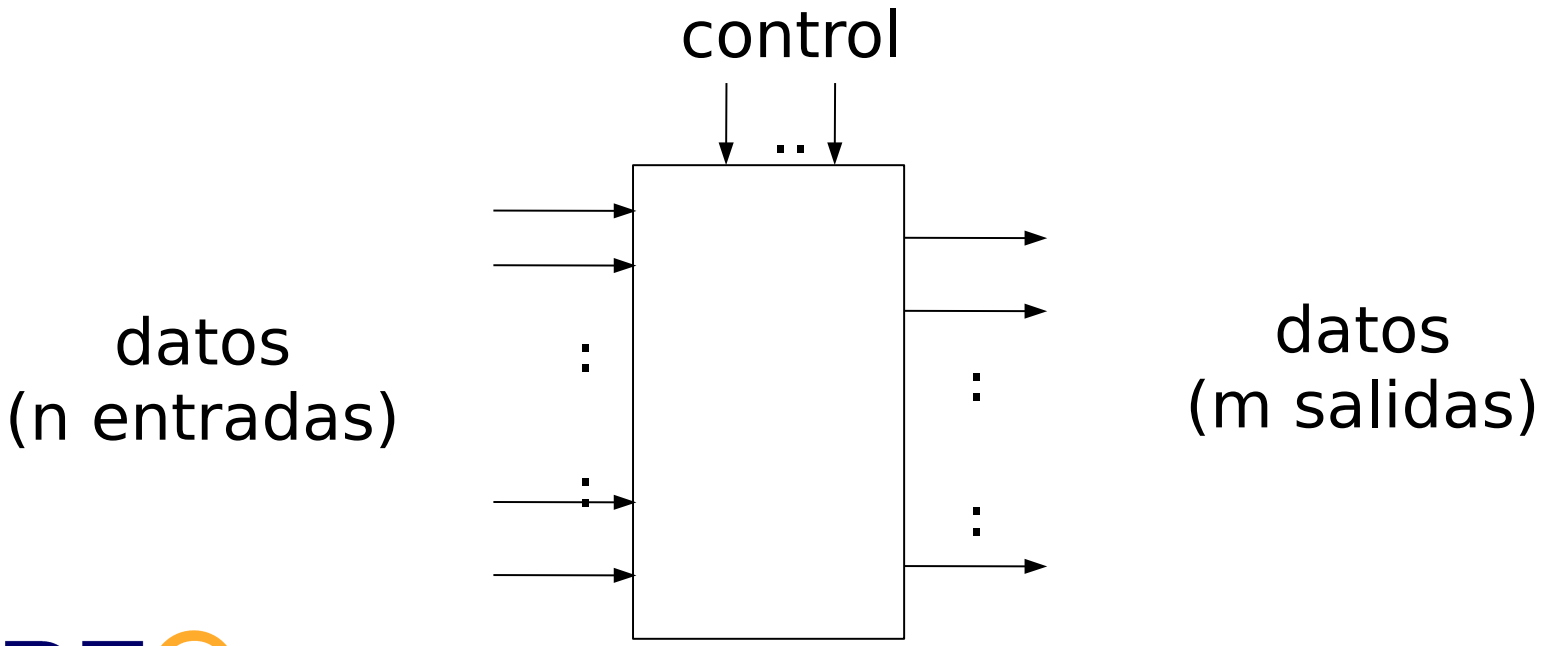
Tema 5
Subsistemas Combinacionales

Índice

- **Aspectos generales**
- **Subsistemas de propósito específico**:
  - Decodificadores y codificadores
  - Codificadores de prioridad
  - Convertidores de código
  - Comparadores
  - Demultiplexores
- **Subsistemas de propósito general**:
  - Multiplexores
  - Dispositivos programables (fuera del alcance de este tema)

1. Aspectos generales

- Facilitan el proceso de diseño, pues **realizan funciones complejas** habituales que requieren un gran número de puertas (100 a 1000; CI MSI/LSI) y son **no conmutativas**
- Tienen un número que puede ser grande de entradas (n) y salidas (m)
- Poseen dos tipos de terminales: datos y control

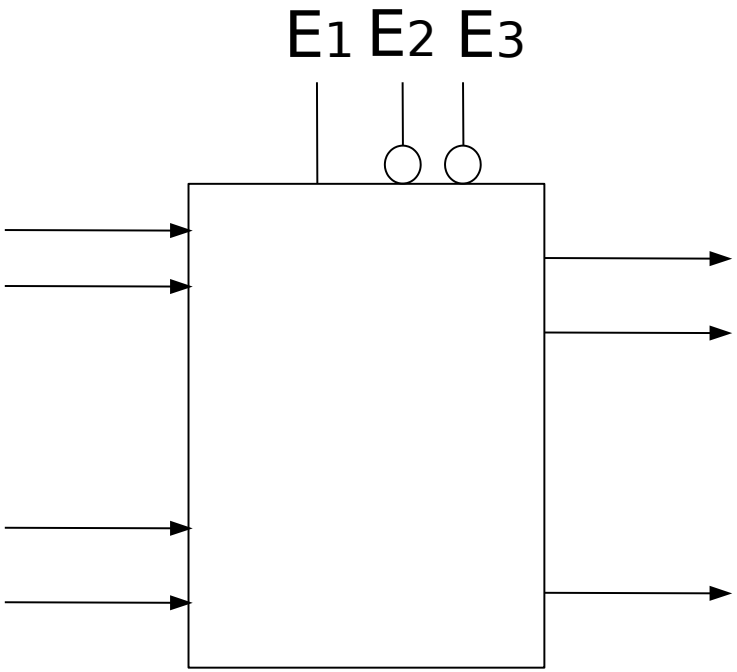


Aspectos generales (entradas y salidas)

- Señales de control:

  condicionan el funcionamiento del subsistema (habilitan, inhiben, etc)

- Niveles de activación de las señales: en alto o en bajo



Ejemplo:

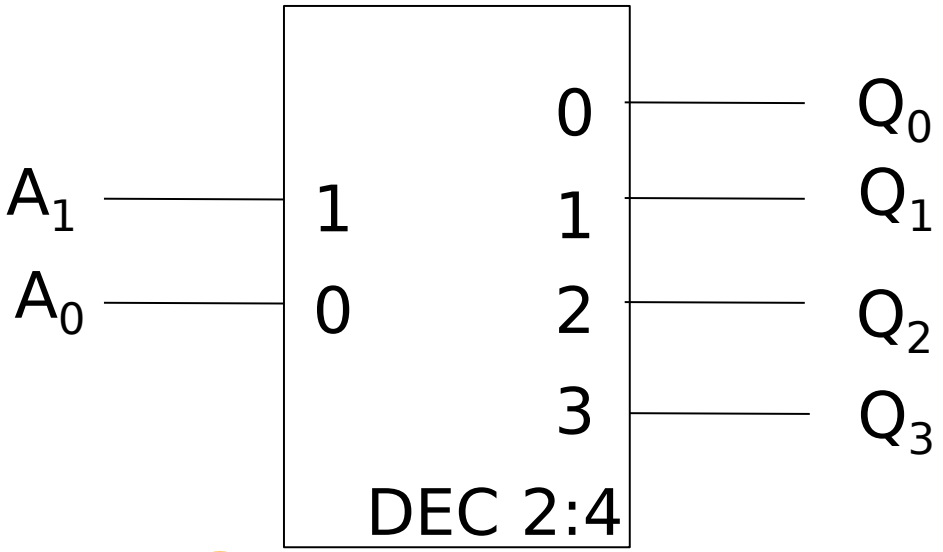
- E1, ENABLE activo en alto
- E2 y E3, ENABLES activos en bajo
- Activado, si y solo si:

  E1=1, E2=E3=0

Subsistemas de propósito específico: Decodificadores

- Su función es“decodificar” un dato de entrada
- Tienen n entradas y m salidas ($m \leq 2^n$)
- Se nombran DEC n:m
- Para cada combinación de entrada solo una salida se activa (toma un valor distinto al de las demás)

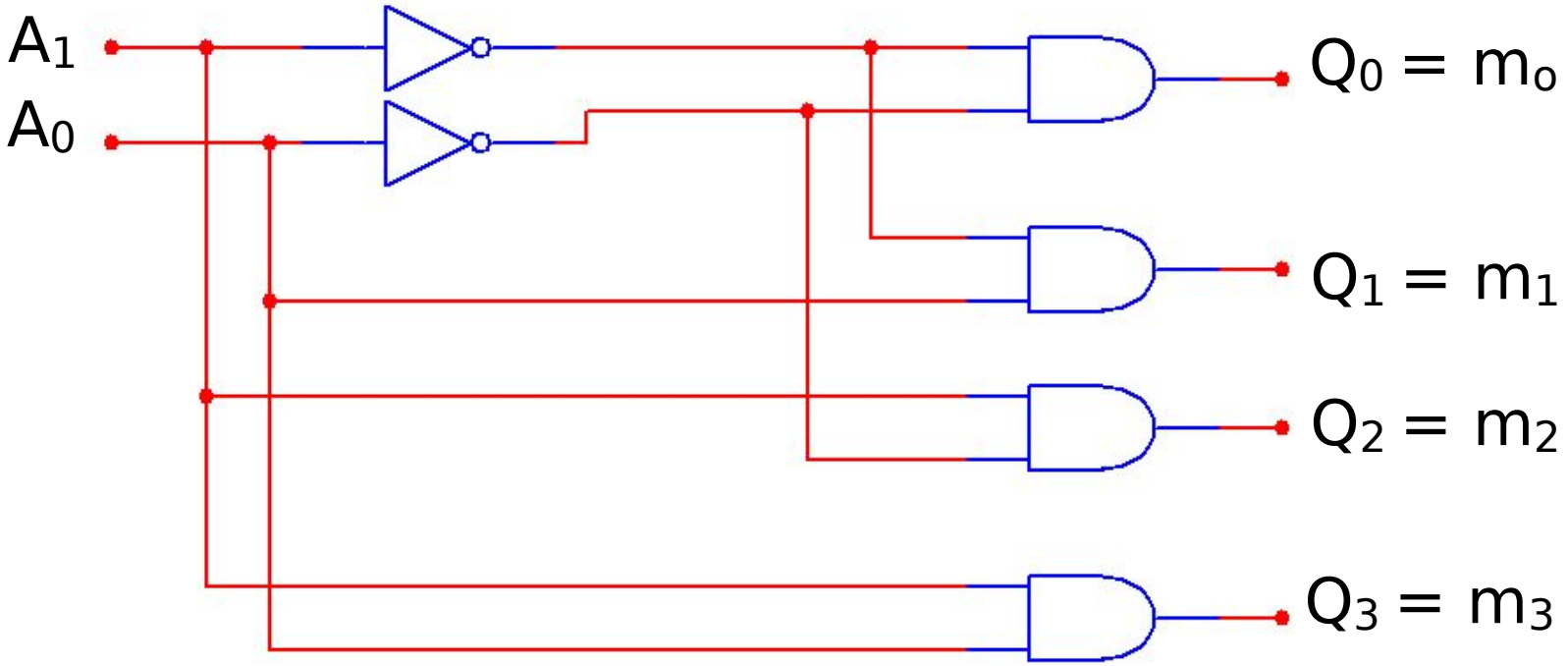
Ejemplo: DEC 2:4 con salidas activas en alto



A1	A0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decodificadores (diseño interno)

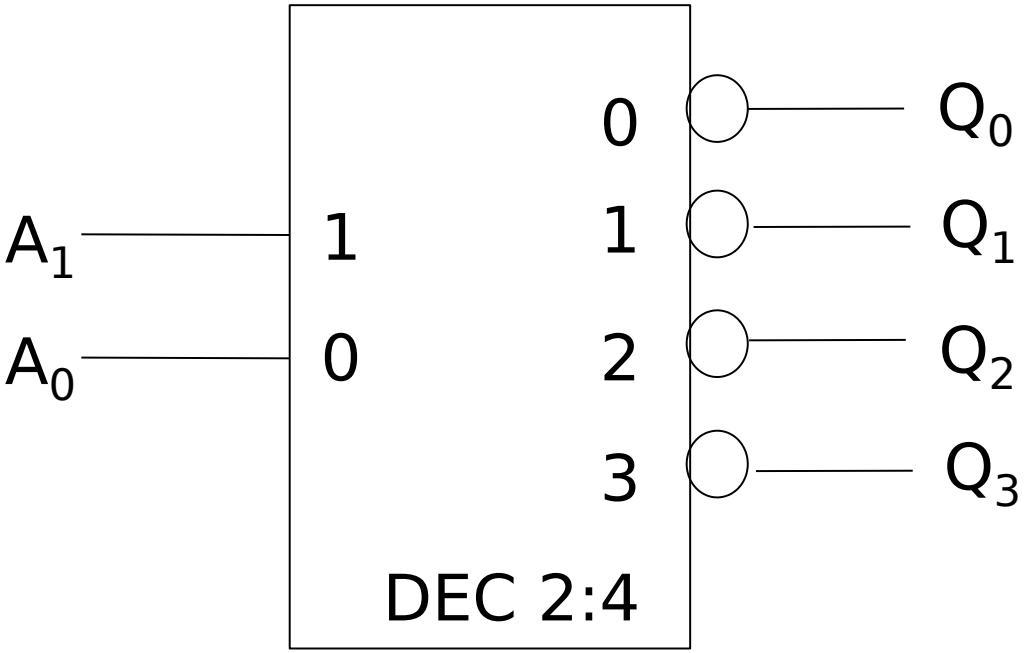
Ejemplo: *DEC 2:4 con salidas activas en alto*



(a partir de su tabla de verdad)

Decodificadores (funciones de salida)

Ejemplo: DEC 2:4 con salidas activas en bajo

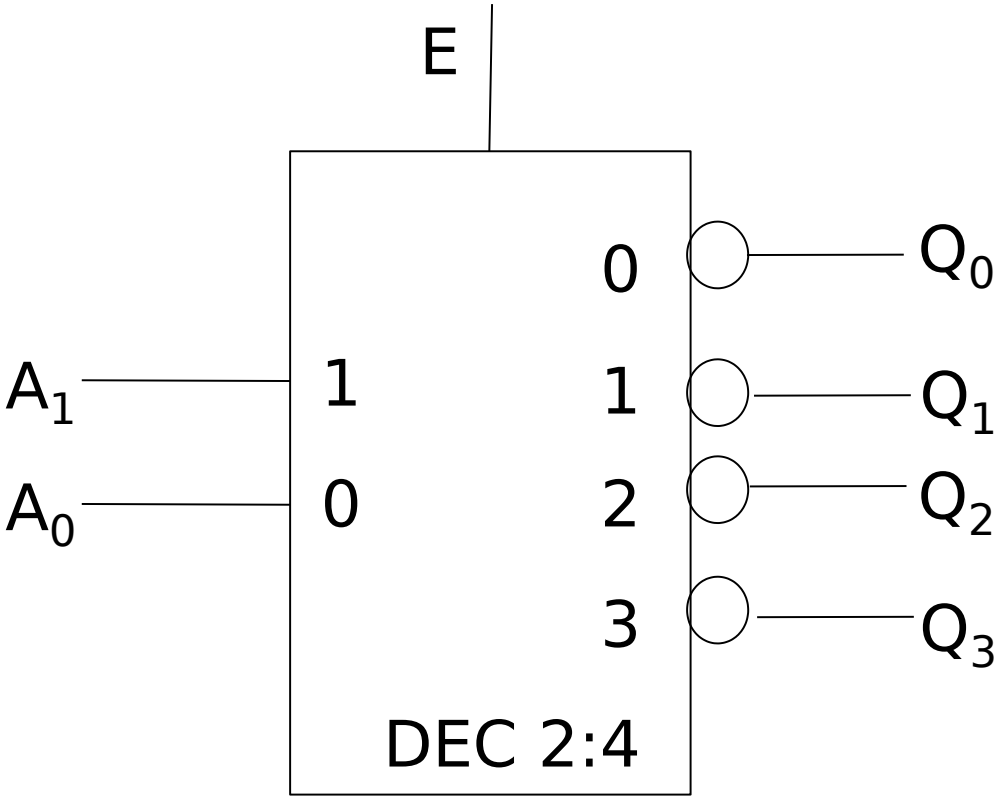


A1	A0	Q0	Q1	Q2	Q3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

- DEC (**salidas en alto**): genera **mintérminos** ($Q_i = m_i$)
- DEC (**salidas en bajo**): genera **Maxtérminos** ($Q_i = M_i$)

Decodificadores (funciones de salida)

Ejemplo: DEC 2:4 con salidas activas en bajo (con enable en alto)

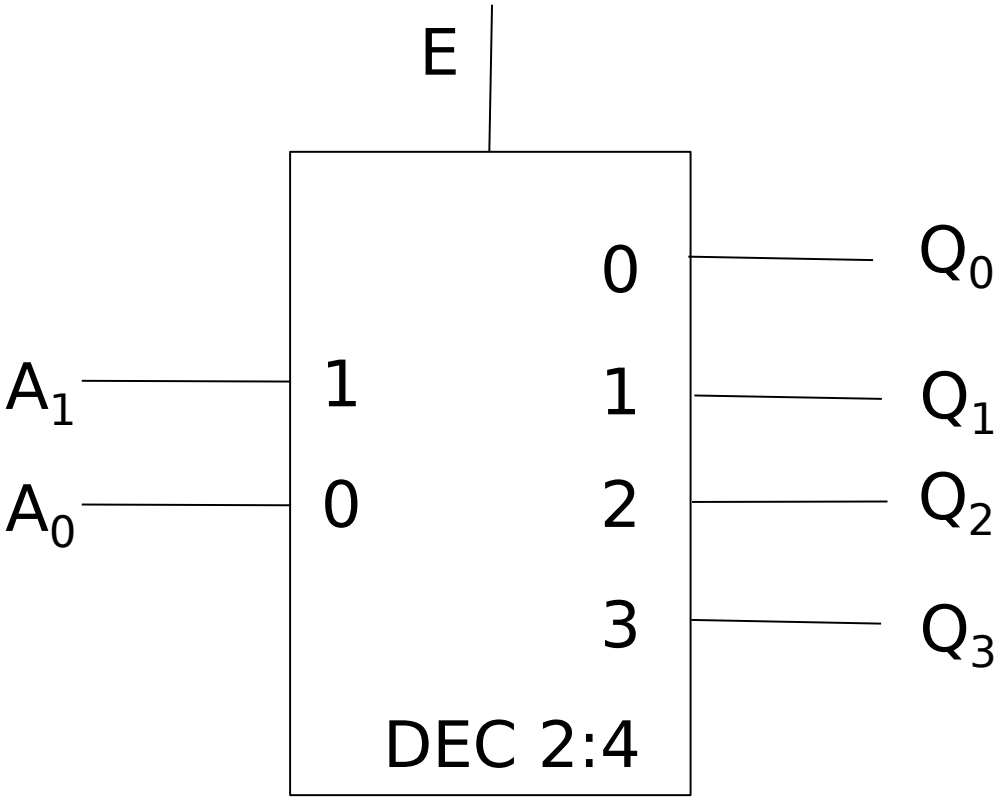


E	A1	A0	Q0	Q1	Q2	Q3
0	X	X	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	1	1	1	0

$$Q_i = M_i + E$$

Decodificadores (funciones de salida)

Ejemplo: DEC 2:4 con salidas activas en alto (con enable en alto)



E	A1	A0	Q0	Q1	Q2	Q3
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

$$Q_i = m_i \cdot E$$

Decodificador (descripción Verilog)

Ejemplo: Descripción Verilog "flujo de datos"

DEC 2:4 con salidas activas en alto (con enable en alto)

```
// Decodificador 2:4 con salidas en alto y  habilitación en alto
// descripción verilog "flujo de datos"

module decodificador_2_a_4_df_v(EN, A, Q);
    input EN, [1:0]A;
    output [3:0]Q;

    assign Q[0]= EN & ~A[1] & ~A[0];
    assign Q[1]= EN & ~A[1] & A[0];
    assign Q[2]= EN & A[1] & ~A[0];
    assign Q[3]= EN & A[1] & A[0];

endmodule
```

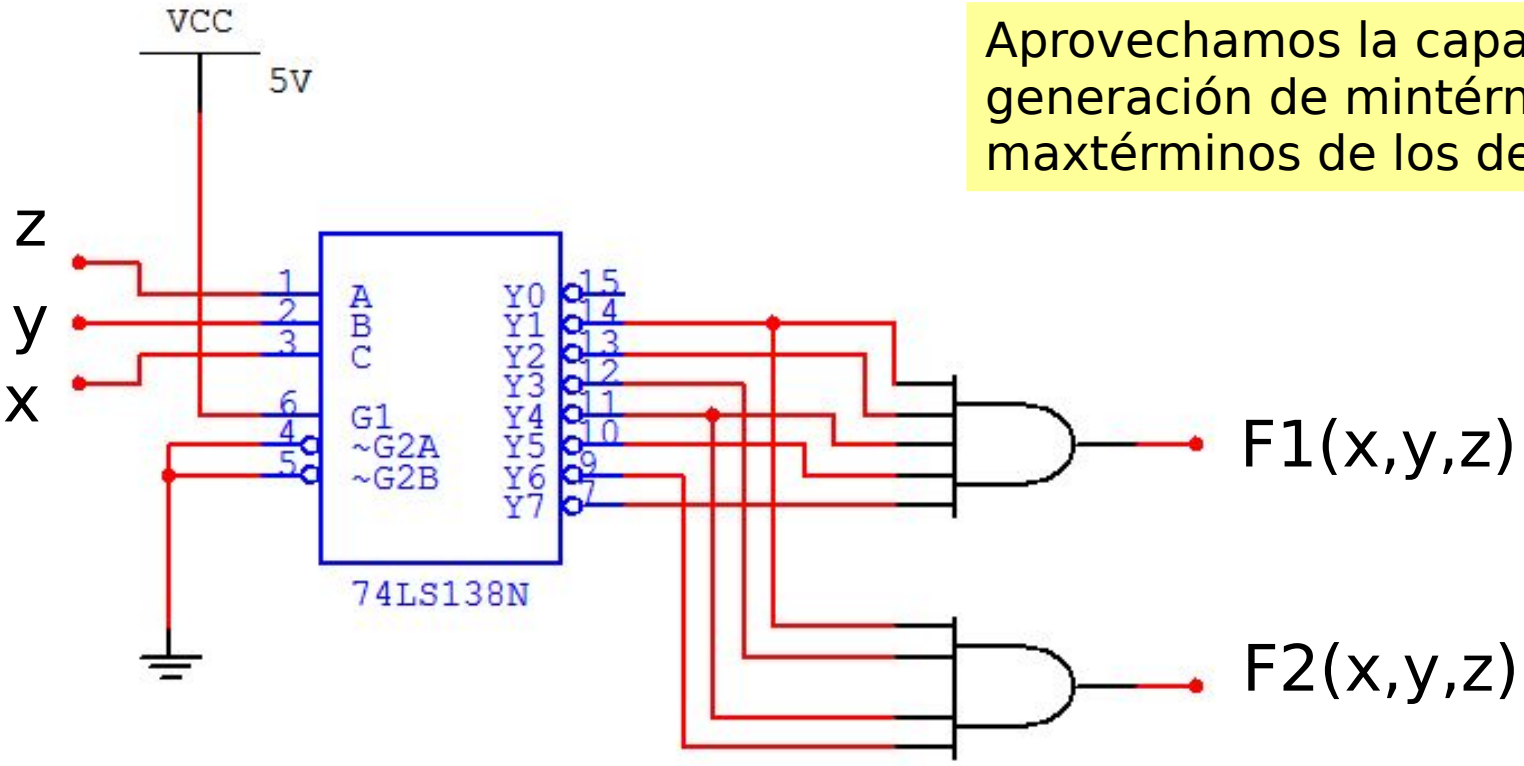
Diseño con decodificadores y puertas

Ejemplo: Realice las siguientes funciones con decodificadores y puertas

$$F1(x,y,z) = \sum(0,3,6)$$

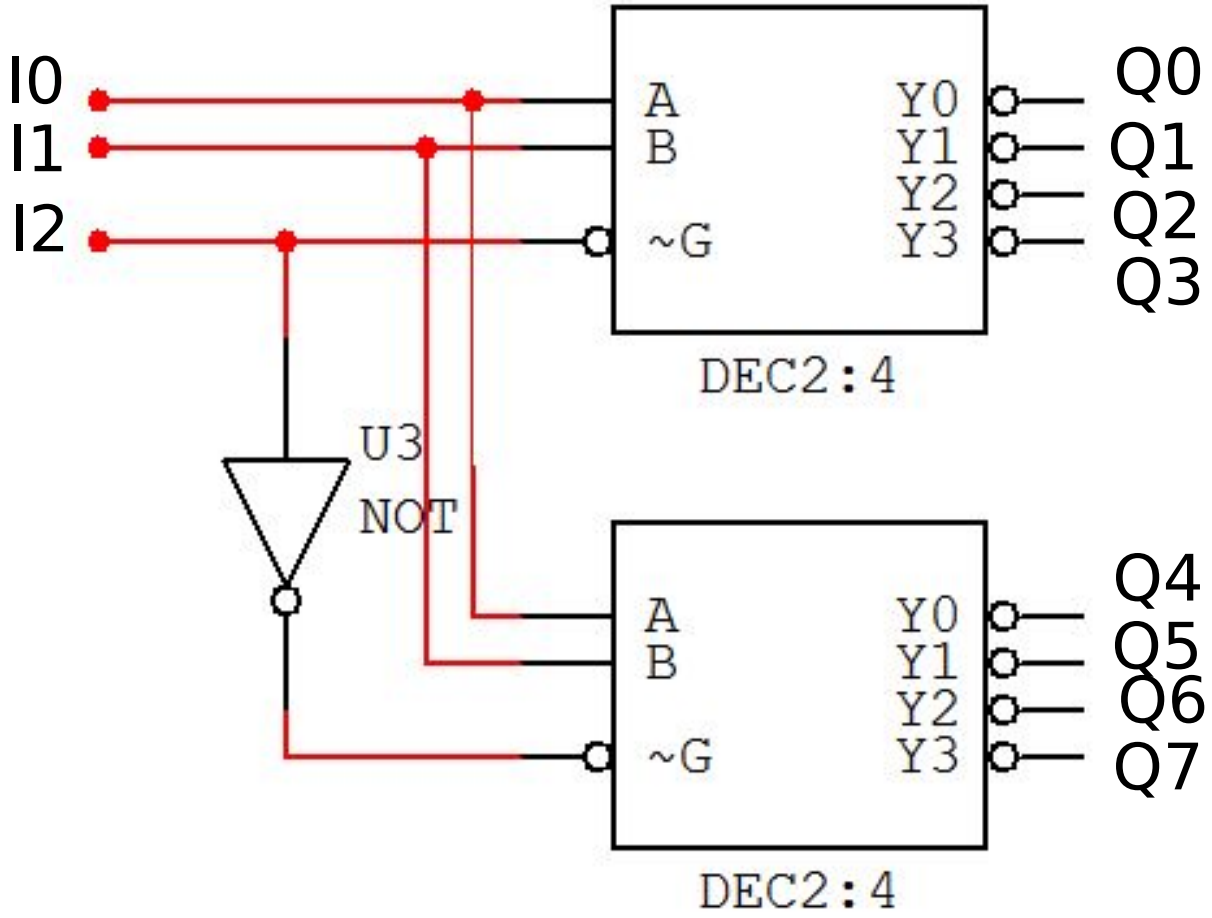
$$F2(x,y,z) = \prod(1,3,4,6)$$

Aprovechamos la capacidad de generación de mintérminos o maxtérminos de los decodificadores



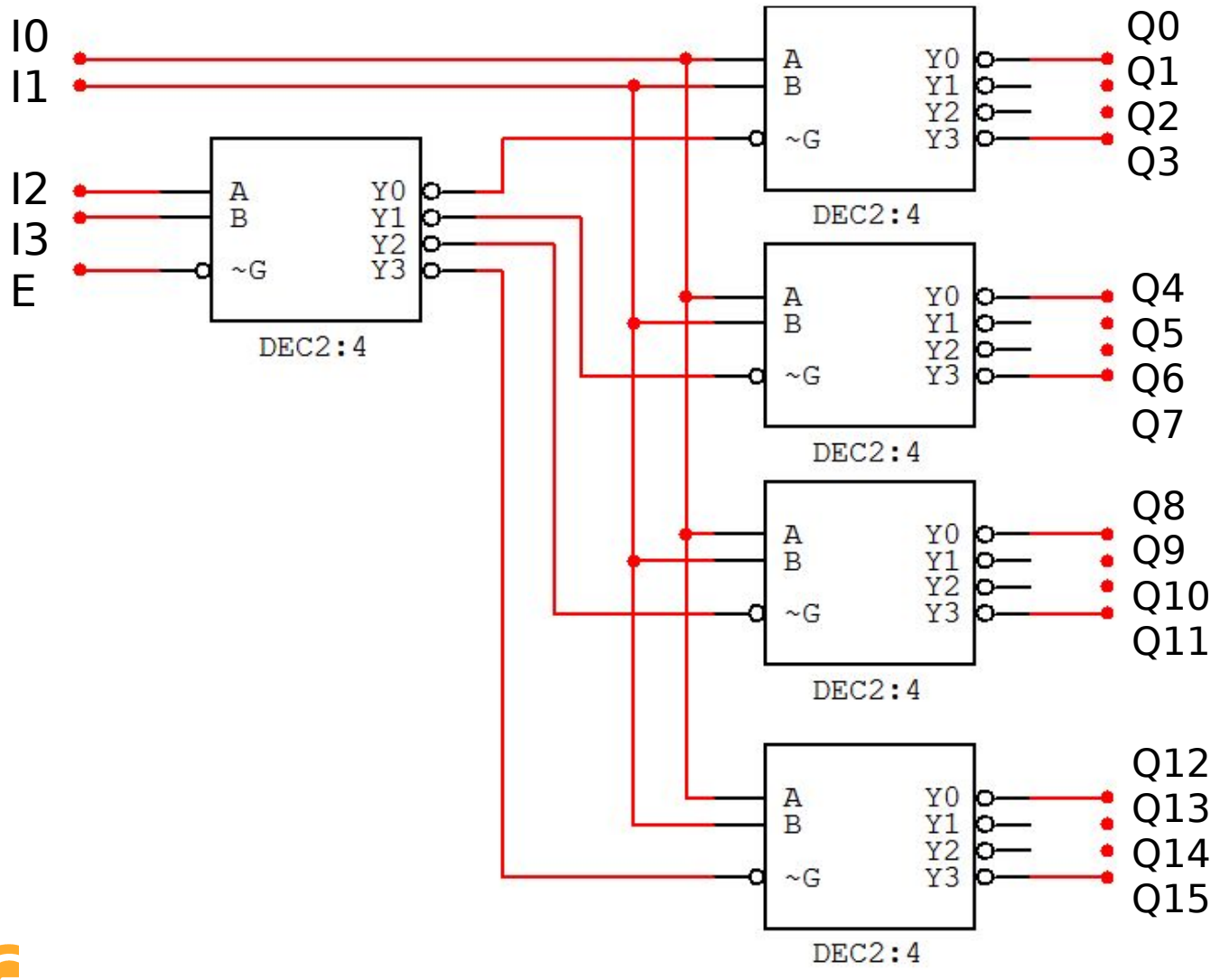
Asociación de decodificadores

Ejemplo: Realizar un DEC 3:8 a partir de dos DEC 2:4



Asociación de decodificadores (i)

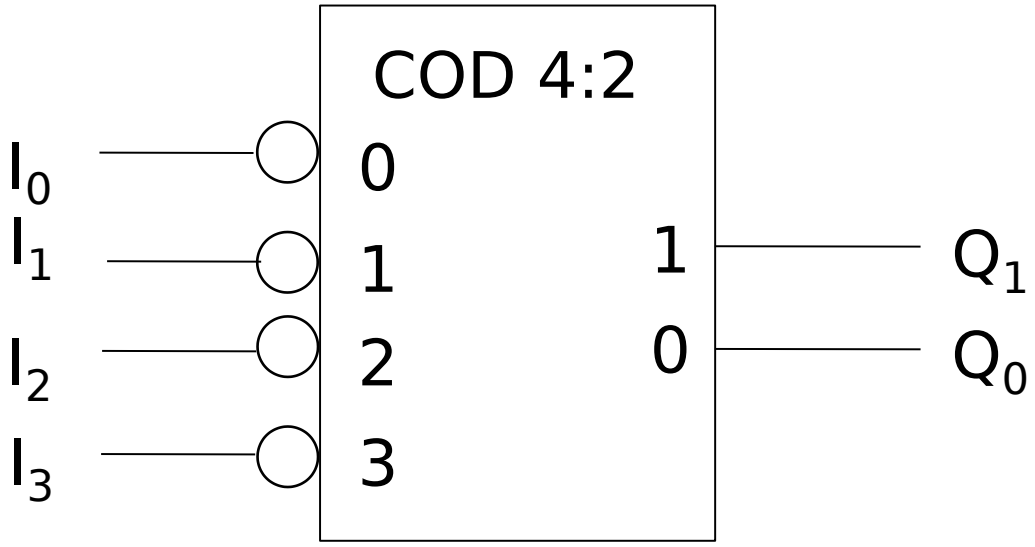
Ejemplo: Realizar un DEC 4:16 a partir de DEC 2:4



Subsistemas de propósito específico: Codificadores

- Realizan la función inversa a los decodificadores:
  - admiten solo una entrada activada
  - nos muestra cuál es en un código concreto

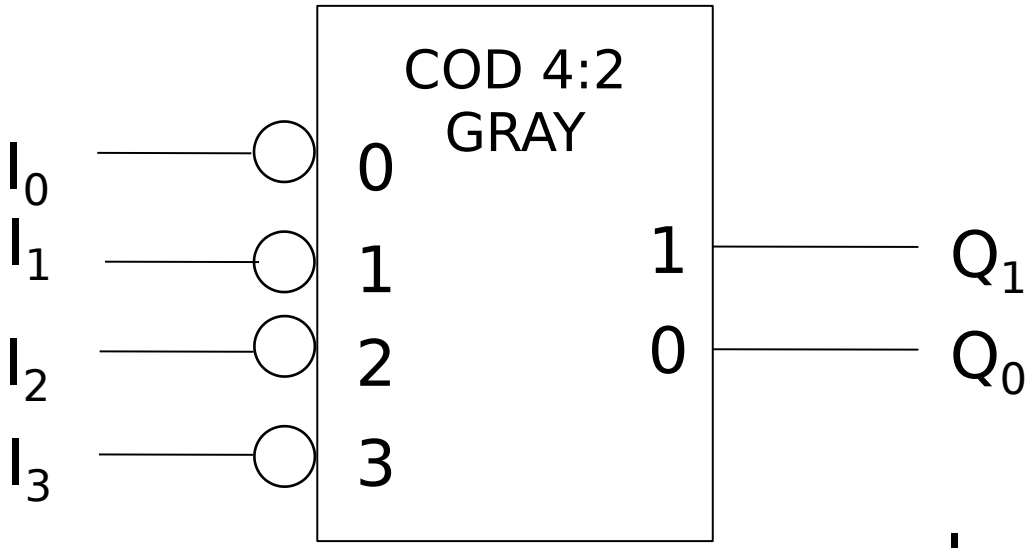
Ejemplo: COD 4:2 con entradas activas en bajo y salidas activas en alto (binario natural)



I0	I1	I2	I3	Q1	Q0
0	1	1	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	0	1	1

Codificadores (diseño interno)

Ejemplo: Realizar un COD 4:2 con entradas activas en bajo y salidas activas en alto (GRAY)

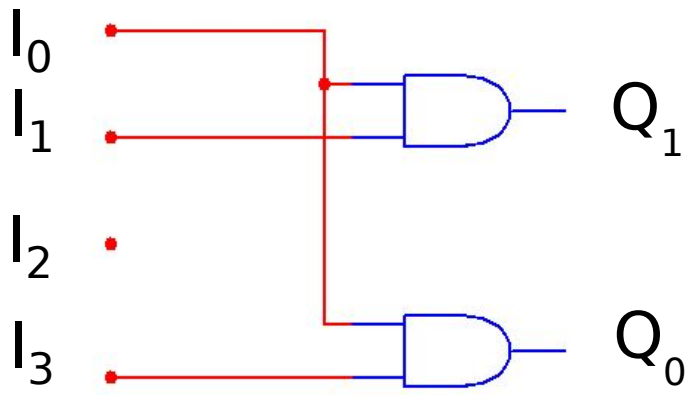


I0	I1	I2	I3	Q1	Q0
0	1	1	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	0	1	0

Simplificando k-mapa:

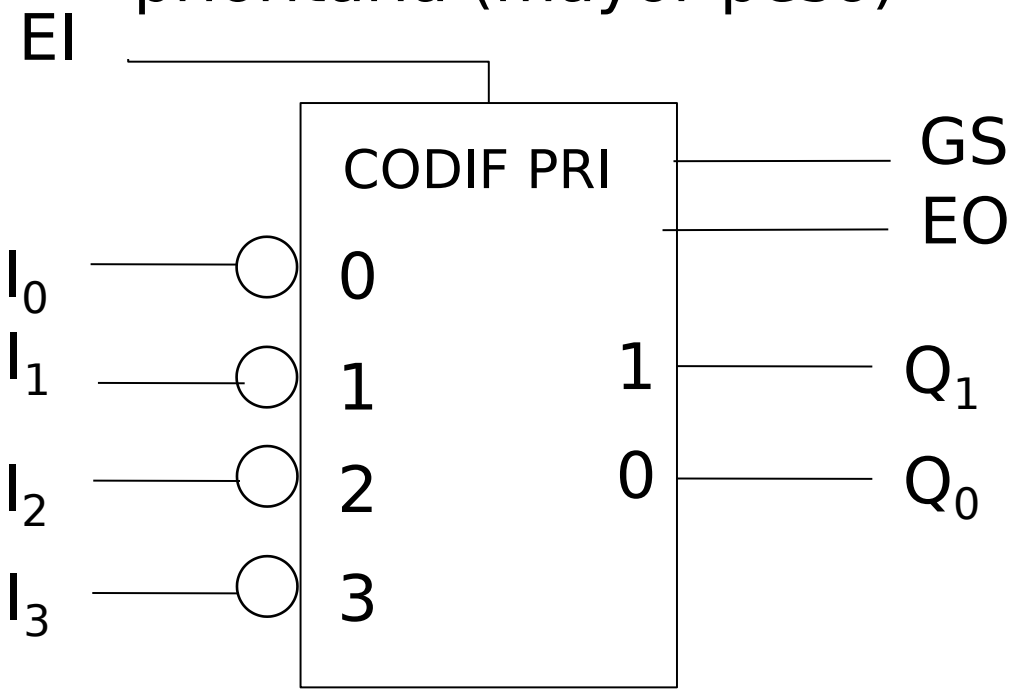
$Q_1 = I_0 I_1$

$Q_0 = I_0 I_3$



Subsistemas de propósito específico: Codificadores de prioridad

- Son codificadores que admiten más de una entrada activa en cada momento.
- Generan a la salida el código de la entrada más prioritaria (mayor peso)



EI	I0	I1	I2	I3	Q1	Q0	GS	E0
0	X	X	X	X	0	0	0	0
1	0	1	1	1	0	0	1	0
1	X	0	1	1	0	1	1	0
1	X	X	0	1	1	0	1	0
1	X	X	X	0	1	1	1	0
1	1	1	1	1	0	0	0	1

GS: indica si el codificador está habilitado y hay alguna entrada activa
EO: indica si no hay ninguna entrada activa

Codificador de prioridad(descripción Verilog)

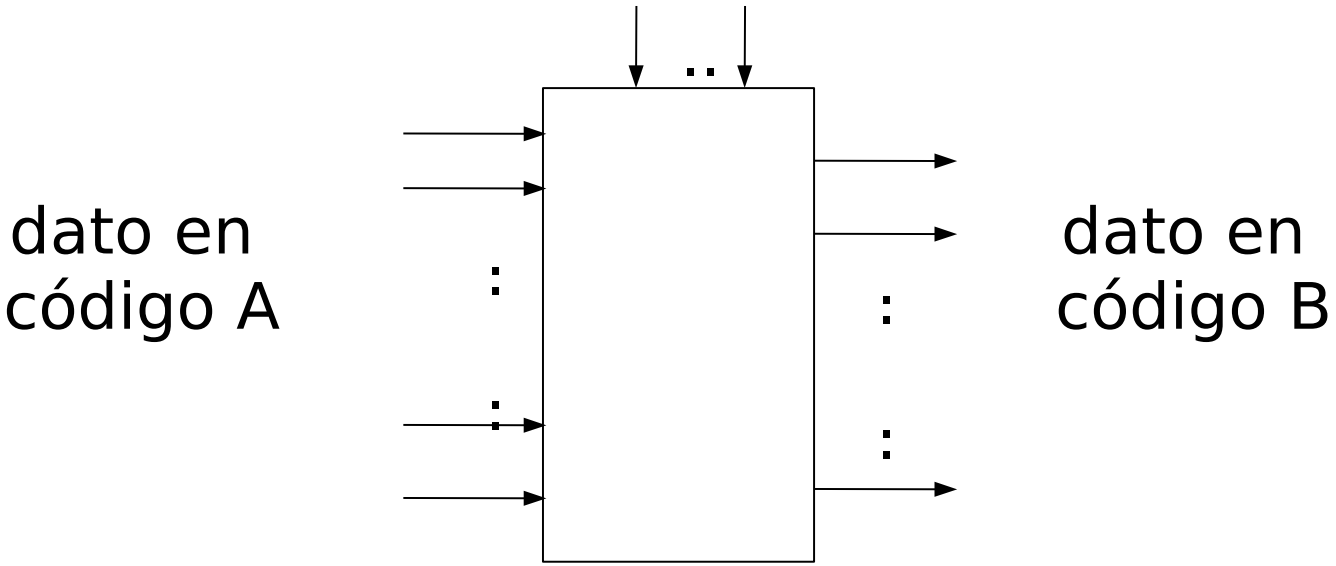
```
 module cod8(
  input [7:0] in,output reg [2:0] out, output reg e);

always @(in) begin
      e = 0;
   /* 'case' también es una buena forma de describir el
    * codificador porque la decisión se toma en función de una
    * única señal (in). 'casex'es como 'case' pero trata valores
    * desconocidos ('x') como inespecificaciones, lo que permite
    * expresar de forma muy compacta las comparaciones. */
       casex (in)
           8'b1xxxxxxx: out = 3'h7;
           8'b01xxxxxx: out = 3'h6;
           8'b001xxxxx: out = 3'h5;
           8'b0001xxxx: out = 3'h4;
           8'b00001xxx: out = 3'h3;
           8'b000001xx: out = 3'h2;
           8'b0000001x: out = 3'h1;
           8'b00000001: out = 3'h0;
           default: begin        // ninguna entrada activa
                   out = 3'h0;
                   e = 1;
           end
           endcase
     end
endmodule // cod8
```

Subsistemas de propósito específico: Convertidores de código

Su función cambiar la codificación del dato de entrada de un código binario a otro

Ejemplos: *Binario/gray, Gray/binario, BCD/7 segmentos*

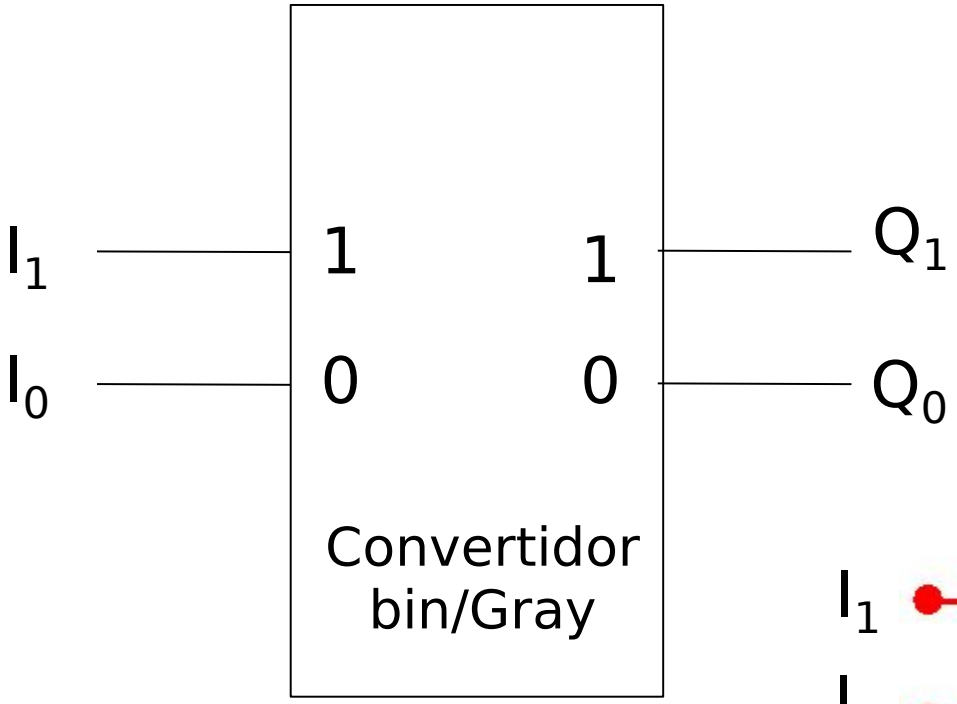


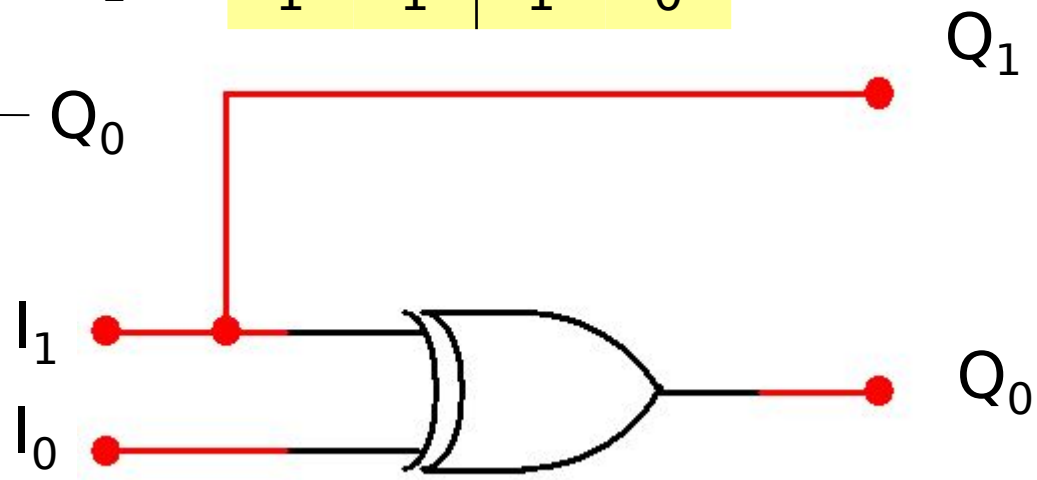
Convertidores de código (diseño interno)

Opción 1: a partir de su tabla de verdad

Ejemplo: Convertidor binario/Gray (2 bits)

I1	I0	Q1	Q0
0	1	0	0
0	1	0	1
1	0	1	1
1	1	1	0

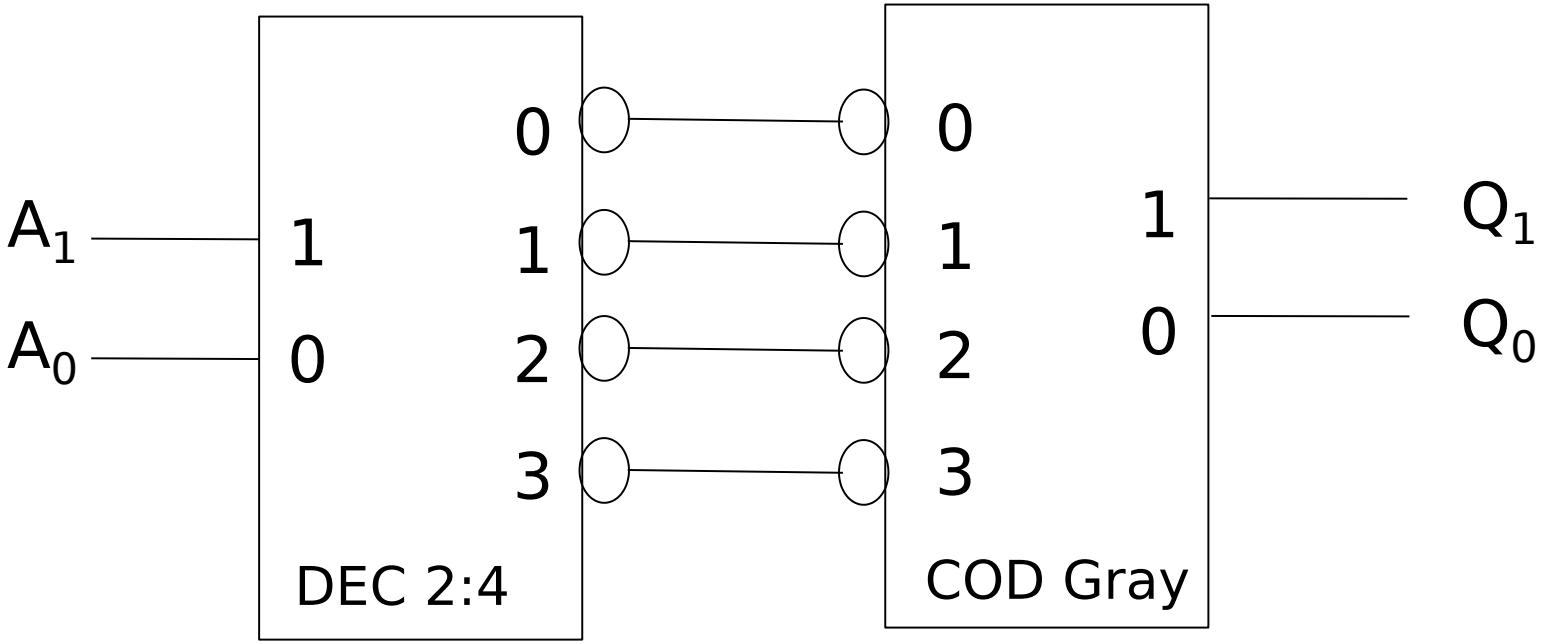




Diseño de convertidores de código

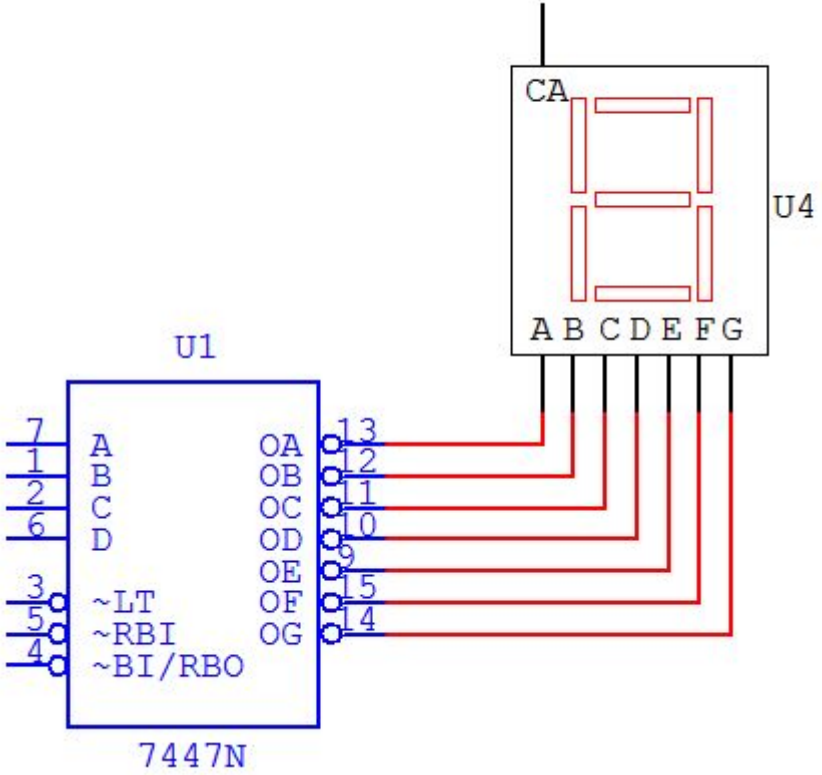
Opción 2: Asociando DEC-COD (con los códigos adecuados)

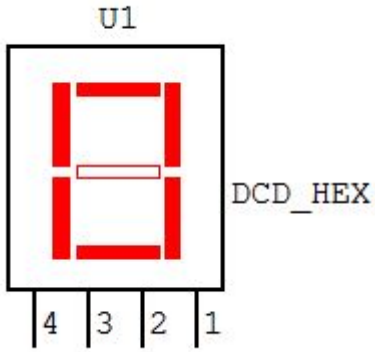
Ejemplo: Convertidor binario/Gray (2 bits)



Convertidores de código comerciales

7447 Convertidor BCD a 7 segmentos

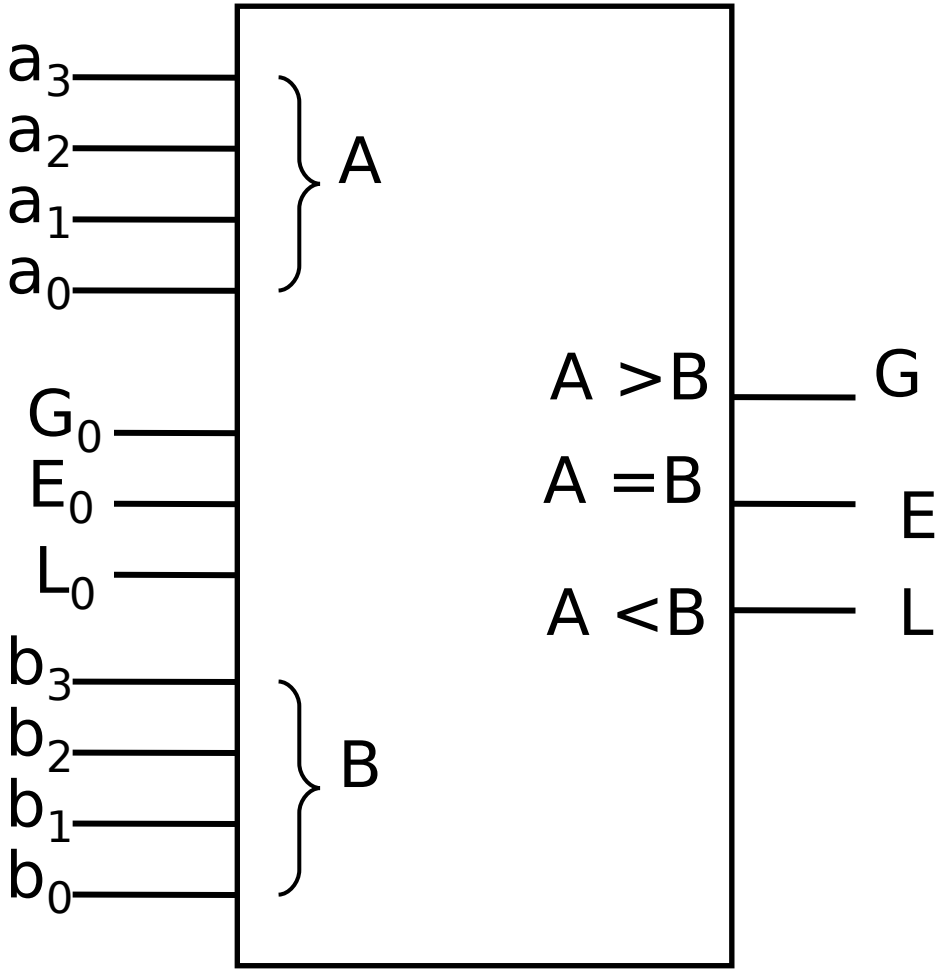




Visualizador
(con convertidor incluido)

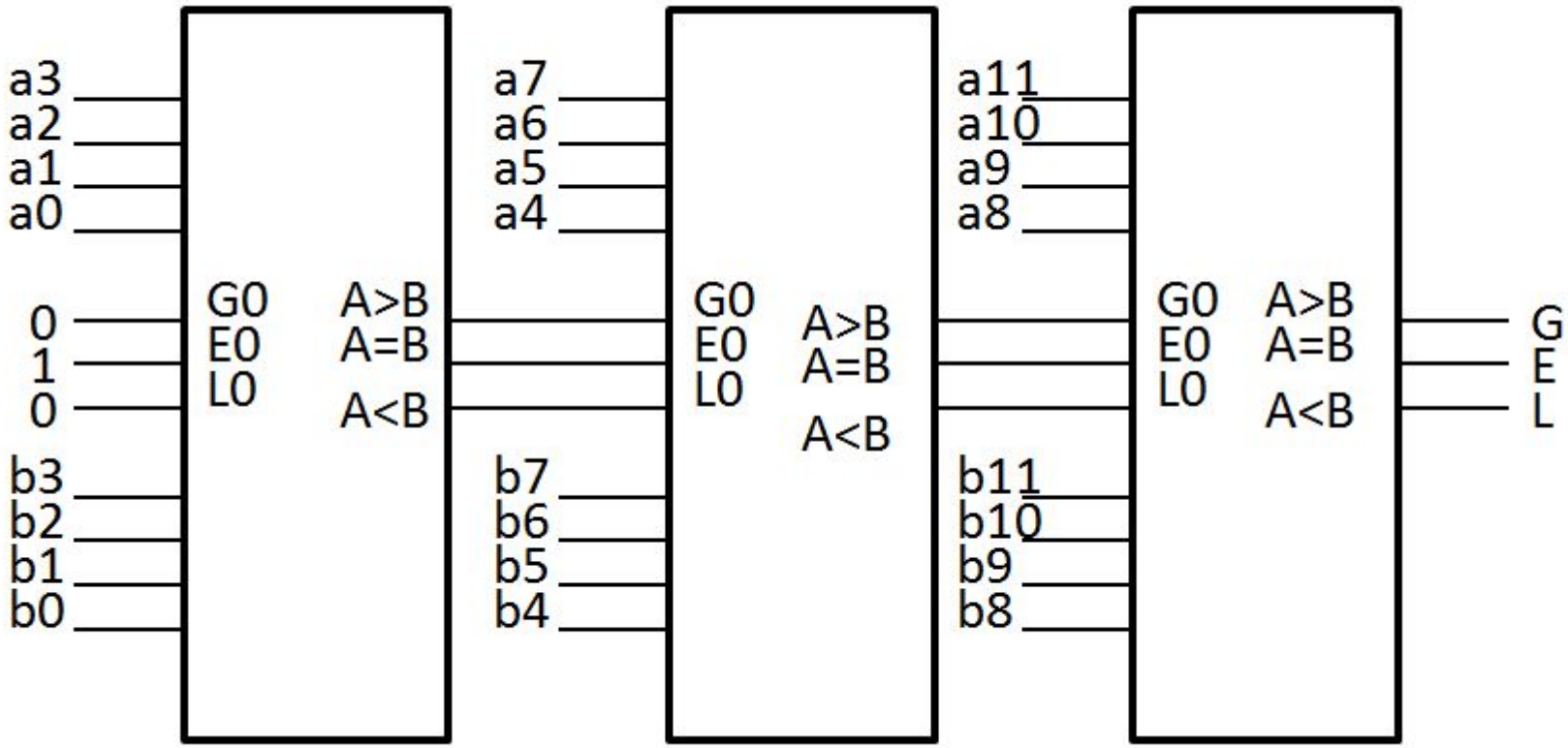
Subsistemas de propósito específico: Comparadores de magnitud

Comparan las magnitudes de dos números de n bits



A B	G	E	L
A>B	1	0	0
A=B	G_0	E_0	L_0
A<B	0	0	1

Conexión en cascada de comparadores



Conexión en cascada para formar un comparador de números de 12 bits

Comparador de magnitudes en Verilog

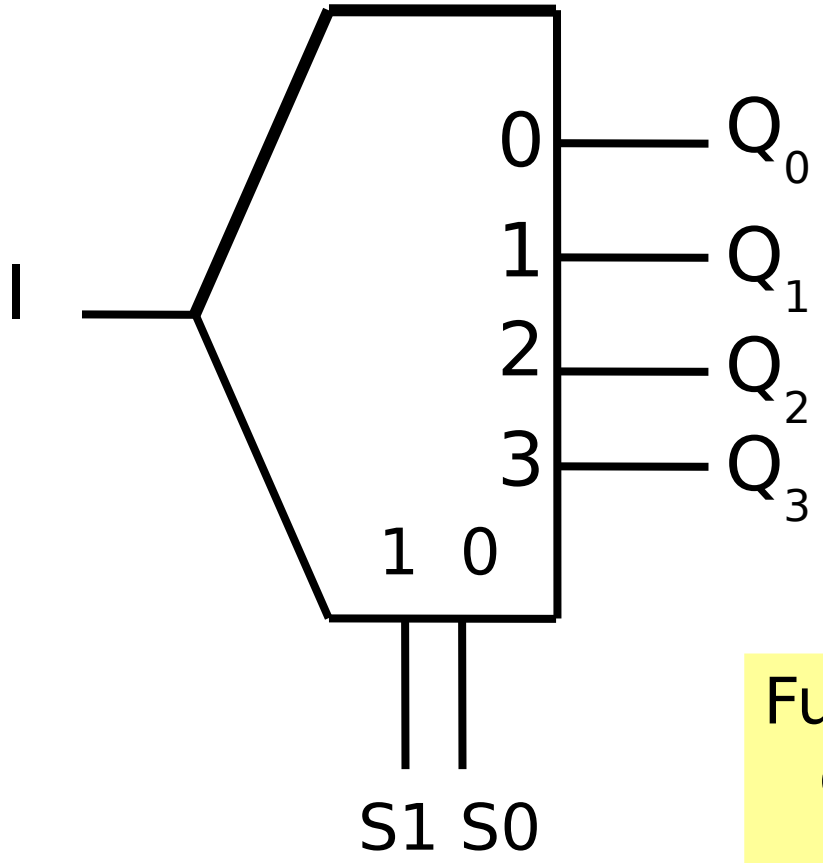
```
module comp4(
        input [3:0] a,          // número a
        input [3:0] b,          // número b
        input g0,               // entradas para conexión en cascada
        input e0,               // y salidas de la comparación
        input l0,               //   si a > b => (g,e,l) = (1,0,0)
        output g,               //   si a < b => (g,e,l) = (0,0,1)
        output e,               //   si a = b => (g,e,l) = (g0,e0,l0)
        output l
);

        reg g, e, l;

        /* Obsérvese cómo se ha empleado el operador de concatenaciónn '{...}'
         * que combina varios vectores para formar un vector mayor. Aquí se ha
         * empleado para hacer más claras y compactas las asignaciones de
         * valores a 'g', 'e' y 'l'. */
        always @(a, b, g0, e0, l0) begin
                if (a > b)
                        {g,e,l} = 3'b100;
                else if (a < b)
                        {g,e,l} = 3'b001;
                else
                        {g,e,l} = {g0,e0,l0};
        end
endmodule
```

Subsistemas de propósito específico: Demultiplexores

Permiten llevar la información de una línea de entrada, a alguno de los 2^n canales de salida, usando n líneas de selección



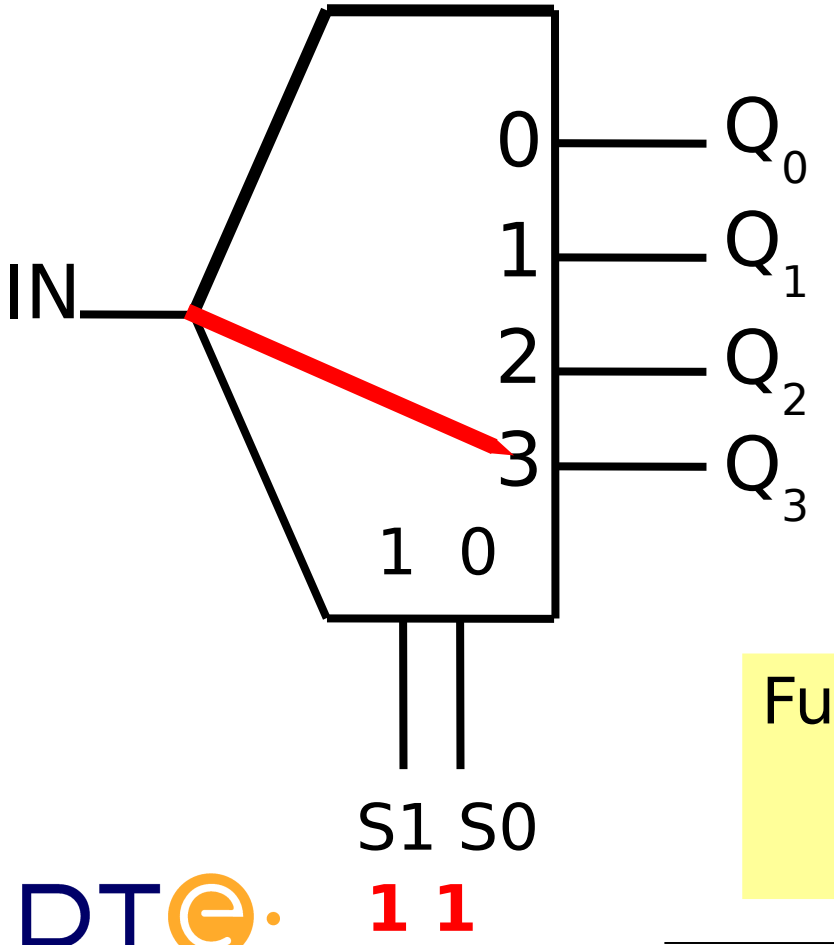
DEMUX 1:4

I	S1	S0	Q0	Q1	Q2	Q3
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

Funcionalmente equivalente a un decodificador con habilitación (DEC2:4/DEMUX1:4)

Subsistemas de propósito específico: Demultiplexores

Permiten llevar la información de una línea de entrada, a alguno de los 2^n canales de salida, usando n líneas de selección



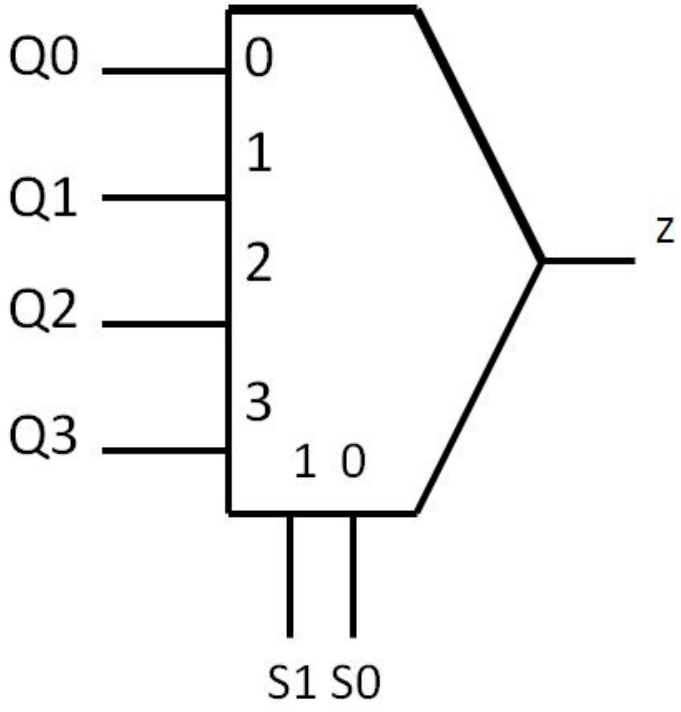
DEMUX 1:4

IN	S1	S0	Q0	Q1	Q2	Q3
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

Funcionalmente equivalente a un Decodificador (DEC2:4/DEMUX1:4)

Subsistemas de propósito general: Multiplexor MUX 2^n:1 (MUX n)

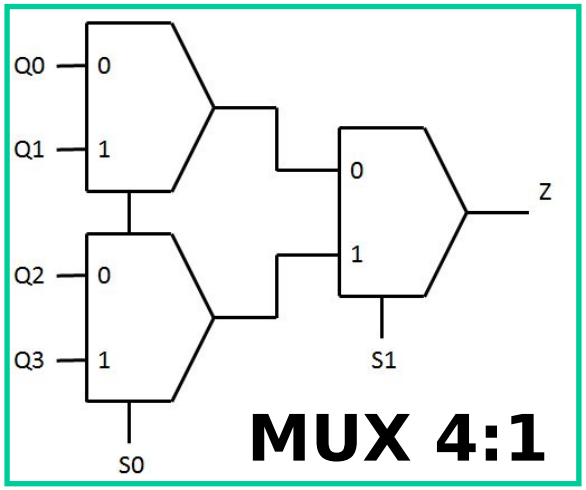
Dispone de 2^n canales de entrada, 1 línea de salida y n entradas de selección de canal, que permiten elegir el dato de entrada que estará presente en la salida

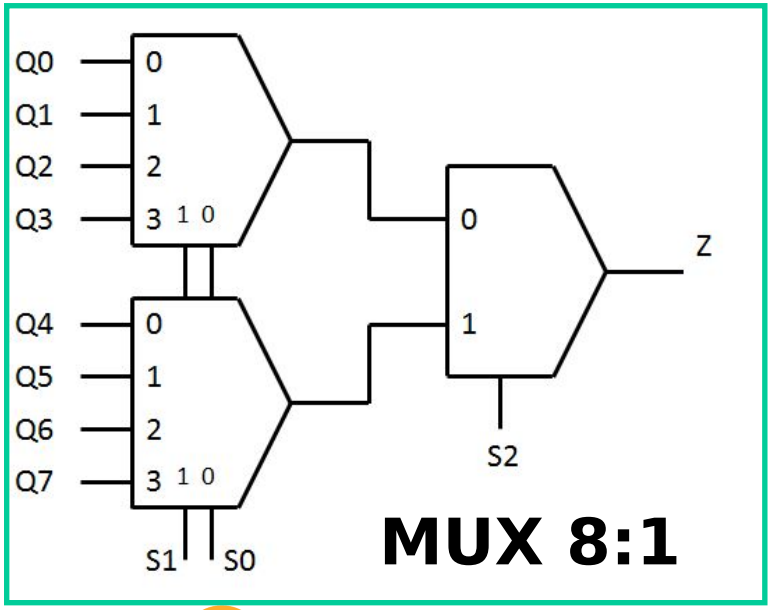


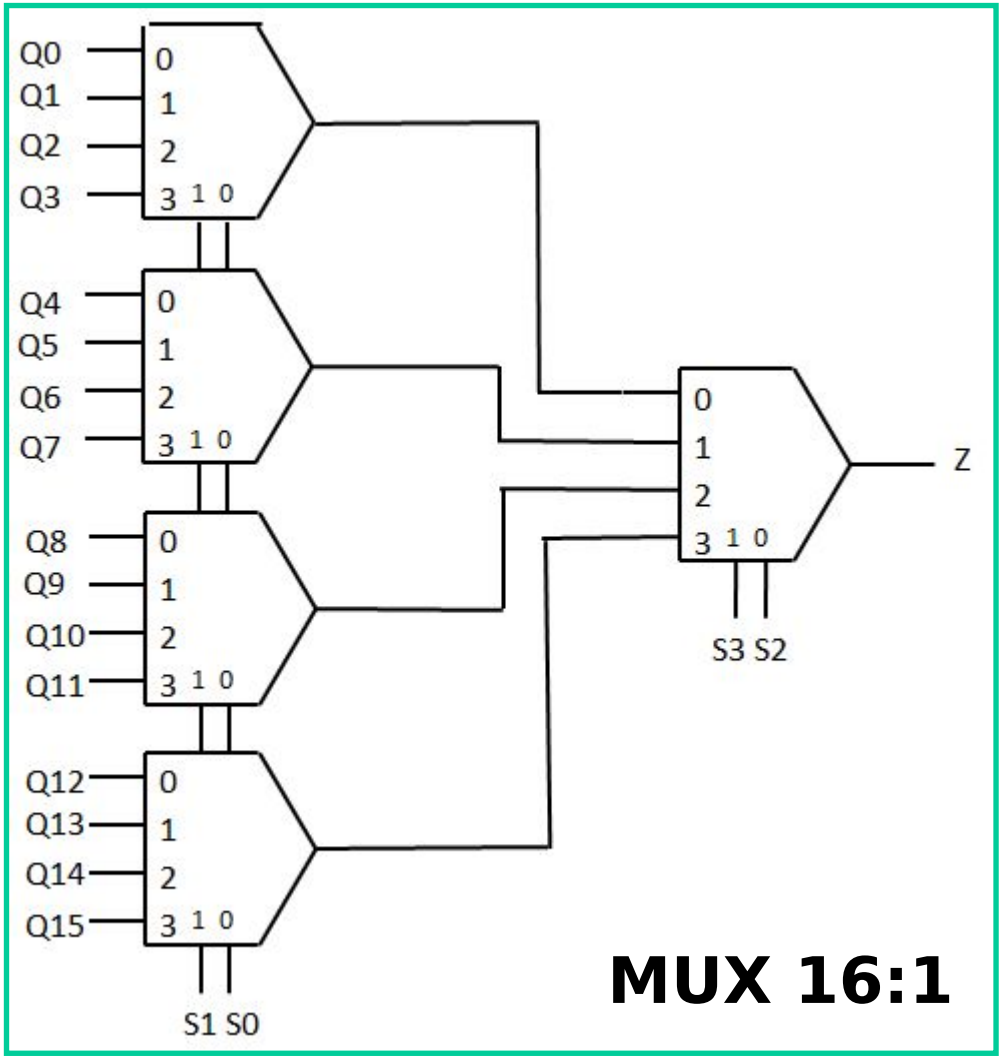
S1	S0	Z
0	0	Q0
0	1	Q1
1	0	Q2
1	1	Q3

$$Z = \overline{S_1}\,\overline{S_0}Q_0 + \overline{S_1}S_0Q_1 + S_1\overline{S_0}Q_2 + S_1S_0Q_3$$

Asociación de multiplexores



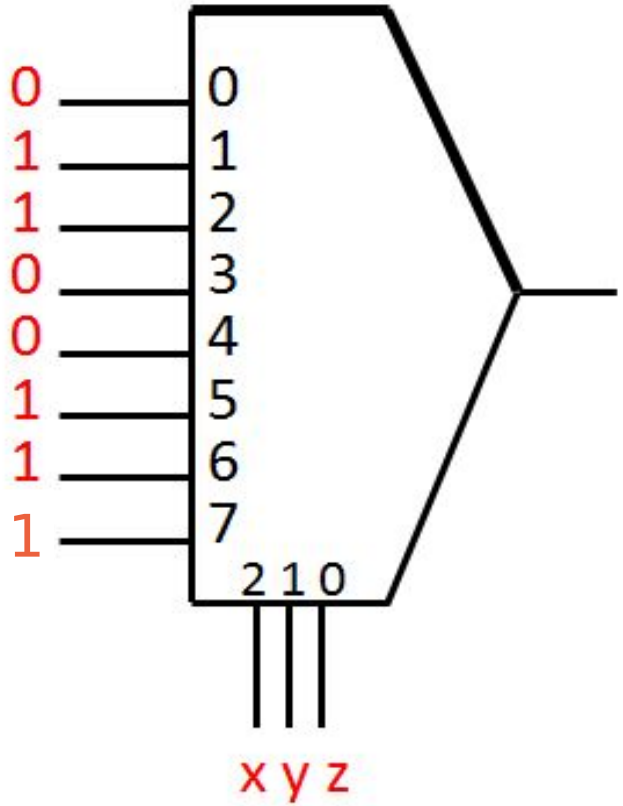




Realización de funciones de *n* variables con multiplexores

MUX n: permite implementar cualquier función de **n variables**

Ejemplo 1: Función de 3 variables con MUX3

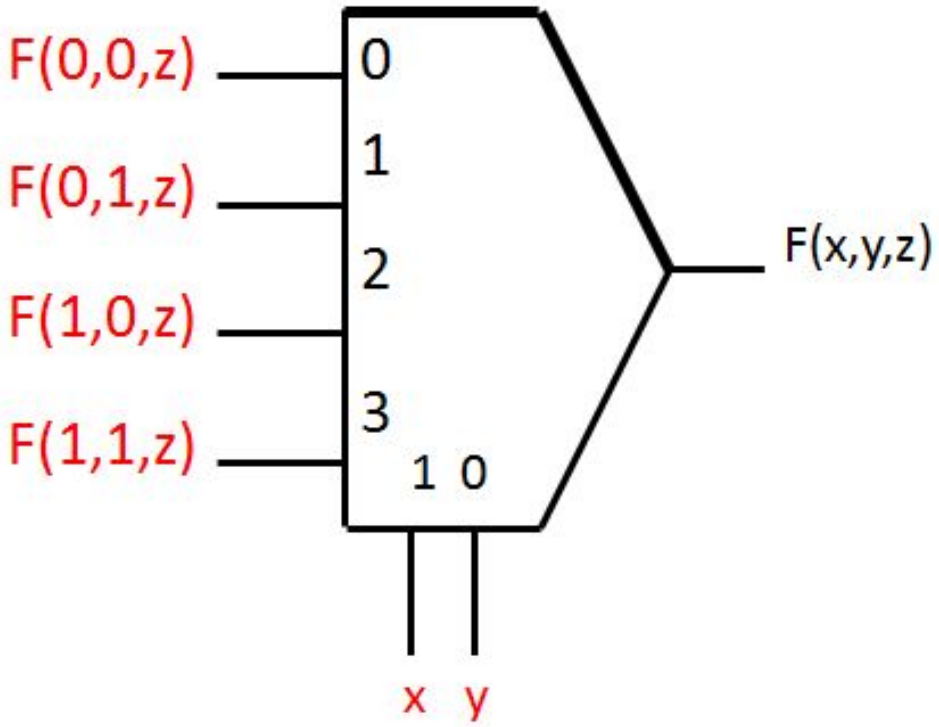


$F(x,y,z) = \sum(1,2,5,6,7)$

Realización de funciones de n variables con Multiplexores (ii)

MUX n-1 + NOT: permite implementar cualquier función de **n variables**

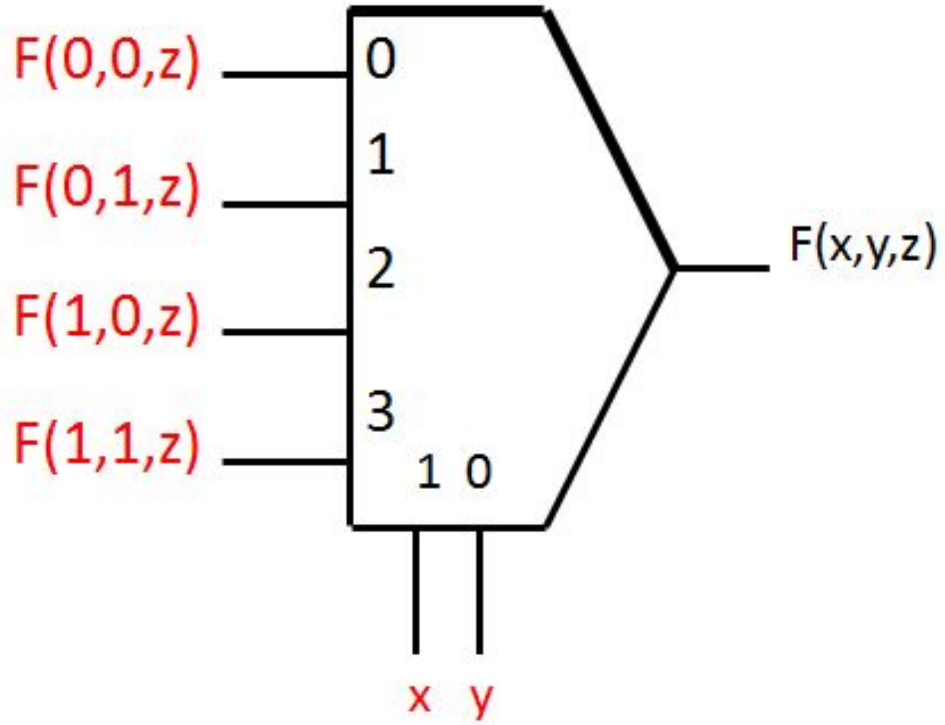
Ejemplo 2: Función de 3 variables con MUX 2



z \ xy	00	01	11	10
0	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**1**

Realización de funciones de n variables con Multiplexores (ii)

Ejemplo 2: Función de 3 variables con MUX 2 (continuación)

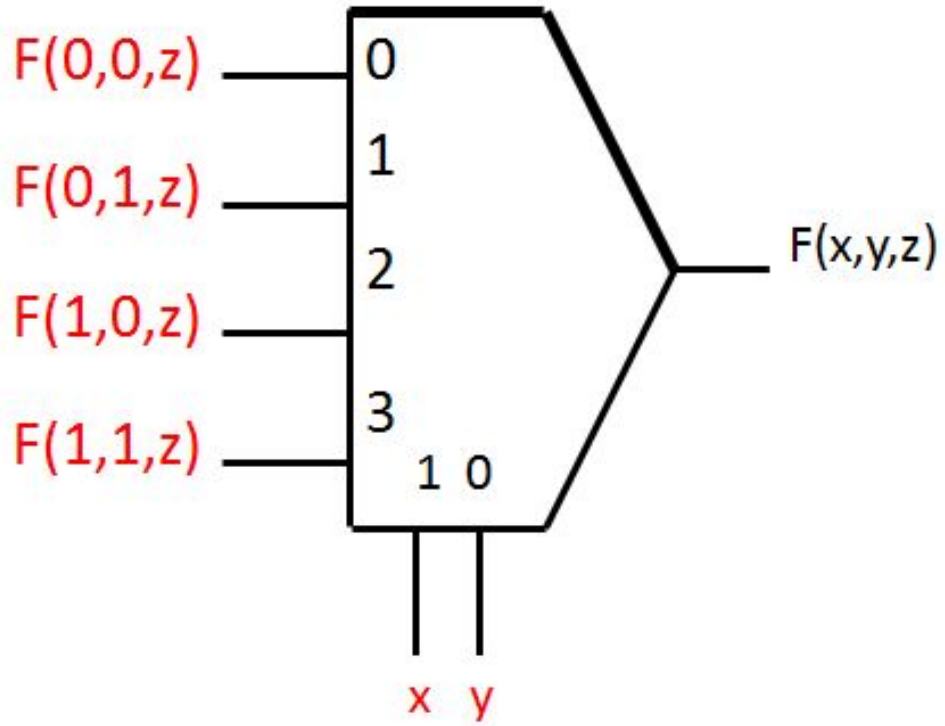


z \ xy	00	01	11	10
0	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**1**

$F(0,0,z) = z$

Realización de funciones de n variables con Multiplexores (ii)

Ejemplo 2: Función de 3 variables con MUX 2 (continuación)



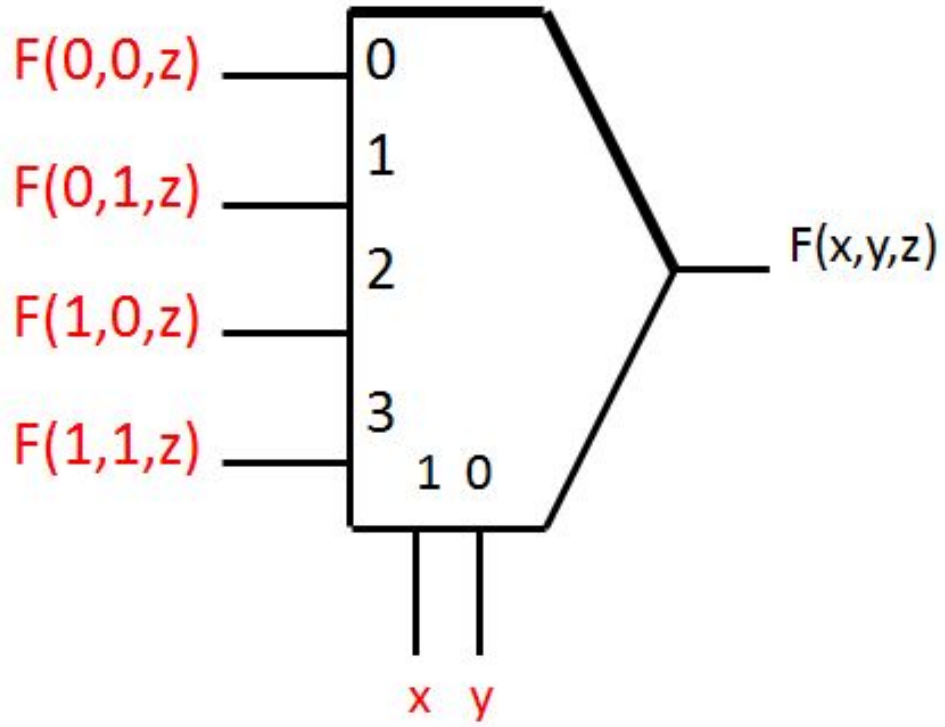
z \ xy	00	01	11	10
0	0	1	1	0
1	1	0	1	1

$F(0,0,z) = z$

$F(0,1,z) = \overline{z}$

Realización de funciones de n variables con Multiplexores (ii)

Ejemplo 2: Función de 3 variables con MUX 2 (continuación)



z \ xy	00	01	11	10
0	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**1**

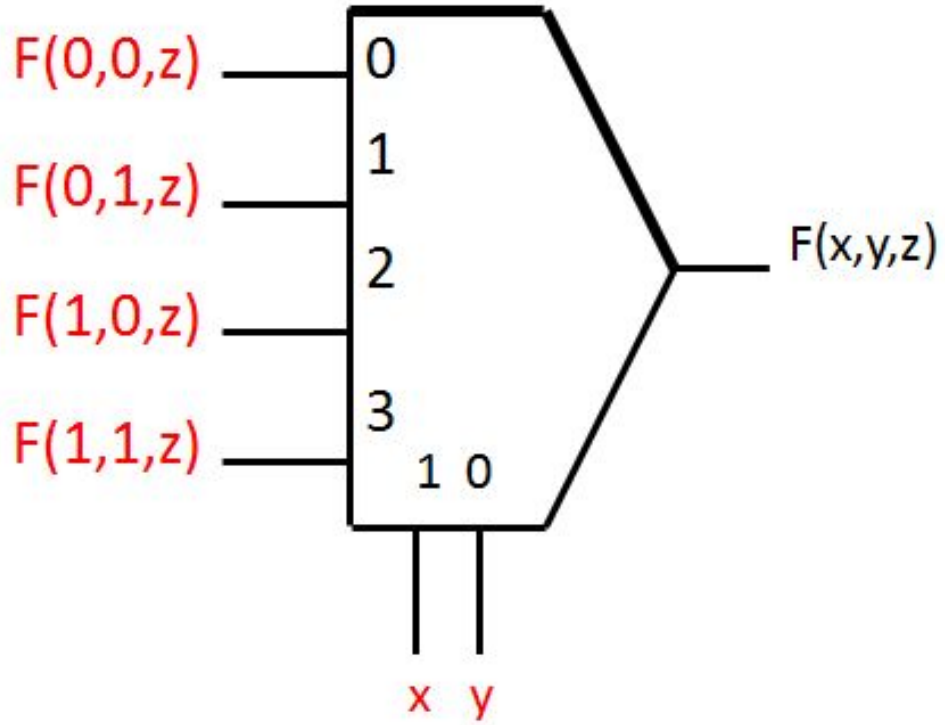
F(0,0,z) = z

F(0,1,z) = z

F(1,0,z) = z

Realización de funciones de n variables con Multiplexores (ii)

Ejemplo 2: Función de 3 variables con MUX 2 (continuación)



z \ xy	00	01	11	10
0	0	1	1	0
1	1	0	1	1

$F(0,0,z) = z$

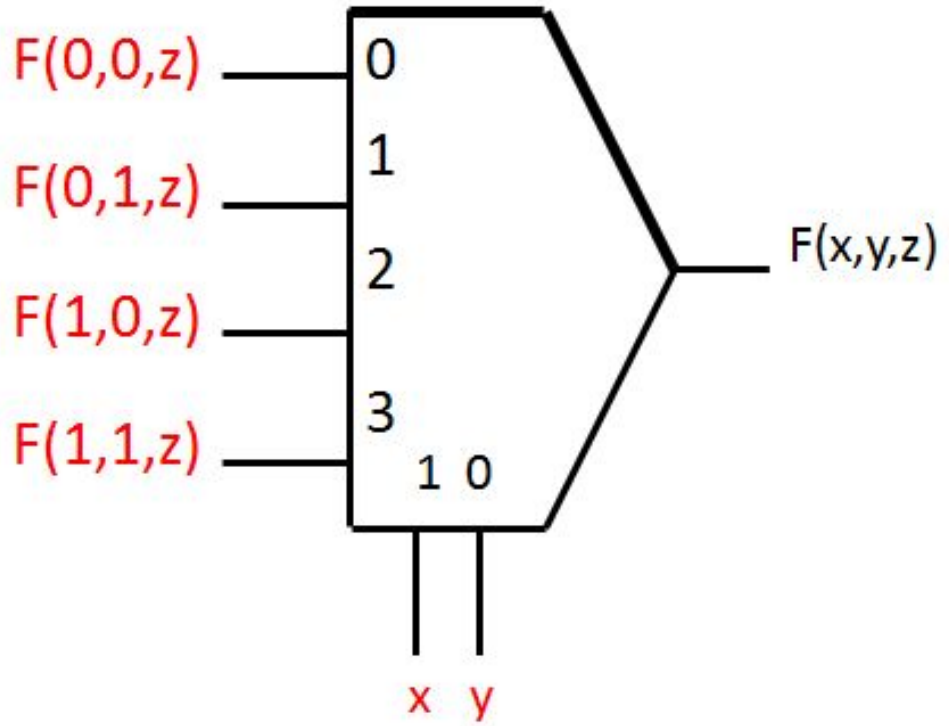
$F(0,1,z) = \overline{z}$

$F(1,0,z) = z$

$F(1,1,z) = 1$

Realización de funciones de n variables con Multiplexores (ii)

Ejemplo 2: Función de 3 variables con MUX 2 (continuación)



z \ xy	00	01	11	10
0	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**1**

$F(0,0,z) = z$

$F(0,1,z) = \overline{z}$

$F(1,0,z) = z$

$F(1,1,z) = 1$

Realización de funciones de n variables con Multiplexores (iii)

Ejemplo 3: Función de 5 variables con MUX 2

xyz / uv	000	001	011	010	110	111	101	100
00	1	1	1	1	0	1	1	1
01	1	1	1	0	0	1	1	0
11	1	1	0	1	0	1	0	1
10	1	1	0	0	0	1	0	0

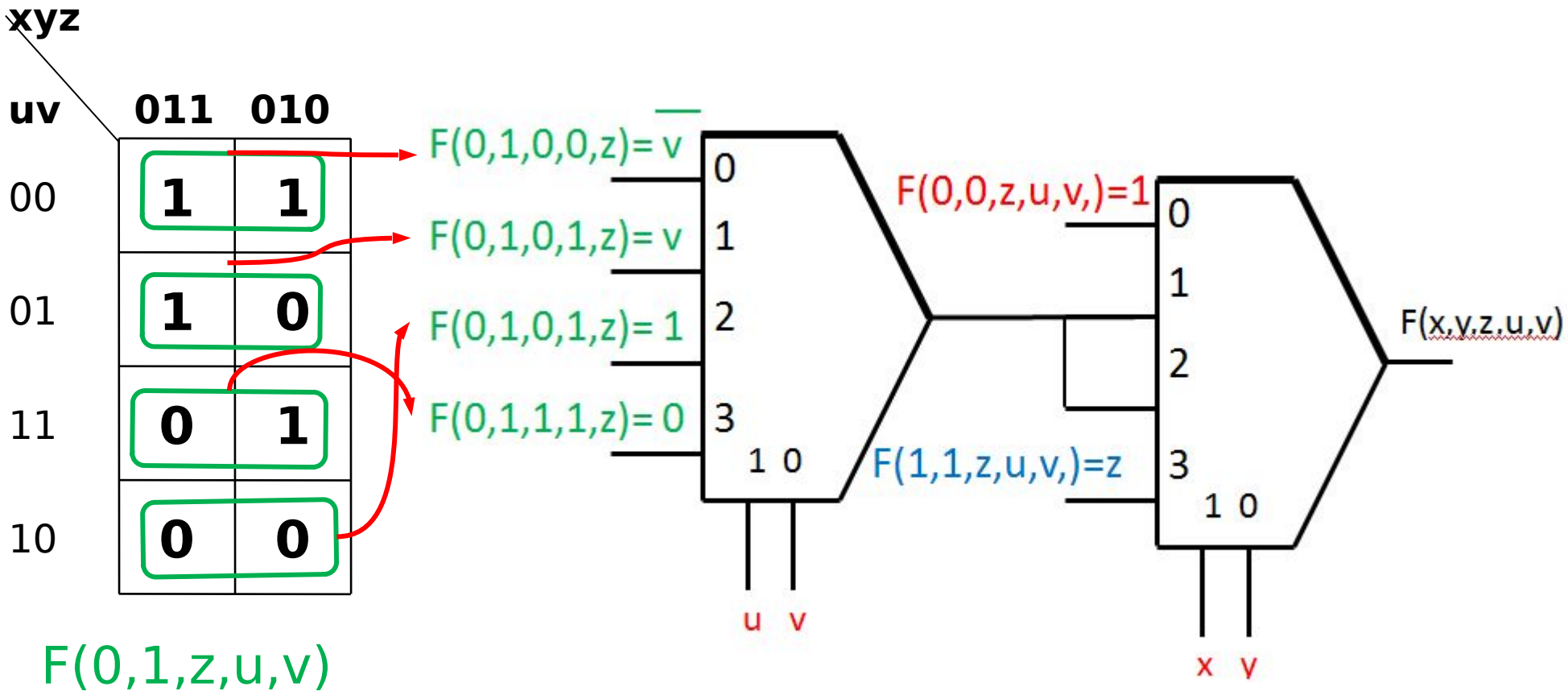
F(0,0,z,u,v) = 1

F(0,1,z,u,v) = F(1,0,z,u,v)

F(1,1,z,u,v) = z

Realización de funciones de n variables con Multiplexores (iii)

Ejemplo 3: Función de 5 variables con MUX 2



Multiplexor en Verilog

Procedimental

```
module mux4_1(
        input [1:0] sel,        // entradas de selecciÃ³n
        input [3:0] in,         // entradas de datos
        output out              // salida
        );

        reg out;

        always @(sel, in)
                /* La forma mÃ¡s directa de describir un multiplexor es mediante
                 * un bloque 'case'. */
                case (sel)
                  2'h0: out = in[0];
                  2'h1: out = in[1];
                  2'h2: out = in[2];
                  default: out = in[3];
                endcase
endmodule   // mux4_1
```

Funcional

```
module mux4_1(
        input [1:0] sel,        // entradas de selecciÃ³n
        input [3:0] in,         // entradas de datos
        output out              // salida
        );

assign out=  ~ sel[1] &  ~ sel[0] & in[0] |~ sel[1] &  sel[0] & in[1] |sel[1] &
~ sel[0] & in[2] | sel[1] &  sel[0] & in[3];
endmodule   // mux4_1
```