

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	Status PLA	9
4.2.2	ROM	10
4.2.3	datapath	10
5	Butterfly	11
6	Appendice - Simulazioni	12
7	Appendice - File VHDL	13
7.1	Sommatore	13
7.2	MUX	13
7.2.1	MUX 2	13
7.2.2	MUX 3	14
7.3	Sottrattore	14
7.4	Moltiplicatore/Shifter	15
7.5	ROM rounding	16
7.5.1	Blocco ROM rounding	16
7.5.2	Test Bench del ROM rounding	18
7.5.3	ROM	19
7.5.4	Test Bench della ROM	21
7.6	Datapath	21
7.7	Control Unit	30
7.7.1	Datapath	30
7.7.2	ROM	33
7.7.3	PLA	40
7.7.4	Test Bench della Control Unit	41
7.8	Butterfly	42
7.8.1	Test Bench della Butterfly singola	42

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'operazione butterfly, che è una struttura di manipolazione dei dati che esegue combinazioni lineari di dati complessi mediante somma, sottrazione e moltiplicazione con coefficienti complessi.

Lo scopo di questo progetto è progettare un'unità di elaborazione dedicata per eseguire la Butterfly FFT, utilizzando tecniche di microprogrammazione e considerando 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 della Scansione in Virgola Mobile a Blocco Incondizionata per gestire il sovraccarico e dell'implementazione di un datapath ottimizzato dati i vincoli di risorse computazionali limitate e pipeline interna. Il lavoro include la derivazione del diagramma di flusso dei dati dell'algoritmo, 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 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.

Una volta completata la singola butterfly è stato creato il processore che esegue la FFT unendo tra loro le varie unità necessarie per adempiere alla richiesta finale del progetto. Una volta implementato il tutto il sistema è stato testato nella sua interezza per constatare l'effettivo funzionamento.

2 Data Flow Diagram

In questo capitolo si parlerà delle specifiche imposte sui blocchi operazionali. Si andrà 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. In grado di poter svolgere sia l'operazione di moltiplicazione tra due numeri in ingresso sia come un moltiplicatore per 2 di un dato in ingresso. Le due operazioni sono selezionabili attraverso un segnale di controllo esterno al blocco operatore. 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. Mentre l'operazione di shift (moltiplicazione per 2) avesse un livello di pipeline, dunque l'uscita sarebbe 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.

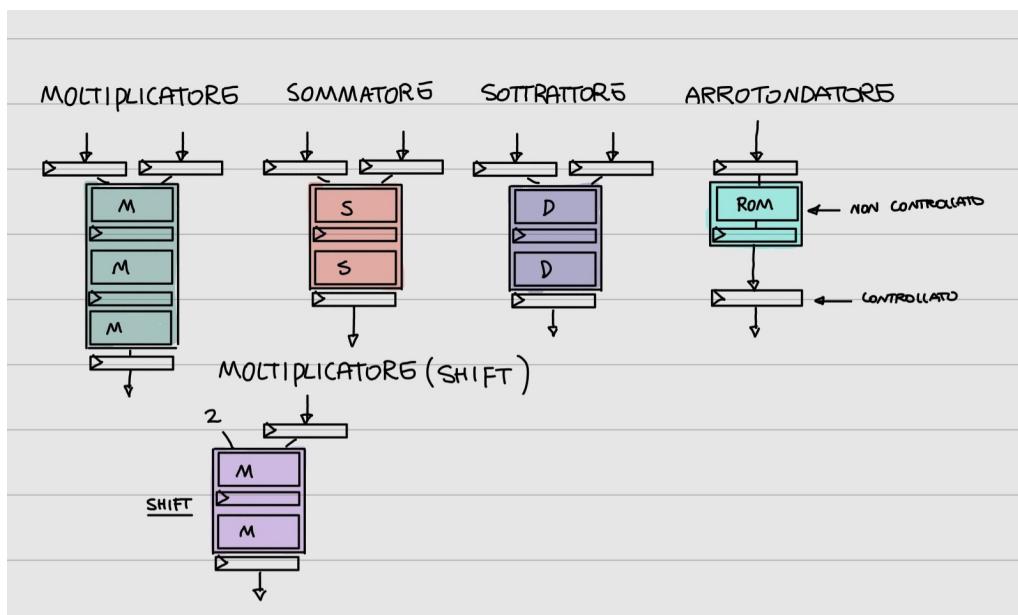


Figura 1: Blocchi elementari del data flow diagram

2.2 Approccio ASAP

L'approccio "As Soon As Possible", è un approccio che predilige lo svolgimento delle operazioni non appena si ha disponibilità. Come si può notare in *Fig. 2* sarebbero necessari 6 blocchi moltiplicatori e 2 blocchi sommatore e sottrattore. Questo tipo di schema non ci permette di rispettare dunque la specifica sul numero di blocchi operazionali.

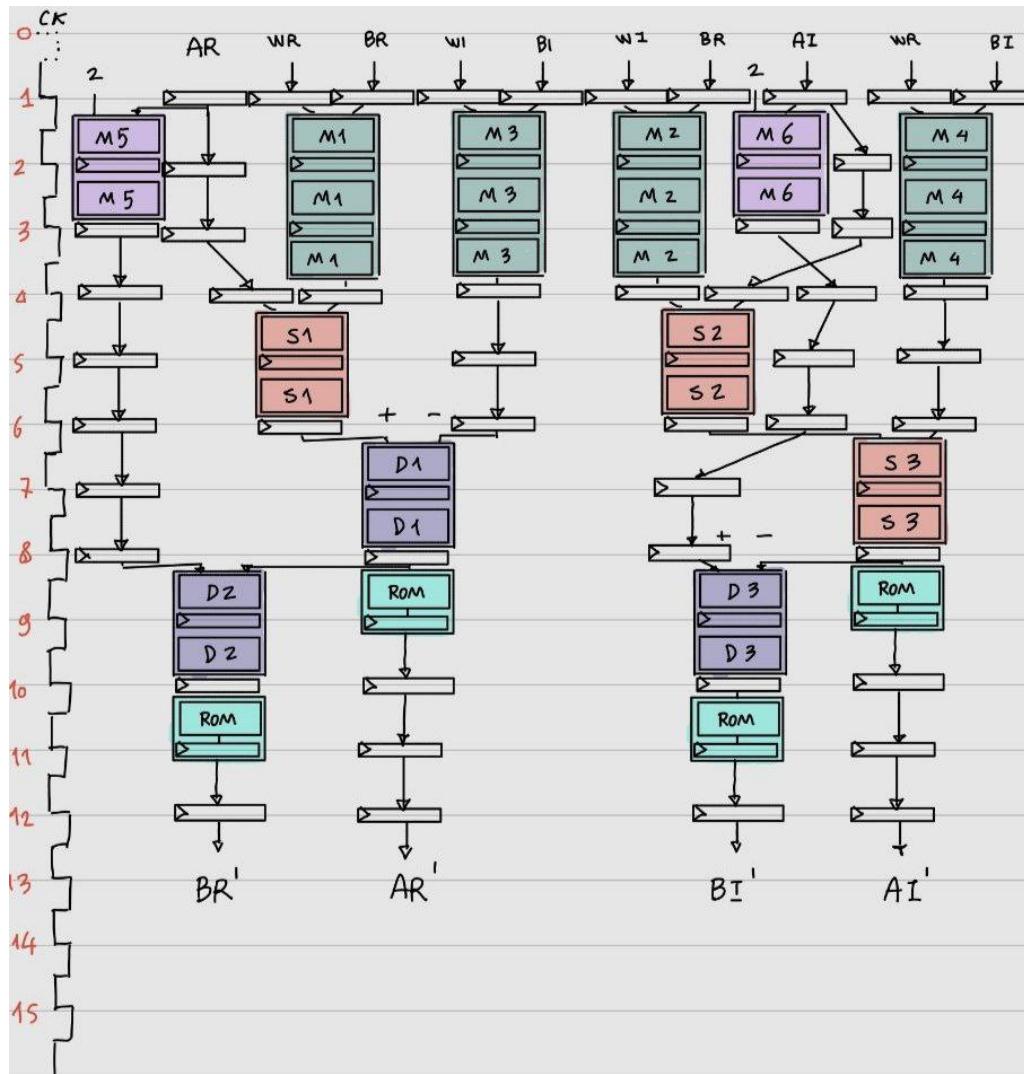


Figura 2: Data Flow Diagram con pproccio ASAP

2.3 Approccio ALAP

L'approccio "As Late As Possible", 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 è inferiore rispetto all'approccio ASAP, infatti vengono 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 per Butterfly. Verrà dunque studiato un approccio che ci permetta di rimanere entro le specifiche numeriche.

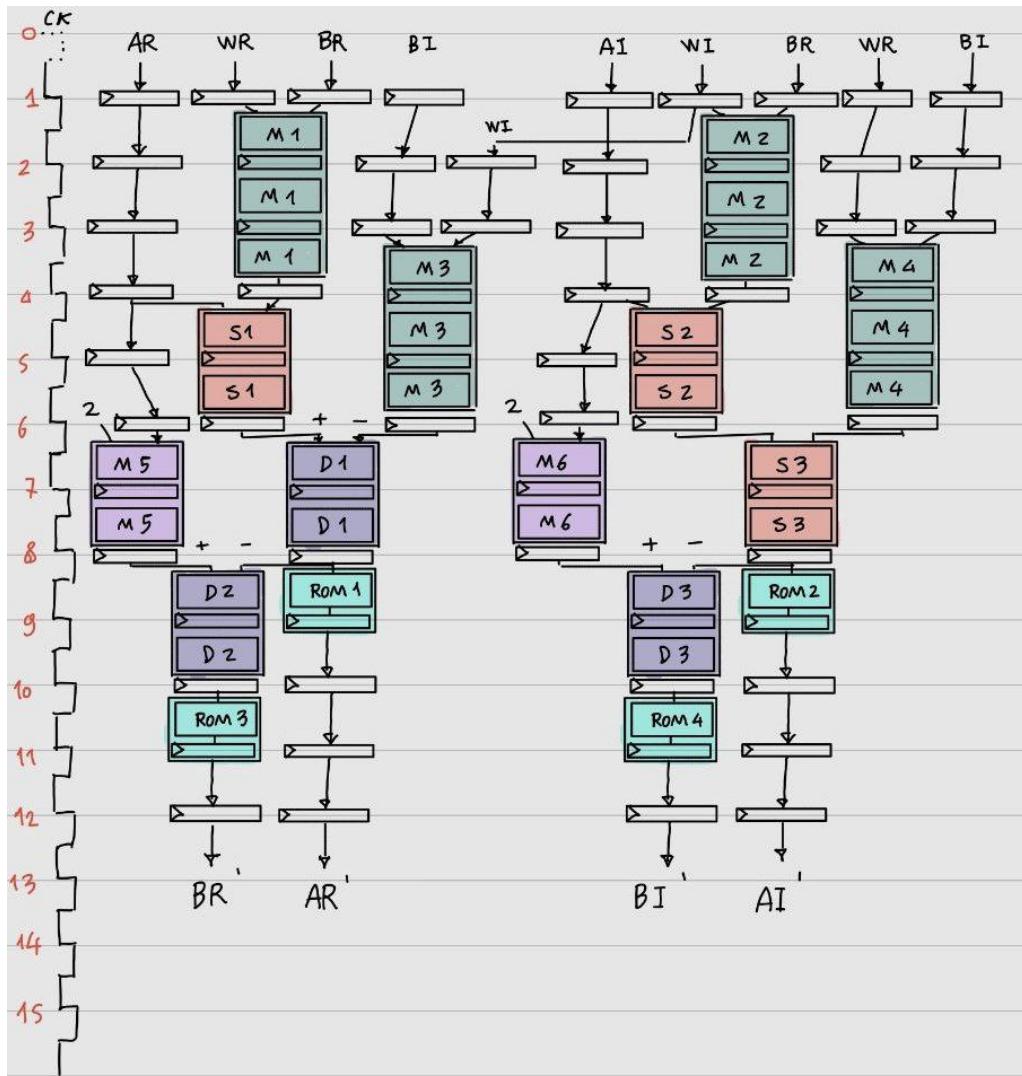


Figura 3: Data Flow Diagram con approccio ALAP

2.4 Approccio scelto

Per ottimizzare le tempistiche dell'algoritmo avendo delle restrizioni sul numero di operatori 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 operazionale per tipo di operazione, rientrando nelle specifiche imposte sul progetto.

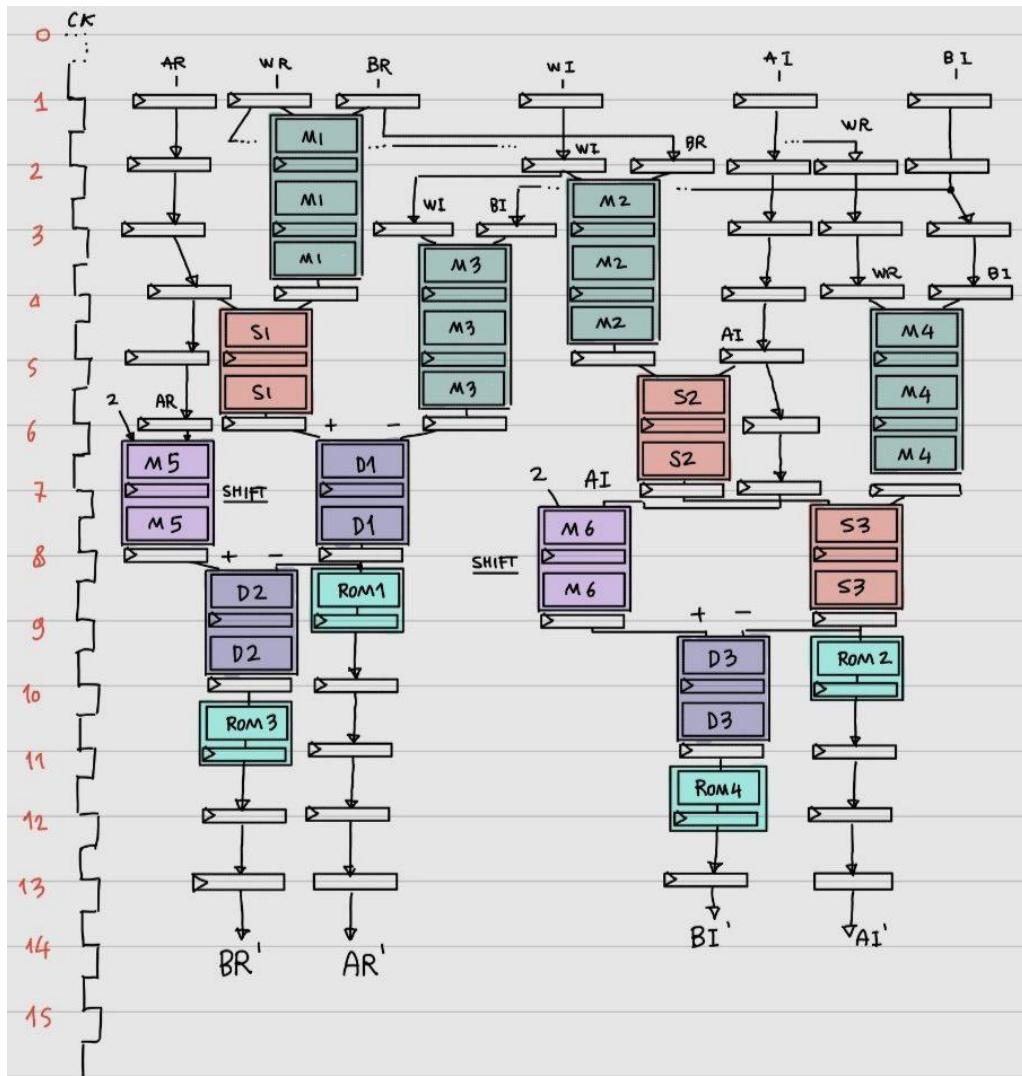


Figura 4: Data flow diagram ottimizzato

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 utilizzato. Questo ci è utile per capire quando un segnale deve essere conservato in un determinato registro, da questo possiamo ricavare i segnali, derivanti dalla control unit, per controllare i registri. Nel nostro schema gli unici registri che necessitano un segnale di controllo esterno sono i registri di ingresso e uscita della Butterfly. I registri posizionati tra i blocchi operazionali, dato che sono "vivi" per un solo colpo di clock non hanno bisogno di essere controllati.

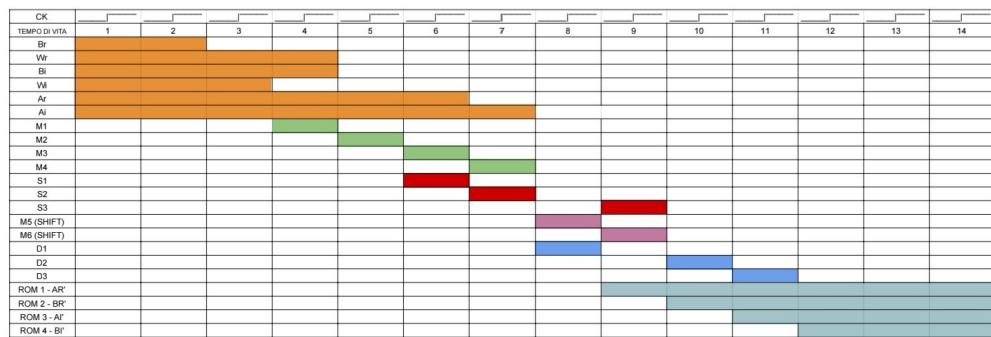


Figura 5: Tempo di vita delle variabili

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.

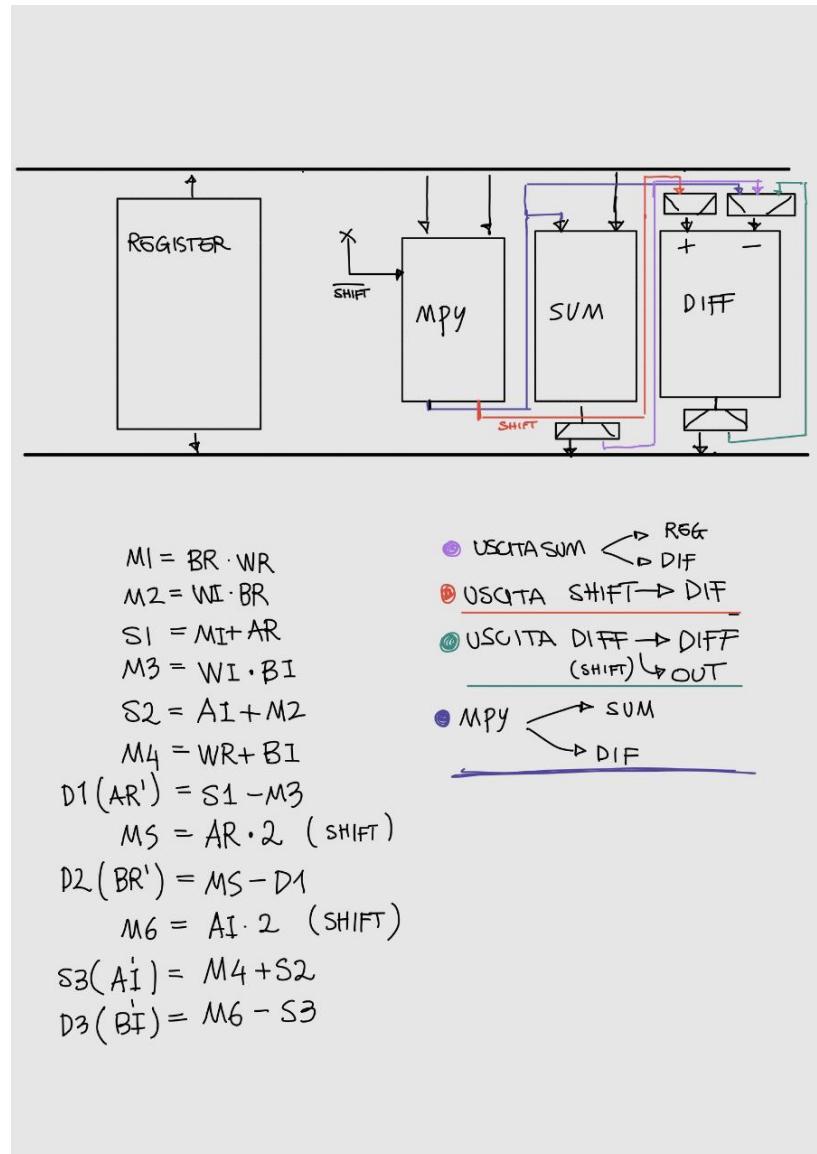


Figura 6: Schema del datapath iniziale

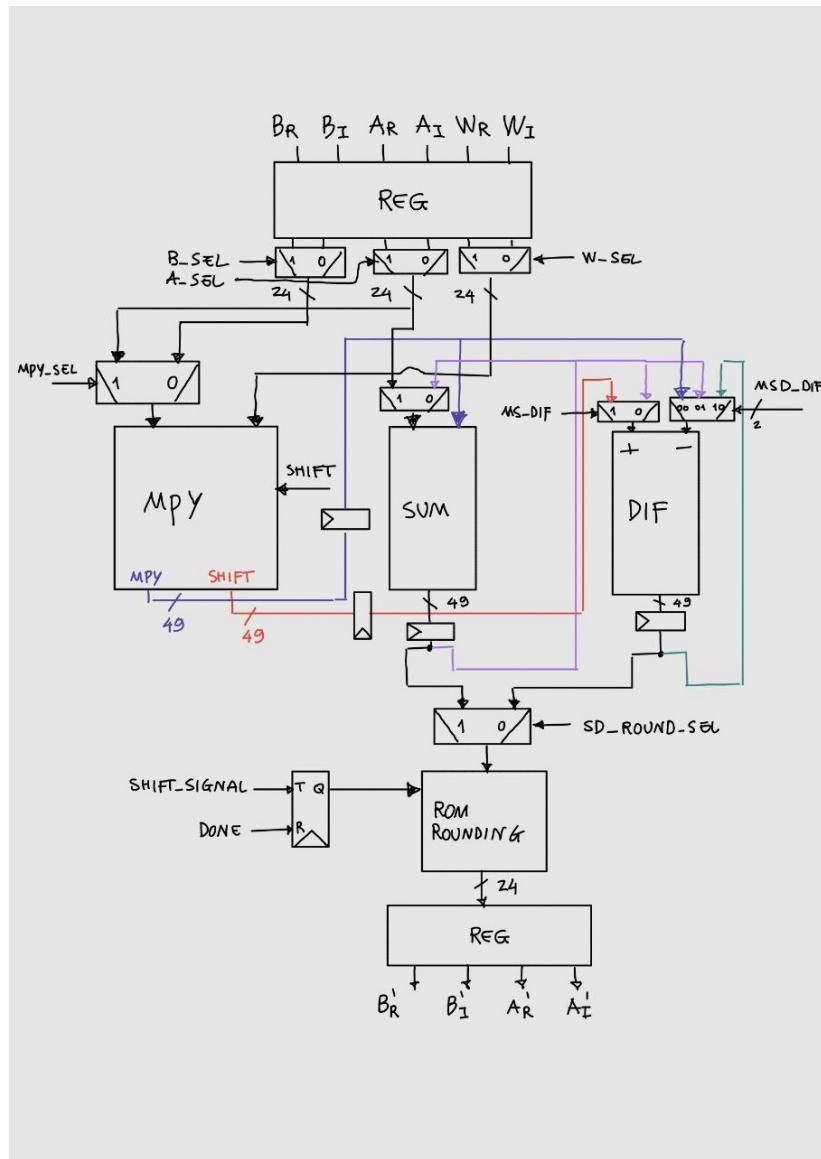


Figura 7: Schema del datapath finale

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'interna 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 peggio. 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. Considerando che si volevano salvare 5 bit per riga è stato scelto come numero di bit per l'indirizzo 5. Si riporta di seguito una tabella che mette in relazione il numero di bit dell'indirizzo con il numero di righe della ROM e con il numero di bit totali da memorizzare.

bit indirizzo	righe ROM	bit totali
3	8	40
5	32	160
7	128	640
9	512	2560

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:

bit indirizzo	bias	errore

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

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). 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 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.

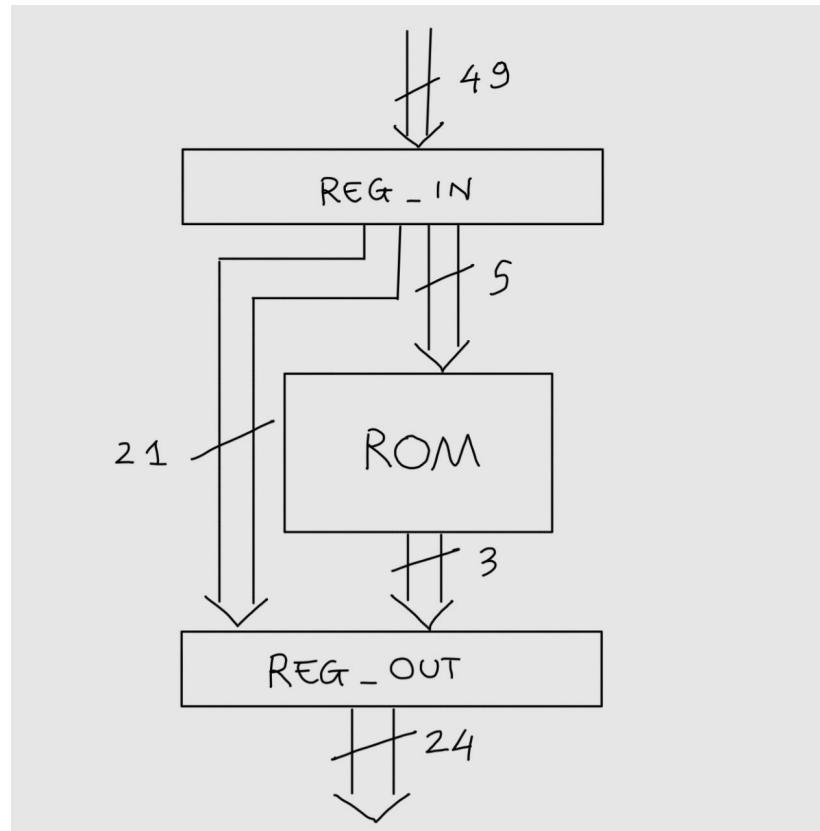


Figura 8: Schema del ROM rounding implementato

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* l'intera unità di controllo può essere suddivisa in tre parti: la status PLA, la ROM e un datapath in miniatura utilizzato per far muovere i segnali all'interno del sistema. Data l'importanza di questi tre macro blocchi verranno dedicati successivamente tre paragrafi per la descrizione dettagliata di quest'ultimi.

Si vuole poi porre enfasi sulla stati e sui comandi utilizzati. La scelta della codifica degli stati e dei comandi da utilizzare all'interno della butterfly è fondamentale per un corretto funzionamento del sistema. Per questo motivo si dedicherà un capitolo specifico dove verranno riportati i comandi e gli stati.

4.1 Comandi e stati

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

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	0	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 Struttura dell'unità di controllo

4.2.1 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 da un indirizzo all'altro, ciò che discrimina la necessità di effettuare o no un salto è il bit meno significativo dell'indirizzo stesso. Per facilitare questi

salti la ROM è stata suddivisa in locazioni pari e dispari (vedasi paragrafo "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 ROM

Come spiegato nel paragrafo precedente la ROM è stata suddivisa in indirizzi pari e dispari. Questo serve per regolare/facilitare i vari salti che può essere necessario compiere durante lo svolgimento delle operazioni. Si fa notare al lettore che dalla ROM esce contemporaneamente sia un il dato memorizzato in un indirizzo pari che uno memorizzato in un indirizzo dispari. Ciò che effettivamente arriva in uscita e su cui vengono fatte le successive considerazioni viene selezionato da un MUX posto in uscita alla ROM. Facendo in questo modo evitiamo di dover aspettare tutto il tempo necessario all'accesso in memoria.

4.2.3 datapath

Questo "datapath" non è da intendere come il datapath della butterfly spiegato nel relativo capitolo. In questo caso si vuole intendere solamente quell'insieme di collegamenti che permettono il corretto flusso dei dati che devono transitare all'interno dell'unità di controllo. Banalmente, serviranno una serie di bus per collegare le varie parti della Control Unit tra di loro e con il resto della butterfly per ottenere il corretto funzionamento del sistema. Si è voluto specificare quanto appena detto poiché nell'Appendice (*List. 7.7.1*) è presente del codice chiamato "control unit datapath" e si voleva fare un distinguo tra quello e il datapath vero e proprio.

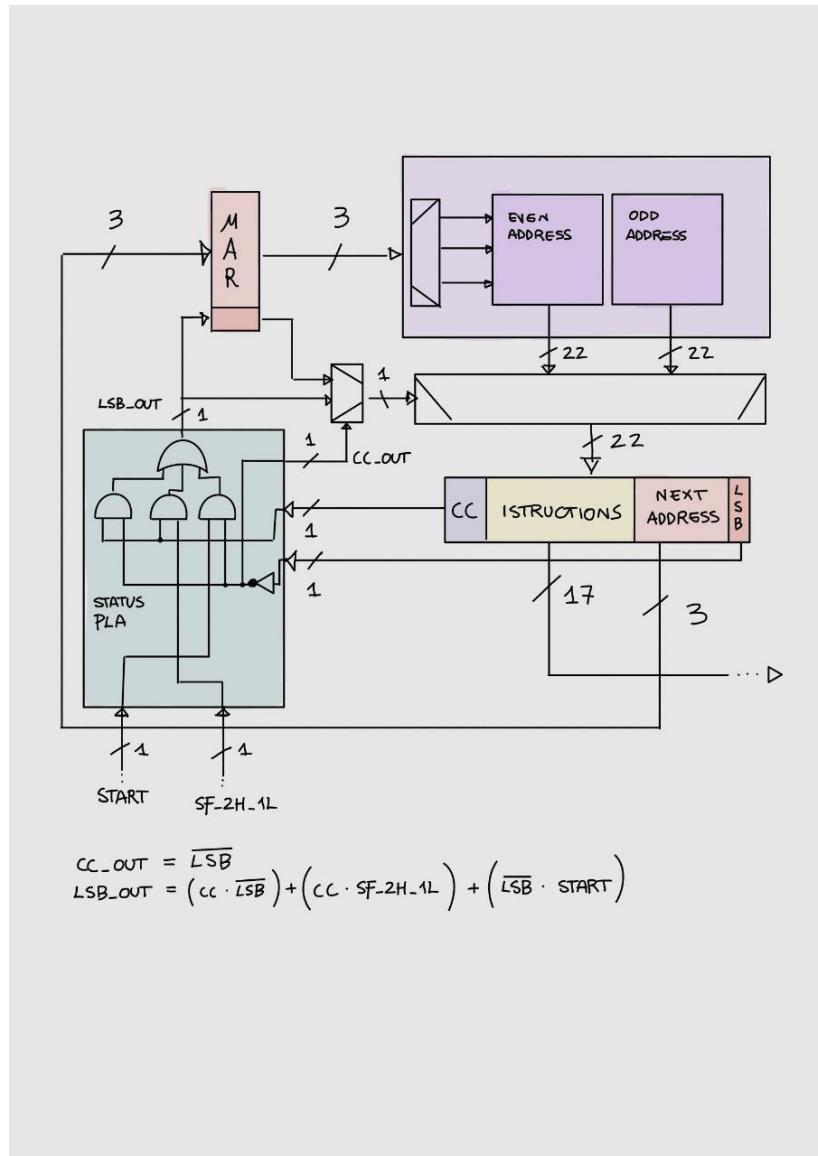


Figura 9: Schema della CU implementato

5 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.

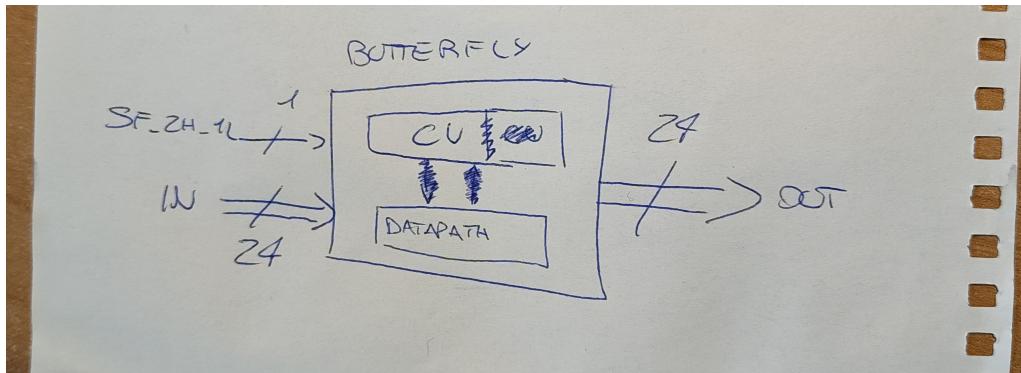


Figura 10: Rappresentazione di una butterfly singola

6 Appendice - Simulazioni

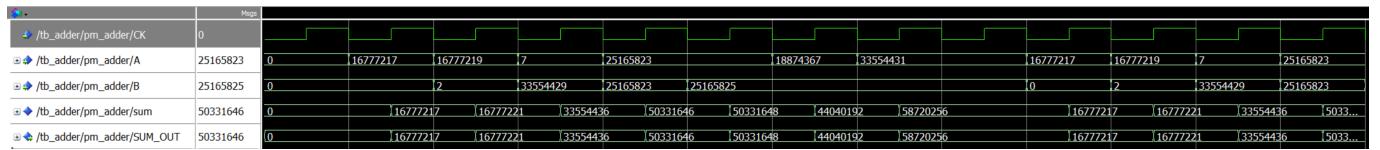


Figura 11: Simulazione del sommatore

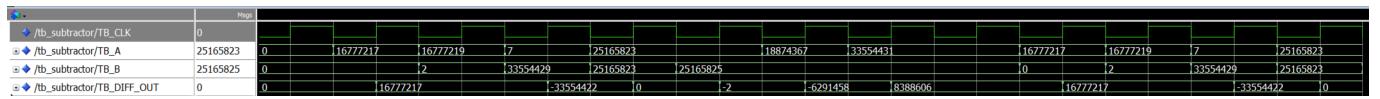


Figura 12: Simulazione del sottrattore

7 Appendice - File VHDL

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     PSYNC: 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

```

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

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity BFLY_MULTIPLIER is
    port ( A:      in      STD_LOGIC_VECTOR (23 downto 0);
           B:      in      STD_LOGIC_VECTOR (23 downto 0);
           SHIFT:  in STD_LOGIC;
           CK:     in      STD_LOGIC;
           S_OUT:   out     STD_LOGIC_VECTOR(48 downto 0);
           M_OUT:   out     STD_LOGIC_VECTOR(48 downto 0)
         );
end BFLY_MULTIPLIER;

architecture behavioral of BFLY_MULTIPLIER is

    signal op_A, op_B: STD_LOGIC_VECTOR (23 downto 0) := (others=>'0');
    signal product: STD_LOGIC_VECTOR (47 downto 0) := (others=>'0');
    signal S_OUT_tmp, M_OUT_tmp: STD_LOGIC_VECTOR (47 downto 0) := (others=>'0');
    signal S_OUT_trunc, M_OUT_trunc: STD_LOGIC_VECTOR (46 downto 0) := (others=>'0');

begin
    begin
        op_A <= A;
        op_B <= "000000000000000000000000010" when SHIFT = '1' else B;
        product <= std_logic_vector(signed(op_A)*signed(op_B));
        S_OUT_trunc <= S_OUT_tmp(46 downto 0);
        M_OUT_trunc <= M_OUT_tmp(46 downto 0);
        S_OUT <= '0' & '0' & S_OUT_trunc;
        M_OUT <= '0' & '0' & M_OUT_trunc;
    end;

```

```

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(2 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 begin
53
54      -----
55      -- Port map
56      -----
57      pm_reg_rom_out : FD
58          generic map (
59              bus_length => 24
60          )
61          port map (
62              D => reg_in,
63              E => '1',
64              CK => Clock,
65              Q => rounding_out
66          );
67
68      pm_ROM : ROM
69          port map(
70              address => address_memory,
71              memory_out => dummy_memory_out
72          );
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      -- Creazione mantissa e bit di scarto
83      -----
84
85      mantissa    <= shift_dummy(46 downto 23);
86      bit_scarto <= shift_dummy(22 downto 21);

87      -----
88      -- Creazione dell'indirizzo per leggere dalla ROM
89      -----
90
91      -- address = (3 bit LSB mantissa) + (1 bit MSB scarto) + (1 bit OR con tutti gli
92      -- altri dello scarto)
93      address_memory <= mantissa(2 downto 0) & bit_scarto;

94
95      -----
96      -- Inserimento dati nel registro d'uscita del blocco
97      -----
98
99      reg_in <= mantissa(23 downto 3) & dummy_memory_out; -- 21 bit di mantissa & 3 bit
100         arrotondamento
101
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 <= "1100101110101110110110000010001110010110011100111";
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 <= "1000010011010101101000100011100110111000101101";
58     shift_signal <= '1';
59     wait for Tclk;
60     -- Caso 5
61     rounding_in <= "110000101111000010011001111001010000011110110100";
62     shift_signal <= '1';
63     wait for Tclk;
64     -- Caso 6

```

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 -- Architetture della ROM
20
21 architecture ROM_rounding of ROM is
22
23     -- Spazio per segnali interni
24
25 begin
26
27 selection: process(address)
28 begin
```

```
29
30  if address = "00000" then
31      memory_out <= "000";
32  elsif address = "00001" then
33      memory_out <= "000";
34  elsif address = "00010" then
35      memory_out <= "000";
36  elsif address = "00011" then
37      memory_out <= "001";
38  elsif address = "00100" then
39      memory_out <= "001";
40  elsif address = "00101" then
41      memory_out <= "001";
42  elsif address = "00110" then
43      memory_out <= "010";
44  elsif address = "00111" then
45      memory_out <= "010";
46  elsif address = "01000" then
47      memory_out <= "010";
48  elsif address = "01001" then
49      memory_out <= "010";
50  elsif address = "01010" then
51      memory_out <= "010";
52  elsif address = "01011" then
53      memory_out <= "011";
54  elsif address = "01100" then
55      memory_out <= "011";
56  elsif address = "01101" then
57      memory_out <= "011";
58  elsif address = "01110" then
59      memory_out <= "100";
60  elsif address = "01111" then
61      memory_out <= "100";
62  elsif address = "10000" then
63      memory_out <= "100";
64  elsif address = "10001" then
65      memory_out <= "100";
66  elsif address = "10010" then
67      memory_out <= "100";
68  elsif address = "10011" then
69      memory_out <= "101";
70  elsif address = "10100" then
71      memory_out <= "101";
72  elsif address = "10101" then
73      memory_out <= "101";
74  elsif address = "10110" then
75      memory_out <= "110";
76  elsif address = "10111" then
77      memory_out <= "110";
78  elsif address = "11000" then
79      memory_out <= "110";
80  elsif address = "11001" then
81      memory_out <= "110";
82  elsif address = "11010" then
83      memory_out <= "110";
84  elsif address = "11011" then
85      memory_out <= "111";
86  elsif address = "11100" then
87      memory_out <= "111";
88  elsif address = "11101" then
89      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  -----
24  --Inizializzazione componenti
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');
140
141  --Uscite e ingressi del multiplier
142  SIGNAL dp_X_MPY_in , dp_Y_MPY_in : STD_LOGIC_VECTOR (23 downto 0) := (others =>
143    '0');
144  SIGNAL dp_MPY_product_out , dp_MPY_shift_out : STD_LOGIC_VECTOR (48 downto 0) := (
145    others => '0');

```

```

137
138 --Uscita e ingressi dell'adder
139 SIGNAL dp_SUM_out, dp_X_SUM_in, dp_Y_SUM_in : STD_LOGIC_VECTOR (48 downto 0) := (
140     others => '0');
141
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
146 --Uscita del registro di pipe della somma
147 SIGNAL dp_SUM_reg_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
148
149 --Uscita del registro di pipe del sottrattore
150 SIGNAL dp_DIFF_reg_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
151
152 --Uscita del registro di pipe del prodotto e dello shift
153 SIGNAL dp_MPY_M_reg_out, dp_MPY_S_reg_out : STD_LOGIC_VECTOR (48 downto 0) := (
154     others => '0');
155
156 --Ingresso 1 del multiplexer in entrata al sommatore, ovvero l'uscita del MUX di
157 --Ar/Ai con zeri aggiunti
158 SIGNAL dp_AS_A_MUX_in : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
159 --Uscita del multiplexer in entrata al sommatore
160 SIGNAL dp_AS_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
161
162 --Uscita del MUX A/B in ingresso al multiplier
163 SIGNAL dp_AB_MUX_out : STD_LOGIC_VECTOR (23 downto 0) := (others => '0');
164
165 --Uscita del MUX dell'ingresso positivo del sottrattore
166 SIGNAL dp_MS_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
167
168 --Uscita del MUX dell'ingresso negativo del sottrattore
169 SIGNAL dp_MSD_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
170
171 --Uscita del MUX dell'ingresso dello shifter a destra
172 SIGNAL dp_SD_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
173
174 begin
175 -----
176 --Port map dei registri a 24 bit
177 -----
178
179 pm_regin_Br : FD
180     generic map (
181         bus_length => 24
182     )
183     port map (
184         D => Br_in,
185         E => dp_REG_IN,
186         CK => Clock,
187         Q => dp_Br_MUX_in
188 );
189
190 pm_regin_Bi : FD
191     generic map (
192         bus_length => 24
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 ,
321      Q => dp_MS_MUX_out
322  );
323
324  pm_mux_rshift : MUX_2      --Multiplexer dell'ingresso allo shifter a destra
325  generic map (
326      bus_length => 49
327  )
328  port map (
329      A => dp_SUM_reg_out ,      --l'uscita del sommatore
330      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                 ralzentata di un colpo di Clock
339             B => dp_SUM_reg_out,          --l'uscita del sommatore
340                 ralzentata di un colpo di Clock
341             C => dp_DIFF_reg_out,          --l'uscita del sottrattore
342                 ralzentata 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,
387             DIFF_OUT => dp_DIFF_out
388     );
389
390     pm_ft_shift : T_FF        --Port map del flip flop T che ha come uscita il segnale
391             di SF_2H_1L per il blocco rounding

```

```

383      port map (
384          T => dp_SHIFT_SIGNAL,      --Segnale che viene dalla CU
385          R => dp_DONE,             --Segnale che viene dalla CU
386          CK => Clock,
387          Q => dp_SF_2H_1L
388      );
389
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
399      pm_CU : BFLY CU DATAPATH           --Port map della Control unit
400          port map (
401              START => START,
402              SF_2H_1L => SF_2H_1L,
403              CK => Clock,
404              INSTRUCTION_OUT => dp_INSTRUCTION_OUT
405          );
406
407          --Segnali della parte di istruzione del uIR della CU
408          dp_SHIFT_SIGNAL <= dp_INSTRUCTION_OUT(16);
409          dp_REG_IN <= dp_INSTRUCTION_OUT(15);
410          dp_SUM_REG <= dp_INSTRUCTION_OUT(14);
411          dp_AR_SEL <= dp_INSTRUCTION_OUT(13);
412          dp_BR_SEL <= dp_INSTRUCTION_OUT(12);
413          dp_WR_SEL <= dp_INSTRUCTION_OUT(11);
414          dp_MS_DIFFp <= dp_INSTRUCTION_OUT(10);
415          dp_MSD_DIFFm <= dp_INSTRUCTION_OUT(9 downto 8);
416          dp_AS_SUM_SEL <= dp_INSTRUCTION_OUT(7);
417          dp_SD_ROUND_SEL <= dp_INSTRUCTION_OUT(6);
418          dp_REG_RND_BR <= dp_INSTRUCTION_OUT(5);
419          dp_REG_RND_BI <= dp_INSTRUCTION_OUT(4);
420          dp_REG_RND_AR <= dp_INSTRUCTION_OUT(3);
421          dp_REG_RND_AI <= dp_INSTRUCTION_OUT(2);
422          dp_SHIFT <= dp_INSTRUCTION_OUT(1);
423          dp_DONE <= dp_INSTRUCTION_OUT(0);
424
425          DONE <= dp_DONE;
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
443          pm_reg_MPY_shift_out : FD           --Port map del registro all'uscita shift del
444              multiplier
445                  generic map (
446                      bus_length => 49
447                  )

```

```
446      port map (
447          D => dp_MPY_shift_out ,
448          E => '1',
449          CK => Clock,
450          Q => dp_MPY_S_reg_out
451      );
452
453      pm_reg_SUM_out : FD      --Port map del registro all'uscita del sommatore
454      generic map (
455          bus_length => 49
456      )
457      port map (
458          D => dp_SUM_out ,
459          E => '1',
460          CK => Clock,
461          Q => dp_SUM_reg_out
462      );
463
464      pm_reg_DIFF_out : FD      --Port map del registro all'uscita del sottrattore
465      generic map (
466          bus_length => 49
467      )
468      port map (
469          D => dp_DIFF_out ,
470          E => '1',
471          CK => Clock,
472          Q => dp_DIFF_reg_out
473      );
474
475      -----
476      --Port map dei registri di uscita a 24 bit
477      -----
478
479      pm_regout_Br : FD
480      generic map (
481          bus_length => 24
482      )
483      port map (
484          D => dp_ROM_round_out ,
485          E => dp_REG_RND_BR ,
486          CK => Clock,
487          Q => Br_out
488      );
489
490      pm_regout_Bi : FD
491      generic map (
492          bus_length => 24
493      )
494      port map (
495          D => dp_ROM_round_out ,
496          E => dp_REG_RND_BI ,
497          CK => Clock,
498          Q => Bi_out
499      );
500
501      pm_regout_Ar : FD
502      generic map (
503          bus_length => 24
504      )
505      port map (
506          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
87  dp_STATUS(0) <= START;
88  dp_STATUS(1) <= SF_2H_1L;
89
90  INSTRUCTION_OUT <= microIR_out (20 downto 4);
91  CC_validation <= microIR_out (21);
92  next_Address_LSB <= microIR_out (0);
93
94  next_Address_MSB(2 downto 0) <= microIR_out (3 downto 1);
95
96  microAR_in_MSB <= next_Address_MSB;
97  microAR_in_LSB <= status_PLA_LSB_out;
98
99  microIR_in <= PLA_ROM_mux_out;
100
101 --PLA
102 pm_PLA : BFLY CU_LATE_STATUS_PLA
103 port map (
104     STATUS => dp_STATUS,
105     LSB_in => next_Address_LSB,
106     CC_Validation_in => CC_validation,
107     CC_Validation_out => status_PLA_CC_validation_out,
108     LSB_out => status_PLA_LSB_out
109 );
110
111 --ROM della PLA
112 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;
59     out_tmp_even(14) <= MS_DIFFp_even;
60     out_tmp_even(13 downto 12) <= MSD_DIFFm_even;
61     out_tmp_even(11) <= AS_SUM_SEL_even;
62     out_tmp_even(10) <= SD_ROUND_SEL_even;
63     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              -- IDLE / START
100
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 --M1 , SH0 / M1 , SH1
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
184      elsif A = "010" then --M2 / M3
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);
49     REG_RND_BI <= TB_INSTRUCTION_OUT(4);
50     REG_RND_AR <= TB_INSTRUCTION_OUT(3);
51     REG_RND_AI <= TB_INSTRUCTION_OUT(2);
52     SHIFT <= TB_INSTRUCTION_OUT(1);
53     DONE <= TB_INSTRUCTION_OUT(0);

```

```

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

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; -- libreria IEEE con definizione tipi standard logic
7 use IEEE.numeric_std.all;
8
9
10 entity TB_BFLY_SINGLE is
11 end TB_BFLY_SINGLE;
12
13 -----
14
15 architecture behavioral of TB_BFLY_SINGLE is
16
17     component bfly_datapath is
18     port(
19         Br_in, Bi_in, Ar_in, Ai_in, Wr_in, Wi_in : in STD_LOGIC_VECTOR (23 downto
20             0);
21         Clock, START, SF_2H_1L : in STD_LOGIC;
22         Br_out, Bi_out, Ar_out, Ai_out : out STD_LOGIC_VECTOR (23 downto 0);
23         DONE : out STD_LOGIC
24     );
25     end component;
26
27     constant period : time := 100 ns;
28
29     SIGNAL TB_Clock, TB_START, TB_SF_2H_1L, TB_DONE : STD_LOGIC := '0';
30     SIGNAL TB_Br_in, TB_Bi_in, TB_Ar_in, TB_Ai_in, TB_Wr_in, TB_Wi_in :
31         STD_LOGIC_VECTOR (23 downto 0) := (others => '0');
32     SIGNAL TB_Br_out, TB_Bi_out, TB_Ar_out, TB_Ai_out : STD_LOGIC_VECTOR (23 downto 0)
33         := (others => '0');
34
35     SIGNAL UNS_Br, UNS_Bi, UNS_Ar, UNS_Ai, UNS_Wr, UNS_Wi : UNSIGNED (23 downto 0);

```

```
34
35     begin
36
37         TB_Clock <= not TB_Clock after period/2;
38
39     process
40     begin
41         wait for 2*period/2;
42         TB_START <= '1';
43         TB_SF_2H_1L <= '0';
44
45         wait for 2*period/2;
46         TB_START <= '0';
47
48         wait for 38*period/2;
49
50         wait for 2*period/2;
51
52
53         TB_Br_in <= "010001000000010000000011";
54         TB_Bi_in <= "010001000000010000000011";
55
56         TB_Ar_in <= "000100010000000010000000";
57         TB_Ai_in <= "000100010000000010000000";
58
59
60         TB_Wr_in <= "00000000000000000000000010";
61         TB_Wr_in <= "00000000000000000000000010";
62
63
64         TB_START <= '1';
65         TB_SF_2H_1L <= '0';
66
67         wait for 2*period/2;
68         TB_START <= '0';
69
70
71         wait for 38*period/2;
72
73         wait for 2*period/2;
74
75
76         TB_Br_in <= "00000000000000000000000010";
77         TB_Bi_in <= "00000000000000000000000010";
78
79         TB_Ar_in <= "00000000000000000000000010";
80         TB_Ai_in <= "00000000000000000000000010";
81
82
83         TB_Wr_in <= "00000000000000000000000010";
84         TB_Wi_in <= "00000000000000000000000010";
85
86
87         TB_START <= '1';
88         TB_SF_2H_1L <= '0';
89
90         wait for 2*period/2;
91         TB_START <= '0';
92
93
94         wait for 38*period/2;
95
96     end process;
97
98
99     pm_bfly_datapath : bfly_datapath port map (
```

```
100      Br_in  => TB_Br_in ,
101      Bi_in  => TB_Bi_in ,
102      Ar_in  => TB_Ar_in ,
103      Ai_in  => TB_Ai_in ,
104      Wr_in  => TB_Wr_in ,
105      Wi_in  => TB_Wi_in ,
106      Clock  => TB_Clock ,
107      START   => TB_START ,
108      SF_2H_1L => TB_SF_2H_1L ,
109      Br_out  => TB_Br_out ,
110      Bi_out  => TB_Bi_out ,
111      Ar_out  => TB_Ar_out ,
112      Ai_out  => TB_Ai_out ,
113      DONE    => TB_DONE
114      );
115
116 end behavioral;
```

Listing 15: Test Bench della Control Unit