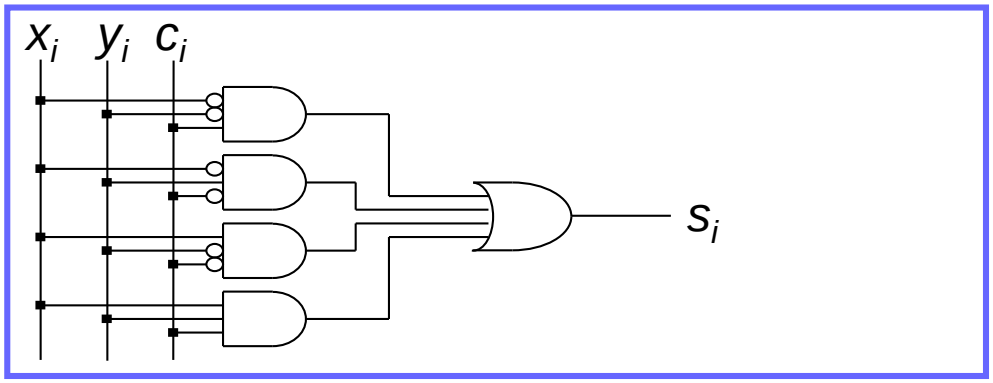
Architetture aritmetiche

Sommatori: Full Adder, Ripple Carry
Sommatori: Carry Look-Ahead, Carry Save, Add/Subtract
Moltiplicatori: Combinatori, Wallace, Sequenziali
Circuiti per aritmetica in virgola mobile

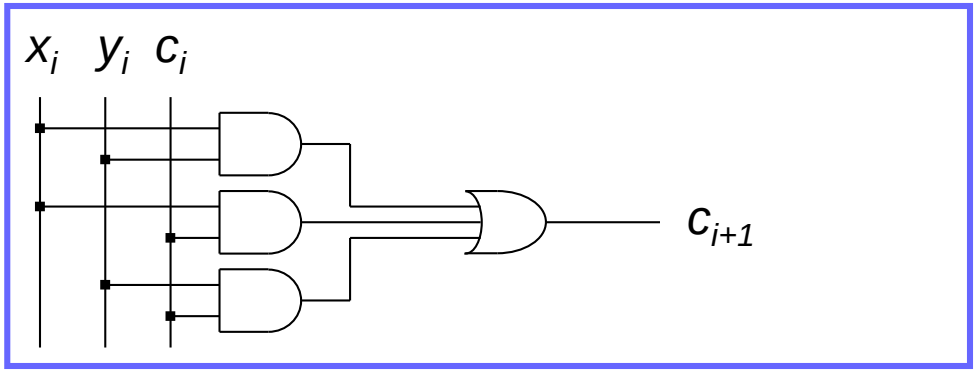
versione del 11/11/03

Sommatori: Full Adder

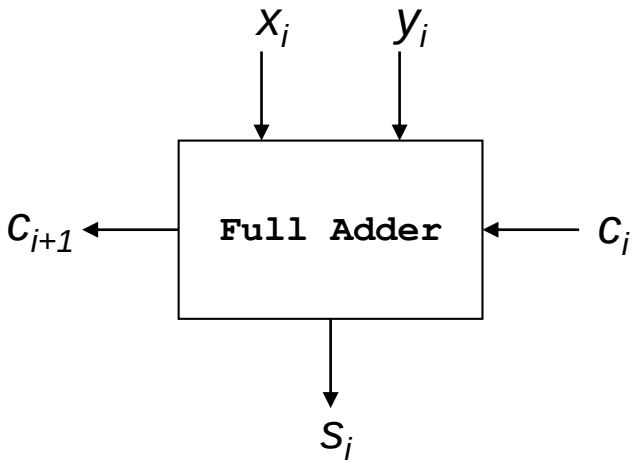
$$s_i = \bar{x}_i \bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i$$



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

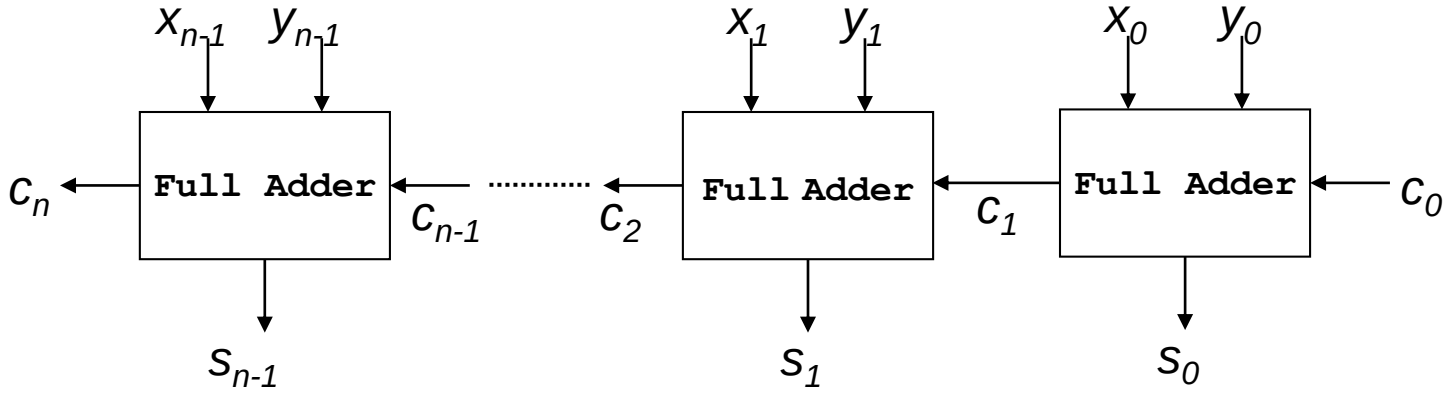


Full Adder

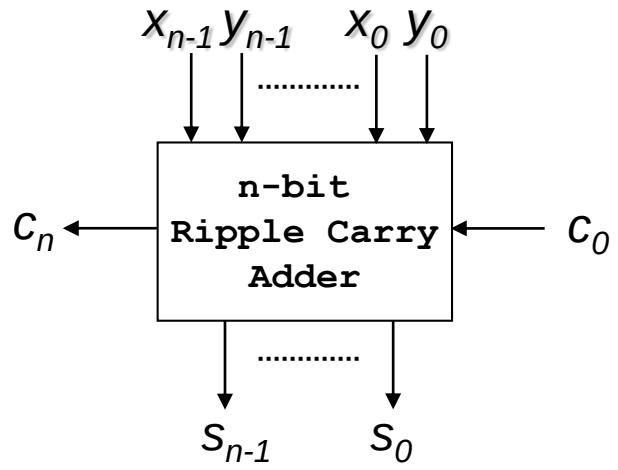


Sommatori: Ripple Carry [1]

Ripple-Carry Architecture

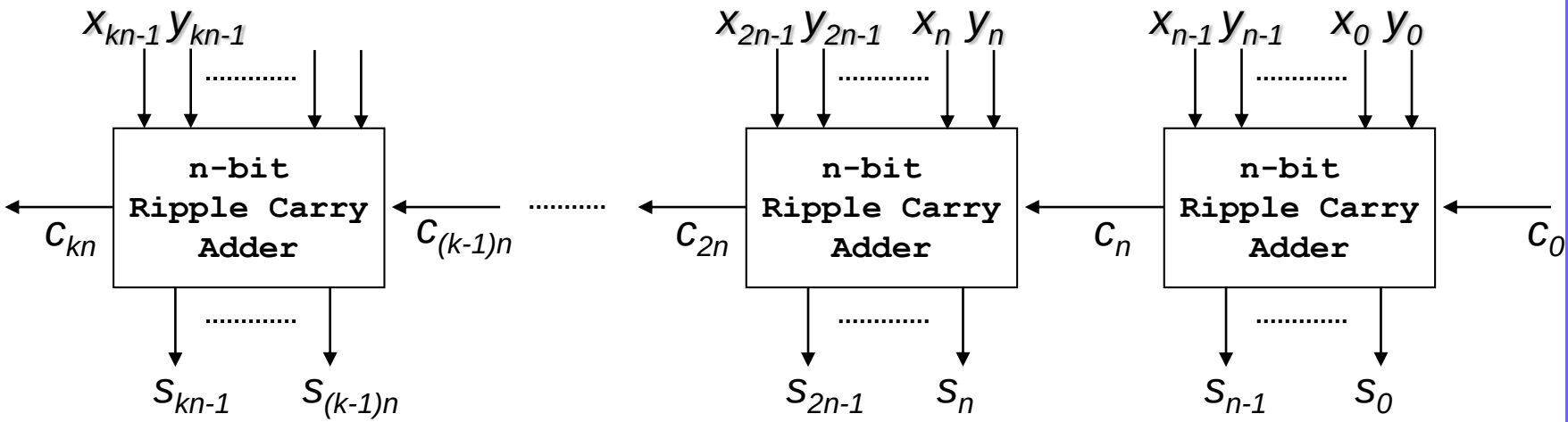


Ripple-Carry Adder



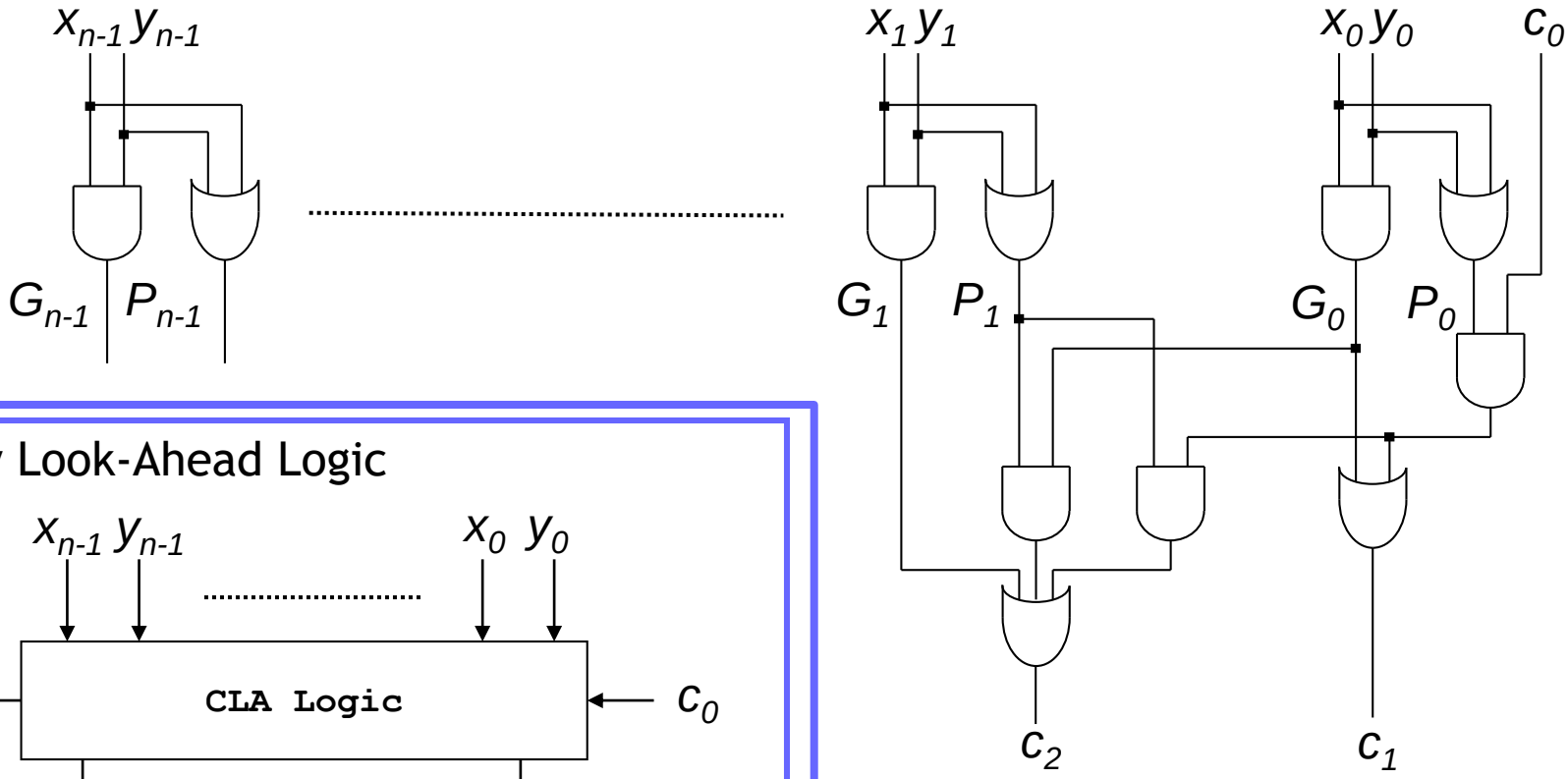
Sommatori: Ripple Carry [2]

Ripple-Carry Block Architecture



Sommatori: Carry Look-Ahead [1]

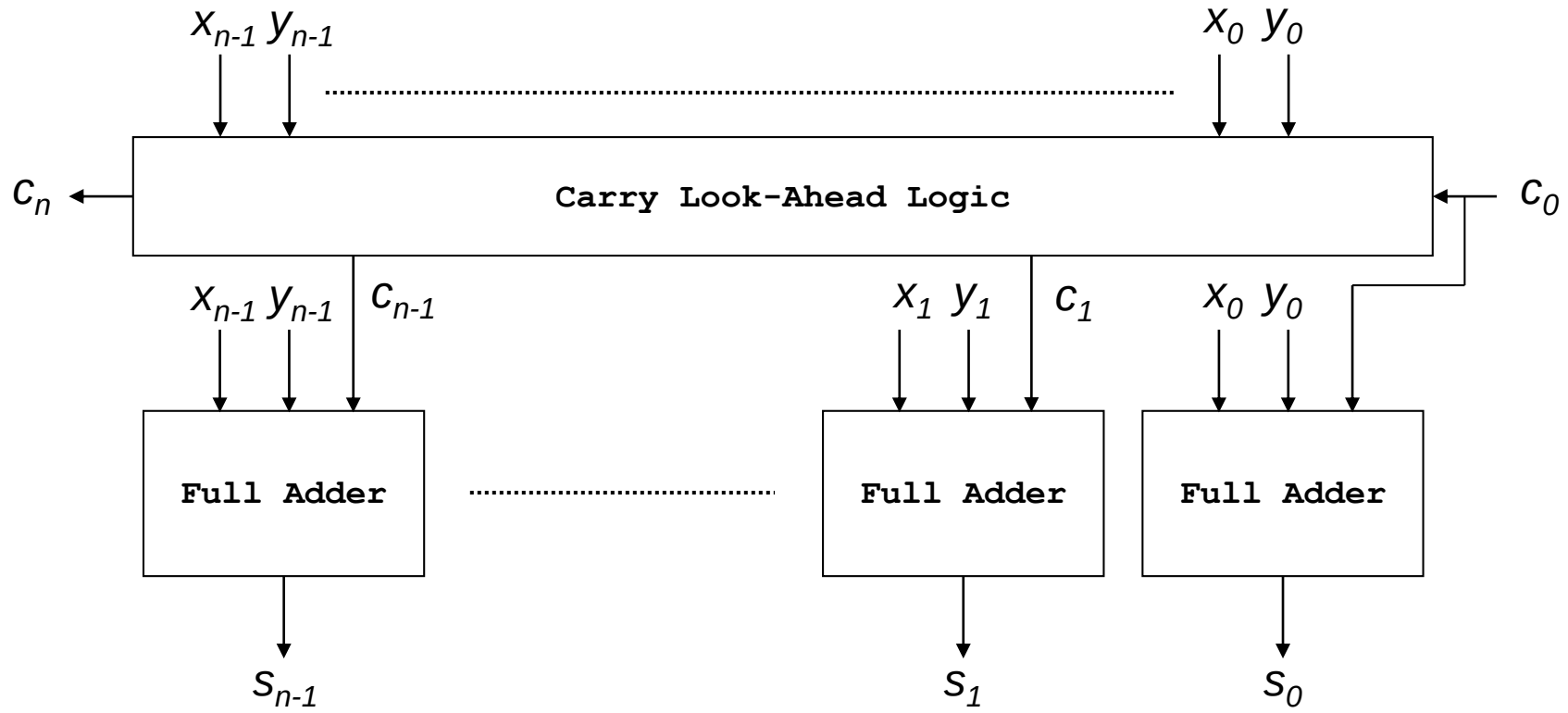
Carry Look-Ahead Logic: Internal architecture



Carry Look-Ahead Logic

Sommatori: Carry Look-Ahead [2]

Carry Look-Ahead Logic



Somma di più valori

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

$$W = X + Y + Z$$

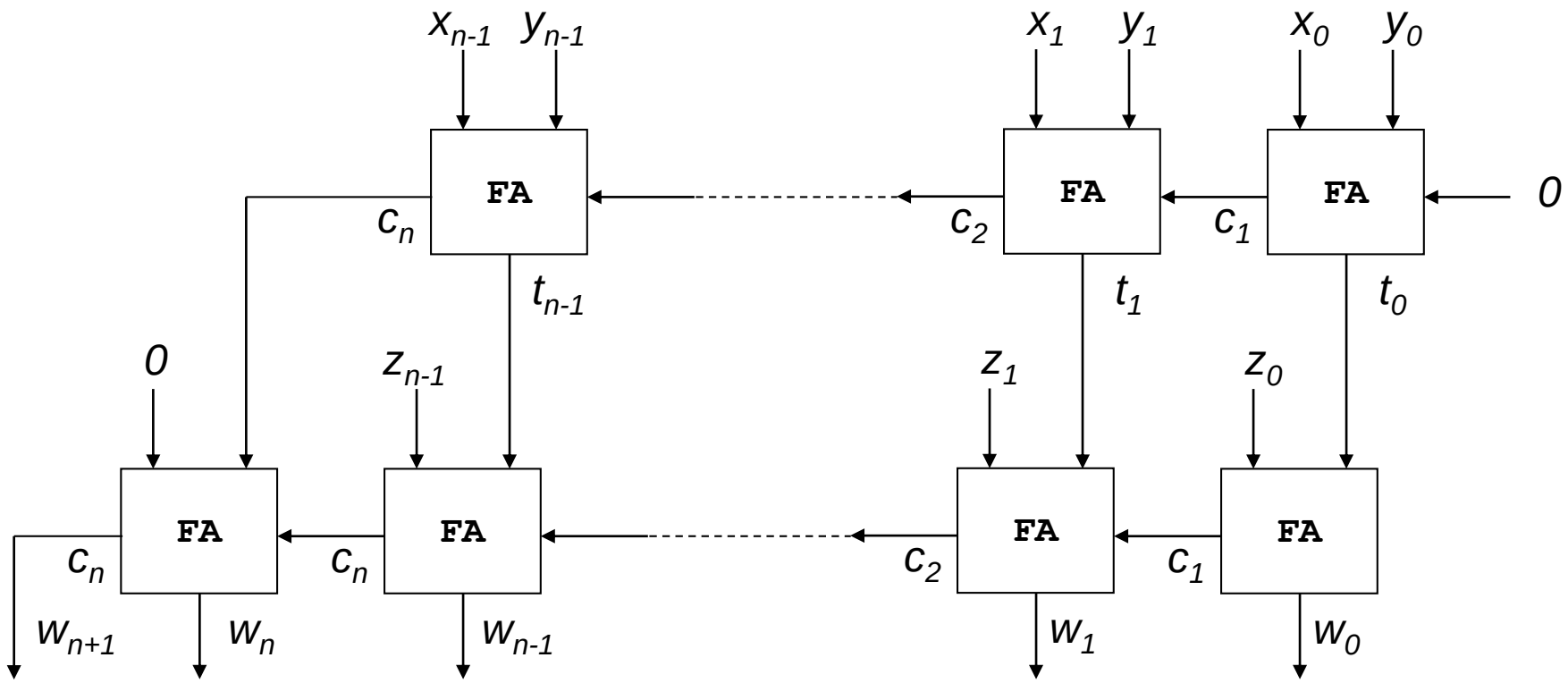
- Soluzione:
  - Calcolare una somma intermedia

    $T = X + Y$
  - 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

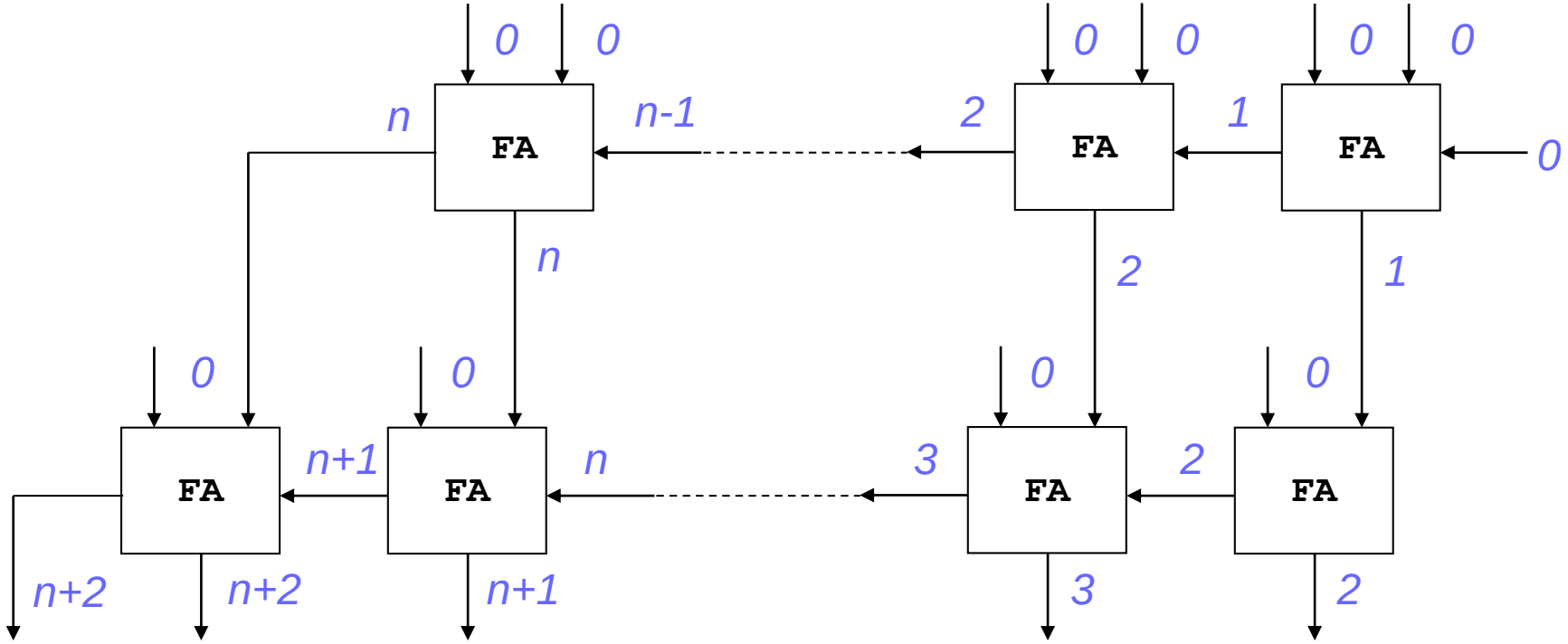
Soluzione con sommatori ripple-carry W = (X + Y) + Z = T + Z



Somma di tre addendi - Prestazioni con sommatori ripple-carry

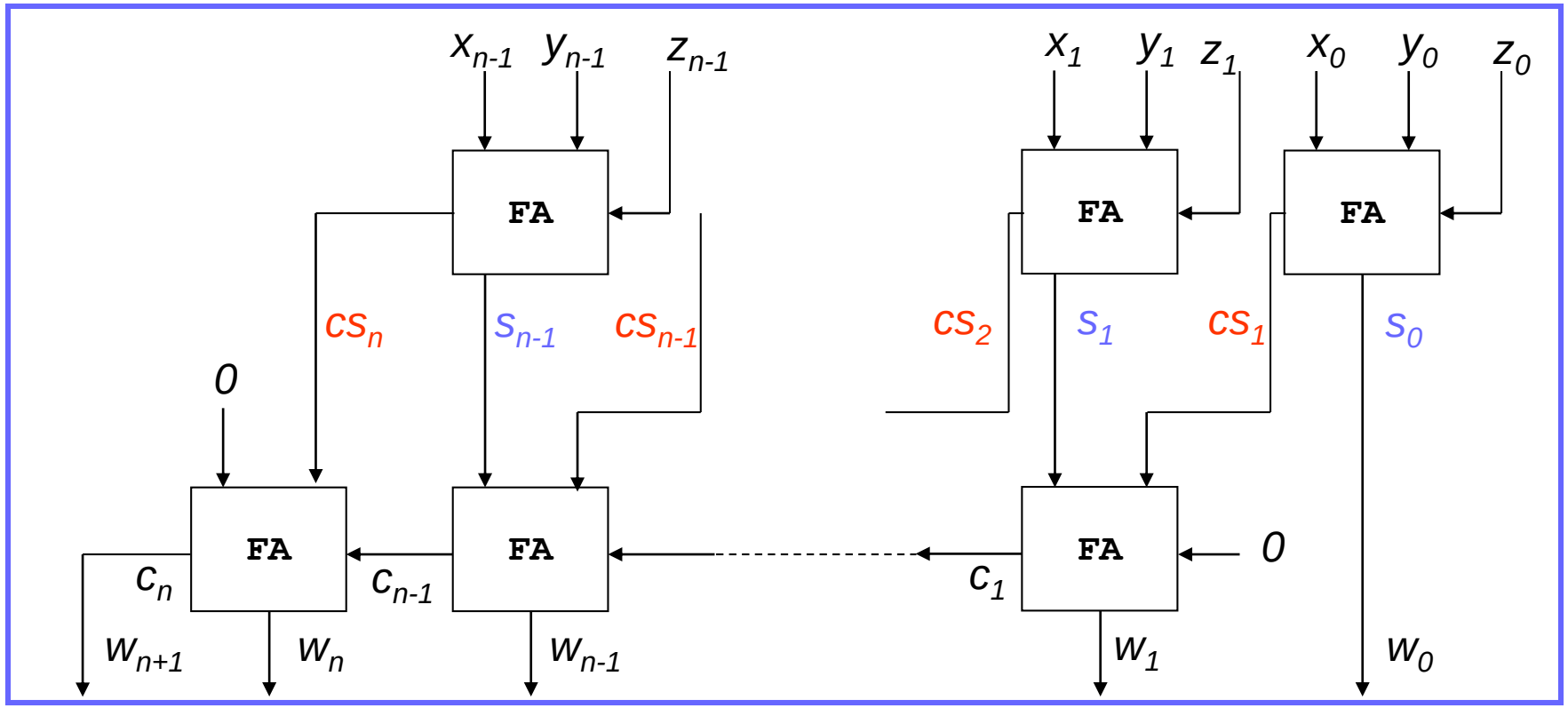
Prestazioni con sommatori ripple-carry (in blu il ritardo di ogni segnale)
Ritardo $R = (n + 2)\Delta T$ con ΔT ritardo di un Full-Adder

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



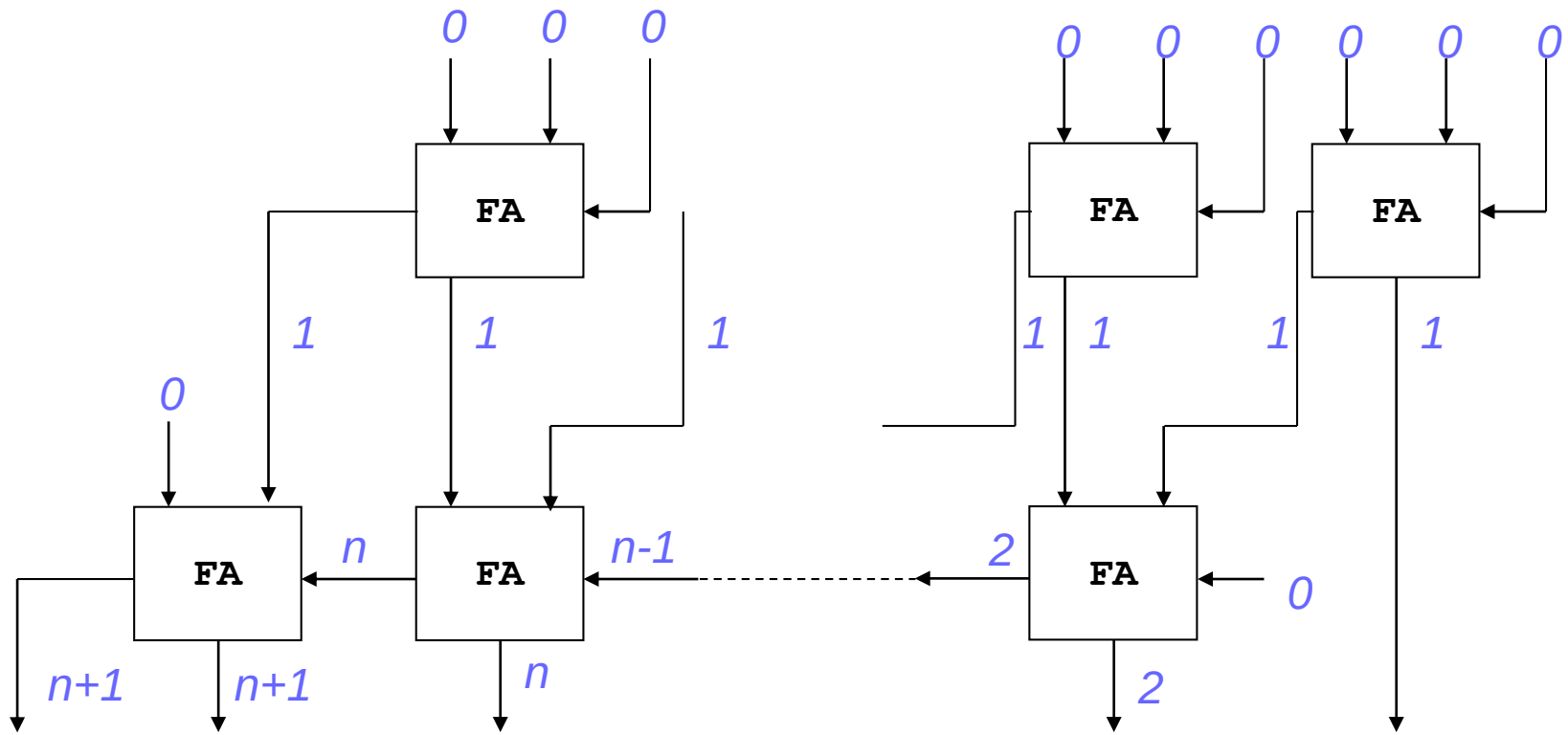
Somma di tre addendi con Sommatori Carry Save

- 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



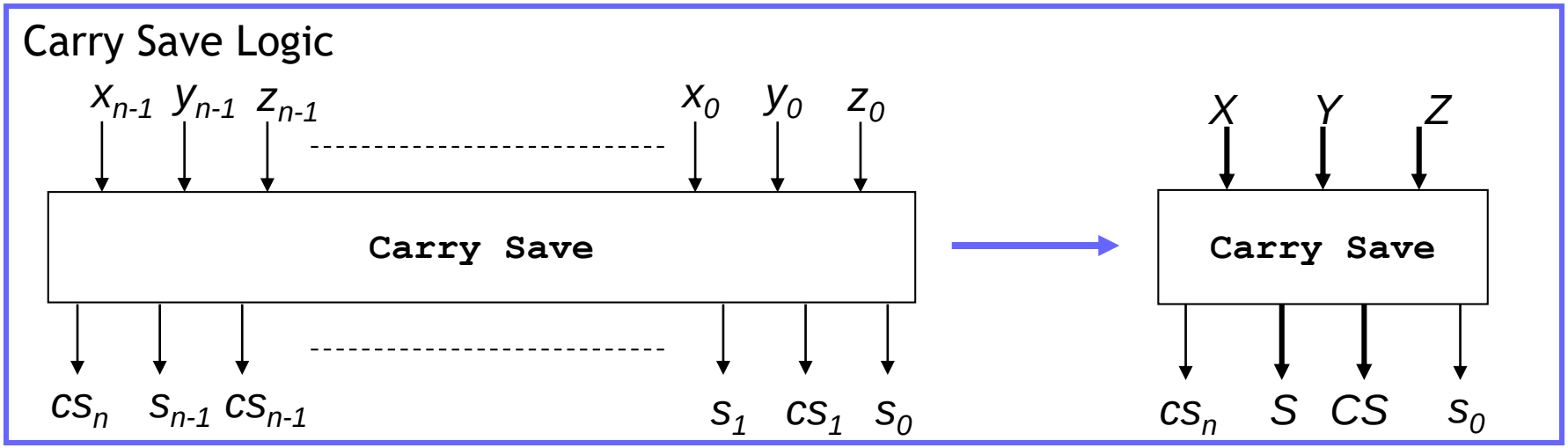
Somma di tre addendi - Prestazioni Carry Save

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

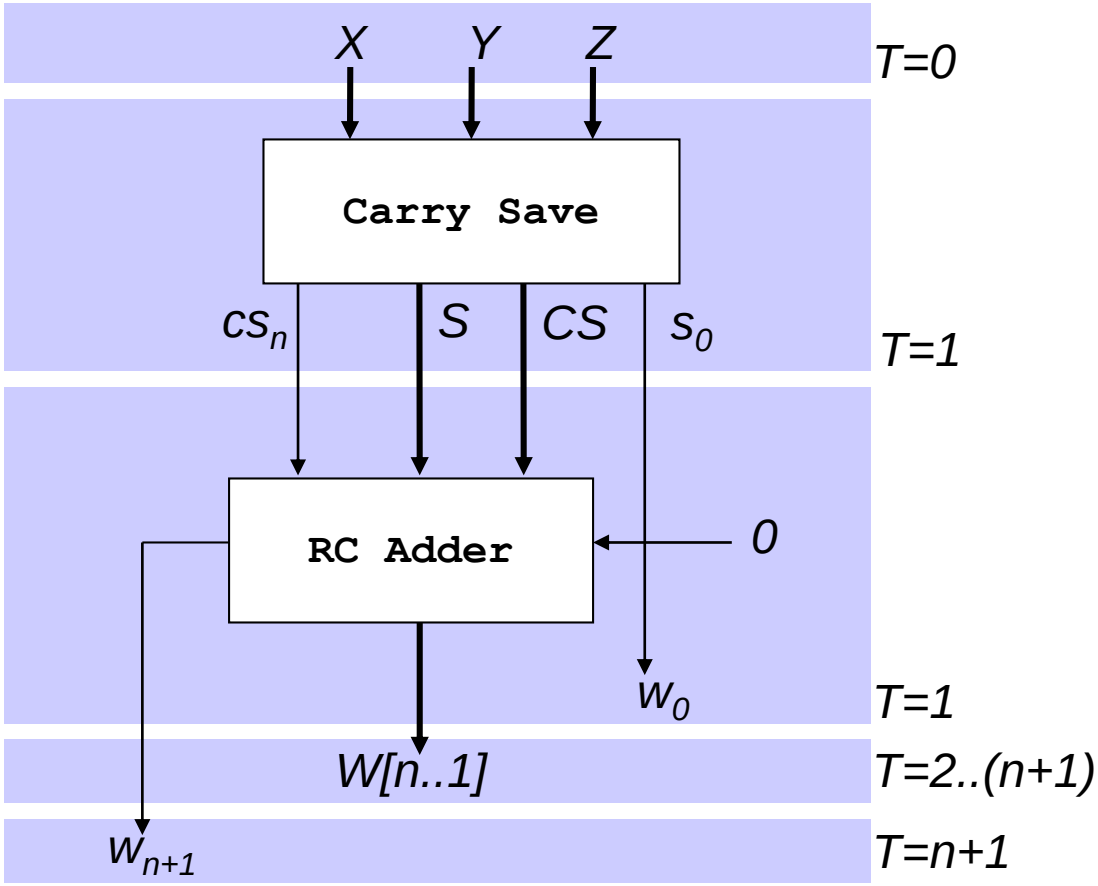


Sommatore Carry Save come blocco

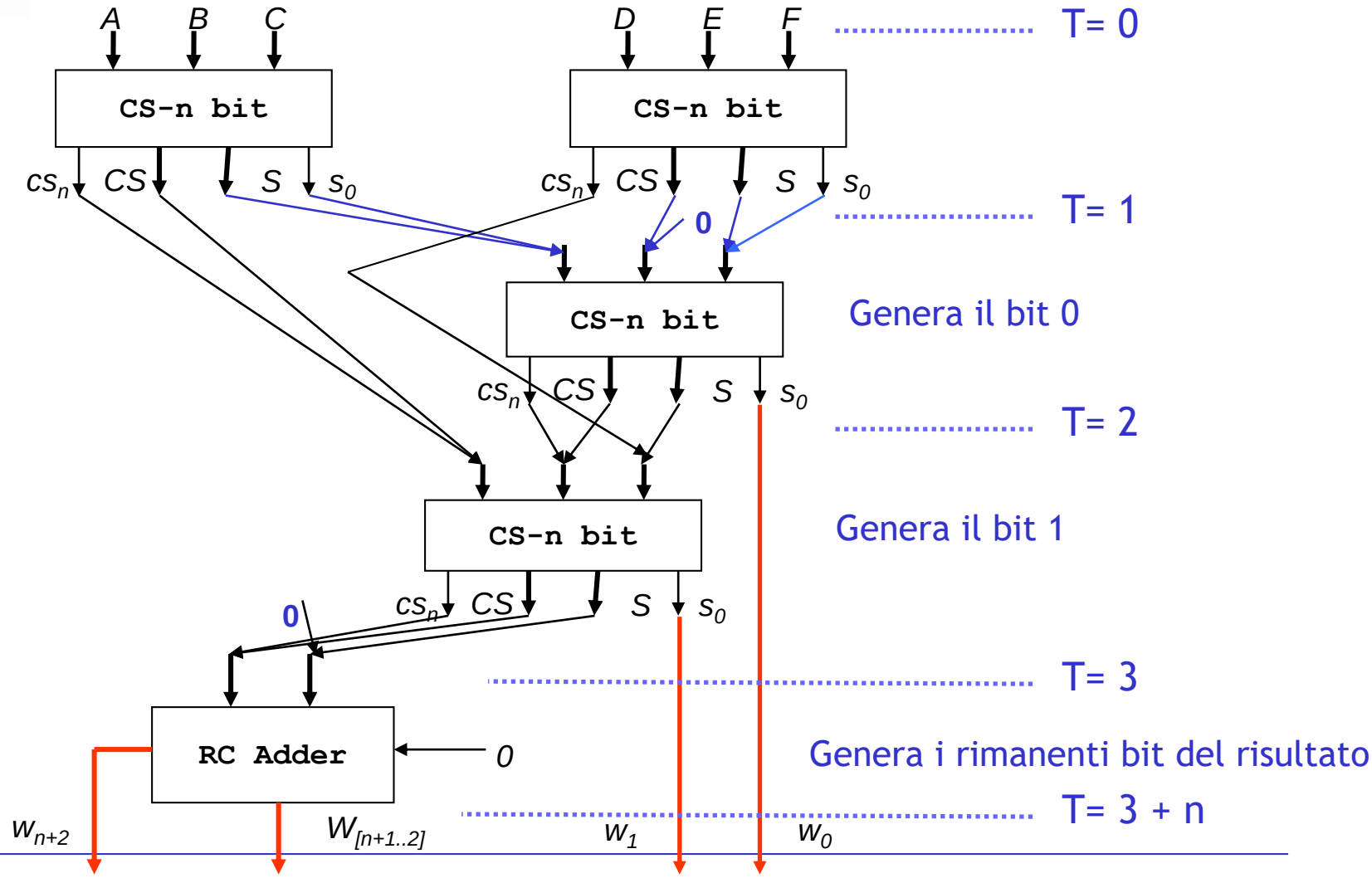
- Sommatore Carry Save composto da due unità
  - Blocco Carry Save:
    - Produce i due vettori S e CS
    - Ritardo: $R_{CS} = 1$
  - Sommatore Ripple-Carry:
    - Produce il risultato finale
    - Ritardo: $R_{RC} = n + 1$



- Istanti di generazione dei bit di uscita

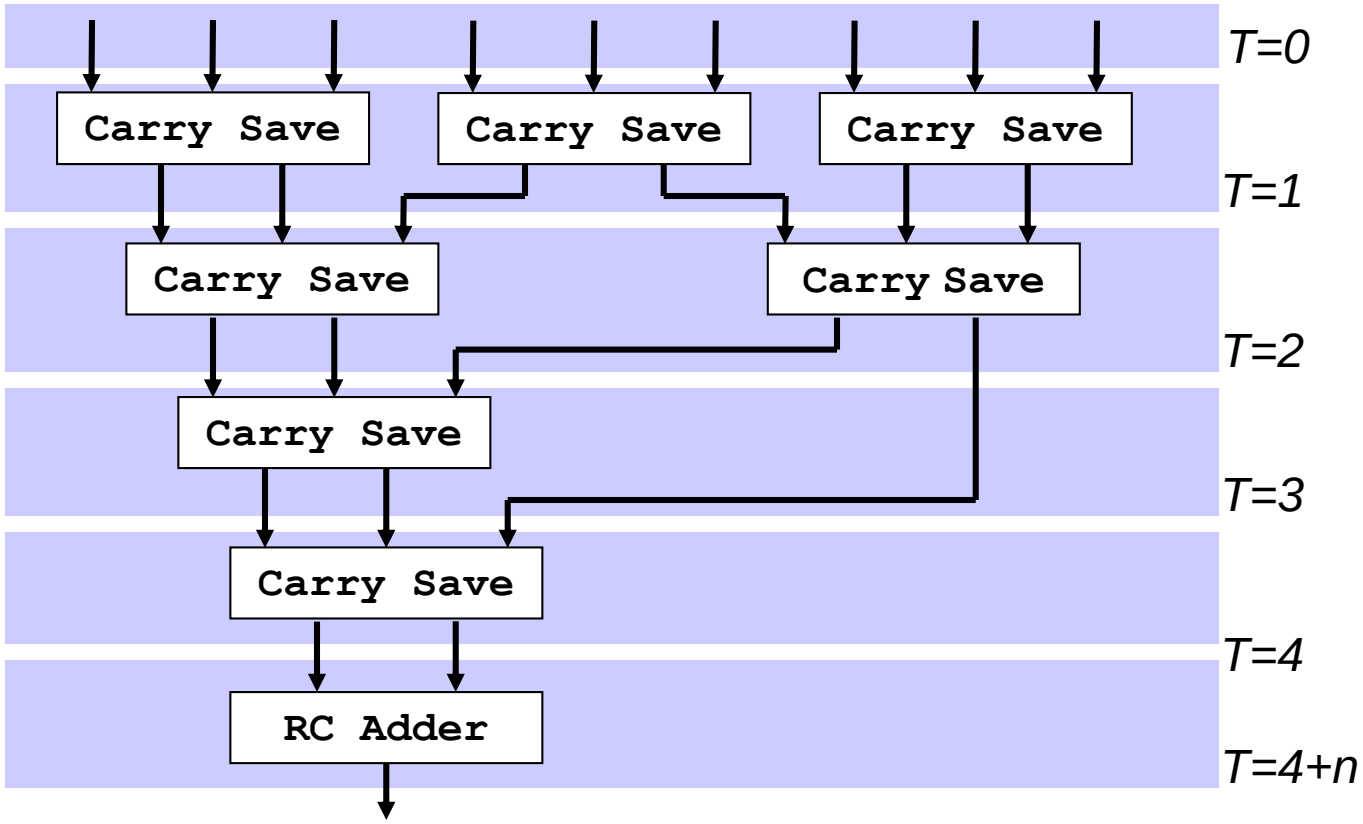


Esempio
sommatore a 6 addendi con blocchi Carry Save da 3 addendi



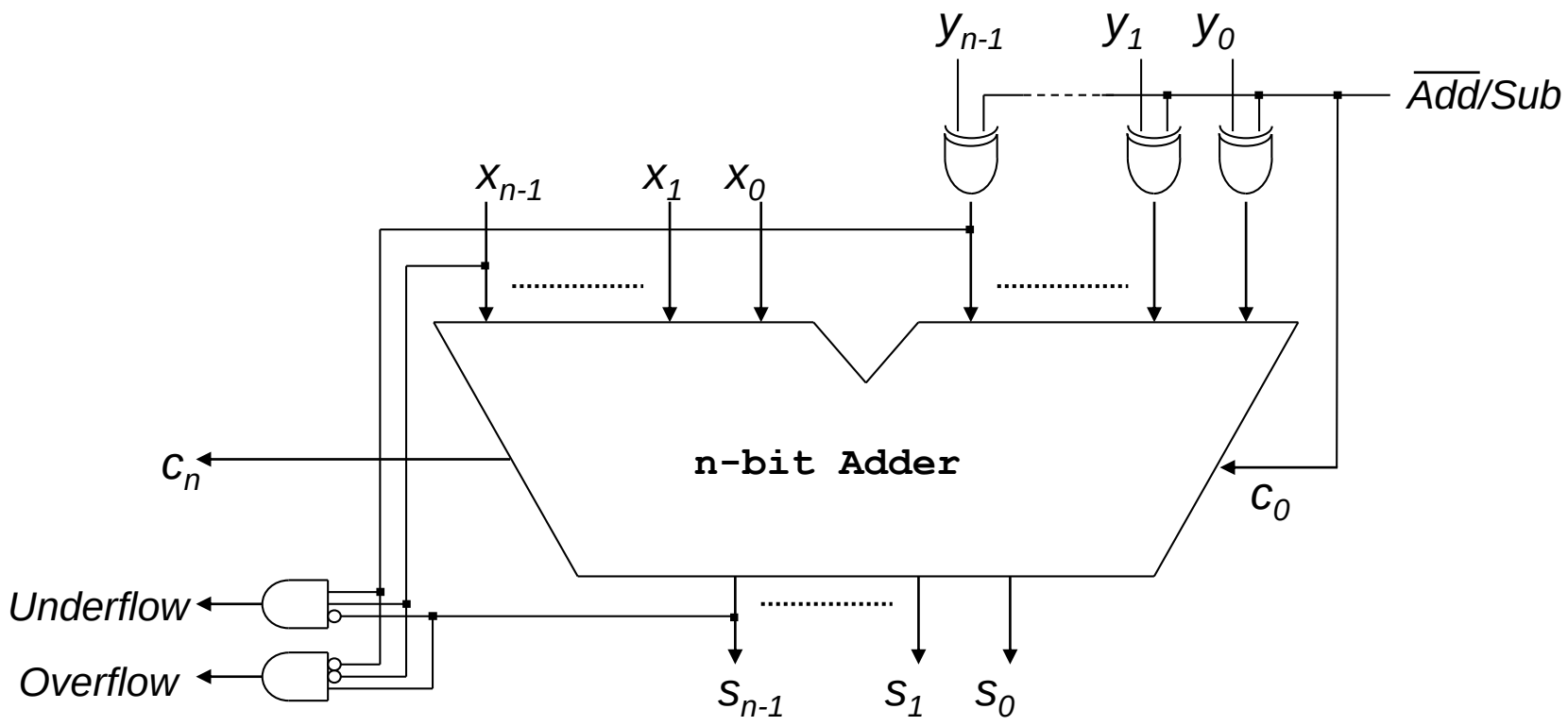
Esempio
sommatore a 9 addendi con blocchi Carry Save da 3 addendi

- Vantaggi più evidenti al crescere del numero degli operandi



Sommatori Add/Subtract: operazioni in complemento a 2

Add/Subtract Architecture



Moltiplicatori combinatori

- Prodotto di due numeri positivi di 3 bit (n bit - $2n$ bit prodotto)

Moltiplicazione bit a bit

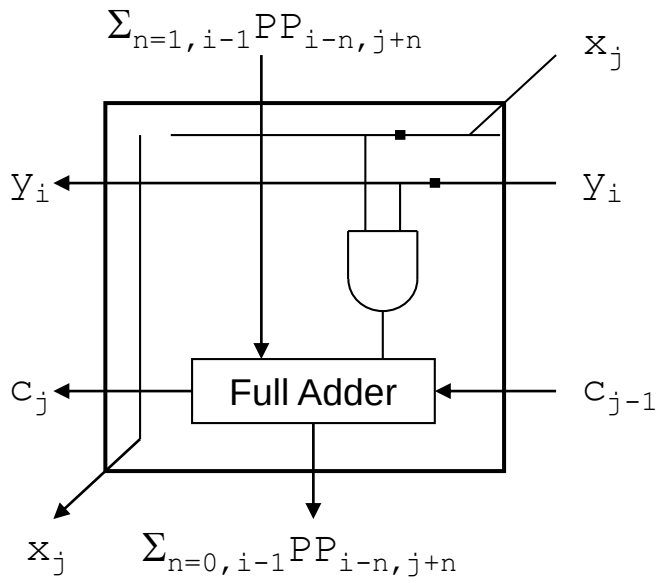
			x_2	x_1	x_0	$\times$
			y_2	y_1	y_0	$=$
			y_0x_2	y_0x_1	y_0x_0	
		y_1x_2	y_1x_1	y_1x_0		
	y_2x_2	y_2x_1	y_2x_0			
			PP_{02}	PP_{01}	PP_{00}	
		PP_{12}	PP_{11}	PP_{10}		
	PP_{22}	PP_{21}	PP_{20}			
p_5	p_4	p_3	p_2	p_1	p_0	

Matrice di prodotti parziali costituita da n righe

Moltiplicatori combinatori: *somma per righe*

- Somma per righe

Multiplier Cell



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

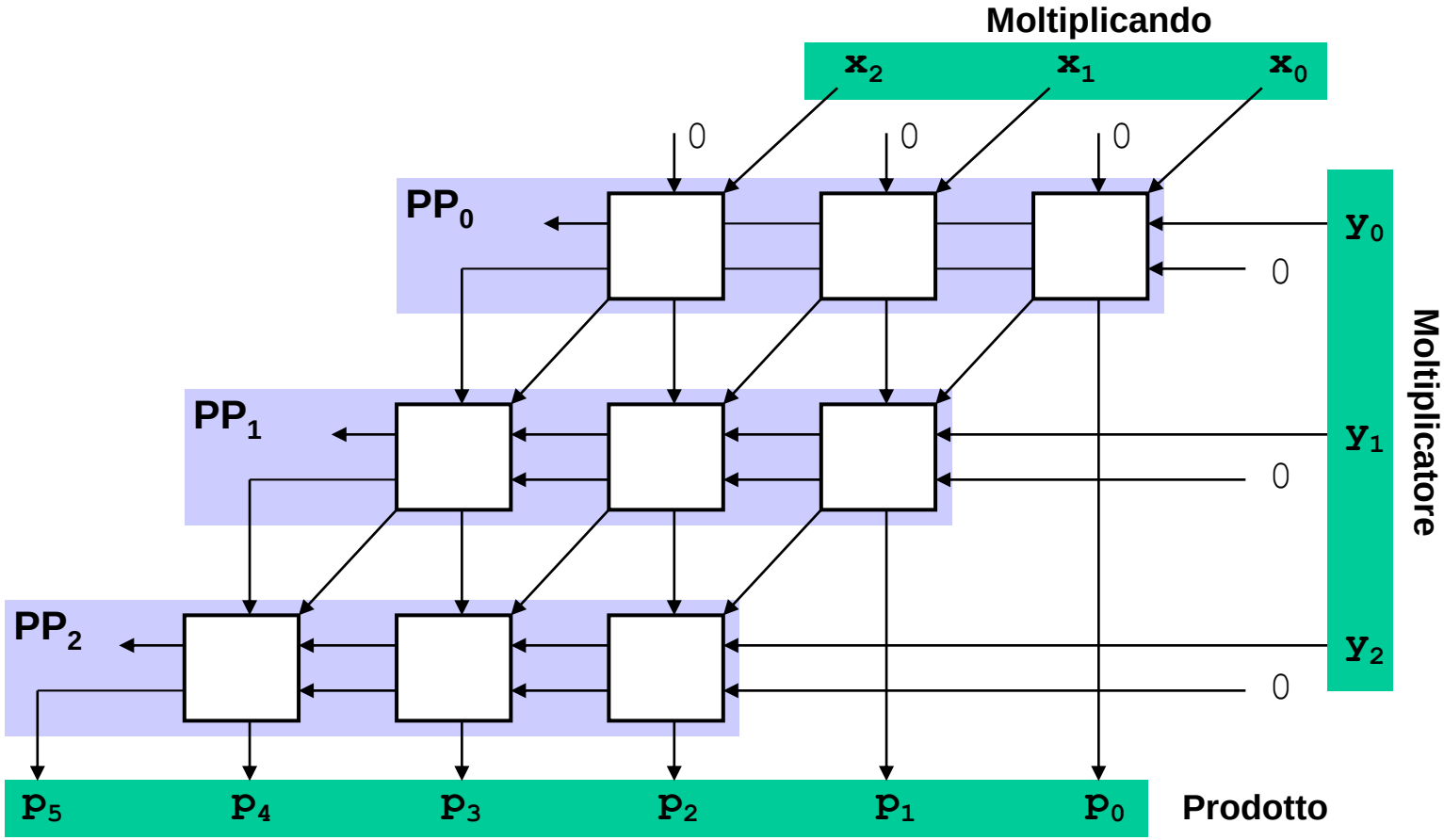
Per il calcolo del prodotto parziale, X si propaga in diagonale e Y in verticale

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*

Moltiplicatori combinatori: *somma per righe*

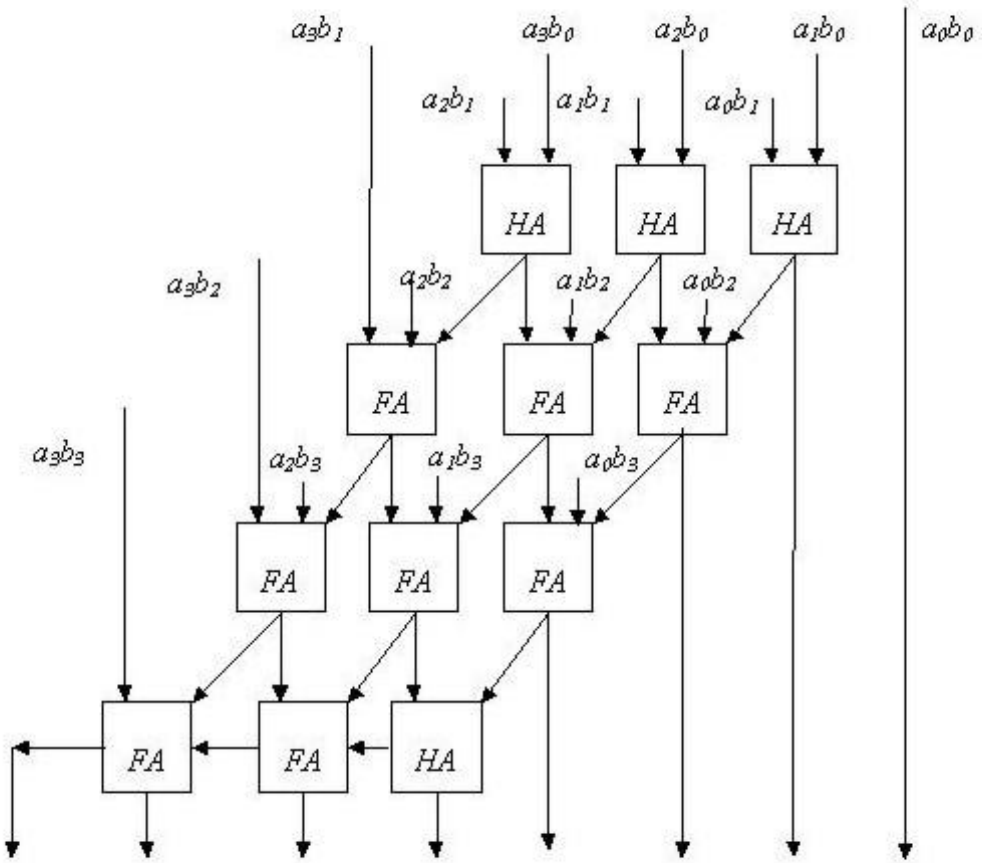


Moltiplicatori combinatori: *somma per diagonali*

- 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.



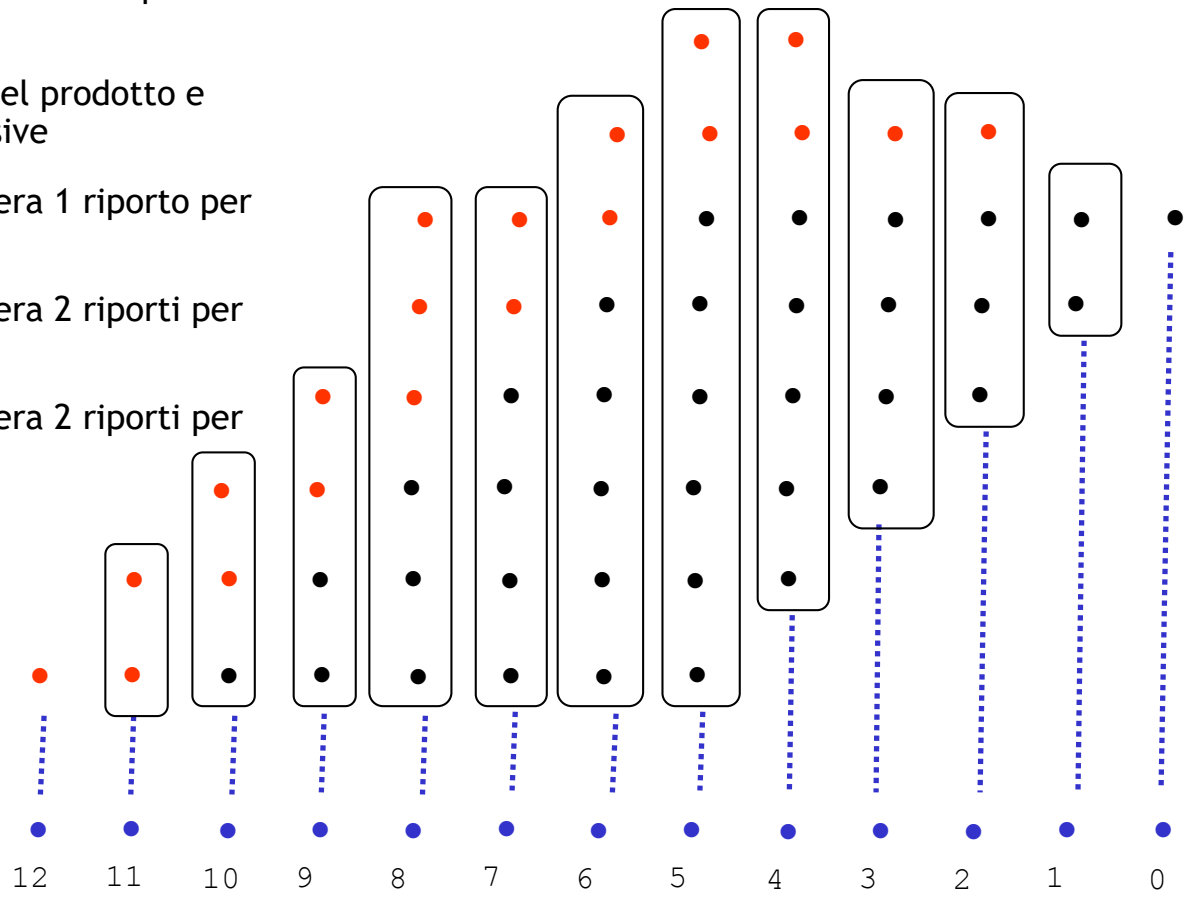
Moltiplicatori combinatori: somma per colonne

- 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

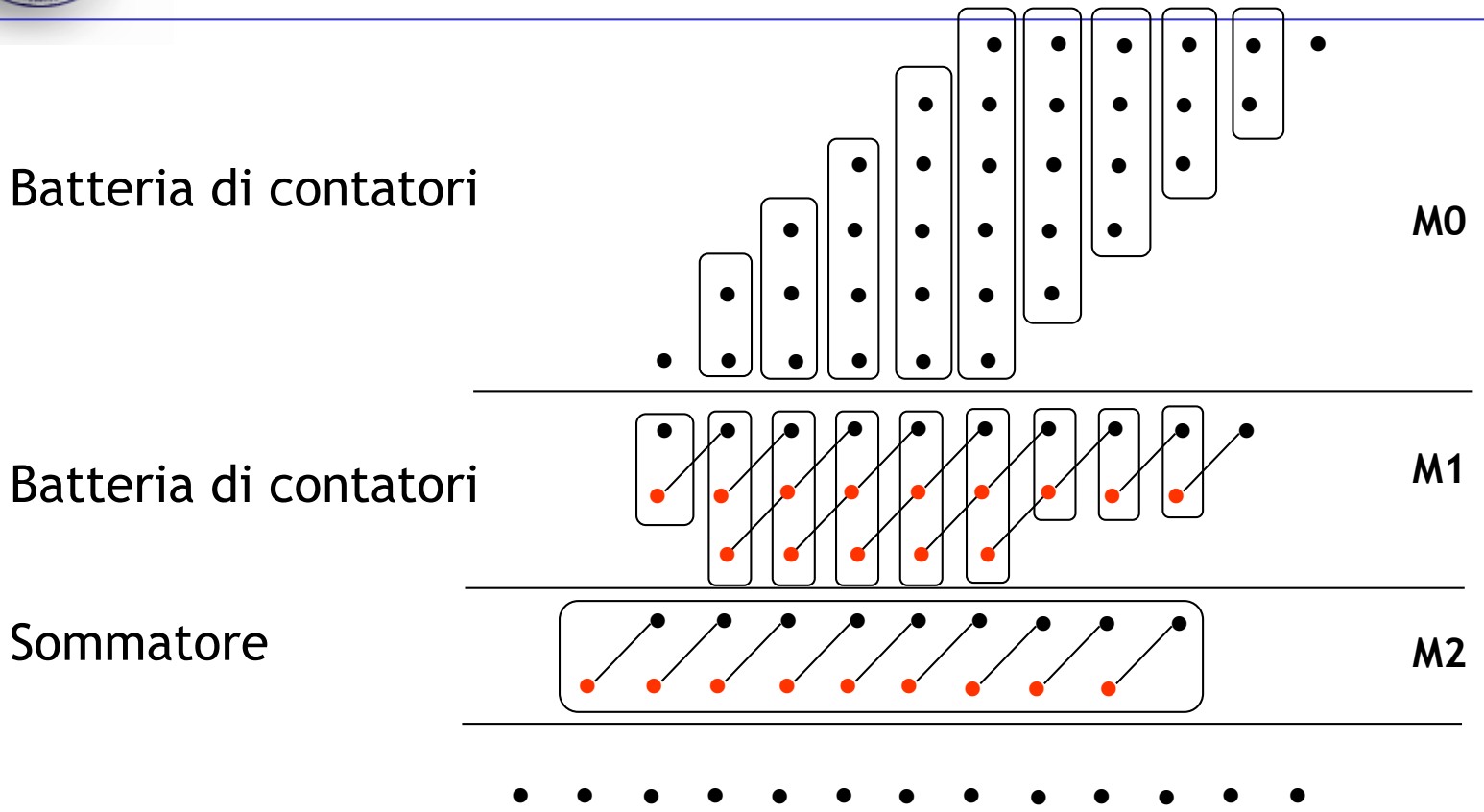
- Moltiplicando e moltiplicatore da 6 bit
- In nero la matrice di AND, in rosso i riporti generati dai contatori
- ogni contatore genera 1 bit del prodotto e riporti per le colonne successive
- il contatore di colonna 1 genera 1 riporto per colonna 2
- il contatore di colonna 2 genera 2 riporti per colonna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

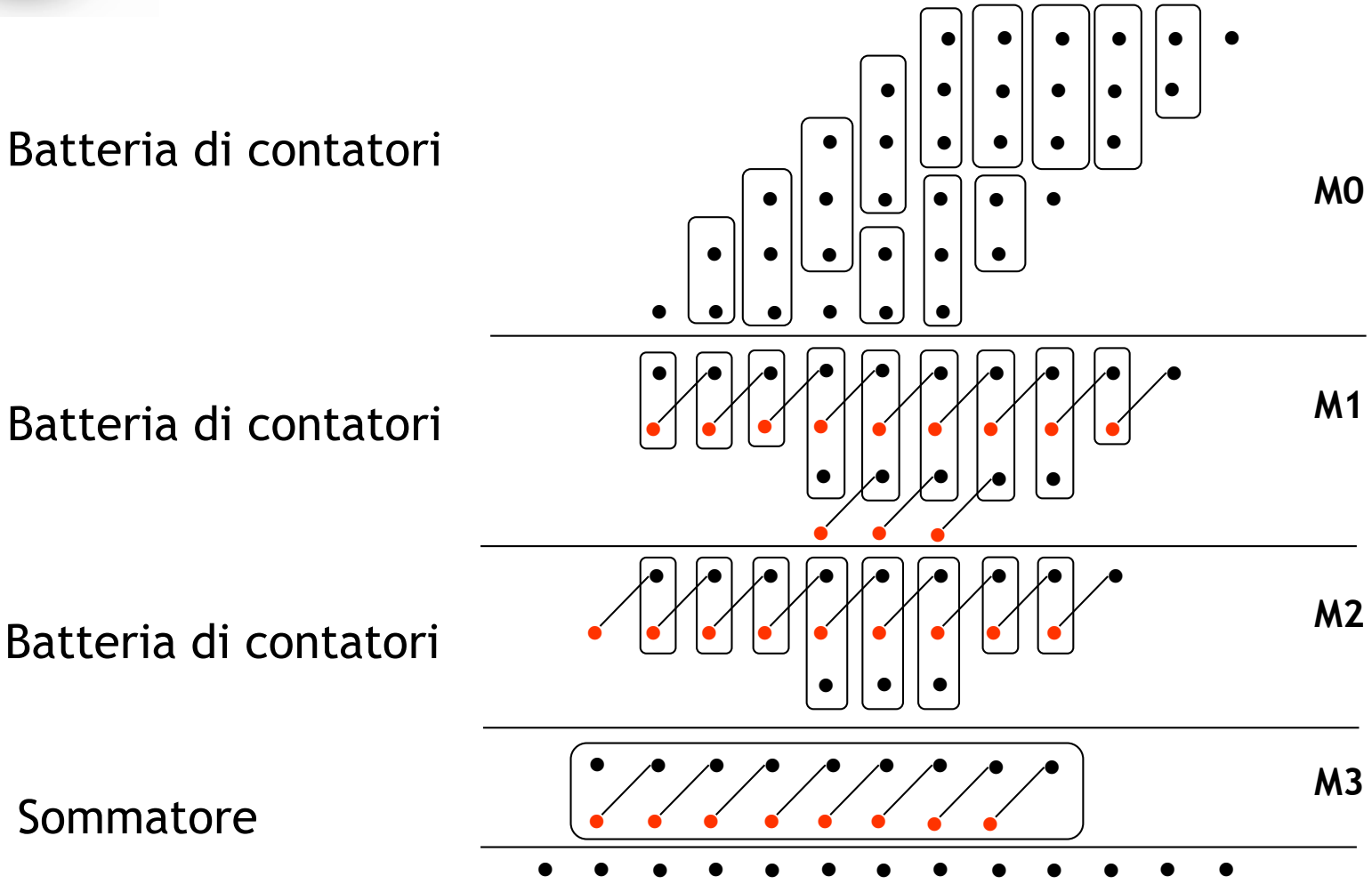
Moltiplicatori combinatori:
somma per colonne con matrici successive



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)

Moltiplicatori combinatori: moltiplicatore di Wallace

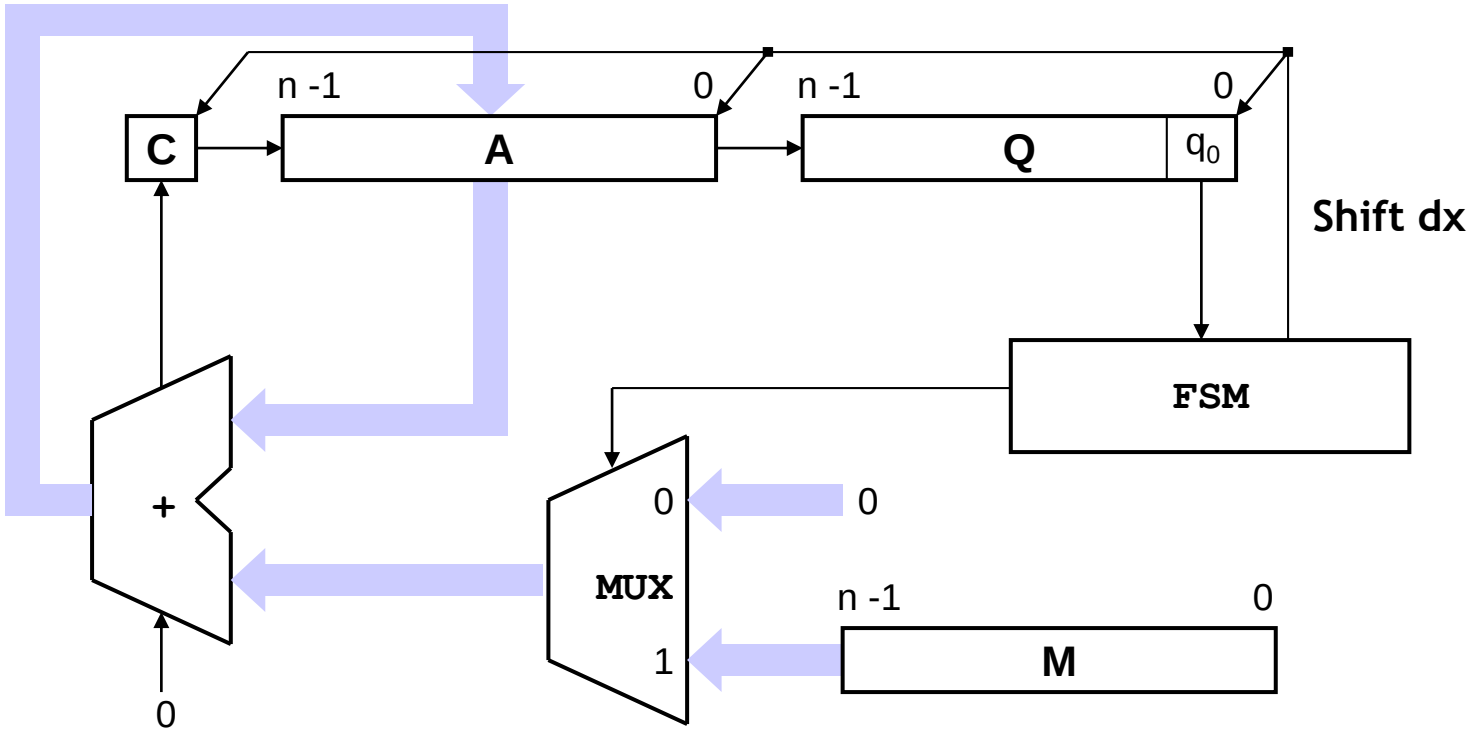


Moltiplicatori sequenziali [1]

- 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



Sommatore in virgola mobile [1]

- I circuiti per la realizzazione delle operazioni in virgola mobile sono molto complessi
- Si consideri l'algoritmo per la somma secondo lo standard IEEE Single Precision:
  - Si sceglie il numero con esponente minore e si fa scorrere la sua mantissa a destra un numero di bit pari alla differenza dei due esponenti
  - Si assegna all'esponente del risultato il maggiore tra gli esponenti degli operandi
  - Si esegue l'operazione di somma tra le mantisse per determinare il valore ed il segno del risultato
  - Si normalizza il risultato cosi' ottenuto
    - Non sempre quest'ultima operazione è necessaria
- Nota:
  - se A o B = $\pm\infty$
  - se A o B = 0
  - se la differenza tra gli esponenti è maggiore o uguale al numero di bit a disposizione per le mantisse
- è inutile fare la somma

Sommatore in virgola mobile [2]

- Nel seguito viene sviluppato un sommatore floating point
- I numeri A e B sono rappresentati
  - Su 32 bit
  - Secondo lo standard IEEE Single Precision
- Gli operandi A e B sono composti come segue:

$$A = \{ S_A, E_A, M_A \}$$

$$B = \{ S_B, E_B, M_B \}$$

- In cui:
  - S_A, S_B Segno,1 bit
  - E_A, E_B Esponente in eccesso 127, 8 bit
  - M_A, M_B Mantissa,23 bit

Sommatore in virgola mobile [3]

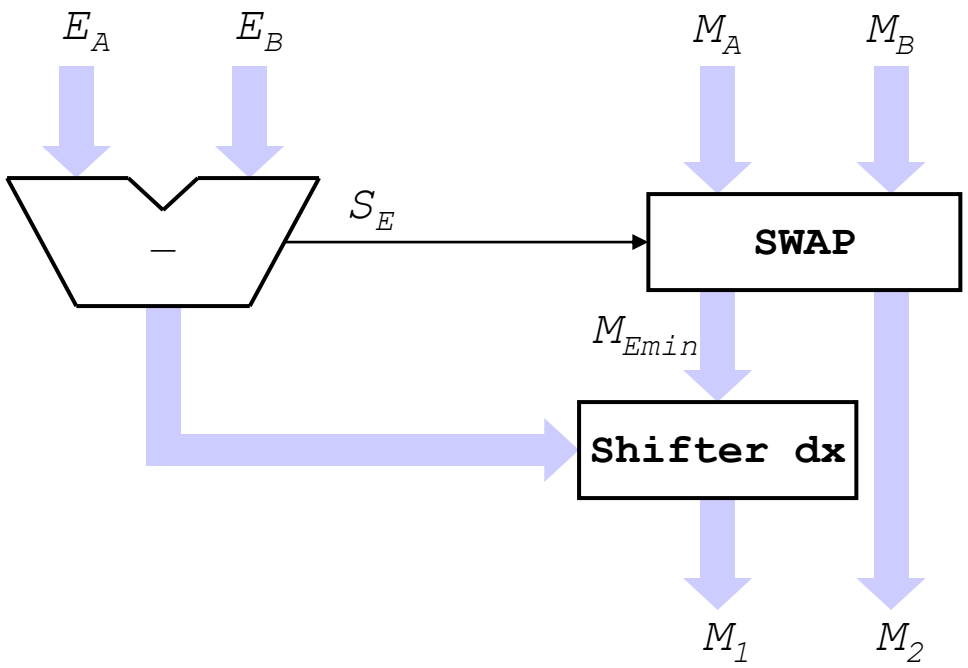
- Passo 1
  - *Si sceglie il numero con esponente minore e si fa scorrere la sua mantissa a destra un numero di bit pari alla differenza dei due esponenti*

- Richiede le seguenti operazioni:
  - Individuazione dell'esponente minore E_{min}
  - Calcolo della differenza tra gli esponenti $d = |E_A - E_B|$
  - Selezione della mantissa dell'operando con esponente M_{Emin}
  - Scorrimento della mantissa M_{Emin} di d posizioni a dx (tenendo conto dell'1 implicito)
- Il calcolo della differenza tra gli esponenti consente allo stesso tempo (analizzandone il segno S_E) di individuare l'esponente minore

Sommatore in virgola mobile [3]

- Questa prima sezione del sommatore calcola:
  - M_1 La mantissa del numero con esponente minore, opportunamente shiftata
  - M_2 La matissa del numero con esponente maggiore

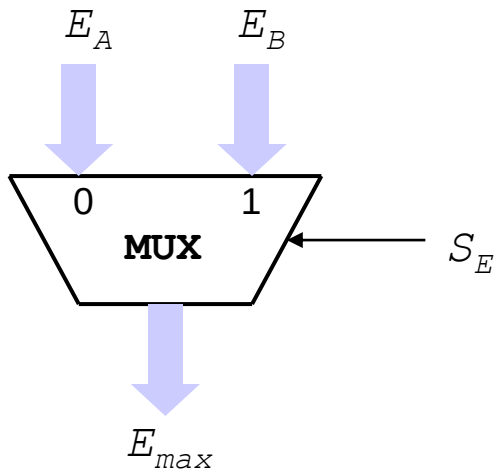
Passo 1



Sommatore in virgola mobile

- Passo 2
  - *Si assegna all'esponente del risultato il maggiore tra gli esponenti degli operandi*
- Richiede le seguenti operazioni:
  - Selezione dell'esponente minore E_{max} in base a S_E.
  - Si riutilizza il segno S_E della differenza tra gli esponenti

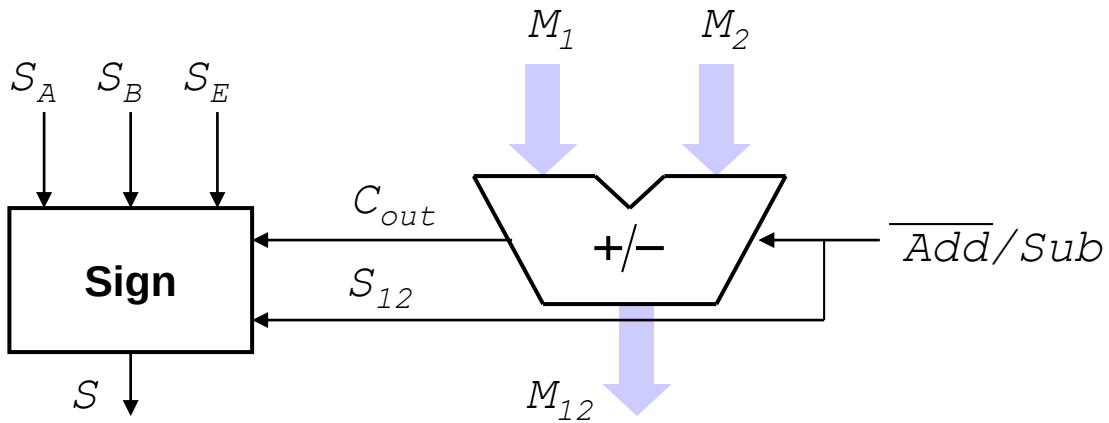
Passo 2



Sommatore in virgola mobile

- Passo 3
  - *Si esegue l'operazione di somma tra le mantisse per determinare il valore ed il segno del risultato*
- Richiede le seguenti operazioni:
  - Calcolo della somma algebrica M_{12} delle mantisse M_1 ed M_2 ottenute al primo passo, e relativo segno
  - Si utilizza un sommatore/sottrattore su 24 bit con riporto

Passo 3



Sommatore in virgola mobile

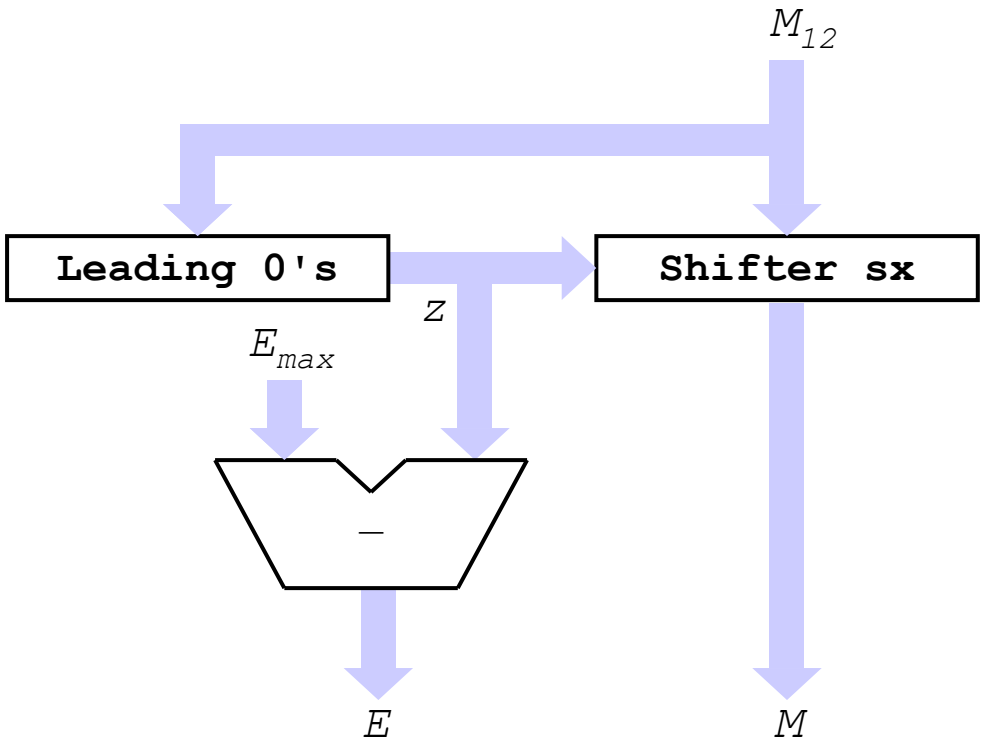
- Se C_{out} = 0 e M_{12} normalizzata $E = E_{max}$

- Passo 4
  - *Si normalizza il risultato così ottenuto*

- Richiede le seguenti operazioni
  - Se C_{out} = 1, Shdx M_{12} e $E = E_{max} + 1$, eventuale troncamento
- Altrimenti (C_{out} = 0 e M_{12} non normalizzata)
  - Individuazione del numero z degli zeri nei bit più significativi della mantissa M_{12}
  - Shift sx della mantissa M_{12}
  - Calcolo del nuovo esponente $E = E_{max} - z$
  - A tale scopo sono necessari:
    - Un circuito per il calcolo dei *leading zeroes*
    - Un sottrattore su 8 bit
    - Uno *shifter* per l'allineamento della mantissa

Sommatore in virgola mobile

Passo 4



Sommatore in virgola mobile