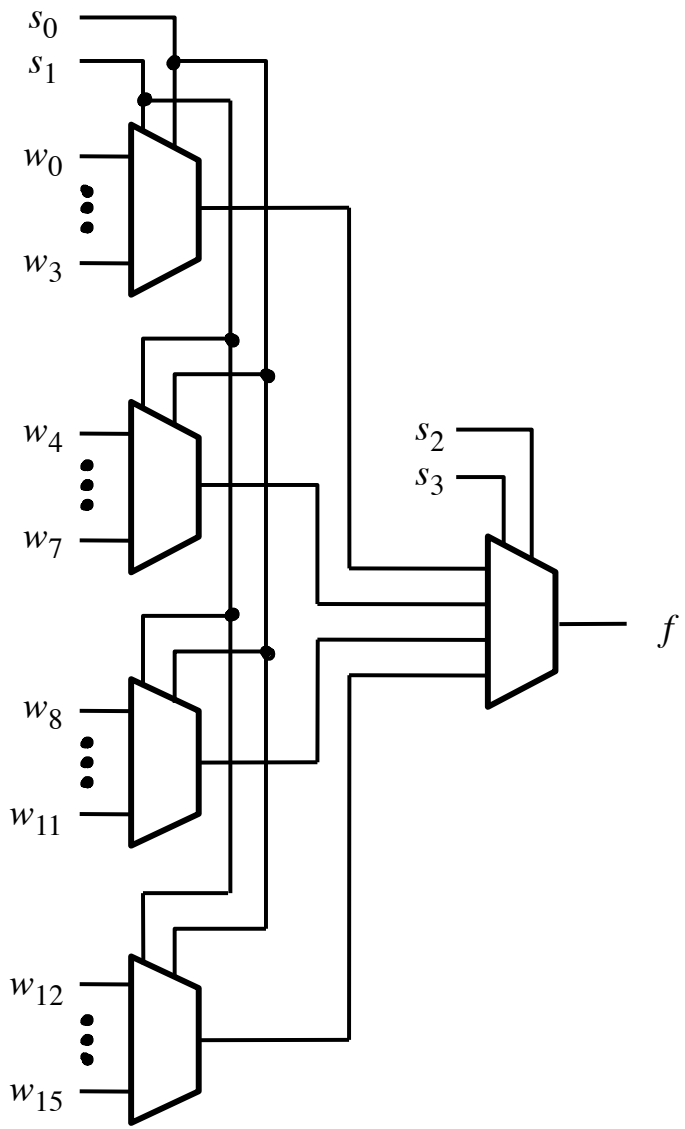


IC-UNICAMP

MUX 4:1 – VHDL

```
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 ;
```

Mux 16:1



IC-UNICAMP

MUX 4:1 – Declaração de Component

```
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 ;
```

Neste exemplo:

•Declaração de um componente dentro de um “package” a ser referenciado posteriormente

IC-UNICAMP

MUX 16:1 hierárquico – VHDL (1)

Usa o pacote definido

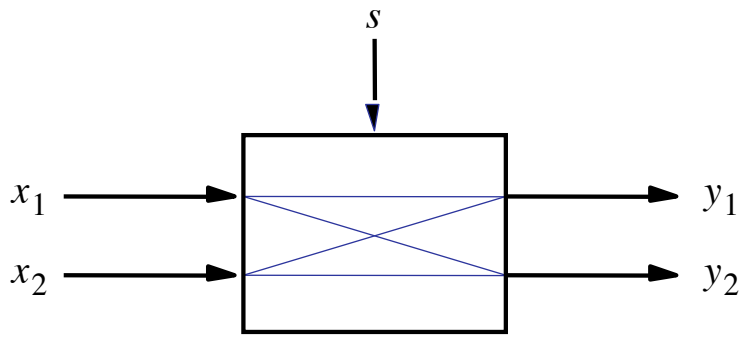
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
LIBRARY work ;
USE work.mux4to1_package.all ;

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 ;
```

IC-UNICAMP

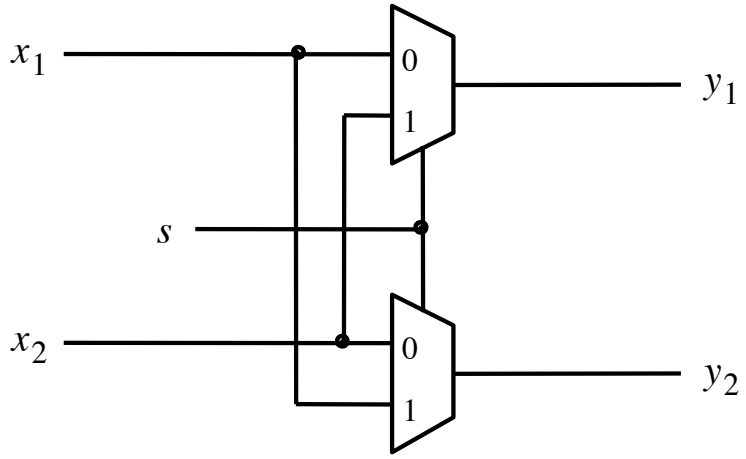
MUX 16:1 hierárquico – VHDL (2)

```
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 ;
```

Chave crossbar implementada c/ Mux



(a) A 2x2 crossbar switch



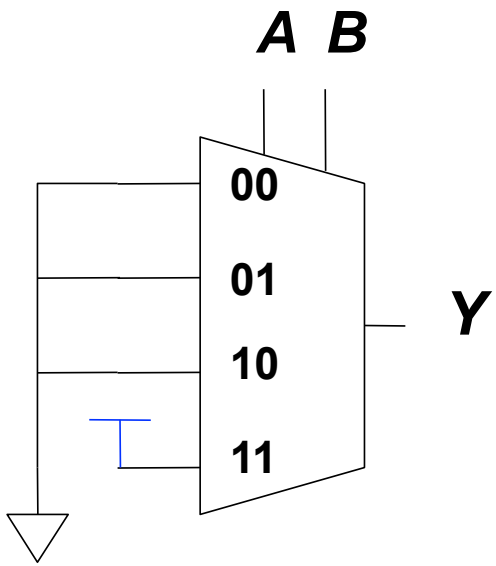
(b) Implementation using multiplexers

Lógica Usando Mux

- Usand o mux como uma lookup table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$Y = AB$



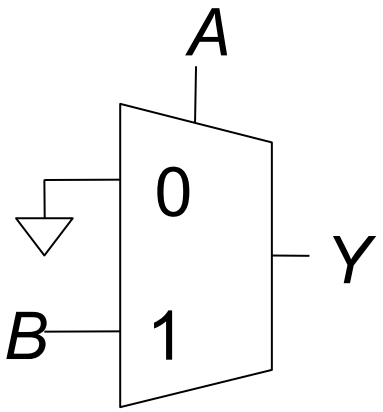
Implementando Funções com Mux

- Reduzindo o tamanho do Mux

$Y = AB$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

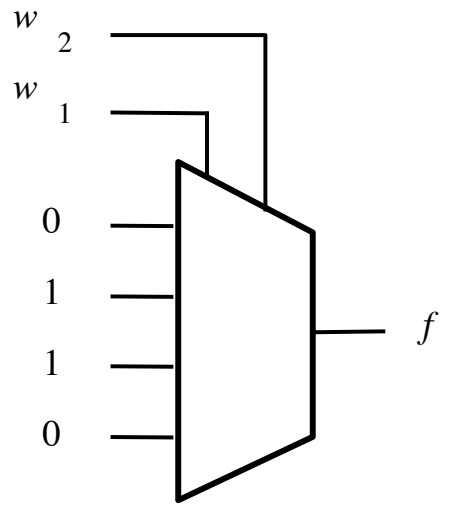
A	Y
0	0
1	B



Mux: implementação de funções lógicas

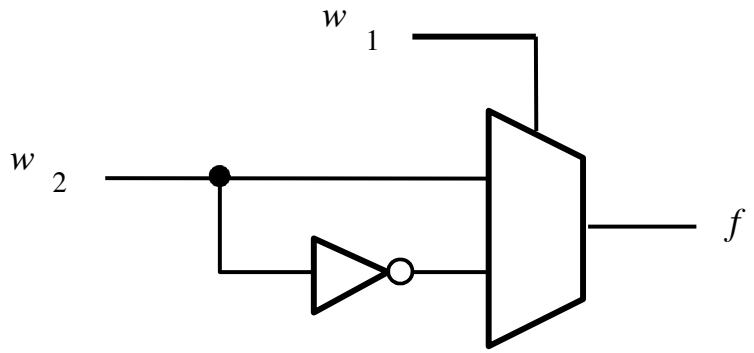
IC-UNICAMP

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



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

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

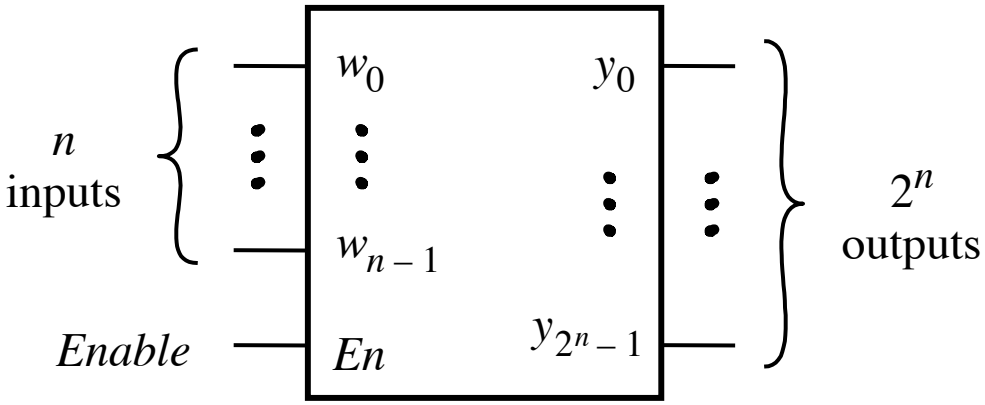


IC-UNICAMP

Implementação com VHDL (2ª alt)

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

ENTITY circuito IS
   PORT (w1, w2    : IN  STD_LOGIC ;
         f         : OUT       STD_LOGIC ) ;
END circuito ;

ARCHITECTURE Behavior OF circuito IS
BEGIN
   WITH w1 SELECT
      f <=  w2 WHEN '0',
            NOT w2 WHEN OTHERS ;
END Behavior ;
```

Decodificadores

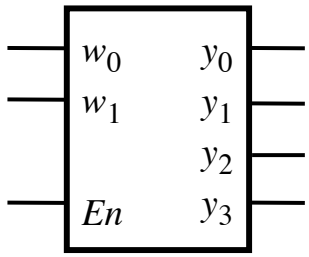


Decodificador n-to-2^n

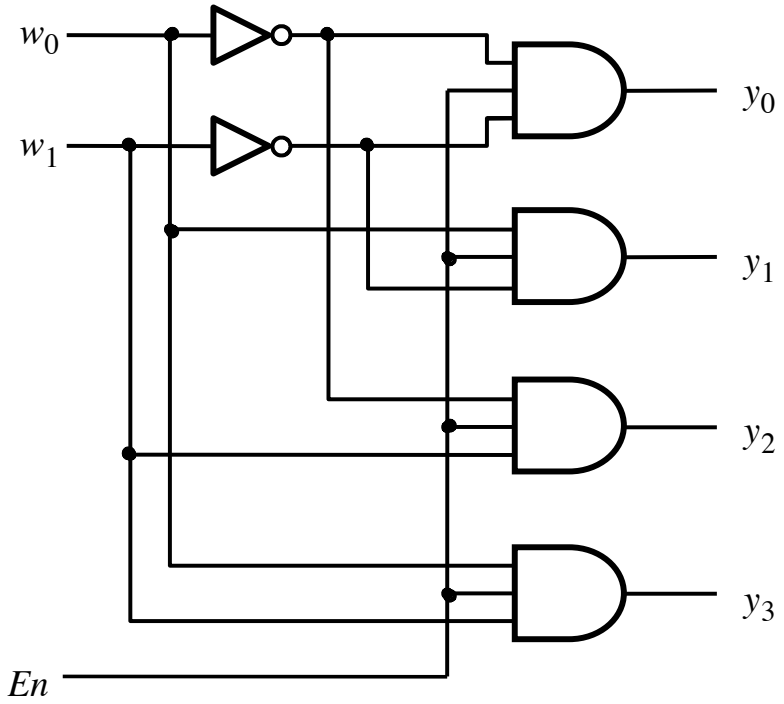
Decod. 2:4

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) Truth table



(b) Graphic symbol



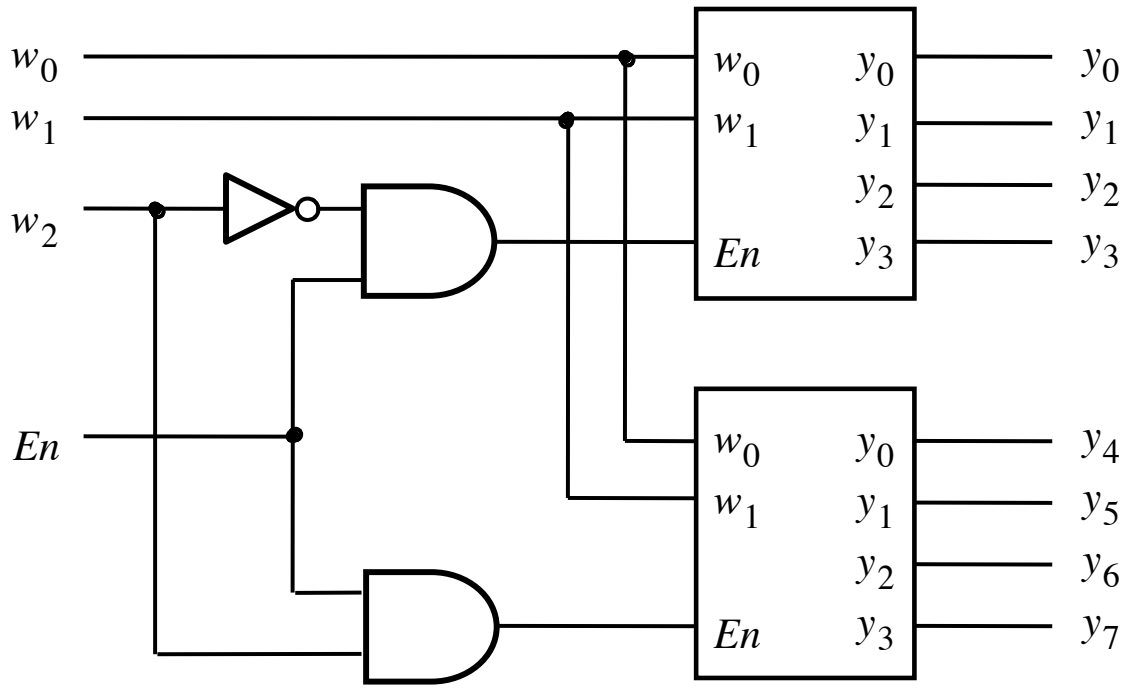
(c) Logic circuit

IC-UNICAMP

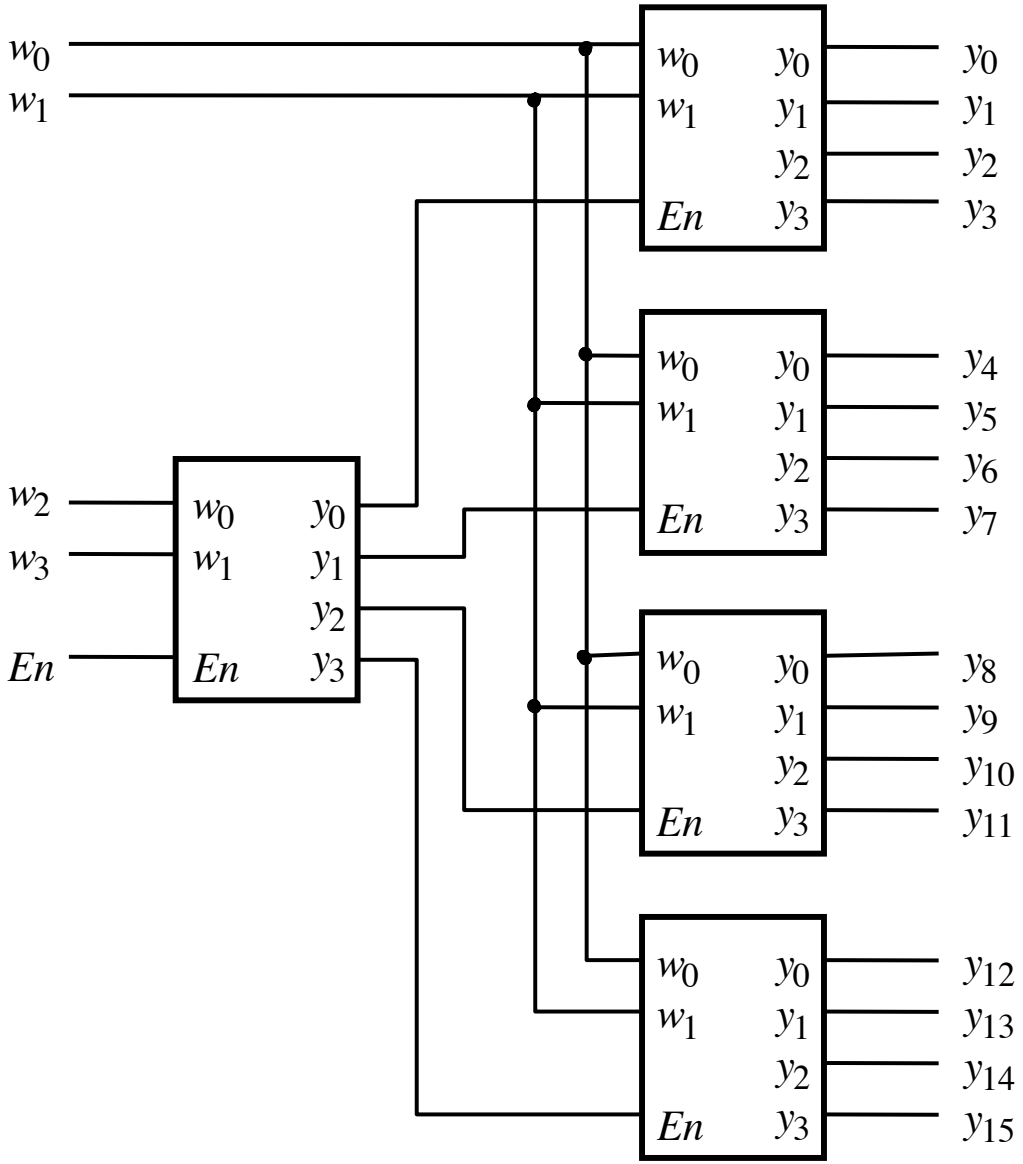
Decoder 2:4 – VHDL

```
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 ;
```

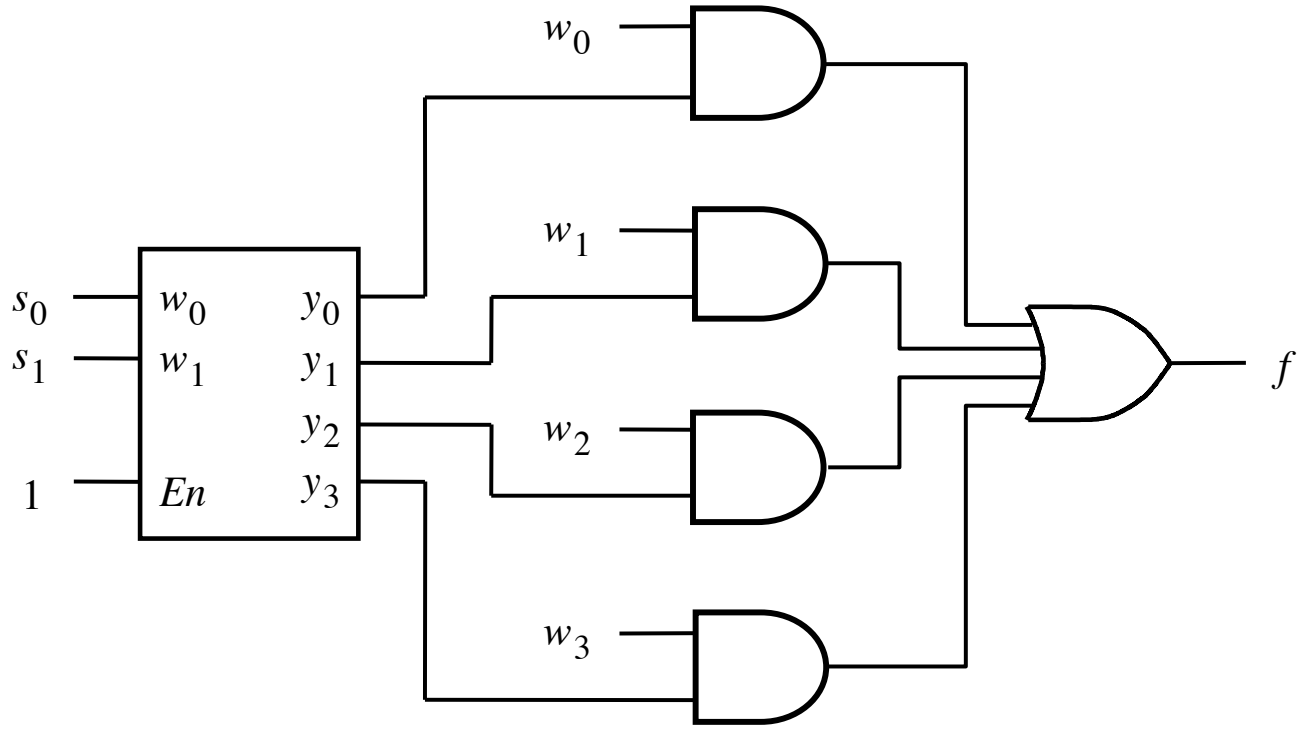
Um Decod. 3:8 usando decod. 2:4



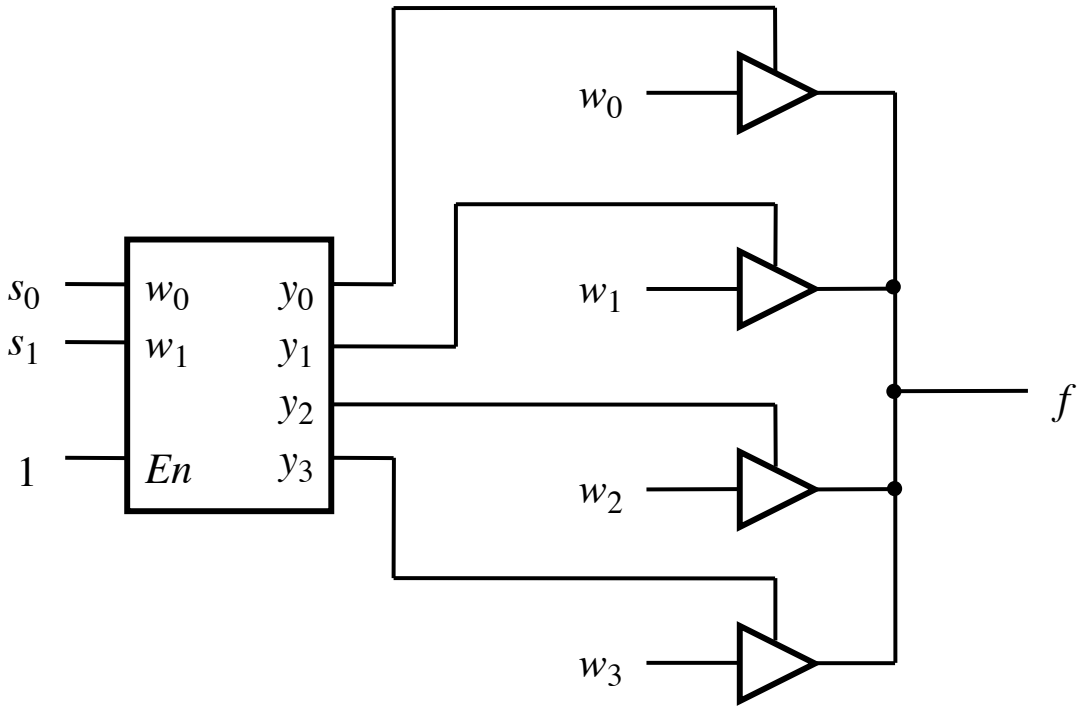
Decod. 4:16 usando árvore de decod.



Mux impl. a partir de decod



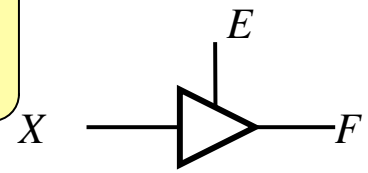
Mux c/ decod. e buffers tri-state



Buffers tri-state em VHDL

IC-UNICAMP

Permite configurar parâmetros (n bits)




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

ENTITY zbuffer IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT ( X   : IN   STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
           E   : IN   STD_LOGIC ;
           F   : OUT  STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END zbuffer ;

ARCHITECTURE Behavior OF zbuffer IS
BEGIN
    F <= (OTHERS => 'Z') WHEN E = '0' ELSE X ;
END Behavior ;
```

Construção (**OTHERS => '1'**)

- usada principalmente em vetores para atribuir um mesmo valor para todos os bits (aqui todos 8 bits de F ← 'Z')

IC-UNICAMP

Component Buffer

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

PACKAGE ZBuffer_package IS
  COMPONENT ZBuffer
     GENERIC (N : INTEGER) ;
     PORT (X, IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
          E: IN STD_LOGIC ;
          f: OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0)) ;
  END COMPONENT ;
END ZBuffer_package ;
```

Encapsulando o buffer tri-state em componente

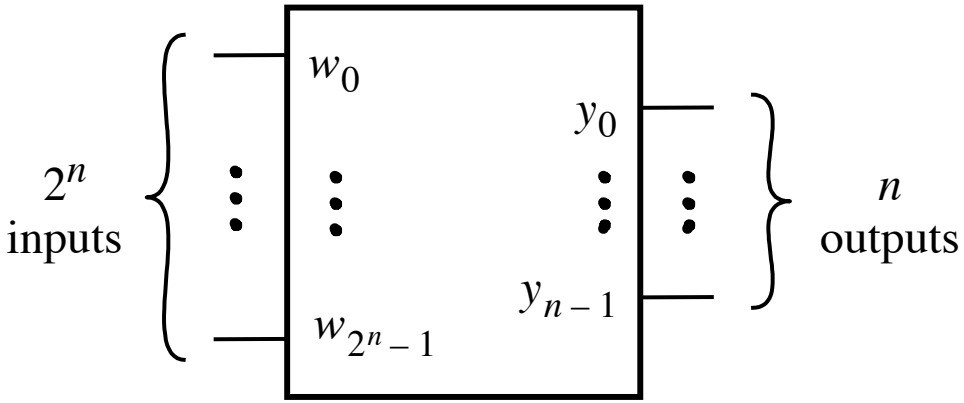
Precisa informar o tipo do parâmetro N

IC-UNICAMP

Gerando Adaptador 32-bits

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

ENTITY bus32adapter IS
   GENERIC ( N : INTEGER := 32 );
   PORT (X: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
         Z: IN STD_LOGIC ;
         B: OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END bus32adapter ;

ARCHITECTURE behavior OF bus32adapter IS
BEGIN
   buf:  ZBuffer
         GENERIC MAP( N =>32 )
         port map (X, Z, B) ;
END behavior ;
```

Codificadores

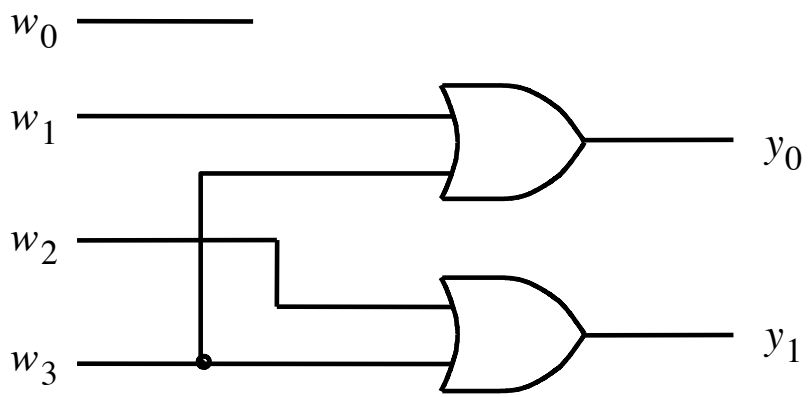


Codificador 2^n-to-n

Codificador 4:2

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

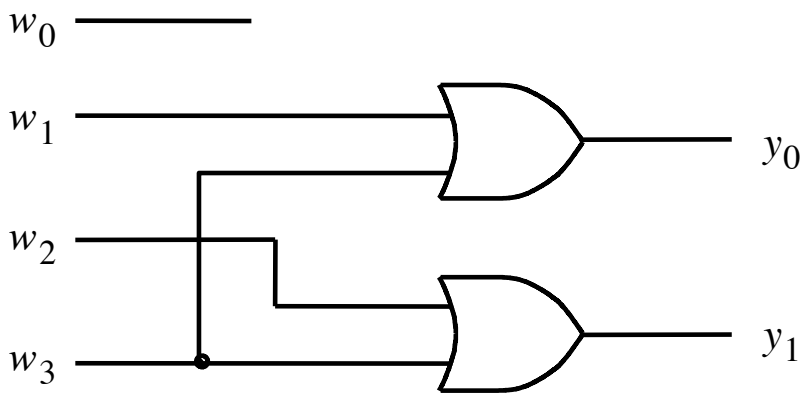
(a) Truth table

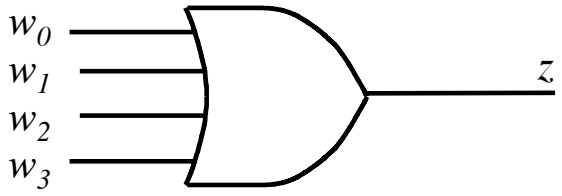


(b) Circuit

Outro codificador 4:2

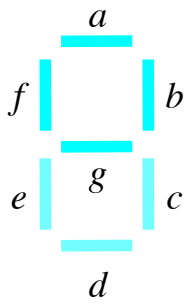
w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1



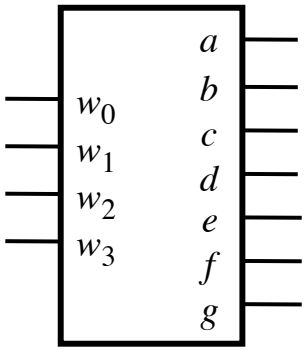


IC-UNICAMP

Conversor Bin →BCD (7 segmentos)



(b) 7-segment display



(a) Code converter

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

Implementação convencional

Segment a

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

Segment f

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

Segment b

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

Segment g

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

Segment e

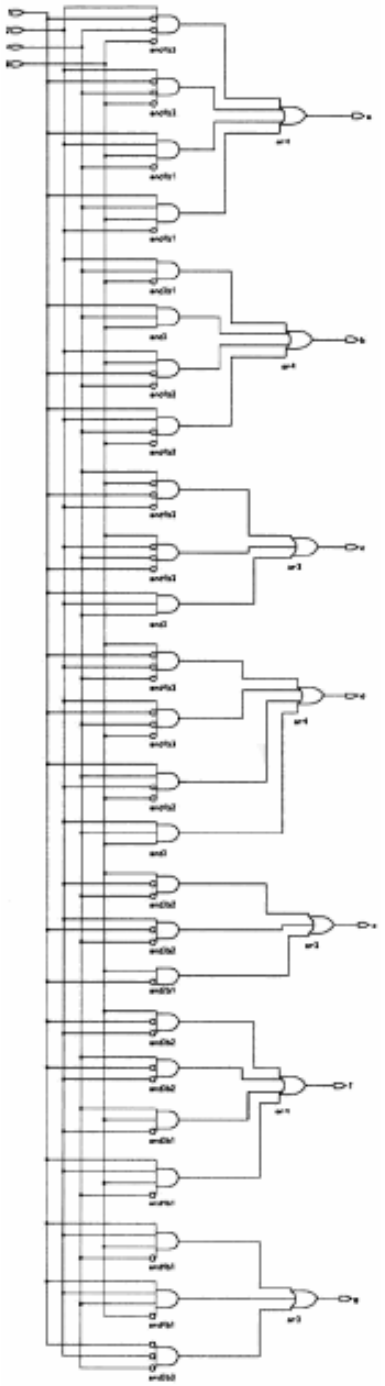
	00	01	11	10
00	0	1	0	0
01	1	1	0	1
11	1	1	0	0
10	0	0	0	0

Segment c

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

Segment d

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



IC-UNICAMP

Implementação VHDL

- Implementar de forma mais elegante
- Usando a construção do tipo

```
WITH num SELECT
   leds <=
       "1111110" WHEN "0000",
       "0110000" WHEN "0001" ,
       ……
```