DIPARTIMENTO DI ELETTRONICA E INFORMAZIONE

POLITECNICO
DI MILANO

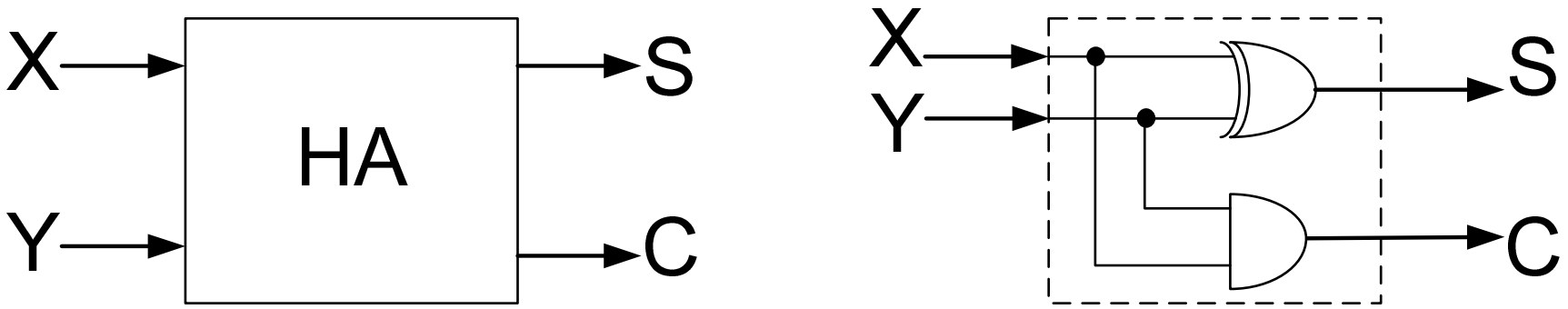
Aritmetica dei calcolatori





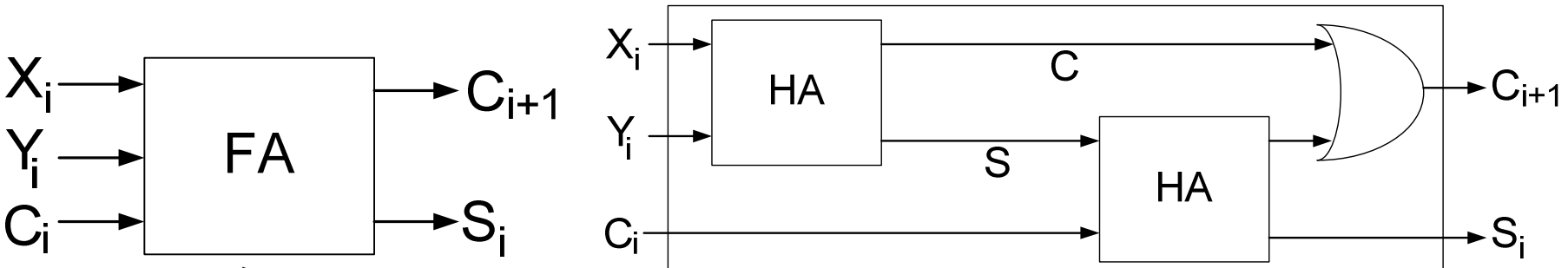
- Addizione a propagazione di riporto
- Addizione veloce
- Addizione con segno
- Moltiplicazione

Half adder



X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

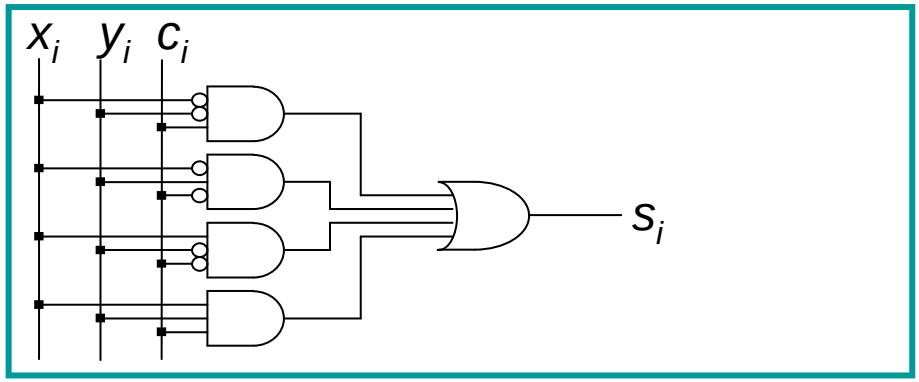
Full Adder



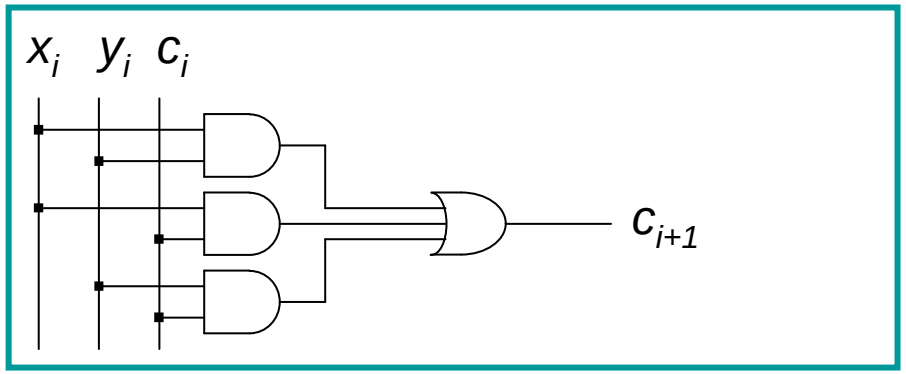
X_i	Y_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

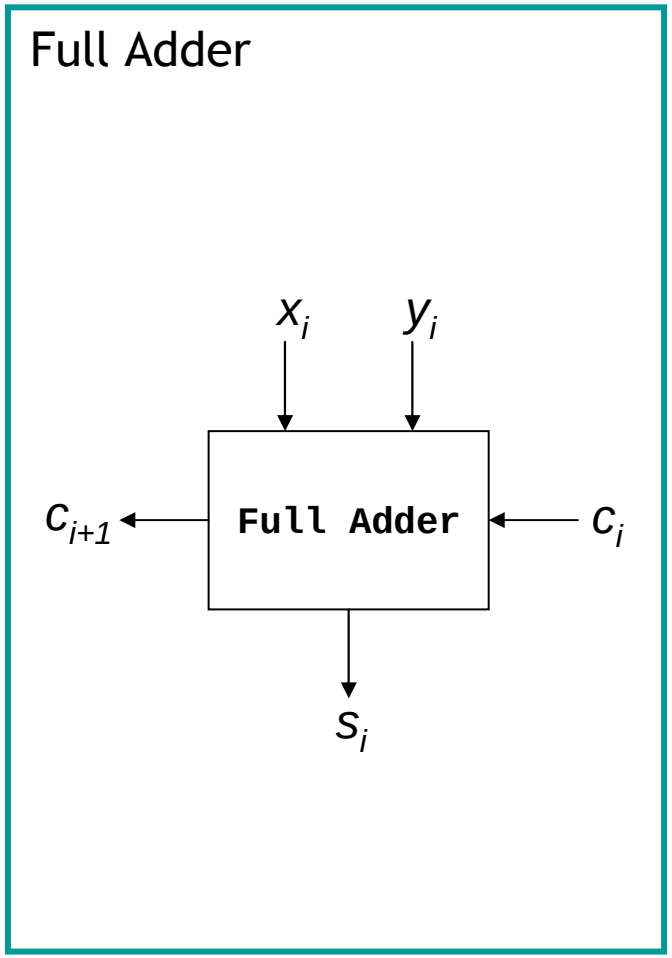
Full Adder - espressione

$$s_i = \overline{x}_i \overline{y}_i c_i + \overline{x}_i y_i \overline{c}_i + x_i \overline{y}_i \overline{c}_i + x_i y_i c_i$$



$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$





Addizione senza segno

bit-slice a propagazione di riporto

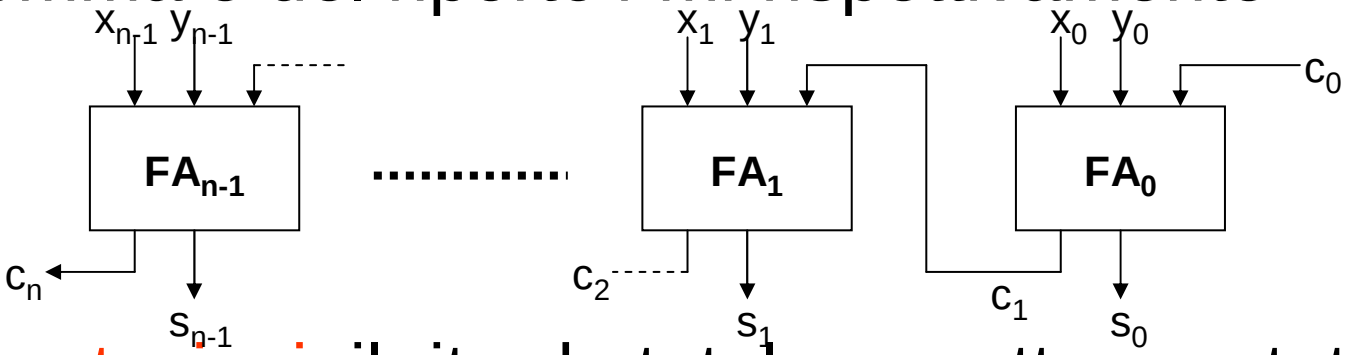
- Un sommatore *bit-slice ripple carry* è strutturato in modo che il modulo in posizione *i*-esima:
  - Riceve in ingresso i bit x_i e y_i degli operandi
  - Riceve in ingresso il riporto c_i del modulo precedente
  - Produce la somma $s_i = x_i'y_i'c_i + x_i'y_ic_i' + x_iy_i'c_i' + x_iy_ic_i$
    - $= (x_i \text{ xor } y_i)'c_i + (x_i \text{ xor } y_i)c_i' = x_i \text{ xor } y_i \text{ xor } c_i$
  - Produce il riporto $c_{i+1} = x_iy_i + x_ic_i + y_ic_i$
    - $= x_i\, y_i + (x_i \text{ xor } y_i)c_i$
- Il modulo in posizione 0 ha il bit di riporto $c_0=0$
- Il riporto c_0 può essere sfruttato per sommare il valore 1
  - Necessario per il calcolo del complemento a 2
- La somma di numero ad *n* bit richiede un tempo pari ad *n* volte circa quello richiesto da un modulo di somma

c_i	x_i	y_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

Addizione senza segno *ripple-carry*

Prestazioni: calcolo dei ritardi

- Il calcolo esatto del ritardo si effettua basandosi sulla seguente architettura
- Siano T_s e T_r i ritardi per il calcolo della somma e del riporto *i-mi* rispettivamente



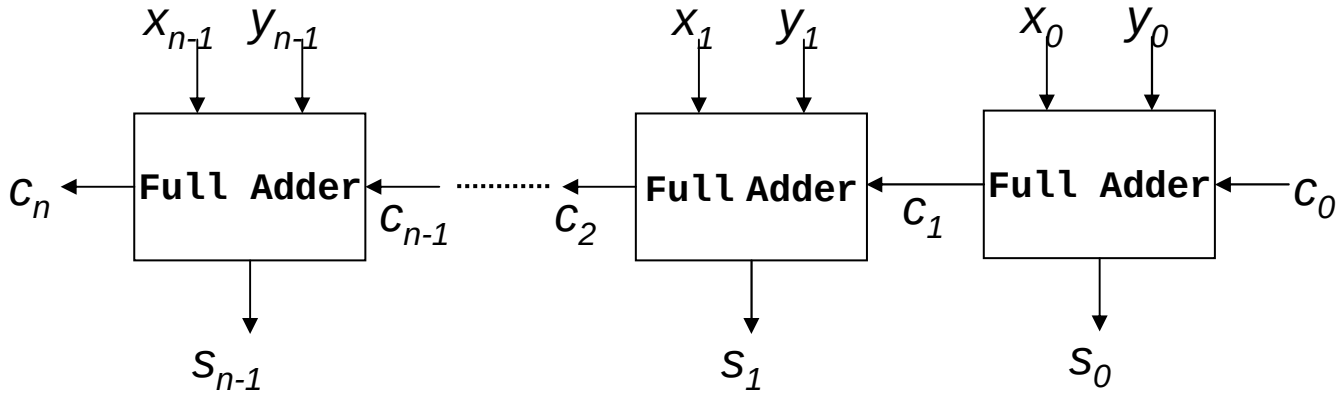
- Prestazioni: il ritardo totale per ottenere tutti i bit della somma è dato dall'espressione:

$$T_{tot} = (n-1)T_r + T_s$$

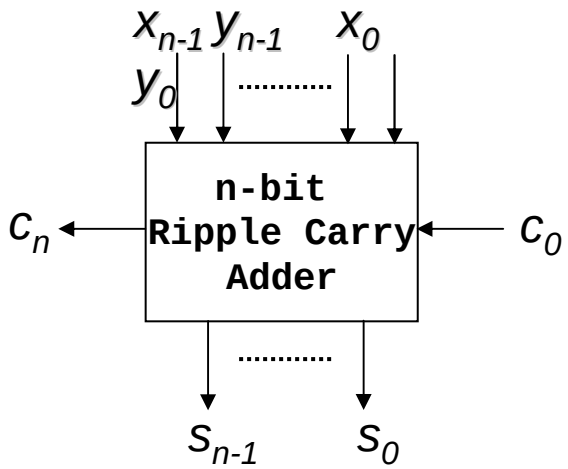
- Il percorso critico è quindi quello del riporto

Sommatori: Ripple Carry[1]

Ripple-Carry Architecture

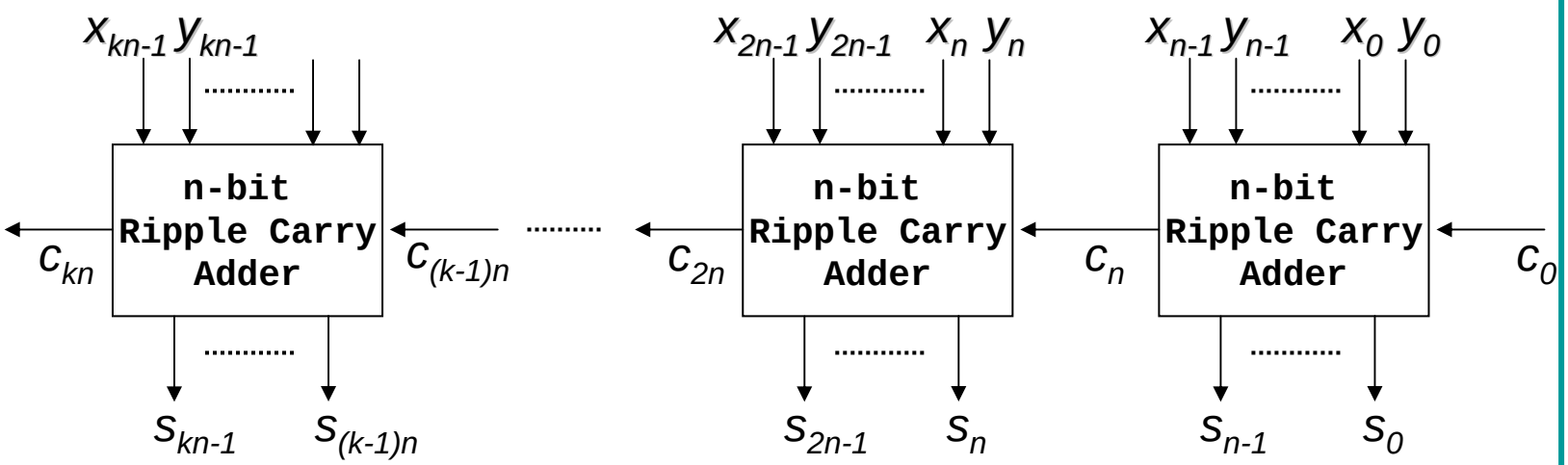


Ripple-Carry Adder



Sommatori: Ripple Carry[2]

Ripple-Carry Block Architecture



Addizione veloce (ad anticipazione di riporto)

Funzioni di generazione e di propagazione del riporto

Si basa sulle seguenti considerazioni

- Le espressioni di somma e riporto per lo stadio *i* sono:

$$s_i = x_i'y_i'c_i + x_i'y_ic_i' + x_iy_i'c_i' + x_iy_ic_i$$

$$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$$

- L'espressione del riporto in uscita può essere riscritta come:

$$c_{i+1} = G_i + P_ic_i \quad \text{con} \quad G_i = x_iy_i \quad \text{e} \quad P_i = x_i + y_i$$

(o anche $P_i = x_i \oplus y_i$)

- Le funzioni G_i e P_i
  - Sono dette funzioni di generazione e propagazione
    - G_i: se $x_i=y_i=1$, allora il riporto in uscita deve essere generato
    - P_i: se x_i o $y_i=1$ e $c_i=1$, allora il riporto in ingresso deve essere propagato in uscita
  - Possono essere calcolate in parallelo, per tutti gli stadi, rispetto alle rispettive somme.

Addizione veloce

calcolo dei riporti in parallelo

- L'espressione per il riporto $c_{i+1} = G_i + P_i c_i$ può essere calcolata in modo iterativo.
- Sostituendo $c_i = G_{i-1} + P_{i-1} c_{i-1}$ nell'espressione di c_{i+1} si ha:

$$c_{i+1} = G_i + P_i(G_{i-1}+P_{i-1}c_{i-1}) = G_i + P_iG_{i-1}+ P_iP_{i-1}c_{i-1}$$

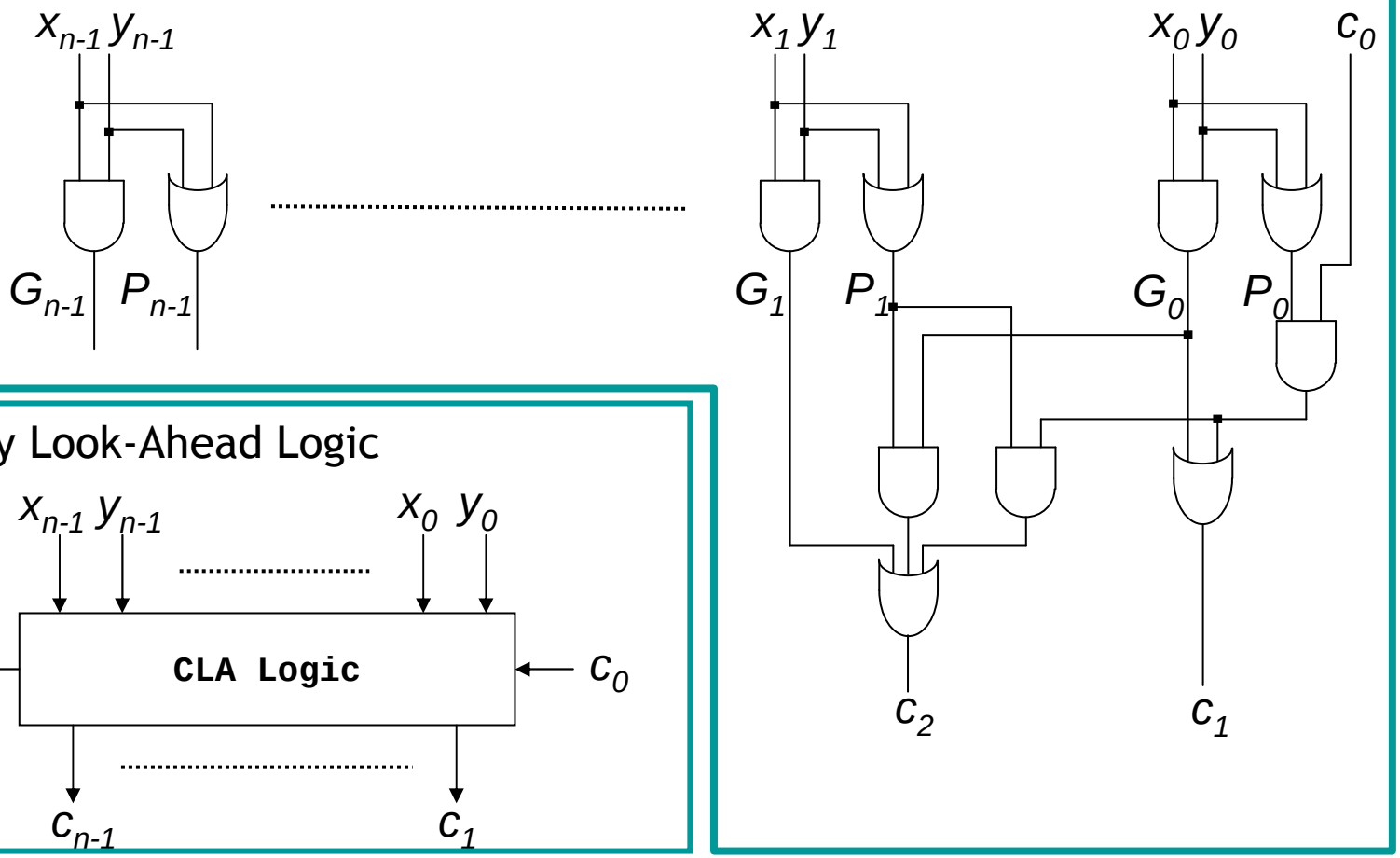
- Continuando con l'espansione fino a c_0 si ottiene:

$$\begin{aligned} c_{i+1} = G_i \quad & + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \\ & + \ldots + \\ & + P_iP_{i-1}\ldots P_1G_0 + \\ & + P_iP_{i-1}\ldots P_1P_0c_0 \end{aligned}$$

- I riporti in uscita di ogni singolo stadio possono essere calcolati tutti in parallelo e con ritardo identico (realizzazione SOP) tramite:
  - le *i* funzioni di generazione G_i e le *i* funzioni di propagazione P_i
  - il riporto in ingresso allo stadio 0, c_0
- I sommatori che sfruttano il meccanismo della generazione dei riporti in anticipo sono detti Carry-Look-Ahead Adders o CLA Adders

Sommatori: Carry Look-Ahead[1]

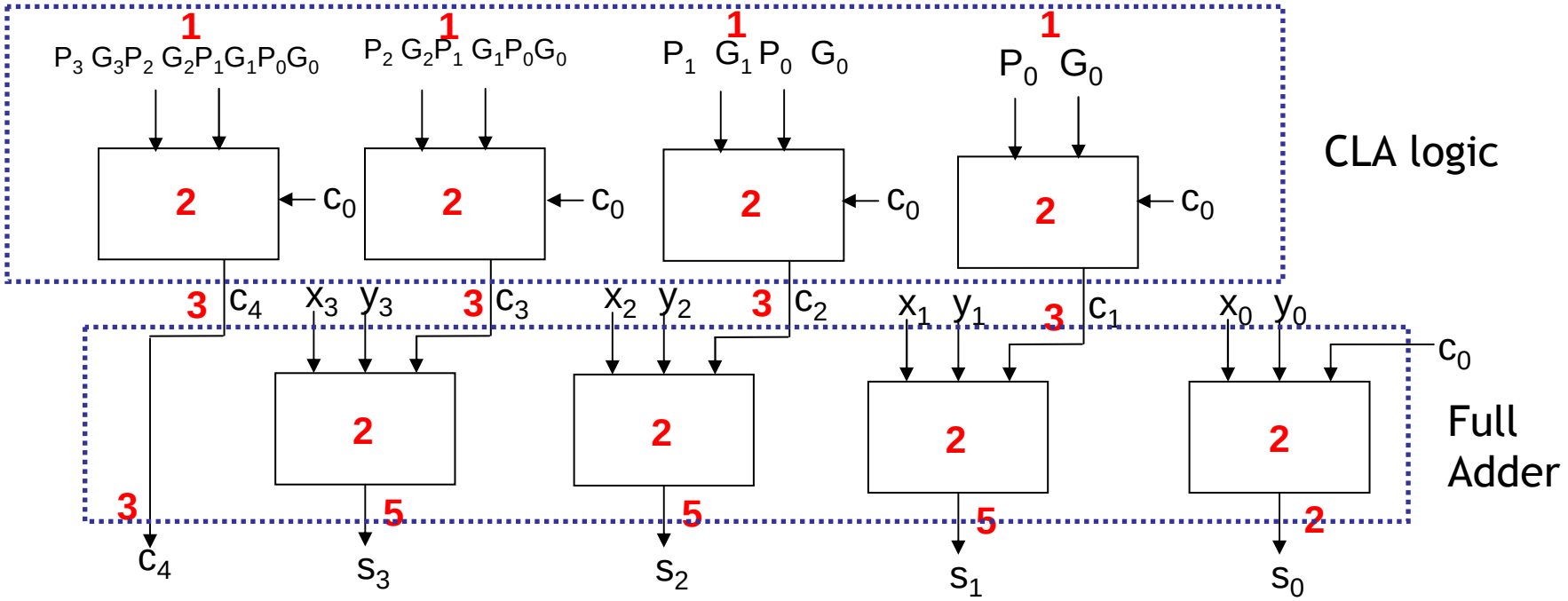
Carry Look-Ahead Logic: Internal architecture



Carry Look-Ahead Logic

Addizione veloce

Esempio: prestazioni per un CLA a 4 bit



- $c_1=G_0+P_0c_0$
- $c_2= G_1+P_1G_0+P_1P_0c_0$
- $c_3= G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0c_0$
- $c_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$

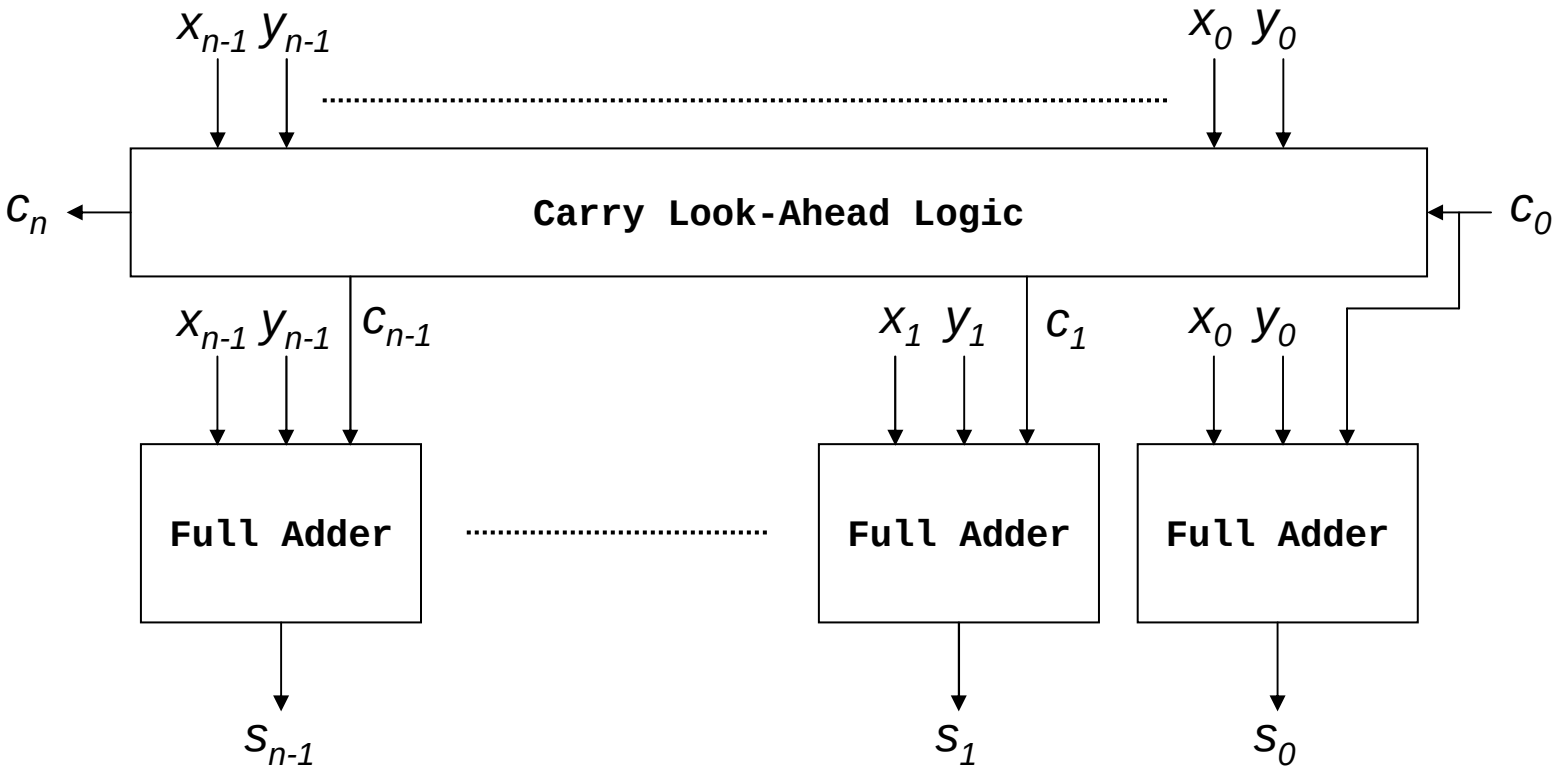
dove

$G_i=x_iy_i$

$P_i=x_i+y_i$

Sommatori Carry Look-Ahead

Carry Look-Ahead Logic



Somma di più valori

- Calcolo della somma di 3 (o più) valori come:

$$W = X + Y + Z$$

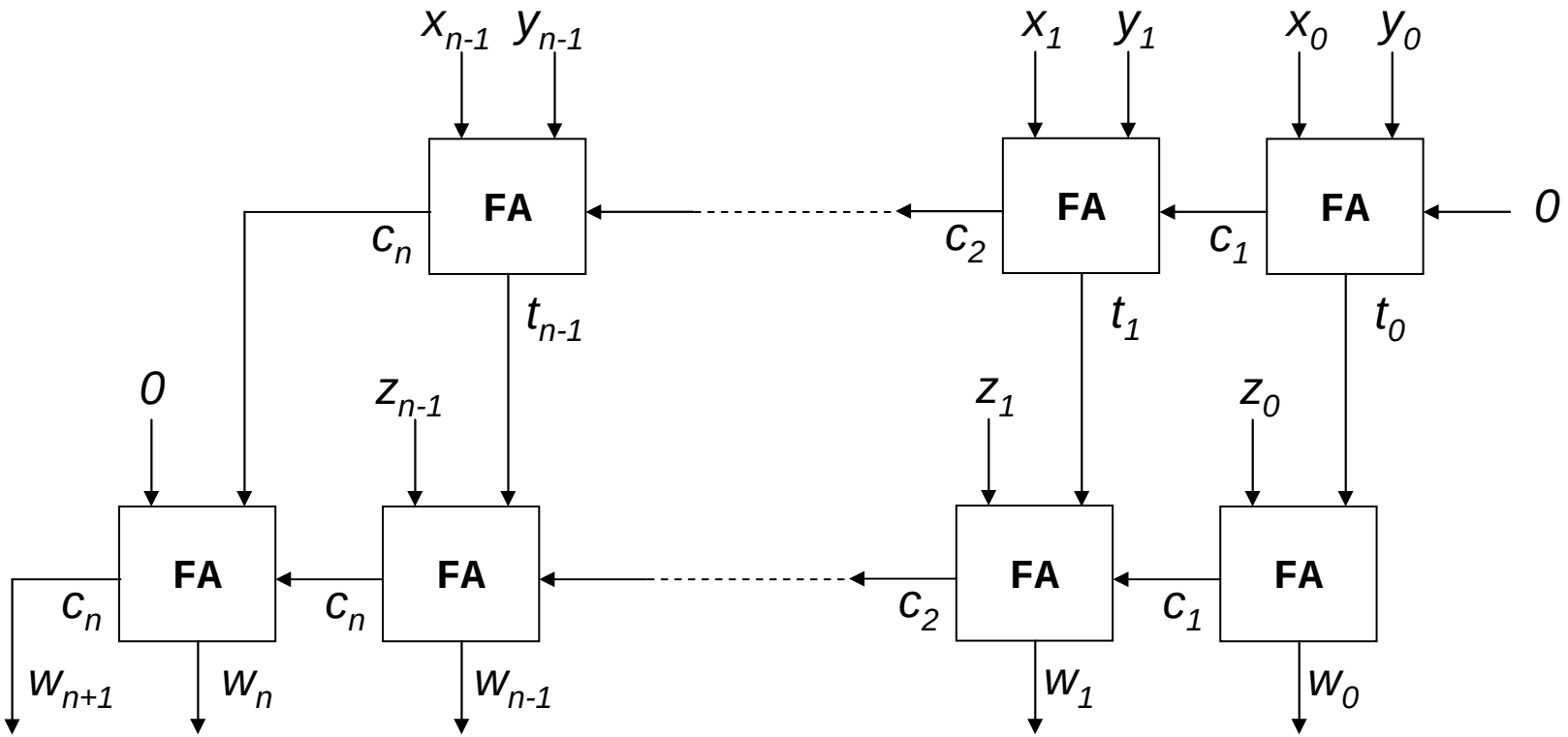
- Soluzione:
  - Calcolare una somma intermedia

    $T = X + Y$
  - E quindi calcolare il risultato finale:

    $W = T + Z$
  - Le somme possono essere realizzate mediante
    - Due sommatori ripple-carry connessi in cascata
    - Due sommatori carry look-ahead connessi in cascata
- Ricorda la somma di N addendi da n bit richiede $n+\lceil log_2 N \rceil$ bit per il risultato

Somma di tre addendi - Architettura con sommatori ripple-carry

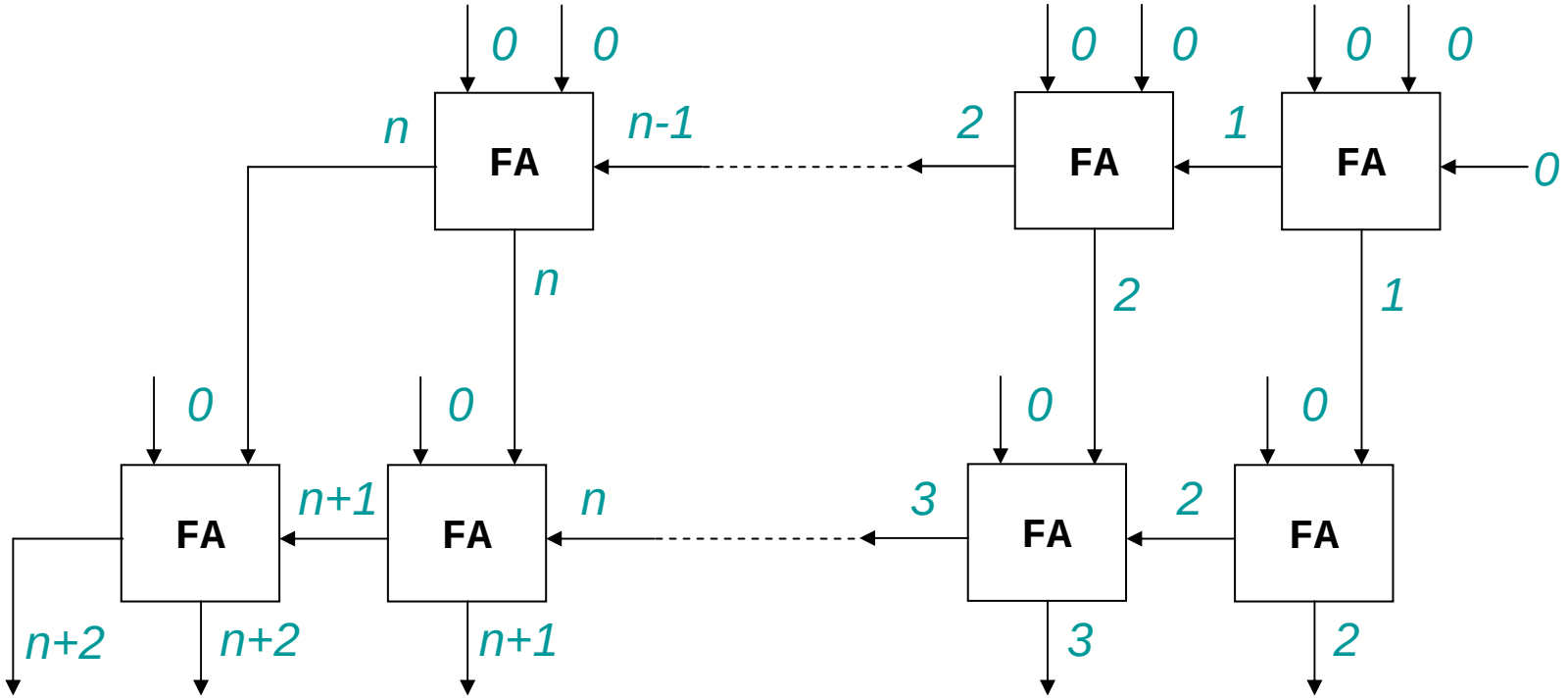
Soluzione con sommatori ripple-carry W = (X + Y) + Z = T + Z



Somma di tre addendi - Prestazioni con sommatori ripple-carry

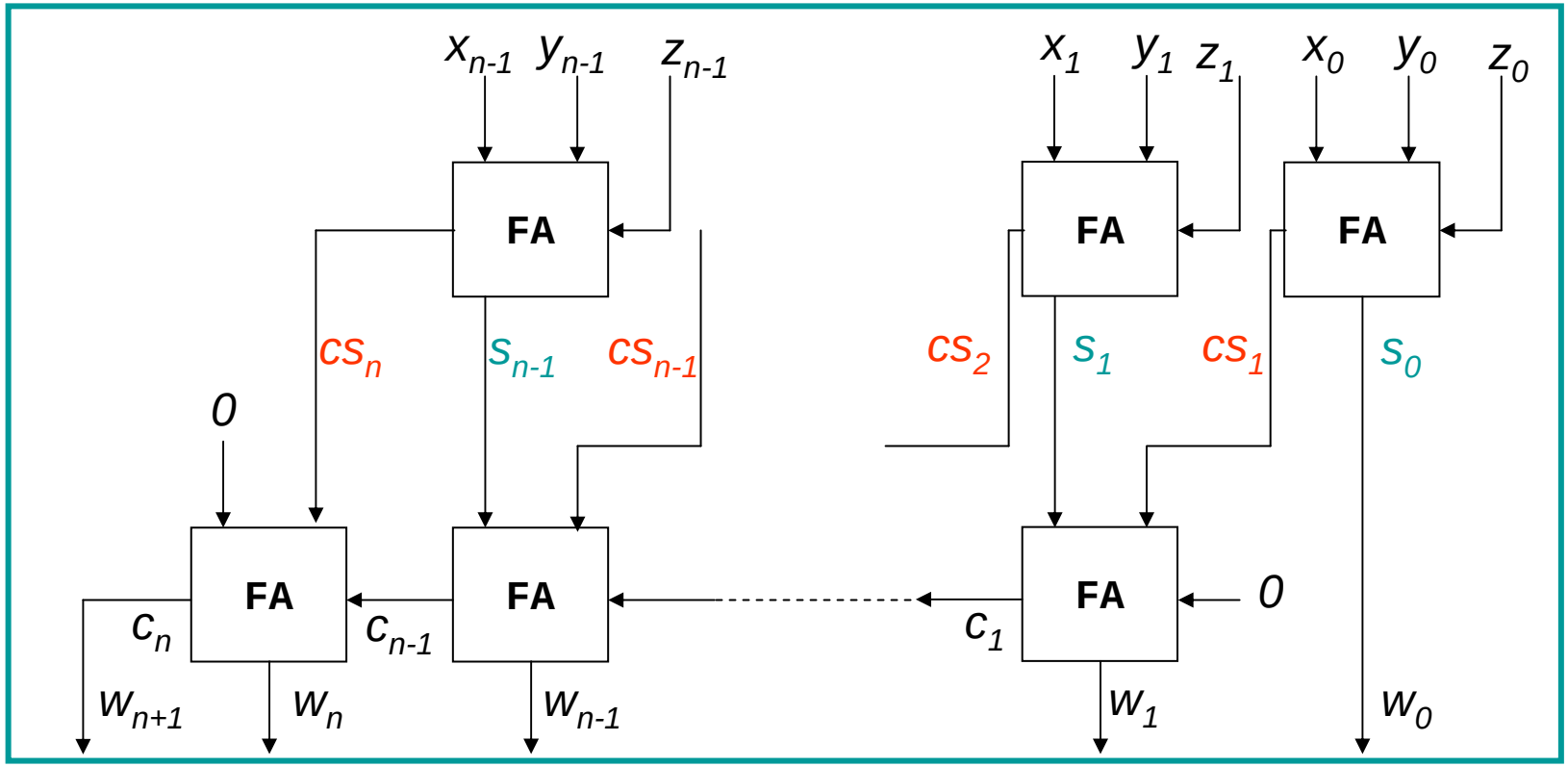
Prestazioni con sommatori ripple-carry (in blu il ritardo di ogni segnale)
Ritardo $R = (n + 2)\Delta T$ con ΔT ritardo di un Full-Adder

Somma di N addendi da n bit: N-1 stadi di somma, risultato su $n+\lceil log_2 N \rceil$ bit, ritardo = $(n+\lceil log_2 N \rceil)\ \Delta T$



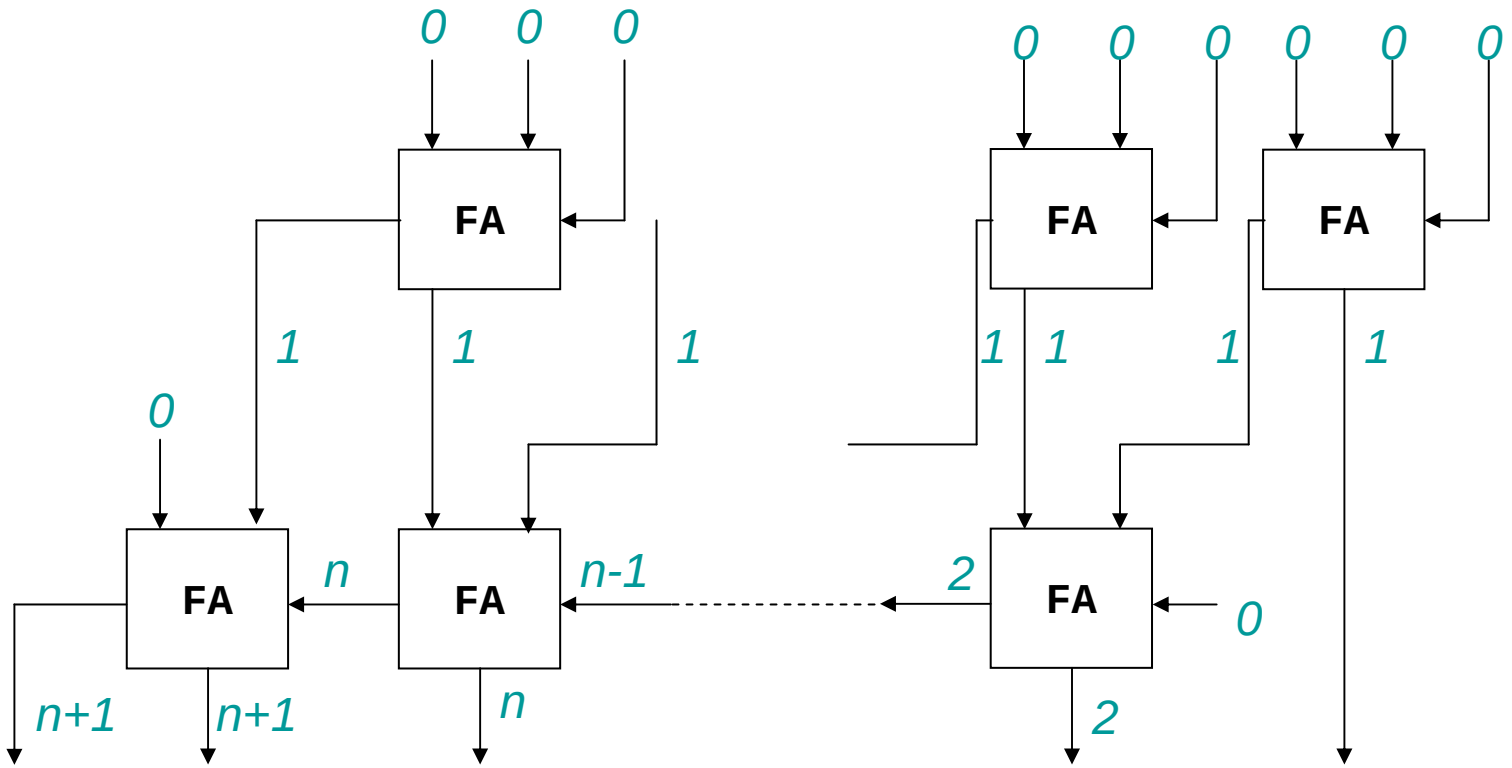
Somma di tre addendi con Sommatori Carry Save

- Il primo stadio calcola le somme S (parziali e senza propagazione di riporto) e i riporti CS (Carry Save Adder)
- Il secondo stadio somma (con propagazione di riporto) i valori provenienti dal primo stadio



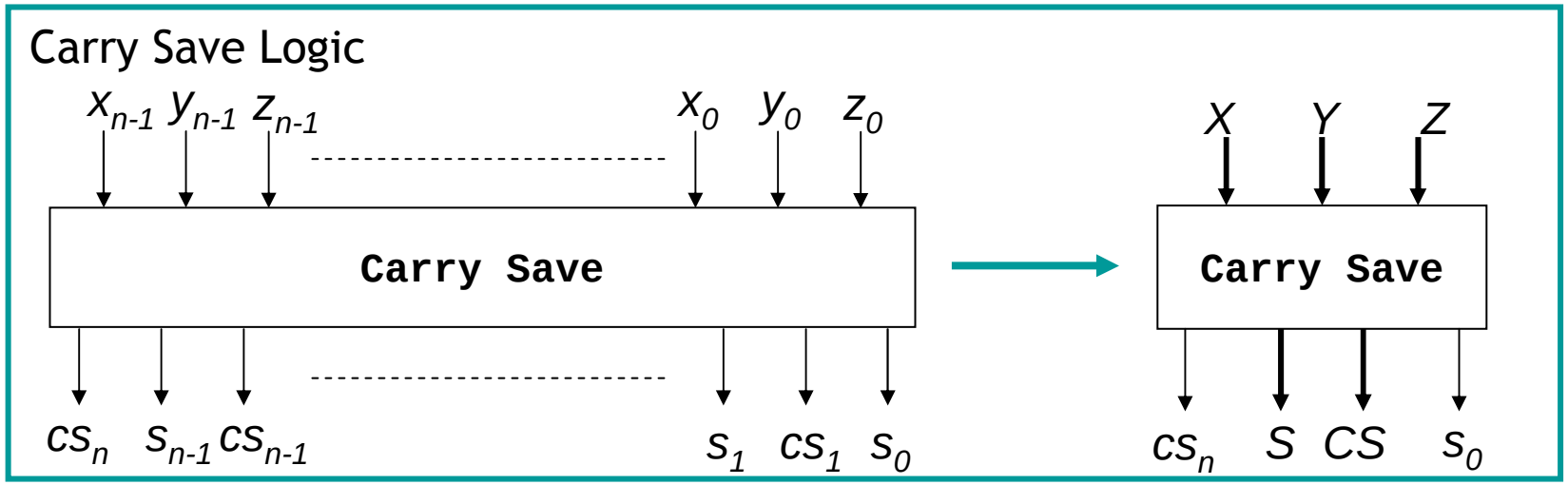
Somma di tre addendi - Prestazioni Carry Save

Prestazioni con sommatore ripple-carry per l'ultimo stadio (in blu il ritardo di ogni segnale)
Ritardo $R = (n + 1)\Delta T$ con ΔT ritardo di un Full-Adder



Sommatore Carry Save come blocco

- Sommatore Carry Save composto da due unità
  - Blocco Carry Save:
    - Produce i due vettori S e CS
    - Ritardo: $R_{CS} = 1$
  - Sommatore Ripple-Carry:
    - Produce il risultato finale
    - Ritardo: $R_{RC} = n + 1$



- Il ritardo totale per ottenere tutte le somme ed il riporto più a sinistra c_{i+1} è dato dalla somma di:
  - Un ritardo di porta per il calcolo delle funzioni di generazione e di propagazione ($G_i = x_i y_i$ e $P_i = x_i + y_i$)
  - Due ritardi di porta logica per calcolare il riporto *i*-esimo (SOP)
  - Due ritardi di porta logica per calcolare la somma *i*-esima (SOP)
- Totale:
  - 5 ritardi di porta logica
- Il ritardo è costante e indipendente dalla lunghezza degli operandi
- Problema:
  - La realizzazione circuitale dei moduli che calcolano i riporti per operandi lunghi (ad esempio 32 bit) fa uso di porte con un *fan-in* molto elevato: non praticabile!!
  - Soluzione: addizionatore veloce a blocchi

Addizione veloce a blocchi

- Il sommatore completo a *n* bit è ottenuto utilizzando un insieme di blocchi costituiti da CLA a *m* bit e della logica CLA
- Esempio: blocco è costituito da un sommatore CLA a 4 bit (ragionevole)
- Struttura del blocco di un CLA a 4 bit
  - Il riporto finale di questo sommatore ha la seguente espressione:

$$c_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0\,c_0$$

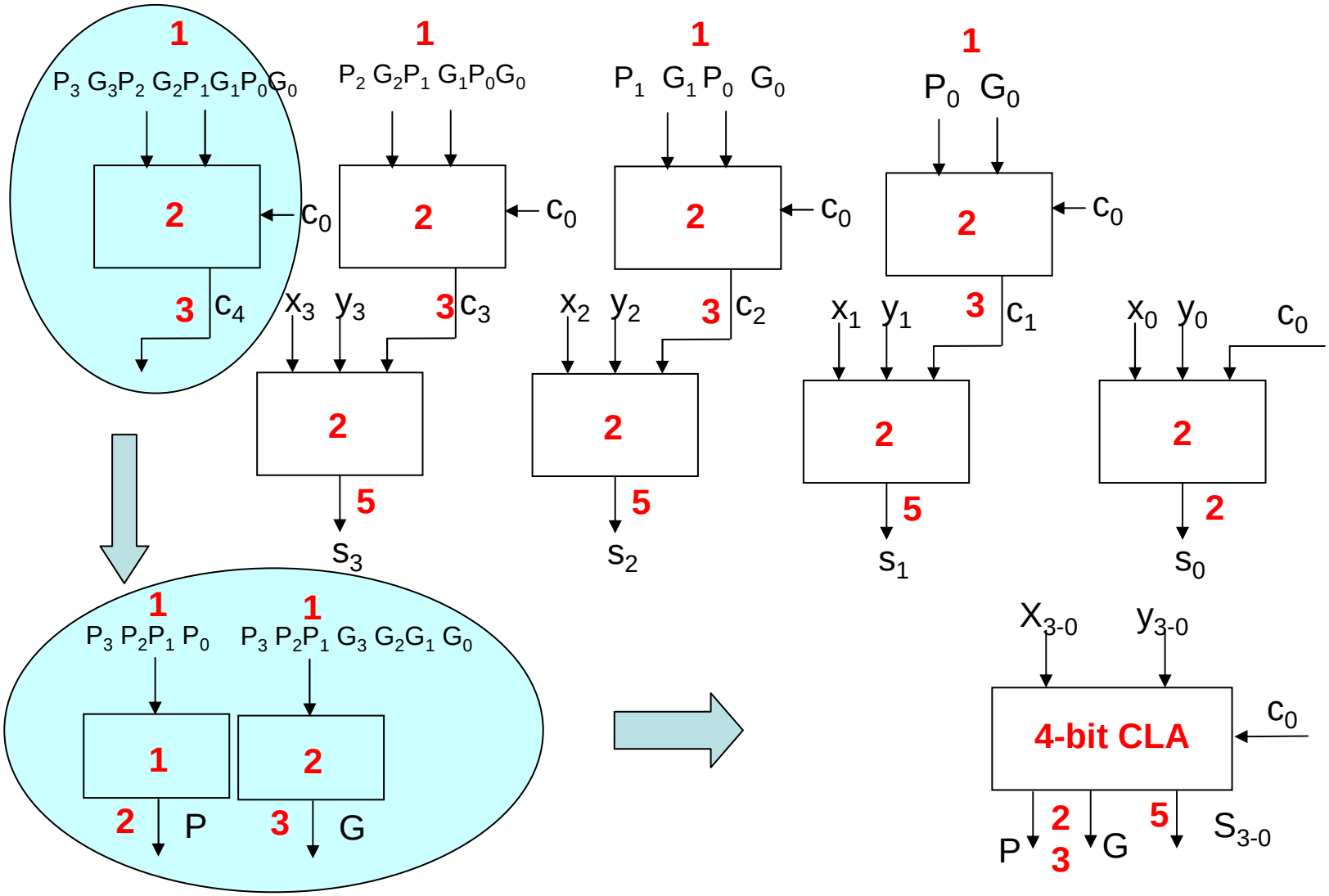
  - che può essere riscritta come

$$c_{uscita} = G + Pc_0$$

  - con il tempo di ritardo per il calcolo di P e G:
    - P = attraversamento di 2 porte logiche (1 per calcolare P_3, P_2, P_1 *e* P_0, 1 per calcolare il prodotto)
    - G = attraversamento di 3 porte logiche (calcolo di P_i e G_i, calcolo dei prodotti, calcolo della somma)

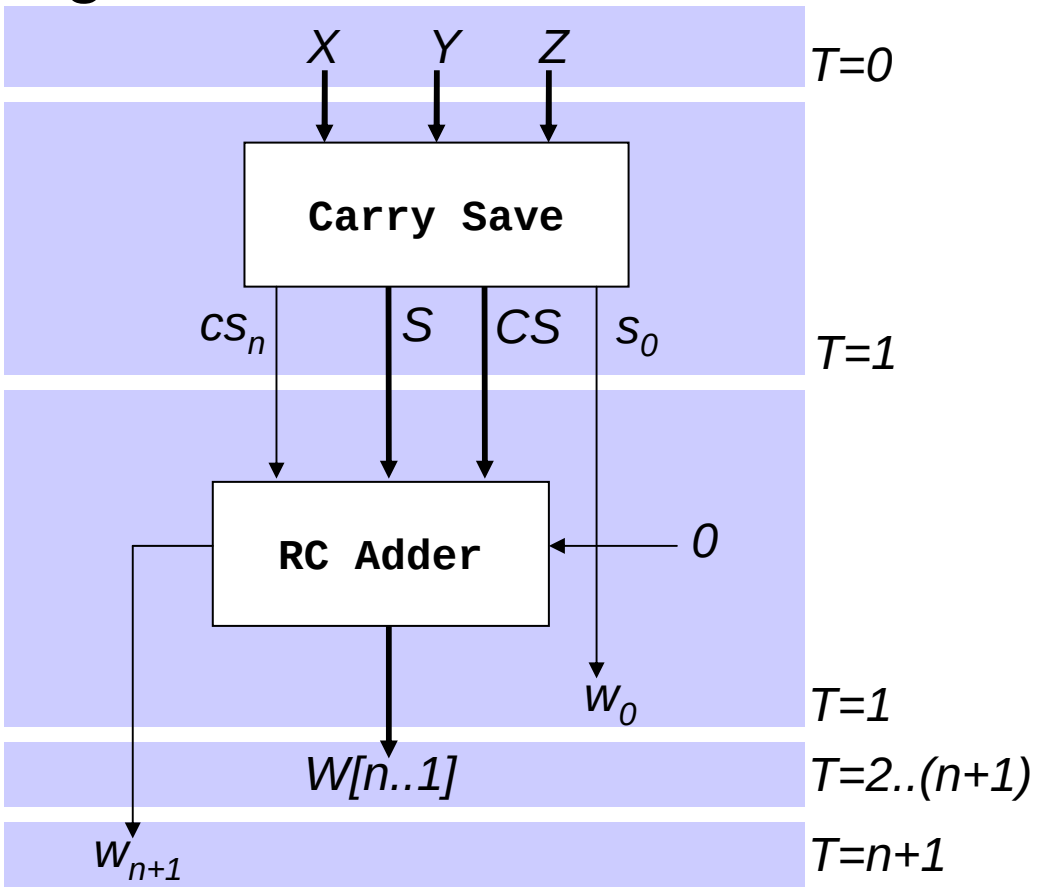
Addizione veloce

blocco CLA a 4 bit

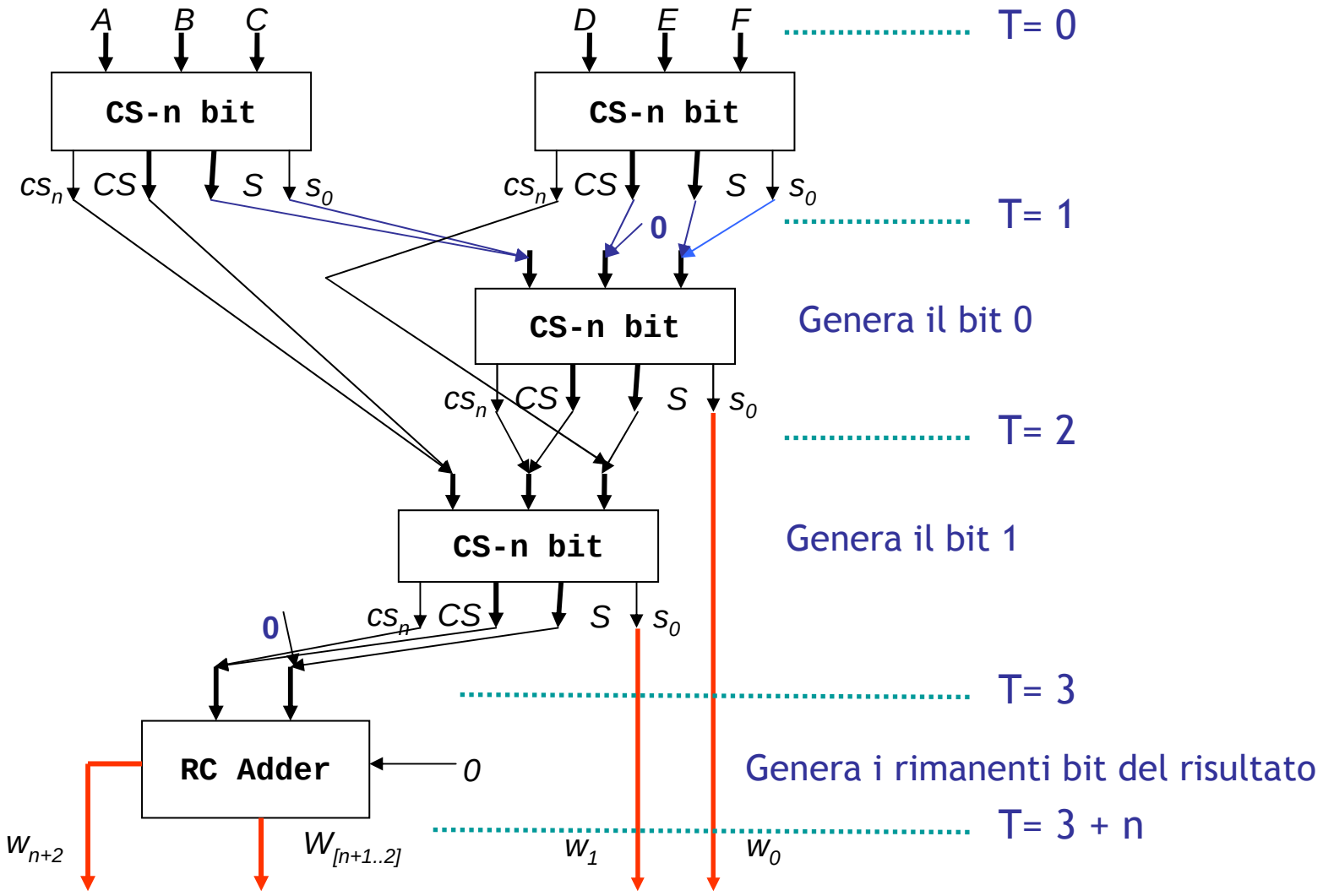


Sommatore Carry Save come blocco

- Istanti di generazione dei bit di uscita

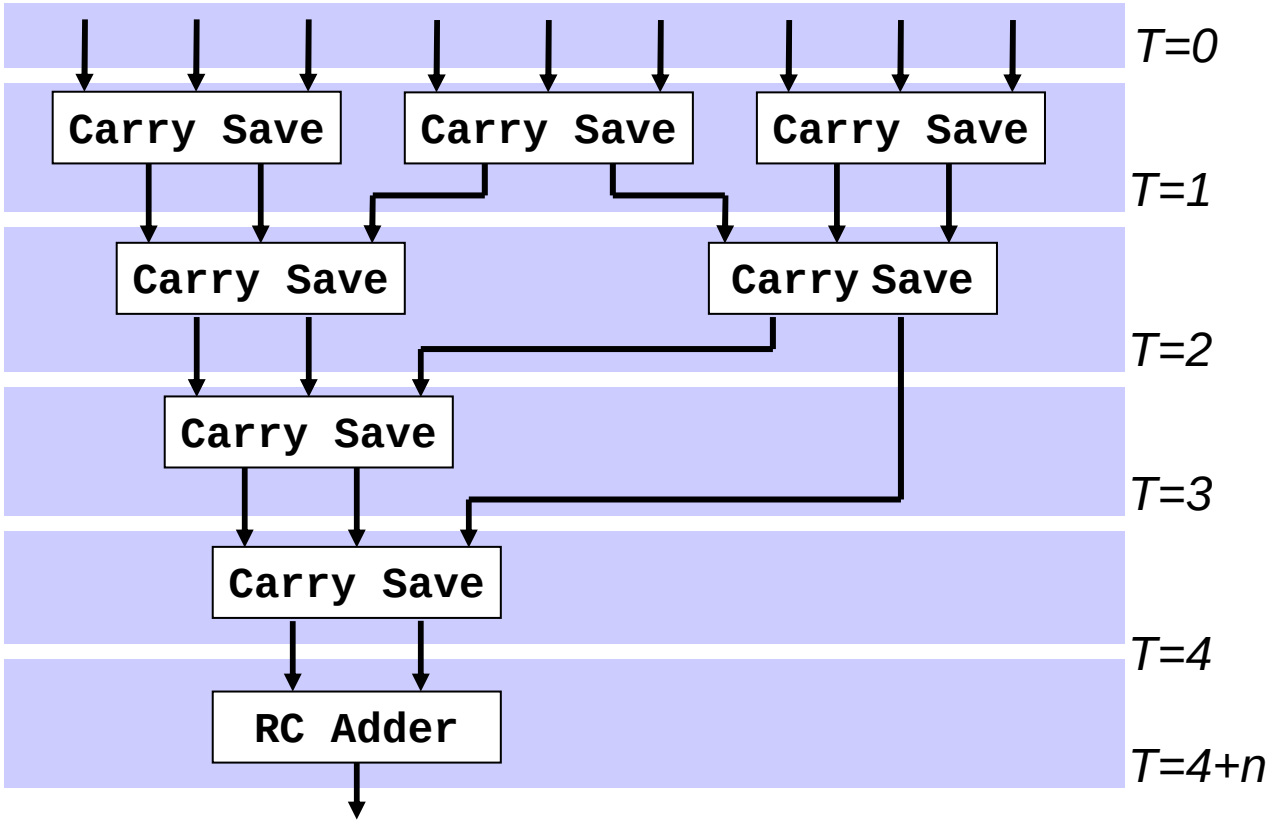


Esempio sommatore a 6 addendi con blocchi Carry Save da 3 addendi



Esempio sommatore a 9 addendi con blocchi Carry Save da 3 addendi

- Vantaggi più evidenti al crescere del numero degli operandi



Addizione e sottrazione per valori rappresentati in complemento a 2

- Condizioni di overflow e di underflow per somme e sottrazioni in complemento a 2 su *n* bit

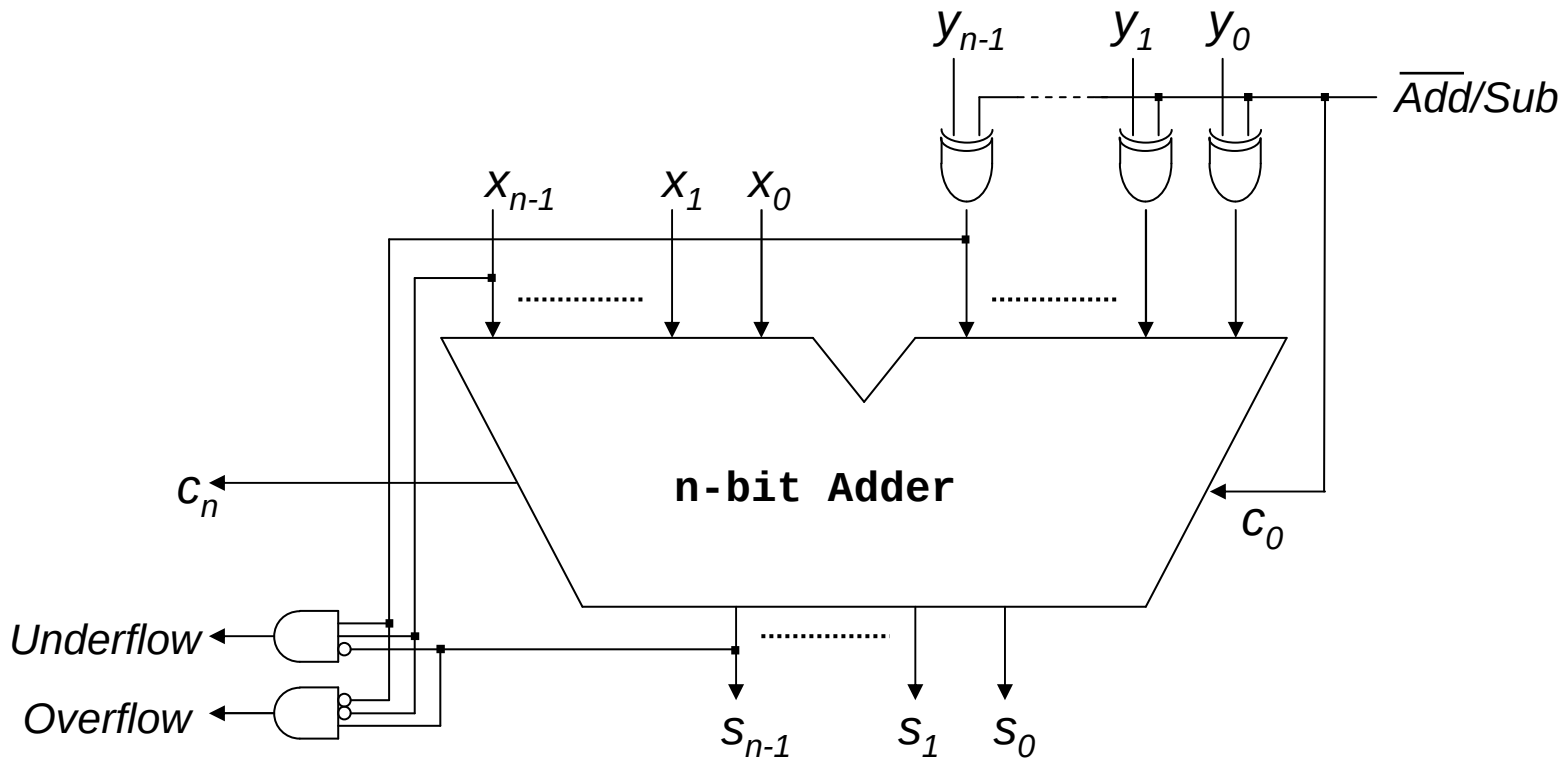
A+B			
A	B	Segno somma	Ov/Un
> 0	> 0	0	Si-Ov
> 0	< 0		no
< 0	> 0		no
< 0	< 0	1	Si-Un

A-B=A+(-B)				
A	B	-B = B_{CPL2}	Segno somma	Ov/Un
> 0	> 0	< 0		no
> 0	< 0	>0	0	Si-Ov
< 0	> 0	< 0	1	Si-Un
< 0	< 0	> 0		no

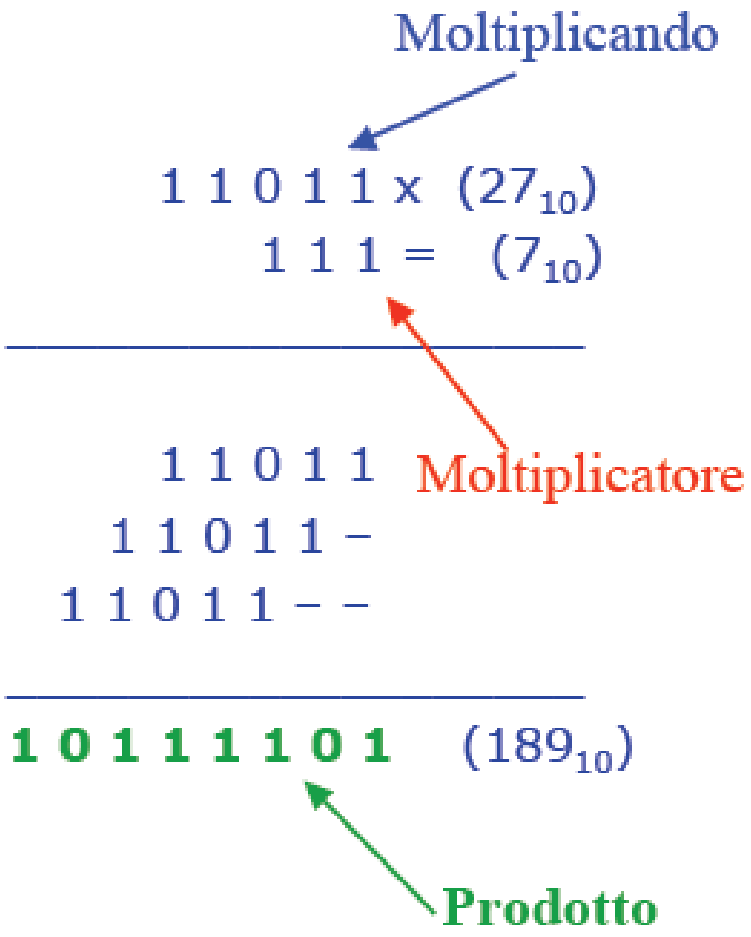
- overflow per somma = 0 0 1 (segno addendi e segno somma)
- underflow per somma = 1 1 0
- overflow per sottrazione = 0 1 1
- underflow per sottrazione = 1 0 0

Sommatori Add/Subtract operazioni in complemento a 2

Add/Subtract Architecture



Moltiplicazione



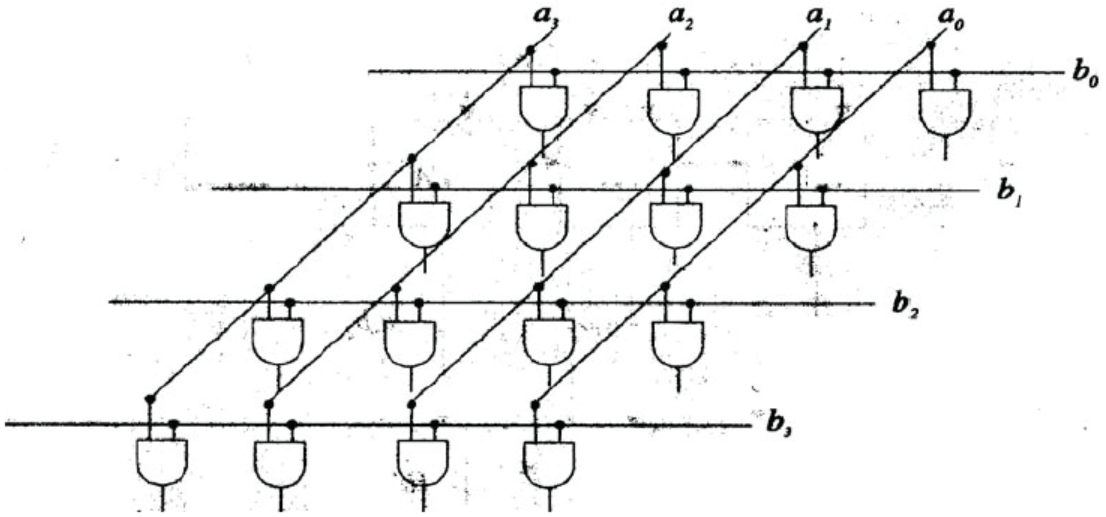
- Prima Fase: prodotti parziali
  - AND logico dei bit del moltiplicando e del moltiplicatore
- Seconda Fase: somma
  - Somma dei bit dei prodotti parziali per ottenere il risultato finale

Prima Fase: prodotti parziali

- 4 x 4 bit

		a_3	a_2	a_1	a_0	
		$a_3 b_0$	$a_2 b_0$	$a_1 b_0$	$a_0 b_0$	b_0
	$a_3 b_1$	$a_2 b_1$	$a_1 b_1$	$a_0 b_1$		b_1
$a_3 b_2$	$a_2 b_2$	$a_1 b_2$	$a_0 b_2$			b_2
$a_3 b_3$	$a_2 b_3$	$a_1 b_3$	$a_0 b_3$			b_3

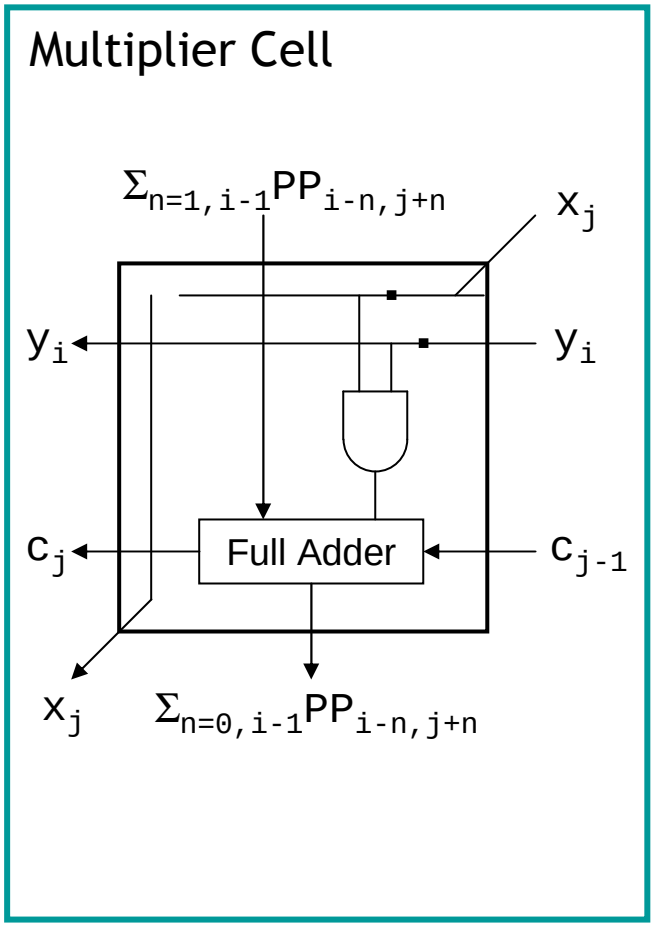
- Circuito della prima fase



Seconda Fase: somma

- Vi sono tre modi per eseguire la seconda fase:
  - Somma per righe
  - Somma per diagonale
  - Somma per colonne

- Somma per righe



Ogni cella del moltiplicatore calcola

- **il prodotto parziale corrispondente e**
- **una somma parziale**

Il riporto delle somme parziali si propaga lungo la riga

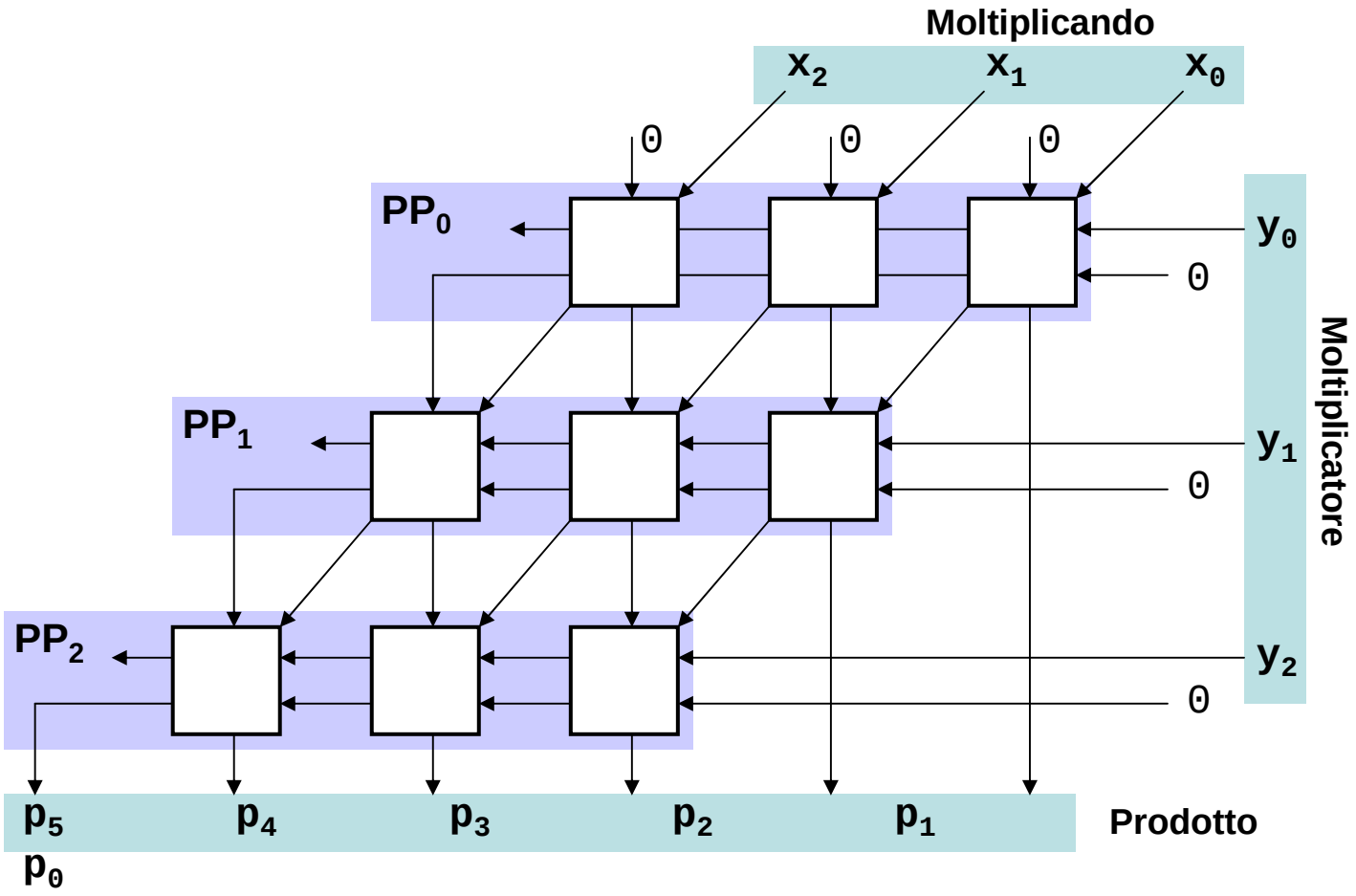
Le somme si propagano in verticale

Per il calcolo del prodotto parziale, X si propaga in diagonale e Y in verticale

Sono necessari *n-1* sommatori a *n* bit (con eventuale calcolo del prodotto parziale). Il primo non genera riporti

La struttura è regolare

Prestazioni: dipendono dai sommatori, con sommatori non veloci ordine di *2n*

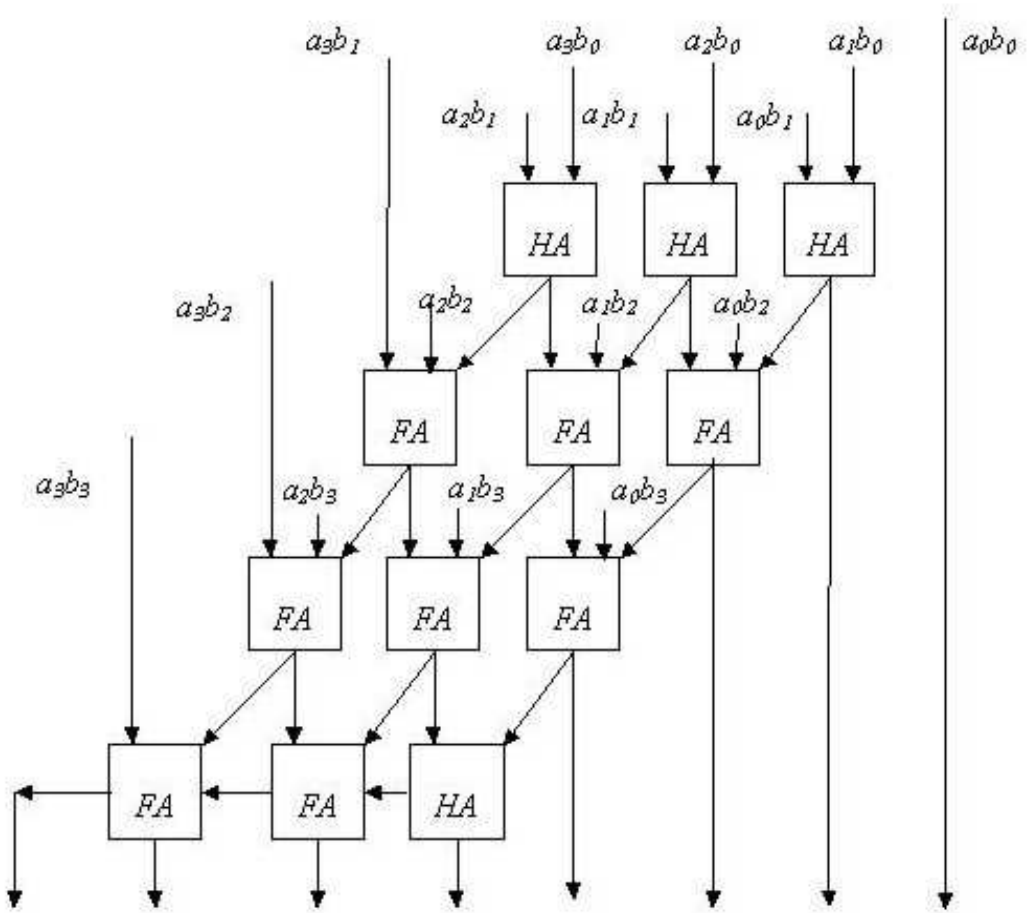

Moltiplicando
x2
x1
x0
0
0
0
PP0
y0
0
PP1
y1
0
PP2
y2
0
Moltiplicatore
p5
p4
p3
p2
p1
p0
Prodotto

- Somma per diagonali
  - Ogni cella del moltiplicatore (tranne quelle dell'ultima riga) calcola il prodotto parziale corrispondente e una somma parziale
  - Il riporto delle somme parziali si propaga lungo le diagonali
  - Le somme si propagano in verticale
  - Per il calcolo del prodotto parziale, X si propaga in diagonale e Y in verticale
  - Sono necessari *n* sommatori a *n* bit (di cui il primo non genera riporti)
  - La struttura è regolare
  - Prestazioni: dipendono dai sommatori, con sommatori non veloci ordine di *2n*

Moltiplicatori combinatori

somma per diagonali

Circuito per la somma per diagonali.



Moltiplicatori combinatori

somma per colonne

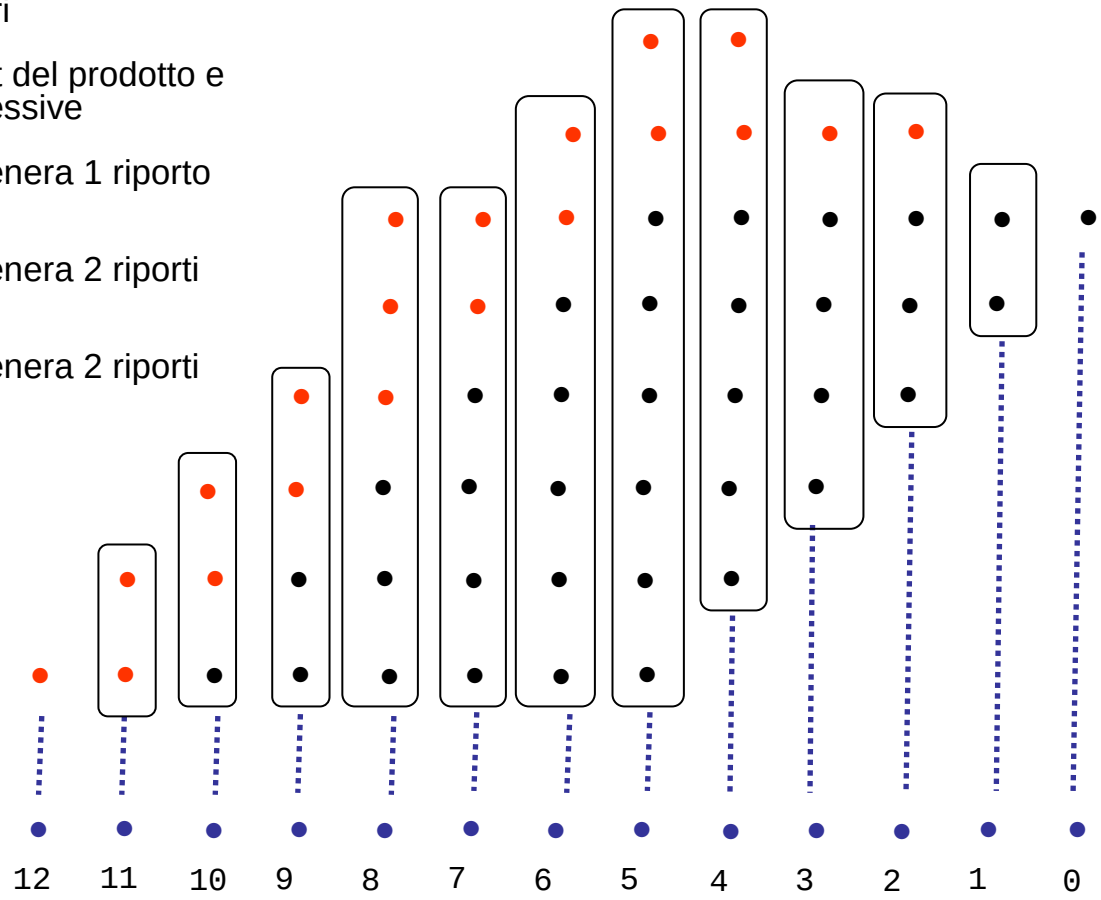
- Il metodo è simile a quello utilizzato a mano per effettuare la moltiplicazione

- Si utilizza la matrice dei prodotti parziali (matrice di AND) e un insieme di contatori paralleli
- Il generico contatore parallelo riceve in ingresso una colonna di prodotti parziali (e gli eventuali riporti dagli stadi precedenti) e genera il conteggio degli 1 della colonna
- Il conteggio generato in ogni stadio produce il bit del prodotto per lo stadio considerato e eventuali riporti per gli stadi successivi
- Irregolare (contatori diversi)
- Prestazioni: paragonabili a quelle per somma per righe, infatti si ha propagazione di riporti in tutte le colonne

Moltiplicatori combinatori

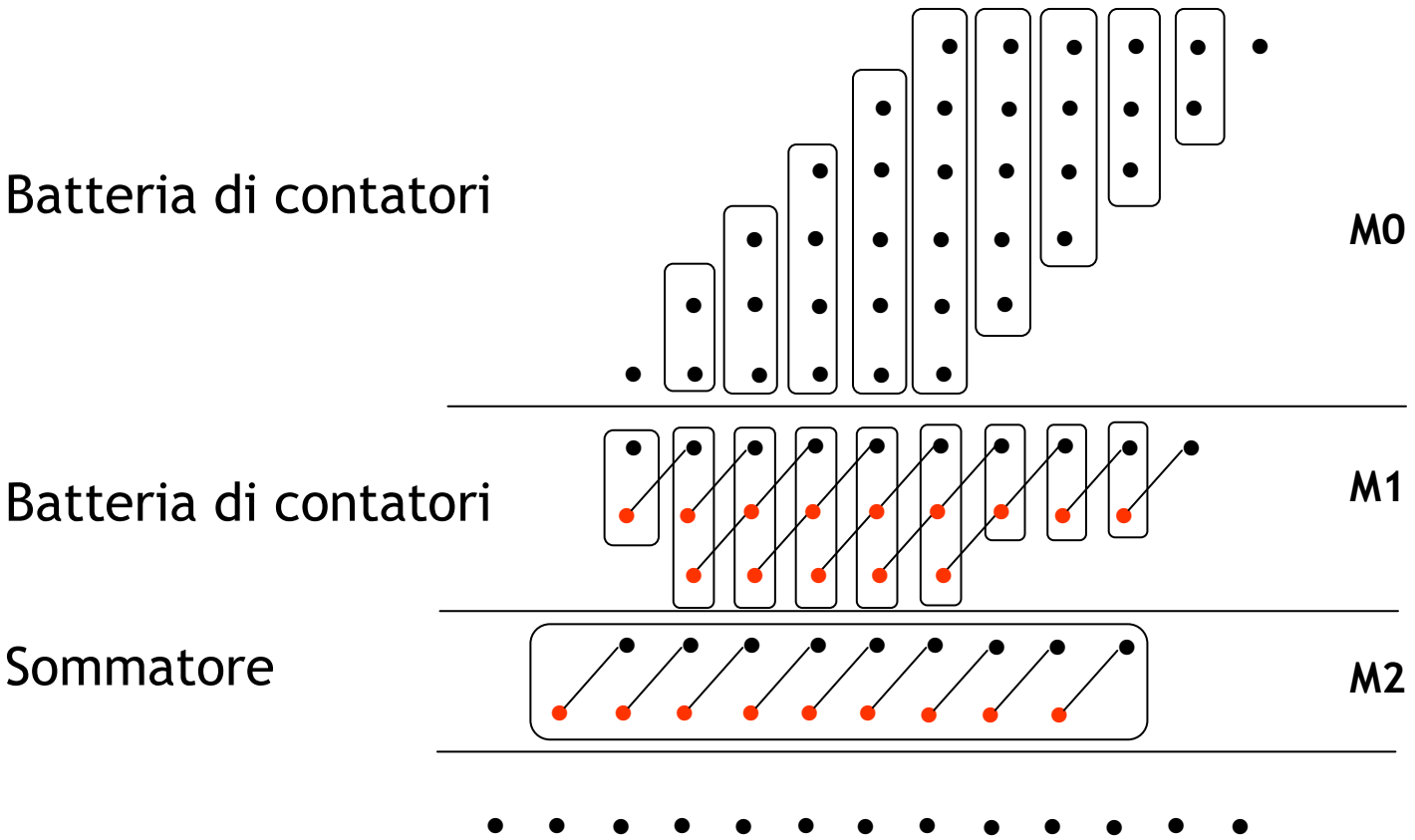
somma per colonne

- Moltiplicando e moltiplicatore da 6 bit
- In nero la matrice di AND, in rosso i riporti generati dai contatori
- ogni contatore genera 1 bit del prodotto e riporti per le colonne successive
- il contatore di colonna 1 genera 1 riporto per colonna 2
- il contatore di colonna 2 genera 2 riporti per colonna 3 e 4
- il contatore di colonna 3 genera 2 riporti per colonna 4 e 5
- e così via…..



Moltiplicatori combinatori: somma per colonne con riduzione della matrice dei termini prodotto

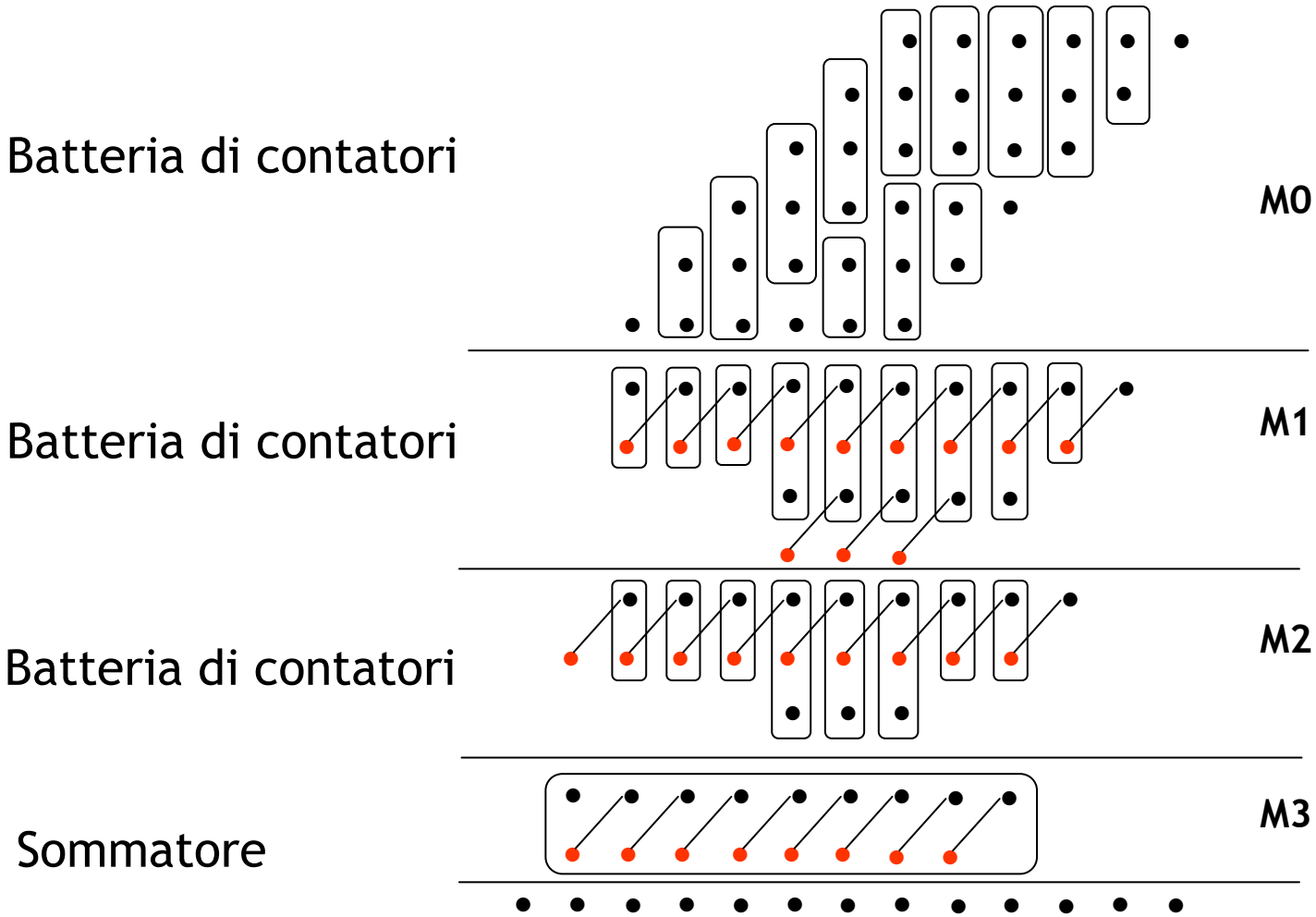
- Riduzione successiva della matrice dei prodotti parziali
  - La matrice dei prodotti parziali M0 viene ridotta, in termini di righe, tramite contatori paralleli per colonna che non propagano i riporti, ma li usano (insieme ai bit di somma) per costruire la matrice ridotta
  - Il risultato generato dai contatori crea una matrice successiva M1, costituita da un numero inferiore di righe. In questo modo non c'è propagazione dei riporti all'interno della stessa matrice
  - Il procedimento viene iterato fino a quando non si ottiene una matrice di sole due righe
  - Le due righe costituiscono l'ingresso ad un sommatore
- La riduzione è rapida
- La struttura è irregolare
- Le prestazioni aumentano
  - ipotesi: il tempo di un contatore è identico a quello di un Full-Adder
  - domina il tempo del sommatore finale


Batteria di contatori
M0
Batteria di contatori
M1
Sommatore
M2

Moltiplicatori combinatori

moltiplicatore di Wallace

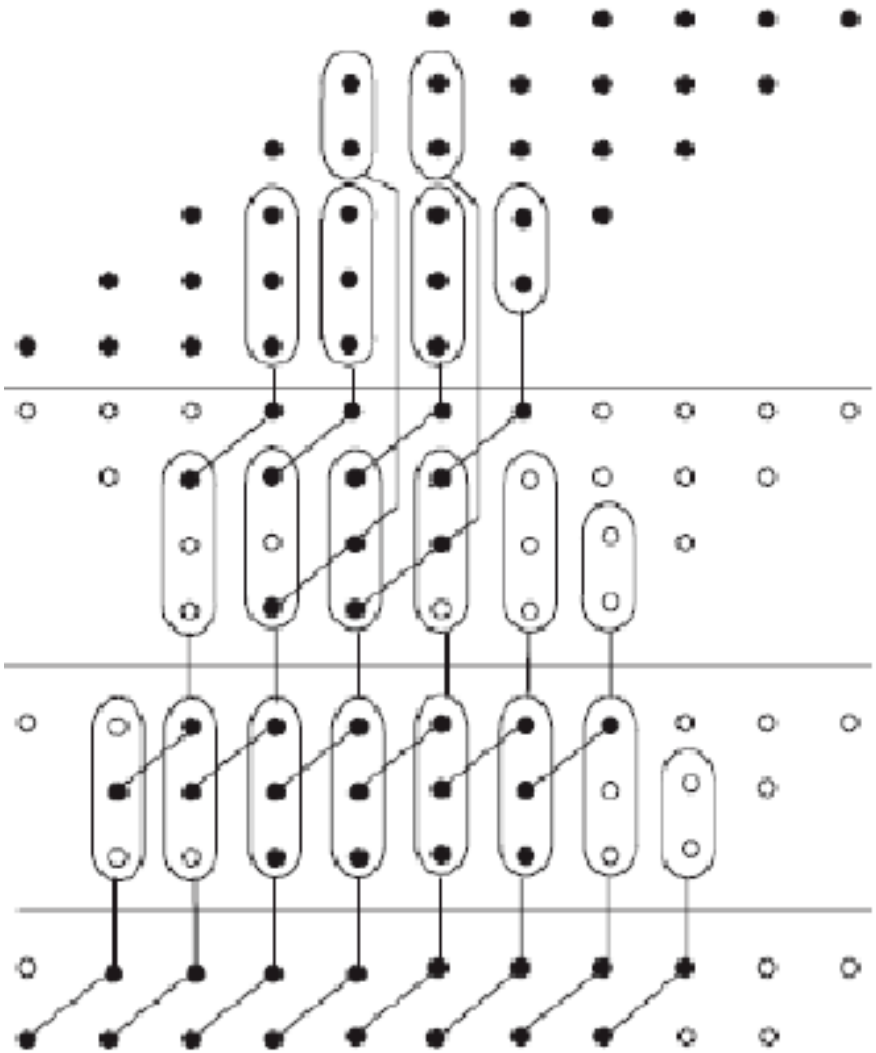
- E’ basato sulla riduzione successiva della matrice M0
- Prevede l’utilizzo di soli contatori a 2 o 3 ingressi, che sono equivalenti rispettivamente ad un Half-Adder e a un Full-Adder
- Il procedimento di riduzione della matrice a 2 sole righe è più lento rispetto al caso di contatori a ingressi qualsiasi, ma comunque rapido ($log_{3/2}\ n$ passi)
  - M0 di n righe
  - M1 di (2/3)n righe
  - M2 di $(2/3)^2$n righe
  - …..
  - Mh di $(2/3)^h$n righe: se il n° di righe è uguale a 2 la riduzione termina
- La struttura è “regolare”
- Le prestazioni sono dominate dal sommatore finale (veloce)


Batteria di contatori
M0
Batteria di contatori
M1
Batteria di contatori
M2
Sommatore
M3

- Le tre fasi caratterizzanti:
  - Moltiplicazione
  - Riduzione
  - Raggruppamento

- Wallace cerca di massimizzare il *conto, riducendo* il più possibile
- Dadda cerca di *contare* il meno possibile

- Il moltiplicatore Dadda costa meno



Moltiplicatori sequenziali [1]

- Moltiplicazione sequenziale tra due numeri di *n*
- I passi da eseguire sono:
  1. Inizializza a zero un registro accumulatore A
  2. Inizializza a zero un bistabile C per il riporto
  3. Salva nei registri Q ed M moltiplicatore e moltiplicando
  4. Se il bit meno significativo di Q vale 1
     - Somma A ed M
     - Memorizza il risultato in A
  5. Shift a destra del registro [C; A; Q] di una posizione
  6. Ripeti dal punto 4 per *n* volte
  7. Preleva il risultato della moltiplicazione dai registro [A; Q]

Moltiplicatori sequenziali [2]

Architettura di un moltiplicatore sequenziale