

Circuitos ARITMETICOS

Fche 2011

Antes. Bcd/7seg conVHDL

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY bcd IS PORT(
    i0, i1, i2, i3: IN STD_LOGIC;
    a, b, c, d, e, f, g: OUT STD_LOGIC);
END bcd;
ARCHITECTURE Dataflow OF bcd IS
BEGIN
    a <= i3 OR i1 OR (i2 XNOR i0);           -- seg a
    b <= (NOT i2) OR NOT (i1 XOR i0);       -- seg b
    c <= i2 OR (NOT i1) OR i0;             -- seg c
    d <= (i1 AND NOT i0) OR (NOT i2 AND NOT i0)
        OR (NOT i2 AND i1) OR (i2 AND NOT i1 AND i0); -- seg d
    e <= (i1 AND NOT i0) OR (NOT i2 AND NOT i0); -- seg e
    f <= i3 OR (i2 AND NOT i1)
        OR (i2 AND NOT i0) OR (NOT i1 AND NOT i0); -- seg f
    g <= i3 OR (i2 XOR i1) OR (i1 AND NOT i0); -- seg g
END Dataflow;
```

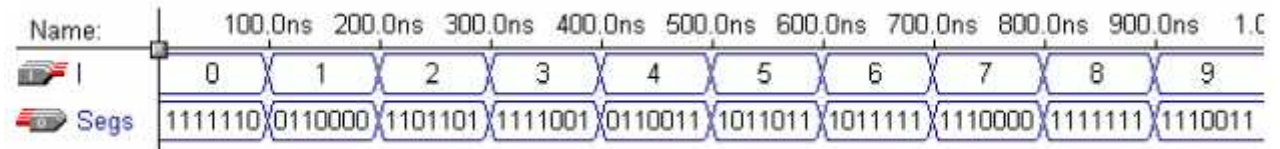
Dataflow VHDL code of the BCD to 7-segment decoder.

Utilizando case, procesos y vector inicial

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY bcd IS PORT (
  I: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
  Segs: OUT STD_LOGIC_VECTOR (1 TO 7));
END bcd;
```

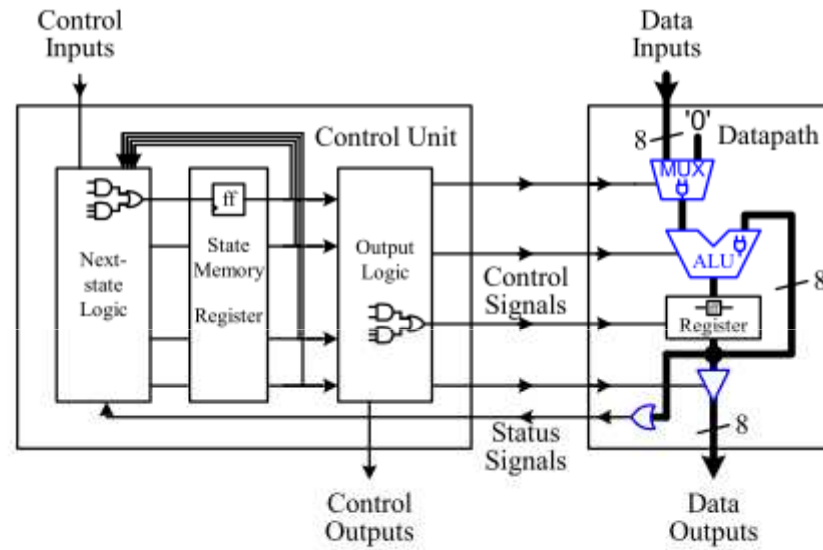
```
ARCHITECTURE Behavioral OF bcd IS
BEGIN
  PROCESS (I)
  BEGIN
    CASE I IS
      WHEN "0000" => Segs <= "1111110";
      WHEN "0001" => Segs <= "0110000";
      WHEN "0010" => Segs <= "1101101";
      WHEN "0011" => Segs <= "1111001";
      WHEN "0100" => Segs <= "0110011";
      WHEN "0101" => Segs <= "1011011";
      WHEN "0110" => Segs <= "1011111";
      WHEN "0111" => Segs <= "1110000";
      WHEN "1000" => Segs <= "1111111";
      WHEN "1001" => Segs <= "1110011";
      WHEN OTHERS => Segs <= "0000000";
    END CASE;
  END PROCESS;
END Behavioral;
```



A sample simulation trace of the behavioral 7-segment decoder code.

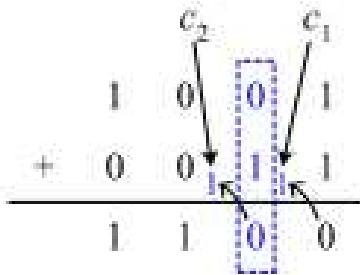
Behavioral VHDL code of the BCD to 7-segment decoder.

Ctos. Aritmeticos

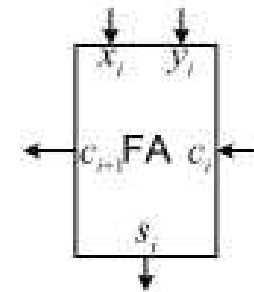


Sumador Completo

two 4-bit binary numbers, $X = 1001$ and $Y = 0011$.



x_i	y_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY fa IS PORT (
    Ci, Xi, Yi: IN STD_LOGIC;
    Ci1, Si: OUT STD_LOGIC);
END fa;

ARCHITECTURE Dataflow OF fa IS
BEGIN
    Ci1 <= (Xi AND Yi) OR (Ci AND (Xi XOR Yi));
    Si <= Xi XOR Yi XOR Ci;
END Dataflow;
    
```

Dataflow VHDL code for a 1-bit full adder.

Sumador Completo en Cascada

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Adder4 IS PORT (
  A, B: IN STD_LOGIC_VECTOR(3 DOWNT0 0);
  Cout: OUT STD_LOGIC;
  SUM: OUT STD_LOGIC_VECTOR(3 DOWNT0 0));
END Adder4;

ARCHITECTURE Structural OF Adder4 IS
  COMPONENT FA PORT (
    ci, xi, yi: IN STD_LOGIC;
    co, si: OUT STD_LOGIC);
  END COMPONENT;

  SIGNAL Carryv: STD_LOGIC_VECTOR(4 DOWNT0 0);

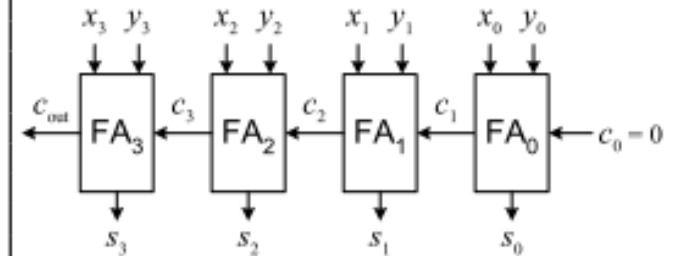
BEGIN
  Carryv(0) <= '0';

  Adder: FOR k IN 3 DOWNT0 0 GENERATE
    FullAdder: FA PORT MAP (Carryv(k), A(k), B(k), Carryv(k+1), SUM(k));
  END GENERATE Adder;

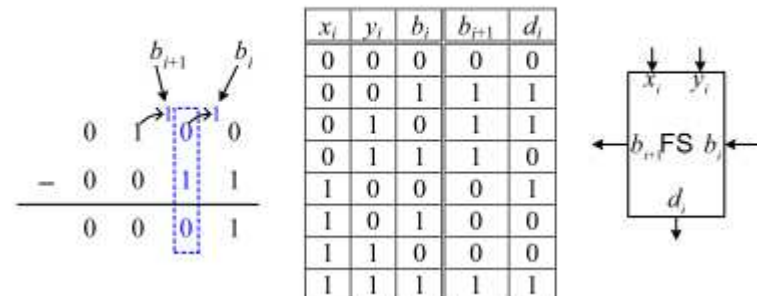
  Cout <= Carryv(4);
END Structural;

```

VHDL code for a 4-bit ripple-carry adder using a FOR-GENERATE statement.



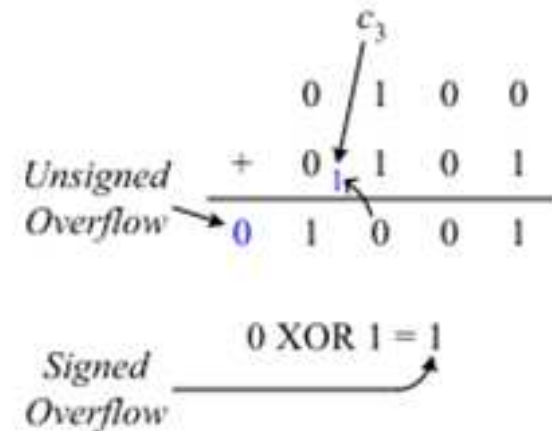
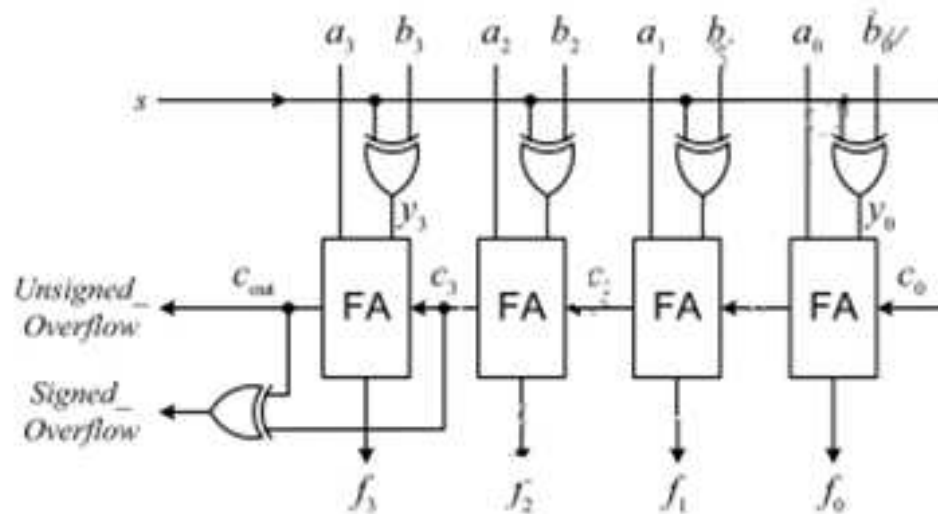
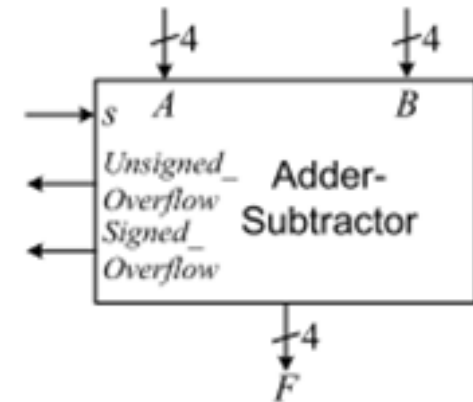
restador completo



Sumador/restador

s	Function	Operation
0	Add	$F = A + B$
1	Subtract	$F = A + B' + 1$

s	b_i	y_i	c_0
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1



```

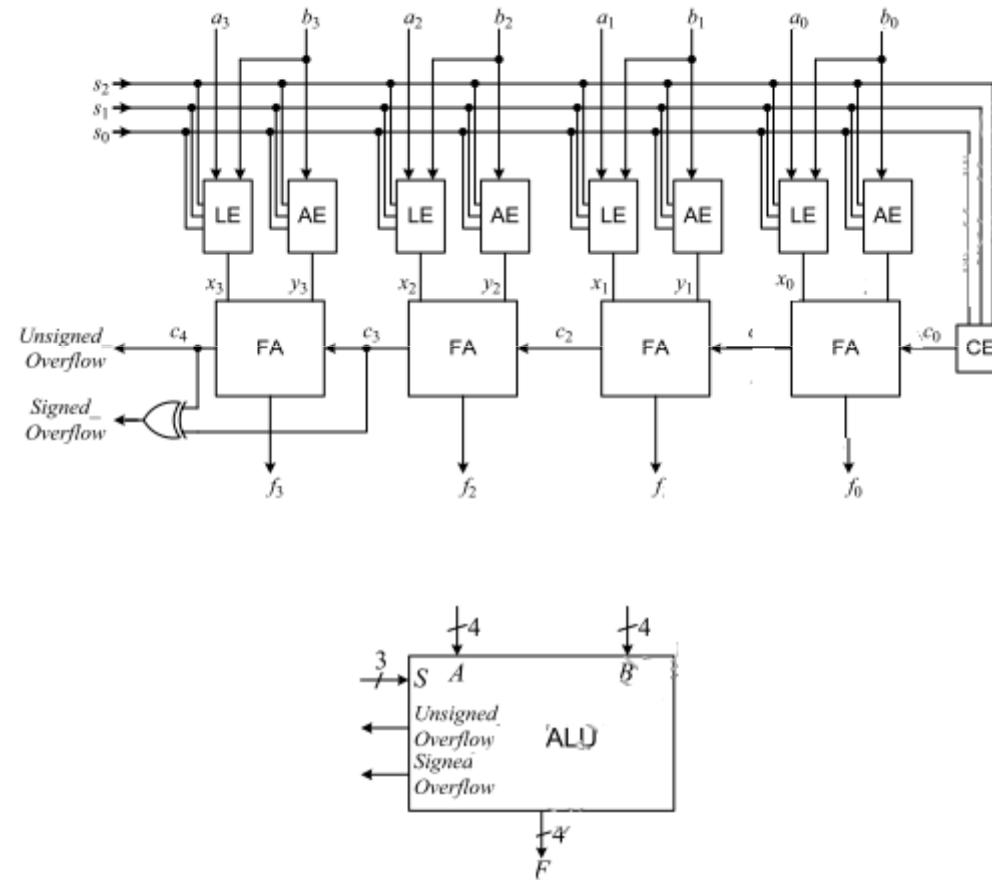
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY AddSub IS
GENERIC(n: INTEGER :=4); -- default number of bits = 4
PORT(S: IN STD_LOGIC; -- select subtract signal
      A: IN STD_LOGIC_VECTOR(n-1 DOWNT0 0);
      B: IN STD_LOGIC_VECTOR(n-1 DOWNT0 0);
      F: OUT STD_LOGIC_VECTOR(n-1 DOWNT0 0);
      unsigned_overflow: OUT STD_LOGIC;
      signed_overflow: OUT STD_LOGIC);
END AddSub;

ARCHITECTURE Behavioral OF AddSub IS
  -- temporary result for extracting the unsigned overflow bit
  SIGNAL result: STD_LOGIC_VECTOR(n DOWNT0 0);
  -- temporary result for extracting the c3 bit
  SIGNAL c3: STD_LOGIC_VECTOR(n-1 DOWNT0 0);
BEGIN
  PROCESS(S, A, B)
  BEGIN
    IF (S = '0') THEN -- addition
      -- the two operands are zero extended one extra bit before adding
      -- the & is for string concatenation
      result <= ('0' & A) + ('0' & B);
      c3 <= ('0' & A(n-2 DOWNT0 0)) + ('0' & B(n-2 DOWNT0 0));
      F <= result(n-1 DOWNT0 0); -- extract the n-bit result
      unsigned_overflow <= result(n); -- get the unsigned overflow bit
      signed_overflow <= result(n) XOR c3(n-1); -- get signed overflow bit
    ELSE -- subtraction
      -- the two operands are zero extended one extra bit before subtracting
      -- the & is for string concatenation
      result <= ('0' & A) - ('0' & B);
      c3 <= ('0' & A(n-2 DOWNT0 0)) - ('0' & B(n-2 DOWNT0 0));
      F <= result(n-1 DOWNT0 0); -- extract the n-bit result
      unsigned_overflow <= result(n); -- get the unsigned overflow bit
      signed_overflow <= result(n) XOR c3(n-1); -- get signed overflow bit
    END IF;
  END PROCESS;
END Behavioral;

```

Behavioral VHDL code for a 4-bit adder-subtractor combination component.

Unidad Aritmética/ Lógica (ALU)



s_2	s_1	s_0	Operation Name	Operation	x_i (LE)	y_i (AE)	c_0 (CE)
0	0	0	Pass	Pass A to output	a_i	0	0
0	0	1	AND	A AND B	a_i AND b_i	0	0
0	1	0	OR	A OR B	a_i OR b_i	0	0
0	1	1	NOT	A'	a_i'	0	0
1	0	0	Addition	$A + B$	a_i	b_i	0
1	0	1	Subtraction	$A - B$	a_i	b_i'	1
1	1	0	Increment	$A + 1$	a_i	0	1
1	1	1	Decrement	$A - 1$	a_i	1	0

(a)

s_2	s_1	s_0	x_i
0	0	0	a_i
0	0	1	$a_i b_i$
0	1	0	$a_i + b_i$
0	1	1	a_i'
1	\times	\times	a_i

(b)

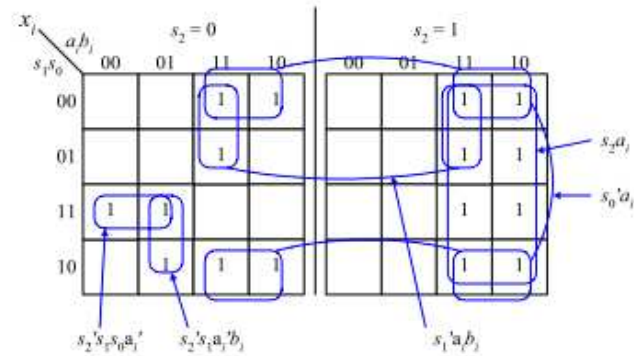
s_2	s_1	s_0	b_i	y_i
0	\times	\times	\times	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

(c)

s_2	s_1	s_0	c_0
0	\times	\times	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

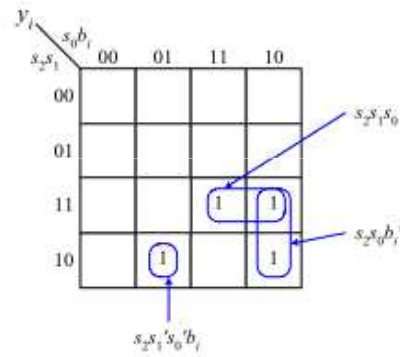
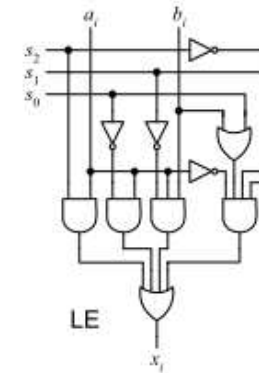
(d)

ALU operations: (a) function table; (b) LE truth table; (c) AE truth table; (d) CE truth table.



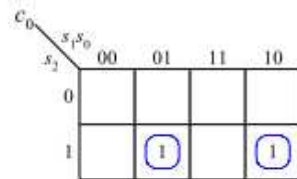
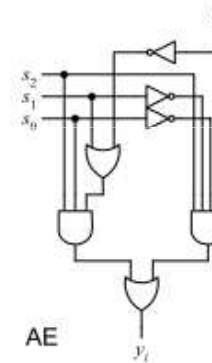
$$\begin{aligned}
 x_i &= s_2 a_i + s_0' a_i + s_1' a_i b_i + s_2' s_1 a_i' b_i + s_2' s_1 s_0 a_i' \\
 &= s_2 a_i + s_0' a_i + s_1' a_i b_i + s_2' s_1 a_i' (b_i + s_0)
 \end{aligned}$$

(a)



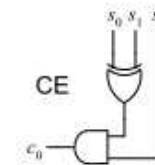
$$\begin{aligned}
 y_i &= s_2 s_1 s_0 + s_2 s_0 b_1' + s_2 s_1' s_0' b_1 \\
 &= s_2 s_0 (s_1 + b_1') + s_2 s_1' s_0' b_1
 \end{aligned}$$

(b)



$$\begin{aligned}
 c_0 &= s_2 s_1' s_0 + s_2 s_1 s_0' \\
 &= s_2 (s_1 \oplus s_0)
 \end{aligned}$$

(c)



K-maps, equations, and schematics for: (a) LE; (b) AE; and (c) CE.

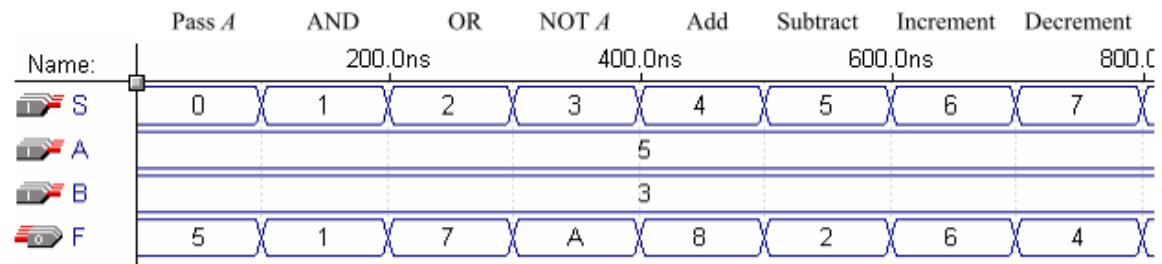
```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- The following package is needed so that the STD_LOGIC_VECTOR signals
-- A and B can be used in unsigned arithmetic operations.
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY alu IS PORT (
  S: IN STD_LOGIC_VECTOR(2 DOWNTO 0);    -- select for operations
  A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- input operands
  F: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));  -- output
END alu;

ARCHITECTURE Behavior OF alu IS
BEGIN
  PROCESS(S, A, B)
  BEGIN
    CASE S IS
      WHEN "000" => -- pass A through
        F <= A;
      WHEN "001" => -- AND
        F <= A AND B;
      WHEN "010" => -- OR
        F <= A OR B;
      WHEN "011" => -- NOT A
        F <= NOT A;
      WHEN "100" => -- add
        F <= A + B;
      WHEN "101" => -- subtract
        F <= A - B;
      WHEN "110" => -- increment
        F <= A + 1;
      WHEN OTHERS => -- decrement
        F <= A - 1;
    END CASE;
  END PROCESS;
END Behavior;

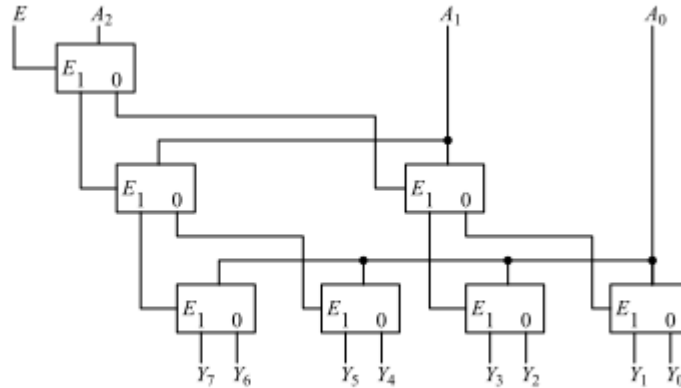
```



Sample simulation trace with the two input operands, 5 and 3, for all of the eight operations.

Behavioral VHDL code for an ALU.

Decodificador



A 3-to-8 decoder implemented with seven 1-to-2 decoders

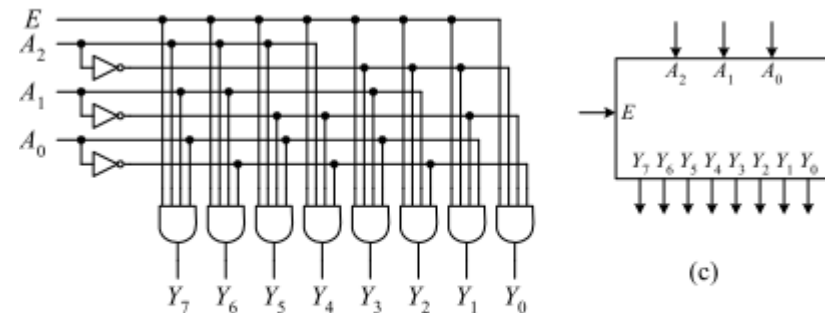
```
-- A 3-to-8 decoder
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Decoder IS PORT(
    E: IN STD_LOGIC;           -- enable
    A: IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- 3 bit address
    Y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); -- data bus output
END Decoder;

ARCHITECTURE Behavioral OF Decoder IS
BEGIN
    PROCESS (E, A)
    BEGIN
        IF (E = '0') THEN           -- disabled
            Y <= (OTHERS => '0');   -- 8-bit vector of 0
        ELSE
            CASE A IS              -- enabled
                WHEN "000" => Y <= "00000001";
                WHEN "001" => Y <= "00000010";
                WHEN "010" => Y <= "00000100";
                WHEN "011" => Y <= "00001000";
                WHEN "100" => Y <= "00010000";
                WHEN "101" => Y <= "00100000";
                WHEN "110" => Y <= "01000000";
                WHEN "111" => Y <= "10000000";
                WHEN OTHERS => NULL;
            END CASE;
        END IF;
    END PROCESS;
END Behavioral;
```

E	A ₂	A ₁	A ₀	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

(a)



(b)

(c)

A 3-to-8 decoder: (a) truth table; (b) circuit; (c) logic symbol.

Codificador

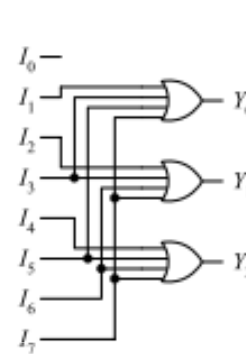
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$Y_0 = I_1 + I_3 + I_5 + I_7$$

$$Y_1 = I_2 + I_3 + I_6 + I_7$$

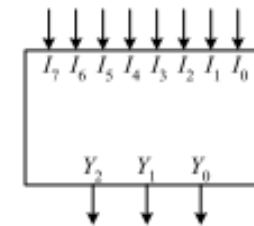
$$Y_2 = I_4 + I_5 + I_6 + I_7$$

(b)



(c)

(a)



(d)

An 8-to-3 encoder: (a) truth table; (b) equations; (c) circuit; (d) logic symbol.

Priority Encoder

I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0	Z
0	0	0	0	0	0	0	0	x	x	x	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	x	0	0	1	1
0	0	0	0	0	1	x	x	0	1	0	1
0	0	0	0	1	x	x	x	0	1	1	1
0	0	0	1	x	x	x	x	1	0	0	1
0	0	1	x	x	x	x	x	1	0	1	1
0	1	x	x	x	x	x	x	1	1	0	1
1	x	x	x	x	x	x	x	1	1	1	1

$$v_0 = I_7' I_6' I_5' I_4' I_3' I_2' I_1' I_0$$

$$v_1 = I_7' I_6' I_5' I_4' I_3' I_2' I_1$$

$$v_2 = I_7' I_6' I_5' I_4' I_3' I_2$$

$$v_3 = I_7' I_6' I_5' I_4' I_3$$

$$v_4 = I_7' I_6' I_5' I_4$$

$$v_5 = I_7' I_6' I_5$$

$$v_6 = I_7' I_6$$

$$v_7 = I_7$$

$$Y_0 = v_1 + v_3 + v_5 + v_7$$

$$Y_1 = v_2 + v_3 + v_6 + v_7$$

$$Y_2 = v_4 + v_5 + v_6 + v_7$$

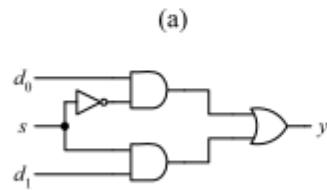
$$Z = I_7 + I_6 + I_5 + I_4 + I_3 + I_2 + I_1 + I_0$$

Multiplexor

s	d_1	d_0	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

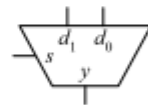
$$\begin{aligned}
 y &= s'd_1'd_0 + s'd_1d_0 + sd_1d_0' + sd_1d_0 \\
 &= s'd_0(d_1' + d_1) + sd_1(d_0' + d_0) \\
 &= s'd_0 + sd_1
 \end{aligned}$$

(b)



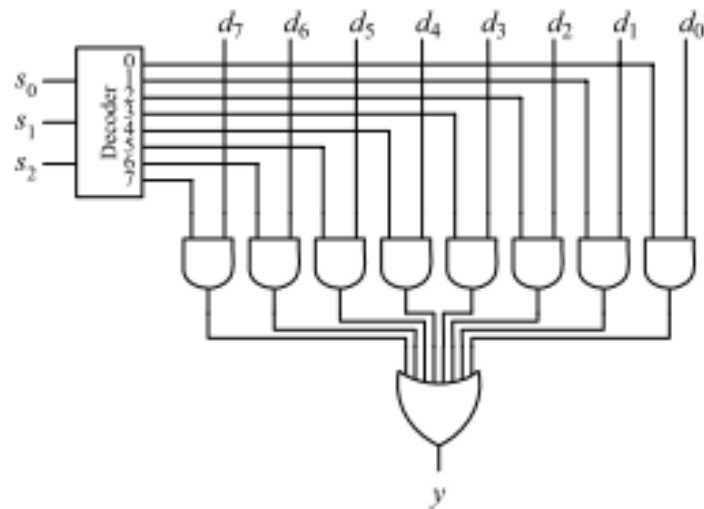
(a)

(b)

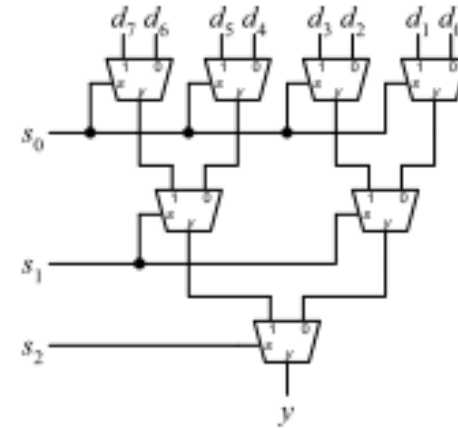


(c)

A 2-to-1 multiplexer: (a) truth table; (b) equation; (c) circuit; (d) logic symbol



(a)



(b)

An 8-to-1 multiplexer implemented using: (a) a 3-to-8 decoder; (b) seven 2-to-1 multiplexers.

```

-- A 4-to-1 8-bit wide multiplexer
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Multiplexer IS PORT (
    S: IN STD_LOGIC_VECTOR(1 DOWNTO 0);           -- select lines
    D0, D1, D2, D3: IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- data bus input
    Y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));         -- data bus output
END Multiplexer;

-- Behavioral level code
ARCHITECTURE Behavioral OF Multiplexer IS
BEGIN
    PROCESS (S, D0, D1, D2, D3)
    BEGIN
        CASE S IS
            WHEN "00" => Y <= D0;
            WHEN "01" => Y <= D1;
            WHEN "10" => Y <= D2;
            WHEN "11" => Y <= D3;
            WHEN OTHERS => Y <= (OTHERS => 'U'); -- 8-bit vector of U
        END CASE;
    END PROCESS;
END Behavioral;

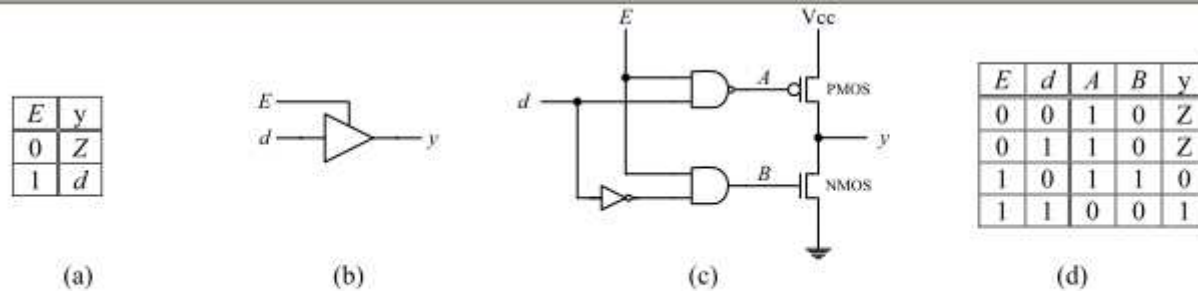
```

```

-- Dataflow level code
ARCHITECTURE Dataflow OF Multiplexer IS
BEGIN
    WITH S SELECT Y <=
        D0 WHEN "00",
        D1 WHEN "01",
        D2 WHEN "10",
        D3 WHEN "11",
        (OTHERS => 'U') WHEN OTHERS; -- 8-bit vector of U
END Dataflow;

```

Bufer de 3 estados(0,1,Z)



Tri-state buffer: (a) truth table; (b) logic symbol; (c) circuit; (d) truth table for the control portion of the tri-state buffer circuit.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY TriState_Buffer IS PORT (
    E: IN STD_LOGIC;
    d: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END TriState_Buffer;

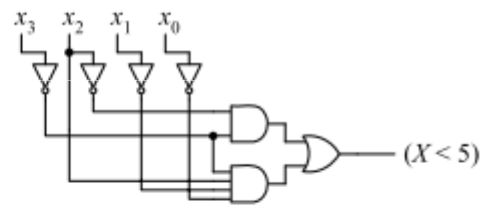
ARCHITECTURE Behavioral OF TriState_Buffer IS
BEGIN
    PROCESS (E, d)
    BEGIN
        IF (E = '1') THEN
            y <= d;
        ELSE
            y <= (OTHERS => 'Z'); -- to get 8 Z values
        END IF;
    END PROCESS;
END Behavioral;
    
```

VHDL code for an 8-bit wide tri-state buffer.

Comparador



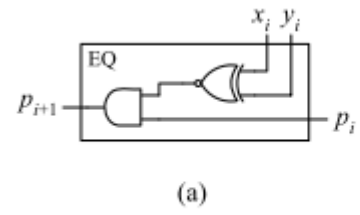
x_3	x_2	x_1	x_0	$(X < 5)$
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	x	x	x	0



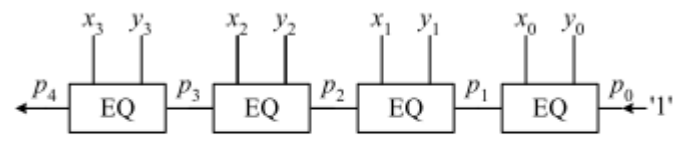
$$(X < 5) = x_3'x_2' + x_3'x_2x_1'x_0'$$

(c)

Simple 4-bit comparators for: (a) $X = 3$; (b) $X \neq Y$; (c) $X < 5$.



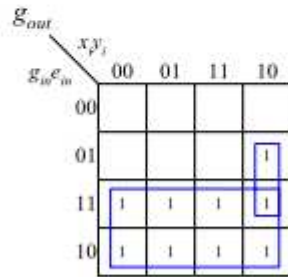
(a)



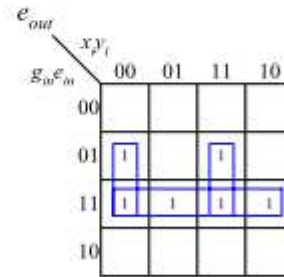
(b)

Iterative comparators: (a) 1-bit slice for $x_i = y_i$; (b) 4-bit $X = Y$.

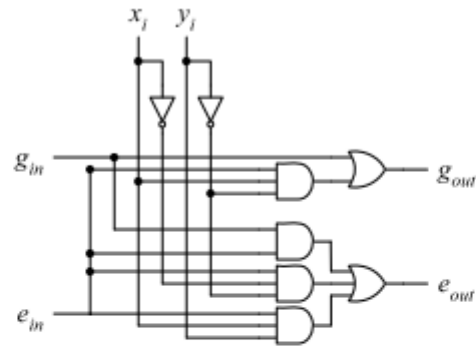
g_{in}	e_{in}	x_i	y_i	Meaning	g_{out}	e_{out}
0	0	x	x	<	0	0
0	1	0	0	=	0	1
0	1	0	1	<	0	0
0	1	1	0	>	1	0
0	1	1	1	=	0	1
1	0	x	x	>	1	0
1	1	x	x	Invalid	1	1



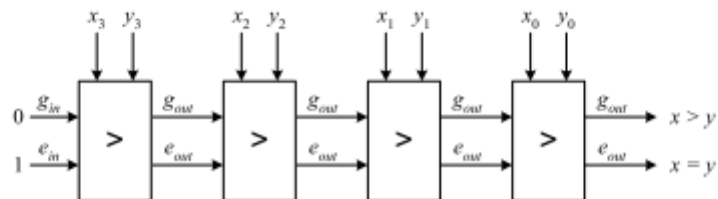
$$g_{out} = g_{in} + e_{in}x_i y_i'$$



$$e_{out} = g_{in}e_{in} + e_{in}x_i'y_i' + e_{in}x_i y_i$$

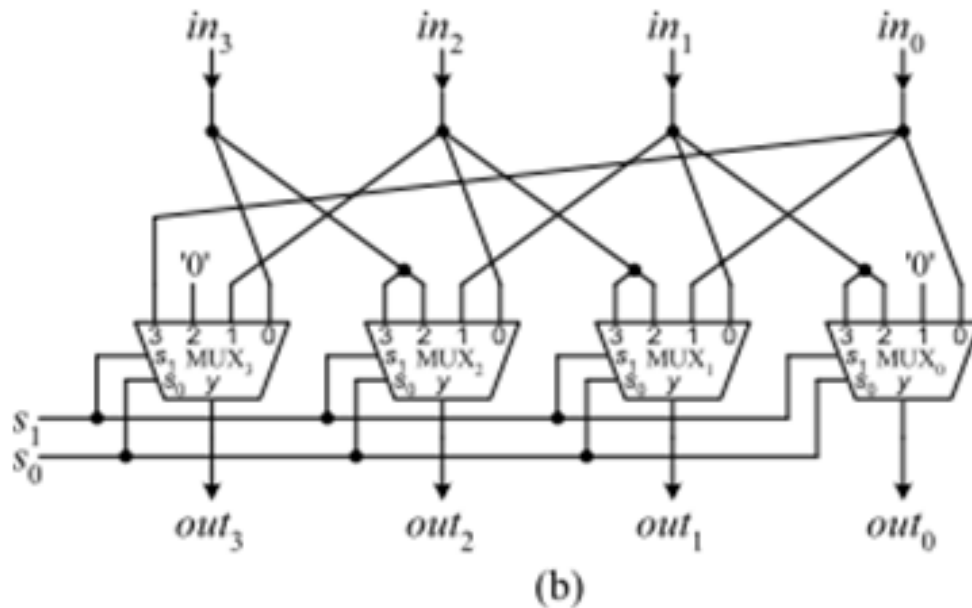


Condition	e_{out}	g_{out}
Invalid	1	1
$x = y$	1	0
$x > y$	0	1
$x < y$	0	0



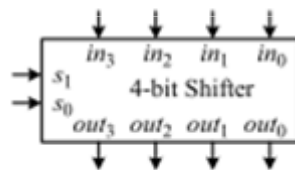
Desplazador: Shifter. (Bit cambia de posición)

Operation	Comment	Example
Shift left with 0	Shift bits to the left one position. The leftmost bit is discarded and the rightmost bit is filled with a 0.	<pre> 10110100 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ X01101000← </pre>
Shift left with 1	Same as above, except that the rightmost bit is filled with a 1.	<pre> 10110100 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ X01101001← </pre>
Shift right with 0	Shift bits to the right one position. The rightmost bit is discarded and the leftmost bit is filled with a 0.	<pre> 10110100 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ →01011010X </pre>
Shift right with 1	Same as above, except that the leftmost bit is filled with a 1.	<pre> 10110100 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ →11011010X </pre>
Rotate left	Shift bits to the left one position. The leftmost bit is moved to the rightmost bit position.	<pre> 10110100 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ 01101001 </pre>
Rotate right	Shift bits to the right one position. The rightmost bit is moved to the leftmost bit position.	<pre> 10110100 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ 01011010 </pre>



s_1	s_0	Operation
0	0	Pass through
0	1	Shift left and fill with 0
1	0	Shift right and fill with 0
1	1	Rotate right

(a)



A 4-bit shifter: (a) operation table; (b) circuit; (c) logic symbol.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY shifter IS PORT (
    S: IN STD_LOGIC_VECTOR(1 DOWNTO 0);    -- select for operation:
    input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);    -- input
    output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));    -- output
END shifter;

ARCHITECTURE Behavior OF shifter IS
BEGIN
    PROCESS(S, input)
    BEGIN
        CASE S IS
            WHEN "00" =>    -- pass through
                output <= input;
            WHEN "01" =>    -- shift left with 0
                output <= input(6 DOWNTO 0) & '0';
            WHEN "10" =>    -- shift right with 0
                output <= '0' & input(7 DOWNTO 1);
            WHEN OTHERS =>    -- rotate right
                output <= input(0) & input(7 DOWNTO 1);
        END CASE;
    END PROCESS;
END Behavior;

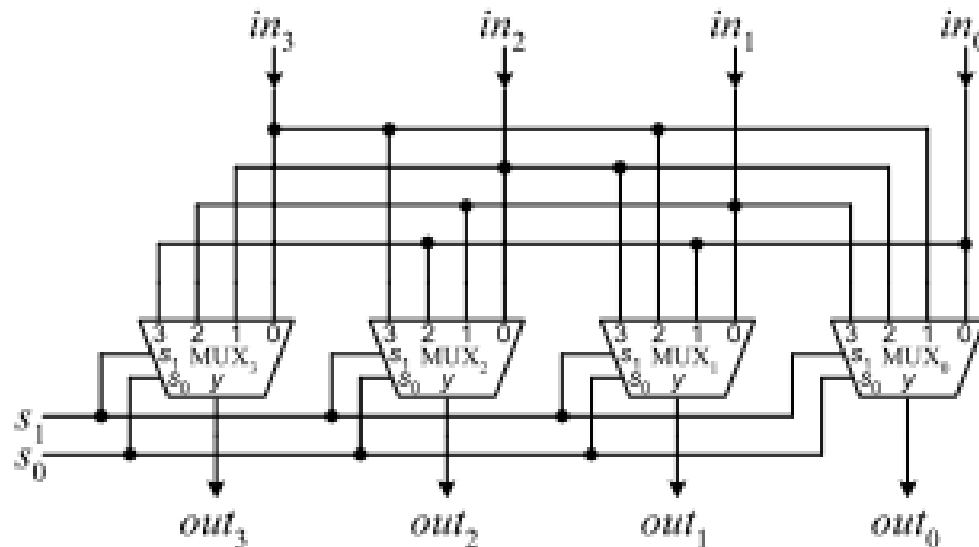
```

Barrera de desplazamiento

Select $s_1 s_0$	Operation	Output $out_3 out_2 out_1 out_0$
00	No rotation	$in_3 in_2 in_1 in_0$
01	Rotate left by 1 bit position	$in_2 in_1 in_0 in_3$
10	Rotate left by 2 bit positions	$in_1 in_0 in_3 in_2$
11	Rotate left by 3 bit positions	$in_0 in_3 in_2 in_1$

Desplaza o rota datos por un numero de bits en una sola operación

(a) operation table;



A 4-bit barrel shifter for the rotate left operation:

Multiplicación

Multiplicand (M) 1 1 0 1
 Multiplier (Q) \times 1 0 1 1
 Intermediate products {
 1 1 0 1
 0 0 0 0
 + 1 1 0 1
 Product (P) 1 0 0 0 1 1 1 1

m_3 m_2 m_1 m_0
 \times q_3 q_2 q_1 q_0
 m_3q_0 m_2q_0 m_1q_0 m_0q_0
 m_3q_1 m_2q_1 m_1q_1 m_0q_1
 m_3q_2 m_2q_2 m_1q_2 m_0q_2
 $+ m_3q_3$ m_2q_3 m_1q_3 m_0q_3
 p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0

