**Come si realizza una macchina aritmetiche?**

Allora puó essere realizzata sia con un approccio combinatorio che con un approccio sequenziale. Proprio i moltiplicatori d esempio rappresnetano una espressione fondamentaledi macchina aritmeticha in quanto rappresentano la base per molti modelli e algoritmo di machine learning su reti neurali.

Si puó realizzare a livello porta, un circuito per ciascuna operazione, a livello comportamentale a livello strutturale. I bit degli operand possono essere presentati in modo diversi e la modalitá con cui essi verranno presentati impatterá sulla natura, complessitá e velocitá della macchina aritmetica: presentazione parallela (i bit degli op sono presentati in ingresso e i risultati sono calcolati contemporaneamente a meno di ritardi di programmazione), seriale pura (bit degli op e delle uscite vengono presentati in serie), seriale a gruppi o mista (un op è presentato in serie e l’altro in parallelo).

In generale per i dati in parallelo, si considerano i ritardi si una sola rete combinatoria, maggior velocitá.

**Cos’è un Half Adder? E un Full Adder?**

È un componente che somma due bit e presenta un uscita S e un riporto C. MANCA IL RIPORTO IN INGRESSO che invece è presente in un full adder.

Un full adder è un componente che somma due numeri binari, è realizzato con due half adder e una porta OR per gestire il riporto uscente

**Cos’è un Ripple Carry Adder?**

È detto sommatore a propagazione di riporto e deriva dall’algoritmo manuale per la somma di due numeri interi positivi di n bit. Presenta una struttura formata da un half hadder in entrata e tanti full adder collegati in cascata ed è fatto in modo che alla prima cella a destra vengano dati in ingresso i primi bit ei due operandi. Tale cella produrrá una somma s0 ed un riporto c1 che sará il carry entrante verso la seconda cella che lavorerá in questo modo a sua volta. Ponendo c0 pari a 0 è possibile eliminare l half adder iniziale e mettere un full in quanto il riportoé 0. Ad ogni modo il ritardo è pari a 2 delta n con delta ritardo di porta e l area occupata è 5n pari al numero di porte logiche per ogni pezzo el ripple carry per n numero di full adder necessari. I limiti di questo sommatore stanno proprio nei tempi di propagazione dei riporti che finiscono per diventare molto elevati per numeri estremamente grandi

Puó anche ssere usato come sommatore, semplicemente considerando l’operazione A-B come A+(-B) dove – B non è altro che il complemento a 2 di B che si ottiene facendo il complemento diminuito di B e aggiungendo 1. Per questo motivo la struttura sarebbe la stessa ma il riporto entrante alla prima cella sarebbe 1 e ci sarebbe una porta nor in corrispondenza di ogni bit proveniente dal secondo operando.

**E invece come si realizza un sommatore/sottrattore, quindi che varia in base alla scelta dell’utente, senza basarsi strettamente sull rca?**

Allora si basta considerare la proprietà della porta XOR la quale fornisce in uscita, dato un ingresso Y posto in XOR con 1, l’uscita notY. Viceversa se nella xor c’è 0 allora il numero in uscita sará uguale a quello in ingresso. In questo modo otterremo i bit di un operando negati facendo la XOR con 1. Quindi fondamentalmente ci sará un segnale di subtract che entrerá nella XOR e nel momento in cui è pari a 1 allora si fará la sottrazione altrimenti l addizione. Uscita della Xor e altro operando entrano in un RCA nel quale entra anche il segnale di substract perché nel momento in cui è pari a 1 devo effettivamente portare il riporto pari a 1 in ingresso. 0 altrimenti se si vuole fare solo la somma.

**Cos’è il sommatore Carry Look Ahead, come funziona e cosa cambia dall’RCA**

Il Carry L A si chiama cosí perché appunto guarda avanti, prova a non aspettare tutta la propagazione dei riporti classica di un rca ma prova a calcolarli tutti il prima possibile in modo da darli quanto prima insieme alle cifre da sommare da ogni unitá di un rca.

Si introducono le condizioni di propagazione e di generazione Pi = xi + yi e G^i = xiyi per cui posso scrivere, ricordando la regola rel riporto C = XY + Z(X+oY) , il riporto in funzione rispettivamente di Progapage e Generate

Carry successivo: Ci+1= xiyi + (xi+yi)= Gi+Pici con

Ci = xi-1yi-1 +(xi-1+yi-1)ci-1 = gi-1 + pi-1ci-1

Da cui si ottiene Ci+1= xiyi + (xi+yi)= Gi+Pi(gi-1 + pi-1ci-1) = Gi + PiGi-1 + PiPi-1ci-1

Ripetendo lo stesso procedimento

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

Posso scrivermi tutti i riporti in qualsiasi stato in funzione di P e G che dipendono solo dagli addendi e da c0. Quindi mi calcolo prima tutti i G, tutti i P e tutti i ci . Alla fine mi basta sommare tutti i riporti e tutti gli addendi in full adder non connessi. Si calcolano generate e propagate usando una sola porta, quindi avremo 1∆, poi da Generate e Propagate si calcolano i carry, una or e una and quindi 2∆. Noti ii carry si calcolano le somme nei fa, 2 porte, quindi 1∆+2∆+2∆=5∆ ritardo complessivo

L’area occupata sará pari a (n2+9n)/2. Il ritardo totale è sempre 5delta ma è ottimizzabile a 4∆ sostituendo i FA con porte XOR perché si puó scrivere la somma come S = x +o y +o Cin = P+o Cin e usando le XOR appunto al posto dei FA.

**Cos’è un sommatore Carry-Select?**

E’un tipo di sommatore che si propone di rislvere il problema dell rca inerente al fatto che la catena di FA cresce all’aumentare del numero di bit. L’IDEA è quella di dividere queste catene in P blocchi, ciascuno dei quali somma M bit. Il primo è un RCA di M bit che restituisce la somma ed il riporto, i blocchi successivi hanno 2 RCA che fanno la somma degli STESSI M bit ma uno ha Cin 0 e l'altro C in 1. In questo modo non si deve attendere che il riporto si propaghi attraverso ogni singolo adder ma abbiamo giá pronti i risultati corretti da selezionare per ogni blocco. Il ritardo toale è notevolmente ridotto i nquanto si passa da un problema di propagazione sequenziale dove i ritardi si sommano ad uno parallelo dove il ritardo è dominato dal blocco piú lento piú un piccolo overhead dato dal mux che deve selezionare. Si definiscono TFA e TMUX i tempi di propagazione dei FA e del MUX:

il carry in uscita dal primo blocco è disponibile al tempo M\*TFA perché è lungo M.

Somme e carry all’uscita del secondo saranno disponibili dopo M TFA+ TMUX perché c’è stata una selezione. Quindi alla ine avremo un tempo totale t= M TFA + (P-1)TMUX

**Cos’è un sommatore Carry save?**

Nasce inizialmente come gli altri per tentare di migliorare i tempi di propagazione ma in realtá risulta molto utile quando bisogna sommare piú operandi. Se quindi vogliamo effettuare S = x+y+z, se volessimo realizzare questa operazione con un classico rca per sommare le prime due stringhe e poi sommare la terza con il risultato avremo un ritardo significativo perché i riporti si devono propagare attraverso due sommatori praticamente. Il carry save invece somma allo stesso tempo i bit corrispondenti dei 3 operandi producendo una stringa di somma e una stringa di riporto che andrá sommata alla cifre successive. La prima parte viene fatta da un fll adder (cs). Da tale rete escono le due stringhe che dovranno essere sommate con un RCA. La particolarità sta nel fatto che i blocchi carry save operano in parallelo perché non sono soggetti alla propagazione del riporto.

**Differenza tra moltiplicatori paralleli e seriali?**

Mai dimenticare che, se moltiplico una stringa di n bit con una di m bit l’uscita sará di n+m bit

La moltiplicazione di interi SENZA SEGNO avviene in decimale come in binario, stesso metodo.

I paralleli effettuano il prodotto dele cifre + la somma delle righe e sono di tipo multiply and accumulate, quelli seriali sono basati su codifiche come Booth e derivano da una procedura manuale Robertson.

**Fammi un esempio di moltiplicatore Parallelo**

Allora di base in un moltiplicatore parallelo l’operazione si divide in due fasi, una prima fase in cui si determina la matrice dei prodotti parziali utilizzando opportune AND e una seconda fase, la somma dei riporti parziali che puó essere realizzata secondo varie tecniche:

Un esempio è il prodotto somma per righe (ripple carry multiplication) in uci dopo aver calcolato la matrice dei prodotti parziali si calcola il prodotto come somma delle n righe della matrice con n-1 rca. Le prime due righe della matrice vengono sommate con due HA e due FA, il primo e l’ultimo stadio invece presenta dei HA perché presentano solo entrate. L’uscita dal primo rca deve essere sommata alla terza righe, l uscita dal secondo deve essere sommata alla quarta e ultima riga. La struttura cos’i gestita porta problemi di propagazione di ritardo simili a quelli del singolo rca a causa della propagazione dei riporti. Gli adders piú vicini agli operandi forniscono un risultato dopo 3T perché il primo T viene usato per il calcolo dei prodotti parziali e i successivi 2 T per il calcolo nel FA o HA. Il ritardo complessivo del moltiplicatore è 17T assumendo che il ritardo per ogni adder sia 1t dopo la disponibilità degli operandi. Possiamo calcolare l istante di tempo in cui sono disponibili le due uscite del FA come tcout,s=max(ta,tb,tcin) +1. Dato che i riporti si propagano tutti nella stessa direzione, i tempi richiesti per la moltiplicazione dipendono linearmente dal numero di bit dei fattori (i ritardi sono quindi circa doppi di quelli necessari per la somma).

Un altro esempio è il moltiplicatore che fa il prodotto come somma per diagonali sfruttando una struttura di tipo carry save in cui il riporto non viene piú propagato sulla riga ma in diagonale. Ha una struttura regolare e ben integrabile. Guardando una matrice dei prodotti parziali, tutte le righe tranne l’ultima non propagano il riporto ma lo mandano ogni volta allo stadio successivo. Quindi tutti i FA dello stesso livello daranno i risultati dopo lo stesso tempo, i ritardi che devono aspettare i blocchi del livello successivo saranno sempre gli stessi. In genere questo circuito è piú veloce del primo.

Di base dalla precedente cambia che ci sono n sommatori, 1 per ogni riga e non n-1 e che l’architettura di base è di tipo carry save.

Di base il costo per le due architetture è lo stesso ma la seconda viene eseguita piú rapidamente: 3 righe di 4 HA/FA contr 4 righe di 3 HA/FA.

**Cos’è un moltiplicatore a celle MAC?**

È un modello di moltiplicatore che si propone di risolvere i problemi e effettuare tutto in un'unica fase di operazione. Una cella MAC è composta da una porta AND per il prodotto parziale tra i bit iesimi dei due operandi e da un FA per la somma de riporto parziale con i riporti e le somme provenienti dai livelli adiacenti. È un’architettura regolare ed iterativa, simile a quella del prodotto come somma per diagonali. Ogni blocco condivide un termine con la stessa riga e un termine con la stessa diagonale.

Fondamentalmente non si ha un grande risparmio di tempistiche ma c’è un guadagno in termini di stabilità, riproducibilità ed interattività del circuito.

**Ma cos’è sto algoritmo manuale di cui si parla e dal quale deriva Robertson?**

È fondamentalmente il procedimento di base della moltiplicazione tra due numeri binari in colonna dove ad ogni passo devo shiftare il prodotto di j posizioni a sinistra per poter concorrere al calcolo del prodotto finale attraverso la somma di tutti i prodotti j-esimi. P=Sumj=0->n-1 xj2^jY. Quel 2 alla j sta ad indicare lo shift di j posizioni. Volendo si potrebbe implementare anche con un mux che seleziona i risultati in base a 0 e 1 ed uno shift register che shifta ad ogni colpo di clock.

Esiste un modo peró per evitare di dover memorizzare tutti i prodotti Pj per la somma finale, ad ogni passo si puó calcolare una somma parziale dei prodotto pari a Pi+1=Pi + xj2^iY, P0=0, CONVIENE AD OGNI PASSO SHIFTARE P VERSO DESTRA piuttosto che il prodotto parziale.

Quindi invece di sommare tutti insieme ne sommo uno alla volta, sommando la nuova stringa ad ogni iterazione con la somma parziale. In questo modo l ‘algoritmo è iterativo ma per implementarlo avrei bisogno di un sommatore, di un moltiplicatore, di uno shift register. Notiamo che la porzione P interessata nella moltiplicazione non è l’intera stringa ma solo la metá dei bit ogni volta traslata di una posizione. Quindi shiftiamo P verso destra.

Versione alternativa: moltiplicazione xi e yi, somma del prodotto parziale con Pi calcolato al passo precedente, shift a DESTRA di una posizione della somma parziale calcolata al punto 2 per calcolare Pi+1. Pi=Pi+xiY, Pi+1=2^-1Pi. Questa alternativa ha il vantaggio che ad ogni passo si effettua sempre e solo uno shift a destra di 1.

L’ALGORITMO MANUALE NON FUNZIONA PER NUMERI INTERI RELATIVI:

se i numeri sono codificati in complemento a due una soluzione semplice come idea sarebbe quella del negare tutti gli operandi negativi, effettuare una operazione unsigned sui numeri positivi risultanti e poi negare il risultato se necessario ma per fare ció servono altri cicli di clock. Sulla base di alcune proprietá della rappresentazione in complementi è possibile applicare uno schema usando una tecnica di moltiplicazione unsigned apportando dele modifiche all’algoritmo manuale:

Se X>0 e Y>0 si effettua una moltiplicazione tra unsigned

Se X>0 e Y<0 ogni volta che Y moltiplica xj=/0 il prodotto parziale sará negativo, quindi il bit piú significativo di A, registro in cui sto appoggiando il risultato, deve essere sempre 1. Si puó usare un latch in cui viene inserito 1 e che viene usato per alimentare lo shift register in tutte le iterazioni

Se X<0 e Y>0 per i primi n-1 prodotti parziali, il res è positivo mentre per l’ennesimo è necessario un passo di correzione con la sottrazione A-M.

Se X<0 e Y<0 come nel caso 2, il bit piú significativo diA è 0 finché xj=0 e diventa 1 quando moltiplico xj=1 per Y. Per l’ennesimo prodotto è necessario fare la correzione A-M.

**Come funziona il moltiplicatore di Robertson?**

Allora il moltiplicatore di Robertson contiene: un registro M per il moltiplicando Y, un registro Q che inizialmente carica X e funge come shift register, concatenato ad A che funge da accumulatore del risultato parziale dell’operazione. F un fliflop/latch in testa allo shift register che tiene traccia del segno durante la moltiplicazione. Un adder/sottrattore parallelo, una control unit, un mux che seleziona tra M e 0 a seconda del bit corrente di Q durante la moltiplicazione ed un contatore che tiene traccia del numero di iterazioni e passaggi effettuati. La porta XOR serve in quanto vi entra il segnale substract che esce dalla cu come abbiamo visto nl sottrattore sommatore all inizio.

Allora in pratica M viene caricato con Y, Q con X e A inizializzato a 0 mentre il counter viene impostato al numero di iterazioni che servono in base alla lunghezza dei bit di X e Y;

PER OGNI BIT DI Q, partendo dal meno significativo, se il bit corrente è 1, Il sommatore fa M+A altrimenti se è 0, A resta invariato (questa cosa è gestita dal mux).

Il latch viene messo ad 1 se il bit piú significativo di M e il bit corrente di Q sono entrambi 1 perché dobbiamo riflettere il valore negativo.

Dopo ogni addizione (o dopo una operazione “vuota” se Q0 era 0), A e Q vengono shiftati a destra di 1 ed il contatore viene incrementato. Infine se il bit meno significativo di Q è 1, indicando che il numero originale era negativo, si fa A-M per correggere il risultato finale che, dopo l’ultimo shift a destra, si troverá nel registro concatenato A.Q.

**E invece come funziona il moltiplicatore di Booth? E qual’è la differenza con Robertson?**

Di base mi consenste di velocizzare le moltiplicazioni disegnando la CU saltando la fase di somma e risparmiando cicli di clok.

Il moltiplicatore di booth si basa sulla codifica di Booth. Praticamente sia X un intero che viene rappresentato in compleemnti a due come xn-1xn-2…x0, definiamo y0=-x0, y1=-x1+x0, y2=-x2+x1 ecc., moltiplicando ciascun yi per 2^i corrispondente (y0 per 2^0) si ottiene una rappresentazione posizionale, che altro non è che una codifica diversa per il complemento a due, che usa quindi un alfabeto diverso, -1 0 1. È come se trasformassi il mio numero 1111111 in 000000-1 risparmiando tante operazioni aumentando gli zeri. I lvettorepuó essere rappresentato in forma di Booth-1 semplicemente sostituendo ciascuna coppia di bit adiacenti di x con un valore comepreso tra -1 0 1 secondo la regola: 00/11 ->0

01->1

10->-1

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Fondamentalmente basta modificare il moltiplicatore di Robertson facendo in modo che dentro Q devo guardare non slo Q0 ma anche un altro bit, devo aggiungere un bit a destra di Q0, che sarebbe Q-1 che fará da bit numero 1 per vedere la coppia. Sulla base dell’analisi di questi due bit si decide se si deve sommare, sottrarre o solamente shiftare ed in base a ció si metterá substract pari a 0 o a 1. In piú non c’è piú il passo di correzione alla fine perché sommo o sottraggo quando mi serve.

**Cos’è l’handshaking?**

Allora l’handshaking è un protocollo di comunicazione, cioè un insieme di regole che definiscono la comunicazione tra due o piú entitá. In particolare, l’handshaking è asincrono (non c’è un segnale di tempificazione ma la comunicazione è gestita dai segnali di controllo che le due entitá si scambiano) e prevede lo scambio di un certo numero di messaggi il cui numero minimo è pari a 2. In questo caso infatti si usa un segnale di strobe che viene inviato da A a B per segnalare a B che puó leggere il dato sul bus e B risponde con ack che significa che ha letto.

Puó essere implementato peró anche a 3 vie, in modo piú completo: c’è un segnale di REQ che segnala a B che A vuole inviare un dato, B risponde con ACK che significa che è disponibile e dopo aver effettuato la sua elaborazione, B invia un altro segnale DONE che significa operazione completata/dato ricevuto. È utile questo modello quando B deve fare operazioni lunghe.

Poi un protocollo puó essere anche sincorno o semisincrono. In quest ultimo caso A e B hanno stesso riferimento temporale ma in caso di perdita di sincronismo B puó inviare un segnale di stato che significa aspetta.

Inoltre con un protocollo di HS Semplice le due entitá si mettono accordo sullo scambio di un dato ma A non è in grado di sapere quando B ha terminato eventualmente una elaborazione a partire da quel dato. Si potrebbe usare un terzo segnale DONE con il quale b SEGNALEREBBE AD a CHE HA FINITO MA CI SAREBBE DI INTERLACCIARE done con un altro segnale per avere la sicurezza che tutto vada bene. Per evitare ció si potrebbe semplicemente usare il fronte di discesa di accepted (della coppia data ready e accepted) che viene mantenuto alto fintanto che B elabora e viene abbassato quando ha terminato. In questo modo A capisce senza che sia aggiunta un'altra coppia di segnali.

**Cos’è la timing analysis?**

I processi di sintesi ed implementazione possono essere configurati per tener conto di specifici constraint, riguardanti sia aspetti specifici di tali processi (placement e synthesis constraints) sia aspetti di tempificazione (timing constraints). I constraint possono essere specificati sia in VHDL all’interno del design (prevalentemente quando si tratta di vincoli su come un design debba essere implementato o ottimizzato, es. max fanout) sia attraverso appositi file, di tipo .XDC. Il risultato dell’applicazione dei timing constraint durante l’implementazione è tipicamente quello di posizionare la logica in blocchi quanto più possibile vicini in modo da usare il meno possibile le risorse di routing.

Primary clock: definisce un riferimento temporale per il progetto e viene utilizzato dalla timing engine per derivare i requisiti del timing path e la relazione di fase con altri clock. Il delay associato ad un primary clock viene calcolato dal pin/porto sorgente sul quale il clock è definito ai clock pin delle celle sequenziali ai quali esso viene collegato, per cui è opportuno definire i primary clock su segnali che appartengono al boundary del design.Se nel comando create\_clock non si specificano sorgenti (pin, ports,nets) per il clock con il comando get\_ports allora viene generato un **virtual clock**. Un **generated clock** è un clock derivato da un altro clock esistente, chiamato master clock, che viene trasformato attraverso un blocco logico che opera un divisione/moltiplicazione di frequenza, uno shift di fase, un’inversione, una modifica nel duty cycle, o una combinazioni delle precedenti (es. attraverso un Clock Modifying Block come MMCM o PLL sui device 7 series).

L’analisi della tempificazione può essere effettuata già a valle della sintesi, però in quel caso i valori di tempificazione ottenuti saranno delle stime abbastanza approssimate poiché i tempi legati al routing, che di solito valgono tra i 45% e il 65% del tempo totale, non sono ancora disponibili. I parametri di configurazione della timing analysis, che specificano il tipo di report da generare e cosa effettivamente mostrare nel report, devono essere opportunamente settati attraverso il pannello «Report Timing Summary». Per verificare quale sia la frequenza massima di funzionamento (FMAX) di un design è possibile diminuire progressivamente la frequenza del clock del design ed eseguire una timing analysis finchè non si ottiene uno slack negativo (worst negative slack). Attenzione, la FMAX non è fornita esplicitamente nei report ma può essere stimata calcolando 1/(T –WNS) dove T è il periodo del clock target e WNS è il worst negative slack nel caso intra-clock. Per ottenere una stima dei ritardi di un design puramente combinatorio, occorre inserire dei registri clockati a monte e a valle del design, che rappresentano rispettivamente gli input e gli output della macchina implementata. Il tool di sintesi cercherà di ottimizzare il ritardo fra i due registri (rispettando gli eventuali constraint forniti).