



Tema 4:

# Módulos combinacionales básicos

Fundamentos de computadores

**José Manuel Mendías Cuadros**

*Dpto. Arquitectura de Computadores y Automática  
Universidad Complutense de Madrid*



# Contenidos



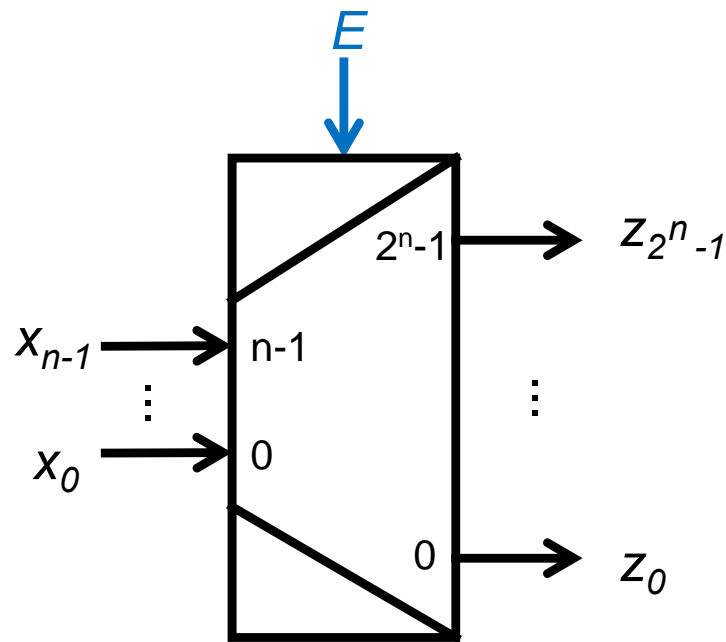
- ✓ Decodificador.
- ✓ Multiplexor.
- ✓ Bus.
- ✓ Codificador.
- ✓ ROM (Read Ony Memory).
- ✓ Sumador/Restador.
- ✓ Comparador.
- ✓ ALU (Arithmetic Logic Unit).

Transparencias basadas en los libros:

- R. Hermida, F. Sánchez y E. del Corral. *Fundamentos de computadores.*
- D. Gajsky. *Principios de diseño digital.*



# Decodificador



Decodificador n a  $2^n$

$\underline{x}$  n entradas de datos

$\underline{z}$   $2^n$  salidas de datos

$E$  1 entrada de capacitación (op)

*si la entrada toma la configuración binaria  $p$ , la salida  $(p)_{10}$ -ésima se activa*

$$z_i = \begin{cases} 1 & \text{si } E=1 \text{ y } (\underline{x})_{10} = i \\ 0 & \text{en caso contrario} \end{cases}$$

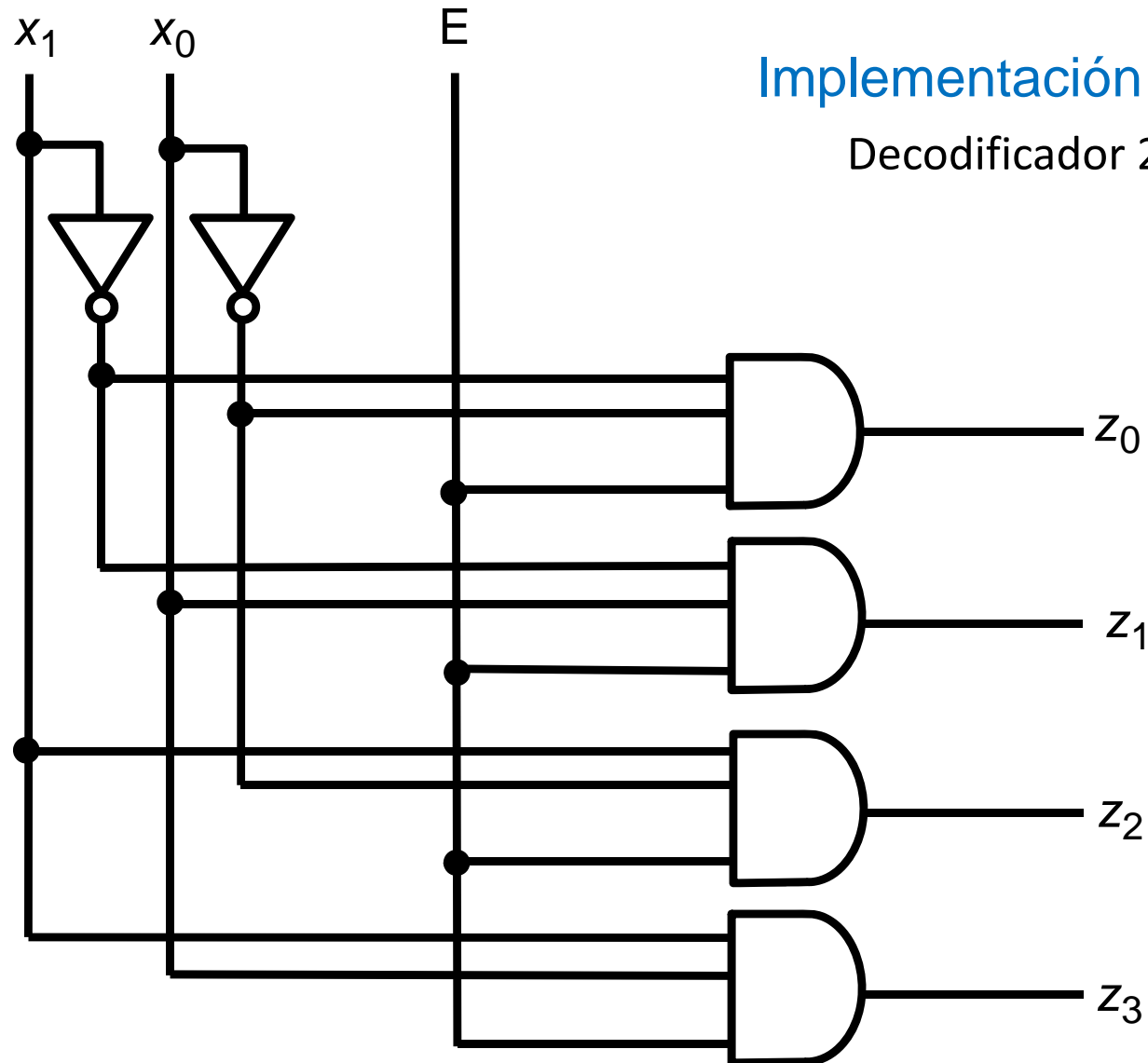
$$z_i = E \cdot m_i(\underline{x})$$



# Decodificador

Implementación directa

Decodificador 2 a 4

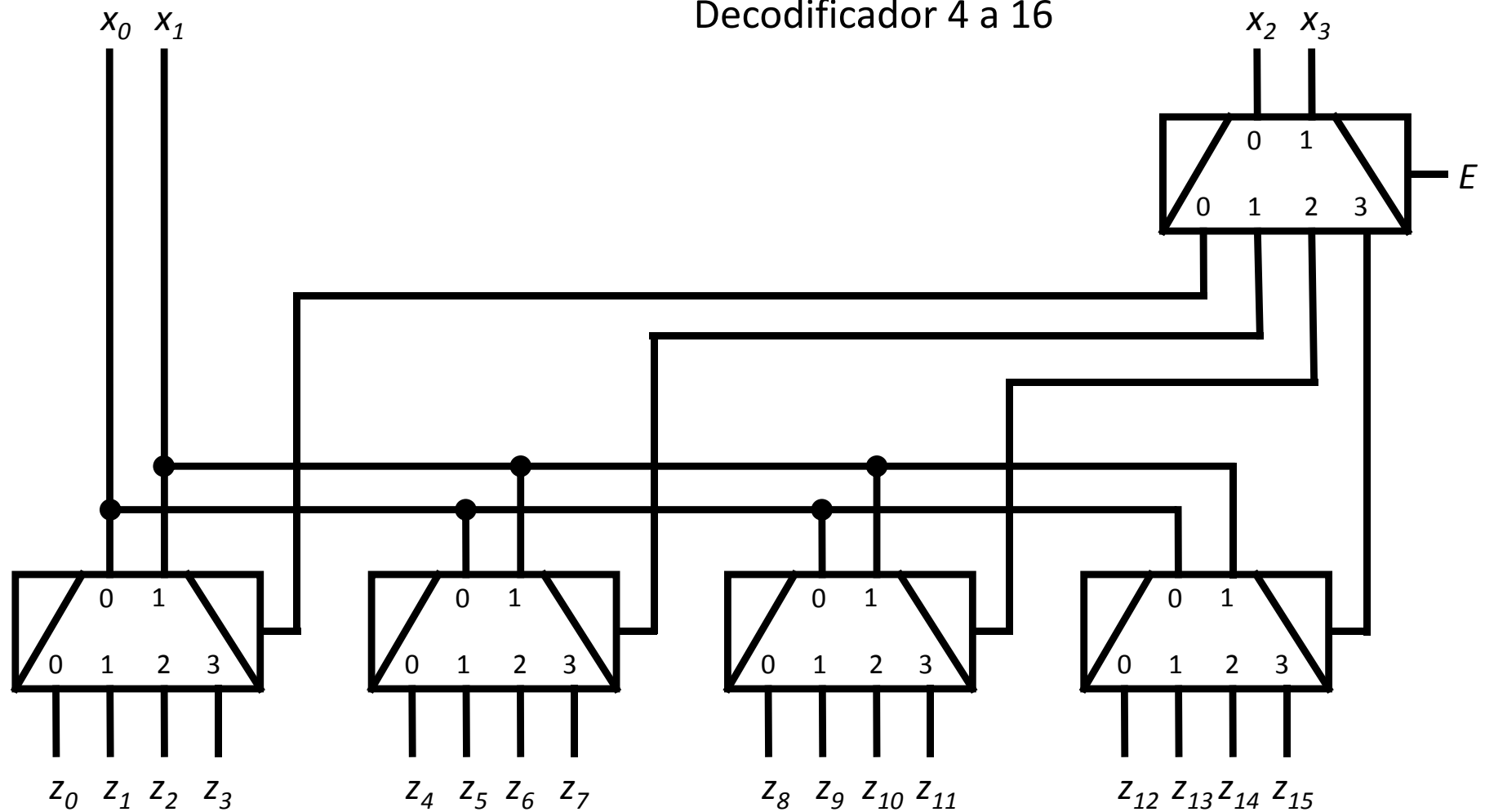




# Decodificador

## Implementación en árbol

Decodificador 4 a 16

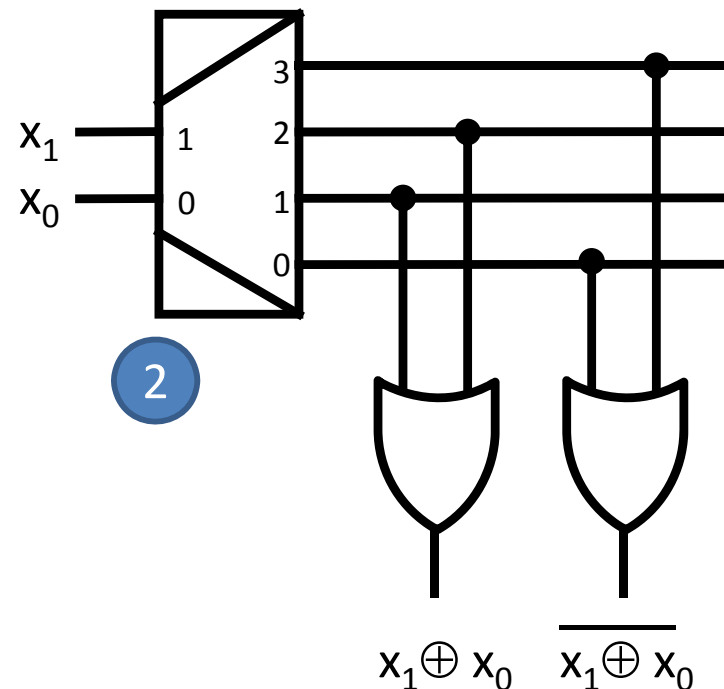




# Decodificador

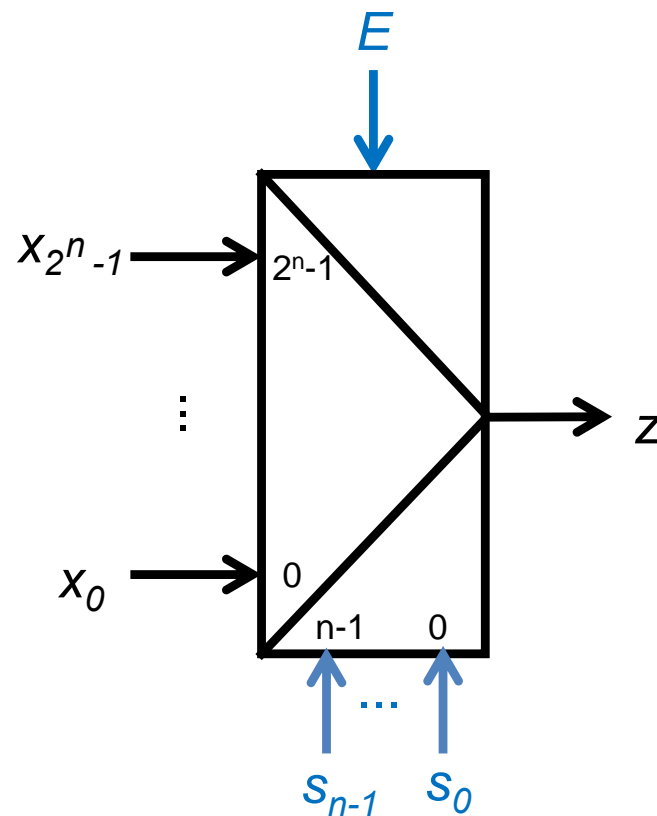
## ■ Aplicaciones al diseño:

1. Habilitar selectivamente 1 de n subcomponentes cada uno asociado a un índice (dirección) binaria.
2. Implementar directamente SPC usando puertas OR adicionales (que sumen cada unos de los mintérminos de la FC).





# Multiplexor



Multiplexor  $2^n$  a 1

$\underline{x}$   $2^n$  entradas de datos

$\underline{s}$   $n$  entradas de control

$E$  1 entrada de capacitación (op)

$z$  1 salida de datos

*si la entrada de control toma la configuración binaria  $p$ , la salida equivale a la entrada  $(p)_{10}$ -ésima*

$$z = \begin{cases} x_i & \text{si } E=1 \text{ y } (\underline{s})_{10} = i \\ 0 & \text{en caso contrario} \end{cases}$$

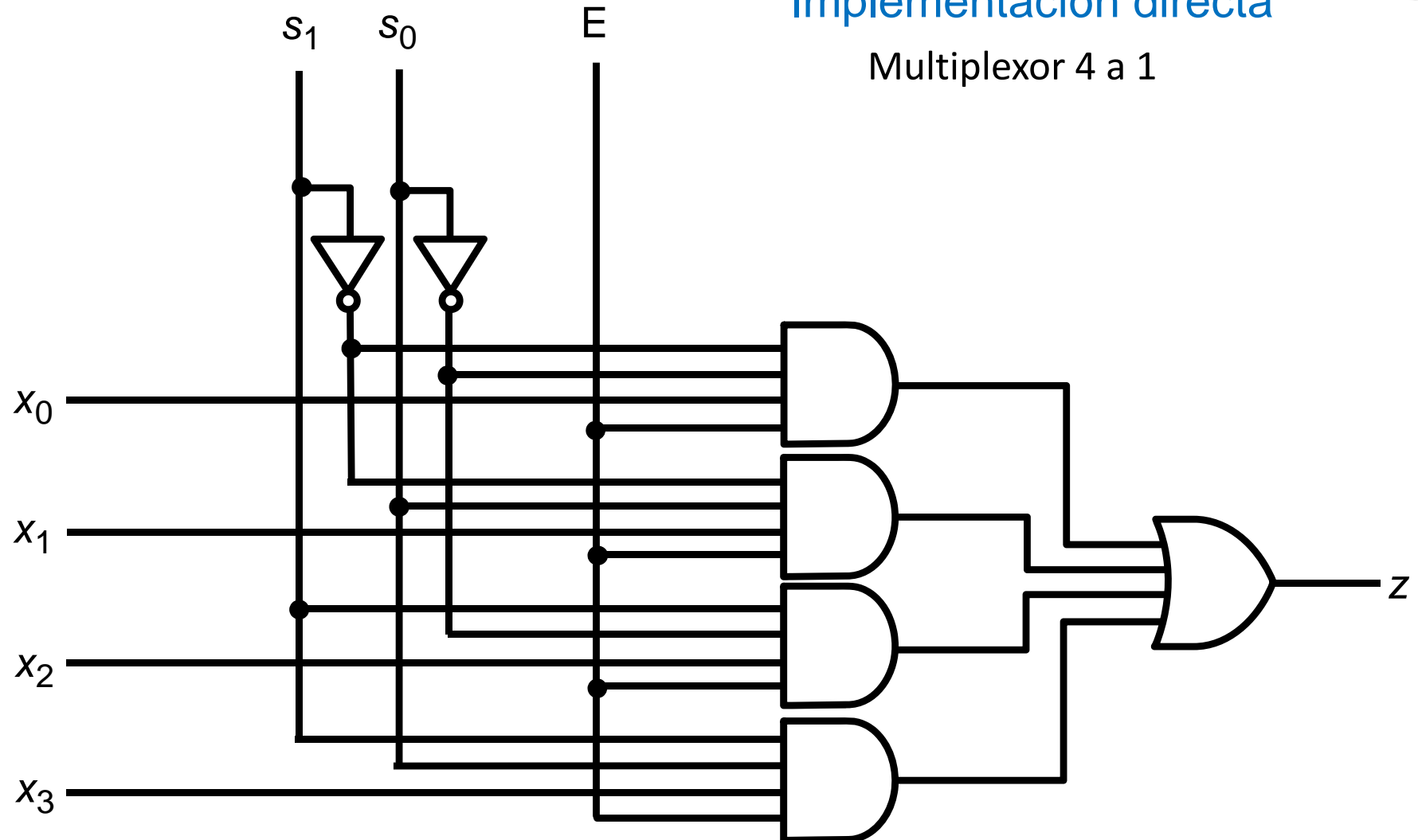
$$z = E \cdot \sum (x_i \cdot m_i(\underline{s}))$$



# Multiplexor

## Implementación directa

Multiplexor 4 a 1



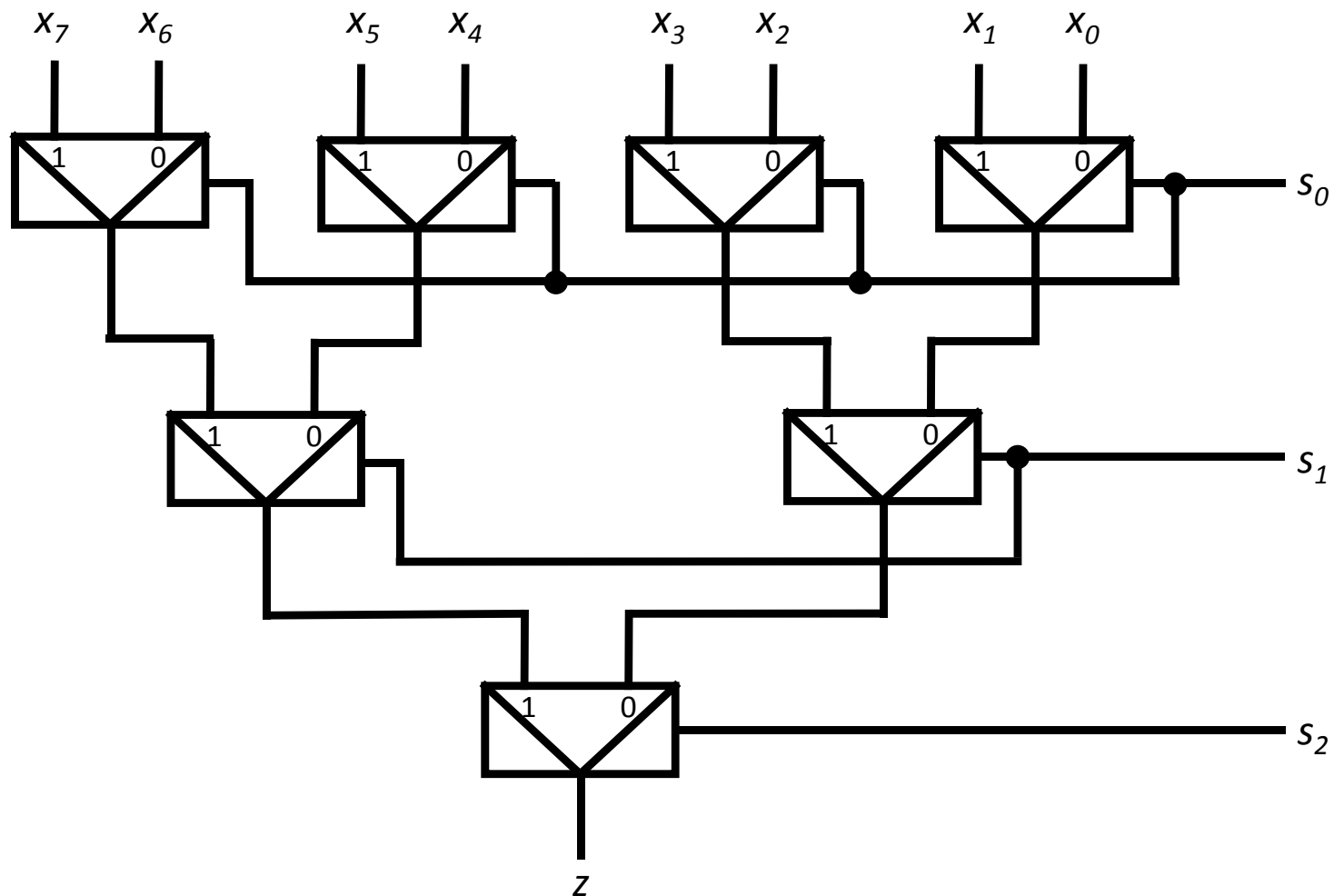




# Multiplexor

## Implementación en árbol

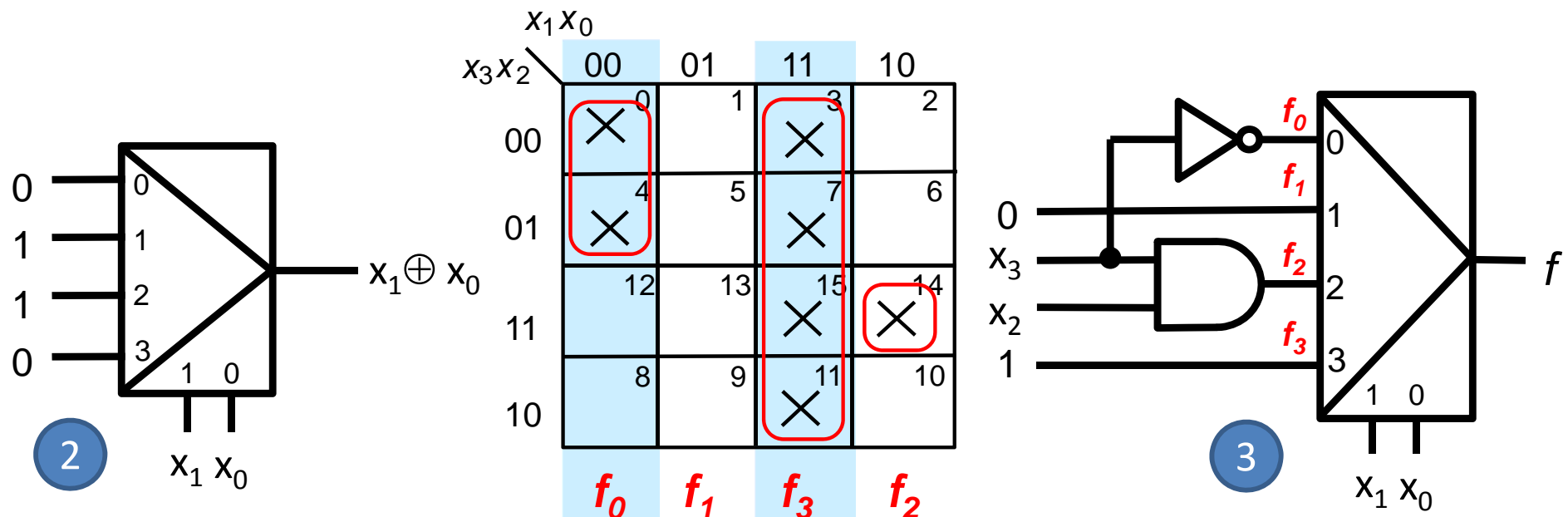
### Multiplexor 8 a 1



# Multiplexor

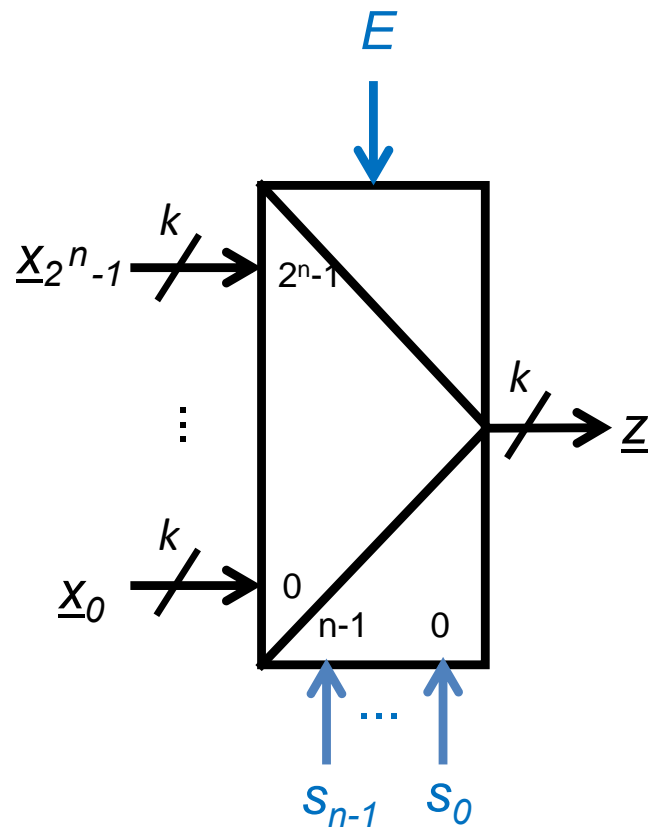
## ■ Aplicaciones al diseño:

1. Conectar selectivamente varias entradas una misma salida.
2. Implementar directamente FC que tengan el mismo número de variables que entradas de control (transcribiendo su tabla de verdad).
3. Implementar funciones de manera que las EC a simplificar tengan menos variables.





# Multiplexor vectorial



Multiplexor  $2^n$  a 1 de k bits

- $\underline{x}$   $2^n$  entradas de datos de k bits
- $\underline{s}$  n entradas de control
- E 1 entrada de capacitación (op)
- $\underline{z}$  1 salida de datos de k bits

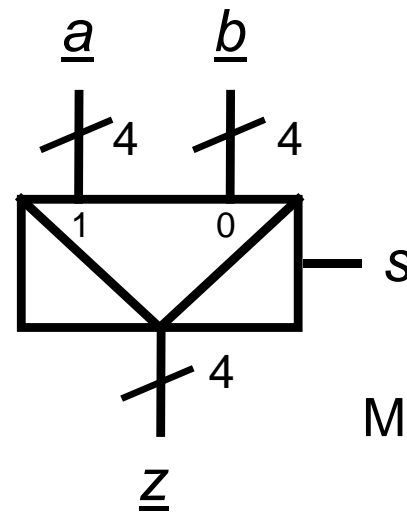
si la entrada de control toma la configuración binaria p, la salida equivale a la entrada  $(p)_{10}$ -ésima

$$\underline{z} = \begin{cases} \underline{x}_i & \text{si } E=1 \text{ y } (\underline{s})_{10} = i \\ 0 & \text{en caso contrario} \end{cases}$$

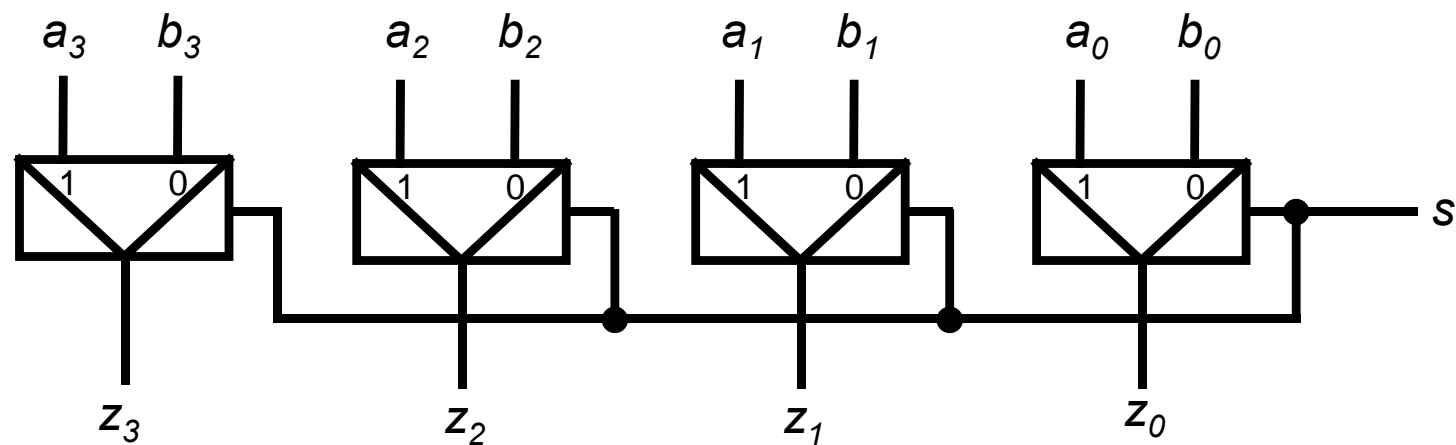
$$z_j = E \cdot \sum (x_{ij} \cdot m_i(\underline{s}))$$



# Multiplexor vectorial

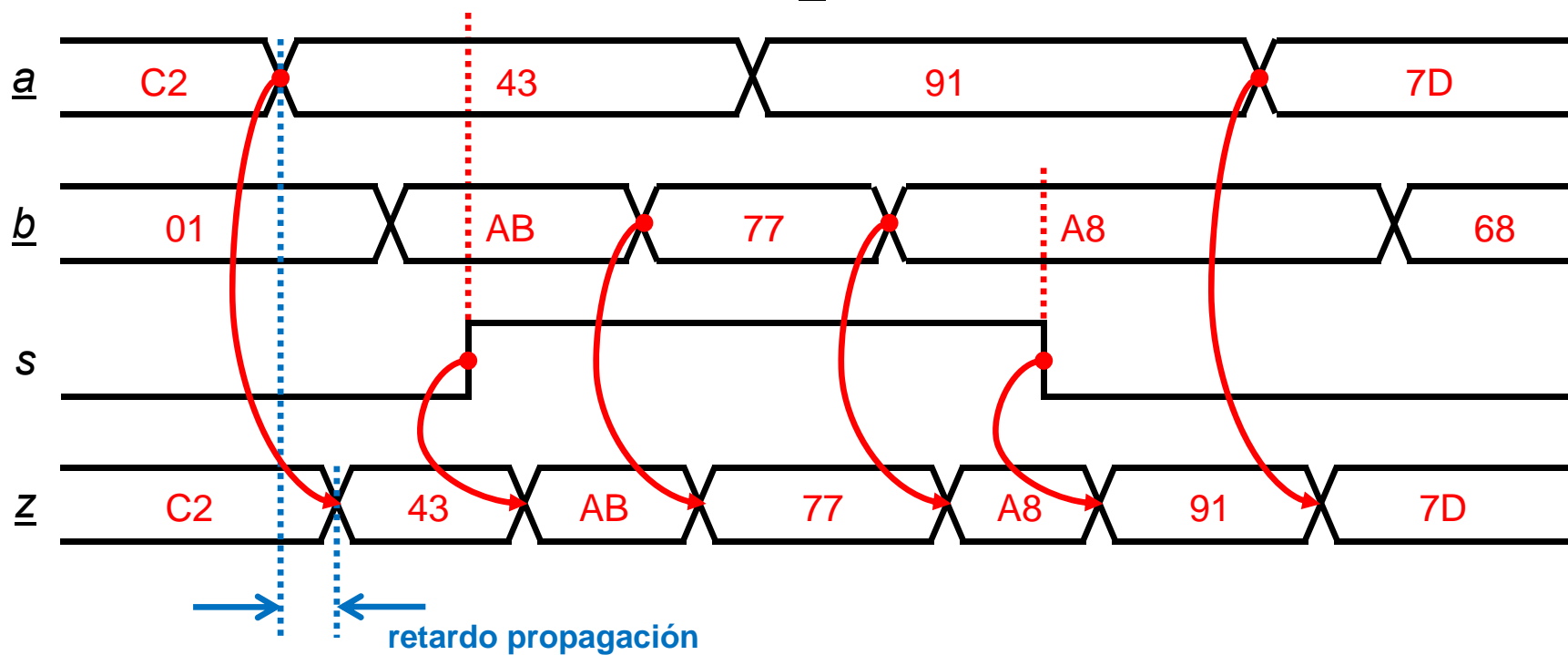
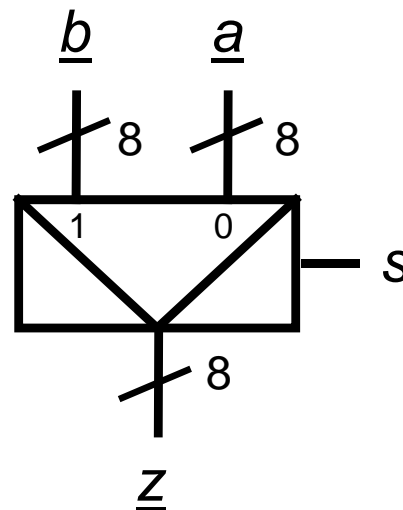


Multiplexor 2 a 1 de 4 bits

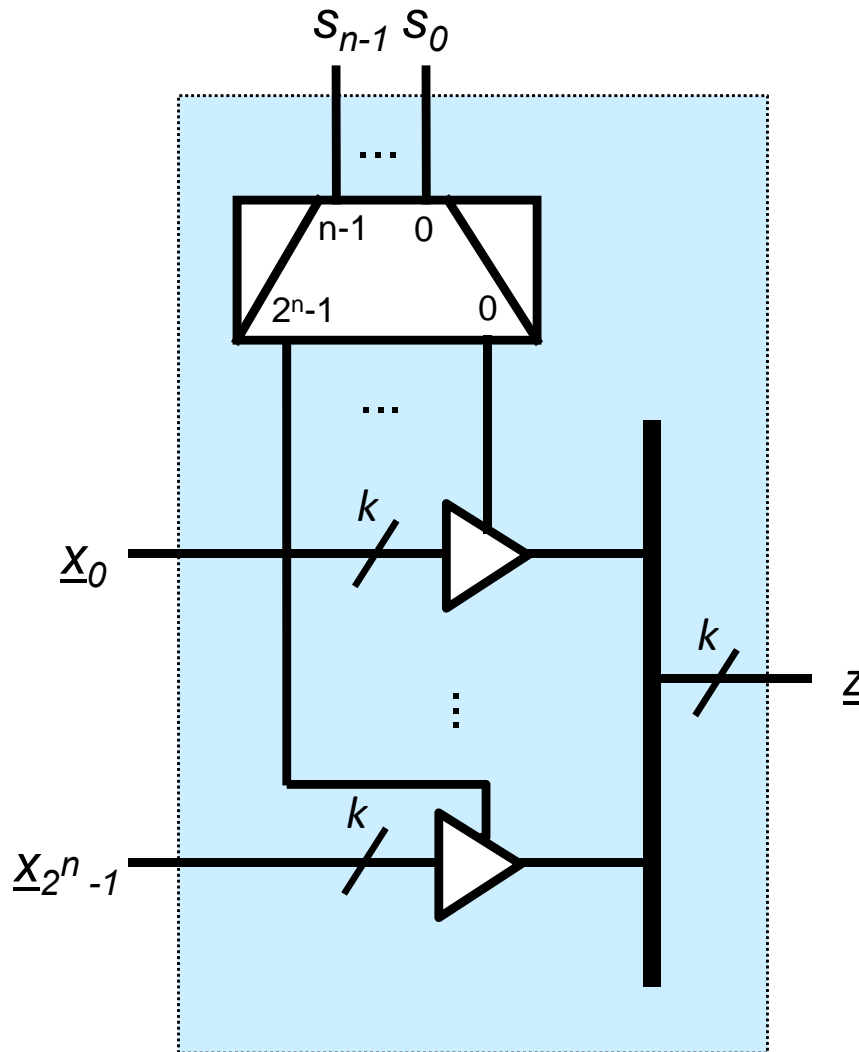




# Multiplexor vectorial



# Bus



$\underline{x}$   $2^n$  entradas de datos de  $k$  bits

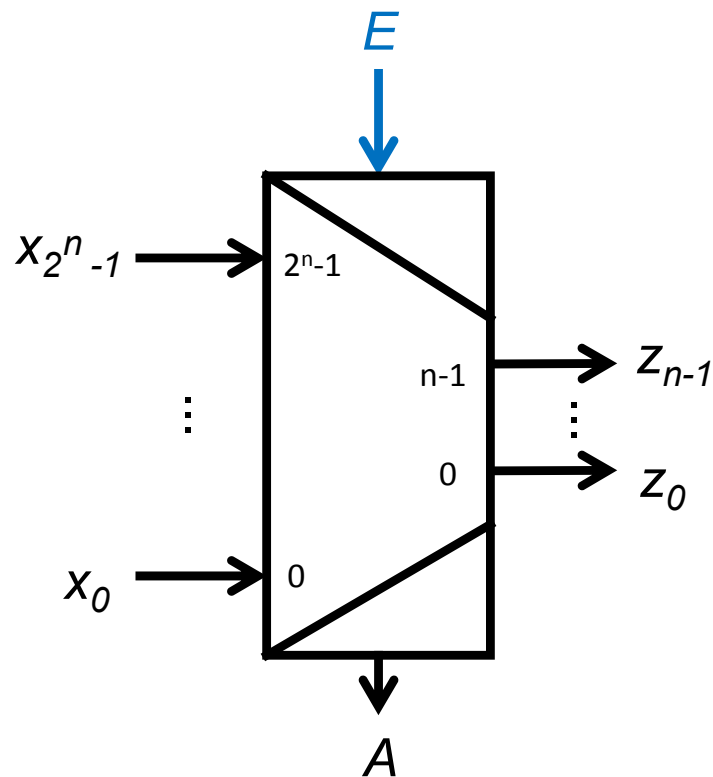
$\underline{s}$   $n$  entradas de control

$\underline{z}$  1 salida de datos de  $k$  bits

*si la entrada de control toma la configuración binaria  $p$ , la salida equivale a la entrada  $(p)_{10}$ -ésima*



# Codificador



Codificador  $2^n$  a  $n$

$\underline{x}$   $2^n$  entradas de datos

$\underline{z}$   $n$  salidas de datos

$E$  1 entrada de capacitación (op)

$A$  1 salida de actividad

si se activa la entrada  $p$ -ésima **y solo esa**, la salida codifica  $p$  en binario

$$\underline{z} = \begin{cases} (i)_2 & \text{si } E=1 \text{ y } x_i = 1 \text{ y } \forall j, j \neq i, x_j=0 \\ 0 & \text{en caso contrario} \end{cases}$$

$$A = \begin{cases} 1 & \text{si } E=1 \text{ y } \exists i, x_i=1 \\ 0 & \text{en caso contrario} \end{cases}$$

$$z_i = E \cdot \sum (x_j) \text{ con } j \in \{ (a_{n-1} \dots a_0)_2 / a_i = 1 \}$$
$$A = E \cdot \sum (x_i)$$

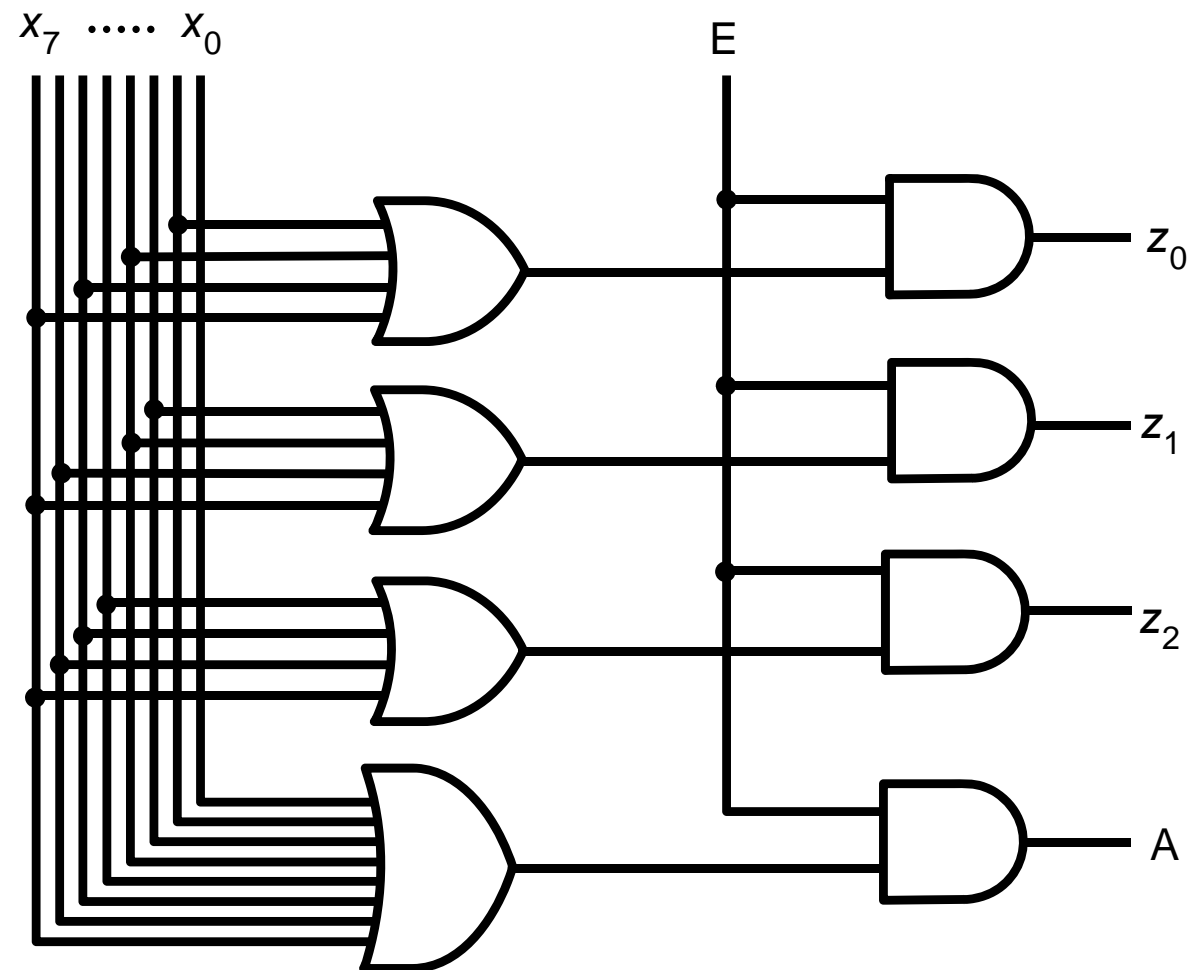


# Codificador

## Implementación directa

Codificador 8 a 3

entrada activada	$z_2$	$z_1$	$z_0$
$x_0$	0	0	0
$x_1$	0	0	1
$x_2$	0	1	0
$x_3$	0	1	1
$x_4$	1	0	0
$x_5$	1	0	1
$x_6$	1	1	0
$x_7$	1	1	1







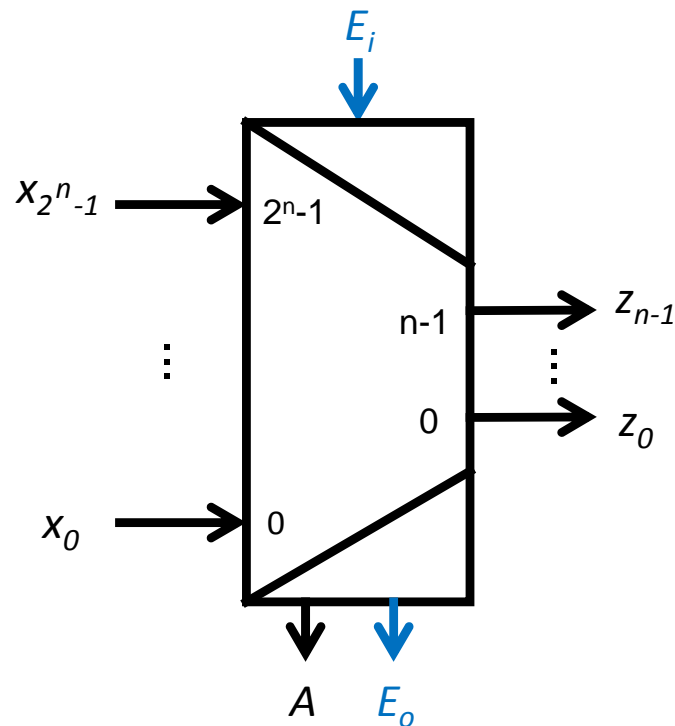
# Codificador de prioridad

versión 12/09/14

tema 4:  
Módulos combinatoriales básicos

FC

26



Codificador de  
prioridad  $2^n$  a  $n$

$\underline{x}$   $2^n$  entradas de datos

$\underline{z}$   $n$  salidas de datos

$E_i$  1 entrada de capacitación (op)

$E_o$  1 salida de capacitación (op)

$A$  1 salida de actividad

la salida codifica en binario la  
entrada activa de **más peso**

$$\underline{z} = \begin{cases} (i)_2 & \text{si } E_i=1 \text{ y } x_i = 1 \text{ y } \forall j, j>i, x_j=0 \\ 0 & \text{en caso contrario} \end{cases}$$

$$A = \begin{cases} 1 & \text{si } E_i=1 \text{ y } \exists i, x_i=1 \\ 0 & \text{en caso contrario} \end{cases}$$

$$E_o = \begin{cases} 1 & \text{si } E_i=1 \text{ y } \forall j, x_j = 0 \\ 0 & \text{en caso contrario} \end{cases}$$



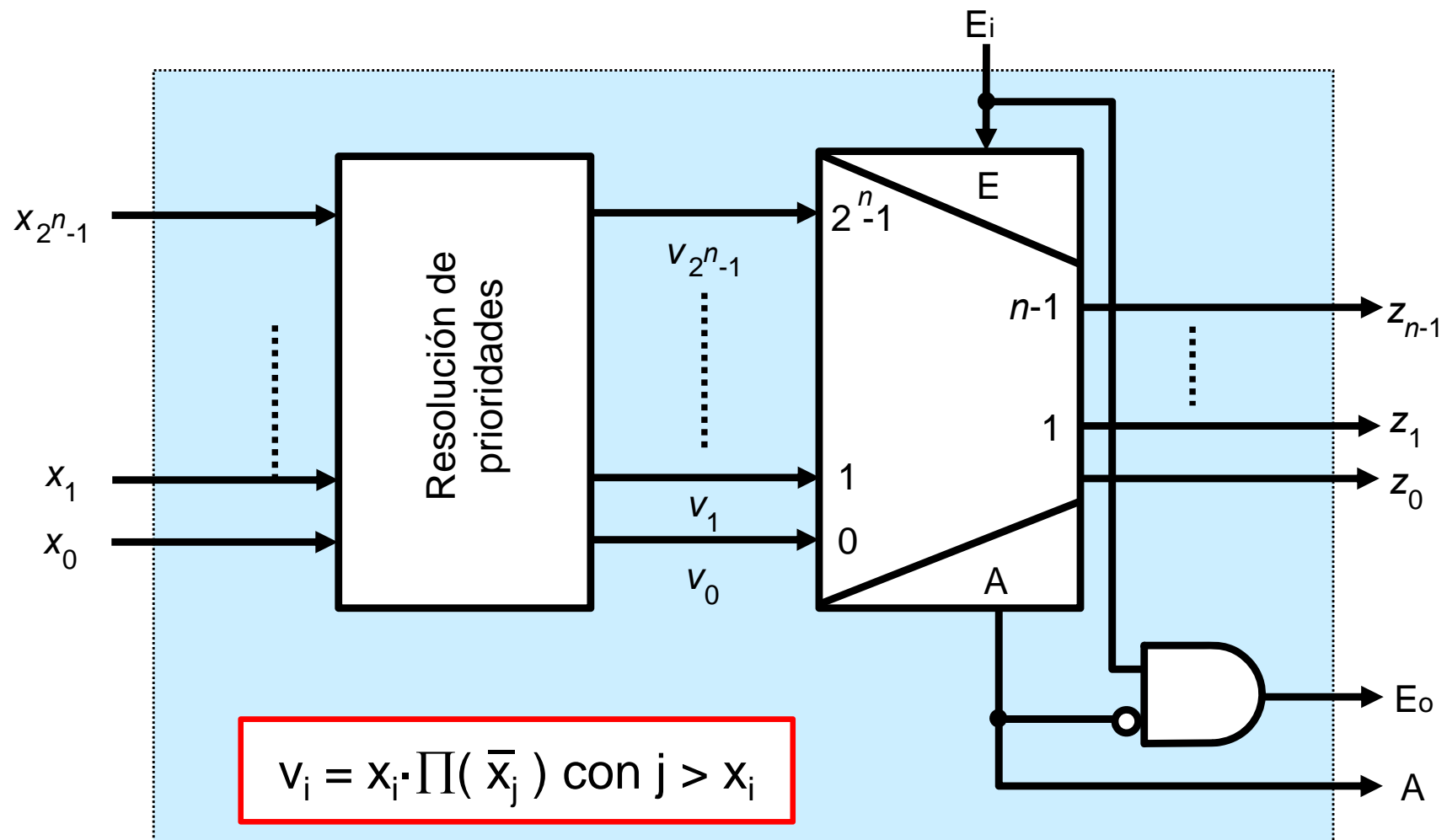
# Codificador de prioridad

versión 12/09/14

tema 4:  
Módulos combinacionales básicos

FC

27



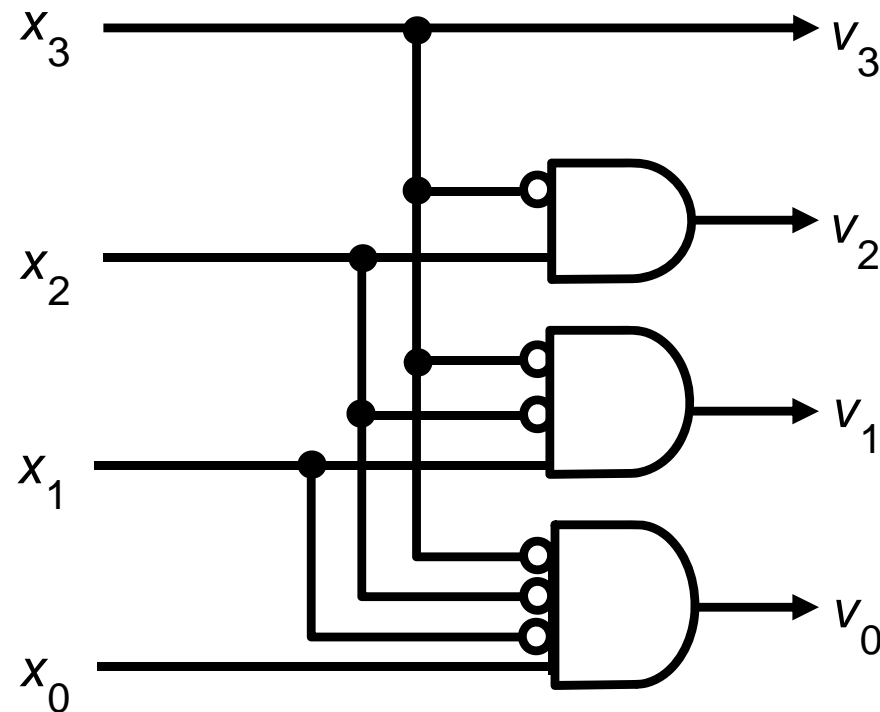


# Codificador de prioridad

## Implementación directa

Resolución de prioridades

Codificador 4 a 2





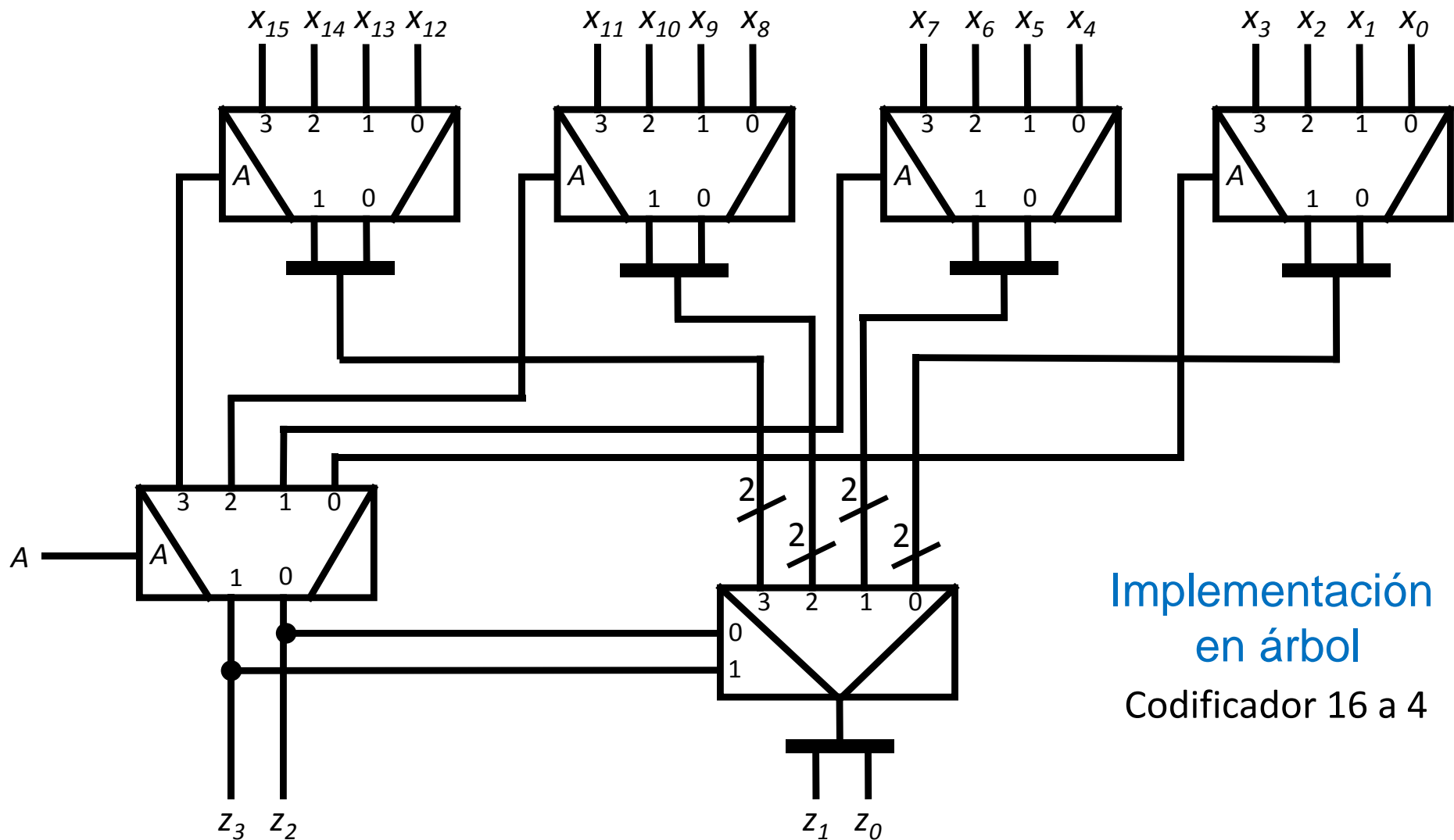
# Codificador de prioridad

versión 12/09/14

tema 4:  
Módulos combinatoriales básicos

FC

32

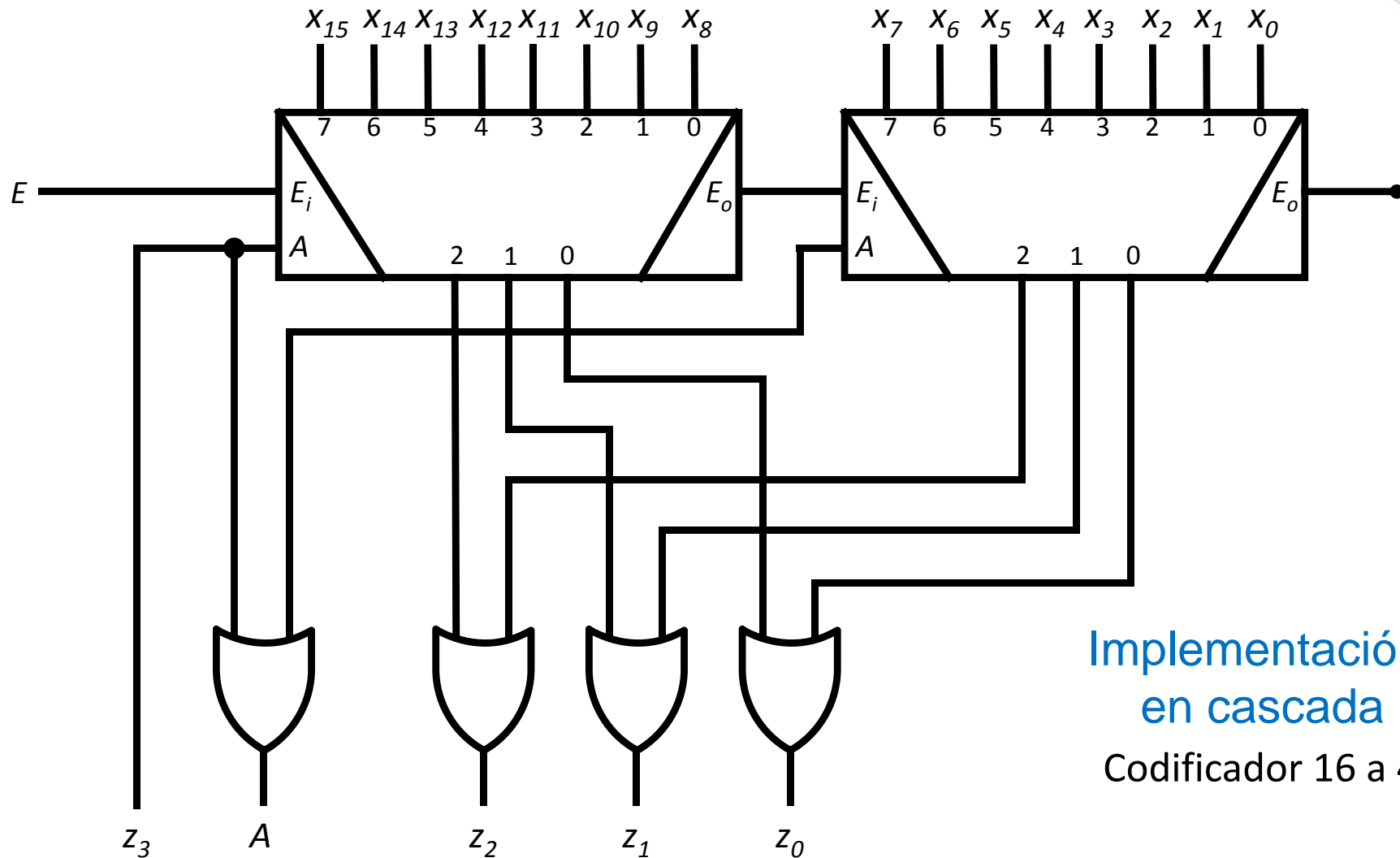


Implementación  
en árbol

Codificador 16 a 4



# Codificador de prioridad



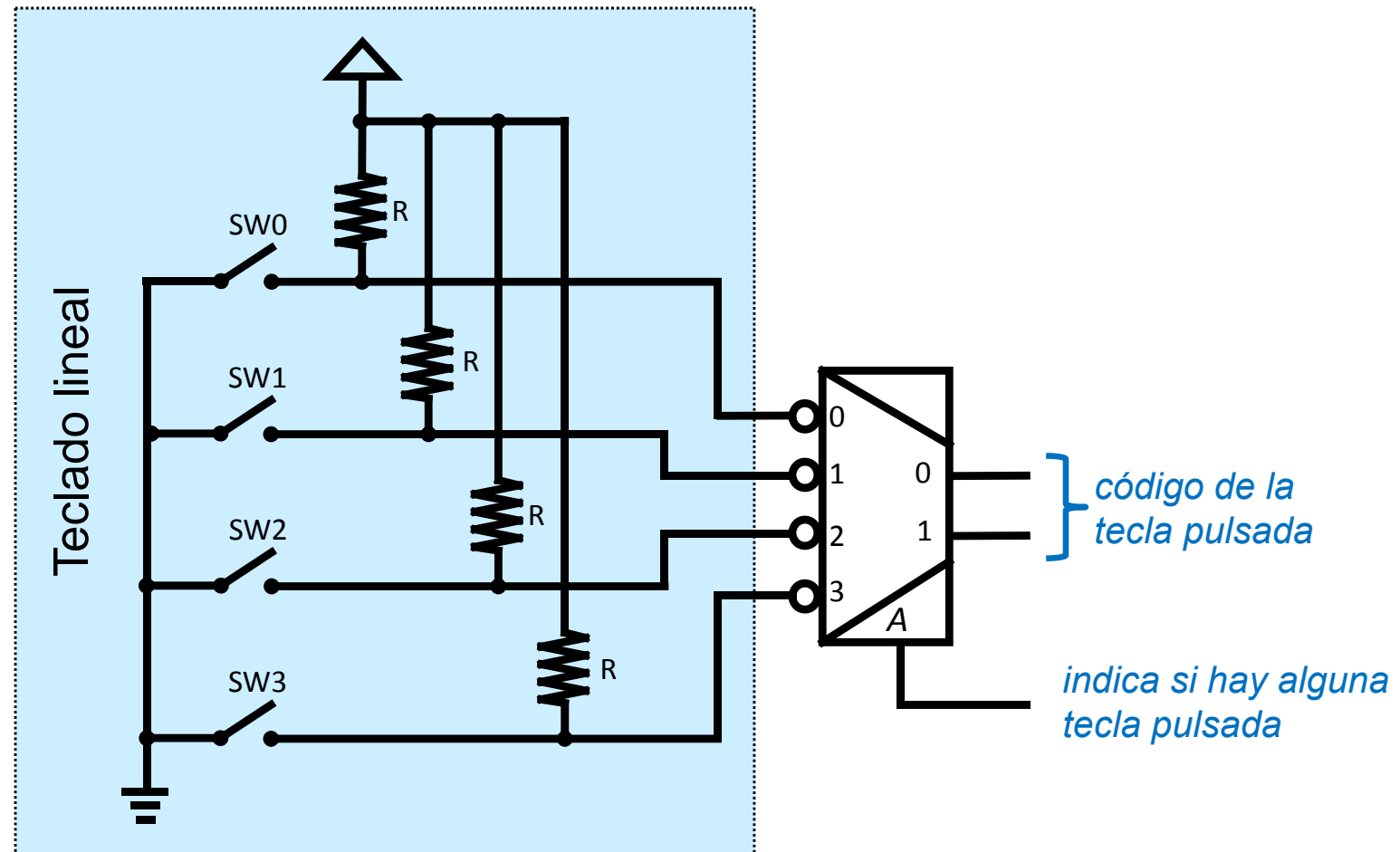
Implementación  
en cascada

Codificador 16 a 4

# Codificador de prioridad

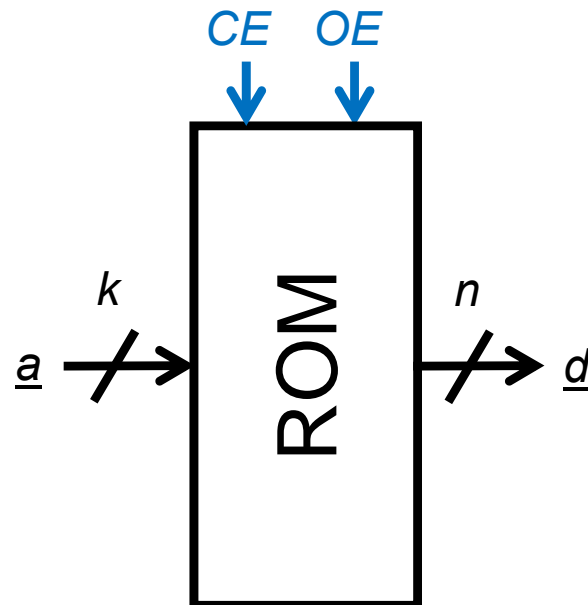
## ■ Aplicaciones al diseño:

1. Asociar un código a cada componente de un vector de entrada.





# ROM (Read Only Memory)



ROM  $2^k \times n$   
( $2^k$  palabras de  $n$  bits)

a 1 entrada de dirección de  $k$  bits

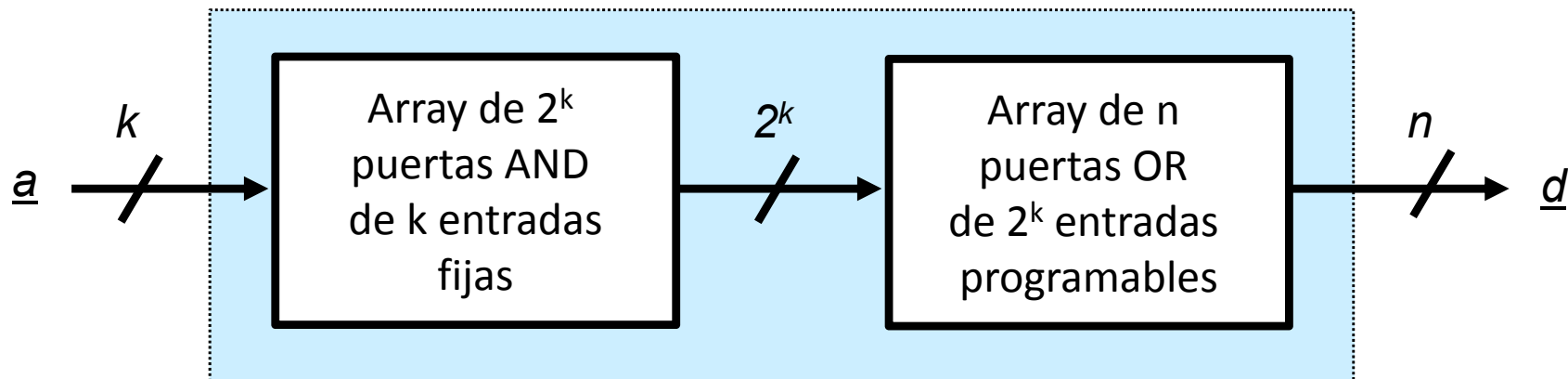
d 1 salida de datos de  $n$  bits

CE 1 entrada de capacitación (op)

OE 1 entrada de capacitación de lectura (op)

*dispositivo programable capaz de  
implementar  $n$  FC de  $k$  variables  
almacenando sus tablas de verdad*

*memoria no volátil de capaz de  
almacenar  $2^k$  palabras de  $n$  bits cada una*





# ROM (Read Only Memory)

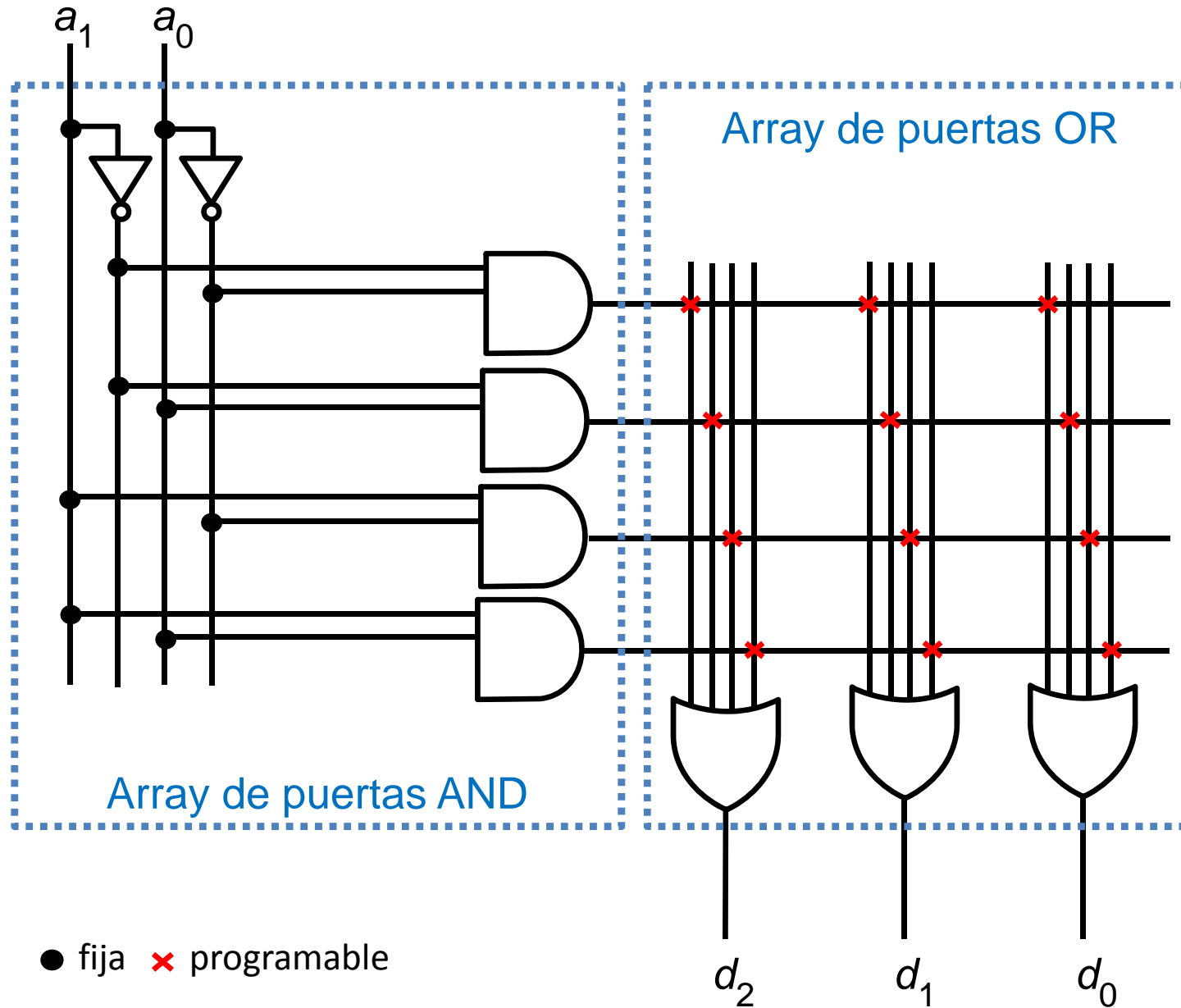
versión 12/09/14

tema 4:  
Módulos combinatoriales básicos

FC

40

ROM 4x3







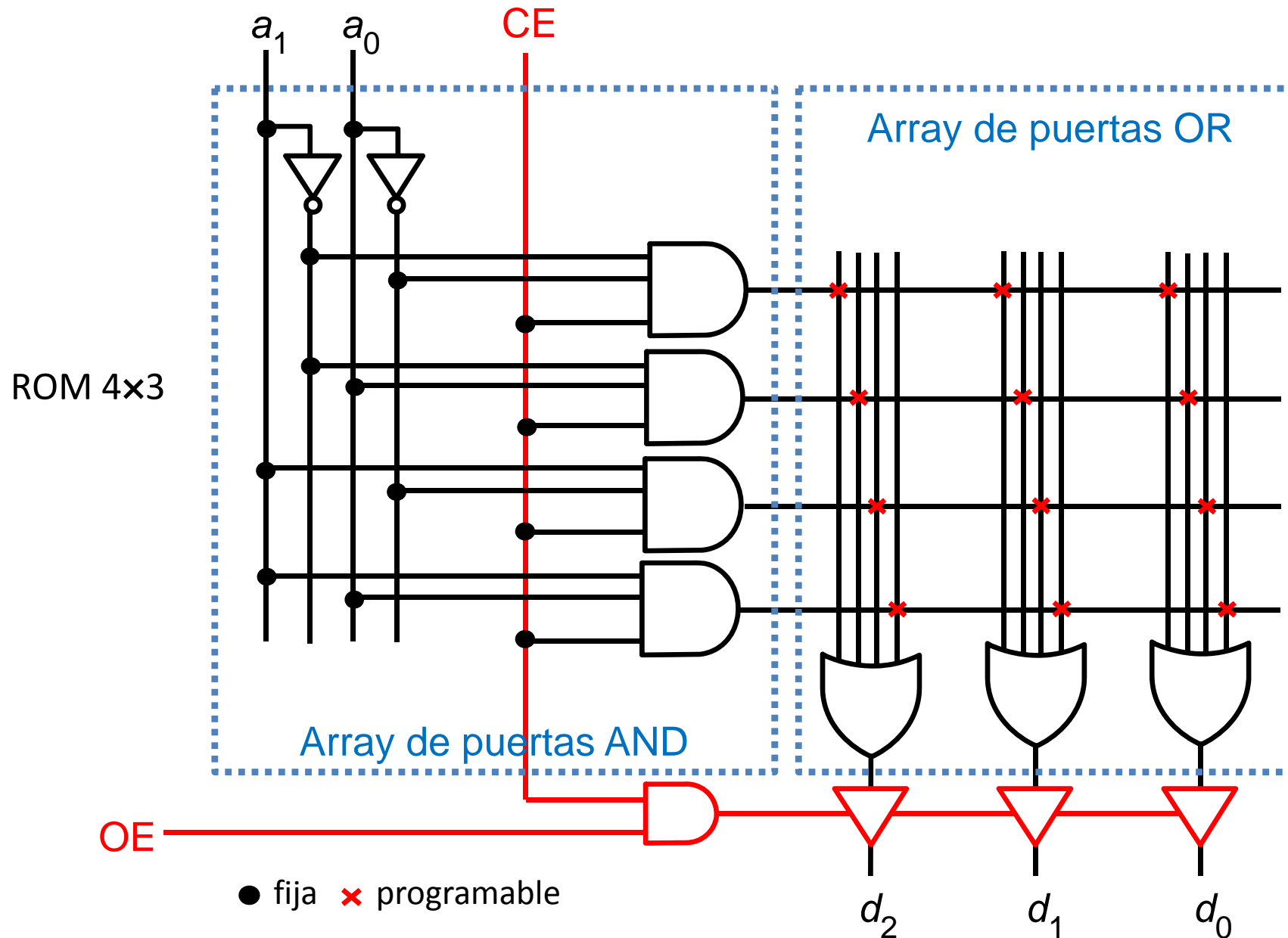
# ROM (Read Only Memory)

versión 12/09/14

tema 4:  
Módulos combinatoriales básicos

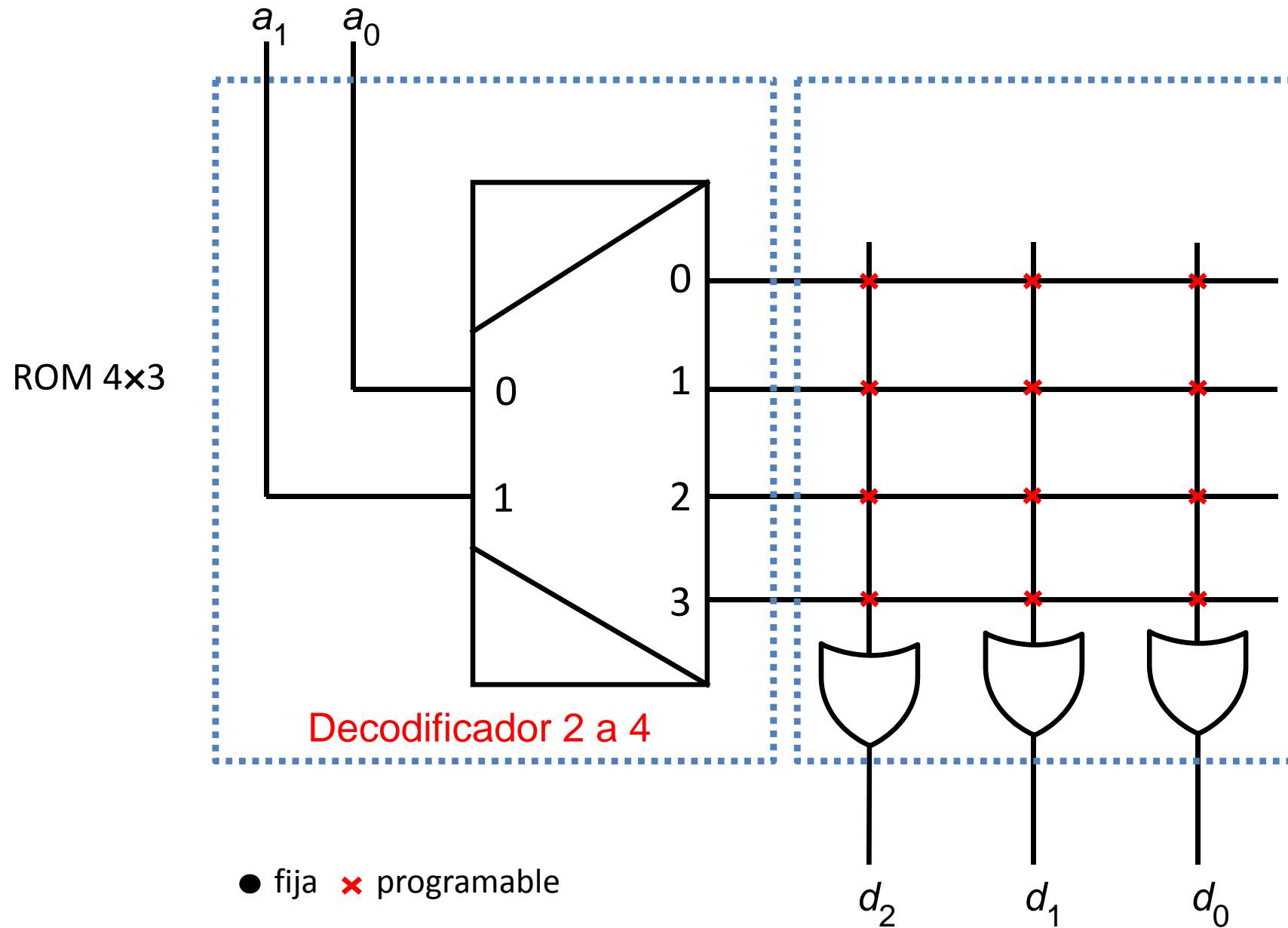
FC

41





# ROM (Read Only Memory)





# ROM (Read Only Memory)

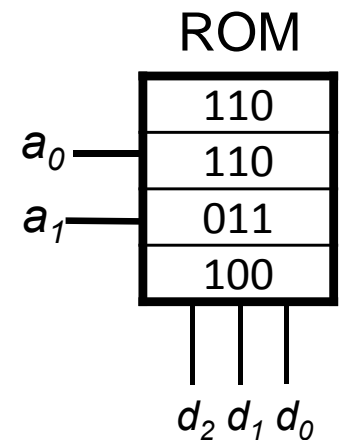
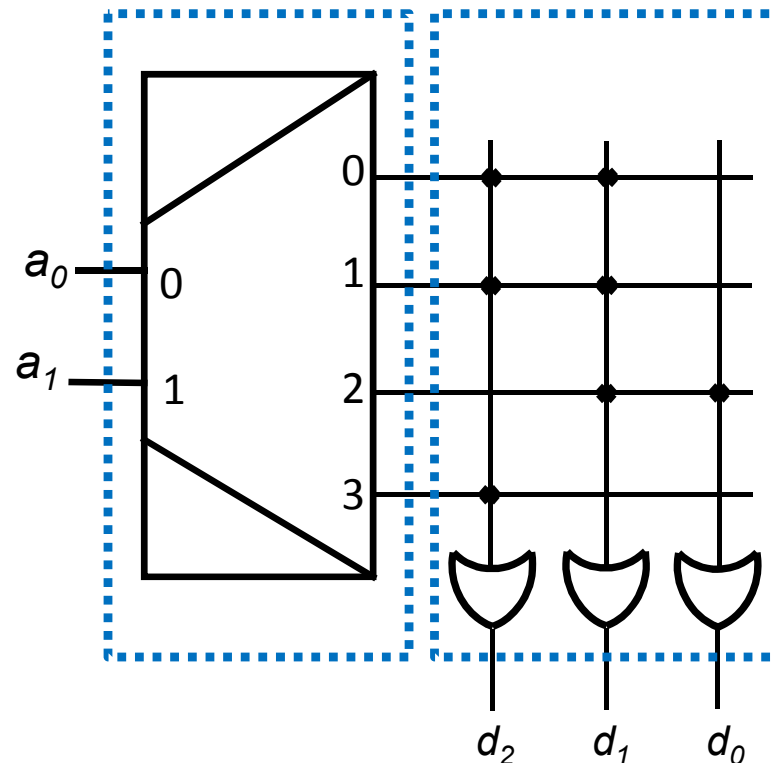
- Aplicaciones al diseño:
  - Implementar directamente FC almacenando su tabla de verdad.

	$a_1$	$a_0$	$d_2$	$d_1$	$d_0$
0	0	0	1	1	0
1	0	1	1	1	0
2	1	0	0	1	1
3	1	1	1	0	0

$$d_2 = \bar{a}_1 + \bar{a}_0$$

$$d_1 = a_1 + a_0$$

$$d_0 = a_1 \cdot \bar{a}_0$$



# ROM (Read Only Memory)



- **Mask Programmable ROM**
  - Se programa durante la fabricación del chip.
  - No puede borrarse/reprogramarse.
- **PROM (Programmable ROM)**
  - Se programa eléctricamente usando un programador.
  - No puede borrarse/reprogramarse.
- **EPROM (Erasable Programmable ROM)**
  - Se programa eléctricamente usando un programador.
  - Se borra (chip completo) exponiéndola a luz ultravioleta.
- **EEPROM (Electrically Erasable Programmable ROM)**
  - Se programa/borra (palabra) eléctricamente usando un programador.
- **Flash memory**
  - Se programa/borra (bloque) eléctricamente sin requerir programador.

# ROM (Read Only Memory)

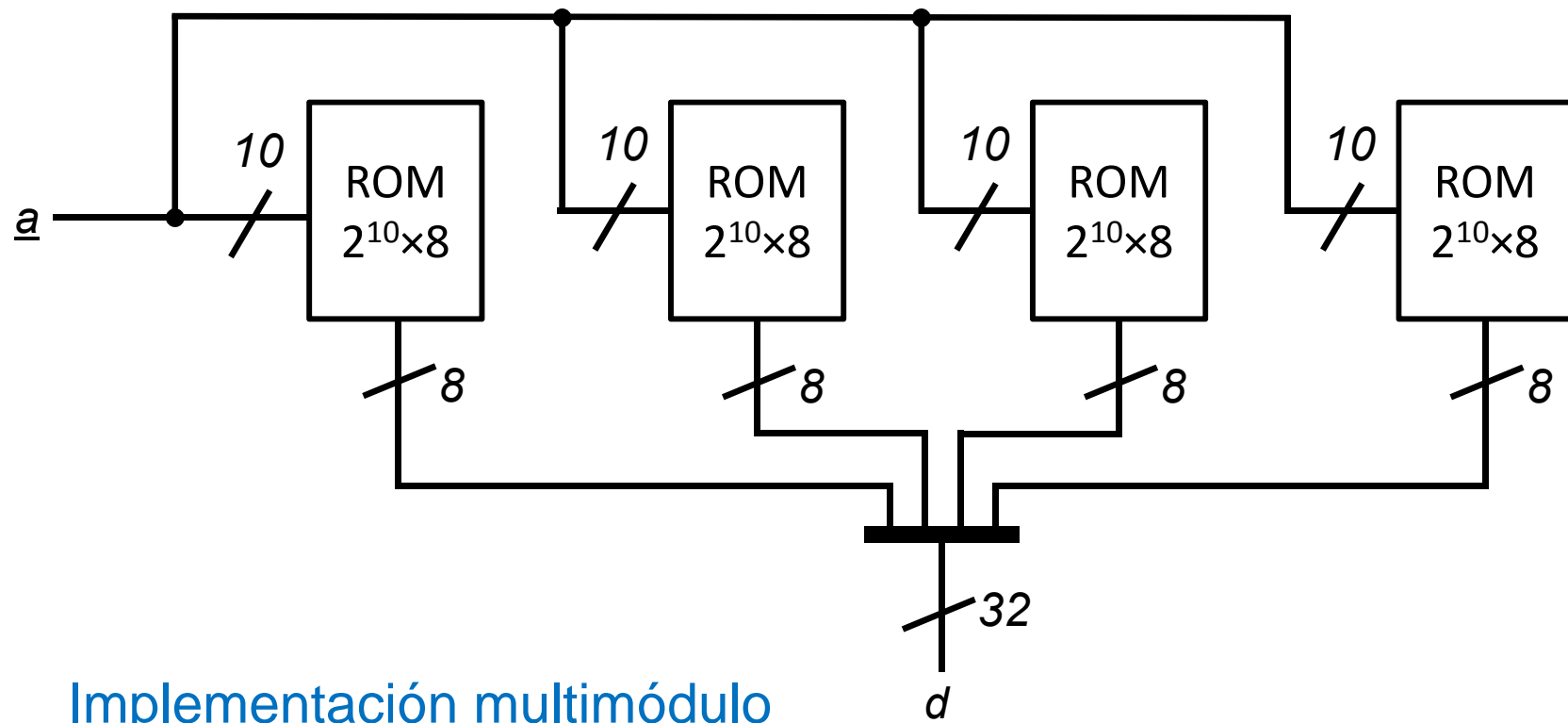


- La **capacidad** de las memorias se mide en bytes (8 bits)
  - Cuando el número de bytes es alto, se utilizan prefijos.
- Históricamente, los prefijos indican cantidades potencias de 2
  - Kilobyte (KB)            =  $2^{10}$  bytes    = 1.024 bytes
  - Megabyte (MB)        =  $2^{20}$  bytes    = 1.048.576 bytes
  - Gigabyte (GB)        =  $2^{30}$  bytes    = 1.073.741.824 bytes
- Sin embargo, desde hace algunos años su significado se ha homogeneizado con el definido en el Sistema Internacional de unidades (**potencias de 10**)
  - Kilobyte (kB)            =  $10^3$  bytes    = 1.000 bytes
  - Megabyte (MB)        =  $10^6$  bytes    = 1.000.000 bytes
  - Gigabyte (GB)        =  $10^9$  bytes    = 1.000.000.000 bytes
  - Y se han definido **nuevos prefijos** para indicar las **potencias de 2**
    - Kibibyte (KiB)        =  $2^{10}$  bytes    = 1.024 bytes
    - Mebibyte (MiB)       =  $2^{20}$  bytes    = 1.048.576 bytes
    - Gibibyte (GiB)       =  $2^{30}$  bytes    = 1.073.741.824 bytes
  - No obstante, todavía no está generalizado el uso de los nuevos prefijos .



# ROM (Read Only Memory)

- Varias ROM se pueden componer para comportarse como una ROM de **mayor anchura de palabra**.



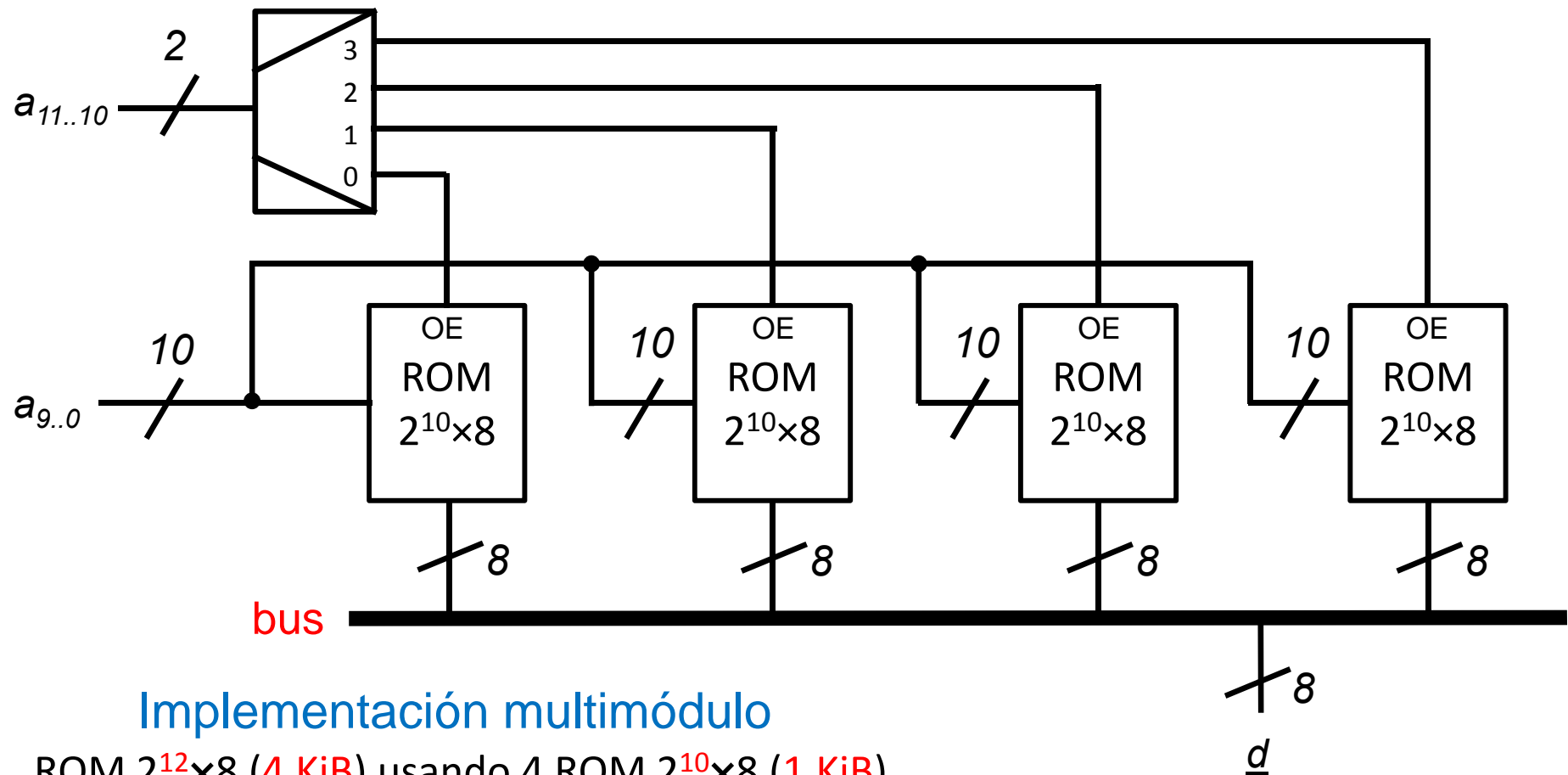
Implementación multimódulo

ROM  $2^{10} \times 32$  (4 KiB) usando 4 ROM  $2^{10} \times 8$  (1 KiB)



# ROM (Read Only Memory)

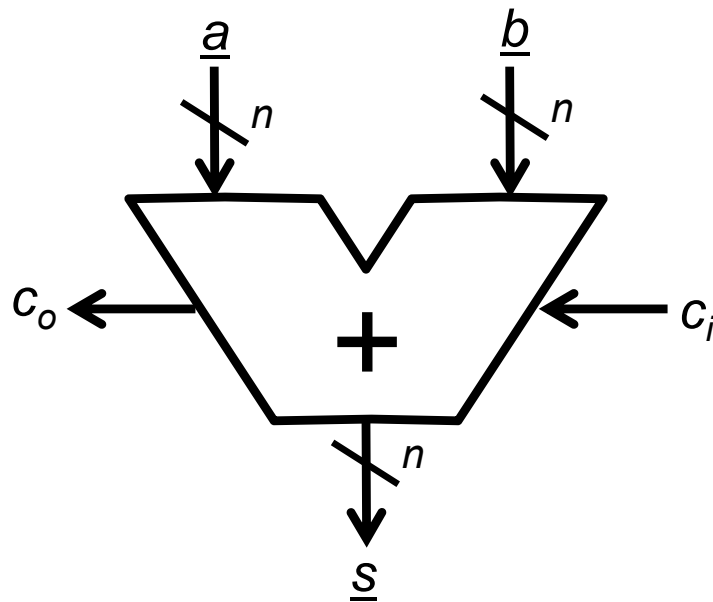
- Varias ROM se pueden componer para comportarse como una ROM de **mayor profundidad**.



## Implementación multimódulo

ROM  $2^{12} \times 8$  (4 KiB) usando 4 ROM  $2^{10} \times 8$  (1 KiB)

# Sumador



$\underline{a}, \underline{b}$  2 entradas de datos de  $n$  bits

$c_i$  1 entrada de acarreo

$\underline{s}$  1 salida de datos de  $n$  bits

$c_o$  1 salida de acarreo

realiza la suma binaria de  $\underline{a} + \underline{b} + c_i$

$$\underline{s} = (\underline{a} + \underline{b} + c_i) \bmod 2^n$$

$$c_o = \begin{cases} 1 & (\underline{a} + \underline{b} + c_i) \geq 2^n \\ 0 & \text{en caso contrario} \end{cases}$$





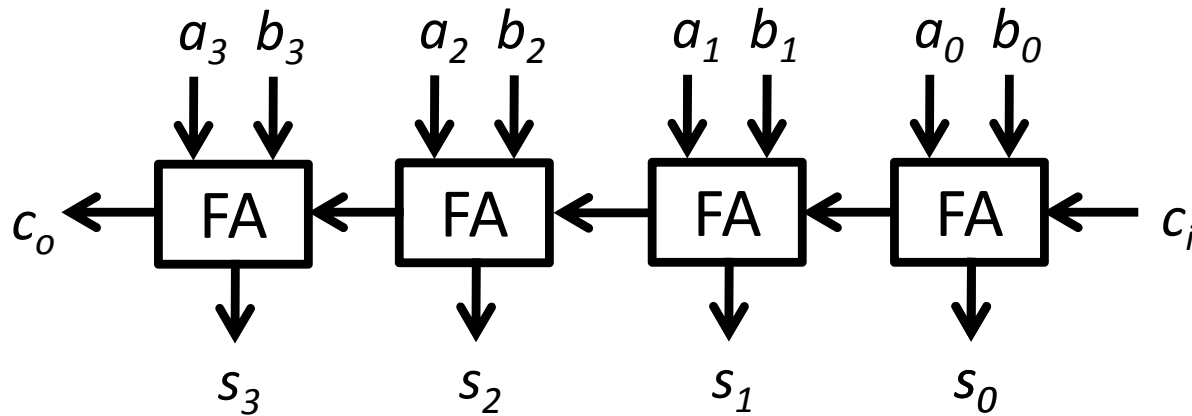
# Sumador

versión 12/09/14

tema 4:  
Módulos combinacionales básicos

FC

50



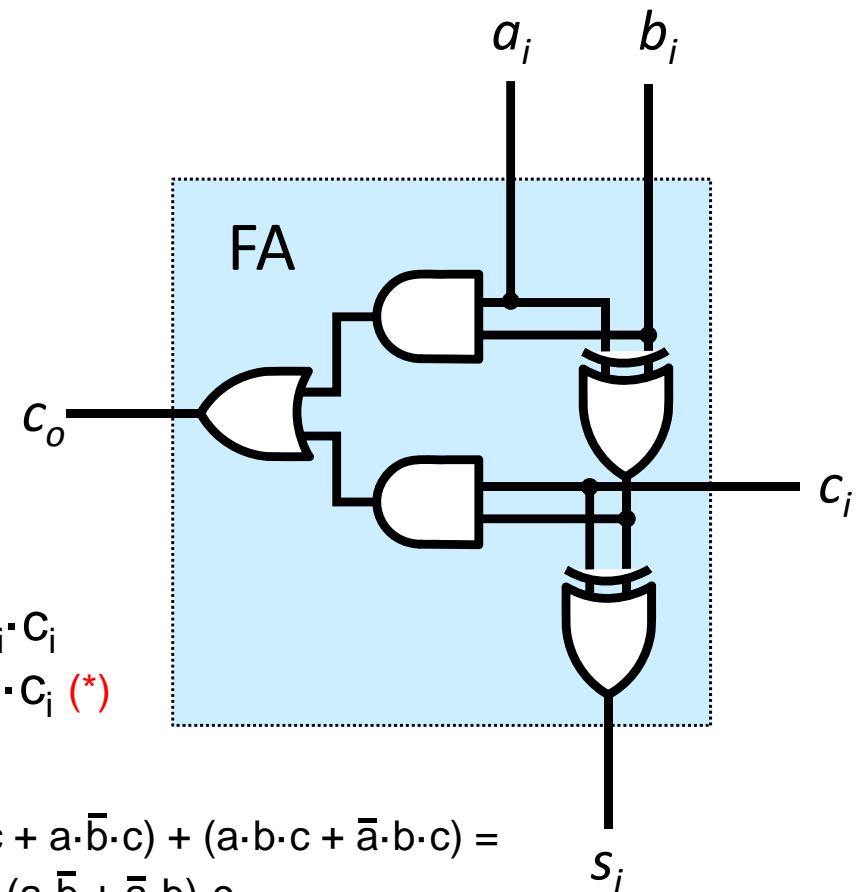
Implementación con  
propagación de acarreo

Sumador de 4 bits

$c_i$	$a_i$	$b_i$	$c_o$	$s_i$
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

$$s_i = (a_i \oplus b_i) \oplus c_i$$

$$\begin{aligned} c_o &= a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i \\ &= a_i \cdot b_i + (a_i \oplus b_i) \cdot c_i (*) \end{aligned}$$

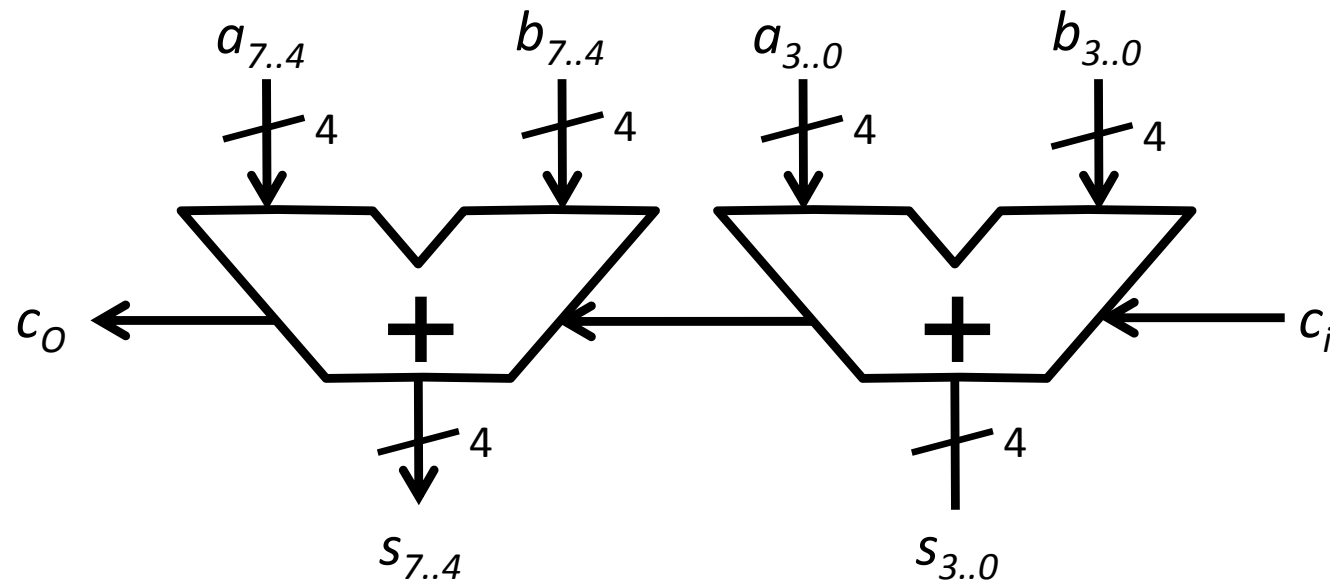


$$\begin{aligned} (*) \quad a \cdot b + a \cdot c + b \cdot c &= a \cdot b + (a \cdot b \cdot c + a \cdot \bar{b} \cdot c) + (a \cdot b \cdot c + \bar{a} \cdot b \cdot c) = \\ &= a \cdot b + a \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot c = a \cdot b + (a \cdot \bar{b} + \bar{a} \cdot b) \cdot c \end{aligned}$$

# Sumador



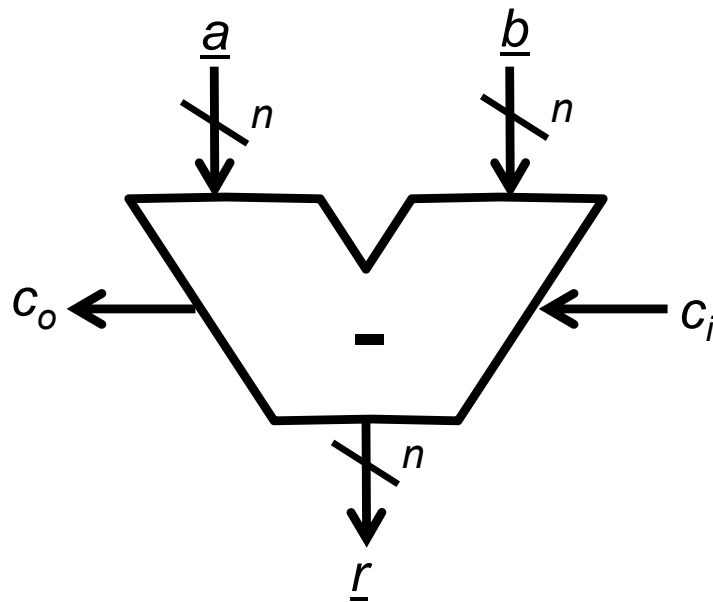
- Varios sumadores se pueden componer en serie para para comportarse como un sumador de **mayor anchura**.



## Implementación serie

Sumador de 8 bits

# Restador



$\underline{a}, \underline{b}$  2 entradas de datos de  $n$  bits

$c_i$  1 entrada de acarreo

$\underline{r}$  1 salida de datos de  $n$  bits

$c_o$  1 salida de acarreo

realiza la resta binaria de  $\underline{a} - \underline{b} - c_i$

$$\underline{s} = (\underline{a} - \underline{b} - c_i) \bmod 2^n$$

$$c_o = \begin{cases} 1 & (\underline{a} - \underline{b} - c_i) < 0 \\ 0 & \text{en caso contrario} \end{cases}$$



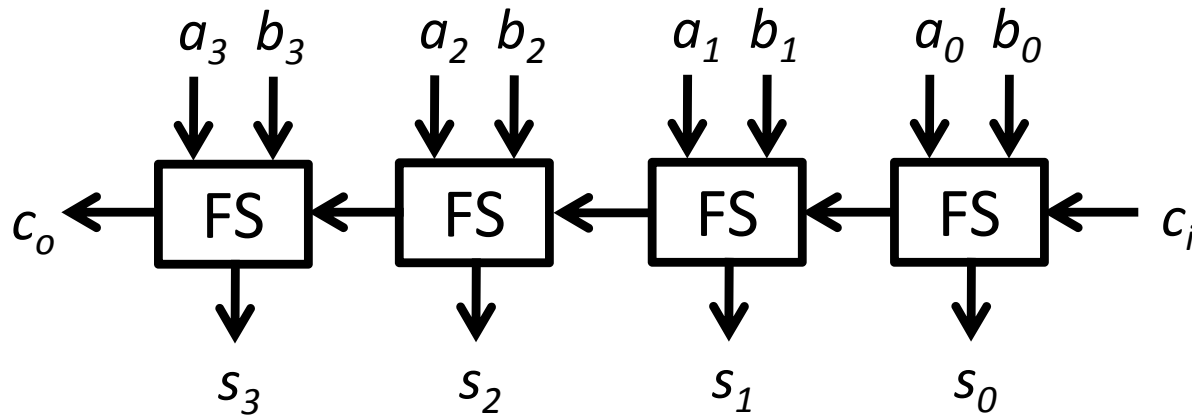
# Restador

versión 12/09/14

tema 4:  
Módulos combinatoriales básicos

FC

53



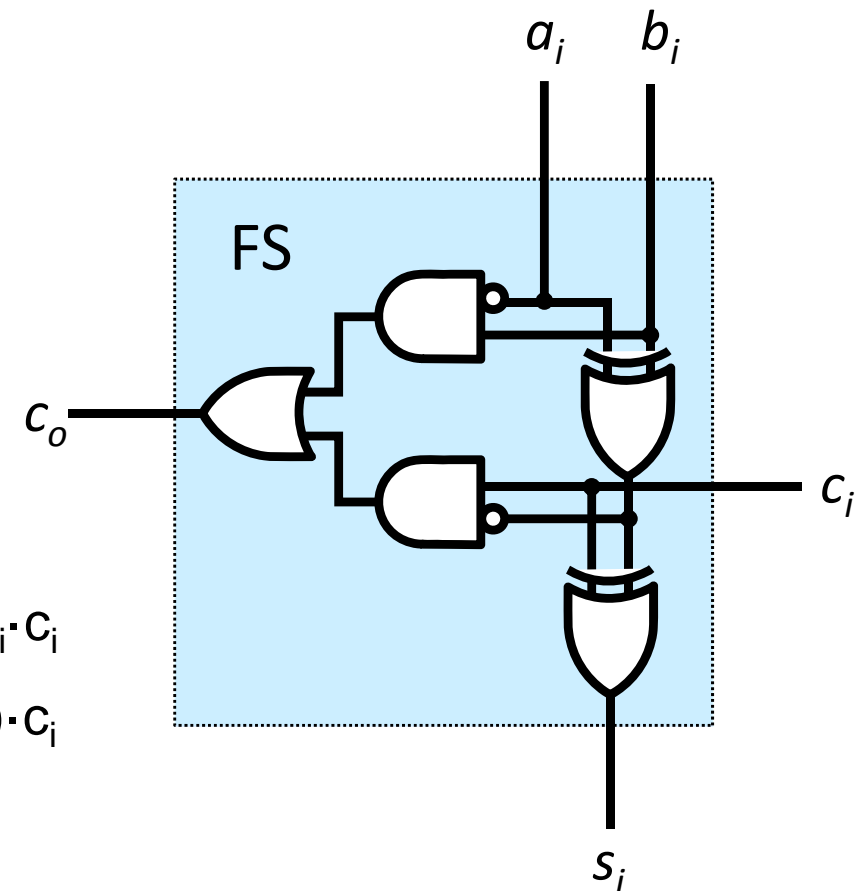
Implementación con  
propagación de acarreo

Restador de 4 bits

$c_i$	$a_i$	$b_i$	$c_o$	$r_i$
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

$$r_i = (a_i \oplus b_i) \oplus c_i$$

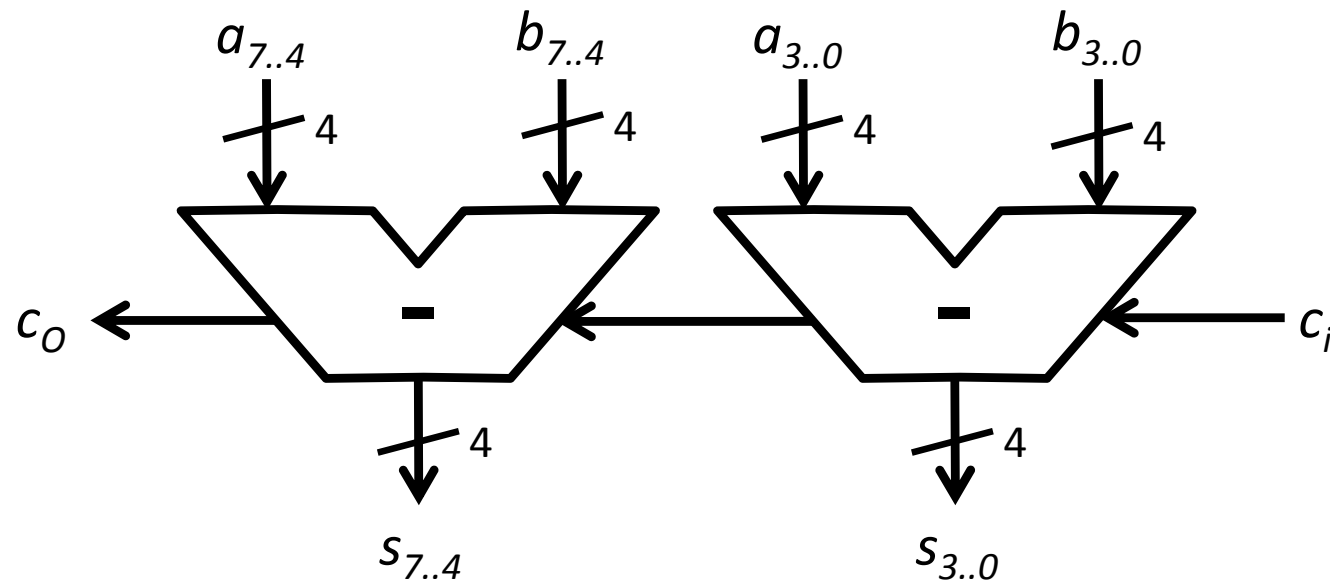
$$\begin{aligned} c_o &= \bar{a}_i \cdot b_i + \bar{a}_i \cdot c_i + b_i \cdot c_i \\ &= \bar{a}_i \cdot b_i + \overline{(a_i \oplus b_i)} \cdot c_i \end{aligned}$$



# Restador



- Varios restadores se pueden componer en serie para para comportarse como un restador de **mayor anchura**.

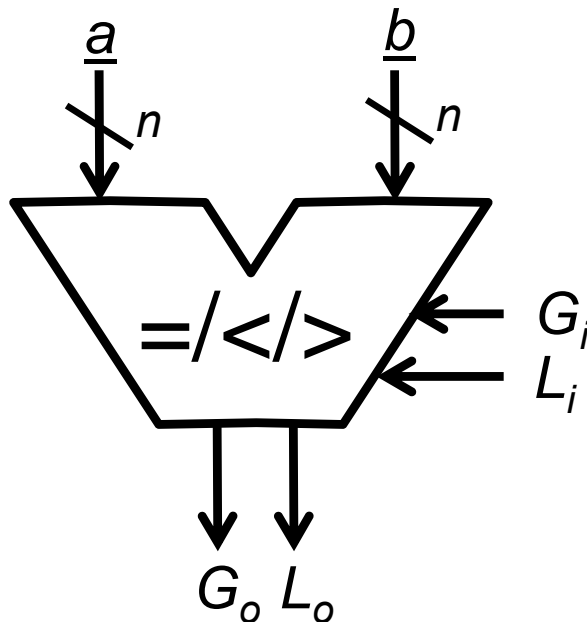


## Implementación serie

Restador de 8 bits



# Comparador de magnitud



$\underline{a}, \underline{b}$  2 entradas de datos de  $n$  bits

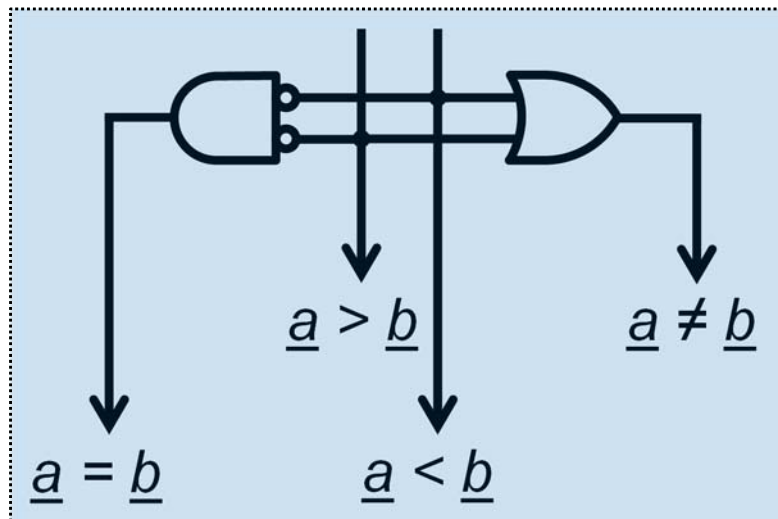
$G_i, L_i$  2 entrada de acarreo

$G_o, L_o$  2 salidas de comparación

*compara 2 números binarios*

$$G_o = \begin{cases} 1 & \text{si } (a > b) \text{ o } (a = b \text{ y } G_i > L_i) \\ 0 & \text{en caso contrario} \end{cases}$$

$$L_o = \begin{cases} 1 & \text{si } (a < b) \text{ o } (a = b \text{ y } G_i < L_i) \\ 0 & \text{en caso contrario} \end{cases}$$





# Comparador de magnitud

versión 12/09/14

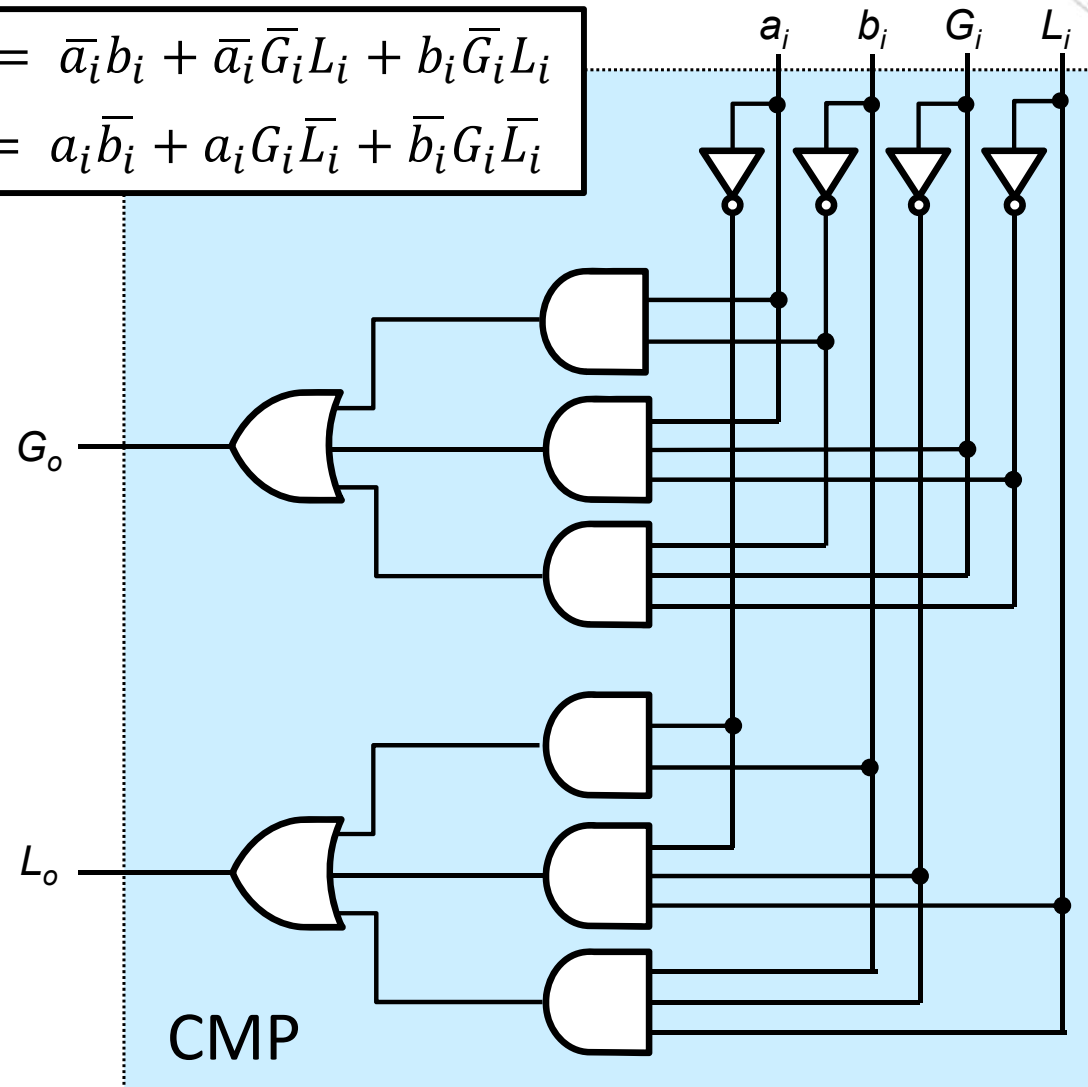
tema 4:  
Módulos combinacionales básicos

FC

56

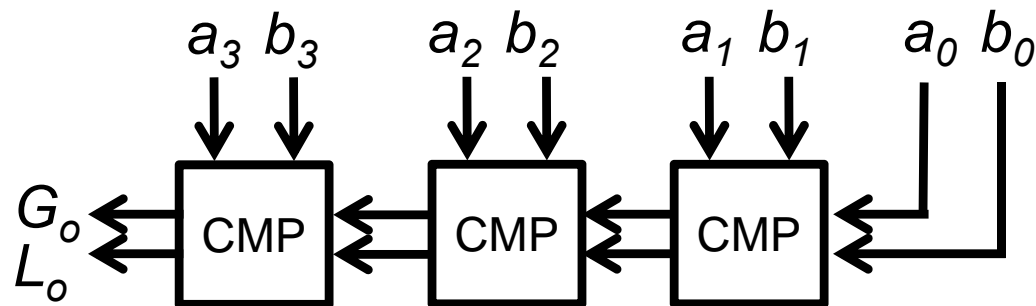
$a_i$	$b_i$	$G_i$	$L_i$	$G_o$	$L_o$
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	0	0

$$L_o = \bar{a}_i b_i + \bar{a}_i \bar{G}_i L_i + b_i \bar{G}_i L_i$$
$$G_o = a_i \bar{b}_i + a_i G_i \bar{L}_i + \bar{b}_i G_i \bar{L}_i$$



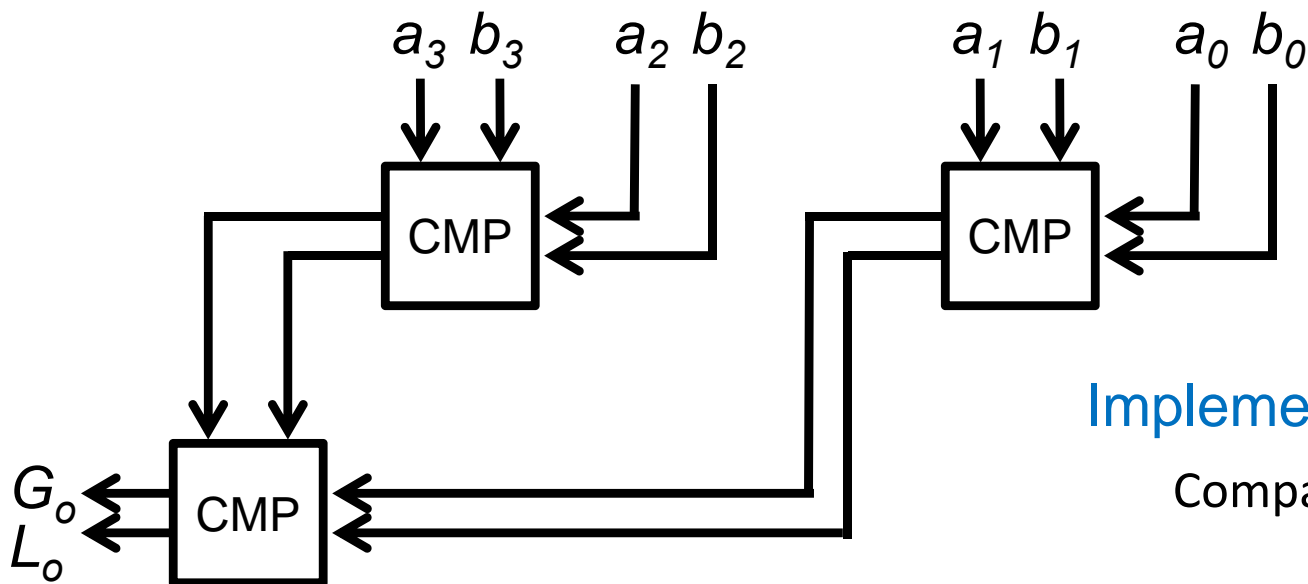


# Comparador de magnitud



Implementación en serie

Comparador de 4 bits



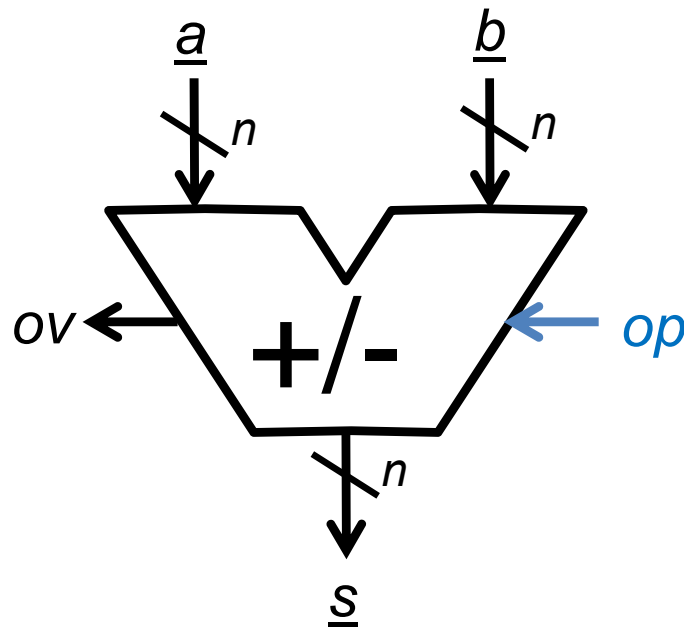
Implementación en árbol

Comparador de 4 bits





# Sumador/restador



$\underline{a}, \underline{b}$  2 entradas de datos de n bits

$op$  1 entrada de selección de operación

$\underline{s}$  1 salida de datos de n bits

$ov$  1 salida de overflow

*realiza la suma/resta en  $\underline{a}$  y  $\underline{b}$   
(interpretados en C2)*

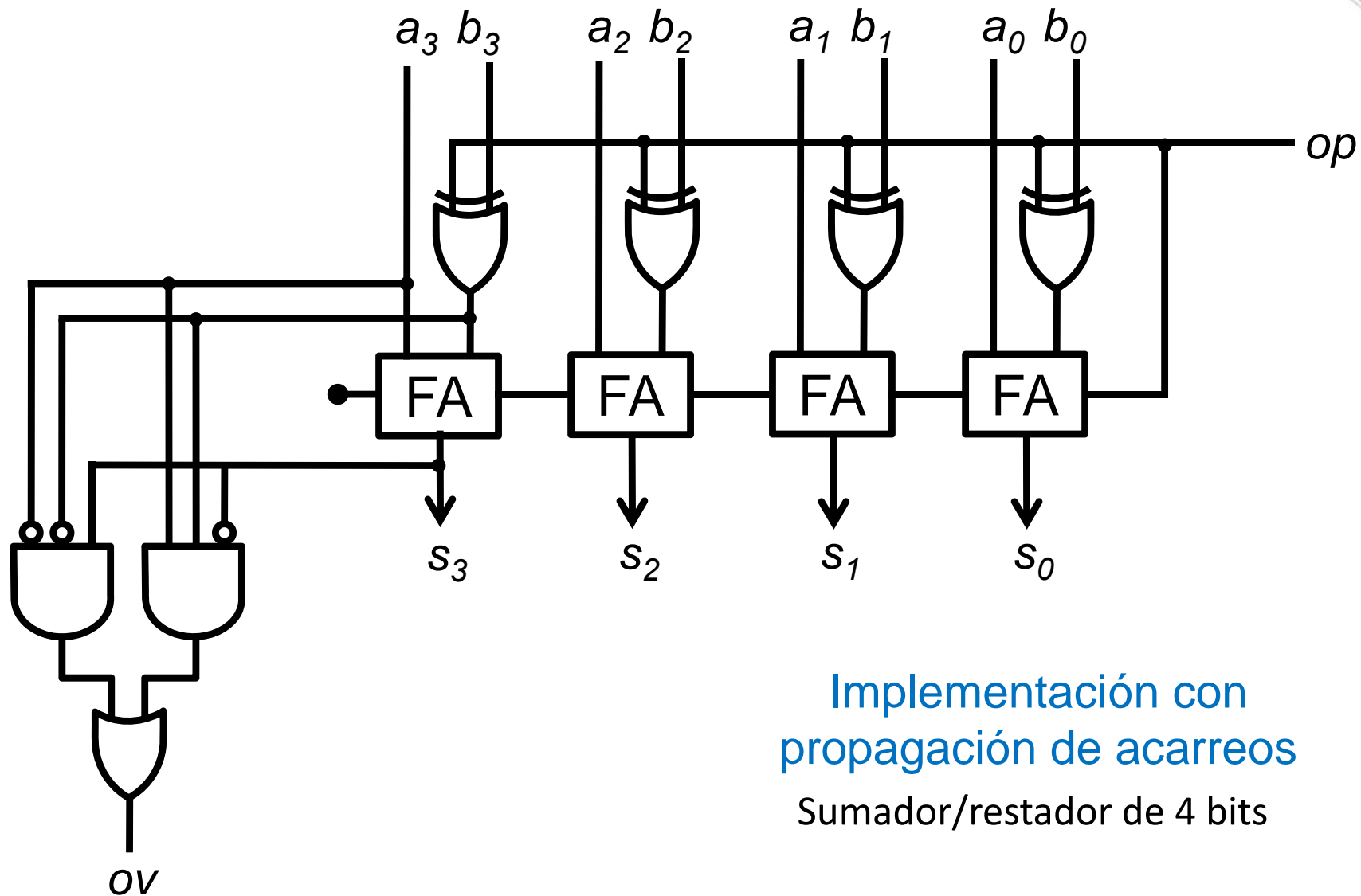
$$\underline{a} - \underline{b} = \underline{a} + (-\underline{b}) =_{C2} \underline{a} + C2(\underline{b}) = \underline{a} + C1(\underline{b}) + 1 = \underline{a} + \overline{\underline{b}} + 1$$

$$\underline{s} = \left\{ \begin{array}{ll} \underline{a} + \underline{b} & \text{si } op = 0 \\ \underline{a} + \overline{\underline{b}} + 1 & \text{si } op = 1 \end{array} \right\} = \underline{a} + (\underline{b} \oplus op) + op$$

$$ov = \left\{ \begin{array}{ll} 1 & (b_{n-1} \oplus op) = 0 \text{ y } a_{n-1} = 0 \text{ y } s_{n-1} = 1 \\ & \text{ó} \\ & (b_{n-1} \oplus op) = 1 \text{ y } a_{n-1} = 1 \text{ y } s_{n-1} = 0 \\ 0 & \text{en caso contrario} \end{array} \right.$$



# Sumador/restador

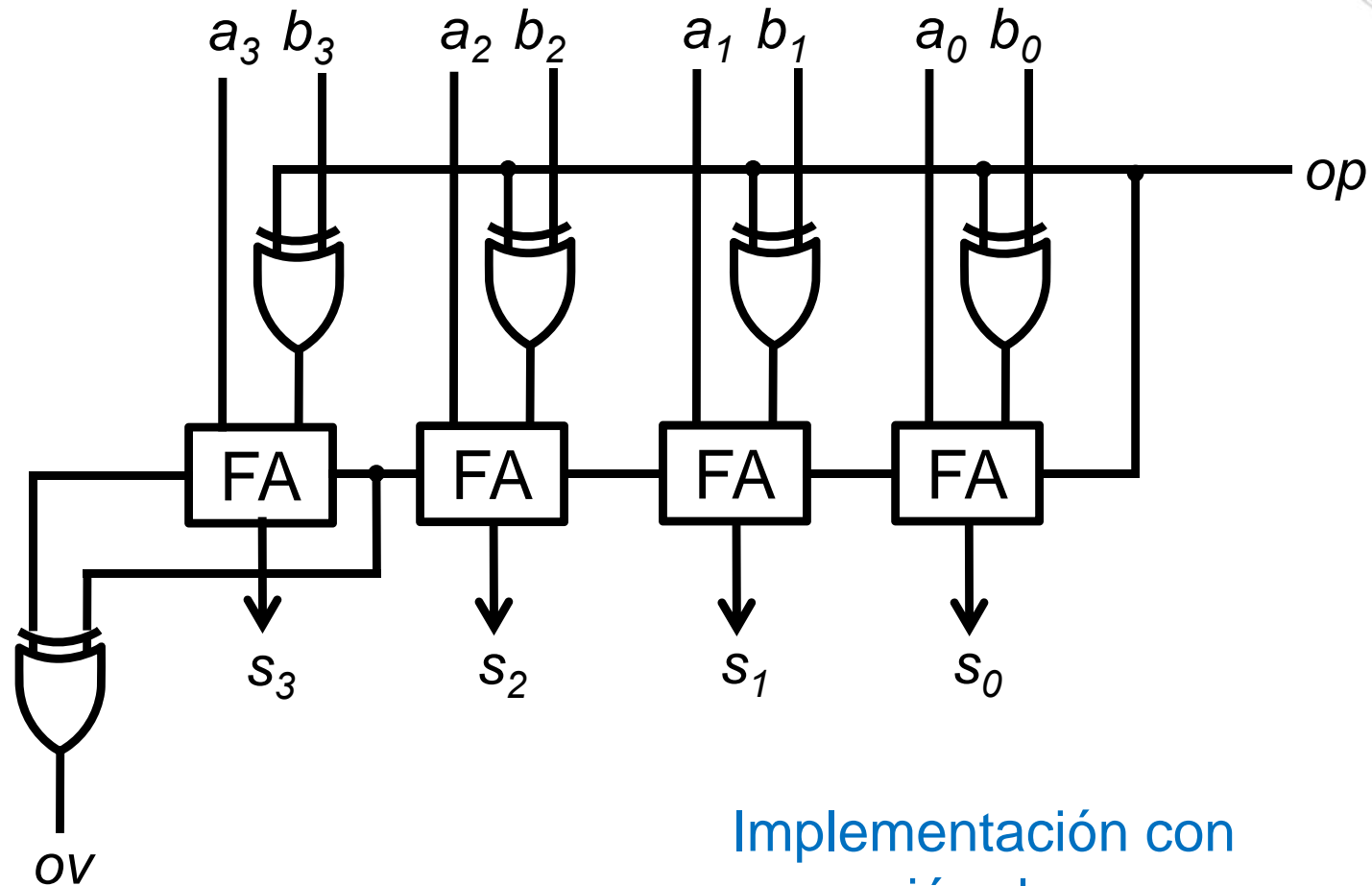


Implementación con  
propagación de acarreo

Sumador/restador de 4 bits



# Sumador/restador

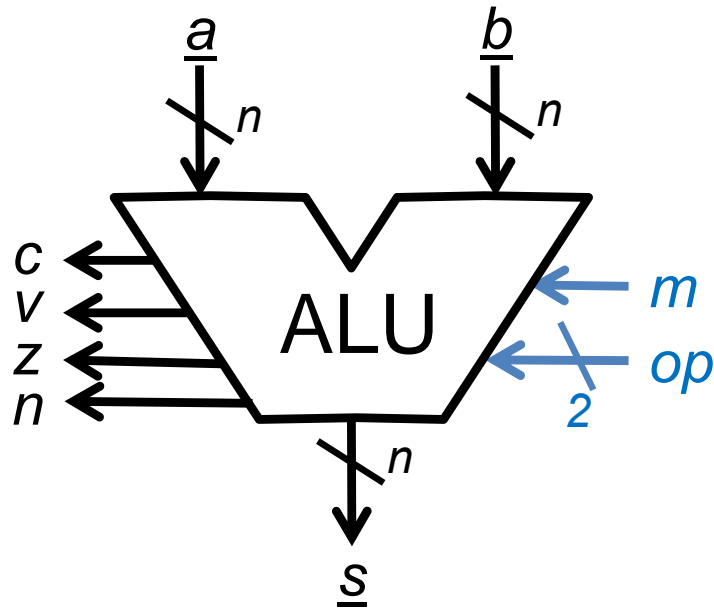


Implementación con  
propagación de acarreo

Sumador/restador de 4 bits



# ALU (Arithmetic-Logic Unit)



- $\underline{a}, \underline{b}$  2 entradas de datos de n bits
- $\underline{m}$  1 entrada de selección de modo
- $\underline{op}$  1 entrada de selección de operación
- $\underline{s}$  1 salida de datos de n bits
- $\underline{c}$  1 salida de acarreo
- $\underline{v}$  1 salida de overflow
- $\underline{z}$  1 salida de detección de cero
- $\underline{n}$  1 salida de detección de negativo

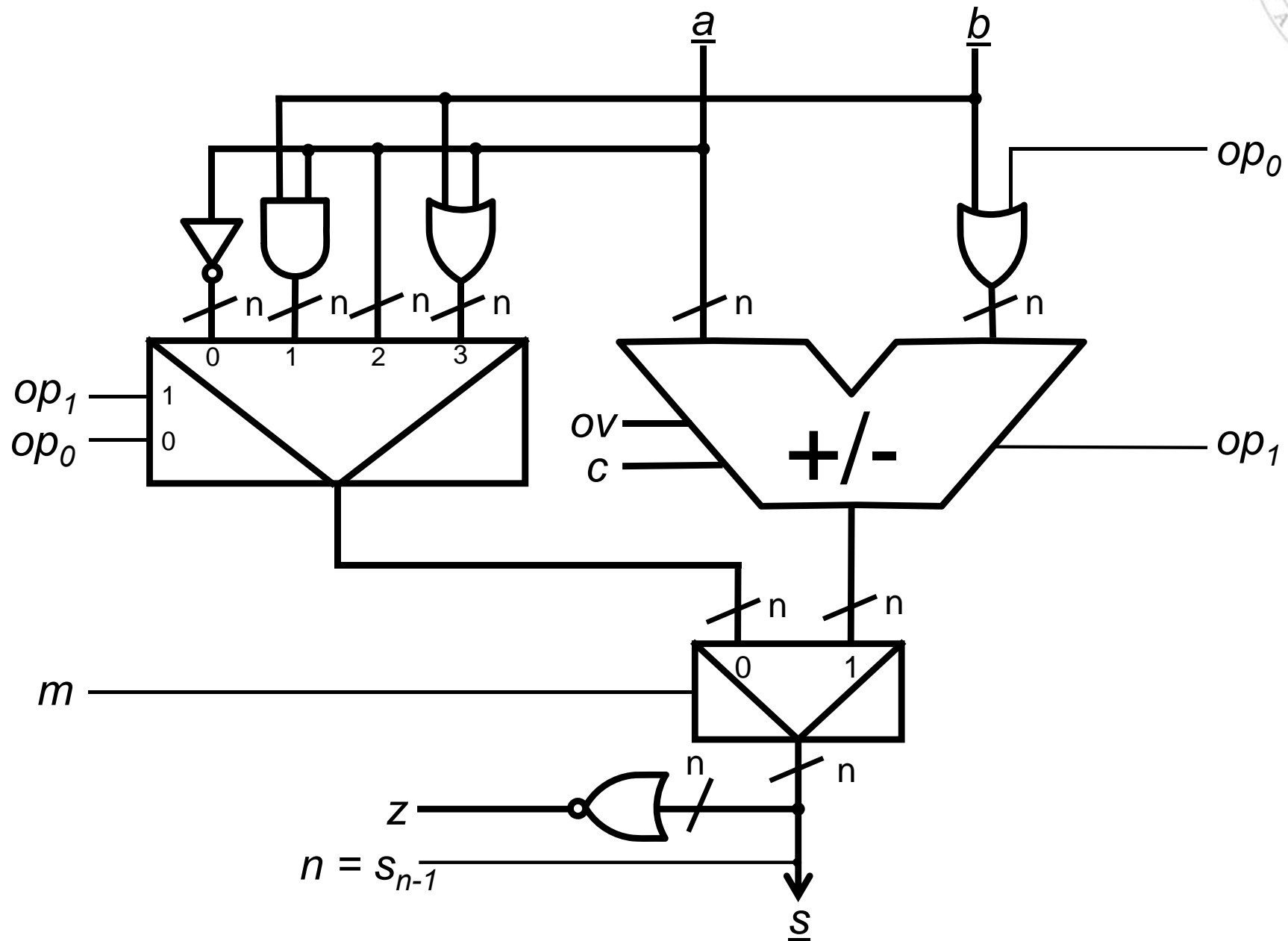
## operaciones lógicas

$\underline{m}$	$\underline{op}_1$	$\underline{op}_0$	$\underline{z}$
0	0	0	$\text{not}(\underline{a})$
0	0	1	$\text{and}(\underline{a}, \underline{b})$
0	1	0	$\underline{a}$
0	1	1	$\text{or}(\underline{a}, \underline{b})$

## operaciones aritméticas

$\underline{m}$	$\underline{op}_1$	$\underline{op}_0$	$\underline{z}$
1	0	0	$\underline{a} + \underline{b}$
1	0	1	$\underline{a} - 1 = \underline{a} + (-1) =_{c2} \underline{a} + \underline{1}$
1	1	0	$\underline{a} - \underline{b}$
1	1	1	$\underline{a} + 1 = \underline{a} - (-1) =_{c2} \underline{a} - \underline{1}$

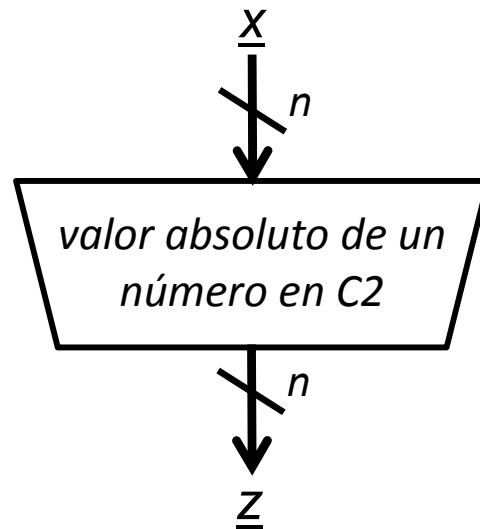
# ALU (Arithmetic-Logic Unit)





# Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.



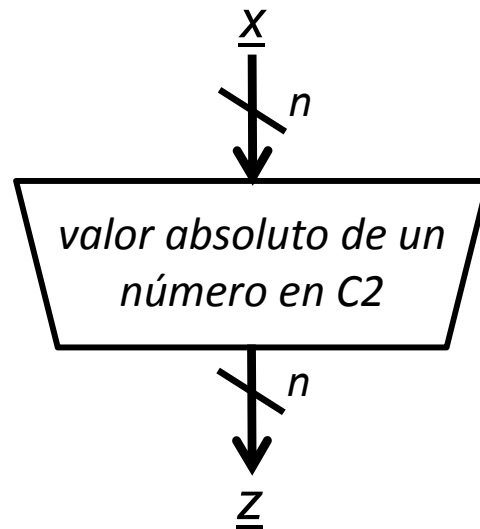
$$\underline{z} = \text{abs}(\underline{x}) = \begin{cases} \underline{x} & \text{si } x \geq 0 \\ -\underline{x} & \text{si } x < 0 \end{cases}$$

$$\underline{z} = \text{abs}(\underline{x}) =_{C2} \begin{cases} \underline{x} & \text{si } x_{n-1} = 0 \\ C2(\underline{x}) = \text{not}(\underline{x}) + 1 & \text{si } x_{n-1} = 1 \end{cases}$$



# Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.

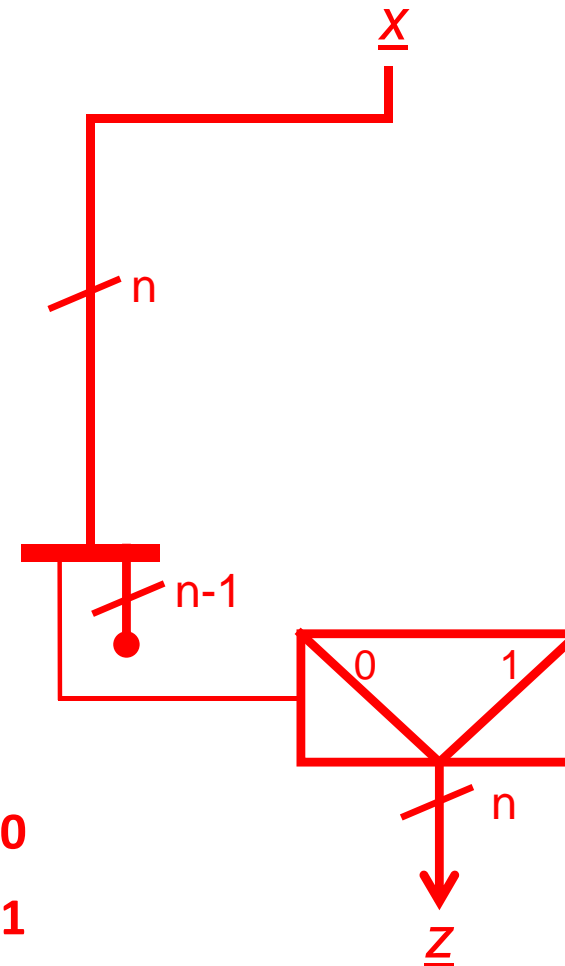


$$z = \text{abs}(x) = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$$

$$z = \text{abs}(x) =_{C2} \begin{cases} x \\ C2(x) = \text{not}(x) + 1 \end{cases}$$

$$\text{si } x_{n-1} = 0$$

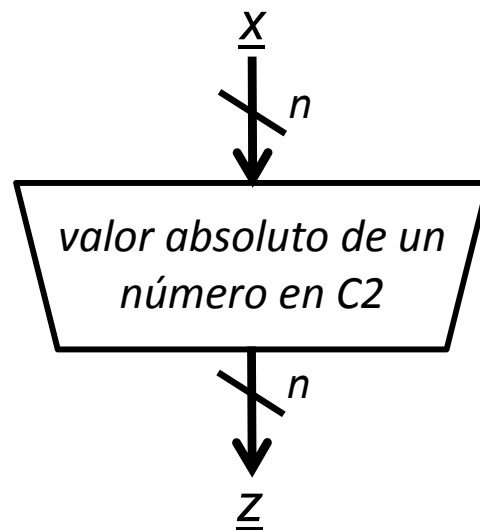
$$\text{si } x_{n-1} = 1$$





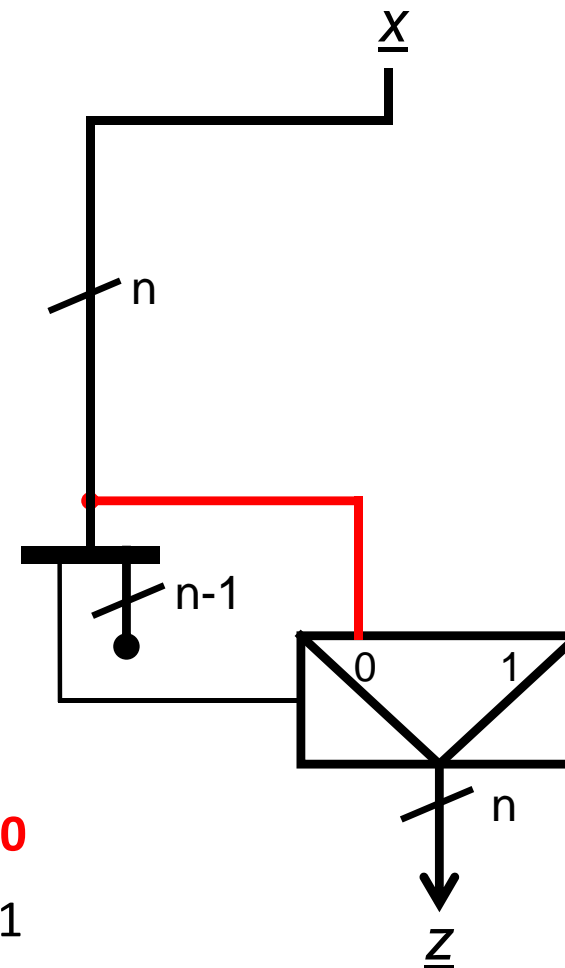
# Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.



$$z = \text{abs}(x) = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$$

$$z = \text{abs}(x) =_{C2} \begin{cases} x & \text{si } x_{n-1} = 0 \\ C2(x) = \text{not}(x) + 1 & \text{si } x_{n-1} = 1 \end{cases}$$

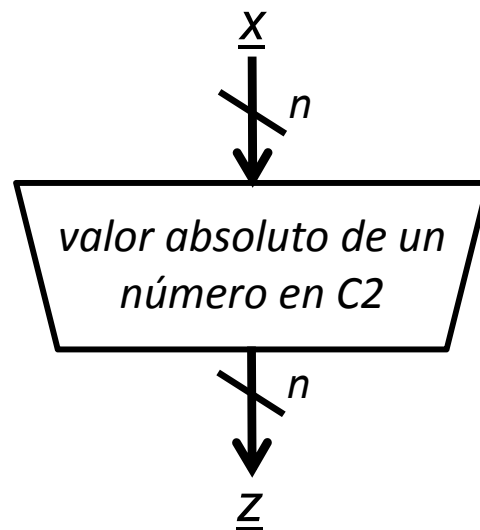






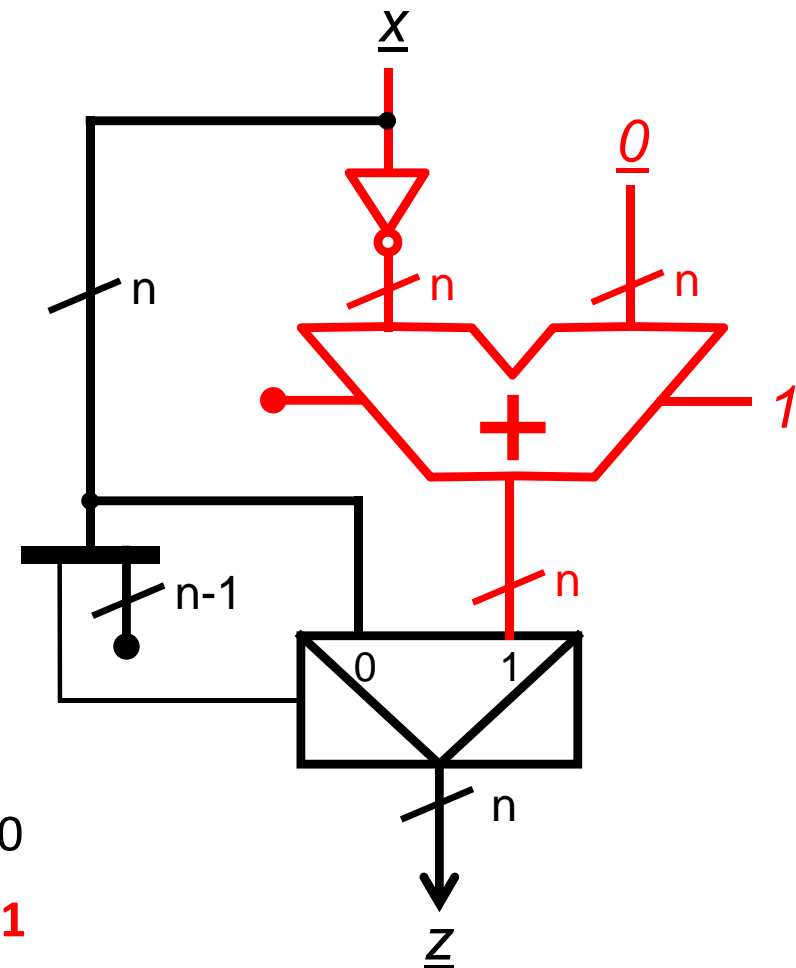
# Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.



$$z = \text{abs}(x) = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$$

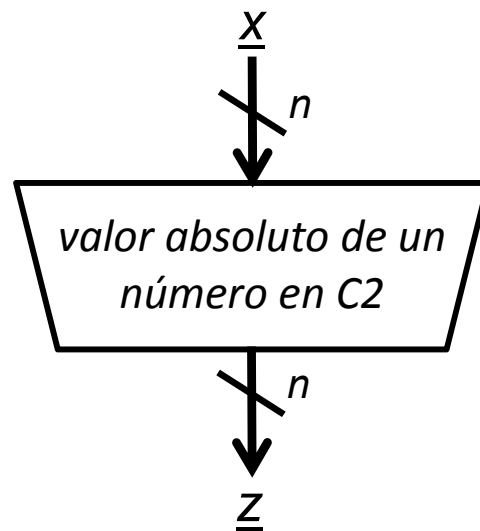
$$z = \text{abs}(x) =_{C2} \begin{cases} x & \text{si } x_{n-1} = 0 \\ C2(x) = \text{not}(x) + 1 & \text{si } x_{n-1} = 1 \end{cases}$$





# Recapitulación

- En muchos casos es posible obtener directamente una red de módulos combinacionales desde un enunciado.



$$z = \text{abs}(x) = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$$

$$z = \text{abs}(x) =_{C2} \begin{cases} x & \text{si } x_{n-1} = 0 \\ C2(x) = \text{not}(x) + 1 & \text{si } x_{n-1} = 1 \end{cases}$$

