Progettazione e simulazione di un'unità Butterfly

Sistemi Digitali Integrati

Professor Maurizio Zamboni

Studenti:

Calabrese Deborah	241408
Lanieri Valerio	234526
Mazzia Claudia	235208

18 novembre 2018

Sommario

Questo report descrive la progettazione, la realizzazione in VHDL e il testing, attraverso Modelsim, di un'unità Butterfly, ossia di un'unità di calcolo che fornisce in uscita il risultato di un'operazione di somma e moltiplicazione fra numeri complessi. Tale unità costituisce l'elemento base di una struttura più complessa, preposta a realizzare la DFT (trasformata di Fourier discreta) dei dati in input.

Indice

1 Introduzione e progettazione

Il progetto consiste nella realizzazione di un'unità Butterfly, ossia di un elemento hardware in grado di svolgere le seguenti operazioni, dati gli input complessi A, B e W:

$$A' = A + BW^k \qquad B' = A + BW^{k+\frac{N}{2}}$$

Dato che N rappresenta il numero totale di campioni, sul piano dei numeri complessi $\frac{N}{2}$ rappresenta una rotazione di 180°, dunque:

$$W^{k+\frac{N}{2}} = -W^k$$

e quindi:

$$B' = A - BW^k$$

Dato che, nel calcolare B', avremo già a disposizione A', possiamo semplificare l'operazione nel modo seguente:

$$B' = 2A - A'$$

A questo punto possiamo pianificare le singole operazioni da svolgere per giungere ai risultati complessi A' e B':

$$A' = A_r + B_rW_r - B_iW_i + j(A_i + B_rW_i + B_iW_r)$$
$$B' = 2A_r - A'_r + j(2A_i - A'_i)$$

$$\begin{aligned} M1 &= B_rW_r & M2 &= B_iW_i \\ M3 &= B_rW_i & M4 &= B_iW_r \\ S1 &= 2A_r & S2 &= 2A_i \\ \Sigma_1 &= A_r + M1 & \Sigma_2 &= \Sigma_1 - M2 \\ \Sigma_3 &= A_i + M3 & \Sigma_4 &= \Sigma_4 + M4 \\ \Sigma_5 &= S1 - \Sigma_2 & \Sigma_6 &= S2 - \Sigma_4 \end{aligned}$$

Ci siamo dunque impegnati nella realizzazione di un Data Flow Diagram per far sì che tali operazioni fossero svolte nella maniera più rapida possibile, cercando al contempo di limitare il numero di registri temporanei e di linee di bus allo stretto necessario.

Altre caratteristiche fornite dalle specifiche tecniche sono l'utilizzo di un parallelismo esterno a 16 bit ed uno interno a 31 bit. Per quanto riguarda il primo, ci è stato detto che, usando due bit di guardia, siamo in grado di evitare overflow se consideriamo la singola unità butterfly. In parole povere, dato che in forma frazionaria i bit 15 e 14 rappresentano rispettivamente 2^{-1} e 2^{-2}, se l'input è inferiore, in modulo, a 0.25, non corriamo rischi di overflow. Ci è stato detto di dare per scontato che gli input rispetteranno tale restrizione, dunque non ci siamo occupati della gestione degli eventuali overflow, e, in fase di testing, ci siamo aspettati risultati consistenti solo per input compresi fra 0.001111111111111 e 1.110000000000000.

Per quanto riguarda il parallelismo interno, 31 bit sono sufficienti perché il risultato delle moltiplicazioni non sia mai pari ad uno (né superiore), pertanto è possibile troncare il bit più significativo. In altre parole, dato che il risultato di una moltiplicazione fra due numeri binari con 1 solo bit intero e 15 bit frazionari è un numero a 32 bit con due cifre intere e 30 frazionare è possibile scartare il bit più significativo, dato che non sarà mai utilizzato per fornire informazioni sul risultato (ossia, sarà sempre pari a 0 se il numero è positivo e pari a 1 se il numero è negativo). Per raggiungere tale parallelismo interno a partire da 16 bit, inoltre, è sufficiente aggiungere 15 '0' al termine dell'input, dato che rappresentano un'aggiunta in termini di risoluzione. La cifra rappresentata, in questo modo, rimane inalterata, sia che si tratti di un numero positivo che negativo.

Per quanto riguarda invece il passaggio da 31 a 16 bit, la consegna richiede di implementare un circuito che effettui un arrotondamento utilizzando la tecnica del Rounding to Nearest Even, che consiste nell'arrotondare per eccesso solo se:

- il 16° bit è pari a 0, il 15° bit è pari ad 1 e uno qualsiasi dei bit seguenti è pari a 1;
- il 16° bit è pari a 1 e il 15° bit è pari a 1;

Dato che in uscita verranno inviati solo i 16 bit più significativi, l'arrotondamento per eccesso consiste nell'aggiungere 1 LSB (ossia il bit di peso 2^{-15}) a tale cifra.

Infine, l'unità di controllo è stata realizzata utilizzando un'architettura di tipo Late Status. Tale architettura prevede l'utilizzo di una microrom fatta di due memorie, contenenti le istruzioni da inviare al datapath. Tali memorie vengono lette in contemporanea in base all'indirizzo presente in ingresso, proveniente dal micro address register. L'istruzione che viene inviata al datapath (e da cui si ricava l'indirizzo successivo) viene selezionata, fra le due possibili, da un multiplexer. Nel nostro caso, c'è bisogno di effettuare una scelta fra due possibilità solo quando ci si aspetta un segnale di start. Dato che viene richiesto dalla consegna di considerare sia il caso in cui si debba effettuare un'operazione singola (modalità singola) sia il caso in cui altri dati vengano forniti in ingresso prima che l'operazione corrente si sia conclusa (modalità continua), la presenza del segnale di start influenza il passaggio da uno stato di inattività (idle) all'inizio della modalità singola e da una fase ben precisa della modalità singola all'inizio della modalità continua.

Nelle prossime sezioni presenteremo tutti gli strumenti grafici realizzati in fase di progettazione, ossia Data Flow Diagram, Data Dependencies, Datapath, ASM chart e Timing Diagram. Nelle sezioni successive invece discuteremo individualmente ciascuno dei punti ora brevemente introdotti in termini di progettazione, criticità, realizzazione e validazione complessiva della struttura.

1.1 Data Flow Diagram

Il primo passo ai fini della realizzazione della Butterfly è stato quello di ricavare il DFD a partire dalle specifiche di progetto assegnate. Sappiamo infatti che abbiamo a disposizione:

- un moltiplicatore pipelinato che restituisce il prodotto nel colpo di clock successivo a quello necessario per svolgere l'operazione, e ciò implica che si potrà svolgere un'unica mltiplicazione per colpo di clock;
- due sommatori/sottrattori, perciò si avranno al più due somme/sottrazioni per colpo di clock;
- uno shifter che esegue la moltiplicazione per 2 e restituisce il risulato istantaneamente;
- un arrotondatore che arrotonda i dati su 16 bit prima che vengano mandati in uscita usando, come detto prima, l'approccio round to nearest even;

A livello di timing, stando al Data Flow Diagram che abbiamo realizzato, sono necessari nove colpi di clock perché siano svolte tutte le operazioni necessarie e si ottengano i nuovi valori in uscita.

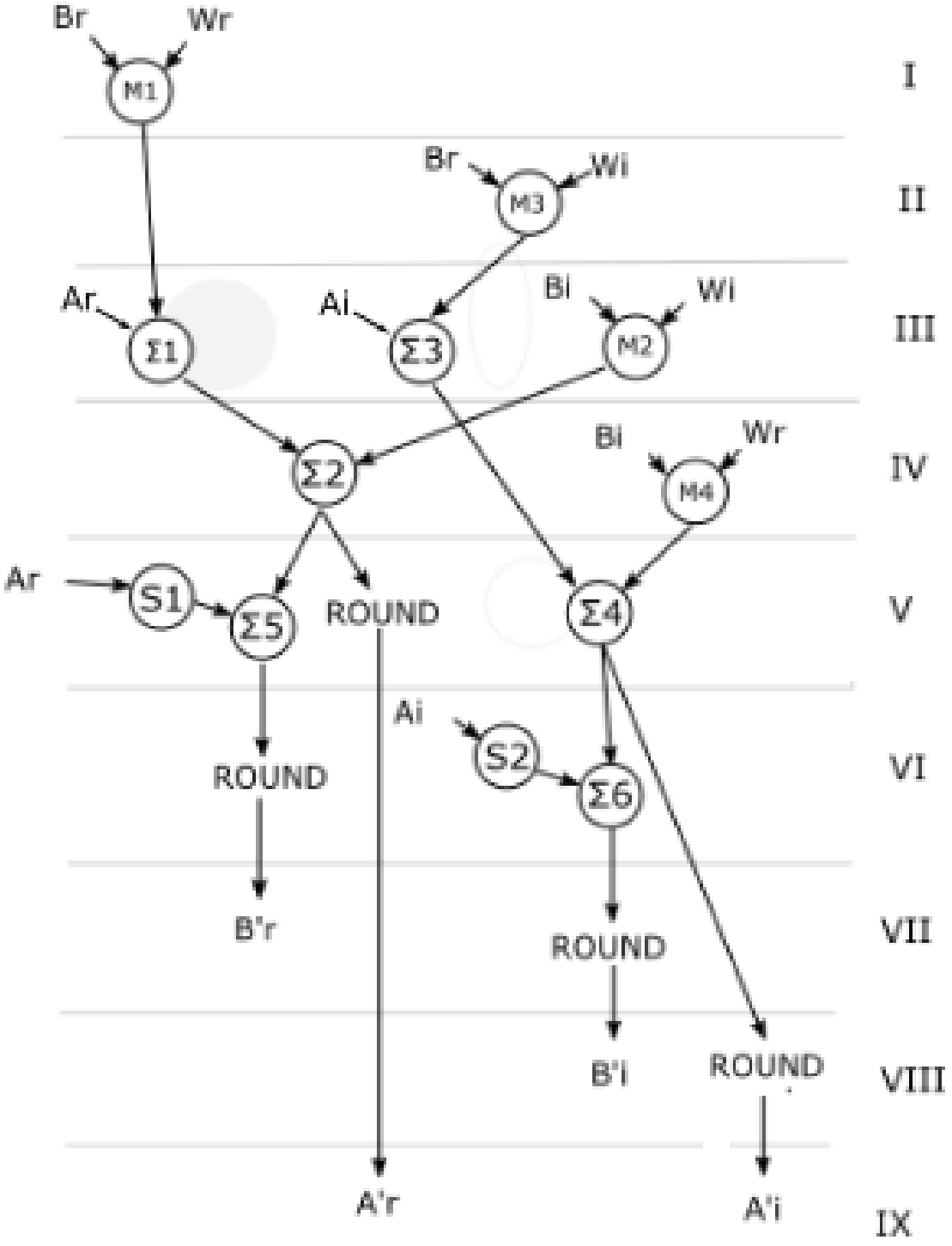

Figura 1: Data Flow Diagram

1.2 Data Dependencies

Dallo studio del Data Flow Diagram abbiamo ricavato le Data Dependencies e quindi i tempi di vita delle variabili. Notare che il massimo numero di variabili temporanee che ci servono contemporaneamente è sette.

	I	II	III	IV	V	VI	VII	VIII	IX
Ar			X	X	X				
Ai			X	X	X	X			
Br	X	X							
Bi			X	X					
Wr	X	X	X	X					
Wi		X	X						
M1		X	X						
M3			X						
M2				X					
Σ1				X					
Σ2					X				
Σ3				X	X				
M4					X				
S1					X				
Σ4						X	X	X	
Σ5						X			
S2						X			
Σ6							X		
A'r						X	X	X	X
A'i									X
B'r							X		
B'i								X	

Tabella 1: Data Dependencies

1.3 Datapath

Nel nostro datapath abbiamo rispettato le specifiche di progetto, ossia:

- ogni operazione impiega un colpo di clock;
- abbiamo a disposizione un solo moltiplicatore con un livello di pipe;
- abbiamo a disposizione due sommatori/sottrattori, progettati come macchine combinatorie. Ciascuna ha un ingresso di selezione (C0 e C1) che pilota l'operazione di somma o sottrazione dei due componenti;
- abbiamo a disposizione un unico blocco combinatorio per effettuare uno shift aritmetico. Dato che è costituito solamente da fili, esso NON è composto da porte logiche e pertanto fornisce un risultato senza alcun ritardo considerevole;
- abbiamo infine a disposizione un unico blocco combinatorio, il ROUNDER, che effettua l'arrotondamento dei dati prima che vengano mandati in uscita dalla nostra unità;

A partire poi dal Data Flow Diagram e dalle Data Dependencies, abbiamo effettuato delle ottimizzazioni nel nostro datapath in modo tale da utilizzare il minor numero possibile di variabili temporanee e di bus globali.

Abbiamo deciso di dare l'assoluta precedenza al numero di variabili temporanee, per cui, oltre ai due registri necessari in uscita dai sommatori per rispettare la prima specifica (si tratta dei registri R2 ed R3), abbiamo utilizzato solo altri due registri, uno per memorizzare A'$_i$ (R4) e un altro registro in ingresso al moltiplicatore (R1). Quest'ultima scelta, in particolare, è stata fatta per poter utilizzare solo tre bus in uscita dal register file e non quattro, che sarebbero stati necessari nel caso in cui avessimo messo il registro R1 all'uscita del moltiplicatore.

Abbiamo inoltre utilizzato sette multiplexer, di cui SHF_MUX serve per rispettare la specifica secondo cui abbiamo un unico blocco di shift. Vi è poi il multiplexer 6 che utilizziamo per poter inviare M1 al registro in uscita dal sommatore 2, poiché dallo studio delle Data Dependencies ci siamo resi conto che le due variabili non sono mai 'vive' nello stesso colpo di clock.

Come si può notare dal datapath, inoltre, l'uscita di R3 va sia all'ingresso del sommatore 2 (mediante il multiplexer 3) che all'ingresso del sommatore 1 (mediante il multiplexer 2) e ciò è stato fatto per far sì che $\Sigma 4$ resti memorizzata nel registro R3 fin quando non viene mandata al Rounder, evitando così l'utilizzo di un ulteriore registro.

Per rispettare il protocollo di scambio dati con l'esterno e le scelte fatte sul datapath servono inoltre due bus in ingresso al register file e tre bus globali in uscita dal register file.

Il Register File ha il compito di ricevere i dati da elaborare da due bus a 16 bit, memorizzare tali dati e restituirli in uscita sui tre bus dati nell'ordine richiesto e progettato nel DFD. La sua struttura interna è molto semplice ed è formata da 6 registri, ciascuno per memorizzare la parte reale o immaginaria dei dati, e da 2 Multiplexer per selezionare quale dato mandare sul rispettivo bus dati di uscita. I segnali di selezione dei due multiplexer, SEL_BUS0 e SEL_BUS1, provengono dalla unità di controllo così come i segnali WR0, WR1 e WR2 che comandano l'abilitazione dei singoli registri.

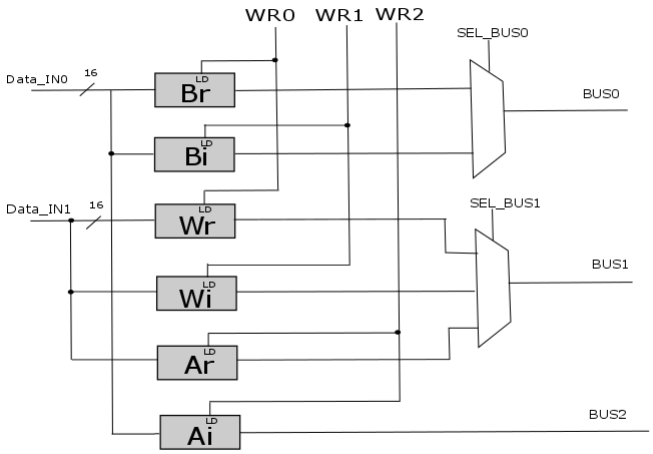

Figura 2: Register File

Il Rounder è un circuito puramente combinatorio con un parallelismo dati in ingresso a 31 bit e un parallelismo dati in uscita a 16 bit. Il suo scopo è quello di troncare 15 bit del dato in ingresso (il quale è a 31 bit), così da far uscire un dato a 16 bit, rispettando il parallelismo in ingresso. Come detto prima, il troncamento viene svolto contestualmente con un arrotondamento di tipo rounding to nearest even, precedentemente descritto.

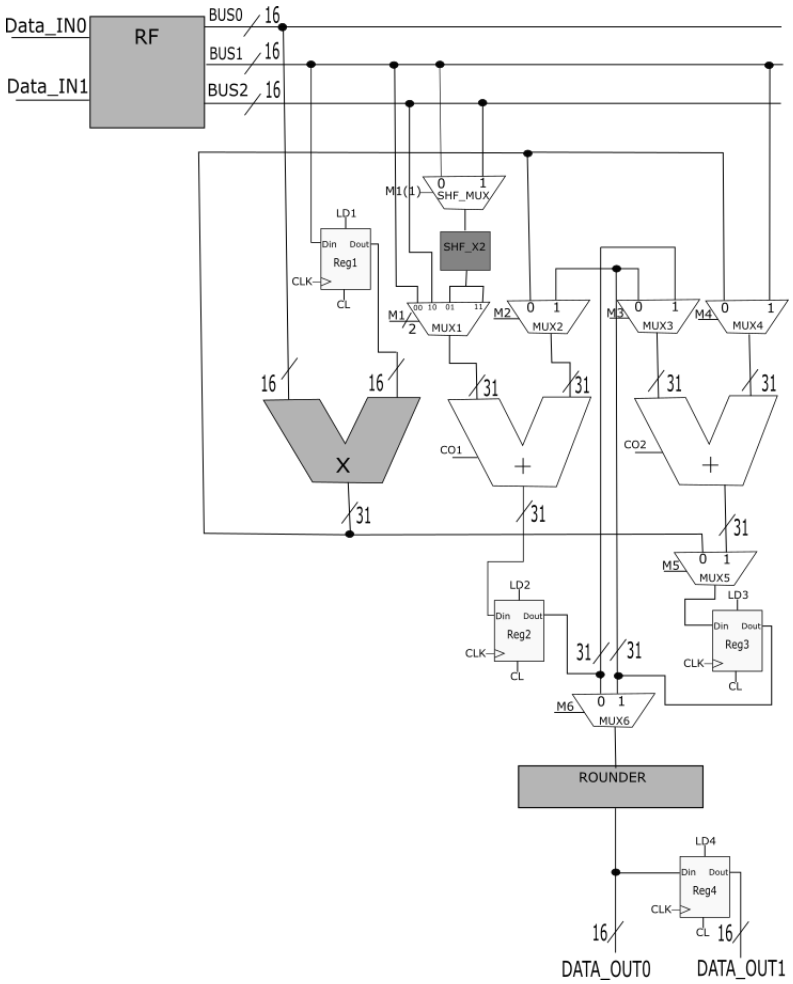

Figura 3: Datapath della Butterfly

2 Unità di Controllo

L'unità di controllo imposta dal progetto è di tipo Late Status. Nel nostro caso consiste di una micro ROM a due colonne per 7 righe, un multiplexer da 2 a 1 posto a valle della micro ROM e da due registri, a loro volta posti di seguito, che fungono da micro instruction register e micro address register. Gli ultimi due componenti sono una PLA (programmable logic array) ed un multiplexer di bypass. Il funzionamento di questi componenti è descritto di seguito:

- l'indirizzo fornito dal micro address register alla micro ROM viene comunicato a entrambe le colonne, dunque vengono lette due istruzioni in contemporanea. Nello specifico, l'indirizzo è composto da 4 bit. I primi 3 bit rappresentano la riga, e sono questi ad essere inviati alla micro ROM. Il bit meno significativo, invece, viene utilizzato per selezionare la colonna;
- le due istruzioni selezionate di volta in volta vengono inviate al multiplexer successivo. Il comando di selezione di questo multiplexer proviene dalla PLA;
- il multiplexer fornisce l'istruzione corrente al micro instruction register, il quale campiona su un fronte di clock diverso rispetto al micro instruction register. Nello specifico, il micro instruction register campiona sul fronte di salita, il micro address register sul fronte di discesa. Ciò permette di eseguire un'istruzione ad ogni colpo di clock;

- l'istruzione consiste in 1 bit di condition code, che determina la possibilità o meno di saltare da un'istruzione a un'altra, 4 bit per identificare l'istruzione seguente da eseguire e 22 bit di segnali di controllo effettivi, inviati al datapath;
- i primi 3 bit dell'indirizzo successivo vengono inviati direttamente al micro address register. Il bit meno significativo invece viene prima fatto passare per la PLA assieme al condition code;
- se il condition code è basso, il bit meno significativo viene restituito tale e quale dalla PLA al micro address register. Se invece il condition code è alto, e se i segnali di stato che comportano un salto sono asseriti, il bit più significativo viene modificato di conseguenza;
- qualora si siano determinate le condizioni per un salto, il bit meno significativo esce modificato dalla PLA e il segnale di jump validation esce asserito dalla PLA. Il di segnale jump validation entra nel pin di selezione di un altro multiplexer, denominato di bypass. Ciò fa sì che il comando di selezione dell'altro multiplexer posto a valle della micro ROM non venga preso dal micro address register ma arrivi direttamente dalla PLA, così da accelerare il processo di selezione;

Fatte tutte le considerazioni precedenti, la PLA, nel nostro caso, è semplicemente un circuito combinatorio che, dati il condition code e il segnale di stato, fa uscire il bit meno significativo invertito se entrambi sono asseriti, oppure il bit meno significativo così come è entrato in tutti gli altri casi.

Gli unici salti si verificano nel passaggio da stato di IDLE a modalità singola e dalla modalità singola alla modalità continua. In parole povere, il salto avviene quando l'unità di controllo sa che ci sono nuovi dati da elaborare in arrivo. Però, dato che il numero di registri è stato ottimizzato in modo che fossero esattamente sufficienti per svolgere un'elaborazione, non è sempre possibile accettare nuovi valori per A, B e W perché significherebbe sovrascrivere quelli vecchi. Di conseguenza, il passaggio dalla modalità singola alla modalità continua è possibile solo durante uno stato. Il condition code è dunque alto per due sole istruzioni, nello specifico la I (stato di IDLE) e la VII.

L'istruzione VII infatti si trova due istruzioni prima che i valori B_r e W_r non siano più necessari. Se il segnale di start è alto, a partire dall'istruzione successiva (VIIIC) viene abilitata la scrittura del register file in ingresso per accettare i nuovi B_r e W_r. Poi, durante quella ancora successiva, cioè esattamente quando i valori vecchi non servono più (IXC), B_r e W_r vengono sovrascritti. Durante le istruzioni successive della modalità continua, oltre a terminare l'elaborazione corrente, si continua ad abilitare il register file a riscrivere i dati vecchi man mano che non servono più. Nello specifico, B_i e W_i vengono riscritti passando all'istruzione XC, mentre A_r e A_i vengono riscritti al termine della fase in modalità continua (passaggio da istruzione XC a istruzione V).

L'istruzione da eseguire in stato di IDLE è stata posta nella prima riga della micro rom, colonna di sinistra. Dato che l'istruzione di IDLE deve richiamare sempre se stessa, in assenza di salti, i primi 3 bit dell'indirizzo seguente devono necessariamente puntare alla stessa riga. Pertanto, la prima istruzione da eseguire in modalità singola è stata posta nella stessa riga, colonna di destra. Da qui fino all'istruzione VII non c'è più la possibilità di saltare.

Per ottimizzare lo spazio della microrom abbiamo fatto in modo che le istruzioni seguenti fino alla VIII fossero disposte in maniera consecutiva da destra verso sinistra e dall'alto verso il basso. In parole povere, se stiamo eseguendo un'istruzione nella colonna di sinistra, la prossima istruzione si troverà sempre nella colonna di destra, mentre se stiamo eseguendo un'istruzione nella colonna di destra la prossima si troverà sempre nella riga successiva, colonna di sinistra.

Una volta giunti alla VII istruzione, in assenza del segnale di start, si prosegue lungo la stessa colonna, di riga in riga. Altrimenti, se il segnale di start viene asserito, si salta alla colonna di destra della riga successiva per eseguire l'istruzione denominata IXC e da lì si prosegue di riga in riga lungo la stessa colonna.

Da notare che, mentre la modalità singola termina con l'istruzione dell'ultima riga (X), la modalità continua prosegue dall'istruzione XC all'istruzione V per poter terminare l'operazione corrente, avendo però già concluso quella precedente. Strutturando la micro ROM in questo modo rimane inutilizzato lo spazio

della quarta riga, colonna di destra. Questo è uno spreco tutto sommato accettabile e inevitabile, dato che, avendo 13 istruzioni, abbiamo bisogno di una microrom a 7 righe.

Ciascuna istruzione, a livello di segnali, è strutturata come segue:

|RST|CS|WR2|WR1|WR0|SB0|SB11|SB10|LD1|LD2|LD3|LD4|M11|M10|M2|M3|M4|M5|M6|C01|C02|DONE|

RST: segnale di clear inviato a tutti i registri;
CS: segnale di chip select inviato al register file;
WRx: segnale di write enable inviato ai registri del register file;
SBx: selettori dei multiplexer che forniscono le tre uscite dal register file a partire dai registri interni;
LDx: segnali di load dei registri;
Mx: segnali di selezione dei multiplexer esterni al register file;
C0x: segnale di selezione dei sommatori, usato per definire se vada eseguita una somma o una sottrazione;
DONE: segnale di DONE, inviato al termine di ciascuna elaborazione;

Di seguito riportiamo le due micro ROM così come sono state descritte nel codice VHDL. Accanto ad ogni riga abbiamo aggiunto il simbolo con cui abbiamo identificato ciascuna istruzione.

ID	CC	Indirizzo seguente	Istruzione
I	1	0000	10000000000000000000
III	0	0011	01010000100000000000
V	0	0101	01000010011000001010 00
VII	1	1000	01000110011101110111 01
VIII	0	1010	01000000010011100001 00
IX	0	1100	00000000000000000000
X	0	0000	00000000000000010000

Tabella 2: μROM pari

ID	CC	Indirizzo seguente	Istruzione
II	0	0010	01001000000000000000
IV	0	0100	01100001101000000000
VI	0	0110	01000100101000000010 10
N/A	0	0000	00000000000000000000
VIIIC	0	1011	01001000010011100001 00
IXC	0	1101	01010000100000000000
XC	0	0100	01100001101000000100 00

Tabella 3: μROM dispari

2.1 ASM Chart

Di seguito viene riportata l'ASM chart del nostro progetto. Ogni riquadro contiene il nome identificativo dello stato ed i segnali di controllo asseriti oppure settati ad un valore diverso da 00, se sono a due bit, durante ciascuno stato.

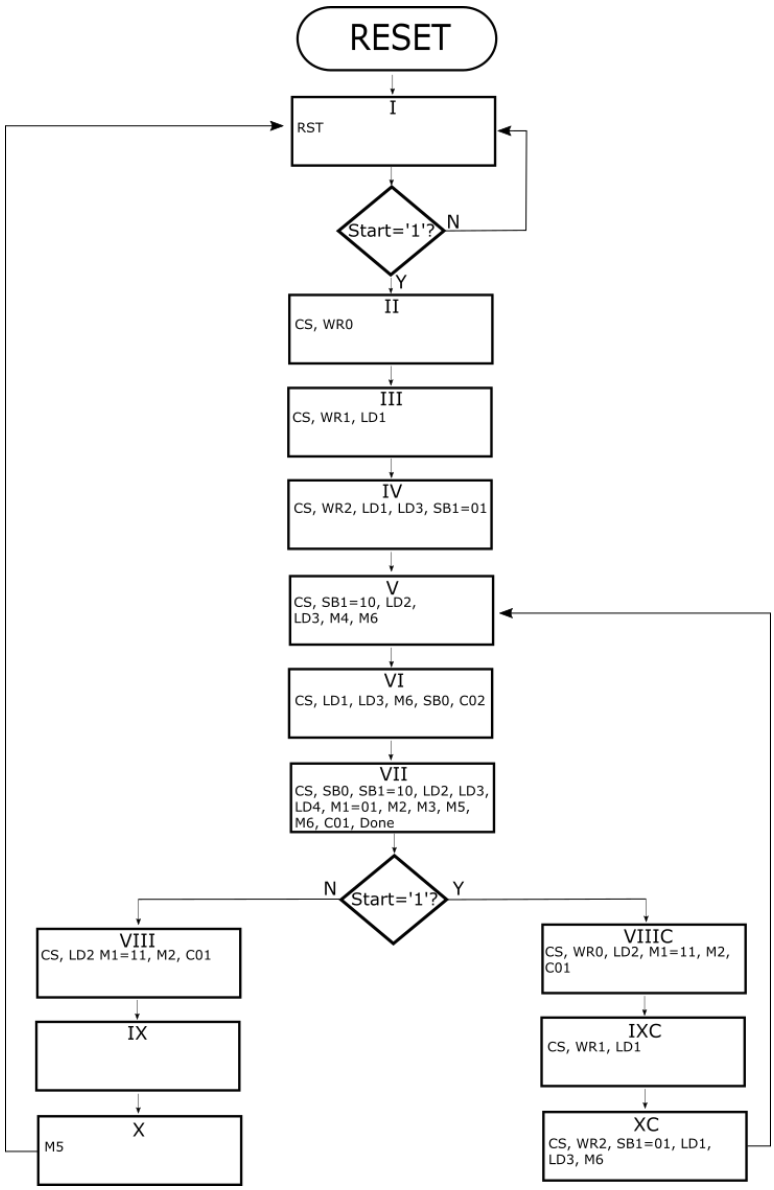

Figura 4: ASM chart

Nelle prossime sezioni descriveremo brevemente ciascuna istruzione. Nella lettura di questa sezione bisogna tenere conto che, qualora un segnale non venga menzionato, significa o che non è asserito o che è pari a "00" se si tratta di un segnale a due bit. Inoltre, a meno che la descrizione non dica diversamente, ogni istruzione punta a quella immediatamente successiva, considerato il sistema di numerazione romano.

2.2 I

Indirizzo: 0000
Stato di IDLE, con clear alto e resto dell'istruzione contenente solo 0. Il segnale di clear è asincrono e raggiunge tutti i registri nel datapath, resettandoli. Questo stato punta sempre verso se stesso e ha il condition code asserito, dunque, in presenza del segnale di start, la prossima istruzione sarà quella nella colonna di destra della stessa riga, ossia la II, rappresentante il primo stato della modalità singola.

2.3 II

Indirizzo: 0001
Primo stato della modalità singola. Vengono alzati solo i segnali di Chip Select e WR0, uno dei 3 segnali di write enable del Register File, per lasciare campionare, negli appositi registri, i primi due dati, ossia B_r e W_r. Il primo dato è atteso dal canale Data In 0 e il secondo dal canale Data In 1.

2.4 III

Indirizzo: 0010
Secondo stato della modalità singola. Chip Select è ancora asserito e vengono attivati altri due registri nel Register File per campionare B_i e W_i, in arrivo rispettivamente dai canali Data In 0 e Data In 1. Per fare ciò, WR1 viene asserito. Il segnale di load del registro 1 viene asserito così da poter campionare W_r, passante per il Bus 1. A tale scopo, il multiplexer del register file che selezionano cosa viene immesso nel bus due è settato a 00. Nell'istruzione si tratta dei segnali SB10 e SB11, e per praticità d'ora in poi questi due segnali saranno definiti come SB1.

2.5 IV

Indirizzo: 0011
Terzo stato della modalità singola. Chip Select è ancora asserito. Viene asserito WR2 per poter campionare nei due registri rimanenti A_r e A_i, in ingresso rispettivamente dal canale Data In 1 e Data In 0. Il multiplexer che seleziona cosa fare entrare nel Bus 1 è girato per far passare W_i, che a sua volta sarà campionato dal registro 1. Per fare ciò, il segnale SB1 è settato a 01 e il segnale LD1 è ancora asserito. Il segnale di load del registro 3 (LD3) viene asserito così da poter campionare M1, in uscita dal moltiplicatore, dato che all'altro ingresso esso riceve B_r visto che il selettore del bus 0 SB0 è a 0. Per permettere il salvataggio di M1, il selettore del multiplexer 6 (M6) rimane a 0.

2.6 V

Indirizzo: 0100
Quarto stato della modalità singola. Chip select è ancora asserito, SB1 passa a 10, così da far transitare nel Bus 1 A_r. LD2 e LD3 vengono asseriti così da campionare il risultato in uscita dal Sommatore 1 (Σ 3) all'interno del registro R2 e il risultato in uscita dal sommatore 2 (Σ1) all'interno del registro R3. A tale scopo, i selettori dei multiplexer 4 e 6 (M4 e M6) sono settati ad 1, così da far entrare A_r all'interno del Sommatore 2 per poter eseguire Σ 1 e condurlo poi all'interno del registro 3.

2.7 VI

Indirizzo: 0101
Quinto stato della modalità singola. Chip Select è ancora asserito. LD1 viene asserito per campionare W_r all'interno del registro 1 (pertanto, SB1 è settato a 00 per far passare W_r all'interno del Bus 1). SB0 è settato a 1 per far entrare B_i all'interno del moltiplicatore. LD3 è asserito per poter campionare Σ2 all'interno di R3 (M6 infatti rimane settato a 1). Dato che Σ2 è una sottrazione, C02 è asserito (si tratta del comando di selezione dell'operazione eseguita dal secondo sommatore). Notare che i multiplexer 3 e 4 hanno il selettore a 0 per fare entrare all'interno del secondo sommatore Σ1 e M2.

2.8 VII

Indirizzo: 0110
Sesto stato della modalità singola. Dato che dopo la prossima istruzione (ossia, dall'istruzione IX in poi)

i dati in ingresso A, B e W non sono più necessari, il condition code è asserito cosicché, in presenza del segnale di start, si possa passare alla modalità continua. Più precisamente, in tale condizione si passerà all'indirizzo 1001 (VIIIC) anziché 1000 (VIII). Chip Select è ancora asserito, SB0 è settato a 1 per far entrare B_i all'interno del moltiplicatore e SB1 è viene impostato a 10 per far transitare A_r nel bus 1. LD2, LD3 e LD4 vengono asseriti così da poter campionare nei rispettivi registri $\Sigma 5$, $\Sigma 4$ e A'_r. I selettori dei rispettivi multiplexer M1, M2, M3, M5 ed M6 sono impostati a 01, 1, 1, 1, 1 così da far entrare:

- A_r, proveniente dal Bus 1 e passante per il blocco di shift, in modo tale che venga moltiplicato per 2, all'interno del Sommatore 1. A tale scopo, il bit più significativo di M1 viene inviato al selettore posto a valle del blocco moltiplicatore per 2 per selezionare tale ingresso in uscita;
- $\Sigma 2$ all'interno del Sommatore 1;
- $\Sigma 3$ all'interno del Sommatore 2;
- $\Sigma 2$ all'interno del circuito di arrotondamento, così da fornire in ingresso al Registro 4 A'_r;
- $\Sigma 4$, in uscita dal Sommatore 2, all'interno del Registro 3;

C01 viene asserito, dato che l'operazione eseguita dal sommatore 1, $\Sigma 5$, è una sottrazione. Infine, dato che a partire dalla prossima istruzione (sia che si tratti di VIII che di VIIIC) si faranno uscire in sequenza B'_r, B'_i e A'_r e A'_i, il segnale di Done viene alzato, per indicare che nei prossimi 3 colpi di clock verranno inviati i risultati dell'operazione corrente.

2.9 VIII

Indirizzo: 1000
Settimo stato della modalità singola. Chip Select è ancora asserito, LD2 è asserito così da poter campionare $\Sigma 6$ all'interno del registro 2. M1 e M2 sono impostati rispettivamente a 11 e 1, così da far entrare A_i (moltiplicato per due) e $\Sigma 4$ all'interno del Sommatore 1. C01 è ancora asserito dato che $\Sigma 6$ è a sua volta una sottrazione. A questo punto è possibile campionare B'_r, dato che $\Sigma 5$, contenuto nel registro 2, viene fatto passare lungo il circuito di arrotondamento e il risultato è fornito direttamente all'uscita Data Out 0.

2.10 IX

Indirizzo: 1010
Ottavo stato della modalità singola. Dato che da qui in avanti il Register file non viene più letto né tanto meno scritto, Chip Select non è più asserito. L'intera istruzione consiste di comandi non asseriti in quanto l'unica operazione da svolgere è far passare $\Sigma 6$, contenuto nel Registro 2, all'interno del circuito di arrotondamento, così da far uscire B'_i lungo Data Out 0 e permetterne il campionamento.

2.11 X

Indirizzo: 1100
Nono ed ultimo stato della modalità singola. M5 viene settato a 1 così da far entrare $\Sigma 4$ nel circuito di arrotondamento. A questo punto, dunque, A'_r e A'_i vengono fatti uscire rispettivamente da Data Out 1 e da Data Out 0 ed è possibile campionarli. Logicamente, l'istruzione successiva puntata da questo stato è quella di IDLE (indirizzo 0000).

2.12 VIIIC

Indirizzo: 1001
Primo stato della modalità continua. Identica alla VIII, salvo che Chip Select è ancora attivo e WR0 viene asserito, così da poter campionare i nuovi B_r e W_r nei rispettivi registri del Register file.

2.13 IXC

Indirizzo: 1011
Secondo stato della modalità continua. Identica alla IX, salvo che Chip Select è ancora attivo e WR1 viene asserito, così da poter campionare i nuovi B_i e W_i nei rispettivi registri del Register file.

2.14 XC

Indirizzo: 1101
Terzo ed ultimo stato della modalità continua. Identica alla X, salvo che Chip Select è ancora attivo e WR2 viene asserito, così da poter campionare i nuovi A_r e A_i nei rispettivi registri del Register file. L'operazione successiva è la V, dunque si torna (nominalmente) alla modalità singola, così da poter completare l'istruzione corrente. Va da sé che sarà nuovamente possibile entrare in modalità continua durante l'esecuzione dell'istruzione VII.

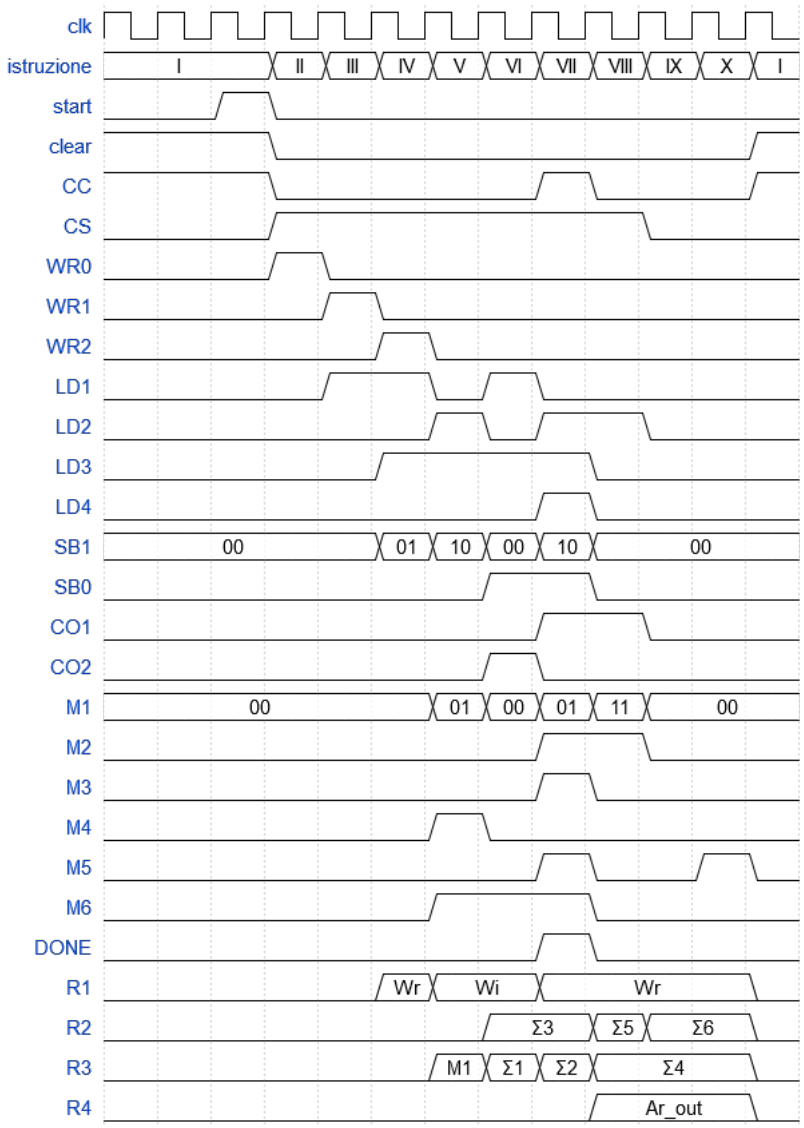

Figura 5: Timing diagram modalità singola

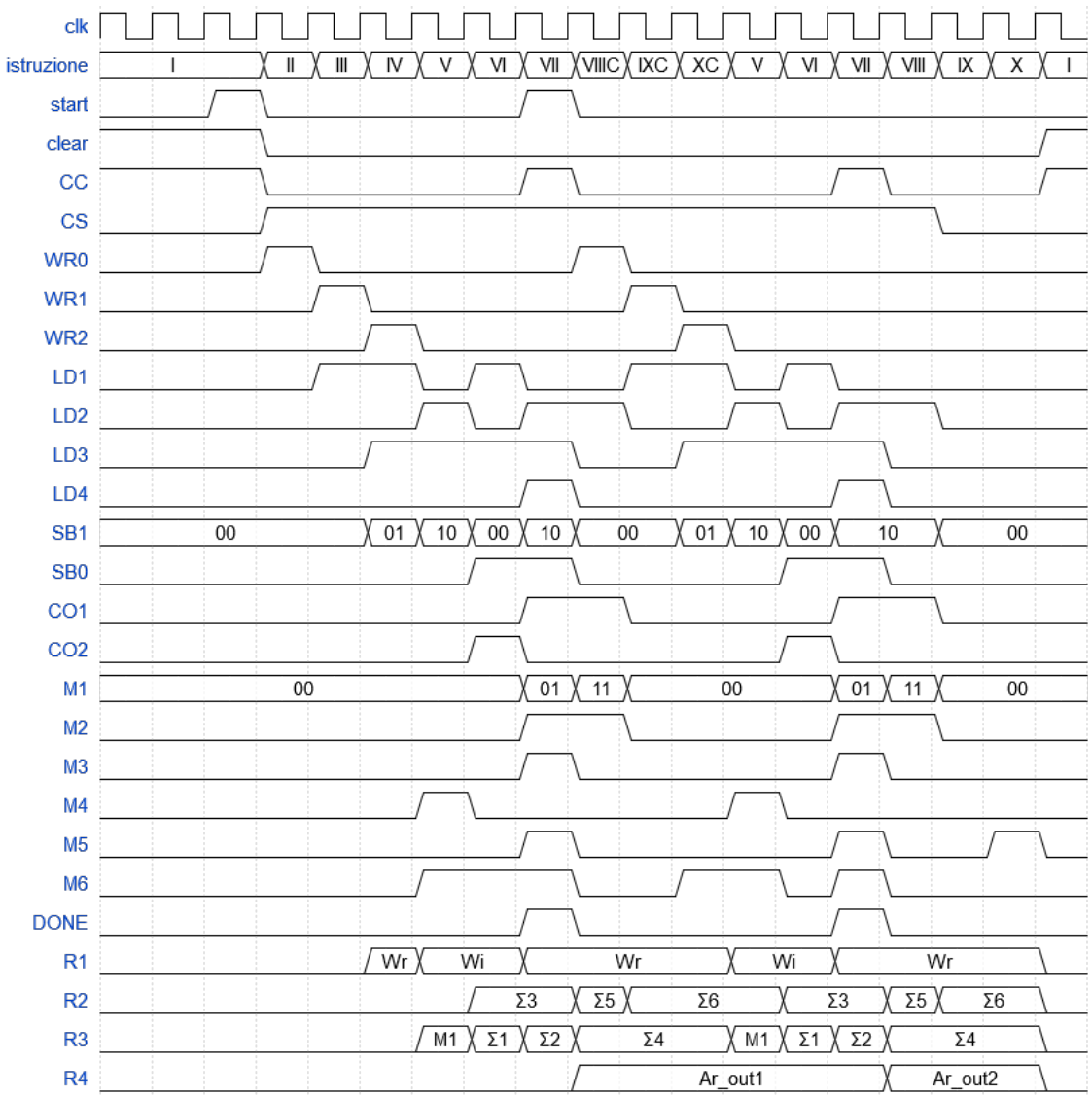

Figura 6: Timing diagram modalità continua

3 Test del progetto

Per validare il lavoro da noi svolto abbiamo selezionato una serie di vettori test da immettere nella butterfly per verificarne il corretto funzionamento. In particolare, abbiamo scelto due vettori realizzati in maniera completamente casuale e due aventi valori ai limiti del range consentito. Nello specifico, A e B sono stati scelti con parte reale e immaginaria in modulo pari a $0.25\text{-}2^{-15}$, mentre W è stato scelto a sua volta pari al massimo e al minimo valore rappresentabile, ossia 1 e -1.

E' stato svolto sia un test in modalità singola, ossia fornendo un singolo vettore, che in modalità continua, fornendo 3 vettori di fila. Il testbench denominato butterfly_tb, fra i file in allegato, è stato utilizzato proprio per testare tali vettori. I risultati soddisfano pienamente le aspettative di progetto, nel senso che entrambe le modalità funzionano in maniera congrua rispetto all'algoritmo da noi ideato.

Dall'istruzione X si torna allo stato di IDLE, ossia la modalità singola termina come dovrebbe, e dall'istruzione XC si prosegue con la V, ossia terminando le istruzioni della modalità continua e proseguendo

con l'elaborazione del dato corrente. Inoltre è stato dimostrato che, in presenza del segnale di start, si passa sempre dall'istruzione VII alla VIIIC (modalità continua).

Il risultato a 31 bit è sempre corretto, mentre il risultato a 16 bit ha un errore, in modulo, sempre minore del massimo errore ottenibile con troncamento e arrotondamento di tipo "rounding to nearest even". E' stata validata l'operazione del circuito di arrotondamento, nel senso che esso arrotonda per eccesso sempre e solo nei casi giusti. Infine è stato dimostrato che, scegliendo dei valori pari al limite massimo consentito, non si hanno overflow.

Di seguito riportiamo delle tabelle contenenti i vettori test da noi scelti per validare il nostro progetto, i risultati ottenuti in uscita prima dell'arrotondamento, i risultati ad arrotondamento avvenuto e l'errore commesso in ciascun caso. Qualora fosse avvenuto un arrotondamento per eccesso, il risultato a 16 bit appare in corsivo.

Ciascun risultato a 31 bit è stato validato confrontandolo con il risultato ottenuto svolgendo la stessa operazione tramite una calcolatrice scientifica *Casio fx-82ES PLUS*. Ciascun numero è riportato sia in forma frazionaria che esadecimale.

3.0.1 Test in modalità singola

Per questo test è stato fornito un singolo vettore generato casualmente.

Rapp	A_r	A_i	B_r	B_i	W_r	W_i
FRAC	-0.109405517578125	-0.15576171875	-0.00048828125	0.125213623046875	0.062835693359375	-0.0078125
HEX	F1FF	EC10	FFF0	1007	080B	FF00

Tabella 4: Vettori test

Rapp	A'_r	A'_i	B'_r	B'_i
FRAC	-0.108457967638969	-0.147890019230544	-0.11035306751728	-0.163633418269455
HEX	790F0650	7688F84D	78EFF9B0	758707B3

Tabella 5: Risultato a 31 bit dei vettori test

Rapp	A'_r	A'_i	B'_r	B'_i
FRAC	-0.10845947265625	*-0.14788818359375*	*-0.1103515625*	-0.16363525390625
HEX	F21E	ED12	F1E0	EB0E

Tabella 6: Risultato a 16 bit del vettore test

A'_r	A'_i	B'_r	B'_i
-0.000001505017281	0.000001835636794	0.00000150501728	-0.000001835636795

Tabella 7: Errore risultato (risultato a 16 bit - risultato a 31 bit)

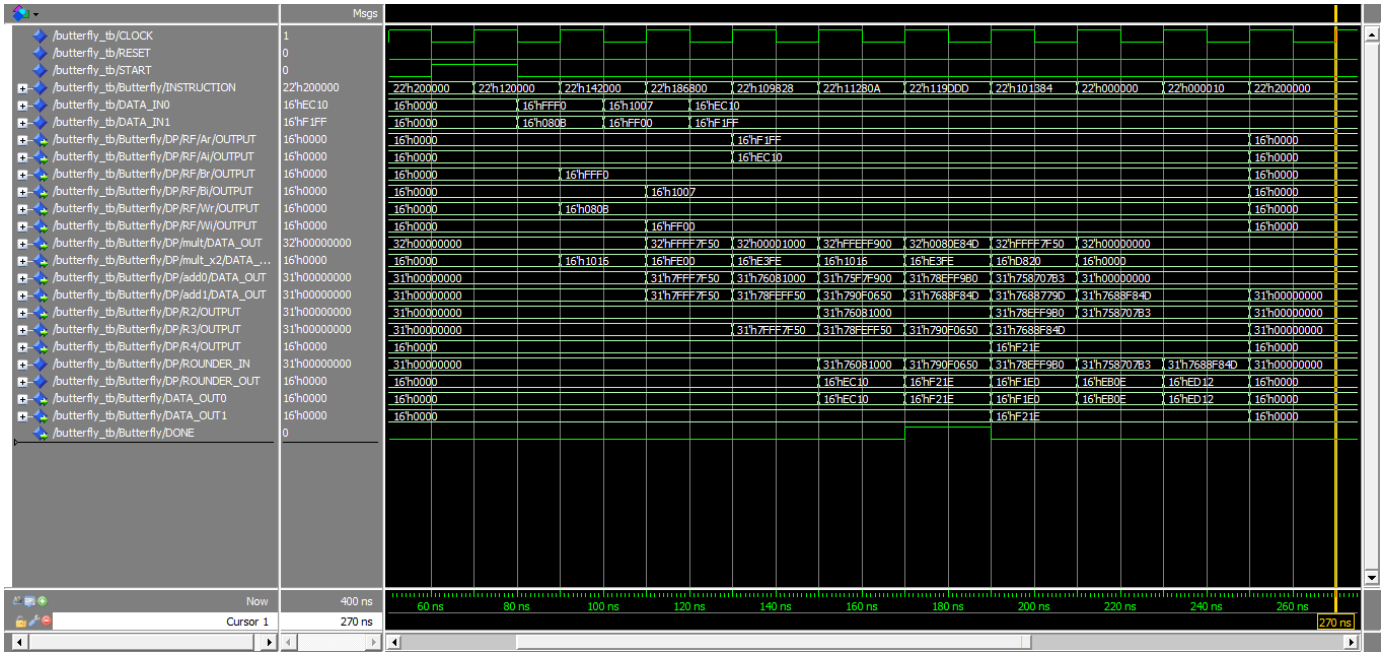

Figura 7: Test in modalità singola. Ricordiamo che, dopo l'asserimento del segnale di DONE, vediamo B_r e poi B_i uscire lungo DATA_OUT_0, dopo di che vediamo A_r e A_i uscire in contemporanea rispettivamente lungo DATA_OUT_1 e DATA_OUT_0.

3.0.2 Test in modalità continua

Per questo test sono stati forniti 3 vettori: uno generato casualmente e due generati in modo da avere valori al limite fra quelli consentiti per evitare overflow.

Rapp	A_r	A_i	B_r	B_i	W_r	W_i
FRAC	0.2391052246	-0.1483764648	0.163848877	-0.2160949707	-0.7873840332	0.8538208008
	0.2499694824	0.2499694824	0.2499694824	0.2499694824	0.9999694824	0.9999694824
	-0.2499694824	-0.2499694824	-0.2499694824	-0.2499694824	-1	-1
HEX	1E9B	ED02	14F9	E457	9B37	6D4A
	1FFF	1FFF	1FFF	1FFF	7FFF	7FFF
	E001	E001	E001	E001	8000	8000

Tabella 8: Vettori test

Rapp	A'_r	A'_i	B'_r	B'_i
FRAC	0.294599616	0.1616708441	0.1836108332	-0.4584237738
	0.2499694824	0.7498931903	0.2499694824	-0.2499542255
	-0.2499694824	0.2499694824	-0.2499694824	-0.7499084473
HEX	12DAB859	0A58D0AB	0BC047A7	62A92F55
	0FFF8000	2FFE4002	0FFF8000	7000BFFE
	70008000	0FFF8000	70008000	50018000

Tabella 9: Risultato a 31 bit dei vettori test

Rapp	A'$_r$	A'$_i$	B'$_r$	B'$_i$
FRAC	0.2945861816	*0.1616821289*	*0.1836242676*	-0.4584350586
	0.2499694824	*0.7499084473*	0.2499694824	0.2499694824
	-0.2499694824	0.2499694824	-0.2499694824	-0.7499084473
HEX	25B5	*14B2*	*1781*	C552
	1FFF	*5FFD*	1FFF	E001
	E001	1FFF	E001	A003

Tabella 10: Risultato a 16 bit dei vettori test

A'$_r$	A'$_i$	B'$_r$	B'$_i$
-0.00001343432814	0.00001128483564	0.00001343432814	-0.00001128483564
0	0.00001525692642	0	-0.00001525692642
0	0	0	0

Tabella 11: Errore risultati (risultato a 16 bit - risultato a 31 bit)

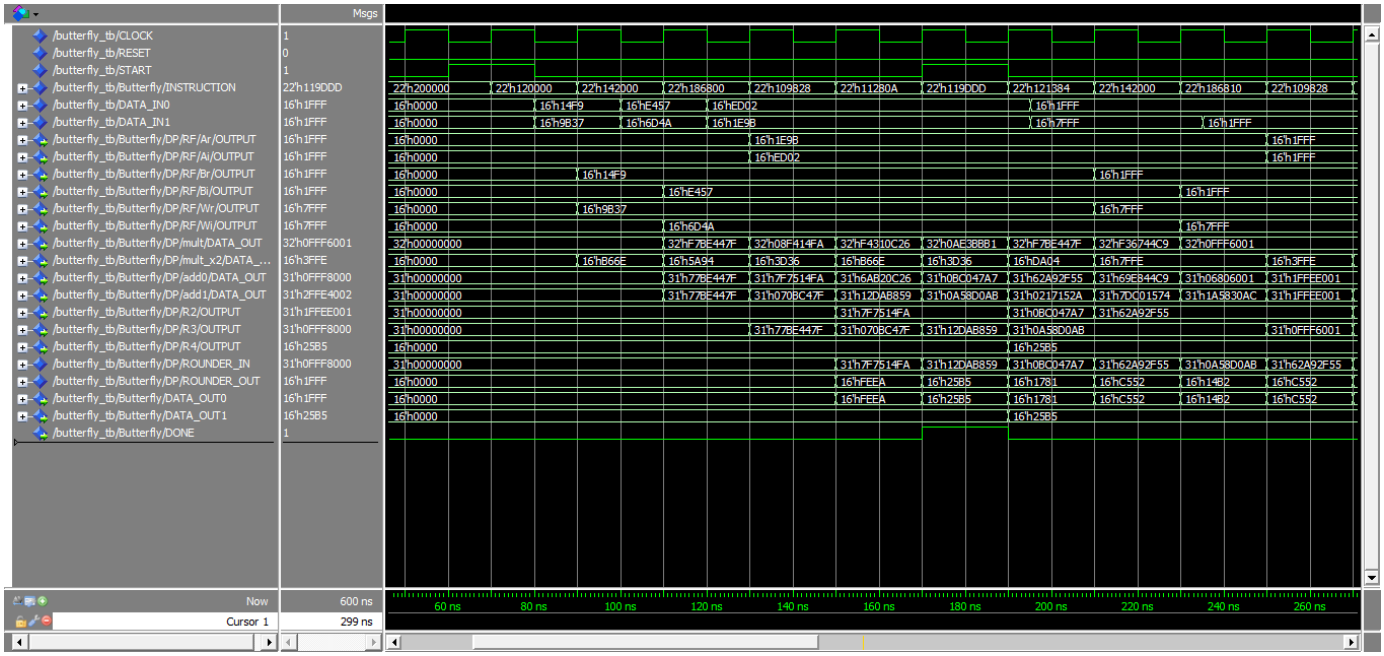

Figura 8: Test in modalità continua

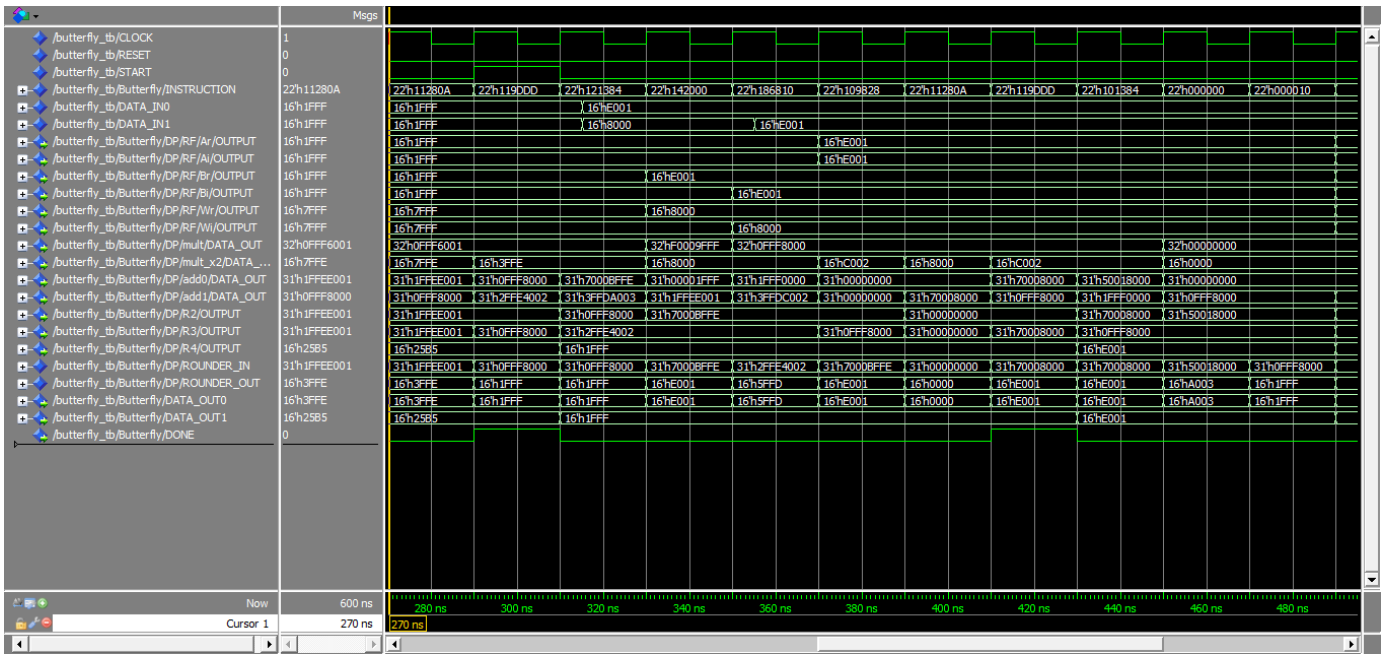

Figura 9: Test in modalità continua

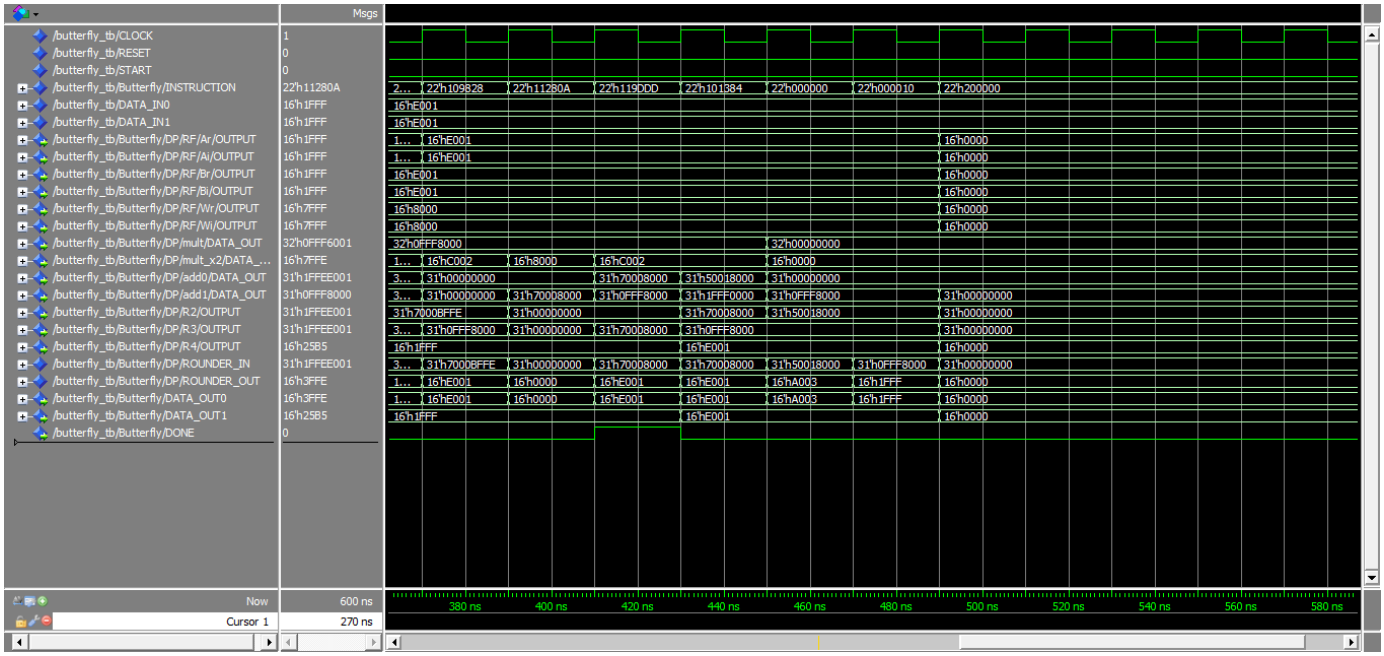

Figura 10: Test in modalità continua

4 Conclusione

Il progetto è stato realizzato nella maniera più ottimizzata possibile. Il numero di ingressi e di uscite è stato settato a due perché in questo modo otteniamo contemporaneamente un numero sufficiente di dati (B_r e W_r) per poter iniziare il prima possibile a svolgere la prima moltiplicazione (M1).

Togliere un bus avrebbe significato rallentare l'esecuzione dell'algoritmo di un colpo di clock, mentre aggiungerne altri non avrebbe comportato alcun considerevole vantaggio dato che:

- anche avendo disponibile in contemporanea, ad esempio, B_r, W_r e A_r, non potremmo comunque eseguire $\Sigma 1$ in contemporanea ad M1. Infatti, a meno che il moltiplicatore non fosse particolarmente performante (e dunque costoso), non è possibile eseguire moltiplicazione e somma nello stesso colpo di clock senza ridurre di molto la frequenza, peggiorando le prestazioni;
- anche avessimo disponibili in contemporanea B_r, B_i, W_r e W_i non potremmo comunque eseguire in contemporanea M1 ed un'altra moltiplicazione, dato che abbiamo un solo moltiplicatore a disposizione;

I registri usati sono in numero superiore rispetto al numero massimo delle variabili temporanee richiesto dall'algoritmo (utilizziamo 4 registri più 6 all'interno del Register File, ossia 10, 3 in più rispetto al numero di registri strettamente necessario).

Ciò è stato fatto perché da specifiche ci veniva richiesto di ottimizzare anche l'uso dei bus. Per utilizzare solo 7 registri avremmo dovuto impiegare un numero di connessioni e di multiplexer superiore a quello effettivamente utilizzato, così da complicare il progetto e al contempo da rendere necessario l'uso di ancora più risorse.

Il registro 1 ci permette di risparmiare un bus, dato che dal register file vengono utilizzati al massimo 4 dati durante un'istruzione (istruzioni V e VII). Pertanto, in sua assenza sarebbe stato necessario utilizzare 4 bus. Ridurre ulteriormente il numero di bus avrebbe significato aumentare il numero di registri interni al datapath e al contempo aumentare il numero di istruzioni da eseguire per completare l'algoritmo, riducendo quindi le performance.

Inoltre, il nostro progetto prevede l'integrabilità fra più butterfly dello stesso tipo, tenendo conto di due fattori. Innanzitutto, il segnale di Done di ciascuna butterfly può essere inviato direttamente all'ingresso del segnale di start della butterfly successiva. Infatti, qualora la prima butterfly fosse in modalità continua, il segnale di done viene sempre alzato quando la butterfly successiva è allo stato VII, ossia quando essa è sensibile al segnale di start, come dimostra la seguente immagine.

I	II	III	IV	V	VI	**VII**	VIIIC	IXC	XC	V	VI	**VII**	VIIIC	IXC	XC	V	VI	**VII**
I	I	I	I	I	I	I	II	III	IV	V	VI	**VII**	VIIIC	IXC	XC	V	VI	**VII**

Figura 11: Evoluzione degli stati di due butterfly in cascata. La riga di sopra rappresenta lo stato della butterfly posta a monte, quella di sotto della butterfly posta a valle.

Infine, come già detto, l'uscita di A' e B' lungo i canali di output è stata pensata in modo che tali canali si possano connettere direttamente ai rispettivi canali di input di una butterfly posta immediatamente dopo, ossia Data Out 0 con Data In 0 e Data Out 1 con Data In 1. Difatti, B_r, B_i e A_i sono attesi all'input Data In 0 ed escono da Data Out 0, mentre A_r, atteso al canale Data In 1, esce da Data Out 1. Chiaramente, il canale di ingresso Data In 1 della butterfly successiva deve essere preceduto quantomeno da un multiplexer per poter scegliere se accettare A_r o il tweedle factor, ma questo aspetto esula dagli obiettivi di questo progetto.