Capitolo 3

Architetture di moltiplicatori

3.0 Introduzione

La moltiplicazione è un'altra operazione molto comune nell'elaborazione digitale del segnale, necessaria ad esempio per effettuare correlazioni, convoluzioni, filtraggio. Anche in questo caso si tratta di moltiplicazione di operandi espressi in virgola fissa.
La moltiplicazione può essere vista come una serie di addizioni ripetute: il numero che deve essere sommato è il moltiplicando, mentre il numero somme che devono essere effettuate è legato al valore del moltiplicatore: il risultato dell'operazione è il prodotto tra i due fattori. Ad ogni passo di elaborazione viene generato un prodotto parziale.
Questo metodo di procedere, attraverso somme parziali, si rivela spesso troppo lento: è per questa ragione che, accanto a strutture di moltiplicatori che utilizzano algoritmi classici per la computazione del prodotto, sono stati ideati altri algoritmi che tendono a ridurre il numero di somme parziali che devono essere computate (algoritmi di Booth), e strutture che tendono a velocizzare l'operazione di somma.

3.1 Moltiplicatore parallelo con architettura *Ripple*

Un modo semplice per realizzare un moltiplicatore parallelo è quello di riprodurre, attraverso un circuito, l'algoritmo con in quale usualmente si opera la moltiplicazione "a mano" (Figura 3.1 .1).
Un esempio di moltiplicatore parallelo per numeri espressi in modulo che segue questa strategia è illustrato in Figura 3.1 .2.

$a_3\,a_2\,a_1\,a_0 \times b_3\,b_2\,b_1\,b_0 = p_7\,p_6\,p_5\,p_4\,p_3\,p_2\,p_1\,p_0$ $\qquad$ 1 0 1 1 ×0 1 1 1

				$a_3a_2a_1a_0$ x			
				$b_3b_2b_1b_0$			
				a_3b_0	a_2b_0	a_1b_0	a_0b_0
			a_3b_1	a_2b_1	a_1b_1	a_0b_1	
		a_3b_2	a_2b_2	a_1b_2	a_0b_2		
	a_3b_3	a_2b_3	a_1b_3	a_0b_3			
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

				1	0	1	1
				0	1	1	1
				1	0	1	1
			1	0	1	1	
		1	0	1	1		
	0	0	0	0			
0	1	0	0	1	1	0	1

Figura 3.1.1. Algoritmo elementare di moltiplicazione: esempio di moltiplicazione tra fattori caratterizzati da lunghezza di parola pari a 4 espressi in modulo.

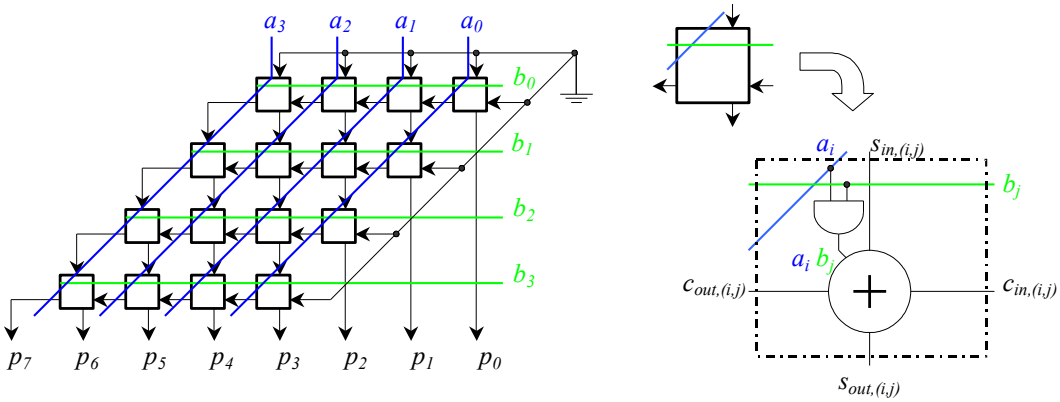

Figura 3.1.2. Architettura di un moltiplicatore parallelo con Ripple Carry Adders: esempio di moltiplicazione 4x4 e cella elementare.

Il risultato della moltiplicazione di una parola di lunghezza pari a N bit per una parola di lunghezza pari a M bit è un parola di $N+M$ bit, quindi sono necessarie per questa realizzazione $N \times M$ celle elementari: un problema rappresentato da questa struttura è l'area occupata.
Per quanto riguarda la velocità, si può notare che tutti i prodotti parziali ($a_i b_j$) vengono eseguiti in parallelo. Il limite temporale nell'elaborazione è determinato dai sommatori (Figura 3.1 .3):

$$t_{MAX} = t_{AND} + (M\text{-}1)\, t_{cout} + (M\text{-}1)\, t_s + (N\text{-}1)\, t_{cout} + t_s = t_{AND} + (2M+N\text{-}2)\, t_{full\ adder}$$

È conveniente avere un sommatore con eguali (e minimi) tempi di elaborazione di *carry* e somma, dato che sul percorso critico si vengono a trovare elaborazioni sia del bit di somma sia del *carry*.
In genere la profondità di parola dei due fattori è la medesima, che si suppone nel resto della trattazione uguale a N.

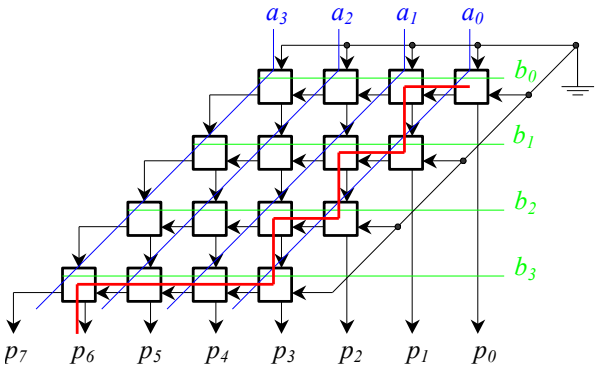

Figura 3.1.3. Moltiplicatore parallelo con sommatori Ripple Carry: ritardo introdotto.

3.2 *Moltiplicatore Parallelo con architettura Carry Save (Braun Parallel Multiplier)*

È possibile adottare alcuni accorgimenti che migliorano le caratteristiche della struttura presentata nel paragrafo precedente. I sommatori possono essere ad esempio organizzati secondo l'architettura *Carry Save* invece che secondo l'architettura *Ripple Carry* (Figura 3.2 .4).

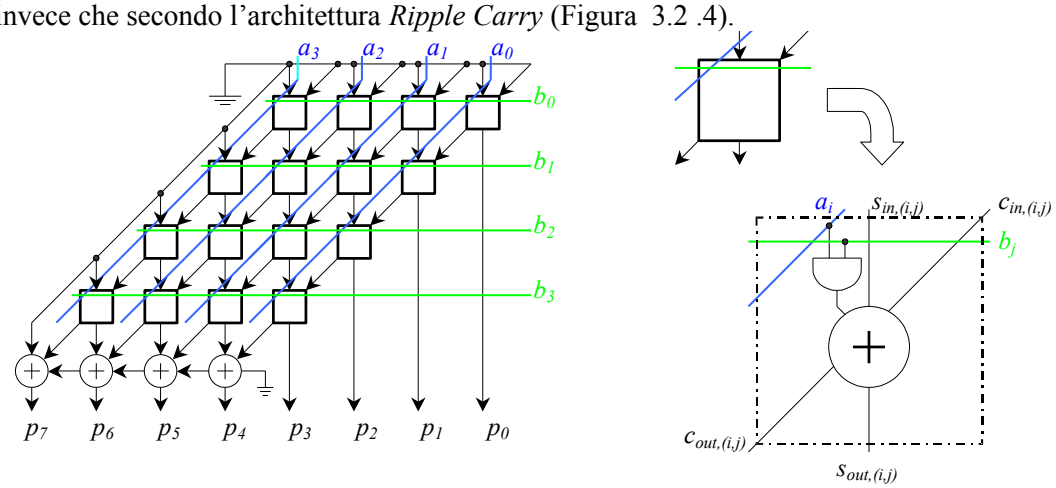

Figura 3.2.4. Architettura di un Braun Parallel Multiplier: esempio 4x4.

Anche in questo caso tutti i prodotti parziali vengono eseguiti in parallelo.
L'architettura con la quale vengono organizzati i sommatori permette di eseguire la moltiplicazione in un tempo inferiore rispetto al moltiplicatore parallelo con architettura Ripple descritto nel Par. 3.1 (si veda Figura 3.2 .5):

$$t_{MAX} = t_{\text{AND}} + (N+M)\, t_{full\ adder}$$

Si noti in particolare che l'ultima riga di sommatori è ancora organizzata in modo *Ripple Carry*: anche in questo caso è possibile ottimizzare ulteriormente questa parte rendendo più veloce il moltiplicatore.

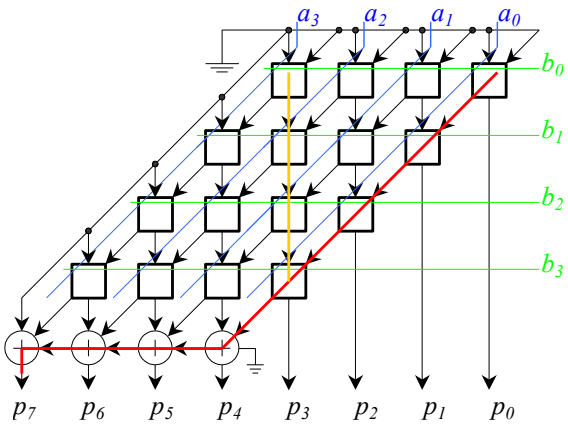

Figura 3.2.5. Moltiplicatore parallelo: discussione del ritardo introdotto.

È possibile ottimizzare la struttura notando che le celle della prima e seconda riga possono essere semplificate rispettivamente in una riga di AND e di *half adders*. Analoghe modifiche sono possibili sulle celle alla sinistra della struttura e, mantenendo la struttura *Ripple Carry Adder*, per il sommatore dell'ultima riga (Figura 3.2 .6).

Un semplice riarrangiamento della topologia del circuito, illustrato in Figura 3.2 .7, propone una struttura rettangolare, che può essere riprodotta facilmente sul *layout*.

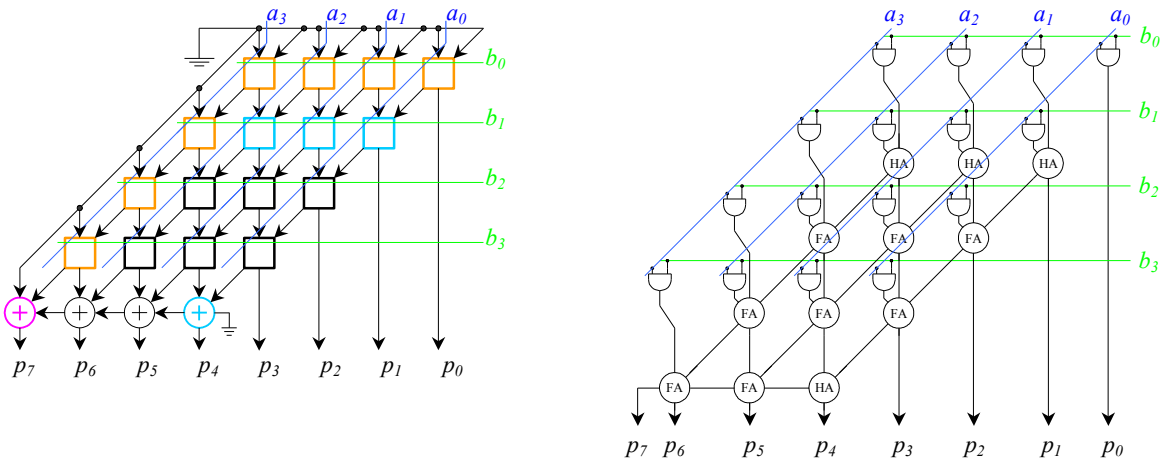

Figura 3.2.6. Ottimizzazione della struttura.

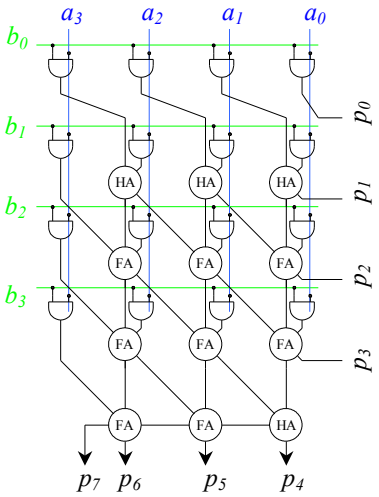

Figura 3.2.7. Disposizione delle celle legata alla realizzazione del layout.

3.3 Moltiplicatore parallelo per numeri espressi in complemento a 2 (*Baugh-Wooley Multiplier*)

La struttura illustrata nel paragrafo precedente può essere modificata per implementare un moltiplicatore parallelo in grado di operare su numeri espressi in complemento a 2.

Si considerino i due numeri da moltiplicare A e B:

$$A = (a_{N-1},....,a_0) = -a_{N-1}2^{N-1} + \sum_{i=0}^{N-2} a_i 2^i$$

$$B = (b_{N-1},....,b_0) = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$$

Allora il prodotto tra i due numeri è espresso da:

$$AB = a_{N-1}b_{N-1}2^{2N-2} + \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} a_i b_j 2^{i+j} - a_{N-1}2^{N-1}\sum_{i=0}^{N-2} b_i 2^i - b_{N-1}2^{N-1}\sum_{i=0}^{N-2} a_i 2^i$$

(eq. 3.3.1)

Si possono trasformare le sottrazioni che compaiono nella eq. 3.3 .1 in somme:

$$-a_{N-1}\sum_{i=0}^{N-2} b_i 2^{N+i-1} = a_{N-1}2^{N-1}\left[-2^{N-1} + 1 + \sum_{i=0}^{N-2}\overline{b_i}2^i\right]$$

Quindi:

$$AB = a_{N-1}b_{N-1}2^{2N-2} + \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} a_i b_j 2^{i+j} +$$
$$+ a_{N-1}2^{N-1}\left[-2^{N-1} + 1 + \sum_{i=0}^{N-2}\overline{b_i}2^i\right] + b_{N-1}2^{N-1}\left[-2^{N-1} + 1 + \sum_{i=0}^{N-2}\overline{a_i}2^i\right]$$

Dato che per numeri espressi in complemento a due vale l'equazione:

$$-(a_{N-1} + b_{N-1})2^{2N-2} = -2^{2N-1} + (\overline{a_{N-1}} + \overline{b_{N-1}})2^{2N-2}$$

Si ottiene, come equazione finale:

$$AB = -2^{2N-1} + (\overline{a_{N-1}} + \overline{b_{N-1}} + a_{N-1}b_{N-1})2^{2N-2} + \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} a_i b_j 2^{i+j} +$$
$$+ (a_{N-1} + b_{N-1})2^{N-1} + a_{N-1}2^{N-1}\sum_{i=0}^{N-2}\overline{b_i}2^i + b_{N-1}2^{N-1}\sum_{i=0}^{N-2}\overline{a_i}2^i \qquad \text{(eq. 3.3.2)}$$

Poiché il risultato deve essere espresso su $2N$ bit, sommare il termine -2^{2N-1} della eq. 3.3 .2 equivale a sommare 1 in corrispondenza del blocco che genera il bit più significativo:

$$AB = 2^{2N-1} + (\overline{a_{N-1}} + \overline{b_{N-1}} + a_{N-1}b_{N-1})2^{2N-2} + \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} a_i b_j 2^{i+j} +$$
$$+ (a_{N-1} + b_{N-1})2^{N-1} + a_{N-1}2^{N-1}\sum_{i=0}^{N-2}\overline{b_i}2^i + b_{N-1}2^{N-1}\sum_{i=0}^{N-2}\overline{a_i}2^i$$

Il circuito è mostrato in Figura 3.3 .8.

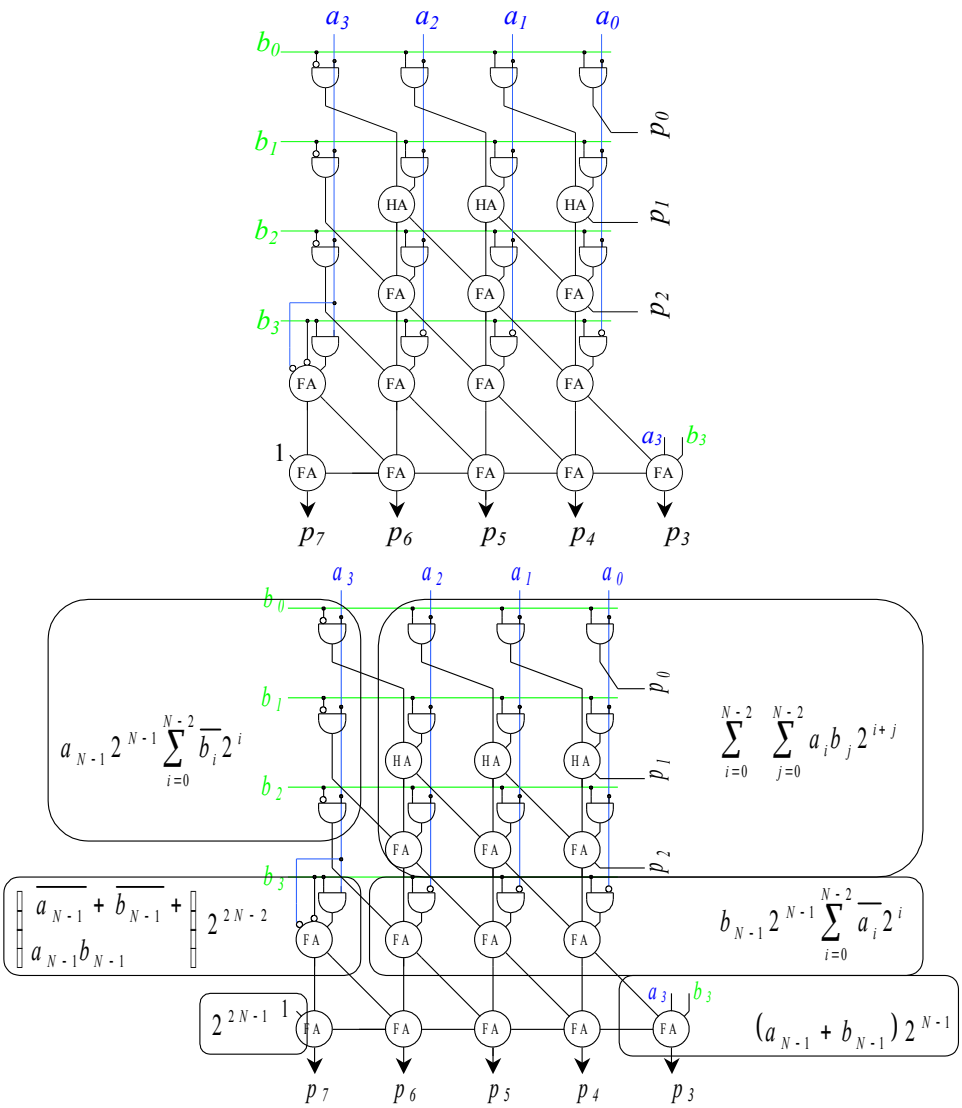

Figura 3.3.8. Moltiplicazione tra due operandi espressi in complemento a 2.

3.4 Moltiplicatore di BOOTH

È possibile, come già visto per i sommatori, migliorare le caratteristiche dei moltiplicatori.
In particolare, è importante riuscire a superare i principali difetti dell'architettura riportata nel paragrafo precedente, ovvero l'aumento (la funzione è quadratica) della complessità totale e dell'area occupata su silicio all'aumentare della lunghezza delle parole che devono essere moltiplicate e la diminuzione della velocità di computazione all'aumentare della profondità di parola degli operandi.
È utile pensare all'operazione di moltiplicazione come all'insieme di due operazioni distinte: la generazione dei prodotti parziali, e la somma opportuna di questi ultimi. Gli algoritmi basati sulla codifica di Booth, che agiscono sulla parte di generazione, vengono utilizzati perché riducono il numero di prodotti parziali da calcolare e permettono quindi di computare mediante la somma il risultato in modo più semplice. I prodotti parziali sono inoltre spesso ridotti a semplici operazioni di scorrimento a sinistra (moltiplicazione per due, quattro, otto,…) e di complemento a due (moltiplicazione per -1), che sono facilmente implementabili.
L'algoritmo di Booth prevede, partendo dalla parola binaria originale, un particolare ordine di raggruppamento dei bit, in modo da formare una nuova codifica della parola di partenza.
Questa operazione permette di ridurre la lunghezza della parola pur non perdendo precisione; è grazie a questa riduzione di lunghezza che si riesce a ridurre il numero di prodotti parziali.

Si definisce *radice dell'algoritmo di Booth* il valore 2^n dove n è il numero di bit raggruppati.

Il procedimento per la codifica di Booth viene descritto su un esempio concreto.
Si supponga di avere una parola di lunghezza pari a 8 bit, la cui espressione formale, in complemento a 2, risulta essere:

$$A = -2^7 a_7 + \sum_{i=0}^{6} 2^i a_i \ .$$

Possibili radici dell'algoritmo per un numero espresso da una parola di lunghezza pari a 8 bit sono 2^2, 2^3, 2^4, 2^5, etc. cioè *radice* 4, 8, 16, 32, etc.

Il procedimento per effettuare correttamente il raggruppamento è il seguente:

1. a destra del bit meno significativo (*LSB*) si aggiunge un bit di valore 0 e di posizione a_{-1}: $a_{-1} = 0$;
2. a partire da a_{-1} si formano i gruppi di bit in base alla radice scelta, facendo in modo che il bit più significativo (*MSB*) di ogni gruppo sia anche *LSB* per il gruppo successivo;
3. se si devono aggiungere dei bit oltre l'*MSB* per completare un gruppo, si effettua un'estensione di segno (aggiungendo uno o più bit dello stesso valore dell'*MSB*).

In base al procedimento appena descritto si ha:

- Radice 4: $\underbrace{a_7 a_6}_{2}\ \underbrace{a_5 a_4}_{2}\ \underbrace{a_3 a_2}_{1}\ \underbrace{a_1 a_0}_{0}\ \underbrace{a_{-1}}_{1}$
- Radice 8: $\underbrace{a_7 a_6 a_5}_{1}\ \underbrace{a_5 a_4 a_3}_{1}\ \underbrace{a_2 a_1 a_0}_{0}\ \underbrace{a_{-1}}_{1}$
- Radice 16: $\underbrace{a_7 a_6 a_5 a_4}_{1}\ \underbrace{a_3 a_2 a_1 a_0}_{0}\ \underbrace{a_{-1}}_{1}$
- Radice 32: $\underbrace{a_7 a_6 a_5 a_4 a_3}_{1}\ \underbrace{a_4 a_3 a_2 a_1 a_0}_{0}\ \underbrace{a_{-1}}_{1}$

Le radici successive alla 16 non portano vantaggi poiché comportano sempre due raggruppamenti.
A questo punto si può scrivere, con notazione di Booth:

$$A = \sum_{m=0}^{G-1} 2^{n\,m} a_m^*$$

dove G = # *gruppi*, n = # bit della parola originale raggruppati, a_m^* = coefficienti nuova codifica.
Per le codifiche radice 4, 8 e 16 si ha:

$$A_{radice4} = \sum_{m=0}^{G-1} 2^{nm} a_m^* = \sum_{m=0}^{G-1} 2^{nm} a_m^* = a_0^* + 2^2 \cdot a_1^* + 2^4 \cdot a_2^* + 2^6 \cdot a_3^*$$

$$A_{radice8} = \sum_{m=0}^{G-1} 2^{nm} a_m^* = \sum_{m=0}^{2} 2^{3m} a_m^* = a_0^* + 2^3 \cdot a_1^* + 2^6 \cdot a_2^*$$

$$A_{radice16} = \sum_{m=0}^{G-1} 2^{nm} a_m^* = \sum_{m=0}^{1} 2^{4m} a_m^* = a_0^* + 2^4 \cdot a_1^*$$

I coefficienti a_m^* possono essere espressi in termini dei bit della parola di partenza:

$$a_m^* = \sum_{i=0}^{n} c_{n(m+i-1)} a_{n(m+i-1)}$$

Nei tre casi si ha:

- Radice 4:

$$A = \sum_{m=0}^{3} 2^{2m} a_m^*; \qquad a_m^* = c_{2m+1} \cdot a_{2m+1} + c_{2m} \cdot a_{2m} + c_{2m-1} \cdot a_{2m-1}$$

- Radice 8 :

$$A = \sum_{m=0}^{2} 2^{3m} a_m^*; \qquad a_m^* = c_{3m+2} \cdot a_{3m+2} + c_{3m+1} \cdot a_{3m+1} + c_{3m} \cdot a_{3m} + c_{3m-1} \cdot a_{3m-1}$$

- Radice 16:

$$A = \sum_{m=0}^{1} 2^{4m} a_m^*;$$

$$a_m^* = c_{4m+3} \cdot a_{4m+3} + c_{4m+2} \cdot a_{4m+2} + c_{4m+1} \cdot a_{4m+1} + c_{4m} \cdot a_{4m} + c_{4m-1} \cdot a_{4m-1}$$

Rimangono da calcolare i coefficienti della codifica c_i: per determinarli si deve risolvere l'equazione che lega la notazione di partenza e la notazione di Booth:

$$-2^7 a_7 + \sum_{i=0}^{6} 2^i a_i = \sum_{m=0}^{G-1} 2^{nm} a_m^*$$

Raggruppando i coefficienti a pari a_i si trova:

- radice 4: $c_{2m+1} = -2$; $c_{2m} = 1$; $c_{2m-1} = 1$;
- radice 8: $c_{3m+2} = -4$; $c_{3m+1} = 2$; $c_{3m} = 1$; $c_{3m-1} = 1$;
- Radice 16: $c_{4m+3} = -8$; $c_{4m+2} = 4$; $c_{4m+1} = 2$; $c_{4m} = 1$; $c_{4m-1} = 1$.

Sostituendo ad A tutte le possibili combinazioni dei bit in ingresso come riportato nelle Figura 3.4 .9 e Figura 3.4 .10, si vede che i valori di a_m^* sono limitati ai valori ± 1, ± 2 per la radice 4 e ± 1, ± 2, ± 3, ± 4 per la radice 8 [1].

A questo punto, nota la codifica di Booth dei coefficienti, la moltiplicazione di A per un generico valore numerico B diventa semplicemente:

$$A \cdot B = B \cdot \sum_{m=0}^{G-1} 2^{nm} a_m^* = B \cdot a_0^* + B \cdot 2^n \cdot a_1^* + B \cdot 2^{2n} \cdot a_2^* + \ldots$$

[1] Non viene da qui in avanti analizzata nel dettaglio la situazione corrispondente alla radice 16.

RADICE 4	000	001	010	011	100	101	110	111
$a_0^* = -2 \cdot a_1 + a_0 + a_{-1}$	0	NO	1	NO	-2	NO	-1	NO
$a_1^* = -2 \cdot a_3 + a_2 + a_1$	0	1	1	2	-2	-1	-1	0
$a_2^* = -2 \cdot a_5 + a_4 + a_3$	0	1	1	2	-2	-1	-1	0
$a_3^* = -2 \cdot a_7 + a_6 + a_5$	0	1	1	2	-2	-1	-1	0

Figura 3.4.9. Valori di a_m^* in un moltiplicatore di Booth radice 4 per tutte le possibili combinazioni dei bit di A. L'indicazione NO esprime una combinazione non accettabile dei bit

RADICE 8	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$a_0^* = -4 \cdot a_2 + 2 \cdot a_1 + a_0 + a_{-1}$	0	NO	1	NO	2	NO	3	NO	-4	NO	-3	NO	-2	NO	-1	NO
$a_1^* = -4 \cdot a_5 + 2 \cdot a_4 + a_3 + a_2$	0	1	1	2	2	3	3	4	-4	-3	-3	-2	-2	-1	-1	0
$a_2^* = -4 \cdot a_7 + 2 \cdot a_7 + a_6 + a_5$	0	1	1	2	2	3	3	4	-4	-3	-3	-2	-2	-1	-1	0

Figura 3.4.10. Valori di a_m^* in un moltiplicatore di Booth radice 8 per tutte le possibili combinazioni dei bit di A. L'indicazione NO esprime una combinazione non accettabile dei bit

RADICE 16	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111
a_0^*	0	NO	1	NO	2	NO	3	NO	4	NO	5	NO	6	NO	7	NO
a_1^*	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8

RADICE 16	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111
a_0^*	-8	NO	-7	NO	-6	NO	-5	NO	-4	NO	-3	NO	-2	NO	-1	NO
a_1^*	-8	-7	-7	-6	-6	-5	-5	-4	-4	-3	-3	-2	-2	-1	-1	0

Figura 3.4.11. Valori di a_m^* ($a_0^* = -8a_3 + 4 \cdot a_2 + 2 \cdot a_1 + a_0 + a_{-1}$, $a_1^* = -8a_7 + 4 \cdot a_6 + 2 \cdot a_5 + a_4 + a_3$) in un moltiplicatore di Booth radice 16 per tutte le possibili combinazioni dei bit di A. L'indicazione NO esprime una combinazione non accettabile dei bit

A parte il caso ± 3, l'operazione $A\ B$ si riduce ad operazioni di complemento a 2 e scorrimento a sinistra del valore di B.

A livello *hardware* tutto questo si traduce nel produrre contemporaneamente tutti i possibili prodotti parziali $B \cdot a_0^*$, $B \cdot a_1^*$, $B \cdot a_2^*$,... manipolando il valore B ed usando i bit di A come ingressi di selezione di multiplexer i cui ingressi dati sono costituiti dai prodotti parziali.

Il numero di multiplexer dipende dal numero di gruppi e quindi di prodotti parziali.
Le uscite dei multiplexer dovranno essere moltiplicate per il peso corrispondente: 1, 2^n, 2^{n-2}, etc.

Questa operazione corrisponde ad effettuare un ulteriore scorrimento a sinistra prima che le uscite vengano sommate.
Nella Figura 3.4 .12 viene rappresentato lo schema a blocchi del moltiplicatore di Booth radice 4.

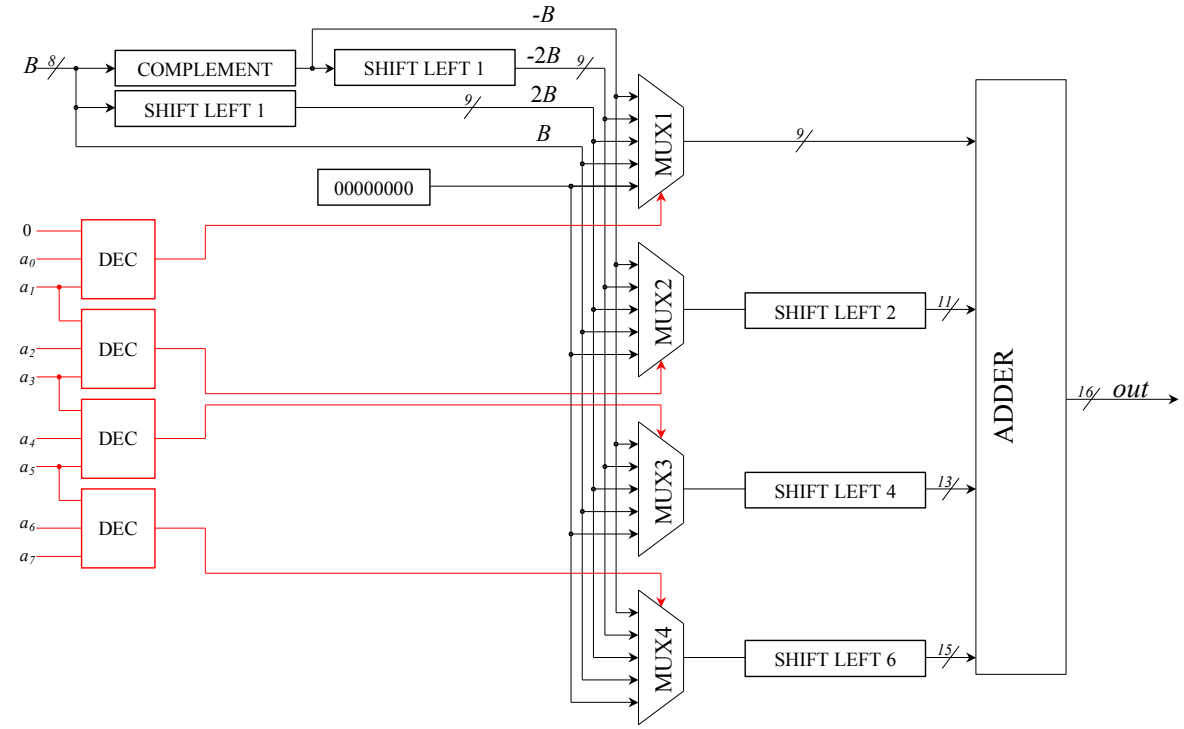

Figura 3.4.12. Schema a blocchi del moltiplicatore di Booth radice 4

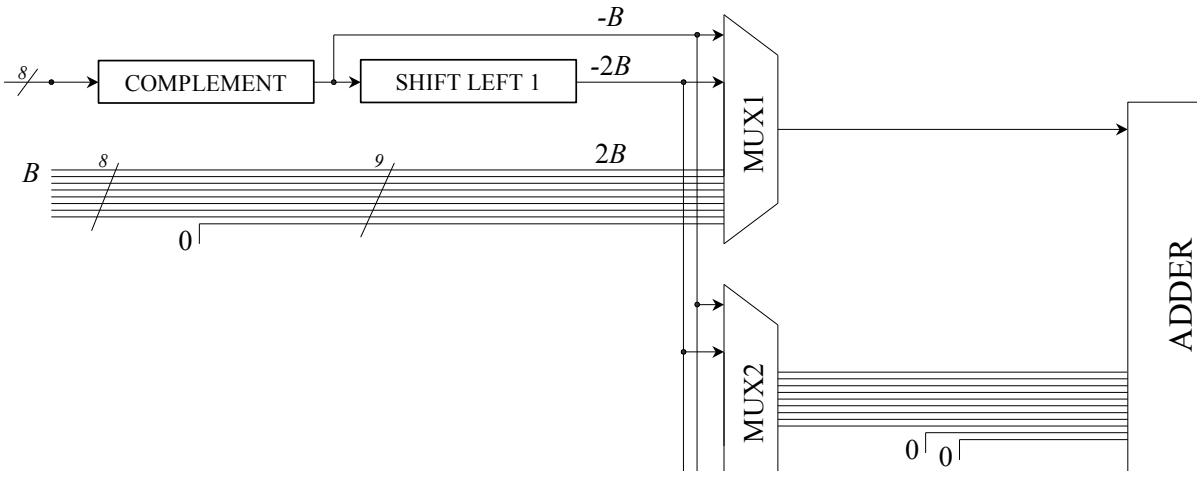

Figura 3.4.13. Moltiplicatore di Booth radice 4: dettaglio

3.5 Moltiplicatore: approccio seriale

Anche il prodotto tra due numeri può essere calcolato in modo seriale combinando opportunamente operazioni di *shift* e somma.
A titolo di esempio, si consideri la situazione mostrata in Figura 3.1 .1, dove le due parole A e B da moltiplicare sono espresse in modulo su 4 bit Si noti come la somma dei prodotti parziali calcolati ad ogni iterazione riguardi sempre parole di lunghezza uguale alla lunghezza del moltiplicando (A). Il sommatore che implementa questa operazione è quindi un sommatore fissato, che elabora semplicemente ad ogni ciclo parole differenti.
In Figura 3.5 .15 viene mostrata l'implementazione *hardware* del circuito, nel caso specifico in cui il moltiplicando abbia lunghezza pari a *N*, sia nella versione derivante direttamente dall'algoritmo, sia in una versione "ottimizzata", in cui il moltiplicatore va ad occupare inizialmente le posizioni meno significative dello *shift register* destinato ad immagazzinare il prodotto: ad ogni fronte attivo del segnale di clock, un nuovo bit (dal meno significativo al più significativo) della parola B viene indirizzato verso il *multiplexer*, liberando posizioni che vengono occupate dal prodotto. Si noti come l'operazione di moltiplicazione venga effettuata ripetendo A o sommando '0' a seconda del bit b_i corrente.

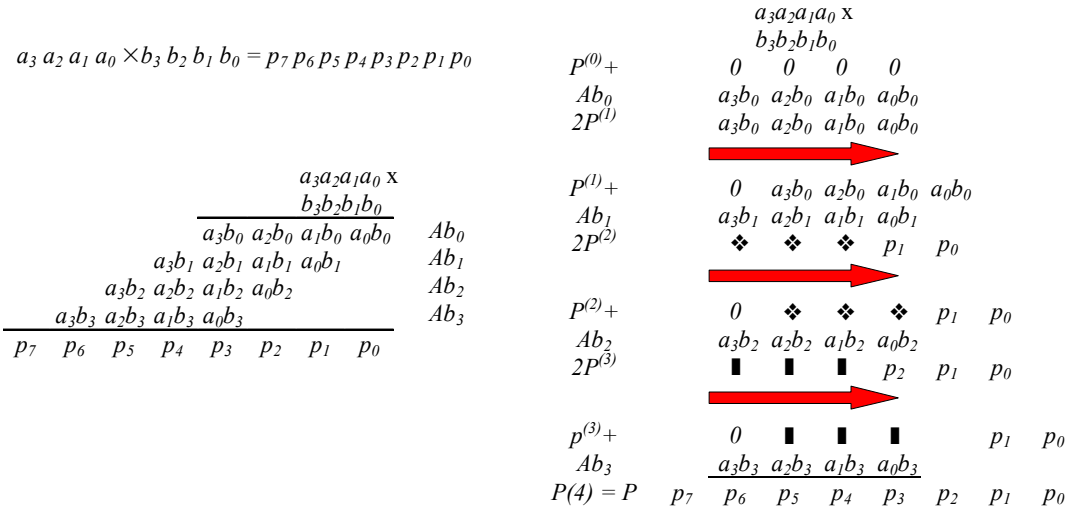

Figura 3.5.14. Algoritmo elementare di moltiplicazione di due numeri espressi in modulo su 4 bit implementato mediante shift e somme.

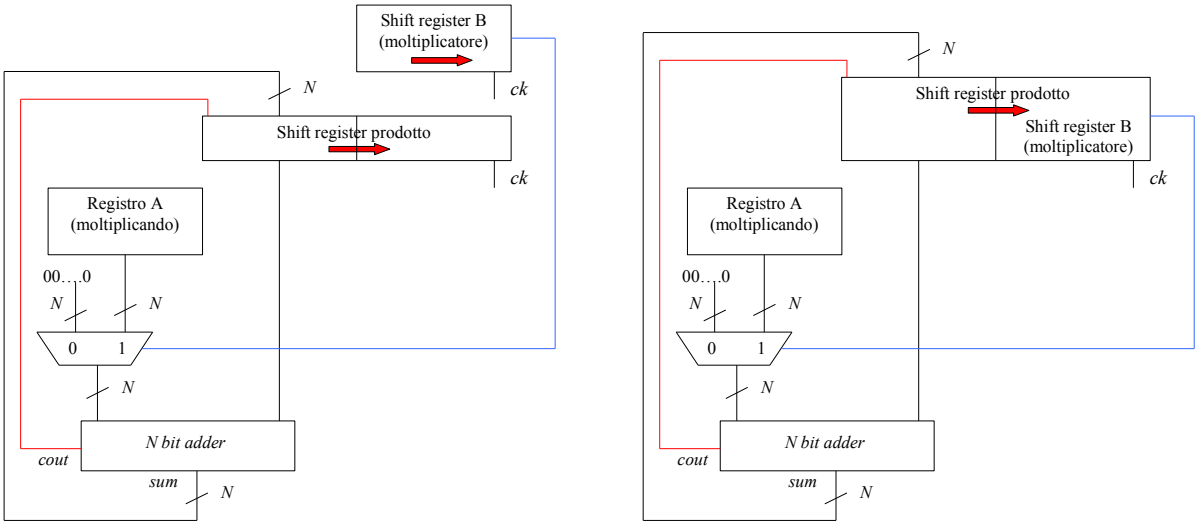

Figura 3.5.15. Implementazione della moltiplicazione di due numeri espressi in modulo mediante shift e somme.

L'architettura può essere adattata anche alla moltiplicazione di numeri espressi in complemento a 2, avendo l'avvertenza di espendere il segno degli addendi prima di effettuare le somme.

Indicando con t_{Nadder} il ritardo introdotto dal sommatore su N bit (il valore dipende dall'implementazione *hardware* del sommatore), e con t_{mux} il ritardo introdotto dal *multiplexer*, si ricava che il periodo del segnale *ck* (clock) minimo che deve essere assicurato per garantire il corretto funzionamento della rete sincrona è:

$$T_{ck,min} = t_{ck,q} + t_{Nadder} + t_{mux} + t_{su}$$

Il tempo impiegato dal circuito per calcolare la somma è legato a $T_{ck,min}$ e alla lunghezza N delle parole A e B da moltiplicare:

$$T_{moltiplicatoreseriale} = N\, T_{ck,min}$$