RELAZIONE PROGETTO FINALE ELETTRONICA DEI SISTEMI DIGITALI

Luigi Lella

A.A. 2021/2022

Introduzione.

Il progetto prevede che venga realizzata un'unità funzionale che calcoli, forniti i tre ingressi A, B e C,

$$A/B + C.$$

Gli operandi in ingresso e il risultato sono da considerare come numeri fixed point.

La stessa operazione è stata realizzata seguendo due diversi approcci, il primo utilizza un blocco specifico per effettuare la divisione mentre il secondo è una soluzione approssimata che non prevede l'utilizzo del blocco divisore ma riduce il problema ad una moltiplicazione per una costante e differenza.

Verranno poi analizzate e discusse le prestazioni delle varie soluzioni.

Fase 1.

In questa prima fase viene utilizzato un divisore combinatorio per effettuare la divisione tra A e B, il risultato viene poi sommato a C.

- A è un numero W(16, 8, 8) range 100 – 205.
- B è un numero W(11, 3, 8) range 3,75 – 6.
- C è un numero W(16, 6, 10) range 0 – 63.

Quando andiamo ad effettuare una divisione tra numeri fixed point il numero di bit usato per rappresentare la parte frazionaria della soluzione sarà uguale alla differenza tra il numero di bit della precisione usati per il dividendo e quelli usati per il divisore.

Nel nostro caso, ad esempio, A/B (W(16,8,) / W(11,3,8)) produrrebbe un risultato con 8-8 = 0 bit per la parte frazionaria. Andando a lavorare con un risultato in questa forma il numero prodotto in uscita avrebbe una precisione troppo bassa in quanto il valore assunto dalla parte frazionaria del risultato dipenderebbe solo ed esclusivamente dal valore degli ultimi 10 bit dell'operando C.

Per aumentare la precisione allora, al momento del campionamento, ad A vengono concatenati a destra 10 bit, tutti uguali a '0', (operazione che corrisponde ad una moltiplicazione per 2^{10}). A_int in questo caso sarà interpretato come un numero W(26, 8, 18), permettendo di ottenere un risultato per la divisione con 18-8 = 10 bit di precisione per la parte frazionaria.

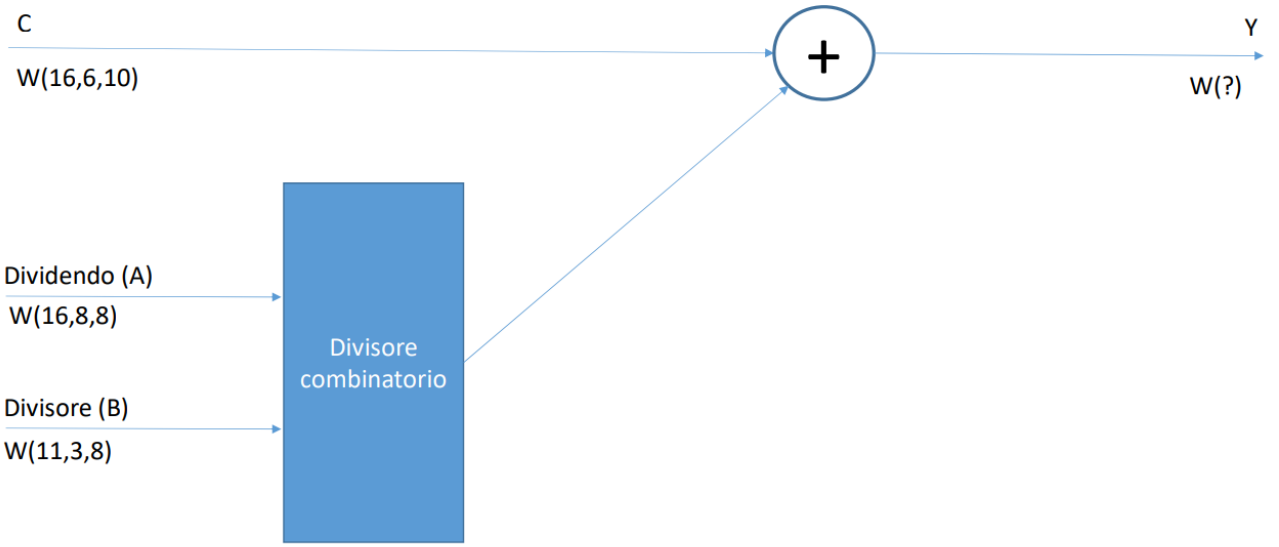

Figura 1-Datapath 1

Per la scelta del numero di bit da utilizzare per esprimere il risultato finale è stato considerato il caso peggiore, ovvero è stato calcolato il risultato massimo ottenibile dallo svolgimento dell'operazione che corrisponde a:

$$\frac{\max(A)}{\min(B)} + \max(C) = \frac{205}{3{,}75} + 63 = 117{,}\overline{6}$$

Per esprimere questo numero sono necessari 7 bit per la parte intera ai quali si aggiungono i 10 bit di precisione per la parte frazionaria che producono un risultato complessivo che sarà nella forma W(17,7,10).

La soluzione in questo caso viene fornita nel clock successivo a quello in cui vengono campionati i dati.

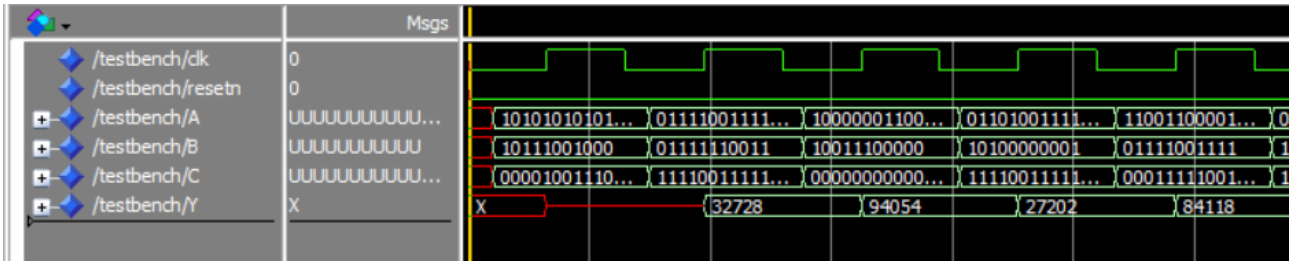

Figura 2 - Simulazione funzionale datapath 1

Inizialmente è stata effettuata la sintesi di questo codice su un dispositivo FPGA della famiglia *Cyclone IV* il "EP4CE6E22C9L" le cui specifiche sono:

- Tensione di alimentazione 1V
- Numero di logic elements 6272
- 92 Pin I/O

I risultati ottenuti dalla compilazione sono riportati nella tabella nella pagina seguente.

Period (ns)	Freq. (MHz)	Latency (ns)	Throughput (MS/s)	Max freq. (MHz) (w.c)	Setup check		Hold check		Total LE	Total reg	Total pins
					P, V,T	Slack (ns)	P, V,T	Slack (ns)			
129	7,75	129	7,75	7,81	Slow,1V,85	0,98	Fast,1V,0	0,41	577	60	62

Figura 3 - Tabella risultati timing datapath 1

La sintesi su questo dispositivo permette di arrivare ad avere un periodo di clock di 129 ns e per come stato scritto il codice la latenza equivale ad un ciclo di clock.

In termini di logic element utilizzati rispetto alla soluzione della fase 2 questa ne utilizza un numero sicuramente maggiore, alla soluzione successiva, in quanto è presente il modulo divisore per svolgere l'operazione che risulta essere molto oneroso.

L'utilizzo di questo elemento ha però un vantaggio fondamentale, i risultati ottenuti avranno un errore relativo molto basso.

Per dare una stima qualitativa dei risultati che si possono ottenere con questo metodo sono state effettuate diverse operazioni i risultati ottenuti sono stati riportati in una tabella.

A	B	C	RISULTATO ATTESO	RISULTATO DATAPATH 1	ERRORE
170,6641	5,78125	2,441406	31,96167652	31,9609375	0,002%
121,8945	3,949219	60,98438	91,84985472	91,84960938	0,000%
129,5039	4,875	0	26,56490385	26,56445313	0,002%
105,8945	5,003906	60,98438	82,14674815	82,14648438	0,000%
204,2344	3,808594	7,782227	61,40684195	61,40625	0,001%
108,2344	4,25	32,5	57,96691176	57,96679688	0,000%
131,0039	4,9375	7,535156	34,06759296	34,06738281	0,001%
184,6875	5,75	49,81641	81,93597147	81,93554688	0,001%

Figura 4 - Risultati analisi precisione

La componente principale dell'errore in questo caso è data dall'errore di quantizzazione ovvero dal fatto

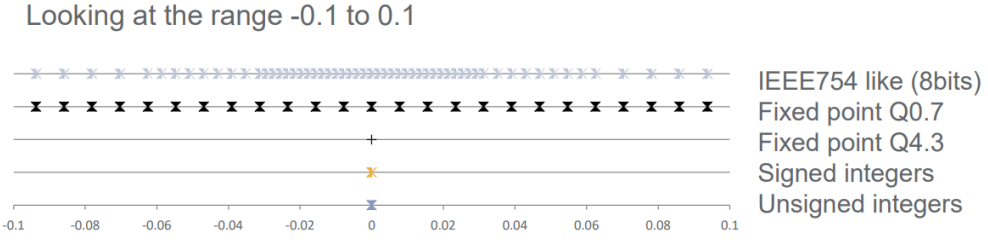

Figura 5 - Esempio di densità dei numeri floating point

che utilizzando una notazione fixed point non si riesce a rappresentare tutti i numeri presenti in un determinato range ma la distribuzione dei numeri possibili risulta più fitta in alcune zone e meno in altre.

Della soluzione in FPGA è stata effettuata anche un'analisi dei consumi nelle condizioni di coefficiente di attività pari a 0.1, temperatura ambiente e ipotizzando di non avere alcun dissipatore.

I risultati ottenuti, espressi in mW, sono:

TOTAL	Core Dyn	Core Static	I/O	Core dyn/freq	En/op
74,24	0,65	41,21	32,38	0,0839	0,0839

In generale le prestazioni che riesco ad ottenere con l'utilizzo di un dispositivo FPGA sono molto inferiori, in termini di area, velocità e consumo, rispetto ad altre soluzioni. Per effettuare un confronto tra le possibili soluzioni è stata effettuata anche la sintesi dello stesso codice utilizzando una metodologia standard cells.

Questo approccio punta a creare un circuito con area e numero di pin di I/O minimi perché utilizza soltanto le celle che sono strettamente necessarie ai fini del nostro progetto e si basa su delle librerie di celle che sono state precedentemente caratterizzate.

Period (ns)	Freq. MHz	Latency in numero di cicli di clock	Throughput (MS/s)	slack (ns)	Area (um^2)	Area (Kgate,eq)	Leakage (uW)	Dynamic Power (uW)	Total Power (uW)	DynP/freq (uW/MHz)	En/op (uW/MHz)
13	76,92307692	1	76,92307692	0.01	5139,652	6,5141343	83,0165	204,5442	287,5605	2,659075	2,659075

Figura 6 - Risultati analisi su Standard Cells datapath 1

Come si può notare dai dati riportati in tabella anche il periodo, rispetto alla soluzione precedente, risulta essere molto più basso (13 ns contro i 129 ns della prima soluzione).

Se ci muoviamo poi ad analizzare il consumo di potenza vedo che questo cala di diversi ordini di grandezza rispetto passando da mW a uW.

Fase 2.

In particolari applicazioni, dove ad esempio bisogna rispettare vincoli di velocità molto stringenti, la latenza introdotta dal divisore utilizzato nella fase 1 potrebbe rendere impossibile avere determinate prestazioni.

Per questo allora una possibile soluzione è quella di andare ad utilizzare un'architettura che vada ad effettuare per il calcolo di1/B un'approssimazione permettendoci di non dover più utilizzare un blocco divisore ma semplicemente un moltiplicatore.

Questa soluzione porta vantaggi in termini di area, latenza e consumo al costo di una ridotta precisione.

Per la curva 1/B viene fatta un'approssimazione lineare a tratti. Sono stati definiti 5 intervalli e in ciascuno di questi la curva viene sostituita da una retta.

Gli intervalli sono:

- 3,75 <= B <4
- 4 <= B < 4,5
- 4,5 <= B < 5
- 5 <= B < 5,5
- 5,5 <= B <= 6

La formula che ci permette di calcolare il valore approssimato di 1/B è la seguente:

$$inverso = baseY - displacement * coeff$$

Per effettuare questo calcolo ho bisogno di avere precedentemente calcolato 5 triplette di costanti, i calcoli sono riportati nel file Excel CalcoloCoeff, ognuna da attribuire ad un intervallo, che in questa fase verranno memorizzate come costanti all'interno del programma in 3 array da 5 elementi ciascuno.

- *BaseX* è l'array che contiene la definizione delle soglie e che verrà usato per individuare l'intervallo nel quale è contenuto il divisore, la dimensione degli elementi contenuti al suo interno è uguale alla dimensione di B.
- *BaseY* è un array che contiene il valore dell'inverso della soglia. La dimensione degli elementi contenuti al suo interno è di 17 bit, vado a considerarli come floating point W(17,0,17). Il numero 17 è frutto di analisi sperimentale, si sono confrontate diverse soluzioni e questa è risultata essere la migliore tra numero di bit usati e precisione.
- *Coeff* contiene invece il coefficiente angolare della retta presente in ogni tratto, anche in questo caso si è deciso di considerare una precisione a 17 bit.

Le soglie scelte sono state in maniera tale da essere, tranne la prima, tutte della stessa dimensione e distanti 0,5.

La scelta di considerare la prima soglia grande la metà delle altre è stata fatta per allineare e far partire al valore 4 le altre e questo ci permette anche di non dover considerare delle soglie di 0,45 numero che non è esprimibile in fixed point.

Per quanto riguarda il calcolo di BaseY e Coeff ci si è affidati a una funzione di excel *REGR.LIN* che forniti in ingresso i valori campionati di x e y di una curva utilizza il metodo dei minimi quadrati per calcolare i coefficienti della retta che meglio approssima, e rende minimo l'errore totale, i valori forniti in ingresso. Questa funzione fornisce il coefficiente angolare della retta, che verrà preso così com'è ed un altro valore che servirà per il calcolo di BaseY.

All'interno del campo architecture è stato definito un processo denominato *soglie* che tramite l'utilizzo di un ciclo for individua in quale range è contenuto il dividendo B e

in base a dove cade quest'ultimo inizializza il valore di index che verrà usato per andare a selezionare i valori corretti di BaseY e Coeff.

Una volta calcolato il valore dell'indice, in diff viene salvata la differenza tra il valore di BaseX e il valore del dividendo B, ovvero quanto quest'ultimo si discosta dalla soglia. Diff è stato definito come un W(11,3,8) ma in realtà i 3 bit di parte intera sono sempre "000" perché la distanza massima tra B e la soglia non sarà mai superiore a 1 quindi dall'operazione successiva verranno considerati soltanto i bit di parte frazionaria.

Il valore di diff viene moltiplicato per il valore del coefficiente producendo un risultato che è W(8,0,8)*W(17,0,17) = W(25,0,25).

Arrivati a questo punto per calcolare il valore di 1/B mi basta semplicemente andare a sottrarre il valore ottenuto al Base Y dell'intervallo che sto considerando. Verranno presi soltanto i 17 bit più significativi di diff per allinearlo e permettere la sottrazione.

Ottenuto 1/B questo viene moltiplicato per A producendo una stringa di bit che va interpretata come W(33,8,25).

Il risultato finale è ottenuto prendendo solo una parte di A*B (W(17,7,10)), i bit che non prendo o sono sempre a 0 o non sono importanti per il calcolo, e a questo sommo il valore contenuto in C.

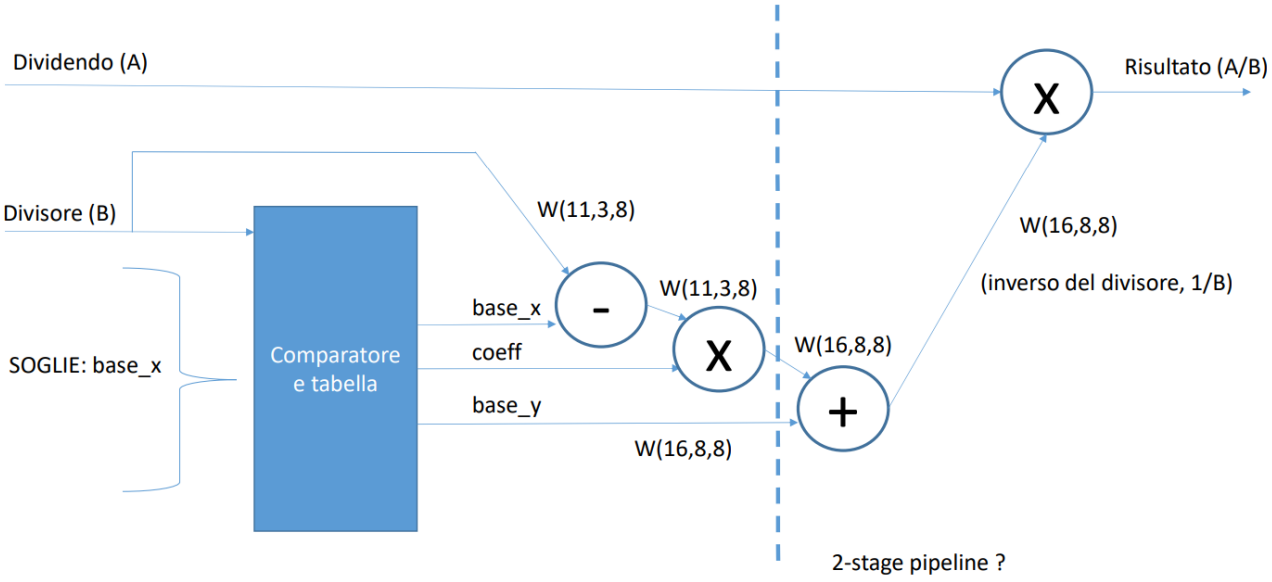

Figura 7 - Datapath 2, schema semplificato

La stessa soluzione è stata poi implementata anche andando ad inserire uno stadio di pipeline nella posizione indicata in figura.

L'algoritmo utilizzato per il calcolo non è stato modificato, sono stati introdotti 4 registri: uno per A, uno per B, uno per BaseY e uno per il risultato della moltiplicazione. Questa soluzione non va ad impattare la precisione del sistema ma aumenta il throughput.

Rispetto alla soluzione precedente, come previsto, cresce l'errore percentuale che però si mantiene sotto lo 0,1% ottenendo comunque un'ottima approssimazione del risultato finale.

A	B	C		RISULTATO ATTESO		RISULTATO DATAPATH 1	ERRORE	RISULTATO DATAPATH 2	ERRORE
170,6641	5,78125	2,441406		31,96167652		31,9609375	0,002%	31,98632813	-0,077%
121,8945	3,949219	60,98438		91,84985472		91,84960938	0,000%	91,82421875	0,028%
129,5039	4,875	0		26,56490385		26,56445313	0,002%	26,57617188	-0,042%
105,8945	5,003906	60,98438		82,14674815		82,14648438	0,000%	82,13085938	0,019%
204,2344	3,808594	7,782227		61,40684195		61,40625	0,001%	61,4140625	-0,012%
108,2344	4,25	32,5		57,96691176		57,96679688	0,000%	57,99316406	-0,045%
131,0039	4,9375	7,535156		34,06759296		34,06738281	0,001%	34,04785156	0,058%
184,6875	5,75	49,81641		81,93597147		81,93554688	0,001%	81,96386719	-0,034%

Figura 9 - Confronto risultati precisione dei primi due datapath

Period (ns)	Freq. (MHz)	Latency (number of clock cycles)	Throughput (MS/s)	Setup check		Hold check		Total LE	Total reg	Total pins
				P, V,T	Slack	P, V,T	Slack			
25	40	1	40	Slow 1000mV 85C	0,227	Fast 1000mV 0C	0,415	92	44	62

Figura 8 - Analisi timing datapath 2 su FPGA

Il calo della precisione però porta ad un notevole aumento in termini di performance, nella simulazione dell'architettura non pipeline su FPGA, il periodo di clock si riduce fino a 25 ns portando ad avere una frequenza di funzionamento di 40 MHz.

Calano molto anche il numero totale di Logic Element utilizzati, in quanto non è più presente il blocco divisore che è molto costoso in termini di area.

L'abbassamento del numero di registri usati è dovuto invece dal fatto che il tool di sintesi utilizza dei registri appartenenti al blocco moltiplicatore, come indicato nel report.

TOTAL	Core Dyn	Core Static	I/O	Core dyn/freq	En/op
77,65	0,99	41,18	35,48	0,0248	0,0248

Per quanto riguarda la potenza, in FPGA, aumenta molto quella dinamica ma questo è dettato dal fatto che il circuito riesce a funzionare a una frequenza molto maggiore rispetto alla prima soluzione.

Andando però a guardare l'energia che viene impiegata per svolgere una singola operazione questa risulta notevolmente abbattuta in quanto il circuito divisore oltre ad occupare più area ha anche dei consumi maggiori.

Period (ns)	Freq. MHz	Latency in numero di cicli di clock	Throughput (MS/s)	slack (ns)	Area (um^2)	Area (Kgate,eq)	Leakage (uW)	Dynamic Power (uW)	Total Power (uW)	DynP/freq (uW/MHz)	En/op (uW/MHz)
5	200	1	200	0.0	2546,95	3,2280735	37,3431	218,8045	256,1475	1,094023	1,094023

Figura 10 - Risultati ottenuti da sintesi su Standard Cells

Anche se andiamo a confrontare i circuiti ottenuti su Standard Cells possiamo notare notevoli vantaggi: sia per quanto riguarda l'area che per il periodo abbiamo una riduzione di un fattore intorno al 2. Riducendo l'area cala anche la potenza di leakage mentre quella dinamica, come nel caso FPGA, cresce di circa 10uW.

Complessivamente però la Total Power si riduce, perché come detto la componente statica si è ridotta di molto, così come si riduce l'energia richiesta per operazione.

Per la soluzione con pipeline, come già detto in precedenza, è bastato andare a introdurre alcuni registri che servono per campionare i risultati parziali dell'elaborazione. I registri introdotti sono A_int_pip, C_int_pip che servono per campionare dei dati che serviranno nella parte finale dell'elaborazione e quindi per come è stata impostata la pipeline devono essere ritardati di un ciclo di clock perché servono nella parte del datapath che interviene nel secondo ciclo.

BaseY_pip e diffcoff_pip campionano rispettivamente il valore di BaseY corrispondente e il prodotto tra il coefficiente angolare e il displacement.

Disponendo la pipeline dopo il calcolo di diffcoeff permette di avere un datapath il più bilanciato possibile, dato che a determinare il periodo di clock è lo stadio più lento.

La soluzione con pipeline permette di abbassare il periodo di clock da 25 ns a 17, nel caso di sintesi su FPGA, e quindi di aumentare la frequenza di funzionamento e il througput da 40 MHz a 58,82.

Ovviamente in questo caso crescono sia il numero di LE usati che il numero di registri che devono essere sintetizzati (i dati sono riportati nel file divisore_approx). Per quanto riguarda la potenza aumenta quella dinamica (stiamo lavorando a una frequenza maggiore) e aumenta anche l'energia richiesta per operazione visto che per completarla servono due cicli di clock.

Anche in standard cells vedo dei risultati molto simili a quelli visti in FPGA, ovvero diminuzione della latenza con conseguente aumento della frequenza di funzionamento, aumento dell'area dovuta all'aggiunta dei nuovi registri oltre che a un aumento generale del consumo di potenza.

Per confrontare la soluzione pipeline con quella non-pipeline solitamente viene definito un parametro chiamato *SPEEDUP* che ci permette capire quanto una soluzione è più veloce rispetto ad un'altra:

$$Speedup = \frac{Tseq}{Tpip} = \frac{ThroughputPip}{ThroughputSeq}$$

Lo speedup nel caso di pipeline perfettamente bilanciata è uguale al numero di stadi nei quali divido il sistema, nel nostro caso 2, se il bilanciamento è imperfetto il valore ottenuto sarà sicuramente minore di questo valore.

Nel nostro caso abbiamo:

- Nel caso FPGA lo Speedup è uguale a $^{25}/_{17} = 1{,}47$
- Nel caso Standard cell lo Speed up è uguale a $^{5}/_{4} = 1{,}25$

Questo vuole dire che la pipeline nel caso FPGA riesce ad ottenere un incremento migliore della velocità della stessa pipeline sintetizzata in standard cells.

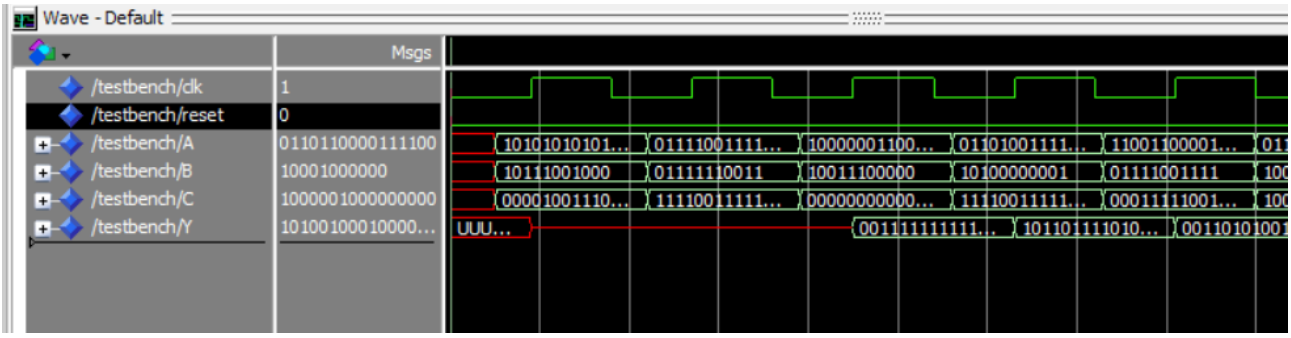

Figura 11 - Simulazione soluzione con Pipeline

Nelle due figure presenti in questa pagina vengono messe a confronto le simulazioni funzionali ottenute dalle due soluzioni, con e senza pipeline.

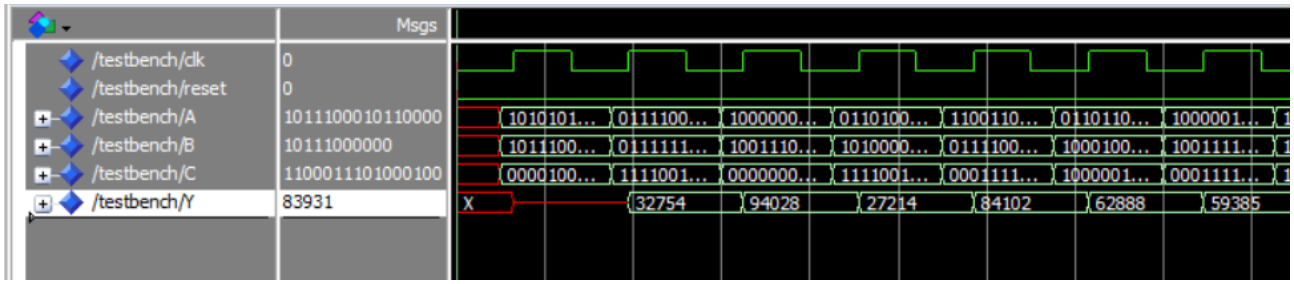

Figura 12 - Simulazione soluzione senza pipeline

Fase 3.

In questa fase sono state aggiunte due novità:

- La possibilità di caricare BaseX, BaseY e coeff su dei registri
- Una FSM per gestire gli ingressi e le uscite

Gestione della fase di Load:

Durante la fase di Load il programma deve andare salvare i coefficienti, che prima venivano definiti come costanti, su dei registri. Le 5 triplette di coefficienti vengono fornite in una fase iniziale dall'utente tramite appositi pin di ingresso.

L'utente, quando pronto, imposta il segnale di LOAD a 1 e contemporaneamente fornisce ad ogni ciclo di clock i valori di BaseX, BaseY e coeff su tre diversi ingressi.

Per permettere un corretto funzionamento di questa fase è stato necessario:

- Andare a definire un contatore, che conta da 0 a 4, che serve per indicizzare la posizione dell'array nella quale devo salvare l'i-esimo coefficiente.
- Un sistema per gestire l'arrivo di un segnale di START durante questa fase. Infatti, se per caso l'utente fornisse il segnale di inizio divisione durante il caricamento si potrebbero avere degli errori. Per ovviare a questo problema si è deciso di rendere il sistema non sensibile al segnale durante la fase di LOAD permettendo al programma di uscire dalla fase di idle ed andare in quella di processing **SOLO SE** il segnale di start è alto **E** quello di Load è basso.

Creazione della FSM per gestire ingressi e uscite:

Si è scelto di realizzare una FSM con 3 stati: idle, processing, data_out.

- IDLE: è uno stato di attesa, il programma aspetta di ricevere il segnale di START per portarsi nella fase di processing. Durante questa fase, come detto prima, viene effettuato anche il caricamento dei coefficienti nei registri.
- PROCESSING: All'arrivo del segnale di START i dati vengono campionati e la FSM si porta nello stato di processing dove viene eseguita la divisione. Si resta in questo stadio per un solo ciclo di clock durante il quale la divisione è svolta e il risultato è salvato in un registro Y_temp.
- DATA_OUT: In questo stato il calcolo è stato eseguito e il sistema manda il risultato sui pin di uscita informando anche che il dato in uscita è corretto ponendo il segnale di VALID = 1. Il sistema una volta fatto ciò torna nello stato di Idle a meno che non arrivino altri dati in ingresso, in quel caso torna direttamente nella fase di processing così da avere nel clock successivo nuovamente il risultato in uscita.

Per quanto riguarda la precisione, nel nostro caso, è esattamente la stessa della fase due perché è uguale l'algoritmo utilizzato per effettuare la divisione.

Con la sintesi su Standard Cells si è avuto un leggero peggioramento delle prestazioni rispetto alla soluzione precedente. La latenza in questo caso è passata da 5 ns a 7 ns e si nota che c'è stato anche un peggioramento del Througput perché

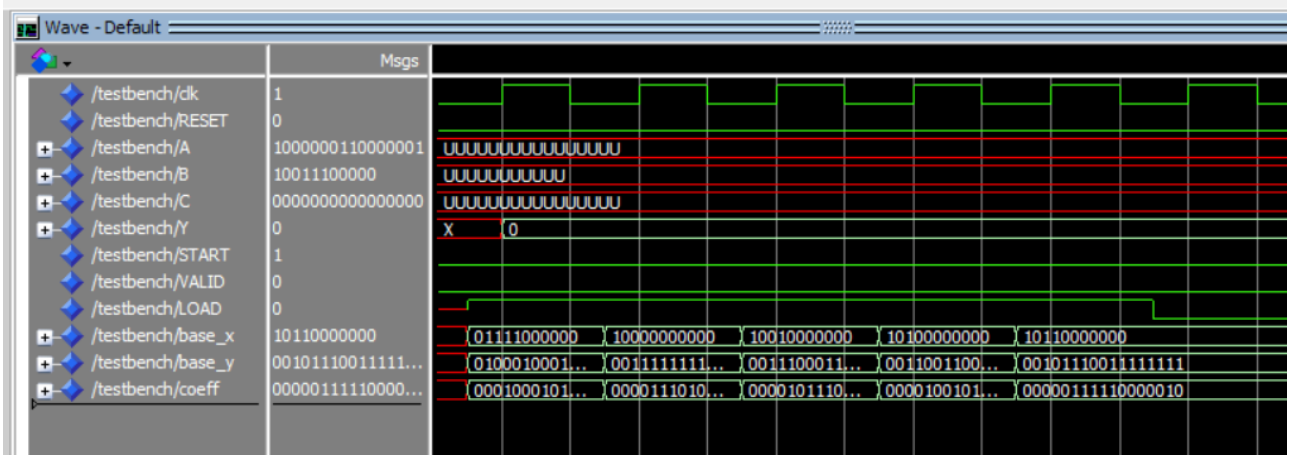

Figura 13 - Fase di Load delle triplette nel datapath 3

con questa soluzione gli ingressi possono essere applicati ogni 2 fronti di clock e non più ad ogni fronte di salita.

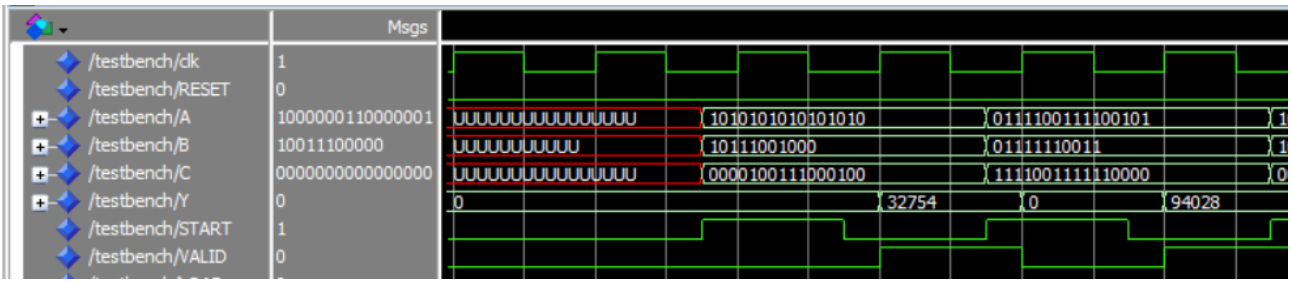

Figura 14 - Fase di esecuzione del datapath 3

Period (ns)	Freq. MHz	Latency in numero di cicli di clock	Throughput (MS/s)	slack (ns)	Area (um^2)	Area (Kgate,eq)	Leakage (uW)	Dynamic Power (uW)	Total Power (uW)	DynP/freq (uW/MHz)	En/op (uW/MHz)
7	142,8571429	1	71,42857143	0	4876,3119	6,18037	72,8084	243,85	316,6584	1,70695	1,70695

Figura 15 - Tabella risultati analisi su Standard Cells

Come possiamo notare, rispetto alla soluzione precedente cresce l'area dato che bisogna aggiungere l'area dei registri che servono per memorizzare i coefficienti, quella del contatore e quella della FSM. Conseguentemente all'aggiunta di questi crescono anche la potenza statica e quella dinamica.

L'analisi su FPGA in questo caso non è stata effettuata in quanto avrebbe richiesto l'utilizzo di un dispositivo differente, dato che quello usato per le altre soluzioni ovvero "EP4CE6E22C9L" possiede solamente 92 pin di I/O mentre in questo caso per gestire tutti gli ingressi e le uscite ne servirebbero almeno 110.

Conclusioni.

Con il metodo proposto nella fase 2 si riesce ad ottenere un circuito in grado di svolgere l'operazione richiesta con una precisione comunque molto alta, in percentuale, ma che rispetto alla prima soluzione riesce ad essere molto più efficiente in termini sia di velocità, che di consumo che di area.

Tale soluzione è quindi da preferire nelle applicazioni dove è più importante ottenere un risultato in maniera veloce, o consumare poco, rispetto ad avere un'elevatissima precisione, che comunque rimane alta anche nella soluzione 2 (0,001% per il datapath 1, 0,045 per il datapath 2).