Bloques combinacionales estándar

Norberto Malpica

norberto.malpica@urjc.es

Ingeniería de Tecnologías Industriales

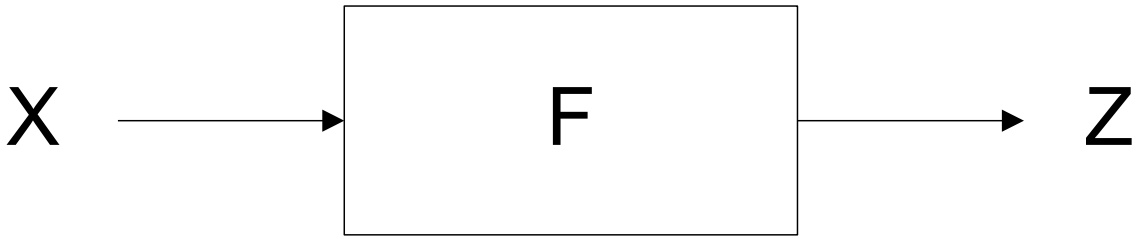
Contenido

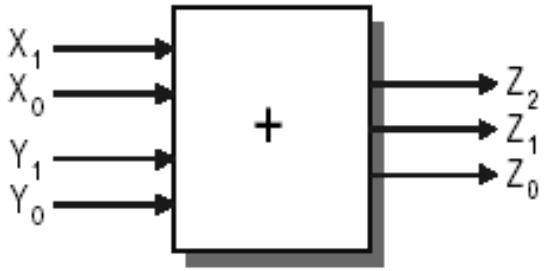
1. Introducción
2. Codificadores
3. Decodificadores
4. Multiplexores
5. Demultiplexores
6. Desplazadores
7. Dispositivos Lógicos Programables
   - 7.1 Memorias ROM.
   - 7.2 PLA y PAL.
8. Comparadores Binarios
9. Sumadores Binarios.
10. Restadores Binarios.
11. Unidad aritmético lógica combinacional (UAL, ALU).

1. Introducción a los circuitos combinacionales

En los circuitos combinacionales **la salida Z** en un determinado instante de tiempo t_i **sólo depende de la entrada X** en ese mismo instante de tiempo t_i , es decir que **no tienen capacidad de memoria**

$Z(t) = F(X(t))$ $Z = F(X)$

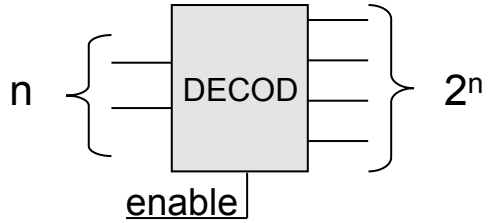


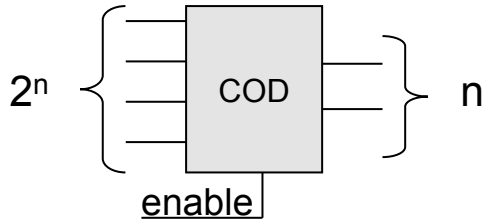


Ejemplo: Sumador

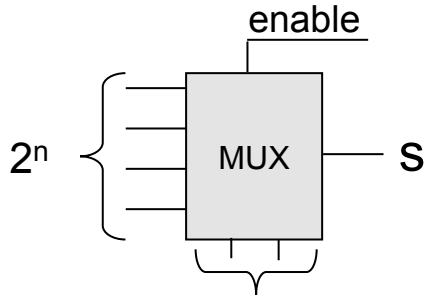
Introducción a los circuitos combinacionales

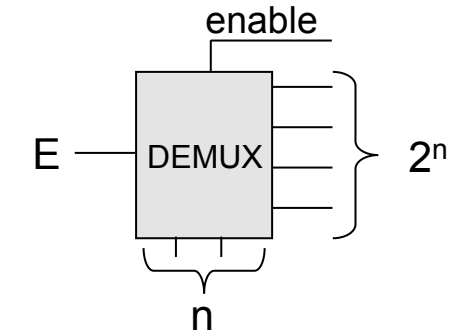
- Decodificadores y Codificadores
  - Decodificador: Se activa la salida correspondiente al número binario codificado en la entrada.
  - Codificador: Se codifica en binario sobre la salida el número de la entrada que esté activa.





- Multiplexores y Demultiplexores
  - Multiplexor: La salida corresponde a la entrada codificada por las señales de control
  - Demultiplexor: El valor de la entrada sale por la salida codificada por las señales de control

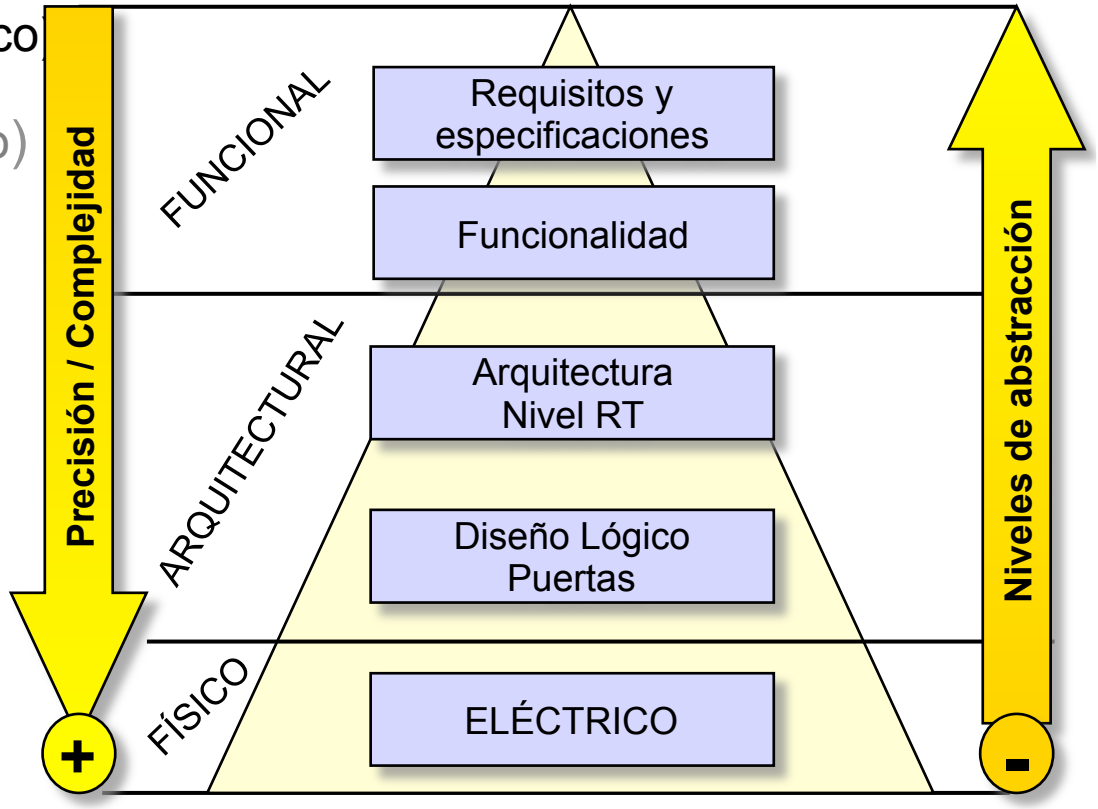




Introducción a los circuitos combinacionales

Niveles de descripción de un circuito digital

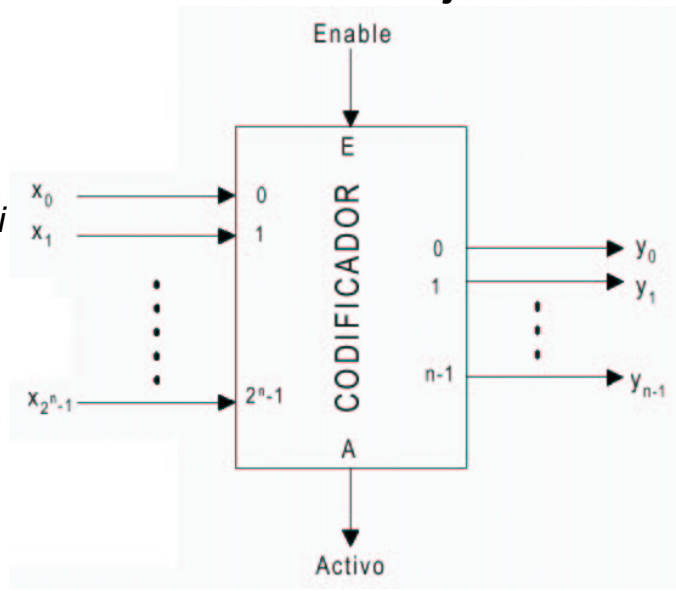
- Nivel algorítmico o comportamental: describe la función
- Nivel RTL (álgebra de señales, tabla de verdad)
- Nivel estructural (lógico o esquemático)
- Nivel conmutador (circuital y eléctrico)



2. Codificadores

El codificador identifica qué entrada de las 2^n está activa y genera como salida su representación binaria, siempre y cuando el módulo esté activo.

Un codificador tiene **2^n entradas de datos x_i** y **n salidas de datos y_j** y **una salida A**



- Si *E* está inactivo, todas las salidas inactivas.
- Si *E* está activo y todas las entradas de datos x_i están inactivas, todas las salidas (y_j y *A*) permanecen inactivas.
- Si se activa la entrada de datos x_i y *E* está activo:
  - Las salidas y_j componen el número *i* codificado en binario.
  - Se activa la salida *A*.

Expresión de conmutación: $y_i = E \cdot \sum x_j$

Codificadores

El **comportamiento** del codificador sin prioridad es:

$$Y = \sum_{j=0}^{n-1} y_j \cdot 2^j$$

$$Y = \begin{cases} i & \text{si } x_i = H \text{ y } E = H \text{ y } x_k = L \; \forall k \neq i \\ \text{indefinido} & \text{si } \exists x_i, x_j / x_i = x_j = H \text{ con } i \neq j \\ L & \text{resto de casos} \end{cases}$$

La salida de actividad A se define como: $A = \begin{cases} H & \text{si algún } x_i = H \text{ y } E = H \\ L & \text{resto de casos} \end{cases}$

Expresión de conmutación: $y_i = E \cdot \sum x_j$

Codificadores

Ejemplo: diseño de un codificador sin prioridad de 8 a 3.

Comercial: 74148

E	X_{activa}	Y_2	Y_1	Y_0	A
L	x	L	L	L	L
H	X_0	L	L	L	H
H	X_1	L	L	H	H
H	X_2	L	H	L	H
H	X_3	L	H	H	H
H	X_4	H	L	L	H
H	X_5	H	L	H	H
H	X_6	H	H	L	H
H	X_7	H	H	H	H

Funciones lógicas

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

$$Y_0 = E \cdot (x_1+x_3+x_5+x_7)$$

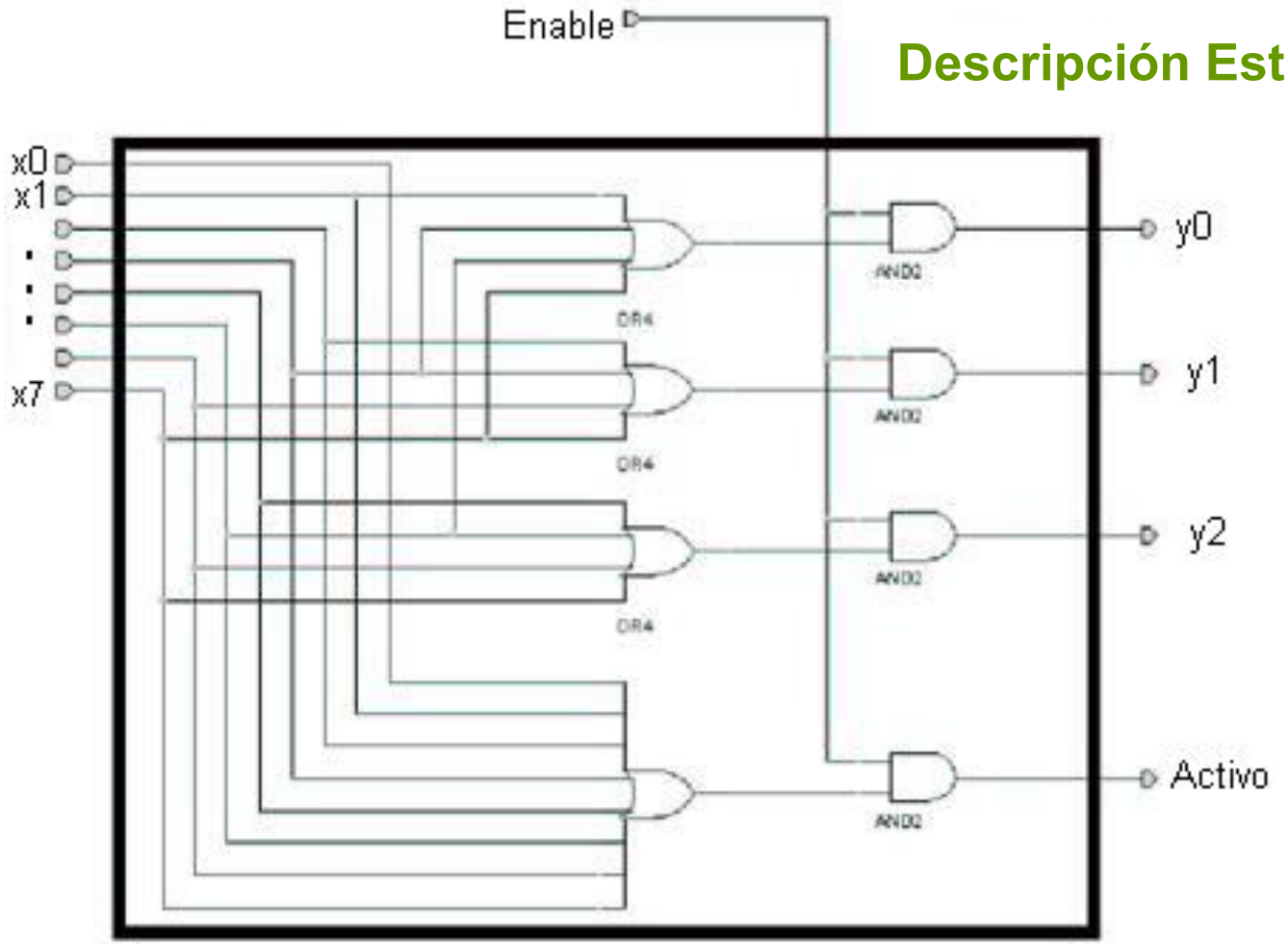
$$Y_1 = E \cdot (x_2+x_3+x_6+x_7)$$

$$Y_2 = E \cdot (x_4+x_5+x_6+x_7)$$

Codificadores

Ejemplo: diseño de un codificador sin prioridad de 8 a 3.

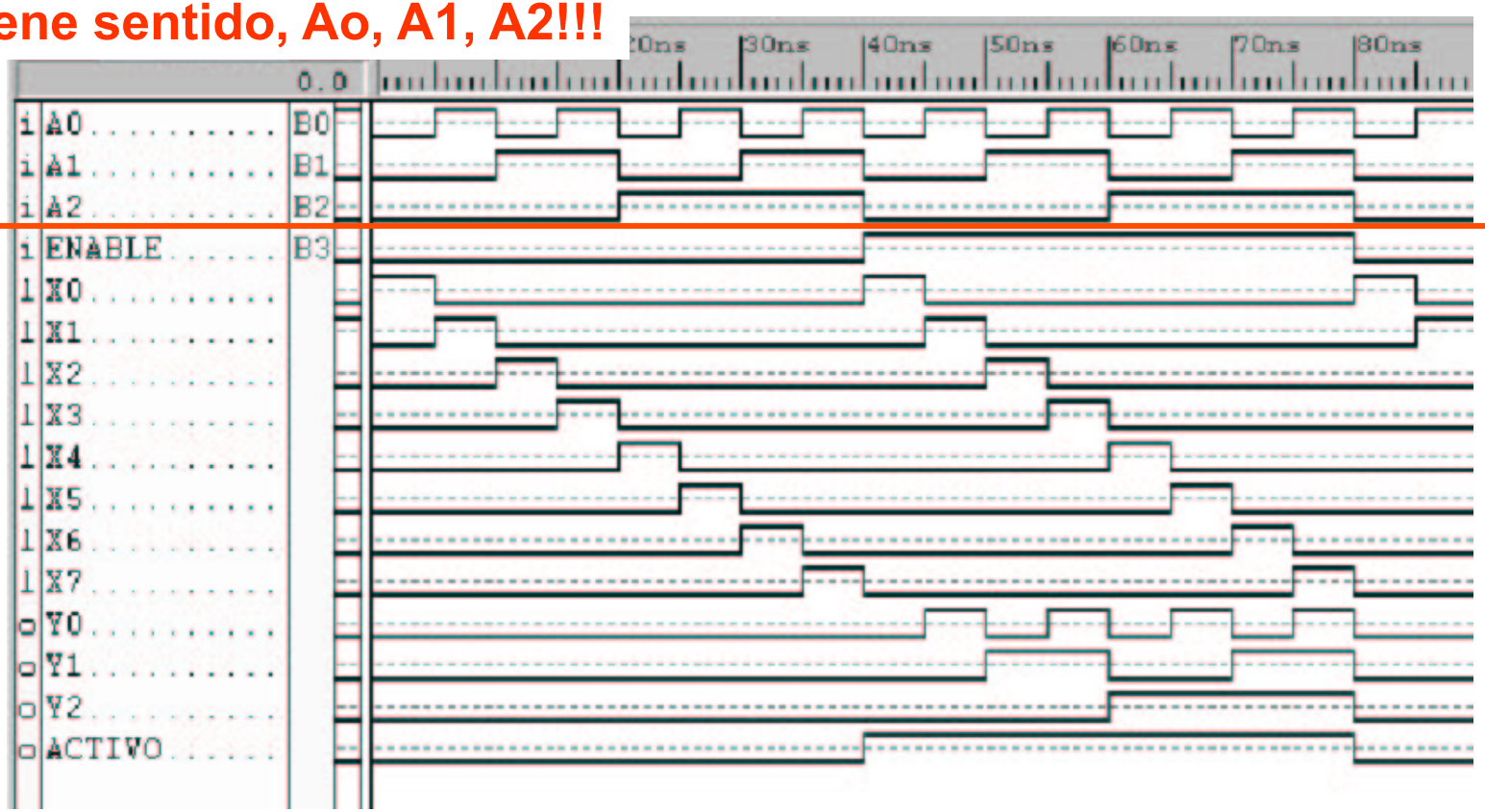
Descripción Estructural



Codificadores

Ejemplo: diseño de un codificador sin prioridad de 8 a 3. **Simulación**

No tiene sentido, Ao, A1, A2!!!



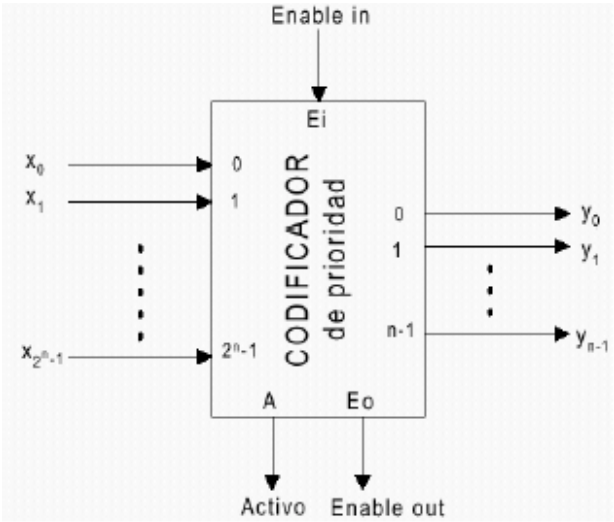
Codificadores con prioridad

¿Qué pasa si hay de más de una entrada activa?

¿Qué aparecerá en la salida?

Codificadores con Prioridad:

Las salidas y_j codifican en binario el número correspondiente a la **entrada activa con mayor peso**



Los codificadores pueden encadenarse para formar codificadores con mayor número de bits.

Eout se activa cuando ***Ein*** está activo y no hay ninguna entrada de datos activa.

Si la salida E_{out} se conecta a la entrada E_{in} de otro codificador permite su encadenamiento.

Codificadores con prioridad

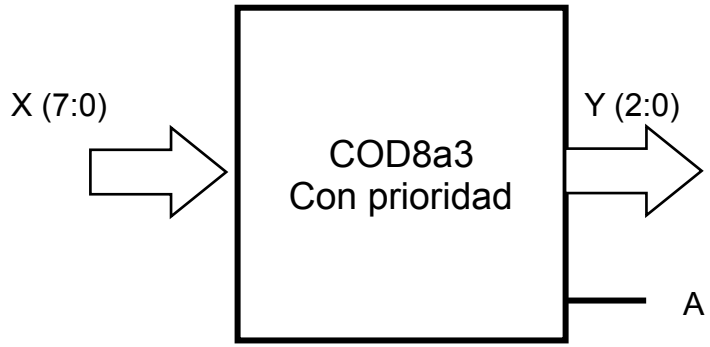
El **comportamiento** del codificador con prioridad es:

$$Y = \begin{cases} i & \text{si } x_i = H \; y \; E_{in} = H \text{ y } x_k = L \; \forall k > i \\ L & \text{resto de los casos} \end{cases}$$

$$A = \begin{cases} H & \text{si algún } x_i = H \; y \; E = H \\ L & \text{resto de los casos} \end{cases}$$

$$E_{out} = \begin{cases} H & \text{si } x_i = L(\forall i) \text{ y } E = H \\ L & \text{resto de los casos} \end{cases}$$

Codificador de 8 a 3 con prioridad




```
entity COD8a3_prior is
 port( X     : in  std_logic_vector (7 downto 0);
       Y  : out std_logic_vector (2 downto 0);
       A      : out std_logic);
end COD8a3_prior;
```

```
architecture comportamental1  of COD8a3_prior is
begin
Y <= "111" when X(7) ='1' else
                "110" when X(6) ='1' else
                "101" when X (5) ='1' else
                "100" when X (4) ='1' else
                "011" when X (3) ='1' else
                "010" when X (2) ='1' else
                "001" when X(1) ='1' else
                "000";
A <= '0' when DATAIN = "00000000" else '1';
end comportamental1;
```

Codificadores con prioridad

Materialización del codificador de prioridad:

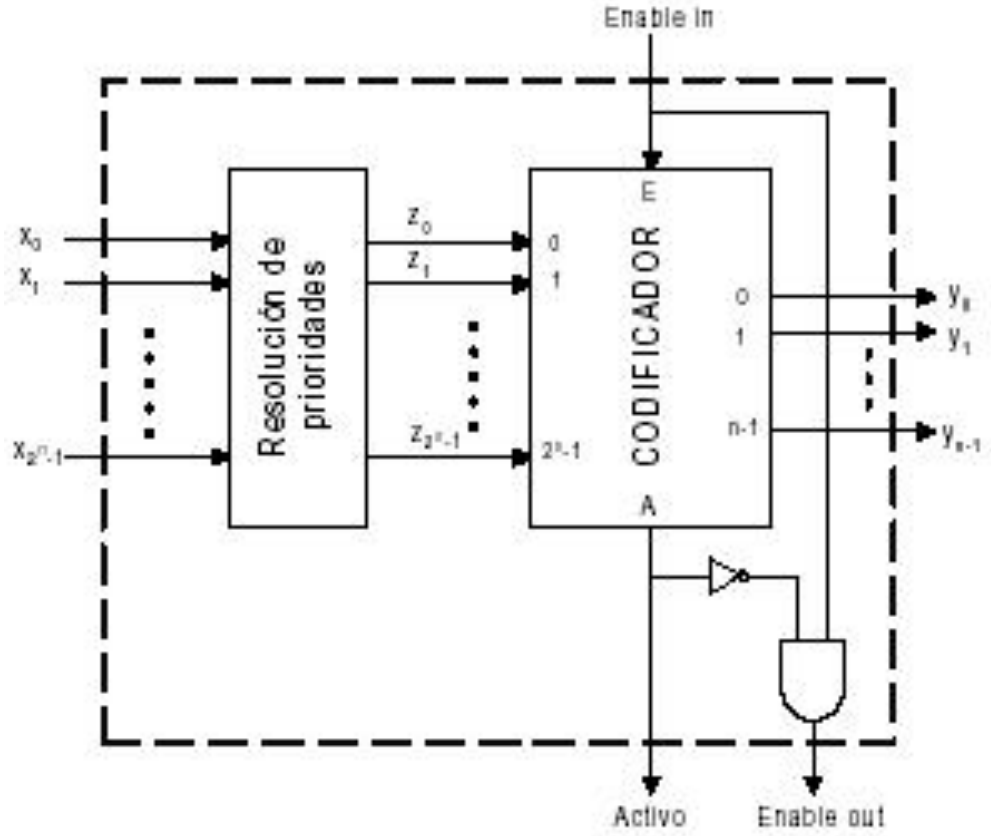
El bloque de resolución de prioridades materializa las **expresiones de conmutación** siguientes:

$$Z_{2^n-1} = X_{2^n-1}$$

$$Z_{2^n-2} = \overline{X_{2^n-1}} \cdot X_{2^n-2}$$

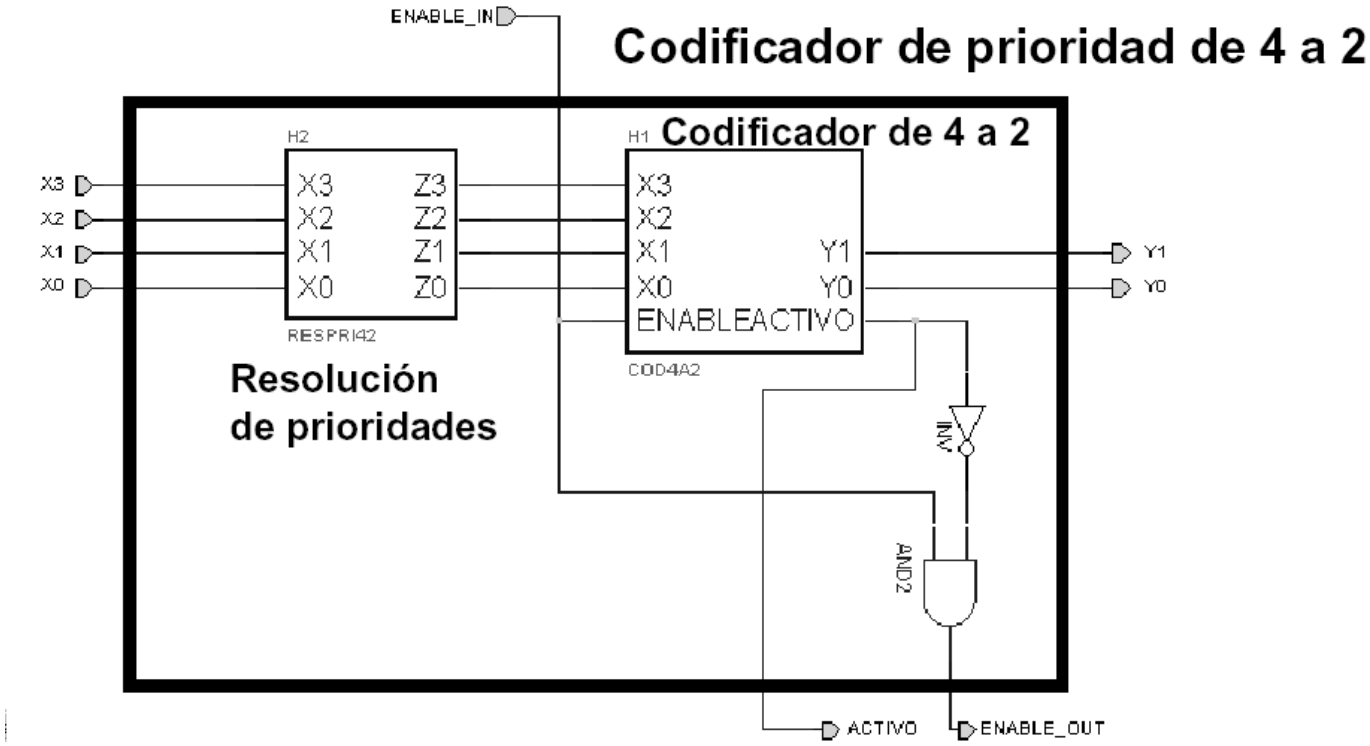
$$\ldots$$

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



Codificadores con prioridad: ejemplo Cod4a2

Ejemplo: codificador con prioridad de 4 a 2.

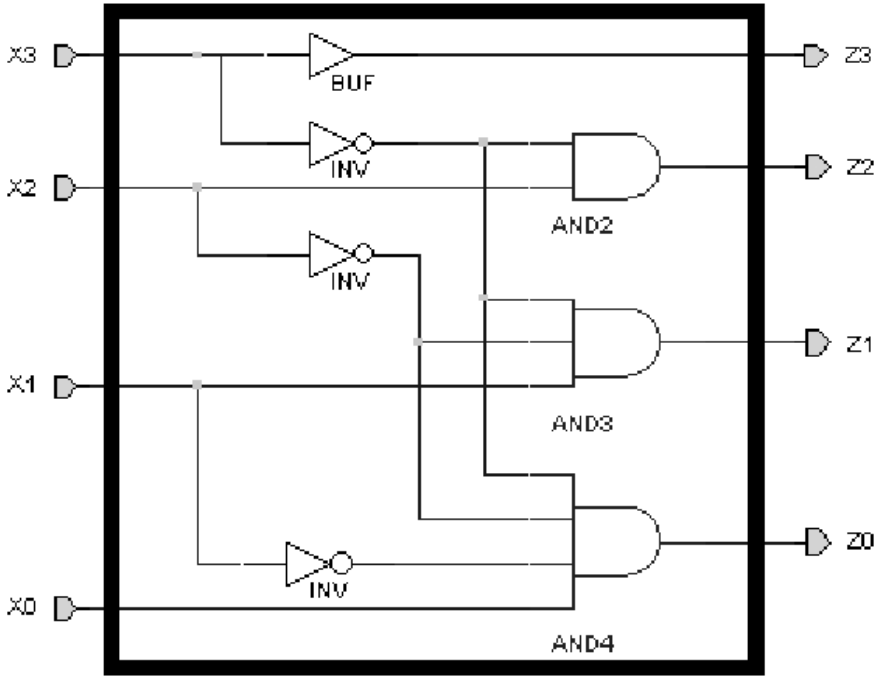


Codificadores con prioridad: ejemplo Cod4a2

Ejemplo: codificador de prioridad de 4 a 2.

Bloque de resolución de prioridades

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

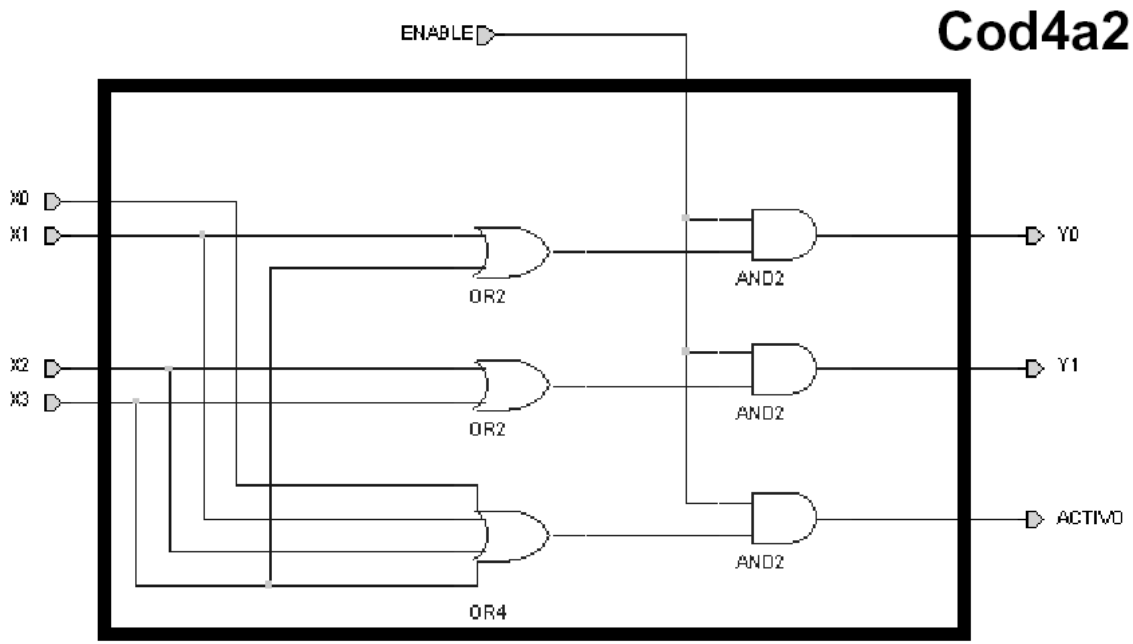


Codificadores con prioridad: ejemplo Cod4a2

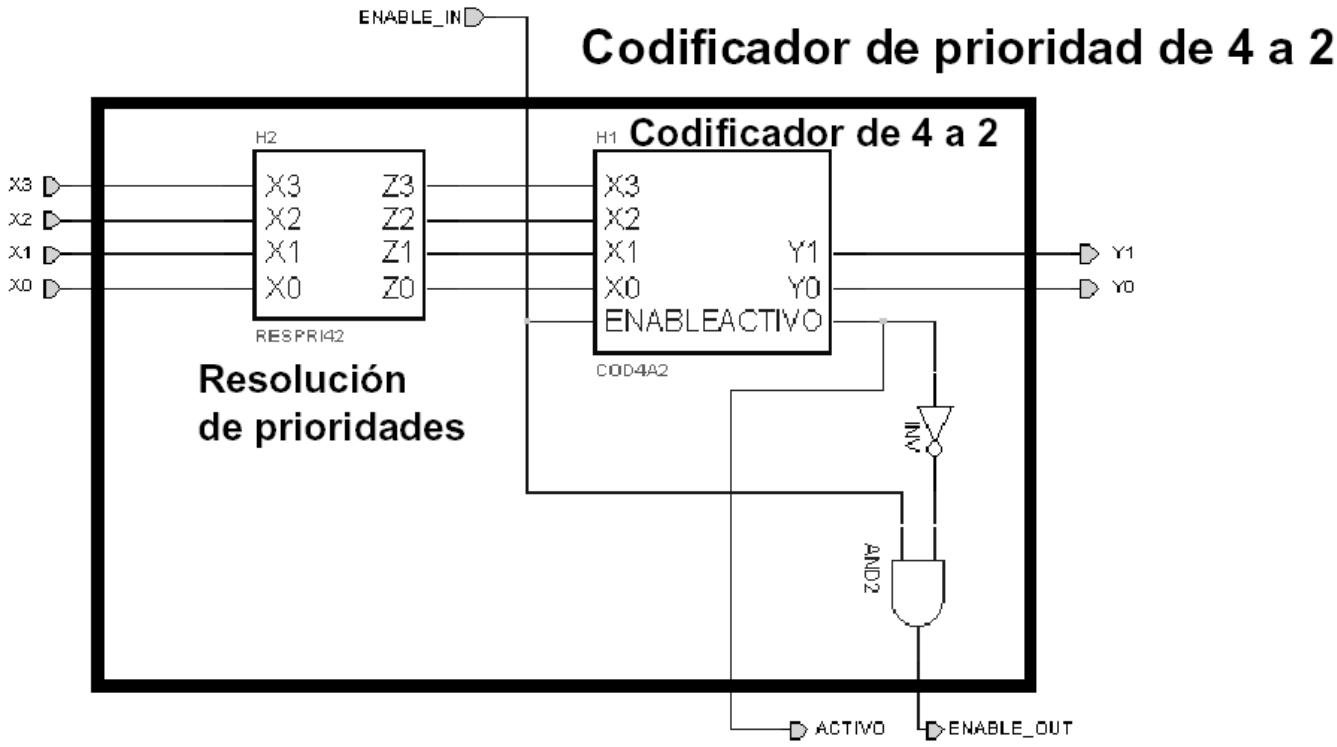
E	X_{activa}	Y_1	Y_0	A
L	x	L	L	L
H	X_0	L	L	H
H	X_1	L	H	H
H	X_2	H	L	H
H	X_3	H	H	H

Codificador de 4 a 2

Sin prioridad



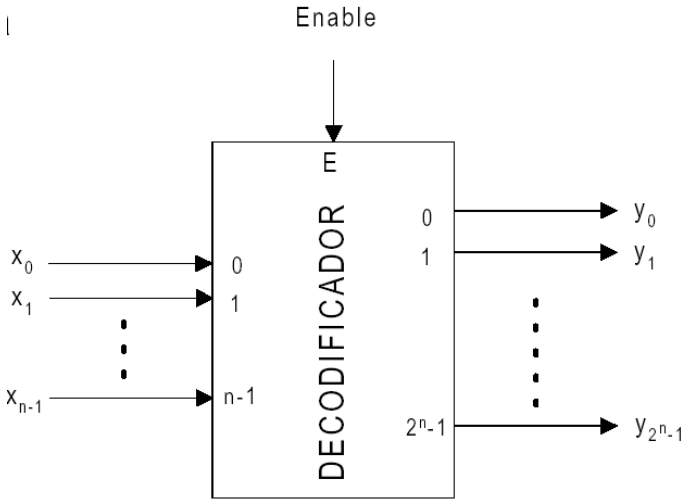
Codificadores con prioridad: ejemplo Cod4a2



3. Decodificadores

Un decodificador (o decodificador de n a 2^n) es un módulo con **n entradas y 2^n salidas,** además de una **señal de activación (*Enable*) de entrada**

El decodificador activa la salida 2i- ésima cuando se presenta la combinación binaria i en las entradas, siempre y cuando el módulo esté activo. Es decir, **se activa la salida correspondiente al número binario codificado en la entrada.**



El **comportamiento** del decodificador

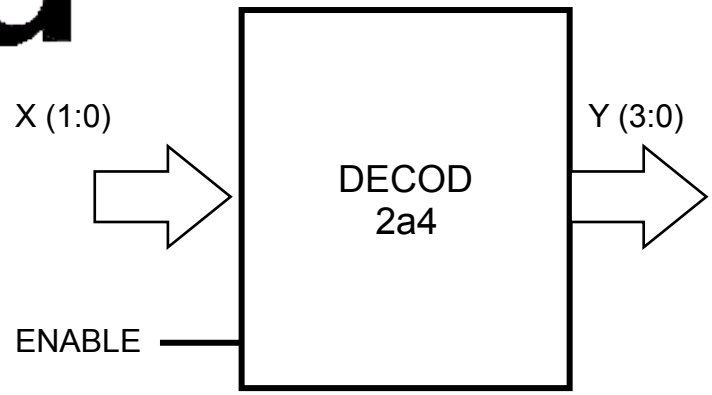
$$X = \sum_{j=0}^{n-1} x_j \cdot 2^j$$

$$y_i = \begin{cases} H & \text{si } X = i \text{ y } E = H \\ L & \text{resto de casos} \end{cases} \qquad \forall i = 0,2^n - 1$$

Expresión de conmutación:

$$y_i = E \cdot m_i(x_{n-1},\ldots x_0) \quad \forall i = 0,2^n - 1$$

Decodificador 2 a 4




```
entity DECOD2a4 is
 port( X: in  std_logic_vector (1 downto 0);
       ENABLE     : in std_logic;
       Y : out std_logic_vector (3 downto 0));
end DECOD2a4;
```

```
architecture comportamental1  of DECOD2a4 is
signal DataEnable: std_logic_vector (2 downto 0);
begin
DataEnable <= ENABLE & DATAIN;
with dataEnable SELECT
Y<= “1000” when “111”;
                “0100” when “110”;
                “0010” when “101”;
                “0001” when “100”;
                “0000” when others;
end comportamental1;
```

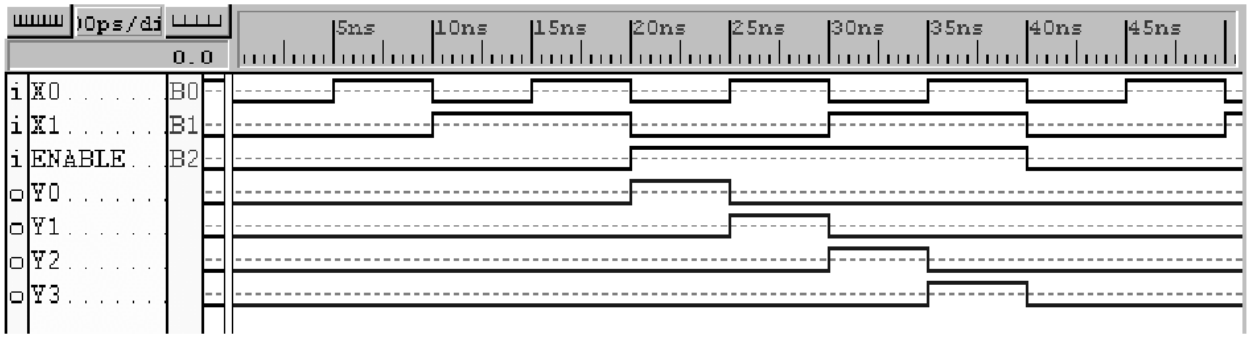
Decodificadores

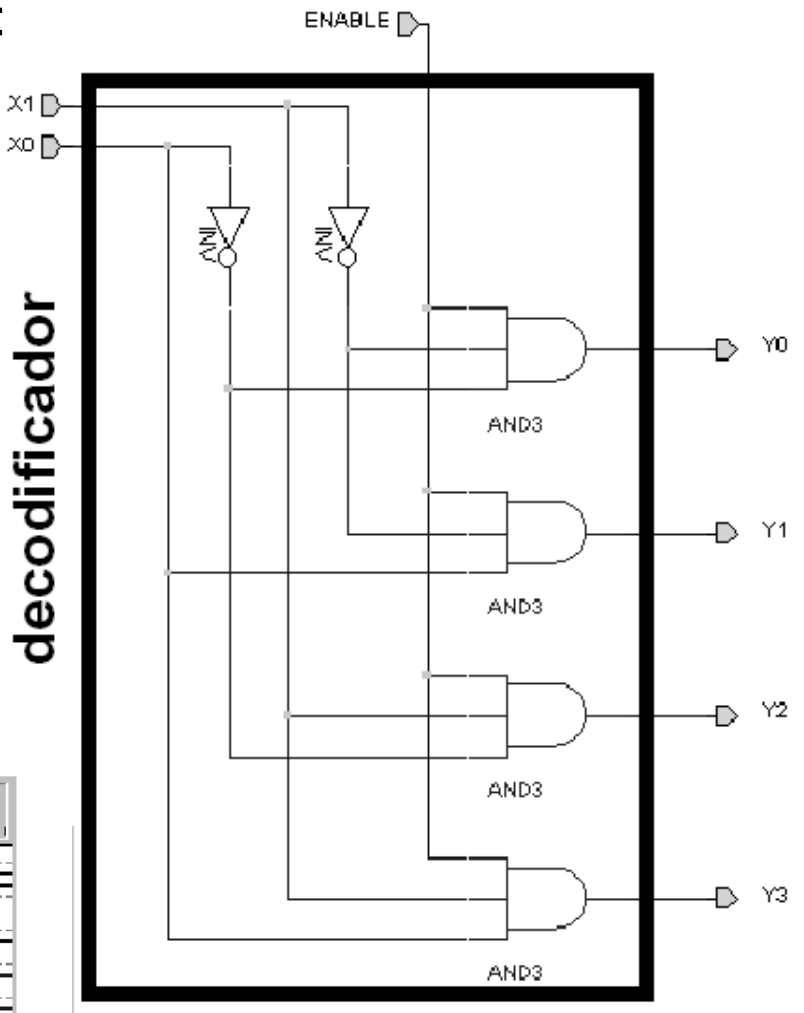
Ejemplo: diseño de un decodificador de 2 a 4:

E	X_1	X_0	Y_3	Y_2	Y_1	Y_0
L	x	x	L	L	L	L
H	L	L	L	L	L	H
H	L	H	L	L	H	L
H	H	L	L	H	L	L
H	H	H	H	L	L	L

Funciones lógicas

$$y_0 = E \cdot m_0(x_1, x_0) = E \cdot \overline{x_1} \cdot \overline{x_0}$$
$$y_1 = E \cdot m_1(x_1, x_0) = E \cdot \overline{x_1} \cdot x_0$$
$$y_2 = E \cdot m_2(x_1, x_0) = E \cdot x_1 \cdot \overline{x_0}$$
$$y_3 = E \cdot m_3(x_1, x_0) = E \cdot x_1 \cdot x_0$$





Decodificadores: diseño jerárquico

Comercial: 74238

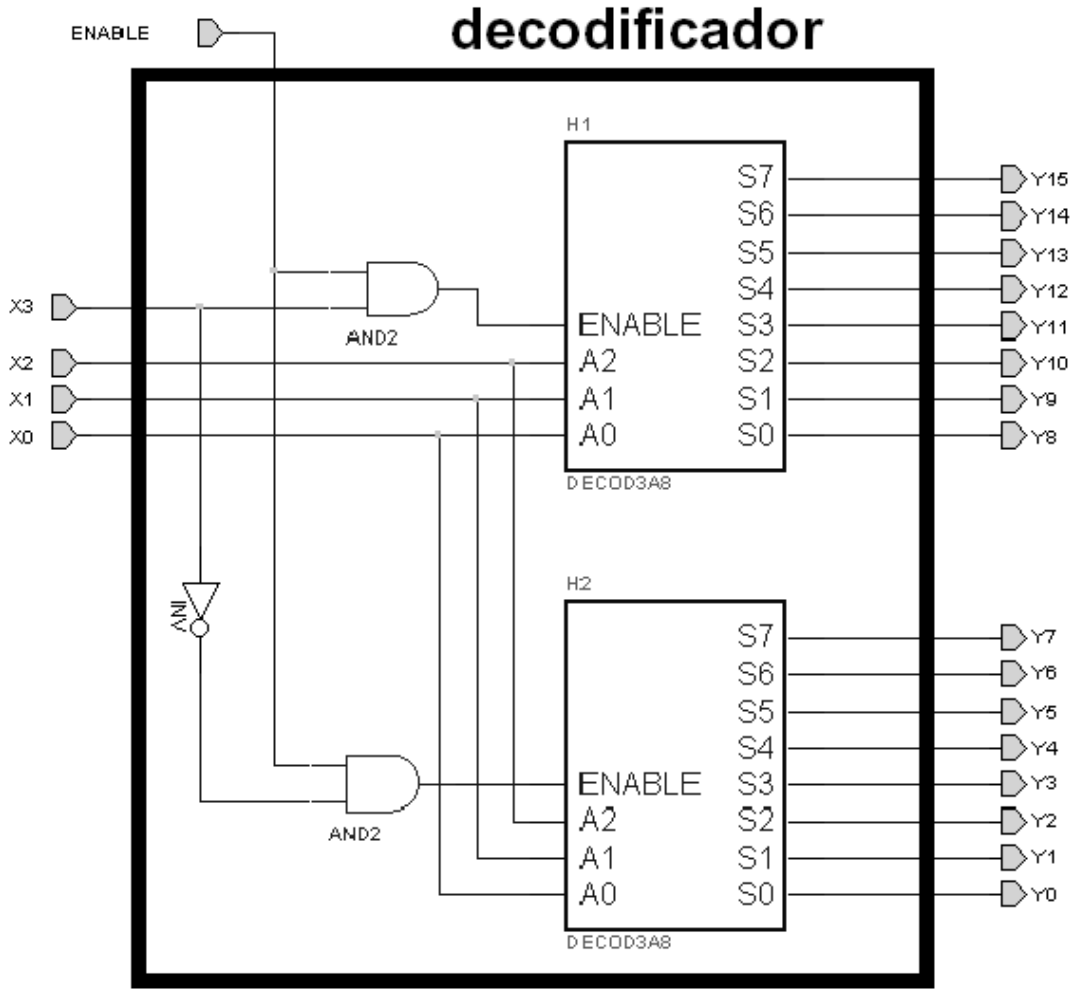
Ejemplo: diseño jerárquico de un decodificador de 4 a 16 mediante decodificadores de 3 a 8.

Decodificador de 3 a 8

E	X_3	X_2	X_1	X_0	Y_{15}	Y_{14}	Y_{13}	Y_{12}	Y_{11}	Y_{10}	Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
L	x	x	x	x	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
H	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	H
H	L	L	L	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L	H	L
H	L	L	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L	H	L	L
H	L	L	H	H	L	L	L	L	L	L	L	L	L	L	L	L	H	L	L	L
H	L	H	L	L	L	L	L	L	L	L	L	L	L	L	L	H	L	L	L	L
H	L	H	L	H	L	L	L	L	L	L	L	L	L	L	H	L	L	L	L	L
H	L	H	H	L	L	L	L	L	L	L	L	L	L	H	L	L	L	L	L	L
H	L	H	H	H	L	L	L	L	L	L	L	L	H	L	L	L	L	L	L	L
H	H	L	L	L	L	L	L	L	L	L	L	H	L	L	L	L	L	L	L	L
H	H	L	L	H	L	L	L	L	L	L	H	L	L	L	L	L	L	L	L	L
H	H	L	H	L	L	L	L	L	L	H	L	L	L	L	L	L	L	L	L	L
H	H	L	H	H	L	L	L	L	H	L	L	L	L	L	L	L	L	L	L	L
H	H	H	L	L	L	L	L	H	L	L	L	L	L	L	L	L	L	L	L	L
H	H	H	L	H	L	L	H	L	L	L	L	L	L	L	L	L	L	L	L	L
H	H	H	H	L	L	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L
H	H	H	H	H	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L

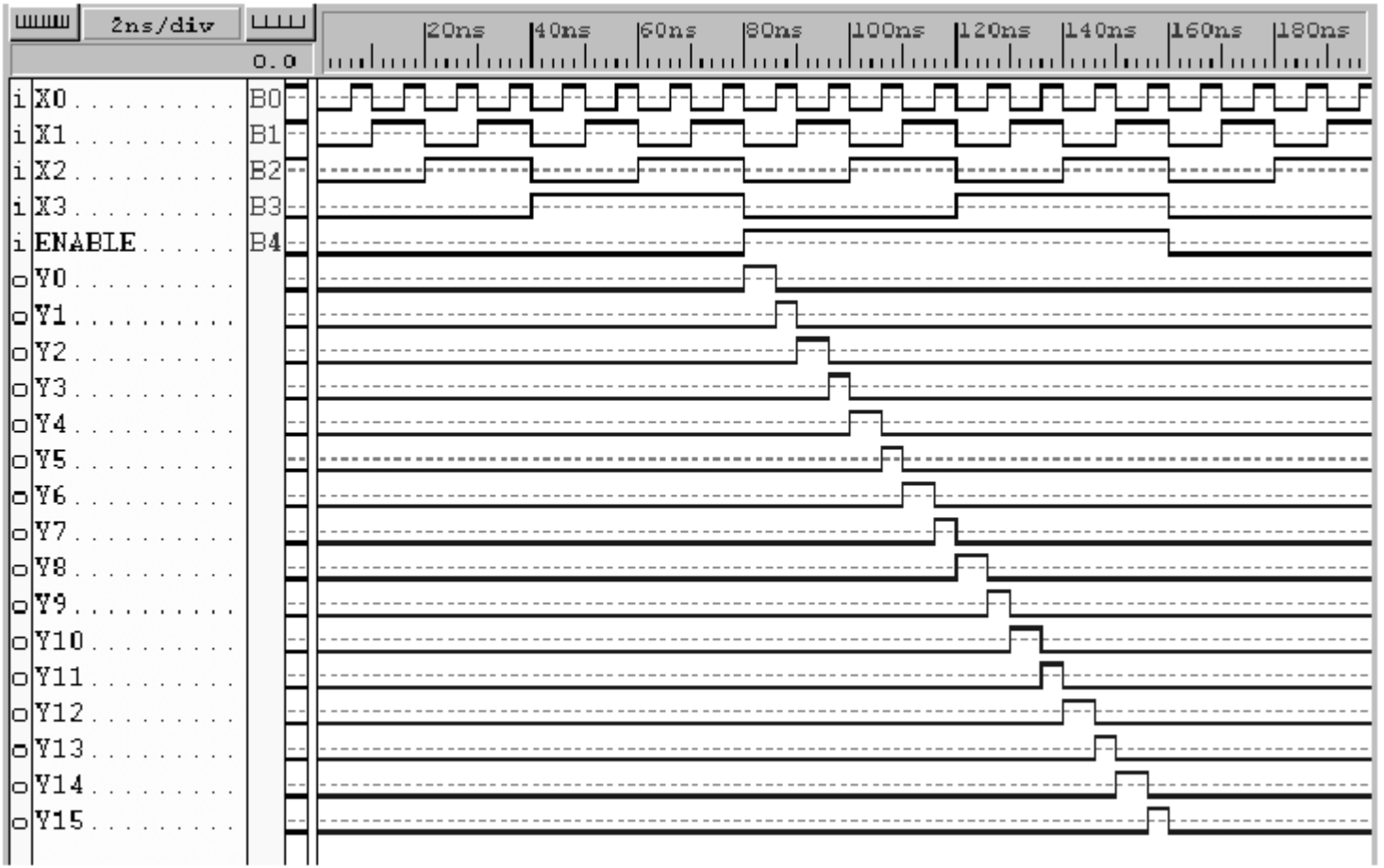
Decodificadores: diseño jerárquico

Ejemplo: diseño jerárquico de un decodificador de 4 a 16 mediante decodificadores de 3 a 8.



Decodificadores: diseño jerárquico

Ejemplo: diseño jerárquico de un decodificador de 4 a 16 mediante decodificadores de 3 a 8. Resultados de simulación.

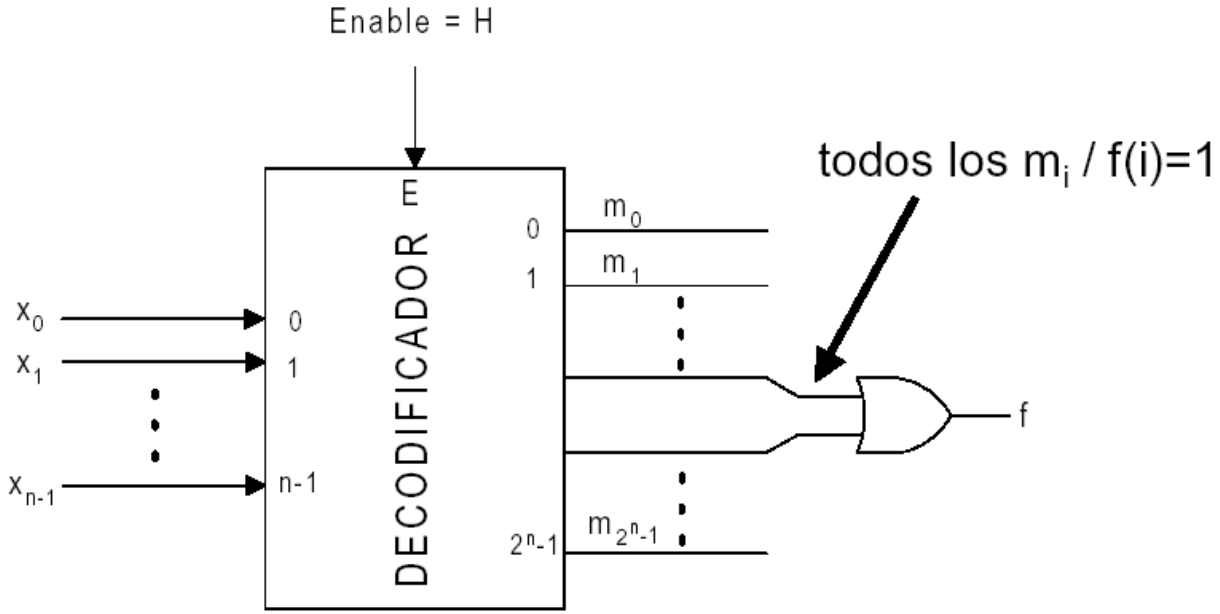


Síntesis de FC con Decodificadores

Un decodificador (o decodificador de n a 2^n) es un módulo que **materializa todos los minterms de una función de n variables.**

Expresión de conmutación: $y_i = E \cdot m_i(x_{n-1},...x_0) \quad \forall i = 0,2^n - 1$

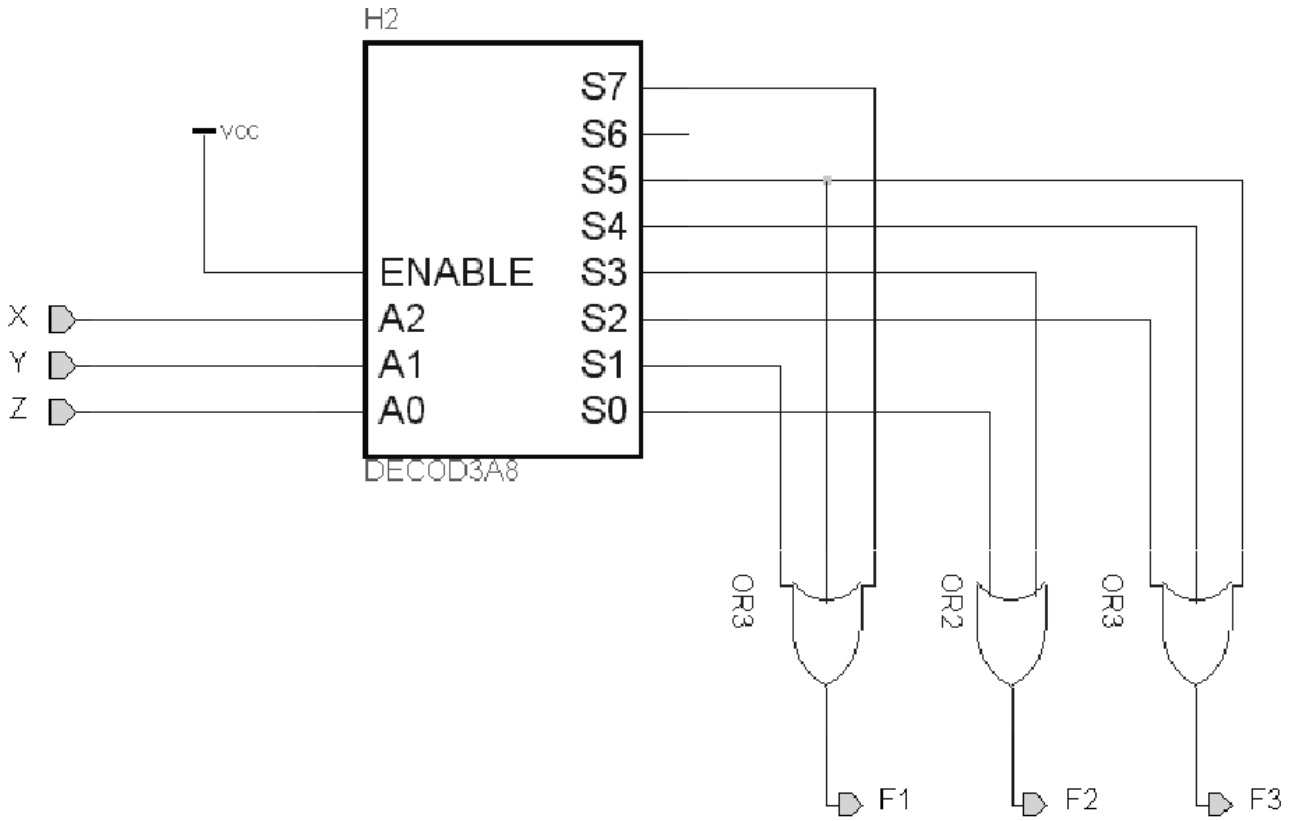
Se puede materializar cualquier FC de n variables expresada como suma de minterms sin más que usar **un decodificador de n a 2^n y una puerta OR con tantas entradas como sumandos tenga la expresión de la FC.**



Síntesis de FC con Decodificadores

Ejemplo: diseño de las funciones f1,f2 y f3 mediante decodificadores.

$$f_1(x,y,z) = \sum m(1,5,7)$$
$$f_2(x,y,z) = \sum m(0,3)$$
$$f_3(x,y,z) = \sum m(2,4,5)$$



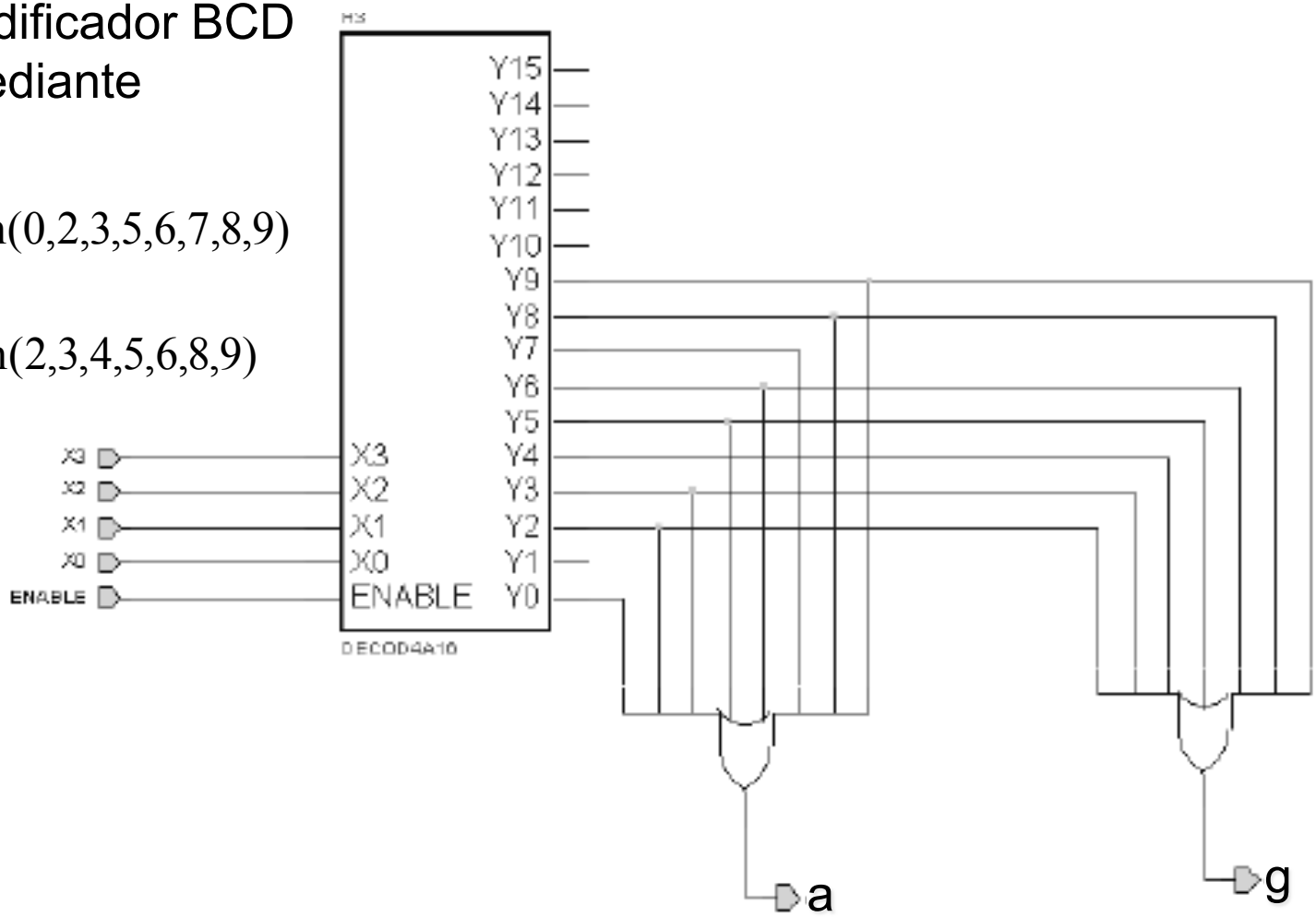
Síntesis de FC con Decodificadores

Materializar un codificador BCD a 7 segmentos mediante decodificadores

$$a(x_3, x_2, x_1, x_0) = \sum m(0,2,3,5,6,7,8,9)$$

…

$$g(x_3, x_2, x_1, x_0) = \sum m(2,3,4,5,6,8,9)$$



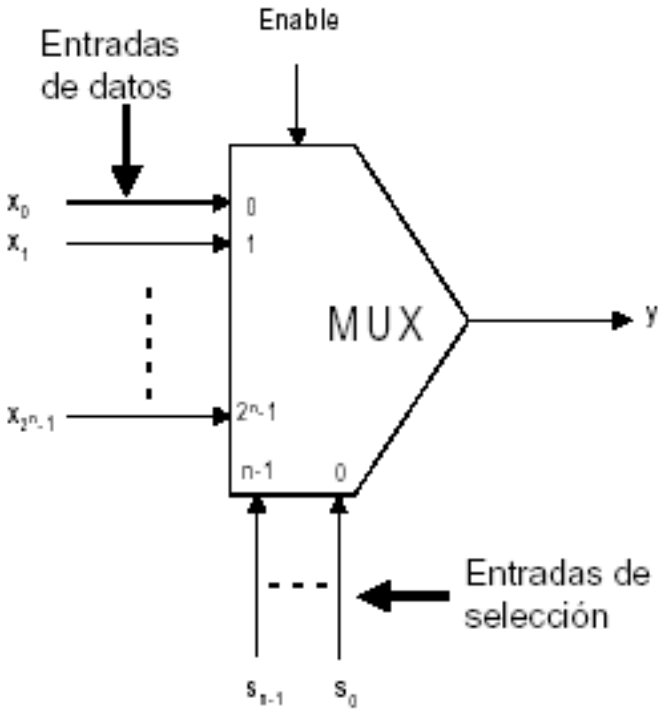
Aplicaciones de los Decodificadores

- Chips de memoria : Convierte el nº binario que designa la dirección de una celda de memoria en la fila o columna correspondiente.
- Sistemas de memoria de microprocesadores: Seleccionando diferentes bancos de memoria.
- Sistemas de entrada/salida de microprocesadores: Seleccionando distintos dispositivos.
- Descodificando instrucciones de microprocesadores: Habilitando diferentes unidades funcionales
- Displays (descodificador BCD-7 segmentos, etc)
- Teclados (asignando valores binarios a códigos ASCII, etc)

4. Multiplexores

Un multiplexor (o multiplexor de 2^n a 1) es un módulo combinacional con 2^n entradas y 1 salida, además de una señal de activación y n señales de control.

El multiplexor conecta una de las 2^n entradas a la salida salidas. Esta entrada se selecciona con la palabra de control S.



El **comportamiento** del multiplexor:

$$S = \sum_{j=0}^{n-1} s_j \cdot 2^j \qquad \text{entonces:}$$

$$y = \begin{cases} x_i & \text{si } S = i \text{ y } E = H \\ L & \text{resto de casos} \end{cases}$$

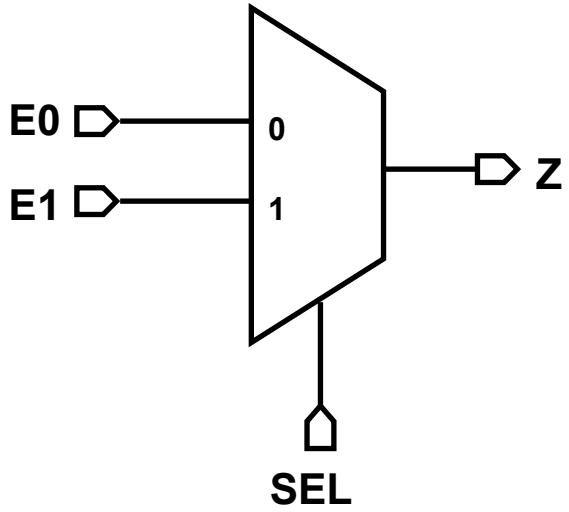
Expresión de conmutación:

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

MUX 2 a 1

Comercial: 74157

```
entity MUX is
 port( E0,E1 : in  bit;
       SEL: in  bit;
       Z   : out bit);
end MUX;
```



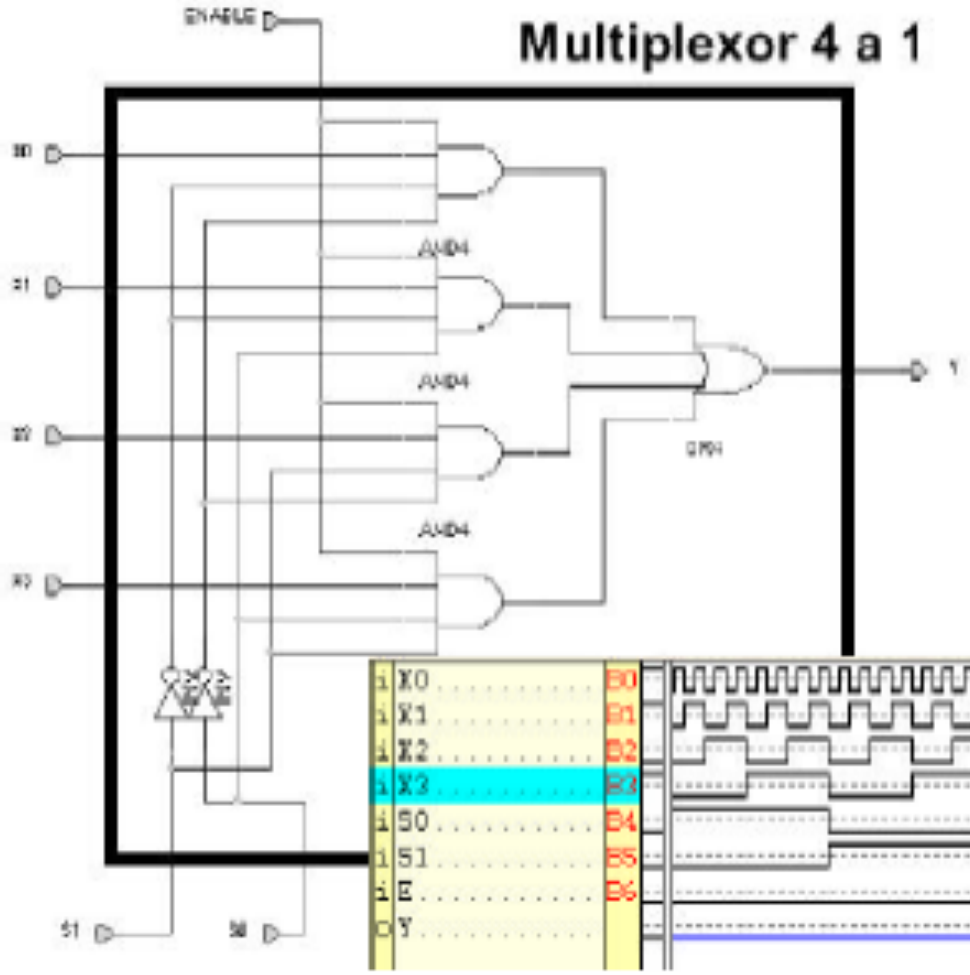

```
architecture comportamental1 of MUX is
begin
 Z <= E0 when SEL = '0'
      else E1;
end comportamental1;
```

```
architecture comportamental2 of MUX is
begin
 P1: process(E0, E1, SEL)
 begin
    if SEL = '0' then
      Z <= E0;
    else
      Z <= E1;
    end if;
 end process P1;
end comportamental2;
```

Multiplexores

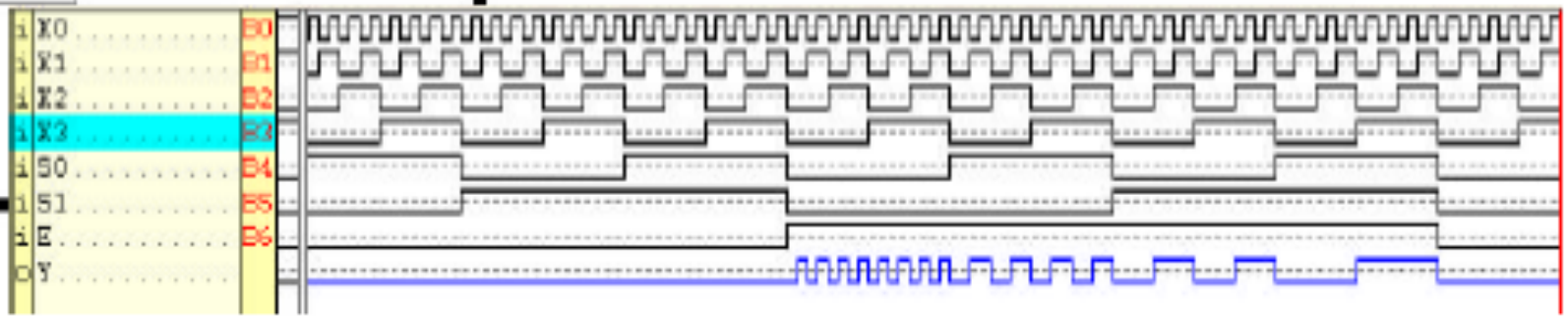
Ejemplo: diseño de un multiplexor de 4 entradas (4 a 1).

Comercial: 74293



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

$$= E \cdot (x_0 \cdot \overline{s_1} \cdot \overline{s_0} + x_1 \cdot \overline{s_1} \cdot s_0 + x_2 \cdot s_1 \cdot \overline{s_0} + x_3 \cdot s_1 \cdot s_0)$$



Síntesis de FC con multiplexores

Un único **multiplexor de 2^n a 1 permite materializar cualquier función de conmutación de n variables.**

La expresión de conmutación 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:

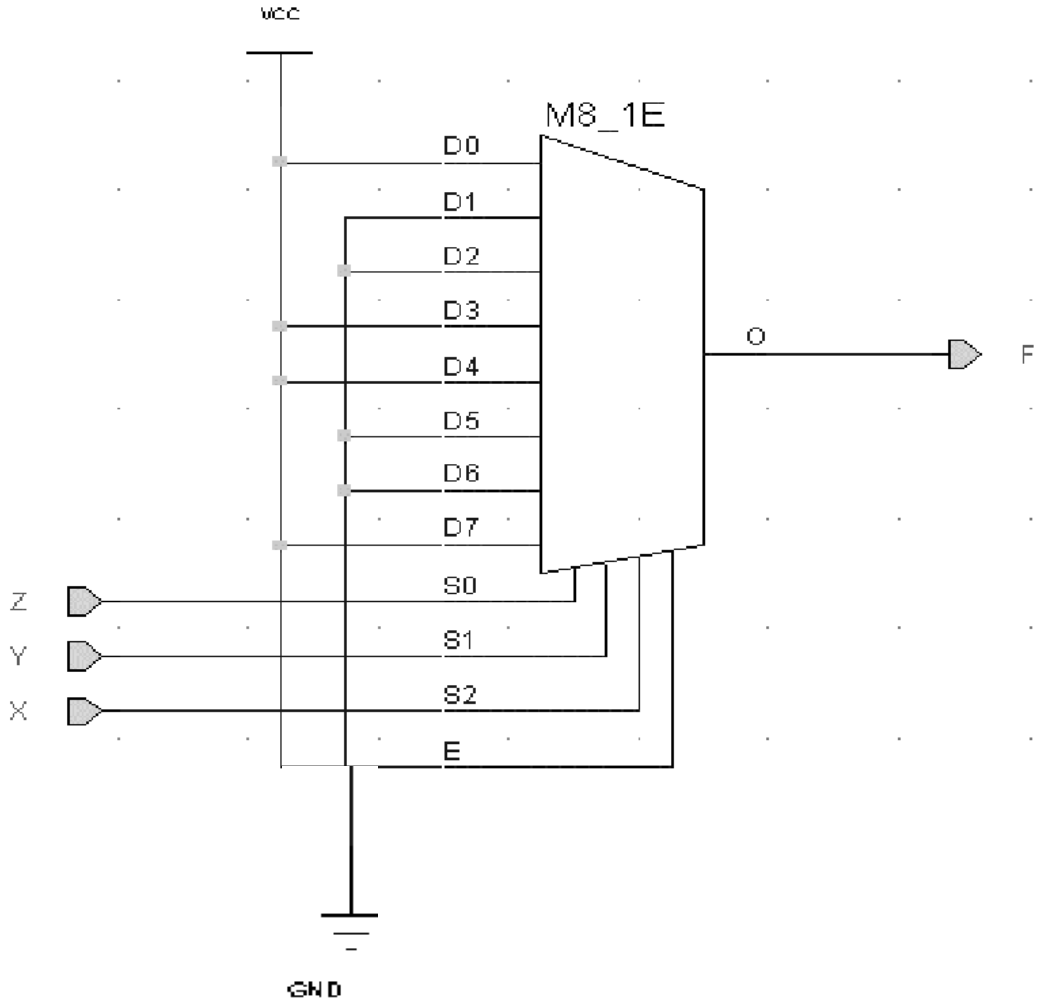
$$f(a_{n-1},\ldots,a_0)=\sum_{i=0}^{2^n-1} f(i)\cdot m_i(a_{n-1},\ldots,a_0) \quad \approx \quad y=E\cdot\sum_{i=0}^{2^n-1} x_i\cdot m_i(s_{n-1},\ldots,s_0)$$

Obviando E, esta expresión coincide con la expresión del multiplexor si se identifican: $x_i = f(i)$ $\forall i=0,\ldots,2^n-1$ y $(s_{n-1},\ldots,s_0)=(a_{n-1},\ldots,a_0)$.

Esto significa que para materializar f basta con conectar las entradas binarias de la función a las entradas de control del multiplexor y conectar el valor f(i) que toma la función (fila i de la tabla de verdad) con la entrada de datos x_i del multiplexor.

Síntesis de FC con multiplexores

Ejemplo: diseño mediante un único multiplexor de la función f siguiente.



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

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

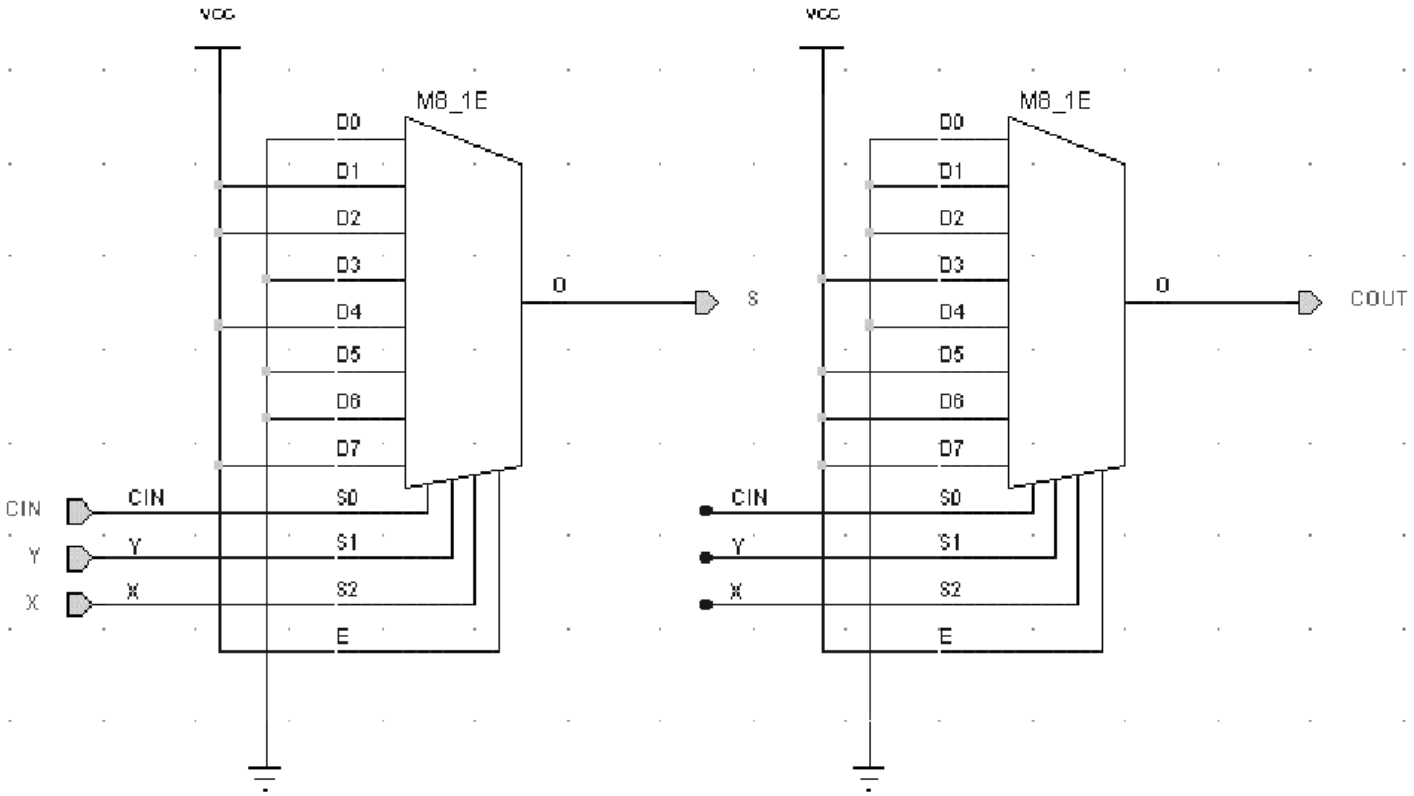
Síntesis de FC con multiplexores

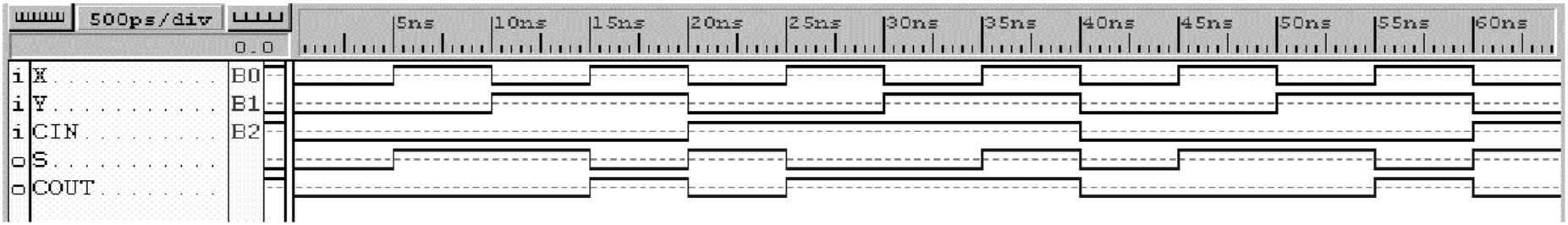
Ejemplo: diseño mediante multiplexores de un sumador binario completo de 1 bit.

Síntesis de FC con multiplexores

Ejemplo: diseño mediante multiplexores de un sumador binario com de 1 bit.

$$S = \sum m(1,2,4,7) \qquad COUT = \sum m(3,5,6,7)$$



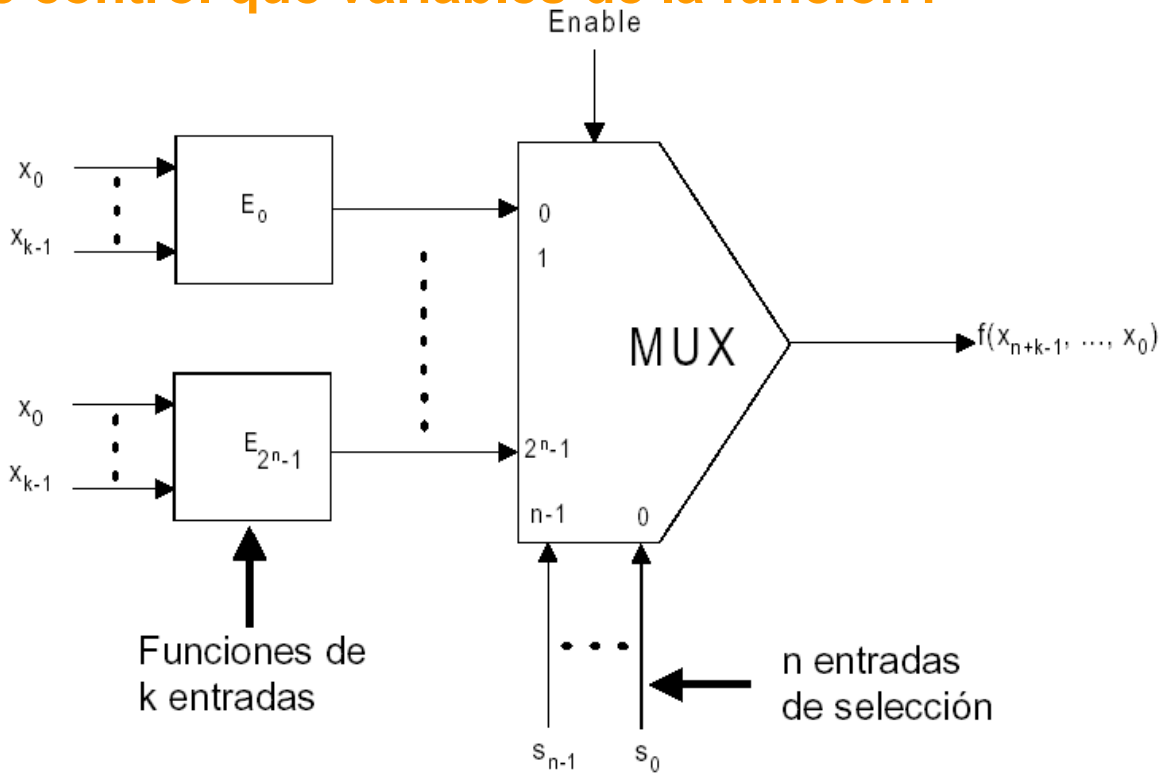


Síntesis de FC con multiplexores

¿Qué pasa si la materialización se hace con un multiplexor con menor nº de entradas (n) de control que variables de la función?

- Seleccionar n de las variables de entrada para usarlas como control y utilizar las restantes para generar los valores de las 2^n entradas del multiplexor

- Cada una de las funciones de k entradas puede realizarse con multiplexores de k entradas de control o con otros módulos.



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

Síntesis de FC con multiplexores

Ejemplo: diseño de la función f siguiente mediante un multiplexor de 4 a 1 y puertas lógicas.

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

$$f(a,b,c,d) = \bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}cd + +\bar{a}b\bar{c}\bar{d} + +\bar{a}bc\bar{d} + +\bar{a}bcd + a\bar{b}\bar{c}d + a\bar{b}c\bar{d} + a\bar{b}cd + abc\bar{d} =$$

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

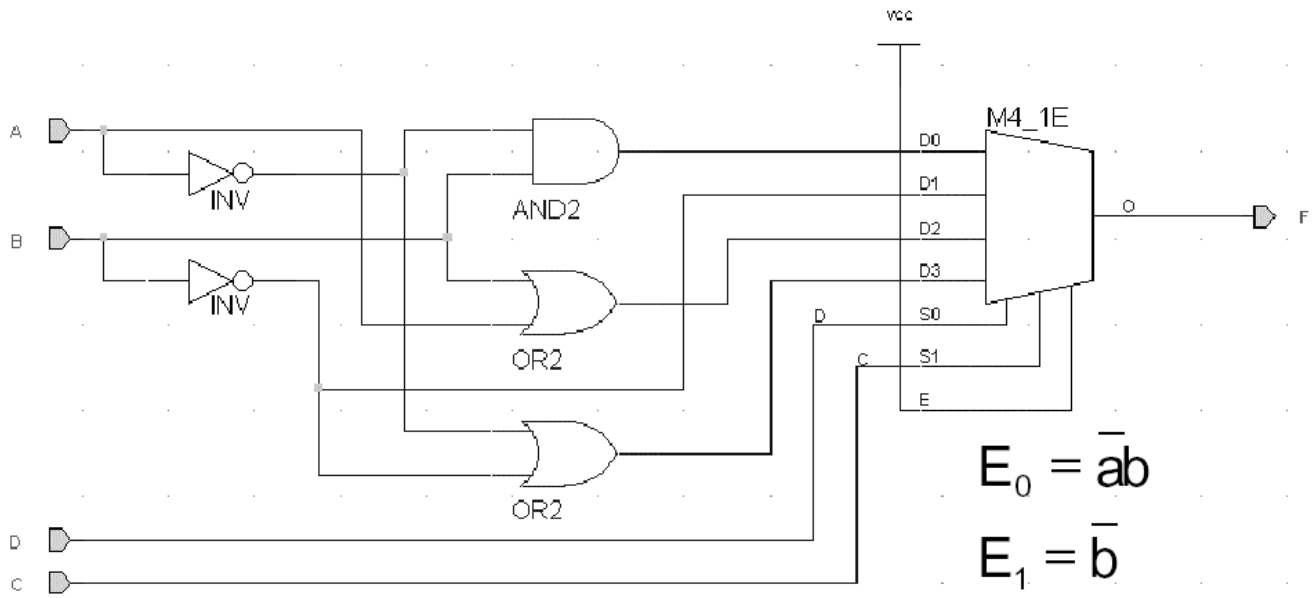
$(z_1, z_0) = (a,b)$

$(s_1, s_0) = (c,d)$

$$E_0 = (\bar{a}b)$$
$$E_1 = (\bar{a}\bar{b} + a\bar{b}) = \bar{b}$$
$$E_2 = (\bar{a}\bar{b} + a\bar{b} + ab) = a + b$$
$$E_3 = (\bar{a}\bar{b} + \bar{a}b + a\bar{b}) = \bar{a} + \bar{b}$$

} Simplificación algebraica.

Síntesis de FC con multiplexores

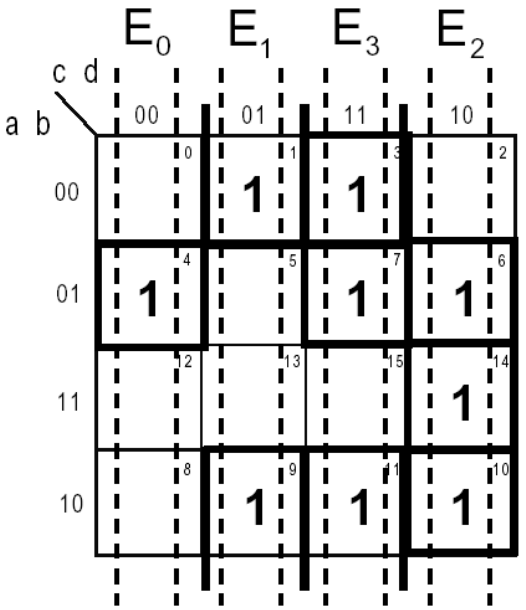


$$E_0 = \bar{a}b$$

$$E_1 = \bar{b}$$

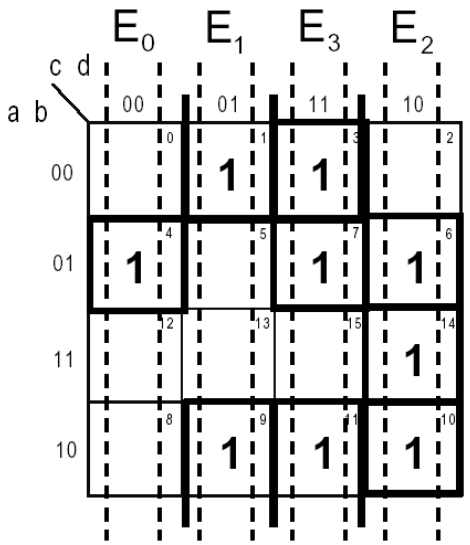
$$E_2 = a + b$$

$$E_3 = \bar{a} + \bar{b}$$

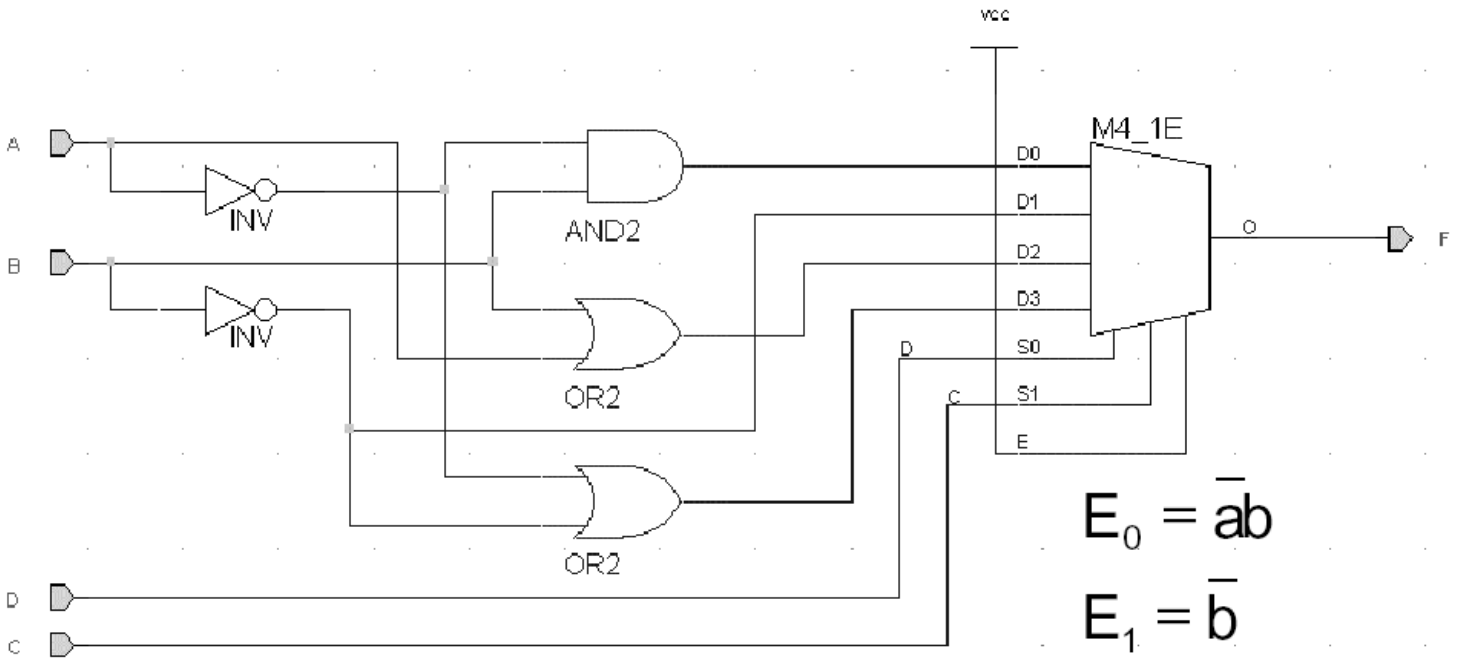


Simplificación con Karnaugh

Síntesis de FC con multiplexores



Simplificación con Karnaugh



$$E_0 = \bar{a}b$$

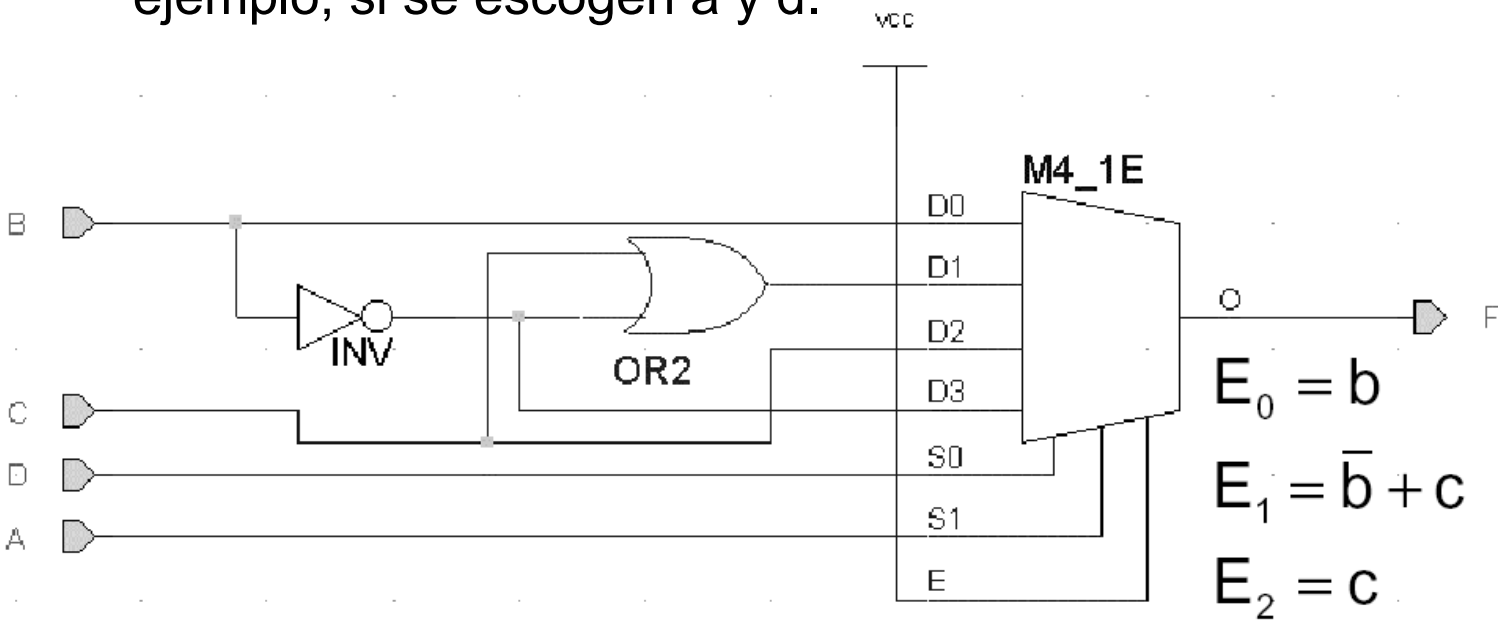
$$E_1 = \bar{b}$$

$$E_2 = a + b$$

$$E_3 = \bar{a} + \bar{b}$$

Síntesis de FC con multiplexores

La selección de otras señales de control puede variar la materialización. Por ejemplo, si se escogen a y d.

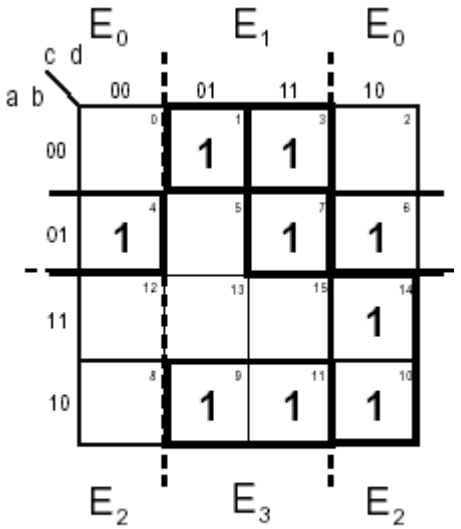


$$E_0 = b$$

$$E_1 = \bar{b} + c$$

$$E_2 = c$$

$$E_3 = \bar{b}$$



Simplificación con Karnaugh

Aplicaciones de los Multiplexores

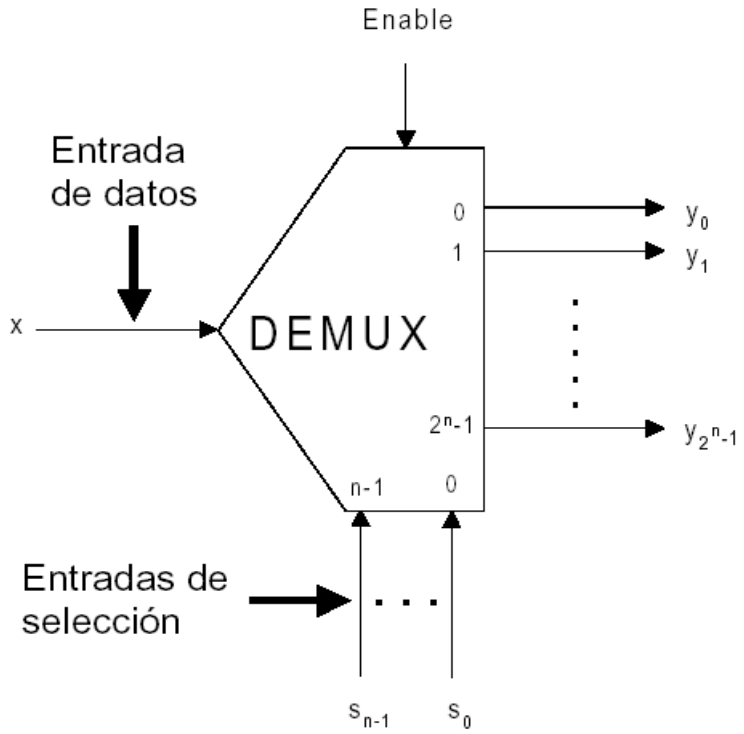
- Un MUX puede implementar cualquier función lógica con un nº de variables igual a los selectores.
- Un MUX se implementa con un nº razonable de puertas NAND.
- MUX de pocas entradas se utilizan como elemento básico de los bloques lógicos programables de FPGAs.
- Conversión paralelo-serie.
- Multiplexación de bits de dirección en memorias grandes

5. Demultiplexores

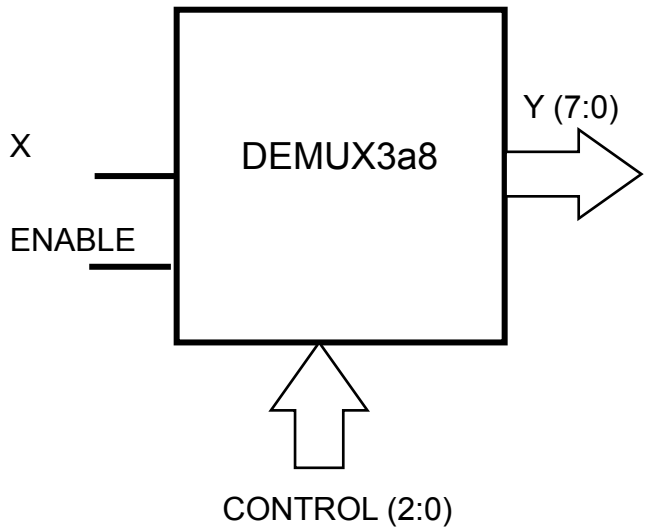
Un demultiplexor (o demultiplexor de 2^n a 1) es un módulo con 1 **entrada y 2^n salidas**, además de **una señal de activación y n señales de control.**

El demultiplexor conecta la entrada con una de las 2^n salidas que se selecciona con la palabra de control S.

Su funcionamiento es inverso al realizado por el multiplexor.



DEMUX 3 a 8



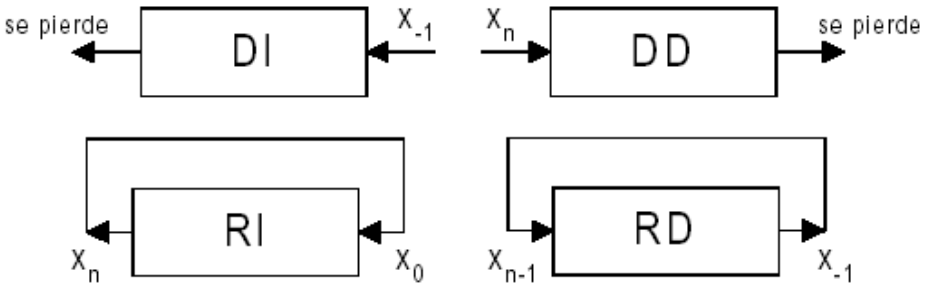

```
entity DEMUX3a8 is
 port( X,ENABLE : in std_logic;
       CONTROL          : in  std_logic_vector (2 downto 0);
       Y          : out std_logic_vector (7 downto 0));
end DEMUX3a8;
```

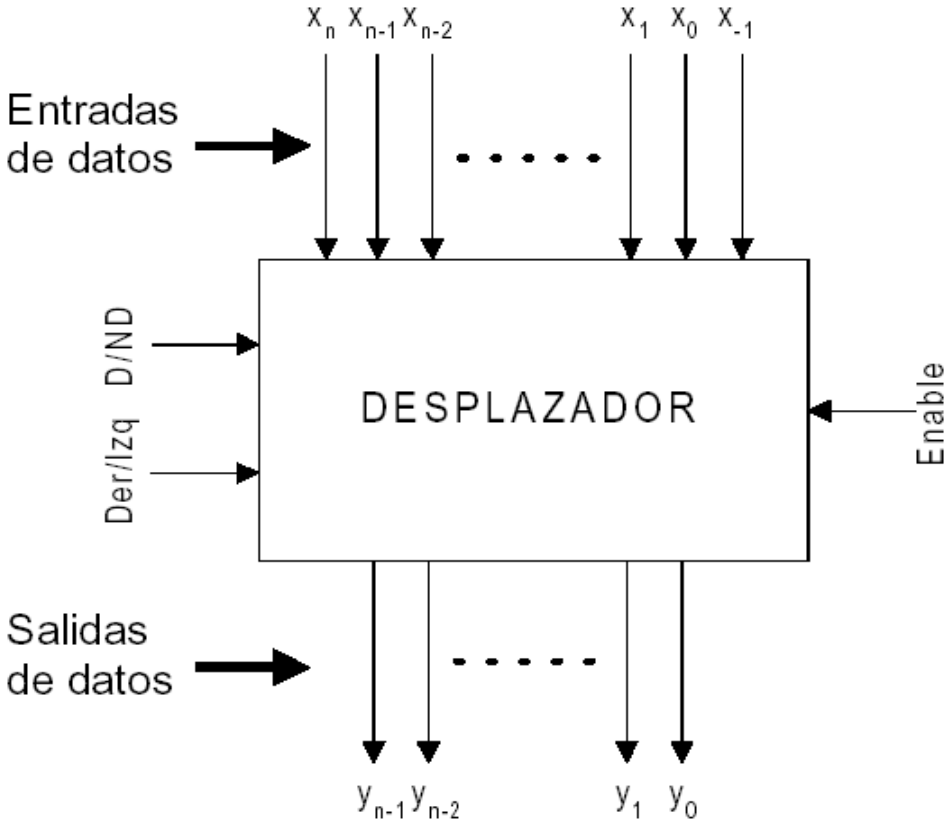
```
architecture comportamental  of DEMUX3a8 is
signal DataControl: std_logic_vector (3 downto 0);
signal Entrada: std_logic;
begin
DataEnable <= ENABLE & CONTROL;
Entrada <= X;
with dataEnable SELECT
Y(7)<= Entrada when “1111”;
Y(6)<= Entrada when “1110”;
Y(5)<= Entrada when “1101”;
Y(4)<= Entrada when “1100”;
Y(3)<= Entrada when “1011”;
Y(2)<= Entrada when “1010”;
Y(1)<= Entrada when “1001”;
Y(0)<= Entrada when “1000”;
Y    <= (others =>’0’)when others;
end comportamental;
```

6. Desplazadores

Un desplazador (*shifter*) es un módulo combinacional con **n+2 entradas** de datos y **n salidas**, además de **una señal de activación** y **señales de control.**

El desplazador puede mover o no bits a derecha e izquierda en desplazamientos abiertos o cerrados (rotaciones) bajo las órdenes de las señales control.



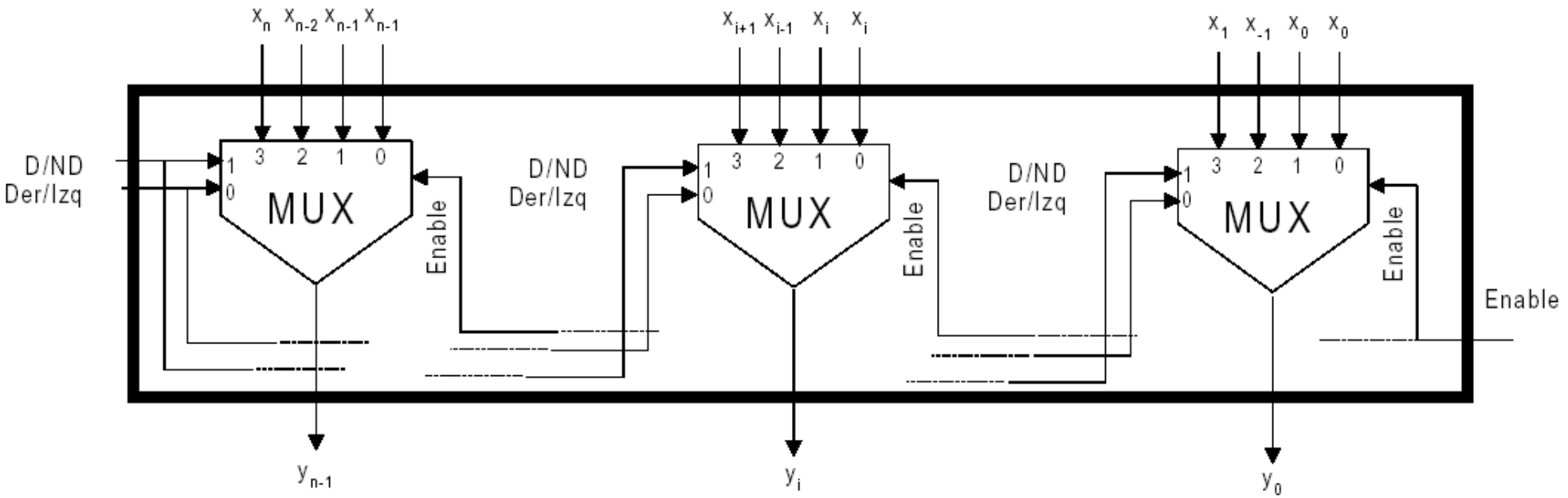


La construcción habitual suele consistir en un conjunto de multiplexores.

Desplazadores

Ejemplo: materializar mediante multiplexores el siguiente desplazador :

$$y_i = \begin{cases} x_{i-1} & \text{si D/ND} = H \text{ y Der/Izq} = L \text{ y E} = H \\ x_i & \text{si D/ND} = L \text{ y E} = H \\ x_{i+1} & \text{si D/ND} = H \text{ y Der/Izq} = H \text{ y E} = H \end{cases}$$



7. Dispositivos lógicos programables

Dispositivos lógicos programables: **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**.

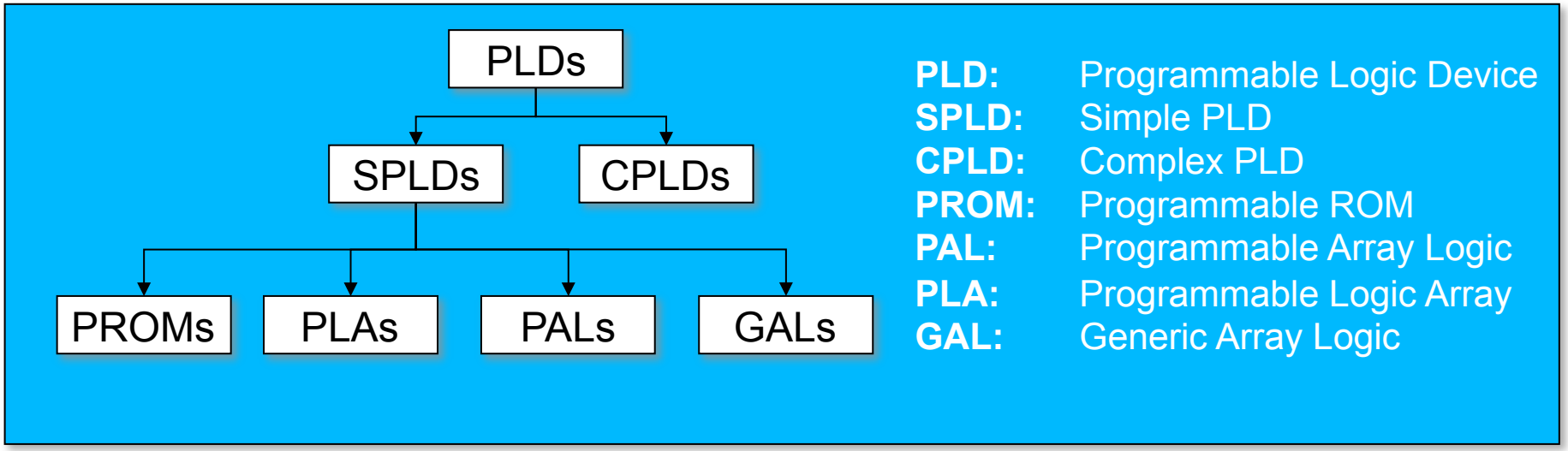
La gran ventaja de estos dispositivos reside en que los fabricantes pueden realizar grandes tiradas de estos CI lo que abarata sus costes de producción y los usuarios posteriormente pueden personalizar sus diseños en sus propios laboratorios sin grandes inversiones:

- Consumos medios, aunque hay familias especializadas en bajo consumo
- Velocidad intermedia
- Fiabilidad alta
- Tiempo de desarrollo muy bajo, sin dependencia de terceros
- Metodología sencilla
- Equipamiento sencillo
- Aumentan la confidencialidad de las placas

Dispositivos lógicos programables

Evolución temporal y escala de integración:

PAL,PLDs	Suma de productos de entradas y salidas realimentadas	200-1000
CPLDs	Varias PALs interconexionadas entre sí	1k-10k
FPGAs	Bloques lógicos configurables con rutas de interconexión no prefijadas	10k-10M



PLD: Programmable Logic Device
SPLD: Simple PLD
CPLD: Complex PLD
PROM: Programmable ROM
PAL: Programmable Array Logic
PLA: Programmable Logic Array
GAL: Generic Array Logic

FPGAs

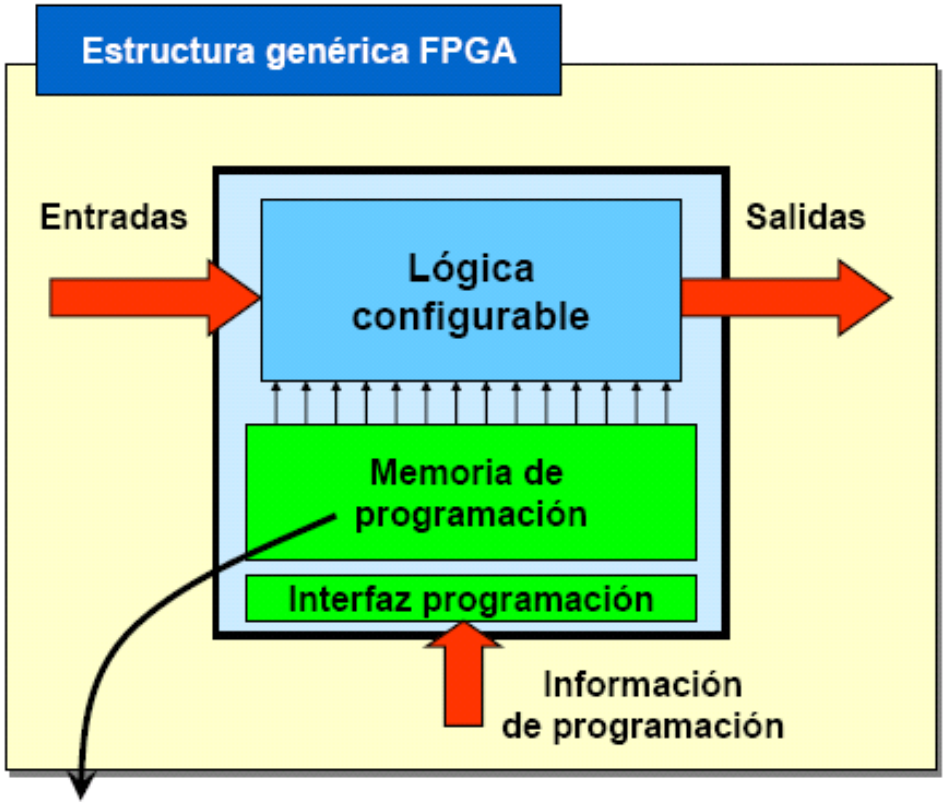
Dispositivo lógico programable (su funcionalidad se fija por el usuario después de la fabricación)
Field Programmable Gate Array

Programación distinta a la de un ordenador o microprocesador

Ordenador o µP	FPGA
Programar consiste en cambiar las instrucciones que le llegan al microprocesador	Programar consiste en cambiar las conexiones y entradas/salidas de la lógica del dispositivo
⇓	⇓
Cambia el SOFTWARE	Cambia el HARDWARE

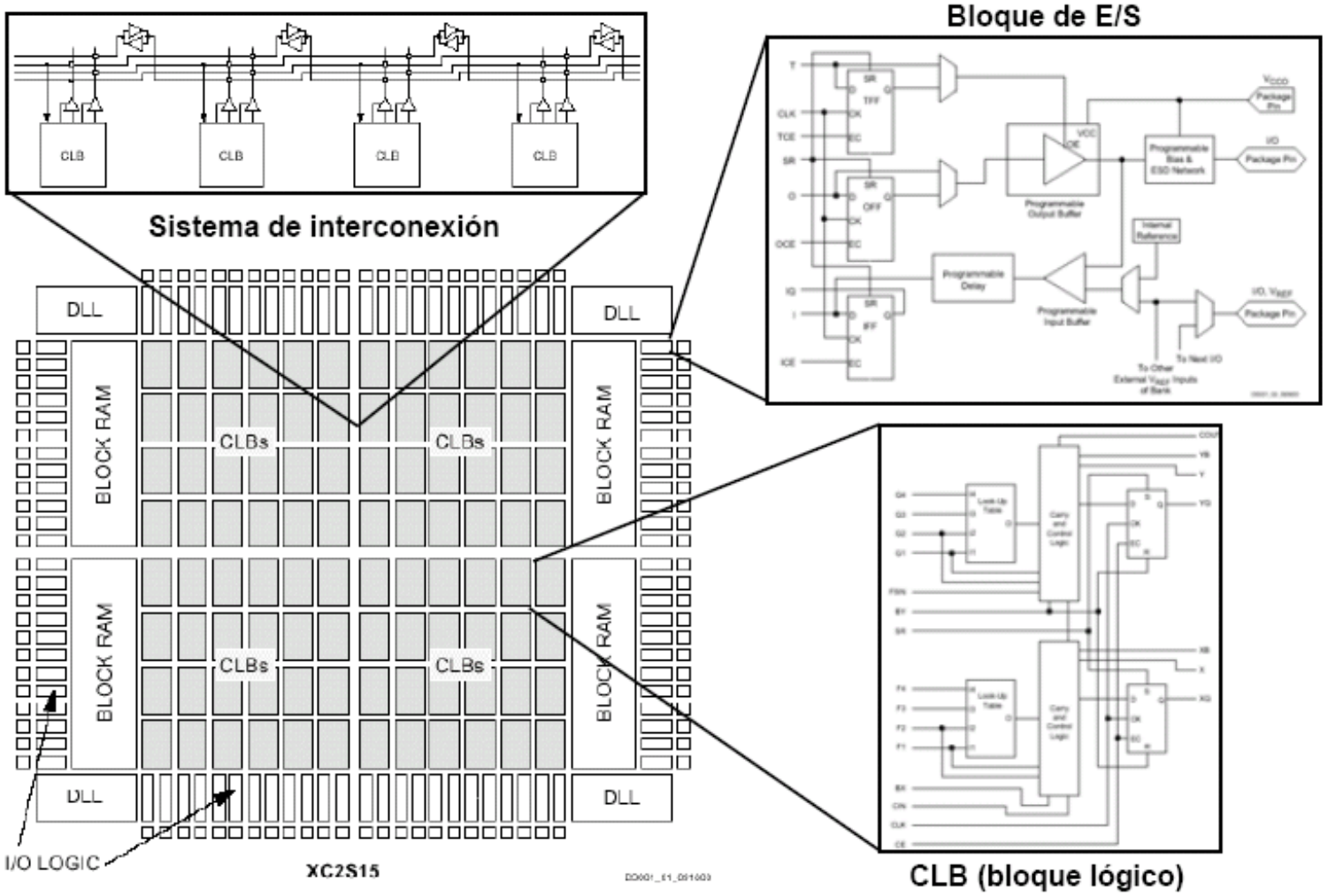
FPGAs

Programar: cambiar las funciones lógicas y las conexiones



La implementación física de la memoria sirve para clasificar las **FPGAs**

FPGAs. Arquitectura



Comparativa de las FPGAs frente a:

Lógica discreta

- ✓ **Tiempo de diseño** (cambios sin "soldar")
- ✓ **Densidad integración** (menor peso, tamaño)
- ✓ **Mayores prestaciones** (consumo, velocidad)
- ✓ **Menor coste** para circuitos medios-complejos
- × **Mayor coste** para circuitos muy simples
- × **Herramientas específicas** (fabricante)

ASICs

- ✓ **Tiempo de diseño** (cambios en el laboratorio)
- ✓ **Reconfigurable** (pruebas o actualizaciones)
- ✓ **Menor coste** para tiradas cortas y medias
- × **Mayor coste** para tiradas muy largas
- × **Menores prestaciones** (consumo, velocidad)
- × **Menor densidad integración**

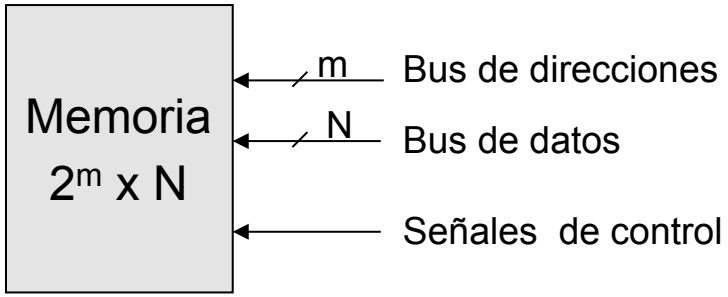
- **complejidad** +
- **número de unidades (tirada)** +
- **velocidad** +

Lógica discreta	FPGAs	ASICs

7.1. Memorias ROM

Una memoria es un dispositivo de almacenamiento de información

- RAM Random Acess Memory
- ROM Read-Only Memory
- PROM Programmable ROM
- EPROM Erasable and Programmable ROM
- EEPROM Electrically erasable PROM



Estructura interna:

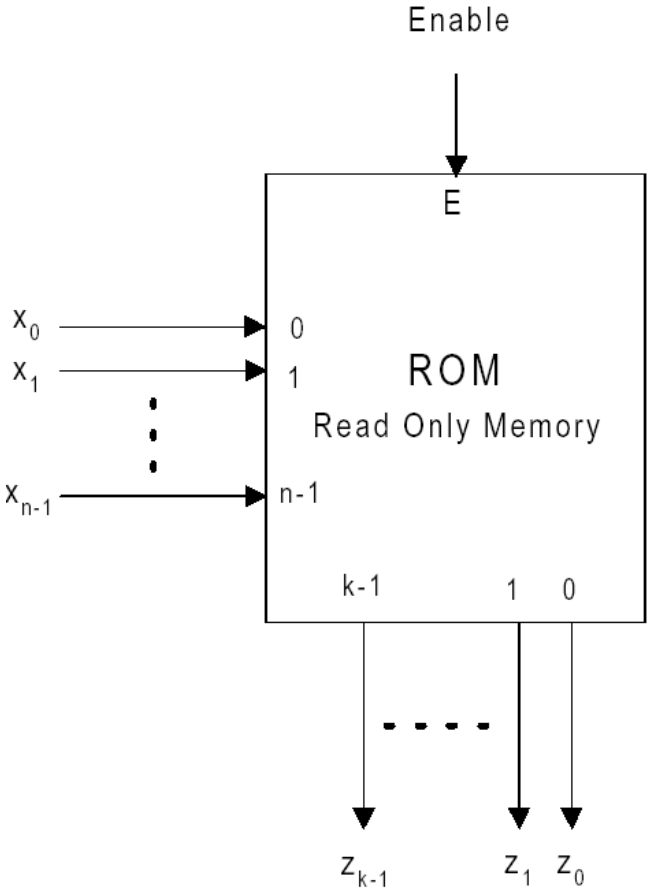
- Los datos se almacenan en grupos llamados posiciones de memoria. Cada posición de memoria tiene una dirección. Sólo se puede acceder a una dirección a la vez
- Las direcciones están codificadas y se indican a través el bus de direcciones
- Los datos son leídos o escritos por el bus de datos
- Capacidad de una memoria: M x N bits (M direcciones de N bits cada una)

Memorias ROM

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

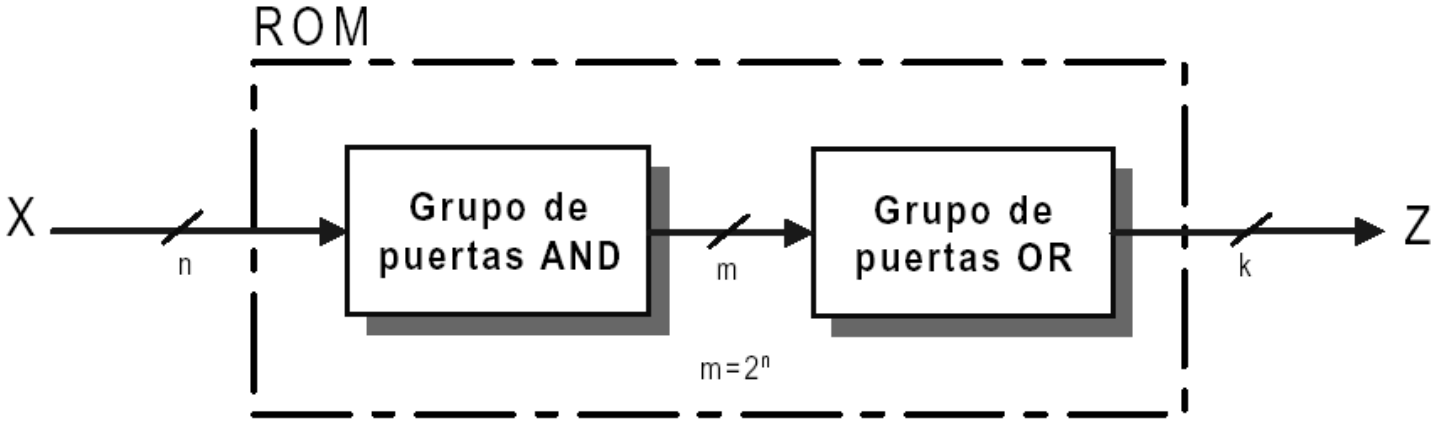
Una memoria ROM es un CI programable (por el fabricante o los usuarios) en el que se pueden personalizar ciertas conexiones.

Existen distintos tipos según los datos sean o no permanentes (ROM, PROM o EPROM, EEPROM), sean no programables o programables (**ROM** o **PROM**, **EPROM**, **EEPROM**) , y cuantas veces, y como se realice físicamente el borrado y la programación (EPROM - UV, EEPROM - eléctrica).



Memorias 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.

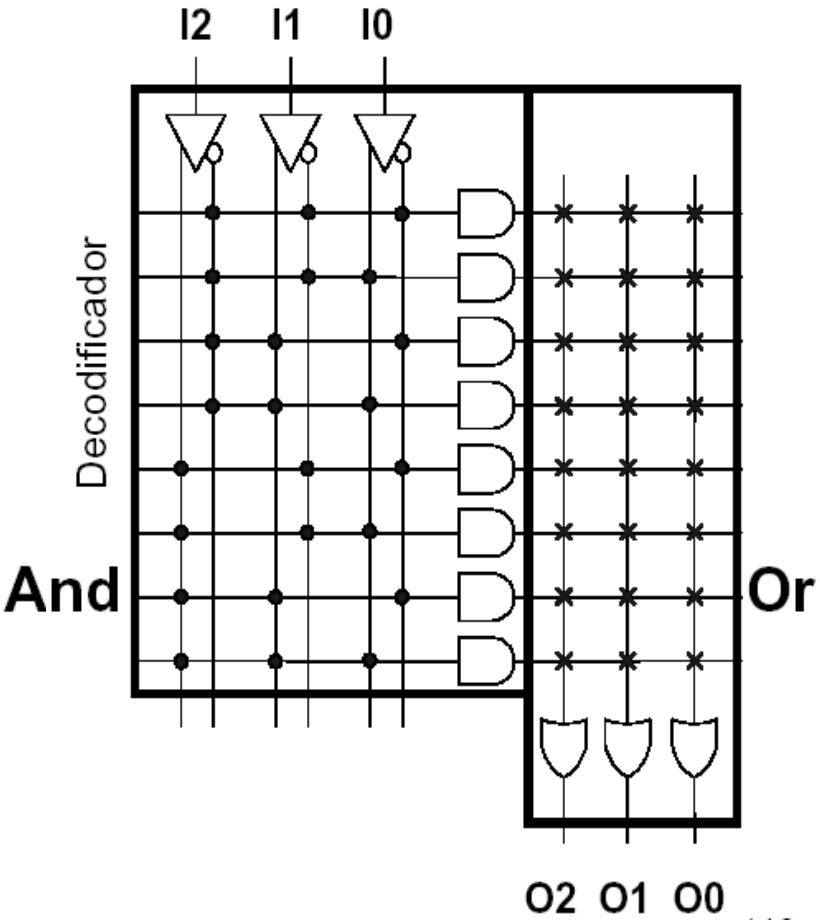


El grupo de puertas **AND están programadas de antemano** y conectadas de forma inalterable, mientras que el grupo de puertas **OR son programables por el usuario.**

Memorias ROM

El grupo de puertas AND se puede ya entender como un decodificador de n a 2^n con el que se generan todos los minterms para cualquier función de n variables (direcciones).

Ese decodificador (prefijado) junto a un grupo de puertas OR programables permite materializar cualquier FC de n variables (ver Síntesis de FC con decodificadores).



Memorias ROM

Ejemplo: Materializar el comparador de magnitud de dos palabras binarias de dos bits que cumple lo siguiente:

Z	$z_2\ z_1\ z_0$
IG	1 0 0
MA	0 1 0
ME	0 0 1

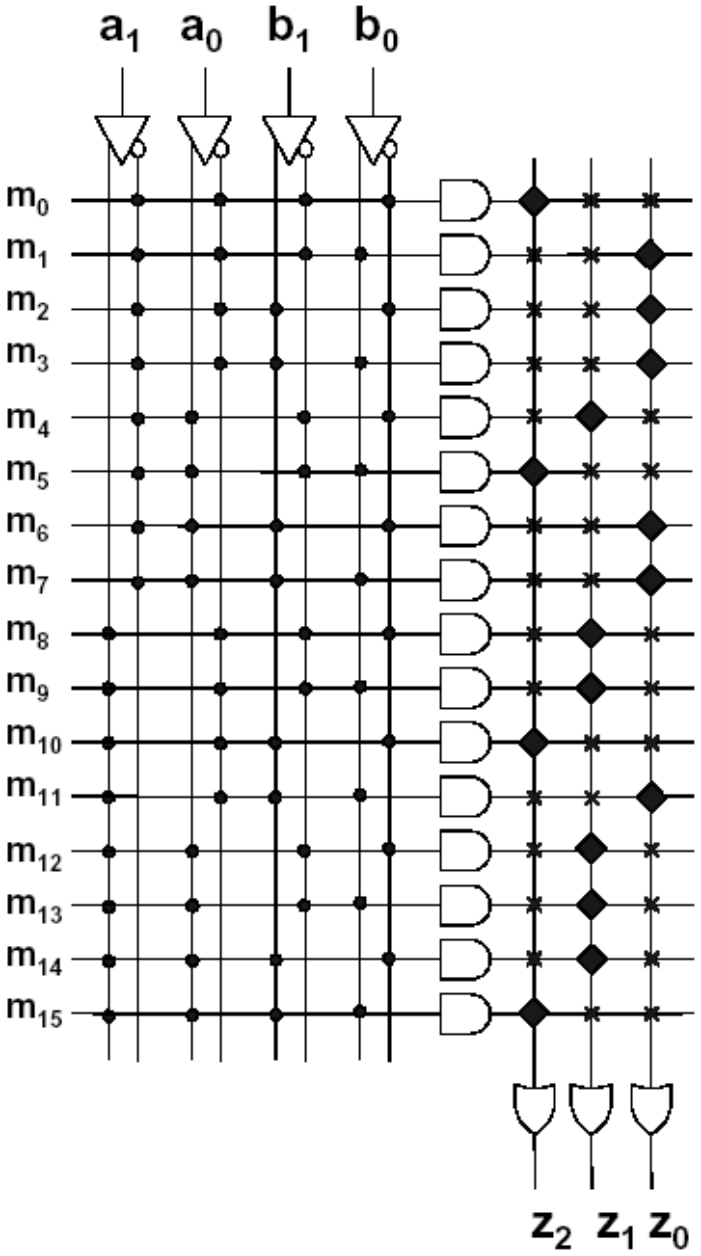
$$Z = \begin{cases} IG & \text{si } A = B \\ MA & \text{si } A > B \\ ME & \text{si } A < B \end{cases}$$

$$z_2 = \sum m(0,5,10,15)$$

$$z_1 = \sum m(4,8,9,12,13,14)$$

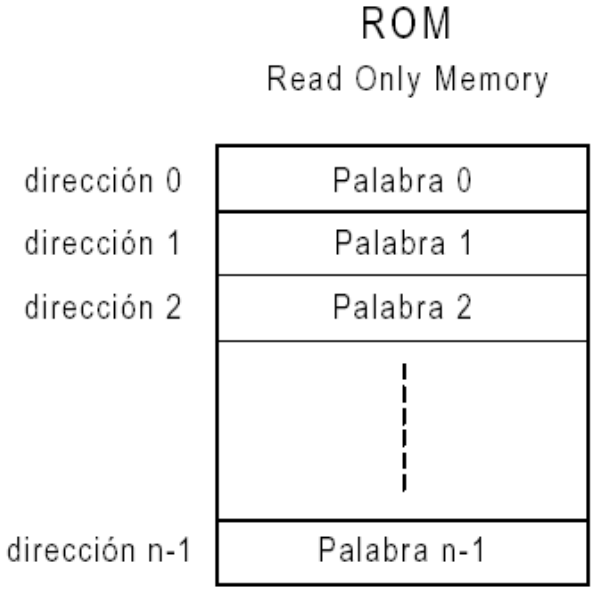
$$z_0 = \sum m(1,2,3,6,7,11)$$

La solución consiste en seleccionar las salidas que generan los minterms de las funciones y programar las conexiones en el grupo OR para cada una de las salidas. Se almacena directamente la tabla de verdad.



Memorias ROM

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

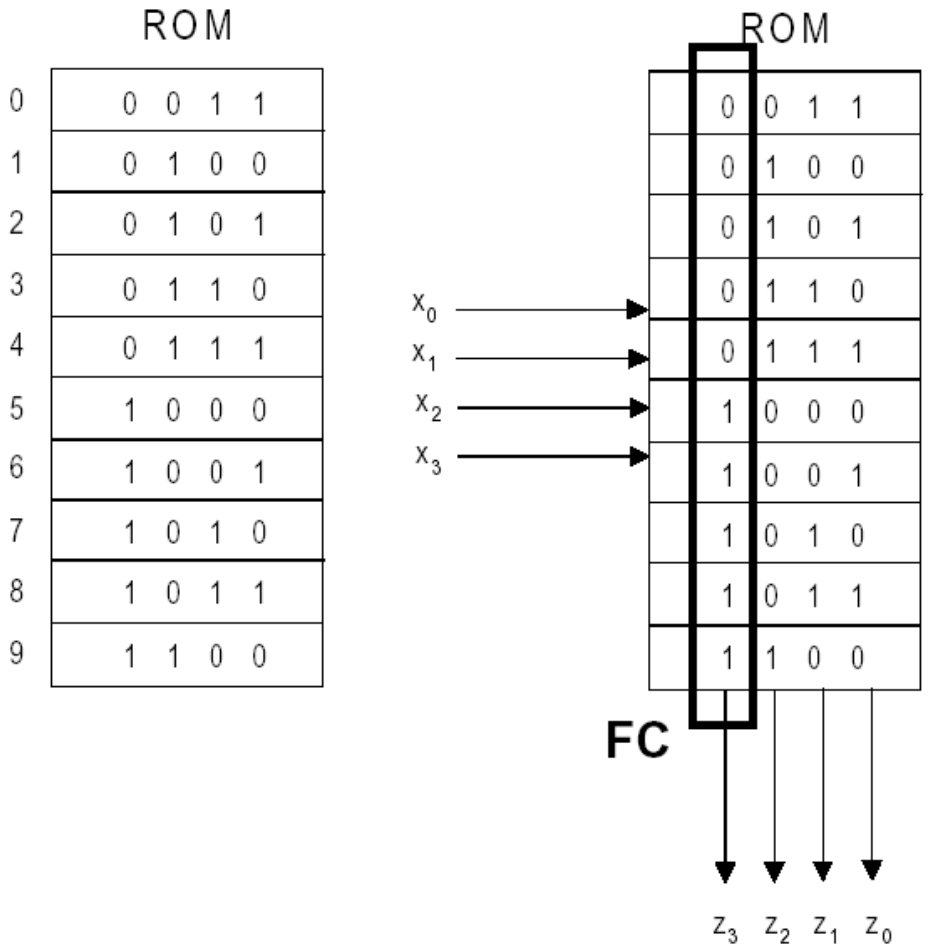


$$ROM[\text{dirección } i] = \text{Palabra } i$$

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

Memorias ROM

Ejemplo: Diseñar en ROM un conversor de código BCD a XS-3

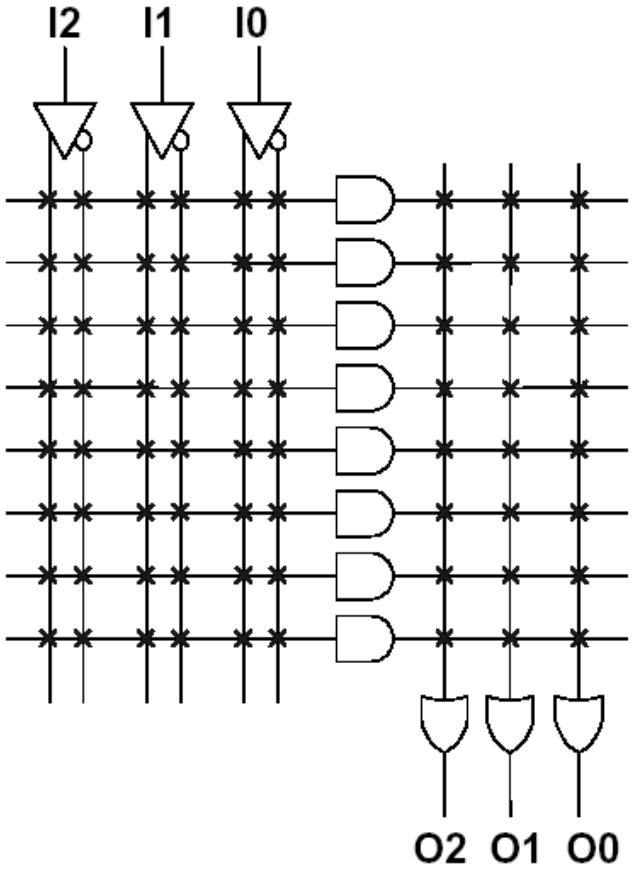


7.2. Dispositivos programables PLA,PAL

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 con conexiones programables tanto en el grupo de puertas AND como en el grupo de puertas OR.

Las PALs son un caso particular de las PLA con conexiones OR preprogramadas.



8. Comparadores

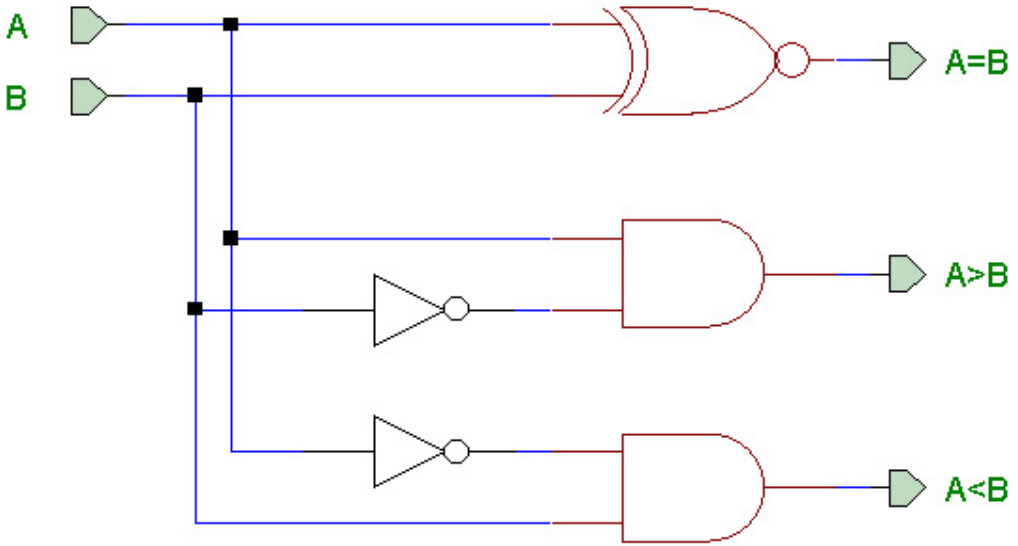
Son sistemas combinacionales que reciben dos datos de entrada A y B y los comparan en binario puro, devolviendo 3 salidas que indican si A>B, A=B ó A<B

Comparador de 1 bit

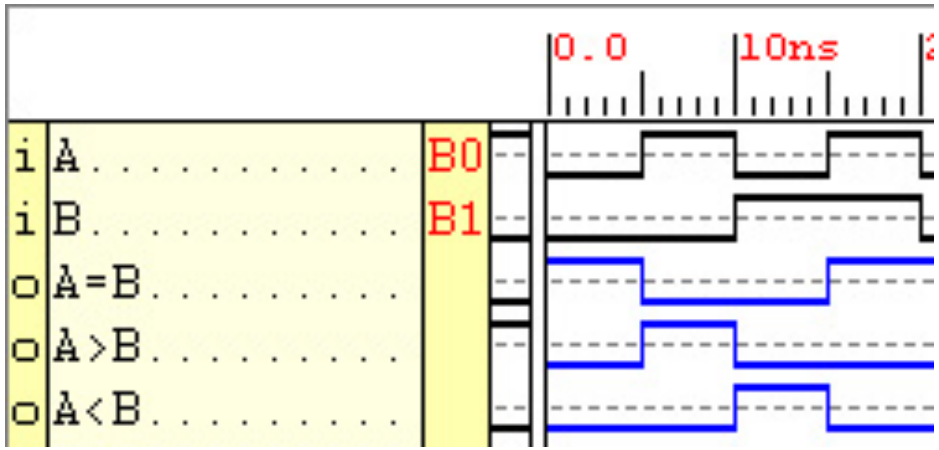
Tabla de verdad

A	B	I_0 (A=B)	M_0(A>B)	m_0(A<B)
L	L	H	L	L
L	H	L	L	H
H	L	L	H	L
H	H	H	L	L

Circuito lógico



Cronograma



Símbolo

Comparadores

Comparador de 2 bits

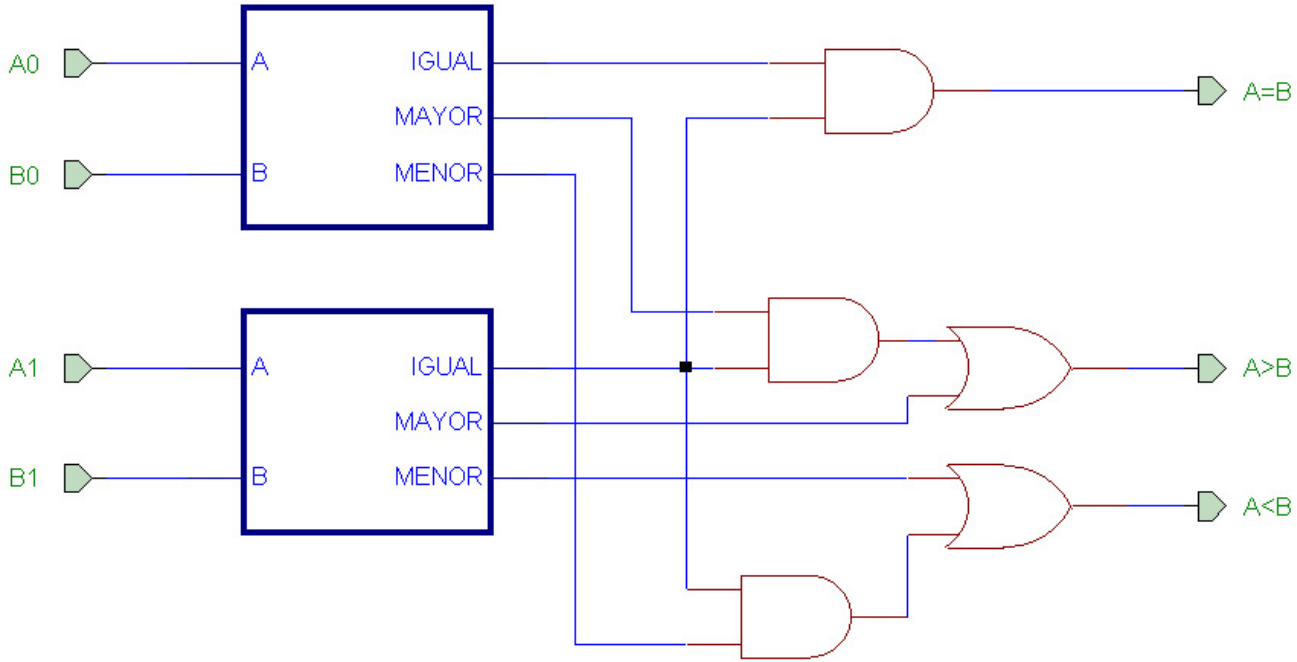
Ecuaciones

$$M_{10} = M_1 + I_1 \cdot M_0$$

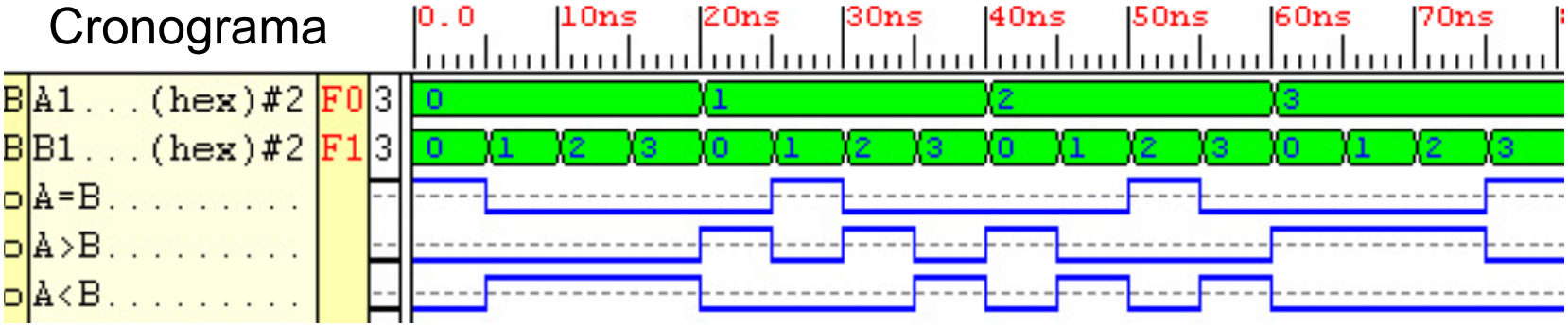
$$m_{10} = m_1 + I_1 \cdot m_0$$

$$I_{10} = I_1 \cdot I_0$$

Circuito lógico



Cronograma



Comparadores conectables en cascada

Estos comparadores disponen de tres entradas que permiten introducir en un comparador las salidas de otro que compara bits de menor peso. El resultado final de la comparación lo dan las salidas del comparador de los bits de mayor peso.

Tabla de verdad de un comparador conectable en cascada para números de 4 bits

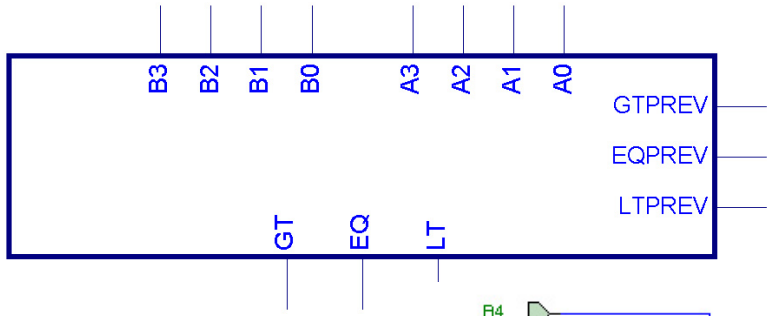
COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
				A>B	A<B	A=B	A>B	A<B	A=B
A3>B3	X	X	X	X	X	X	H	L	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2=B2	A1>B1	X	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	L	H	H	L
				X	X	H	L	L	H
				L	H	L	L	H	L
				H	L	L	H	L	L
				H	H	L	L	L	L
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	L	H	L
A3=B3	A2=B2	A1<B1	X	X	X	X	L	H	L
A3=B3	A2<B2	X	X	X	X	X	L	H	L
A3<B3	X	X	X	X	X	X	L	H	L

Comparadores conectables en cascada

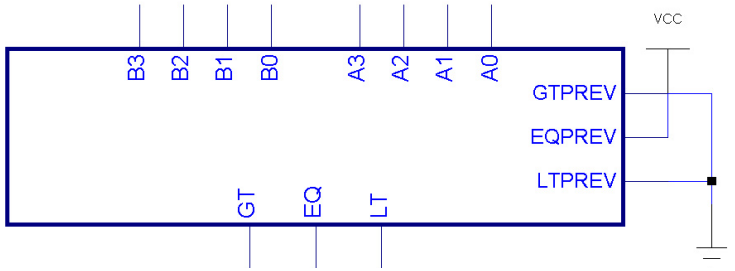
Comercial: 74HC85

Disponen de tres entradas que permiten introducir en un comparador las salidas de otro que compara bits de menor peso. El resultado final de la comparación lo dan las salidas del comparador de los bits de mayor peso.

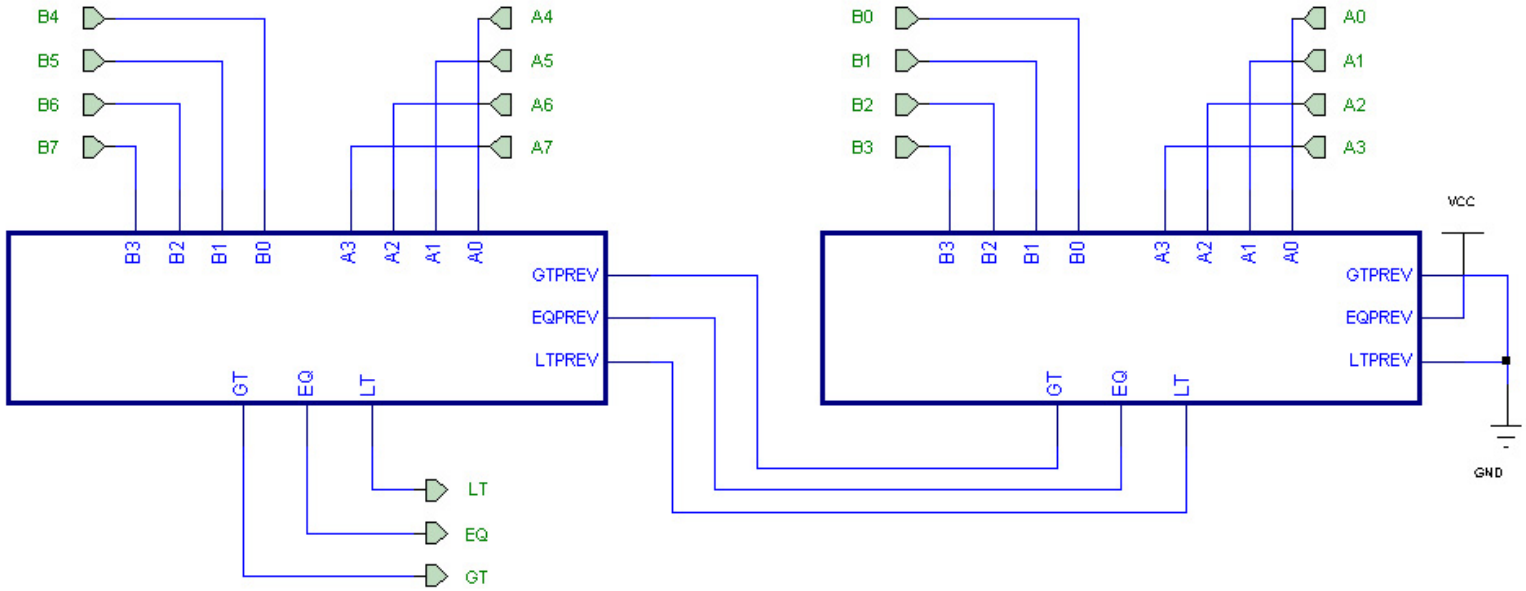
Símbolo



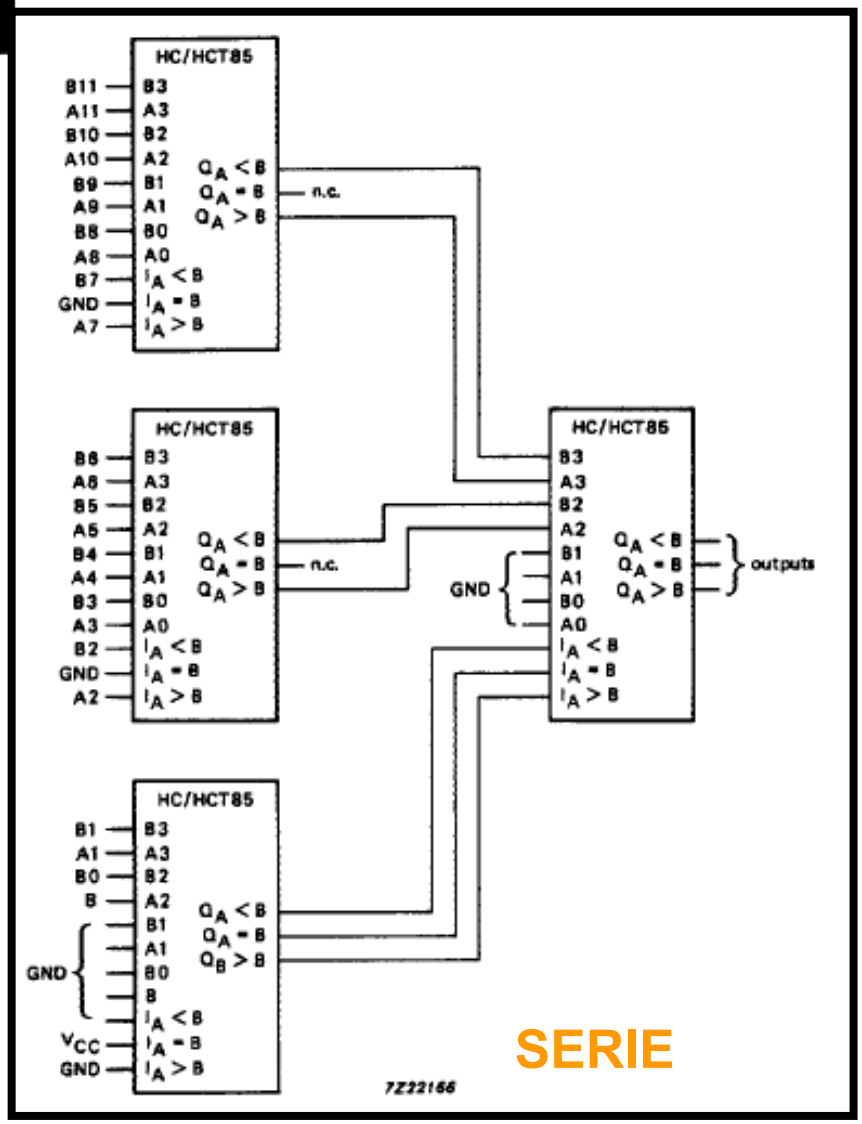
Comparador de 4 bits

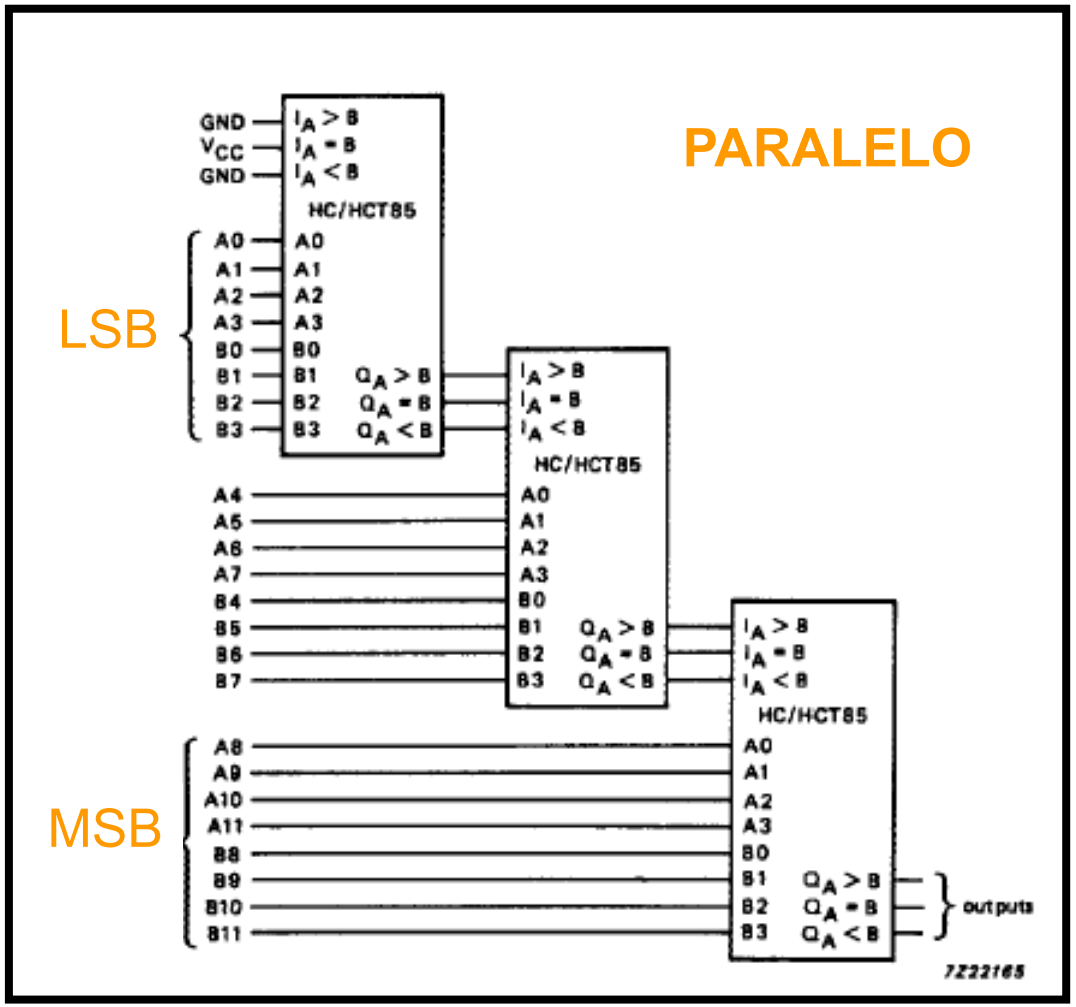


Conexión de dos comparadores en serie



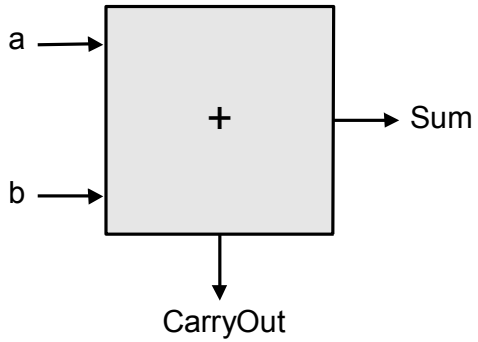
Comparadores conectables en cascada





9. Sumadores: semisumador elemental

El **semisumador (*half adder*)** es un circuito que suma dos bits de entrada a_i y b_i y devuelve un bit de resultado z_i y un bit de acarreo c_i.



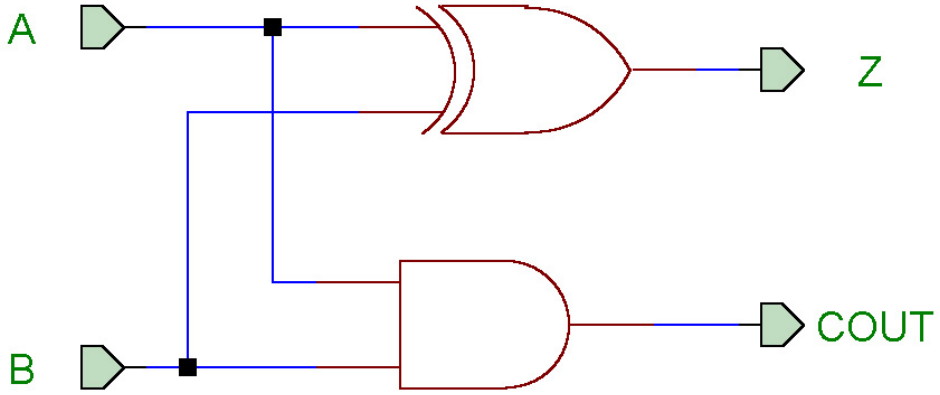
$c_{out} = a \cdot b$

$Sum = a \oplus b$

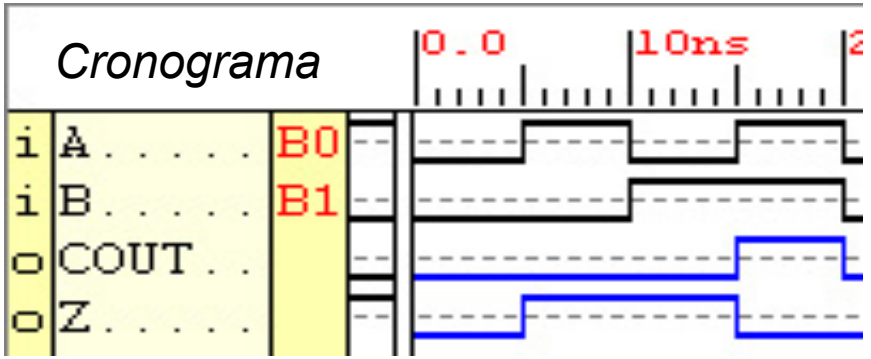
Tabla de verdad

A_i	B_i	C_i	Z_i
L	L	L	L
L	H	L	H
H	L	L	H
H	H	H	L

Circuito con puertas lógicas

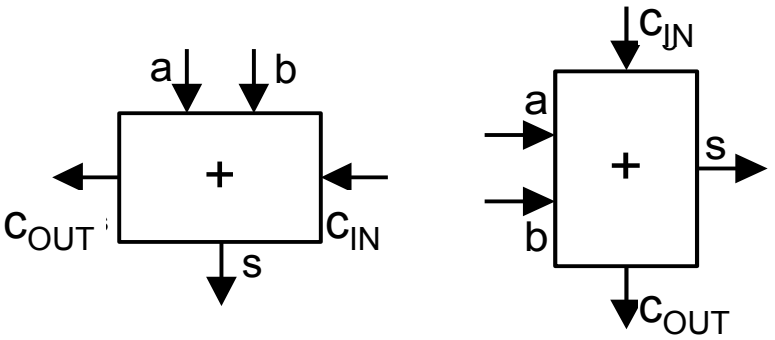


Cronograma



Sumadores: sumador elemental completo

El **sumador completo (*full adder*)** es un circuito que suma dos bits de entrada a_i y b_i más un acarreo de entrada c_{i-1} y devuelve un bit de resultado z_i y un bit de acarreo c_i.

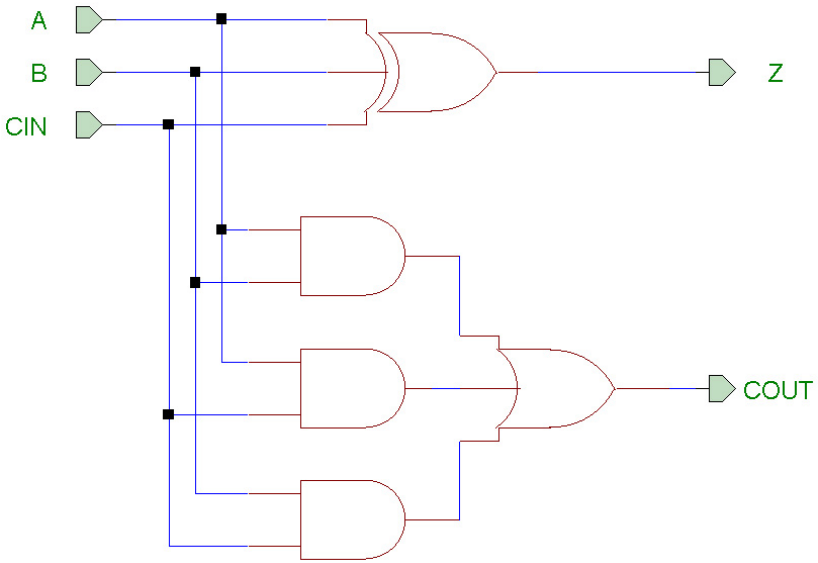


$$c_{OUT} = a \cdot b + a \cdot c_{IN} + b \cdot c_{IN}$$

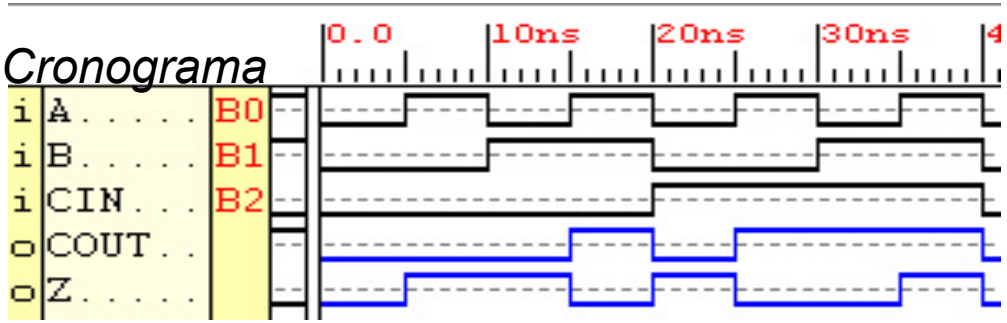
$$s = a \oplus b \oplus c_{IN}$$

Tabla de verdad

A_i	B_i	C_{i-1}	C_i	Z_i
L	L	L	L	L
L	L	H	L	H
L	H	L	L	H
L	H	H	H	L
H	L	L	L	H
H	L	H	H	L
H	H	L	H	L
H	H	H	H	H



Cronograma



Sumadores:

Comercial: 74283
(4 bits, acarreo anticipado)

Sumador paralelo con acarreo en serie

En los ejemplos anteriores el acarreo se va propagando consecutivamente de un sumador a otro.

Sumador paralelo con acarreo anticipado (“carry look ahead”)

Se usa una estrategia combinacional (“generate”, ”propagate”) para adelantar la salida del acarreo en una etapa y reducir su tiempo de propagación, a partir del conocimiento de todas las entradas. Requiere varios niveles de lógica, compensa el retardo en sumas de muchos bits.

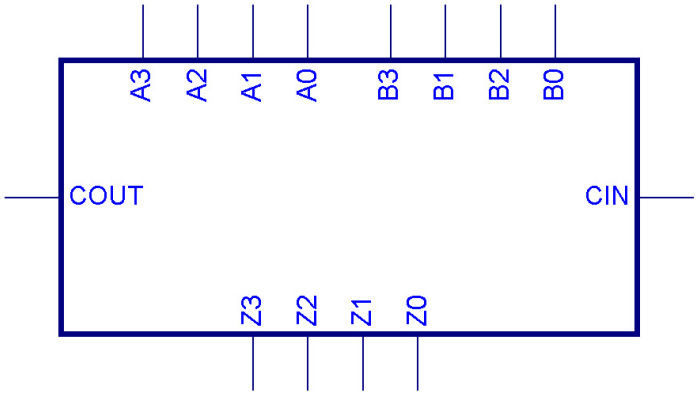
Los integrados 74HC283 y 74LS283 son sumadores de dos nºs de 4 bits, e incorporan un diseño de acarreo anticipado.

Extensión de sumadores a nºs de más bits.

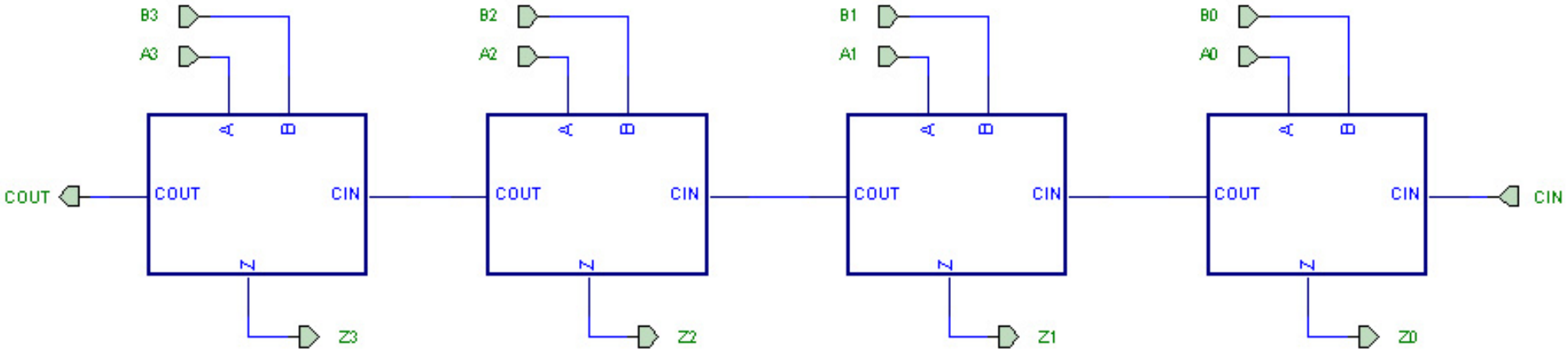
Un sumador de números de 8 bits se obtiene fácilmente conectando el acarreo de salida (C4) del sumador de 4 bits al de entrada (C0) del otro.

Sumador con propagación de acarreo en serie

Se construye asociando n sumadores elementales completos (full adder) que reciben y procesan todos ellos los datos en paralelo, si bien el acarreo se propaga en serie de un sumador a otro (Circuito lento)



Circuito con sumadores elementales



Sumador con propagación de acarreo en serie

El retardo del sumador elemental es:

- Z: 1 puerta.
- COUT: 2 puertas.

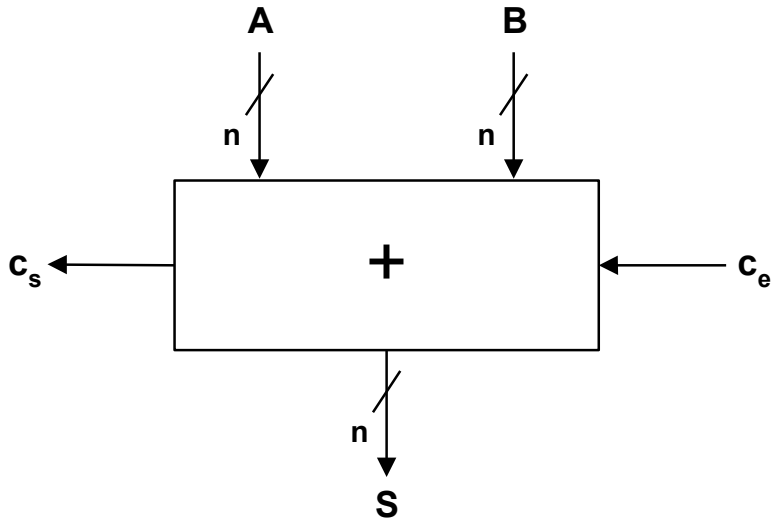
El retardo del sumador serie con acarreo propagado es muy significativo.

Ejemplo: sumar A=1011 y B=0101.

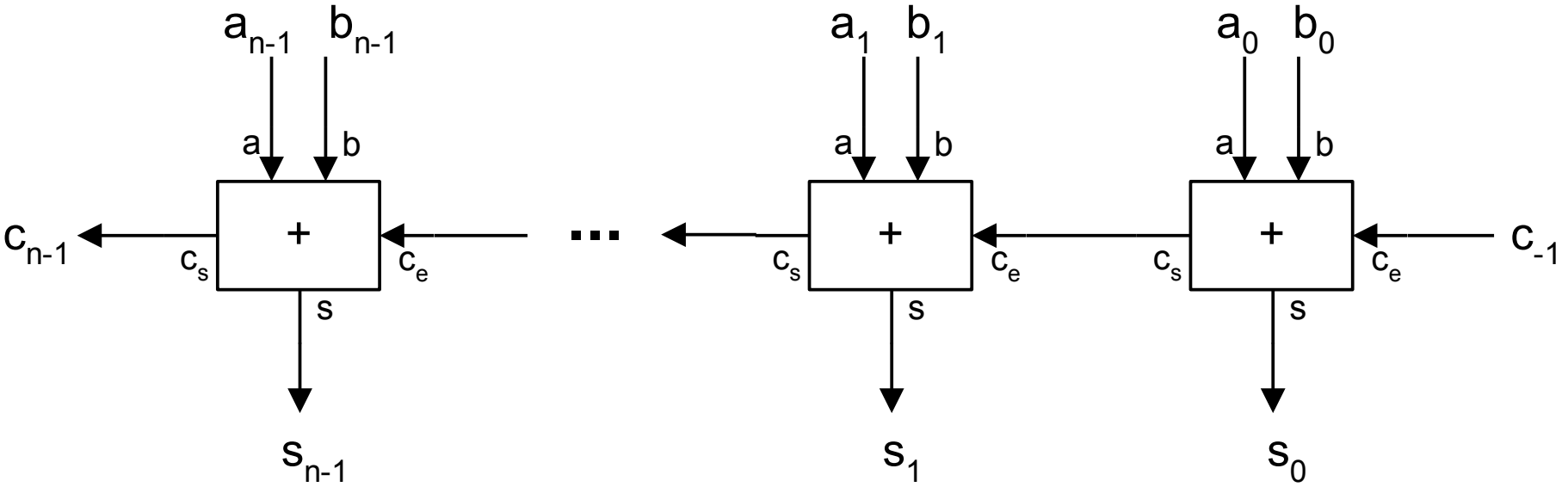
El resultado es Z=0000 con COUT=1, y se produce un acarreo en el primer sumador que se va propagando hasta el último. Los retardos son:

- retardo(Z_0) = 1
- retardo(C_0) = 2
- retardo(Z_1) = 1 + retardo(C_0) = 3
- retardo(C_1) = 2 + retardo(C_0) = 4
- retardo(Z_2) = 1 + retardo(C_1) = 1+4 = 5
- retardo(C_2) = 2 + retardo(C_1) = 2+4 = 6
- retardo(Z_3) = 1 + retardo(C_2) = 1+6 = 7
- retardo(C_3) = 2 + retardo(C_2) = 2+6 = 8

Sumador de n bits con propagación de acarreo en serie



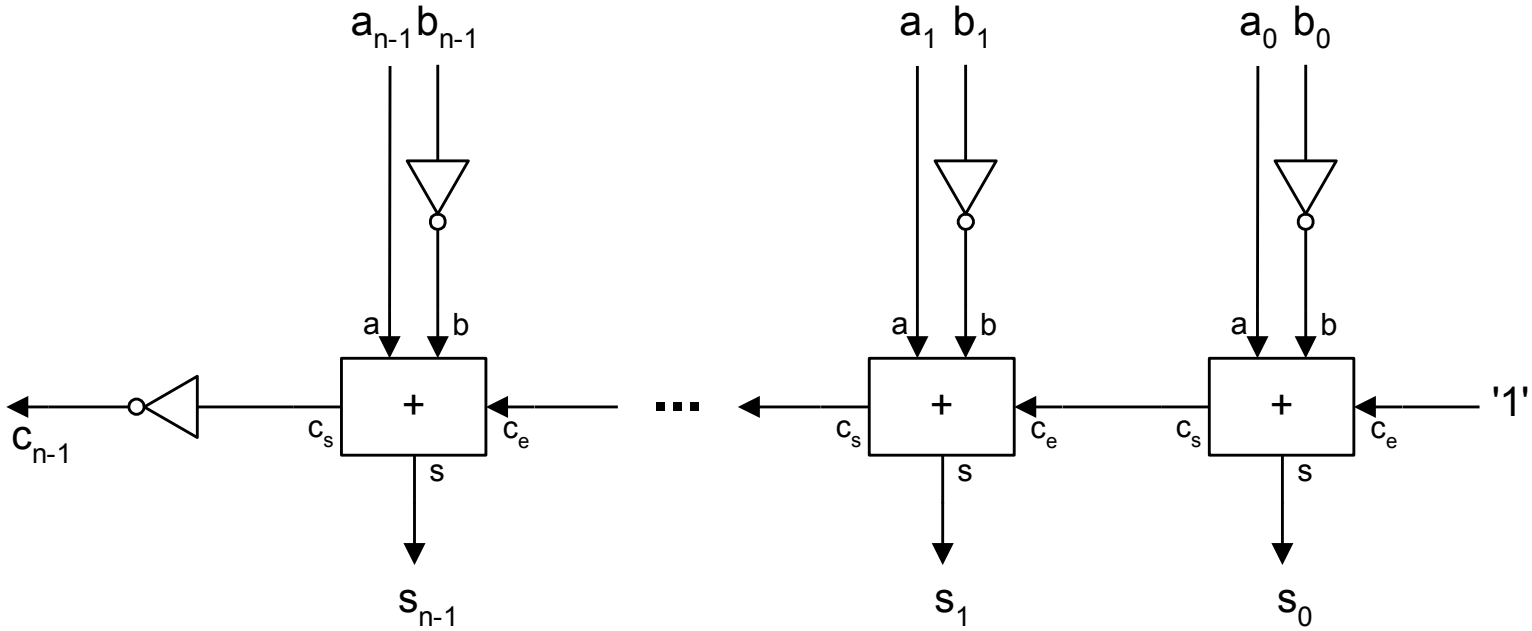
Circuito con sumadores elementales



10. Restadores binarios

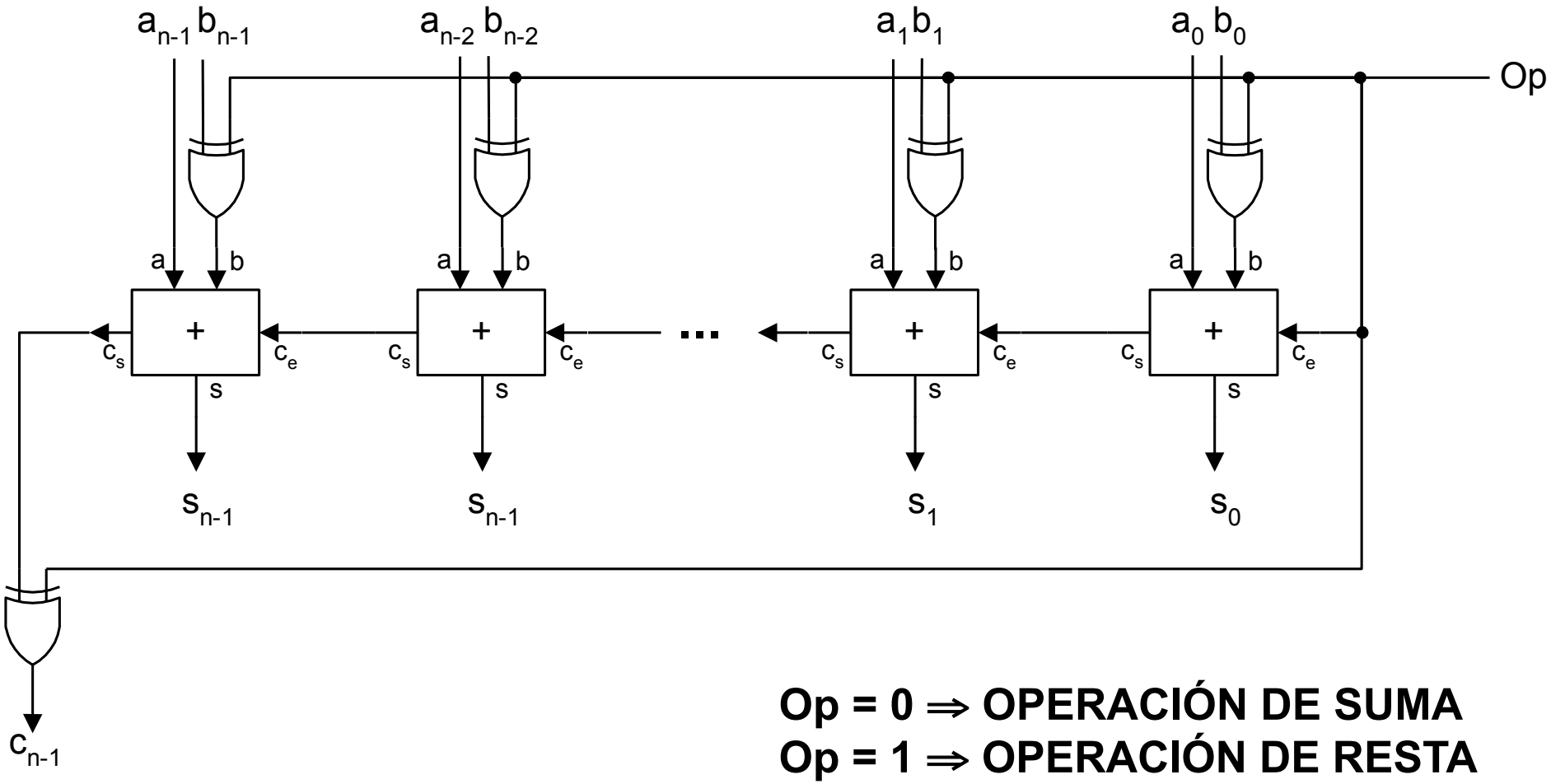
Para restar dos números binarios de *n* bits podemos hacer una suma del minuendo con el complemento a 2 del sustraendo:

- ➲ Para complementar el sustraendo, invertimos todos sus bits e introducimos un 1 en el acarreo de entrada del sumador menos significativo.
- ➲ Por este procedimiento también había que invertir el acarreo de salida.
- ➲ Esto funciona tanto para binario puro como para complemento a 2 (en complemento a 2 el acarreo se desprecia, y habría que detectar el posible desbordamiento de otro modo).



Sumar y restar números binarios

Podemos unir los circuitos anteriores y construir uno que haga sumas y restas en función de una señal de control ⇒ SUMADOR / RESTADOR DE N BITS.



Op = 0 ⇒ OPERACIÓN DE SUMA
Op = 1 ⇒ OPERACIÓN DE RESTA

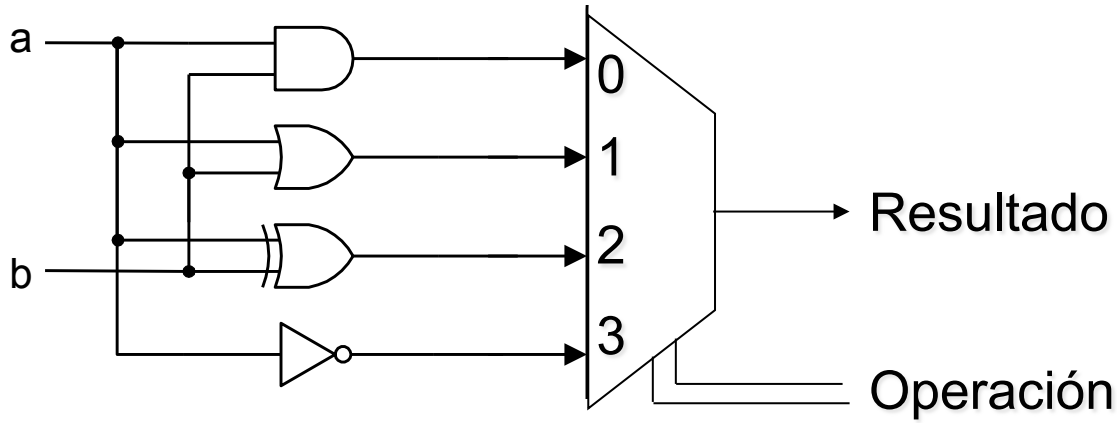
11. Unidad Aritmético Lógica Combinacional

Una unidad aritmética y lógica (UAL, ALU) es un circuito combinacional que realiza las operaciones aritméticas y lógicas básicas en el computador.

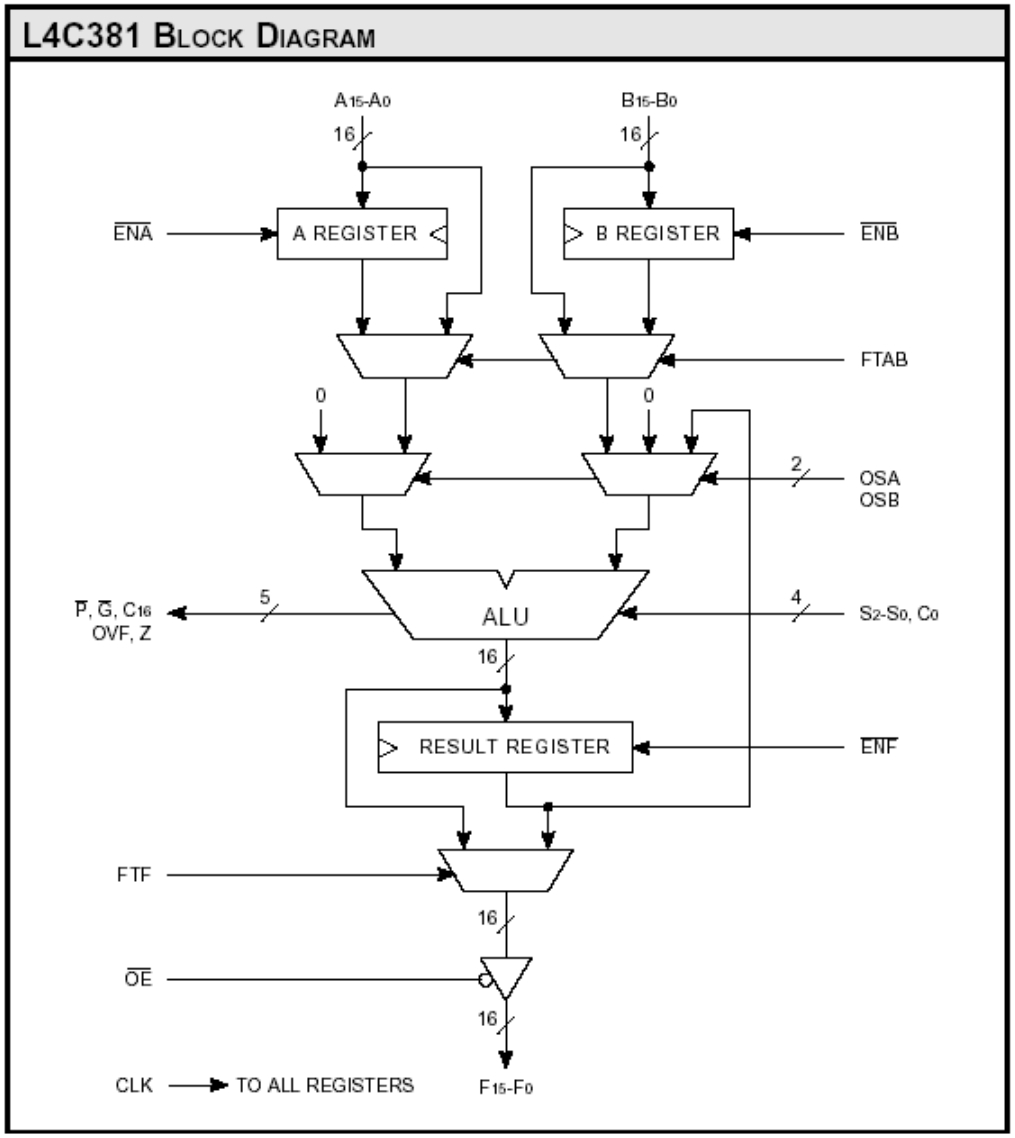
- ➲ Operaciones aritméticas básicas: suma y resta de enteros y desplazamientos unitarios.
- ➲ Operaciones lógicas básicas: NOT, AND, OR, EXOR, NAND, NOR.

Las implementación de circuitos para operaciones lógicas es muy sencilla: basta simplemente con una batería de puertas lógicas y un multiplexor accionado por las correspondientes señales de selección.

Ejemplo: circuito para realizar AND y OR, AND y XOR para dos datos de un bit.



Unidad Aritmético Lógica Combinacional



FEATURES

- High-Speed (15ns), Low Power 16-bit Cascadable ALU
- Implements Add, Subtract, Accumulate, Two's Complement, Pass, and Logic Operations
- All Registers Have a Bypass Path for Complete Flexibility
- 68-pin PLCC, J-Lead

Table 1. ALU Functions

S_2-S_0	FUNCTION
000	CLEAR (F = 00 ••• 00)
001	NOT(A) + B
010	A + NOT(B)
011	A + B
100	A XOR B
101	A OR B
110	A AND B
111	PRESET (F = 11 ••• 11)

Otro ejemplo: IDT7381