Laurea Magistrale in Ingegneria Elettronica

Relazione di progetto sulle architetture integrate

FFT e Butterfly

A.A. 2021/2022

Massetti, Marco 301163
s301163@studenti.polito.it

Fagnani, Pietro 303490
s303490@studenti.polito.it

Montorsi, Pietro 291057
s291057@studenti.polito.it

Salpietro, Salvatore 304137
s304137@studenti.polito.it

Docente: Prof. Maurizio Zamboni

Indice

Introduzione

L'obiettivo di questo progetto è la realizzazione di un'unità di elaborazione che esegua la FFT basata sul processore Butterfly.
La trasformata veloce di Fourier (FFT) è un algoritmo che permette di calcolare la trasformata discreta di Fourier (DFT) con una bassa complessità computazionale.
L'analisi di Fourier converte un segnale dal suo dominio originale (spesso tempo o spazio) in una rappresentazione nel dominio della frequenza. Nei sistemi digitali non è possibile fare la trasformata continua di un segnale in quanto richiederebbe di eseguire degli integrali e, quindi, di avere infiniti campioni. Per questo motivo, si usa la trasformata discreta, la quale è calcolabile decomponendo un segnale continuo in un insieme di valori discreti. Partendo da N campioni di ingresso nel dominio del tempo mediante un algoritmo, è possibile ottenere le N componenti in frequenza corrispondenti. La trasformata di Fourier discreta applicata a una sequenza di dati viene effettuata mediante la seguente formula:

$$X_N(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{1.1}$$

dove il fattore W_N prende il nome di "twiddle factor" e può essere espresso come segue:

$$W_N^m = cos(\frac{2\pi}{N}m) - jsin(\frac{2\pi}{N}m) \tag{1.2}$$

Attraverso alcuni passaggi matematici è facile verificare inoltre che $W_N^2 = W_{(N/2)}$. Questo approccio richiede un numero di operazioni dell'ordine di N^2 ed è quindi molto lento. L'algoritmo di Cooley Tukey permette di minimizzare il numero di operazioni ripetute, scomponendo ricorsivamente la trasformata come somme di trasformate su insiemi più piccoli di valori di ingresso:

$$X_N(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-j\frac{2\pi}{(N/2)}nk} + e^{-j\frac{2\pi}{N}k} \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-j\frac{2\pi}{(N/2)}nk} = G(k) + W_N^k H(k) \tag{1.3}$$

Da questo risulta evidente che il calcolo della DFT su N campioni è stato scomposto nel calcolo di due DFT entrambe su $\frac{N}{2}$ campioni. Tenendo conto, inoltre, che la DFT è periodica, di conseguenza, $G(k)$ e $H(k)$ sono anch'esse periodiche di periodo $\frac{N}{2}$, quindi: $G(k) = G(k + \frac{N}{2})$ e $H(k) = H(k + \frac{N}{2})$.
Reiterando l'operazione di suddivisione in sottoinsiemi via via più piccoli, l'insieme minimo su cui l'algoritmo opera sarà di due soli elementi. Il compito dell'algoritmo è, infatti, quello di calcolare $X(k) = G(k) + W_N^k H(k)$ e $X(k+\frac{N}{2}) = G(k) + W_N^k H(k)$ a partire da una coppia di campioni e di reiterare tale operazione per un numero di volte pari al $log_2(N)$.

Da queste espressioni deriva la struttura base dell'algoritmo, che, quindi, necessita di soli tre dati e prende il nome di Butterfly 1.1. Ripetendo questa unità di base, opportunamente interconnessa, si è in grado di passare dal set di dati in ingresso alla corretta trasformata discreta.

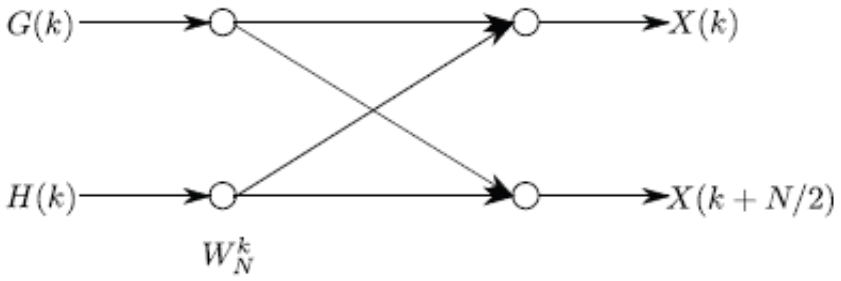


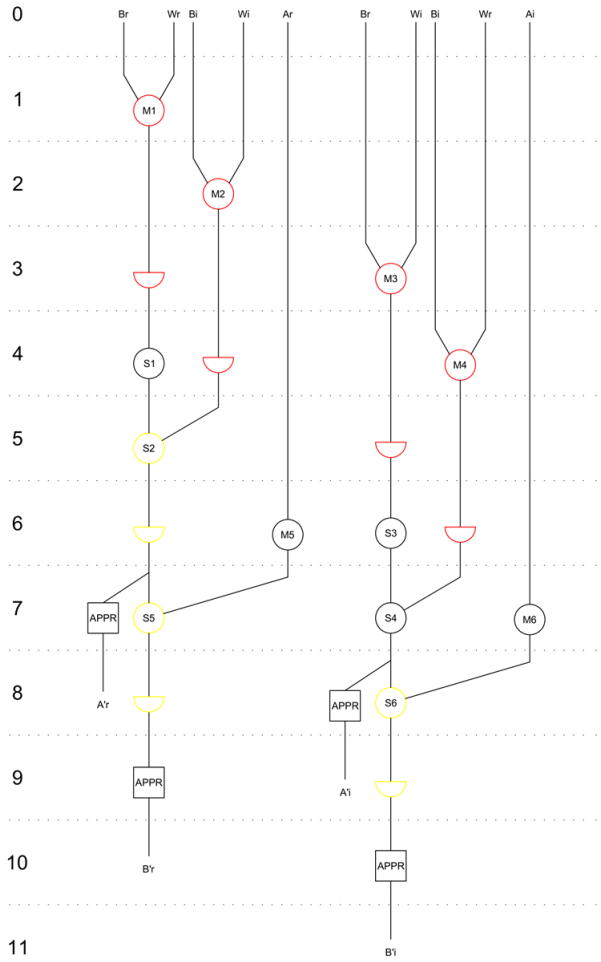
Figura 1.1: Butterfly

Per la realizzazione della singola Butterfly si è proceduto con i seguenti passaggi:

- Analisi Specifiche
  - Caratteristiche I/O:
    - Input: Ar, Ai, Br, Bi, Wi, Wr
    - Output: A'r, A'i, B'r, B'i
    - I dati A, B e W^k (sia le parti reali che immaginarie) sono definiti in forma frazionaria tra -1 e 1 in complemento C2 su 20 bit.
  - Hardware disponibile:
    - Moltiplicatore / shift
    - Sottrattore
    - Sommatore
- Studio di fattibilità
  - L'algoritmo da implementare non risulta complesso, di conseguenza il progetto è stato realizzato direttamente in VHDL senza test preliminari in altri linguaggi di programmazione.
- Scheduling
  - Confronto tra soluzioni ASAP e ALAP
- Ottimizzazione dei registri
  - Studio tramite timelife diagram
- Descrizione comportamentale

Data Flow Diagram

Per poter analizzare meglio il problema si è eseguito un architecture derivation, ossia un processo utile a trovare l'architettura migliore per il problema in esame.
Uno degli strumenti più efficaci per far ciò è il Data Flow Diagram (DFD). Il DFD è un metodo per rappresentare l'evoluzione del sistema tramite il flusso di dati. Si è dunque cercato il miglior scheduling che ottimizzi il timing e l'uso degli operatori. A tal fine sono state realizzate due versioni di DFD: ALAP (As Late As Possible) e ASAP (As Sone As Possible).
Come da specifiche, le soluzioni elaborate possiedono solo un sottrattore con un livello di pipe (rappresentato in giallo), un sommatore puramente combinatorio (rappresentato in nero), e un moltiplicatore che può eseguire operazioni di shift in modo combinatorio (rappresentato in nero) e operazioni di moltiplicazione con due livelli di pipe (rappresentato in rosso).



(a) Soluzione ALAP

$$M_1 = B_R W_R$$
$$M_2 = B_I W_I$$
$$M_3 = B_R W_I$$
$$M_4 = B_I W_R$$
$$M_5 = 2A_R$$
$$M_6 = 2A_I$$
$$\Sigma_1 = A_R + M_1$$
$$\Sigma_2 = \Sigma_1 - M_2 = A'_R$$
$$\Sigma_3 = A_I + M_3$$
$$\Sigma_4 = \Sigma_3 + M_4 = A'_I$$
$$\Sigma_5 = M_5 - \Sigma_2 = B'_R$$
$$\Sigma_6 = M_6 - \Sigma_4 = B'_I$$

(b) Operazioni

Figura 2.1: DFD

Nella versione ALAP, mostrato in 2.1a, ogni operazione è effettuata il più tardi possibile, mentre nella versione ASAP, mostrato in 2.2, tutti gli operatori lavorano appena hanno gli ingressi disponibili e possono generare le uscite.

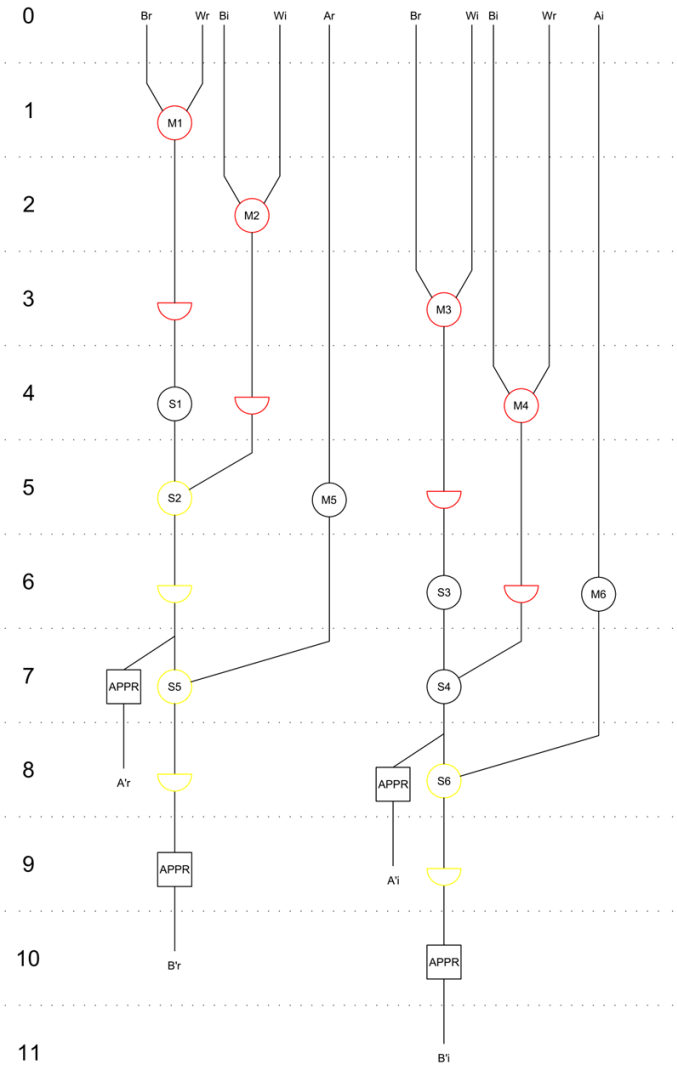


Figura 2.2: ASAP

Dato che il DFD risulta con due rami, si è deciso di iniziare con le operazioni del ramo di sinistra, ovvero M1. Se si fosse fatto il contrario, e quindi se la prima operazione fosse stata M3, allora il ramo opposto avrebbe dovuto aspettare un colpo di clock in più prima di poter effettuare la somma. Inoltre, avrebbe dovuto aspettare quattro colpi di clock per le sottrazioni, con un totale di sei dopo M1; in questo caso invece sono solo quattro dopo M3.
Confrontando le due versioni, è possibile notare come le differenze siano minime; infatti, l'unica considerevole è che le operazioni M5 e M6 avvengono un colpo di clock prima nella sequenza ASAP rispetto alla ALAP. Questa differenza apparentemente minima ha però dei riscontri rilevanti nel funzionamento continuo; infatti, con la soluzione ASAP è possibile iniziare una nuova elaborazione ogni sei colpi di clock mentre con la soluzione ALAP ogni sette. Questo avviene a causa del moltiplicatore, che deve poter essere utilizzabile nel momento in cui viene ricevuto un nuovo dato in ingresso. Per questo motivo, si è scelto di implementare la soluzione ASAP. Considerando quindi le operazioni come eseguite in parallelo, è possibile visualizzare la sequenza delle operazioni come illustrato in 2.3.

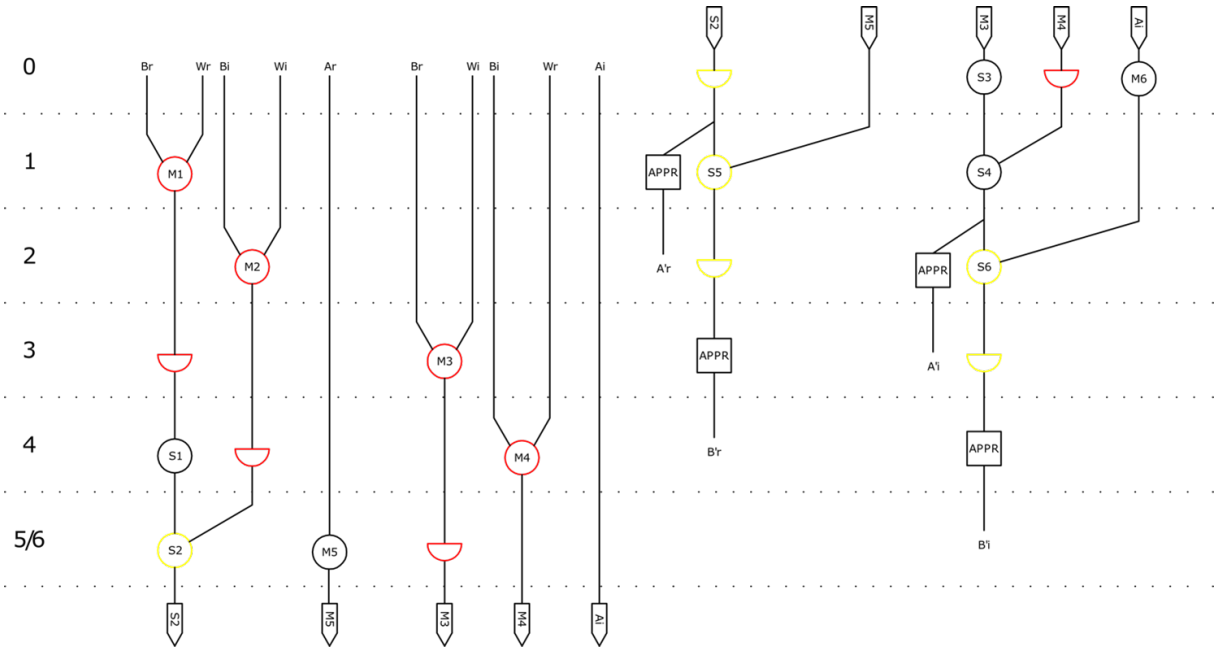


Figura 2.3: Sequenza operazioni risultante

Se non si considerasse la completa architettura di una FFT, siccome il primo dato che viene elaborato è B'r, si potrebbe dire di poterlo trasmettere immediatamente e dare il DONE un colpo di clock prima. Inoltre, se si analizza la natura della FFT, l'uscita A di una Butterfly potrebbe essere la B di quella successiva e, quindi, si potrebbe campionare in ingresso un colpo di clock più tardi A'i e B'i.
Questa scelta è stata scartata siccome avrebbe voluto dire dover utilizzare dei registri per la parte immaginaria dei numeri in ingresso alla FFT a monte della prima Butterfly se si considerano i segnali validi solo nel colpo di clock in cui lo start è 1. Inoltre, si sarebbero dovuti utilizzare dei registri anche per le parti reali delle ultime Butterfly, e il DONE di queste ultime avrebbe dovuto essere ritardato di un colpo di clock per poter avere i dati completi in uscita dalla FFT. Questo ragionamento avrebbe portato ad avere un segnale in uscita dalla FFT 3 colpi di clock prima, ma avrebbe richiesto l'utilizzo di una memoria maggiore.
Quindi, considerando i risultati ottenuti, si può dire che ogni Butterfly avrà un troughput di 12 colpi di clock, che però nelle esecuzioni successive può essere ridotto a 6 se lavora in modalità continua.

Studio del tempo di vita delle variabili

Dopo aver stabilito il miglior scheduling è stato necessario ottimizzare il numero di registri andando ad analizzare il tempo di vita delle variabili in ingresso e delle variabili temporanee (M1, M2, M3, M4, M5, M6, S1, S2, S3, S4, S5, S6). Analizzando in dettaglio la soluzione ASAP, è possibile notare che Ar serve a fino al quinto colpo di clock, Ai fino al sesto, Br fino al terzo e Bi fino al quarto.
Per quanto riguarda le variabili temporanee, M1 è valido nel quarto colpo di clock, M2 nel quinto, M3 nel sesto, M4 nel settimo, M5 dal sesto al settimo, M6 dal settimo all'ottavo, S1 nel quinto, S2 dal settimo all' undicesimo, S3 nel settimo, S4 dall' ottavo all' undicesimo, S5 dal nono all'undicesimo e S6 dal decimo all'undicesimo.
La tabella del tempo di vita delle variabili è mostrata in 3.1. Per la numerazione delle colonne all'interno della tabella è stata seguita quella del DFD e tra parentesi è stato indicato il numero dello stato corrispondente. Come è stato visto durante lo scheduling e come si nota dal timelife diagram, è possibile eseguire tutte le operazioni in soli sei stati diversi senza aggiungere nessun operatore o registro.

	Stato											
Variabile	0	1	2	3	4	5	6(0)	7(1)	8(2)	9(3)	10(4)	11(6)
Ar		Ar	Ar	Ar	Ar	Ar		Ar	Ar	Ar	Ar	Ar
Ai		Ai	Ai	Ai	Ai	Ai	Ai	Ai	Ai	Ai	Ai	Ai
Br		Br	Br	Br				Br	Br	Br		
Bi		Bi	Bi	Bi	Bi			Bi	Bi	Bi	Bi	
Wr												
Wi												
M1					A						A	
M2						A						A
M3							A					
S1						B						B
M4								A				
S2								C	1	1	1	1
M5							1	1				
S3								B				
M6								2	2			
S5										C	Br	Br
S4									B	2	2	2
S6											C	Bi
												DONE

N° registri	
20 bit	6
39 bit	3

Usati solo in modalità continua

Uscite

Figura 3.1: TimeLife Variabili (ALAP)

Ogni dato che deve essere salvato viene indicato con il nome del registro. Le variabili evidenziate in azzurro sono presenti perché, potendo ricevere un nuovo dato in ingresso ogni sei colpi di clock, è possibile gestire fino a due gruppi di dati alla volta lavorando in modo continuo. Quindi esse indicano i registri occupati dal secondo dato ricevuto.
Dal colpo di clock 7 della tabella, si evince che sono necessari nove registri per il corretto funzionamento della Butterfly, di cui sei a 20 bit e tre a 39, dato che in uscita dalle moltiplicazioni si hanno più bit.

Dalla tabella, inoltre, si deduce che sia conveniente utilizzare dei registri sparsi per le variabili di ingresso invece di un register file, perché essi sono relativamente pochi. Quindi, l'uso di un register file comporterebbe soltanto un aumento di complessità, di tempo di progettazione e una riduzione di velocità. Il principale svantaggio dei registri sparsi è che, avendo ridotto il numero di stati, anche quando viene usata la modalità singola alcuni registri campionano inutilmente come se si stesse lavorando in modalità continua, provocando quindi un aumento dei consumi.

DataPath

Partendo dalla tabella delle variabili e dal data flow diagram è stato realizzato il datapath della singola Butterfly, ossia una struttura con tutti gli operatori necessari per effettuare l'algoritmo

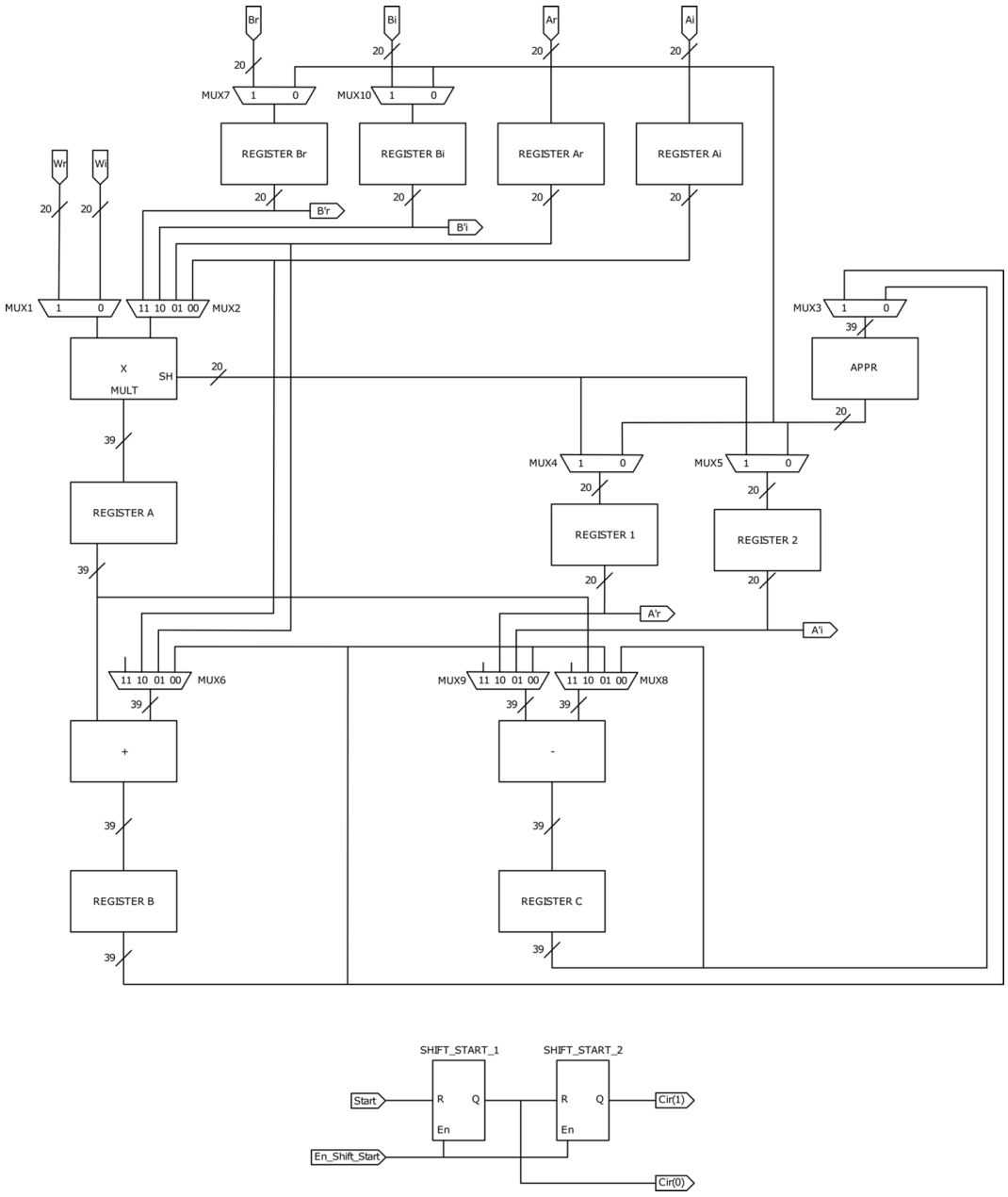


Figura 4.1: DataPath Butterfly

Come richiesto dalle specifiche, nel datapath della singola Butterfly è stato utilizzato un solo moltiplicatore, un sommatore e un sottrattore.
Dopo un'attenta analisi della tabella di vita delle variabili, è stato ricavato che il numero ottimale di registri per il corretto funzionamento della Butterfly è 9. Si possono infatti notare nel datapath tre registri a 39 bit (REGISTER A, B, C) e sei registri a 20 bit (REGISTER 1, 2, Ar, Ai, Br, Bi) per un totale di nove registri. Sono presenti anche 10 multiplexer per selezionare ad ogni colpo di clock il dato corretto da fornire agli operatori.
Le variabili di ingresso Ar, Ai, Br e Bi vengono caricate nei rispettivi registri (REGISTER Ar, Ai, Br, Bi). Inoltre, per quanto concerne le uscite, B'r e B'i vengono salvati rispettivamente nei registri Br e Bi, mentre le uscite A'r e A'i vengono salvati rispettivamente nei registri 1 e 2, e non nei registri Ar e Ai siccome potrebbero contenere un nuovo dato in modalità continua.

Nel datapath è presente anche un approssimatore, che funziona in modo combinatorio ed effettua l'arrotondamento solo sui dati in uscita alla Butterfly con la tecnica del Rounding to half-up. Infatti, si può notare come in ingresso all'approssimatore siano presenti 39 bit, mentre in uscita 20. Viene fatto questo perché qualunque FFT può essere vista come una serie di Butterfly collegate tra loro e, quindi, l'uscita di ognuna deve essere compatibile con l'ingresso della Butterfly seguente.
Inoltre, in ingresso a ciascuna Butterfly è necessario ricevere i segnali A e B nello stesso istante. Per far ciò, la macchina è stata realizzata in maniera tale che solo quando ha in uscita i segnali A e B validi allora dia il DONE in maniera tale che la Butterfly successiva possa campionarli correttamente.
La scelta di utilizzare registri sparsi anziché register file permette di avere uno schema più semplice. Essi, infatti, vengono posizionati tra un operatore e quello successivo in modo da ridurre la lunghezza dei percorsi combinatori; inoltre, il circuito risulta più veloce nel momento in cui nello stesso colpo di clock i dati vengono salvati in più registri. Usando un register file questo non sarebbe stato possibile, dal momento che sarebbe stato necessario selezionare l'indirizzo della memoria e, quindi, scrivere un solo dato alla volta, con conseguente allungamento delle tempistiche di lavoro della Butterfly.
Quindi, la scelta di cercare di massimizzare le prestazioni si riflette sulla decisione di non adoperare bus globali che avrebbero rallentato l'esecuzione dell'algoritmo.

Operatori

Moltiplicatore

Il moltiplicatore è stato realizzato attraverso la libreria *fixed_pkg*. Secondo le Sizing Rules il risultato ha come formalismo in uscita la somma dei bit interi dei fattori in ingresso più uno e la somma dei bit decimali. Quindi nel caso in esame, dati gli ingressi definiti signed Q0.19, si ottiene in uscita il risultato signed Q1.38. Per ottenere i due livelli di PIPE richiesti sono stati introdotti due registri prima dell'uscita. Da specifiche, il moltiplicatore deve avere una seconda uscita dove viene fornito, senza latenza, il dato sul secondo ingresso shiftato a sinistra di una posizione, realizzando una moltiplicazione per 2. È stato possibile implementarlo attraverso la funzione dedicata *Shift_left* nella libreria *numeric_std*.

Sottrattore e Sommatore

Il Sottrattore e il Sommatore sono stati realizzati con la libreria *Sfixed_pkg*. Implementano rispettivamente la propria funzione aritmetica, e hanno lo stesso formato d'uscita. Facendo sempre riferimento alle Sizing Rules del pacchetto, si ottiene il risultato definito Q1.38, poiché in ingresso si hanno due valori su 39 bit del tipo Q0.38. Il risultato ottenuto è formato dal massimo numero di bit interi più uno e il minimo numero di bit decimali degli ingressi. Nel caso del Sottrattore è richiesto un livello di PIPE realizzato introducendo un registro in uscita.

Approssimatore

L'approssimazione Rounding Hulf-Up è stata implementata seguendo la definizione, cioè, supponendo di voler arrotondare al n-esimo bit, si somma 1 al bit (n+1) e si tronca per ottenere gli n bit voluti. In altre parole, al valore da arrotondare viene sommato 0.5LSB (dove LSB è il peso del n-esimo bit), quindi si ottiene un arrotondamento asimmetrico. Infatti, nel caso in cui il valore si trovi a metà intervallo, l'arrotondamento viene sempre eseguito per eccesso, differentemente per ogni valore per cui è previsto l'arrotondamento per eccesso il valore complementare viene arrotondato per difetto.
In questo caso, l'arrotondamento desiderato è per ottenere in uscita un valore su 20 bit avendo in ingresso 39 bit, quindi, considerando che il dato deve essere successivamente scalato di un bit, viene sommato il diciannovesimo bit dopo il punto decimale, ovvero mezzo LSB del valore scalato. Se non fosse stato necessario scalare l'uscita, la somma del bit sarebbe dovuta avvenire sul ventesimo.

Scaling

Il singolo blocco Butterfly realizzato a virgola fissa fornirebbe in uscita i dati su un numero sempre maggiore di bit. Per mantenerli su una dinamica fissa si utilizza il metodo dell'Unconditional Block Floating Point Scaling, che combina i vantaggi della cifra fissa e di quella mobile; infatti, il dato in uscita viene scalato per riottenere la dinamica d'ingresso dello stadio precedente.
In questo caso vengono utilizzati due bit di guardia in ingresso alla FFT e viene scalato un bit dopo ogni stadio per recuperare il possibile bit di overflow.
Nella Butterfly realizzata lo Scaling viene eseguito internamente tra il blocco di Rounding e il registro in uscita attraverso il cablaggio dei due blocchi.

Control Unit

L'unità di controllo è stata progettata utilizzando un sequenziatore con indirizzamento implicito mediante la tecnica del "CONDITIONAL SEQUENCING PLA" per la gestione degli indirizzi di salto.
Studiando le operazioni effettuate e il tempo di vita delle variabili, si è constatato come il numero di colpi di clock necessari per avere i dati in uscita fosse di 12. Tuttavia, risulta possibile ricevere un nuovo dato in ingresso ogni 6 colpi di clock e quindi gestire fino a due gruppi di dati alla volta.
Questo può essere fatto con la gestione della modalità di lavoro continua da parte della CU con stati distinti, oppure, come nel caso proposto, decidere di effettuare ad ogni colpo di clock tutte le operazioni come se si lavorasse a pieno regime e dare il segnale di DONE solo quando si ha un dato valido. Per questo gli stati sono stati ridotti da 12 a solamente 7, di cui due (il 5 e il 6) differiscono solo per il bit "DONE".

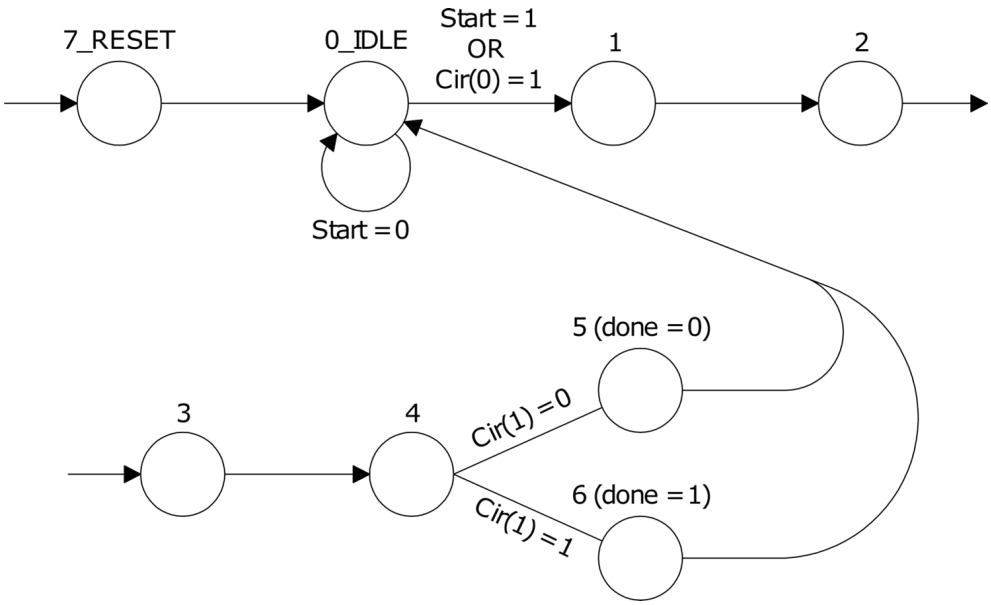


Figura 5.1: Diagramma degli Stati

Per gestire il segnale e sapere quando è valida l'uscita, si è deciso di utilizzare due flip flop in cascata, e quindi come una FIFO. Essi campionano il segnale di start nello stato zero, ovvero lo stato di IDLE. I due FF permettono di capire quando un segnale in uscita è pronto sia che si stia lavorando in modo continuo oppure no.
Questa gestione degli stati è sicuramente più dispendiosa dal punto di vista energetico, siccome in stato di IDLE vengono effettuate dei campionamenti che non sempre risultano utili. Tuttavia, viene dimezzato l'utilizzo della ROM permettendo in questo modo di poterla indirizzare con solamente 3 bit.

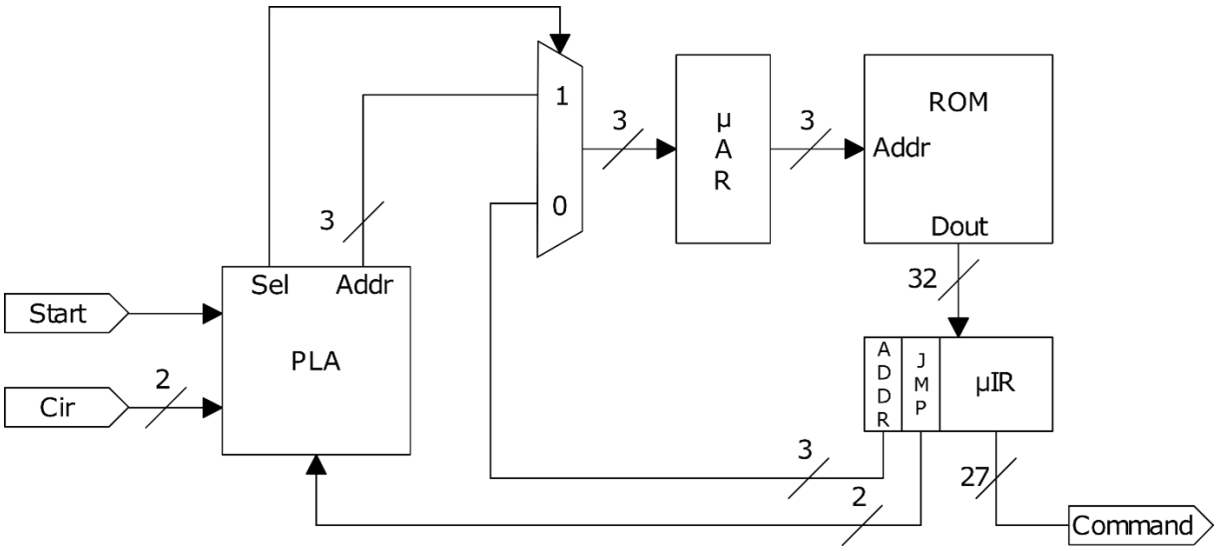


Figura 5.2: Schema Control Unit

Per l'indirizzamento della memoria viene utilizzato un micro address register (μAr) da 3 bit che campiona sul fronte di salita del clock. In uscita dalla microROM è presente un micro instruction register (μIr) a 32 bit che campiona i valori della memoria sul fronte di discesa del clock. Come si può poi osservare in 5.2, di questi 32 bit: 3 sono l'indirizzo successivo e sono collegati in ingresso ad un multiplexer a monte del μAr, 2 bit vengono utilizzati per gestire le condizioni di jump e sono collegati in ingresso alla PLA, infine gli altri 27 servono per la gestione del datapath.

La PLA riceve in ingresso i 2 bit di jump dalla ROM grazie ai quali è possibile comprendere se c'è una condizione di jump ed in quale indirizzo si trova la macchina; inoltre, essa riceve in ingresso lo start e i due bit dei due FF dal datapath. Il primo jump può avvenire nello stato zero, ovvero lo stato di IDLE. In questo caso si verifica che, quando è presente un segnale di start oppure il LSB dei FF è ad 1, viene fatto un salto allo stato 1 della CU. Nello stato di IDLE gli ingressi A e B vengono campionati in continuazione con i quattro registri: questo permette di partire non appena viene ricevuto lo start e, di conseguenza, dallo stato successivo si interrompe il campionamento dei registri in ingresso. Inoltre, nello stato di IDLE viene effettuato un campionamento continuo con i FF; quando viene ricevuto lo start, sarà esso stesso la condizione per il jump, mentre i FF nello stato successivo diventeranno "01". Quando la macchina tornerà nello stato di IDLE e non starà lavorando in modo continuo, sarà il LSB dei FF ad abilitare il jump, ottenendo poi lo stato successivo "10" per i FF. Il secondo jump possibile è quello che va dall'indirizzo 4 al 6, dato dal valore '1' del MSB dei FF. Se al primo ciclo non sono state terminate le operazioni, allora la macchina passerà dallo stato 4 al 5; dal secondo giro in uscita dai FF, si avrà "1x", e questo permette di accedere allo stato 6, che differisce dallo stato 5 per il solo segnale di DONE, pari ad '1'.

L'indirizzo di jump della PLA viene collegato in ingresso al mux a monte del μAr e ne gestisce il selettore (in caso di jump è l'indirizzo della PLA ad essere campionato dal μAr altrimenti quello presente nel μIR).
La ROM è un array di 8 righe da 32 bit, i quali contengono l'indirizzo successivo, la condizione di jump e i comandi per il datapath.

Nelle immagini 5.3 e 5.4 sono rappresentati i bit salvati all'interno della ROM necessari al funzionamento della Butterfly. Per poter essere meglio rappresentati sono stati divisi in due immagini. In caso di casella vuota significa che il bit deve essere a zero nel caso dei registri o è indifferente per i mux; in ogni caso è stato salvato a zero nella ROM. Il "NEXT_add" rappresenta lo stato successivo e a destra del "/" lo stato in cui si andrebbe nel caso di jump. I bit di "JUMP" sono quelli collegati alla PLA. I bit da "REG_AR" a "C_CLEAR" vengono direttamente trasmessi al datapath. I primi bit (REG_x) indicano l'enable di un registro, il quale deve essere sempre attivo nello stato precedente a quello in cui il dato deve essere salvato. I bit dei "MUX" controllano i

multiplexer all'interno del datapath. Seguendo lo stesso ragionamento fatto sopra, i multiplexer a monte di un registro vengono impostati un colpo di clock prima del campionamento; per gli operatori, invece, nello stesso colpo di clock, siccome hanno tutti la prima parte combinatoria (se ad esempio si parla di più livelli di pipe). Lo "SHIFT" controlla il moltiplicatore e, quando è ad 1, seleziona la moltiplicazione per 2. Il "DONE" è l'unico bit che differenzia lo stato 5 e 6; esso è attivo solo in quest'ultimo. Il "CIRCLE" è l'enable dei flip flop; è stato scelto questo nome perché è pratico pensare che comandi la parte che fa lavorare in modo circolare e continuo la Butterfly. Il "C_CLEAR" e il reset dei FF ed è presente solo nello stato di reset.

Seguendo il ragionamento fatto fino ad ora, è possibile distinguere le operazioni effettuate appena viene ricevuto il segnale di start (verde) da quelle che gestiscono il dato nel secondo "giro" della CU (azzurro). Grazie a questa distinzione, è facile capire come non ci sia alcun contrasto per nessun segnale; ciò ha permesso, quindi, di lavorare con due dati in un singolo stato. Per semplificare la visualizzazione del contenuto della ROM, esso è stato diviso in due tabelle: 5.3 e 5.4.

		USCITE								
Stato	NEXT_add	JUMP	REG_Ar	REG_Ai	REG_Br	REG_Bi	REG_A	REG_B	REG_C	REG_1	REG_2
0	000/001	01	1	1	1	1	1	1	1		1
1	010							1		1	
2	011								1		1
3	100				1		1		1		
4	101/110	10				1	1	1			
5/6	000		1				1			1	
Reset	000										

Figura 5.3: Contenuto μROM

	USCITE													
Stato	MUX1	MUX2	MUX3	MUX4	MUX5	MUX6	MUX7	MUX8	MUX9	MUX10	SHIFT	DONE	CIRCLE	C_CLEAR
0		00			1	10	1			1	1		1	
1	1	11	0	0		00		00	10					
2	0	10	1		0			01	01					
3	0	11												
4	1	10				01								
5/6		01		1			1	10	00	1	1	0/1		
Reset														1

Figura 5.4: Contenuto μROM

Segue una spiegazione di che cosa avviene in ogni stato, diviso in due parti: la prima per il primo "ciclo" e l'altra per il secondo, ovvero quando il dato è pronto a essere poi trasmesso.

- Stato 0 (IDLE):
  - Primo ciclo:
    Nello stato zero gli ingressi vengono campionati in continuazione; quando viene ricevuto il segnale di start il campionamento viene interrotto e si passa allo stato successivo. Nella tabella si può infatti notare che l'enable dei quattro registri di ingresso viene abilitato, e che i "MUX7" e "MUX10" sono ad '1' proprio per caricare i valori Br e Bi nei corrispondenti registri al colpo di clock seguente.
    Inoltre, il bit di "CIRCLE" è attivo perché è lo stato in cui viene effettuato lo shift dei flip flop.

- Secondo ciclo:
  REG_A = '1' per caricare M4 al prossimo colpo di clock;
  REG_C = '1' per caricare S2 al prossimo colpo di clock;
  REG_B = '1' per caricare S3 al prossimo colpo di clock;
  REG_2 = '1' per caricare M6 al prossimo colpo di clock;
  MUX2 = "00", MUX5=1 e MUL/SH = '1' per fare M6 = 2Ai;
  MUX6 = "10" per inviare M3 e Ai nel sommatore e calcolare S3;
  "SHIFT" è uguale ad 1 per calcolare M6.

- Stato 1:

  - Primo ciclo:
    Nello stato uno MUX1 = '1' e MUX2 = "11" per inviare Wr e Br nel moltiplicatore e calcolare M1.

  - Secondo ciclo:
    MUX8 = "00" e MUX9 = "10" per inviare M5 e S2 nel sottrattore e calcolare S5;
    MUX6 = "00" per inviare S3 e M4 nel sommatore e REG_B = '1' per caricare il risultato S4 in "REG_B" al prossimo colpo di clock;
    MUX3 = '0' per inviare S2 nell'approssimatore e MUX4 = '0' e REG_1 = '1' per caricare A'r in "REG_1" al prossimo colpo di clock.

- Stato 2:

  - Primo ciclo:
    Nello stato due MUX1 = '0' e MUX2 = "10" per inviare Wi e Bi nel moltiplicatore e calcolare M2.

  - Secondo ciclo:
    Registro REG_C = '1' per caricare S5 al prossimo colpo di clock;
    MUX8 = "01" e MUX9 = "01" per inviare S4 e M6 nel sottrattore e calcolare S6;
    MUX3 = '1' per inviare S4 nell' approssimatore e MUX5 = '0' e REG_2 = '1' per caricare A'i in "REG_2" al prossimo colpo di clock.

- Stato 3:

  - Primo ciclo:
    Nello stato tre MUX1 = '0' e MUX2 = "11" per inviare Wi e Br nel moltiplicatore e calcolare M3; inoltre, viene abilitato l'enable del "REG_A" per caricare al prossimo colpo di clock M1.

  - Secondo ciclo:
    Quindi REG_Br = '1' per caricare B'r al prossimo colpo di clock;
    REG_C = '1' per caricare S6 al prossimo colpo di clock;

- Stato 4:

  - Primo ciclo:
    Nello stato quattro MUX1 = '1' e MUX2 = "10" per inviare Wr e Bi nel moltiplicatore e calcolare M4; MUX6 = "01" per inviare Ar nel sommatore e calcolare S1;
    REG_B = '1' per caricare S1 al prossimo colpo di clock;

REG_A = ‘1’ per caricare M2 al prossimo colpo di clock.

- Secondo ciclo:
  Reg_Bi = ‘1’ per caricare Bi al prossimo colpo di clock.

- Stato 5:
  - Primo ciclo:
    Nello stato cinque MUX8 = “10” e MUX9 = “00” per inviare S1 e M2 nel sottrattore e calcolare S2;
    REG_A = ‘1’ per caricare M3 al prossimo colpo di clock;
    MUX2 = “01” e MUL/SH = ‘1’ per fare M5 = 2Ar;
    MUX4 = ‘1’ e REG_1 = ‘1’ per caricare M5 in “REG_1” al prossimo colpo di clock.

  - Secondo ciclo:
    Il registro REG_Ar = ‘1’;
    MUX7 = ‘1’, MUX10 = ‘1’ per campionare Br e Bi nello stato zero nel caso di modalità continua.

- Stato 6:
  Lo stato 6 equivale allo stato 5, l’unica differenza è il “DONE” attivo.

FFT

È stato quindi ora ottenuto il componente "Butterfly" che viene utilizzato per la creazione di un'unità di elaborazione che esegua la FFT. Dato che c'è la necessita di gestire 16 campioni in ingresso, l'architettura sarà organizzata come in 6.1, dove è possibile identificare le colonne come "livelli", ognuno dei quali è composto da otto Butterfly.

Collegando opportunatamente le uscite delle Butterfly agli ingressi di quelle successive come in 6.1 e utilizzando il segnale di DONE di un livello come START per il livello successivo, vengono ottenuti i segnali in uscita al 45° colpo di clock dopo aver campionato l'ingresso.

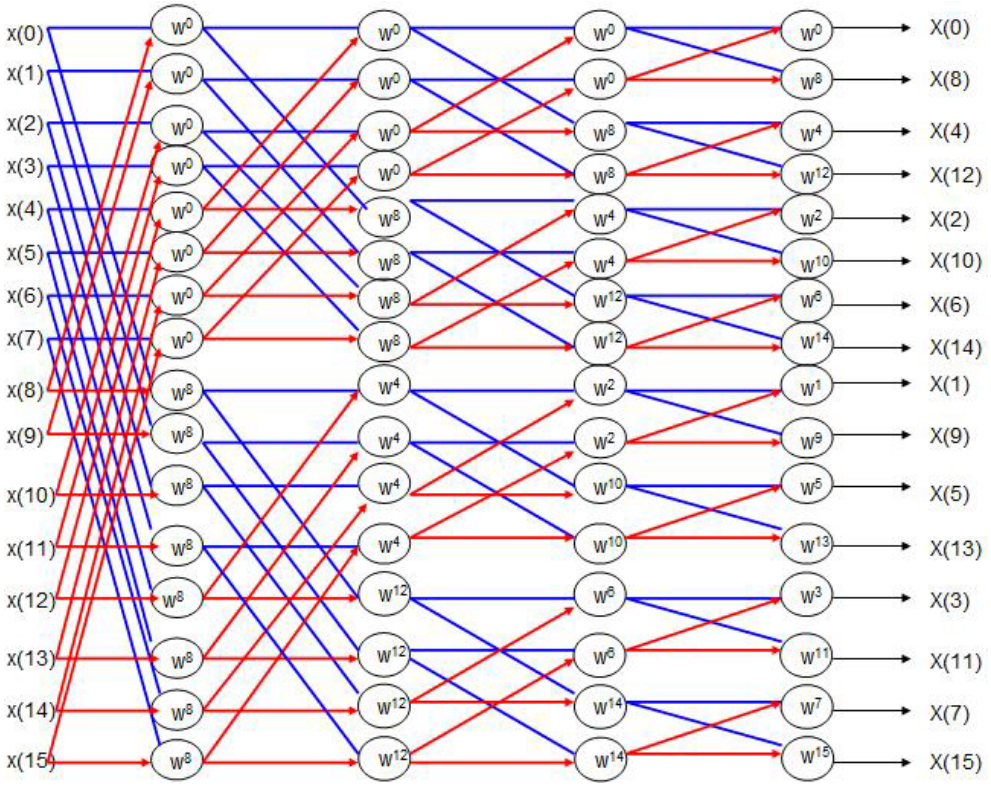


Figura 6.1: Collegamenti Butterfly

Inoltre, ogni Butterfly ha come ingresso oltre ad A e B il corrispettivo "twiddle factor" indicato in 6.1 come W^n. Il suo valore non è salvato in alcun registro, ma è stato descritto in VHDL come valore costante in ingresso ad ogni singolo componente.
Grazie allo scaling di un bit effettuato ad ogni livello in uscita, si ottiene il risultato teorico atteso diviso per 2^4.
Gli output della FFT non seguono la numerazione mostrata in 6.1, vengono invece numerati in ordine crescente a partire da quello più in alto, come fatto per gli ingressi.

Simulazione

Al fine di verificare il funzionamento dei dispositivi, è stato necessario eseguire delle simulazioni. I test sono stati svolti usando il software Modelsim che, dopo aver verificato le descrizioni dei componenti e quella di un testbench in cui vengono descritti gli stimoli da generare, è in grado di simulare il funzionamento del dispositivo e di rendere disponibili gli output ottenuti.

I diversi test sono stati eseguiti in modo gerarchico, partendo dai componenti a basso livello fino alle strutture più complesse. Vengono di seguito riportati i procedimenti seguiti per la simulazione della singola Butterfly e della catena in grado di eseguire la FFT.

Test Butterfly

Per testare il componente è stato necessario descrivere un tesbench in grado di fornire gli input con i tempismi adeguati e, allo stesso tempo, di restituire i risultati ottenuti.

Data l'elevata quantità di combinazioni di input possibili, si è scelto di creare un testbench che ottenga i valori di ingresso da un file e salvi gli output in un secondo file. Questo modo di operare ha il vantaggio di rendere automatizzabile tutta la procedura di verifica, infatti, è stato sviluppato uno script in linguaggio Python in grado di eseguire tutte le operazioni necessarie.

Tale script esegue la seguente sequenza di operazioni:

- Generazione sequenze di input: specificando il numero di variabili ed il rispettivo numero di bit, vengono generate in modo casuale delle sequenze di valori (Ar, Ai, Br, Bi, Wr, Wi).
- Calcolo output: contestualmente alla generazione degli ingressi, viene calcolato il risultato dell'operazione di Butterfly e viene salvato su un file.
- Esecuzione della simulazione: la simulazione Modelsim viene lanciata restando all'interno dello script.
- Verifica delle uscite: il file di output generato dalla simulazione viene comparato con il file di riferimento generato dallo script e le eventuali incongruenze tra i due vengono riportate in un file di log.

Dato il grande numero di possibili combinazioni di input, risulta impossibile eseguire un test esaustivo. Sono quindi stati eseguiti due test separati:

- Ingressi casuali: eseguito generando un grande numero di combinazioni casuali degli ingressi per verificare che non ci siano errori su bit specifici.
- Ingressi ai limiti della dinamica: eseguito generando delle combinazioni in cui gli ingressi si trovano ai limiti della loro possibile dinamica, per rilevare eventuali errori di overflow.

Dato che la Butterfly ha la possibilità di lavorare a due regimi differenti, è stato necessario testarli entrambi. Sfortunatamente, data la temporizzazione diversa tra le due modalità, non è possibile eseguire un test univoco per entrambe, quindi, sono stati realizzati due testbench separati. Il resto necessario alla simulazione rimane invariato, ad esclusione del file contenente i comandi richiesti da Modelsim per conoscere il file di testbench da usare.

Test FFT

Per verificare il funzionamento del dispositivo in grado di fare il calcolo della FFT, è stato eseguito un test di tipo manuale. Questo perché, avendo già verificato il funzionamento della singola Butterfly, non è necessario generare un grande numero di input per assicurarsi che il sistema funzioni. È comunque importante verificare che non esistano problemi di temporizzazione o di collegamento tra i diversi moduli.

Il tesbench questa volta è stato realizzato impostando manualmente la sequenza ed i valori dei segnali di ingresso e, a seguito della simulazione, è stata verificata la correttezza degli output analizzando le forme d'onda ottenute.

Le combinazioni di ingresso sono state ricavate in modo casuale, calcolando per ognuna di esse le uscite corrispondenti sfruttando il software MATLAB.

Anche in questo caso il sistema ha due modalità di lavoro separate. Per testarle è stato esclusivamente modificata la temporizzazione del testbench a seconda della modalità da simulare.

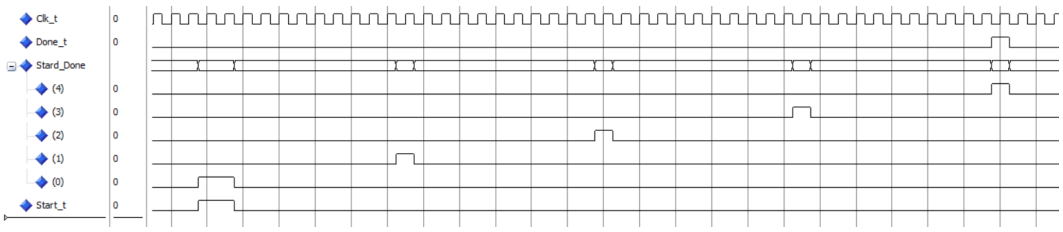


Figura 7.1: Verifica propagazione segnali butterfly

Il segnale di DONE, come atteso, viene propagato da una Butterfly alla successiva giungendo all'uscita complessiva del dispositivo dopo un ritardo di 44 periodi di clock.

Dopo una prima simulazione in cui si nota la corretta propagazione dei segnali di DONE tra uno stadio e il successivo, è possibile verificare la correttezza dei risultati generati.

Sono state utilizzate diverse sequenze di input per verificare correttamente che non siano presenti errori di calcolo; in questo caso ne vengono riportate solo un paio a titolo esemplificativo.

È sufficiente confrontare solamente 9 valori dell'output, in quanto i restanti sono i valori complessi coniugati.

- Simulazione con impulso come segnale di ingresso:

$$INPUT = \begin{bmatrix} 0.25 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad OUTPUT = \begin{bmatrix} 0.015625 \\ 0.015625 \\ 0.015625 \\ 0.015625 \\ 0.015625 \\ 0.015625 \\ 0.015625 \\ 0.015625 \\ 0.015625 \end{bmatrix}$$

Figura 7.2: Valori di riferimento impulso

Figura 7.3: Risultato simulazione con impulso in ingresso

- Simulazione con gradino come segnale di ingresso:

$$INPUT = \begin{bmatrix} 0.125 \\ 0.125 \\ 0.125 \\ 0.125 \\ 0.125 \\ 0.125 \\ 0.125 \\ 0.125 \\ 0.125 \\ -0.125 \\ -0.125 \\ -0.125 \\ -0.125 \\ -0.125 \\ -0.125 \\ -0.125 \end{bmatrix} \quad OUTPUT = \begin{bmatrix} 0.015625 \\ -78.55 \cdot 10^{-3} i \\ 0.015625 \\ -23.39 \cdot 10^{-3} i \\ 0.015625 \\ -10.44 \cdot 10^{-3} i \\ 0.015625 \\ -3.11 \cdot 10^{-3} i \\ 0.015625 \end{bmatrix}$$

Figura 7.4: Valori di riferimento gradino

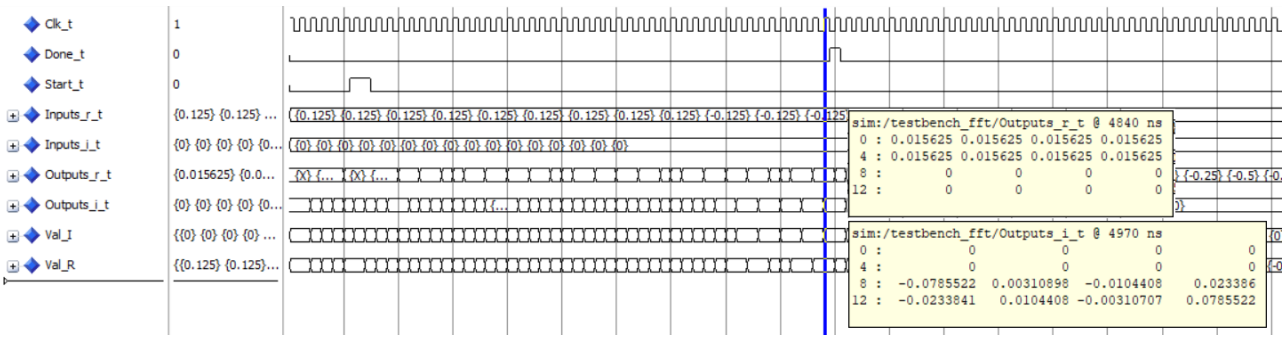


Figura 7.5: Risultato simulazione con gradino in ingresso

I risultati delle simulazioni coincidono con quanto atteso, infatti, come è visibile dalle onde generate da Modelsim, il dispositivo è in grado di calcolare correttamente le uscite. Si può notare che gli indici degli output seguono la numerazione spiegata in precedenza, ad esempio X(1) teorico corrisponde all'uscita numero 8. Dopo aver verificato il funzionamento della modalità di lavoro non continua, è stato infine verificato il funzionamento in modalità continua.

- Simulazione in modalità continua:

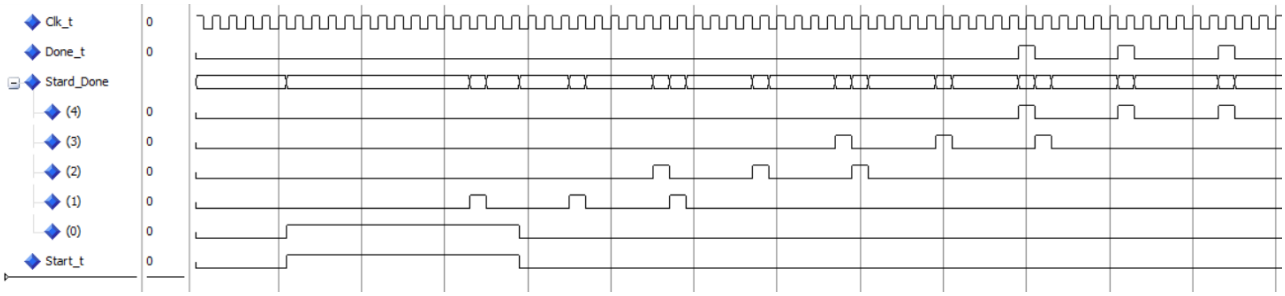


Figura 7.6: Verifica propagazione segnali di done in modalita continua

Anche in modalità continua la propagazione dei segnali risulta corretta. Tutte le Butterfly intermedie ricevono il segnale di START quando hanno appena asserito il segnale di DONE, e, quindi, si trovano nello stato "0". In questo modo il primo segnale di DONE in uscita ha lo stesso ritardo della modalità continua, ma i seguenti sono separati da solo 6 periodi di clock. Quindi, si ha un throughput di una FFT ogni sei colpi di clock.

Il test della modalità continua è stato ripetuto verificando la correttezza dei calcoli. Non vengono qui riportati i risultati ottenuti con Modelsim a causa della difficoltà di lettura, ma tutti i risultati ottenuti rispecchiano quelli attesi.

Conclusioni

Il dispositivo realizzato è in grado di eseguire operazioni di Butterfly in modo rapido e preciso su degli ingressi complessi espressi in formato fractional point su venti bit. I risultati forniti in uscita hanno un'alta precisione grazie ad un parallelismo interno che permette di evitare troncamenti ed un circuito di arrotondamento per rendere i valori rappresentabili su venti bit minimizzando gli errori di approssimazione.

Le operazioni vengono svolte in una sequenza che ottimizza il timing e l'uso degli operatori in modo da ridurre i cicli necessari a terminare le operazioni.

L'architettura interna è stata sviluppata in modo da non avere bus globali e minimizzando i collegamenti tra gli elementi del datapath: così facendo, ogni operazione ha una linea dedicata su cui trasmettere i dati. Questo permette di avere una latenza ridotta e di minimizzare il numero di elementi logici richiesti per la realizzazione del circuito. Le uscite di ogni operatore sono inoltre campionate da dei registri intermedi in modo da massimizzare la frequenza di funzionamento riducendo la lunghezza dei percorsi tra elementi combinatori.

La Butterfly è in grado di lavorare in una modalità continua adatta a raddoppiare il troughput delle operazioni senza alcun compromesso sulla precisione di risultati o sulla frequenza di lavoro. Il componente è inoltre studiato in modo da essere direttamente interfacciabile con altre sue istanze. Così facendo, è possibile eseguire dei calcoli di FFT seguendo l'algoritmo di Cooley-Tukey senza la possibilità di avere overflow e senza la di necessità di alcun componente aggiuntivo, quindi, di ulteriori ritardi sui risultati. Infatti, l'esecuzione di una FFT dipende esclusivamente dal numero di Butterfly concatenate.

Vengono di seguito riassunte le principali caratteristiche:

- I/O frazionari in complemento a due su 20 bit
- Latenza di 11 colpi di clock per la Butterfly
- Latenza di 44 colpi di clock per la FFT
- Troughput di 6 colpi di clock (in modalità continua)
- Parallelismo interno massimo di 39 bit
- Approssimazione Half Up delle uscite
- Possibilità di lavoro in modalità singola o continua
- Possibilità di concatenazione delle Butterfly per eseguire la FFT

Appendice

VHDL

FFT

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;
USE work.types_pkg.ALL;

-------------------------------------------------
-- FFT 16x16 a 20 bit
-------------------------------------------------

ENTITY FFT IS
        PORT (
                Clock, Start : IN STD_LOGIC;
                Inputs_r, Inputs_i : IN vector_of_16_sfixed;
                Done : OUT STD_LOGIC;
                Outputs_r, Outputs_i : OUT vector_of_16_sfixed
        );
END FFT;

ARCHITECTURE Behavior OF FFT IS

        COMPONENT BUTTERFLY IS
                PORT (
                        Clock, Start : IN STD_LOGIC;
                        Ar, Ai, Br, Bi, Wr, Wi : IN sfixed(0 DOWNTO -19);
                        Done : OUT STD_LOGIC;
                        A1r, A1i, B1r, B1i : OUT sfixed(0 DOWNTO -19)
                );
        END COMPONENT;

        --Valori reali e immaginari tra le diverse butterfly.
```

```
	--Sono matrici di sfixed, primo indice indica il livello,
	--secondo indice indica il numero della labbetfly nel livello
	SIGNAL Val_R, Val_I : array_of_sfixed;

	--Segnali di start e done tra i diversi stadi di butterfly.
	--Viene usato il done di una sola butterfly di uno stadio
	--per dare lo start a tutte le butterfly dello stadio successivo
	SIGNAL Stard_Done : STD_LOGIC_VECTOR(4 DOWNTO 0);

	--Parte reale dei twiddle factors
	CONSTANT Wr : vector_of_8_sfixed := (
		to_sfixed(1, 0, -19), to_sfixed(0.923879532511287, 0, -19),
		to_sfixed(0.707106781186548, 0, -19),
		to_sfixed(0.382683432365090, 0, -19), to_sfixed(0, 0, -19),
		to_sfixed(-0.382683432365090, 0, -19),
		to_sfixed(-0.707106781186547, 0, -19),
		to_sfixed(-0.923879532511287, 0, -19)
	);

	--Parte immaginaria dei twiddle factors
	CONSTANT Wi : vector_of_8_sfixed := (
		to_sfixed(0, 0, -19), to_sfixed(-0.382683432365090, 0, -19),
		to_sfixed(-0.707106781186547, 0, -19),
		to_sfixed(-0.923879532511287, 0, -19), to_sfixed(-1, 0, -19),
		to_sfixed(-0.923879532511287, 0, -19),
		to_sfixed(-0.707106781186547, 0, -19),
		to_sfixed(-0.382683432365090, 0, -19)
	);

BEGIN

	Stard_Done(0) <= Start; --Collegamento segnale di start primo stadio di butterfly
	Val_R(0) <= Inputs_r; --Collegamento ingresso parte reale primo stadio di butterfly
	Val_I(0) <= Inputs_i; --Collegamento ingresso immaginaria reale primo stadio di butterfly

	Outputs_r <= Val_R(4); --Collegamento uscita parte reale ultimo stadio di butterfly
	Outputs_i <= Val_I(4); --Collegamento uscita parte immaginaria ultimo stadio di butterfly
	Done <= Stard_Done(4); --Collegamento segnale di done ultimo stadio di butterfly

	---------------------------------
	--Primo stadio di butterfly
	---------------------------------
	--I numeri nel nome del componente indicano rispettivamente:
	--il numero dello stadio; il numero del gruppo; il numero della butterfly nel gruppo
	BUTTERFLY_0_0_0 : BUTTERFLY PORT MAP(
		Clock => Clock, Start => Stard_Done(0), Done => Stard_Done(1),
		Ar => Val_R(0)(0), Ai => Val_I(0)(0), Br => Val_R(0)(8), Bi => Val_I(0)(8),
		Wr => Wr(0), Wi => Wi(0),
		A1r => Val_R(1)(0), A1i => Val_I(1)(0), B1r => Val_R(1)(8), B1i => Val_I(1)(8));
```

```
BUTTERFLY_0_0_1 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(0),
        Ar => Val_R(0)(1), Ai => Val_I(0)(1), Br => Val_R(0)(9), Bi => Val_I(0)(9),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(1)(1), A1i => Val_I(1)(1), B1r => Val_R(1)(9), B1i => Val_I(1)(9));

BUTTERFLY_0_0_2 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(0),
        Ar => Val_R(0)(2), Ai => Val_I(0)(2), Br => Val_R(0)(10), Bi => Val_I(0)(10),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(1)(2), A1i => Val_I(1)(2), B1r => Val_R(1)(10), B1i => Val_I(1)(10));

BUTTERFLY_0_0_3 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(0),
        Ar => Val_R(0)(3), Ai => Val_I(0)(3), Br => Val_R(0)(11), Bi => Val_I(0)(11),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(1)(3), A1i => Val_I(1)(3), B1r => Val_R(1)(11), B1i => Val_I(1)(11));

BUTTERFLY_0_0_4 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(0),
        Ar => Val_R(0)(4), Ai => Val_I(0)(4), Br => Val_R(0)(12), Bi => Val_I(0)(12),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(1)(4), A1i => Val_I(1)(4), B1r => Val_R(1)(12), B1i => Val_I(1)(12));

BUTTERFLY_0_0_5 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(0),
        Ar => Val_R(0)(5), Ai => Val_I(0)(5), Br => Val_R(0)(13), Bi => Val_I(0)(13),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(1)(5), A1i => Val_I(1)(5), B1r => Val_R(1)(13), B1i => Val_I(1)(13));

BUTTERFLY_0_0_6 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(0),
        Ar => Val_R(0)(6), Ai => Val_I(0)(6), Br => Val_R(0)(14), Bi => Val_I(0)(14),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(1)(6), A1i => Val_I(1)(6), B1r => Val_R(1)(14), B1i => Val_I(1)(14));

BUTTERFLY_0_0_7 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(0),
        Ar => Val_R(0)(7), Ai => Val_I(0)(7), Br => Val_R(0)(15), Bi => Val_I(0)(15),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(1)(7), A1i => Val_I(1)(7), B1r => Val_R(1)(15), B1i => Val_I(1)(15));

---------------------------------
--Secondo stadio di butterfly
---------------------------------
BUTTERFLY_1_0_0 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(1), Done => Stard_Done(2),
        Ar => Val_R(1)(0), Ai => Val_I(1)(0), Br => Val_R(1)(4), Bi => Val_I(1)(4),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(2)(0), A1i => Val_I(2)(0), B1r => Val_R(2)(4), B1i => Val_I(2)(4));
```

```
BUTTERFLY_1_0_1 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(1),
        Ar => Val_R(1)(1), Ai => Val_I(1)(1), Br => Val_R(1)(5), Bi => Val_I(1)(5),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(2)(1), A1i => Val_I(2)(1), B1r => Val_R(2)(5), B1i => Val_I(2)(5));

BUTTERFLY_1_0_2 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(1),
        Ar => Val_R(1)(2), Ai => Val_I(1)(2), Br => Val_R(1)(6), Bi => Val_I(1)(6),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(2)(2), A1i => Val_I(2)(2), B1r => Val_R(2)(6), B1i => Val_I(2)(6));

BUTTERFLY_1_0_3 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(1),
        Ar => Val_R(1)(3), Ai => Val_I(1)(3), Br => Val_R(1)(7), Bi => Val_I(1)(7),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(2)(3), A1i => Val_I(2)(3), B1r => Val_R(2)(7), B1i => Val_I(2)(7));

BUTTERFLY_1_1_0 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(1),
        Ar => Val_R(1)(8), Ai => Val_I(1)(8), Br => Val_R(1)(12), Bi => Val_I(1)(12),
        Wr => Wr(4), Wi => Wi(4),
        A1r => Val_R(2)(8), A1i => Val_I(2)(8), B1r => Val_R(2)(12), B1i => Val_I(2)(12));

BUTTERFLY_1_1_1 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(1),
        Ar => Val_R(1)(9), Ai => Val_I(1)(9), Br => Val_R(1)(13), Bi => Val_I(1)(13),
        Wr => Wr(4), Wi => Wi(4),
        A1r => Val_R(2)(9), A1i => Val_I(2)(9), B1r => Val_R(2)(13), B1i => Val_I(2)(13));

BUTTERFLY_1_1_2 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(1),
        Ar => Val_R(1)(10), Ai => Val_I(1)(10), Br => Val_R(1)(14), Bi => Val_I(1)(14),
        Wr => Wr(4), Wi => Wi(4),
        A1r => Val_R(2)(10), A1i => Val_I(2)(10), B1r => Val_R(2)(14), B1i => Val_I(2)(14));

BUTTERFLY_1_1_3 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(1),
        Ar => Val_R(1)(11), Ai => Val_I(1)(11), Br => Val_R(1)(15), Bi => Val_I(1)(15),
        Wr => Wr(4), Wi => Wi(4),
        A1r => Val_R(2)(11), A1i => Val_I(2)(11), B1r => Val_R(2)(15), B1i => Val_I(2)(15));

---------------------------------
--Terzo stadio di butterfly
---------------------------------
BUTTERFLY_2_0_0 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(2), Done => Stard_Done(3),
        Ar => Val_R(2)(0), Ai => Val_I(2)(0), Br => Val_R(2)(2), Bi => Val_I(2)(2),
        Wr => Wr(0), Wi => Wi(0),
```

```
        A1r => Val_R(3)(0), A1i => Val_I(3)(0), B1r => Val_R(3)(2), B1i => Val_I(3)(2));

BUTTERFLY_2_0_1 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(2),
        Ar => Val_R(2)(1), Ai => Val_I(2)(1), Br => Val_R(2)(3), Bi => Val_I(2)(3),
        Wr => Wr(0), Wi => Wi(0),
        A1r => Val_R(3)(1), A1i => Val_I(3)(1), B1r => Val_R(3)(3), B1i => Val_I(3)(3));

BUTTERFLY_2_1_0 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(2),
        Ar => Val_R(2)(4), Ai => Val_I(2)(4), Br => Val_R(2)(6), Bi => Val_I(2)(6),
        Wr => Wr(4), Wi => Wi(4),
        A1r => Val_R(3)(4), A1i => Val_I(3)(4), B1r => Val_R(3)(6), B1i => Val_I(3)(6));

BUTTERFLY_2_1_1 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(2),
        Ar => Val_R(2)(5), Ai => Val_I(2)(5), Br => Val_R(2)(7), Bi => Val_I(2)(7),
        Wr => Wr(4), Wi => Wi(4),
        A1r => Val_R(3)(5), A1i => Val_I(3)(5), B1r => Val_R(3)(7), B1i => Val_I(3)(7));

BUTTERFLY_2_2_0 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(2),
        Ar => Val_R(2)(8), Ai => Val_I(2)(8), Br => Val_R(2)(10), Bi => Val_I(2)(10),
        Wr => Wr(2), Wi => Wi(2),
        A1r => Val_R(3)(8), A1i => Val_I(3)(8), B1r => Val_R(3)(10), B1i => Val_I(3)(10));

BUTTERFLY_2_2_1 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(2),
        Ar => Val_R(2)(9), Ai => Val_I(2)(9), Br => Val_R(2)(11), Bi => Val_I(2)(11),
        Wr => Wr(2), Wi => Wi(2),
        A1r => Val_R(3)(9), A1i => Val_I(3)(9), B1r => Val_R(3)(11), B1i => Val_I(3)(11));

BUTTERFLY_2_3_0 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(2),
        Ar => Val_R(2)(12), Ai => Val_I(2)(12), Br => Val_R(2)(14), Bi => Val_I(2)(14),
        Wr => Wr(6), Wi => Wi(6),
        A1r => Val_R(3)(12), A1i => Val_I(3)(12), B1r => Val_R(3)(14), B1i => Val_I(3)(14));

BUTTERFLY_2_3_1 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(2),
        Ar => Val_R(2)(13), Ai => Val_I(2)(13), Br => Val_R(2)(15), Bi => Val_I(2)(15),
        Wr => Wr(6), Wi => Wi(6),
        A1r => Val_R(3)(13), A1i => Val_I(3)(13), B1r => Val_R(3)(15), B1i => Val_I(3)(15));

---------------------------------
--Quarto stadio di butterfly
---------------------------------
BUTTERFLY_3_0_0 : BUTTERFLY PORT MAP(
        Clock => Clock, Start => Stard_Done(3), Done => Stard_Done(4),
        Ar => Val_R(3)(0), Ai => Val_I(3)(0), Br => Val_R(3)(1), Bi => Val_I(3)(1),
```

```
            Wr => Wr(0), Wi => Wi(0),
            A1r => Val_R(4)(0), A1i => Val_I(4)(0), B1r => Val_R(4)(1), B1i => Val_I(4)(1));

    BUTTERFLY_3_1_1 : BUTTERFLY PORT MAP(
            Clock => Clock, Start => Stard_Done(3),
            Ar => Val_R(3)(2), Ai => Val_I(3)(2), Br => Val_R(3)(3), Bi => Val_I(3)(3),
            Wr => Wr(4), Wi => Wi(4),
            A1r => Val_R(4)(2), A1i => Val_I(4)(2), B1r => Val_R(4)(3), B1i => Val_I(4)(3));

    BUTTERFLY_3_2_2 : BUTTERFLY PORT MAP(
            Clock => Clock, Start => Stard_Done(3),
            Ar => Val_R(3)(4), Ai => Val_I(3)(4), Br => Val_R(3)(5), Bi => Val_I(3)(5),
            Wr => Wr(2), Wi => Wi(2),
            A1r => Val_R(4)(4), A1i => Val_I(4)(4), B1r => Val_R(4)(5), B1i => Val_I(4)(5));

    BUTTERFLY_3_3_3 : BUTTERFLY PORT MAP(
            Clock => Clock, Start => Stard_Done(3),
            Ar => Val_R(3)(6), Ai => Val_I(3)(6), Br => Val_R(3)(7), Bi => Val_I(3)(7),
            Wr => Wr(6), Wi => Wi(6),
            A1r => Val_R(4)(6), A1i => Val_I(4)(6), B1r => Val_R(4)(7), B1i => Val_I(4)(7));

    BUTTERFLY_3_4_4 : BUTTERFLY PORT MAP(
            Clock => Clock, Start => Stard_Done(3),
            Ar => Val_R(3)(8), Ai => Val_I(3)(8), Br => Val_R(3)(9), Bi => Val_I(3)(9),
            Wr => Wr(1), Wi => Wi(1),
            A1r => Val_R(4)(8), A1i => Val_I(4)(8), B1r => Val_R(4)(9), B1i => Val_I(4)(9));

    BUTTERFLY_3_5_5 : BUTTERFLY PORT MAP(
            Clock => Clock, Start => Stard_Done(3),
            Ar => Val_R(3)(10), Ai => Val_I(3)(10), Br => Val_R(3)(11), Bi => Val_I(3)(11),
            Wr => Wr(5), Wi => Wi(5),
            A1r => Val_R(4)(10), A1i => Val_I(4)(10), B1r => Val_R(4)(11), B1i => Val_I(4)(11));

    BUTTERFLY_3_6_6 : BUTTERFLY PORT MAP(
            Clock => Clock, Start => Stard_Done(3),
            Ar => Val_R(3)(12), Ai => Val_I(3)(12), Br => Val_R(3)(13), Bi => Val_I(3)(13),
            Wr => Wr(3), Wi => Wi(3),
            A1r => Val_R(4)(12), A1i => Val_I(4)(12), B1r => Val_R(4)(13), B1i => Val_I(4)(13));

    BUTTERFLY_3_7_7 : BUTTERFLY PORT MAP(
            Clock => Clock, Start => Stard_Done(3),
            Ar => Val_R(3)(14), Ai => Val_I(3)(14), Br => Val_R(3)(15), Bi => Val_I(3)(15),
            Wr => Wr(7), Wi => Wi(7),
            A1r => Val_R(4)(14), A1i => Val_I(4)(14), B1r => Val_R(4)(15), B1i => Val_I(4)(15));

END ARCHITECTURE;
```

Butterfly

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;

-----------------------------------------------
-- Butterfly a 20 bit
-----------------------------------------------

ENTITY BUTTERFLY IS
        PORT (
          Clock, Start : IN STD_LOGIC;
          Ar, Ai, Br, Bi, Wr, Wi : IN sfixed(0 DOWNTO -19);
          Done : OUT STD_LOGIC;
          A1r, A1i, B1r, B1i : OUT sfixed(0 DOWNTO -19)
        );
END BUTTERFLY;

ARCHITECTURE Behavior OF BUTTERFLY IS

  COMPONENT DP IS
        PORT (
                Clock : IN STD_LOGIC;
                En_Reg_Ar, En_Reg_Ai, En_Reg_Br, En_Reg_Bi, En_Reg_A, En_Reg_B, En_Reg_C, En_Reg_1;
                En_Reg_2, MUL_SH : IN STD_LOGIC;
                Sel_Mux_1, Sel_Mux_3, Sel_Mux_4, Sel_Mux_5, Sel_Mux_7, Sel_Mux_10 : IN STD_LOGIC;
                Sel_Mux_2, Sel_Mux_6, Sel_Mux_8, Sel_Mux_9 : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                Start, Reset_Shift_Start, Enable_Shift_Start : IN STD_LOGIC;
                Out_Shift_Start_1 : BUFFER STD_LOGIC;
                Out_Shift_Start_2 : OUT STD_LOGIC;
                Wr, Wi, Ar, Ai, Br, Bi : IN sfixed(0 DOWNTO -19);
                Ar_1, Ai_1, Br_1, Bi_1 : OUT sfixed(0 DOWNTO -19)
                );
        END COMPONENT;

        COMPONENT CU IS
          PORT (
                  Clk : IN STD_LOGIC;
                  Cir : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                  Start : IN STD_LOGIC;
                  Commands : OUT STD_LOGIC_VECTOR (26 DOWNTO 0)
                );
        END COMPONENT;

        SIGNAL Command_i : STD_LOGIC_VECTOR(26 DOWNTO 0); --Comandi control unit
        SIGNAL Shift_Start_i : STD_LOGIC_VECTOR(1 DOWNTO 0); --Uscita shift register start
```

```
BEGIN

    DATAPATH : DP PORT MAP(
          Clock => Clock, En_Reg_Ar => Command_i(26), En_Reg_Ai => Command_i(25),
          En_Reg_Br => Command_i(24), En_Reg_Bi => Command_i(23), En_Reg_A => Command_i(22),
          En_Reg_B => Command_i(21), En_Reg_C => Command_i(20), En_Reg_1 => Command_i(19),
          En_Reg_2 => Command_i(18), MUL_SH => Command_i(3), Sel_Mux_1 => Command_i(17),
          Sel_Mux_2 => Command_i(16 DOWNTO 15), Sel_Mux_3 => Command_i(14), Sel_Mux_4 => Command_i(13),
          Sel_Mux_5 => Command_i(12), Sel_Mux_6 => Command_i(11 DOWNTO 10), Sel_Mux_7 => Command_i(9),
          Sel_Mux_8 => Command_i(8 DOWNTO 7), Sel_Mux_9 => Command_i(6 DOWNTO 5),
          Sel_Mux_10 => Command_i(4),Start => Start, Reset_Shift_Start => Command_i(0),
          Enable_Shift_Start => Command_i(1), Out_Shift_Start_1 => Shift_Start_i(0),
          Out_Shift_Start_2 => Shift_Start_i(1), Wr => Wr, Wi => Wi, Ar => Ar, Ai => Ai, Br => Br,
          Bi => Bi, Ar_1 => A1r, Ai_1 => A1i, Br_1 => B1r, Bi_1 => B1i);

          C_U : CU PORT MAP(Clk => Clock, Cir => Shift_Start_i, Start => Start, Commands => Command_i);

        Done <= Command_i(2);

END ARCHITECTURE;
```

Control Unit

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

----------------------------------------------------
-- Control Unit
----------------------------------------------------

ENTITY CU IS
    PORT (
        Clk : IN STD_LOGIC;
        Cir : IN STD_LOGIC_VECTOR(1 DOWNTO 0); -- bit dei FF
        Start : IN STD_LOGIC;
        Commands : OUT STD_LOGIC_VECTOR (26 DOWNTO 0) -- bit di comando del DP
    );
END CU;

ARCHITECTURE Behaviuor OF CU IS

    COMPONENT ROM IS
        PORT (
            Adr : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
            Dout : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT uIR IS
        GENERIC (N : INTEGER := 31); --Numero di bit
        PORT (
            R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
            Clock : IN STD_LOGIC;
            Q : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT PLA IS
        PORT (
            Jump, Cir : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
            Start : IN STD_LOGIC;
            Sel : OUT STD_LOGIC;
            Address : OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
        );
    END COMPONENT;
```

```
    COMPONENT MUX2to1_Nbit IS
        GENERIC (N : INTEGER := 3); --Numero di bit
        PORT (
            X, Y : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
            S : IN STD_LOGIC;
            M : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT uAR IS
        GENERIC (N : INTEGER := 3); --Numero di bit
        PORT (
            R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
            Clock : IN STD_LOGIC;
            Q : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0)
        );
    END COMPONENT;

    SIGNAL Code_i : STD_LOGIC_VECTOR (31 DOWNTO 0);
    SIGNAL Adr_i, NextAdr_i, PlaAdr_i, MuxAdr_i : STD_LOGIC_VECTOR (2 DOWNTO 0);
    SIGNAL Jump_i : STD_LOGIC_VECTOR(1 DOWNTO 0);
    SIGNAL Sel_i : STD_LOGIC;

BEGIN

    ROM_CU : ROM PORT MAP(
        Adr => Adr_i, Dout => Code_i);

    uIR_CU : uIR GENERIC MAP(N => 32)
    PORT MAP(
        R => Code_i, Clock => Clk, Q(31 DOWNTO 29) => NextAdr_i,
        Q(28 DOWNTO 27) => Jump_i, Q(26 DOWNTO 0) => Commands);

    PLA_CU : PLA PORT MAP(
        Jump => Jump_i, Cir => Cir, Start => Start, Sel => Sel_i, Address => PlaAdr_i);

    MUX_CU : MUX2to1_Nbit GENERIC MAP(N => 3)
    PORT MAP(X => NextAdr_i, Y => PlaAdr_i, S => Sel_i, M => MuxAdr_i);

    uAR_CU : uAR GENERIC MAP(N => 3)
    PORT MAP(R => MuxAdr_i, Clock => Clk, Q => Adr_i);

END ARCHITECTURE;
```

Componenti

9.2.1.1 ROM

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

----------------------------
---ROM
----------------------------

ENTITY ROM IS
    PORT (
        Adr : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
        Dout : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END ROM;

ARCHITECTURE rtl OF ROM IS

    TYPE Mem IS ARRAY (0 TO 7) OF STD_LOGIC_VECTOR(31 DOWNTO 0);
    CONSTANT Rom_7 : Mem := (

        --la prima riga indica la suddivsione dei bit e serve a comprendere meglio il loro compito

        --adr  jump    register      mux               done   cir  c_clr
        --0 IDLE
        "000" & "01" & "111111101" & "000001101000011" & "0" & "1" & "0",
        --1
        "010" & "00" & "000001010" & "111000000001000" & "0" & "0" & "0",
        --2
        "011" & "00" & "000000101" & "010100000010100" & "0" & "0" & "0",
        --3
        "100" & "00" & "001010100" & "011000000000000" & "0" & "0" & "0",
        --4
        "101" & "10" & "000111000" & "110000010000000" & "0" & "0" & "0",
        --5
        "000" & "00" & "000010010" & "001010001100011" & "0" & "0" & "0",
        --6 DONE è uguale allo stato 5 tranne per il done
        "000" & "00" & "000010010" & "001010001100011" & "1" & "0" & "0",
        --7 RESET
        "000" & "00" & "000000000" & "000000000000000" & "0" & "0" & "1"
    );

BEGIN
```

```
    memory : PROCESS (Adr)
    BEGIN
        Dout <= Rom_7(to_integer(unsigned(Adr)));
    END PROCESS memory;

END ARCHITECTURE;
```

9.2.1.2 μIR

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

-------------------------------------------------
-- Registro senza reset
-- campiona sul fronte di discesa del clock
-------------------------------------------------

ENTITY uIR IS
    GENERIC (N : INTEGER := 32); --Numero di bit
    PORT (
        R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        Clock : IN STD_LOGIC;
        Q : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0)
    );
END uIR;

ARCHITECTURE Behavior OF uIR IS

    -- inizializzo il primo valore per avere lo stato di reset iniziale
    SIGNAL Data : STD_LOGIC_VECTOR(N - 1 DOWNTO 0) := "11100000000000000000000000000000";

BEGIN

    Register_process : PROCESS (Clock)

    BEGIN

        IF (Clock'EVENT AND Clock = '0') THEN --fronte di discesa

            Data <= R; --Campionamento dell'ingresso

        END IF;

    END PROCESS;

    Q <= Data;

END Behavior;
```

9.2.1.3 PLA

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

-------------------------------------------------
-- PLA
-- a seconda dei bit ricevuti dal DP e dai bit di jump
-- decide se fare o meno il salto di indirizzo
-- in caso di salto cambia il bit di selezione del mux
-------------------------------------------------

ENTITY PLA IS
    PORT (
        Jump, Cir : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        Start : IN STD_LOGIC;
        Sel : OUT STD_LOGIC;
        Address : OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
    );
END PLA;

ARCHITECTURE rtl OF PLA IS

BEGIN

    PLA : PROCESS (Jump, Cir, Start)

    BEGIN

        IF (Jump = "01" AND(Start = '1' OR Cir(0) = '1')) THEN
            -- salto dallo stato 0 ad 1
            Sel <= '1';
            Address <= "001";
        ELSIF (Jump = "10" AND Cir(1) = '1') THEN
            -- salto dallo stato 4 a 6 in caso in cui devo dare il done
            Sel <= '1';
            Address <= "110";
        ELSE
            -- non ho un jump e quindi l'indirizzo successivo è quello contenuto nella ROM
            Sel <= '0';
        END IF;

    END PROCESS;

END ARCHITECTURE;
```

9.2.1.4 MUX 2 to 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-------------------------------------------------
-- Multiplexer a due ingressi con numero
-- di bit impostabile parametricamente
-------------------------------------------------

ENTITY MUX2to1_Nbit IS
    GENERIC (N : INTEGER := 3); --Numero di bit
    PORT (
        X, Y : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        S : IN STD_LOGIC;
        M : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0)
    );
END MUX2to1_Nbit;

ARCHITECTURE Behavior OF MUX2to1_Nbit IS

    COMPONENT MUX2to1_1bit IS
        PORT (
            X, Y, S : IN STD_LOGIC;
            M : OUT STD_LOGIC
        );
    END COMPONENT;

BEGIN

    WITH S SELECT
        M <= X WHEN '0',
        Y WHEN OTHERS;

END ARCHITECTURE;
```

9.2.1.5 μAR

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

-------------------------------------------------
-- Registro senza reset con primo valore 111
-- campiona sul fronte di salita del clock
-------------------------------------------------

ENTITY uAR IS
    GENERIC (N : INTEGER := 3); --Numero di bit
    PORT (
        R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        Clock : IN STD_LOGIC;
        Q : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0)
    );
END uAR;

ARCHITECTURE Behavior OF uAR IS

    -- inizializzo il primo valore per avere lo stato di reset iniziale
    SIGNAL Data : STD_LOGIC_VECTOR(N - 1 DOWNTO 0) := "111";

BEGIN

    Register_process : PROCESS (Clock)

    BEGIN

        IF (Clock'EVENT AND Clock = '1') THEN

            Data <= R; --Campionamento dell'ingresso

        END IF;

    END PROCESS;

    Q <= Data;

END Behavior;
```

Data Path

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;

ENTITY DP IS
        PORT (
                Clock : IN STD_LOGIC;
                En_Reg_Ar, En_Reg_Ai, En_Reg_Br, En_Reg_Bi, En_Reg_A, En_Reg_B, En_Reg_C, En_Reg_1,
                En_Reg_2, MUL_SH : IN STD_LOGIC;
                Sel_Mux_1, Sel_Mux_3, Sel_Mux_4, Sel_Mux_5, Sel_Mux_7, Sel_Mux_10 : IN STD_LOGIC;
                Sel_Mux_2, Sel_Mux_6, Sel_Mux_8, Sel_Mux_9 : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                Start, Reset_Shift_Start, Enable_Shift_Start : IN STD_LOGIC;
                Out_Shift_Start_1 : BUFFER STD_LOGIC;
                Out_Shift_Start_2 : OUT STD_LOGIC;
                Wr, Wi, Ar, Ai, Br, Bi : IN sfixed(0 DOWNTO -19);
                Ar_1, Ai_1, Br_1, Bi_1 : OUT sfixed(0 DOWNTO -19)
        );
END DP;

ARCHITECTURE Behavior OF DP IS

        COMPONENT FFT_Multiplier IS
                PORT (
                        Multiplier_In1, Multiplier_In2 : IN sfixed(0 DOWNTO -19);
                        XnSH, CLK : IN STD_LOGIC;
                        Multiplier_Out : OUT sfixed(1 DOWNTO -38);
                        SHout : OUT sfixed(0 DOWNTO -19));
        END COMPONENT;

        COMPONENT FFT_Adder IS
                PORT (
                        Adder_In1, Adder_In2 : IN sfixed(0 DOWNTO -38);
                        Adder_Out : OUT sfixed(1 DOWNTO -38));
        END COMPONENT;

        COMPONENT FFT_Subtractor IS
                PORT (
                        Subtractor_In1, Subtractor_In2 : IN sfixed(0 DOWNTO -38);
                        Subtractor_Out : OUT sfixed(1 DOWNTO -38);
                        CLK : IN STD_LOGIC);
        END COMPONENT;

        COMPONENT MUX2to1_Nbit_sf IS
                GENERIC (N : INTEGER := 1); --Numero di bit
```

```
        PORT (
                X, Y : IN sfixed(0 DOWNTO -N + 1);
                S : IN STD_LOGIC;
                M : OUT sfixed(0 DOWNTO -N + 1)
        );
END COMPONENT;

COMPONENT MUX4to1_Nbit IS
        GENERIC (N : INTEGER := 1); --Numero di bit
        PORT (
                X, Y, Z, W : IN sfixed(0 DOWNTO -N + 1);
                S : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                M : OUT sfixed(0 DOWNTO -N + 1)
        );
END COMPONENT;

COMPONENT REG IS
        GENERIC (N : INTEGER := 8); --Numero di bit
        PORT (
                R : IN sfixed(0 DOWNTO -N + 1);
                Clock, Clear, Enable : IN STD_LOGIC;
                Q : OUT sfixed(0 DOWNTO -N + 1)
        );
END COMPONENT;

COMPONENT FFT_RoundingHU IS
        PORT (
                Rounding_In : IN sfixed(0 DOWNTO -38);
                Rounding_Out : OUT sfixed(0 DOWNTO -19));

END COMPONENT;

COMPONENT FF_D IS
        PORT (
                R : IN STD_LOGIC;
                Clock, Clear, Enable : IN STD_LOGIC;
                Q : OUT STD_LOGIC
        );
END COMPONENT;

SIGNAL Out_Reg_Ar, Out_Reg_Ai, Out_Reg_Br, Out_Reg_Bi, Out_Reg_1,
       Out_Reg_2 : sfixed(0 DOWNTO -19); --Uscite registri a 20 bit
SIGNAL Out_Reg_A, Out_Reg_B, Out_Reg_C : sfixed(0 DOWNTO -38); --Uscite registri a 39 bit
SIGNAL Out_Mux_1, Out_Mux_2, Out_Mux_4, Out_Mux_5, Out_Mux_7,
       Out_Mux_10 : sfixed(0 DOWNTO -19); --Uscite multiplexer a 20 bit
SIGNAL Out_Mux_3, Out_Mux_6, Out_Mux_8, Out_Mux_9 : sfixed(0 DOWNTO -38); --Uscite mux a 39 bit
SIGNAL Out_Multiplier : sfixed(1 DOWNTO -38); --Uscite moltiplicatore
SIGNAL Out_Shifter, Out_Rounder : sfixed(0 DOWNTO -19); --Uscite shift moltiplicatore e rounder
SIGNAL Out_Adder, Out_Subtractor : sfixed(1 DOWNTO -38); --Uscite sommatore e sottrattore
SIGNAL Extended_Ar, Extended_Ai, Extended_Reg_1,
```

```
        Extended_Reg_2 : sfixed(0 DOWNTO -38); --Uscite dei registri estese su 39 bit
    SIGNAL Scaled_Out_Rounder : sfixed(0 DOWNTO -19); --Uscita del rounder scalata di un bit

BEGIN

    REG_AR : REG GENERIC MAP(N => 20) --Registro di ingresso usato per campionare Ar
    PORT MAP(Clock => Clock, Clear => '0', R => Ar, Enable => En_Reg_Ar, Q => Out_Reg_Ar);
    --Estensione su 39 bit dell'uscita del registro
    Extended_Ar(0 DOWNTO -19) <= Out_Reg_Ar;
    Extended_Ar(-20 DOWNTO -38) <= (OTHERS => '0');

    REG_AI : REG GENERIC MAP(N => 20) --Registro di ingresso usato per campionare Ai
    PORT MAP(Clock => Clock, Clear => '0', R => Ai, Enable => En_Reg_Ai, Q => Out_Reg_Ai);
    --Estensione su 39 bit dell'uscita del registro
    Extended_Ai(0 DOWNTO -19) <= Out_Reg_Ai;
    Extended_Ai(-20 DOWNTO -38) <= (OTHERS => '0');

    REG_BR : REG GENERIC MAP(N => 20) --Registro usato per campionare l'ngresso Br e l'uscita B'r
    PORT MAP(Clock => Clock, Clear => '0', R => Out_Mux_7, Enable => En_Reg_Br, Q => Out_Reg_Br);

    REG_BI : REG GENERIC MAP(N => 20) --Registro usato per campionare l'ngresso Bi e l'uscita B'i
    PORT MAP(Clock => Clock, Clear => '0', R => Out_Mux_10, Enable => En_Reg_Bi, Q => Out_Reg_Bi);

    REG_1 : REG GENERIC MAP(N => 20) --Registro temporaneo usato per campionare uscita shift e rounder
    PORT MAP(Clock => Clock, Clear => '0', R => Out_Mux_4, Enable => En_Reg_1, Q => Out_Reg_1);
    --Estensione su 39 bit dell'uscita del registro
    Extended_Reg_1(0 DOWNTO -19) <= Out_Reg_1;
    Extended_Reg_1(-20 DOWNTO -38) <= (OTHERS => '0');

    REG_2 : REG GENERIC MAP(N => 20) --Registro temporaneo usato per campionare uscita shift e rounder
    PORT MAP(Clock => Clock, Clear => '0', R => Out_Mux_5, Enable => En_Reg_2, Q => Out_Reg_2);
    --Estensione su 39 bit dell'uscita del registro
    Extended_Reg_2(0 DOWNTO -19) <= Out_Reg_2;
    Extended_Reg_2(-20 DOWNTO -38) <= (OTHERS => '0');

    REG_A : REG GENERIC MAP(N => 39) --Registro temporaneo usato per campionare uscita moltiplicatore
    PORT MAP(Clock => Clock, Clear => '0', R => Out_Multiplier(0 DOWNTO -38),
             Enable => En_Reg_A, Q => Out_Reg_A);

    REG_B : REG GENERIC MAP(N => 39) --Registro temporaneo usato per campionare uscita sommatore
    PORT MAP(Clock => Clock, Clear => '0', R => Out_Adder(0 DOWNTO -38),
             Enable => En_Reg_B, Q => Out_Reg_B);

    REG_C : REG GENERIC MAP(N => 39) --Registro temporaneo usato per campionare uscita sottrattore
    PORT MAP(Clock => Clock, Clear => '0', R => Out_Subtractor(0 DOWNTO -38),
             Enable => En_Reg_C, Q => Out_Reg_C);

    MUX_1 : MUX2to1_Nbit_sf GENERIC MAP(N => 20) --Multiplexer selezione ingresso moltiplicatore
    PORT MAP(X => Wi, Y => Wr, S => Sel_Mux_1, M => Out_Mux_1);
```

```
MUX_2 : MUX4to1_Nbit GENERIC MAP(N => 20) --Multiplexer selezione ingresso moltiplicatore
PORT MAP(
        X => Out_Reg_Ai, Y => Out_Reg_Ar, Z => Out_Reg_Bi, W => Out_Reg_Br,
        S => Sel_Mux_2, M => Out_Mux_2);

MUX_3 : MUX2to1_Nbit_sf GENERIC MAP(N => 39) --Multiplexer selezione ingresso rounder
PORT MAP(X => Out_Reg_C, Y => Out_Reg_B, S => Sel_Mux_3, M => Out_Mux_3);

MUX_4 : MUX2to1_Nbit_sf GENERIC MAP(N => 20) --Multiplexer selezione ingresso registro 1
PORT MAP(X => Scaled_Out_Rounder, Y => Out_Shifter, S => Sel_Mux_4, M => Out_Mux_4);

MUX_5 : MUX2to1_Nbit_sf GENERIC MAP(N => 20) --Multiplexer selezione ingresso registro 2
PORT MAP(X => Scaled_Out_Rounder, Y => Out_Shifter, S => Sel_Mux_5, M => Out_Mux_5);

MUX_6 : MUX4to1_Nbit GENERIC MAP(N => 39) --Multiplexer selezione ingresso sommatore
PORT MAP(
        X => Out_Reg_B, Y => Extended_Ar, Z => Extended_Ai, W => (OTHERS => '0'),
        S => Sel_Mux_6, M => Out_Mux_6);

MUX_7 : MUX2to1_Nbit_sf GENERIC MAP(N => 20) --Multiplexer selezione ingresso registro Br
PORT MAP(X => Scaled_Out_Rounder, Y => Br, S => Sel_Mux_7, M => Out_Mux_7);

MUX_8 : MUX4to1_Nbit GENERIC MAP(N => 39) --Multiplexer selezione ingresso sottrattore
PORT MAP(
        X => Out_Reg_C, Y => Out_Reg_B, Z => Out_Reg_A, W => (OTHERS => '0'),
        S => Sel_Mux_8, M => Out_Mux_8);

MUX_9 : MUX4to1_Nbit GENERIC MAP(N => 39) --Multiplexer selezione ingresso sottrattore
PORT MAP(
        X => Out_Reg_B, Y => Extended_Reg_2, Z => Extended_Reg_1, W => (OTHERS => '0'),
        S => Sel_Mux_9, M => Out_Mux_9);

MUX_10 : MUX2to1_Nbit_sf GENERIC MAP(N => 20) --Multiplexer selezione ingresso registro Bi
PORT MAP(X => Scaled_Out_Rounder, Y => Bi, S => Sel_Mux_10, M => Out_Mux_10);

MULTIPLIER : FFT_Multiplier --Moltiplicatore a 20 bit
PORT MAP(
        Multiplier_In1 => Out_Mux_1, Multiplier_In2 => Out_Mux_2,
        XnSH => MUL_SH, CLK => Clock, Multiplier_Out => Out_Multiplier, SHout => Out_Shifter);

ADDER : FFT_Adder --Sommatore a 39 bit
PORT MAP(Adder_In1 => Out_Reg_A, Adder_In2 => Out_Mux_6, Adder_Out => Out_Adder);

SUBTRACTOR : FFT_Subtractor --Sottrattore a 39 bit
PORT MAP(Subtractor_In1 => Out_Mux_9, Subtractor_In2 => Out_Mux_8,
         Subtractor_Out => Out_Subtractor, CLK => Clock);

SHIFT_START_1 : FF_D --FF usato per tenere conto dei segnali di start ricevuti
PORT MAP(Clock => Clock, Clear => Reset_Shift_Start, R => Start, Enable => Enable_Shift_Start,
         Q => Out_Shift_Start_1);
```

```
        SHIFT_START_2 : FF_D --FF usato per tenere conto dei segnali di start ricevuti
        PORT MAP(Clock => Clock, Clear => Reset_Shift_Start, R => Out_Shift_Start_1,
                 Enable => Enable_Shift_Start, Q => Out_Shift_Start_2);

        ROUNDER : FFT_RoundingHU --Half up rounder da 39 a 20 bit
        PORT MAP(Rounding_In => Out_Mux_3, Rounding_Out => Out_Rounder);

        Scaled_Out_Rounder <= Out_Rounder(0) & Out_Rounder(0 DOWNTO -18); --Scalamento uscita rounder

        --Assegnazioni dei segnali di uscita
        Ar_1 <= Out_Reg_1;
        Ai_1 <= Out_Reg_2;
        Br_1 <= Out_Reg_Br;
        Bi_1 <= Out_Reg_Bi;

END ARCHITECTURE;
```

Componenti

9.3.1.1 Moltiplicatore

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;
USE ieee.numeric_std.ALL;

ENTITY FFT_Multiplier IS
        PORT (
                Multiplier_In1, Multiplier_In2 : IN sfixed(0 DOWNTO -19);
                XnSH, CLK : IN STD_LOGIC;
                Multiplier_Out : OUT sfixed(1 DOWNTO -38);
                SHout : OUT sfixed(0 DOWNTO -19));

END FFT_Multiplier;

ARCHITECTURE Behavior OF FFT_Multiplier IS

        COMPONENT PIPE IS
                GENERIC (N : INTEGER := 39);
                PORT (
                        D : IN sfixed (1 DOWNTO - (N - 1));
                        CLK : IN STD_LOGIC;
                        Q : OUT sfixed (1 DOWNTO - (N - 1)));
        END COMPONENT;

        SIGNAL A, B : sfixed(0 DOWNTO -19);
        SIGNAL Xout, Q1, Q2 : sfixed(1 DOWNTO -38);

BEGIN

        A <= Multiplier_In1;
        B <= Multiplier_In2;
        PROCESS (A, B, XnSH)

        BEGIN
                IF (XnSH = '0') THEN

                        Xout <= (A * B);

                ELSE

                        SHout <= shift_left(sfixed(B), 1);

                END IF;
```

```
        END PROCESS;

        livello1 : PIPE
        GENERIC MAP(N => 39)
        PORT MAP(D => Xout, CLK => CLK, Q => Q1);

        livello2 : PIPE
        GENERIC MAP(N => 39)
        PORT MAP(D => Q1, CLK => CLK, Q => Multiplier_Out);

END Behavior;
```

9.3.1.2 Sommatore

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;
--LIBRARY work;
--USE work.fixed_pkg.ALL;

ENTITY FFT_Adder IS
        PORT (
                Adder_In1, Adder_In2 : IN sfixed(0 DOWNTO -38);
                Adder_Out : OUT sfixed(1 DOWNTO -38));
END FFT_Adder;

ARCHITECTURE Behavior OF FFT_Adder IS
BEGIN

        PROCESS (Adder_In1, Adder_In2)

        BEGIN

                Adder_Out <= (Adder_In1 + Adder_In2);

        END PROCESS;

END Behavior;
```

9.3.1.3 MUX 4 to 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;

-------------------------------------------------
-- Multiplexer fixed point a quattro ingressi con numero
-- di bit impostabile parametricamente
-------------------------------------------------

ENTITY MUX4to1_Nbit IS
    GENERIC (N : INTEGER := 1); --Numero di bit
    PORT (
        X, Y, Z, W : IN sfixed(0 DOWNTO -N + 1);
        S : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        M : OUT sfixed(0 DOWNTO -N + 1)
    );
END MUX4to1_Nbit;

ARCHITECTURE Behavior OF MUX4to1_Nbit IS

BEGIN

    WITH S SELECT
        M <= X WHEN "00",
        Y WHEN "01",
        Z WHEN "10",
        W WHEN OTHERS;

END ARCHITECTURE;
```

9.3.1.4 Sottrattore

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;
--LIBRARY work;
--USE work.fixed_pkg.ALL;

ENTITY FFT_Subtractor IS
        PORT (
                Subtractor_In1, Subtractor_In2 : IN sfixed(0 DOWNTO -38);
                Subtractor_Out : OUT sfixed(1 DOWNTO -38);
                CLK : IN STD_LOGIC);
END FFT_Subtractor;

ARCHITECTURE Behavior OF FFT_Subtractor IS

        COMPONENT PIPE IS
                GENERIC (N : INTEGER := 20);
                PORT (
                        D : IN sfixed (1 DOWNTO - (N - 1));
                        CLK : IN STD_LOGIC;
                        Q : OUT sfixed (1 DOWNTO - (N - 1)));

        END COMPONENT;

        SIGNAL SubOut, Q1 : sfixed(1 DOWNTO -38);

BEGIN

        PROCESS (Subtractor_In1, Subtractor_In2)

        BEGIN

                SubOut <= (Subtractor_In1 - Subtractor_In2);

        END PROCESS;

        livello1 : PIPE
        GENERIC MAP(N => 39)
        PORT MAP(D => SubOut, CLK => CLK, Q => Subtractor_Out);

END Behavior;
```

9.3.1.5 MUX 2 to 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;

-------------------------------------------------
-- Multiplexer fixed point a due ingressi con numero
-- di bit impostabile parametricamente
-------------------------------------------------

ENTITY MUX2to1_Nbit_sf IS
    GENERIC (N : INTEGER := 1); --Numero di bit
    PORT (
        X, Y : IN sfixed(0 DOWNTO -N + 1);
        S : IN STD_LOGIC;
        M : OUT sfixed(0 DOWNTO -N + 1)
    );
END MUX2to1_Nbit_sf;

ARCHITECTURE Behavior OF MUX2to1_Nbit_sf IS

    COMPONENT MUX2to1_1bit IS
        PORT (
            X, Y, S : IN STD_LOGIC;
            M : OUT STD_LOGIC
        );
    END COMPONENT;

BEGIN

    WITH S SELECT
        M <= X WHEN '0',
        Y WHEN OTHERS;

END ARCHITECTURE;
```

9.3.1.6 Registro

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;

-------------------------------------------------
-- Registro fixed point con numero di bit parametrico
-------------------------------------------------

ENTITY REG IS
        GENERIC (N : INTEGER := 8); --Numero di bit
        PORT (
                R : IN sfixed(0 DOWNTO -N + 1);
                Clock, Clear, Enable : IN STD_LOGIC;
                Q : OUT sfixed(0 DOWNTO -N + 1)
        );
END REG;

ARCHITECTURE Behavior OF REG IS

BEGIN

        Register_process : PROCESS (Clock)

        BEGIN

                IF (Clock'EVENT AND Clock = '1') THEN

                        IF (Clear = '1') THEN
                                Q <= (OTHERS => '0'); --Reset del valore memorizzato

                        ELSIF (Enable = '1') THEN
                                Q <= R; --Campionamento dell'ingresso

                        END IF;

                END IF;

        END PROCESS;

END Behavior;
```

9.3.1.7 Arrotondatore

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;
USE ieee.numeric_std.ALL;

ENTITY FFT_RoundingHU IS
        PORT (--CLK : in std_logic;
                Rounding_In : IN sfixed(0 DOWNTO -38);
                Rounding_Out : OUT sfixed(0 DOWNTO -19));

END FFT_RoundingHU;

ARCHITECTURE Behavior OF FFT_RoundingHU IS

        SIGNAL Res, one : sfixed(0 DOWNTO -19);
        SIGNAL Sum : sfixed(1 DOWNTO -19);

BEGIN

        Res <= Rounding_In(0 DOWNTO -19);
        one <= "00000000000000000001";
        Sum <= Res + one;

        Rounding_Out <= Sum(0 DOWNTO -19);
END Behavior;
```

9.3.1.8 Registro di PIPE

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

entity PIPE is
generic  (N : integer := 20);
        port (  D : in sfixed ( 1 downto -(N-1));
                CLK : in std_logic;
                Q : out sfixed (1 downto -(N-1)));
end PIPE;

architecture Behavior of PIPE is
begin

  process (CLK)
  begin
    if CLK'event and CLK = '1' then
       Q <= D;
    end if;
  end process;
end Behavior;
```

9.3.1.9 Flip-Flop D

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-------------------------------------------------
-- Flip-flop di tipo D
-------------------------------------------------

ENTITY FF_D IS
        PORT (
                R : IN STD_LOGIC;
                Clock, Clear, Enable : IN STD_LOGIC;
                Q : OUT STD_LOGIC
        );
END FF_D;

ARCHITECTURE Behavior OF FF_D IS

BEGIN

        Register_process : PROCESS (Clock)

        BEGIN

                IF (Clock'EVENT AND Clock = '1') THEN

                        IF (Clear = '1') THEN
                                Q <= '0'; --Reset del valore memorizzato

                        ELSIF (Enable = '1') THEN
                                Q <= R; --Campionamento dell'ingresso

                        END IF;

                END IF;

        END PROCESS;

END Behavior;
```

Test Bench

9.4.0.1 Control Unit

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench_CU IS
END testbench_CU;

ARCHITECTURE arch OF testbench_CU IS

    COMPONENT CU IS
        PORT (
            Clk : IN STD_LOGIC;
            Cir : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
            Start : IN STD_LOGIC;
            Commands : OUT STD_LOGIC_VECTOR (24 DOWNTO 0)
        );
    END COMPONENT;

    SIGNAL Clk_t : STD_LOGIC := '1';
    SIGNAL Cir_t : STD_LOGIC_VECTOR(1 DOWNTO 0);
    SIGNAL Start_t : STD_LOGIC;
    SIGNAL Commands_t : STD_LOGIC_VECTOR (24 DOWNTO 0);

BEGIN

    Clk_t <= NOT Clk_t AFTER 50 ns;

    test : PROCESS
    BEGIN
        Cir_t <= "00";
        Start_t <= '0';
        WAIT FOR 1000 ns;

        Cir_t <= "00";
        Start_t <= '1';
        WAIT FOR 100 ns;

        Cir_t <= "01";
        Start_t <= '0';
        WAIT FOR 600 ns;

        Cir_t <= "10";
```

```
        Start_t <= '0';
        WAIT FOR 600 ns;

        Cir_t <= "00";
        Start_t <= '0';
        WAIT FOR 2000 ns;

        WAIT;
    END PROCESS;

    DUT : CU PORT MAP(
        Clk => Clk_t, Cir => Cir_t, Start => Start_t, Commands => Commands_t);

END ARCHITECTURE;
```

9.4.0.2 FFT

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;
USE work.types_pkg.ALL;

--------------------------------------------------
-- Testbench FFT
--------------------------------------------------

ENTITY testbench_FFT IS
END testbench_FFT;

ARCHITECTURE Behavior OF testbench_FFT IS

        COMPONENT FFT IS
                PORT (
                        Clock, Start : IN STD_LOGIC;
                        Inputs_r, Inputs_i : IN vector_of_16_sfixed;
                        Done : OUT STD_LOGIC;
                        Outputs_r, Outputs_i : OUT vector_of_16_sfixed
                );
        END COMPONENT;

        SIGNAL Clk_t : STD_LOGIC := '1';
        SIGNAL Start_t, Done_t : STD_LOGIC;
        SIGNAL Inputs_r_t, Inputs_i_t, Outputs_r_t, Outputs_i_t : vector_of_16_sfixed;

BEGIN

        Clk_t <= NOT Clk_t AFTER 50 ns;

        Inputs_i_t <= (OTHERS => to_sfixed(0, Inputs_i_t(0)));


        test : PROCESS
        BEGIN

                Inputs_r_t <= (--1
                        to_sfixed(-0.25, Inputs_r_t(0)), to_sfixed(-0.25, Inputs_r_t(0)),
                        to_sfixed(-0.25, Inputs_r_t(0)), to_sfixed(-0.25, Inputs_r_t(0)),
                        to_sfixed(-0.25, Inputs_r_t(0)), to_sfixed(-0.25, Inputs_r_t(0)),
                        to_sfixed(-0.25, Inputs_r_t(0)), to_sfixed(-0.25, Inputs_r_t(0)),
                        to_sfixed(-0.25, Inputs_r_t(0)), to_sfixed(-0.25, Inputs_r_t(0)),
                        to_sfixed(-0.25, Inputs_r_t(0)), to_sfixed(-0.25, Inputs_r_t(0)),
                        to_sfixed(-0.25, Inputs_r_t(0)), to_sfixed(-0.25, Inputs_r_t(0)),
                        to_sfixed(-0.25, Inputs_r_t(0)), to_sfixed(-0.25, Inputs_r_t(0)));
```

```
            Start_t <= '0';
            WAIT FOR 550 ns;

            Start_t <= '1';
            WAIT FOR 200 ns;

            Inputs_r_t <= (--3
                    to_sfixed(0.25, Inputs_r_t(0)), to_sfixed(0, Inputs_r_t(0)),
                    to_sfixed(0, Inputs_r_t(0)), to_sfixed(0, Inputs_r_t(0)),
                    to_sfixed(0, Inputs_r_t(0)), to_sfixed(0, Inputs_r_t(0)),
                    to_sfixed(0, Inputs_r_t(0)), to_sfixed(0, Inputs_r_t(0)),
                    to_sfixed(0, Inputs_r_t(0)), to_sfixed(0, Inputs_r_t(0)),
                    to_sfixed(0, Inputs_r_t(0)), to_sfixed(0, Inputs_r_t(0)),
                    to_sfixed(0, Inputs_r_t(0)), to_sfixed(0, Inputs_r_t(0)));
            WAIT FOR 600 ns;

            Inputs_r_t <= (--5
                    to_sfixed(0.125, Inputs_r_t(0)), to_sfixed(0.125, Inputs_r_t(0)),
                    to_sfixed(0.125, Inputs_r_t(0)), to_sfixed(0.125, Inputs_r_t(0)),
                    to_sfixed(0.125, Inputs_r_t(0)), to_sfixed(0.125, Inputs_r_t(0)),
                    to_sfixed(0.125, Inputs_r_t(0)), to_sfixed(0.125, Inputs_r_t(0)),
                    to_sfixed(0.125, Inputs_r_t(0)), to_sfixed(-0.125, Inputs_r_t(0)),
                    to_sfixed(-0.125, Inputs_r_t(0)), to_sfixed(-0.125, Inputs_r_t(0)),
                    to_sfixed(-0.125, Inputs_r_t(0)), to_sfixed(-0.125, Inputs_r_t(0)),
                    to_sfixed(-0.125, Inputs_r_t(0)), to_sfixed(-0.125, Inputs_r_t(0)));
            WAIT FOR 600 ns;

            Start_t <= '0';
            WAIT FOR 2 ms;

            WAIT;
        END PROCESS;

        DUT : FFT PORT MAP(
            Clock => Clk_t, Start => Start_t, Inputs_r => Inputs_r_t, Inputs_i => Inputs_i_t,
            Done => Done_t, Outputs_r => Outputs_r_t, Outputs_i => Outputs_i_t);

END ARCHITECTURE;
```

9.4.0.3 Butterfly

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;
USE STD.textio.ALL;
USE ieee.std_logic_textio.ALL;

------------------------------------------------
-- Testbench per provare la modalità di savoro non continua della butterfly
------------------------------------------------

ENTITY testbench_BUTTERFLY IS
END testbench_BUTTERFLY;

ARCHITECTURE Behavior OF testbench_BUTTERFLY IS

  COMPONENT BUTTERFLY IS
    PORT (
      Clock, Start : IN STD_LOGIC;
      Ar, Ai, Br, Bi, Wr, Wi : IN sfixed(0 DOWNTO -19);
      Done : OUT STD_LOGIC;
      A1r, A1i, B1r, B1i : OUT sfixed(0 DOWNTO -19)
    );
  END COMPONENT;

  SIGNAL Ar_t, Ai_t, Br_t, Bi_t, Wr_t, Wi_t, A1r_t, A1i_t, B1r_t, B1i_t : sfixed(0 DOWNTO -19);
  SIGNAL Clock_t : STD_LOGIC := '1';
  SIGNAL Start_t, Done_t : STD_LOGIC;

  FILE FileIn : text;
  FILE FileOut : text;

BEGIN

  DUT : BUTTERFLY
  PORT MAP(Clock => Clock_t, Start => Start_t, Done => Done_t, Ar => Ar_t, Ai => Ai_t, Br => Br_t,
           Bi => Bi_t, Wr => Wr_t, Wi => Wi_t, A1r => A1r_t, A1i => A1i_t, B1r => B1r_t, B1i => B1i_t);

  Clock_t <= NOT Clock_t AFTER 50 ns;

  --Processo di gestione dei file
  file_io : PROCESS

    --Variabili per lettura file
    VARIABLE InputLine : line;
    VARIABLE OutputLine : line;
    VARIABLE Ar_f, Ai_f, Br_f, Bi_f, Wr_f, Wi_f : sfixed(0 DOWNTO -19);
```

```
  VARIABLE Space : CHARACTER;

BEGIN

  --Apertura file di input e di output
  file_open(FileIn, "input_vectors.log", read_mode);
  file_open(FileOut, "output_results.log", write_mode);

  --Letura input dal file input_vectors.txt
  WHILE NOT endfile(FileIn) LOOP
    readline(FileIn, InputLine); --Lettura riga completa
    read(InputLine, Ar_f);
    read(InputLine, Space); --Lettura spazio
    read(InputLine, Ai_f);
    read(InputLine, Space); --Lettura spazio
    read(InputLine, Br_f);
    read(InputLine, Space); --Lettura spazio
    read(InputLine, Bi_f);
    read(InputLine, Space); --Lettura spazio
    read(InputLine, Wr_f);
    read(InputLine, Space); --Lettura spazio
    read(InputLine, Wi_f);
    read(InputLine, Space); --Lettura spazio

    --Passaggio delle variabili ai segnali della butterfly
    Ar_t <= Ar_f;
    Ai_t <= Ai_f;
    Br_t <= Br_f;
    Bi_t <= Bi_f;
    Wr_t <= Wr_f;
    Wi_t <= Wi_f;
    Start_t <= '0';
    WAIT FOR 500 ns; --Attesa reset

    Start_t <= '1';
    WAIT FOR 200 ns; --Attesa campionamento segnale di start

    Start_t <= '0';
    WAIT FOR 950 ns; --Attesa risultati

    --Scrittura risultati nel file output_results.txt
    write(OutputLine, A1r_t);
    write(OutputLine, STRING'(" "));
    write(OutputLine, A1i_t);
    write(OutputLine, STRING'(" "));
    write(OutputLine, B1r_t);
    write(OutputLine, STRING'(" "));
    write(OutputLine, B1i_t);
    write(OutputLine, STRING'(" "));
    writeline(FileOut, OutputLine);
```

```
    END LOOP;

    --Chiusura file di input e di output
    file_close(FileIn);
    file_close(FileOut);

    WAIT;

  END PROCESS;

END Behavior;
```

9.4.0.4 Butterfly Modalità Continua

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.fixed_pkg.ALL;
USE STD.textio.ALL;
USE ieee.std_logic_textio.ALL;

------------------------------------------------
-- Testbench per provare la modalità di savoro continua della butterfly
------------------------------------------------

ENTITY testbench_BUTTERFLY_MOD_CONTINUA IS
END testbench_BUTTERFLY_MOD_CONTINUA;

ARCHITECTURE Behavior OF testbench_BUTTERFLY_MOD_CONTINUA IS

  COMPONENT BUTTERFLY IS
    PORT (
      Clock, Start : IN STD_LOGIC;
      Ar, Ai, Br, Bi, Wr, Wi : IN sfixed(0 DOWNTO -19);
      Done : OUT STD_LOGIC;
      A1r, A1i, B1r, B1i : OUT sfixed(0 DOWNTO -19)
    );
  END COMPONENT;

  SIGNAL Ar_t, Ai_t, Br_t, Bi_t, Wr_t, Wi_t, A1r_t, A1i_t, B1r_t, B1i_t : sfixed(0 DOWNTO -19);
  SIGNAL Clock_t : STD_LOGIC := '1';
  SIGNAL Start_t, Done_t : STD_LOGIC;

  FILE FileIn : text;
  FILE FileOut : text;

BEGIN

  DUT : BUTTERFLY
  PORT MAP(Clock => Clock_t, Start => Start_t, Done => Done_t, Ar => Ar_t, Ai => Ai_t, Br => Br_t,
           Bi => Bi_t, Wr => Wr_t, Wi => Wi_t, A1r => A1r_t, A1i => A1i_t, B1r => B1r_t, B1i => B1i_t);

  Clock_t <= NOT Clock_t AFTER 50 ns;

  --Processo di gestione dei file
  file_io : PROCESS

    --Variabili per lettura file
    VARIABLE InputLine : line;
    VARIABLE OutputLine : line;
    VARIABLE Ar_f, Ai_f, Br_f, Bi_f, Wr_f, Wi_f : sfixed(0 DOWNTO -19);
```

```
  VARIABLE Space : CHARACTER;

BEGIN

  --Apertura file di input e di output
  file_open(FileIn, "input_vectors.log", read_mode);
  file_open(FileOut, "output_results.log", write_mode);

  --Invio e ricezione primi valori in modo singolo per gestire correttamente
  --i tempismi di avvio della macchina
  readline(FileIn, InputLine); --Lettura riga completa
  read(InputLine, Ar_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Ai_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Br_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Bi_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Wr_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Wi_f);
  read(InputLine, Space); --Lettura spazio

  --Passaggio delle variabili ai segnali della butterfly
  Ar_t <= Ar_f;
  Ai_t <= Ai_f;
  Br_t <= Br_f;
  Bi_t <= Bi_f;
  Wr_t <= Wr_f;
  Wi_t <= Wi_f;
  Start_t <= '0';
  WAIT FOR 450 ns; --Attesa reset

  Start_t <= '1';
  WAIT FOR 500 ns; --Attesa campionamento segnale di start

  readline(FileIn, InputLine); --Lettura riga completa
  read(InputLine, Ar_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Ai_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Br_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Bi_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Wr_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Wi_f);
  read(InputLine, Space); --Lettura spazio
```

```
--Passaggio delle variabili ai segnali della butterfly
Ar_t <= Ar_f;
Ai_t <= Ai_f;
Br_t <= Br_f;
Bi_t <= Bi_f;
Wr_t <= Wr_f;
Wi_t <= Wi_f;

WAIT FOR 600 ns; --Attesa risultati

--Invio e ricezione valori in modo continuo
WHILE NOT endfile(FileIn) LOOP

  readline(FileIn, InputLine); --Lettura riga completa
  read(InputLine, Ar_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Ai_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Br_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Bi_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Wr_f);
  read(InputLine, Space); --Lettura spazio
  read(InputLine, Wi_f);
  read(InputLine, Space); --Lettura spazio

  --Passaggio delle variabili ai segnali della butterfly
  Ar_t <= Ar_f;
  Ai_t <= Ai_f;
  Br_t <= Br_f;
  Bi_t <= Bi_f;
  Wr_t <= Wr_f;
  Wi_t <= Wi_f;

  --Scrittura risultati nel file output_results.txt
  write(OutputLine, A1r_t);
  write(OutputLine, STRING'(" "));
  write(OutputLine, A1i_t);
  write(OutputLine, STRING'(" "));
  write(OutputLine, B1r_t);
  write(OutputLine, STRING'(" "));

  write(OutputLine, B1i_t);
  write(OutputLine, STRING'(" "));
  writeline(FileOut, OutputLine);

  WAIT FOR 600 ns; --Attesa risultati
```

```
    END LOOP;

    --Chiusura file di input e di output
    file_close(FileIn);
    file_close(FileOut);

    WAIT;

  END PROCESS;

END Behavior;
```

Pyton

Simluzaione

9.5.1.1 Script

```
import os
import subprocess
import random
from math import floor



#---------------- DEFINIZIONE FUNZIONI -----------------
def genera_random_input(_din, _n): #Generazione sequenze di input casuali
        bit_guardia = 2
        for i in range(_n): #Generazione di n sequenze
                data = []
                for j in _din: #Generazione di n variabili
                        #possibili = [-2**(j-1-bit_guardia), 2**(j-1-bit_guardia) - 1]
                        #Input ai limiti della dinamica
                        #data.append(random.choice(possibili))
                        data.append(random.randint(-2**(j-1-bit_guardia), 2**(j-1-bit_guardia) - 1))
                        #Salvataggio dati per calcolo output
                        file_in.write(print_binary(data[-1], 20)) #Srittura sequenza sul file

                file_in.write("\n")
                genera_output(data) #Calcolo output previsto



def print_binary(_value, _bits): #Stampa del valore binario in formato sfixed
        str_format = '{0:0'+str(_bits)+'b}'
        if(_value < 0):
                _value = 2**_bits + _value
        to_return = str_format.format(_value)
        to_return = to_return[:1] + "." + to_return[1:] + " "

        return to_return



def genera_output(_data): #Calcola output previsto
        Ar = _data[0]
        Ai = _data[1]
        Br = _data[2]
```

```
        Bi = _data[3]
        Wr = _data[4] / 2**19
        Wi = _data[5] / 2**19

        M1 = Br*Wr
        M2 = Bi*Wi
        M3 = Br*Wi
        M4 = Bi*Wr
        M5 = 2*Ar
        M6 = 2*Ai

        S1 = Ar + M1
        S2 = S1 - M2
        S3 = Ai + M3
        S4 = S3 + M4
        S5 = M5 - S2
        S6 = M6 - S4

        A1r = S2
        A1i = S4
        B1r = S5
        B1i = S6

        A1r = floor((A1r + 1) / 2)
        A1i = floor((A1i + 1) / 2)
        B1r = floor((B1r + 1) / 2)
        B1i = floor((B1i + 1) / 2)

        file_out.write(print_binary(A1r, 20) + print_binary(A1i, 20) + print_binary(B1r, 20) +
                        + print_binary(B1i, 20) + "\n")



def verifica_output(): #Confronta l'output ottenuto con quello previsto

        simulation_output = open('output_results.log', 'r') #Apertura risultato della simulazione
        reference_output = open('output_vectors.log', 'r') #Apertura output atteso
        input_file = open('input_vectors.log', 'r') #Apertura sequenze di input
        inputs = input_file.readlines() #Lettura sequenza di input

        i = 0 #Conteggio linee lette

        for line in simulation_output: #Lettura risultato simulazione una riga alla volta
                line2 = reference_output.readline() #Lettura risultato atteso una riga alla volta
                if line != line2: #Verifica differenze tra i due outout
                        log_file.write("Input = " + inputs[i][:-1] + ', Output ottenuto = ' + line[:-1] +
                                + ", Output atteso = " + line2) #Segnala la differenza nel file di log

                i += 1 #Incremento contatore linea
```

```
        simulation_output.close()
        reference_output.close()
        input_file.close()



#---------------- INIZIO SCRIPT -----------------
try:

        Din = (20, 20, 20, 20, 22, 22) #Numero di bit delle variabili da generare

        file_in = open('input_vectors.log', 'w') #Apertura file per scrittura variabili di ingresso
        file_out = open('output_vectors.log', 'w') #Apertura file per scrittura dei risultati attesi


        genera_random_input(Din, 10000) #Generazione sequenze di input casuali

        file_in.close()
        file_out.close()

        process = subprocess.call(["vsim", "-c", "-do", "compile.do"]) #Avvio simulazione

        log_file = open('log.log', 'w')
        #Apertura file per scrittura differenze tra risultati attesi e ottenuti
        verifica_output() #Calcola output previsto
        log_file.close()

except:
   print("Errore")
   input("PREMERE ENTER PER USCIRE")
```
