Circuiti Addizionatori

Corso di Reti Logiche, a.a. 2008/09 - ing. Alessandro Cilardo

Si ringrazia Dario Socci per il contributo fornito alla realizzazione di queste dispense

Circuiti Addizionatori

Gli addizionatori sono un elemento critico all'interno dei sistemi digitali, sia perché coinvolti in un vasto numero di operazioni elementari, sia perché spesso responsabili del massimo ritardo combinatorio presente all'interno del sistema (in un sistema sincrono, tale ritardo condiziona la massima frequenza operativa raggiungibile). Per questi motivi, nel corso degli ultimi decenni è stata presentata un'ampia varietà di soluzioni circuitali per l'operazione di addizione, tuttora oggetto di ricerca. In questo capitolo vengono analizzati alcuni degli addizionatori più importanti. Per ogni addizionatore, viene presentato lo schema architetturale ed alcune considerazioni legate alle prestazioni ed al costo realizzativo. Vari tipi di addizionatori, data la loro importanza, sono descritti in diversi testi, tra cui [1, 2, 3]. In [16] sono presentate anche molte implementazioni specifiche.

1 Addizionatori binari

Per comprendere la strutturazione dei circuiti addizionatori, è utile enfatizzare il procedimento elementare tramite il qualche può essere calcolata la somma di due numeri rappresentati in binario (Figura 1). La figura riporta i passi elementari

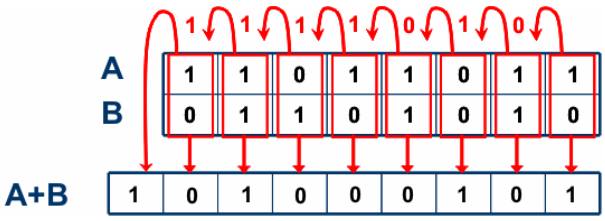

Figura 1: Un esempio di addizione tra numeri in rappresentazione binaria.

dell'operazione di somma, procedendo dalla coppia di cifre meno significative, quelle più a destra, verso le cifre più significative. Ogni coppia di cifre è sommata insieme ad un eventuale riporto entrante prodotto dalla coppia di cifre a destra, producendo a sua volta un bit di somma ed un eventuale bit di riporto che sarà tenuto in conto nella somma di bit successiva. L'operazione ha una strutturazione evidentemente **iterativa**, e pertanto si presta ad una realizzazione "bit-sliced", ovvero ottenuta implementando circuitalmente l'operazione elementare (somma di due bit più il riporto), ed iterando n volte il circuito per ottenere la somma ad n bit.

Si noti che, a differenza delle generiche coppie di cifre interne, la coppia più a destra non necessita di ricevere un riporto in ingresso, e richiede quindi un'operazione di somma più semplice.

Per studiare in dettaglio l'implementazione circuitale dell'addizione, partiamo proprio dalla coppia di cifre più a destra. In questo caso, abbiamo bisogno di una rete che riceva due bit in ingresso (A_i e B_i) e produca due bit in uscita (S, il bit di somma, e C_{out}, il riporto, o *carry* in uscita). La somma richiesta può essere vista come la funzione logica definita dalla seguente tabella di verità:

A_i	B_i	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabella 1: Tabella di verità dell'Half-Adder

Dalla tabella si deduce che la funzione somma può essere descritta dalle seguenti equazioni:

$$S = A_i \oplus B_i \tag{1}$$

$$C_{out} = A_i \cdot B_i \tag{2}$$

La rete che implementa questa coppia di operazioni è chiamata *Half-Adder* (Figura 2).

Per le coppie di cifre interne occorre tenere in conto anche il riporto entrante da destra. In altre parole, la cella che implementa l'operazione elementare dovrà questa volta avere tre ingressi, A_i, B_i ed il riporto entrante C_{in}. La rete che implementa queste operazioni è chiamata *Full-Adder* e realizza la tabella di verità riportata sotto (Tabella 2). L'uscita S del Full-Adder è ancora una XOR dei tre ingressi, mentre l'uscita C_{out} è alta se sono contemporaneamente alti almeno due dei tre ingessi (funzione maggioranza).

Si noti che, assegnando peso $2^1 = 2$ all'uscita C_{out} e peso $2^0 = 1$ all'uscita S, il Full-Adder genera la codifica binaria del numero di 1 presenti ingresso. In altri termini, il valore $2C_{out} + S$ è uguale al numero di ingressi pari ad 1, come si può facilmente verificare dalla tabella. Ad esempio, quando in ingresso si ha

A_i	B_i	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

Tabella 2: Tabella di verità di un Full-Adder.

$A_i = 1$, $B_i = 0$ e $C_{in} = 1$ (due 1 in ingresso), l'uscita sarà $C_{out} = 1$ e $S = 0$, che può essere letta come il valore 2 in binario. Questa proprietà si interpreta facilmente osservando l'esempio di somma in Figura 1: il bit di somma S ha lo stesso peso dei bit sommati A_i e B_i, e viene pertanto scritto nella stessa colonna; il bit C_{out} ha invece peso aumentato di 1 e viene sommato insieme alla cifre alla sinistra di quelle correnti A_i e B_i. Questa proprietà è vera anche per l'Half-Adder, sebbene in quel caso siano presenti solo due ingressi, e quindi il massimo valore codificato dalle due uscite C_{out} ed S sarà pari a $(10)_2 = 2$. In molte circostanze, è utile considerare Full-Adder ed Half-Adder come *contatori*, sulla base della proprietà appena vista.

Dalla tabella di verità, è possibile notare che le equazioni del Full-Adder possono essere scritte come:

$$S = A_i \oplus B_i \oplus C_{in} = (A_i \oplus B_i) \oplus C_{in}$$

e, minimizzando opportunamente la funzione C_{out},

$$C_{out} = A_i B_i + A_i C_{in} + B_i C_{in} = A_i B_i + C_{in}(A_i + B_i)$$

È facile notare che l'espressione di C_{out} può essere riscritta come

$$C_{out} = A_i B_i + A_i C_{in} + B_i C_{in} = A_i B_i + A_i B_i C_{in} + A_i \overline{B_i} C_{in} + A_i B_i C_{in} + \overline{A_i} B_i C_{in}$$

per assorbimento, questa espressione diventa

$$A_i B_i + A_i \overline{B_i} C_{in} + \overline{A_i} B_i C_{in} = A_i B_i + C_{in}(A_i \oplus B_i)$$

Nella realizzazione del circuito Full-Adder, è spesso utile costruire prima i segnali

$$P = A_i \oplus B_i$$

$$G = A_i B_i$$

Questi segnali possono essere generati da un Half-Adder. Avendo a disposizione i segnali P e G, per ottenere le uscite del Full-Adder occorre ancora calcolare

$$S = P \oplus C_{in}$$

$$C_{out} = G + C_{in} \cdot P$$

Le operazioni di XOR e AND necessarie per ottenere S e C_{out} a partire da P e G possono essere ottenute con un secondo Half-Adder, come mostrato nella Figura 3. Un Full-Adder può quindi essere realizzato a partire da due Half-Adder, più una porta OR.

2 Ripple-Carry Adder (RCA)

Il metodo più diretto per realizzare un addizionatore ad n bit è rappresentato dal Ripple-Carry Adder (RCA), o addizionatore a propagazione del riporto. Questo circuito richiede n Full-Adder in cascata, ovvero con il riporto uscente dell'i-esimo Full-Adder collegato al riporto entrante dell'$(i+1)$-esimo Full-Adder, come mostrato in Figura 4.

Figura 2: Half-Adder

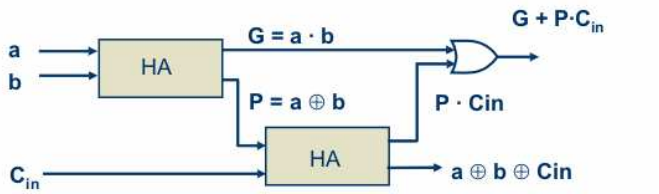

Figura 3: Full-Adder realizzato con due Half-Adder

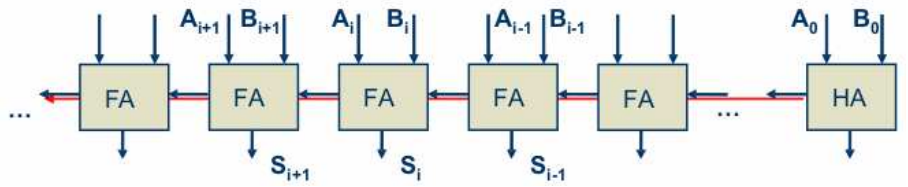

Figura 4: Ripple-Carry Adder. In rosso è evidenziato il percorso critico.

L'addizionatore RCA non fa nient'altro che calcolare la somma così come verrebbe calcolata a mano (vedi esempio nel paragrafo precedente). La sua architettura è semplice, ma anche lenta. Il tempo di calcolo nel caso peggiore, infatti, dipende linearmente dal numero di stadi, in quanto bisogna attendere che il riporto si propaghi dalla prima all'ultima cella per avere il risultato corretto. Se si indica con T_{FA} il tempo di propagazione di un Full-Adder, il tempo di propagazione di un RCA vale, nel caso peggiore,

$$T_{RCA} = nT_{FA} \tag{3}$$

La Figura 4 mostra in rosso il *cammino critico* del circuito, ovvero il percorso lungo il quale si propagherà il ritardo peggiore. In maniera simile, il numero di porte necessarie per realizzarlo risulta essere:

$$A_{RCA} = nA_{FA} \tag{4}$$

2.1 Ottimizzazione dell'RCA

In alcune tecnologie è più semplice realizzare un addizionatore che calcoli $\overline{S}$ ed $\overline{C_{out}}$. In tal caso, il componente elementare che si ha a disposizione è quello rappresentato in Figura 5. Per costruire un addizionatore a propagazione del

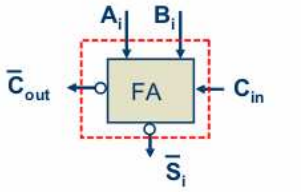

Figura 5: Full-Adder con uscite negate.

riporto, pertanto, occorrerà utilizzare due porte NOT per avere i valori di S

e C_{out}. Come è facile notare in Figura 6, un RCA costruito in questo modo presenta delle porte NOT aggiuntive sul cammino critico, che rallentano ulteriormente il calcolo. Si può tuttavia osservare che la tabella di verità del

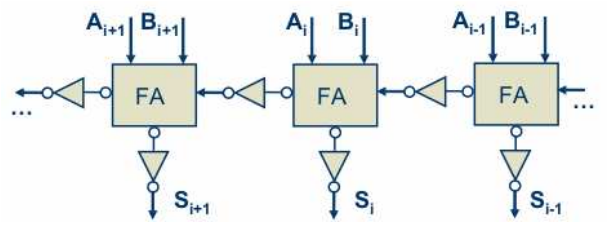

Figura 6: Addizionatore a propagazione basato su FA con uscite negate.

Full-Adder (Tabella 4) gode della seguente proprietà:

$$S(\overline{A}, \overline{B}, \overline{C_{in}}) = \overline{S(A, B, C_{in})}$$

$$C_{out}(\overline{A}, \overline{B}, \overline{C_{in}}) = \overline{C_{out}(A, B, C_{in})}$$

Possiamo, quindi, negare tutti gli ingressi, spostare le NOT dall'uscita all'ingresso, come in Figura 7, diminuendo così il tempo di propagazione lungo il cammino critico. Inoltre avendo entrambi gli ingressi negati sul secondo Full-Adder, non abbiamo bisogno della NOT in uscita. Complessivamente, si utilizzano tre invertitori ogni due Full-Adder (uno in meno rispetto a prima), ottenendo anche un risparmio nel numero di porte.

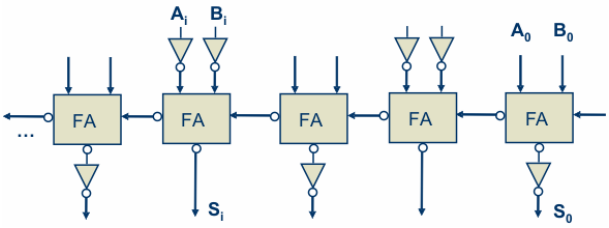

Figura 7: Addizionatore a propagazione basato su FA con uscite negate. Schema migliorato.

3 Sottrattore

Un sottrattore può essere costruito in maniera molto simile ad un addizionatore, ancora una volta osservando la natura iterativa dell'operazione. La Figura 8 mostra un esempio di sottrazione. Anche in questo caso l'operazione è effettuata bit a bit, ma per ogni coppia occorre osservare che i bit del secondo operando, B, sono sottratti (ovvero, moltiplicati per -1 e sommati algebricamente). Inoltre, il valore propagato da una coppia di bit alla successiva non è un riporto, ma un *prestito* (in inglese, *borrow*), ovvero va anch'esso sommato con peso negativo. La sottrazione relativa ad ogni coppia di bit, a sua volta, produce un bit del risultato ed un bit di prestito che sarà considerato dalla coppia di bit immediatamente a sinistra.

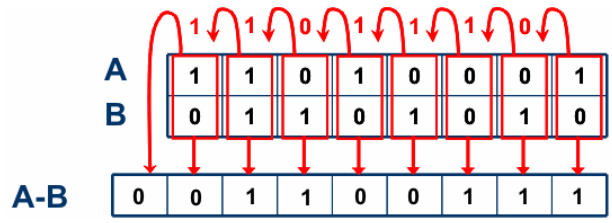

Figura 8: Un esempio di sottrazione tra numeri in rappresentazione binaria.

Anche in questo caso è possibile distinguere due tipi di operazioni: una, più semplice, non prevede prestito entrante ed è effettuata sulla coppia di bit meno significativa, l'altra, che invece prevede un prestito entrante, è effettuata su tutte le altre coppie di bit. La prima operazione è effettuata dal circuito Semi-Sottrattore (Half-Subtractor), la cui tabella di verità è riportata sotto

A_i	B_i	S	C_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

La sottrazione sulle coppie di bit interne, invece, è realizzata dal circuito Sottrattore Completo (Full-Subtractor), la cui tabella di verità è riportata sotto.

A_i	B_i	C_{in}	S	C_{out}
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

Si noti che anche in questo caso, come per i circuiti addizionatori, la coppia C_{out},S codifica la somma degli 1 in ingresso. Questa volta, però, sia gli ingressi che le uscite possono avere pesi negativi. In particolare, l'ingresso A_i ha peso 1, mentre B_i e C_{in} hanno peso -1. In uscita, S ha peso 1, mentre C_{out} ha peso -2. In altri termini, il Sottrattore implementa la funzione:

$$(-2)C_{out} + S = A_i + (-1)B_i + (-1)C_{in}$$

Dalle tabelle di verità è possibile notare che, per il Semi-Sottrattore, si ha:

$$S = A_i \oplus B_i = \overline{(\overline{A_i} \oplus B_i)}$$

$$C_{out} = \overline{A_i} B_i$$

mentre per il Sottrattore Completo si ha:

$$S = A_i \oplus B_i \oplus C_{in} = \overline{(\overline{A_i} \oplus B_i \oplus C_{in})}$$

$$C_{out} = \overline{A_i B_i} C_{in} + \overline{A_i} B_i \overline{C_{in}} + \overline{A_i} B_i C_{in} + A_i B_i C_{in} =$$
$$(\overline{A_i B_i} C_{in} + \overline{A_i} B_i C_{in}) + (\overline{A_i} B_i \overline{C_{in}} + \overline{A_i} B_i C_{in}) + (\overline{A_i} B_i C_{in} + A_i B_i C_{in}) =$$
$$\overline{A_i} C_{in} + \overline{A_i} B_i + B_i C_{in} = \overline{A_i} B_i + C_{in}(\overline{A_i} + B_i)$$

E' chiaro dalle precedenti formule che un Semi-Sottrattore ed un Sottrattore Completo possono essere ottenuti da un Semi-Addizionatore e da un Addizionatore Completo negando l'ingresso A_i e l'uscita S, come riportato in Figura 9.

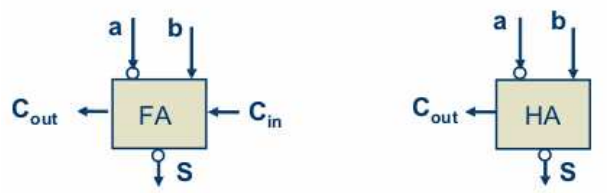

Figura 9: Sottrattori binari realizzati tramite Full-Adder e Half-Adder.

Nella pratica, tuttavia, è preferibile fare uso di circuiti addizionatori senza alcuna modifica circuitale interna. La tecnica più ovvia consiste nello sfruttare le proprietà della rappresentazione in complementi, grazie alla quale è possibile usare solo l'operazione di addizione per manipolare numeri che possono eventualmente essere negativi. La sottrazione consisterà pertanto nel trasformare il numero da sottrarre nel suo opposto. Questa trasformazione, come ben noto, può essere effettuata invertendo tutti i bit del numero ed aggiungendo uno al valore così ottenuto. L'aggiunta di uno, che richiederebbe un secondo addizionatore, può essere fatta in maniera immediata se l'addizionatore è implementato usando un Full-Adder invece che un Half-Adder sulla cifra meno significativa. In tal caso, infatti, è possibile aggiungere 1, visto come riporto entrante sulla cifra di peso più basso. In definitiva, è possibile usare un addizionatore come sottrattore negando i bit del numero da sottrarre ed inserendo un riporto entrante sulla cifra più bassa pari ad 1. Questa tecnica è mostrata nella Figura 10.a. Rendendo controllabile dall'esterno la possibilità di negare o meno i bit dell'operando B e l'inserimento di un riporto entrante, lo stesso componente può essere usato sia come sottrattore che come addizionatore. Nella Figura 10.b è riportato uno schema di addizionatore/sottrattore. Il circuito esegue l'addizione quando $sub = 0$ e la sottrazione quando $sub = 1$.

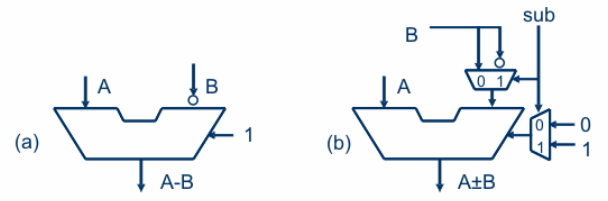

Figura 10: Addizionatore usato come sottrattore.

4 Carry Select Adder

È possibile costruire un addizionatore più veloce del Ripple-Carry introducendo un certo grado di *ridondanza* all'interno del circuito. Possiamo, infatti, dividere l'addizionatore in p blocchi di m bit. Per come abbiamo definito queste grandezze il numero totale di bit dell'addizionatore sarà quindi $n = p \cdot m$. Il primo blocco sarà costituito da un RCA a m bit, mentre i blocchi dal secondo al p-esimo saranno costituiti da due RCA ad m bit, l'uno col segnale $C_{in} = 0$ e l'altro con $C_{in} = 1$. Le uscite di questi blocchi saranno selezionate da un multiplexer che riceverà in ingresso come segnale di selezione il riporto uscente del blocco precedente.

Il parallelismo introdotto all'interno del circuito permette di velocizzare il calcolo. È infatti importante notare che i calcoli eseguiti nei diversi blocchi possono procedere indipendentemente, e completarsi tutti allo stesso istante, pari al ritardo del RCA di m bit che è usato per il singolo blocco. Chiaramente, questo vantaggio comporta un prezzo da pagare in termini di porte logiche usate: il numero di componenti è circa doppio rispetto all'RCA.

Un addizionatore del genere prende il nome di *Carry Select Adder* ed è mostrato in Figura 11, con $n = 16$ ed $m = 4$; in rosso è evidenziato il cammino critico.

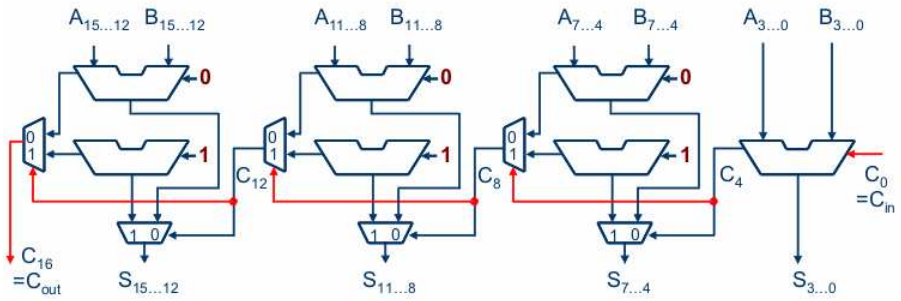

Figura 11: Addizionatore Carry Select.

Analizziamone in dettaglio il ritardo. Supponendo che i blocchi elementari siano degli RCA, i singoli blocchi avranno effettuato le loro somme in un tempo $T_{blocco} = mT_{FA}$ (vedi equazione (3)). I primi m bit della somma saranno quindi disponibili dopo il tempo T_{blocco}. I bit del secondo blocco saranno disponibili dopo un tempo $T_{blocco} + T_{mux}$ (dove T_{mux} è il ritardo del multiplexer[1]), quelli del terzo dopo un tempo $T_{blocco} + 2T_{mux}$ e così via. Il tempo totale di computazione del circuito è quindi:

$$T_{CSA} = mT_{FA} + (p-1)T_{mux} \tag{5}$$

Scegliendo adeguatamente m e p, è possibile rendere T_{CSA} minimo. Nella (5) sostituiamo $m = n/p$ e deriviamo rispetto a p:

$$\frac{dT_{CSA}}{dp} = -\frac{nT_{FA}}{p^2} + T_{mux}$$

[1] il tempo di propagazione del multiplexer è costante rispetto a n

Studiando il segno della derivata, è facile verificare che il minimo di T_{CSA} si ha in corrispondenza di

$$p_{min} = \sqrt{\frac{nT_{FA}}{T_{mux}}} \Rightarrow m_{min} = \frac{n}{p_{min}} = \sqrt{\frac{nT_{mux}}{T_{FA}}} \tag{6}$$

Le (6) indicano i valori ottimali di p ed m. Sostituendo questi valori nella (5) otteniamo il tempo di propagazione minimo di un Carry Select:

$$\begin{aligned} T_{CSAmin} &= m_{min}T_{FA} + (p_{min} - 1)T_{mux} \\ &= \sqrt{nT_{FA}T_{mux}} - T_{mux} + \sqrt{nT_{FA}T_{mux}} \\ &= 2\sqrt{nT_{FA}T_{mux}} - T_{mux} \end{aligned} \tag{7}$$

Il ritardo è dunque proporzionale alla radice di n.

4.1 Ottimizzazione del Carry Select Adder

In un Carry Select Adder come definito nel paragrafo precedente, le somme calcolate dai vari blocchi RCA sono disponibili tutte allo stesso tempo $T = mT_{FA}$, mentre i segnali di selezione dei blocchi dal secondo al p-esimo sono disponibili solamente dopo un tempo $T_{CSA} = mT_{FA}+qT_{mux}$, con $q \in [1, (p-1)]$, perché occorre aspettare che il riporto si propaghi da uno stadio all'altro.

Fatta questa osservazione possiamo migliorare il Carry Select Adder eliminando il vincolo che tutti i blocchi siano della stessa dimensione m. L'idea è quella di assegnare un carico computazionale maggiore ai blocchi il cui risultato è richiesto dopo un tempo maggiore.

Facciamo un esempio per chiarire questo concetto. Supponiamo per semplificare i calcoli che $T_{mux} = T_{FA} = 1$. Il segnale C_8 (si veda la Figura 11) è disponibile al tempo 5, mentre gli altri segnali in ingresso ai due multiplexer che esso governa sono già disponibili al tempo 4. Possiamo quindi aumentare la dimensione del terzo blocco da 4 a 5 senza peggiorare le prestazioni.

In generale ogni stadio avrà il segnale di riporto entrante disponibile dopo un ritardo T_{mux} rispetto allo stadio che lo precede. Quindi, se si rimane nell'ipotesi che $T_{mux} = T_{FA}$ le lunghezze ottimali dei blocchi sono

$$m,\ m,\ m+1,\ m+2,\ m+3,\ \ldots$$

o più in generale

$$m,\ m,\ m+q(1),\ m+q(2),\ldots\ m+q(i),\ \ldots$$

con $q(i)$ pari all'approssimazione intera di $i \cdot T_{mux}/T_{FA}$

Se costruiamo il circuito in questo modo il percorso critico rimane lo stesso, ma possiamo ridurre la dimensione del primo blocco (che è l'unico sul percorso critico, come risulta evidente in Figura 11) e, per n sufficientemente grande, possiamo ridurre il numero di livelli.

Per esempio per $n = 64$, con $T_{FA} = T_{mux} = 1$, possiamo costruire gli stadi secondo la sequenza:

$$7\ \ 7\ \ 8\ \ 9\ \ 10\ \ 11\ \ 12$$

Questo circuito ha 7 stadi, di cui il primo a 7 bit. Il ritardo è quindi (vedi equazione(5))

$$T = 7T_{FA} + 6T_{mux} = 13$$

n	2	4	8	16	32	64	128	256
T	4	6	8	10	12	14	16	18
m	1	2	3	4	5	6	7	8

Tabella 3: Complessità di un Conditional Sum Adder.

Se avessimo costrutito il circuito con blocchi di uguale dimensione, per le (6) avremmo avuto

$$p = m = 8$$

ed il ritardo sarebbe stato:

$$T = 8T_{FA} + 7T_{mux} = 15$$

5 Conditional Sum Adder

Il Conditional Sum Adder può essere visto come un evoluzione del Carry Select. In effetti, come quest'ultimo sfrutta la ridondanza per calcolare tutti i possibili risultati, e dei multiplexer per selezionare di volta in volta il risultato corretto. La differenza sta nel modo in cui disponiamo i multiplexer, che in questo caso sono strutturati ad albero per ottimizzare la velocità.

In Figura 12 è mostrato come esempio un Conditional Sum Adder a 4 bit. I segnali rossi sono relativi ai full adder con riporto entrante posto a 1, quelli blu a quelli con riporto posto a 0. Dopo ogni multiplexer è segnato il ritardo. In Tabella 3 sono mostrati dati relativi alla complessità del Conditional Sum

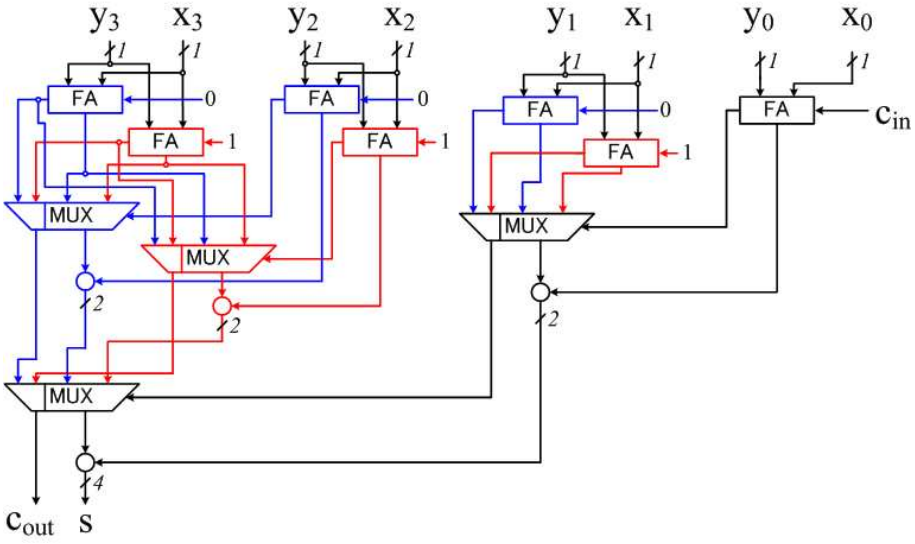

Figura 12: Conditional Sum Adder

Adder al variare della dimesione degli ingressi. In particolare T rappresenta il ritardo, considerando $T_{FA} = T_{MUX} = 1$, dove T_{FA} e T_{MUX} sono i ritardi,

A_i	B_i	C_i	S	C_{i+1}	P_i	G_i	K_i
0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	1
0	1	0	1	0	1	0	0
0	1	1	0	1	1	0	0
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	0
1	1	0	0	1	0	1	0
1	1	1	1	1	0	1	0

Tabella 4: Tabella di verità di un Full-Adder, con i relativi segnali di *propagate*, *generate* e *kill*.

rispettivamente, del Full Adder e del multiplexer, mentre m è il numero di livelli di cui è costituito il circuito. In generale il ritardo per un Conditional Sum Adder ad n bit è dato da

$$2\log(n) + 2$$

risultando molto più veloce del CSA. Il problema maggiore del Conditional Sum Adder, d'altro canto, è la notevole ridondanza, che comporta un costo in termini di porte logiche estremamente alto.

Come verrà mostrato in seguito, la struttura ad albero è una caratteristica comune a tutti gli addizionatori veloci.

6 Generazione e Propagazione del Riporto

Il riporto entrante in un Full-Adder ha effetto sul riporto uscente soltanto se i due bit in ingresso sono diversi. Infatti (come si può evincere dalla Tabella 4) se $A_i = B_i = 0$, avremo riporto in uscita $C_{i+1} = 0$, indipendentemente dal riporto entrante C_i; in questo caso si dice che il riporto è stato "eliminato" (killed) allo stadio i. Viceversa, se $A_i = B_i = 1$, avremo $C_{i+1} = 1$ indipendentemente da C_i; in questo caso si dice che il riporto è stato "generato" nello stadio i. Infine se $A_i \neq B_i$ (ovvero, $A_i = 0$ e $B_i = 1$ oppure $A_i = 1$ e $B_i = 0$), avremo $C_i = C_{i+1}$, in questo caso si dice che il riporto si è "propagato" dallo stadio i allo stadio $i + 1$.

A valle di queste considerazioni introduciamo i segnali di *propagazione*, *generazione* ed *eliminazione* del riporto, indicati rispettivamente con P_i, G_i e K_i e definiti come segue:

$$P_i = A_i \oplus B_i \tag{8}$$

$$G_i = A_i \cdot B_i \tag{9}$$

$$K_i = \overline{A_i} \cdot \overline{B_i} = \overline{A_i + B_i} \tag{10}$$

Come si può facilmente evincere dalle considerazioni precedenti, possiamo calcolare il riporto uscente in funzione di P_i e G_i

$$C_{i+1} = C_i \cdot P_i + G_i \tag{11}$$

Immaginando di effettuare la somma su più bit, per ciascuno dei quali sia calcolata la coppia di segnali P_i G_i, ed utilizzando ricorsivamente la 11, otteniamo

$$\begin{aligned} C_1 &= G_0 + P_0 C_{in} \\ C_2 &= G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in} \\ C_3 &= G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in} \\ C_4 &= G_3 + P_3 C_2 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in} \\ &\dots \end{aligned} \tag{12}$$

Considerando ad esempio C_3 (riporto in ingresso alla terza cifra), possiamo riscriverne l'espressione come:

$$C_3 = BG_{2-0} + BP_{2-0} C_{in} \tag{13}$$

avendo introdotto i segnali di *Block Propagate* e *Block Generate*, definiti rispettivamente come:

$$\begin{aligned} BG_{2-0} &= G_2 + P_2 G_1 + P_2 P_1 G_0 \\ BP_{2-0} &= P_2 P_1 P_0 \end{aligned} \tag{14}$$

Questa equazione ha una forma del tutto simile alla (11), in cui invece di prendere in considerazione un singolo Full-Adder, consideriamo un "blocco". Volendo dare un significato intuitivo all'equazione (13), possiamo interpretarla nel modo seguente:

> Si ha un riporto uscente dal "blocco" di cifre $0 \dots 2$, se viene generato internamente al blocco un riporto ($BG_{2-0} = 1$), oppure vi è un riporto entrante che viene propagato fino all'uscita ($C_{in} = 1$ e $BP_{2-0} = 1$).

In generale, se $BP_{j-i} = 1$ il riporto C_{j+1} uscente dal blocco di cifre $i \dots j$ coincide con il riporto entrante nel blocco, e in questo caso non è necessario attendere che la somma venga calcolata per conoscere il riporto. Questa osservazione è alla base del funzionamento di molti addizionatori, tra cui il Carry-Skip, descritto nel prossimo paragrafo.

7 Carry-Skip Adder

L'adder Carry-Skip è formato, come il Carry Select Adder, da p blocchi di m bit, con $n = p \cdot m$. Ogni blocco è composto da un Ripple Carry ad m bit ed un multiplexer che riporta in uscita il riporto generato nel blocco se $BP = 0$, o il riporto entrante, se $BP = 1$. Un circuito del genere è mostrato in Figura 13. Come si può notare dalla figura, questo circuito è notevolmente più compatto del Carry-Select Adder, in quanto, non introducendo ridondanza nel calcolo, necessita di un numero di blocchi molto inferiore. Il segnale BP, definito secondo la (14), è ottenibile come una AND di tutti i segnali $P_i = A_i \oplus B_i$ contenuti all'interno del blocco. Considerando che il segnale P_i è comunque prodotto all'intero dei Full-Adder, la realizzazione del segnale BP richiede soltanto l'aggiunta di una porta AND ad m ingressi, come mostrato in Figura 14. Il segnale BP viene in generale calcolato più velocemente rispetto alla somma, in quanto generato da una rete più semplice.

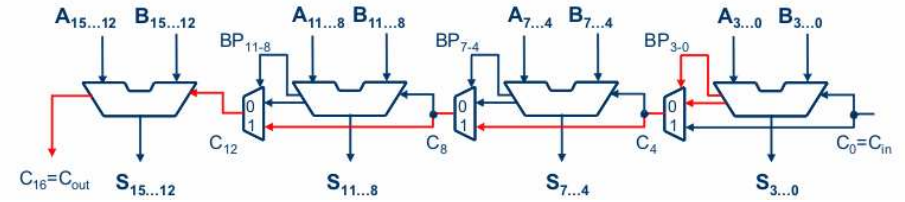

Figura 13: Carry-Skip Adder con $n = 16$ e $m = 4$

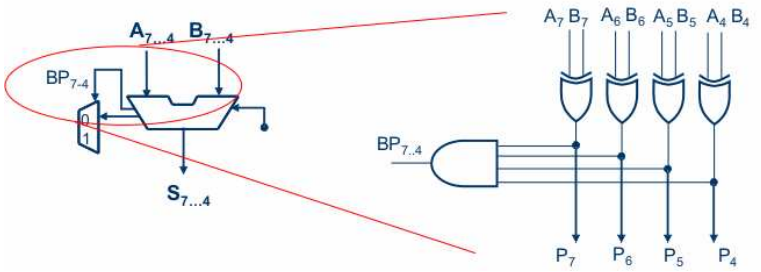

Figura 14: Generazione del segnale BP nel Carry-Skip Adder

Per calcolare il ritardo di questo addizionatore, prendiamo come esempio quello di Figura 13. Consideriamo il valore di S_{15}. Questo è funzione del riporto C_{12}. Il riporto C_{12} è a sua volta funzione di C_8 se $BP_{11-8} = 1$, mentre dipende dal risultato della somma calcolata nel terzo blocco se $BP_{11-8} = 0$. A sua volta il riporto C_8 è funzione del riporto C_4 se $BP_{7-4} = 1$, della somma calcolata nel secondo blocco altrimenti. Proseguendo con questo ragionamento è facile rendersi conto che il caso peggiore nel Carry-Skip Adder si ha quando il riporto viene generato nel primo blocco e propagato fino all'ultimo. Nell'esempio, il cammino critico attraversa il primo RCA, 3 multiplexer e l'ultimo RCA (cammino in rosso in figura). Il ritardo massimo vale quindi:

$$T_{skip} = 4T_{FA} + 3T_{mux} + 4T_{FA} \tag{15}$$

e più in generale:

$$T_{skip} = 2mT_{FA} + (p-1)T_{mux} \tag{16}$$

Confrontando la (16) con la (5), notiamo che questo addizionatore è sensibilmente più lento del Carry-Select. Paghiamo quindi in prestazioni il risparmio d'area.

Calcoliamo ora la scelta migliore di m e p, dato n:

$$\begin{aligned} \frac{dT_{skip}}{dp} = -\frac{2nT_{FA}}{p^2} + T_{mux} = 0 \quad &\Rightarrow \quad p_{min} = \sqrt{\frac{2nT_{FA}}{T_{mux}}} \\ &\Rightarrow \quad m_{min} = \frac{n}{p_{min}} = \sqrt{\frac{nT_{mux}}{2T_{FA}}} \end{aligned} \tag{17}$$

Le (17) indicano i valori ottimali di p ed m. Sostituendo questi valori nella (16) otteniamo il tempo di propagazione minimo di un Carry-Skip:

$$T_{skip-min} \quad = \quad 2m_{min}T_{FA} + (p_{min} - 1)T_{mux}$$

$$= \sqrt{2nT_{FA}T_{mux}} - T_{mux} + \sqrt{2nT_{FA}T_{mux}}$$
$$= 2\sqrt{2nT_{FA}T_{mux}} - T_{mux} \quad (18)$$

Anche in questo caso abbiamo un ritardo proporzionale alla radice di n.

7.1 Ottimizzazione del Carry-Skip

Come abbiamo fatto per il Carry Select, possiamo migliorare le prestazioni dell'addizionatore Carry-Skip eliminando il vincolo che i blocchi siano tutti della stessa dimensione m.

In Figura 15 è rappresentato un Carry-Skip con $m = 4$ e $n = 16$, con i ritardi dei vari segnali racchiusi in un cerchio. Si è supposto per semplicità $T_{FA} = T_{mux} = 1$. Il primo e l'ultimo blocco si trovano sul cammino critico,

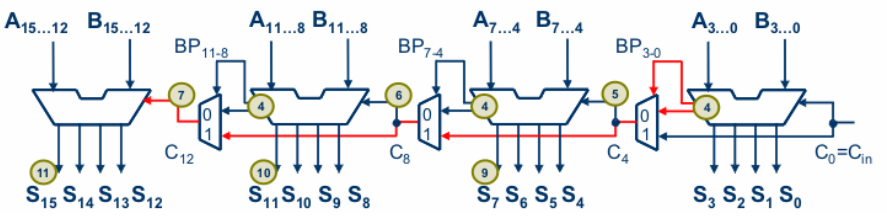

Figura 15: Valutazione dei ritardi nel Carry Skip.

per cui non possiamo aumentarne le dimensioni senza deteriorare le prestazioni complessive. Il multiplexer del secondo blocco ha a disposizione il riporto C_4 al tempo 5, e il riporto generato dal secondo RCA al tempo 4[2]. Potremmo quindi aggiungere un bit al secondo addizionatore senza peggiorare le prestazioni. Il multiplexer del terzo blocco ha a disposizione il riporto C_8 al tempo 6, e il riporto generato dal terzo RCA al tempo 4. Sembrerebbe che si possa aumentare il numero di bit di 2 in tal caso. Così facendo, però, il segnale S_{11}, che è normalmente disponibile al tempo 10, diverrebbe disponibile al tempo 12, superando il tempo di propagazione del cammino critico, pari ad 11. Pertanto, in questo caso possiamo aumentare la dimensione del terzo blocco di un solo bit.

In generale si può verificare che è possibile aumentare le dimensioni dei blocchi dal secondo al penultimo e che i blocchi di dimensione maggiore sono quelli in posizioni centrali. Strutturando lo schema in questa maniera, possiamo sperare di diminuire il numero di blocchi e/o il numero di bit nei blocchi sul percorso critico (il primo e l'ultimo). Per esempio, per $n = 128$ e $T_{FA} = T_{mux} = 1$, progettando con i blocchi a dimensione fissa, abbiamo per le (17) $p = 16$ e $m = 8$ e quindi, sostituendo nella (16)

$$T = 2 \cdot 8 + 15 = 31$$

mentre se utilizziamo blocchi di dimensioni rispettivamente:

$$4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 12,\ 11,\ 10,\ 9,\ 8,\ 7,\ 6,\ 5$$

[2]Si noti che il riporto uscente dall'RCA richiede più di 4 unità di tempo quando dipende dal riporto entrante. Questo caso, però, è proprio quello in cui $BP = 1$, ovvero quello nel quale non ci interessa il valore del riporto dell'RCA poiché il multiplexer selezionerà direttamente il riporto entrante nel blocco.

il ritardo comlpessivo sarà

$$T = 4T_{FA} + 15T_{mux} + 5T_{FA} = 24$$

.

In generale, per ottenere un ritardo ottimo è possibile seguire la distribuzione nel grafico di Figura 16. Le due rette hanno una pendenza pari a T_{mux}/T_A, ed il massimo si ha ad $p/2$.

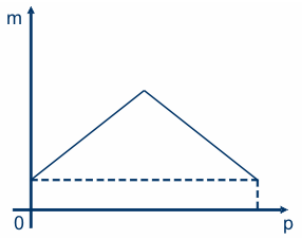

Figura 16: Distribuzione delle lunghezze dei blocchi all'interno di un Carry-Skip. La pendenza delle rette è T_{mux}/T_A

8 Carry Look-Ahead

Le (12) ci permettono di considerare il riporto k-esimo indipendente dai riporti $C_{k-1},\ C_{k-2}, \ldots,\ C_1$, ma dipendente esclusivamente da $C_0 = C_{in}$ e dai segnali P e G. Possiamo dunque ottimizzare il calcolo del riporto k-esimo e valutare il riporto uscente da ogni blocco prim'ancora di conoscere i riporti interni e le uscite somma. L'addizionatore Carry Look-Ahead (Figura 17), si basa su questa osservazione ed è composto da 3 blocchi fondamentali:

- Il primo blocco (blocco PG), è dedicato al calcolo dei segnali P_i e G_i, ed è costituito da k Half-Adder (le equazioni (1) e (8), così come la (2) e la (9) sono identiche).
- Il secondo blocco (blocco CLA), è invece dedicato al calcolo veloce dei riporti C_i utilizzando le 12, e rappresenta il blocco critico per le ragioni spiegate nel successivo Paragrafo 8.1).
- Il terzo ed ultimo blocco (blocco S) si occupa del calcolo degli S_i, e si realizza con k porte XOR, in quanto l'i-esimo bit della somma può essere espresso come

$$S_i = A_i \oplus B_i \oplus C_i = P_i \oplus Ci$$

8.1 Il Blocco CLA

Dalle equazioni (12) possiamo dedurre che è possibile calcolare i riporti parallelamente con 2 livelli logici. Il blocco CLA è però limitato dalla complessità delle porte logiche necessarie per realizzarlo. Infatti, sempre osservando le (12), possiamo renderci conto che, in generale, per calcolare il riporto i-esimo abbiamo bisogno di porte logiche con un numero di ingressi pari a $i + 1$.

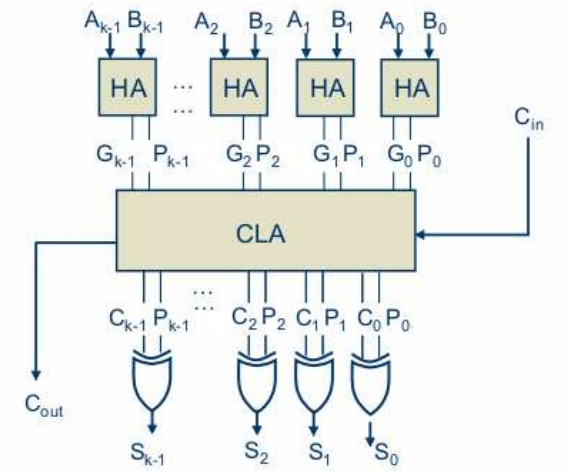

Figura 17: Carry Look-Ahead adder

In molte tecnologie il ritardo delle porte logiche cresce significativamente con il numero di ingressi, rendendo poco conveniente realizzare CLA molto larghi. Una dimensione ragionevole, spesso adottata nella pratica, è 4, che richiede porte con numero di ingressi massimo pari a 5[3].

Per estendere questo approccio per $n > 4$ utilizziamo i segnali di *Block Propagate* e *Block Generate*, di cui riportiamo qui le definizioni per blocchi di 4 bit, indicando per semplicità $BG_{j-i} = BG_i$ e $BP_{j-i} = BP_i$:

$$\begin{aligned} BG_i = \quad & G_{4i+3} + G_{4i+2}P_{4i+3} + G_{4i+1}P_{4i+2}P_{4i+3} \\ & + G_{4i}P_{4i+1}P_{4i+2}P_{4i+3} \\ BP_i = \quad & P_{4i}P_{4i+1}P_{4i+2}P_{4i+3} \end{aligned} \tag{19}$$

BG_i alto indicherà che si è verificata una generazione di riporto tra le cifre $4i$ e $4i+3$, mentre BP_i alto indicherà che un eventuale riporto entrante nella cifra $4i$ si propagherà identico oltre la cifra $4i+3$. Possiamo quindi scrivere, confrontando le (19) con le (12):

$$\begin{aligned} C_4 \quad & = BG_0 + BP_0 \cdot C_0 = BG_0 + BP_0 \cdot C_{in} \\ C_8 \quad & = BG_1 + BP_1 \cdot C_4 \\ C_{12} \quad & = BG_2 + BP_2 \cdot C_8 \\ & \vdots \\ C_n \quad & = BG_{\frac{n}{4}-1} + BP_{\frac{n}{4}-1} \cdot C_{n-4} \end{aligned} \tag{20}$$

Le (20) possono essere sviluppate ricorsivamente come per la (11).

Per $n = 16$ abbiamo ad esempio:

$$\begin{aligned} C_{16} \quad & = \quad BG_3 + BP_3 \cdot C_{12} = \ldots \\ & = \quad [BG_3 + BP_3 \cdot BG_2 + BP_3 \cdot BP_2 \cdot BG_1 + BP_3 \cdot BP_2 \cdot BP_1 \cdot BG_0] \\ & \quad + [BP_3 \cdot BP_2 \cdot BP_1 \cdot BP_0] \cdot C_{in} \end{aligned} \tag{21}$$

[3]Adottando la strategia di Ling, basata su una definizione leggermente diversa di riporto, par.8.3, è possibile limitare il numero di ingressi a 4.

E' importante notare la seguente proprietà: la (21), applicata ai segnali di generate/ propagate a livello di blocco, è identica alla (12), applicata ai segnali generate/ propagate a livello di bit. Di conseguenza, la (21) può essere realizzata con un circuito identico a quello che implementa la (12): un CLA. Un addizionatore Carry Look-Ahead a 16 bit può quindi essere costruito secondo lo schema ad albero di Figura 18.

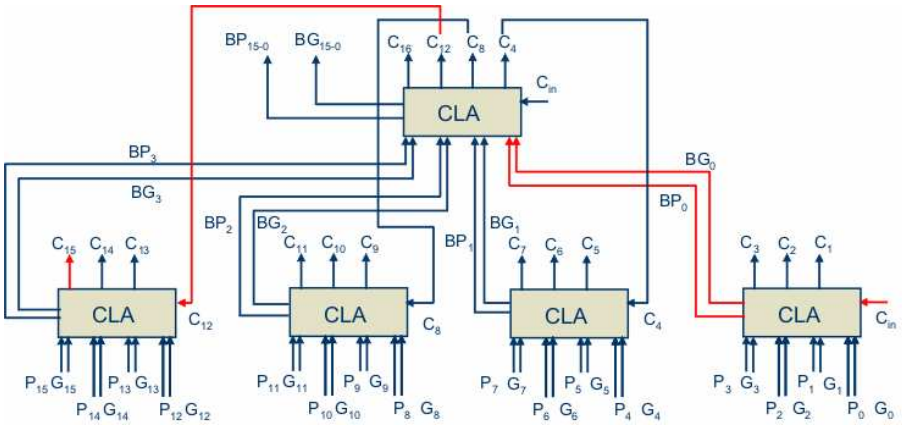

Figura 18: Carry Look-Ahead a 16 bit. In rosso è evidenziato il percorso critico

L'idea può essere applicata ricorsivamente. Il CLA di livello più alto in Figura 18 produce infatti dei segnali di generate/propagate relativi a tutti i 16 bit in ingresso. Avendo ad esempio 64 bit, sarebbe possibile dividere i 64 bit in quattro gruppi di 16 bit, realizzare un CLA come quello di Figura 18 per ciascuno dei 4 gruppi di cifre, ottenendo per ognuno una coppia generate/propagate, ed usare un ulteriore CLA di terzo livello per accorpare le informazioni di generate/propagate che vengono dai quattro gruppi di 16 bit. Usando CLA di 4 bit, avremo così realizzato un circuito anticipatore del riporto a 64 bit con tre livelli di CLA (infatti, $4^3 = 64$). In generale, avendo CLA a m bit strutturati in un albero di l livelli, si otterrà un anticipatore di riporto capace di lavorare su $n = m^l$ bit. In altre parole, il numero di livelli (attraverso cui si propaga l'informazione sul riporto, responsabile del peggiore ritardo nel circuito) è *logaritmico* rispetto al numero di bit n dell'addizionatore.

8.2 Stima dei Ritardi

In generale (vedi Figura 17) il ritardo di un addizionatore Carry Look-Ahead è dato da

$$T = T_{PG} + T_{CLA_{tot}} + T_{XOR} \tag{22}$$

I tempi di propagazione T_{PG} e T_{XOR} sono costanti rispetto ad n. Per quanto riguarda il blocco CLA, possiamo valutare il ritardo per $n = 16$, osservando la Figura 18. Chiamando T_{CLA_4} il ritardo del CLA a 4 bit, abbiamo che

$$T = T_{PG} + 3T_{CLA_4} + T_{XOR}$$

Aumentando ulteriormente il parallelismo, aumenta anche l'altezza h dell'albero, che in generale vale:

$$h = \log_4 n \tag{23}$$

Il percorso critico attraversa l'albero 2 volte, prima dalle foglie alla radice per propagare l'informazione su generazione/propagazione nei vari sottoalberi, e poi viceversa dalla radice ai singoli nodi per calcolare i riporti uscenti da ogni blocco. Abbiamo quindi che:

$$T_{CLA_{tot}} = 2\log_4 n - 1 = \log_2(n) - 1 \tag{24}$$

da cui, sostituendo la (24) nella (22):

$$T = T_{PG} + (\log_2 n - 1)T_{CLA_4} + T_{XOR} \tag{25}$$

Il Carry Look-Ahead presenta come il Conditional Sum Adder una struttura ad albero e, come questo, presenta un ritardo logaritmico. Tuttavia, la quantità di porte logiche è molto più contenuta, facendo del Carry Look-Ahead una delle migliori soluzioni per l'implementazione dell'addizione. L'unico difetto di questo tipo di addizionatore è il layout estremamente irregolare dell'albero di CLA, che richiede collegamenti piuttosto complessi tra le diverse parti della struttura.

8.3 Carry Look-Ahead di Ling

Ling propone una variante del Carry-Look Ahead, che permette di realizzare un blocco CLA leggermente più efficiente a scapito di una maggiore complessità del blocco dedicato alla somma.

Si noti che, ai fini della determinazione del riporto in uscita $C_{i+1} = G_i + C_i P_i$, è indifferente definire il segnale di propagazione P_i come $A_i \oplus B_i$ oppure $A_i + B_i$ (vedi Paragrafo 1). Nella struttura del Carry Look-Ahead, pertanto, il calcolo veloce del riporto può essere basato sulla definizione di P_i che usa la OR ($A_i + B_i$) invece che la più lenta XOR. Usando tale definizione, è immediato verificare la relazione $G_i P_i = G_i$. In effetti, si ha $G_i P_i = A_i B_i (A_i + B_i) = A_i B_i + A_i B_i = A_i B_i = G_i$. L'espressione nella (11) può essere quindi riscritta nel modo seguente:

$$\begin{aligned} C_4 &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in} \\ &= G_3 P_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in} \\ &= P_3(G_3 + G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}) \\ &= P_3 H_3 \end{aligned} \tag{26}$$

L'idea di Ling è quella di calcolare gli pseudo-riporti H_i, invece dei riporti. Questi ultimi possono essere calcolati con una AND fra H_i e P_i. Il vantaggio è che possiamo utilizzare porte con un numero di ingressi inferiore, aumentando le prestazioni del blocco CLA. Questa tecnica comporta però, come precedentemente anticipato, una complicazione del blocco dedicato alla somma. Per ulteriori dettagli si rimada a [4].

9 Altri tipi di addizionatori

La panoramica fornita in precedenza non esaurisce tutti gli approcci all'implementazione circuitale dell'addizione. In particolare, nell'ambito del corso di Reti Logiche non è possibile trattare, per brevità, i cosiddetti *addizionatori a prefissi* [17, 1, 5, 6, 8, 9, 10, 11]. Tutti si basano sull'idea di valutare le condizioni di

generate/propagate per sottoinsiemi delle coppie di bit in ingresso, combinando queste informazioni in uno schema ad albero in modo da migliorare (in maniera diversa a seconda del tipo di addizionatore) numero di porte logiche, ritardo massimo, complessità delle interconnessioni tra le porte.

Bibliografia

[1] B. Parhami. *Computer Arithmetic, Algorithms and Hardware design.* Oxford University Press, 2000.

[2] J.-P. Deschamps, G. J. Antonie Bioul, G. D. Sutter. *Syntesis Of Arithmetic Circuits.* Wiley, 2006.

[3] U. De Carlini, B. Fadini. *Macchine per l'elaborazione delle informazioni* Liguori editore, 1995.

[4] H. Ling. *High-Speed Binary adder* IBM J. RES DEVELOP, 1981.

[5] D. Harris. "A Taxonomy of Parallel Prefix Network", *IEEE Transactions on Computers*, 2003.

[6] A. Beaumont-Smith, C.-C. Lim. "Parallel Prefix Adder Design", 15^{th} *IEEE Symposium on Computer Arithmetic*, 2001.

[7] N.M. Martin and S.P. Hufnagel, "Conditional-Sum Early Completion Adder Logic", *IEEE Transactions on Computers*, vol. C-29, no. 8, 1980.

[8] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations", , *IEEE Transactions on Computers*, vol. C-22, 1973, pp. 786-793.

[9] R.E. Ladner and M.J. Fischer, "Parallel Prefix Computation", *JACM*, vol. 27-4, pp. 831-838, 1980.

[10] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders", *IEEE Transactions on Computers*, vol. C-31, pp. 260-264, 1982.

[11] T. Han and D. A. Carlson, "Fast Area-Efficient VLSI Adders", *Proc. 8th IEEE Symposium on Computer Arithmetic*, pp. 49-56, 1987.

[12] S. Knowles, "A Family of Adders", *Proc. 15th IEEE Symposium on Computer Arithmetic*, 2001, pp 277-281.

[13] J. Slansky, "Conditional-sum addition logic", *IRE Trans. Electronic Computer*, vol. EC-9 pp 226-231, 1960

[14] I. Koren *Computer Arithmetic Algorithms*, Prentice Hall, 1993

[15] R.Beigel, B. Gasarch, M. Li, L. Zhang. "Addition in $\log_2(n) + O(1)$ Steps On Average: A Simple Analysis", *Theoretical Computer Science*, vol. 191, 1998.

[16] R. Zimmermann, *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*. Diss. ETH No. 12480. 1997.

[17] Y.Choi. *Parallel Prefix Adder Design*. 2004.

[18] L.G. Johnson, *Logic Synthesis and Simulation*. 2003.