Capítulo V
Bloques Digitales

Cristián Tejos
Segundo Semestre 2017

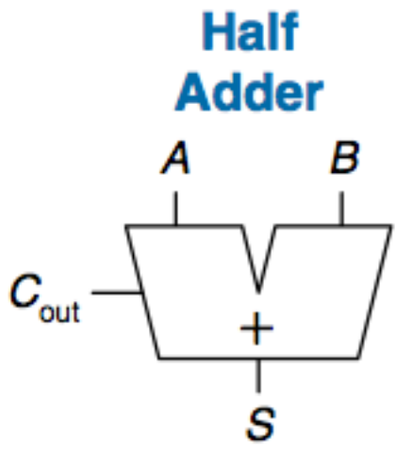
Basado en apuntes de Marcelo Guarini
y el libro
Digital Design and Computer Architecture (1st edition),
D. M. Harris & S. L. Harris, Elsevier 2007

Introducción

- Hasta ahora hemos examinado el diseño de circuitos combinacionales y secuenciales utilizando álgebra de Boole, esquemáticos y Verilog.
- En este capítulo introduciremos bloques combinacionales y secuenciales más elaborados que normalemente se utilizan en sistemas digitales más avanzados.
- Estos bloques incluyen: circuitos aritméticos, contadores registros de desplazamiento, memorias y arreglos de memorias y arreglos lógicos.
- Estos bloques son ensamblados jerárquicamente a partir de componentes simples tales como: compuertas lógicas, multiplexores y decodificadores.

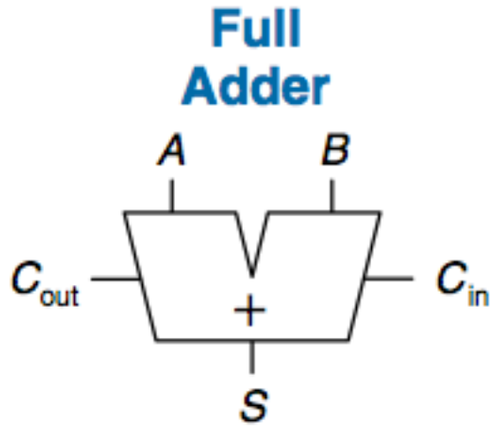
Circuitos Aritméticos

Suma



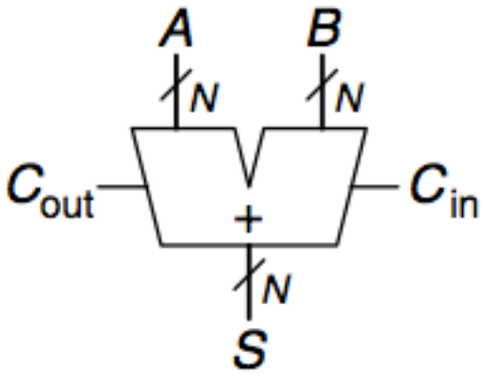
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = AB$$



C_{in}	A	B	C_{out}	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

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

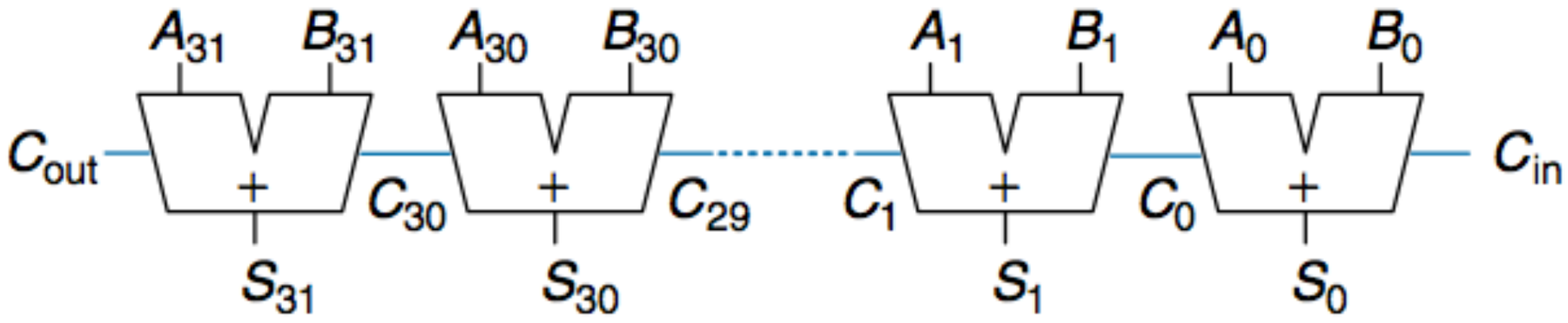


Circuitos Aritméticos

Suma

Sumador *Ripple Carry*

$$t_{\text{ripple}} = Nt_{FA}$$



Circuitos Aritméticos

Suma

Sumador con *Carry Lookahead*

- El sumador se divide en bloques y se provee los circuitos necesarios para determinar el carry en forma rápida.
- Por ejemplo, podemos dividir el sumador de 32 bits en 8 bloques de 4 bits c/u.
- Estos sumadores utilizan dos señales, *generate* (*G*) y *propagate* (*P*) que describen cómo una columna o bloque determina el Cout.

Circuitos Aritméticos

Suma

Sumador con *Carry Lookahead*

- La i-ésima columna de un sumador es seguro que genera un carry, C_i, si A_i y B_i son ambos 1.
- Se dice que la i-ésima columna propaga un carry si produce un carry cuando existe un Cin (C_{i-1}=1)
- Utilizando estas definiciones, podemos re-escribir la lógica correspondiente al carry para cualquier columna del sumador

$$C_i = A_i B_i + (A_i + B_i) C_{i-1} = G_i + P_i C_{i-1}$$

Circuitos Aritméticos

Suma

Sumador con *Carry Lookahead*

- La definición de *generate* y de *propagate* se pueden extender a bloques de múltiples bits.
- Se dice que un bloque genera un carry si genera un carry independiente de si el bloque recibe un carry.
- Se dice que un bloque propaga un carry si produce un carry cuando existe un carry que entra al bloque.
- Definimos entonces Gi:j y Pi:j como las señales correspondientes. Entonces

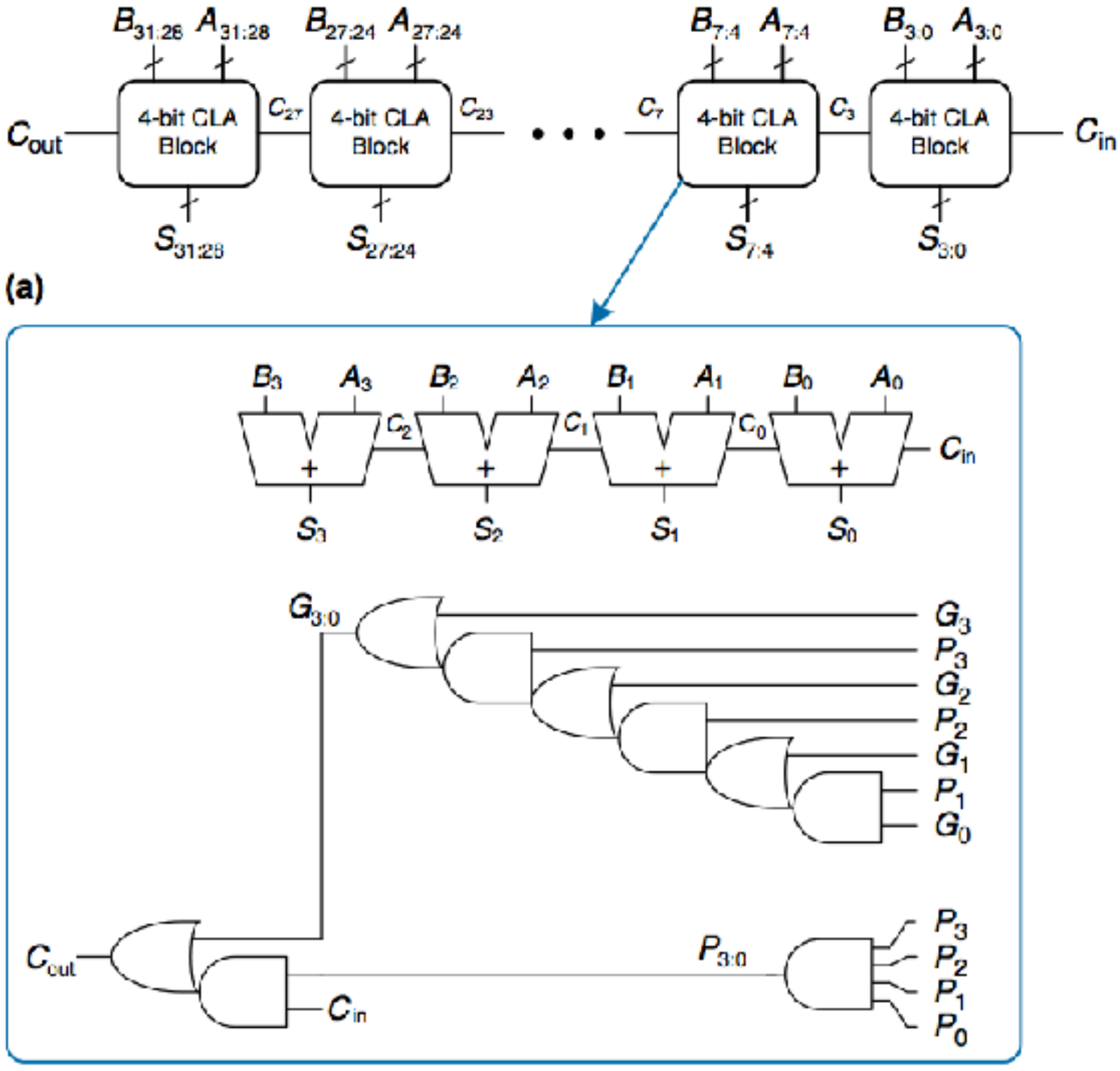
$$G_{3:0} = G_3 + P_3(G_2 + P_2(G_1 + P_1G_0))$$

$$P_{3:0} = P_3P_2P_1P_0$$

Circuitos Aritméticos

Suma

Sumador con *Carry Lookahead*



Circuitos Aritméticos

Suma

Retardo del sumador con *Carry Lookahead*

$$t_{CLA} = t_{pg} + t_{pg_\text{block}} + \left(\frac{N}{k} - 1\right) t_{\text{AND_OR}} + k t_{FA}$$

- Tiempo de retardo de un sumador *ripple* y de un sumador con *carry lookahead*, ambos de 32 bits.
- suponiendo que el retardo de una compuerta es de 100 ps y el de un full adder de 300 ps.

Ripple adder:

$$32 \times 300 \text{ ps} = 9.6 \text{ ns}$$

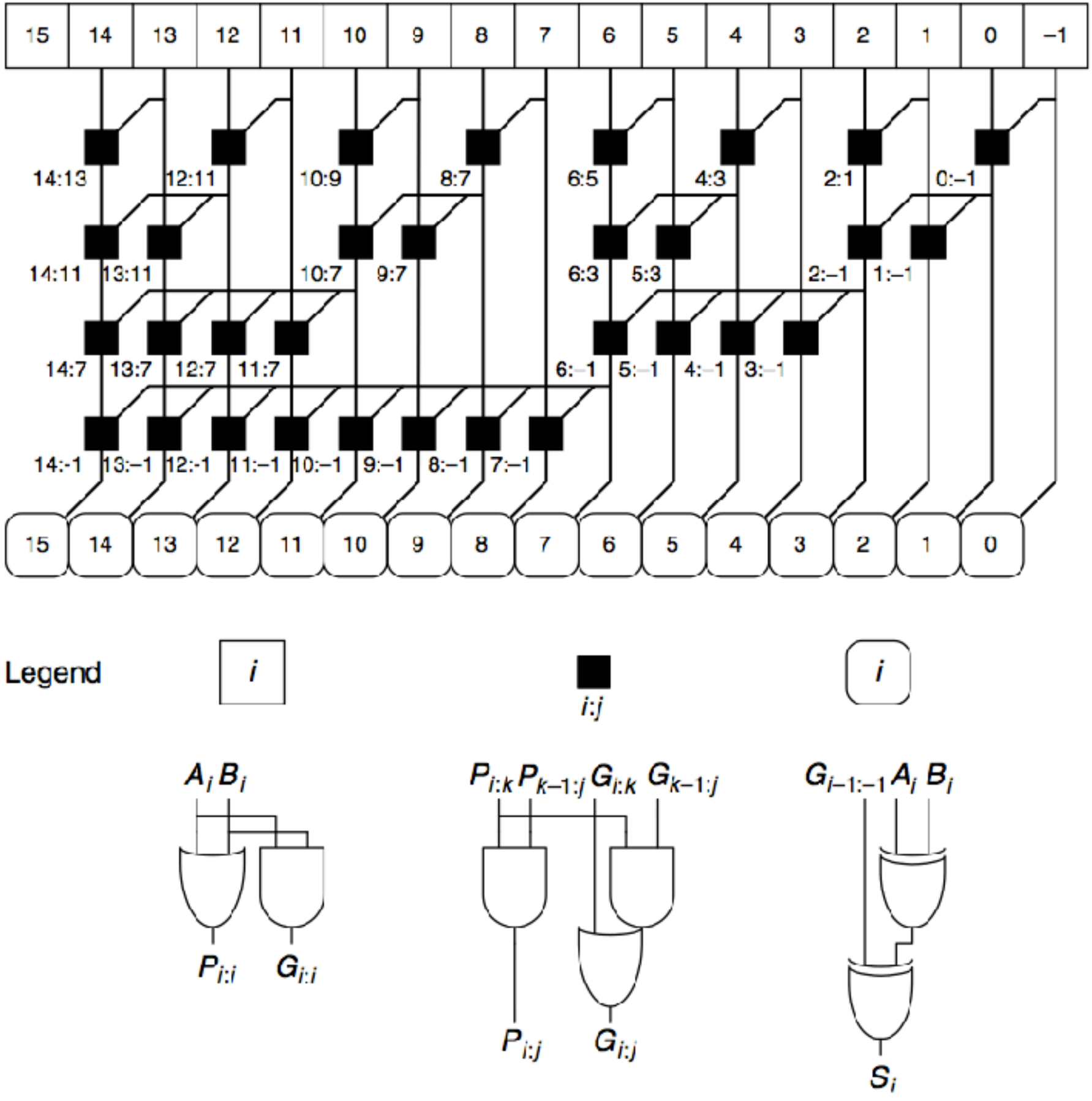
Carry lookahead adder:

$$100 \text{ ps} + (6 \times 100 \text{ ps}) + (\frac{32}{4} - 1)(2 \times 100 \text{ ps}) + (4 \times 300 \text{ ps}) = 3.3 \text{ ns}$$

Circuitos Aritméticos

Suma

Sumador con prefijo



Circuitos Aritméticos

Suma

Retardo del sumador con prefijo

$$t_{PA} = t_{pg} + \log_2 N(t_{pg_\text{prefix}}) + t_{\text{XOR}}$$

- Un sumador de 32 bits con prefijo y las mismas compuertas que los dos anteriores, tiene un retardo de 1.2 ns

Circuitos Aritméticos

Suma

Descripción de un sumador *ripple* de 8 bits en Verilog

```
module adder #(parameter N = 8)
              (input  [N-1:0] a, b,
               input          cin,
               output [N-1:0] s,
               output         cout);

  assign {cout, s} = a + b + cin;
endmodule
```

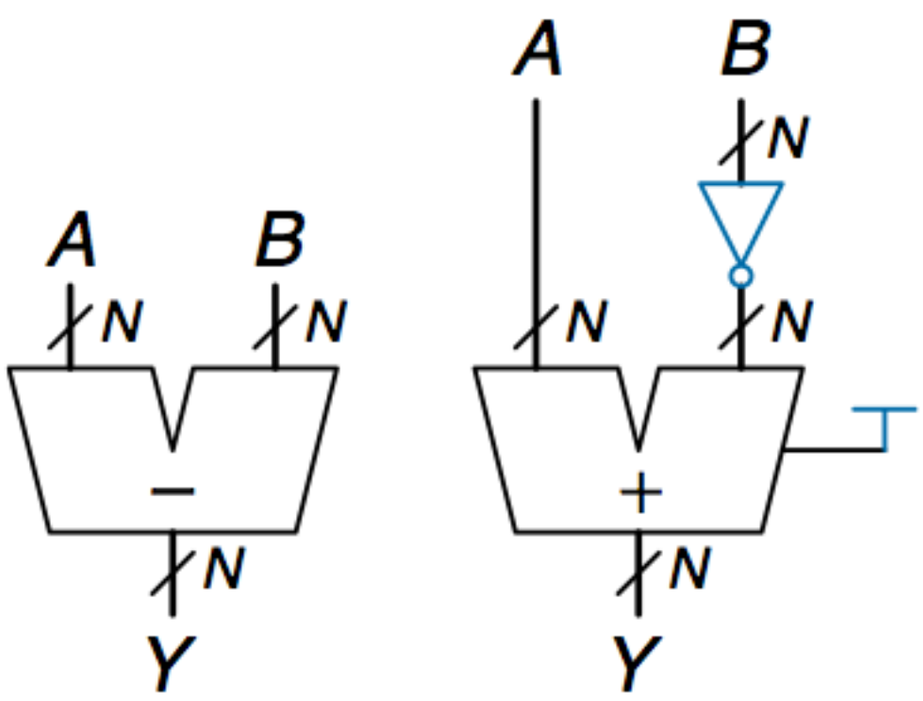
Circuitos Aritméticos

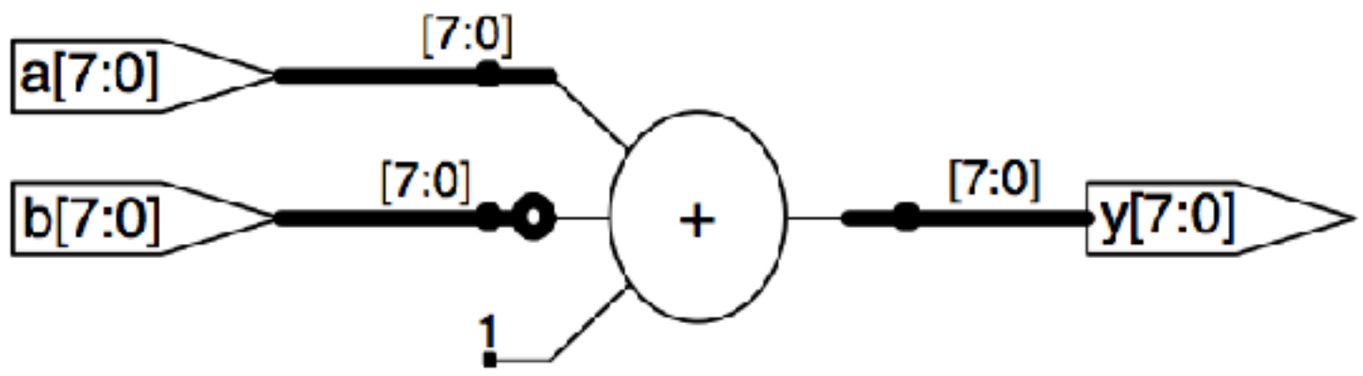
Resta

- Recordemos que la resta se realiza con un sumador utilizando código complemento de dos

```
module subtractor #(parameter N = 8)
                   (input  [N-1:0] a, b,
                    output [N-1:0] y);

  assign y = a - b;
endmodule
```



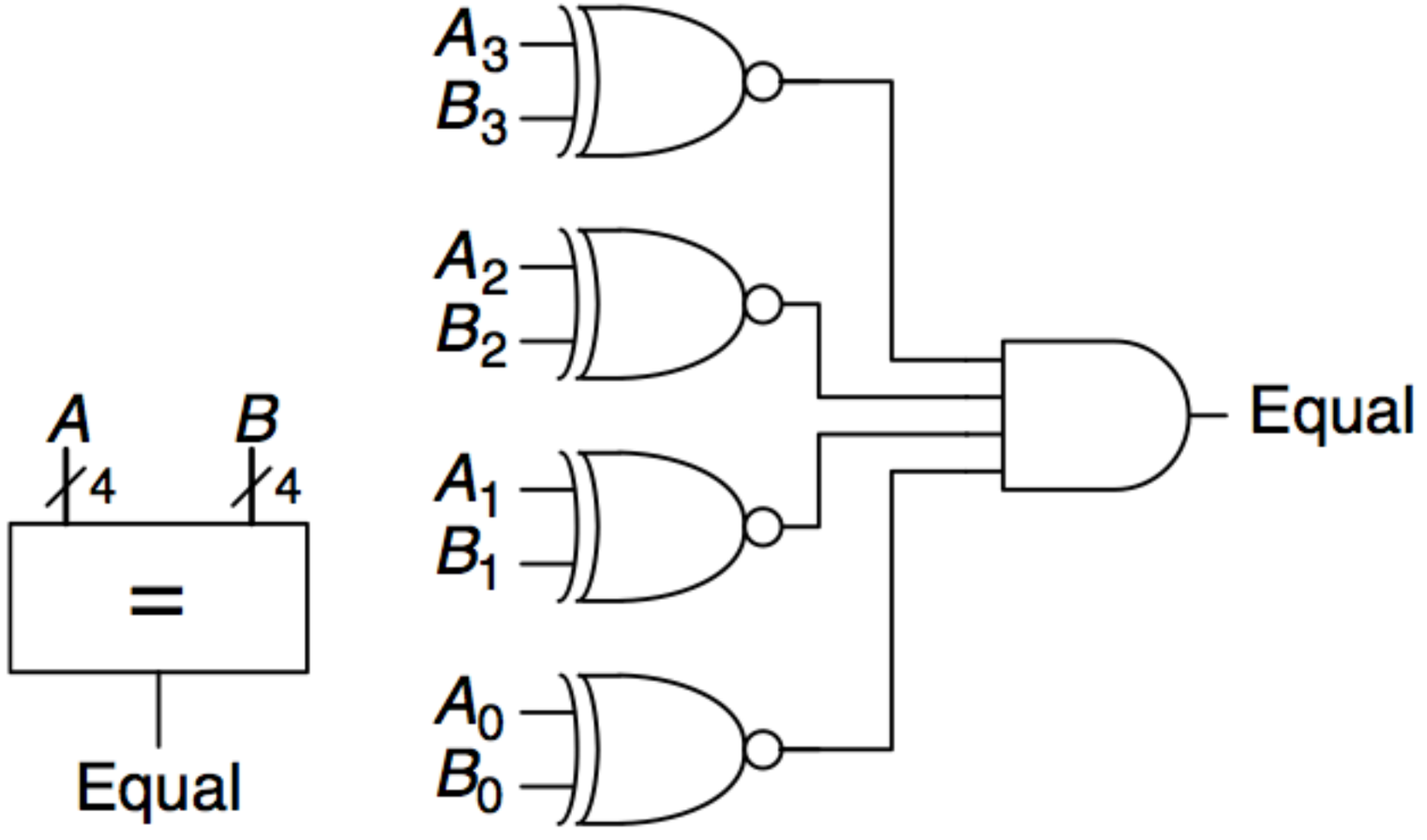


Circuitos Aritméticos

Comparadores

Comparador de igualdad

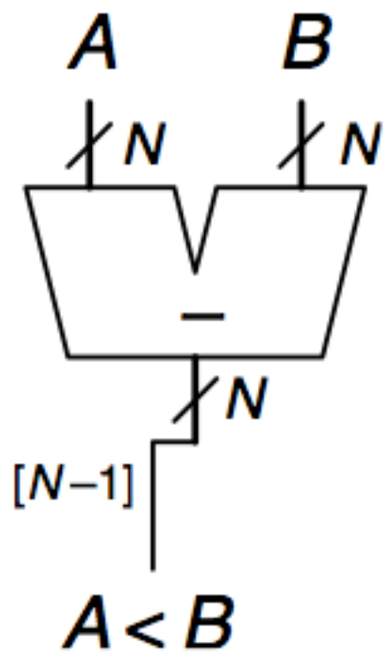
- Un comparador binario determina si dos números son iguales o si uno es mayor que el otro.



Circuitos Aritméticos

Comparadores

Comparador de magnitud



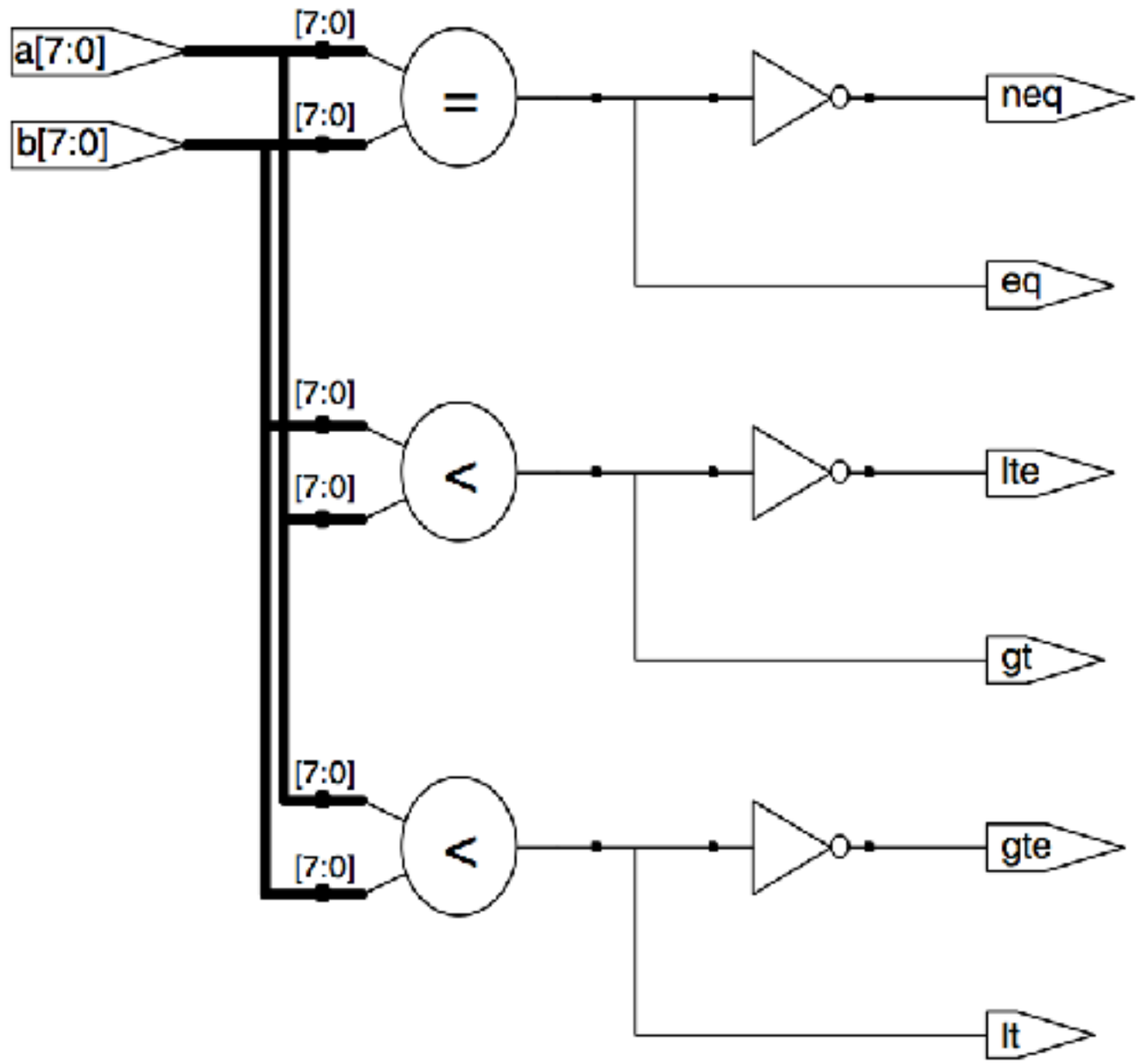

```
module comparators # (parameter N = 8)
                    (input [N-1:0] a, b,
                     output        eq, neq,
                     output        lt, lte,
                     output        gt, gte);

  assign eq  = (a == b);
  assign neq = (a != b);
  assign lt  = (a < b);
  assign lte = (a <= b);
  assign gt  = (a > b);
  assign gte = (a >= b);
endmodule
```

Circuitos Aritméticos

Comparadores

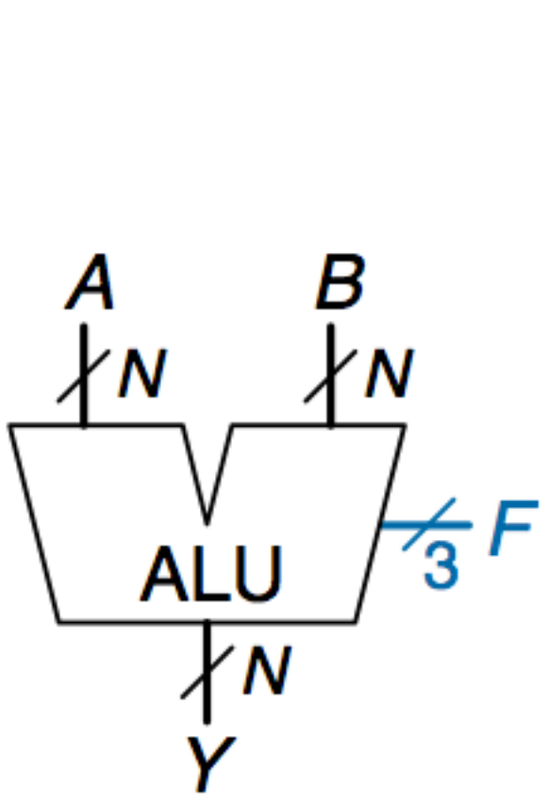
Comparador de magnitud sintetizado



Circuitos Aritméticos

ALU

Unidad Aritmética Lógica



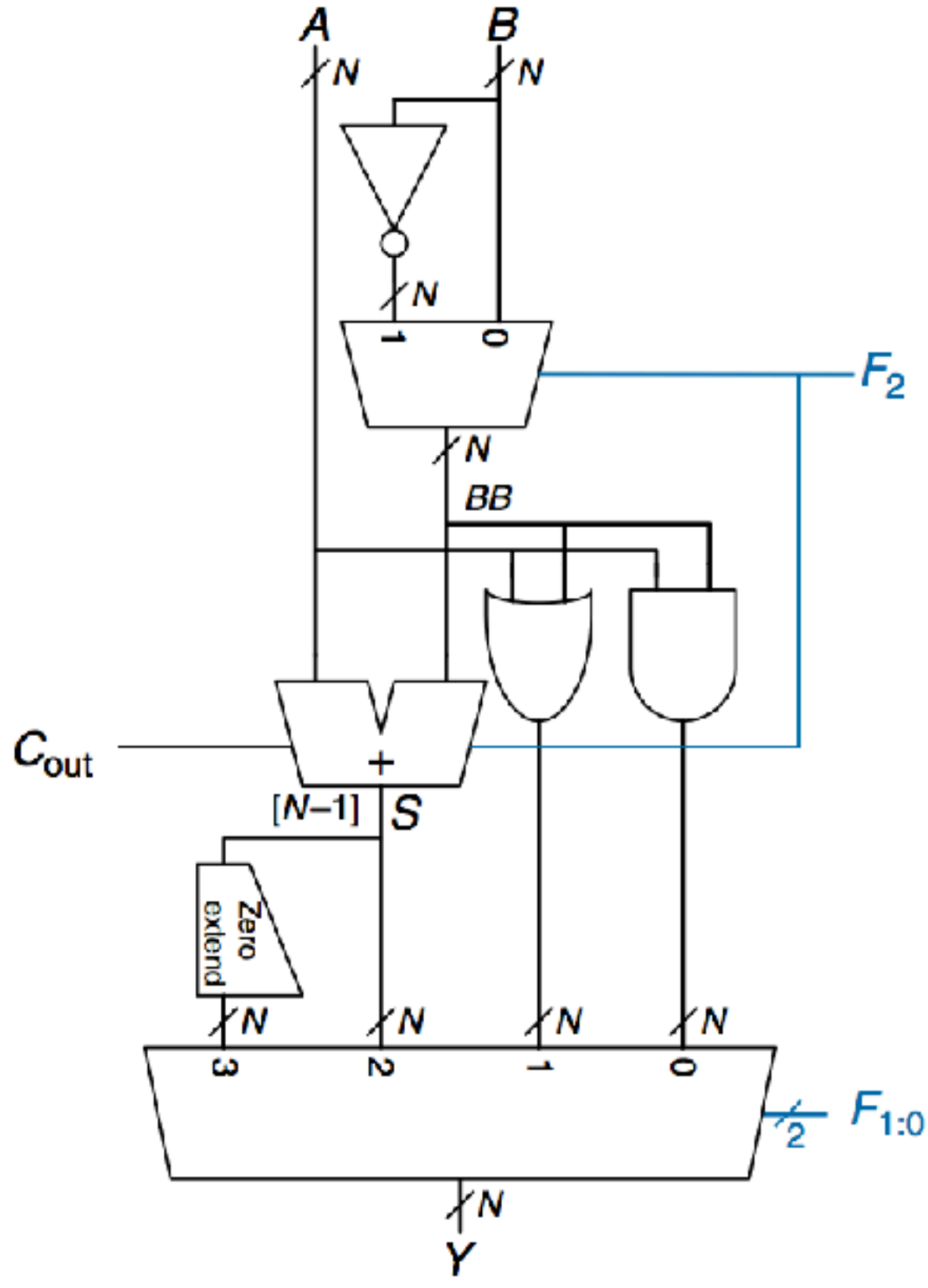
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A − B
111	SLT

Circuitos Aritméticos

ALU

Unidad Aritmética Lógica

$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A − B
111	SLT



Circuitos Aritméticos

ALU

Unidad Aritmética Lógica

- Algunas ALUs generan salidas extras llamadas *flags*, que indican información sobre la salida de la ALU.
- El flag *overflow* indica cuando una operación de suma se sale de la representación posible dado el número de bits.
- El flag *zero* indica cuando la operación ha generado un cero como salida.

Circuitos Aritméticos

Desplazadores y Rotadores

Desplazador Lógico

- Un desplazador lógico desplaza el número a la izquierda (LSL) o a la derecha (LSR) y llena los espacios vacíos con 0s.
- Ej: $11001 \text{ LSR } 2 = 00110; 11001 \text{ LSL } 2 = 00100$

Circuitos Aritméticos

Desplazadores y Rotadores

Desplazador Aritmético

- El desplazador aritmético es similar al desplazador lógico, pero cuando desplaza a la derecha llena los espacios vacíos con una copia del antiguo bit más significativo. Esto es útil para multiplicar y dividir números con signo.
- El desplazamiento aritmético a la izquierda es igual que el desplazamiento lógico a la izquierda.
- Ejs: 11001; ASR 2 = 01110; 11001 ASL 2 = 00100

Circuitos Aritméticos

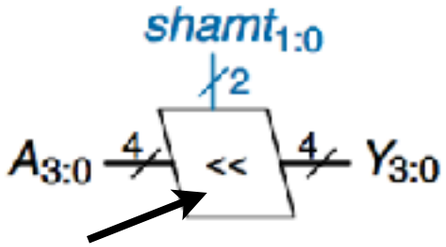
Desplazadores y Rotadores

Rotador

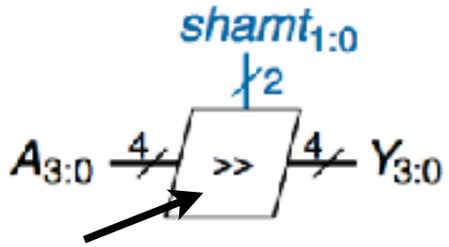
- Un rotador rota los bits a la derecha o izquierda llenando los espacios vacíos con los bits que salen fuera por el otro extremo.
- Ejs: $11001\ \text{ROR}\ 2 = 01110;\ 11001\ \text{ROL}\ 2 = 00111$

Circuitos Aritméticos

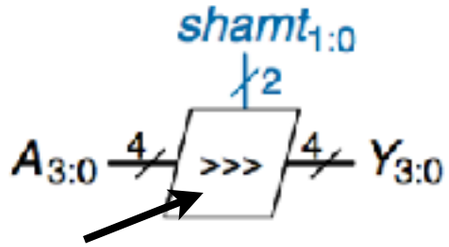
Desplazadores y Rotadores



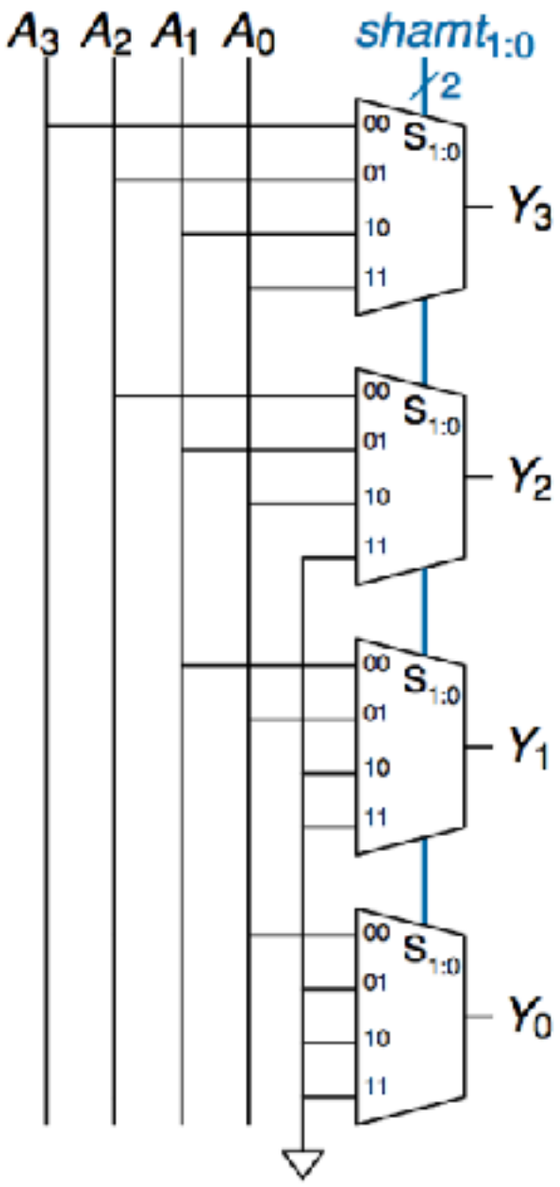
Shift Left

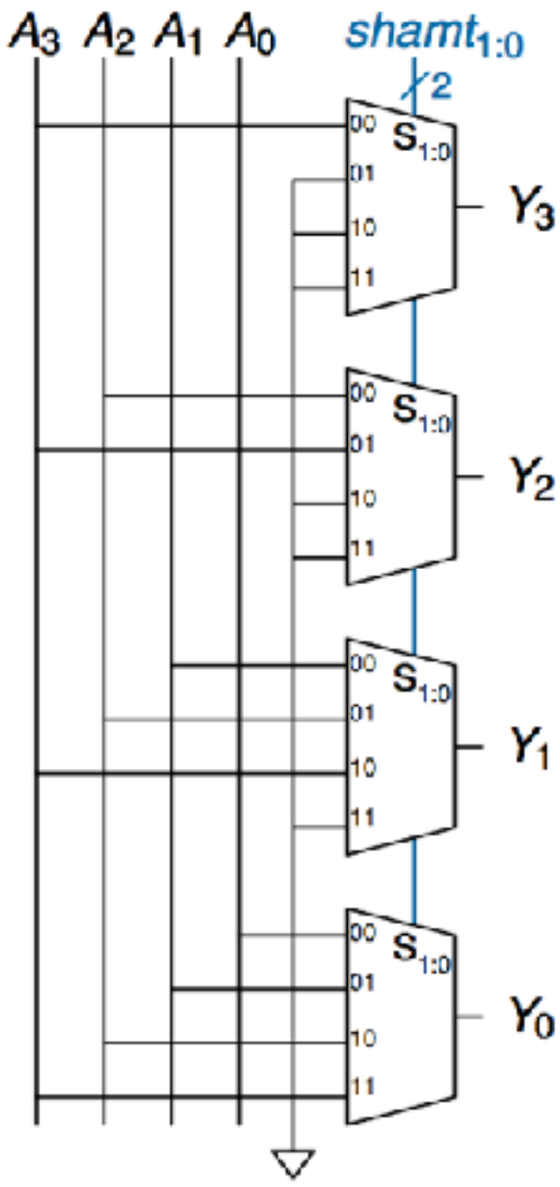


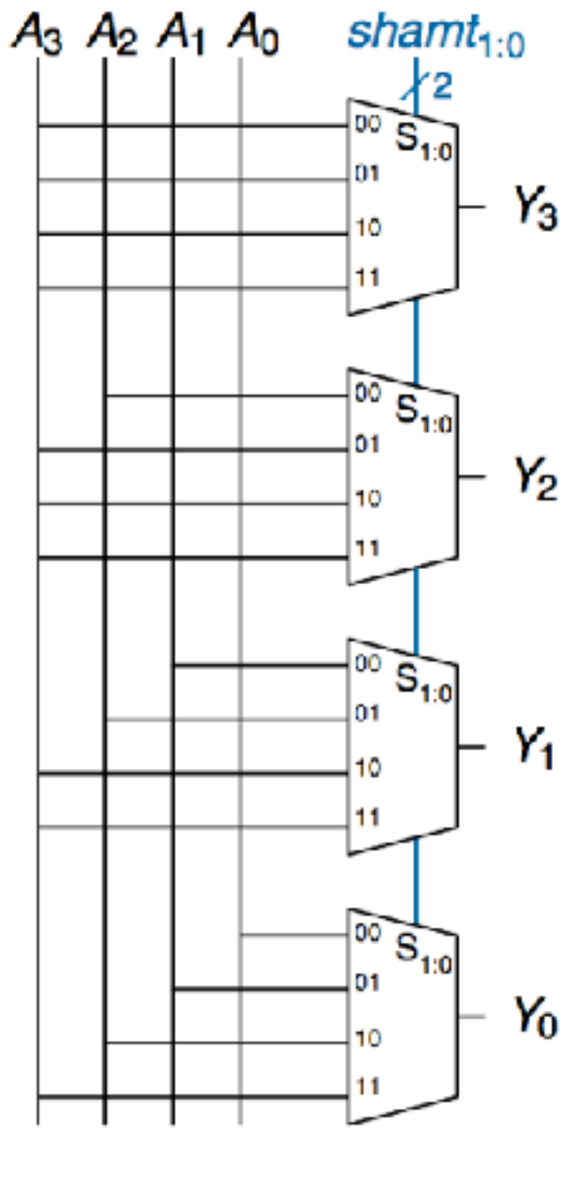
Logical Shift Right



Arithmetic Shift Right







Circuitos Aritméticos

Desplazadores y Rotadores

- Un desplazamiento a la izquierda es un caso particular de multiplicación. Ej: 000011<< 4 = 110000 es equivalente a $3 \times 2^4 = 48$.
- Un desplazamiento aritmético a la derecha es un caso particular de división. Ej: 11100 >>> 2 = 11111, es equivalente a $-4 / 2^2 = -1$.

Circuitos Aritméticos

Multiplicación

- La multiplicación de números binarios sin signo es similar a la multiplicación decimal considerando sólo 1s y 0s.

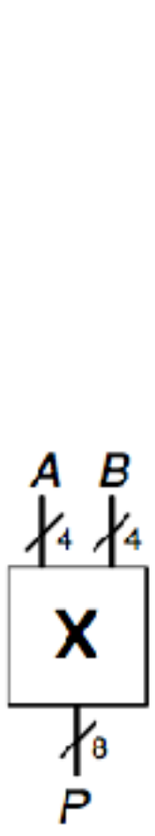
230	multiplicand	0101
× 42	multiplier	× 0111
460	partial products	0101
+ 920		0101
9660		0101
		+ 0000
	result	0100011

$2\,30 \times 42 = 9660$ $\qquad$ $5 \times 7 = 35$

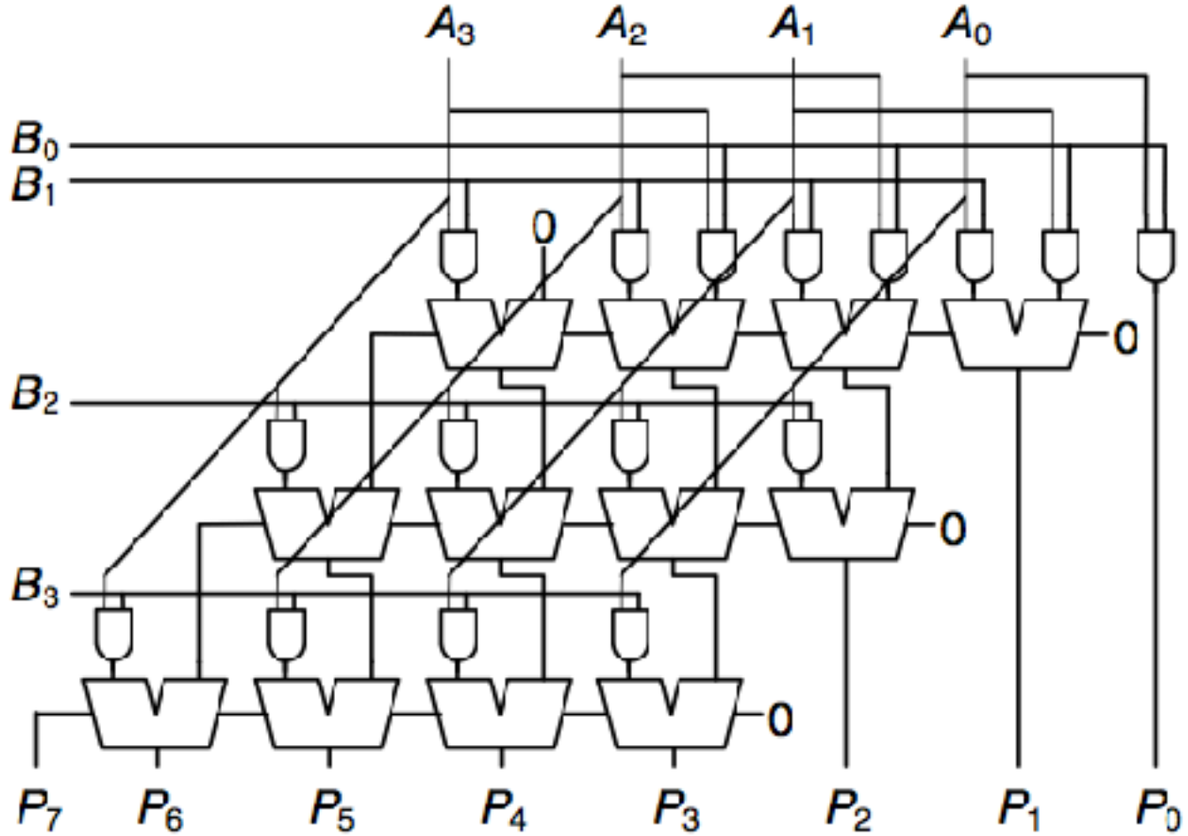
Circuitos Aritméticos

Multiplicación

- multiplicador 4 X 4



$$
\begin{array}{cccccccc}
 & & & A_3 & A_2 & A_1 & A_0 & \\
 & \times & & B_3 & B_2 & B_1 & B_0 & \\
\hline
 & & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 & & \\
 & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & & & \\
A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & & & & \\
+ \quad A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 & & & & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
$$

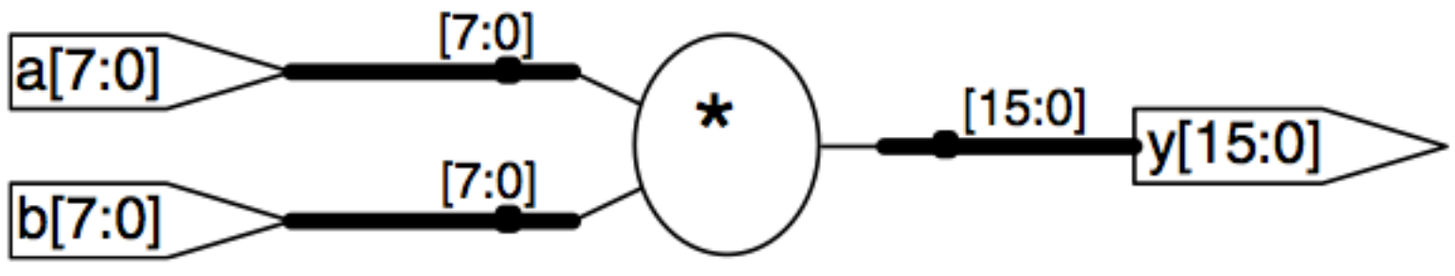


Circuitos Aritméticos

Multiplicación

```
module multiplier # (parameter N = 8)
                    (input  [N-1:0]   a, b,
                     output [2*N-1:0] y);

  assign y = a * b;
endmodule
```



Circuitos Aritméticos

División

- La división binaria puede realizarse utilizando el siguiente algoritmo para números normalizados sin signo, en el rango $[2^N-1, 2^{N-1}]$
- El resto parcial R, se inicializa al valor del dividendo A.
- El divisor B, es restado repetidamente del resto parcial para determinar si cabe.

```
R = A
for i = N−1 to 0
  D = R − B
  if D < 0 then    Qi = 0, R′ = R   // R < B
  else             Qi = 1, R′ = D   // R ≥ B
  if i ≠ 0 then R = 2R′
```

Circuitos Aritméticos

División

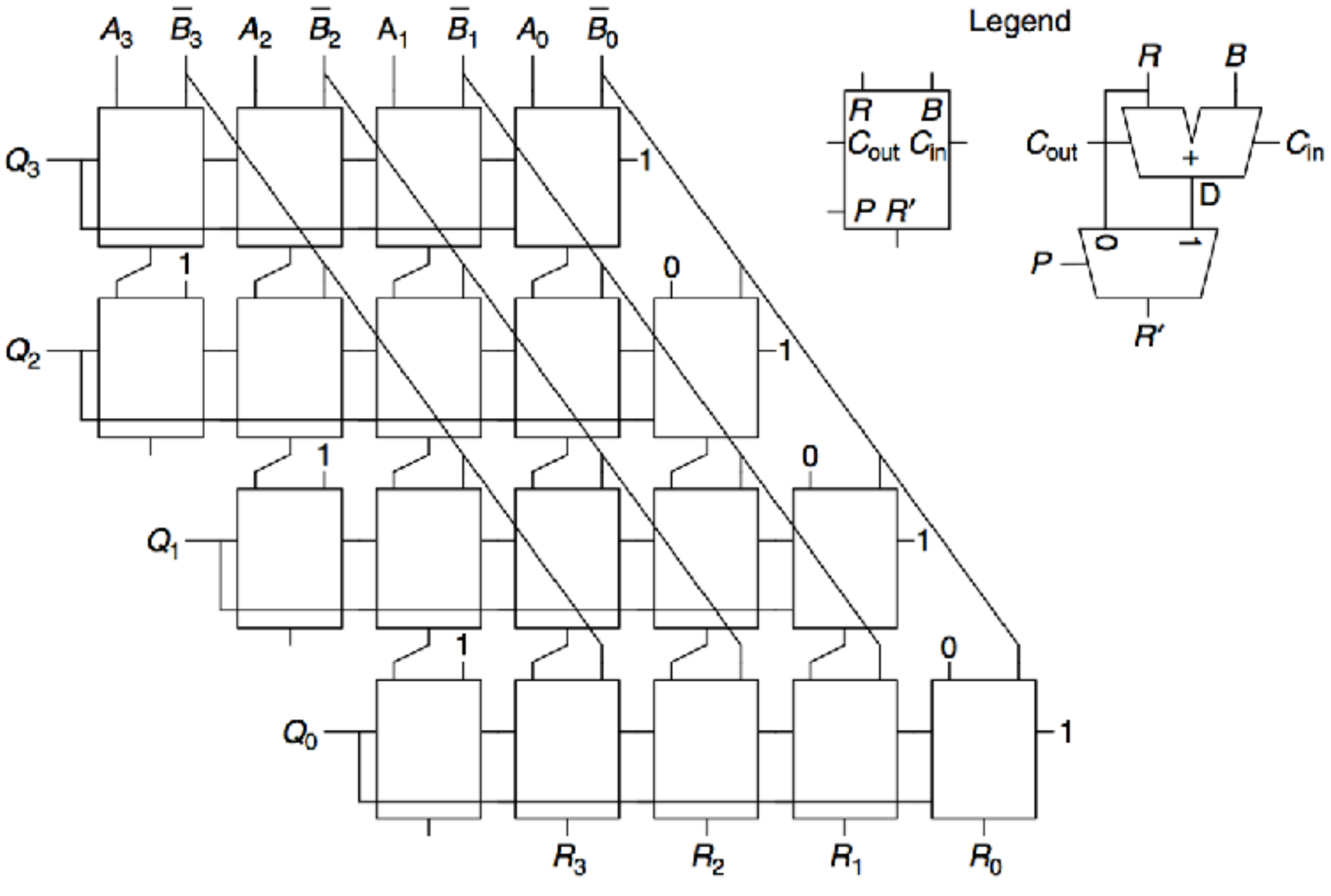
- Si la diferencia D es negativa, entonces el bit del cuociente Q_i es 0 y la diferencia es descartada.
- De lo contrario, Q_i es 1 y el resto parcial es actualizado con el valor de la diferencia.
- En cualquiera de los dos casos, inmediatamente después, el resto parcial es multiplicado por dos y el proceso se repite.
- El resultado satisface

$$\frac{A}{B} = \left(Q + \frac{R}{B}\right) 2^{-(N-1)}.$$

Circuitos Aritméticos

División

- Arreglo divisor

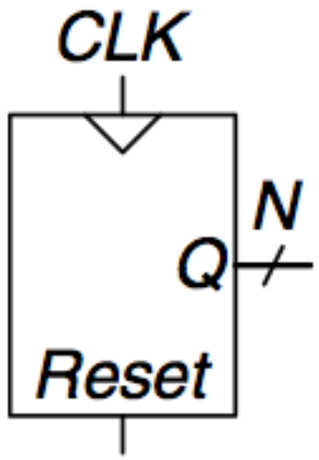


Bloques Secuenciales

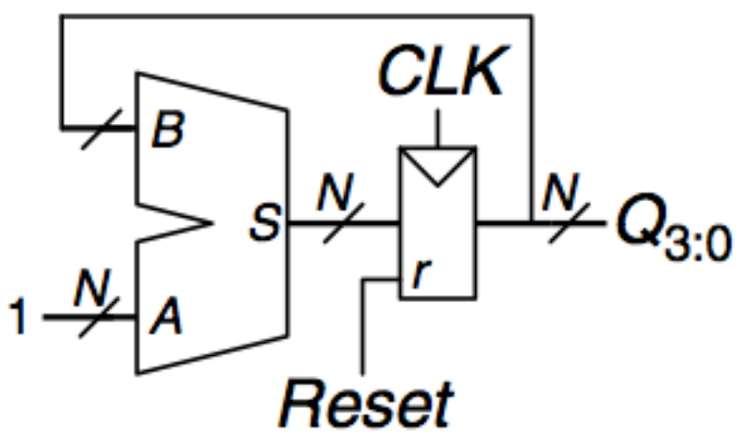
Contadores

- Contador de N bits

Símbolo



Posible Circuito

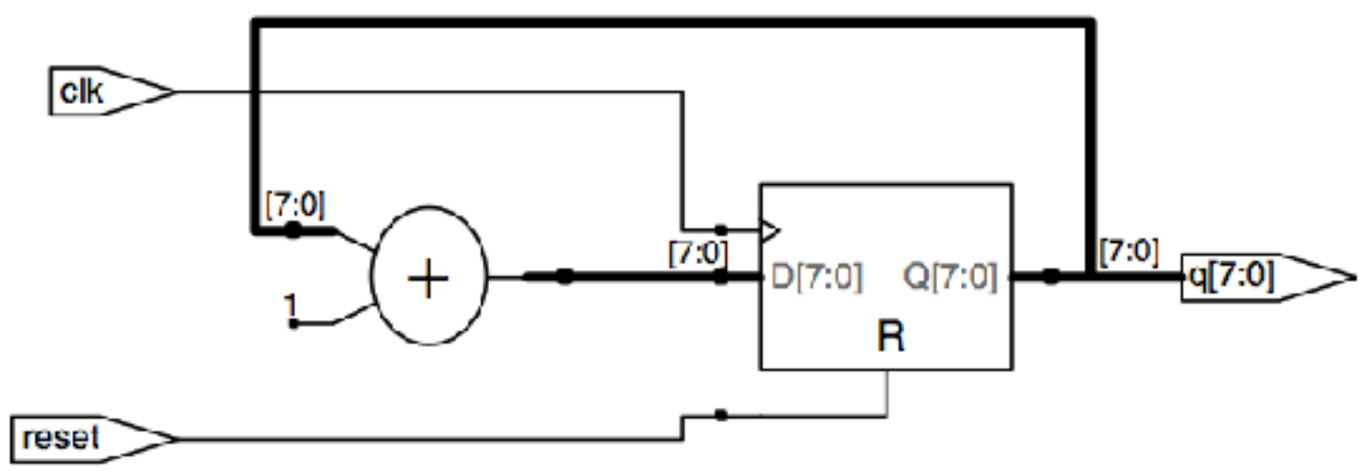


Descripción Verilog

```
module counter #(parameter N = 8)
                (input                clk,
                 input                reset,
                 output reg [N-1:0] q);

  always @ (posedge clk or posedge reset)
    if (reset) q <= 0;
    else       q <= q + 1;
endmodule
```

Circuito Sintetizado

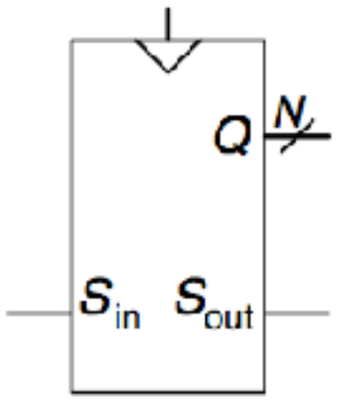


Bloques Secuenciales

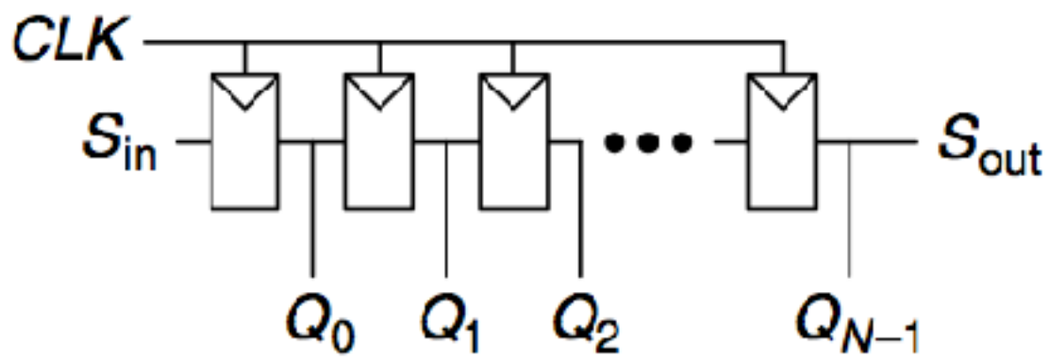
Registros de Desplazamiento

- Registro de desplazamiento de N bits

Símbolo

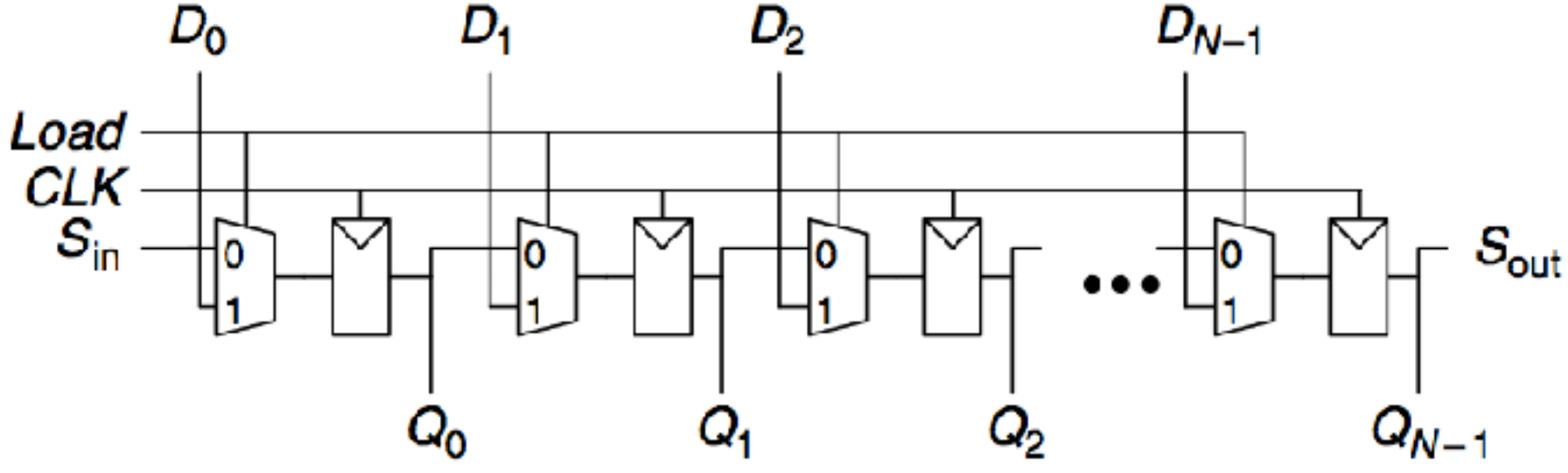


Circuito



- Registro de desplazamiento de N bits con carga paralela

Circuito



Bloques Secuenciales

Registros de Desplazamiento

- Descripción Verilog para el Registro de desplazamiento de N bits con Carga Paralela.

```
module shiftreg # (parameter N = 8)
                  (input                clk,
                   input                reset, load,
                   input                sin,
                   input       [N-1:0]  d,
                   output reg  [N-1:0]  q,
                   output               sout);

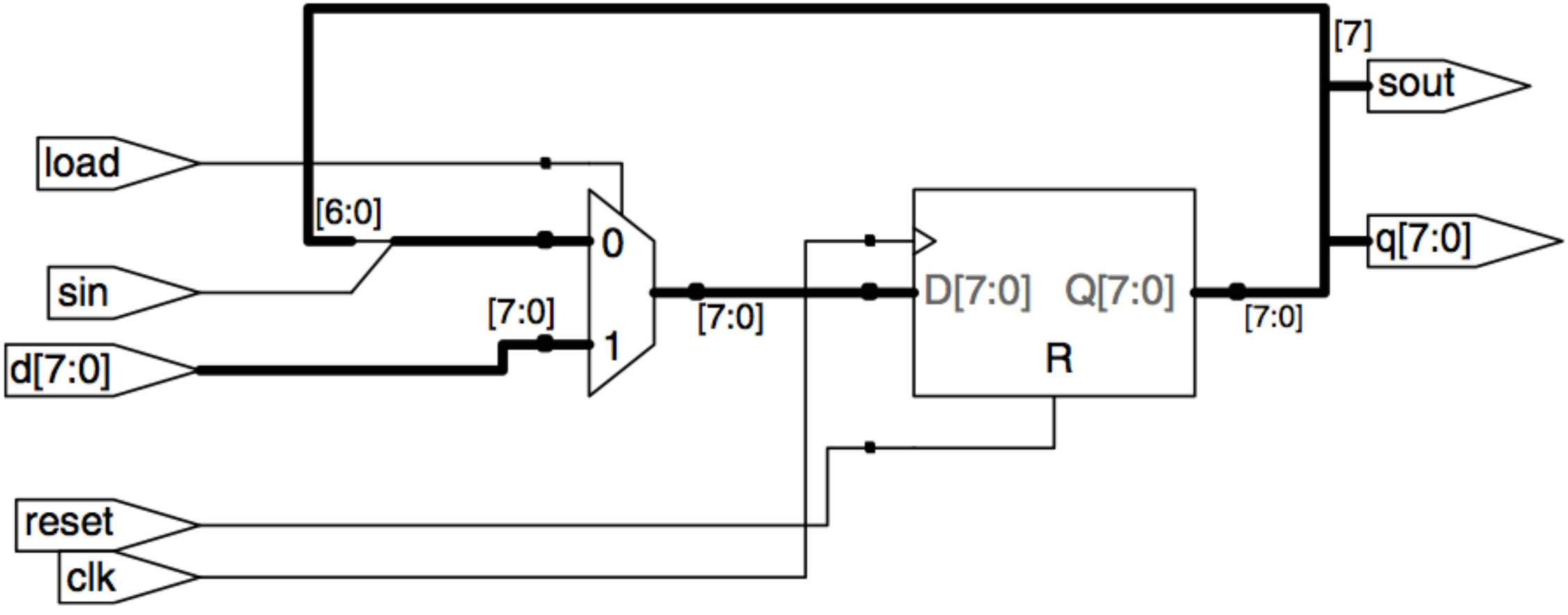
  always @ (posedge clk or posedge reset)
    if (reset)       q <= 0;
    else if (load)   q <= d;
    else             q <= {q[N-2:0], sin};

  assign sout = q[N-1];
endmodule
```

Bloques Secuenciales

Registros de Desplazamiento

- Circuito Sintetizado a partir de la Descripción Verilog para el Registro de desplazamiento de N bits con Carga Paralela.

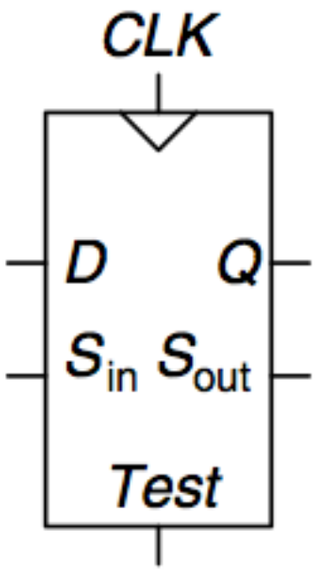


Bloques Secuenciales

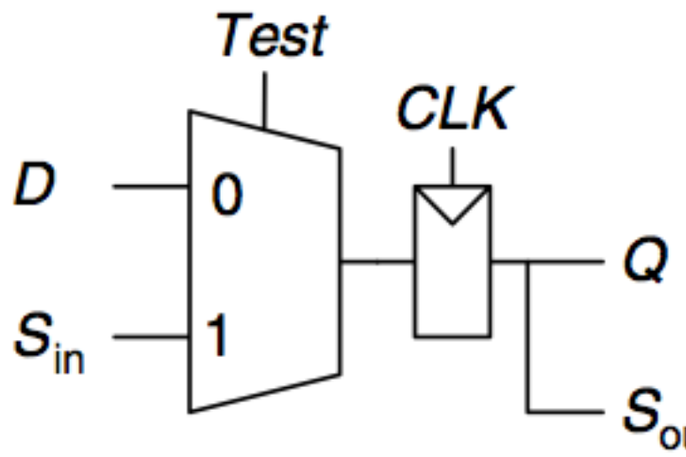
Registros de Desplazamiento

- Flip-Flops "Escaneables"

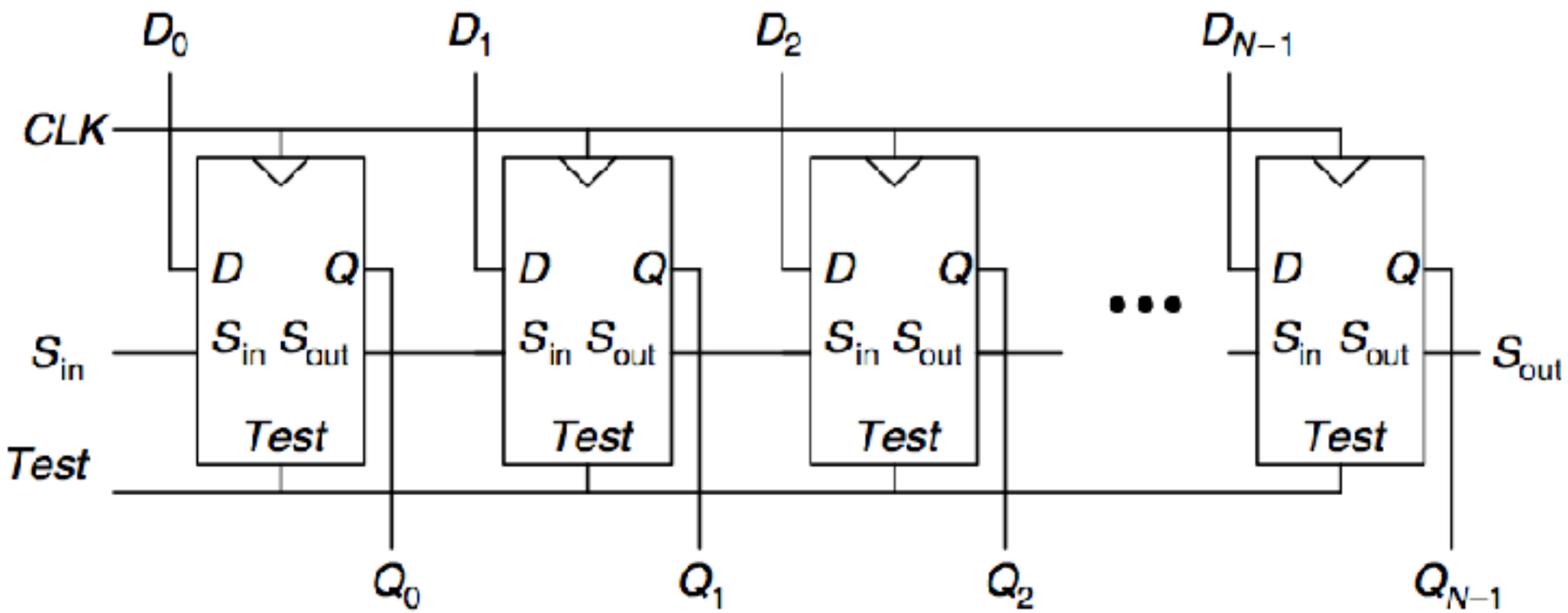
Símbolo



Circuito



Conexión



Bloques Secuenciales

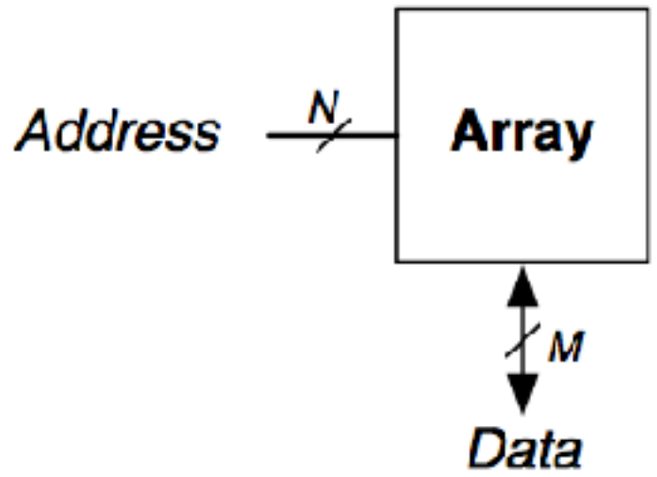
Memorias

- Características compartidas de los arreglos de memoria
- Memorias dinámicas (DRAM)
- Memorias estáticas (SRAM)
- Memorias sólo de lectura (ROM)

Bloques Secuenciales

Memorias

- Símbolo genérico

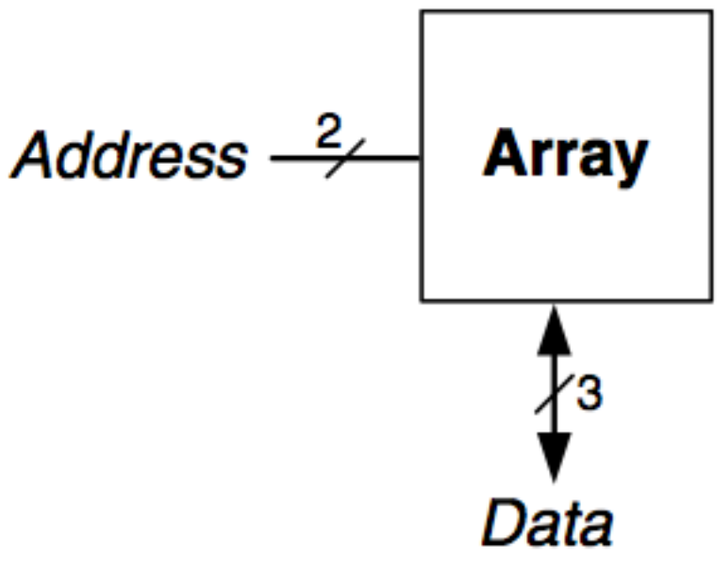


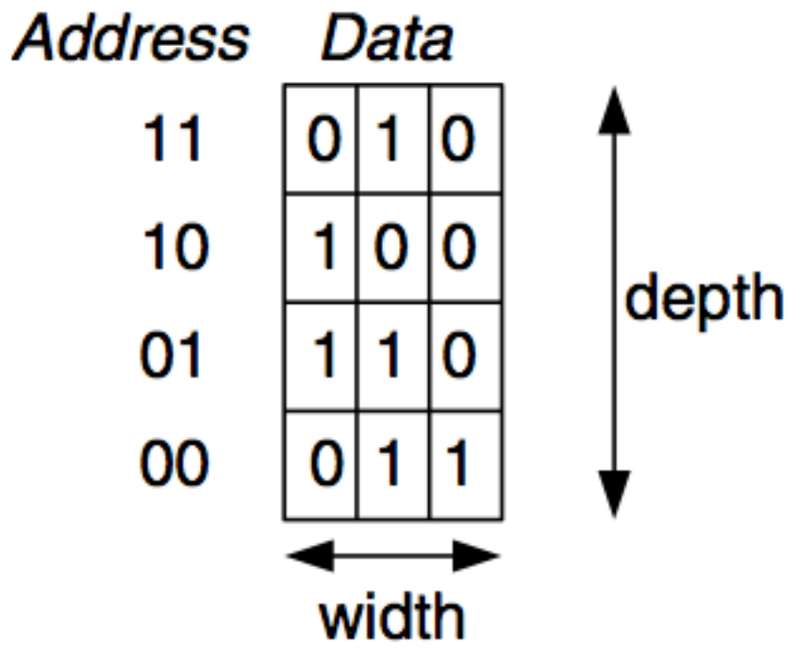
- Una memoria está organizada en la forma de un arreglo bidimensional de celdas de memoria.
- Una memoria lee o escribe el contenido de una de las líneas del arreglo.
- La línea del arreglo a leer o escribir se especifica por una dirección.
- El valor a leer o escribir se llama dato.

Bloques Secuenciales

Memorias

- Un arreglo con *N*-bits de direccionamiento y datos de *M*-bits tiene 2^N filas y *M* columnas.
- Cada fila se llama “palabra”





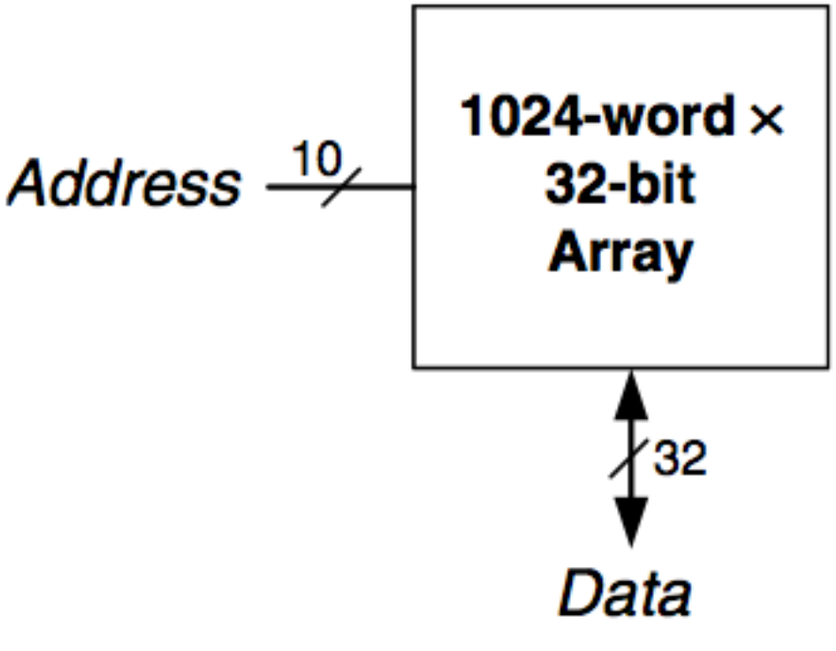
Bloques Secuenciales

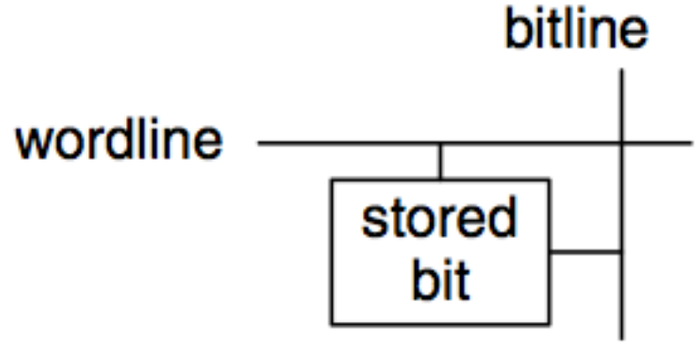
Memorias

- Los arreglos de memoria se construyen como un arreglo de celdas, cada una de las cuales almacena un bit del dato.

- Para leer una celda, la línea de bit se pone en Z y la línea de palabra en 1.

- Para escribir una celda la línea de bit se pone en el valor a escribir y la línea de palabra en 1.

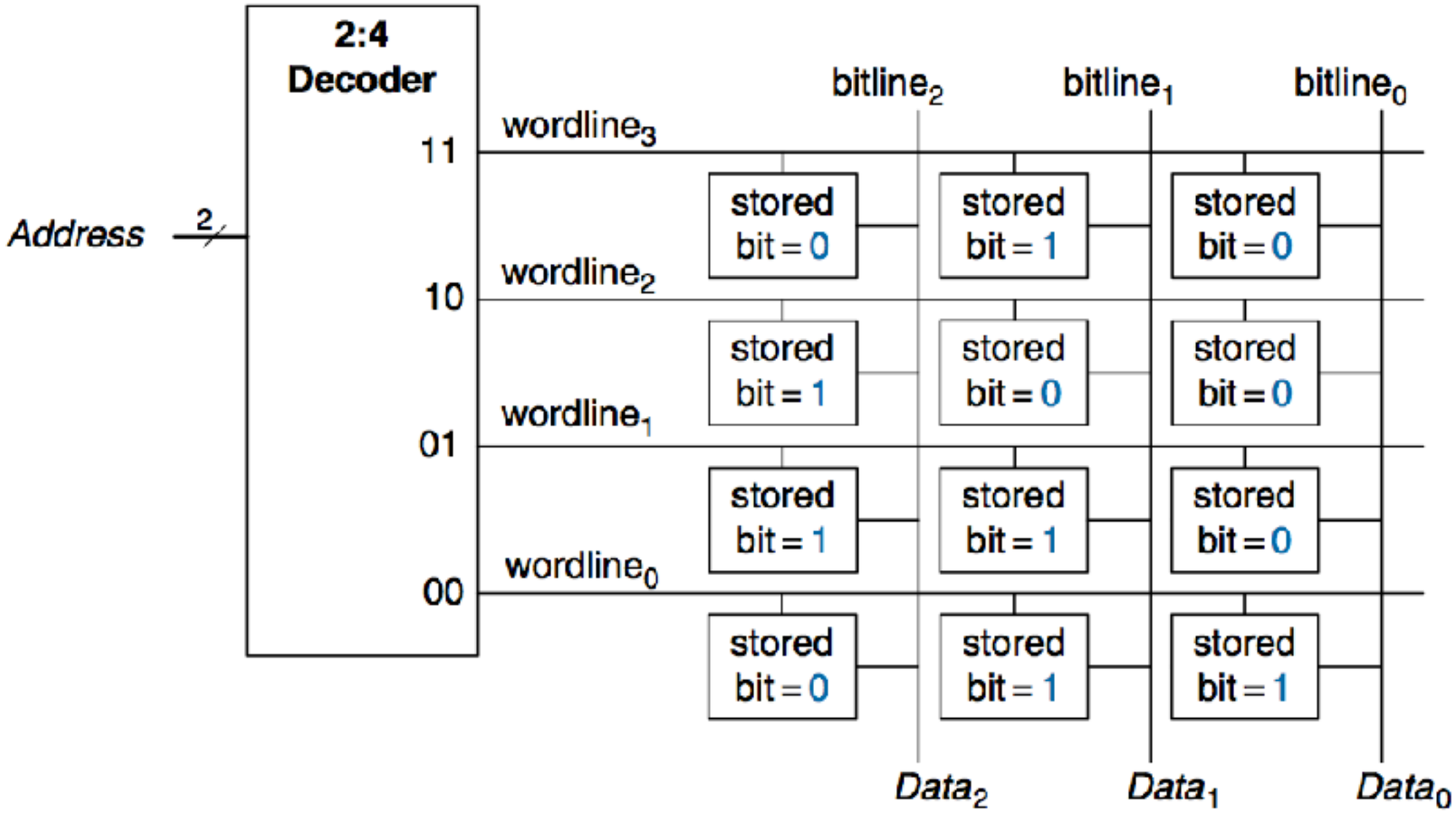




Bloques Secuenciales

Memorias

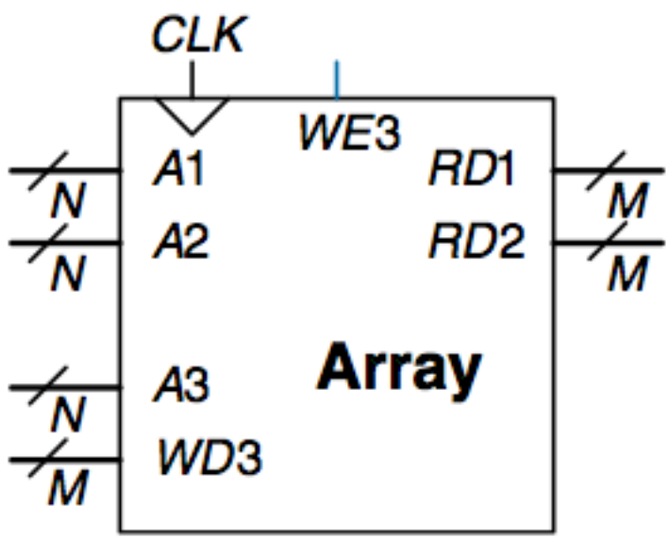
- Arreglo de 4 x 3.



Bloques Secuenciales

Memorias

- Los arreglos que hemos visto son de un puerto.
- Existen memorias multi-puerto.
- Pueden accesar varias direcciones simultáneamente.



Bloques Secuenciales

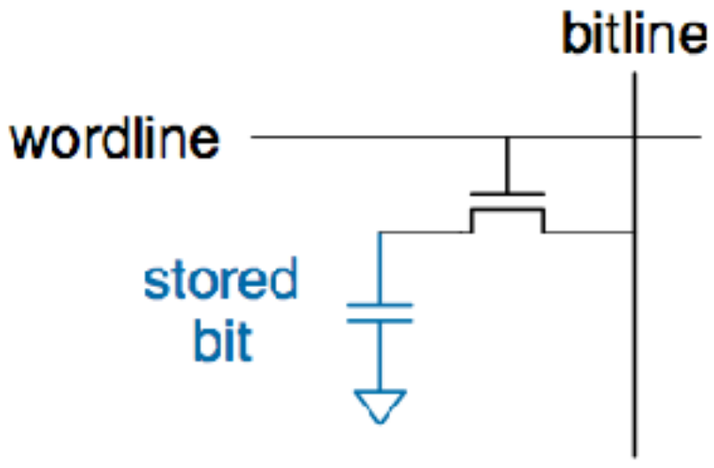
Tipos de Memorias

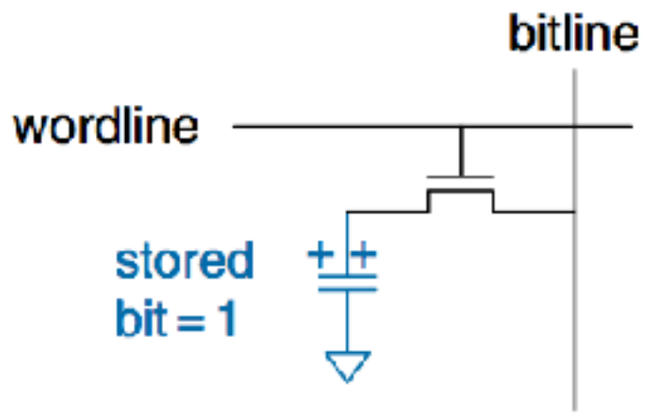
- La clasificación más amplia es Memorias de Acceso Aleatorio (RAM) versus Memorias Sólo de Lectura (ROM).
- Las RAM son volátiles (pierden su contenido cuando se apagan).
- Las ROM son no volátiles (retienen el contenido para siempre).
- Las RAM y las ROM recibieron su nombre por razones históricas que hoy no son tan válidas.
- Dentro de la clasificación de memorias RAM existen las RAM dinámicas y las RAM estáticas.

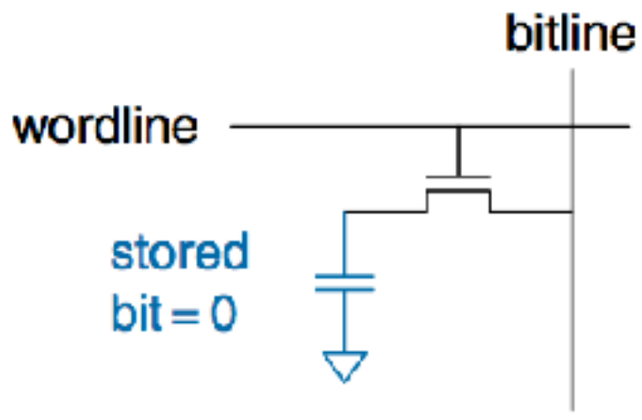
Bloques Secuenciales

Tipos de Memorias

- Memorias RAM dinámicas



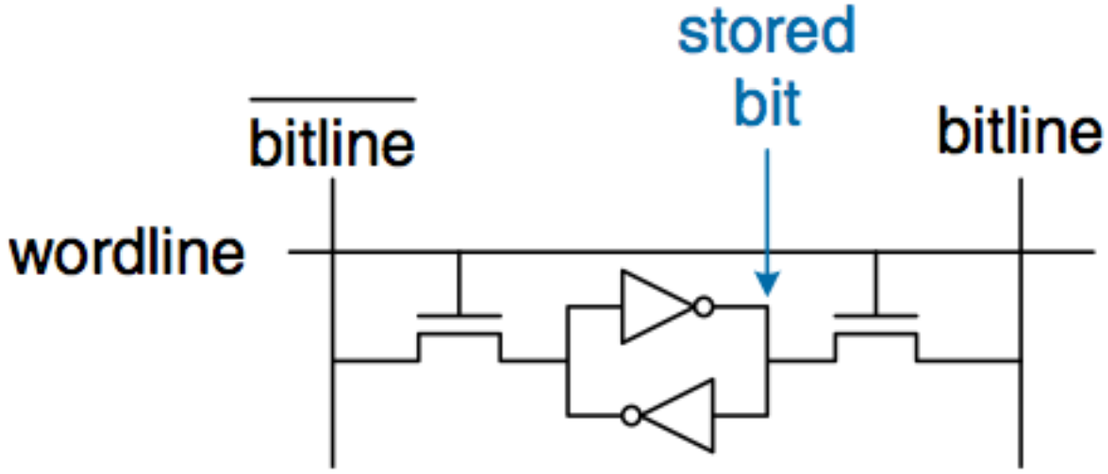




Bloques Secuenciales

Tipos de Memorias

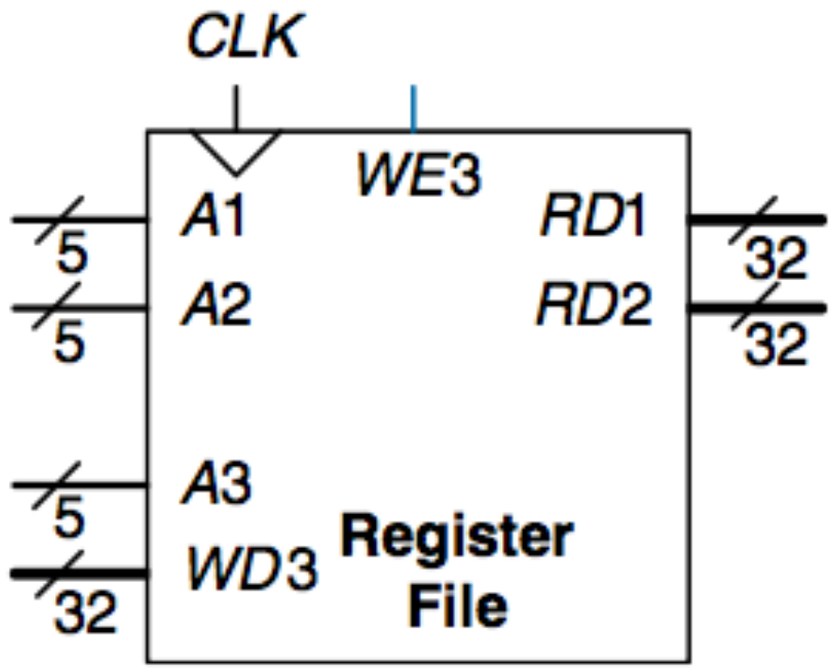
- Memorias RAM estáticas



Bloques Secuenciales

Tipos de Memorias

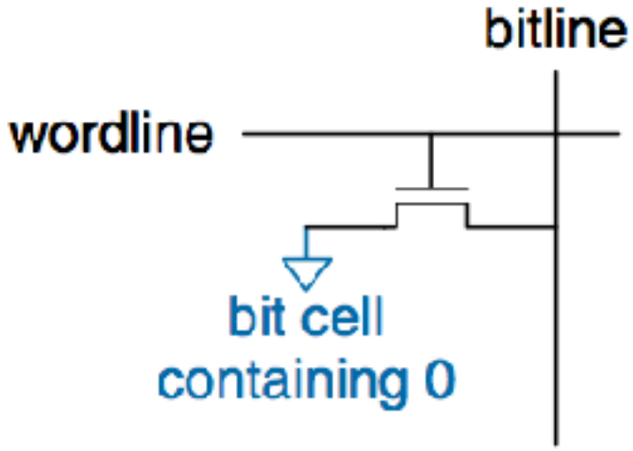
- Memorias RAM estáticas.
- Arreglo de registros

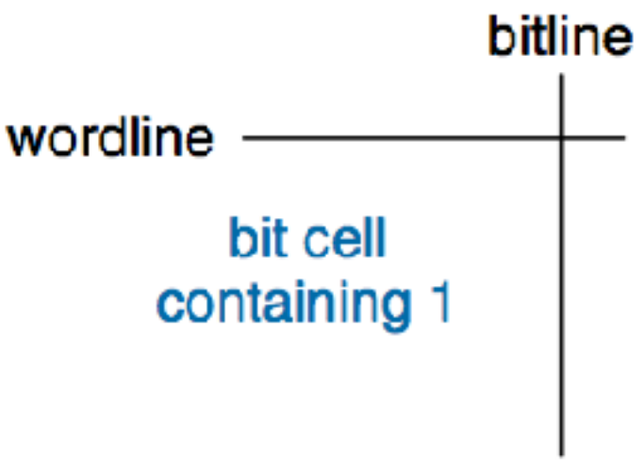


Bloques Secuenciales

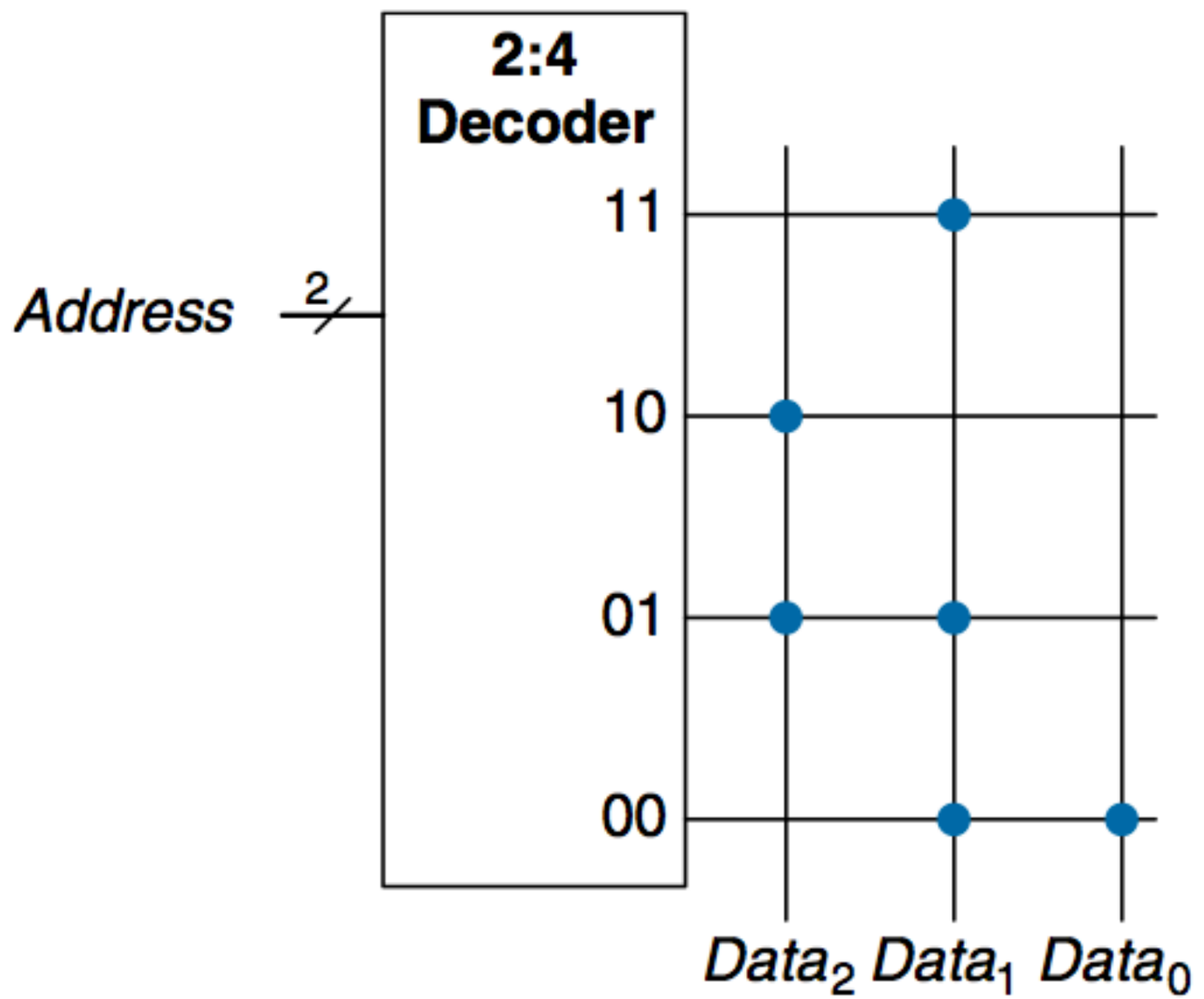
Tipos de Memorias

- Memorias ROM





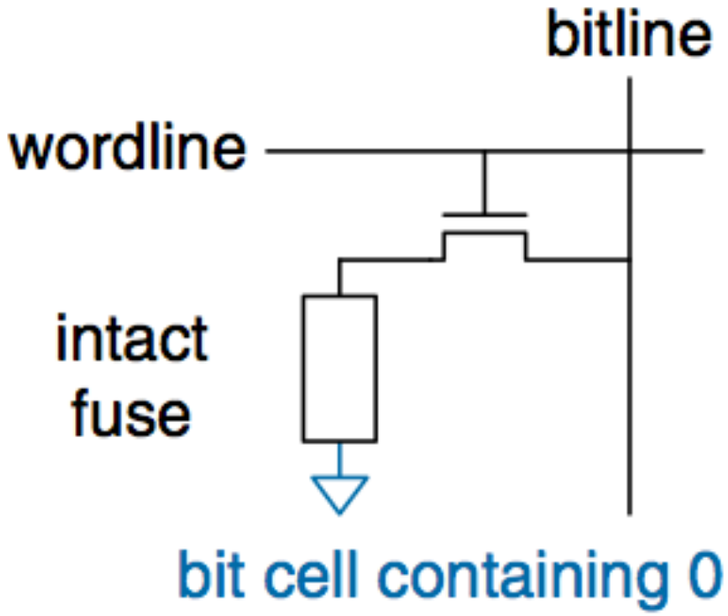
Notación *dot*.

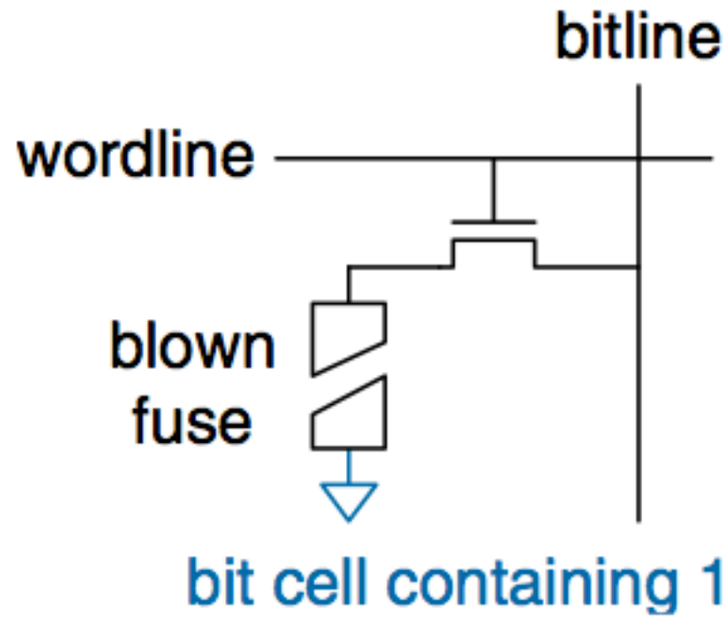


Bloques Secuenciales

Tipos de Memorias

- Memorias ROM programables una sola vez



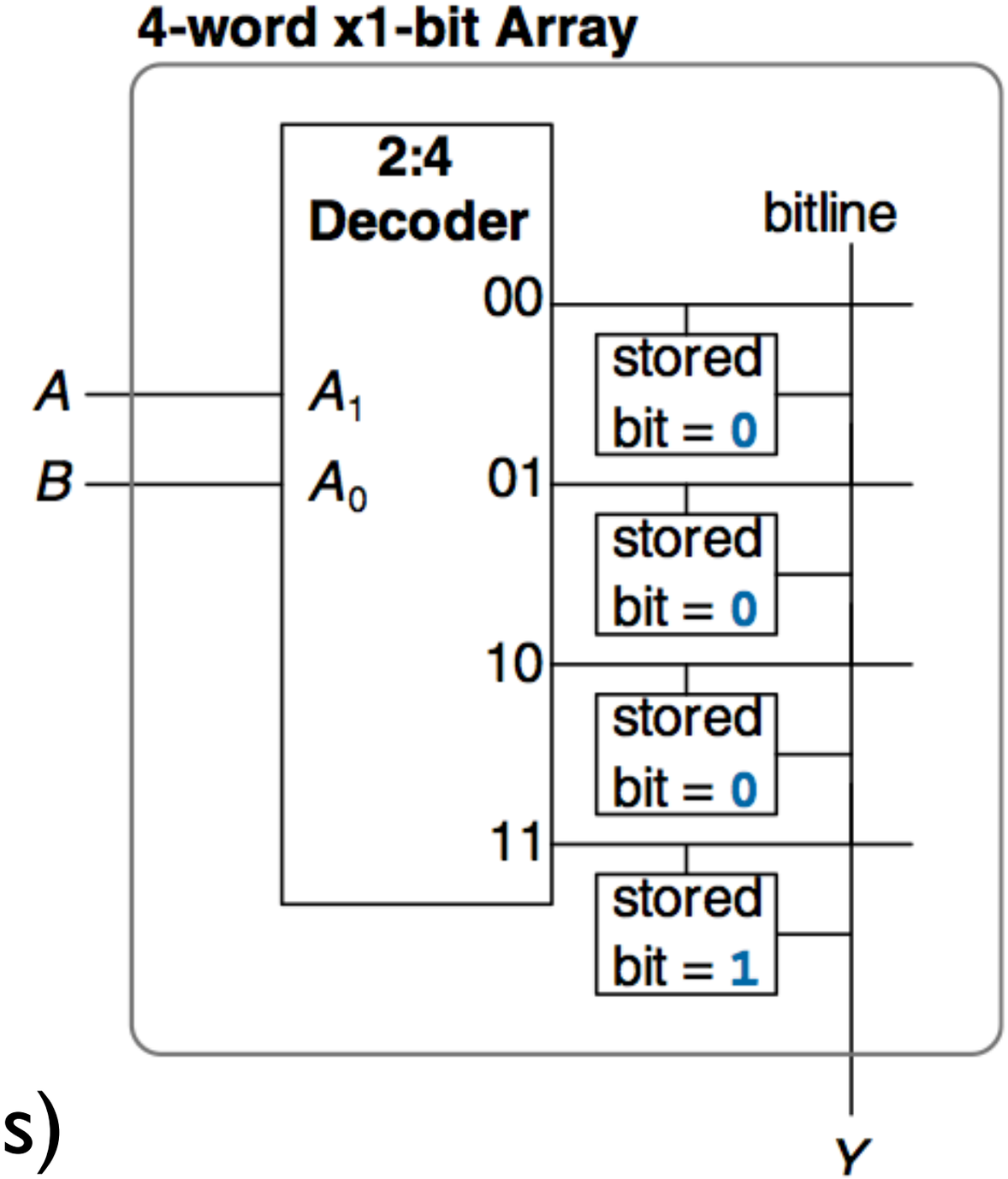


Bloques Secuenciales

Funciones Lógicas utilizando Memorias

Truth Table

A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



LUT (Look Up Tables)

Bloques Secuenciales

Código Verilog para especificar una memoria

Código

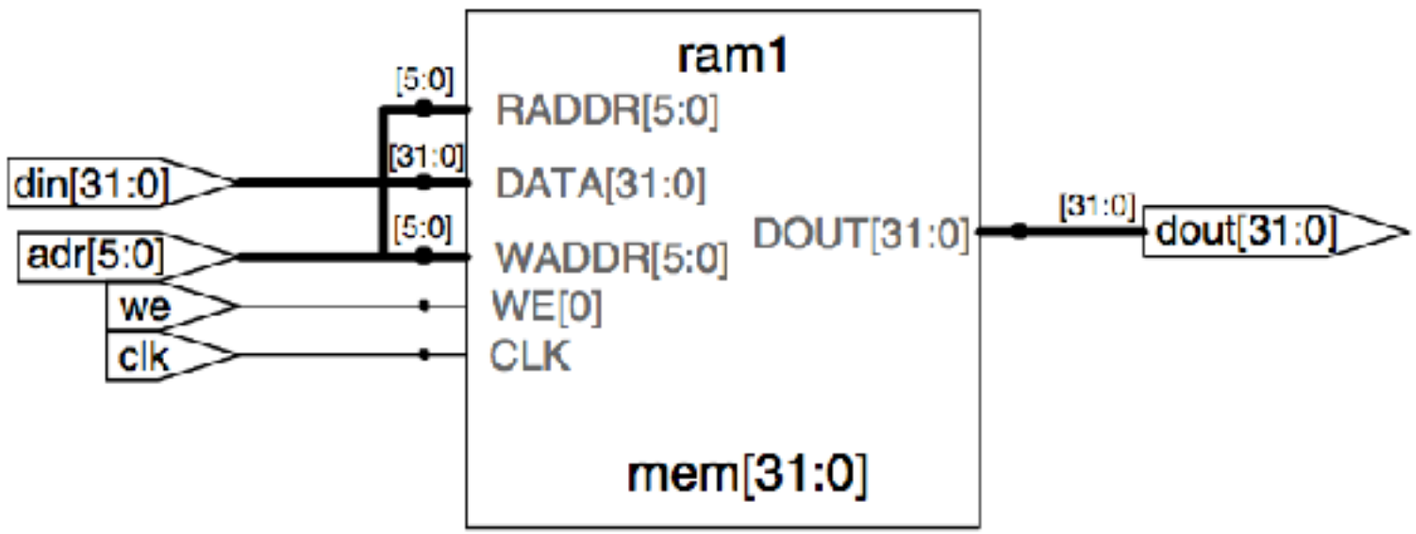
```
module ram # (parameter N = 6, M = 32)
                  (input             clk,
                   input             we,
                   input  [N-1:0]    adr,
                   input  [M-1:0]    din,
                   output [M-1:0]    dout);

  reg [M-1:0] mem [2**N-1:0];

  always @ (posedge clk)
    if (we) mem [adr] <= din;

  assign dout = mem[adr];
endmodule
```

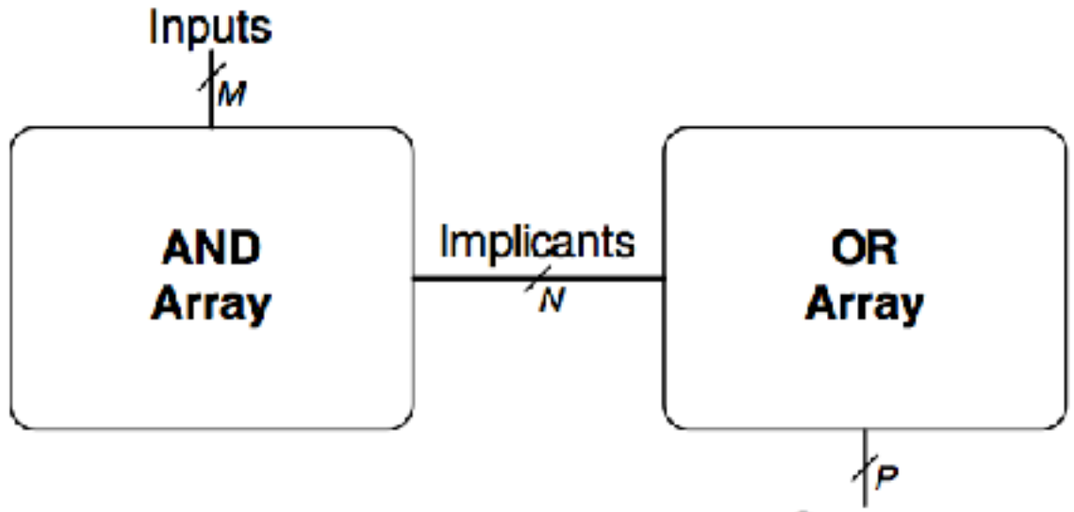
Síntesis



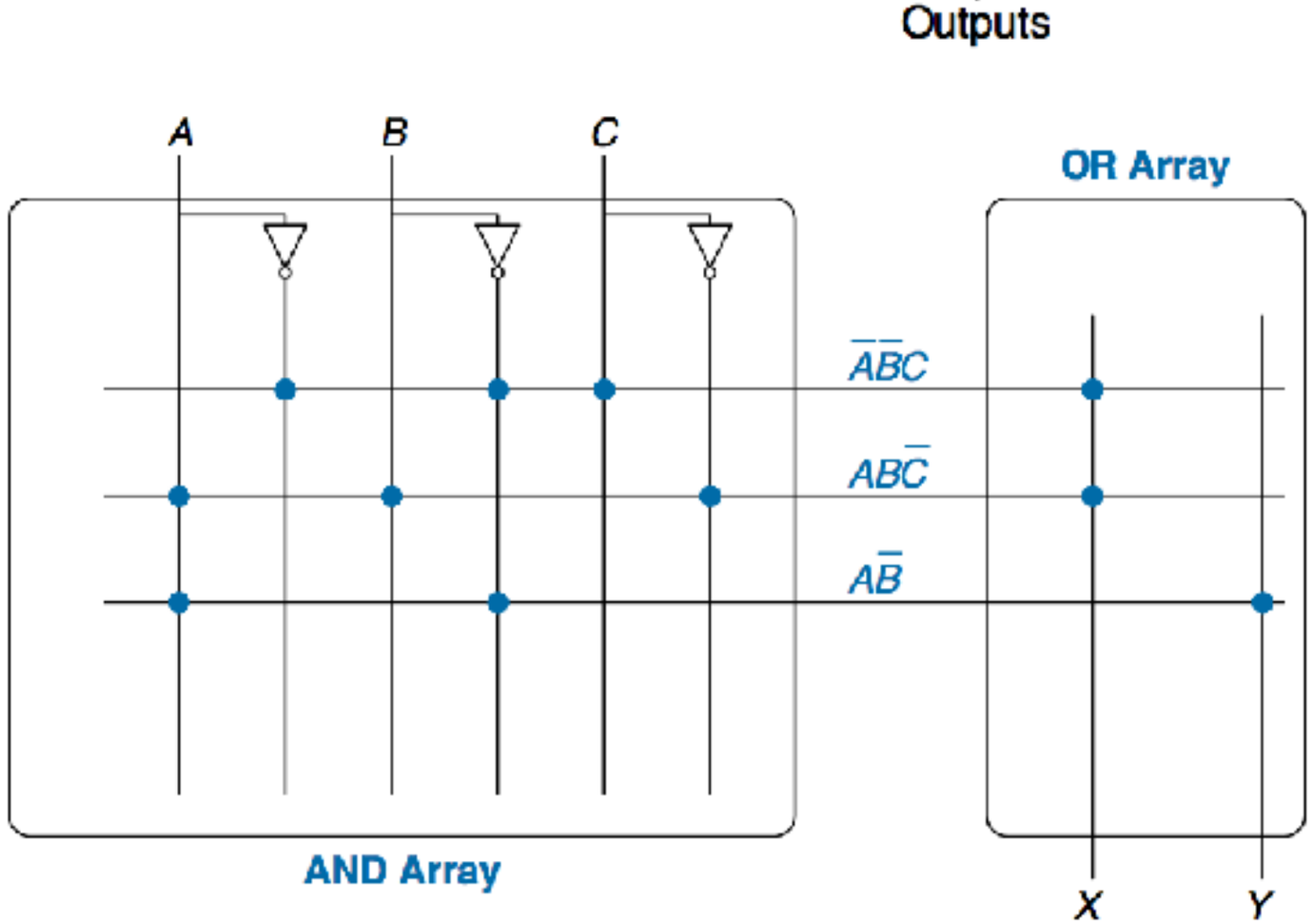
Bloques Secuenciales

Arreglos Lógicos Programables (PLAs)

Esquema general

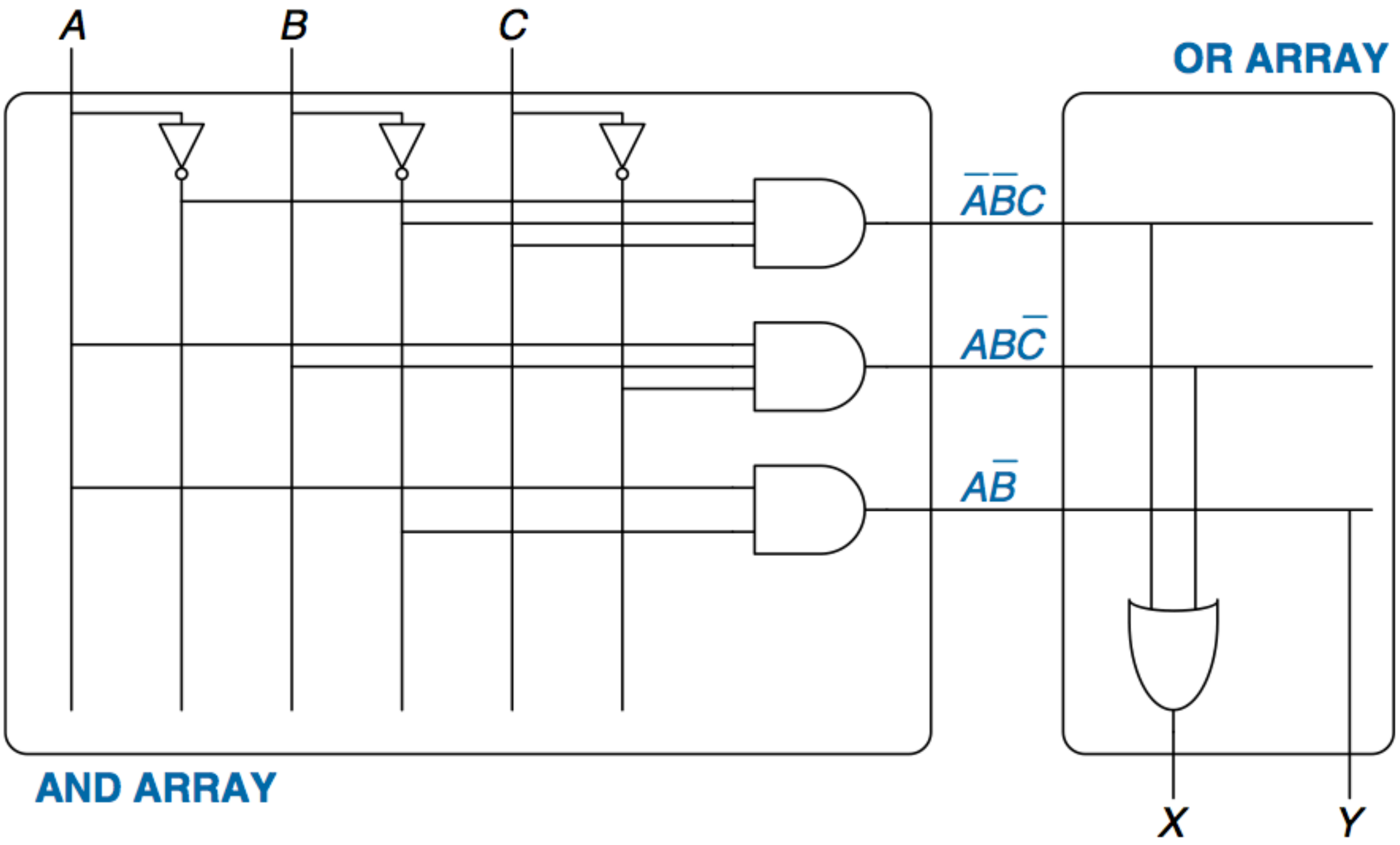


Notación *dot*



Bloques Secuenciales

Arreglos Lógicos Programables (PLAs)

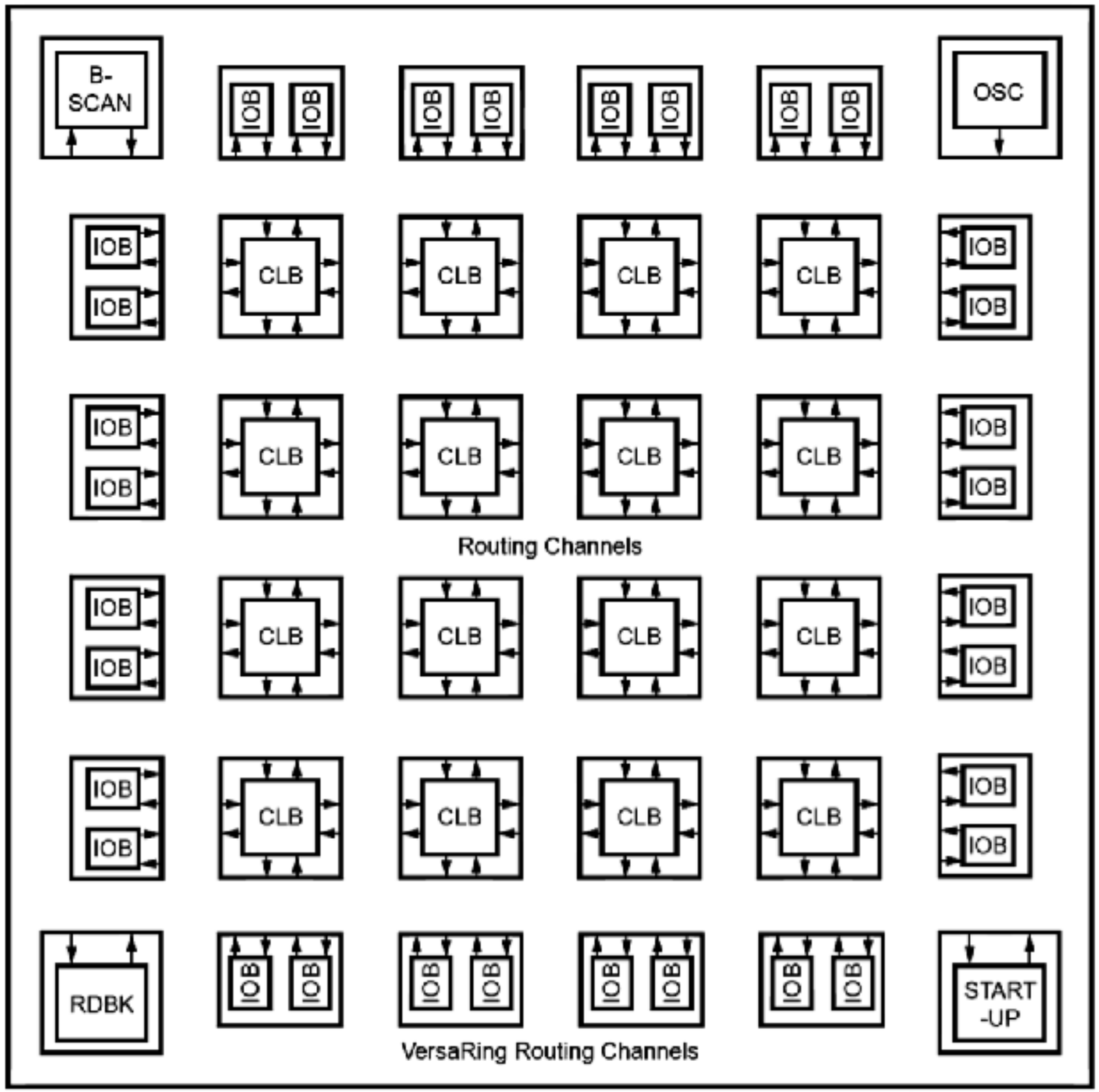


Bloques Secuenciales

FPGA
(Field
Programmable
Gate Array)

Spartan

CLB:
Configurable
Logic Block

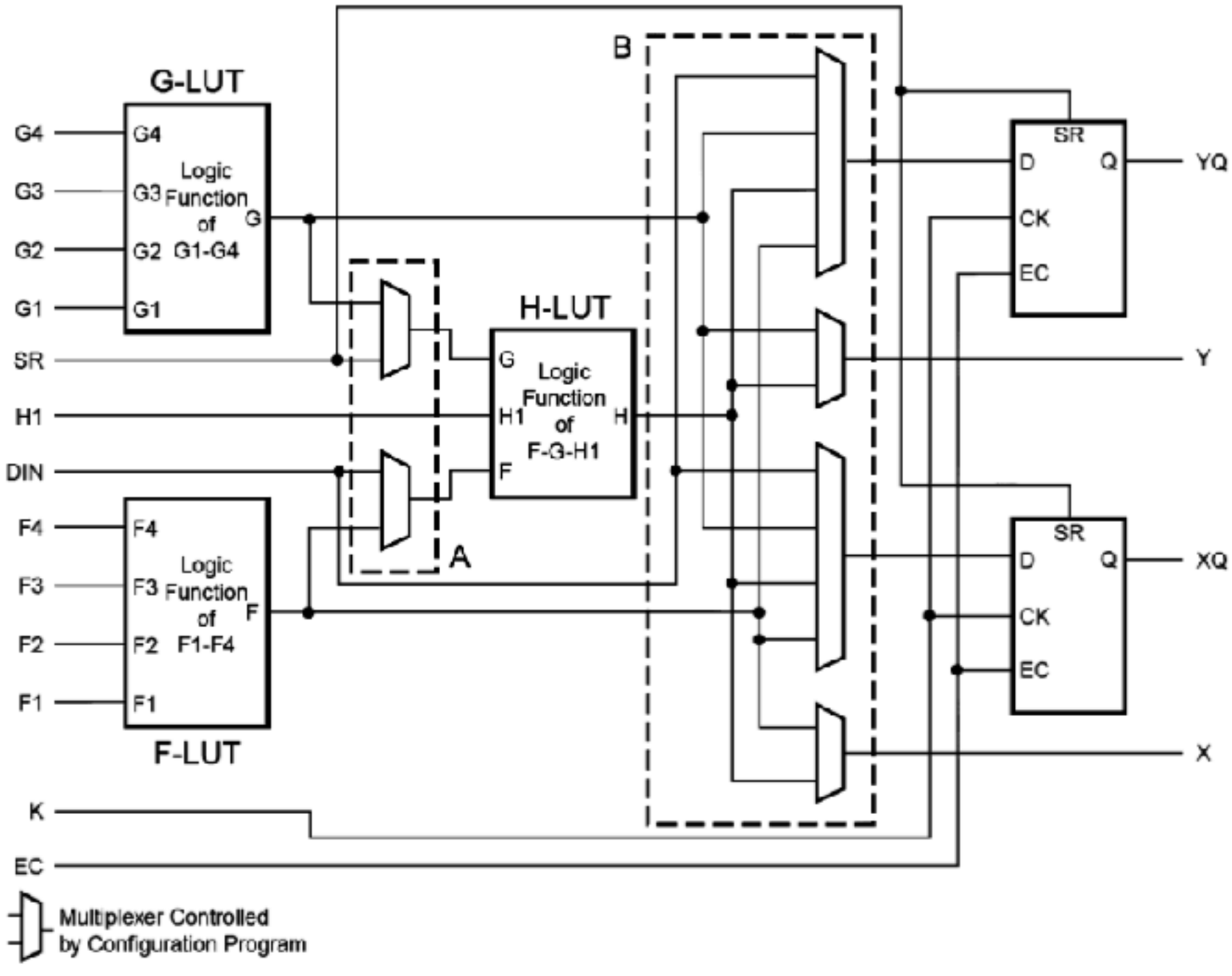


Bloques Secuenciales

FPGA

Spartan

CLB



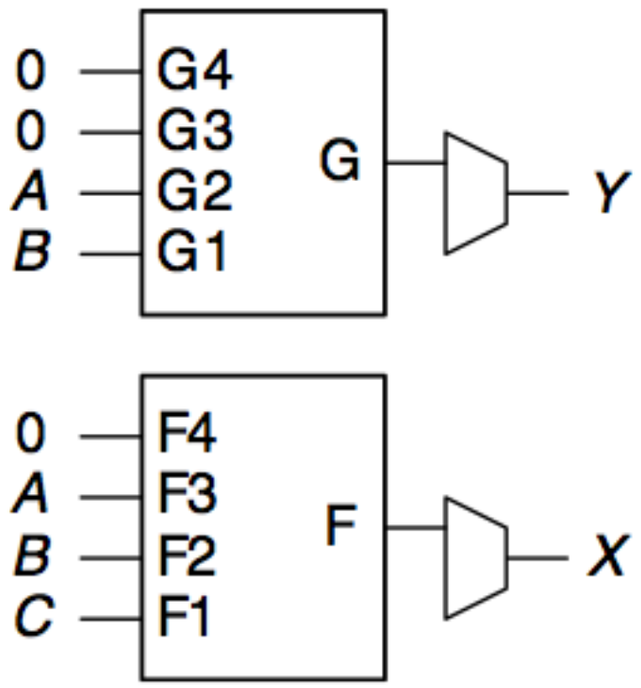
Bloques Secuenciales

FPGA Spartan CLB

- La configuración del CLB permite implementar dos funciones de hasta 4 entradas c/u.

	(A)	(B)	(C)	(X)
F4	F3	F2	F1	F
X	0	0	0	0
X	0	0	1	1
X	0	1	0	0
X	0	1	1	0
X	1	0	0	0
X	1	0	1	0
X	1	1	0	1
X	1	1	1	0

		(A)	(B)	(Y)
G4	G3	G2	G1	G
X	X	0	0	0
X	X	0	1	0
X	X	1	0	1
X	X	1	1	0



Bloques Secuenciales

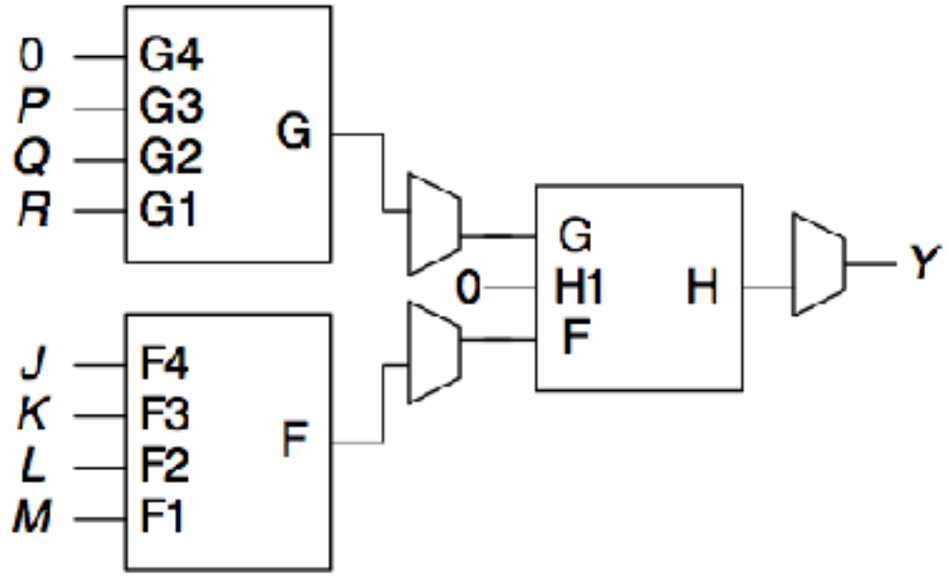
FPGA Spartan CLB

- La configuración del CLB permite implementar una función de más de 4 entradas.

(J) F4	(K) F3	(L) F2	(M) F1	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

G4	(P) G3	(Q) G2	(R) G1	G
X	0	0	0	0
X	0	0	1	0
X	0	1	0	0
X	0	1	1	0
X	1	0	0	0
X	1	0	1	0
X	1	1	0	0
X	1	1	1	1

H1	F	G	(Y) H
X	0	0	0
X	0	1	0
X	1	0	0
X	1	1	1



Bloques Secuenciales

FPGA Spartan CLB

- Permite además configurar una máquina de estados finitos con dos bits por estado.

F4	F3	(S_1) F2	(S_0) F1	(S_0') F
X	X	0	0	1
X	X	0	1	0
X	X	1	0	0
X	X	1	1	0

G4	G3	(S_1) G2	(S_0) G1	(S_1') G
X	X	0	0	0
X	X	0	1	1
X	X	1	0	0
X	X	1	1	0

