UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INGEGNERIA ELETTRONICA

TESI DI LAUREA

ALGORITMO PER LA DIVISIONE HIGH RADIX E IMPLEMENTAZIONE

RELATORE : Prof. Cester Andrea

LAUREANDO: Viel Daniele

A.A. 2008/2009

INDICE

INTRODUZIONE

Nei calcolatori in genere una delle parti fondamentali è l'unita di calcolo detta ALU per i numeri a virgola fissa e FPU per i numeri a virgola mobile la seconda non sempre presente soprattutto nei processori semplici e meno costosi. Al interno di una FPU dove i numeri sono in rappresentazione a virgola mobile come descritto nello standard IEEE 745, si trovano operazioni come addizione /sottrazione moltiplicazione divisione radice quadrata. L'addizione e la moltiplicazione sono di gran lunga le operazioni più utilizzate in testa l'addizione, mentre la divisione viene eseguita molte meno volte e per questo non è sempre presente. Un divisore hardware riesce comunque a essere più veloce di una divisione fatta come software per questo in applicazione particolari o in processori per uso generico vengono utilizzati dei circuiti per fare la divisione. Ci sono vari algoritmi per implementare un divisore ma si possono distinguere prevalentemente in due categorie, algoritmi basati sulla moltiplicazione e algoritmi a cifre ricorrenti. I primi tra cui il Newton-Raphson e espansione serie, sono basati su metodi per il calcolo numerico e tramite circuito moltiplicatore approssimano il risultato fino a convergere alla soluzione. La seconda tipologia di algoritmi si basa principalmente sulla procedura utilizzata per fare la divisione mano con l'utilizzo di somme e sottrazioni. Ne fanno parte la divisone con ripristino e senza ripristino e divisione SRT (Sweeney, Robertson, Tocher). Un rapporto fondamentale da valutare per un circuito integrato è il rapporto prestazioni consumo d'area ed è proprio questo rapporto che permette di scegliere tra le due tipologie di algoritmi, gli algoritmi basati su moltiplicazione risultano molto veloci perché convergono quadraticamente verso la soluzione mentre gli altri basati su somme sottrazioni convergono linearmente ma i primi oltre ad essere più complessi hanno un occupazione d'area elevata che in certi calcolatori è difficilmente accettabile. I secondi invece non raggiungo la rapidità dei primi però permettono di ridurre il consumo d'area al interno della FPU. Ha senso parlare di questi algoritmi perché vengo implementati in processori per uso generico dove non sono richieste prestazioni eccezionali anche se comunque molto migliori rispetto a metodi software ed è richiesta una riduzione del occupazione d'area. Nei processori per uso generico è richiesta una semplice ed immediata facile integrazione con il resto dei componenti della FPU per la limitata possibilità di inserire soluzioni ad hoc. Nel caso di unità di calcolo per attività specifiche con esigenze di rapidità e con poche esigenze di integrazione allora si ricade nella tipologia degli algoritmi basati sulla moltiplicazione, se viene richiesto è possibile eseguire la divisone in un ciclo di clock. Per la scelta di un divisore bisogna tenere in considerazione la percentuale di utilizzo del divisore rispetto alle altre operazioni per capire quanto influisce la lentezza di questa operazione. Potrebbe essere controproducente fare una divisore troppo veloce a discapito del area se la divisone viene usata molto poco frequentemente.
In questa tesina verrà illustrato l'algoritmo per fare la divisione SRT basandosi prima sul algoritmo senza ripristino e verranno poi mostrate alcune tecniche per migliorarne le prestazioni.

PRINCIPI DELLA DIVISIONE RICORRENTE E NORESTORING DIVISION

Come ci è stato insegnato da ragazzi, per fare la divisione si usa un algoritmo molto semplice ma meccanico quindi facilmente ripetibile su un calcolatore. Questo consiste nel confrontare il divisore D e il dividendo X tramite delle sottrazioni, per andare a determinare resto R e le cifre del quoziente Q. Si può notare subito dalla definizione:

Dividendo = Quoziente * Divisore + Resto

che assumendo il quoziente Q come il più grande numero ottenibile con n bit deve essere rispettata la condizione che $X \leq 2^{n-1} \cdot D$ altrimenti può essere prodotta un indicazione di overflow dall'unità aritmetica evitabile comunque con la normalizzazione degli operandi. Quindi se per esempio consideriamo un divisore intero dovremmo prevedere che il quoziente Q abbia un numero doppio di bit rispetto a divisore D e quoziente Q quindi:

$$Q \leq 2^{n-1} \text{ e } D \leq 2^{n-1} \text{ allora } X \leq 2^{n-1} \cdot 2^{n-1} = 2^{2n-2} \leq 2^{2n-1}$$

da questi valori si vede che nel caso intero se il divisore/quoziente viene contenuto su un registro a singola precisione il dividendo dovrà essere ospitato in uno a doppia precisione per evitare la condizione di overflow ed eventuali normalizzazioni. Un ulteriore condizione fondamentale è che il divisore D non può essere uguale a 0, questo per la definizione di divisione nel caso di una divisone di tipo floating point è previsto il lancio di un eccezione per segnalare l'accaduto nel caso della virgola fissa deve essere eseguito un controllo per rilevare l'errore. Per la divisione ricorrente che si basa appunto sul algoritmo che ci viene insegnato da ragazzi il risultato è composto da resto R e quoziente Q dalla definizione si ricava che $X = Q \cdot D + R$ e deve essere per forza che $R \leq X$ quindi si ha che $R \leq 2^{n-1}$ può quindi essere ospitato su un registro a singola precisione.
Per analizzare l'algoritmo che esegue la divisone conviene utilizzare operandi frazionari i per questo motivo si introducono delle semplificazioni rispetto al caso intero, c'è da precisare che la seguente spiegazione gli operandi sono in rappresentazione binaria. In presenza di operandi frazionari la condizione di overflow per garantire l'esistenza del dividendo X diventa $X < D$ da qui si vede che non serve più differenziare la capienza dei registri tutti gli operandi possono essere espressi con la stessa precisione, il quoziente Q verrà espresso nella forma $Q=0,q_1q_2q_3\cdots q_m$ (m = n-1) con $q_j = \{0,1\}$ le varie cifre del quoziente si ricavano da una sequenza di sottrazioni e shift utilizzando l'espressione:

$$p_{j+1} = 2 \cdot p_j - q_{j+1} \cdot D$$

nel generico passo j-esimo D viene comparato con il r_j detto resto parziale dopo uno shift di 2 bit per ricavare un valore di q_{j+1} che soddisfi la condizione $p_{j+1} \leq D$, tale condizione, nel caso dio rappresentazione binaria, equivale a dire che se $2 \cdot p_j \geq D$ allora $q_{j+1} = 1$ altrimenti $q_{j+1} = 0$.
Per verificare che la procedura sopra esposta funzioni vediamo come si evolve l'iterazione dal primo passo; il primo resto parziale è il dividendo quindi si può scrivere $r_0=X$ da qui l'iterazione:

$$\begin{aligned} r_m &= 2 \cdot r_{m-1} - q_m \cdot D \\ &= 2 \cdot (2 \cdot r_{m-2} - q_{m-1} \cdot D) - q_m \cdot D \\ &= 2^m \cdot r_0 - (q_m + 2 \cdot q_{m-1} + \ldots\ldots + 2^{m-1} \cdot q_1) \end{aligned}$$

da qui si ottiene dividendo tutto per 2^m:

$$r_m \cdot 2^{-m} = X - (q_1 2^{-1} + q_2 2^{-2} + \ldots\ldots + 2^{-m} \cdot q_m) \cdot D$$

Questa è proprio la definizione di divisione dove

$$R = r_m \cdot 2^{-m}$$
$$Q = (q_1 2^{-1} + q_2 2^{-2} + \ldots\ldots + 2^{-m} \cdot q_m) = q_0 + \sum q_j 2^{-i} \quad i=1,\ldots,m$$

q_0 può essere non considerato nel nostro caso perché il operandi sono solo positivi e vale quindi 0.
Analogo è il procedimento per i numeri interi , analizzando la condizione di overflow $X \leq 2^{n-1} \cdot D$ e date le problematiche che abbiamo constato precedentemente con i numeri interi i quali richiedono lunghezze di registri differenziate è possibile pensare di normalizzare questi operandi interi. La normalizzazione viene fatta dividendo per 2^{n-1} il divisore e per 2^{2n-1} il dividendo considerando la lunghezza dei loro registri. Date queste ipotesi la condizione di overflow diventa

$$X_f < D_f \quad \text{dove} \quad X_f = X \cdot 2^{-(2n-1)} \quad \text{e} \quad D_f = D \cdot 2^{-(n-1)}.$$

La dimostrazione del esistenza vien data con la l'esistenza della analoga per i numeri frazionari perché dalla definizione:

$$X = Q \cdot D + R$$

è uguale a

$$X_f \cdot 2^{(2n-1)} = Q_f \cdot 2^{(n-1)} \cdot D_f \cdot 2^{(n-1)} + R_f \cdot 2^{(n-1)}$$

ora per ritornare alla versione fratta si basta dividere tutto per $2^{(2n-1)}$ e si ottiene:

$$X_f = Q_f \cdot D_f + R_f \cdot 2^{-(n-1)}$$

con questa rappresentazione si può usare la formula iterativa già dimostrata descritta per il caso frazionale dato che tutti i termini sono frazionari, il resto deve essere ulteriormente normalizzato alla fine della divisione $R_f \cdot 2^{-(n-1)}$. Sia nel caso frazionale che intero nasce il problema di capire come fare per selezionare la cifra del quoziente tra quelle previste nel nostro caso {0,1} allora si sfrutta la sottrazione che comunque deve essere fatta per trovare il resto parziale e se $2 \cdot r_i - q_{i+1} \cdot D < 0$ allora $q_{i+1} = 0$ e il prossimo resto parziale è ottenuto ripristinando il precedente resto parziale con un somma. Il costo di questa somma ulteriore è molto pesante perché al caso peggiore sono costretto a fare due una sottrazione una somma, ma questo non è inaccettabile.
Un algoritmo che non prevede queste somme di ripristino è detto norestoring division. La differenza principale con il precedente è che questo accetta resti parziali negativi. Il nuovo procedimento si basa sulla seguente proprietà:
dato $2 \cdot r_i - D < 0$ senza fare il ripristino ma sommando il divisore al resto parziale con shift negativo si ottiene analogamente al caso con il ripristino

$$2 \cdot (2 \cdot r_i - D) + D = 4 \cdot r_i - D$$

riottenendo la stessa espressione che nel caso con ripristino. Un ulteriore modifica va fatta sul assegnazione delle cifre del quoziente perché accettando un valore negativo del resto parziale significa che il valore del quoziente accettato cioè 1 è sovrabbondante quindi al prossimo deve essere -1 per compensare. La regola di assegnazione del quoziente diventa quindi:

$q_j = 1$ quando $2 \cdot r_{i-1} > 0$ e $q_{j-1} = -1$ quando $2 \cdot r_{i-1} < 0$.

Es: $q_i\, q_{i+1}$=10 (troppo grande) $q_i\, q_{i+1}$=1-1 = 01 giusto.
Si vede come la regola di discriminazione nella selezione del quoziente si è semplificata è ora sufficiente decidere se il resto parziale è maggiore o minore di 0, però questo tipo di rappresentazione ha bisogno di una conversione nella forma binaria standard solitamente il complemento a due.

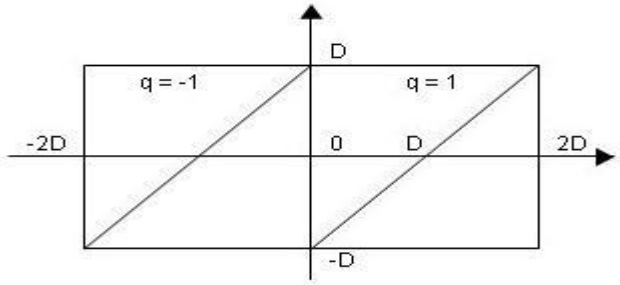

Fig. 1: grafico per la determinazione del quoziente.

Un vantaggio fondamentale per questo algoritmo è permettere la divisione tra dividendo e divisore di qualsiasi segno la semplice modifica di sostituire la verifica del segno del resto parziale con il confronto del segno con i divisore perché se il divisore è positivo ho la regalo sopra scritta se negativo o la regola contraria quindi la nuova regola per la selezione del quoziente diventa:

$q_j = 1$ quando $2 \cdot r_{i-1}$ e D hanno lo stesso segno;
$q_j = -1$ quando $2 \cdot r_{i-1}$ e D non hanno lo stesso segno.

Il vantaggio di rappresentare divisore e resto parziale in complemento a due è di agevolare la sottrazione e somma. L'algoritmo sopra descritto non è sufficiente a determinare valori corretti di resto e quoziente se vengono utilizzati operandi negativi ma possono essere fatte alcune assunzioni che poi si tradurranno in modifiche da apportare a fine processo:
il resto finale ha lo stesso segno del dividendo quindi se non è giusto alla fine va corretto, in più secondo questa rappresentazione l'ultimo bit non può essere 0 quindi numeri pari non posso essere generati allora nel caso che:

- Se dividendo e divisore hanno segni concordi al resto finale viene sommato D e al quoziente tolto un ulp (peso della cifra meno significativa).
- Se dividendo e divisore hanno segni discordi al resto finale viene sottratto D e al quoziente viene sommato un ulp.

La correzione del quoziente è dovuta al fatto che non si potranno mai ottiene numeri pari se lo 0 non è permesso, questo porta con se che va corretta anche la situazione in cui venga generato un resto parziale pari a 0 durante l'esecuzione del algoritmo in questo caso va sommato il divisore al resto e sommato un ulp al quoziente.
Esempio: norestoring division:

X = -0.375, D = 0.75
r_0 = X 1 .1 0 1

$2r_0$		1	.0	1	0	$q_1 = -1$	
Add D	+	0	.1	1	0		
r_1		0	.0	0	0		resto nullo
$2r_1$		0	.0	0	0	$q_2 = 1$	
Add -D	+	1	.0	1	0		
r_2		1	.0	1	0		
$2r_2$		1	0	.1	0	0	$q_3 = -1$
Add D	+	0	0	.1	1	0	
r_3		1	1	.0	1	0	

Quoziente convertito in complemento a due:

Q = 1.101 = -0.375

$R = r_3 2^{-3} = -0.09375$

$r_{corretto} = r_3 + D = 1.010 + 0.110 = 0.000 = 0$

$Q_{corretto} = Q - ulp = 1.101 - 0.001 = 1.100 = -0.5$

ALGORITMO HIGH RADIX

Caratteristiche generali degli algoritmi high radix

Come visto nella categorizzazione fatta in precedenza questo tipo di divisori fa parte dei divisori a ”cifre ricorrenti” i quali hanno un meccanismo di base che nasce dal quello della divisione presentata precedentemente utilizzando anche i numeri negativi come dimostrato con la norestroring division, si può intuire quindi che venga eseguito un calcolo iterativo in cui dati gli operandi al ingresso si ottengono dei valori del quoziente come del resto. Diversamente da prima che ad ogni iterazione veniva fatto uno shift a destra di un bit del resto parziale ora si cerca di velocizzare l'algoritmo andando a fare lo shift di più di un bit per volta, questo porta l'immediato vantaggio di ridurre la latenza del divisore perché è ridotto il numero di cicli eseguiti. In base a quanti bit vengono tolti per iterazione dal resto parziale si ricava la radice del quoziente cioè il peso per ogni cifra del quoziente quindi se prendo le cifre del quoziente generate per trovare il valore in decimale faccio:

$$\sum q_i \cdot R^{-i} \quad i=0,.....,i_{MSB}$$ (gli operandi per motivi già descritti sono frazionari)

la radice e il fattore fondamentale nel per questo i vari algoritmi ne prendono il nome (2 radix, 4 radix, 8 radix, 10 radix, 16 radix, ...).

Per l'implementazione di questi algoritmi si usa una tecnica molto simile alla norestoring divison chiamata SRT divison (dai nomi degli inventori Sweeney, Robertson, Tocher) la quale originariamente è una norestoring division con alcune differenze. In primo luogo viene

permesso anche il valore 0 tra le cifre del quoziente questo porta come prima conseguenza che per trovare il quoziente non basta più valutare il segno del resto parziale ma diventa necessario un pieno confronto con il divisore. Per evitare questo il divisore deve essere in forma normalizzata, andando ad eliminare prima di compiere la divisione gli 0 se positivo e gli 1 se negativo più significativi. La conseguenza è che l'intervallo del divisore è fisso cioè [½ , 1] questo porta come conseguenza che se $-½ \le 2 \cdot r_i \le ½$ allora $q_{i+1}=0$ se $2 \cdot r_i \ge ½$ $q_{i+1}=1$ e se $2 \cdot r_i \le -½$ $q_{i+1}=-1$. Questa regola può essere facilmente generalizzata al coso di operandi negativi diventando, sempre ricordando i vantaggi di una rappresentazione in complemento a due, quindi più in generale:

$|2 \cdot r_i| < ½$ $q_{i+1}= 0$
$|2 \cdot r_i| \ge ½$ $q_{i+1}= 1$ quando dividendo e divisore hanno lo stesso segno.
$|2 \cdot r_i| \ge ½$ $q_{i+1}=-1$ quando dividendo e divisore hanno segni opposti.

Esempio: divisione SRT

X=0.00111111 (63/256)

D=0.1001 (9/16)

$r_0 = X$		0	.0	0	1	1	1	1	1	1	
$2r_0 =>$ r_1		0	.0	1	1	1	1	1	1	0	< 1/2 set $q_1 = 0$
$2r_1$		0	.1	1	1	1	1	1	0	0	>= 1/2 set $q_2 = 1$
Add -D	+	1	.0	1	1	1	0	0	0	0	
r_2		0	.0	1	1	0	1	1	0	0	
$2r_2$		0	.1	1	0	1	1	0	0	0	>= 1/2 set $q_3 = 1$
Add -D	+	1	.0	1	1	1	0	0	0	0	
r_3		0	.0	1	0	0	1	0	0	0	
$2r_3$		0	.1	0	0	1	0	0	0	0	>= 1/2 set $q_4 = 1$
Add -D	+	1	.0	1	1	1	0	0	0	0	
r_4		0	.0	0	0	0	0	0	0	0	Resto nullo
$2r_4 =>$ r_5		0	.0	0	0	0	0	0	0	0	< 1/2 set $q_5 = 0$
$2r_5 =>$ r_6		0	.0	0	0	0	0	0	0	0	< 1/2 set $q_6 = 0$

$2r_6 \Rightarrow r_7$ 0 .0 0 0 0 0 0 0 0 < 1/2 set $q_7 = 0$

$2r_7 \Rightarrow r_8$ 0 .0 0 0 0 0 0 0 0 < 1/2 set $q_8 = 0$

Q = 0.01110000 = 0.4375

Come si vede dal esempio il numero di somme con questo metodo viene ridotto perché non serve più fare la somma/sottrazione se sappiamo già quale è il quoziente e prossimo resto parziale.

Principi della divisione high radix

Come è già stato detto la divisione high radix consiste nel prendere l'algoritmo SRT e trovare delle tecniche per aumentarne la radice. Come per ogni algoritmo della categoria a cifre ricorrenti l'algoritmo si basa su un passo iterativo che viene presentato con radice generica del tipo:

p_0 = dividendo
p_j = resto parziale
rp_j = resto parziale dopo moltiplicato per la radice
q_{j+1}= prossima cifra del quoziente
d = divisore
R = resto finale
Q = quoziente $Q=0,q_1q_2q_3\cdots q_m$

$$p_{j+1}= rp_j - q_{j+1} \cdot D$$

In genere per selezionare la cifra del quoziente nella iterazione corrente si usa la funzione SEL(d, rp_j).
Si può vedere che questo algoritmo ripetuto iterativamente va a soddisfare le proprietà della divisione, considerando operandi a n bit (m = n-1):
Dividendo = Quoziente · Divisore + Resto

$$\begin{aligned} p_m &= r \cdot p_{m-1} - q_m \cdot D \\ &= r \cdot (r \cdot p_{m-2} - q_{m-1} \cdot D) - q_m \cdot D \\ &= r^m \cdot p_0 - (q_m + r \cdot q_{m-1} + \ldots\ldots + r^{m-1} \cdot q_1) \end{aligned}$$

da qui si ottiene dividendo tutto per r^m:

$$p_m \cdot r^{-m} = p_0 - (q_1 r^{-1} + q_2 r^{-2} + \ldots\ldots + r^{-m} \cdot q_m) \cdot D$$

Questa è proprio la definizione di divisione dove

$$R = p_m \cdot r^{-m}$$
$$Q = (q_1 r^{-1} + q_2 r^{-2} + \ldots\ldots + r^{-m} \cdot q_m) = q_0 + \sum q_i \cdot r^{-i} \quad i=1,\ldots,m$$

Dalla relazione iterativa si nota dove avviene il vantaggio del aumento della radice cioè il

resto parziale ad ogni iterazione viene moltiplicato per la radice questo corrisponde più concretamente a uno shift a sinistra del resto parziale di un numero di bit pari a $b = \log_2 r$, quindi se raddoppia la radice aumenta di uno il numero di bit che vengono tolti e vengono dimezzati il numero di iterazioni da eseguire cicli = n/b, la relazione con il logaritmo base due fa capire perché si preferiscano implementazioni con radici che sono quadrati di due. La funzione di selezione del quoziente non è più esprimibile con confronti, al crescere della radice, e come si vedrà questa parte è una delle più lente del circuito e quindi maggiormente analizzata per cercare di aumentare le prestazioni. Al crescere della radice rimane il problema di somme e sottrazioni e si aggiunge la difficoltà di generare i multipli del quoziente. Es: radice di 4 cifre permesse [0,1,2,3], per creare $3 \cdot q$ è necessario eseguire un prodotto o una serie di shift più una somma.

Rappresentazioni delle cifre del quoziente

Nel caso di radici superiori a 2 è possibile selezionare le cifre dl divisore nel insieme {0, 1, 2 ,...., r-1}. Per evitare il problema di generare multipli troppo alti ed in alcuni casi i multipli dispari l'insieme delle cifre possibili è allargato a valori sia positivi che negativi del tipo {-n,-n-1,.....,n-1,n}, dove $(r-1)/2 \leq n \leq r-1$. La motivazione di questa restrizione si spiega facilmente perché n non dovrebbe comunque essere più grande di di r-1 altrimenti le cifre apparterrebbero a una radice superiore, nel altro senso servono almeno r cifre per rappresentare una cifra radice r quindi $2n+1 \geq r$ il (+1 è dato dallo zero). Questo risolve per esempio il problema del algoritmo radice di 4 perché avendo a disposizione un numero di cifre anche superiore a quelle richieste la rappresentazione che scegliamo può escludere le cifre che sono scomode per esempio il 3 quindi un possibile insieme di cifre potrebbe essere {-2,-1,0,1,2}.
Questo tipo di rappresentazione permette di avere un numero di cifre superiore a quello necessario nel caso $n > (r-1)/2$ quindi ci sono più simboli che cifre da rappresentare si può affermare allora che viene introdotta una ridondanza.
Es.: radice 4 con insieme delle cifre {-2,-1,0,1,2} il numero 6 può essere rappresentato come:

$$22 \text{ dato che } 4^{1.}\,2 + 4^{0.}\,(-2) = 8 - 2 = 6$$
$$12 \text{ dato che } 4^{1.}\,1 + 4^{0.}\,2 = 4 + 2 = 6$$

La ridondanza nelle cifre del quoziente da la possibilità che per alcuni valori del divisore e del resto parziale con shift sia possibile selezionare più di una cifra del quoziente, questo fatto è rilevante perché introduce un certo grado di libertà nella funzione di selezione. Se ben sfruttato questa proprietà permette di ridurre notevolmente la complessità della funzione che seleziona i quoziente. Più concretamente il vantaggio deriva dal fatto che non diventa più necessario andare ad analizzare con la massima precisione cioè per ogni bit divisore e resto parziale con shift, perché grazie alla libertà concessa dalla ridondanza è possibile determinare degli intervalli di valori in cui è possibile selezionare un certo quoziente, si capisce che la presenza di intervalli di valori rende inutile analizzare con la massima precisione i valori quindi è sufficiente un numero di bit ristretto quindi precisione limitata. Questo fatto porta notevoli vantaggi nella funzione di selezione perché riduce il numero di ingressi, però l'utilizzo di cifre ridondanti introduce qualche problema, per esempio è necessaria una conversione in coda al processo iterativo per poter rendere utili al resto del sistema i valori trovati.

Restrizioni nei valori di resto parziale

È necessario avere una restrizione sul resto parziale $0 \leq p_{j+1} < d$ perché se non è verificata significa che il quoziente selezionato q_{j+1} non è corretto. Considerando che sono ammessi sia valori positivi che negativi la relazione diventa

$$0 \leq | p_{j+1} | < | d | .$$

Se esiste una certa ridondanza come descritto precedentemente nelle cifre del quoziente (simmetrica) si ottiene il risultato che il l'intervallo di valori si restringere a

$$0 \leq | p_{j+1} | < K | d |$$

dove K è un parametro legato alla ridondanza e soprattutto al fatto che le cifre del quoziente sono distribuite tra valori positivi e negativi, la sua descrizione analitica è ricavata in seguito. Per verificare che queste restrizioni rimangano attuando l'algoritmo iterativo che esegue la divisone per ogni ciclo verifichiamo che un resto parziale p_j appartiene all'intervallo limitato. Quindi dato p_j che rispetta le restrizioni si ottiene $|r \cdot p_j| \leq r \cdot K \cdot |d|$ attuando uno shift, ora utilizzando l'espressione ricorsiva dobbiamo verificare che $-K \cdot d \leq r \cdot p_j - q_{j+1} \cdot d < K \cdot d$. Per fare questo per far questo si può usare una procedura grafica che stampa i valori del prossimo resto parziale in relazione al resto parziale attuale. Il grafico si ottiene normalizzando la relazione iterativa della divisione

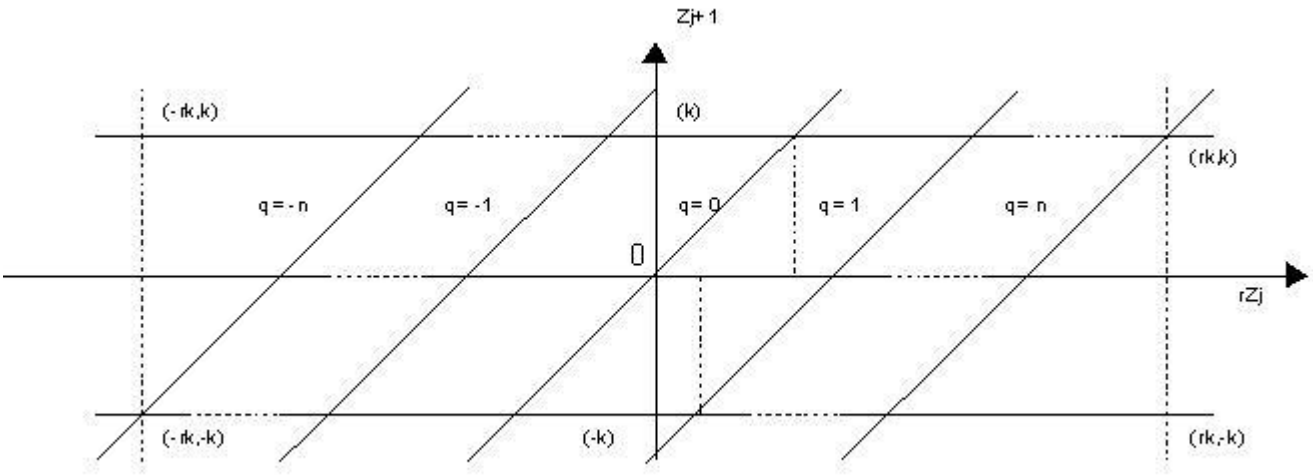

Fig. 2: grafico per la determinazione del quoziente r >2.

rispetto al divisore:

$$z_{j+1} = r \cdot z_{j+1} - q_{j+1} \quad \text{dove} \quad z_j = p_j/d$$

Si ricorda che le cifre del quoziente posso avere valori su {-n...n}, si verifica che in ogni punto del grafico

$$-r \cdot k \leq r \cdot z_j \leq r \cdot k$$

e per ogni valore di r ·z_j troviamo un q tale che z_{j+1} sia un valore compreso nel range [-k,k] quindi il grafico riscontra che il resto parziale successivo è limitato correttamente. È possibile esprimere k in base a n e r, notando che il massimo valore che può assumere z_j = rK questo avviene in presenza della massima cifra possibile n in quel punto il grafico assume un valore di z_{j+1} ben definito cioè K con questi dati si può calcolare K:
z_{j+1} = r ·K - n che diventa K = r ·K - n da qui:

$$K = \frac{n}{r-1}$$

Sapendo che n è compreso tra (r-1)/2 ≤ n ≤ r-1 si vede che ½ ≤ k ≤ 1. Siccome il grafico associa a ogni valore delle ascisse almeno un valore delle ordinate questo verifica la nostra assunzione che c'è sempre un q per verificare l'ipotesi di restrizione del intervallo.

Ridondanza nella rappresentazione e selezione delle cifre del quoziente.

Si può notare la ridondanza nella rappresentazione delle cifre del quoziente, proprio dal precedente grafico perché le zone in cui per un dato resto parziale con shift vengono puntati più valori, allora è possibile scegliere quale cifra usare. La massima ampiezza di queste zone è data per K=1 quando ho il massimo numero di cifre. Per vedere come la ridondanza può portare vantaggi nella selezione del quoziente viene proposto una altro tipo di rappresentazione che mette in relazione resto parziale con shift e divisore. La costruzione è basata sulla formula iterativa illustrata al inizio e sulla limitazione data sul resto parziale,

$r\,p_j = p_{j+1} + q_{j+1}d$ e $|p_{j+1}| \le K|d|$ quindi

$$\left|p_{j+1}\right| \le \frac{n}{r-1} \cdot |d|$$

Andando a rappresentare rp_j in funzione di d come $rp_j = (K + q_{j+}1)d$ per far questo si è usata la limitazione su p_{j+1} cioè $p_{j+1} \le Kd$. Siccome questa relazione dipende da K possiamo trovare delle funzioni limite superiore ed inferiore:

$$r \cdot p_{j\max} = \frac{n}{r-1} + q_{j+1} \cdot d$$

$$r \cdot p_{j\min} = \frac{-n}{r-1} + q_{j+1} \cdot d$$

Rappresentando queste due funzioni si ottiene la suddivisone in aree del piano tali aree prendono il nome dal quoziente che le ha generate e se per un certo divisore e un certo resto parziale si trova un punto in una di queste aree la cifra del quoziente a loro relativa è proprio quella che ha dato il nome al area.

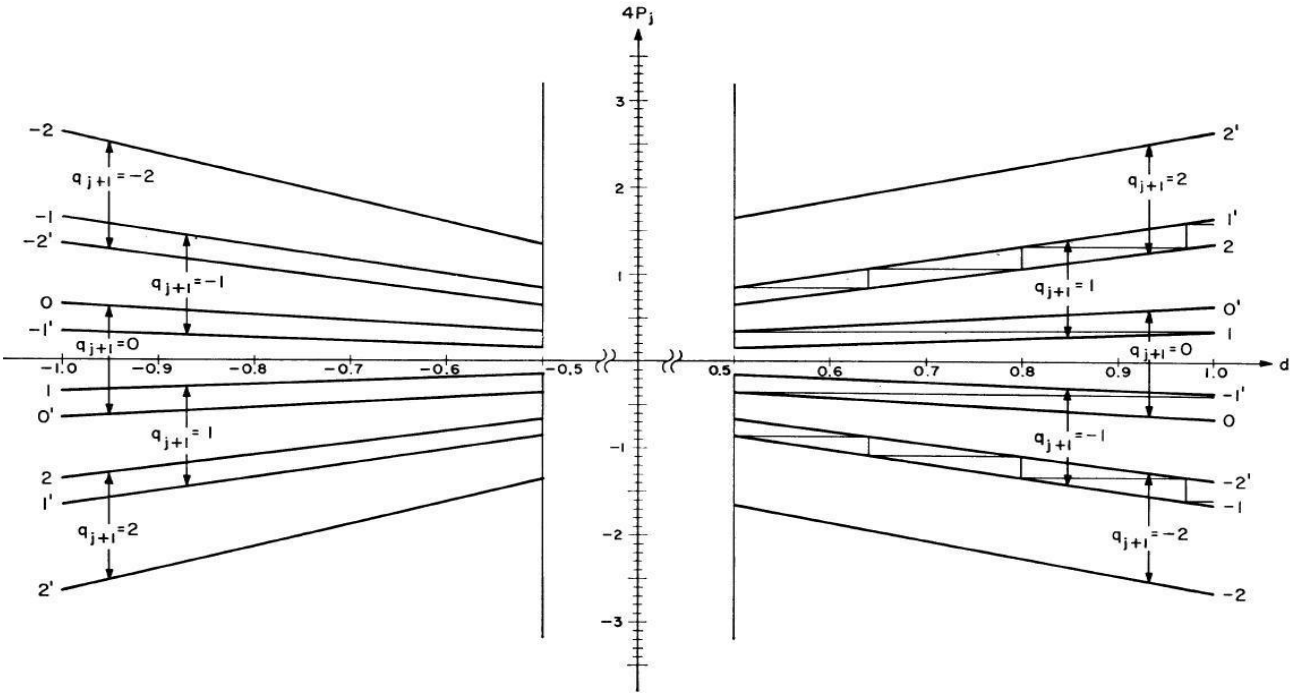

Fig. 3: grafico resto parziale con shift \ divisore, suddivisione in aree (r = 4 , n = 2).

Si vede che queste aree sono in alcuni punti sovrapposte e proprio in quei punti è possibile selezionare entrambe le cifre di quoziente relative a quelle aree. Il vantaggio di avere una zona dove è possibile scegliere quale quoziente selezionare permette di poter decidere in che modo suddividere l'area.
Un modo è quello di creare degli scalini cioè parti in cui il divisore è costante e parti in cui il resto parziale è costante. Avendo delimitato delle zone con valori costanti in un intervallo è possibile individuare quelle zone con una precisione minore, la precisione richiesta è quella necessaria ad individuare l'intervallo più piccolo. Questo discorso vale sia per il divisore che per il resto parziale. La possibilità data dalle aree di ridondanza è quella di poter definire delle zone individuabili con minor precisione e nelle quali il quoziente è noto con certezza.

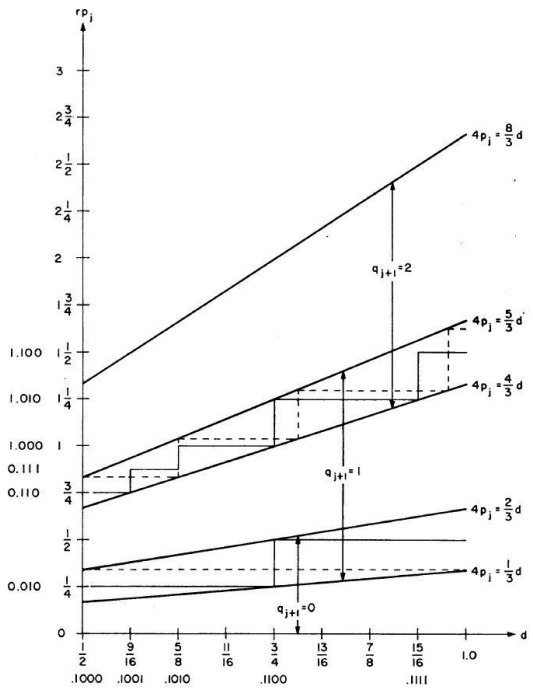

Fig 4: dettaglio grafico resto parziale con shift \ divisore con zone di decisone.

La scelta della configurazione di gradini non è banale è verrà analizzata in seguito ma si può già vedere che se il numero di gradini diminuisce la loro lunghezza aumenta quindi diminuisce la precisione richiesta per individuare una zona da loro delimitata. L'area della superficie sovrapposta dipende solo da k cioè il fattore di ridondanza, l'area può essere ricavata dalla formula: area sovrapposta tra le aree j e j-1:

$$Area = \int_{1/2}^{1} r \cdot p_{j-1\max} - r \cdot p_{j-1\min} dd = \int_{1/2}^{1} (2 \cdot k - 1) dd = \frac{3}{8} \cdot (2 \cdot k - 1)$$

Un altra caratteristica è che la linea centrale tra i margini della zona ridondante è stazionaria cioè rimane invariata al variare della ridondanza:

$$r \cdot p_j = \frac{1}{2} \cdot [(k + j - 1) + (k + j)] d = \frac{1}{2} (2j - 1) d$$

questo porta anche la conseguenza che la distanza centro nel grafico è sempre uguale per un dato divisore. La possibilità di utilizzare una precisione inferiore per localizzare un intervallo delimitato dalle scalinate sopra descritte si riflette sul fatto che è possibile analizzare solo i primi N_p e N_d bit più significativi di divisore e resto parziale. Più grande è l'intervallo più valori comprende quindi servono meno bit per localizzarlo e da qui il fatto che l'intervallo più piccolo è quello che richiede più precisione. La precisione richiesta non viene individuata solo in base al minimo intervallo ma anche in base al fatto che gli altri intervalli dovranno essere multipli di tale intervallo, questo perché la precisione cioè il numero di bit rimane costante,

quindi non è assicurato che anche i multipli di tale intervallo cadano completamente nel area di sovrapposizione. Quindi parametri fondamentali sono l'altezza minima della zona di sovrapposizione per trovare la precisione richiesta al resto parziale e la larghezza minima della zona di sovrapposizione per trovare la precisione richiesta al divisore.

$$Altezza = r \cdot p_{j-1\max} - r \cdot p_{j-1\min} = (k-1) \cdot d$$

$$Larghezza = d_2 - d_1 = \frac{r \cdot p}{-k+j} - \frac{r \cdot p}{k-j-1} = r \cdot p \cdot \frac{2 \cdot k - 1}{j^2 - j + k - k^2}$$

Ora valutiamo la precisione richiesta nei due casi assumendo che il numero di bit che rappresentano resto parziale e divisore siano n, mentre la versione troncata del resto parziale ne ha ε mentre il divisore ne ha ω. Sapendo il numero di bit si trova l'errore di troncamento
$e_p = 2^{-\omega} - 2^{-n}$ che può essere approssimato a $2^{-\omega}$
$e_d = 2^{-\varepsilon} - 2^{-n}$ che può essere approssimato a $2^{-\varepsilon}$
Quindi è possibile trovare il numero dei bit sapendo la lunghezza del intervallo più corto con il risultato:

$\omega \geq -\log_2 (½ (2k - 1)/(n - k))$ bit richiesti al resto parziale
$\varepsilon \geq -\log_2 (½ (2k - 1))$ bit richiesti del divisore

r	n	k	ω	ε
4				
	2	2/3	3	3
	3	1	2	1
8				
	4	4/7	6	4
	5	5/7	5	3
	6	6/7	4	2
	7	1	4	1
16				
	8	8/15	8	5
	9	3/5	7	4
	10	2/3	6	3
	11	11/15	6	3
	12	4/5	6	2
	13	13/15	6	2
	14	14/15	5	2
	15	1	5	1

Tabella del minimo numero di bit richiesti al variare di r e n

IMPLEMENTAZIONE

Schema generale

Noto l'algoritmo per fare la divisione è immediato introdurre nel circuito dei registri a scorrimento per contenere i vari operandi nella varie fasi del algoritmo in più è necessario un sommatore/sottrattore un circuito per la selezione del quoziente e se viene utilizzata una rappresentazione ridondante vari circuiti per la conversione del quoziente e normalizzazione degli operandi di ingresso. Le varie metodologie per realizzare questi parti del circuito permettono di ottenere ulteriori vantaggi e ridurre i tempi di latenza in più bisogna tenere in considerazione le ottimizzazioni che si possono fare a livello di transistor data una particolare implementazione.

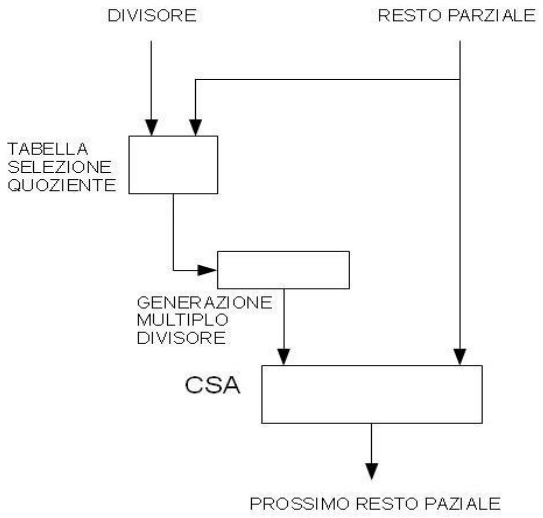

Fig 5: Schema elementare di un divisore

Il divisore introduce nel circuito nel quale è implementato una latenza di ciclo e una totale, la prima è il tempo che impiega per restituire un cifra del quoziente la seconda è il tempo per ottenere il risultato completo. Le componenti più pesanti in termini di latenza sono il sommatore e il circuito per fare la selezione del quoziente entrambi tramite ridondanza vengono velocizzati ma rimangono i punti critici. La maggior parte dei metodi per velocizzare il circuito si basano sul concetto di parallelizzazione quindi replica delle varie parti per farle lavorare in parallelo, quindi viene barattata del area sul chip per ottenere maggior prestazioni. E possibile apportare migliorie anche nel verso del risparmio di energia pur mantenendo accettabili le prestazioni.

Utilizzo carry save adder

L'utilizzo di un sommatore è essenziale per trovare il resto parziale perché deve essere eseguita la sottrazione tra il valore del divisore moltiplicato per il quoziente e del resto parziale dopo lo shift, siccome c'è bisogno della massima precisione è necessario fare la somma di tutti i bit quindi per esempio nel caso di singola precisione secondo lo standard IEE.745 nel caso di rappresentazione a singola precisione il sommatore è da 23 bit mentre in doppia da 52 bit questo fa capire che per quanto veloce si possa fare un sommatore la propagazione del riporto diventa lunga per un iterazione del algoritmo e se dovessero essere usati sommatori ad alte prestazioni con riporto, comunque non si riuscirebbe a finire un iterazione in meno di un ciclo macchina, ma soprattutto si perderebbero tutti vantaggi di usare un algoritmo di questo tipo che occupa poco spazio.

In alternativa al sommatore classico con propagazione del riporto viene solitamente utilizzato, nei divisori, il sommatore carry save questo tipo di sommatore fa la somma indipendente di ogni bit e salva il riporto in un ulteriore bit associato al bit di somma, il vantaggio fondamentale è che la somma dura il tempo di sommare un bit perché può essere fatta in parallelo tra tutti senza attendere nessun riporto. La presenza di questo sommatore prevede che il resto parziale sia salvato in forma carry save cioè una particolare forma ridondante perché si utilizzano più bit per rappresentare gli stessi numeri. Siccome non è possibile mantenere questa forma anche in altre parti del divisore come la tabella di selezione del quoziente saranno necessarie delle conversioni senza dimenticare la conversione finale per poter rappresentare il resto parziale nella forma utilizzata dal calcolatore per esempio

secondo lo standard IEEE.745 è il complemento a due. Tutte queste conversioni richiedo di sommare i bit di somma e i bit di riporto quindi è richiesta comunque una propagazione del riporto che va a limitare i vantaggi di questa tecnica. La conversione sarà tratta in seguito comunque grazie ad una tecnica di conversione al volo si il problema della latenza del riporto viene eliminato. Quindi il vantaggio di avere questo sommatore deriva dal fatto che occupa relativamente poco spazio e grazie ad una tecnica di conversione al volo non introduce il tempo di latenza finale. La struttura di un carry save è composta da tanti full adder quanti sono i bit da sommare

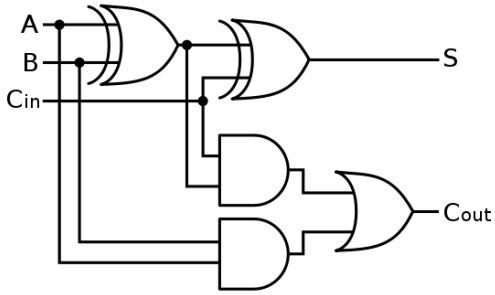

Fig 6: Schema full adder.

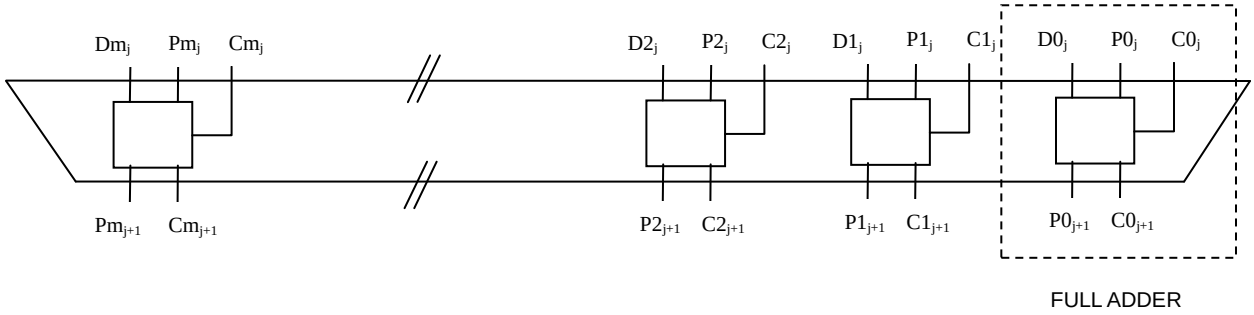

Fig. 7: Schema Carry Save Adder.

Questo tipo di schema prevede la presenza del doppio dei registri per salvare i bit di riporto e somma.

Generazione dei multipli del divisore

Nel momento in cui deve entrare in azione il sommatore è necessario che siano presenti: il multiplo del divisore dato dalla moltiplicazione tra d, il quoziente appena trovato, e il resto parziale precedente con shift di un numero di bit pari a $\log_2 r$. Queste che sembrano operazioni banali possono influenzare molto il tempo di latenza del divisore se non vengono scelti i parametri adeguati. La generazione del multiplo del divisore deve essere fatta dopo la generazione del quoziente e prima della somma quindi entra a far parte del cammino critico del divisore [Fig]. Di solito le cifre del quoziente sono prese in rappresentazione ridondante, per aver tutti i vantaggi nella selezione del quoziente e possono assumere i valori [-a,a] dove $(r-1)/2 \leq a \leq r-1$. Se esistono valori di a pari, il multiplo del divisore può essere trovato come shift del divisore nel caso dispari ci sono dei problemi perché l'unico modo per trovare il multiplo del divisore e ` fare uno shift più la somma del divisore ad eccezione del caso a=1. [Radice 4 n=3 deve essere trovato il valore 3d = 2d + d]. Questa somma va fatta con sommatore che propaga il riporto quindi è molto onerosa anche se sempre meglio del ritardo di un moltiplicatore . Questo è uno dei motivi per cui si fa fatica ad andare a divisori con radici

superiori a 4 e cifre superiori a due. Se non ci sono particolari problemi di spazio invece di generare i multipli al volo, si può utilizzare un multiplexer che seleziona i valori del divisore precedentemente elaborati, però anche qui oltre alla presenza del sommatore nel caso di cifre dispari diverse da 1 c'è la presenza di un numero di registri a massima precisione pari al numero dei multipli da generare nel caso di divisore radice di 4 vengono introdotti 5 registri che rendono pesante il multiplexer e occupano notevole area.

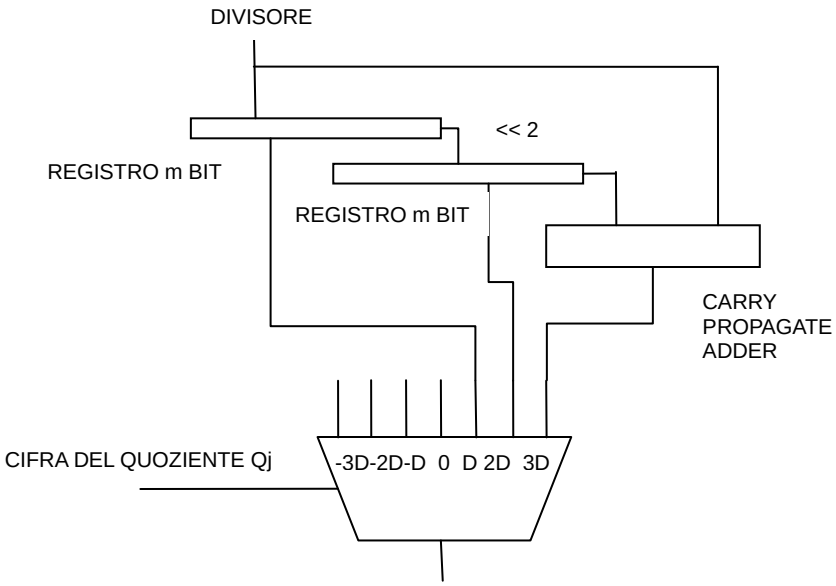

Fig. 8: Schema con le parti fondamentali per la generazione del multiplo del divisore.

Per la generazione del resto parziale basta fare uno shift pari alla radice dato che essa è sempre multiplo di due, in questo caso viene previsto un registro che permette lo shift oppure per risparmiare spazio si possono salvare i bit in uscita dal sommatore in versione con shift aggiungendo nei bit meno significativi degli 0 se il resto parziale è positivo mentre degli 1 se negativo. Allo scopo esistono shifter che riportano la prima cifra quella più significativa nelle posizioni che non hanno valore dopo lo shift cosi da mantenere il segno corretto. Nel caso della forma in carry save vengono aggiunti degli zeri con un semplice shifter.
Bisogna far notare che è necessario introdurre un ulteriore multiplexer in testa al sommatore che agisce solo al primo ciclo perché proprio li deve essere inserito il dividendo come primo resto parziale.

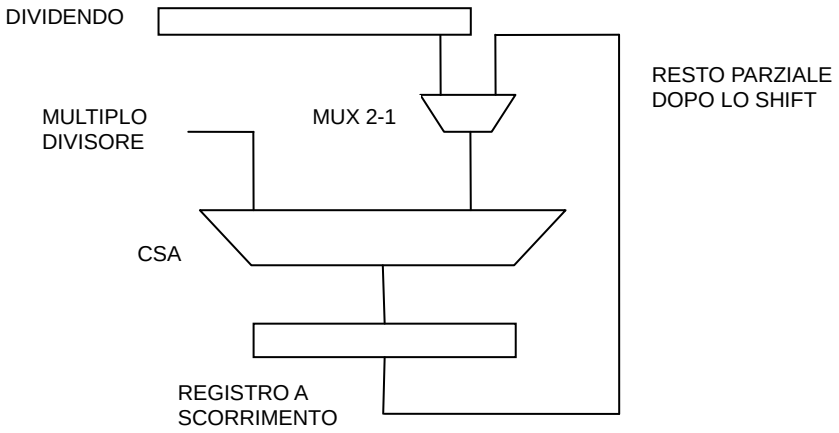

Fig. 9: Schema che evidenzia il mux per l'inserimento del dividendo e il registro per lo shift.

Gestione dei registri e temporizzazione

Per creare una struttura iterativa in cui si riutilizza il circuito per più cicli c'è sicuramente bisogno di registri temporizzati con il ciclo di clock del processore, siccome questo circuito è modulare con varie funzioni ben separate con tempi costanti nei vari moduli si presta bene a essere temporizzato. È possibile utilizzare varie strutture che hanno dei pro e contro in termini di prestazioni utilizzo di area e consumo energetico. Viene presentata un analisi in [] dove si analizzano varie strutture. La prima e immediata e l'utilizzo di flip flop edge triggered come registri

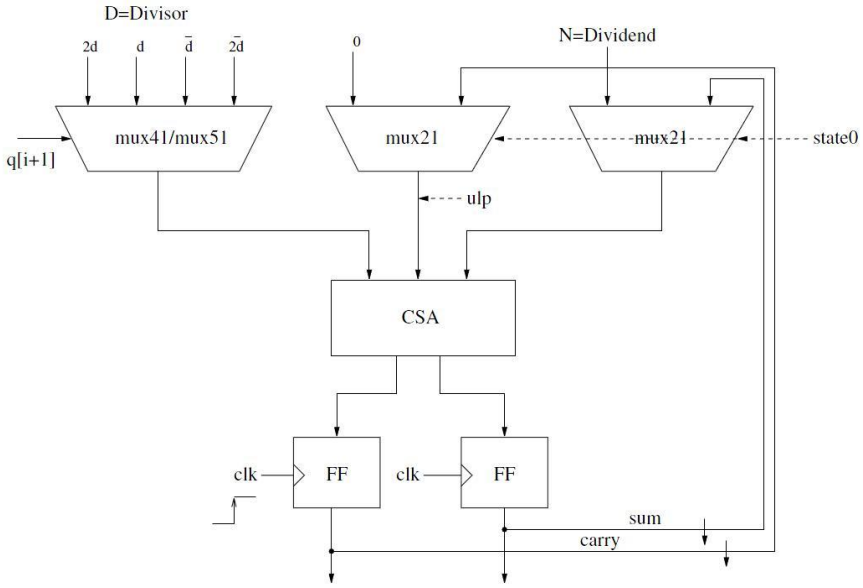

Fig. 10: Schema che evidenzia la posizione dei registri flip flop edge triggered.

che ad ogni ciclo di clock aggiorna i registri quindi un ciclo del divisore deve essere minore di un ciclo del clock. Oltre al tempo per fare un iterazione del algoritmo deve essere previsto un tempo di propagazione del comando nel registro un tempo di setup del registro un tempo dato dallo skew del clock. Si possono usare i latch singoli invece del flip flop quindi sensibili al livello di clock, in questo caso rimangono i tempi di setup dei due letch in serie e lo skew del clock ma non c'è più il tempo per la propagazione del comando. La pecca e che questa implementazione necessita di un latch in più 5 invece che 4. Un altra soluzione è utilizzare i

latch come dei flip flop essendo

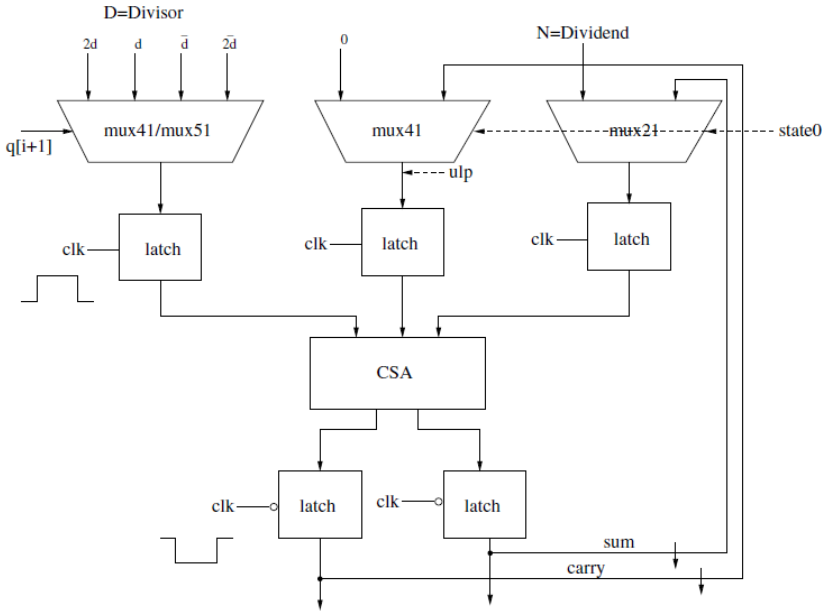

Fig.11: Schema che evidenzia la posizione dei latch.

un latch attivo al livello è possibile con un particolare segnale di clock attivarlo per un breve periodo necessario a far passare il bit ma poi spegnerlo subito, questo è possibile con un clock ad impulso. Il vantaggio è che si riduce il numero dei latch in serie quindi il tempo di propagazione attraverso di loro.

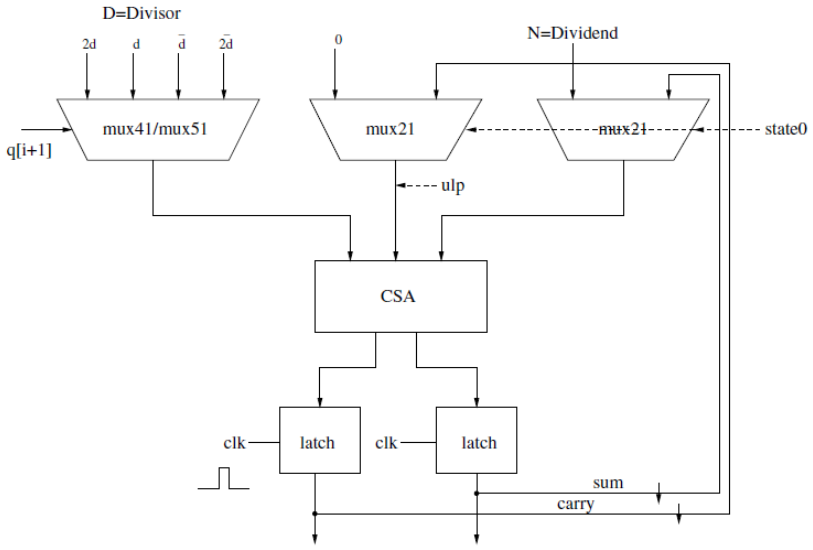

Fig.12: Schema che utilizza dei latch come flip flop sfruttando un segnale di clock impulsivo.

Analisi della complessità delle tabelle per la selezione del quoziente

Per implementare la funzione di selezione del quoziente essendo costituita solo da logica combinatoria, devono essere combinati gli ingressi per ottenere una cifra del quoziente, è

preferito ad altri sistemi l'utilizzo di una LPA (look up table) come viene analizzato da [] perché permette di utilizzare ingressi con precisione inferiore, quindi ridurre la complessità e la latenza del circuito. Per analizzare la complessità della realizzazione di una tabella di selezione (LPA) del quoziente non basta sapere la minima precisione richiesta cioè il numero di bit per ottenere un stima del divisore e del resto parziale per ottenere un quoziente corretto. Per ridurre la complessità e la grandezza di una tabella di selezione decisi la radice r e il coefficiente di ridondanza K quindi l'intervallo dei valori [-a,a] è preferibile avere al ingresso il numero di bit più basso possibile. Si assuma che il numero minimo di bit del resto parziale troncato è c e dato che si tratta di un numero con shift avrà i bit interi e f = ω frazionari la cui somma sarà sempre c. Il divisore è comunque sempre normalizzato quindi non interessa conoscere il suo primo bit per questo gli ingressi della tabella si riducono. Il numero di bit diventa b= ε - 1 bit tutti frazionari. Da questi dati otteniamo che deve esserci una PLA da 2^{b+c} ingressi. Per il resto parziale, si può assumere come ingresso la sua forma ridondante quindi con un numero di bit maggiore che in forma non ridondante a causa del fatto che è necessario rappresentare più cifre, oppure una non ridondante tramite l'utilizzo di un sommatore con riporto messo al ingresso della tabella che riduce la forma ridondante questo semplifica la tabella perché si trae beneficio dal minor numero di ingressi, comunque il vantaggio si ottiene se il sommatore e la tabella non ridondante sono più veloci della tabella ridondante. Vediamo la diversa complessità disponendo di una ingresso ridondante rispetto a uno non ridondante. Nel secondo caso si vede che il massimo errore permesso a divisore e resto parziale corrisponde con l'errore teorico cioè:

$$e_d \le 2^{-b} - 2^{-m} \qquad \text{approssimabile} \quad 2^{-b}$$
$$e_p \le 2^{-f} - 2^{-m} \qquad \text{approssimabile} \quad 2^{-f}$$

Si può vedere il peggioramento nel caso di rappresentazione ridondante per esempio nella forma carry save del resto parziale, ma valido anche per altre forme di ridondanza, siccome il numero di bit entranti aumenta in questo caso c` la presenza di una somma e di un riporto, errore commesso aumenta:

$$e_{ps} \le 2^{-f} - 2^{-m}$$
$$e_{pc} \le 2^{-f} - 2^{-m}$$

quindi il massimo errore è dato da

$$e_{p(cs)} \le 2\,(2^{-f} - 2^{-m})$$

Si vede che l'errore massimo raddoppia quindi per ottenere la precisione richiesta devo aggiungere un ulteriore bit alla rappresentazione sia per la somma che per il riporto.
Nel caso del resto parziale sono ancore da determinare il numero di bit interi i, siccome

$$rp_{j(max)} - rp_{j(min)} \le 2^{i}$$

sono richiesti resto parziale massimo e minimo. Ritornando alle formule già viste per l'analisi del grafico RP-D si ha: $rp_{j(max)} = rKd_{max}$ e prendendo il caso dello standard IEEE.745 $d_{max} = 2 - 2^{-m}$ se la normalizzazione è di altro tipo $d_{max}= 1 - 2^{-m}$. Quindi

$$rp_{j(max)} = [rK(2 - 2^{-m})2^{f}] / 2^{f}$$
$$rp_{j(min)} = - [rK(2 - 2^{-m})2^{f}] / 2^{f} - e_{p(cs)}$$

Si ottiene:

$$i \geq \log_2(rp_{j(max)} - rp_{j(min)})$$

Stabiliti i numero di bit minimo al ingresso della tabella in relazione alla minima precisione richiesta, ci si trova con un numero definito di valori molto inferiore a tutti i valori permessi, tra i quali ci sono degli intervalli di incertezza. Dalla teoria sviluppata sul grafico RPD questi intervalli di incertezza incrociati sul grafico diventano dei rettangoli. Se tutti i rettangoli sono in un area di selezione tale per cui il valore del quoziente è ben definito allora questa suddivisione quindi questa precisione è accettabile. Il quoziente è ben definito se il rettangolo è totalmente in un area di selezione oppure in un area di selezione e una area sovrapposta , non lo è se è a cavallo di un area sovrapposta e comprende due area di selezione.

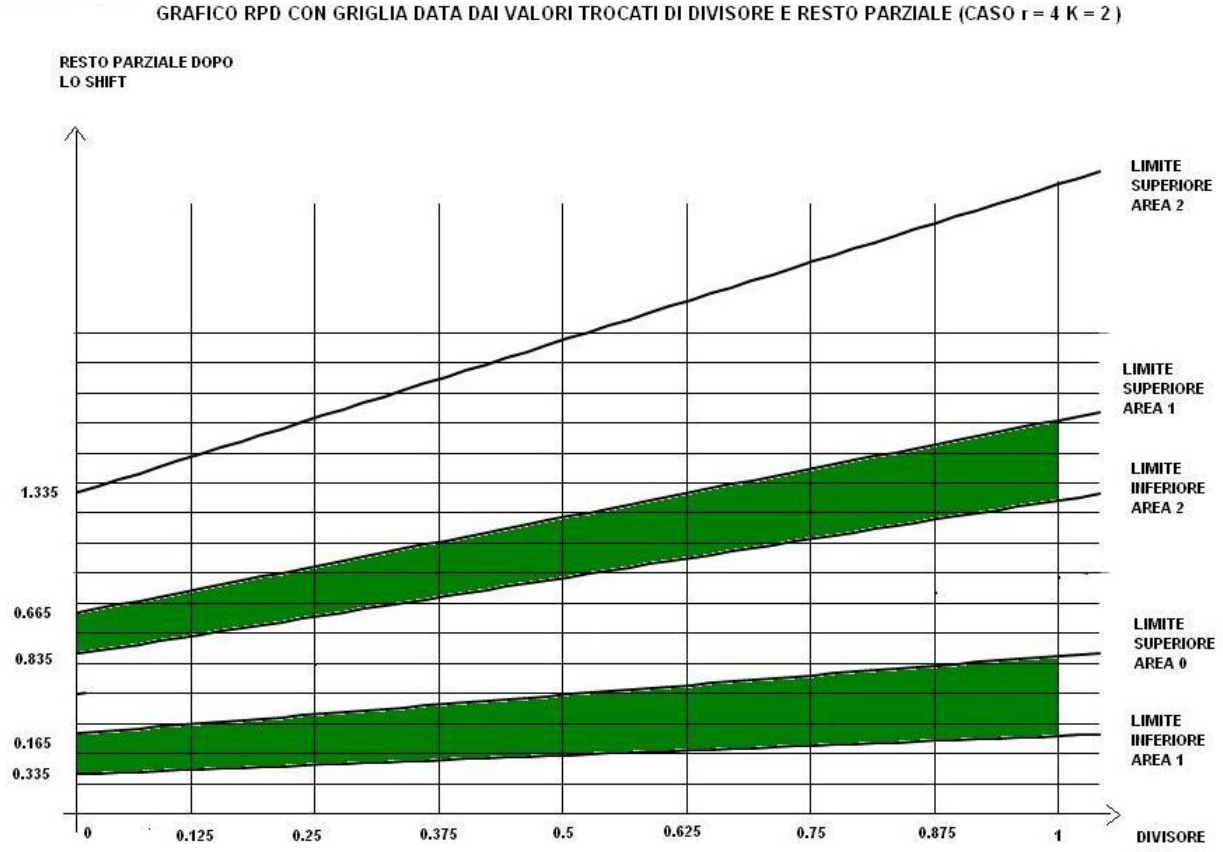

Fig. 13 :Grafico RPD con evidenziata la griglia data dal utilizzo di valori troncati di resto parziale e divisore e le zone di sovrapposizione.

Si vede dal grafico che se una zona è troppo larga può andare ad invadere due aree dove è ben definito il valore del quoziente in quel caso non si può utilizzare questo grado di errore ma deve essere ridotto cosi la regione di incertezza viene ristretta. Questa procedura richiede di fare dei tentavi finché non si riesce a trovare una precisione adeguata per divisore e resto parziale tendo in considerazione che la minima è ben precisa e definita dalla larghezza e altezza minime delle aree sovrapposte. Per fare il controllo basta andare a controllare i punti più esposti ad errore data la pendenza del grafico come l'angolo in basso a destra e quello in alto a sinistra di ogni rettangolo per valori positivi del resto e quello in alto a destra e in basso

a sinistra per valori negativi del resto.

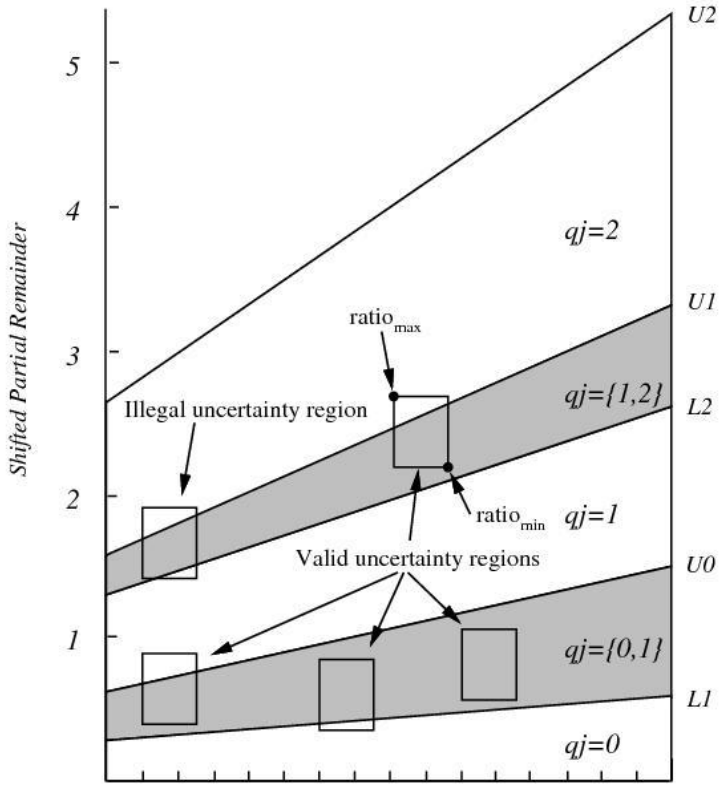

Fig. 14:Schema RPD dove sono evidenziate le zone di incertezza valide e non valide.

Riduzione complessità tavola di selezione

La grandezza della tavola è un fattore determinante per le prestazioni del divisore perché essendo un circuito combinatorio con molti ingressi introduce un notevole tempo di latenza. Un primo modo per abbassare il numero di ingressi della tavola è quello di riutilizzare la stessa tavola si per valori positivi sia per valori negativi, questo può essere fatto trasformando la rappresentazione in complemento a due che abbiamo visto portare un vantaggio in termini di ingressi, con una rappresentazione in modulo e segno permettendo alla stessa tavola di essere utilizzata sia per valori positivi che per valori negativi. Naturalmente questa riduzione non viene gratis perché è richiesto un banco di funzioni XOR per trasformare la rappresentazione da complemento a due a modulo e segno, aumentano la complessità e la latenza. Nel caso di rappresentazione ridondante, come visto prima, quando nella tabella introduco un resto parziale con rappresentazione carry save si commette un errore $e_{p(cs)} = 2^{-(f-1)}$ più grande che nel caso di rappresentazione non ridondante $e_p = 2^{-f}$, è possibile ridurre l'errore del primo caso fino al massimo all'errore del secondo caso. Questo avviene utilizzando come ingressi del carry save non solo gli f bit che servono ma un numero g con g > f. Nella tabella entrano comunque solo i primi f. Con questo trucco si riduce l'errore perché nella tabella non tronco più su m bit ma bensì sui g bit al uscita del sommatore. Il massimo errore sarà allora:

$$e_{p(adder)}=2^{-g} + 2^{-g} – 2^{-n} - 2^{-n}$$

gli ultimi due termini sono dovuti al fatto che quello e l'errore più grosso che si può compiere, se l'ennesimo termine fosse 0 non ci sarebbe errore.

Quindi sulla tabella si vede:

$$e_{p(cs)} = 2^{-f} - 2^{-g} + e_{(adder)}$$
$$e_{p(cs)} = 2^{-f} + 2^{-g} - 2^{1-n}$$

Si vede che g=f l'errore rimane quello di prima mentre se g = m ottengo l'errore minimo cioè 2^{-f}. Con questo metodo si riduce l'errore quindi viene diminuita l'altezza delle regioni di incertezza quindi a parità di bit entranti nella tabella la tabella sarà più veloce, nella realizzazione bisogna tenere conto del compromesso da fare sul sommatore con riporto che deve fare la somma di un numero maggiore di bit.

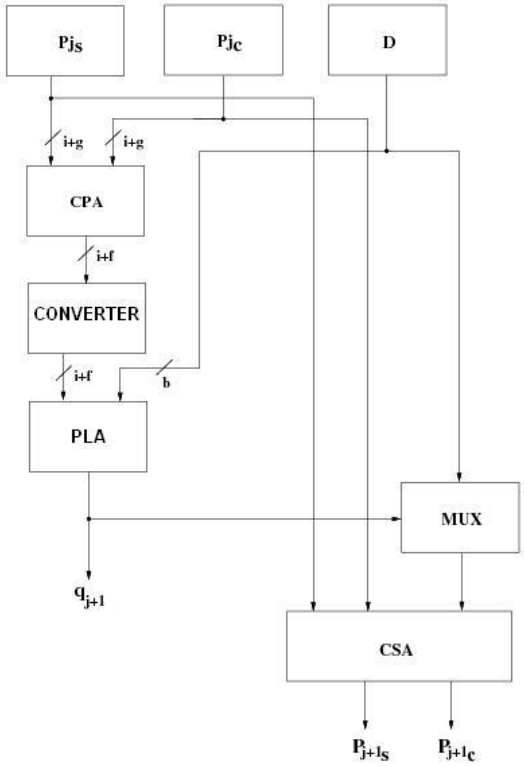

Fig. 15: Schema con CPA e convertitore a monte della tabella di selezione.

Conversione a complemento a 2 da cifre ridondanti

L'approccio generale per fare la conversione è dividere il numero in due registri uno contenente tutte le cifre positive e l'altro negative naturalmente nelle loro posizioni significative poi tramite sommatore fare l'unione. Questo procedimento però richiede la somma dei registri una volta ottenuti tutti valori. Siccome potrebbe essere un peso troppo alto in termini di tempo si fa solitamente un altro tipo di algoritmo, detto conversione al volo.

Le cifre del quoziente vengono convertite appena sono prodotte dal circuito per la selezione del quoziente, questo tramite la formazione su due registri ad ogni iterazione, di due possibili quozienti parziali che di ciclo e in ciclo vengono selezionati in base alle specifiche descritte in seguito. Questo meccanismo ha un ritardo che rimane contenuto nel tempo per eseguire il passo iterativo quindi proporzionale al tempo per eseguire una somma con un CSA. Un numero in forma ridondante e la sua forma normalizzata sono del tipo:

$p = \sum p_i \cdot r^{-i}$ forma con radice superiore a due

$q = q_0 + \sum q_i \cdot 2^{-i}$ forma in complemento a due

da qui per fare la conversione si creano due registri che vengono aggiornati ad ogni nuovo elemento da convertire quindi con l'elemento di posizione K:

$q[k]= A[k-1] + p_k \cdot r^{-k}$ $\quad p^k >=0$
$q[k]= B[k-1]+ (r - |p_k|) \cdot r^{-k}$ $\quad p^k < 0$

dove

1] $A[k-1]= q[k-1]$
2] $B[k-1]= A[k-1]-r^{-(k-1)}$

Verifica per induzione che le relazioni sopra scritte portano al giusto aggiornamento del registro q.
Caso base:
$A[1] = > p_1 r^{-1}$ $\quad (0,p_1)$ $\quad p>0$
$= > -|p_1|r^{-1}$ $\quad (1,r-|p_1|)$ $\quad p<0$

$B[1] = > (p_1-1)r^{-1}$ $\quad (0,(p_1-1))$ $\quad p>0$
$= > -(|p_1|+1)r^{-1}$ $\quad (1,(r-|p_1|-1))$ $\quad p<0$

Dove $p = p^0$.
Formule ricorsive:

$A[k+1] = > A[k] + p_{k+1}r^{-(k+1)}$ $\quad p_{k+1} \geq 0$
$= > B[k] + (r-|p_{k+1}|)r^{-(k+1)}$ $\quad p_{k+1} < 0$

$B[k+1] = > A[k] + (p_{k+1}-1)r^{-(k+1)}$ $\quad p_{k+1} > 0$
$= > B[k] + ((r-1) -|p_{k+1}|)r^{-(k+1)}$ $\quad p_{k+1} \leq 0$

Verifica formule ricorsive:

$A[k+1] = A[k] + p_{k+1}$
$= q[k] + p_{k+1}r^{-(k+1)}=q[k+1]$ Verifica 1]

$A[k+1] = B[k] + (r-|p_{k+1}|)r^{-(k+1)}$
$= A[k] – r^{-k} + r\, r^{-(k+1)}-|p_{k+1}|r^{-(k+1)}$
$= A[k] -|p_{k+1}|r^{-(k+1)} = q[k+1]$ Verifica 1]

$B[k+1] = A[k] + (p_{k+1}-1)r^{-(k+1)}$
$= A[k+1] - r^{-(k+1)}$ Verifica 2]

$B[k+1] = B[k] + ((r-1) -|p_{k+1}|)r^{-(k+1)}$
$= A[k] - r^{-k} +(r-1)r^{-(k+1)} -|p_{k+1}|r^{-(k+1)}$
$= A[k] - r^{-(k+1)} -|p_{k+1}|r^{-(k+1)}$
$= A[k+1] -r^{-(k+1)}$ Verifica 2]

Esempio di conversione radix 4
considerando una rappresentazione a radix 4 minimamente ridondante si ha p_i= {-2,-1,0,1,2} quindi servono tre bit per rappresentarla su un normale calcolatore ad esempio in una rappresentazione modulo e segno si può scrive re 000 -> 0 001->1 010->2 101->-1 110->-2 dove il primo bit s è il segno mentre m0 ed m1 sono i bit che determinano il modulo. Quindi ad ogni cifra che viene trovata nel divisore si shifta di 2 bit a sinistra i bit dei registri A e B e viene caricata la configurazione esatta data per esempio da un circuito combinatorio che data la cifra in rappresentazione ridondante ne trova il corrispettivo da immettere nel quoziente definitivo

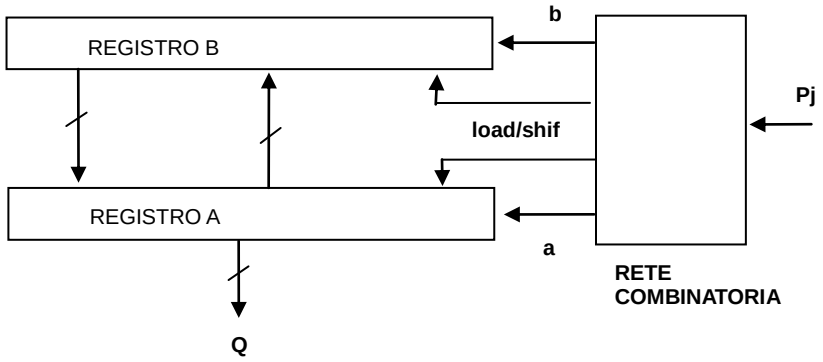

Fig. 16:Schema implementazione convertitore al volo nel quale entra il j-esima cifra Pj.

Però lo standard IEEE per i numeri in virgola mobile prevede che ci sia un arrotondamento del quoziente ottenuto, questo potrebbe richiedere la somma di 1 ulp cioè di una cifra meno significativa se venisse fatto alla fine del ciclo di iterazioni porterebbe a un grosso svantaggio perché potrebbe anche richiedere la propagazione del riporto su tutto il quoziente. Allora viene proposta una tecnica al volo anche qui con l'utilizzo di un ulteriore registro dove salvare la versione arrotondata del quoziente cioè:

$$C[k]= A[k] + r^{-m}$$

In questo caso deve avvenire l'aggiornamento di questo registro secondo le modalità

C[k+1] = > shlC[k] +0 se p_k=r-1
= > shlA[k] +p_k +1 se p_k> -1 p_k<=r-2
= > shlB[k] +r -|p_k|+1 se p_k< -1

In questo registro è contenuto il valore del quoziente convertito nel caso sia richiesta la somma di 1 ulp alla fine, ma questo lo decide il controllo della fpu.

MODIFICHE DELLA STRUTTURA PER MIGLIORARE LE PRESTAZIONI

Struttura multipla

Con l'obbiettivo di rimuovere più bit ad ogni iterazione può essere utilizzata la semplice tecnica di replicare il circuito che fa la divisone di base n volte con l'obbiettivo di andare a moltiplicare per n la radice. Questa tecnica è semplice e immediata, ma ha lo svantaggio di replicare anche tutto l'hardware quindi poco è efficiente. Comunque il vantaggio è limitato rispetto alla versione base perché ogni divisore deve attendere che il precedente finisca di eseguire le operazioni. Può permettere di risparmiare qualche registro ed essere utilizzata nei casi in cui il clock del sistema è troppo lento cosi invece usare un clock più veloce o sprecare del tempo, si utilizza questa struttura.

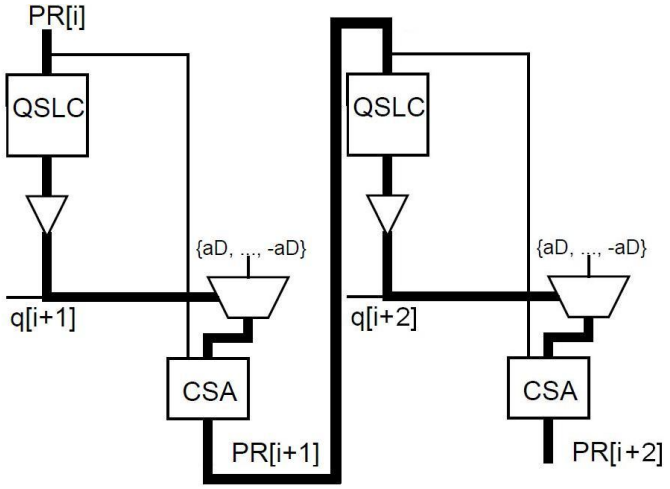

Fig.17: Schema della divisore con struttura multipla

Sovrapposizione delle operazioni

A partire dallo schema elementare maggiormente inefficiente in cui tutte le operazioni vengono eseguite in sequenza si possono creare a discapito dell'occupazione di area degli schemi che tramite parallelizzazione riducono i tempi per eseguire le varie parti del circuito. Nei i vari schemi presentati vengono inseriti più blocchi in serie, ogni blocco viene diviso in due parti cioè la parte di selezione del quoziente e la parte di generazione del prossimo resto parziale che lavorano all` interno dello stesso blocco. Siccome l'operazione di divisone nasce sequenziale per permettere la parallelizzazione al interno di ogni blocco c'è la sovrapposizione ne vari blocchi di più stage di radice più bassa rispetto alla radice del blocco pertanto ammettendo che un divisore di questo tipo ritiri b bit per ogni stage di radice r all' interno di un blocco e ammettendo che ci siano s stage per blocco e m blocchi si ottengono un numero di bit ritirati ogni iterazione pari a $b'=m \cdot s \cdot b$ quindi al variare dello schema si possono ottenere divisori di radici diverse da quelle di partenza: $r' = 2^{b'}$ di conseguenza il numero di iterazioni necessarie a completare la divisione diventa: $k' = n/b'$

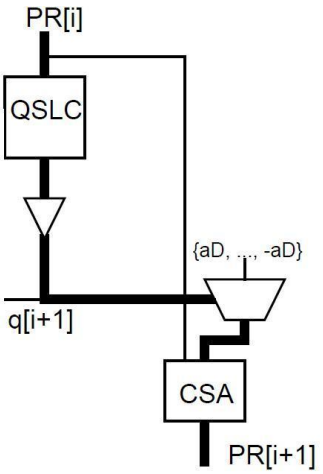

Fig. 18: Schema divisore base.

In questo schema si vede che non è stata attuata nessuna tecnica di sovrapposizione, questo schema viene usato in applicazioni in cui non è possibile andare a utilizzare ulteriore spazio oltre al minimo indispensabile quindi a basso costo, si vede che il percorso critico è quello che passa per la selezione del quoziente e successivamente in sequenza per il sommatore tutto per generare solo un bit, il tempo critico diventa $T_{block} = t_{qs} + t_{buf} + t_{mux} + t_{csa}$.

Sovrapposizione della selezione del quoziente

Per dare senso alla sovrapposizione della selezione del quoziente è necessario immettere almeno due stage nel blocco altrimenti la parallelizzazione perde di significato. In questo schema mentre uno stage sta generando la cifra del quoziente un secondo stage sta già cominciando ad elaborare anche la sua cosi quando il primo stage finisce la generazione del resto parziale il secondo stage ha già eseguito la selezione del quoziente.

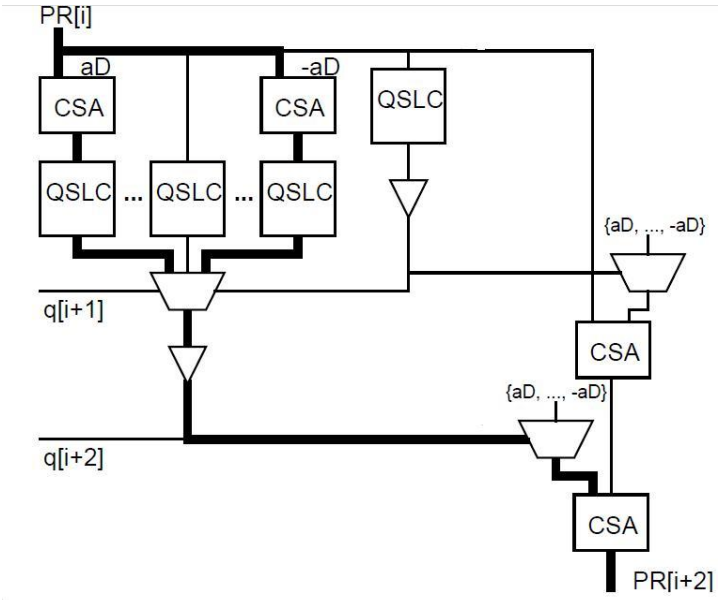

Fig. 19: Schema divisore con sovrapposizione della fase di selezione del quoziente

come si vede mentre viene generato la cifra del quoziente del primo stage il secondo fa delle somme tramite carry save adder, esiste un circuito parallelo per ogni cifra del quoziente che può essere generata nel primo stage. Appena tale valore è disponibile viene selezionata la cifra del quoziente del secondo stage che poi deve essere utilizzata per generare il resto parziale del secondo stage. Il numero di bit del sommatore usato per creare i bit corretti da inserire nel secondo stage può anche avere una precisione ridotta quanto basta alla tabella di selezione. Con questa struttura del blocco che prevede due stage si elaborano ad ogni iterazione il doppio dei bit rispetto quelli che riuscirebbe a elaborare un singolo stage. Per fare un esempio ammettendo che ogni stage ha radice 4 il blocco risulta radice 16, e se due blocchi fossero messi in sequenza si otterrebbe un divisore radice 256. Queste alte radici però sono irraggiungibili per l'aumento considerevole del area utilizzata.
In questo schema il percorso critico è quello per generare l'ultimo resto parziale quindi:

$$T_{block} = (t_{csa(ridotto)} + t_{qsl}) + t_{buf} + 2t_{mux} + t_{csa}$$

Sovrapposizione formazione del resto parziale

Un ragionamento analogo alla sezione che riguarda la sovrapposizione della selezione del quoziente si può fare nella sezione relativa alla formazione del resto parziale, come si vede dallo schema per ogni stage vengono elaborati tanti resti parziali quante sono le cifre possibili nel quoziente per quello stage, questo comporta una replica di sommatori carry save il valore corretto poi vine selezionato tramite multiplexer, il vantaggio è che mentre si fa la selezione del quoziente posso essere elaborati anche i resti parziali. Questa configurazione è possibile anche con un solo stage alla volta e serve per eliminare dal cammino critico il sommatore che può essere lento quanto una tabella di selezione. Il percorso critico risulta:

$$T_{block} = t_{qsl} + t_{mux} + t_{buf.}$$

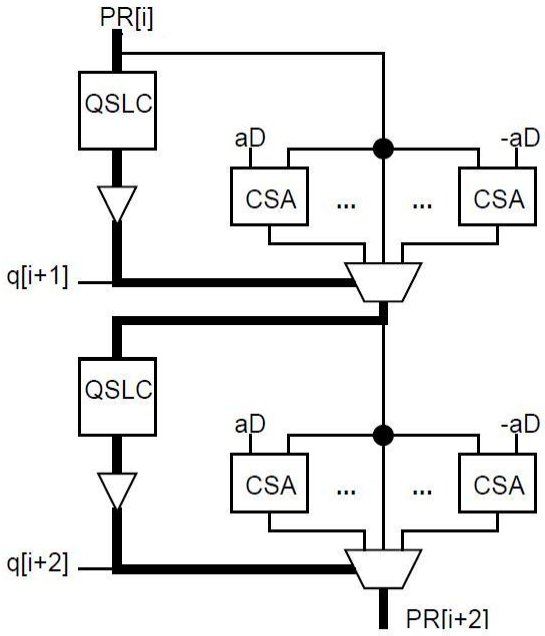

Fig. 20:Schema divisore con sovrapposizione della fase di generazione resto parziale.

Sovrapposizione di selezione del quoziente e formazione del resto parziale

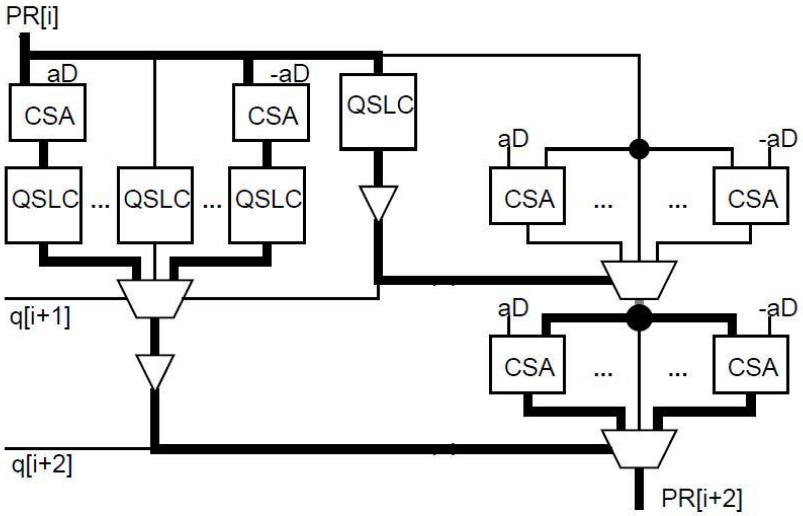

Fig. 21:Schema divisore con sovrapposizione doppia

Combinando i vantaggi delle due soluzioni è possibile ottenere un risultato più veloce di entrambe le configurazioni ma è vero che anche gli svantaggi maggiori. È necessario utilizzare un doppio stage per avere il vantaggio nella selezione del quoziente e diventa necessario introdurre il doppio di sommatori nella parte di generazione del resto parziale.
Dallo schema osserva che esistono due cammini critici uno che passa per la selezione nel primo stage e finisce a generare il resto parziale del secondo stage un altro con le stesse tempistiche che parte dalla generazione del quoziente dal secondo. Il secondo può risultare più veloce se il sommatore in testa al cammino viene implementato con un numero minore di bit data la ridotta precisione delle tabelle. In generale allora:

$$T_{block} = t_{csa} + 2t_{mu\ x} + t_{buf}\ t_{qsl}.$$

Riduzione della radice

Un divisore con radice elevata posso essere realizzati più efficacemente che con la tecnica diretta con o senza sovrapposizione delle varie fasi, questo attraverso l'unione di due divisori a radice più bassa messi in cascata. Diversamente da semplice moltiplicazione degli stage in cui tra uno stage e un altro avviene lo shift del resto parziale quindi risulta una semplice replicazione del hardware, in questo caso si utilizza lo stesso resto parziale per i due stage con la differenza che nel primo si riduce l'intervallo di valori del resto parziale per fare in modo che sia accettabile dallo stage successivo. Praticamente le cifre del quoziente vengono decise da due sottrazioni successivamente senza compiere nessuno shift. E' il caso del divisore radice di otto presentato in [] questo è composto da un normale divisore radice di 4 SRT minimamente ridondante quindi le sue cifre sono [-2,-1,0,1,2]. Si vede dalle proprietà precedentemente illustrate che il resto parziale prodotto da questo divisore ad ogni ciclo deve essere ristretto nel range 2/3D, però nel radice di 8 ad ogni l'iterazione vengono elaborati 3 bit che corrispondono ad uno shift a sinistra di 3 cioè una moltiplicazione per 8, ne consegue

che la restrizione dopo lo shift diventa 16/3D non è contenuto nel range per un divisore radice di a cioè 8/3D. Per questo prima di immettere il resto parziale nel divisore radice di 4 viene eseguita una riduzione del range che va a togliere per cosi dire un bit dal resto parziale il più significativo. Questo divisore radice di due particolare ha come cifre del quoziente [-4,4] quindi va ha togliere o 4D o -4D da resto parziale, questo permette di far rientrare agevolmente il resto parziale nel range 8/3D. Per fare la selezione del primo bit del quoziente basta determinare se il resto parziale è positivo o negativo e se è compreso nel range -2/3D, per far ciò è sufficiente analizzare le prime tre cifre del resto parziale. Questa configurazione porta il vantaggio che per trovar il primo bit basta un confronto di segno per essere determinato $t_{block}=t_{reg}+t_{sign}+t_{mux}+t_{csa}+t_{shortcpa}+t_{qsel}+t_{mux}+t_{csa}$ naturalmente questo schema si presta molto alla sovrapposizione soprattutto nella generazione del quoziente ma anche del primo resto parziale. Ne risulta alla fine un divisore radice 8 con coefficiente di ridondanza k=6/7 dato che le cifre di quoziente generate nei due divisori q_h=[-4,4] q_l=[-2,-1,0,1,2]. Questo schema risulta particolarmente vantaggioso se è necessario realizzare anche un circuito per la radice quadrata nel processore e si presta molto bene a integrare i due circuiti condividendo i sommatori e le tavole di selezione del quoziente solo con l'aggiunta di qualche multiplexer.

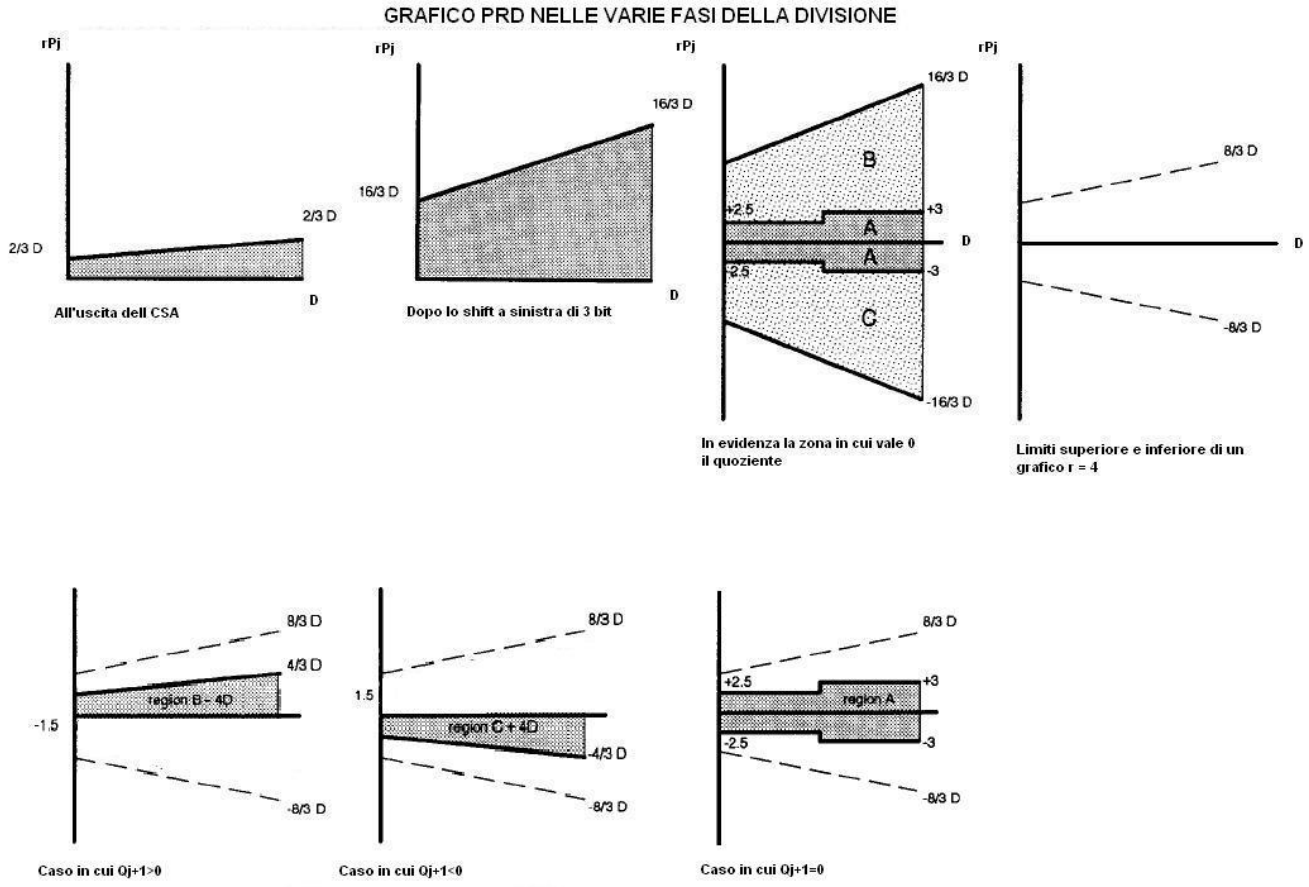

Fig. 22: Grafici RPD che mostrano l'intervallo di valori dopo aver sommato ± 4D al resto parziale

RIDUZIONE DEL CONSUMO DI POTENZA

Il consumo di potenza è una richiesta sempre più frequente per chi realizza un circuito integrato a causa del incremento della densità dei transistor e della frequenza di funzionamento dei circuiti. Ci sono vari accorgimenti da prendere, a partire dalla configurazione del circuito per fare in modo di isolare il percorso critico cosi da utilizzare componenti più veloci solo per quella parte e rilassare il resto del circuito con componenti meno performanti e onerosi, in secondo luogo usare accorgimenti per evitare glitch che causano sprechi inutili anche se quest'ultimo problema non è sempre e facilmente risolvibile. Se il consumo di potenza è importante bisogna abbandonare l'idea di usare logiche dinamiche come domino dual rail sono difficile da usare, anche se potrebbero esser utilizzate solo in alcune parti. Per ottimizzare bisogna analizzare due aspetti del circuito la parte iterativa che esegue l'algoritmo e la parte di arrotondamento e conversione, naturalmente se si usa una rappresentazione ridondante.

Accorgimenti per ridurre la potenza nel circuito iterativo

Come è stato descritto nel algoritmo per la divisone i bit del divisore e resto parziale che devono essere analizzati nella PLA per la selezione del quoziente sono solo una parte che dipende dalla ridondanza, e come si vede dallo schema generale il quoziente viene analizzato subito, perché il sommatore CSA deve aspettare che venga selezionato un quoziente. La modifica principale consiste di spostare la selezione del quoziente nella ultima fase del ciclo precedente invece che nella prima dell'attuale, questo prevede di avere al inizio del ciclo successivo il quoziente pronto quindi potrà essere selezionato il divisore e fatta subito la somma sul CSA per tutti i bit del resto parziale. Quindi sarà necessario predisporre un registro per salvare il quoziente regolato dallo stesso clock del divisore. Lo scopo della modifica è rendere indipendente la selezione del quoziente dal resto delle operazioni, con questa modifica si ottiene anche il risultato di regolare l'uscita della PLA che può portare dei glich sul multiplexer e sul sommatore.

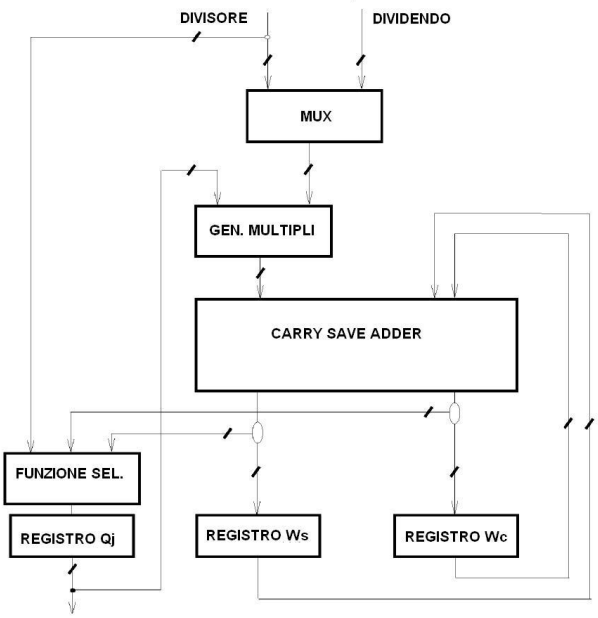

Fig. 23:Schema con modifica alla posizione del blocco selezione del quoziente.

La presenza di questa modifica introduce un ciclo di latenza in più perché al primo ciclo il quoziente non è disponibile e con il trucco di impostare il quoziente a 1 e i registri di somma e riporto del sommatore a 0, il primo ciclo il generatore di multipli lascia passare il dividendo che viene sommato a 0 quindi salvato sui registri. I bit più significativi del resto parziale che oltre a essere sommati devono passare per la PLA sono i bit più lenti e critici rispetto agli altri, viene in mente quindi che non ha senso eseguire tutti bit nel CSA al massimo delle prestazioni ma solo i più significativi.

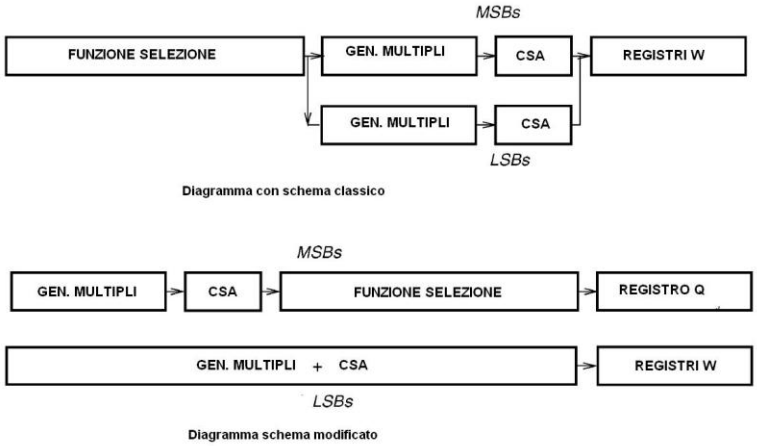

Fig. 24:Diagramma che mette in evidenza i percorsi diversi da MSBs e LSBs.

Circuiti meno veloci significano meno consumi meno complessità e in generale meno costi. La diminuzione delle prestazioni viene ottenuta usando logiche cmos statiche e diminuendo le capacità in eccesso, necessarie a pilotare velocemente delle porte, dove il percorso non è critico.

Modifica al CSA

Siccome i registri sono una componente molto dispendiosa, per ridurre i consumi potrebbe essere vantaggioso diminuire la dimensione dei registri che contengono somma e riporto. Per far questo è possibile diminuire le cifre del riporto assegnando un bit di riporto a una serie di cifre e non più un riporto per ogni cifra, questo può essere fatto modificando il CSA. Un CSA comune è formato da tanti full adder che generano, dati i bit di somma e riporto più multiplo del divisore, la prossima somma e resto parziale. La modifica consiste di utilizzare piccoli sommatori con riporto che sommano pochi bit ognuno, ma con cosi facendo non c'è bisogno di tenere tutte le cifre del riporto, risparmiando quindi in registri per salvarle. Ad esempio sapendo che per rappresentare una cifra in un radice 4 servono $\log_2 r$ bit più il riporto allora potrebbe essere creato un riporto per ogni cifra e non più per ogni bit. Questo implica l'introduzione di ulteriore ritardo sul sommatore quindi questa tecnica grazie alle proprietà del nuovo schema visto in precedenza può essere applicata solo ai bit meno significativi del sommatore che non si trovano sul percorso critico.

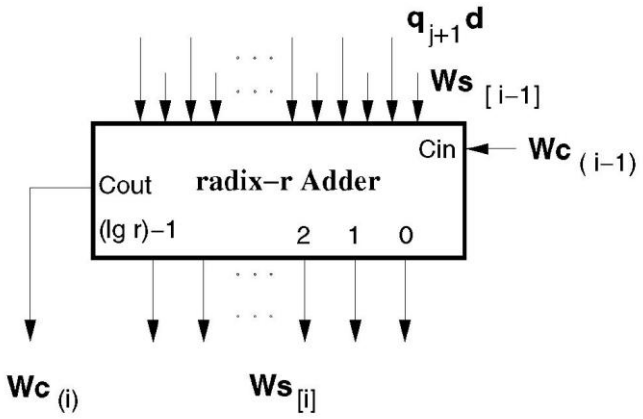

Fig. 25: Nuovo sommatore con riporto che somma più cifre alla volta.

Doppia tensione

Siccome l'energia consumata da un circuito dipende dal quadrato della tensione di alimentazione diminuendo la tensione di alimentazione diminuiscono le prestazioni quindi nelle celle che non hanno bisogno di essere particolarmente performanti è possibile utilizzare questo metodologia. Il passaggio di un bit da una parte del circuito a tensione minore a una altra a tensione maggiore necessita di un consumo di energia durante il ciclo che non ci sarebbe nel caso di tensione uniforme, questa componente va presa in considerazione per valutare i vantaggi. Per limitare la latenza conviene eseguire il cambio di tensione un sul bit prima del bit che poi dovrà andare veloce cosi il circuito veloce non dovrà perdere tempo ad aumentare la tensione del suo primo bit perché è già stato fatto nel circuito più lento.

Ridurre i glich con equalizzazione dei percorsi

Rendendo le tempistiche dei vari percorsi dei segnali che entrano in un blocco logico di uguale durata si ottiene il risultato di ridurre il numero di glicth e quindi diminuire le perdite. E' il caso per esempio del ingresso di un blocco CSA che riceve i segnali dal blocco che crea i multipli del divisore/dividendo, senza essere interfacciato da nessun registro può capitare che dei segnali arrivino al ingresso del blocco con tempi diversi creando commutazioni indesiderate nel blocco. Una soluzione per sincronizzare gli ingressi potrebbe essere quella di rallentare leggermente il clock dei registri dai quali partono i segnali più veloci per esempio nel caso del sommatore i registri che contengono le cifre del resto parziale che devono aspettare mentre il divisore crea i sui multipli.

Spegnere le parti non attive

Per esempio il rilevatore di segno e zero che determina se il resto della divisone è positivo negativo o nullo deve funzionare solo alla fine del processo quando nel registro è salvato l'ultimo resto parziale quindi nelle altre iterazioni può rimanere spento. Questo avviene per esempio forzando un valore logico costante al ingresso del blocco che cosi teoricamente rimane congelato in realtà le correnti di perdita dei transistor farebbero consumare del energia che ci è contenuta quindi anche la scelta del valore può essere un fattore rilevante.

Andando a vedere i risultati ottenuti in [] si vede che applicando queste tecniche a circuiti di simulazione si ottengono miglioramenti del circa 20% sul energia consumata da un divisore radice di 4 del tipo presentato pur mantenendo le prestazioni elevate, gli autori affermano che si ottengono guadagni energetici simili con divisori a radice più elevata come radice8 e radice16.

BIBLIOGRAFIA

[1] D. E. Atkins, “Higher-radix division using estimates of the divisor and partial remainders,” IEEE Trans. Computers, vol. C-17, no. 10, Oct. 1968.
[2] K. G. Tan, “The theory and implementation of high radix division,” in Proc. 4th IEEE Symp. Computer Arithmetic, pp. 154–163, June 1978.
[3] S. F. Oberman and M. J. Flynn, “Division algorithms and implementations,” to appear in IEEE Trans. Computers, 1997.
[4] J. Fandrianto, “Algorithmfor high-speed shared radix 8 division and radix 8 square root,” in Proc. 9th IEEE Symp. Computer Arithmetic, pp. 68–75, July 1989.
[5] M. D. Ercegovac , T Lang, On-the-fly conversion of redundant into conventional representations, IEEE Transactions on Computers, v.36 n.7, p.895-897, July 1987.
[6] A. Nannarelli and T. Lang. "Low-Power Divider", IEEE Transactions on Computers, Vol. 48, pages 2-14, January 1999.
[7] D. Harris, S. Oberman, and M. Horowitz, “SRT division architectures and implementations,” in Proc. 13th IEEE Symp. Computer Arithmetic, July 1997, pp. 18–25.
[8] Measuring the Complexity of SRT Tables, Stuart F. Oberman and Michael J. Flynn, November 1995.
[9] B. Parhami, “Precision Requirements for Quotient Digit Selection in High-Radix Division,” Proc. 35th Asilomar Conf. Signals, Systems, and Computers,
pp. 1670-1673, Nov. 2001.
[10] Saurabh Upadhyay and James E. Stine “*Pipelining High-Radix SRT Division*”, Electrical and Computer Engineering Department Oklahoma State University Stillwater, 2007.