

SISTEMI DIGITALI INTEGRATI
OPERAZIONE SAN SILVESTRO

BUTTERFLY

Antonio Tudisco

Lorenzo Lagostina

Maria Elena D'Agostino

23 giugno 2022



Indice

1	Introduzione	3
2	Cenni Teorici	3
3	Data Flow Diagram	5
4	Soluzioni alternative	9
4.1	PROVA_1	9
4.2	PROVA_2	12
4.3	Confronto	15
5	Datapath	16
5.1	Register file	18
5.2	Bus	20
5.3	Moltiplicatore/shift	21
5.4	Sommatore/sottrattore	22
5.5	Blocco Approssimazione	24
6	Protocollo di ingresso/uscita	26
7	Parallelismo	27
8	Timing	28
9	Cenni teorici sulla control unit	33
9.1	Sequenziatore e generatore di comandi	33
9.2	Late status PLA	35
10	Control unit della butterfly	36
10.1	Schema generale e stati	36
10.2	Organizzazione della uROM	39
10.3	Late status PLA	42
10.4	uInstruction Register e uAddress Register	44
11	Timing della CU	45
12	Realizzazione butterfly 16x16	46
13	Simulazioni	47
13.1	Spiegazione generale	47
13.2	Singola butterfly	48
13.3	Butterfly 16x16	51

14 Appendice	55
14.1 Listati VHDL e Python	55
14.2 Adder.vhd	55
14.3 approx.vhd	56
14.4 Butterfly.vhd	58
14.5 Butterfly16x16.vhd	61
14.6 butterflyDatapath.vhd	63
14.7 cu.vhd	70
14.8 FF.vhd	73
14.9 gen_mux_2_1.vhd	74
14.10late_status_pla.vhd	75
14.11moltiplicatore.vhd	76
14.12mux2to1.vhd	78
14.13mux_2_1.vhd	78
14.14n_bit_register.vhd	79
14.15packageBF.vhd	80
14.16registerFF.vhd	80
14.17RegisterFile.vhd	81
14.18sommatore.vhd	85
14.19tbAdder.vhd	87
14.20tbButterfly16x16.vhd	88
14.21tb_approx.vhd	91
14.22tb_Butterfly.vhd	91
14.23TB CU BUTTERFLY.vhd	96
14.24tb_decoder.vhd	97
14.25tb_moltiplicatore.vhd	98
14.26tb_RegisterFile.vhd	99
14.27tb_sommatore.vhd	103
14.28uAR.vhd	105
14.29uIR.vhd	106
14.30uROM.vhd	107
14.31butterfly16x16Simulation.py	109
14.32butterflySimulation.py	111
14.33computeFFT.py	113
14.34computeSingle.py	115
14.35dataConversion.py	118
14.36inputGen.py	119
14.37listatiLatex.py	120
14.38tableGen.py	121

1 Introduzione

L'obiettivo dell'elaborato è progettare un Processing Element in grado di eseguire la FFT basata sul processore Butterfly. Soddisfacendo le specifiche date, è stata realizzata una possibile implementazione tra le infinite appartenenti nello spazio delle soluzioni.

2 Cenni Teorici

La trasformata di Fourier Veloce o FFT è un algoritmo utilizzato in moltissime applicazioni, come quelle basate sull'elaborazione di segnali, per calcolare la DFT (Discrete Fourier Transform). I vantaggi dell'utilizzo di tale algoritmo essendo eseguito in modo ricorsivo, sono molteplici. Questo infatti richiede una bassa complessità computazionale che porta a una conseguente riduzione dei tempi di calcolo e dei consumi energetici.

Dato il numero di campioni N del segnale a tempo discreto, la trasformata discreta di Fourier è così definita:

$$X_N(k) = \sum_{n=0}^{N-1} x(n)e^{-j(2/N)nk}$$

Le componenti in frequenza, non sono altro che una sommatoria estesa sui campioni di un prodotto complesso. In tale equazione:

- k -> è l'indice del campione in uscita
- n -> è l'indice della sommatoria
- N -> sono il numero dei campioni in ingresso
- $e^{-j(2/N)}$ -> si definisce W_N = TWIDDLE FACTOR, fasore che giace sul cerchio unitario di Eulero.

Dunque si ottiene tale equazione base per la Trasformata di Fourier veloce:

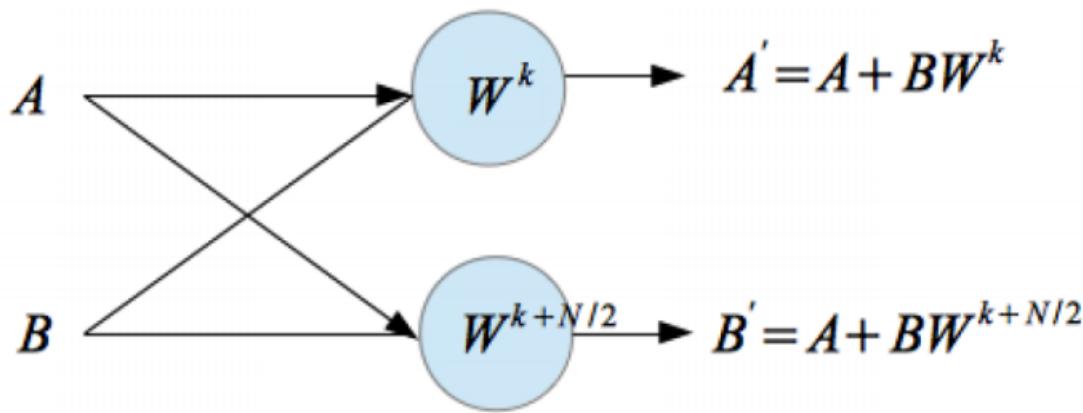
$$X_N(k) = \sum_{(n=0)}^{(N-1)} x(n)W_N^{nk}$$

La complessità di tale algoritmo è dell'ordine N^2 . Grazie all'algoritmo di Cooley-Tukey (in Radix = 2) il numero di operazioni ricorsive viene ridotto drasticamente fino a $N*\log(N)$.

$$X_N(k) = \sum_{(n=0)}^{((N/2)-1)} x(2n)e^{-j(2/N)2nk} + (e^{-j(2/N)k}) * \sum_{(n=0)}^{((N/2)-1)} x(2n+1) * e^{-j(2/N)2nk}$$

La trasformazione puramente matematica si basa sulla scomposizione ricorsiva della trasformata (su N campioni), in somme su insiemi più piccoli di valori di ingresso (su N/2 campioni), sfruttando le simmetrie e le periodicità della DFT. La trasformata di dimensioni minime è la Butterfly, che realizza l'algoritmo su due punti.

La generica Butterfly può essere implementata nel seguente modo

Figura 1: *Butterfly generica*

dove A e B sono gli input complessi, A' e B' gli output complessi, W_N^{nk} e $W_N^{(nk+N/2)}$ i due Twiddle Factor (dove il secondo si può derivare dal primo cambiando il segno). Le equazioni per il calcolo della Butterfly sono:

$$\begin{aligned} A' &= (A_R + jA_I) + (W_R + jW_I) * (B_R + jB_I) = \\ &= (A_R + B_R W_R - B_I W_I) + j(A_I + B_R W_I + B_I W_R) \end{aligned}$$

$$\begin{aligned} B' &= (A_R + jA_I) + (-W_R - jW_I) * (B_R + jB_I) = \\ &= (A_R - B_R W_R + B_I W_I) + j(A_I - B_R W_I - B_I W_R). \end{aligned}$$

$$B' = -A'_R + 2A_R + j(2A_I - A'_I).$$

La realizzazione della Butterfly parte da queste considerazioni utilizzando le tecniche della microprogrammazione (anche se un po' spurate data la semplicità dell'algoritmo).

3 Data Flow Diagram

Il primo step per realizzare la macchina richiesta è quello di esplorare lo spazio di tutte le possibili soluzioni. Questa fase preliminare consiste nella scelta di un opportuno Data Flow Diagram che rispetti le specifiche:

- utilizzare un blocco moltiplicatore, avente un livello di pipe, in grado di effettuare sia la moltiplicazione semplice che la moltiplicazione x2 (shifter aritmetico), grazie ad un opportuno segnale di controllo.
- utilizzare al massimo due blocchi per la somma/sottrazione, aventi un livello di pipe, che siano in grado di effettuare sia la somma che la differenza degli operandi, grazie ad un opportuno segnale di controllo.
- ricevere i dati in ingresso definiti in forma frazionaria ($-1 < \text{dato} < \text{complemento a due (C2)}$) su 20 bit.
- definire un protocollo di invio dei dati verso l'esterno coincidente con il protocollo dei dati che vengono ricevuti in ingresso.

La scelta del DFD ottimo è avvenuta in seguito all'analisi di diverse implementazioni, tutte generate con il fine di cercare di ottimizzare sia il throughput, la velocità della macchina, il numero di registri, dell'hardware e dei bus da utilizzare.

Si parla direttamente di Control Data Flow Diagram (CDFD) aggiungendo i registri di pipeline negli operatori, per avere dal primo momento chiaro il flusso dei dati ad ogni step algoritmo. Questo passaggio è stato necessario, poiché i blocchi hardware da utilizzare presentano un registro nel mezzo. Sia il blocco di moltiplicazione/shift che somma/sottrazione presentano un registro interno, per cui il risultato delle loro operazioni del ciclo n saranno disponibili al ciclo n+1. Il Control Data Flow Diagram utilizzato definitivamente per la realizzazione della Butterfly è di seguito riportato:

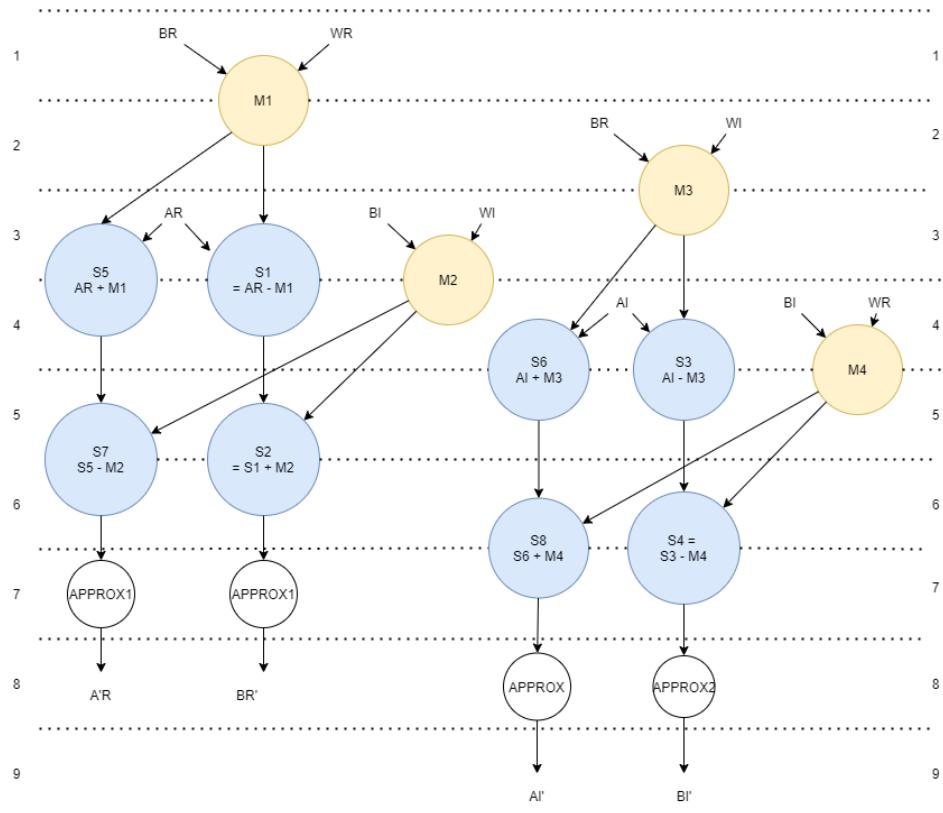


Figura 2: CDFD scelto con 2 blocchi sommatore/sottrattore e 1 blocco moltiplicatore/shift

Il fatto di dover utilizzare da specifiche al massimo un moltiplicatore, rende l'organizzazione della macchina più complicata, poiché questo componente agisce da collo di bottiglia. Utilizzando tale CDFD il numero di moltiplicazioni da effettuare sono minime, e questo ha portato alla conseguente riduzione della latenza e del throughput. Il ragionamento fatto per ricavare tale diagramma è scaturito dall'osservazione delle formule presentate nella sezione precedente e di seguito di nuovo riportate.

$$\begin{aligned}
 A' &= (A_R + jA_I) + (W_R + jW_I) * (B_R + jB_I) = \\
 &= (A_R + B_R W_R - B_I W_I) + j(A_I + B_R W_I + B_I W_R)
 \end{aligned}$$

$$\begin{aligned}
 B' &= (A_R + jA_I) + (-W_R - jW_I) * (B_R + jB_I) = \\
 &= (A_R - B_R W_R + B_I W_I) + j(A_I - B_R W_I - B_I W_R).
 \end{aligned}$$

Si nota che gli operandi per ricavare A'_r sono gli stessi di quelli necessari per ricavare i B'_r . Allo stesso modo gli operandi per ricavare A'_i e B'_i . La differenza sta nel tipo di operazioni da fare, se somma o sottrazione. Avendo a disposizione 2 sommatori/sottrattori ci permette di ricavare in parallelo i coefficienti delle parti reali, e di conseguenza anche quelli delle parti immaginarie.

Questa soluzione in rispetto delle Data Dependencies dell'algoritmo, ci permette di avere una latenza massima di 9 colpi di clock dal momento in cui viene salvato il primo dato in ingresso, ossia il B'_r . Il vantaggio migliore di questa soluzione è la riduzione notevole del throughput, infatti ogni 4 colpi di clock si può far partire una nuova esecuzione. LATENZA = 9 THROUGHTPUT = 5 Questo scheduling implementato è un tipo ALAP, soluzione utile nel caso in esame per poter andare sia più veloce, che per risparmiare il numero di registri temporanei.

Di seguito viene riportato il tempo di vita delle variabili.

	I	II	III	IV	V	VI	VII	VIII	IX
AR			X						
AI				X					
BR	X	X							
BI			X	X					
WR	X	0	0	X					
WI		X	X						
M1			X						
M3				X					
S5					X				
S1					X				
M2					X				
S6						X			
S3						X			
M4						X			
S7							X		
S2							X		
S8								X	
S4								X	
APPRO X_A'R								A'R	
APPRO X_B'R								B'R	
APPRO X_A'I									A'I
APPRO X_B'R									B'R

Figura 3: analisi del tempo di vita delle variabili del CDFD scelto

Si nota come sono necessari al massimo 4 registri temporanei. Il protocollo di ingresso/uscita è stato pensato in modo tale da inviare su quattro bus i dati nel Register File della struttura, nel modo seguente:

- Br, Bi
- Ar, Ai
- Wr
- Wi

4 Soluzioni alternative

4.1 PROVA_1

Prima di giungere alla soluzione definitiva appena riportata, sono state valutate ulteriori due implementazioni [PROVA_1 e PROVA_2].

Entrambe sono state pensate con il fine di ridurre al minimo la quantità di hardware utilizzato.

La prima [PROVA_1] è stata ricavata dalle seguenti equazioni

$$A' = (A_R + B_R W_R - B_I W_I) + j(A_I + B_R W_I + B_I W_R)$$

$$B' = -A'_R + 2A_R + j(2A_I - A'_I)$$

Dalle quali è stato ricavato il seguente Control Data Flow Diagram

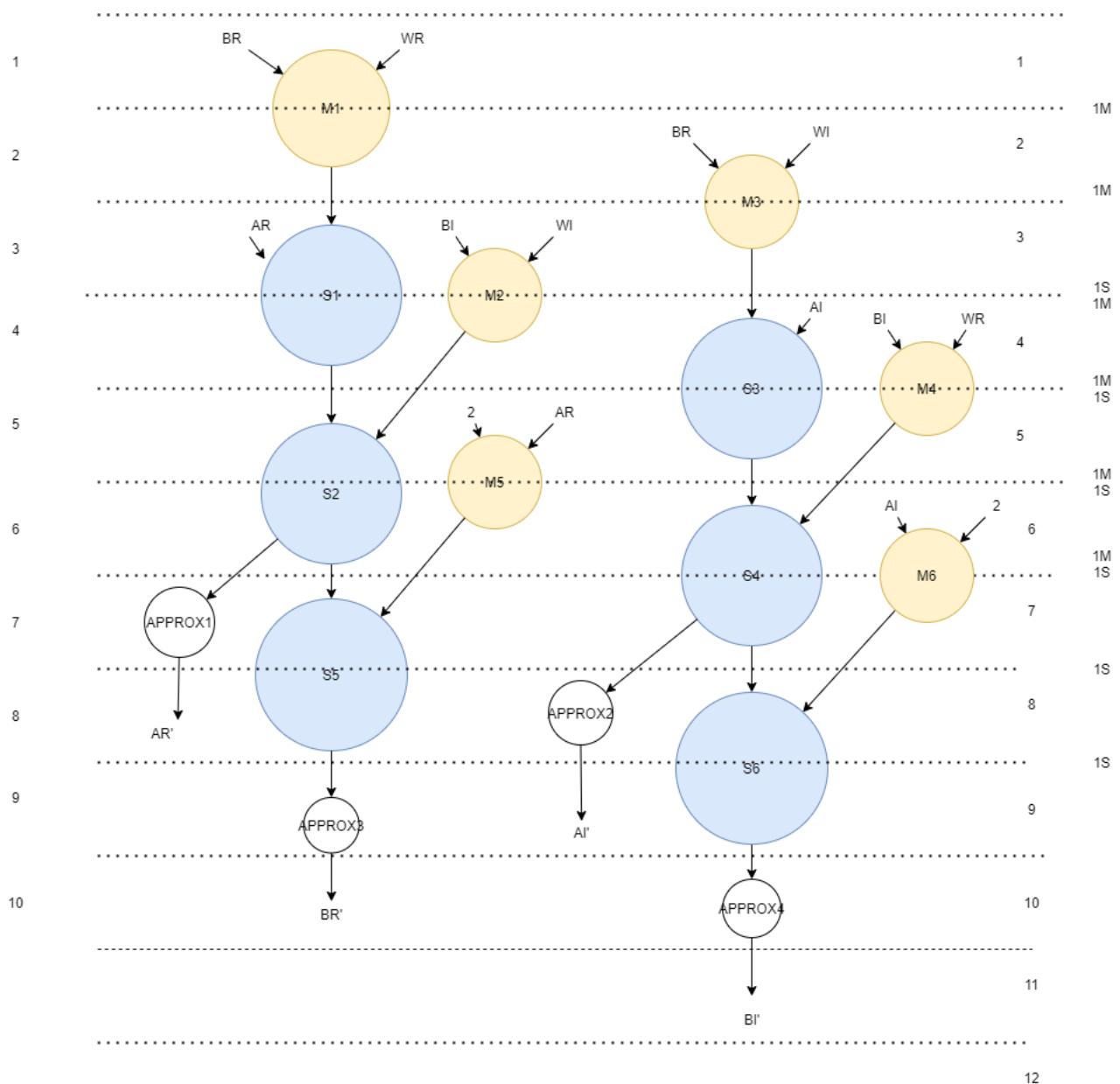


Figura 4: CDFD di PROVA_1

Da questo si ricava la seguente tabella relativa al tempo di vita delle variabili:

	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
AR			X	()	X							
AI				X	()	X						
BR	X	X										
BI			X	X								
WR	X	()	()	X								
WI		X	X									
M1			X									
M3				X								
S1					X							
M2					X							
S3						X						
M4						X						
S2							X					
M5							X					
S4								X				
M6								X				
APPX1								AR'	AR'	AR'	AR'	AR'
S5									X			
APPX2									AI'	AI'	AI'	AI'
APPX3										BR'	BR'	BR'
S6										X		
APPX4											BI'	BI'

Figura 5: analisi del tempo di vita delle variabili del CDFD di PROVA_1

4.2 PROVA_2

La seconda prova invece è stata ricavata dalle seguenti equazioni:

$$B' = (A_R - B_R W_R + B_I W_I) + j(A_I - B_R W_I - B_I W_R)$$

$$A' = -B'_R + 2A_R + j(2A_I - B'_I).$$

Dalle quali si ricava tale diagramma del controllo:

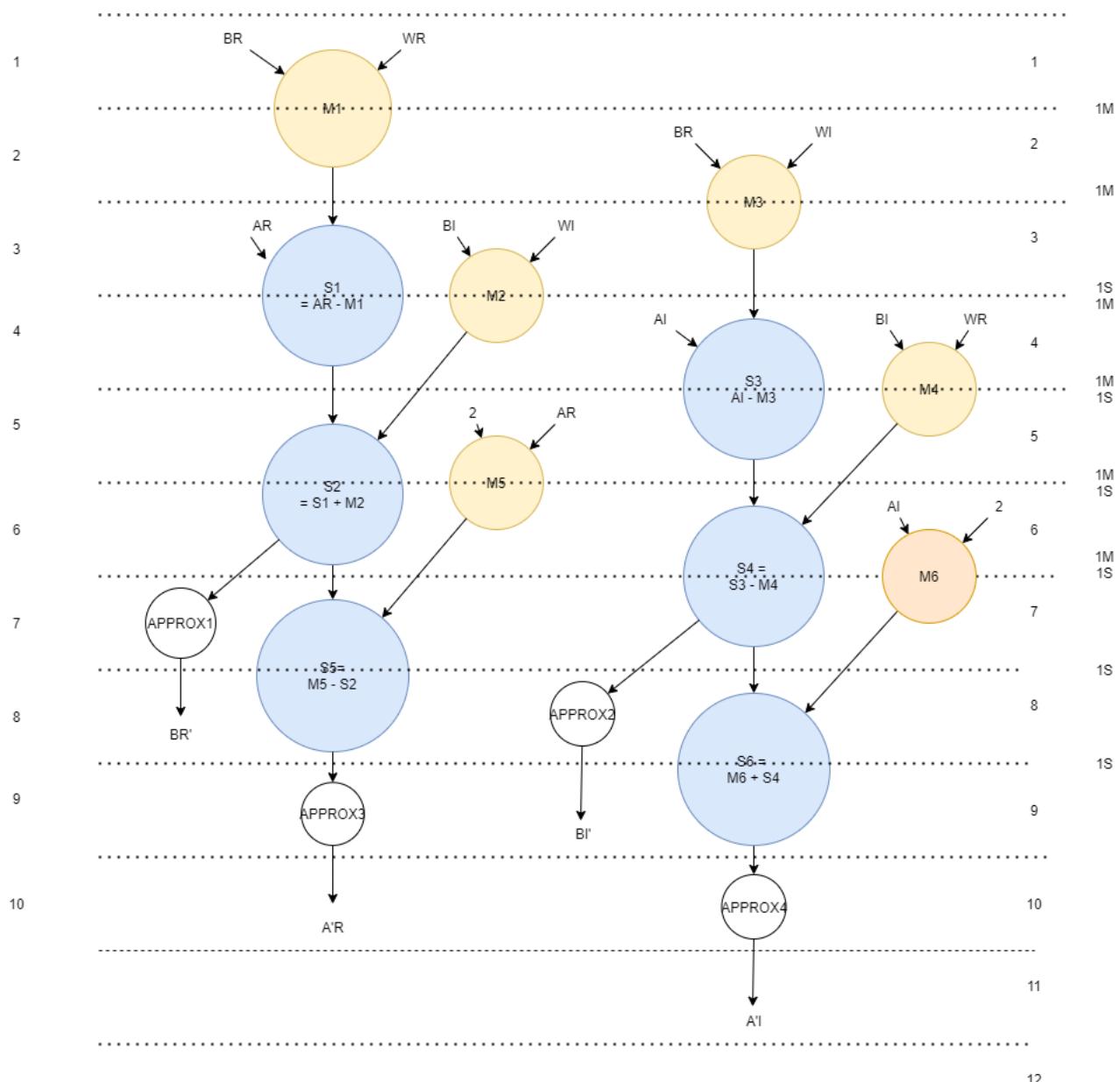


Figura 6: CDFD di PROVA_2

Il tempo di vita delle variabili di questa soluzione è il seguente:

	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
AR			X	()	X							
AI				X	()	X						
BR	X	X										
BI			X	X								
WR	X	0	0	X								
WI		X	X									
M1			X									
M3				X								
S1					X							
M2					X							
S3						X						
M4						X						
S2							X					
M5							X					
S4								X				
M6								X				
APPX1								BR'	BR'	BR'	BR'	BR'
S5									X			
APPX2									BL'	BL'	BL'	BL'
APPX3										AR'	AR'	AR'
S6										X		
APPX4											AI'	AI'

Figura 7: analisi del tempo di vita delle variabili del CDFD di PROVA_2

4.3 Confronto

Apparentemente le due soluzioni sembrano molto simili. Entrambe infatti utilizzano un solo blocco sommatore/sottrattore, un moltiplicatore/shift e un blocco per l'approssimazione. Sia la soluzione PROVA₁ che la PROVA₂ provano

La differenza che c'è tra queste due soluzioni prova_1 e prova_2 appena mostrate, sta nel fatto di come vengono gestiti i dati in ingresso e in uscita alla macchina. Le specifiche richiedono di rispettare lo stesso protocollo con cui i dati arrivano in ingresso.

Si può pensare ad un protocollo organizzato in modo tale da inviare separatamente i coefficienti dai W, nel seguente ordine:

- Br, Bi, Ar, Ai
- Wr, Wi

Mentre la soluzione prova_2 ha il vantaggio di avere i dati che vengono elaborati in uscita nello stesso ordine dei dati ricevuti in ingresso, la soluzione prova_1, richiede l'aggiunta di ulteriore hardware (registri e multiplexer), per far rispettare il protocollo stabilito.

Viene riportata una tabella riassuntiva delle tre soluzioni analizzate.

	LATENZA	THROUGHPUT	BUS INGRESSO	REGISTRI TEMPORANEI
Soluzione scelta	9	5	4	4
PROVA_1	11	7	2	4
PROVA_2	11	7	2	4

Figura 8: tabella di confronto delle tre soluzioni analizzate

La soluzione prova_1 è stata scartata soprattutto a causa dell'aggiunta di registri in uscita, che porta ad un rallentamento della macchina in termini di prestazioni. Le soluzioni a 'minimo hardware', che usano un singolo bus d'ingresso e di uscita ed un solo sommatore, hanno un inconveniente non trascurabile: nella butterfly 16 x 16, la gestione dell'avanzamento dei dati è estremamente complicata. Nonostante ci siano vantaggi in termini di costo dell'attrezzatura hardware, proprio per la suddetta ragione, la soluzione vincente è stata la scelta di implementare una macchina con un hardware che più ricco, ma sicuramente molto veloce e di facile gestione.

5 Datapath

Il datapath relativo alla soluzione realizzata è il seguente.

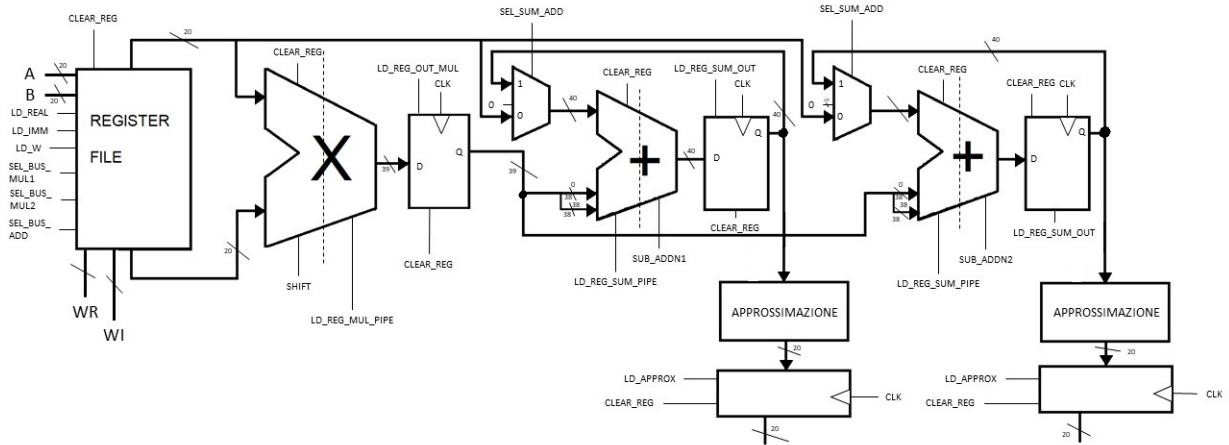


Figura 9: *Datapath della Butterfly*

La si nota come la struttura implementata, ottimizza sia il numero di bus che il numero di registri temporanei. Prima di raggiungere tale organizzazione, è stata fatta un'analisi del sistema ipotizzando che questo abbia risorse infinite. L'hardware a disposizione è composto da due blocchi sommatore/sottrattore e un blocco moltiplicatore/shift. Secondo tale analisi, i risultati di tutte le operazioni devono essere salvate nel Register File, il quale sarebbe dovuto essere composto da 12 registri per memorizzare tutti i risultati, (4 moltiplicazioni e 8 somme/sottrazioni) e 6 registri per memorizzare gli ingressi provenienti dall'esterno (A_r , A_i , B_r , B_i , Wr , Wi). La soluzione a risorse infinite prevede l'utilizzo di un numero di bus globali pari a 9, per inviare i dati necessari ai tre blocchi del datapath e ricevere i risultati in uscita. Sono inoltre previsti 6 bus di ingresso/uscita, per ricevere ed inviare i dati elaborati.

Ovviamente tale soluzione è impraticabile, dunque grazie all'analisi dell'algoritmo e del tempo di vita delle variabili, sono state implementate le seguenti ottimizzazioni:

- nel Register File sono memorizzati solo i dati provenienti dall'esterno (A_r , A_i , B_r , B_i , Wr , Wi).
- sono stati inseriti 4 registri temporanei per contenere le variabili che si ottengono dalle operazioni
- il numero di bus globali viene ridotto drasticamente, da 9 a 4.
- è necessaria l'introduzione di multiplexer dedicati, per discriminare i giusti ingressi da inviare ai due blocchi sommatore/sottrattore.

Come si può osservare dalla figura del datapath, è stata implementata un’ulteriore ottimizzazione circa il numero di registri temporanei utilizzati. Sono stati inseriti tre registri, per memorizzare le variabili interne dell’algoritmo, e 2 registri di uscita dopo il blocco approssimazione; dunque, un registro in più rispetto a quanto ricavato dallo studio del tempo di vita delle variabili. Il fine di tale scelta progettuale, avvenuta senza introdurre ulteriori step algoritmici, è dovuto al fatto che il throughput resta ottimizzato al massimo. Si è preferito utilizzare un registro in più a fronte di condividere quello in uscita dal moltiplicatore; tale scelta avrebbe portato a un rallentamento delle prestazioni e l’aggiunta di ulteriore hardware per garantire la condivisione del suddetto registro.

5.1 Register file

All'interno del Register File sono presenti 6 registri necessari per la memorizzazione dei coefficienti reali e immaginari A, B e W. In esso sono presenti anche dei multiplexer che servono a selezionare il dato da mandare sul relativo bus globale. Di seguito è riportata la struttura interna realizzata:

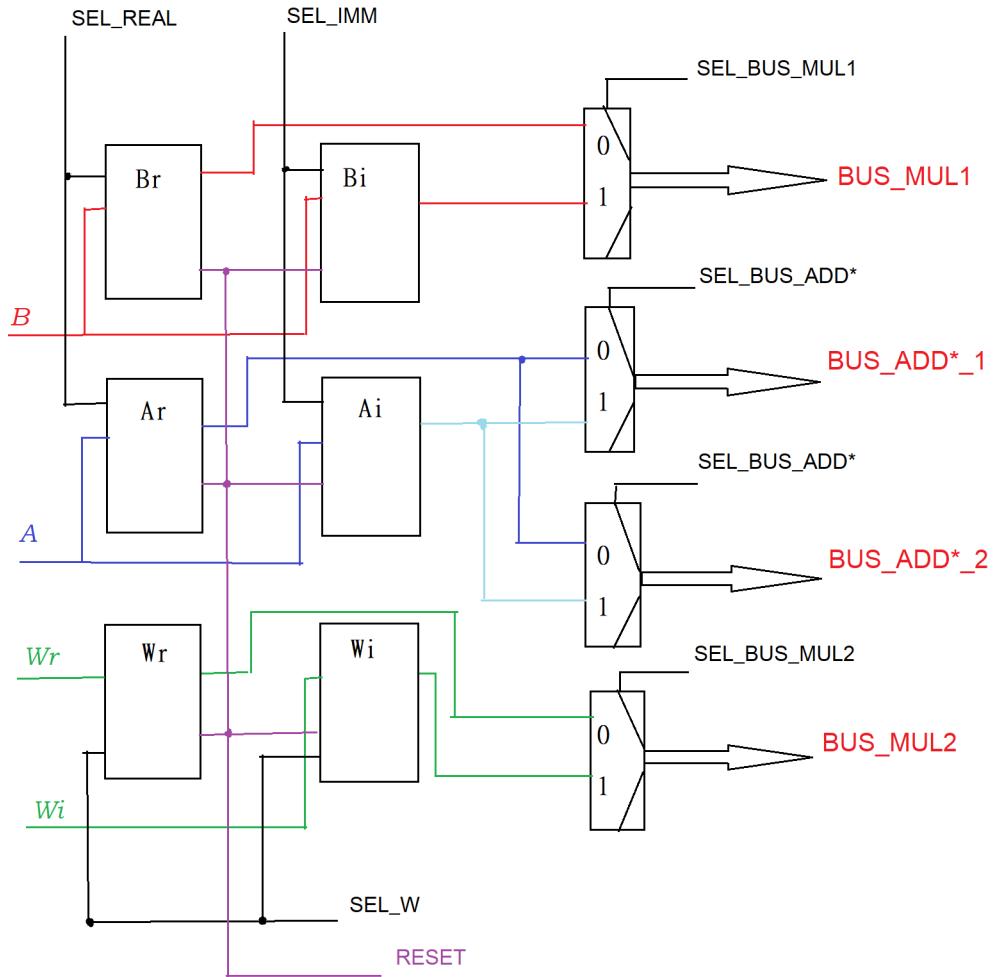


Figura 10: struttura interna del Register File

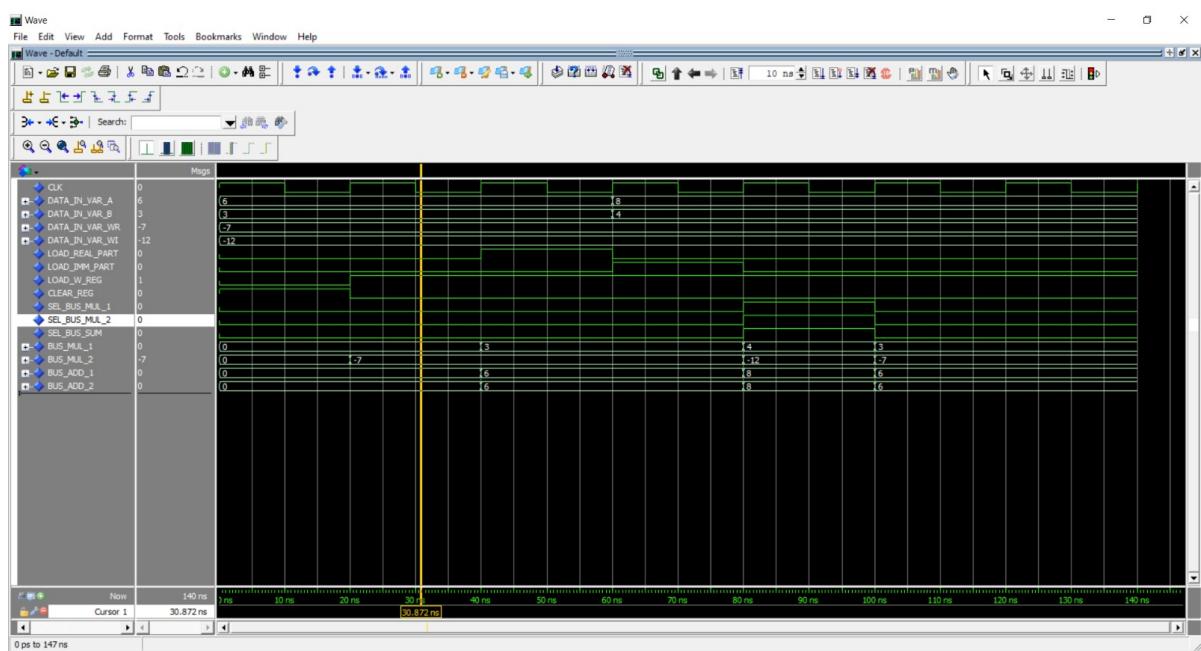


Figura 11: *Simulazione Modelsim del Register File*

5.2 Bus

Nella struttura presentata, sono necessari 4 bus globali: due bus sono utilizzati per mandare i dati in ingresso al moltiplicatore, altri due sono riservati all'invio dei dati in ingresso ai due sommatori (il secondo ingresso è la retroazione del risultato). Tale numero di bus globali risulta essere l'ottimo poiché non penalizza le prestazioni della macchina.

Il numero di connessioni dedicate per collegare gli elementi interni, sono tali da garantire il Resource Sharing dei blocchi. Dunque oltre ai collegamenti diretti tra registri e moltiplicatore/shift, nonché sommatori/sottrattori, sono stati inseriti ulteriori due collegamenti per effettuare correttamente la retroazione.

Il numero di bus interni, ognuno con il proprio parallelismo, è 10, numero ottimo che permette di ottenere la macchina con il throughput più veloce.

Se fosse stata scelta la stessa soluzione con 4 registri interni, il throughput sarebbe peggiorato, e il numero di bus interni sarebbe aumentato a 11 (per realizzare la retroazione in modo corretto nel registro da condividere).

Se fosse stata scelta la seconda soluzione di prova, presentata nella sezione precedente, il numero di bus interni sarebbe stato inferiore (6 bus interni), e il numero di bus globali pari a 3. Dunque per coerenza con la scelta progettuale cardine, si utilizza una struttura che è sia la più veloce, che quella che ottimizza il costo dei bus.

5.3 Moltiplicatore/shift

Il blocco moltiplicatore utilizzato nel Datapath scelto è di seguito riportato:

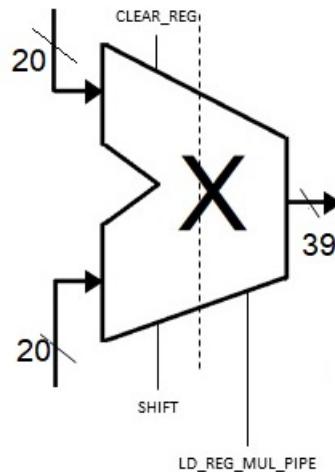


Figura 12: *blocco moltiplicatore/shift*

Questo presenta uno stadio di pipe per rispettare le specifiche, e anche un segnale di controllo a cui è assegnata la possibilità di moltiplicare per un fattore 2. Da come si può dedurre dalla precedente analisi del progetto, questo segnale non viene mai utilizzato. Il risultato della moltiplicazione viene memorizzato in un registro temporaneo e l'uscita di questo viene mandata su entrambi i 2 sommatori mediante l'utilizzo di 2 bus da 39 bit l'uno.

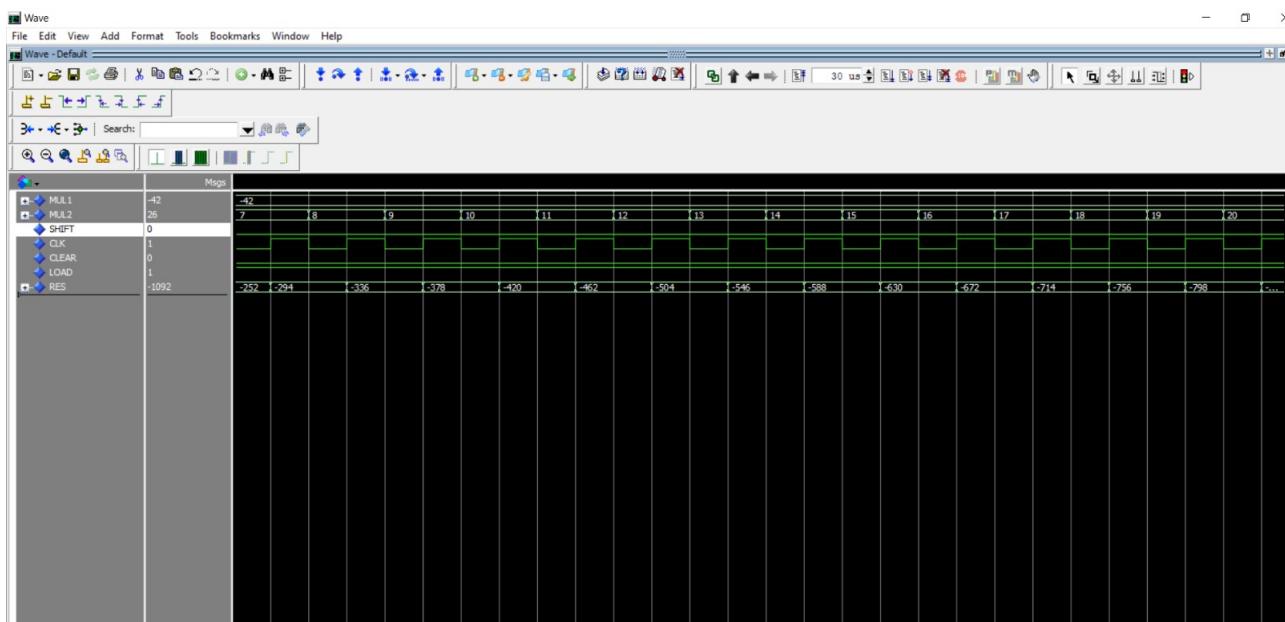


Figura 13: *simulazione Modelsim del blocco moltiplicatore/shift*

5.4 Sommatore/sottrattore

Ogni blocco sommatore/sottrattore è stato così realizzato.

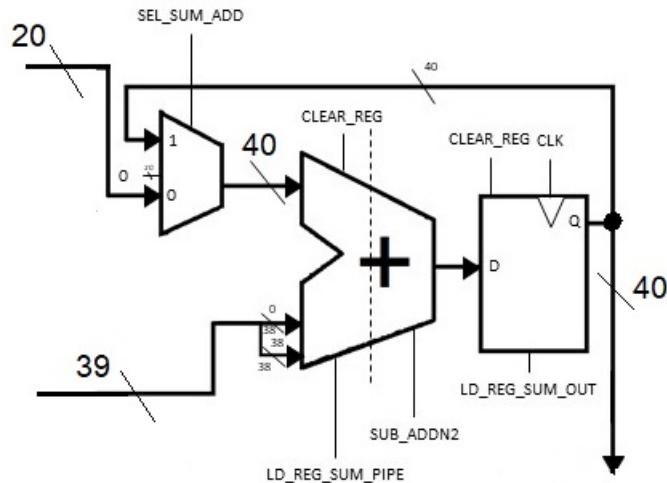


Figura 14: *blocco sommatore/sottrattore*

Nel datapath sono presenti 2 multiplexer (uno per ogni blocco sommatore/sottrattore) che ricevono in ingresso due dati su 40 bit:

- il primo è il dato proveniente dal Register File di cui i primi 20 bit (quelli più significativi) sono il dato e i restanti 20 bit sono posti a 0
- il secondo dato in ingresso è l'uscita del registro temporaneo in cui viene memorizzato il risultato della moltiplicazione.

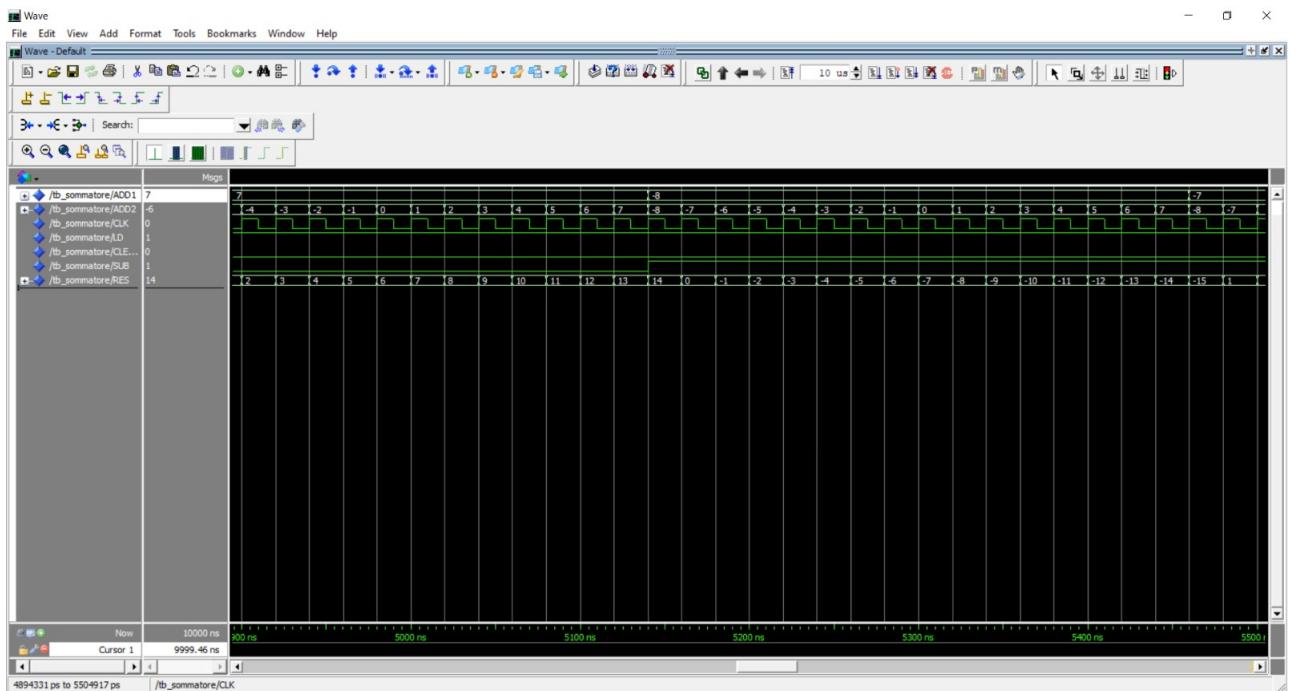


Figura 15: *similazione Modelsim del blocco sommatore/sottrattore*

5.5 Blocco Approssimazione

A valle di ogni registro temporaneo contenente il risultato del blocco somma/sottrazione, è presente il blocco presentato di seguito.

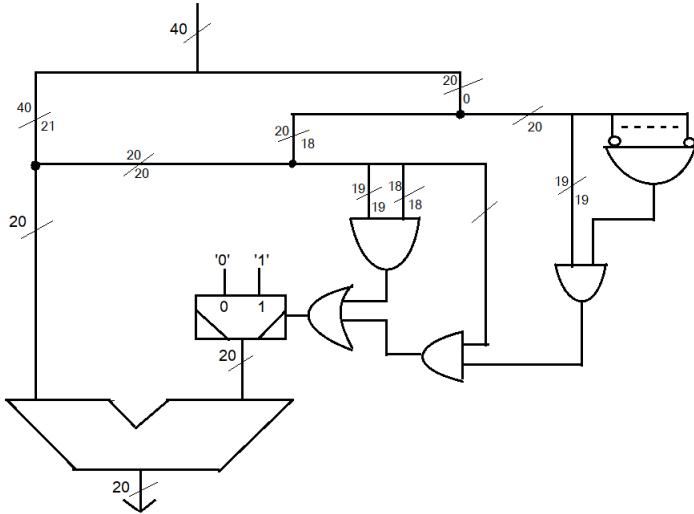


Figura 16: *blocco combinatorio dedicato all'operazione di approssimazione*

Si realizzano 2 blocchi di approssimazione che forniranno per via combinatoria il risultato finale che verrà memorizzato da 2 ulteriori registri di 20 bit, che conterranno i dati in uscita della Butterfly. Tali registri saranno attivati, in rispetto del timing, grazie a particolari segnali di enable provenienti dall'unità di controllo. Questi due blocchi sono stati progettati in modo tale da poter essere posizionati in fondo al Datapath, una volta che tutti gli step algoritmici sono conclusi, per evitare al minimo errori dovuti all'approssimazione.

E' stato realizzato il blocco approssimazione, puramente combinatorio, in grado di eseguire le regole dell'arrotondamento di tipo Round To Nearest Even.

Dato un numero in ingresso su N bit così formato X.d, dove il punto decimale rappresenta la cifra dalla quale bisogna effettuare l'arrotondamento e 'd' il numero di bit da eliminare, si avrà:

- troncamento quando il numero in ingresso è < metà intervallo: ossia quando le prime due cifre dopo il punto decimale sono 00 o 01
- somma di 1 quando il numero in ingresso è > della metà dell'intervallo: ossia quando le prime due cifre dopo il punto decimale sono 11
- approssimazione all'intero più vicino: se il numero si trova a metà intervallo

Nel caso in cui vengono rilevati i bit '10' dopo il punto decimale, bisogna verificare se il numero in ingresso si trova esattamente a metà intervallo. Questo si verifica quando i bit dopo la virgola sono 1 seguito da tutti 0 fino all'LSB.

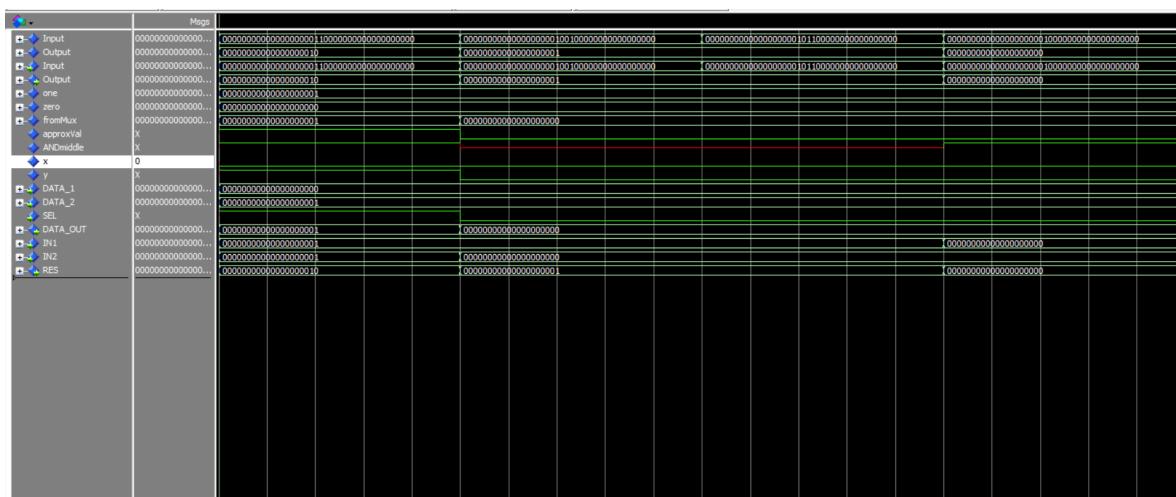


Figura 17: *similazione Modelsim del blocco approssimazione*

6 Protocollo di ingresso/uscita

Salendo ad un livello di astrazione più alto, la singola Butterfly sarà costituita da 2 bus dati per quanto concerne i valori di A e B, e 2 ulteriori bus dati per i valori di Wr e Wi. Dunque un possibile eventuale package contenente solo la Butterfly progettata avrà sei piedini dedicati agli ingressi e alle uscite.

In ingresso i dati saranno ricevuti nel seguente modo:

- Br, Bi
- Ar, Ai
- Wr
- Wi

In uscita i dati saranno inviati nel seguente modo:

- B'r, B'i
- A'r, A'i

Una soluzione alternativa poteva essere quella di dedicare un unico bus per l'invio dei Twiddle Factor, Wr e Wi. Nel caso progettato però, questa soluzione risulta essere limitante rispetto a quella con 2 bus.

Infatti i bus relativi ai W non devono necessariamente essere connessi ad una memoria, bensì ogni filo del bus parallelo può essere connesso a massa o all'alimentazione a seconda se si vuole avere il corrispondente bit a '0' oppure a 1.

Questa organizzazione degli ingressi e delle uscite sarà ottimale per poter interfacciare la Butterfly progettata, con altre identiche e per far lavorare la macchina nelle due modalità di funzionamento: isolata o continua.

7 Parallelismo

I dati in ingresso al PE sono definiti in forma frazionaria ($-1 < \text{dato} < 1$) in complemento a due (C2) su 20 bit. Per recuperare i possibili due bit di overflow teorici nella Butterfly, ottenibili dall'esecuzione delle operazioni complesse dell'algoritmo, i blocchi del datapath sono stati progettati con il seguente parallelismo.

Gli ingressi del moltiplicatore sono forniti su 2 bus da 20 bit ciascuno. Rispettando le specifiche fornite dal testo, l'uscita sarà su 39 bit.

L'uscita del sommatore sarà su 40 bit per il seguente ragionamento:

le 2 costanti W_r e W_i sono tra loro in relazione, infatti giacciono sul cerchio unitario di Eulero. Dunque W_r sarà pari a $\cos(\phi)$ mentre W_i pari a $\sin(\phi)$. Osservando le operazioni che vengono eseguite per ottenere i quattro risultati finali, si analizza il caso peggiore. Il valore massimo dei Twiddle Factor è rappresentato dall'uguaglianza in modulo della parte reale con la parte immaginaria, ossia quando sono pari entrambi a $\sqrt{2}/2$.

I dati A e B saranno forniti in ingresso nell'intervallo $-0.5 < \text{dato} < 0.5$ (estremi esclusi), per l'inserimento del bit di guardia. [la dinamica in ingresso in questo range, che permette di recuperare uno dei due bit di overflow teorici, è fornita direttamente dall'utente; pertanto non è stata prevista nessun tipo di operazione interna alla Butterfly capace di inserire il guard bit].

Rispettando le operazioni da eseguire, si osserva che il numero massimo ottenibile sarà pari a:

$$A'r = A_r + B_r W_r - B_i W_i = 0.5 + 0.5 * \sqrt{2} / 2 + 0.5 * \sqrt{2} / 2$$

Questo numero, calcolato in modo approssimativo poiché gli estremi dei dati in ingresso sono esclusi, risulta essere inferiore a 2.

Sapendo che il numero in uscita dal moltiplicatore è nel range $[-1, 1]$, otteniamo che sarà necessario 1 bit in più per non avere overflow; dunque le somme saranno fatte su un parallelismo di 40 bit.

8 Timing

Per quanto concerne il timing si riportano tre possibilità:

- modalità ONE SHOT
- modalità continua attivata 1 volta sola
- modalità continua attivata 2 volte.

Lo studio di queste tre possibilità è stato derivato dal fatto che la latenza massima del PE è di 9 colpi di clock da quando le parti reali di A e B sono disponibili, e che ogni 4 colpi di clock è possibile ricevere un nuovo dato.

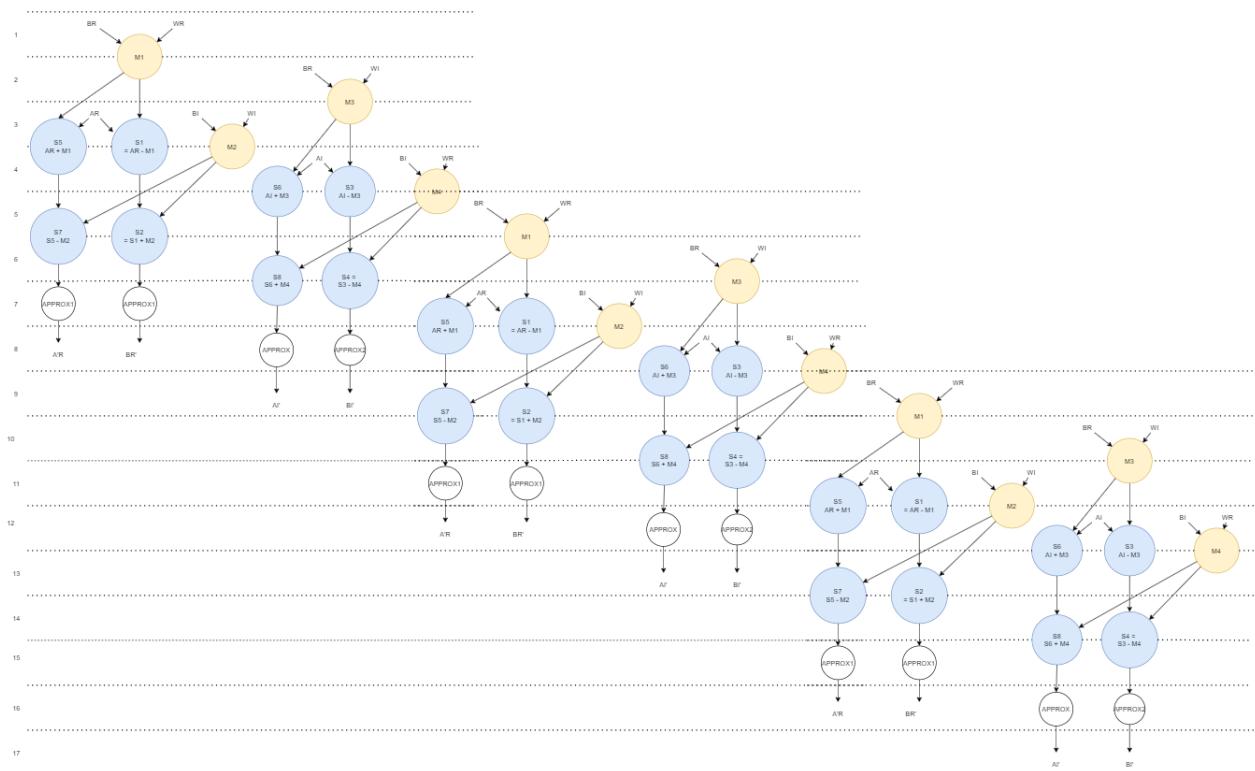
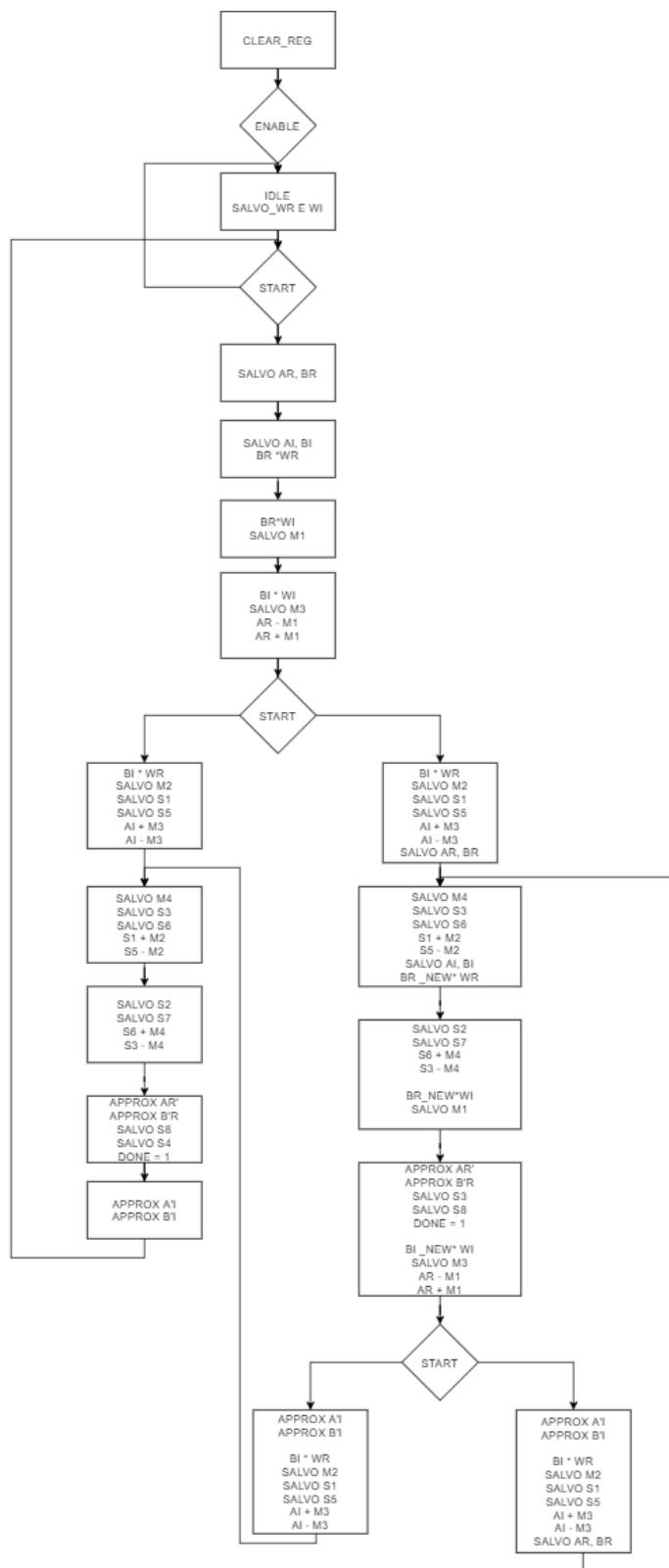


Figura 18: CDFD in esecuzione continua

Ciò permette di derivare l'ASM, di decidere il momento in cui ricevere il segnale di START (un colpo prima di ricevere i dati), e quando inviare il segnale di DONE. Di seguito è riportata la ASM complessiva:

Figura 19: *ASM della Butterfly*

I tre timing analizzati sono di seguito riportati. 1) funzionamento della macchina in modalità single_shot

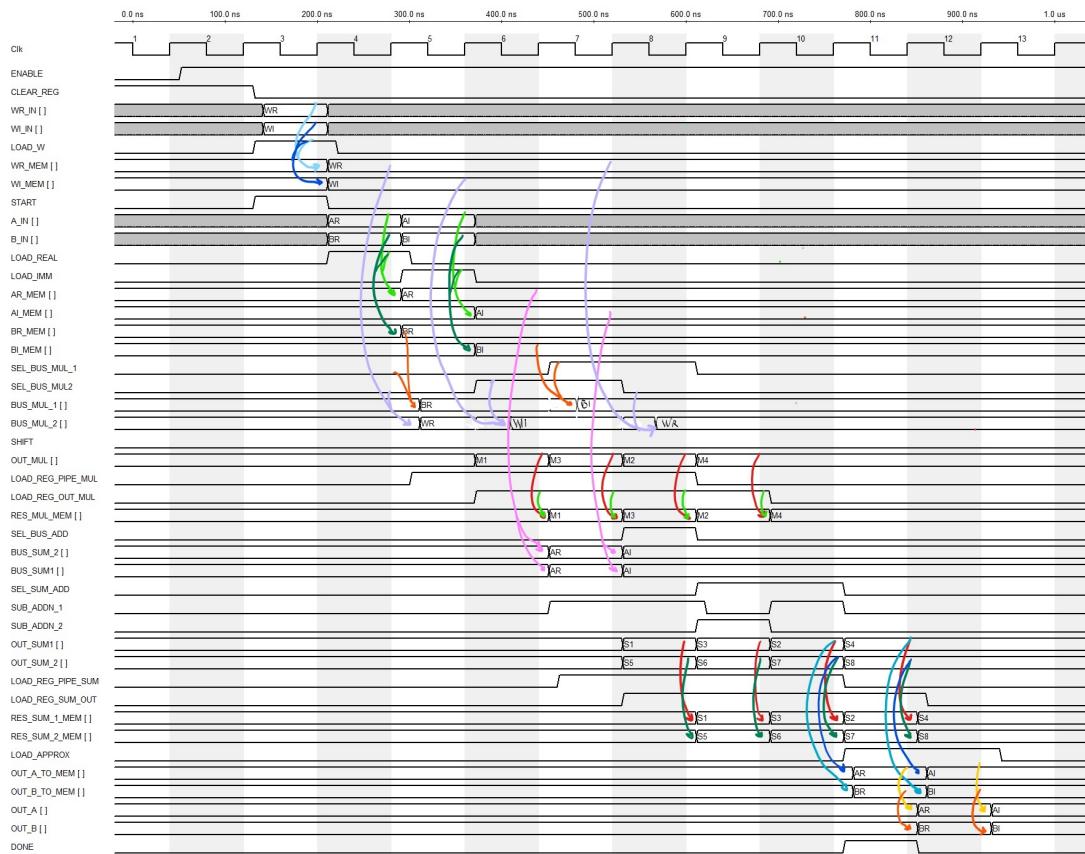


Figura 20: funzionamento single shot

2) funzionamento della macchina in modalità continua attivata una sola volta

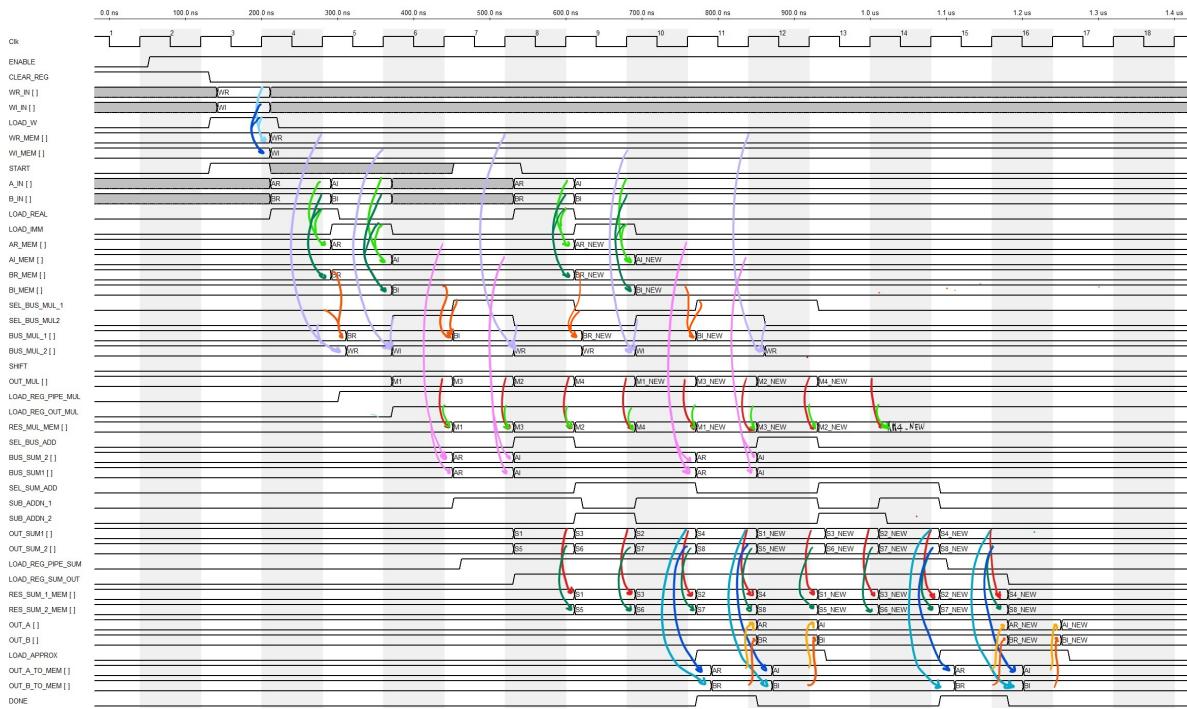


Figura 21: funzionamento in modalità continua attivata una volta

3) funzionamento della macchina in modalità continua attivata due volte

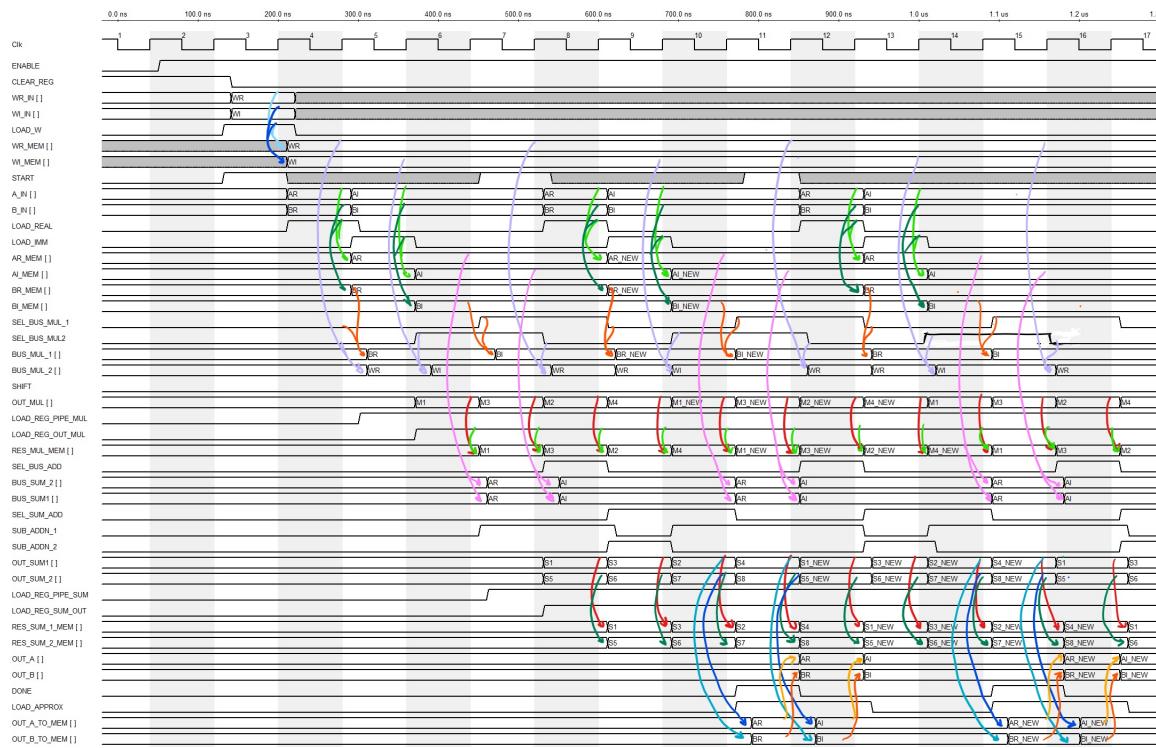


Figura 22: funzionamento in modalità continua attivata due volte

Il meccanismo che è stato implementato permette alla macchina di sapere se è un'esecuzione isolata o continua, evitando tempi morti tra una butterfly e l'altra. Per la sincronizzazione dei dati in ingresso e in uscita sono stati studiati i segnali di controllo START e DONE per avviare la macchina e per segnalare che un'elaborazione e' terminata e i dati sono validi.

9 Cenni teorici sulla control unit

9.1 Sequenziatore e generatore di comandi

L'Unità di Controllo è formata da due oggetti, il Sequencer e il Command Generator. Al primo è associato il compito di eseguire l'evoluzione della macchina a livello di pallogramma. Infatti, grazie agli Status e agli Inputs questo genera lo Step Corrente e determina quale dovrà essere il prossimo stato. Gli stati possono presentarsi in due modalità Coded o Uncoded (One Hot) e l'algoritmo di controllo può essere di due tipi: puramente sequenziale o con parte condizionale, per cui può prendere delle decisioni. Il command generator genera, in uscita, i comandi relativi allo stato corrente.

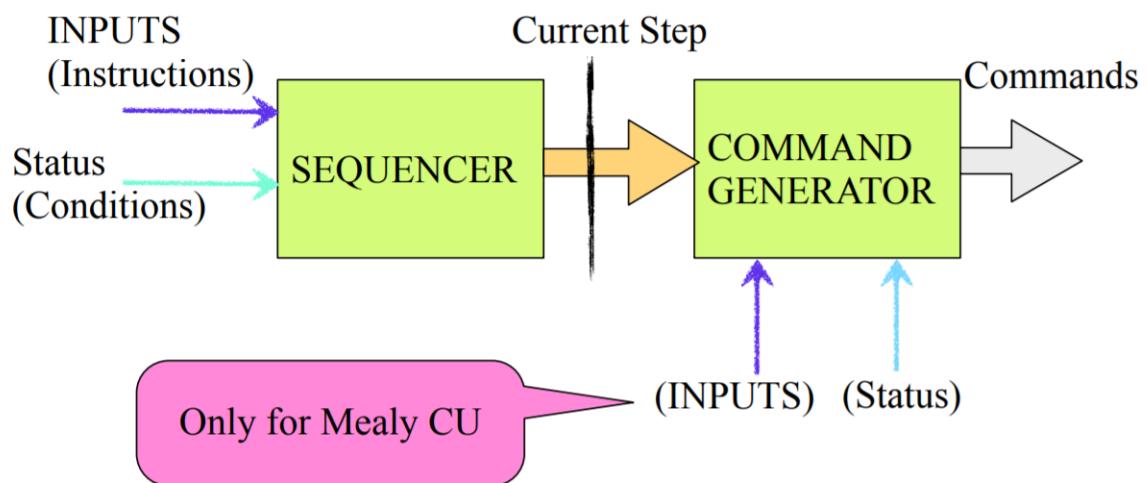


Figura 23: schema base di una unità di controllo

In prima analisi, per realizzare un sequenziatore con indirizzamento di tipo esplicito, è necessario utilizzare una tabella contenente gli stati futuri e puntata dallo stato presente.

Per quanto concerne la realizzazione di una macchina che sia in grado di compiere delle scelte, e dunque che abbia la capacità di effettuare dei salti in base a particolari condizioni, l'indirizzamento condizionale esplicito prevede varie implementazioni frutto di compromessi tra flessibilità, dimensioni e consumi.

Poiché nella maggior parte delle macchine a stati, in base all'algoritmo implementato, i salti che si effettuano sono del tipo ONLY-TWO-WAY-BRANCHES ALLOWED, l'architettura del sequenziatore si semplifica notevolmente. Nel caso di salti a due vie nell'implementazione con indirizzamento esplicito, infatti, nella tabella verrà sempre memorizzato il next address e verrà introdotto un blocco puramente combinatorio; lo STATUS PLA. Se l'algoritmo prevede di andare in sequenza, nel uAR sarà memorizzato il nuovo indirizzo.

Se l'algoritmo prevede un salto condizionale anche in questo caso il next address sarà unico ma privato dell'LSB poiché punterà a una coppia di locazioni successive [XXXX0 XXXX1]. Con la decisione dello Status Pla, un blocco che avrà in ingresso tutti i segnali necessari per discriminare l'effettiva condizione di salto, sarà generato il bit meno significativo opportuno, da memorizzare nel uAR.

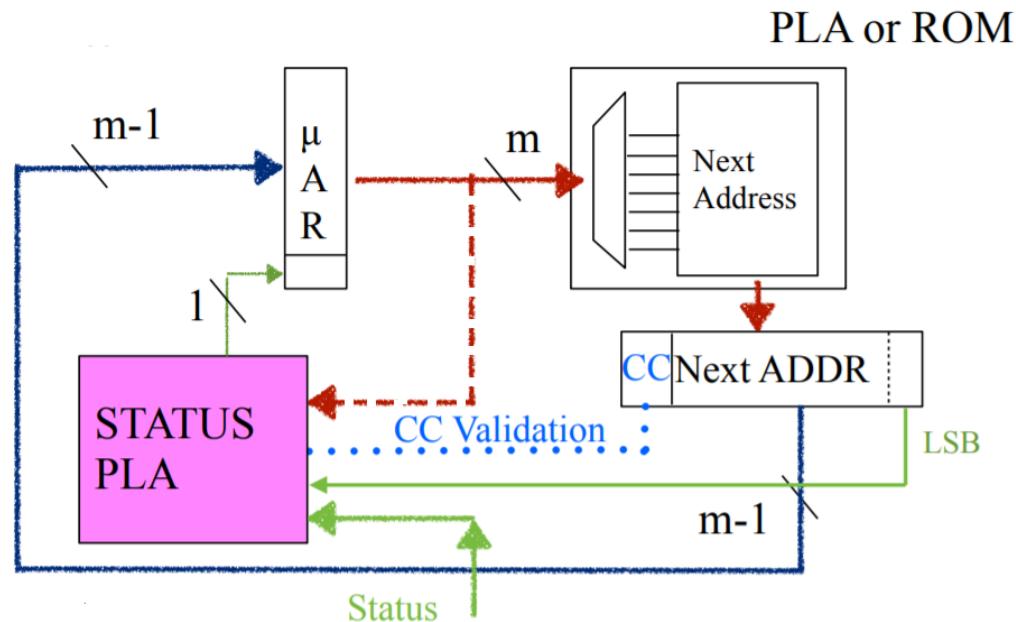


Figura 24: schema di una macchina in grado di effettuare salti con indirizzamento esplicito

Il Generatore di Comandi definisce per ogni stato corrente, lo stato dei comandi da mandare in uscita. È sempre una tabella puntata dal uAR (indirizzo che può essere codificato o non codificato (One Hot)), la cui uscita è memorizzata nel uIR. Per una macchina di Moore ad ogni stato è associata una riga dedicata nella tabella.

9.2 Late status PLA

La soluzione del Late Status PLA, presenta il seguente schema di principio.

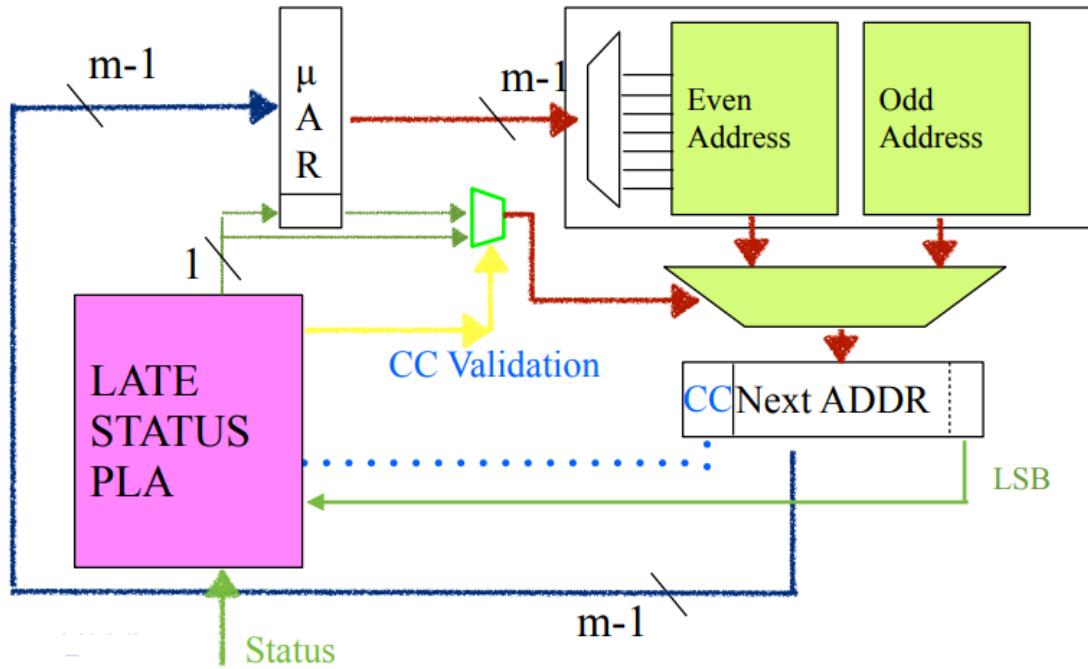


Figura 25: schema della soluzione Late Status PLA

È una soluzione che funziona solo per l'indirizzamento di tipo esplicito, per cui nella uROM c'è sempre l'indirizzo della prossima istruzione. Nel caso di salto l'organizzazione della tabella avviene sempre in modo tale che la nuova coppia di indirizzi sia allocata in due postazioni adiacenti. La decisione del next address viene gestita dal LATE STATUS PLA, il blocco combinatorio che riceve in ingresso le Condition Code, l'LSB e gli status dal Datapath.

L'organizzazione della uROM presenta un'architettura per cui le celle sono divise in Even Address e Odd Address; infatti, ad ogni accesso, poiché su ogni riga sono presenti sia la cella pari che la dispari, vengono lette due locazioni per volta. Questo implica che l'accesso alla tabella viene fatto indipendentemente dal Datapath e le performance vengono notevolmente aumentate perché l'operazione di accesso alla uROM viene fatta in maniera assolutamente concorrente con la scelta dell'LSB che dipenderà dalla decisione del late status.

10 Control unit della butterfly

10.1 Schema generale e stati

Nel particolare progetto dell'Unità di Controllo della Butterfly, vengono presi in esame gli aspetti descritti in precedenza. È stato necessario apportare delle modifiche allo schema di principio ottenendo la seguente organizzazione dell'Unità di controllo.

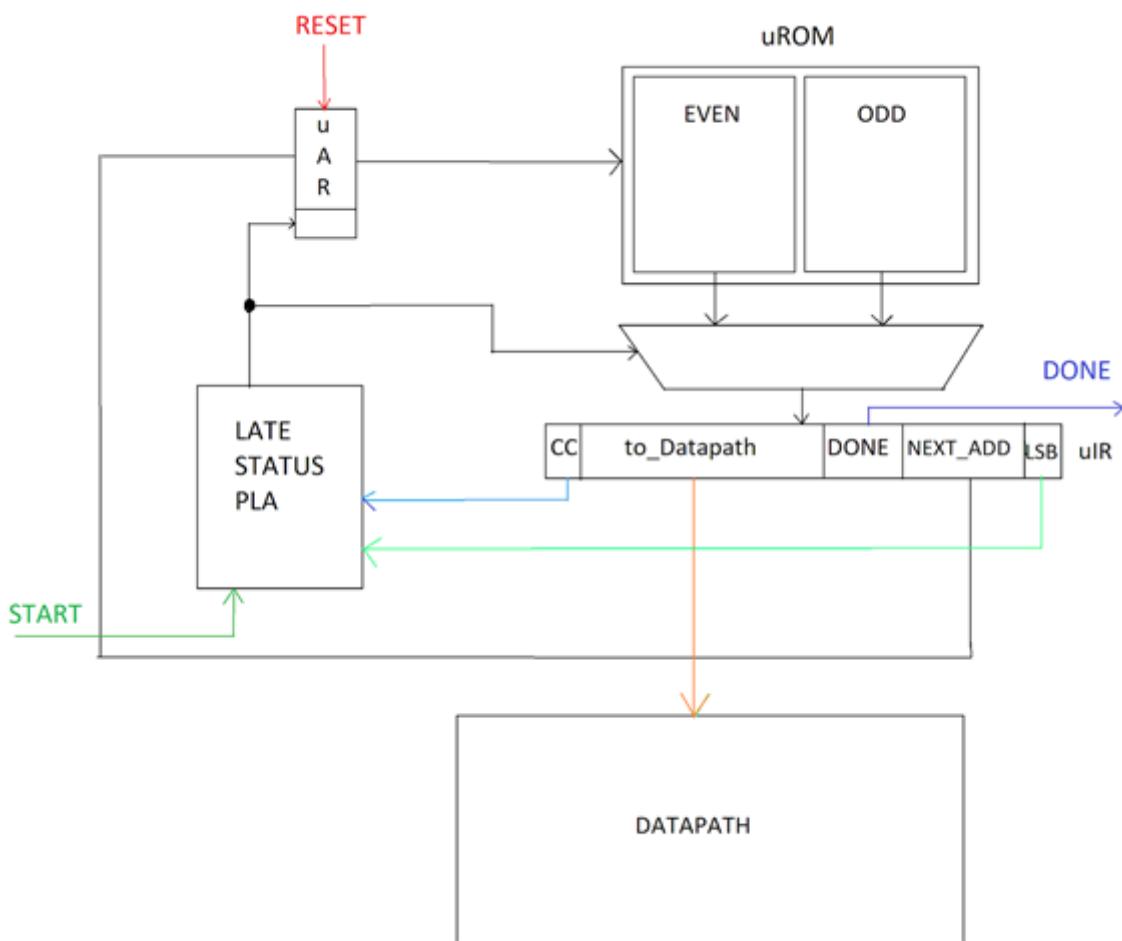


Figura 26: struttura dell'unità di controllo della Butterfly progettata

L'unico cambiamento strutturale effettuato sullo schema di principio è stata la rimozione del multiplexer che permette la scelta tra la provenienza dell'LSB dal late status oppure dal uAR.

Per comprendere a pieno le scelte fatte in merito all'implementazione dell'unità di Controllo, viene riportato il pallogramma di riferimento.

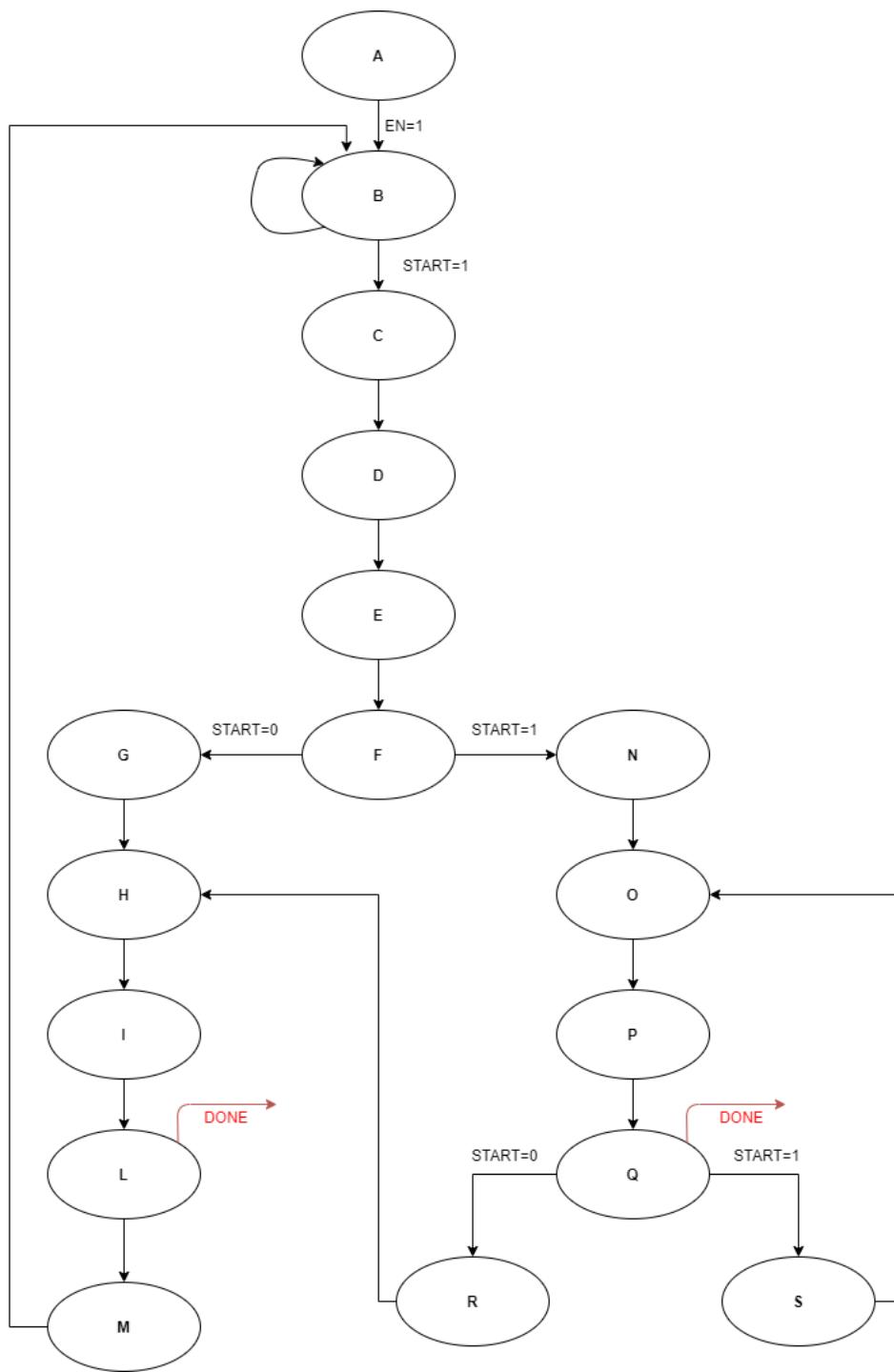


Figura 27: pallogramma della Butterfly

Si può notare come sia effettivamente minimo il numero di stati, esplicitamente descritti nella tabella seguente.

STA TO	
A	Stato di <u>enable</u> : clear dei registri
B	Stato di <u>idle</u> e caricamento dei W
C	Caricamento degli Ar e Br
D	Caricamento degli Ai e Bi. Moltiplicazione $M1 = Br^*Wr$
E	Moltiplicazione $M3=Br^*Wi$. Salvataggio di M1
F	Moltiplicazione $M2=Bi^*Wi$. Somma/sottrazione Ar+M1 e Ar-M1. Salvataggio M3
G	Moltiplicazione $M4=Bi^*WR$. Somma/sottrazione $AI+M3$ e $AI-M3$. Salvataggio M2, S1, S5
H	Somma/sottrazione $S1+M2$ e $S5-M2$. Salvataggio M4, S3, S6
I	Somma/sottrazione $S6+M4$ e $S3-M4$. Salvataggio S2, S7
L	Salvataggio S8, S4. Approssimazione Ar' , Br' . DONE=1
M	Approssimazione Aj , Bi .
N	Moltiplicazione $M4=Bi^*WR$. Somma/sottrazione $AI+M3$ e $AI-M3$. Salvataggio M2, S1, S5. Caricamento Ar, Br.
O	Somma $S1+M2$ e $S5-M2$. Salvataggio M4, S3, S6. Caricamento Ai, Bi. Moltiplicazione BR_NEW^*W
P	Somma/sottrazione $S6+M4$ e $S3-M4$. salvataggio S2, S7, M1. Moltiplicazione BR_NEW^*WI
Q	Salvataggio S3, S8. Approssimazione AR' e $B'R$. DONE = 1 moltiplicazione BI_NEW^*WI . Somma/sottrazione $AR-M1$, $AR+M1$. Salvataggio M3.
R	Approssimazione $A'l$ e $B'l$ Moltiplicazione $M4=Bi^*WR$. Somma/sottrazione $AI+M3$ e $AI-M3$. Salvataggio M2, S1, S5
S	Approssimazione $A'l$ e $B'l$ Moltiplicazione $M4=Bi^*WR$. Somma/sottrazione $AI+M3$ e $AI-M3$. Salvataggio M2, S1, S5. Salvataggio AR, BR

Figura 28: tabella degli stati

10.2 Organizzazione della uROM

Vengono riportate le scelte progettuali fatte in merito all'organizzazione della struttura interna della uROM. Ogni uIstruzione presenta la struttura mostrata nella tabella seguente:

	CC	to_Datapath	DONE	NEXT_ADD	LSB
n_bit	1	16	1	4	1

Figura 29: organizzazione della uIstruzione

Dalla seguente organizzazione si può comprendere come le dimensioni della uROM siano davvero esigue. Gli stati da codificare sono diciassette, dunque, per puntare alla singola cella di memoria sono necessari 5 bit [next_add + LSB]. Grazie alla tecnica del late status, gli stati saranno organizzati e distribuiti sulle celle pari e sulle celle dispari (dunque la struttura della uIstruzione è replicata sulle due colonne Even e Odd). Nel caso specifico, quindi, saranno necessarie solo dieci righe.

È stato previsto un bit per l'invio del segnale di DONE all'esterno, per avvisare che la Butterfly ha terminato la sua elaborazione.

È stato necessario l'inserimento di un ulteriore bit dedicato per il campo delle condizioni da inviare alla late status pla, per discriminare se effettuare o meno un salto. Dal pallogramma riportato in precedenza si può notare come tutti i salti presenti nella struttura avvengono sempre a causa della presenza o l'assenza del segnale di start proveniente dall'esterno. Non esistono infatti ulteriori condizioni tali da provocare ulteriori salti nella struttura. Dunque, l'unico bit contenuto in questo campo è stato nominato NO_JMP. È intuitivo il suo utilizzo: se NO_JMP = 1, allora non bisogna effettuare un salto; se NO_JMP = '0', allora bisogna effettuare un salto. Il campo to_Datapath è formato da 16 bit che saranno inviati direttamente all'unità di esecuzione. Questi bit sono così organizzati:

BIT	
0	Op_shift
1	Op_sub_addn_2
2	Op_sub_addn_1
3	Sel_sub_add
4	Sel_bus_sum
5	Sel_bus_mul_2
6	Sel_bus_mul_1
7	Load_approx
8	Load_req_out_sum
9	Load_req_pipe_sum
10	Load_req_out_mul
11	Load_req_pipe_mul
12	Load_imm
13	Load_real
14	Load_W
15	Clear

Figura 30: organizzazione dei bit da inviare al Datapath

All'MSB è assegnata l'operazione di clear dei registri, che verrà attivata durante lo stato di idle per far avvenire la pulizia della macchina.

Poiché i comandi da inviare al Datapath costituiscono un numero molto piccolo, è stata preferita la codifica di tipo One Hot per organizzare il Command Generator.

In base a questa preferenza di progetto in uscita dal seguente campo ci sarà una corrispondenza univoca tra il bit di stato presente nella uIstruzione e ogni singolo comando da mandare ai blocchi della Unità di Esecuzione. L'alternativa sarebbe stata una compressione dei comandi presenti nei sottoblocchi, in modo da ottenere un numero di bit utili nella sezione sicuramente inferiore a $n=16$. Le dimensioni della uROM sarebbero state più piccole e sicuramente sarebbero stati più veloci i tempi di accesso in memoria. Questa alternativa, allo stesso tempo, avrebbe portato l'inserimento necessario, a valle del registro uIR, di un blocco più o meno complesso specificamente realizzato per decodificare i comandi.

Nel caso implementato, questa seconda alternativa è stata scartata poiché nella macchina a stati realizzata, non è presente una complessità tale da essere necessaria una codifica e una successiva decodifica a valle.

Inoltre, come riportato nella descrizione della Unità di Esecuzione, in realtà una sorta di compressione e concorrenza è stata già adottata. Si nota come infatti molti blocchi vengono comandati dai medesimi segnali di controllo al fine di effettuare le operazioni differenti nello stesso colpo di clock. Esempi possono essere appunto: i segnali di clear dei registri, i segnali di load e così via.

Il vantaggio della scelta progettuale verso l'implementazione di tipo One Hot dei comandi da inviare all'Unità di Esecuzione ricade sul fatto che:

- Il tempo di accesso sarà più lento ma non sarà così significativamente superiore rispetto al caso in cui i bit appartenenti alla sezione to_Datapath sono codificati, poiché i bit in gioco sono un numero davvero ridotto e dunque il vantaggio sarebbe stato non considerevole.
- È certo che il tempo di accesso nella uROM sarà superiore, ma sicuramente è conveniente accettare un maggiore ritardo a questo livello, nel caso sotto esame, piuttosto che inserire blocchi combinatori di decodifica a valle, i quali richiedono un tempo computazionale opportuno, e quindi un nuovo ritardo addizionale da tenere in considerazione.

Infine, si riporta la tabella finale che schematizza la uROM:

ADD	CC	COMMANDS	DONE	NEXT_ADD	LSB	CC	COMMANDS	DONE	NEXT_ADD	LSB
0000X	1	1000000000000000	0	0000	1	0	0100000000000000	0	1111	(0)
0001X	1	0001100000000000	0	0001	1	1	0000110000100000	0	0010	0
0010X	0	0000111001100100	0	1110	(1)	1	0000011100001010	0	0011	0
0011X	1	0000001100001100	0	0011	1	1	0000000110000000	1	0100	0
0100X	1	0000000010000000	0	0000	1					
0101X	1	0001111100001010	0	0101	1	1	0000111100101100	0	0110	0
0110X	0	000011111100100	1	1101	(1)					
1101X	1	0000111111010100	0	0010	1	1	0010111111010100	0	0101	0
1110X	1	0000111101010100	0	0010	1	1	0010111101010100	0	0101	0
1111X	0	0100000000000000	0	0000	1	1	0010000000000000	0	0001	0

Figura 31: tabella contenuta nella uROM

La parte superiore della tabella è riservata alla gestione degli stati in sequenza. La parte inferiore della tabella è riservata alla gestione dei salti. L'organizzazione presentata è molto compatta. È presente solo una ripetizione di uno stato: l'IDLE. Questo è stato riportato sia nella postazione 00001 che 11110 per poter garantire la corretta esecuzione del primo salto. La presenza di eventuali errori nell'esecuzione del uCodice è stata risolta prevedendo che tutte le eventuali locazioni non esplicitamente codificate, ritornano nello stato A, ossia lo stato di RESET.

10.3 Late status PLA

Il blocco combinatorio della late status pla, nel progetto in esame è stato derivato in base allo studio dell'algoritmo della Butterfly. Come già descritto nella sezione precedente, la condizione per cui avvengono dei salti nella macchina a stati è unica, ossia quando in particolari istanti di clock è presente o meno il segnale di START proveniente dall'esterno. I segnali di stato provenienti dal datapath in questa particolare applicazione non ci sono, infatti non esistono condizioni dell'unità di esecuzione discriminanti per fare avvenire un salto.

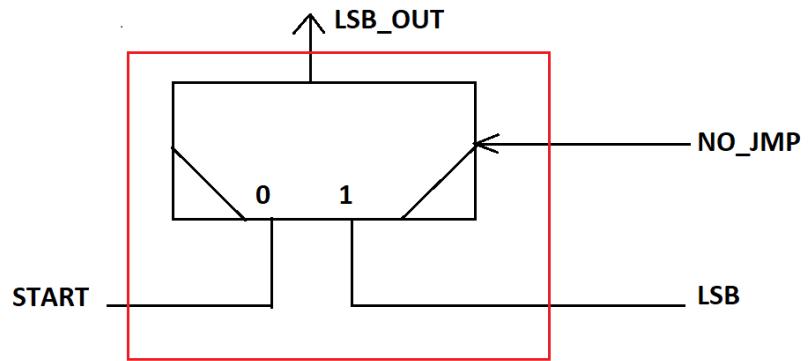
La macchina a stati della Butterfly è stata infatti pensata in modo tale che questa possa lavorare nelle due modalità richieste ossia la One Shot e la Full Speed.

- Il criterio di realizzazione della late status è il seguente:
- Quando non è previsto un salto, l'ultima cifra significativa da memorizzare nel uAR coincide con l'lsb del next address proveniente dal uIR
- Quando è previsto un salto, allora l'ultima cifra significativa da memorizzare nel uAR dipenderà dal segnale di status (lo START esterno)

Il blocco combinatorio della Late Status Pla riceve in ingresso:

- Status => START
- LSB
- CC => NO_JMP

Data la semplicità dell'algoritmo e data l'organizzazione della uROM, nel caso in esame, il blocco combinatorio realizzato è composto da un multiplexer. È necessario solo questo blocco combinatorio, dunque anche il tempo di decodifica a esso riservato, sarà molto breve. Di seguito è riportato lo schema realizzato.

Figura 32: *Late Status PLA*

10.4 uInstruction Register e uAddress Register

Gli attori che svolgono un ruolo principale nella gestione del controllo sono il uIR e il uAR, i due registi dai compiti singolari.

Il contenuto del uIR sarà l'indirizzo dell'istruzione successiva proveniente dalla uROM. Questo viene salvato nel seguente registro nel momento in cui viene discriminata, in fase decisionale, la riga e la colonna opportuna, da selezionare in base alla operazione da eseguire in un particolare colpo di clock. La struttura del uIR è la stessa di ogni colonna del uROM. Vengono infatti memorizzati 23 bit nell'ordine già descritto in precedenza:

- 1 MSB => CC
- 16 bit => to_Datapath
- 1 bit => DONE
- 4 bit => next_address
- 1 bit => LSB

Ogni gruppo di bit sarà opportunamente indirizzato. Il bit di ConditionCode e l'LSB entreranno nella late status pla. I 16 bit di comando saranno indirizzati all'unità di esecuzione. Il bit di done sarà mandato direttamente verso l'esterno.

Il uAR è un registro, invece, che campiona l'indirizzo dell'istruzione successiva (NS) e dà in uscita l'indirizzo dell'istruzione presente (PS). Le sue dimensioni sono inferiori rispetto al uIR, infatti nel caso implementato questo registro contiene solo gli ultimi 5 bit del registro descritto in precedenza, quattro dei quali punteranno ad un particolare indirizzo contenuto nella tabella della uROM. I quattro bit denominati con la voce 'next_address' nel uIR saranno salvati direttamente nel registro di indirizzo, mentre l'LSB verrà salvato dopo aver attraversato il percorso combinatorio del Late Status Pla.

Si nota inoltre che è stato inserito un segnale di RESET esterno, per poter resettare la macchina in qualsiasi momento. Nello schema del controllo implementato per la Butterfly non è stato previsto l'inserimento del multiplexer deputato alla scelta dell'Lsb. Il motivo di tale scelta progettuale si riferisce al timing dell'unità di controllo.

11 Timing della CU

Lo schema dell'unità di controllo è formato da due registri e, per fare eseguire una singola uIstruzione in un periodo di clock, è dunque opportuno valutare il timing del sistema. Essendo la macchina a stati della Butterfly estremamente semplice, è stato pensato di utilizzare un sistema ad una sola fase e campionare i due registri (uAR e uIR) sui due fronti del clock.

Non è stato necessario implementare un sistema multifase per varie ragioni. La prima tra tutte è sicuramente la semplicità dell'algoritmo sotto esame. Infatti, per come è stato presentato fino ad ora, non esistono salti in cui vengono discriminate condizioni prima del campionamento del uPC. La scelta della colonna opportuna della uROM può tranquillamente avvenire dal registro contenente il uIndirizzo.

In particolare, è stata implementata una struttura in cui il uIR e uAR vengono campionati nel seguente modo:

- uIR sul fronte di salita del colock
- uAR sul fronte di discesa del clock
- STATUS sul fronte di salita del clock

In questo modo è stato aggiunto un livello di pipe del controllo, ottenendo un sistema più veloce e concorrente. Questa scelta progettuale è risultata vincente poiché non è stato necessario inserire alcun tipo di NOP, dato che tutte le decisioni di salto avvengono immediatamente nello stesso colpo di clock (perchè è discriminante solo il segnale di start esterno, nonché bit già presenti nella uIstruzione).

12 Realizzazione butterfly 16x16

La butterfly 16x16 è stata realizzata tramite tre for generate annidati, i cui indici indicano rispettivamente:

- $i \rightarrow$ stadio delle butterfly;
- $j \rightarrow$ blocco dentro lo stadio;
- $k \rightarrow$ butterfly dentro al blocco.

Tra due stadi di butterfly vi sono due array di dimensione 16, X e S_D , rispettivamente relativi agli A e B su 20 bit e ad ai segnali di start e done. Per gli A ed i B questo array è stato definito come stage in un package apposito, BFpackage, in modo da poter essere usato anche nel file di testbench.

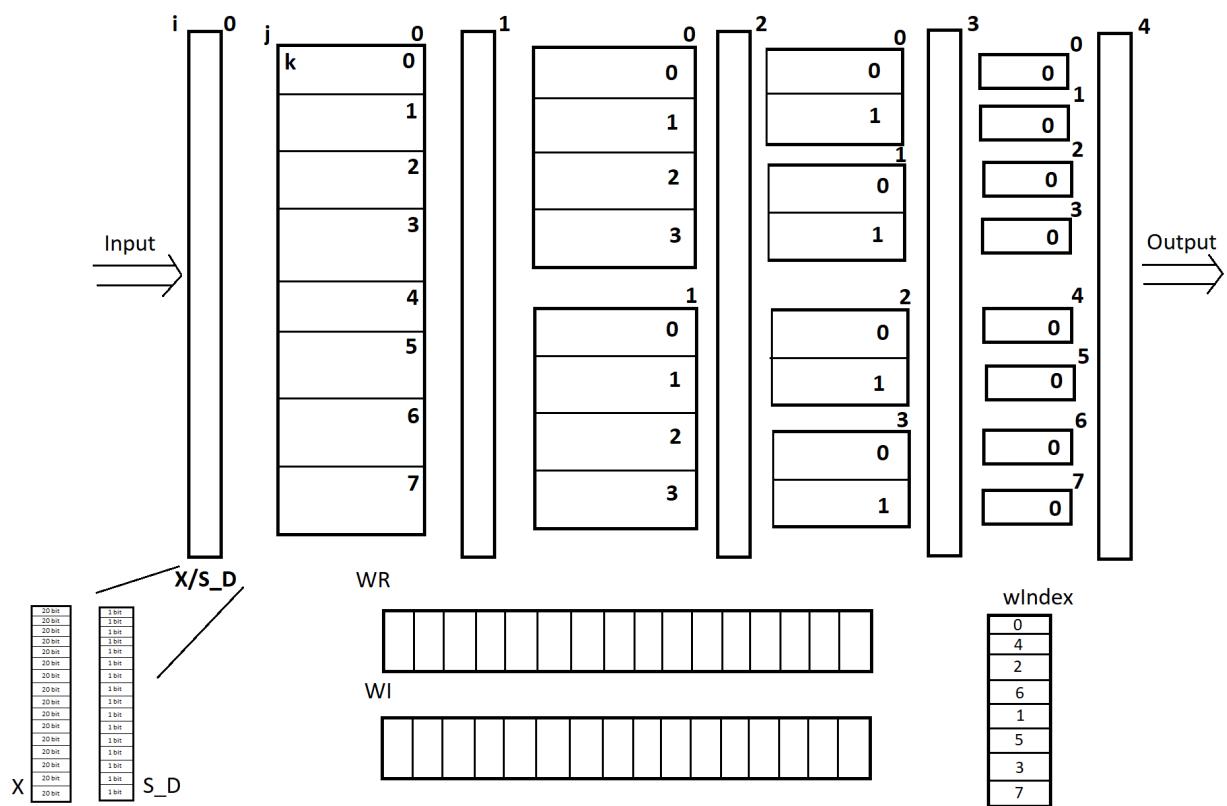


Figura 33: Schema della realizzazione della butterfly 16x16

Per collegare gli ingressi e le uscite delle singole butterfly è stato necessario individuare un metodo per ricongegare gli indici di X ed S_D a quelli dei for, ossia i , j e k ; i quali seguono la ramificazione della fft, ne è risultato il seguente port map:

- $A \rightarrow X(i)(k + j2^{4-i})$
- $B \rightarrow X(i)(k + j2^{4-i} + 2^{3-i})$
- $A' \rightarrow X(i+1)(k + j2^{4-i})$
- $B' \rightarrow X(i+1)(k + j2^{4-i} + 2^{3-i})$

- start -> $S_D(i)(k + j2^{3-i})$
- done -> $S_D(i+1)(k + j2^{3-i})$
- WR -> $WR(wIndex(j))$
- WI -> $WR(wIndex(j))$

Per i W è stato creato un array di interi, contenente gli indici ai quali la butterfly dovrà puntare per ottenere i W in base al blocco in cui si trova; infatti questo array wIndex sarà puntato da j.

13 Simulazioni

13.1 Spiegazione generale

I testbench della butterfly singola e 16x16 sono stati automatizzati tramite degli script in python; i quali generano i file con gli ingressi in numeri float, li convertono in formato binario in complemento a 2, e successivamente lanciano la compilazione dei listati ed il testbench. Il testbench si occuperà di estrarre dai file tutti i valori presenti, e di passare questi al DUT insieme al segnale di START, generato con opportuno anticipo. Ogni volta che il DUT restituisce il segnale di DONE, il testbench salverà i risultati ottenuti in un altro file. Una volta completato il testbench, lo script di python si occuperà di calcolare l'fft partendo dai dati in ingresso, confrontando poi il risultato atteso con quello del testbench, andando a calcolare gli errori massimo e medio, ed il rapporto tra l'errore massimo ed il massimo errore atteso; questo tipo di controllo viene utilizzato insieme ad un metodo Montecarlo per verificare il comportamento della macchina al di fuori dei test prefissati.

13.2 Singola butterfly

La simulazione della singola butterfly è stata eseguita testando le combinazioni tra i W considerati nel caso 16x16, e degli A e B compresi tra -1 ed 1 con un passo di 2^{-N} . Di sotto sono riportate le immagini dei testbench e le tabelle coi risultati del caso N = 2 (ossia un passo di 0.25).

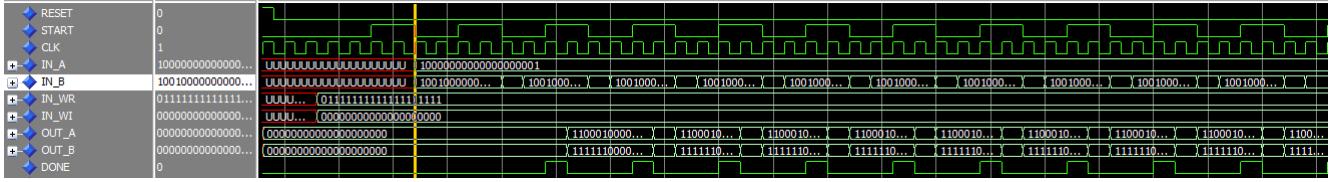


Figura 34: Testbench della singola butterfly

A	B	W	A' atteso	A' ottenuto	B'atteso	B' ottenuto
-1.0	-1.0j	-0.875	-0.875j1.0 -0.0j-1.875 -1.875j-1.8749961853027344	-1.8749961853027344j-0.125 -0.125j	-0.125	-0.125j
-1.0	-1.0j	-0.75	-0.875j1.0 -0.0j -1.75 -1.875j -1.7499961853027344	-1.8749961853027344j -0.25 -0.125j	-0.25	-0.125j
-1.0	-1.0j	-0.625	-0.875j1.0 -0.0j-1.625 -1.875j-1.6249961853027344	-1.8749961853027344j-0.375 -0.125j	-0.375	-0.125j
-1.0	-1.0j	-0.5	-0.875j1.0 -0.0j -1.5 -1.875j -1.4999961853027344	-1.8749961853027344j -0.5 -0.125j	-0.5	-0.125j
-1.0	-1.0j	-0.375	-0.875j1.0 -0.0j-1.375 -1.875j	-1.375 -1.8749961853027344j	-0.625	-0.125j
-1.0	-1.0j	-0.25	-0.875j1.0 -0.0j -1.25 -1.875j	-1.25 -1.8749961853027344j	-0.75	-0.125j
-1.0	-1.0j	-0.125	-0.875j1.0 -0.0j-1.125 -1.875j	-1.125 -1.8749961853027344j	-0.875	-0.125j
-1.0	-1.0j	0.0	-0.875j1.0 -0.0j -1.0 -1.875j	-1.0 -1.8749961853027344j	-1.0	-0.125j
-1.0	-1.0j	0.125	-0.875j1.0 -0.0j-0.875 -1.875j	-0.875 -1.8749961853027344j	-1.125	-0.125j
-1.0	-1.0j	0.25	-0.875j1.0 -0.0j -0.75 -1.875j	-0.75 -1.8749961853027344j	-1.25	-0.125j
-1.0	-1.0j	0.375	-0.875j1.0 -0.0j-0.625 -1.875j	-0.625 -1.8749961853027344j	-1.375	-0.125j
-1.0	-1.0j	0.5	-0.875j1.0 -0.0j -0.5 -1.875j	-0.5 -1.8749961853027344j	-1.5	-0.125j -1.4999961853027344 -0.125j
-1.0	-1.0j	0.625	-0.875j1.0 -0.0j-0.375 -1.875j	-0.375 -1.8749961853027344j	-1.625	-0.125j-1.6249961853027344 -0.125j
-1.0	-1.0j	0.75	-0.875j1.0 -0.0j -0.25 -1.875j	-0.25 -1.8749961853027344j	-1.75	-0.125j -1.7499961853027344 -0.125j
-1.0	-1.0j	0.875	-0.875j1.0 -0.0j-0.125 -1.875j	-0.125 -1.8749961853027344j	-1.875	-0.125j-1.8749961853027344 -0.125j
-1.0	-1.0j	1.0	-0.875j1.0 -0.0j 0.0 -1.875j	0.0 -1.8749961853027344j	-2.0	-0.125j -1.9999961853027344 -0.125j
-0.875	-1.0j	-0.875	-0.875j1.0 -0.0j -1.75 -1.875j	-1.75 -1.8749961853027344j	0.0	-0.125j
-0.875	-1.0j	-0.75	-0.875j1.0 -0.0j-1.625 -1.875j	-1.625 -1.8749961853027344j	-0.125	-0.125j
-0.875	-1.0j	-0.625	-0.875j1.0 -0.0j -1.5 -1.875j	-1.5 -1.8749961853027344j	-0.25	-0.125j
-0.875	-1.0j	-0.5	-0.875j1.0 -0.0j-1.375 -1.875j	-1.375 -1.8749961853027344j	-0.375	-0.125j
-0.875	-1.0j	-0.375	-0.875j1.0 -0.0j -1.25 -1.875j	-1.25 -1.8749961853027344j	-0.5	-0.125j
-0.875	-1.0j	-0.25	-0.875j1.0 -0.0j-1.125 -1.875j	-1.125 -1.8749961853027344j	-0.625	-0.125j
-0.875	-1.0j	-0.125	-0.875j1.0 -0.0j -1.0 -1.875j	-1.0 -1.8749961853027344j	-0.75	-0.125j
-0.875	-1.0j	0.0	-0.875j1.0 -0.0j-0.875 -1.875j	-0.875 -1.8749961853027344j	-0.875	-0.125j
-0.875	-1.0j	0.125	-0.875j1.0 -0.0j -0.75 -1.875j	-0.75 -1.8749961853027344j	-1.0	-0.125j
-0.875	-1.0j	0.25	-0.875j1.0 -0.0j-0.625 -1.875j	-0.625 -1.8749961853027344j	-1.125	-0.125j
-0.875	-1.0j	0.375	-0.875j1.0 -0.0j -0.5 -1.875j	-0.5 -1.8749961853027344j	-1.25	-0.125j
-0.875	-1.0j	0.5	-0.875j1.0 -0.0j-0.375 -1.875j	-0.375 -1.8749961853027344j	-1.375	-0.125j
-0.875	-1.0j	0.625	-0.875j1.0 -0.0j -0.25 -1.875j	-0.25 -1.8749961853027344j	-1.5	-0.125j
-0.875	-1.0j	0.75	-0.875j1.0 -0.0j-0.125 -1.875j	-0.125 -1.8749961853027344j	-1.625	-0.125j
-0.875	-1.0j	0.875	-0.875j1.0 -0.0j 0.0 -1.875j	0.0 -1.8749961853027344j	-1.75	-0.125j
-0.875	-1.0j	1.0	-0.875j1.0 -0.0j 0.125 -1.875j	0.12499618530273438 -1.8749961853027344j-1.875 -0.125j-1.8749961853027344 -0.125j	-1.875	-0.125j-1.8749961853027344 -0.125j
-0.75	-1.0j	-0.875	-0.875j1.0 -0.0j-1.625 -1.875j	-1.625 -1.8749961853027344j	0.125	-0.125j
-0.75	-1.0j	-0.75	-0.875j1.0 -0.0j -1.5 -1.875j	-1.5 -1.8749961853027344j	0.0	-0.125j
-0.75	-1.0j	-0.625	-0.875j1.0 -0.0j-1.375 -1.875j	-1.375 -1.8749961853027344j	-0.125	-0.125j

-0.75 -1.0j	-0.5 -0.875j	1.0 -0.0j	-1.25 -1.875j	-1.25 -1.8749961853027344j	-0.25 -0.125j	-0.25 -0.125j
-0.75 -1.0j	-0.375 -0.875j	1.0 -0.0j	-1.125 -1.875j	-1.125 -1.8749961853027344j	-0.375 -0.125j	-0.375 -0.125j
-0.75 -1.0j	-0.25 -0.875j	1.0 -0.0j	-1.0 -1.875j	-1.0 -1.8749961853027344j	-0.5 -0.125j	-0.5 -0.125j
-0.75 -1.0j	-0.125 -0.875j	1.0 -0.0j	-0.875 -1.875j	-0.875 -1.8749961853027344j	-0.625 -0.125j	-0.625 -0.125j
-0.75 -1.0j	0.0 -0.875j	1.0 -0.0j	-0.75 -1.875j	-0.75 -1.8749961853027344j	-0.75 -0.125j	-0.75 -0.125j
-0.75 -1.0j	0.125 -0.875j	1.0 -0.0j	-0.625 -1.875j	-0.625 -1.8749961853027344j	-0.875 -0.125j	-0.875 -0.125j
-0.75 -1.0j	0.25 -0.875j	1.0 -0.0j	-0.5 -1.875j	-0.5 -1.8749961853027344j	-1.0 -0.125j	-1.0 -0.125j
-0.75 -1.0j	0.375 -0.875j	1.0 -0.0j	-0.375 -1.875j	-0.375 -1.8749961853027344j	-1.125 -0.125j	-1.125 -0.125j
-0.75 -1.0j	0.5 -0.875j	1.0 -0.0j	-0.25 -1.875j	-0.25 -1.8749961853027344j	-1.25 -0.125j	-1.25 -0.125j
-0.75 -1.0j	0.625 -0.875j	1.0 -0.0j	-0.125 -1.875j	-0.125 -1.8749961853027344j	-1.375 -0.125j	-1.375 -0.125j
-0.75 -1.0j	0.75 -0.875j	1.0 -0.0j	0.0 -1.875j	0.0 -1.8749961853027344j	-1.5 -0.125j	-1.5 -0.125j
-0.75 -1.0j	0.875 -0.875j	1.0 -0.0j	0.125 -1.875j	0.125 -1.8749961853027344j	-1.625 -0.125j	-1.625 -0.125j
-0.75 -1.0j	1.0 -0.875j	1.0 -0.0j	0.25 -1.875j	0.24999618530273438 -1.8749961853027344j	-1.75 -0.125j	-1.7499961853027344 -0.125j
-0.625 -1.0j	-0.875 -0.875j	1.0 -0.0j	-1.5 -1.875j	-1.5 -1.8749961853027344j	0.25 -0.125j	0.25 -0.125j
-0.625 -1.0j	-0.75 -0.875j	1.0 -0.0j	-1.375 -1.875j	-1.375 -1.8749961853027344j	0.125 -0.125j	0.125 -0.125j
-0.625 -1.0j	-0.625 -0.875j	1.0 -0.0j	-1.25 -1.875j	-1.25 -1.8749961853027344j	0.0 -0.125j	0.0 -0.125j
-0.625 -1.0j	-0.5 -0.875j	1.0 -0.0j	-1.125 -1.875j	-1.125 -1.8749961853027344j	-0.125 -0.125j	-0.125 -0.125j
-0.625 -1.0j	-0.375 -0.875j	1.0 -0.0j	-1.0 -1.875j	-1.0 -1.8749961853027344j	-0.25 -0.125j	-0.25 -0.125j
-0.625 -1.0j	-0.25 -0.875j	1.0 -0.0j	-0.875 -1.875j	-0.875 -1.8749961853027344j	-0.375 -0.125j	-0.375 -0.125j
-0.625 -1.0j	-0.125 -0.875j	1.0 -0.0j	-0.75 -1.875j	-0.75 -1.8749961853027344j	-0.5 -0.125j	-0.5 -0.125j
-0.625 -1.0j	0.0 -0.875j	1.0 -0.0j	-0.625 -1.875j	-0.625 -1.8749961853027344j	-0.625 -0.125j	-0.625 -0.125j
-0.625 -1.0j	0.125 -0.875j	1.0 -0.0j	-0.5 -1.875j	-0.5 -1.8749961853027344j	-0.75 -0.125j	-0.75 -0.125j
-0.625 -1.0j	0.25 -0.875j	1.0 -0.0j	-0.375 -1.875j	-0.375 -1.8749961853027344j	-0.875 -0.125j	-0.875 -0.125j
-0.625 -1.0j	0.375 -0.875j	1.0 -0.0j	-0.25 -1.875j	-0.25 -1.8749961853027344j	-1.0 -0.125j	-1.0 -0.125j
-0.625 -1.0j	0.5 -0.875j	1.0 -0.0j	-0.125 -1.875j	-0.125 -1.8749961853027344j	-1.125 -0.125j	-1.125 -0.125j
-0.625 -1.0j	0.625 -0.875j	1.0 -0.0j	0.0 -1.875j	0.0 -1.8749961853027344j	-1.25 -0.125j	-1.25 -0.125j
-0.625 -1.0j	0.75 -0.875j	1.0 -0.0j	0.125 -1.875j	0.125 -1.8749961853027344j	-1.375 -0.125j	-1.375 -0.125j
-0.625 -1.0j	0.875 -0.875j	1.0 -0.0j	0.25 -1.875j	0.25 -1.8749961853027344j	-1.5 -0.125j	-1.5 -0.125j
-0.625 -1.0j	1.0 -0.875j	1.0 -0.0j	0.375 -1.875j	0.3749961853027344 -1.8749961853027344j	-1.625 -0.125j	-1.6249961853027344 -0.125j
-0.5 -1.0j	-0.875 -0.875j	1.0 -0.0j	-1.375 -1.875j	-1.375 -1.8749961853027344j	0.375 -0.125j	0.375 -0.125j
-0.5 -1.0j	-0.75 -0.875j	1.0 -0.0j	-1.25 -1.875j	-1.25 -1.8749961853027344j	0.25 -0.125j	0.25 -0.125j
-0.5 -1.0j	-0.625 -0.875j	1.0 -0.0j	-1.125 -1.875j	-1.125 -1.8749961853027344j	0.125 -0.125j	0.125 -0.125j
-0.5 -1.0j	-0.5 -0.875j	1.0 -0.0j	-1.0 -1.875j	-1.0 -1.8749961853027344j	0.0 -0.125j	0.0 -0.125j
-0.5 -1.0j	-0.375 -0.875j	1.0 -0.0j	-0.875 -1.875j	-0.875 -1.8749961853027344j	-0.125 -0.125j	-0.125 -0.125j
-0.5 -1.0j	-0.25 -0.875j	1.0 -0.0j	-0.75 -1.875j	-0.75 -1.8749961853027344j	-0.25 -0.125j	-0.25 -0.125j
-0.5 -1.0j	-0.125 -0.875j	1.0 -0.0j	-0.625 -1.875j	-0.625 -1.8749961853027344j	-0.375 -0.125j	-0.375 -0.125j
-0.5 -1.0j	0.0 -0.875j	1.0 -0.0j	-0.5 -1.875j	-0.5 -1.8749961853027344j	-0.5 -0.125j	-0.5 -0.125j
-0.5 -1.0j	0.125 -0.875j	1.0 -0.0j	-0.375 -1.875j	-0.375 -1.8749961853027344j	-0.625 -0.125j	-0.625 -0.125j
-0.5 -1.0j	0.25 -0.875j	1.0 -0.0j	-0.25 -1.875j	-0.25 -1.8749961853027344j	-0.75 -0.125j	-0.75 -0.125j
-0.5 -1.0j	0.375 -0.875j	1.0 -0.0j	-0.125 -1.875j	-0.125 -1.8749961853027344j	-0.875 -0.125j	-0.875 -0.125j
-0.5 -1.0j	0.5 -0.875j	1.0 -0.0j	0.0 -1.875j	0.0 -1.8749961853027344j	-1.0 -0.125j	-1.0 -0.125j
-0.5 -1.0j	0.625 -0.875j	1.0 -0.0j	0.125 -1.875j	0.125 -1.8749961853027344j	-1.125 -0.125j	-1.125 -0.125j
-0.5 -1.0j	0.75 -0.875j	1.0 -0.0j	0.25 -1.875j	0.25 -1.8749961853027344j	-1.25 -0.125j	-1.25 -0.125j
-0.5 -1.0j	0.875 -0.875j	1.0 -0.0j	0.375 -1.875j	0.375 -1.8749961853027344j	-1.375 -0.125j	-1.375 -0.125j
-0.5 -1.0j	1.0 -0.875j	1.0 -0.0j	0.5 -1.875j	0.4999961853027344 -1.8749961853027344j	-1.5 -0.125j	-1.4999961853027344 -0.125j
-0.375 -1.0j	-0.875 -0.875j	1.0 -0.0j	-1.25 -1.875j	-1.25 -1.8749961853027344j	0.5 -0.125j	0.5 -0.125j
-0.375 -1.0j	-0.75 -0.875j	1.0 -0.0j	-1.125 -1.875j	-1.125 -1.8749961853027344j	0.375 -0.125j	0.375 -0.125j
-0.375 -1.0j	-0.625 -0.875j	1.0 -0.0j	-1.0 -1.875j	-1.0 -1.8749961853027344j	0.25 -0.125j	0.25 -0.125j
-0.375 -1.0j	-0.5 -0.875j	1.0 -0.0j	-0.875 -1.875j	-0.875 -1.8749961853027344j	0.125 -0.125j	0.125 -0.125j
-0.375 -1.0j	-0.375 -0.875j	1.0 -0.0j	-0.75 -1.875j	-0.75 -1.8749961853027344j	0.0 -0.125j	0.0 -0.125j
-0.375 -1.0j	-0.25 -0.875j	1.0 -0.0j	-0.625 -1.875j	-0.625 -1.8749961853027344j	-0.125 -0.125j	-0.125 -0.125j

-0.375 -1.0j-0.125 -0.875j1.0 -0.0j -0.5 -1.875j	-0.5 -1.8749961853027344j	-0.25 -0.125j	-0.25 -0.125j
-0.375 -1.0j 0.0 -0.875j 1.0 -0.0j-0.375 -1.875j	-0.375 -1.8749961853027344j	-0.375 -0.125j	-0.375 -0.125j
-0.375 -1.0j 0.125 -0.875j 1.0 -0.0j -0.25 -1.875j	-0.25 -1.8749961853027344j	-0.5 -0.125j	-0.5 -0.125j
-0.375 -1.0j 0.25 -0.875j 1.0 -0.0j-0.125 -1.875j	-0.125 -1.8749961853027344j	-0.625 -0.125j	-0.625 -0.125j
-0.375 -1.0j 0.375 -0.875j 1.0 -0.0j 0.0 -1.875j	0.0 -1.8749961853027344j	-0.75 -0.125j	-0.75 -0.125j
-0.375 -1.0j 0.5 -0.875j 1.0 -0.0j 0.125 -1.875j	0.125 -1.8749961853027344j	-0.875 -0.125j	-0.875 -0.125j
-0.375 -1.0j 0.625 -0.875j 1.0 -0.0j 0.25 -1.875j	0.25 -1.8749961853027344j	-1.0 -0.125j	-1.0 -0.125j
-0.375 -1.0j 0.75 -0.875j 1.0 -0.0j 0.375 -1.875j	0.375 -1.8749961853027344j	-1.125 -0.125j	-1.125 -0.125j
-0.375 -1.0j 0.875 -0.875j 1.0 -0.0j 0.5 -1.875j	0.5 -1.8749961853027344j	-1.25 -0.125j	-1.25 -0.125j
-0.375 -1.0j 1.0 -0.875j 1.0 -0.0j 0.625 -1.875j	0.6249961853027344 -1.8749961853027344j	-1.375 -0.125j	-1.3749961853027344 -0.125j
-0.25 -1.0j -0.875 -0.875j1.0 -0.0j-1.125 -1.875j	-1.125 -1.8749961853027344j	0.625 -0.125j	0.625 -0.125j

Come possiamo osservare il singolo blocco rispetta il comportamento previsto. L'errore massimo ottenuto corrisponde esattamente a 2^{-18} , ossia l'errore massimo atteso dalla singola butterfly.

13.3 Butterfly 16x16

Di seguito sono riportati il testbench ed i risultati dei test eseguiti con gli esempi forniti dal docente.

Input	Atteso	Risultato
-0.5	-8.0	-0.0j -8.0 0.0j
-0.5	0.0	0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j
-0.5	0.0	0.0j 0.0 0.0j
-0.5	0.0	0.0j 0.0 0.0j
-0.5	0.0	0.0j 0.0 0.0j
-0.5	0.0	0.0j 0.0 0.0j
-0.5	0.0	0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j

Input	Atteso	Risultato
-0.5	0.0	-0.0j 0.0 0.0j
0	0.0	0.0j 0.0 0.0j
0.5	0.0	-0.0j 0.0 0.0j
0	0.0	0.0j 0.0 0.0j
-0.5	-4.0	0.0j -4.0 0.0j
0	0.0	0.0j 0.0 0.0j
0.5	0.0	0.0j 0.0 0.0j
0	0.0	0.0j 0.0 0.0j
-0.5	0.0	-0.0j 0.0 0.0j
0	0.0	-0.0j 0.0 0.0j
0.5	0.0	-0.0j 0.0 0.0j
0	0.0	-0.0j 0.0 0.0j
-0.5	-4.0	-0.0j -4.0 0.0j
0	0.0	-0.0j 0.0 0.0j
0.5	0.0	0.0j 0.0 0.0j
0	0.0	-0.0j 0.0 0.0j

Input	Atteso	Risultato
0.5	0.5	-0.0j
0	0.5	0.0j
0	0.5	-0.0j
0	0.5	0.0j
0	0.5	-0.0j

Input	Atteso	Risultato
-0.5	0.0	-0.0j
-0.5	0.0	0.0j
0.5	0.0	-0.0j
0.5	0.0	0.0j
-0.5	-4.0	4.0j
-0.5	0.0	0.0j
0.5	0.0	0.0j
0.5	0.0	0.0j
-0.5	0.0	-0.0j
-0.5	0.0	-0.0j
0.5	0.0	-0.0j
0.5	0.0	-0.0j
-0.5	-4.0	-4.0j
-0.5	0.0	-0.0j
0.5	0.0	0.0j
0.5	0.0	-0.0j

Input	Atteso	Risultato
0.5	0.0 -0.0j	0.0 0.0j
0.5	1.0 -5.027339492125848j	1.0 -5.02734375j
0.5	0.0 -0.0j	0.0 0.0j
0.5	1.0 -1.496605762665489j	1.0 -1.49658203125j
0.5	0.0 0.0j	0.0 0.0j
0.5	1.0 -0.6681786379192987j	1.0 -0.668212890625j
0.5	0.0 0.0j	0.0 0.0j
0.5	1.0 -0.19891236737965823j	1.0 -0.19891357421875j
-0.5	0.0 -0.0j	0.0 0.0j
-0.5	1.0 0.19891236737965823j	1.0 0.19891357421875j
-0.5	0.0 -0.0j	0.0 0.0j
-0.5	1.0 0.6681786379192987j	1.0 0.66815185546875j
-0.5	0.0 -0.0j	0.0 0.0j
-0.5	1.0 1.496605762665489j	1.0 1.49658203125j
-0.5	0.0 0.0j	0.0 0.0j
-0.5	1.0 5.027339492125848j	1.0 5.02734375j

Input	Atteso	Risultato
0	0.365 -0.0j	0.364990234375 0.0j
0	-0.365 0.0j	-0.364990234375 0.0j
0	0.365 -0.0j	0.364990234375 0.0j
0	-0.365 0.0j	-0.364990234375 0.0j
0	0.365 0.0j	0.364990234375 0.0j
0	-0.365 0.0j	-0.364990234375 0.0j
0	0.365 0.0j	0.364990234375 0.0j
0	-0.365 0.0j	-0.364990234375 0.0j
0.365	0.365 -0.0j	0.364990234375 0.0j
0	-0.365 -0.0j	-0.364990234375 0.0j
0	0.365 -0.0j	0.364990234375 0.0j
0	-0.365 -0.0j	-0.364990234375 0.0j
0	0.365 -0.0j	0.364990234375 0.0j
0	-0.365 -0.0j	-0.364990234375 0.0j
0	0.365 0.0j	0.364990234375 0.0j
0	-0.365 -0.0j	-0.364990234375 0.0j

Input	Atteso	Risultato
1	16.0 -0.0j	15.99993896484375 0.0j
1	0.0 0.0j	0.0 0.0j
1	0.0 -0.0j	0.0 0.0j
1	0.0 0.0j	0.0 0.0j
1	0.0 0.0j	0.0 0.0j
1	0.0 0.0j	0.0 0.0j
1	0.0 0.0j	0.0 0.0j
1	0.0 0.0j	0.0 0.0j
1	0.0 0.0j	0.0 0.0j
1	0.0 -0.0j	0.0 0.0j
1	0.0 -0.0j	0.0 0.0j
1	0.0 -0.0j	0.0 0.0j
1	0.0 -0.0j	0.0 0.0j
1	0.0 -0.0j	0.0 0.0j
1	0.0 0.0j	0.0 0.0j
1	0.0 0.0j	0.0 0.0j

Input	Atteso	Risultato
-1	-16.0 -0.0j	-15.99993896484375 0.0j
-1	0.0 0.0j	0.0 0.0j
-1	0.0 -0.0j	0.0 0.0j
-1	0.0 0.0j	0.0 0.0j
-1	0.0 0.0j	0.0 0.0j
-1	0.0 0.0j	0.0 0.0j
-1	0.0 0.0j	0.0 0.0j
-1	0.0 0.0j	0.0 0.0j
-1	0.0 -0.0j	0.0 0.0j
-1	0.0 -0.0j	0.0 0.0j
-1	0.0 -0.0j	0.0 0.0j
-1	0.0 -0.0j	0.0 0.0j
-1	0.0 0.0j	0.0 0.0j
-1	0.0 0.0j	0.0 0.0j
-1	0.0 0.0j	0.0 0.0j

In queste simulazioni è stato osservato un errore massimo di $6.103515625\text{e-}05$, approssimabile a 2^{-14} , ed un errore medio di $1.5537020555279196\text{e-}06$. Per avere una stima migliore del comportamento della macchina sotto questo punto di vista, è stata eseguita una simulazione con 10000 valori casuali, la quale ha riportato i seguenti risultati:

- Errore massimo: 0.00010455330198633206 ;
- Errore medio: $2.2084092421059654\text{e-}05$.

Possiamo così sostenere oltre alla correttezza comportamentale della butterfly 16x16 la sua stabilità e precisione, che in una stima pessimistica possiamo ipotizzare valere 2^{-13} .

14 Appendice

14.1 Listati VHDL e Python

Di seguito sono riportati i listati VHDL e Python

14.2 Adder.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Adder is
    generic(N : positive);
    port(
        IN1, IN2: in signed(N-1 downto 0);
        RES: out signed(N-1 downto 0));
end entity Adder;

architecture behavioral of Adder is

begin
    RES <= IN1 + IN2;
end behavioral;
```

14.3 approx.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity approx is
    generic(N : positive);
    port(
        --ingresso proveniente dal registro del sommatore
        Input: in signed((2*N)-1 downto 0);

        --uscita approssimata
        Output: out signed(N-1 downto 0));
end entity approx;

architecture behavioral of approx is

component mux2to1 is
    generic (N : positive);
    port (
        DATA_1 : IN SIGNED (N - 1 downto 0);
        DATA_2 : IN SIGNED (N - 1 downto 0);
        SEL : IN STD_LOGIC;
        DATA_OUT : OUT SIGNED
    );
end component;

component Adder is
    generic(N : positive);
    port(
        IN1, IN2: in signed(N-1 downto 0);
        RES: out signed(N-1 downto 0));
end component;

--segnale per sommare 1 secondo le regole di approssimazione
--del round to nearest even
signal one: signed(N-1 downto 0);

--segnale per sommare 0 secondo le regole di approssimazione
--del round to nearest even
signal zero: signed(N-1 downto 0);

--segnale in uscita al multiplexer che ha come output il secondo addendo
--della somma (1 o 0)
signal fromMux: signed (N-1 downto 0);

--segnale di selezione del multiplexer
signal approxVal: std_logic;

```

```

--segnale utilizzato per verificare se si è a metà intervallo
signal ANDmiddle: std_logic;

signal x: std_logic;           --segnale intermedio
signal y: std_logic;           --segnale intermedio

begin

--loop generato per verificare di essere a metà intervallo
ANDloop: for i in 0 to (N-3) generate
begin
ANDmiddle <= ((not(Input(i))) and (not(Input (i+1)))); 
end generate ANDloop;

x <= ( (Input(N-1)) and (Input(N-2)));
y <= ( (Input(N)) and (( (Input (N-1)) and (ANDmiddle) ) ) );
approxVal <= x or y;

--conversione del valore '0' su 20 bit nel formato signed
zero <= to_signed(0,20);

--conversione del valore '1' su 20 bit nel formato signed
one <= to_signed(1,20);

AppMUX : mux2to1 generic map (n => N)
port map (
    DATA_1 => zero,
    DATA_2 => one,
    SEL => approxVal,
    DATA_OUT => fromMUX);

AppADD : Adder generic map (N => N)
port map(
    IN1 => Input((2*N)-1 downto (N)),
    IN2 => fromMUX,
    RES => Output);

end behavioral;

```

14.4 Butterfly.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Butterfly is
    generic(N : positive := 20);
    port (
        RESET : IN STD_LOGIC;
        START : IN STD_LOGIC;
        CLK : IN STD_LOGIC;
        IN_A : IN SIGNED(N - 1 DOWNTO 0);
        IN_B : IN SIGNED(N - 1 DOWNTO 0);
        IN_WR : IN SIGNED(N - 1 DOWNTO 0);
        IN_WI : IN SIGNED(N - 1 DOWNTO 0);
        OUT_A : OUT SIGNED(N - 1 DOWNTO 0);
        OUT_B : OUT SIGNED(N - 1 DOWNTO 0);
        DONE : OUT STD_LOGIC
    );
end Butterfly;

architecture beh of Butterfly is
    component cu is
        port(
            reset, start, clock: in std_logic;
            toDatapath: out std_logic_vector(15 downto 0);
            done : out std_logic
        );
    end component;

    component butterflyDatapath is
        generic (N : positive);
        port (
            CLK : IN STD_LOGIC;

            -- data ingresso
            DATA_IN_VAR_A : IN SIGNED (N - 1 downto 0);
            DATA_IN_VAR_B : IN SIGNED (N - 1 downto 0);
            DATA_IN_VAR_WR : IN SIGNED (N - 1 downto 0);
            DATA_IN_VAR_WI : IN SIGNED (N - 1 downto 0);

            -- decoder per input A e B
            LOAD_REAL : IN STD_LOGIC;
            LOAD_IMM : IN STD_LOGIC;

            -- segnali per W
            LOAD_W_REG : IN STD_LOGIC;

            CLEAR_REG : IN STD_LOGIC;

            -- selezione bus
        );
    end component;

```

```

SEL_BUS_MUL_1 : IN STD_LOGIC;
SEL_BUS_MUL_2 : IN STD_LOGIC;
SEL_BUS_SUM : IN STD_LOGIC;

-- segnali moltiplicatore
LOAD_MUL_REG_PIPE : IN STD_LOGIC;
SHIFT : IN STD_LOGIC;

-- somma pipe
LOAD_SUM_REG_PIPE : IN STD_LOGIC;
SUB_ADDN1 : IN STD_LOGIC;
SUB_ADDN2 : IN STD_LOGIC;

-- load approx
LOAD_APPROX: in std_logic;

LOAD_REG_RES_SUM : IN STD_LOGIC;
LOAD_REG_RES_MUL : IN STD_LOGIC;

SEL_SUM_ADD : IN STD_LOGIC;

-- bus uscita
OUT_A : OUT SIGNED(N - 1 downto 0);
OUT_B : OUT SIGNED(N - 1 DOWNTO 0)

);

end component;

signal SIGNAL_TO_DATAPATH : std_logic_vector(15 downto 0);

begin
CU_PART : cu
port map(
    reset => RESET, start => START, clock => CLK,
    toDatapath => SIGNAL_TO_DATAPATH,
    done => DONE
);

DATAPATH: butterflyDatapath
generic map(N => N)
port map(
    CLK => CLK,

-- data ingresso
    DATA_IN_VAR_A => IN_A,
    DATA_IN_VAR_B => IN_B,
    DATA_IN_VAR_WR => IN_WR,
    DATA_IN_VAR_WI => IN_WI,

-- decoder per input A e B

```

```

LOAD_REAL => SIGNAL_TO_DATAPATH(13),
LOAD_IMM => SIGNAL_TO_DATAPATH(12),
LOAD_W_REG => SIGNAL_TO_DATAPATH(14),

CLEAR_REG => SIGNAL_TO_DATAPATH(15),

-- selezione bus
SEL_BUS_MUL_1 => SIGNAL_TO_DATAPATH(6),
SEL_BUS_MUL_2 => SIGNAL_TO_DATAPATH(5),
SEL_BUS_SUM => SIGNAL_TO_DATAPATH(4),

-- segnali moltiplicatore
LOAD_MUL_REG_PIPE => SIGNAL_TO_DATAPATH(11),
SHIFT => SIGNAL_TO_DATAPATH(0),

-- somma pipe
LOAD_SUM_REG_PIPE => SIGNAL_TO_DATAPATH(9),
SUB_ADDN1 => SIGNAL_TO_DATAPATH(2),
SUB_ADDN2 => SIGNAL_TO_DATAPATH(1),
LOAD_APPROX => SIGNAL_TO_DATAPATH(7),

LOAD_REG_RES_SUM => SIGNAL_TO_DATAPATH(8),
LOAD_REG_RES_MUL => SIGNAL_TO_DATAPATH(10),

SEL_SUM_ADD => SIGNAL_TO_DATAPATH(3),

OUT_A => OUT_A,
OUT_B => OUT_B

);

end beh;

```

14.5 Butterfly16x16.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--importo il package creato apposta per la butterfly 16x16
library work;
use work.bf.all;

entity Butterfly16x16 is
port(samplesIN, WI, WR: in stage;
      reset, start, clock: in std_logic;
      done: out std_logic;
      samplesOUT: out stage);
end entity Butterfly16x16;

architecture behavioral of Butterfly16x16 is

component Butterfly is
generic(N : positive := 20);
  port (
    RESET : IN STD_LOGIC;
    START : IN STD_LOGIC;
    CLK : IN STD_LOGIC;
    IN_A : IN SIGNED(N - 1 DOWNTO 0);
    IN_B : IN SIGNED(N - 1 DOWNTO 0);
    IN_WR : IN SIGNED(N - 1 DOWNTO 0);
    IN_WI : IN SIGNED(N - 1 DOWNTO 0);
    OUT_A : OUT SIGNED(N - 1 DOWNTO 0);
    OUT_B : OUT SIGNED(N - 1 DOWNTO 0);
    DONE : OUT STD_LOGIC
  );
end component;

type samples is array(0 to 4) of stage;
signal X: samples;
type middle is array (0 to 4) of std_logic_vector(0 to 7);
signal S_D: middle;
signal doneOut: std_logic_vector(0 to 7);
type indArr is array (0 to 7) of integer;
constant wIndex: indArr := (0,4,2,6,1,5,3,7);
begin

valoriIngresso: for i in 0 to 15 generate
begin
  X(0)(i) <= samplesIN(i);
end generate;
samplesOUT <= X(4);

S_D(0)(0) <= start;
doneOut(0) <= S_D(4)(0);

```

```

doneGen: for i in 1 to 7 generate
begin
    S_D(0)(i) <= start;
    doneOut(i) <= S_D(4)(i-1) and S_D(4)(i);
end generate;
--il segnale di done e' dato dall'AND di tutti i
--segnali di done nell'ultimo stadio
done <= doneOut(7);

firstLoop: for i in 0 to 3 generate

begin
    secondLoop: for j in 0 to (2**i)-1 generate
    begin
        thirdLoop: for k in 0 to (2**((3-i))-1 generate
        begin
            BF: Butterfly
            port map(
                start => S_D(i)( k +(j*(2**((3-i)))) ) ,
                reset => reset , clk => clock ,
                done => S_D(i+1)( k +(j*(2**((3-i)))) ) ,
                IN_A => X(i)(k +(j*(2**((4-i)))) ) ,
                IN_B => X(i)(k +(j*(2**((4-i)))) +(2**((3-i)))) ,
                IN_WR => WR(wIndex(j)) ,
                IN_WI => WI(wIndex(j)) ,
                OUT_A => X(i+1)(k +(j*(2**((4-i)))) ) ,
                OUT_B => X(i+1)(k +(j*(2**((4-i)))) +(2**((3-i)))) );
        end generate thirdLoop;
    end generate secondLoop;
end generate firstLoop;
end behavioral;

```

14.6 butterflyDatapath.vhd

```

library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity butterflyDatapath is
  generic (N : positive := 20);
  port (
    CLK : IN STD_LOGIC;

    — data ingresso
    DATA_IN_VAR_A : IN SIGNED (N - 1 downto 0);
    DATA_IN_VAR_B : IN SIGNED (N - 1 downto 0);
    DATA_IN_VAR_WR : IN SIGNED (N - 1 downto 0);
    DATA_IN_VAR_WI : IN SIGNED (N - 1 downto 0);

    — decoder per input A e B
    LOAD_REAL : IN STD_LOGIC;
    LOAD_IMM : IN STD_LOGIC;

    — segnali per W
    LOAD_W_REG : IN STD_LOGIC;

    CLEAR_REG : IN STD_LOGIC;

    — selezione bus
    SEL_BUS_MUL_1 : IN STD_LOGIC;
    SEL_BUS_MUL_2 : IN STD_LOGIC;
    SEL_BUS_SUM : IN STD_LOGIC;

    — segnali moltiplicatore
    LOAD_MUL_REG_PIPE : IN STD_LOGIC;
    SHIFT : IN STD_LOGIC;

    — somma pipe
    LOAD_SUM_REG_PIPE : IN STD_LOGIC;
    SUB_ADDN1 : IN STD_LOGIC;
    SUB_ADDN2 : IN STD_LOGIC;

    — load approx
    LOAD_APPROX: in std_logic;

    LOAD_REG_RES_SUM : IN STD_LOGIC;
    LOAD_REG_RES_MUL : IN STD_LOGIC;

    SEL_SUM_ADD : IN STD_LOGIC;

    — bus uscita
    OUT_A : OUT SIGNED(N - 1 downto 0);
    OUT_B : OUT SIGNED(N - 1 DOWNTO 0)
  );
end entity butterflyDatapath;
  
```

```

);
end butterflyDatapath;

architecture beh of butterflyDatapath is

component approx is
  generic(N : positive);
  port(
    Input: in signed((2*N)-1 downto 0);--ingresso su n+3 bit
    Output: out signed(N-1 downto 0));
end component;

component moltiplicatore is
  generic (N : positive);
  port (
    MUL1 : IN SIGNED (N - 1 downto 0);
    MUL2 : IN SIGNED (N - 1 downto 0);
    CLK : IN STD_LOGIC;
    CLEAR : IN STD_LOGIC;
    LOAD : IN STD_LOGIC;
    SHIFT : IN STD_LOGIC';
    RES : OUT SIGNED (2 * N - 2 downto 0)
  );
end component;

component sommatore is
  generic (N : positive);
  port (
    ADD1 : IN SIGNED (N - 1 downto 0);
    ADD2 : IN SIGNED (N - 1 downto 0);
    CLK : IN STD_LOGIC;
    LD : IN STD_LOGIC;
    CLEAR : IN STD_LOGIC;
    SUB : IN STD_LOGIC;
    RES : OUT SIGNED (N - 1 downto 0)
  );
end component;

component mux2to1 is
  generic (N : positive);
  port (
    DATA_1 : IN SIGNED (N - 1 downto 0);
    DATA_2 : IN SIGNED (N - 1 downto 0);
    SEL : IN STD_LOGIC;
    DATA_OUT : OUT SIGNED
  );

```

```

end component;

component RegisterFile is
  generic (N : positive);
  port (
    —CLK
    CLK : IN STD_LOGIC;

    — DATI INGRESSO REGISTER FILE
    DATA_IN_VAR_A : IN SIGNED (N - 1 downto 0);
    DATA_IN_VAR_B : IN SIGNED (N - 1 downto 0);
    DATA_IN_VAR_WR : IN SIGNED (N - 1 downto 0);
    DATA_IN_VAR_WI : IN SIGNED (N - 1 downto 0);
    — input controlli che sono gestiti da CU

    — decoder per input A e B
    LOAD_REAL_PART : IN STD_LOGIC;
    LOAD_IMM_PART : IN STD_LOGIC;

    — segnali per W
    LOAD_W_REG : IN STD_LOGIC;

    CLEAR_REG : IN STD_LOGIC;

    — selezione bus
    SEL_BUS_MUL_1 : IN STD_LOGIC;
    SEL_BUS_MUL_2 : IN STD_LOGIC;
    SEL_BUS_SUM : IN STD_LOGIC;

    — BUS di USCITA DATI
    BUS_MUL_1 : OUT SIGNED (N - 1 downto 0);
    BUS_MUL_2 : OUT SIGNED (N - 1 downto 0);
    BUS_ADD_1 : OUT SIGNED (N - 1 downto 0);
    BUS_ADD_2 : OUT SIGNED (N - 1 downto 0)
  );
end component;

component registerFF is
  generic (N : positive);
  port (
    DATA_IN : IN SIGNED(N - 1 downto 0);
    CLK : IN STD_LOGIC;
    CLEAR : IN STD_LOGIC;
    LOAD : IN STD_LOGIC;
    DATA_OUT : OUT SIGNED(N - 1 downto 0)
  );
end component;

signal BUS_MUL_1 : SIGNED (N - 1 downto 0);
signal BUS_MUL_2 : SIGNED (N - 1 downto 0);
signal BUS_ADD_1 : SIGNED (N - 1 downto 0);

```

```

signal BUS_ADD_2 : SIGNED (N - 1 downto 0);

signal RES_OUT_MUL : SIGNED(2 * N - 2 downto 0);
signal RES_MUL_MEM : SIGNED(2 * N - 2 downto 0);

signal ADD2_SUM1 : SIGNED(2 * N - 1 downto 0);
signal ADD2_SUM2 : SIGNED(2 * N - 1 downto 0);

signal ADD1_SUM1 : SIGNED(2 * N - 1 downto 0);
signal ADD1_SUM2 : SIGNED(2 * N - 1 downto 0);

signal A_SUM1 : SIGNED(2 * N - 1 downto 0);
signal A_SUM2 : SIGNED(2 * N - 1 downto 0);

signal RES_OUT_SUM1 : SIGNED(2 * N - 1 downto 0);
signal RES_SUM_MEM1 : SIGNED(2 * N - 1 downto 0);
signal RES_OUT_SUM2 : SIGNED(2 * N - 1 downto 0);
signal RES_SUM_MEM2 : SIGNED(2 * N - 1 downto 0);
signal DATA_A_TO_APPROX : SIGNED((2*N)-1 downto 0);
signal DATA_B_TO_APPROX : SIGNED((2*N)-1 downto 0);

signal OUT_A_TO_MEM : SIGNED(N - 1 downto 0);
signal OUT_B_TO_MEM : SIGNED(N - 1 DOWNTO 0);

```

```

begin
RF: RegisterFile
  generic map (N => N)
  port map(
    CLK => CLK,
    DATA_IN_VAR_A => DATA_IN_VAR_A,
    DATA_IN_VAR_B => DATA_IN_VAR_B,
    DATA_IN_VAR_WR => DATA_IN_VAR_WR,
    DATA_IN_VAR_WI => DATA_IN_VAR_WI,
    LOAD_REAL_PART => LOAD_REAL,
    LOAD_IMM_PART => LOAD_IMM,
    LOAD_W_REG => LOAD_W_REG,
    CLEAR_REG => CLEAR_REG,
    SEL_BUS_MUL_1 => SEL_BUS_MUL_1,
    SEL_BUS_MUL_2 => SEL_BUS_MUL_2,
    SEL_BUS_SUM => SEL_BUS_SUM,
    BUS_MUL_1 => BUS_MUL_1,
    BUS_MUL_2 => BUS_MUL_2,
    BUS_ADD_1 => BUS_ADD_1,
    BUS_ADD_2 => BUS_ADD_2
  );

```

MUL: moltiplicatore

```
generic map(N => N)
port map(
    MUL1 => BUS_MUL_1,
    MUL2 => BUS_MUL_2,
    CLK => CLK,
    CLEAR => CLEAR_REG,
    LOAD => LOAD_MUL_REG_PIPE,
    SHIFT => SHIFT,
    RES => RES_OUT_MUL
);
```

REG_OUT_MUL : registerFF

```
generic map(N => 2 * N - 1)
port map(
    DATA_IN => RES_OUT_MUL,
    CLK => CLK,
    CLEAR => CLEAR_REG,
    LOAD => LOAD_REG_RES_MUL,
    DATA_OUT => RES_MUL_MEM
);
```

```
ADD2_SUM1 <= RES_MUL_MEM(2 * N - 2) & RES_MUL_MEM;
ADD2_SUM2 <= RES_MUL_MEM(2 * N - 2) & RES_MUL_MEM;
```

```
A_SUM1 (2 * N - 1 downto N - 1) <= BUS_ADD_1(N - 1) & BUS_ADD_1;
```

```
A_SUM1 (N - 2 downto 0) <= (others => '0');
```

```
A_SUM2 (2 * N - 1 downto N - 1) <= BUS_ADD_2(N - 1) & BUS_ADD_2;
```

```
A_SUM2 (N - 2 downto 0) <= (others => '0');
```

MUX_ADD1_SUM1 : mux2to1

```
generic map(N => 2 * N)
port map(
    DATA_1 => A_SUM1,
    DATA_2 => RES_SUM_MEM1,
    SEL => SEL_SUM_ADD,
    DATA_OUT => ADD1_SUM1
);
```

MUX_ADD1_SUM2 : mux2to1

```
generic map(N => 2 * N)
port map(
    DATA_1 => A_SUM2,
    DATA_2 => RES_SUM_MEM2,
    SEL => SEL_SUM_ADD,
    DATA_OUT => ADD1_SUM2
);
```

SUM1 : sommatore

```
generic map(N => 2 * N)
```

```

port map(
    ADD1 => ADD1_SUM1,
    ADD2 => ADD2_SUM1,
    CLK => CLK,
    LD => LOAD_SUM_REG_PIPE,
    CLEAR => CLEAR_REG,
    SUB => SUB_ADDN1,
    RES => RES_OUT_SUM1
);

SUM2 : sommatore
generic map(N => 2 * N)
port map(
    ADD1 => ADD1_SUM2,
    ADD2 => ADD2_SUM2,
    CLK => CLK,
    LD => LOAD_SUM_REG_PIPE,
    CLEAR => CLEAR_REG,
    SUB => SUB_ADDN2,
    RES => RES_OUT_SUM2
);

REG_OUT_SUM1 : registerFF
generic map(N => 2 * N)
port map(
DATA_IN => RES_OUT_SUM1,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_REG_RES_SUM,
DATA_OUT => RES_SUM_MEM1
);

REG_OUT_SUM2 : registerFF
generic map(N => 2 * N)
port map(
DATA_IN => RES_OUT_SUM2,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_REG_RES_SUM,
DATA_OUT => RES_SUM_MEM2
);

DATA_A_TO_APPROX <= RES_SUM_MEM2;
DATA_B_TO_APPROX <= RES_SUM_MEM1;

APPROX_B: approx
generic map(N => N)
port map(
    Input => DATA_B_TO_APPROX,
    Output => OUT_B_TO_MEM
);

```

```
APPROX_A: approx
generic map(N => N)
port map(
    Input => DATA_A_TO_APPROX,
    Output => OUT_A_TO_MEM
);

REG_OUT_APPROX_A : registerFF
generic map(N => N)
port map(
DATA_IN => OUT_A_TO_MEM,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_APPROX,
DATA_OUT => OUT_A
);

REG_OUT_APPROX_B : registerFF
generic map(N => N)
port map(
DATA_IN => OUT_B_TO_MEM,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_APPROX,
DATA_OUT => OUT_B
);

end beh;
```

14.7 cu.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cu is
port(
    —segnale di reset proveniente dall'esterno.
    —Resetta la macchina in qualsiasi momento
    reset: in std_logic;

    —segnale di start dall'esterno
    start: in std_logic;

    —segnale di clock
    clock: in std_logic;

    —segnali da inviare ai blocchi del DP
    toDatapath: out std_logic_vector(15 downto 0);

    —segnale di done per indicare la fine dell'algoritmo
    done : out std_logic
);
end entity cu;

architecture beh of cu is

component gen_mux_2_1 is
generic(n: positive);
port(
    xs: in std_logic_vector(n-1 downto 0);
    ys: in std_logic_vector(n-1 downto 0);
    s1: in std_logic;
    ms: out std_logic_vector(n-1 downto 0));
end component;

component mux_2_1 is
port(
    x: in std_logic;
    y: in std_logic;
    s: in std_logic;
    m: out std_logic );
end component;

component uIR is
generic (N : positive);
port (
    data_in: in std_logic_vector (N-1 downto 0);
    CLK: in std_logic;
    load: in std_logic;

```

```

    clear: in std_logic;
    data_out: out std_logic_vector (N -1 downto 0)
);

end component;

component uAR is
generic (n: positive);
port (
    dataIn: in std_logic_vector(n - 1 downto 0);
    CLK: in std_logic;
    reset: in std_logic;
    enable: in std_logic;
    dataOut: out std_logic_vector(n - 1 downto 0)
);
end component;

component uROM is
port (
    add: in std_logic_vector(3 downto 0);
    evenOutput,oddOutput: out std_logic_vector(22 downto 0)
);
end component;

component late_status_pla is
port (
START: in std_logic;      —segnale di start proveniente dall'esterno
no_jmp: in std_logic;    —condition code
lsb: in std_logic;       —bit proveniente dal uIR
lsb_out: out std_logic   —uscita del blocco combinatorio
);
end component;

—uscita del late status pla
signal fromLateStatusPLA: std_logic;

—ingresso del uAR
signal touAR: std_logic_vector(4 downto 0);

—uscita del uAR
signal fromuAR: std_logic_vector(4 downto 0);

—uscita della uROM celle pari
signal evenCell: std_logic_vector(22 downto 0);

—uscita della uROM delle dispari
signal oddCell: std_logic_vector(22 downto 0);

—uscita del multiplexer, per selezionare l'ingresso da inviare al uIR
signal touIR: std_logic_vector(22 downto 0);

```

```

--uscita del uIR
signal fromuIR: std_logic_vector(22 downto 0);

begin

microAR: uAR generic map (n => 5)
port map (
dataIn => touAR,
CLK => clock,
reset => reset,
enable => '1',
dataOut => fromuAR);

pla: late_status_pla
port map(
START => start,
no_jmp => fromuIR (22),
lsb => fromuIR (0),
lsb_out => fromLateStatusPLA
);

uROMCU: uROM port map(
add => fromuAR(4 downto 1),
evenOutput => evenCell,
oddOutput => oddCell);

choiceMUX: gen_mux_2_1
generic map (n => 23)
port map (
xs => evenCell,
ys => oddCell,
s1 => fromuAR(0),
ms => touIR);

microIR: uIR generic map (n => 23)
port map (
data_in => touIR,
CLK => clock,
load => '1',
clear => '0',
data_out => fromuIR);

touAR <= fromuIR(4 downto 1) & fromLateStatusPLA;
toDatapath <= fromuIR(21 downto 6);
done <= fromuIR (5);

end beh;

```

14.8 FF.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity FF is
    port (data_in: in std_logic;
          CLK, reset, enable: in std_logic;
          data_out: out std_logic);
end entity FF;

architecture beh of FF is

begin

process(CLK)
begin

    if(CLK'event and CLK = '1') then
        if (reset = '0') then
            data_out <= '0';
        elsif enable = '1' then
            data_out <= data_in;
        end if;
    end if;
end process;

end architecture beh;
```

14.9 gen_mux_2_1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gen_mux_2_1 is
  generic(n: positive);
  port(xs,ys: in std_logic_vector(n-1 downto 0);
       sl: in std_logic;
       ms: out std_logic_vector(n-1 downto 0));
end gen_mux_2_1;

architecture beh of gen_mux_2_1 is

  component mux_2_1 is
  port(x,y,s: in std_logic;
       m: out std_logic );
  end component;

begin

a: for i in 0 to n-1 generate
  muxes: mux_2_1 port map (x => xs(i), y => ys(i), s => sl, m => ms(i));
end generate;

end beh;
```

14.10 late_status_pla.vhd

```

library ieee;
use ieee.std_logic_1164.all;

entity late_status_pla is

port(
START: in std_logic;           --segnale di start proveniente dall'esterno
no_jmp: in std_logic;          --condition code
lsb: in std_logic;             --bit proveniente dal uIR
lsb_out: out std_logic;         --uscita del blocco combinatorio
);
end late_status_pla;

architecture beh of late_status_pla is

component mux_2_1 is
port(
    x: in std_logic;
    y: in std_logic;
    s: in std_logic;
    m: out std_logic );
end component;

signal jmp:std_logic;

begin

jmp <= start;
jmpMUX: mux_2_1 port map (
    x => jmp,
    y => lsb ,
    s => no_jmp,
    m => lsb_out );

end beh;

```

14.11 moltiplicatore.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity moltiplicatore is
    generic (N : positive);
    port (
        MUL1 : IN SIGNED (N - 1 downto 0);
        MUL2 : IN SIGNED (N - 1 downto 0);
        CLK : IN STD_LOGIC;
        CLEAR : IN STD_LOGIC;
        LOAD : IN STD_LOGIC;
        SHIFT : IN STD_LOGIC;
        RES : OUT SIGNED (2 * N - 2 downto 0)
    );
end moltiplicatore;

architecture beh of moltiplicatore is

component registerFF
    generic (N : positive);
    port (
        DATA_IN : IN SIGNED(N - 1 downto 0);
        CLK : IN STD_LOGIC;
        CLEAR : IN STD_LOGIC;
        LOAD : IN STD_LOGIC;
        DATA_OUT : OUT SIGNED(N - 1 downto 0)
    );
end component;

component mux2to1
    generic (N : positive);
    port (
        DATA_1 : IN SIGNED (N - 1 downto 0);
        DATA_2 : IN SIGNED (N - 1 downto 0);
        SEL : IN STD_LOGIC;
        DATA_OUT : OUT SIGNED (N - 1 downto 0)
    );
end component;

begin
    RES_MUL <= MUL1 * MUL2;
    RES_TO_MEM_1 <= RES_MUL(2 * N - 2 downto 0);

    RES_TO_MEM_2(2 * N - 2 downto N) <= MUL1(N - 2 downto 0);
    RES_TO_MEM_2(N - 1 downto 0) <= (others => '0');

```

```
MUX_MUL_SHIFT : mux2to1
generic map (N => 2 * N - 1)
port map(
    DATA_1 => RES_TO_MEM_1,
    DATA_2 => RES_TO_MEM_2,
    SEL => SHIFT,
    DATA_OUT => RES_TO_MEM
);

REG_PIPE : registerFF
generic map(N => 2 * N - 1)
port map (
    DATA_IN => RES_TO_MEM,
    CLK => CLK,
    CLEAR => CLEAR,
    LOAD => LOAD,
    DATA_OUT => RES
);
end beh;
```

14.12 mux2to1.vhd

```

library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity mux2to1 is
    generic (N : positive);
    port (
        DATA_1 : IN SIGNED (N - 1 downto 0);
        DATA_2 : IN SIGNED (N - 1 downto 0);
        SEL : IN STD_LOGIC;
        DATA_OUT : OUT SIGNED
    );
end mux2to1;

architecture beh of mux2to1 is
begin
    with SEL select DATA_OUT <=
        DATA_1 when '0',
        DATA_2 when others;
end beh;

```

14.13 mux_2_1.vhd

```

library ieee;
use ieee.std_logic_1164.all;

entity mux_2_1 is
    port(x,y,s: in std_logic;
         m: out std_logic );
end mux_2_1;

architecture beh of mux_2_1 is
begin
    m<= ((x and (not (s))) or (y and s));
end beh;

```

14.14 n_bit_register.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity n_bit_register is
    generic (n_bit: integer);
    port (data_in: in std_logic_vector(n_bit - 1 downto 0);
          CLK, reset, enable: in std_logic;
          data_out: out std_logic_vector(n_bit - 1 downto 0));
end entity n_bit_register;

architecture beh of n_bit_register is
begin

process(CLK)
begin

if(CLK'event and CLK = '1') then

    if (reset = '1') then

        data_out <= (others => '0');

    elsif enable = '1' then

        data_out <= data_in;

    end if;
end if;
end process;

end architecture beh;

```

14.15 packageBF.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
package bf is
    type stage is array (0 to 15) of signed(19 downto 0);
end bf;
```

14.16 registerFF.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity registerFF is
    generic (N : positive);
    port (
        DATA_IN : IN SIGNED(N - 1 downto 0);
        CLK : IN STD_LOGIC;
        CLEAR : IN STD_LOGIC;
        LOAD : IN STD_LOGIC;
        DATA_OUT : OUT SIGNED(N - 1 downto 0)
    );
end registerFF;

architecture beh of registerFF is
begin
process(CLK)
begin
if CLK'EVENT AND CLK = '1' then
    if CLEAR = '1' then
        DATA_OUT <= (OTHERS => '0');
    else
        if LOAD = '1' then
            DATA_OUT <= DATA_IN;
        end if;
    end if;
end if;
end process;
end beh;
```

14.17 RegisterFile.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity RegisterFile is
    generic (N : positive);
    port (
        —CLK
        CLK : IN STD_LOGIC;

        — DATI INGRESSO REGISTER FILE
        DATA_IN_VAR_A : IN SIGNED (N - 1 downto 0);
        DATA_IN_VAR_B : IN SIGNED (N - 1 downto 0);
        DATA_IN_VAR_WR : IN SIGNED (N - 1 downto 0);
        DATA_IN_VAR_WI : IN SIGNED (N - 1 downto 0);
        — input controlli che sono gestiti da CU

        — decoder per input A e B
        LOAD_REAL_PART : IN STD_LOGIC;
        LOAD_IMM_PART : IN STD_LOGIC;

        — segnali per W
        LOAD_W_REG : IN STD_LOGIC;

        CLEAR_REG : IN STD_LOGIC;

        — selezione bus
        SEL_BUS_MUL_1 : IN STD_LOGIC;
        SEL_BUS_MUL_2 : IN STD_LOGIC;
        SEL_BUS_SUM : IN STD_LOGIC;

        — BUS di USCITA DATI
        BUS_MUL_1 : OUT SIGNED (N - 1 downto 0);
        BUS_MUL_2 : OUT SIGNED (N - 1 downto 0);
        BUS_ADD_1 : OUT SIGNED (N - 1 downto 0);
        BUS_ADD_2 : OUT SIGNED (N - 1 downto 0)
    );
end RegisterFile;

architecture beh of RegisterFile is

component registerFF
    generic (N : positive);
    port (
        DATA_IN : IN SIGNED(N - 1 downto 0);
        CLK : IN STD_LOGIC;
        CLEAR : IN STD_LOGIC;
        LOAD : IN STD_LOGIC;
        DATA_OUT : OUT SIGNED(N - 1 downto 0)
    );
end component;

```

```

);
end component;

component mux2to1
generic (N : positive);
port (
    DATA_1 : IN SIGNED (N - 1 downto 0);
    DATA_2 : IN SIGNED (N - 1 downto 0);
    SEL : IN STD_LOGIC;
    DATA_OUT : OUT SIGNED
);
end component;

signal AR_MEM : SIGNED(N - 1 downto 0);
signal BR_MEM : SIGNED(N - 1 downto 0);
signal AI_MEM : SIGNED(N - 1 downto 0);
signal BI_MEM : SIGNED(N - 1 downto 0);

signal WR_MEM : SIGNED(N - 1 downto 0);
signal WI_MEM : SIGNED(N - 1 downto 0);

signal BUS_ADD : SIGNED (N - 1 downto 0);

begin

REG_BR : registerFF
generic map (N => N)
port map(
DATA_IN => DATA_IN_VAR_B,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_REAL_PART,
DATA_OUT => BR_MEM
);

REG_BI : registerFF
generic map (N => N)
port map(
DATA_IN => DATA_IN_VAR_B,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_IMM_PART,
DATA_OUT => BI_MEM
);

REG_AR : registerFF
generic map (N => N)
port map(
DATA_IN => DATA_IN_VAR_A,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_REAL_PART,

```

```

    DATA_OUT => AR_MEM
);

REG_AI : registerFF
generic map (N => N)
port map(
DATA_IN => DATA_IN_VAR_A,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_IMM_PART,
DATA_OUT => AI_MEM
);

```



```

MUX_BUS_ADD : mux2to1
generic map(N => N)
port map (
    DATA_1 => AR_MEM,
    DATA_2 => AI_MEM,
    SEL => SEL_BUS_SUM,
    DATA_OUT => BUS_ADD
);

```



```

BUS_ADD_1 <= BUS_ADD;
BUS_ADD_2 <= BUS_ADD;

```



```

MUX_BUS_MUL_1 : mux2to1
generic map(N => N)
port map(
    DATA_1 => BR_MEM,
    DATA_2 => BI_MEM,
    SEL => SEL_BUS_MUL_1,
    DATA_OUT => BUS_MUL_1
);

```



```

REG_WR : registerFF
generic map(N => N)
port map(
DATA_IN => DATA_IN_VAR_WR,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_W_REG,
DATA_OUT => WR_MEM
);

```



```

REG_WI : registerFF
generic map(N => N)
port map(
DATA_IN => DATA_IN_VAR_WI,
CLK => CLK,
CLEAR => CLEAR_REG,
LOAD => LOAD_W_REG,

```

```
    DATA_OUT => WI_MEM  
);  
  
MUX_BUS_DOWN : mux2to1  
generic map (N => N)  
port map (  
    DATA_1 => WR_MEM,  
    DATA_2 => WI_MEM,  
    SEL => SEL_BUS_MUL_2,  
    DATA_OUT => BUS_MUL_2  
);  
end beh ;
```

14.18 sommatore.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sommatore is
    generic (N : positive);
    port (
        ADD1 : IN SIGNED (N - 1 downto 0);
        ADD2 : IN SIGNED (N - 1 downto 0);
        CLK : IN STD_LOGIC;
        LD : IN STD_LOGIC;
        CLEAR : IN STD_LOGIC;
        SUB : IN STD_LOGIC;
        RES : OUT SIGNED (N - 1 downto 0)
    );
end sommatore;

architecture beh of sommatore is

component registerFF
    generic (N : positive);
    port (
        DATA_IN : IN SIGNED(N - 1 downto 0);
        CLK : IN STD_LOGIC;
        CLEAR : IN STD_LOGIC;
        LOAD : IN STD_LOGIC;
        DATA_OUT : OUT SIGNED(N - 1 downto 0)
    );
end component;

signal RES_TO_MEM : SIGNED(N - 1 downto 0);

begin

ADD_SUB_PROCESS : process( ADD1, ADD2, SUB )
begin
    if SUB = '0' then
        RES_TO_MEM <= ADD1 + ADD2;
    else
        RES_TO_MEM <= ADD1 - ADD2;
    end if;
end process ; — ADD_SUB_PROCESS

REG_PIPE : registerFF
    generic map(N => N)
    port map (
        DATA_IN => RES_TO_MEM,

```

```
CLK => CLK,  
CLEAR => CLEAR,  
LOAD => LD,  
DATA_OUT => RES  
);  
  
end beh;
```

14.19 tbAdder.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tbAdder is
end entity tbAdder;

architecture behavioral of tbAdder is

component Adder is
    generic(N : positive);
    port(
        IN1, IN2: in signed(N-1 downto 0);
        RES: out signed(N-1 downto 0));
end component;

signal in1,in2,res: signed (19 downto 0);

begin

in1 <= to_signed(0,20),to_signed(-20,20) after 20 ns,
to_signed(58,20) after 50 ns;
in2 <= to_signed(0,20), to_signed(15,20) after 15 ns,
to_signed(-66,20) after 40 ns;

DUT: Adder generic map (N => 20)
      port map(IN1 => in1, IN2 => in2, RES => res);

end behavioral;

```

14.20 tbButterfly16x16.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;

library work;
use work.bf.all;

entity tbButterfly16x16 is
end entity tbButterfly16x16;

architecture behavioral of tbButterfly16x16 is

component Butterfly16x16 is
port(samplesIN , WR,WI:  in stage;
      reset ,start , clock: in std_logic;
      done: out std_logic;
      samplesOUT: out stage);
end component;

file testFile , resFile: text;

signal clock , start , reset , done: std_logic;
signal dI , dO,dM, WR,WI: stage;
begin

reset <= '1' , '0' after 40 ns;

clock_gen: process
begin
  clock <= '1';
  wait for 10 ns;
  clock <= '0';
  wait for 10 ns;
end process clock_gen;

readInput: process
  variable testLine: line;
  variable testData: std_logic_vector(19 downto 0);

  variable space: character;
  variable index: integer;

begin
  start <= '0';
  —carico i W durante la fase di reset
  file_open(testFile , "butterflyW.txt" ,read_mode);
  index := 0;

```

```

while not endfile(testFile) and index < 16 loop

    readline(testFile, testLine);

        read(testLine, testData);
        read(testLine, space);
        WR(index) <= signed(testData);
        read(testLine, testData);
        WI(index) <= signed(testData);
        index := index + 1;
end loop;
file_close(testFile);

wait for 50 ns;

file_open(testFile, "butterfly16x16InputData.txt", read_mode);
while not endfile(testFile) loop

    start <= '1';
    wait for 40 ns;
    readline(testFile, testLine);
    for i in 0 to 15 loop
        —invio i dati dal file al DUT
        read(testLine, testData);
        read(testLine, space);
        dI(i) <= signed(testData);
    end loop;

    —la parte immaginaria dei dati in ingresso e' zero
    start <= '0';
    wait for 20 ns;
    for i in 0 to 15 loop
        dI(i) <= to_signed(0,20) / 2;
    end loop;
    wait for 20 ns;
end loop;

file_close(testFile);

wait;

end process readInput;

file_open(resFile, "butterfly16x16TbResults.txt", write_mode);

writing: process

variable resLine: line;
begin
    wait for 20 ns;
—quando ricevo il segnale di done dal DUT
—salvo i dati in uscita su un file
    if done='1' then
        wait for 20 ns;

```

```

        for i in 0 to 15 loop
            write(resLine, to_integer(dO(i)*2));
            write(resLine, ',');
        end loop;
        writeline(resFile, resLine);

        wait for 20 ns;
        for i in 0 to 15 loop
            write(resLine, to_integer(dO(i)*2));
            write(resLine, ',');
        end loop;
        writeline(resFile, resLine);
    end if;
end process writing;
file_close(resFile);
--end process saveOutput;

```

DUT: Butterfly16x16

```

port map(
    samplesIn => dI, WR => WR, WI => WI,
    reset => reset, start => start, clock => clock,
    done => done, samplesOut => dM);
dO(0) <= dM(0);
dO(1) <= dM(8);
dO(2) <= dM(4);
dO(3) <= dM(12);
dO(4) <= dM(2);
dO(5) <= dM(10);
dO(6) <= dM(6);
dO(7) <= dM(14);
dO(8) <= dM(1);
dO(9) <= dM(9);
dO(10) <= dM(5);
dO(11) <= dM(13);
dO(12) <= dM(3);
dO(13) <= dM(11);
dO(14) <= dM(7);
dO(15) <= dM(15);
end behavioral;

```

14.21 tb_approx.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_approx is
end entity tb_approx;

architecture behavioral of tb_approx is

component approx is
generic(N : positive);
port(
    Input: in signed((2*N)-1 downto 0);
    Output: out signed(N-1 downto 0));
end component approx;

constant N : positive := 20;
signal Input: signed((2*N)-1 downto 0);
signal Output: signed(N-1 downto 0);

begin

DUT: approx
generic map(N => N)
port map (
Input => Input,
Output => Output
);

S : process
begin
Input <= "000000000000000000001100000000000000000000";
wait for 10 ns;
Input <= "00000000000000000000100100000000000000000";
wait for 10 ns;
Input <= "00000000000000000000101100000000000000000";
wait for 10 ns;
Input <= "00000000000000000000100000000000000000000";
wait for 10 ns;
Input <= "00000000000000000000111100000000000000000000";
wait;
end process;
end behavioral;

```

14.22 tb_Butterfly.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;

entity tb_Butterfly is
end tb_Butterfly;

architecture beh of tb_Butterfly is
constant N : positive := 20;

component Butterfly is
generic(N : positive := 20);
port (
    RESET : IN STD_LOGIC;
    START : IN STD_LOGIC;
    CLK : IN STD_LOGIC;
    IN_A : IN SIGNED(N - 1 DOWNTO 0);
    IN_B : IN SIGNED(N - 1 DOWNTO 0);
    IN_WR : IN SIGNED(N - 1 DOWNTO 0);
    IN_WI : IN SIGNED(N - 1 DOWNTO 0);
    OUT_A : OUT SIGNED(N - 1 DOWNTO 0);
    OUT_B : OUT SIGNED(N - 1 DOWNTO 0);
    DONE : OUT STD_LOGIC
);
end component;

signal RESET : STD_LOGIC;
signal START : STD_LOGIC;
signal CLK : STD_LOGIC;
signal IN_A : SIGNED(N - 1 DOWNTO 0);
signal IN_B : SIGNED(N - 1 DOWNTO 0);
signal IN_WR : SIGNED(N - 1 DOWNTO 0);
signal IN_WI : SIGNED(N - 1 DOWNTO 0);
signal OUT_A : SIGNED(N - 1 DOWNTO 0);
signal OUT_B : SIGNED(N - 1 DOWNTO 0);
signal DONE : STD_LOGIC;

file testFileR ,testFileI ,wFile , resFile: text;

begin
UUT: Butterfly
generic map(N => N)
port map(
    RESET => RESET,
    START => START,
    CLK => CLK,
    IN_A => IN_A,
    IN_B => IN_B,
    IN_WR => IN_WR,
    IN_WI => IN_WI,
    OUT_A => OUT_A,
    OUT_B => OUT_B,

```

```

        DONE => DONE
    );

readInput: process
    variable testLine ,testLineR ,testLineI: line;
    variable testData ,testDataAI ,testDataBI: std_logic_vector(19 downto 0);
    variable space: character;

begin
    —scorro il file coi W
    file_open(wFile , "butterflyW.txt" ,read_mode);
    —wait for 60 ns;
    RESET <= '1';
    start <= '0';
    wait for 40 ns;
    RESET <= '0';

    wait for 40 ns;
    while not endfile(wFile) loop
        readline(wFile , testLine);

        read(testLine , testData);
        read(testLine , space);
        IN_WR <= signed(testData);
        read(testLine , testData);
        read(testLine , space);
        IN WI <= signed(testData);

    wait for 50 ns;
    —scorro il file con la parte immaginaria dei dati
    file_open(testFileI , "butterflyInputImagData.txt" ,read_mode);
    while not endfile(testFileI) loop
        readline(testFileI , testLineI);

        read(testLineI , testDataAI);
        read(testLineI , space);

        read(testLineI , testDataBI);
        read(testLineI , space);
    —scorro il file con le parti reali dei dati
    file_open(testFileR , "butterflyInputRealData.txt" , read_mode);
    while not endfile(testFileR) loop
        start <= '1';
        wait for 40 ns;

        readline(testFileR , testLineR);

        read(testLineR , testData);

```

```

        read(testLineR, space);
        IN_A <= signed(testData);

        read(testLineR, testData);
        read(testLineR, space);
        IN_B <= signed(testData);

        start <= '0';
        wait for 20 ns;

        IN_A <= signed(testDataAI);
        IN_B <= signed(testDataBI);

        wait for 20 ns;
    end loop;

    file_close(testFileR);

end loop;
file_close(testFileI);

end loop;
file_close(wFile);

wait;

end process readInput;

file_open(resFile, "butterflyTbResults.txt", write_mode);

writing: process
variable resLine: line;
begin
    wait for 20 ns;
—quando ricevo dal DUT il segnale di done —salvo i dati in uscita
    if done='1' then
        wait for 20 ns;

        write(resLine, to_integer(OUT_A));
        write(resLine, ',');
        write(resLine, to_integer(OUT_B));
        write(resLine, ',');
        writeline(resFile, resLine);

        wait for 20 ns;
        write(resLine, to_integer(OUT_A));
        write(resLine, ',');
        write(resLine, to_integer(OUT_B));
        write(resLine, ',');

```

```
        writeline( resFile , resLine );
    end if;
end process writing;
file_close(resFile);

CLK_GEN : process
begin
    CLK <= '0';
    wait for 10 ns;
    CLK <= '1';
    wait for 10 ns;
end process;

end beh ;
```

14.23 TB_CU_BUTTERFLY.vhd

```

library ieee;
use ieee.std_logic_1164.all;

entity TB_CU_BUTTERFLY is
end entity TB_CU_BUTTERFLY;

architecture beh of TB_CU_BUTTERFLY is

component cu is
port(
    reset , start , clock: in std_logic;
    toDatapath: out std_logic_vector(15 downto 0);
    done : out std_logic
);
end component;

signal reset , start , clock , done: std_logic;
signal toDatapath: std_logic_vector(15 downto 0);

begin

reset <= '1' , '0' after 40 ns;
start <= '0' , '1' after 50 ns;

clockGen: process
begin
    clock <= '0';
    wait for 10 ns;
    clock <= '1';
    wait for 10 ns;

end process clockGen;

DUT: cu port map (
    reset => reset ,
    start => start ,
    clock => clock ,
    toDatapath => toDatapath ,
    done => done );

end beh;

```

14.24 tb_decoder.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_decoder is
end tb_decoder;

architecture beh of tb_decoder is
constant N : positive := 3;
component decoder is
generic (N : positive);
port (
    ADD : IN UNSIGNED (N - 1 downto 0);
    EN : IN STD_LOGIC;
    DATA_OUT : OUT STD_LOGIC_VECTOR ((2 ** N) - 1 downto 0)
);
end component;

signal EN : STD_LOGIC;
signal ADD : UNSIGNED(N - 1 downto 0);
signal DATA_OUT : STD_LOGIC_VECTOR ((2 ** N) - 1 downto 0);

begin
UUT : decoder
generic map(N => N)
port map(
    ADD => ADD,
    EN => EN,
    DATA_OUT => DATA_OUT
);

SIG_GEN : process
begin
    EN <= '0';
    ADD <= (others => '0');
    wait for 10 ns;
    EN <= '1';
    for i in 0 to 2 ** N - 1 loop
        ADD <= to_unsigned(i, N);
        wait for 10 ns;
    end loop;
    EN <= '0';
    wait for 10 ns;
    EN <= '1';
    wait;
end process;

end beh;

```

14.25 tb_moltiplicatore.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_moltiplicatore is
end tb_moltiplicatore;

architecture beh of tb_moltiplicatore is
constant N : positive := 8;
component moltiplicatore
generic (N : positive);
port (
    MUL1 : IN SIGNED (N - 1 downto 0);
    MUL2 : IN SIGNED (N - 1 downto 0);
    CLK : IN STD_LOGIC;
    CLEAR : IN STD_LOGIC;
    LOAD : IN STD_LOGIC;
    SHIFT: IN STD_LOGIC;
    RES : OUT SIGNED (2* N - 2 downto 0)
);
end component;

signal MUL1 : SIGNED (N - 1 downto 0);
signal MUL2 : SIGNED (N - 1 downto 0);
signal CLK : STD_LOGIC;
signal CLEAR : STD_LOGIC;
signal LOAD : STD_LOGIC;
signal RES : SIGNED (2 * N - 2 downto 0);
signal SHIFT : STD_LOGIC;

begin
UUT : moltiplicatore
generic map (N => N)
port map (
    MUL1 => MUL1,
    MUL2 => MUL2,
    CLK => CLK,
    CLEAR => CLEAR,
    LOAD => LOAD,
    SHIFT => SHIFT,
    RES => RES
);

CLK_GEN : process
begin
    CLK <= '1';
    wait for 10 ns;

```

```

CLK <= '0';
wait for 10 ns;
end process ;

SIGNAL_GEN : process
begin
CLEAR <= '1';
LOAD <= '1';
SHIFT <= '0';
MUL1 <= (others => '0');
MUL2 <= (others => '0');
wait for 10 ns;
CLEAR <= '0';
for i in -(2 ** (N - 2)) to (2 ** (N - 2) - 1) loop
  for j in -(2 ** (N - 2) - 1) to (2 ** (N - 2) - 1) loop
    MUL1 <= to_signed(i, N);
    MUL2 <= to_signed(j, N);
    wait for 20 ns;
  end loop;
end loop;
MUL1 <= to_signed(3, N);
MUL2 <= to_signed(5, N);
SHIFT <= '1';
wait for 20 ns;
LOAD <= '0';
MUL1 <= to_signed(7, N);
wait for 20 ns;
CLEAR <= '1';
wait for 20 ns;
LOAD <= '1';
CLEAR <= '0';
wait;
end process;
end beh;

```

14.26 tb_RegisterFile.vhd

```

library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity tb_RegisterFile is
end tb_RegisterFile;

architecture beh of tb_RegisterFile is

constant N : positive := 5;

component RegisterFile
  generic (N : positive);
  port (

```

```

—CLK
CLK : IN STD_LOGIC;

— DATI INGRESSO REGISTER FILE
DATA_IN_VAR_A : IN SIGNED (N - 1 downto 0);
DATA_IN_VAR_B : IN SIGNED (N - 1 downto 0);
DATA_IN_VAR_WR : IN SIGNED (N - 1 downto 0);
DATA_IN_VAR_WI : IN SIGNED (N - 1 downto 0);
— input controlli che sono gestiti da CU

— decoder per input A e B
LOAD_REAL_PART : IN STD_LOGIC;
LOAD_IMM_PART : IN STD_LOGIC;

— segnali per W
LOAD_W_REG : IN STD_LOGIC;

CLEAR_REG : IN STD_LOGIC;

— selezione bus
SEL_BUS_MUL_1 : IN STD_LOGIC;
SEL_BUS_MUL_2 : IN STD_LOGIC;
SEL_BUS_SUM : IN STD_LOGIC;

— BUS di USCITA DATI
BUS_MUL_1 : OUT SIGNED (N - 1 downto 0);
BUS_MUL_2 : OUT SIGNED (N - 1 downto 0);
BUS_ADD_1 : OUT SIGNED (N - 1 downto 0);
BUS_ADD_2 : OUT SIGNED (N - 1 downto 0)
);
end component;

signal CLK : STD_LOGIC;

— DATI INGRESSO REGISTER FILE
signal DATA_IN_VAR_A : SIGNED (N - 1 downto 0);
signal DATA_IN_VAR_B : SIGNED (N - 1 downto 0);
signal DATA_IN_VAR_WR : SIGNED (N - 1 downto 0);
signal DATA_IN_VAR_WI : SIGNED (N - 1 downto 0);

— decoder per input A e B
signal LOAD_REAL_PART : STD_LOGIC;
signal LOAD_IMM_PART : STD_LOGIC;

— segnali per W
signal LOAD_W_REG : STD_LOGIC;

signal CLEAR_REG : STD_LOGIC;

— selezione bus
signal SEL_BUS_MUL_1 : STD_LOGIC;

```

```

signal SEL_BUS_MUL_2 : STD_LOGIC;
signal SEL_BUS_SUM : STD_LOGIC;
— BUS di USCITA DATI
signal BUS_MUL_1 : SIGNED (N - 1 downto 0);
signal BUS_MUL_2 : SIGNED (N - 1 downto 0);
signal BUS_ADD_1 : SIGNED (N - 1 downto 0);
signal BUS_ADD_2 : SIGNED (N - 1 downto 0);

```

begin

```

UUT: RegisterFile
generic map (N => N)
port map (
    CLK => CLK,
    DATA_IN_VAR_A => DATA_IN_VAR_A,
    DATA_IN_VAR_B => DATA_IN_VAR_B,
    DATA_IN_VAR_WR => DATA_IN_VAR_WR,
    DATA_IN_VAR_WI => DATA_IN_VAR_WI,
    LOAD_REAL_PART => LOAD_REAL_PART,
    LOAD_IMM_PART => LOAD_IMM_PART,
    LOAD_W_REG => LOAD_W_REG,
    CLEAR_REG => CLEAR_REG,
    SEL_BUS_MUL_1 => SEL_BUS_MUL_1,
    SEL_BUS_MUL_2 => SEL_BUS_MUL_2,
    SEL_BUS_SUM => SEL_BUS_SUM,
    BUS_MUL_1 => BUS_MUL_1,
    BUS_MUL_2 => BUS_MUL_2,
    BUS_ADD_1 => BUS_ADD_1,
    BUS_ADD_2 => BUS_ADD_2
);

```

```

CLK_GEN : process
begin
    CLK <= '1';
    wait for 10 ns;
    CLK <= '0';
    wait for 10 ns;
end process;

```

```

SIGNAL_GEN : process
begin

    LOAD_W_REG <= '0';
    LOAD_REAL_PART <= '0';
    LOAD_IMM_PART <= '0';
    DATA_IN_VAR_WR <= to_signed(-7, N);
    DATA_IN_VAR_WI <= to_signed(-12, N);
    DATA_IN_VAR_A <= to_signed(6, N);
    DATA_IN_VAR_B <= to_signed(3, N);

```

```

CLEAR_REG <= '1';
SEL_BUS_MUL_1 <= '0';
SEL_BUS_MUL_2 <= '0';
SEL_BUS_SUM <= '0';

wait for 20 ns;

CLEAR_REG <= '0';
LOAD_W_REG <= '1';
wait for 20 ns;

LOAD_REAL_PART <= '1';
wait for 20 ns;

DATA_IN_VAR_A <= to_signed(8, N);
DATA_IN_VAR_B <= to_signed(4, N);
LOAD_IMM_PART <= '1';
LOAD_REAL_PART <= '0';
wait for 20 ns;
DATA_IN_VAR_A <= to_signed(8, N);
DATA_IN_VAR_B <= to_signed(4, N);
LOAD_IMM_PART <= '0';
SEL_BUS_MUL_1 <= '1';
SEL_BUS_MUL_2 <= '1';
SEL_BUS_SUM <= '1';

wait for 20 ns;
SEL_BUS_MUL_1 <= '0';
SEL_BUS_MUL_2 <= '0';
SEL_BUS_SUM <= '0';
wait;

end process ; -- identifier
end beh ; -- beh

```

14.27 tb_sommatore.vhd

```

library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity tb_sommatore is
end tb_sommatore;

architecture beh of tb_sommatore is
constant N : positive := 5;
component sommatore is
generic (N : positive);
port (
    ADD1 : IN SIGNED (N - 1 downto 0);
    ADD2 : IN SIGNED (N - 1 downto 0);
    CLK : IN STD_LOGIC;
    LD : IN STD_LOGIC;
    CLEAR : IN STD_LOGIC;
    SUB : IN STD_LOGIC;
    RES : OUT SIGNED (N - 1 downto 0)
);
end component;
signal ADD1 : SIGNED (N - 1 downto 0);
signal ADD2 : SIGNED (N - 1 downto 0);
signal CLK : STD_LOGIC;
signal LD : STD_LOGIC;
signal CLEAR : STD_LOGIC;
signal SUB : STD_LOGIC;
signal RES : SIGNED (N - 1 downto 0);

begin
UUT : sommatore
generic map(N => N)
port map(
    ADD1 => ADD1,
    ADD2 => ADD2,
    CLK => CLK,
    LD => LD,
    CLEAR => CLEAR,
    SUB => SUB,
    RES => RES
);
CLK_GEN : process
begin
CLK <= '1';
wait for 10 ns;
CLK <= '0';
wait for 10 ns;
end process;

```

```

SIG_GEN : process
begin
CLEAR <= '1';
LD <= '1';
ADD1 <= to_signed(- 2 ** (N - 2), N);
ADD2 <= to_signed(- 2 ** (N - 2), N);
SUB <= '0';
wait for 20 ns;
CLEAR <= '0';
for i in -(2 ** (N - 2)) to (2 ** (N - 2) - 1) loop
  for j in -(2 ** (N - 2)) to (2 ** (N - 2) - 1)loop
    ADD1 <= to_signed(i, N);
    ADD2 <= to_signed(j, N);
    wait for 20 ns;
    end loop;
  end loop;
SUB <= '1';
for i in -(2 ** (N - 2)) to (2 ** (N - 2) - 1) loop
  for j in -(2 ** (N - 2)) to (2 ** (N - 2) - 1)loop
    ADD1 <= to_signed(i, N);
    ADD2 <= to_signed(j, N);
    wait for 20 ns;
    end loop;
  end loop;
wait;
end process;

end beh ;

```

14.28 uAR.vhd

```

library ieee;
use ieee.std_logic_1164.all;

entity uAR is
generic (n: positive);
port (
    —ingresso del uAR
    dataIn: in std_logic_vector(n - 1 downto 0);

    —segnale di clock
    CLK: in std_logic;

    —segnale di reset, se reset='1' la CU viene resettata
    reset: in std_logic;

    —segnale si enable, posto sempre a '1'
    enable: in std_logic;

    —uscita
    dataOut: out std_logic_vector(n - 1 downto 0)
);

end entity uAR;

architecture beh of uAR is

begin

process(CLK)
begin
    —registro campionato sul fronte di discesa

    if(CLK'event and CLK = '0') then
        if (reset = '1') then
            dataOut <= (others => '0');

        elsif enable = '1' then
            dataOut <= dataIn;

        end if;
    end if;
end process;

end architecture beh;

```

14.29 uIR.vhd

```

library ieee;
use ieee.std_logic_1164.all;

entity uIR is

generic (N : positive);
port (
    —ingresso del uIR
    data_in: in std_logic_vector (N-1 downto 0);

    —segnale di clock
    CLK: in std_logic;

    —segnale di load posto sempre a '1'
    load: in std_logic;

    —segnale di clear posto sempre a '0'
    clear: in std_logic;

    —uscita
    data_out: out std_logic_vector (N -1 downto 0)
);
end entity uIR;

architecture beh of uIR is

begin

process(CLK)
begin
    —registro campionato sul fronte di salita
    if(CLK'event and CLK = '1') then
        if (clear = '1') then
            data_out <= (others =>'0');
        elsif load = '1' then
            data_out <= data_in;
        end if;
    end if;
end process;

end architecture beh;

```

14.30 uROM.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity uROM is
port (
    —indirizzo su 4 bit
    add: in std_logic_vector(3 downto 0);

    —colonna pari e colonna dispari della uROM
    evenOutput,oddOutput: out std_logic_vector(22 downto 0));
end entity uROM;

architecture beh of uROM is

signal regAddress: integer range 0 to (2**4)-1;

type memory is array (0 to (2**4)-1) of std_logic_vector(22 downto 0);

—primo bit: CC => NO JMP
—16 bit: comandi to_datapath
— 1 BIT: DONE
—4 bit: next address
—ultimo bit: LSB

constant evenRom: memory :=(
0 => "1" & "1000000000000000" & "0" & "0000" & "1" ,
1 => "1" & "0001100000000000" & "0" & "0001" & "1" ,
2 => "0" & "0000111001100100" & "0" & "1110" & "1" ,
3 => "1" & "0000001100001100" & "0" & "0011" & "1" ,
4 => "1" & "0000000010000000" & "0" & "0000" & "1" ,
5 => "1" & "0001111100001010" & "0" & "0101" & "1" ,
6 => "0" & "000011111100100" & "1" & "1101" & "1" ,

13 => "1" & "000011111010100" & "0" & "0010" & "1" ,
14 => "1" & "0000111101010100" & "0" & "0010" & "1" ,
15 => "1" & "0100000000000000" & "0" & "0000" & "1" ,

others => "1" & "1000000000000000" & "0" & "0000" & "0" );

constant oddRom: memory :=(
0 => "0" & "0100000000000000" & "0" & "1111" & "0" ,
1 => "1" & "0000110000100000" & "0" & "0010" & "0" ,
2 => "1" & "0000011100001010" & "0" & "0011" & "0" ,
3 => "1" & "0000000110000000" & "1" & "0100" & "0" ,
5 => "1" & "0000111100101100" & "0" & "0110" & "0" ,

13 => "1" & "001011111010100" & "0" & "0101" & "0" ,
14 => "1" & "0010111101010100" & "0" & "0101" & "0" ,

```

```
15 => "1" & "0010000000000000" & "0" & "0001" & "0" ,  
others => "1" & "1000000000000000" & "0" & "0000" & "0");
```

begin

```
evenOutput <= evenRom( to_integer( unsigned( add )));  
oddOutput <= oddRom( to_integer( unsigned( add )));
```

end beh;

14.31 butterfly16x16Simulation.py

```

#!/usr/bin/env python
import os
import subprocess
from inputGen import generateInputs
from inputGen import generateW

#from dataConversion import convertToInt
from dataConversion import convertTo2C

from computeFFT import computeFFT

wFloat = "wFloat.txt"
#da commentare se si usano gli esempi del professore
inputFloat = "butterfly16x16FloatData.txt"
#da commentare se si usano gli input casuale
#inputFloat = "esempiZamboni.txt"

w2C = "butterflyW.txt"
input2C = "butterfly16x16InputData.txt"

resTb = "butterfly16x16TbResults.txt"
resTbFloat = "butterfly16x16fftFloatResults.txt"
log = "log16x16.txt"
resFloat = "butterfly16x16tbFloatResults.txt"

#generazione dei file di input

#questa riga serve per il test col metodo Montecarlo
#da commentare se si usano gli esempi del professore
generateInputs(inputFloat, "random")
generateW(wFloat)

#conversione dei valori da float a binario complemento a 2

convertTo2C(wFloat, w2C)
convertTo2C(inputFloat, input2C)

#compliazione dei listati vhdl ed avvio del testbench

# Setting up environement variables
os.environ["PATH"] += os.pathsep +
    "/software/mentor/modelsim_6.5c/modeltech/linux_x86_64/"
os.environ["LM_LICENSE_FILE"] = "1717@led-x3850-1.polito.it"

# Print out environement variables
os.system("echo $PATH")
os.system("echo $LM_LICENSE_FILE")

```

```
# Launch Modelsim simulation
print ("Starting simulation ... ")
process = subprocess.call(["vsim", "-c", "-do", "compile.do"])
print ("Simulation completed")

#calcolo dei risultati attesi ed analisi degli errori

computeFFT(inputFloat, resTb, log, resFloat, resTbFloat)

ext = input('premi per uscire ')
```

14.32 butterflySimulation.py

```

import os
import subprocess

from inputGen import generateCouples
from inputGen import generateW

from dataConversion import convertTo2C

from computeSingle import computeSingle

inputFloatReal = "butterflyFloatRealData.txt"
inputFloatImag = "butterflyFloatImageData.txt"

wFloat = "wFloat.txt"

#generazione dei file di input
generateCouples(inputFloatReal,2)
generateCouples(inputFloatImag,2)
generateW(wFloat)

#conversione dei valori float in complemento a 2

input2CR = "butterflyInputRealData.txt"
input2CI = "butterflyInputImageData.txt"
w2C = "butterflyW.txt"

convertTo2C(wFloat, w2C)
convertTo2C(inputFloatReal, input2CR)
convertTo2C(inputFloatImag, input2CI)

resTb = "butterflyTbResults.txt"

#compilazione dei listati vhdl ed avvio del testbench

# Setting up environment variables
os.environ["PATH"] += os.pathsep +
    "/software/mentor/modelsim_6.5c/modeltech/linux_x86_64/"
os.environ["LM_LICENSE_FILE"] = "1717@led-x3850-1.polito.it"

# Print out environment variables
os.system("echo $PATH")
os.system("echo $LM_LICENSE_FILE")

# Launch Modelsim simulation
print ("Starting simulation...")
process = subprocess.call(["vsim", "-c", "-do", "compileSingle.do"])
print ("Simulation completed")

#calcolo dei valori attesi ed analisi degli errori
computeSingle(wFloat, inputFloatReal, inputFloatImag, resTb)

```

```
ext=input( 'Premi per uscire ')
```

14.33 computeFFT.py

```

import sys
sys.path.append( ".." )

import numpy as np
from scipy.fft import fft

def computeFFT( fileInput , fileResult , fileLog , fileResFloat , fileResTbFloat ):

    fileIn = open( fileInput , "r" )
    fileRes = open( fileResult , "r" )
    fileL = open( fileLog , "w" )
    fileResF = open( fileResFloat , "w" )
    fileResFFT = open( fileResTbFloat , "w" )
    values = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
    relativeError = []
    meanErr=0.0
    nErr=0
    maxErr=0

    N = 2**15

    for line in fileIn:

        numbers = line.split(" ")
        if len(numbers) == 18:

            numbers.pop(0)
            for i in range(16):

                values[i] = float(numbers[i])

            #calcolo i valori attesi dell'fft
            attended = fft(values)

            resLine = fileRes.readline()
            results = resLine.split(" ")
            if len(results) == 17:
                results.pop(16)
                resLine = ""
                resFL = ""
                relativeError = []
                for i in range(16):
                    print(attended[i], float(results[i])/N)
                    relativeError.append(abs((attended[i].real -
                                              (float(results[i]) / N))))
                resLine+=(str(attended[i].real)+" ")
                meanErr+=abs(attended[i].real - float(results[i]) / N)
                nErr +=1
                resFL += str(float(results[i]) / N) + " "

```

```

if abs( ( attended[i].real -
          (float(results[i]) / N) )) > maxErr:
    maxErr = abs( ( attended[i].real - (float(results[i]) / N) ))

fileResFFT.write(resLine+"\n")
fileResF.write(resFL+"\n")
resLine = fileRes.readline()
results = resLine.split(" ")
if len(results) == 17:
    results.pop(16)
    resLine = ""
    resFL = ""
    relativeError = []
for i in range(16):

    print(attended[i].imag, float(results[i])/N)
    relativeError.append( (
        attended[i].imag - (float(results[i]) / N) ))
    resLine+=(str(attended[i].imag)+" ")
    meanErr+=abs(attended[i].imag - float(results[i]) / N)
    nErr+=1
    resFL += str(float(results[i]) / N) + " "
    if abs( ( attended[i].imag -
              (float(results[i]) / N) )) > maxErr:
        maxErr = abs( ( attended[i].imag - (float(results[i]) / N) ))
fileResFFT.write(resLine+"\n")

fileResF.write(resFL+"\n")
resLine = ""

meanErr /=nErr
attErr = (2**(-13))
fileL.write(" Errore massimo: "+str(maxErr)+"\n")
fileL.write(" Errore medio: "+str(meanErr)+"\n")
fileL.write(" Rapporto tra errore ottenuto ed atteso:
             "+str(maxErr/attErr)+"\n")

```

14.34 computeSingle.py

```

def updateMaxErr(max, err):
    if err > max:
        return err
    else:
        return max

def computeSingle(inWName, inRName, inIName, resName):
    fileInW = open(inWName, "r")
    fileInR = open(inRName, "r")
    fileInI = open(inIName, "r")
    fileRes = open(resName, "r")
    fileTable = open('singleTable.txt', 'w')
    fileLog = open('logSingle.txt', 'w')
    convFact = 2**18

    totErr = 0.0
    nErr = 0
    meanErr = 0
    maxErr = 0
    fileTable.write(r'\begin{center}\n\begin{tabular}{|c|c|c|c|c|}\hline & A & B & W & \\ \hline A & att & att & & [0.5ex] \\ \hline\hline & & & & \\ \hline')
    for lineW in fileInW:
        inW = lineW.split(" ")
        Wr = float(inW[0])
        Wi = float(inW[1])
        fileInI.seek(0)

        for lineI in fileInI:
            fileInR.seek(0)

            inI = lineI.split(" ")
            Ai = float(inI[0])
            Bi = float(inI[1])
            for lineR in fileInR:
                inR = lineR.split(" ")
                Ar = float(inR[0])
                Br = float(inR[1])

```

```

resR = fileRes.readline().split(" ")
ArRes = float(resR[0])/convFact
BrRes = float(resR[1])/convFact

resI = fileRes.readline().split(" ")
AiRes = float(resI[0])/convFact
BiRes = float(resI[1])/convFact

ArAtt = Ar + Br*Wr - Bi*Wi
BrAtt = Ar - Br*Wr + Bi*Wi
AiAtt = Ai + Br*Wi + Bi*Wr
BiAtt = Ai - Br*Wi - Bi*Wr

err = abs(ArAtt - ArRes)
totErr += err
maxErr = updateMaxErr(maxErr, err)

err = abs(BrAtt - BrRes)
totErr += err
maxErr = updateMaxErr(maxErr, err)

err = abs(AiAtt - AiRes)
totErr += err
maxErr = updateMaxErr(maxErr, err)

err = abs(BiAtt - BiRes)
totErr += err
maxErr = updateMaxErr(maxErr, err)

nErr += 4

fileTable.write(str(Ar) + ' ' + str(Ai) + ' j&' + str(Br) + '
' + str(Bi) + ' j&' + str(Wr) + '
' + str(Wi) + ' j&' + str(ArAtt) + '
' + str(AiAtt) + ' j&' + str(ArRes) + '
' + str(AiRes) + ' j&' + str(BrAtt) + '
' + str(BiAtt) + ' j&' + str(BrRes) + '
' + str(BiRes) + ' j&\n\\hline\n')

meanErr = totErr / nErr

fileLog.write("Errore massimo: " + str(maxErr) + '\n')
fileLog.write("Errore medio: " + str(meanErr) + '\n')
fileLog.write("Rapporto tra errore ottenuto ed
atteso: " + str(maxErr/(2**18)) + '\n')

fileTable.write(' \\\end{tabular} \\\end{center} ')
fileLog.close()
fileTable.close()
fileInI.close()
fileInR.close()

```

```
fileInW.close()  
fileRes.close()
```

14.35 dataConversion.py

```

N = 19
def TransformToBin(val):
    floatVal = val * (2 ** N)
    intVal = int(round(floatVal))
    if intVal >= 2 ** 19:
        intVal -= 1

    if intVal <= -2**19:
        intVal += 1

    if intVal < 0:
        intVal = (intVal ^ ((2**20)-1)) +1
    return intVal

def convertTo2C(intName, twoCName):
    try:
        #apri file
        #"butterflyGreatData.txt"
        fileReader = open(intName, "r")
        #"butterflySignedData.txt"
        fileWriter = open(twoCName, "w")
        for line in fileReader:
            listVal = line.split()
            for i in listVal:

                retVal = TransformToBin(float(i))

                if str(format(retVal, "020b"))[0] == '-':
                    fileWriter.write(format(retVal, "020b")[1:] + "\u2022")
                else:
                    fileWriter.write(format(retVal, "020b") + "\u2022")

                fileWriter.write("\n")
        fileReader.close()
        fileWriter.close()

    except Exception as e:
        print(e)

```

14.36 inputGen.py

```

import random
import math

def generateInputs(fileName, mode):
    #"butterfly2CData.txt"
    fileOut = open(fileName, "w")

N = 10000

if mode == "random":
    #genero N righe di 16 valori casuali tra -0.5 e 0.5
    for _ in range(N):
        values = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0]
        for i in values:
            i = (random.random() - 0.5)
            fileOut.write(" " + str(i))

        fileOut.write("\n")

def generateW(fileName):
    #creo i w seguendo le formule della teoria dell'fft
    fIn = open(fileName, "w")
    for m in range(0, 17):
        WR = math.cos((math.pi / 8) * m)
        WI = -math.sin((math.pi / 8) * m)
        fIn.write(str(WR) + " " + str(WI) + "\n")

def generateCouples(fileName, N):
    fileOut = open(fileName, "w")
    couple = [-1.0, -1.0]

    #salvo nel file le coppie di valori float per la singola butterfly
    #le coppie sono le combinazioni dei numeri tra -1
    #ed 1 con un passo di 2**(-N)

    for _ in range(2 ** (N + 1)):
        couple[1] = -1.0
        for _ in range(2 ** (N + 1)):

```

```
couple[1] += 2**(-N)
fileOut.write(str(couple[0])+' '+str(couple[1])+'\n')
```

```
couple[0] += 2**(-N)
```

14.37 listatiLatex.py

```
import os

ext = '.py'
language='Python'
fileOut = open('listatiLatex.txt', 'w')
for file in os.listdir():

    if file.endswith(ext):

        fileIn = open(file, 'r')

        fileOut.write('\\subsection{' + file + '}\n\\begin{lstlisting}[language=' + language + ']\n')

        for line in fileIn:

            fileOut.write(line)

        fileOut.write('\\end{lstlisting}')
        fileIn.close()
fileOut.close()
```

14.38 tableGen.py

```

fileTable = open( 'latexTable.txt' , 'w' )

fileInput = open( 'esempiZamboni.txt' , 'r' )
fileAttended = open( 'butterfly16x16fftFloatResults.txt' , 'r' )
fileResults = open( 'butterfly16x16tbFloatResults.txt' , 'r' )

for lineIn in fileInput:
    fileTable . write( '\begin{center}\n\begin{tabular}{||c|c|c||}\n\\hline\nInput & Atteso &
Risultato\\\\[0.5ex]\\n\\hline\\hline\\n' )
    cleanLineIn = lineIn [:-2]

    dataIn = cleanLineIn . split( ' ' )

    lineAtt = fileAttended . readline () [:-2]
    dataAttReal = lineAtt . split( ' ' )

    lineAtt = fileAttended . readline () [:-2]
    dataAttImag = lineAtt . split( ' ' )

    lineRes = fileResults . readline () [:-2]
    dataResReal = lineRes . split( ' ' )

    lineRes = fileResults . readline () [:-2]
    dataResImag = lineRes . split( ' ' )

    for i in range(16):
        fileTable . write( dataIn [ i+1]+
            '&' + dataAttReal [ i] + '+' + dataAttImag [ i]
            +'j&' + dataResReal [ i] + '+' + dataResImag [ i]+ 'j&\n\\hline\\n' )

    fileTable . write( '\end{tabular}\\end{center}' )

```