CAPITOLO 7

7.0 Generalità sui Circuiti Aritmetici fondamentali

Un circuito aritmetico è un qualsiasi circuito che può eseguire delle somme, sottrazioni, prodotti, divisioni o qualsiasi altra funzione aritmetica con numeri binari. Gli elementi aritmetici appartengono certamente nel dominio della tecnologia dei computer e delle relative tecnologie di calcolatori elettronici. Anche se i computer e i calcolatori elettronici non sono solo sistemi elettronici digitali, che fanno soltanto calcoli aritmetici, è in quella area che gli elementi aritmetici sono stati sviluppati al loro presente grado di versatilità, velocità e totale sofisticazione.
L'argomento degli elementi aritmetici si è sviluppato da molti anni e quindi potrebbe essere a loro dedicato un libro intero. In questo capitolo descriveremo soltanto i concetti fondamentali della addizione e sottrazione binaria e daremo esempi di circuiti che realizzano quelle funzioni. Introdurremo anche qualche comune variante dei metodi di somma e sottrazione.
Con tutta la complessità dei circuiti dei computer, è rassicurante sapere che la base di tutte le operazioni aritmetiche che un computer fa è un semplice circuito che consiste di un piccolo numero di porte: il **full adder**. Questo esegue semplicemente la somma di due bit (e di un riporto) e a sua volta può essere usato per sottrarre, moltiplicare, dividere, estrarre la radice quadrata e fare molte altre funzioni matematiche. La chiave di tutte queste funzioni è quella di seguire una particolare sequenza di passi. Questi passi, o procedure, sono noti con il nome di algoritmi e sono scritti in un programma di computer, o costruiti direttamente in circuiti di controllo che azionano il circuito di somma. Quindi, dopo che è stata fatta una disanima del circuito sommatore, la nostra attenzione sarà spostata dalla descrizione dell'hardware (cioè dei circuiti) e ometteremo lo sviluppo e la descrizione dei vari algoritmi associati agli elementi aritmetici.
Diremo qui soltanto che, poiché esistono questi algoritmi, ci sono dei circuiti ausiliari che manipolano e distribuiscono i dati in modo tale che il full adder può essere usato per fare una particolare operazione aritmetica. Infatti, nel progetto di un computer bisogna fare una scelta fondamentale – progettare un circuito più complesso attorno al full adder e permettere all'hardware di eseguire l'operazione o usare un programma più lungo e eseguirlo in software. Per esempio, dobbiamo progettare un circuito separato per eseguire il prodotto di due numeri, oppure dobbiamo usare più e più volte il circuito somma utilizzando istruzioni di programma? L'hardware è più veloce, ma il software è meno costoso (una volta che il programma è scritto e pagato può essere usato quante volte si vuole). Quindi i grossi computer sono versatili, veloci e costosi, e i piccoli computer richiedono più tempo per eseguire le stesse operazioni in software.
Infine, c'è una ultima nota da fare riguardo alla descrizione dei circuiti aritmetici. La maggior parte dei computer usano il codice binario puro, 8421, per rappresentare i numeri che debbono essere memorizzati e manipolati. Tuttavia è possibile eseguire le operazioni aritmetiche, con lo stesso full adder in altri codici binari a patto di seguire un appropriato algoritmo. Per esempio, le calcolatrici

generalmente usano il codice BCD, poiché le entrate provengono da una tastiera decimale, i risultati sono mostrati in forma decimale ed è più facile fare la conversione dei numeri BCD in decimale (e decimale in BCD) che binario a decimale.

7.1 Funzioni aritmetiche Integrate

La discussione che ora faremo avrà come scopo finale, oltre alla spiegazione dei principi fondamentali su cui il full adder si basa, anche quello di presentare la materia in modo da illustrare il funzionamento di queste funzioni realizzate con circuiti integrati.
Ciascun elemento aritmetico fondamentale è usato spesso come un semplice circuito integrato. Il circuito full adder, per esempio, è tipicamente disponibile nella forma di quattro indipendenti full adder nello stesso package, oppure come un sommatore a 4 bit che può sommare due parole a 4 bit o nibble. I circuiti full adder sono poi combinati con altri elementi logici per realizzare sommatori/sottrattori, moltiplicatori, e unità logiche di uso generale. Queste Unità Logiche-Aritmetiche (ALU) hanno la possibilità di eseguire molteplici e differenti operazioni logico aritmetiche. I chip di tali circuiti integrati contengono l'equivalente di un numero molto grande di porte logiche.
Il massimo della integrazione di molte funzioni in un solo chip è il microprocessore. Questi chip LSI (**L**arge **S**cale **I**ntegration) tipicamente contengono tutta la logica che è presente in una completa unità di un computer. Un chip microprocessore è tipicamente venduto assieme ad altri chip che eseguono funzioni di controllo, ingresso/uscita, e memoria. Tutti questi chip assieme formano un semplice computer.

7.2 Somma binaria

Le regole per fare la somma di due numeri binari sono le stesse di quelle usate da tutti noi per fare la somma di numeri decimali (essendo, le due rappresentazioni numeriche, pesate). Questo è stato spiegato nel Cap. 1. Nel caso della somma di due numeri ad un bit le regole sono rappresentate nella seguente tabella di verità, Tab. 7.1.

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0
C= AB			
S=A'B+AB'=A ⊕B			

Tab. 7.1

La colonna intestata, '*S*' fornisce la funzione Booleana per la somma (0+0=0; 0+1=1; 1+0=1; e la cifra meno significativa quando la somma è $1+1=2_{10}=(10)_2$)
Quella intestata, '*C*' o carry fornisce la funzione Booleana del riporto o bit più significativo del risultato (è necessario esplicitarlo quando il risultato della somma supera la rappresentazione simbolica della cifra nella base scelta). Nel caso presente il carry è diverso da zero solo per la somma di $1+1=2=(10)_2$. Le funzioni Booleane che realizzano le due funzioni sono riportate nella stessa tabella

7.3 Half Adder

Un circuito logico che può essere usato per iniziare a realizzare l'addizione, in accordo all'algoritmo presentato nella Tab. 7.1, è il circuito di Fig. 7.1. Notare che non è realizzabile con una sola porta in quanto la corretta interpretazione della somma prevede due funzione Booleane in uscita. Perché un Half adder possa essere utile deve essere modificato. Infatti esso mentre è adeguato per la somma di due bit binari, certamente non lo è per due numeri binari costituiti da più di un bit.

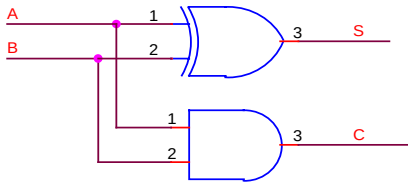


Fig. 7.1

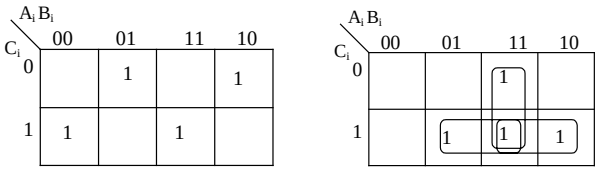
7.4 **Full Adder**

Quando i numeri da sommare hanno più di un bit bisogna disporre di un sommatore separato per ogni coppia di bit e l'addizione può essere fatta in parallelo.

Così facendo è facile vedere che la somma di due bit in una certa posizione influenza il risultato della somma dei bit in posizione più significativa (alla sua sinistra) in quanto è necessario che questa tenga conto se, nella somma del bit di peso immediatamente inferiore, ci sia stato un riporto. Questo è mostrato nella tabella di verità Tab. 7.2. In questa una delle colonne è intestata, C_i, in cui vengono rappresentati i valori del bit di carry provenienti dalla posizione *(i-1).ma* che bisogna sommare con i bit in posizione *i.ma*. Le altre due colonne di ingresso sono intestate con i simboli precedentemente usati, *A* e *B*, per indicare il primo ed secondo addendo rispettivamente e sono stati dotati di un indice di posizione del bit. Le colonne di uscita sono intestate come nel caso precedente, *C* e *S*, ma in questo caso *C* è stato indiciato con *i+1* (per indicare che il bit di carry che va sommato ai bit di ingresso della posizione immediatamente più significativa *(i+1).ma*). Ovviamente l'indice di *S* è proprio *i* per indicare che si tratta della bit *i.mo* della somma. La Fig. 7.2 mostra le mappe relative alla tab. 7.2 e la minimizzazione delle funzioni Booleane ivi rappresentate.

C_i	A_i	B_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1
$C_{i+1}= (A_i \oplus B_i)C_i + A_iB_i$				
$S_i=A_i \oplus B_i \oplus C_i$				

Tab. 7.2



$S_i = A_i \oplus B_i \oplus C_i$ $C_{i+1} = A_iB_i + B_iC_i + A_iC_i$

Fig. 7.2

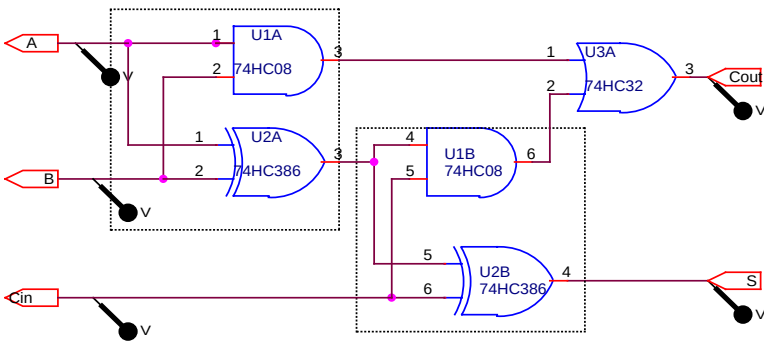


Fig. 7.3

Sebbene il full adder può essere costruito in molti modi noi daremo prima una realizzazione (vedi

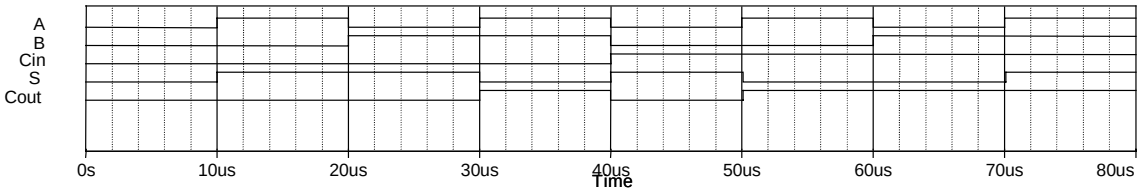


Fig 7.4

Fig 7.3) che usa due half adder del tipo già presentato e che derivano dalla funzione booleana scritta, come in Tab. 7.2, sia per S_i, sia per C_{i+1} (anche se quest'ultima non è nella forma minima presentata in Fig. 7.2). Nel seguito le stesse funzioni saranno espresse in altra forma. Nel circuito di Fig. 7.3, per meglio evidenziarne graficamente l'uso dei due half adder che realizzano il full adder, ciascuno di essi, è stato racchiuso in un rettangolo tratteggiato. Il timing presentato nella Fig. 7.4 mostra l'uscita di un full adder che somma una generica coppia di bit. Infatti, al bit di carry è dato il valore zero (come deve essere nel caso della somma dei bit meno significativi), ed il valore 1 (simula il carry proveniente dalla somma di una coppia di bit immediatamente meno significativa).

7.5 Parallel Adder

Un parallel adder consiste nella interconnessione di parecchi stadi di circuiti full adder in modo che il carry di uscita di uno stadio diventi il carry di ingresso dello stadio successivo; la Fig. 7.5 mostra il collegamento di 4 di tali stadi. Il sommatore parallelo a 4 stadi che ne deriva ha una uscita carry che potrà essere usato come ingresso ad un altro sommatore parallelo a 4 bit.
Evidenziamo ora le caratteristiche più significative di un sommatore parallelo.

- Anche se chiamato parallelo, il sommatore lavora in modo sequenziale. Nella somma di due numeri per ogni stadio della somma è generato un bit di carry. Il primo adder, *Fulladder_0*, deve completare la somma di A0 e B0 per poter generare il carry, C1 (che è in ingresso al secondo stadio, *Fulladder_0_1*). Quindi *Fulladder_0_1* non può fare correttamente la sua parte, nella sequenza di somme, se prima *Fulladder_0* non ha completata la somma ed il calcolo del carry. Allo stesso modo si comporta, *Fulladder_0_2* e i successivi.

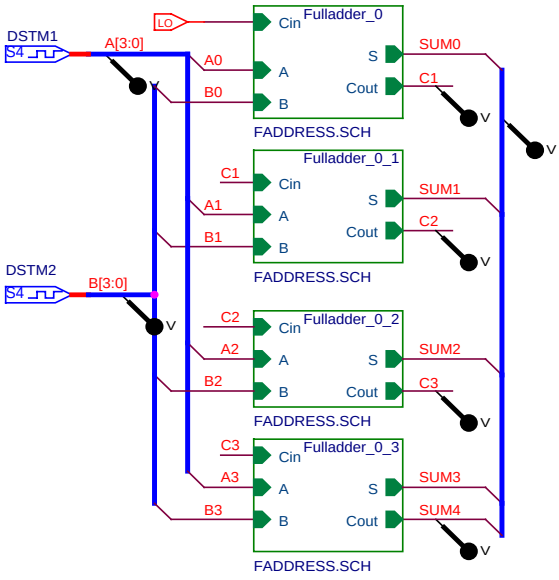


Fig 7.5

- I riporti passano sequenzialmente attraverso tutti gli stadi del sommatore parallelo e l'uscita dell'ultima somma non è corretta fino a che non

sia stato generato l'ultimo riporto. Per questo motivo il sommatore di Fig. 7.5 è conosciuto come sommatore parallelo con "*ripple carry*".

Il timing di Fig. 7.6 mostra il comportamento del full adder di Fig. 74, quando è connesso nella serie di quattro stadi di Fig. 7.5.

Questo, quando visto con una risoluzione temporale non sufficiente ad evidenziare gli effetti dei ritardi sopra evidenziati, sembra fornire i risultati corretti (nel timing il bit sum4 è incluso nel risultato come bit più significativo).

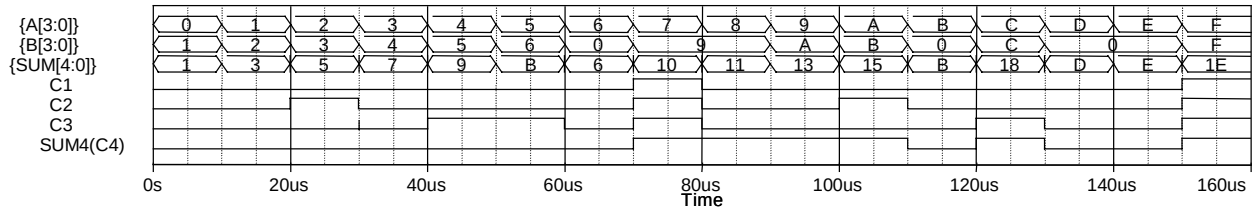


Fig. 7.6

Il timing di Fig. 7.7 è stato ricavato da quello di Fig. 7.6 "zoommando" attorno a 70 μs, che è la parte tra le più interessante; la somma passa da 6+0=6 (tutti i carry sono a zero) a 7+9=16=10_H (in cui tutti i carry, uno dopo l'altro, vanno ad 1 logico). La somma è corretta solo a partire da 70.082

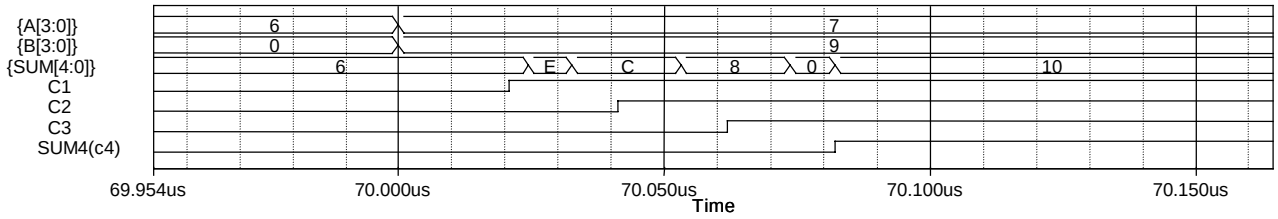


Fig 7.7

μs, cioè dopo 82 ns che sono stati presentati i nuovi addendi. Nel frattempo si vede che l'uscita cambia nella sequenza errata: E, C, 8, 0. Per evitare di fare la lettura di un risultato sbagliato, delle volte, lo si fa passare attraverso dei buffer 3-state che vengono abilitati solo dopo che è trascorso un tempo sufficiente a che il risultato raggiunga il valore corretto.

Se si esegue la stessa espansione del timing, in altre situazioni cioè con due diversi addendi, si può notare che il tempo che deve passare per avere un risultato corretto è minore. Per es. se si somma 9+10=19 si può notare che, non essendoci riporti, il risultato è corretto subito dopo che è trascorso il tempo di propagazione di un solo stadio. La differenza nei tempi che si deve aspettare per avere una somma corretta, ci porta a concludere che i risultati di alcune somme possono essere prelevate prima di altre. Nella maggior parte dei sistemi tuttavia, i sommatori sono campionati in modo sincrono con gli impulsi di clock, quindi nel nostro caso gli impulsi di campionamento non possono essere vicini tra di loro per più di 90-100 μs. Con il che il sistema risulterà lento.

Se la somma potesse essere campionata in modo asincrono, i tempi di campionamento potrebbero essere scelti opportunamente, a seconda dei numeri da sommare. Purtroppo il sommatore qui presentato non ha la capacità di sapere quando la somma è completa. I circuiti necessari, per ottenere ciò, a partire da questo circuito sono complicati e inoltre meno efficienti di quanto possa

essere ottenuto realizzando il circuito sommatore con un schema diverso chiamato *Look-Ahead Carry Adder*, che è oggetto del paragrafo successivo.

7.6 Look-Ahead Carry Adder

Il *Look-ahead Carry Adder* è un sommatore parallelo che non deve aspettare che i carry si propaghino dal full adder precedente al successivo. Ciò è ottenuto *usando della logica aggiuntiva, che anticipa il segnale di carry, prima ancora che le singole somme siano state calcolate*. Questa logica produce il carry corretto, in ingresso, a tutti gli stadi del sommatore, contemporaneamente e prima che essi, separatamente, lo possano produrre. Inoltre può anche anticipare quale sarà l'uscita carry dell'ultimo adder. In questo modo i Look-Ahead Carry Adder sono notevolmente più veloci di quelli descritti precedentemente e come tali sono usati nei computer veloci.
Per capire il principio che sta sotto la logica ausiliaria che anticipa il carry, riprendiamo le soluzione già trovate con le mappe di Fig. 7.2. Manipoleremo algebricamente queste funzioni minime in modo da ottenere una forma ricorsiva sia per la somma e sia per il carry.

- $C_{i+1} = A_iB_i + B_iC_i + A_iC_i = A_iB_i + C_i(A_i + B_i)$ e
- $S_i = A_i \oplus B_i \oplus C_i = (A_i \odot B_i)' \oplus C_i = (A'_iB'_i + A_iB_i)' \oplus C_i =$
  $= (A_i+B_i)(A_iB_i)' \oplus C_i$

Paragonando le ultime espressioni scritte per C_{i+1} e per S_i si nota che il termine, $P_i = (A_i + B_i)$, conosciuto come *Carry Propagate* (propagatore di riporto), è contenuto in entrambi le funzioni e che il termine, $G_i= A_iB_i$, conosciuto come *Carry Generate* (generatore di riporto), è contenuto nel riporto in forma normale e nella somma in forma complementata. Usando la simbologia ora introdotta possiamo scrivere:

- $C_{i+1} = G_i + P_iC_i$
- $S_i = P_i G'_i \oplus C_i$.

Il riporto anticipato, rispetto alla somma, può essere generato, ad ogni stadio del sommatore, realizzando le funzioni sopra ricavate e usando le funzioni ausiliare P_i e G_i in modo opportuno. La funzione G_i è chiamata *Carry Generate* perché, se essa è diversa da zero, ha, già pronto, il riporto per il successivo sommatore. La funzione P_i è chiamato *Carry Propagate* perché, se c'è un riporto nello stadio presente del sommatore, induce un riporto nel prossimo stadio. In altri termini, G_i determina la generazione di un riporto all'iesimo stadio del sommatore e contemporaneamente lo presenta allo stadio successivo, mentre P_i determina la propagazione di un riporto esistente, all'ingresso dell'iesimo stadio, C_i, allo stadio successivo (se uno dei 2 bit è 1).

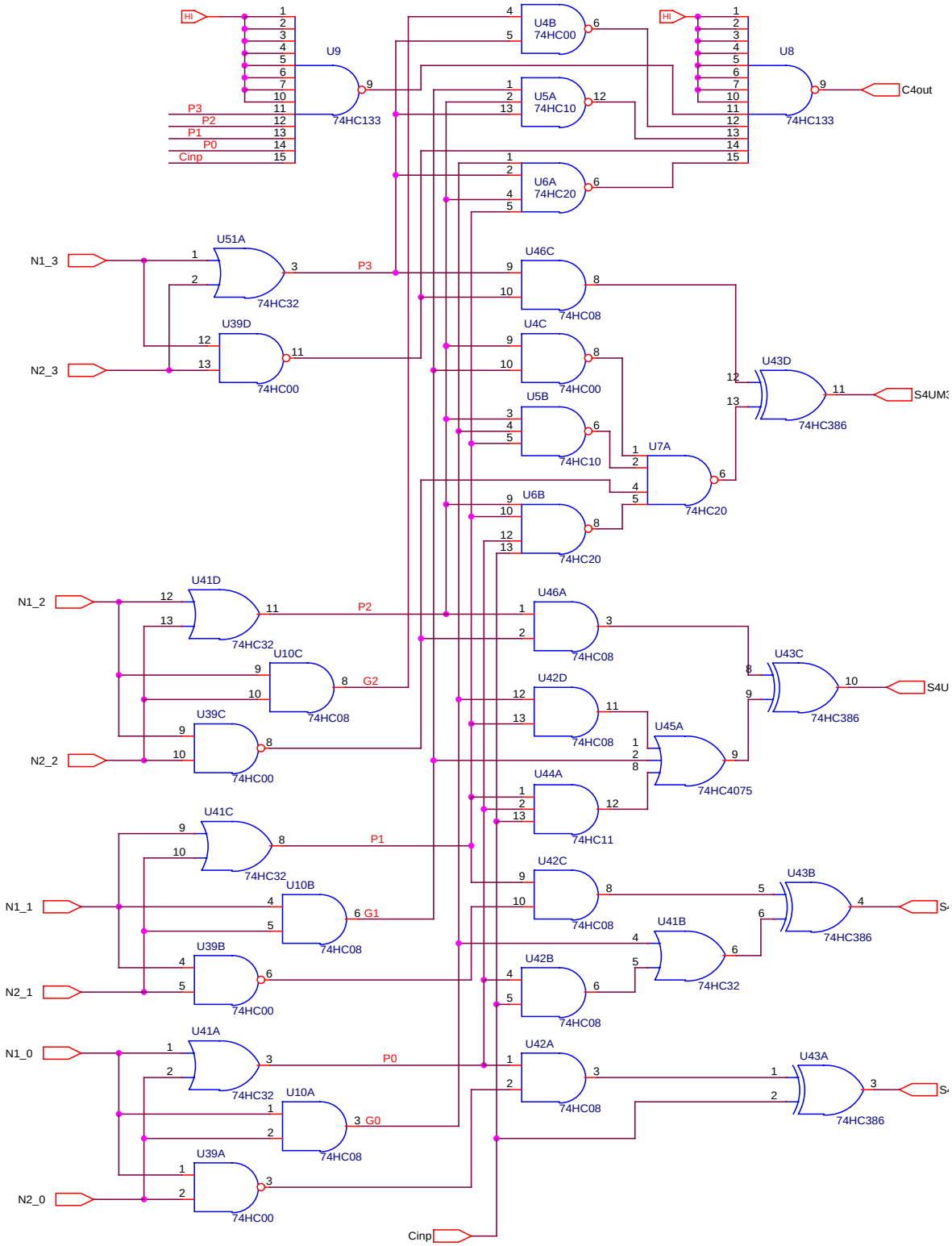


Fig. 7.8

Scriviamo adesso le funzioni Booleane della somma e del riporto richieste per un sommatore a 4 bit.

- $S_0 = P_0\, G'_0 \oplus C_0$ $\qquad\qquad C_1 = G_0 + P_0C_0$
- $S_1 = P_1G'_1 \oplus C_1 = P_1G'_1 \oplus (G_0 + P_0C_0)$
- $S_2 = P_2G'_2 \oplus C_2 = P_2G'_2 \oplus (G_1 + P_1C_1) =$
  $= P_2G'_2 \oplus [G_1 + P_1(G_0 + P_0C_0)] = P_2G'_2 \oplus (G_1 + P_1G_0 + P_0P_1C_0)$
- $S_3 = P'_3G'_3 \oplus C_3 = P_3G'_3 \oplus (G_2 + P_2G_1 + P_2P_1G_0 + P_0P_1P_2C_0)$

- $C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$

Dai risultati sopra elencati è importante notare che tutte le funzioni Booleane delle somme e del carry finale, C_4, possono essere scritte in termini degli ingressi del sommatore, A_i, B_i e C_0.

La realizzazione con porte logiche delle funzioni Booleane sopra scritte è data nella Fig.7.8. Questo è noto come **Zero-Level Look-Ahead Carry Adder**. Questa figura è divisa in due parti: la parte sinistra mostra la realizzazione delle funzioni ausiliarie P_i e G_i mentre la parte destra mostra la logica necessaria per realizzare la somma di una coppia di bit in ciascuna posizione significativa. Notare che i riporti, C_3 e C_4, per mancanza di circuiti logici OR con un numero sufficiente di ingressi, sono stati realizzati con NAND utilizzando la legge di De Morgan. La Fig. 7.9 mostra l'uso di questo circuito per la realizzazione del sommatore di due numeri binari ad 8 bit. Nel timing di Fig. 7.11 si può notare che adesso il circuito somma, il caso peggiore, cioè 9+7=16= $(10)_H$, in circa la metà tempo e che, in tutti i casi, il carry per ogni una delle somme interne, C1-C3, e quindi anche C4, anticipano il calcolo della somma. Ciò è anche evidente dal fatto che ciascun stadio della somma usa il carry relativo come ingresso al XOR da cui si ottiene la somma finale.

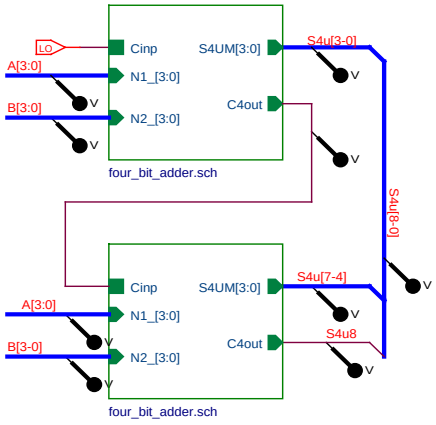


Fig 7.9

Il timing di Fig. 7.10 è stato ottenuto utilizzando il sommatore a 8 bit ottenuto usando due da 4 e inviando il carry del sommatore del nible meno significativo all'altro sommatore (vedi Fig. 7.9). Per la lettura del timing considerare che le due coppie di nibble sono gli stessi e che il risultato della somma del nibble più significativo tiene conto del carry ottenuto dal nibble precedente. Per la scelta fatta si noti che il problema del tempo di propagazione del ripple non è ancora superato, anche se incide meno del caso precedente.

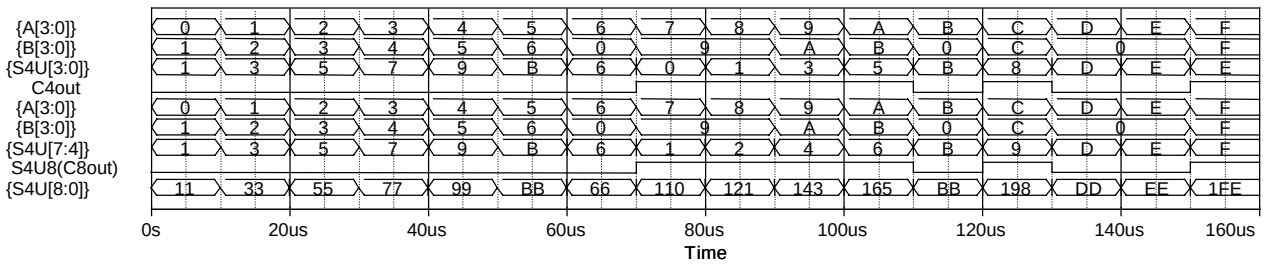


Fig. 7.10

Osserviamo ora, con una risoluzione temporale maggiore, il timing di Fig. 7.10 espandendolo attorno a 70 µs, allo scopo di vedere cosa succede in quell'intorno (sappiamo essere molto critico!).

Intanto il risultato corretto per la coppia di nibble meno significativa si ottiene dopo 48 ns (contro gli 82 del ripple carry). Il risultato corretto per la somma delle due coppie di nibble si ottiene invece dopo 54 ns (circa due terzi di 82 ns ma differisce di appena 6 ns dalla somma dei nibble meno significativi). Una differenza così piccola è dovuta al fatto che il carry, per la coppia di nibble più significativa, era pronto prima che la somma della coppia di nibble meno significativo fosse completa.

Un serio svantaggio del lookhead carry adder è che non appena il numero di bit *n* degli addendi aumenta, la funzione per il carry diventa sempre più complessa ed impraticabile, sia per il grande

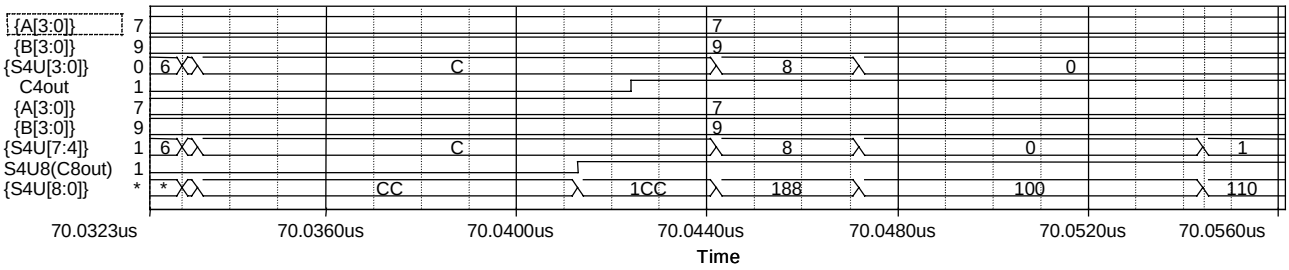


Fig 7.11

numero delle interconnessioni, sia per il fan-out richiesto.

Un modo per utilizzare l'idea produttiva dell'anticipazione del carry al caso di un sommatore a più nibble è la seguente. Consideriamo il carry della somma di una coppia di nible

$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

E notiamo che in esso possiamo distinguere due termini

- Carry Generate dalla coppia del nibble i.esimo $G4_i = (G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0)_i$
- Carry Propagate dalla coppia del nibble i.esimo $P4_i = (P_3P_2P_1P_0)_i$

che dipendono entrambi soltanto dai bit del nibble i.esimo.

Entrambi i termini possono essere generati dal sommatore a 4 bit già mostrato (modificato per realizzare le equazioni ora definite) e possono costituire due sue uscite. Si noti che, poiché all'interno del sommatore così modificato la funzione carry non è più generata, il numero delle sue uscite aumenta di una sola unità.

Adesso introduciamo l'ulteriore circuito elettronico, rappresentato come blocco funzionale in Fig. 7.12. Esso ha in ingresso C_0 e *n* coppie di ingressi $G4_i$ e $P4_i$, uno per ogni nible da sommare e produce in uscita *n* riporti $C4_{i+1}$ (i =1,2,…,n) da usare agli ingressi carry dei relativi sommatori di nible. Dentro questo blocco sono ovviamente implementate le funzioni:

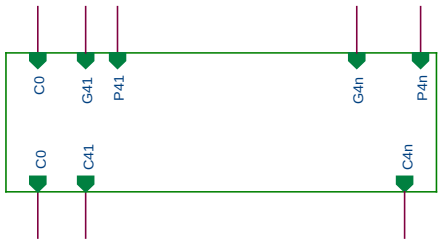


Fig 7.12

- $C4_1 = G4_1 + P4_1C_0$
- $C4_2 = G4_2 + P4_2C4_2 = G4_2 + P4_2(G4_1 + P4_1C_0) = G4_2 + P4_2G4_1 + P4_2P4_1C_0$

per i nible 1 e 2 rispettivamente. Per tutte le altre coppie di nibble, da 3 a n+1), le funzioni possono essere scritte per ricorsività. Questa realizzazione richiede l'introduzione di due ulteriori livelli, per

il raggiungimento delle funzioni Booleane della somma, e quindi il nuovo sommatore risulterà più lento del precedente ma questo è compensato largamente in quanto il risultato è pronto in tempi inferiori. Questa realizzazione è nota come **First Level Auxiliary Carry Function**.

7.6.1 Overflow

Quando si sommano due numeri con *n* cifre il risultato può averne bisogno di *n+1*. Questo è evidente nella somma di 15 + 87 = 102, gli addendi hanno entrambi due cifre decimali ma, il risultato, è di 3 cifre poiché c'è stato un riporto nella somma delle cifre più significative. Se *non* avessimo possibilità di scrivere la cifra di riporto il risultato sarebbe riportato come 02 e quindi sbagliato. Cosa analoga succede quando si sommano due numeri binari ad *n* bit: il risultato può essere di *n+1* bit e se *non* abbiamo una cella di memoria in cui porre il riporto abbiamo un risultato sbagliato; questo errore è detto di *Overflow*. **Un overflow si verifica quando** il risultato di una qualsiasi operazione ha un valore che supera l'intervallo di valori che, con i bit a disposizione, può essere rappresentato, ossia **la capacità di rappresentazione numerica viene superata**.

7.7 Sottrazione Binaria

Prima di introdurre la sottrazione tra numeri binari è necessaria fare alcune osservazioni sull'argomento. Queste nascono dal fatto che il risultato della sottrazione può essere sia un numero positivo sia negativo. La rappresentazione binaria di un numero negativo, storicamente, si è evoluta parecchio. Sostanzialmente sono stati usati tre metodi:

1) Il primo, chiamato del ***segno e grandezza***, prevede di anteporre al numero binario, che ne esprime la grandezza, un bit di segno: se il numero è positivo si antepone uno zero, se invece è negativo si antepone un uno
2) Il secondo, chiamato in ***complemento ad uno***, prevede la rappresentazione dei numeri positivi come nel primo metodo, mentre per quelli negativi se ne fa una rappresentazione in complemento ad 1. Questa si ottiene, complementando i singoli bit della rappresentazione del numero positivo (compreso il bit di segno).
3) Il terzo, chiamato in ***complemento ad due***, prevede la rappresentazione dei numeri positivi come nel primo e secondo metodo, mentre i numeri negativi vengono rappresentati in complemento a due (vedi definizione successiva).

È ovvio che, l'esecuzione dell'operazione di sottrazione dipende anche dal metodo scelto per la sua codifica. Poiché oggi il primo metodo non è più usato, in quel che segue, focalizzeremo la nostra attenzione sulla rappresentazione in complemento a due e ad uno.
Queste rappresentazioni sono preferite perché permette di usare lo stesso circuito, che esegue la somma, anche per la sottrazione. È evidente che ne discende un grande risparmio di costi e complessità. Per spiegare come questo metodo funziona, richiamiamo quanto studiato alle scuole elementari. Ricordiamo che, assieme alla sottrazione, venne introdotto il concetto di complemento a 10 e complemento a 9 dei numeri e tale rappresentazione venne utilizzata come metodo alternativo per eseguire la sottrazione.

7.7.1 Complemento a 10 e complemento a 9

Come si ricorderà, il complemento a 10 di un numero è ottenuto sottraendolo da un numero decimale formato da: *tanti zeri quanti sono le cifre significative del numero, preceduti da un 1*. Per es. il complemento a 10 di 4 = 10 – 4 = 6; il complemento a 10 di 640 = 1000-640 = 360 e così via.
Per semplificare l'operazione del complemento a due veniva spiegato un metodo così riassunto:

- *riscrivere tutti gli zeri meno significativi*
- *al posto della prima cifra, diversa da zero, scrivere il risultato della sua sottrazione da 10*
- *al posto di ogni altra successiva cifra scrivere il risultato della sua sottrazione da 9.*

Il metodo può essere provato funzionare per gli esempio precedenti.
La regola per trovare un numero decimale in complemento a 10 può essere enunciata anche nel modo seguente:

- *Fare il **complementare a 9** di tutte le cifre che formano il numero decimale e poi sommare 1 al risultato*

Il complemento a 9 di un numero decimale si trova sottraendo ciascuna cifra da 9.
Nel nostro caso il complemento a 10 di 640 = 999-640 +1 = 359+1=360 che è la stessa rappresentazione trovata precedentemente.

7.7.2 Complemento a 2 e complemento ad 1

L'equivalente del complemento a 10, per i numeri binari, è il complemento a due. Infatti 10 e 2 sono le basi numeriche usate, rispettivamente, nei due casi. Il numero da cui sottrarre è ottenuto con lo stesso metodo. Il complemento a 2 di $(18)_{10} = (01\ 0010)_2$ è dato da

$$(100\ 0000)_2 - (01\ 0010)_2 = (10\ 1110)_2.$$

La regola di complemento a due si enuncia:

- *riscrivere tutti gli zeri meno significativi*
- *al posto della prima cifra, diversa da zero, scrivere il risultato della sua sottrazione da 2*
- *al posto di ogni altra successiva cifra scrivere il risultato della sua sottrazione da 1*.

Nel caso di numeri binari, la stessa si traduce più semplicemente:

- *riscrivere tutti i bit a zero meno significativi e il primo bit diverso da zero* (Infatti 2-1 = 1)
- *al posto di ogni altro bit scrivere il valore del bit complimentato* (infatti 1- 0 = 1, e 1-1 = 0)

Notare che, nell'esempio precedente abbiamo scritto 18 come $(01\ 0010)_2$ cioè con uno zero in posizione più significativa per indicare che si tratta di un numero positivo.
Il risultato del complemento a due porta, naturalmente, la cifra più significativa ad 1.
La regola per trovare un numero binario in complemento a due può essere enunciata anche nel modo seguente:

- *complementare tutti i bit che formano il numero binario (**complemento ad 1**) e poi sommare 1 al risultato*

Nel nostro caso il complemento di

$$(01\ 0010)_2 \text{ è } (10\ 1101)_2 + 1 = (10\ 1110)_2$$

che è la stessa rappresentazione trovata precedentemente.

Dec Pos.	Bin Pos.	Dec Pos.	Bin. Neg.
0	0000	-0	0000
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010
7	0111	-7	1001
		-8	1000

Tab. 7.3

La Tab. 7.3 mostra i primi numeri interi binari, sia positivi sia negativi, in complemento a due.
Si noti che eseguendo due volte consecutive l'operazione di complemento a 10 o a 2 si ritrova il numero di partenza. Per es. $[(32)_c]_c = (68)_c = 32$.

7.7.3 Sottrazione con numeri scritti in complemento a 10 e a 9

Adesso vogliamo giustificare come la sottrazione può essere ricondotta alla somma del minuendo e del sottraendo, **scritto in complemento a 10**, e dedurre delle osservazioni importanti.

a) Supponiamo di dover calcolare 32 – 15 = 17. Possiamo scrivere $32\text{-}15=32+(10^2\text{-}15)\text{-}10^2 = 32+(15)_c\text{-}10^2 = 32 + 85\text{-}10^2 = (1)17\text{-}10^2 = 17$. La cifra tra parentesi è il riporto della somma 32 + 85. Il risultato è corretto sia eseguendo la sottrazione 117-100 sia scartando la cifra di riporto

(che equivale a fare la detta sottrazione), cioè se teniamo, nel risultato, lo stesso numero di cifre di partenza.

b) Supponiamo ora di dover calcolare $15 - 32 = 15 + (10^2 - 32) - 10^2 = 15 + 68 - 10^2 = (0)83 - 10^2 = -17$ Notare che, in questo caso, abbiamo riporto zero e, non considerando la sottrazione di 10^2, il numero resta scritto in complemento a 10 infatti $(83)_c = 17$.

Da quanto trovato osserviamo che:

- *il risultato dell'operazione (non considerando la cifra di riporto) è in forma normale se il riporto è diverso da zero ed, in forma complementata, se il riporto = 0.*

In conclusione possiamo dire che, ***usando i numeri in complemento a 10 e scartando, nel risultato dell'operazione, la cifra di riporto si ottiene, in ogni caso, il risultato corretto se interpretato in complemento a 10***.

Riassumiamo nel seguente modo quanto osservato :

1) *Prima di eseguire la sottrazione fare il complemento a 10 del sottraendo.*
2) *Eseguire la somma e scartare, dal risultato, il riporto.*

Notare che allo stesso risultato si arriva usando il complemento a 9 se

1) *Prima di eseguire la sottrazione facciamo il complemento a 9 del sottraendo.*
2) *Alla somma dei due numeri **sommiamo 1** e scartiamo, dal risultato, il riporto*

7.7.4 Sottrazione con numeri scritti in complemento a 2 e ad 1

Facciamo ora le stesse operazioni con i corrispondenti numeri binari. Per motivi pratici scriviamo prima i numeri a cui siamo interessati sia in forma normale che in complemento a due:

$(32)_{10} = (010\ 0000)_2$; $(-32)_{10} = (110\ 0000)_2$; $(15)_{10} = (000\ 1111)_2$; $(-15)_{10} = (111\ 0001)_2$;

Notare che:

- Nella rappresentazione numerica dei positivi abbiamo posto uno zero in posizione più significativa (il numero è diventato di sette cifre).
- Il negativo del numero è stato ottenuto eseguendo il complemento a due del numero positivo (incluso il bit di segno = 0). Il numero 15 è stato scritto con un numero di cifre pari a quelle necessarie per scrivere 32, ponendo due 0 in posizione più significativa e abbiamo trovato la forma binaria, in complemento a due, di -15 usando la regola già data. Notare che, se non si conosce il numero di cifre con cui deve essere fatta la somma, si può scrivere il numero, in complemento a due, con il numero di cifre indispensabili alla sua rappresentazione, poi, al momento in cui si conosce il numero di cifre necessarie si esegue l'operazione di ***estensione del segno***. Questa consiste nel porre in posizione più significativa tante bit uguali a quello di segno quante ne sono necessarie. In altri termini se il bit di segno è 0 si premettono tanti 0 quanti necessari, se invece il bit di segno è a 1 si premettono tanti 1 quanti necessari.

Eseguendo le operazioni indicate otteniamo:

$32+ (-15) = (010\ 0000)_2 + (111\ 0001)_2 = ((1)001\ 0001)_2 => 17;$

$15+ (-32) = (000\ 1111)_2 + (110\ 0000)_2 = ((0)110\ 1111)_2 = (-)001\ 0001 => -17;$

Osservazione: fatta la somma,

- nel primo caso abbiamo avuto un riporto diverso da zero e (come prima) ciò significa che il risultato è positivo;
- nel secondo caso, abbiamo avuto un riporto pari a zero, e (come prima) ciò significa che il numero è negativo e scritto in complemento a due.

In conclusione possiamo dire che, ***usando i numeri in complemento a due e scartando, nel risultato dell'operazione, la cifra di riporto si ottiene, in ogni caso, il risultato corretto se interpretato in complemento a due***.

Riassumiamo nel seguente modo quanto osservato :

3) *Prima di eseguire la sottrazione fare il complemento a due del sottraendo.*
4) *Eseguire la somma e scartare, dal risultato, il riporto.*

Notare che allo stesso risultato si arriva se

3) Prima di eseguire la sottrazione facciamo il complemento a uno del sottraendo.
*4) Alla somma dei due numeri **sommiamo 1** e scartiamo, dal risultato, il riporto*

È opportuno evidenziare le seguenti ulteriori caratteristiche:

a) Quando si sommano due numeri in complemento a due *con segno diverso* non è possibile avere un overflow (cioè il numero di cifre da cui si è partiti è sufficiente a contenere il risultato).
b) Quando si sommano due numeri con lo stesso segno (entrambi positivi o negativi) è possibile avere un overflow.
c) L'overflow, se è presente, è segnalato dal fatto che il bit di segno del risultato è diverso dal comune bit di segno dei singoli addendi. Cioè se i due addendi erano negativi (bit di segno=1) il bit di segno del risultato è = 0, se i due addendi erano positivi (bit di segno=0) il bit di segno del risultato è = 1. In effetti la somma di numeri positivi non può essere negativa e viceversa.
d) Le Unità Logiche Aritmetiche (ALU), capaci di eseguire la somma di numeri in complemento a

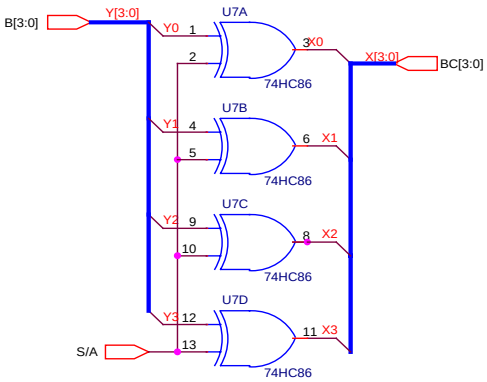


Fig. 7.13 True Complement

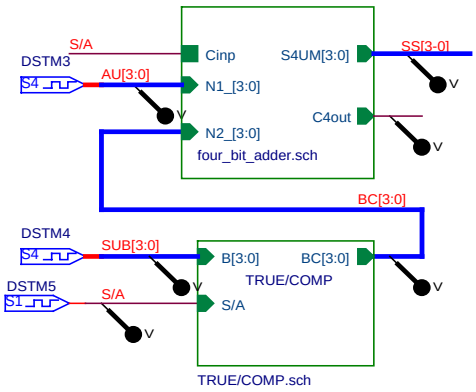


Fig 7.14

due, debbono avere un circuito rivelatore di Overflow che usa la seguente funzione booleana $E = S_aS_bS_r' + S_a'S_b'S_r$ (dove S_x indica il bit di segno del numero x "x = a per il primo addendo; x= b per il secondo addendo; x= r per il risultato").

e) L'ALU deve essere in grado di distinguere tra l'operazione di somma e di sottrazione. Per ciò è necessario l'uso di un bit di controllo. Questo bit, codificherà con 0 la somma e con 1 la sottrazione. Pertanto se il bit di controllo è zero NON bisogna fare la complementazione del secondo addendo, se è 1 bisogna fare la complementazione a due del secondo addendo.

f) Considerato che l'operazione di sottrazione trova, come vedremo, una realizzazione più semplice usando *il numero da sottrarre in complemento ad 1* e, considerato quanto definito nel punto e), possiamo usare il circuito presentato in Fig. 7.13 per realizzare o meno la complementazione ad uno del secondo addendo. Infatti, in quel circuito, la linea S/A (sottrazione o somma) è il bit di controllo di cui sopra e il dato di uscita è in complemento ad 1 se S/A=1 (sottrazione) o in forma normale se S/A=0 (somma).

g) Per quanto detto precedentemente usando, per l'operando da sottrarre, una rappresentazione in complemento ad uno bisogna aggiungere al risultato una unità. Ciò può essere facilmente fatto usando lo stesso bit di controllo S/A come ingresso carry al sommatore. Infatti se S/A è uguale a 1=> si deve eseguire la sottrazione e il carry è ad 1 se è S/A= 0 => si deve eseguire la somma e quindi il carry è a 0 come è necessario.

La Fig. 7.14 mostra la realizzazione completa di un full **subtractor,** che usa il Full Adder a 4 bit già discusso. Il timing di Fig. 7.15 mostra il risultato che si ottiene con l'uso di tale circuito. Fino al tempo pari a 80 μs si effettua la differenza (S/A = 1) dei due numeri AU e SUB, mentre successivamente (S/A = 0) si effettua la somma degli stessi due numeri.

Notare infine che quando il risultato della differenza è negativo il bit di CarryOut= 0, quando il risultato invece è positivo carry=1.

La traccia BC rappresenta il complemento ad 1 o meno di SUB.

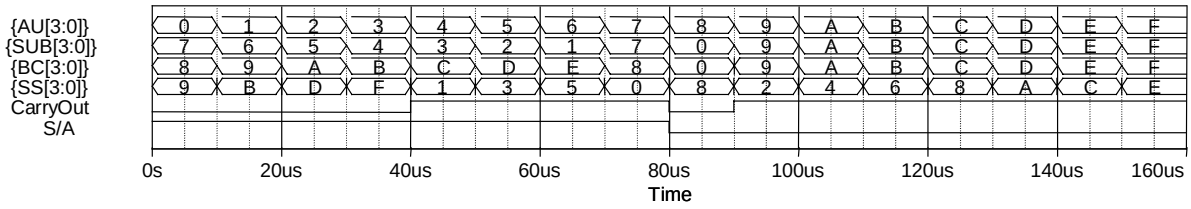


Fig 7.15

7.8 Somma e Sottrazione BCD

Nei computer, generalmente, sia la somma sia la sottrazione è fatta in forma binaria pura. In quasi tutti le calcolatrici elettroniche invece la somma è fatta in BCD. Questi circuiti non sono sostanzialmente diversi da quelli binari già studiati. Essi sommano e sottraggono utilizzando i circuiti già presentati, tuttavia, i dati debbono essere maneggiati in BCD invece che in binario.

Dec.	Gruppi di cifre BCD		
	Cent.	Diec.	Unità
386	0011	1000	0110
243	0010	0100	0011
629	0101	1100	1001

Tab 7.4

Prima di descrivere come è realizzato un sommatore BCD, esaminiamo cosa accade quando si sommano due numeri espressi in codice BCD. Nella Tab. 7.4 è mostrata la somma di due numeri sia in decimale che in BCD. Nella forma BCD la somma **separata** di ciascuna cifra ha come risultato rispettivamente i numeri 5, 12 e 9. È chiaro che il risultato non è corretto anche perché abbiamo, come cifra della somma delle diecine, un numero che non è una rappresentazione BCD valida. Dovremmo eseguire una correzione che, come risultato, ci deve fornire 2 per le diecine ed un riporto unitario per le centinaia. Quindi quando le cifre da sommare sono in BCD bisogna aggiungere dei circuiti ausiliari che siano in grado di rivelare il riporto e correggano la cifra che lo ha generato.

CD / AB	00	01	11	10
00				
01				
11	1	1	1	1
10			1	1

C1=AB+AC

Tab 7.5

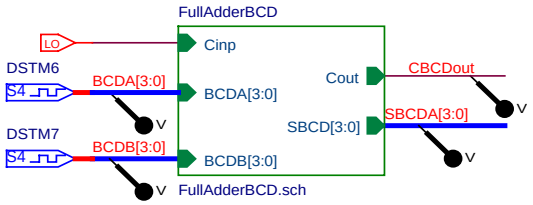


Fig. 7.16

Un riporto decimale deve essere generato quando il risultato della somma delle cifre BCD (indipendentemente dalla sua posizione decimale) è maggiore di 9. Notiamo che, operando in binario, un riporto decimale si ottiene o quando il risultato è minore di 16 cioè 10, 11, 12, 13, 14, 15 (configurazioni BCD non valide) oppure, se il risultato è maggiore di 15. In quest'ultimo caso otterremo un bit di carry binario ed una configurazione dei 4 bit meno significativi consentita dalla codifica BCD.

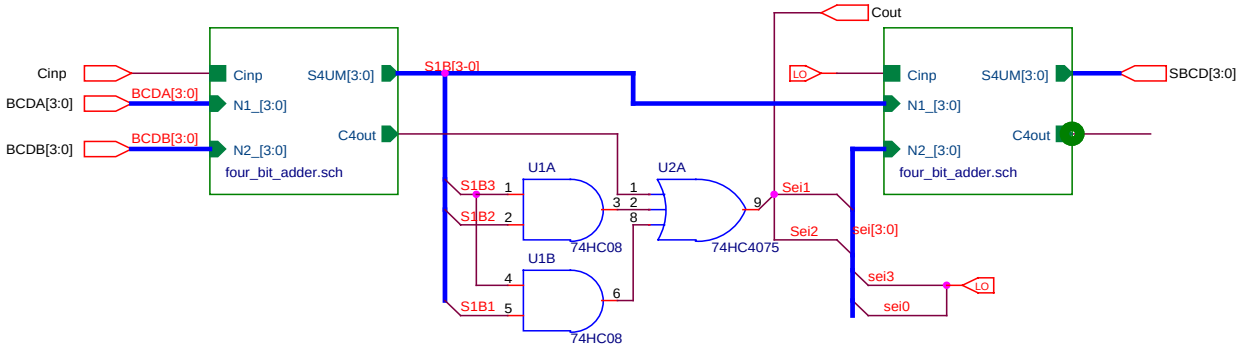


Fig. 7.17

Quindi un circuito che possa rivelare il carry decimale è una *OR del bit di carry binario e del rivelatore di configurazione non consentita di codice BCD.*

La mappa del rivelatore di configurazione non consentita è rappresentata nella Tab 7.5 assieme alla sua funzione Booleana minimizzata. Si noti che l'effettuazione della correzione del risultato, nella cifra che ha generato il carry decimale, si ottiene dalla somma di 6 al numero binario ottenuto (senza considerare l'eventuale carry binario). Nel nostro caso 1100+0110=(1)0010 e non considerando il carry binario: otteniamo 0010= 2 decimale.

Un circuito che somma una coppia generica di cifre BCD, capace di rivelare il riporto e correggere il risultato consta quindi di due sommatori binari a 4 bit e di un circuito rivelatore di carry decimale come mostrato in Fig. 7.17. Il primo sommatore fornisce la somma (da eventualmente correggere). Da questa si ricava il bit di carry decimale (porte U1A, U1B e U2A). Il secondo sommatore corregge il risultato del primo, sommando zero se non c'è carry decimale (nessuna correzione) o sommando 6 se c'è il carry decimale (correzione del somma ottenuta con il primo sommatore). Si

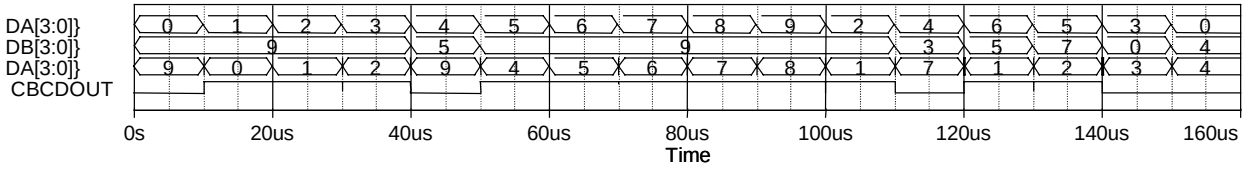


Fig. 7.18

noti che il numero 0 o 6 è ottenuto automaticamente utilizzando il bit carry decimale. Infatti, il secondo addendo del secondo sommatore è formato a partire dalla configurazione 0xx0 dove x = carry.

Il timing di Fig. 7.18 mostra il risultato attenuto con tale circuito. L'interpretazione corretta del risultato si trova sommando dieci al risultato se il bit di carry è ad 1.

Lo stadio di somma, ora descritto, può essere connesso in parallelo con molti stadi identici, per formare un sommatore con più cifre decimali. Ovviamente bisogna connettere il carry decimale di uscita dalla somma di una coppia di cifre decimali all'ingresso di carry della somma delle cifre decimali immediatamente più significative.

Cifra BCD				Complem. a 9			
A	B	C	D	a	b	c	d
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X
a=A'B'C'				b=BC'+B'C			
c= C				d= D'			

Tab. 7.6

7.8.1Sottrazione BCD

Questo stesso circuito può essere usato, utilizzando della logica ausiliaria, per fare la sottrazione di numeri codificati in BCD. Malgrado tale sottrazione possa essere effettuata in diversi modi noi illustreremo soltanto quella che usa il complemento a 9.

Ricordiamo che il complemento a 9 di un numero decimale si trova dal complemento a 9 di ciascuna cifra che lo compone. Come mostrato in

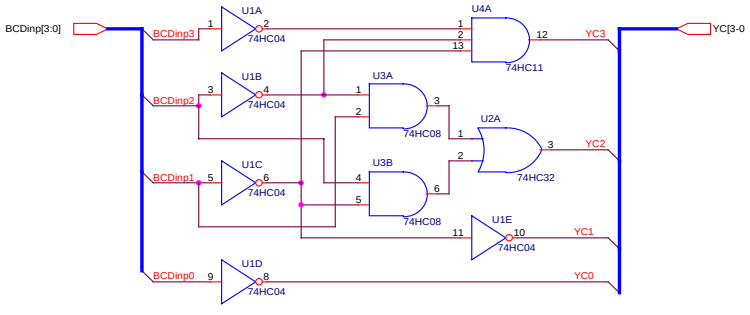


Fig 7.19

precedenza, la sottrazione può essere fatta facendo la somma tra il minuendo ed il sottraendo (scritto in complemento a 10) e trascurando il riporto. Se invece del complemento a 10 del numero si usa il complemento a 9, al risultato ottenuto, bisogna sommare ancora una unità. Quest'ultimo algoritmo è stato usato con il sottrattore binario.

Ricordiamo che la somma di questa unità era realizzata ponendo il bit di carry, della cifra meno significativa, ad 1 invece che a zero nel caso di sottrazione. Le funzioni Booleane per il circuito che fa il complemento a nove, di una cifra decimale, può essere ricavata dalla tabella di verità Tab.7.6, in cui le X, come al solito, indicano le condizioni di don't care.

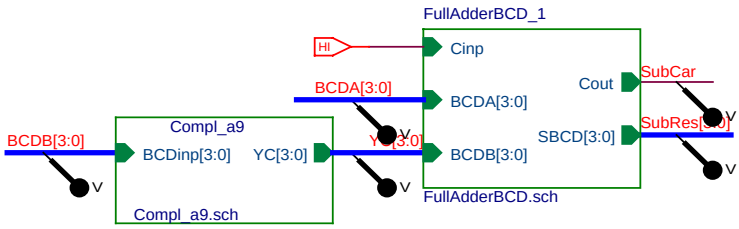


Fig. 7.20

Ai piedi della tabella sono state scritte le funzioni Booleane per ogni una delle corrispondenti variabili binarie che realizzano il complemento a 9. La Fig. 7.19 mostra la realizzazione del circuito di complemento a 9 di una cifra BCD.

La Fig. 7.20 mostra la realizzazione di un circuito sottrattore BCD utilizzando l'algoritmo sopra spiegato ed il circuito di complemento a 9 ora illustrato. Notare che il bit di carry è stato imposto ad

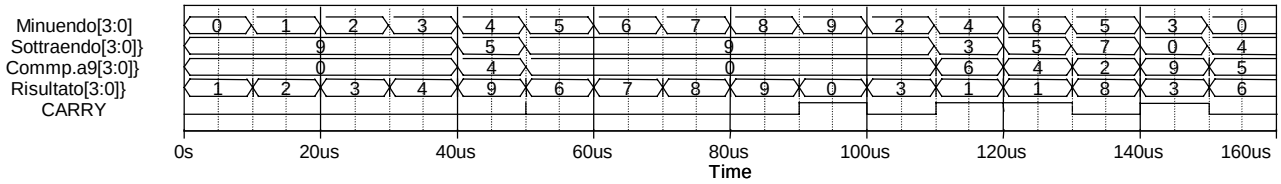


Fig 7.21

uno logico per sommare la prevista unità. Il timing di Fig. 7.21 dimostra i risultati che con esso si ottengono per la sottrazione BCD (una delle traccia mostra il risultato dell'operazione di complemento a 9). Notare che, quando il carry è 0 il risultato è negativo e quindi fa fatto il complemento a 10 del numero e premesso il segno meno per avere il risultato corretto in grandezza e segno.

7. 9 Somma Binaria Seriale

Per realizzare un sommatore seriale si utilizza un solo circuito full adder, una coppia di registri a scorrimento del tipo Parallel-In/Serial-Out (PISO), un registro Serial-In/Parallel-Out (SIPO) (che conterrà il risultato), un F/F D (che memorizzerà il carry), ed un contatore (per contare il numero di coppie di bit ancora da sommare). Poiché questi sono circuiti sequenziali, che saranno introdotti in un capitolo successivo, si rimanda la sua spiegazione al capitolo 11.

Problemi

P7.1 Ottenere il complemento a 1 ed il complemento a 2 dei seguenti numeri binari 1010101, 0111000, 000001, 10000, 000000

P7.2 Ottenere il complemento a 9 ed il complemento a 10 dei seguenti numeri decimali 13579, 09900, 90090, 10000, 000000

P7.3 Ottenere il complemento a 10 di $(935)_{11}$

P7.4 Fare la sottrazione con i seguenti numeri decimali usando, (1) il complemento a 10 e (2) il complemento a 9. Provare il risultato eseguendo le operazioni direttamente. (a) 5250-321; (b) 3570-2100; (c) 753-864; 20 – 1000.

P7.5 Progettare un circuito logico combinatorio che accetta un numero di tre bit in ingresso e genera, in uscita, un numero binario uguale al quadrato del numero in ingresso.

P7.6 È necessario moltiplicare due numeri binari, ciascuno formato da due bit, per ottenere il loro prodotto in binario. Supponendo che i due bit sono rappresentati come a_1,a_0 e b_1,b_0, dove il pedice 0 denota il bit meno significativo
a) determinare il numero di linee di uscite richieste
b) trovare le espressioni Booleane per ciascuna uscita.

P7.7 Ripeter il Problema P7.6 per formare la somma invece del prodotto dei due numeri binari.

P7.8 Progettare un circuito combinatorio il cui ingresso è un numero a 4 bit e la cui uscita è il complemento a due del numero di ingresso.

P7.9 Progettare un circuito combinatorio che moltiplica per 5 un numero decimale in ingresso rappresentato in BCD. L’uscita sia pure in BCD. Mostrare come l’uscita possa essere ottenuta senza l’uso di circuiti aritmetici.

P7.10 Progettare un Half-Subtractor considerando che il numero è scritto in segno e grandezza.

P7.11 Progettare un Full-Subtractor con due Half-Subtractor (trovato con la soluzione del Problema P7.10) e porte OR.

NLOGCAP8.DOC