Politecnico di Torino

Dipartimento di Elettronica e Telecomunicazioni

Corso di laurea magistrale in Ingegneria Elettronica

Sistemi Digitali Integrati

Operazione "San Silvestro"

FFT - Butterfly

Prof. Maurizio Zamboni

Ribaldone Elia 265613

Trufini Andrea 262848

Fabiano Christian 256852

a.a. 2018/19

Indice

Sommario

Descrizione

Il progetto si è basato sullo sviluppo di una unità hardware digitale in grado di eseguire la **Butterfly**, l'elemento base necessario per la realizzazione di una FFT classica. Le operazioni eseguite dall'unità Butterfly sono somme algebriche e moltiplicazioni su numeri complessi. La progettazione della macchina è stata effettuata mediante l'utilizzo di un moltiplicatore e di due sommatori, basando l'unità di controllo su una struttura microprogrammata. Sia i sommatori che il moltiplicatore sono costituiti da un livello di *pipeline*. In particolare, la macchina è stata progettata in modo tale che l'ordine dei dati in ingresso sia uguale a quello dei dati in uscita per permettere la connessione in serie di più blocchetti; inoltre il *funzionamento continuo* con più unità Butterfly connesse in serie è stato implementato anche in caso di dati d'ingresso sfasati temporalmente.
Il Successivamente più unità di elaborazione Butterfly sono state utilizzate per la realizzazione di una **FFT completa**.

Caratteristiche

La Butterfly progettata ha le seguenti caratteristiche:

- **Frequenza massima:** La frequenza massima del circuito è limitata dal percorso combinatorio più lungo, nel nostro caso si ha una frequenza massima di 154.08 MHz;
- **Throughput:** Con la frequenza ottenuta è possibile processare $38.5M$ di coppie di campioni in ingresso, ovvero una coppia di campioni ogni $26\,ns$
- **Modalità continua sfasata:** La butterfly accetta in ingresso dei nuovi valori dopo 4, 5, 6 e 7 colpi di clock dall'utimo start, dunque oltre alla modalità di funzionamento continua (ogni 4 colpi di clock) il funzionamento è garantito anche in qualsiasi altro istante arrivi lo start (a patto che sia dopo il 3° colpo di clock dal precedente).
- **Valori in uscita:** In uscita i valori sono scalati di un fattore 4.

Le caratteristiche dell'FFT progettata tramite l'utilizzo di Butterfly ha le seguenti caratteristiche:

- **Frequenza massima:** 150.12 MHz;
- **Throughput:** Il tempo di processamento di un dato in modalità singola (latenza) è di $175ns$, se i dati vengono dati in sequenza si hanno dati processati in uscita ogni 4 colpi di clock ossia $26ns$.
- **Valori in uscita:** Poiché è realizzata da tre schiere di butterfly si ha in uscita uno scalamento in uscita di 64.

NOTA: la frequenza massima è letta dal *TimeAnalyzer* di Quartus II. Poiché alcuni componenti del circuito non sono descritti con vista architetturale *structural* ma invece *behavioural*, non è un valore attendibile. Tuttavia l'ordine di grandezza è ragionevole.

Capitolo 1

Specifiche di progetto

Il circuito da progettare è, da specifiche, un circuito "butterfly", cioè un circuito che effettui parte della computazione necessaria per eseguire il calcolo di una FFT.

1.1 Introduzione

1.1.1 Discrete Fourier Transform [DFT]

In gran parte dei circuiti di condizionamento digitali, i campioni acquisiti da un convertitore ADC vengono elaborati tramite una trasformata "discreta" di Fourier (DFT), generando lo spettro in frequenza dei campioni di uscita:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi \frac{n \cdot k}{N}}$$

1.1.2 Algoritmo di Cooley-Tukey [FFT]

Se il numero di campioni da elaborare è una potenza di due, è possibile eseguire un algoritmo che riduce la complessità computazionale della trasformata:

$$O(N^2) => O\big(N \log(N)\big)$$

Esprimendo l'indice n come $n = rn' + i$, si riesce a scrivere in forma ricorsiva la DFT effettuando delle semplificazioni man mano che si rende la DFT più piccola. La trasformata di Fourier discreta, utilizzando tale algoritmo, prende il nome di Fast Fourier Transform [FFT].

1.1.3 Radix-2 Butterfly Diagram

Scegliendo come r (radix) il numero 2, si separa l'elaborazione dei campioni pari da quella dei campioni dispari. Analizzando passo passo le operazioni che l'algoritmo prevede è evidente come dei calcoli fatti in passi precenti possano essere riutilizzati per il calcolo di campioni successivi. L'operazione atomica da replicare, per via della forma grafica, viene chiamata Butterfly.

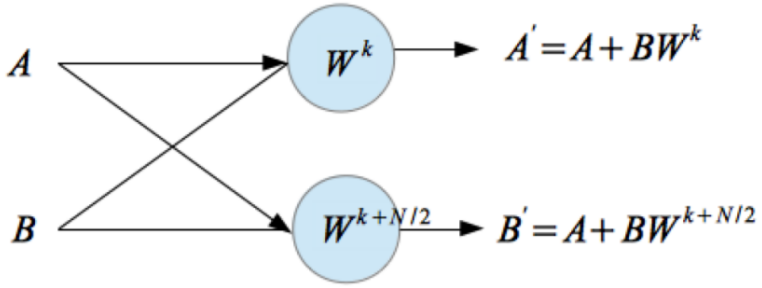


Figura 1.1: Butterfly

Replicando il circuito base in modo che le uscite della butterfly siano gli ingressi di un'altra butterfly è possibile, se disposte secondo il giusto ordine (che varia a seconda del numero di campioni da trasformare), ottenere dalla schiera di butterfly finale i campioni trasformati.

1.1.4 Dati di ingresso e dati di uscita

Esprimendo i dati di uscita come $A^{'} = A^{'}_R + jA^{'}_I$ e $B^{'} = B^{'}_R + j\,B^{'}_I$ come parte reale e immaginaria, le operazioni da fare una volta esplicitati tutti i calcoli sono le seguenti:

$$A^{'}_R = A_R + B_R W_R - B_I W_I$$
$$A^{'}_I = A_I + B_R W_I + B_I W_R$$
$$B^{'}_R = -A^{'}_R + 2\,A_R$$
$$B^{'}_I = 2\,A_I - A^{'}_I$$

1.2 Specifiche

1.2.1 Parallelismo dei dati

Il numero di bit su cui sono rappresentati le parti reali e immaginarie dei dati è di 16 bit. Per far si che non ci siano overflow non viene usata la tecnica dei bit di guarda ma è richiesto l'utilizzo dell' *Unconditional Floating Point Scaling*: all'interno del circuito si usa il parallelismo necessario per poi alla fine scalare il risultato e tenere conto del fattore di scala. **Rispetto alla soluzione con i bit di guardia**, questa tecnica non limita la dinamica di ingresso dei dati e dunque è migliore: la dinamica di ingresso si ridurrebbe di due bit per ogni stadio di butterfly presente nella FFT. Se il numero di campioni da elaborare è 16, perdendo due bit per ogni butterfly e considerando che ne sono necessarie quattro, si riduce la dinamica di ingresso di 8 bit.

1.2.2 Formato numerico

I dati su 16 bit sono rappresentati in *fractional point*, dunque 1 bit di segno e 15 bit frazionari (Q0.15). Tenendo conto dei possibili overflow all'uscita, **prima dello scalamento**, i dati hanno rappresentazione Q2.15, che scalato diventa nuovamente Q0.15 ma con un **fattore di scala 4**.

1.2.3 Arrotondamento

I dati in uscita devono essere arrotondati tramite il *Rounding to nearest even*. Nella sezione **DATAPATH** viene dettagliato il funzionamento del blocco di rounding.

1.2.4 Hardware utilizzabile

Il circuito deve essere progettato usando non più di un moltiplicatore e due sommatori. Tutti e tre hanno uno stadio di pipeline. Per il blocco logico che effettua l'arrotondamento non sono presenti vincoli. L'unità di controllo deve essere microprogrammata e deve avere un sequenziatore a indirizzamento esplicito (**LATE STATUS**).

Capitolo 2

Progetto

2.1 Scelte di progetto

Le specifiche date lasciano un ampio margine sulle decisioni progettuali. Durante tutte le fasi di progetto, prima di prendere una scelta, è stato valutato l'impatto di tale scelta su:

- Flessibilità di funzionamento dell'algoritmo
- Ritardi combinatori e frequenza massima di lavoro
- Latenza dall'ingresso all'uscita dei campioni
- Numero di interconnessioni
- Dimensioni µROM
- Consumo energetico
- Complessità e facilità di debug del circuito

Molti di questi fattori sono in contrasto tra di loro perciò in alcuni casi si è cercata una soluzione di compromesso. Seguono i dettagli sulle scelte fatte.

2.1.1 Flessibilità di funzionamento dell'algoritmo

Si è scelto di garantire il funzionamento della Butterfly in qualsiasi istante arrivi un segnale di *START* (distanziato da un minimo di 4 colpi di clock dal precedente, che in modalità di funzionamento continua rappresenta il throughput della macchina). In questo modo se si decide di passare da funzionamento "singolo" a "continuo" non bisogna aspettare che sia terminata l'elaborazione singola prima di dare un nuovo *START* e viceversa, evitando tempi morti tra una modalità e l'altra. Proprio perché la macchina è pipelinata e già deve poter lavorare in modalità continua, la scelta di aumentare la flessibilità **non ha** avuto alcun impatto sul datapath, sui ritardi o sul numero di interconnessioni ma soltanto un numero maggiore di stati nell'unità di controllo. L'incremento degli stati **non ha** tuttavia aumentato le dimensioni della µROM (non essendo aumentato il numero di bit, poiché erano presenti locazioni vuote).

2.1.2 Ritardi combinatori e frequenza massima di lavoro

Nel datapath viene dettagliato come l'utilizzo di registri di pipeline tra i blocchi combinatori aumenti la frequenza massima di lavoro del circuito.

2.1.3 Latenza dall'ingresso all'uscita dei campioni e throughput

Poiché pipelinata, la latenza **Start - Done** nell'esecuzione singola è di 9 colpi di clock. Tuttavia, si è fatto in modo che in modalità continua, mettendo in serie più circuiti butterfly (nell'ottica del calcolo di una FFT completa), ogni 4 colpi di clock viene completata l'elaborazione dei due ingressi e iniziata quella dei successivi. Inoltre, per diminuire la latenza, le equazioni della butterfly sono state riorganizzate in modo tale che i primi dati in uscita sono i primi a essere richiesti dalla butterfly successiva.

2.1.4 Numero di interconnessioni

Il datapath è stato progettato senza BUS globali (che implicherebbero alte capacità parassite, tanta area occupata e costi/consumi maggiori). Le interconnessioni presenti localmente sono soltanto di tipo *diretto*, pertanto non posso essere presenti conflitti. Non c'è dunque motivo di incrementare il numero di bus presenti, in quanto ci sarebbero solo svantaggi: il massimo delle prestazioni è già stato raggiunto, il limite non sono le interconnessioni ma il moltiplicatore.

2.2 Fasi progettuali

2.2.1 Modifica equazioni Butterfly

Dalle analisi fatte sulle *data dependencies*, si è scoperto che i primi dati ad essere utilizzati non sono in realtà i numeri A'_R e A'_I bensì B'_R e B'_I. Nelle equazioni mostrate in precedenza inoltre, i dati B'_R e B'_I dipendevano da A'_R e A'_I. Dunque, si è fatto in modo che i primi risultati delle elaborazioni fossero B'_R e B'_I, in modo da semplificare il progetto. Le nuove equazioni sono:

$$B^{'}{}_R = A_R - B_R\,W_R + B_I\,W_R$$

$$B^{'}{}_I = A_I - B_R\,W_I - B_I\,W_R$$

$$A^{'}{}_R = 2\,A_R - B^{'}{}_R$$

$$A^{'}{}_I = 2\,A_I - B^{'}{}_I$$

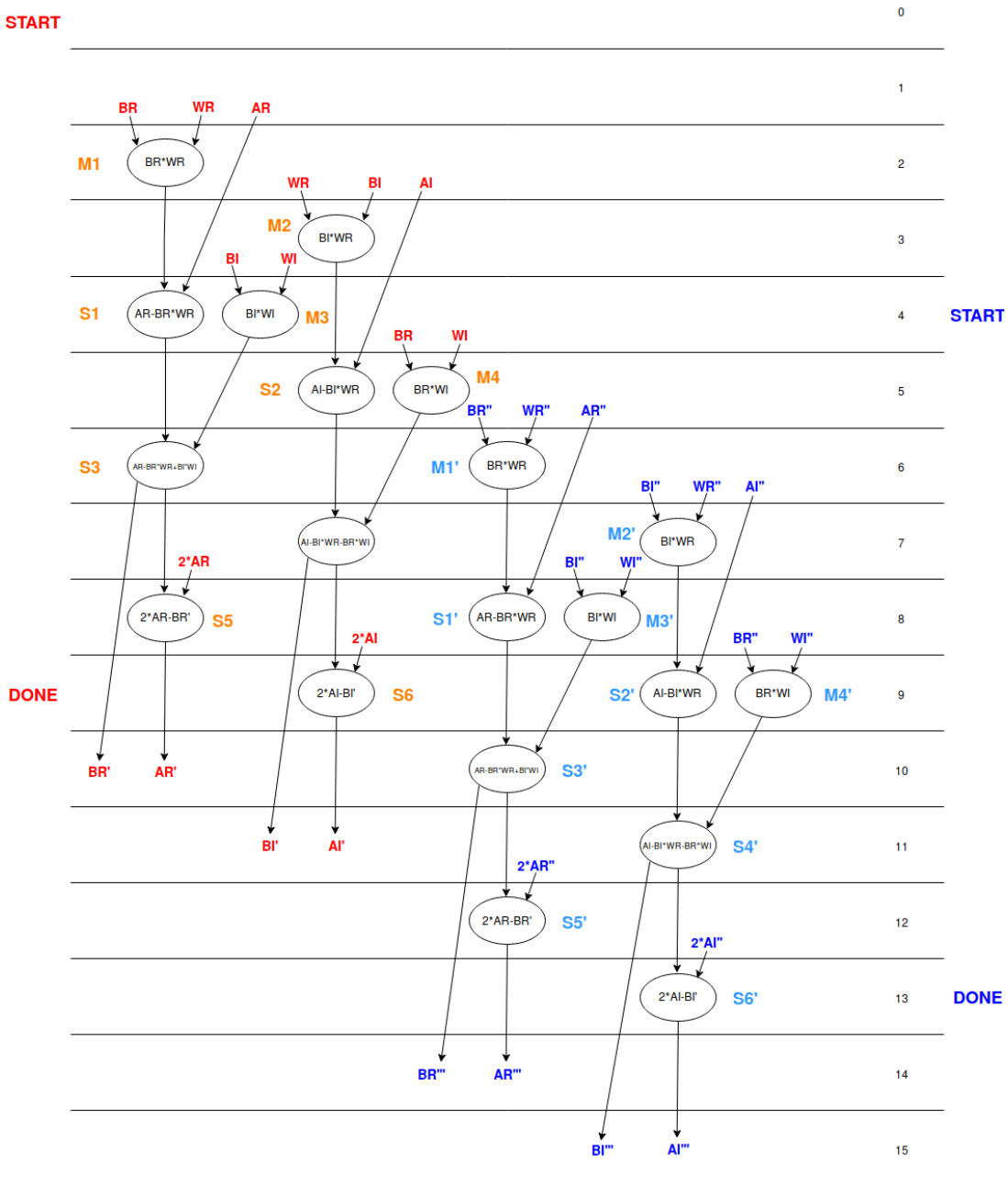
2.2.2 Control Flow Diagram

Dopo aver fatto uno studio approfondito tra tutte le possibili combinazioni tra ASAP e ALAP, è stato trovato il Control Flow Diagram migliore e più efficiente che permettesse di lavorare in:

- Modalità singola
- Modalità continua (Nuovi dati ogni 4 colpi di clock)
- Modalità continua sfasata (Nuovi dati ogni 5, 6, 7 colpi di clock)

Nel seguente Control Flow Diagram viene evidenziata l'evoluzione dei dati in modalità continua: dopo tre colpi di clock dal primo *START* vengono dati alla butterfly gli ingressi dell'elaborazione successiva:

Control Flow Diagram



2.2.3 Tempo di vita delle variabili

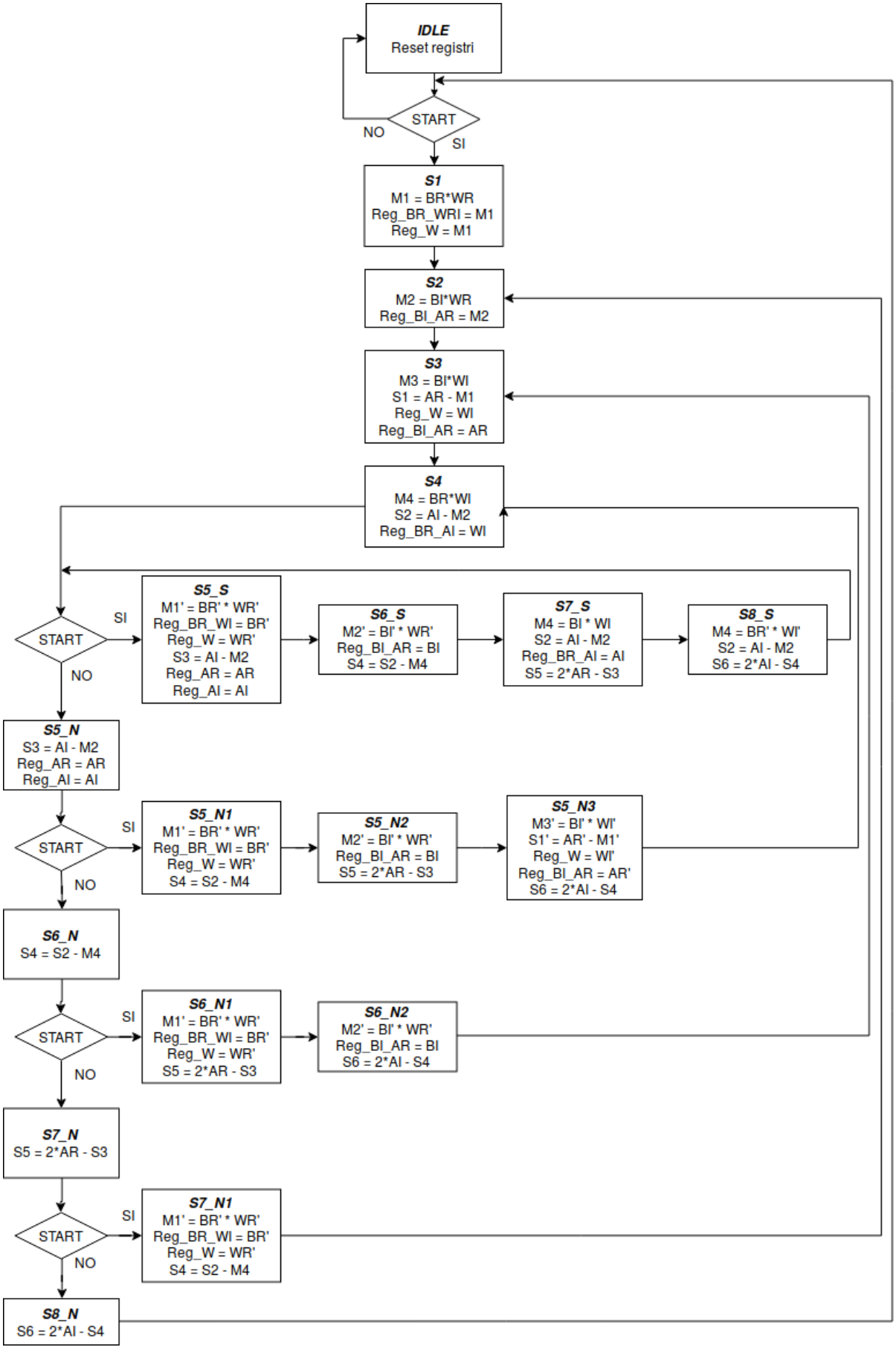
Per poter un'accurata analisi sulla possibile condivisione dell'hardware e dei registri, quindi, per effettuare delle ottimizzazioni nella progettazione del sistema, è stata elaborata la seguente tabelle in cui viene riportato il tempo di vita delle variabili. In particolare, nella tabella sono riportate le variabili per due esecuzioni in serie.

TEMPO DI VITA DELLE VARIABILI															
Clock n°	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Br	X	X	X	X											
Wr	X	X													
Bi		X	X												
Ar	X	X	X	X	X	X	X								
Ai		X	X	X	X	X	X	X							
Wi			X	X											
M1		X	X												
M2			X	X											
S1				X	X										
S2					X	X	X								
M3				X	X										
M4					X	X									
S3						X	X	X	X	X					
S4							X	X	X	X	X				
S5								X	X	X					
S6									X	X	X				
Br''					X	X	X	X							
Wr''					X	X									
Bi''						X	X								
Ar''					X	X	X	X	X	X	X				
Ai''						X	X	X	X	X	X	X			
Wi'							X	X							
M1'						X	X								
M2'							X	X							
S1'								X	X						
S2'									X	X					
M3'								X	X						
M4'									X	X					
S3'										X	X	X	X	X	
S4'											X	X	X	X	X
S5'												X	X	X	
S6'													X	X	X
Tot. Registri	0	2	5	4	4	6	9	7	7	8	5	3	3	4	2

Dall'analisi del tempo di vita delle variabili emerge che il minimo numero di dati da conservare memorizzati *contemporaneamente* in ogni colpo di clock è di 9, dunque servono almeno nove registri.

2.2.4 ASM Chart

L'ASM Chart mostra l'evoluzione degli stati tenendo conto anche della *Modalità continua sfasata (start 5, 6, 7 colpi di clock dal primo)*. In questo modo è possibile fornire al circuito dati con un periodo anche non multiplo di quattro periodi di clock, se per esempio i dati vengono prelevati con un periodo di 5 colpi di clock è possibile darli subito in ingresso alla Butterfly senza dover aspettare l'ottavo colpo di clock. Ciò permette in modalità non continua di risparmiare un numero di colpi di clock proporzionale al numero di esecuzioni. Se per esempio si hanno $20kB$ di dati ognuno acquisito ogni 5 colpi di clock, si risparmiano $60k$ colpi di clock rispetto al caso in cui non si consideri questa eventualità.


IDLE
Reset registri
START
NO
SI
S1
M1 = BR*WR
Reg_BR_WRI = M1
Reg_W = M1
S2
M2 = BI*WR
Reg_BI_AR = M2
S3
M3 = BI*WI
S1 = AR - M1
Reg_W = WI
Reg_BI_AR = AR
S4
M4 = BR*WI
S2 = AI - M2
Reg_BR_AI = WI
START
SI
NO
S5_S
M1' = BR' * WR'
Reg_BR_WI = BR'
Reg_W = WR'
S3 = AI - M2
Reg_AR = AR
Reg_AI = AI
S6_S
M2' = BI' * WR'
Reg_BI_AR = BI
S4 = S2 - M4
S7_S
M4 = BI * WI
S2 = AI - M2
Reg_BR_AI = AI
S5 = 2*AR - S3
S8_S
M4 = BR' * WI'
S2 = AI - M2
S6 = 2*AI - S4
S5_N
S3 = AI - M2
Reg_AR = AR
Reg_AI = AI
START
SI
NO
S5_N1
M1' = BR' * WR'
Reg_BR_WI = BR'
Reg_W = WR'
S4 = S2 - M4
S5_N2
M2' = BI' * WR'
Reg_BI_AR = BI
S5 = 2*AR - S3
S5_N3
M3' = BI' * WI'
S1' = AR' - M1'
Reg_W = WI'
Reg_BI_AR = AR'
S6 = 2*AI - S4
S6_N
S4 = S2 - M4
START
SI
NO
S6_N1
M1' = BR' * WR'
Reg_BR_WI = BR'
Reg_W = WR'
S5 = 2*AR - S3
S6_N2
M2' = BI' * WR'
Reg_BI_AR = BI
S6 = 2*AI - S4
S7_N
S5 = 2*AR - S3
START
SI
NO
S7_N1
M1' = BR' * WR'
Reg_BR_WI = BR'
Reg_W = WR'
S4 = S2 - M4
S8_N
S6 = 2*AI - S4

2.2.5 Datapath

Il datapath è stato realizzato utilizzando il minimo hardware possibile, con il minimo numero di registri come ricavato dall'analisi del tempo di vita delle variabili. Dato il ridotto numero di registri, non è stato necessario organizzarli in *register file*.

Ottimizzazione numero BUS

Per evitare costi/consumi maggiori (capacità parassite, area occupata) non sono stati inseriti BUS globali ma connessioni dirette locali. La struttura assomiglia ad una struttura *register based* in cui sono presenti degli accumulatori, e con i comandi opportuni si riesce a tenere il datapath sempre occupato senza "bolle" di dati che aumenterebbero la latenza.La microconcorrenza delle operazioni è massima con questa soluzione.

Moltiplicatore e sommatori

Come da specifiche, il moltiplicatore ed i due sommatori hanno uno stadio di pipeline per cui il risultato dell'elaborazione esce al colpo di clock successivo.

Ritardi combinatori e frequenza massima di lavoro

A discapito di una latenza maggiore per ottenere i campioni di uscita si è scelto di inserire dei registri di pipeline tra ogni blocco combinatorio che non sia un multiplexer. In particolare sono stati aggiunti registri di pipe anche tra l'uscita del moltiplicatore e l'ingresso del sommatore, poiché nonostante siano già pipelinati viene spezzato in due il ritardo tra l'elaborazione della seconda metà della moltiplicazione e della prima metà della somma. Tale percorso combinatorio rappresenta ragionevolmente il ritardo maggiore del circuito, dunque dopo aver inserito il registro ci si aspetta una frequenza di lavoro più alta. In particolare l'aumento di frequenza è direttamente proporzionale al ritardo della prima metà della somma $\frac{T_{SUM}}{2}$, che è di fatto stato sottratto dal T_{cycle} massimo.

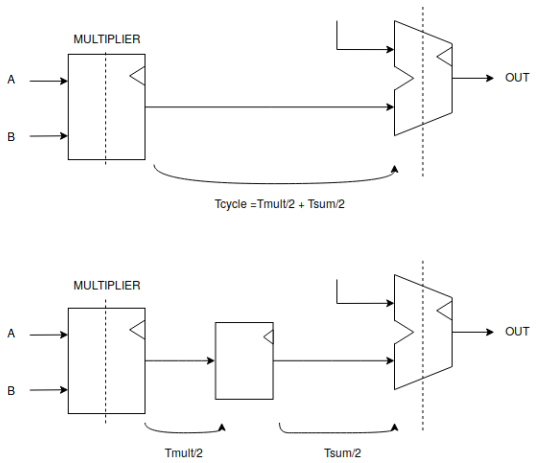
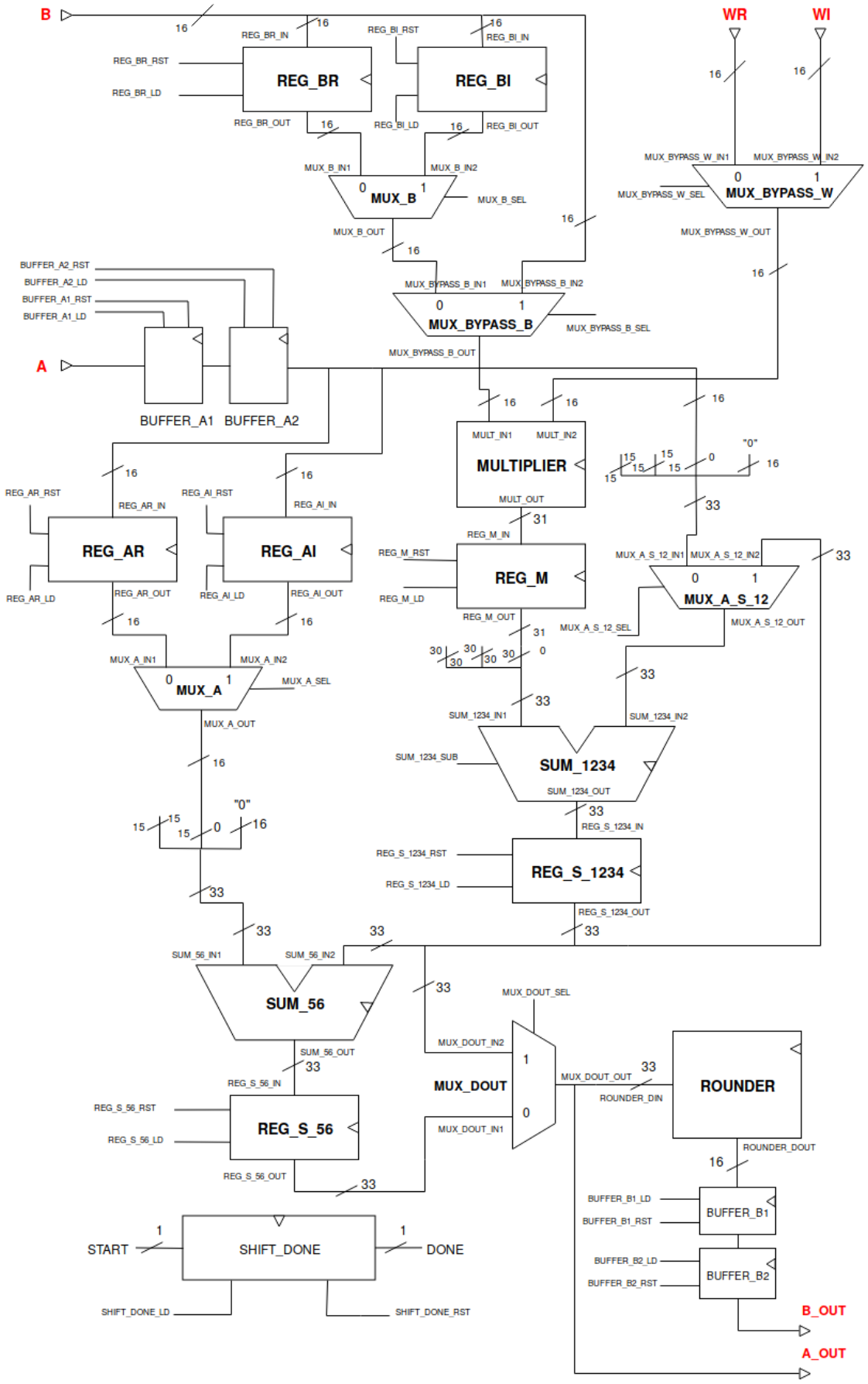


Figura 2.1: Ritardi combinatori con e senza pipeline

Schema

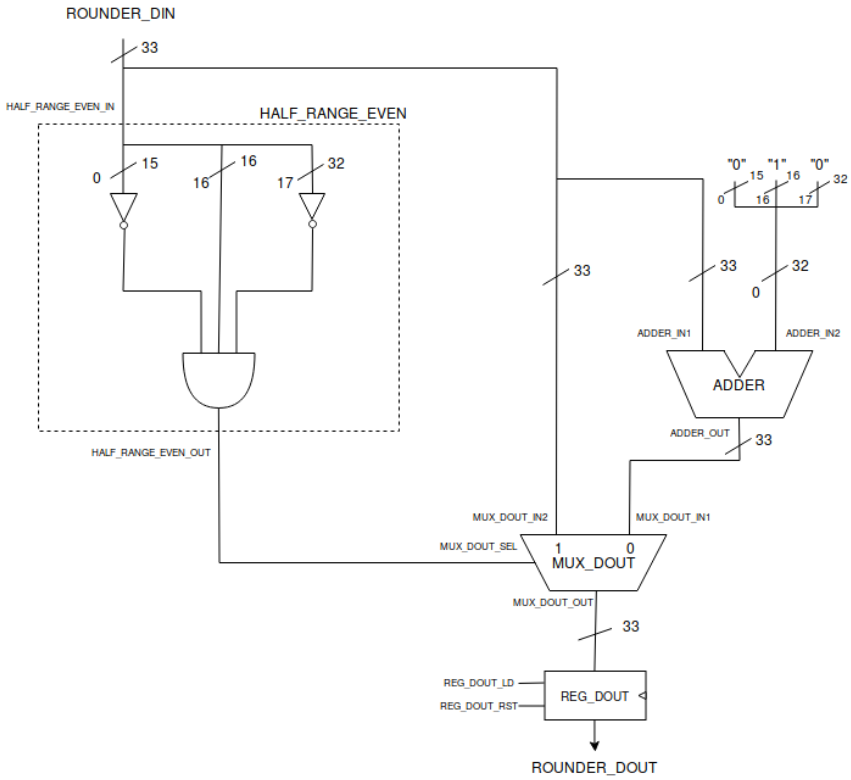

B
WR
WI
REG_BR
REG_BI
MUX_B
MUX_BYPASS_W
MUX_BYPASS_B
A
BUFFER_A1
BUFFER_A2
MULTIPLIER
REG_AR
REG_AI
REG_M
MUX_A_S_12
MUX_A
SUM_1234
REG_S_1234
SUM_56
REG_S_56
MUX_DOUT
ROUNDER
SHIFT_DONE
START
DONE
BUFFER_B1
BUFFER_B2
B_OUT
A_OUT

Rounder

I dati in uscita devono essere arrotondati tramite il *Rounding to nearest even* in modo che non siano presenti errori di arrotondamento polarizzati (bias error). L'arrotondamento deve essere effettuato **dopo dello scalamento**, ossia:

- Prima del rounder il formato numerico è Q2.30
- Si effettua lo scalamento *logico spostando il punto a sinistra*
- Si ottiene il formato frazionario Q0.32
- Si effettua il *rounding to nearest even* a partire dalla 15° cifra frazionaria

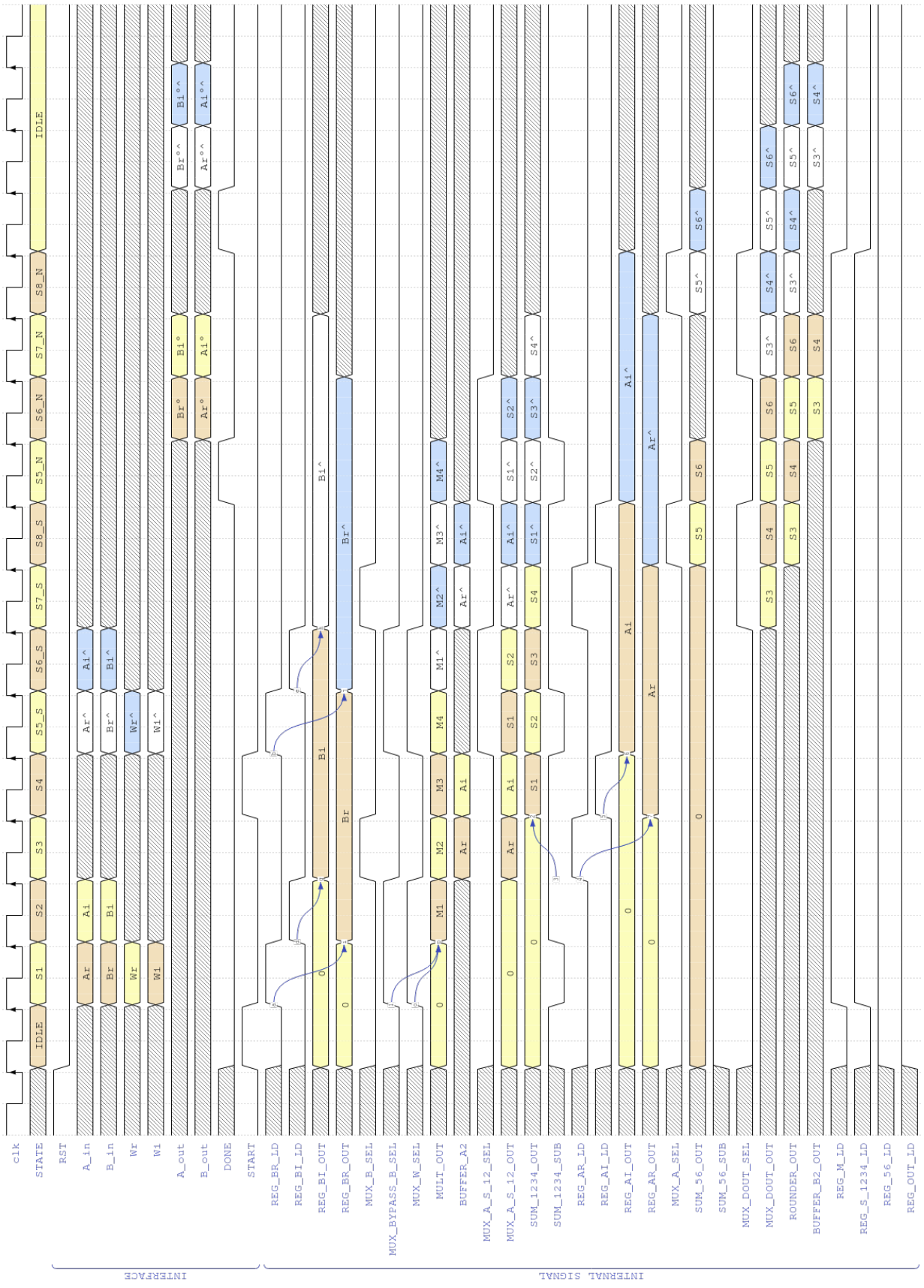
In questo modo, non c'è bisogno di effettuare alcun troncamento (che introdurrebbe errori polarizzati), e quindi si ottiene un errore di quantizzazione minore.



Il multiplexer è comandato dal segnale "half_range_even": se il numero da approssimare si trova a metà dinamica ed è pari, per arrotondare al numero pari più vicino è necessario un troncamento. In tutti gli altri casi viene sommato 0.5 nella posizione immediatamente successiva all'arrotondamento e solo successivamente il dato viene troncato.

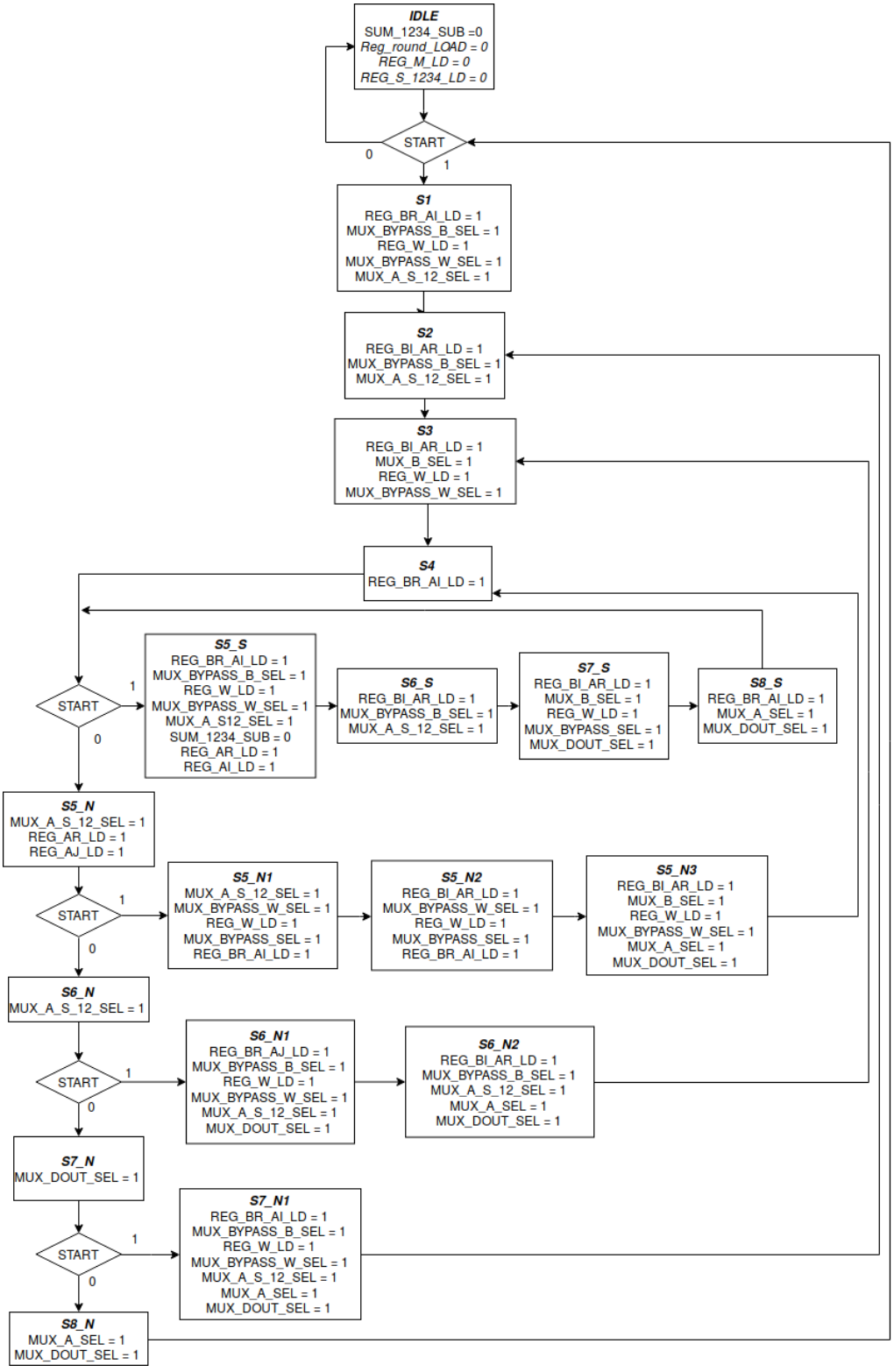
2.2.6 Timing

Dopo aver disegnato l'ASM chart è stato creato il timing della macchina per poter ricavare facilmente i bit di controllo, in particolare ci si è accorti che l'esecuzione di un singolo dato è controllata fino all'ottavo colpo di clock dopo lo start. Il timing è stato quindi particolarmente utile per ricavare i segnali di comando nel caso di ingresso traslato di 5, 6 e 7 colpi di clock, infatti in questi casi essendo la macchina piplinata è bastato fare l'OR fra i alcune sequenze di controllo, i comandi dello stato S5_N1 per esempio sono stati ricavati facendo l'OR dei comandi di S1 con i comandi di S5_N e ciò è proprio possibile grazie alla non sovrapposizione delle variabili e quindi dei controlli. **Segue il timing della simulazione nella sezione "Testbench e simulazione"**


clk
STATE
RST
A_in
B_in
Wr
Wi
A_out
B_out
DONE
START
REG_BR_LD
REG_BI_LD
REG_BI_OUT
REG_BR_OUT
MUX_B_SEL
MUX_BYPASS_B_SEL
MUX_W_SEL
MULT_OUT
BUFFER_A2
MUX_A_S_12_SEL
MUX_A_S_12_OUT
SUM_1234_OUT
SUM_1234_SUB
REG_AR_LD
REG_AI_LD
REG_AI_OUT
REG_AR_OUT
MUX_A_SEL
SUM_56_OUT
SUM_56_SUB
MUX_DOUT_SEL
MUX_DOUT_OUT
ROUNDER_OUT
BUFFER_B2_OUT
REG_M_LD
REG_S_1234_LD
REG_56_LD
REG_OUT_LD
INTERFACE
INTERNAL SIGNAL

2.2.7 Control ASM

Negli stati della ASM chart vengono indicati i comandi dati al datapath:



2.2.8 Unità di controllo

Per l'unità di controllo è stata utilizzata una macchina microprogrammata di tipo Late Status dove viene utilizzato un multiplexer di bypass per andare a effettuare direttamente una scelta fra l'istruzione pari e dispari in uscita dalla µROM, in particolare il Conditional Sequential PLA (che comanda il bypass e il registro UPC_SEL_REG) è stato sostituito con un semplice multiplexer dato che l'unico segnale di controllo della macchina è lo start (non c'è gestione di overflow da specifiche). In particolare se nella stringa di controllo si asserisce il CC il MUX_MEM verrà comandato direttamente dallo Start bypassando il registro UPC_SEL_REG, altrimenti esso viene comandato dall'LSB del Next State che decide quindi dove saltare. Per quanto riguarda i clock nella control unit si è deciso di utilizzare lo stesso clock per tutti i registri ma UPC_REG e UPC_SEL_REG campionano sul fronte di discesa del clock a differenza degli altri che campionano sul fronte di discesa, in questo modo ogni decisione della control unit impiega un colpo di clock. Per prima cosa è quindi stata creata la tabella da inserire nella µROM dove sono stati inseriti dei nomi agli indirizzi per semplificare il debug e la leggibilità della tabella.

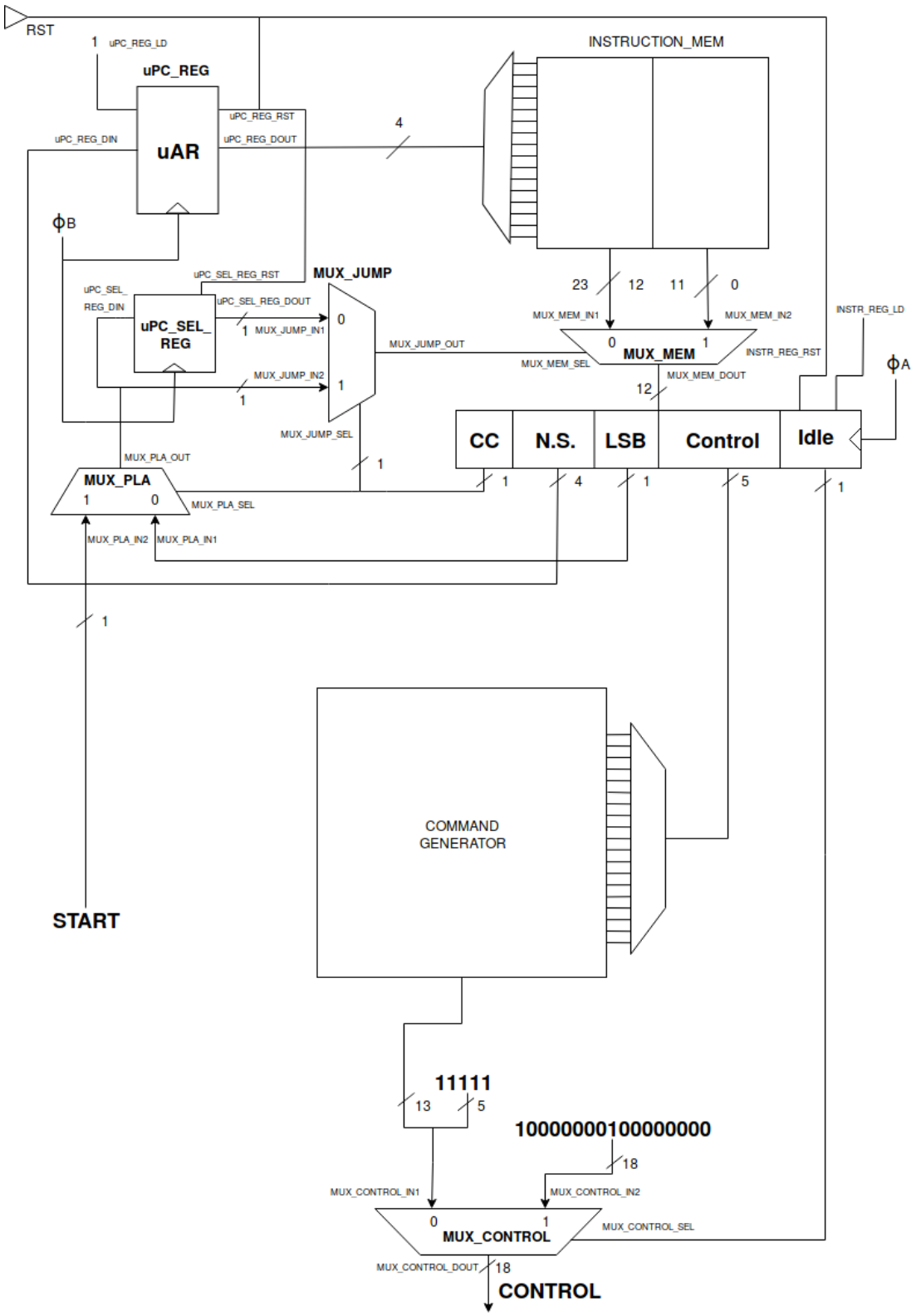


Figura 2.2: Unità di controllo

Tipologia di microprogrammazione

Per rendere il progetto più modulare e facilitare le modifiche in fase di debug è stato deciso di orientarsi verso una microprogrammazione più verticale e quindi codificare i controlli in 5 bit utilizzati poi per indirizzare una PLA che li decodifichi. Come si nota nello schema dell'unità di controllo i bit di default sono stati aggiunti al segnale di controllo in uscita dato che non vengono mai modificati.

Address	Odd State	CC	Next State	LSB	Control	Even State	CC	Next State	LSB	Control
0000 → A	IDLE	1	A 0000	1	00000 | 1	S1	0	J 1001	0	00000 | 0
0001 → B	S4	1	I 1000	-	00011 | 0	S6_S	0	C 0010	0	01001 | 0
0010 → C	S7_S	0	C 0010	1	01010 | 0	S8_S	1	B 1000	0	01011 | 0
0011 → D	S5_N2	0	D 0011	1	01101 | 0	S5_N3	0	B 0001	0	01110 | 0
0100 → E	S6_N2	0	J 1001	1	10000 | 0	IDLE	1	A 0000	1	00000 | 1
0101 → F	S8_N	1	A 0000	-	00111 | 0	S7_N1	0	J 1001	0	10001 | 0
0110 → G	S7_N	1	F 0101	-	00110 | 0	S6_N1	0	E 0100	0	01111 | 0
0111 → H	S6_N	1	G 0110	-	00101 | 0	S5_N1	0	D 0011	0	01100 | 0
1000 → I	S5_N	1	H 0111	-	00100 | 0	S5_S	0	B 0001	1	01000 | 0
1001 → J	S2	1	J 1001	1	00001 | 0	S3	0	B 0001	0	00010 | 0
1010 → K	IDLE	1	A 0000	1	00000 | 1	IDLE	1	A 0000	1	00000 | 1
1011 → L	IDLE	1	A 0000	1	00000 | 1	IDLE	1	A 0000	1	00000 | 1
1100 → M	IDLE	1	A 0000	1	00000 | 1	IDLE	1	A 0000	1	00000 | 1
1101 → N	IDLE	1	A 0000	1	00000 | 1	IDLE	1	A 0000	1	00000 | 1
1110 → O	IDLE	1	A 0000	1	00000 | 1	IDLE	1	A 0000	1	00000 | 1
1111 → P	IDLE	1	A 0000	1	00000 | 1	IDLE	1	A 0000	1	00000 | 1

Figura 2.3: µROM

Decodifica dei comandi

Utilizzando il timing è quindi stata creata la tabella di decodifica dei comandi, di seguito anche la legenda dei bit di comando.

COMMAND GENERATOR												
ADDRESS	STATE	COMMAND										
		16	15	14	13	12	11	10	9	8	7	6
00000	S1	1	0	0	1	1	0	1	0	0	0	0
00001	S2	0	1	0	1	1	0	1	0	0	0	0
00010	S3	0	0	1	0	0	0	1	1	0	0	0
00011	S4	0	0	0	0	0	0	1	0	1	0	0
00100	S5_N	0	0	0	0	0	1	0	0	0	0	0
00101	S6_N	0	0	0	0	0	1	1	0	0	0	0
00110	S7_N	0	0	0	0	0	0	1	0	0	0	1
00111	S8_N	0	0	0	0	0	0	1	0	0	1	1
01000	S5_S	1	0	0	1	1	1	0	0	0	0	0
01001	S6_S	0	1	0	1	1	1	1	0	0	0	0
01010	S7_S	0	0	1	0	0	0	1	1	0	0	1
01011	S8_S	0	0	0	0	0	0	1	0	1	1	1
01100	S5_N1	1	0	0	1	1	1	1	0	0	0	0
01101	S5_N2	0	1	0	1	1	0	1	0	0	0	1
01110	S5_N3	0	0	1	0	0	0	1	1	0	1	1
01111	S6_N1	1	0	0	1	1	0	1	0	1	0	1
10000	S6_N2	0	1	0	1	1	0	1	0	0	1	1
10001	S7_N1	1	0	0	1	1	0	1	1	0	1	1

Figura 2.4: Command generator

COMMAND BIT NUMBER	COMMAND NAME
16	REG_BR_AI_LD
15	REG_BI_AR_LD
14	MUX_B_SEL
13	MUX_BYPASS_SEL
12	MUX_BYPASS_W_SEL
11	MUX_A_S_12_SEL
10	SUM_1234_SUB
9	REG_AR_LD
8	REG_AI_LD
7	MUX_A_SEL
6	MUX_DOUT_SEL

Figura 2.5: Command generator - Legenda

2.3 Testbench e simulazione

2.3.1 Schema della simulazione

Per verificare la correttezza delle operazioni svolte dal circuito, si è voluto evitare di realizzare in VHDL tale controllo in quanto facendo una descrizione ad alto livello con lo stesso linguaggio gli output generati dal testbench in VHDL potrebbero più facilmente essere affetti dagli stessi errori del circuito.
Si è dunque proceduto nel seguente modo:

a. Creazione di un file contenente vettori di test generati in modo casuale;

b. Esecuzione calcoli dal circuito butterfly;

c. Scrittura su di un file degli output generati dal circuito;

d. Comparazione dei file di output con i file generati da procedura software MATLAB, bash.

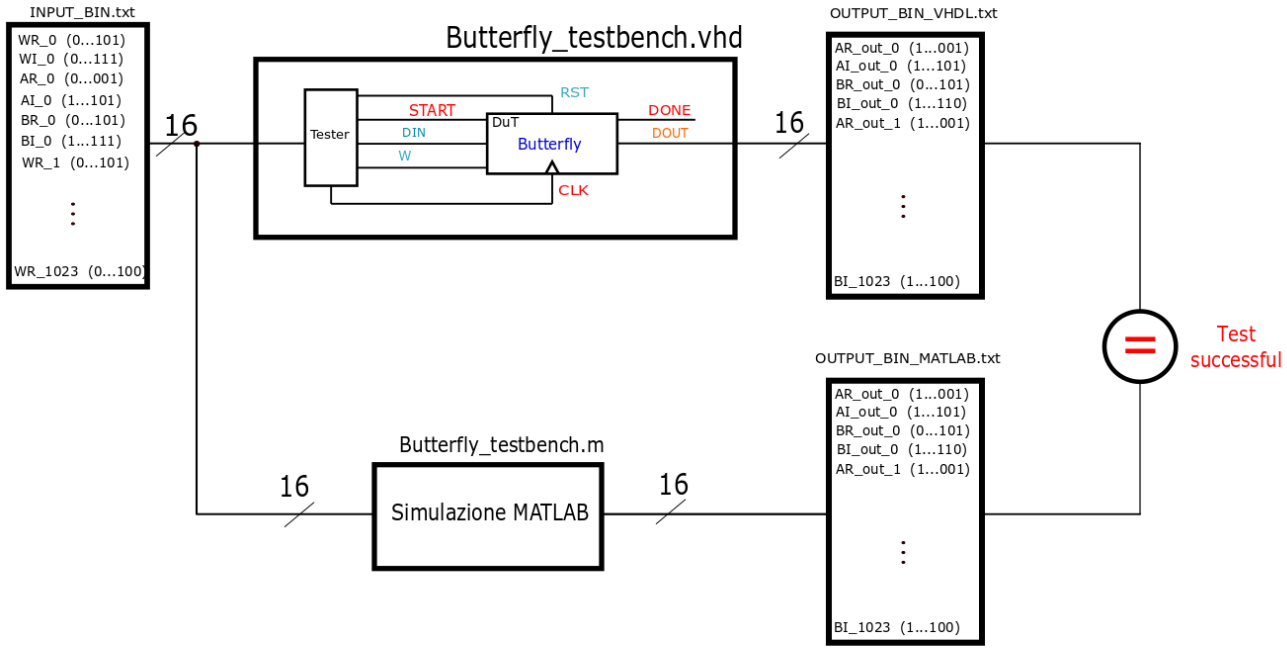


Figura 2.6: Schema della simulazione

Dopo diverse prove di congruenza tra dati generati dal circuito dal programma ed i dati generati in Matlab si è ritenuto il circuito funzionante. **Seguono in appendice i codici di testbench e delle simulazioni software.**

2.3.2 Vettori di test e simulazione Modelsim

I file di test sono così strutturati:

Ingressi

Input.txt	Dato in ingresso	Codifica decimale	Codifica fixed point
0111011101101010	W_R_0	30570	0.93292236328125
1001111011110011	W_I_0	-24845	-0.758209228515625
0111110100110101	A_R_0	32053	0.978179931640625
1011110010110111	A_I_0	-17225	-0.525665283203125
1100000111100001	B_R_0	-15903	-0.485321044921875
0100001110101011	B_I_0	17323	0.528656005859375
1111110001100010	W_R_1	-926	-0.02825927734375
...	...	...	...
1110101101011001	B_I_{1023}	-5287	-0.161346435546875

Uscite

Considerare che i seguenti dati in uscita devono essere moltiplicati per il fattore di scala (4).

Output.txt	Dato in uscita	Codifica decimale	Codifica fixed point
0001110110100100	$A\_R'_0$	7588	0.2315673828125
0000101010111100	$A\_I'_0$	2748	0.0838623046875
0010000011110111	$B\_R'_0$	8439	0.257537841796875
1101001110011111	$B\_I'_0$	-11361	-0.346710205078125
1110000010001011	$A\_R'_1$	-8053	-0.245758056640625
...	...	...	...
0001100000111011	$B\_I'_{1023}$	5559	0.169647216796875

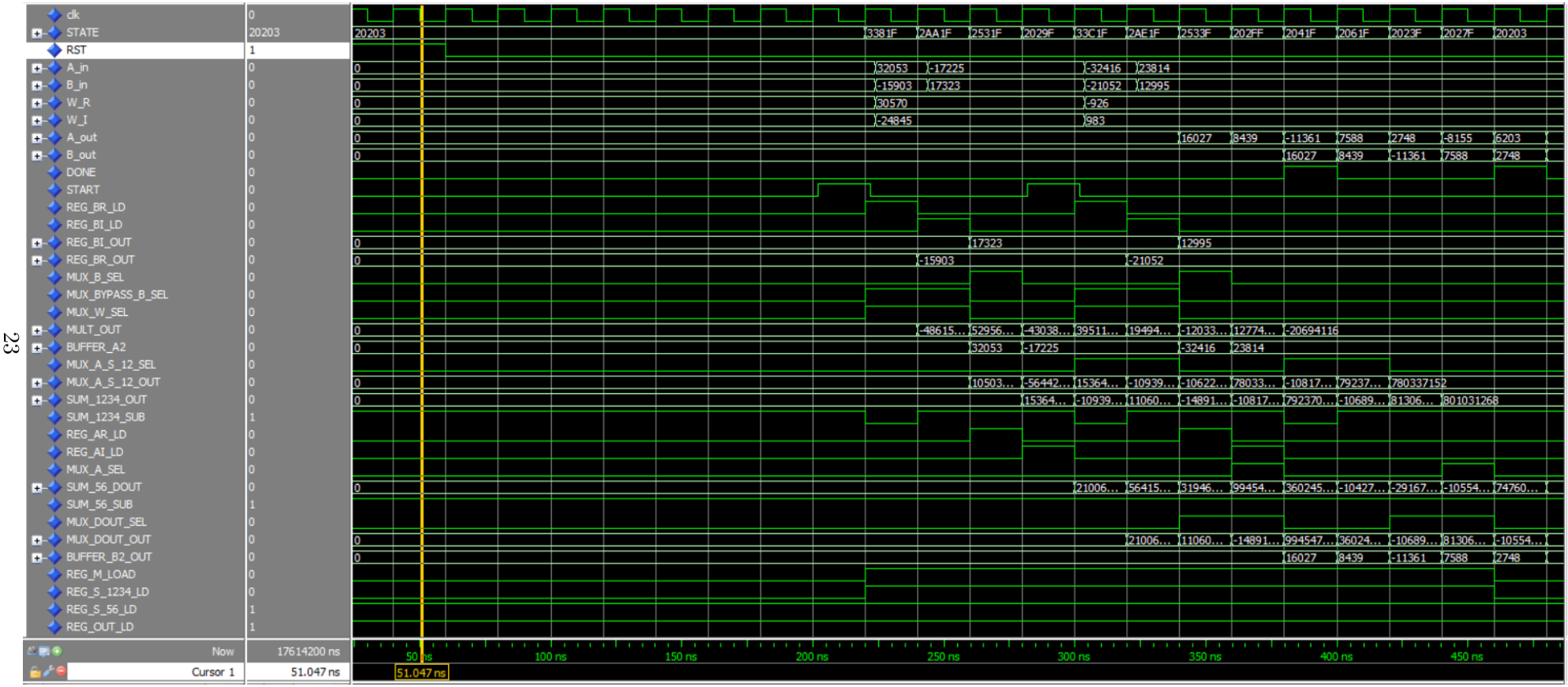
I 6144 numeri casuali generati e usati come vettori di test sono ritenuti rappresentativi per il funzionamento della macchina per i seguenti motivi:

- Analizzando i casi limite riportati nella tabella della pagina successiva, si è posta l'attenzione sui casi più critici ossia quando ci sono tre somme consecutive per trovare un dato, questi casi sono un terzo di tutti i possibili 64 casi, per questo motivo si è deciso di utilizzare solo numeri casuali per il test poiché si ha un'alta probabilità di ricadere in molti casi critici.
- Il test è stato effettuato più e più volte con nuovi file casuali

Per quanto riguarda il debug su modelsim, sono stati osservati i valori dei dati trasformati in decimale, come mostrato nella figura successiva.

INGRESSI						MOLTIPLICAZIONI				SENZA ROUNDING			
Br	Bi	Ar	Ai	Wr	Wi	BrWr	BiWr	BrWi	BiWi	Br'	Bi'	Ar'	Ai'
-	-	-	-	-	-	+	+	+	+	1	0	1	2
-	-	-	-	-	+	+	+	-	-	0	1	2	1
-	-	-	-	+	-	-	-	+	+	2	1	0	1
-	-	-	-	+	+	-	-	-	-	1	2	1	0
-	-	-	+	-	-	+	+	+	+	1	1	1	3
-	-	-	+	-	+	+	+	-	-	0	2	2	2
-	-	-	+	+	-	-	-	+	+	2	2	0	2
-	-	-	+	+	+	-	-	-	-	1	3	1	1
-	-	+	-	-	-	+	+	+	+	2	0	2	2
-	-	+	-	-	+	+	+	-	-	1	1	3	1
-	-	+	-	+	-	-	-	+	+	3	1	1	1
-	-	+	-	+	+	-	-	-	-	2	2	2	0
-	-	+	+	-	-	+	+	+	+	2	1	2	3
-	-	+	+	-	+	+	+	-	-	1	2	3	2
-	-	+	+	+	-	-	-	+	+	3	2	1	2
-	-	+	+	+	+	-	-	-	-	2	3	2	1
-	+	-	-	-	-	+	-	+	-	0	1	2	1
-	+	-	-	-	+	+	-	-	+	1	2	1	0
-	+	-	-	+	-	-	+	+	-	1	0	1	2
-	+	-	-	+	+	-	+	-	+	2	1	0	1
-	+	-	+	-	-	+	-	+	-	0	2	2	2
-	+	-	+	-	+	+	-	-	+	1	3	1	1
-	+	-	+	+	-	-	+	+	-	1	1	1	3
-	+	-	+	+	+	-	+	-	+	2	2	0	2
-	+	+	-	-	-	+	-	+	-	1	1	3	1
-	+	+	-	-	+	+	-	-	+	2	2	2	0
-	+	+	-	+	-	-	+	+	-	2	0	2	2
-	+	+	-	+	+	-	+	-	+	3	1	1	1
-	+	+	+	-	-	+	-	+	-	1	2	3	2
-	+	+	+	-	+	+	-	-	+	2	3	2	1
-	+	+	+	+	-	-	+	+	-	2	1	2	3
-	+	+	+	+	+	-	+	-	+	3	2	1	2

INGRESSI						MOLTIPLICAZIONI				SENZA ROUNDING			
Br	Bi	Ar	Ai	Wr	Wi	BrWr	BiWr	BrWi	BiWi	Br'	Bi'	Ar'	Ai'
+	-	-	-	-	-	-	+	-	+	2	1	0	1
+	-	-	-	-	+	-	+	+	-	1	0	1	2
+	-	-	-	+	-	+	-	-	+	1	2	1	0
+	-	-	-	+	+	+	-	+	-	0	1	2	1
+	-	-	+	-	-	-	+	-	+	2	2	0	2
+	-	-	+	-	+	-	+	+	-	1	1	1	3
+	-	-	+	+	-	+	-	-	+	1	3	1	1
+	-	-	+	+	+	+	-	+	-	0	2	2	2
+	-	+	-	-	-	-	+	-	+	3	1	1	1
+	-	+	-	-	+	-	+	+	-	2	0	2	2
+	-	+	-	+	-	+	-	-	+	2	2	2	0
+	-	+	-	+	+	+	-	+	-	1	1	3	1
+	-	+	+	-	-	-	+	-	+	3	2	1	2
+	-	+	+	-	+	-	+	+	-	2	1	2	3
+	-	+	+	+	-	+	-	-	+	2	3	2	1
+	-	+	+	+	+	+	-	+	-	1	2	3	2
+	+	-	-	-	-	-	-	-	-	1	2	1	0
+	+	-	-	-	+	-	-	+	+	2	1	0	1
+	+	-	-	+	-	+	+	-	-	0	1	2	1
+	+	-	-	+	+	+	+	+	+	1	0	1	2
+	+	-	+	-	-	-	-	-	-	1	3	1	1
+	+	-	+	-	+	-	-	+	+	2	2	0	2
+	+	-	+	+	-	+	+	-	-	0	2	2	2
+	+	-	+	+	+	+	+	+	+	1	1	1	3
+	+	+	-	-	-	-	-	-	-	2	2	2	0
+	+	+	-	-	+	-	-	+	+	3	1	1	1
+	+	+	-	+	-	+	+	-	-	1	1	3	1
+	+	+	-	+	+	+	+	+	+	2	0	2	2
+	+	+	+	-	-	-	-	-	-	2	3	2	1
+	+	+	+	-	+	-	-	+	+	3	2	1	2
+	+	+	+	+	-	+	+	-	-	1	2	3	2
+	+	+	+	+	+	+	+	+	+	2	1	2	3


clk 0
STATE 20203
RST 1
A_in 0
B_in 0
W_R 0
W_I 0
A_out 0
B_out 0
DONE 0
START 0
REG_BR_LD 0
REG_BI_LD 0
REG_BI_OUT 0
REG_BR_OUT 0
MUX_B_SEL 0
MUX_BYPASS_B_SEL 0
MUX_W_SEL 0
MULT_OUT 0
BUFFER_A2 0
MUX_A_S_12_SEL 0
MUX_A_S_12_OUT 0
SUM_1234_OUT 0
SUM_1234_SUB 1
REG_AR_LD 0
REG_AI_LD 0
MUX_A_SEL 0
SUM_56_DOUT 0
SUM_56_SUB 1
MUX_DOUT_SEL 0
MUX_DOUT_OUT 0
BUFFER_B2_OUT 0
REG_M_LOAD 0
REG_S_1234_LD 0
REG_S_56_LD 1
REG_OUT_LD 1
Now 17614200 ns
Cursor 1 51.047 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
450 ns
51.047 ns

Capitolo 3

Applicazione Butterfly - FFT completa

Con l'utilizzo del blocchetto "Butterfly" è stato possibile effettuare l'FFT su **otto** campioni complessi attraverso la combinazione di dodici diversi blocchetti. Lo schema finale dell'FFT permette di avere in uscita i campioni in frequenza ordinati a patto che i dati nel tempo vengano dati in ingresso seguendo l'ordine chiamato 'Reverse Bit', per questo motivo è stato utilizzato il seguente schema:

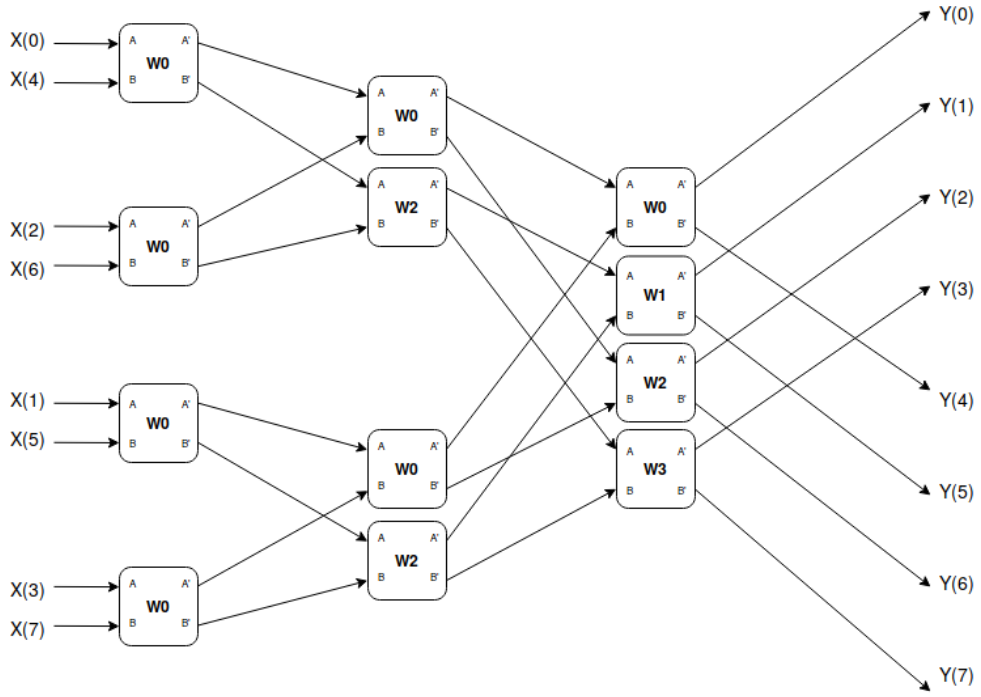
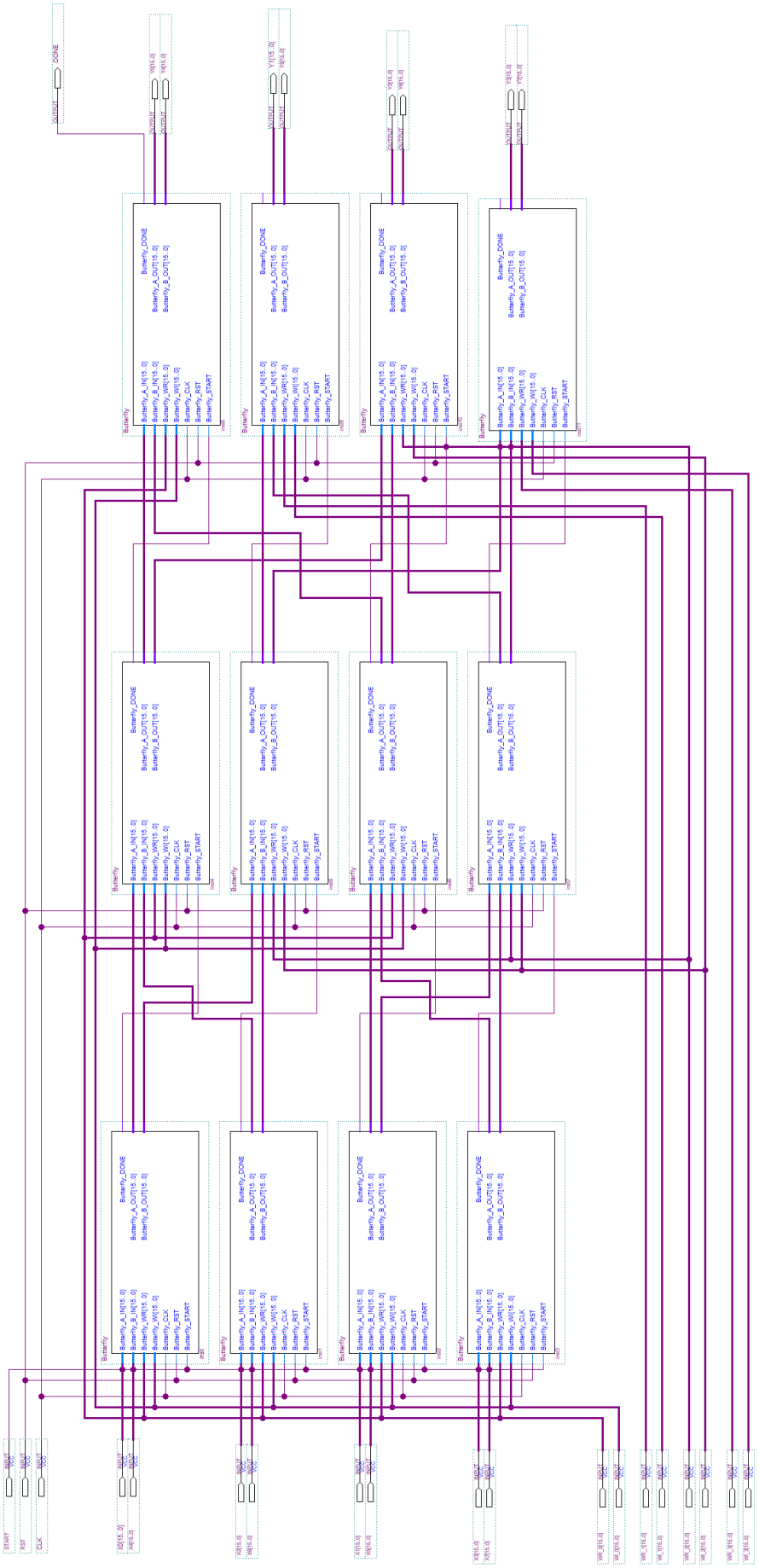


Figura 3.1: Schema FFT

Usando l'editor schematic capture in Quartus è stato quindi creato il datapath completo dell'FFT usando la "Butterfly" come blocco elementare. I coefficienti W sono, relativamente:

$$W_0 = 1 + j\,0 \qquad W_2 = 0 + j\,1$$

$$W_1 = 0.707106 + j\,0.707106 \qquad W_3 = -0.707106 + j\,0.707106$$

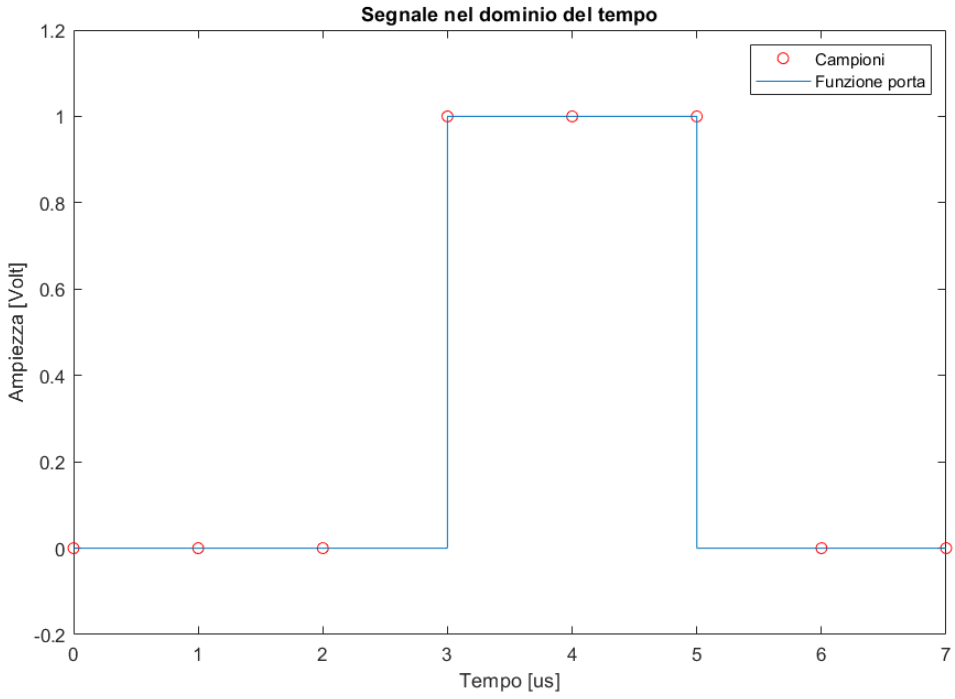


3.1 Spettro di un segnale in ingresso

Per testarne il funzionamento è stato creato un vettore di ingressi che rappresenta un segnale **porta** campionato nel dominio del tempo (8 campioni) I campioni derivanti da un segnale reale non presentano parte immaginaria, i valori dati come ingresso alla FFT sono:

Input.txt	Dato in ingresso	Codifica decimale	Codifica fixed point
0000000000000000	x_0_R	0	0
0000000000000000	x_0_I	0	0
0000000000000000	x_1_R	0	0
0000000000000000	x_1_I	0	0
0000000000000000	x_2_R	0	0
0000000000000000	x_2_I	0	0
0111111111111111	x_3_R	32767	0.999969482421875
0000000000000000	x_3_I	0	0
0111111111111111	x_4_R	32767	0.999969482421875
0000000000000000	x_4_I	0	0
0111111111111111	x_5_R	32767	0.999969482421875
0000000000000000	x_5_I	0	0
0000000000000000	x_6_R	0	0
0000000000000000	x_6_I	0	0
0000000000000000	x_7_R	0	0
0000000000000000	x_7_I	0	0

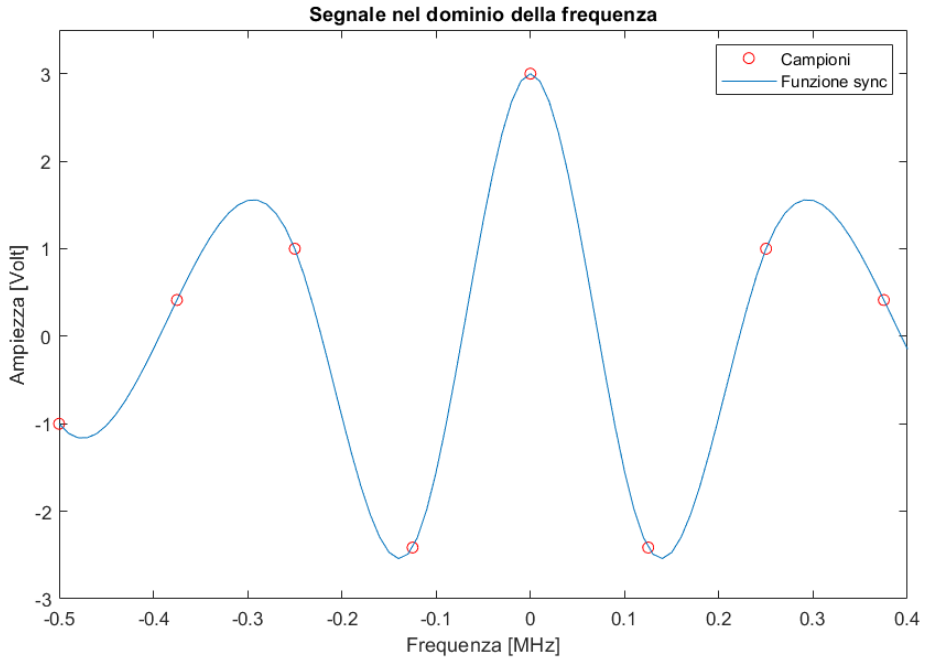
I campioni sono stati ottenuti dal seguente segnale:



I seguenti risultati in uscita sono **già stati moltiplicati per il valore 64**, avendo tre schiere di butterfly ognuna con un fattore di scala di 4.

Output.txt	**Dato in uscita**	**Valore fixed point * 64**
0000011000000000	y_0_R	3
0000000000000000	y_0_I	0
1111101100101100	y_1_R	-2.414214
0000000000000000	y_1_I	0
0000001000000000	y_2_R	1
0000000000000000	y_2_I	0
0000000011010100	y_3_R	0.414214
0000000000000000	y_3_I	0
1111111000000000	y_4_R	-1
0000000000000000	y_4_I	0
0000000011010100	y_5_R	0.414214
0000000000000000	y_5_I	0
0000001000000000	y_6_R	1
0000000000000000	y_6_I	0
1111101100101100	y_7_R	-2.414214
0000000000000000	y_7_I	0

I campioni che si ottengono da qualsiasi DFT mostrano prima le frequenze positive e poi le negative, dunque per ricostruire lo spettro del segnale occorre traslare le uscite ottenute da metà in poi alle frequenze negative. Interpolando i campioni di uscita e traslando le frequenze negative si ottiene il seguente segnale, che corrisponde ad una sinc, come è da aspettarsi da una trasformata di Fourier di una funzione porta.



Capitolo 4

Appendice

4.1 Testbench

4.1.1 Testbench Butterfly

Matlab - testbench.m

```
clc;
clear;
close all;

%Creazione matrice binaria casuale (1024 casi diversi, 6 numeri ognuno)
 for i = 1:6144
   INPUT_BIN(i, 1:16) = randi([0 1], 1, 16);
 end

% Generazione matrici output

k = 1; %Indice per scrittura su matrice output

%Preallocazione memoria per velocit
INPUT_DEC = zeros(6144,1,'int16');
OUTPUT_BIN = repmat(' ',[4096 16]);
OUTPUT_DEC = zeros(4096,1,'int16');

for i = 1:6:6144;


%Prendo i numeri binari dalla matrice generata casualmente
W_R_BIN = INPUT_BIN(i, 1:16);
W_I_BIN = INPUT_BIN(i+1, 1:16);
A_R_BIN = INPUT_BIN(i+2, 1:16);
A_I_BIN = INPUT_BIN(i+3, 1:16);
B_R_BIN = INPUT_BIN(i+4, 1:16);
B_I_BIN = INPUT_BIN(i+5, 1:16);

%Trasformo i numeri da binari a decimali signed
W_R_dec = typecast(uint16(bin2dec(num2str(W_R_BIN))), 'int16');
```

```
32 W_I_dec = typecast(uint16(bin2dec(num2str(W_I_BIN))), 'int16');
33 A_R_dec = typecast(uint16(bin2dec(num2str(A_R_BIN))), 'int16');
34 A_I_dec = typecast(uint16(bin2dec(num2str(A_I_BIN))), 'int16');
35 B_R_dec = typecast(uint16(bin2dec(num2str(B_R_BIN))), 'int16');
36 B_I_dec = typecast(uint16(bin2dec(num2str(B_I_BIN))), 'int16');
37
38 %Creo una matrice che contiene i numeri convertiti per avere il confronto
39 INPUT_DEC(i)   = W_R_dec;
40 INPUT_DEC(i+1) = W_I_dec;
41 INPUT_DEC(i+2) = A_R_dec;
42 INPUT_DEC(i+3) = A_I_dec;
43 INPUT_DEC(i+4) = B_R_dec;
44 INPUT_DEC(i+5) = B_I_dec;
45
46 %Ottengo i numeri frazionari dividendo per 2^15, formato Q0.15
47 W_R = fi(double(W_R_dec)/(2^15), 1, 16, 15);
48 W_I = fi(double(W_I_dec)/(2^15), 1, 16, 15);
49 A_R = fi(double(A_R_dec)/(2^15), 1, 16, 15);
50 A_I = fi(double(A_I_dec)/(2^15), 1, 16, 15);
51 B_R = fi(double(B_R_dec)/(2^15), 1, 16, 15);
52 B_I = fi(double(B_I_dec)/(2^15), 1, 16, 15);
53
54 %Svolgimento operazioni
55
56 %Aggiusta in automatico il parallelismo, per  si riserva qualche bit in pi..
57 A_R_out = A_R + B_R*W_R - B_I*W_I; %DIVENTA Q3.30
58 A_I_out = A_I + B_R*W_I + B_I*W_R; %DIVENTA Q3.30
59 B_R_out = - A_R_out + A_R*2;%DIVENTA Q4.30
60 B_I_out = A_I*2 - A_I_out; %DIVENTA Q4.30
61
62 %Alla fine ne ha 34/35 invece che 33, ma successivamente vengono tolti.
63
64 %Non considerando le estensioni di segno aggiuntive a sinistra:
65
66 %All'uscita il formato deve essere Q0.15 invece che Q2.30
67 %Per usare la funzione rounding per, Q0.15 deve essere Q15.0
68 %Poich alla fine il fattore di scalamento deve essere di 4, moltiplico per
69 %2^(-2) e poi per 2^15 (per arrotondare).
70 A_R_out_BIN_temp = bin(convergent(fi(A_R_out*2^13, 1, 34, 18)));
71 A_I_out_BIN_temp = bin(convergent(fi(A_I_out*2^13, 1, 34, 18)));
72 B_R_out_BIN_temp = bin(convergent(fi(B_R_out*2^13, 1, 35, 18)));
73 B_I_out_BIN_temp = bin(convergent(fi(B_I_out*2^13, 1, 35, 18)));
74
75 %Ora ho Q16.0 e Q17.0, ma il numero binario  lo stesso di Q 1.15 e Q 2.15
76 %Tolgo i bit interi in eccesso, tanto l'overflow non c'  stato
77 A_R_out_BIN = A_R_out_BIN_temp(2:17);
78 A_I_out_BIN = A_I_out_BIN_temp(2:17);
79 B_R_out_BIN = B_R_out_BIN_temp(3:18);
80 B_I_out_BIN = B_I_out_BIN_temp(3:18);
81
82 OUTPUT_BIN(k, 1:16) = A_R_out_BIN;
83 OUTPUT_BIN(k+1, 1:16) = A_I_out_BIN;
84 OUTPUT_BIN(k+2, 1:16) = B_R_out_BIN;
```

```
85  OUTPUT_BIN(k+3, 1:16) = B_I_out_BIN;
86
87  A_R_out_DEC = typecast(uint16(bin2dec(num2str(A_R_out_BIN))), 'int16');
88  A_I_out_DEC = typecast(uint16(bin2dec(num2str(A_I_out_BIN))), 'int16');
89  B_R_out_DEC = typecast(uint16(bin2dec(num2str(B_R_out_BIN))), 'int16');
90  B_I_out_DEC = typecast(uint16(bin2dec(num2str(B_I_out_BIN))), 'int16');
91
92  OUTPUT_DEC(k) = A_R_out_DEC;
93  OUTPUT_DEC(k+1) = A_I_out_DEC;
94  OUTPUT_DEC(k+2) = B_R_out_DEC;
95  OUTPUT_DEC(k+3) = B_I_out_DEC;
96
97  k = k + 4;
98  end
99
100 %Creazione FILE
101 dlmwrite("INPUT_BIN.txt", INPUT_BIN, 'delimiter', '');
102 dlmwrite("OUTPUT_BIN.txt", OUTPUT_BIN, 'delimiter', '');
103 dlmwrite("INPUT_DEC.txt", INPUT_DEC, 'delimiter', '\n');
104 dlmwrite("OUTPUT_DEC.txt", OUTPUT_DEC, 'delimiter', '\n');
```

VHDL - Butterfly_TB.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4  USE std.Textio.all;
5  USE ieee.std_logic_textio.all;
6
7
8  ENTITY Butterfly_TB IS
9  END Butterfly_TB;
10
11 ARCHITECTURE behaviour OF Butterfly_TB is
12
13     COMPONENT Butterfly
14         PORT
15     (
16         Butterfly_A_IN: IN SIGNED(15 DOWNTO 0);
17         Butterfly_B_IN: IN SIGNED(15 DOWNTO 0);
18         Butterfly_WR: IN SIGNED(15 DOWNTO 0);
19         Butterfly_WI : IN SIGNED(15 DOWNTO 0);
20         Butterfly_CLK: IN STD_LOGIC;
21         Butterfly_RST: IN STD_LOGIC;
22         Butterfly_START: IN STD_LOGIC;
23         Butterfly_DONE: OUT STD_LOGIC;
24         Butterfly_A_OUT: OUT SIGNED(15 DOWNTO 0);
25         Butterfly_B_OUT: OUT SIGNED(15 DOWNTO 0)
26     );
27     END COMPONENT;
28
```

```
29      -- Definizione segnali
30      SIGNAL A_IN_s, B_IN_s, WI_s, WR_s, A_OUT_s, B_OUT_s : SIGNED(15 DOWNTO 0);
31      SIGNAL CLK_s, RST_s, START_s: STD_LOGIC;
32      SIGNAL DONE_s: STD_LOGIC;
33
34      -- Definzizione costanti
35      CONSTANT CLK_PERIOD : TIME := 20 ns;
36      CONSTANT BIT_TIME : TIME := CLK_PERIOD*217;
37      CONSTANT TRANSMISSION_TIME : TIME := BIT_TIME*10;
38
39      BEGIN
40          DUT: Butterfly
41              PORT MAP
42              (
43                  Butterfly_A_IN => A_IN_s,
44                  Butterfly_B_IN => B_IN_s,
45                  Butterfly_WR => WR_s,
46                  Butterfly_WI => WI_s,
47                  Butterfly_CLK => CLK_s,
48                  Butterfly_RST => RST_s,
49                  Butterfly_START => START_s,
50                  Butterfly_A_OUT => A_OUT_s,
51                  Butterfly_B_OUT => B_OUT_s,
52                  Butterfly_DONE => DONE_s
53              );
54
55              -- Clock a 25 MHz (Funziona a quella velocit??)
56              Clock:
57              PROCESS IS
58                  BEGIN
59                      CLK_s <= '1', '0' AFTER CLK_PERIOD/2;
60                      WAIT FOR CLK_PERIOD;
61              END PROCESS;
62
63              --PROCESSO PER IL RESET
64              RST_process:
65              PROCESS IS
66                  BEGIN
67                      RST_s <= '1';
68                      WAIT FOR CLK_PERIOD*3;
69                      RST_s <='0';
70                      WAIT;
71              END PROCESS;
72
73              --PROCESSO PER LO START
74              START_process:
75              PROCESS IS
76                  BEGIN
77                      START_s <= '0';
78                      WAIT FOR CLK_PERIOD*10;
79                      WAIT FOR CLK_PERIOD/10; -- Piccolo delta
80
81                      FOR i IN 0 TO 1023 LOOP --
```

```
82                          START_s <='1';
83                          WAIT FOR CLK_PERIOD;
84                          START_s <='0';
85                          WAIT FOR CLK_PERIOD*3;
86                      END LOOP;
87
88                      WAIT;
89              END PROCESS;
90
91              READ_PROCESS:
92              PROCESS IS
93              FILE INPUT_BIN: text;
94              VARIABLE riga_read: LINE;
95              VARIABLE WR: STD_LOGIC_VECTOR(15 DOWNTO 0);
96              VARIABLE WI: STD_LOGIC_VECTOR(15 DOWNTO 0);
97              VARIABLE AR: STD_LOGIC_VECTOR(15 DOWNTO 0);
98              VARIABLE AI: STD_LOGIC_VECTOR(15 DOWNTO 0);
99              VARIABLE BR: STD_LOGIC_VECTOR(15 DOWNTO 0);
100             VARIABLE BI: STD_LOGIC_VECTOR(15 DOWNTO 0);
101
102                 BEGIN
103                 file_open(INPUT_BIN, "INPUT_BIN.txt", read_mode);
104
105                 -- Valori di DEFAULT
106                     WI_s <= "0000000000000000";
107                     WR_s <= "0000000000000000";
108                     A_IN_s <= "0000000000000000";
109                     B_IN_s <= "0000000000000000";
110
111                     while not endfile(INPUT_BIN) LOOP -- 1024 elaborazioni
112                         WAIT UNTIL START_s = '1';
113                         WAIT FOR CLK_PERIOD/10; -- Altrimenti i load non ...
                                funzionano bene
114                         WAIT FOR CLK_PERIOD;
115                         -- Lettura dati
116                         readline(INPUT_BIN, riga_read);
117                         read(riga_read, WR);
118                         readline(INPUT_BIN, riga_read);
119                         read(riga_read, WI);
120                         readline(INPUT_BIN, riga_read);
121                         read(riga_read, AR);
122                         readline(INPUT_BIN, riga_read);
123                         read(riga_read, AI);
124                         readline(INPUT_BIN, riga_read);
125                         read(riga_read, BR);
126                         readline(INPUT_BIN, riga_read);
127                         read(riga_read, BI);
128
129                         -- Dati
130                         WR_s <= signed(WR); -- DO IL COEFFICIENTE WR
131                         WI_s <= signed(WI); -- DO IL COEFFICIENTE WI
132                         A_IN_s <= signed(AR); -- DO IL COEFFICIENTE AR
133                         B_IN_s <= signed(BR); -- DO IL COEFFICIENTE BR
```

```
134                         WAIT FOR CLK_PERIOD;
135
136                         A_IN_s <= signed(AI); -- DO IL COEFFICIENTE AI
137                         B_IN_s <= signed(BI); -- DO IL COEFFICIENTE BI
138                         WAIT FOR CLK_PERIOD;
139                         -- :)
140                     END LOOP;
141
142                 file_close(INPUT_BIN);
143                 WAIT;
144             END PROCESS;
145
146             WRITE_PROCESS:
147             PROCESS IS
148             FILE OUTPUT_BIN: text;
149             VARIABLE riga_write: LINE;
150             VARIABLE AR_OUT : SIGNED(15 DOWNTO 0);
151             VARIABLE AI_OUT : SIGNED(15 DOWNTO 0);
152             VARIABLE BR_OUT : SIGNED(15 DOWNTO 0);
153             VARIABLE BI_OUT : SIGNED(15 DOWNTO 0);
154
155                 BEGIN
156                 file_open(OUTPUT_BIN, "OUTPUT_BIN_VHDL.txt", write_mode);
157                 -- Valori di DEFAULT
158
159
160                     FOR i IN 0 TO 1023 LOOP -- 1024 elaborazioni
161                         WAIT UNTIL DONE_s = '1';
162                         WAIT FOR CLK_PERIOD/10; -- Altrimenti i load non ...
                                funzionano bene
163                         WAIT FOR CLK_PERIOD;
164                         BR_OUT := B_OUT_s;
165                         AR_OUT := A_OUT_s;
166                         WAIT FOR CLK_PERIOD;
167                         AI_OUT := A_OUT_s;
168                         BI_OUT := B_OUT_s;
169
170                         -- Scrittura dati
171                         write(riga_write, std_logic_vector(AR_OUT)); -- AR_OUT
172                         writeline(OUTPUT_BIN, riga_write);
173                         write(riga_write, std_logic_vector(AI_OUT)); -- AI_OUT
174                         writeline(OUTPUT_BIN, riga_write);
175                         write(riga_write, std_logic_vector(BR_OUT)); -- BR_OUT
176                         writeline(OUTPUT_BIN, riga_write);
177                         write(riga_write, std_logic_vector(BI_OUT)); -- BI_OUT
178                         writeline(OUTPUT_BIN, riga_write);
179                     END LOOP;
180                     file_close(OUTPUT_BIN);
181                     WAIT;
182             END PROCESS;
183
184 END behaviour;
```

4.1.2 Testbench FFT 8 campioni

C - FFT eseguita in linguaggio C

Dati

Input.txt

```
0 0
0 0
0 0
1 0
1 0
0 0
0 0
```

Output.txt

```
0 2.000000 -0.000000 4.000000
1 -1.707107 0.707107 3.414214
2 1.000000 -1.000000 2.000000
3 -0.292893 0.707107 0.585786
4 0.000000 0.000000 0.000000
5 -0.292893 -0.707107 0.585786
6 1.000000 1.000000 2.000000
7 -1.707107 -0.707107 3.414214
```

Header files

CommonInclude.h

```
//
// Created by Elia Christian and Andrea on 10/01/19.
//

#ifndef DFT_COMMONINCLUDE_H
#define DFT_COMMONINCLUDE_H


#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <math.h>


///// DEFINE

struct comp{
    float r; // parte reale
```

```
19      float i; // parte immaginaria
20  };
21
22  typedef struct comp Comp;
23
24
25  #define LOG_NSAMPLE 3
26  #define NSAMPLE 8
27  // SMPLELEN    ricavato come FS/Fmin
28  //    il numero di campioni per ogni calcolo dell'fft
29  // gli zero aggiunti sono NSAMPLE-SAMPLEUSED
30  #define SAMPLEUSED 8
31  #define PI 3.14159265
32  #define DEBUG 1
33
34  #if DEBUG
35      #define DEBUG_PRINT(_fmt, ...)  fprintf(stderr, "[file %s, line %d]: " ...
            _fmt, __FILE__, __LINE__, __VA_ARGS__)
36      #define DEBUG_PRINT_COMP_VECT(beginIndex, endIndex, x) ...
            printCompVect(beginIndex, endIndex, x)
37  #else
38      #define  DEBUGPRINT(_fmt, ...) " "
39  #endif
40
41
42  #endif //DFT_COMMONINCLUDE_H
```

DFT.h

```
1  //
2  // Created by Elia Christian and Andrea on 04/01/19.
3  //
4
5  #ifndef DFT_DFTLIBRARY_H
6  #define DFT_DFTLIBRARY_H
7
8  #include "CommonInclude.h"
9
10 int16_t getReverseBit(int16_t n);
11 void Butterfly(Comp *A, Comp *B, Comp w);
12 void ReverseVect(Comp *X);
13 void FFT(Comp X[]);
14 void twiddle();
15
16 int16_t getFrequency(int16_t index, int16_t fs);
17 void printComp(char* name, Comp C, char* dopo);
18 int16_t findMaxPower(Comp* X);
19 #endif //DFT_DFTLIBRARY_H
```

SupportFunc.h

```
//
// Created by Elia Christian and Andrea on 11/01/19.
//

#ifndef DFT_SUPPORTFUNC_H
#define DFT_SUPPORTFUNC_H


#include "CommonInclude.h"


///// FUNCTION
void error(int8_t condition, char* str);
void printComp(char* name, Comp C, char* dopo);
void printCompVect(int16_t begin, int16_t end, Comp* x);
float power(Comp value);

#endif //DFT_SUPPORTFUNC_H
```

C files

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "../inc/DFT.h"
#include "../inc/SupportFunc.h"


void twiddle(Comp* W, int len);
int importFile(char* filename,Comp *x,int len);
int  importFileint(char* filename,int16_t x[],int len);
int saveOnFile(char* filename, Comp *X,int  len);
int  importFileZeroPadding(char* filename,Comp* x);

int main() {
    Comp x[NSAMPLE];
    importFile("../dati/Input.txt",x,SAMPLEUSED);
    FILE *fp=fopen("../dati/Output.txt","w+");
    int i;
    printf("File di ingresso:\n");
    printf("r  i\n");
    for(i=0; i<SAMPLEUSED; i++)
        printf("%d: %f %f\n",i, x[i].r, x[i].i);
    FFT(x);

    printf("FFT:\n");
    printf("r  i\n");
    for(i=0; i<SAMPLEUSED; i++)
        printf("%d: %f %f\n",i, x[i].r, x[i].i);
```

```
29
30      saveOnFile("../dati/Output.txt",x,SAMPLEUSED);
31      return 0;
32  }
33
34
35  int  importFile(char* filename,Comp* x,int len){
36      /*
37       * This function import "len" real value from file "filename"
38       * and save it in "x" rite in immaginary part 0
39       */
40      FILE *fp;
41      fp=fopen(filename,"r");
42      if(fp==0){
43          printf("error in importFile");
44          exit(1);
45      }
46      int i;
47      int supp;
48      for (i = 0; i < len; ++i) {
49          fscanf(fp,"%f %f",&x[i].r,&x[i].i);
50      }
51      return len;
52  }
53
54  int  importFileZeroPadding(char* filename,Comp* x){
55      /*
56       * This function import "len" real value from file "filename"
57       * and save it in "x" rite in immaginary part 0
58       */
59      FILE *fp;
60      fp=fopen(filename,"r");
61      if(fp==0){
62          printf("error in importFile");
63          exit(1);
64      }
65      int i;
66      for (i = 0; i < SAMPLEUSED; ++i) {
67          fscanf(fp,"%f",&x[i].r);
68          fscanf(fp,"%f", &x[i].i);
69      }
70      //for (int j = SAMPLEUSED; j < NSAMPLE; ++j) {
71      //    x[j].r=0, x[j].i=0;
72      //}
73      return NSAMPLE;
74  }
75
76  int saveOnFile(char* filename, Comp *X,int len){
77      /*
78       * This function save vector X in a file
79       * column 1 -> save i that is equal to number of element
80       * column 2 -> save real part
81       * column 3 -> save immaginary part
```

```
82       * column 4 -> save power
83      */
84      FILE *fp;
85      fp=fopen(filename,"a");
86      if(fp==0){
87          printf("error in saveOnFile");
88          exit(1);
89      }
90      int i;
91      int supp;
92      for (i = 0; i < len; ++i) {
93          fprintf(fp,"%d %f %f %f\n",i,X[i].r,X[i].i,X[i].r*X[i].r+X[i].i*X[i].i);
94      }
95      fclose(fp);
96      return len;
97  }
98  int  importFileint(char* filename,int16_t x[],int len){
99      /*
100      * This function import "len" real value from file "filename"
101      * and save it in "x" rite in immaginary part 0
102      */
103     FILE *fp;
104     fp=fopen(filename,"r");
105     if(fp==0){
106         printf("error in importFile");
107         exit(1);
108     }
109     int i;
110     int supp;
111     for (i = 0; i < len; ++i) {
112         fscanf(fp,"%d",&x[i]);
113     }
114     return len;
115 }
```

DFT.c

```
1  /*
2  * Created by Elia Christian and Andrea on 15/12/18.
3  * This file require  definition of NSAMPLE in an other header file to ...
       correctly work
4  */
5
6
7  #include <stdint.h>
8  #include <nss.h>
9  #include "../inc/DFT.h"
10
11
12 ///////////////  DEFINE    ////////////////////////////
13 // this variable set if windowing function used saved value for window
```

```
14 // or every time calculate value by window function
15     #define MEMORY_ON 0
16
17     #if MEMORY_ON
18         // I create a Comp structure to contain twiddle factor for decrease ...
               computing power
19         // but increase use of memory
20         Comp Tw_Factor[NSAMPLE/2];
21         // this two function are used in code and substitute previous vector ...
               with correct
22         // index
23         #define TWFR(index) Tw_Factor[index].r
24         #define TWFI(index) Tw_Factor[index].i
25         int twiddleFlag = 0;
26     #else
27         // With this alternative memory is less used but every time that i ...
               need a twiddle
28         // factor i must evaluate this two function, however this two define ...
               substiture the
29         // following expression in preprocessor elaboration
30         #define TWFR(index) (float)cos(2*PI*(index)/((float)NSAMPLE))
31         #define TWFI(index) (float)sin(2*PI*(index)/((float)NSAMPLE))
32     #endif
33
34 /////////////// FUNCTION  ////////////////////////////////
35
36
37 int16_t getReverseBit(int16_t n){
38     /*
39      * All this calculus is made to make inversion bit-to-bit of n parameter ...
            passed
40      * as argument and return it
41      */
42     #if NSAMPLE==8
43         // 3 bit
44         n= (n&0x0001)<<2 | (n&0x0004)>>2 | (n&0X0002);
45         return n;
46     #elif NSAMPLE==4
47         // 2 bit
48         n= (n&0x0001)<<1 | (n&0x0002) >>1;
49         return n
50     #elif NSAMPLE==16
51         // 4 bit
52         n=(n&0x0003)<<2 | (n&0x000C)>>2;
53         n=(n&0x0004)<<1 | (n&0x0008)>>1 |(n&0x0002)>>1 |(n&0x0001)<<1;
54         return n;
55     #elif NSAMPLE==32
56         // 5 bit
57         n=(n&0x0003)<<3 | (n&0x0018)>>3 | (n&0x0004);
58         n=(n&0x0008)<<1 | (n&0x0010)>>1 |(n&0x0002)>>1 |(n&0x0001)<<1 | ...
               (n&0x0004);
59         return n;
60     #elif NSAMPLE==64
```

```
61          // 6 bit
62          n=(n&0x0038)>>3 | (n&0x0007)<<3;
63          n=(n&0x0020)>>2 | (n&0x0008)<<2 | (n&0x0004)>>2 | (n&0x0001)<<2;
64          return n;
65      #elif NSAMPLE==2048
66          // 11 bit
67          n=(n&0x07C0)>>6 | (n&0x001F)<<6 | (n&0x0020);
68          n=(n&0x0600)>>3 | (n&0x00C0)<<3 | (n&0x0100) | (n&0x0020) | ...
                (n&0x0018)>>3 | (n&0x0003)<<3 | (n&0x0004);
69          n=(n&0x0400)>>1 | (n&0x0200)<<1 | (n&0x0080)>>1 | (n&0x0040)<<1 | ...
                (n&0x0010)>>1 | (n&0x0008)<<1 |  (n&0x0002)>>1 | (n&0x0001)<<1 ...
                |(n&0x0100)| (n&0x0020) | (n&0x0004);
70          return n;
71      #else
72          #error DEBUGPRINT("error due to NSAMPLE constant which refer to a not ...
                implemented getReverseBit function\n");
73      #endif
74  }
75
76  void Butterfly(Comp *A, Comp *B, Comp w){
77      /*
78       * A -> primo numero complesso dove verr  messo il primo risultato A +W*B
79       * B -> secondo numero complesso verss  messo qua A- W*B
80       * w -> twiddle factor
81       * i calcoli che vengono fatti sono un'ottimizzazione del prodotto e somma
82       * fra numeri complessi ma sono comunque i calcoli indicati in precedenza
83       */
84      Comp sup;
85      //printf("prima ");
86      //printComp("A",*A," "), printComp("B",*B," "), printComp("W",w,"\n");
87      sup.r = A->r + B->r*w.r - B->i*w.i;
88      sup.i = A->i + B->i*w.r + B->r*w.i;
89      B->r = 2*A->r - sup.r;
90      B->i = 2*A->i - sup.i;
91      A->r = sup.r;
92      A->i = sup.i;
93      //printf("dopo ");
94      //printComp("A",*A," "), printComp("B",*B," "), printComp("W",w,"\n");
95  }
96  void ReverseVect(Comp *X){
97      /*
98       * X    il vettore nel tempo che dev'essere invertito secondo l'ordine
99       * reverse bit.
100      *
101      * IMPORTANTE!! LOGNSAMPLE non    altro che il logaritmo base 2 di ...
             NSAMPLE, essendo
102      * questo un algoritmo da eseguire su un sistema embedded sono state ...
             omessi i controlli
103      * sul fatto che NSAMPLE sia effettivamente una potenza di 2 e LOGNSAMPLE ...
             il suo logaritmo
104      *
105      * La funzione salva in X il nuovo ordine corretto del vettore
106      *
```

```
107       * V1 nella prima versione l'algoritmo non    ottimizzato perch  ripete ...
              molte sostituzioni
108       *      che vengono quindi fatte due volte durante il ciclo
109       */
110      int16_t i,newi;
111      Comp temp;
112      for(i=0; i< NSAMPLE; i++) {
113          // trovo il reverse bit corrispondente
114          newi = getReverseBit(i);
115          //printf("%d, ", newi);
116          // se newi<i vuol dire che ho gi  fatto lo scambio
117          if (newi > i) {
118              //printf("-%f %f   ", X[newi].r, X[i].r);
119              temp.r = X[newi].r;
120              //printf("--%f\n",temp.r);
121              X[newi].r = X[i].r;
122              X[i].r = temp.r;
123          }
124      }
125      //DEBUG_PRINT_COMP_VECT(0,NSAMPLE,X);
126
127  }
128
129  void FFT(Comp X[]){
130      /*
131       * X    il vettore nel tempo che dev'essere portato nel dominio del tempo
132       * NSAMPLE    la lunghezza di X
133       * ts    il tempo di campionamento
134       * LOGNSAMPLE    il logaritmo in base N di NSAMPLE
135       *
136       * IMPORTANTE!! LOGNSAMPLE non    altro che il logaritmo base 2 di ...
              NSAMPLE, essendo
137       * questo un algoritmo da eseguire su un sistema embedded sono state ...
              omessi i controlli
138       * sul fatto che NSAMPLE sia effettivamente una potenza di 2 e LOGNSAMPLE ...
              il suo logaritmo
139       *
140       * La funzione porta nel dominio del tempo X che considera come campioni nel
141       * tempo campionati con tempo di campionamento pari a ts, viene quindi ...
              effettuata la
142       * trasformata di Fourier complessa utilizzando l'algoritmo di Cooley-Tuckey
143       */
144      int i;
145      int n=1; // viene moltiplicato per 2 ad ogni strato di butterfly
146      int a=NSAMPLE/2; // viene diviso per 2 ad ogni strato
147      int s ;
148
149      // inverto l'ordine del vettore d'ingresso
150      ReverseVect(X);
151
152      // creo i twiddle factor salvandoli in Tw_Factor vector only if MEMORY_ON ...
             macro is asserted
153      #if MEMORY_ON
```

```
154         if(!twiddleFlag)
155             twiddle();
156     #endif
157     // twiddle factor di supporto
158     Comp Twf,sup1,sup2;
159     //FILE *fp=fopen("FFT.txt","a+");
160 
161     for(s=0;s<LOG_NSAMPLE; s++) {
162         for (i = 0; i < NSAMPLE; i++) {
163             // looking as butterfly must done there is a pattern in bits ...
                    sequence that
164             // are equal to its calculation
165             if (!(i & n)) {
166                 // inizializzo il twiddle factor usando la direttiva di ...
                        preprocessore
167                 Twf.i = TWFI((i * a));
168                 Twf.r = TWFR((i * a));
169                 printf("TWfn :%d r:%f i:%f\n",i*a%8,Twf.i,Twf.r);
170                 sup1=X[i],sup2=X[i+n];
171                 Butterfly(&X[i], &X[i + n], Twf);
172                 printf("Butterfly: %d %d -> B=%f %f,  A=%f %f\n\n",s, i, ...
                        X[i+n].r, X[i+n].i,  X[i].r, X[i].i);
173 
174             }
175         }
176         a /= 2;
177         n *= 2;
178     }
179 }
180 
181 
182 int16_t getFrequency(int16_t index, int16_t fs){
183     return fs*index/NSAMPLE;
184 }
185 
186 
187 
188 int16_t findMaxPower(Comp* X){
189     /*
190      * this function loop in X vector and find it maximum value, returning ...
            its value
191      */
192     float max=0,sup;
193     int16_t index,i;
194     for (i = 0; i < NSAMPLE; ++i) {
195         sup = X[i].r*X[i].r+X[i].i*X[i].i;
196         if(sup>max)
197             max=sup, index=i+1;
198     }
199     return index;
200 }
201 
202 
```

```
203 #if MEMORY_ON
204     void twiddle(){
205         /*
206          * W    il vettore vuoto dove verranno salvati i twiddle factor calcolati
207          * NSAMPLE    il numero di campioni
208          */
209         int i;
210         for(i=0; i<NSAMPLE/2; i++){
211             Tw_Factor[i].r = (float)cos(2*PI*i/((float)NSAMPLE));
212             Tw_Factor[i].i = (float)sin(2*PI*i/((float)NSAMPLE));
213         }
214         twiddleFlag=1;
215     }
216 #endif
```

SupportFunc.c

```
1  //
2  // Created by Elia Christian and Andrea on 11/01/19.
3  //
4
5  #include "../inc/SupportFunc.h"
6
7  void error(int8_t condition, char* str){
8      if(condition){
9          printf("%s",str);
10         exit(1);
11     }
12 }
13
14 // funzioni di debug
15
16 void printComp(char* name, Comp C, char* dopo){
17     printf("%s=%.2f i%.2f%s",name,C.r,C.i,dopo);
18 }
19
20 void printCompVect(int16_t begin, int16_t end, Comp* x){
21     int i;
22     for(i=begin;i<end+1;i++){
23        DEBUG_PRINT("----%f i*%f\n",x[i].r, x[i].i);
24     }
25 }
26 float power(Comp value){
27     return (value.r*value.r);
28 }
```