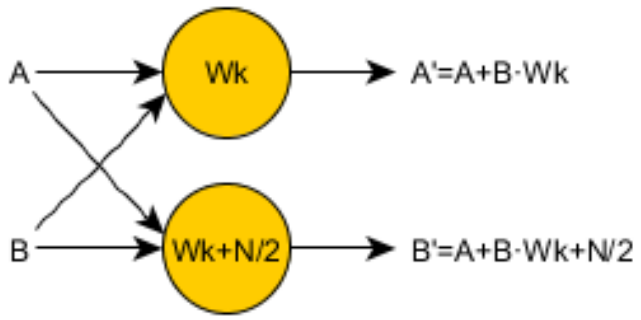
Progetto “Butterfly”

Autori:

Fabio Grassi, Alessandro Nadal, Stefano Iuliano.

1.0 Introduzione:

Il progetto “Butterfly” ha come scopo la realizzazione di un Processing Element in grado di svolgere l’operazione base della Fast Fourier Transformer. Di seguito è mostrata la sua architettura:



Come si può notare la Butterfly opera su due ingressi A e B fornendo in uscita A’ e B’ attraverso l’ausilio di due coefficienti W e W’. Nel caso della FFT tutti i dati utilizzati sono numeri complessi.

2.0 Algoritmo di calcolo

I dati utilizzati nell’algoritmo sono:

- Ingressi: $A = A_r + jA_i, B = B_r + jB_i$;
- Coefficienti: $W^k = W_r + jW_i, W^{k+\frac{N}{2}} = -W_r - jW_i$.
- Uscite: $A' = A'_r + jA'_i, B' = B'_r + jB'_i$.

Mentre le operazioni possono essere riassunte in:

- $A' = A_r + B_rW_r - B_iW_i + j(A_i + B_rW_i + B_iW_r)$
- $B' = A_r - B_rW_r + B_iW_i + j(A_i - B_rW_i - B_iW_r)$

Con: $B'_r = 2A_r - A'_r \; e \; B'_i = 2A_i - A'_i$.

Considerando che in un sistema digitale non è possibile utilizzare un numero complesso è conveniente suddividere le operazioni trattando parte immaginaria e parte reale separatamente. Quindi le operazioni presentate precedentemente diventano:

$$M_1 = B_r \cdot W_r \qquad M_2 = B_i \cdot W_i \qquad M_3 = B_r \cdot W_i$$
$$M_4 = B_i \cdot W_r \qquad M_5 = A_r \cdot 2 \qquad M_6 = A_i \cdot 2$$

$$S_1 = A_r + M_1 \qquad S_2 = A'_r = S_1 - M_2 \qquad S_3 = A'_i + M_4$$
$$S_4 = A'_i = S_3 + M_4 \qquad S_5 = B'_r = M_5 - A'_r \qquad S_6 = B'_i = M_6 - A'_i$$

3.0 Progetto:

3.1 Specifiche:

Le specifiche richiedono di realizzare una Butterfly attraverso una architettura "Micro-Programmata". Essa deve realizzata con l'ausilio di un solo sommatore ed un solo moltiplicatore con un livello di Pipeline. Inoltre, tutti i dati su cui opera sono in notazione Fractional Point 0.X a 16 bit. I dati di ingressi A e B sono forniti dall'utente con due bit di guarda e sono quindi limitati tra -0.25 e 0.25. I coefficienti W invece, sono in formato a 16 bit senza bit di guardia. I risultati devono mantenere un formato a 16 bit ed è richiesto l'uso di un arrotondamento del tipo "Nearest Even".

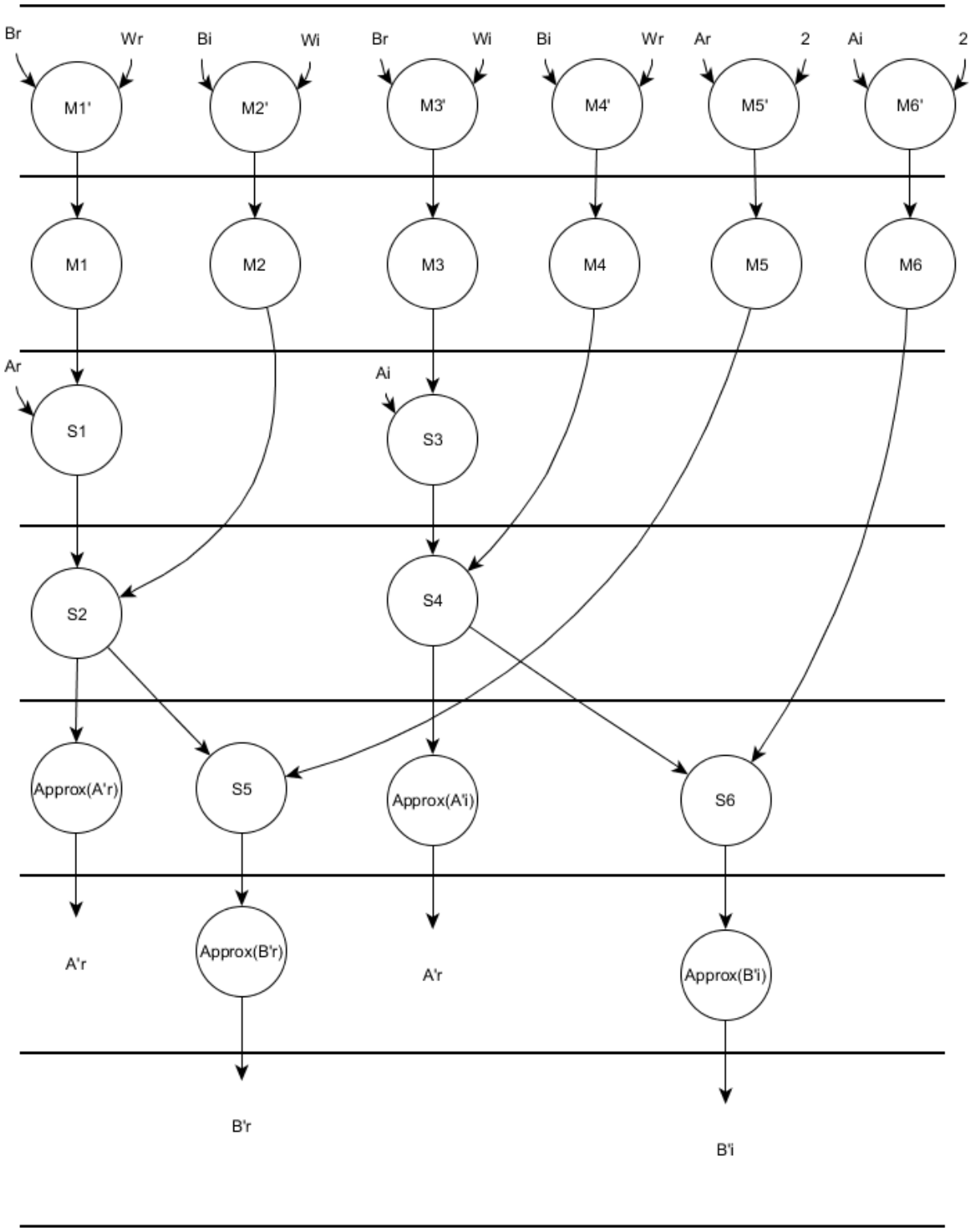
La Control Unit deve essere realizzata con indirizzamento Esplicito e completare un'istruzione in un ciclo di clock. La Butterfly inoltre attraverso alcuni segnali di controllo, definiti in fase di progetto, deve poter svolgere le operazioni in modalità singola e continua.

Nel nostro caso è stato scelto di realizzare una Butterfly in grado di elaborare dati sia autonomamente che insieme ad altre Butterfly attraverso un collegamento in cascata.

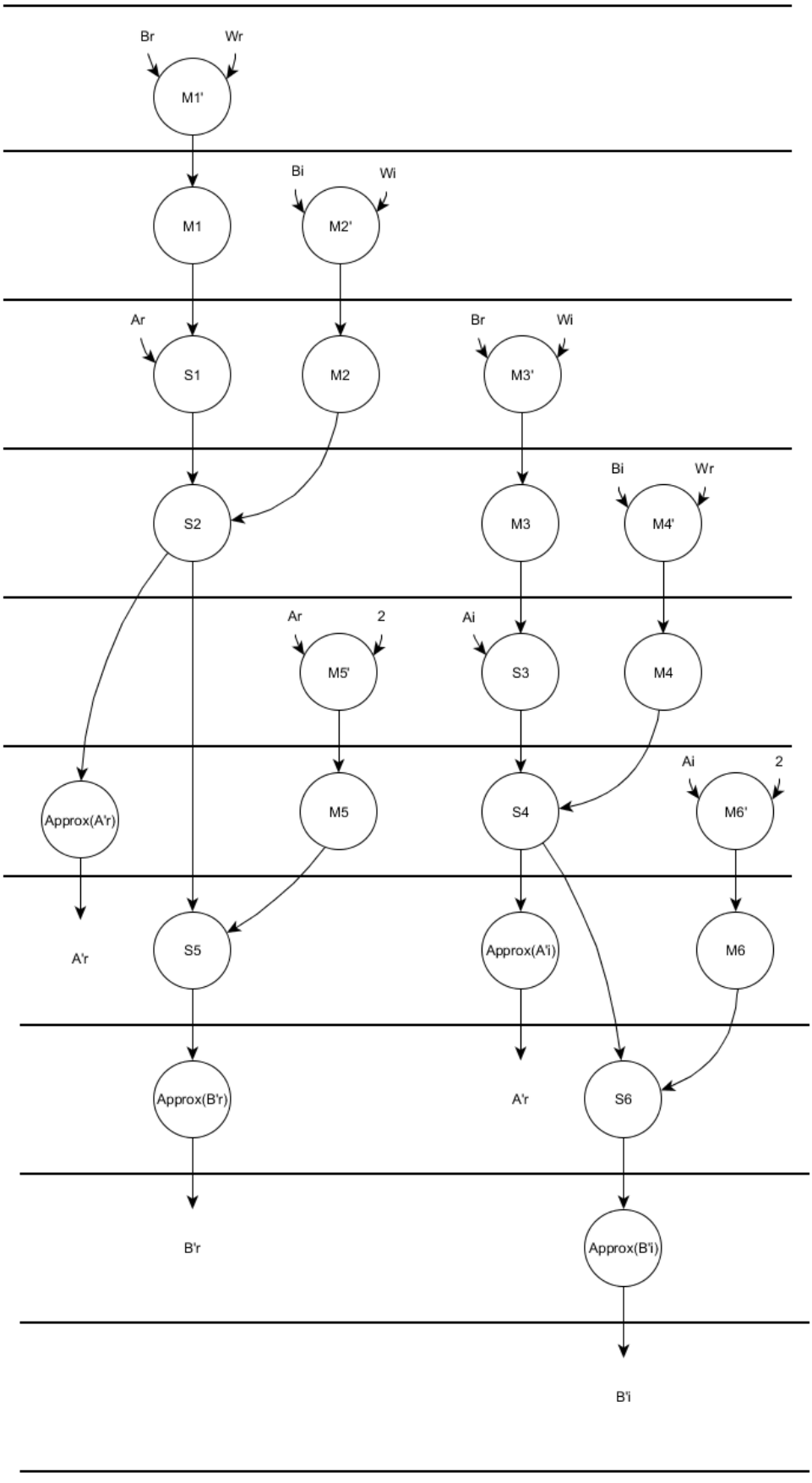
3.2 Dataflow Diagram

Per ottenere uno Scheduling delle operazioni efficiente si è partiti dalle sequenze ASAP e ALAP. Di seguito sono riportati i relativi diagrammi.

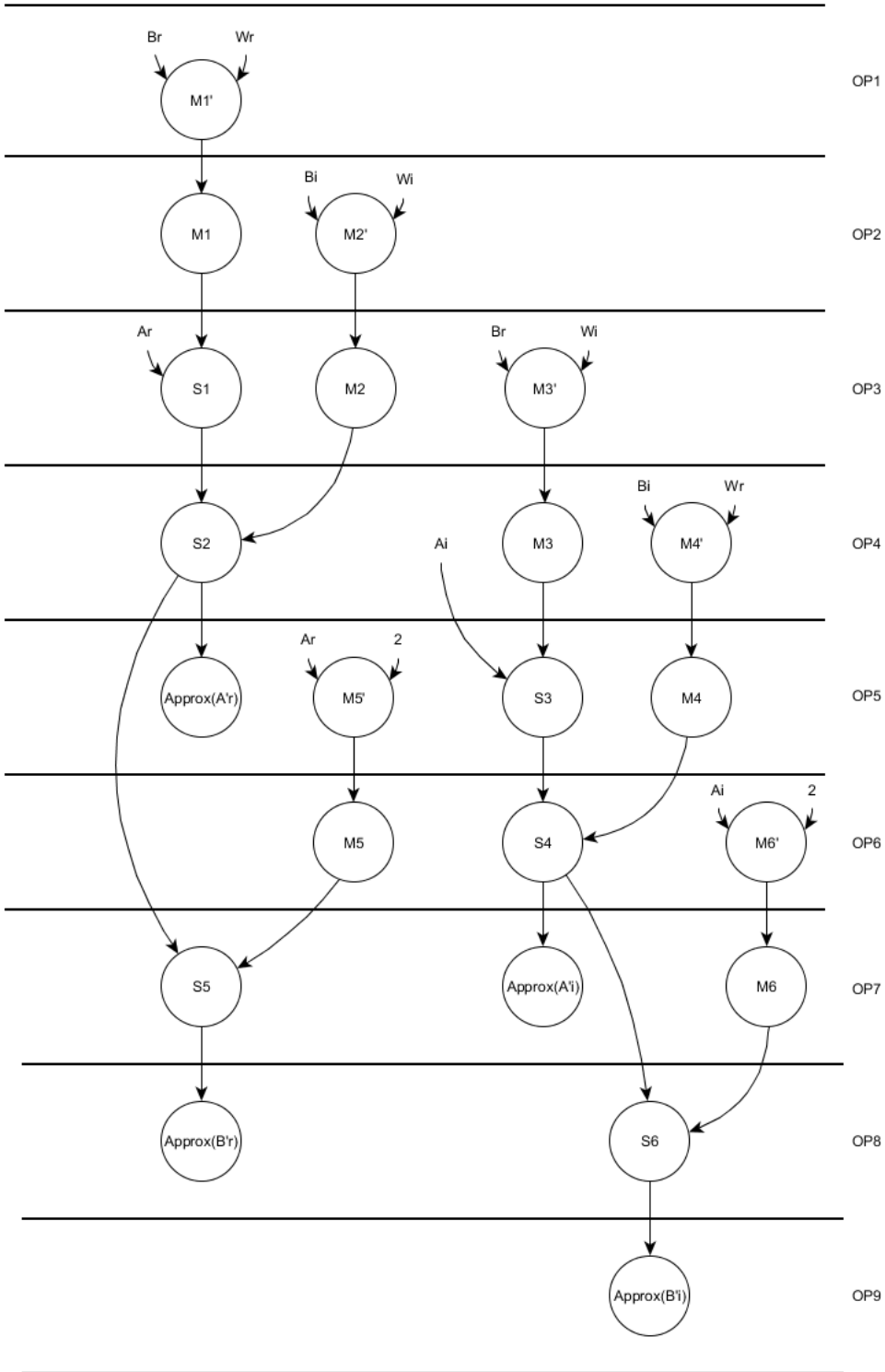
ASAP:



ALAP:

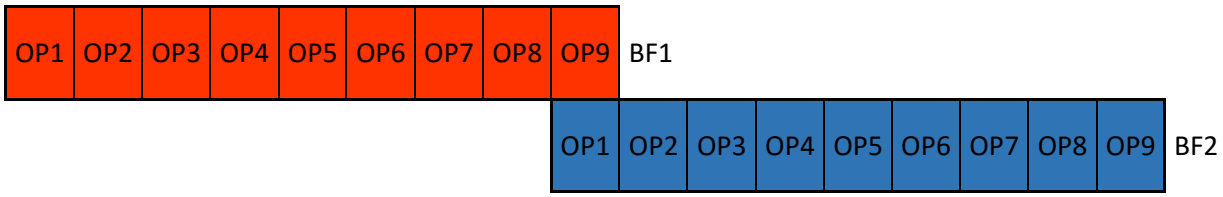

Br
Wr
M1'
M1
Bi
Wi
M2'
Ar
S1
M2
Br
Wi
M3'
S2
M3
Bi
Wr
M4'
Ar
2
M5'
Ai
S3
M4
Approx(A'r)
M5
S4
Ai
2
M6'
A'r
S5
Approx(A'i)
M6
Approx(B'r)
A'r
S6
B'r
Approx(B'i)
B'i

Come si può notare la sequenza *ASAP* non è realizzabile a causa delle specifiche che limitano l'uso del numero di moltiplicatori e sommatori. La soluzione *ALAP* invece risulta poco ottimizzata nelle prestazioni. L'utilizzo delle Butterfly per il calcolo della *FFT* prevede di collegare più blocchi in cascata pertanto è stato scelto di fornire i risultati nello stesso ordine con cui vengono memorizzati i dati di ingresso. Inoltre è stata prevista una modalità di caricamento dei dati sequenziale: B_r, B_i, A_r, A_i.
Queste richieste non sono possibili in uno Scheduling *ALAP* a causa di A_i. È stato quindi derivato uno schema di calcolo in cui gli arrotondamenti vengono eseguiti con un approccio ASAP mentre il resto delle operazioni con un approccio *ALAP*.

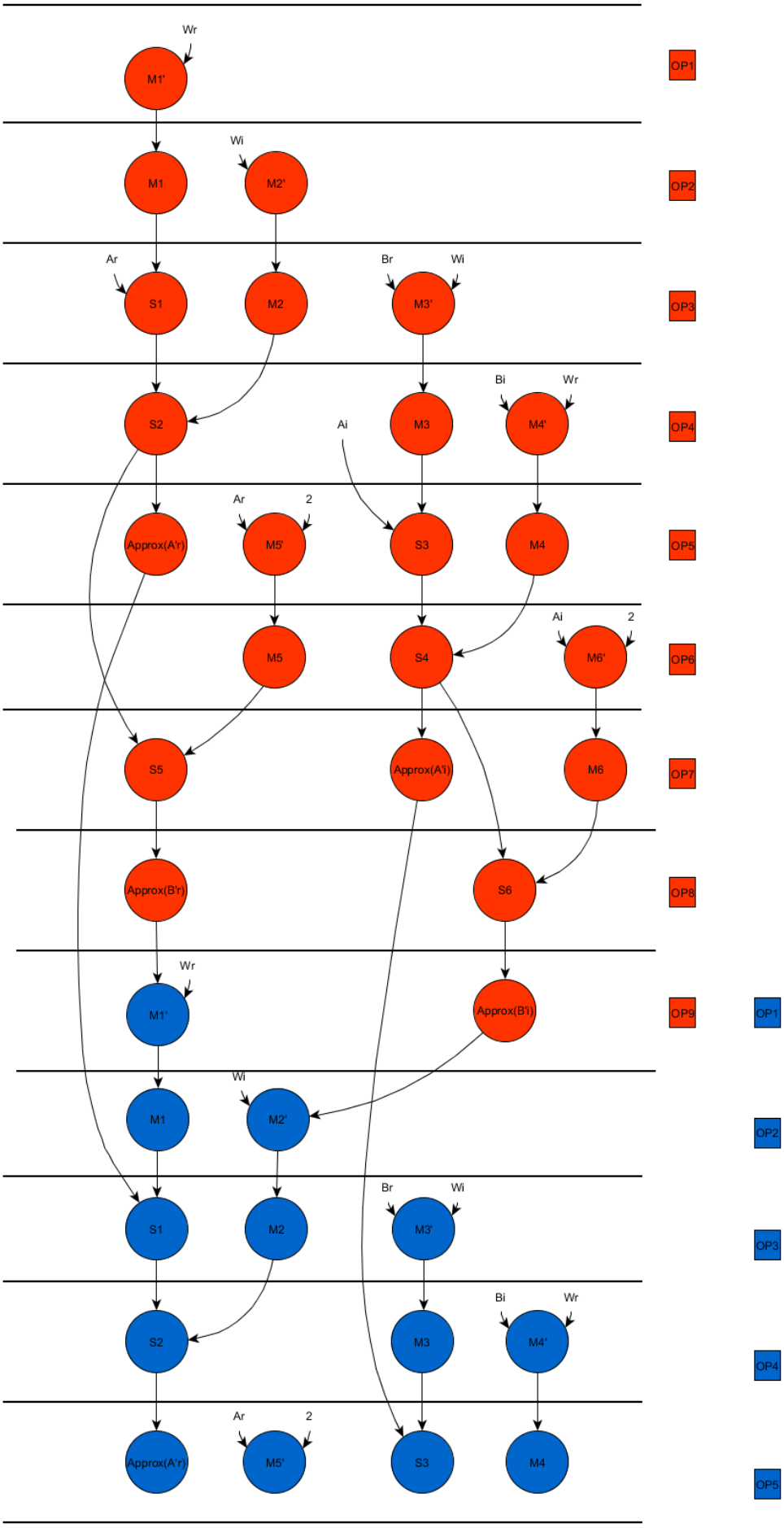


3.3 Butterfly in Sequenza

Osservando il **Dataflow Diagram** si nota che la sequenza dei dati di ingresso è B_r, B_i, A_r, A_i mentre la sequenza dei dati di uscita è A'_r, A'_i, B'_r, B'_i. Si può quindi pensare di permettere alla seconda Butterfly di iniziare le operazioni mentre la prima sta ancora terminando il calcolo di B'_i. Si ottiene quindi il seguente schema di calcolo:

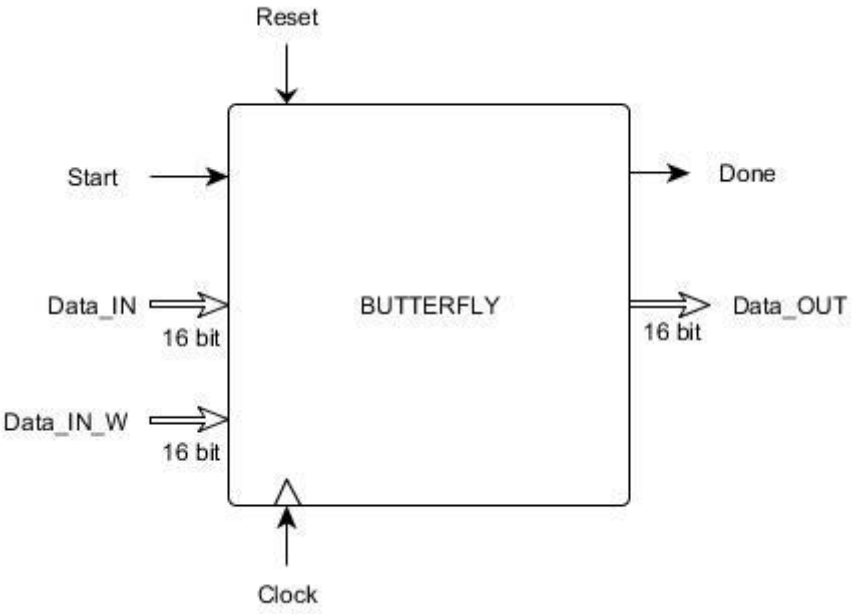


Come si può notare mentre la prima Butterfly (ROSSO) sta terminando la sequenza, la seconda (BLU) inizia iniziare la propria elaborazione. Di seguito è mostrato il **Dataflow Diagram** della modalità sequenza.


Wr
M1'
OP1
Wi
M1
M2'
OP2
Ar
Br
Wi
S1
M2
M3'
OP3
Bi
Wr
S2
Ai
M3
M4'
OP4
Ar
2
Approx(A'r)
M5'
S3
M4
OP5
Ai
2
M5
S4
M6'
OP6
S5
Approx(A'i)
M6
OP7
Approx(B'r)
S6
OP8
Wr
M1'
Approx(B'i)
OP9
OP1
Wi
M1
M2'
OP2
Br
Wi
S1
M2
M3'
OP3
Bi
Wr
S2
M3
M4'
OP4
Ar
2
Approx(A'r)
M5'
S3
M4
OP5

3.4 Struttura della Butterfly

Dal *Dataflow Diagram* si osserva che se si memorizzano i dati nell'ordine è B_r, B_i, A_r, A_i si può utilizzare una sola porta di ingresso, **Data_IN**. I coefficienti W sono invece memorizzati attraverso la porta **Data_IN_W** nell'ordine W_r, W_i. I risultati sono mostrati all'utente secondo la sequenza B'_r, B'_i, A'_r, A'_i usando la porta **Data_OUT**. La Butterfly può essere quindi schematizzata come:



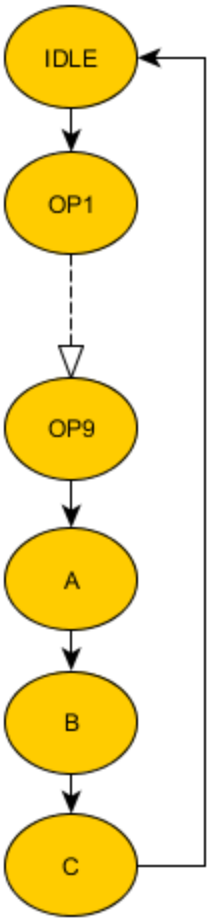
- **Start:** segnale di controllo che permette di iniziare le operazioni della macchina. Esso è utilizzato inoltre per gestire le modalità "Singola" e "Continua";
- **Done:** segnale di uscita che viene attivato dalla Butterfly al termine delle operazioni, in corrispondenza dei dati di uscita;
- **Data_IN:** porta di ingresso dei dati B_r, B_i, A_r, A_i;
- **Data_IN_W:** porta di ingresso dei coefficienti W_r, W_i;
- **Data_OUT:** porta di uscita dei risultati B'_r, B'_i, A'_r, A'_i;
- **Reset**: segnale di azzeramento della Butterfly;
- **Clock:** ingresso del segnale di temporizzazione.

3.5 Definizione degli Stati e Modalità di Funzionamento

3.5.1 Definizione delle operazioni

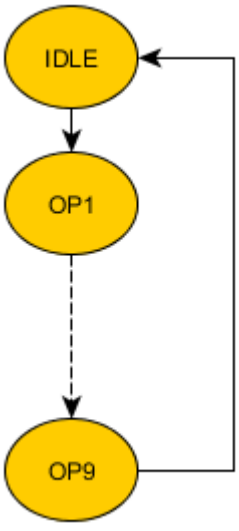
I dati di ingresso dalla porta **Data_IN** devono essere memorizzati prima di poter essere usati all'interno dell'algoritmo di calcolo descritto dalle operazioni *OP1…OP9*. Per eseguire la memorizzazione di B_r e W_r è necessario aggiungere l'operazione **IDLE** prima di *OP1*. Inoltre, per generare in uscita la sequenza dei dati B'_r, B'_i, A'_r, A'_i è necessario aggiungere tre operazioni successive a *OP9*: *A, B, C*. Si ottiene quindi il seguente Flow Chart:

- *IDLE*: memorizzazione di B_r e W_r;
- *OP1*: memorizzazione di B_i e W_i;
- *OP2*: memorizzazione di A_r;
- *OP3*: memorizzo di A_i;
- *OP4…OP8*: svolgimento delle operazioni;
- *OP9*: presentazione in uscita del dato B'_r;
- *A*: presentazione in uscita del dato B'_i;
- *B*: presentazione in uscita del dato A'_r;
- *C*: presentazione in uscita del dato A'_i;



Come si può osservare, è possibile ottimizzare il diagramma utilizzando solo lo stato *IDLE* in aggiunta a *OP1…OP9*. Si eliminano così A, B, C e si ottiene il seguente *Flow Chart*:

- *IDLE*: presentazione in uscita di B'_i e memorizzazione di B_r e W_r;
- *OP1*: presentazione in uscita di A'_r e memorizzazione di B_i e W_i;
- *OP2*: presentazione in uscita di A'_i e memorizzazione di A_r;
- *OP3*: memorizzazione di A_i e svolgimento di OP3;



- *OP4*...*OP8*: svolgimento delle operazioni;
- *OP9*: presentazione in uscita di B_r' e svolgimento di OP9;
- *IDLE*…*OP2*: presentazione in uscita di B_i', A_r', A_i' e inizio di un nuovo ciclo di operazioni;

Come può osservare dal precedente *Flow Chart* l'elaborazione dei dati e la presentazione dei risultati in uscita si svolge dallo stato *IDLE* fino a *OP2* del ciclo successivo. In questo modo sono stati eliminati *A, B* e *C* ottimizzando il numero di operazioni.

3.4.2 Modalità di Funzionamento

Modalità Singola:

Nella modalità ***Singola*** la Butterfly esegue un solo ciclo di operazioni. Dopo il Reset è possibile iniziare l'elaborazione attivando il segnale di Start e fornendo in ingresso il primo operando. Se lo Start rimane attivo durante tutto il caricamento dei dati di ingresso la Butterfly comincia l'elaborazione. Viene quindi eseguito un ciclo completo ritornando in OP2 dove termina le sequenza dei dati di uscita presentati attraverso **Data_OUT**.

	IDLE	OP1	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	IDLE	OP1	OP2	IDLE	IDLE
Start	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
Data_IN	Br	Bi	Ar	Ai											
Data_IN_W	Wr	Wi									Wr	Wi			
Done	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
Data_OUT										B'r	B'i	A'r	A'i		

Modalità Continua:

La modalità ***Continua*** permette alla Butterfly di eseguire in sequenza più cicli di operazioni. Dopo il Reset è possibile iniziare l'elaborazione attivando il segnale di Start e fornendo in ingresso il primo operando. Se lo Start rimane attivo durante tutto il caricamento dei dati la Butterfly comincia l'elaborazione. Viene quindi eseguito un ciclo completo e non appena viene raggiunto lo stato *IDLE* l'utente, attivando parallelamente il segnale di Start, può caricare i nuovi dati di ingresso prelevando contemporaneamente i dati di uscita. In questo modo la Butterfly incomincia un nuovo ciclo di elaborazione. In caso contrario dopo *OP2* la Butterfly si azzera e ritorna in *IDLE*.

	IDLE	OP1	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	IDLE	OP1	OP2	OP3	OP4	OP5	OP6	OP7
Start	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0
Data_IN	Br	Bi	Ar	Ai							Br	Bi	Ar	Ai				
Data_IN_W	Wr	Wi									Wr	Wi						
Done	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
Data_OUT										B'r	B'i	A'r	A'i					

Butterfly in Sequenza:

La modalità ***Sequenza*** permette di collegare due Butterfly in cascata sfruttando il **Done** della prima come segnale di Start della seconda. Si realizza quindi il seguente schema di elaborazione:

Butterfly 1	IDLE	OP1	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	IDLE	OP1	OP2									
Butterfly 2	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	OP1	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	IDLE	OP1	OP2
Start1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Data_IN1	Br	Bi	Ar	Ai																		
Data_IN_W1	Wr1	Wi1																				
Done1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Data_OUT1										B'r1	B'i1	A'r1	A'i1									
Start2	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Data_IN2										B'r1	B'i1	A'r1	A'i1									
Data_IN_W2										Wr2	Wi2											
Done2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Data_OUT2																			B'r2	B'i2	A'r2	A'i2

3.4 Tempo di Vita delle Variabili

Il tempo di vita permette di valutare la durata di ogni operando all'interno dell'algoritmo di elaborazione. Di seguito è riportata la tabella riassuntiva:

	IDLE	OP1	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	IDLE	OP1	OP2
Ar													
Ai													
Br													
Bi													
Wr													
Wi													
M1													
M3													
S1													
S3													
M2													
M4													
M5													
M6													
S2													
S4													
S5													
S6													
A'r													
A'i													
B'r													
B'i													

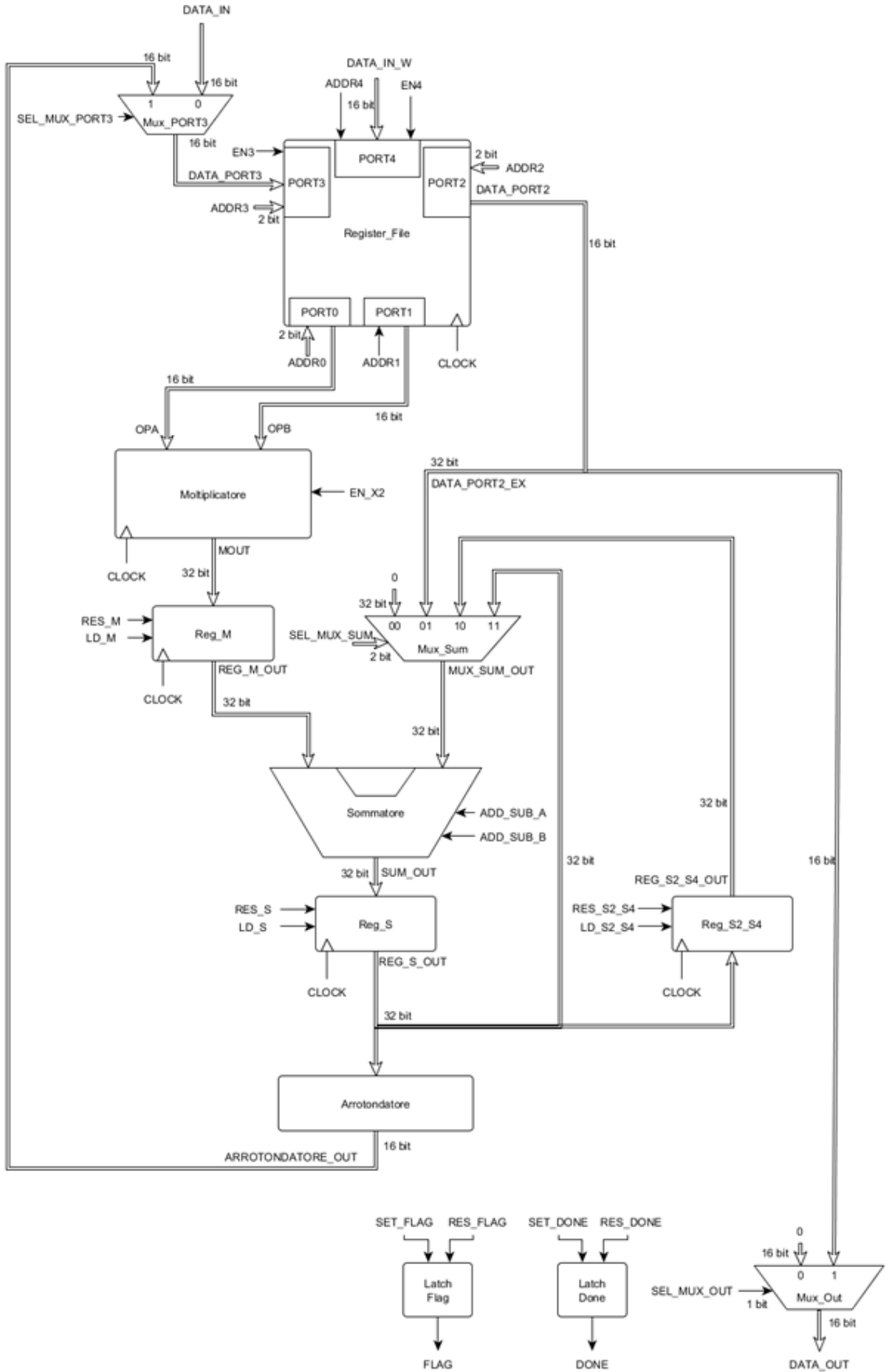
In blu sono indicati i tempi di vita delle variabili appartenenti allo stesso ciclo di elaborazione. In arancione invece viene indicato un ipotetico ciclo di elaborazione successivo al primo. In prima analisi sarebbe necessario un numero di registri pari al numero di variabili. In realtà è possibile ottimizzare la memorizzazione ottenendo la seguente tabella.

	IDLE	OP1	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	IDLE	OP1	OP2
REG_WR		Wr	Wr	Wr	Wr	Wr	Wr	Wr	Wr	Wr		Wr	Wr
REG_WI			Wi	Wi	Wi	Wi	Wi	Wi	Wi	Wi			Wi
REG_BR		Br	Br	Br						B'r		Br	Br
REG_BI			Bi	Bi	Bi						B'i		Bi
REG_AR				Ar	Ar	Ar	A'r	A'r	A'r	A'r	A'r	A'r	
REG_AI					Ai	Ai	Ai		A'i	A'i	A'i	A'i	A'i
REG_M				M1	M2	M3	M4	M5	M6				
REG_S					S1	S2	S3	S4	S5	S6			
REG_S2_S4							S2	S2	S4				

Come si può notare i registri per la memorizzazione dei dati di ingresso sono stati utilizzati anche per la memorizzazione dei rispettivi risultati di uscita. Il registro **Reg_M** invece permette di ottimizzare i collegamenti tra il moltiplicatore ed il sommatore. Infatti tutti i risultati forniti dal moltiplicatore vengono utilizzati successivamente dal sommatore. Il registro **Reg_S** è usato per memorizzare il risultato precedente del sommatore che verrà nelle operazioni successive. Infine il registro **Reg_S2_S4** memorizza le somme S2 e S4 che vengono utilizzate durante lo svolgimento dell'algoritmo. Anche in questo caso le caselle in Blu indicano il tempo di vita delle variabili che fanno parte dello stesso ciclo di elaborazione. In arancione sono indicate le variabili di un ipotetico ciclo successivo.

3.5 Execution Unit:

3.5.2 Descrizione



L'Execution Unit è stata progettata utilizzando il **Data Flow Diagram** e la tabella relativa al **Tempo di Vita delle Variabili**. Per minimizzare il numero di connessioni interne è stato progettato il blocco **Register_File**. Esso permette al sommatore e al moltiplicatore di accedere agli operandi e ai coefficienti durante l'elaborazione e di salvare i risultati B'_r, B'_i, A'_r *e* A'_i. Il **Register_File** contiene quindi i registri: **Reg_Br, Reg_Bi, Reg_Ar, Reg_Ai, Reg_Wr e Reg_Wi**. Il loro parallelismo è a 16 bit. L'uso del moltiplicatore richiede un parallelismo doppio rispetto agli operandi pertanto il risultato è fornito su 32 bit. I restanti registri **Reg_S**, **Reg_M** e **Reg_S2_S4** però hanno un parallelismo a 31 bit in quanto un bit del moltiplicatore viene ignorato. Ulteriori dettagli sono mostrati nei successivi paragrafi. Come si osserva la porta *PORT2* del **Register_File** è collegata al registro **Mux_Sum** che permette di selezionare il secondo operando del sommatore. Il **Register_File** ha un parallelismo a 16 bit mentre il sommatore lavora su operandi a 31 bit è quindi necessario estendere il numero di bit della porta *PORT2* mantenendo invariato il dato. Questa operazione, nella notazione Fractional Point è eseguita nel seguente modo:

Dato: XXXXXXXXXXXXXXXX dove con X si indica un generico valore del bit.

Dato Esteso: XXXXXXXXXXXXXXXX0000000000000000.

Queste scelte hanno permesso di ottimizzare il numero di bus e di ridurre l'area globale di memoria. I dati B_r, B_i, A_r, A_i provenienti dall'esterno, sono memorizzati attraverso la porta *PORT3* del **Register_File,** la stessa utilizzata per la memorizzazione dei risultati. È stato quindi inserito il multiplexer **Mux_PORT3** per permettere la scelta del dato da memorizzare. I coefficienti Wr e Wi invece sono caricati attraverso la porta *PORT4* del **Register_File**. I risultati dell'elaborazione vengono forniti in uscita tramite il multiplexer **Mux_Out**. Esso permette di mostrarli attraverso la porta *PORT2* del **Register_File**. Il **Latch_Done** viene utilizzato per asserire la linea **Done** in corrispondenza dei dati di uscita forniti dalla Butterfly. Il **Latch_Flag** invece viene utilizzato per permettere alla Butterfly di concludere correttamente il ciclo delle operazioni da *IDLE* a *OP2*. Essendo un segnale di controllo, il suo funzionamento dettagliato verrà illustrato nelle sezioni successive. Di seguito sono descritti tutti i blocchi dell'Execution-Unit ed il loro ruolo nell'elaborazione.

3.5.3 Descrizione dei Blocchi

Reg_M, Reg_S2_S4, Reg_S:

I registri **Reg_M, Reg_S2_S4** e **Reg_S** sono registri attivi sul fronte di salita del clock che vengono utilizzati per memorizzare i calcoli parziali, utili al completamento dell'elaborazione. Essi presentano le seguenti porte di I/O e segnali di controllo:

- *LD*: segnale di abilitazione al caricamento sincrono del dato presente all'ingresso *Input*. Esso è attivo con livello logico '1';
- *RES*: segnale di reset asincrono che porta il contenuto del dato presente all'interno del registro al valore 0. Esso è attivo con livello logico '1';
- *Input*: porta di ingresso del dato con parallelismo 31 bit;
- *Output*: porta di uscita del dato contenuto all'interno del registro, parallelismo 31 bit;
- *Clock*: ingresso del segnale di temporizzazione.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity RegisterDataPath is
PORT(
Clock,Reset,Load:IN std_logic;
Input: IN std_logic_vector(30 downto 0);
Output : OUT std_logic_vector(30 downto 0));
end entity;


architecture behaviour of RegisterDataPath is

BEGIN
Process(Reset,Clock)
BEGIN
if(Reset='1')then
Output<=(others=>'0');
elsif(Clock'EVENT and Clock='1')then
if(Load='1')then
Output<=Input;
end if;
end if;
end process;
end architecture;
```

Mux_Sum:

Il componente **Mux_Sum** è un multiplexer a quattro vie che è utilizzato all'interno della Execution Unit per selezionare il secondo operando del sommatore. Esso presenta le seguenti porte di I/O e segnali di controllo:

- *SEL_MUX*: segnale di controllo, con parallelismo a 2 bit, che permette di scegliere quale dei quattro ingressi viene collegato all'uscita *Output*;
- *A, B, C, D*: ingressi dei dati. Il loro parallelismo è a 31 bit;
- *Output*: porta di uscita del multiplexer con parallelismo a 31 bit.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Mux4to1 is
GENERIC ( N: integer:=16);
PORT( A,B,C,D:IN std_logic_vector((N-1) downto 0);
Sel:IN std_logic_vector(1 downto 0);
Output :OUT std_logic_vector((N-1) downto 0));
end entity;

architecture behaviour of Mux4to1 is
BEGIN
with Sel select
Output <= A when "00",
          B when "01",
          C when "10",
          D when "11",
          (others=>'0') when others;
end architecture;
```

Mux_PORT3, Mux_Out:

I componenti **Mux_PORT3 e Mux_Out** sono dei multiplexer a due vie che vengono utilizzati per dirigere il flusso dei dati in ingresso alla porta *PORT3* del **Register_File** e in uscita dalla Butterfly. Essi presentano le seguenti porte di I/O e segnali di controllo:

- *SEL_MUX*: segnale di controllo, con parallelismo a 1 bit, che permette di scegliere quale dei due ingressi viene collegato all'uscita *Output*;
- *A, B*: ingressi dei dati. Il loro parallelismo è a 16 bit;
- *Output*: porta di uscita del multiplexer con parallelismo a 16 bit.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;


entity Mux2to1 is
GENERIC ( N: integer:=8);
PORT(A,B: std_logic_vector((N-1) downto 0);
Sel:IN std_logic;
Output :OUT std_logic_vector((N-1) downto 0));
end entity;


architecture behaviour of Mux2to1 is

BEGIN
Output<= A when (Sel='0') else B;

end architecture;
```

Moltiplicatore:

Il **Moltiplicatore** possiede un livello di Pipeline pertanto, il risultato è disponibile solo dopo un ciclo di clock. Esso presenta le seguenti porte di I/O e segnali di controllo:

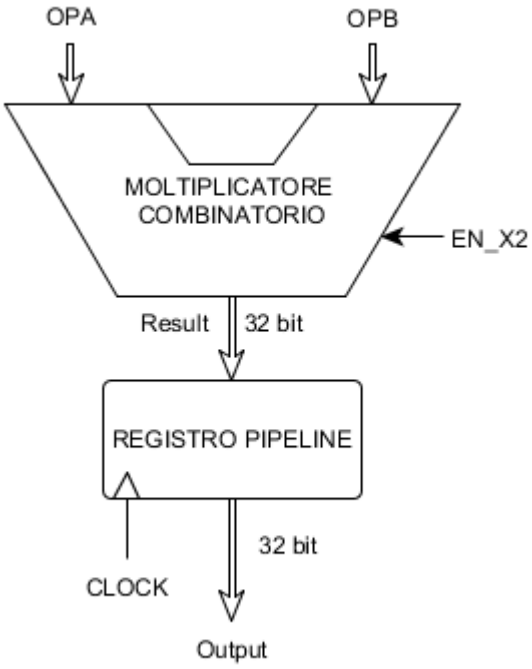
- *OPA, OPB*: ingressi degli operandi con parallelismo a 16 bit;
- *EN_X2*: segnale di controllo che permette di selezionare l'operazione di moltiplicazione per 2 sull'operando *OPA*. Esso è attivo con livello logico '1';
- *Output*: porta di uscita del risultato con parallelismo a 32 bit;

Architettura interna:

La progettazione del **Moltiplicatore** su numeri rappresentati con notazione Fractional Point presenta delle differenze rispetto ad uno convenzionale. Dati due operandi di ingresso con formato X.X il risultato di uscita è presentato con formato XX.XX. Viene quindi raddoppiato il parallelismo sia della parte intera che della parte decimale. Nel nostro caso, avendo connesso direttamente il moltiplicatore al sommatore, è necessario fornire un risultato di formato concorde agli operandi di ingresso. Questa operazione viene eseguita nel seguente modo:

I. *OPA*: X.XXXXXXXXXXXXXXX, Operando A;

II. *OPB*: Y.YYYYYYYYYYYYYYY, Operando B;



III. *Result*: ZZ.ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ, Operando A * Operando B;

IV. Output: U.UUUUUUUUUUUUUUUUUUUUUUUUUUUUUU0, il dato di uscita nel formato corretto è ottenuto traslando a sinistra *Result* e forzando a zero il bit meno significativo. Questa operazione è possibile in quanto i dati su cui la Butterfly esegue le operazioni sono limitati tra -1 e 1.

Il registro di Pipeline è stato inserito tra il dato di uscita del moltiplicatore combinatorio *Result* e l'uscita Output complessiva. Esso permette anche di effettuare il cambiamento del formato presentato in precedenza. Il **Moltiplicatore** può inoltre eseguire l'operazione di moltiplicazione per due, attraverso l'abilitazione del segnale *EN_X2*. Essa avviene sfruttando l'operazione di traslazione.

Codice VHDL:

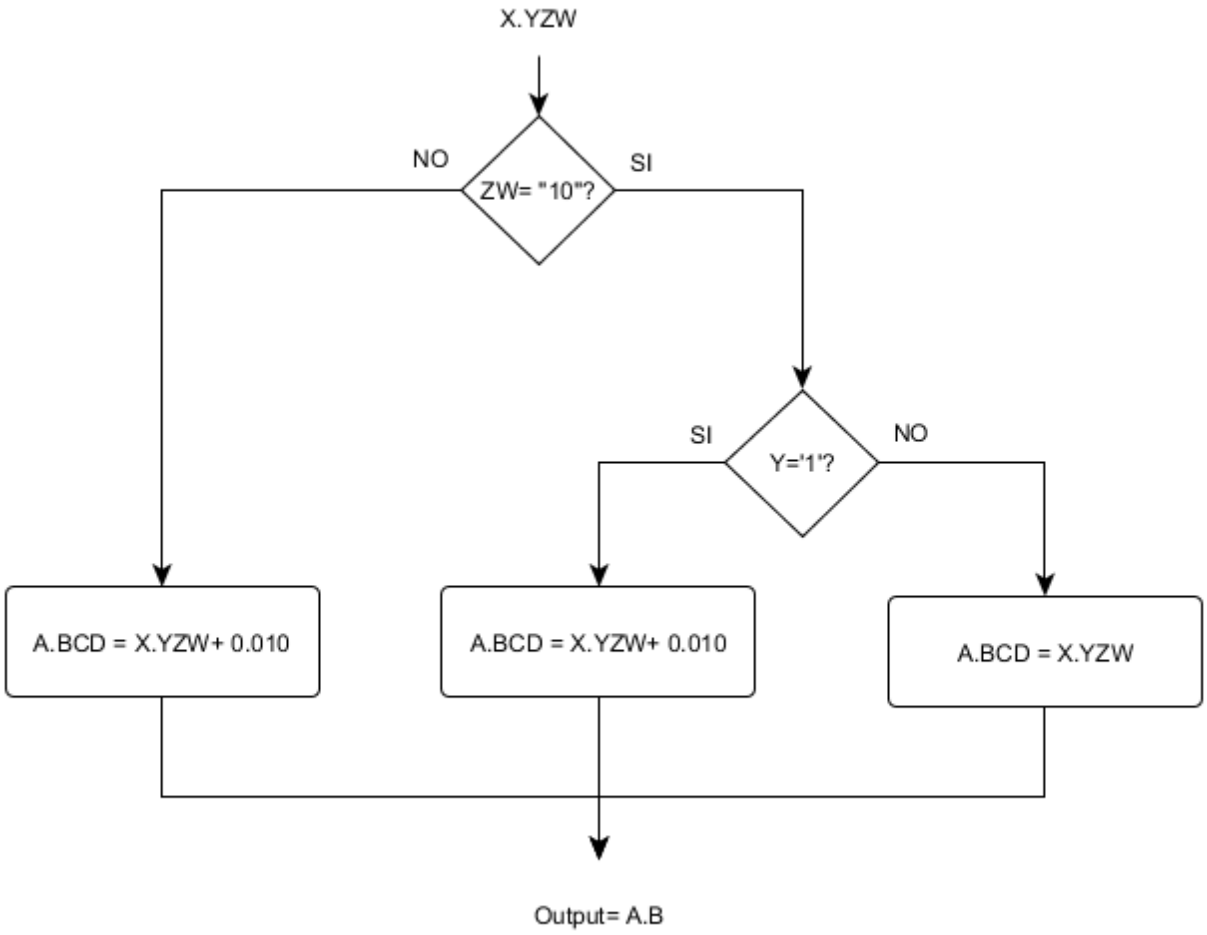
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Moltiplicatore is
PORT(
A,B: IN std_logic_vector(15 downto 0);
Clock,En_X2: IN std_logic;
Output : OUT std_logic_vector(31 downto 0));
end entity;


architecture behaviour of Moltiplicatore is

signal Result: SIGNED ( 31 downto 0);
BEGIN
Process(A,B,En_X2)
BEGIN
if(En_X2='0')then
Result<=signed(A) * signed(B);
else
Result<=signed(A) & "0000000000000000";
end if;
end process;

Process(Clock)
BEGIN
if(Clock'EVENT AND Clock='1')then
Output<=std_logic_vector(Result(30 downto 0) & '0');
end if;
end process;

end architecture;
```

Arrotondatore:

Il blocco **Arrotondatore** viene utilizzato per approssimare i risultati dell'elaborazione su un formato a 16 bit. Esso utilizza l'approssimazione **Nearest Even**. Il suo funzionamento è puramente combinatorio. L' **Arrotondatore** presenta le seguenti porte di I/O:

- *Input*: ingresso del dato da approssimare con parallelismo a 31 bit;
- *Output*: porta di uscita del dato approssimato con parallelismo a 16 bit;

Algoritmo:



L'algoritmo **Nearest Even** permette di ottenere un *"Errore di Bias"* nullo sui dati approssimati. Si consideri un dato di ingresso binario del tipo X.YZW da approssimare in A.B. Inizialmente è necessario verificare se i bit da trascurare assumono valore pari a metà della risoluzione desiderata, in questo ZW = "10". Se questa condizione è verificata allora:se il bit Y è 1 (numero dispari) viene sommato a X.YZW la metà della risoluzione 0.100.

A.BCD = X.YZW + "0.010"

Il dato approssimato risulta quindi pari ad A.B eseguendo un approssimazione per eccesso al pari superiore. Se invece Y = ‘0’ (numero pari) il dato approssimato risulta invece:

A.BCD = X.YZW

Quindi A.B è ottenuto con un approssimazione per difetto al pari inferiore.

Qualora ZW sia diverso da metà della risoluzione viene sommato al dato di ingresso X.YZW il numero “0.010”:

A.BCD = X.YZW + “0.010”

Quindi ottengo A.B con un arrotondamento tradizionale.

Codice VHDL:

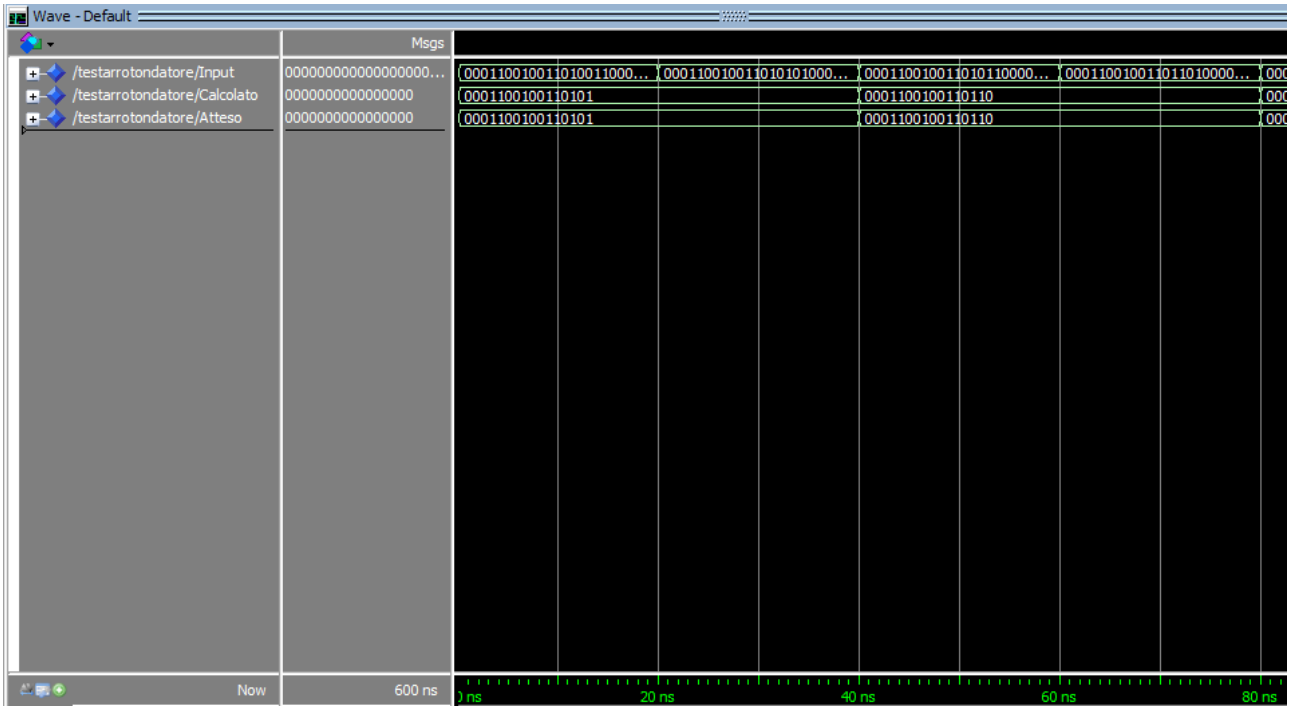
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Arrotondatore is
PORT(
Input:IN std_logic_vector ( 30 downto 0);
Output : OUT std_logic_vector (15 downto 0));
end entity;

architecture behaviour of Arrotondatore is
Signal Result : SIGNED(30 downto 0);
BEGIN

Process(Input)
BEGIN
if( Input(14 downto 0) = "100000000000000")then

if( Input(15)='1')then --Dispari
Result<=signed(Input) + to_signed(16384,31);
else
Result<=signed(Input);
end if;

else
Result<=signed(Input) +to_signed(16384,31) ;
end if;
end process;

Output<=std_logic_vector(Result(30 downto 15));

end architecture;
```

Simulazione:

Per verificare il corretto funzionamento del blocco **Arrotondatore** è stato creato un TestBench VHDL in grado di verificare tutte le tipologie di approssimazioni possibili. Di seguito è mostrato il codice VHDL e un'immagine della simulazione.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity TestArrotondatore is
end entity;

architecture behaviour of TestArrotondatore is
component Arrotondatore is
PORT(
Input:IN std_logic_vector ( 30 downto 0);
Output : OUT std_logic_vector (15 downto 0));
end component;

Signal Input:std_logic_vector ( 30 downto 0);
signal Calcolato,Atteso : std_logic_vector (15 downto 0);

BEGIN
Process
BEGIN
input<="00011001001101001100000000000000"; -- Partenza: 0.196922302246094
Atteso<="0001100100110101";
-- Risultato:"0001100100110101"  0.196929931640625 Approssimazione per eccesso del tipo 0.8 -> 1
--Arrotondamento per eccesso semplice

wait for 20 ns;
input<="00011001001101010100000000000000"; -- Partenza: 0.196937561035156
Atteso<="0001100100110101";
-- Risultato:"0001100100110101"  0.196937561035156  Approssimazione per difetto semplice del tipo 1.2 -> 1

wait for 20 ns;
input<="00011001001101011000000000000000"; -- Partenza:  0.196945190429688
Atteso<="0001100100110110";
-- Risultato:"0001100100110110" 0.19696044921875  Approssimazione per eccesso a metà intervallo per eccesso del
tipo 1.5 -> 2

wait for 20 ns;
input<="00011001001101101000000000000000"; -- Partenza: 0.196975708007813
Atteso<="0001100100110110";
-- Risultato:"0001100100110110" 0.19696044921875 Approssimazione per difetto a metà intervallo per difetto del
tipo 2.5 -> 2
wait for 20 ns;
input<=(others=>'0');
Atteso<=(others=>'0');
wait;
end process;

ARR: Arrotondatore PORT MAP(Input,Calcolato);

end architecture;
```



Come si può osservare il risultato atteso corrisponde con quello calcolato. Il primo dato viene arrotondato con un arrotondamento per eccesso, il secondo per difetto, il terzo per eccesso al pari superiore e l'ultimo per difetto al pari inferiore.

Sommatore:

Il blocco **Sommatore** è un componente puramente combinatorio che permette di svolgere le operazioni di somma e sottrazione tra gli ingressi A, B. Esso presenta le seguenti porte di I/O e segnali di controllo:

- *A, B*: ingressi degli operandi con parallelismo a 31 bit;

- *ADD_SUB_A, ADD_SUB_B*: segnali di controllo che permettono di selezionare tre possibili operazioni. Nel caso in cui siano entrambi a 0 o a 1 viene eseguita la somma. Quando sono di valore opposto viene eseguita la differenza. In particolare il suffisso _A o _B indica quale operando viene complementato e quindi sottratto:

ADD_SUB_A	ADD_SUB_B	Output
0	0	A+B
0	1	A-B
1	0	-A+B
1	1	A+B

- *Output*: porta di uscita con parallelismo a 31 bit in cui viene fornito il risultato dell'operazione;

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Sum is
PORT(
A,B: IN std_logic_vector(30 downto 0);
AddSubA,AddSubB: IN std_logic;
Output : OUT std_logic_vector(30 downto 0));
end entity;

architecture behaviour of Sum is

BEGIN

Process(A,B,AddSubB,AddSubA)
BEGIN
if(AddSubA='0' AND AddSubB='0')then
Output<=std_logic_vector(signed(A)+signed(B));

elsif(AddSubA='0' AND AddSubB='1')then
Output<=std_logic_vector(signed(A)-signed(B));

elsif(AddSubA='1' AND AddSubB='0')then
Output<=std_logic_vector(signed(B)-signed(A));
else
Output<=std_logic_vector(signed(A)+signed(B));
end if;
end process;

end architecture;
```

Register_File:

Il **Register_File** costituisce il blocco di memorizzazione dei dati e dei coefficienti W della Butterfly. Esso presenta tre porte di lettura e due di scrittura. La sua architettura è stata ottimizzata per ottenere il numero minimo di segnali di controllo. La lettura è asincrona mentre la scrittura è sincrona. Il numero totale di registri interi è 6: **Reg_Br, Reg_Bi, Reg_Ar, Reg_Ai, Reg_Wr e Reg_Wi.**

PORT0, PORT2, PORT3

Le porte *PORT0*, *PORT2* e *PORT3* sono utilizzate per la lettura e per la scrittura dei registri **Reg_Br, Reg_Bi, Reg_Ar** e **Reg_Ai**. In particolare la *PORT3* è una porta di scrittura mentre le porte *PORT2* e *PORT0* sono di lettura. Di seguito è riportata la corrispondenza tra indirizzo e registro interno:

ADDRESS	REGISTRO
00	Reg_B_r
01	Reg_B_i
10	Reg_A_i
11	Reg_A_r

PORT1, PORT4

Le porte *PORT1, PORT4* vengono utilizzate per la lettura e per la scrittura dei registri **Reg_Wr e Reg_Wi**.In particolare la porta *PORT4* è una porta di scrittura mentre la porta *PORT1* è di lettura. Di seguito è riportata la corrispondenza tra indirizzo e registro:

ADDRESS	REGISTRO
0	Reg_W_r
1	Reg_W_i

PORTE I/O e Segnali di Controllo

Elenco delle porte di I/O e dei segnali di controllo:

- *DATA_PORT0*: porta di uscita per la lettura dei registri **Reg_Wr e Reg_Wi;**
- *ADDR_PORT0*: porta di indirizzo con parallelismo 1 bit per la selezione del registro da leggere attraverso *DATA_PORT0*;
- *DATA_PORT1*: porta di uscita per la lettura di registri **Reg_Br, Reg_Bi, Reg_Ar, Reg_Ai;**
- *ADDR_PORT1*: porta di indirizzo con parallelismo 2 bit per la selezione del registro da leggere attraverso *DATA_PORT1*;
- *DATA_PORT2*: porta di uscita per la lettura di registri **Reg_Br, Reg_Bi, Reg_Ar, Reg_Ai;**
- *ADDR_PORT2*: porta di indirizzo con parallelismo 2 bit per la selezione del registro da leggere attraverso *DATA_PORT2*;
- *DATA_PORT3*: porta di ingresso per la scrittura nei registri **Reg_Br, Reg_Bi, Reg_Ar, Reg_Ai;**
- ADDR_PORT3: porta di indirizzo con parallelismo 2 bit per la selezione del registro in scrivere attraverso *DATA_PORT3*;
- *EN3*: segnale di controllo che permette di abilitare la scrittura sincrona attraverso *DATA_PORT3*;
- *DATA_PORT4*: porta di ingresso per la scrittura nei registri **Reg_Wr e Reg_Wi;**
- *ADDR_PORT4*: porta di indirizzo con parallelismo 1 bit per la selezione del registro in scrivere attraverso *DATA_PORT4*;
- *EN4*: segnale di controllo che permette di abilitare la scrittura sincrona attraverso *DATA_PORT4*;

- *CLOCK*: ingresso del segnale di temporizzazione attivo sul fronte positivo;
- *RESET:* segnale per portare il contenuto dei registri **Reg_Br, Reg_Bi, Reg_Ar, Reg_Ai, Reg_Wr e Reg_Wi** a 0;

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;


entity RegisterFile is
PORT(
Addr0:IN std_logic_vector(1 downto 0);
Addr1:IN std_logic;
Addr2,Addr3: IN std_logic_vector(1 downto 0);
Addr4: std_logic;
DataOut0,DataOut1,DataOut2: OUT std_logic_vector( 15 downto 0);
DataIn3,DataIn4: IN std_logic_vector(15 downto 0);
En_PORT3,En_PORT4,Clock,Reset: IN std_logic);
end entity;


architecture behaviour of RegisterFile is
SIGNAL Br,Ar,Ai,Bi,Wi,Wr:Std_logic_vector(15 downto 0):=(others=>'0');
BEGIN

PORT0:Process(Addr0,Br,Bi,Ar,Ai)
BEGIN
case Addr0 is
When "00" => DataOut0<=Br;
When "01" => DataOut0<=Bi;
When "10" => DataOut0<=Ar;
when others => DataOut0<=Ai;
end case;
end process;

PORT2:Process(Addr2,Br,Bi,Ar,Ai)
BEGIN
case Addr2 is
When "00" => DataOut2<=Br;
When "01" => DataOut2<=Bi;
When "10" => DataOut2<=Ar;
when others => DataOut2<=Ai;
end case;
end process;

PORT1:Process(Addr1,Wr,Wi)
BEGIN
case Addr1 is
When '0' => DataOut1<=Wr;
when others => DataOut1<=Wi;
end case;
end process;

PORT3:Process(Clock,Reset)
```

```
BEGIN
if(Reset='1')then
Ai<=(others=>'0');
Ar<=(others=>'0');
Br<=(others=>'0');
Bi<=(others=>'0');
elsif(Clock'EVENT AND Clock='1')then
if(En_PORT3='1')then
case Addr3 is
When "00" => Br<=DataIn3;
When "01" => Bi<=DataIn3;
When "10" => Ar<=DataIn3;
when others => Ai<=DataIn3;
end case;
end if;
end if;
end process;

PORT4:Process(Clock,Reset)
BEGIN
if(Reset='1')then
Wi<=(others=>'0');
Wr<=(others=>'0');

elsif(Clock'EVENT AND Clock='1')then
if(En_PORT4='1')then

case Addr4 is
When '0' => Wr<=DataIn4;
When '1' => Wi<=DataIn4;
when others => Wr<=DataIn4;
end case;

end if;
end if;
end process;
end architecture;
```

Latch Done e Latch Flag

I componenti **Latch Flag** e **Latch Done** sono due latch Set-Reset che permettono di asserire o azzerare i rispettivi segnali di **Flag** e **Done**. Il primo è inviato alla Control-Unit mentre il secondo è un segnale di uscita usato per segnalare all'utente o ad un'altra Butterfly la disponibilità dei dati di uscita. Esso presenta i segnali di *SET*, *RESET* e l'uscita Output.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity LatchSR is
PORT(
set,reset:IN std_logic;
output : BUFFER std_logic);
end entity;
```

```
architecture behaviour of LatchSR is
BEGIN
Process(set,reset)
BEGIN
if(reset='1' AND set='0')then
output<='0';
elsif(reset='0' AND set='1')then
output<='1';
end if;
end process;

end architecture;
```

3.6.3 Codice VHDL Execution Unit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity ExecutionUnit is
PORT(Input: IN std_logic_vector( 15 downto 0);
    InputW: IN std_logic_vector( 15 downto 0);
    Output: OUT std_logic_vector( 15 downto 0);
    Clock: IN std_logic;
    Done,Flag: OUT std_logic;
    Instruction: IN std_logic_vector( 24 downto 0));
end entity;

architecture behaviour of ExecutionUnit is

component RegisterDataPath is
PORT(
Clock,Reset,Load:IN std_logic;
Input: IN std_logic_vector(30 downto 0);
Output : OUT std_logic_vector(30 downto 0));
end component;


component  Moltiplicatore is
PORT(
A,B: IN std_logic_vector(15 downto 0);
Clock,En_X2: IN std_logic;
Output : OUT std_logic_vector(31 downto 0));
end component;



component Sum is
PORT(
A,B: IN std_logic_vector(30 downto 0);
AddSubA,AddSubB: IN std_logic;
Output : OUT std_logic_vector(30 downto 0));
end component;

component RegisterFile is
PORT(
```

```
Addr0:IN std_logic_vector(1 downto 0);
Addr1:IN std_logic;
Addr2,Addr3: IN std_logic_vector(1 downto 0);
Addr4: std_logic;
DataOut0,DataOut1,DataOut2: OUT std_logic_vector( 15 downto 0);
DataIn3,DataIn4: IN std_logic_vector(15 downto 0);
En_PORT3,En_PORT4,Clock,Reset: IN std_logic);
end component;

component Mux2to1 is
GENERIC ( N: integer:=8);
PORT(A,B: std_logic_vector((N-1) downto 0);
Sel:IN std_logic;
Output :OUT std_logic_vector((N-1) downto 0));
end component;

component Mux4to1 is
GENERIC ( N: integer:=16);
PORT( A,B,C,D:IN std_logic_vector((N-1) downto 0);
Sel:IN std_logic_vector(1 downto 0);
Output :OUT std_logic_vector((N-1) downto 0));
end component;

component LatchSR is
PORT(
set,reset:IN std_logic;
output : BUFFER std_logic);
end component;

component Arrotondatore is
PORT(
Input:IN std_logic_vector ( 30 downto 0);
Output : OUT std_logic_vector (15 downto 0));
end component;

Signal Arrotondatore_Out,Data_Port3,OPA,OPB,Data_Port2,Zero16: std_logic_vector( 15 downto 0);
Signal M_Out: std_logic_Vector( 31 downto 0);
Signal Reg_M_Out,Mux_Sum_Out,Data_PORT2_ex,Reg_S2_S4_Out,Reg_S_Out,Sum_Out,Zero31: std_logic_vector( 30
downto 0);
Signal Sel_Mux_Sum,Addr0,Addr2,Addr3: std_logic_vector( 1 downto 0);
Signal
Sel_Mux_Out,Sel_Mux_Port3,LD_M,Res_Reg,LD_S,LD_S2_S4,EN_X2,ADD_SUB_B,ADD_SUB_A,EN_PORT3,EN_PORT4,A
ddr1,Addr4,Set_Flag,Res_Flag,
Res_Done,Set_Done: std_logic;

BEGIN
Addr0<=Instruction(24 downto 23);
Addr1<=Instruction(22);
Addr2<=Instruction(21 downto 20);
Addr3<=Instruction(19 downto 18);
En_PORT3<=Instruction(17);
Addr4<=Instruction(16);
En_PORT4<=Instruction(15);
Sel_Mux_PORT3<=Instruction(14);
Sel_Mux_Sum<=Instruction(13 downto 12);
Sel_Mux_Out<=Instruction(11);
En_X2<=Instruction(10);
ADD_SUB_A<=Instruction(9);
```

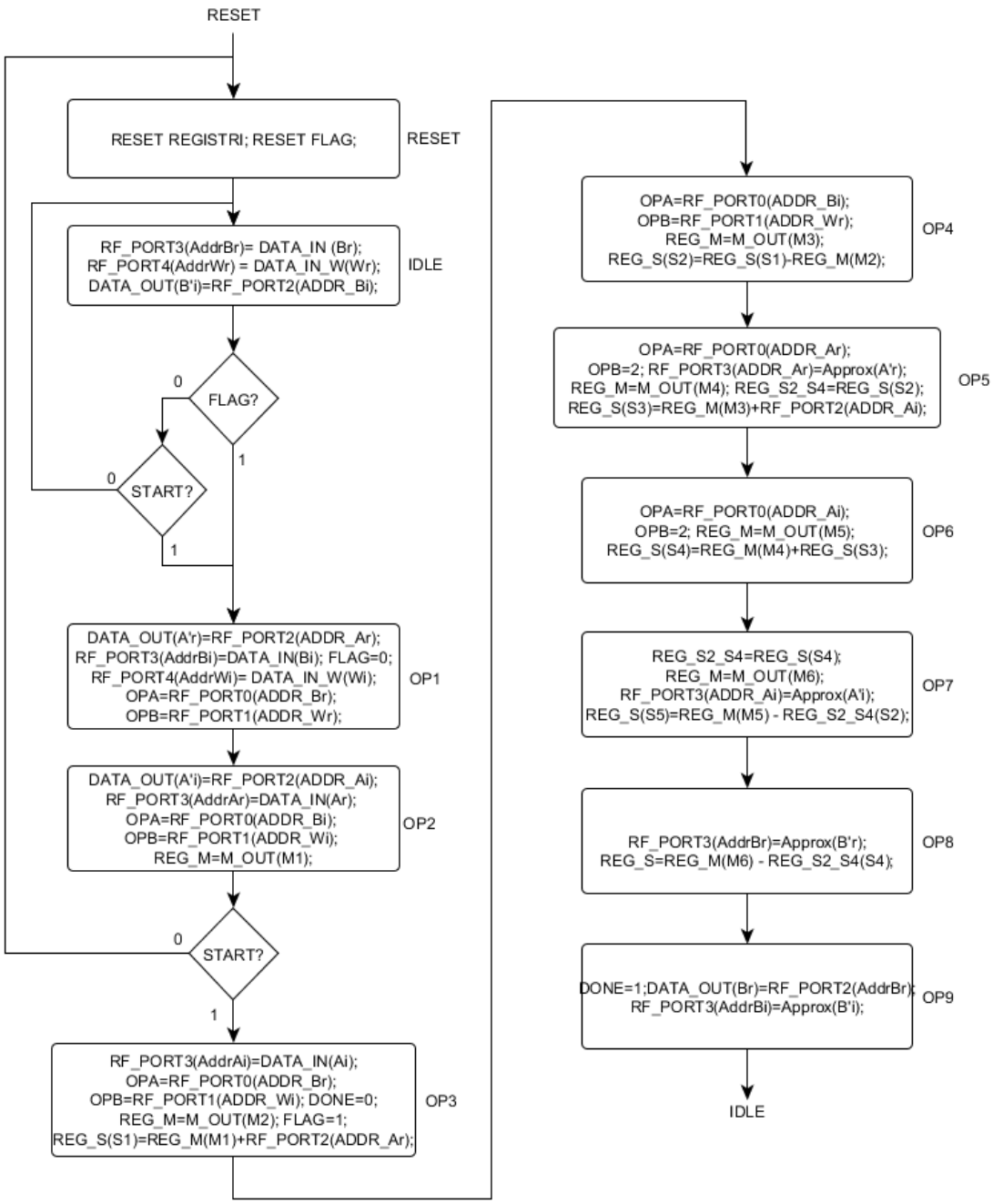
```
ADD_SUB_B<=Instruction(8);
Res_Reg<=Instruction(7);
LD_M<=Instruction(6);
LD_S<=Instruction(5);
LD_S2_S4<=Instruction(4);
Set_Flag<=Instruction(3);
Res_Flag<=Instruction(2);
Set_Done<=Instruction(1);
Res_Done<=Instruction(0);
Data_PORT2_ex(14 downto 0)<=(Others=>'0');
Data_PORT2_ex(30 downto 15)<=Data_PORT2;
Zero16<=(others=>'0');
Zero31<=(others=>'0');

MUX_PORT3: Mux2to1 GENERIC MAP(16) PORT MAP(Input,Arrotondatore_Out,Sel_Mux_PORT3,Data_PORT3);
MUX_SUM: Mux4to1 GENERIC MAP(31) PORT
MAP(Zero32,Data_PORT2_ex,Reg_S2_S4_Out,Reg_S_Out,Sel_Mux_Sum,Mux_Sum_Out);
MUX_OUT: Mux2to1 GENERIC MAP(16) PORT MAP(Zero16,Data_PORT2,Sel_Mux_Out,Output);
REG_M: RegisterDataPath PORT MAP(Clock,Res_Reg,LD_M,M_Out(31 downto 1),Reg_M_Out);
REG_S: RegisterDataPath PORT MAP(Clock,Res_Reg,LD_S,Sum_Out,Reg_S_Out);
REG_S2_S4: RegisterDataPath PORT MAP(Clock,Res_Reg,LD_S2_S4,Reg_S_Out,Reg_S2_S4_Out);
MOLTIPLICATORE_PIPELANED: Moltiplicatore PORT MAP(OPA,OPB,Clock,EN_X2,M_Out);
SOMMATORE: Sum PORT MAP(Reg_M_Out,Mux_Sum_Out,ADD_SUB_A,ADD_SUB_B,Sum_Out);
RF: RegisterFile PORT
MAP(Addr0,Addr1,Addr2,Addr3,Addr4,OPA,OPB,Data_PORT2,Data_PORT3,InputW,En_PORT3,En_PORT4,Clock,Res_Re
g);
ARROTONDA: Arrotondatore PORT MAP(Reg_S_Out,Arrotondatore_Out);
LATCHFLAG: LatchSR PORT MAP( Set_Flag,Res_Flag,Flag);
LATCHDONE: LatchSR PORT MAP( Set_Done,Res_Done,Done);
end architecture;
```

3.6 Control Unit

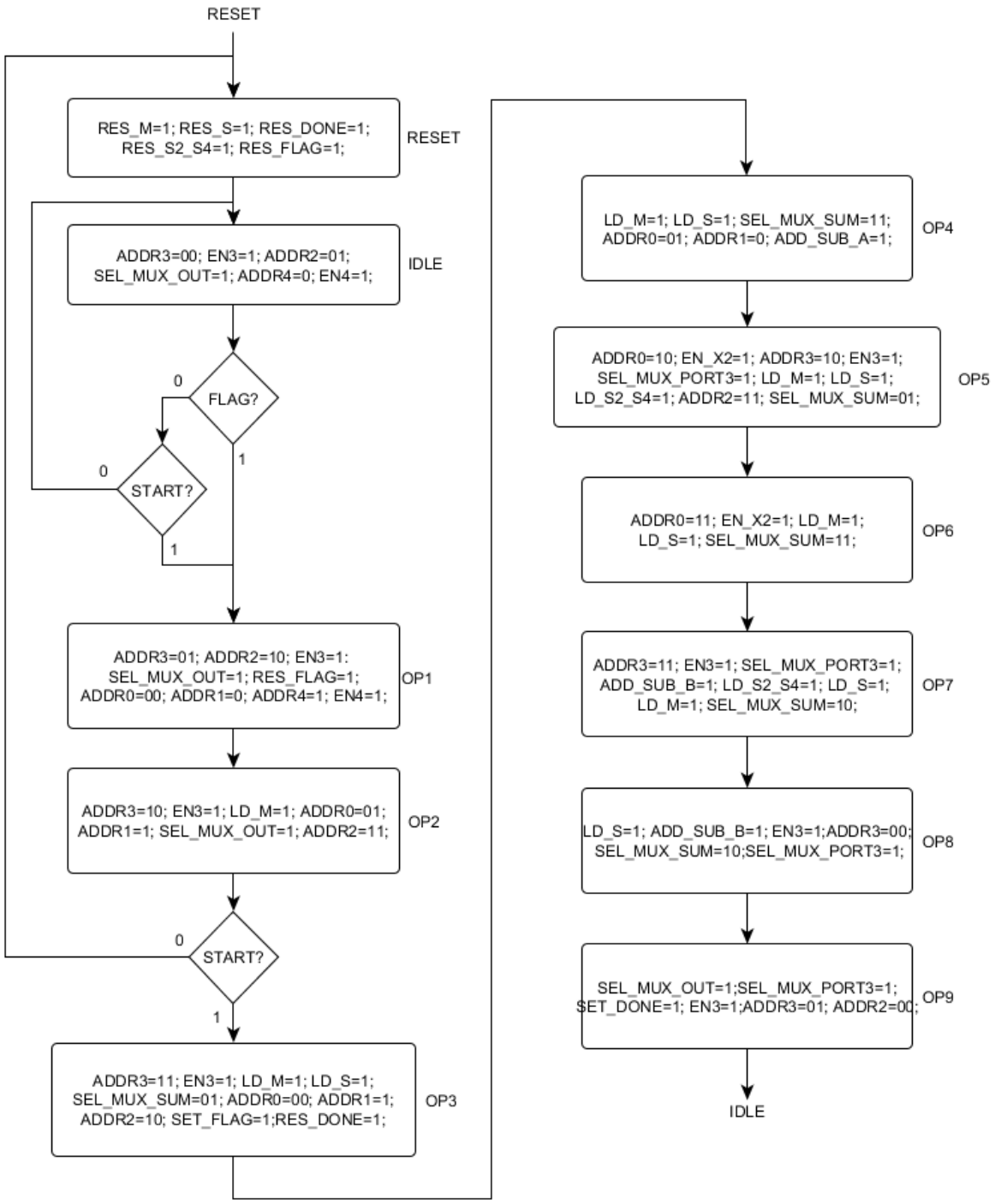
La Control Unit, come definito dalle specifiche, deve essere realizzata utilizzando un indirizzamento esplicito. Inoltre, ogni operazione deve essere eseguita in un colpo di clock. Per poterla progettare e ottimizzare è necessario definire per ogni stato il valore di tutti i segnali di controllo e l'eventuale influenza di segnali di Status provenienti dall'esterno o dalla Execution Unit. Nel nostro caso l'Execution Unit non fornisce alcun segnale. Gli unici segnali che condizionano l'elaborazione sono **Start** e **Flag**. Di seguito sono riportati i diagrammi ***Asm-Data*** e ***Asm-Control*** eseguiti dalla Control Unit.

3.6.1 Asm-Data e Asm-Control

Asm-Data



Asm-Control


RESET
RES_M=1; RES_S=1; RES_DONE=1;
RES_S2_S4=1; RES_FLAG=1;
RESET
ADDR3=00; EN3=1; ADDR2=01;
SEL_MUX_OUT=1; ADDR4=0; EN4=1;
IDLE
FLAG?
0
1
START?
0
1
ADDR3=01; ADDR2=10; EN3=1:
SEL_MUX_OUT=1; RES_FLAG=1;
ADDR0=00; ADDR1=0; ADDR4=1; EN4=1;
OP1
ADDR3=10; EN3=1; LD_M=1; ADDR0=01;
ADDR1=1; SEL_MUX_OUT=1; ADDR2=11;
OP2
START?
0
1
ADDR3=11; EN3=1; LD_M=1; LD_S=1;
SEL_MUX_SUM=01; ADDR0=00; ADDR1=1;
ADDR2=10; SET_FLAG=1;RES_DONE=1;
OP3
LD_M=1; LD_S=1; SEL_MUX_SUM=11;
ADDR0=01; ADDR1=0; ADD_SUB_A=1;
OP4
ADDR0=10; EN_X2=1; ADDR3=10; EN3=1;
SEL_MUX_PORT3=1; LD_M=1; LD_S=1;
LD_S2_S4=1; ADDR2=11; SEL_MUX_SUM=01;
OP5
ADDR0=11; EN_X2=1; LD_M=1;
LD_S=1; SEL_MUX_SUM=11;
OP6
ADDR3=11; EN3=1; SEL_MUX_PORT3=1;
ADD_SUB_B=1; LD_S2_S4=1; LD_S=1;
LD_M=1; SEL_MUX_SUM=10;
OP7
LD_S=1; ADD_SUB_B=1; EN3=1;ADDR3=00;
SEL_MUX_SUM=10;SEL_MUX_PORT3=1;
OP8
SEL_MUX_OUT=1;SEL_MUX_PORT3=1;
SET_DONE=1; EN3=1;ADDR3=01; ADDR2=00;
OP9
IDLE

Il diagramma ASM Control definisce per ogni stato solamente i segnali di controllo il cui valore cambia rispetto allo stato di Default definito di seguito:

ADDR0=”00”;
ADDR1=0;
ADDR2=”00”;
ADDR3=”00”;
EN3=0;
SEL_MUX_PORT3=0;
EN_X2=0;
RES_M=0;
LD_M=0;
SEL_MUX_SUM=”00”;
ADD_SUB_A=0;
ADD_SUB_B=0;
RES_S=0;
LD_S=0;
RES_S2_S4=0;
LD_S2_S4=0;
SET_FLAG=0;
RES_FLAG=0;
SET_DONE=0;
RES_DONE=0;
SEL_MUX_OUT=00;

Nota: con il segnale RES_REG si intendono tutti i segnali di Reset dei blocchi all’interno della Execution Unit.

3.6.2 Descrizione degli Stati

Di seguito è riportata la descrizione di ogni stato ed il funzionamento della macchina nelle modalità Singola, Continua e Sequenza.

- *RESET: lo stato di Reset permette alla Butterfly di azzerare il contenuto di ogni registro interno e del **Register_File**. Vengono inoltre portate a livello logico zero le uscite **Done** e **Flag**;*

- *IDLE: nello stato di IDLE viene:*

  1. *Predisposto il caricamento del dato B_r e del coefficiente W_r all’interno del **Register_File;***
  2. *Letto dal **Register_File** il risultato B_i' fornendolo in uscita attraverso **Data_Out;***

- *OP1: nello stato OP1 viene:*

  1. *Predisposto il caricamento del dato B_i e del coefficiente W_i all’interno del **Register_File;***

2. *Mostrato in uscita il risultato A'_r letto dal **Register_File**;*
3. *Forzata a zero l'uscita **Flag** e assegnato il valore gli operandi OPA e OPB attraverso la lettura di B_r e W_r dal **Register_File** per l'inizio dell'operazione M1;*

- *OP2: nello stato OP2 viene:*

1. *Predisposto il caricamento del dato A_r all'interno del **Register_File**;*
2. *Mostrato in uscita il risultato A'_i letto dal **Register_File;***
3. *Assegnato il valore gli operandi OPA e OPB attraverso la lettura di B_i e W_i dal **Register_File** per l'inizio dell'operazione M e predisposto l'aggiornamento del registro **Reg_M** con il risultato di M1;*

- *OP3: nello stato OP3 viene:*

1. *Predisposto il caricamento del dato A_i all'interno del **Register_File**;*
2. *Assegnato il valore gli operandi OPA e OPB attraverso la lettura di B_r e W_i dal **Register_File** per l'inizio dell'operazione M3;*
3. *Eseguita la somma S1, predisposto l'aggiornamento di **Reg_M** con il risultato di M2 e di **Reg_S** con il risultato di S1;*
4. *Forzato a zero il segnale di **Done**, forzato a uno il segnale **Flag**;*

- *OP4: nello stato OP4 viene:*

1. *Assegnato il valore gli operandi OPA e OPB attraverso la lettura di B_i e W_r dal **Register_File** per l'inizio dell'operazione M4;*
2. *Eseguita la differenza S2 e predisposto l'aggiornamento di **Reg_M** con il risultato di M3 e di **Reg_S** con il risultato di S2;*

- *OP5: nello stato OP5 viene:*

1. *Assegnato il valore gli operandi OPA e OPB attraverso la lettura di A_r dal **Register_File** per l'inizio dell'operazione M5;*
2. *Eseguita la differenza S3 e predisposto l'aggiornamento di **Reg_M** con il risultato di M4, di **Reg_S** con il risultato di S3 e di **Reg_S2_S4** con il risultato S2 contenuto in **Reg_S;***
3. *Predisposta la scrittura nel **Register_File** del risultato A'_r arrotondato;*

- *OP6: nello stato OP6 viene:*
1. *Assegnato il valore gli operandi OPA e OPB attraverso la lettura di A_i dal **Register_File** per l'inizio dell'operazione M6;*

2. *Eseguita la somma S4 e predisposto l'aggiornamento di **Reg_M** con il risultato di M5 e di **Reg_S** con il risultato di S4;*

- *OP7: nello stato OP7 viene:*

  1. *Eseguita la somma S5 e predisposta la scrittura nel **Register_File** del risultato A'_i arrotondato;*
  2. *Predisposto l'aggiornamento di **Reg_M** con il risultato di M6, di **Reg_S** con il risultato di S5 e di **Reg_S2_S4** con il risultato S4 contenuto in **Reg_S**;*

- *OP8: nello stato OP8 viene:*

  1. *Eseguita la somma S6 e predisposto l'aggiornamento di **Reg_S** con il risultato di S6;*
  2. *Predisposta la scrittura nel **Register_File** del risultato B'_r arrotondato;*

- *OP9: nello stato OP9 viene:*

  1. *Letto dal **Register_File** il risultato B'_r fornendolo in uscita attraverso **Data_Out;***
  2. *Asserito il segnale di **Done** e predisposta la scrittura nel **Register_File** del risultato B'_i arrotondato;*

Modalità Singola, Continua e Sequenza:

Dopo il segnale di Reset la Butterfly esegue l'operazione *RESET* e si porta nello stato *IDLE* con **Flag** e **Done** a 0. Se dall'esterno, viene fornito il segnale di **Start** essa passa allo stato OP1 campionando il dato presente sulla porta **Data_IN** ed il coefficiente sulla porta **Data_IN_W**. Per i successivi tre colpi di clock devono essere forniti gli operandi sulle due porte di ingresso con segnale di Start attivo. I dati di uscita forniti inizialmente negli stati *IDLE, OP1 e OP2* sono nulli e non devono essere considerati validi in quanto il segnale **Done** è 0. Successivamente vengono eseguite tutte le operazioni giungendo allo stato *OP9*. Qui, viene attivato il segnale Done fornendo in uscita contemporaneamente il dato B'_r. In seguito, la Butterfly, si porta nello stato *IDLE* con **Done e Flag a** 1. Viene quindi fornito in uscita il dato B'_i. Essendo **Flag** uguale a 1, il successivo colpo di Clock permette di raggiungere lo stato *OP1* fornendo in uscita il risultato A'_r e azzerando **Flag**. Viene successivamente eseguito lo stato *OP2* fornendo A'_i. Se il segnale di Start è attivo, la macchina assume che i nuovi operandi e coefficienti caricati durante *IDLE* e *OP1* siano validi incominciando un nuovo ciclo di elaborazione, passando in OP3. In caso contrario ritorna in *RESET* e poi in *IDLE* attendendo un nuovo segnale di **Start**. Il segnale di **Flag** viene quindi utilizzato dalla Butterfly per capire se dopo lo stato di *IDLE* deve proseguire in *OP1*, per terminare un ciclo di elaborazione (**Flag** = 1) oppure se deve incominciare un ciclo (**Flag**=0 e **Start**=1). In questo modo possono essere eseguite entrambe le modalità richieste. Il Timing per l'utente viene mostrato nei successivi paragrafi in modo dettagliato.

3.6.3 Istruzioni

Contenuto delle Istruzioni

Dopo aver definito per ogni stato, i segnali di controllo rispettivi e lo stato successivo, è necessario codificare ogni combinazione con un'istruzione. Ad ogni istruzione viene successivamente assegnato un numero univoco che ne rappresenta l'indirizzo all'interno della memoria codice. L'algoritmo, presentato precedentemente, mostra come gli stati IDLE e OP2 a differenza degli altri possano evolvere in due possibili stati successivi, in funzione dei segnali **Start** e **Flag**. Osservando l'Asm Data si nota che è possibile definire un unico segnale **Status** il cui valore è dato dall'operazione OR tra **Start** e **Flag**. In questo modo, scegliendo un indirizzamento della memoria codice su quattro bit, è possibile coprire tutti gli stati dell'algoritmo compresi eventuali salti condizionati dal segnale **Status**. Ogni istruzione quindi contiene al suo interno i seguenti campi:

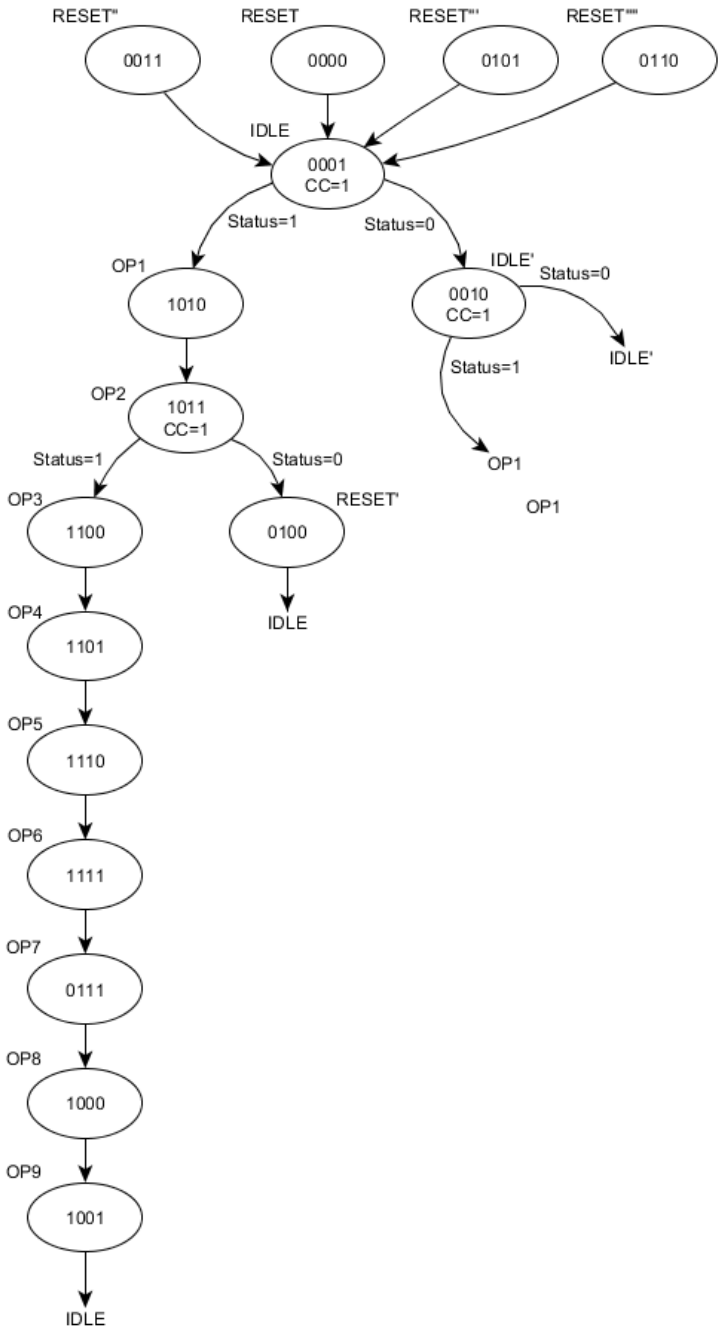
CC	MSB	NextAddress	Execution Unit Signals Control

- *CC*: Condition Code. Esso permette di definire, quando è attivo, se l'istruzione corrente prevede un salto condizionato in due possibili stati successivi (attivo in IDLE e OP2);
- *MSB*: Most Significative Bit. Esso corrisponde al bit più significativo dell'indirizzo della prossima istruzione da eseguire. Viene considerato quando CC è disattivo;
- *NextAddress*: campo che contiene i tre bit meno significativi dell'indirizzo della prossima istruzione;
- *Execution Unit Signals Control*: campo che contiene la combinazione definita dal Asm-Control di tutti i segnali di controllo dell'Execution Unit.

Indirizzamento delle Istruzioni

Scegliendo una codifica dell'indirizzo su quattro bit si può eseguire tutto l'algoritmo. Di seguito è mostrato un diagramma, dove all'interno di ogni stato, è contenuto il valore dell'indirizzo che lo rappresenta e lo stato di CC. L'indirizzo di ogni istruzione è ottenuta come:

- CC=0 -> Indirizzo = (MSB + NextAddress);
- CC=1 -> Indirizzo = (Status + NextAddress);



Come si può notare, scegliendo opportunamente il numero binario dell'indirizzo, si può creare uno schema dove, negli stati *OP2* e *IDLE* (CC = 1), il bit più significativo dell'indirizzo successivo è definito dal valore del segnale Status. In questo modo si ottimizza notevolmente la scelta del salto condizionato da compiere. In tutti gli altri stati in cui CC = 0 il bit più significativo dell'indirizzo successivo è definito univocamente da MSB. Questa scelta porta a definire diversi indirizzi per gli stati *IDLE, OP2 e RESET* in modo da garantire tutti i possibili salti condizionati presenti nell'algoritmo. Tuttavia, ciò non comporta un aumento nel parallelismo a 4 bit dell'indirizzo della memoria codice in quanto il numero di istruzioni complessivo è 16. Si ottiene quindi la seguente tabella degli stati.

ADDRESS	STATE
0000	RESET
0001	IDLE
0010	IDLE'
0011	RESET''
0100	RESET'
0101	RESET'''
0110	RESET''''
0111	OP7
1000	OP8
1001	OP9
1010	OP1
1011	OP2
1100	OP3
1101	OP4
1110	OP5
1111	OP6

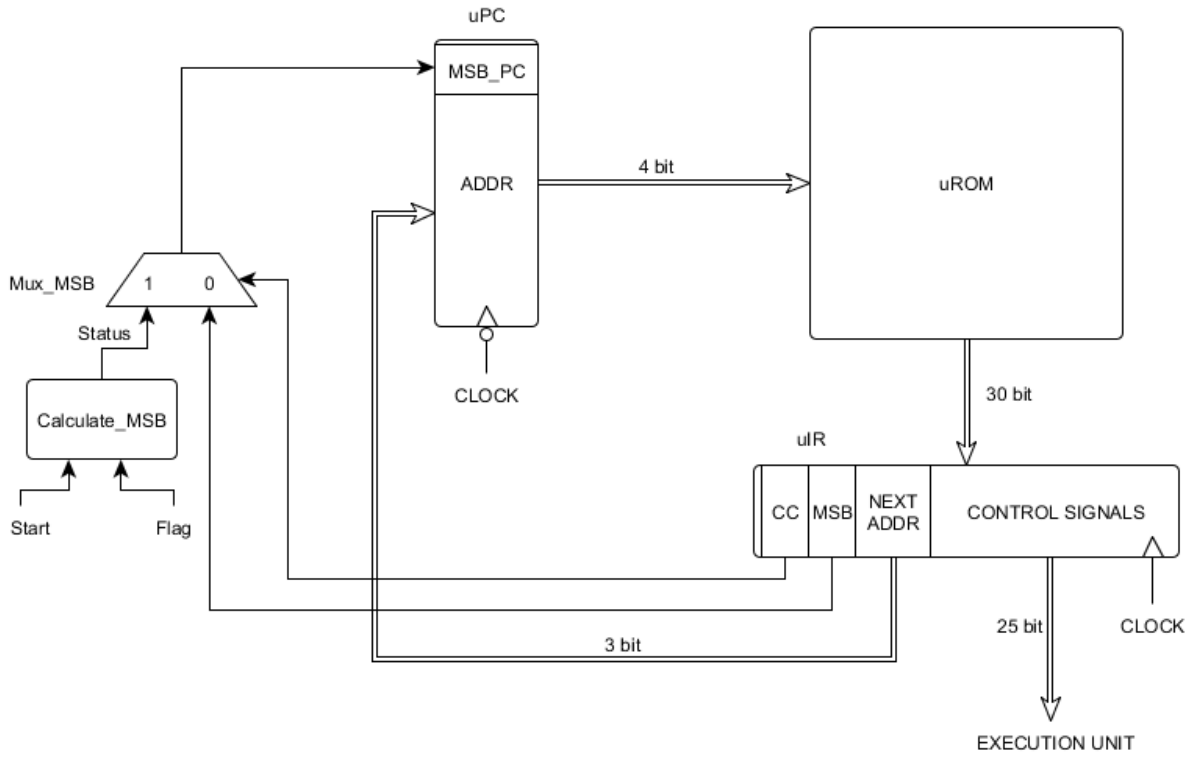
Dopo aver definito per ogni stato un indirizzo che la codifica e una combinazione dei segnali di controllo è possibile definire una tabella il cui contenuto corrisponde alla memoria codice del Processing Element.

STATO	ADDRESS	CC	MSB	NEXTADDRESS	ADDR0	ADDR1	ADDR2	ADDR3	EN3	ADDR4	EN4	SEL_MUX_PORT3	SEL_MUX_SUM	SEL_MUX_OUT	EN_X2	ADD_SUB_A	ADD_SUB_B	RES_REG	LD_M	LD_S	LD_S2_S4	SET_FLAG	RES_FLAG	SET_DONE	RES_DONE
RESET	0000	0	0	001	00	0	00	00	0	0	0	0	00	0	0	0	0	1	0	0	0	0	1	0	1
IDLE	0001	1	0	010	00	0	01	00	1	0	1	0	00	1	0	0	0	0	0	0	0	0	0	0	0
IDLE'	0010	1	0	010	00	0	01	00	1	0	1	0	00	1	0	0	0	0	0	0	0	0	0	0	0
RESET''	0011	0	0	001	00	0	00	00	0	0	0	0	00	0	0	0	0	1	0	0	0	0	1	0	1
RESET'	0100	0	0	001	00	0	00	00	0	0	0	0	00	0	0	0	0	1	0	0	0	0	1	0	1
RESET'''	0101	0	0	001	00	0	00	00	0	0	0	0	00	0	0	0	0	1	0	0	0	0	1	0	1
RESET''''	0110	0	0	001	00	0	00	00	0	0	0	0	00	0	0	0	0	1	0	0	0	0	1	0	1
OP7	0111	0	1	000	00	0	00	11	1	0	0	1	10	0	0	0	1	0	1	1	1	0	0	0	0
OP8	1000	0	1	001	00	0	00	00	1	0	0	1	10	0	0	0	1	0	0	1	0	0	0	0	0
OP9	1001	0	0	001	00	0	00	01	1	0	0	1	00	1	0	0	0	0	0	0	0	0	0	1	0
OP1	1010	0	1	011	00	0	10	01	1	1	1	0	00	1	0	0	0	0	0	0	0	0	1	0	0
OP2	1011	1	0	100	01	1	11	10	1	0	0	0	00	1	0	0	0	0	1	0	0	0	0	0	0
OP3	1100	0	1	101	00	1	10	11	1	0	0	0	01	0	0	0	0	0	1	1	0	1	0	0	1
OP4	1101	0	1	110	01	0	00	00	0	0	0	0	11	0	0	1	0	0	1	1	0	0	0	0	0
OP5	1110	0	1	111	10	0	11	10	1	0	0	1	01	0	1	0	0	0	1	1	1	0	0	0	0
OP6	1111	0	0	111	11	0	00	00	0	0	0	0	11	0	1	0	0	0	1	1	0	0	0	0	0

Osservando la tabella si nota che per ogni istruzione sono presenti tutti i segnali di controllo dell'Execution Unit oltre ai campi CC, MSB e NextAddress. Per ridurre al minimo il parallelismo di ogni istruzione è stato scelto di unire tutti i segnali di Reset dei blocchi in uno unico, chiamato Res_Reg. Si ottiene quindi un parallelismo istruzioni di 30 bit di cui 25 verso l'Execution Unit e 5 utilizzati internamente nella Control Unit.

3.6.4 Architettura della Control Unit

La Control Unit progettata, è mostrata nell'immagine sottostante.



Come si può osservare sono presenti i blocchi µIR, µPC e µROM più un insieme di componenti utili per determinare durante un salto condizionato il bit più significativo del indirizzo dell'istruzione successiva.

µPC:

Il registro **µPC** memorizza l'indirizzo dell'istruzione successiva. Esso è attivo sul fronte di discesa del clock ed è composto dai campi MSB_PC e ADDR. Il primo viene determinato in funzione del stato di CC, MSB, Start e Flag. Il secondo invece viene fornito dall'istruzione corrente. Il suo parallelismo è 4 bit.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity RegisterSequencer is
```

```
GENERIC ( N: integer:=28;
          EDGE: std_logic:='1');
PORT(
Clock,Reset:IN std_logic;
Input: IN std_logic_vector((N-1) downto 0);
Output : OUT std_logic_vector((N-1) downto 0));

end entity;

architecture behaviour of RegisterSequencer is

BEGIN
Process(Reset,Clock)
BEGIN
if(Reset='1')then
Output<=(others=>'0');
elsif(Clock'EVENT and Clock=EDGE)then
Output<=Input;
end if;
end process;
end architecture;
```

µIR:

Il registro **µIR** memorizza il contenuto dell'istruzione attuale. Esso è attivo sul fronte positivo del clock. Il suo parallelismo è 30 bit.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity RegisterSequencer is
GENERIC ( N: integer:=28;
          EDGE: std_logic:='1');
PORT(
Clock,Reset:IN std_logic;
Input: IN std_logic_vector((N-1) downto 0);
Output : OUT std_logic_vector((N-1) downto 0));

end entity;

architecture behaviour of RegisterSequencer is

BEGIN
Process(Reset,Clock)
BEGIN
if(Reset='1')then
Output<=(others=>'0');
elsif(Clock'EVENT and Clock=EDGE)then
Output<=Input;
end if;
end process;
end architecture;
```

µROM:

La **µROM** contiene tutte le istruzioni che possono essere eseguite dal Processing Element. Ogni istruzione viene selezionata attraverso l'indirizzo di ingresso. Essa contiene sedici istruzioni selezionabili con un indirizzo a 4 bit. Il parallelismo è 30 bit.

***Codice VHDL**:*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity uROM is
PORT(
Address: IN std_logic_vector(3 downto 0);
Output : OUT std_logic_vector(29 downto 0));

end entity;

architecture behaviour of uROM is

type Istruction is array (0 to 15) of std_logic_vector( 29 downto 0);
signal ROM: Istruction:=(

"000010000000000000000010000101", -- RESET
"100100000100101000100000000000", -- IDLE
"100100000100101000100000000000",  -- IDLE'
"000010000000000000000010000101", -- RESET
"000010000000000000000010000101", -- RESET
"000010000000000000000010000101",-- RESET
"000010000000000000000010000101",-- RESET
"010000000011100110000101110000",-- OP7
"010010000000100110000100100000",-- OP8
"000010000001100100100000000010", -- OP9
"010110001001111000100000000100", -- OP1
"101000111110100000100001000000",-- OP2
"011010011011100001000001101001", -- OP3
"011100100000000011001001100000",-- OP4
"011111001110100101010001110000",-- OP5
"001111100000000011010001100000"-- OP6
);

BEGIN

Process(Address,ROM)
BEGIN
Output<=ROM(to_integer(unsigned(Address)));

end process;

end architecture;
```

Mux_MSB:

Il componente **Mux_MSB** è un multiplexer a due vie che viene comandato da CC e permette di leggere o ignorare il segnale **Status**. Esso presenta le seguenti porte di I/O e segnali di controllo:

- *SEL*: segnale controllato da CC che permette di scegliere quale dei due ingressi viene collegato all'uscita del multiplexer. Quando CC=1 il bit il valore di uscita è pari a **Status** in caso contrario è pari al valore *MSB*;
- *A, B*: ingressi dei segnali **Status** e *MSB*;
- *Output*: segnale di uscita;

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;


entity Mux2to1_1bit is
PORT(A,B: IN std_logic;
Sel:IN std_logic;
Output :OUT std_logic);
end entity;

architecture behaviour of Mux2to1_1bit is

BEGIN
Output<= A when (Sel='0') else B;

end architecture;
```

Calculate_MSB:

Il blocco **Calculate_MSB** permette di generare in funzione dello stato di **Flag** e **Start** un segnale di uscita che corrisponde al segnale **Status**. L'operazione eseguita è la OR tra **Flag** e **Start**.

Codice VHDL:

```
Status<= Flag OR Start;
```

3.6.5 Codice VHDL Control Unit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity ControlUnit is
PORT(Clock,Reset,Flag,Start: IN std_logic;
    Instruction: OUT std_logic_vector( 24 downto 0));
end entity;
```

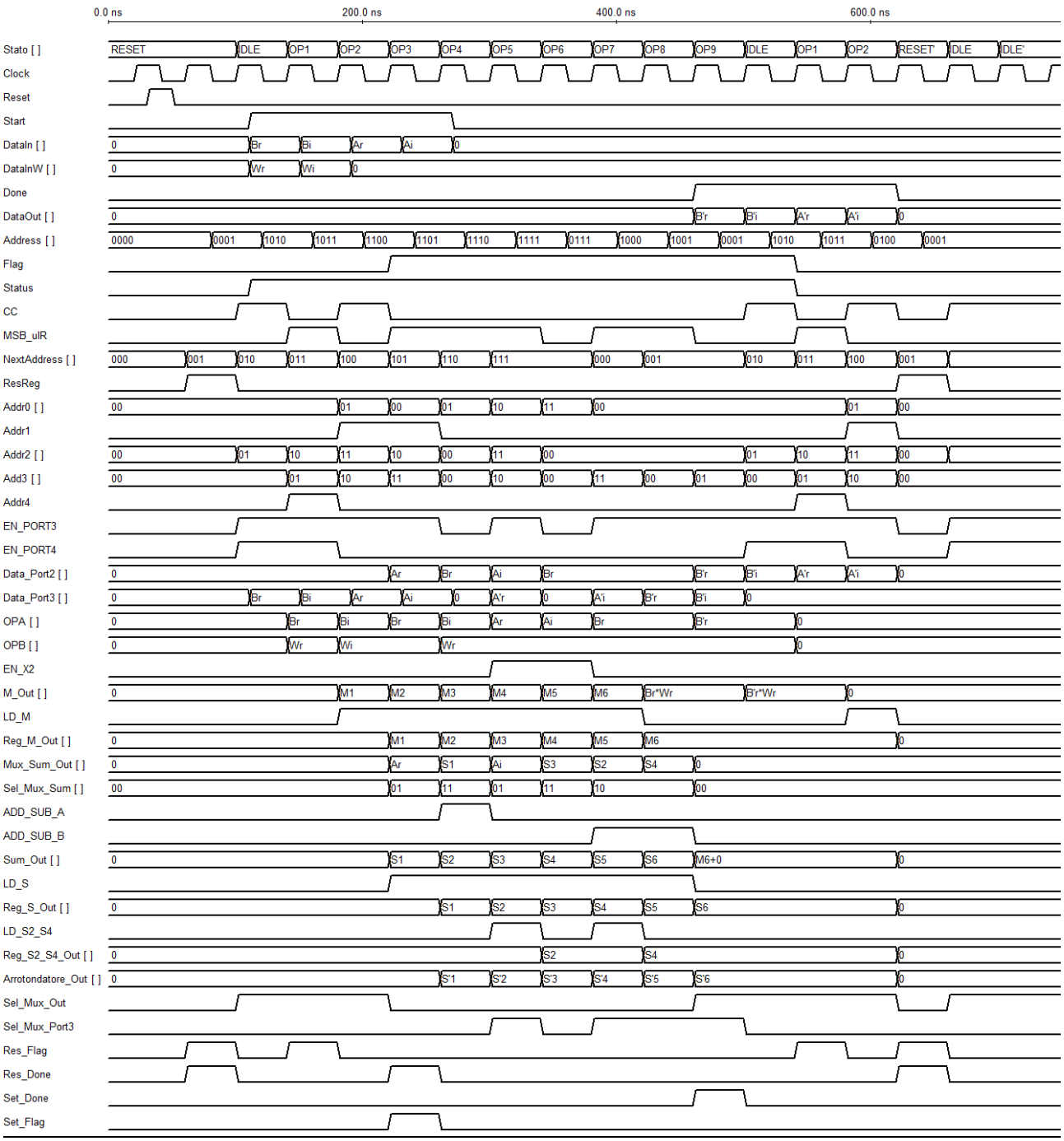
```
architecture behaviour of ControlUnit is

component Mux2to1_1bit is
PORT(A,B: IN std_logic;
Sel:IN std_logic;
Output :OUT std_logic);
end component;

component RegisterSequencer is
GENERIC ( N: integer:=28; EDGE: std_logic:='1');
PORT(
Clock,Reset:IN std_logic;
Input: IN std_logic_vector((N-1) downto 0);
Output : OUT std_logic_vector((N-1) downto 0));

end component;

component uROM is
PORT(
Address: IN std_logic_vector(3 downto 0);
Output : OUT std_logic_vector(29 downto 0));

end component;

Signal Status,CC,MSB_uIR,MSB_uPC: std_logic;
Signal NextAddress: std_logic_vector( 2 downto 0);
Signal Address: std_logic_vector( 3 downto 0);
Signal uIR_In,uIR_Out: std_logic_vector ( 29 downto 0);
Signal uPC_In: std_logic_vector ( 3 downto 0);

BEGIN

Status<= Flag OR Start;

uPC_In<=MSB_uPC & NextAddress;

CC<=uIR_Out(29);
MSB_uIR<=uIR_Out(28);
NextAddress<=uIR_Out(27 downto 25);
Instruction<=uIR_Out(24 downto 0);


MUX: Mux2to1_1bit PORT MAP(MSB_uIR,Status,CC,MSB_uPC);
uPC: RegisterSequencer GENERIC MAP(4,'0') PORT MAP(Clock,Reset, uPC_In , Address);
u_ROM: uROM PORT MAP(Address,uIR_In);
uIR: RegisterSequencer GENERIC MAP(30,'1') PORT MAP(Clock,Reset,uIR_In,uIR_Out);


end architecture;
```

3.7 Codice VHDL Butterfly

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Butterfly is
PORT(Clock,Reset,Start: IN std_logic;
    Done: OUT std_logic;
    DataIn: IN std_logic_vector( 15 downto 0);
    DataInW: IN std_logic_vector( 15 downto 0);
    DataOut: OUT std_logic_vector( 15 downto 0));

end entity;

architecture behaviour of Butterfly is

Component ControlUnit is
PORT(Clock,Reset,Flag,Start: IN std_logic;
    Instruction: OUT std_logic_vector( 24 downto 0));
end Component;

component ExecutionUnit is
PORT(Input: IN std_logic_vector( 15 downto 0);
    InputW: IN std_logic_vector( 15 downto 0);
    Output: OUT std_logic_vector( 15 downto 0);
    Clock: IN std_logic;
    Done,Flag: OUT std_logic;
    Instruction: IN std_logic_vector( 24 downto 0));
end component;

Signal Flag: std_logic;
Signal Instruction: std_logic_vector( 24 downto 0);


BEGIN

CONTROLUNITBLOCK: ControlUnit PORT MAP(Clock,Reset,Flag,Start,Instruction);
EXECUTIONUNITBLOCK: ExecutionUnit PORT MAP(DataIn,DataInW,DataOut,Clock,Done,Flag,Instruction);


end architecture;
```

3.8 Timing

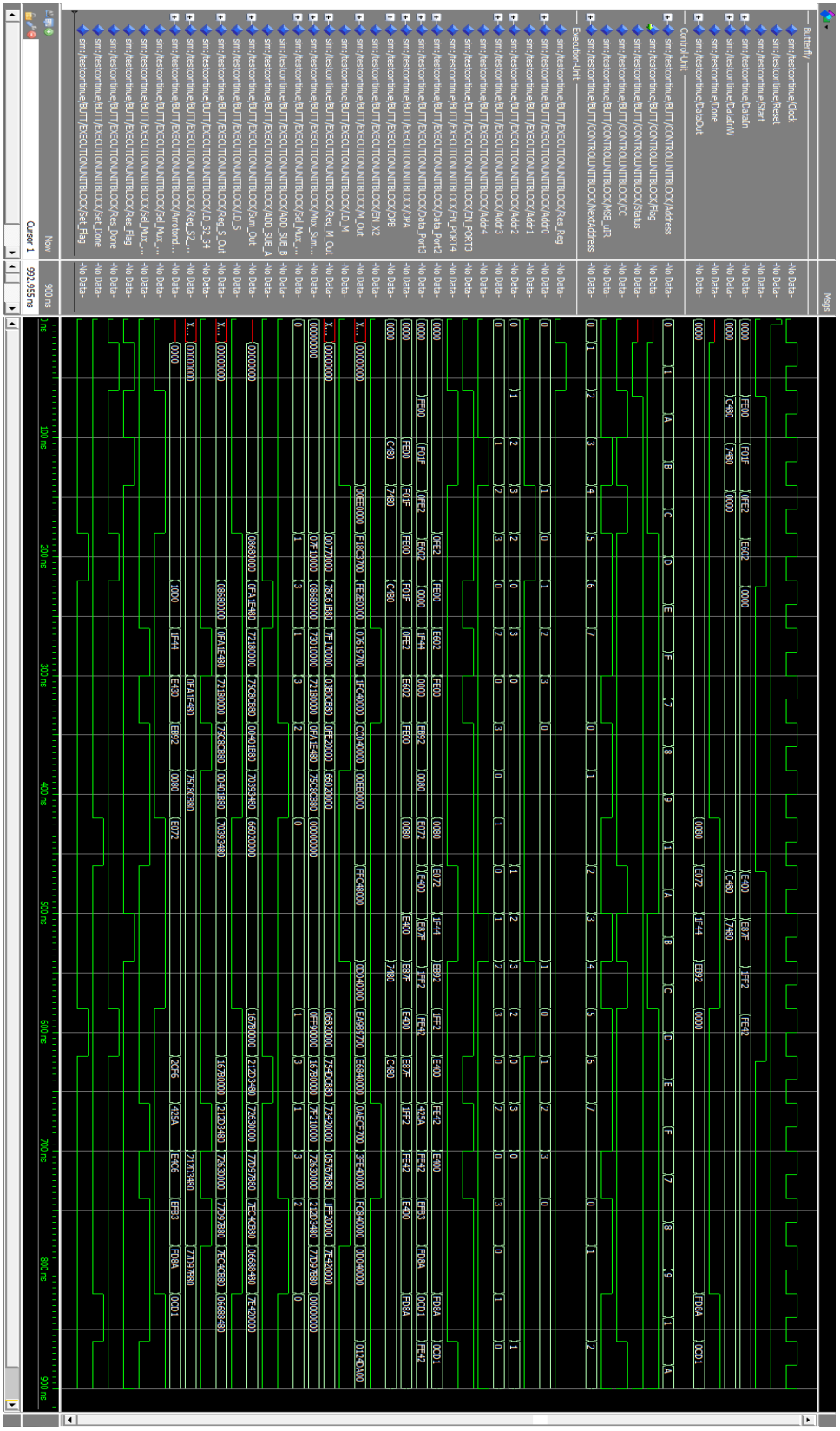
Di seguito è mostrato il Timing della Butterfly durante un ciclo di elaborazione.



4.0 Simulazione

4.1 Timing

Di seguito è mostrato il Timing della Butterfly simulato con il software ModelSim.



4.2 Elaborazione

Per verificare il corretto funzionamento della Butterfly è stato creato un TestBench in VHDL per ogni modalità di funzionamento. Per ognuno, inoltre, sono stati analizzati i risultati ottenuti confrontandoli con quelli forniti da uno Script MATLAB.

4.2.1 Modalità Singola

In modalità singola sono stati considerati due insiemi di operandi inviati in tempi diversi alla stessa Butterfly.

Stream Operandi 1:

Dati:

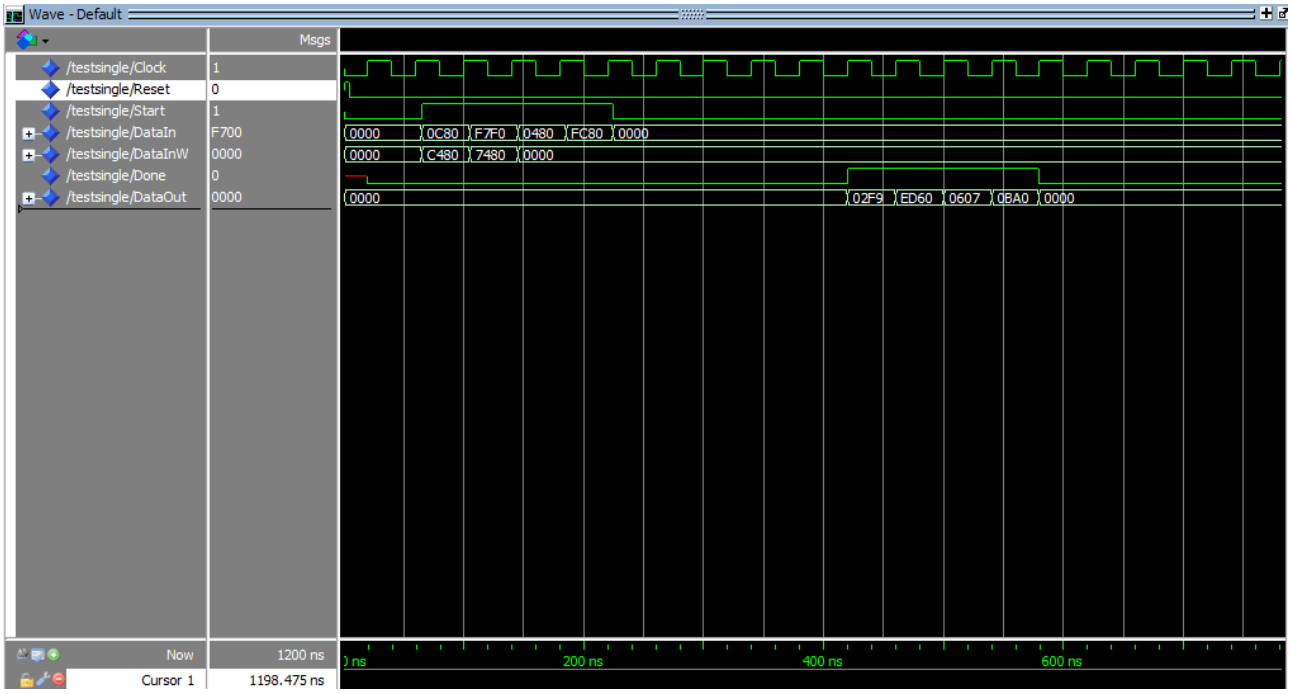
- B_r= “0000110010000000” -> + 0.097656250000000;
- B_i= “1111011111110000” -> - 0.062988281250000;
- A_r= “0000010010000000”-> + 0.035156250000000;
- A_i= “1111110010000000”-> - 0.027343750000000;

Coefficienti:

- W_r=”1100010010000000”-> - 0.464843750000000;
- W_i=”0111010010000000”-> + 0.910156250000000;

Risultati forniti dalla Butterfly:

- B'_r= “0000001011111001” -> + 0.023223876953125;
- B'_i= “1110110101100000” -> - 0.145507812500000;
- A'_r= “0000011000000111”-> + 0.047088623046875;
- A'_i= “0000101110100000”-> + 0.090820312500000;



I risultati ottenuti con lo script MATLAB sono:

Risultati forniti da MATLAB:

- B'_r= + 0.0232219696044922;
- B'_i= - 0.1455059051513670;
- A'_r= + 0.0470905303955078;
- A'_i= + 0.0908184051513672;

Sono stati inoltre calcolati i Residui definiti come differenza tra i risultati della Butterfly e quelli di MATLAB.

Residui:

- $Residuo\ B'_r$= - 1.9073486328125e-06;
- $Residuo\ B'_i$= + 1.9073486328125e-06;
- $Residuo\ A'_r$= + 1.9073486328125e-06;
- $Residuo\ A'_i$= - 1.9073486328125e-06;

Stream Operandi 2:

Dati:

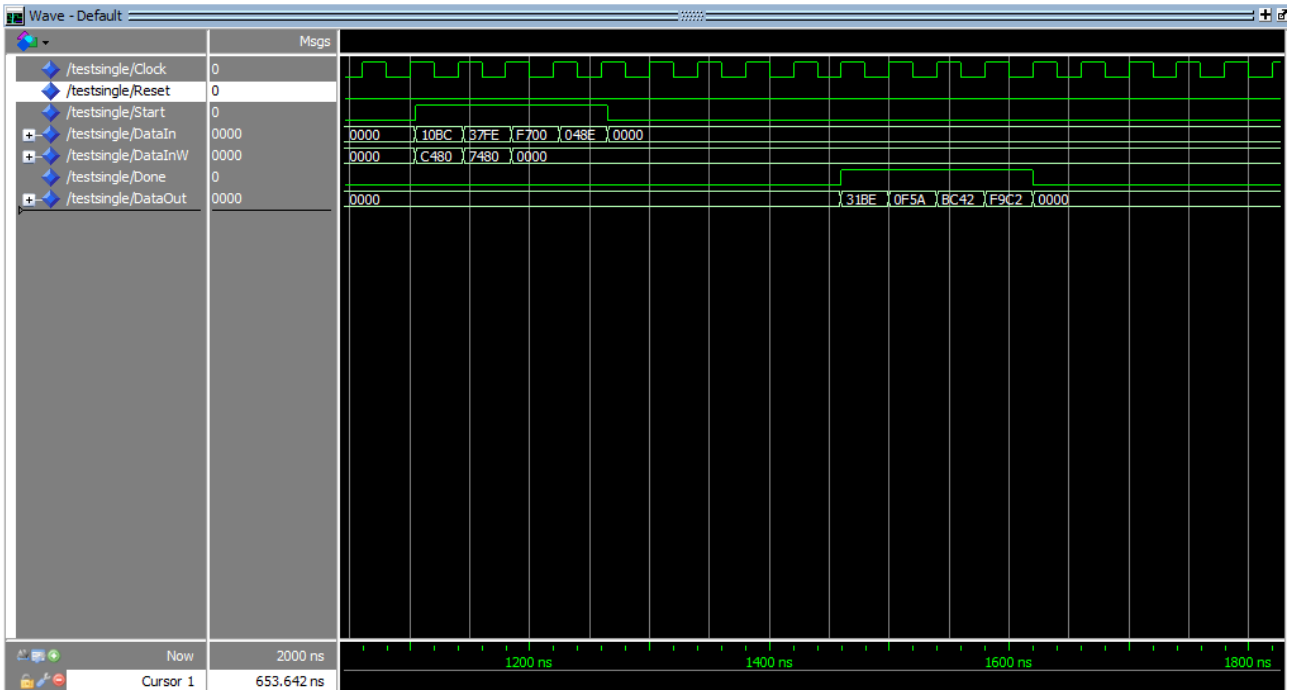
- B_r= “0001000010111100” -> + 0.13073730468750;
- B_i= “0011011111111110” -> + 0.43743896484375;
- A_r= “1111011100000000”-> - 0.07031250000000;
- A_i= “0000010010001110”-> + 0.03558349609375;

Coefficienti:

- W_r=”1100010010000000”-> - 0.464843750000000;
- W_i=”0111010010000000”-> + 0.910156250000000;

Risultati forniti dalla Butterfly:

- B'_r= “0011000110111110” -> + 0.38861083984375;
- B'_i= “0000111101011010” -> + 0.11993408203125;
- A'_r= “1011110001000010”-> - 0.52923583984375;
- A'_i= “1111100111000010”-> - 0.04876708984375;



I risultati ottenuti con lo script MATLAB invece sono:

Risultati forniti da MATLAB:

- B'_r= + 0.388597726821899;
- B'_i= + 0.119932889938354;
- A'_r= - 0.529222726821899;
- A'_i= - 0.0487658977508545;

Sono stati infine calcolati i Residui:

Residui:

- $Residuo\ \mathrm{B}'_{\mathrm{r}}$= - 1.31130218505859e-05;
- $Residuo\ \mathrm{B}'_{\mathrm{i}}$= - 1.19209289550781e-06;
- $Residuo\ \mathrm{A}'_{\mathrm{r}}$= + 1.31130218505859e-05;
- $Residuo\ \mathrm{A}'_{\mathrm{i}}$= + 1.19209289550781e-06;

Come si può notare il valore dei residui è sempre minore o uguale a metà della risoluzione rappresentabile con la notazione Fractional Point utilizzata all'interno della Butterfly pari a 2^(-16) = 1.52587890625e-05. Pertanto i risultati ottenuti sono corretti e la modalità rispetta le specifiche di progetto.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity TestSingle is
end entity;

architecture behaviour of TestSingle is

component Butterfly is
PORT(Clock,Reset,Start: IN std_logic;
    Done: OUT std_logic;
    DataIn: IN std_logic_vector( 15 downto 0);
    DataInW: IN std_logic_vector( 15 downto 0);
    DataOut: OUT std_logic_vector( 15 downto 0));

end component;

Signal Clock,Reset,Start,Done: Std_logic;
Signal DataIn,DataInW,DataOut: std_logic_vector(15 downto 0);

BEGIN

Process
BEGIN
Reset<='1';
Start<='0';
DataIn<=(others=>'0');
DataInW<=(others=>'0');
wait for 5 ns;
Reset<='0';
wait for 5 ns;
Reset<='0';
wait for 55 ns;
DataIn<="0000110010000000";  -- Dato Br
DataInW<="1100010010000000"; -- Dato Wr;
```

```
Start<='1';
wait for 40 ns;
DataIn<="1111011111110000"; -- Dato Bi
DataInW<="0111010010000000"; -- Dato Wi;
wait for 40 ns;
DataIn<="0000010010000000"; -- Dato Ar
DataInW<=(others=>'0');
wait for 40 ns;
DataIn<="1111110010000000"; -- Dato Ai
wait for 40 ns;
DataIn<=(others=>'0');
Start<='0';
wait for 880 ns;

DataIn<="0001000010111100"; -- Dato Br
DataInW<="1100010010000000"; -- Dato Wr;
Start<='1';
wait for 40 ns;
DataIn<="0011011111111110";  -- Dato Bi
DataInW<="0111010010000000"; -- Dato Wi;
wait for 40 ns;
DataInW<=(others=>'0');
DataIn<="1111011100000000"; -- Dato Ar
wait for 40 ns;
DataIn<="0000010010001110"; -- Dato Ai
wait for 40 ns;
DataIn<=(others=>'0');
Start<='0';
wait;
end process;

Process
BEGIN
Clock<='0';
wait for 20 ns;
Clock<='1';
wait for 20 ns;
end process;

BUTTERFLYBLOCK: Butterfly PORT MAP(Clock,Reset,Start,Done,DataIn,DataInW,DataOut);

end architecture;
```

Script MATLAB Stream 1:

```
clc;
clear all;
format LONG;

%Dichiarazione dei dati di ingresso da elaborare in notazione
%binaria vettoriale: Ordine [ MSB, ..... ,LSB ].

br=[0,0,0,0,1,1,0,0,1,0,0,0,0,0,0,0];
br=Numero(br);
bi=[1,1,1,1,0,1,1,1,1,1,1,1,0,0,0,0];
bi=Numero(bi);
ar=[0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0];
ar=Numero(ar);
ai=[1,1,1,1,1,1,0,0,1,0,0,0,0,0,0,0];
```

```
ai=Numero(ai);
wi=[0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,0];
wi=Numero(wi);
wr=[1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0];
wr=Numero(wr);

%Dati Calcolati dalla Butterfly in notazione binaria vettoriale
% [ MSB, ... , LSB].

BUTTERFLY_b_r=[0,0,0,0,0,0,1,0,1,1,1,1,1,0,0,1];
BUTTERFLY_b_r=Numero(BUTTERFLY_b_r);
BUTTERFLY_b_i=[1,1,1,0,1,1,0,1,0,1,1,0,0,0,0,0];
BUTTERFLY_b_i=Numero(BUTTERFLY_b_i);
BUTTERFLY_a_r=[0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,1];
BUTTERFLY_a_r=Numero(BUTTERFLY_a_r);
BUTTERFLY_a_i=[0,0,0,0,1,0,1,1,1,0,1,0,0,0,0,0];
BUTTERFLY_a_i=Numero(BUTTERFLY_a_i);

%Elaborazione dei dati di ingresso con MATLAB

m1=br*wr;
m2=wi*bi;
m3=br*wi;
m4=bi*wr;
m5=2*ar;
m6=2*ai;
s1=ar+m1;
a_r=s1-m2;
s3=ai+m3;
a_i=s3+m4;
b_r=m5-a_r;
b_i=m6-a_i;

%Calcolo dei residui definiti come differenza tra i valori calcolati da
%MATLAB e i valori calcolati dalla Butterfly. La differenza tra questi deve
%risultare minore o uguale a 2^-16. Questo è dovuto dalla differenza in
%termini di precisione tra la Butterfly e MATLAB.

b_r
b_i
a_r
a_i
Residuo_b_r=b_r-BUTTERFLY_b_r
Residuo_b_i=b_i-BUTTERFLY_b_i
Residuo_a_r=a_r-BUTTERFLY_a_r
Residuo_a_i=a_i-BUTTERFLY_a_i
```

Script MATLAB Stream 2:

```
clc;
clear all;
format LONG;

%Dichiarazione dei dati di ingresso da elaborare in notazione
% binaria vettoriale: Ordine [ MSB, ..... ,LSB ].

br=[0,0,0,1,0,0,0,0,1,0,1,1,1,1,0,0]; %
br=Numero(br)
bi=[0,0,1,1,0,1,1,1,1,1,1,1,1,1,1,0];
bi=Numero(bi)
ar=[1,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0];
ar=Numero(ar)
```

```
ai=[0,0,0,0,0,1,0,0,1,0,0,0,0,1,1,1,0];
ai=Numero(ai)
wi=[0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,0,0];
wi=Numero(wi)
wr=[1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0];
wr=Numero(wr)

%Dati Calcolati dalla Butterfly in notazione binaria vettoriale
% [ MSB, ... , LSB].

BUTTERFLY_b_r=[0,0,1,1,0,0,0,1,1,0,1,1,1,1,1,0];
BUTTERFLY_b_r=Numero(BUTTERFLY_b_r)
BUTTERFLY_b_i=[0,0,0,0,1,1,1,1,0,1,0,1,1,0,1,0];
BUTTERFLY_b_i=Numero(BUTTERFLY_b_i)
BUTTERFLY_a_r=[1,0,1,1,1,1,0,0,0,1,0,0,0,0,1,0];
BUTTERFLY_a_r=Numero(BUTTERFLY_a_r)
BUTTERFLY_a_i=[1,1,1,1,1,0,0,1,1,1,0,0,0,0,1,0];
BUTTERFLY_a_i=Numero(BUTTERFLY_a_i)

%Elaborazione dei dati di ingresso con MATLAB

m1=br*wr;
m2=wi*bi;
m3=br*wi;
m4=bi*wr;
m5=2*ar;
m6=2*ai;
s1=ar+m1;
a_r=s1-m2;
s3=ai+m3;
a_i=s3+m4;
b_r=m5-a_r;
b_i=m6-a_i;

%Calcolo dei residui definiti come differenza tra i valori calcolati da
%MATLAB e i valori calcolati dalla Butterfly. La differenza tra questi deve
%risultare minore o uguale a 2^-16. Questo è dovuto dalla differenza in
%termini di precisione tra la Butterfly e MATLAB.
b_r
b_i
a_r
a_i

Residuo_b_r=b_r-BUTTERFLY_b_r
Residuo_b_i=b_i-BUTTERFLY_b_i
Residuo_a_r=a_r-BUTTERFLY_a_r
Residuo_a_i=a_i-BUTTERFLY_a_i
```

4.2.2 Modalità Continua

In modalità continua sono stati inviati in successione due Stream di dati e coefficienti verificando che la Butterfly fosse in grado di eseguire correttamente i calcoli anche in questa modalità.

Stream Operandi 1:

Dati:

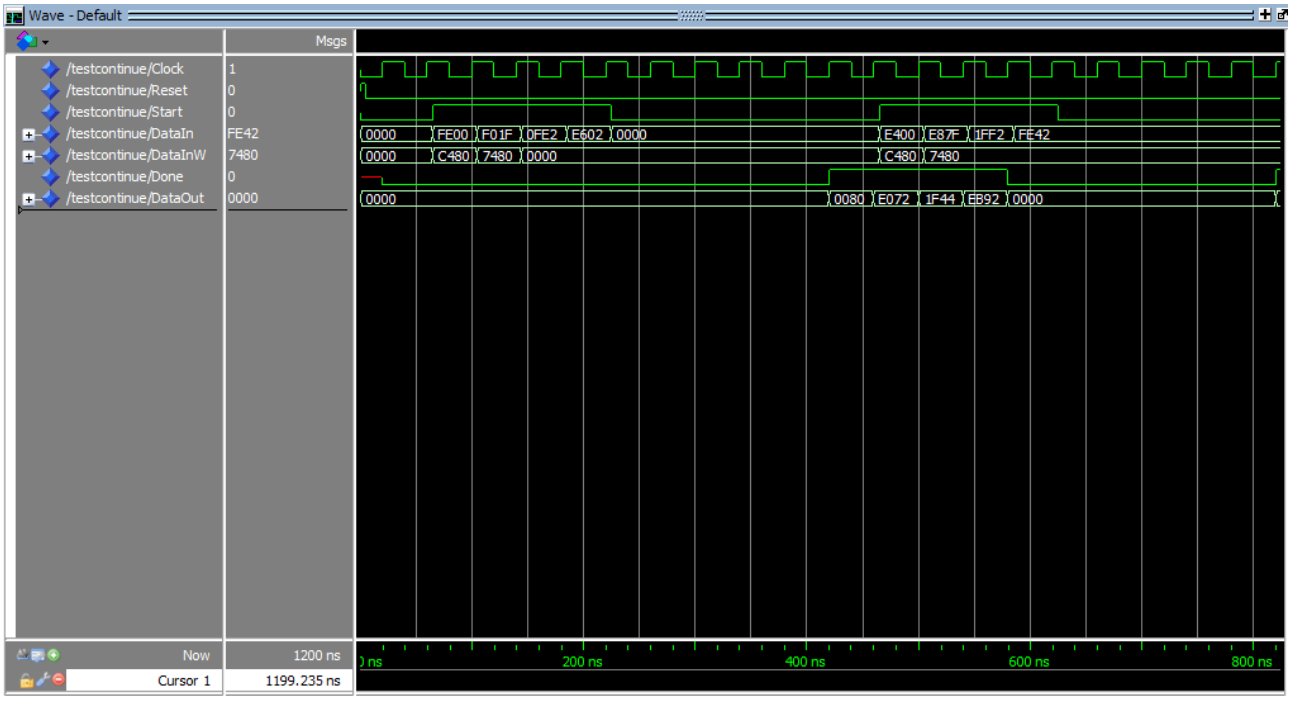
- B_r= “1111111000000000” -> - 0.015625000000000;
- B_i= “1111000000011111” -> - 0.124053955078125;
- A_r= “0000111111100010”-> + 0.124084472656250;
- A_i= “1110011000000010”-> - 0.203063964843750;

Coefficienti:

- W_r=”1100010010000000”-> - 0.464843750000000;
- W_i=” 0111010010000000”-> + 0.910156250000000;

Risultati forniti dalla Butterfly:

- B'_r= “0000000010000000” -> + 0.003906250000000;
- B'_i= “1110000001110010” -> - 0.246520996093750;
- A'_r= “0001111101000100”-> + 0.244262695312500;
- A'_i= “1110101110010010”-> - 0.159606933593750;



I risultati ottenuti con lo script MATLAB sono:

Risultati forniti da MATLAB:

- B'_r= + 0.003912806510925;
- B'_i= - 0.246508479118347;
- A'_r= + 0.244256138801575;
- A'_i= - 0.159619450569153;

Sono stati inoltre calcolati i Residui definiti come differenza tra i risultati della Butterfly e quelli di MATLAB.

Residui:

- $Residuo\ B'_r$= + 6.556510925292969e-06;
- $Residuo\ B'_i$= + 1.251697540283203e-05;
- $Residuo\ A'_r$= - 6.556510925292969e-06;
- $Residuo\ A'_i$= - 1.251697540283203e-05;

Stream Operandi 2:

Dati:

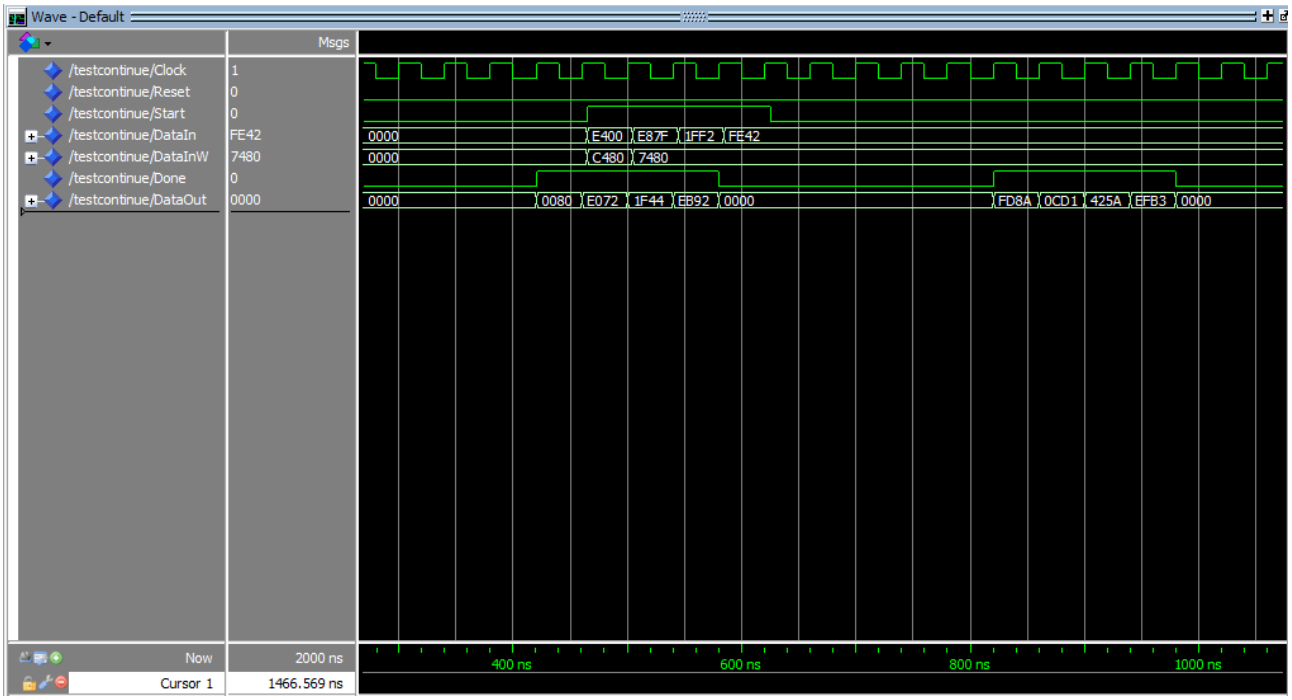
- B_r= “1110010000000000” -> - 0.218750000000000;
- B_i= “1110100001111111” -> - 0.183624267578125;
- A_r= “0001111111110010”-> + 0.249572753906250;
- A_i= “1111111001000010”-> - 0.013610839843750;

Coefficienti:

- W_r=”1100010010000000”-> - 0.464843750000000;
- W_i=”0111010010000000”-> + 0.910156250000000;

Risultati forniti dalla Butterfly:

- B'_r= “1111110110001010” -> - 0.019226074218750;
- B'_i= “0000110011010001” -> + 0.100128173828125;
- A'_r= “0100001001011010” -> + 0.518371582031250;
- A'_i= “1110111110110011” -> - 0.127349853515625;



I risultati ottenuti con lo script MATLAB sono:

Risultati forniti da MATLAB:

- B'_r= - 0.019238591194153;
- B'_i= + 0.100129246711731;
- A'_r= + 0.518384099006653;
- A'_i= - 0.127350926399231;

Sono stati inoltre calcolati i Residui:

Residui:

- $Residuo\ B'_r$= - 1.251697540283203e-05;
- $Residuo\ B'_i$= + 1.072883605957031e-06;
- $Residuo\ A'_r$= + 1.251697540283203e-05;
- $Residuo\ A'_i$= - 1.072883605957031e-06;

Come si può notare, anche in questo caso, il valore dei residui è sempre minore o uguale a metà della risoluzione rappresentabile con la notazione Fractional Point. È quindi verificato il corretto funzionamento della Modalità continua.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity TestContinue is
end entity;

architecture behaviour of TestContinue is

component Butterfly is
PORT(Clock,Reset,Start: IN std_logic;
    Done: OUT std_logic;
    DataIn: IN std_logic_vector( 15 downto 0);
    DataInW: IN std_logic_vector( 15 downto 0);
    DataOut: OUT std_logic_vector( 15 downto 0));
end component;

Signal Clock,Reset,Start,Done:Std_logic;
Signal DataIn,DataInW,DataOut: std_logic_vector(15 downto 0);

BEGIN
Process
BEGIN
Reset<='1';
Start<='0';
DataInW<=(others=>'0');
DataIn<=(others=>'0');
wait for 5 ns;
Reset<='0';
wait for 5 ns;
Reset<='0';
wait for 55 ns;
Start<='1';
DataIn<="1111111000000000"; --Dato Br
DataInW<="1100010010000000"; -- Dato Wr;
wait for 40 ns;
DataIn<="1111000000011111"; --Dato Bi
DataInW<="0111010010000000"; -- Dato Wi;
wait for 40 ns;
DataInW<=(others=>'0');
DataIn<="0000111111100010"; --Dato Ar
wait for 40 ns;
DataIn<="1110011000000010"; --Dato Ai
wait for 40 ns;
DataIn<=(others=>'0');
Start<='0';
wait for 240 ns;
Start<='1';
DataIn<="1110010000000000"; --Dato Br
DataInW<="1100010010000000"; -- Dato Wr;
wait for 40 ns;

DataIn<="1110100001111111"; --Dato Bi
DataInW<="0111010010000000"; -- Dato Wi;
wait for 40 ns;
```

```
DataIn<="0001111111110010"; --Dato Ar
wait for 40 ns;
DataIn<="1111111001000010"; --Dato Ai
wait for 40 ns;
Start<='0';
wait;
end process;

Process
BEGIN
Clock<='0';
wait for 20 ns;
Clock<='1';
wait for 20 ns;
end process;


BUTT: Butterfly PORT MAP(Clock,Reset,Start,Done,DataIn,DataInW,DataOut);

end architecture;
```

Script MATLAB Stream 1:

```
clc;
clear all;
format LONG;

%Dichiarazione dei dati di ingresso da elaborare in notazione
% binaria vettoriale: Ordine [ MSB, ..... ,LSB ].

br=[1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0]; %
br=Numero(br)
bi=[1,1,1,1,0,0,0,0,0,0,0,1,1,1,1,1];
bi=Numero(bi)
ar=[0,0,0,0,1,1,1,1,1,1,1,0,0,0,1,0];
ar=Numero(ar)
ai=[1,1,1,0,0,1,1,0,0,0,0,0,0,0,1,0];
ai=Numero(ai)
wi=[0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,0];
wi=Numero(wi)
wr=[1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0];
wr=Numero(wr)

%Dati Calcolati dalla Butterfly in notazione binaria vettoriale
% [ MSB, ... , LSB].

BUTTERFLY_b_r=[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0];
BUTTERFLY_b_r=Numero(BUTTERFLY_b_r)
BUTTERFLY_b_i=[1,1,1,0,0,0,0,0,0,1,1,1,0,0,1,0];
BUTTERFLY_b_i=Numero(BUTTERFLY_b_i)
BUTTERFLY_a_r=[0,0,0,1,1,1,1,1,0,1,0,0,0,1,0,0];
BUTTERFLY_a_r=Numero(BUTTERFLY_a_r)
BUTTERFLY_a_i=[1,1,1,0,1,0,1,1,1,0,0,1,0,0,1,0];
BUTTERFLY_a_i=Numero(BUTTERFLY_a_i)

%Elaborazione dei dati di ingresso con MATLAB

m1=br*wr;
m2=wi*bi;
m3=br*wi;
m4=bi*wr;
```

```
m5=2*ar;
m6=2*ai;
s1=ar+m1;
a_r=s1-m2;
s3=ai+m3;
a_i=s3+m4;
b_r=m5-a_r;
b_i=m6-a_i;

 %Calcolo dei residui definiti come differenza tra i valori calcolati da
%MATLAB e i valori calcolati dalla Butterfly. La differenza tra questi deve
%risultare minore o uguale a 2^-16. Questo è dovuto dalla differenza in
%termini di precisione tra la Butterfly e MATLAB.

b_r
b_i
a_r
a_i
Residuo_b_r=b_r-BUTTERFLY_b_r
Residuo_b_i=b_i-BUTTERFLY_b_i
Residuo_a_r=a_r-BUTTERFLY_a_r
Residuo_a_i=a_i-BUTTERFLY_a_i
```

Script MATLAB Stream 2:

```
clc;
clear all;
format LONG;

%Dichiarazione dei dati di ingresso da elaborare in notazione
% binaria vettoriale: Ordine [ MSB, ..... ,LSB ].

br=[1,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0]; %
br=Numero(br)
bi=[1,1,1,0,1,0,0,0,0,1,1,1,1,1,1,1];
bi=Numero(bi)
ar=[0,0,0,1,1,1,1,1,1,1,1,1,0,0,1,0];
ar=Numero(ar)
ai=[1,1,1,1,1,1,1,0,0,1,0,0,0,0,1,0];
ai=Numero(ai)
wi=[0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,0];
wi=Numero(wi)
wr=[1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0];
wr=Numero(wr)

%Dati Calcolati dalla Butterfly in notazione binaria vettoriale
% [ MSB, ... , LSB].

BUTTERFLY_b_r=[1,1,1,1,1,1,0,1,1,0,0,0,1,0,1,0];
BUTTERFLY_b_r=Numero(BUTTERFLY_b_r)
BUTTERFLY_b_i=[0,0,0,0,1,1,0,0,1,1,0,1,0,0,0,1];
BUTTERFLY_b_i=Numero(BUTTERFLY_b_i)
BUTTERFLY_a_r=[0,1,0,0,0,0,1,0,0,1,0,1,1,0,1,0];
BUTTERFLY_a_r=Numero(BUTTERFLY_a_r)
BUTTERFLY_a_i=[1,1,1,0,1,1,1,1,1,0,1,1,0,0,1,1];
BUTTERFLY_a_i=Numero(BUTTERFLY_a_i)

%Elaborazione dei dati di ingresso con MATLAB
m1=br*wr;
m2=wi*bi;
m3=br*wi;
m4=bi*wr;
m5=2*ar;
```

```
m6=2*ai;
s1=ar+m1;
a_r=s1-m2;
s3=ai+m3;
a_i=s3+m4;
b_r=m5-a_r;
b_i=m6-a_i;

%Calcolo dei residui definiti come differenza tra i valori calcolati da
%MATLAB e i valori calcolati dalla Butterfly. La differenza tra questi deve
%risultare minore o uguale a 2^-16. Questo è dovuto dalla differenza in
%termini di precisione tra la Butterfly e MATLAB.
b_r
b_i
a_r
a_i

Residuo_b_r=b_r-BUTTERFLY_b_r
Residuo_b_i=b_i-BUTTERFLY_b_i
Residuo_a_r=a_r-BUTTERFLY_a_r
Residuo_a_i=a_i-BUTTERFLY_a_i
```

4.2.3 Modalità Sequenza

Per la modalità sequenza è stato simulato il funzionamento di due Butterfly in cascata. I risultati della prima vengono utilizzati come operandi della seconda. I coefficienti W invece sono forniti dall'esterno ad entrambe. Questo modalità di funzionamento implica l'esistenza di un controllore esterno che fornisca ad entrambe i coefficienti durante l'elaborazione. Per garantire che i risultati siano corretti è necessario considerare quattro bit di guardia per gli operandi in ingresso alla Butterfly 1.

Stream Operandi 1:

Dati:

- B_r= "0000111000000000" -> + 0.109375000000000;
- B_i= "1111000110000011" -> - 0.113189697265625;
- A_r= "0000100001100010"-> + 0.065490722656250;
- A_i= "1111111110000010"-> - 0.003845214843750;

Coefficienti Butterfly1:

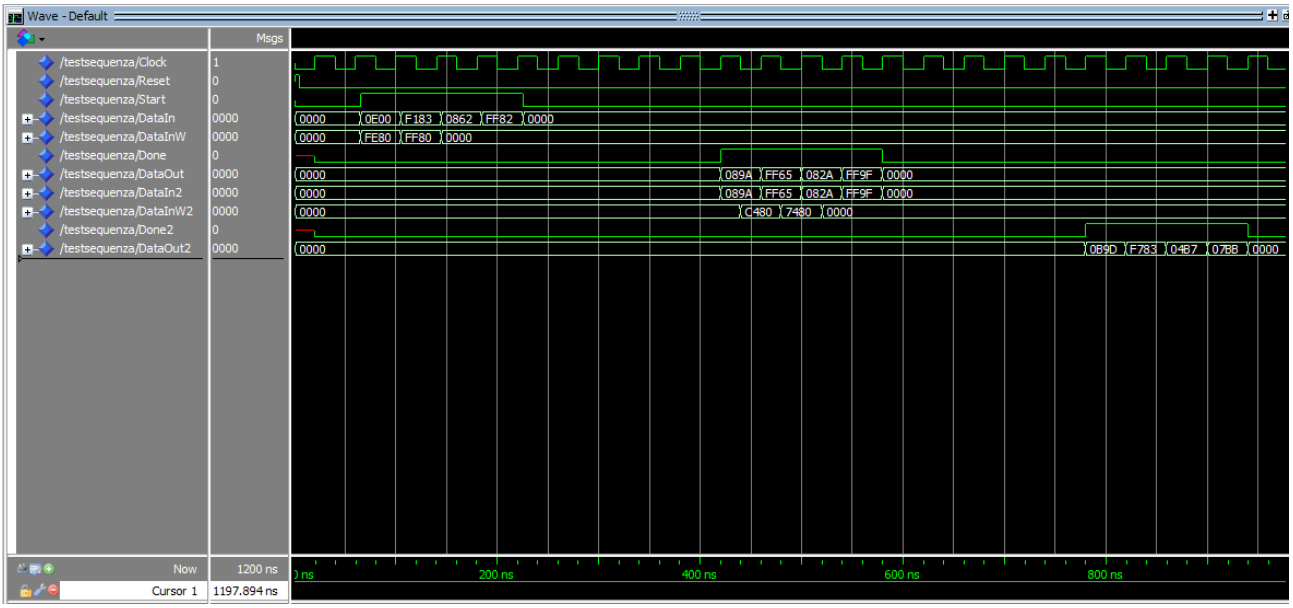
- W_r="1111111010000000"-> - 0.011718750000000;
- W_i=" 1111111110000000"-> - 0.003906250000000;

Coefficienti Butterfly2:

- W_r="1100010010000000"-> - 0.464843750000000;
- W_i="0111010010000000"-> + 0.910156250000000;

Risultati forniti dalla Butterfly2:

- B'_r= “0000000010000000” -> + 0.090728759765625;
- B'_i= “1110000001110010” -> - 0.066314697265625;
- A'_r= “0001111101000100”-> + 0.036834716796875;
- A'_i= “1110101110010010”-> + 0.060394287109375;



I risultati ottenuti con lo script MATLAB invece sono:

Risultati forniti da MATLAB:

- B'_r= + 0.090713858604431;
- B'_i= - 0.066321253776550;
- A'_r= + 0.036849617958069;
- A'_i= + 0.060400843620300;

Sono stati inoltre calcolati i Residui:

Residui:

- $Residuo\ B'_r$= - 1.490116119384766e-05;
- $Residuo\ B'_i$= - 6.556510925292969e-06;
- $Residuo\ A'_r$= + 1.490116119384766e-05;
- $Residuo\ A'_i$= + 6.556510925292969e-06;

Come si può notare anche in questo caso i residui sono minori o uguali a metà della risoluzione rappresentabile dalla Butterfly. Pertanto anche questa modalità funziona correttamente.

Codice VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity TestSequenza is
end entity;

architecture behaviour of TestSequenza is


component Butterfly is
PORT(Clock,Reset,Start: IN std_logic;
    Done: OUT std_logic;
    DataIn: IN std_logic_vector( 15 downto 0);
    DataInW: IN std_logic_vector( 15 downto 0);
    DataOut: OUT std_logic_vector( 15 downto 0));
end component;

Signal Clock,Reset,Start,Done,Done2: Std_logic;
Signal DataIn,DataIn2,DataInW,DataInW2,DataOut,DataOut2: std_logic_vector(15 downto 0);

BEGIN

DataIn2<=DataOut;

Process
BEGIN

Reset<='1';
Start<='0';
DataInW<=(others=>'0');
DataInW2<=(others=>'0');
DataIn<=(others=>'0');

wait for 5 ns;
Reset<='0';
wait for 5 ns;
Reset<='0';
wait for 55 ns;
Start<='1';
DataIn<="0000111000000000"; --Dato Br
DataInW<="1111111010000000"; -- Dato Wr;
wait for 40 ns;
DataIn<="1111000110000011"; --Dato Bi
DataInW<="1111111110000000"; -- Dato Wi;
wait for 40 ns;
DataInW<=(others=>'0');
DataIn<="0000100001100010"; --Dato Ar
wait for 40 ns;
DataIn<="1111111110000010"; --Dato Ai
wait for 40 ns;
DataIn<=(others=>'0');
DataInW<=(others=>'0');
Start<='0';
wait for 215 ns;
DataInW2<="1100010010000000";
```

```
wait for 40 ns;
DataInW2<="0111010010000000"; -- Dato Wi;
wait for 40 ns;
DataInW<=(others=>'0');
DataIn<=(others=>'0');
DataInW2<=(others=>'0');
wait;
end process;

Process
BEGIN
Clock<='0';
wait for 20 ns;
Clock<='1';
wait for 20 ns;
end process;



BUTT: Butterfly PORT MAP(Clock,Reset,Start,Done,DataIn,DataInW,DataOut);
BUTT2: Butterfly PORT MAP(Clock,Reset,Done,Done2,DataIn2,DataInW2,DataOut2);

end architecture;
```

Script MATLAB:

```
clc;
clear all;
format LONG;
%Dichiarazione dei dati di ingresso alla Butterfly 1
%Notazione binaria vettoriale: Ordine [ MSB, ..... ,LSB ].
br=[0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0]; %
br=Numero(br)
bi=[1,1,1,1,0,0,0,1,1,0,0,0,0,0,1,1];
bi=Numero(bi)
ar=[0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0];
ar=Numero(ar)
ai=[1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,0];
ai=Numero(ai)
wr=[1,1,1,1,1,1,1,0,1,0,0,0,0,0,0,0];
wr=Numero(wr)
wi=[1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0];
wi=Numero(wi)
%Dati Calcolati dalla Butterfly 1 in notazione binaria vettoriale
% [ MSB, ... , LSB].
BUTTERFLY1_b_r=[0,0,0,0,1,0,0,0,1,0,0,1,1,0,1,0];
BUTTERFLY1_b_r=Numero(BUTTERFLY1_b_r)
BUTTERFLY1_b_i=[1,1,1,1,1,1,1,1,0,1,1,0,0,1,0,1];
BUTTERFLY1_b_i=Numero(BUTTERFLY1_b_i)
BUTTERFLY1_a_r=[0,0,0,0,1,0,0,0,0,0,1,0,1,0,1,0];
BUTTERFLY1_a_r=Numero(BUTTERFLY1_a_r)
BUTTERFLY1_a_i=[1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1];
BUTTERFLY1_a_i=Numero(BUTTERFLY1_a_i)
%Elaborazione dei dati di ingresso
m1=br*wr;
m2=wi*bi;
m3=br*wi;
m4=bi*wr;
m5=2*ar;
m6=2*ai;
s1=ar+m1;
a_r=s1-m2;
s3=ai+m3;
```

```
a_i=s3+m4;
b_r=m5-a_r;
b_i=m6-a_i;
b_r
b_i
a_r
a_i
Residuo_b_r=b_r-BUTTERFLY1_b_r
Residuo_b_i=b_i-BUTTERFLY1_b_i
Residuo_a_r=a_r-BUTTERFLY1_a_r
Residuo_a_i=a_i-BUTTERFLY1_a_i
% Elaborazione dei Risultati della Butterfly 1 nella Butterfly 2
wr2=[1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0];
wr2=Numero(wr2)
wi2=[0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,0];
wi2=Numero(wi2)

m1=BUTTERFLY1_b_r*wr2;
m2=wi2*BUTTERFLY1_b_i;
m3=BUTTERFLY1_b_r*wi2;
m4=BUTTERFLY1_b_i*wr2;
m5=2*BUTTERFLY1_a_r;
m6=2*BUTTERFLY1_a_i;
s1=BUTTERFLY1_a_r+m1;
a_r2=s1-m2;
s3=BUTTERFLY1_a_i+m3;
a_i2=s3+m4;
b_r2=m5-a_r2;
b_i2=m6-a_i2;


%Dati Calcolati dalla Butterfly 2 in notazione binaria vettoriale
% [ MSB, ... , LSB].

BUTTERFLY2_b_r=[0,0,0,0,1,0,1,1,1,0,0,1,1,1,0,1];
BUTTERFLY2_b_r=Numero(BUTTERFLY2_b_r)
BUTTERFLY2_b_i=[1,1,1,1,0,1,1,1,1,0,0,0,0,0,1,1];
BUTTERFLY2_b_i=Numero(BUTTERFLY2_b_i)
BUTTERFLY2_a_r=[0,0,0,0,0,1,0,0,1,0,1,1,0,1,1,1];
BUTTERFLY2_a_r=Numero(BUTTERFLY2_a_r)
BUTTERFLY2_a_i=[0,0,0,0,0,1,1,1,1,0,1,1,1,0,1,1];
BUTTERFLY2_a_i=Numero(BUTTERFLY2_a_i)

%Calcolo dei residui definiti come differenza tra i valori calcolati da
%MATLAB e i valori calcolati dalla Butterfly2. La differenza tra questi deve
%risultare minore o uguale a 2^-16. Questo è dovuto dalla differenza in
%termini di precisione tra la Butterfly2 e MATLAB.
b_r2
b_i2
a_r2
a_i2

Residuo_b_r2=b_r2-BUTTERFLY2_b_r
Residuo_b_i2=b_i2-BUTTERFLY2_b_i
Residuo_a_r2=a_r2-BUTTERFLY2_a_r
Residuo_a_i2=a_i2-BUTTERFLY2_a_i
```

5.0 Manuale Utente

Per utilizzare correttamente la Butterfly è necessario considerare tre operazioni principali.

- Reset della Butterfly;
- Caricamento dei dati e dei coefficienti;
- Lettura dei risultati;

5.1 Reset della Butterfly

Per portare allo stato di Reset la Butterfly è necessario, un volta generato l'impulso di Reset, attendere due fronti positivi del Clock prima di poter attivare il segnale di Start e presentare sulla porta di ingresso il dato Br. Il Reset è di tipo asincrono.

5.2 Caricamento dei dati e dei coefficienti

Raggiunto lo stato di *IDLE* la Butterfly può, su ogni fronte di Clock, cominciare un ciclo di operazioni. Per iniziare correttamente l'elaborazione è necessario, prima del successivo fronte, fornire sulla porta **Data_IN** il dato $\mathrm{B_r}$, sulla porta **Data_IN_W** Wr e attivare il segnale di **Start**. Lo stesso per ogni operando e coefficiente successivo. Tutte le porte di ingresso sono campionate sul fronte positivo del Clock. È quindi necessario stabilizzare il dato prima del fronte. Il segnale di **Start** deve essere mantenuto per tutto il ciclo di caricamento, ossia per quattro colpi di clock.

5.3 Lettura dei risultati

Al termine dell'elaborazione, cinque fronti dopo all'ultimo dato caricato, la Butterfly porta il segnale di **Done** a livello logico 1 e parallelamente viene mostrato il dato $\mathrm{B_r'}$ valido per un ciclo di clock. Durante i successivi tre periodi di clock vengono mostrati $B_i', A_r' \; e \; A_i'$. Durante tutto il ciclo di lettura dei risultati, mostrati sulla porta **Data_OUT**, il segnale **Done** è a livello logico 1.

5.4 Timing Utente

Di seguito è mostrato un esempio delle fasi di reset, caricamento degli operandi e lettura dei risultati.