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

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

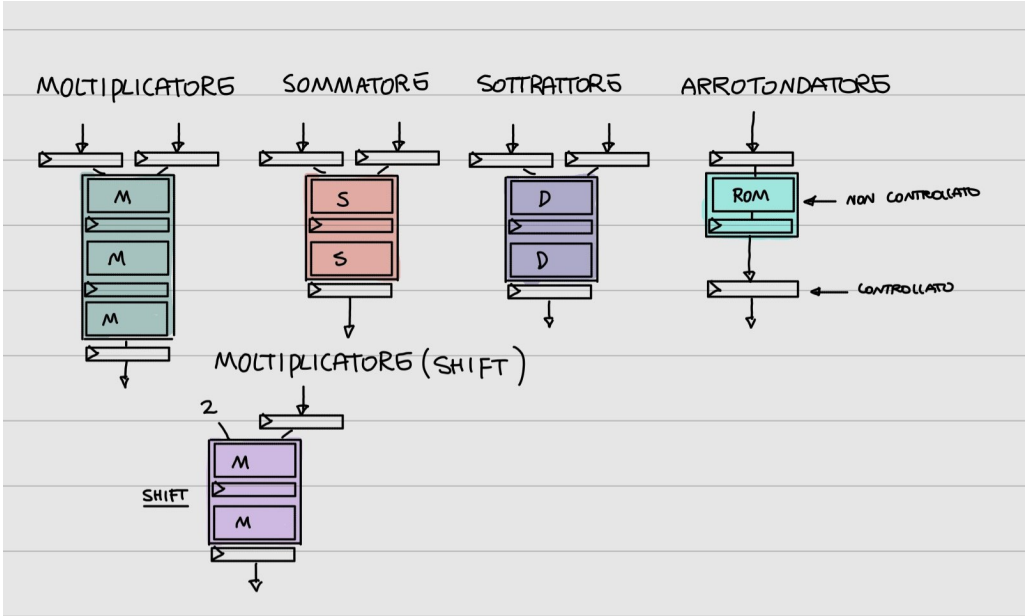

Figura 1: Blocchi elementari del data flow diagram

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.

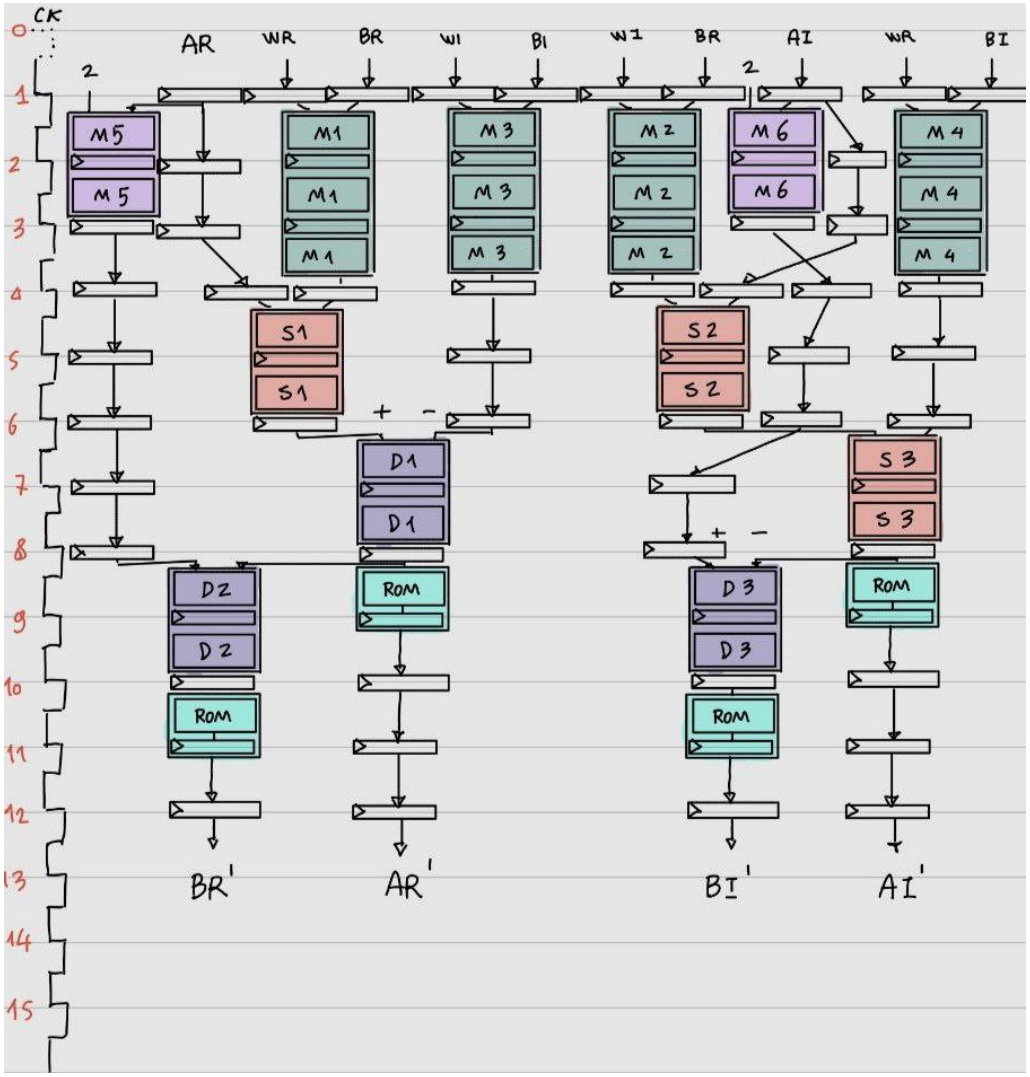

Figura 2: Data Flow Diagram con pproccio ASAP

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 operzione 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.

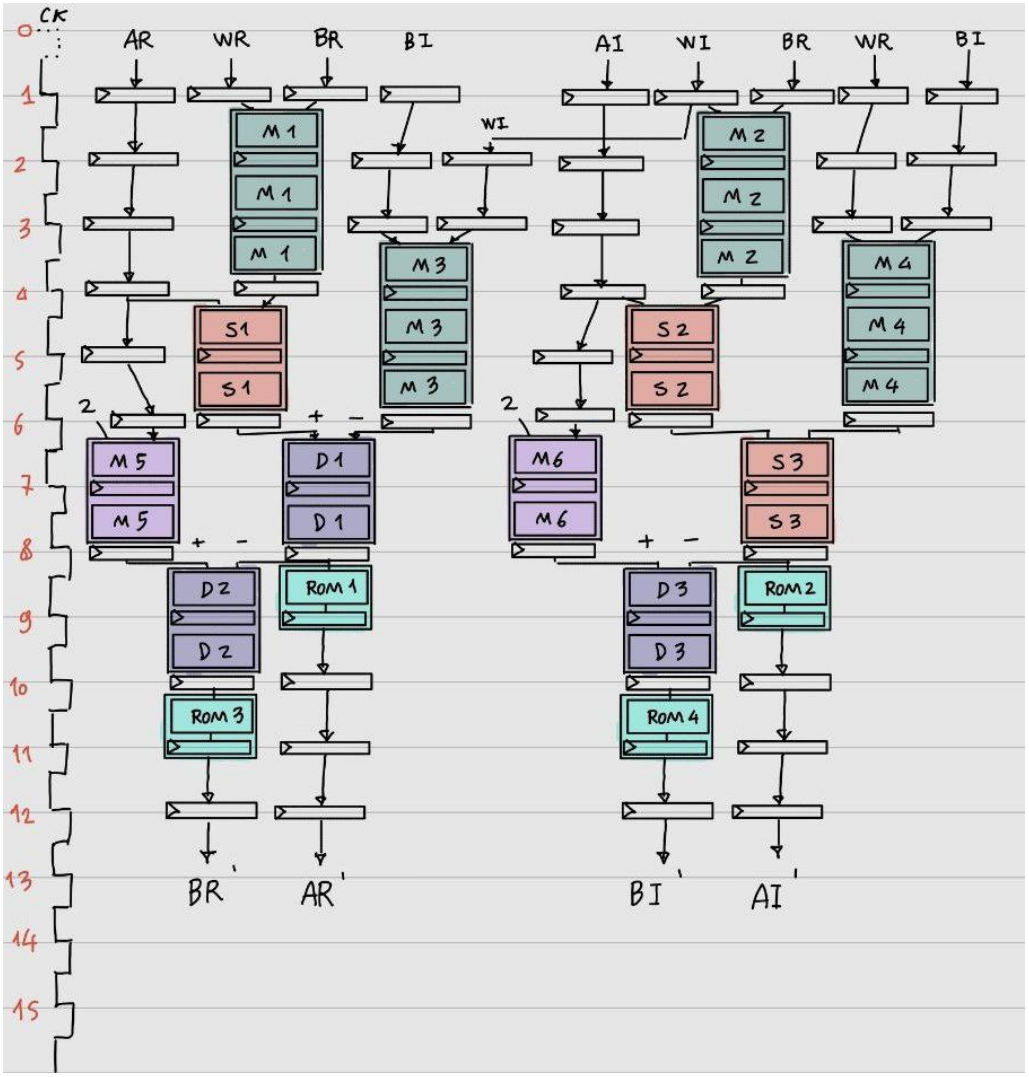

Figura 3: Data Flow Diagram con approccio ALAP

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.

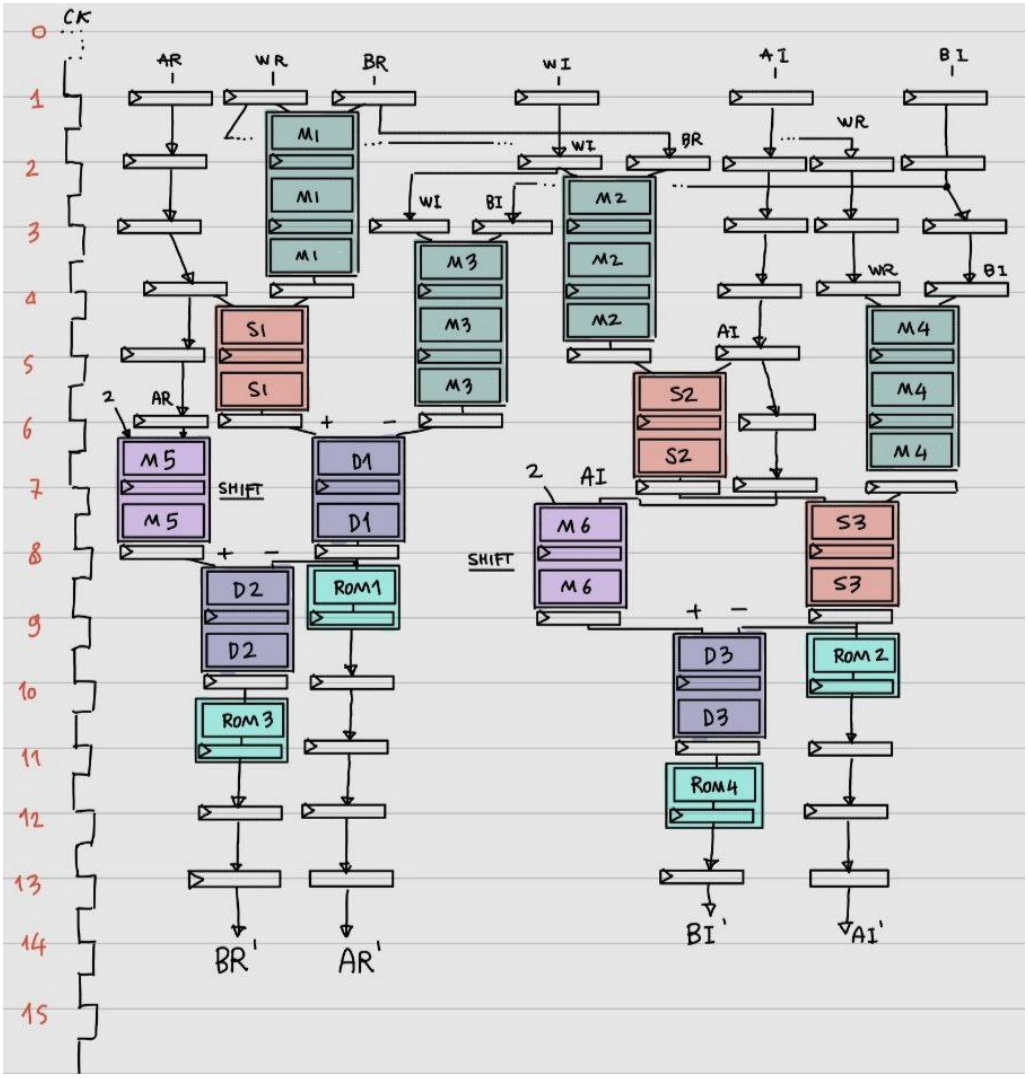

Figura 4: Data flow diagram ottimizzato

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.

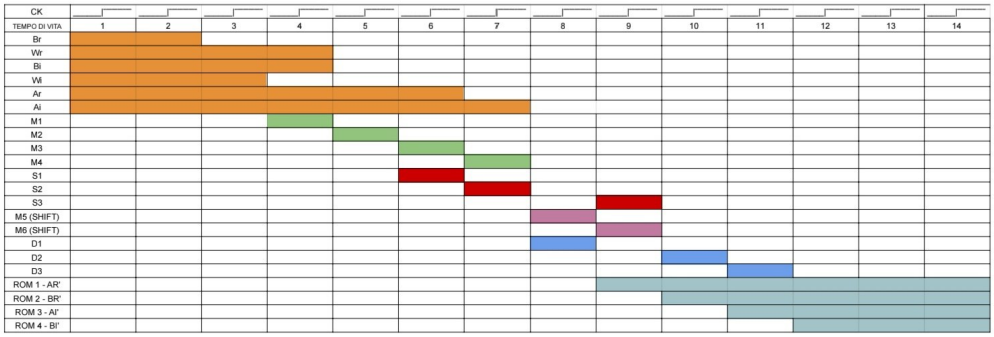

Figura 5: Tempo di vita delle variabili

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.

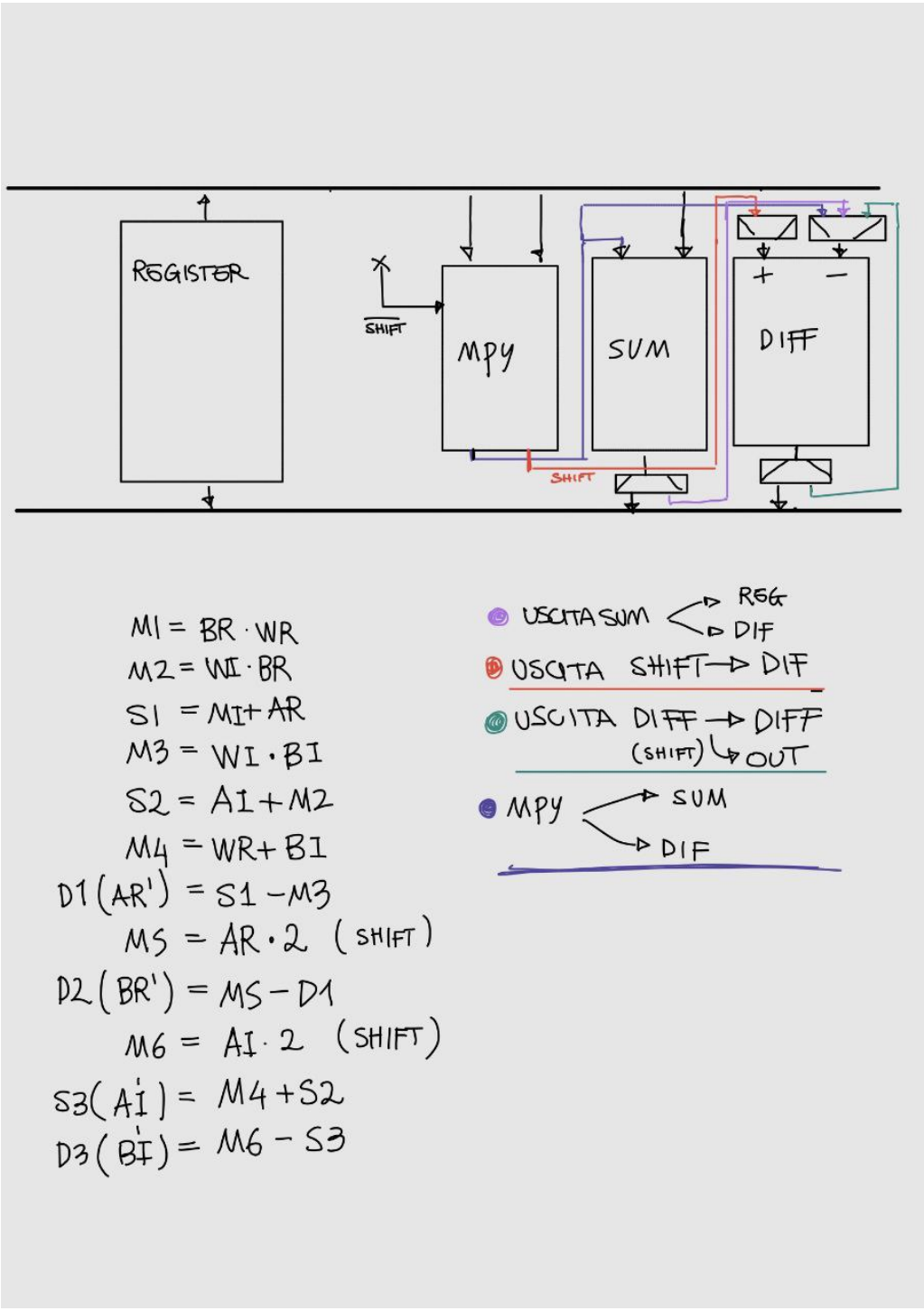

Figura 6: Schema del datapath iniziale

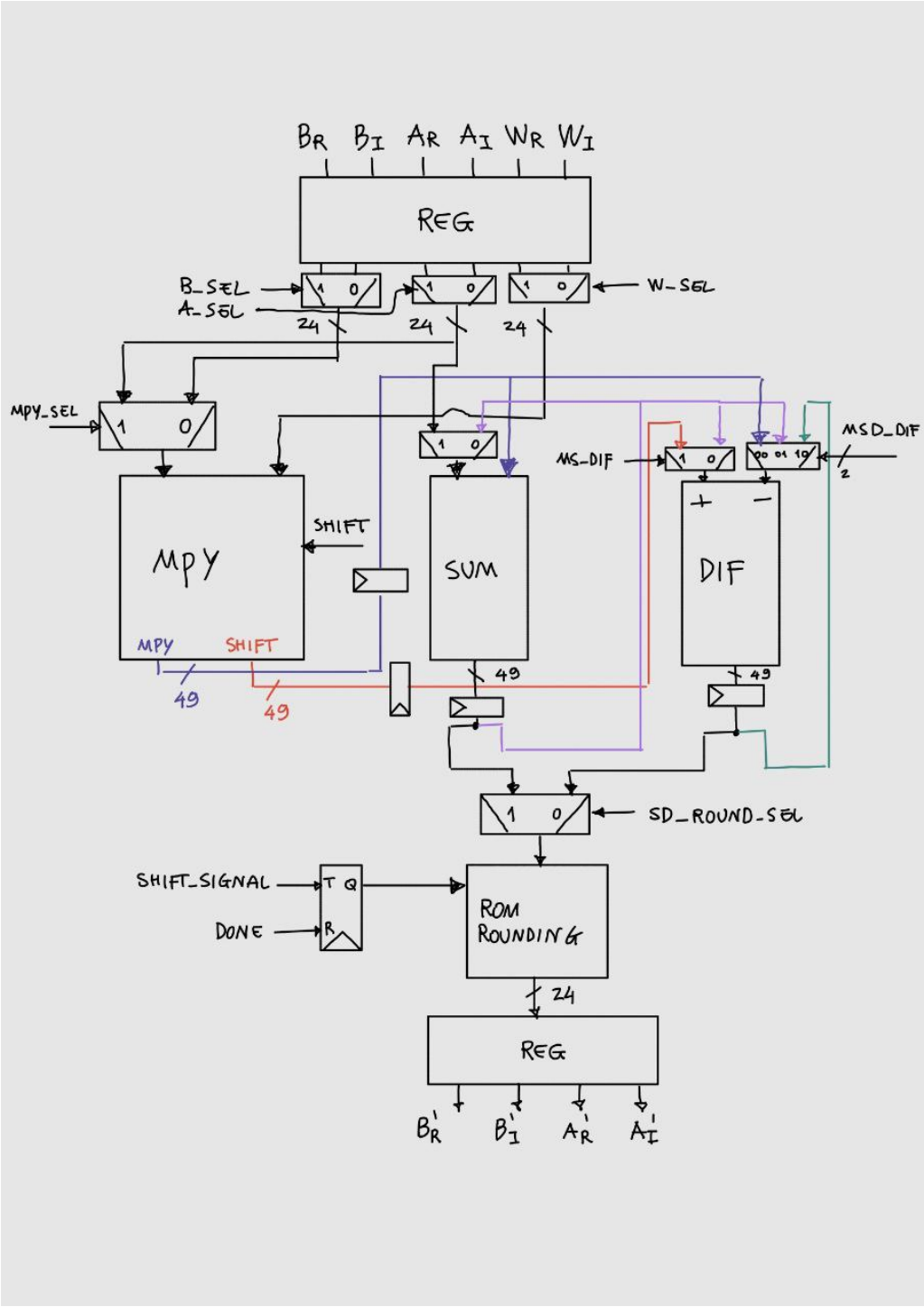

Figura 7: Schema del datapath finale

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 dell 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.

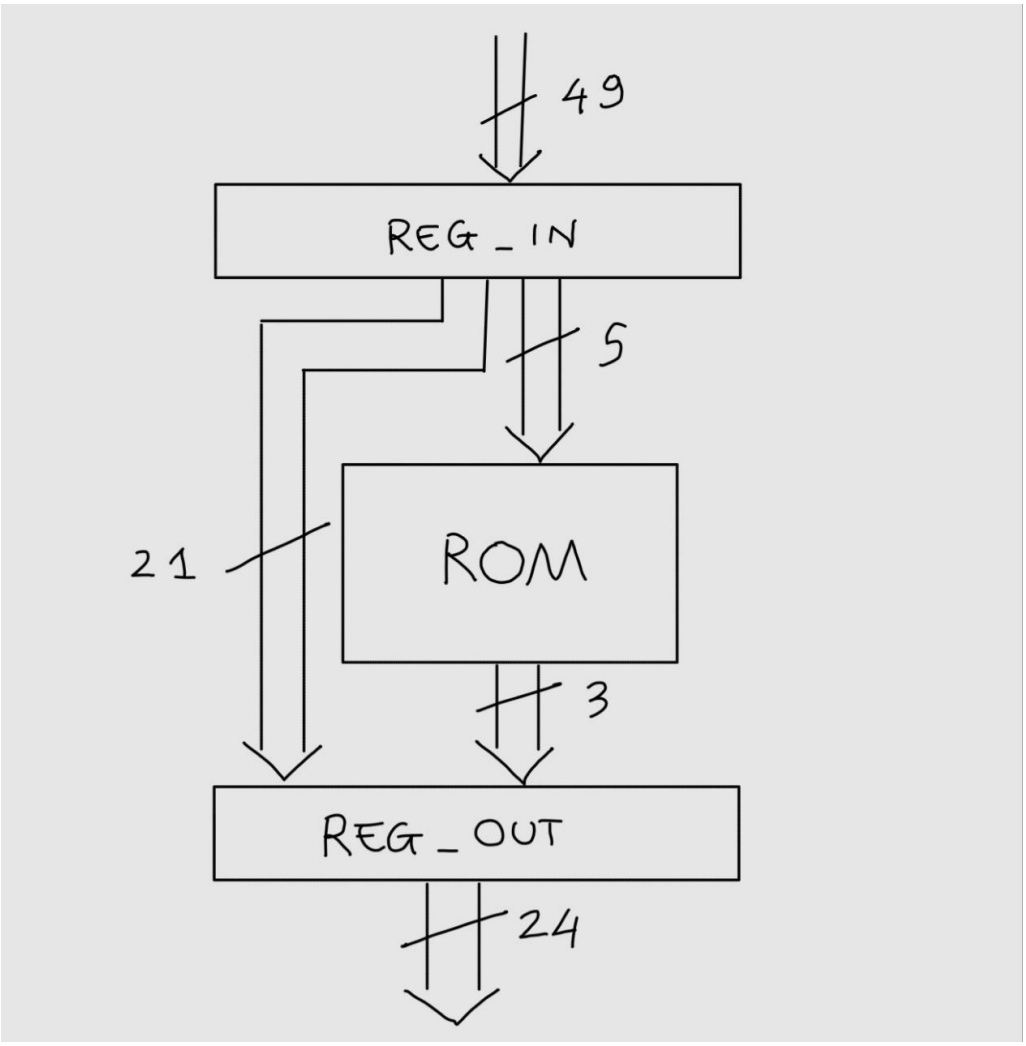

Figura 8: Schema del ROM rounding implementato

4 Control Unit

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 poichè nell'Appendice (*List. 5.7.1*) è presente del codice chiamato "control unit datapath" e si voleva fare un distinguo tra quello e il datapath vero e proprio.

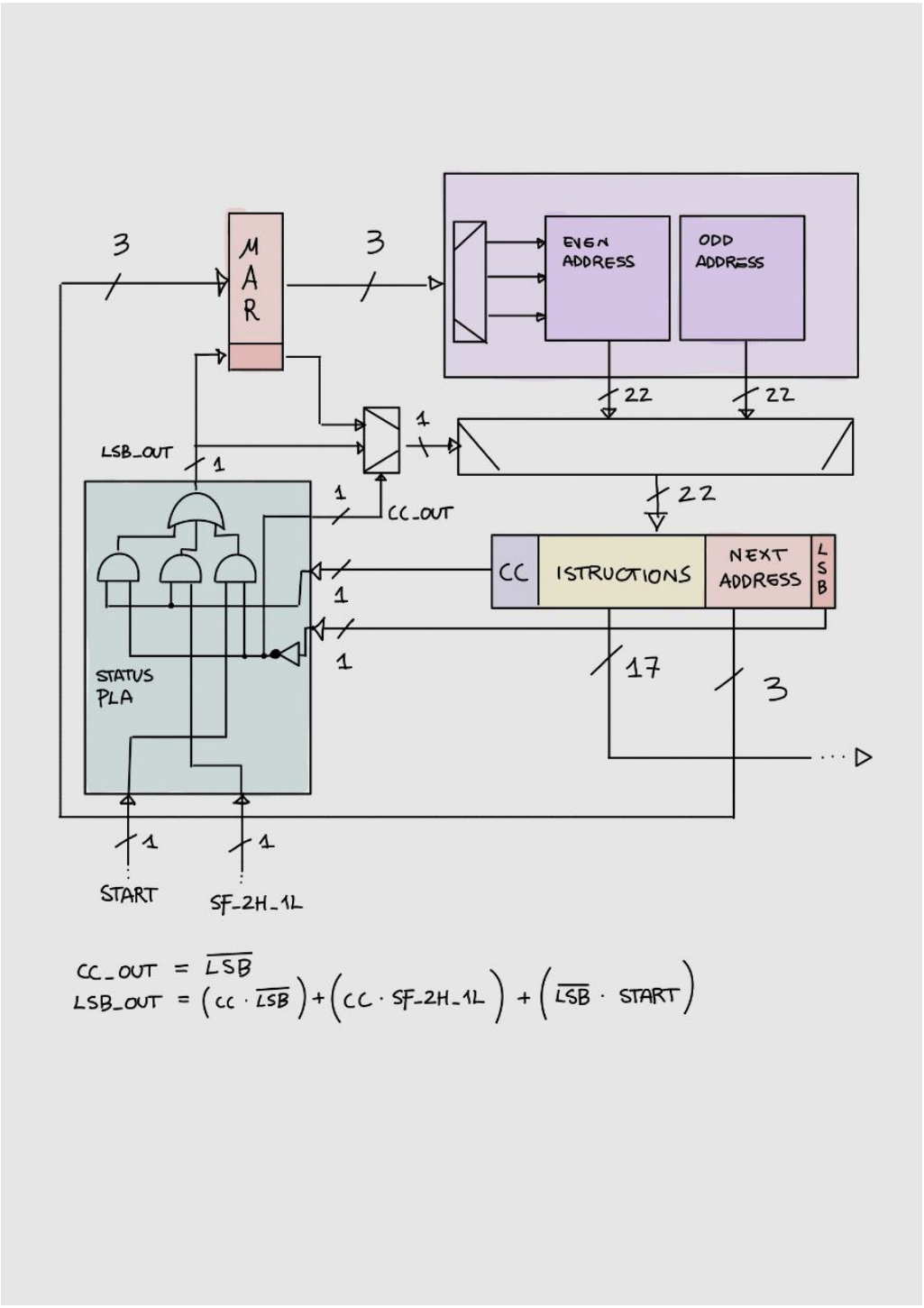

Figura 9: Schema della CU implementato

5 Appendice

5.1 Sommatore

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

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


entity BFLY_ADDER is
        port (  A:      in      STD_LOGIC_VECTOR (48 downto 0);
                B:      in      STD_LOGIC_VECTOR (48 downto 0);
                CK:     in      STD_LOGIC;
                SUM_OUT:        out     STD_LOGIC_VECTOR(48 downto 0)
                );
end BFLY_ADDER;


architecture behavioral of BFLY_ADDER is

        signal sum: STD_LOGIC_VECTOR (48 downto 0) := (others=>'0');

        begin

        sum <= std_logic_vector(signed(A)+signed(B));

                PSYNCH: process(CK)
                begin
                        if CK'event and CK='1' then -- positive edge triggered:
                                SUM_OUT <= sum;

                        end if;
                end process;

end behavioral;
```

Listing 1: Sommatore

5.2 MUX

5.2.1 MUX 2

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

library IEEE;
use IEEE.std_logic_1164.all;

entity MUX_2 is
        generic(
                bus_length: INTEGER:= 24
        );
        port (  A,B:    in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
                S: in STD_LOGIC;
                Q:      out     STD_LOGIC_VECTOR((bus_length-1) downto 0));
end MUX_2;


architecture behavioral of MUX_2 is

```

```
begin

        Q <= A when S = '1' else B;

end behavioral;
```

Listing 2: MUX 2

5.2.2 MUX 3

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

library IEEE;
use IEEE.std_logic_1164.all;

entity MUX_3 is
        generic(
                bus_length: INTEGER:= 49
        );
        port (  A,B,C:  in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
                S: in STD_LOGIC_VECTOR (1 downto 0);
                Q:      out     STD_LOGIC_VECTOR((bus_length-1) downto 0));
end MUX_3;


architecture behavioral of MUX_3 is

begin

        p_mux: process (S, A, B, C)
        begin
        case S is
                when "00" =>
                        Q <= A;
                when "01" =>
                        Q <= B;
                when "10" =>
                        Q <= C;
                when "11" =>
                        Q <= (others=>'0');
                when others =>
                        Q <= (others=>'0');
        end case;
        end process;

end behavioral;
```

Listing 3: MUX 3

5.3 Sottrattore

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

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


entity BFLY_SUBTRACTOR is
        port (  A:      in      STD_LOGIC_VECTOR (48 downto 0);
```

```
                B:      in      STD_LOGIC_VECTOR (48 downto 0);
                CK:     in      STD_LOGIC;
                DIFF_OUT:       out     STD_LOGIC_VECTOR(48 downto 0)
                );
end BFLY_SUBTRACTOR;


architecture behavioral of BFLY_SUBTRACTOR is

        signal diff: STD_LOGIC_VECTOR (48 downto 0) := (others=>'0');

        begin

        diff <= std_logic_vector(signed(A)-signed(B));

                PSYNCH: process(CK)
                begin
                        if CK'event and CK='1' then -- positive edge triggered:
                                DIFF_OUT <= diff;

                        end if;
                end process;

end behavioral;
```

Listing 4: Sottrattore

5.4 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

        op_A <= A;
        op_B <= "000000000000000000000010" 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;
```

```
37
38
39                  PSYNCH: 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');
```

```
        signal reg_in : std_logic_vector(23 downto 0):= (others=>'0');
        signal bit_scarto : std_logic_vector(1 downto 0):= (others=>'0');
        signal shift_dummy: std_logic_vector(48 downto 0) := (others=>'0');

begin

        -------------------------------------------------
        -- Port map
        -------------------------------------------------
        pm_reg_rom_out : FD
                generic map (
                        bus_length => 24
                )
                port map (
                        D => reg_in,
                        E => '1',
                        CK => Clock,
                        Q => rounding_out
        );

        pm_ROM : ROM
                port map(
                address => address_memory,
                memory_out => dummy_memory_out
        );


        -------------------------------------------------
    -- Shift senza processo logico (non impiega colpi di clock)
    -------------------------------------------------
    shift_dummy <=
        '0' & '0' & rounding_in(48 downto 2) when shift_signal = '1' else '0' &
            rounding_in(48 downto 1);

        -------------------------------------------------
        -- Creazione mantissa e bit di scarto
        -------------------------------------------------

        mantissa   <= shift_dummy(46 downto 23);
        bit_scarto <= shift_dummy(22 downto 21);

        -------------------------------------------------
        -- Creazione dell'indirizzo per leggere dalla ROM
        -------------------------------------------------

        -- address = (3 bit LSB mantissa) + (1 bit MSB scarto) + (1 bit OR con tutti gli
            altri dello scarto)
        address_memory <= mantissa(2 downto 0) & bit_scarto;

        -------------------------------------------------
        -- Inserimento dati nel registro d'uscita del blocco
        -------------------------------------------------

        reg_in <= mantissa(23 downto 3) & dummy_memory_out; -- 21 bit di mantissa & 3 bit
            arrotondamento

end architecture behavioural;
```

Listing 6: ROM rounding completo

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

library ieee;
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 <= "11001011101011101101100000100011100101100111001111";
47                 shift_signal <= '1';
48         wait for Tclk;
49         rounding_in <= "10001111101001011101000011001011001111110011111000";
50                 shift_signal <= '1';
51         wait for Tclk;
52                 rounding_in <= "00010111010110111010001011111011100111010010000000";
53                 shift_signal <= '0';
54         wait for Tclk;
55         rounding_in <= "10000100110101011010001000111001110101110001011101";
56                 shift_signal <= '1';
57         wait for Tclk;
58                 rounding_in <= "11000010111100001001100111100101000001111110110100";
59                 shift_signal <= '1';
60         wait for Tclk;
61                 rounding_in <= "01010100101001100001101010111001010110011111111011";
62                 shift_signal <= '0';
63         wait for Tclk;
64                 rounding_in <= "11011010100111101001111011011011011101111100011011";
65                 shift_signal <= '1';
66         wait for Tclk;
67                 rounding_in <= "00001100011100110001101001110010000110011111101011";
68                 shift_signal <= '0';
69         wait for Tclk;
70                 rounding_in <= "11010101001010110101010000001110011101101110010011";
71                 shift_signal <= '1';
72         wait for Tclk;
```

```
                rounding_in <= "10110010010100001101001110101111100110111011111100";
                shift_signal <= '1';
        wait for Tclk;
                rounding_in <= "00010010011000010111111001111010010000110011001000";
                shift_signal <= '0';
        wait for Tclk;
        rounding_in <= "11111111111111111111111111111111111111111111111111";
                shift_signal <= '1';
        wait for Tclk;

        wait;
    end process;

end architecture sim;
```

Listing 7: TB_ROUNDING

5.5.1 ROM

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

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


-- Creazione entity
entity ROM is
        port(
                address : IN std_logic_vector(4 downto 0);
                memory_out: OUT std_logic_vector(2 downto 0)
        );
end entity;

-- Architecture della ROM
architecture ROM_rounding of ROM is

        -- Spazio per segnali interni

begin

selection: process(address)
begin

if address = "00000" then
        memory_out <= "000";
elsif address = "00001" then
        memory_out <= "000";
elsif address = "00010" then
        memory_out <= "000";
elsif address = "00011" then
        memory_out <= "001";
elsif address = "00100" then
        memory_out <= "001";
elsif address = "00101" then
        memory_out <= "001";
elsif address = "00110" then
        memory_out <= "010";
elsif address = "00111" then
        memory_out <= "010";
elsif address = "01000" then
        memory_out <= "010";
elsif address = "01001" then
```

```
        memory_out <= "010";
elsif address = "01010" then
        memory_out <= "010";
elsif address = "01011" then
        memory_out <= "011";
elsif address = "01100" then
        memory_out <= "011";
elsif address = "01101" then
        memory_out <= "011";
elsif address = "01110" then
        memory_out <= "100";
elsif address = "01111" then
        memory_out <= "100";
elsif address = "10000" then
        memory_out <= "100";
elsif address = "10001" then
        memory_out <= "100";
elsif address = "10010" then
        memory_out <= "100";
elsif address = "10011" then
        memory_out <= "101";
elsif address = "10100" then
        memory_out <= "101";
elsif address = "10101" then
        memory_out <= "101";
elsif address = "10110" then
        memory_out <= "110";
elsif address = "10111" then
        memory_out <= "110";
elsif address = "11000" then
        memory_out <= "110";
elsif address = "11001" then
        memory_out <= "110";
elsif address = "11010" then
        memory_out <= "110";
elsif address = "11011" then
        memory_out <= "111";
elsif address = "11100" then
        memory_out <= "111";
elsif address = "11101" then
        memory_out <= "111";
elsif address = "11110" then
        memory_out <= "111";
elsif address = "11111" then
        memory_out <= "111";
end if;

end process;

end architecture ROM_rounding;
```

Listing 8: ROM

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

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

entity TBROM is
end entity;

architecture sim of TBROM is

```

```
    signal address  : std_logic_vector(4 downto 0);
    signal memory_out : std_logic_vector(2 downto 0);

begin

        -------------------------------------------------
        -- Port map
        -------------------------------------------------
    DUT: entity work.ROM
        port map (
            address  => address,
            memory_out => memory_out
        );

    stim_proc: process
    begin
        for i in 0 to 31 loop
            address <= std_logic_vector(to_unsigned(i, 5));
            wait for 10 ns;
        end loop;

        wait;
    end process;

end architecture sim;
```

Listing 9: TB_ROM

5.6 Datapath

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


library IEEE;
use IEEE.std_logic_1164.all; --  libreria IEEE con definizione tipi standard logic
use IEEE.numeric_std.all;


entity bfly_datapath is
port(
        Br_in, Bi_in, Ar_in, Ai_in, Wr_in, Wi_in : in STD_LOGIC_VECTOR (23 downto 0);
        Clock, START, SF_2H_1L : in STD_LOGIC;
        Br_out, Bi_out, Ar_out, Ai_out : out STD_LOGIC_VECTOR (23 downto 0);
        DONE : out STD_LOGIC
);
end     bfly_datapath;

-------------------------------------------

architecture structural of bfly_datapath is

        -------------------------------------------------
        --Inizializzazione componenti
        -------------------------------------------------

        --Multiplier
        component 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)
```

```
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
```

```
            shift_signal: IN STD_LOGIC -- segnale per shiftare
        );
        end component;

        --Control Unit
        component BFLY_CU_DATAPATH is
        port (  START:  in      STD_LOGIC;
                SF_2H_1L: in STD_LOGIC;
                CK:     in      STD_LOGIC;
                INSTRUCTION_OUT:        out     STD_LOGIC_VECTOR(16 downto 0)
                );
        end component;


        ------------------------------------------------
        --Dichiarazione segnali datapath
        ------------------------------------------------

        --Segnali uIR
        SIGNAL dp_SHIFT_SIGNAL, dp_REG_IN, dp_SUM_REG, dp_AR_SEL, dp_BR_SEL, dp_WR_SEL,
            dp_MS_DIFFp, dp_AS_SUM_SEL, dp_SD_ROUND_SEL, dp_SHIFT, dp_SF_2H_1L,
            dp_REG_RND_BR, dp_REG_RND_BI, dp_REG_RND_AR, dp_REG_RND_AI, dp_DONE :
            STD_LOGIC := '0';
        SIGNAL dp_MSD_DIFFm     : STD_LOGIC_VECTOR (1 downto 0) := (others => '0');
        SIGNAL dp_INSTRUCTION_OUT : STD_LOGIC_VECTOR (16 downto 0) := (others => '0');

        --Ingressi al MUX di Br/Bi
        SIGNAL dp_Br_MUX_in, dp_Bi_MUX_in : STD_LOGIC_VECTOR (23 downto 0) := (others =>
            '0');
        --Ingressi al MUX di Ar/Ai
        SIGNAL dp_Ar_MUX_in, dp_Ai_MUX_in : STD_LOGIC_VECTOR (23 downto 0) := (others =>
            '0');
        --Ingressi al MUX di Wr/Wi
        SIGNAL dp_Wr_MUX_in, dp_Wi_MUX_in : STD_LOGIC_VECTOR (23 downto 0) := (others =>
            '0');

        --Uscite dei MUX di B, A e W
        SIGNAL dp_B_MUX_out, dp_A_MUX_out, dp_W_MUX_out : STD_LOGIC_VECTOR (23 downto 0)
            := (others => '0');

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

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

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

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

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

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

        --Ingresso 1 del multiplexer in entrata al sommatore, ovvero l'uscita del MUX di
            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,
```

```
                Q => dp_Ai_MUX_in
        );

        pm_regin_Wr : FD
                generic map (
                        bus_length => 24
                )
                port map (
                        D => Wr_in,
                        E => dp_REG_IN,
                        CK => Clock,
                        Q => dp_Wr_MUX_in
        );

        pm_regin_Wi : FD
                generic map (
                        bus_length => 24
                )
                port map (
                        D => Wi_in,
                        E => dp_REG_IN,
                        CK => Clock,
                        Q => dp_Wi_MUX_in
        );

        ------------------------------------------------
        --Port map dei Multiplexer a due ingressi
        ------------------------------------------------

        pm_mux_B : MUX_2
        generic map (
                bus_length => 24
        )
        port map (
                A => dp_Br_MUX_in,
                B => dp_Bi_MUX_in,
                S => dp_BR_SEL,
                Q => dp_B_MUX_out
        );

        pm_mux_A : MUX_2
        generic map (
                bus_length => 24
        )
        port map (
                A => dp_Ar_MUX_in,
                B => dp_Ai_MUX_in,
                S => dp_AR_SEL,
                Q => dp_A_MUX_out
        );

        pm_mux_W : MUX_2
        generic map (
                bus_length => 24
        )
        port map (
                A => dp_Wr_MUX_in,
                B => dp_Wi_MUX_in,
                S => dp_WR_SEL,
                Q => dp_W_MUX_out
        );

        pm_mux_Mult : MUX_2     --Multiplexer del multiplier
        generic map (
                bus_length => 24
        )
```

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

```
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'
                uscita del multiplexer A/B
349         dp_Y_MPY_in <= dp_W_MUX_out;    --L'ingresso 2 del multiplier e' connesso all'
                uscita del multiplexer Wr/Wi
350
351         pm_Multiplier : BFLY_MULTIPLIER --Port map del multiplier
352         port map (
353                 A => dp_X_MPY_in,
354                 B => dp_Y_MPY_in,
355                 SHIFT => dp_SHIFT,
356                 CK => Clock,
357                 M_OUT => dp_MPY_product_out,
358                 S_OUT => dp_MPY_shift_out
359         );
360
361         dp_X_SUM_in <= dp_AS_MUX_out;           --L'ingresso 1 dell'adder e' connesso all'
                uscita del multiplexer A/Somma
362         dp_Y_SUM_in <= dp_MPY_M_reg_out;        --L'ingresso 2 dell'adder e' connesso all'
                uscita moltiplicazione del multiplier
363
364         pm_Adder : BFLY_ADDER   --Port map dell'adder
365         port map (
366                 A => dp_X_SUM_in,
367                 B => dp_Y_SUM_in,
368                 CK => Clock,
369                 SUM_OUT => dp_SUM_out
370         );
371
372         dp_X_DIFF_in <= dp_MS_MUX_out;
373         dp_Y_DIFF_in <= dp_MSD_MUX_out;
374
375         pm_Subractor : BFLY_SUBTRACTOR  --Port map del sottrattore
376         port map (
377                 A => dp_X_DIFF_in,
378                 B => dp_Y_DIFF_in,
379                 CK => Clock,
380                 DIFF_OUT => dp_DIFF_out
381         );
382
383         pm_ft_shift : T_FF      --Port map del flip flop T che ha come uscita il segnale
                di SF_2H_1L per il blocco rounding
384         port map (
385                 T => dp_SHIFT_SIGNAL,   --Segnale che viene dalla CU
386                 R => dp_DONE,                   --Segnale che viene dalla CU
387                 CK => Clock,
388                 Q => dp_SF_2H_1L
389         );
390
391         pm_rounding : rounding  --Port map del blocco unico shifter a destra e ROM
                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         ------------------------------------------------
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
                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
                multiplier
444             generic map (
445                     bus_length => 49
446             )
447             port map (
448                     D => dp_MPY_shift_out,
449                     E => '1',
450                     CK => Clock,
451                     Q => dp_MPY_S_reg_out
452         );
453
454         pm_reg_SUM_out : FD     --Port map del registro all'uscita del sommatore
455             generic map (
456                     bus_length => 49
457             )
458             port map (
459                     D => dp_SUM_out,
460                     E => '1',
461                     CK => Clock,
462                     Q => dp_SUM_reg_out
463         );
464
465         pm_reg_DIFF_out : FD    --Port map del registro all'uscita del sottrattore
466             generic map (
467                     bus_length => 49
```

```
468                 )
469                 port map (
470                         D => dp_DIFF_out,
471                         E => '1',
472                         CK => Clock,
473                         Q => dp_DIFF_reg_out
474         );
475 
476         ------------------------------------------------
477         --Port map dei registri di uscita a 24 bit
478         ------------------------------------------------
479 
480         pm_regout_Br : FD
481                 generic map (
482                         bus_length => 24
483                 )
484                 port map (
485                         D => dp_ROM_round_out,
486                         E => dp_REG_RND_BR,
487                         CK => Clock,
488                         Q => Br_out
489         );
490 
491         pm_regout_Bi : FD
492                 generic map (
493                         bus_length => 24
494                 )
495                 port map (
496                         D => dp_ROM_round_out,
497                         E => dp_REG_RND_BI,
498                         CK => Clock,
499                         Q => Bi_out
500         );
501 
502         pm_regout_Ar : FD
503                 generic map (
504                         bus_length => 24
505                 )
506                 port map (
507                         D => dp_ROM_round_out,
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

5.7.1 Datapath

```
1 -- Federico Cobianchi - 332753
```

```
-- Onice Mazzi - 359754
-- Antonio Telmon - 353781

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


entity BFLY_CU_DATAPATH is
        port (  START:  in      STD_LOGIC;
                SF_2H_1L: in STD_LOGIC;
                CK:     in      STD_LOGIC;
                INSTRUCTION_OUT:        out     STD_LOGIC_VECTOR(16 downto 0)
                );
end BFLY_CU_DATAPATH;


architecture structural of BFLY_CU_DATAPATH is

        component BFLY_CU_LATE_STATUS_PLA is
        port (  STATUS: in STD_LOGIC_VECTOR(1 downto 0);
                LSB_in: in STD_LOGIC;
                CC_Validation_in: in STD_LOGIC;
                CC_Validation_out: out STD_LOGIC;
                LSB_out: out STD_LOGIC
                );
        end component;

        component BFLY_CU_ROM is
        generic(
                in_length: INTEGER:= 3;
                next_Address_length :INTEGER := 4;
                out_length: INTEGER:= 22
        );
        port (  A:      in      STD_LOGIC_VECTOR ((in_length-1) downto 0);
                OUT_EVEN:       out     STD_LOGIC_VECTOR((out_length-1) downto 0);
                OUT_ODD:        out     STD_LOGIC_VECTOR((out_length-1) downto 0)
                );
        end component;

        component FD is
        generic(
                bus_length: INTEGER:= 24
        );
        port (  D:      in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
                E: in STD_LOGIC;        --ENABLE attivo alto
                CK:     in      STD_LOGIC;
                Q:      out     STD_LOGIC_VECTOR((bus_length-1) downto 0));
        end component;

        component MUX_2 is
        generic(
                bus_length: INTEGER:= 24
        );
        port (  A,B:    in      STD_LOGIC_VECTOR ((bus_length-1) downto 0);
                S: in STD_LOGIC;
                Q:      out     STD_LOGIC_VECTOR((bus_length-1) downto 0));
        end component;

        SIGNAL microAR_in_MSB : STD_LOGIC_VECTOR (3 downto 1) := (others=>'0');
        SIGNAL microAR_out_MSB : STD_LOGIC_VECTOR (3 downto 1)  := (others=>'0');
        SIGNAL microAR_in_LSB : STD_LOGIC := '0';
        SIGNAL microAR_out_LSB : STD_LOGIC := '0';

        SIGNAL CC_mux_out : STD_LOGIC := '0';

```

```
68          SIGNAL PLA_ROM_out_even, PLA_ROM_out_odd : STD_LOGIC_VECTOR (21 downto 0):= (
                others=>'0');
69
70          SIGNAL PLA_ROM_mux_out : STD_LOGIC_VECTOR (21 downto 0):= (others=>'0');
71
72          SIGNAL status_PLA_LSB_out : STD_LOGIC := '0';
73          SIGNAL status_PLA_CC_validation_out : STD_LOGIC := '0';
74
75          SIGNAL microIR_in : STD_LOGIC_VECTOR (21 downto 0)      := (others=>'0');
76          SIGNAL microIR_out : STD_LOGIC_VECTOR (21 downto 0)     := (others=>'0');
77
78          SIGNAL CC_validation : STD_LOGIC := '0';
79          SIGNAL next_Address_LSB : STD_LOGIC := '0';
80          SIGNAL next_Address_MSB : STD_LOGIC_VECTOR (2 downto 0) := (others=>'0');
81
82          SIGNAL dp_STATUS : STD_LOGIC_VECTOR (1 downto 0) := (others=>'0');
83
84          begin
85
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
```

```
        );

        --Registro dell'LSB del uAR
        FF_D_uAR: process(CK)
        begin
                if CK'event and CK='1' then -- positive edge triggered:
                        microAR_out_LSB <= microAR_in_LSB;
                end if;
        end process;

        --Registro del uIR
        pm_microIR_reg : FD
        generic map (
                bus_length => 22
        )
        port map (
                D => microIR_in,
                E => '1',
                CK => CK,
                Q => microIR_out
        );

        --MUX a due ingressi a 21 bit, che seleziona tra l'uscita pari o dispari della ROM
        pm_ROM_mux : MUX_2
        generic map (
                bus_length => 22
        )
        port map (
                A => PLA_ROM_out_odd,
                B => PLA_ROM_out_even,
                S => CC_mux_out,
                Q => PLA_ROM_mux_out
        );

        --MUX a due ingressi a 1 bit
        --L'uscita e' il segnale di select per il MUX even/odd della ROM
        CC_mux_out <= microAR_out_LSB when status_PLA_CC_validation_out = '0' else
            status_PLA_LSB_out;


end structural;
```

Listing 11: Control Unit Datapath

5.7.2 ROM

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

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


entity BFLY_CU_ROM is
        generic(
                in_length: INTEGER:= 3;
                next_Address_length :INTEGER := 3;
                out_length: INTEGER:= 22
        );
        port (  A:      in      STD_LOGIC_VECTOR ((in_length-1) downto 0);
                OUT_EVEN:       out     STD_LOGIC_VECTOR((out_length-1) downto 0);
                OUT_ODD:        out     STD_LOGIC_VECTOR((out_length-1) downto 0)
                );
```

```
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) := (
               others=>'0');
26         SIGNAL next_Address_even, next_Address_odd : STD_LOGIC_VECTOR ((
               next_Address_length-1) downto 0) := (others=>'0');
27
28         SIGNAL REG_IN_even, SUM_REG_even, AR_SEL_even, BR_SEL_even, WR_SEL_even,
               MS_DIFFp_even, AS_SUM_SEL_even, SD_ROUND_SEL_even, REG_RND_BR_even,
               REG_RND_BI_even, REG_RND_AR_even, REG_RND_AI_even, SHIFT_even, DONE_even :
               STD_LOGIC := '0';
29         SIGNAL REG_IN_odd, SUM_REG_odd, AR_SEL_odd, BR_SEL_odd, WR_SEL_odd, MS_DIFFp_odd,
               AS_SUM_SEL_odd, SD_ROUND_SEL_odd, REG_RND_BR_odd, REG_RND_BI_odd,
               REG_RND_AR_odd, REG_RND_AI_odd, SHIFT_odd, DONE_odd : STD_LOGIC := '0';
30         SIGNAL SF_2H_1L_even, SF_2H_1L_odd : STD_LOGIC := '0';
31
32
33         SIGNAL MSD_DIFFm_even, MSD_DIFFm_odd : STD_LOGIC_VECTOR (1 downto 0) := "00";
34
35         SIGNAL CC_Validation_even, CC_Validation_odd : STD_LOGIC := '0';
36
37         begin
38
39         OUT_EVEN <= out_tmp_even;
40         OUT_ODD <= out_tmp_odd;
41
42
43         --CC validation
44         out_tmp_even(21) <= CC_Validation_even;
45
46         --Instruction part
47         out_tmp_even(20) <= SF_2H_1L_even;
48         out_tmp_even(19) <= REG_IN_even;
49         out_tmp_even(18) <= SUM_REG_even;
50         out_tmp_even(17) <= AR_SEL_even;
51         out_tmp_even(16) <= BR_SEL_even;
52         out_tmp_even(15) <= WR_SEL_even;
53         out_tmp_even(14) <= MS_DIFFp_even;
54         out_tmp_even(13 downto 12) <= MSD_DIFFm_even;
55         out_tmp_even(11) <= AS_SUM_SEL_even;
56         out_tmp_even(10) <= SD_ROUND_SEL_even;
57         out_tmp_even(9) <= REG_RND_BR_even;
58         out_tmp_even(8) <= REG_RND_BI_even;
59         out_tmp_even(7) <= REG_RND_AR_even;
60         out_tmp_even(6) <= REG_RND_AI_even;
61         out_tmp_even(5) <= SHIFT_even;
62         out_tmp_even(4) <= DONE_even;
63
64         --Next address
65         out_tmp_even((next_Address_length) downto 1) <= next_Address_even;
66         out_tmp_even(0) <= '0';
67
68
69
70
71         --CC validation
72         out_tmp_odd(21) <= CC_Validation_odd;
73
74         --Instruction part
75         out_tmp_odd(20) <= SF_2H_1L_odd;
76         out_tmp_odd(19) <= REG_IN_odd;
77         out_tmp_odd(18) <= SUM_REG_odd;
78         out_tmp_odd(17) <= AR_SEL_odd;
```

```
79          out_tmp_odd(16) <= BR_SEL_odd;
80          out_tmp_odd(15) <= WR_SEL_odd;
81          out_tmp_odd(14) <= MS_DIFFp_odd;
82          out_tmp_odd(13 downto 12) <= MSD_DIFFm_odd;
83          out_tmp_odd(11) <= AS_SUM_SEL_odd;
84          out_tmp_odd(10) <= SD_ROUND_SEL_odd;
85          out_tmp_odd(9) <= REG_RND_BR_odd;
86          out_tmp_odd(8) <= REG_RND_BI_odd;
87          out_tmp_odd(7) <= REG_RND_AR_odd;
88          out_tmp_odd(6) <= REG_RND_AI_odd;
89          out_tmp_odd(5) <= SHIFT_odd;
90          out_tmp_odd(4) <= DONE_odd;
91
92          --Next address
93          out_tmp_odd((next_Address_length) downto 1) <= next_Address_odd;
94          out_tmp_odd(0) <= '1';
95
96          p_rom : process (A)
97          begin
98                  if A = "000" then                                       --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
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';
```

```
                        AR_SEL_odd <= '0';
                        BR_SEL_odd <= '0';
                        WR_SEL_odd <= '0';
                        MS_DIFFp_odd <= '0';
                        MSD_DIFFm_odd <= "00";
                        AS_SUM_SEL_odd <= '0';
                        SD_ROUND_SEL_odd <= '0';
                        REG_RND_BR_odd <= '0';
                        REG_RND_BI_odd <= '0';
                        REG_RND_AR_odd <= '0';
                        REG_RND_AI_odd <= '0';
                        SHIFT_odd <= '0';
                        DONE_odd <= '0';
                        next_Address_odd <= "011";


                elsif A = "011" then                                    --M4,S1 / S2

                        --M4, S1
                        SF_2H_1L_even <= '0';
                        CC_Validation_even <= '1';
                        REG_IN_even <= '0';
                        SUM_REG_even <= '0';
                        AR_SEL_even <= '1';
                        BR_SEL_even <= '0';
                        WR_SEL_even <= '1';
                        MS_DIFFp_even <= '0';
                        MSD_DIFFm_even <= "00";
                        AS_SUM_SEL_even <= '1';
                        SD_ROUND_SEL_even <= '0';
                        REG_RND_BR_even <= '0';
                        REG_RND_BI_even <= '0';
                        REG_RND_AR_even <= '0';
                        REG_RND_AI_even <= '0';
                        SHIFT_even <= '0';
                        DONE_even <= '0';
                        next_Address_even <= "011";

                        --S2
                        SF_2H_1L_odd <= '0';
                        CC_Validation_odd <= '0';
                        REG_IN_odd <= '0';
                        SUM_REG_odd <= '0';
                        AR_SEL_odd <= '0';
                        BR_SEL_odd <= '0';
                        WR_SEL_odd <= '0';
                        MS_DIFFp_odd <= '0';
                        MSD_DIFFm_odd <= "00";
                        AS_SUM_SEL_odd <= '1';
                        SD_ROUND_SEL_odd <= '0';
                        REG_RND_BR_odd <= '0';
                        REG_RND_BI_odd <= '0';
                        REG_RND_AR_odd <= '0';
                        REG_RND_AI_odd <= '0';
                        SHIFT_odd <= '0';
                        DONE_odd <= '0';
                        next_Address_odd <= "100";


                elsif A = "100" then                                    --M5,D1 / M6,S3

                        --M5, D1
                        SF_2H_1L_even <= '0';
                        CC_Validation_even <= '1';
                        REG_IN_even <= '0';
                        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';
```

```
                        REG_RND_BI_odd <= '0';
                        REG_RND_AR_odd <= '0';
                        REG_RND_AI_odd <= '0';
                        SHIFT_odd <= '0';
                        DONE_odd <= '0';
                        next_Address_odd <= "000";

        end if;
        end process;


end behavioral;
```

Listing 12: Control Unit ROM

5.7.3 PLA

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

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


entity BFLY_CU_LATE_STATUS_PLA is
        port (  STATUS: in STD_LOGIC_VECTOR(1 downto 0);
                LSB_in: in STD_LOGIC;
                CC_Validation_in: in STD_LOGIC;
                CC_Validation_out: out STD_LOGIC;
                LSB_out: out STD_LOGIC
                );
end BFLY_CU_LATE_STATUS_PLA;


architecture behavioral of BFLY_CU_LATE_STATUS_PLA is

        SIGNAL START, SF_2H_1L : STD_LOGIC := '0';

        begin

                START <= STATUS(0);
                SF_2H_1L <= STATUS(1);

                LSB_out <= (CC_Validation_in AND (NOT LSB_in)) OR (CC_Validation_in AND
                    SF_2H_1L) OR ((NOT LSB_in) AND START);
                CC_Validation_out <= NOT(LSB_in);

end behavioral;
```

Listing 13: Control Unit PLA

5.7.4 Test Bench

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


library IEEE;
use IEEE.std_logic_1164.all; --  libreria IEEE con definizione tipi standard logic
use IEEE.numeric_std.all;

```

```
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
                , 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);
54
55
56          TB_CLK <= not TB_CLK after period/2;
57
58          process
59          begin
60                  wait for period*3;
61                  TB_START <= '1';
62                  wait for period*1;
63                  TB_START <= '0';
64                  wait for period*20;
65          end process;
66
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: TBCU