

Politecnico di Torino
Scuola di Ingegneria e Architettura

Sistemi Digitali Integrati

Prof. Massimo Rou Roch
Prof. Maurizio Zamboni

Relazione FFT



**Politecnico
di Torino**

Federico Cobianchi - 332753
Onice Mazzi - 359754
Antonio Telmon - 353781

A.A. 2025/2026

Indice

1	Introduzione	1
2	Data Flow Diagram	2
2.1	Specifiche sui blocchi operazionali	2
2.2	Approccio ASAP	2
2.3	Approccio ALAP	3
2.4	Approccio scelto	4
2.5	Tempo di vita delle variabili	5
3	Datapath	6
3.1	ROM Rounding	7
4	Control Unit	9
4.1	Comandi e stati	9
4.2	Struttura dell'unità di controllo	9
4.2.1	Late Status PLA	9
4.2.2	uROM	9
4.2.3	datapath, uAR e uIR	10
5	Butterfly e FFT	11
5.1	Butterfly	11
5.2	FFT	11
6	Simulazioni	13
7	Appendice - File VHDL	20
7.1	Sommatore	20
7.2	MUX	20
7.2.1	MUX 2	20
7.2.2	MUX 3	21
7.3	Sottrattore	21
7.4	Moltiplicatore/Shifter	22
7.5	ROM rounding	23
7.5.1	Blocco ROM rounding	23
7.5.2	Test Bench del ROM rounding	25
7.5.3	ROM	26
7.5.4	Test Bench della ROM	28
7.6	Datapath	28
7.7	Control Unit	37
7.7.1	Datapath	37
7.7.2	ROM	40
7.7.3	PLA	47
7.7.4	Test Bench della Control Unit	48
7.8	FFT	49
7.8.1	Test Bench della Butterfly singola	58

1 Introduzione

La FFT (Fast Fourier Transform) è un'operazione fondamentale per tutti i sistemi di elaborazione dei segnali digitali. È utilizzata nelle telecomunicazioni, nell'elaborazione audio e nei sistemi embedded ad alte prestazioni. L'algoritmo FFT si basa sull'operatore butterfly, che è una struttura di manipolazione dei dati che esegue combinazioni lineari di dati complessi mediante somma, sottrazione e moltiplicazione con coefficienti anch'essi complessi.

Lo scopo di questo progetto è progettare un'unità di elaborazione dedicata per eseguire la FFT mediante la concatenazione di più stati composti dalla singola Butterfly. Si utilizzeranno tecniche di microprogrammazione e verranno considerati vincoli realistici dell'architettura hardware. Più specificamente, questo progetto si occupa della gestione di dati complessi in una rappresentazione frazionaria a complemento a due di 24 bit, dell'uso di un'aritmetica fixed point e dell'arrotondamento dei dati per tornare ad avere in uscita un parallelismo di 24 bit. Il lavoro include la derivazione del diagramma di flusso dei dati (DFD), l'ottimizzazione del datapath e dell'unità di controllo, la completa descrizione dell'architettura in VHDL e la verifica funzionale attraverso simulazioni. Infine, la Butterfly implementata deve essere utilizzata come blocco di base per l'implementazione e il collaudo di una FFT 16x16, che ne dimostra la validità e la scalabilità della soluzione.

Per creare la singola butterfly sono stati seguiti i seguenti passi:

- Creazione del Data Flow Diagram
- Stima del tempo di vita delle variabili
- Creazione del Datapath
- Creazione della Control Unit (CU)
- Test finali

Data la necessità di utilizzare diversi blocchi logici quali moltiplicatori, sommatori, sottrattori, registri e multiplexer sono state eseguite delle simulazioni intermedie rispetto ai punti appena descritti per facilitare il lavoro di debug. Si è proceduto nel modo descritto in quanto è da preferire rispetto ad un approccio "trial and error" dove tutti i blocchi non vengono testati e si procede solamente al test finale della Butterfly. Nel caso fosse stata scelta questa strategia progettuale sarebbe stato pressoché impossibile andare a trovare dove fosse l'errore nel caso si fosse verificato qualche malfunzionamento.

2 Data Flow Diagram

In questo capitolo si parlerà delle specifiche imposte sui blocchi logici. Si andranno a confrontare gli approcci "As Soon As Possible" ASAP e "As Late As Possible" ALAP. Infine verrà illustrato l'approccio che è stato utilizzato per ottimizzare le tempistiche dell'algoritmo.

2.1 Specifiche sui blocchi operazionali

In questo Progetto si supponeva di poter utilizzare per ciascuna Butterfly un unico blocco moltiplicatore. Questo blocco è in grado di poter svolgere sia la moltiplicazione tra due numeri in ingresso sia lo shift (moltiplicazione per 2). Le due operazioni sono selezionabili attraverso un segnale di controllo esterno al blocco operatore che verrà fornito dalla Contro Unit (CU). Si è supposto che il moltiplicatore avesse due livelli di pipeline, ovvero che il risultato fosse disponibile al registro di uscita dopo 3 colpi di clock. L'operazione di shift, al contrario, ha un singolo livello di pipeline, dunque l'uscita è disponibile dopo 2 colpi di clock. Il blocco moltiplicatore è stato rappresentato in *Fig. 1* con il blocco *verde* e l'operazione di shift è stata rappresentata con il blocco *viola*.

Si è supposto di avere a disposizione un singolo elemento per le operazioni di somma e uno per le sottrazioni. I blocchi sommatore e sottrattore hanno ciascuno un livello di pipeline e sono rappresentati rispettivamente dal blocco *rosso* e *blu*.

Al fine di rappresentare anche l'operazione di ROM Rounding presente alla fine dell'algoritmo è stato deciso di impiegare un colpo di clock per l'operazione di arrotondamento (blocco *azzurro*) e utilizzare successivamente un registro controllato esternamente per mantenere in vita le variabili di uscita della butterfly.

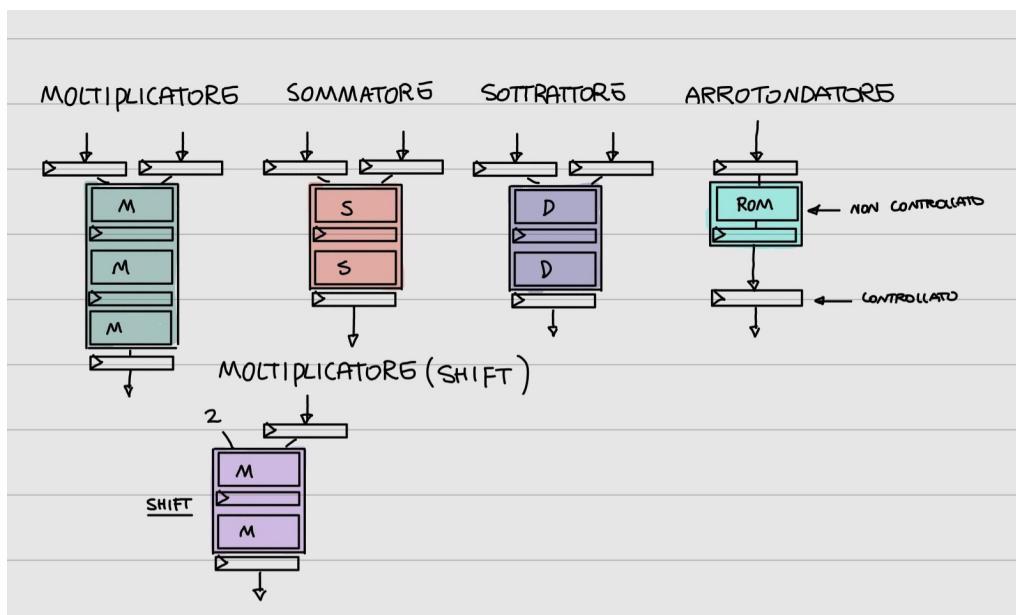


Figura 1: Blocchi elementari del data flow diagram

2.2 Approccio ASAP

L'approccio "As Soon As Possible", è un approccio che predilige lo svolgimento delle operazioni non appena si ha disponibilità di blocchi logici. Come si può notare in *Fig. 2* sarebbero necessari 6 blocchi moltiplicatori e 2 blocchi sommatore e sottrattore. Andando a confrontare le richieste di questa metodologia con le specifiche di progetto risulta subito evidente che questo approccio non è applicabile.

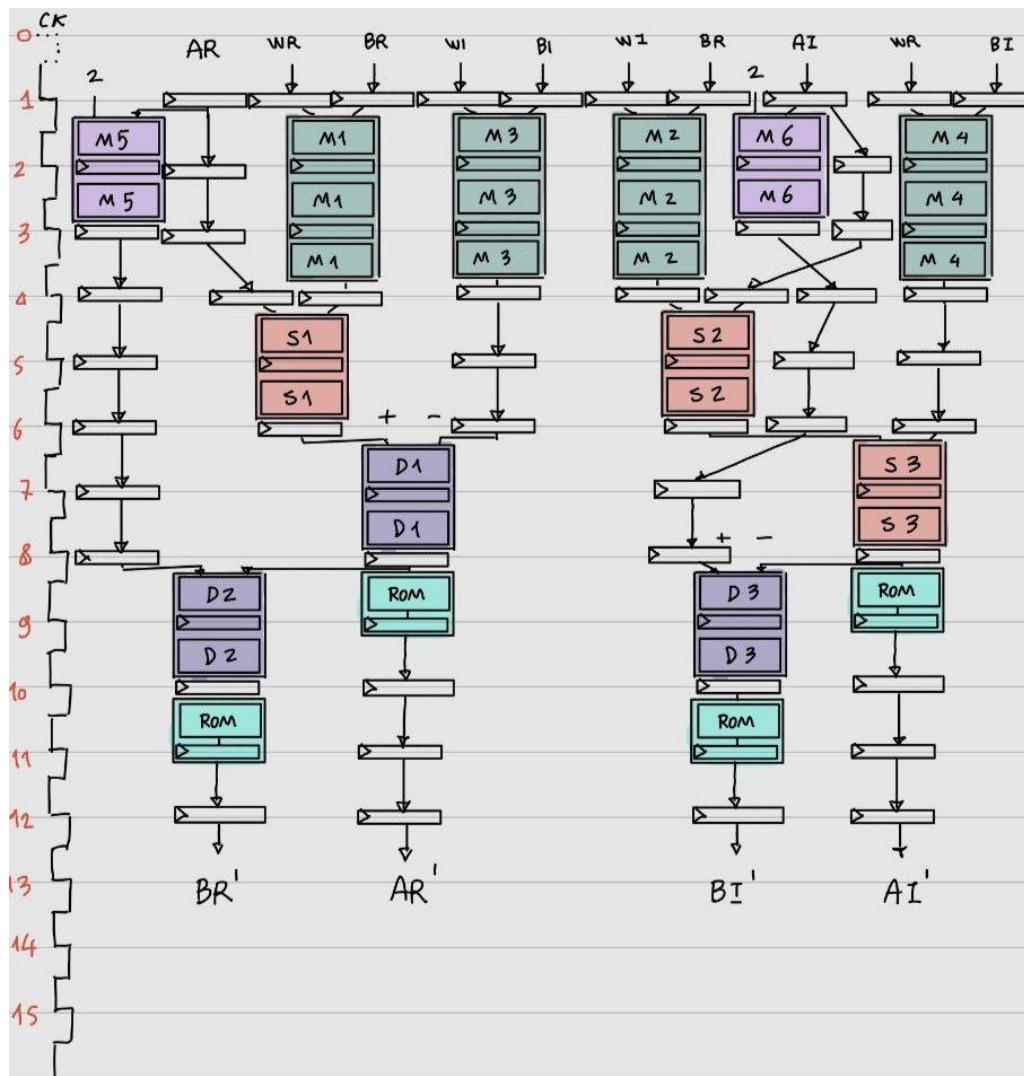


Figura 2: Data Flow Diagram con pproccio ASAP

2.3 Approccio ALAP

Questo approccio predilige lo svolgimento delle operazioni il più tardi possibile. Lo schema riportato in Fig. 3 mostra come il numero di blocchi operazionali richiesti sia inferiore rispetto all'approccio ASAP. Vengono infatti utilizzati 2 elementi per ciascun operazione di moltiplicazione, somma, differenza e arrotondamento. Anche in questo caso però non viene rispettato il limite numerico di 1 blocco moltiplicatore per Butterfly. È stato dunque studiato un approccio che ci permettesse di rimanere entro le specifiche richieste dal progetto.

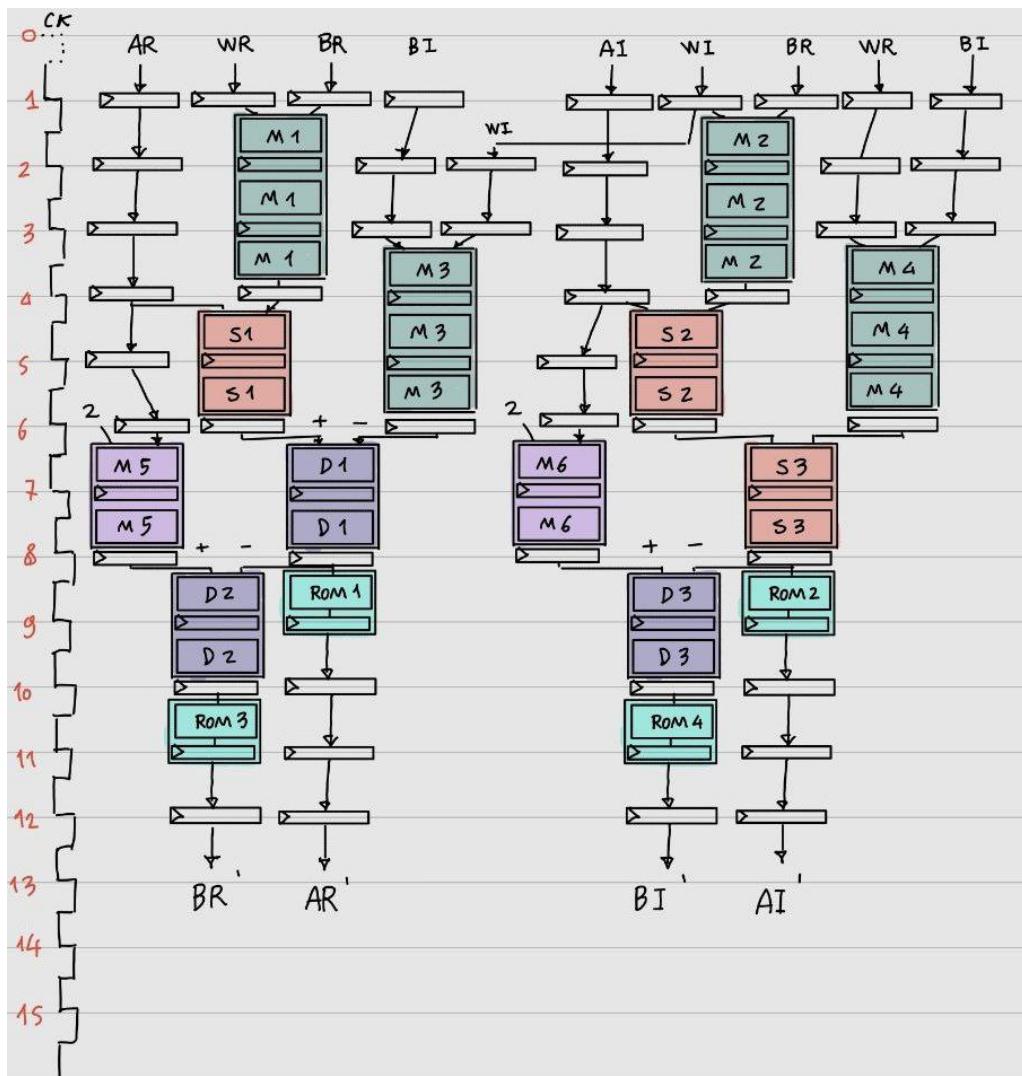


Figura 3: Data Flow Diagram con approccio ALAP

2.4 Approccio scelto

Volendo ottimizzare al massimo il sistema pur rimanendo all'interno delle specifiche imposte si è optato per il Data Flow Diagram illustrato in *Fig. 4*. Questo approccio è stato studiato per l'esecuzione dell'algoritmo in modo da avere ad ogni stadio un unico blocco logico per tipo di operazione.

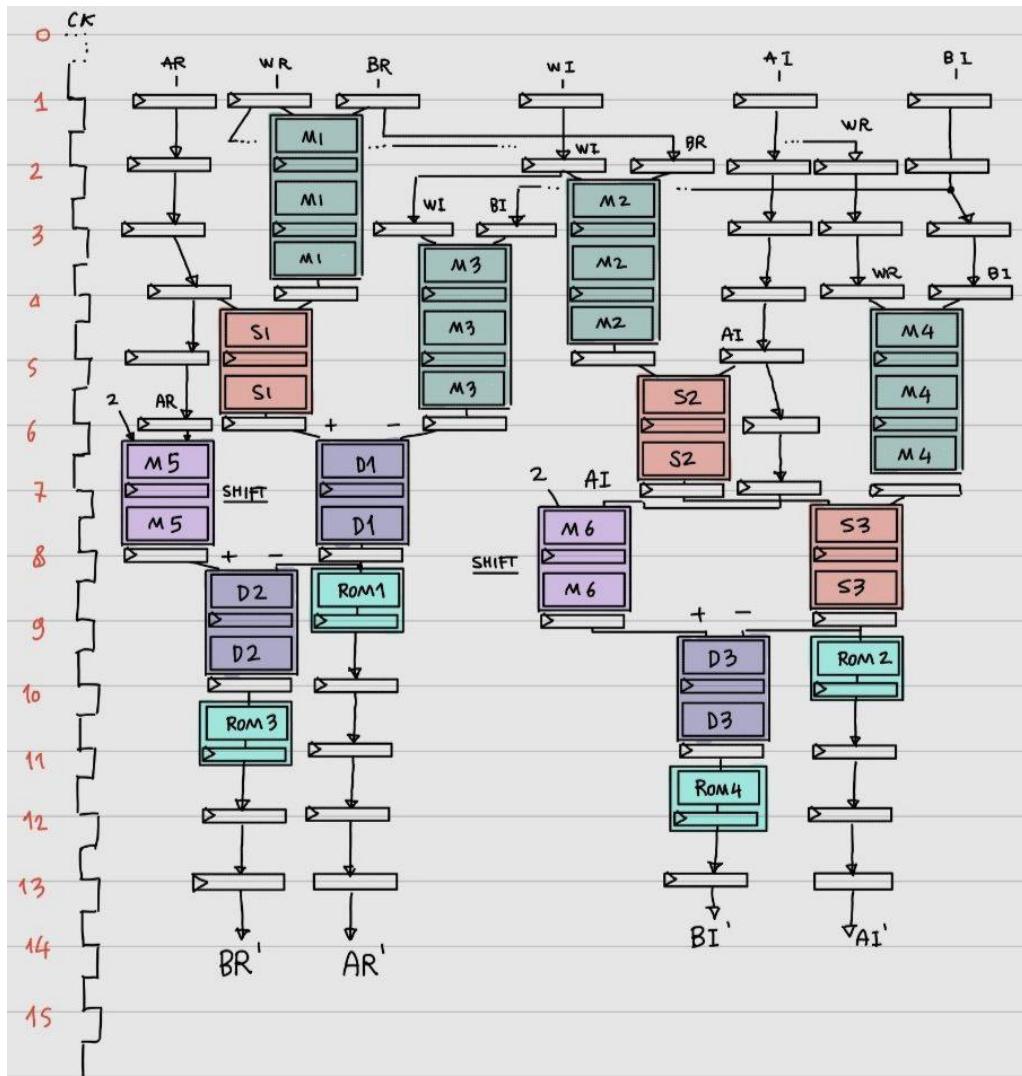


Figura 4: Data flow diagram ottimizzato

2.5 Tempo di vita delle variabili

In Fig. 5 viene illustrato il tempo di vita delle variabili derivato dal Data Flow Diagram che è stato scelto. Questo studio risulta utile per capire quanto una variabile deve rimanere in vita e, di conseguenza, quanti registri sono necessari per gestire tutte le variabili all'interno del sistema. Andando a studiare qual è il numero massimo richiesto di registri si riduce il costo e l'area occupata su Silicio. Inoltre, si facilita lo studio di quali e quanti segnali sono richiesti dalla CU per controllare il flusso dei dati.

In questo caso però, per come è stato progettato il sistema, gli unici registri che necessitano un segnale di controllo esterno sono i registri di ingresso e d'uscita della Butterfly. I registri posizionati tra i blocchi logici, dato che sono "vivi" per un solo colpo di clock, non necessitano di segnali di controllo.

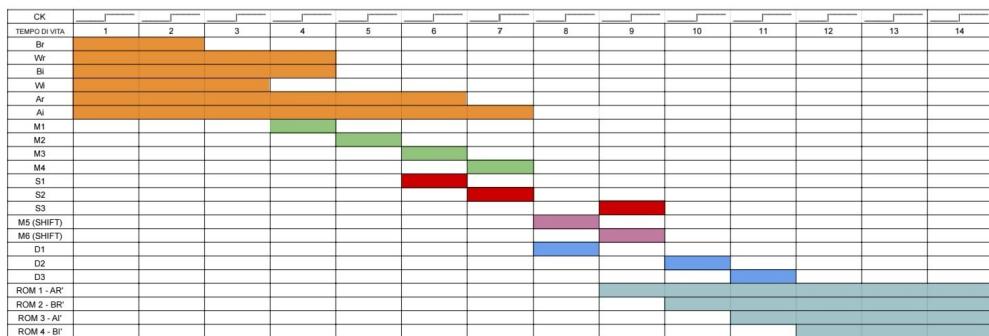


Figura 5: Tempo di vita delle variabili

3 Datapath

Dopo aver stimato e valutato il tempo di vita delle variabili si è iniziato a progettare il datapath necessario a svolgere tutte le operazioni richieste della CU. Il primo datapath studiato è rappresentato in *Fig. 6*. Come si vede dallo schema non è stato apportato ancora nessun miglioramento volto all'ottimizzazione del numero di BUS e/o al loro parallelismo.

Successivamente si è preso come riferimento il Data Flow Diagram (DFD) e si sono apportate delle migliorie per ciò che concerne l'efficienza dello schema circuitale. Come si vede da *Fig. 7* il register file è stato mantenuto e tutti i segnali che devono entrare all'interno del Datapath passano attraverso di esso. A valle sono stati inseriti dei MUX con lo scopo di selezionare i vari dati da mandare ai blocchi logici. Dal DFD è chiaramente visibile il fatto che i vari segnali, durante il loro tempo di vita, entreranno solo in specifici blocchi logici, risulta perciò superfluo e deleterio avere dei collegamenti (BUS) tra ogni uscita del register file e ogni blocco logico. Dato che l'uscita di un blocco logico potrebbe dover essere riutilizzata in uno step successivo si è fatto uso di registri intermedi che permettono di memorizzare e di riportare il dato in ingresso quando risulta necessario. Ciò è conveniente in quanto, facendo in questo modo, si risparmiano molte scritture su BUS che risultano essere lente e dispendiose in termini energetici. Infine, è stato esplicitato il blocco ROM Rounding che serve per arrotondare.

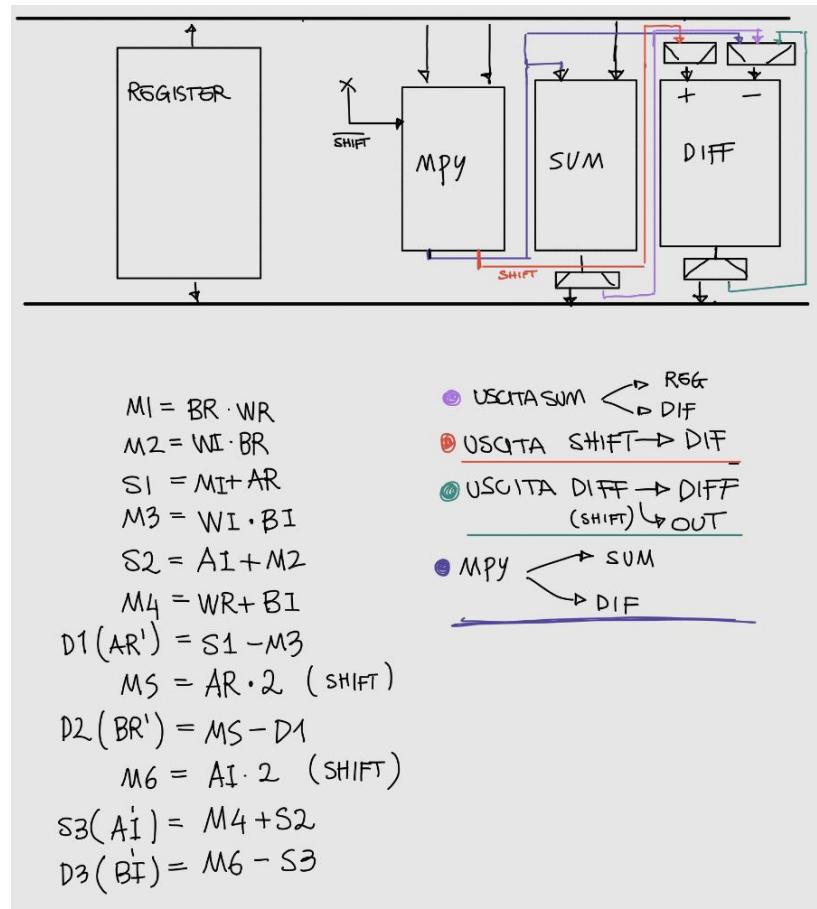


Figura 6: Schema del datapath iniziale

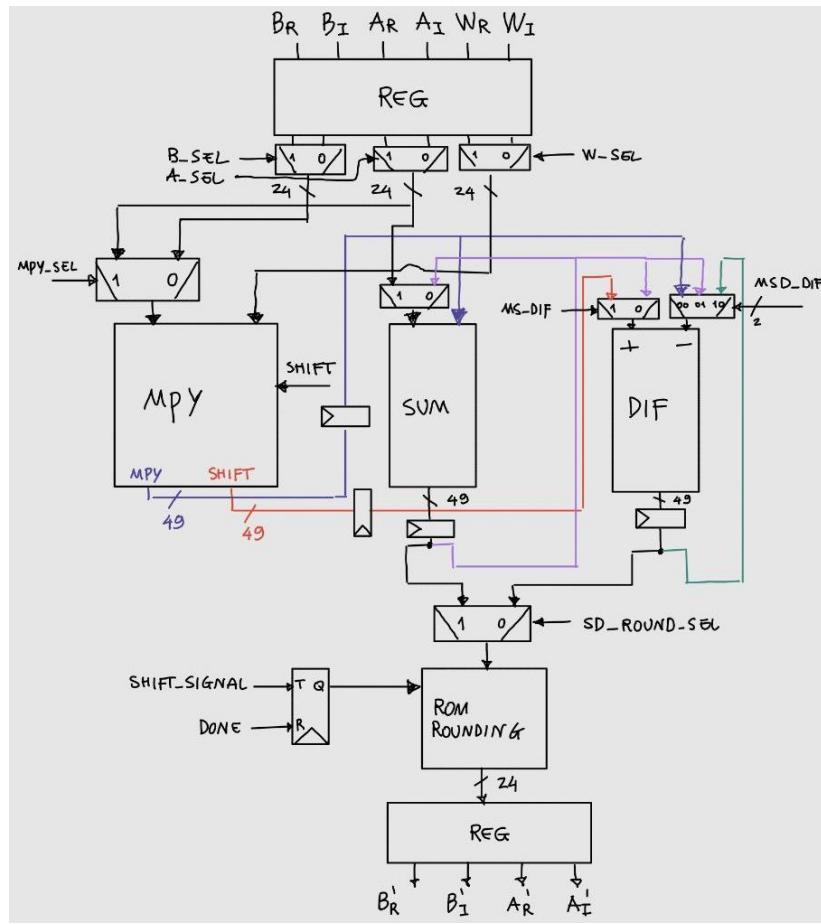


Figura 7: Schema del datapath finale

3.1 ROM Rounding

Come richiesto nella consegna del progetto per avere un uscita nel formato Q1.23 è necessario fare uso del ROM rounding. Questa tecnica consiste nell'arrotondare gli N bit in ingresso al blocco arrotondatore con una look-up-table salvata all'interno di una memoria ROM. Per leggere i dati presenti all'interno della memoria sarà necessario fornire all'ingresso un indirizzo. La scelta del numero di bit dell'indirizzo, e conseguentemente del numero di celle della memoria, è una scelta critica per il progetto in quanto un indirizzo di pochi bit consente di avere una memoria piccola (e che quindi richiede poco spazio su Silicio) ma permette un arrotondamento peggiore. Nel caso sia necessario ottenere un arrotondamento più preciso, e quindi con errore minore, allora risulta obbligatorio aumentare il numero di bit di indirizzo per permettere l'indirizzamento di più celle di memoria. Si riporta di seguito una tabella che mette in relazione il numero di bit dell'indirizzo con il numero di righe dell'ROM e con il numero di bit totali da memorizzare.

bit indirizzo	righe ROM	bit totali (singola butterfly)	bit totali (FFT)
3	8	24	768
4	14	56	1792
5	32	160	5120
6	64	384	12288

Tabella 1: Relazione tra il numero di bit di indirizzo della ROM, il numero di righe ed il numero totale di bit memorizzati

Bisogna anche considerare il bias e l'errore medio. Avendo come specifica di progetto l'utilizzo del metodo "Round to Nearest Even" si è dovuto scegliere un indirizzo composto da un numero di bit della mantissa e bit di scarto disposti in maniera tale da minimizzare sia il bias che l'errore. Di seguito si riportano i test effettuati:

Dopo aver considerato tutte le opzioni, sia dal lato di area occupata che dal lato bias/errore, è stato scelto di comporre l'indirizzo della ROM con gli ultimi 3 bit della mantissa (LSB mantissa) e con i primi 2 bit dello scarto (MSB scarto). Il numero totale di bit che compongono l'indirizzo è quindi 5. Si riporta in Fig. 8 lo schema del ROM rounding implementato. Si può apprezzare la presenza della ROM, un registro posto in ingresso e uno in uscita usati per

bit indirizzo	bias	errore
5 (2 Mantissa + 3 Scarto)	-1/16	-4
5 (3 Mantissa + 2 Scarto)	1/16	-1
6 (3 Mantissa + 3 Scarto)	0	-4

Tabella 2: Relazione tra il numero di bit di indirizzo della ROM, il bias e l'errore

rendere i dati disponibili sul fronte del clock dato che la ROM è puramente combinatoria e, infine, il parallelismo dei bus espresso col numero di fianco al bus stesso.

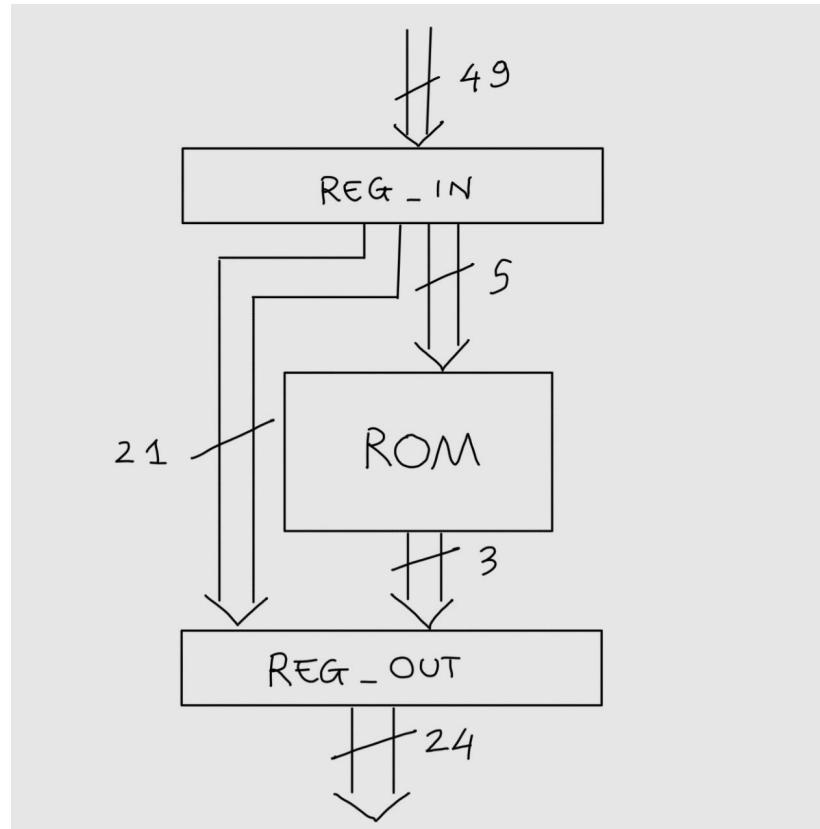


Figura 8: Schema del ROM rounding implementato

4 Control Unit

La Control Unit è l'unità logica che decide le operazioni da far svolgere al datapath. Come si vede da *Fig. 9* nell'unità di controllo si distinguono tre componenti: la late status PLA, la μ ROM, e il datapath con il micro Instruction Register e il micro Address Register; questi verranno discussi in capitoli separati.

4.1 Comandi e stati

Gli stati della control unit sono stati codificati in modo da trarre il maggiore beneficio dalla tecnica del LATE STATUS: gli indirizzi di stati che si susseguono sono distanti di solo un LSB.

Gli stati di IDLE e DONE sono caratterizzati da $LSB = 0$ e $CC_Validation = 0$: la late status pla usa questa informazione e determina il LSB dello stato successivo solamente in base al segnale di START, in modo da ripetere lo stato di IDLE se è negato o di passare allo stato di START se è asserito. Quando la butterfly ha terminato un'esecuzione passa allo stato di DONE e poi in IDLE se non riceve un segnale di START. Lo stato di IDLE si ripete finché non viene ricevuto il segnale di START.

Lo stato di START è caratterizzato da $LSB = 1$ e $CC_Validation = 1$: il LSB dello stato successivo è determinato dal segnale di SF_2H_1L , che viene asserito o negato dall'esterno un colpo di clock dopo il segnale di START; in questo stato vengono anche campionati A, B e W.

Lo stato successivo gestisce i segnali di controllo per lo svolgimento della prima operazione, Br^*Wr . Se nello stato precedente SF_2H_1L è stato asserito, viene anche operato un flip flop di tipo T per asserire il segnale di shift per il blocco rounding.

Gli stati successivi gestiscono i segnali per le altre operazioni di moltiplicazione, addizione, sottrazione e rounding.

STATO	CC_VALIDATION	INDIRIZZO
IDLE	0	0000
START	1	0001
M_1, SH_0	0	0010
M_1, SH_1	0	0011
M_2	1	0100
M_3	0	0101
M_4, S_1	1	0110
S_2	0	0111
M_5, D_1	1	1000
M_6, S_3	0	1001
D_2, SH_1	1	1010
D_3, SH_2	0	1011
SH_3	1	1100
SH_4	0	1101
DONE	0	1110

Tabella 3: Stati del sistema

4.2 Struttura dell'unità di controllo

4.2.1 Late Status PLA

Come da specifiche di progetto l'unità di controllo fa uso di un indirizzamento esplicito e della tecnica "Late Status". Per velocizzare il sistema ed evitare un calo delle prestazioni la macchina può saltare il ritardo dovuto al campionamento del LSB da parte del μ AR se il resto dell'indirizzo rimane uguale, ovvero se un salto non avviene, ciò che discrimina la necessità di effettuare o no un salto è il bit meno significativo dell'indirizzo stesso. La μ ROM è stata suddivisa in locazioni pari e dispari (vedasi paragrafo " μ ROM" per maggiore dettagli). In *Fig. 9* è possibile vedere le porte logiche che compongono la Status PLA e, riportati sotto la figura, le espressioni logiche che determinano il valore dei bit in uscita da questo blocco.

4.2.2 μ ROM

Come spiegato nel paragrafo precedente la μ ROM è stata suddivisa in indirizzi pari e dispari. I 3 MSB del micro Address Register entrano sia nella parte pari che nella parte dispari, e la μ ROM porta in uscita entrambi i dati su 22 bit. Un MUX posto in uscita alla μ ROM determina quale dei due lati deve essere campionato dal micro Instruction Register, usando come segnale di select o il LSB del micro Address Register in caso di salto o il LSB collegato direttamente all'uscita della Status PLA.

CC	LSB	START	SF_2H_1L	LSB_OUT	CC_OUT
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	1	1	0

Tabella 4: Comandi

4.2.3 datapath, μ AR e μ IR

il μ IR è un registro a 22 bit, contiene un bit di CC_ Validation usato dalla Late Status PLA per determinare il prossimo LSB, 17 bit di segnali di controllo per il datapath, e 4 bit di indirizzo per il prossimo stato di cui il LSB viene passato alla Late Status PLA e i restanti bit al μ AR. Per evitare ritardi eccessivi, è stato deciso di collegare al μ AR il clock senza alterazioni e al μ IR il clock negato. Le micro istruzioni raggiungono il datapath della FFT prima del successivo fronte positivo del clock. Per implementare la tecnica del Late Status è stato aggiunto un multiplexer a 1 bit che dà il segnale di select al MUX in uscita alla μ ROM.

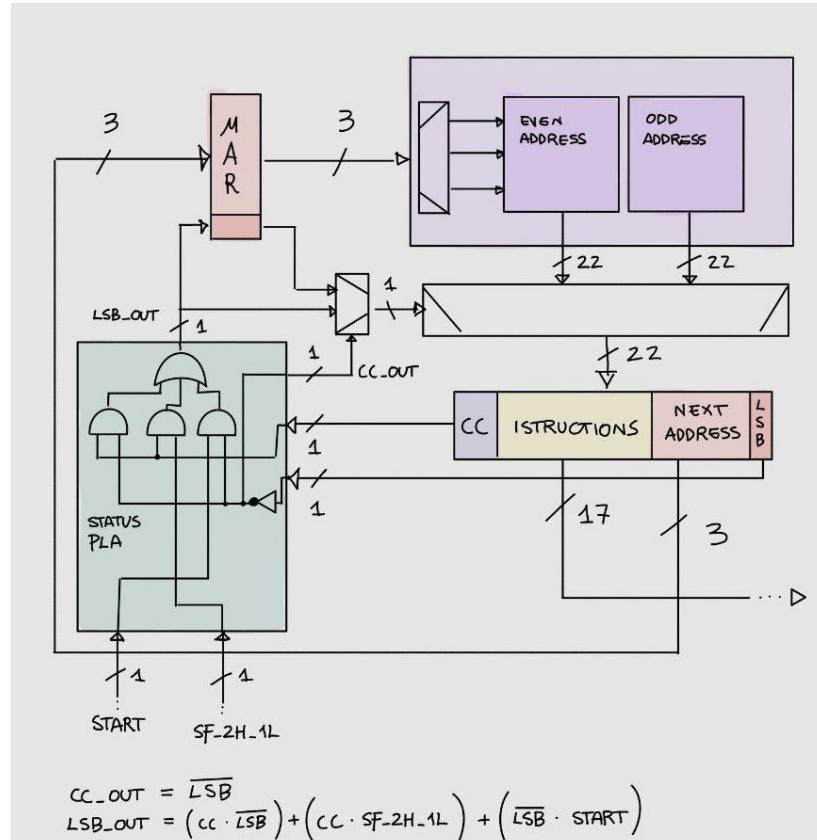


Figura 9: Schema della CU implementato

5 Butterfly e FFT

5.1 Butterfly

Dopo aver descritto, tramite linguaggio VHDL, i vari blocchi precedentemente descritti si è passati alla creazione di un blocco butterfly singolo per un test intermedio e per accertarsi che tutto funzionasse in modo sinergico. I listati possono essere consultati nel *Cap. 7*.

Questo test è risultato importante perché ha permesso di correggere alcuni piccoli errori che non erano stati individuati durante i test precedenti. Grazie a queste modifiche il singolo blocco butterfly ha funzionato correttamente durante i successivi test e ciò a permesso di proseguire nell'implementazione della FFT 16x16 senza doversi preoccupare di possibili errori commessi in precedenza.

Come si vede da *Fig. 10* la butterfly singola riceve in ingresso segnali con parallelismo 24 bit e il segnale SF_2H_1L per gestire lo shift. In uscita presenta un bus con parallelismo 24 bit. Al suo interno sono presenti due blocchi, la Control Unit e il Datapath. Tutto ciò è conforme con quanto richiesto dalla consegna del progetto ovvero ingresso a 24 bit e uscita a 24 bit a prescindere dal parallelismo adottato internamente per sviluppare i calcoli ed effettuare le numerose operazioni logiche.

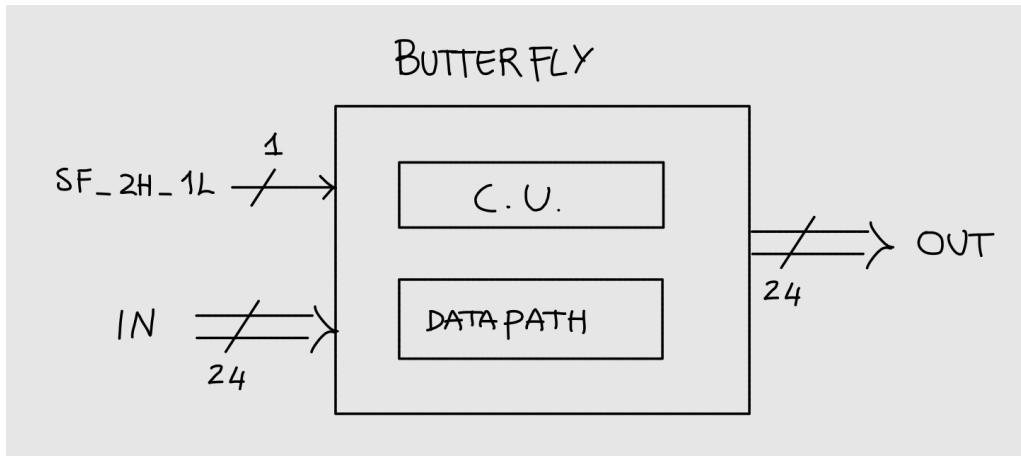


Figura 10: Rappresentazione di una butterfly singola

5.2 FFT

Una volta creato la singola butterfly si è proceduto con la creazione della struttura che permette l'implementazione hardware della FFT. Per fare ciò la singola butterfly è stata replicata trentadue volte. Volendo rendere più semplice la gestione dei segnali in ciascuna butterfly sono stati assegnati staticamente i valori di SF_2H_1L, Wr e Wi. Di seguito si riporta la struttura implementata su VHDL che forma la FFT.

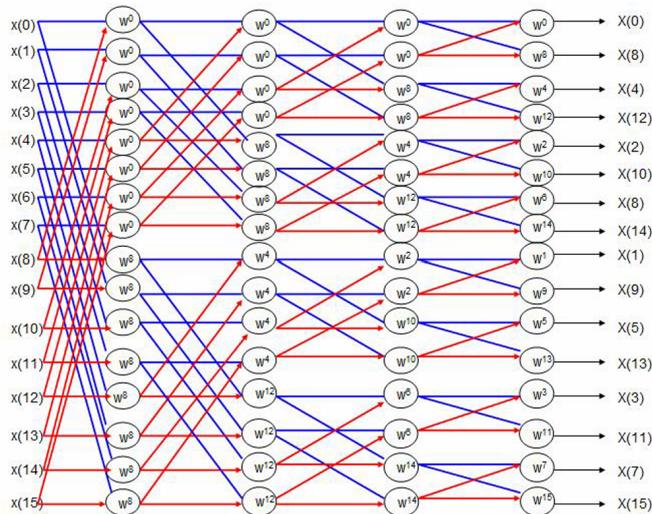


Figura 11: schema logico della FFT

Per poter testare il sistema creato è stato necessario porre in ingresso dei vettori di test la cui FFT è nota. In questo modo è stato possibile confrontare il risultato atteso con quello ottenuto. I vettori di test e i risultati attesi sono riportati nella seguente tabella mentre i risultati delle simulazioni sono esposti nel capitolo successivo.

V ₁	FFT(x32)	V ₂	FFT(x32)	V ₃	FFT(x32)	V ₄	FFT(x32)	V ₅	FFT(x32)	V ₆	FFT(x32)
-1	-16	-1	0	1	1	-1	0	0.5	1	0	0.75
-1	0	0	0	0	1	-1	0	0.5	0-5.0273i	0	-0.75
-1	0	1	0	0	1	1	0	0.5	1	0	0.75
-1	0	0	0	0	1	1	0	0.5	0-1.4966i	0	-0.75
-1	0	-1	-8	0	1	-1	-8+8i	0.5	1	0	0.75
-1	0	0	0	0	1	-1	0	0.5	0-0.6682i	0	-0.75
-1	0	1	0	0	1	1	0	0.5	1	0	0.75
-1	0	0	0	0	1	1	0	0.5	0-0.1989i	0	-0.75
-1	0	-1	0	0	1	-1	0	0.5	1	0.75	0.75
-1	0	0	0	0	1	-1	0	-0.5	0+0.1989i	0	-0.75
-1	0	1	0	0	1	1	0	-0.5	1	0	0.75
-1	0	0	0	0	1	1	0	-0.5	0+0.6682i	0	-0.75
-1	0	-1	-8	0	1	-1	-8-8i	-0.5	1	0	0.75
-1	0	0	0	0	1	-1	0	-0.5	0+1.4966i	0	-0.75
-1	0	1	0	0	1	1	0	-0.5	1	0	0.75
-1	0	0	0	0	1	1	0	-0.5	0+5.0273	0	-0.75

Tabella 5: Vettori di test e risultati attesi della FFT

6 Simulazioni

Le specifiche impongono due specifici modi di funzionamento della FFT: singola e continua. Tutti i test riportati sotto sono stati presi dalla modalità continua ma questo non è ben visibile. Per sopperire ad ogni possibile dubbio verranno riportate, prima dei test, due immagini che mostrano il sistema che lavora in modalità singola e continua.

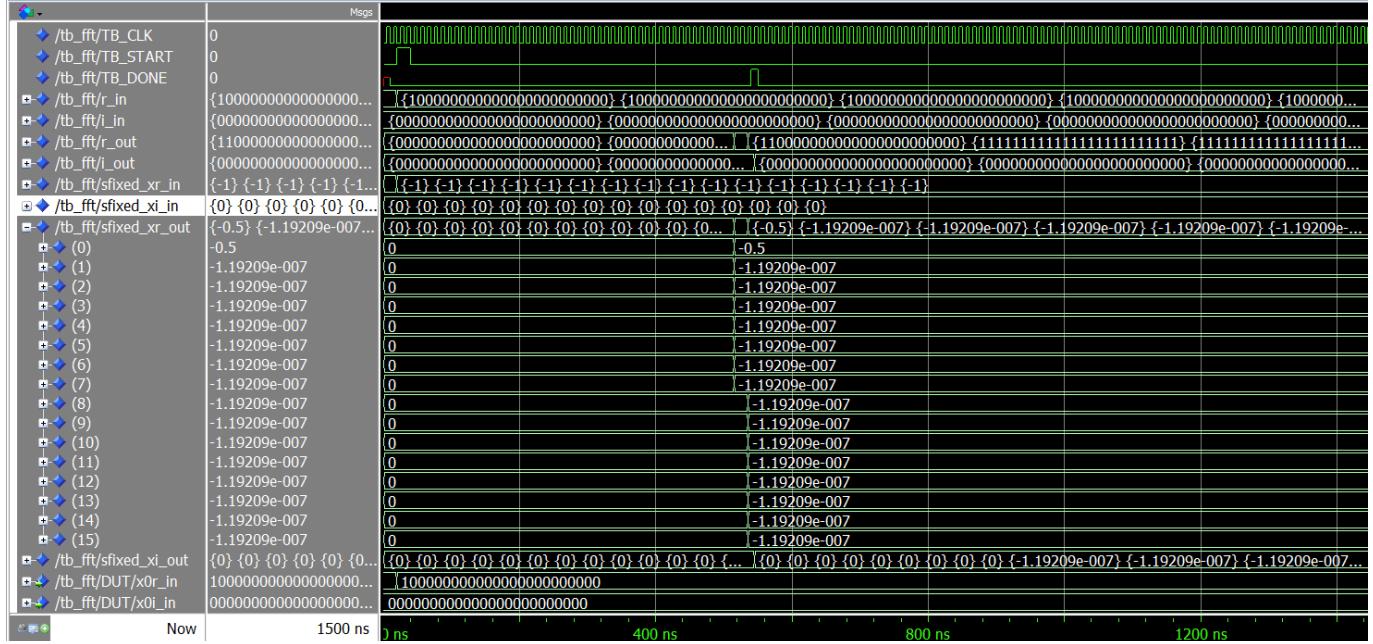


Figura 12: Funzionamento in modalità singola

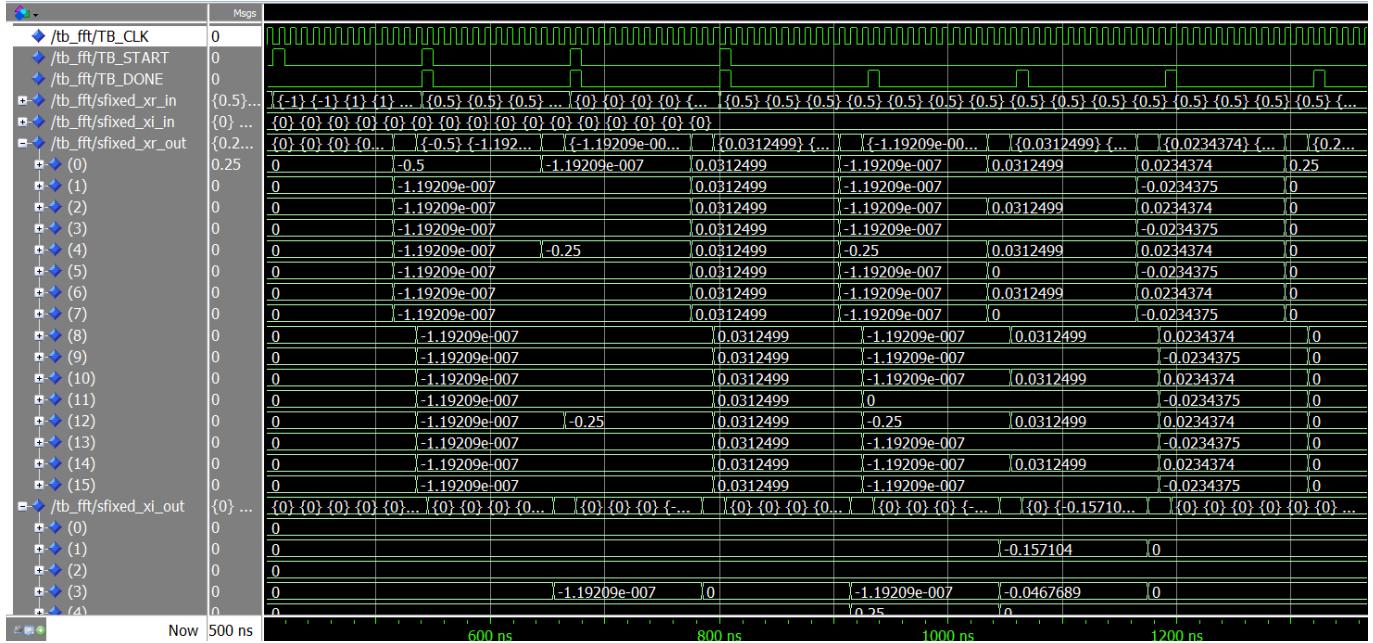


Figura 13: Funzionamento in modalità continua

A causa dell'aritmetica fixed point utilizzata nella FFT, alcuni campioni che dovrebbero essere zero presentano valori molto piccoli ma non nulli. Questi valori sono dell'ordine di un LSB del formato Q1.23 ($LSB = 1.19209 \cdot 10^{-7}$). Sono causati dagli errori di quantizzazione e di arrotondamento che si verificano durante le operazioni di somma, moltiplicazione e regolazione all'interno della Butterfly. Queste componenti residue non hanno significato fisico né impatto sul contenuto spettrale del segnale e possono essere considerate nulle entro la risoluzione numerica del sistema.

Oltre a questo si fa presente che i risultati riportati in Tab. 5 sono stati moltiplicati per 32 mentre le immagini presentano valori ancora da scalare. Pertanto ogni valore letto nelle immagini va moltiplicato per 32 per ottenere il valore descritto nella tabella.

Infine, dalle figure sottostanti si può notare che alcuni campioni che escono dal blocco FFT iniziano a stabilizzarsi prima che il segnale DONE venga asserito. Tuttavia, per un corretto utilizzo dei dati, le uscite sono considerate valide e campionabili solo quando il segnale DONE è alto. Il segnale DONE, quindi, funge da indicatore di validità per l'intero vettore di output, garantendo che tutti i passaggi di calcolo interni e di propagazione dei dati siano completati. Qualsiasi valore presente alle uscite prima dell'asserimento di DONE non deve essere utilizzato poiché potrebbero essere presenti dei dati non corretti.

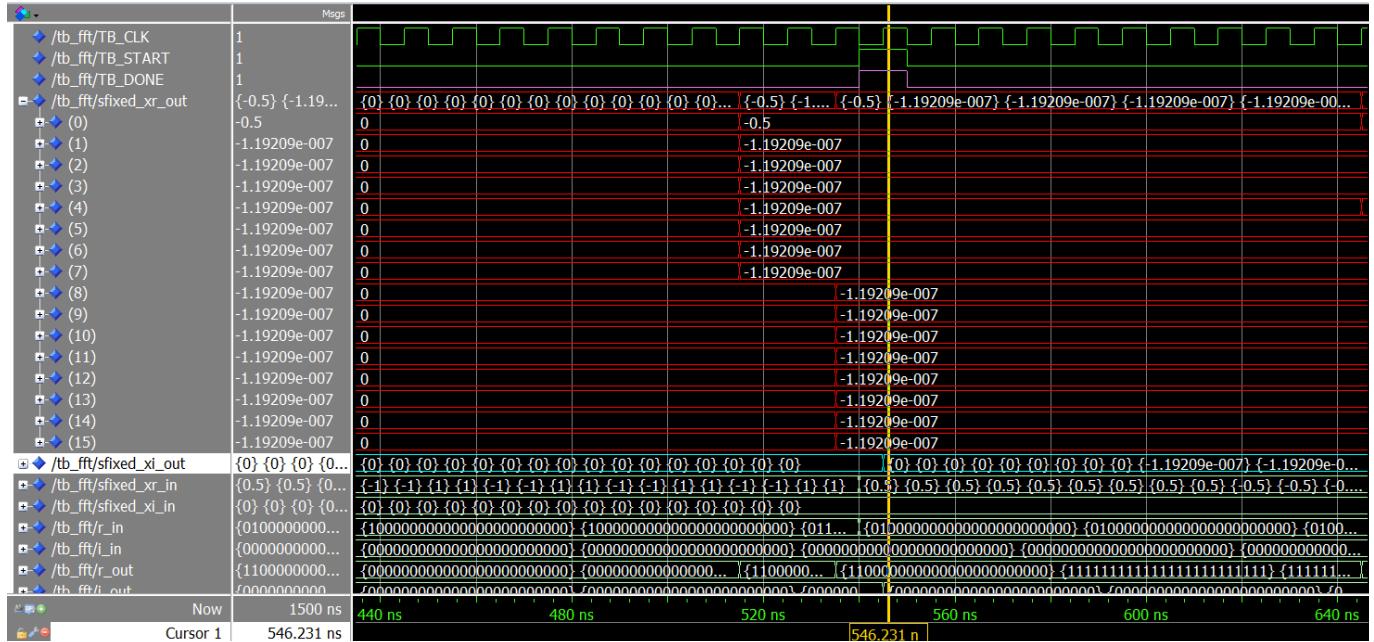


Figura 14: Risultati FFT di V_1 - parte reale

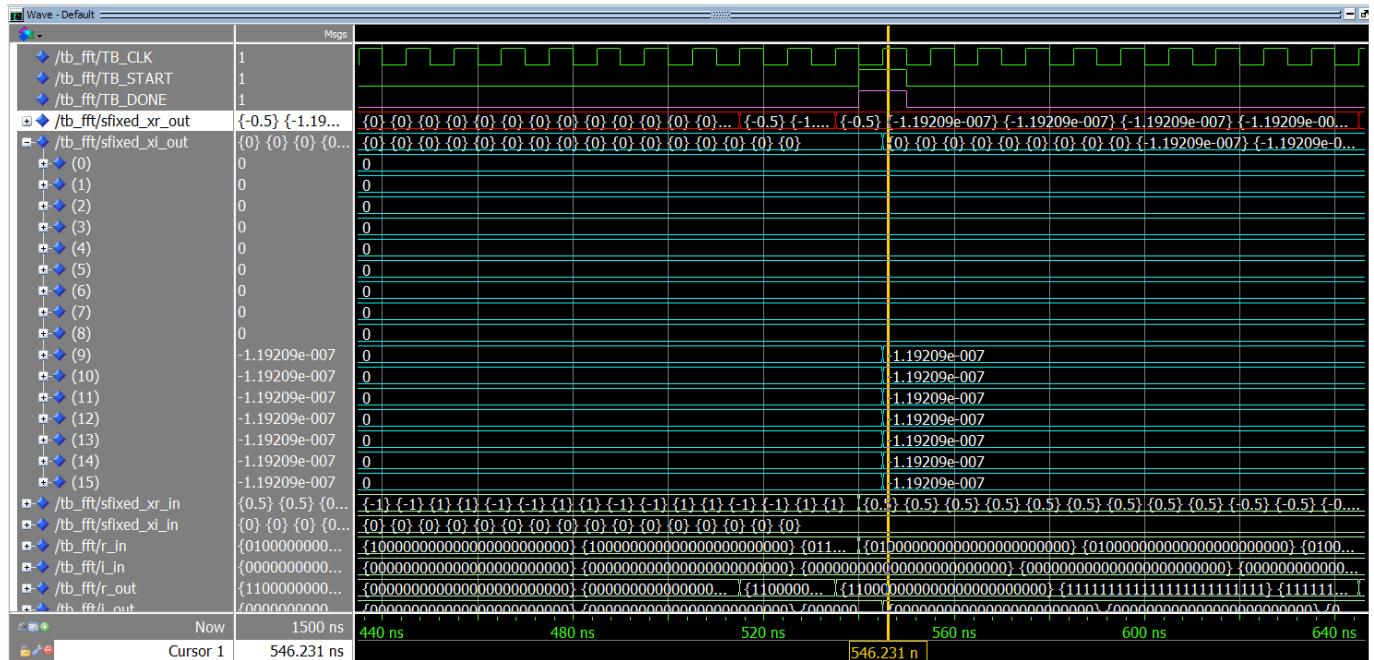
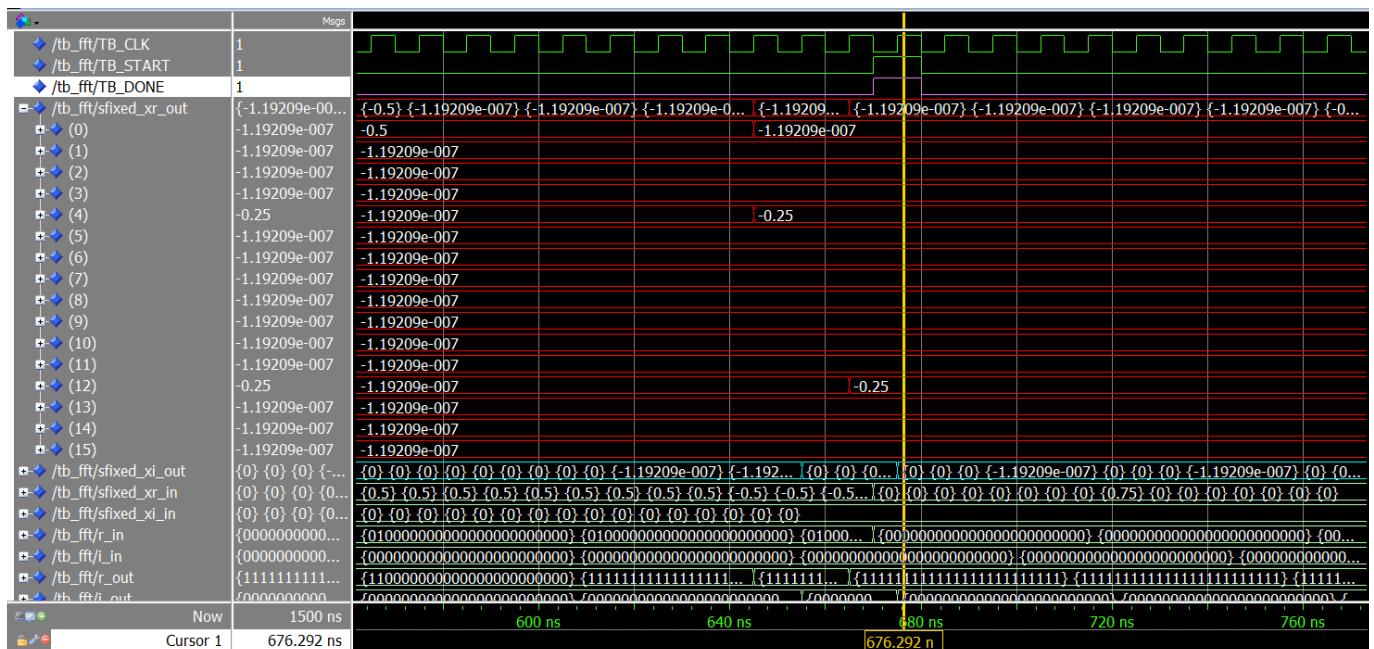
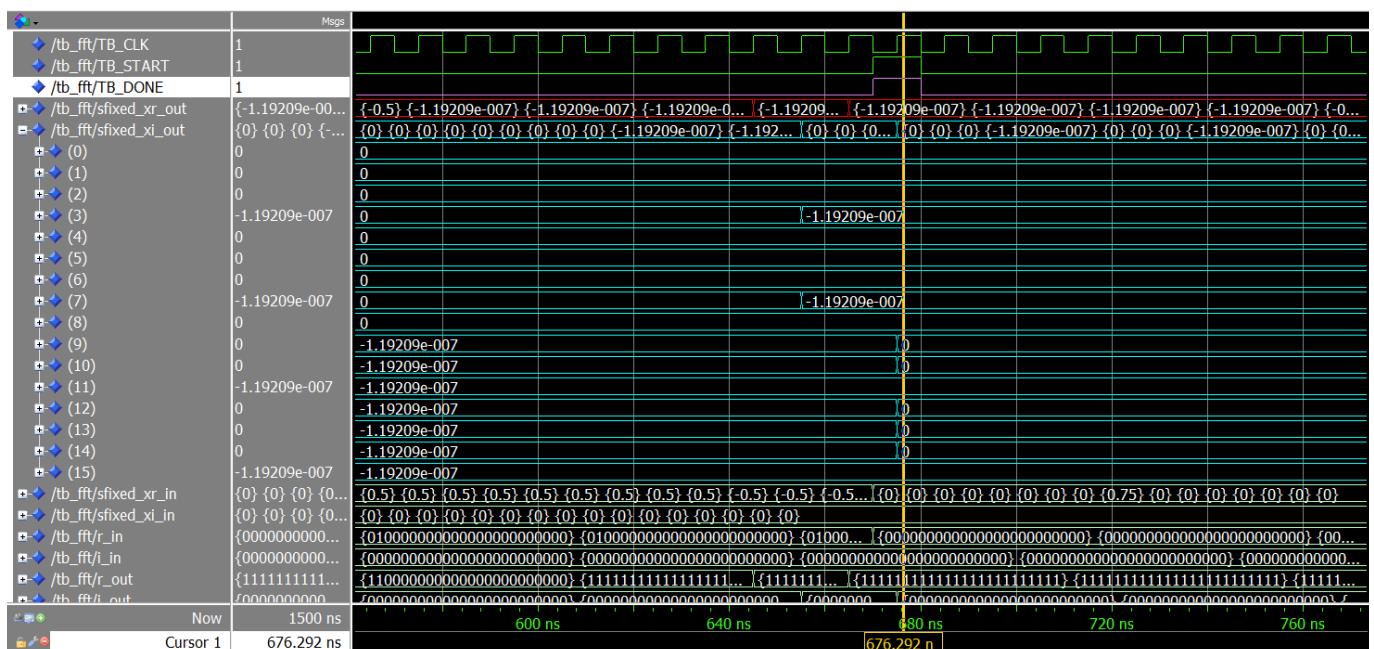
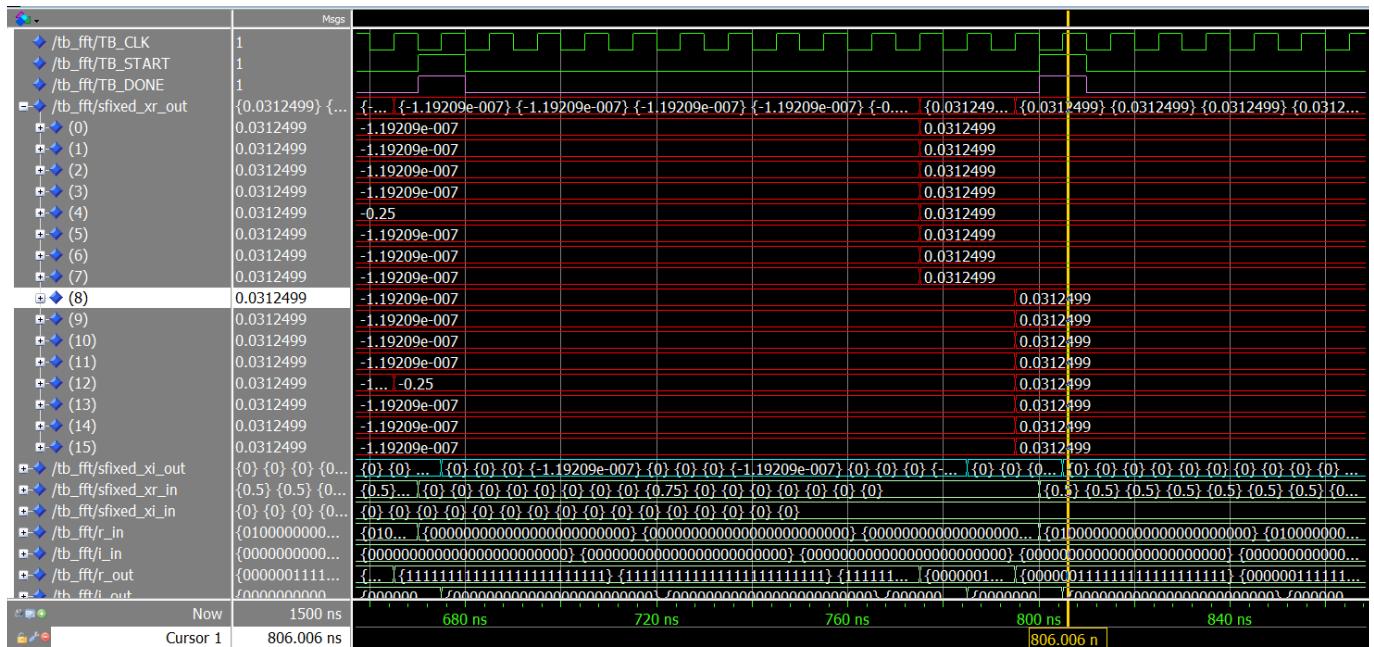
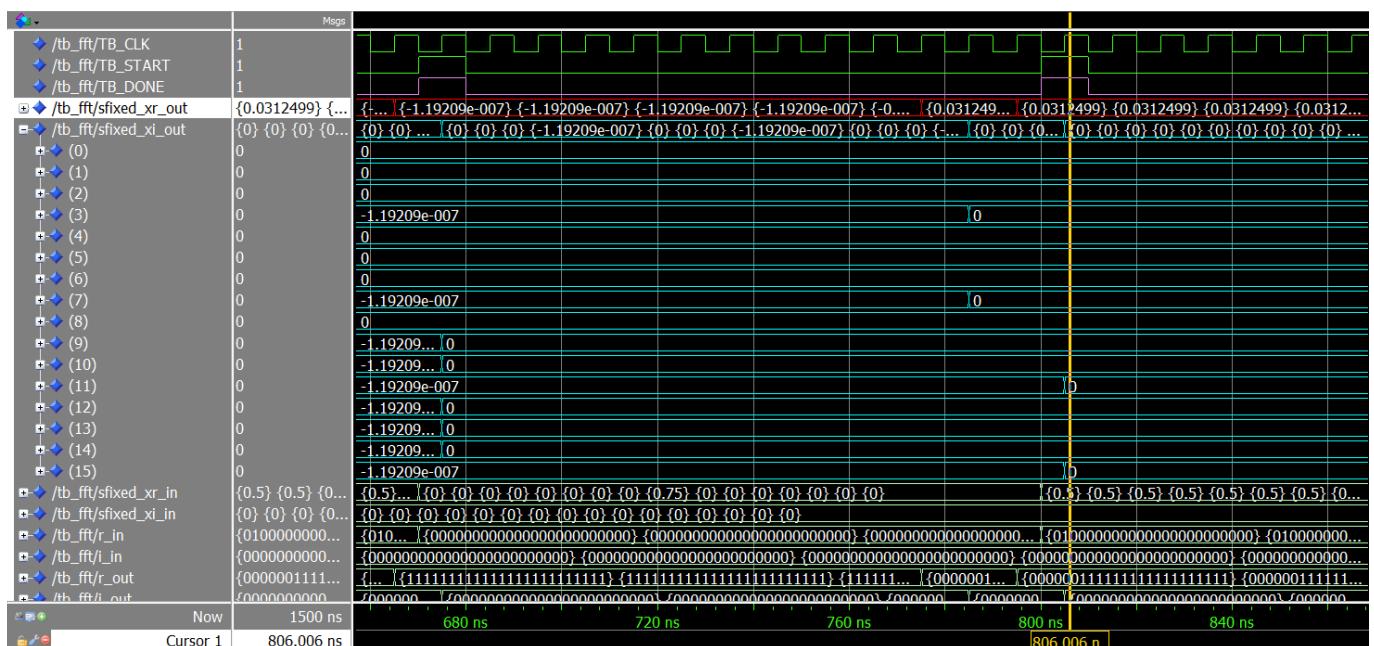
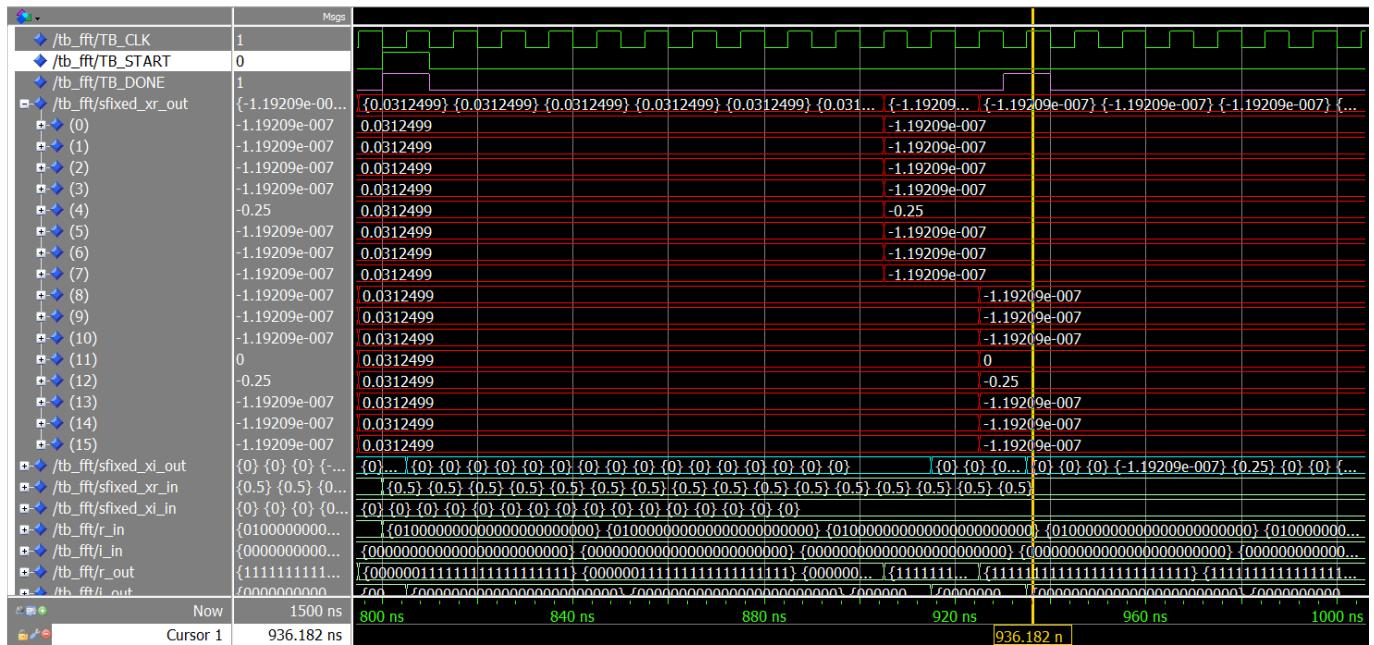
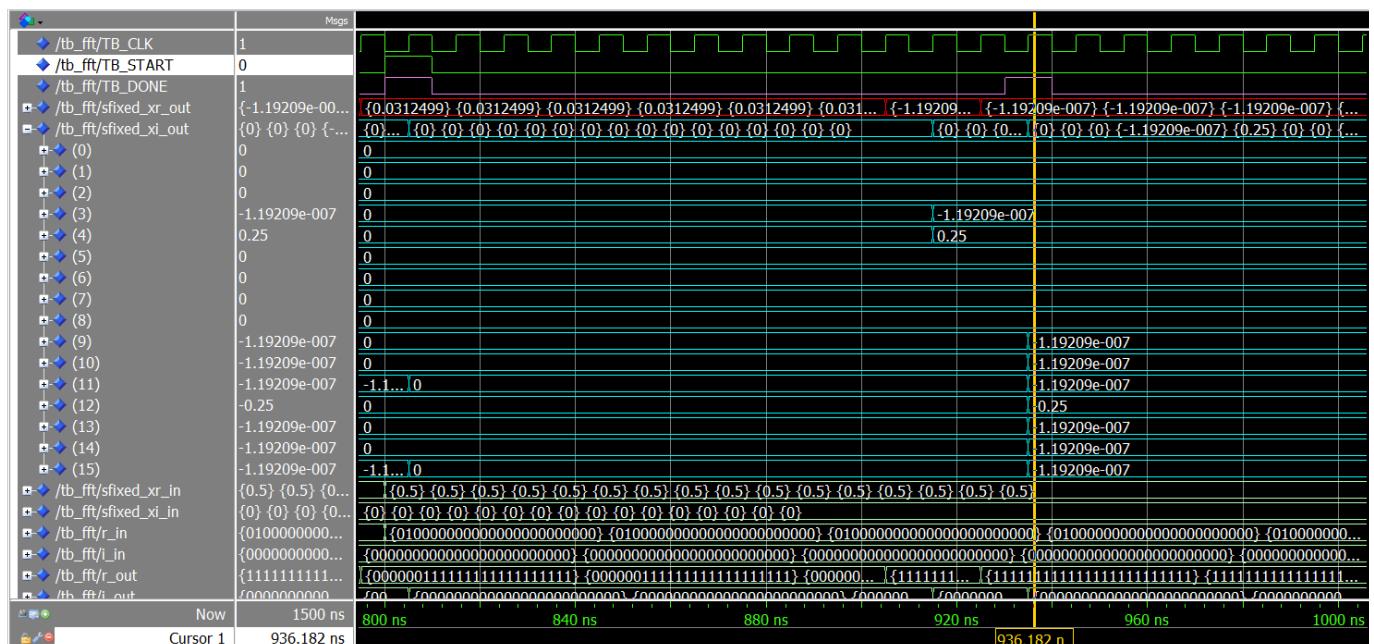
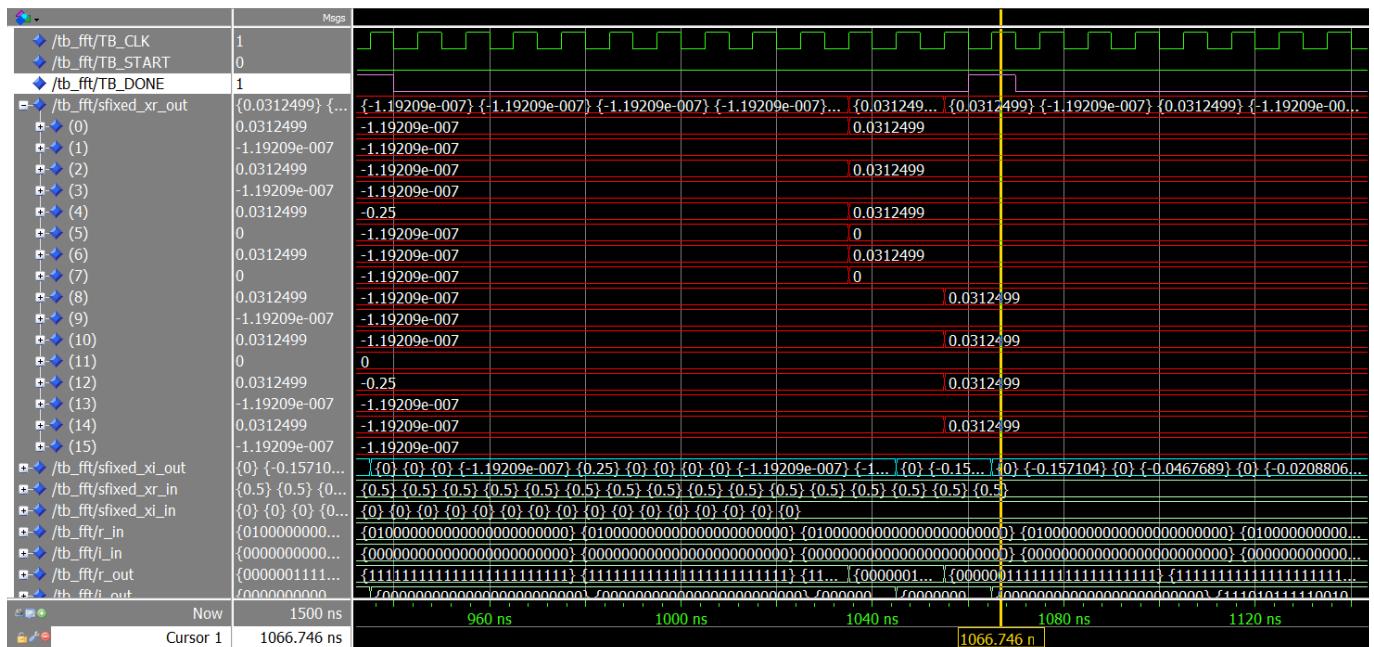
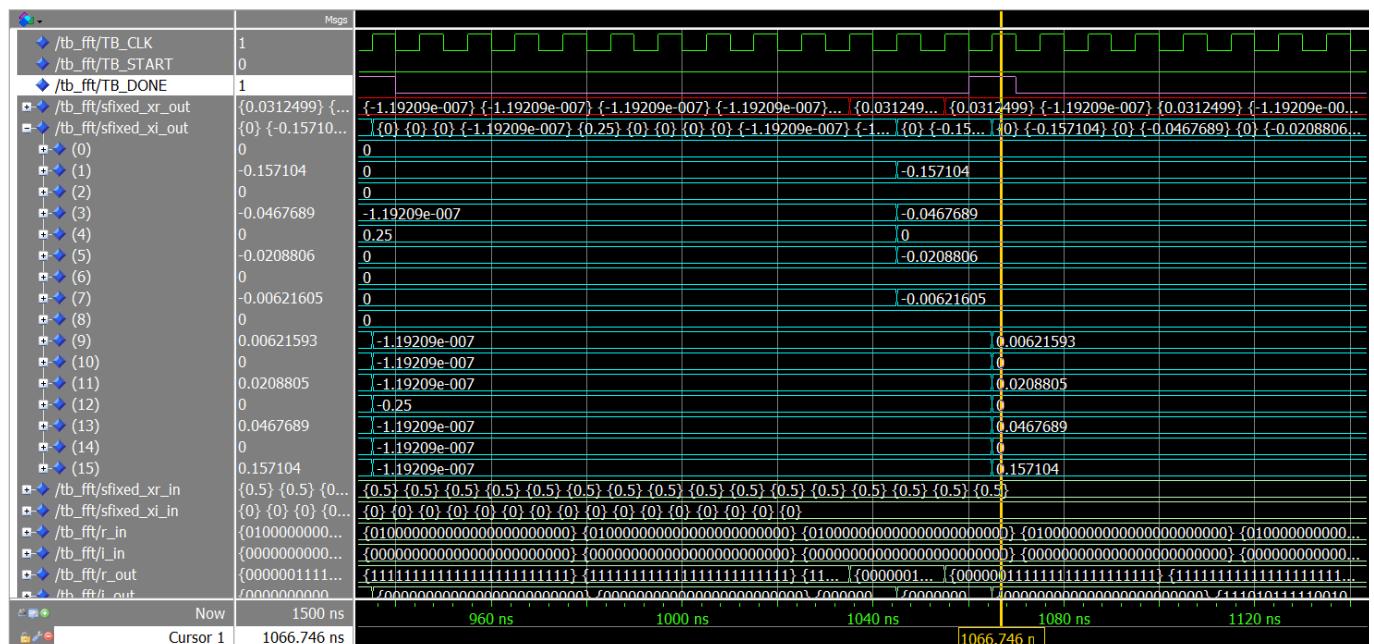


Figura 15: Risultati FFT di V_1 - parte immaginaria

Figura 16: Risultati FFT di V_2 - parte realeFigura 17: Risultati FFT di V_2 - parte immaginaria

Figura 18: Risultati FFT di V_3 - parte realeFigura 19: Risultati FFT di V_3 - parte immaginaria

Figura 20: Risultati FFT di V₄ - parte realeFigura 21: Risultati FFT di V₄ - parte immaginaria

Figura 22: Risultati FFT di V_5 - parte realeFigura 23: Risultati FFT di V_5 - parte immaginaria

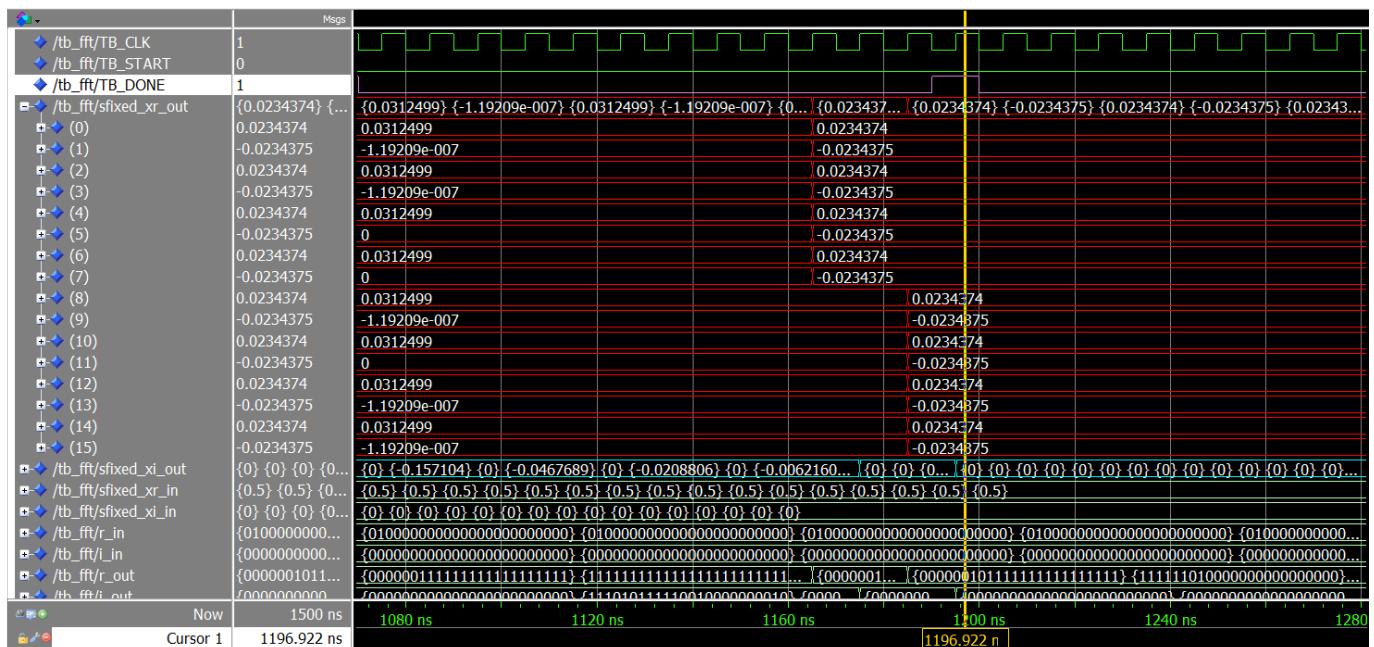


Figura 24: Risultati FFT di V_6 - parte reale

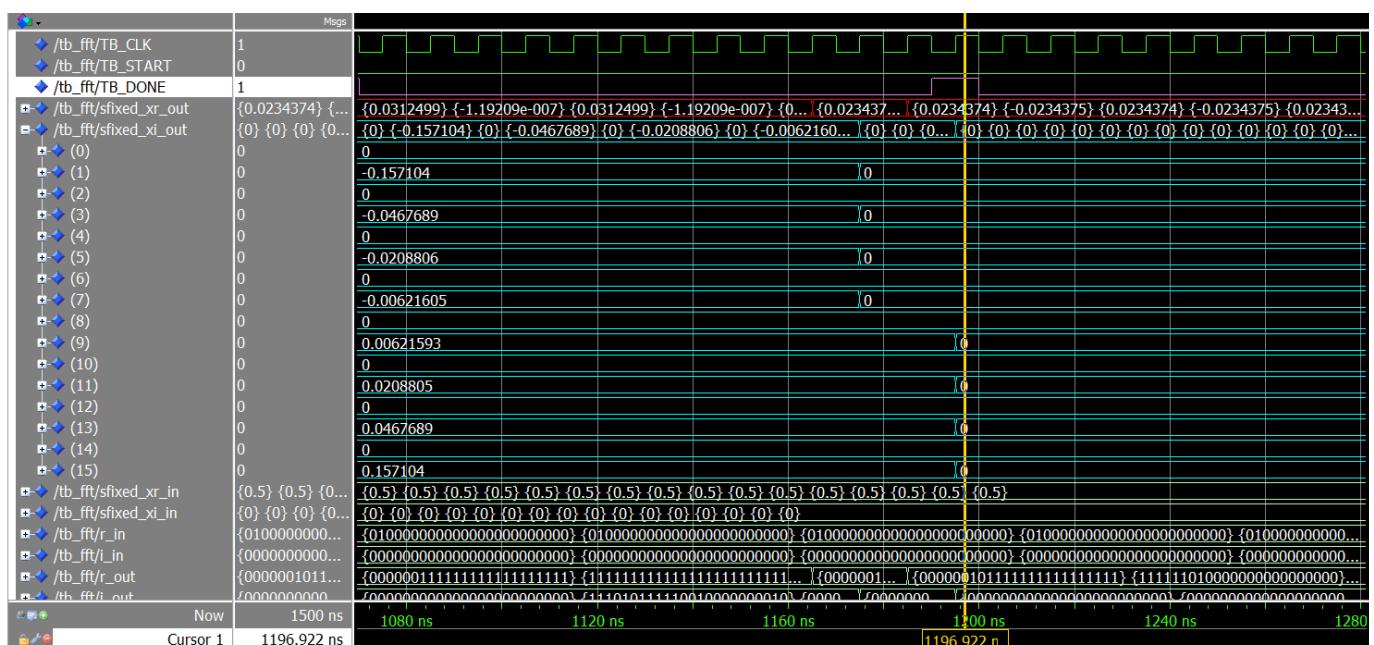


Figura 25: Risultati FFT di V_6 - parte immaginaria

7 Appendice - File VHDL

Come richiesto nelle specifiche del progetto si riportano di seguito i listati VHDL utilizzati per sviluppare/testare il progetto. Si è inserito il nome e la matricola di ogni componente del gruppo nei vari listati.

7.1 Sommatore

```

1  -- Federico Cobianchi - 332753
2  -- Onice Mazzi - 359754
3  -- Antonio Telmon - 353781
4
5  library IEEE;
6  use IEEE.std_logic_1164.all;
7  use IEEE.numeric_std.all;
8
9
10 entity BFLY_ADDER is
11     port ( A:      in      STD_LOGIC_VECTOR (48 downto 0);
12            B:      in      STD_LOGIC_VECTOR (48 downto 0);
13            CK:      in      STD_LOGIC;
14            SUM_OUT:    out      STD_LOGIC_VECTOR(48 downto 0)
15        );
16 end BFLY_ADDER;
17
18
19 architecture behavioral of BFLY_ADDER is
20
21     signal sum: STD_LOGIC_VECTOR (48 downto 0) := (others=>'0');
22
23 begin
24
25     sum <= std_logic_vector(signed(A)+signed(B));
26
27     PSYNC: process(CK)
28     begin
29         if CK'event and CK='1' then -- positive edge triggered:
30             SUM_OUT <= sum;
31
32         end if;
33     end process;
34
35 end behavioral;

```

Listing 1: Sommatore

7.2 MUX

7.2.1 MUX 2

```

1  -- Federico Cobianchi - 332753
2  -- Onice Mazzi - 359754
3  -- Antonio Telmon - 353781
4
5  library IEEE;
6  use IEEE.std_logic_1164.all;
7
8  entity MUX_2 is
9      generic(
10          bus_length: INTEGER:= 24
11      );
12      port ( A,B:      in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
13             S:      in      STD_LOGIC;
14             Q:      out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
15 end MUX_2;
16

```

```

17
18 architecture behavioral of MUX_2 is
19
20 begin
21
22     Q <= A when S = '1' else B;
23
24 end behavioral;

```

Listing 2: MUX 2

7.2.2 MUX 3

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library IEEE;
6 use IEEE.std_logic_1164.all;
7
8 entity MUX_3 is
9     generic(
10         bus_length: INTEGER:= 49
11     );
12     port ( A,B,C:  in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
13             S:  in      STD_LOGIC_VECTOR (1 downto 0);
14             Q:  out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
15 end MUX_3;
16
17
18 architecture behavioral of MUX_3 is
19
20 begin
21
22     p_mux: process (S, A, B, C)
23     begin
24         case S is
25             when "00" =>
26                 Q <= A;
27             when "01" =>
28                 Q <= B;
29             when "10" =>
30                 Q <= C;
31             when "11" =>
32                 Q <= (others=>'0');
33             when others =>
34                 Q <= (others=>'0');
35         end case;
36     end process;
37
38 end behavioral;

```

Listing 3: MUX 3

7.3 Sottrattore

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library IEEE;
6 use IEEE.std_logic_1164.all;
7 use IEEE.numeric_std.all;
8

```

```

9
10 entity BFLY_SUBTRACTOR is
11     port ( A:      in      STD_LOGIC_VECTOR (48 downto 0);
12             B:      in      STD_LOGIC_VECTOR (48 downto 0);
13             CK:      in      STD_LOGIC;
14             DIFF_OUT:    out      STD_LOGIC_VECTOR(48 downto 0)
15             );
16 end BFLY_SUBTRACTOR;
17
18
19 architecture behavioral of BFLY_SUBTRACTOR is
20
21     signal diff: STD_LOGIC_VECTOR (48 downto 0) := (others=>'0');
22
23 begin
24
25     diff <= std_logic_vector(signed(A)-signed(B));
26
27     PSYNCH: process (CK)
28     begin
29         if CK'event and CK='1' then -- positive edge triggered:
30             DIFF_OUT <= diff;
31
32         end if;
33     end process;
34
35 end behavioral;

```

Listing 4: Sottrattore

7.4 Moltiplicatore/Shifter

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library IEEE;
6 use IEEE.std_logic_1164.all;
7 use IEEE.numeric_std.all;
8
9
10 entity BFLY_MULTIPLIER is
11     port ( A:      in      STD_LOGIC_VECTOR (23 downto 0);
12             B:      in      STD_LOGIC_VECTOR (23 downto 0);
13             SHIFT: in STD_LOGIC;
14             CK:      in      STD_LOGIC;
15             S_OUT:   out      STD_LOGIC_VECTOR(48 downto 0);
16             M_OUT:   out      STD_LOGIC_VECTOR(48 downto 0)
17             );
18 end BFLY_MULTIPLIER;
19
20
21 architecture behavioral of BFLY_MULTIPLIER is
22
23     signal op_A, op_B: STD_LOGIC_VECTOR (23 downto 0) := (others=>'0');
24     signal product: STD_LOGIC_VECTOR (47 downto 0) := (others=>'0');
25     signal S_OUT_tmp,M_OUT_tmp: STD_LOGIC_VECTOR (47 downto 0) := (others=>'0');
26     signal S_OUT_trunc,M_OUT_trunc: STD_LOGIC_VECTOR (46 downto 0) := (others=>'0');
27
28 begin
29
30     op_A <= A;
31     op_B <= "0000000000000000000000000010" when SHIFT = '1' else B;
32     product <= std_logic_vector(signed(op_A)*signed(op_B));
33     S_OUT_trunc <= S_OUT_tmp(46 downto 0);

```

```

34      M_OUT_trunc <= M_OUT_tmp(46 downto 0);
35      S_OUT <= '0' & '0' & S_OUT_trunc;
36      M_OUT <= '0' & '0' & M_OUT_trunc;
37
38
39      PSYNC: process(CK)
40      begin
41          if CK'event and CK='1' then -- positive edge triggered:
42              S_OUT_tmp <= product;
43              M_OUT_tmp <= S_OUT_tmp;
44
45          end if;
46      end process;
47
48 end behavioral;

```

Listing 5: Moltiplicatore/shifter

7.5 ROM rounding

7.5.1 Blocco ROM rounding

```

1  -- Federico Cobianchi - 332753
2  -- Onice Mazzi - 359754
3  -- Antonio Telmon - 353781
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7 use ieee.numeric_std.all;
8
9
10 -- Creazione entity
11 entity rounding is
12     port(
13         Clock: IN STD_LOGIC; -- Clock
14         rounding_in : IN std_logic_vector(48 downto 0); -- 49 bit da arrotondare
15         rounding_out: OUT std_logic_vector(23 downto 0); -- 24 bit arrotondati
16         shift_signal: IN STD_LOGIC -- segnale per shiftare
17     );
18 end entity;
19
20 -- Architecture del blocco rounding
21 architecture behavioural of rounding is
22
23     -----
24     -- Inizializzazione componenti
25     -----
26     component ROM is
27         port(
28             address : IN std_logic_vector(4 downto 0);
29             memory_out: OUT std_logic_vector(23 downto 0));
30         end component;
31
32     component FD is
33         generic(
34             bus_length: INTEGER:= 24
35         );
36         port ( D: in STD_LOGIC_VECTOR ((bus_length-1) downto 0);
37                 E: in STD_LOGIC; --ENABLE attivo alto
38                 CK: in STD_LOGIC;
39                 Q: out STD_LOGIC_VECTOR((bus_length-1) downto 0));
40         end component;
41
42     -----
43     -- Segnali interni al blocco rounding

```

```

44 -----
45
46 signal mantissa : std_logic_vector(23 downto 0):= (others=>'0');
47 signal dummy_memory_out: std_logic_vector(2 downto 0):= (others=>'0');
48 signal address_memory : std_logic_vector(4 downto 0):= (others=>'0');
49 signal reg_in : std_logic_vector(23 downto 0):= (others=>'0');
50 signal bit_scarto : std_logic_vector(1 downto 0):= (others=>'0');
51 signal shift_dummy: std_logic_vector(48 downto 0) := (others=>'0');
52
53 begin
54
55 -----
56 -- Port map
57 -----
58 pm_reg_rom_out : FD
59     generic map (
60         bus_length => 24
61     )
62     port map (
63         D => reg_in,
64         E => '1',
65         CK => Clock,
66         Q => rounding_out
67     );
68
69 pm_ROM : ROM
70     port map(
71         address => address_memory,
72         memory_out => dummy_memory_out
73     );
74
75 -----
76 -- Shift senza processo logico (non impiega colpi di clock)
77 -----
78 shift_dummy <=
79     '0' & '0' & rounding_in(48 downto 2) when shift_signal = '1' else '0' &
80     rounding_in(48 downto 1);
81
82 -----
83 -- Creazione mantissa e bit di scarto
84 -----
85
86 mantissa    <= shift_dummy(46 downto 23);
87 bit_scarto  <= shift_dummy(22 downto 21);
88
89 -----
90 -- Creazione dell'indirizzo per leggere dalla ROM
91 -----
92
93 -- address = (3 bit LSB mantissa) + (1 bit MSB scarto) + (1 bit OR con tutti gli
94 -- altri dello scarto)
95 address_memory <= mantissa(2 downto 0) & bit_scarto;
96
97 -----
98 -- Inserimento dati nel registro d'uscita del blocco
99 -----
100
100 reg_in <= mantissa(23 downto 3) & dummy_memory_out; -- 21 bit di mantissa & 3 bit
101 -- arrotondamento
102
103
104
105
106

```

```
107 | end architecture behavioural;
```

Listing 6: Blocco intero adibito al ROM rounding

7.5.2 Test Bench del ROM rounding

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7 use ieee.numeric_std.all;
8
9 entity tb_rounding is
10 end entity;
11
12 architecture sim of tb_rounding is
13
14 -- Segnali per collegare il DUT
15 signal Clock : std_logic := '0';
16 signal rounding_in : std_logic_vector(48 downto 0);
17 signal rounding_out : std_logic_vector(23 downto 0);
18 signal shift_signal: std_logic := '0';
19
20 constant Tclk : time := 10 ns;
21
22 begin
23
24 -- Istanziazione del DUT
25 DUT : entity work.rounding
26     port map (
27         Clock      => Clock,
28         rounding_in => rounding_in,
29         rounding_out => rounding_out,
30             shift_signal => shift_signal
31     );
32
33 -- Clock a 100 MHz
34 clk_proc : process
35 begin
36     Clock <= '0';
37     wait for Tclk/2;
38     Clock <= '1';
39     wait for Tclk/2;
40 end process;
41
42 stim_proc : process
43 begin
44     -- Caso 1
45     rounding_in <= "110010111010111010110000010001110010110011100111";
46     shift_signal <= '1';
47     wait for Tclk;
48     -- Caso 2
49     rounding_in <= "1000111101001011101000011001011001111100111000";
50     shift_signal <= '1';
51     wait for Tclk;
52     -- Caso 3
53     rounding_in <= "0001011101011011101000101111101110011101001000000";
54     shift_signal <= '0';
55     wait for Tclk;
56     -- Caso 4
57     rounding_in <= "1000010011010101101000100011100111010111000101101";
58     shift_signal <= '1';
59     wait for Tclk;

```

Listing 7: Test Bench del ROM rounding

7.5.3 ROM

```
1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7 use ieee.numeric_std.all;
8
9
10 -- Creazione entity
11
12 entity ROM is
13     port(
14         address : IN std_logic_vector(4 downto 0);
15         memory_out: OUT std_logic_vector(2 downto 0)
16     );
17 end entity;
18
19 -- Architecture della ROM
20
21 architecture ROM_rounding of ROM is
22
23     -- Spazio per segnali interni
```

```
24 begin
25 selection: process(address)
26 begin
27
28 if address = "00000" then
29     memory_out <= "000";
30 elsif address = "00001" then
31     memory_out <= "000";
32 elsif address = "00010" then
33     memory_out <= "000";
34 elsif address = "00011" then
35     memory_out <= "001";
36 elsif address = "00100" then
37     memory_out <= "001";
38 elsif address = "00101" then
39     memory_out <= "001";
40 elsif address = "00110" then
41     memory_out <= "010";
42 elsif address = "00111" then
43     memory_out <= "010";
44 elsif address = "01000" then
45     memory_out <= "010";
46 elsif address = "01001" then
47     memory_out <= "010";
48 elsif address = "01010" then
49     memory_out <= "010";
50 elsif address = "01011" then
51     memory_out <= "011";
52 elsif address = "01100" then
53     memory_out <= "011";
54 elsif address = "01101" then
55     memory_out <= "011";
56 elsif address = "01110" then
57     memory_out <= "100";
58 elsif address = "01111" then
59     memory_out <= "100";
60 elsif address = "10000" then
61     memory_out <= "100";
62 elsif address = "10001" then
63     memory_out <= "100";
64 elsif address = "10010" then
65     memory_out <= "100";
66 elsif address = "10011" then
67     memory_out <= "101";
68 elsif address = "10100" then
69     memory_out <= "101";
70 elsif address = "10101" then
71     memory_out <= "101";
72 elsif address = "10110" then
73     memory_out <= "110";
74 elsif address = "10111" then
75     memory_out <= "110";
76 elsif address = "11000" then
77     memory_out <= "110";
78 elsif address = "11001" then
79     memory_out <= "110";
80 elsif address = "11010" then
81     memory_out <= "110";
82 elsif address = "11011" then
83     memory_out <= "111";
84 elsif address = "11100" then
85     memory_out <= "111";
86 elsif address = "11101" then
87     memory_out <= "111";
```

```

90    elsif address = "11110" then
91        memory_out <= "111";
92    elsif address = "11111" then
93        memory_out <= "111";
94    end if;
95
96    end process;
97
98 end architecture ROM_rounding;

```

Listing 8: ROM

7.5.4 Test Bench della ROM

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7 use ieee.numeric_std.all;
8
9 entity TBROM is
10 end entity;
11
12 architecture sim of TBROM is
13
14     signal address : std_logic_vector(4 downto 0);
15     signal memory_out : std_logic_vector(2 downto 0);
16
17 begin
18
19     -- Istanziazione della ROM
20     DUT: entity work.ROM
21         port map (
22             address => address,
23             memory_out => memory_out
24         );
25
26     stim_proc: process
27     begin
28         for i in 0 to 31 loop
29             address <= std_logic_vector(to_unsigned(i, 5));
30             wait for 10 ns;
31         end loop;
32
33         -- Fine simulazione
34         wait;
35     end process;
36
37 end architecture sim;

```

Listing 9: Test Bench della ROM

7.6 Datapath

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library IEEE;
6 use IEEE.std_logic_1164.all; -- libreria IEEE con definizione tipi standard logic
7 use IEEE.numeric_std.all;
8

```

```

9
10 entity bfly_datapath is
11 port(
12     Br_in, Bi_in, Ar_in, Ai_in, Wr_in, Wi_in : in STD_LOGIC_VECTOR (23 downto 0);
13     Clock, START, SF_2H_1L : in STD_LOGIC;
14     Br_out, Bi_out, Ar_out, Ai_out : out STD_LOGIC_VECTOR (23 downto 0);
15     DONE : out STD_LOGIC
16 );
17 end bfly_datapath;
18
19 -----
20
21 architecture structural of bfly_datapath is
22
23 -----Inizializzazione componenti-----
24
25 -----
26
27 --Multiplier
28 component BFLY_MULTIPLIER is
29 port ( A:      in      STD_LOGIC_VECTOR (23 downto 0);
30        B:      in      STD_LOGIC_VECTOR (23 downto 0);
31        SHIFT:  in      STD_LOGIC;
32        CK:      in      STD_LOGIC;
33        S_OUT:   out    STD_LOGIC_VECTOR(48 downto 0);
34        M_OUT:   out    STD_LOGIC_VECTOR(48 downto 0)
35      );
36 end component;
37
38 --Adder
39 component BFLY_ADDER is
40 port ( A:      in      STD_LOGIC_VECTOR (48 downto 0);
41        B:      in      STD_LOGIC_VECTOR (48 downto 0);
42        CK:      in      STD_LOGIC;
43        SUM_OUT:  out    STD_LOGIC_VECTOR(48 downto 0)
44      );
45 end component;
46
47 --Sottrattore
48 component BFLY_SUBTRACTOR is
49 port ( A:      in      STD_LOGIC_VECTOR (48 downto 0);
50        B:      in      STD_LOGIC_VECTOR (48 downto 0);
51        CK:      in      STD_LOGIC;
52        DIFF_OUT: out    STD_LOGIC_VECTOR(48 downto 0)
53      );
54 end component;
55
56 --Registro FF con enable
57 component FD is
58 generic(
59     bus_length: INTEGER:= 24
60 );
61 port ( D:      in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
62        E: in STD_LOGIC;           --ENABLE attivo alto
63        CK:      in      STD_LOGIC;
64        Q:      out    STD_LOGIC_VECTOR((bus_length-1) downto 0));
65 end component;
66
67 --Flip Flop di tipo T con reset sincrono attivo alto
68 component T_FF is
69 port ( T:      in      STD_LOGIC;
70        R: in STD_LOGIC;           --RESET attivo alto
71        CK:      in      STD_LOGIC;
72        Q:      out    STD_LOGIC);
73 end component;
74

```

```

75      --Multiplexer a tre ingressi con due bit di select
76      component MUX_3 is
77          generic(
78              bus_length: INTEGER:= 49
79          );
80          port ( A,B,C:  in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
81                  S:  in      STD_LOGIC_VECTOR (1 downto 0);
82                  Q:  out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
83      end component;
84
85      --Multiplexer a due ingressi con un bit di select
86      component MUX_2 is
87          generic(
88              bus_length: INTEGER:= 24
89          );
90          port ( A,B:   in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
91                  S:  in      STD_LOGIC;
92                  Q:  out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
93      end component;
94
95      --Blocco unico di shift a destra e rom rounding
96      component rounding is
97          port(
98              Clock:  IN      STD_LOGIC; -- Clock
99              rounding_in : IN std_logic_vector(48 downto 0); -- 49 bit da arrotondare
100             rounding_out: OUT std_logic_vector(23 downto 0); -- 24 bit arrotondati
101             shift_signal: IN STD_LOGIC -- segnale per shiftare
102         );
103     end component;
104
105    --Control Unit
106    component BFLY_CU_DATAPATH is
107        port ( START:  in      STD_LOGIC;
108                SF_2H_1L:  in      STD_LOGIC;
109                CK:      in      STD_LOGIC;
110                INSTRUCTION_OUT:       out      STD_LOGIC_VECTOR(16 downto 0)
111            );
112    end component;
113
114
115  -----
116  --Dichiarazione segnali datapath
117  -----
118
119  --Segnali uIR
120  SIGNAL dp_SHIFT_SIGNAL , dp_REG_IN , dp_SUM_REG , dp_AR_SEL , dp_BR_SEL , dp_WR_SEL ,
121      dp_MS_DIFFP , dp_AS_SUM_SEL , dp_SD_ROUND_SEL , dp_SHIFT , dp_SF_2H_1L ,
122      dp_REG_RND_BR , dp_REG_RND_BI , dp_REG_RND_AR , dp_REG_RND_AI , dp_DONE :
123      STD_LOGIC := '0';
124  SIGNAL dp_MSD_DIFFM : STD_LOGIC_VECTOR (1 downto 0) := (others => '0');
125  SIGNAL dp_INSTRUCTION_OUT : STD_LOGIC_VECTOR (16 downto 0) := (others => '0');
126
127  --Ingressi al MUX di Br/Bi
128  SIGNAL dp_Br_MUX_in , dp_Bi_MUX_in : STD_LOGIC_VECTOR (23 downto 0) := (others =>
129      '0');
130  --Ingressi al MUX di Ar/Ai
131  SIGNAL dp_Ar_MUX_in , dp_Ai_MUX_in : STD_LOGIC_VECTOR (23 downto 0) := (others =>
132      '0');
133  --Ingressi al MUX di Wr/Wi
134  SIGNAL dp_Wr_MUX_in , dp_Wi_MUX_in : STD_LOGIC_VECTOR (23 downto 0) := (others =>
135      '0');
136
137  --Uscite dei MUX di B, A e W
138  SIGNAL dp_B_MUX_out , dp_A_MUX_out , dp_W_MUX_out : STD_LOGIC_VECTOR (23 downto 0)
139      := (others => '0');

```

```

134      --Uscite e ingressi del multiplier
135      SIGNAL dp_X_MPY_in, dp_Y_MPY_in : STD_LOGIC_VECTOR (23 downto 0) := (others =>
136          '0');
137      SIGNAL dp_MPY_product_out, dp_MPY_shift_out : STD_LOGIC_VECTOR (48 downto 0) := (
138          others => '0');

139      --Uscita e ingressi dell'adder
140      SIGNAL dp_SUM_out, dp_X_SUM_in, dp_Y_SUM_in : STD_LOGIC_VECTOR (48 downto 0) := (
141          others => '0');

142      --Uscita e ingressi del sottrattore
143      SIGNAL dp_DIFF_out, dp_X_DIFF_in, dp_Y_DIFF_in : STD_LOGIC_VECTOR (48 downto 0) := (
144          others => '0');

145      --Uscita del registro di pipe della somma
146      SIGNAL dp_SUM_reg_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');

147      --Uscita del registro di pipe del sottrattore
148      SIGNAL dp_DIFF_reg_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');

149      --Uscita del registro di pipe del prodotto e dello shift
150      SIGNAL dp_MPY_M_reg_out, dp_MPY_S_reg_out : STD_LOGIC_VECTOR (48 downto 0) := (
151          others => '0');

152      --Ingresso 1 del multiplexer in entrata al sommatore, ovvero l'uscita del MUX di
153      --Ar/Ai con zeri aggiunti
154      SIGNAL dp_AS_A_MUX_in : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');

155      --Uscita del multiplexer in entrata al sommatore
156      SIGNAL dp_AS_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');

157      --Uscita del MUX A/B in ingresso al multiplier
158      SIGNAL dp_AB_MUX_out : STD_LOGIC_VECTOR (23 downto 0) := (others => '0');

159      --Uscita del MUX dell'ingresso positivo del sottrattore
160      SIGNAL dp_MS_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');

161      --Uscita del MUX dell'ingresso negativo del sottrattore
162      SIGNAL dp_MSD_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');

163      --Uscita del MUX dell'ingresso dello shifter a destra
164      SIGNAL dp_SD_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');

165      --Uscita del blocco shift + rom rounding
166      SIGNAL dp_ROM_round_out : STD_LOGIC_VECTOR (23 downto 0) := (others => '0');

167      begin
168
169      -----
170      --Port map dei registri a 24 bit
171      -----
172
173      pm_regin_Br : FD
174          generic map (
175              bus_length => 24
176          )
177          port map (
178              D => Br_in,
179              E => dp_REG_IN,
180              CK => Clock,
181              Q => dp_Br_MUX_in
182          );
183
184      pm_regin_Bi : FD
185          generic map (
186              bus_length => 24
187          )
188
189
190
191
192
193

```

```

194      port map (
195          D => Bi_in,
196          E => dp_REG_IN,
197          CK => Clock,
198          Q => dp_Bi_MUX_in
199      );
200
201      pm_regin_Ar : FD
202          generic map (
203              bus_length => 24
204          )
205          port map (
206              D => Ar_in,
207              E => dp_REG_IN,
208              CK => Clock,
209              Q => dp_Ar_MUX_in
210      );
211
212      pm_regin_Ai : FD
213          generic map (
214              bus_length => 24
215          )
216          port map (
217              D => Ai_in,
218              E => dp_REG_IN,
219              CK => Clock,
220              Q => dp_Ai_MUX_in
221      );
222
223      pm_regin_Wr : FD
224          generic map (
225              bus_length => 24
226          )
227          port map (
228              D => Wr_in,
229              E => dp_REG_IN,
230              CK => Clock,
231              Q => dp_Wr_MUX_in
232      );
233
234      pm_regin_Wi : FD
235          generic map (
236              bus_length => 24
237          )
238          port map (
239              D => Wi_in,
240              E => dp_REG_IN,
241              CK => Clock,
242              Q => dp_Wi_MUX_in
243      );
244
245      -----
246      --Port map dei Multiplexer a due ingressi
247      -----
248
249      pm_mux_B : MUX_2
250          generic map (
251              bus_length => 24
252          )
253          port map (
254              A => dp_Br_MUX_in,
255              B => dp_Bi_MUX_in,
256              S => dp_BR_SEL,
257              Q => dp_B_MUX_out
258      );
259

```

```

260      pm_mux_A : MUX_2
261      generic map (
262          bus_length => 24
263      )
264      port map (
265          A => dp_Ar_MUX_in,
266          B => dp_Ai_MUX_in,
267          S => dp_AR_SEL,
268          Q => dp_A_MUX_out
269      );
270
271      pm_mux_W : MUX_2
272      generic map (
273          bus_length => 24
274      )
275      port map (
276          A => dp_Wr_MUX_in,
277          B => dp_Wi_MUX_in,
278          S => dp_WR_SEL,
279          Q => dp_W_MUX_out
280      );
281
282      pm_mux_Mult : MUX_2      --Multiplexer del multiplier
283      generic map (
284          bus_length => 24
285      )
286      port map (
287          A => dp_A_MUX_out,
288          B => dp_B_MUX_out,
289          S => dp_SHIFT,
290          Q => dp_AB_MUX_out
291      );
292
293      --Ingresso 1 del multiplexer in entrata al sommatore, ovvero l'uscita del MUX di
294      --Ar/Ai
295      dp_AS_A_MUX_in (48 downto 24) <= (others => '0');           --Aggiungo zeri perche' l'
296      --uscita del MUX di Ar/Ai e' solo su 24 bit
297      dp_AS_A_MUX_in (23 downto 0) <= dp_A_MUX_out;
298
299      pm_mux_Adder : MUX_2      --Multiplexer dell'adder
300      generic map (
301          bus_length => 49
302      )
303      port map (
304          A => dp_AS_A_MUX_in,      --l'uscita del MUX Ar/Ai
305          B => dp_SUM_reg_out,    --l'uscita del sommatore rallentata di un colpo di
306          Clock
307          S => dp_AS_SUM_SEL,
308          Q => dp_AS_MUX_out
309      );
310
311      pm_mux_Sub_plus : MUX_2      --Multiplexer dell'ingresso positivo del
312      sottrattore
313      generic map (
314          bus_length => 49
315      )
316      port map (
317          A => dp_MPY_S_reg_out,  --l'uscita SHIFT del moltiplicatore
318          B => dp_SUM_reg_out,   --l'uscita del sommatore rallentata di un colpo di
319          Clock
320          S => dp_MS_DIFFp,
            Q => dp_MS_MUX_out
      );
321
322      pm_mux_rshift : MUX_2      --Multiplexer dell'ingresso allo shifter a destra
323      generic map (

```

```

321         bus_length => 49
322     )
323     port map (
324         A => dp_SUM_reg_out,           --l'uscita del sommatore
325         B => dp_DIFF_reg_out,        --l'uscita del sottrattore
326         S => dp_SD_ROUND_SEL,
327         Q => dp_SD_MUX_out
328     );
329
330     --Port map del MUX a tre ingressi
331     pm_mux_Sub_minus : MUX_3          --Multiplexer dell'ingresso negativo del
332             sottrattore
333     generic map (
334         bus_length => 49
335     )
336     port map (
337         A => dp_MPY_M_reg_out,          --l'uscita MPY del moltiplicatore
338             rallentata di un colpo di Clock
339         B => dp_SUM_reg_out,           --l'uscita del sommatore
340             rallentata di un colpo di Clock
341         C => dp_DIFF_reg_out,          --l'uscita del sottrattore
342             rallentata di un colpo di Clock
343         S => dp_MSD_DIFFm,
344         Q => dp_MSD_MUX_out
345     );
346
347     -----
348     --Port map degli operatori
349     -----
350
351     dp_X_MPY_in <= dp_AB_MUX_out;    --L'ingresso 1 del multiplier e' connesso all'
352             uscita del multiplexer A/B
353     dp_Y_MPY_in <= dp_W_MUX_out;      --L'ingresso 2 del multiplier e' connesso all'
354             uscita del multiplexer Wr/Wi
355
356     pm_Multiplier : BFLY_MULTIPLIER --Port map del multiplier
357     port map (
358         A => dp_X_MPY_in,
359         B => dp_Y_MPY_in,
360         SHIFT => dp_SHIFT,
361         CK => Clock,
362         M_OUT => dp_MPY_product_out,
363         S_OUT => dp_MPY_shift_out
364     );
365
366     dp_X_SUM_in <= dp_AS_MUX_out;      --L'ingresso 1 dell'adder e' connesso all'
367             uscita del multiplexer A/Somma
368     dp_Y_SUM_in <= dp_MPY_M_reg_out;    --L'ingresso 2 dell'adder e' connesso all'
369             uscita moltiplicazione del multiplier
370
371     pm_Adder : BFLY_ADDER      --Port map dell'adder
372     port map (
373         A => dp_X_SUM_in,
374         B => dp_Y_SUM_in,
375         CK => Clock,
376         SUM_OUT => dp_SUM_out
377     );
378
379     dp_X_DIFF_in <= dp_MS_MUX_out;
380     dp_Y_DIFF_in <= dp_MSD_MUX_out;
381
382     pm_Subractor : BFLY_SUBTRACTOR --Port map del sottrattore
383     port map (
384         A => dp_X_DIFF_in,
385         B => dp_Y_DIFF_in,
386         CK => Clock.

```

```

379           DIFF_OUT => dp_DIFF_out
380       );
381
382       pm_ft_shift : T_FF      --Port map del flip flop T che ha come uscita il segnale
383           di SF_2H_1L per il blocco rounding
384       port map (
385           T => dp_SHIFT_SIGNAL,    --Segnale che viene dalla CU
386           R => dp_DONE,           --Segnale che viene dalla CU
387           CK => Clock,
388           Q => dp_SF_2H_1L
389       );
390
390       pm_rounding : rounding   --Port map del blocco unico shifter a destra e ROM
391           rounding
392       port map (
393           Clock => Clock,
394           rounding_in => dp_SD_MUX_out,
395           rounding_out => dp_ROM_round_out,
396           shift_signal => dp_SF_2H_1L
397       );
398
398       pm_CU : BFLY CU DATAPATH      --Port map della Control unit
399       port map (
400           START => START,
401           SF_2H_1L => SF_2H_1L,
402           CK => Clock,
403           INSTRUCTION_OUT => dp_INSTRUCTION_OUT
404       );
405
406       --Segnali della parte di istruzione del uIR della CU
407       dp_SHIFT_SIGNAL <= dp_INSTRUCTION_OUT(16);
408       dp_REG_IN <= dp_INSTRUCTION_OUT(15);
409       dp_SUM_REG <= dp_INSTRUCTION_OUT(14);
410       dp_AR_SEL <= dp_INSTRUCTION_OUT(13);
411       dp_BR_SEL <= dp_INSTRUCTION_OUT(12);
412       dp_WR_SEL <= dp_INSTRUCTION_OUT(11);
413       dp_MS_DIFFp <= dp_INSTRUCTION_OUT(10);
414       dp_MSD_DIFFm <= dp_INSTRUCTION_OUT(9 downto 8);
415       dp_AS_SUM_SEL <= dp_INSTRUCTION_OUT(7);
416       dp_SD_ROUND_SEL <= dp_INSTRUCTION_OUT(6);
417       dp_REG_RND_BR <= dp_INSTRUCTION_OUT(5);
418       dp_REG_RND_BI <= dp_INSTRUCTION_OUT(4);
419       dp_REG_RND_AR <= dp_INSTRUCTION_OUT(3);
420       dp_REG_RND_AI <= dp_INSTRUCTION_OUT(2);
421       dp_SHIFT <= dp_INSTRUCTION_OUT(1);
422       dp_DONE <= dp_INSTRUCTION_OUT(0);
423
424       DONE <= dp_DONE;
425
426
427       -----
428       --Port map dei registri a 49 bit
429       -----
430
431       pm_reg_MPY_product_out : FD      --Port map del registro all'uscita prodotto del
432           multiplier
433           generic map (
434               bus_length => 49
435           )
436           port map (
437               D => dp_MPY_product_out,
438               E => '1',
439               CK => Clock,
440               Q => dp_MPY_M_reg_out
441           );

```

```

442      pm_reg_MPY_shift_out : FD           --Port map del registro all'uscita shift del
443          multiplier
444              generic map (
445                  bus_length => 49
446              )
447              port map (
448                  D => dp_MPY_shift_out,
449                  E => '1',
450                  CK => Clock,
451                  Q => dp_MPY_S_reg_out
452      );
453
454      pm_reg_SUM_out : FD           --Port map del registro all'uscita del sommatore
455          generic map (
456              bus_length => 49
457          )
458          port map (
459              D => dp_SUM_out,
460              E => '1',
461              CK => Clock,
462              Q => dp_SUM_reg_out
463      );
464
465      pm_reg_DIFF_out : FD           --Port map del registro all'uscita del sottrattore
466          generic map (
467              bus_length => 49
468          )
469          port map (
470              D => dp_DIFF_out,
471              E => '1',
472              CK => Clock,
473              Q => dp_DIFF_reg_out
474      );
475
476      -----
477      --Port map dei registri di uscita a 24 bit
478      -----
479
480      pm_regout_Br : FD
481          generic map (
482              bus_length => 24
483          )
484          port map (
485              D => dp_ROM_round_out,
486              E => dp_REG_RND_BR,
487              CK => Clock,
488              Q => Br_out
489      );
490
491      pm_regout_Bi : FD
492          generic map (
493              bus_length => 24
494          )
495          port map (
496              D => dp_ROM_round_out,
497              E => dp_REG_RND_BI,
498              CK => Clock,
499              Q => Bi_out
500      );
501
502      pm_regout_Ar : FD
503          generic map (
504              bus_length => 24
505          )
506          port map (
507              D => dp_ROM_round_out,

```

```

507      E => dp_REG_RND_AR ,
508      CK => Clock ,
509      Q => Ar_out
510  );
511
512  pm_regout_Ai : FD
513    generic map (
514      bus_length => 24
515    )
516    port map (
517      D => dp_ROM_round_out ,
518      E => dp_REG_RND_AI ,
519      CK => Clock ,
520      Q => Ai_out
521  );
522
523 end structural;

```

Listing 10: Datapath

7.7 Control Unit

7.7.1 Datapath

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library IEEE;
6 use IEEE.std_logic_1164.all;
7 use IEEE.numeric_std.all;
8
9
10 entity BFLY CU DATAPATH is
11   port ( START:  in     STD_LOGIC;
12          SF_2H_1L: in     STD_LOGIC;
13          CK:      in     STD_LOGIC;
14          INSTRUCTION_OUT:       out    STD_LOGIC_VECTOR(16 downto 0)
15        );
16 end BFLY CU DATAPATH;
17
18
19 architecture structural of BFLY CU DATAPATH is
20
21   component BFLY CU LATE_STATUS_PLA is
22     port ( STATUS: in STD_LOGIC_VECTOR(1 downto 0);
23            LSB_in: in STD_LOGIC;
24            CC_Validation_in: in STD_LOGIC;
25            CC_Validation_out: out STD_LOGIC;
26            LSB_out: out STD_LOGIC
27          );
28   end component;
29
30   component BFLY CU ROM is
31     generic(
32       in_length: INTEGER:= 3;
33       next_Address_length :INTEGER := 4;
34       out_length: INTEGER:= 22
35     );
36     port ( A:      in     STD_LOGIC_VECTOR ((in_length-1) downto 0);
37            OUT_EVEN:   out    STD_LOGIC_VECTOR((out_length-1) downto 0);
38            OUT_ODD:   out    STD_LOGIC_VECTOR((out_length-1) downto 0)
39          );
40   end component;
41

```

```

42      component FD is
43        generic(
44          bus_length: INTEGER:= 24
45        );
46        port ( D:      in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
47              E:      in      STD_LOGIC;           --ENABLE attivo alto
48              CK:     in      STD_LOGIC;
49              Q:      out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
50      end component;
51
52      component MUX_2 is
53        generic(
54          bus_length: INTEGER:= 24
55        );
56        port ( A,B:      in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
57              S:      in      STD_LOGIC;
58              Q:      out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
59      end component;
60
61      SIGNAL microAR_in_MSB : STD_LOGIC_VECTOR (3 downto 1) := (others=>'0');
62      SIGNAL microAR_out_MSB : STD_LOGIC_VECTOR (3 downto 1) := (others=>'0');
63      SIGNAL microAR_in_LSB : STD_LOGIC := '0';
64      SIGNAL microAR_out_LSB : STD_LOGIC := '0';
65
66      SIGNAL CC_mux_out : STD_LOGIC := '0';
67
68      SIGNAL PLA_ROM_out_even, PLA_ROM_out_odd : STD_LOGIC_VECTOR (21 downto 0):= (
69        others=>'0');
70
71      SIGNAL PLA_ROM_mux_out : STD_LOGIC_VECTOR (21 downto 0):= (others=>'0');
72
73      SIGNAL status_PLA_LSB_out : STD_LOGIC := '0';
74      SIGNAL status_PLA_CC_validation_out : STD_LOGIC := '0';
75
76      SIGNAL microIR_in : STD_LOGIC_VECTOR (21 downto 0)      := (others=>'0');
77      SIGNAL microIR_out : STD_LOGIC_VECTOR (21 downto 0)      := (others=>'0');
78
79      SIGNAL CC_validation : STD_LOGIC := '0';
80      SIGNAL next_Address_LSB : STD_LOGIC := '0';
81      SIGNAL next_Address_MSB : STD_LOGIC_VECTOR (2 downto 0) := (others=>'0');
82
83      SIGNAL dp_STATUS : STD_LOGIC_VECTOR (1 downto 0) := (others=>'0');
84
85      begin
86
86        dp_STATUS(0) <= START;
87        dp_STATUS(1) <= SF_2H_1L;
88
89        INSTRUCTION_OUT <= microIR_out (20 downto 4);
90        CC_validation <= microIR_out (21);
91        next_Address_LSB <= microIR_out (0);
92
93        next_Address_MSB(2 downto 0) <= microIR_out (3 downto 1);
94
95        microAR_in_MSB <= next_Address_MSB;
96        microAR_in_LSB <= status_PLA_LSB_out;
97
98        microIR_in <= PLA_ROM_mux_out;
99
100       --PLA
101       pm_PLA : BFLY_CU_LATE_STATUS_PLA
102       port map (
103         STATUS => dp_STATUS,
104         LSB_in => next_Address_LSB,
105         CC_Validation_in => CC_validation,
106         CC_Validation_out => status_PLA_CC_validation_out,

```

```

107         LSB_out => status_PLA_LSB_out
108     );
109
110     --ROM della PLA
111     pm_CU_ROM : BFLY CU ROM
112     generic map(
113         in_length => 3,
114         next_Address_length => 3,
115         out_length => 22
116     )
117     port map (
118         A => microAR_out_MSB,
119         OUT_EVEN => PLA_ROM_out_even,
120         OUT_ODD => PLA_ROM_out_odd
121     );
122
123     --Registro del uAR eccetto l'LSB
124     pm_microAR_MSB_reg : FD
125     generic map (
126         bus_length => 3
127     )
128     port map (
129         D => microAR_in_MSB,
130         E => '1',
131         CK => CK,
132         Q => microAR_out_MSB
133     );
134
135     --Registro dell'LSB del uAR
136     FF_D_uAR: process(CK)
137     begin
138         if CK'event and CK='1' then -- positive edge triggered:
139             microAR_out_LSB <= microAR_in_LSB;
140         end if;
141     end process;
142
143     --Registro del uIR
144     pm_microIR_reg : FD
145     generic map (
146         bus_length => 22
147     )
148     port map (
149         D => microIR_in,
150         E => '1',
151         CK => CK,
152         Q => microIR_out
153     );
154
155     --MUX a due ingressi a 21 bit, che seleziona tra l'uscita pari o dispari della ROM
156     pm_ROM_mux : MUX_2
157     generic map (
158         bus_length => 22
159     )
160     port map (
161         A => PLA_ROM_out_odd,
162         B => PLA_ROM_out_even,
163         S => CC_mux_out,
164         Q => PLA_ROM_mux_out
165     );
166
167     --MUX a due ingressi a 1 bit
168     --L'uscita e' il segnale di select per il MUX even/odd della ROM
169     CC_mux_out <= microAR_out_LSB when status_PLA_CC_validation_out = '0' else
170         status_PLA_LSB_out;
171

```

```
172 | end structural;
```

Listing 11: Control Unit datapath

7.7.2 ROM

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library IEEE;
6 use IEEE.std_logic_1164.all;
7 use IEEE.numeric_std.all;
8
9
10 entity BFLY CU ROM is
11     generic(
12         in_length: INTEGER:= 3;
13         next_Address_length :INTEGER := 3;
14         out_length: INTEGER:= 22
15     );
16     port ( A:      in      STD_LOGIC_VECTOR ((in_length-1) downto 0);
17            OUT_EVEN:    out      STD_LOGIC_VECTOR((out_length-1) downto 0);
18            OUT_ODD:    out      STD_LOGIC_VECTOR((out_length-1) downto 0)
19        );
20 end BFLY CU ROM;
21
22
23 architecture behavioral of BFLY CU ROM is
24
25     SIGNAL out_tmp_even, out_tmp_odd : STD_LOGIC_VECTOR ((out_length-1) downto 0) := (
26         others=>'0');
27     SIGNAL next_Address_even, next_Address_odd : STD_LOGIC_VECTOR ((next_Address_length-1) downto 0) := (others=>'0');
28
29     SIGNAL REG_IN_even, SUM_REG_even, AR_SEL_even, BR_SEL_even, WR_SEL_even,
30     MS_DIFFp_even, AS_SUM_SEL_even, SD_ROUND_SEL_even, REG_RND_BR_even,
31     REG_RND_BI_even, REG_RND_AR_even, REG_RND_AI_even, SHIFT_even, DONE_even :
32     STD_LOGIC := '0';
33     SIGNAL REG_IN_odd, SUM_REG_odd, AR_SEL_odd, BR_SEL_odd, WR_SEL_odd, MS_DIFFp_odd,
34     AS_SUM_SEL_odd, SD_ROUND_SEL_odd, REG_RND_BR_odd, REG_RND_BI_odd,
35     REG_RND_AR_odd, REG_RND_AI_odd, SHIFT_odd, DONE_odd : STD_LOGIC := '0';
36     SIGNAL SF_2H_1L_even, SF_2H_1L_odd : STD_LOGIC := '0';
37
38
39     SIGNAL MSD_DIFFm_even, MSD_DIFFm_odd : STD_LOGIC_VECTOR (1 downto 0) := "00";
40
41     SIGNAL CC_Validation_even, CC_Validation_odd : STD_LOGIC := '0';
42
43 begin
44
45     OUT_EVEN <= out_tmp_even;
46     OUT_ODD <= out_tmp_odd;
47
48
49     --CC validation
50     out_tmp_even(21) <= CC_Validation_even;
51
52     --Instruction part
53     out_tmp_even(20) <= SF_2H_1L_even;
54     out_tmp_even(19) <= REG_IN_even;
55     out_tmp_even(18) <= SUM_REG_even;
56     out_tmp_even(17) <= AR_SEL_even;
57     out_tmp_even(16) <= BR_SEL_even;
58     out_tmp_even(15) <= WR_SEL_even;

```

```

53     out_tmp_even(14) <= MS_DIFFp_even;
54     out_tmp_even(13 downto 12) <= MSD_DIFFm_even;
55     out_tmp_even(11) <= AS_SUM_SEL_even;
56     out_tmp_even(10) <= SD_ROUND_SEL_even;
57     out_tmp_even(9) <= REG_RND_BR_even;
58     out_tmp_even(8) <= REG_RND_BI_even;
59     out_tmp_even(7) <= REG_RND_AR_even;
60     out_tmp_even(6) <= REG_RND_AI_even;
61     out_tmp_even(5) <= SHIFT_even;
62     out_tmp_even(4) <= DONE_even;

63
64     --Next address
65     out_tmp_even((next_Address_length) downto 1) <= next_Address_even;
66     out_tmp_even(0) <= '0';

67
68
69
70
71     --CC validation
72     out_tmp_odd(21) <= CC_Validation_odd;
73
74     --Instruction part
75     out_tmp_odd(20) <= SF_2H_1L_odd;
76     out_tmp_odd(19) <= REG_IN_odd;
77     out_tmp_odd(18) <= SUM_REG_odd;
78     out_tmp_odd(17) <= AR_SEL_odd;
79     out_tmp_odd(16) <= BR_SEL_odd;
80     out_tmp_odd(15) <= WR_SEL_odd;
81     out_tmp_odd(14) <= MS_DIFFp_odd;
82     out_tmp_odd(13 downto 12) <= MSD_DIFFm_odd;
83     out_tmp_odd(11) <= AS_SUM_SEL_odd;
84     out_tmp_odd(10) <= SD_ROUND_SEL_odd;
85     out_tmp_odd(9) <= REG_RND_BR_odd;
86     out_tmp_odd(8) <= REG_RND_BI_odd;
87     out_tmp_odd(7) <= REG_RND_AR_odd;
88     out_tmp_odd(6) <= REG_RND_AI_odd;
89     out_tmp_odd(5) <= SHIFT_odd;
90     out_tmp_odd(4) <= DONE_odd;

91
92     --Next address
93     out_tmp_odd((next_Address_length) downto 1) <= next_Address_odd;
94     out_tmp_odd(0) <= '1';

95
96     p_rom : process (A)
97 begin
98     if A = "000" then
99
100         --IDLE
101         SF_2H_1L_even <= '0';
102         CC_Validation_even <= '0';
103         REG_IN_even <= '0';
104         SUM_REG_even <= '0';
105         AR_SEL_even <= '0';
106         BR_SEL_even <= '0';
107         WR_SEL_even <= '0';
108         MS_DIFFp_even <= '0';
109         MSD_DIFFm_even <= "00";
110         AS_SUM_SEL_even <= '0';
111         SD_ROUND_SEL_even <= '0';
112         REG_RND_BR_even <= '0';
113         REG_RND_BI_even <= '0';
114         REG_RND_AR_even <= '0';
115         REG_RND_AI_even <= '0';
116         SHIFT_even <= '0';
117         DONE_even <= '0';
118         next_Address_even <= "000";

```

```

119
120      --START
121      SF_2H_1L_odd <= '0';
122      CC_Validation_odd <= '1';
123      REG_IN_odd <= '1';
124      SUM_REG_odd <= '0';
125      AR_SEL_odd <= '0';
126      BR_SEL_odd <= '0';
127      WR_SEL_odd <= '0';
128      MS_DIFFp_odd <= '0';
129      MSD_DIFFm_odd <= "00";
130      AS_SUM_SEL_odd <= '0';
131      SD_ROUND_SEL_odd <= '0';
132      REG_RND_BR_odd <= '0';
133      REG_RND_BI_odd <= '0';
134      REG_RND_AR_odd <= '0';
135      REG_RND_AI_odd <= '0';
136      SHIFT_odd <= '0';
137      DONE_odd <= '0';
138      next_Address_odd <= "001";
139
140
141      elsif A = "001" then
142
143          --M1 , SH0
144          SF_2H_1L_even <= '0';
145          CC_Validation_even <= '0';
146          REG_IN_even <= '0';
147          SUM_REG_even <= '0';
148          AR_SEL_even <= '0';
149          BR_SEL_even <= '1';
150          WR_SEL_even <= '1';
151          MS_DIFFp_even <= '0';
152          MSD_DIFFm_even <= "00";
153          AS_SUM_SEL_even <= '0';
154          SD_ROUND_SEL_even <= '0';
155          REG_RND_BR_even <= '0';
156          REG_RND_BI_even <= '0';
157          REG_RND_AR_even <= '0';
158          REG_RND_AI_even <= '0';
159          SHIFT_even <= '0';
160          DONE_even <= '0';
161          next_Address_even <= "010";
162
163          --M1 , SH1
164          SF_2H_1L_odd <= '1';
165          CC_Validation_odd <= '0';
166          REG_IN_odd <= '0';
167          SUM_REG_odd <= '0';
168          AR_SEL_odd <= '0';
169          BR_SEL_odd <= '1';
170          WR_SEL_odd <= '1';
171          MS_DIFFp_odd <= '0';
172          MSD_DIFFm_odd <= "00";
173          AS_SUM_SEL_odd <= '0';
174          SD_ROUND_SEL_odd <= '0';
175          REG_RND_BR_odd <= '0';
176          REG_RND_BI_odd <= '0';
177          REG_RND_AR_odd <= '0';
178          REG_RND_AI_odd <= '0';
179          SHIFT_odd <= '0';
180          DONE_odd <= '0';
181          next_Address_odd <= "010";
182
183      elsif A = "010" then
184

```

```

185
186      --M2
187      SF_2H_1L_even <= '0';
188      CC_Validation_even <= '1';
189      REG_IN_even <= '0';
190      SUM_REG_even <= '0';
191      AR_SEL_even <= '0';
192      BR_SEL_even <= '1';
193      WR_SEL_even <= '0';
194      MS_DIFFp_even <= '0';
195      MSD_DIFFm_even <= "00";
196      AS_SUM_SEL_even <= '0';
197      SD_ROUND_SEL_even <= '0';
198      REG_RND_BR_even <= '0';
199      REG_RND_BI_even <= '0';
200      REG_RND_AR_even <= '0';
201      REG_RND_AI_even <= '0';
202      SHIFT_even <= '0';
203      DONE_even <= '0';
204      next_Address_even <= "010";
205
206      --M3
207      SF_2H_1L_odd <= '0';
208      CC_Validation_odd <= '0';
209      REG_IN_odd <= '0';
210      SUM_REG_odd <= '0';
211      AR_SEL_odd <= '0';
212      BR_SEL_odd <= '0';
213      WR_SEL_odd <= '0';
214      MS_DIFFp_odd <= '0';
215      MSD_DIFFm_odd <= "00";
216      AS_SUM_SEL_odd <= '0';
217      SD_ROUND_SEL_odd <= '0';
218      REG_RND_BR_odd <= '0';
219      REG_RND_BI_odd <= '0';
220      REG_RND_AR_odd <= '0';
221      REG_RND_AI_odd <= '0';
222      SHIFT_odd <= '0';
223      DONE_odd <= '0';
224      next_Address_odd <= "011";
225
226
227      elsif A = "011" then          --M4 ,S1 / S2
228
229          --M4 , S1
230          SF_2H_1L_even <= '0';
231          CC_Validation_even <= '1';
232          REG_IN_even <= '0';
233          SUM_REG_even <= '0';
234          AR_SEL_even <= '1';
235          BR_SEL_even <= '0';
236          WR_SEL_even <= '1';
237          MS_DIFFp_even <= '0';
238          MSD_DIFFm_even <= "00";
239          AS_SUM_SEL_even <= '1';
240          SD_ROUND_SEL_even <= '0';
241          REG_RND_BR_even <= '0';
242          REG_RND_BI_even <= '0';
243          REG_RND_AR_even <= '0';
244          REG_RND_AI_even <= '0';
245          SHIFT_even <= '0';
246          DONE_even <= '0';
247          next_Address_even <= "011";
248
249          --S2
250          SF_2H_1L_odd <= '0';

```

```

251      CC_Validation_odd <= '0';
252      REG_IN_odd <= '0';
253      SUM_REG_odd <= '0';
254      AR_SEL_odd <= '0';
255      BR_SEL_odd <= '0';
256      WR_SEL_odd <= '0';
257      MS_DIFFp_odd <= '0';
258      MSD_DIFFm_odd <= "00";
259      AS_SUM_SEL_odd <= '1';
260      SD_ROUND_SEL_odd <= '0';
261      REG_RND_BR_odd <= '0';
262      REG_RND_BI_odd <= '0';
263      REG_RND_AR_odd <= '0';
264      REG_RND_AI_odd <= '0';
265      SHIFT_odd <= '0';
266      DONE_odd <= '0';
267      next_Address_odd <= "100";
268
269
270      elsif A = "100" then --M5 ,D1 / M6 ,S3
271
272          --M5 , D1
273          SF_2H_1L_even <= '0';
274          CC_Validation_even <= '1';
275          REG_IN_even <= '0';
276          SUM_REG_even <= '0';
277          AR_SEL_even <= '1';
278          BR_SEL_even <= '0';
279          WR_SEL_even <= '0';
280          MS_DIFFp_even <= '0';
281          MSD_DIFFm_even <= "00";
282          AS_SUM_SEL_even <= '0';
283          SD_ROUND_SEL_even <= '0';
284          REG_RND_BR_even <= '0';
285          REG_RND_BI_even <= '0';
286          REG_RND_AR_even <= '0';
287          REG_RND_AI_even <= '0';
288          SHIFT_even <= '1';
289          DONE_even <= '0';
290          next_Address_even <= "100";
291
292          --M6 , S3
293          SF_2H_1L_odd <= '0';
294          CC_Validation_odd <= '0';
295          REG_IN_odd <= '0';
296          SUM_REG_odd <= '0';
297          AR_SEL_odd <= '0';
298          BR_SEL_odd <= '0';
299          WR_SEL_odd <= '0';
300          MS_DIFFp_odd <= '0';
301          MSD_DIFFm_odd <= "00"; --Product
302          AS_SUM_SEL_odd <= '0';
303          SD_ROUND_SEL_odd <= '0';
304          REG_RND_BR_odd <= '0';
305          REG_RND_BI_odd <= '0';
306          REG_RND_AR_odd <= '0';
307          REG_RND_AI_odd <= '0';
308          SHIFT_odd <= '1';
309          DONE_odd <= '0';
310          next_Address_odd <= "101";
311
312
313      elsif A = "101" then --D2 ,SH1 / D3 ,SH2
314
315          --D2 , SH1
316          SF_2H_1L_even <= '0';

```

```

317     CC_Validation_even <= '1';
318     REG_IN_even <= '0';
319     SUM_REG_even <= '0';
320     AR_SEL_even <= '0';
321     BR_SEL_even <= '0';
322     WR_SEL_even <= '0';
323     MS_DIFFp_even <= '1';
324     MSD_DIFFm_even <= "10";          --Difference
325     AS_SUM_SEL_even <= '0';
326     SD_ROUND_SEL_even <= '0';
327     REG_RND_BR_even <= '0';
328     REG_RND_BI_even <= '0';
329     REG_RND_AR_even <= '0';
330     REG_RND_AI_even <= '0';
331     SHIFT_even <= '0';
332     DONE_even <= '0';
333     next_Address_even <= "101";
334
335     --D3 , SH2
336     SF_2H_1L_odd <= '0';
337     CC_Validation_odd <= '0';
338     REG_IN_odd <= '0';
339     SUM_REG_odd <= '0';
340     AR_SEL_odd <= '0';
341     BR_SEL_odd <= '0';
342     WR_SEL_odd <= '0';
343     MS_DIFFp_odd <= '1';
344     MSD_DIFFm_odd <= "01";          --Sum
345     AS_SUM_SEL_odd <= '0';
346     SD_ROUND_SEL_odd <= '1';
347     REG_RND_BR_odd <= '0';
348     REG_RND_BI_odd <= '0';
349     REG_RND_AR_odd <= '1';
350     REG_RND_AI_odd <= '0';
351     SHIFT_odd <= '0';
352     DONE_odd <= '0';
353     next_Address_odd <= "110";
354
355
356     elsif A = "110" then           --SH3 / SH4
357
358     --SH3
359     SF_2H_1L_even <= '0';
360     CC_Validation_even <= '1';
361     REG_IN_even <= '0';
362     SUM_REG_even <= '0';
363     AR_SEL_even <= '0';
364     BR_SEL_even <= '0';
365     WR_SEL_even <= '0';
366     MS_DIFFp_even <= '0';
367     MSD_DIFFm_even <= "00";
368     AS_SUM_SEL_even <= '0';
369     SD_ROUND_SEL_even <= '0';
370     REG_RND_BR_even <= '1';
371     REG_RND_BI_even <= '0';
372     REG_RND_AR_even <= '0';
373     REG_RND_AI_even <= '0';
374     SHIFT_even <= '0';
375     DONE_even <= '0';
376     next_Address_even <= "110";
377
378     --SH4
379     SF_2H_1L_odd <= '0';
380     CC_Validation_odd <= '0';
381     REG_IN_odd <= '0';
382     SUM_REG_odd <= '0';

```

```

383     AR_SEL_odd <= '0';
384     BR_SEL_odd <= '0';
385     WR_SEL_odd <= '0';
386     MS_DIFFp_odd <= '0';
387     MSD_DIFFm_odd <= "00";
388     AS_SUM_SEL_odd <= '0';
389     SD_ROUND_SEL_odd <= '0';
390     REG_RND_BR_odd <= '0';
391     REG_RND_BI_odd <= '0';
392     REG_RND_AR_odd <= '0';
393     REG_RND_AI_odd <= '1';
394     SHIFT_odd <= '0';
395     DONE_odd <= '0';
396     next_Address_odd <= "111";
397
398
399 elsif A = "111" then                                --DONE
400
401     --DONE
402     SF_2H_1L_even <= '0';
403     CC_Validation_even <= '0';
404     REG_IN_even <= '0';
405     SUM_REG_even <= '0';
406     AR_SEL_even <= '0';
407     BR_SEL_even <= '0';
408     WR_SEL_even <= '0';
409     MS_DIFFp_even <= '0';
410     MSD_DIFFm_even <= "00";
411     AS_SUM_SEL_even <= '0';
412     SD_ROUND_SEL_even <= '0';
413     REG_RND_BR_even <= '0';
414     REG_RND_BI_even <= '1';
415     REG_RND_AR_even <= '0';
416     REG_RND_AI_even <= '0';
417     SHIFT_even <= '0';
418     DONE_even <= '1';
419     next_Address_even <= "000";
420
421     --UNUSED
422     SF_2H_1L_odd <= '0';
423     CC_Validation_odd <= '0';
424     REG_IN_odd <= '0';
425     SUM_REG_odd <= '0';
426     AR_SEL_odd <= '0';
427     BR_SEL_odd <= '0';
428     WR_SEL_odd <= '0';
429     MS_DIFFp_odd <= '0';
430     MSD_DIFFm_odd <= "00";
431     AS_SUM_SEL_odd <= '0';
432     SD_ROUND_SEL_odd <= '0';
433     REG_RND_BR_odd <= '0';
434     REG_RND_BI_odd <= '0';
435     REG_RND_AR_odd <= '0';
436     REG_RND_AI_odd <= '0';
437     SHIFT_odd <= '0';
438     DONE_odd <= '0';
439     next_Address_odd <= "000";
440
441 else
442
443     --DONE
444     SF_2H_1L_even <= '0';
445     CC_Validation_even <= '0';
446     REG_IN_even <= '0';
447     SUM_REG_even <= '0';
448     AR_SEL_even <= '0';

```

```

449      BR_SEL_even <= '0';
450      WR_SEL_even <= '0';
451      MS_DIFFp_even <= '0';
452      MSD_DIFFm_even <= "00";
453      AS_SUM_SEL_even <= '0';
454      SD_ROUND_SEL_even <= '0';
455      REG_RND_BR_even <= '0';
456      REG_RND_BI_even <= '0';
457      REG_RND_AR_even <= '0';
458      REG_RND_AI_even <= '0';
459      SHIFT_even <= '0';
460      DONE_even <= '0';
461      next_Address_even <= "000";
462
463      SF_2H_1L_odd <= '0';
464      CC_Validation_odd <= '0';
465      REG_IN_odd <= '0';
466      SUM_REG_odd <= '0';
467      AR_SEL_odd <= '0';
468      BR_SEL_odd <= '0';
469      WR_SEL_odd <= '0';
470      MS_DIFFp_odd <= '0';
471      MSD_DIFFm_odd <= "00";
472      AS_SUM_SEL_odd <= '0';
473      SD_ROUND_SEL_odd <= '0';
474      REG_RND_BR_odd <= '0';
475      REG_RND_BI_odd <= '0';
476      REG_RND_AR_odd <= '0';
477      REG_RND_AI_odd <= '0';
478      SHIFT_odd <= '0';
479      DONE_odd <= '0';
480      next_Address_odd <= "000";
481
482  end if;
483 end process;
484
485
486 end behavioral;

```

Listing 12: Control Unit ROM

7.7.3 PLA

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library IEEE;
6 use IEEE.std_logic_1164.all;
7 use IEEE.numeric_std.all;
8
9
10 entity BFLY CU_LATE_STATUS_PLA is
11     port ( STATUS: in STD_LOGIC_VECTOR(1 downto 0);
12             LSB_in: in STD_LOGIC;
13             CC_Validation_in: in STD_LOGIC;
14             CC_Validation_out: out STD_LOGIC;
15             LSB_out: out STD_LOGIC
16         );
17 end BFLY CU_LATE_STATUS_PLA;
18
19
20 architecture behavioral of BFLY CU_LATE_STATUS_PLA is
21
22     SIGNAL START, SF_2H_1L : STD_LOGIC := '0';

```

```

23
24     begin
25
26         START <= STATUS(0);
27         SF_2H_1L <= STATUS(1);
28
29         LSB_out <= (CC_Validation_in AND (NOT LSB_in)) OR (CC_Validation_in AND
30             SF_2H_1L) OR ((NOT LSB_in) AND START);
31         CC_Validation_out <= NOT(LSB_in);
32
33     end behavioral;

```

Listing 13: Control Unit PLA

7.7.4 Test Bench della Control Unit

```

1  -- Federico Cobianchi - 332753
2  -- Onice Mazzi - 359754
3  -- Antonio Telmon - 353781
4
5  library IEEE;
6  use IEEE.std_logic_1164.all; -- libreria IEEE con definizione tipi standard logic
7  use IEEE.numeric_std.all;
8
9
10 entity tb_CU is
11 end tb_CU;
12
13 -----
14
15 architecture behavioral of tb_CU is
16
17     component BFLY CU DATAPATH is
18         port ( START:  in STD_LOGIC;
19                 SF_2H_1L: in STD_LOGIC;
20                 CK:      in STD_LOGIC;
21                 INSTRUCTION_OUT:        out STD_LOGIC_VECTOR(16 downto 0)
22             );
23     end component;
24
25     constant period : time := 100 ns;
26
27     SIGNAL TB_CLK, TB_SF_2H_1L, TB_START : STD_LOGIC := '0';
28     SIGNAL TB_INSTRUCTION_OUT: STD_LOGIC_VECTOR(16 downto 0) := (others=>'0');
29
30     SIGNAL REG_IN, SUM_REG, AR_SEL, BR_SEL, WR_SEL, MS_DIFFp, AS_SUM_SEL, SD_ROUND_SEL
31         , REG_RND_BR, REG_RND_BI, REG_RND_AR, REG_RND_AI, SHIFT, DONE : STD_LOGIC := '0';
32     SIGNAL MSD_DIFFm : STD_LOGIC_VECTOR (1 downto 0) := "00";
33     SIGNAL SF_2H_1L_out : STD_LOGIC := '0';
34
35     begin
36
37         --Instruction part
38         SF_2H_1L_out <= TB_INSTRUCTION_OUT(16);
39         REG_IN <= TB_INSTRUCTION_OUT(15);
40         SUM_REG <= TB_INSTRUCTION_OUT(14);
41         AR_SEL <= TB_INSTRUCTION_OUT(13);
42         BR_SEL <= TB_INSTRUCTION_OUT(12);
43         WR_SEL <= TB_INSTRUCTION_OUT(11);
44         MS_DIFFp <= TB_INSTRUCTION_OUT(10);
45         MSD_DIFFm <= TB_INSTRUCTION_OUT(9 downto 8);
46         AS_SUM_SEL <= TB_INSTRUCTION_OUT(7);
47         SD_ROUND_SEL <= TB_INSTRUCTION_OUT(6);
48         REG_RND_BR <= TB_INSTRUCTION_OUT(5);

```

```

48      REG_RND_BI <= TB_INSTRUCTION_OUT(4);
49      REG_RND_AR <= TB_INSTRUCTION_OUT(3);
50      REG_RND_AI <= TB_INSTRUCTION_OUT(2);
51      SHIFT <= TB_INSTRUCTION_OUT(1);
52      DONE <= TB_INSTRUCTION_OUT(0);

53
54
55      TB_CLK <= not TB_CLK after period/2;

56
57      process
58      begin
59          wait for period*3;
60          TB_START <= '1';
61          wait for period*1;
62          TB_START <= '0';
63          wait for period*20;
64      end process;

65
66
67      pm_CU : BFLY CU DATAPATH port map (
68          TB_START,
69          TB_SF_2H_1L,
70          TB_CLK,
71          TB_INSTRUCTION_OUT
72      );
73
74  end behavioral;

```

Listing 14: Test Bench della Control Unit

7.8 FFT

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5 library IEEE;
6 use IEEE.std_logic_1164.all; -- libreria IEEE con definizione tipi standard logic
7 use IEEE.numeric_std.all;
8
9
10 entity BFLY_TOP_ENTITY is
11 port(
12     x0r_in : in STD_LOGIC_VECTOR (23 downto 0);
13     x0i_in : in STD_LOGIC_VECTOR (23 downto 0);
14
15     x1r_in : in STD_LOGIC_VECTOR (23 downto 0);
16     x1i_in : in STD_LOGIC_VECTOR (23 downto 0);
17
18     x2r_in : in STD_LOGIC_VECTOR (23 downto 0);
19     x2i_in : in STD_LOGIC_VECTOR (23 downto 0);
20
21     x3r_in : in STD_LOGIC_VECTOR (23 downto 0);
22     x3i_in : in STD_LOGIC_VECTOR (23 downto 0);
23
24     x4r_in : in STD_LOGIC_VECTOR (23 downto 0);
25     x4i_in : in STD_LOGIC_VECTOR (23 downto 0);
26
27     x5r_in : in STD_LOGIC_VECTOR (23 downto 0);
28     x5i_in : in STD_LOGIC_VECTOR (23 downto 0);
29
30     x6r_in : in STD_LOGIC_VECTOR (23 downto 0);
31     x6i_in : in STD_LOGIC_VECTOR (23 downto 0);
32
33     x7r_in : in STD_LOGIC_VECTOR (23 downto 0);

```

```
34      x7i_in : in STD_LOGIC_VECTOR (23 downto 0);
35
36      x8r_in : in STD_LOGIC_VECTOR (23 downto 0);
37      x8i_in : in STD_LOGIC_VECTOR (23 downto 0);
38
39      x9r_in : in STD_LOGIC_VECTOR (23 downto 0);
40      x9i_in : in STD_LOGIC_VECTOR (23 downto 0);
41
42      x10r_in : in STD_LOGIC_VECTOR (23 downto 0);
43      x10i_in : in STD_LOGIC_VECTOR (23 downto 0);
44
45      x11r_in : in STD_LOGIC_VECTOR (23 downto 0);
46      x11i_in : in STD_LOGIC_VECTOR (23 downto 0);
47
48      x12r_in : in STD_LOGIC_VECTOR (23 downto 0);
49      x12i_in : in STD_LOGIC_VECTOR (23 downto 0);
50
51      x13r_in : in STD_LOGIC_VECTOR (23 downto 0);
52      x13i_in : in STD_LOGIC_VECTOR (23 downto 0);
53
54      x14r_in : in STD_LOGIC_VECTOR (23 downto 0);
55      x14i_in : in STD_LOGIC_VECTOR (23 downto 0);
56
57      x15r_in : in STD_LOGIC_VECTOR (23 downto 0);
58      x15i_in : in STD_LOGIC_VECTOR (23 downto 0);
59
60      Clock, START : in STD_LOGIC;
61
62      x0r_out : out STD_LOGIC_VECTOR (23 downto 0);
63      x0i_out : out STD_LOGIC_VECTOR (23 downto 0);
64
65      x1r_out : out STD_LOGIC_VECTOR (23 downto 0);
66      x1i_out : out STD_LOGIC_VECTOR (23 downto 0);
67
68      x2r_out : out STD_LOGIC_VECTOR (23 downto 0);
69      x2i_out : out STD_LOGIC_VECTOR (23 downto 0);
70
71      x3r_out : out STD_LOGIC_VECTOR (23 downto 0);
72      x3i_out : out STD_LOGIC_VECTOR (23 downto 0);
73
74      x4r_out : out STD_LOGIC_VECTOR (23 downto 0);
75      x4i_out : out STD_LOGIC_VECTOR (23 downto 0);
76
77      x5r_out : out STD_LOGIC_VECTOR (23 downto 0);
78      x5i_out : out STD_LOGIC_VECTOR (23 downto 0);
79
80      x6r_out : out STD_LOGIC_VECTOR (23 downto 0);
81      x6i_out : out STD_LOGIC_VECTOR (23 downto 0);
82
83      x7r_out : out STD_LOGIC_VECTOR (23 downto 0);
84      x7i_out : out STD_LOGIC_VECTOR (23 downto 0);
85
86      x8r_out : out STD_LOGIC_VECTOR (23 downto 0);
87      x8i_out : out STD_LOGIC_VECTOR (23 downto 0);
88
89      x9r_out : out STD_LOGIC_VECTOR (23 downto 0);
90      x9i_out : out STD_LOGIC_VECTOR (23 downto 0);
91
92      x10r_out : out STD_LOGIC_VECTOR (23 downto 0);
93      x10i_out : out STD_LOGIC_VECTOR (23 downto 0);
94
95      x11r_out : out STD_LOGIC_VECTOR (23 downto 0);
96      x11i_out : out STD_LOGIC_VECTOR (23 downto 0);
97
98      x12r_out : out STD_LOGIC_VECTOR (23 downto 0);
99      x12i_out : out STD_LOGIC_VECTOR (23 downto 0);
```

```

100
101     x13r_out : out STD_LOGIC_VECTOR (23 downto 0);
102     x13i_out : out STD_LOGIC_VECTOR (23 downto 0);
103
104     x14r_out : out STD_LOGIC_VECTOR (23 downto 0);
105     x14i_out : out STD_LOGIC_VECTOR (23 downto 0);
106
107     x15r_out : out STD_LOGIC_VECTOR (23 downto 0);
108     x15i_out : out STD_LOGIC_VECTOR (23 downto 0);
109
110     DONE : out STD_LOGIC
111 );
112 end BFLY_TOP_ENTITY;
113
114 -----
115
116 architecture structural of BFLY_TOP_ENTITY is
117
118     component bfly_datapath is
119         port(
120             Br_in, Bi_in, Ar_in, Ai_in, Wr_in, Wi_in : in STD_LOGIC_VECTOR (23 downto
121                         0);
122             Clock, START, SF_2H_1L : in STD_LOGIC;
123             Br_out, Bi_out, Ar_out, Ai_out : out STD_LOGIC_VECTOR (23 downto 0);
124             DONE : out STD_LOGIC
125         );
126     end component;
127
128     type sampleArray is array (15 downto 0) of STD_LOGIC_VECTOR (23 downto 0);
129     signal W_r, W_i : sampleArray := (others => (others=>'0'));
130
131     type outputArray is array (31 downto 0) of STD_LOGIC_VECTOR (23 downto 0);
132     signal Ar_out1, Ai_out1, Br_out1, Bi_out1 : outputArray := (others => (others
133                         =>'0'));
134
135     signal DONE_out : STD_LOGIC_VECTOR (31 downto 0) := (others=>'0');
136
137
138 begin
139
140     Ar_out1(0) <= x0r_in;
141     Ai_out1(0) <= x0i_in;
142     Ar_out1(1) <= x1r_in;
143     Ai_out1(1) <= x1i_in;
144     Ar_out1(2) <= x2r_in;
145     Ai_out1(2) <= x2i_in;
146     Ar_out1(3) <= x3r_in;
147     Ai_out1(3) <= x3i_in;
148     Ar_out1(4) <= x4r_in;
149     Ai_out1(4) <= x4i_in;
150     Ar_out1(5) <= x5r_in;
151     Ai_out1(5) <= x5i_in;
152     Ar_out1(6) <= x6r_in;
153     Ai_out1(6) <= x6i_in;
154     Ar_out1(7) <= x7r_in;
155     Ai_out1(7) <= x7i_in;
156
156     Br_out1(0) <= x8r_in;
157     Bi_out1(0) <= x8i_in;
158     Br_out1(1) <= x9r_in;
159     Bi_out1(1) <= x9i_in;
160     Br_out1(2) <= x10r_in;
161     Bi_out1(2) <= x10i_in;
162     Br_out1(3) <= x11r_in;
163     Bi_out1(3) <= x11i_in;
164     Br_out1(4) <= x12r_in;

```

```

164     Bi_out1(4) <= x12i_in;
165     Br_out1(5) <= x13r_in;
166     Bi_out1(5) <= x13i_in;
167     Br_out1(6) <= x14r_in;
168     Bi_out1(6) <= x14i_in;
169     Br_out1(7) <= x15r_in;
170     Bi_out1(7) <= x15i_in;
171
172
173     gen_bfly1a : for j in 0 to 3 generate
174       pm_bfly1a_j : bfly_datapath port map (
175         Br_in => Br_out1(j),
176         Bi_in => Bi_out1(j),
177         Ar_in => Ar_out1(j),
178         Ai_in => Ai_out1(j),
179         Wr_in => W_r(0),
180         Wi_in => W_i(0),
181         Clock => Clock,
182         START => START,
183         SF_2H_1L => '1',
184         Br_out => Ar_out1(12+j),
185         Bi_out => Ai_out1(12+j),
186         Ar_out => Ar_out1(8+j),
187         Ai_out => Ai_out1(8+j),
188         DONE => DONE_out(j)
189       );
190     end generate;
191
192     gen_bfly1b : for j in 0 to 3 generate
193       pm_bfly1b_j : bfly_datapath port map (
194         Br_in => Br_out1(4+j),
195         Bi_in => Bi_out1(4+j),
196         Ar_in => Ar_out1(4+j),
197         Ai_in => Ai_out1(4+j),
198         Wr_in => W_r(0),
199         Wi_in => W_i(0),
200         Clock => Clock,
201         START => START,
202         SF_2H_1L => '1',
203         Br_out => Br_out1(12+j),
204         Bi_out => Bi_out1(12+j),
205         Ar_out => Br_out1(8+j),
206         Ai_out => Bi_out1(8+j),
207         DONE => DONE_out(4+j)
208       );
209     end generate;
210
211 -----
212
213     gen_bfly2a : for j in 0 to 1 generate
214       pm_bfly2a_j : bfly_datapath port map (
215         Br_in => Br_out1(8+j),
216         Bi_in => Bi_out1(8+j),
217         Ar_in => Ar_out1(8+j),
218         Ai_in => Ai_out1(8+j),
219         Wr_in => W_r(0),
220         Wi_in => W_i(0),
221         Clock => Clock,
222         START => DONE_out(0+j),
223         SF_2H_1L => '0',
224         Br_out => Ar_out1(18+j),
225         Bi_out => Ai_out1(18+j),
226         Ar_out => Ar_out1(16+j),
227         Ai_out => Ai_out1(16+j),
228         DONE => DONE_out(8+j)
229       );

```

```

230      end generate;
231
232      gen_bfly2b : for j in 0 to 1 generate
233          pm_bfly2b_j : bfly_datapath port map (
234              Br_in => Br_out1(10+j),
235              Bi_in => Bi_out1(10+j),
236              Ar_in => Ar_out1(10+j),
237              Ai_in => Ai_out1(10+j),
238              Wr_in => W_r(0),
239              Wi_in => W_i(0),
240              Clock => Clock,
241              START => DONE_out(2+j),
242              SF_2H_1L => '0',
243              Br_out => Br_out1(18+j),
244              Bi_out => Bi_out1(18+j),
245              Ar_out => Br_out1(16+j),
246              Ai_out => Bi_out1(16+j),
247              DONE => DONE_out(10+j)
248      );
249      end generate;
250
251      gen_bfly2c : for j in 0 to 1 generate
252          pm_bfly2c_j : bfly_datapath port map (
253              Br_in => Br_out1(12+j),
254              Bi_in => Bi_out1(12+j),
255              Ar_in => Ar_out1(12+j),
256              Ai_in => Ai_out1(12+j),
257              Wr_in => W_r(4),
258              Wi_in => W_i(4),
259              Clock => Clock,
260              START => DONE_out(4+j),
261              SF_2H_1L => '0',
262              Br_out => Ar_out1(22+j),
263              Bi_out => Ai_out1(22+j),
264              Ar_out => Ar_out1(20+j),
265              Ai_out => Ai_out1(20+j),
266              DONE => DONE_out(12+j)
267      );
268      end generate;
269
270      gen_bfly2d : for j in 0 to 1 generate
271          pm_bfly2d_j : bfly_datapath port map (
272              Br_in => Br_out1(14+j),
273              Bi_in => Bi_out1(14+j),
274              Ar_in => Ar_out1(14+j),
275              Ai_in => Ai_out1(14+j),
276              Wr_in => W_r(4),
277              Wi_in => W_i(4),
278              Clock => Clock,
279              START => DONE_out(6+j),
280              SF_2H_1L => '0',
281              Br_out => Br_out1(22+j),
282              Bi_out => Bi_out1(22+j),
283              Ar_out => Br_out1(20+j),
284              Ai_out => Bi_out1(20+j),
285              DONE => DONE_out(14+j)
286      );
287      end generate;
288
289 -----
290
291      pm_bfly3a_1 : bfly_datapath port map (
292          Br_in => Br_out1(16),
293          Bi_in => Bi_out1(16),
294          Ar_in => Ar_out1(16),
295          Ai_in => Ai_out1(16),

```

```
296     Wr_in => W_r(0),
297     Wi_in => W_i(0),
298     Clock => Clock,
299     START => DONE_out(8),
300     SF_2H_1L => '0',
301     Br_out => Ar_out1(25),
302     Bi_out => Ai_out1(25),
303     Ar_out => Ar_out1(24),
304     Ai_out => Ai_out1(24),
305     DONE => DONE_out(16)
306   );
307
308   pm_bfly3b_1 : bfly_datapath port map (
309     Br_in => Br_out1(17),
310     Bi_in => Bi_out1(17),
311     Ar_in => Ar_out1(17),
312     Ai_in => Ai_out1(17),
313     Wr_in => W_r(0),
314     Wi_in => W_i(0),
315     Clock => Clock,
316     START => DONE_out(9),
317     SF_2H_1L => '0',
318     Br_out => Br_out1(25),
319     Bi_out => Bi_out1(25),
320     Ar_out => Br_out1(24),
321     Ai_out => Bi_out1(24),
322     DONE => DONE_out(17)
323   );
324
325   pm_bfly3a_2 : bfly_datapath port map (
326     Br_in => Br_out1(18),
327     Bi_in => Bi_out1(18),
328     Ar_in => Ar_out1(18),
329     Ai_in => Ai_out1(18),
330     Wr_in => W_r(4),
331     Wi_in => W_i(4),
332     Clock => Clock,
333     START => DONE_out(10),
334     SF_2H_1L => '0',
335     Br_out => Ar_out1(27),
336     Bi_out => Ai_out1(27),
337     Ar_out => Ar_out1(26),
338     Ai_out => Ai_out1(26),
339     DONE => DONE_out(18)
340   );
341
342   pm_bfly3b_2 : bfly_datapath port map (
343     Br_in => Br_out1(19),
344     Bi_in => Bi_out1(19),
345     Ar_in => Ar_out1(19),
346     Ai_in => Ai_out1(19),
347     Wr_in => W_r(4),
348     Wi_in => W_i(4),
349     Clock => Clock,
350     START => DONE_out(11),
351     SF_2H_1L => '0',
352     Br_out => Br_out1(27),
353     Bi_out => Bi_out1(27),
354     Ar_out => Br_out1(26),
355     Ai_out => Bi_out1(26),
356     DONE => DONE_out(19)
357   );
358
359   pm_bfly3a_3 : bfly_datapath port map (
360     Br_in => Br_out1(20),
361     Bi_in => Bi_out1(20),
```

```
362     Ar_in => Ar_out1(20),
363     Ai_in => Ai_out1(20),
364     Wr_in => W_r(2),
365     Wi_in => W_i(2),
366     Clock => Clock,
367     START => DONE_out(12),
368     SF_2H_1L => '0',
369     Br_out => Ar_out1(29),
370     Bi_out => Ai_out1(29),
371     Ar_out => Ar_out1(28),
372     Ai_out => Ai_out1(28),
373     DONE => DONE_out(20)
374 );
375
376 pm_bfly3b_3 : bfly_datapath port map (
377     Br_in => Br_out1(21),
378     Bi_in => Bi_out1(21),
379     Ar_in => Ar_out1(21),
380     Ai_in => Ai_out1(21),
381     Wr_in => W_r(2),
382     Wi_in => W_i(2),
383     Clock => Clock,
384     START => DONE_out(13),
385     SF_2H_1L => '0',
386     Br_out => Br_out1(29),
387     Bi_out => Bi_out1(29),
388     Ar_out => Br_out1(28),
389     Ai_out => Bi_out1(28),
390     DONE => DONE_out(21)
391 );
392
393 pm_bfly3a_4 : bfly_datapath port map (
394     Br_in => Br_out1(22),
395     Bi_in => Bi_out1(22),
396     Ar_in => Ar_out1(22),
397     Ai_in => Ai_out1(22),
398     Wr_in => W_r(6),
399     Wi_in => W_i(6),
400     Clock => Clock,
401     START => DONE_out(14),
402     SF_2H_1L => '0',
403     Br_out => Ar_out1(31),
404     Bi_out => Ai_out1(31),
405     Ar_out => Ar_out1(30),
406     Ai_out => Ai_out1(30),
407     DONE => DONE_out(22)
408 );
409
410 pm_bfly3b_4 : bfly_datapath port map (
411     Br_in => Br_out1(23),
412     Bi_in => Bi_out1(23),
413     Ar_in => Ar_out1(23),
414     Ai_in => Ai_out1(23),
415     Wr_in => W_r(6),
416     Wi_in => W_i(6),
417     Clock => Clock,
418     START => DONE_out(15),
419     SF_2H_1L => '0',
420     Br_out => Br_out1(31),
421     Bi_out => Bi_out1(31),
422     Ar_out => Br_out1(30),
423     Ai_out => Bi_out1(30),
424     DONE => DONE_out(23)
425 );
426
427 -----
```

```
428
429     pm_bfly4_1 : bfly_datapath port map (
430         Br_in => Br_out1(24),
431         Bi_in => Bi_out1(24),
432         Ar_in => Ar_out1(24),
433         Ai_in => Ai_out1(24),
434         Wr_in => W_r(0),
435         Wi_in => W_i(0),
436         Clock => Clock,
437         START => DONE_out(16),
438         SF_2H_1L => '0',
439         Br_out => x8r_out,
440         Bi_out => x8i_out,
441         Ar_out => x0r_out,
442         Ai_out => x0i_out,
443         DONE => DONE_out(24)
444     );
445
446     pm_bfly4_2 : bfly_datapath port map (
447         Br_in => Br_out1(25),
448         Bi_in => Bi_out1(25),
449         Ar_in => Ar_out1(25),
450         Ai_in => Ai_out1(25),
451         Wr_in => W_r(4),
452         Wi_in => W_i(4),
453         Clock => Clock,
454         START => DONE_out(17),
455         SF_2H_1L => '0',
456         Br_out => x12r_out,
457         Bi_out => x12i_out,
458         Ar_out => x4r_out,
459         Ai_out => x4i_out,
460         DONE => DONE_out(25)
461     );
462
463     pm_bfly4_3 : bfly_datapath port map (
464         Br_in => Br_out1(26),
465         Bi_in => Bi_out1(26),
466         Ar_in => Ar_out1(26),
467         Ai_in => Ai_out1(26),
468         Wr_in => W_r(2),
469         Wi_in => W_i(2),
470         Clock => Clock,
471         START => DONE_out(18),
472         SF_2H_1L => '0',
473         Br_out => x10r_out,
474         Bi_out => x10i_out,
475         Ar_out => x2r_out,
476         Ai_out => x2i_out,
477         DONE => DONE_out(26)
478     );
479
480     pm_bfly4_4 : bfly_datapath port map (
481         Br_in => Br_out1(27),
482         Bi_in => Bi_out1(27),
483         Ar_in => Ar_out1(27),
484         Ai_in => Ai_out1(27),
485         Wr_in => W_r(6),
486         Wi_in => W_i(6),
487         Clock => Clock,
488         START => DONE_out(19),
489         SF_2H_1L => '0',
490         Br_out => x14r_out,
491         Bi_out => x14i_out,
492         Ar_out => x6r_out,
493         Ai_out => x6i_out,
```

```
494         DONE => DONE_out(27)
495     );
496
497     pm_bfly4_5 : bfly_datapath port map (
498         Br_in => Br_out1(28),
499         Bi_in => Bi_out1(28),
500         Ar_in => Ar_out1(28),
501         Ai_in => Ai_out1(28),
502         Wr_in => W_r(1),
503         Wi_in => W_i(1),
504         Clock => Clock,
505         START => DONE_out(20),
506         SF_2H_1L => '0',
507         Br_out => x9r_out,
508         Bi_out => x9i_out,
509         Ar_out => x1r_out,
510         Ai_out => x1i_out,
511         DONE => DONE_out(28)
512     );
513
514     pm_bfly4_6 : bfly_datapath port map (
515         Br_in => Br_out1(29),
516         Bi_in => Bi_out1(29),
517         Ar_in => Ar_out1(29),
518         Ai_in => Ai_out1(29),
519         Wr_in => W_r(5),
520         Wi_in => W_i(5),
521         Clock => Clock,
522         START => DONE_out(21),
523         SF_2H_1L => '0',
524         Br_out => x13r_out,
525         Bi_out => x13i_out,
526         Ar_out => x5r_out,
527         Ai_out => x5i_out,
528         DONE => DONE_out(29)
529     );
530
531     pm_bfly4_7 : bfly_datapath port map (
532         Br_in => Br_out1(30),
533         Bi_in => Bi_out1(30),
534         Ar_in => Ar_out1(30),
535         Ai_in => Ai_out1(30),
536         Wr_in => W_r(3),
537         Wi_in => W_i(3),
538         Clock => Clock,
539         START => DONE_out(22),
540         SF_2H_1L => '0',
541         Br_out => x11r_out,
542         Bi_out => x11i_out,
543         Ar_out => x3r_out,
544         Ai_out => x3i_out,
545         DONE => DONE_out(30)
546     );
547
548     pm_bfly4_8 : bfly_datapath port map (
549         Br_in => Br_out1(31),
550         Bi_in => Bi_out1(31),
551         Ar_in => Ar_out1(31),
552         Ai_in => Ai_out1(31),
553         Wr_in => W_r(7),
554         Wi_in => W_i(7),
555         Clock => Clock,
556         START => DONE_out(23),
557         SF_2H_1L => '0',
558         Br_out => x15r_out,
559         Bi_out => x15i_out,
```

```

560      Ar_out => x7r_out ,
561      Ai_out => x7i_out ,
562      DONE => DONE_out(31)
563  );
564
565
566 -----
567
568  DONE <= DONE_out(24) or DONE_out(25) or DONE_out(26) or DONE_out(27) or DONE_out(28) or
569    DONE_out(29) or DONE_out(30) or DONE_out(31);
570
571 -----
572
573
574
575      W_r(0) <= "01111111111111111111111111111111";           -- 1.000000000000000 +
576      0.0000000000000000i
577      W_i(0) <= "00000000000000000000000000000000";
578
579      W_r(1) <= "011101100100001101011111";           -- 0.923879532511287 -
580      0.382683432365090i
581      W_i(1) <= "110011110000010000111011";
582
583      W_r(2) <= "010110101000001001111001";           -- 0.707106781186547 -
584      0.707106781186548i
585      W_i(2) <= "10100101011110110000111";
586
587      W_r(3) <= "00110000111101111000101";           -- 0.382683432365090 -
588      0.923879532511287i
589      W_i(3) <= "10001001101111001010001";
590
591      W_r(4) <= "00000000000000000000000000000000";           -- -0.000000000000000 -
592      1.000000000000000i
593      W_i(4) <= "10000000000000000000000000000001";
594
595      W_r(5) <= "110011110000010000111011";           -- -0.382683432365090 -
596      0.923879532511287i
597      W_i(5) <= "10001001101111001010001";
598
599      W_r(6) <= "10100101011110110000111";           -- -0.707106781186548 -
600      0.707106781186547i
601      W_i(6) <= "10100101011110110000111";
602
603      W_r(7) <= "10001001101111001010001";           -- -0.923879532511287 -
604      0.382683432365089i
605      W_i(7) <= "110011110000010000111011";
606
607
608 end structural;

```

Listing 15: FFT

7.8.1 Test Bench della Butterfly singola

```

1  -- Federico Cobianchi - 332753
2  -- Onice Mazzi - 359754
3  -- Antonio Telmon - 353781
4
5  library IEEE;
6  use IEEE.std_logic_1164.all;
7  use IEEE.numeric_std.all;
8  use IEEE.fixed_float_types.all;
9  use IEEE.fixed_pkg.all;
10

```

```

entity tb_fft is
end tb_fft;
13
14 architecture behavioral of tb_fft is
15
16 component BFLY_TOP_ENTITY is
17     port(
18         -- segnali reali ingresso
19         x0r_in, x1r_in, x2r_in, x3r_in, x4r_in, x5r_in, x6r_in, x7r_in: in
20             STD_LOGIC_VECTOR (23 downto 0);
21         x8r_in, x9r_in, x10r_in, x11r_in, x12r_in, x13r_in, x14r_in, x15r_in: in
22             STD_LOGIC_VECTOR (23 downto 0);
23         -- segnali immaginari ingresso
24         x0i_in, x1i_in, x2i_in, x3i_in, x4i_in, x5i_in, x6i_in, x7i_in: in
25             STD_LOGIC_VECTOR (23 downto 0);
26         x8i_in, x9i_in, x10i_in, x11i_in, x12i_in, x13i_in, x14i_in, x15i_in: in
27             STD_LOGIC_VECTOR (23 downto 0);
28         -- segnali reali uscita
29         x0r_out, x1r_out, x2r_out, x3r_out, x4r_out, x5r_out, x6r_out, x7r_out:
30             out STD_LOGIC_VECTOR (23 downto 0);
31         x8r_out, x9r_out, x10r_out, x11r_out, x12r_out, x13r_out, x14r_out,
32             x15r_out: out STD_LOGIC_VECTOR (23 downto 0);
33         -- segnali immaginari uscita
34         x0i_out, x1i_out, x2i_out, x3i_out, x4i_out, x5i_out, x6i_out, x7i_out:
35             out STD_LOGIC_VECTOR (23 downto 0);
36         x8i_out, x9i_out, x10i_out, x11i_out, x12i_out, x13i_out, x14i_out,
37             x15i_out: out STD_LOGIC_VECTOR (23 downto 0);
38         -- segnali di controllo
39         Clock: in STD_LOGIC;
40         START : in STD_LOGIC;
41         DONE : out STD_LOGIC
42     );
43 end component;
44
45 constant period : time := 10 ns; -- clock da 100MHz
46
47 signal TB_CLK: STD_LOGIC := '0';
48 signal TB_START: STD_LOGIC := '0';
49 signal TB_DONE: STD_LOGIC := '0';
50 type array_dati is array (0 to 15) of STD_LOGIC_VECTOR(23 downto 0);
51 signal r_in : array_dati := (others => (others => '0')) ; -- other piu' annidato mette
52     tutto a 0, l'altro lo fa per tutte le posizioni
53 signal i_in : array_dati := (others => (others => '0'));
54 signal r_out : array_dati;
55 signal i_out : array_dati;
56
57 type sfixed_array is array (0 to 15) of sfixed (0 downto -23);
58
59 signal sfixed_xr_in, sfixed_xi_in : sfixed_array;
60 signal sfixed_xr_out, sfixed_xi_out : sfixed_array;
61
62
63 begin
64
65     for_sfixed : for j in 0 to 15 generate
66         sfixed_xr_in(j) <= to_sfixed((r_in(j)),0,-23);
67         sfixed_xi_in(j) <= to_sfixed((i_in(j)),0,-23);
68
69         sfixed_xr_out(j) <= to_sfixed((r_out(j)),0,-23);
70         sfixed_xi_out(j) <= to_sfixed((i_out(j)),0,-23);
71
72     end generate;
73
74     TB_CLK <= not TB_CLK after period/2;
75
76     DUT : BFLY_TOP_ENTITY port map (
77

```

```
68      -- assegnazione dei segnali di controllo
69          Clock => TB_CLK,
70          START => TB_START,
71          DONE => TB_DONE,
72      -- assegnazione dei segnali ingresso reali
73          x0r_in => r_in(0),
74          x1r_in => r_in(1),
75          x2r_in => r_in(2),
76          x3r_in => r_in(3),
77          x4r_in => r_in(4),
78          x5r_in => r_in(5),
79          x6r_in => r_in(6),
80          x7r_in => r_in(7),
81          x8r_in => r_in(8),
82          x9r_in => r_in(9),
83          x10r_in => r_in(10),
84          x11r_in => r_in(11),
85          x12r_in => r_in(12),
86          x13r_in => r_in(13),
87          x14r_in => r_in(14),
88          x15r_in => r_in(15),
89      -- assegnazione dei segnali ingresso immaginari
90          x0i_in => i_in(0),
91          x1i_in => i_in(1),
92          x2i_in => i_in(2),
93          x3i_in => i_in(3),
94          x4i_in => i_in(4),
95          x5i_in => i_in(5),
96          x6i_in => i_in(6),
97          x7i_in => i_in(7),
98          x8i_in => i_in(8),
99          x9i_in => i_in(9),
100         x10i_in => i_in(10),
101         x11i_in => i_in(11),
102         x12i_in => i_in(12),
103         x13i_in => i_in(13),
104         x14i_in => i_in(14),
105         x15i_in => i_in(15),
106     -- assegnazione dei segnali d'uscita reali
107         x0r_out => r_out(0),
108         x1r_out => r_out(1),
109         x2r_out => r_out(2),
110         x3r_out => r_out(3),
111         x4r_out => r_out(4),
112         x5r_out => r_out(5),
113         x6r_out => r_out(6),
114         x7r_out => r_out(7),
115         x8r_out => r_out(8),
116         x9r_out => r_out(9),
117         x10r_out => r_out(10),
118         x11r_out => r_out(11),
119         x12r_out => r_out(12),
120         x13r_out => r_out(13),
121         x14r_out => r_out(14),
122         x15r_out => r_out(15),
123     -- assegnazione dei segnali d'uscita immaginari
124         x0i_out => i_out(0),
125         x1i_out => i_out(1),
126         x2i_out => i_out(2),
127         x3i_out => i_out(3),
128         x4i_out => i_out(4),
129         x5i_out => i_out(5),
130         x6i_out => i_out(6),
131         x7i_out => i_out(7),
132         x8i_out => i_out(8),
133         x9i_out => i_out(9),
```

```
134      x10i_out => i_out(10),
135      x11i_out => i_out(11),
136      x12i_out => i_out(12),
137      x13i_out => i_out(13),
138      x14i_out => i_out(14),
139      x15i_out => i_out(15)
140  );
141
142
143
144  test: process
145 begin
146    TB_START <= '0';
147    wait for 2*period; -- Aspetto 2 period prima di iniziare la simulazione
148    TB_START <= '1';
149
150    ----- TEST 1 -----
151    -1 -1 -1 -1 -1 -1 -1
152    r_in(0) <= "10000000000000000000000000000000";
153    r_in(1) <= "10000000000000000000000000000000";
154    r_in(2) <= "10000000000000000000000000000000";
155    r_in(3) <= "10000000000000000000000000000000";
156    r_in(4) <= "10000000000000000000000000000000";
157    r_in(5) <= "10000000000000000000000000000000";
158    r_in(6) <= "10000000000000000000000000000000";
159    r_in(7) <= "10000000000000000000000000000000";
160    r_in(8) <= "10000000000000000000000000000000";
161    r_in(9) <= "10000000000000000000000000000000";
162    r_in(10) <= "10000000000000000000000000000000";
163    r_in(11) <= "10000000000000000000000000000000";
164    r_in(12) <= "10000000000000000000000000000000";
165    r_in(13) <= "10000000000000000000000000000000";
166    r_in(14) <= "10000000000000000000000000000000";
167    r_in(15) <= "10000000000000000000000000000000";
168
169    i_in(0) <= "00000000000000000000000000000000";
170    i_in(1) <= "00000000000000000000000000000000";
171    i_in(2) <= "00000000000000000000000000000000";
172    i_in(3) <= "00000000000000000000000000000000";
173    i_in(4) <= "00000000000000000000000000000000";
174    i_in(5) <= "00000000000000000000000000000000";
175    i_in(6) <= "00000000000000000000000000000000";
176    i_in(7) <= "00000000000000000000000000000000";
177    i_in(8) <= "00000000000000000000000000000000";
178    i_in(9) <= "00000000000000000000000000000000";
179    i_in(10) <= "00000000000000000000000000000000";
180    i_in(11) <= "00000000000000000000000000000000";
181    i_in(12) <= "00000000000000000000000000000000";
182    i_in(13) <= "00000000000000000000000000000000";
183    i_in(14) <= "00000000000000000000000000000000";
184    i_in(15) <= "00000000000000000000000000000000";
185
186    wait for 2*period;
187    TB_START <= '0';
188
189    wait for 11*period;
190    TB_START <= '1';
191
192    ----- TEST 2 -----
193    0 1 0
194    r_in(0) <= "10000000000000000000000000000000";
195    r_in(1) <= "00000000000000000000000000000000";
196    r_in(2) <= "01111111111111111111111111111111";
197    r_in(3) <= "00000000000000000000000000000000";
```

```

198      r_in(6)  <= "0111111111111111111111111111111111";
199      r_in(7)  <= "00000000000000000000000000000000";
200      r_in(8)  <= "10000000000000000000000000000000";
201      r_in(9)  <= "00000000000000000000000000000000";
202      r_in(10) <= "01111111111111111111111111111111";
203      r_in(11) <= "00000000000000000000000000000000";
204      r_in(12) <= "10000000000000000000000000000000";
205      r_in(13) <= "00000000000000000000000000000000";
206      r_in(14) <= "01111111111111111111111111111111";
207      r_in(15) <= "00000000000000000000000000000000";
208
209      i_in(0)  <= "00000000000000000000000000000000";
210      i_in(1)  <= "00000000000000000000000000000000";
211      i_in(2)  <= "00000000000000000000000000000000";
212      i_in(3)  <= "00000000000000000000000000000000";
213      i_in(4)  <= "00000000000000000000000000000000";
214      i_in(5)  <= "00000000000000000000000000000000";
215      i_in(6)  <= "00000000000000000000000000000000";
216      i_in(7)  <= "00000000000000000000000000000000";
217      i_in(8)  <= "00000000000000000000000000000000";
218      i_in(9)  <= "00000000000000000000000000000000";
219      i_in(10) <= "00000000000000000000000000000000";
220      i_in(11) <= "00000000000000000000000000000000";
221      i_in(12) <= "00000000000000000000000000000000";
222      i_in(13) <= "00000000000000000000000000000000";
223      i_in(14) <= "00000000000000000000000000000000";
224      i_in(15) <= "00000000000000000000000000000000";
225
226      wait for 1*period;
227      TB_START <= '0';
228
229      wait for 12*period;
230      TB_START <= '1';
231
232      ----- TEST 3 -----
233      0 0 0 0 0 0 0 0
234      r_in(0)  <= "01111111111111111111111111111111";
235      r_in(1)  <= "00000000000000000000000000000000";
236      r_in(2)  <= "00000000000000000000000000000000";
237      r_in(3)  <= "00000000000000000000000000000000";
238      r_in(4)  <= "00000000000000000000000000000000";
239      r_in(5)  <= "00000000000000000000000000000000";
240      r_in(6)  <= "00000000000000000000000000000000";
241      r_in(7)  <= "00000000000000000000000000000000";
242      r_in(8)  <= "00000000000000000000000000000000";
243      r_in(9)  <= "00000000000000000000000000000000";
244      r_in(10) <= "00000000000000000000000000000000";
245      r_in(11) <= "00000000000000000000000000000000";
246      r_in(12) <= "00000000000000000000000000000000";
247      r_in(13) <= "00000000000000000000000000000000";
248      r_in(14) <= "00000000000000000000000000000000";
249      r_in(15) <= "00000000000000000000000000000000";
250
251      i_in(0)  <= "00000000000000000000000000000000";
252      i_in(1)  <= "00000000000000000000000000000000";
253      i_in(2)  <= "00000000000000000000000000000000";
254      i_in(3)  <= "00000000000000000000000000000000";
255      i_in(4)  <= "00000000000000000000000000000000";
256      i_in(5)  <= "00000000000000000000000000000000";
257      i_in(6)  <= "00000000000000000000000000000000";
258      i_in(7)  <= "00000000000000000000000000000000";
259      i_in(8)  <= "00000000000000000000000000000000";
260      i_in(9)  <= "00000000000000000000000000000000";
261      i_in(10) <= "00000000000000000000000000000000";
262      i_in(11) <= "00000000000000000000000000000000";
263      i_in(12) <= "00000000000000000000000000000000";

```

```

263      i_in(13) <= "0000000000000000000000000000";
264      i_in(14) <= "0000000000000000000000000000";
265      i_in(15) <= "0000000000000000000000000000";
266
267      wait for 1*period;
268      TB_START <= '0';
269
270      wait for 12*period;
271      TB_START <= '1';
272
273      ----- TEST 4 -----
274      1 -1 -1 1 1
275      r_in(0) <= "1000000000000000000000000000";
276      r_in(1) <= "1000000000000000000000000000";
277      r_in(2) <= "0111111111111111111111111111";
278      r_in(3) <= "0111111111111111111111111111";
279      r_in(4) <= "1000000000000000000000000000";
280      r_in(5) <= "1000000000000000000000000000";
281      r_in(6) <= "0111111111111111111111111111";
282      r_in(7) <= "0111111111111111111111111111";
283      r_in(8) <= "1000000000000000000000000000";
284      r_in(9) <= "1000000000000000000000000000";
285      r_in(10) <= "0111111111111111111111111111";
286      r_in(11) <= "0111111111111111111111111111";
287      r_in(12) <= "1000000000000000000000000000";
288      r_in(13) <= "1000000000000000000000000000";
289      r_in(14) <= "0111111111111111111111111111";
290      r_in(15) <= "0111111111111111111111111111";
291
292      i_in(0) <= "0000000000000000000000000000";
293      i_in(1) <= "0000000000000000000000000000";
294      i_in(2) <= "0000000000000000000000000000";
295      i_in(3) <= "0000000000000000000000000000";
296      i_in(4) <= "0000000000000000000000000000";
297      i_in(5) <= "0000000000000000000000000000";
298      i_in(6) <= "0000000000000000000000000000";
299      i_in(7) <= "0000000000000000000000000000";
300      i_in(8) <= "0000000000000000000000000000";
301      i_in(9) <= "0000000000000000000000000000";
302      i_in(10) <= "0000000000000000000000000000";
303      i_in(11) <= "0000000000000000000000000000";
304      i_in(12) <= "0000000000000000000000000000";
305      i_in(13) <= "0000000000000000000000000000";
306      i_in(14) <= "0000000000000000000000000000";
307      i_in(15) <= "0000000000000000000000000000";
308
309      wait for 1*period;
310      TB_START <= '0';
311
312      wait for 12*period;
313      TB_START <= '1';
314
315      ----- TEST 5 -----
316      0.5 0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5
317      r_in(0) <= "0100000000000000000000000000";
318      r_in(1) <= "0100000000000000000000000000";
319      r_in(2) <= "0100000000000000000000000000";
320      r_in(3) <= "0100000000000000000000000000";
321      r_in(4) <= "0100000000000000000000000000";
322      r_in(5) <= "0100000000000000000000000000";
323      r_in(6) <= "0100000000000000000000000000";
324      r_in(7) <= "0100000000000000000000000000";
325      r_in(8) <= "0100000000000000000000000000";
326      r_in(9) <= "1100000000000000000000000000";
327      r_in(10) <= "1100000000000000000000000000";

```



```
392     TB_START <= '0';
393
394     wait for 12*period;
395     TB_START <= '1';
396
397     ----- TEST 7 ----- 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
398     0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
399     r_in(0) <= "01000000000000000000000000000000";
400     r_in(1) <= "01000000000000000000000000000000";
401     r_in(2) <= "01000000000000000000000000000000";
402     r_in(3) <= "01000000000000000000000000000000";
403     r_in(4) <= "01000000000000000000000000000000";
404     r_in(5) <= "01000000000000000000000000000000";
405     r_in(6) <= "01000000000000000000000000000000";
406     r_in(7) <= "01000000000000000000000000000000";
407     r_in(8) <= "01000000000000000000000000000000";
408     r_in(9) <= "01000000000000000000000000000000";
409     r_in(10) <= "01000000000000000000000000000000";
410     r_in(11) <= "01000000000000000000000000000000";
411     r_in(12) <= "01000000000000000000000000000000";
412     r_in(13) <= "01000000000000000000000000000000";
413     r_in(14) <= "01000000000000000000000000000000";
414     r_in(15) <= "01000000000000000000000000000000";
415
416     i_in(0) <= "00000000000000000000000000000000";
417     i_in(1) <= "00000000000000000000000000000000";
418     i_in(2) <= "00000000000000000000000000000000";
419     i_in(3) <= "00000000000000000000000000000000";
420     i_in(4) <= "00000000000000000000000000000000";
421     i_in(5) <= "00000000000000000000000000000000";
422     i_in(6) <= "00000000000000000000000000000000";
423     i_in(7) <= "00000000000000000000000000000000";
424     i_in(8) <= "00000000000000000000000000000000";
425     i_in(9) <= "00000000000000000000000000000000";
426     i_in(10) <= "00000000000000000000000000000000";
427     i_in(11) <= "00000000000000000000000000000000";
428     i_in(12) <= "00000000000000000000000000000000";
429     i_in(13) <= "00000000000000000000000000000000";
430     i_in(14) <= "00000000000000000000000000000000";
431     i_in(15) <= "00000000000000000000000000000000";
432
433     wait for 1*period;
434     TB_START <= '0';
435
436     wait;
437
438 end process;
439
440
441
442     ----- TEST 1 ----- -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
443     -1 -1 -1 -1 -1 -1
444     -- r_in(0) <= "10000000000000000000000000000000";
445     -- r_in(1) <= "10000000000000000000000000000000";
446     -- r_in(2) <= "10000000000000000000000000000000";
447     -- r_in(3) <= "10000000000000000000000000000000";
448     -- r_in(4) <= "10000000000000000000000000000000";
449     -- r_in(5) <= "10000000000000000000000000000000";
450     -- r_in(6) <= "10000000000000000000000000000000";
451     -- r_in(7) <= "10000000000000000000000000000000";
452     -- r_in(8) <= "10000000000000000000000000000000";
453     -- r_in(9) <= "10000000000000000000000000000000";
454     -- r_in(10) <= "10000000000000000000000000000000";
455     -- r_in(11) <= "10000000000000000000000000000000";
456     -- r_in(12) <= "10000000000000000000000000000000";
```

```

456      -- r_in(13) <= "10000000000000000000000000000000";
457      -- r_in(14) <= "10000000000000000000000000000000";
458      -- r_in(15) <= "10000000000000000000000000000000";
459
460      -- i_in(0) <= "0000000000000000000000000000000";
461      -- i_in(1) <= "0000000000000000000000000000000";
462      -- i_in(2) <= "0000000000000000000000000000000";
463      -- i_in(3) <= "0000000000000000000000000000000";
464      -- i_in(4) <= "0000000000000000000000000000000";
465      -- i_in(5) <= "0000000000000000000000000000000";
466      -- i_in(6) <= "0000000000000000000000000000000";
467      -- i_in(7) <= "0000000000000000000000000000000";
468      -- i_in(8) <= "0000000000000000000000000000000";
469      -- i_in(9) <= "0000000000000000000000000000000";
470      -- i_in(10) <= "0000000000000000000000000000000";
471      -- i_in(11) <= "0000000000000000000000000000000";
472      -- i_in(12) <= "0000000000000000000000000000000";
473      -- i_in(13) <= "0000000000000000000000000000000";
474      -- i_in(14) <= "0000000000000000000000000000000";
475      -- i_in(15) <= "0000000000000000000000000000000";
476
477
478      ----- TEST 2 ----- -1 0 1 0 -1 0 1 0 -1 0 1 0 -1
479      0 1 0
480      -- r_in(0) <= "10000000000000000000000000000000";
481      -- r_in(1) <= "0000000000000000000000000000000";
482      -- r_in(2) <= "0111111111111111111111111111111";
483      -- r_in(3) <= "0000000000000000000000000000000";
484      -- r_in(4) <= "1000000000000000000000000000000";
485      -- r_in(5) <= "0000000000000000000000000000000";
486      -- r_in(6) <= "0111111111111111111111111111111";
487      -- r_in(7) <= "0000000000000000000000000000000";
488      -- r_in(8) <= "1000000000000000000000000000000";
489      -- r_in(9) <= "0000000000000000000000000000000";
490      -- r_in(10) <= "0111111111111111111111111111111";
491      -- r_in(11) <= "0000000000000000000000000000000";
492      -- r_in(12) <= "1000000000000000000000000000000";
493      -- r_in(13) <= "0000000000000000000000000000000";
494      -- r_in(14) <= "0111111111111111111111111111111";
495      -- r_in(15) <= "0000000000000000000000000000000";
496
497      -- i_in(0) <= "0000000000000000000000000000000";
498      -- i_in(1) <= "0000000000000000000000000000000";
499      -- i_in(2) <= "0000000000000000000000000000000";
500      -- i_in(3) <= "0000000000000000000000000000000";
501      -- i_in(4) <= "0000000000000000000000000000000";
502      -- i_in(5) <= "0000000000000000000000000000000";
503      -- i_in(6) <= "0000000000000000000000000000000";
504      -- i_in(7) <= "0000000000000000000000000000000";
505      -- i_in(8) <= "0000000000000000000000000000000";
506      -- i_in(9) <= "0000000000000000000000000000000";
507      -- i_in(10) <= "0000000000000000000000000000000";
508      -- i_in(11) <= "0000000000000000000000000000000";
509      -- i_in(12) <= "0000000000000000000000000000000";
510      -- i_in(13) <= "0000000000000000000000000000000";
511      -- i_in(14) <= "0000000000000000000000000000000";
512      -- i_in(15) <= "0000000000000000000000000000000";
513
514      ----- TEST 3 ----- 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
515      0 0 0
516      -- r_in(0) <= "0111111111111111111111111111111";
517      -- r_in(1) <= "0000000000000000000000000000000";
518      -- r_in(2) <= "0000000000000000000000000000000";
519      -- r_in(3) <= "0000000000000000000000000000000";
520      -- r_in(4) <= "0000000000000000000000000000000";

```



```
585
586      ----- TEST 5 ----- 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
587      0.5 0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5
588      -- r_in(0) <= "01000000000000000000000000000000";
589      -- r_in(1) <= "01000000000000000000000000000000";
590      -- r_in(2) <= "01000000000000000000000000000000";
591      -- r_in(3) <= "01000000000000000000000000000000";
592      -- r_in(4) <= "01000000000000000000000000000000";
593      -- r_in(5) <= "01000000000000000000000000000000";
594      -- r_in(6) <= "01000000000000000000000000000000";
595      -- r_in(7) <= "01000000000000000000000000000000";
596      -- r_in(8) <= "01000000000000000000000000000000";
597      -- r_in(9) <= "11000000000000000000000000000000";
598      -- r_in(10) <= "11000000000000000000000000000000";
599      -- r_in(11) <= "11000000000000000000000000000000";
600      -- r_in(12) <= "11000000000000000000000000000000";
601      -- r_in(13) <= "11000000000000000000000000000000";
602      -- r_in(14) <= "11000000000000000000000000000000";
603      -- r_in(15) <= "11000000000000000000000000000000";
604
605      -- i_in(0) <= "00000000000000000000000000000000";
606      -- i_in(1) <= "00000000000000000000000000000000";
607      -- i_in(2) <= "00000000000000000000000000000000";
608      -- i_in(3) <= "00000000000000000000000000000000";
609      -- i_in(4) <= "00000000000000000000000000000000";
610      -- i_in(5) <= "00000000000000000000000000000000";
611      -- i_in(6) <= "00000000000000000000000000000000";
612      -- i_in(7) <= "00000000000000000000000000000000";
613      -- i_in(8) <= "00000000000000000000000000000000";
614      -- i_in(9) <= "00000000000000000000000000000000";
615      -- i_in(10) <= "00000000000000000000000000000000";
616      -- i_in(11) <= "00000000000000000000000000000000";
617      -- i_in(12) <= "00000000000000000000000000000000";
618      -- i_in(13) <= "00000000000000000000000000000000";
619      -- i_in(14) <= "00000000000000000000000000000000";
620      -- i_in(15) <= "00000000000000000000000000000000";
621
622      ----- TEST 6 ----- 0 0 0 0 0 0 0 0 0.75 0 0 0 0
623      0 0 0
624      -- r_in(0) <= "00000000000000000000000000000000";
625      -- r_in(1) <= "00000000000000000000000000000000";
626      -- r_in(2) <= "00000000000000000000000000000000";
627      -- r_in(3) <= "00000000000000000000000000000000";
628      -- r_in(4) <= "00000000000000000000000000000000";
629      -- r_in(5) <= "00000000000000000000000000000000";
630      -- r_in(6) <= "00000000000000000000000000000000";
631      -- r_in(7) <= "00000000000000000000000000000000";
632      -- r_in(8) <= "01100000000000000000000000000000";
633      -- r_in(9) <= "00000000000000000000000000000000";
634      -- r_in(10) <= "00000000000000000000000000000000";
635      -- r_in(11) <= "00000000000000000000000000000000";
636      -- r_in(12) <= "00000000000000000000000000000000";
637      -- r_in(13) <= "00000000000000000000000000000000";
638      -- r_in(14) <= "00000000000000000000000000000000";
639      -- r_in(15) <= "00000000000000000000000000000000";
640
641      -- i_in(0) <= "00000000000000000000000000000000";
642      -- i_in(1) <= "00000000000000000000000000000000";
643      -- i_in(2) <= "00000000000000000000000000000000";
644      -- i_in(3) <= "00000000000000000000000000000000";
645      -- i_in(4) <= "00000000000000000000000000000000";
646      -- i_in(5) <= "00000000000000000000000000000000";
647      -- i_in(6) <= "00000000000000000000000000000000";
648      -- i_in(7) <= "00000000000000000000000000000000";
649      -- i_in(8) <= "00000000000000000000000000000000";
```

```
649      -- i_in(9)  <= "00000000000000000000000000000000";
650      -- i_in(10) <= "00000000000000000000000000000000";
651      -- i_in(11) <= "00000000000000000000000000000000";
652      -- i_in(12) <= "00000000000000000000000000000000";
653      -- i_in(13) <= "00000000000000000000000000000000";
654      -- i_in(14) <= "00000000000000000000000000000000";
655      -- i_in(15) <= "00000000000000000000000000000000";
656
657
658      ----- TEST 7 ----- 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
659      0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
660      -- r_in(0)  <= "01000000000000000000000000000000";
661      -- r_in(1)  <= "01000000000000000000000000000000";
662      -- r_in(2)  <= "01000000000000000000000000000000";
663      -- r_in(3)  <= "01000000000000000000000000000000";
664      -- r_in(4)  <= "01000000000000000000000000000000";
665      -- r_in(5)  <= "01000000000000000000000000000000";
666      -- r_in(6)  <= "01000000000000000000000000000000";
667      -- r_in(7)  <= "01000000000000000000000000000000";
668      -- r_in(8)  <= "01000000000000000000000000000000";
669      -- r_in(9)  <= "01000000000000000000000000000000";
670      -- r_in(10) <= "01000000000000000000000000000000";
671      -- r_in(11) <= "01000000000000000000000000000000";
672      -- r_in(12) <= "01000000000000000000000000000000";
673      -- r_in(13) <= "01000000000000000000000000000000";
674      -- r_in(14) <= "01000000000000000000000000000000";
675      -- r_in(15) <= "01000000000000000000000000000000";
676
677      -- i_in(0)  <= "00000000000000000000000000000000";
678      -- i_in(1)  <= "00000000000000000000000000000000";
679      -- i_in(2)  <= "00000000000000000000000000000000";
680      -- i_in(3)  <= "00000000000000000000000000000000";
681      -- i_in(4)  <= "00000000000000000000000000000000";
682      -- i_in(5)  <= "00000000000000000000000000000000";
683      -- i_in(6)  <= "00000000000000000000000000000000";
684      -- i_in(7)  <= "00000000000000000000000000000000";
685      -- i_in(8)  <= "00000000000000000000000000000000";
686      -- i_in(9)  <= "00000000000000000000000000000000";
687      -- i_in(10) <= "00000000000000000000000000000000";
688      -- i_in(11) <= "00000000000000000000000000000000";
689      -- i_in(12) <= "00000000000000000000000000000000";
690      -- i_in(13) <= "00000000000000000000000000000000";
691      -- i_in(14) <= "00000000000000000000000000000000";
692      -- i_in(15) <= "00000000000000000000000000000000";
```

Listing 16: Test Bench della FFT