Circuitos Aritméticos

Para qué queremos circuitos aritméticos ?.

- Implementación de funciones aritméticas para el procesamiento digital de datos.

Qué operaciones son las más comunes?.

- Es variado pero por ejemplo para el diseño de filtros digitales es muy común el empleo de sólo sumas y multiplicaciones en punto fijo y flotante.
- Para esos casos y aplicándolas en circuitos lógicos programables, las FPGA son diseñadas para lograr ese cometido.

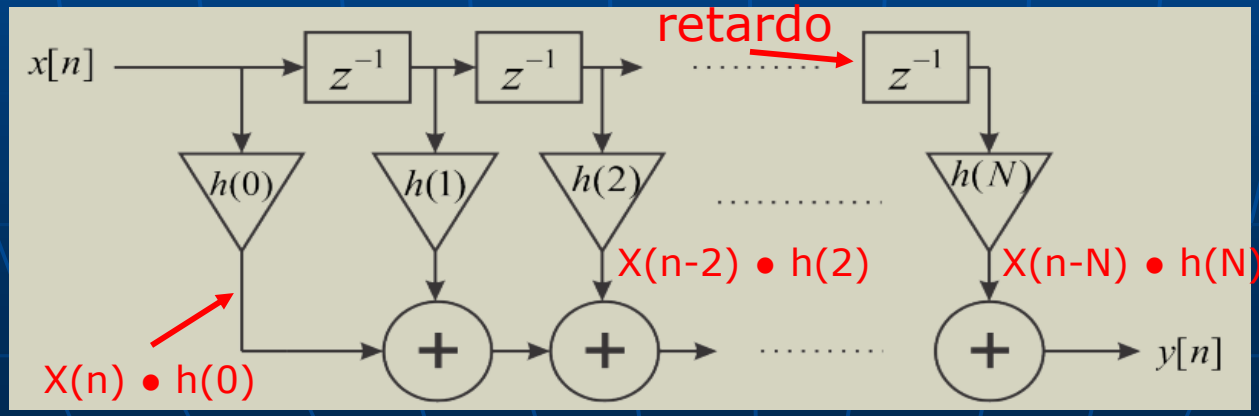
Ejemplo de diseño de Filtros de Respuesta Impulsiva (FIR)

- Función de transferencia general:

$$y(n)=\sum_{k=0}^{N} h(k)\,x(n-k)$$

La salida y(n) es función de la entrada x(n) y de la función de transferencia del filtro h.

- Esquema básico de implementación con sumadores y multiplicadores:



Ejemplo:
X y h de 8 bits.
y de 12 bits.

CIRCUITOS ARITMÉTICOS

Clasificación según función:

- Sumadores.
- Restadores.
- Multiplicadores.
- Divisores.

Clasificación según formato:

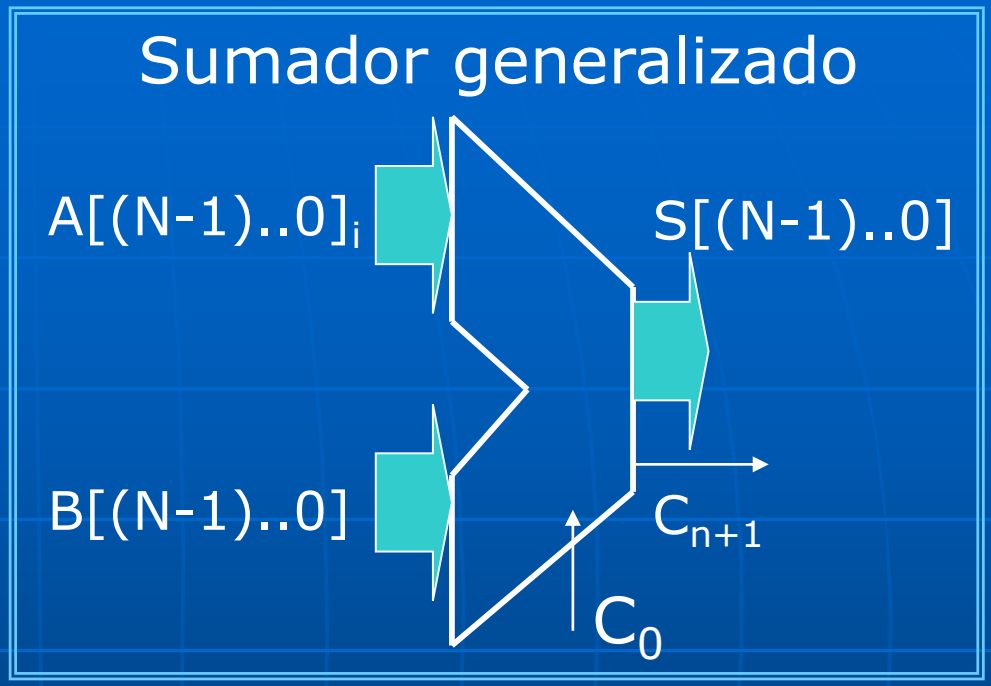
Paralelo: Mas complejo con mayor consumo de hardware pero generalmente muy rápido al procesar parcialmente funciones en forma simultánea.

Serie: Mas simple, con mejor optimización de recursos de hardware pero lento con latencias que dependen de la extensión de bits a procesar.

Procesamiento paralelo

Sumador

- Ripple-carry
- Look-ahead carry
- Carry-save
- Carry-select



El tipo de sumador de n bits a elegir depende de la forma en que se procesa el acarreo (carry) de un bit hacia la otra posición mas significativa (de [i] a [i+1]).
Existen para la selección situaciones de compromiso (trade-off) entre simplicidad circuital, velocidad de respuesta, consumo de energía y disponibilidad estructural (en el caso de circuitos lógicos programables).

Sumador Ripple-carry (sin signo)

Es la topología mas simple pero que posee la menor velocidad de respuesta ya que el bit de suma en cada posición de bit depende de los carry anteriores por lo que se genera un efecto de retardo acumulativo que será mayor cuanto mayor sea la cantidad de bits que tenga el sumador.

Para solucionar esto, existen estructuras alternativas como las de look-ahead carry, carry-save, etc.

Sumador Ripple-carry (sin signo)

SEMI-SUMADOR DE UN BIT (HALF-ADDER)

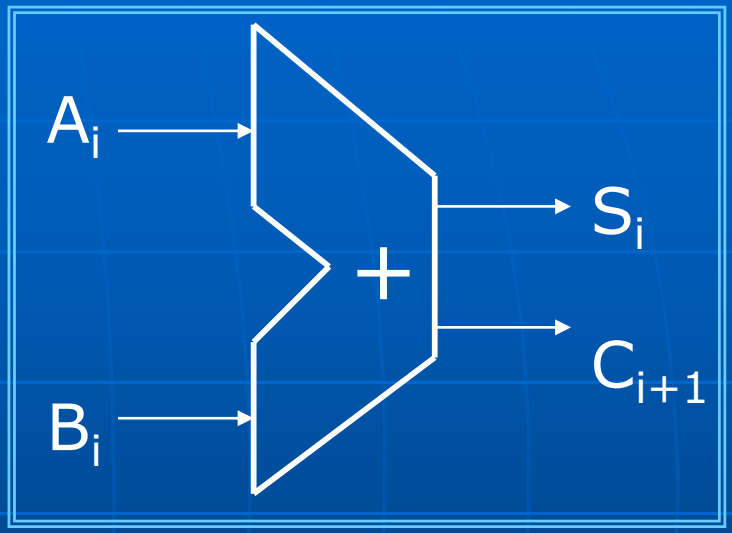
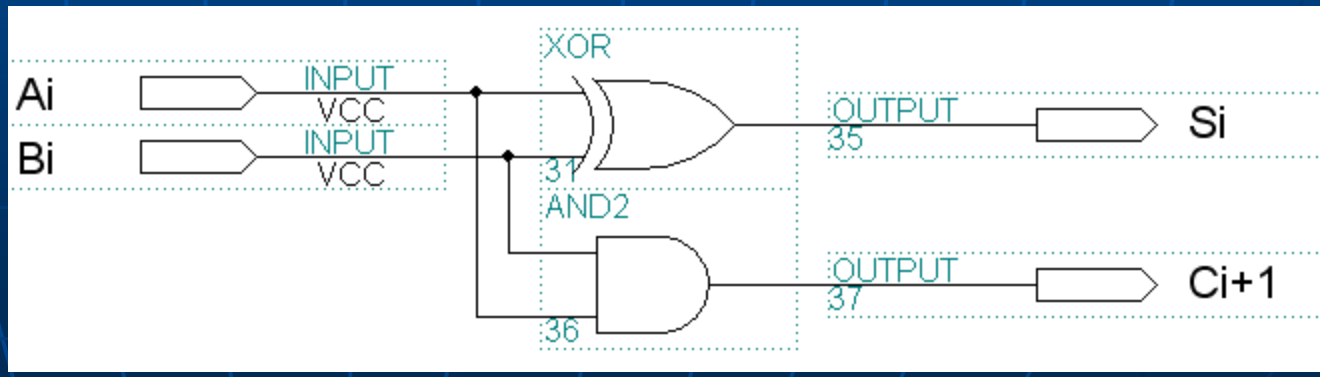


Tabla de verdad

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$Si = Ai \oplus Bi \quad ; \quad Ci = Ai \bullet Bi$



Sumador Ripple-carry (sin signo)

SUMADOR COMPLETO DE UN BIT (FULL-ADDER)

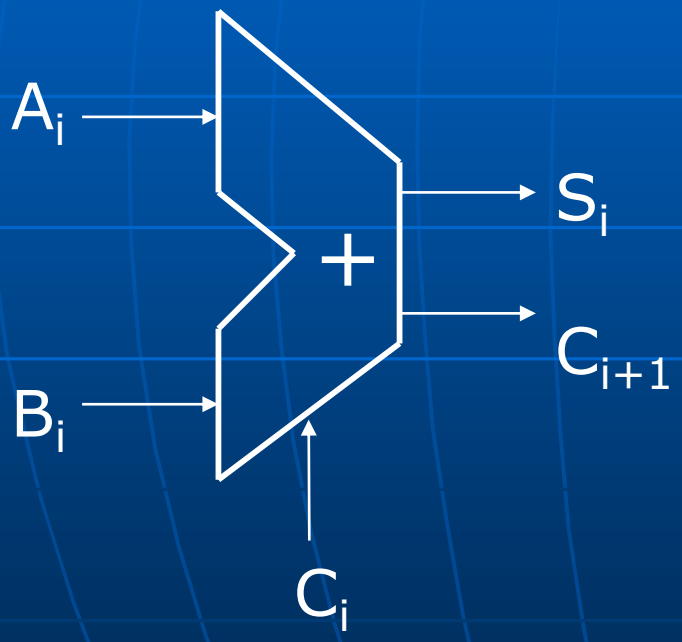


Tabla de verdad

C_i	A_i	B_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sumador Ripple-carry (sin signo)

SUMADOR COMPLETO (FULL-ADDER) DE UN BIT

C_i \ AB	$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}_i$	0 (0)	1 (1)	0 (3)	1 (2)
C_i	1 (4)	0 (5)	1 (7)	0 (6)

$$S_i = A \oplus B \oplus C_i$$

ó

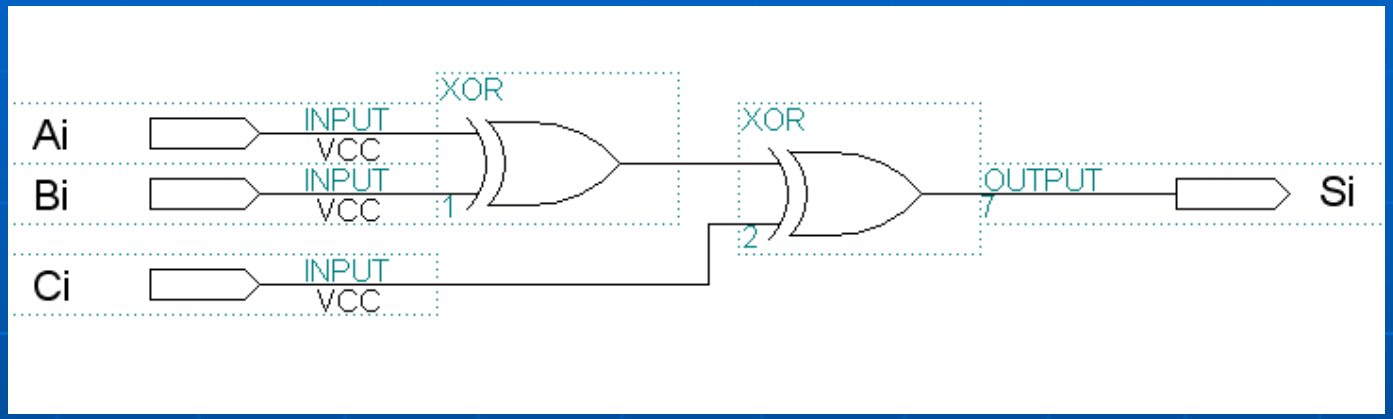
$$S_i = /A\ B\ /C_i + A\ /B\ /C_i + /A\ /B\ C_i + A\ B\ C_i$$

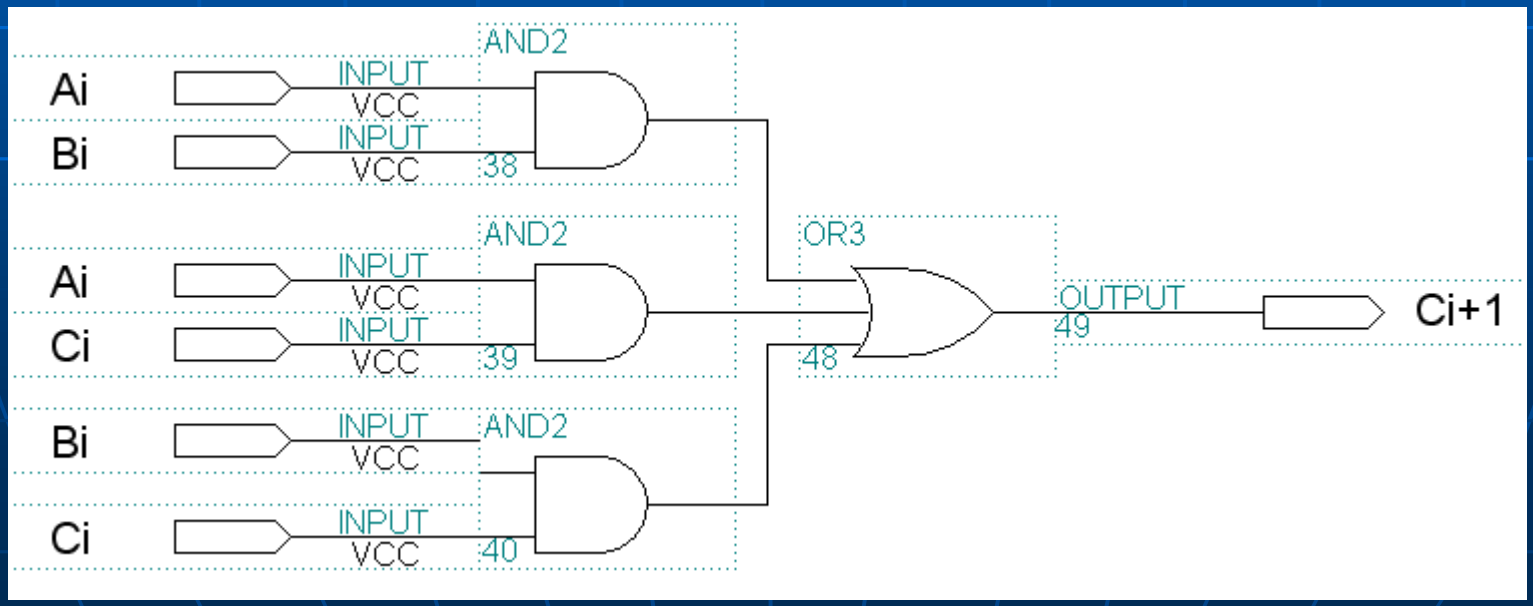
C_i \ AB	$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}_i$	0 (0)	0 (1)	1 (3)	0 (2)
C_i	0 (4)	1 (5)	1 (7)	1 (6)

$$C_{i+1} = A\ B + A\ C_i + B\ C_i$$

Sumador Ripple-carry (sin signo)

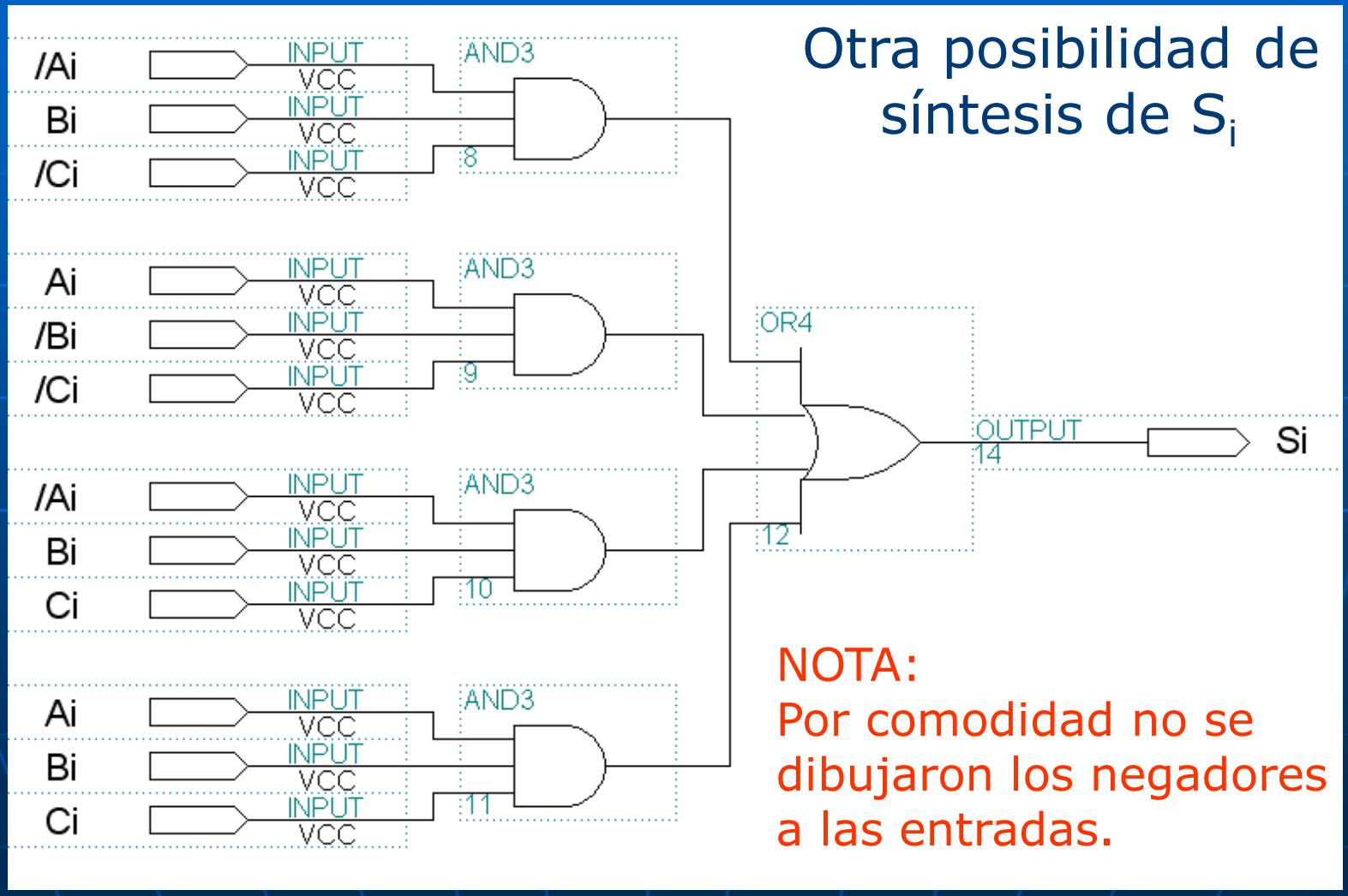
SUMADOR COMPLETO (FULL-ADDER)





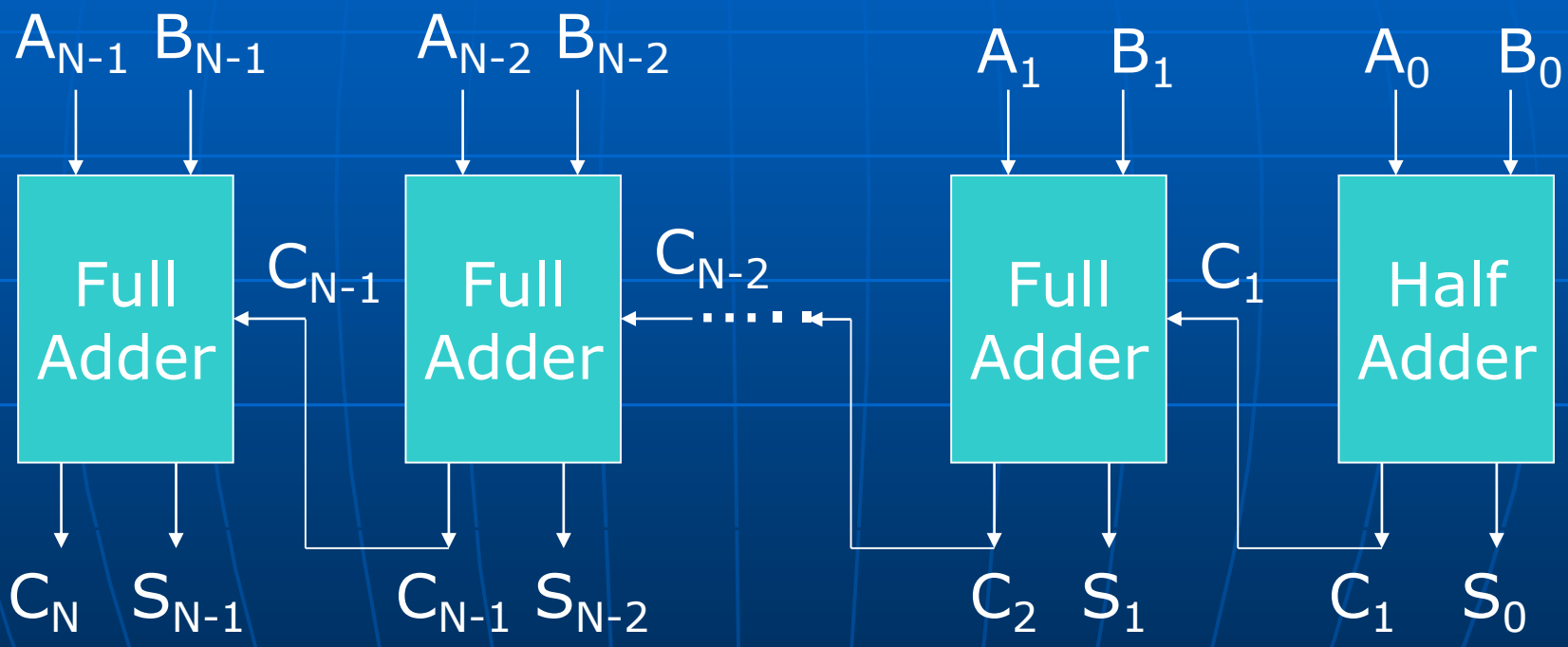
Sumador Ripple-carry (sin signo)

SUMADOR COMPLETO (FULL-ADDER)



Sumador Ripple-carry (sin signo)

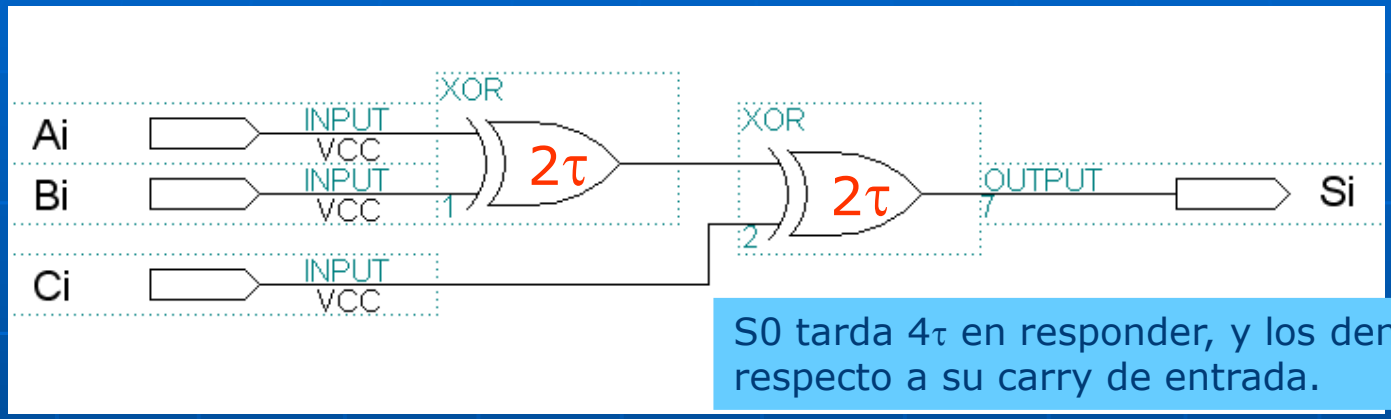
SUMADOR DE "N" BITS



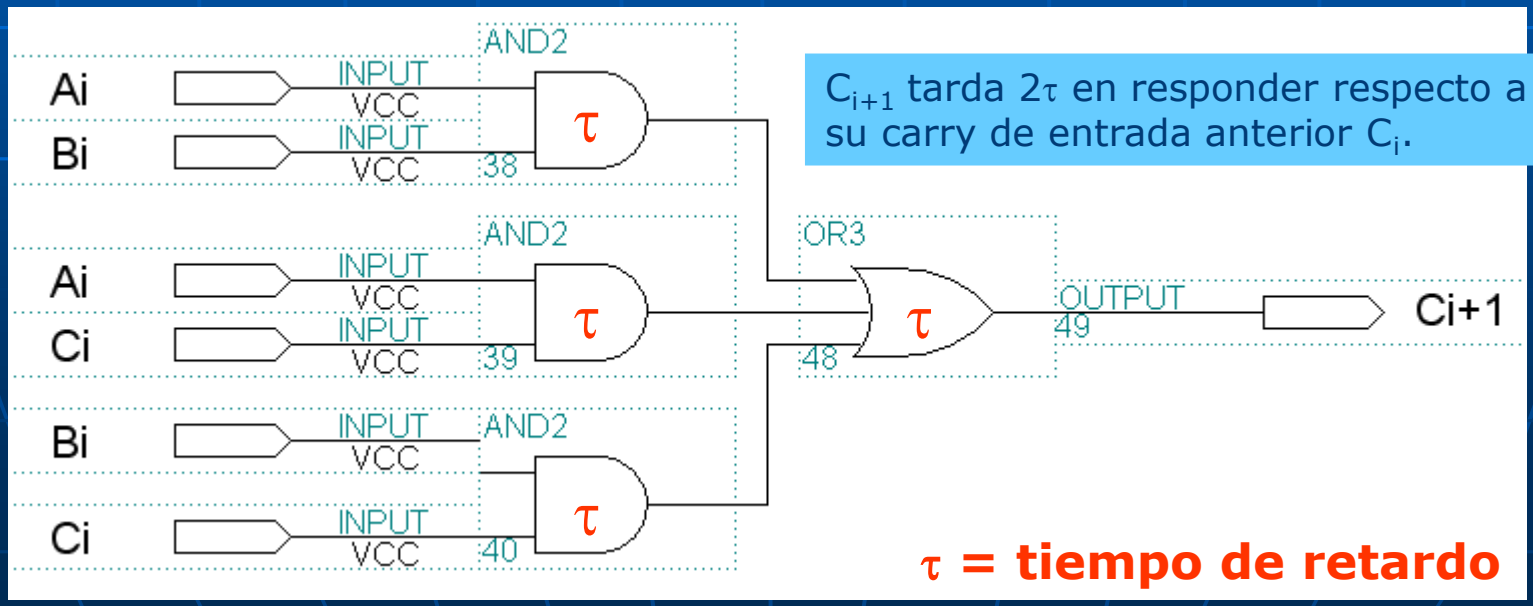
Sumador Ripple-carry (sin signo)

VELOCIDAD DE RESPUESTA

SUMADOR COMPLETO (FULL-ADDER)



S0 tarda 4τ en responder, y los demás: 2τ respecto a su carry de entrada.



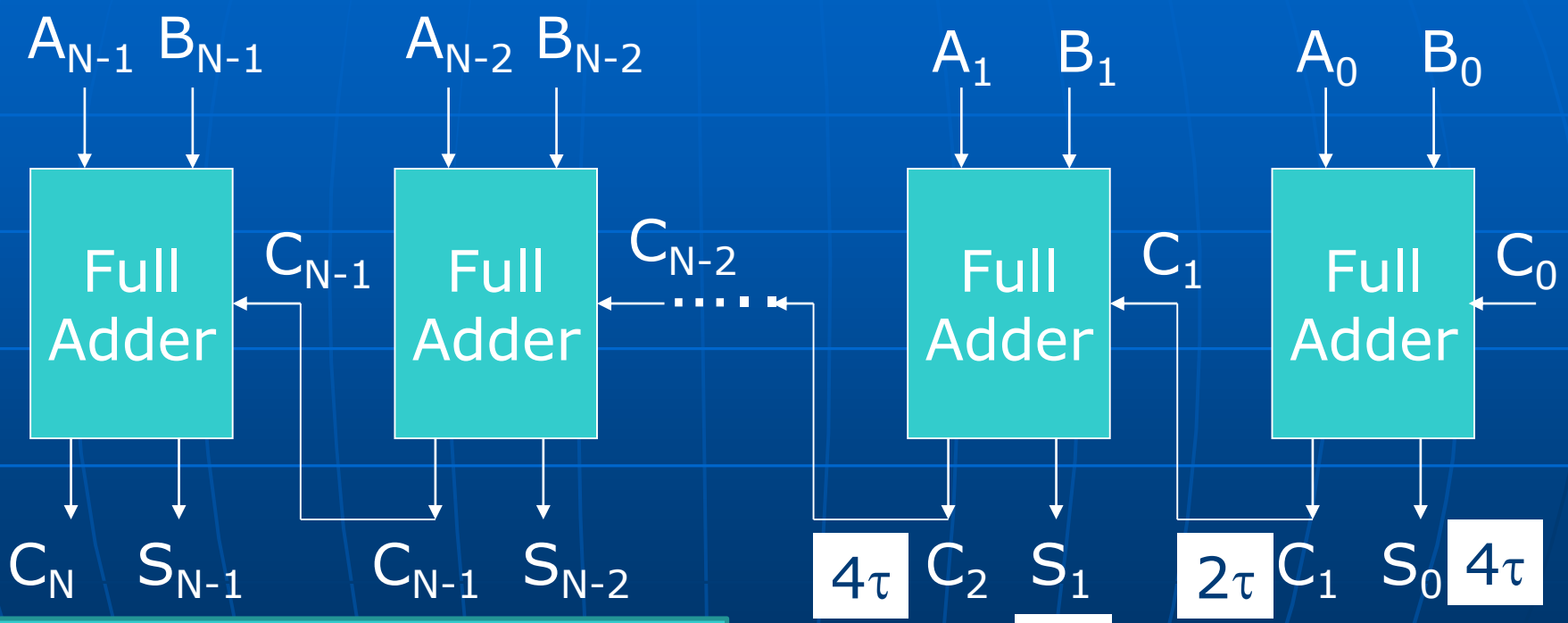
C_{i+1} tarda 2τ en responder respecto a su carry de entrada anterior C_i.

τ = tiempo de retardo

Sumador Ripple-carry (sin signo)

SUMADOR DE "N" BITS

VELOCIDAD DE RESPUESTA



$$t_{pd}(C_{i+1}) = t_{pd}(C_i) + 2\tau$$

$$t_{pd}(S_i) = t_{pd}(C_i) + 2\tau$$

$i \neq 0$

Para N=8 ➔ $t_{pd}(C_i) = t_{pd}(S_i) = 16\tau$

Sumador Ripple-carry (sin signo)

SUMADOR DE "N" BITS

VELOCIDAD DE RESPUESTA

Ventajas:

La estructura es simple y repetible tantas veces como el número de bits que tenga el sumador.

Desventajas:

Se observa como el retardo crece cuanto mayor cantidad de bits tenga el sumador.

La peor condición es cuando debe cambiar la salida del bit mas significativo.

Sumador Look-ahead carry (sin signo)

Esta estructura se basa en generar una lógica que trate de predecir el "carry" para la próxima etapa.
De esta manera se gana en velocidad.

Se definen dos funciones denominadas generate "G" y propagate "P" tal que en una etapa genérica "i" responden a:

$G_i = A_i B_i$ ➔ Indica cuando hay un C_{i+1} independiente de C_i

$P_i = A_i \oplus B_i$ ➔ Indica cuando hay dependencia con C_i

Un sumador de un bit basado en esta estructura responde a:

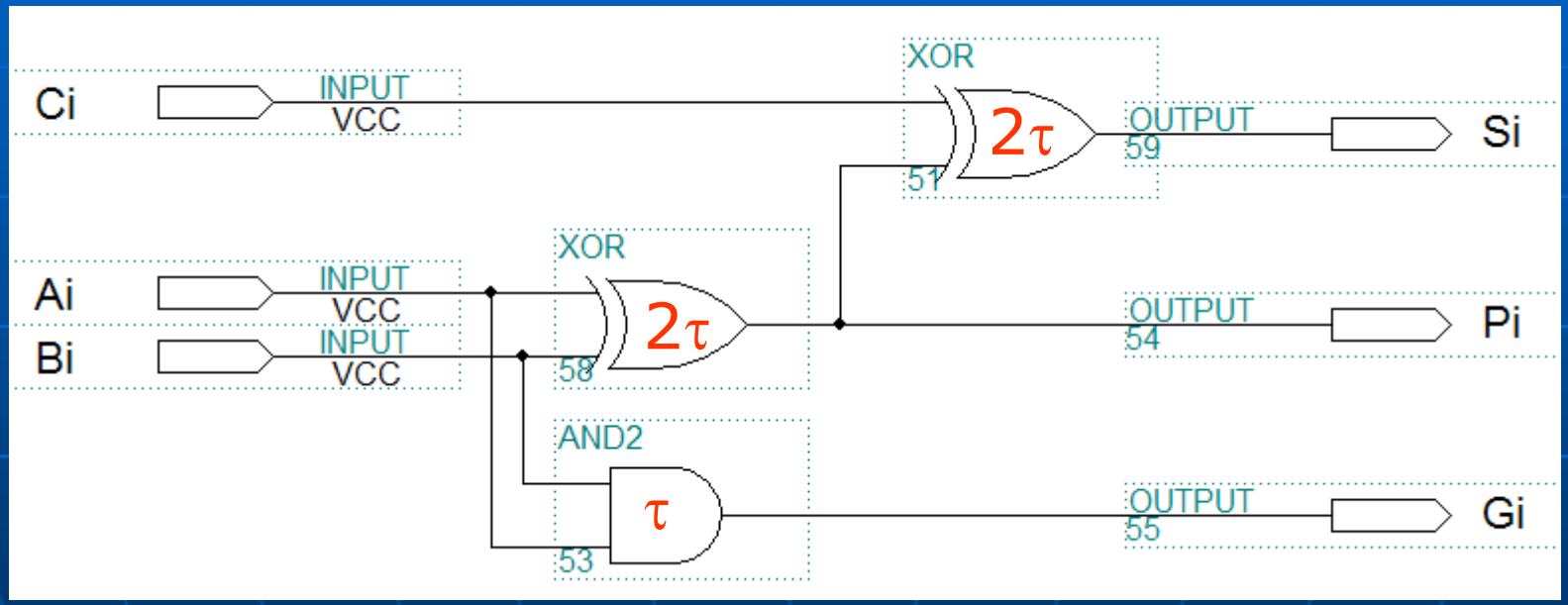
$$Si = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$$

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

Sumador Look-ahead carry (sin signo)

SUMADOR COMPLETO DE 1 BIT

En función de lo anterior se puede construir una etapa genérica:



Nota: Por compatibilidad con algunos textos se considerará aquí que una compuerta XOR tiene el doble de retardo que una compuerta básica (AND u OR).

Sumador Look-ahead carry (sin signo)

SUMADOR COMPLETO DE “4” BIT

$$C1 = G0 + P0\ C0$$

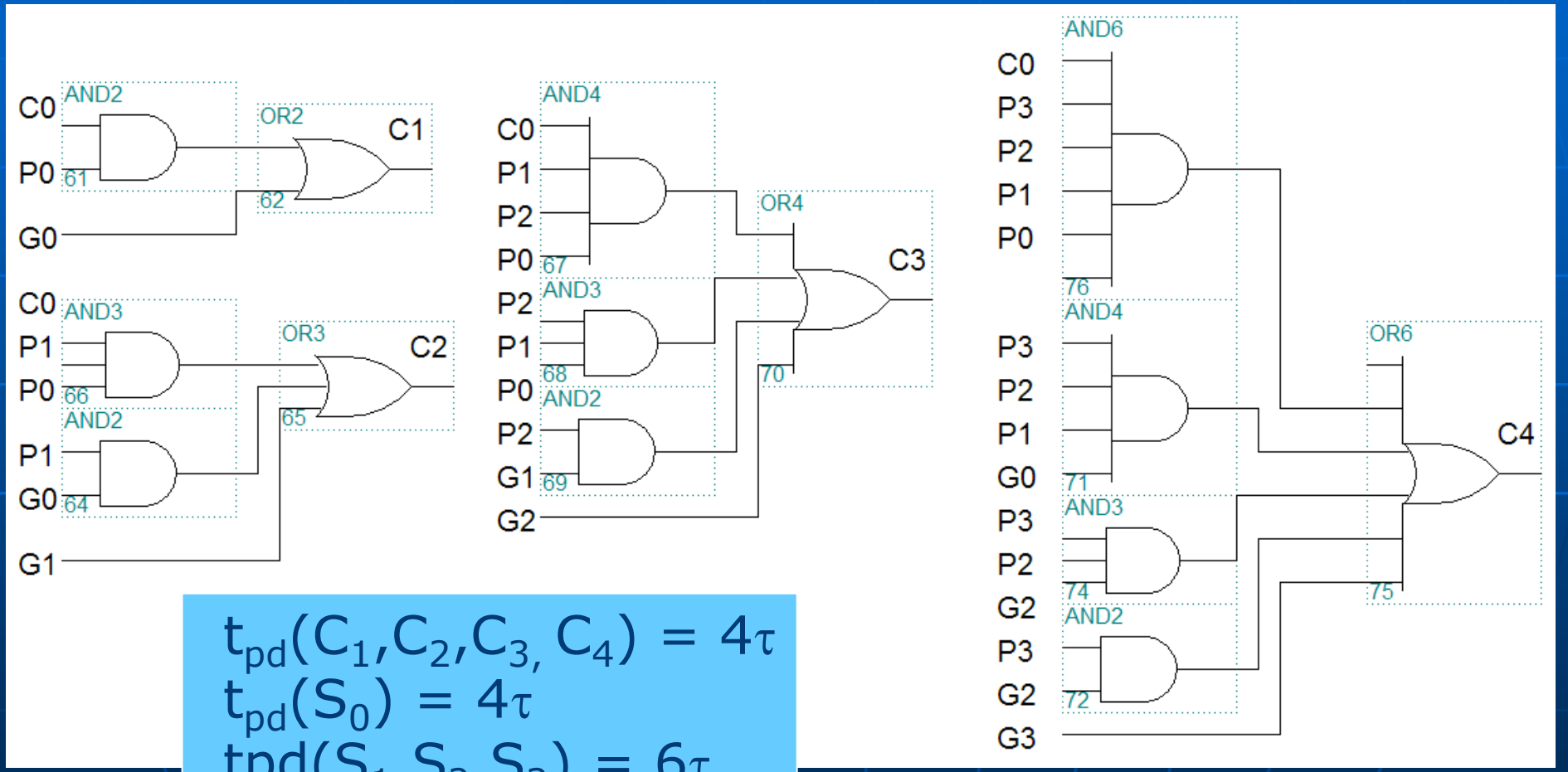
$$C2 = G1 + P1\ C1 = G1 + P1\ G0 + P1\ P0\ C0$$

$$C3 = G2 + P2\ C2 = G2 + P2\ G1 + P2\ P1\ G0 + P2\ P1\ P0\ C0$$

$$C4 = G3 + P3\ C3 = G3 + P3\ G2 + P3\ P2\ G1 + P3\ P2\ P1\ G0 + P3\ P2\ P1\ P0\ C0$$

Sumador Look-ahead carry (sin signo)

SUMADOR COMPLETO DE "4" BIT



$t_{pd}(C_1, C_2, C_3, C_4) = 4\tau$

$t_{pd}(S_0) = 4\tau$

$tpd(S_1, S_2, S_3) = 6\tau$

Sumador Look-ahead-carry (sin signo)

SUMADOR DE “4” BITS

VELOCIDAD DE RESPUESTA

Ventajas:

La estructura basada el la predicción del carry permite disminuir los tiempos de retardo.

Desventajas:

A medida que crece el número de bits a implementar, la lógica de generación de carry se hace mas compleja necesitando de mayor número de compuertas.

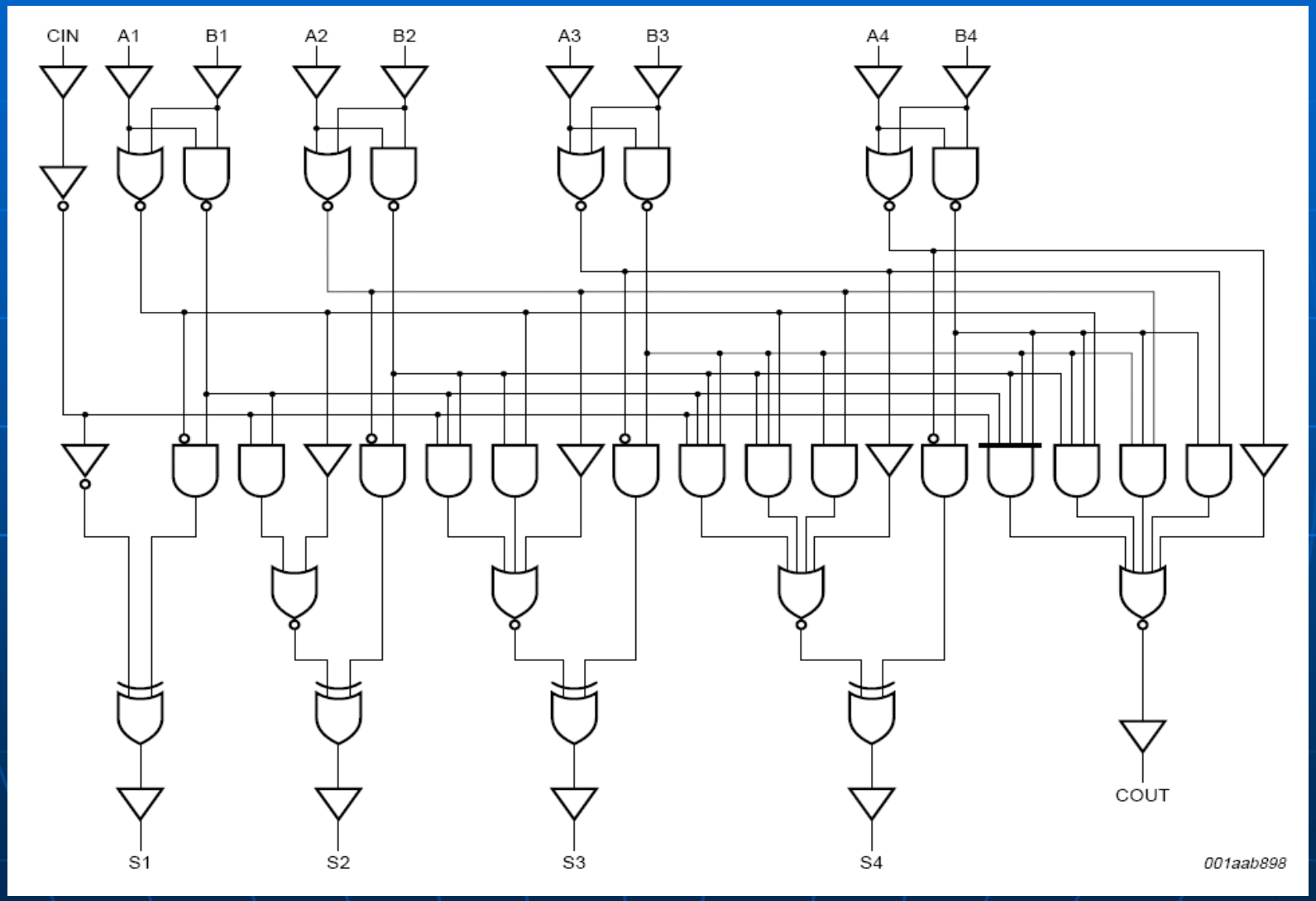
Para evitar esto y no perder la ventaja de la velocidad de respuesta se puede implementar por ejemplo grupos de 4 bits interconectados en cascada (ripple-carry).

CIRCUITOS ARITMÉTICOS

Procesamiento paralelo

Sumador Look-ahead-carry de 4BITS

EJEMPLO:74HC283



Sumador Look-ahead-carry de 4BITS

EJEMPLO:74HC283

De las hojas de datos de este sumador:

Retardo desde Cin a S1 = 52 ns
Retardo desde Cin a S2 = 58 ns
Retardo desde Cin a S3 = 63 ns
Retardo desde Cin a S1 = 74 ns
Retardo desde Ai ó Bi a Si = 74 ns.
Retardo desde Ai ó Bi a Cout = 63 ns.
Retardo desde Cin a Cout = 63 ns.

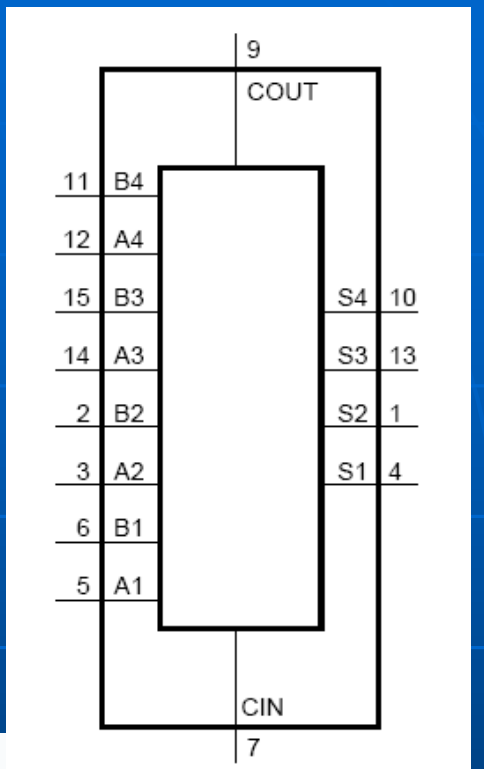


Table 4: Function table [1]

Pins	Input									Output				
	CIN	A4	A3	A2	A1	B4	B3	B2	B1	COUT	S4	S3	S2	S1
Logic levels	L	H	L	H	L	H	L	L	H	H	L	L	H	H
Active HIGH [2]	0	1	0	1	0	1	0	0	1	1	0	0	1	1
Active LOW [3]	1	0	1	0	1	0	1	1	0	0	1	1	0	0

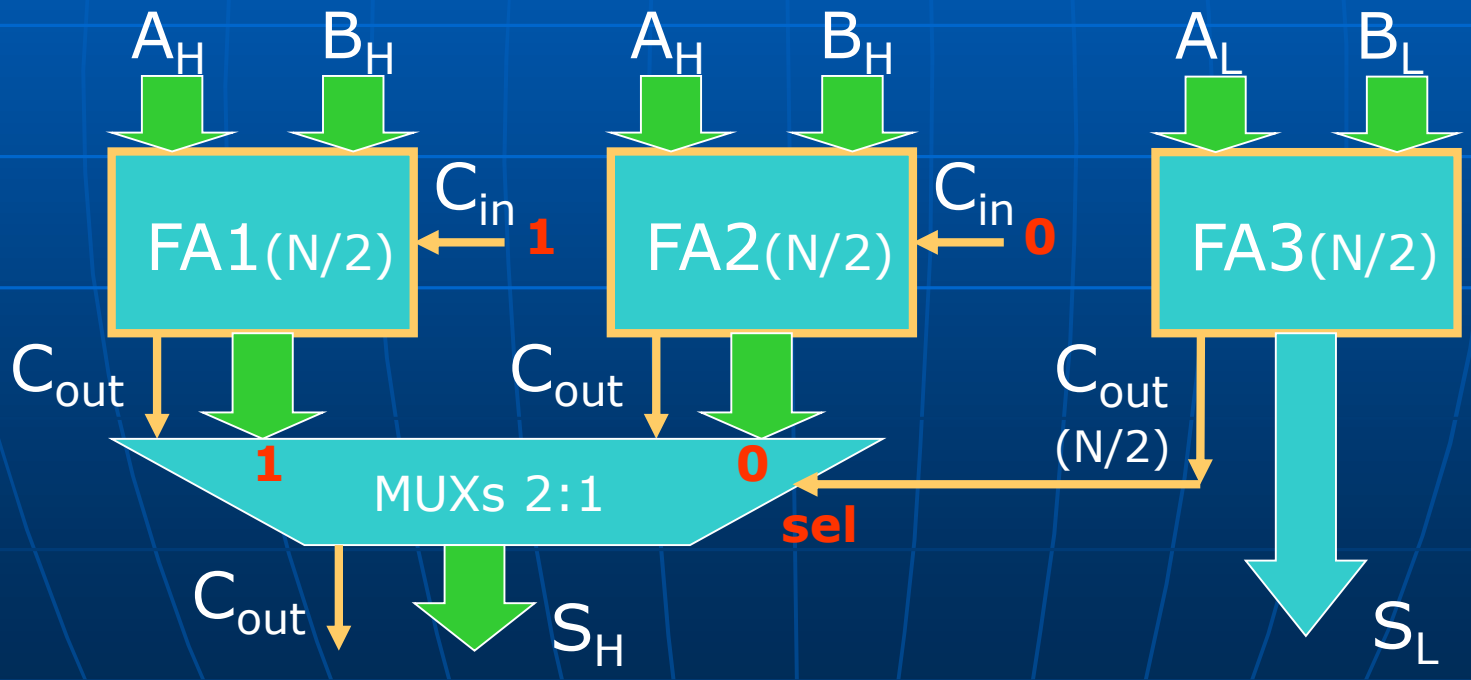
[1] H = HIGH voltage level;
L = LOW voltage level.

[2] Example for active HIGH: 10 + 9 (0 + 1010 + 1001) = 19 (10011).

[3] Example for active LOW: 5 + 6 (1 + 0101 + 0110) = 12 (01100).

Sumador Carry-Select de 8 bits

Esta estructura se basa en la división de un sumador de N bits en dos partes:
La que contiene los bits menos significativos constituído por un FA y la que contiene los bits mas significativos formado por 2 FA y dos MUX´s: uno de 2:1 simple y otro de 2:1 de N/2 bits.



Sumador Carry-Select de 8 bits

Para la suma $A_L + B_L$ se emplea el sumador FA3.
Dependiendo del valor de Carry final de FA3, se presenta en la salida S_H el resultado de la suma ($A_H + B_H$) del FA1 ó FA2A.
Si Carry out(N/2) es "0" ➔ S_H proviene de FA2 y viceversa.

De esta manera, se puede reducir a casi la mitad el tiempo de retardo respecto a un ripple-carry a expensas de mayor complejidad en el diseño.

Sumador Carry-Save

Sirve en general para realizar operaciones de suma cuando se necesitan mas de dos operandos. Además presenta una mejora en la velocidad de respuesta respecto del Ripple-Carry.
La metodología es la sumar los operandos por un lado sin considerar los carry; sumar sólo los carry por el otro y por último realizar esas dos sumas parciales para obtener el resultado correcto de la suma. Existen multiplicadores basados en sumas "carry-save".

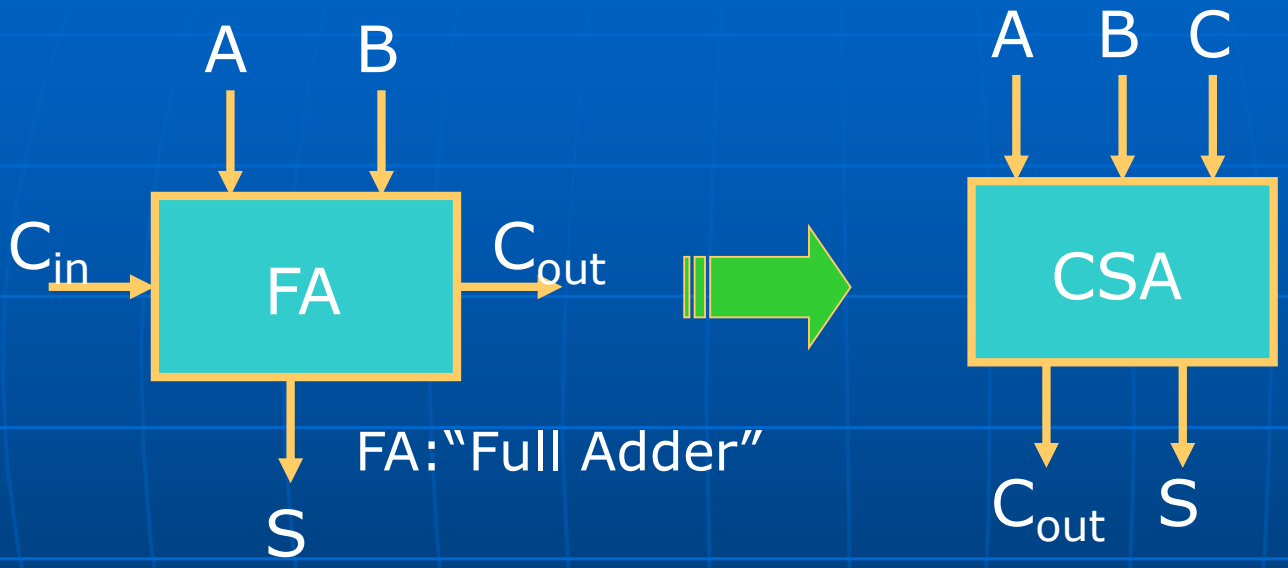
EJEMPLO: Suma de A+B+C, donde: A=10011;B=11001;Z=01011

$A = 10011\ (19_{10})$
$B = 11001\ (25_{10})$
$C = 01011\ (11_{10})$
$s := 00001$

$A = 10011$
$B = 11001$
$C = 01011$
$c := 11011-$

$s := 00001$
$c := 11011-$
$S = 110111\ (55_{10})$

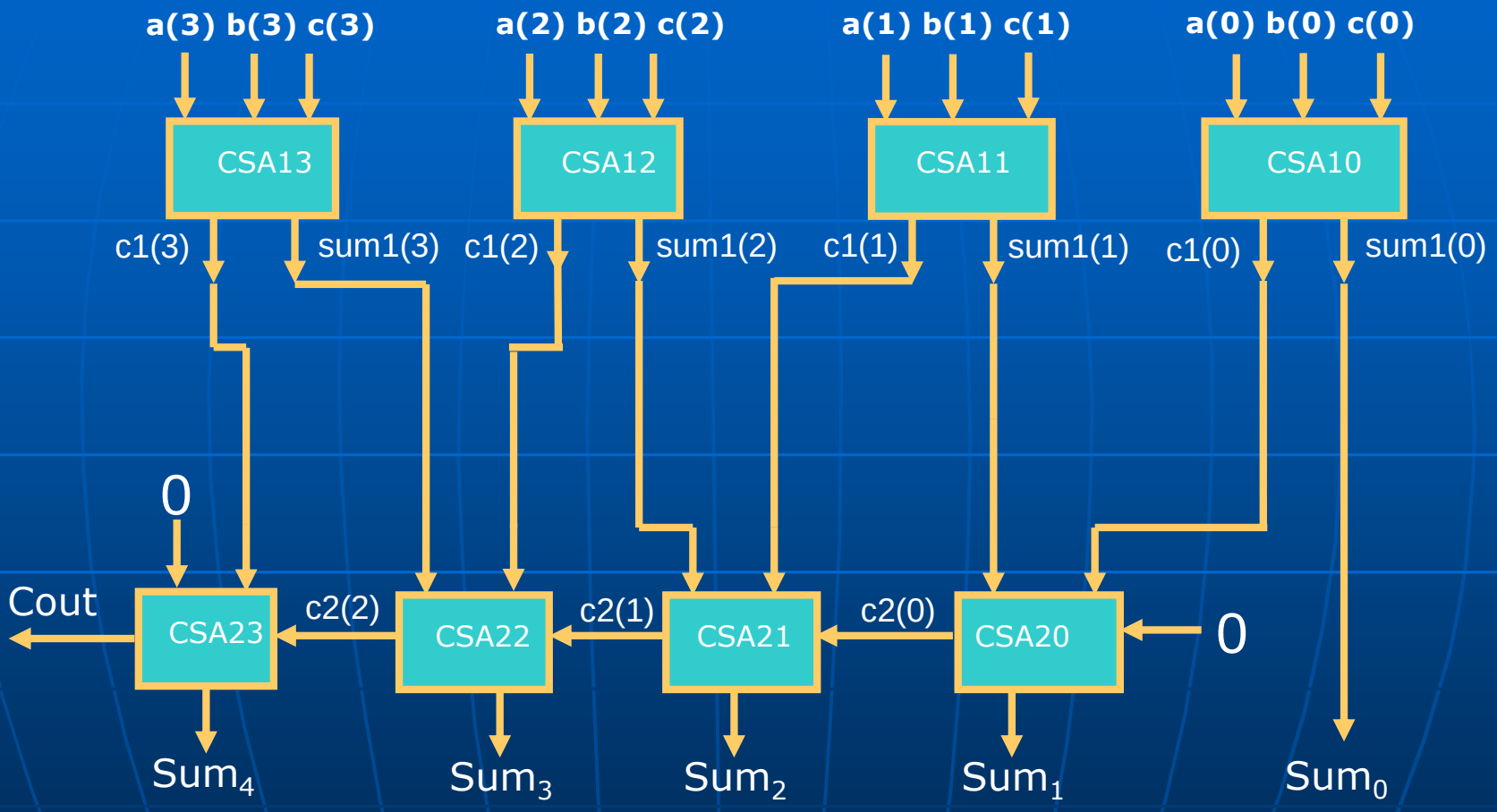
Sumador Carry-Save de un bit de 3 operandos



Hasta aquí sólo reordenamos un FA normal

FA: Full Adder
CSA: Carry-Save Adder

Sumador Carry-Save de 4 bits de 3 operandos



SE PUEDE APRECIAR QUE TIENE MENOS CADENA DE RETARDOS QUE UN RIPPLE-CARRY

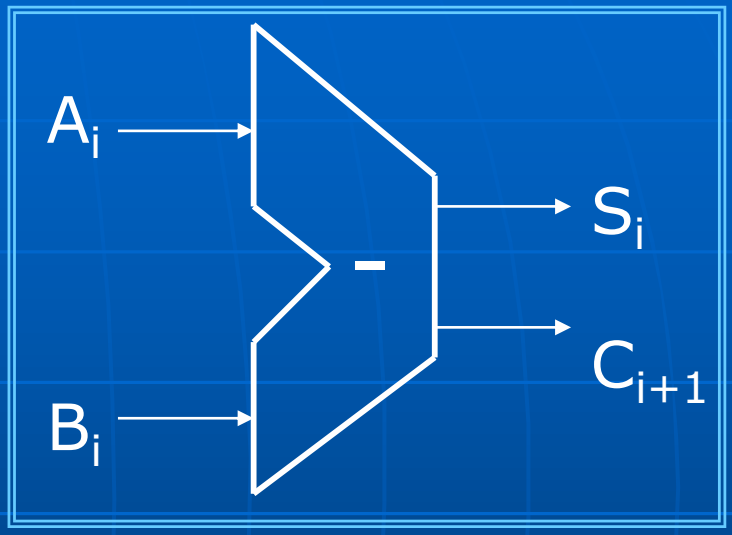
Restador Ripple-carry (sin signo)

SEMI-RESTADOR DE UN BIT (HALF-SUBTRACTER)

Tabla de verdad

A	B	S	C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$S = A - B$



$Si = Ai \oplus Bi \quad ; \quad Ci = /Ai \bullet Bi$

C se denomina “borrow” préstamo.

Restador Ripple-carry (sin signo)

RESTADOR COMPLETO DE UN BIT (FULL-SUBTRACTER)

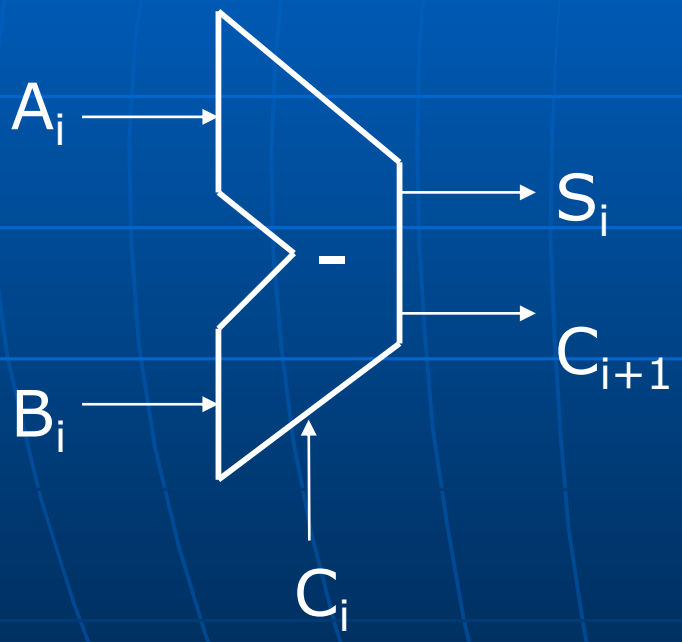


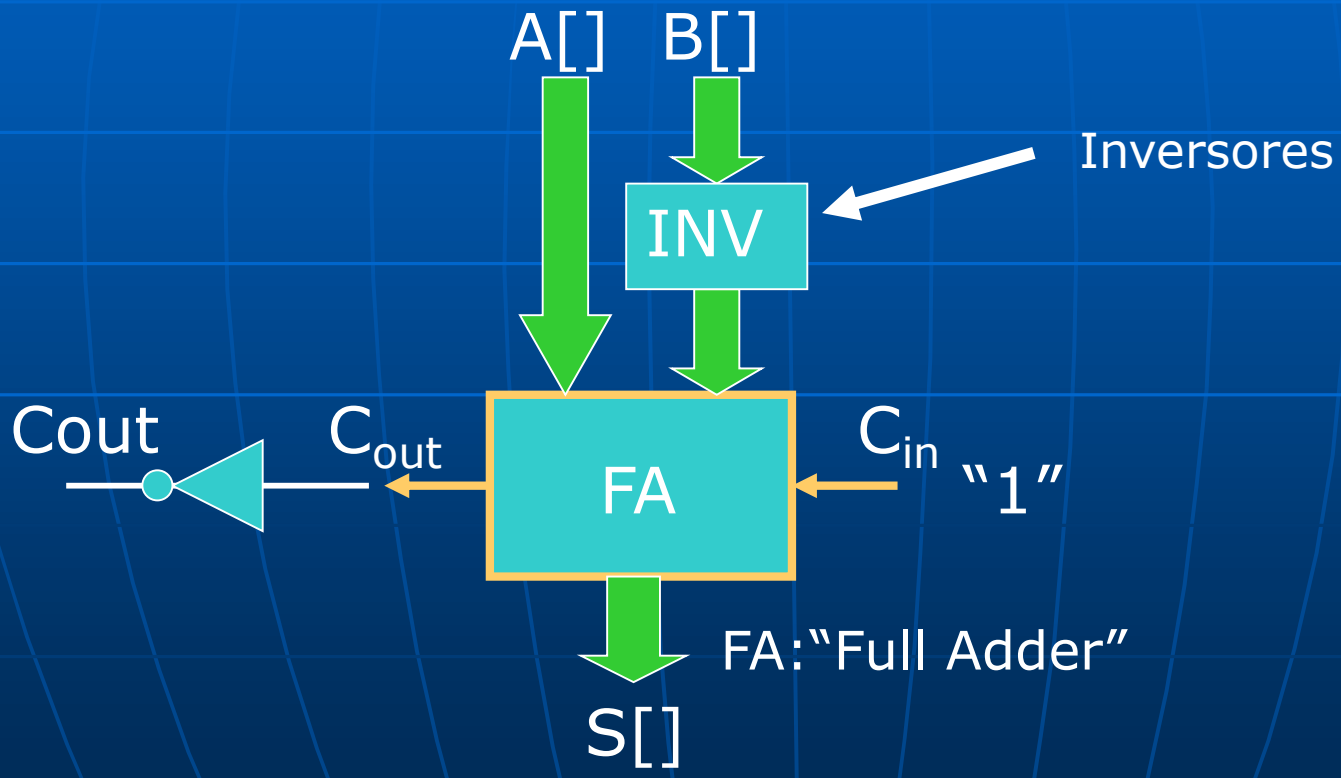
Tabla de verdad

A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Restador en CA2 basado en Sumador FA

Para realizar (A – B) en Ca2 podemos plantear:

$S = A\text{-}B = A + 2^n - B = A - B_{CA1} + 1 = A + (/B) + 1$



Multiplicadores por número potencia de 2 (sin signo)

Desplazamientos de bits a izquierda son equivalentes a multiplicar por números en potencia de 2.

Dado: A = 00010111 = 23_{10}
desplazando un lugar: B = 00101110 = 46_{10}
desplazando 2 lugares: C = 01011100 = 92_{10}
desplazando 3 lugares: D = 10111000 = 184_{10}

Desplazamientos de bits a derecha son equivalentes a dividir por números en potencia de 2.

Dado: A = 10010000 = 144_{10}
desplazando un lugar: B = 01001000 = 72_{10}
desplazando 2 lugares: C = 00100100 = 36_{10}
desplazando 3 lugares: D = 00010010 = 18_{10}

Solución: Empleo de registros de desplazamiento ó circuitos basados en barrel-shifters.

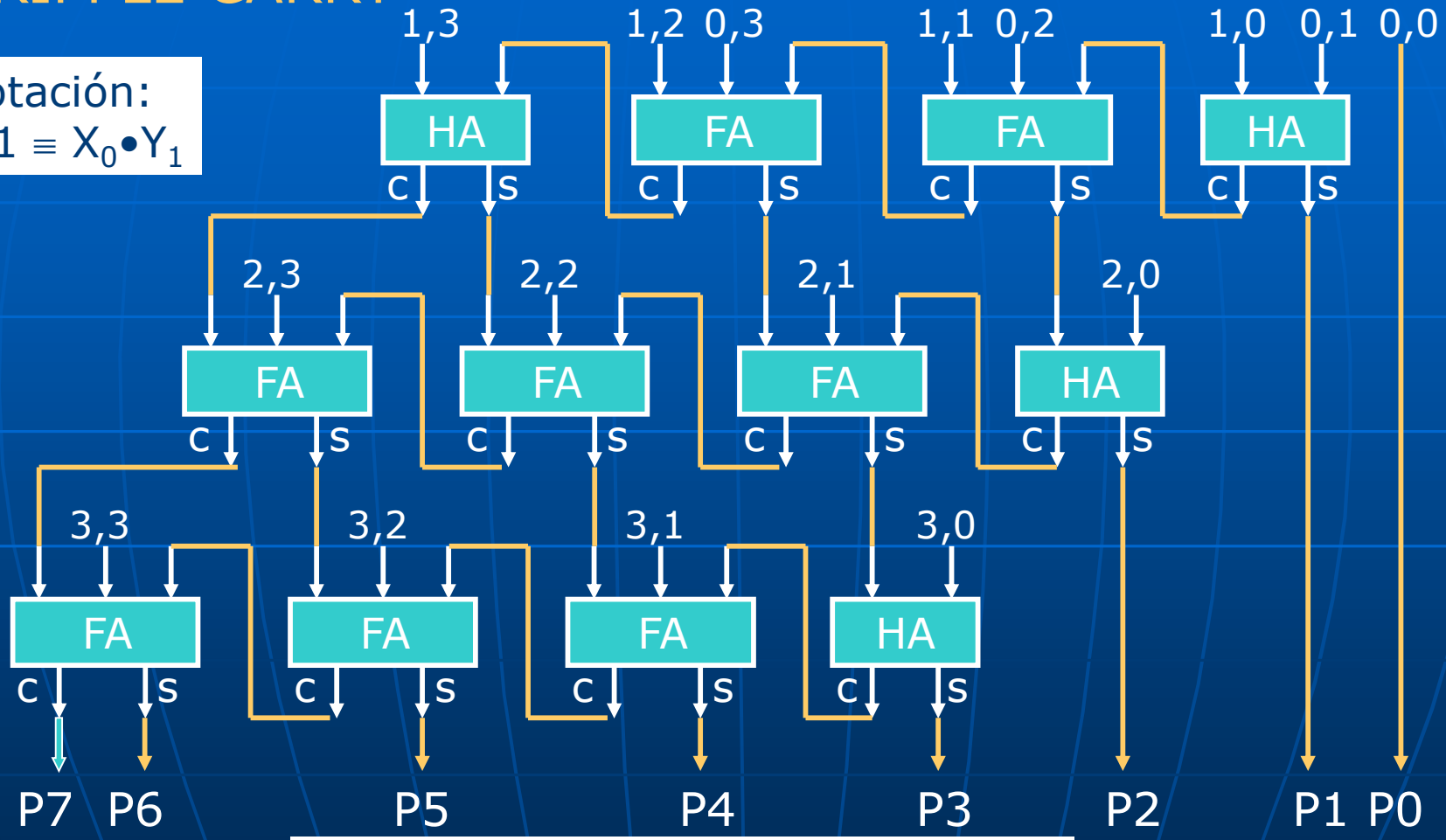
Multiplicadores sin signo (algoritmo convencional)

```
                          1011
                        x 1010
                        ------
Producto parcial 1 →   1  0000
Producto parcial 2 →     1011
Producto parcial 3 →    0000
Producto parcial 4 →   1010
                     ---------
                       1101110
```

Las operación de multiplicación se reduce en una serie de operaciones AND entre el multiplicando y cada bit del multiplicador considerando el peso de cada operación a través de desplazamiento a izquierda. Luego se realizan las las sumas de los productos parciales obtenidos.

Multiplicadores sin signo (algoritmo convencional)
RIPPLE CARRY

Notación:
$0,1 \equiv X_0 \bullet Y_1$



$$P[7..0] = X[3..0] \bullet Y[3..0]$$

Multiplicadores sin signo (algoritmo de Booth)

						X3	**X2**	**X1**	**X0**
Only 2 LSB used	>				x			**Y1**	**Y0**
if Y1=0,Y0=0	|	0	0	0	0	0	0	0	0
if Y1=0, Y0=1	|	0	0	0	0	X3	X2	X1	X0
if Y1=1, Y0=0	|	0	0	0	X3	X2	X1	X0	0
if Y1=1, Y0=1	|	0	0	0	X3	X3+X2	X2+X1	X1+X0	0
Multiplexer A result	>	0	0	PPA5	PPA4	PPA3	PPA2	PPA1	PPA0

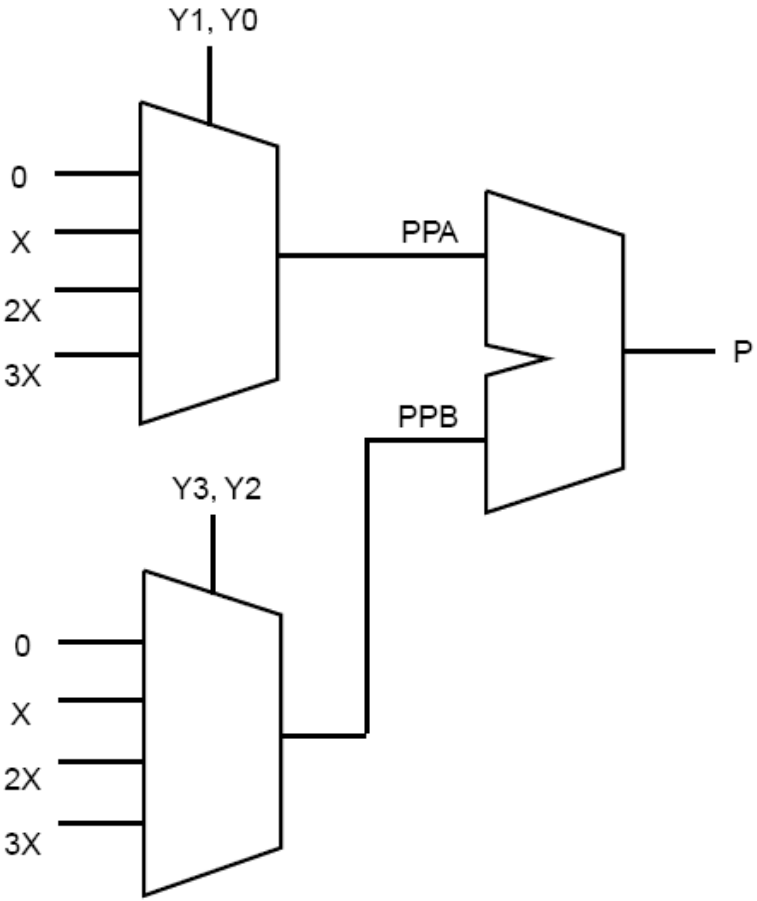
x

						X3	**X2**	**X1**	**X0**
Only 2 MSB used	>				x	**Y3**	**Y2**		
if Y3=0,Y2=0	|	0	0	0	0	0	0	0	0
if Y3=0, Y2=1	|	0	0	X3	X2	X1	X0	0	0
if Y3=1, Y2=0	|	0	X3	X2	X1	X0	0	0	0
if Y2=1, Y2=1	|	0	X3	X3+X2	X2+X1	X1+X0	0	0	0
Multiplexer B result	>	PPB7	PPB6	PPB5	PPB4	PPB3	PPB2	0	0

x

	0	0	PPA5	PPA4	PPA3	PPA2	PPA1	PPA0
+	PPB7	PPB6	PPB5	PPB4	PPB3	PPB2	0	0
Final product >	P7	P6	P5	P4	P3	P2	P1	P0

Multiplicadores sin signo (algoritmo de Booth)



El ejemplo ilustra un multiplicador X[3..0] • Y[3..0] (4 x 4 bits)

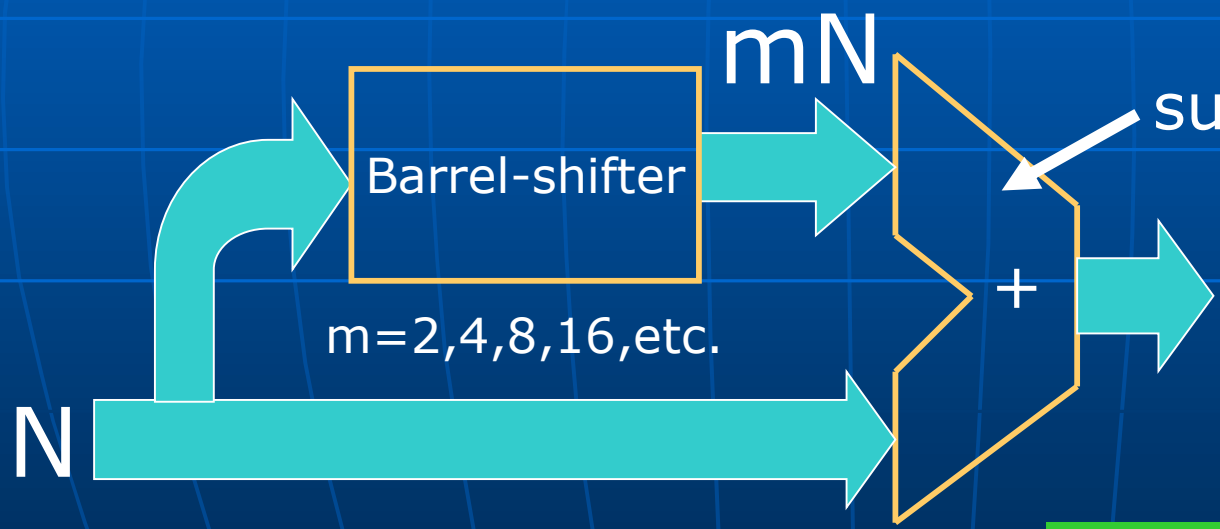
Se realizan dos multiplicaciones parciales (PPA y PPB) empeando sumadores, compuertas AND y MUXs.
Y[1..0] e Y[3..2] manejan por separado un MUX.

La ventaja de este diseño es que en las FPGA los MUX son un recurso muy común lo que hace un circuito mas compacto además de velocidad razonable.

NOTA DE APLICACIÓN: "Implementing multipliers with Actel FPGAs" de ACTEL

Multiplicadores sin signo (dato por una constante)

Puede multiplicar ó dividir por m donde $m=2^n$, siendo n un número entero (+) ó (-) que representa las veces que se desplaza el dato N.

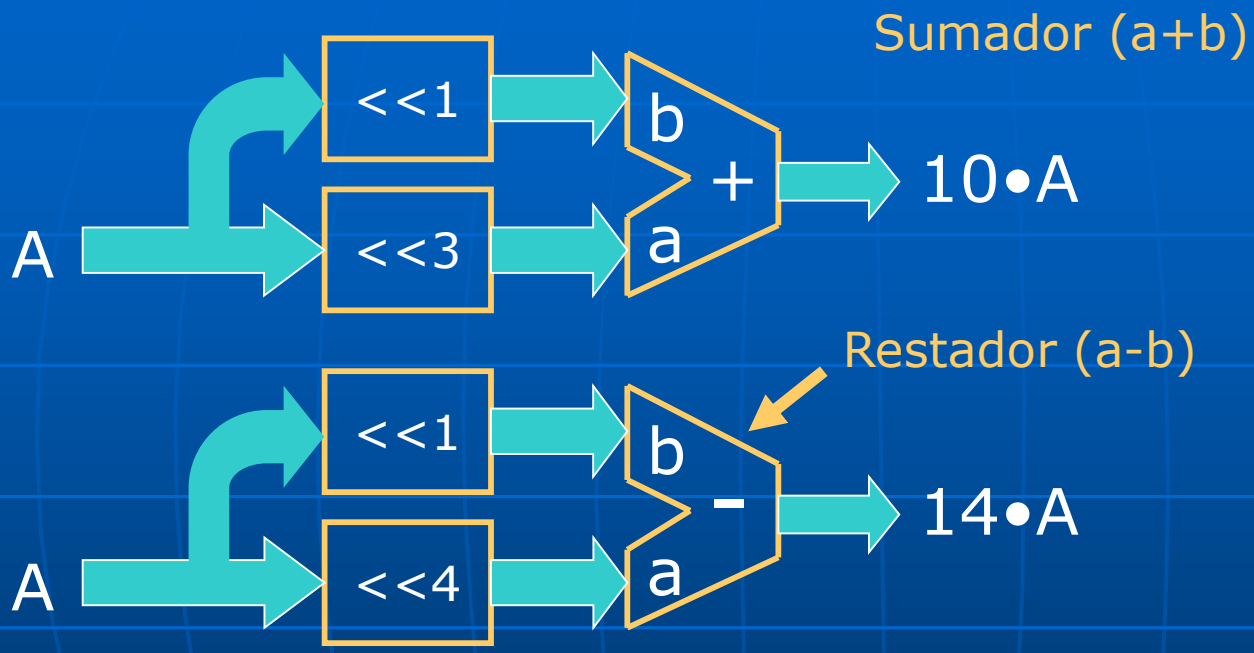


mN+N = (m+1)N

Ejemplo: m=2 ➔ se tiene 3xN

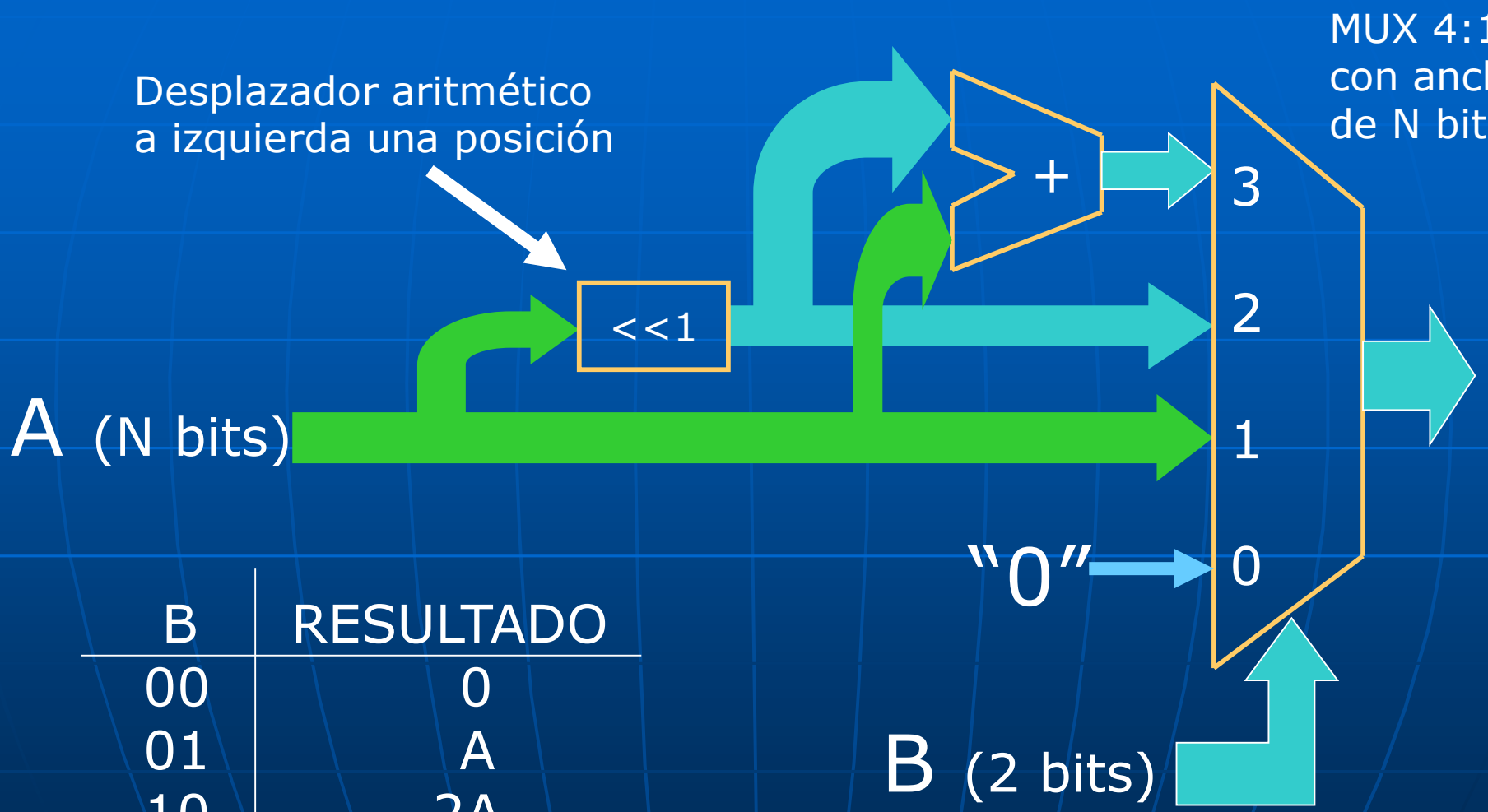
Cómo se puede hacer: 9/8N..???

Multiplicadores sin signo (dato por una constante)



<<1: Desplazamiento aritmético hacia izquierda una posición ➔ (x2)
<<3: Desplazamiento aritmético hacia izquierda 3 posiciones ➔ (x8)
<<4: Desplazamiento aritmético hacia izquierda 4 posiciones ➔ (x16)

Multiplicadores de Productos Parciales (usado por Xilinx)

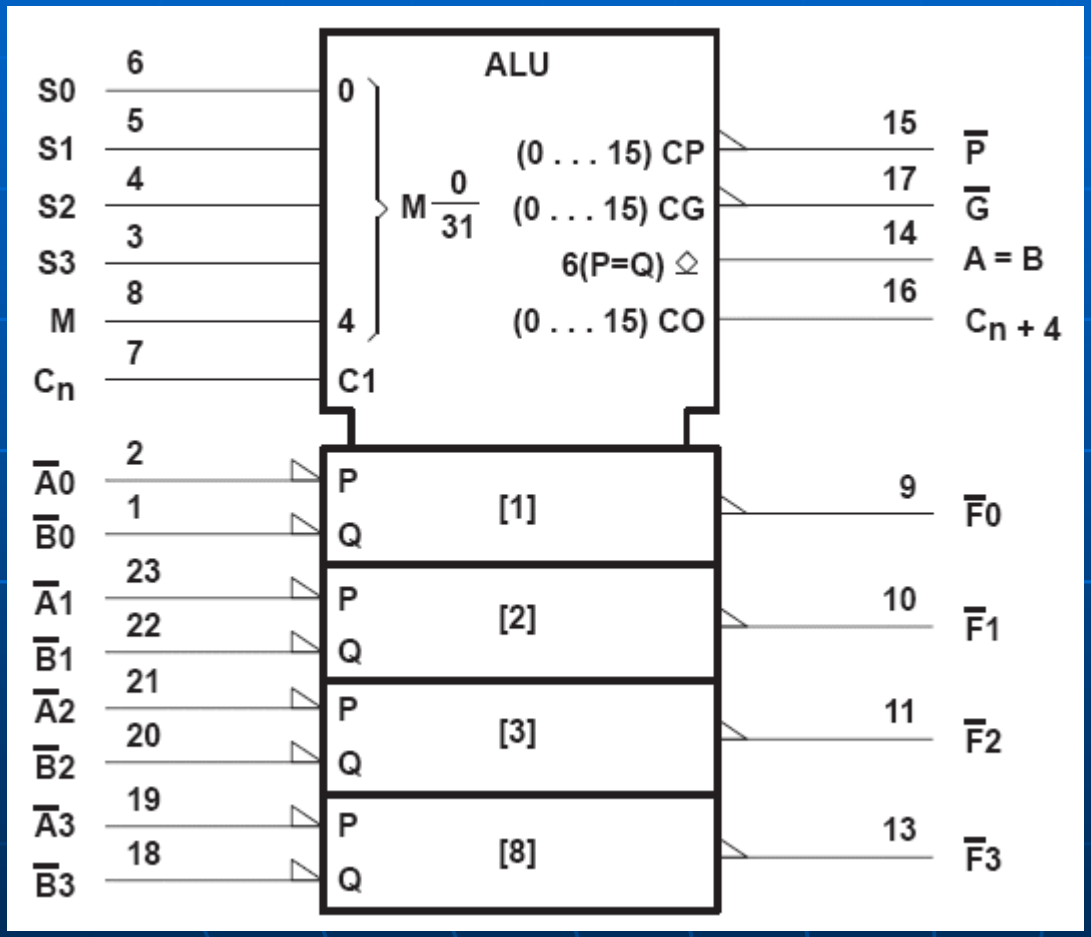


B	RESULTADO
00	0
01	A
10	2A
11	3A

Unidad Aritmético-Lógica

Procesamiento paralelo

EJEMPLO: SN74AS181



Circuito que puede realizar funciones lógicas ó aritmé-ticas de 4 bits según entrada de selección M.
Los operandos de entrada lógicos son A y B y el de salida F.
Se agregan el carry de entrada C_n y de salida C_{n+4} para operaciones como números.

Unidad Aritmético-Lógica

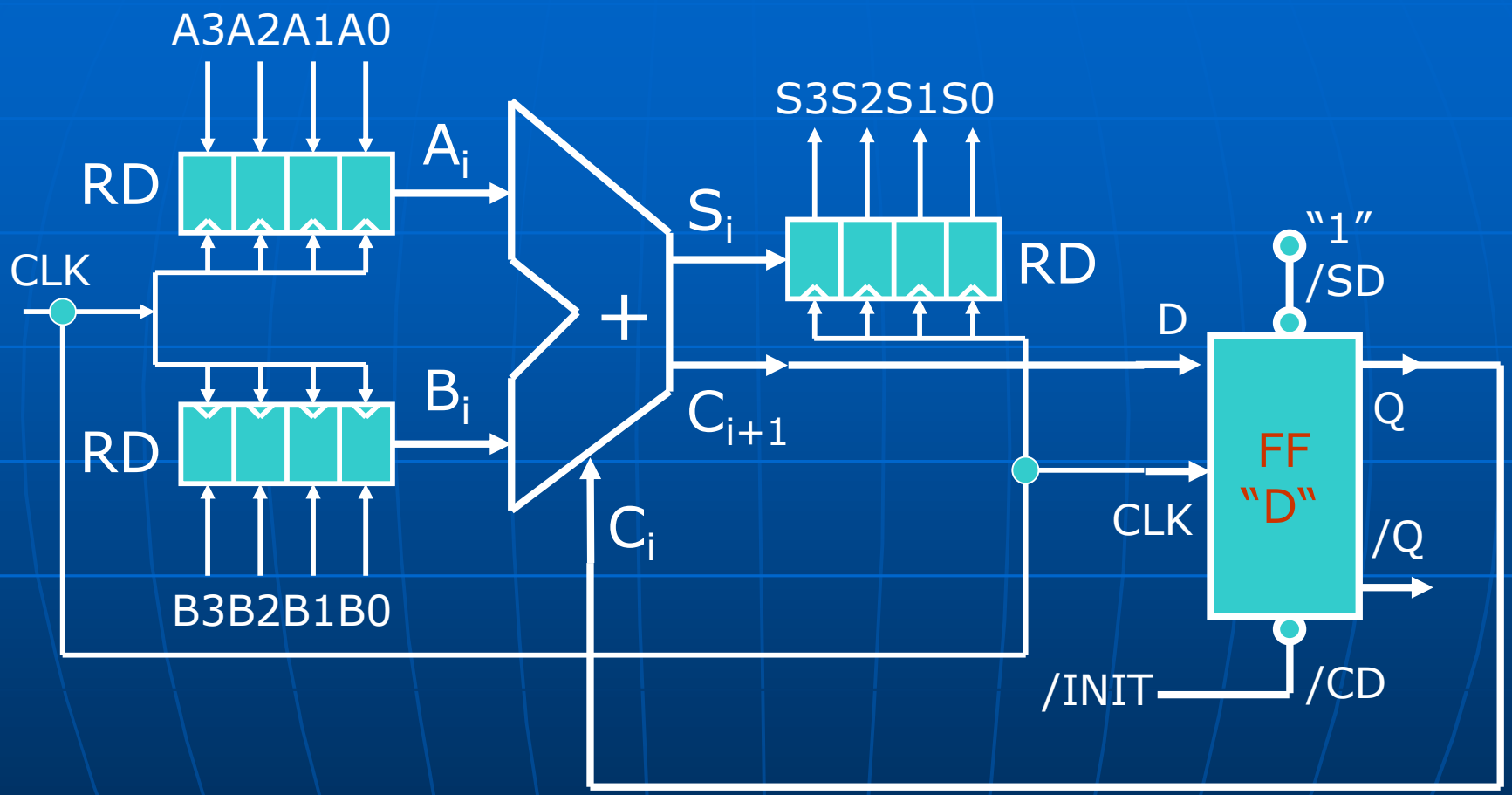
EJEMPLO: SN74AS181

SELECTION				ACTIVE-LOW DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	C_n = L (no carry)	C_n = H (with carry)
L	L	L	L	$F = \overline{A}$	F = A MINUS 1	F = A
L	L	L	H	$F = \overline{AB}$	F = AB MINUS 1	F = AB
L	L	H	L	$F = \overline{A} + B$	F = $A\overline{B}$ MINUS 1	F = $A\overline{B}$
L	L	H	H	F = 1	F = MINUS 1 (2's COMP)	F = ZERO
L	H	L	L	$F = \overline{A + B}$	F = A PLUS (A + $\overline{B}$)	F = A PLUS (A + $\overline{B}$) PLUS 1
L	H	L	H	$F = \overline{B}$	F = AB PLUS (A + $\overline{B}$)	F = AB PLUS (A + $\overline{B}$) PLUS 1
L	H	H	L	$F = \overline{A \oplus B}$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A + \overline{B}$	$F = A + \overline{B}$	F = (A + $\overline{B}$) PLUS 1
H	L	L	L	$F = \overline{AB}$	F = A PLUS (A + B)	F = A PLUS (A + B) PLUS 1
H	L	L	H	$F = A \oplus B$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	F = $A\overline{B}$ PLUS (A + B)	F = $A\overline{B}$ PLUS (A + B) PLUS 1
H	L	H	H	F = A + B	F = (A + B)	F = (A + B) PLUS 1
H	H	L	L	F = 0	F = A PLUS A†	F = A PLUS A PLUS 1
H	H	L	H	$F = A\overline{B}$	F = AB PLUS A	F = AB PLUS A PLUS 1
H	H	H	L	F = AB	F = $A\overline{B}$ PLUS A	F =$A\overline{B}$ PLUS A PLUS 1
H	H	H	H	F = A	F = A PLUS 1	F = A PLUS 1

† Each bit is shifted to the next more significant position.

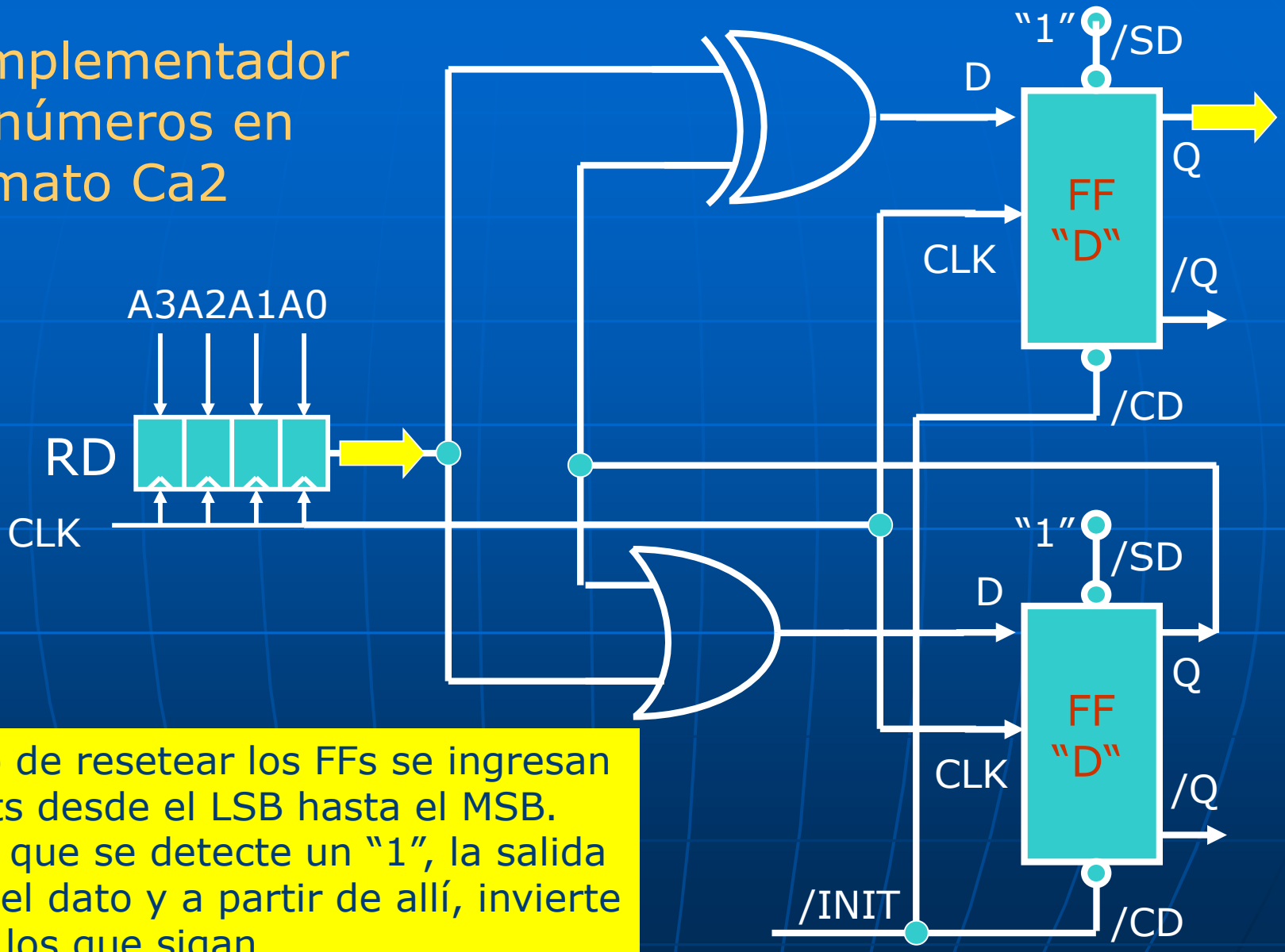
CIRCUITOS ARITMÉTICOS

Sumador sin signo



CIRCUITOS ARITMÉTICOS

Complementador de números en formato Ca2



Luego de resetear los FFs se ingresan los bits desde el LSB hasta el MSB. Hasta que se detecte un "1", la salida copia el dato y a partir de allí, invierte todos los que sigan.

Bibliografía:

Libros:

- "Sistemas Digitales". R. Tocci, N. Widmer, G. Moss. Ed. Prentice Hall.
- "Diseño Digital". M. Morris Mano. Ed. Prentice Hall. 3ra edición.
- "Diseño de Sistemas Digitales". John Vyemura. Ed. Thomson.
- "Diseño Lógico". Antonio Ruiz, Alberto Espinosa. Ed. McGraw-Hill.
- "Digital Design:Principles & Practices". John Wakerly. Ed. Prentice Hall.
- "Diseño Digital". Alan Marcovitz. Ed. McGraw-Hill.
- "Electrónica Digital". James Bignell, R. Donovan. Ed. CECSA.
- "Técnicas Digitales con Circuitos Integrados". M. Ginzburg.
- "Fundamentos de Diseño Lógico y Computadoras". M. Mano, C. Kime. Ed. Prentice Hall.
- "Teoría de conmutación y Diseño lógico". F. Hill, G. Peterson. Ed. Limusa.
- "Aplicaciones aritméticas usando lógica programable". Guillermo Jaquenod, Marisa DeGuisti, Roberto de La Vega. UniCen, CIC.