Arquitectura de Computadoras para Ingeniería

(Cód. 7526)
1° Cuatrimestre 2016

Dra. Dana K. Urribarri
DCIC - UNS

Operaciones Aritméticas

Implementación de las operaciones aritméticas básicas:

1) Suma
2) Resta
3) Multiplicación
4) División

Simulador

http://www.ecs.umass.edu/ece/koren/arith/simulator/

Sumadores

Sumadores

- Sea b una base genérica.
- Dos números de n dígitos:

  $X = x_{n-1} \ldots x_0$

  $Y = y_{n-1} \ldots y_0$

- La suma $S = X + Y$

  $S = s_n s_{n-1} \ldots s_0$

- Para cualquier base b, el acarreo S_n vale 0 o 1.
- Si $s_n=1$, hay overflow.

Algoritmo básico

1) $c_0 \leftarrow 0$ (acarreo inicial)

2) For i=0 step 1 until n-1 do

   – $s_i \leftarrow (x_i + y_i + c_i) \bmod b$

   – $c_{i+1} \leftarrow \lfloor (x_i + y_i + c_i) / b \rfloor$

3) $s_n \leftarrow c_n$

Algoritmo básico b=2

En binario, se traducen los pasos en términos de 0 y 1.

1) $c_0 \leftarrow 0$ (acarreo inicial)

2) For i=0 step 1 until n-1 do

   - $s_i \leftarrow$ LSB de $x_i + y_i + c_i$
   - $c_{i+1} \leftarrow$ MSB de $x_i + y_i + c_i$

3) $s_n \leftarrow c_n$

Algoritmo básico $b=2$

- La complejidad del algoritmo el lineal O(n).
- El tiempo de ejecución es $T(n) = kn$
- El k va a depender de la tecnología.

- La complejidad del algoritmo no puede mejorarse, dado que hay que recorrer todos los dígitos.
- Lo que se mejora es la implementación.

Half adder y Full adder

- Para *b* = 2 vamos a buscar las expresiones lógicas que resuelvan la operación de suma.
  - *Half adder o semisumador*
  - *Full adder o sumador completo*

Half adder

- El sumador más simple.
- Suma dos operandos de 1 bit.
- Resultado de 2 bits dentro del rango 0 – 2.
  - 0 + 0 = 0
  - 0 + 1 = 1 + 0 = 1
  - 1 + 1 = 10

Se puede resumir en la siguiente tabla de verdad:

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

Half adder

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

S

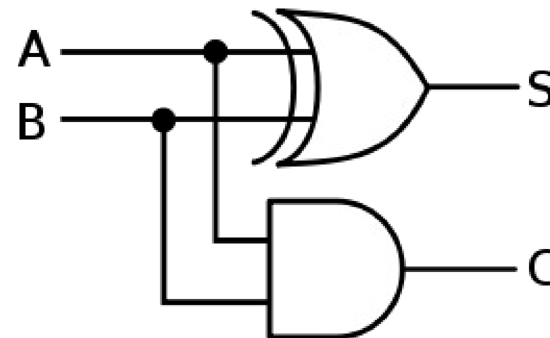
A \ B	0	1
0	0	1
1	1	0

$$S \leftarrow A \oplus B$$

C

A \ B	0	1
0	0	0
1	0	1

$$C \leftarrow A \cdot B$$



Full adder

- Realiza la suma de 3 bits.
- Resultado de 2 bits dentro del rango 0 – 3.

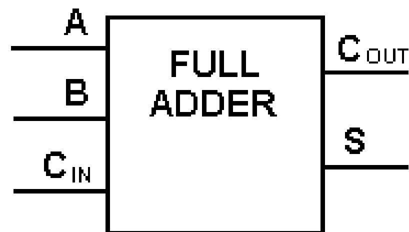
A	B	C	S	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full adder

- Es el bloque para construir sumadores de varios bits.
- El carry al sumar dos bits hay que sumarlo junto a los siguientes bits más significativos.
- De las tres entradas:
  - Dos de las entradas (A y B) son las mismas en el half adder.
  - La tercer entrada (C) es el carry de la suma de los bits (menos significativos) anteriores.

Full adder

A	B	C_{IN}	S	C_{OUT}
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



S:

C_{IN} \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$S \leftarrow A \oplus B \oplus C_{IN}$$

C_{OUT}:

C_{IN} \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

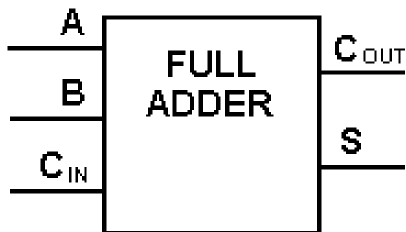
$$C_{OUT} \leftarrow A \cdot B + (A + B) \cdot C_{IN}$$

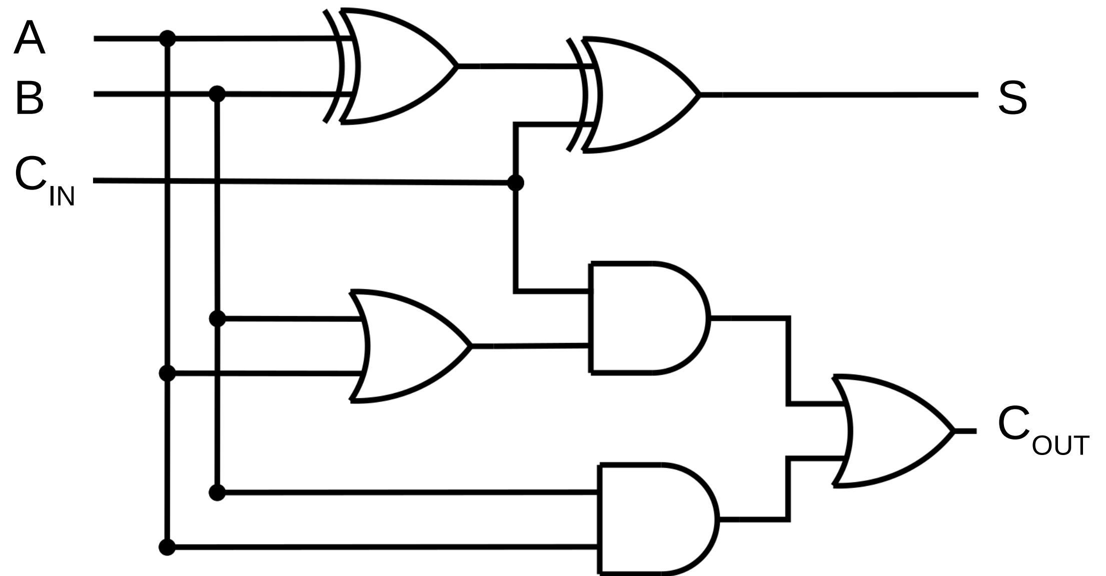
Full adder

A	B	C_{IN}	S	C_{OUT}
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

$$S \leftarrow A \oplus B \oplus C_{IN}$$

$$C_{OUT} \leftarrow A \cdot B + (A + B) \cdot C_{IN}$$





Algoritmo básico b=2

Completamos el algoritmo básico de la suma en binario.

1) $c_0 \leftarrow 0$ (acarreo inicial)

2) For i=0 step 1 until n-1 do

   – $s_i \leftarrow x_i \oplus y_i \oplus c_i$

   – $c_{i+1} \leftarrow x_i \cdot y_i + (x_i + y_i) \cdot c_i$

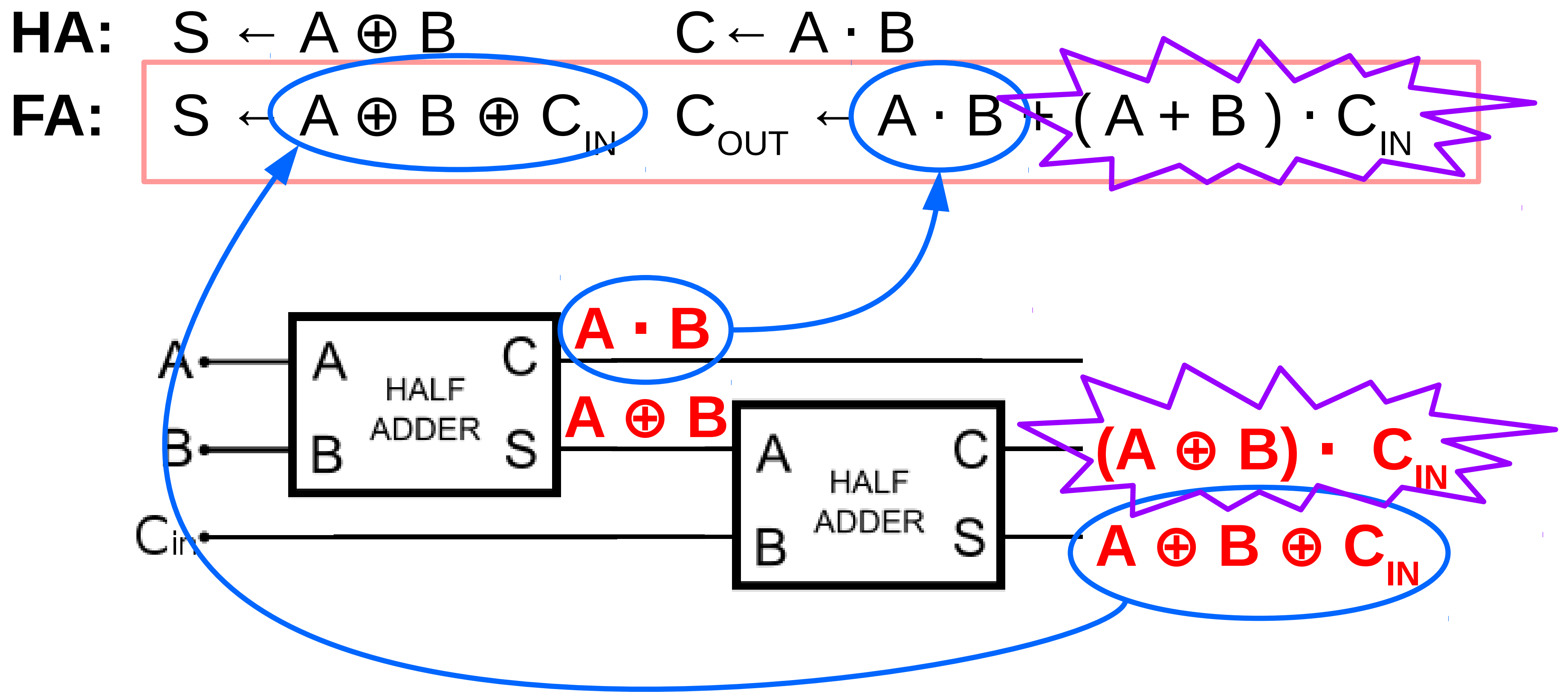
3) $s_n \leftarrow c_n$

La XOR tiene muchas implementaciones posibles.

Dependerá de las compuertas disponibles: OR, AND, NOT, NAND, NOR.

Full adder

- Se puede implementar conectando dos half-adders



Full adder

- Se busca otra forma de expresar el C_{OUT}

C_{IN} \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

ABC' + A'BC + ABC + AB'C =

ABC' + ABC + A'BC + AB'C =

ABC' + ABC + (A ⊕ B) C =

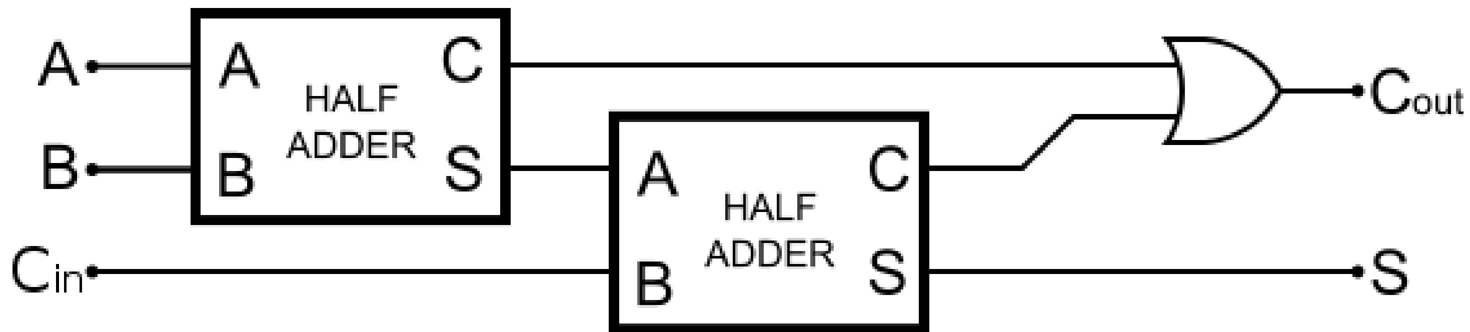
AB (C' + C) + (A ⊕ B) C =

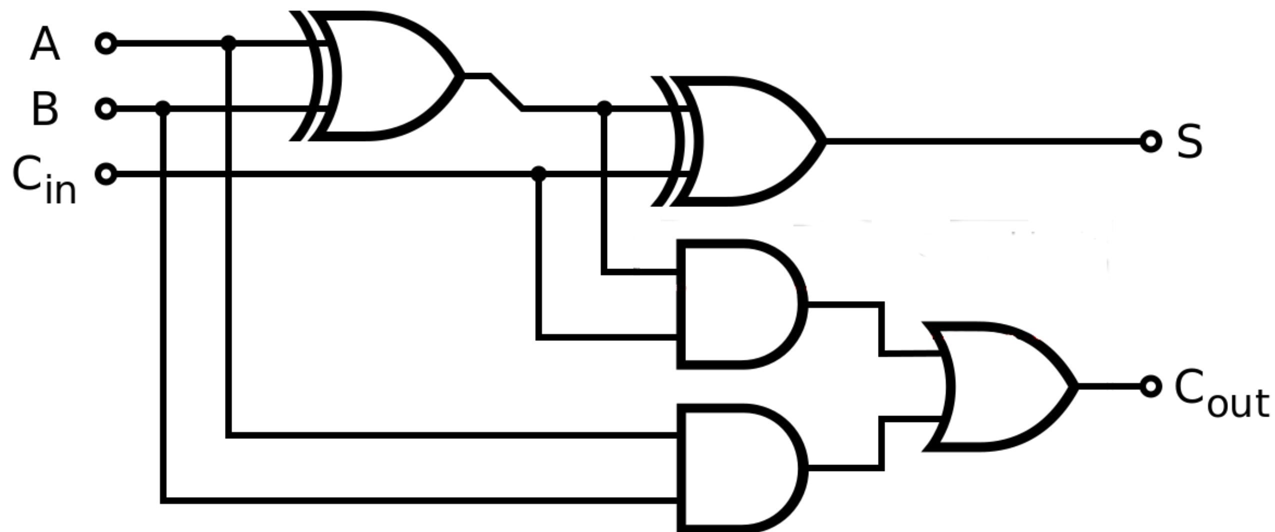
AB + (A ⊕ B) C

$$A \cdot B + (A + B) \cdot C_{IN} \equiv A \cdot B + (A \oplus B) \cdot C_{IN}$$

Full adder

$S \leftarrow A \oplus B \oplus C_{IN}$ $C_{OUT} \leftarrow A \cdot B + (A \oplus B) \cdot C_{IN}$





Sumadores de *n* bits

Un sumador binario es un circuito digital que realiza la suma de dos números binarios.

- Sumador serie
- Sumadores paralelos
  - Ripple adder
  - Carry look-ahead adder
  - Carry skip adder
  - Carry select adder
  - Carry save adder

Sumadores de *n* bits

- Sumador serie
- Sumadores paralelos
  - Ripple adder
  - Carry look-ahead adder
  - Carry skip adder
  - Carry select adder
  - Carry save adder

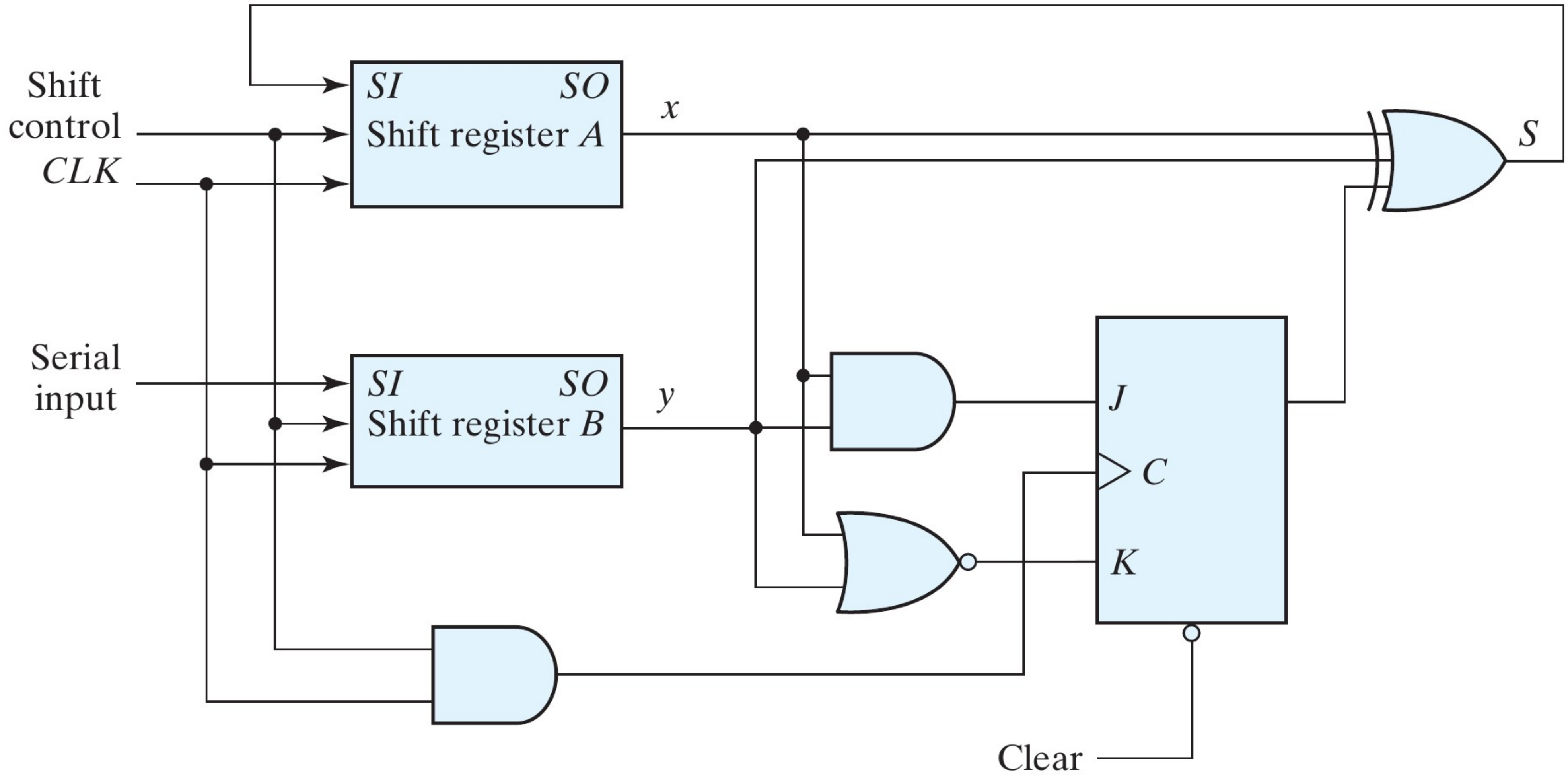
Sumador serie

- Las operaciones en paralelo son más rápidas, pero las operaciones serie requieren menos hardware.
- Se puede construir un sumador binario a partir de un Full adder, dos registros de desplazamiento y un FF D.
- Los operandos A y B se almacenan en los registros de desplazamiento.
- El FF D guarda el carry resultante de la suma de cada par de bits.
- El resultado de la suma se almacena en uno de los registros de entradas.

Sumador serie

- Inicialmente, el registro A y el FF del carry se inicializan en 0.
- Se ubica uno de los operandos en el registro B y con la suma serie se ubica en el registro A.
- Se ubica el otros operando en el registro B.
- La suma se almacena en A, reemplazando el primer operando.

Sumador serie



Sumadores de *n* bits

- Sumador serie
- Sumadores paralelos
  - Ripple adder
  - Carry look-ahead adder
  - Carry skip adder
  - Carry select adder
  - Carry save adder

Sumadores paralelos

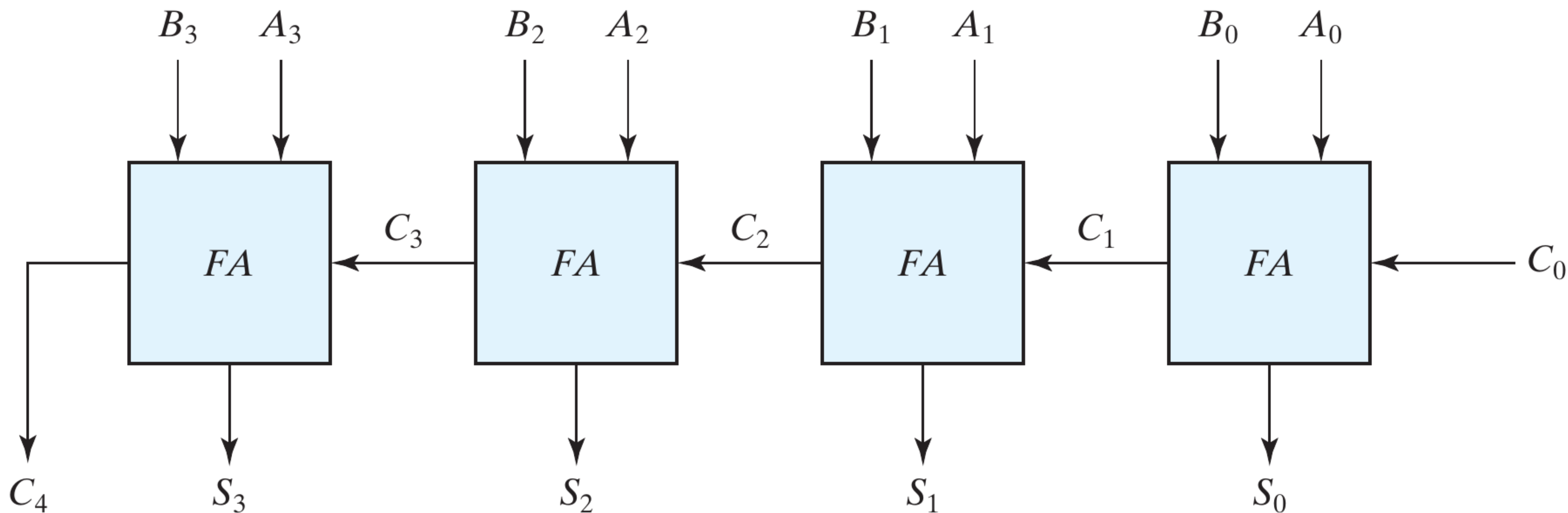
- Todos los 2*n* bits de entrada están disponibles al mismo tiempo.
- El principal problema al construir un sumador paralelo es cómo propagar el carry de un bit al siguiente.
- La forma más simple de resolverlo es con un *ripple adder*.

Sumadores de *n* bits

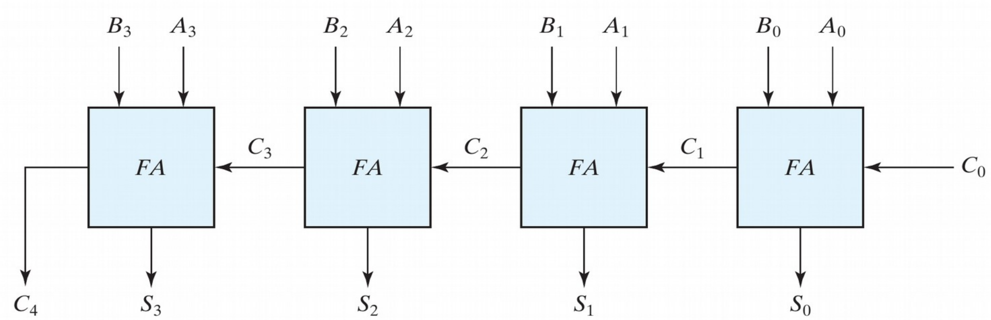
- Sumador serie
- Sumadores paralelos
  - Ripple adder
  - Carry look-ahead adder
  - Carry skip adder
  - Carry select adder
  - Carry save adder

Ripple adder

- Se puede construir un sumador binario paralelo conectando varios full adders en cascada.



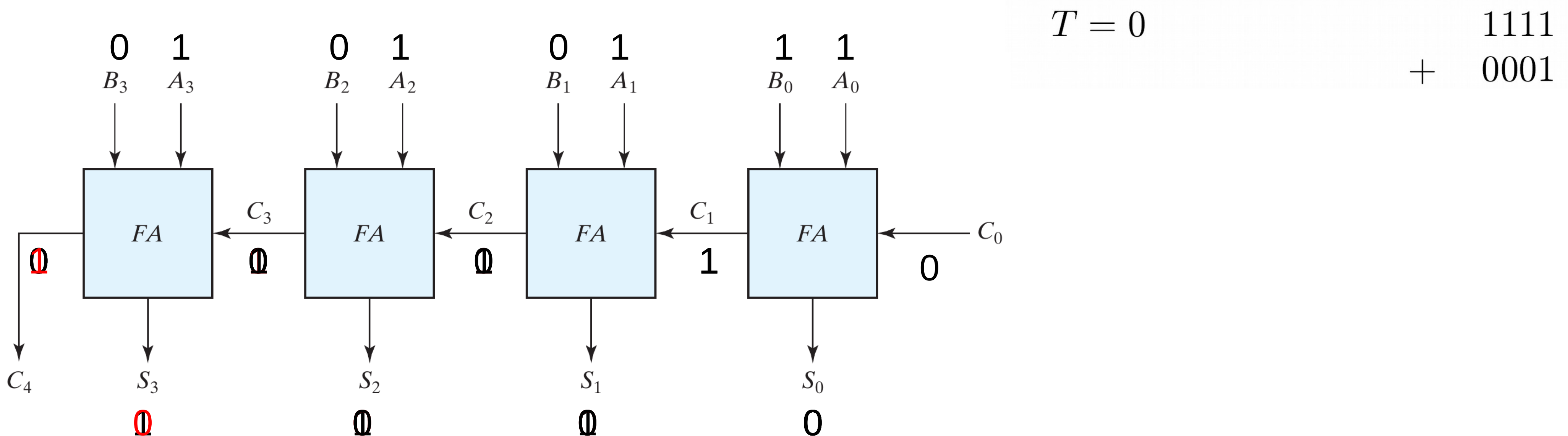
Ripple adder



- El carry debe propagarse desde el FA en la posición 0 (entradas A_0 y B_0) a la posición i antes de que el FA en esa posición pueda producir la suma y el carry correctos.
- El carry debe atravesar los *n* FA antes de que se pueda decir que la salida es la correcta.
- Al ppio de la operación el FA en la posicion i tiene un carry de entrada $c_i = 0$ y produce S_i en función de eso.
- Pero, c_i puede cambiar y eso va a producir un cambio en S_i.

Ripple adder

- Supongamos un FA de retardo Δ_{FA} y el mismo retardo para la suma que para el carry.
- Al suma $A_3A_2A_1A_0$=1111 y $B_3B_2B_1B_0$=0001



Ripple adder

- El tiempo que un circuito demora en producir la salida es proporcional a la máxima cantidad de niveles de compuertas que debe atravesar.
- El tiempo exacto es muy dependiente de la tecnología.
- Lo que vamos a comparar son simplemente los niveles de compuertas y un tiempo estimado de retardo por compuerta.
- Asumiendo compuertas de más de dos entradas, *cualquier expresión lógica puede resolverse en dos niveles de compuertas*.
- Si Δ_G es el retardo de una compuerta, $\Delta_{FA} = 2\Delta_G$

Ripple adder sincrónico

- El tiempo de la suma debe ser el retardo del peor caso: $n\Delta_{FA} = 2n\Delta_{G}$
- El tiempo de suma es fijo. No tiene en cuenta el tiempo *real* de propagación del carry (0101+0010)
- La complejidad de la suma sincrónica es O(*n*).

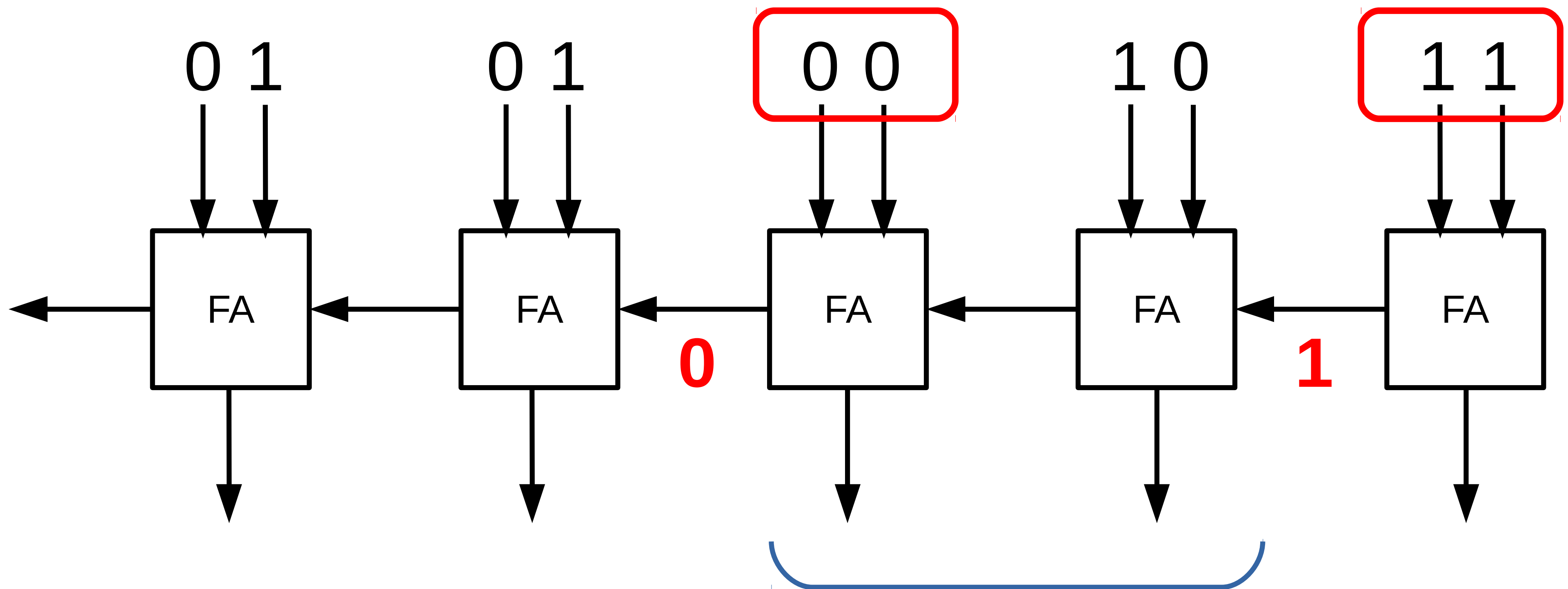
Ripple adder asincrónico

- Todos los FA suman a la vez.
- La suma se estabiliza cuando termina la cadena de propagación de carry más larga.
- En el peor caso la cadena tiene longitud *n*.

✔ La complejidad se reduce a O(log *n*)

✗ Tiempo de terminación variable.

Ripple adder asincrónico



Sumadores de *n* bits

- Sumador serie
- Sumadores paralelos
  - Ripple adder
  - Carry look-ahead adder
  - Carry skip adder
  - Carry select adder
  - Carry save adder

Carry-lookahead adder (CLAA)

- El objetivo es generar todos los carries de entrada en paralelo.
- Sería posible dado que el carry depende de las entradas $A_{n-1} \ldots A_0$ y $B_{n-1} \ldots B_0$ y esa es información disponible inicialmente.
- Es impracticable porque requeriría que todas las etapas del sumador tengan todas las entradas.
- El número de entradas se reduce usando información sobre la generación o propagación del carry en cada etapa.

Propagación del Carry

- Si analizamos la ecuación del carry tiene dos partes:

$$C_{OUT} \leftarrow \underbrace{A \cdot B}_{\text{Generación}} + \underbrace{(A + B) \cdot C_{IN}}_{\text{Propagación}}$$

- Generación de carry:

  El par de bits de genera carry (11)

- Propagación de carry:

  El par de bits no genera carry (10,01) pero propagan el carry de entrada.

  El par 00 no genera ni propaga el carry.

Propagación del Carry

Llamamos

- $G_i = A_i \cdot B_i$ al carry generado
- $P_i = A_i + B_i$ al carry propagado

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

Reemplazando $C_i = G_{i-1} + C_{i-1} \cdot P_{i-1}$ en C_{i+1}

$$C_{i+1} = G_i + G_{i-1} \cdot P_i + C_{i-1} \cdot P_{i-1} \cdot P_i$$

Haciendo más reemplazos $C_{i-1} = G_{i-2} + C_{i-2} \cdot P_{i-2}$

$$C_{i+1} = G_i + G_{i-1} \cdot P_i + G_{i-2} \cdot P_{i-1} \cdot P_i + C_{i-2} \cdot P_{i-2} \cdot P_{i-1} \cdot P_i$$

$$= G_i + G_{i-1} \cdot P_i + G_{i-2} \cdot P_{i-1} \cdot P_i + \ldots + C_0 \cdot P_0 \cdot P_1 \ldots P_i$$

Propagación del Carry

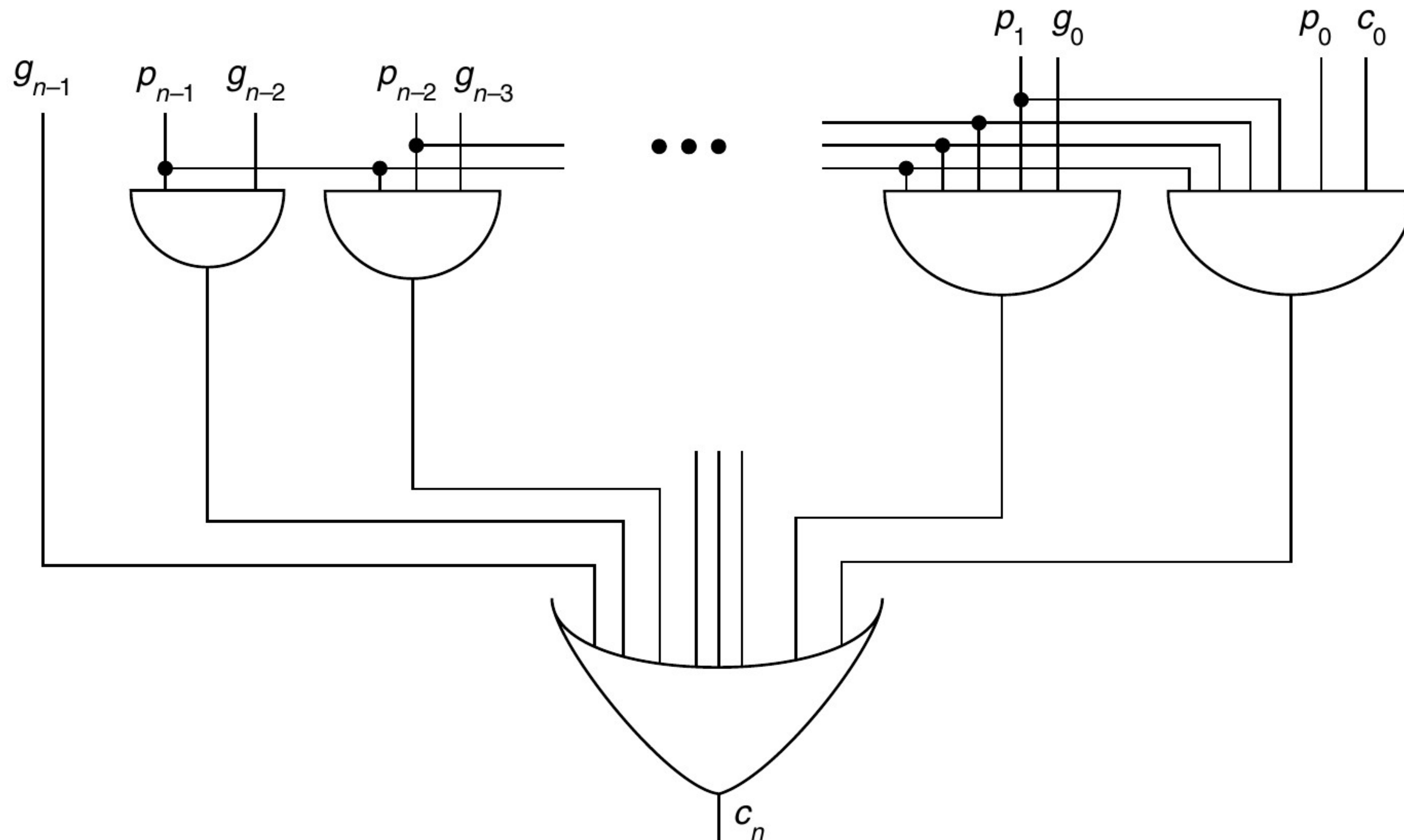
- Se puede calcular el carry de entrada a cualquier posición en función del carry inicial y los carries propagados y generados en las posiciones previas.
- Ahora todos los carries pueden ser calculados en paralelo forzando c_0.
- En un sumador de 4 bits:

$$
\begin{aligned}
c_1 &= G_0 + c_0 P_0, \\
c_2 &= G_1 + G_0 P_1 + c_0 P_0 P_1, \\
c_3 &= G_2 + G_1 P_2 + G_0 P_1 P_2 + c_0 P_0 P_1 P_2, \\
c_4 &= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + c_0 P_0 P_1 P_2 P_3
\end{aligned}
$$

Propagación del Carry

- En general

$$c_n = g_{n-1} + p_{n-1}\, g_{n-2} + \ldots + p_{n-1}\, p_{n-2} \ldots p_1 g_0 + p_{n-1}\, p_{n-2} \ldots p_0 c_0$$



Retardo del CLAA

- Supongamos Δ_G el retardo de una compuerta.

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

- El retardo total será:
  - Δ_G para generar todos los P_i y G_i (una compuerta cada uno).
  - $2\Delta_G$ para genera todos los C_{i+1} en función de los P y G anteriores (dos niveles de compuertas).
  - $2\Delta_G$ para generar la suma S_i a partir de C_i, A_i y B_i.
- En total $5\Delta_G$ (5 niveles de compuertas) para sumar cualquier cantidad de bits.

Retardo del CLAA

CLAA	Ripple Adder
$5\Delta_G$	$2n\Delta_G$
O(1)	O(n)

- En la práctica hay limitaciones en
  - el número de entradas de las compuertas (fan-in): a mayor cantidad de entradas más lenta la compuerta.
  - El número de entradas a los que se puede conectar una salida (fan-out) de manera segura
- La alternativa es aumentar la cantidad de niveles para calcular los P y G.
- No se sacrifica al carry independiente de la suma.

CLAA en varios niveles

- Tecnología MSI
  - CLAA en ripple
  - Carry-lookahead generator (CLAG)
- Tecnología VLSI
  - Lookahead tree adder

CLAA en ripple

- Dividir las *n* etapas en grupos, cada uno con su propio CLAA y conectados en ripple.
- Grupos de igual tamaño beneficia la modularidad y el diseño de un único CI.
- Consideramos grupos de 4 bits:
  - 4 es divisor de la mayoría de los tamaños de palabras.
  - Existe el CI (74F382)

CLAA en ripple

- Para n bits y grupos de 4, se necesitan n/4 grupos.
- Se necesita:
  - 1ΔG para todos los P_i y G_i.

    2ΔG para propagar el carry en un grupo una vez conocidos P_i, G_i y C_0.
  - (n/4)2ΔG = (n/2)ΔG para propagar el carry a través de todos los grupos.
  - 2ΔG para propagar la suma final
- Total: 1ΔG + (n/2)ΔG + 2ΔG = (n/2+3)ΔG
- Sigue siendo O(n), pero es una reducción de casi un 75% frente al ripple adder de 2nΔG

Carry-lookahead sobre grupos

- Además del lookahead interno a cada grupo, se puede proveer el carry generado y el carry propagado grupales.
- G* es el carry generado
- P* es el carry propagado
- Si G* = 1 el grupo genera (internamente) carry de salida.
- Si P* = 1 el carry de entrada al grupo se puede propagar para producir un carry de salida del grupo.

Carry-lookahead sobre grupos

- Para un grupo de 4 bits, teníamos que C_4:

$$C_4 = \underbrace{G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3}_{\text{Generación grupal}} + C_0\underbrace{P_0P_1P_2P_3}_{\text{Propagación grupal}}$$

- Luego los carries generados y propagados grupales son:

$$G^* = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$$

$$P^* = P_0P_1P_2P_3$$

Carry-lookahead generator (CLAG)

Los P* y G* de varios grupos pueden usarse para generar los carries de entrada a cada grupo (similar a los carries de entrada a un único bit).

Carry-lookahead generator: CI que implementa estas ecuaciones.

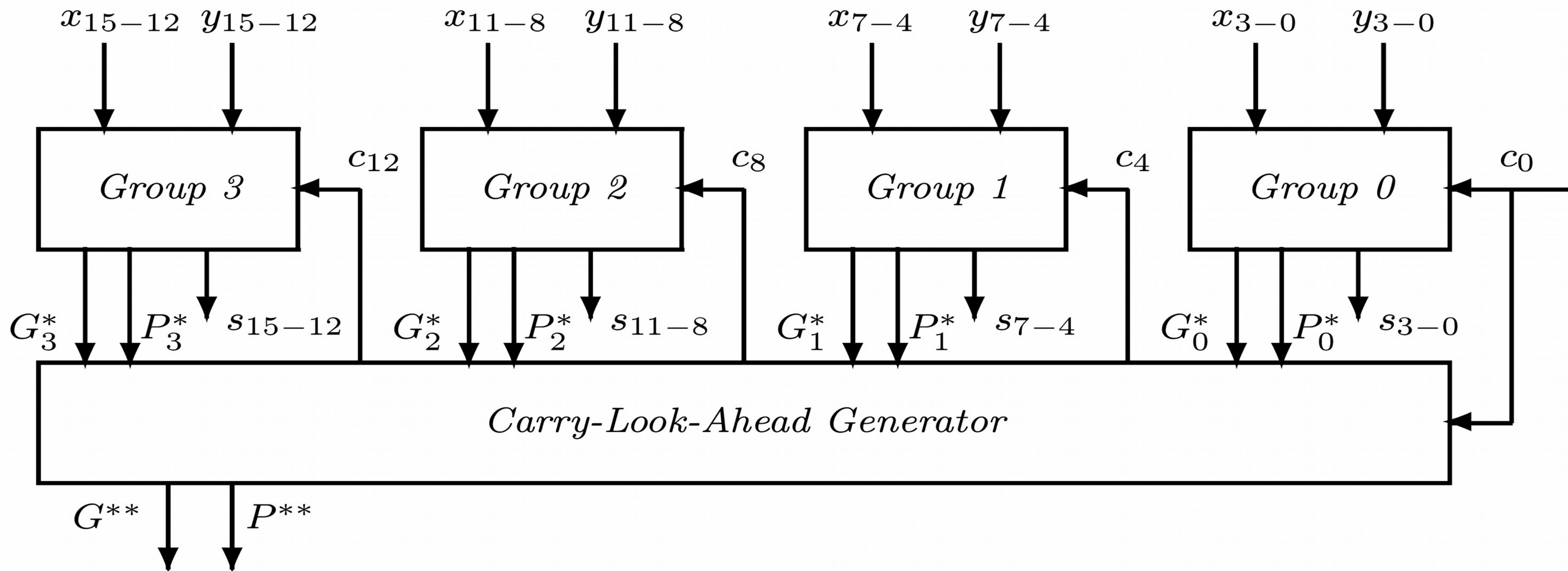
- CI 74F381 CLAA de 4 bits con salidas P* y G*.
- CI 74F182 CLAG con 4 P* y G* de entrada.

CLAA de 16 bits en 2 niveles

- Para n = 16 hay 4 grupos con salidas P_0*, G_0*, P_1*, G_1*, P_2*, G_2* y P_3*, G_3*.
- Estos P* y G* y C_0 son las entradas al CLAG.
- Las salidas del CLAG serán C_4, C_8 y C_{12}

$$
\begin{aligned}
c_4 &= G_0^* + c_0 P_0^*, \\
c_8 &= G_1^* + G_0^* P_1^* + c_0 P_0^* P_1^*, \\
c_{12} &= G_2^* + G_1^* P_2^* + G_0^* P_1^* P_2^* + c_0 P_0^* P_1^* P_2^*
\end{aligned}
$$

CLAA de 16 bits en 2 niveles



CLAA de 16 bits en 2 niveles

Pasos y retardos de la suma:

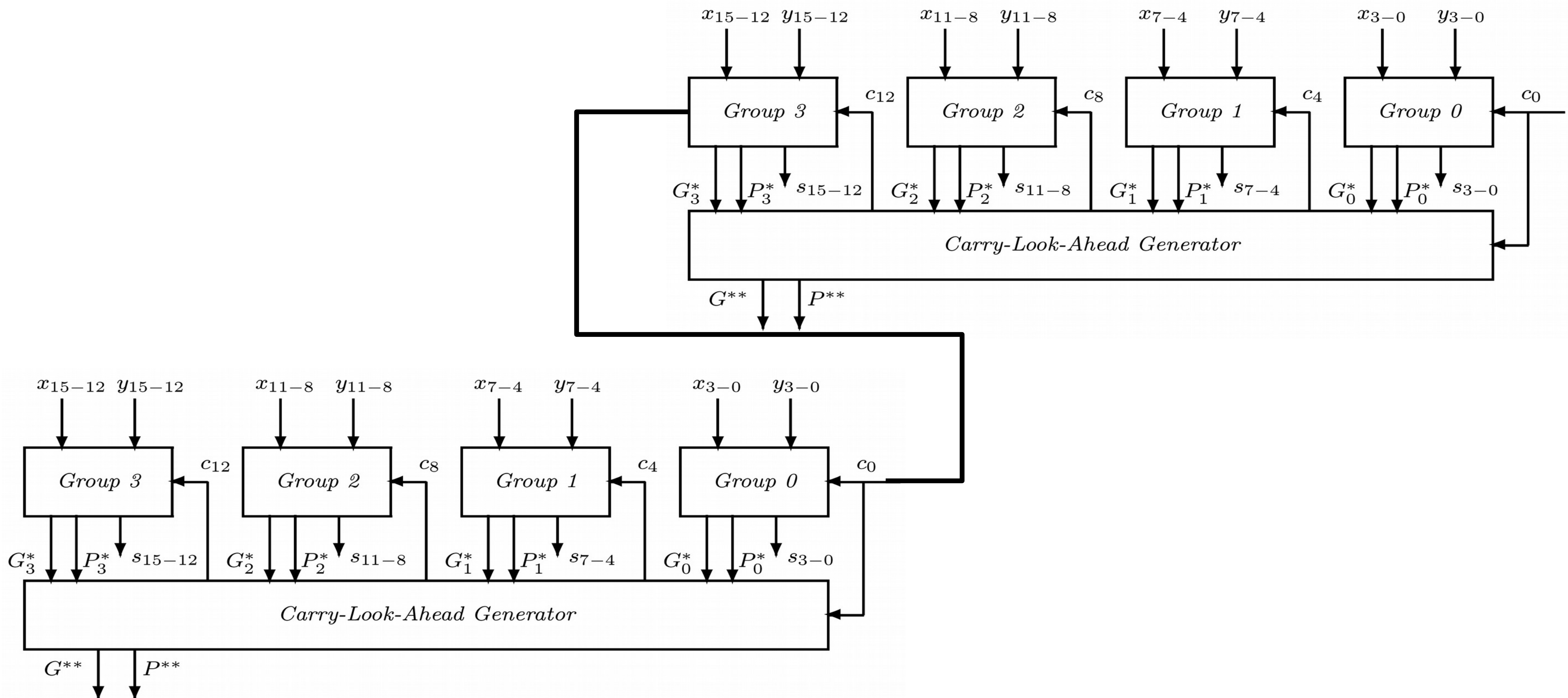
- Retardo Δ_G: Todos los grupos generan en paralelo los 4 P_i y G_i.
- Retardo $2\Delta_G$: Todos los grupos generan en paralelo P* y G*
- Retardo $2\Delta_G$: El CLAG produce los C_4, C_8 y C_{12} de salida, que son entrada a cada uno de los grupos.
- Retardo $4\Delta_G$: cada grupo calcula internamente y en paralelo la suma de los 4 bits ($2\Delta_G$ para generar los bits de carry y $2\Delta_G$ para generar los bits de suma).

Tiempo total: $9\Delta_G$

Suma de 16 bits con 4 CLAA en ripple: $(16/2+3)\Delta_G = 11\Delta_G$

CLAG en ripple

- Para lograr sumas de más bits (32bits) los CLAG se pueden conectar en ripple.



Salidas P** y G** en el CLAG

- El CLAG también produce salidas G y P que representan el carry generado de la sección (G**) y el carry propagado de la sección (P**).
- Las expresiones son equivalentes a generar el P* y G* en un CLAA

$$G^{**} = G^*_3 + G^*_2P^*_3 + G^*_1P^*_2P^*_3 + G^*_0P^*_1P^*_2P^*_3$$

$$P^{**} = P^*_0P^*_1P^*_2P^*_3$$

CLAG en varios niveles

- Estas salidas P** y G** permiten agregar un segundo nivel de CLAG.
- El CLAG en la raíz recibe los 4 pares de G** y P** y produce los carries C_{16}, C_{32} y C_{48}. (suma hasta 64 bits)
- A medida que *n* crece, se pueden agregar más niveles de CLAG.
- Para *n* bits y bloques de *b* bits, se necesitan $\log_b n$ niveles.

CLAG en varios niveles