RELAZIONE PROGETTO RTL

Andrea Castronovo
n. matricola 0001029122
andrea.castronovo3@studio.unibo.it

FASE 1 – DIVISORE COMBINATORIO

In questa prima fase di progetto si ha l'obbiettivo di realizzare un semplice datapath in grado di calcolare il risultato *Y = A/B + C* avendo come ingressi (A, B, C) degli operandi con notazione a virgola fissa senza segno, quindi numeri decimali positivi.

- **Divisione.**
  Nelle ipotesi di progetto viene richiesto di utilizzare un semplice divisore combinatorio così da avere una bassa latenza; infatti, per il mio elaborato ho deciso di campionare solamente gli ingressi e le uscite non inserendo un registro intermedio che spezzasse le due operazioni così da ottenere un datapath con latenza pari ad 1 T_{CK}.
  Per poter utilizzare correttamente il quoziente è utile comprendere il funzionamento di una divisione binaria con ingressi numeri a virgola fissa. Pensando di poter rappresentare un semplice numero decimale come il suo intero moltiplicato per un fattore di scala (2,34 = 234 x 1/100) allora la divisione fra due numeri decimali è il quoziente intero ottenuto dalla divisione degli interi equivalenti, moltiplicato per un fattore di scala ottenuto come quoziente dei fattori di scala fra dividendo e divisore.

  $34{,}56/1{,}234 = 28{,}00648.. \longleftrightarrow 3456/1234 = 2 \times \frac{1}{100} / \frac{1}{1000} = 20$
  $1{,}23/6{,}25 = 0{,}1968 \longleftrightarrow 123/625 = 0 \times \frac{1}{100} / \frac{1}{100} = 0$

  Possiamo allora notare che è possibile aumentare la precisione della divisione aumentando il fattore di scala del dividendo.

  $1{,}23/6{,}25 = 0{,}1968 \longleftrightarrow 123\,000/625 = 196 \times \frac{1}{100000} / \frac{1}{100} = 0{,}196$

  Lo stesso ragionamento lo possiamo riportare al divisore combinatorio, in quanto esso non può distinguere un numero binario in rappresentazione intera con uno a rappresentazione in virgola fissa; pertanto, il suo risultato sarà dato come quoziente intero degli interi equivalenti con un numero di bit decimali che è dato dalla sottrazione fra il numero di bit rappresentanti la parte decimale di dividendo e divisore. Nel caso d'interesse, dati gli ingressi:

  - A (8+8 bit) ➡ W(16,8,8)
  - B (3+8 bit) ➡ W(11,3,8)

  ⟹ A/B (16+0) ⟶ W(16, 16, 8 – 8 = 0)

  Quindi il quoziente <u>non</u> avrebbe un numero di bit rappresentanti la parte decimale, per tanto la precisione risulterebbe estremamente bassa.
  La soluzione quindi, come nel caso dei numeri decimali, è quella di aggiungere un numero *'n'* di bit (0) a destra della parte decimale del dividendo, così da ottenere un quoziente con *'n'* bit per la parte decimale; ma aggiungere un elevato numero di bit aumenterebbe sì la precisione a discapito però di un aumento d'area, capacità e quindi consumo e tempi di propagazione.

Non avendo uno specifico criterio da rispettare per questo progetto, riguardo precisione o area e consumo, ho deciso di fare diverse simulazioni di progetto al variare di *'n'* e quindi aumentando la precisione.

Precisione [n.Bit]	AREA Tot.L.E.	%	TEMPO [ns] Periodo Min.	%	POTENZA [mW] Statica	Dinamica	%	Tot.
0	326	-	71	-	41,19	0,49	-	73,52
2	376	15,34%	85	19,72%	41,19	0,52	6,12%	74,02
5	451	38,34%	100	40,85%	41,20	0,54	10,20%	74,16
8	529	62,27%	122	71,83%	41,20	0,57	16,33%	74,29
10	576	76,69%	133	87,32%	41,35	0,63	28,57%	74,91
16	732	124,54%	162	128,17%	41,40	0,83	69,39%	76,34

Un'elevata precisione (16 bit) notiamo che comporta un elevato incremento di area e consumo, al contrario un loro basso incremento comporta però una bassa precisione poiché con 5 bit si avrebbe un errore sulla seconda cifra decimale. Pertanto, ho deciso di orientarmi su una scelta intermedia e siccome fra 10 e 8 bit di precisione la differenza di incremento delle prestazioni è minima ho scelto di aggiungere 10 bit al dividendo, così che si abbia un errore sulla quarta cifra decimale ($\frac{1}{2^{10}} = 0{,}000976 \ldots$) e inoltre così facendo il quoziente è già allineato per avere una corretta somma con C.
In tutto questo il resto della divisione non entra in gioco nel calcolo della divisione fra numeri a virgola fissa, quindi ho deciso di omettere il suo calcolo non essendo necessario.

- **Somma.**
  Avendo aggiunto 10 bit all'ingresso A, si otterrà un nuovo dividendo che sarà quindi rappresentato complessivamente da 26 (16 + 10 di precisione) bit, come prima 8 rimangono per la parte intera mentre diventano 18 (8 + 10) per la parte decimale; invece Il divisore B rimane invariato.

  • A_int (8 + 18) ➡ W(26,8,18)
  • B (3 + 8) ➡ W(11,3,8)
  ⟹ OUT_DIV (16 + 10) ⟶ W(26, 16, 18 – 8 = 10)

  A questo punto otterremo un quoziente che sarà formato dal maggior numero di bit fra dividendo e divisore (26 di A) con 10 (18 - 8) bit per la parte decimale e quindi 16 bit rimanenti per la parte intera, essendo operandi Fixed Point senza segno.
  A questo punto non rimane altro che fare la semplice somma essendo due operandi già allineati poiché hanno lo stesso numero di bit decimali; in realtà ci vengono forniti i range degli ingressi: A range 100,000 -> 205,000 e B range 3,750 -> 6,000 .
  Pertanto, il maggior numero che si possa verificare per il quoziente è $\frac{205}{3{,}75} = 54{,}6666$ e quindi si può effettuare un troncamento sulla parte intera del quoziente, andando quindi a lasciare 7 bit per la parte intera (54 → 110110); 7 e non 6 bit poiché bisogna tenere a mente che a 54,6666 può essere sommato al valore massimo rappresentabile da C (63 → 111111). Quindi Y → W(17,7,10).

FASE 2 – DIVISORE APPROSSIMATO

In questa seconda fase di progetto viene richiesto di realizzare sempre la stessa funzione ma evitando di utilizzare un divisore combinatorio, aspettandosi quindi una precisione ridotta ma bensì vantaggi in area e consumo, con conseguenti riduzioni nei tempi di propagazione e capacità.
Per realizzare la divisione si richiede di fare una moltiplicazione fra il dividendo A e l'inverso del divisore B, ottenuto come approssimazione di una spezzata: $\frac{1}{B}$ = *base_y + coeff * displacment.*

Innanzitutto, è bene capire quale metodo utilizzare per approssimare la spezzata alla curva determinando i vari coefficienti angolari per i diversi segmenti e poi l'intercetta.

- **Coefficiente Angolare.**
  Per quanto riguarda il coefficiente angolare ho deciso di utilizzare la derivata della curva calcolata nel punto medio del segmento, così da avere una spezzata con pendenza dettata dalla tangente alla curva nel punto medio; quindi, mi aspetto che sia sempre più precisa all'aumentare di B ovvero a mano a mano che la curva tende ad appiattirsi.

$$Coeff. = \frac{d}{dx}\frac{1}{x}\Big|_{P.medio} = -\frac{1}{x^2}\Big|_{P.medio}$$

- **Intercetta.**
  Per quanto riguarda l'intercetta ho deciso di determinarla dall'espressione della spezzata facendo coincidere i punti medi della spezzata e della curva.

$$\text{Base\_y} = \frac{1}{P.medio} - [Coeff. \times (P.medio - Soglia)]$$

In questo modo si avrà quindi una spezzata che è esattamente la tangente alla curva nel punto medio del segmento determinato da soglia a soglia e propagata da soglia a soglia.
A questo punto risulta necessario determinare il numero di soglie

A	B	A/B
128	3,75	34,13333333
128	3,75	34,13333333
128	3,825	33,46405229
128	3,9	32,82051282
128	3,975	32,20125786
128	4	32
128	4,05	31,60493827
128	4,125	31,03030303
128	4,2	30,47619048
128	4,275	29,94152047
128	4,35	29,42528736
128	4,425	28,92655367
128	4,5	28,44444444
128	4,5	28,44444444
128	4,575	27,97814208
128	4,65	27,52688172
128	4,725	27,08994709
128	4,8	26,66666667
128	4,875	26,25641026
128	4,95	25,85858586
128	5	25,6
128	5,025	25,47263682
128	5,1	25,09803922
128	5,175	24,73429952
128	5,25	24,38095238
128	5,325	24,03755869
128	5,4	23,7037037
128	5,475	23,37899543
128	5,5	23,27272727
128	5,55	23,06306306
128	5,625	22,75555556
128	5,7	22,45614035
128	5,775	22,16450216
128	5,85	21,88034188
128	5,925	21,60337553
128	6	21,33333333
128	6	21,33333333

Coeff.	-0,066597294
base_y	0,266389178
P.medio	3,875

Coeff.	-0,055363322
base_y	0,249134948
P.medio	4,25

Coeff.	-0,04432133
base_y	0,221606648
P.medio	4,75

Coeff.	-0,036281179
base_y	0,199546485
P.medio	5,25

Coeff.	-0,030245747
base_y	0,18147448
P.medio	5,75

A/B 5 segmenti	Errore Relativo
34,09781478	-0,10%
34,09781478	-0,10%
33,45848075	-0,02%
32,81914672	0,00%
32,1798127	-0,07%
31,88927336	-0,35%
31,5349481	-0,22%
31,00346021	-0,09%
30,47197232	-0,01%
29,94048443	0,00%
29,40899654	-0,06%
28,87750865	-0,17%
28,36565097	-0,28%
28,36565097	-0,28%
27,9401662	-0,14%
27,51468144	-0,04%
27,08919668	0,00%
26,66371191	-0,01%
26,23822715	-0,07%
25,81274238	-0,18%
25,54195011	-0,23%
25,42585034	-0,18%
25,07755102	-0,08%
24,7292517	-0,02%
24,38095238	0,00%
24,03265306	-0,02%
23,68435374	-0,08%
23,33605442	-0,18%
23,22873346	-0,19%
23,03516068	-0,12%
22,74480151	-0,05%
22,45444234	-0,01%
22,16408318	0,00%
21,87372401	-0,03%
21,58336484	-0,09%
21,29300567	-0,19%
21,29300567	-0,19%

coeff.	-0,06449987402
base_y	0,26606198035
P. medio	3,9375

coeff.	-0,05377021634
base_y	0,24196597353
P.medio	4,3125

coeff.	-0,04551111111
base_y	0,22186666667
P.medio	4,6875

coeff.	-0,03901844231
base_y	0,20484682213
P.medio	5,0625

coeff.	-0,03382216938
base_y	0,19024970273
P.medio	5,4375

coeff.	-0,029598798
base_y	0,17759278529
P.medio	5,8125

A/B 6 segmenti	Errore Relativo
34,05593348	-0,23%
34,05593348	-0,23%
33,43673469	-0,08%
32,8175359	-0,01%
32,19833711	-0,01%
31,99193752	-0,03%
31,57913832	-0,08%
30,97164461	-0,19%
30,45545054	-0,07%
29,93925646	-0,01%
29,42306238	-0,01%
28,9068683	-0,07%
28,39067423	-0,19%
28,39893333	-0,16%
27,96202667	-0,06%
27,52512	-0,01%
27,08821333	-0,01%
26,65130667	-0,06%
26,22039323	-0,14%
25,84581619	-0,05%
25,59609816	-0,02%
25,47123914	-0,01%
25,09666209	-0,01%
24,72208505	-0,05%
24,35196195	-0,12%
24,02726912	-0,04%
23,7025763	0,00%
23,37788347	0,00%
23,26965253	-0,01%
23,05319065	-0,04%
22,73187652	-0,10%
22,44772806	-0,04%
22,1635796	0,00%
21,87943115	0,00%
21,59528269	-0,04%
21,31113424	-0,10%
21,31113424	-0,10%

Con l'aumentare del numero di soglie sicuramente aumenta la precisione, ma con 7 o più si avrà un numero binario ad indicare la soglia avente un elevato numero di decimali perciò difficile da rappresentare. Una diversa implementazione del progetto, quindi, potrebbe essere quella di simulare il circuito con un numero di soglie pari a 6 ma per quanto riguarda le ipotesi di progetto

le soglie richieste erano 5 e pertanto ci si aspetta una precisione che per le scelte adottate tenderà ad essere più bassa nel centro della spezzata e più alta nei punti di soglia, con un minimo di 0.00% e un massimo di -0.35% sulla soglia 4.
A questo punto bisogna scegliere quale rappresentazione dare alle triplette di costanti

- **Base_x → W(11,3,8)**
  Siccome le soglie devono essere confrontate con il divisore (B →W(11,3,8)) ho deciso di adottare la stessa rappresentazione; in questo modo è possibile fare un confronto in *standard logic*, ovvero bit a bit, così da poter determinare il segmento nel quale si trova l'inverso di B e quindi scegliere le diverse costanti utili nell'algoritmo. Scelta utile anche per la sottrazione nel calcolo del *displacment*, poiché già allineati.

```
("01111000000", --W(11,3,8)     --3.75
 "10000000000",                 --4.00
 "10010000000",                 --4.50
 "10100000000",                 --5.00
 "10110000000");                --5.50
```

- **Coeff & Base_y → W(16,0,16)**
  Al contrario, per le altre due cinquine di costanti ho scelto di rappresentarli con 0 bit per la parte intera essendo numeri minori di 1, e dandogli 16 bit per rappresentare la parte decimale così da avere una maggior precisione. Bisogna comunque tenere a mente che l'esatto numero decimale richiederebbe un numero di bit veramente elevato per essere rappresentato e perciò si fa un'ulteriore approssimazione; questo però va ad incidere oltre la seconda cifra percentuale di precisione.

```
coeff := ("0001000100001100", --W(16,0,16) --0.06658935546875    != 64/961
          "0000111000101100",              --0.05535888671875    != 16/289
          "0000101101011000",              --0.0443115234375     != 16/361
          "0000100101001001",              --0.0362701416015625  != 16/441
          "0000011110111110");             --0.030242919921875   != 16/529

base_y := ("0100010000110010", --W(16,0,16) --0.266387939453125   != 256/961
          "0011111111000111",              --0.2491302490234375  != 72/289
          "0011100010111011",              --0.2216033935546875  != 80/361
          "0011001100010101",              --0.1995391845703125  != 88/441
          "0010111001110101");             --0.1814727783203125  != 96/529
```

Nel susseguirsi dell'algoritmo e quindi delle varie operazioni è possibile eseguire diversi troncamenti per rendere più efficiente il codice, ad esempio: "*displacment*" (*B - Soglia*) avrà una rappresentazione W(11,3,8) ma siccome per le soglie adottate e i range di B, tale valore sarà compreso fra 0 e 0,5 allora si possono eliminare i 3 bit della parte intera; "*mx*" (*displacment*coeff*) sarà costituito da 24 bit (8 + 16) con rappresentazione W(24,0,24) ma per allinearlo a "*base_y*", col quale deve essere sommato, verranno presi soltanto i 16 bit più significativi della parte decimale poiché la parte intera è 0, facendo così un'altra approssimazione; "*out_div*" (*A * 1/B*) sarà rappresentato da 32 bit (16 + 16) con un W(32,8,24) ma per i motivi descritti nella fase_1 inerenti ai range di A e B è possibile rappresentare l'uscita della divisione con solo 7 bit per la parte intera, mentre per la sua parte decimale si deve necessariamente fare un troncamento a 10 bit per avere il dato allineato con C ed eseguire correttamente la somma. Per queste scelte adottate si avrà un Y che avrà la stessa rappresentazione della fase_1, Y → W(17,7,10).
Diversamente da quanto scelto nella fase_1 ho inserito i registri di pipeline per spezzare le operazioni, in questo modo la latenza è paria 2 T_{CK} ma avendo inserito dei registri per tutti gli ingressi compresi A e C, il Throughput rimane pari a $\frac{1}{\mathrm{TCK}}$.

Confrontando le simulazioni delle due fasi

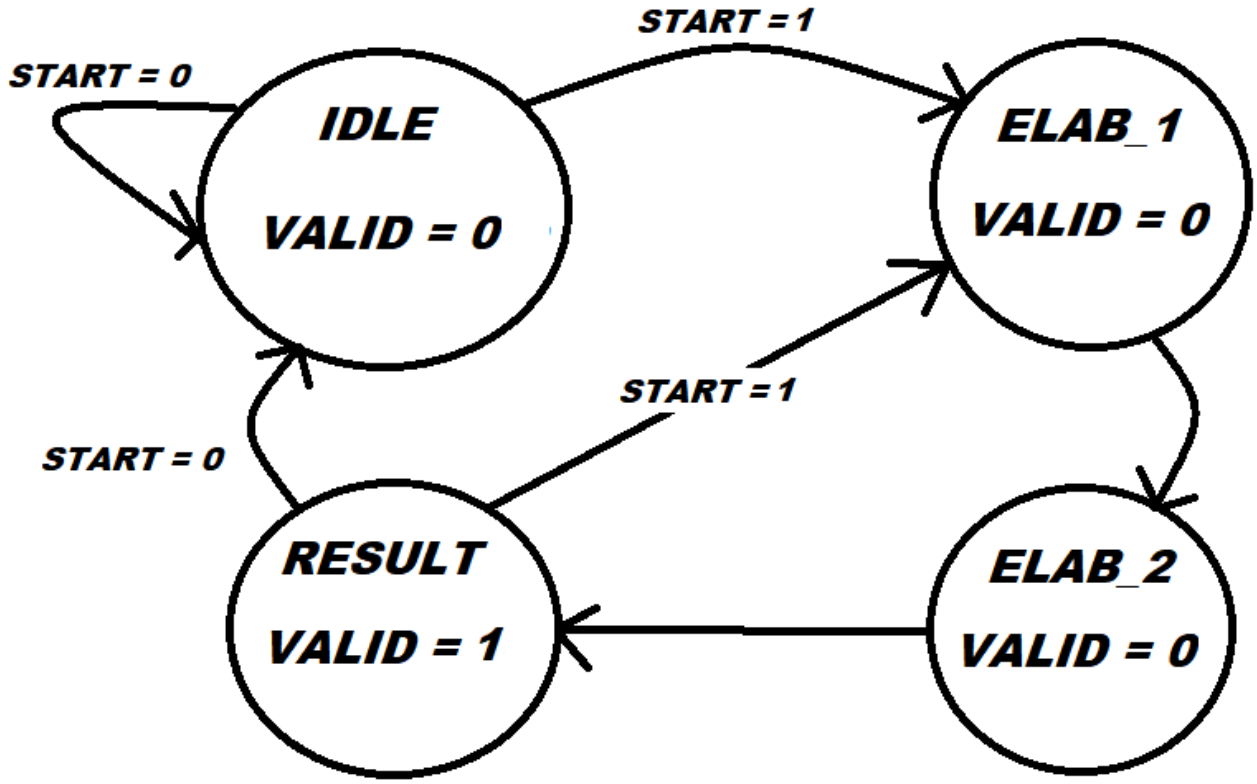
		F.P.G.A.				S.C.			
		COMBINATORIO	APRROX. PIPE	APPROX. SEQ		COMBINATORIO	APRROX. PIPE	APPROX. SEQ	
TEMPI	Periodo min. [ns]	133	16	26		13	4	6	
	Freq. Max [MHz]	7,52	62,5	38,46		76,92	250	166,67	
FIGURE DI MERITO	Latenza [Tck]	1	2	1		1	2	1	
	Throughput [MS/s]	7,52	62,5	38,46		76,92	250	166,67	
AREA	Tot. L.E.	576	139	99					
	[Kgate,eq]					6,475	3,776	3,752	
POWER	Dyn.	0,63	1,89	0,92	[mW]	214,46	334,07	202,62	[uW]
	Static	41,35	41,33	41,32	[mW]	84,13	45,15	46,84	[uW]
	En/Op	0,0838	0,0302	0,0239	[mW/MHz]	2,7879	1,3363	1,2157	[uW/MHz]
PRECISIONE		0,10%	-0,35%			0,10%	-0,35%		

Si può notare quanto ci si aspettava in partenza, ovvero sia per la simulazione su F.P.G.A che quella su Standard Cell, per quanto riguarda l'algoritmo con divisione approssimata ad una retta si ha sicuramente una precisione ridotta ma con una netta efficienza nelle prestazioni in termini di area e consumo (Energia/Operazione). Nel caso di divisore approssimato sequenziale, quindi in assenza dei registri di pipeline, si ha un'ulteriore vantaggio in prestazioni ma a discapito di una riduzione della frequenza massima operativa.

Fase 3 – UNITA' COMPLETA.

In quest'ultima fase si ha l'obbiettivo di realizzare l'unità completa composta dal datapath_2 rivisto e adattato per far si che le triplette di costanti siano caricate dall'esterno per poi far gestire il tutto da una macchina a stati. In questo modo le costanti vengono quindi salvate su dei registri ed essendo triplette di 5 elementi richiederanno appunto 5 cicli di clock per essere memorizzate; successivamente all'interno del datapath viene eseguito il confronto per ottenere il corretto segmento nel quale si trova il divisore, quindi le tre costanti verranno passate al vecchio datapath della fase 2 che eseguendo le diverse operazioni precedentemente spiegate calcola il risultato della funzione Y. Non avendo variato i dati, ma solo il modo in cui verranno gestiti, le diverse rappresentazioni rimarranno quelle presentate e spiegate nella fase precedente.
Per la gestione della macchina a stati (START e VALID) viene eseguita una macchina di MOORE formata da 4 stati:

- IDLE: Per l'inizializzazione della macchina, soltanto quando l'ingresso START va alto allora si passa ad ELAB1.
- ELAB1: Avendo mantenuto la scelta di pipeline, in questo stato verranno gestite le prime operazioni quindi al successivo fronte di clock la macchina passerà ad ELAB2.
- ELAB2: Come previsto, in questo secondo stato si esegue la seconda porzione di operazioni per ottenere in uscita il risultato Y_int (A/B + C) che verrà campionato al successivo fronte.
- RESULT: Y_int è stato campionato perciò VALID è alto per un solo ciclo a segnalare la presenza del corretto risultato d'elaborazione sull'uscita Y; a questo punto se START torna alto si inizia una nuova elaborazione passando in ELAB1, altrimenti IDLE.



Al contrario della Fase due, con registri di pipeline s'intende il solo campionamento di *mx* (*coeff*displacment*) poiché si ha l'ipotesi che i dati in ingresso non debbano cambiare durante l'elaborazione ovvero tra i valori alti di START e VALID. Al contrario un'altra soluzione non adottata poteva essere quella di avere dei segnali di START l'uno successivo all'altro, poiché mantenendo tutti i registri di pipeline della fase 2 sarebbe stato possibile campionare i nuovi dati ed elaborarli in periodi subito seguenti; questa scelta però richiedeva una gestione della macchina a stati più complessa.

Concludendo, per le scelte fatte la latenza rimane a 2 T_{CK}, dal momento in cui si campionano gli ingressi con START alto fino al momento in cui si campiona sull'uscita Y. Il throughput è pari a $\frac{1}{3\,Tck}$ inteso come frequenza con la quale si possano dare gli ingressi, questo perché dal momento in cui si campiona il primo ingresso, fino al momento del nuovo campionamento degli ingressi in RESULT passano 3 periodi di clock: 2 per l'elaborazione e 1 per la validazione del risultato sull'uscita.