di GIORGIO OBER (GIOBE2000)

l'angolo di Mr A.KEER

(parte diciassettesima)

PROGETTARE con le PORTE LOGICHE

Moltiplicatori

Dopo i sommatori e i sottrattori, dopo la sofisticata ALU, terminiamo con i moltiplicatori la rassegna delle macchine combinatorie a supporto delle operazioni aritmetiche.

Nelle puntate precedenti abbiamo potuto evidenziare l'importanza dei sommatori aritmetici, non solo al fine di assicurare (ovviamente) l'addizione degli operandi binari, ma anche per poter esercitare la loro eventuale differenza; vedremo ora che gli stessi componenti saranno coinvolti anche nel calcolo del prodotto.

Moltiplicatori "shift and add"

La moltiplicazione dei numeri (decimali) è sempre stata fatta (almeno fino all'avvento delle benedette/sciagurate calcolatrici tascabili) con quel particolare metodo che moltiplica il moltiplicando per ciascuna cifra del moltiplicatore (a cominciare da quella meno significativa) e poi somma i prodotti così ottenuti, dopo averli spostati di una posizione verso sinistra.

Lo stesso metodo è utilizzabile anche per la moltiplicazione di numeri binari positivi (o senza segno) a più bit, spesso definito "*repeated left-shift and add*" (sinteticamente "*shift and add*", per ribadire la sequenza di operazioni necessarie per ottenere il risultato; possiamo fissare le idee con un esempio concreto, supponendo di voler moltiplicare $A=(1010)_2=(10)_{10}$ per $B=(1101)_2=(13)_{10}$; la *Figura 1* riassume visivamente operandi e prodotto finale, $A*B=(10000010)_2=(130)_{10}$.

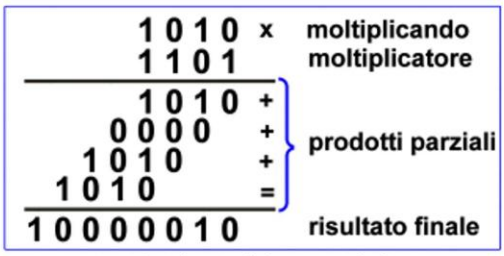

Figura 1 - 4-Bit by 4-bit Parallel Binary Multipliers: Esempio

Già osservando lo schema di questa operazione appare evidente la struttura del circuito necessario per realizzarla: bisogna anzitutto predisporre delle reti in grado fornire ciascuno dei

4 *prodotti parziali*, moltiplicando uno dopo l'altro ciascun bit del moltiplicatore con tutti quelli del moltiplicando.

E' importante sottolineare che questi prodotti aritmetici sono soggetti alle stesse regole del prodotto logico, per cui ciascuno di essi potrà essere realizzato con una **AND** a 2 ingressi; se ancora ce ne fosse bisogno, la *Figura 2* offre una possibile soluzione, mostrando il primo *prodotto parziale* relativo al moltiplicando per (=AND) il bit meno significativo del moltiplicatore.

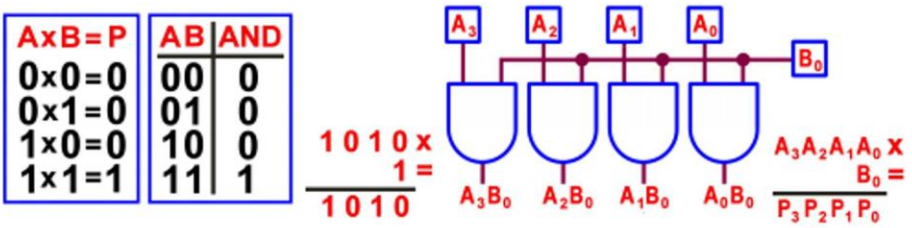

Figura 2 - 4-Bit by 4-bit Parallel Binary Multipliers: Primo prodotto parziale

La stessa batteria di 4 **AND** andrà bene anche per gli altri; possiamo notare che ciascuno di essi sarà uguale al moltiplicando (se il bit corrente B_n del moltiplicatore è uguale a 1) oppure ad una stringa di bit tutti a 0 (se bit B_n=0).

Per disporre del risultato finale bisogna ora operare la somma dei 4 *prodotti parziali*, non prima di aver spostato verso sinistra di un bit i 3 più significativi, in accordo con le regole dell'algoritmo; ricordiamo che operare uno *shift a sinistra* di un numero binario equivale in moltiplicarlo per 2.

La *Figura 3* mostra gli elementi necessari e sufficienti per operare la somma dei primi due prodotti parziali: bastano 2 **HA** e 2 **FA**; da notare che il prodotto (**AND**) dei bit meno significativi di entrambi gli operandi non è coinvolto nella somma e diventa immediatamente il bit meno significativo del risultato.

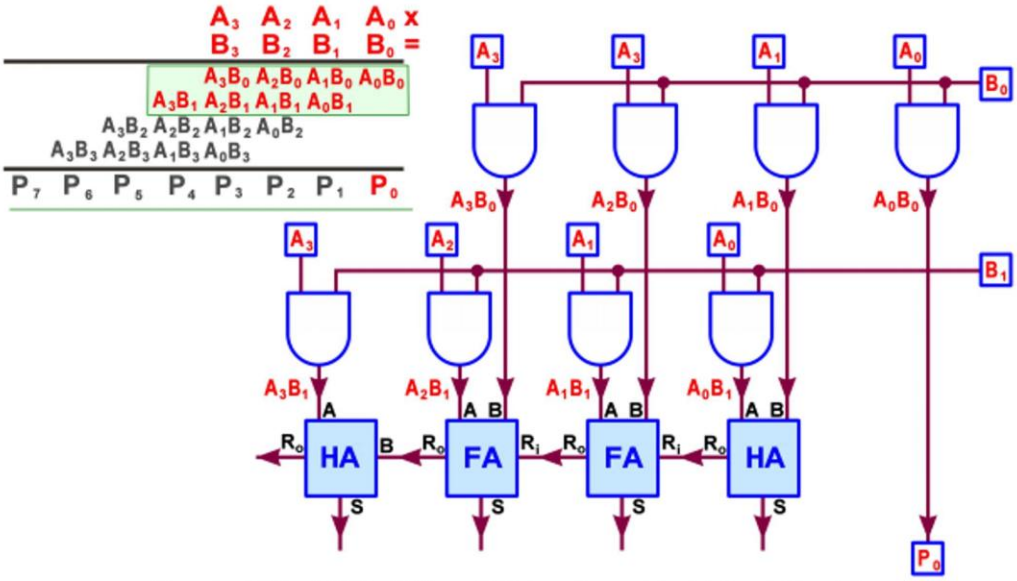

Figura 3 - 4-Bit by 4-bit Parallel Binary Multipliers: Somma dei primi 2 prodotti parziali

Dopo l'aggiunta dei 2 rimanenti prodotti parziali il circuito finale (vedi *Figura 4*) sarà realizzato con 16 porte **AND** a 2 ingressi, 4 sommatori **HA** e 8 sommatori **FA**; naturalmente gli **HA**

possono essere sostituiti da **FA** forzando a massa il rispettivo ingresso di *Carry in*; per la sua struttura questo circuito è detto anche "moltiplicatore a matrice" (*Array Multilpier*).

Tutti i *prodotti parziali* sono calcolati contemporaneamente, ma per poter disporre di un risultato stabile bisogna attendere la propagazione di tutti i riporti (*ripple carry*) e in particolare quelli che dal secondo *prodotto parziale* va al terzo e dal terzo va al quarto; da notare che il bit più significativo del risultato è uguale al *Carry Out* del **FA** più significativo dell'ultimo *prodotto parziale*.

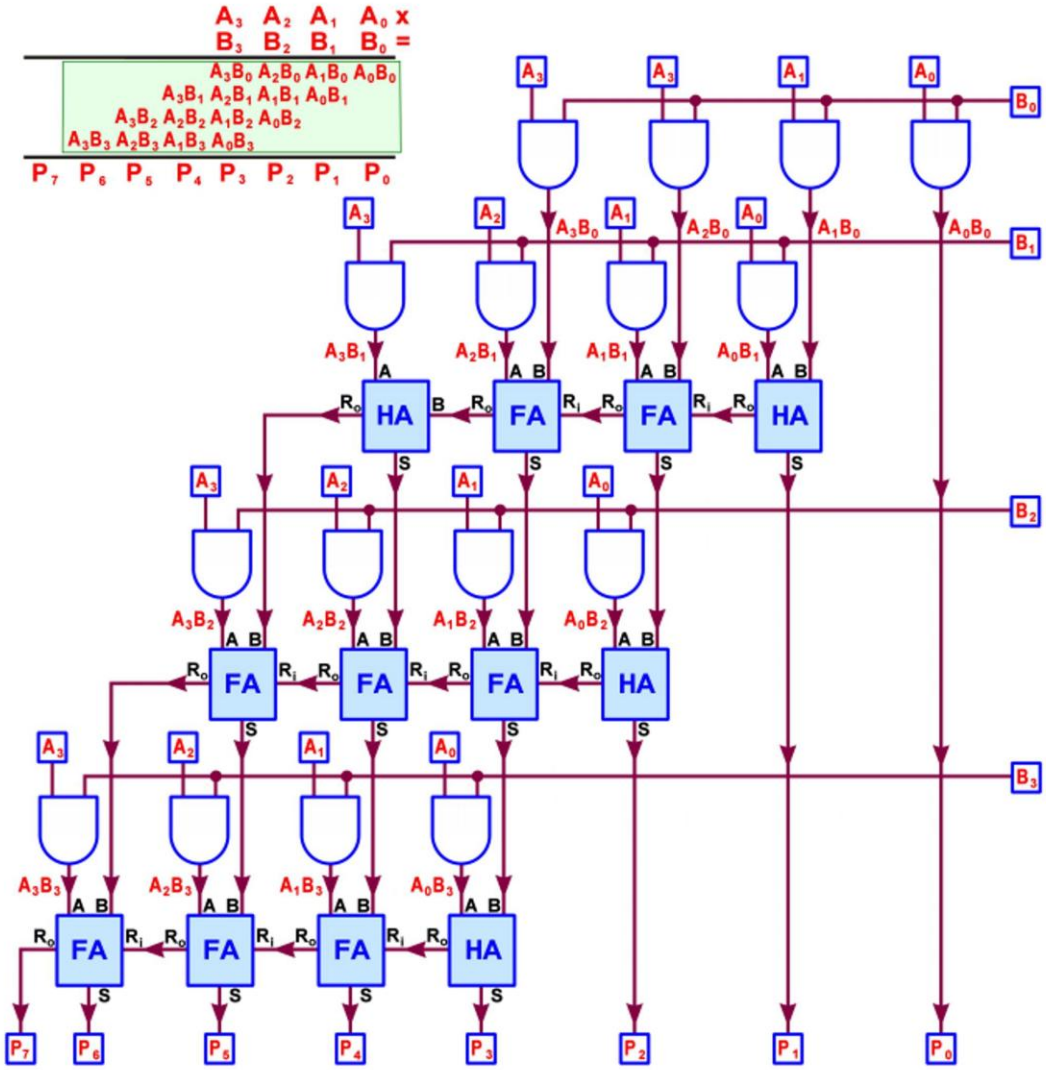

Figura 4 - 4-Bit by 4-bit Parallel Binary Multipliers: Circuito completo

Seguendo il percorso dei sommatori elementari che producono i bit del risultato è facile verificare che il ritardo massimo è pari a 8 volte quello di ciascuno di loro: con entrambi gli operandi di n bit il ritardo è pari a **2*n**; possiamo notare che il prodotto avrà sempre n*m bit, con n e m pari al numero di bit rispettivamente del moltiplicando e del moltiplicatore.

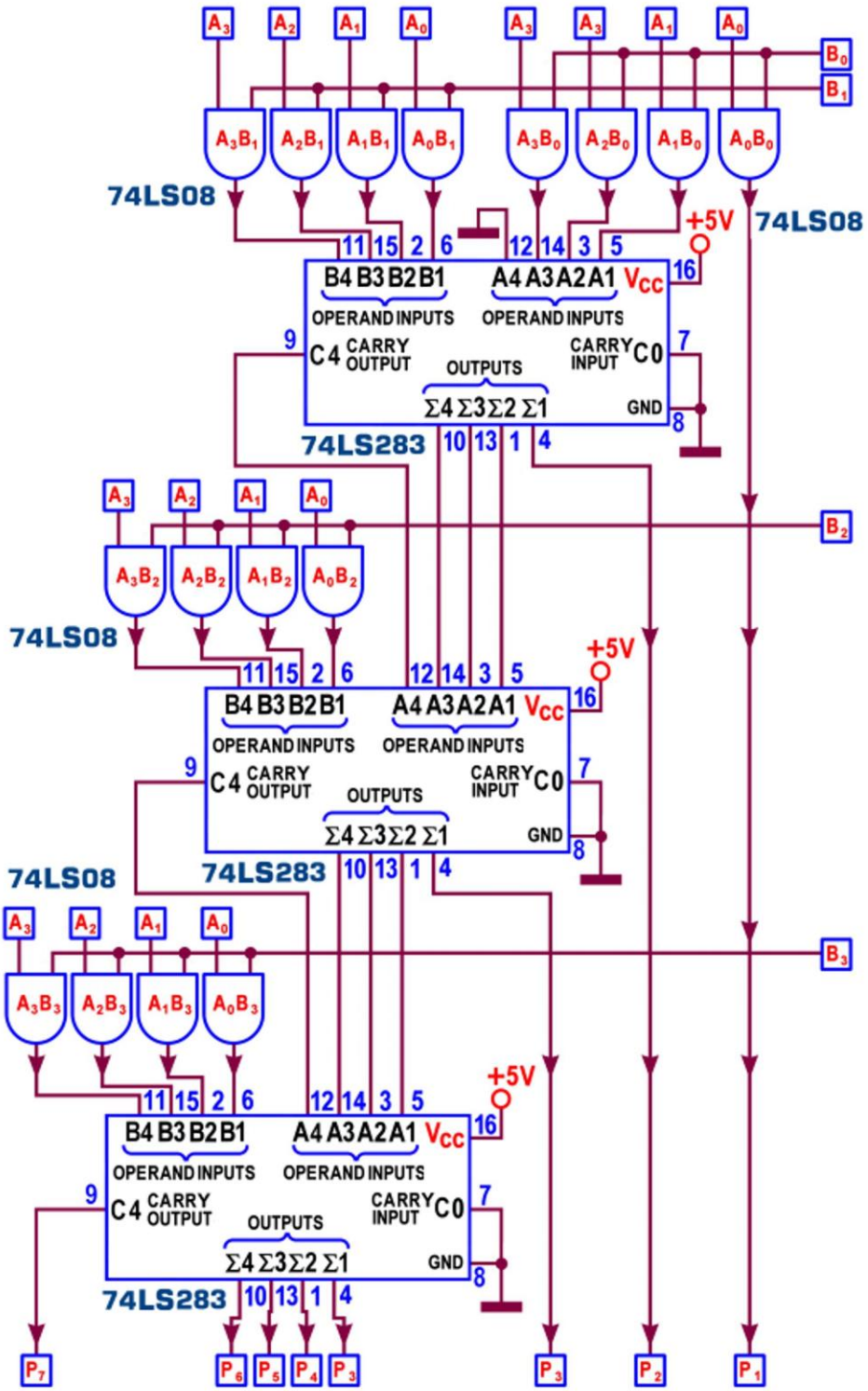

Figura 5 - 4-Bit by 4-bit Parallel Binary Multipliers con Fast Carry Adder 74LS283

Va ribadito anche che, per garantire un risultato corretto, il progetto appena realizzato dà per scontato che sia gli operandi che il prodotto finale siano da intendere "senza segno" o intrinsecamente positivi (cioè espressi come il rispettivo modulo).

La *Figura 5* mostra un moltiplicatore a matrice realizzato con 3 sommatori integrati (nella fattispecie 3 *Fast Carry Adder* **74LS283**, funzionalmente uguale al **74LS83**, corredato da logiche interne in grado di assicurare tecniche di tipo ***Carry Look Ahead***) e 4 integrati **74LS08** (ciascuno con 4 **AND** a 2 ingressi).

Questo progetto si presta a gestire anche operandi di dimensione più piccola; la *Figura 6* suggerisce la soluzione per un moltiplicatore da 2 per 2 bit.

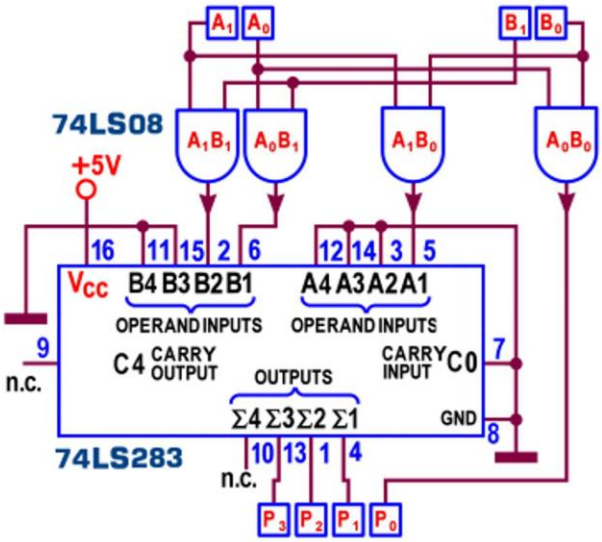

Figura 6 - 2-Bit by 2-bit Parallel Binary Multipliers con Fast Carry Adder 74LS283

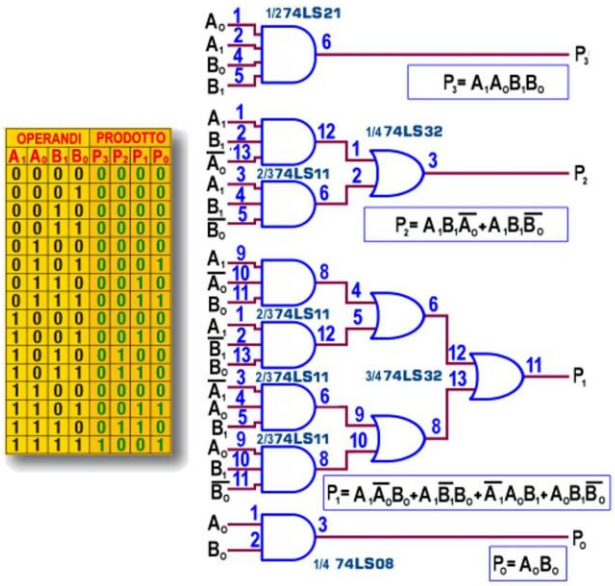

Figura 7 - 2-Bit by 2-bit Parallel Binary Multipliers con porte logiche [1]

A questo proposito è interessante osservare che è possibile progettare un moltiplicatore anche facendo riferimento alle tecniche di minimizzazione di base, con le Mappe di Karnaugh; è facile capire che tale tecnica sarà possibile solo con operandi A e B di piccola dimensione: già se ciascuno di essi ha solo 2 bit (come nel progetto appena proposto) sarà necessaria una Tabella di verità composta da 16 righe, tante quante sono i possibili prodotti A*B; la *Figura 7* mostra la rete combinatoria frutto della minimizzazione, con le relative formule di progetto.

La rete combinatoria si avvale di 3 integrati usati completamente (un **74LS32** [**OR**] e due **74LS11** [**AND** a 3 ingressi]) e 3 integrati usati parzialmente (un **74LS21** [**AND** a 4 ingressi], un **74LS08** [**AND** a 2 ingressi] e un **74LS04** [**inverter**]); applicando il teorema di *De Morgan* alle formule di progetto è possibile risparmiare un integrato, riducendo il progetto a un **74LS20** [**NAND** a 4 ingressi], due **74LS10** [**NAND** a 3 ingressi]) un **74LS00** [**NAND** a 2 ingressi], più l'inverter **74LS04**).

Ma il circuito combinatorio più vantaggioso si ottiene facendo riferimento alla struttura "*shift and add*", oggetto della nostra analisi: alle 4 **AND**, necessarie per disporre dei *prodotti parziali*, si devono aggiungere un solo **HA** e un solo **FA**, per cui, sostituendoli con le rispettive porte logiche, otteniamo una rete realizzata con appena 4 integrati, come si vede in *Figura 8*.

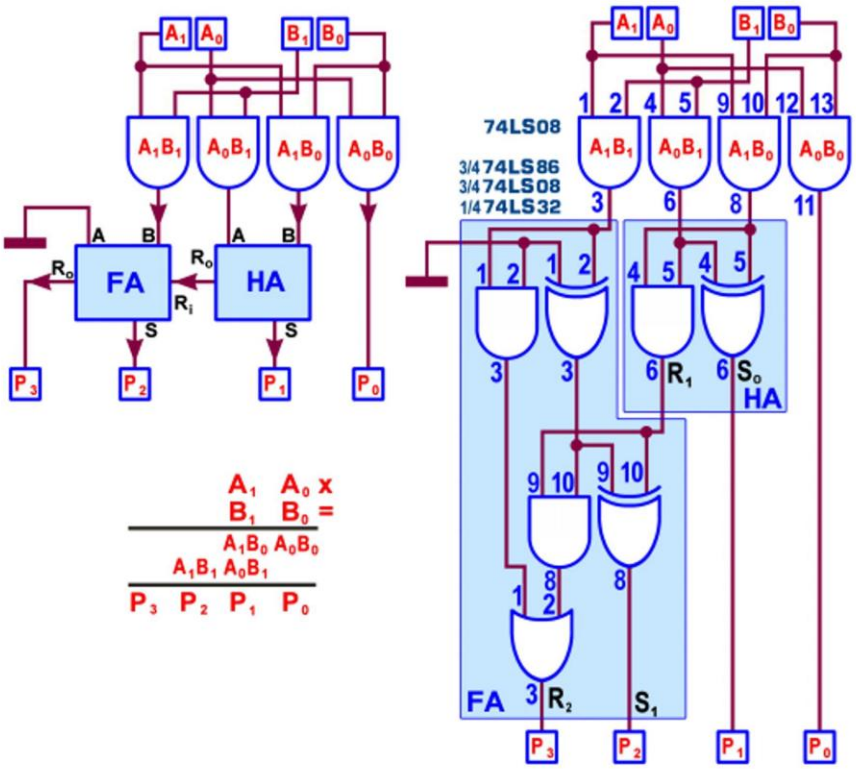

Figura 8 - 2-Bit by 2-bit Parallel Binary Multipliers con porte logiche [2]

Inoltre, poichè il **FA** previsto dalla struttura non sfrutta uno dei suoi ingressi dato, è possibile (solo in questa applicazione, vedi *Figura 9*) sostituirlo con un **HA**, fissando le esigenze di progetto a soli 3 integrati: due **74LS08** [**AND** a 2 ingressi] e un **74LS86** [**OREX** a 2 ingressi], di cui 2 usati a metà.

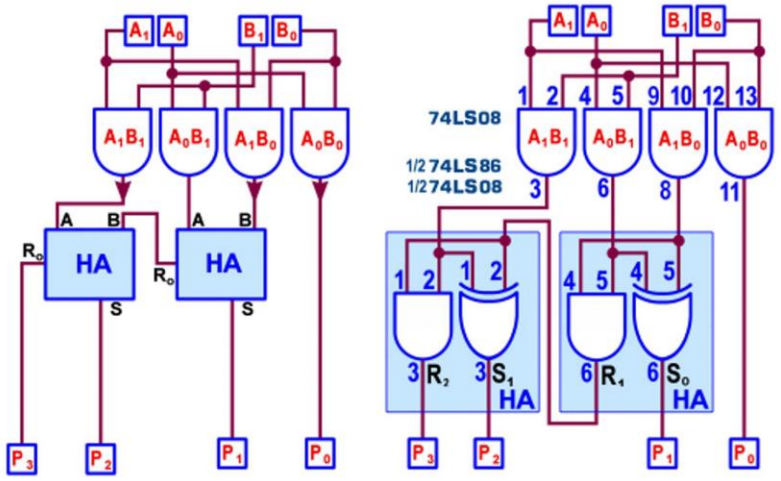

Figura 9 - 2-Bit by 2-bit Parallel Binary Multipliers con porte logiche [3]

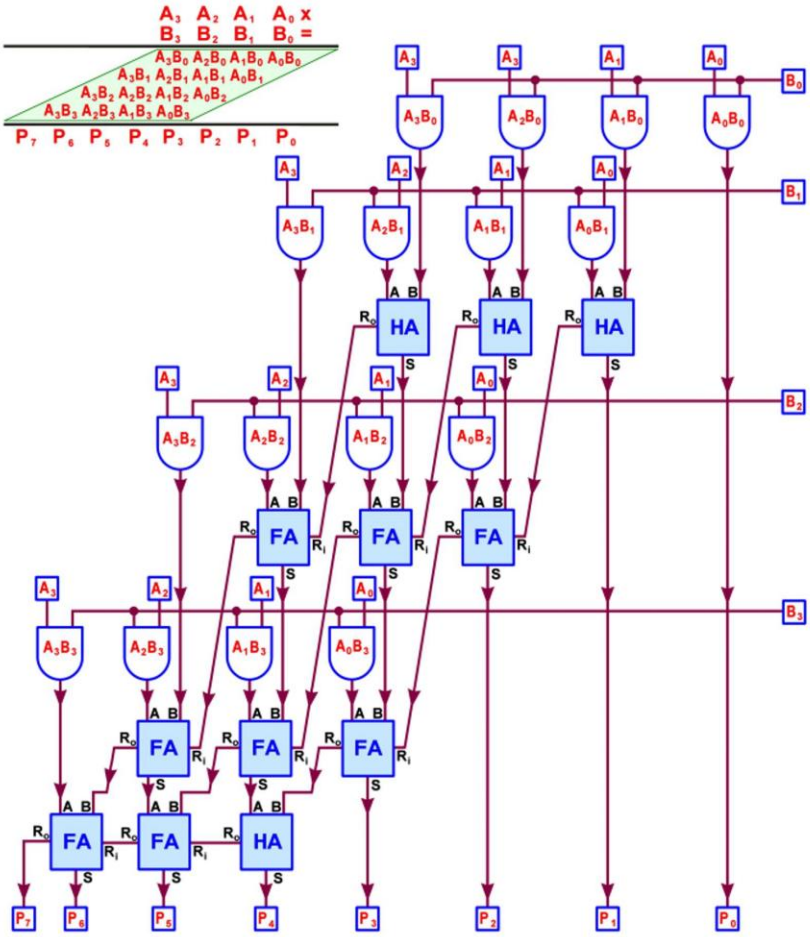

Figura 10 - 4-Bit by 4-bit Parallel Carry Save Multiplier

Un buon "moltiplicatore a matrice" (*array multilpier*) per operandi intesi senza segno può essere ottenuto da quello appena descritto partendo dal fatto che *il prodotto finale non cambia* se il riporto dei singoli sommatori elementari (**HA** o **FA**) viene propagato **in diagonale** anziché verso sinistra.

La variante così ottenuta è detta "***Carry Save Multiplier***", per mettere in evidenza l'azione di salvataggio (memorizzazione) esercitata sui riporti, al fine di poterli utilizzare negli stadi di somma successivi, fino a convogliarli in un "sommatore di raccolta" finale ("*Vector Merging Adder*") che, per l'importante compito a cui è chiamato, dovrebbe essere ottimizzato al meglio e di tipo *carry-look-ahead*.

Sebbene sostanzialmente non cambi nulla (la struttura ***carry-save***, *Figura 10*, è sempre realizzata con 16 porte **AND** a 2 ingressi, 4 sommatori **HA** e 8 sommatori **FA**) anche in questo caso per poter disporre di un risultato stabile bisogna attendere la propagazione di tutti i riporti; seguendo il percorso dei sommatori elementari che producono i bit del risultato è facile verificare che il ritardo massimo è ora pari a 6 volte quello di ciascuno di loro; con entrambi gli operandi di n bit il ritardo è pari a **(2*n)-2** per cui questa scelta è più efficiente rispetto alla precedente e sarà sempre più vantaggiosa al crescere del numero di bit degli operandi.

In ogni caso, data la rilevante quantità di elementi logici coinvolti (anche in prospettiva di applicare le stesse strutture a numeri di maggiore dimensione) la realizzazione dei moltiplicatori di numeri binari positivi può essere affidata a dispositivi come i **PLA** (Programmable Logic Array) oppure raccogliendo i possibili valori del prodotto dei 2 numeri in Tabelle appositamente approntate in **ROM** (Read Only Memory, memorie a sola lettura); questa tecnica è usata di frequente nell'ambito delle strutture controllate da microprocessori (come computer o microcontrollori) fornendo loro la possibilità di disporre in tempo reale di un dato binario anche composto da molti bit, non solo associato al prodotto degli operandi ma anche al risultato di operazioni matematiche di ogni tipo.

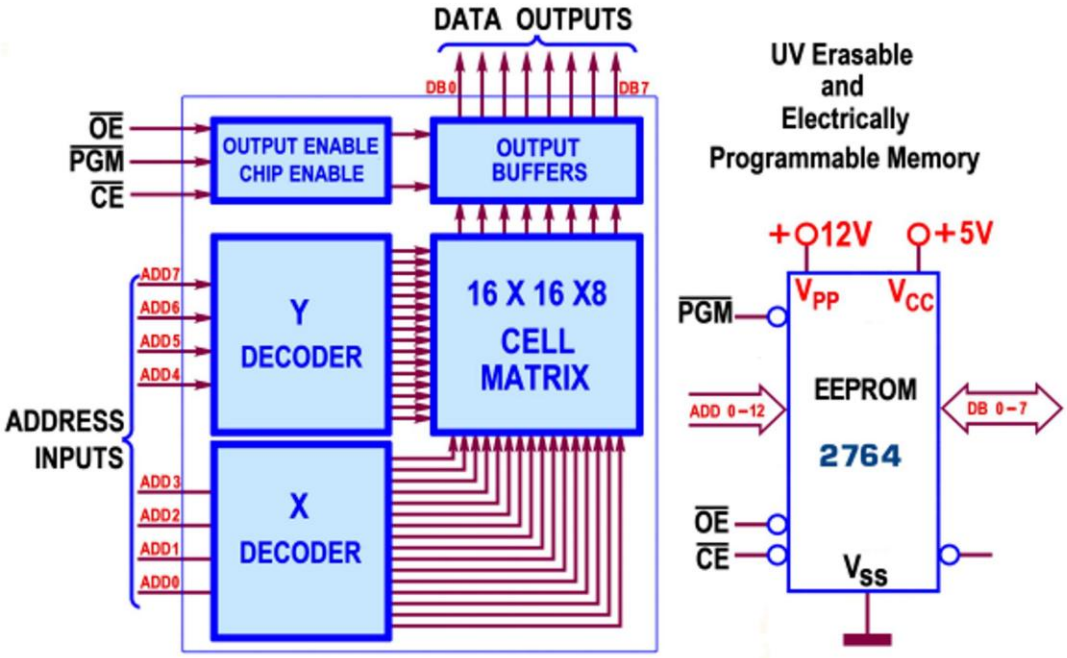

Figura 11 - 4-Bit by 4-bit Binary Multipliers con Memoria ROM

Per rimanere nel campo della moltiplicazione di 2 numeri ciascuno a 4 bit è facile capire che le due quaterne possono essere unite per formare un indirizzo a 8 bit, per esempio associando al moltiplicando A i 4 bit più significativi e al moltiplicatore B i 4 bit meno significativi; con un

indirizzo a 8 bit si possono puntare (in lettura) 2^8=256 locazioni, nelle quali è stato memorizzato (nella precedente fase di scrittura) il valore dei singoli prodotti A*B, a loro volta mai più grande di 8 bit.

Per esempio, nell'ipotesi di disporre di una ROM da 256 bytes, indirizzandola con 11111111 [rappresentativo di A=$(1111)_2$=$(15)_{10}$ e B=$(1111)_2$=$(15)_{10}$] si potrà disporre del risultato [P=$(11100001)_2$=$(225)_{10}$] in un tempo molto piccolo, pari al tempo di accesso della memoria ROM (qualche decina di ns, anche dovendosi riferire a più moderne Flash EPROM, per altro disponibili con capacità ben più grande dei 256 bytes necessari al nostro esempio, in grado di gestire la moltiplicazione di parole più grandi (per esempio a 8 bit, con risultati a 16 bit distribuiti su 2 memorie).

La *Figura 11* schematizza la struttura interna di una ROM da 256 locazioni ciascuna contenente un byte: se il dispositivo (*Chip*) di memoria è abilitato (*Enable*), cioè se CE =0, le 2 parole a 4 bit fornite sugli *Address Inputs* sono internamente decodificate per identificare la riga Y e la colonna X che puntano l'unica locazione ad esse corrispondente, tra le 16x16 possibili; la logica di controllo consente la disponibilità degli 8 bit in essa contenuti non appena la linea di *Out Enable* è attivata (bassa, OE=0).

La stessa *Figura 11* mostra un possibile moderno dispositivo di questo tipo, programmabile elettricamente (con l'aiuto di una apposita tensione di programmazione $\mathbf{V_{PP}}$) e cancellabile esponendo l'integrato ai raggi ultravioletti, da fornire attraverso una piccola finestrella circolare presente nel suo corpo; si tratta della **EEPROM** CMOS **2764** organizzata per garantire 8192 parole da 8 bit (8kBytes = 64kBit) localizzate con un indirizzo a 13 bit (2^{13}=8192), in un tempo (di accesso massimo) di 180 ns; inutile sottolineare che per il nostro esempio è ampiamente sovradimensionata.

Moltiplicatori "add and shift"

In questa trattazione non può mancare un riferimento ai processori: come per altre operazioni aritmetiche esistono precise e potenti istruzioni in grado di trattare e risolvere la moltiplicazione binaria, basate su circuiti molto diversi da quelli strettamente combinatori, trattati finora; il metodo utilizzato dalla CPU è detto "*repeated add and right-shift*" (sinteticamente "*add and shift*") ed è relativamente più conveniente di quello ("*shift and add*") descritto in precedenza.

La struttura che stiamo per analizzare è decisamente sequenziale; la CPU memorizza infatti gli operandi e il risultato in alcuni suoi *registri* (veloci memorie ospitate direttamente dentro il processore stesso) coinvolgendo la sua **ALU** per operare la somma richiesta (come vedremo) dal procedimento; è sottointeso che, per cadenzare le varie fasi di calcolo, sarà necessario un segnale di sincronismo (clock).

La *Figura 12* mostra lo schema di un moltiplicatore sequenziale limitato (per semplicità) ad operandi di 4 bit, fermo restando che la capacità di calcolo reale può essere assicurata anche per operandi di dimensione molto maggiore; essa è divisa in 4 parti, per tentare di evidenziare l'***effetto tempo***, tipico di una serie di eventi che si evolvono in sequenza.

Il registro del moltiplicando [nell'esempio pari a A=$(0110)_2$=$(6)_{10}$] mantiene inalterato il suo valore iniziale, per tutta la procedura; il moltiplicatore [supposto uguale a B=$(0101)_2$=$(5)_{10}$] è affidato ad un registro a scorrimento (*shift register*): il suo bit meno significativo viene utilizzato per controllare una batteria di **AND** che lascia passare il valore del moltiplicando (se vale 1) o una serie di zero (se vale 0).

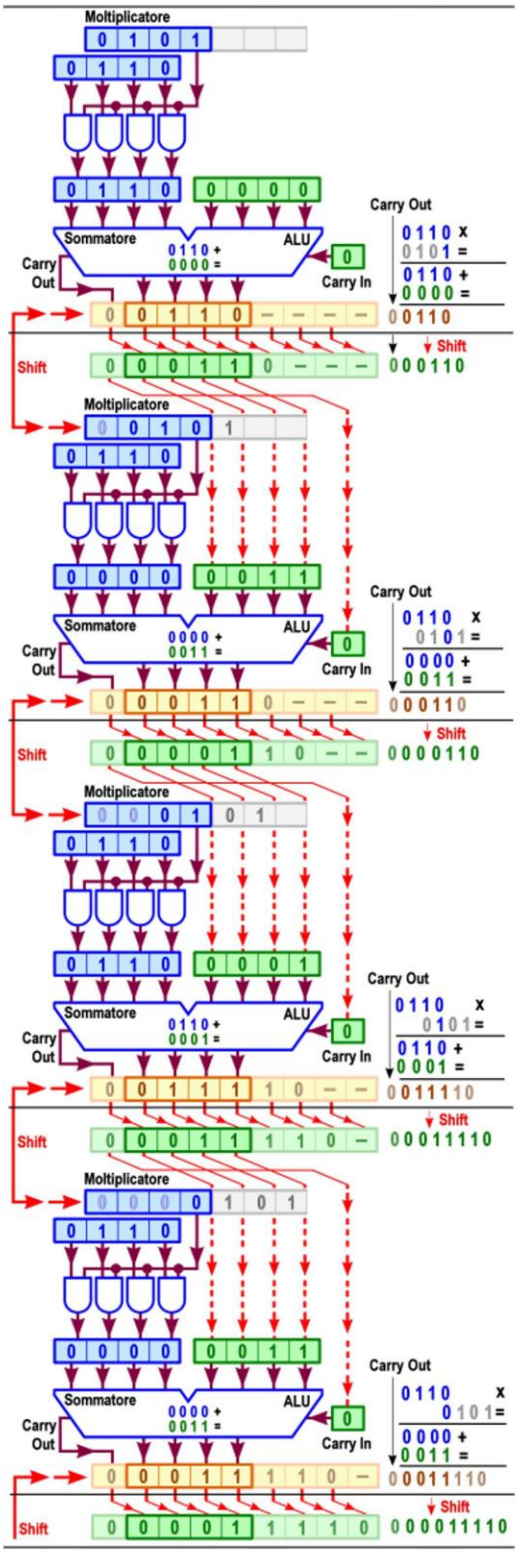

Figura 12 - 4-Bit by 4-bit Binary Multipliers: struttura sequenziale

Viene così generato il *prodotto parziale* utilizzato come primo addendo dal sommatore assicurato dalla ALU del processore, mentre il secondo addendo è fornito dal valore corrente di un registro accumulatore, azzerato all'inizio e chiamato ad ospitare il risultato progressivo, man mano che la procedura si sviluppa; da notare che la somma accumulata terrà conto anche dell'eventuale riporto prodotto dal sommatore (esso pure azzerato, prima di cominciare la sequenza).

Non appena la somma è disponibile ha inizio la fase successiva, all'inizio della quale sia il moltiplicatore che l'accumulatore (con l'ultimo risultato calcolato) sono sottoposti ad uno spostamento (*shift*) di una posizione a destra, di solito introducendo un bit a 0 da sinistra; vengono così stabiliti gli addendi della prossima somma, seguendo il criterio descritto per la prima fase e ben illustrato dalla preziosa figura.

Il processo avrà fine quando saranno stati utilizzati tutti i bit del moltiplicatore; nel nostro caso saranno operate 4 fasi, al termine delle quali il registro del moltiplicatore sarà azzerato e il registro accumulatore conterrà il risultato di moltiplicazione: verificandolo dalla *Figura 12* esso è effettivamente pari a $(00011110)_2$, formattato su 8 bit (tanti quanti sono previsti per operandi a 4 bit), uguale a $(30)_{10}=(6*5)_{10}$.

Se il processore non fosse dotato dell'hardware "*add and shift*" appena descritto sarebbe comunque in grado di simularne l'esecuzione da software, cioè eseguendo le medesime operazioni di somma e di shift con l'aiuto delle rispettive istruzioni, sempre disponibili; di certo una CPU priva di questa struttura è meno complessa e più economica ma l'esecuzione del programma richiederebbe tempi decisamente più elevati rispetto a quelli assicurati dall'hardware.

Moltiplicatori per numeri con segno

I moltiplicatori descritti finora sono piccoli gioielli di logica applicata ma sono in grado di operare esclusivamente numeri positivi (cioè la sequenza dei loro bit è da ritenersi "valore assoluto" di numeri davanti ai quali si suppone implicitamente anteposto uno o più bit a 0, intesi come segno "+").

Naturalmente è sempre possibile manipolare eventuali operandi negativi in modo da esprimerli in "modulo e segno", coinvolgendo nel prodotto il loro valore assoluto e riservandosi di adeguare poi il segno del risultato in base alle note regole algebriche (positivo con operandi di segno uguale, negativo con operandi di segno diverso); di certo le modifiche necessarie per convertire in positivi i numeri negativi (sia operandi che risultato, di solito espressi in complemento a 2) comporterebbero un'aggiunta di dispositivi che rendono improbabile perseguire tale via.

Tuttavia, con un semplice artificio, l'algoritmo classico della moltiplicazione può essere reso funzionante anche con numeri negativi; partiamo dal presupposto che, senza nessuna presa di posizione, gli 1 posti nella posizione più significativa di un numero non sono necessariamente da intendersi come bit di segno negativo; a sostegno di questa ipotesi riconsideriamo l'esempio di *Figura 1*, nel quale abbiamo supposto di trattare i numeri positivi $(1010)_2=(10)_{10}$ e $(1101)_2=(13)_{10}$ ottenendone il prodotto $(10000010)_2=(130)_{10}$; ma che succede se i numeri coinvolti sono ritenuti con segno? Gli operandi sono entrambi negativi, $(1010)_{Ca2}=(-6)_{10}$ e $(1101)_{Ca2}=(-3)_{10}$, e il loro prodotto dovrebbe dare $(-6*-3)_{10}=(+18)_{10}=(010010)_2$, cosa evidentemente non verificata.

Ragionandoci, risulta evidente che davanti a ciascuno dei 4 prodotti parziali si da per scontata la presenza di 0, anche se non esplicitamente scritta; questo fatto rende erroneamente "positivi" i numeri negativi coinvolti e fa pensare che se, invece, si estende verso sinistra il loro segno, in quantità sufficiente a coprire la dimensione prevista per il risultato, le cose possono tornare a posto!

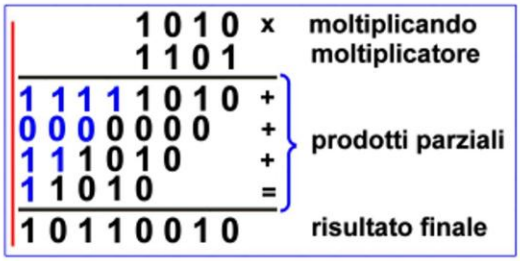

Figura 13 - 4-Bit by 4-bit Binary Multipliers: con numeri negativi [1]

La *Figura 13* riprende l'esercizio alla luce di queste considerazioni ma, nonostante il fatto di aver applicato la regola, il risultato non solo non è quello atteso ma è addirittura negativo: $(10110010)_{Ca2}=(-78)_{10}$; in realtà per ottenere un risultato corretto non basta estendere il segno dei *prodotti parziali* ma, se il moltiplicatore è negativo, è necessario farlo anche per il suo segno, sempre in quantità tale da raggiungere le cifre del risultato, in modo da produrre altrettanti prodotti parziali.

La nuova situazione è illustrata in *Figura 14*, dalla quale si riscontra che il risultato è ora quello atteso, appunto $(+18)_{10}=(010010)_2$; in essa è anche evidenziato (con una riga rossa verticale) che ogni riporto della somma (ed ogni bit eccedente la lunghezza prevista per il risultato) non è significativo e non va preso in considerazione.

Come si può facilmente vedere, confrontando le due figure, questa ulteriore specifica ci costringe a generare un numero doppio di *prodotti parziali*, richiedendo più circuiti combinatori e allungando i tempi necessari per disporre della soluzione, già intrinsecamente lenti.

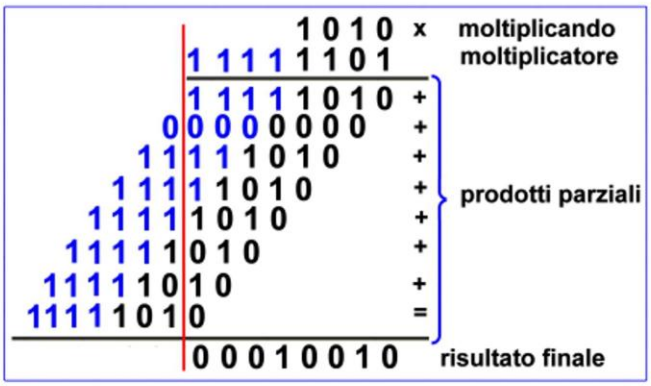

Figura 14 - 4-Bit by 4-bit Binary Multipliers: con numeri negativi [2]

Per risolvere questo problema (anche in presenza di operandi negativi) sono state studiate delle sofisticate soluzioni in grado di aumentare drasticamente le prestazioni; una di esse è nota come ***moltiplicatore di Booth***: il suo algoritmo analizza il moltiplicatore "ricodificandolo" al fine ridurre il numero di somme consecutive.

Altre importanti soluzioni sono il ***moltiplicatore*** ad albero ***di Wallace*** (*Wallace Tree*) e il ***moltiplicatore di Dadda***; ciascuna di esse è qui citata per amore di completezza, essendo obiettivamente insostenibile ogni tentativo di descrivere con sufficiente gratificazione le complicate regole che le governano.

Moltiplicatori (TTL): 74LS284, 74LS285, 74LS261, 74LS274

La serie **TTL** prevede la coppia di integrati **74LS284** con il **74LS285**, detti **4-Bit by 4-bit Parallel Binary Multipliers**, progettati per essere utilizzati in applicazioni di moltiplicazione parallela ad alte prestazioni di due numeri binari a 4 bit, ma facilmente espandibile anche per realizzare il prodotto di operandi con dimensioni grandi a piacere (da 8, 16, 32, .., bit).

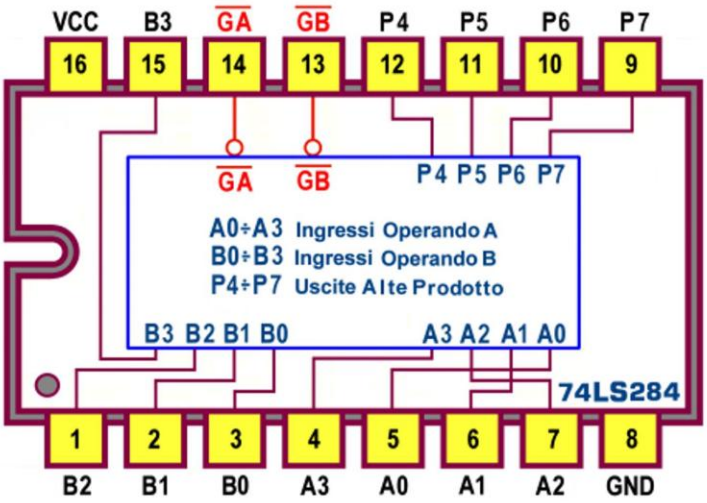

Figura 15 - 4-Bit by 4-bit Parallel Binary Multipliers 74LS284: pin-out

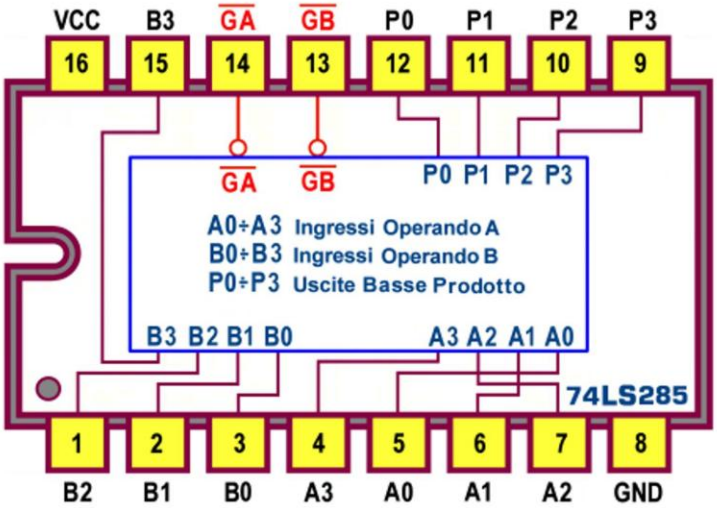

Figura 16 - 4-Bit by 4-bit Parallel Binary Multipliers 74LS285: pin-out

I rispettivi *pin-out* sono visibili in *Figura 15* e in *Figura 16*; il primo si occupa dei 4 bit di ordine più alto del prodotto, mentre il secondo implementa i 4 bit meno significativi.

La *Figura 17* riorganizza i segnali coinvolti nei corrispondenti schemi funzionali; possiamo notare che entrambi permettono di controllare lo stato delle uscite con una coppia di segnali d'abilitazione attivi bassi, G_A e G_B: le uscite sono valide quando sono entrambi sono a 0, altrimenti sono forzate nello stato di alta impedenza.

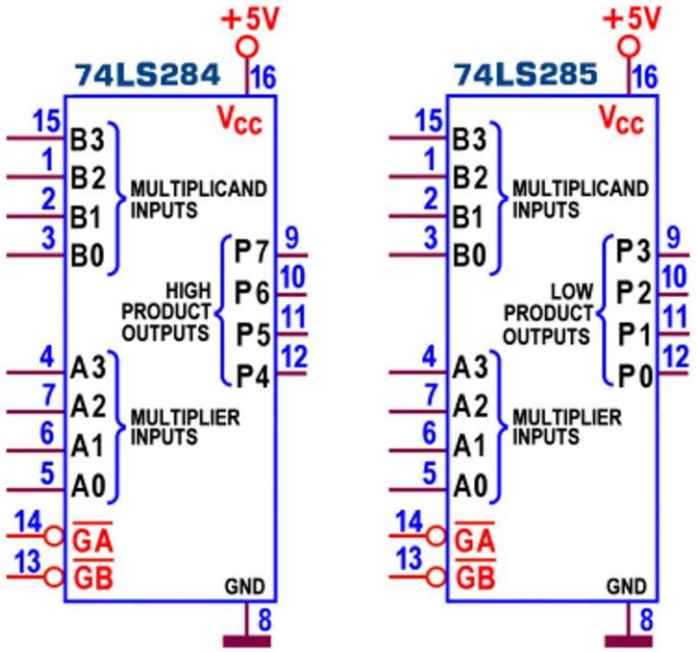

Figura 17 - 4-Bit by 4-bit Parallel Binary Multipliers 74LS284/74LS285: Schema funzionale

Il *simbolo logic*o predisposto dallo *standard IEEE* è visibile, per entrambi, in *Figura 18*.

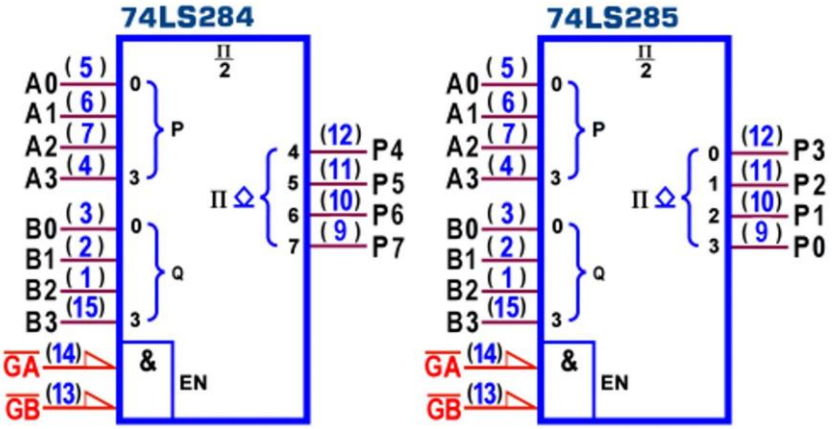

Figura 18 - 4-Bit by 4-bit Parallel Binary Multipliers 74LS284/74LS285: Simbolo logico ANSI/IEEE Std. 91-1984

La *Figura 19* mostra i collegamenti necessari per assicurare la moltiplicazione aritmetica di due parole binarie positive (o senza segno) di 4 bit; come abbiamo già sottolineato, in presenza di operandi negativi è necessario prima convertirli nel corrispondente numero positivo, a partire dal complemento a 2 che li esprime; se gli operandi hanno segno diverso tra loro anche il valore assoluto del risultato dovrà essere cambiato nel suo complemento a 2, essendo certamente negativo.

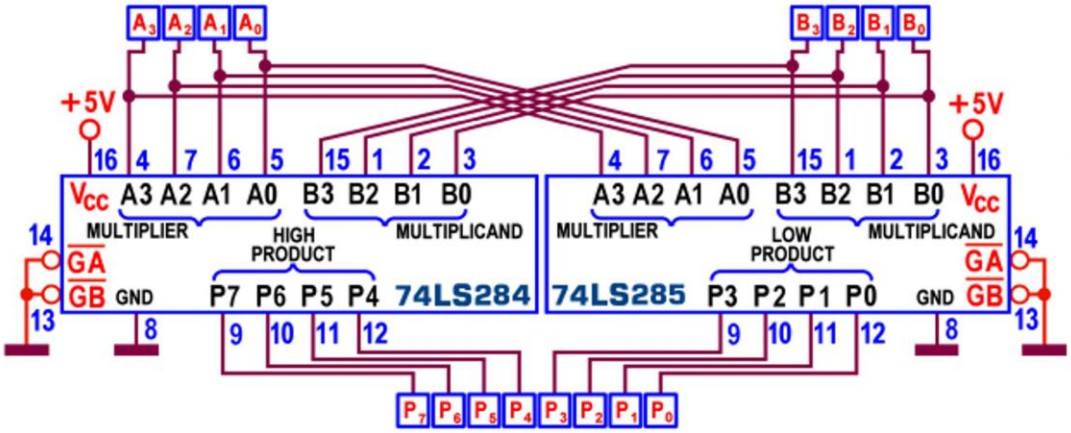

Figura 19 - 4-Bit by 4-bit Parallel Binary Multipliers 74LS284/74LS285: schema applicativo

La potenza dissipata massima è, per entrambi, pari a **650 mW** e il ritardo di propagazione massimo tra ingressi e uscite (misurato con carico di **30**pF/**600**ohm) è di **60**ns per entrambe le transizioni $\mathbf{t_{PLH}}$ e $\mathbf{t_{PHL}}$); poichè questa coppia di dispositivi può essere facilmente coinvolta più volte (per realizzare moltiplicatori in grado di assicurare risultati di più bytes) i manuali ne vantano la sua buona velocità di calcolo dichiarando tempi *tipici* di **40**ns (per prodotto a 8 bit), **70**ns (per prodotto a 16 bit) e **103**ns (per prodotto a 32 bit).

Il progetto di moltiplicatori con operandi di grande dimensione è, per altro, piuttosto impegnativo; basti pensare che per moltiplicare due numeri a 8 bit (con risultato offerto su 8*2=16 bit) sono necessarie 4 coppie **74LS284**/**74LS285** (per il calcolo dei *prodotti parziali*), 4 sommatori **74LS183** (per la gestione delle somme intermedie), 3 unità aritmetico-logiche **74LS181** e una unità carry look-ahead **74LS182** (per determinare la somma finale, da gestire con calcolo veloce dei riporti): in tutto ben 16 integrati.

Alla serie **TTL** appartiene anche il **74LS261**, noto come **2-Bit by 4-bit Parallel Binary Multipliers**; la sua disponibilità è rivolta al progetto dei sofisticati moltiplicatori basati sulla "ricodifica" del moltiplicatore al fine ridurre il numero di prodotti parziali da sommare; in particolare viene utilizzato un algoritmo detto "2 bit alla volta" (per il quale ciascun *prodotto parziale* è spostato 2 posti a sinistra invece di un posto, come avviene nella normale moltiplicazione); questo componente è destinato alle richieste dell'hardware dell'albero di ***Wallace***, consentendo una riduzione sostanziale dei tempi di moltiplicazione e dei costi.

In assoluta coerenza con quanto detto in precedenza non è opportuno aggiungere ulteriori dettagli, complicati anche didatticamente; basti sapere che, per un moltiplicatore 16*16 saranno necessari 32 di questi integrati **74LS261**, insieme a 2 **74LS00** (NAND), 2 **74LS00** (AND), 56 **74LS183** (sommatori carry-save), 7 **74LS181** (ALU) e 2 **74LS182** (look-aheed generator).

In chiusura vanno citati altri 3 dispositivi **TTL**: il **74LS274** (una versione **4-Bit by 4-Bit Binary Multipliers** con uscite 3-state, in grado di assicurare un prodotto a 8 bit *tipicamente* in **45**ns); il **74LS275**, la cui definizione (**7-Bit Slice Wallace Trees**) tradisce immediatamente la sua utilità nelle sofisticate strutture di calcolo del prodotto con l'algoritmo omonimo; e il **74LS384** (detto **8-Bit by 1-Bit Two's-Complement Multiplier**, logicamente diverso dai precedenti perchè di tipo sequenziale (il moltiplicando e il prodotto finale sono gestiti serialmente, rispettivamente in ingresso e in uscita); coinvolge 2 numeri espressi in complemento a 2 per produrre un prodotto in complemento a 2, senza alcun componente esterno, utilizzando internamente l'algoritmo di ***Booth***).

Moltiplicatori (CMOS): 4554

La serie **CMOS** prevede per la moltiplicazione binaria il **4554**, detto **2 x 2-Bit Parallel Binary Multipliers**; il suo *pin-out* è mostrato in *Figura 20.*

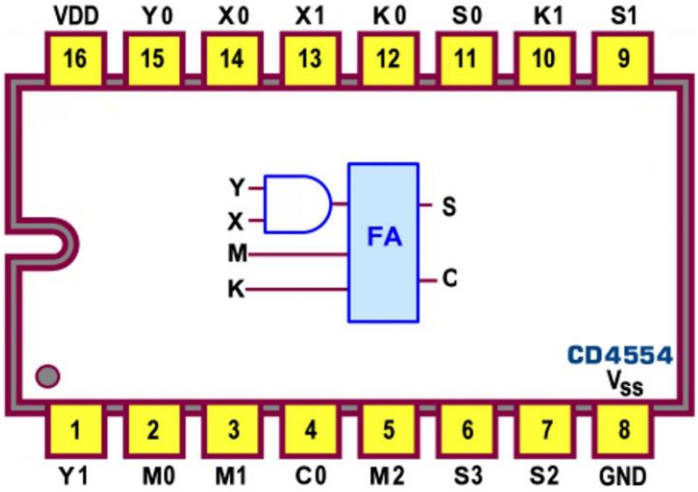

Figura 20 - 2-Bit by 2-bit Parallel Binary Multipliers 4554: pin-out

Contiene quattro celle di calcolo uguali tra loro la cui funzione (vedi *Figura 21*) è quella di sommare (tramite un FA) le due variabili K e M con il prodotto delle altre due, X e Y, secondo la formula **S=(X*Y)+K+M**.

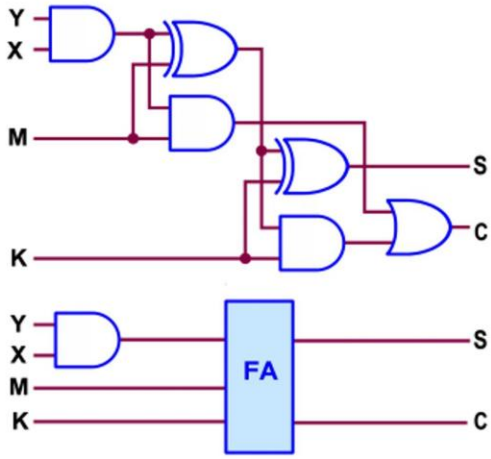

Figura 21 - 2-Bit by 2-bit Parallel Binary Multipliers 4554: cella di calcolo

In parte le 4 celle di calcolo sono già internamente collegate in cascata (vedi *Figura 22*): il riporto del primo FA va sull'ingresso K del secondo, la somma di quest'ultimo va sull'ingresso M del terzo e il riporto di quest'ultimo va sull'ingresso K del quarto; si tratta dunque di un componente in grado di eseguire la moltiplicazione di due numeri binari a 2 bit (moltiplicando X0,X1 e moltiplicatore Y0,Y1) e, contemporaneamente, sommare altri due numeri binari a 2 bit (K0,K1 e M0,M1) al prodotto; il risultato è disponibile su 4 linee d'uscita, S0,S1,S2,S3.

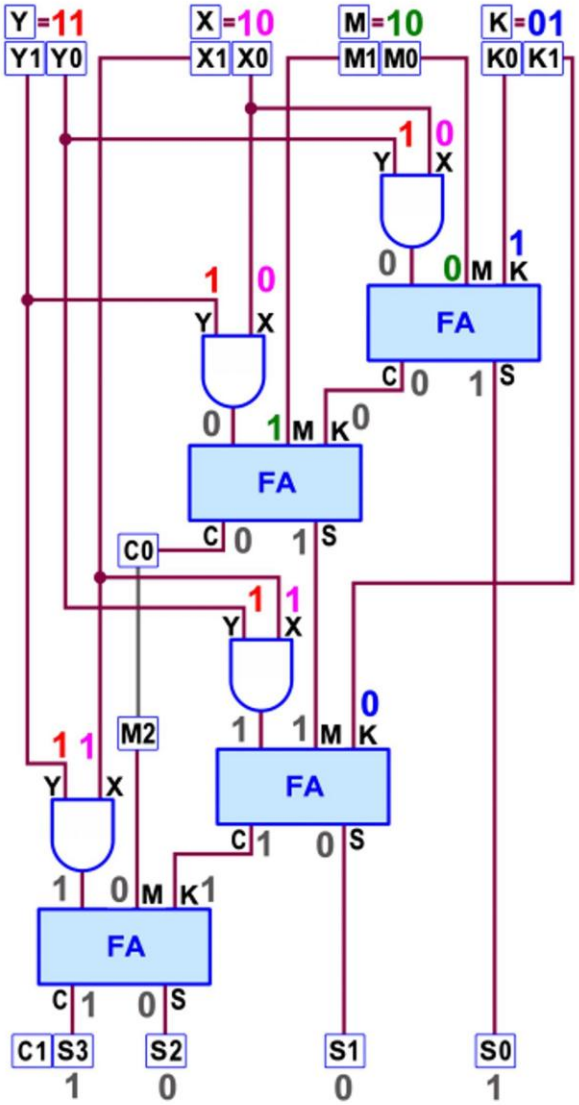

Figura 22 - 2-Bit by 2-bit Parallel Binary Multipliers 4554: schema funzionale

Le 4 celle sono indipendenti a 2 a 2, ma possono essere coordinate in un unico blocco collegando esternamente il pin 4 (C0) con il pin5 (M2); questa situazione è adottata nello schema di *Figura 22*, nel quale sono evidenziati i livelli logici nei suoi vari punti dopo aver imposto sugli ingressi i valori **Y**=$(3)_{10}$=$(11)_2$, **X**=$(2)_{10}$=$(10)_2$, **M**=$(2)_{10}$=$(10)_2$ e **K**=$(1)_{10}$=$(01)_2$, a verifica del particolare funzionamento; il risultato è quello atteso: S=(X*Y)+K+M=(3*2)+1+2=$(9)_{10}$=$(1001)_2$.

Come è noto la tecnologia **CMOS** assicura una *potenza dissipata* trascurabile ma i *ritardi di propagazione* sono maggiori rispetto alla **TTL** e dipendono dalla tensione d'alimentazione, valutabili da **1700**ns (**V_{DD}=5V**) a **570**ns (**V_{DD}=15V**), con il solito carico di **200**kohm/**50**pF.