



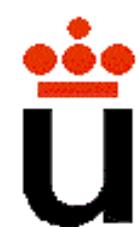
Universidad

Rey Juan Carlos

Bloques combinacionales estándar

Área de Tecnología Electrónica

Ingeniería de Sistemas Audiovisuales y Multimedia



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

Los dispositivos electrónicos más elementales son **las puertas lógicas** y **los bloques lógicos**, que forman los **circuitos lógicos**. Estos últimos se pueden ver como un conjunto de dispositivos que manipulan de una manera determinada las señales electrónicas que les llegan (las señales de entrada), y generan como resultado otro conjunto de señales (las señales de salida).

Existen dos grandes tipos de circuitos lógicos:

- 1) **Los circuitos combinacionales:** que se caracterizan porque el valor de las señales de salida en un momento determinado depende del valor de las señales de entrada en ese mismo momento.
- 2) **Los circuitos secuenciales:** en los que el valor de las señales de salida en un momento determinado depende de los valores que han llegado por las señales de entrada desde la puesta en funcionamiento del circuito. Por tanto, tienen capacidad de memoria.



1. Introducción a los circuitos combinacionales

Objetivos: Conocer a fondo los circuitos lógicos combinacionales, es decir, saber cómo están formados y ser capaces de utilizarlos con agilidad, hasta el punto de familiarizarnos totalmente con ellos.

- ✓ Entender el álgebra de Boole y las diferentes maneras de expresar funciones lógicas.
- ✓ Conocer las diferentes puertas lógicas, ver cómo se pueden utilizar para sintetizar funciones lógicas y ser capaces de hacerlo. Entender por qué se desea minimizar el número de puertas de los circuitos y saberlo hacer.
- Conocer la funcionalidad de un conjunto de bloques combinacionales básicos y ser capaces de utilizarlos en el diseño de circuitos.

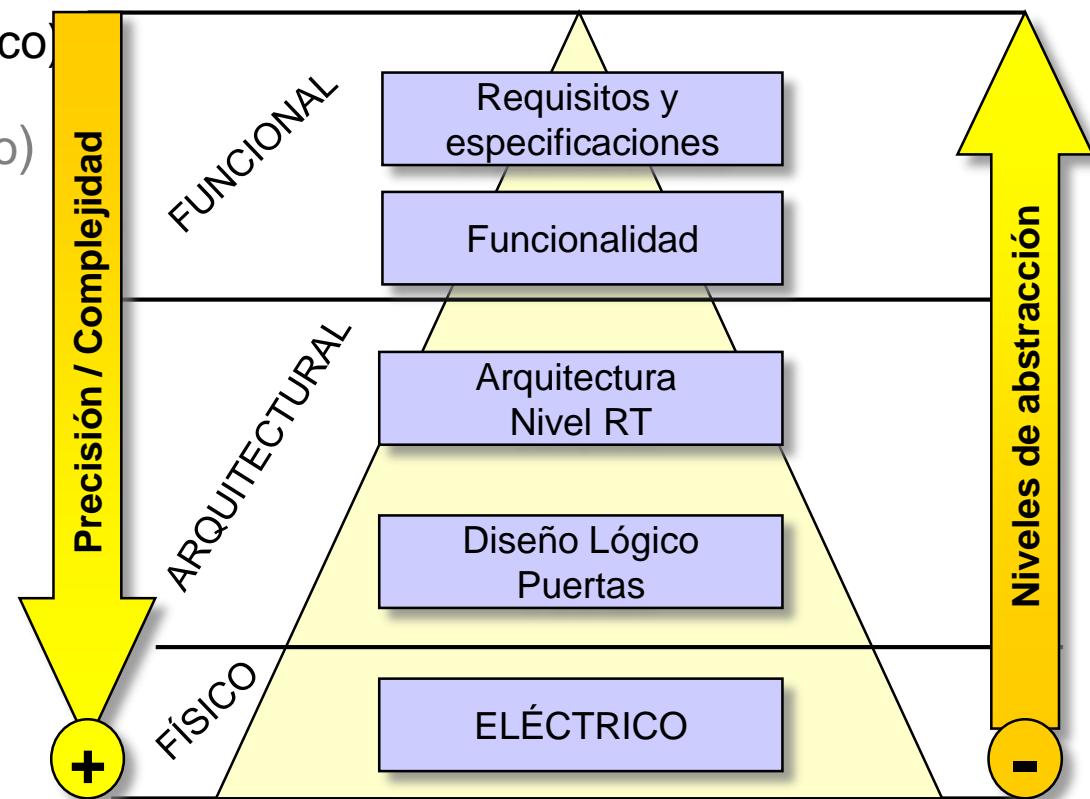
En definitiva, después del estudio de este tema debemos ser capaces de construir fácilmente un circuito cualquiera usando los diferentes dispositivos que se habrán conocido y entender la funcionalidad de cualquier circuito dado.



1. 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 (circital y eléctrico)

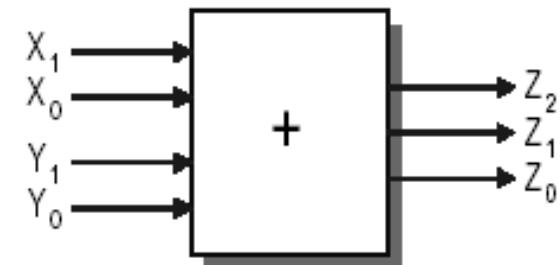
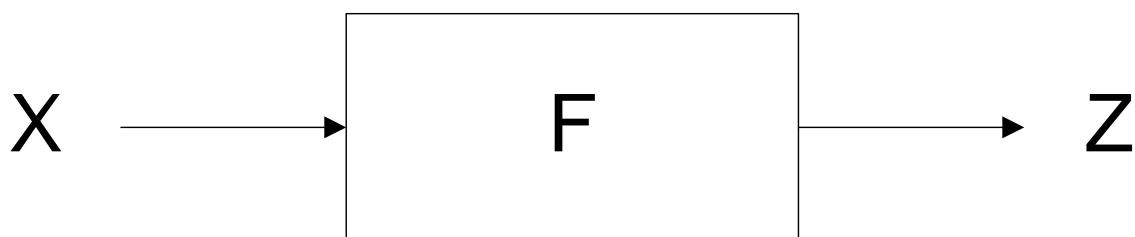




1. Introducción a los circuitos combinacionales

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**

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



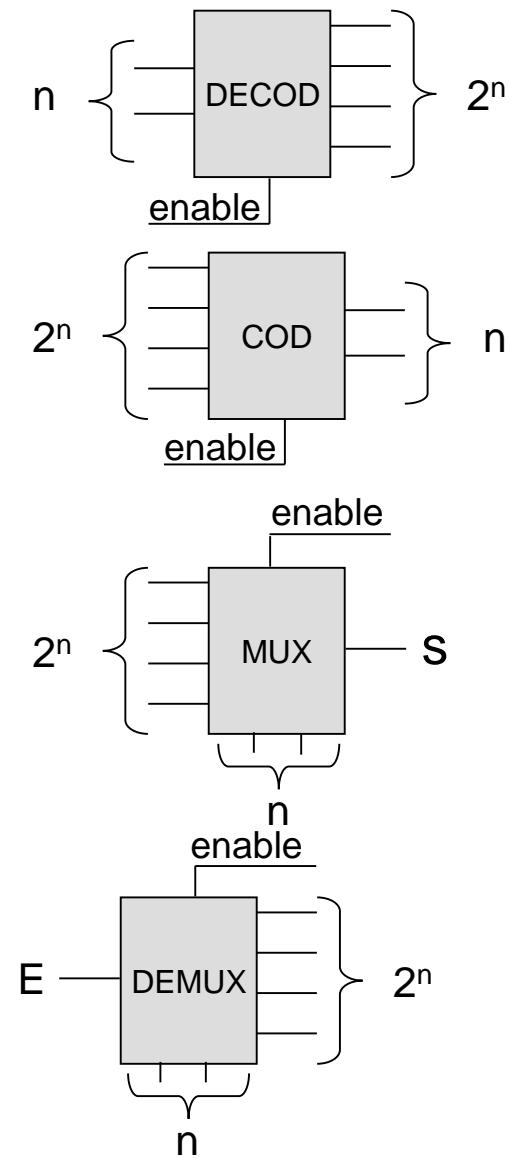
Ejemplo: Sumador



1. 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

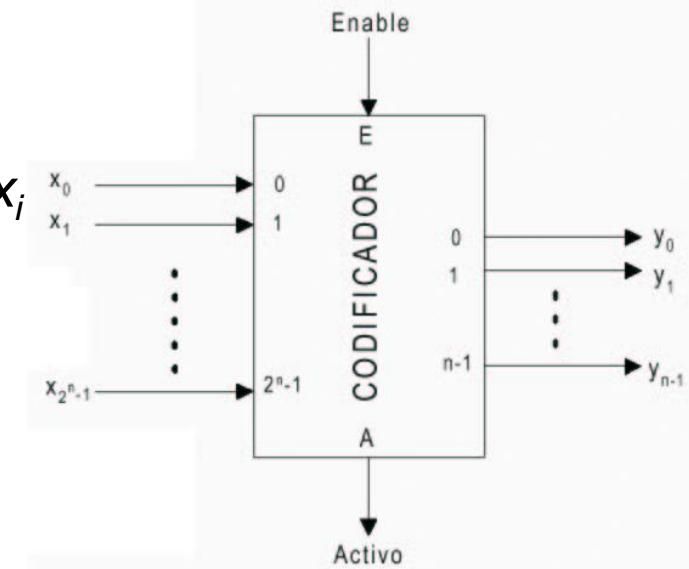


2. Codificadores

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.

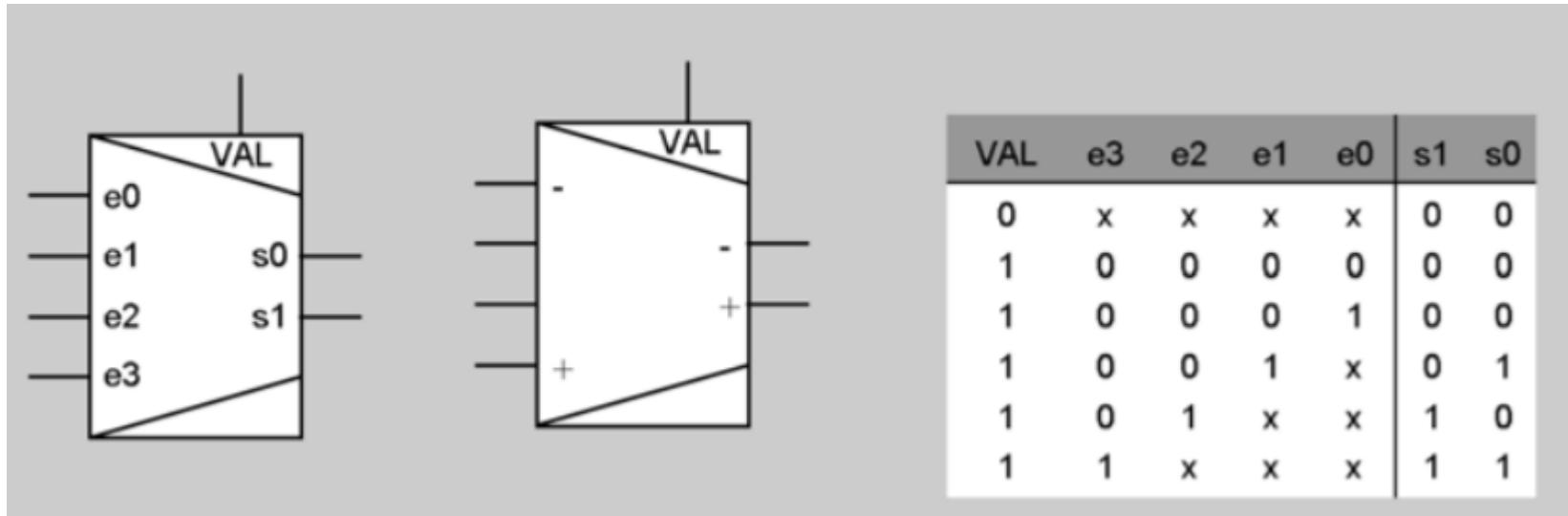


Expresión de conmutación:

$$y_i = E \cdot \sum x_j$$

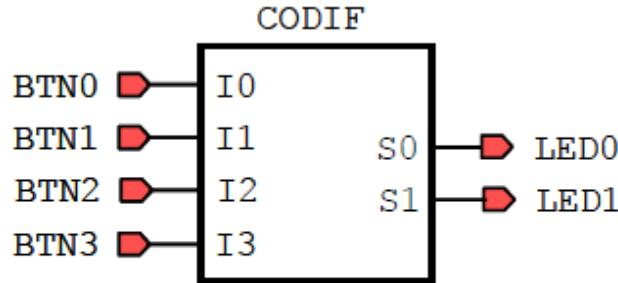


Codificadores sin prioridad: ejemplo Cod4a2





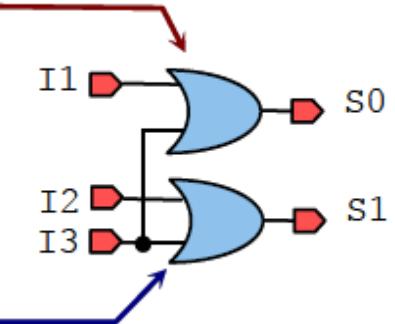
Codificadores sin prioridad: ejemplo Cod4a2



entrada activa	salidas	
	S1	S0
I0	0	0
I1	0	1
I2	1	0
I3	1	1

S0 se enciende cuando **I1** está activa o (**OR**) cuando **I3** está activa

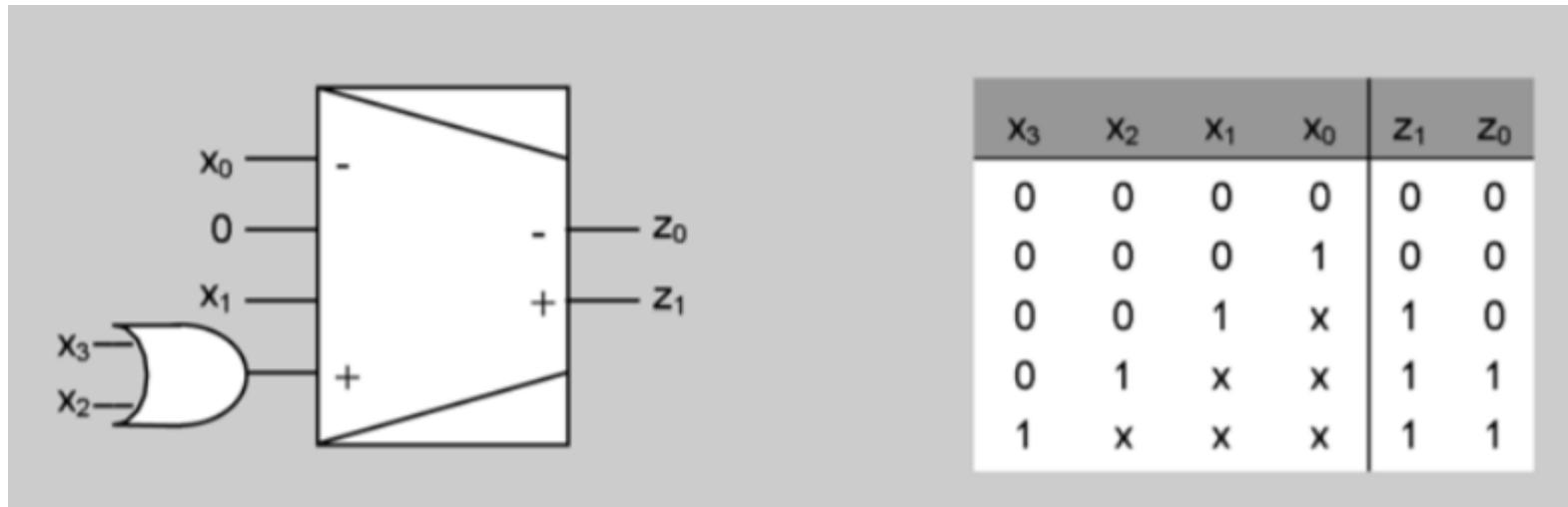
S1 se enciende cuando **I2** está activa o (**OR**) cuando **I3** está activa



Codificador de 4 a 2
Sin prioridad



Codificadores sin prioridad: otro ejemplo Cod4a2





Codificadores sin prioridad

No tiene sentido

entrada activa	salidas	
	S1	S0
--	0	0
I0	0	0
I1	0	1
I2	1	0
I3	1	1

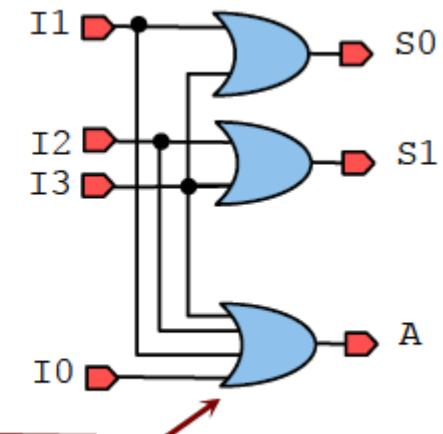
Solución



entrada activa	salidas		
	S1	S0	A
--	0	0	0
I0	0	0	1
I1	0	1	1
I2	1	0	1
I3	1	1	1

Diferencia entre pulsar I0 y no pulsar nada

A se enciende cuando cualquier entrada esté activa:
 $I0 + I1 + I2 + I3$



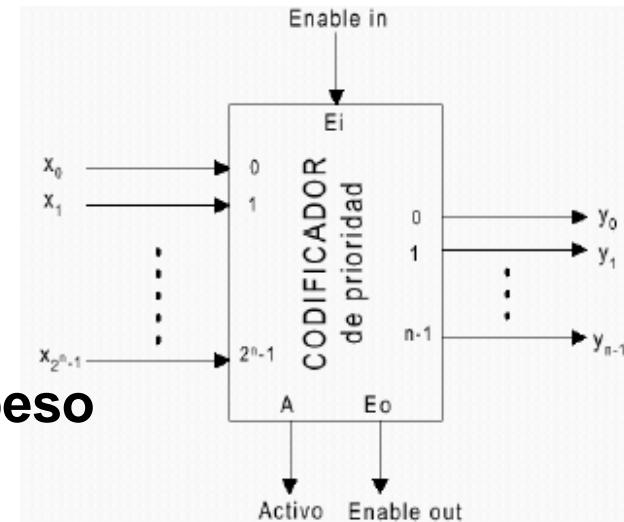


Codificadores con prioridad

¿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**



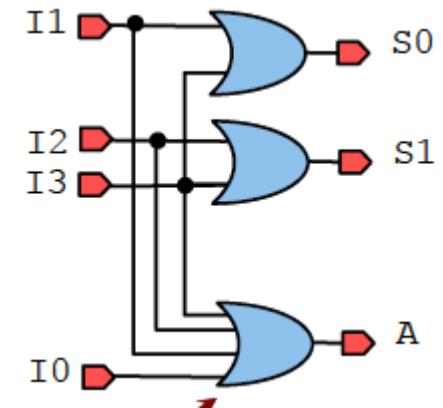
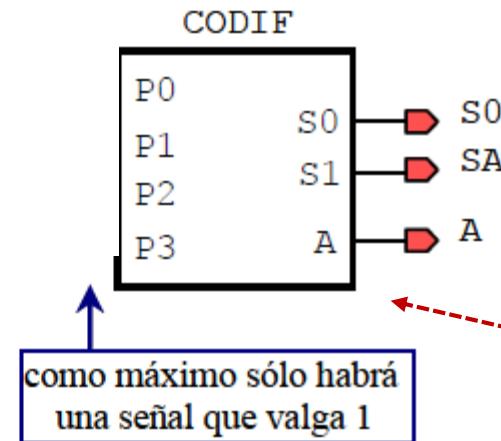
Los codificadores pueden encadenarse para formar codificadores con mayor número de bits.

E_{out} se activa cuando **E_{in}** 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.

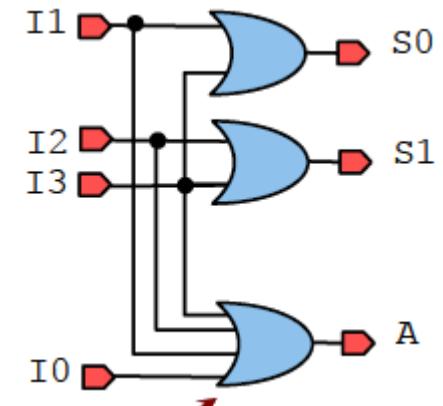
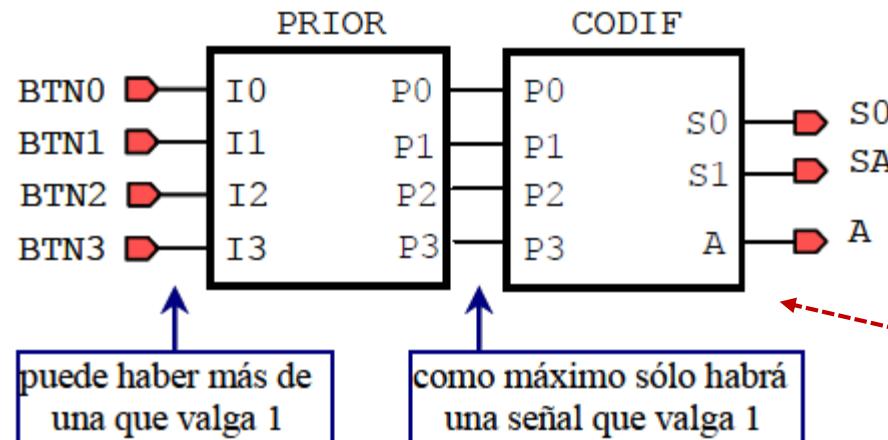


Codificadores con prioridad: ejemplo Cod4a2



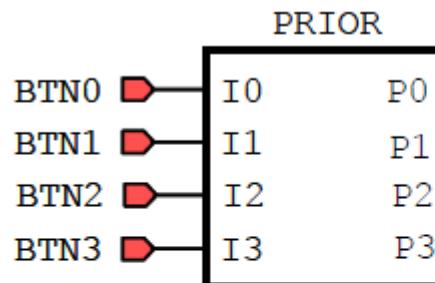


Codificadores con prioridad: ejemplo Cod4a2



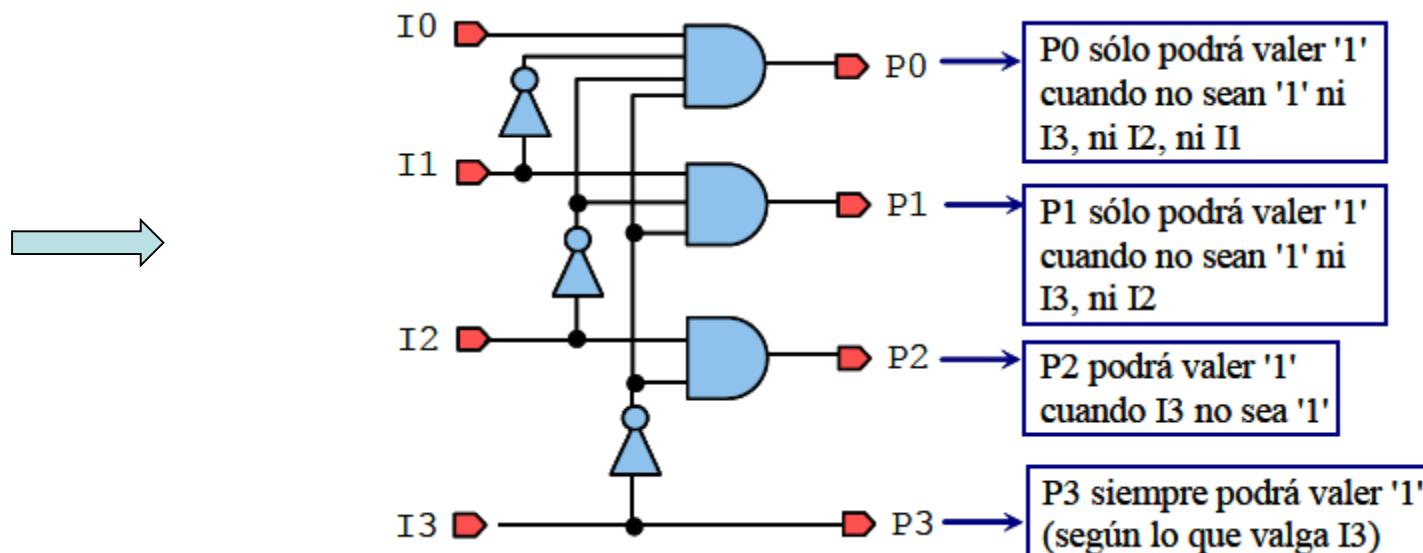


Codificadores con prioridad: ejemplo Cod4a2



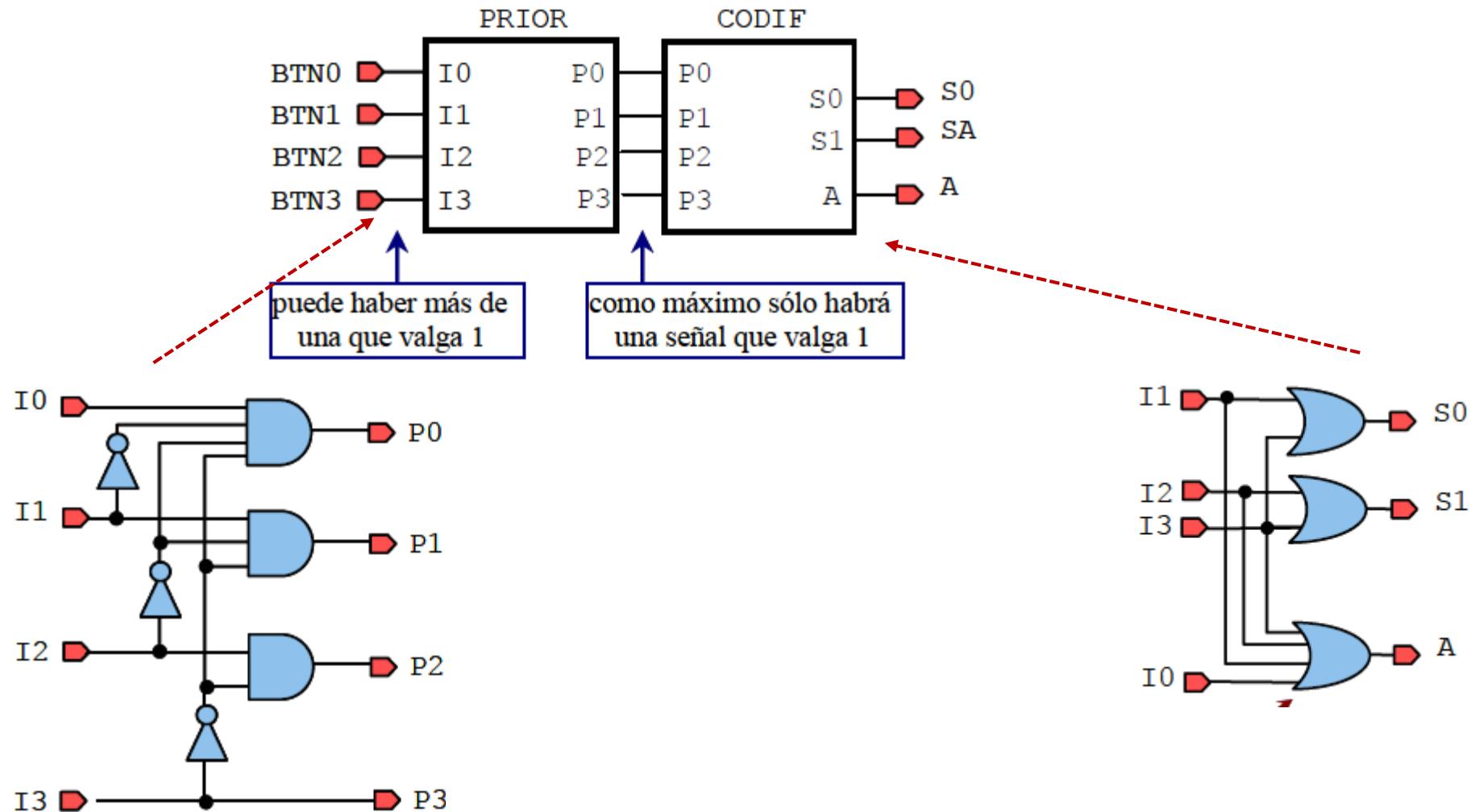
I3	I2	I1	I0	P3	P2	P1	P0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

independientemente de lo que valga I0,
P0 = '0' porque I3='1'





Codificadores con prioridad: ejemplo Cod4a2

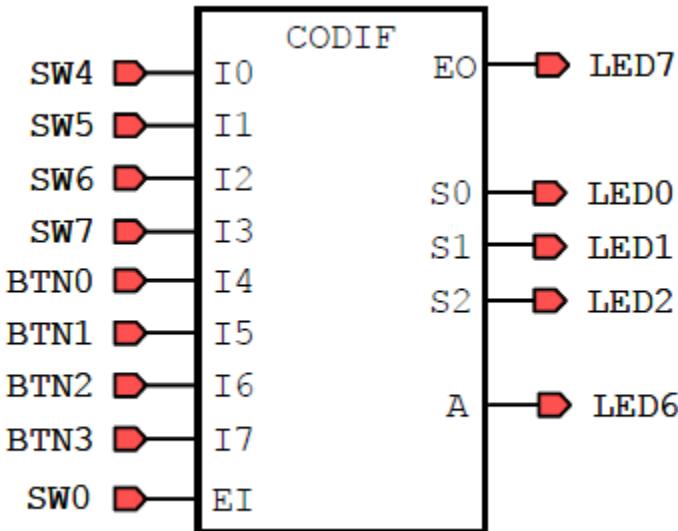




Codificadores con prioridad: ejemplo Cod8a3

1era opción: igual a la de Cod4a2

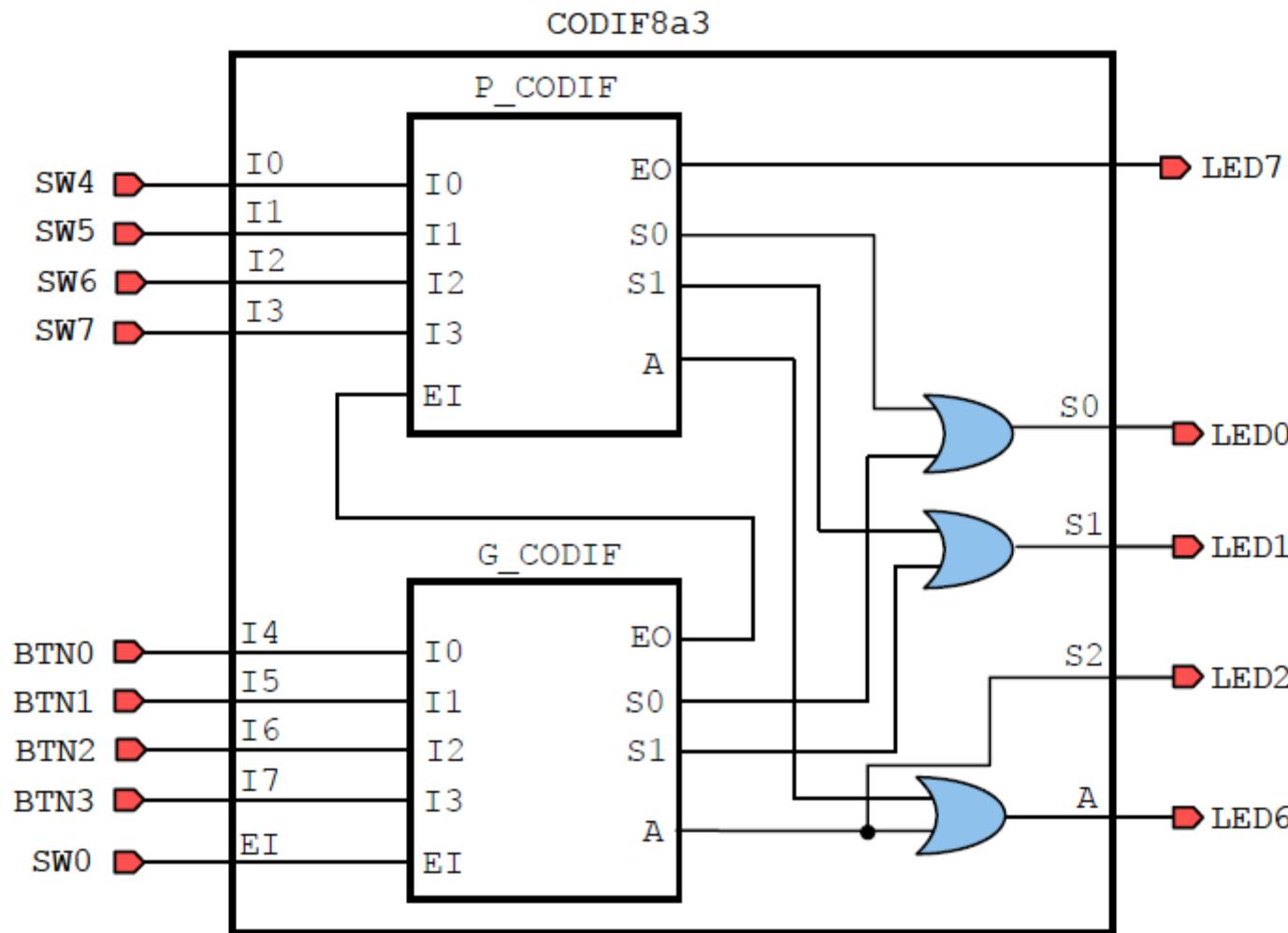
2da opción: usar 2 Cod



EI	I7	I6	I5	I4	I3	I2	I1	I0	S2	S1	S0	A	EO
0	X	X	X	X	X	X	X	X	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0	1	0
1	0	0	0	0	0	0	1	X	0	0	1	1	0
1	0	0	0	0	0	1	X	X	0	1	0	1	0
1	0	0	0	0	1	X	X	X	0	1	1	1	0
1	0	0	0	1	X	X	X	X	1	0	0	1	0
1	0	0	1	X	X	X	X	X	1	0	1	1	0
1	0	1	X	X	X	X	X	X	1	1	0	1	0
1	1	X	X	X	X	X	X	X	1	1	1	1	0

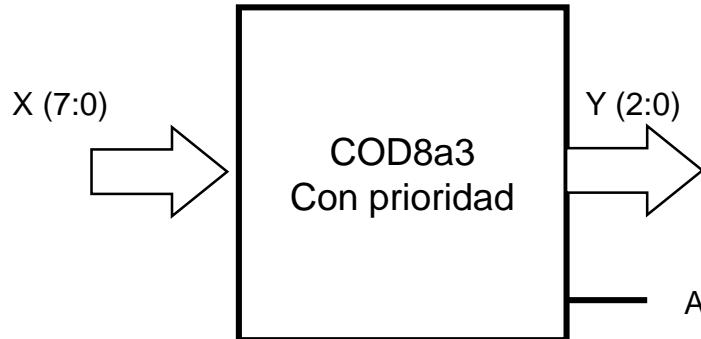


Codificadores con prioridad: ejemplo Cod8a3





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;
```



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

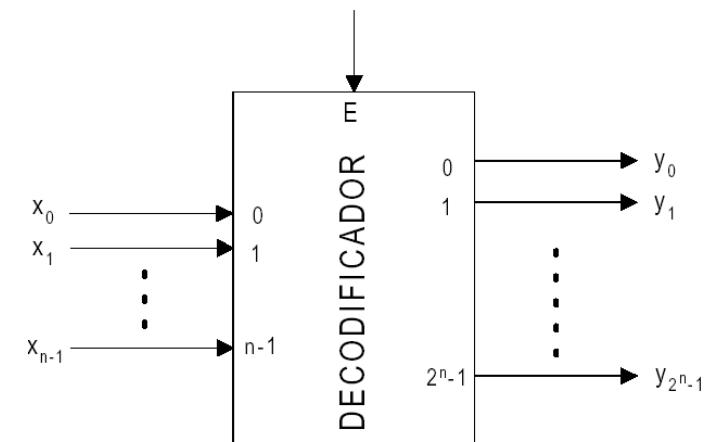
- Chips de memoria : Convierte el n^o binario que designa la dirección de una celda de memoria en la fila o columna correspondiente.

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

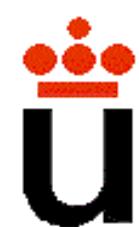
$$X = \sum_{j=0}^{n-1} x_j \cdot 2^j$$

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



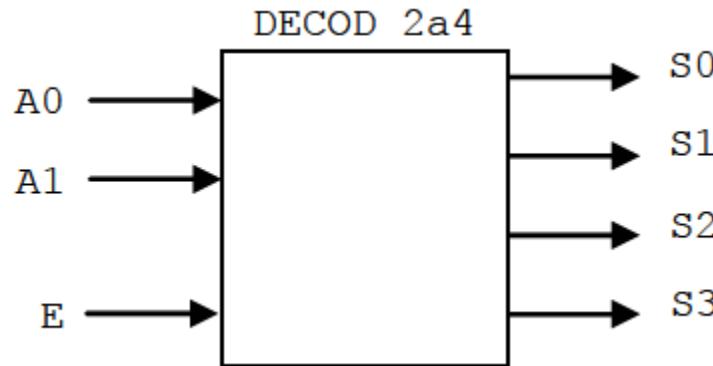
Expresión de conmutación:

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



3. Decodificadores: ejemplo decod 2 a 4

Ejemplo: diseño de un decodificador de 2 a 4:



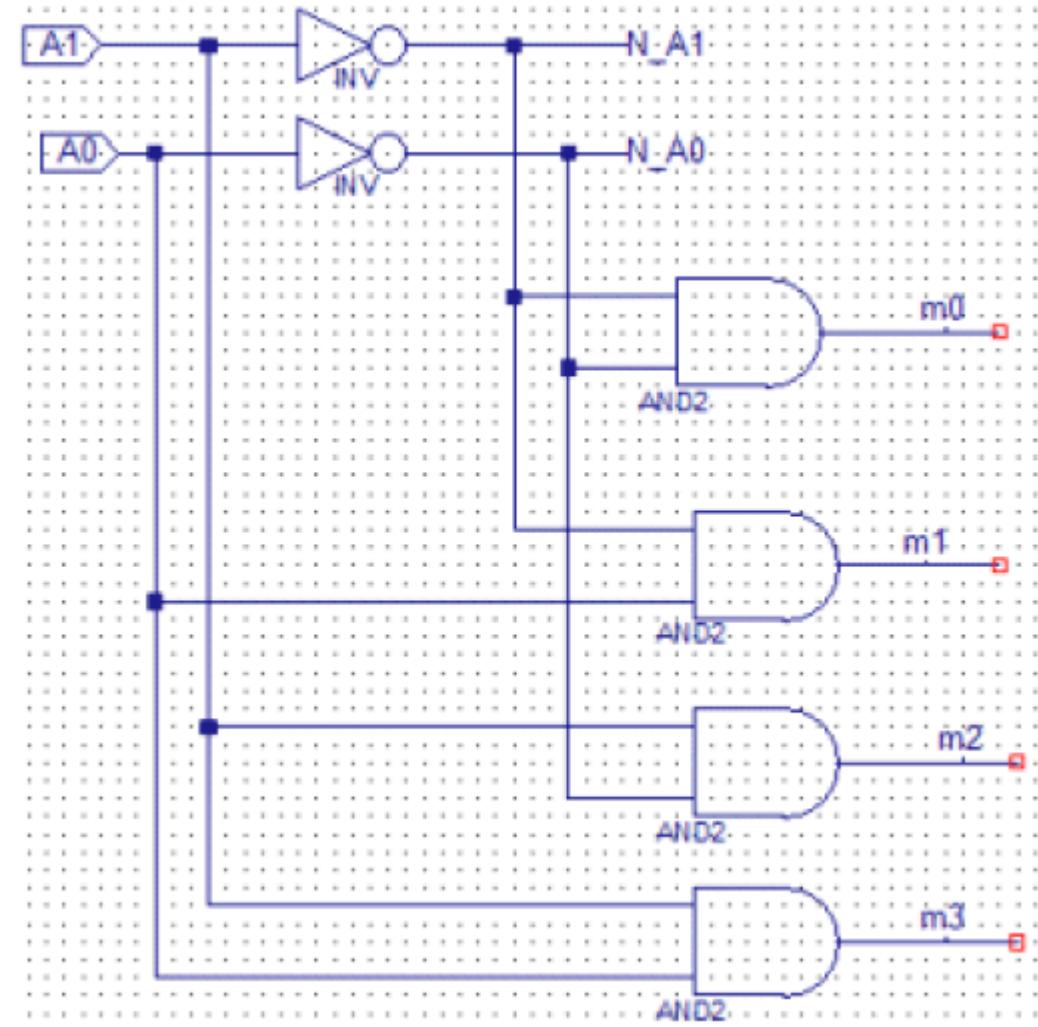
E	A1	A0	S3	S2	S1	S0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



3. Decodificadores: ejemplo decod 2 a 4

Ejemplo: diseño de un decodificador de 2 a 4:

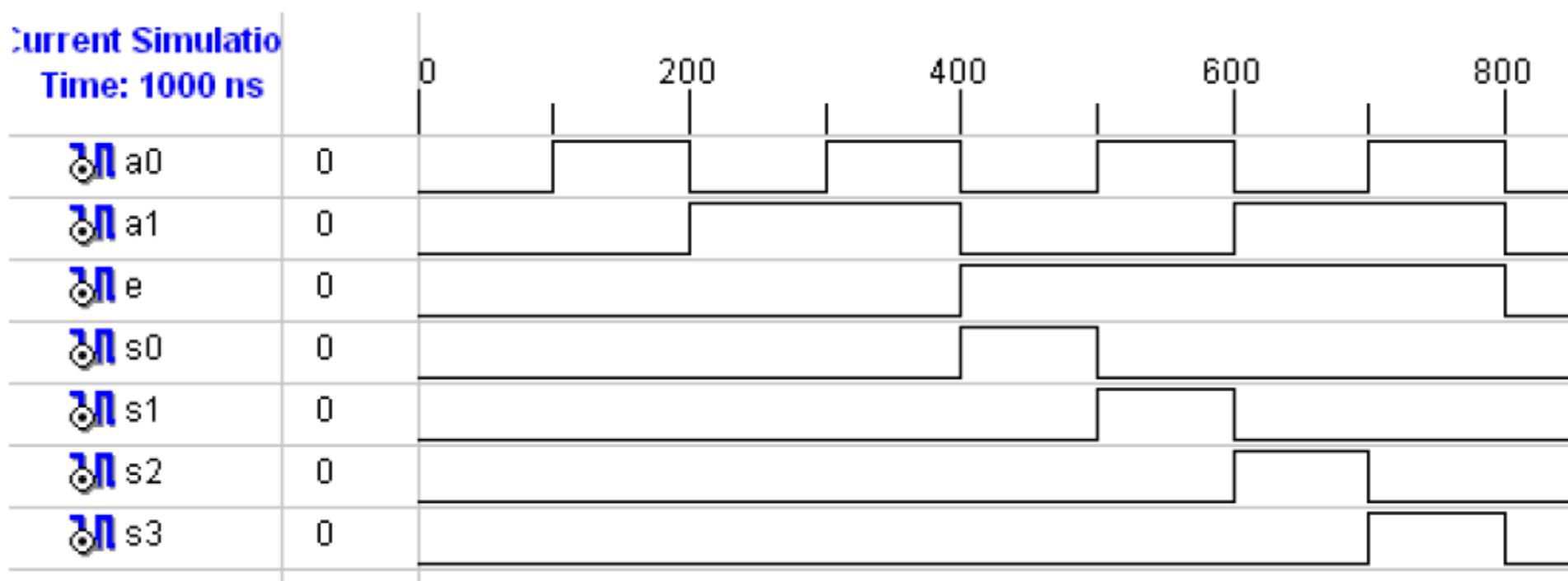
E	A1	A0	S3	S2	S1	S0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



E	A1	A0	S3	S2	S1	S0
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

Current Simulation

Time: 1000 ns

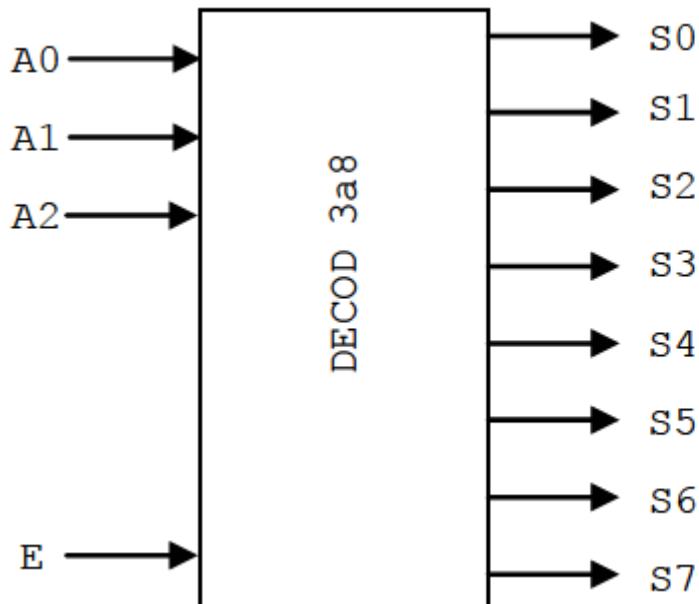




Decodificadores: decod 3 a 8

1era opción: igual a la de decodificador 2 a 4

2da opción: usar 2 decodificador 2 a 4



E	A2	A1	A0	S7	S6	S5	S4	S3	S2	S1	S0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0



Decodificadores: decod 3 a 8

E	A2	A1	A0	S7	S6	S5	S4	S3	S2	S1	S0
0	X	X	X	0	0	0	0	0	0	0	0

todas '0'

E	A2	A1	A0	S7	S6	S5	S4	S3	S2	S1	S0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0

} A2 = '0'

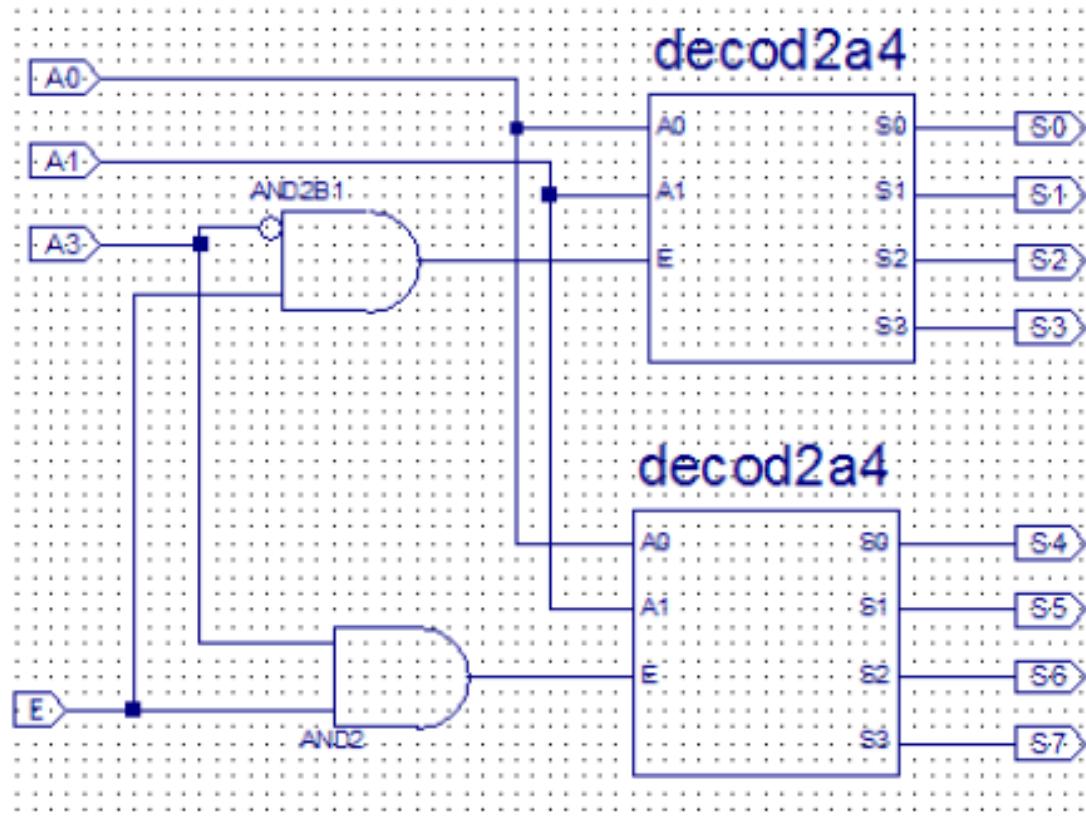
misma
secuencia

E	A2	A1	A0	S7	S6	S5	S4	S3	S2	S1	S0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

} A2 = '1'



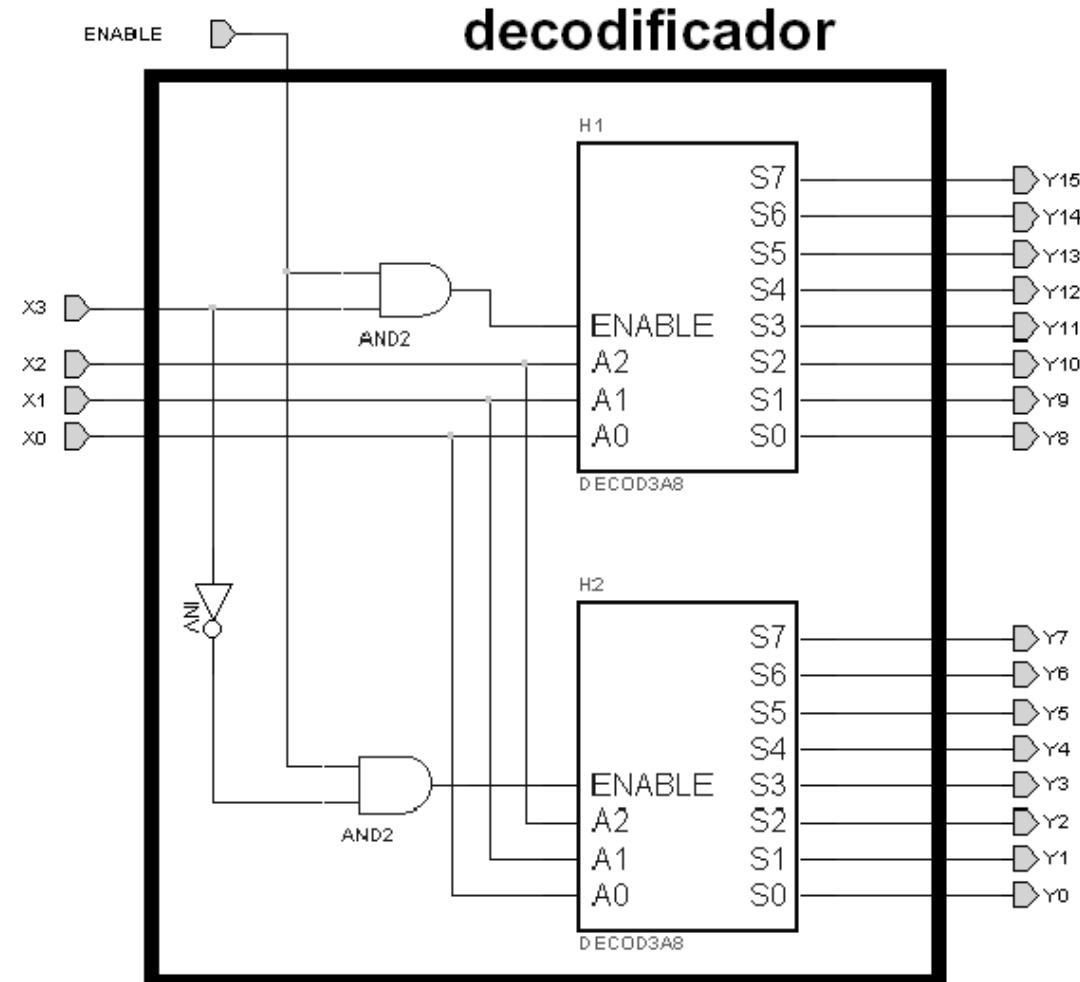
Decodificadores: decod 3 a 8





Decodificadores: diseño jerárquico

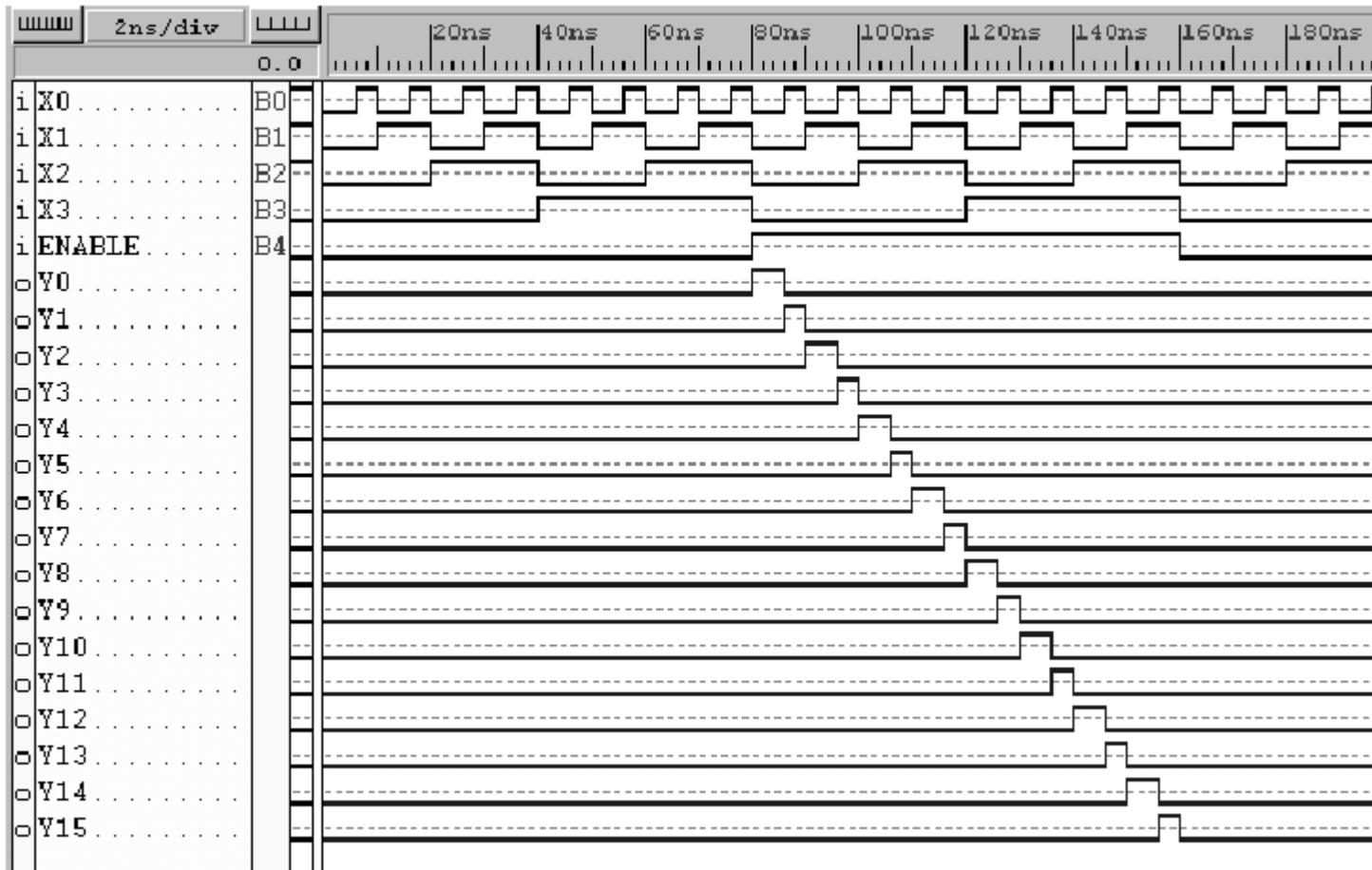
Ejemplo: diseño jerárquico de un decodificador de 4 a 16 mediante decodificadores de 3 a 8.

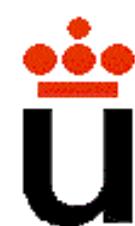




Decodificadores: diseño jerárquico

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





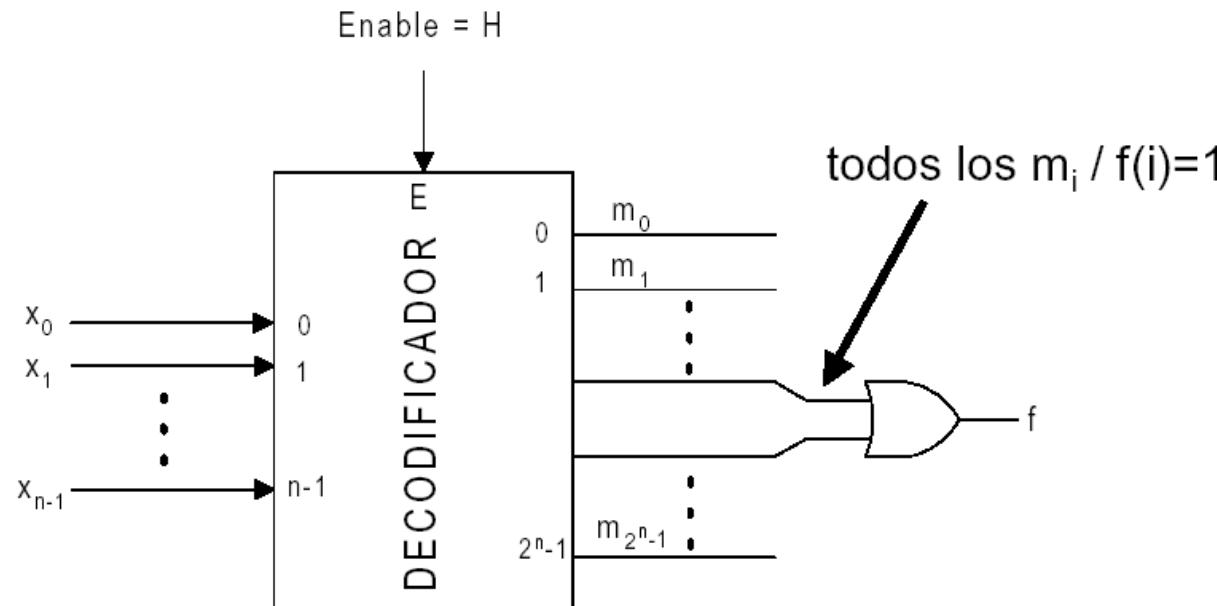
Síntesis de FC con Decodificadores

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:

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

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.**





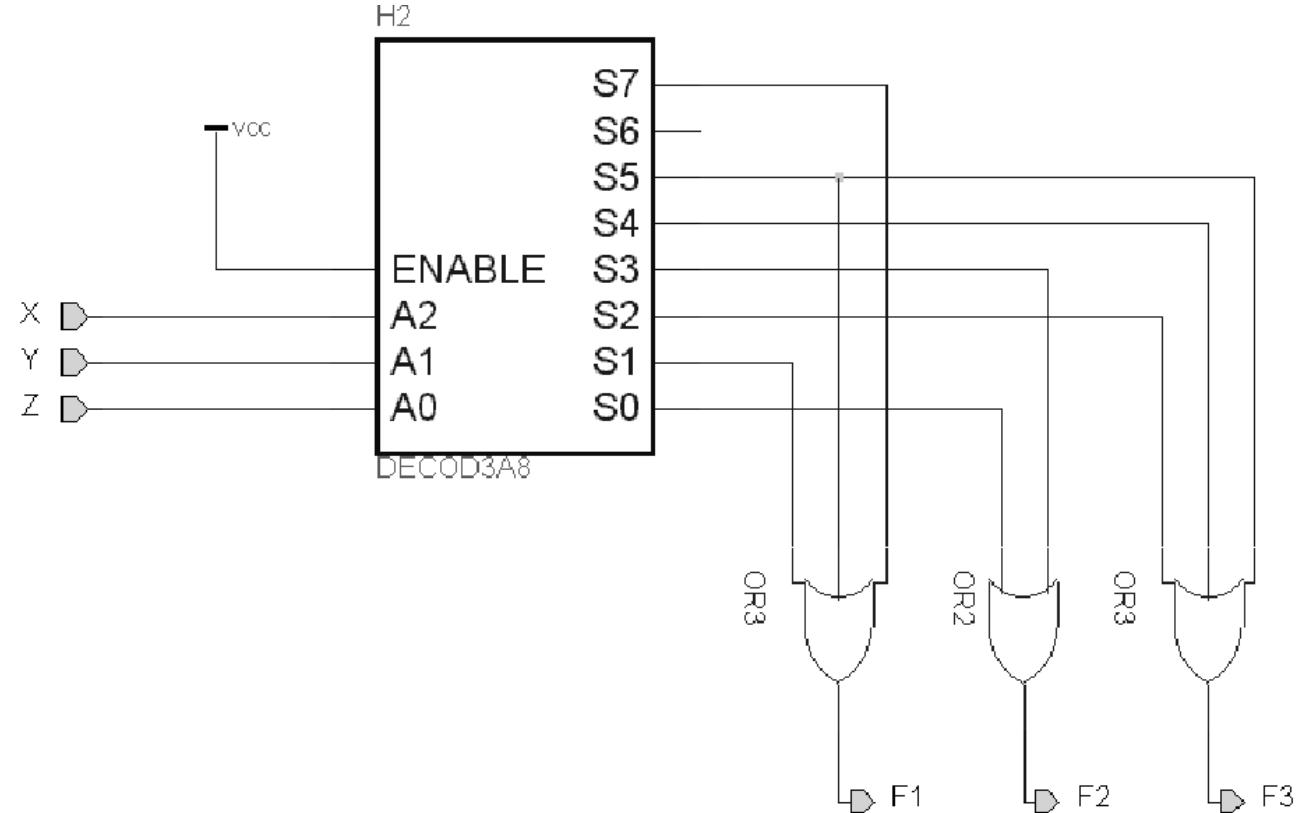
Síntesis de FC con Decodificadores

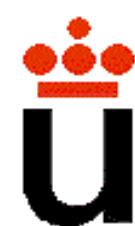
Ejemplo: diseño de las funciones f₁, f₂ y f₃ mediante decodificadores.

$$f_1(x, y, z) = \sum m(1, 5, 7)$$

$$f_2(x, y, z) = \sum m(0, 3)$$

$$f_3(x, y, z) = \sum m(2, 4, 5)$$





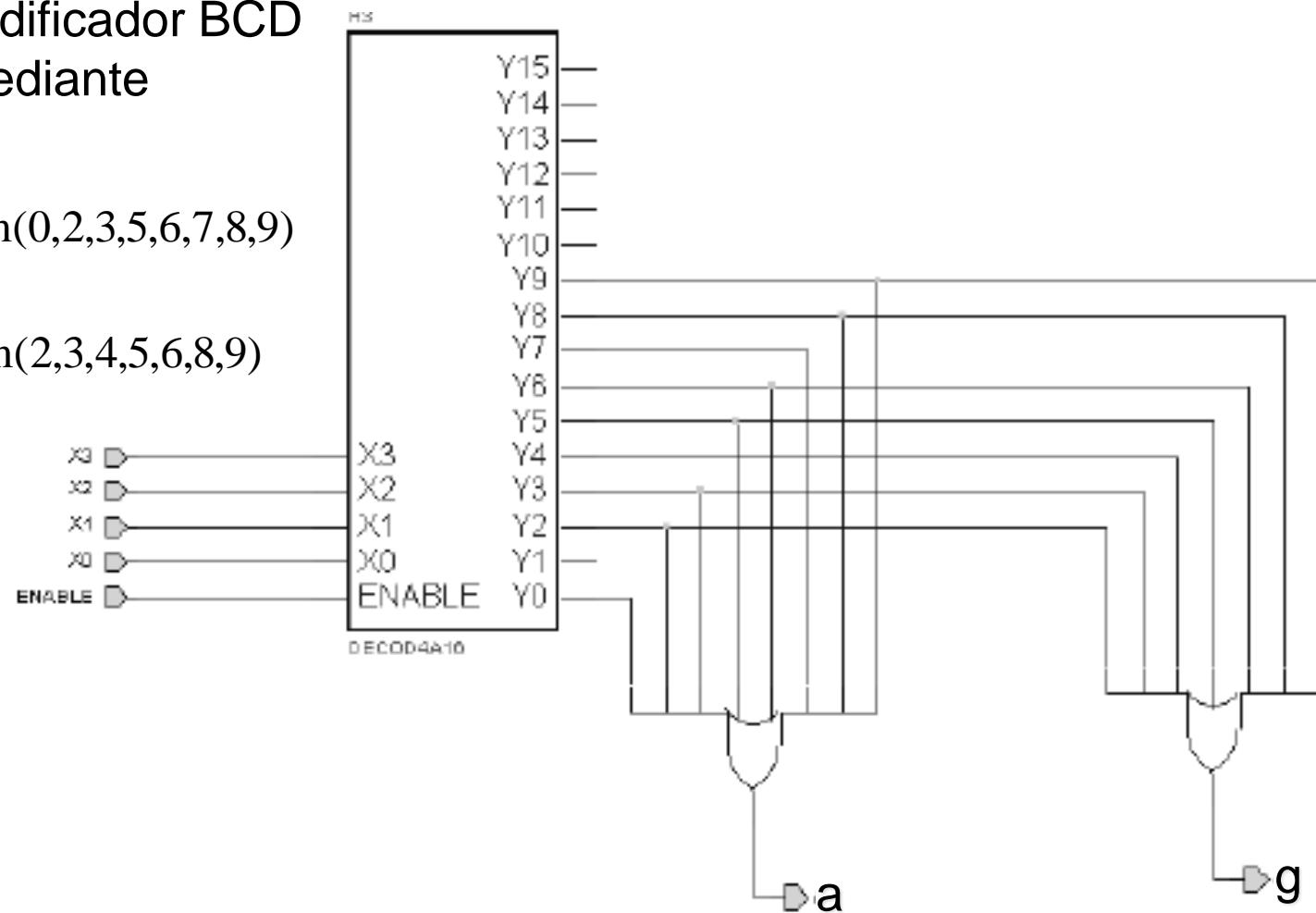
Síntesis de FC con Decodificadores

Materializar un codificador BCD a 7 segmentos mediante decodificadores

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

...

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

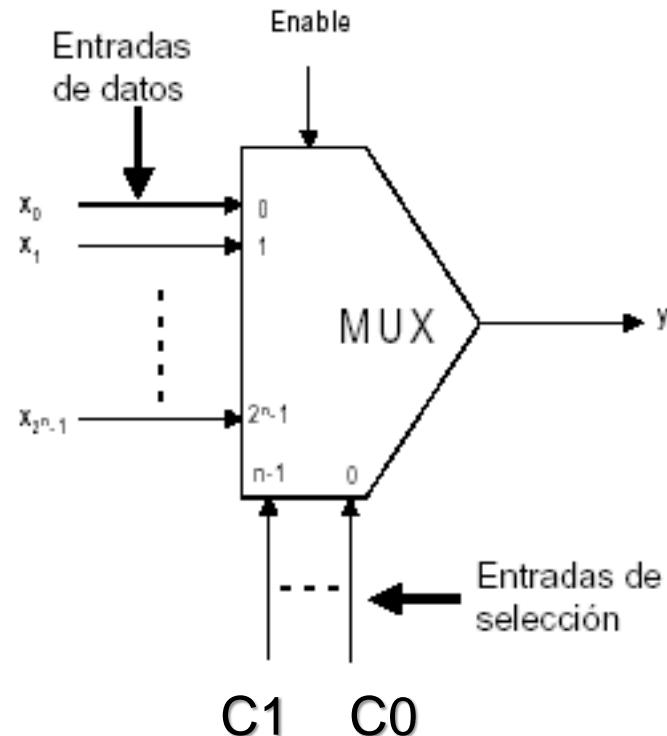




4. Multiplexores

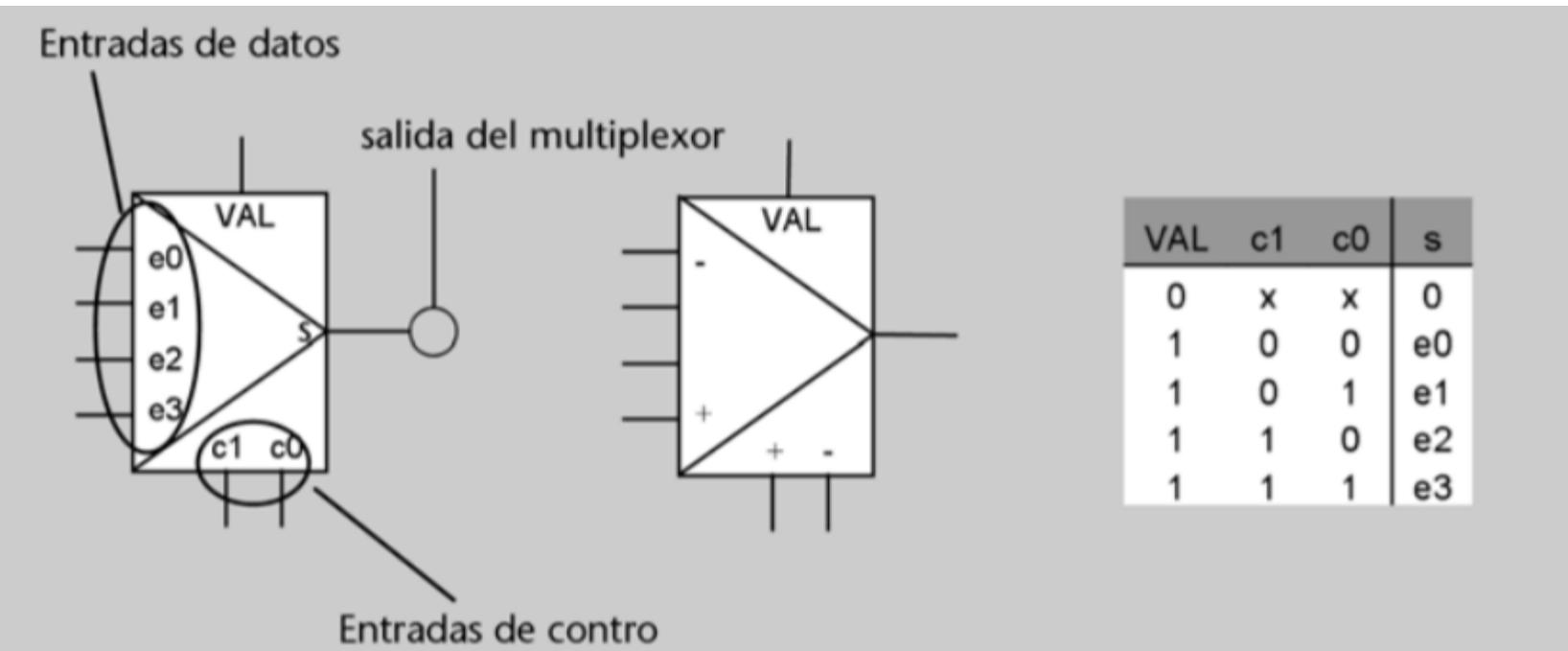
Un multiplexor es un bloque que cumple la función de guardia urbano en circuitos electrónicos. Tiene un determinado número de señales de entrada que “compiten” para conectarse a una señal única de salida, y unas señales de control que sirven para determinar qué señal de entrada se conecta en cada momento con la salida.

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



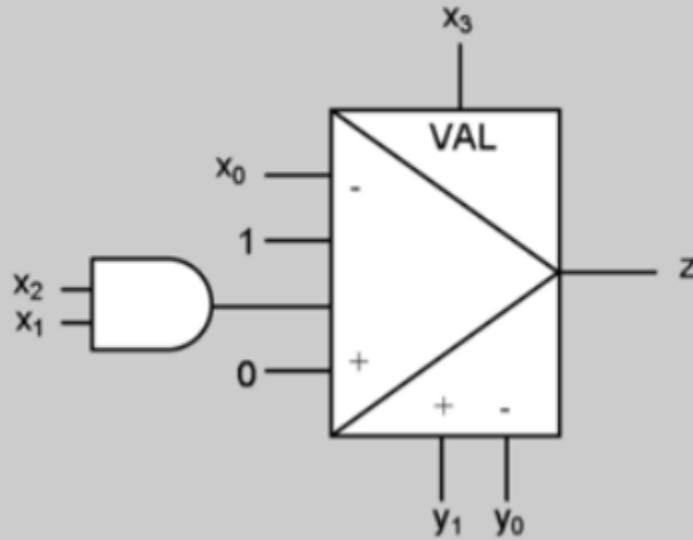


4. Multiplexores



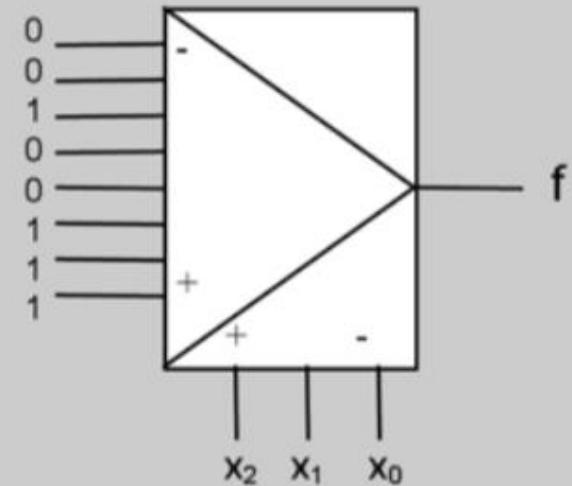


4. Multiplexores: ejemplos

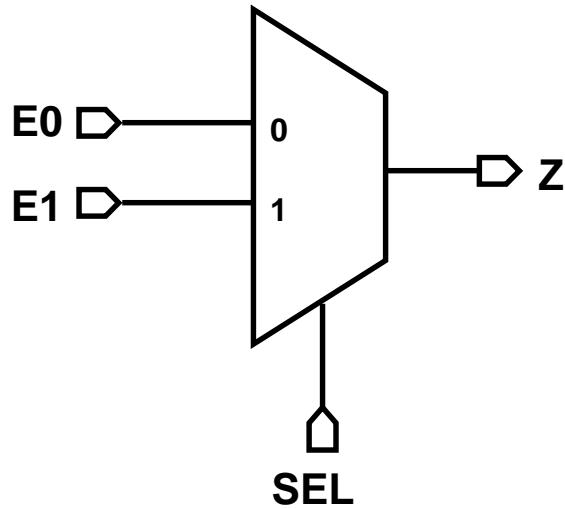


x_3	y_1	y_0	z
0	x	x	0
1	0	0	x_0
1	0	1	1
1	1	0	$x_2 x_1$
1	1	1	0

x_2	x_1	x_0	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



```
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;
```

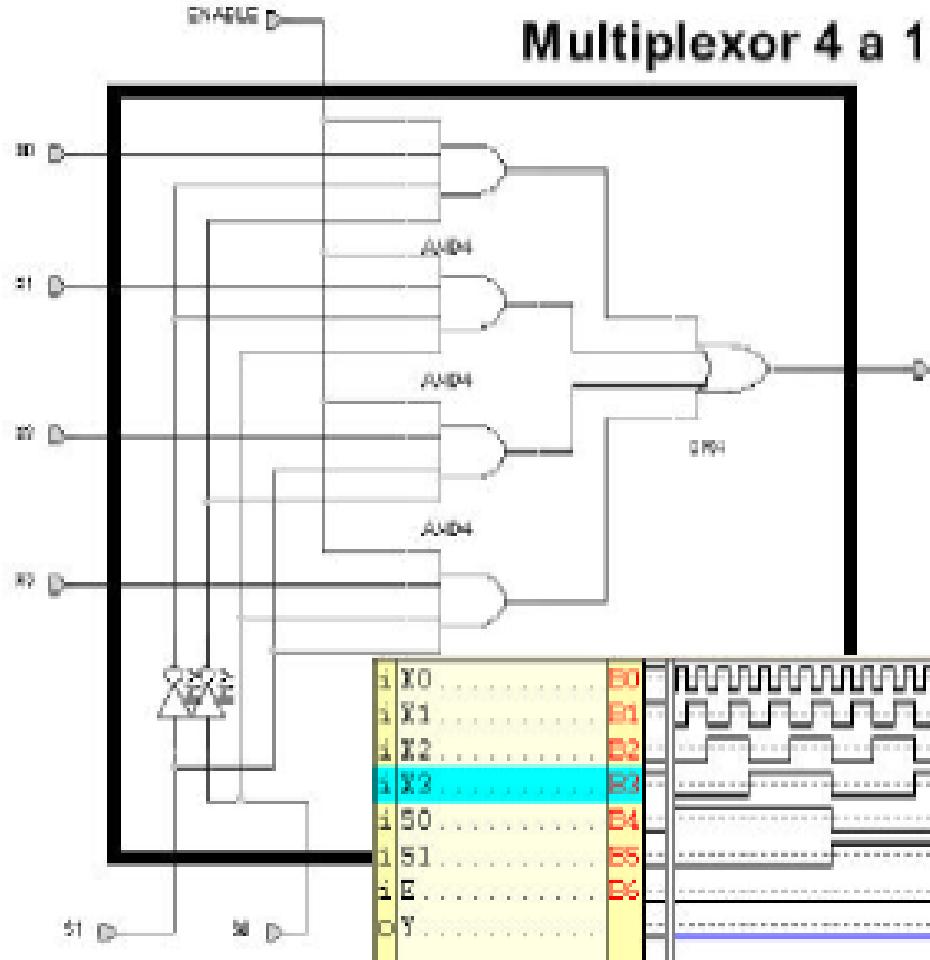
```
architecture comportamental2 of MUX is
begin
  P1: process(E0, E1, SEL)
  begin
    if SEL = '0' then
      Z <= E0;
    else
      Z <= E1;
    end if;
  end process P1;
end comportamental2;
```



Multiplexores

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

Comercial: 74293



$$\begin{aligned}y &= E \cdot (x_0 \cdot m_0(s_1, s_0) + x_1 \cdot m_1(s_1, s_0) + \\&\quad + x_2 \cdot m_2(s_1, s_0) + x_3 \cdot m_3(s_1, s_0)) : \\&= E \cdot (x_0 \cdot \overline{s}_1 \cdot \overline{s}_0 + x_1 \cdot \overline{s}_1 \cdot \overline{s}_0 + \\&\quad + x_2 \cdot \overline{s}_1 \cdot s_0 + x_3 \cdot s_1 \cdot s_0)\end{aligned}$$



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:

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

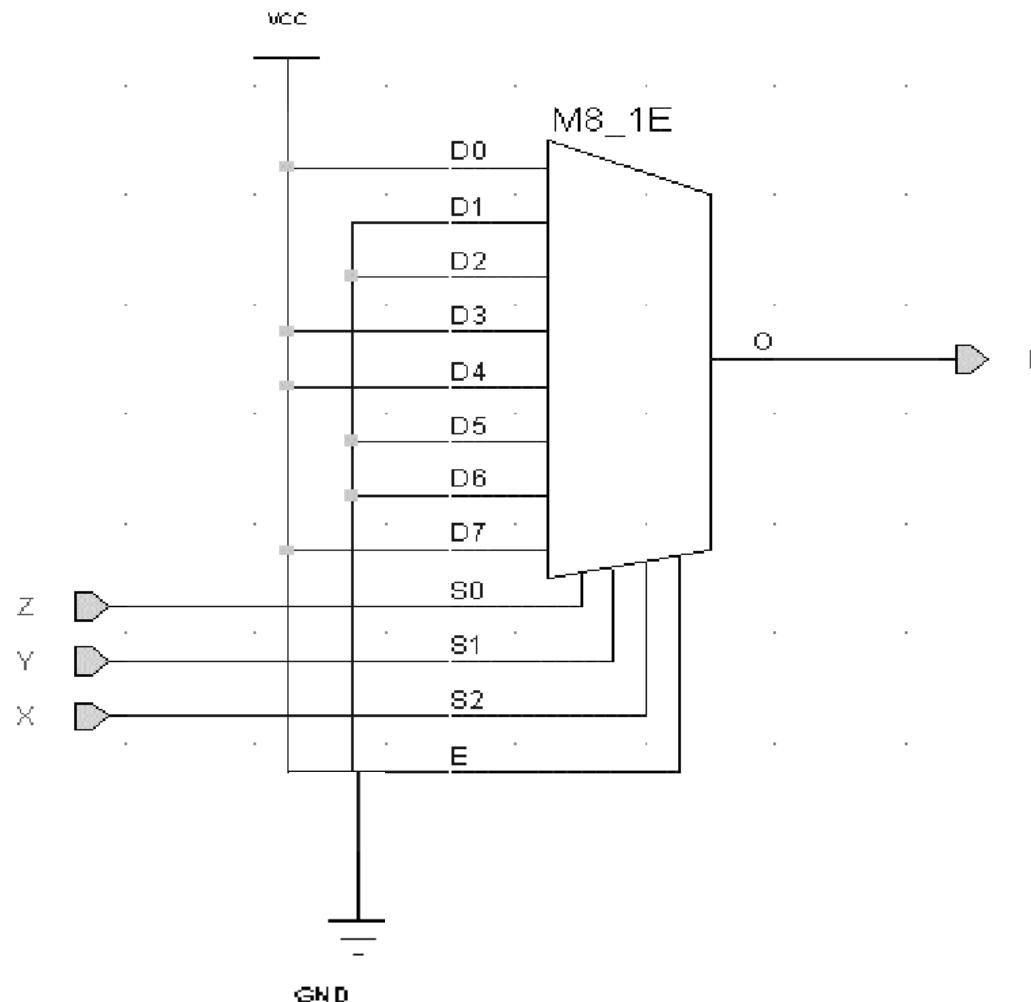
Obviando E , esta expresión coincide con la expresión del multiplexor si se identifican: $x_i = f(i) \quad \forall i=0, \dots, 2^n-1$ y $(s_{n-1}, \dots, s_0) = (a_{n-1}, \dots, a_0)$.

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 3 bit.

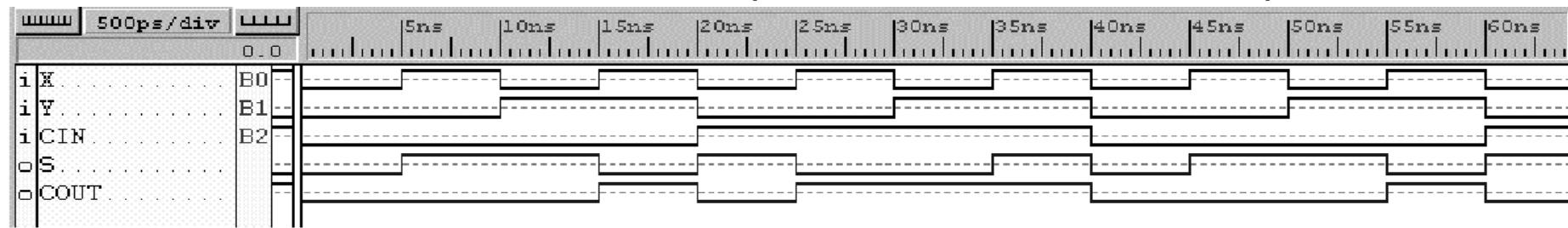
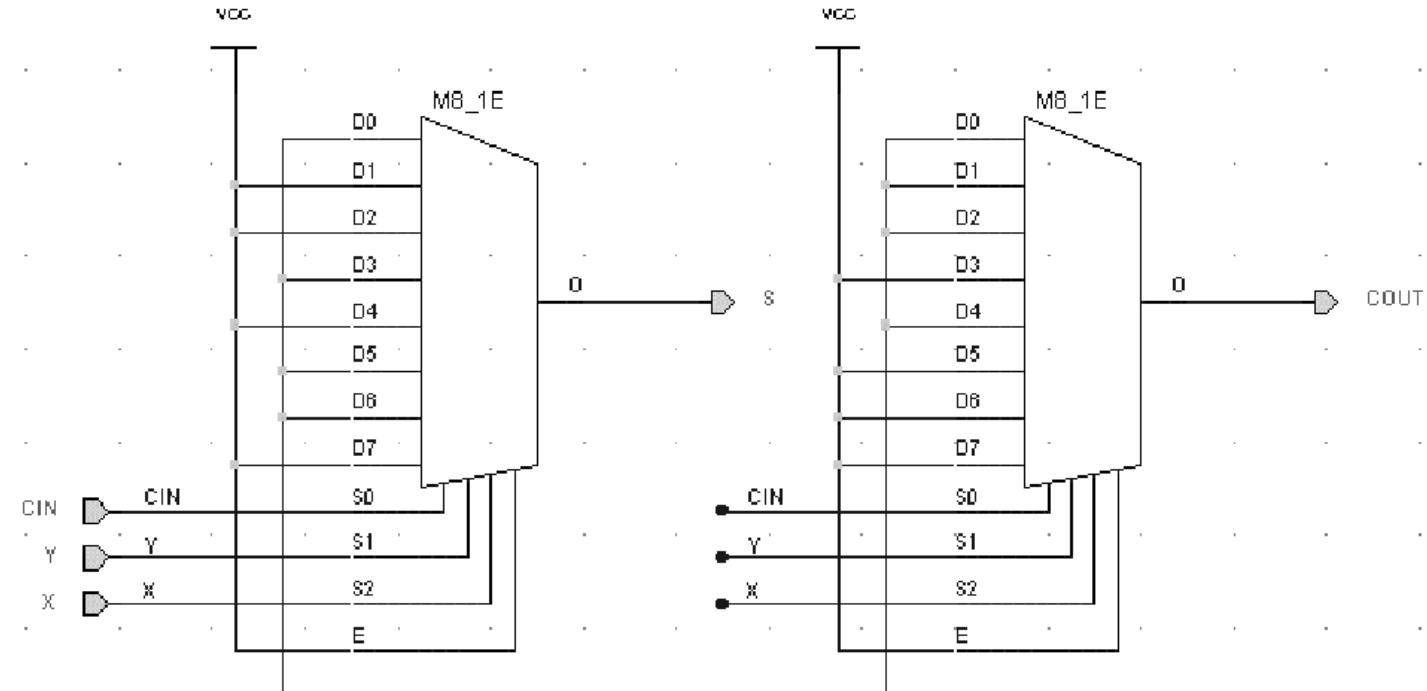


Síntesis de FC con multiplexores

Ejemplo: diseño mediante multiplexores de un sumador binario completo de 3 bit.

$$S = \sum m(1, 2, 4, 7)$$

$$COUT = \sum m(3, 5, 6, 7)$$

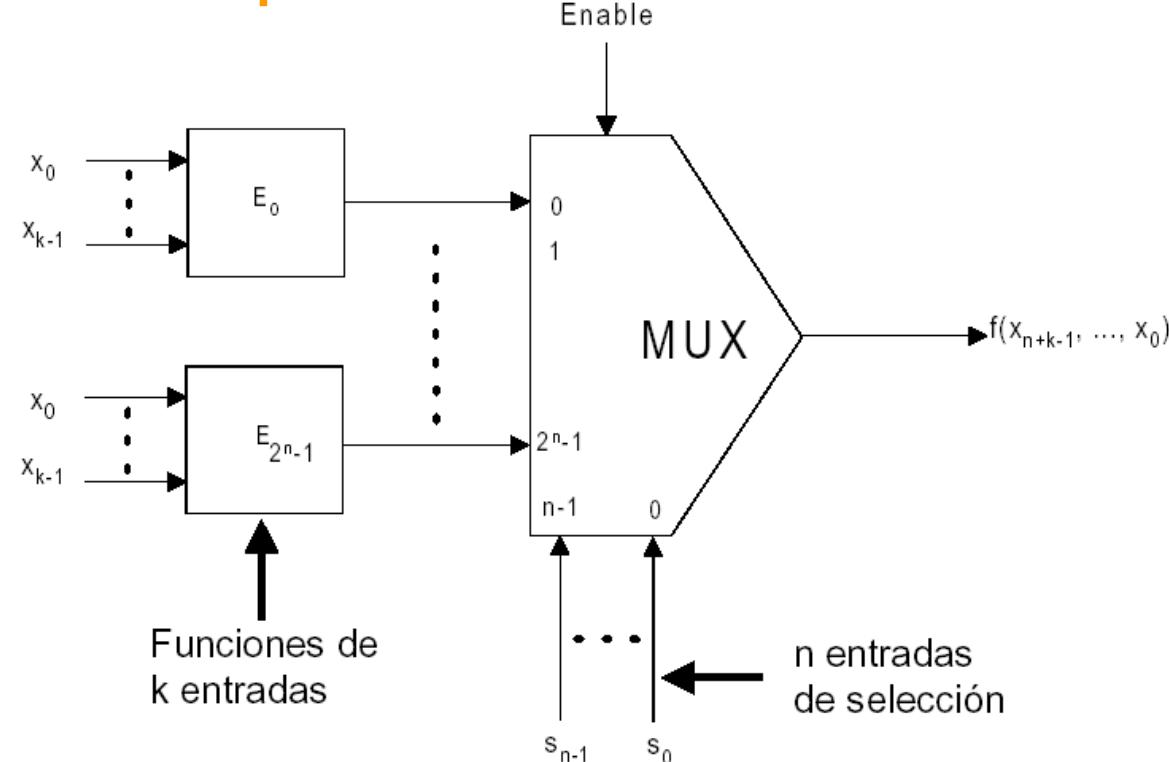




Síntesis de FC con multiplexores

¿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.



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



Síntesis de FC con multiplexores

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

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

$$\begin{aligned} f(a,b,c,d) &= \overline{\overline{abcd}} + \overline{\overline{abcd}} = \\ &= (\overline{ab})\overline{cd} + (\overline{ab} + ab)\overline{cd} + (\overline{ab} + a\overline{b} + ab)\overline{cd} + (\overline{ab} + \overline{ab} + a\overline{b})cd \end{aligned}$$

$$(z_1, z_0) = (a, b) \quad E_0 = (\overline{ab})$$

$$(s_1, s_0) = (c, d) \quad E_1 = (\overline{\overline{ab}} + ab) = \overline{b}$$

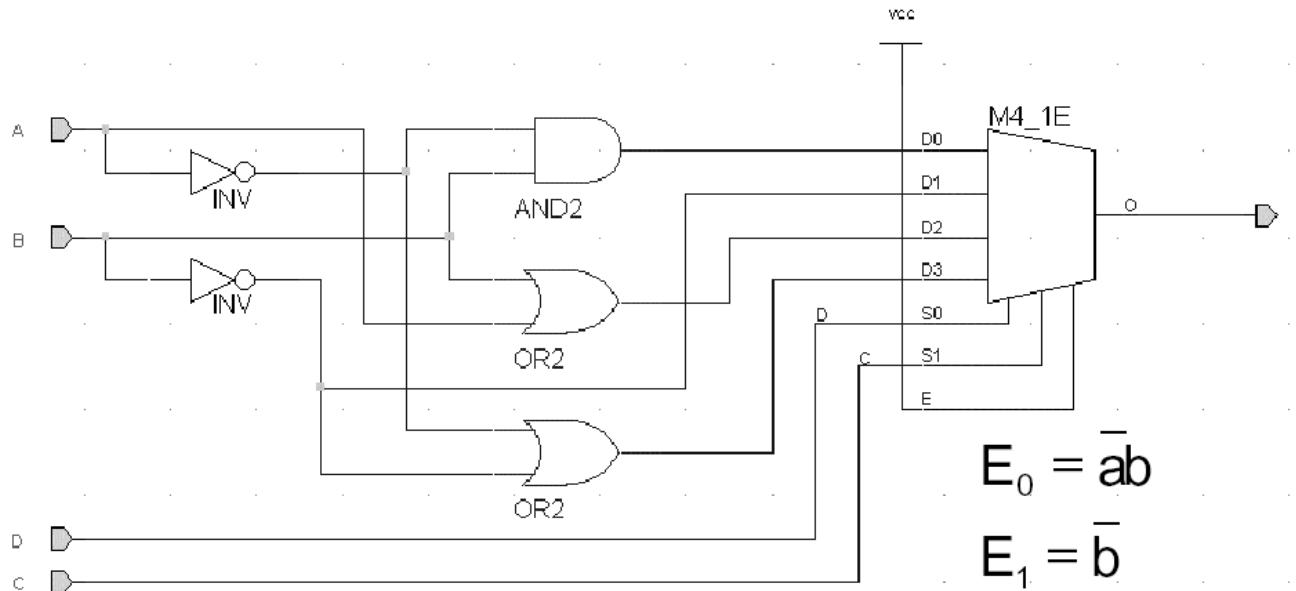
$$E_2 = (\overline{\overline{ab}} + a\overline{b} + ab) = a + b$$

$$E_3 = (\overline{\overline{ab}} + \overline{ab} + a\overline{b}) = \overline{a} + \overline{b}$$

Simplificación algebraica.



Síntesis de FC con multiplexores

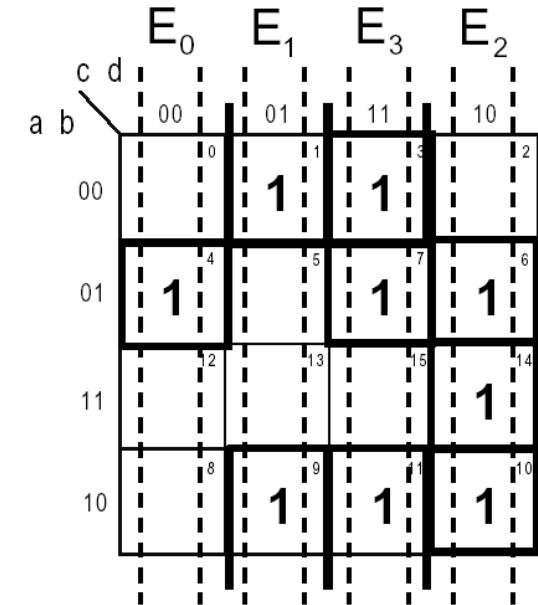


$$E_0 = \bar{a}b$$

$$E_1 = \bar{b}$$

$$E_2 = a + b$$

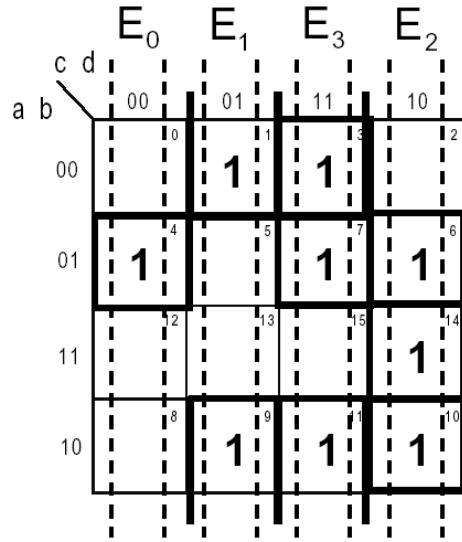
$$E_3 = \bar{a} + \bar{b}$$



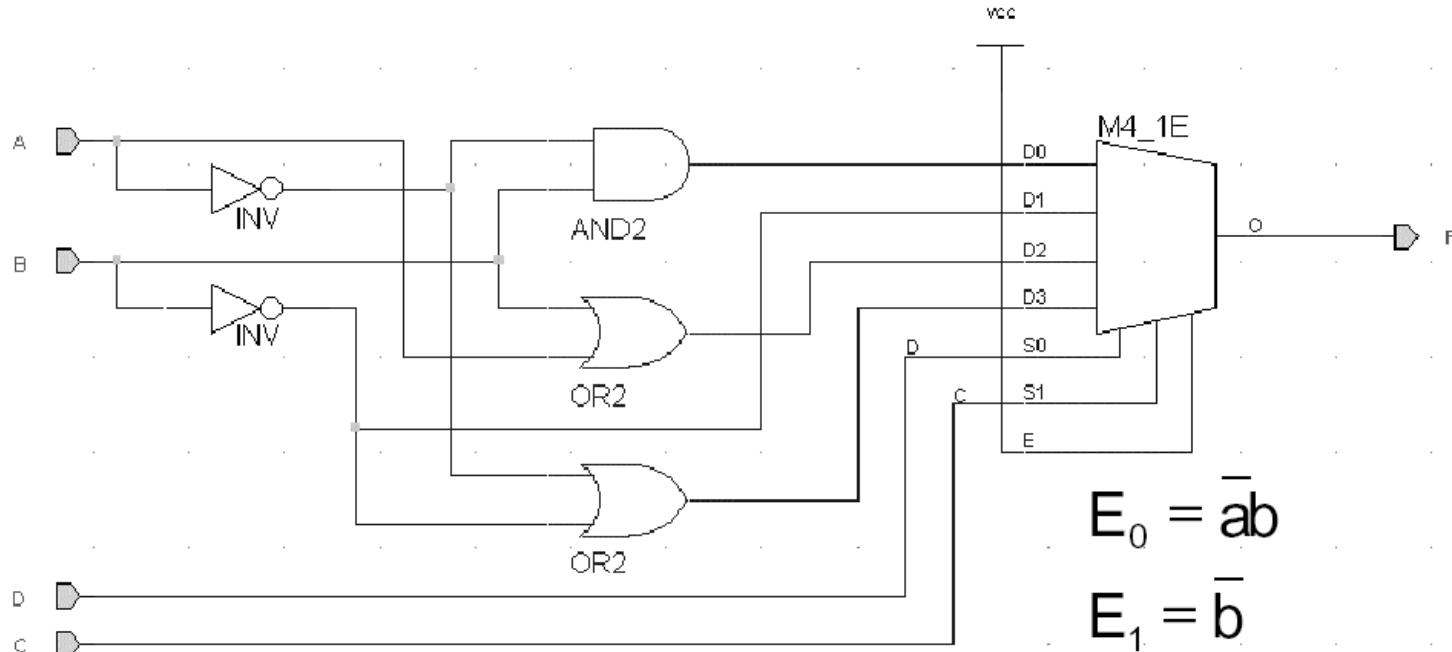
Simplificación
con Karnaugh



Síntesis de FC con multiplexores



Simplificación
con Karnaugh



$$E_0 = \overline{ab}$$

$$E_1 = \overline{b}$$

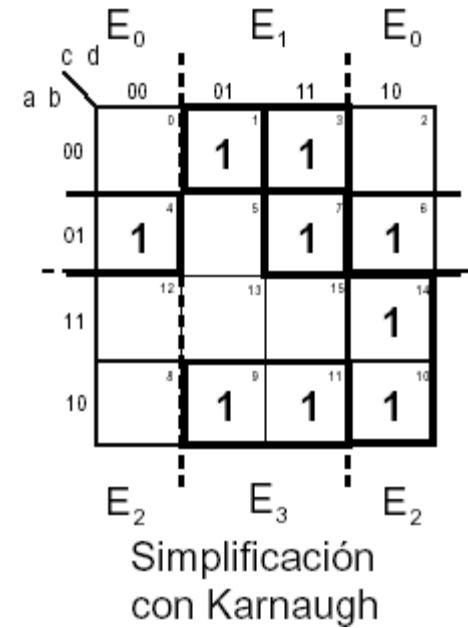
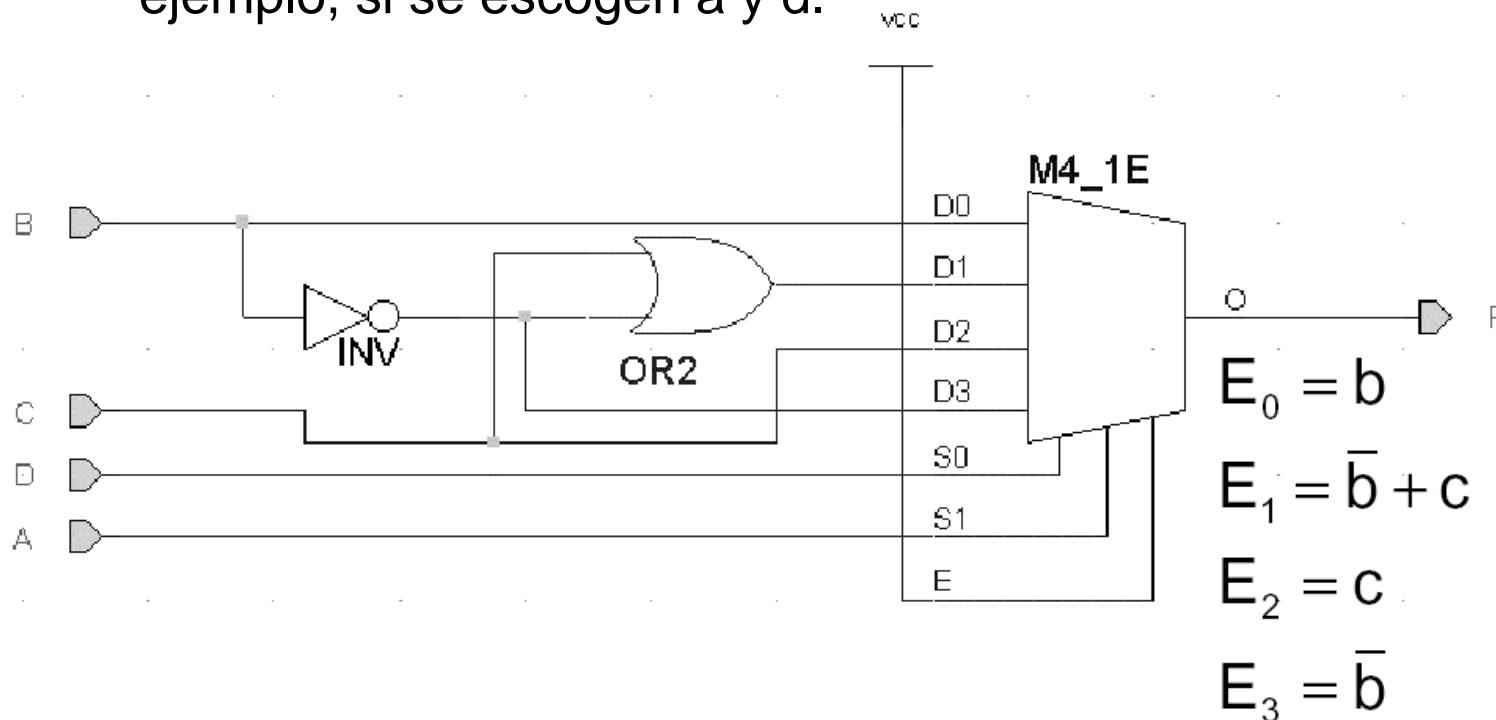
$$E_2 = a + b$$

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



Síntesis de FC con multiplexores

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





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

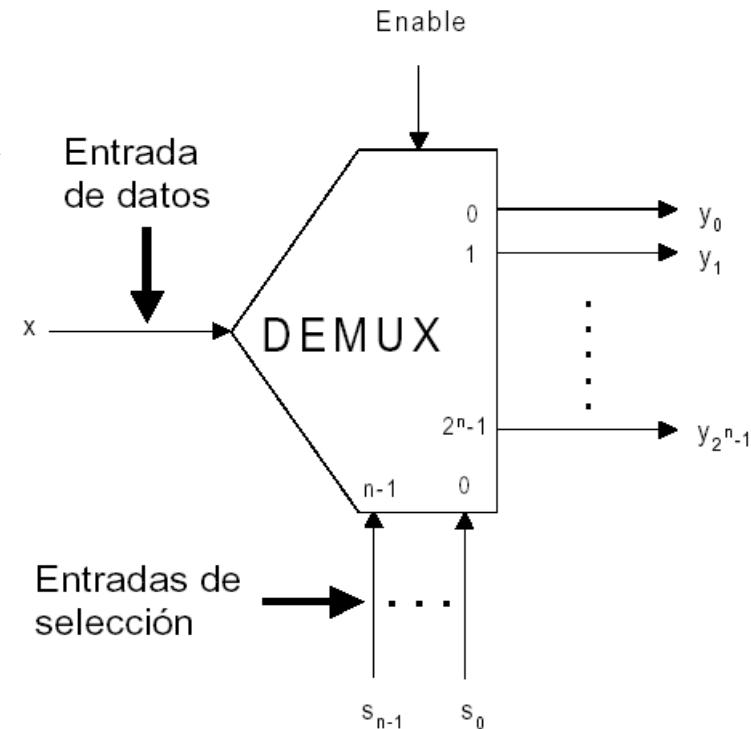


5. Demultiplexores

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**.

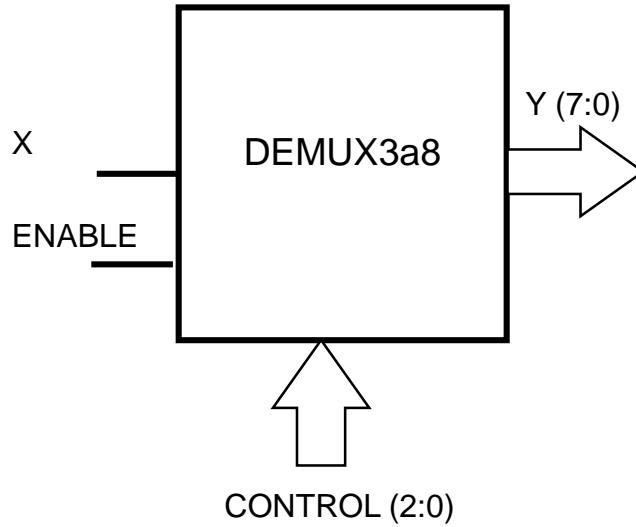
El demultiplexor conecta la entrada con una de las 2^n salidas que se selecciona con la palabra de control S.

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 "111";
Y(6)<= Entrada when "110";
Y(5)<= Entrada when "101";
Y(4)<= Entrada when "100";
Y(3)<= Entrada when "011";
Y(2)<= Entrada when "010";
Y(1)<= Entrada when "001";
Y(0)<= Entrada when "000";
Y <= (others => '0')when others;
end comportamental;
```



6. Desplazadores

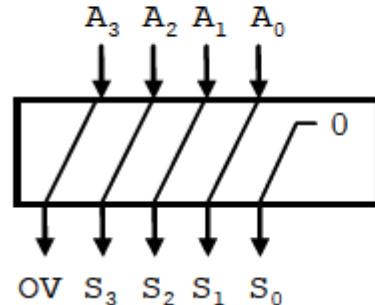
Un desplazador (*shifter*) es un bloque combinacional que tiene la función de desplazar bits hacia la izquierda o hacia la derecha.

Los desplazadores tienen una señal de entrada y una señal de salida, las dos de **n bits**. La señal de salida se obtiene desplazando los bits de entrada m veces hacia la derecha o hacia la izquierda.

Desplazamiento lógico

- ✓ Si el desplazamiento es hacia la izquierda, los m bits de menos peso de la salida se ponen a 0: multiplicación por 2^m .

Multiplicador x2



Ejemplos:

6: 0110 ₂	x2	12: 1100 ₂	18: 10010 ₂
		10: 1010 ₂	overflow no cabe en 4 bits



6. Desplazadores

- ✓ Si el desplazamiento es hacia la derecha, los m bits de mas peso de la salida se ponen a 0: división por 2^m .

Ejemplos

6: 0110

:2

3: 0011

8: 1000

:2

4: 0100



6. Desplazadores

Desplazamiento aritmético

Los desplazamientos aritméticos son similares a los desplazamientos lógicos, solo que los aritméticos están pensados para trabajar sobre números enteros con signo en representación de complemento a dos en lugar de enteros sin signo.

- Los desplazamientos aritméticos permiten la multiplicación y la división por dos, de números enteros con signo, por una potencia de dos. Desplazar n bits hacia la izquierda o a la derecha equivale a multiplicar o dividir por 2^n

**Ejemplo: Desplazamiento
a izquierda**

$$\begin{aligned}11110100_{C2} &= -12_{10} \\11101000_{C2} &= -24_{10}\end{aligned}$$

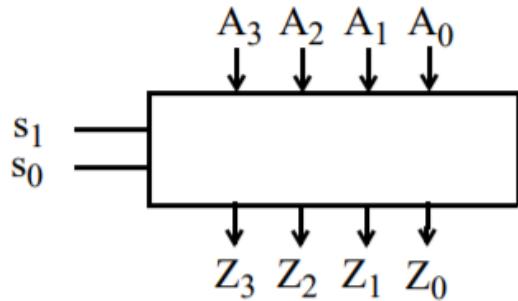
Desplazamiento a derecha

$$\begin{aligned}11110100_{C2} &= -12_{10} \\11111010_{C2} &= -6_{10}\end{aligned}$$

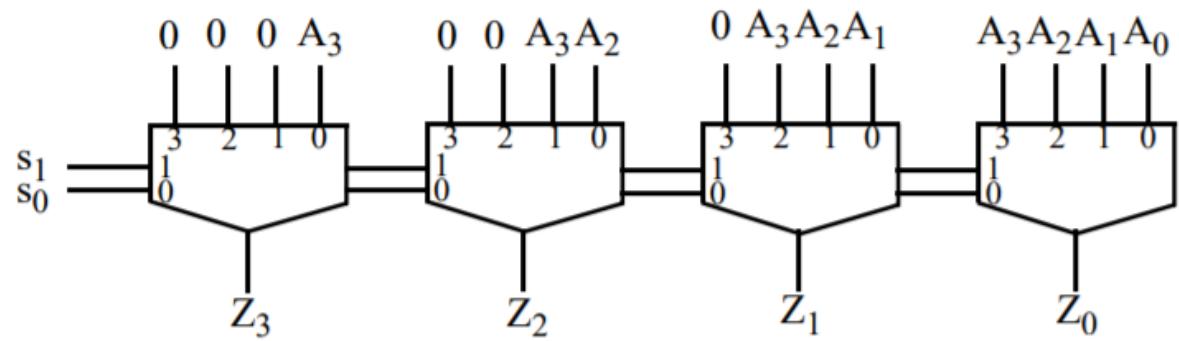


6. Desplazadores

Desplazamiento a derecha



s_1	s_0	Z_3	Z_2	Z_1	Z_0
0	0	A_3	A_2	A_1	A_0
0	1	0	A_3	A_2	A_1
1	0	0	0	A_3	A_2
1	1	0	0	0	A_3





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:

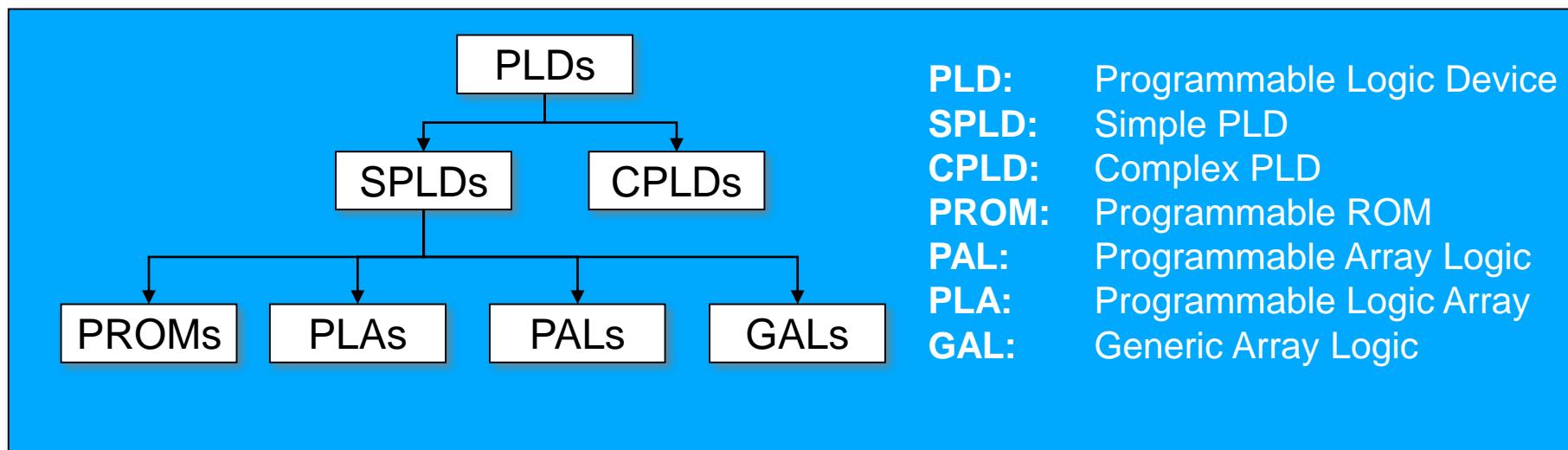
- ▶ Consumos medios, aunque hay familias especializadas en bajo consumo
- ▶ Velocidad intermedia
- ▶ Fiabilidad alta
- ▶ Tiempo de desarrollo muy bajo, sin dependencia de terceros
- ▶ Metodología sencilla
- ▶ Equipamiento sencillo
- ▶ Aumentan la confidencialidad de las placas



7. Dispositivos lógicos programables

Evolución temporal y escala de integración:

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



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 μ P

Programar consiste en cambiar las instrucciones que le llegan al microprocesador



Cambia el SOFTWARE

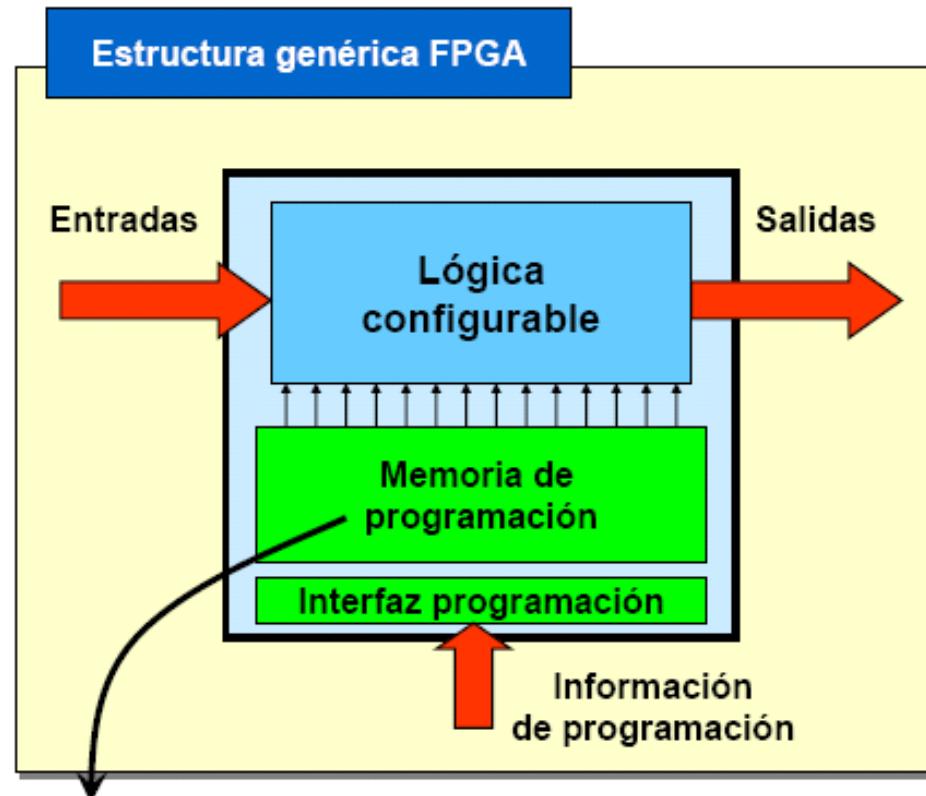
FPGA

Programar consiste en cambiar las conexiones y entradas/salidas de la lógica del dispositivo



Cambia el HARDWARE

Programar: cambiar las funciones lógicas y las conexiones



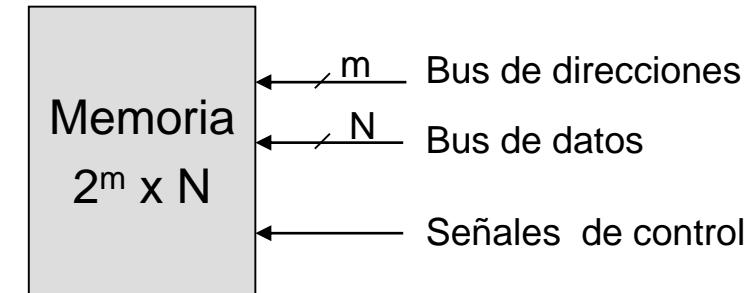
La implementación física de la memoria sirve para clasificar las FPGAs



7.1. Memorias ROM

Una memoria es un dispositivo de almacenamiento de información

- ⇒ RAM Random Access 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)

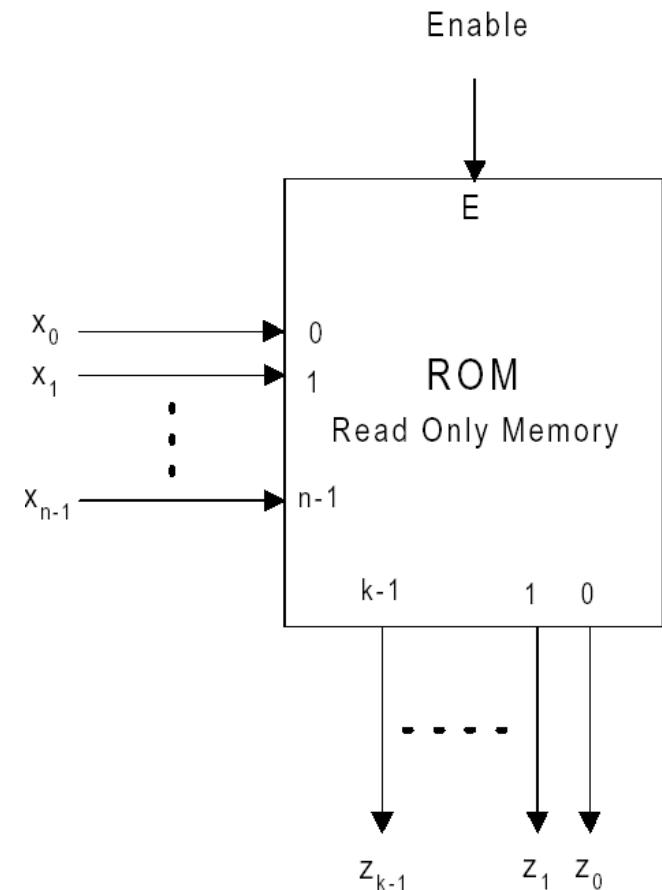


Memorias ROM

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.

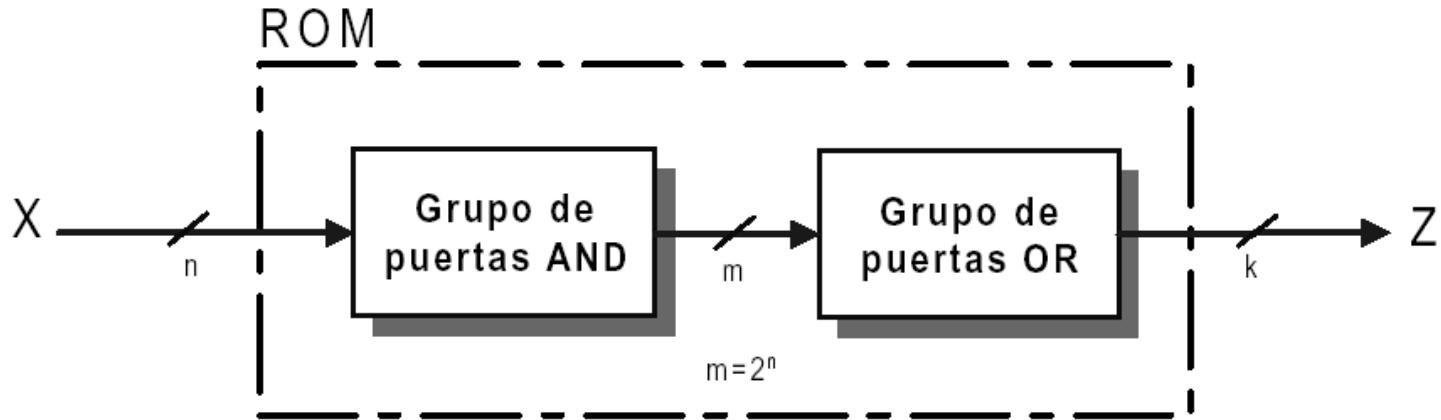
Existen distintos tipos según los datos sean o no permanentes (ROM, PROM o EPROM, EEPROM), sean no programables o programables (**ROM o PROM, EPROM, EEPROM**), y cuantas veces, y como se realice físicamente el borrado y la programación (EPROM - UV, EEPROM - eléctrica).





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.



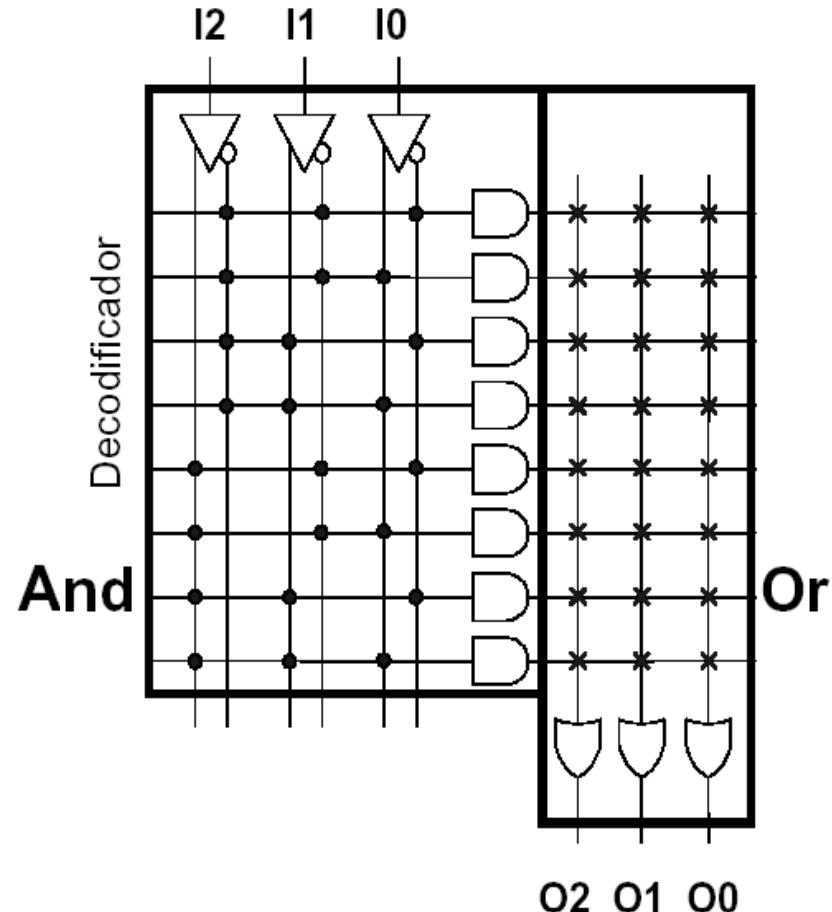
El grupo de puertas **AND** están programadas de antemano y conectadas de forma inalterable, mientras que el grupo de puertas **OR** son programables por el usuario.



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

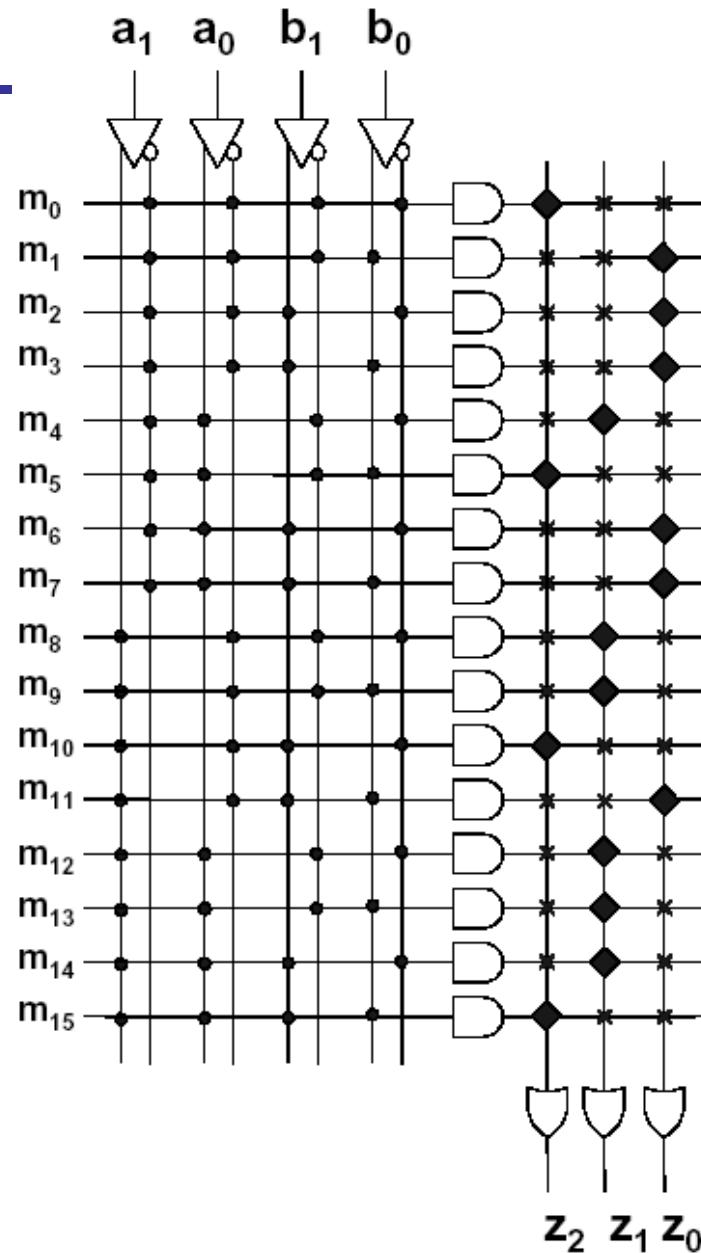
$$Z = \begin{cases} \text{IG} & \text{si } A = B \\ \text{MA} & \text{si } A > B \\ \text{ME} & \text{si } A < B \end{cases}$$

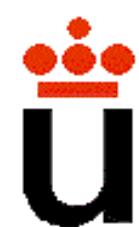
$$z_2 = \sum m(0,5,10,15)$$

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

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

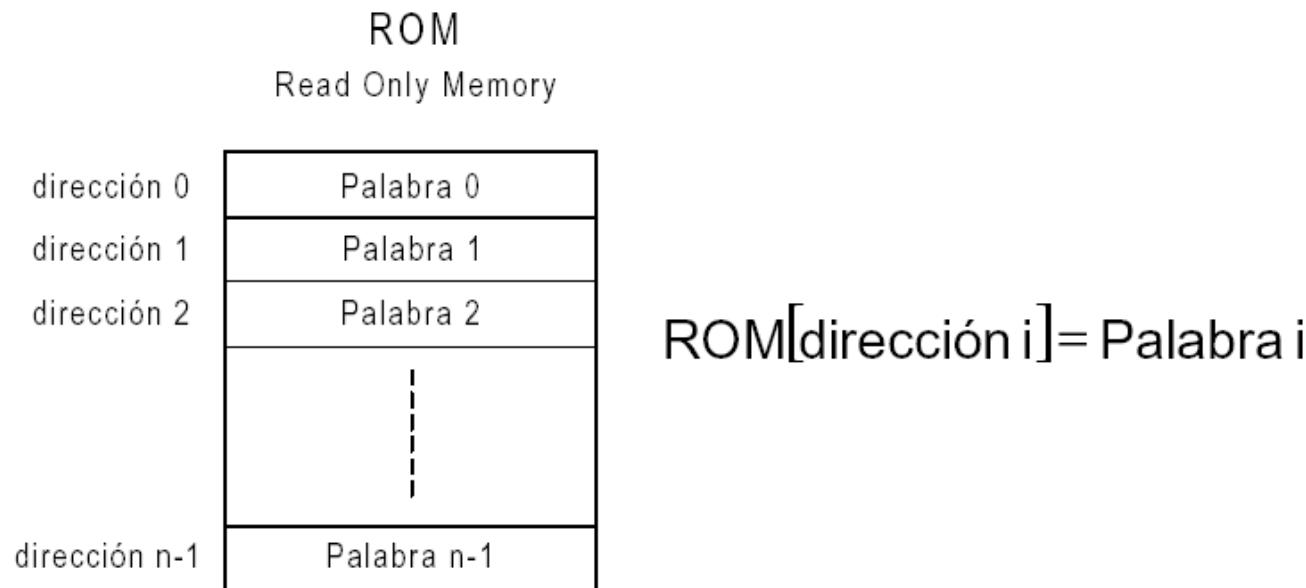
La solución consiste en seleccionar las salidas que generan los minterms de las funciones y programar las conexiones en el grupo OR para cada una de las salidas. Se almacena directamente la tabla de verdad.





Memorias ROM

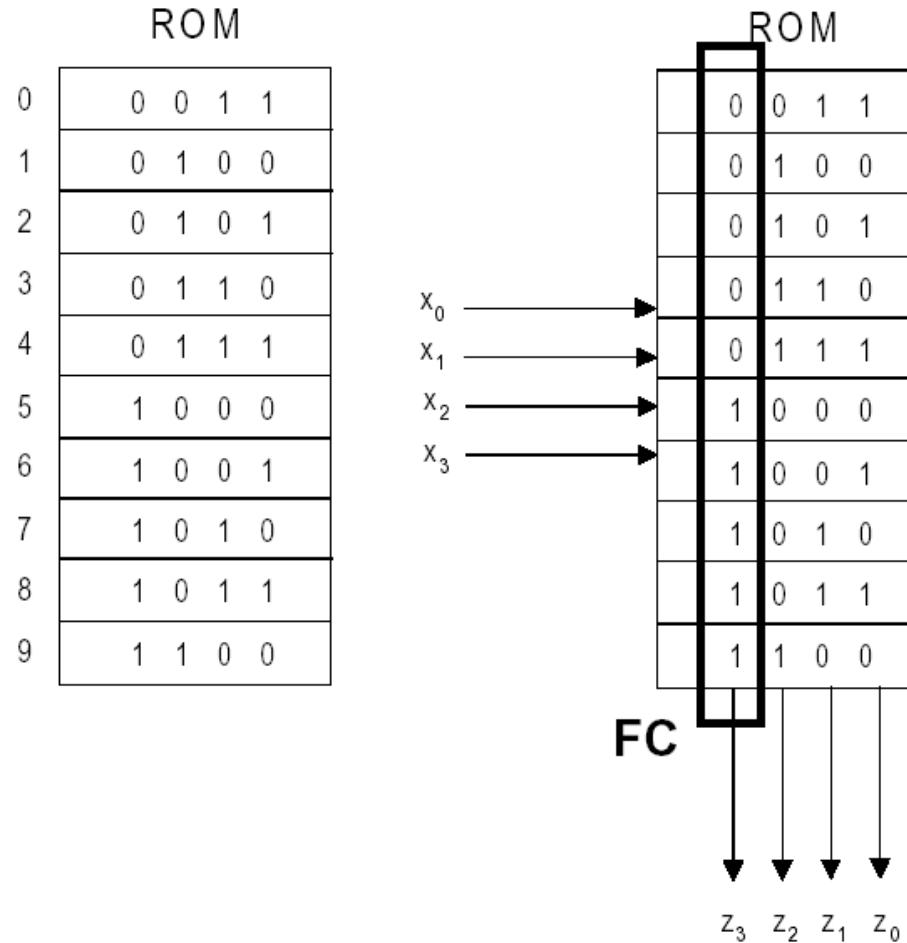
Una ROM se puede entender como una tabla que almacena datos con la siguiente estructura interna abstracta, donde cada dato ocupa una posición de la tabla denominada dirección.



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.



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



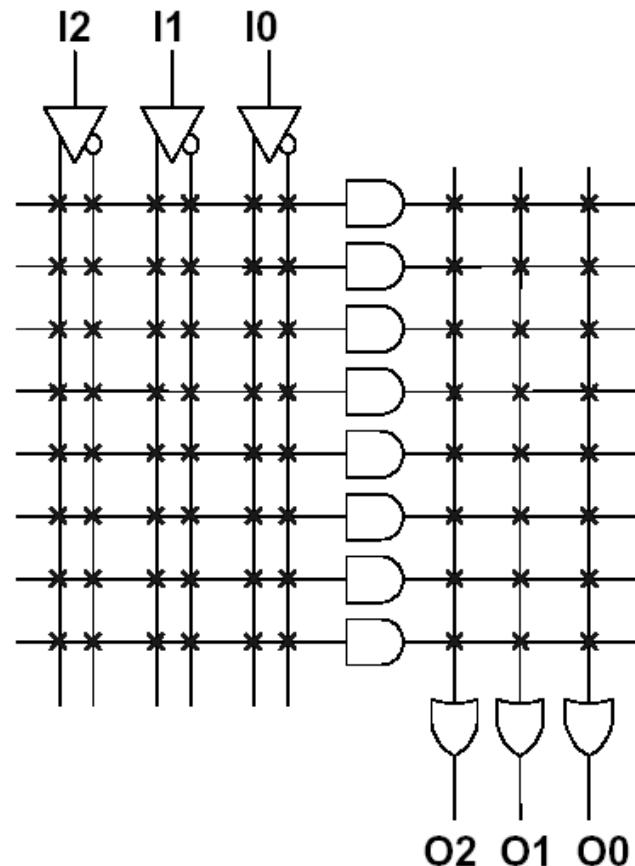


7.2. Dispositivos programables PLA, PAL

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

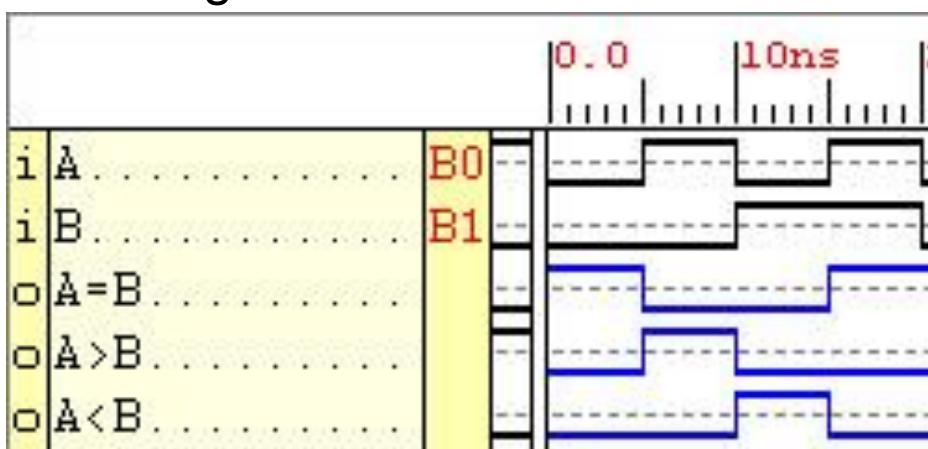
Son sistemas combinacionales que reciben dos datos de entrada A y B y los comparan en binario puro, devolviendo 3 salidas que indican si $A > B$, $A = B$ ó $A < B$

Comparador de 1 bit

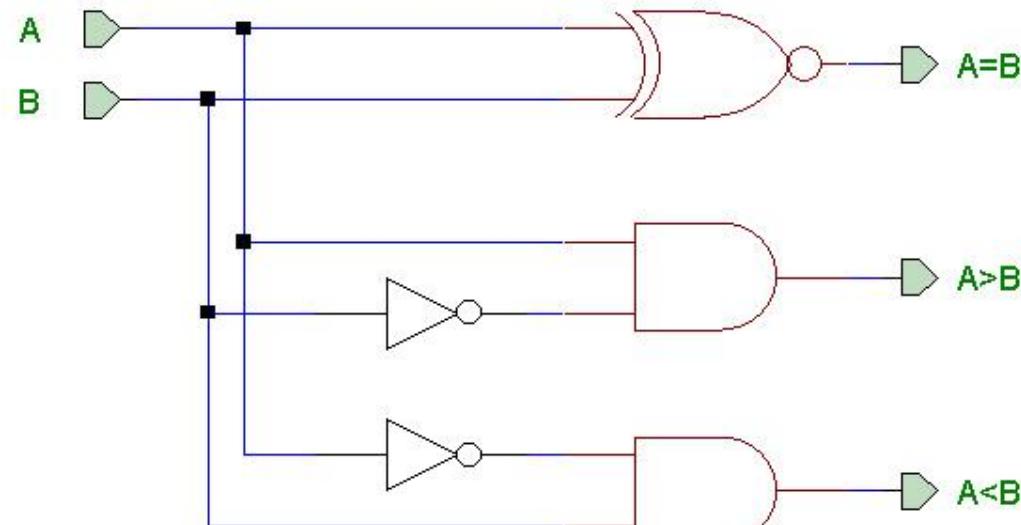
Tabla de verdad

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

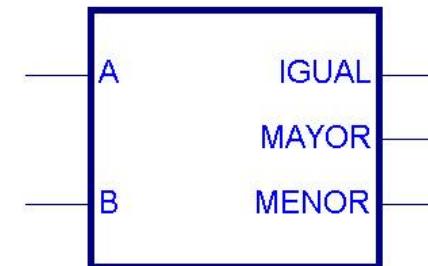
Cronograma



Circuito lógico



Símbolo





Comparadores

Comparador de 2 bits

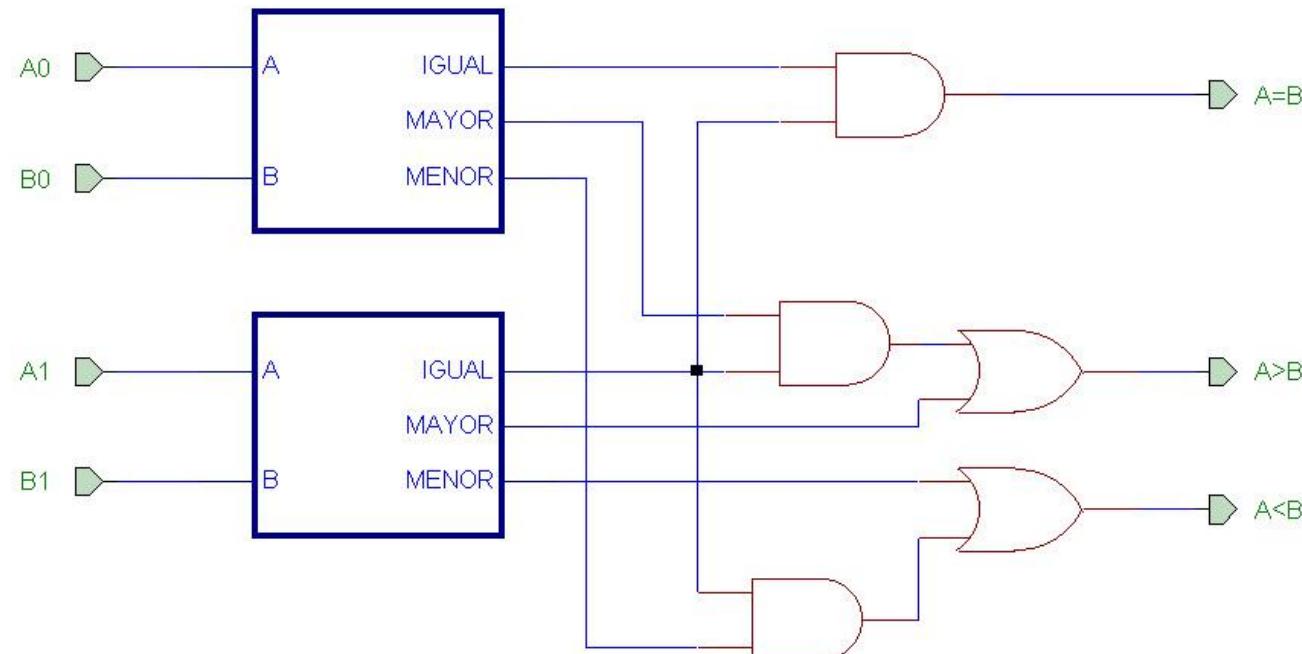
Ecuaciones

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

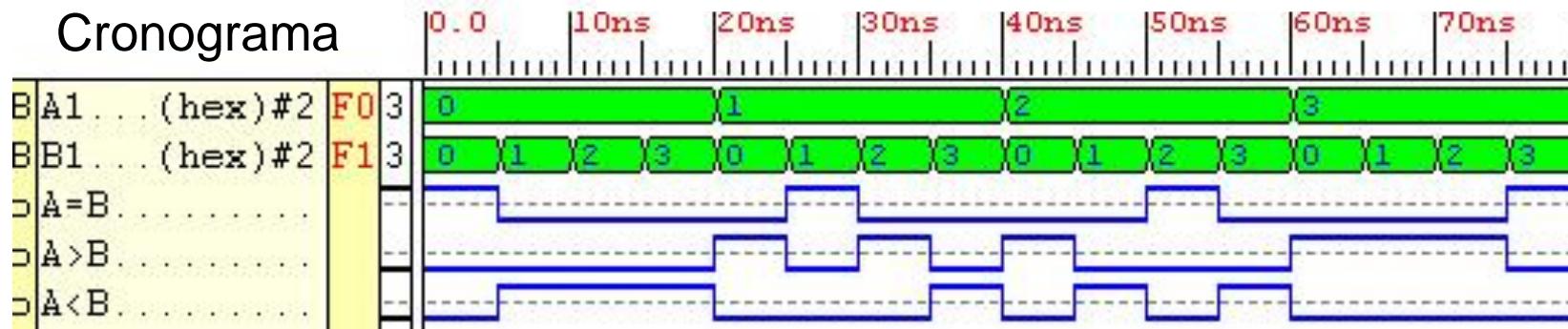
$$m_{10} = m_1 + I_1 \cdot m_0$$

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

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		
A3>B3	X	X	X	A>B	A<B	A=B	A>B	A<B	A=B
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

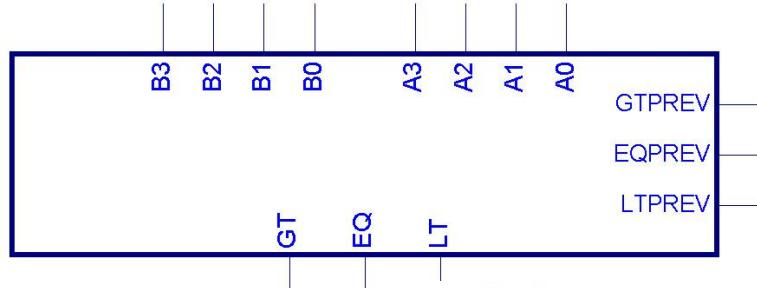


Comparadores conectables en cascada

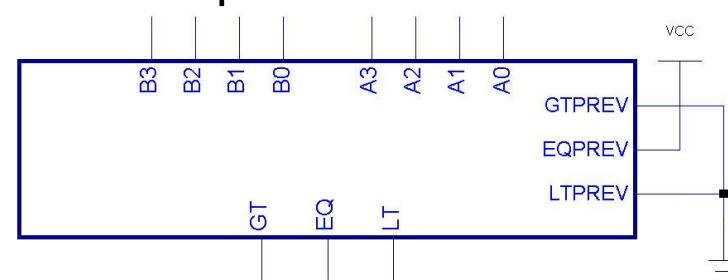
Comercial: 74HC85

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.

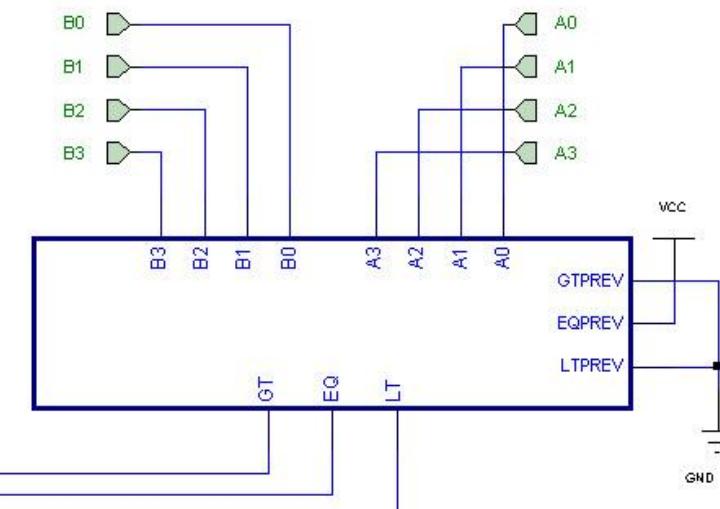
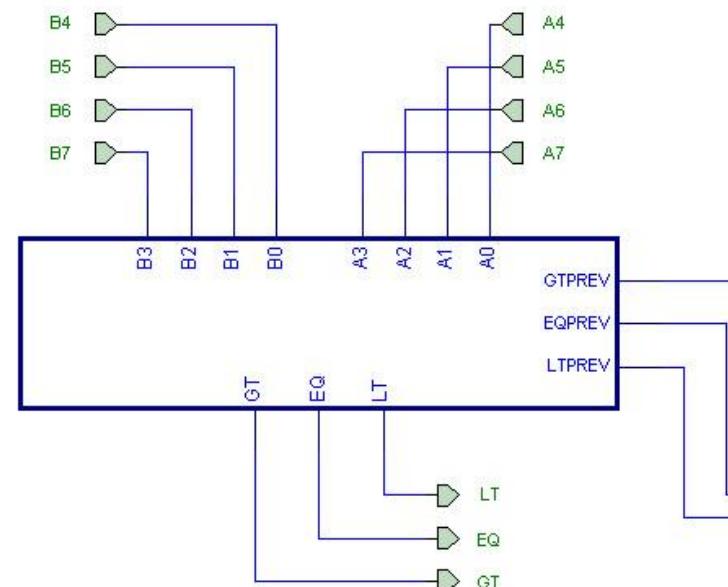
Símbolo



Comparador de 4 bits



Conexión de dos comparadores en serie

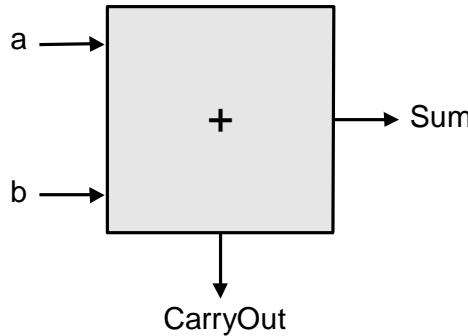




9. Sumadores: semisumador elemental

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 .

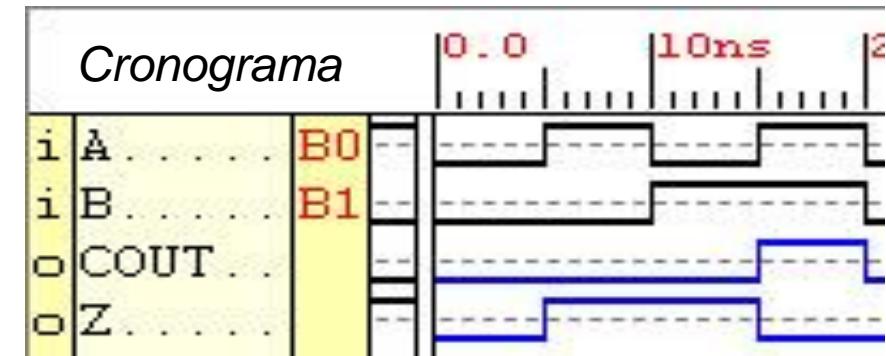
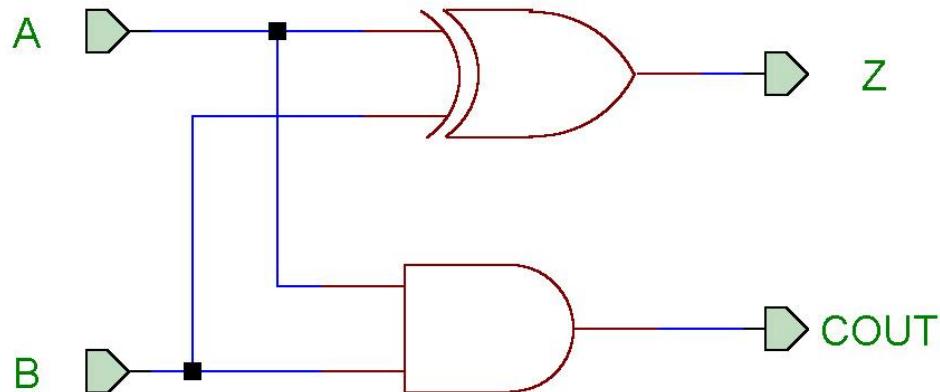
Tabla de verdad



$$c_{\text{out}} = a \cdot b$$
$$\text{Sum} = a \oplus b$$

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

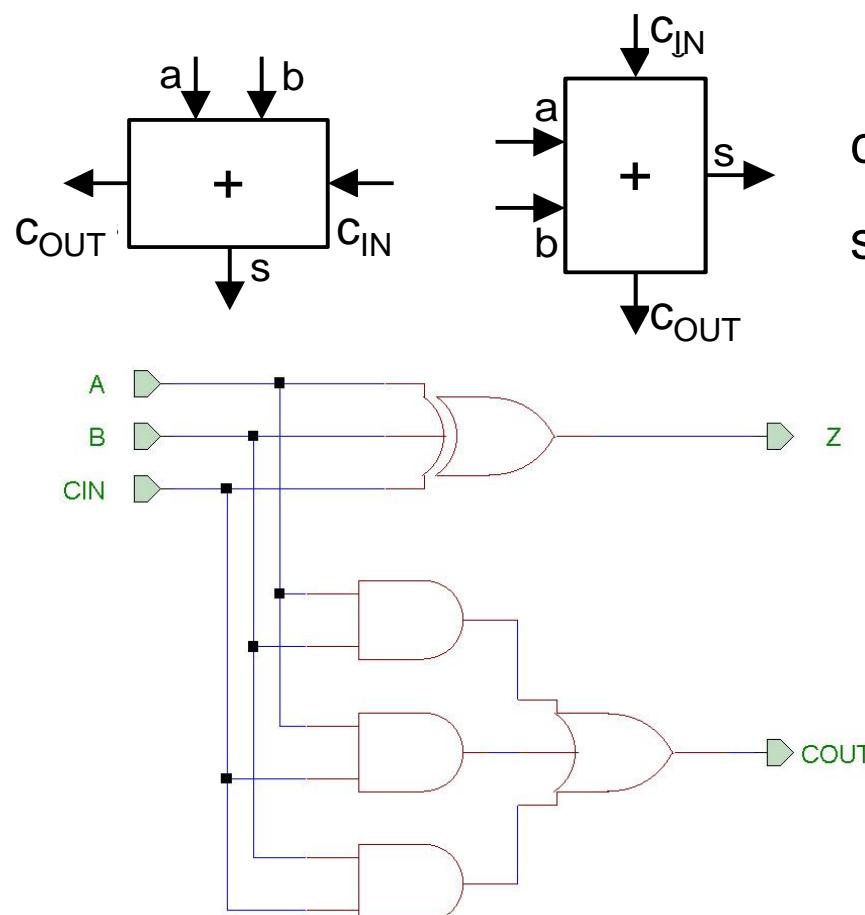


Acabamos de realizar un semi-sumador, y se llama así porque no hemos incluido el acarreo de entrada



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 .

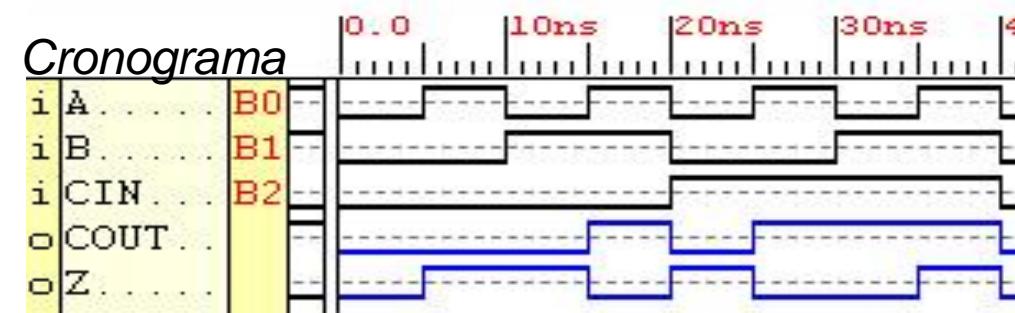


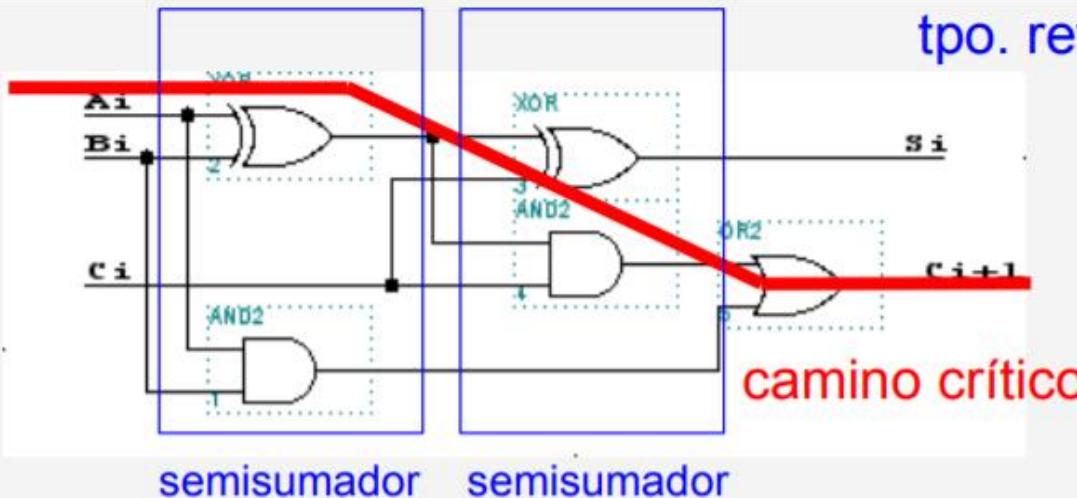
$$c_{OUT} = a \cdot b + a \cdot c_{IN} + b \cdot c_{IN}$$
$$s = a \oplus b \oplus c_{IN}$$

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

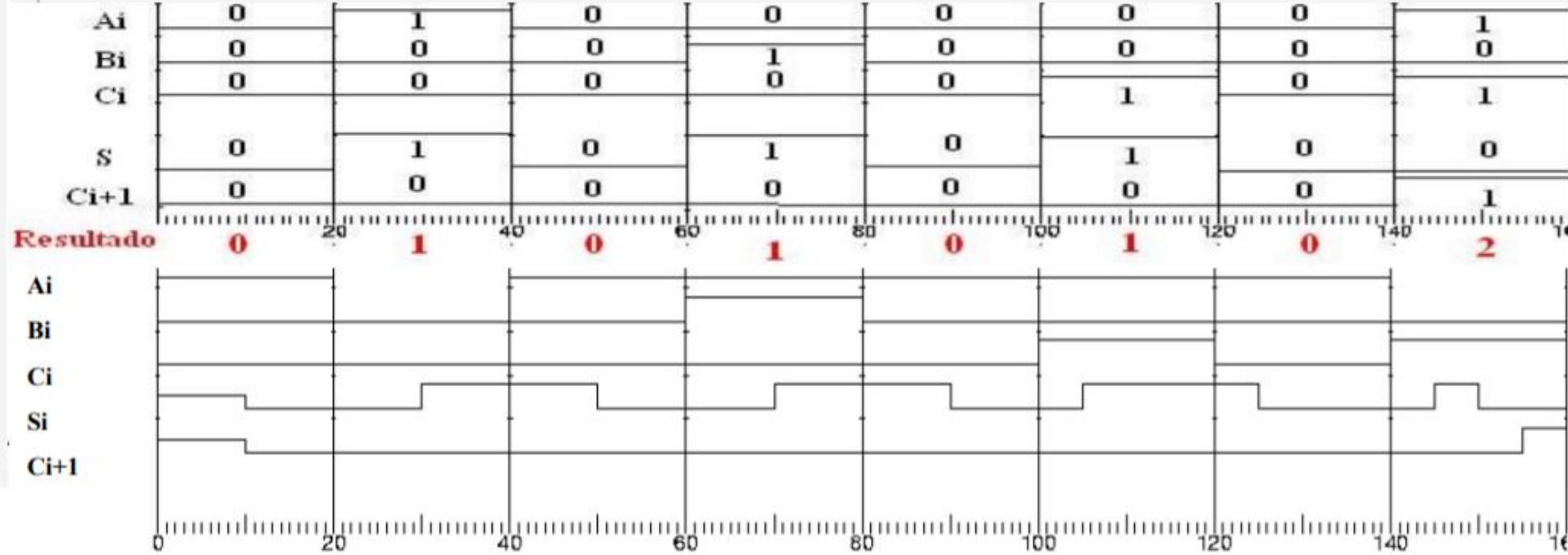




tpo. retardo por puerta = 5 ns

La salida S_i llega 10 ns de retraso respecto a las entradas A y B y sólo 5 ns respecto a la entrada C

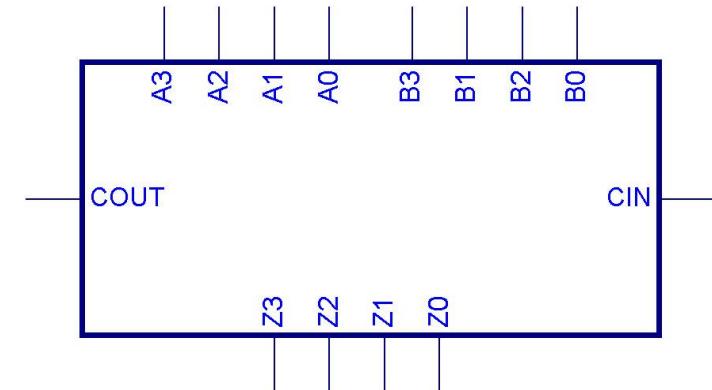
La salida C_{i+1} llega 15 ns de retraso respecto a las entradas A y B y sólo 10 ns respecto a la entrada C



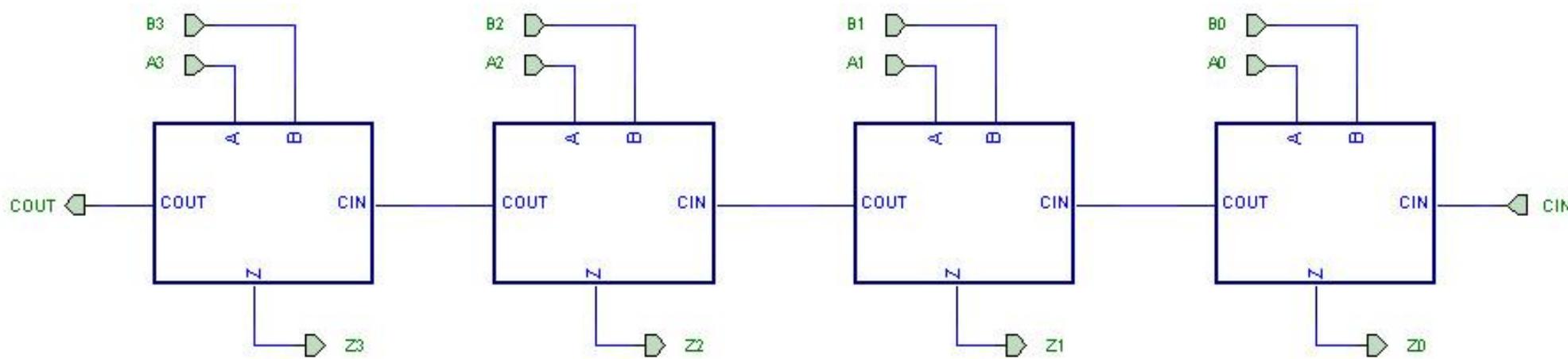


Sumador con propagación de acarreo en serie

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)

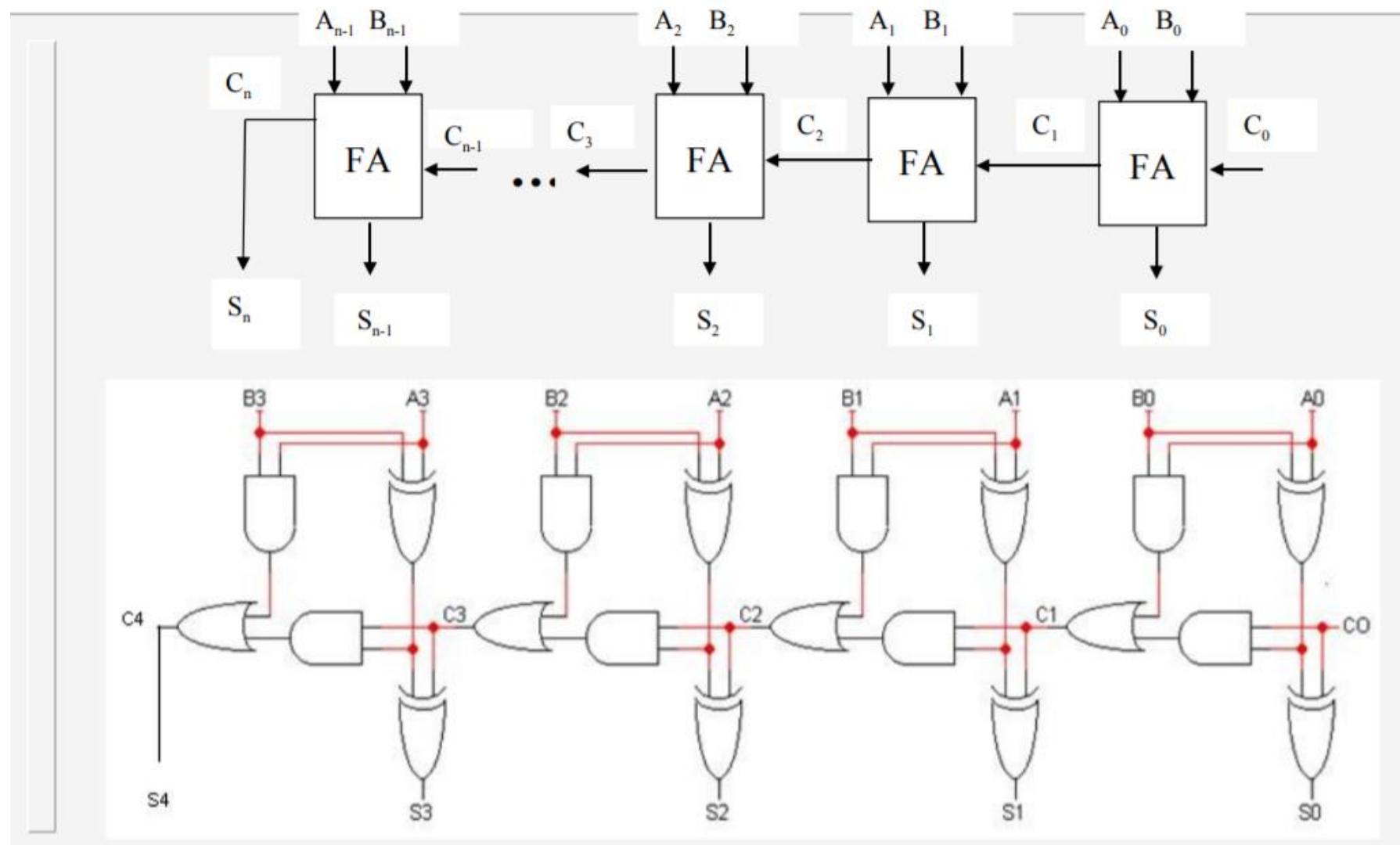


Circuito con sumadores elementales





Sumadores: Sumador paralelo con acarreo serie

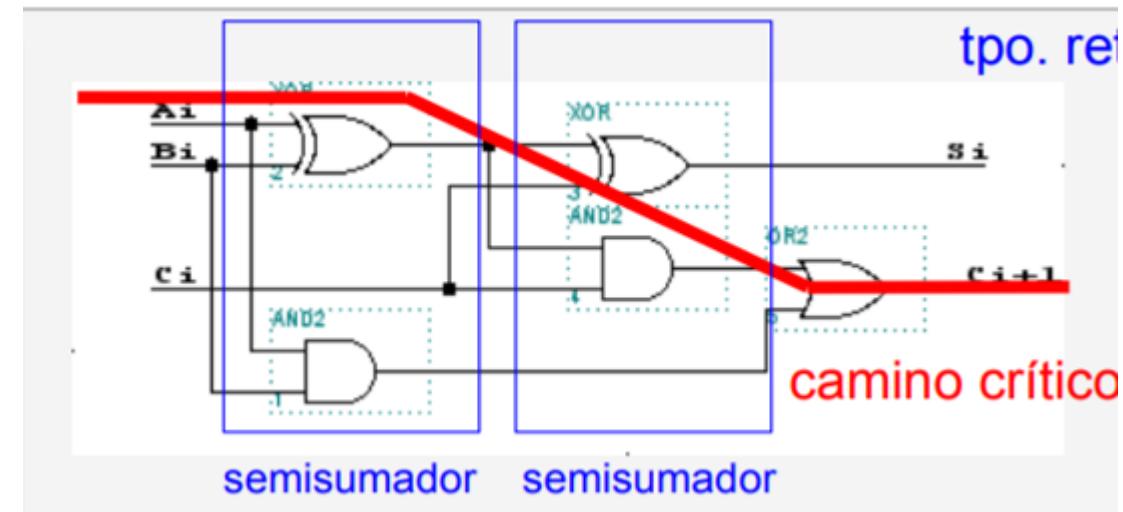




Sumadores: Sumador paralelo con acarreo serie

Cada sumador completo realiza una suma:

- Genera un acarreo que se le transmite al sumador siguiente
- Los tiempos se van acumulando
- Si t_s es el tiempo para realizar una suma y t_c el tiempo para realizar un acarreo, resulta:
- Dato en $S_0\ C_1\ S_1\ \dots\ C_2\ \dots\ S_{n-1}\ \dots\ S_n = C_n$
- Tiempo $t_s\ t_c\ t_s + t_c\ 2*t_c\ \dots\ t_s + (n-1)*t_c\ n*t_c$





Sumador con propagación de acarreo en serie

El retardo del sumador elemental es:

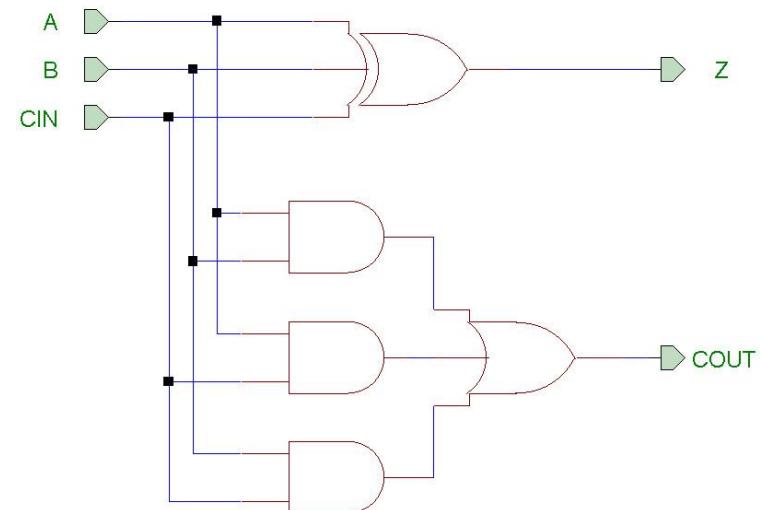
- ⇒ Z: 1 puerta.
- ⇒ COUT: 2 puertas.

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: Sumador con acarreo anticipado

- ✓ Los acarreos se evalúan anticipadamente con lógica de 2 niveles de puertas
- ✓ Las sumas se realizan posteriormente en paralelo
- ✓ En primer lugar se obtienen los términos de generación y propagación:
 $P_i = A_i \oplus B_i$
 $G_i = A_i B_i$ **Todos los términos se calculan en paralelo desde el primer momento**

Calculo del acarreo:

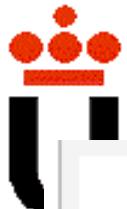
Desarrollando la fórmula iterativa $C_{i+1} = P_i C_i + G_i$ todos los acarreos dependen de propagaciones, generaciones y acarreo inicial

$$C_1 = P_0 C_0 + G_0$$

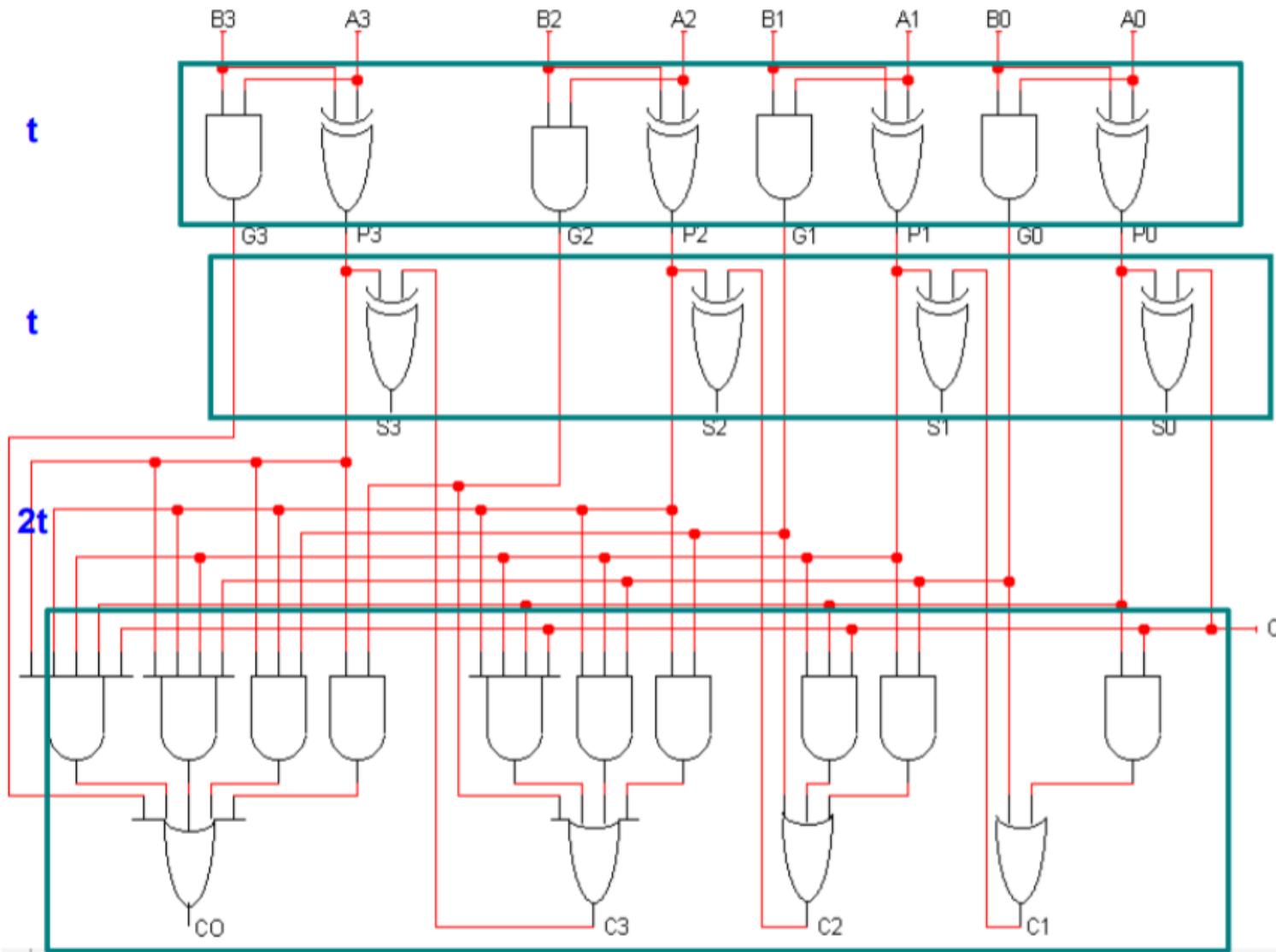
$$C_2 = P_1 C_1 + G_1 = P_1 (P_0 C_0 + G_0) + G_1 = P_1 P_0 C_0 + P_1 G_0 + G_1 \dots$$

Cálculo de las sumas

$$S_i = P_i \oplus C_i$$



Sumador: Sumador con acarreo anticipado



1.- Propagación y Generación

3.- Sumas

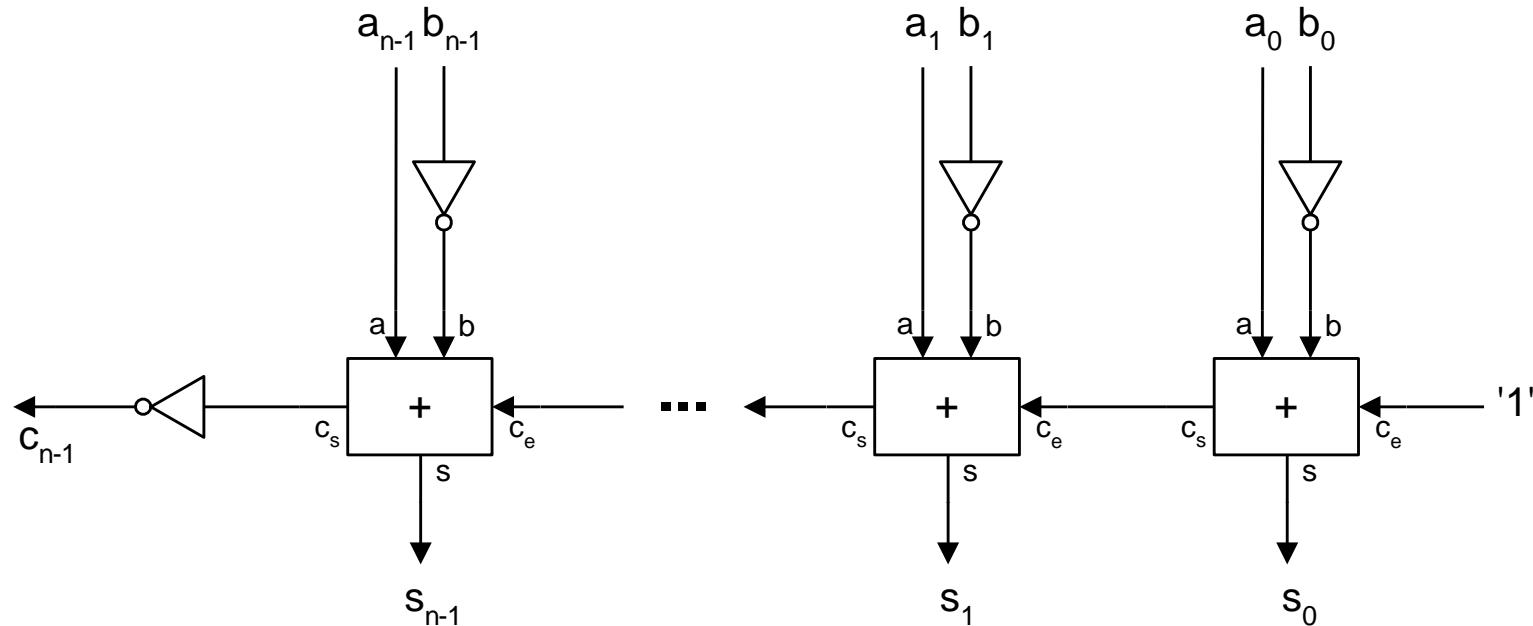
2.- Acarreos



10. Restadores binarios

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

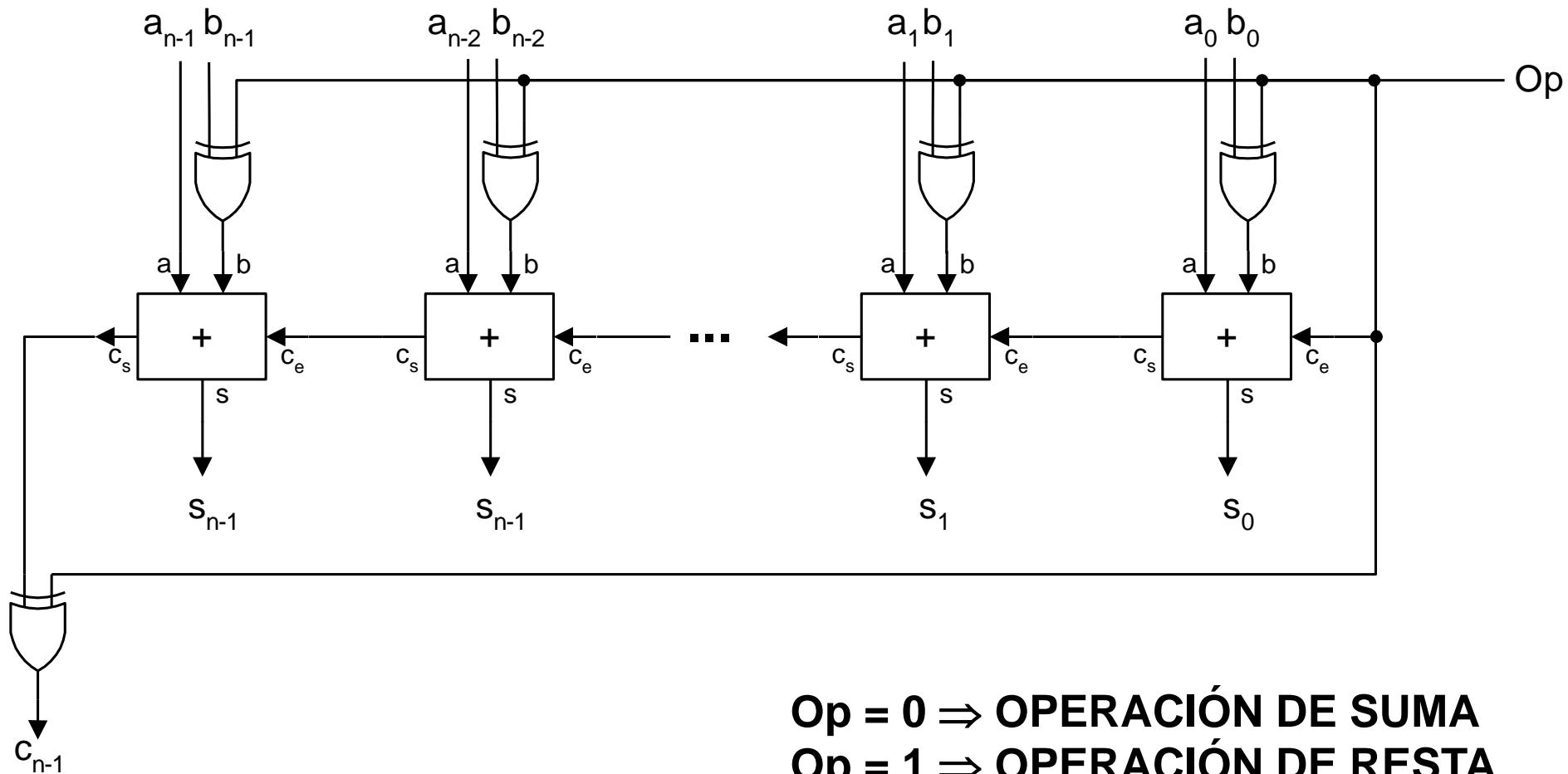
- Para complementar el sustraendo, invertimos todos sus bits e introducimos un 1 en el acarreo de entrada del sumador menos significativo.
- Por este procedimiento también habrá que invertir el acarreo de salida.
- Esto funciona tanto para binario puro como para complemento a 2 (en complemento a 2 el 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 \Rightarrow SUMADOR / RESTADOR DE N BITS.





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.

