

# Arquitectura de Computadoras para Ingeniería

(Cód. 7526)  
1° Cuatrimestre 2016

Dra. Dana K. Urribarri  
DCIC - UNS



# Sumadores (continuación)

# Lookahead tree adder

- Generalización de las ecuaciones
- Dado el grupo de bits en las posiciones  $j, j+1, \dots, i$  ( $j \leq i$ ) notamos:
  - $P_{j:i}$  carry propagado
  - $G_{j:i}$  carry generado
- Si  $P_{j:i} = 1$ , el carry de entrada en la posición  $j$  se propaga hasta la posición  $i+1$
- Si  $G_{j:i} = 1$ , el carry se genera en alguna posición entre  $i$  y  $j$  y se propaga hasta  $i+1$
- $G_{j:i} = G_i$  y  $P_{j:i} = P_i$

# Lookahead tree adder

Las funciones P y G pueden calcularse usando las ecuaciones:

$$P_{j:i} = \begin{cases} P_i & \text{Si } i = j \\ P_i \cdot P_{j:i-1} & \text{Si } j < i \end{cases}$$

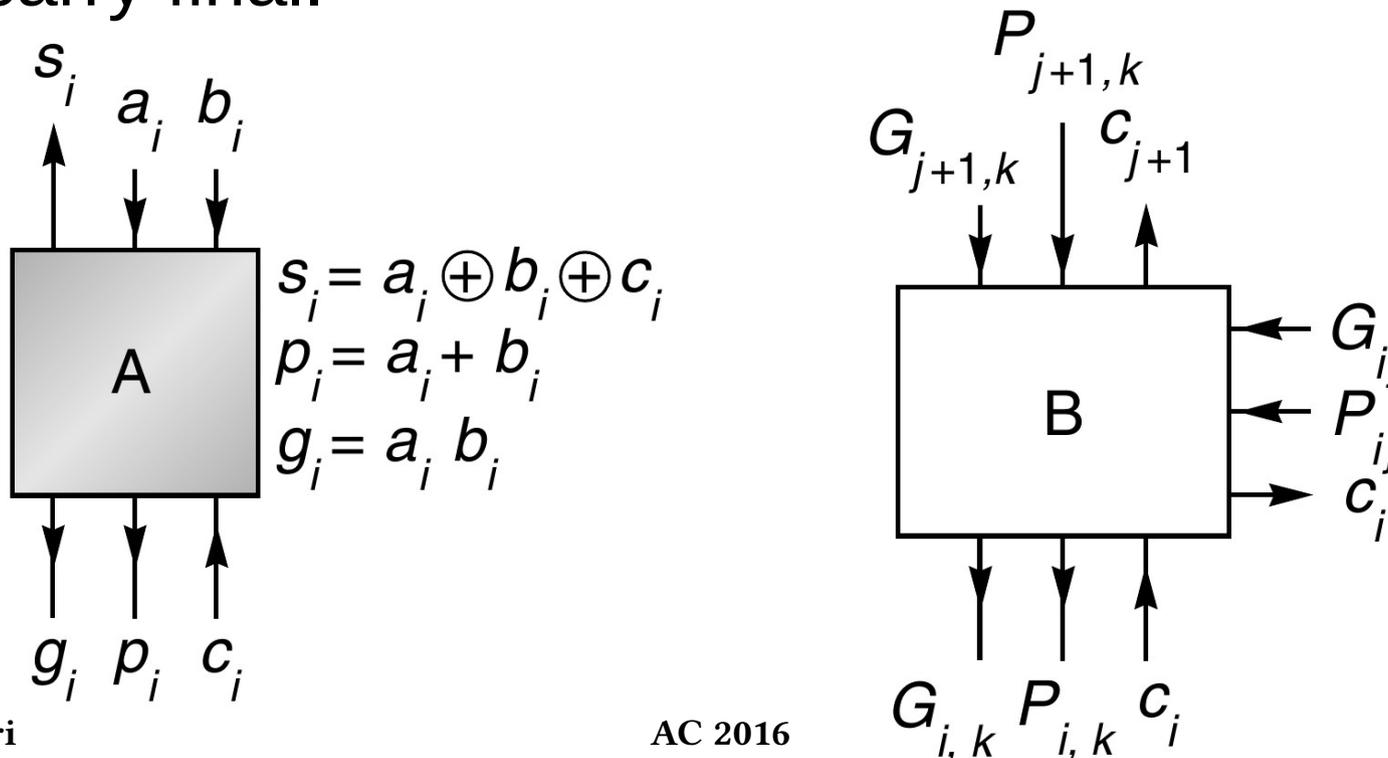
$$G_{j:i} = \begin{cases} G_i & \text{Si } i = j \\ G_i + P_i \cdot G_{j:i-1} & \text{Si } j < i \end{cases}$$

De manera recursiva pueden generalizarse a ( $j+1 \leq m \leq i$ ):

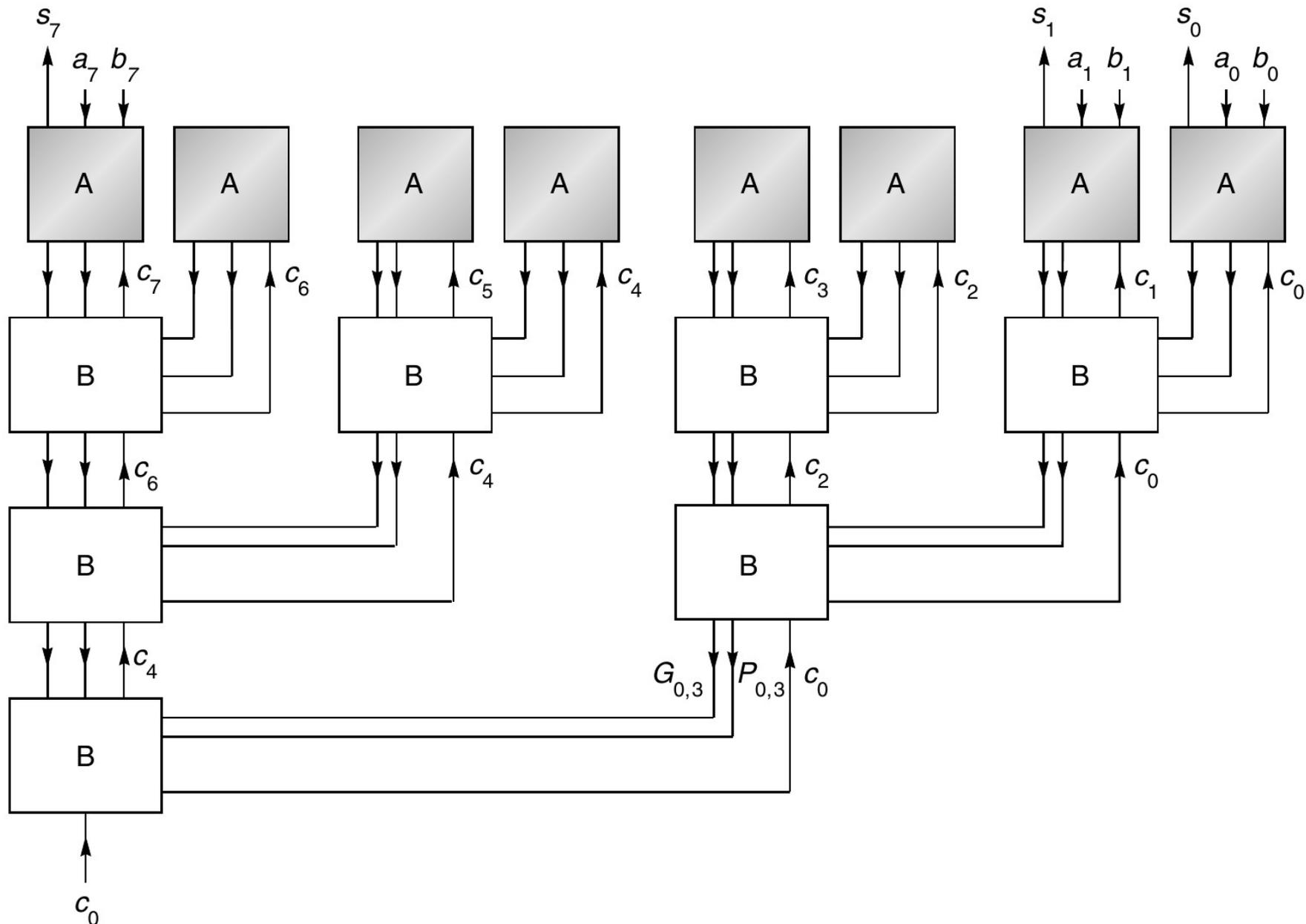
- $P_{j:i} = P_{m:i} \cdot P_{j:m-1}$
- $G_{j:i} = G_{m:i} + P_{m:i} \cdot G_{j:m-1}$

# Lookahead tree adder

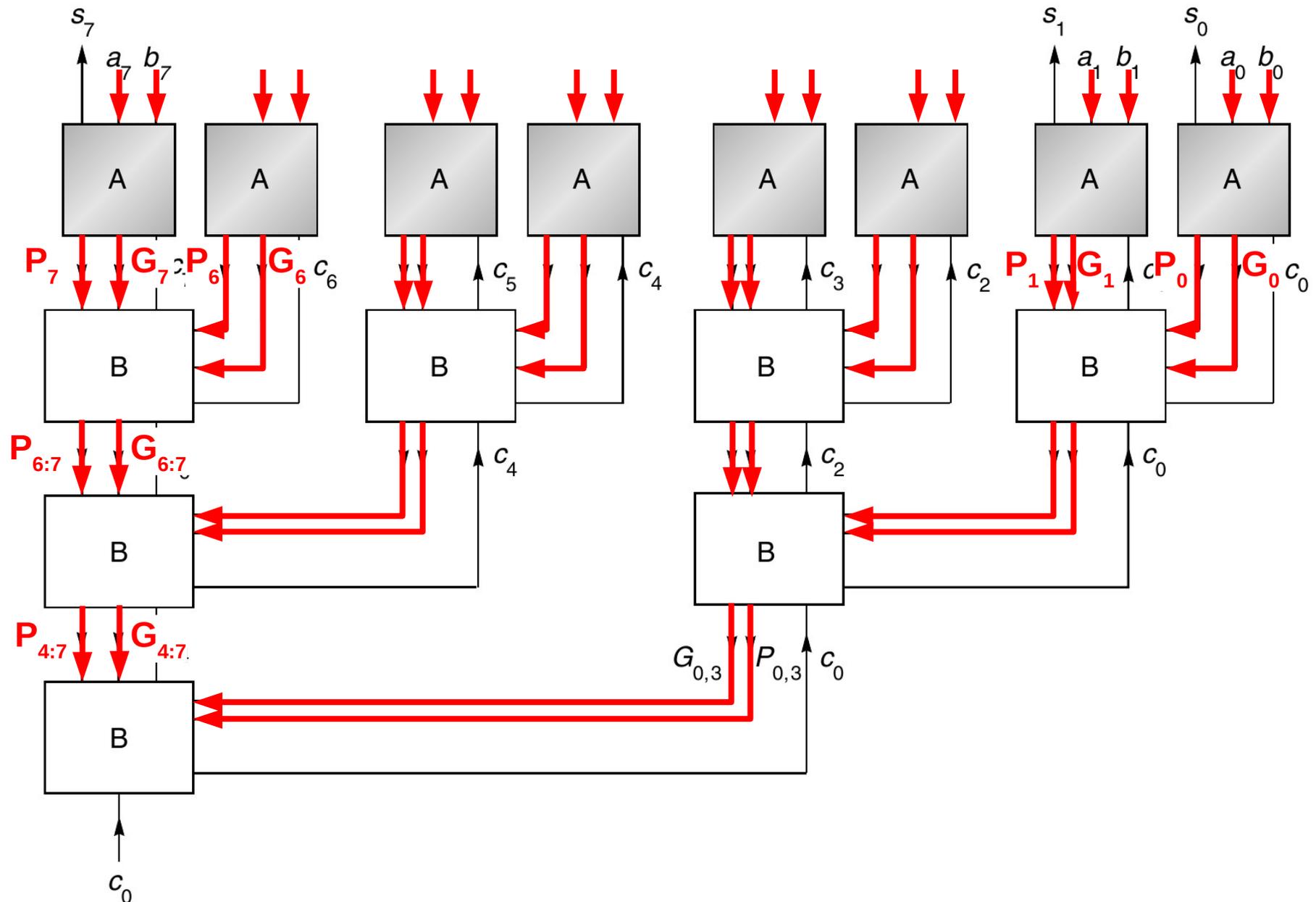
- Tenemos dos tipos de bloques:
  - A: a partir de los bits  $a_i$ ,  $b_i$  y  $c_i$  calcula:  $p_i$ ,  $g_i$  y  $s_i$
  - B: a partir de los P y G de dos grupos distintos consecutivos, calcula los P y G del grupo completo y el carry final.



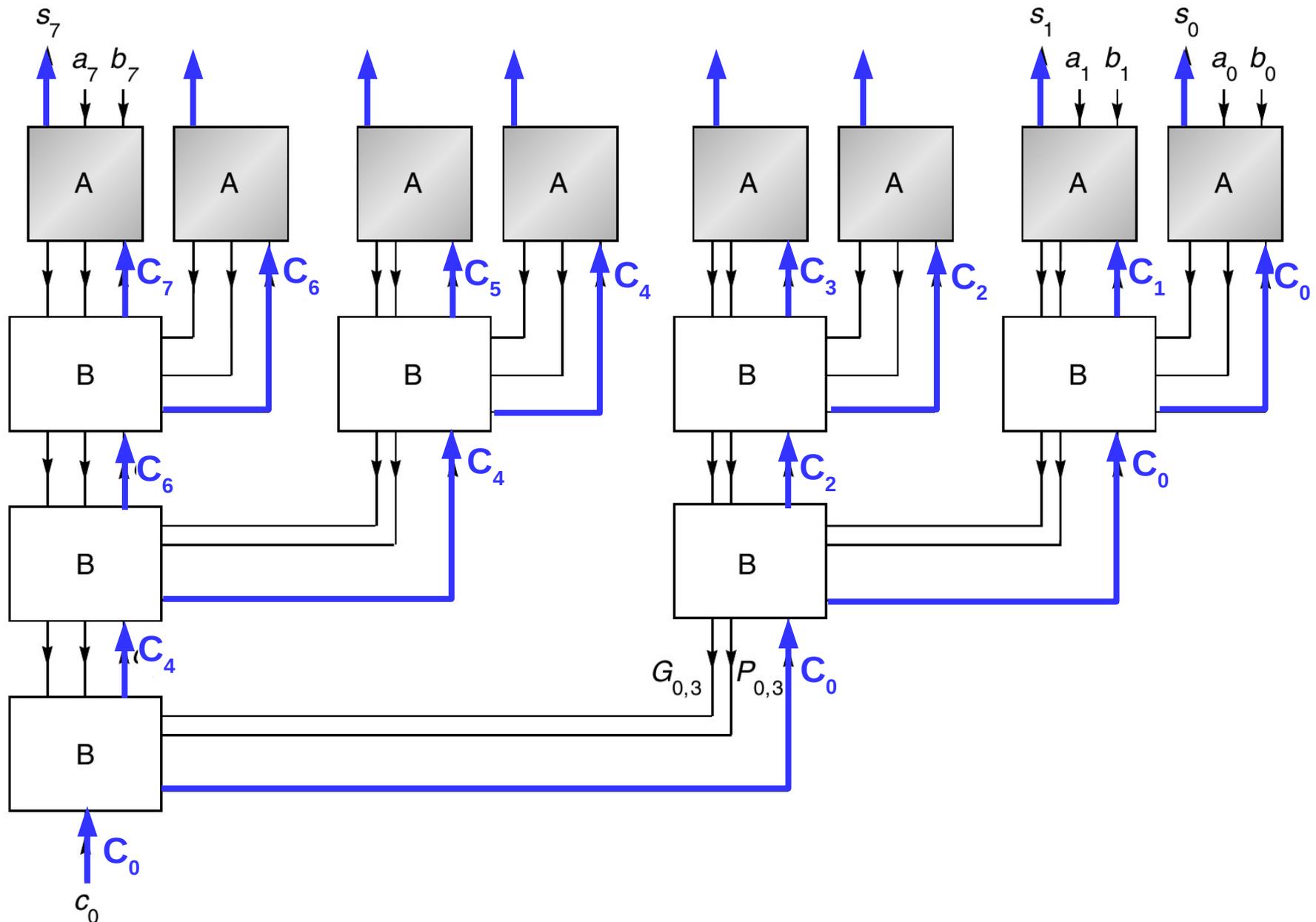
# Lookahead tree adder



# Lookahead tree adder



# Lookahead tree adder



# Lookahead tree adder

- Gran mejora en performance:
  - Los bits deben atravesar en el orden de  $\log_2 n$  niveles lógicos, comparados con los  $2n$  del ripple.
- Incremento en el tamaño:
  - $2n$  celdas en un espacio  $n \log_2 n$  (nodos de dos entradas) frente a  $n$  celdas en el ripple.

# Sumadores de $n$ bits

- Sumador serie
- Sumadores paralelos
  - Ripple adder
  - Carry-lookahead adder
  - Carry-skip adder
  - Carry-select adder
  - Carry-save adder

# Carry-skip adder

- La tecnología ripple es simple y rápida.
- Pero no tiene buen rendimiento para grandes valores de  $n$ .
- Soluciones intermedias son
  - Carry-skip adder
  - Carry select adder

# Carry-skip adder

- Divide los operandos en grupos no necesariamente iguales.
- Reduce el tiempo de propagación del carry saltando sobre grupos consecutivos.
- Nota:
  - Se hizo popular con la tecnología VLSI.
  - Para los tamaños de palabras usuales, la velocidad es comparable a las técnicas de CLA. Además, requiere menos área y consume menos.

# Carry-skip adder

- El funcionamiento se basa en que:
  - La propagación del carry al sumar los bits  $i$  puede saltarse si  $x_i \neq y_i$  ( $P_i = x_i \oplus y_i = 1$ )
  - La propagación del carry puede saltarse la suma de varios bits consecutivos si todos satisfacen  $x_i \neq y_i$ .
- Un sumador de  $n$  bits puede dividirse en grupos de bits consecutivos.
- Cada grupo suma los  $m$  bits con un esquema ripple.

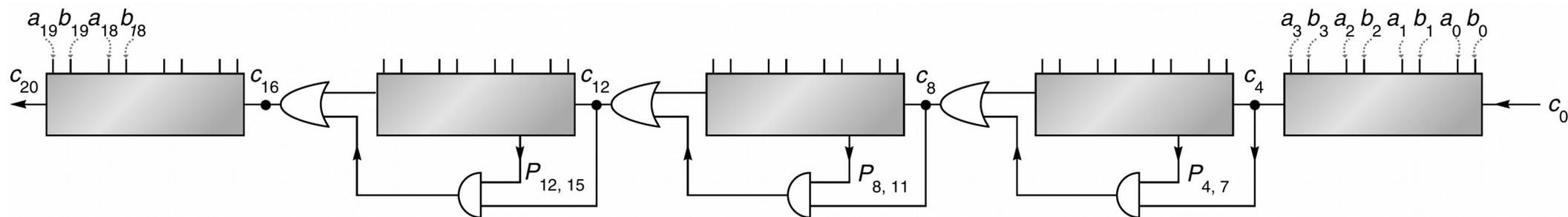
# Carry-skip adder

- El cálculo de  $P$  es más simple que el cálculo de  $G$ .
  - Cada grupo calcula sólo  $P_m$ 
    - $P_m = 1$  si el grupo propaga el carry de entrada. De esta forma, permite que al carry de entrada saltear el grupo.
  - Si un bloque genera carry, habrá carry de salida aunque el carry de entrada no sea el correcto.
  - Si todos los carry de entrada a los grupos se inicializan en cero, el carry de salida puede interpretarse como  $G$ .
- El carry-skip adder es práctico sólo si esto puede hacerse de manera fácil al comienzo de cada operación.



# Carry-skip adder

- El carry-out del bloque  $i$  es el carry-in del bloque  $i+1$ .
- Luego, si el bloque  $i+1$  propaga carry ( $P_{i+1} = 1$ ) y hay carry-in ( $C_{i+1} = 1$ ), ya se sabe que habrá carry-in en el bloque  $i+2$  ( $C_{i+2} = 1$ ).



# Tamaño de grupos óptimo

Todos los grupos de igual tamaño  $k$  y  $n$  múltiplo de  $k$

- $k$  debe minimizar la cadena de propagación de carry más larga.
- La propagación más larga se da cuando el carry se genera en el bloque 1 (ripple), luego se propaga por los bloques 2,3,..., $n/k-1$  y finalmente hace ripple en el bloque  $n/k$ .

$$T = t_{\text{Ripple en } k \text{ bits}} + (n/k - 2) (t_{\text{AND}} + t_{\text{OR}}) + t_{\text{Ripple en } k \text{ bits}}$$

# Tamaño de grupos óptimo

$$T = \underbrace{t_{\text{Ripple en k bits}}}_{\text{Primer sumador}} + \underbrace{(n/k - 2) (t_{\text{AND}} + t_{\text{OR}})}_{\text{Tiempo de saltar los sumadores del medio}} + \underbrace{t_{\text{Ripple en k bits}}}_{\text{Último sumador}}$$

Si cada compuerta tiene retardo  $\Delta_G$

El tiempo del ripple es  $2\Delta_G \times$  cantidad de bits en ripple ( $k$ )

- $$T = 2k\Delta_G + (n/k - 2) (\Delta_G + \Delta_G) + 2k\Delta_G$$

$$= \Delta_G (4k + 2n/k - 4)$$

# Tamaño de grupos óptimo

$$T = \Delta_G (4k + 2n/k - 4)$$

Para hallar el óptimo  $k$ , se deriva  $T$  con respecto a  $k$  y se iguala a 0.

- $dT/dk = 4\Delta_G - 2n\Delta_G/k^2 = 0$
- $k = \sqrt{(n/2)}$

El tamaño del grupo óptimo y el tiempo de propagación del carry son proporcionales a  $\sqrt{n}$ .

Para 32 bits, el óptimo es  $k = 4$

$$T = 28\Delta_G \text{ contra } T_{\text{ripple}} = 64\Delta_G$$

# Tamaño de grupos óptimo

Asumiendo bloques de diferentes tamaños

- El tiempo de saltar un sumador no depende de la cantidad de bits a sumar.
- Se puede optimizar:
  - Incrementando el tamaño de los grupos centrales
  - Reduciendo el tamaño del primer y del último grupo.

# Sumadores de $n$ bits

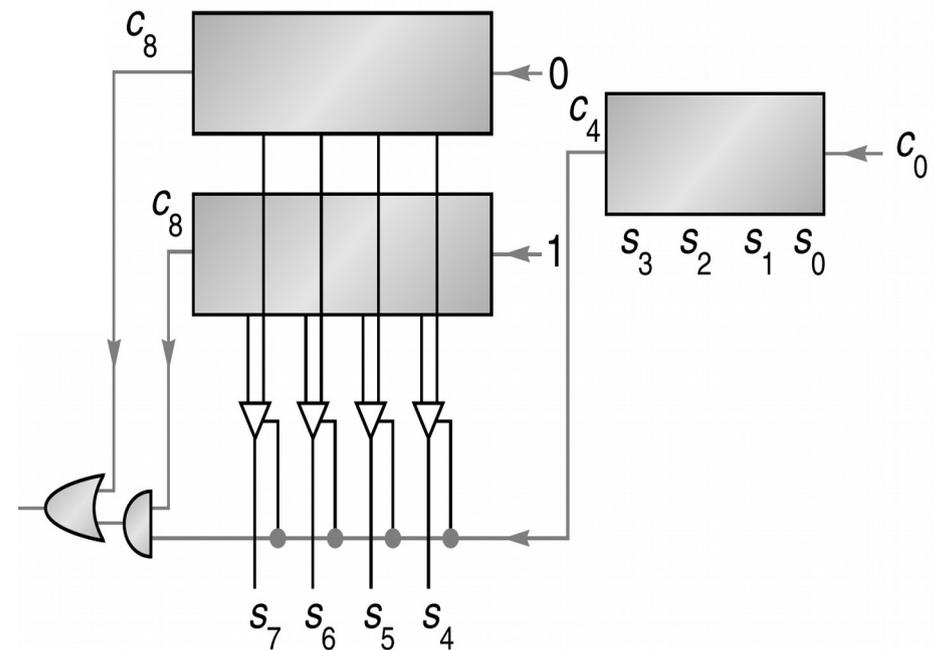
- Sumador serie
- Sumadores paralelos
  - Ripple adder
  - Carry-lookahead adder
  - Carry-skip adder
  - Carry-select adder
  - Carry-save adder

# Carry-select adder

- Los  $n$  bits a sumar se dividen en grupos (pueden ser de diferentes tamaños)
- Cada grupo genera dos sumas y dos carries.
  - Una suma y un carry se corresponden a la suma con carry inicial 0.
  - La otra suma y el otro carry se corresponden a la suma con carry inicial 1.
- Cuando se tiene el carry de entrada real, se selecciona la suma con su carry correspondientes.

# Carry-select adder

- Cada bloque calcula las sumas en ripple.
- Una vez que termina el primer bloque, y se tiene el carry real, se seleccionan los carries de los bloques siguientes.
- Para bloques de tamaño  $k$ 
  - $2k\Delta_G$  para calcular todas las sumas
  - $2\Delta_G(n/k - 1)$  para seleccionar el carry correcto y multiplexar (2 niveles de compuertas) la suma.
- Total:  $2k\Delta_G + 2\Delta_G(n/k - 1) = (2k + 2n/k - 2)\Delta_G$



# Tamaño de grupos óptimo

- Sin embargo, para el Carry-select adder el mejor diseño es con bloques de tamaño variable:
  - Si a cada bloque le lleva  $2k\Delta_G$  realizar la suma de números de  $k$  bits
  - Y seleccionar el carry correcto lleva  $2\Delta_G$
  - Entonces, lo ideal es que cada bloque sea un bit más ancho que el anterior.
- Los grupos deberían seguir la serie  $1, 1, 2, 3, \dots$

# Tamaño de grupos óptimo

- Los grupos deberían seguir la serie 1,1,2,3,...
- Si  $L_{\max}$  es el tamaño del mayor bloque, para  $n$  bits se debe satisfacer que:

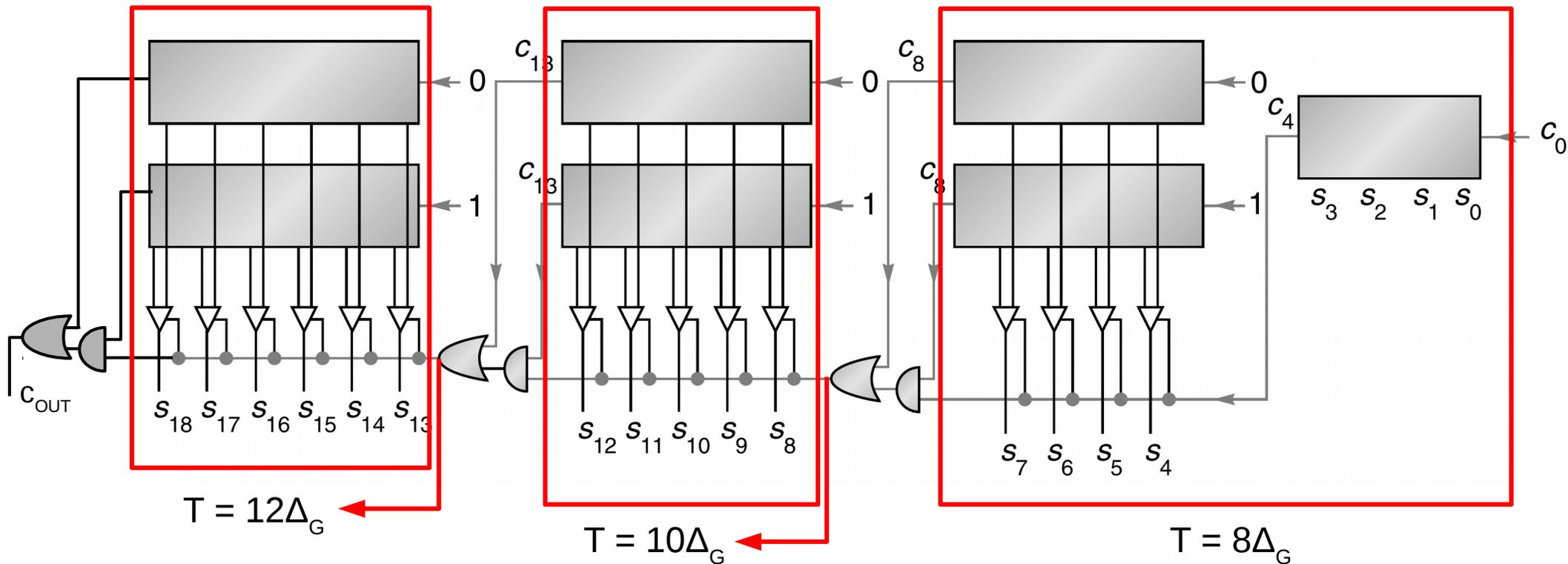
$$1 + L_{\max} (L_{\max} - 1) / 2 > n$$

- Por lo tanto

$$L_{\max} (L_{\max} - 1) > 2(n - 1)$$

- El tamaño del mayor grupo y el tiempo de ejecución son del orden de  $\sqrt{n}$

# Carry-select adder



$T = 0$  → Comienza

$T = 8\Delta_G$  → Terminan el primero y el segundo bloque

$T = 10\Delta_G$  → Terminan el tercer bloque y ya se calculó el carry out del segundo

$T = 12\Delta_G$  → Termina el cuarto bloque y ya se calculó el carry out del tercero.  
Luego del multiplexado (despreciable), ya está la suma final.

$T = 14\Delta_G$  → Calculó el carry out final

# Sumadores de $n$ bits

- Sumador serie
- Sumadores paralelos
  - Ripple adder
  - Carry-lookahead adder
  - Carry-skip adder
  - Carry-select adder
  - Carry-save adder

# Carry-save adder (CSA)

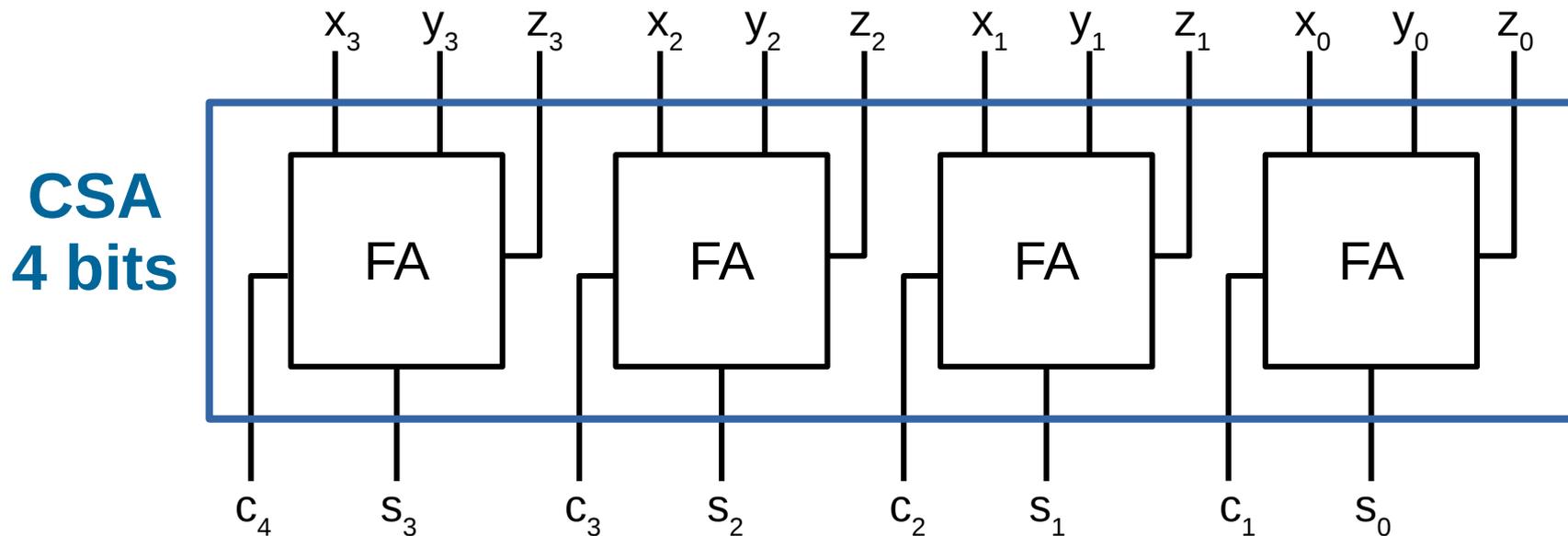
- Sumar más de 2 operandos simultáneamente (como en la multiplicación) usando sumadores de dos operandos es costoso:
  - Para  $k$  operandos la propagación del carry debe repetirse  $k - 1$  veces
- La estrategia del Carry-save adder reduce ese costo.
  - Genera sumas parciales y secuencias de carry
  - Propaga el carry sólo en el último paso.

# Carry-save adder (CSA)

- El CSA recibe 3 operandos de  $n$  bits y genera dos resultados  $n$  bits:
  - La suma parcial de  $n$  bits
  - El carry de  $n$  bits.
- Reduce la suma de 3 operandos a la suma de 2 operandos.
- La implementación más simple es con  $n$  FA en paralelo.

# Carry-save adder (CSA)

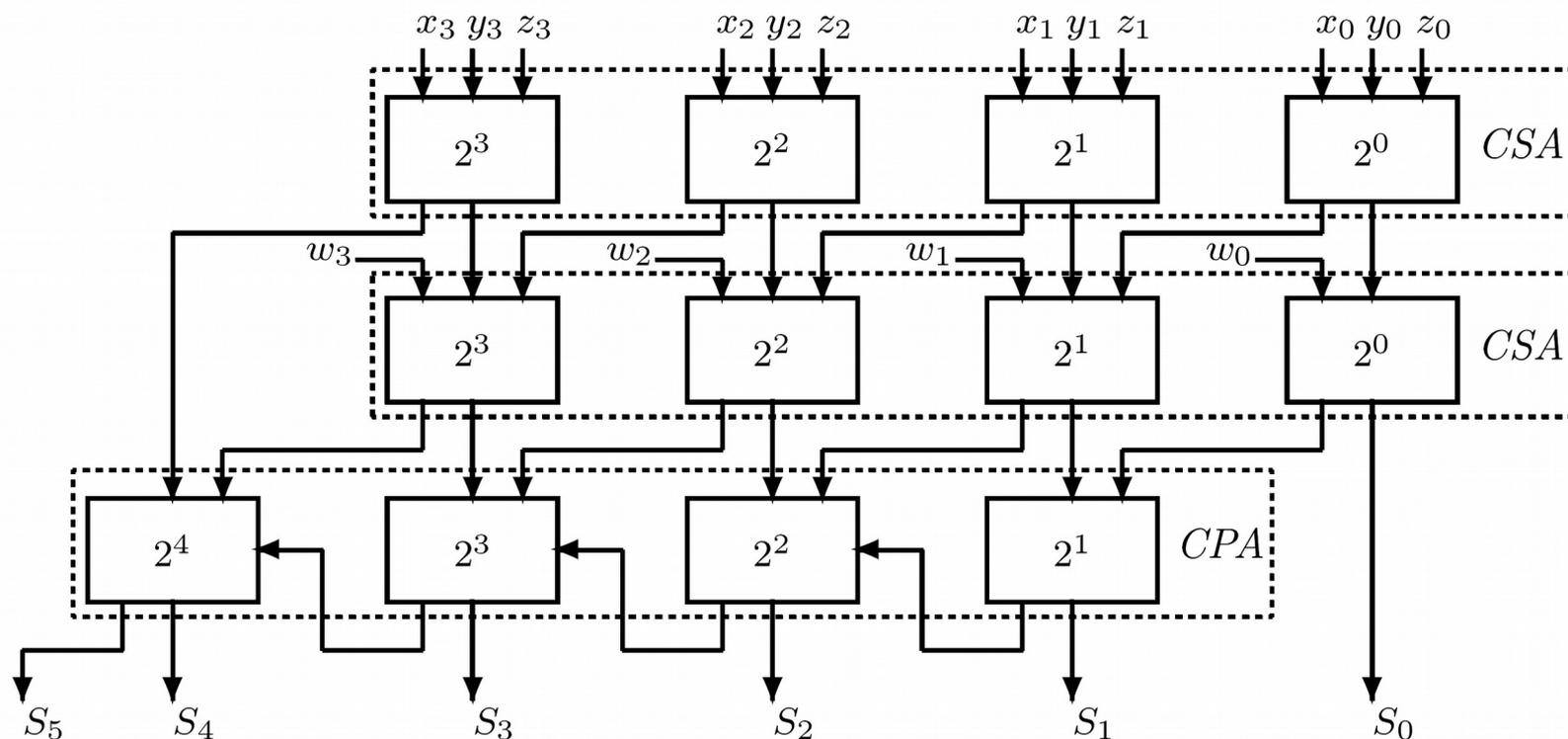
- La implementación más simple es con  $n$  FA en paralelo.



$$\begin{array}{r} + \quad S_3 S_2 S_1 S_0 \\ C_4 C_3 C_2 C_1 C_0 \end{array} \quad \rightarrow \quad \begin{array}{r} + \quad S_3 S_2 S_1 S_0 \\ C_4 C_3 C_2 C_1 0 \end{array}$$

# Carry-save adder (CSA)

- Sumador de 4 operandos ( $x, y, z, w$ ) de 4 bits cada uno usando dos niveles de CSA.
- El último nivel debe ser un sumador paralelo que propague carry (CPA: ripple, lookahead, skip, etc)



# Carry-save adder (CSA)

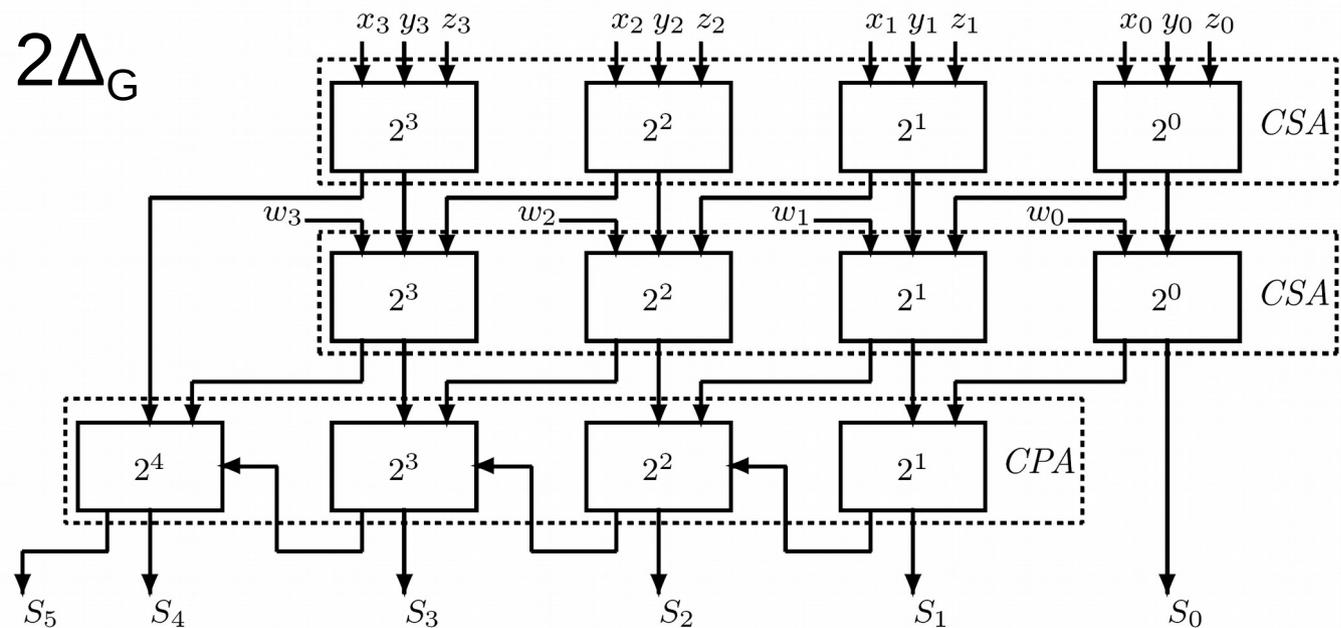
Sumar  $k$  operandos de  $n$  bits

- Requiere  $k - 2$  CSA y un CPA
- El tiempo de suma será:

$$(k - 2) T_{\text{CSA}} + T_{\text{CPA}}$$

-  $T_{\text{CPA}} =$  retardo del sumador paralelo.

-  $T_{\text{CSA}} = \Delta_{\text{FA}} = 2\Delta_{\text{G}}$



# Carry-save adder (CSA)

La suma de  $k$  operandos de  $n$  bits, puede valer hasta  $(2^n - 1) k$ .

El resultado final puede tener hasta  $n + \lceil \log_2 k \rceil$  bits

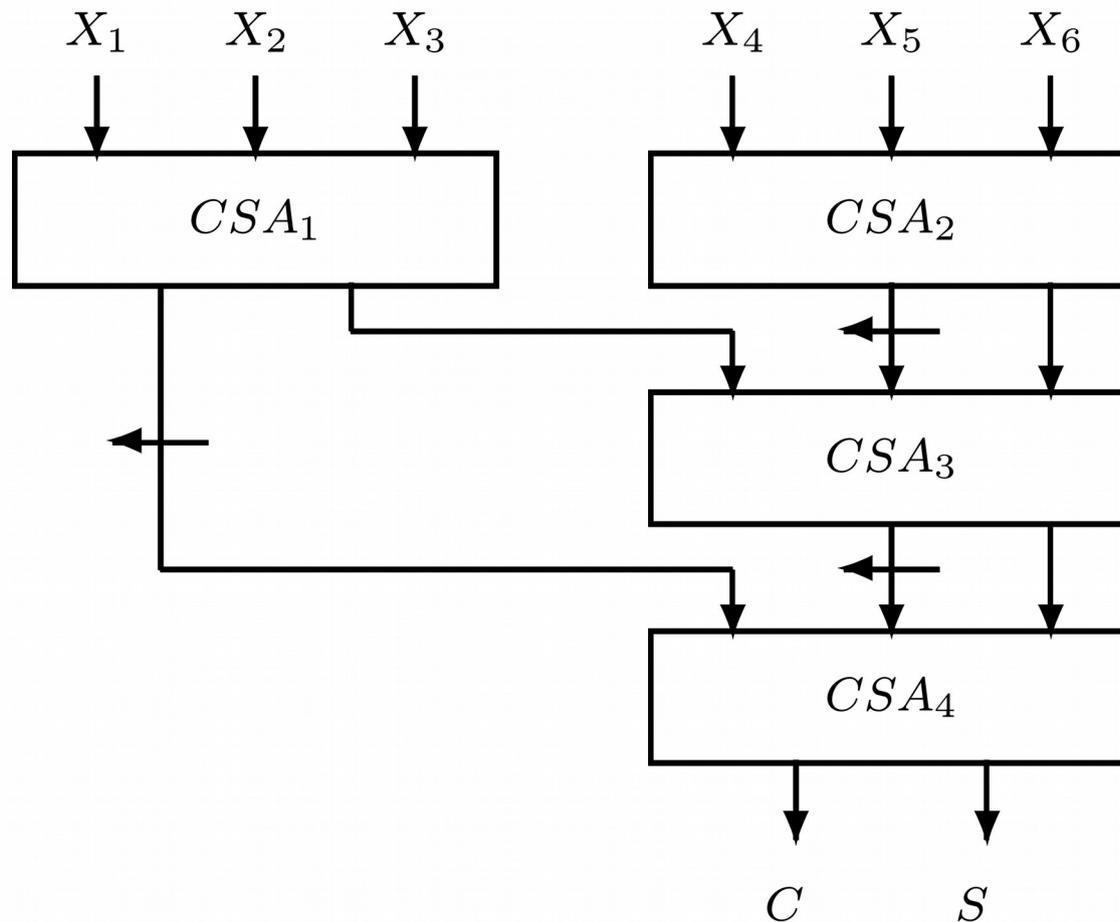
El tiempo total de suma:

$$(k - 2) 2\Delta_G + T_{CPA}(n + \lceil \log_2 k \rceil)$$

Dependerá del sumador paralelo de la última etapa.

# Carry-save adder (CSA)

- La mejor organización de CSA es el Wallace tree



# Carry-save adder (CSA)

- En cada nivel, el número de operandos se reduce en  $2/3$ .
- Por lo tanto, si  $L$  es la cantidad de niveles requeridos

$$k (2/3)^L \leq 2$$

- Es decir que se necesitarán una cantidad estimada de niveles:

$$\text{Log } (2/k) / \text{Log } (2/3)$$

# Restadores

- Los sumadores pueden sumar números signados o no signados.
- Con un sumador puede realizarse la resta usando complemento a la base.
  - Puede implementarse circuito restador que realice la operación directamente.
- Para calcular  $A - B$ 
  - 1) Se complementa  $B$  :  $X \leftarrow B'$
  - 2) Se calcula  $A + X$  forzando el carry inicial  $c_0$  a 1.

# Bibliografía

- *Capítulo 5. Computer Arithmetic Algorithms.* Israel Koren, 2da Edición, A K Peters, Natick, MA, 2002.  
*Adapted from Koren, UMass. Copyright 2008 Koren, UMass and A.K. Peters.*
- *Capítulo 6. M. Morris Mano & Michael D. Celetti. Digital Design: With an Introduction to the Verilog HDL.* Pearson. (2015, 5ta Ed.)

## Suplementaria

- *Apéndice J.* J. Hennessy & D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers INC. 2011, 5ta Ed.
- *Apéndice B.* David A. Patterson & John L. Hennessy. *Computer Organization and Design. The Hardware/Software Interface.* Elsevier. (5ta Ed. 2014)