

**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

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Data Flow Diagram</b>	<b>2</b>
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
<b>3</b>	<b>Datapath</b>	<b>6</b>
3.1	ROM Rounding . . . . .	8
<b>4</b>	<b>Control Unit</b>	<b>11</b>
<b>5</b>	<b>Appendice</b>	<b>12</b>
5.1	Sommatore . . . . .	12
5.2	MUX . . . . .	12
a.	MUX 2 . . . . .	12
b.	MUX 3 . . . . .	13
5.3	Sottrattore . . . . .	13
5.4	Moltiplicatore/Shifter . . . . .	14
5.5	ROM rounding . . . . .	15
a.	ROM . . . . .	18
5.6	Datapath . . . . .	20
5.7	Control Unit . . . . .	28
a.	Datapath . . . . .	28
b.	ROM . . . . .	31
c.	PLA . . . . .	39
d.	Test Bench . . . . .	39

# 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 due elementi per le operazioni di somma e differenza. 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.

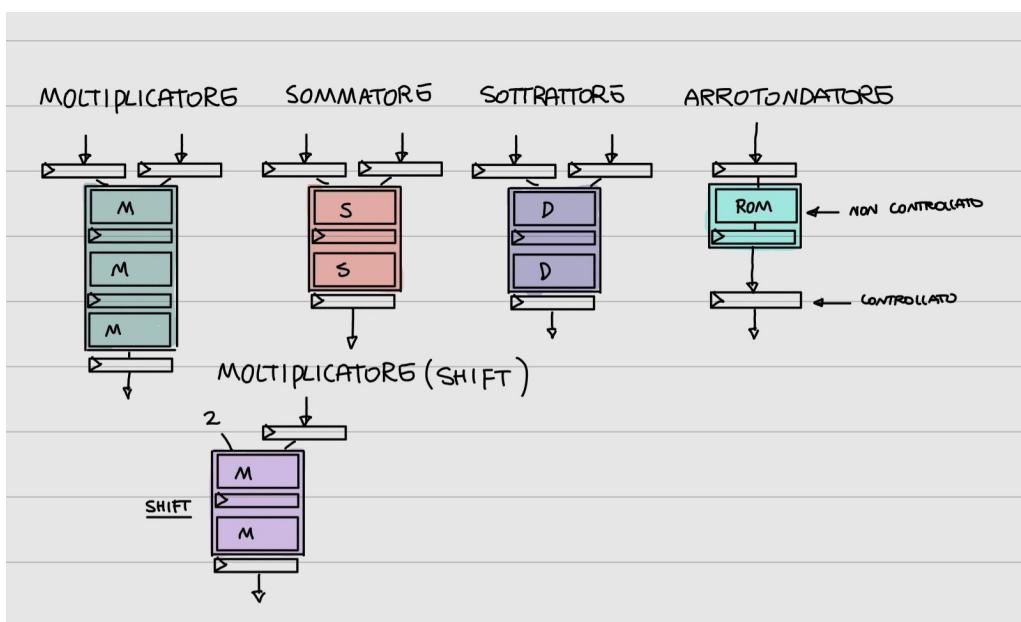


Figura 1: Blocchi elementari del data flow diagram

### 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.

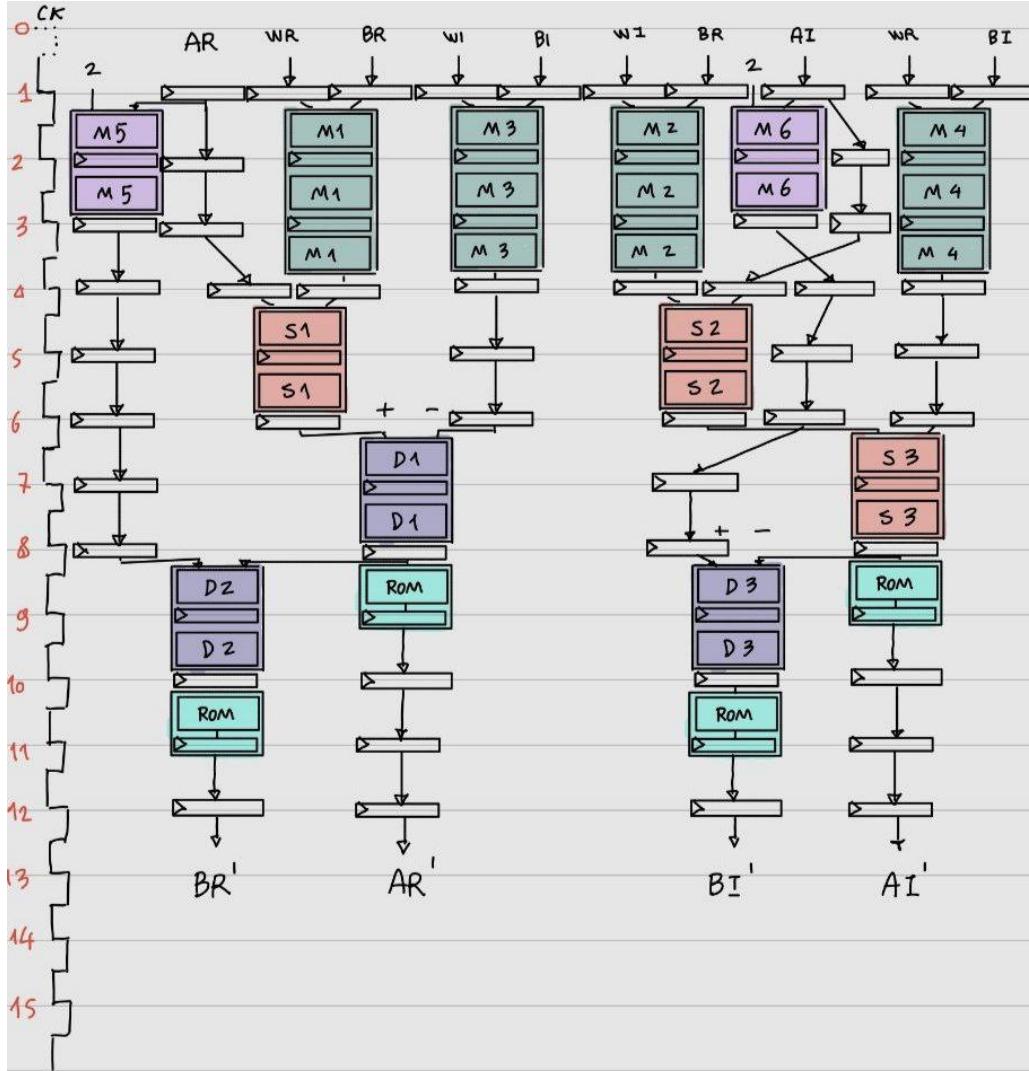


Figura 2: Data Flow Diagram con pproccio ASAP

### 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.

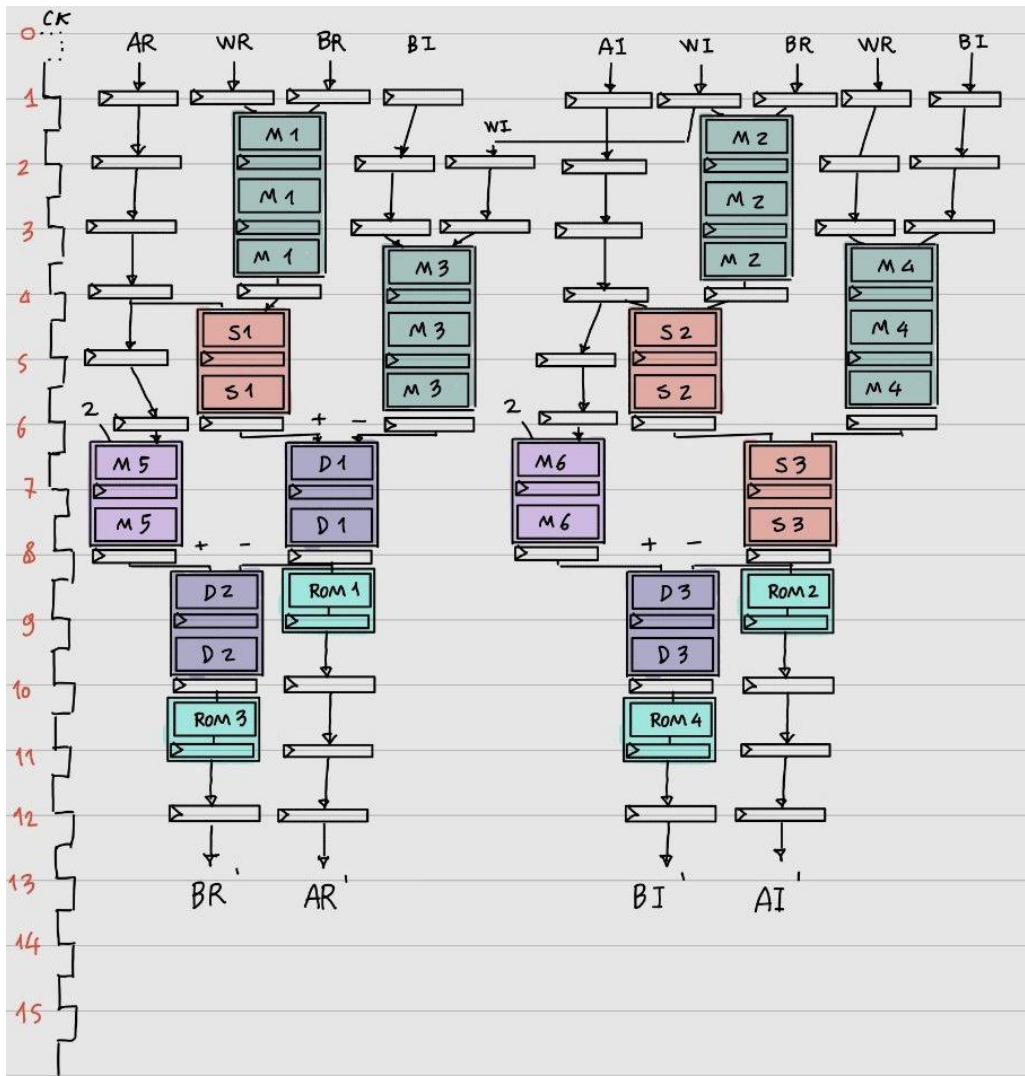


Figura 3: Data Flow Diagram con approccio ALAP

## 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.

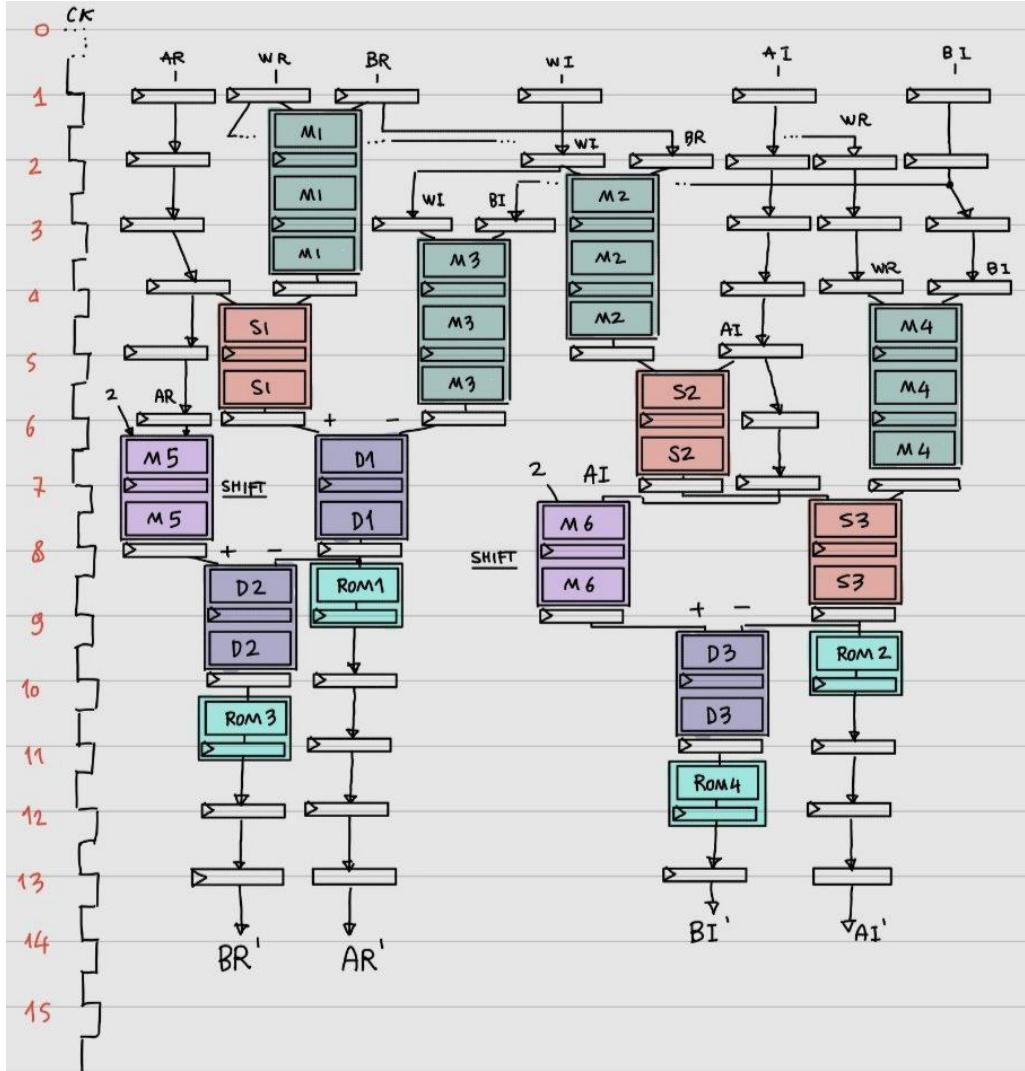


Figura 4: Data flow diagram ottimizzato

## 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.

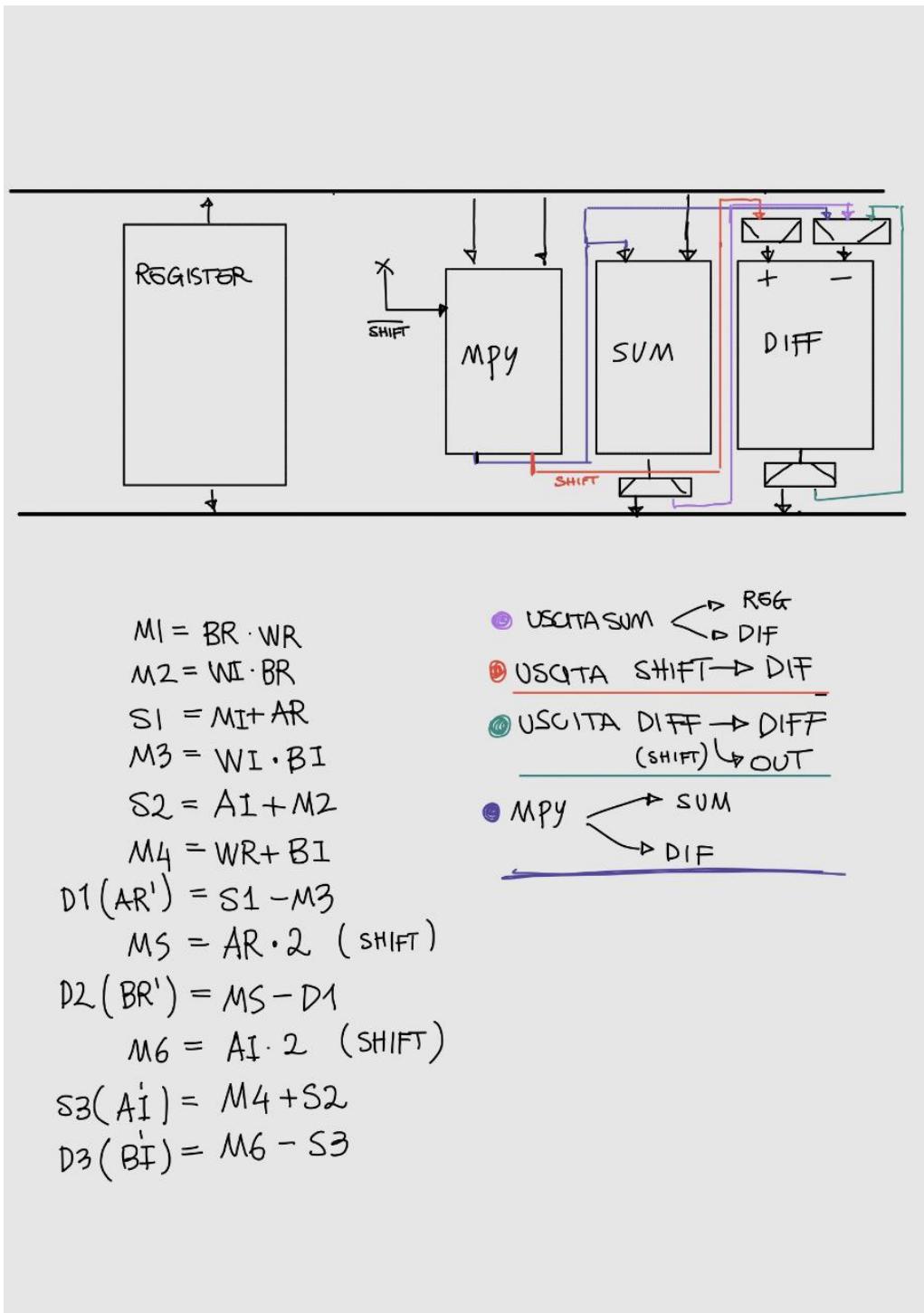


Figura 6: Schema del datapath iniziale

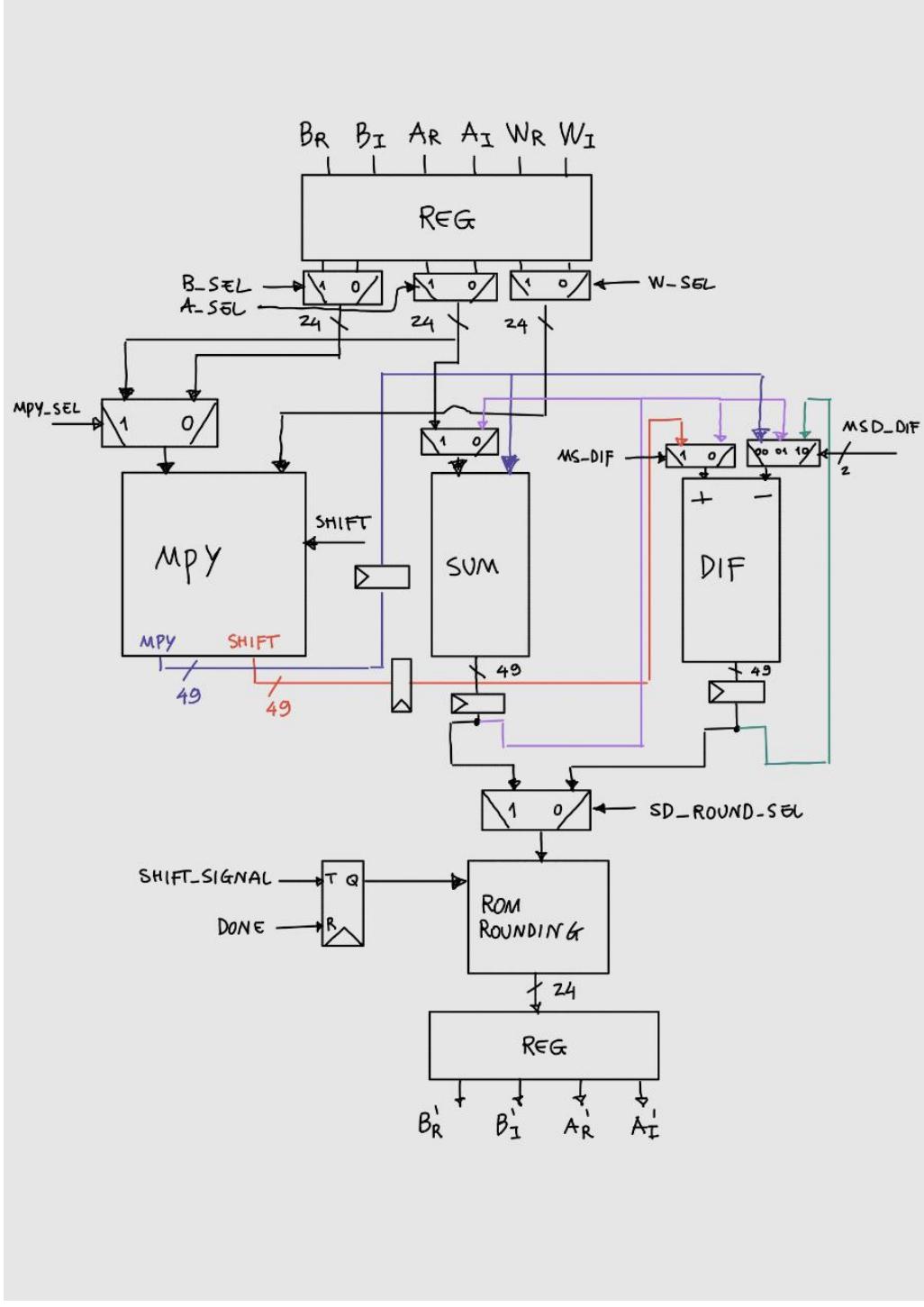


Figura 7: Schema del datapath finale

### 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
3		
5		
7		
9		

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.

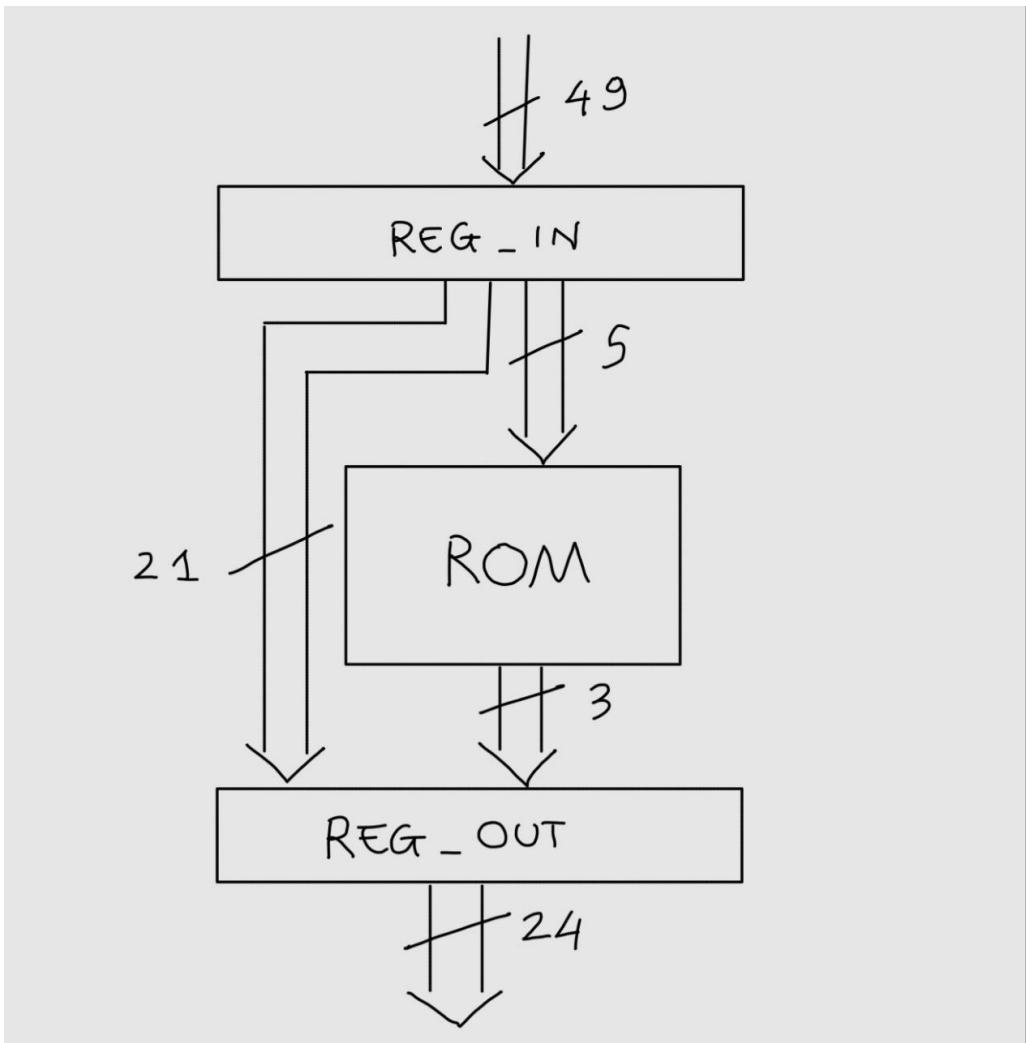


Figura 8: Schema del ROM rounding implementato

## 4 Control Unit

## 5 Appendice

### 5.1 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     PSYNCN: 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

### 5.2 MUX

#### a. 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

### b. 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

### 5.3 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

## 5.4 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 <= "000000000000000000000000010" 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

## 5.5 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 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 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      -----
77      -- Shift senza processo logico (non impiega colpi di clock)
78      -----
79      shift_dummy <=
80          '0' & '0' & rounding_in(48 downto 2) when shift_signal = '1' else '0' &
81          rounding_in(48 downto 1);
82
83      -----
84      -- Creazione mantissa e bit di scarto
85      -----
86
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)
94      address_memory <= mantissa(2 downto 0) & bit_scarto;
95
96      -----
97      -- Inserimento dati nel registro d'uscita del blocco
98      -----
99
100     reg_in <= mantissa(23 downto 3) & dummy_memory_out; -- 21 bit di mantissa & 3 bit
101        arrotondamento
102
102 end architecture behavioural;

```

Listing 6: ROM rounding completo

```

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  -- Creazione entity
10 entity tb_rounding is
11 end entity;
12
13 -- Architettura del TB
14 architecture sim of tb_rounding is
15
16     signal Clock : std_logic := '0';
17     signal rounding_in : std_logic_vector(48 downto 0);
18     signal rounding_out : std_logic_vector(23 downto 0);
19     signal shift_signal: std_logic := '0';
20
21 constant Tclk : time := 10 ns; -- 10ns -> 100MHz
22
23 begin
24
25     -----
26     -- Port map
27     -----
28 DUT : entity work.rounding
29 port map (
30     Clock      => Clock,
31     rounding_in => rounding_in,
32     rounding_out => rounding_out,
33             shift_signal => shift_signal
34 );
35
36 clk_proc : process
37 begin
38     Clock <= '0';
39     wait for Tclk/2;
40     Clock <= '1';
41     wait for Tclk/2;
42 end process;
43
44 stim_proc : process
45 begin
46     rounding_in <= "1100101110101110110110000010001110010110011100111";
47         shift_signal <= '1';
48     wait for Tclk;
49     rounding_in <= "1000111101001011101000011001011001111100111000";
50         shift_signal <= '1';
51     wait for Tclk;
52         rounding_in <= "000101110101101110100010111101110011101001000000";
53         shift_signal <= '0';
54     wait for Tclk;
55     rounding_in <= "10000100110101101000100011100111010111000101101";
56         shift_signal <= '1';
57     wait for Tclk;
58         rounding_in <= "1100001011110000100110011110010100000111110110100";
59         shift_signal <= '1';
60     wait for Tclk;
61         rounding_in <= "010101001010011000011010101110010101100111111011";
62         shift_signal <= '0';
63     wait for Tclk;
64         rounding_in <= "1101101010011101001111011011011011100011011011";
65         shift_signal <= '1';
66     wait for Tclk;
67         rounding_in <= "0000110001110011000110100111001000011001111101011";
68         shift_signal <= '0';
69     wait for Tclk;
70         rounding_in <= "11010101001011010101000000111001110110111010011";
71         shift_signal <= '1';
72     wait for Tclk;

```

```

73         rounding_in <= "101100100101000011010011101011110011011101111100";
74         shift_signal <= '1';
75     wait for Tclk;
76         rounding_in <= "000100100110000101111100111101001000011001100100";
77         shift_signal <= '0';
78     wait for Tclk;
79         rounding_in <= "1111111111111111111111111111111111111111111111111111111111111111";
80         shift_signal <= '1';
81     wait for Tclk;
82
83     wait;
84 end process;
85
86 end architecture sim;

```

Listing 7: TB\_ROUNDING

### a. 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 entity ROM is
12     port(
13         address : IN std_logic_vector(4 downto 0);
14         memory_out: OUT std_logic_vector(2 downto 0)
15     );
16 end entity;
17
18 -- Architettura della ROM
19 architecture ROM_rounding of ROM is
20
21     -- Spazio per segnali interni
22
23 begin
24
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";
88 elsif address = "11110" then
89      memory_out <= "111";
90 elsif address = "11111" then
91      memory_out <= "111";
92 end if;
93
94 end process;
95
96 end architecture ROM_rounding;

```

Listing 8: 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     -----
20     -- Port map
21     -----
22 DUT: entity work.ROM
23 port map (
24     address => address,
25     memory_out => memory_out
26 );
27
28 stim_proc: process
29 begin
30     for i in 0 to 31 loop
31         address <= std_logic_vector(to_unsigned(i, 5));
32         wait for 10 ns;
33     end loop;
34
35     wait;
36 end process;
37
38 end architecture sim;

```

Listing 9: TB\_ROM

## 5.6 Datapath

```

1  -- Federico Cobianchi - 332753
2  -- Onice Mazzi - 359754
3  -- Antonio Telmon - 353781
4
5
6 library IEEE;
7 use IEEE.std_logic_1164.all; -- libreria IEEE con definizione tipi standard logic
8 use IEEE.numeric_std.all;
9
10
11 entity bfly_datapath is
12 port(
13     Br_in, Bi_in, Ar_in, Ai_in, Wr_in, Wi_in : in STD_LOGIC_VECTOR (23 downto 0);
14     Clock, START, SF_2H_1L : in STD_LOGIC;
15     Br_out, Bi_out, Ar_out, Ai_out : out STD_LOGIC_VECTOR (23 downto 0);
16     DONE : out STD_LOGIC
17 );
18 end bfly_datapath;
19
20 -----
21
22 architecture structural of bfly_datapath is
23
24     -----
25     --Inizializzazione componenti
26     -----
27
28     --Multiplier
29     component BFLY_MULTIPLIER is
30         port ( A:      in      STD_LOGIC_VECTOR (23 downto 0);
31                 B:      in      STD_LOGIC_VECTOR (23 downto 0);
32                 SHIFT: in      STD_LOGIC;
33                 CK:      in      STD_LOGIC;
34                 S_OUT:  out      STD_LOGIC_VECTOR(48 downto 0);
35                 M_OUT:  out      STD_LOGIC_VECTOR(48 downto 0)

```

```

36 );
37 end component;

38
39 --Adder
40 component BFLY_ADDER is
41 port ( A:      in      STD_LOGIC_VECTOR (48 downto 0);
42        B:      in      STD_LOGIC_VECTOR (48 downto 0);
43        CK:     in      STD_LOGIC;
44        SUM_OUT:    out      STD_LOGIC_VECTOR(48 downto 0)
45      );
46 end component;

47
48 --Sottrattore
49 component BFLY_SUBTRACTOR is
50 port ( A:      in      STD_LOGIC_VECTOR (48 downto 0);
51        B:      in      STD_LOGIC_VECTOR (48 downto 0);
52        CK:     in      STD_LOGIC;
53        DIFF_OUT:   out      STD_LOGIC_VECTOR(48 downto 0)
54      );
55 end component;

56
57 --Registro FF con enable
58 component FD is
59 generic(
60           bus_length: INTEGER:= 24
61      );
62 port ( D:      in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
63        E: in STD_LOGIC;          --ENABLE attivo alto
64        CK:     in      STD_LOGIC;
65        Q:      out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
66 end component;

67
68 --Flip Flop di tipo T con reset sincrono attivo alto
69 component T_FF is
70 port ( T:      in      STD_LOGIC;
71        R: in STD_LOGIC;          --RESET attivo alto
72        CK:     in      STD_LOGIC;
73        Q:      out      STD_LOGIC);
74 end component;

75
76 --Multiplexer a tre ingressi con due bit di select
77 component MUX_3 is
78 generic(
79           bus_length: INTEGER:= 49
80      );
81 port ( A,B,C:  in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
82        S: in STD_LOGIC_VECTOR (1 downto 0);
83        Q:      out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
84 end component;

85
86 --Multiplexer a due ingressi con un bit di select
87 component MUX_2 is
88 generic(
89           bus_length: INTEGER:= 24
90      );
91 port ( A,B:  in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
92        S: in STD_LOGIC;
93        Q:      out      STD_LOGIC_VECTOR((bus_length-1) downto 0));
94 end component;

95
96 --Blocco unico di shift a destra e rom rounding
97 component rounding is
98 port(
99           Clock:  IN      STD_LOGIC; -- Clock
100          rounding_in : IN std_logic_vector(48 downto 0); -- 49 bit da arrotondare
101          rounding_out: OUT std_logic_vector(23 downto 0); -- 24 bit arrotondati

```

```

102         shift_signal: IN STD_LOGIC -- segnale per shiftare
103     );
104 end component;
105
106 --Control Unit
107 component BFLY_CU_DATAPATH is
108 port ( START:  in STD_LOGIC;
109        SF_2H_1L: in STD_LOGIC;
110        CK:      in STD_LOGIC;
111        INSTRUCTION_OUT:      out STD_LOGIC_VECTOR(16 downto 0)
112        );
113 end component;
114
115
116 -----
117 --Dichiarazione segnali datapath
118 -----
119
120 --Segnali uIR
121 SIGNAL dp_SHIFT_SIGNAL, dp_REG_IN, dp_SUM_REG, dp_AR_SEL, dp_BR_SEL, dp_WR_SEL,
122     dp_MS_DIFFp, dp_AS_SUM_SEL, dp_SD_ROUND_SEL, dp_SHIFT, dp_SF_2H_1L,
123     dp_REG_RND_BR, dp_REG_RND_BI, dp_REG_RND_AR, dp_REG_RND_AI, dp_DONE : 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 => '0');
129
130 --Ingressi al MUX di Ar/Ai
131 SIGNAL dp_Ar_MUX_in, dp_Ai_MUX_in : STD_LOGIC_VECTOR (23 downto 0) := (others => '0');
132
133 --Ingressi al MUX di Wr/Wi
134 SIGNAL dp_Wr_MUX_in, dp_Wi_MUX_in : STD_LOGIC_VECTOR (23 downto 0) := (others => '0');
135
136 --Uscite dei MUX di B, A e W
137 SIGNAL dp_B_MUX_out, dp_A_MUX_out, dp_W_MUX_out : STD_LOGIC_VECTOR (23 downto 0) := (others => '0');
138
139 --Uscite e ingressi del multiplier
140 SIGNAL dp_X_MPY_in, dp_Y_MPY_in : STD_LOGIC_VECTOR (23 downto 0) := (others => '0');
141 SIGNAL dp_MPY_product_out, dp_MPY_shift_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
142
143 --Uscita e ingressi dell'adder
144 SIGNAL dp_SUM_out, dp_X_SUM_in, dp_Y_SUM_in : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
145
146 --Uscita e ingressi del sottrattore
147 SIGNAL dp_DIFF_out, dp_X_DIFF_in, dp_Y_DIFF_in : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
148
149 --Uscita del registro di pipe della somma
150 SIGNAL dp_SUM_reg_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
151
152 --Uscita del registro di pipe del sottrattore
153 SIGNAL dp_DIFF_reg_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
154
155 --Uscita del registro di pipe del prodotto e dello shift
156 SIGNAL dp_MPY_M_reg_out, dp_MPY_S_reg_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
157
158 --Ingresso 1 del multiplexer in entrata al sommatore, ovvero l'uscita del MUX di
159     Ar/Ai con zeri aggiunti

```

```

155 SIGNAL dp_AS_A_MUX_in : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
156 --Uscita del multiplexer in entrata al sommatore
157 SIGNAL dp_AS_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
158
159 --Uscita del MUX A/B in ingresso al multiplier
160 SIGNAL dp_AB_MUX_out : STD_LOGIC_VECTOR (23 downto 0) := (others => '0');
161
162 --Uscita del MUX dell'ingresso positivo del sottrattore
163 SIGNAL dp_MS_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
164
165 --Uscita del MUX dell'ingresso negativo del sottrattore
166 SIGNAL dp_MSD_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
167
168 --Uscita del MUX dell'ingresso dello shifter a destra
169 SIGNAL dp_SD_MUX_out : STD_LOGIC_VECTOR (48 downto 0) := (others => '0');
170
171 --Uscita del blocco shift + rom rounding
172 SIGNAL dp_ROM_round_out : STD_LOGIC_VECTOR (23 downto 0) := (others => '0');
173
174 begin
175
176 -----
177 --Port map dei registri a 24 bit
178 -----
179
180 pm_regin_Br : FD
181     generic map (
182         bus_length => 24
183     )
184     port map (
185         D => Br_in,
186         E => dp_REG_IN,
187         CK => Clock,
188         Q => dp_Br_MUX_in
189 );
190
191 pm_regin_Bi : FD
192     generic map (
193         bus_length => 24
194     )
195     port map (
196         D => Bi_in,
197         E => dp_REG_IN,
198         CK => Clock,
199         Q => dp_Bi_MUX_in
200 );
201
202 pm_regin_Ar : FD
203     generic map (
204         bus_length => 24
205     )
206     port map (
207         D => Ar_in,
208         E => dp_REG_IN,
209         CK => Clock,
210         Q => dp_Ar_MUX_in
211 );
212
213 pm_regin_Ai : FD
214     generic map (
215         bus_length => 24
216     )
217     port map (
218         D => Ai_in,
219         E => dp_REG_IN,
220         CK => Clock,

```

```

221             Q => dp_Ai_MUX_in
222         );
223
224     pm_regin_Wr : FD
225         generic map (
226             bus_length => 24
227         )
228         port map (
229             D => Wr_in,
230             E => dp_REG_IN,
231             CK => Clock,
232             Q => dp_Wr_MUX_in
233         );
234
235     pm_regin_Wi : FD
236         generic map (
237             bus_length => 24
238         )
239         port map (
240             D => Wi_in,
241             E => dp_REG_IN,
242             CK => Clock,
243             Q => dp_Wi_MUX_in
244         );
245
246     -----
247     --Port map dei Multiplexer a due ingressi
248     -----
249
250     pm_mux_B : MUX_2
251     generic map (
252         bus_length => 24
253     )
254     port map (
255         A => dp_Br_MUX_in,
256         B => dp_Bi_MUX_in,
257         S => dp_BR_SEL,
258         Q => dp_B_MUX_out
259     );
260
261     pm_mux_A : MUX_2
262     generic map (
263         bus_length => 24
264     )
265     port map (
266         A => dp_Ar_MUX_in,
267         B => dp_Ai_MUX_in,
268         S => dp_AR_SEL,
269         Q => dp_A_MUX_out
270     );
271
272     pm_mux_W : MUX_2
273     generic map (
274         bus_length => 24
275     )
276     port map (
277         A => dp_Wr_MUX_in,
278         B => dp_Wi_MUX_in,
279         S => dp_WR_SEL,
280         Q => dp_W_MUX_out
281     );
282
283     pm_mux_Mult : MUX_2      --Multiplexer del multiplier
284     generic map (
285         bus_length => 24
286     )

```

```

287     port map (
288         A => dp_A_MUX_out ,
289         B => dp_B_MUX_out ,
290         S => dp_SHIFT ,
291         Q => dp_AB_MUX_out
292     );
293
294     --Ingresso 1 del multiplexer in entrata al sommatore , ovvero l'uscita del MUX di
295     --Ar/Ai
296     dp_AS_A_MUX_in (48 downto 24) <= (others => '0');           --Aggiungo zeri perche' 1,
297     --uscita del MUX di Ar/Ai e' solo su 24 bit
298     dp_AS_A_MUX_in (23 downto 0) <= dp_A_MUX_out;
299
300     pm_mux_Adder : MUX_2      --Multiplexer dell'adder
301     generic map (
302         bus_length => 49
303     )
304     port map (
305         A => dp_AS_A_MUX_in,      --l'uscita del MUX Ar/Ai
306         B => dp_SUM_reg_out,    --l'uscita del sommatore rallentata di un colpo di
307         --Clock
308         S => dp_AS_SUM_SEL ,
309         Q => dp_AS_MUX_out
310     );
311
312     pm_mux_Sub_plus : MUX_2      --Multiplexer dell'ingresso positivo del
313     --sottrattore
314     generic map (
315         bus_length => 49
316     )
317     port map (
318         A => dp_MPY_S_reg_out,   --l'uscita SHIFT del moltiplicatore
319         B => dp_SUM_reg_out,    --l'uscita del sommatore rallentata di un colpo di
320         --Clock
321         S => dp_MS_DIFFp ,
322         Q => dp_MS_MUX_out
323     );
324
325     pm_mux_rshift : MUX_2      --Multiplexer dell'ingresso allo shifter a destra
326     generic map (
327         bus_length => 49
328     )
329     port map (
330         A => dp_SUM_reg_out,      --l'uscita del sommatore
331         B => dp_DIFF_reg_out,    --l'uscita del sottrattore
332         S => dp_SD_ROUND_SEL ,
333         Q => dp_SD_MUX_out
334     );
335
336     --Port map del MUX a tre ingressi
337     pm_mux_Sub_minus : MUX_3      --Multiplexer dell'ingresso negativo del
338     --sottrattore
339     generic map (
340         bus_length => 49
341     )
342     port map (
343         A => dp_MPY_M_reg_out,    --l'uscita MPY del moltiplicatore
344         --rallentata di un colpo di Clock
345         B => dp_SUM_reg_out,      --l'uscita del sommatore
346         --rallentata di un colpo di Clock
347         C => dp_DIFF_reg_out,    --l'uscita del sottrattore
348         --rallentata di un colpo di Clock
349         S => dp_MSD_DIFFm ,
350         Q => dp_MSD_MUX_out
351     );
352
353

```

```

344 -----
345 --Port map degli operatori
346 -----
347
348     dp_X_MPY_in <= dp_AB_MUX_out;      --L'ingresso 1 del multiplier e' connesso all'
349         uscita del multiplexer A/B
350     dp_Y_MPY_in <= dp_W_MUX_out;      --L'ingresso 2 del multiplier e' connesso all'
351         uscita del multiplexer Wr/Wi
352
353 pm_Multiplier : BFLY_MULTIPLIER --Port map del multiplier
354 port map (
355     A => dp_X_MPY_in,
356     B => dp_Y_MPY_in,
357     SHIFT => dp_SHIFT,
358     CK => Clock,
359     M_OUT => dp_MPY_product_out,
360     S_OUT => dp_MPY_shift_out
361 );
362
363     dp_X_SUM_in <= dp_AS_MUX_out;          --L'ingresso 1 dell'adder e' connesso all'
364         uscita del multiplexer A/Somma
365     dp_Y_SUM_in <= dp_MPY_M_reg_out;        --L'ingresso 2 dell'adder e' connesso all',
366         uscita moltiplicazione del multiplier
367
368 pm_Adder : BFLY_ADDER    --Port map dell'adder
369 port map (
370     A => dp_X_SUM_in,
371     B => dp_Y_SUM_in,
372     CK => Clock,
373     SUM_OUT => dp_SUM_out
374 );
375
376     dp_X_DIFF_in <= dp_MS_MUX_out;
377     dp_Y_DIFF_in <= dp_MSD_MUX_out;
378
379 pm_Subractor : BFLY_SUBTRACTOR --Port map del sottrattore
380 port map (
381     A => dp_X_DIFF_in,
382     B => dp_Y_DIFF_in,
383     CK => Clock,
384     DIFF_OUT => dp_DIFF_out
385 );
386
387 pm_ft_shift : T_FF      --Port map del flip flop T che ha come uscita il segnale
388     di SF_2H_1L per il blocco rounding
389 port map (
390     T => dp_SHIFT_SIGNAL,    --Segnale che viene dalla CU
391     R => dp_DONE,           --Segnale che viene dalla CU
392     CK => Clock,
393     Q => dp_SF_2H_1L
394 );
395
396 pm_rounding : rounding   --Port map del blocco unico shifter a destra e ROM
397     rounding
398 port map (
399     Clock => Clock,
400     rounding_in => dp_SD_MUX_out,
401     rounding_out => dp_ROM_round_out,
402     shift_signal => dp_SF_2H_1L
403 );
404
405 pm_CU : BFLY CU DATAPATH      --Port map della Control unit
406 port map (
407     START => START,
408     SF_2H_1L => SF_2H_1L,
409     CK => Clock,
410

```

```

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 =====
429 --Port map dei registri a 49 bit
430 =====
431
432 pm_reg_MPY_product_out : FD          --Port map del registro all'uscita prodotto del
433   multiplier
434   generic map (
435     bus_length => 49
436   )
437   port map (
438     D => dp_MPY_product_out,
439     E => '1',
440     CK => Clock,
441     Q => dp_MPY_M_reg_out
442 );
443
444 pm_reg_MPY_shift_out : FD           --Port map del registro all'uscita shift del
445   multiplier
446   generic map (
447     bus_length => 49
448   )
449   port map (
450     D => dp_MPY_shift_out,
451     E => '1',
452     CK => Clock,
453     Q => dp_MPY_S_reg_out
454 );
455
456 pm_reg_SUM_out : FD            --Port map del registro all'uscita del sommatore
457   generic map (
458     bus_length => 49
459   )
460   port map (
461     D => dp_SUM_out,
462     E => '1',
463     CK => Clock,
464     Q => dp_SUM_reg_out
465 );
466
467 pm_reg_DIFF_out : FD        --Port map del registro all'uscita del sottrattore
468   generic map (
469     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 ,
508             E => dp_REG_RND_AR ,
509             CK => Clock,
510             Q => Ar_out
511         );
512
513     pm_regout_Ai : FD
514         generic map (
515             bus_length => 24
516         )
517         port map (
518             D => dp_ROM_round_out ,
519             E => dp_REG_RND_AI ,
520             CK => Clock,
521             Q => Ai_out
522         );
523
524 end structural;

```

Listing 10: Datapath

## 5.7 Control Unit

### a. 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

## b. 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
42     SIGNAL CC_Validation_even, CC_Validation_odd : STD_LOGIC := '0';
43
44 begin
45
46     OUT_EVEN <= out_tmp_even;
47     OUT_ODD <= out_tmp_odd;
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;
64     out_tmp_even(8) <= REG_RND_BI_even;
65     out_tmp_even(7) <= REG_RND_AR_even;
66     out_tmp_even(6) <= REG_RND_AI_even;
67     out_tmp_even(5) <= SHIFT_even;
68     out_tmp_even(4) <= DONE_even;
69
70     --Next address
71     out_tmp_even((next_Address_length) downto 1) <= next_Address_even;
72     out_tmp_even(0) <= '0';
73
74
75     --CC validation
76     out_tmp_odd(21) <= CC_Validation_odd;
77
78     --Instruction part
79     out_tmp_odd(20) <= SF_2H_1L_odd;
80     out_tmp_odd(19) <= REG_IN_odd;
81     out_tmp_odd(18) <= SUM_REG_odd;
82     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                                         -- IDLE / START
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         elsif A = "001" then                                         --M1 , SH0 / M1 , SH1
141
142             --M1 , SH0
143             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
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

### c. 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

### d. Test Bench

```

1 -- Federico Cobianchi - 332753
2 -- Onice Mazzi - 359754
3 -- Antonio Telmon - 353781
4
5
6 library IEEE;
7 use IEEE.std_logic_1164.all; -- libreria IEEE con definizione tipi standard logic
8 use IEEE.numeric_std.all;
9

```

```

10
11 entity tb_CU is
12 end tb_CU;
13
14 -----
15
16 architecture behavioral of tb_CU is
17
18     component BFLY CU DATAPATH is
19         port ( START: in STD_LOGIC;
20                 SF_2H_1L: in STD_LOGIC;
21                 CK: in STD_LOGIC;
22                 INSTRUCTION_OUT: out STD_LOGIC_VECTOR(16 downto 0)
23             );
24     end component;
25
26     constant period : time := 100 ns;
27
28     SIGNAL TB_CLK, TB_SF_2H_1L, TB_START : STD_LOGIC := '0';
29     SIGNAL TB_INSTRUCTION_OUT: STD_LOGIC_VECTOR(16 downto 0) := (others=>'0');
30
31     SIGNAL REG_IN, SUM_REG, AR_SEL, BR_SEL, WR_SEL, MS_DIFFp, AS_SUM_SEL, SD_ROUND_SEL
32         , REG_RND_BR, REG_RND_BI, REG_RND_AR, REG_RND_AI, SHIFT, DONE : STD_LOGIC :=
33         '0';
34     SIGNAL MSD_DIFFm : STD_LOGIC_VECTOR (1 downto 0) := "00";
35     SIGNAL SF_2H_1L_out : STD_LOGIC := '0';
36
37 begin
38
39     --Instruction part
40     SF_2H_1L_out <= TB_INSTRUCTION_OUT(16);
41     REG_IN <= TB_INSTRUCTION_OUT(15);
42     SUM_REG <= TB_INSTRUCTION_OUT(14);
43     AR_SEL <= TB_INSTRUCTION_OUT(13);
44     BR_SEL <= TB_INSTRUCTION_OUT(12);
45     WR_SEL <= TB_INSTRUCTION_OUT(11);
46     MS_DIFFp <= TB_INSTRUCTION_OUT(10);
47     MSD_DIFFm <= TB_INSTRUCTION_OUT(9 downto 8);
48     AS_SUM_SEL <= TB_INSTRUCTION_OUT(7);
49     SD_ROUND_SEL <= TB_INSTRUCTION_OUT(6);
50     REG_RND_BR <= TB_INSTRUCTION_OUT(5);
51     REG_RND_BI <= TB_INSTRUCTION_OUT(4);
52     REG_RND_AR <= TB_INSTRUCTION_OUT(3);
53     REG_RND_AI <= TB_INSTRUCTION_OUT(2);
54     SHIFT <= TB_INSTRUCTION_OUT(1);
55     DONE <= TB_INSTRUCTION_OUT(0);
56
57     TB_CLK <= not TB_CLK after period/2;
58
59     process
60     begin
61         wait for period*3;
62         TB_START <= '1';
63         wait for period*1;
64         TB_START <= '0';
65         wait for period*20;
66     end process;
67
68     pm_CU : BFLY CU DATAPATH port map (
69         TB_START,
70         TB_SF_2H_1L ,
71         TB_CLK ,
72         TB_INSTRUCTION_OUT
73     );

```

```
74  
75 end behavioral;
```

Listing 14: TBROM