Titulación: Grado en Ingeniería de Computadores

Asignatura: Tecnología de Computadores

Bloque 2: Sistemas combinacionales

Tema 6: Módulos combinacionales básicos

Pablo Huerta Pellitero

ÍNDICE

BIBLIOGRAFÍA

- Román Hermida, Ana Mº del Corral, Enric Pastor, Fermín Sánchez
  "Fundamentos de Computadores" , cap 3
  Editorial Síntesis
- Thomas L. Floyd
  "Fundamentos de Sistemas Digitales", cap 6,7
  Editorial Prentice Hall
- Daniel D. Gajski
  "Principios de Diseño Digital", cap 5
  Editorial Prentice Hall
- M. Morris Mano
  "Diseño Digital", cap 4,5
  Editorial Prentice Hall

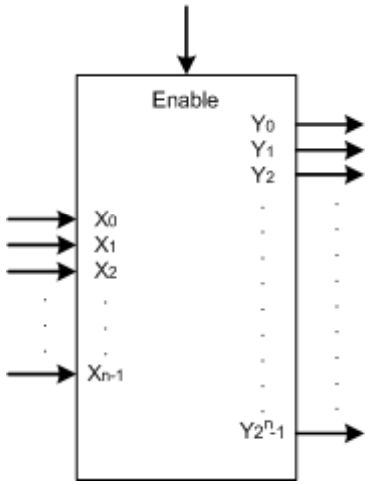
INTRODUCCIÓN

- Las materializaciones en forma de redes de puertas básicas no son adecuadas cuando la complejidad del diseño es grande. **En estos casos se realiza diseño jerárquico en base a redes de módulos combinacionales básicos**, y no mediante redes de puertas básicas.
- Ese diseño jerárquico y modular se puede llevar a cabo si se dispone de módulos que realicen funciones más complejas que las puertas básicas y que permitan dividir el diseño en partes más sencillas.
- En lo que sigue de tema se explican algunos de los módulos combinacionales de los que es conveniente disponer y cómo utilizarlos como módulos básicos para realizar diseños más complejos.
- Estos módulos son: decodificador, codificador, multiplexor, demultiplexor, desplazador, comparador, algunos módulos aritméticos y dispositivos programables.

ÍNDICE

DECODIFICADORES

- Un decodificador (o decodificador de n a 2^n) es un módulo combinacional con 'n' entradas (X_0, X_1, … X_{n-1}), y 2^n salidas (Y_0, Y_1, Y_2, …), además de una señal de activación (Enable).
- El funcionamiento del decodificador es el siguiente:
  - Si Enable está desactivada todas las salidas Yi toman el valor '0'.
  - Si Enable está activada se activará la sálida de indice I, siendo I el número decimal codificado en las entradas X. El resto salidas toman valor '0'.



- La expresión de conmutación que describe cada una de las salidas es:

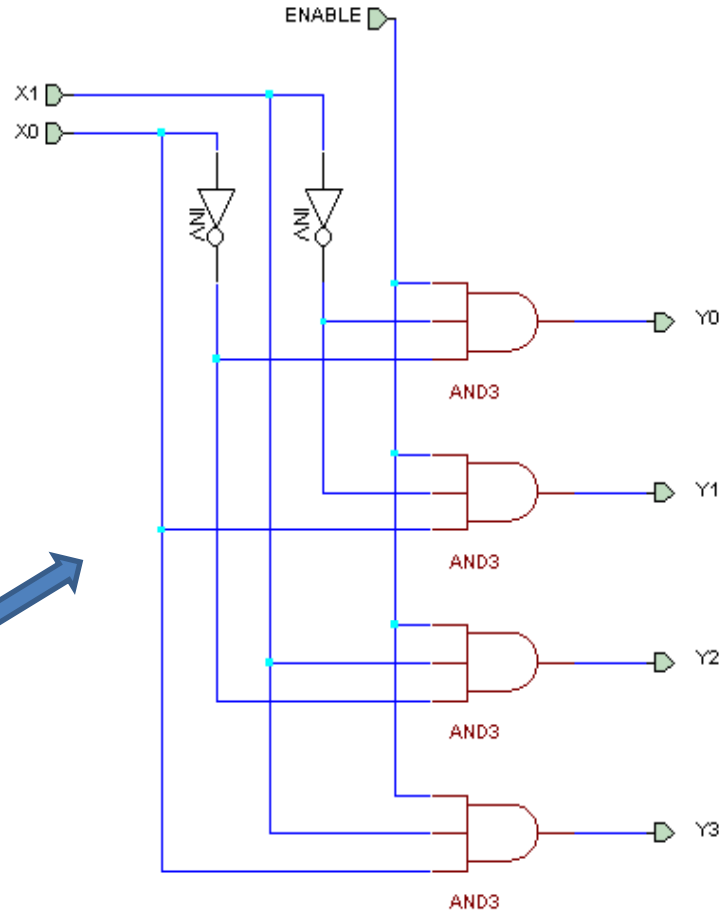
$$Y_i = Enable \cdot m_i(X_{n-1},\ldots,X_0)$$

DECODIFICADORES

- Ejemplo: implementación de un decodificador de 2 a 4 con puertas lógicas.

E	X_1	X_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

$$Y_0 = E \cdot \overline{X_1} \cdot \overline{X_0}$$
$$Y_1 = E \cdot \overline{X_1} \cdot X_0$$
$$Y_2 = E \cdot X_1 \cdot \overline{X_0}$$
$$Y_3 = E \cdot X_1 \cdot X_0$$



DECODIFICADORES

- Ejemplo (continuación): descripción VHDL.

```
library ieee;
use ieee.std_logic_1164.all;

entity deco2a4 is
  port(enable: in std_logic;
       x: in std_logic_vector(1 downto 0);
       y: out std_logic_vector(3 downto 0));
end deco2a4;

architecture puertas of deco2a4 is
signal not_x0, not_x1: std_logic;
begin
  inv_1: entity work.not1 port map(x(0), not_x0);
  inv_2: entity work.not1 port map(x(1), not_x1);
  and_1: entity work.and3 port map(enable, not_x0, not_x1, y(0));
  and_2: entity work.and3 port map(enable, x(0), not_x1, y(1));
  and_3: entity work.and3 port map(enable, not_x0, x(1), y(2));
  and_4: entity work.and3 port map(enable, x(0), x(1), y(3));
end puertas;
```

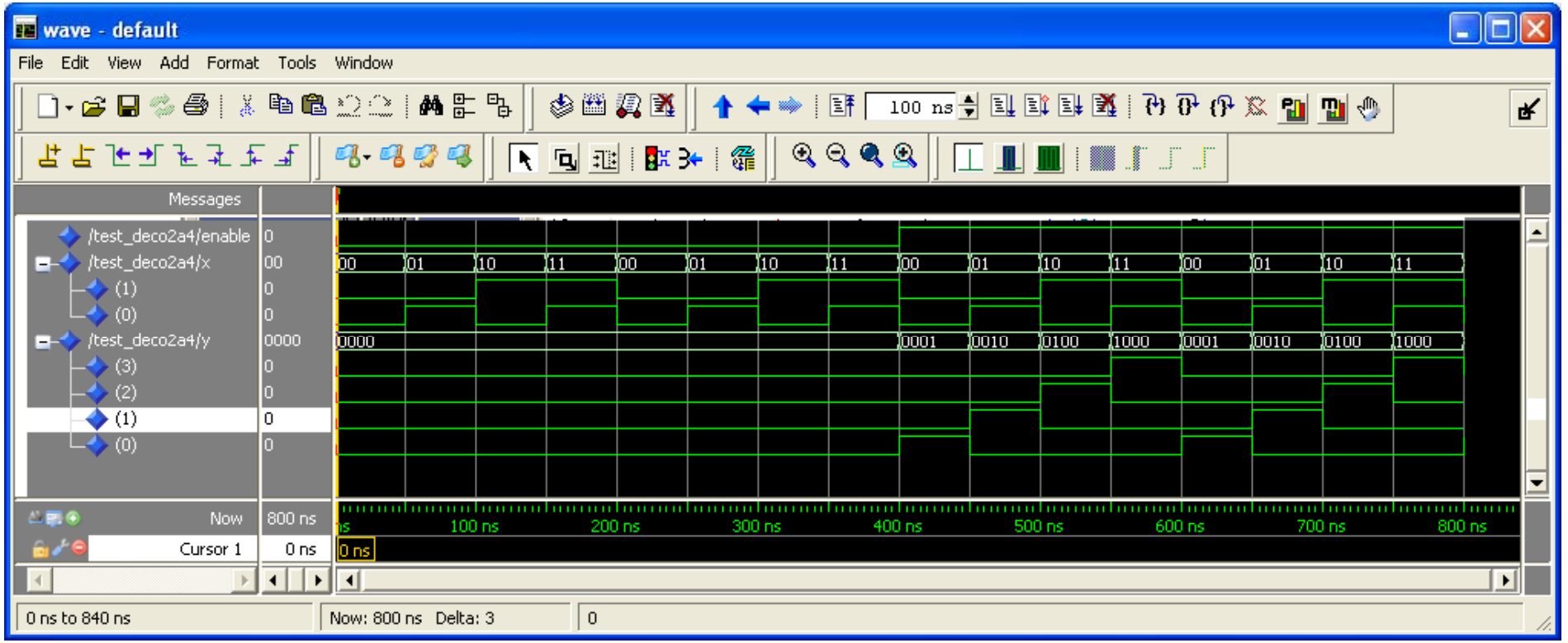
DECODIFICADORES

- Ejemplo (continuación): test-bench.

```
library ieee;
use ieee.std_logic_1164.all;

entity test_deco2a4 is
end test_deco2a4;

architecture test of test_deco2a4 is

signal enable: std_logic;
signal x: std_logic_vector(1 downto 0) := "00";
signal y: std_logic_vector(3 downto 0);

begin

inst_1: entity work.deco2a4(puertas) port map(enable, x, y);
enable <= '0', '1' after 400 ns;
x(0) <= not x(0) after 50 ns;
x(1) <= not x(1) after 100 ns;

end test;
```

DECODIFICADORES

- Ejemplo (continuación): resultado de la simulación.

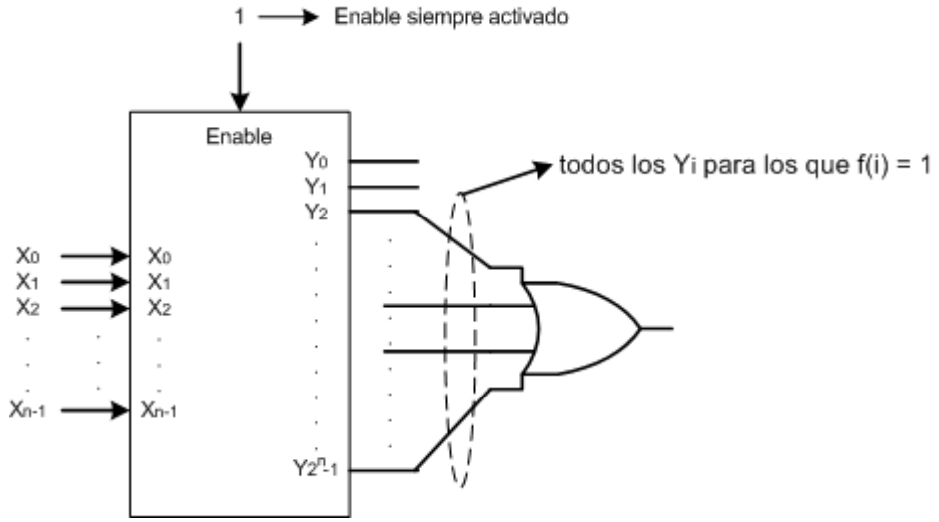


SÍNTESIS DE FC CON DECODIFICADORES

- Un decodificador de n a 2^n materializa todos los minterms de una función de n variables:

$$Y_i = Enable \cdot m_i(X_{n-1},..., X_0)$$

- Por tanto se puede materializar cualquier FC de n variables expresada como suma de minterms con **un decodificador de n a 2^n y una puerta Or con tantas entradas como sumandos tenga la expresión de la FC.**

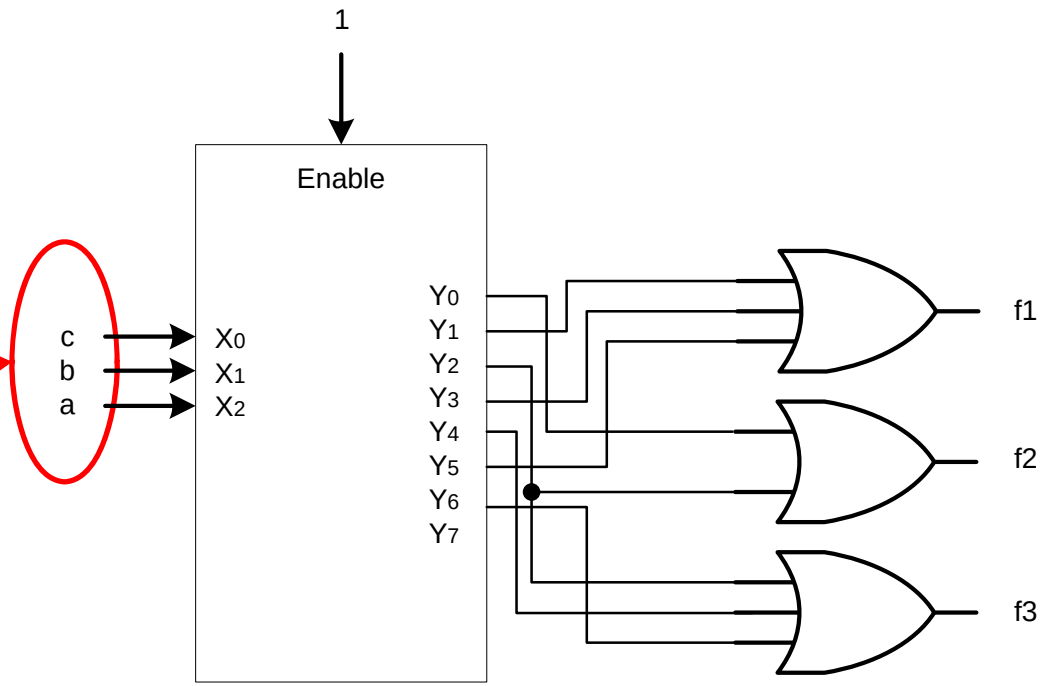


SÍNTESIS DE FC CON DECODIFICADORES

- Ejemplo: sintetizar las funciones f_1, f_2 y f_3 con un decodificador de 3 a 8.

$$f_1(a,b,c) = \sum m(1,3,5)$$
$$f_2(a,b,c) = \sum m(0,2)$$
$$f_3(a,b,c) = \sum m(2,4,6)$$

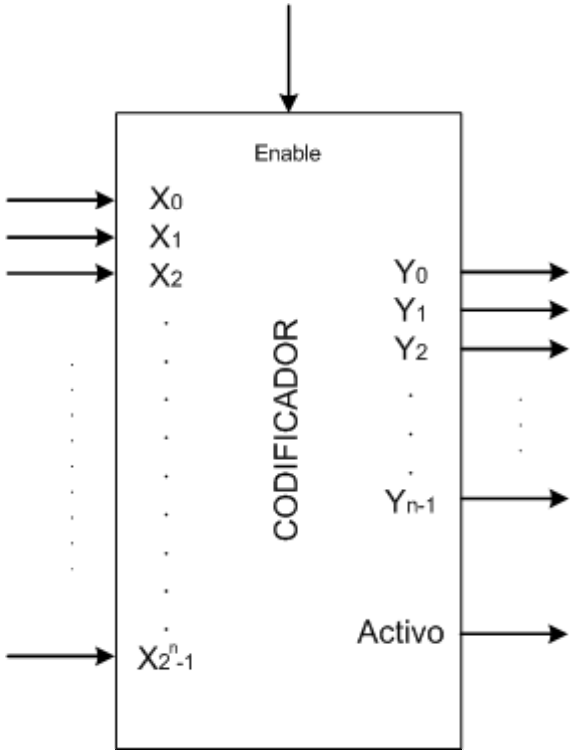
IMPORTANTE:
Cuidado con el orden de significancia de las entradas



CODIFICADORES SIN PRIORIDAD

- Un codificador (codificador de 2^n a n) es un módulo con 2^n entradas y n salidas de datos, una entrada de activación (Enable) y una salida de actividad para diseño modular.

- El funcionamiento del codificador sin prioridad es el siguiente:
  - Si Enable está desactivada todas las salidas Yi toman el valor '0'.
  - Si Enable está activada las salidas Yi codificarán el número I, siendo I el índice de la entrada X que esté activa (sólo una entrada X puede valer '1' en un instante determinado)
  - La salida Activo vale '1' si alguna de las entradas X_i vale '1' y Enable está activa. En caso contrario Activo vale '0'.



CODIFICADORES SIN PRIORIDAD

- Las expresiones de conmutación de las salidas Y_i y Activo son:

$$Y_i = E \cdot \sum X_j \qquad / \ X_j \text{ se incluye en la suma si el bit i-ésimo de la representación binaria de j es 1}$$

$$A = E \cdot (X_{2^n-1} + X_{2^n-2} + ... + X_1 + X_0)$$

- Ejemplo: codificador de 8 a 3:

E	X_{activa}	Y_2	Y_1	Y_0	A
0	-	0	0	0	0
1	X_0	0	0	0	1
1	X_1	0	0	1	1
1	X_2	0	1	0	1
1	X_3	0	0	1	1
1	X_4	1	0	0	1
1	X_5	1	0	1	1
1	X_6	1	1	0	1
1	X_7	1	1	1	1

$$y_0 = E \cdot (x_1 + x_3 + x_5 + x_7)$$

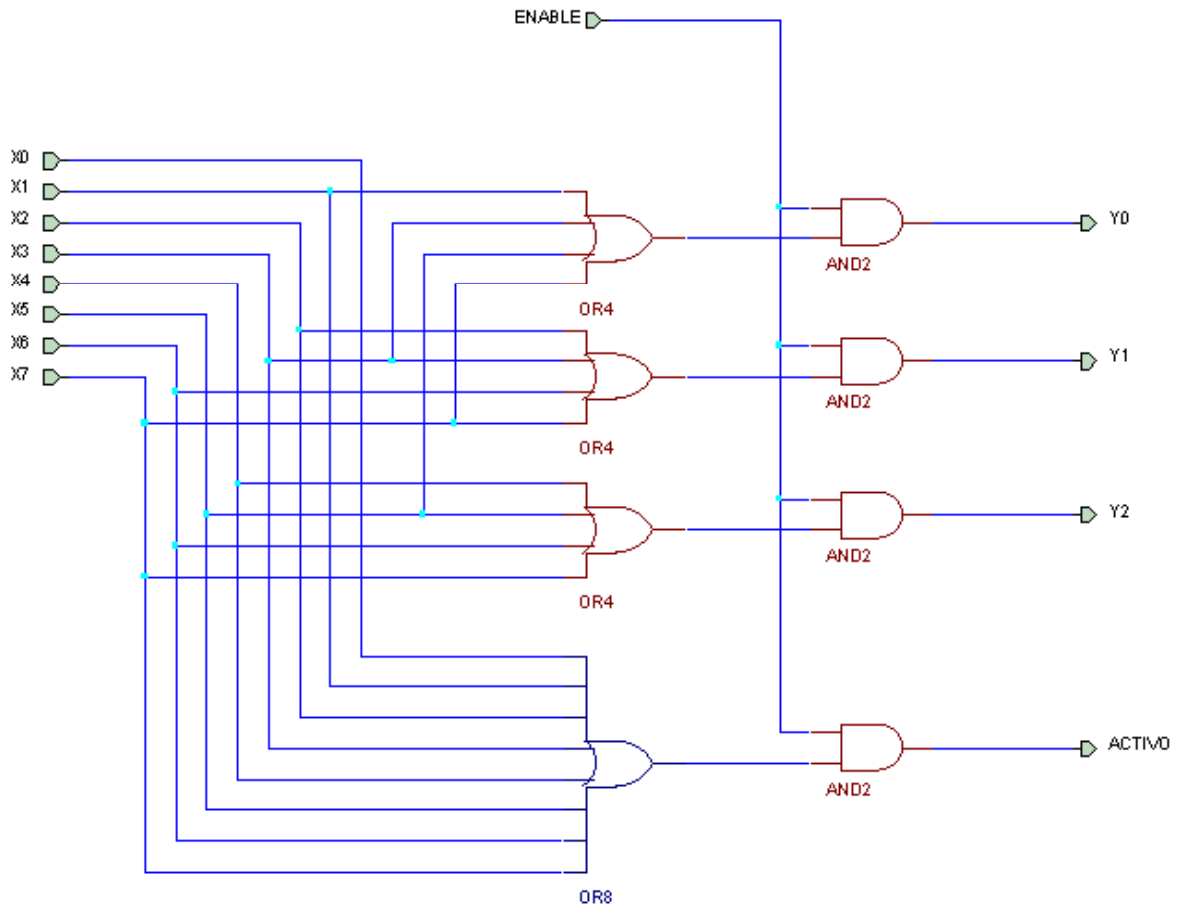
$$y_1 = E \cdot (x_2 + x_3 + x_6 + x_7)$$

$$y_2 = E \cdot (x_4 + x_5 + x_6 + x_7)$$

$$A = E \cdot (x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7)$$

CODIFICADORES SIN PRIORIDAD

- Ejemplo: codificador de 8 a 3 (continuación):



CODIFICADORES SIN PRIORIDAD

- Ejemplo (continuación): descripción estructural en VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity deco2a4 is
  port(enable: in std_logic;
       x: in std_logic_vector(1 downto 0);
       y: out std_logic_vector(3 downto 0));
end deco2a4;

architecture puertas of deco2a4 is
signal not_x0, not_x1: std_logic;
begin
  inv_1: entity work.not1 port map(x(0), not_x0);
  inv_2: entity work.not1 port map(x(1), not_x1);
  and_1: entity work.and3 port map(enable, not_x0, not_x1, y(0));
  and_2: entity work.and3 port map(enable, x(0), not_x1, y(1));
  and_3: entity work.and3 port map(enable, not_x0, x(1), y(2));
  and_4: entity work.and3 port map(enable, x(0), x(1), y(3));
end puertas;
```

CODIFICADORES SIN PRIORIDAD

- Ejemplo (continuación): otra posible descripción del circuito.

```
architecture concurrente of cod8a3 is
begin
  y(0) <= enable and (x(1) or x(3) or x(5) or x(7));
  y(1) <= enable and (x(2) or x(3) or x(6) or x(7));
  y(2) <= enable and (x(4) or x(5) or x(6) or x(7));
  activo <= enable and (x(0) or x(1) or x(2) or x(3) or x(4) or
            x(5) or x(6) or x(7));
end concurrente;
```

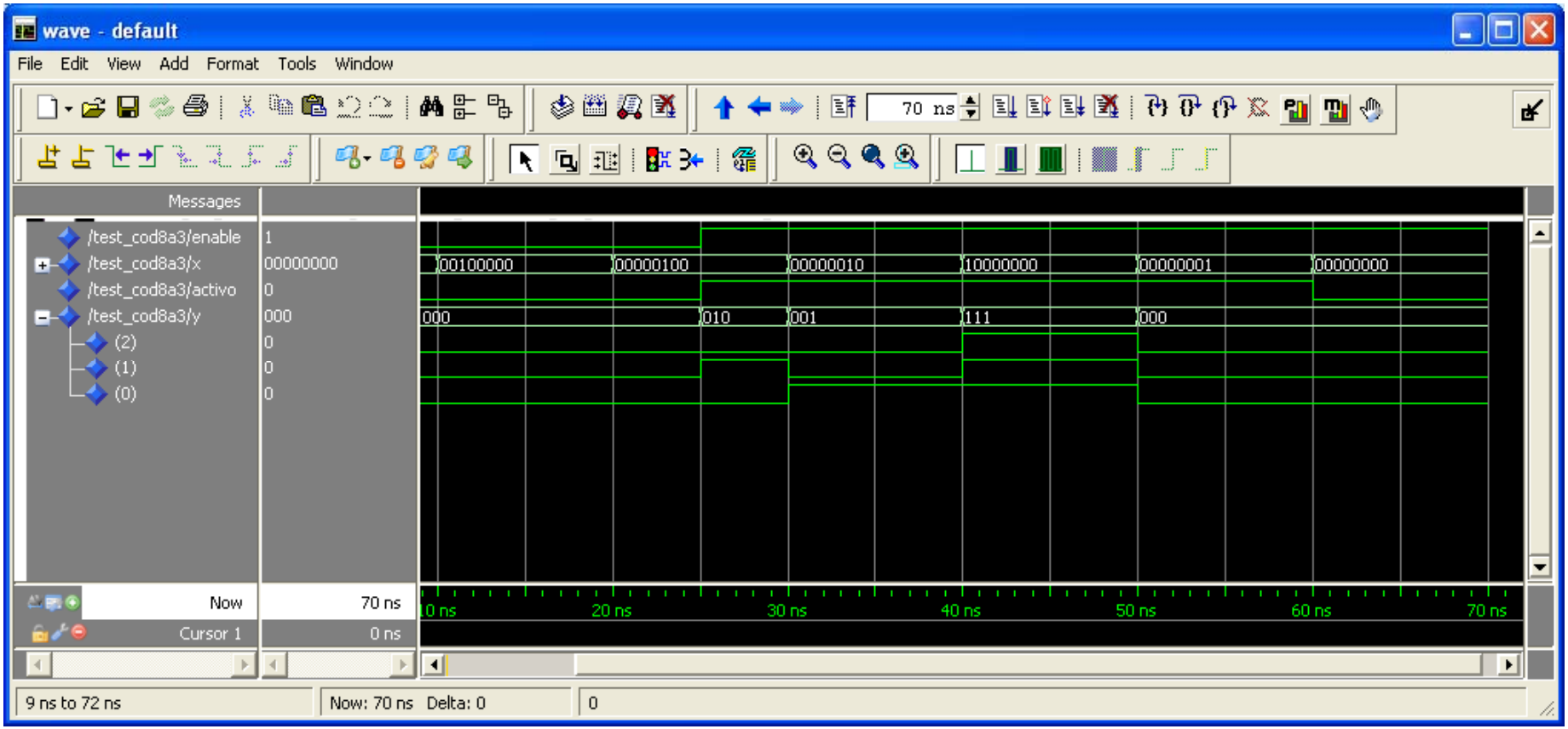
CODIFICADORES SIN PRIORIDAD

- Ejemplo (continuación): test-bench.

```
library ieee;
use ieee.std_logic_1164.all;

entity test_deco2a4 is
end test_deco2a4;

architecture test of test_deco2a4 is

signal enable: std_logic;
signal x: std_logic_vector(1 downto 0) := "00";
signal y: std_logic_vector(3 downto 0);

begin

inst_1: entity work.deco2a4(puertas) port map(enable, x, y);
enable <= '0', '1' after 400 ns;
x(0) <= not x(0) after 50 ns;
x(1) <= not x(1) after 100 ns;

end test;
```

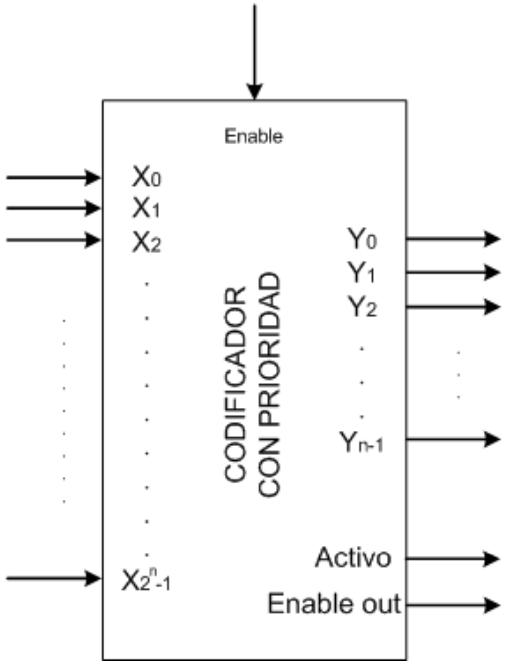
CODIFICADORES SIN PRIORIDAD

- Ejemplo (continuación): resultado de la simulación.



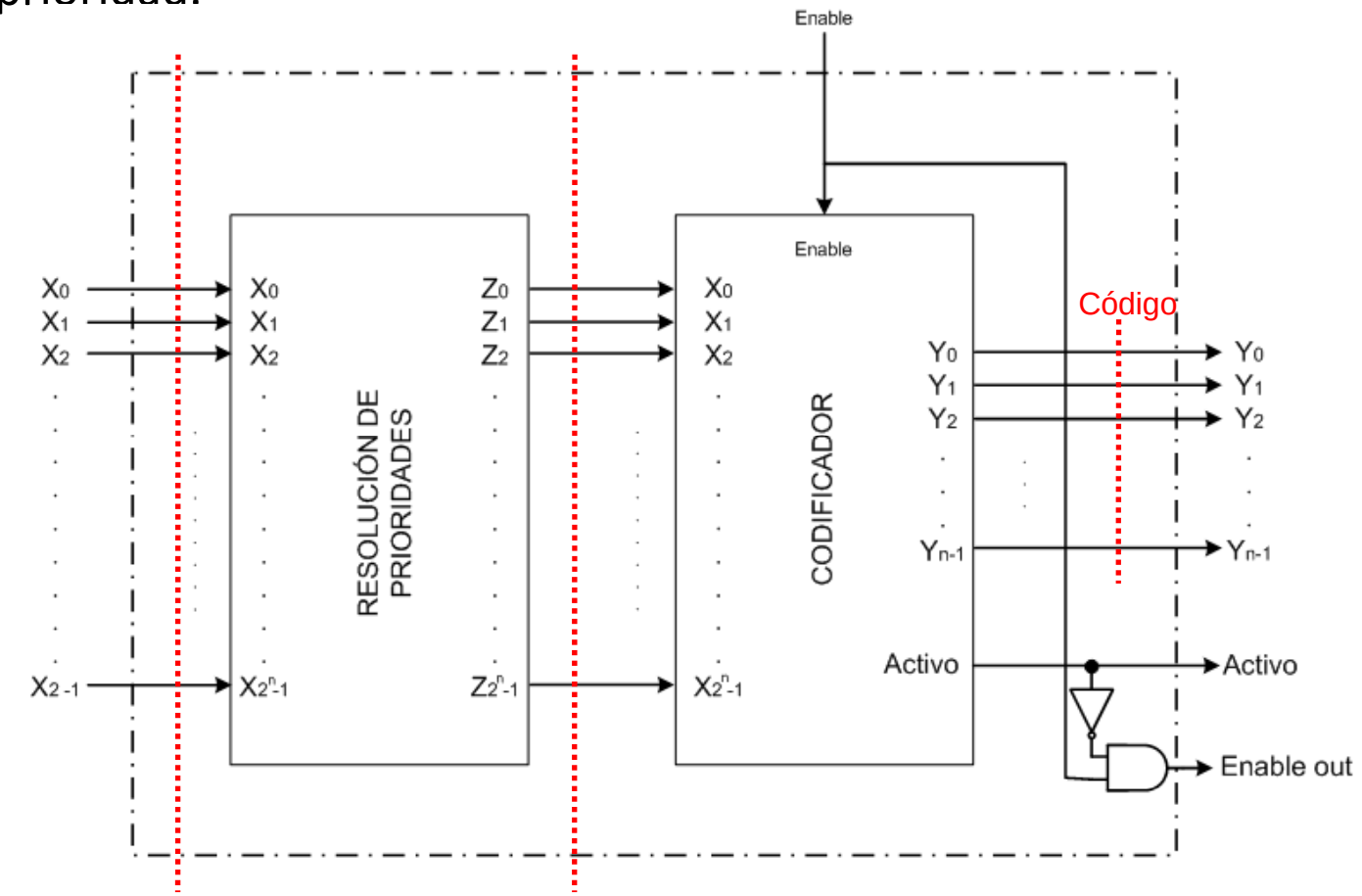
CODIFICADORES CON PRIORIDAD

- Los codificadores con prioridad reciben varias entradas de las cuales pueden estar activas más de una a la vez. La salida codifica el índice de la entrada activa más alta (o más baja si es un codificador de bit menos significativo).
- Materializan una red de resolución de prioridades que decide cuál de las entradas tiene más prioridad.
- Funcionamiento:
  - Si Enable está inactiva: Y_i, Activo y Enable out valen todas '0'.
  - Si Enable está activa:
    - Si hay algún X_i activo, la salida Y codifica el índice del mayor X_i que esté activo. La salida Activo vale '1' y la salida Enable out vale '0'.
    - Si no hay ningún X_i activo, la salida Activo vale '0' y la salida Enable out vale '1'.



CODIFICADORES CON PRIORIDAD

- Ejemplo: codificador con prioridad en función de un codificador sin prioridad.



CODIFICADORES CON PRIORIDAD

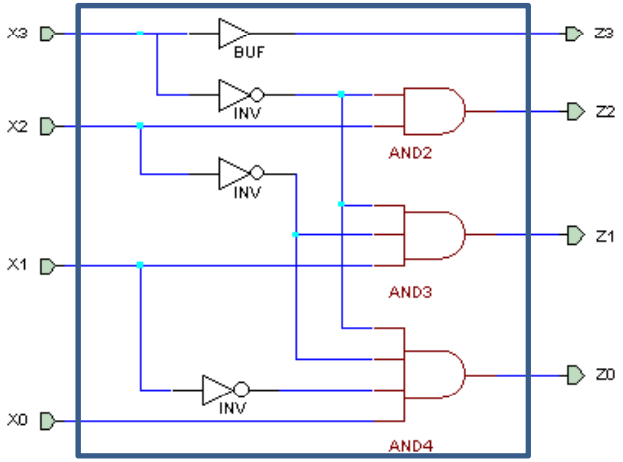
- El módulo de resolución de prioridades de bit más significativo se materializa con las siguientes expresiones de conmutación:

$$Z_{2^n-1} = X_{2^n-1}$$
$$Z_{2^n-2} = X_{2^n-2} \cdot \overline{X_{2^n-1}}$$
$$Z_{2^n-3} = X_{2^n-3} \cdot \overline{X_{2^n-2}} \cdot \overline{X_{2^n-1}}$$
$$\cdot$$
$$\cdot$$
$$Z_i = X_i \cdot \overline{X_{i+1}} \cdot \ldots \cdot \overline{X_{2^n-2}} \cdot \overline{X_{2^n-1}}$$

CODIFICADORES CON PRIORIDAD

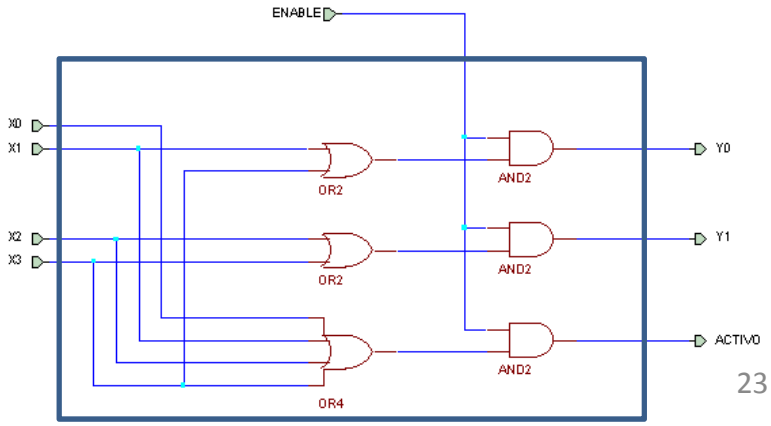
- Ejemplo: sintetizar un codificador con prioridad de 4 a 2.
- Módulo de resolución de prioridades:

$$Z_3 = X_3$$
$$Z_2 = X_2 \cdot \overline{X_3}$$
$$Z_1 = X_1 \cdot \overline{X_2} \cdot \overline{X_3}$$
$$Z_0 = X_0 \cdot \overline{X_1} \cdot \overline{X_2} \cdot \overline{X_3}$$

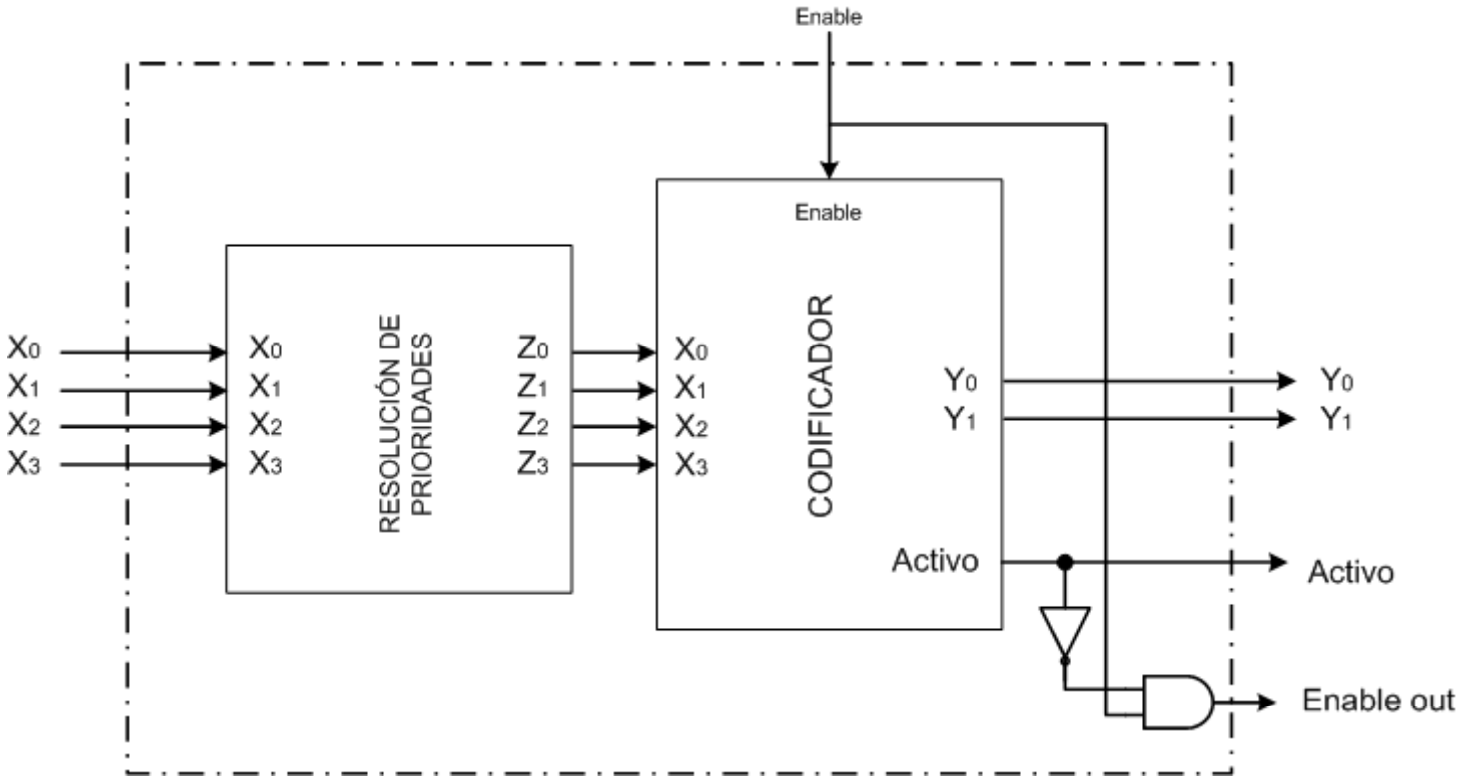


- Codificador 4 a 2:

$$Y_0 = E \cdot (X_1 + X_3)$$
$$Y_1 = E \cdot (X_2 + X_3)$$
$$A = E \cdot (X_0 + X_1 + X_2 + X_3)$$



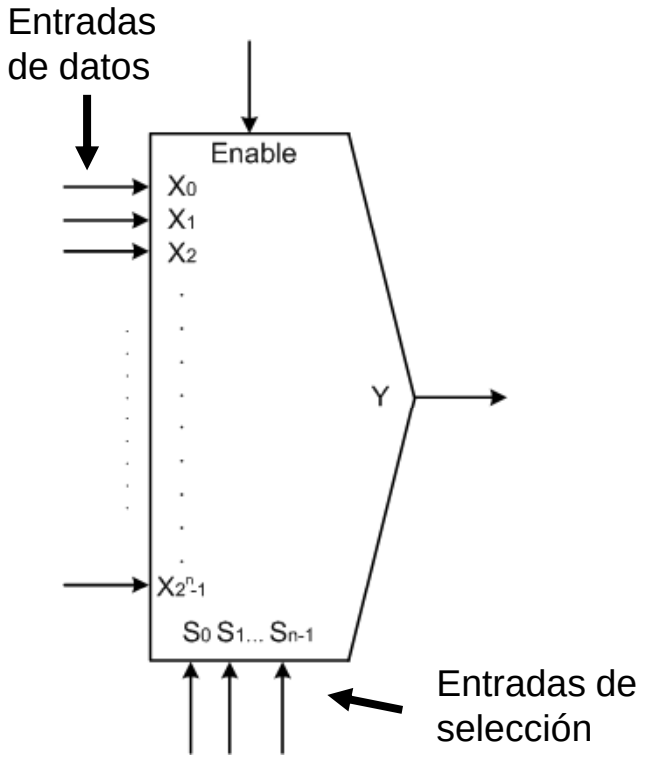
CODIFICADORES CON PRIORIDAD



ÍNDICE

MULTIPLEXORES

- Un multiplexor (o multiplexor de 2^n a 1) es un módulo combinacional con 2^n entradas de datos, 'n' entradas de selección y una entrada de activación (Enable). Tiene una única salida.
- El multiplexor conecta una de las 2^n entradas a la salida en función de una palabra de control S:
  - Si las entradas de selección S_i codifican el número I, la salida Y tomará el valor de X_i siempre que Enable esté activo.
  - Si Enable está inactivo la salida Y vale '0'.



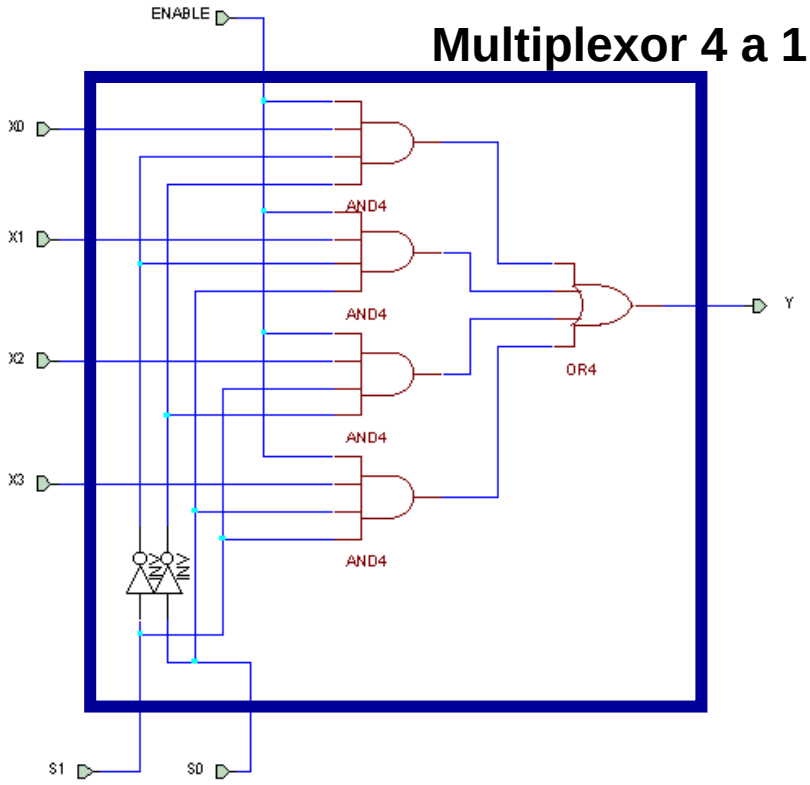
- Puesto en forma de expresión de conmutación:

$$y = E \cdot (x_0 \cdot m_0(s_{n-1},..., s_0) + x_1 \cdot m_1(s_{n-1},..., s_0) + ...) = E \cdot \sum_{i=0}^{2^n-1} x_i \cdot m_i(s_{n-1},..., s_0)$$

MULTIPLEXORES

- Ejemplo: implementar un multiplexor de 4 entradas.

$$y=E\cdot(x_0\cdot m_0(s_1,s_0)+x_1\cdot m_1(s_1,s_0)+x_2\cdot m_2(s_1,s_0)+x_3\cdot m_3(s_1,s_0))=$$
$$E\cdot(x_0\cdot\overline{s_1}\cdot\overline{s_0}+x_1\cdot\overline{s_1}\cdot s_0+x_2\cdot s_1\cdot\overline{s_0}+x_3\cdot s_1\cdot s_0)$$



MULTIPLEXORES

- Ejemplo (continuación): descripción estructural en VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity mux4a1 is
  port (enable: in std_logic;
        x: in std_logic_vector(3 downto 0);
        sel: in std_logic_vector(1 downto 0);
        y: out std_logic);
end mux4a1;

architecture puertas of mux4a1 is
signal not_sel1, not_sel0: std_logic;
signal s0, s1, s2, s3: std_logic;
begin
    not1: entity work.not1 port map(sel(0), not_sel0);
    not2: entity work.not1 port map(sel(1), not_sel1);
    and1: entity work.and4 port map(enable, not_sel0, not_sel1, x(0), s0);
    and2: entity work.and4 port map(enable, sel(0), not_sel1, x(1), s1);
    and3: entity work.and4 port map(enable, not_sel0, sel(1), x(2), s2);
    and4: entity work.and4 port map(enable, sel(0), sel(1), x(3), s3);
    or1:  entity work.or4  port map(s0, s1, s2, s3, y);
end puertas;
```

MULTIPLEXORES

- Ejemplo (continuación): otra posible descripción en VHDL

```
architecture concurrente of mux4a1 is
begin
  y <= enable and (
       ( (not sel(0)) and (not sel(1)) and x(0) ) or
       ( sel(0) and (not sel(1)) and x(1) ) or
       ( (not sel(0)) and sel(1) and x(2) ) or
       ( sel(0) and sel(1) and x(3) )
       );
end concurrente;
```

MULTIPLEXORES

- Ejemplo (continuación): y otra posible descripción más en VHDL utilizando un ***process***.

```
architecture funcional of mux4a1 is
begin
  process(x, sel)
    begin
      if enable = '0' then
        y <= '0';
      else
        case sel is
          when "00" => y <= x(0);
          when "01" => y <= x(1);
          when "10" => y <= x(2);
          when others => y <= x(3);
        end case;
      end if;
    end process;
end funcional;
```

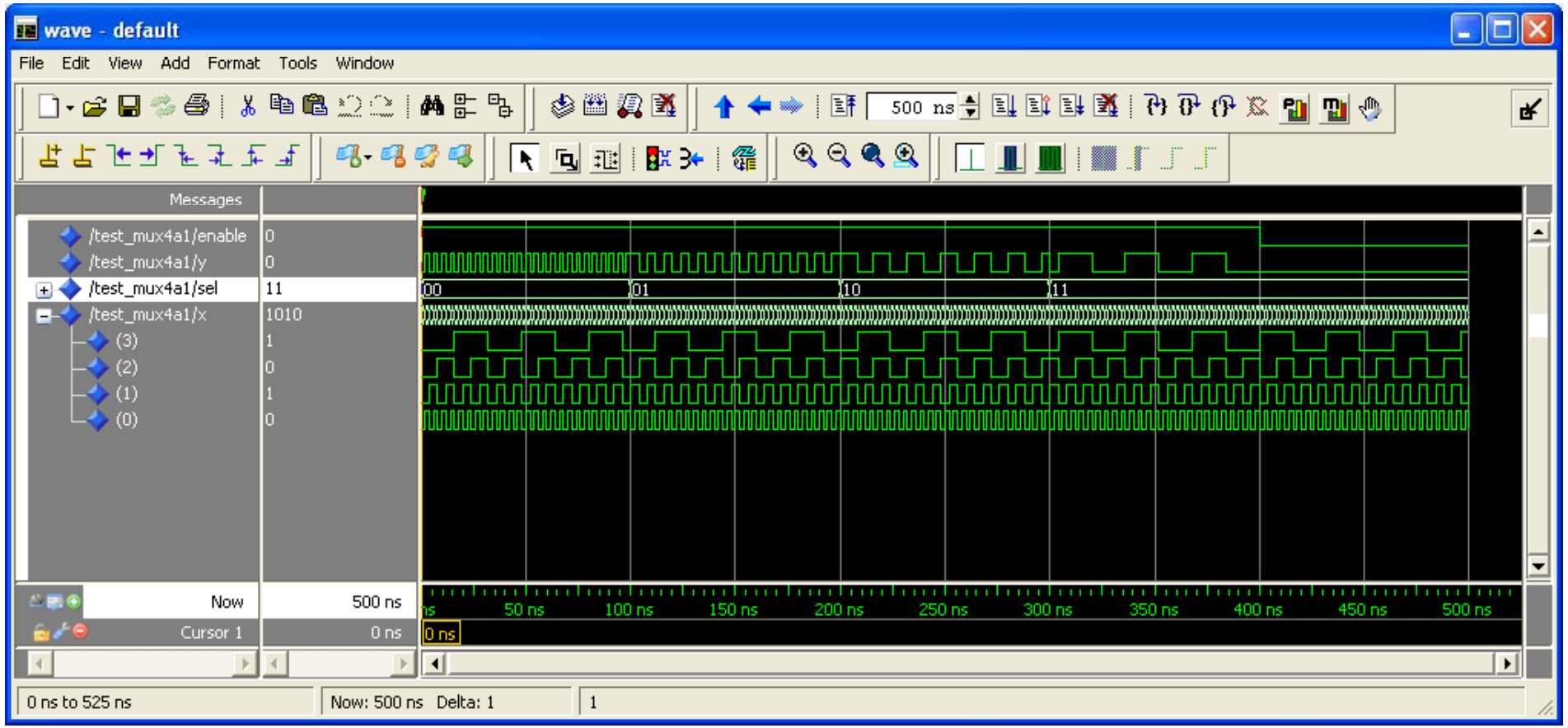
MULTIPLEXORES

- Ejemplo (continuación): test-bench

```
library ieee;
use ieee.std_logic_1164.all;

entity test_mux4a1 is
end test_mux4a1;

architecture test of test_mux4a1 is
signal enable, y: std_logic;
signal x: std_logic_vector(3 downto 0) := "0000";
signal sel: std_logic_vector(1 downto 0);
begin
  inst1: entity work.mux4a1(puertas) port map(enable, x, sel, y);

  enable <= '1', '0' after 400 ns;
  x(0) <= not x(0) after 2 ns;
  x(1) <= not x(1) after 4 ns;
  x(2) <= not x(2) after 8 ns;
  x(3) <= not x(3) after 16 ns;
  sel <= "00", "01" after 100 ns, "10" after 200 ns, "11" after 300 ns;

end test;
```

MULTIPLEXORES

- Ejemplo (continuación): resultado de la simulación



SÍNTESIS DE FC CON MULTIPLEXORES

- Un único multiplexor de 2^n a 1 permite materializar cualquier función de conmutación de 'n' variables.
- La EC de una FC como suma de productos consiste en la suma de los minterms m_i para los que la FC, f(i) toma valor cierto, es decir:

$$f(a_{n-1},..., a_0) = \sum_{i=0}^{2^n-1} f(i) \cdot m_i(a_{n-1},..., a_0)$$

- La EC de un multiplexor es:

$$y = E \cdot \sum_{i=0}^{2^n-1} x_i \cdot m_i(s_{n-1},..., s_0)$$

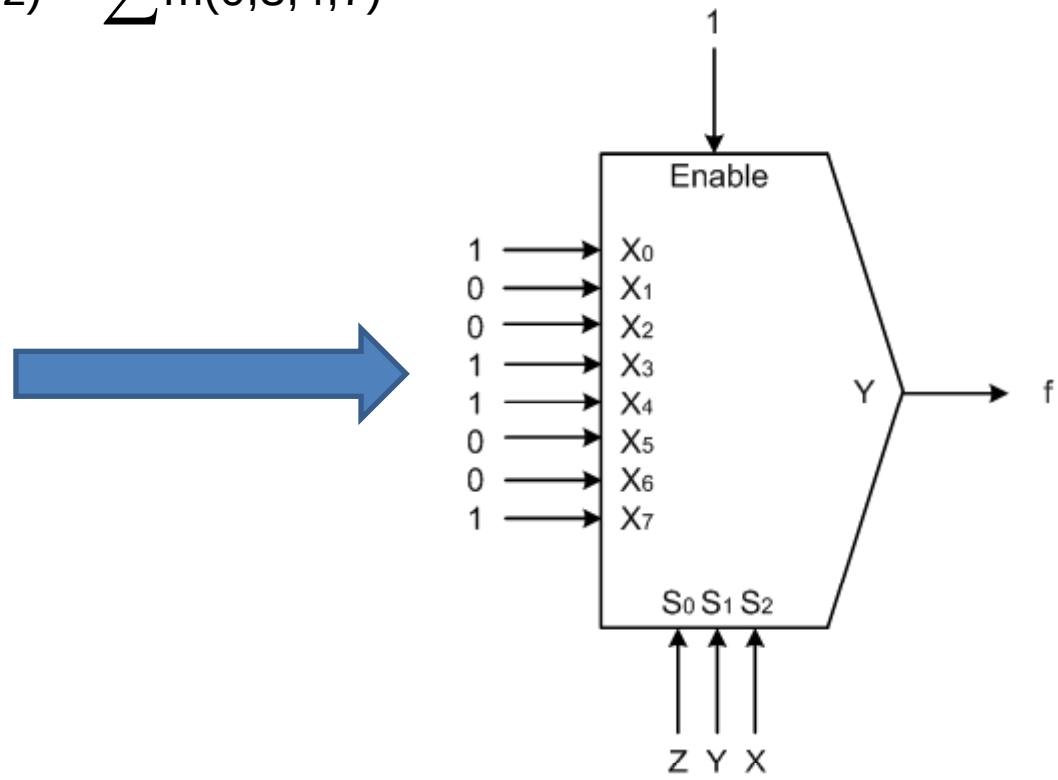
- Por tanto, podemos hacer que $y = f(a_{n-1},\ldots,a_0)$ si hacemos:
  - Que Enable = '1'.
  - Que las entradas S_i sean las variables de la función.
  - Que cada entrada X_i del multiplexor sea igual a f(i) (el valor de la fila i de la tabla de verdad).

SÍNTESIS DE FC CON MULTIPLEXORES

- Ejemplo: sintetizar f(x,y,z) mediante un único multiplexor.

$$f(x, y, z) = \sum m(0,3,4,7)$$

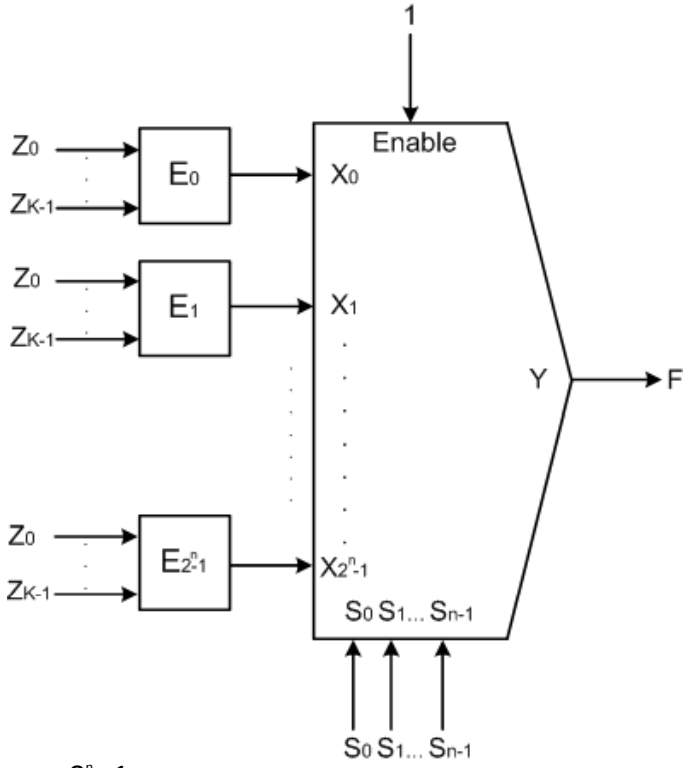
X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



SÍNTESIS DE FC CON MULTIPLEXORES

- La materialización de FCs mediante multiplexores no es directa cuando no se dispone de un multiplexor con tantas entradas de control como variables binarias tiene la FC.

- Dado un multiplexor de 2^n a 1 (es decir, con 'n' entradas de selección) y una función con 'n+k' variables binarias, se seleccionan 'n' variables como entrada de control y se usan las 'k' restantes como variables de las 2^n funciones de entrada al multiplexor.



$$f(x_{n+k-1},\ldots, x_n, x_{n-1},\ldots, x_0) = f(z_{k-1},\ldots, z_0, s_{n-1},\ldots, s_0) = \sum_{i=0}^{2^n-1} E_i(z_{k-1},\ldots, z_0)\cdot m_i(s_{n-1},\ldots, s_0)$$

SÍNTESIS DE FC CON MULTIPLEXORES

- Cada función de k entradas puede realizarse con multiplexores de k entradas de control (árbol de multiplexores), con decodificadores, con puertas lógicas, etc.
- Ejemplo: implementar f mediante un multiplexor de 4 a 1 y puertas lógicas. (Se ha escogido c y d como entradas de selección del multiplexor)

$$f(a,b,c,d)=\sum m(1,3,4,6,7,9,10,11,14)$$

$$f(a,b,c,d)=\bar{a}\cdot\bar{b}\cdot\bar{c}\cdot d+\bar{a}\cdot\bar{b}\cdot c\cdot d+\bar{a}\cdot b\cdot\bar{c}\cdot\bar{d}+\bar{a}\cdot b\cdot c\cdot\bar{d}+\bar{a}\cdot b\cdot c\cdot d+a\cdot\bar{b}\cdot\bar{c}\cdot d+a\cdot\bar{b}\cdot c\cdot\bar{d}+a\cdot\bar{b}\cdot c\cdot d+a\cdot b\cdot c\cdot\bar{d}=$$

$$=(\bar{a}\cdot b)\cdot\bar{c}\cdot\bar{d}+(\bar{a}\cdot\bar{b}+a\cdot\bar{b})\cdot\bar{c}\cdot d+(\bar{a}\cdot\bar{b}+a\cdot\bar{b}+a\cdot b)\cdot c\cdot\bar{d}+(\bar{a}\cdot\bar{b}+\bar{a}\cdot b+a\cdot\bar{b})\cdot c\cdot d$$

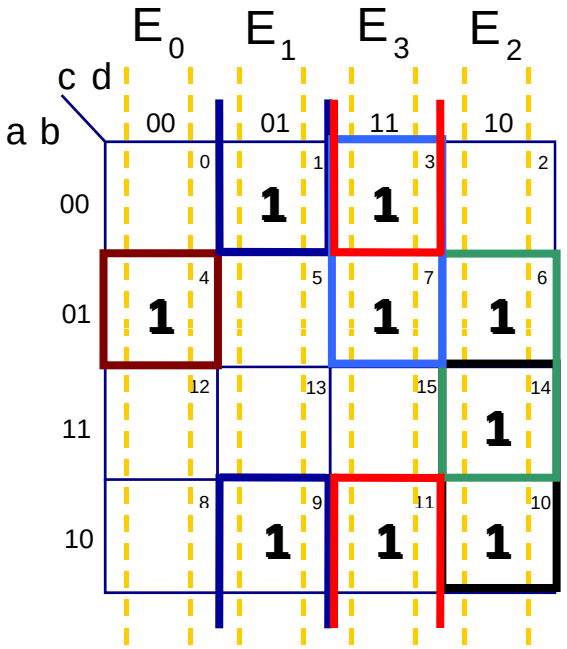
E0 E1 E2 E3

$$E_0=\bar{a}\cdot b$$
$$E_1=\bar{a}\cdot\bar{b}+a\cdot\bar{b}=\bar{b}$$
$$E_2=\bar{a}\cdot\bar{b}+a\cdot\bar{b}+a\cdot b=a+b$$
$$E_3=\bar{a}\cdot\bar{b}+\bar{a}\cdot b+a\cdot\bar{b}=\bar{a}+\bar{b}$$

Simplificación algebraica

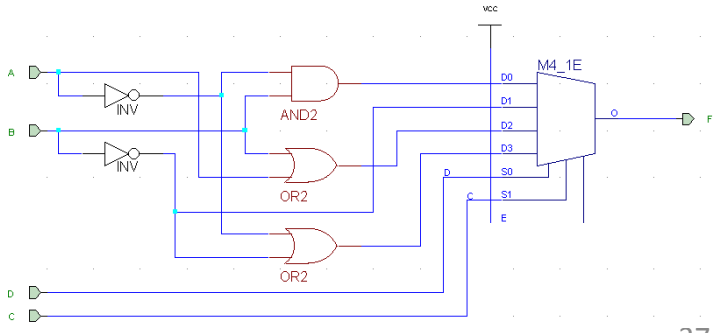
SÍNTESIS DE FC CON MULTIPLEXORES

- También se puede resolver de forma gráfica con mapas de Karnaugh.
- Se trata de dividir el mapa en regiones en las que las variables que se utilizarán como entradas de selección del multiplexor tienen valores constantes.
- Ejemplo: para la misma función que la transparencia anterior:



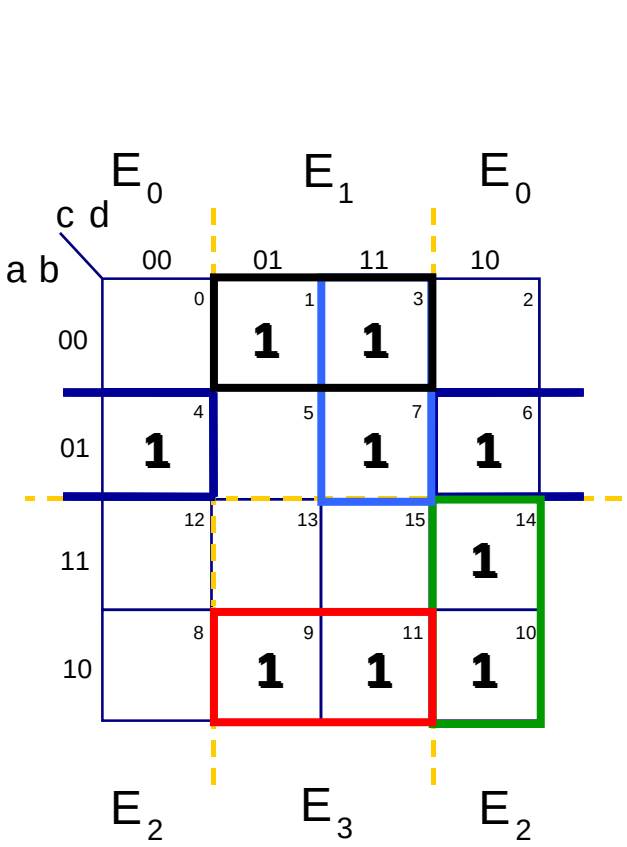
Se divide el mapa en las regiones en las que ‘c’ y ‘d’ valen ‘00’ => E0; ‘01’ => E1, etc.
Cada región se trata como un mapa de Karnaugh de 2 variables (‘a’ y ‘b’)

$$E_0 = \overline{a} \cdot b$$
$$E_1 = \overline{b}$$
$$E_2 = a + b$$
$$E_3 = \overline{a} + \overline{b}$$

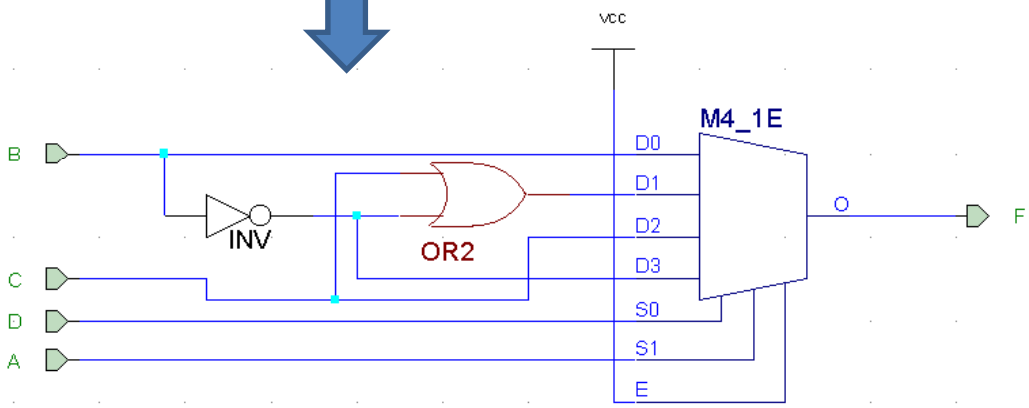


SÍNTESIS DE FC CON MULTIPLEXORES

- Si se escogen otras variables de selección (‘a’ y ‘d’ en este ejemplo) el circuito resultante es diferente.



$$E_0 = b$$
$$E_1 = \bar{b} + c$$
$$E_2 = c$$
$$E_3 = \bar{b}$$



SÍNTESIS DE FC CON MULTIPLEXORES

- Ejemplo: descripción en VHDL de los dos circuitos anteriores.

```
library ieee;
use ieee.std_logic_1164.all;

entity sintesis_mux is
  port(a,b,c,d: in std_logic;
       z: out std_logic);
end sintesis_mux;

architecture arq_1 of sintesis_mux is
signal E: std_logic_vector(3 downto 0);
signal sel: std_logic_vector(1 downto 0);
begin
  E(0) <= (not a) and b;
  E(1) <= (not b);
  E(2) <= a or b;
  E(3) <= (not a) or (not b);
  sel(0) <= d;
  sel(1) <= c;
  mux: entity work.mux4a1(funcional) port map('1', E, sel, z);
end arq_1;
```

SÍNTESIS DE FC CON MULTIPLEXORES

- Ejemplo (continuación): descripción en VHDL de los dos circuitos anteriores.

```
architecture arq_2 of sintesis_mux is
signal E: std_logic_vector(3 downto 0);
signal sel: std_logic_vector(1 downto 0);
begin
  E(0) <= b;
  E(1) <= (not b) or c;
  E(2) <= c;
  E(3) <= (not b);
  sel(0) <= d;
  sel(1) <= a;
  mux: entity work.mux4a1(concurrente) port map('1', E, sel, z);
end arq_2;
```

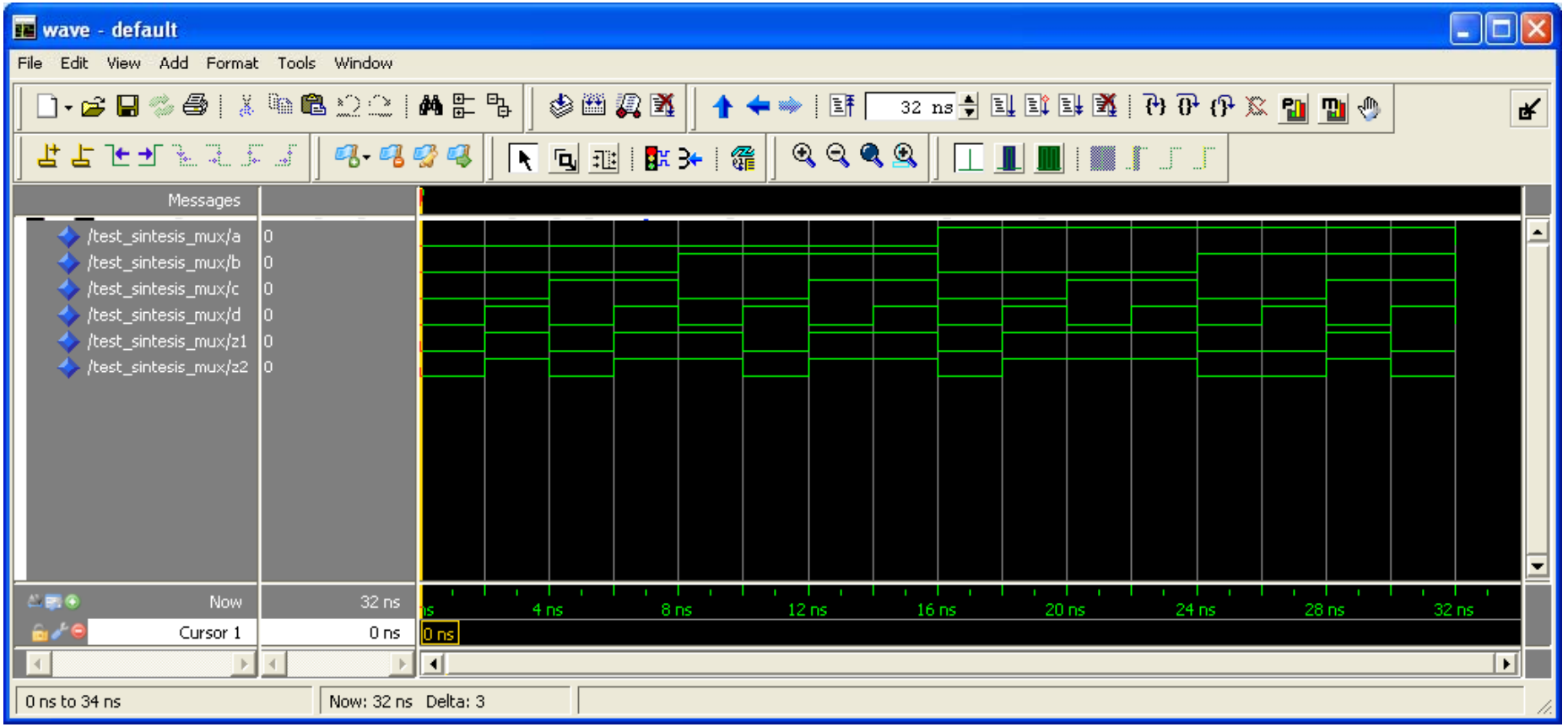
SÍNTESIS DE FC CON MULTIPLEXORES

- Ejemplo (continuación): test-bench de las dos arquitecturas

```
library ieee;
use ieee.std_logic_1164.all;

entity test_sintesis_mux is
end test_sintesis_mux;

architecture test of test_sintesis_mux is
signal a: std_logic := '0';
signal b: std_logic := '0';
signal c: std_logic := '0';
signal d: std_logic := '0';
signal z1, z2: std_logic;
begin
  inst_1: entity work.sintesis_mux(arq_1) port map(a,b,c,d,z1);
  inst_2: entity work.sintesis_mux(arq_2) port map(a,b,c,d,z2);
  d <= not d after 2 ns;
  c <= not c after 4 ns;
  b <= not b after 8 ns;
  a <= not a after 16 ns;
end test;
```

SÍNTESIS DE FC CON MULTIPLEXORES

- Ejemplo (continuación): resultado de la simulación



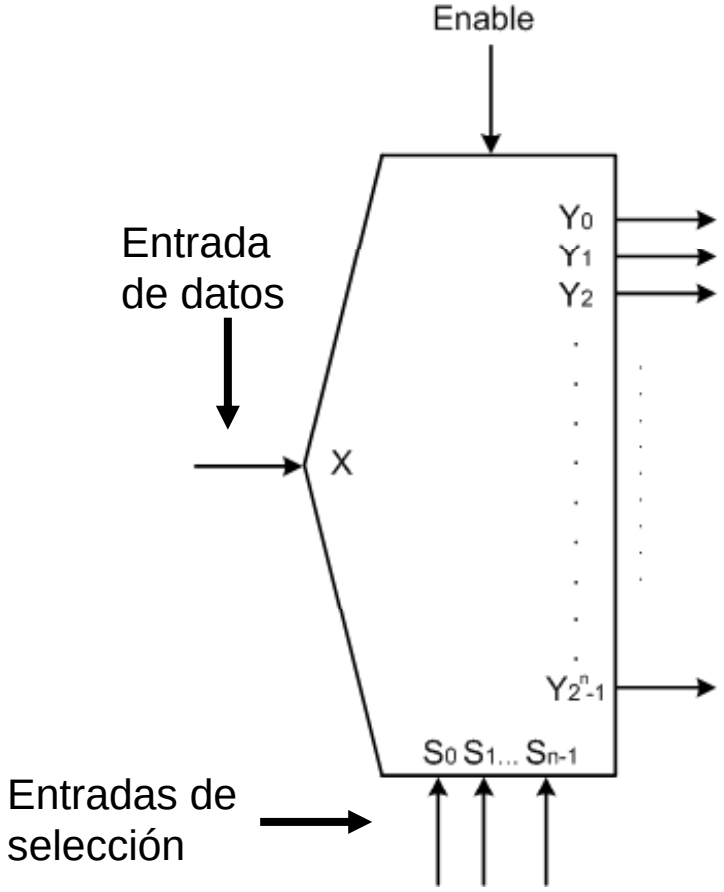
DEMULTIPLEXORES

- Un demultiplexor (o demultiplexor de 1 a 2^n) es un módulo combinacional con 1 entrada y 2^n salidas, además de una entrada de activación y 'n' entradas de control.

- La salida de indice 'i' vale lo mismo que la entrada X, siempre que Enable esté activado y las entradas de selección estén codificando el número decimal 'i'. El resto de salidas valen '0'.

- Puesto en forma de expresión de conmutación:

$$y_i = E \cdot x \cdot m_i(s_{n-1},..., s_0)$$



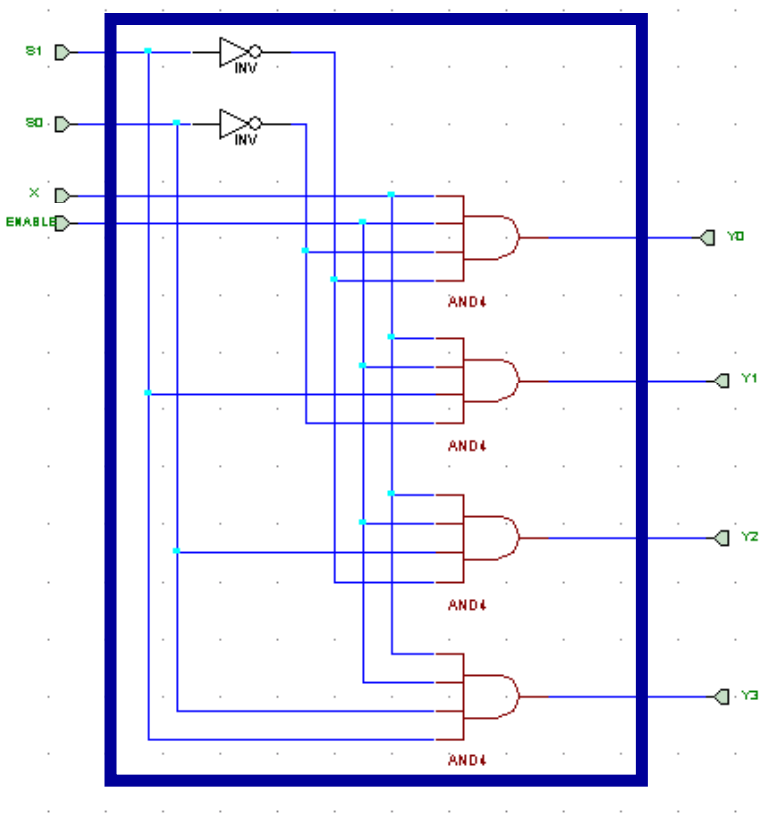
DEMULTIPLEXORES

- Ejemplo: implementar un demultiplexor de 1 a 4.

$$y_i = E \cdot x \cdot m_i(s_{n-1},..., s_0)$$

$$y_0 = E \cdot x \cdot \overline{s}_1 \cdot \overline{s}_0$$
$$y_1 = E \cdot x \cdot \overline{s}_1 \cdot s_0$$
$$y_2 = E \cdot x \cdot s_1 \cdot \overline{s}_0$$
$$y_3 = E \cdot x \cdot s_1 \cdot s_0$$

Demux 4a1



DEMULTIPLEXORES

- Ejemplo (continuación): descripción concurrente en VHDL.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity demux1a4 is
  port(enable, x: in std_logic;
       sel: in std_logic_vector(1 downto 0);
       y: out std_logic_vector(3 downto 0));
end demux1a4;

architecture concurrente of demux1a4 is
begin
  y(0) <= enable and x and (not sel(0)) and (not sel(1));
  y(1) <= enable and x and sel(0) and (not sel(1));
  y(2) <= enable and x and (not sel(0)) and sel(1);
  y(3) <= enable and x and sel(0) and sel(1);
end concurrente;
```

DEMULTIPLEXORES

- Ejemplo (continuación): una posible descripción funcional del mismo demultiplexor.

```
architecture funcional_1 of demux1a4 is
begin
  process(x, sel)
    begin
      y <= "0000";
      if enable = '1' then
        case sel is
          when "00" => y(0) <= x;
          when "01" => y(1) <= x;
          when "10" => y(2) <= x;
          when others => y(3) <= x;
        end case;
      end if;
    end process;
end funcional_1;
```

DEMULTIPLEXORES

- Ejemplo (continuación): otra posible descripción funcional del mismo demultiplexor.

```
architecture funcional_2 of demux1a4 is
begin
  process(x, sel)
  begin
    y <= "0000";
    if enable = '1' then
      y(conv_integer(sel)) <= x;
    end if;
  end process;
end funcional_2;
```

- La función *conv_integer(parametro)* convierte un vector de bits a su valor decimal. Esta función se encuentra en la librería *ieee.std_logic_unsigned*

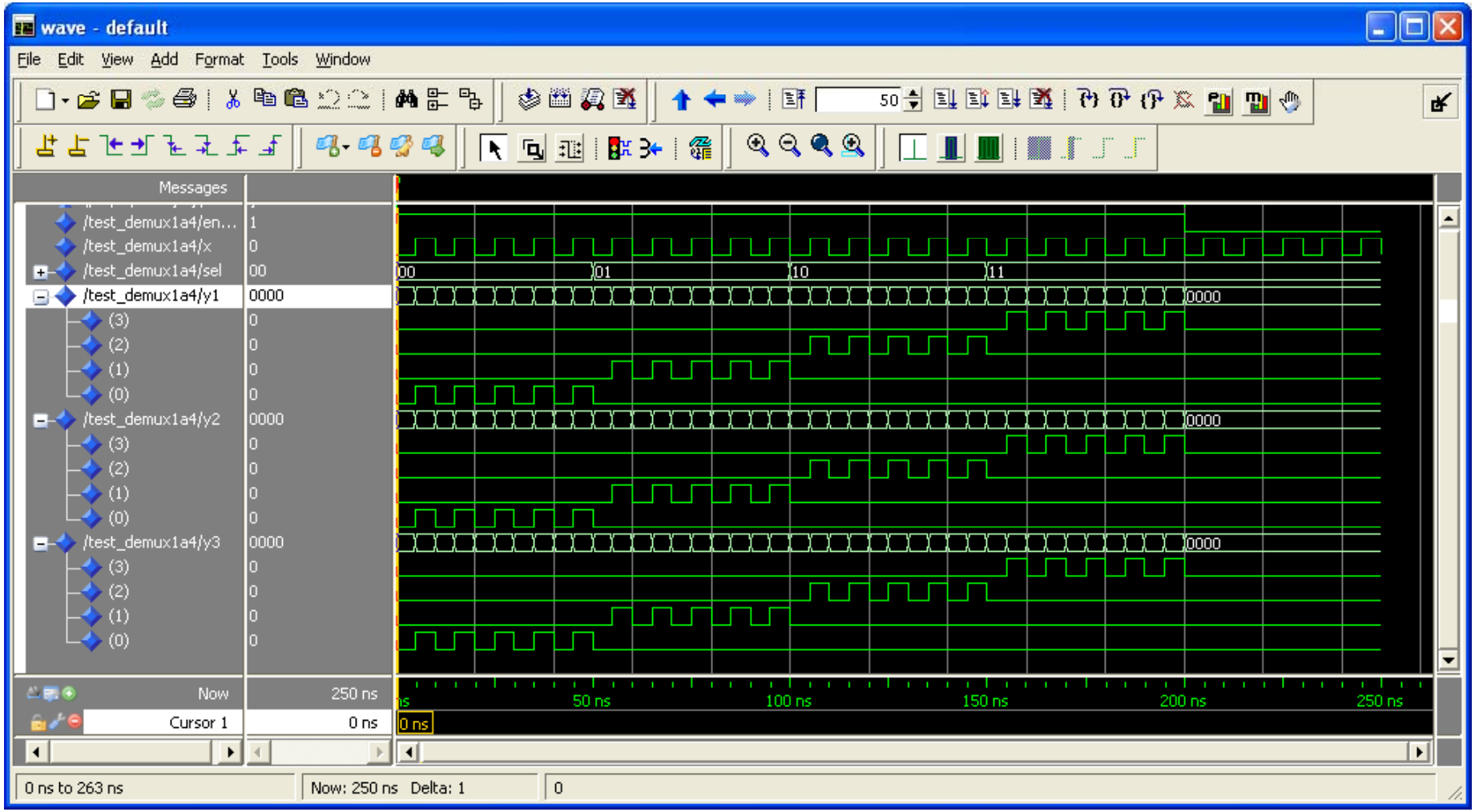
DEMULTIPLEXORES

- Ejemplo (continuación): test-bench para probar las 3 arquitecturas simultáneamente.

```
library ieee;
use ieee.std_logic_1164.all;

entity test_demux1a4 is
end test_demux1a4;

architecture test of test_demux1a4 is
signal enable: std_logic;
signal x: std_logic := '0';
signal sel: std_logic_vector(1 downto 0);
signal y1, y2, y3: std_logic_vector(3 downto 0);
begin
  inst1: entity work.demux1a4(concurrente) port map(enable, x, sel, y1);
  inst2: entity work.demux1a4(funcional_1) port map(enable, x, sel, y2);
  inst3: entity work.demux1a4(funcional_2) port map(enable, x, sel, y3);

  enable <= '1', '0' after 200 ns;
  x <= not x after 5 ns;
  sel <= "00", "01" after 50 ns, "10" after 100 ns, "11" after 150 ns;
end test;
```

DEMULTIPLEXORES

- Ejemplo (continuación): resultado de la simulación.

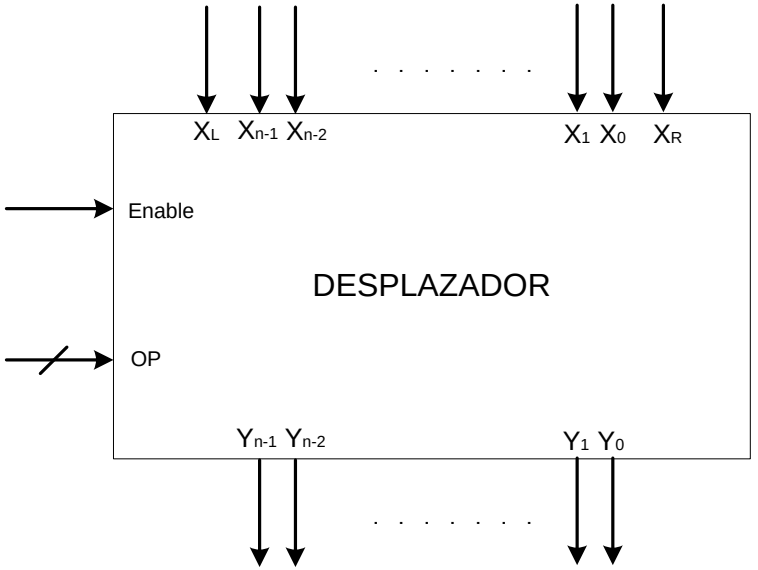


ÍNDICE

DESPLAZADORES

- Un desplazador (*shifter*) es un módulo combinacional con n+2 entradas y n salidas, además de una señal de activación y varias señales de control.
- El desplazador mueve a derecha o izquierda en desplazamientos abiertos o cerrados (rotaciones) en función de las señales de control.

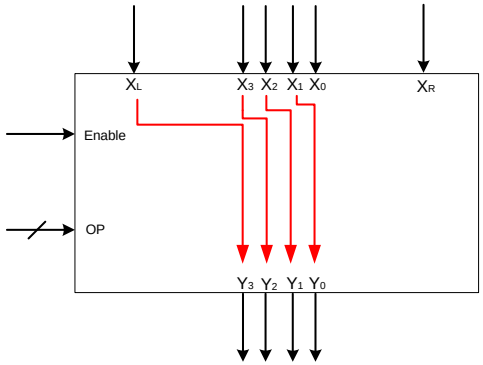
- Aunque se pueden materializar mediante expresiones de conmutación a través de puertas lógicas la construcción habitual suele consistir en un conjunto de multiplexores.



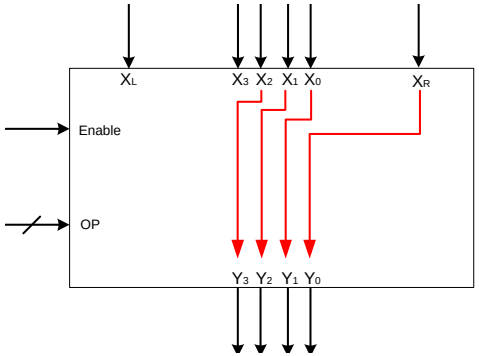
DESPLAZADORES

- Ejemplo: desplazador de 4 bits que realiza las siguientes operaciones:

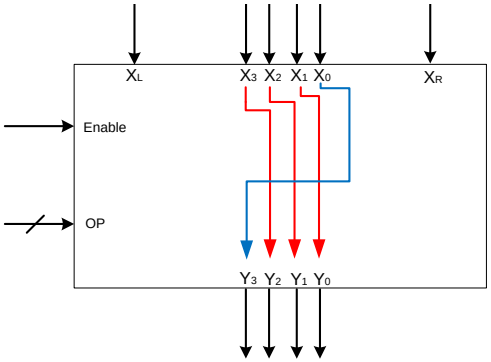
OP = “00” => Desplazamiento a derecha



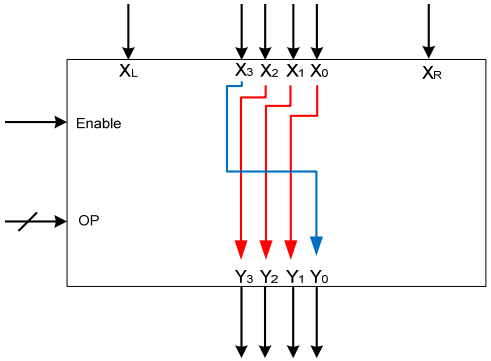
OP = “01” => Desplazamiento a izquierda



OP = “10” => Rotación a derecha

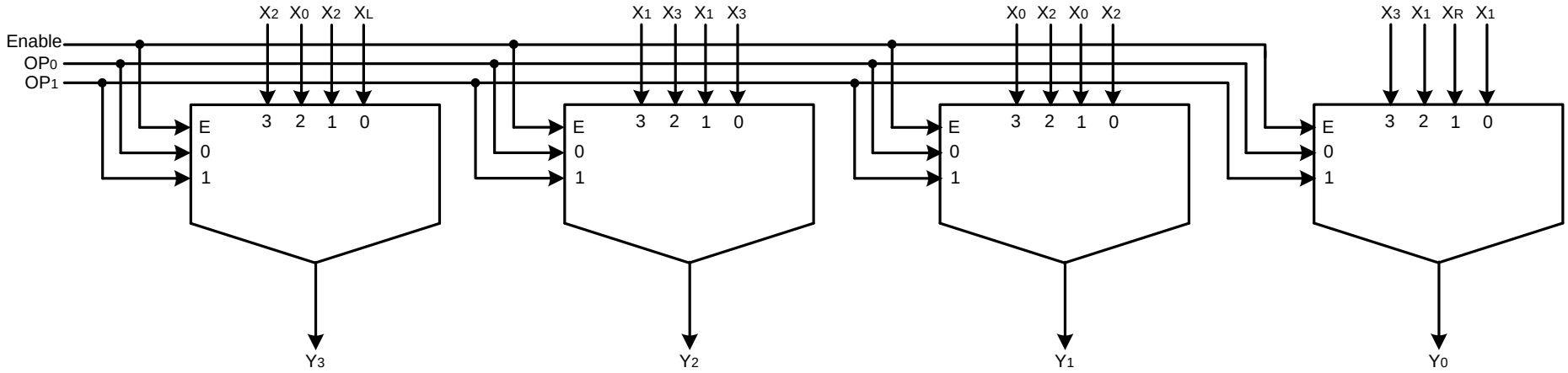


OP = “11” => Rotación a izquierda



DESPLAZADORES

- Ejemplo (continuación): se puede implementar utilizando multiplexores:

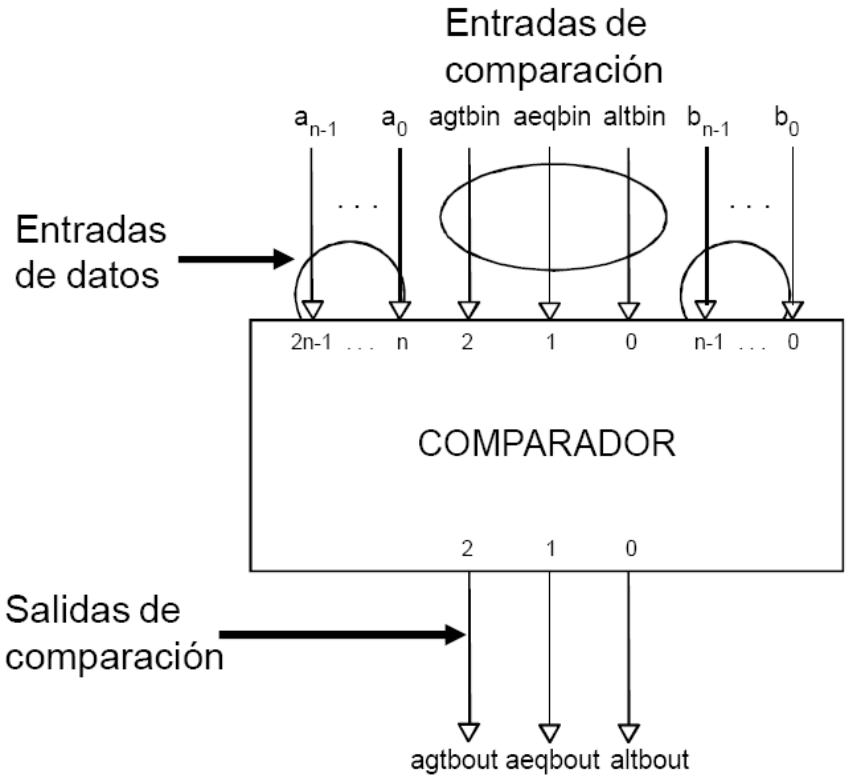


ÍNDICE

COMPARADORES

- Un comparador es un módulo combinacional con 2n entradas de datos (a y b de n bits cada una), 3 entradas para encadenamiento de comparación (de entradas menos a más significativas) y 3 salidas de comparación.

- El comparador determina la relación de magnitud de dos vectores binarios indicando si el primero es mayor (**agtb**), menor (**altb**) o igual (**aeqb**) que el segundo.



$$agtbout = (a>b) + (a=b)\cdot agtbin$$

$$aeqbout = (a=b)\cdot aeqbin$$

$$altbout = (a<b) + (a=b)\cdot altbin$$

COMPARADORES

- Ejemplo: comparador de un bit encadenable.

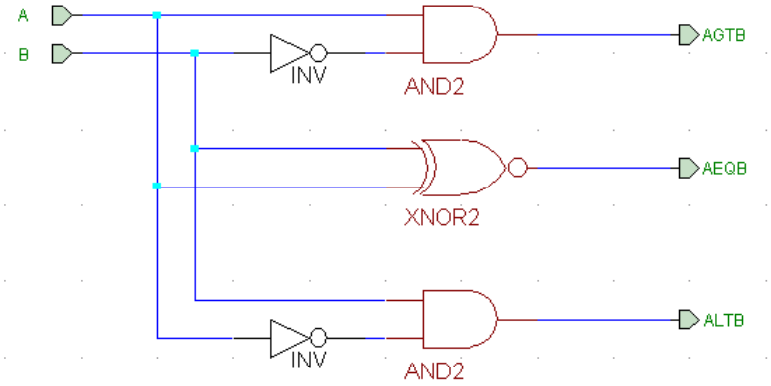
- Partimos primero de un comparador de un bit NO encadenable:

a	b	(a>b)	(a=b)	(a<b)
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

$(a>b) = a \cdot \overline{b}$

$(a=b) = a \text{ xnor } b$

$(a<b) = \overline{a} \cdot b$



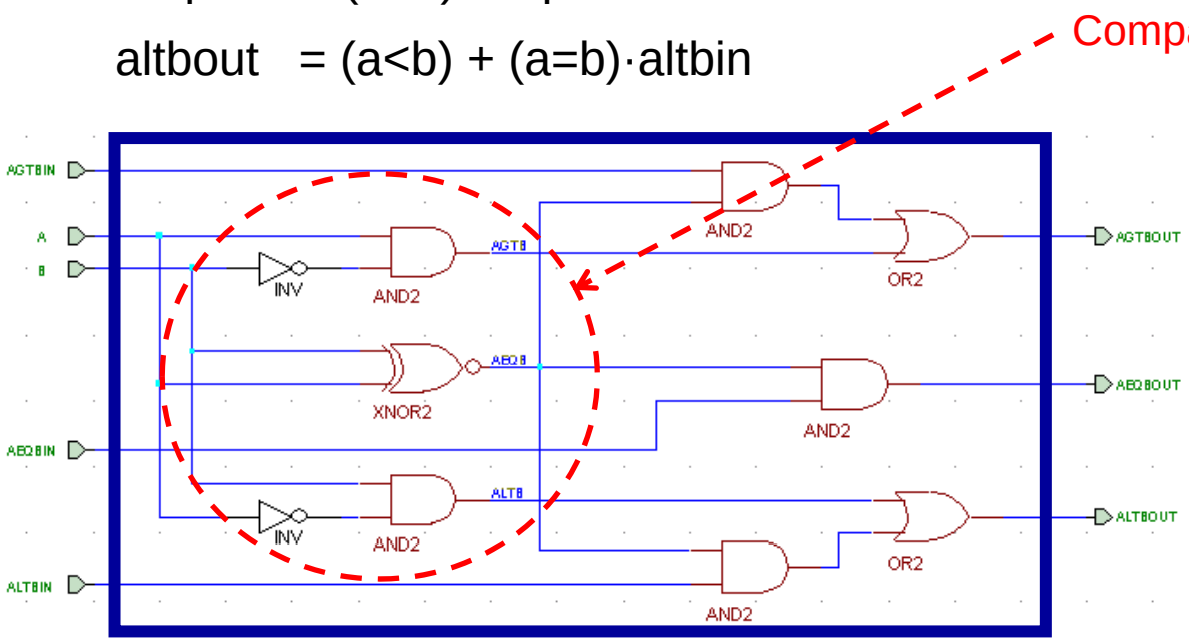
COMPARADORES

- Ejemplo (continuación): a partir del comparador no encadenable añadimos las entradas de comparación se calculan las ecuaciones de las salidas teniendo en cuenta las entradas de comparación.

$$agtbout = (a>b) + (a=b)\cdot agtbin$$

$$aeqbout = (a=b)\cdot aeqbin$$

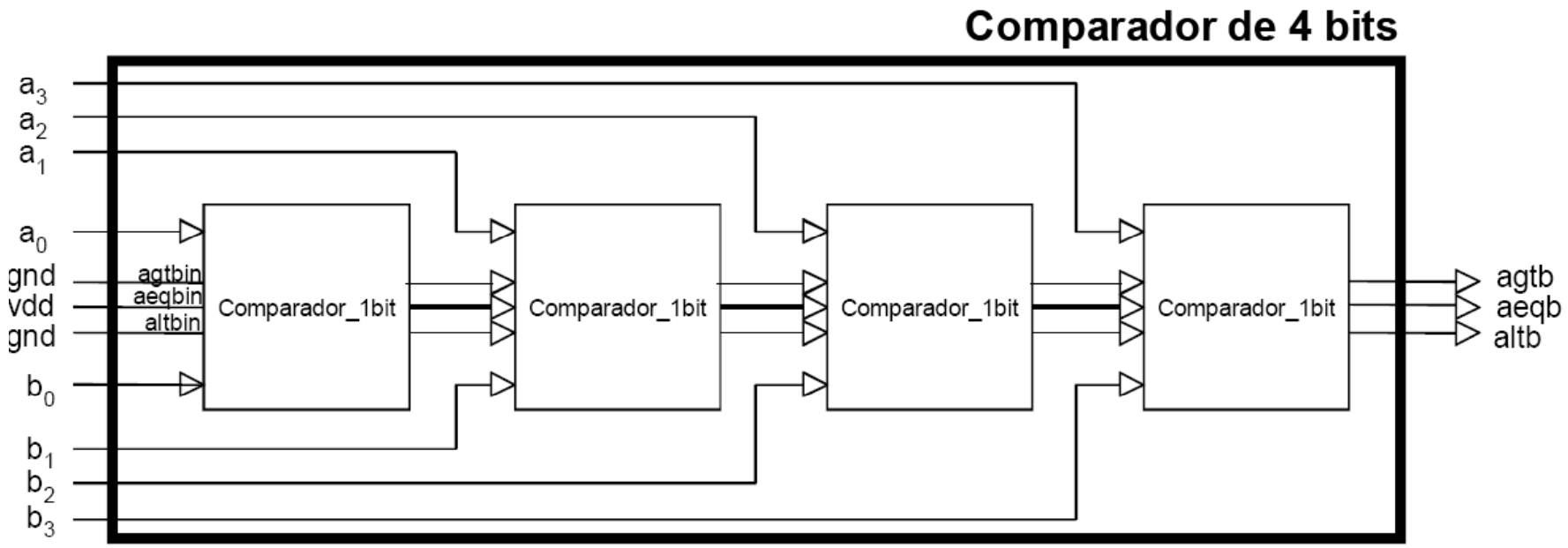
$$altbout = (a<b) + (a=b)\cdot altbin$$



Comparador de 1 bit encadenable

COMPARADORES

- Encadenando varios comparadores se pueden construir otros de mayor tamaño.
- Ejemplo: comparador de 4 bits a partir del comparador encadenable de 1 bit.



ÍNDICE

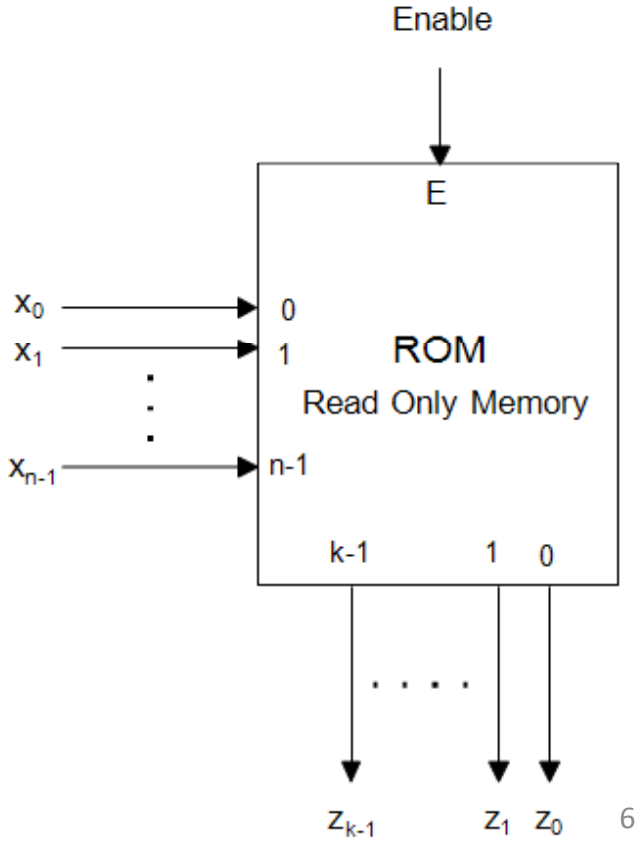
DISPOSITIVOS LÓGICOS PROGRAMABLES

- Bajo la denominación de dispositivos lógicos programables se encuadran un **conjunto de circuitos integrados** formados por cierto número de **puertas lógicas y/o módulos básicos y/o biestables cuyas conexiones pueden ser personalizadas o programadas, bien sea por el fabricante o por el usuario**.
- La gran ventaja de estos dispositivos reside en que los fabricantes pueden realizar grandes tiradas de estos CI lo que abarata sus costes de producción y los usuarios posteriormente pueden personalizar sus diseños en sus propios laboratorios sin grandes inversiones.
- Dentro de ese conjunto de dispositivos haremos mención de las memorias ROM, las PLA y las PAL.
- Las diferencias fundamentales entre sendos tipos de dispositivos estriba en los grados de libertad que disponen cada uno de ellos en cuanto a su capacidad y flexibilidad de programación.

DISPOSITIVOS LÓGICOS PROGRAMABLES: ROM

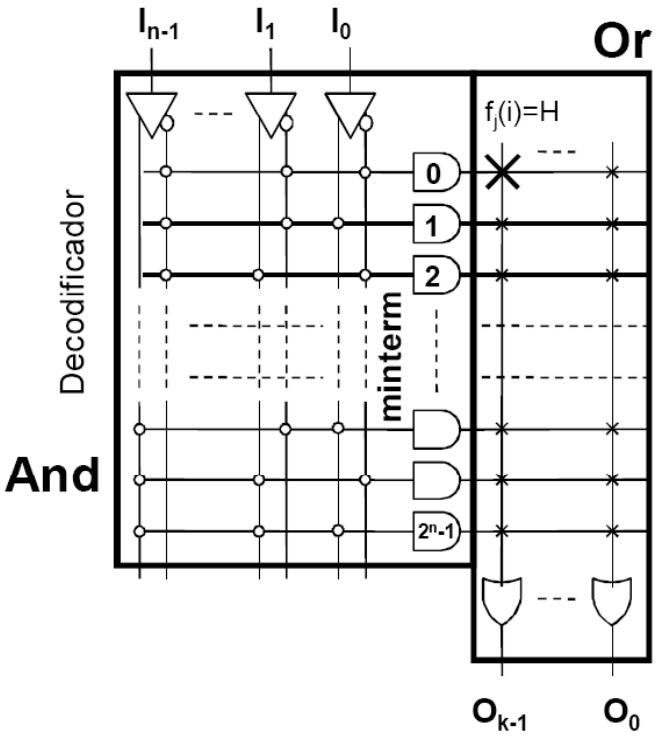
- Una memoria ROM (Read Only Memory - memoria de sólo lectura) es un módulo combinacional con ‘n’ entradas de direcciones y ‘k’ salidas de datos, además de una o varias señales de activación.

- Existen distintos tipos, según se puedan o no programar, borrar, como se realiza el borrado, etc:
  - ROM Read-Only Memory
  - PROM Programmable ROM
  - EPROM Erasable and Programmable ROM
  - EEPROM Electrically erasable PROM



DISPOSITIVOS LÓGICOS PROGRAMABLES: ROM

- Una ROM se compone internamente de dos grupos de puertas: un grupo de puertas And (en realidad incluye también un conjunto de inversores) y un grupo de puertas Or.
- El grupo de puertas And están programadas de antemano y conectadas de forma inalterable, mientras que el grupo de puertas Or son programables por el usuario.
- En general, una ROM con 'n' entradas de direcciones y 'k' salidas de datos consta de un grupo de puertas And ('n' inversores y 2^n puertas And de n entradas) y un grupo de puertas Or ('k' puertas Or de 2^n entradas, tantas como puertas And).



DISPOSITIVOS LÓGICOS PROGRAMABLES: ROM

- Las puertas And están conectadas de tal forma que cada una de ellas implementa uno de los 2^n minterms de las 'n' variables de entrada (es como un decodificador de n a 2^n .
- El grupo de puertas Or es programable, de tal forma que cada una de las puertas Or puede implementar una suma de algunos de los 2^n minterms.
- Cualquier salida de datos de la ROM implementa la siguiente expresión de conmutación:

$$z_j = \sum_{i=0}^{2^n-1} m_i(x_{n-1},\ldots,x_0)\cdot f_j(i)$$

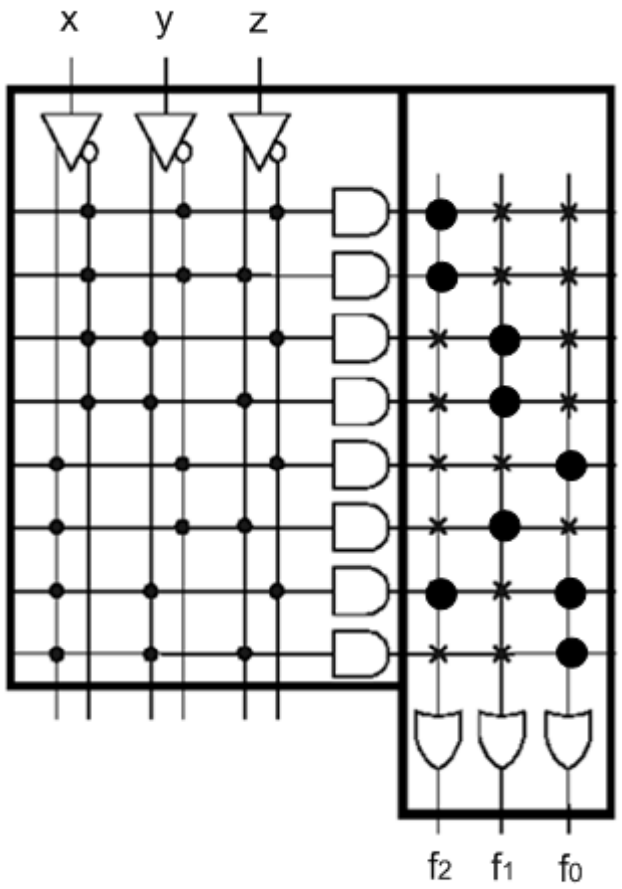
  - $f_j(i)=1$ si existe conexión (fila i, columna j) en el grupo Or
  - $f_j(i)=0$ si no existe dicha conexión

DISPOSITIVOS LÓGICOS PROGRAMABLES: ROM

- Ejemplo: implementar las siguientes funciones de conmutación con una ROM:

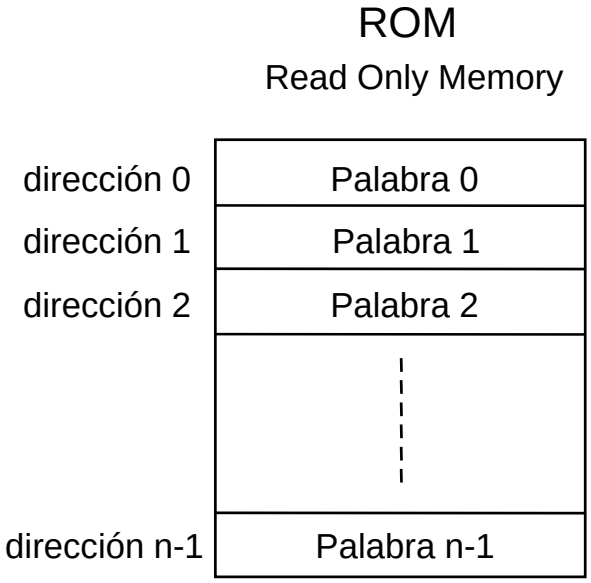
$$f_2(x,y,z) = \sum m(0,1,6)$$
$$f_1(x,y,z) = \sum m(2,3,5)$$
$$f_0(x,y,z) = \sum m(4,6,7)$$

- Como son funciones de 3 variables se necesita una ROM con 3 entradas de direcciones.
- Como hay que implementar 3 funciones se necesita una ROM con 3 salidas de datos.



DISPOSITIVOS LÓGICOS PROGRAMABLES: ROM

- Una ROM se puede ver como una tabla que almacena datos con la siguiente estructura interna abstracta, donde cada dato ocupa una posición de la tabla denominada dirección.

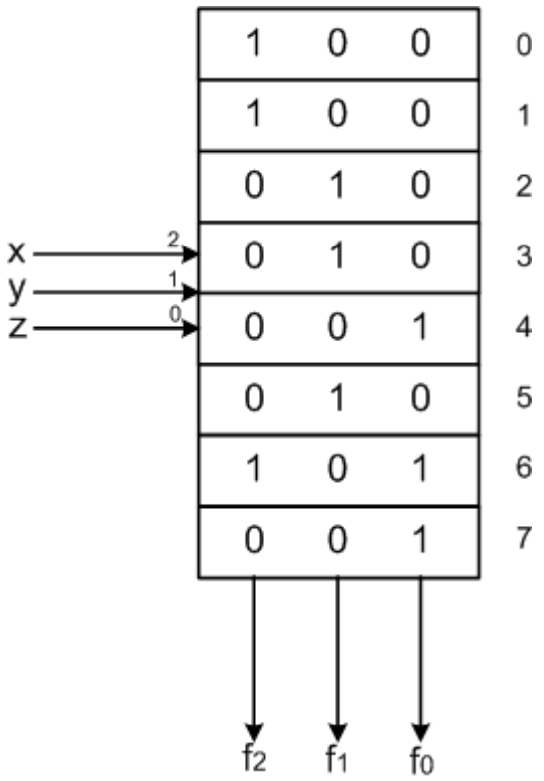


- Como la única parte programable es la Or se suele representar mediante la matriz de conexiones Or con 1 y 0 indicando conexión o no conexión respectivamente, de nuevo materializando directamente la tabla de verdad.

DISPOSITIVOS LÓGICOS PROGRAMABLES: ROM

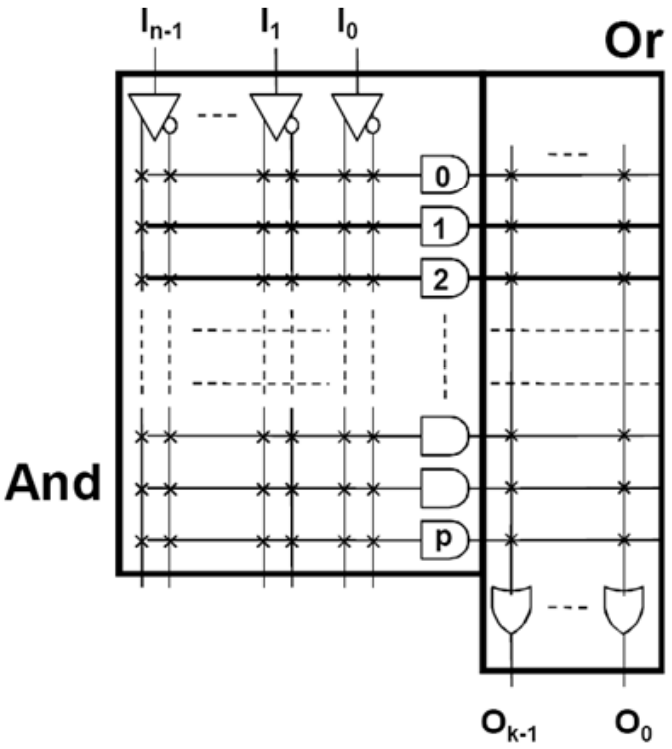
- Ejemplo: las mismas tres funciones que en el ejemplo anterior:

$$f_2(x,y,z) = \sum m(0,1,6)$$
$$f_1(x,y,z) = \sum m(2,3,5)$$
$$f_0(x,y,z) = \sum m(4,6,7)$$



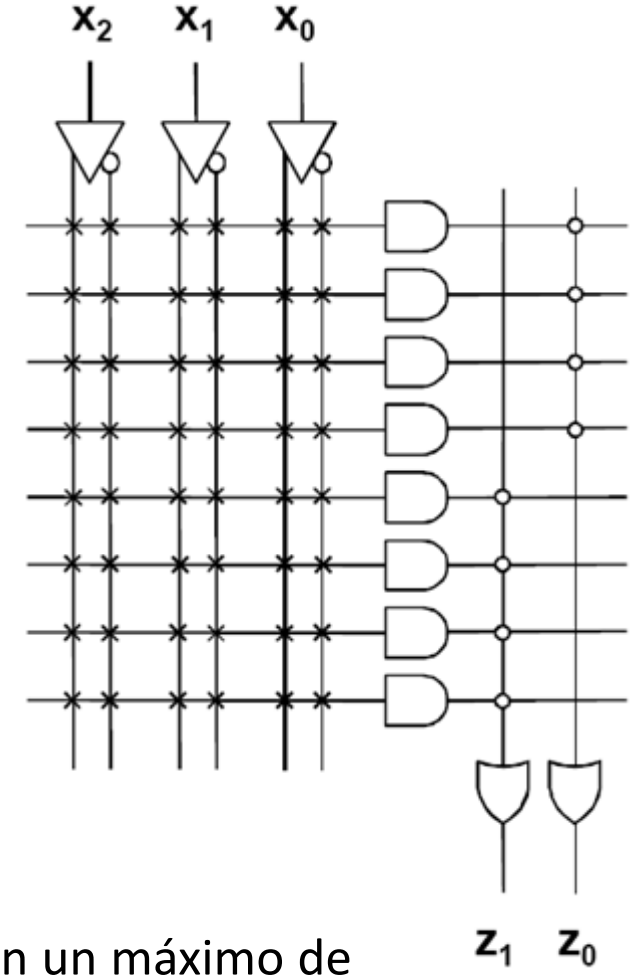
DISPOSITIVOS LÓGICOS PROGRAMABLES: PLA

- Una memoria ROM materializa FCs directamente como suma de minterms ya que el grupo de puertas And está prefijado.
- Cuando una FC sólo utiliza unos pocos minterms o admite una fuerte simplificación utilizar una ROM puede ser un despilfarro.
- Para este tipo de situaciones se utilizan dispositivos PLA (Programmable Logic Array) con conexiones programables tanto en el grupo de puertas And como en el grupo de puertas Or.



DISPOSITIVOS LÓGICOS PROGRAMABLES: PAL

- Las PAL (Programmable Array Logic) son un caso particular de PLA con conexiones Or preprogramadas.
- Una PAL con ‘n’ entradas y ‘k’ salidas presenta un grupo de ‘m’ puertas And de 2^n entradas y un grupo de ‘k’ puertas Or de p entradas, usualmente p=m/k.
- Con estos parámetros una PAL puede materializar ‘k’ funciones de conmutación de ‘n’ variables que se expresen como suma de productos de un máximo de ‘p’ sumandos.
- Ejemplo: representar una PAL con n=3, m=8 y k=2. ¿Qué FCs se pueden materializar?



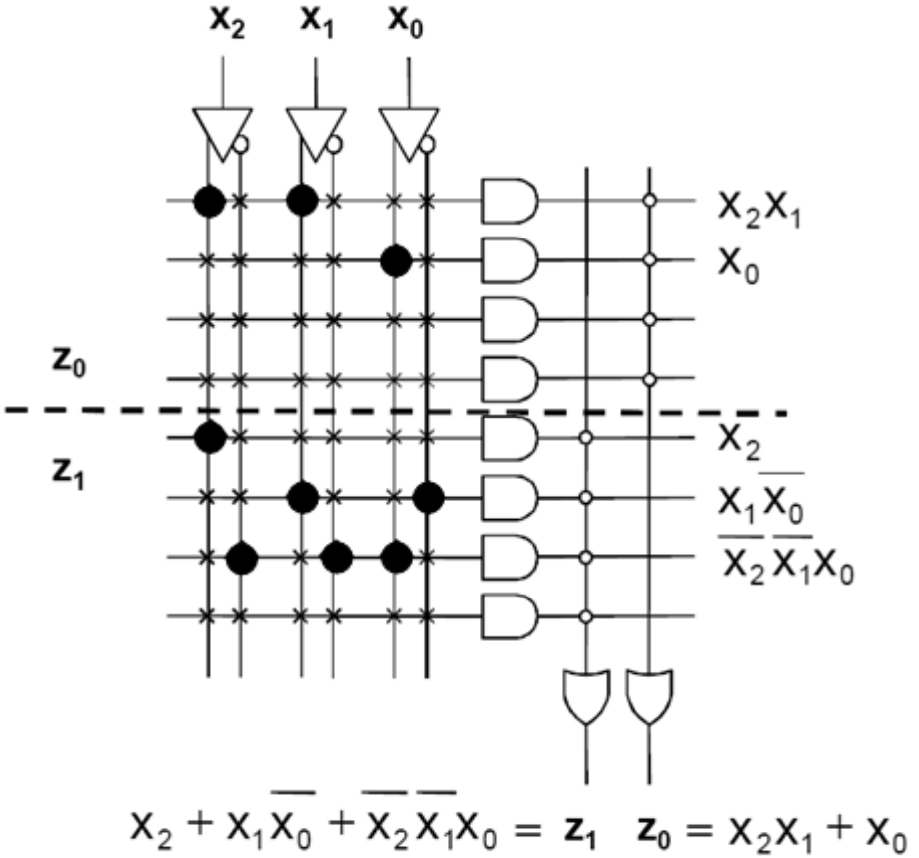
- Se pueden implementar 2 FCs de 3 entradas con un máximo de p=m/k=4 términos producto cada una.

DISPOSITIVOS LÓGICOS PROGRAMABLES: PAL

- Ejemplo: materializar las siguientes funciones de conmutación con una PAL con las características anteriores.

$$Z_1 = X_2 + X_1\overline{X_0} + \overline{X_2}\,\overline{X_1}X_0$$

$$Z_0 = X_2X_1 + X_0$$

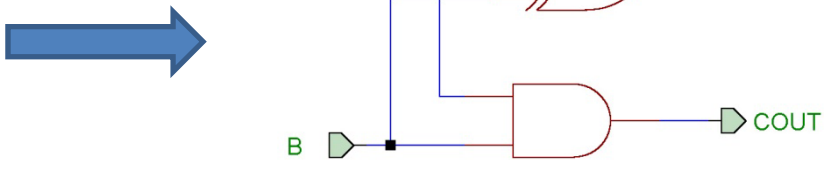


ÍNDICE

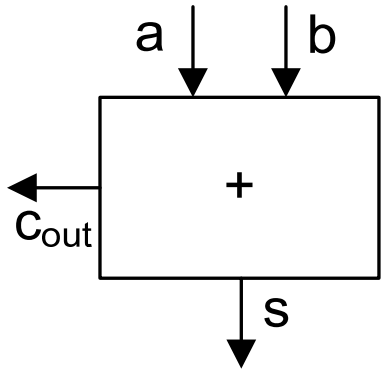
SUMADORES

- El semisumador (*half adder*) es un circuito que suma dos bits de entrada *A* y *B*, y devuelve un bit de resultado *S* y un bit de acarreo c_{out}.

a	b	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



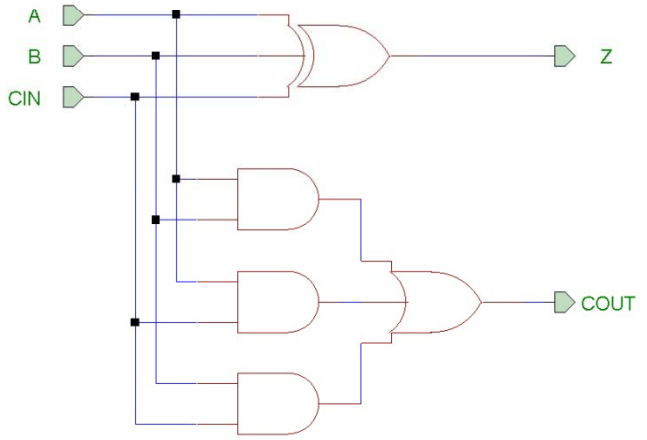
- Se suele representar con este símbolo:



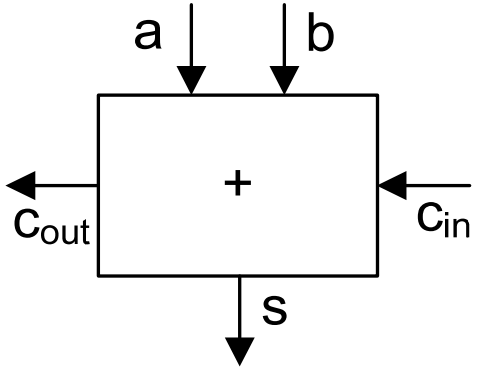
SUMADORES

- El sumador completo (*full adder*) es un circuito que suma dos bits de entrada *A* y *B* más un acarreo de entrada C_{in} y devuelve un bit de resultado *S* y un bit de acarreo C_{out}.

a	b	Cin	S	Cout
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



- Se suele representar con este símbolo:



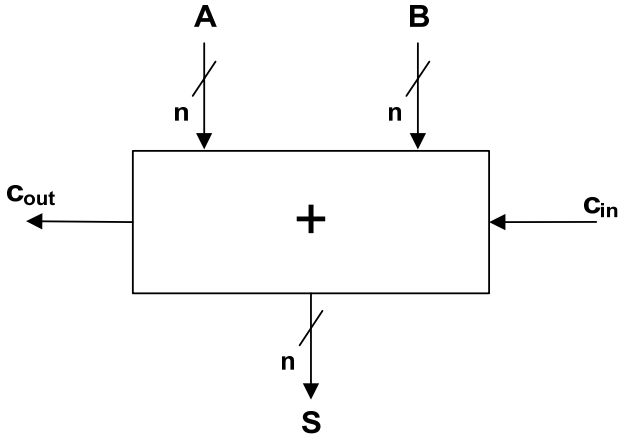
SUMADORES

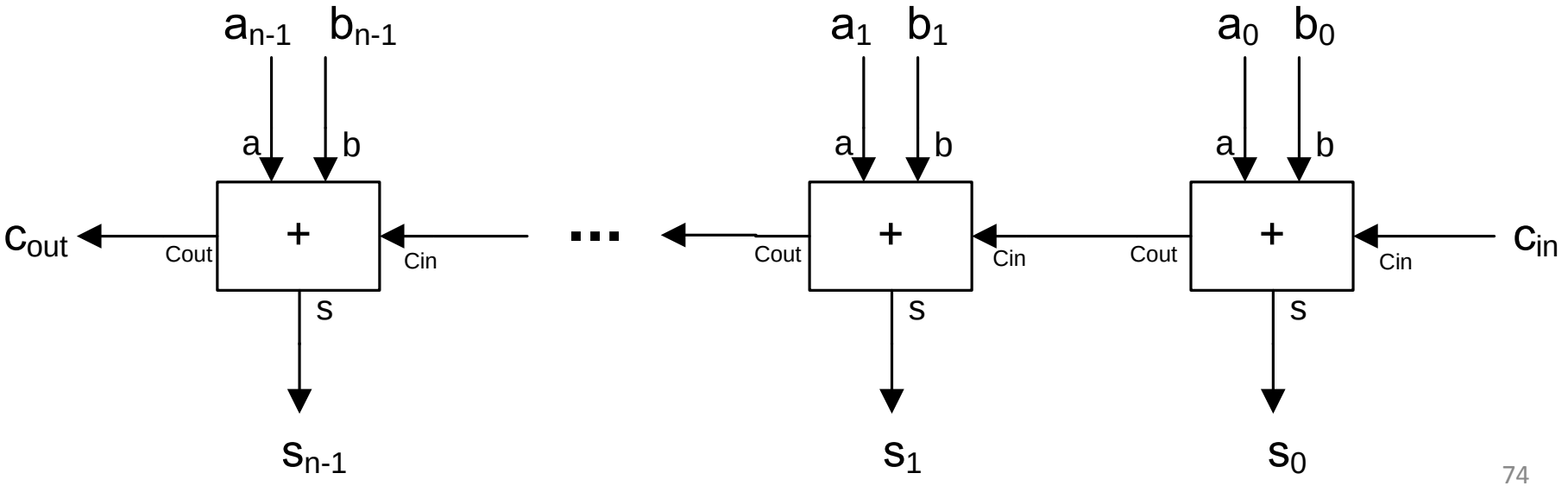
- Ejemplo: sumador completo de 1 bit en VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity add1 is
  port(a,b,cin: in std_logic;
       s, cout: out std_logic);
end add1;

architecture concurrente of add1 is
begin
  cout <= (a and b) or (a and cin) or (b and cin);
  s <= a xor b xor cin;
end concurrente;
```

SUMADORES

- Sumador de ‘n’ bits con propagación de acarreo:
  - Se construye conectando en cascada varios sumadores completos de 1 bit.





SUMADORES

- Ejemplo: sumador de 4 bits en VHDL.

```
library ieee;
use ieee.std_logic_1164.all;

entity add4 is
  port(a,b: in std_logic_vector(3 downto 0);
       cin: in std_logic;
       s:   out std_logic_vector(3 downto 0);
       cout:out std_logic);
end add4;

architecture estructural of add4 is
signal c: std_logic_vector(4 downto 0);
begin
  c(0) <= cin;
  sum0: entity work.add1 port map(a(0), b(0), c(0), s(0), c(1));
  sum1: entity work.add1 port map(a(1), b(1), c(1), s(1), c(2));
  sum2: entity work.add1 port map(a(2), b(2), c(2), s(2), c(3));
  sum3: entity work.add1 port map(a(3), b(3), c(3), s(3), c(4));
  cout <= c(4);
end estructural;
```

SUMADORES

- Ejemplo (continuación): cuando la estructura interna de un circuito es repetitiva, como en el sumador de 4 bits, se puede utilizar una construcción del lenguaje que permite crear varias instancias iguales (***for generate***).

```
architecture estructural_2 of add4 is
signal c: std_logic_vector(4 downto 0);
begin
  c(0) <= cin;
  sum4: for i in 0 to 3 generate
        sum: entity work.add1 port map(a(i), b(i), c(i), s(i), c(i+1));
      end generate;
  cout <= c(4);
end estructural_2;
```

SUMADORES

- El lenguaje VHDL permite definir una entidad que tenga parámetros. Estos parámetros reciben el nombre de *generics*, y sirven por ejemplo para describir circuitos que no tienen un tamaño fijo si no que se puede fijar el tamaño a la hora de crear una instancia de ese circuito.
- Vamos a ver un ejemplo de uso de *generics* con un sumador de 'n' bits:

```
library ieee;
use ieee.std_logic_1164.all;

entity addn is
  generic(n: integer);
  port(a,b: in std_logic_vector(n-1 downto 0);
       cin: in std_logic;
       s:   out std_logic_vector(n-1 downto 0);
       cout:out std_logic);
end addn;
```

SUMADORES

- Ejemplo (continuación):

```
architecture estructural of addn is
signal c: std_logic_vector(n downto 0);
begin
  c(0) <= cin;
sum4: for i in 0 to n-1 generate
        sum: entity work.add1 port map(a(i), b(i), c(i), s(i), c(i+1));
      end generate;
  cout <= c(n);
end estructural;
```

SUMADORES

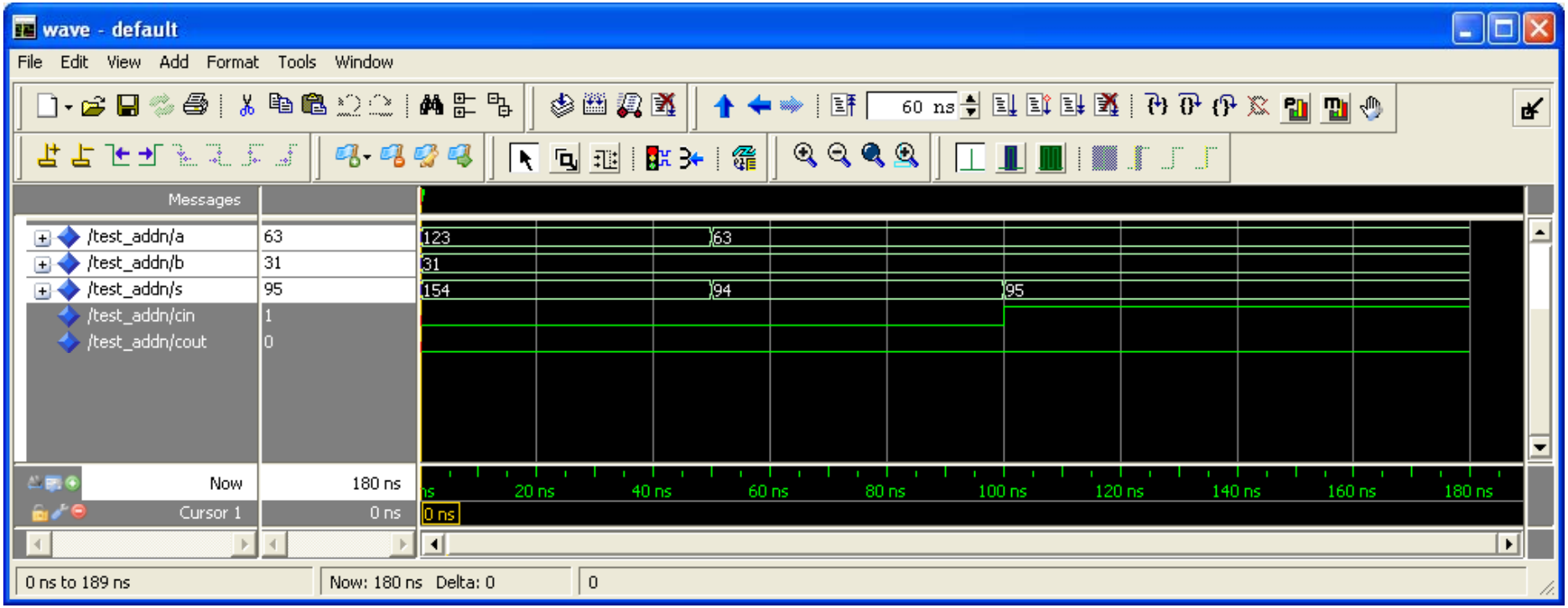
- Ejemplo: es a la hora de utilizar el sumador de ‘n’ bits en otro circuito, o en un test-bench, cuando se le dará valor al parámetro ‘n’.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity test_addn is
end test_addn;

architecture test of test_addn is
signal a,b,s: std_logic_vector(7 downto 0);
signal cin, cout: std_logic;
begin
  cin <= '0', '1' after 100 ns;
  a <= conv_std_logic_vector(123,8), conv_std_logic_vector(63,8) after 50 ns;
  b <= conv_std_logic_vector(31,8);
  inst_1: entity work.addn generic map(8) port map(a,b,cin,s,cout);
end test;
```

- La función *conv_std_logic_vector(valor, n)* convierte el número ‘*valor*’ a un std_logic_vector de ‘*n*’ bits. Esta función se encuentra en la librería *ieee.std_logic_arith*

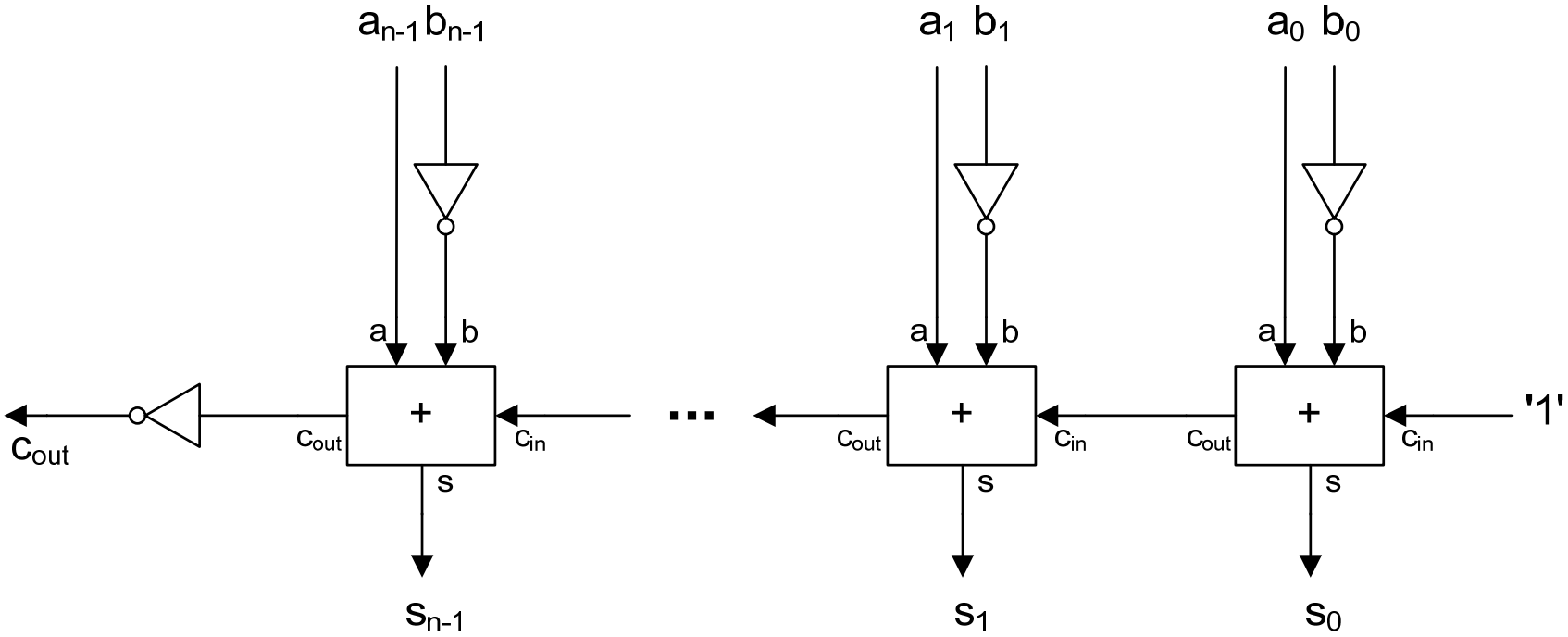
SUMADORES

- Ejemplo (continuación): resultado de la simulación.



RESTADORES

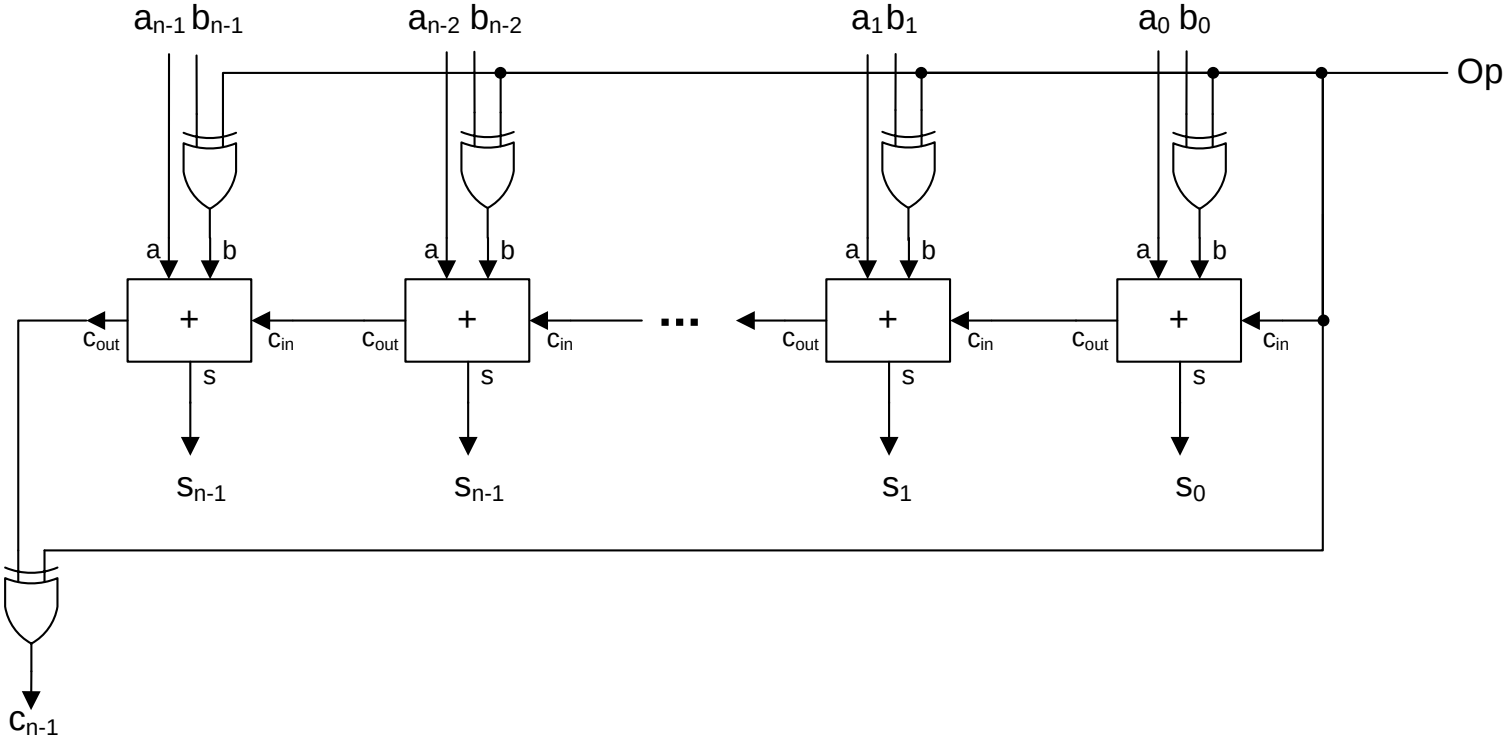
- Recordatorio: para restar dos números en binario se hace la suma del minuendo con el complemento a 2 del sustraendo. El complemento a 2 es el complemento a 1 más ‘1’. En la resta binaria hay que invertir el acarreo final.



SUMADORES RESTADORES

- Se pueden combinar el sumador y el restador en un único circuito que realice la operación de suma o de resta en función de una señal de control. Recordar que: $X \oplus 0 = X$

$$X \oplus 1 = \overline{X}$$



ÍNDICE

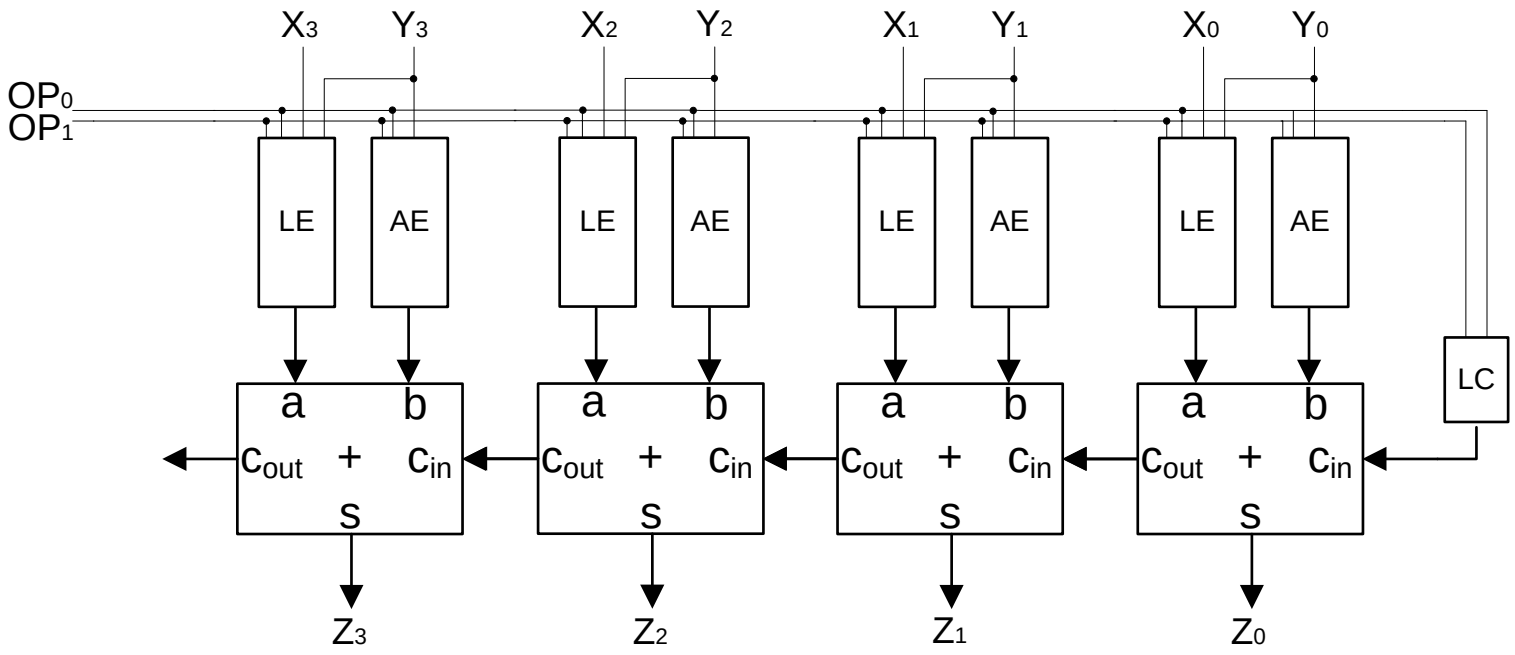
UNIDAD ARITMÉTICO LÓGICA

- Una unidad aritmético lógica (ALU) es un circuito combinacional que permite hacer distintas operaciones aritméticas (sumas, restas, desplazamientos) y lógicas (AND, OR, NOT, XOR, etc) entre dos operandos, en función de unas señales de control.
- Suelen estar implementadas con sumadores-restadores, multiplexores y lógica adicional.
- Ejemplo de diseño: ALU con dos entradas de datos de 4 bits, que realiza las siguientes operaciones en función de una entrada de control de 2 bits:

OP_1	OP_0	OPERACIÓN
0	0	X + Y
0	1	X - Y
1	0	X AND Y
1	1	X OR Y

UNIDAD ARITMÉTICO LÓGICA

- Ejemplo (continuación):



UNIDAD ARITMÉTICO LÓGICA

- Con la estructura propuesta, los bloques LE y AE se encargan de proporcionar a las entradas A y B de los sumadores los datos necesarios para que se realice la operación deseada. El bloque LC se encarga de proporcionar el acarreo de entrada a la cadena de sumadores.
  - Si OP1 y OP0 = “00”
    - $A_i = X_i$, $B_i = Y_i$, LC = 0, y por tanto los sumadores harán la operación X + Y + 0.
  - Si OP1 y OP0 = “01”
    - $A_i = X_i$, $B_i = \text{‘}Y_i$, LC = 1, y por tanto los sumadores harán la operación X + ‘Y + 1, es decir X – Y.
  - Si OP1 y OP0 = “10”
    - $A_i = X_i$ AND Y_i, B_i = 0, LC = 0, y por tanto los sumadores harán la operación (X AND Y) + 0 + 0, es decir X AND Y.
  - Si OP1 y OP0 = “11”
    - $A_i = X_i$ OR Y_i, B_i = 0, LC = 0, y por tanto los sumadores harán la operación (X OR Y) + 0 + 0, es decir X OR Y.

UNIDAD ARITMÉTICO LÓGICA

LE

OP1 OP0 \ x y	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	1	1	1
10	0	0	1	0

$$a_i = OP_1 \cdot OP_0 \cdot y_i + x_i \cdot y_i + \overline{OP_1} \cdot x_i + OP_0 \cdot x_i$$

AE

yi \ OP1 OP0	00	01	11	10
00	0	1	0	0
01	1	0	0	0

$$b_i = \overline{OP_1} \cdot \overline{OP_0} \cdot y_i + \overline{OP_1} \cdot OP_0 \cdot \overline{y_i}$$

LC => vale ‘1’ cuando OP = “01” => $LC = \overline{OP_1} \cdot OP_0$