Politecnico di Torino

Master degree in Electronic Engineering

Progetto di Architetture Integrate

Progetto di un circuito digitale per il calcolo FFT basato sull'algoritmo Cooley-Tukey

Sistemi Digitali Integrati

Professore: Maurizio Zamboni

Autore:
Lombardini Luca (277972)

February 12, 2021

Contents

List of Figures

List of Tables

CHAPTER 1

Progettazione unità Butterfly

1.1 Introduzione

Questo documento di progetto tratta dello sviluppo di un circuito digitale di calcolo, il cui scopo è quello di descrivere una possibile implementazione dell'elemento base *Butterfly*, utilizzato in vari ambiti tra cui crittografia e, più classicamente, il calcolo di FFT basate sull'algoritmo di Cooley-Tuckey Radix-2. Il circuito alla fine verrà testato in una batteria modulare per il calcolo di FFT 16x16. Il linguaggio di descrizione dell'Hardware di tale progetto è *VHDL*.

Durante lo sviluppo di tale progetto, un insieme di specifiche e limiti forniti dal committente, è stato seguito e soddisfatto. Tali imposizioni sono:

- implementazione Radix-2 a valori complessi
- dati e e coefficienti in rappresentati in forma frazionaria, complemento a 2 su 20 bits
- dati in valore compreso tra -0.5 e +0.5, estremi esclusi
- overflow impedito con tecniche *"Guard bit"* e *"Unconditional block floating point scaling"*
- control unit microprogrammata con sequenziatore a indirizzamento esplicito e *Late Status*
- 1 solo moltiplicatore/raddoppiatore a singolo livello di pipeline interna
- 2 sommatori/sottrattori a singolo livello di pipeline interna
- interfacce esterne a 20 bits, ma calcoli interni senza perdita di precisione
- arrotondamento solo sul risultato finale, basato su *Rounding to nearest even*
- ottimizzazione volta a ridurre al minimo l'utilizzo di variabili temporanee
- modalità di esecuzione isolata o continua senza segnale di discrimine
- interfacciabilità tra unità dello stesso tipo
- dati e coefficienti inviati/ricevuti su porte diverse
- segnale di avvio *START* e di termine esecuzione *DONE*
- ottimizzazione del numero di bus globali e locali per ridurre costo senza penalità in prestazioni

1.2 Derivazione del Data Flow Diagram

Osservando i tipi di calcolo che l'unità *Butterly* deve eseguire, tutto si riconduce a prodotti e somme. Si realizza inizialmente un progetto a livello algortmico che non tiene conto delle risorse Hardware limitate, partendo così dal caso in cui si hanno le prestazioni più elevate possibili, in termini di *Latenza* e *Throughput*: tale situazione è di fatto un *"Upper Bound"* per l'insieme delle soluzioni circuitali (aggiungere ulteriore Hardware non incrementa le prestazioni in termini di riduzione degli step algoritmici richiesti per la computazione).
Partendo dall'insieme delle operazioni eseguite dalla singola unità, si possono etichettare e raggruppare in operazioni semplici, ove l'apice indica il risultato:

- $A'_R = A_R + B_R * W_R - B_I * W_I$
- $A'_I = A_I + B_R * W_I - B_I * W_R$
- $B'_R = 2 * A_R - A'_R$
- $B'_I = 2 * A_I - A'_I$

Raggruppabili in:

- $M_1 = B_R * W_R$
- $M_2 = B_I * W_I$
- $M_3 = B_R * W_I$
- $M_4 = B_I * W_R$
- $M_5 = 2 * A_R$
- $M_6 = 2 * A_I$
- $S_1 = A_R + M_1$
- $S_2 = S_1 - M_2 \rightarrow A'_R$
- $S_3 = A_I + M_3$
- $S_4 = S_3 + M_4 \rightarrow A'_I$
- $S_5 = M_5 - S_2 \rightarrow B'_R$
- $S_6 = M_6 - S_4 \rightarrow B'_I$

1.2.1 Analisi con approccio ASAP

L'approccio che permette di ottenere il minor numero di unità lungo un cammino (e quindi step algoritmici quando si considererà il *CDFG*) e che instanzia il maggior numero di componenti Hardware è l'approccio *ASAP*: ogni calcolo è eseguito (o instanziato o allocato) appena i dati da cui dipende sono disponibili. Il *DFG* che ne consegue è il mostrato in *figura 1.1*.

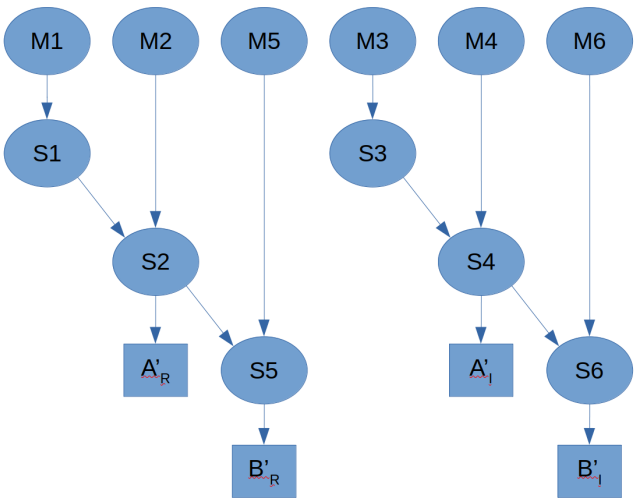


Figure 1.1: Data Flow Graph della soluzione *ASAP* senza limitazioni

Come si può notare, le *Data Dependencies*, che sono le relazioni di dipendenza tra operatori e dati richiesti per quell'elaborazione, sono il limite per cui anche potendo instanziare più Hardware, le prestazioni non aumenterebbero.

Partendo da questo risultato si osserva che il circuito equivalente al grafo di *figura 1.1*, se rapportato ai limiti imposti dal numero massimo di operatori utilizzabili, necessiterebbe di 6 moltiplicatori al primo step, a fronte del singolo moltiplicatore disponibile. Sarebbe una soluzione compatibile dal punto di vista dei sommatori.

Di fatto questo algoritmo è distinguibile in 2 sotto-alberi: l'albero di sinistra genera i valori reali, quello di destra i valori complessi dei 2 rispettivi risultati in uscita.

1.2.2 Analisi con approccio ALAP

Sulla base di questo risultato ora si procede con un approccio *ALAP*: ogni calcolo è eseguito (o instanziato o allocato) il più tardi possibile, in modo che gli operatori a valle abbiano i dati per il loro calcolo giusto in tempo. Tale approccio permette di ridurre considerevolmente il costo in Hardware in termini di operatori paralleli e variabili temporanee necessarie (rispetto ad *ASAP*). Il *DFG* che ne consegue è il mostrato in *figura 1.2*.

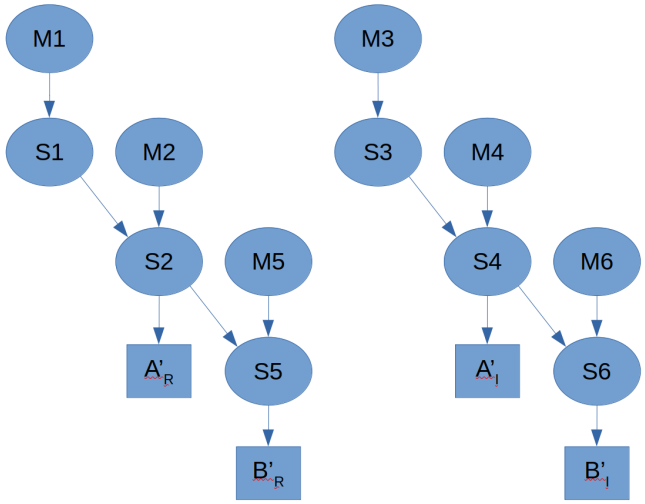


Figure 1.2: Data Flow Graph della soluzione *ALAP* senza limitazioni

Anche l'approccio puramente *ALAP*, seppur implementabile dal punto di vista dei sommatori, richiederebbe 2 moltiplicatori per ognuno dei primi 3 step algoritmici, a fronte del singolo disponibile.

1.2.3 Analisi con approccio Ibrido

Dato che gli approcci base puramente *ASAP* o *ALAP* non sono implementabili, è possibile applicare approcci ibridi, ovvero che non ricadono nella definizione dei due precedenti.
Dato che l'approccio *ALAP* è quello che tra i due ha oltrepassato il limite con il minor numero di moltiplicatori, è l'attuale punto di partenza. Questa volta, inoltre, verrà considerata anche la durata degli operatori e la relazione con gli step algoritmici, relativa ad una macchina dotata di *Control Unit*.
Operando un *interleaving* sullo scheduling degli operatori, ovvero favorendo le diverse assegnazioni in cicli dedicati, si passa da un grafo con caratteristiche parallele ad uno con caratteristiche più seriali. Otterremo pertanto un *CDFG*, ovvero il Control Data Flow Graph come mostrato in *figura 1.3*.

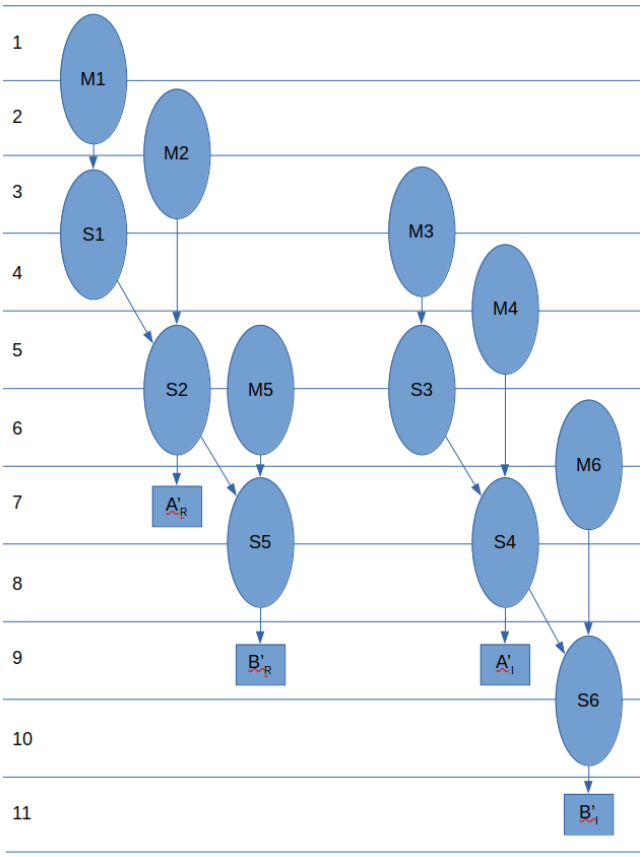
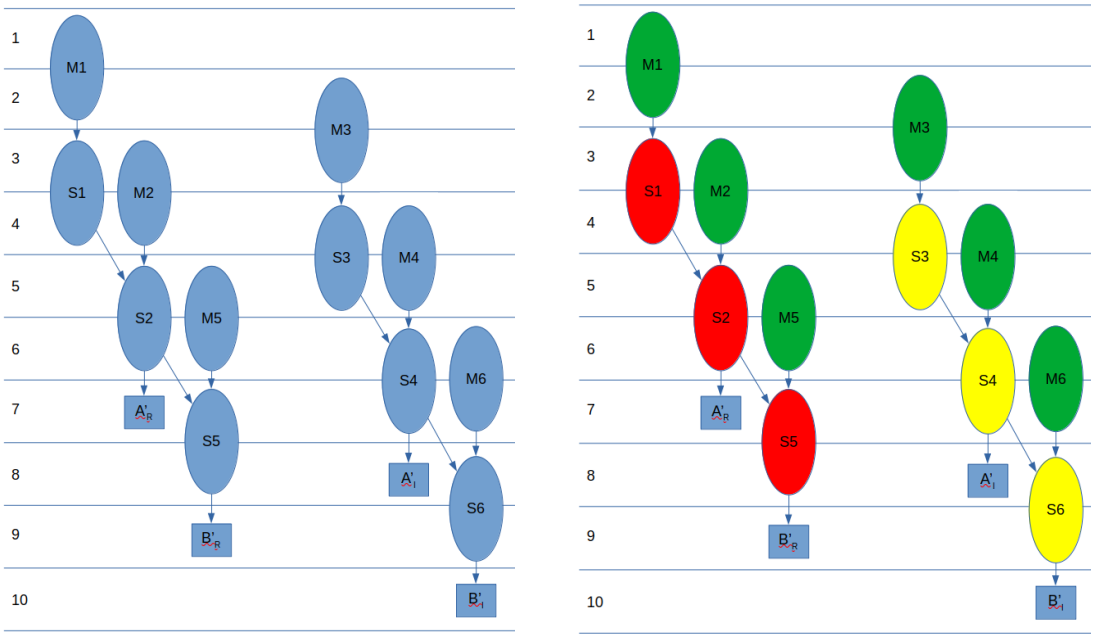


Figure 1.3: Control Data Flow Graph della soluzione ibrida

Ogni step è racchiuso all'interno di un livello di pipeline, i cui registri in questo grafo sono stati omessi per questioni grafiche e di lettura.
Tale organizzazione è implementabile, dato che il numero massimo di sommatori e moltiplicatori è rispettato per ogni step algoritmico. Però, presenta una caratteristica che non è ottimale: è presente un registro in più tra le coppie di operazioni M_2 ed S_2, M_4 ed S_4, M_6 ed S_6.
Questa condizione può essere risolta riarrangiando gli operatori ritardando M_2, e idealmente traslando l'albero di destra verso l'alto di uno step, ma mantenendo M_4 e M_6 nella posizione attuale. Tale scelta, che comporta solo un cambiamento nell'ordine di input e output, verrà motivata alla *sezione 1.2.5*. Il *CDFG* ottenuto è qui sotto riportato in *figura 1.4*.



(a) Scheduling

(b) I colori identificano l'operatore utilizzato: verde per il moltiplicatore, rosso il primo sommatore e giallo il secondo

Figure 1.4: Control Data Flow Graph della soluzione ibrida, con riordinamento

L'assegnazione dei due sommatori per i due rami non ha importanza a primo impatto. In realtà certe combinazioni di allocazione/ordine permettono di semplificare le interconnessioni degli elementi di calcolo e memoria all'interno del circuito digitale. Tale ordine verrà discusso più avanti alla *sezione 1.2.6*. Osservando la *figura 1.4*, si nota che il momento in cui la singola unità *Butterfly* può iniziare un nuovo calcolo, in previsione del funzionamento in modalità continua, è quando il moltiplicatore è svincolato da nuove moltiplicazioni: ciò avviene allo step 7, ove può essere instanziata M_1 del prossimo gruppo di dati. Il moltiplicatore è dunque il collo di bottiglia.
Una possibile ottimizzazione, non applicabile per questioni didattiche (prodotto per 2 non è da implementare in modi differenti dall'uso del moltiplicatore), è la realizzazione dei prodotti M_5 ed M_6 come somma di un dato con sé stesso ($2 * A_X \rightarrow A_X + A_X$), e riassegnando le operazioni di somma, più precisamente assegnando S_1, S_2, S_3 ed S_4 al primo sommatore, S_5 ed S_6 al secondo: in questo modo il carico computazionale sarebbe equidistribuito sulle varie unità (4 calcoli ciascuna), permettendo di poter iniziare un nuovo calcolo ogni 4 cicli, invece degli attuali teorici.
Ne risulterebbe il *CDFG* di *figura 1.5*

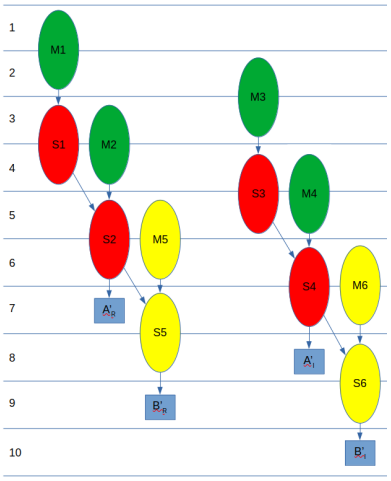


Figure 1.5: Control Data Flow Graph della soluzione ottimizzata, ma non accettabile

ATTENZIONE: il numero di cicli citati fino a questo punto sono da considerarsi relativi alla sola rappresentazione dei *CDFG*, l'effettivo numero di step per l'architettura finale, con *Control Unit*, potrebbe variare a causa di stati aggiuntivi per arrotondamento, caricamento dati e stati di idle. Da ora in poi verranno considerati, dato che man mano il *Data Flow Diagram* definitivo sta prendendo forma.

1.2.4 Analisi e ottimizzazione delle variabili temporanee

L'unità *Butterfly* durante la sua esecuzione produce risultati parziali, utilizzati successivamente da altre unità di calcolo. Questo comporta l'uso di elementi di memoria, ovvero registri, che fungono da interfacciamento tra le unità e storage temporaneo. Ottimizzarle significa ridurre il numero di registri al minimo indispensabile, ottenendo un'area occupata minore.
Un primo accorgimento è già stato applicato alla *sezione 1.2.3*, il quale permette di risparmiare il suddetto registro.

Nota: Il numero di registri impiegati dipende dal tipo di algoritmo scelto, ovvero come trattato alla *sezione 1.2.5*, dall'ordine in cui i dati entrano nell'unità di calcolo e da come vi escono.

Nota: Il numero di registri impiegati dipende inoltre anche da come viene risolta la problematica trattata nella *sezione 1.2.6* degli step durante i quali avviene la lettura di 3 operatori, a fronte dei normali 2.
Pertanto, i casi in analisi sono 5:

- Ordine ingressi e uscite $[A_R, A_I, B_R, B_I]$
- Ordine ingressi e uscite $[B_R, B_I, A_R, A_I]$
- Ordine ingressi e uscite a coppia $[A_R, B_R]$ e $[A_I, B_I]$

Dove entrambi hanno la risolto la dipendenza con:

- Utilizzo di un BUS dedicato (unica soluzione possibile per ordine $[B_R, B_I, A_R, A_I]$)
- Ri-uso di registri già presenti

I diagrammi del tempo di vita delle variabili ottenuti considerano anche gli step non prettamente di calcolo, necessari per la soddisfazione delle specifiche e aventi costo in termini di *Hardware* e tempo. Tali diagrammi caratteristici sono mostrati in *figura 1.6*.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Ar	Saved					S1		M5							
Ai		Saved					S3		M6						
Br			Saved	M1	M3										
Bi				Saved		M2	M4								
Wr			Saved	M1			M4								
Wi				Saved	M3	M2									
M1						S1									
M2								S2							
M3							S3								
M4									S4						
M5										S5					
M6											S6				
SD1															
SD2															
S1								S2							
S2										S5					
S3									S4						
S4											S6				
S5												Round B'r			
S6													Round B'i		
OUT											SEND A'r	SEND A'i	SEND B'r	SEND B'i	
# di Registri	1	2	4	6	6	6	5	4	3	3	3	2	1	1	
Operazioni				Br * Wr	Br * Wi	Bi * Wi	Bi * Wr	2 * Ar	2 * Ai	Round A'r	Round A'i	Round B'r	Round B'i		
						Ar + M1	Ai + M3	S1 – M2	S3 + M4	M5 – S2	M6 – S4				

(a) Ordine $[A_R, A_I, B_R, B_I]$ con BUS dedicato al feed di S1 ed S3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Ar	Saved	SD1						M5							
Ai		Saved	SD2						M6						
Br			Saved	M1	M3										
Bi				Saved		M2	M4								
Wr			Saved	M1			M4								
Wi				Saved	M3	M2									
M1						S1									
M2								S2							
M3							S3								
M4									S4						
M5										S5					
M6											S6				
SD1				Ar		S1									
SD2					AI		S3								
S1								S2							
S2										S5					
S3									S4						
S4											S6				
S5												Round B'r			
S6													Round B'i		
Operazioni				Br * Wr	Br * Wi	Bi * Wi	Bi * Wr	2 * Ar	2 * Ai	Round A'r	Round A'i	Round B'r	Round B'i		
		Ar + 0	Ai + 0			Ar + M1	Ai + M3	S1 – M2	S3 + M4	M5 – S2	M6 – S4				
OUT											SEND A'r	SEND A'i	SEND B'r	SEND B'i	
# di Registri	1	2	4	7	8	8	6	4	3	3	3	2	1	1	

(b) Ordine $[A_R, A_I, B_R, B_I]$ con riuso dei registri temporanei in feedback ai sommatori

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Ar			Saved	S1		M5									
Ai				Saved	S3		M6								
Br	Saved	M1	M3												
Bi		Saved		M2	M4										
Wr	Saved	M1			M4										
Wi		Saved	M3	M2											
M1				S1											
M2						S2									
M3					S3										
M4							S4								
M5								S5							
M6									S6						
S1						S2									
S2								S5							
S3							S4								
S4									S6						
S5										Round B'r					
S6											Round B'i				
OUT											SEND B'r	SEND B'i	SEND A'r	SEND A'i	
# di Registri	2	4	5	6	5	4	3	2	3	3	4	3	2	1	
Operazioni		Br * Wr	Br * Wi	Bi * Wi	Bi * Wr	2 * Ar	2 * Ai	Round A'r	Round A'i	Round B'r	Round B'i				
				Ar + M1	Ai + M3	S1 – M2	S3 + M4	M5 – S2	M6 – S4		Send B'r	Send B'i	Send A'r	Send A'i	

(c) Ordine $[B_R, B_I, A_R, A_I]$

Figure 1.6: Diagrammi del tempo di vita delle variabili ibrido: la colorazione suggerisce le dipendenze come per *figura 1.4*, ma con le nuove azioni da considerare e il loro scheduling e dipendenze

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Ar	Saved			S1		M5									
Ai		Saved			S3		M6								
Br	Saved	M1	M3												
Bi		Saved		M2	M4										
Wr	Saved	M1			M4										
Wi		Saved	M3	M2											
M1				S1											
M2						S2									
M3					S3										
M4							S4								
M5								S5							
M6									S6						
S1						S2									
S2								S5							
S3							S4								
S4									S6						
S5										Round B'r					
S6											Round B'i				
Operazioni		Br * Wr	Br * Wi	Bi * Wi	Bi * Wr	2 * Ar	2 * Ai	Round A'r	Round A'i	Round B'r	Round B'i				
				Ar + M1	Ai + M3	S1 – M2	S3 + M4	M5 – S2	M6 – S4						
OUTA											SEND A'r	SEND A'i			
OUTB											SEND B'r	SEND B'i			

(a) Ordine a coppia $[A_R, B_R][A_I B_I]$ con BUS dedicato al feed di S1 ed S3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Ar	Saved	SD1				M5									
Ai		Saved	SD2				M6								
Br	Saved	M1	M3												
Bi		Saved		M2	M4										
Wr	Saved	M1			M4										
Wi		Saved	M3	M2											
SD1				S1											
SD2					S3										
M1				S1											
M2						S2									
M3					S3										
M4							S4								
M5								S5							
M6									S6						
S1						S2									
S2								S5							
S3							S4								
S4									S6						
S5										Round B'r					
S6											Round B'i				
Operazioni		Br * Wr	Br * Wi	Bi * Wi	Bi * Wr	2 * Ar	2 * Ai	Round A'r	Round A'i	Round B'r	Round B'i				
				Ar + M1	Ai + M3	S1 – M2	S3 + M4	M5 – S2	M6 – S4						
OUTA											SEND A'r	SEND A'i			
OUTB											SEND B'r	SEND B'i			

(b) Ordine a coppia $[A_R, B_R][A_I B_I]$ con riuso dei registri temporanei in feedback ai sommatori

Figure 1.7: Proseguimento

Osservando le varie *figure 1.6* si nota subito che il numero di registri cambia nei vari step algoritmici, ma il risultato che conta è il numero totale di registri:

- La soluzione in *figura 1.6a* ha un massimo di 6 registri in uso
- La soluzione in *figura 1.6b* ha un massimo di 8 registri in uso
- La soluzione in *figura 1.6c* ha un massimo di 6 registri in uso
- La soluzione in *figura 1.7a* ha un massimo di 6 registri in uso
- La soluzione in *figura 1.7b* ha un massimo di 7 registri in uso

Dal punto di vista dei registri utillizzati convengono le soluzioni con ordine $[A_R, A_I, B_R, B_I]$, $[B_R, B_I, A_R, A_I]$ e ordine a coppia $[A_R B_R]$ $[A_I B_I]$ (tutte con BUS dedicato per il feed di $S1$ ed $S3$). A questo punto, la scelta della soluzione da implementare dipende da altri fattori, tra cui *Throughput*, costo architetturale e della *Control Unit* e interfacciabilità.

Nota: Il costo introdotto dal riordino dei dati mostrato nelle *figure 1.6c, 1.7a e 1.7b* è quello per l'unità di riordino al fondo dell'algoritmo (colore azzurro), che consta di fatto in registri aggiuntivi e multiplexers, interposti tra l'unità di arrotondamento e i registri di uscita.

1.2.5 Analisi delle interfacce verso l'esterno

Analizzando il comportamento dell'algoritmo di *Cooley-Tukey* per quanto riguarda gli ingressi e uscite in termini di relazione temporale, si osserva che l'ordine di utilizzo dei dati di ingresso è $[B_x, B_y, A_x, A_y]$, mentre i dati in uscita vengono prodotti nell'ordine $[A_x, A_y, B_x, B_y]$. In pratica, l'ordine del numero A e del numero B è invertito da ingresso a uscita. L'ordine discende dalle *Data Dependencies* e dalle scelte di ordinamento degli operatori.
I pedici x e y sono relativi alla parte reale e immaginaria, la quale dipende da quale dei due alberi, citati ne *sezione 1.2.3*, è anticipato rispetto all'altro. La x corrispone a parte reale e y a parte immaginaria se l'albero di sinitra è anticipato ripetto a quello di destra e viceversa. La scelta di uno è equivalente all'altro, dal punto di vista delle risorse *Hardware* e temporali, a causa delle dipendenze che producono lo stesso tipo di scheduling all'interno dell'albero.
La soluzione adottata è quella di anticipare l'albero sinistro delle parti reali.

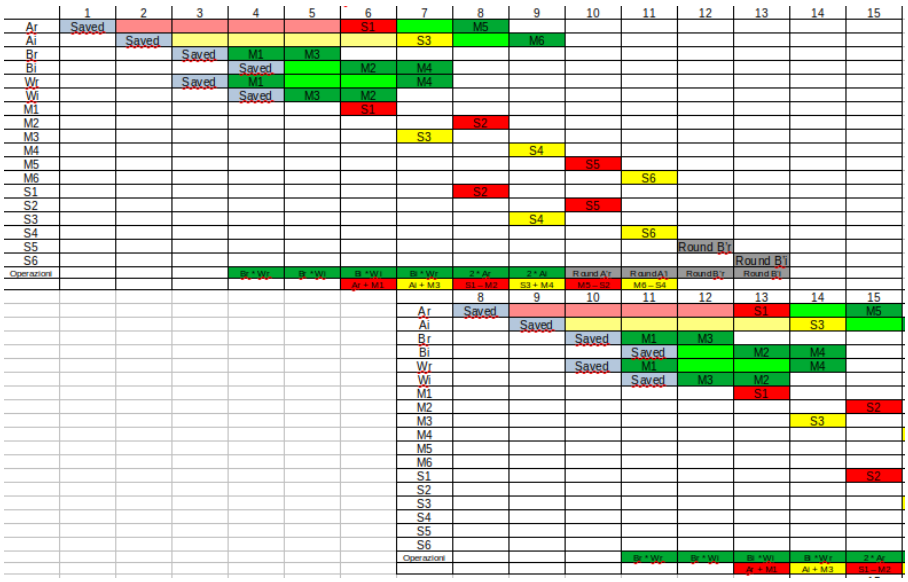
Ciò che conta a questo punto è che per far si che ogni unità sia identica alle altre e tra loro interfacciabili, la sequenza dei dati accettati in ingresso deve coincidere con la sequenza dei dati prodotti in uscita. Le soluzioni sono le stesse mostrate alla *sezione 1.2.3*: $[A_R, A_I, B_R, B_I]$, $[B_R, B_I, A_R, A_I]$ o a coppia $[A_R B_R]$ $[A_I B_I]$.
Dal punto di vista della durata dell'algoritmo della singola *Butterfly*, le prime due soluzioni sono equivalenti: entrabe hanno latenza di 14 cicli, come mostrato in *figura 1.6*. La terza presenta una latenza di 12 cicli. Inoltre le varie soluzioni presentano diverse frequenze di ripetizione di calcolo in modalità continua. Come si può vedere nella *figura 1.9* le diverse frequenze sono:

- La soluzione in *figura 1.8a* può iniziare un nuovo calcolo ogni 7 cicli
- La soluzione in *figura 1.8b* può iniziare un nuovo calcolo ogni 9 cicli
- La soluzione in *figura 1.8c* può iniziare un nuovo calcolo ogni 6 cicli
- La soluzione in *figura 1.9a* può iniziare un nuovo calcolo ogni 6 cicli
- La soluzione in *figura 1.9b* può iniziare un nuovo calcolo ogni 7 cicli

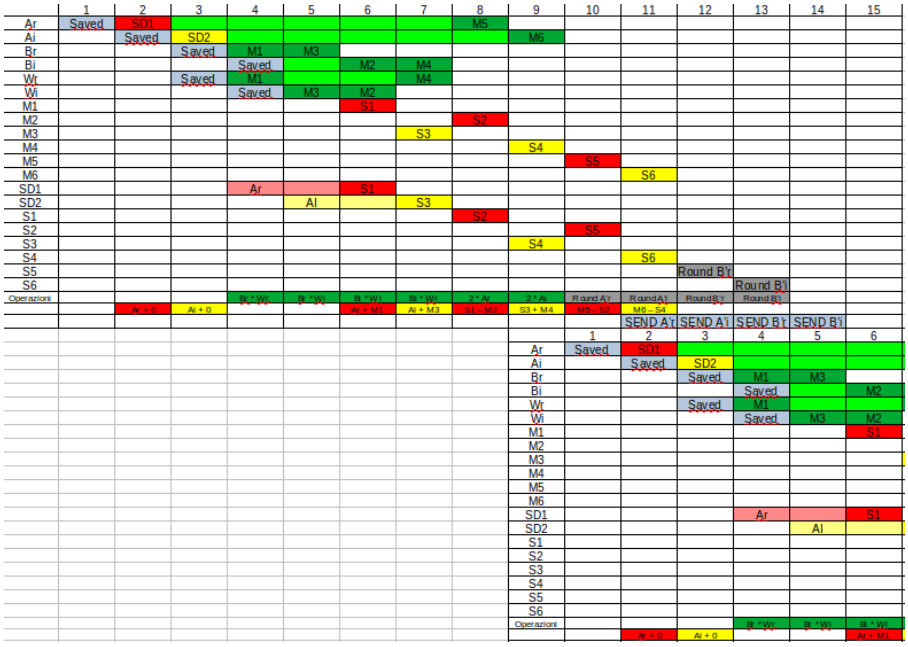
Nota: La soluzione di *figura 1.8b* è quella con minor *Throughput* causato dal diverso collo di bottiglia introdotto dai sommatori usati come buffer, e quindi non può iniziare un nuovo calcolo finché le unità sommatore sono libere.

Le soluzioni più interessanti risultano essere quelle con ordine $[A_R, A_I, B_R, B_I]$, $[B_R, B_I, A_R, A_I]$ e con doppio ingresso a coppia $[A_R, A_I]$ $[B_R, B_I]$ (tutte con BUS dedicato per il feed di $S1$ ed $S3$).

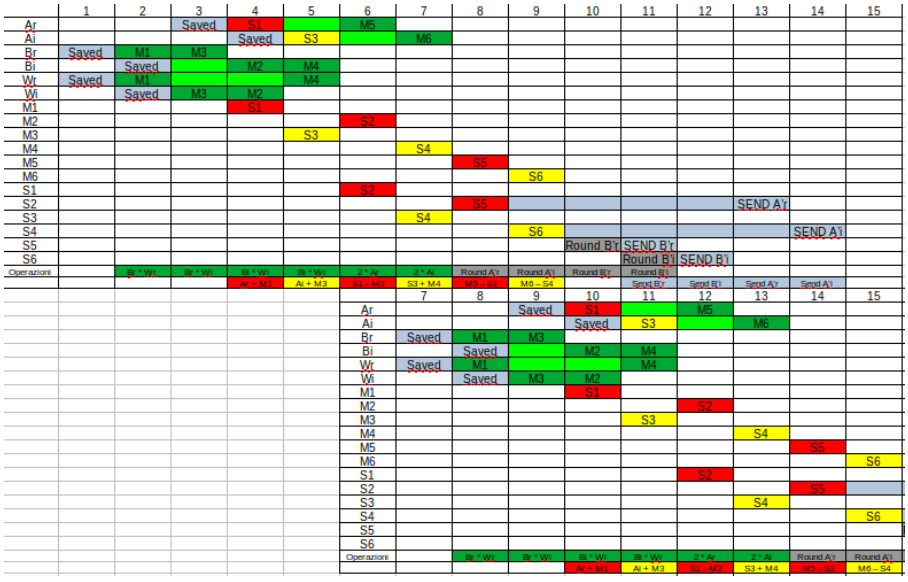
Nota: Riordinare i dati in ingresso è la soluzione più semplice dato che si possono salvare i dati nell'ordine ricevuto e utilizzarli nell'ordine corretto, mentre per poter riordinare i dati in uscita occorre introdurre degli step algoritmici dove salvare i risultati in dei buffer temporanei. Il costo del primo riordino è in termini di step algoritmici (*figura 1.8a*), mentre per il secondo tipo di riordino vi introduce inoltre un costo maggiore dal punto di vista dell'*Hardware* legato all'unità di riordino necessaria (*figura 1.8c*).



(a) Ordine $[A_R, A_I, B_R, B_I]$ con BUS dedicato al feed di S1 ed S3

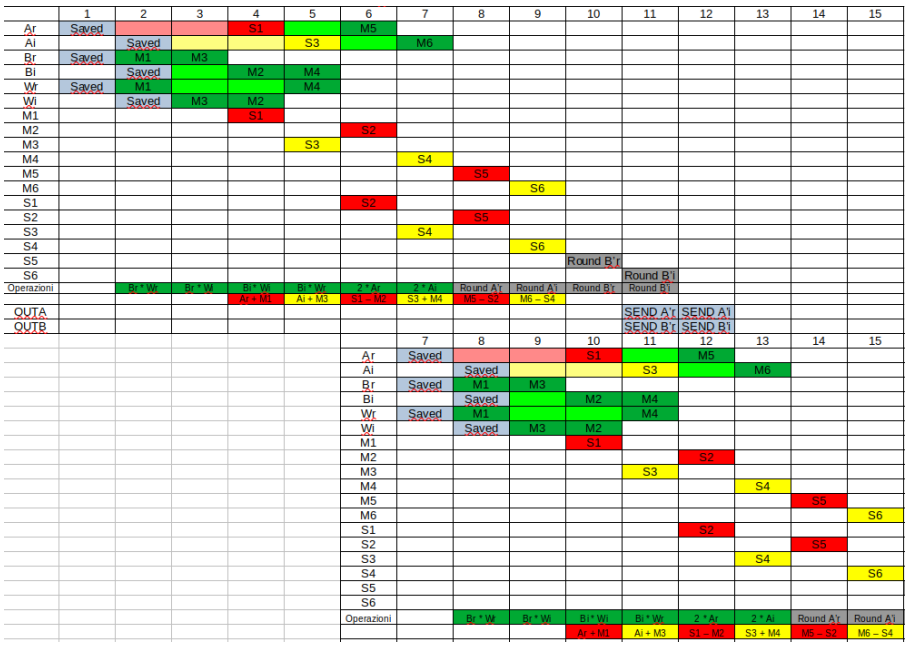


(b) Ordine $[A_R, A_I, B_R, B_I]$ con riuso dei registri in feedback ai sommatori

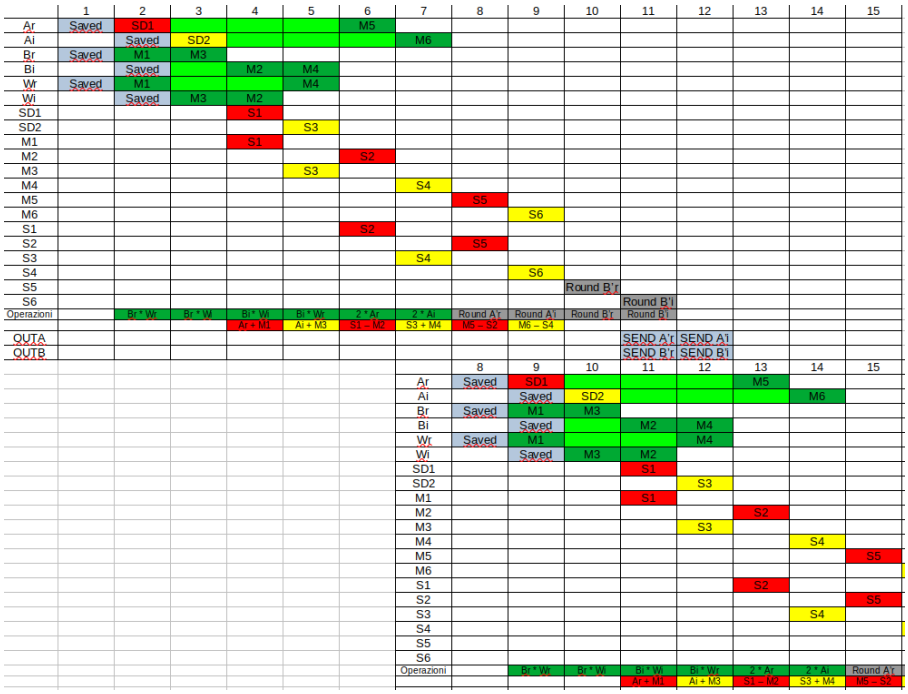


(c) Ordine $[B_R, B_I, A_R, A_I]$

Figure 1.8: Proseguimento



(a) Ordine a coppia $[A_R, B_R][A_I, B_I]$ con BUS dedicato al feed di S1 ed S3



(b) Ordine a coppia $[A_R, B_R][A_I, B_I]$ con riuso dei registri temporanei in feedback ai sommatori

Figure 1.9: Diagrammi del tempo di vita delle variabili ibrido, in modalità continua

L'interfacciamento dell'unità verso l'esterno, considerando la natura dell'algoritmo, potrebbe essere ridotto all'invio seriale dei coefficienti e seriale dei dati, alternando in entrambi i casi parte reale a parte immaginaria. Tale organizzazione si tradurrebbe in due porte di ingresso, una per coefficienti e una per dati, che lavorerebbero con dati ricevuti serialmente e li invierebbero sempre serialmente ma su un'unica porta di output. Questa organizzazione permetterebbe una riduzione del costo dato che il numero di porte è stato ridotto.
Accorpare la porta dati con la porta coefficienti permetterebbe di ridurre ulteriormente il costo, ma comporterebbe una riduzione delle prestazioni a causa del maggior numero di step algoritmici dovuto allo scheduling dei dati letti.

Nota: Questa configurazione di porte comporterebbe inoltre un livello di dispatch tra gli stage nel caso si voglia implementare una FFT classica.

Nota: Le configurazioni a porta singola e quindi a dati seriali in input e output sono implementabili, ma non soddisfano il principio di interfacciabilità tra *Processing element* dello stesso tipo nel caso di una FFT: alcune butterfly predono come ingressi la coppia di due dati $[B_R B_I]$ provenienti da unità diverse, e quindi temporalmente incompatibili, per questo servirebbe il livello di dispatch, il quale incrementa il costo e la latenza complessiva.
Pertanto, sebbene abbia una porta di ingresso e di uscita in più, la soluzione che soddisfa l'interfacciabilità considerando tutte le combinazioni di ingresso possibile, è quella a coppia di ingressi $[A_R, B_R][A_I, B_I]$. Questa soluzione ha un totale di 5 porte: una porta per coefficienti, una per dati A e una per dati B in ingresso, una per dati A e una per dati B in uscita.

La modalità di esecuzione continua si distingue dall'esecuzione isolata per il fatto di controllare parte di una esecuzione e di un'altra in uno stesso step algoritmico. Per passare da esecuzione isolata a continua si deve inviare il segnale di *START* allo step algoritmico adatto, durante il quale vi è attenzione da parte della *Control Unit* su tale segnale. Tale condizione va ripetuta allo stesso step per mantenere l'esecuzione continua. Nel caso non si invii tale segnale durante la condizione continua, l'esecuzione ritorna a singola fino alla fine del calcolo in corso. Alla fine, il circuito ritorna ad uno stato di attesa (*IDLE*).

1.2.6 Analisi e ottimizzazione dei BUS

Per quanto riguarda i *BUS* impiegati, si può usare come punto di partenza il numero massimo di operandi utilizzati nel caso peggiore. Infatti utilizzarne in numero maggiore non porta alcun beneficio. Il numero di *BUS globali* che si occupano di distribuire i dati, salvati nella struttura di memoria discussa ne *sezione 1.3*, alle rispettive unità di calcolo è diverso per le soluzioni proposte finora:

- La soluzione in *figura 1.9a* ha 3 operandi letti al ciclo 4 e 5
- La soluzione in *figura 1.9b* ha 2 operandi letti dal ciclo 2 al ciclo 7

Nota: La soluzione in *figura 1.9b* è stata progettata con l'obiettivo di ridurre il numero di *BUS* dedicati.

I due *BUS* globali comuni a tutte le soluzioni sono il *DATA_BUS* e *COEFF_BUS*, rispettivamente il *BUS* dei dati e dei coefficienti. Il terzo bus aggiuntivo per la soluzione in *figura 1.9a*, ovvero il *DATA2_BUS*, è utilizzato per fornire la parte reale e immaginaria dell'operando A ai sommatori, dato che *DATA_BUS* e *COEFF_BUS* sono impegnati nel trasporto degli operandi coinvolti nella moltiplicazione.
La soluzione con *BUS* dedicato aggiuntivo, sebbene comporti un costo maggiore in termini di *Area*, si traduce in un minor sforzo dei driver di *BUS*, dato che il numero di porte di ingresso che vi si affacciano è minore rispetto alla soluzione con due soli *BUS* (*DATA_BUS* e *COEFF_BUS* sono collegati al solo moltiplicatore e *DATA2_BUS* a due sommatori, contro il *DATA_BUS* collegato a moltiplicatore e due sommatori nel caso siano usati entrambi per località dei dati). Ciò potrebbe comportare minori dimensioni complessive (valutabile dopo sintesi).

1.2.7 Analisi e ottimizzazioni delle Interconnessioni Locali

Per quanto riguarda le *interconnessioni locali*, accade che la loro complessità e connettività dipende fortemente dal tipo di organizzazione della computazione: finora è stato ipotizzato di mantenere separate le esecuzioni dell'albero della parte reale da quello della parte immaginaria. Tale imposizione permette di ottenere la connessione mostrata in *figura 1.10*.

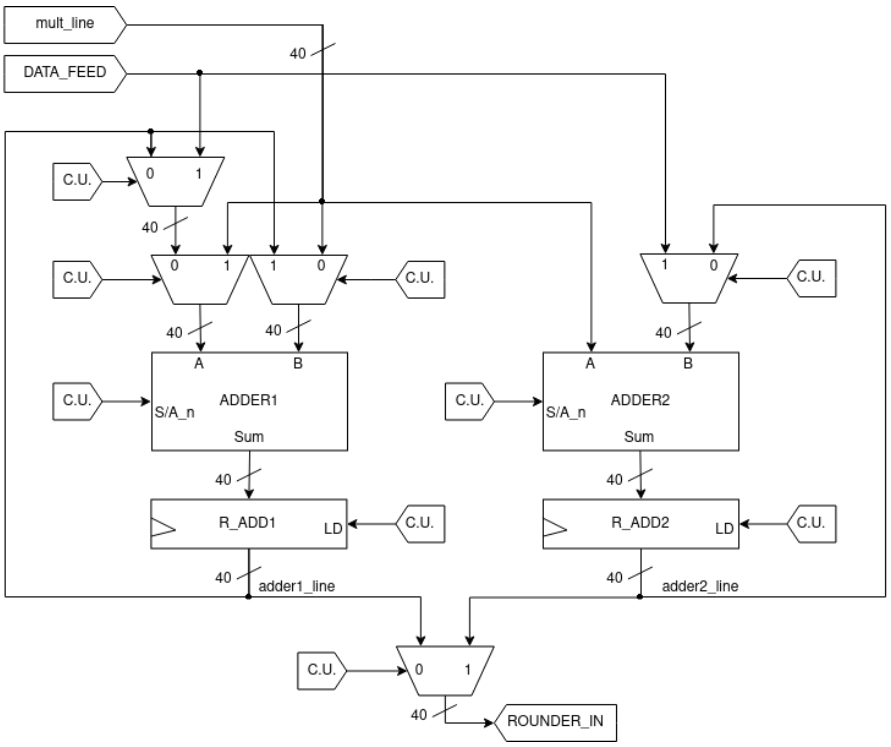


Figure 1.10: Datapath sul particolare delle interconnessioni interne con alberi separati

La linea di interconnessione *mult_line* è legata al risultato bufferizzato del moltiplicatore, la linea *adder1_line* è legata al risultato bufferizzato del primo sommatore e la linea *adder2_line* è legata al risultato bufferizzato del secondo sommatore. Tali linee sono usate per collegare tra loro le unità di calcolo. Facendo decadere l'ipotesi di separazione degli alberi di calcolo si ottiene il circuito di *figura 1.11*

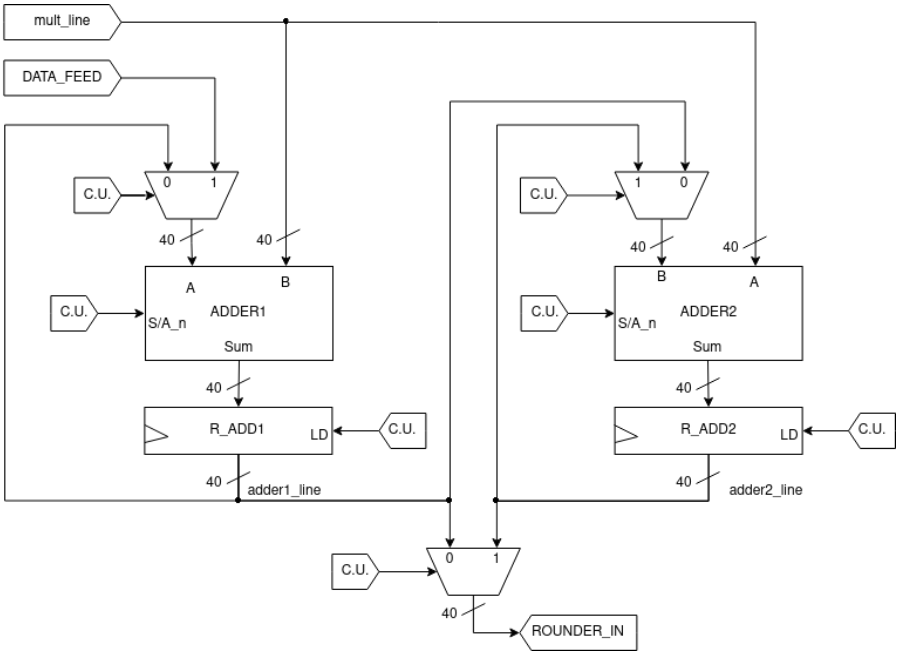


Figure 1.11: Datapath sul particolare delle interconnessioni interne meno congestionate

Nota: la linea *DATA_FEED* indica la linea che porta i coefficienti A_R e A_I, e quindi corrisponde a *DATA_BUS* nel caso di soluzione con riuso dei registri, mentre corrisponde a *DATA2_BUS* nel caso di soluzione a *BUS* dedicato.

Le connessioni dei sommatori sono state derivate considerando la non commutatività della sottrazione e l'ordine degli operandi rispetto al risultato della moltiplicazione.

Le interconnessioni mostrate in *figura 1.10* hanno come obiettivo la separazione degli alberi di calcolo per ottenere dal *sommatore 1* i risultati delle parti reali, mentre dal *sommatore 2* i risultati delle parti immaginarie. Tale struttura porta ad una maggiore congestione a livello degli ingressi dei sommatori, dato che le due sottrazioni S_2 ed $_4$ del *sommatore 1* necessitano di commutazione degli operandi, servendosi per tale scopo di due multiplexer aggiuntivi.
Le interconnessioni mostrate in *figura 1.11* hanno come obiettivo l'assegnazione alle stesse porte degli operandi in ingresso ad entrambi i sommatori, cercando di ridurre la congestione in quelle aree. Per ottenere ciò, al *sommatore 1* sono allocate le operazioni S_1, S_2 ed S_3, mentre S_4, S_5 ed S_6 al *sommatore 2*. Tale soluzione utilizza solamente due multiplexer, ma non mantiene separate le esecuzioni, dato che S_4 ed S_5 dipendono da S_3 ed S_2.

Scelta: Sebbene utilizzare due sommatori e dedicarne ai calcoli uno al ramo destro e sinistro, non sia la scelta più adatta in termini di implementazione, dato che utilizza più risorse *Hardware*, è la scelta optata per il suo valore didattico: maggior uso di segnali di controllo, maggior località dei dati e quindi rintracciabilità degli errori sono i punti di forza per cui tale scelta è stata adottata.

Extra: come sarebbe l'implementazione con un solo sommatore? Comprimendo ancora di più l'esecuzione, in una sorta di tecnica che ricorda il *Pipeline Interleaving* sfruttando il livello di pipe interno del sommatore, è possibile separare le esecuzioni assegnandole nell'ordine $[S_1, S_3, S_2, S_4, S_5, S_6]$. Per quanto riguarda l'*Hardware* utilizzato risulta piuttosto interessante, dato che è esiguo. Tale soluzione inoltre non comporta penalità di latenza o ridotta frequenza di ripetizione di ciclo. Il circuito che ne risulterebbe è mostrato in *figura 1.12*.

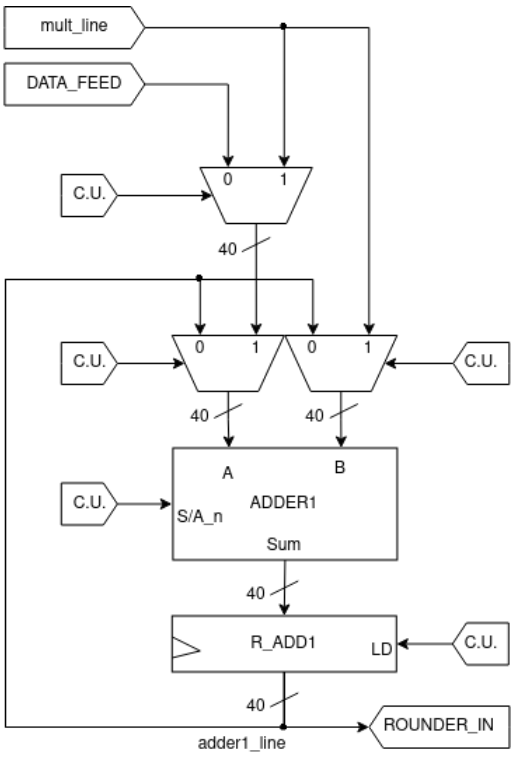


Figure 1.12: Datapath sul particolare delle interconnessioni per implementazione a singolo sommatore

1.2.8 Analisi del parallelismo interno

Il parallelismo interno della macchina è stato valutato considerando la natura frazionaria del formato *Fixed Point*: il moltiplicatore a interi fornisce un risultato con parallelismo pari alla somma dei parallelismi dei dati in ingresso. Perciò il risultato è rappresentato su *40 bit* avendo gli operandi rappresentati su *20 bit*.
In realtà, essendo la rappresentazione *Fixed Point* accade che il risultato ha due *MSB* di segno, e quindi di fatto il risultato sarebbe rappresentato efficacemente su *39 bit*. La scelta di mantenere *40 bit* di rappresentazione discende dal fatto che le due operazioni di somma/sottrazione successive possono generare un overflow, ma non

si sa durante quale delle due. Avere doppio *MSB* è utile per poter rintracciare l'avvenuto overflow e mantenere così il segno originario del dato. Perciò *40 bit* sono sufficienti per mappare i dati internamente. Inoltre permette di avere gli scalamenti con allineamento in modo automatico, sia per quanto riguarda il prodotto per 2 che durante l'arrotondamento.

1.2.9 Analisi di arrotondamento e scalamento

Per quanto riguarda l'unità di arrotondamento e scalamento, tale unità deve applicare l'arrotondamento *Round to nearest even* e scalare il risultato, rappresentato su *40 bit*, in modo da poterlo riportare su *20 bit*, ovvero il parallelismo di interfacciamento.
Tale unità si basa sulla soluzione con *Sticky bit*, la quale comprende un sommatore, il quale prende come operatori il numero da arrotondare, da cui vengono estratti i primi 20 bit a partire dall'*MSB*, a cui viene sommato il risultato della combinazione dei *trailing zeros* con l'*LSB* del sottoinsieme del risultato.

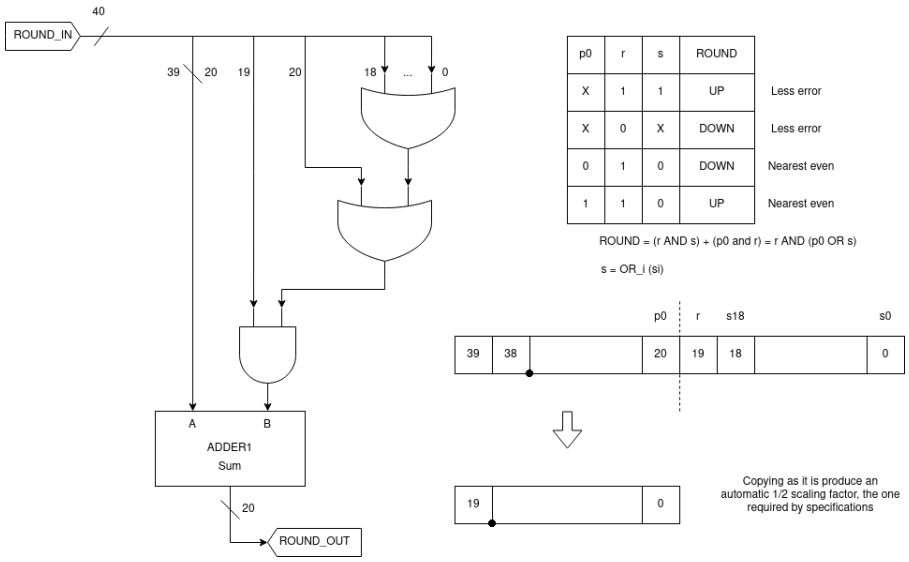


Figure 1.13: Implementazione del circuito di arrotondamento e troncamento

Nota: prendere il numero da arrotondare in questo modo, ignorando il doppio bit di segno, significa avere anche lo scalamento di un bit implicito in modo automatico una volta che viene copiata così com'è la parte dei primi 20 bit a partire dall'*MSB* del risultato arrotondato, condizione richiesta per evitare l'overflow delle unità successive.

1.3 Resoconto delle soluzioni e scelta del Datapath

Le soluzioni finora analizzate portano a diverse, seppur simili, implementazioni dal punto di vista circuitale. Tutte le soluzioni sono basate su alberi di calcolo separati, per questioni le questioni citate alla *sezione 1.2.7*.
La struttura di memoria ipotizzata per i dati in ingresso, per tutte le soluzioni, è la coppia di *Register Files* uno dedicato ai dati e l'altro ai coefficienti. Le soluzioni mostrate sono 3, dove la prima, non accettabile è mostrata solo per questioni didattiche.

1.3.1 Soluzione 1

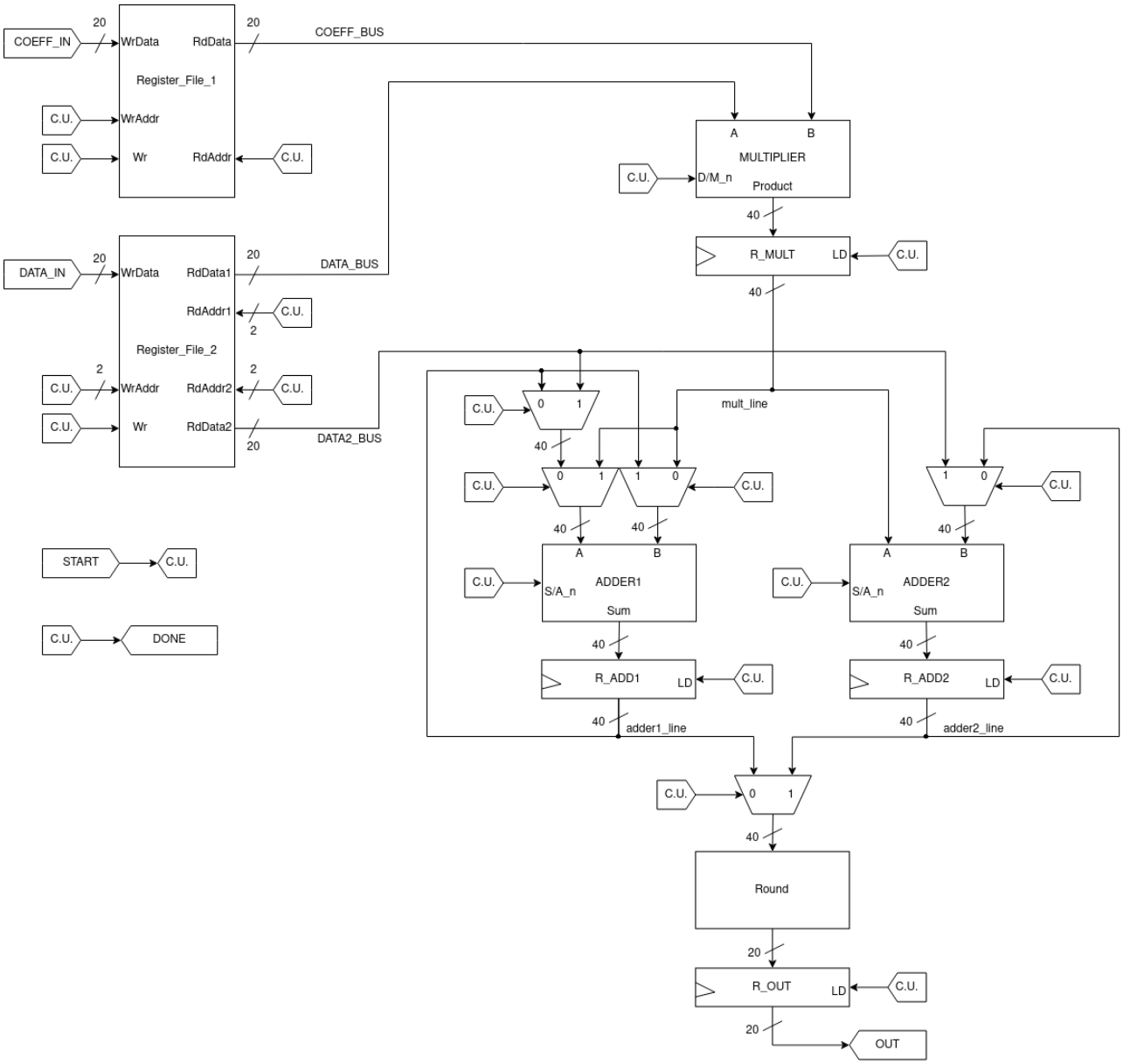


Figure 1.14: Datapath della *Soluzione 1*

La *Soluzione 1* è quella con ordine di ingresso e uscita $[A_R, A_I, B_R, B_I]$ e *BUS* aggiuntivo. Necessita di un register file per i dati con una porta di scrittura e due porte di lettura, di cui una si aggancia al *BUS* aggiuntivo per fornire i dati A_R e A_I al sommatore per eseguire la prima somma, selezionati usando un multiplexer.
L'ordine dei dati viene gestito all'ingresso grazie allo scheduling a livello di *Control Unit*. Ha un *BUS* in più, ma ciò permette di avere una frequenza di ripetizione di ciclo pari a 1/7 e latenza di 14 cicli.

1.3.2 Soluzione 2

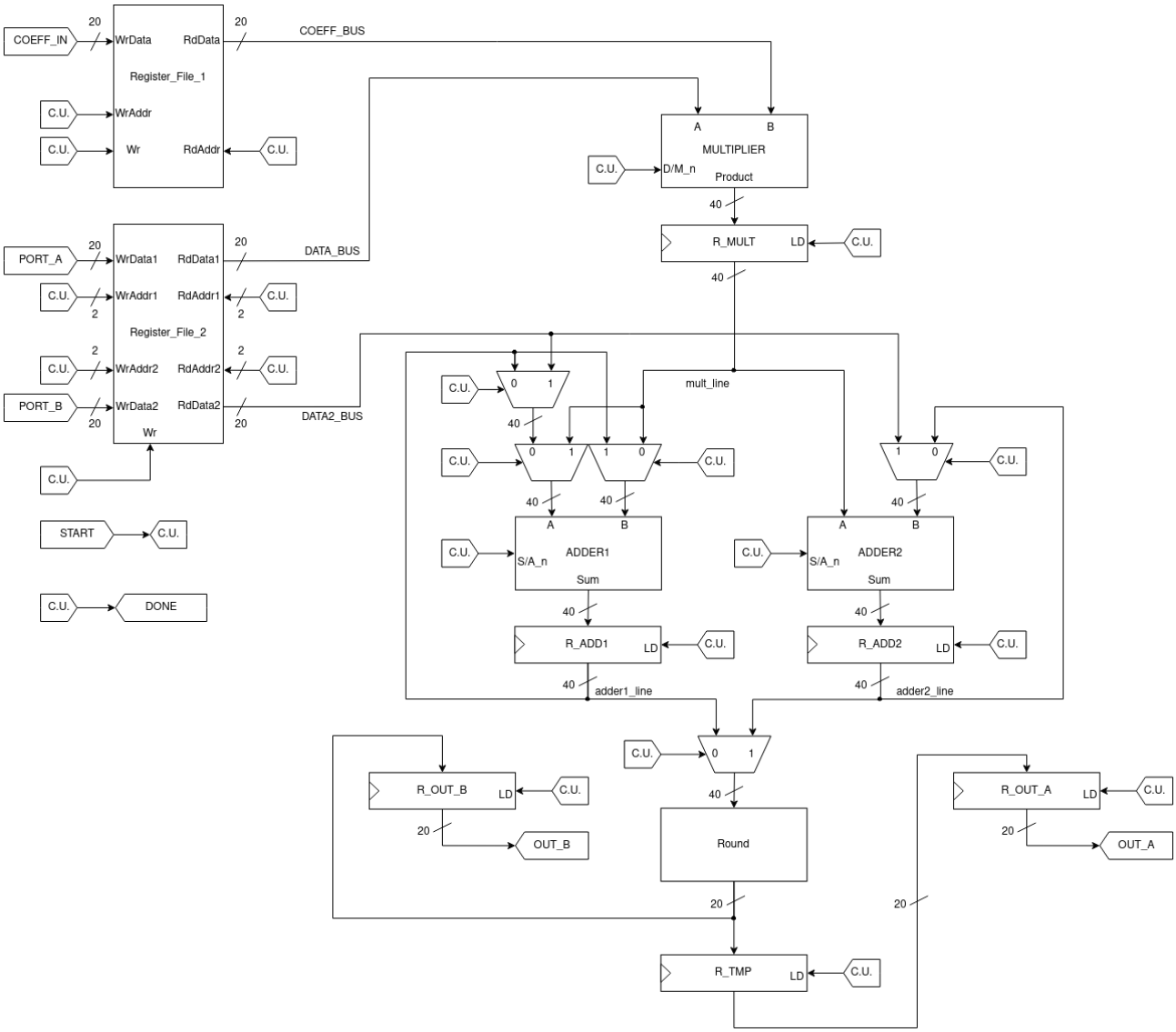


Figure 1.15: Datapath della *Soluzione 2*

La *Soluzione 2* è quella con ingresso a coppia $[A_R, B_R][A_I, B_I]$ e *BUS* aggiuntivo. Necessita di un register file per i dati con due porte di scrittura e due di lettura. Inoltre presenta due registri in più, uno dovuto alluscita aggiuntiva, e uno temporaneo per il riordino dei coefficienti A_R e A_I. Il costo aggiuntivo in *Hardware* viene ripagato dalla maggior frequenza di ripetizione di un calcolo pari a 1/6 e latenza di 12 cicli (il Late status viene trattato accorpando gli stati di *IDLE* e *LD_R* come mostrato alla sezione 1.4).

1.3.3 Soluzione 3

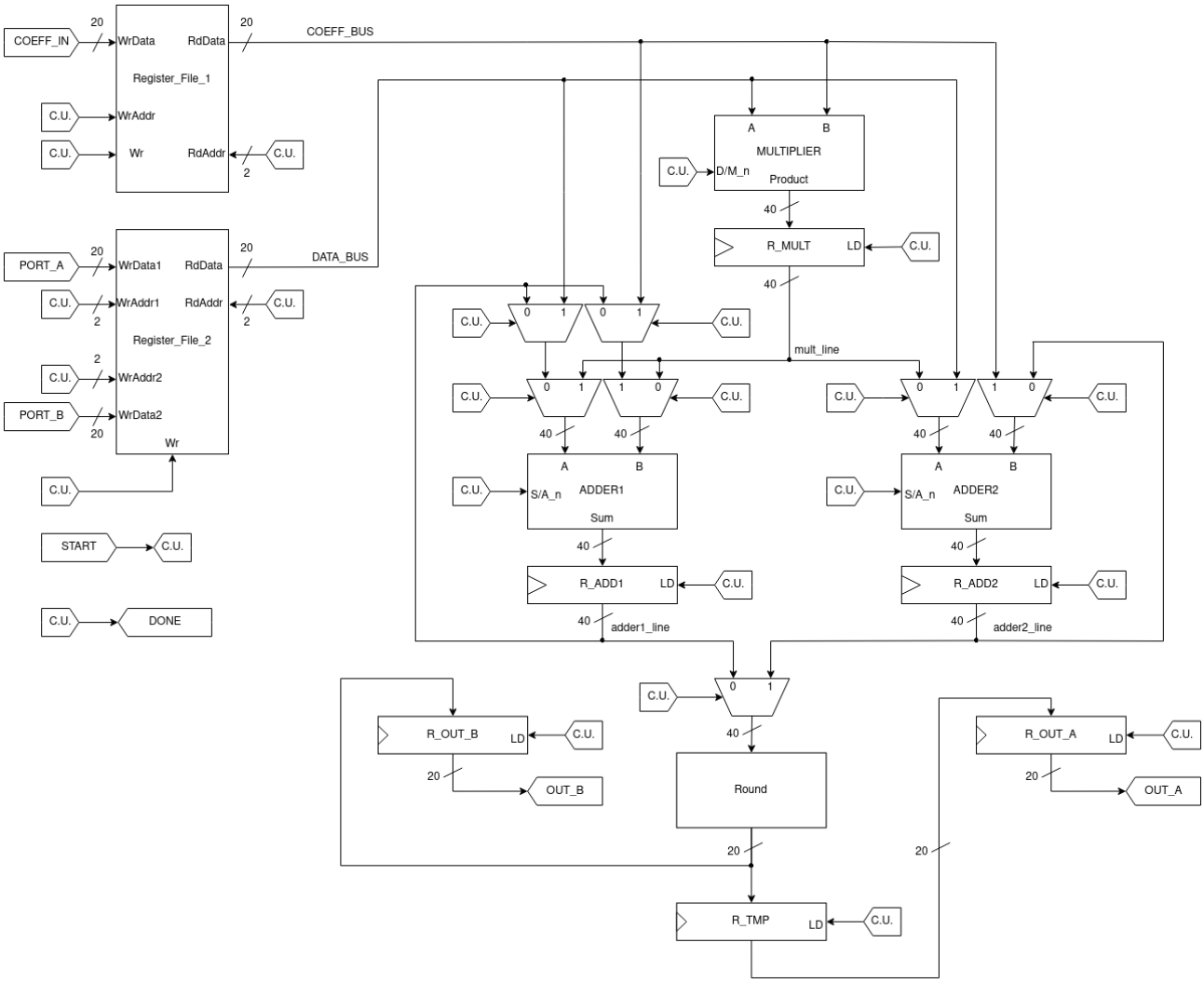


Figure 1.16: Datapath della *Soluzione 3*

La *Soluzione 3* è quella con ingresso a coppia $[A_R, B_R][A_I, B_I]$ e riuso dei registri temporanei dei sommatori. Necessita di un register file per i dati con due porte di scrittura e una di lettura. Ha un *BUS* in meno, ma più multiplexers. Nonostante ciò, raggiunge una frequenza di ripetizione di un calcolo di $1/7$ e latenza di $12 cicli$ (stessa questione trattata alla soluzione precedezzz).

Ciò che cambia per le diverse implementazioni è il numero di porte del *Register File* dedicato ai dati, il quale presenta due porte di uscita nei casi di *DATA2_BUS* dedicato.

1.3.4 Scelta e ultime considerazioni

Tra quelle mostrate, la scelta adottata è quella presentata come *Soluzione 2* (1.3.2). Questa è stata scelta poiché oltre ad essere perfettamente interfacciabile, senza necessità di riordino di dati/dispatch esterni, presenta minor latenza e il più elevato *Throughput* pari a 1/7 (rispetto alla *Soluzione 3*). Una volta terminato un calcolo, il risultato è inviato alle porte di uscita, per un totale di 2 cicli nell'ordine $[A_R, B_R][A_I, B_I]$, dove il segnale di *DONE* accompagna il primo ciclo.
I dati che si presentano all'uscita hanno un fattore di scalamento di 2^5, se si considera il valore originario dei dati di ingresso compresi tra -1 e +1 (estremi esclusi), perché i dati hanno subito 5 scalamenti per evitare overflows. Quindi il risultato andrebbe moltiplicato per 32. Se invece i dati originari non necessitavano di scalamento, il fattore moltiplicativo è 16.

Nota: la scelta di avere due *Register Files* separati, in realtà non è ottimale dal punto di vista della sintesi. Avere maggiore omogeneità di struttura, e quindi un circuito più regolare/modulare, permette di sfruttare al meglio l'area occupata, e potenzialmente avere anche migliori prestazioni. La scelta di separarli è puramente didattica e volta a velocizzare lo sviluppo del circuito (facilitandone il debug).

1.4 Derivazione della Control Unit

La *Control Unit* da implementare risulta del tipo microprogrammato a indirizzamento implicito e avente supporto al *Late Status*. Il circuito di tale unità si ispira al circuito base delle control unit di questo tipo, con gestione della *control-word*, mostrato in *figura 1.17*.

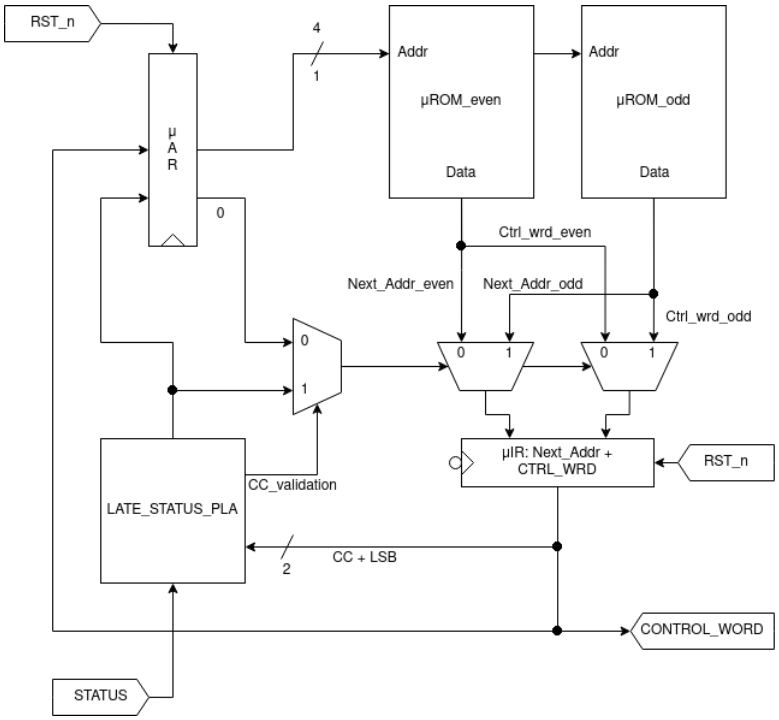


Figure 1.17: Circuito della *Control Unit*

L'idea alla base consiste in una memoria *ROM*, le cui locazioni contengono l'indirizzo della prossima istruzione e l'insieme dei segnali di controllo del *Datapath* per l'istruzione attuale. Dovendo separare la memoria in due, ovvero una per gli indirizzi pari e una per gli indirizzi dispari, per poter supportare il *Late Status*, la scelta più naturale è assegnare le locazioni in base ad un salto: qualora il segnale di avvio *START* si trovi a 1 logico durante uno stato che supporta il salto, questo fa puntare il prossimo indirizzo a quello della *micro-ROM* dispari, mentre se si trova a 0 logico causerà il salto alla *micro-ROM* pari.
L'istruzione di cui fa *fetch* il *micro-instruction register* dipende dalla condizione di salto verificata con il *Late Status PLA*, un circuito combinatorio in grado di stabilire le condizioni di salto che arrivano in ritardo. Questo circuito, essendo semplice, ha una condizione di salto che dipende solo dallo stato attuale e dal segnale di avvio

START. Se fosse più complesso, il segnale *STATUS* conterrebbe anche bit di condizione provenienti dal *Data-path*, come ad esempio Flags di calcolo.
Il segnale che agisce da *Enable* o sensibilizzatore rispetto al segnale di *START* è il bit *CC* contenuto nel *micro-instruction register*. Tale bit, per comodità, abilita alle condizioni di salto quando è asserito (valore attivo alto a 1 logico). Tale imposizione fa comodo nella fase di reset iniziale: cancellare interamente il *micro-instruction register* produce *Next Address*, *Control Word* e *CC* pari a zero. Ponendo lo stato di *IDLE* all'indizzo zero soddisfa la corrispondenza dello stato di IDLE successivo ad un reset.
Lo stato di *IDLE* però risulta sensibile al segnale di *START*, perciò necessiterebbe di *CC* pari a 1. Questo problema si risolve quando il *micro-instruction register* fa una fetch immediatamente dopo un reset, dato che il *Next Address* è quello di un'*IDLE*, e carica conseguentemente il *CC* corretto. Questo produce però una latenza di 1 ciclo di clock da quando avviene un reset a quando la macchina può effettivamente essere sensibile allo *START*. La derivazione dei segnali è fatta sfruttando il diagramma ibrido del tempo di vita delle variabili di *figura 1.7a*, elencando il contenuto di ogni bit durante ogni stato.
Il *micro-address register* contiene l'indirizzo attuale a cui corrisponde l'indirizzo della locazione di memoria a cui si fa accesso alla *micro-ROM* per ottenere lo stato successivo, analogo al *Program Counter* nei microprocessori.

Nota: La *Status PLA* ha come ingresso anche l'*LSB* del *micro-instruction register* poiché se un salto non avviene o si è in modalità seuenziale, questo punta alla locazione sucessiva rispetto a quella di provenienza.

1.4.1 Derivazione stati e segnali di controllo

Il numero totale di stati, considerando le esecuzioni isolate e in modalità continua, pari a 19 (Idle, 12 in esecuzione singola e 6 in esecuzione continua). Tali stati sono organizzati nel modo mostrato in *figura 1.18*.

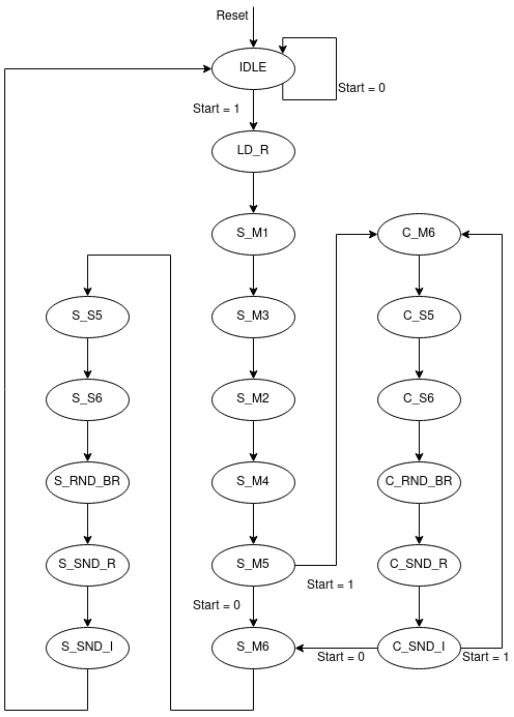


Figure 1.18: Diagramma a stati della macchina

Gli stati nominati nel diagramma sono organizzati nel modo riportato in *tabella 1.4.1*.

Stato	Descrizione
IDLE	Stato di inattività, in attesa di un evento per partire
LD_R	Stato in cui A_R, B_R vengono caricati nel *Register file* per i dati e W_R nel *Register file* per i coefficienti, in esecuzione singola
S_M1	Stato in cui A_I, B_I vengono caricati nei *Register file* per i dati, W_I nel *Register file* per i coefficienti e viene allocata M_1, in esecuzione singola
S_M3	Stato in cui viene allocata M_3, in esecuzione singola
S_M2	Stato in cui vengono allocate M_2 ed S_1, in esecuzione singola
S_M4	Stato in cui vengono allocate M_4 ed S_3, in esecuzione singola
S_M5	Stato in cui vengono allocate M_5 ed S_2, in esecuzione singola
C_M6	Stato in cui vengono allocate M_6 ed S_4 dell'esecuzione corrente e carica A_R, B_R e W_R della nuova esecuzione, in modalità continua
C_S5	Stato in cui vengono allocate S_5 e l'arrotondamento di A'_R dell'esecuzione corrente, alloca M_1 carica A_I B_I e W_I della nuova esecuzione, in modalità continua
C_S6	Stato in cui vengono allocate S_6, l'arrotondamento di A'_I e il salvataggio di A'_R nel registro temporaneo di riordino dell'esecuzione corrente e alloca M_3 della nuova esecuzione, in modalità continua
C_RND_BR	Stato in cui vengono allocate l'arrotondamento di B'_R, A'_I messo nel registro di riordino, A'_R portato al registro dell'uscita OUT_A dell'esecuzione corrente e alloca M_2 ed S_1 della nuova esecuzione, in modalità continua
C_SND_R	Stato in cui vengono allocate l'arrotondamento di B'_I e l'invio di B'_R al registro di OUT_B e l'asserimento di $DONE$ dell'esecuzione corrente e alloca M_4 ed S_3 della nuova esecuzione, in modalità continua
C_SND_I	Stato in cui vengono allocate l'invio di A'_I e B'_I dell'esecuzione corrente e alloca M_5 ed S_2 della nuova esecuzione, in modalità continua
S_M6	Stato in cui vengono allocate M_6 ed S_4, in esecuzione singola
S_S5	Stato in cui vengono allocate S_5 e l'arrotondamento di A'_R, in esecuzione singola
S_S6	Stato in cui vengono allocate S_6, l'arrotondamento di A'_I e l'invio di A'_R al registro di riordino, in esecuzione singola
S_RND_BR	Stato in cui vengono allocate l'arrotondamento di B'_R e l'invio di A'_I al registro di riordino, A'_R portato al registro di OUT_A, in esecuzione singola
S_SND_R	Stato in cui vengono allocate l'arrotondamento di B'_I e l'invio di A'_R (già su OUT_A) e B'_R (su OUT_B), in esecuzione singola
S_SND_I	Stato in cui viene allocato l'invio di A'_I e B'_I, in esecuzione singola

Table 1.1: Nomenclatura e descrizione degli stati in termini di operazioni allocate

1.4.2 Derivazione del sequencer

Per ottenere la mappatura degli stati in memoria, si parte dal diagramma degli stati e si impone la parità della memoria come corrispondenza del valore del segnale di *START* durante una condizione di salto. In questo caso, se *START* è a 1 logico durante uno stato di attenzione, la *Control Unit* farà le successive fetch dalla memoria dispari, altrimenti dalla pari, e così per ogni stato che supporta un salto. Il diagramma risultate è mostrato in *figura 1.19*.

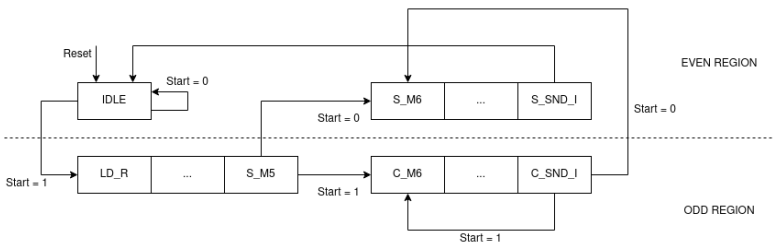


Figure 1.19: Diagramma a stati della macchina

Le locazioni prive di salto vengono duplicate come misura di sicurezza. Gli stati in sequenza, ma appartenenti alle due diverse modalità di funzionamento, sono stati messi allo stesso indirizzo sfruttando la parità delle memorie per avere entrambe le *micro-ROM* con dimensioni ridotte.

1.4.3 Derivazione del command generator

Per quanto riguarda la parte di command generator, questa parte può essere realizzata basandosi sulla *programmazione orizzontale* o *programmazione verticale*.
La prima fornisce un maggior controllo dato che nella *micro-ROM* oltre al prossimo indirizzo è presente anche la *control-word* attuale, mentre la seconda permette di mantere ridotte le dimensioni della *micro-ROM* generando la *control-word* con una *PLA* dedicata prendendo come ingresso il prossimo indirizzo.
La scelta che permette di risparmiare il tempo di accesso alla *PLA* di generazione dei comandi, è quella della programmazione orizzontale, ove le *control-word* sono immagazzinate all'interno delle *micro-ROM* stesse: queste, ad ogni locazione, conterranno perciò l'indirizzo successivo e la command-word per l'istruzione attuale. Entrambe sono selezionate a valle delle *micro-ROM* con un multiplexer basato sull'*LSB* ottenuto dalle valutazioni di *Late Status*.
Tali mappature sono riportate di seguito, ne *figura 1.20*, *figura 1.21*, *figura 1.22*, *figura 1.23* e *figura 1.24*.

Nota: Tali immagini sono estratte dai fogli elettronici usati poi dai rispettivi script per automatizzare il riempimento e finalizzazione delle *micro-ROM*.

STATE	CC	ADDR3	ADDR2	ADDR1	ADDR0	ODD_EVEN_N
IDLE_LD_R	-	0	0	0	0	-
S_M1	-	0	0	0	1	-
S_M3	-	0	0	1	0	-
S_M2	-	0	0	1	1	-
S_M4	-	0	1	0	0	-
S_M5	-	0	1	0	1	-
X_M6	-	0	1	1	0	-
X_S5	-	0	1	1	1	-
X_S6	-	1	0	0	0	-
X_RND_BR	-	1	0	0	1	-
X_SND_R	-	1	0	1	0	-
S_SND_I	-	1	0	1	1	-
C_SND_I	-	1	1	0	0	-
IDLE	-	0	0	0	0	-
S_M6	-	0	1	1	0	-
S_S5	-	0	1	1	1	-
S_S6	-	1	0	0	0	-
S_RND_BR	-	1	0	0	1	-
S_SND_R	-	1	0	1	0	-
C_M6	-	0	1	1	0	-
C_S5	-	0	1	1	1	-
C_S6	-	1	0	0	0	-
C_RND_BR	-	1	0	0	1	-
C_SND_R	-	1	0	1	0	-

Figure 1.20: Tag mnemonici usati per riferirsi agli indirizzi delle ROM

ROM_ADDR	CC	NXT_BASE_ADDR	PARITY	CTRL_WRD
IDLE_LD_R	'1'	IDLE	'0'	CW_IDLE
S_M1	'0'	S_M3	'0'	CW_SM1
S_M3	'0'	S_M2	'0'	CW_SM3
S_M2	'0'	S_M4	'0'	CW_SM2
S_M4	'0'	S_M5	'0'	CW_SM4
S_M5	'1'	S_M6	'0'	CW_SM5
X_M6	'0'	S_S5	'0'	CW_SM6
X_S5	'0'	S_S6	'0'	CW_SS5
X_S6	'0'	S_RND_BR	'0'	CW_SS6
X_RND_BR	'0'	S_SND_R	'0'	CW_SRND_BR
X_SND_R	'0'	S_SND_I	'0'	CW_SSND_R
S_SND_I	'0'	IDLE	'0'	CW_SSND_I
C_SND_I	'1'	S_M6	'0'	CW_CSND_I

Figure 1.21: Contenuto, espresso in tag, della ROM pari

ROM_ADDR	CC	NXT_BASE_ADDR	PARITY	CTRL_WRD
IDLE_LD_R	'0'	S_M1	'1'	CW_SLD_R
S_M1	'0'	S_M3	'1'	CW_SM1
S_M3	'0'	S_M2	'1'	CW_SM3
S_M2	'0'	S_M4	'1'	CW_SM2
S_M4	'0'	S_M5	'1'	CW_SM4
S_M5	'1'	C_M6	'1'	CW_SM5
X_M6	'0'	C_S5	'1'	CW_CM6
X_S5	'0'	C_S6	'1'	CW_CS5
X_S6	'0'	C_RND_BR	'1'	CW_CS6
X_RND_BR	'0'	C_SND_R	'1'	CW_CRND_BR
X_SND_R	'0'	C_SND_I	'1'	CW_CSND_R
S_SND_I	'0'	IDLE	'1'	CW_SSND_I
C_SND_I	'1'	C_M6	'1'	CW_CSND_I

Figure 1.22: Contenuto, espresso in tag, della ROM dispari

sig/stato	RegFile1_Wr_Addr	RegFile1_Wr	RegFile1_Rd_Addr	RegFile2_Wr1_Addr0	RegFile2_Wr1_Addr1	RegFile2_Wr2_Addr0	RegFile2_Wr2_Addr1	RegFile2_Wr1	RegFile2_Wr2	RegFile2_Rd1_Addr0	RegFile2_Rd1_Addr1	RegFile2_Rd2_Addr0	RegFile2_Rd2_Addr1
CW_IDLE	0	0	0	0	0	0	0	0	0	0	0	0	0
CW_SLD_R	0	1	0	0	0	0	1	1	1	0	0	0	0
CW_SM1	1	1	0	1	0	1	1	1	1	0	1	0	0
CW_SM3	0	0	1	0	0	0	0	0	0	0	1	0	0
CW_SM2	0	0	1	0	0	0	0	0	0	1	1	0	0
CW_SM4	0	0	0	0	0	0	0	0	0	1	1	1	0
CW_SM5	0	0	0	0	0	0	0	0	0	0	0	0	0
CW_SM6	0	0	0	0	0	0	0	0	0	1	0	0	0
CW_SS5	0	0	0	0	0	0	0	0	0	0	0	0	0
CW_SS6	0	0	0	0	0	0	0	0	0	0	0	0	0
CW_SRND_BR	0	0	0	0	0	0	0	0	0	0	0	0	0
CW_SSND_R	0	0	0	0	0	0	0	0	0	0	0	0	0
CW_SSND_I	0	0	0	0	0	0	0	0	0	0	0	0	0
CW_CM6	0	1	0	0	0	0	1	1	1	1	0	0	0
CW_CS5	1	1	0	1	0	1	1	1	1	0	1	0	0
CW_CS6	0	0	1	0	0	0	0	0	0	0	1	0	0
CW_CRND_BR	0	0	1	0	0	0	0	0	0	1	1	0	0
CW_CSND_R	0	0	0	0	0	0	0	0	0	1	1	1	0
CW_CSND_I	0	0	0	0	0	0	0	0	0	0	0	0	0
index	RFC_WR_ADDR	RFC_WR	RFC_RD_ADDR	RFD_WR1_ADDR0	RFD_WR1_ADDR1	RFD_WR2_ADDR0	RFD_WR2_ADDR1	RFD_WR1	RFD_WR2	RFD_RD1_ADDR0	RFD_RD1_ADDR1	RFD_RD2_ADDR0	RFD_RD2_ADDR1

Figure 1.23: Tag mnemonici usati per riferirsi alle *Control-Words*, prima parte

Multiplier_Dbl_Mult_n	RegMult_Load	Sel1_Data2_Add1_n	Sel2_mult_mux1_n	Sel3_Add1_mult_n	Adder1_Sub_Add_n	RegAdd1_Load	sel4_data2_add2_n	Adder2_Sub_Add_n	RegAdd2_Load	sel4_add2_add1_n	Reg_reord_load	Reg_out_A_load	Reg_out_B_load	DONE
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	1	0	0	0	0	0
1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	1	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	1	0	0	0	0	0
0	1	0	0	0	0	1	0	1	0	1	1	0	0	0
0	1	1	0	0	0	0	0	0	1	0	1	1	0	0
0	1	0	0	0	0	1	1	0	0	1	0	0	1	1
1	1	0	0	0	1	0	0	0	1	0	0	1	1	0
MULT_DOUBLE	R_MULT_LD	MUX1_SEL	MUX2_SEL	MUX3_SEL	ADD1_SUB_ADD	R_ADD1_LD	MUX4_SEL	ADD2_SUB_ADD	R_ADD2_LD	MUX5_SEL	REORD_LD	OUT_A_BUF_LD	OUT_B_BUF_LD	DONE_BIT

Figure 1.24: Tag mnemonici usati per riferirsi alle *Control-Words*, seconda parte

1.5 Simulazione

Per quanto riguarda la verifica del progetto, una campagna di simulazioni è stata effettuata durante ogni step critico. Tali simulazioni riguardavano le singole unità di calcolo interne alle *Butterfly* per testarne il comportamento, la *Control Unit* per verificarne il corretto sequenziamento, della singola *Butterfly* per confermarne l'effettivo funzionamento. Infine una dimostrazione applicativa è stata realizzata in una simulazione per il calcolo di una *FFT 16x16* con 32 unità *Butterfly* instanziate.
I risultati delle singole unità di calcolo sono stati osservati direttamente nel simulatore e non saranno riportati dato che il loro valore non è molto rilevante, lo sono invece i risultati delle simulazioni di *Control Unit*, *Butterfly* e *FFT 16x16*.

1.5.1 Simulazione Control Unit

I risultati sono stati confrontati in modo automatico con lo script *Python check_state_by_ctrl_wrd.py A.3.4*, il quale traduce il *dump* della *Control-Word* durante i vari cicli nella rispettiva etichetta mnemonica utilizzata durante il progetto (e trattata ne *Flusso di progetto* 3).

1.5.2 Simulazione Butterfly

L'unità *Butterfly* singola è stata testata per verificarne il corretto comportamento e correggerne gli eventuali *bugs*. Il risultato è stato controllato confrontandolo con quello prodotto dal programma in *C* a basso livello che ne replicasse l'effettvo comportamento. I risultati sono stati poi verificati con il comando *diff*, presente nei sistemi operativi GNU-Linux, per controllare che entrambi fossero identici. Al termine di 1000 simulazioni fatte su numeri casuali, il comportamento è stato identico, persino in caso di overflow.

1.5.3 Simulazione in FFT 16x16

L'unità *Butterfly* è stata inoltre testata in una batteria da 32 unità, disposte in modo da realizzare la *FFT 16x16*. Tale unità risultante è stata dotata di una coda di sincronizzazione dei coefficienti per ognuno dei 4 stadi, in modo da supportare coefficienti diversi: ha dunque 16 ingessi per i campioni e 8 per i coefficienti.
I vettori di test impiegati in questo caso sono delle funzioni opportunamente campionate, le quali sono la descrizione matematica di segnali "classici": esponenziale decrescente, delta di Dirac, costante, gradino, coseno, porta e sinc. Per ognuno di questi è stato applicato uno sweep del fattore di scalamento in ampiezza e uno sweep del ritardo iniziale. Tali vettori sono stati generati dallo script *Python in_maker.py A.3.6*, il quale scrive in una sola riga tutti i campioni in codice esadecimale, per maggior leggibilità e compattezza. Oltre ai segnali così come sono, sul file di input a tale modulo vi salva anche la trasformata del segnale stesso, in modo da ottenere dei test anche con numeri complessi in ingresso. Durante il riempimento, genera anche un file contenente le *trasformate di Fourier* da utilizzare come riferimento.
I test effettuati sono stati 1120, dei quali i più significativi sono stati riportati al capitolo *Risultati delle simulazioni 2*. Se si desiderasse visualizzarli tutti, basta eseguire lo script Python *graphics.py A.3.7*, fornendo come argomento il file *fft_out_vectors.hex*, presente nella repository.

1.6 Resoconto e conclusione

Il circuito base dell'unità *Butterfly* progettata mostra un comportamento il linea con i modelli sviluppati, attestando un errore, rispetto al modello, mai superiore ad 1 *LSB* (rispetto alla *FFT* calcolata con il modulo *numpy* di Python). Tale risultato è di buona qualità, in quanto durante il progetto sono stati applicati accorgimenti per simplificarne la descrizione, l'implementazione (*arrotondamenti* e *scalamenti* impliciti) e l'interfacciamento, portando ad avere 3 ingressi e 2 uscite complessive per la singola unità. Come detto finora, la *Latenza* della singola unità è pari a 12 cicli e *Throughput* pari a 1/6, il fattore di scalamento è pari a 16 nel caso il dato in input non necessiti di scalamento per *Guard Bit* altrimenti vale 32. L'elevata modularità, compatibilità e automatizzazione ne han permesso un più rapido sviluppo e validazione del design stesso.
Il modello sviluppato in C per la FFT, purtroppo presenta ancora dei bug. Sarebbe stato bello usare quello per validare il progetto, e poi confrontato con i risultati ottenuti con la FFT presente nel modulo *numpy* di Python.

CHAPTER 2

Risultati delle Simulazioni

2.1 Simulazioni di FFT 16x16

Di seguito, sono riportati graficamente i risultati delle trasformate, confrontando il segnale nel tempo con il suo contenuto in frequenza calcolato, e successivamente valutato il relativo errore. Nei grafici, la traccia blu è relativa alla parte reale del segnale, quella arancione alla sua parte immaginaria (ad eccezione della valutazione dell'errore, dove sono separati). L'ordine dei dati nella stringa esadecimale è $x(0)$ in testa a sinistra, in ordine di indice crescente fino a $x(15)$ all'estremità destra. Ogni elemento è composto da 5 cifre esadecimali.

2.1.1 Esponenziale decrescente

Segnale: Esponenziale decrescente con valore massimo e costante di tempo unitari.

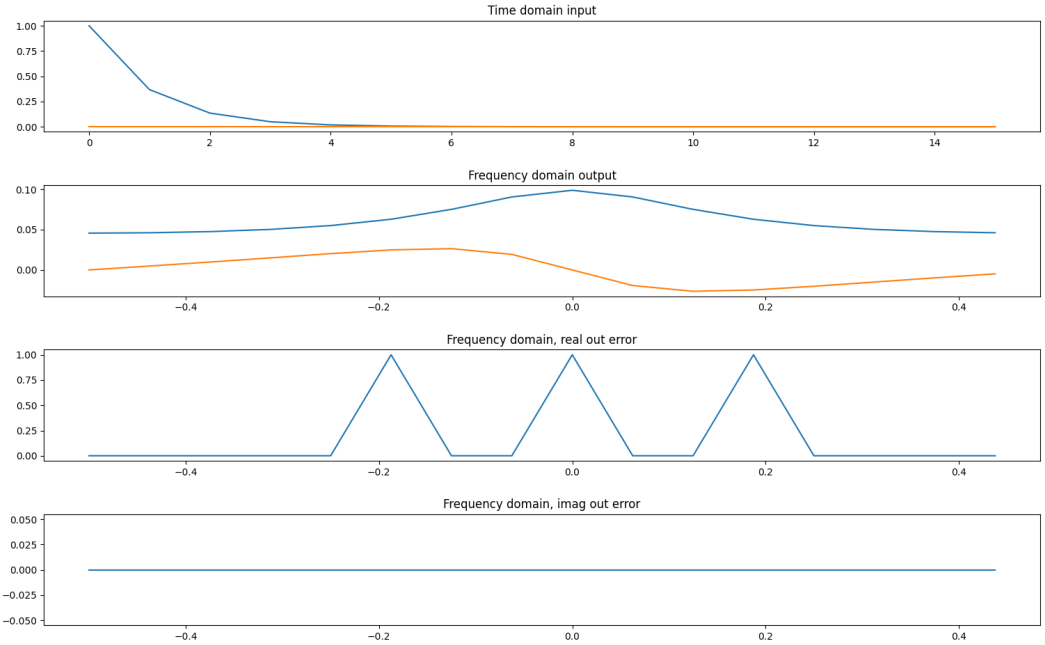


Figure 2.1: Esponenziale decrescente con valore massimo e costante di tempo unitarie, centrato all'istante iniziale

Dato	Contenuto
Re(IN)	3FFFF178B508A95032FB012C1006E60028A000EF00058000200000C000040000200001000000000
Im(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	0653E05CBC04CFC040690385F03387030B602F3E02ECA02F3E030B6033870385F0406904CFC05CBC
Im(OUT)	00000FEC39FE4EFFE686FEB43FF0A6FF5F2FFB0900000004F700A0E00F5A014BD0197A01B11013C7
Re(REF)	0653F05CBC04CFC040680385F03387030B602F3E02ECA02F3E030B6033870385F0406804CFC05CBC
Im(REF)	00000FEC39FE4EFFE686FEB43FF0A6FF5F2FFB0900000004F700A0E00F5A014BD0197A01B11013C7

Table 2.1: Vettori numerici relativi alla simulazione con esponenziale decrescente

Segnale: Esponenziale decrescente con valore massimo e costante di tempo unitari, traslato al campione 8.

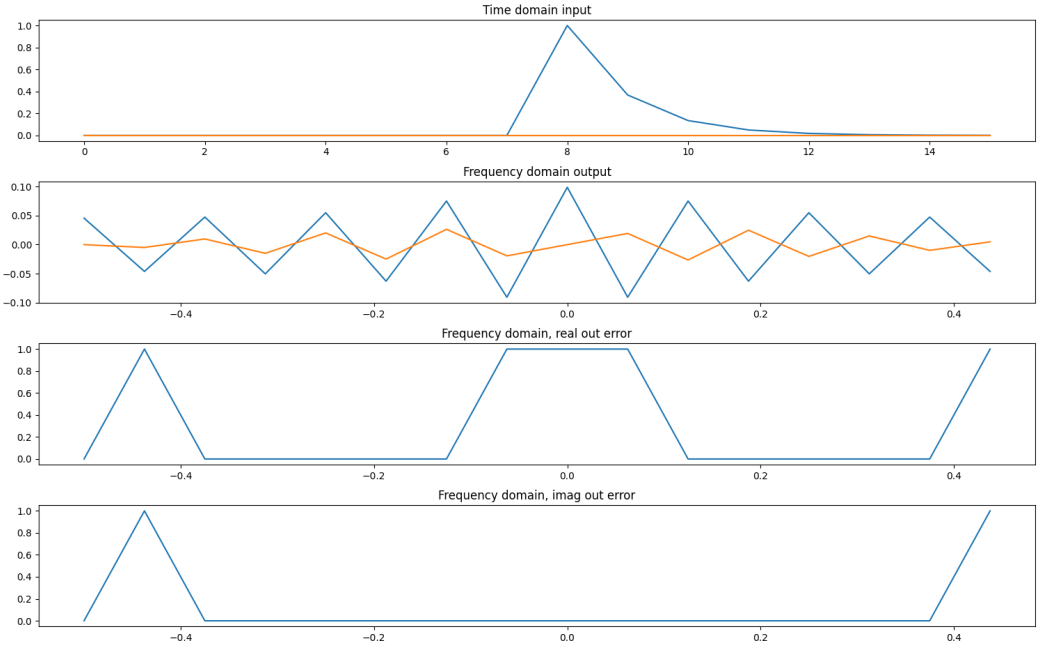


Figure 2.2

Dato	Contenuto
Re(IN)	0000000000000000000000000000000000000003FFFF178B508A95032FB012C1006E60028A000EF
Im(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	06535FA33D04CF6FBF920385AFCC74030B2FD0BD02EC6FD0BD030B2FCC740385AFBF9204CF6FA33D
Im(OUT)	00000013C9FE4F10197CFEB4500F5BFF5F3004F800000FFB0800A0DFF0A5014BBFE68401B0FFEC37
Re(REF)	06536FA33C04CF6FBF920385AFCC74030B2FD0BE02EC6FD0BE030B2FCC740385AFBF9204CF6FA33C
Im(REF)	00000013C9FE4F10197CFEB4500F5BFF5F3004F700000FFB0900A0DFF0A5014BBFE68401B0FFEC37

Table 2.2: Vettori numerici relativi alla simulazione con esponenziale decrescente traslato

2.1.2 Re-FFT della FFT dell'esponenziale decrescente

Segnale: Trasformata della trasformate dell'esponenziale decrescente con valore massimo e costante di tempo unitari, per avere valori immaginari in input e verificare la proprietà di reciprocità.

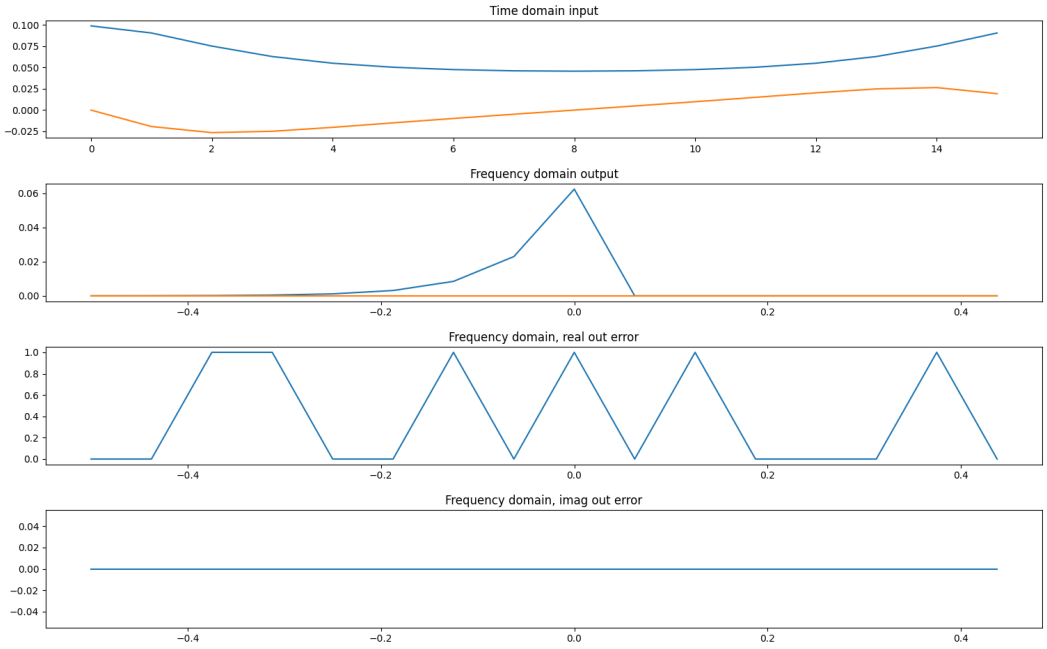


Figure 2.3: Trasformata del risultato precedente

Dato	Contenuto
Re(IN)	0653F05CBC04CFC040680385F03387030B602F3E02ECA02F3E030B6033870385F0406804CFC05CBC
Im(IN)	00000FEC39FE4EFFE686FEB43FF0A6FF5F2FFB0900000004F700A0E00F5A014BD0197A01B11013C7
Re(OUT)	03FFF0000000001000000000000000000000002000060000F000280006F0012C00330008AA0178B
Im(OUT)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	040000000000000000000000000000001000020000600000F000290006E0012C00330008A90178B
Im(REF)	00000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.3: Vettori numerici relativi alla simulazione con esponenziale decrescente ri-trasformato

Segnale: Trasformata della trasformate dell'esponenziale decrescente con valore massimo e costante di tempo unitari, traslato al campione 8.

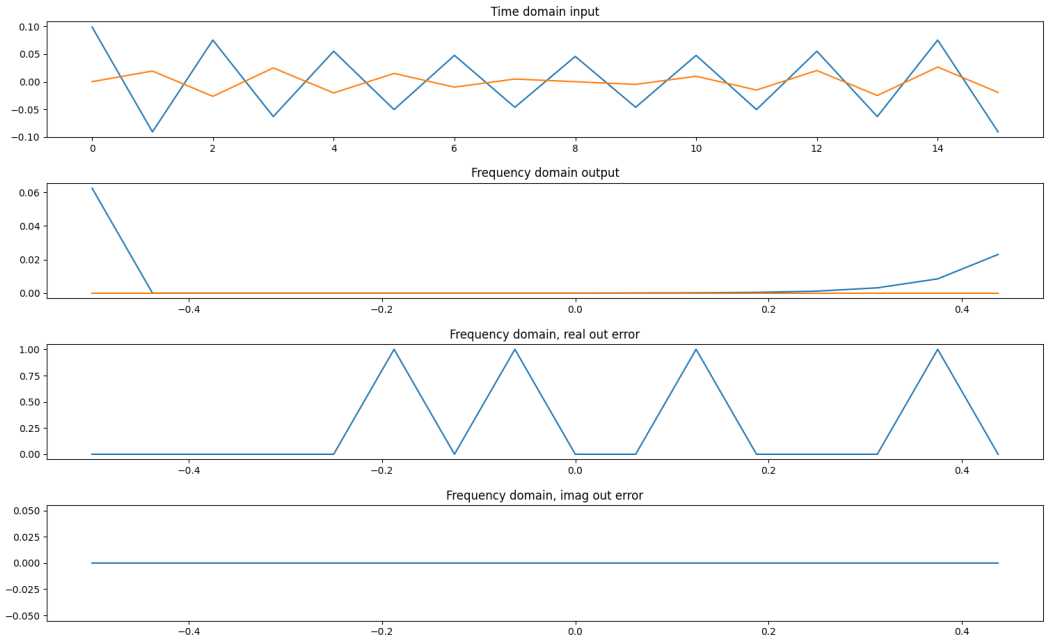


Figure 2.4: Trasformata della trasformata del segnale precedente, traslato.

Dato	**Contenuto**
Re(IN)	06536FA33C04CF6FBF920385AFCC74030B2FD0BE02EC6FD0BE030B2FCC740385AFBF9204CF6FA33C
Im(IN)	00000013C9FE4F10197CFEB4500F5BFF5F3004F700000FFB0900A0DFF0A5014BBFE68401B0FFEC37
Re(OUT)	000000000F000280006E0012C00330008AA0178B040000000000000000000000FFFFF0000000001
Im(OUT)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	000000000F000290006E0012C00330008A90178B040000000000000000000000000000000000000
Im(REF)	00000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.4: Vettori numerici relativi alla simulazione con esponenziale decrescente traslato, ri-trasformato

2.1.3 Delta di Dirac

Segnale: Delta di Dirac posizionata all'origine.

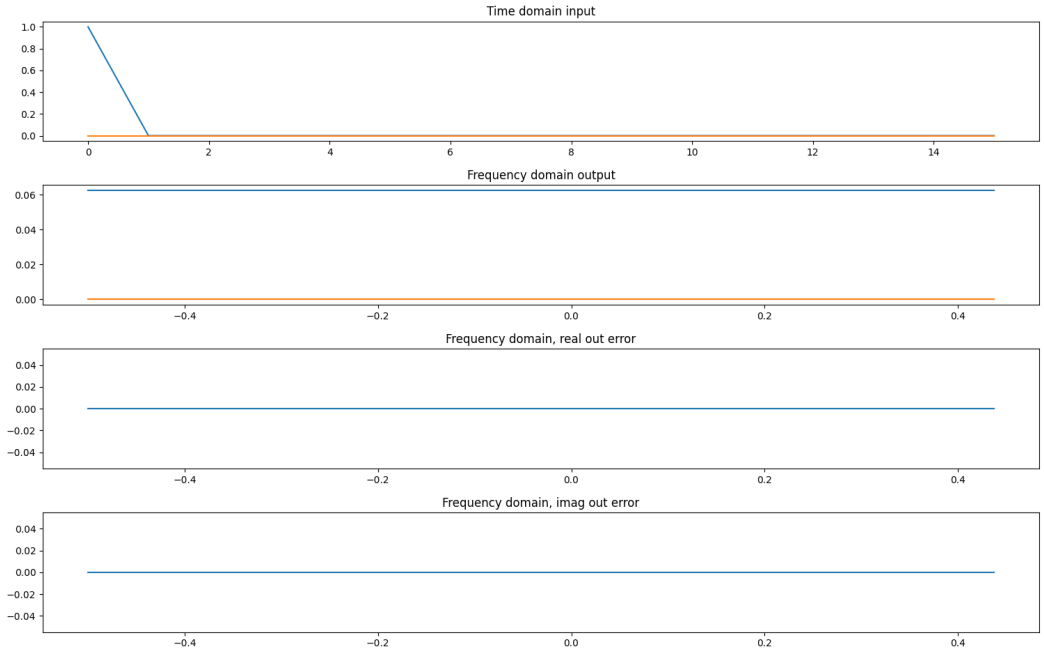


Figure 2.5: Delta di Dirac nell'origine

Dato	Contenuto
Re(IN)	3FFFF000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Im(IN)	000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	040000400004000040000400004000040000400004000040000400004000040000400004000040000400004000
Im(OUT)	000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	040000400004000040000400004000040000400004000040000400004000040000400004000040000400004000
Im(REF)	000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.5: Vettori numerici relativi alla simulazione con delta di Dirac

Segnale: Delta di Dirac posizionata al campione 8.

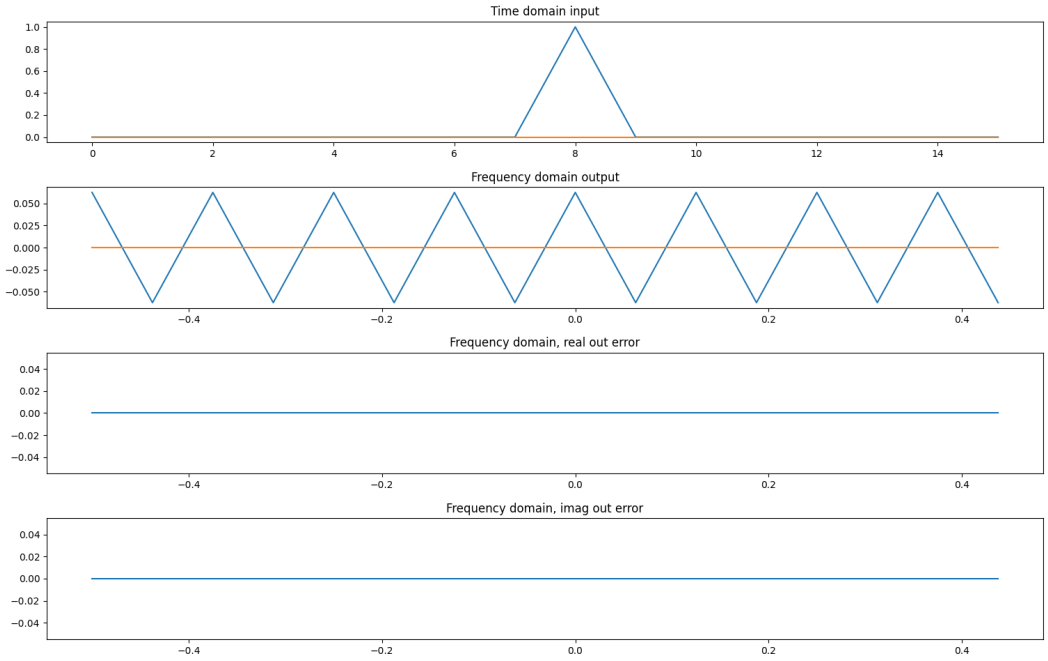


Figure 2.6: Delta di Dirac centrata nel campione 8

Dato	**Contenuto**
Re(IN)	0000000000000000000000000000000000000003FFFF00000000000000000000000000000000
Im(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	04000FC00004000FC00004000FC00004000FC00004000FC00004000FC00004000FC00004000FC000
Im(OUT)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	04000FC00004000FC00004000FC00004000FC00004000FC00004000FC00004000FC00004000FC000
Im(REF)	00000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.6: Vettori numerici relativi alla simulazione con delta di Dirac traslata

2.1.4 Re-FFT della FFT della Delta di Dirac: costante

Segnale: Trasformata di Fourier di una Delta di Dirac, e quindi una costante.

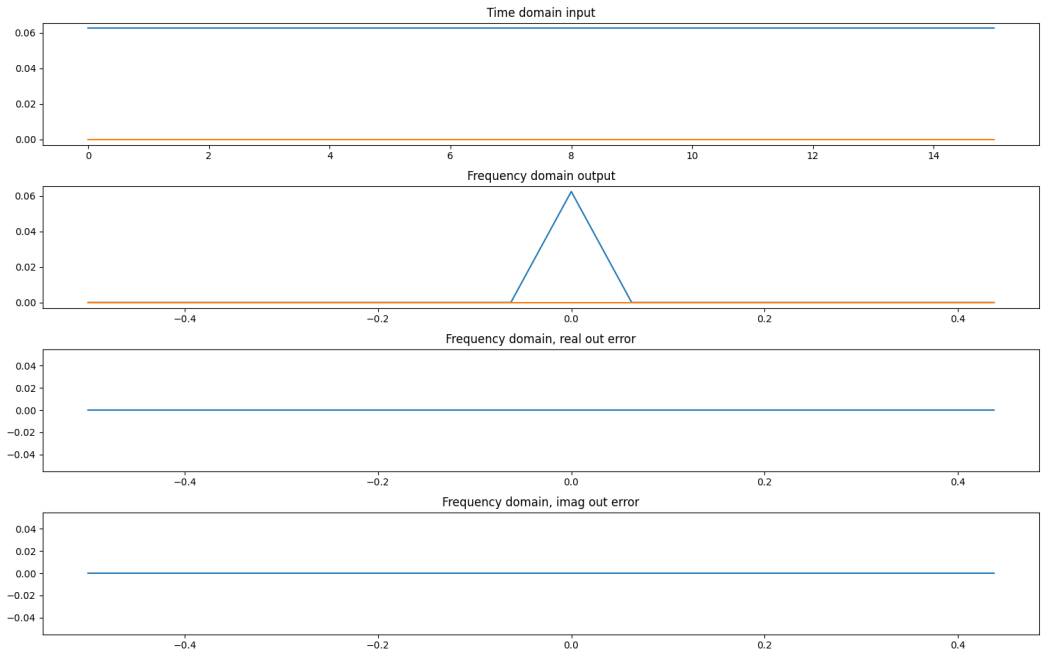


Figure 2.7: Costante

Dato	Contenuto
Re(IN)	0400004000040000400004000040000400004000040000400004000040000400004000040000400004000
Im(IN)	0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	0400000000000000000000000000000000000000000000000000000000000000000000000000000000000
Im(OUT)	0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	0400000000000000000000000000000000000000000000000000000000000000000000000000000000000
Im(REF)	0000000000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.7: Vettori numerici relativi alla simulazione con delta di Dirac ri-trasformata

Segnale: Trasformata di Fourier di una Delta di Dirac traslata.

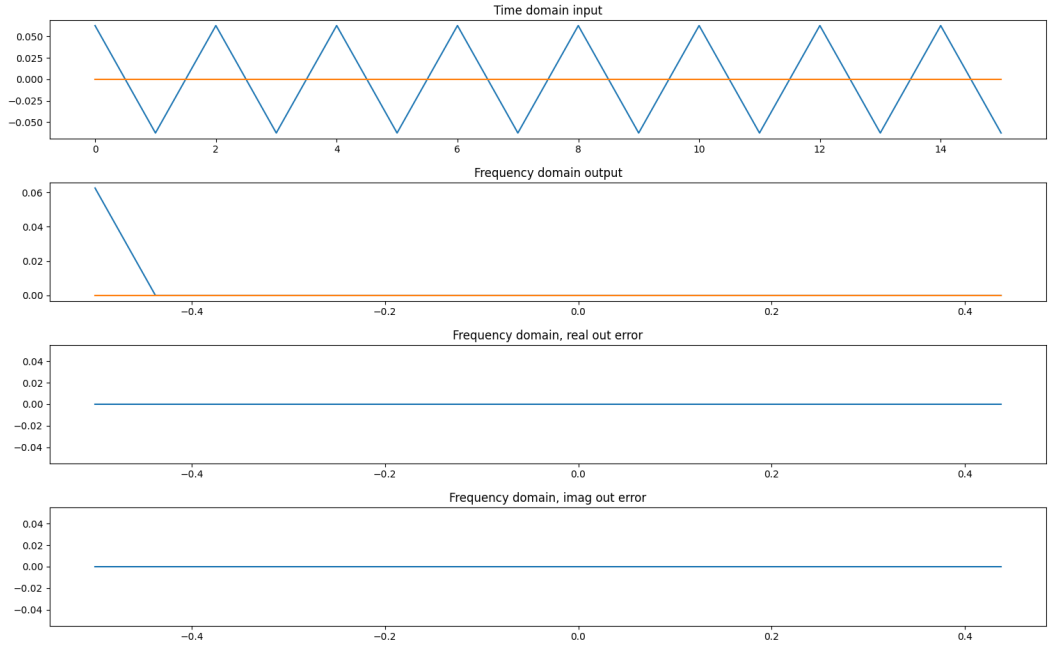


Figure 2.8: Contenuto in frequenza della Delta usato come campionamento nel tempo

Dato	**Contenuto**
Re(IN)	04000FC00004000FC00004000FC00004000FC00004000FC00004000FC00004000FC00004000FC000
Im(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	00000000000000000000000000000000000000004000000000000000000000000000000000000000
Im(OUT)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	00000000000000000000000000000000000000004000000000000000000000000000000000000000
Im(REF)	00000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.8: Vettori numerici relativi alla simulazione con delta traslata, ri-trasformata

2.1.5 Gradino

Segnale: Step unitario con ritardo pari a 8 campioni.

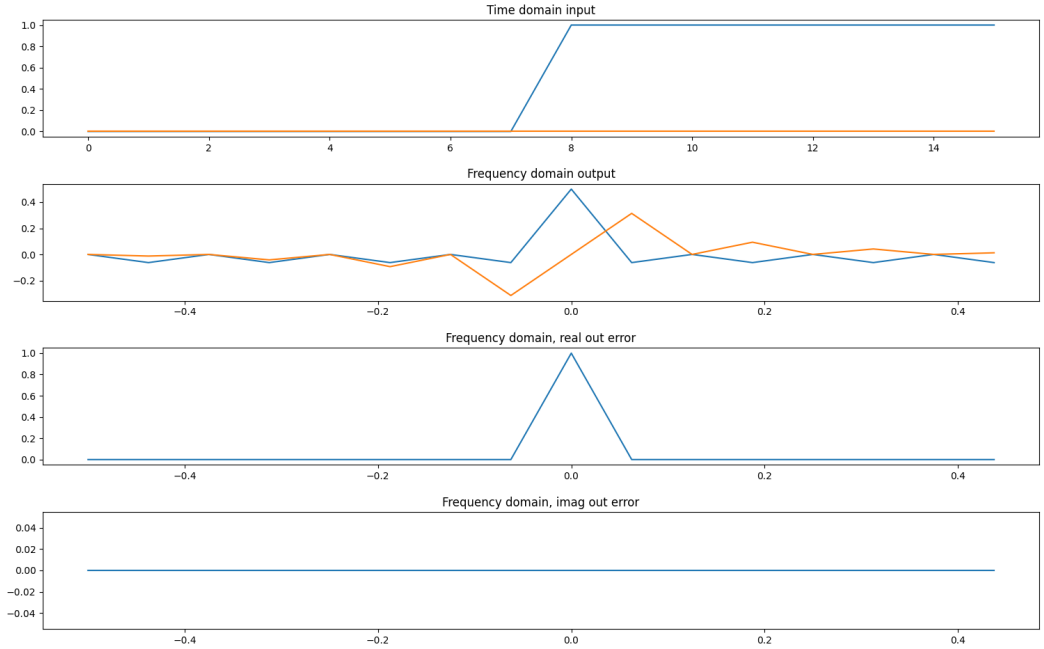


Figure 2.9: Gradino unitario centrato all'istante del campione 8

Dato	**Contenuto**
Re(IN)	0000000000000000000000000000000000000003FFFF3FFFF3FFFF3FFFF3FFFF3FFFF3FFFF3FFFF
Im(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	1FFFFFC00000000FC00000000FC00000000FC00000000FC00000000FC00000000FC000
Im(OUT)	00000141C00000005FC80000002AC30000000CBB00000FF34500000FD53D00000FA03800000EBE40
Re(REF)	20000FC00000000FC00000000FC00000000FC00000000FC00000000FC00000000FC000
Im(REF)	00000141C00000005FC80000002AC30000000CBB00000FF34500000FD53D00000FA03800000EBE40

Table 2.9: Vettori numerici relativi alla simulazione con gradino

2.1.6 Re-FFT della FFT del gradino

Segnale: Il contenuto in frequenza del gradino è ora usato come segnale nel tempo.

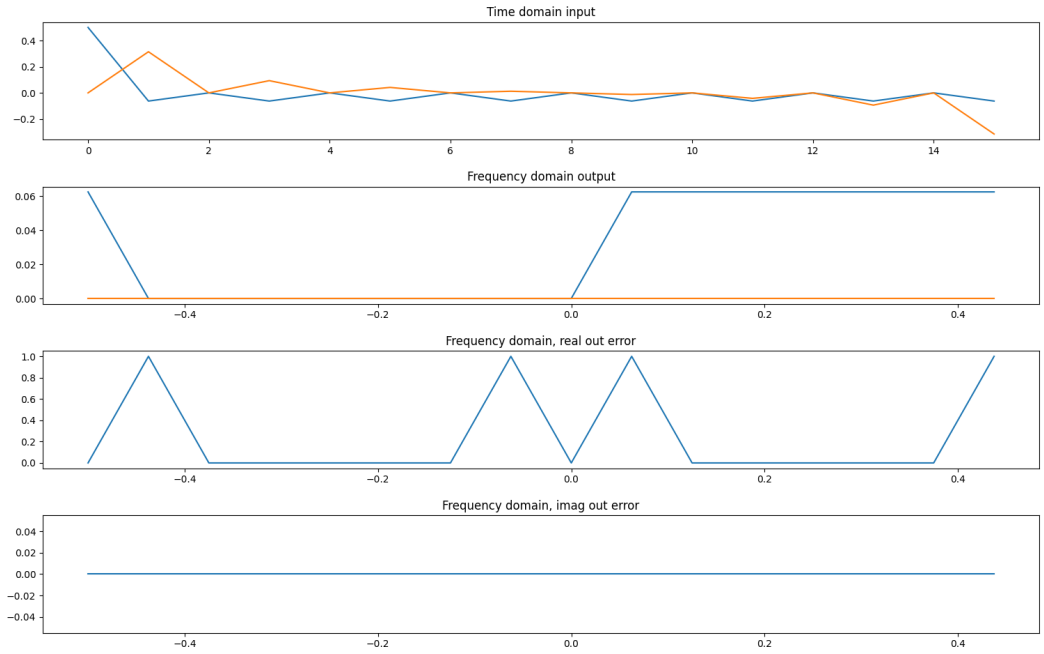


Figure 2.10: Segnale bizzarro, simile a sinc + delta

Dato	Contenuto
Re(IN)	20000FC00000000FC00000000FC00000000FC00000000FC00000000FC00000000FC00000000FC000
Im(IN)	00000141C00000005FC80000002AC30000000CBB00000FF34500000FD53D00000FA03800000EBE40
Re(OUT)	0000003FFF040000400004000040000400003FFF040000000100000000000000000000000000001
Im(OUT)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	00000040000400004000040000400004000040000400000000000000000000000000000000000000
Im(REF)	00000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.10: Vettori numerici relativi alla simulazione con gradino ri-trasformato

2.1.7 Coseno

Segnale: coseno con pulsazione tripla rispetto la finestra di campionamento.

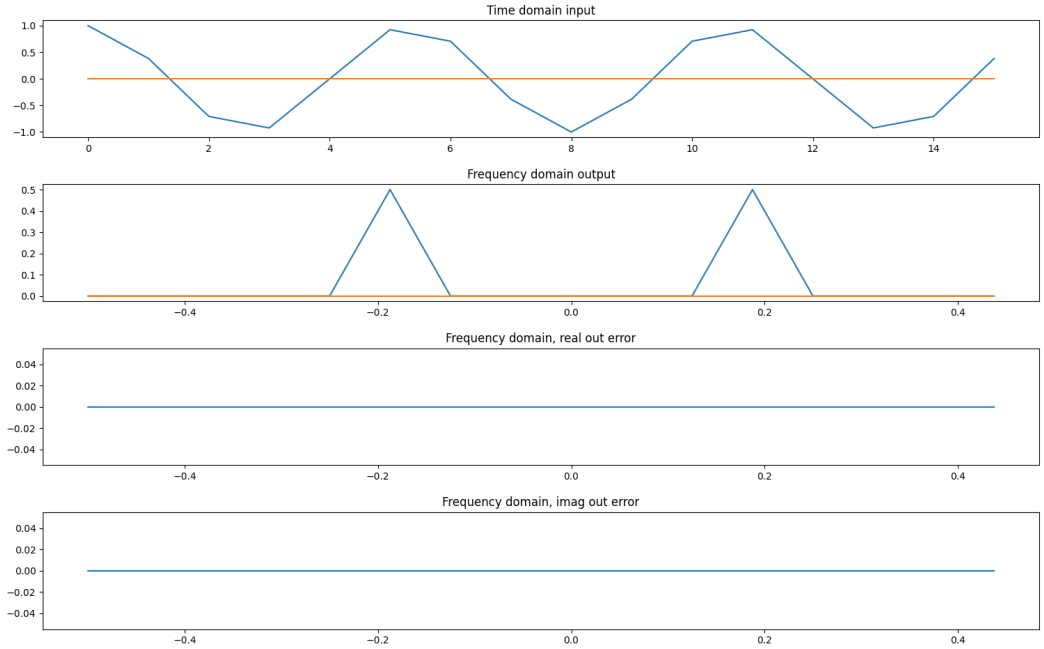


Figure 2.11: Coseno

Dato	Contenuto
Re(IN)	3FFFF187DED2BEDC4DF3000003B20D2D413E7822C0001E78222D4133B20D00000C4DF3D2BED187DE
Im(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	00000000000000020000000000000000000000000000000000000000000000000200000000000000
Im(OUT)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	00000000000000020000000000000000000000000000000000000000000000000200000000000000
Im(REF)	00000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.11: Vettori numerici relativi alla simulazione con coseno

Segnale: coseno con pulsazione tripla rispetto la finestra di campionamento e ritardo di fase pari a un semiperiodo.

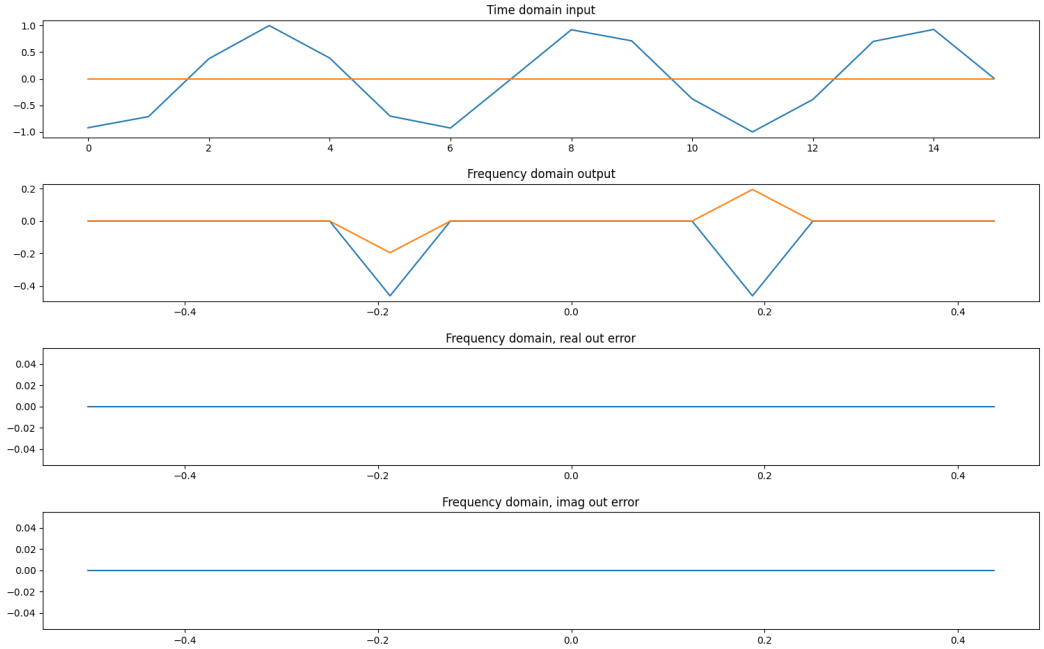


Figure 2.12: Coseno con ritardo di fase di un semi-periodo

Dato	**Contenuto**
Re(IN)	C50D6D26A9180F33FFF818EC3D313BC4B1DFF8863AF2A2D957E7F0DC0008E713D2CEC53B4E30077A
Im(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	000000000000000E286B0000000000000000000000000000000000000000E286B0000000000
Im(OUT)	0000000000000000C76200000000000000000000000000000000000000F389E0000000000
Re(REF)	000000000000000E286B0000000000000000000000000000000000000000E286B0000000000
Im(REF)	0000000000000000C76200000000000000000000000000000000000000F389E0000000000

Table 2.12: Vettori numerici relativi alla simulazione con coseno sfasato

2.1.8 Re-FFT del coseno: doppia delta simmetrica rispetto campione 8

Segnale: Doppia delta dello spettro del coseno.

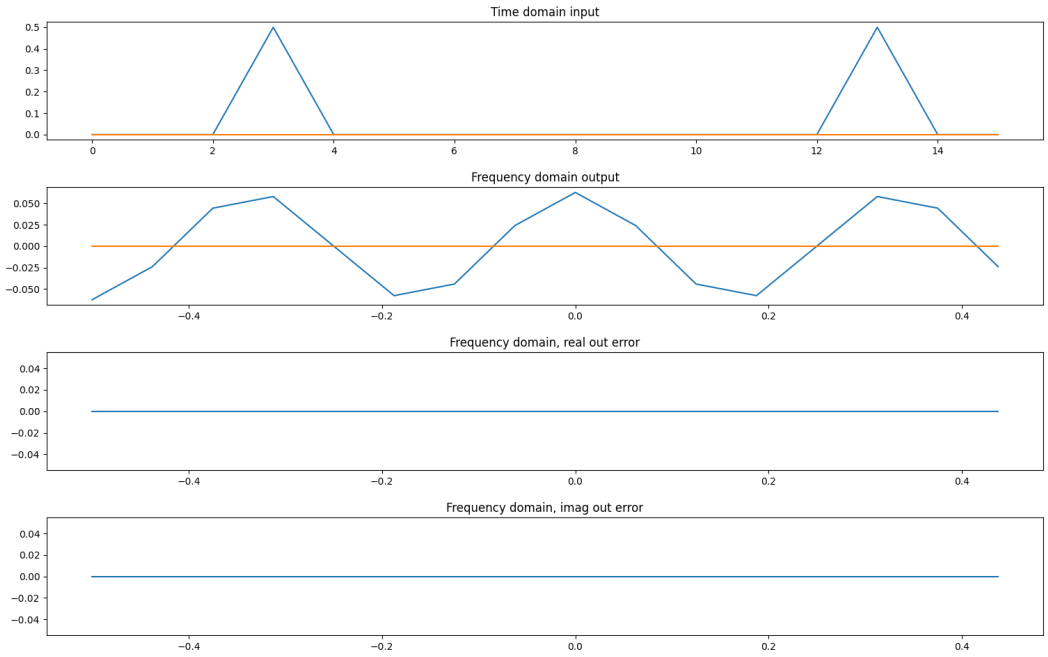


Figure 2.13: Doppia delta dello spettro del coseno usata nel tempo

Dato	Contenuto
Re(IN)	0000000000000002000000000000000000000000000000000000000000000000200000000000000
Im(IN)	0000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	040000187EFD2BFFC4DF0000003B2102D41FE782FC000FE78202D4103B2100000FC4DFFD2BF0187E
Im(OUT)	0000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	040000187EFD2BFFC4DF0000003B2102D41FE782FC000FE78202D4103B2100000FC4DFFD2BF0187E
Im(REF)	0000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.13: Vettori numerici relativi alla simulazione con coseno ri-trasformato

Segnale: delta composte dello spettro del coseno con ritardo di fase.

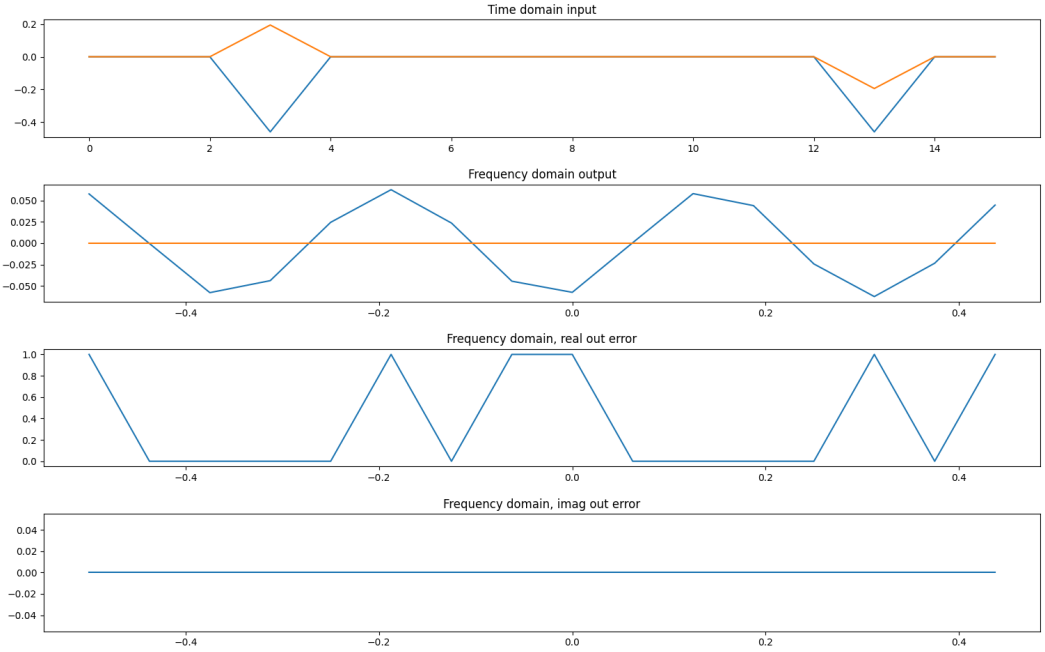


Figure 2.14: Delta composte del coseno con ritardo

Dato	Contenuto
Re(IN)	0000000000000000E286B0000000000000000000000000000000000000000000E286B0000000000
Im(IN)	0000000000000000C7620000000000000000000000000000000000000000000F389E0000000000
Re(OUT)	FC50E0007803B4E02CECFE714FC001FE7F102D9503AF2FFF88FC4B2FD314018EC03FFF0180FFD26B
Im(OUT)	0000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	FC50D0007803B4E02CECFE714FC000FE7F102D9603AF3FFF88FC4B2FD314018EC040000180FFD26A
Im(REF)	0000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.14: Vettori numerici relativi alla simulazione con coseno sfasato ri-trasformato

2.1.9 Costante nulla

Segnale: Una costante nulla per qualsiasi punto.

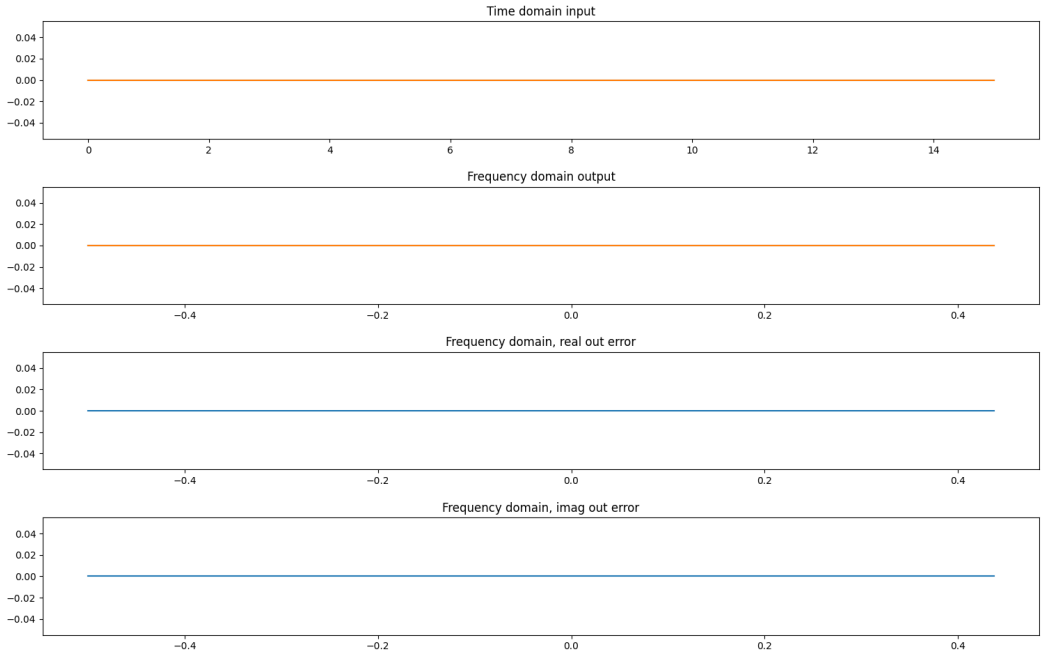


Figure 2.15: Costante nulla

Dato	Contenuto
Re(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Im(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Im(OUT)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	00000000000000000000000000000000000000000000000000000000000000000000000000000000
Im(REF)	00000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.15: Vettori numerici relativi alla simulazione con costante nulla

2.1.10 Porta

Segnale: porta centrata rispetto all'intervallo di campionamento.

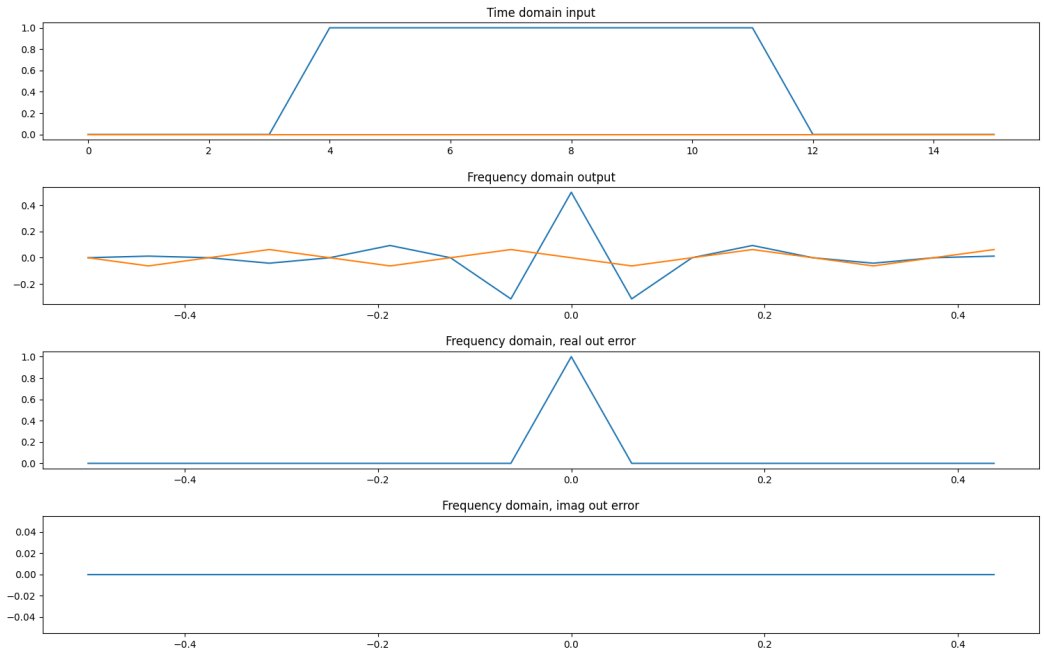


Figure 2.16: Porta centrata

Dato	Contenuto
Re(IN)	0000000000000000000003FFFF3FFFF3FFFF3FFFF3FFFF3FFFF3FFFF3FFFF3FFFF0000000000000000000
Im(IN)	00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(OUT)	1FFFFEBE400000005FC800000FD53D0000000CBB0000000CBB00000FD53D0000005FC800000EBE40
Im(OUT)	00000FC000000000400000000FC000000000400000000FC000000000400000000FC0000000004000
Re(REF)	20000EBE400000005FC800000FD53D0000000CBB0000000CBB00000FD53D0000005FC800000EBE40
Im(REF)	00000FC000000000400000000FC000000000400000000FC000000000400000000FC0000000004000

Table 2.16: Vettori numerici relativi alla simulazione con porta

2.1.11 Re-FFT della FFT della porta

Segnale: componente in frequenza della porta usata come segnale nel tempo.

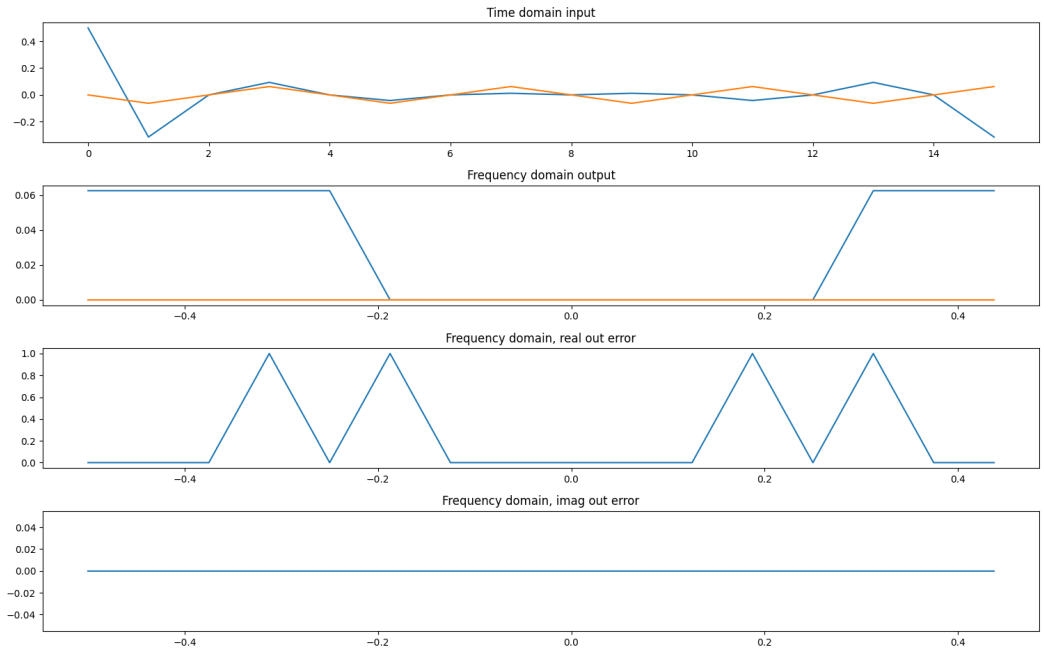


Figure 2.17: Praticamente due sinc agli estremi

Dato	Contenuto
Re(IN)	20000EBE400000005FC800000FD53D0000000CBB0000000CBB00000FD53D0000005FC800000EBE40
Im(IN)	00000FC000000000400000000FC000000000400000000FC000000000400000000FC0000000004000
Re(OUT)	00000000000000000000010000003FFF040000400004000040000400003FFF040000000100000000000
Im(OUT)	00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Re(REF)	00000000000000000000000000004000040000400004000040000400004000040000000000000000000
Im(REF)	00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

Table 2.17: Vettori numerici relativi alla simulazione con porta ri-trasformata: sinc

2.1.12 Errore medio

Al termine delle 1120 simulazioni, l'errore medio, valutato in *LSB*, è mostrato in *figura 2.18*. Il valore massimo attestato è pari a $0.3LSB$, al campione reale centrale.

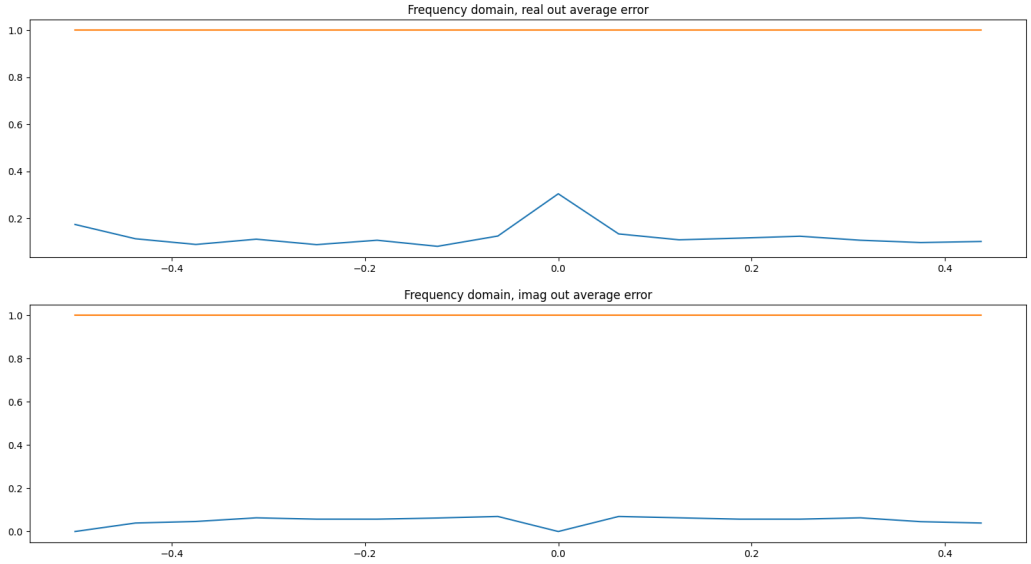


Figure 2.18: Errore punto per punto del risultato, in LSB

CHAPTER 3

Flusso di Progetto

3.1 Progettazione datapath

3.1.1 Descrizione delle unità base e Datapath

Questo passo progettuale è stato molto diretto: per ogni unità utilizzata, è stato descritto il relativo codice *VHDL* comportamentale, il quale è stato poi verificato manualmente dopo una simulazione del relarivo testbench, data la semplicità di queste. Unico step di automatizzazione è l'uso degli script *Bash* per compilare e avviare *ModelSim* da linea di comando con il rispettivo file di comandi da eseguire per la simulazione. Il codice è stato reso piuttosto flessibile e generico grazie all'introduzione dei file-package *definespack.vhd* *A.1.1* e *cupack.vhd* *A.1.2*.

3.2 Progettazione della Contro-Unit

3.2.1 Descrizione della Control-Unit

Questo passo è stato fortemente automatizzato per quanto riguarda la *micro-ROM*. Il codice *VHDL* della Control-Unit è stato prodotto manualmente, mentre quello delle *micro-ROM* parzialmente: la struttura manualmente, la mappatura del contenuto di memoria riempito automaticamente con uno script.
Per ottenere ciò sono state realizzate delle mappature in 4 file di fogli elettronici *LibreOffice*: *labels.ods* *1.20*, *rom_even.ods* *1.21*, *rom_odd.ods* *1.22* e *ctrlwrds.ods* *1.23*. Il primo contiene le etichette che saranno salvate nel file *cupack.vhd* per avere un tag mnemonico durante la lettura dell'indirizzo attuale e successivo della *ROM*, l'ultimo contiene i tag mnemonici a cui sono associati i bit della control-word di un determinato stato e anch'essi mappati nel file *cupack.vhd*. Il terzo e quarto contengono le mappature, espresse in tag, del contenuto delle memorie. Tutta questa complicazione ha in realtà reso molto più semplice il testing e debug della control-unit. I suddetti fogli elettronici sono stati poi convertiti in formato *csv* e utilizzati dagli script *Python* *cupack_updater.py* *A.3.1* e *rom_modifier.py* *A.3.2* per finalizzare i sorgenti *VHDL* delle *micro-ROM* e *cupack_vhd*.

3.2.2 Descrizione dell'unità FFT 16x16

L'unità dimostrativa *fft_unit.vhd* è stata ottenuta quasi completamente in modo automatico con lo script *Python* *fft_instancer.py* *A.3.5*, il quale prende il file *fft_unit.vhd* e ne sostituisce per intero le dichiarazioni dei segnali, interconnessioni e instanziamento delle unità, realizzando il corpo del codice instanziando entità e segnali al bisogno. Ciò ha permesso di ridurre enormemente le possibilità di errore umano e gli errori del codice sono facilmente individuabili grazie alla struttra iterativa dello script.

3.2.3 Verifica dell'unità Butterfly

Per ottenere un modello affidabile dell'unità *Butterfly* dalla buona qualità comportamentale, è stato sviluppato il programma in linguaggio C *butterflyTest*, il quale replica il comportamento del circuito nelle sue caratteristiche numeriche considerando le caratteristiche del tipo *unsigned* rispetto al *signed* e scendendo a basso livello per emulare shift, somme, prodotti e arrotondamento impiegando solo numeri interi per mappare l'algoritmo

all'architettura, che di fatto applica le nozioni necessarie per avere la transizione da intero a fixed point. Di fatto, architettura e programma in C impiegano solo il tipo intero.

Verifica dell'unità FFT 16x16 Anche in questo caso, è stato realizzato il programma in linguaggio C *fftTest* per emulare l'architettura, sfruttando la funzione *_butterfly* usata precedentemente per l'unità *Butterfly*, e chiamata iterativamente. Purtroppo ad oggi presenta ancora qualche bug e non è completamente aderente al comportamento dell'unità *fft_unit.vhd*, mentre il file di riferimento creato in precedenza dallo script *Python in_maker* è praticamente identico, a scanso di qualche errore di arrotondamento.

3.2.4 Automatizzazione delle simulazioni

Le simulazioni sono state rese automatizzate grazie agli script *comp_and_sim.sh* *A.4.1* e *infinite_sim.sh* *A.4.2* (usato nel caso si volessero testare tutti i vettori di un file, basato su asserzione di un failure e Here-file dei sistemi Unix per uscire da ModelSim). Tali script permettono di avviare *ModelSim* in batch-mode, con interfaccia grafica o solo compilare, grazie all'opzione -nogui, -dbg o -nosim. Prende come file di input uno dei file *compile_order_*⟨*unit_name*⟩ per effetturare compilazione e simulazione.
Il file *simulate_*⟨*unit_name*⟩*.do* sono forniti a *ModelSim* durante l'avvio. I dati vengono salvati dai testbench in file dal nome riconoscibile.

APPENDIX A

Source files

I file utilizzati per lo sviluppo e realizzazione del progetto sono organizzati per categoria e in modo gerarchico. Verranno presentati innanzitutto i sorgenti *VHDL*, dato che sono di interesse. Gli altri file sono riportati solo per dimostrazione che quanto detto finora corrisponde al vero, dimostrando l'esistenza dei file citati. Questo progetto è stato sviluppato sfruttando la piattaforma *GitHub*, in una repository privata. Nel caso si fosse interessati a visitarla, prego di contattare l'autore.

A.1 VHDL source files

I file presentanti sono in ordine gerarchico.
Nota: le *micro-ROM* sono state implementate con un codice che le renda simili ad una tabella organizzata a righe, e perciò basata sul costrutto *CASE-WHEN*.

A.1.1 definespack.vhd

```
-- Author:        Luca Lombardini
-- Academic_y:    2020/2021
-- Purpose:       (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:      s277972@studenti.polito.it
--                lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;

PACKAGE definespack IS
--#############################################################################
--#     Top level interface defines
        CONSTANT io_width       : positive := 20;
--#############################################################################
--#     Coefficient's Register file defines
        CONSTANT rfc_data_width : positive := io_width;
        CONSTANT rfc_addr_width : positive := 1;
        CONSTANT rfc_wr_ports   : positive := 1;
        CONSTANT rfc_rd_ports   : positive := 1;
--#############################################################################
--#     Data's Register file defines
        CONSTANT rfd_data_width : positive := io_width;
        CONSTANT rfd_addr_width : positive := 2;
        CONSTANT rfd_wr_ports   : positive := 2;
        CONSTANT rfd_rd_ports   : positive := 2;
--#############################################################################
```

```
--#     Multiplier defines
        CONSTANT m_in_width     : positive := io_width;
        CONSTANT prod_width     : positive := m_in_width*2;
--##############################################################################
--#     Adder_subtractor defines
        CONSTANT add_width      : positive := prod_width;
--##############################################################################
--#     Rounder defines
        CONSTANT round_i_width  : positive := add_width;
        CONSTANT round_o_width  : positive := io_width;
--##############################################################################
--#     Testbenches defines
        CONSTANT clk_period     : time     := 10 ns;
        CONSTANT rst_release    : time     := 13 ns;
        CONSTANT dut_cycle_lat  : positive := 12;
        CONSTANT serial_data    : integer  := 4;
        CONSTANT adder_const    : signed   := to_signed(1,add_width);
        CONSTANT mult_const     : signed   := to_signed(16,m_in_width);
        CONSTANT rf_data_test   : positive := 4;
        CONSTANT rf_addr_test   : positive := 2;
        CONSTANT rf_wr_test     : positive := 4;
        CONSTANT rf_rd_test     : positive := 2;
        CONSTANT maker_width    : positive := rf_data_test*rf_wr_test;
        CONSTANT sink_width     : positive := rf_data_test*rf_rd_test;
--##############################################################################
--#     FFT defines
        CONSTANT n_samples      : positive := 16;
        CONSTANT n_coef         : positive := n_samples/2;
        CONSTANT data_ports     : positive := 1;
        CONSTANT butt_per_level : positive := 8;
        CONSTANT coef_ports     : positive := 8;
        CONSTANT new_coeffs     : positive := n_coef/2;
        CONSTANT out_ports      : positive := 1;
END definespack;
```

A.1.2 cupack.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;

PACKAGE cupack IS
   CONSTANT base_addr    : integer                                  := 4;
   CONSTANT lsb_addr     : integer                                  := 5;
   CONSTANT cc_lsb_addr  : integer                                  := 6;
   CONSTANT command_len  : integer                                  := 28;
   CONSTANT uir_width    : integer                                  := 34;
   CONSTANT IDLE_LD_R    : std_logic_vector(base_addr-1 DOWNTO 0)   := "0000";
   CONSTANT S_M1         : std_logic_vector(base_addr-1 DOWNTO 0)   := "0001";
```

```
CONSTANT S_M3          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0010";
CONSTANT S_M2          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0011";
CONSTANT S_M4          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0100";
CONSTANT S_M5          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0101";
CONSTANT X_M6          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0110";
CONSTANT X_S5          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0111";
CONSTANT X_S6          : std_logic_vector(base_addr-1 DOWNTO 0)  := "1000";
CONSTANT X_RND_BR      : std_logic_vector(base_addr-1 DOWNTO 0)  := "1001";
CONSTANT X_SND_R       : std_logic_vector(base_addr-1 DOWNTO 0)  := "1010";
CONSTANT S_SND_I       : std_logic_vector(base_addr-1 DOWNTO 0)  := "1011";
CONSTANT C_SND_I       : std_logic_vector(base_addr-1 DOWNTO 0)  := "1100";
CONSTANT IDLE          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0000";
CONSTANT S_M6          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0110";
CONSTANT S_S5          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0111";
CONSTANT S_S6          : std_logic_vector(base_addr-1 DOWNTO 0)  := "1000";
CONSTANT S_RND_BR      : std_logic_vector(base_addr-1 DOWNTO 0)  := "1001";
CONSTANT S_SND_R       : std_logic_vector(base_addr-1 DOWNTO 0)  := "1010";
CONSTANT C_M6          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0110";
CONSTANT C_S5          : std_logic_vector(base_addr-1 DOWNTO 0)  := "0111";
CONSTANT C_S6          : std_logic_vector(base_addr-1 DOWNTO 0)  := "1000";
CONSTANT C_RND_BR      : std_logic_vector(base_addr-1 DOWNTO 0)  := "1001";
CONSTANT C_SND_R       : std_logic_vector(base_addr-1 DOWNTO 0)  := "1010";
CONSTANT CW_IDLE       : std_logic_vector(command_len-1 DOWNTO 0):= "00000000000000000000000000000000";
CONSTANT CW_SLD_R      : std_logic_vector(command_len-1 DOWNTO 0):= "01000011100000000000000000000000";
CONSTANT CW_SM1        : std_logic_vector(command_len-1 DOWNTO 0):= "11010111101000000000000000000000";
CONSTANT CW_SM3        : std_logic_vector(command_len-1 DOWNTO 0):= "00100000001000100000000000000000";
CONSTANT CW_SM2        : std_logic_vector(command_len-1 DOWNTO 0):= "00100000011000110000000000000000";
CONSTANT CW_SM4        : std_logic_vector(command_len-1 DOWNTO 0):= "00000000011100100000110000000000";
CONSTANT CW_SM5        : std_logic_vector(command_len-1 DOWNTO 0):= "00000000000001100010001000000000";
CONSTANT CW_SM6        : std_logic_vector(command_len-1 DOWNTO 0):= "00000000010001100001000000000000";
CONSTANT CW_SS5        : std_logic_vector(command_len-1 DOWNTO 0):= "00000000000000101111000100000000";
CONSTANT CW_SS6        : std_logic_vector(command_len-1 DOWNTO 0):= "00000000000000000000101011000000";
CONSTANT CW_SRND_BR    : std_logic_vector(command_len-1 DOWNTO 0):= "00000000000000000000000101100000";
CONSTANT CW_SSND_R     : std_logic_vector(command_len-1 DOWNTO 0):= "00000000000000000000000010011000";
CONSTANT CW_SSND_I     : std_logic_vector(command_len-1 DOWNTO 0):= "00000000000000000000000001100000";
CONSTANT CW_CM6        : std_logic_vector(command_len-1 DOWNTO 0):= "01000011110001100001000000000000";
CONSTANT CW_CS5        : std_logic_vector(command_len-1 DOWNTO 0):= "11010111101000101111000100000000";
CONSTANT CW_CS6        : std_logic_vector(command_len-1 DOWNTO 0):= "00100000001000100000101011000000";
CONSTANT CW_CRND_BR    : std_logic_vector(command_len-1 DOWNTO 0):= "00100000011000110000000101100000";
CONSTANT CW_CSND_R     : std_logic_vector(command_len-1 DOWNTO 0):= "00000000011100100000110010011000";
CONSTANT CW_CSND_I     : std_logic_vector(command_len-1 DOWNTO 0):= "00000000000001100010001001100000";
CONSTANT RFC_WR_ADDR   : integer                                 := 27;
CONSTANT RFC_WR        : integer                                 := 26;
CONSTANT RFC_RD_ADDR   : integer                                 := 25;
CONSTANT RFD_WR1_ADDR0 : integer                                 := 24;
CONSTANT RFD_WR1_ADDR1 : integer                                 := 23;
CONSTANT RFD_WR2_ADDR0 : integer                                 := 22;
CONSTANT RFD_WR2_ADDR1 : integer                                 := 21;
CONSTANT RFD_WR1       : integer                                 := 20;
CONSTANT RFD_WR2       : integer                                 := 19;
CONSTANT RFD_RD1_ADDR0 : integer                                 := 18;
CONSTANT RFD_RD1_ADDR1 : integer                                 := 17;
CONSTANT RFD_RD2_ADDR0 : integer                                 := 16;
CONSTANT RFD_RD2_ADDR1 : integer                                 := 15;
CONSTANT MULT_DOUBLE   : integer                                 := 14;
```

```
    CONSTANT R_MULT_LD    : integer                                  := 13;
    CONSTANT MUX1_SEL     : integer                                  := 12;
    CONSTANT MUX2_SEL     : integer                                  := 11;
    CONSTANT MUX3_SEL     : integer                                  := 10;
    CONSTANT ADD1_SUB_ADD : integer                                  := 9;
    CONSTANT R_ADD1_LD    : integer                                  := 8;
    CONSTANT MUX4_SEL     : integer                                  := 7;
    CONSTANT ADD2_SUB_ADD : integer                                  := 6;
    CONSTANT R_ADD2_LD    : integer                                  := 5;
    CONSTANT MUX5_SEL     : integer                                  := 4;
    CONSTANT REORD_LD     : integer                                  := 3;
    CONSTANT OUT_A_BUF_LD : integer                                  := 2;
    CONSTANT OUT_B_BUF_LD : integer                                  := 1;
    CONSTANT DONE_BIT     : integer                                  := 0;
END cupack;
```

A.1.3 reg.vhd

```
-- Author:        Luca Lombardini
-- Academic_y:    2020/2021
-- Purpose:       (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:      s277972@studenti.polito.it
--                lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;

ENTITY reg IS
        GENERIC(bitwidth: positive        := 8);
        PORT(   CLK     : IN std_logic;
                    RST_n   : IN std_logic;
                LD      : IN std_logic;
                D_IN    : IN signed(bitwidth-1 DOWNTO 0);
                D_OUT   : OUT signed(bitwidth-1 DOWNTO 0));
END ENTITY;


ARCHITECTURE behav OF reg IS
BEGIN
        reg_define: PROCESS(RST_n, CLK)
        BEGIN
                IF RST_n = '0' THEN
                        D_OUT <= to_signed(0, bitwidth);
                ELSIF CLK'EVENT AND CLK = '1' THEN
                        IF LD = '1'THEN
                                D_OUT <= D_IN;
                        END IF;
                END IF;
        END PROCESS;
END ARCHITECTURE;
```

A.1.4 register_file.vhd

```
-- Author:        Luca Lombardini
-- Academic_y:    2020/2021
-- Purpose:       (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:      s277972@studenti.polito.it
--                lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;

ENTITY register_file IS
    GENERIC(addr_width : positive    := 4;
            data_width : positive    := 8;
            wr_ports   : positive    := 1;
            rd_ports   : positive    := 1);
    PORT(   CLK        : IN std_logic;
            WR         : IN std_logic_vector(wr_ports-1 DOWNTO 0);
            WR_ADDR    : IN unsigned(wr_ports*addr_width-1 DOWNTO 0);
            WR_DATA    : IN signed(wr_ports*data_width-1 DOWNTO 0);
            RD_ADDR    : IN unsigned(rd_ports*addr_width-1 DOWNTO 0);
            RD_DATA    : OUT signed(rd_ports*data_width-1 DOWNTO 0));
END ENTITY;



ARCHITECTURE behav OF register_file IS
    SUBTYPE reg_locs IS NATURAL RANGE 0 TO ((2**(addr_width)) -1);
    TYPE reg_array IS ARRAY(reg_locs) OF signed(data_width-1 DOWNTO 0);
    SIGNAL register_block    : reg_array;
BEGIN
    read_define: PROCESS(RD_ADDR, register_block)
    BEGIN
        FOR i IN 0 TO rd_ports-1 LOOP
            RD_DATA((i+1)*data_width-1 DOWNTO i*data_width)
                <= register_block(to_integer(RD_ADDR((i+1)*addr_width-1 DOWNTO i*addr_width)));
        END LOOP;
    END PROCESS;

    write_define:PROCESS(CLK)
    BEGIN
        IF CLK'EVENT AND CLK = '1' THEN
            FOR j IN 0 TO wr_ports-1 LOOP
                IF WR(j) = '1' THEN
                    register_block(to_integer(WR_ADDR((j+1)*addr_width-1 DOWNTO j*addr_width)))
                        <= WR_DATA((j+1)*data_width-1 DOWNTO j*data_width);
                END IF;
            END LOOP;
        END IF;
    END PROCESS;
END ARCHITECTURE;
```

A.1.5 multiplier.vhd

```
-- Author:        Luca Lombardini
-- Academic_y:    2020/2021
-- Purpose:       (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:      s277972@studenti.polito.it
--                lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;

ENTITY multiplier IS
        PORT(   CLK     : IN std_logic;
                A       : IN signed(m_in_width-1 DOWNTO 0);
                B       : IN signed(m_in_width-1 DOWNTO 0);
                D_M_n   : IN std_logic;
                PROD    : OUT signed(prod_width-1 DOWNTO 0));
END ENTITY;


ARCHITECTURE behav OF multiplier IS
BEGIN
        mult_define: PROCESS(CLK)
                VARIABLE tmp        : std_logic_vector(prod_width-1 DOWNTO 0);
        BEGIN
                IF CLK'EVENT AND CLK = '1' THEN
                        IF D_M_n = '0' THEN
                                PROD <= A * B;
                        ELSE
                                tmp(prod_width-1 DOWNTO m_in_width) := std_logic_vector(A);
                                tmp(m_in_width-1 DOWNTO 0) := (OTHERS => '0');
                                PROD <= signed(tmp);
                        END IF;
                END IF;
        END PROCESS;
END ARCHITECTURE;
```

A.1.6 adder.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;

ENTITY adder IS
        PORT(   CLK     : IN std_logic;
                A       : IN signed(add_width-1 DOWNTO 0);
                B       : IN signed(add_width-1 DOWNTO 0);
                S_A_n   : IN std_logic;
                SUM     : OUT signed(add_width-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE behav OF adder IS
BEGIN
        adder_define: PROCESS(CLK)
        BEGIN
                IF CLK'EVENT AND CLK = '1' THEN
                        IF S_A_n = '0' THEN
                                SUM <= A + B;
                        ELSE
                                SUM <= A - B;
                        END IF;
                END IF;
        END PROCESS;
END ARCHITECTURE;
```

A.1.7 mux2to1.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;

ENTITY mux2to1 IS
        GENERIC(bitwidth: positive        := 8);
        PORT(   A       : IN signed(bitwidth-1 DOWNTO 0);
                B       : IN signed(bitwidth-1 DOWNTO 0);
                SEL     : IN std_logic;
                Y       : OUT signed(bitwidth-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE behav OF mux2to1 IS
BEGIN
        Y <= A WHEN SEL = '0' ELSE B;
```

```
END ARCHITECTURE;
```

A.1.8 rounder.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;

ENTITY rounder IS
        PORT(   ROUND_IN        : IN signed(round_i_width-1 DOWNTO 0);
                ROUND_OUT       : OUT signed(round_o_width-1 DOWNTO 0));
END ENTITY;


ARCHITECTURE behav OF rounder IS
        SIGNAL upperHalf                : signed(round_o_width-1 DOWNTO 0);
        SIGNAL trailPart                : std_logic_vector(round_o_width-2 DOWNTO 0);
        SIGNAL round                    : signed(round_o_width-1 DOWNTO 0);
        SIGNAL notZero, isCenter, isOdd : std_logic;
        CONSTANT zero_vect              : std_logic_vector(round_o_width-2 DOWNTO 0) := std_logic_vecto
BEGIN
        trailPart <= std_logic_vector(ROUND_IN(round_o_width-2 DOWNTO 0));
        isCenter  <= ROUND_IN(round_o_width-1);
        isOdd     <= ROUND_IN(round_o_width);
        notZero   <= '0' WHEN trailPart = zero_vect ELSE '1';

        round(round_o_width-1 DOWNTO 1) <= (OTHERS => '0');
        round(0)  <= (isCenter AND (isOdd OR notZero));
        upperHalf <= ROUND_IN(round_i_width-1 DOWNTO round_i_width - round_o_width);
        ROUND_OUT <= upperHalf + round AFTER (3*clk_period)/4;
END ARCHITECTURE;
```

A.1.9 rom_even.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;
USE work.cupack.all;

ENTITY rom_even IS
        PORT(   ADDR    : IN std_logic_vector(base_addr-1 DOWNTO 0);
                DATA    : OUT std_logic_vector(uir_width-1 DOWNTO 0));
END ENTITY;
```

```
ARCHITECTURE behav OF rom_even IS
BEGIN
rom_data_access: PROCESS(ADDR)
BEGIN
        CASE ADDR IS
                WHEN IDLE_LD_R => DATA <= '1' & IDLE & '0' & CW_IDLE;
                WHEN S_M1 => DATA <= '0' & S_M3 & '0' & CW_SM1;
                WHEN S_M3 => DATA <= '0' & S_M2 & '0' & CW_SM3;
                WHEN S_M2 => DATA <= '0' & S_M4 & '0' & CW_SM2;
                WHEN S_M4 => DATA <= '0' & S_M5 & '0' & CW_SM4;
                WHEN S_M5 => DATA <= '1' & S_M6 & '0' & CW_SM5;
                WHEN X_M6 => DATA <= '0' & S_S5 & '0' & CW_SM6;
                WHEN X_S5 => DATA <= '0' & S_S6 & '0' & CW_SS5;
                WHEN X_S6 => DATA <= '0' & S_RND_BR & '0' & CW_SS6;
                WHEN X_RND_BR => DATA <= '0' & S_SND_R & '0' & CW_SRND_BR;
                WHEN X_SND_R => DATA <= '0' & S_SND_I & '0' & CW_SSND_R;
                WHEN S_SND_I => DATA <= '0' & IDLE & '0' & CW_SSND_I;
                WHEN C_SND_I => DATA <= '1' & S_M6 & '0' & CW_CSND_I;
                WHEN OTHERS => DATA <= '1' & IDLE & '0' & CW_IDLE;
        END CASE;
END PROCESS;
END ARCHITECTURE;
```

A.1.10 rom_odd.vhd

```
-- Author:        Luca Lombardini
-- Academic_y:    2020/2021
-- Purpose:       (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:      s277972@studenti.polito.it
--                lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;
USE work.cupack.all;

ENTITY rom_odd IS
        PORT(   ADDR    : IN std_logic_vector(base_addr-1 DOWNTO 0);
                DATA    : OUT std_logic_vector(uir_width-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE behav OF rom_odd IS
BEGIN
rom_data_access: PROCESS(ADDR)
BEGIN
        CASE ADDR IS
                WHEN IDLE_LD_R => DATA <= '0' & S_M1 & '1' & CW_SLD_R;
                WHEN S_M1 => DATA <= '0' & S_M3 & '1' & CW_SM1;
                WHEN S_M3 => DATA <= '0' & S_M2 & '1' & CW_SM3;
                WHEN S_M2 => DATA <= '0' & S_M4 & '1' & CW_SM2;
                WHEN S_M4 => DATA <= '0' & S_M5 & '1' & CW_SM4;
                WHEN S_M5 => DATA <= '1' & C_M6 & '1' & CW_SM5;
                WHEN X_M6 => DATA <= '0' & C_S5 & '1' & CW_CM6;
                WHEN X_S5 => DATA <= '0' & C_S6 & '1' & CW_CS5;
                WHEN X_S6 => DATA <= '0' & C_RND_BR & '1' & CW_CS6;
```

```
                WHEN X_RND_BR => DATA <= '0' & C_SND_R & '1' & CW_CRND_BR;
                WHEN X_SND_R => DATA <= '0' & C_SND_I & '1' & CW_CSND_R;
                WHEN S_SND_I => DATA <= '0' & IDLE & '1' & CW_SSND_I;
                WHEN C_SND_I => DATA <= '1' & C_M6 & '1' & CW_CSND_I;
                WHEN OTHERS => DATA <= '1' & IDLE & '0' & CW_IDLE;
        END CASE;
END PROCESS;
END ARCHITECTURE;
```

A.1.11 reg_n.vhd

```
-- Author:        Luca Lombardini
-- Academic_y:    2020/2021
-- Purpose:       (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:      s277972@studenti.polito.it
--                lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;

ENTITY reg_n IS
        GENERIC(bitwidth: positive        := 8);
        PORT(   CLK     : IN std_logic;
                RST_n   : IN std_logic;
                LD      : IN std_logic;
                D_IN    : IN std_logic_vector(bitwidth-1 DOWNTO 0);
                D_OUT   : OUT std_logic_vector(bitwidth-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE behav OF reg_n IS
BEGIN
        reg_define: PROCESS(RST_n, CLK)
        BEGIN
                IF RST_n = '0' THEN
                        D_OUT <= (OTHERS => '0');
                ELSIF CLK'EVENT AND CLK = '0' THEN
                        IF LD = '1'THEN
                                D_OUT <= D_IN;
                        END IF;
                END IF;
        END PROCESS;
END ARCHITECTURE;
```

A.1.12 late_status_pla.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;

ENTITY late_status_pla IS
        PORT(   STATUS  : IN std_logic;
                LSB     : IN std_logic;
                CC      : IN std_logic;
                LSB_out : OUT std_logic;
                CC_val  : OUT std_logic);
END ENTITY;

ARCHITECTURE behav OF late_status_pla IS
BEGIN
        -- jump when ATTENTION and START asserted
        CC_val <= CC AND STATUS;
        -- if the state is dormant, remain in the same region, otherwise, STATUS tells which region is
        LSB_out<= ( LSB AND NOT(CC) ) OR ( STATUS AND CC );
END ARCHITECTURE;
```

A.1.13 dp_butterfly.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;
USE work.cupack.all;

ENTITY dp_butterfly IS
        PORT(   CLK             : IN std_logic;
                RST_n           : IN std_logic;
                PORT_A          : IN signed(io_width-1 DOWNTO 0);
                PORT_B          : IN signed(io_width-1 DOWNTO 0);
                COEFF_IN        : IN signed(io_width-1 DOWNTO 0);
                CTRL_WORD       : IN std_logic_vector(command_len-1 DOWNTO 0);
                OUT_A           : OUT signed(io_width-1 DOWNTO 0);
                OUT_B           : OUT signed(io_width-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE struct OF dp_butterfly IS

COMPONENT register_file IS
        GENERIC(addr_width      : positive := 4;
                data_width      : positive := 8;
```

```
                wr_ports        : positive := 1;
                rd_ports        : positive := 1);
        PORT(   CLK             : IN std_logic;
                WR              : IN std_logic_vector(wr_ports-1 DOWNTO 0);
                WR_ADDR         : IN unsigned(wr_ports*addr_width-1 DOWNTO 0);
                WR_DATA         : IN signed(wr_ports*data_width-1 DOWNTO 0);
                RD_ADDR         : IN unsigned(rd_ports*addr_width-1 DOWNTO 0);
                RD_DATA         : OUT signed(rd_ports*data_width-1 DOWNTO 0));
END COMPONENT;

COMPONENT multiplier IS
        PORT(   CLK             : IN std_logic;
                A               : IN signed(m_in_width-1 DOWNTO 0);
                B               : IN signed(m_in_width-1 DOWNTO 0);
                D_M_n           : IN std_logic;
                PROD            : OUT signed(prod_width-1 DOWNTO 0));
END COMPONENT;

COMPONENT adder IS
        PORT(   CLK             : IN std_logic;
                A               : IN signed(add_width-1 DOWNTO 0);
                B               : IN signed(add_width-1 DOWNTO 0);
                S_A_n           : IN std_logic;
                SUM             : OUT signed(add_width-1 DOWNTO 0));
END COMPONENT;

COMPONENT rounder IS
        PORT(   ROUND_IN        : IN signed(round_i_width-1 DOWNTO 0);
                ROUND_OUT       : OUT signed(round_o_width-1 DOWNTO 0));
END COMPONENT;

COMPONENT reg IS
        GENERIC(bitwidth: positive := 8);
        PORT(   CLK             : IN std_logic;
                    RST_n           : IN std_logic;
                LD              : IN std_logic;
                D_IN            : IN signed(bitwidth-1 DOWNTO 0);
                D_OUT           : OUT signed(bitwidth-1 DOWNTO 0));
END COMPONENT;

COMPONENT mux2to1 IS
        GENERIC(bitwidth: positive := 8);
        PORT(   A               : IN signed(bitwidth-1 DOWNTO 0);
                B               : IN signed(bitwidth-1 DOWNTO 0);
                SEL             : IN std_logic;
                Y               : OUT signed(bitwidth-1 DOWNTO 0));
END COMPONENT;

--#############################################################################
--# Buses
SIGNAL databus1    : signed(io_width-1 DOWNTO 0);
SIGNAL databus2    : signed(io_width-1 DOWNTO 0);
SIGNAL coeffbus    : signed(io_width-1 DOWNTO 0);
--#############################################################################
--# Local connections
```

```
SIGNAL mult_line   : signed(prod_width-1 DOWNTO 0);
SIGNAL add1_line   : signed(add_width-1 DOWNTO 0);
SIGNAL add2_line   : signed(add_width-1 DOWNTO 0);
--#############################################################################
--# Temporary signals for port connection
SIGNAL rfd_addr_wr : unsigned(rfd_wr_ports*rfd_addr_width-1 DOWNTO 0);
SIGNAL rfd_addr_rd : unsigned(rfd_rd_ports*rfd_addr_width-1 DOWNTO 0);
SIGNAL rfd_wr_bits : std_logic_vector(rfd_wr_ports-1 DOWNTO 0);
SIGNAL aggr_in     : signed(rfd_wr_ports*io_width-1 DOWNTO 0);
SIGNAL bus_concat  : signed(rfd_rd_ports*io_width-1 DOWNTO 0);
SIGNAL mult_out    : signed(prod_width-1 DOWNTO 0);
SIGNAL bus2_allign : signed(add_width-1 DOWNTO 0);
SIGNAL chosen_src  : signed(add_width-1 DOWNTO 0);
SIGNAL add1_portA  : signed(add_width-1 DOWNTO 0);
SIGNAL add1_portB  : signed(add_width-1 DOWNTO 0);
SIGNAL add1_out    : signed(add_width-1 DOWNTO 0);
SIGNAL add2_portB  : signed(add_width-1 DOWNTO 0);
SIGNAL add2_out    : signed(add_width-1 DOWNTO 0);
SIGNAL rndr_in     : signed(add_width-1 DOWNTO 0);
SIGNAL rndr_out    : signed(io_width-1 DOWNTO 0);
SIGNAL reord_out   : signed(io_width-1 DOWNTO 0);

BEGIN
--#############################################################################
--#        Register File for the input Data
        data_reg_file         : register_file
                                GENERIC MAP(    rfd_addr_width,
                                                rfd_data_width,
                                                rfd_wr_ports,
                                                rfd_rd_ports)
                                PORT MAP(       CLK,
                                                rfd_wr_bits,
                                                rfd_addr_wr,
                                                aggr_in,
                                                rfd_addr_rd,
                                                bus_concat);
        aggr_in(2*io_width-1 DOWNTO io_width) <= PORT_B;
        aggr_in(io_width-1 DOWNTO 0)          <= PORT_A;
        databus1 <= bus_concat(io_width-1 DOWNTO 0);
        databus2 <= bus_concat(2*io_width-1 DOWNTO io_width);
        rfd_wr_bits <= CTRL_WORD(RFD_WR2) & CTRL_WORD(RFD_WR1);
        rfd_addr_wr <= CTRL_WORD(RFD_WR2_ADDR1) & CTRL_WORD(RFD_WR2_ADDR0) & CTRL_WORD(RFD_WR1_ADDR1) &
        rfd_addr_rd <= CTRL_WORD(RFD_RD2_ADDR1) & CTRL_WORD(RFD_RD2_ADDR0) & CTRL_WORD(RFD_RD1_ADDR1) &
--#############################################################################
--#        Register File for the coefficients
        coef_reg_file         : register_file
                                GENERIC MAP(    rfc_addr_width,
                                                rfc_data_width,
                                                rfc_wr_ports,
                                                rfc_rd_ports)
                                PORT MAP(       CLK,
                                                CTRL_WORD(RFC_WR DOWNTO RFC_WR),
                                                unsigned(CTRL_WORD(RFC_WR_ADDR DOWNTO RFC_WR_ADDR)),
                                                COEFF_IN,
                                                unsigned(CTRL_WORD(RFC_RD_ADDR DOWNTO RFC_RD_ADDR)),
```

```
                                                coeffbus);
--#############################################################################
--#        Multiplier
        multiplier_op           : multiplier
                                  PORT MAP(       CLK,
                                                  databus1,
                                                  coeffbus,
                                                  CTRL_WORD(MULT_DOUBLE),
                                                  mult_out);
--#############################################################################
--#        Multiplier Result Buffer
        reg_mult                : reg
                                  GENERIC MAP(    prod_width)
                                  PORT MAP(       CLK,
                                                  RST_n,
                                                  CTRL_WORD(R_MULT_LD),
                                                  mult_out,
                                                  mult_line);
--#############################################################################
--#        Adder1 input multiplexers
        mux1_bus2_sum1          : mux2to1
                                  GENERIC MAP(    add_width)
                                  PORT MAP(       add1_line,
                                                  bus2_allign,
                                                  CTRL_WORD(MUX1_SEL),
                                                  chosen_src);
        mux2_prod_mux1          : mux2to1
                                  GENERIC MAP(    add_width)
                                  PORT MAP(       chosen_src,
                                                  mult_line,
                                                  CTRL_WORD(MUX2_SEL),
                                                  add1_portA);
        mux3_sum1_prod          : mux2to1
                                  GENERIC MAP(    add_width)
                                  PORT MAP(       mult_line,
                                                  add1_line,
                                                  CTRL_WORD(MUX3_SEL),
                                                  add1_portB);
        -- duplicate sign and extend to 40 bits adding 19 trailing zeros
        bus2_allign <=  signed(databus2(io_width-1) & std_logic_vector(databus2) &
                        std_logic_vector(to_unsigned(0, io_width-1)));
--#############################################################################
--#        Adder1
        adder1                  : adder
                                  PORT MAP(       CLK,
                                                  add1_portA,
                                                  add1_portB,
                                                  CTRL_WORD(ADD1_SUB_ADD),
                                                  add1_out);
--#############################################################################
--#        Adder1 Result Buffer
        reg_add1                : reg
                                  GENERIC MAP(    add_width)
                                  PORT MAP(       CLK,
                                                  RST_n,
```

```
                                                CTRL_WORD(R_ADD1_LD),
                                                add1_out,
                                                add1_line);
--#############################################################################
--#        Adder2 input multiplexer
        mux4_bus2_sum2          : mux2to1
                                  GENERIC MAP(    add_width)
                                  PORT MAP(       add2_line,
                                                  bus2_allign,
                                                  CTRL_WORD(MUX4_SEL),
                                                  add2_portB);
--#############################################################################
--#        Adder2
        adder2                  : adder
                                  PORT MAP(       CLK,
                                                  mult_line,
                                                  add2_portB,
                                                  CTRL_WORD(ADD2_SUB_ADD),
                                                  add2_out);
--#############################################################################
--#        Adder2 Result Buffer
        reg_add2                : reg
                                  GENERIC MAP(    add_width)
                                  PORT MAP(       CLK,
                                                  RST_n,
                                                  CTRL_WORD(R_ADD2_LD),
                                                  add2_out,
                                                  add2_line);
--#############################################################################
--#        Rounder input multiplexer
        mux5_round              : mux2to1
                                  GENERIC MAP(    add_width)
                                  PORT MAP(       add1_line,
                                                  add2_line,
                                                  CTRL_WORD(MUX5_SEL),
                                                  rndr_in);
--#############################################################################
--#        Rounder
        round_unit              : rounder
                                  PORT MAP(       rndr_in,
                                                  rndr_out);
--#############################################################################
--#        Reordering
        reorder_reg             : reg
                                  GENERIC MAP(    io_width)
                                  PORT MAP(       CLK,
                                                  RST_n,
                                                  CTRL_WORD(REORD_LD),
                                                  rndr_out,
                                                  reord_out);
--#############################################################################
--#        Output Buffers
        output_A_buffer         : reg
                                  GENERIC MAP(    io_width)
                                  PORT MAP(       CLK,
```

```
                                        RST_n,
                                        CTRL_WORD(OUT_A_BUF_LD),
                                        reord_out,
                                        OUT_A);

        output_B_buffer       : reg
                                GENERIC MAP(    io_width)
                                PORT MAP(       CLK,
                                                RST_n,
                                                CTRL_WORD(OUT_B_BUF_LD),
                                                rndr_out,
                                                OUT_B);
END ARCHITECTURE;
```

A.1.14 cu_butterfly.vhd

```
-- Author:        Luca Lombardini
-- Academic_y:    2020/2021
-- Purpose:       (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:      s277972@studenti.polito.it
--                lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.cupack.all;

ENTITY cu_butterfly IS
        PORT(   CLK     : IN std_logic;
                RST_n   : IN std_logic;
                STATUS  : IN std_logic;
                CTRL_WRD: OUT std_logic_vector(command_len-1 DOWNTO 0));
END ENTITY;


ARCHITECTURE struct OF cu_butterfly IS

        COMPONENT rom_even IS
                PORT(   ADDR    : IN std_logic_vector(base_addr-1 DOWNTO 0);
                        DATA    : OUT std_logic_vector(uir_width-1 DOWNTO 0));
        END COMPONENT;

        COMPONENT rom_odd IS
                PORT(   ADDR    : IN std_logic_vector(base_addr-1 DOWNTO 0);
                        DATA    : OUT std_logic_vector(uir_width-1 DOWNTO 0));
        END COMPONENT;

        COMPONENT late_status_pla IS
                PORT(   STATUS  : IN std_logic;
                        LSB     : IN std_logic;
                        CC      : IN std_logic;
                        LSB_out : OUT std_logic;
                        CC_val  : OUT std_logic);
        END COMPONENT;

        COMPONENT reg IS
```

```
                GENERIC(bitwidth: positive := 8);
                PORT(   CLK     : IN std_logic;
                        RST_n   : IN std_logic;
                        LD      : IN std_logic;
                        D_IN    : IN signed(bitwidth-1 DOWNTO 0);
                        D_OUT   : OUT signed(bitwidth-1 DOWNTO 0));
        END COMPONENT;


        COMPONENT reg_n IS
                GENERIC(bitwidth: positive := 8);
                PORT(   CLK     : IN std_logic;
                        RST_n   : IN std_logic;
                        LD      : IN std_logic;
                        D_IN    : IN std_logic_vector(bitwidth-1 DOWNTO 0);
                        D_OUT   : OUT std_logic_vector(bitwidth-1 DOWNTO 0));
        END COMPONENT;

        COMPONENT mux2to1 IS
                GENERIC(bitwidth: positive := 8);
                PORT(   A       : IN signed(bitwidth-1 DOWNTO 0);
                        B       : IN signed(bitwidth-1 DOWNTO 0);
                        SEL     : IN std_logic;
                        Y       : OUT signed(bitwidth-1 DOWNTO 0));
        END COMPONENT;

        SIGNAL next_address     : signed(base_addr DOWNTO 0); --not conn. yet
        SIGNAL roms_address     : signed(base_addr DOWNTO 0);
        SIGNAL even_out         : std_logic_vector(uir_width-1 DOWNTO 0);
        SIGNAL odd_out          : std_logic_vector(uir_width-1 DOWNTO 0);
        SIGNAL micro_addr_lsb   : std_logic;
        SIGNAL late_sel_bit     : std_logic;
        SIGNAL u_next_reg_in    : signed(cc_lsb_addr-1 DOWNTO 0);
        SIGNAL u_command_reg_in : signed(command_len-1 DOWNTO 0);
        SIGNAL u_instr_in       : std_logic_vector(uir_width-1 DOWNTO 0);
        SIGNAL u_instr_out      : std_logic_vector(uir_width-1 DOWNTO 0);
        SIGNAL future_lsb       : std_logic;
        SIGNAL cc_validation    : std_logic;

BEGIN
--#############################################################################
--#       uAR: contains the actual machine's instruction address
        micro_addr_reg: reg GENERIC MAP(base_addr+1)
                            PORT MAP(CLK,RST_n,
                                    '1',
                                    next_address,
                                    roms_address);
--#############################################################################
--#       uROM for the even addresses
        mirco_rom_even: rom_even PORT MAP(std_logic_vector(roms_address(base_addr DOWNTO 1)),
                                    even_out);
--#############################################################################
--#       uROM for the odd addresses
        micro_rom_odd:  rom_odd PORT MAP(std_logic_vector(roms_address(base_addr DOWNTO 1)),
                                    odd_out);
```

```
--#############################################################################
--#        Mux used to select the next address, driven by the late status
        mux_next_addr:  mux2to1 GENERIC MAP(cc_lsb_addr)
                                PORT MAP(signed(even_out(uir_width-1 DOWNTO command_len)),
                                        signed(odd_out(uir_width-1 DOWNTO command_len)),
                                        late_sel_bit,
                                        u_next_reg_in);
--#############################################################################
--#        Mux used to select the actual command word, driven byt the uAR LSB
        mux_ctrl_wrd :  mux2to1 GENERIC MAP(command_len)
                                PORT MAP(signed(even_out(command_len-1 DOWNTO 0)),
                                        signed(odd_out(command_len-1 DOWNTO 0)),
                                        late_sel_bit,
                                        u_command_reg_in);
--#############################################################################
--#        uIR input composition of selected Next_address and actual command word
        u_instr_in <= std_logic_vector(u_next_reg_in) & std_logic_vector(u_command_reg_in);
--#############################################################################
--#        uIR containing the next address, CC validation and actual command word
        u_instruction: reg_n GENERIC MAP(uir_width)
                                PORT MAP(CLK, RST_n,
                                        '1',
                                        u_instr_in,
                                        u_instr_out);
--#############################################################################
--#        Buffered control word sent to the output
        CTRL_WRD <= u_instr_out(command_len-1 DOWNTO 0);
--#############################################################################
--#        Next address roundtrip completion to uAR input, w/out CC and LSB
        next_address <= signed(u_instr_out(uir_width-2 DOWNTO command_len+1) & future_lsb );
--#############################################################################
--#        Late status PLA used to combine LSB and STATUS when CC enabled
        late_status:        late_status_pla PORT MAP(STATUS,
                                        u_instr_out(command_len),
                                        u_instr_out(uir_width-1),
                                        future_lsb,
                                        cc_validation);
--#############################################################################
--#        uAR buffered LSB pickup
        micro_addr_lsb <= roms_address(0);
--#############################################################################
--#        Selection signal generation for the Next Address mux (implicit mux)
        late_sel_bit <= future_lsb WHEN cc_validation = '1' ELSE micro_addr_lsb;
        -- inplace single bit multiplexer, or else too much conversion needed
END ARCHITECTURE;
```

A.1.15 butterfly.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
```

```
USE IEEE.numeric_std.all;
USE work.definespack.all;
USE work.cupack.all;

ENTITY butterfly IS
        PORT(   CLK     : IN std_logic;
                RST_n   : IN std_logic;
                START   : IN std_logic;
                PORT_A  : IN signed(io_width-1 DOWNTO 0);
                PORT_B  : IN signed(io_width-1 DOWNTO 0);
                COEFF_IN: IN signed(io_width-1 DOWNTO 0);
                DONE    : OUT std_logic;
                OUT_A   : OUT signed(io_width-1 DOWNTO 0);
                OUT_B   : OUT signed(io_width-1 DOWNTO 0));
END ENTITY;


ARCHITECTURE struct OF butterfly IS

        COMPONENT dp_butterfly IS
                PORT(   CLK     : IN std_logic;
                        RST_n   : IN std_logic;
                        PORT_A  : IN signed(io_width-1 DOWNTO 0);
                        PORT_B  : IN signed(io_width-1 DOWNTO 0);
                        COEFF_IN: IN signed(io_width-1 DOWNTO 0);
                        CTRL_WORD:IN std_logic_vector(command_len-1 DOWNTO 0);
                        OUT_A   : OUT signed(io_width-1 DOWNTO 0);
                        OUT_B   : OUT signed(io_width-1 DOWNTO 0));
        END COMPONENT;

        COMPONENT cu_butterfly IS
                PORT(   CLK     : IN std_logic;
                        RST_n   : IN std_logic;
                        STATUS  : IN std_logic;
                        CTRL_WRD: OUT std_logic_vector(command_len-1 DOWNTO 0));
        END COMPONENT;

--#############################################################################
--#       Signal used to interface Control Unit with Data Path
        SIGNAL cw_to_datapath   : std_logic_vector(command_len-1 DOWNTO 0);

BEGIN
--#############################################################################
--#       Datapath section of butterfly unit
        datapath : dp_butterfly PORT MAP(CLK,
                                         RST_n,
                                         PORT_A,
                                         PORT_B,
                                         COEFF_IN,
                                         cw_to_datapath,
                                         OUT_A,
                                         OUT_B);
--#############################################################################
--#       Control Unit Section of butterfly unit
        control_unit : cu_butterfly PORT MAP(CLK,
```

```
                                        RST_n,
                                        START,
                                        cw_to_datapath);
--#############################################################################
--#        Done bit connection
        DONE <= cw_to_datapath(DONE_BIT);
END ARCHITECTURE;
```

A.1.16 pipe_machine.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.cupack.all;
USE work.definespack.all;

ENTITY pipe_machine IS
        PORT(   CLK     : IN std_logic;
                RST_n   : IN std_logic;
                START   : IN std_logic;
                DONE    : IN std_logic;
                W_IN    : IN signed(io_width*n_coef-1 DOWNTO 0);
                W_OUT   : OUT signed(io_width*n_coef-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE behav OF pipe_machine IS

        COMPONENT reg IS
                GENERIC(bitwidth: positive        := 8);
                PORT(   CLK     : IN std_logic;
                            RST_n   : IN std_logic;
                        LD      : IN std_logic;
                        D_IN    : IN signed(bitwidth-1 DOWNTO 0);
                        D_OUT   : OUT signed(bitwidth-1 DOWNTO 0));
        END COMPONENT;

        TYPE PIPE_ARRAY IS ARRAY(0 TO 11) OF signed(io_width*n_coef-1 DOWNTO 0); --12 in-between sigs
        SIGNAL pipe : PIPE_ARRAY;

BEGIN
        pipeline: FOR i IN 0 TO 10 GENERATE --11 regs, too much problems encountered with behavioral FS
                pipeReg_i: reg GENERIC MAP(io_width*n_coef)
                                PORT MAP(CLK, RST_n, '1', pipe(i), pipe(i+1));
        END GENERATE;

        pipe(0) <= W_IN;
        W_OUT <= pipe(11);
END ARCHITECTURE;
```

A.1.17 fft_unit.vhd

```
-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;
USE work.cupack.all;

ENTITY fft_unit IS
        PORT(   CLK     : IN std_logic;
                RST_n   : IN std_logic;
                START   : IN std_logic;
                DATA_IN : IN signed(io_width*n_samples-1 DOWNTO 0);
                COEF_IN : IN signed(io_width*n_coef-1 DOWNTO 0);
                DATA_OUT: OUT signed(io_width*n_samples-1 DOWNTO 0);
                DONE    : OUT std_logic);
END ENTITY;

ARCHITECTURE struct OF fft_unit IS

        COMPONENT pipe_machine IS
                PORT(   CLK     : IN std_logic;
                        RST_n   : IN std_logic;
                        START   : IN std_logic;
                        DONE    : IN std_logic;
                        W_IN    : IN signed(io_width*n_coef-1 DOWNTO 0);
                        W_OUT   : OUT signed(io_width*n_coef-1 DOWNTO 0));
        END COMPONENT;

        COMPONENT butterfly IS
                PORT(   CLK     : IN std_logic;
                        RST_n   : IN std_logic;
                        START   : IN std_logic;
                        PORT_A  : IN signed(io_width-1 DOWNTO 0);
                        PORT_B  : IN signed(io_width-1 DOWNTO 0);
                        COEFF_IN: IN signed(io_width-1 DOWNTO 0);
                        DONE    : OUT std_logic;
                        OUT_A   : OUT signed(io_width-1 DOWNTO 0);
                        OUT_B   : OUT signed(io_width-1 DOWNTO 0));
        END COMPONENT;
--#############################################################################
--#       Signal Zone
        SIGNAL lv0_in0 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv0_in8 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv0_out0 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv0_out8 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv0_gr0_done0 : std_logic;
        SIGNAL lv0_in1 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv0_in9 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv0_out1 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv0_out9 : signed(io_width-1 DOWNTO 0);
```

```
SIGNAL lv0_gr0_done1 : std_logic;
SIGNAL lv0_in2 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_in10 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out2 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out10 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_gr0_done2 : std_logic;
SIGNAL lv0_in3 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_in11 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out3 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out11 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_gr0_done3 : std_logic;
SIGNAL lv0_in4 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_in12 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out4 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out12 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_gr0_done4 : std_logic;
SIGNAL lv0_in5 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_in13 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out5 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out13 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_gr0_done5 : std_logic;
SIGNAL lv0_in6 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_in14 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out6 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out14 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_gr0_done6 : std_logic;
SIGNAL lv0_in7 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_in15 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out7 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_out15 : signed(io_width-1 DOWNTO 0);
SIGNAL lv0_gr0_done7 : std_logic;
SIGNAL lev0_w0 : signed(io_width-1 DOWNTO 0);
SIGNAL doneAggr0 : std_logic;
SIGNAL w_pipe0 : signed(io_width*n_coef-1 DOWNTO 0);
SIGNAL lv1_out0 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_out4 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_gr0_done0 : std_logic;
SIGNAL lv1_out1 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_out5 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_gr0_done1 : std_logic;
SIGNAL lv1_out2 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_out6 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_gr0_done2 : std_logic;
SIGNAL lv1_out3 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_out7 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_gr0_done3 : std_logic;
SIGNAL lev1_w0 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_out8 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_out12 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_gr1_done0 : std_logic;
SIGNAL lv1_out9 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_out13 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_gr1_done1 : std_logic;
SIGNAL lv1_out10 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_out14 : signed(io_width-1 DOWNTO 0);
```

```
SIGNAL lv1_gr1_done2 : std_logic;
SIGNAL lv1_out11 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_out15 : signed(io_width-1 DOWNTO 0);
SIGNAL lv1_gr1_done3 : std_logic;
SIGNAL lev1_w4 : signed(io_width-1 DOWNTO 0);
SIGNAL doneAggr1 : std_logic;
SIGNAL w_pipe1 : signed(io_width*n_coef-1 DOWNTO 0);
SIGNAL lv2_out0 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out2 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_gr0_done0 : std_logic;
SIGNAL lv2_out1 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out3 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_gr0_done1 : std_logic;
SIGNAL lev2_w0 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out4 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out6 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_gr1_done0 : std_logic;
SIGNAL lv2_out5 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out7 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_gr1_done1 : std_logic;
SIGNAL lev2_w4 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out8 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out10 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_gr2_done0 : std_logic;
SIGNAL lv2_out9 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out11 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_gr2_done1 : std_logic;
SIGNAL lev2_w2 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out12 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out14 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_gr3_done0 : std_logic;
SIGNAL lv2_out13 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_out15 : signed(io_width-1 DOWNTO 0);
SIGNAL lv2_gr3_done1 : std_logic;
SIGNAL lev2_w6 : signed(io_width-1 DOWNTO 0);
SIGNAL doneAggr2 : std_logic;
SIGNAL w_pipe2 : signed(io_width*n_coef-1 DOWNTO 0);
SIGNAL lv3_out0 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_out1 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_gr0_done0 : std_logic;
SIGNAL lev3_w0 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_out2 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_out3 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_gr1_done0 : std_logic;
SIGNAL lev3_w4 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_out4 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_out5 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_gr2_done0 : std_logic;
SIGNAL lev3_w2 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_out6 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_out7 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_gr3_done0 : std_logic;
SIGNAL lev3_w6 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_out8 : signed(io_width-1 DOWNTO 0);
SIGNAL lv3_out9 : signed(io_width-1 DOWNTO 0);
```

```
        SIGNAL lv3_gr4_done0 : std_logic;
        SIGNAL lev3_w1 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv3_out10 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv3_out11 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv3_gr5_done0 : std_logic;
        SIGNAL lev3_w5 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv3_out12 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv3_out13 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv3_gr6_done0 : std_logic;
        SIGNAL lev3_w3 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv3_out14 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv3_out15 : signed(io_width-1 DOWNTO 0);
        SIGNAL lv3_gr7_done0 : std_logic;
        SIGNAL lev3_w7 : signed(io_width-1 DOWNTO 0);
BEGIN
--#############################################################################
--#        1st Layer
        lv0_gr0_but0 : butterfly PORT MAP (CLK, RST_n, START, lv0_in0, lv0_in8, lev0_w0,
                                           lv0_gr0_done0, lv0_out0, lv0_out8);
        lv0_gr0_but1 : butterfly PORT MAP (CLK, RST_n, START, lv0_in1, lv0_in9, lev0_w0,
                                           lv0_gr0_done1, lv0_out1, lv0_out9);
        lv0_gr0_but2 : butterfly PORT MAP (CLK, RST_n, START, lv0_in2, lv0_in10, lev0_w0,
                                           lv0_gr0_done2, lv0_out2, lv0_out10);
        lv0_gr0_but3 : butterfly PORT MAP (CLK, RST_n, START, lv0_in3, lv0_in11, lev0_w0,
                                           lv0_gr0_done3, lv0_out3, lv0_out11);
        lv0_gr0_but4 : butterfly PORT MAP (CLK, RST_n, START, lv0_in4, lv0_in12, lev0_w0,
                                           lv0_gr0_done4, lv0_out4, lv0_out12);
        lv0_gr0_but5 : butterfly PORT MAP (CLK, RST_n, START, lv0_in5, lv0_in13, lev0_w0,
                                           lv0_gr0_done5, lv0_out5, lv0_out13);
        lv0_gr0_but6 : butterfly PORT MAP (CLK, RST_n, START, lv0_in6, lv0_in14, lev0_w0,
                                           lv0_gr0_done6, lv0_out6, lv0_out14);
        lv0_gr0_but7 : butterfly PORT MAP (CLK, RST_n, START, lv0_in7, lv0_in15, lev0_w0,
                                           lv0_gr0_done7, lv0_out7, lv0_out15);
        pipe0 : pipe_machine PORT MAP(CLK, RST_n, START, doneAggr0, COEF_IN, w_pipe0);
        lv0_in0 <= DATA_IN(io_width*16-1 DOWNTO io_width*15);
        lv0_in8 <= DATA_IN(io_width*8-1 DOWNTO io_width*7);
        lv0_in1 <= DATA_IN(io_width*15-1 DOWNTO io_width*14);
        lv0_in9 <= DATA_IN(io_width*7-1 DOWNTO io_width*6);
        lv0_in2 <= DATA_IN(io_width*14-1 DOWNTO io_width*13);
        lv0_in10 <= DATA_IN(io_width*6-1 DOWNTO io_width*5);
        lv0_in3 <= DATA_IN(io_width*13-1 DOWNTO io_width*12);
        lv0_in11 <= DATA_IN(io_width*5-1 DOWNTO io_width*4);
        lv0_in4 <= DATA_IN(io_width*12-1 DOWNTO io_width*11);
        lv0_in12 <= DATA_IN(io_width*4-1 DOWNTO io_width*3);
        lv0_in5 <= DATA_IN(io_width*11-1 DOWNTO io_width*10);
        lv0_in13 <= DATA_IN(io_width*3-1 DOWNTO io_width*2);
        lv0_in6 <= DATA_IN(io_width*10-1 DOWNTO io_width*9);
        lv0_in14 <= DATA_IN(io_width*2-1 DOWNTO io_width*1);
        lv0_in7 <= DATA_IN(io_width*9-1 DOWNTO io_width*8);
        lv0_in15 <= DATA_IN(io_width*1-1 DOWNTO io_width*0);
        lev0_w0 <= COEF_IN(io_width*8-1 DOWNTO io_width*7);
        doneAggr0 <= lv0_gr0_done0 AND lv0_gr0_done1 AND lv0_gr0_done2 AND lv0_gr0_done3 AND
                     lv0_gr0_done4 AND lv0_gr0_done5 AND lv0_gr0_done6 AND lv0_gr0_done7;
--#############################################################################
--#        2nd Layer
```

```
        lv1_gr0_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr0, lv0_out0, lv0_out4, lev1_w0,
                                           lv1_gr0_done0, lv1_out0, lv1_out4);
        lv1_gr0_but1 : butterfly PORT MAP (CLK, RST_n, doneAggr0, lv0_out1, lv0_out5, lev1_w0,
                                           lv1_gr0_done1, lv1_out1, lv1_out5);
        lv1_gr0_but2 : butterfly PORT MAP (CLK, RST_n, doneAggr0, lv0_out2, lv0_out6, lev1_w0,
                                           lv1_gr0_done2, lv1_out2, lv1_out6);
        lv1_gr0_but3 : butterfly PORT MAP (CLK, RST_n, doneAggr0, lv0_out3, lv0_out7, lev1_w0,
                                           lv1_gr0_done3, lv1_out3, lv1_out7);
        lv1_gr1_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr0, lv0_out8, lv0_out12, lev1_w4,
                                           lv1_gr1_done0, lv1_out8, lv1_out12);
        lv1_gr1_but1 : butterfly PORT MAP (CLK, RST_n, doneAggr0, lv0_out9, lv0_out13, lev1_w4,
                                           lv1_gr1_done1, lv1_out9, lv1_out13);
        lv1_gr1_but2 : butterfly PORT MAP (CLK, RST_n, doneAggr0, lv0_out10, lv0_out14, lev1_w4,
                                           lv1_gr1_done2, lv1_out10, lv1_out14);
        lv1_gr1_but3 : butterfly PORT MAP (CLK, RST_n, doneAggr0, lv0_out11, lv0_out15, lev1_w4,
                                           lv1_gr1_done3, lv1_out11, lv1_out15);
        pipe1 : pipe_machine PORT MAP(CLK, RST_n, doneAggr0, doneAggr1, w_pipe0, w_pipe1);
        lev1_w0 <= w_pipe0(io_width*8-1 DOWNTO io_width*7);
        lev1_w4 <= w_pipe0(io_width*4-1 DOWNTO io_width*3);
        doneAggr1 <= lv1_gr0_done0 AND lv1_gr0_done1 AND lv1_gr0_done2 AND lv1_gr0_done3 AND
                     lv1_gr1_done0 AND lv1_gr1_done1 AND lv1_gr1_done2 AND lv1_gr1_done3;
--#############################################################################
--#        3rd Layer
        lv2_gr0_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr1, lv1_out0, lv1_out2, lev2_w0,
                                           lv2_gr0_done0, lv2_out0, lv2_out2);
        lv2_gr0_but1 : butterfly PORT MAP (CLK, RST_n, doneAggr1, lv1_out1, lv1_out3, lev2_w0,
                                           lv2_gr0_done1, lv2_out1, lv2_out3);
        lv2_gr1_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr1, lv1_out4, lv1_out6, lev2_w4,
                                           lv2_gr1_done0, lv2_out4, lv2_out6);
        lv2_gr1_but1 : butterfly PORT MAP (CLK, RST_n, doneAggr1, lv1_out5, lv1_out7, lev2_w4,
                                           lv2_gr1_done1, lv2_out5, lv2_out7);
        lv2_gr2_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr1, lv1_out8, lv1_out10, lev2_w2,
                                           lv2_gr2_done0, lv2_out8, lv2_out10);
        lv2_gr2_but1 : butterfly PORT MAP (CLK, RST_n, doneAggr1, lv1_out9, lv1_out11, lev2_w2,
                                           lv2_gr2_done1, lv2_out9, lv2_out11);
        lv2_gr3_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr1, lv1_out12, lv1_out14, lev2_w6,
                                           lv2_gr3_done0, lv2_out12, lv2_out14);
        lv2_gr3_but1 : butterfly PORT MAP (CLK, RST_n, doneAggr1, lv1_out13, lv1_out15, lev2_w6,
                                           lv2_gr3_done1, lv2_out13, lv2_out15);
        pipe2 : pipe_machine PORT MAP(CLK, RST_n, doneAggr1, doneAggr2, w_pipe1, w_pipe2);
        lev2_w0 <= w_pipe1(io_width*8-1 DOWNTO io_width*7);
        lev2_w4 <= w_pipe1(io_width*4-1 DOWNTO io_width*3);
        lev2_w2 <= w_pipe1(io_width*6-1 DOWNTO io_width*5);
        lev2_w6 <= w_pipe1(io_width*2-1 DOWNTO io_width*1);
        doneAggr2 <= lv2_gr0_done0 AND lv2_gr0_done1 AND lv2_gr1_done0 AND lv2_gr1_done1 AND
                     lv2_gr2_done0 AND lv2_gr2_done1 AND lv2_gr3_done0 AND lv2_gr3_done1;
--#############################################################################
--#        4th Layer
        lv3_gr0_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr2, lv2_out0, lv2_out1, lev3_w0,
                                           lv3_gr0_done0, lv3_out0, lv3_out1);
        lv3_gr1_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr2, lv2_out2, lv2_out3, lev3_w4,
                                           lv3_gr1_done0, lv3_out2, lv3_out3);
        lv3_gr2_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr2, lv2_out4, lv2_out5, lev3_w2,
                                           lv3_gr2_done0, lv3_out4, lv3_out5);
        lv3_gr3_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr2, lv2_out6, lv2_out7, lev3_w6,
```

```
                                        lv3_gr3_done0, lv3_out6, lv3_out7);
        lv3_gr4_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr2, lv2_out8, lv2_out9, lev3_w1,
                                        lv3_gr4_done0, lv3_out8, lv3_out9);
        lv3_gr5_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr2, lv2_out10, lv2_out11, lev3_w5,
                                        lv3_gr5_done0, lv3_out10, lv3_out11);
        lv3_gr6_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr2, lv2_out12, lv2_out13, lev3_w3,
                                        lv3_gr6_done0, lv3_out12, lv3_out13);
        lv3_gr7_but0 : butterfly PORT MAP (CLK, RST_n, doneAggr2, lv2_out14, lv2_out15, lev3_w7,
                                        lv3_gr7_done0, lv3_out14, lv3_out15);
        lev3_w0 <= w_pipe2(io_width*8-1 DOWNTO io_width*7);
        lev3_w4 <= w_pipe2(io_width*4-1 DOWNTO io_width*3);
        lev3_w2 <= w_pipe2(io_width*6-1 DOWNTO io_width*5);
        lev3_w6 <= w_pipe2(io_width*2-1 DOWNTO io_width*1);
        lev3_w1 <= w_pipe2(io_width*7-1 DOWNTO io_width*6);
        lev3_w5 <= w_pipe2(io_width*3-1 DOWNTO io_width*2);
        lev3_w3 <= w_pipe2(io_width*5-1 DOWNTO io_width*4);
        lev3_w7 <= w_pipe2(io_width*1-1 DOWNTO io_width*0);
        DONE <= lv3_gr0_done0 AND lv3_gr1_done0 AND lv3_gr2_done0 AND lv3_gr3_done0 AND
                lv3_gr4_done0 AND lv3_gr5_done0 AND lv3_gr6_done0 AND lv3_gr7_done0;
--#############################################################################
--#        Output reorder
        DATA_OUT(io_width*16-1 DOWNTO io_width*15) <= lv3_out0;
        DATA_OUT(io_width*15-1 DOWNTO io_width*14) <= lv3_out8;
        DATA_OUT(io_width*14-1 DOWNTO io_width*13) <= lv3_out4;
        DATA_OUT(io_width*13-1 DOWNTO io_width*12) <= lv3_out12;
        DATA_OUT(io_width*12-1 DOWNTO io_width*11) <= lv3_out2;
        DATA_OUT(io_width*11-1 DOWNTO io_width*10) <= lv3_out10;
        DATA_OUT(io_width*10-1 DOWNTO io_width*9) <= lv3_out6;
        DATA_OUT(io_width*9-1 DOWNTO io_width*8) <= lv3_out14;
        DATA_OUT(io_width*8-1 DOWNTO io_width*7) <= lv3_out1;
        DATA_OUT(io_width*7-1 DOWNTO io_width*6) <= lv3_out9;
        DATA_OUT(io_width*6-1 DOWNTO io_width*5) <= lv3_out5;
        DATA_OUT(io_width*5-1 DOWNTO io_width*4) <= lv3_out13;
        DATA_OUT(io_width*4-1 DOWNTO io_width*3) <= lv3_out3;
        DATA_OUT(io_width*3-1 DOWNTO io_width*2) <= lv3_out11;
        DATA_OUT(io_width*2-1 DOWNTO io_width*1) <= lv3_out7;
        DATA_OUT(io_width*1-1 DOWNTO io_width*0) <= lv3_out15;
END ARCHITECTURE;
```

A.2 Extra: C files

A.2.1 butterfly.h

```
/* Author:        Luca Lombardini
 * Academic_y:        2020/2021
 * Purpose:        (Master Degree) Digital Integrated Systems' Final Project
 * Contacts:        s277972@studenti.polito.it
 *                  lombamari2@gmail.com
 */
#ifndef __BUTTERFLY__
#define __BUTTERFLY__

int _butterfly(unsigned int* A_r_in, unsigned int* A_i_in, unsigned int* B_r_in,
unsigned int* B_i_in, unsigned int* w_r, unsigned int* w_i, unsigned int* A_r_out,
unsigned int* A_i_out, unsigned int* B_r_out, unsigned int* B_i_out);
```

```
signed long int _signer(unsigned long int number);

unsigned int _round_and_scale(signed long int number);

#endif
```

A.2.2 butterfly.c

```
/* Author:          Luca Lombardini
 * Academic_y:          2020/2021
 * Purpose:          (Master Degree) Digital Integrated Systems' Final Project
 * Contacts:          s277972@studenti.polito.it
 *                   lombamari2@gmail.com
 */
#ifndef __BUTTERFLY__
#include "butterfly.h"
#endif

#ifndef BIT_LENGTH
#pragma message("BIT_LENGTH definition not found")
#define BIT_LENGTH 20
#endif

/* ALLIGNMENT MOTIVATION
 *
 * mx = |m39|m38|.|m37|...|m0|
 *
 * ab = |b19|.|b18|...|b0|        -> The allignment must be based on dec. point!
 *
 * |m39|m38|.|m37|...|m19| |m18|...|m0|  _\ |m39|m38|.|m37|...|m19|...|m0|
 *                   |b19|.|b18|...|b0|   / |b19|b19|.|b18|...|b0|
 *                                                              |____\ delta=19
 * Note: sign extention automatically performed since a/b coeff. are casted to
 *         signed
 */

int _butterfly(unsigned int* A_r_in, unsigned int* A_i_in, unsigned int* B_r_in,
unsigned int* B_i_in, unsigned int* w_r, unsigned int* w_i, unsigned int* A_r_out,
unsigned int* A_i_out, unsigned int* B_r_out, unsigned int* B_i_out) {
        signed long int m1, m2, m3, m4, m5, m6;
        signed long int s1, s2, s3, s4, s5, s6;
        signed long int _a_r, _a_i, _b_r, _b_i, _w_r, _w_i;

        _a_r = _signer(*A_r_in);
        _a_i = _signer(*A_i_in);
        _b_r = _signer(*B_r_in);
        _b_i = _signer(*B_i_in);
        _w_r = _signer(*w_r);
        _w_i = _signer(*w_i);

        m1 = _b_r * _w_r;
        m2 = _b_i * _w_i;
        m3 = _b_r * _w_i;
        m4 = _b_i * _w_r;
```

```
        m5 = ((2 * _a_r) << (BIT_LENGTH-1));// 2AR ALLIGNMENT
        m6 = ((2 * _a_i) << (BIT_LENGTH-1));// 2AI ALLIGNMENT
        s1 = ((_a_r << (BIT_LENGTH-1)) + m1);// AR ALLIGNMENT
        s2 = s1 - m2;
        s3 = ((_a_i << (BIT_LENGTH-1)) + m3);// AI ALLIGNMENT
        s4 = s3 + m4;
        s5 = m5 - s2;
        s6 = m6 - s4;

        *A_r_out = _round_and_scale(s2);
        *A_i_out = _round_and_scale(s4);
        *B_r_out = _round_and_scale(s5);
        *B_i_out = _round_and_scale(s6);
        return 0;
}

/* SIGN MOTIVATION
 * Signed integer multiply and integer multiply instruction work differently!
 */

signed long int _signer(unsigned long int number) {
        unsigned long int _sign = (1 << (BIT_LENGTH-1));
        unsigned long int _negs = ~((1 << (BIT_LENGTH)) - 1);
        if ((number & _sign) == _sign) {
                return (signed long int) (number | _negs);
        } else {
                return (signed long int) number;
        }
}


/* ROUNDING MOTIVATION
 *
 * mx = |m39|m38|.|m37|...|m20|m19|...|m0|
 *
 * out= |b19|.|b18|...|b0|        -> The trunc. and scale must be based on sizes!
 *
 *         |m39| |m38|.|m37|...|m20||m19||m18|...|m0|
 *         |b19|.|b18| |b17|...|b0 |
 *
 * rMask = | 0 |..............| 0 || 1 || 0 |...| 0 |
 * sMask = | 1 |..................| 1 || 0 |...| 0 |
 * pMask = | 0 |.........| 0 || 1 || 0 |........| 0 |
 *
 * - when to round up?         when the number is at the center while odd or has less
 *                          error going up !
 *
 * WARNING: SIGN EXTENSION TO BE IMPLEMENTED BY HAND!
 */

unsigned int _round_and_scale(signed long int number) {
        // mask to throw away the sign for the print on file
        unsigned long int _signdel = ((1 << (BIT_LENGTH)) - 1);
        unsigned long int _rMask = (1 << (BIT_LENGTH-1));// first bit to be truncated
        unsigned long int _sMask = ~(_rMask - 1);        // mask of all 0s before r
```

```
        unsigned long int _pMask = (1 << BIT_LENGTH);        // mask to tell if even or odd
        if (((number & _rMask) == _rMask) && (((number & _pMask) == _pMask) ||
        ((number | _sMask) != _sMask))) {
                        return (unsigned int) ((( number >> BIT_LENGTH) + 1) & _signdel);
        } else {
                return (unsigned int) ((number >> BIT_LENGTH) & _signdel);
        }

}
```

A.2.3 butterflyTest.c

```
/* Author:        Luca Lombardini
 * Academic_y:        2020/2021
 * Purpose:        (Master Degree) Digital Integrated Systems' Final Project
 * Contacts:        s277972@studenti.polito.it
 *                  lombamari2@gmail.com
 */
#include <stdio.h>
#include <stdlib.h>
#include "butterfly.h"

#ifndef BIT_LENGTH
#pragma message("BIT_LENGTH definition not found")
#define BIT_LENGTH 20
#endif

#define DATA_VECT_ELEMENT 6
#define HEX_LENGTH (BIT_LENGTH / 4)
#define OUT_FORMAT "%.5X\n"

int main(int argc, char** argv) {
        FILE* fp_in;
        FILE* fp_out;
        unsigned int data_v[DATA_VECT_ELEMENT];        //ar, ai, br, bi, wr, wi
        unsigned int out_v[DATA_VECT_ELEMENT-2];
        unsigned int index;
        if (argc <= 2) {
                if (argc == 1) {
                        fp_in = fopen("sample.txt", "r");
                } else {
                        fp_in = fopen(argv[1], "r");
                }

                fp_out = fopen("butterfly_out.txt", "w");
                if ((fp_in != NULL) && (fp_out != NULL)) {
                        index = 0;
                        while (!feof(fp_in)) {
                                fscanf(fp_in, "%x\n", &data_v[index]);
                                index++;
                                if (index == DATA_VECT_ELEMENT) {
                                        _butterfly(data_v, data_v+1, data_v+2, data_v+3,
                                        data_v+4, data_v+5, out_v, out_v+1, out_v+2, out_v+3);
                                        for(index = 0; index < DATA_VECT_ELEMENT-2; index++) {
                                                fprintf(fp_out, OUT_FORMAT, out_v[index]);
```

```
                                        }
                                        index = 0;
                                }
                        }
                } else {
                        puts("[ ERROR ]: input file cannot be opened");
                        return -2;
                }
        } else {
                puts("[ ERROR ]: arguments not compliant with specifications");
                return -1;
        }
        return 0;
}
```

A.2.4 fft.h

```
/* Author:        Luca Lombardini
 * Academic_y:        2020/2021
 * Purpose:        (Master Degree) Digital Integrated Systems' Final Project
 * Contacts:        s277972@studenti.polito.it
 *                 lombamari2@gmail.com
 */
#ifndef __FFT__
#define __FFT__

#include "butterfly.h"
#include <stdio.h>

int _fft(unsigned int* samples_r, unsigned int* samples_i, unsigned int* out_r, unsigned int* out_i);

unsigned int _reverse_kogge_stone(unsigned int a, unsigned int b);

void _level_printer(unsigned int * sre, unsigned int * sim, unsigned int stage);

#endif
```

A.2.5 fft.c

```
/* Author:        Luca Lombardini
 * Academic_y:        2020/2021
 * Purpose:        (Master Degree) Digital Integrated Systems' Final Project
 * Contacts:        s277972@studenti.polito.it
 *                 lombamari2@gmail.com
 */
#ifndef __FFT__
#include "fft.h"

#ifndef N_SAMPLES
#pragma message("N_SAMPLES definition not found")
#define N_SAMPLES 16
#endif

int _fft(unsigned int* samples_r, unsigned int* samples_i, unsigned int* out_r, unsigned int* out_i) {
        unsigned int tmp_re[N_SAMPLES];
        unsigned int tmp_im[N_SAMPLES];
```

```
unsigned int *v_in_re, *v_in_im, *v_out_re, *v_out_im;
unsigned int i, stage, group, max_subgroups, sub_limit;
unsigned int a_index, b_index, w_index;
unsigned int coeff_r[N_SAMPLES/2] = {   524287,
                                        484379,
                                        370728,
                                        200636,
                                        0,
                                        847940,
                                        677848,
                                        564197};
unsigned int coeff_i[N_SAMPLES/2] = {        0,
                                        847940,
                                        677848,
                                        564197,
                                        524288,
                                        564197,
                                        677848,
                                        847940};

max_subgroups = 1;        // tracks how many subgoups in a stage
                          // also used to avoid log2 calculus.
v_in_re = samples_r;        // points to the input arrays just at the
v_in_im = samples_i;        // beginning
v_out_re = tmp_re;
v_out_im = tmp_im;
for(stage = 0; max_subgroups != N_SAMPLES; stage++) {
        sub_limit = N_SAMPLES / (max_subgroups * 2);
        w_index = 0;
        for(group = 0; group < max_subgroups; group++) {
                for(i = 0; i < sub_limit; i++) {
                        a_index = 2 * sub_limit * group + i;
                        b_index = _reverse_kogge_stone(a_index, sub_limit);
                        _butterfly(         v_in_re + a_index,
                                            v_in_im + a_index,
                                            v_in_re + b_index,
                                            v_in_im + b_index,
                                            coeff_r + w_index,
                                            coeff_i + w_index,
                                            v_out_re + a_index,
                                            v_out_im + a_index,
                                            v_out_re + b_index,
                                            v_out_im + b_index);
                }
                w_index = _reverse_kogge_stone(w_index, N_SAMPLES/4);
        }
        #ifdef __LVL_DBG__
        _level_printer(v_out_re, v_out_im, stage);
        #endif
        max_subgroups *= 2; // next stage has double groups than now
        v_in_re = tmp_re;
        v_in_im = tmp_im;
}
// reordering
a_index = 0; // just reuse
```

```
        for(i = 0; i < N_SAMPLES; i++) {
                out_r[i] = v_out_re[a_index];
                out_i[i] = v_out_im[a_index];
                a_index = _reverse_kogge_stone(a_index, N_SAMPLES/2);
        }
        return 0;
}

unsigned int _reverse_kogge_stone(unsigned int a, unsigned int b) {
        // bitwise p and g bit vectors
        unsigned int generate;
        unsigned int propagate;
        unsigned int k;

        generate = a & b;
        propagate = a ^ b;

        // kogge-stone adder has log2(in's bitwidth) levels -> iterations
        for (k = 1; k <= 8*sizeof(generate)/2; k *= 2) {
                generate |= (generate >> k) & propagate;
                propagate &= (propagate >> k);
        }
        return (a ^ b ^ (generate >> 1));
}

void _level_printer(unsigned int * sre, unsigned int * sim, unsigned int stage) {
        int k;
        printf("# %u\n", stage);
        for (k = 0; k < N_SAMPLES; k++) {
                printf("%.5X", *(sre + k));
        }
        printf("\n");
        for (k = 0; k < N_SAMPLES; k++) {
                printf("%.5X", *(sim + k));
        }
        printf("\n");
}

#endif
```

A.2.6 fftTest.c

```
/* Author:    Luca Lombardini
 * Academic_y:    2020/2021
 * Purpose:    (Master Degree) Digital Integrated Systems' Final Project
 * Contacts:    s277972@studenti.polito.it
 *         lombamari2@gmail.com
 */
#include <stdio.h>
#include <stdlib.h>
#include "fft.h"

#ifndef BIT_LENGTH
#pragma message("BIT_LENGTH definition not found")
#define BIT_LENGTH 20
```

```
#endif

#define DATA_VECT_ELEMENT 16
#define HEX_LENGTH (BIT_LENGTH / 4)
#define OUT_FORMAT "%.5X"

int main(int argc, char** argv) {
    FILE* fp_in;
    FILE* fp_out;
    unsigned int data_re[DATA_VECT_ELEMENT], data_im[DATA_VECT_ELEMENT];
    unsigned int out_re[DATA_VECT_ELEMENT], out_im[DATA_VECT_ELEMENT];
    unsigned int index, i, j, k;
    char file_line[(DATA_VECT_ELEMENT)*HEX_LENGTH + 1];
    char tmp[HEX_LENGTH + 1];

    if (argc <= 2) {
        fp_in = fopen(argv[1],"r");
        fp_out = fopen("fft_out.txt", "w");
        if ((fp_in != NULL) && (fp_out != NULL)) {
            while (!feof(fp_in)) {                // A LINE CONTAINS 16 ELEMENTS
                fscanf(fp_in, "%s\n", file_line);         // A0-A7-B0-B7
                index++;                    // data identifiyer: either ar, ai, br, or bi
                for (i = 0; i < DATA_VECT_ELEMENT; i++) {  // from a single file line only half vector
                    for (j = 0; j < HEX_LENGTH; j++) { // copy data temporary inside a string buffer
                        tmp[j] = file_line[j + i*HEX_LENGTH];    //TEMPORARY BUFFERED SMALLER PACK
                    }
                    switch (index) {
                        case 1:
                            sscanf(tmp, "%x", &data_re[i]); // AR(0 TO 7) & BR(0 TO 7) = X_RE(0 TO 15)
                            break;
                        case 2:
                            sscanf(tmp, "%x", &data_im[i]); // AI(0 TO 7) & BI(0 TO 7) = X_IM(0 TO 15)
                            index = 0;
                            _fft(data_re,data_im,out_re,out_im);
                            for (k = 0; k < DATA_VECT_ELEMENT; k++) { // write real part
                                fprintf(fp_out, OUT_FORMAT, out_re[k]);
                            }
                            fprintf(fp_out, "\n");
                            for (k = 0; k < DATA_VECT_ELEMENT; k++) { // write imaginary part
                                fprintf(fp_out, OUT_FORMAT, out_im[k]);
                            }
                            fprintf(fp_out, "\n");
                            break;
                        case 0:
                            break;
                        default:
                            puts("[ ERROR ]: index outside valid range!");
                    }
                }
            }
        } else {
            puts("[ ERROR ]: one or more file/s cannot be opened");
        }
    } else {
        puts("[ ERROR ]: arguments not compliant with specifications");
```

```
        return -1;
    }
    return 0;
}
```

A.3 Extra: Python scripts

A.3.1 cupack_updater.py

```
#! /env/python

# Author:       Luca Lombardini
# Academic_y:   2020/2021
# Purpose:      (Master Degree) Digital Integrated Systems' Final Project
# Contacts:     s277972@studenti.polito.it
#               lombamari2@gmail.com

import sys
import csv

HEADER="""-- Author:       Luca Lombardini
-- Academic_y:   2020/2021
-- Purpose:      (Master Degree) Digital Integrated Systems' Final Project
-- Contacts:     s277972@studenti.polito.it
--               lombamari2@gmail.com
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
USE work.definespack.all;

PACKAGE cupack IS
"""

BASE='\tCONSTANT {}\t: std_logic_vector({}-1 DOWNTO 0) := "{}";\n'
BASE_K = '\tCONSTANT {}\t: integer\t:= {};\n'
FOOTER="END cupack;"
FILENAME = "cu_tracker"
FILEOUT = "../src/cupack.vhd"
COMMAND_LEN = 28
NEXT_ADDR_CC= 6
BASE_ADDR_LEN=4

bit_len_map = {BASE_ADDR_LEN:"base_addr", 5:"lsb_addr", NEXT_ADDR_CC:"cc_lsb_addr",
                COMMAND_LEN:"command_len"}

 # file which will contain the defines and the one containing the filenames of
 # the file containing the mappings
with open(FILEOUT, 'w') as f_out, open(FILENAME,'r') as tracker:
    f_out.write(HEADER)
    for key, value in bit_len_map.items():    # add integer constants
                f_out.write(BASE_K.format(value, key))
    f_out.write(BASE_K.format("uir_width", COMMAND_LEN + NEXT_ADDR_CC))
    # for every file in the tracker read the map composed of tag and bits, and assign
    # them into the package
    for input_filename in tracker:
```

```
        with open(input_filename.rstrip('\n'), 'r') as filein:
            csvbuf = csv.reader(filein)
            for line_no, line_list in enumerate(csvbuf):
                if line_no:    # the first line is just a header with info for humans
                    if "index" in line_list:
                        for position, bit_label in enumerate(line_list[1:]):
                            f_out.write(BASE_K.format(bit_label,len(line_list[1:])-position-1))
                    else:
                        tag = line_list[0]
                        # do not pick - in the bitstring
                        bit_string = ''.join( [ _ for _ in line_list[1:] if _ != '-' ] )
                        # add a label definition
                        f_out.write(BASE.format(tag, bit_len_map[len(bit_string)], bit_string))

    f_out.write(FOOTER)
```

A.3.2 rom_modifier.py

```
#! /env/python
# Author:       Luca Lombardini
# Academic_y:   2020/2021
# Purpose:      (Master Degree) Digital Integrated Systems' Final Project
# Contacts:     s277972@studenti.polito.it
#               lombamari2@gmail.com
import sys
import csv
import os

TRACKER_FILE = "rom_tracker"
OUT_FILENAME = "{}.vhd"
ROM_LINE = "\t\tWHEN {} => DATA <= {};\n"
i=0

with open(TRACKER_FILE, 'r') as tracker:
    for input_filename in tracker:    # loop for each filename contained inside the tracker file
        IN_FILENAME = "../src/" + input_filename.split('.')[-2].split('/')[-1] + ".vhd"
        with open(input_filename.rstrip('\n'),'r') as csv_file, open(IN_FILENAME,'r') as src_file,
        open("tmp",'w') as tmp_dest:
            ignore = False
            csv_buf = csv.reader(csv_file)
            for code_line in src_file:
                # modify only the rom out assignment based on input address
                if "WHEN" in code_line:
                    # except for the default case, this one needs to be kept
                    if "OTHERS" in code_line:
                        tmp_dest.write(code_line)
                    elif ignore == False:
                        ignore = True    # now on, all the previous cases are obsolete
                        # score is a list, with a header useful for humans
                        for line_no, score in enumerate(csv_buf):
                            if line_no:
                                # to support WHEN CASE_x => DATA <= cc & NEXT_ADDR & parity & ctrlwrd
                                tmp_dest.write( ROM_LINE.format( score[0],
                                ' & '.join([ _ for _ in score[1:] if _ != '-']) ))
                else:
```

```
                tmp_dest.write(code_line)
        os.rename("tmp",IN_FILENAME)        # finally point to the original source file
        i+=1
```

A.3.3 butterfly_in_filler.py

```
#! /env/python

# Author:       Luca Lombardini
# Academic_y:   2020/2021
# Purpose:      (Master Degree) Digital Integrated Systems' Final Project
# Contacts:     s277972@studenti.polito.it
#               lombamari2@gmail.com

import random

not_msb = list("0123456789ABCDEF")
msb = list("0123CDEF")

data_set = 1000
many_chars = 5
many_data = 6

with open("../sim/butterfly_in.hex","w") as fileout:
        for set in range(data_set): # how many butterflies to do
                for j in range(many_data): # how many input values
                        tmp = ""
                        for i in range(many_chars): # how many bit-blocks
                                if i:
                                        tmp += random.choice(not_msb)
                                else:
                                        tmp = random.choice(msb)
                        fileout.write(tmp + "\n")
```

A.3.4 check_state_by_ctrl_wrd.py

```
#! /env/python

# Author:       Luca Lombardini
# Academic_y:   2020/2021
# Purpose:      (Master Degree) Digital Integrated Systems' Final Project
# Contacts:     s277972@studenti.polito.it
#               lombamari2@gmail.com

import csv

dict_map = {}

REF_CTRL = "../project_files/ctrlwrds.csv"
FILE_TO_CHECK = "../sim/ctrl_wrd_out.bin"
with open(REF_CTRL, 'r') as ref_file:
        csv_buf = csv.reader(ref_file)
        for line_no, state_wrd in enumerate(csv_buf):
                if line_no:
                        dict_map[''.join(state_wrd[1:])] = state_wrd[0]
```

```
with open(FILE_TO_CHECK, 'r') as chkfile:
        for line in chkfile:
                try:
                        print(dict_map[line.rstrip("\n")])
                except KeyError:
                        print("UNKNOWN_STATE")
```

A.3.5 fft_instancer.py

```
#! /env/python

# Author:       Luca Lombardini
# Academic_y:   2020/2021
# Purpose:      (Master Degree) Digital Integrated Systems' Final Project
# Contacts:     s277972@studenti.polito.it
#               lombamari2@gmail.com

################################################################################
#   Global definitions and imports
import math
import os
SOURCE = "../src/fft_unit.vhd"
N_SAMPLES = 16
n_coef = 8

################################################################################
# can be useful, i believe that

def reverse_kogge_stone(a,b):
    generate = a & b
    propagate = a ^ b
    max_iter = math.ceil( math.log2( max(a,b) ) ) / 2
    k = 1
    while k <= max_iter:
        generate |= (generate >> k) & propagate
        propagate &= (propagate >> k)
        k *= 2
    return (a ^ b ^ (generate >> 1))

ordinal = lambda n: "%d%s" % (n,"tsnrhtdd"[(n//10%10!=1)*(n%10<4)*n%10::4])

################################################################################
#   Basic functions used to manipulate instances

def newInstName(level, group, inst, num):   # creates a string for a new name
    return "lv{}_gr{}_{}{}".format(level, group, inst, num)

def makeMap(port):                          # can map both generic and port maps
    return ", ".join(port)

def newBitSig(name, sigtype):
    return "\tSIGNAL {} : {};".format(name, sigtype)

def newNumSig(name, sigtype, up_limit):
    return "\tSIGNAL {} : {}({}-1 DOWNTO 0);".format(name, sigtype, up_limit)
```

```
def newButterfly(inst_name, port_map):
    return "\t{} : butterfly PORT MAP ({});".format(inst_name, port_map)

def newPipe(inst_name, port_map):
    return "\t{} : pipe_machine PORT MAP({});".format(inst_name, port_map)

def newConnection(dest, source):
    return "\t{} <= {};".format(dest,source)

def newSeparator(text):
    return "--" + "#"*77 + "\n" + "--#\t{}".format(text)

################################################################################
#   Class simulating the content of the file

class vhdl_file():
    def __init__(self):
        self.head = []
        self.foot = ["END ARCHITECTURE;"]
        self.list_of_sigs = []
        self.list_of_inst = []

    def addHeader(self, string):
        self.head += [string]

    def addFooter(self, element):
        self.foot = self.foot[:-1] + [element] + self.foot[-1]

    def addSignal(self, element):
        self.list_of_sigs += [element]

    def addInstance(self, element):
        self.list_of_inst += [element]

    def __str__(self):
        assembled = "\n".join(self.head) + "\n"
        assembled += "\n".join(self.list_of_sigs) + "\n"
        assembled += "BEGIN\n"
        assembled += "\n".join(self.list_of_inst) + "\n"
        assembled += "\n".join(self.foot)
        return assembled
################################################################################
#   iterate inside the fft_unit.vhd to change the instantiation part

outputFile = vhdl_file()

#coef_real = [524287, 484379, 370728, 200636, 0, 847940, 677848, 564197]
#coef_imag = [0, 847940, 677848, 564197, 524288, 564197, 677848, 847940]

FROM_DATA_IN = "DATA_IN(io_width*{0}-1 DOWNTO io_width*{1})"
FROM_COEF_IN = "COEF_IN(io_width*{0}-1 DOWNTO io_width*{1})"
FROM_BETWEEN = "lv{}_out{}"
DONE_BETWEEN = "lv{}_gr{}_done{}"
DONE_AGGR = "doneAggr{}"
```

```
PIPE_MACHINE = "pipe{}"
FROM_COEF_BUF= "w_pipe{0}(io_width*{1}-1 DOWNTO io_width*{2})"
COEF_USED = "lev{0}_w{1}"
TO_DATA_OUT = "DATA_OUT(io_width*{0}-1 DOWNTO io_width*{1})"

with open(SOURCE,"r") as filein:
    for linein in filein:
        if "SIGNAL" not in linein and "BEGIN" not in linein:
            outputFile.addHeader(linein.rstrip("\n"))
        else: # start creating new defs
            break
    # butterfly inst
    stage = 0
    max_subgroups = 1
    while max_subgroups != N_SAMPLES:
        outputFile.addInstance(newSeparator(ordinal(stage+1) + " Layer"))
        sublimit = N_SAMPLES // (max_subgroups*2)
        w_index = 0
        doneList = []
        assList = [] ## store temporaneo delle assegnazioni, fatte al fondo di
        for group in range(max_subgroups):
            for i in range(sublimit):
                a_index = 2 * sublimit * group + i
                b_index = reverse_kogge_stone(a_index, sublimit)
                if stage == 0:          # first stage is different, direct input
                    startBit = "START"  # and reverse ordered data
                    _index = N_SAMPLES-a_index
                    portAIn = FROM_DATA_IN.format(_index, _index-1)
                    dataInA = "lv0_in{}".format(a_index)
                    _index = N_SAMPLES - b_index
                    portBIn = FROM_DATA_IN.format(_index, _index-1)
                    dataInB = "lv0_in{}".format(b_index)
                    _index = n_coef - w_index
                    coefFrom = FROM_COEF_IN.format(_index, _index-1)
                    assList += [newConnection(dataInA, portAIn)]
                    assList += [newConnection(dataInB, portBIn)]
                    outputFile.addSignal(newNumSig(dataInA,"signed","io_width"))
                    outputFile.addSignal(newNumSig(dataInB,"signed","io_width"))
                else:           # the others take inputs from the prev. stage
                    startBit = DONE_AGGR.format(stage-1)
                    dataInA = FROM_BETWEEN.format(stage - 1, a_index)
                    dataInB = FROM_BETWEEN.format(stage - 1, b_index)
                    _index = n_coef - w_index
                    coefFrom = FROM_COEF_BUF.format(stage - 1, _index, _index-1)
                coefIn = COEF_USED.format(stage,w_index)
                # new output signals
                dataOutA = FROM_BETWEEN.format(stage, a_index)
                dataOutB = FROM_BETWEEN.format(stage, b_index)
                doneBit = DONE_BETWEEN.format(stage,group,i)
                doneList += [doneBit]
                ## new butterfly
                newButtName = newInstName(stage, group, "but", i)
                newButtPort = makeMap(["CLK", "RST_n", startBit, dataInA,
                dataInB, coefIn, doneBit, dataOutA, dataOutB])
                _butterfly = newButterfly(newButtName, newButtPort)
```

```
                # add declarations and connections to vhdl file
                outputFile.addSignal(newNumSig(dataOutA,"signed","io_width"))
                outputFile.addSignal(newNumSig(dataOutB,"signed","io_width"))
                outputFile.addSignal(newBitSig(doneBit,"std_logic"))
                outputFile.addInstance(_butterfly)
                #outputFile.addSignal(newBitSig(startBit,"std_logic"))
            ## coefIn da portare fuori, al gruppo, non qui @ butterfly
            outputFile.addSignal(newNumSig(coefIn,"signed","io_width"))
            assList += [newConnection(coefIn, coefFrom)]
            w_index = reverse_kogge_stone(w_index, N_SAMPLES//4)
        max_subgroups *= 2
        if stage != (math.log2(N_SAMPLES)-1): ## last stage doesnt need pipe
            next_start = DONE_AGGR.format(stage)
            outputFile.addSignal(newBitSig(next_start,"std_logic"))
            ## add a new pipe_machine for W propagation
            ## pipemachine prende aggregated done, non il singolo
            pipeIn = "COEF_IN" if stage == 0 else "w_pipe{}".format(stage-1)
            pipeOut = "w_pipe{}".format(stage)
            outputFile.addSignal(newNumSig(pipeOut,"signed","io_width*n_coef"))
            newPipeName = PIPE_MACHINE.format(stage)
            #newPipeGeneric = makeMap(["io_width*n_coef"])
            newPipePort = makeMap(["CLK", "RST_n", startBit, DONE_AGGR.format(stage),
            pipeIn, pipeOut])
            _pipemachine = newPipe(newPipeName,newPipePort)
            #_pipemachine = newPipe(newPipeName,newPipeGeneric,newPipePort)
            outputFile.addInstance(_pipemachine)
        else: ## last stage has output reorder
            next_start = "DONE"
        for _ in assList:
            outputFile.addInstance(_)
        outputFile.addInstance(newConnection(next_start,' AND '.join(doneList)))
        stage += 1

# reorder
outputFile.addInstance(newSeparator("Output reorder"))
a_index = 0
for i in range(N_SAMPLES):
    _outPtr = N_SAMPLES - i
    outputFile.addInstance(newConnection(TO_DATA_OUT.format(_outPtr,_outPtr-1),
    "lv{}_out{}".format(stage-1,a_index)))
    a_index = reverse_kogge_stone(a_index, N_SAMPLES//2)

## RENAME!!!
with open("tmp","w") as fileout:
    fileout.write(outputFile.__str__())

os.rename("tmp",SOURCE)
```

A.3.6 in_maker.py

```
#! /env/python

# Author:       Luca Lombardini
# Academic_y:   2020/2021
# Purpose:      (Master Degree) Digital Integrated Systems' Final Project
```

```
# Contacts:     s277972@studenti.polito.it
#               lombamari2@gmail.com

#import math
import numpy

full_scale = 2**18 -1
offscale = 2**20

################################################################################
#   Functions to be used to create test vectors

def first_ord_lpf(fs, in_vect, k, b):
    unistep = [True if _ >= b else False for _ in in_vect]
    tmp = numpy.multiply(numpy.exp(numpy.subtract(b, in_vect)),unistep)
    scale = fs / max(tmp)
    return numpy.multiply((fs*k), tmp)

def diracs_delta(fs, in_vect, k, b):
    return [ (fs * k) if _ == b else 0 for _ in in_vect]

def constant(fs, in_vect, k, b):
    return [(fs * k) for _ in in_vect]

def step(fs, in_vect, k, b):
    unistep = [True if _ >= b else False for _ in in_vect]
    return [(fs * k) * _ for _ in unistep]

def cosine(fs, in_vect, k, b): # cosine with time shift and dummy phase
    w = (2 * numpy.pi / 16)*3
    arg = numpy.subtract(numpy.multiply(w,numpy.subtract(in_vect,b)),(0.05*b))
    return numpy.multiply((fs*k),numpy.cos(arg))

def door(fs, in_vect, k, b):
    window = [ True if _ >= (b-(b/2)) and _ < (b+(b/2)) else False for _ in in_vect]
    return numpy.multiply((fs*k), window)

def sinc_f(fs, in_vect, k, b):
    val = numpy.divide(numpy.subtract(in_vect,b),0.6)
    tmp = numpy.sinc(val)
    scale = fs/max(tmp)
    return numpy.multiply(scale, tmp)

funcs = [first_ord_lpf, diracs_delta, constant, step, cosine, door, sinc_f]
################################################################################
#   Test vector creation

samples = list(range(0,16))

with open("../sim/fft_in_vectors.hex", "w") as filein,
open("../sim/fft_out_reference.hex","w") as fileref:
    for f in funcs:
        for k in numpy.arange(0.2,1.2,0.2):
            for b in samples:
                to_print = [round(_) for _ in f(full_scale, samples, k, b)]
```

```
                for num in numpy.real(to_print):
                    buf = offscale + num if num < 0 else num
                    filein.write(format(buf,"05X"))
                filein.write("\n")
                for num in numpy.imag(to_print):
                    buf = offscale + num if num < 0 else num
                    filein.write(format(buf,"05X"))
                filein.write("\n")
                # do an fft just to have imag part as input
                to_print = numpy.rint(numpy.divide(numpy.fft.fft(to_print),16))
                for num in numpy.real(to_print):
                    buf = offscale + num if num < 0 else num
                    filein.write(format(round(buf),"05X"))
                    fileref.write(format(round(buf),"05X"))
                filein.write("\n")
                fileref.write("\n")
                for num in numpy.imag(to_print):
                    buf = offscale + num if num < 0 else num
                    filein.write(format(round(buf),"05X"))
                    fileref.write(format(round(buf),"05X"))
                filein.write("\n")
                fileref.write("\n")
                to_print = numpy.rint(numpy.divide(numpy.fft.fft(to_print),16))
                for num in numpy.real(to_print):
                    buf = offscale + num if num < 0 else num
                    fileref.write(format(round(buf),"05X"))
                fileref.write("\n")
                for num in numpy.imag(to_print):
                    buf = offscale + num if num < 0 else num
                    fileref.write(format(round(buf),"05X"))
                fileref.write("\n")


with open("../sim/fft_in_subset.hex", "w") as filein,
open("../sim/fft_out_subset_ref.hex","w") as fileref:
    for f in funcs:
        for b in (0, 8):
            # exp decr. max ampl.
            to_print = [round(_) for _ in f(full_scale, samples, 1, b)]
            for num in numpy.real(to_print):
                buf = offscale + num if num < 0 else num
                filein.write(format(buf,"05X"))
            filein.write("\n")
            for num in numpy.imag(to_print):
                buf = offscale + num if num < 0 else num
                filein.write(format(buf,"05X"))
            filein.write("\n")
            # do an fft just to have imag part as input
            to_print = numpy.rint(numpy.divide(numpy.fft.fft(to_print),16))
            for num in numpy.real(to_print):
                buf = offscale + num if num < 0 else num
                filein.write(format(round(buf),"05X"))
                fileref.write(format(round(buf),"05X"))
            filein.write("\n")
            fileref.write("\n")
```

```
            for num in numpy.imag(to_print):
                buf = offscale + num if num < 0 else num
                filein.write(format(round(buf),"05X"))
                fileref.write(format(round(buf),"05X"))
            filein.write("\n")
            fileref.write("\n")
            to_print = numpy.rint(numpy.divide(numpy.fft.fft(to_print),16))
            for num in numpy.real(to_print):
                buf = offscale + num if num < 0 else num
                fileref.write(format(round(buf),"05X"))
            fileref.write("\n")
            for num in numpy.imag(to_print):
                buf = offscale + num if num < 0 else num
                fileref.write(format(round(buf),"05X"))
            fileref.write("\n")
```

A.3.7 graphics.py

```
#! /env/python

# Author:       Luca Lombardini
# Academic_y:   2020/2021
# Purpose:      (Master Degree) Digital Integrated Systems' Final Project
# Contacts:     s277972@studenti.polito.it
#               lombamari2@gmail.com

import matplotlib.pyplot as plt
import numpy

offscale = 2**20
st_neg = 2**19
full_scale = 2**18 -1

IN_SAS = "../sim/fft_in_subset.hex"
OUT_SAS= "../sim/fft_out_subset.hex"
REF_SUB= "../sim/fft_out_subset_ref.hex"

inDataIm = []
outDataIm= []
refDataIm= []
inDataRe = []
outDataRe= []
refDataRe= []

def makePlot():
    global inDataIm
    global outDataIm
    global refDataIm
    global inDataRe
    global outDataRe
    global refDataRe
    time = numpy.arange(16)
    freq = numpy.fft.fftfreq(time.shape[-1])
    freq = numpy.sort(freq)
```

```
    # translate back to signed number
    inDataIm  =[int(_,16) - offscale if int(_,16) >= st_neg else int(_,16) for _ in inDataIm]
    outDataIm =[int(_,16) - offscale if int(_,16) >= st_neg else int(_,16) for _ in outDataIm]
    refDataIm =[int(_,16) - offscale if int(_,16) >= st_neg else int(_,16) for _ in refDataIm]
    inDataRe  =[int(_,16) - offscale if int(_,16) >= st_neg else int(_,16) for _ in inDataRe]
    outDataRe =[int(_,16) - offscale if int(_,16) >= st_neg else int(_,16) for _ in outDataRe]
    refDataRe =[int(_,16) - offscale if int(_,16) >= st_neg else int(_,16) for _ in refDataRe]

    # make graphs
    fig, (ax1, ax2, ax3, ax4) = plt.subplots(4)
    #fig.suptitle() set tag from list to be defined
    ax1.plot(time, numpy.divide(inDataRe, full_scale), time, numpy.divide(inDataIm,full_scale))
    ax1.set_title("Time domain input")
    ax2.plot(freq, numpy.divide(outDataRe[8:] + outDataRe[:8],full_scale),
    freq, numpy.divide(outDataIm[8:] + outDataIm[:8],full_scale))
    ax2.set_title("Frequency domain output")
    tmp = numpy.abs(numpy.subtract(outDataRe[8:] + outDataRe[:8], refDataRe[8:] + refDataRe[:8]))
    ax3.plot(freq, tmp)
    ax3.set_title("Frequency domain, real out error")
    tmp = numpy.abs(numpy.subtract(outDataIm[8:] + outDataIm[:8], refDataIm[8:] + refDataIm[:8]))
    ax4.plot(freq, tmp)
    ax4.set_title("Frequency domain, imag out error")
    plt.show()



with open(IN_SAS, "r") as inFile, open(OUT_SAS, "r") as outFile, open(REF_SUB, "r") as refFile:
    for lineNo, (lineIn, lineOut, lineRef) in enumerate(zip(inFile, outFile, refFile)):
        if lineNo % 2:
            inDataIm = [lineIn[i:i+5] for i in range(0, len(lineIn) -1, 5)]
            outDataIm= [lineOut[i:i+5] for i in range(0, len(lineOut) -1, 5)]
            refDataIm= [lineRef[i:i+5] for i in range(0, len(lineRef) -1, 5)]
            makePlot()
        else:
            inDataRe = [lineIn[i:i+5] for i in range(0, len(lineIn) -1, 5)]
            outDataRe= [lineOut[i:i+5] for i in range(0, len(lineOut) -1, 5)]
            refDataRe= [lineRef[i:i+5] for i in range(0, len(lineRef) -1, 5)]
```

A.4 Extra: Bash scripts

A.4.1 comp_and_sim.sh

```
#! /bin/bash

# Author:       Luca Lombardini
# Academic_y:   2020/2021
# Purpose:      (Master Degree) Digital Integrated Systems' Final Project
# Contacts:     s277972@studenti.polito.it
#               lombamari2@gmail.com

trap '(( "$?" == 0 )) || echo -e "\e[31m[ ERROR ]: Error occurred!\e[0m"' EXIT

source /software/scripts/_modelsim_init

# check if at least 2 arguments are passed. More than 2 are ignored
```

```
if [[ "$#" > 1 ]] && [[ -e "$2" ]]; then
        [ -e work/_info ] || vlib work
        for hdl_file in $(cat "$2") # iterate over the strings in the file
        do
                if [[ "$hdl_file" == "%"* ]]; then
                        continue
                fi
                filename=$(basename "$hdl_file") # get filename.extension from the string which contain
                if [[ "${filename##*.}" == "vhd" ]]; then
                        echo -e "\e[32m[ INFO ]: compiling $filename ...\e[0m"
                        vcom -93 -work ./work "$hdl_file" || exit -1
                elif [[ "${filename##*.}" == "v" ]]; then
                        echo -e "\e[32m[ INFO ]: compiling $filename ...\e[0m"
                        vlog -work ./work "$hdl_file" || exit -1
                else
                        echo -e "\e[31m[ ERROR ] : source hdl file ( $filename ) not found!\e[0m"
                fi
        done
        postfix="${2##*_}"
        if [[ "$postfix" = "pos" ]]; then
                # do we really have to delete EVERYTHING inside these directories?
                [ -d ../vcd ] && rm -rf ../vcd/* || mkdir ../vcd
                [ -d ../saif ] && rm -rf ../saif/* || mkdir ../saif
        fi
        if [[ "$1" == "-dbg" ]]; then
                vsim -do "simulate_$postfix.do" || exit -1
        elif [[ "$1" == "-nogui" ]]; then
                vsim -c -do "simulate_$postfix.do" || exit -1
                echo -e "\e[1m\nDONE!\n\e[0m"
        elif [[ "$1" == "-nosim" ]]; then
                echo -e "\e[1m\nNO SIMULATION WILL START!\n\e[0m"
                exit 0
        else
                echo "no arguments passed, nothing to do"
                exit -1
        fi
        #mv ./results.txt "./results_$postfix.txt"
else
        echo -e "\e[31m[ SYNTAX ERROR ]: Not enough arguments or inexistent file. Syntax is $0 <opt> <c
        exit -1
fi

exit 0
```

A.4.2 infinite_sim.sh

```
#! /bin/bash

# Author:       Luca Lombardini
# Academic_y:   2020/2021
# Purpose:      (Master Degree) Digital Integrated Systems' Final Project
# Contacts:     s277972@studenti.polito.it
#               lombamari2@gmail.com

trap '(( "$?" == 0 )) || echo -e "\e[31m[ ERROR ]: Error occurred!\e[0m"' EXIT
```

```
if [[ "$#" = 1 ]]; then
        postfix="${1##*_}" # a check on postfix should be done, i am lazy
        echo -e "\e[32m[ INFO ]: modifying the definespack.vhd constant definitions\e[0m"
        ./comp_and_sim.sh -nogui "$1" <<EOF
quit -f
EOF
else
        echo -e "\e[31m[ SYNTAX ERROR ]: Not enugh arguments. One argument is required.\e[0m"
fi
```