

Tema 5. Circuitos MSI

- Multiplexores.
- Decodificadores/demultiplexores.
- Implementaciones de funciones lógicas con multiplexores y decodificadores.
- Codificadores con prioridad.
- Sumadores.
- Comparadores.
- Diseño lógico con circuitos MSI.

Multiplexores

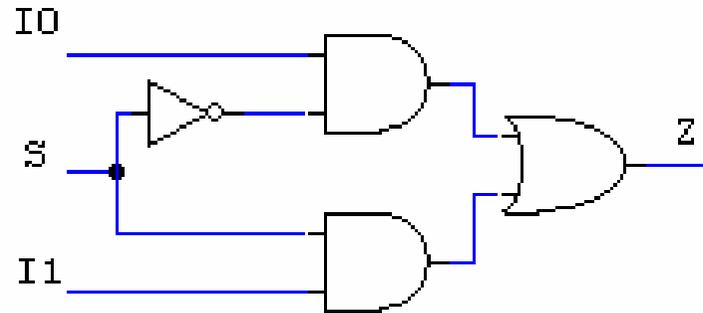
- Un multiplexor es un circuito digital que selecciona una de entre varias entradas de datos I_i y lleva su valor lógico a la única salida Z del circuito. La selección de los datos se realiza mediante una o varias entradas de control S_j . La codificación binaria resultante de las entradas S indica el índice de la entrada I que pasa a la salida.
- En circuitos digitales el número de entradas de datos es normalmente potencia de 2 (2-input MUX, 4-input MUX, 8-input MUX, etc) y el número de entradas de control debe generar las combinaciones suficientes para seleccionar todas las entradas:
 - 1 entrada de control S_0 : 2 combinaciones \Rightarrow 2-input MUX.
 - 2 entradas de control $S_1 S_0$: 4 combinaciones \Rightarrow 4-input MUX.
 - 3 entradas de control $S_2 S_1 S_0$: 8 combinaciones \Rightarrow 8-input MUX.
 - N entradas de control $S_{N-1} \dots S_1 S_0$: 2^N combinaciones \Rightarrow 2^N -input MUX.
- Un multiplexor representa en un circuito digital una situación de múltiple decisión correspondiente a una sentencia **if-else** en VHDL (2-input MUX) o una sentencia **case** (N-input MUX)

Multiplexores

2-INPUT MUX

S0	Z
0	I0
1	I1

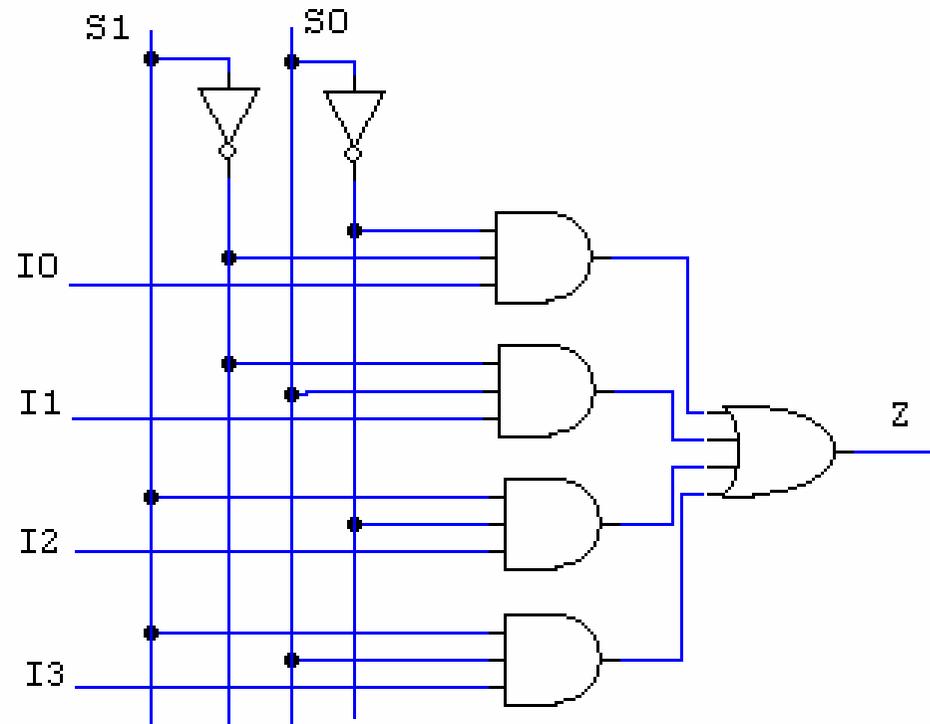
$$Z = \overline{S0} I0 + S0 I1$$



4-INPUT MUX

S1	S0	Z
0	0	I0
0	1	I1
1	0	I2
1	1	I3

$$Z = \overline{S1} \overline{S0} I0 + \overline{S1} S0 I1 + S1 \overline{S0} I2 + S1 S0 I3$$

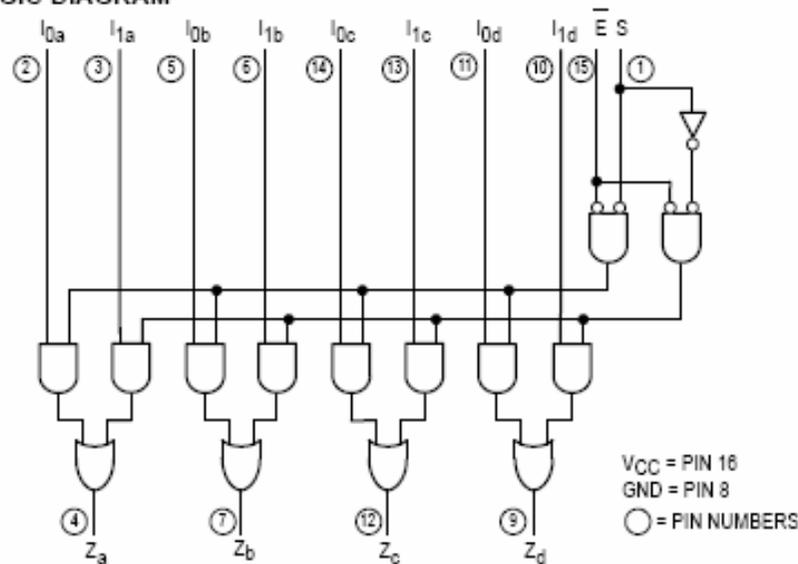


Multiplexores

Circuitos comerciales:

74'157: cuatro 2-Input MUX. Entradas común de selección y de habilitación.

LOGIC DIAGRAM



$$Z_a = \bar{E} \cdot (I_{1a} \cdot S + I_{0a} \cdot \bar{S})$$

$$Z_b = \bar{E} \cdot (I_{1b} \cdot S + I_{0b} \cdot \bar{S})$$

$$Z_c = \bar{E} \cdot (I_{1c} \cdot S + I_{0c} \cdot \bar{S})$$

$$Z_d = \bar{E} \cdot (I_{1d} \cdot S + I_{0d} \cdot \bar{S})$$

AC CHARACTERISTICS (T_A = 25°C)

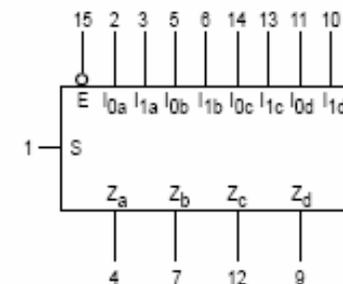
Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
t _{PLH} t _{PHL}	Propagation Delay Data to Output		9.0 9.0	14 14	ns	V _{CC} = 5.0 V C _L = 15 pF
t _{PLH} t _{PHL}	Propagation Delay Enable to Output		13 14	20 21	ns	
t _{PLH} t _{PHL}	Propagation Delay Select to Output		15 18	23 27	ns	

TRUTH TABLE

ENABLE	SELECT INPUT	INPUTS		OUTPUT
E	S	I ₀	I ₁	Z
H	X	X	X	L
L	H	X	L	L
L	H	X	H	H
L	L	L	X	L
L	L	H	X	H

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

LOGIC SYMBOL

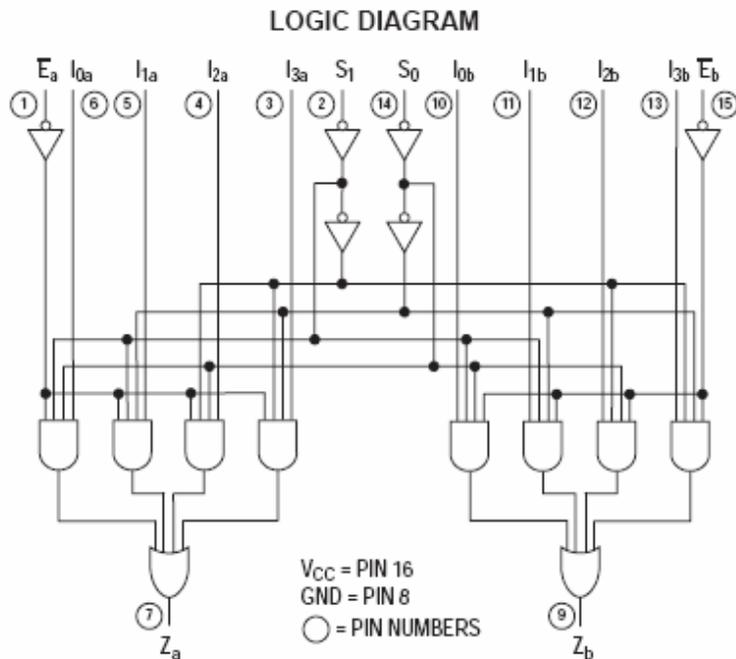


V_{CC} = PIN 16
GND = PIN 8

Multiplexores

Circuitos comerciales:

74'153: dos 4-Input MUX. Entradas común de selección, entradas diferentes de habilitación.



$$Z_a = E_a \cdot (I_{0a} \cdot \bar{S}_1 \cdot \bar{S}_0 + I_{1a} \cdot \bar{S}_1 \cdot S_0 + I_{2a} \cdot S_1 \cdot \bar{S}_0 + I_{3a} \cdot S_1 \cdot S_0)$$

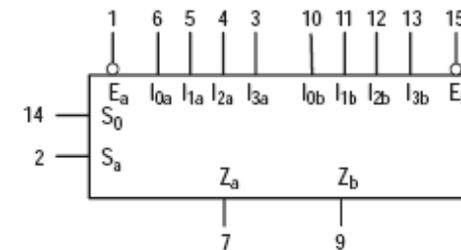
$$Z_b = E_b \cdot (I_{0b} \cdot \bar{S}_1 \cdot \bar{S}_0 + I_{1b} \cdot \bar{S}_1 \cdot S_0 + I_{2b} \cdot S_1 \cdot \bar{S}_0 + I_{3b} \cdot S_1 \cdot S_0)$$

TRUTH TABLE

SELECT INPUTS		INPUTS (a or b)					OUTPUT
S ₀	S ₁	E	I ₀	I ₁	I ₂	I ₃	Z
X	X	H	X	X	X	X	L
L	L	L	L	X	X	X	L
L	L	L	H	X	X	X	H
H	L	L	X	L	X	X	L
H	L	L	X	H	X	X	H
L	H	L	X	X	L	X	L
L	H	L	X	X	H	X	H
H	H	L	X	X	X	L	L
H	H	L	X	X	X	H	H

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

LOGIC SYMBOL

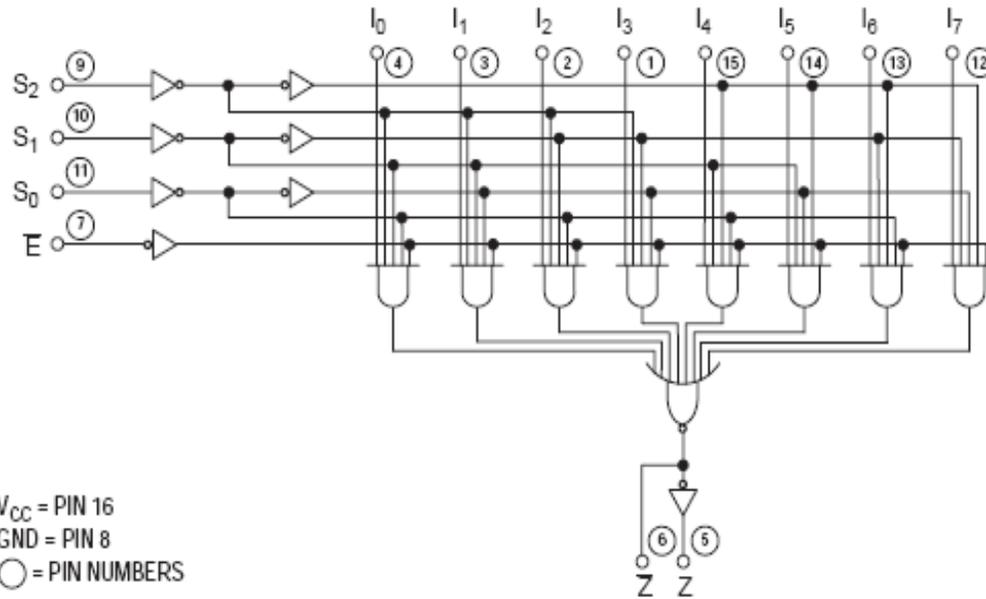


V_{CC} = PIN 16
GND = PIN 8

Multiplexores

Circuitos comerciales: 74'151: un 8-Input MUX.

LOGIC DIAGRAM

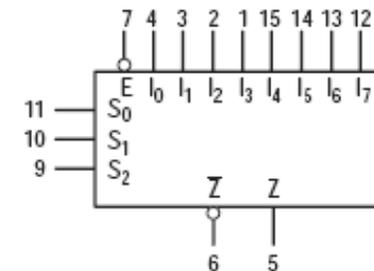


$$Z = E \cdot (I_0 \cdot \bar{S}_0 \cdot \bar{S}_1 \cdot \bar{S}_2 + I_1 \cdot S_0 \cdot \bar{S}_1 \cdot \bar{S}_2 + I_2 \cdot \bar{S}_0 \cdot S_1 \cdot \bar{S}_2 + I_3 \cdot S_0 \cdot S_1 \cdot \bar{S}_2 + I_4 \cdot \bar{S}_0 \cdot \bar{S}_1 \cdot S_2 + I_5 \cdot S_0 \cdot \bar{S}_1 \cdot S_2 + I_6 \cdot \bar{S}_0 \cdot S_1 \cdot S_2 + I_7 \cdot S_0 \cdot S_1 \cdot S_2)$$

TRUTH TABLE

E	S ₂	S ₁	S ₀	I ₀	I ₁	I ₂	I ₃	I ₄	I ₅	I ₆	I ₇	Z	Z
H	X	X	X	X	X	X	X	X	X	X	X	H	L
L	L	L	L	L	X	X	X	X	X	X	X	H	L
L	L	L	L	H	X	X	X	X	X	X	X	L	H
L	L	L	H	X	L	X	X	X	X	X	X	H	L
L	L	L	H	X	H	X	X	X	X	X	X	L	H
L	L	H	L	X	X	L	X	X	X	X	X	H	L
L	L	H	L	X	X	H	X	X	X	X	X	L	H
L	L	H	H	X	X	X	L	X	X	X	X	H	L
L	L	H	H	X	X	X	H	X	X	X	X	L	H
L	H	L	L	X	X	X	X	L	X	X	X	H	L
L	H	L	L	X	X	X	X	H	X	X	X	L	H
L	H	L	H	X	X	X	X	X	L	X	X	H	L
L	H	L	H	X	X	X	X	X	H	X	X	L	H
L	H	H	L	X	X	X	X	X	X	L	X	H	L
L	H	H	L	X	X	X	X	X	X	H	X	L	H
L	H	H	H	X	X	X	X	X	X	X	L	H	L
L	H	H	H	X	X	X	X	X	X	X	H	L	H

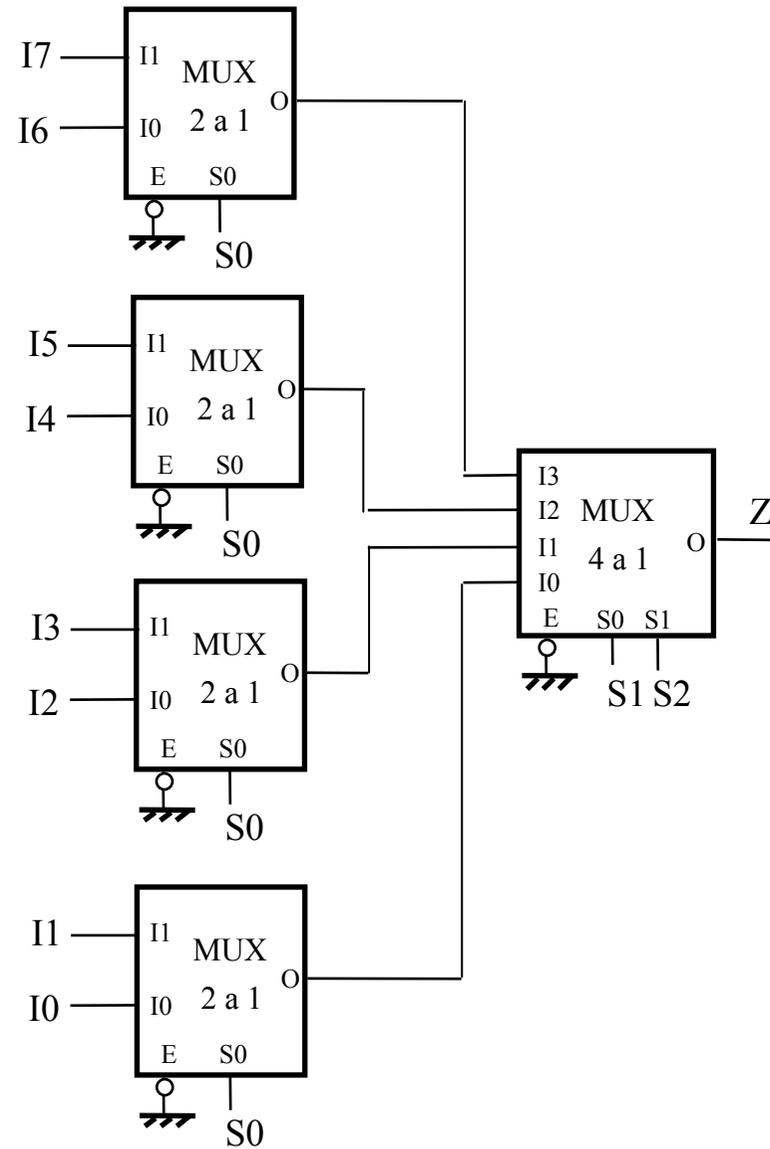
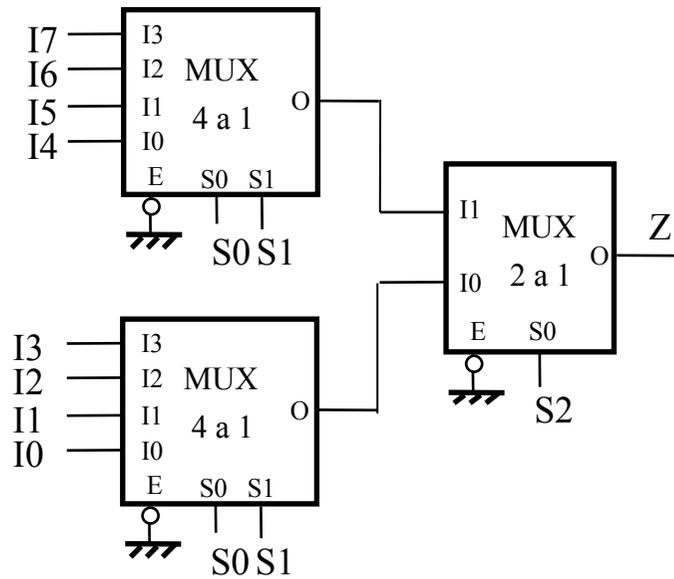
H = HIGH Voltage Level
 L = LOW Voltage Level
 X = Don't Care



Multiplexores

Desarrollo de N-input MUXs en base a M-input MUX ($N > M$)

Desarrollo de 8-input MUXs



Multiplexores

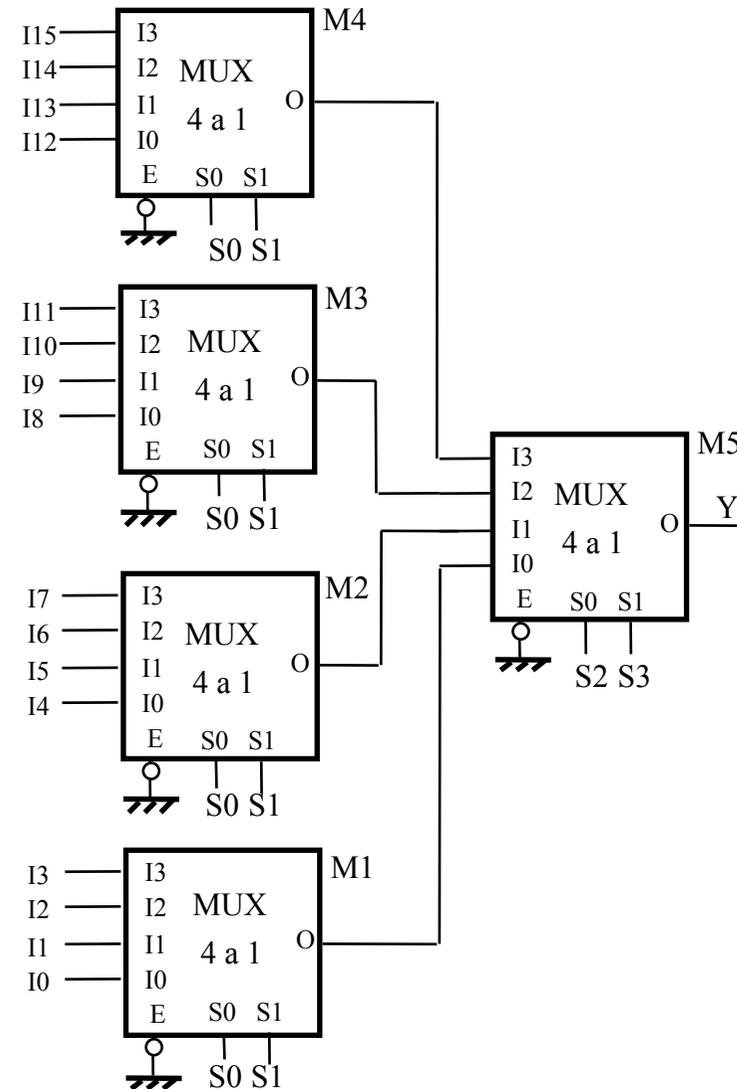
Desarrollo de 16-input MUXs en base a 4-Input MUXs

Si $(S_3S_2) = 00$, $Y \leq I_0$ de M5 que está conectada a la salida de M1.
En función de (S_1S_0) se selecciona en M1 las entradas I3-I0.

Si $(S_3S_2) = 01$, $Y \leq I_1$ de M5 que está conectada a la salida de M2.
En función de (S_1S_0) se selecciona en M2 las entradas I4-I7.

Si $(S_3S_2) = 10$, $Y \leq I_2$ de M5 que está conectada a la salida de M3.
En función de (S_1S_0) se selecciona en M3 las entradas I8-I11.

Si $(S_3S_2) = 11$, $Y \leq I_3$ de M5 que está conectada a la salida de M4.
En función de (S_1S_0) se selecciona en M4 las entradas I12-I15.



Multiplexores

Modelo VHDL de un 4-input MUX: sentencia case

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

entity mux4 is
port (I: in std_logic_vector(3 downto 0); -- Entradas de datos
      S: in std_logic_vector(1 downto 0); -- Entradas de selección
      E: in std_logic; -- Entrada de habilitación
      Z: out std_logic); -- Salida
end mux4;
```

```
architecture comportamiento of mux4 is
begin
process (I, S, E)
begin
if E = '1' then
Z <= '0';
else
case S is
when "00" => Z <= I(0);
when "01" => Z <= I(1);
when "10" => Z <= I(2);
when "11" => Z <= I(3);
when others => Z <= I(3);
end case;
end if;
end process;
end comportamiento;
```

2-Input MUX

```
architecture mux2_1 of mux2 is
begin
process (I0, I1, S)
begin
if ( S = '0' ) then Z <= I0;
else Z <= I1;
end if;
end process;
end mux2_1;
```

Decodificadores/demultiplexores

- Un decodificador es un circuito que convierte la información (la dirección) de entrada A de N bits codificada en un código de tipo binario, en M líneas de salida O_i , donde M es el número de combinaciones del código de entrada. En códigos binarios para N bits de entrada el número de salidas es $M = 2^N$.
Para cada dato binario de entrada se fija una única salida O_i a 1, cuyo índice i corresponde al valor binario del dato de entrada.
- Un demultiplexor es un circuito que pasa el valor lógico de una entrada I a una de sus M salidas O_i . Para determinar la salida a la que se envía la entrada I se usan N entradas de selección (o dirección) A_j . La codificación binaria resultante de las entradas A indica el índice i de la salida O_i a la que se envía la entrada I . Para N bits de dirección el número de salidas posibles donde enviar I es $M = 2^N$.
- Los dos problemas lógicos son similares (aunque no idénticos) y en la práctica se usa un mismo circuito para realizar las dos funciones: decodificación de N a M y demultiplexado 1 de M .

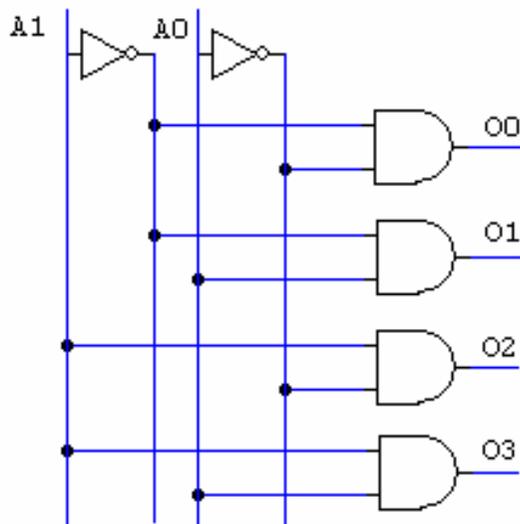
Decodificadores/demultiplexores

2 a 4 DEC

A1	A0	O0	O1	O2	O3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$$O0 = \bar{A}1 \bar{A}0 \quad O1 = \bar{A}1 A0$$

$$O2 = A1 \bar{A}0 \quad O3 = A1 A0$$

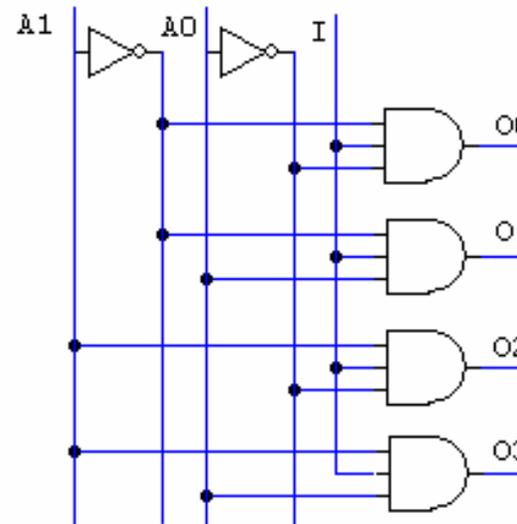


1 de 4 DEMUX

A1	A0	O0	O1	O2	O3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$$O0 = \bar{A}1 \bar{A}0 I \quad O1 = \bar{A}1 A0 I$$

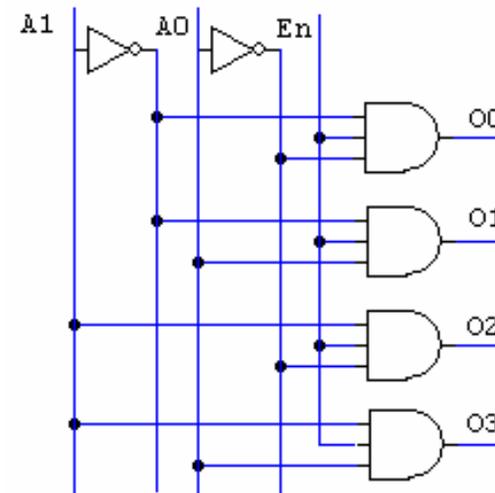
$$O2 = A1 \bar{A}0 I \quad O3 = A1 A0 I$$



Decodificadores/demultiplexores

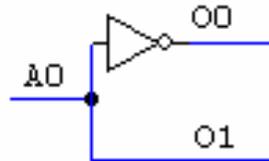
- La entrada de datos de un demultiplexor corresponde a una entrada de habilitación de un decodificador. Por tanto, el demultiplexor y el decodificador con Enable se realizan con el mismo circuito.

E	A1	A0	O0	O1	O2	O3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

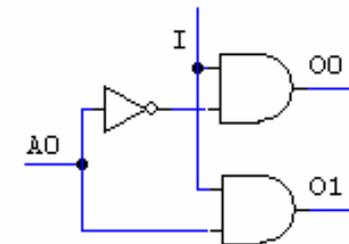


- Un decodificador 1 a 2 sin habilitador puede realizarse sólo con un inversor, con habilitador (o un demultiplexor 1 de 2) no.

A0	O0	O1
0	1	0
1	0	1

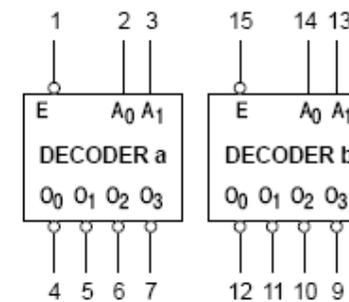
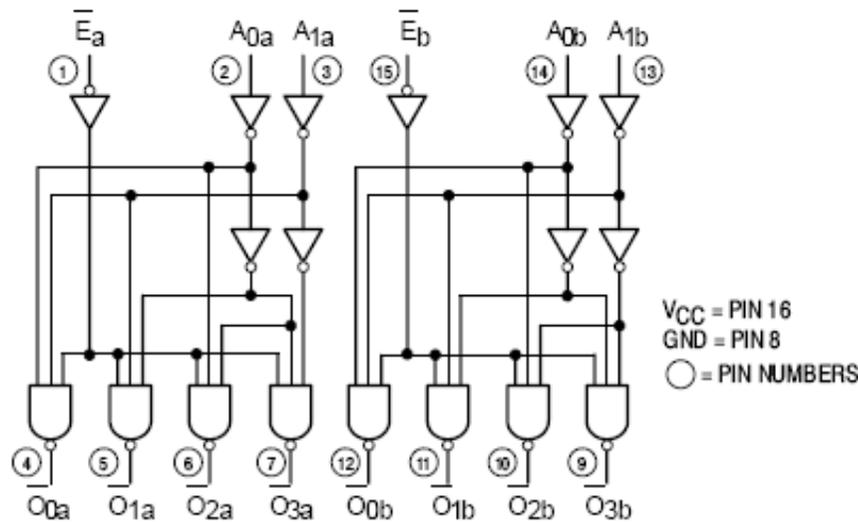


A0	O0	O1
0	1	0
1	0	1



Decodificadores/demultiplexores

Circuitos comerciales: 74'139: dos 2 a 4 DEC/1 de 4 DEMUX.
 Las salidas y las entradas de habilitación están en polaridad negativa.

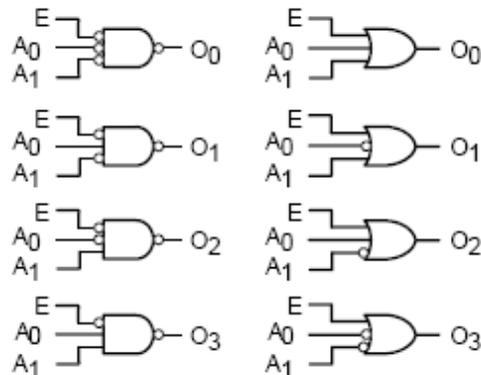


$V_{CC} = \text{PIN } 16$
 $GND = \text{PIN } 8$

TRUTH TABLE

INPUTS			OUTPUTS			
\bar{E}	A_0	A_1	\bar{O}_0	\bar{O}_1	\bar{O}_2	\bar{O}_3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	H	L	H	L	H	H
L	L	H	H	H	L	H
L	H	H	H	H	H	L

H = HIGH Voltage Level
 L = LOW Voltage Level
 X = Don't Care

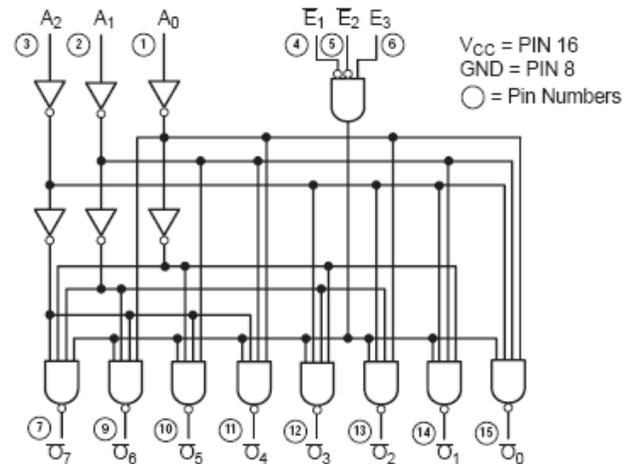
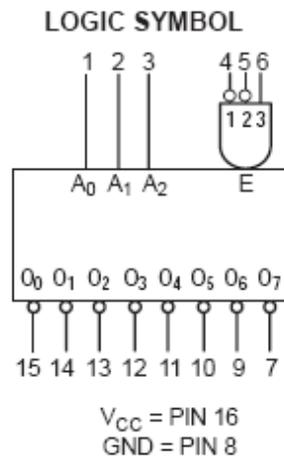


AC CHARACTERISTICS ($T_A = 25^\circ\text{C}$)

Symbol	Parameter	Levels of Delay	Limits			Unit	Test Conditions
			Min	Typ	Max		
t_{PLH} t_{PHL}	Propagation Delay Address to Output	2		13 22	20 33	ns	$V_{CC} = 5.0 \text{ V}$ $C_L = 15 \text{ pF}$
t_{PLH} t_{PHL}	Propagation Delay Address to Output	3		18 25	29 38	ns	
t_{PLH} t_{PHL}	Propagation Delay Enable to Output	2		16 21	24 32	ns	

Decodificadores/demultiplexores

Circuitos comerciales: 74'138: un 3 a 8 DEC/ 1 de 8 DEMUX.
 Las salidas de los decodificadores están en polaridad negativa.
 3 entradas de habilitación, 2 de polaridad negativa y 1 positiva.



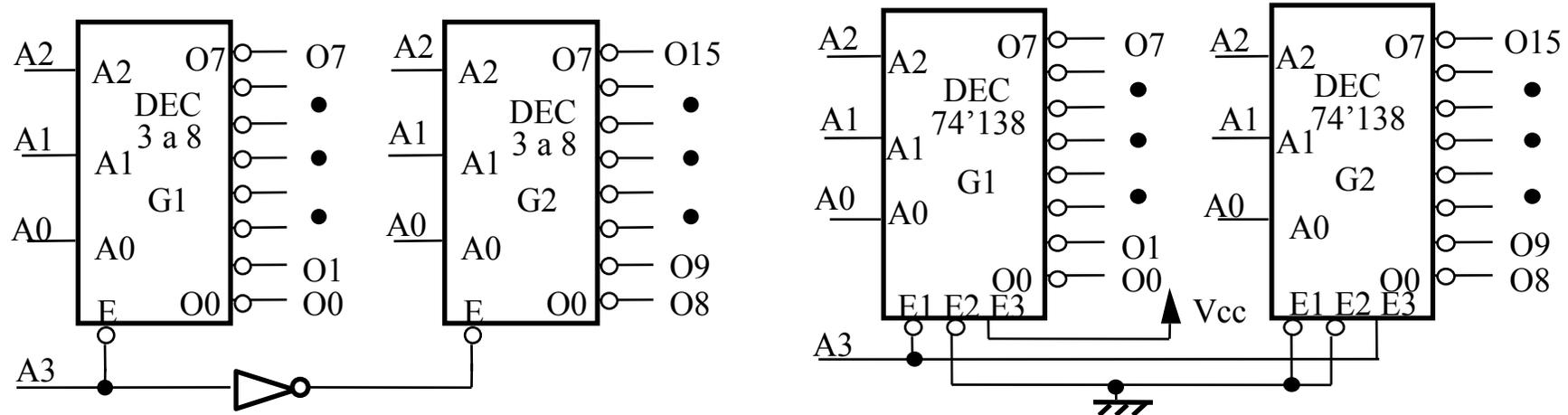
TRUTH TABLE

INPUTS						OUTPUTS							
E_1	E_2	E_3	A_0	A_1	A_2	\bar{O}_0	\bar{O}_1	\bar{O}_2	\bar{O}_3	\bar{O}_4	\bar{O}_5	\bar{O}_6	\bar{O}_7
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

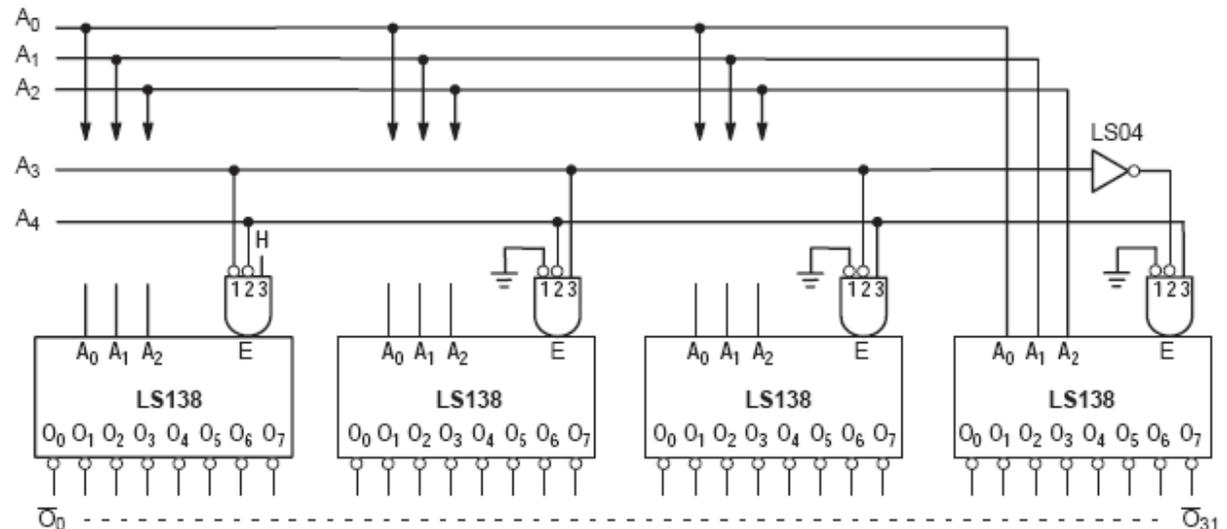
H = HIGH Voltage Level
 L = LOW Voltage Level
 X = Don't Care

Decodificadores/demultiplexores

Desarrollo de un 4 a 16 DEC en base a dos 3 a 8 DEC. El inversor opera como un 1 a 2 DEC.



Desarrollo de un 5 a 32 DEC en base a cuatro 74'138 con un único inversor, gracias a los 3 habilitadores.



Decodificadores/demultiplexores

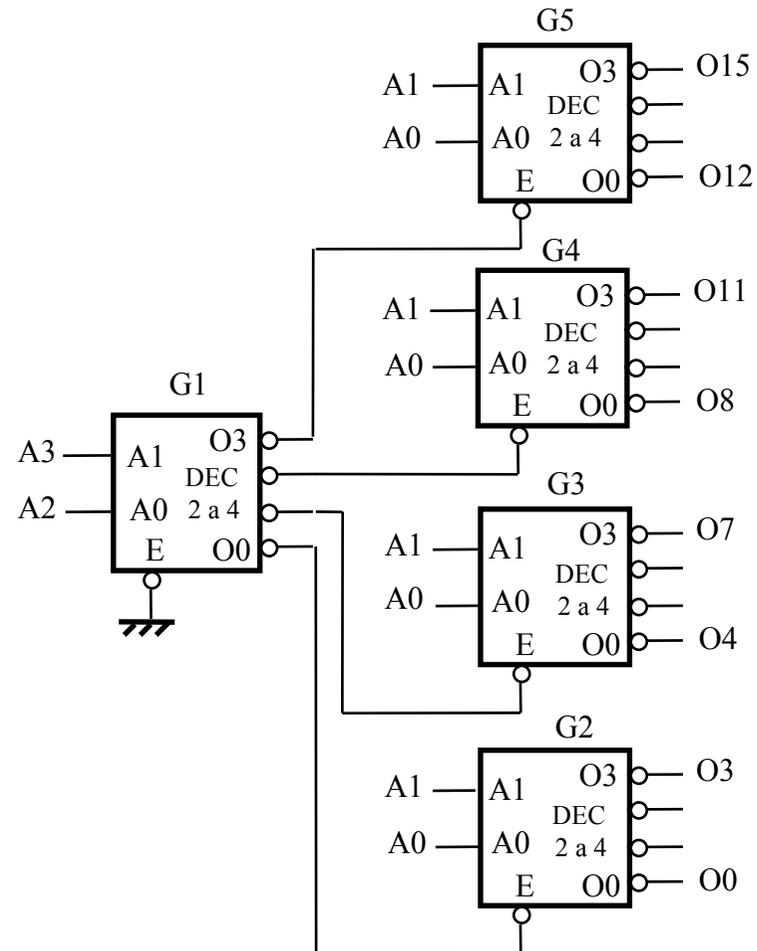
Desarrollo de 4 a 16 DEC en base a 2 a 4 DEC.

Si $(A_3A_2) = 00$, O_0 de G_1 es 1 (L) y G_2 está habilitado y en él, en función de (A_1A_0) se obtienen las salidas O_3-O_0 . G_3 , G_4 , G_5 están deshabilitados: sus salidas son 0.

Si $(A_3A_2) = 01$, O_1 de G_1 es 1 (L) y G_3 está habilitado y en él, en función de (A_1A_0) se obtienen las salidas O_7-O_4 . G_1 , G_4 , G_5 están deshabilitados: sus salidas son 0.

Si $(A_3A_2) = 10$, O_2 de G_1 es 1 (L) y G_4 está habilitado y en él en función de (A_1A_0) se obtienen las salidas $O_{11}-O_8$. G_1 , G_3 , G_5 están deshabilitados: sus salidas son 0.

Si $(A_3A_2) = 11$, O_3 de G_1 es 1 (L) y G_5 está habilitado y en él en función de (A_1A_0) se obtienen las salidas $O_{15}-O_{12}$. G_2 , G_3 , G_4 están deshabilitados: sus salidas son 0.



Decodificadores/demultiplexores

Modelo VHDL de un 2 a 4 DEC (ó 1 de 4 DEMUX)

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

entity dec2to4 is
port (A: in std_logic_vector(1 downto 0);    -- Entradas de dirección
      E: in std_logic;                       -- Entrada de habilitación
      O: out std_logic_vector(3 downto 0)); -- Salidas
end dec2to4;
```

```
architecture DEC of dec2to4 is
begin
process (A, E)
begin
if E = '0' then
O <= "0000";
else
case A is
when "00" => O <= "0001";
when "01" => O <= "0010";
when "10" => O <= "0100";
when "11" => O <= "1000";
when others => O <= "0000";
end case;
end if;
end process;
end DEC;
```

```
architecture DEMUX of dec2to4 is
begin
process (A, E)
begin
case A is
when "00" => O(0) <= E; O(1) <= '0';
              O(2) <= '0'; O(3) <= '0';
when "01" => O(0) <= '0'; O(1) <= E;
              O(2) <= '0'; O(3) <= '0';
when "10" => O(0) <= '0'; O(1) <= '0';
              O(2) <= E; O(3) <= '0';
when "11" => O(0) <= '0'; O(1) <= '0';
              O(2) <= '0'; O(3) <= E;
when others => O(0) <= '0'; O(1) <= '0';
              O(2) <= '0'; O(3) <= '0';
end case;
end process;
end DEMUX;
```

Implementación de funciones lógicas con decodificadores

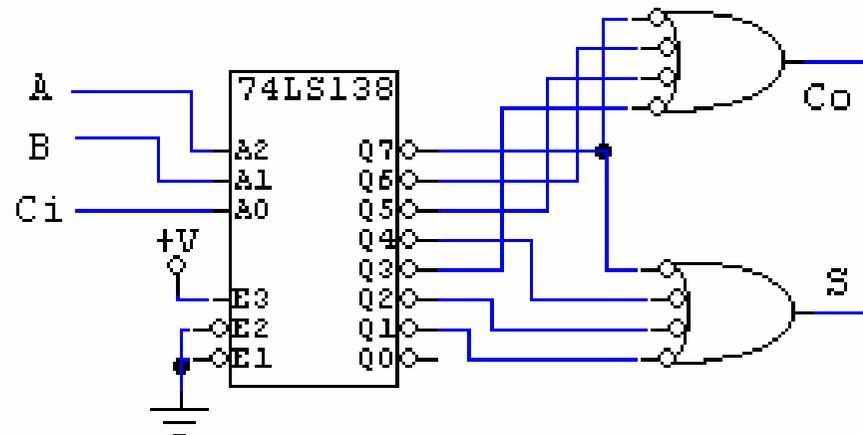
- Los decodificadores permiten implementar funciones lógicas desde sus formas canónicas. Aplicando las entradas a las entradas de dirección del decodificador, cada una de las salidas del decodificador corresponde a cada uno de los minterms de la función lógica: O_0 es el minterm 0 (m_0), O_1 el minterm 1 (m_1), etc. Se puede hacer una forma canónica SOP mediante la OR de los minterms o 1s de la función lógica, es decir mediante el OR de las salidas O_i correspondientes a los minterms de la función. En principio hay que usar un decodificador de N a 2^N , siendo N el número de entradas de la función lógica.

Sumador completo

A	B	Ci	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

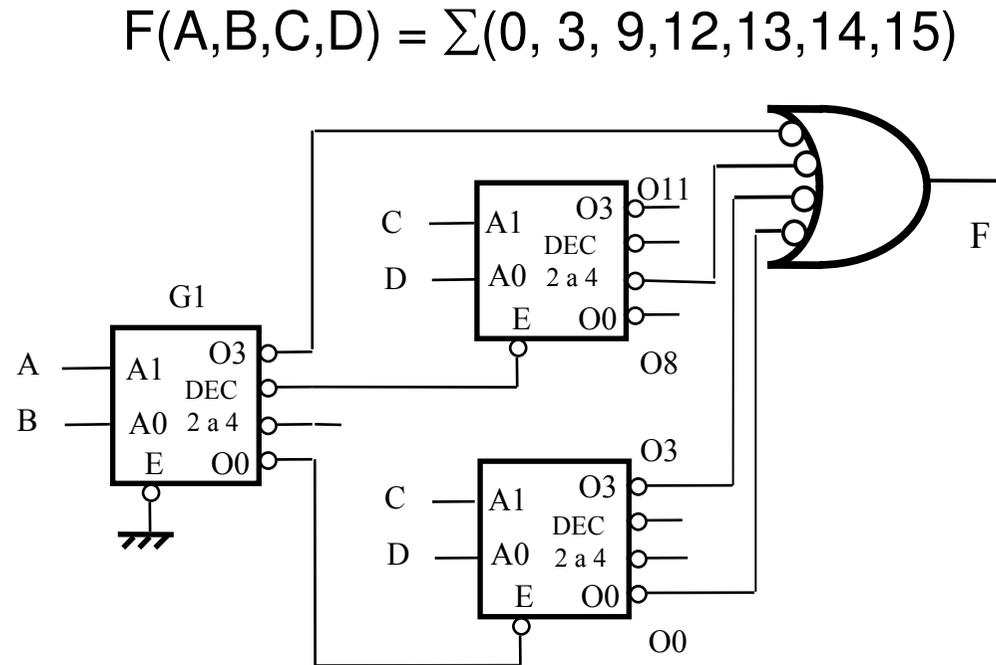
$$S(A, B, C_i) = \sum(1, 2, 4, 7)$$

$$C_o(A, B, C_i) = \sum(3, 5, 6, 7)$$



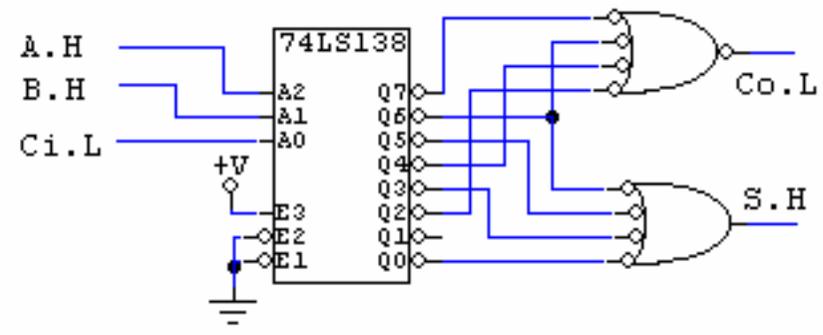
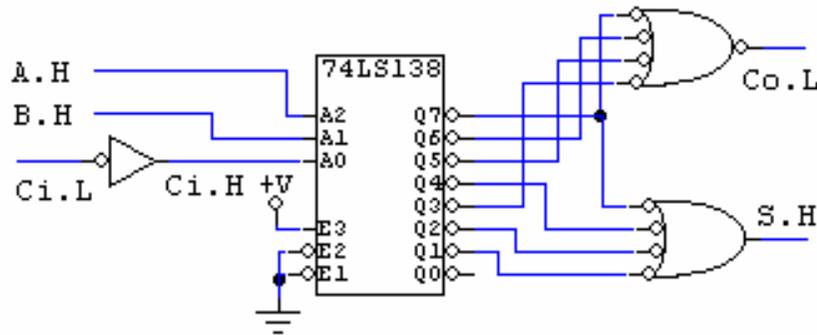
Implementación de funciones lógicas con decodificadores

- Si alguna de las entradas de la función es redundante la función se puede hacer con un decodificador más pequeño. Por ejemplo $F(A,B,C) = \sum(0, 3, 4, 7) \Leftrightarrow G(B,C) = \sum(0, 3)$, se puede implementar con un 2 a 4 DEC en lugar de un 3 a 8 DEC.
- Si la implementación final se realiza con un árbol de decodificadores, si en algún decodificador no se usa ninguna salida se puede eliminar, y si se usan todas las salidas se puede eliminar el decodificador y utilizar su habilitador. Incluso se podrían ordenar las entradas en el árbol para eliminar el mayor número de elementos.



Implementación de funciones lógicas con decodificadores

- Normalmente las entradas de dirección de un decodificador son de polaridad positiva. Si alguna de las entradas de la función es en polaridad negativa hay problemas al hacer la conexión; una posibilidad de solucionarlo es usando inversores. Sumador completo: suponer A.H, B.H, Ci.L, S.H, Co.L



- Se pueden eliminar los inversores realizando la complementación sobre las entradas en la tabla de verdad.

A	B	Ci	Co	S
0	0	1	0	0
0	0	0	0	1
0	1	1	0	1
0	1	0	1	0
1	0	1	0	1
1	0	0	1	0
1	1	1	1	0
1	1	0	1	1

$$S(A, B, Ci) = \sum(0, 3, 5, 6)$$

$$Co(A, B, Ci) = \sum(2, 4, 6, 7)$$

Implementación de funciones lógicas con multiplexores

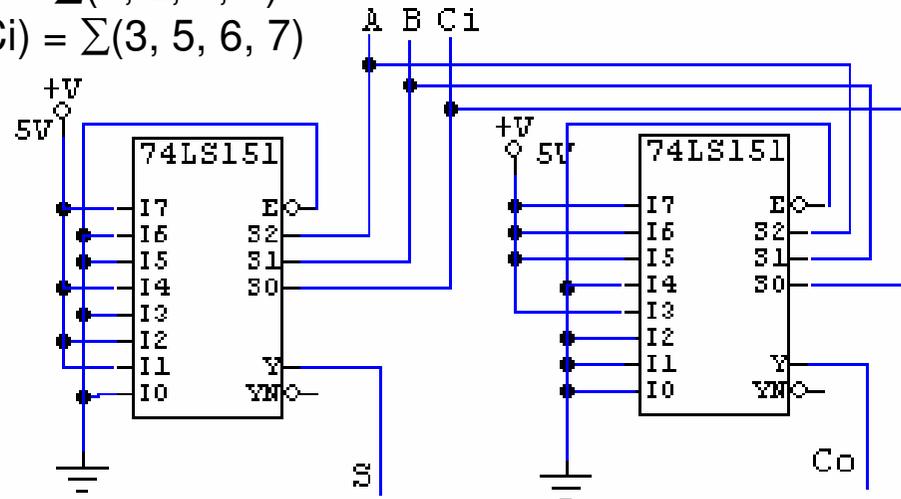
- Los multiplexores permiten implementar directamente funciones lógicas desde sus formas canónicas. Aplicando las entradas a las entradas de selección del multiplexor, cada una de las entradas de datos corresponde a cada uno de los valores lógicos (0 ó 1) de la tabla de verdad: I0 es la fila 0, I1 la fila 1, etc. En principio hay que usar multiplexores de 2^N entradas, siendo N el número de entradas de la función lógica. Este tipo de implementación es de tipo 0, ya que no se conecta ninguna entrada a las entradas de datos del MUX.

Sumador completo

	A	B	Ci	Co	S
I0	0	0	0	0	0
I1	0	0	1	0	1
I2	0	1	0	0	1
I3	0	1	1	1	0
I4	1	0	0	0	1
I5	1	0	1	1	0
I6	1	1	0	1	0
I7	1	1	1	1	1

$$S(A, B, Ci) = \sum(1, 2, 4, 7)$$

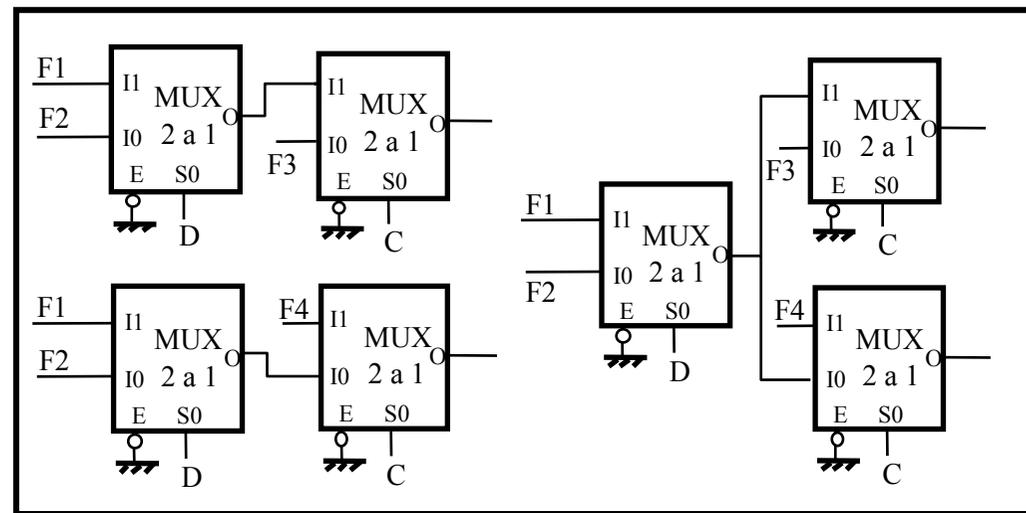
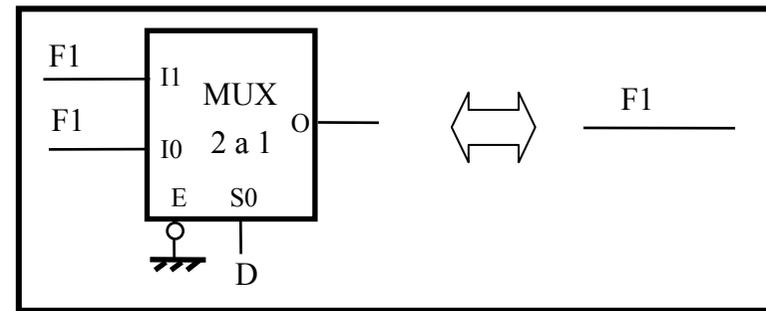
$$Co(A, B, Ci) = \sum(3, 5, 6, 7)$$



Implementación de funciones lógicas con multiplexores

- Si alguna de las entradas de la función es redundante la función se puede hacer con un multiplexor más pequeño. Por ejemplo $F(A,B,C) = \sum(0, 3, 4, 7) \Leftrightarrow G(B,C) = \sum(0, 3)$, se puede implementar con un 4-input MUX en lugar de un 8-input MUX.

- Si la implementación final se realiza con un árbol de multiplexores, si en algún multiplexor todas las entradas de datos tienen el mismo valor se puede eliminar y sustituir por el valor. Si dos multiplexores tienen las mismas entradas de datos y de selección se puede eliminar uno y usar fanout desde el otro. Incluso se podrían ordenar las entradas de selección en el árbol para eliminar el mayor número de MUXs.

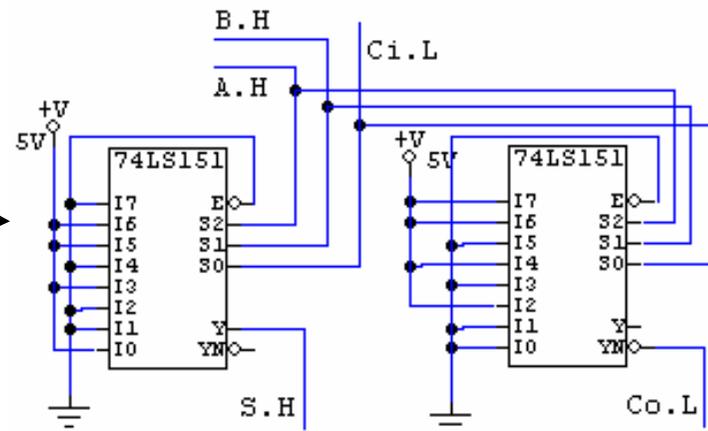
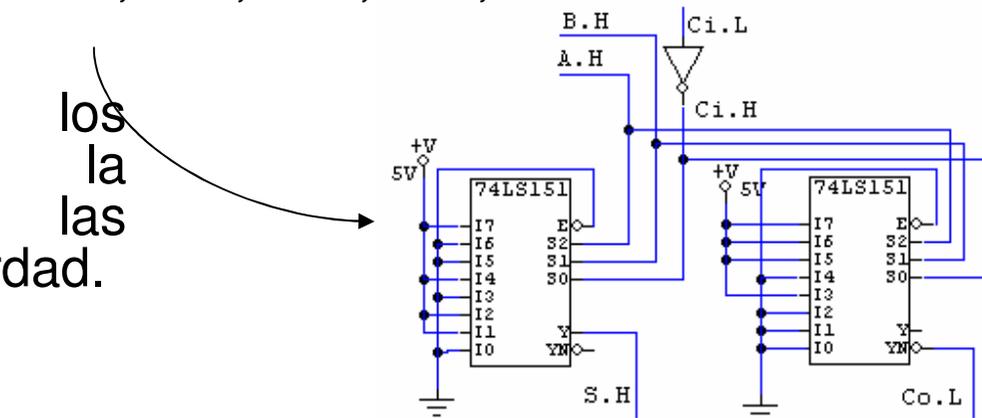


Implementación de funciones lógicas con multiplexores

- Normalmente las entradas de selección y de datos de un multiplexor son de polaridad positiva. Si alguna de las entradas de la función es en polaridad negativa hay problemas al hacer la conexión; una posibilidad de solucionarlo es usando inversores. Sumador completo: suponer A.H, B.H, Ci.L, S.H, Co.L

- Se pueden eliminar los inversores realizando la complementación sobre las entradas en la tabla de verdad.

	A	B	Ci	Co	S
I1	0	0	1	0	0
I0	0	0	0	0	1
I3	0	1	1	0	1
I2	0	1	0	1	0
I5	1	0	1	0	1
I4	1	0	0	1	0
I7	1	1	1	1	0
I6	1	1	0	1	1



Implementación de funciones lógicas con multiplexores

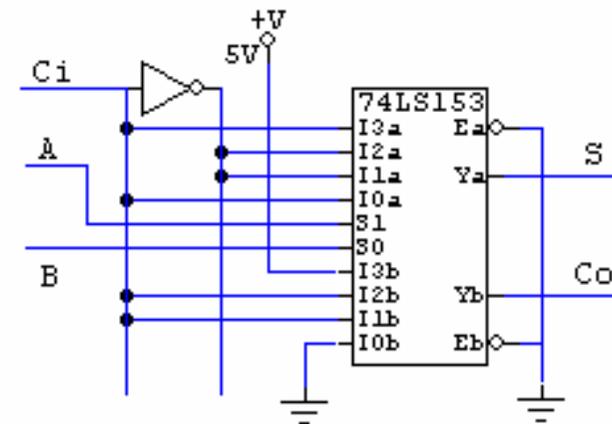
- Se puede reducir el tamaño de los multiplexores conectando alguna de las entradas a las entradas de datos del multiplexor (una entrada: tipo 1, 2 entradas tipo 2, etc). Si se pasa una entrada x a las entradas del multiplexor se puede conectar 0 , 1 , x ó \bar{x} . A cambio de un inversor el tamaño de los multiplexores se reduce a la mitad, ya que se necesita una entrada menos de selección.

En las implementaciones de tipo 2, 3, etc, se deben conectar a las entradas de datos circuitos más complejos, con lo que la implementación no es tan rentable.

A	B	Ci	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



	A	B	Co	S
10	0	0	0	Ci
11	0	1	Ci	$\bar{C}i$
12	1	0	Ci	$\bar{C}i$
13	1	1	1	Ci



Implementación de funciones lógicas con multiplexores

- Un método completo para implementar funciones lógicas con multiplexores sobre el Mapa de Karnaugh se basa en los siguientes puntos:
 1. Plantear el Mapa de Karnaugh de la función.
 2. Comprobar que ninguna de las entradas es redundante sobre el mapa. Si alguna entrada es redundante, se rehace la función lógica eliminando las entradas redundantes.
 3. Se introduce una variable de entrada x en el Mapa de Karnaugh de forma que en las casillas puedan aparecer valores 0 , 1 , x ó \bar{x} . La selección de la entrada se puede elegir aleatoriamente, normalmente la menos significativa.
 4. Se tiene en cuenta la polaridad de las señales de entrada: se complementan los valores lógicos asociados a las entradas de polaridad negativa en los márgenes del mapa de Karnaugh, o los literales de la entrada x si es de polaridad negativa.
 5. Se genera el circuito con multiplexores en función del número de entradas de selección necesarias y del tipo de multiplexores disponibles. Las entradas de selección de cada multiplexor se pueden escoger aleatoriamente, pero normalmente se escogen de más a menos significativas. Las entradas de datos se conectan a Gnd (0), Vcc(1), x directa (x) o x con inversor (\bar{x}).

Implementación de funciones lógicas con multiplexores

$F(A, B, C, D) = \sum(1, 5, 7, 9, 12, 13) + \sum\emptyset(4, 14, 15)$,
para F.H, A.H, B.L, C.H, D.L.

1. Plantear el Mapa de Karnaugh de la función.

2. Si al meter una variable en el Mapa todos los 1-cubos pueden agrupar dos 1s ó dos 0s. La variable es redundante. Si hay un 1 y un 0 en al menos un 1-cubo la variable no es redundante. A es redundante; B, C y D no.

		CD			
	AB	00	01	11	10
00		0	1	0	0
01		∅	1	1	0
11		1	1	∅	∅
10		0	1	0	0

		CD			
	B	00	01	11	10
0		0	1	0	0
1		1	1	1	0

		CD			
	AB	00	01	11	10
00		0	1	0	0
01		∅	1	1	0
11		1	1	∅	∅
10		0	1	0	0

Para D

		CD			
	AB	00	01	11	10
00		0	1	0	0
01		∅	1	1	0
11		1	1	∅	∅
10		0	1	0	0

Para C

		CD			
	AB	00	01	11	10
00		0	1	0	0
01		∅	1	1	0
11		1	1	∅	∅
10		0	1	0	0

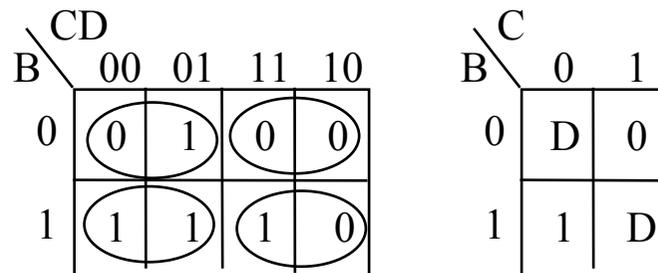
Para B

		CD			
	AB	00	01	11	10
00		0	1	0	0
01		∅	1	1	0
11		1	1	∅	∅
10		0	1	0	0

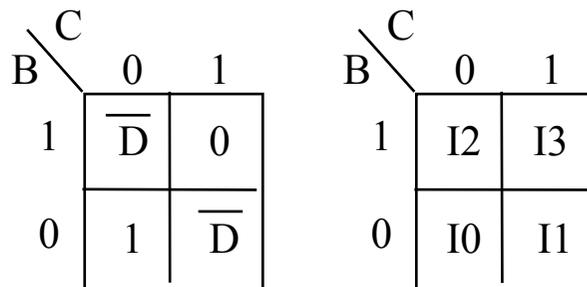
Para A

Implementación de funciones lógicas con multiplexores

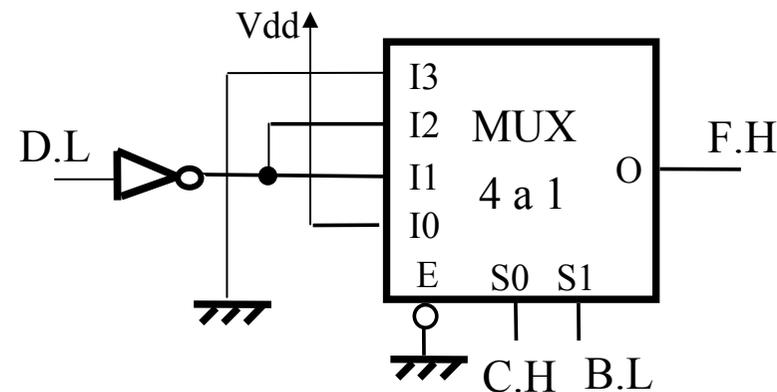
3. Meter D dentro del Mapa. Se generan los 1-cubos de D y se compara el valor de la función con el valor de D en los márgenes del Mapa: dos 0s se mete 0, dos 1 se mete 1, (0, 1) si coincide con los márgenes se mete D, si no coincide \bar{D} .



4. Se ajusta la polaridad se complementan los márgenes para B.L y los literales de D.L



5. Se construye el circuito



Codificadores con prioridad

- Un codificador realiza la operación inversa a la decodificación: convierte la información de entrada I_i de N bits en M líneas de salida A_j , que realizan una codificación binaria del índice i de una de las entradas. En codificadores binarios para N bits de entrada el número de salidas es de forma que $N = 2^M$ (4 a 2, 8 a 3, etc) aunque también puede haber codificadores BCD (10 a 4) etc.
- Si las entradas toman los valores típicos de la salida de un decodificador (sólo se permite una entrada a 1 cada vez) el circuito puede implementarse sólo con puertas OR, por lo que no se necesita un circuito específico.
- Si se permiten que varias entradas puedan tomar valor lógico 1 se debe establecer una prioridad para saber que índice toma la salida. Normalmente se establece alta prioridad (HPRI, el índice más alto ó baja prioridad (LPRI, el índice más bajo).

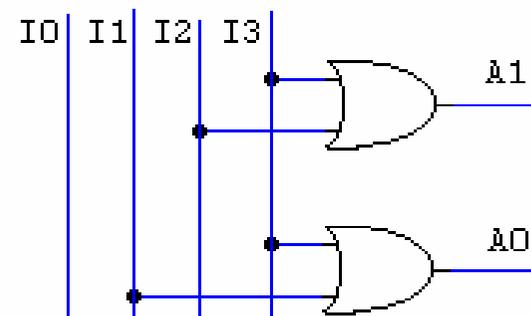
Codificadores con prioridad

- Por ejemplo, un codificador 4 a 2 en el que sólo se permite un 1 cada vez corresponde a la siguiente tabla de verdad.

La función lógica de cada salida se obtiene mediante el OR lógico de las entradas a 1 en las filas que producen 1 en dicha salida.

Al poder realizarse el circuito con puertas OR no se necesita un circuito específico para implementar este tipo de codificador.

I0	I1	I2	I3	A1	A0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



$$A1 = I2 + I3$$

$$A0 = I1 + I3$$

Codificadores con prioridad

- Un codificador 4 a 2 con prioridad alta genera en la salida el índice más alto puesto a 1. En la tabla se añade una salida de control Z que indica si alguna de las entradas está a 1.

La función lógica de cada salida se puede obtener haciendo para cada salida el OR de las filas que producen 1 en dicha salida. Las expresiones se simplifican mediante álgebra de conmutación usando la propiedad distributiva y el teorema de simplificación:

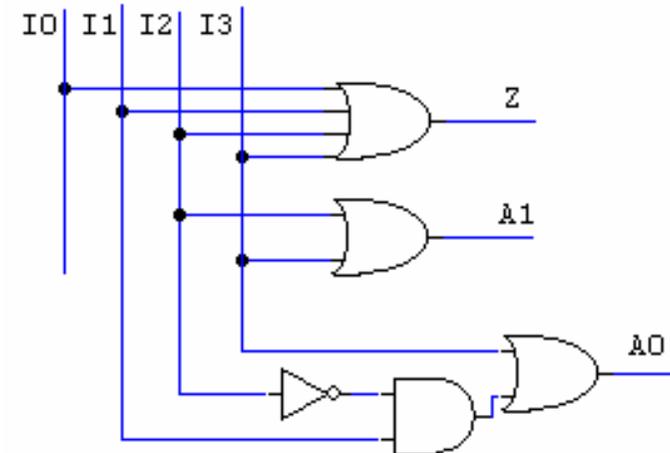
$$(x + \bar{x}y = x + y).$$

	I0	I1	I2	I3	A1	A0	Z
I3	X	X	X	1	1	1	1
\bar{I}_3 I2	X	X	1	0	1	0	1
\bar{I}_3 \bar{I}_2 I1	X	1	0	0	0	1	1
\bar{I}_3 \bar{I}_2 \bar{I}_1 I0	1	0	0	0	0	0	1
	0	0	0	0	0	0	0

$$A1 = I3 + \bar{I}_3 I2 = I3 + I2$$

$$A0 = I3 + \bar{I}_3 \bar{I}_2 I1 = I3 + \bar{I}_2 I1$$

$$Z = I3 + I2 + I1 + I0$$



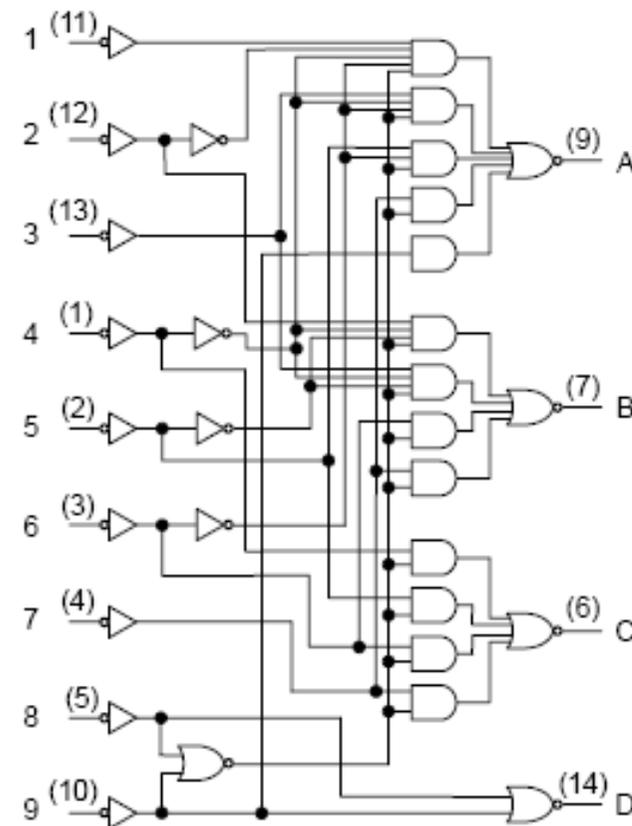
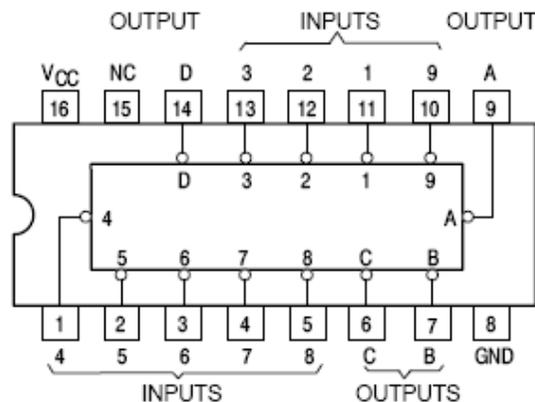
Codificadores con prioridad

Circuitos comerciales: 74'147, codificador con prioridad alta de 10 a 4. Las salidas y las entradas están en polaridad negativa. No se usa la entrada 0, se considera que está a 1 cuando las demás entradas están a 0

SN74LS147
FUNCTION TABLE

INPUTS									OUTPUTS			
1	2	3	4	5	6	7	8	9	D	C	B	A
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	L	H	H	H	H	L	L	L
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Logic Level, L = LOW Logic Level, X = Irrelevant

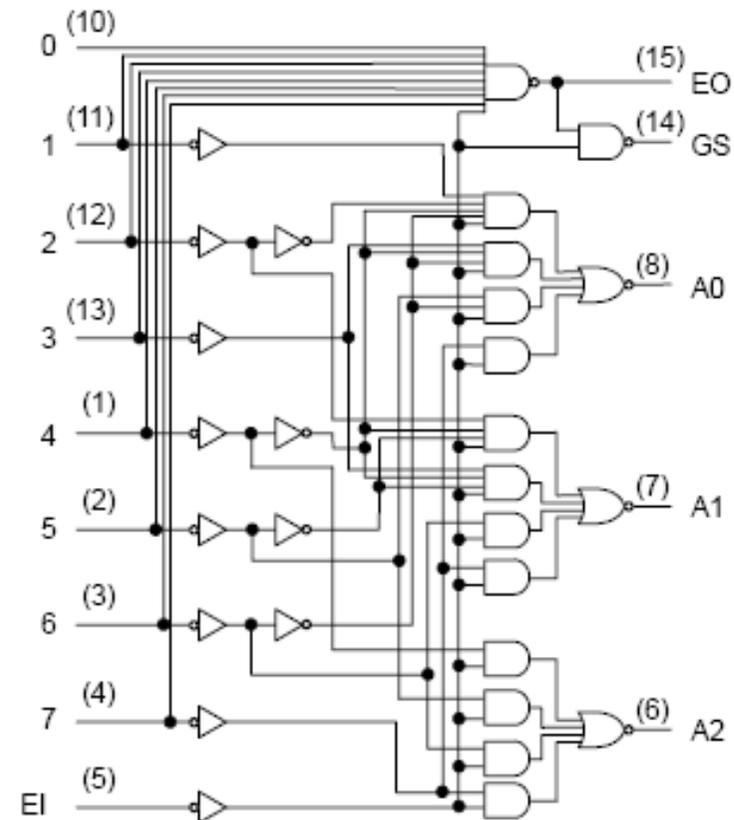
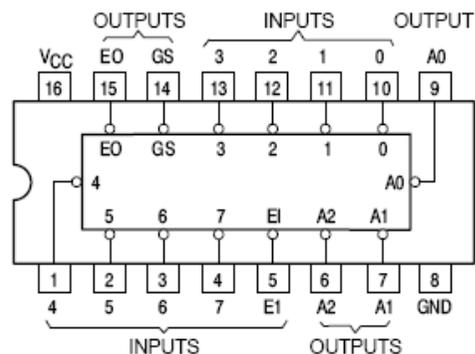


Codificadores con prioridad

Circuitos comerciales: 74'148, codificador con prioridad alta de 8 a 3. Las salidas y las entradas están en polaridad negativa. Dispone de un habilitador de entrada EI, y dos salidas de control GS (a 1 si hay alguna entrada a 1) y E0 (a 1 si ninguna entrada está a 1) que permiten construir codificadores de más bits.

SN74LS148
FUNCTION TABLE

INPUTS								OUTPUTS					
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H



Codificadores con prioridad

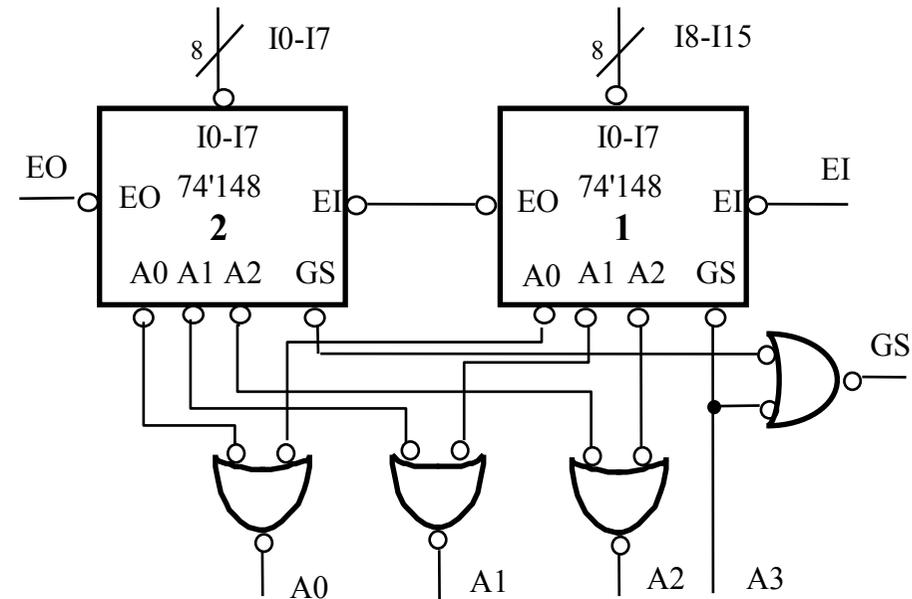
Desarrollo de codificadores de N a 2^N bits en base a codificadores de M a 2^M bits ($N > M$).

Desarrollo en cascada: se utilizan entradas de habilitación como en el 74'148. Por ejemplo: 16 a 4 en base a dos 8 a 3.

Si hay al menos un 1 en I8-I15, GS1 es 1 (A3 es 1), y EO1 es 0, con lo que COD2 queda deshabilitado, luego sus salidas A y GS2 son 0. Las salidas A2, A1, A0 son las del COD1.

Si hay 0s en I8-I15 y al menos un 1 en I0-I7, en el COD1 GS1 y las salidas A son 0 (A3 es 0), y EO1 es 1, con lo que COD2 queda habilitado. Las salidas A2, A1, A0 son las del COD2.

Si no hay ningún 1, en COD1 y COD2, GS y las salidas A son 0. A3-A0 son 0, GS es 0 y EO es 1.



Se puede realizar una cadena con más CODs, bajo una similar estructura, con puertas OR de más entradas. Los bits altos de salida A3, A4, etc, se obtienen de las salidas GS mediante un codificador sin prioridad.

Codificadores con prioridad

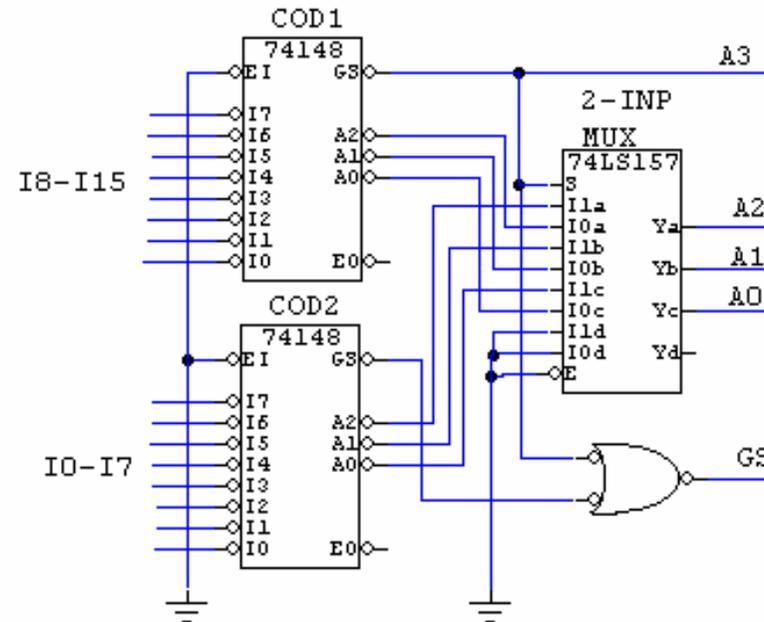
Desarrollo de codificadores de N a 2^N bits en base a codificadores de M a 2^M bits ($N > M$).

Desarrollo en paralelo. Por ejemplo: 16 a 4 en base a dos 8 a 3.

Si hay al menos un 1 en I8-I15, GS1 es 1 (A3 es 1). La entrada de selección del MUX está a 1 luego A2-A0 corresponden a las del COD1.

Si hay 0s en I8-I15 y al menos un 1 en I0-I7, en el COD1 GS1 es 0 (A3 es 0). La entrada de selección del MUX está a 0 luego A2-A0 corresponden a las del COD2.

Si no hay ningún 1, en COD1 y COD2, GS y las salidas A son 0. A3 es 0 por GS1, y A2-A0 son 0, ya que el MUX toma las de COD2 que son 0.



Se puede realizar una cadena con más CODs, bajo una similar estructura, con MUXs de más entradas. Los bits altos de salida A3, A4, etc, (entradas de selección de los MUXs) se obtienen de las salidas GS mediante un codificador con prioridad.

Codificadores con prioridad

Modelo VHDL de un 4 a 2 HPRI COD

```
library ieee;  
use.ieee.std_logic_1164.all;  
  
entity hpri4to2 is  
port (I: in std_logic_vector(3 downto 0);    -- Entradas de datos  
       A: out std_logic_vector(1 downto 0); -- Salidas  
       Z: out std_logic); -- Control de que alguna entrada está a 1  
end hpri4to2;  
  
architecture comportamiento of hpri4to2 is  
begin  
A <= "11" when I(3) = '1' else  
    "10" when I(2) = '1' else  
    "01" when I(1) = '1' else  
    "00";  
Z <= '0' when I = "0000" else '1';  
end comportamiento;
```

Sumadores

- Los circuitos digitales sumadores realizan la suma aritmética de dos números enteros positivos, normalmente descritos en notación posicional binaria, aunque pueden desarrollarse sumadores para otros formatos de descripción numérica.

Los sumadores son un elemento crítico en el desarrollo de circuitos aritméticos por lo que se han desarrollado numerosas estructuras que buscan la mejora de las prestaciones del circuito, balanceando entre su tamaño y su velocidad.

Para operandos A y B de un bit ya se han desarrollado en otros temas el sumador completo (“full-adder”) con acarreo de entrada (Ci) y el semi-sumador (“half-adder) sin acarreo de entrada. Los bits de salida serán la salida de suma S y el acarreo de salida (Co).

A	B	Co	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$

$$Co = A B$$

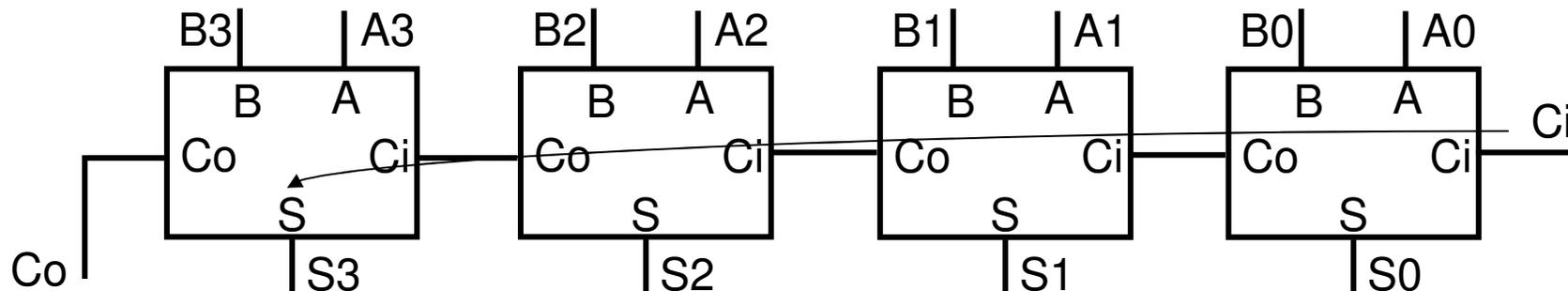
A	B	Ci	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus Ci$$

$$Co = A B + A Ci + B Ci$$

Sumadores

- Para sumadores de N bits un método natural de realizar la suma es situar sumadores completos en modo “ripple” o en serie, de forma que desde el bit menos significativo hacia el más significativo el acarreo de entrada del bit j esté conectado al acarreo de salida del bit $j-1$. El primer bit se puede construir con un semisumador mientras que los demás bits requieren un sumador completo, en este caso se tiene un semisumador de N bits. Si en el primer bit se utiliza un sumador completo, el circuito dispone además de acarreo de entrada C_i y se tiene un sumador completo de N bits. Tener acarreo de entrada permite un mejor funcionalidad del circuito, como por ejemplo poner en serie dos (o más) sumadores de N bits para formar un sumador de $2N$ bits. En cualquier caso se tienen $N+1$ bits de salida: N de suma S y un acarreo de salida C_o . El tiempo de propagación de este sumador es proporcional al número N de sumadores en serie.



Sumadores

- El sumador “ripple” es un sumador lento. Para construir sumadores rápidos hay que intentar paralelizar el cálculo de los acarreos. El método “carry look-ahead” genera los acarreos en paralelo obteniendo su expresión lógica de forma recursiva:
- Para cada bit j se obtiene de A_j y B_j en paralelo el “Carry-Propagate” P_j que indica las condiciones bajo las cuales el acarreo se propaga de la entrada a la salida, y el “Carry-Generate” G_j que indica las condiciones bajo las cuales se genera acarreo de salida independientemente del acarreo de entrada.

T1

$\begin{aligned} P_j &= A_j \oplus B_j \\ G_j &= A_j B_j \end{aligned}$

 hay un 1 en A_j ó en B_j , se propaga el acarreo
 hay dos 1s entre A_j y B_j , se genera acarreo.

$C_0 = G + P C_i$, hay acarreo si se genera o si se propaga con C_i a 1.

- Para 4 bits se generan los acarreos en paralelo:

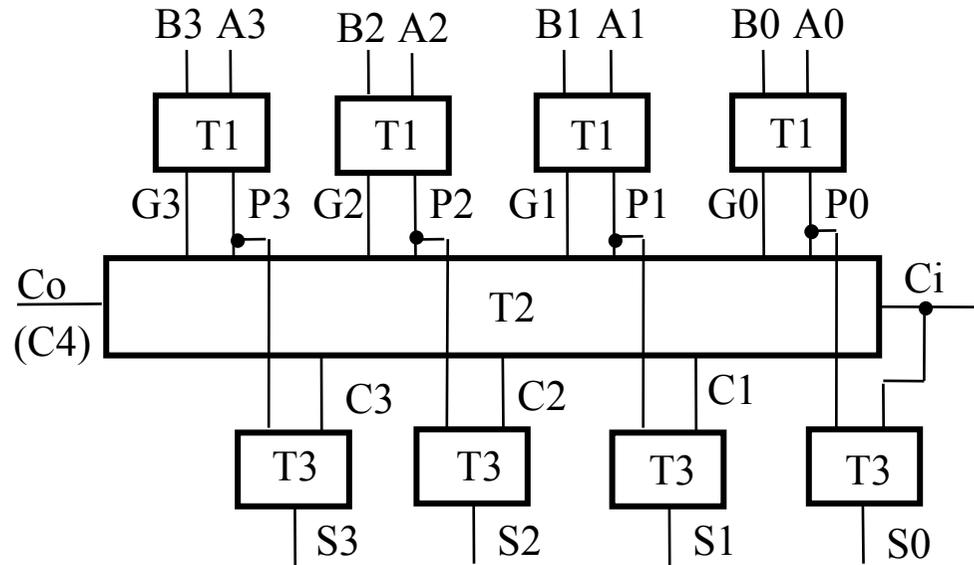
T2

$\begin{aligned} C_1 &= G_0 + P_0 C_i \\ C_2 &= G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_i) = G_1 + P_1 G_0 + P_1 P_0 C_i \\ C_3 &= G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_i \\ C_0 = C_4 &= G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + \\ &\quad + P_3 P_2 P_1 P_0 C_i \end{aligned}$
--

Sumadores

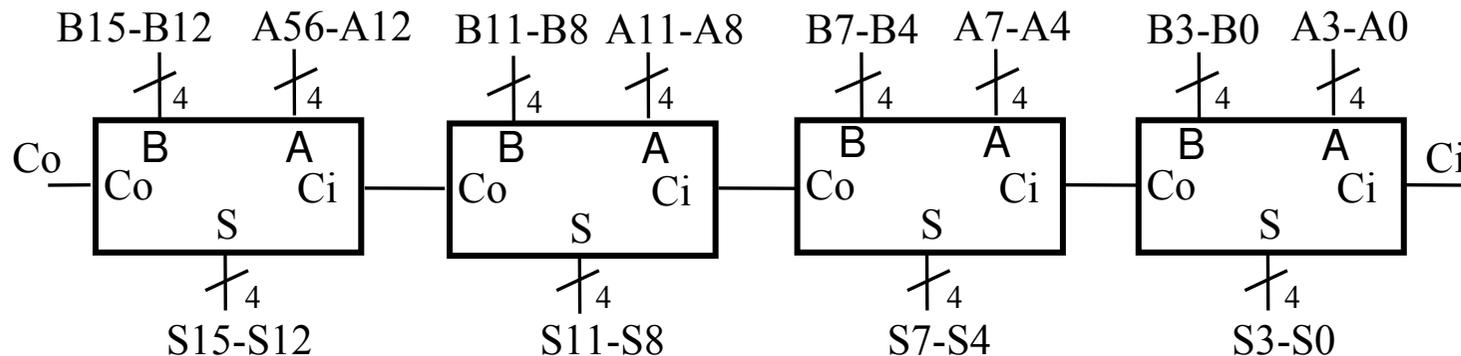
- Una vez generados los acarreos se generan las salidas también en paralelo:

$$\begin{array}{l}
 \boxed{S_0 = P_0 \oplus C_i} \\
 \boxed{S_j = P_j \oplus C_j}
 \end{array}
 \begin{array}{l}
 \text{para } j = 0 \\
 \text{j entre 1 y 3.}
 \end{array}$$



- El circuito se construye en paralelo a a partir de las ecuaciones de T1, T2 y T3.

- Las ecuaciones que definen los acarreos son cada vez más grandes, luego no es rentable continuar aumentando el circuito. Una solución es poner sumadores “carry look-ahead” de 4 bits en serie.



Sumadores

- Otra solución es llevar la estructura de “carry look-ahead” a más niveles, la expresión de C_o también se puede poner de la forma:

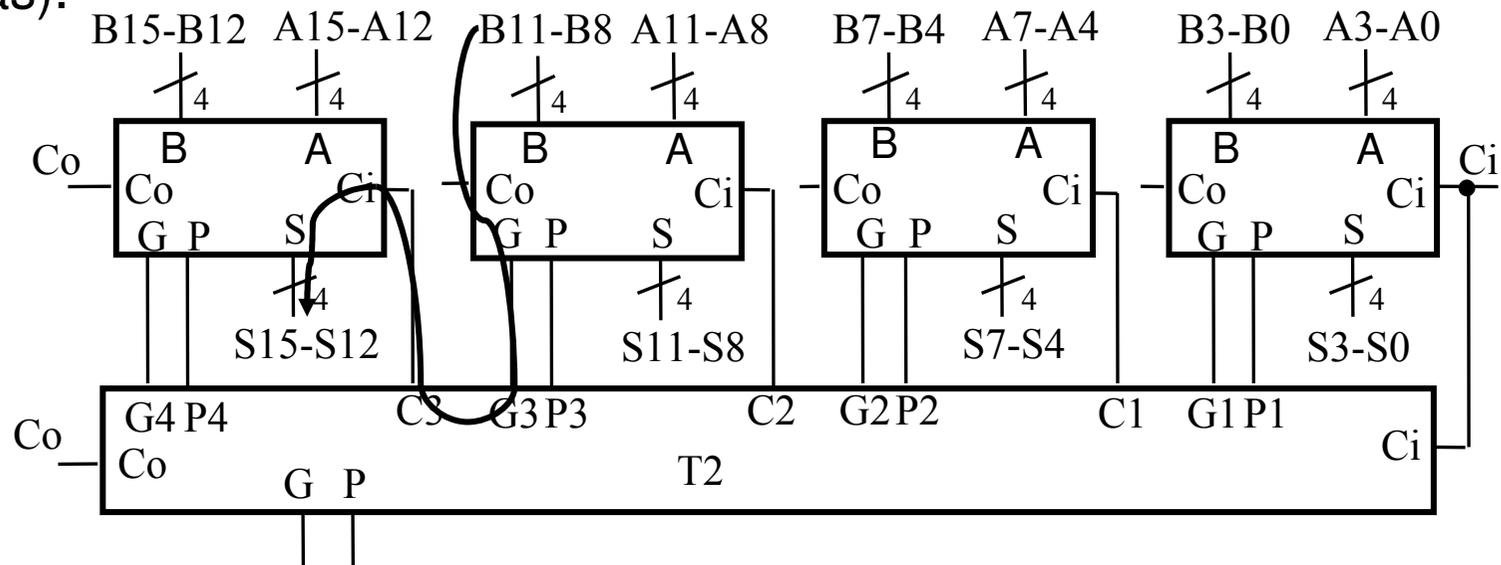
$C_o = G + P C_i$, siendo

$C_o = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_4P_3P_2P_1C_i$, luego

$G = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$

$P = P_4P_3P_2P_1$

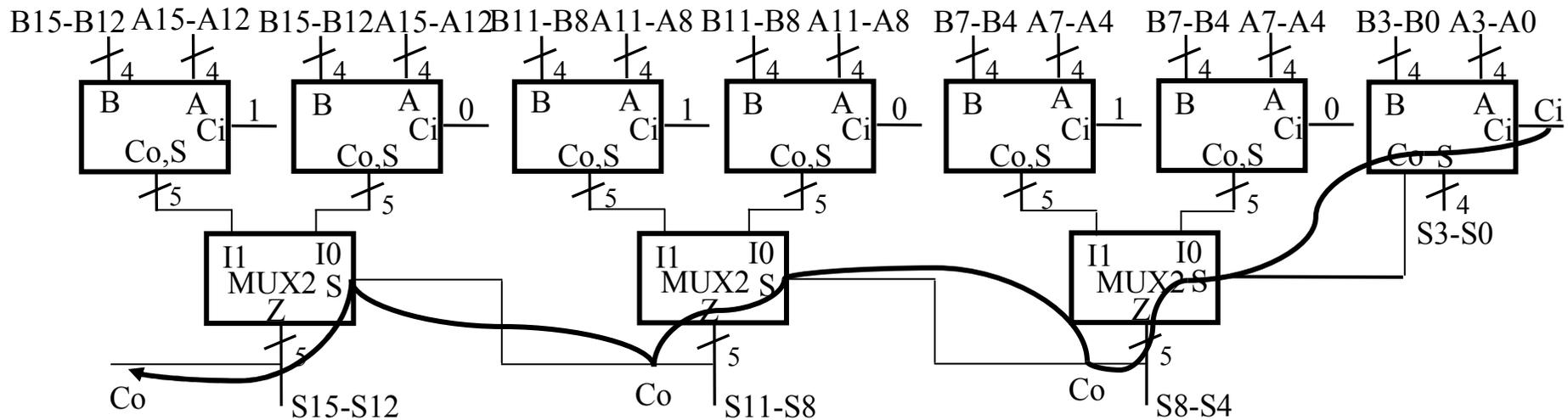
donde G y P son el “carry-generate” y el “carry-propagate” de una suma de 4 bits. Si se añaden las ecuaciones de G y P a T_2 se pueden realizar sumadores de 16 bits con esta estructura, que se puede llevar a más bits utilizando más niveles de T_2 (64 bits, 4 grupos de 16 con un nivel más).



Sumadores

- Otra estructura de suma rápida es el “carry-select” basada en sumadores y multiplexores. Para una suma de M bits en M/N etapas de N bits, en cada etapa de suma, menos la primera, se calcula la suma en paralelo con acarreo de entrada a 0 ó a 1, y luego, cuando se obtiene el acarreo real, se seleccionan con los multiplexores las salidas correctas.

El tiempo de propagación de este sumador depende del tiempo de propagación de la primera etapa, más el tiempo de propagación de los $(M/N-1)$ multiplexores para propagación del acarreo. A cambio el circuito es bastante más grande que la estructura “ripple”.



Sumadores

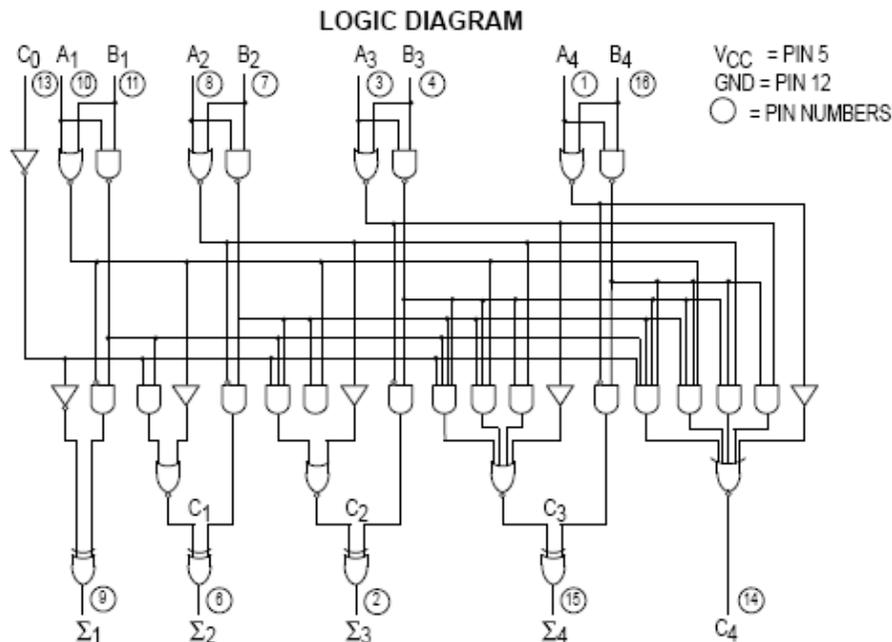
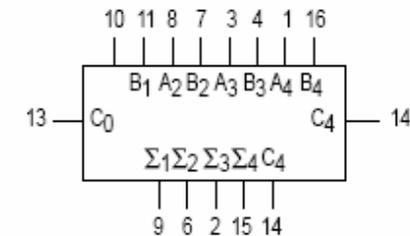
Circuitos comerciales: 74'83. Sumador de 4 bits, con estructura interna de carry look-ahead. El circuito opera como sumador suponiendo las entradas y salidas en polaridad positiva o en polaridad negativa.

$$C_0 + (A_1+B_1)+2(A_2+B_2)+4(A_3+B_3)+8(A_4+B_4) = \Sigma_1+2\Sigma_2+4\Sigma_3+8\Sigma_4+16C_4$$

Where: (+) = plus

	C ₀	A ₁	A ₂	A ₃	A ₄	B ₁	B ₂	B ₃	B ₄	Σ ₁	Σ ₂	Σ ₃	Σ ₄	C ₄
Logic Levels	L	L	H	L	H	H	L	L	H	H	H	L	L	H
Active HIGH	0	0	1	0	1	1	0	0	1	1	1	0	0	1
Active LOW	1	1	0	1	0	0	1	1	0	0	0	1	1	0

(10+9 = 19)
(carry+5+6 = 12)



Los modelos de descripción VHDL de sumadores se han mostrado en el tema II.

Sumadores

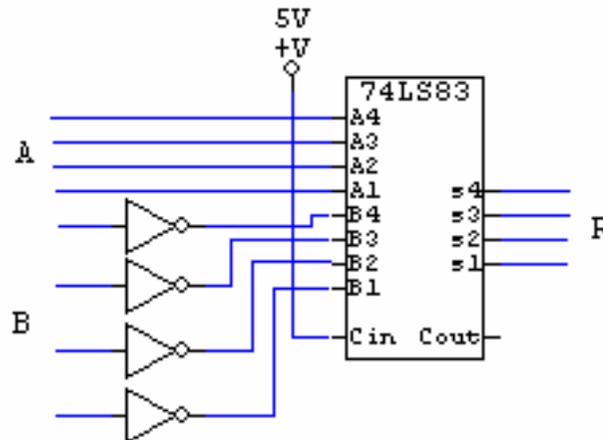
- A partir de un sumador pueden realizarse otras aplicaciones lógicas sencillas. Por ejemplo, un circuito restador o un circuito sumador/restador. Para realizar estas aplicaciones hay que hacer antes un planteamiento del problema, recurriendo a conocimientos adquiridos.

Un restador se realiza con un sumador suponiendo los operandos en complemento-2. De la aplicación del c-a-2 se sabe que $(-B)$ se realiza como $(B)_{c,2}$, que se puede formar complementando los bits de B y sumando 1 en el bit menos significativo. Luego:

$$R = A - B = A + \bar{B} + 1, \text{ donde } 1 \text{ se añade en el acarreo de entrada.}$$

En este caso, al estar A y B en 4 bits en c-a-2, el resultado de la resta R tiene tantos bits como las entradas, el acarreo de salida no se utiliza y existe la posibilidad de que se produzca "overflow".

Restador de 4 bits
usando el sumador
74LS83



Sumadores

- Un sumador/restador necesita una entrada de control C, que indique si se realiza la operación de suma o de resta. Para hacer la resta se requiere el c-a-2, luego los operandos X e Y, y la salida Z están en esta notación.

$$\text{Si } C = 0 \Rightarrow Z = X + Y + 0$$

$$\text{Si } C = 1 \Rightarrow Z = X - Y = X + \bar{Y} + 1$$

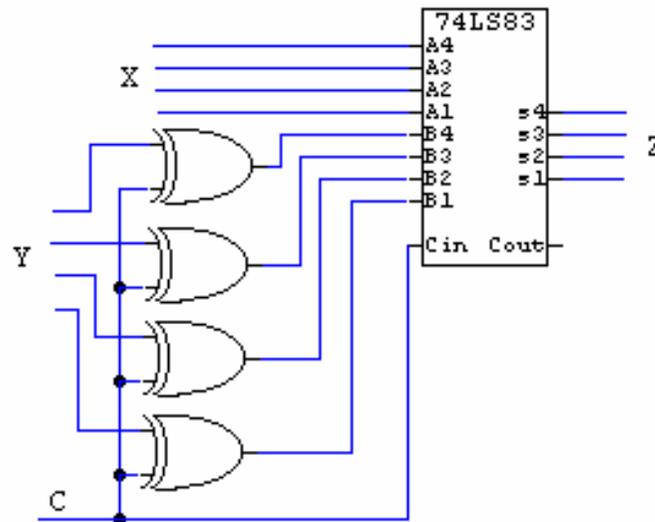
Para el operando A del sumador da igual el valor de C, $A \leftarrow X$

Para el operando B si $C = 0 \Rightarrow B \leftarrow Y$, si $C = 1 \Rightarrow B \leftarrow \bar{Y}$, luego

$$B = \bar{C} Y + C \bar{Y} = C \oplus Y$$

Para Cin, si $C = 0 \Rightarrow \text{Cin} \leftarrow 0$, si $C = 1 \Rightarrow \text{Cin} \leftarrow 1$, luego $\text{Cin} = C$

Sumador/Restador
de 4 bits usando el
sumador 74LS83



Comparadores

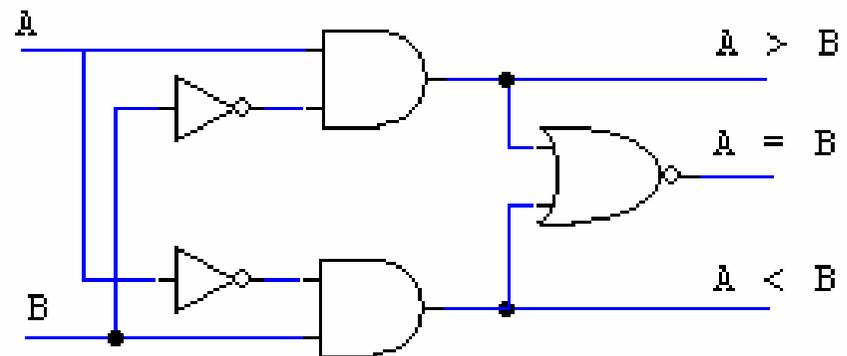
- Un circuito digital comparador realiza la comparación de dos palabras A y B de N bits tomadas como un número entero sin signo e indica si son iguales o si una es mayor que otra en tres salidas A = B, A > B y A < B. Bajo cualesquiera valores de A y B una y sólo una de las salidas estará a 1, permaneciendo las otras dos salidas a 0.
- Para unos operandos A y B de un bit se puede desarrollar un comparador de la siguiente tabla:

A	B	A = B	A > B	A < B
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

$$(A = B) = \overline{A \oplus B} = \overline{A \bar{B} + \bar{A} B}$$

$$(A > B) = A \bar{B}$$

$$(A < B) = \bar{A} B$$



Comparadores

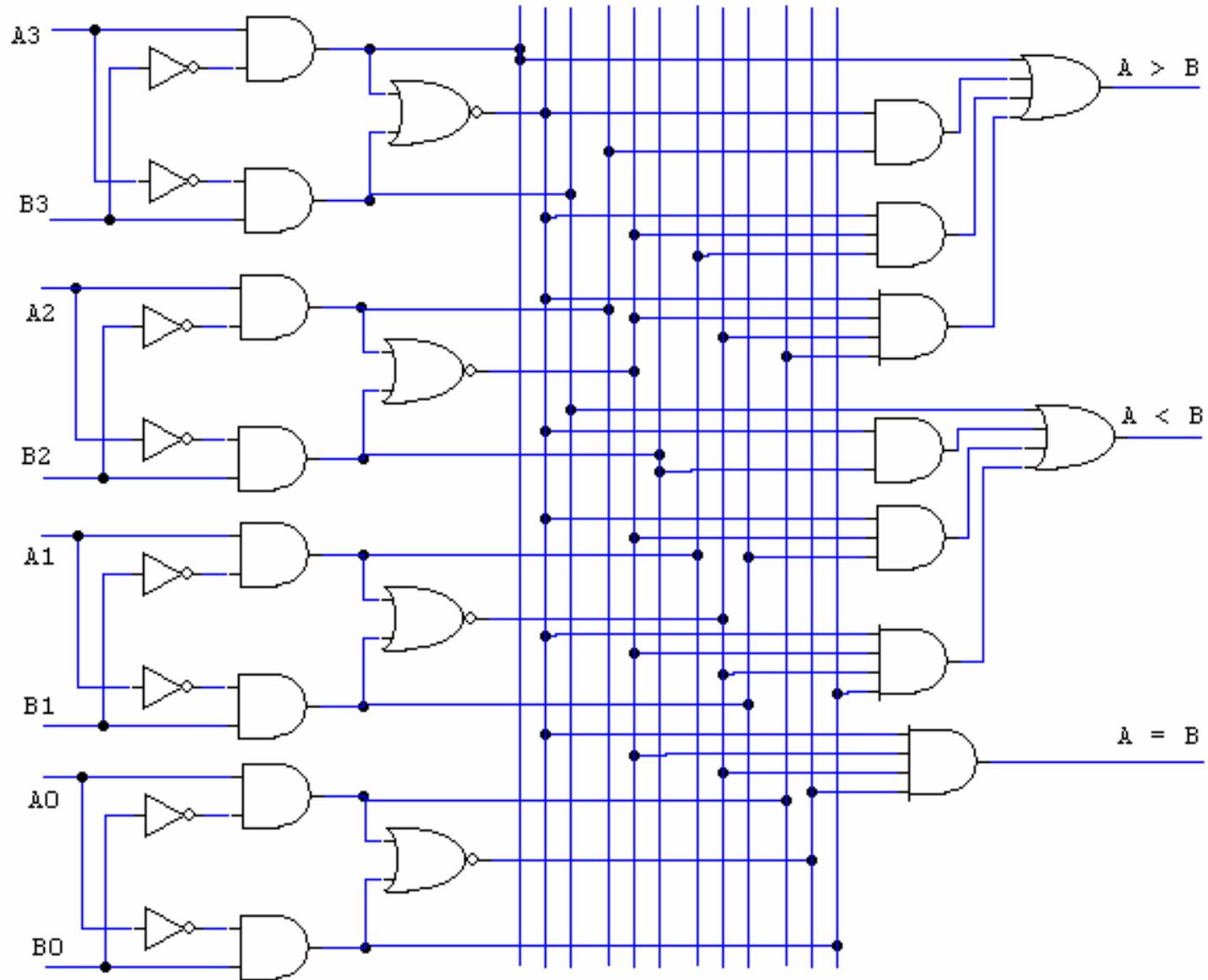
- Para N bits el comparador se puede desarrollar siguiendo estos razonamientos desde el bit más significativo hasta el bit menos significativo:
 - $A = B$, si $A_i = B_i$ para cada bit i de los operandos A y B.
 - $A > B$, si para un bit i $A_i > B_i$ siendo $A_j = B_j$ para todo $j > i$.
 - $A < B$, si para un bit i $A_i < B_i$ siendo $A_j = B_j$ para todo $j > i$.
- Para operandos A y B de 4 bits, conociendo las expresiones de cuando A_i es igual, mayor o menor que B_i , las expresiones lógicas correspondientes quedan así:

$$(A = B) = \overline{A_3 \oplus B_3} \overline{A_2 \oplus B_2} \overline{A_1 \oplus B_1} \overline{A_0 \oplus B_0}$$

$$(A > B) = A_3 \overline{B_3} + \overline{A_3 \oplus B_3} A_2 \overline{B_2} + \overline{A_3 \oplus B_3} \overline{A_2 \oplus B_2} A_1 \overline{B_1} + \\ + \overline{A_3 \oplus B_3} \overline{A_2 \oplus B_2} \overline{A_1 \oplus B_1} A_0 \overline{B_0}$$

$$(A < B) = \overline{A_3} B_3 + \overline{A_3 \oplus B_3} \overline{A_2} B_2 + \overline{A_3 \oplus B_3} \overline{A_2 \oplus B_2} \overline{A_1} B_1 + \\ + \overline{A_3 \oplus B_3} \overline{A_2 \oplus B_2} \overline{A_1 \oplus B_1} A_0 B_0$$

Comparadores

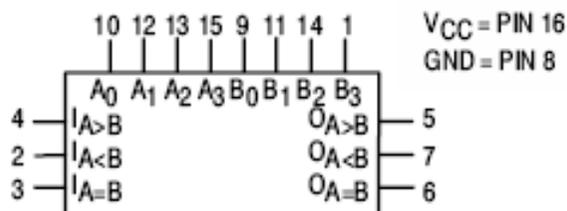


Comparadores

Circuitos comerciales: 74'85. Comparador de 4 bits.

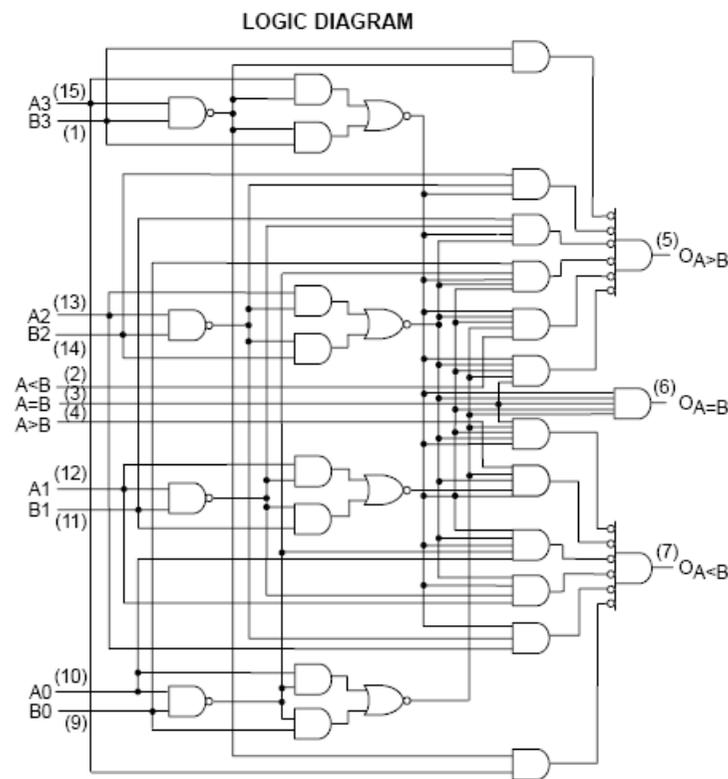
Además de las entradas y salidas habituales tiene 3 entradas de expansión $I_{A>B}$ $I_{A<B}$ $I_{A=B}$. Cuando A y B son iguales se estudian las entradas de expansión para generar el resultado final.

Durante la operación normal $I_{A=B}$ debe estar a H, e $I_{A>B}$ y $I_{A<B}$ a L, pero también se pueden utilizar para comparar un quinto bit.



TRUTH TABLE

COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A ₃ ,B ₃	A ₂ ,B ₂	A ₁ ,B ₁	A ₀ ,B ₀	I _{A>B}	I _{A<B}	I _{A=B}	O _{A>B}	O _{A<B}	O _{A=B}
A ₃ >B ₃	X	X	X	X	X	X	H	L	L
A ₃ <B ₃	X	X	X	X	X	X	L	H	L
A ₃ =B ₃	A ₂ >B ₂	X	X	X	X	X	H	L	L
A ₃ =B ₃	A ₂ <B ₂	X	X	X	X	X	L	H	L
A ₃ =B ₃	A ₂ =B ₂	A ₁ >B ₁	X	X	X	X	H	L	L
A ₃ =B ₃	A ₂ =B ₂	A ₁ <B ₁	X	X	X	X	L	H	L
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ >B ₀	X	X	X	H	L	L
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ <B ₀	X	X	X	L	H	L
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ =B ₀	H	L	L	H	L	L
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ =B ₀	L	H	L	L	H	L
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ =B ₀	X	X	H	L	L	H
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ =B ₀	H	H	L	L	L	L
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ =B ₀	L	L	L	H	H	L



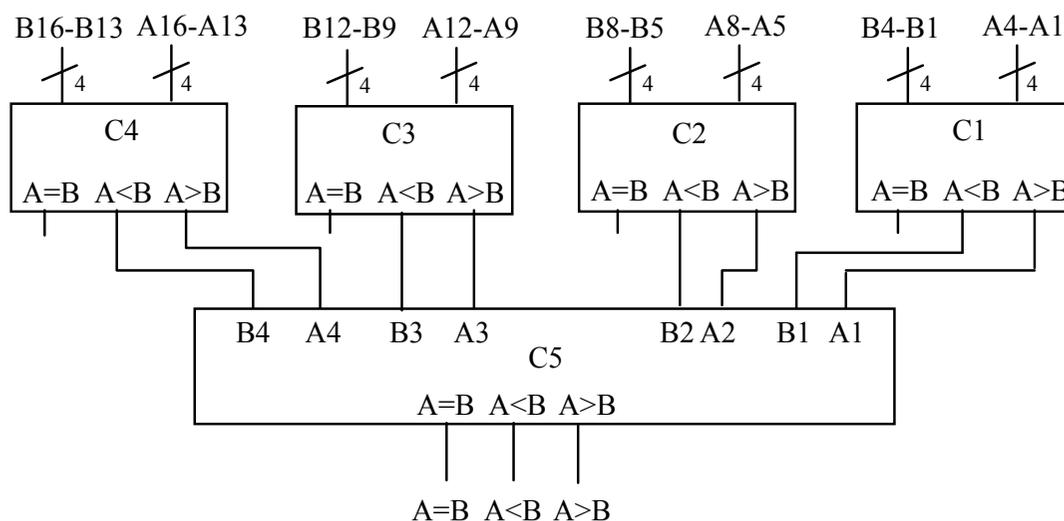
Comparadores

Desarrollo de comparadores de N bits en base a comparadores de M bits ($N > M$).

El desarrollo de un comparador de 4 bits ($N = 4$) se ha hecho en base a comparadores de 1 bit ($M = 1$). El mismo procedimiento puede utilizarse para comparar números de 16 bits en base a comparadores de 4 bits. Se usan las salidas $A > B$ y $A < B$, de forma que si en un grupo de 4 bits los operandos son iguales las dos salidas son 0, y si uno es mayor que otro estas salidas son (1, 0) para $A > B$ ó (0, 1) para $A < B$.

Si en C4 $A > B$ (ó $A < B$), en C5 A4 es 1 (0) y B4 es 0 (1) luego $A > B$ ($A < B$).

Si en C4 $A = B$, A4 y B4 son 0 y habría que estudiar el resultado de C3 bajo un razonamiento similar, luego el de C2 y luego el de C1.



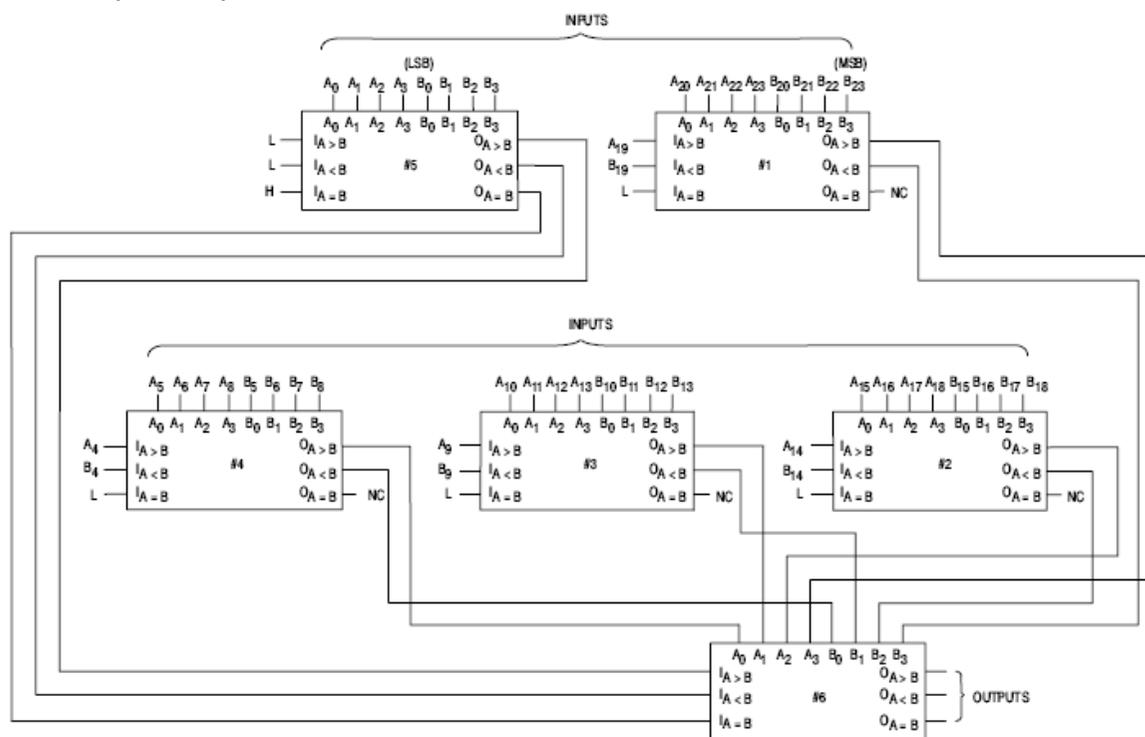
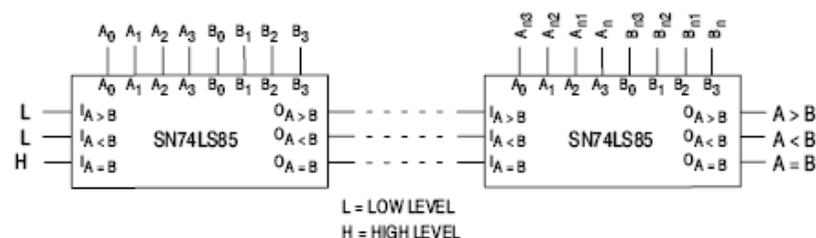
Comparadores

Desarrollo de comparadores de N bits en base a comparadores de M bits ($N > M$).

El circuito 74'85 (M = 4) permite realizar la comparación de números de más bits en dos estructuras:

En serie: utilizando las entradas de expansión: el tp es proporcional al número de 74'85 en serie (N/M).
24 bits en 6 niveles.

En paralelo. Se utilizan las entradas de expansión para usar un comparador más en cada nivel y un bit más por comparador. 24 bits en sólo 2 niveles.



MSB = MOST SIGNIFICANT BIT
LSB = LEAST SIGNIFICANT BIT
L = LOW LEVEL
H = HIGH LEVEL
NC = NO CONNECTION

Comparadores

Modelo VHDL de un comparador de N bits

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity compara is  
generic(N: integer := 4);  
port (A, B: in std_logic_vector(N downto 1);  
       AGB, ALB, AEB: out std_logic);  
end compara4;  
  
architecture comportamiento of compara4 is  
begin  
AGB <= '1' when ( A > B ) else '0'; -- A > B, salida AGB a 1, resto a 0  
ALB <= '1' when ( A < B ) else '0'; -- A < B, salida ALB a 1, resto a 0  
AEB <= '1' when ( A = B ) else '0'; -- A = B, salida AEB a 1, resto a 0  
end comportamiento;
```