



DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATIONS
Master degree in Electronic Engineering

Sistemi Digitali Integrati
Operazione San Silvestro:
Butterfly

Autori:

Antonio Fichera 337213, Elisabetta Vannelli 346477

Febbraio 2025

Indice

1 Introduzione	2
2 Data Flow Diagram	3
3 Datapath Butterfly	4
3.1 Schema generale del Datapath	4
3.2 Moltiplicatore	6
3.3 Sommatore	7
3.4 Arrotondatore Half-Up	7
4 Porte e Architettura del sistema FFT_16x16	8
5 Parallelismo	9
5.1 Parallelismo esterno	9
5.2 Parallelismo interno	9
6 Unità di controllo	10
6.1 CU schema generale	10
6.2 Late Status PLA	12
6.3 Struttura μ ROM	13
7 ASM chart, Protocollo e Timing Diagram	15
8 VHDL	18
9 Testbench	19
10 Prova Matlab	24
11 Conclusioni	25
12 Appendice: codici VHDL	26
12.1 TB.FFT_16x16.vhd	26
12.2 FFT_16x16.vhd	34
12.3 Butterfly.vhd	44
12.4 Multiplier_Shifter.vhd	50
12.5 Adder_Subtractor.vhd	51
12.6 Rounder.vhd	52
12.7 Register_Nbit.vhd	53
12.8 MUX_2TO1_Nbit.vhd	54
12.9 Decoder_4to16.vhd	54
12.10 EVEN_ROM.vhd	55
12.11 ODD_ROM.vhd	56
12.12 LateStatus_PLA.vhd	57
12.13 Register_Nbit_SLV.vhd	58
12.14 MUX_2TO1_Nbit_SLV.vhd	58
12.15 MUX_2TO1.vhd	59

1 Introduzione

L'approccio adottato fin da subito per la realizzazione dell'Operazione San Silvestro, ormai da noi ribattezzata ironicamente Operazione San Valentino, è stato un processo iterativo fatto di interminabili brainstorming, idee buttate su carta, testate, scartate e sostituite con nuove soluzioni.

Abbiamo esplorato lo spazio delle numerose soluzioni fino a trovare la configurazione più efficiente. Molte delle soluzioni studiate da principio sono poi risultate, in fase di VHDL e testbench, impraticabili o incompatibili con la realizzazione della FFT 16X16.

Alcune di queste idee prevedevano un numero di porte decisamente ridotto rispetto all'attuale configurazione; si prevedevano due porte di ingresso, una per i dati, una per i coefficienti e un'unica porta di uscita, per un totale di 3 porte per singola Butterfly.

Credendo che in questo modo avremmo abbattuto notevolmente i costi, ci siamo poi scontrati con la realtà dei timing e delle connessioni fisiche tra una Butterfly e l'altra; in particolare, essendo che i valori A' e B' prodotti da una Butterfly sarebbero andati in ingresso a due Butterfly distinte, questo ha necessariamente portato alla divisione delle porte dei dati in ingresso e in uscita. Quindi, si avranno complessivamente due porte d'ingresso separate per A e B, e altre due porte d'uscita per A' e B'.

Inoltre, visto che i "Twiddle Factor" con cui lavorano le Butterfly sono fissi, si è optato anche per la separazione in due della porta d'ingresso a questi dedicate; ciò è stato fatto anche per poter fornire simultaneamente parti reale ed immaginaria dei suddetti coefficienti, andando a semplificare la vita a un possibile utente, il quale non avrà più l'obbligo di fornirli con un determinato tempismo.

Per questo motivo, la soluzione scelta ha anche semplificato notevolmente la fase di test, in quanto è stato possibile effettuarla senza l'uso continuo di comandi di wait, per fornire alle Butterfly i coefficienti W.

Modifiche come queste sono state fatte durante tutto il processo di elaborazione della Butterfly. Qui proponiamo la configurazione finale che riteniamo sposarsi meglio con le richieste del nostro committente e che rappresenta un compromesso tra costi, efficienza e fattibilità.

2 Data Flow Diagram

Per la stesura del DFD si è adottato il segno grafico di due semicerchi spezzati per interpretare visibilmente il livello di pipe interno alle operazioni di moltiplicazione e somma/sottrazione (mentre l'arrotondamento è rappresentato con due rettangoli). Tali operazioni cominciano in uno step e terminano nel successivo, rendendo il dato disponibile a quello ancora successivo a causa del registro temporaneo, posto in uscita a ogni blocco aritmetico.

Tramite il DFD si è riusciti a ottimizzare il numero di operatori aritmetici usati dall'algoritmo e minimizzato il numero di variabili temporanee.

In particolare, lo scheduling delle operazioni adottato permette di lavorare con un solo moltiplicatore e un solo sommatore, nonostante di quest'ultimo se ne avessero due a disposizione.

Inoltre, lo studio del tempo di vita delle variabili ha reso evidente come fosse necessario introdurre due registri temporanei, come già riportato in precedenza.

Di seguito è riportato il DFD prodotto:

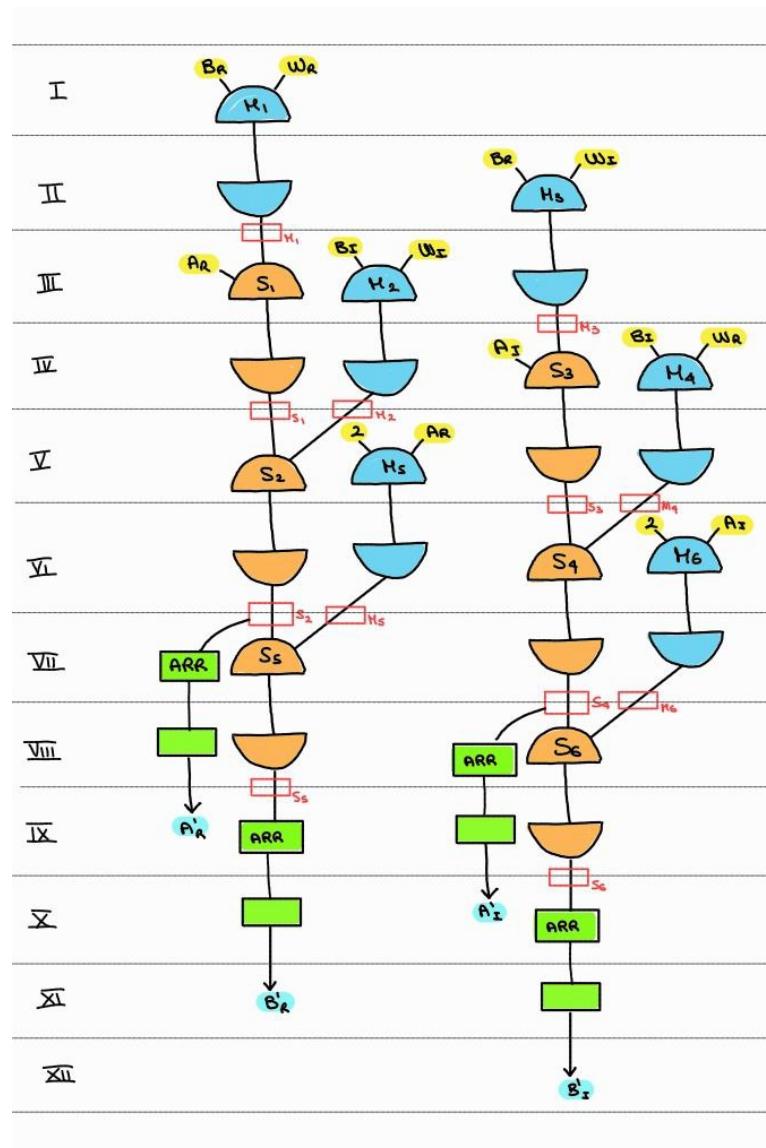


Figura 1: Data Flow Diagram

3 Datapath Butterfly

3.1 Schema generale del Datapath

Di seguito si propone una descrizione generale del Datapath, dei blocchi costituenti, delle connessioni e degli aspetti logici. La trattazione sul parallelismo è gestita separatamente nei paragrafi successivi. Il Datapath riportato in Figura 2 risulta così composto:

Porte di I/O

- Porta INPUT_A → porta d'ingresso del dato A (entrano sia parte reale che immaginaria)
- Porta INPUT_B → porta d'ingresso del dato B (entrano sia parte reale che immaginaria)
- Porta INPUT_WR → porta d'ingresso dedicata alla sola parte reale dei coefficienti W
- Porta INPUT_WI → porta d'ingresso dedicata alla sola parte immaginaria dei coefficienti W
- Porta OUTPUT_A → porta d'uscita del dato A' (escono sia parte reale che immaginaria)
- Porta OUTPUT_B → porta d'uscita del dato B' (escono sia parte reale che immaginaria)

Componenti

- Registro REG_AR → salva la parte reale di A in ingresso
- Registro REG_AI → salva la parte immaginaria di A in ingresso
- Registro REG_BR → salva la parte reale di B in ingresso
- Registro REG_BI → salva la parte immaginaria di B in ingresso
- Registro REG_WR → salva la parte reale di W
- Registro REG_WI → salva la parte immaginaria di W
- Registro REG_TMP_MPY → salva il risultato parziale in uscita dal blocco moltiplicatore
- Registro REG_TMP_SUM → salva il risultato parziale in uscita dal blocco sommatore
- Registro REG_AR_PRIMO → salva la parte reale di A' in uscita
- Registro REG_AI_PRIMO → salva la parte immaginaria di A' in uscita
- Registro REG_BR_PRIMO → salva la parte reale di B' in uscita
- Registro REG_BI_PRIMO → salva la parte immaginaria di B' in uscita
- MUX_A → permette la selezione tra parte reale o immaginaria di A in ingresso
- MUX_B → permette la selezione tra parte reale o immaginaria di B in ingresso
- MUX_W → permette la selezione tra parte reale o immaginaria di W
- MUX_AB → permette la selezione tra A o B prima del blocco moltiplicatore
- MUX_SUM → permette la selezione tra il dato A o la somma parziale in REG_TMP_SUM
- MUX_OUT_A → permette la selezione tra parte reale o immaginaria di A' in uscita

- MUX_OUT_B → permette la selezione tra parte reale o immaginaria di B' in uscita
- MULTIPLIER → effettua la moltiplicazione tra due dati o moltiplica un dato per 2
- ADDER → effettua le operazioni di somma o sottrazione tra due dati
- ROUNDER_HU → effettua le operazioni di arrotondamento e troncamento del dato

Datapath/Execution Unit

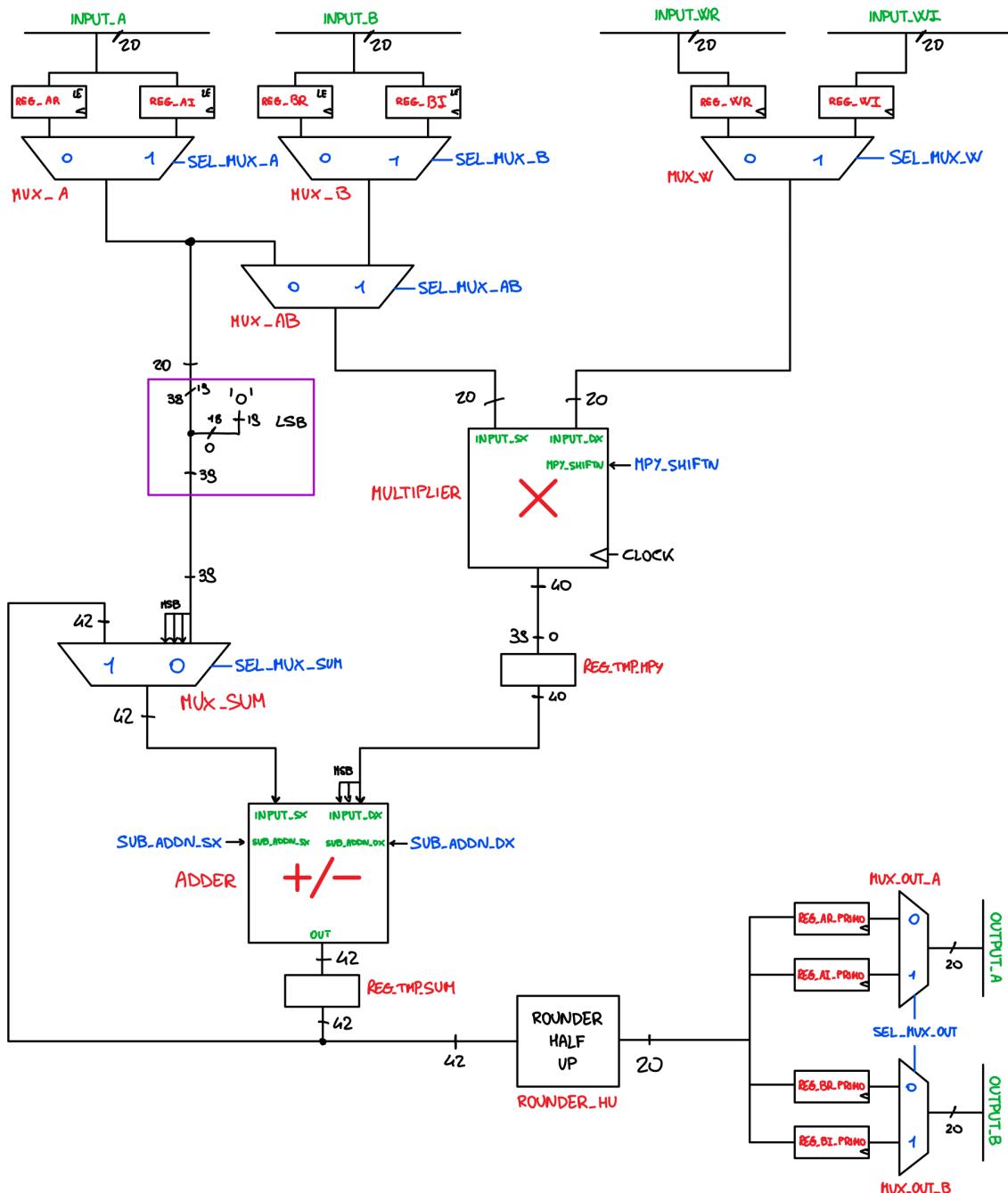


Figura 2: Datapath/Execution Unit della Butterfly

Dalle porte d'ingresso dei dati, entrano prima le parti reali e poi (dopo un colpo di clock) le parti immaginarie. Quindi i dati vengono memorizzati negli appositi registri, le cui uscite sono collegate a dei Multiplexer, MUX_A e MUX_B, pilotati dai selettori SEL_MUX_A e SEL_MUX_B che distingueranno quale componente prendere, se quella reale (0) o quella immaginaria (1).

Le uscite di questi due MUX vanno ad un successivo mux, MUX_AB, che sceglie quale dato prendere, se A o B, da portare all'ingresso del moltiplicatore, dove poi il dato A verrà shiftato o il dato B verrà moltiplicato per il coefficiente W. L'uscita del MUX_A viene portata anche al MUX_SUM dove, in base al valore del selettore SEL_MUX_SUM, prende il valore A (0) o il contenuto del registro REG_TMP_SUM (1); l'uscita di questo mux è collegata in input al blocco sommatore, per cui verrà sommata o sottratta al risultato presente sull'altra porta d'ingresso del sommatore, ossia il dato proveniente dal blocco moltiplicatore.

L'uscita del sommatore è poi inviata al blocco arrotondatore, il quale esegue l'arrotondamento al valore fractional su 20 bit più vicino, e il risultato viene successivamente salvato nei rispettivi registri d'uscita. Le uscite dei registri dove viene salvato A' sono collegate al MUX_OUT_A, analogamente le uscite dei registri in cui viene salvato B' sono collegate al MUX_OUT_B; lo scopo di tali Multiplexer è quello di mandare, sulla porta d'uscita, la componente (reale o immaginaria) corretta, per permettere a una Butterfly di un livello successivo di lavorare con i dati corretti.

Di seguito, sono descritti nello specifico i tre blocchi aritmetici:

3.2 Moltiplicatore

Sui due ingressi del moltiplicatore entrano da una parte A o B "INPUT_SX", mentre dall'altra ("INPUT_DX") entra sempre W. Il dato a sinistra viene sia shiftato di un bit a sinistra (ossia, moltiplicato per 2), ma anche moltiplicato per l'altro ingresso. Un mux stabilisce se far proseguire il risultato della moltiplicazione (1) o il valore d'ingresso shiftato (0) in base al selettore MPY_SHIFTN. L'uscita del mux è seguita dal registro di pipe, che poi manda l'uscita fuori dal blocco moltiplicatore.

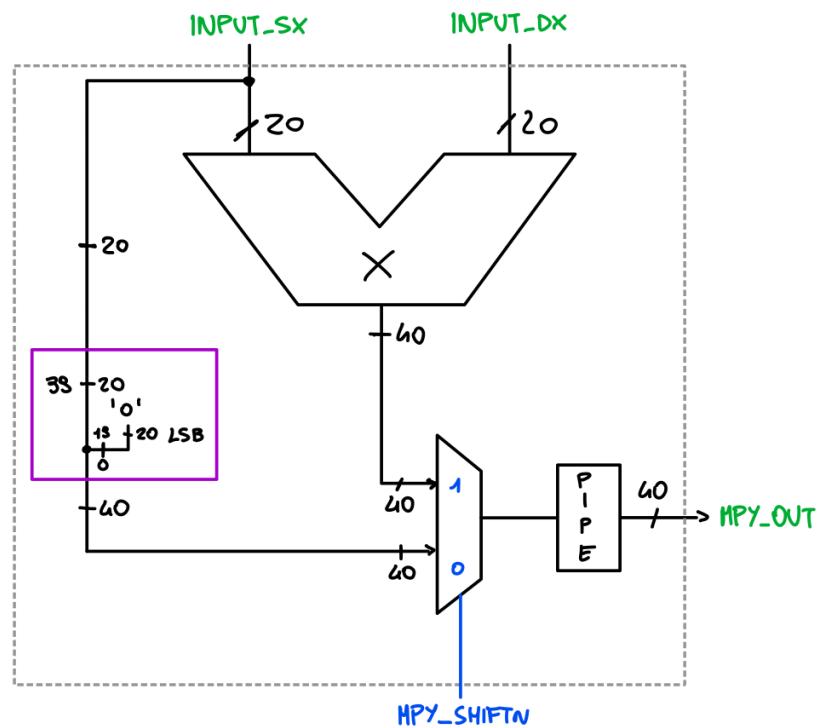


Figura 3: Moltiplicatore

3.3 Sommatore

Questo blocco implementa somma e sottrazione su entrambi i suoi ingressi, tramite l'uso dei segnali SUB_ADDN_RX e SUB_ADDN_SX, selettori dei due mux, per decidere se invertire o meno i dati in ingresso. Il C_IN è dato dall'OR dei due segnali di selezione. Anche qui è presente un registro di pipe.

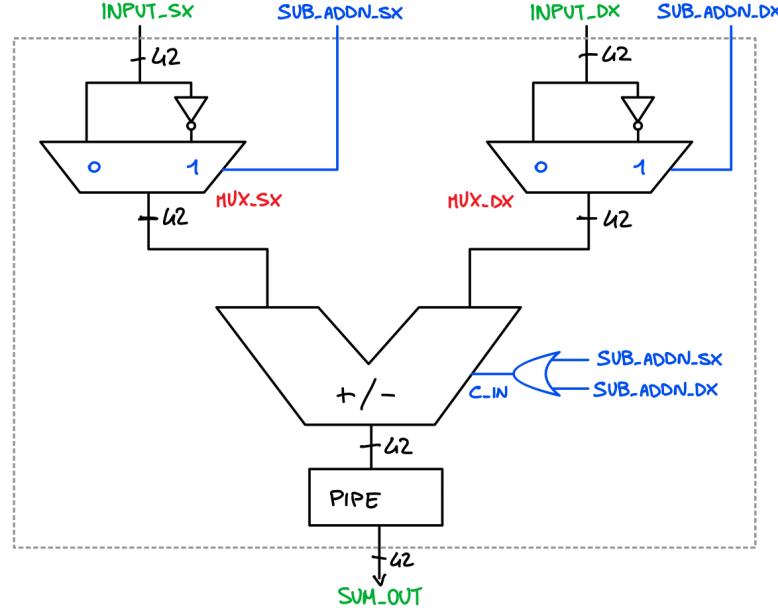


Figura 4: Sommatore

3.4 Arrotondatore Half-Up

È un blocco il cui unico ingresso è il dato da arrotondare, che viene dapprima diviso per 2 (shiftando di un bit verso destra) e poi gli viene sommato il valore 2^{-20} , che equivale a mezzo LSB della dinamica originaria. Il dato viene poi troncato e riportato su 20 bit. Anche qui è presente un registro di pipe.

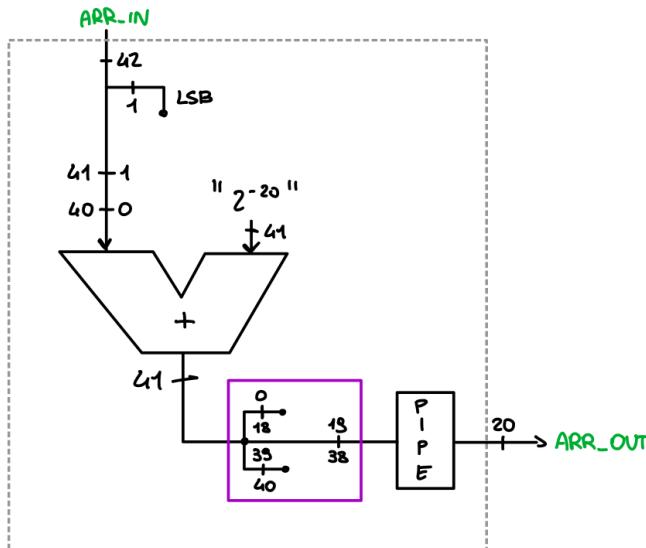


Figura 5: Arrotondatore

4 Porte e Architettura del sistema FFT_16x16

Il sistema che realizza l'FFT 16x16 presenta 32 porte d'ingresso a 20 bit e 16 porte d'uscita a 20 bit, interamente dedicate ai dati da processare; inoltre, sono presenti altre 3 porte d'ingresso a 1 bit, per i segnali di CLOCK, START e RESET, e una porta d'uscita a 1 bit, dedicata al segnale di DONE. A livello architettonico, l'intero sistema è realizzato semplicemente da 32 componenti Butterfly, raggruppati in 4 "stadi" da 8 Butterfly ciascuno. Gli stadi sono tutti collegati in cascata e le connessioni tra le Butterfly sono state realizzate secondo lo schema riportato di seguito:

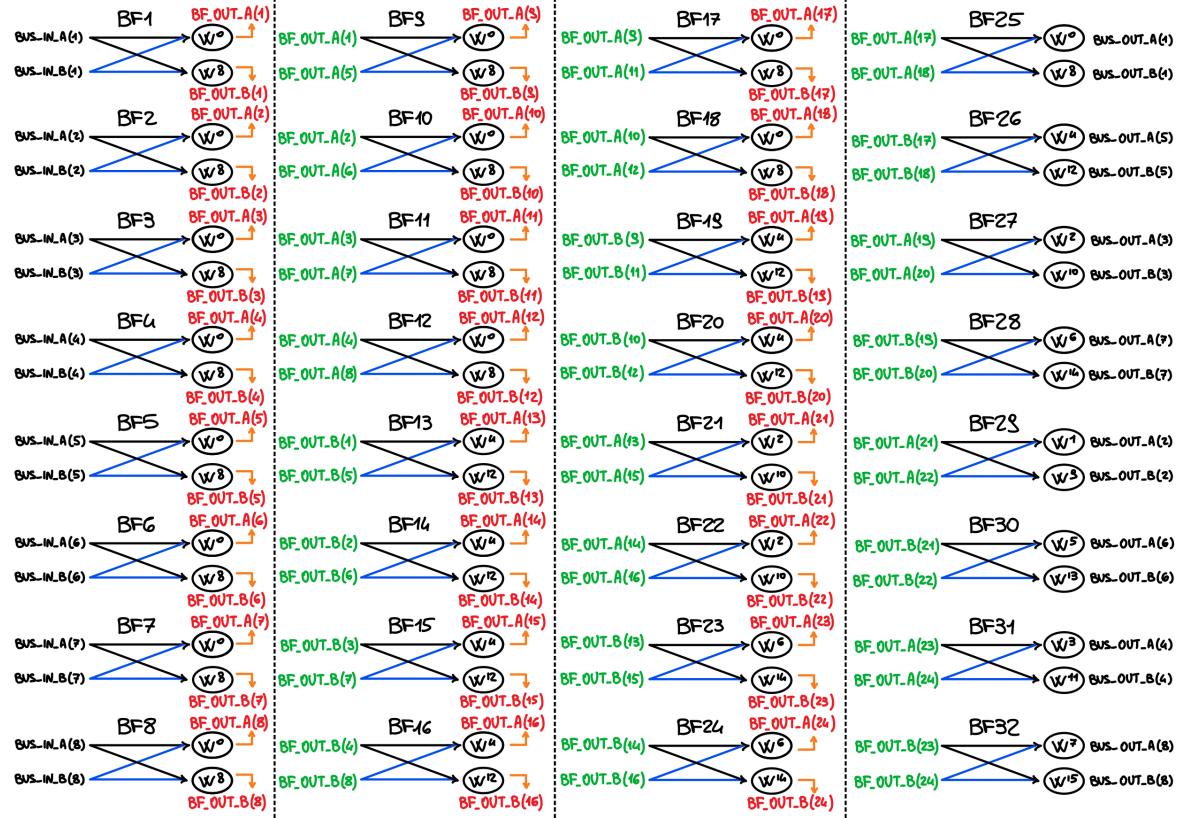


Figura 6: Architettura del sistema FFT

Si noti che gli ingressi delle Butterfly del primo stadio sono collegati direttamente ai bus d'ingresso (segnati in nero), così come le uscite delle Butterfly del quarto e ultimo stadio sono collegate direttamente ai bus d'uscita (anch'essi in nero); queste ultime sono collegate secondo un'ordine ben preciso, in modo tale da vedere in ordine le varie componenti spettrali.

Per quanto riguarda le interconnessioni tra Butterfly, per rendere più semplice la stesura del codice VHDL e la comprensione del sistema, è stato creato un data type custom, chiamato "BF_TYPE": questo non è altro che è un array di 24 componenti, ognuna delle quali è di tipo SIGNED su 20 bit. Sono, quindi, stati dichiarati due signal di tale tipologia, BF_OUT_A e BF_OUT_B, collegati ognuno alla opportuna uscita della corrispettiva Butterfly (segnati in rosso).

Con questi signal sono state realizzate le connessioni tra le uscite e gli ingressi (segnati in verde) di due stadi adiacenti. Per quanto riguarda i Twiddle Factors, la modalità con cui questi arrivano ad ogni Butterfly verrà discussa in seguito, nella sezione dedicata al Testbench.

5 Parallelismo

5.1 Parallelismo esterno

Avendo adottato per ogni Butterfly 4 porte di ingresso e 2 di uscita, risulta un totale di 6 porte per ogni Butterfly. Relativamente al blocco della FFT 16X16, come già anticipato, avremo 4 stadi in cascata, ognuno dei quali composto da 8 Butterfly che lavorano in parallelo. Considerando che ogni Butterfly del primo stadio ha 4 porte di ingresso e che ogni Butterfly dell'ultimo stadio ha 2 porte di uscita, complessivamente avremo 32 bus d'ingresso e 16 bus d'uscita, per un totale di 48 bus da 20 bit ciascuno. Il numero dei bus poteva essere ottimizzato, limitando i costi ma intaccando la velocità di esecuzione della Butterfly, per cui si è deciso che, avendo già risparmiato sul numero di componenti utilizzati, una spesa maggiore sui bus sia un giusto compromesso.

5.2 Parallelismo interno

Teoricamente il numero di bit di overflow che possiamo avere per l'intera FFT ($N=16$) è dato da $\lceil \log_2 N + 1 \rceil$, quindi 5 bit. Di questi, 4 bit di overflow vengono recuperati dividendo per 2 (all'interno del blocco Arrotondatore) ogni risultato prodotto dalla Butterfly: quindi, visto che ogni stadio recupera un bit, avendo 4 stadi che lavorano in cascata si riescono a recuperare 4 bit. Da questo scalamento deriva il fattore di scala finale, che è pari a 16 ($FS = 2^4$).

L'ultimo bit di overflow viene recuperato tramite l'inserzione del **bit di guardia**, utilizzato solo quando la dinamica di ingresso del primo stadio eccede da $[-0.5 \div +0.5]$. Avendo dati che entrano su 20 bit con una dinamica da $[-1 \div +1]$ il bit di guardia è stato inserito replicando l'MSB e shiftando tutti gli altri bit verso destra di una posizione. A questo punto i dati entrano nel primo livello con una corretta dinamica numerica $[-0.5 \div +0.5]$, ma dal secondo livello in avanti le Butterfly lavorano tra $[-1 \div +1]$. Questo ha portato a modificare la nostra prima idea sul parallelismo d'uscita del moltiplicatore, che teoricamente sarebbe potuto essere $2n - 1$ (39 bit teorici) ma che in realtà risulta essere su 2^n bit, più specificatamente 2*20 bit, quindi 40 bit in formato Q1.38 e avendo una dinamica in uscita dal moltiplicatore tra $[-2 \div +2]$. Mantenere 40 bit anziché 39 permette di avere un risultato corretto dell'operazione su tutti e quattro gli stadi, invece 39 bit sarebbero stati sufficienti solo per il corretto prodotto del primo stadio.

Proseguendo verso il sommatore, effettuando 3 somme consecutive con il massimo della dinamica ottenuta finora, per essere sicuri di non avere problemi di overflow la dinamica viene aumentata al range $[-8 \div +8]$ con formato Q3.38, con un aumento del parallelismo di due bit, da 40 bit a 42 bit. Infine, il dato in ingresso all'Arrotondatore è su 42 bit e questo viene subito shiftato a destra di una posizione (ossia, viene diviso per 2) scartando l'LSB; quindi, shiftando tutti gli altri bit a destra di una posizione si ottiene una dinamica da $[-4 \div +4]$. A questo punto viene sommato mezzo LSB, corrispondente a 2^{-20} , poi in uscita al sommatore viene effettuato il troncamento dei bit di peso 2^2 e 2^1 , replicate dell'MSB, e il troncamento anche dei bit di peso da 2^{-20} a 2^{-38} . Viene così ripristinata la dinamica originaria tra $[-1 \div +1]$, mantenendo inalterati i bit di peso fra 2^0 e 2^{-19} (che sarebbero, dal bit 38 al bit 19 in uscita dal sommatore), che sono il risultato già arrotondato.

Per venire incontro allo studio del parallelismo appena proposto, si è provveduto ad effettuare le seguenti operazioni:

- Estensione di segno di 2 bit sulla connessione tra uscita moltiplicatore e ingresso sommatore
- Estensione da 20 bit a 39, aggiungendo 19 zeri come LSB al signal che connette MUX_A e MUX_SUM ed ulteriore estensione di segno di 3 bit in ingresso a MUX_SUM, arrivando a 42 bit
- Estensione da 20 bit a 40 aggiungendo 20 zeri come LSB all'ingresso di sinistra del moltiplicatore, si noti che in tale operazione è presente già la moltiplicazione per 2, essendo che l'MSB, in uscita dal moltiplicatore, ha peso -2^1

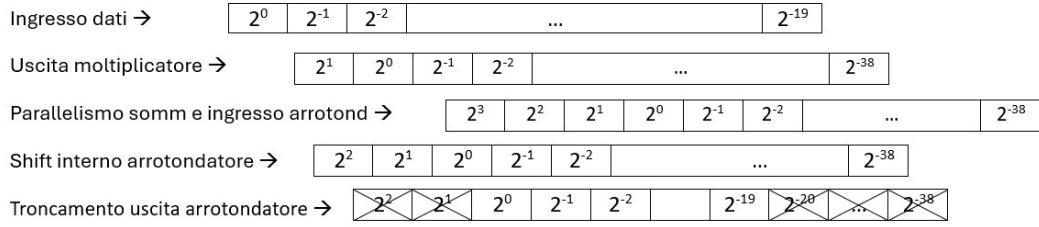


Figura 7: Parallelismo interno

6 Unità di controllo

6.1 CU schema generale

Control Unit

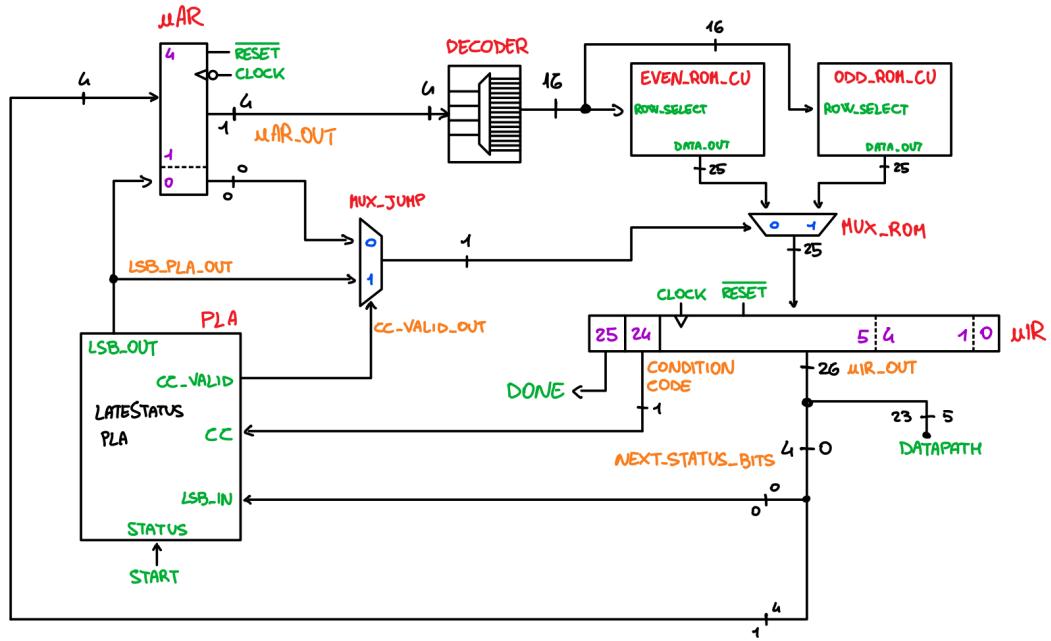


Figura 8: Control Unit

La CU è costituita da:

- **μAR** → il "micro Address Register" è un registro su 5 bit, contenente i bit che codificano lo stato corrente della Macchina. Di questi, i 4 bit più significativi finiscono in ingresso a un Decoder, il quale fornisce alle μROM la locazione del prossimo stato. L'LSB viene, invece, determinato dalla PLA, il cui funzionamento sarà descritto in seguito; tale bit, viene poi mandato in ingresso al MUX_JUMP.
- **μIR** → il "micro Instruction Register" è un registro su 26 bit organizzato come specificato in Figura 11. I bit 23-5 sono i comandi di gestione del Datapath. I bit 4-0 sono il "Next State" e il bit 25 è l'uscita DONE della Butterfly. Il bit 24 è il Condition Code (CC) ed insieme al bit 0 (LSB del Next State) entrano nella Late Status PLA.

- DECODER → riceve un ingresso su 4 bit dal μ AR fornendo sedici possibili uscite, una sola alla volta attiva, corrispondenti a ognuna delle 16 righe presenti nelle due μ ROM. Il decoder attiva una riga su entrambe le memorie indipendentemente dall'LSB.
- EVEN_ROM → μ ROM pari, contiene lo stato di **IDLE** e gli stati della **modalità "Continua"**
- ODD_ROM → μ ROM dispari, contiene gli stati della **modalità "One Shot"**
- MUX_ROM → dopo che il decoder ha selezionato una riga su entrambe le memorie, il mux di selezionare la riga della memoria pari (0) o quella della dispari (1)
- MUX_JUMP → funziona come selettore del mux MUX_ROM
- PLA → gestisce i salti, ossia determina il valore dell'LSB del Next State

Di seguito si riportano alcune considerazioni importanti adottate nella Control Unit.

Innanzitutto, l'architettura da noi adottata sfrutta un **Sequenziatore con indirizzamento esplicito** (evidente, visto l'impiego delle μ ROM, che contengono l'indirizzo nel Next State) e viene impiegata la tecnica del **"Late Status"**, che permette di posticipare la decisione dello stato futuro dopo il tempo di accesso alla μ ROM (ossia, vengono lette entrambe le memorie e poi, tramite l'LSB, viene selezionata una delle due); ciò comporta un notevole incremento delle prestazioni.

Il μ IR campiona i dati in ingresso sul fronte di salita, così come tutti gli altri componenti dell'Unita di Esecuzione; è preferibile che il fronte del clock su cui evolva questo registro sia in accordo con i componenti del Datapath, in quanto da questo vengono forniti tutti i comandi. Al contrario, il μ AR evolve sul fronte di discesa, ciò è necessario affinché ogni stato della macchina duri un colpo di clock. Il MUX_JUMP serve a velocizzare un salto "tardivo", ovvero se ad esempio l'utente attivasse tardi il segnale di START (anziché attivarlo sul fronte di salita del clock lo dà mezzo periodo dopo, quindi in corrispondenza del fronte di discesa), quello che potrebbe succedere è che il μ AR avrà già campionato i dati al suo ingresso, compreso l'LSB "errato".

Tuttavia, grazie alla tecnica del Late Status, che sfrutta l'impiego della Late Status PLA, è comunque possibile effettuare la scelta dello stato futuro correttamente e "saltare" all'ultimo momento, grazie all'LSB_OUT direttamente connesso al MUX_JUMP, selezionato mediante il selettore CC_Validation.

Segnale di DONE: dovendo distinguere le due modalità di funzionamento, alla fine della modalità Continua (nello stato CONT_6) viene attivato il segnale di DONE per un solo colpo di clock, e ciò avvisa l'utente che dei dati sono stati prodotti, ma l'esecuzione sta continuando. Al contrario, in modalità One Shot il segnale di DONE rimane attivo per 3 colpi di clock, negli stati FINISH, SEND_REAL e SEND_IMAG, avvisando l'utente che gli ultimi dati sono stati prodotti e non ne verranno forniti altri. La macchina ha finito totalmente l'esecuzione e ritornerà in stato di IDLE.

Tutti i valori codificati nello stato di IDLE, ad eccezione di CC=0, rappresentano i valori di default dei comandi, dove per comodità, simmetria e simpatia si è posto un unico bit a 1, quello centrale, in corrispondenza del comando MPY_SHIFTN.

DONE	CC	REG_IN LE	DATAPATH COMMANDS	REG_OUT LE	NEXT STATE
0	1	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 0 0 0

Figura 9: Default Values

6.2 Late Status PLA

Si occupa di gestire i salti e lo fa andando ad agire sull'LSB. È un semplice blocco costituito da una porta AND e un MUX in cui arriva l'LSB, bit 0 del μ IR, il cui valore passa all'LSB_OUT invariato o complementato, in base al valore del selettore; questo non è altro che una AND tra l'ingresso di "Status", nel nostro caso il solo segnale START, e il bit di CC, ossia il bit 24 del microIR:

- Se entrambi sono a 1 allora siamo nella condizione di avere un salto: il valore dell'LSB viene complementato dall'inverter e l'LSB_OUT assumerà tale valore, grazie al selettore pari a 1.
- Se CC non è posto a 1 ma accidentalmente dall'esterno arrivasse un segnale di START, questo comunque da solo non è in grado di permettere un salto, poiché il selettore sarebbe pari a 0. L'LSB_OUT manterebbe il valore dell'LSB e l'esecuzione continua sequenzialmente.

In uscita dalla PLA abbiamo due segnali:

- **CC_VALIDATION** → è il selettore del MUX_JUMP, permette la selezione tra l'uscita del μ AR (0) e l'uscita LSB_OUT della PLA (1); inoltre, rende possibile eseguire salti "tardivi"
- **LSB_OUT** → LSB che stabilisce quale sarà lo stato futuro, in quanto viene utilizzato come selettore del MUX_ROM; questo segnale va sia al MUX_JUMP che al μ AR.

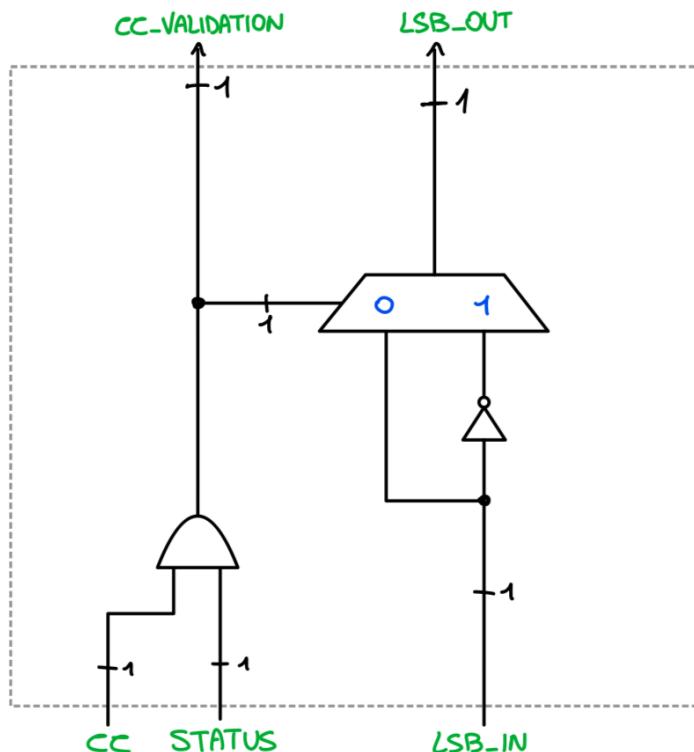


Figura 10: Late Status PLA

6.3 Struttura μ ROM

La Macchina ha 21 stati possibili, per cui sono necessari 5 bit per effettuare la codifica di questi. Tali bit danno vita a 32 celle di memoria, divisi equamente tra **μ ROM pari** e **μ ROM dispari**. Per cui si avranno due "sottocelle", ciascuna da 16 locazioni con parallelismo a 26 bit.

Ogni locazione ha la seguente struttura:

- Bit 25: DONE, collegato alla porta di uscita della BF; segnala che dei dati sono stati prodotti
- Bit 24: CC, bit che rende possibili i "salti"
- Bit 23-18: Enable dei registri di ingresso
- Bit 17-9: Comandi dei componenti combinatori del DP
- Bit 8-5: Enable dei registri di uscita
- Bit 4-0: Codifica del Next State

Ogni riga delle μ ROM viene selezionata mediante il Decoder, il quale ha al suo ingresso i 4 bit più significativi dello stato Stato Corrente.

Lo stato di IDLE è codificato con "0000-0", che rappresenta il punto di partenza dell'algoritmo all'interno della prima cella della μ ROM pari. Nella cella 0000-0 è quindi presente lo stato di IDLE, da questo è possibile prendere due direzioni:

- 1) Se non arriva il segnale di START allora si rimane in IDLE 0000-0 → LSB rimane invariato e il Next State contenuto dentro alla riga dell'IDLE è esattamente 0000-0. Quindi rimaniamo alla prima locazione della μ ROM pari.
- 2) Se arriva il segnale di START allora bisogna saltare verso lo stato GET_REAL 0000-1 → LSB cambia e diventa 1, ma i quattro bit del Next State contenuti nella riga dell'IDLE sono sempre 0000. Quindi ci siamo spostati alla locazione 0000-1, ossia la prima della μ ROM dispari.

È chiaro, quindi, che gli unici stati in cui si può ricadere al momento di un salto si differenziano per il solo LSB. Si agisce su di esso tramite la Late Status PLA, già descritta in precedenza.

Arrivati allo stato GET_REAL proseguiamo l'esecuzione in modalità sequenziale nella μ ROM dispari con i successivi stati (GET_IMAG, DO_M3...), fino ad arrivare allo stato DO_M5 ("01011"), contenuto nella cella 5 della μ ROM, in cui di nuovo è previsto un possibile salto:

- 1) Se non arriva lo START → il Next State contenuto nella riga dello stato DO_M5 è 0110-1, che punta alla sesta cella della μ ROM dispari, dove c'è lo stato DO_M6; quindi, si prosegue sequenzialmente la modalità **ONE SHOT** nella μ ROM dispari e non viene effettuato il salto.
- 2) Se arriva lo START → è necessario saltare per entrare in modalità **CONTINUA**, spostandosi nello stato CONT_1 della μ ROM pari, esattamente posto alla sesta locazione 0110-0 della suddetta memoria.

Ipotizzando di proseguire in sequenza nella modalità Continua, dallo stato CONT_1 fino a CONT_6 ("1011-0"), localizzato nell'undicesima locazione pari, anche qui risulta esserci un possibile salto:

- 1) Se arriva lo START → dobbiamo proseguire con modalità Continua. Il Next State puntato dallo stato CONT_6 è lo stato DO_M6 (cella 0110-1), tuttavia la PLA complementa l'LSB e prosegue la modalità Continua, rimanendo quindi nella μ ROM pari, dove nella locazione 0110-0 si trova lo stato CONT_1.

- 2) Se non arriva lo START → si prosegue in maniera sequenziale verso lo stato indicato dal Next State (0110-1), cioè la sesta cella della μ ROM dispari, corrispondente allo stato DO_M6; da qui si proseguirà l'esecuzione in maniera puramente sequenziale, in quanto non sono previsti ulteriori salti. Dopo lo stato SEND_IMAG, la macchina tornerà in IDLE e l'esecuzione sarà terminata.

Come già anticipato, dallo stato SEND_IMAG il Next State risulta essere l'IDLE, da cui ripartirà tutto l'algoritmo. Specifichiamo che, nonostante si vada da una μ ROM all'altra, questo non è un salto (non essendoci né lo START né il CC pari a 1), ma è semplicemente una prosecuzione sequenziale.

Di seguito è presentato il contenuto delle due μ ROM:

μ ROM parzi																NEXT STATE				CURRENT STATE							
DONE	CC	REG_IN LE				DATAPATH COMMANDS								REG_OUT LE				NEXT STATE					CURRENT STATE				
25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
6	0	0	1	0	1	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	
7	0	0	0	1	0	1	0	1	0	0	0	1	1	1	1	0	1	0	0	0	0	1	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0	0	0	1	0	0	1	0	
9	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0	1	0	1	0	0	
10	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	1	0	1	1	0	0	0	
11	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	1	0	1	0	
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

7 ASM chart, Protocollo e Timing Diagram

L'ASM chart assume un'importanza fondamentale, in quanto rende possibile:

- **Visualizzazione grafica dell'evoluzione della macchina:** ogni rettangolo rappresenta un determinato stato, il cui nome è riportato sopra di esso (in rosso), con la corrispettiva codifica (in viola). All'interno di questo, sono elencate le azioni eseguite dal Datapath (in nero) in quello stato, mentre ai lati i comandi attivi (in blu).
Al contrario, i rombi rappresentano i possibili salti, in funzione dello START.
- **Temporizzazione della macchina:** essendo che il dispositivo evolve da uno stato all'altro ogni colpo di clock, mediante l'ASM è possibile ricavare il giusto tempismo con cui l'utente deve fornire il segnale di START, i dati da processare e un ulteriore segnale di START per poter effettuare l'esecuzione dell'algoritmo in modalità continua.

Il **protocollo** che deve rispettare l'utente, affinché la macchina funzioni correttamente, è il seguente:

- 1) **ACCENSIONE:** bisogna, innanzitutto, fornire un segnale di clock alla frequenza desiderata; la macchina è stata testata con un clock a 10 MHz, per cui si consiglia di lavorare con un segnale a frequenza simile. Inoltre, affinché la macchina possa partire correttamente, bisogna fornire un segnale di RESET (attivo basso), in modo tale che si parta dallo stato di IDLE. Si noti che tale segnale è asincrono, per permettere alla macchina di essere inizializzata correttamente, anche nel caso in cui il clock non fosse ancora presente
- 2) **INIZIO ESECUZIONE:** bisogna fornire un segnale di START (attivo alto, della durata di un colpo di clock), affinché la macchina esca dallo stato di IDLE e cominci l'esecuzione dell'algoritmo. L'utente dovrà poi fornire la **parte REALE** dei dati da processare **dopo un colpo di clock dal segnale di START**, mentre la **parte IMMAGINARIA** deve essere fornita dopo un ulteriore colpo di clock, quindi **dopo due colpi di clock dal segnale di START**.
- 3) **ONE SHOT:** nel caso in cui l'utente abbia un singolo set di dati da processare, non è richiesta alcuna ulteriore azione da parte sua. Ogni signola Butterfly produce dei risultati parziali dopo 12 colpi di clock dal segnale di START ed essendoci 4 livelli che lavorano in cascata, i risultati finali saranno disponibili dopo 48 colpi di clock dal segnale di START. Viene, dunque, attivato un segnale di DONE che, come discusso in precedenza, sarà attivo per 3 colpi di clock: sui bus di uscita sarà disponibile la **parte REALE** dei dati prodotti **un colpo di clock dopo l'attivazione del DONE**, mentre la **parte IMMAGINARIA** sarà disponibile dopo un ulteriore colpo di clock, ossia **due colpi di clock dopo l'attivazione del DONE**.
- 4) **MODALITA' CONTINUA:** nel caso in cui l'utente abbia più di un set di dati da processare, egli dovrà **fornire un nuovo segnale di START dopo 6 colpi di clock dallo START precedente**, rispettando la temporizzazione (sia per lo START, che per i dati) già precedentemente descritta. In questo caso, dopo 48 colpi di clock dal primo segnale di START vengono prodotti i risultati del primo set di dati; essendo che verranno prodotti anche altri risultati e quindi la macchina continuerà a lavorare, il segnale di DONE durerà un solo colpo di clock, ma le **tempistiche con cui le parti reale e immaginaria del dato sono presenti sui bus di uscita sono analoghe a prima**. I risultati di un certo set di dati saranno disponibili in uscita sempre dopo **6 colpi di clock dopo i risultati del set precedente**. Solo nel caso dell'ultimo set di dati il segnale di DONE avrà la durata di 3 colpi di clock, per segnalare all'utente la fine totale dell'esecuzione, per cui la macchina tornerà poi in stato di IDLE.

Lo START risulta quindi essere sia il segnale con cui si avvia l'esecuzione dell'algoritmo, che il segnale che discrimina fra le due modalità.

Di seguito, è riportata l'ASM chart precedentemente descritta. Si noti che sono state differenziate, anche graficamente, le due possibili modalità di funzionamento: in particolare, nel caso di un'esecuzione **One Shot**, viene utilizzato solamente il "blocco superiore", percorso in **senso orario**. Al contrario, la **Modalità Continua** sfrutta anche l'impiego del "blocco inferiore", percorso in **senso antiorario**.

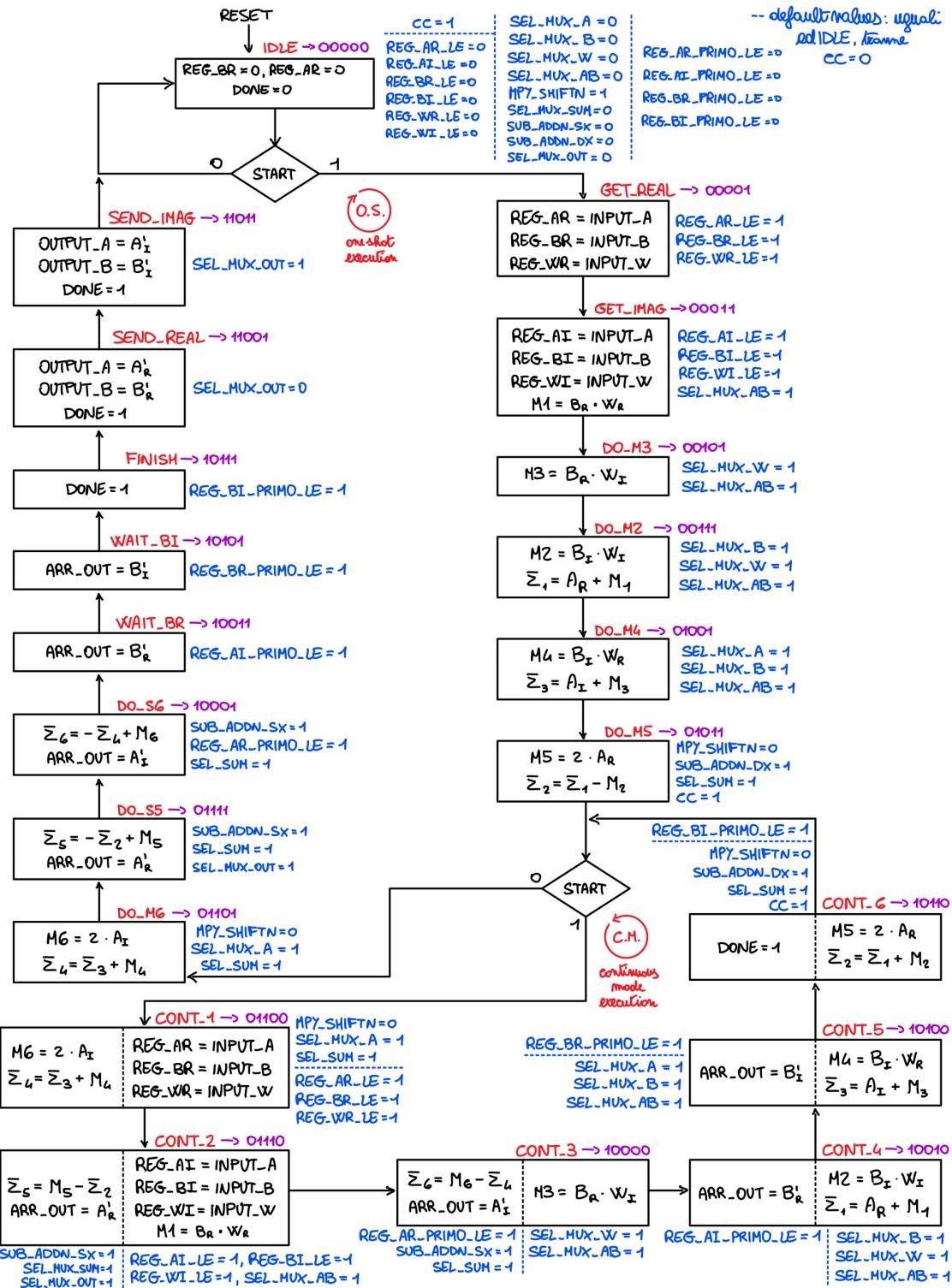


Figura 12: ASM chart

Di seguito, è riportato un breve Timing Diagram, che illustra l'inizio dell'esecuzione di una Butterfly del primo stadio, le cui porte e segnali sono riportati in blu e con l'indice (I).

Al fine di illustrare al meglio l'interfacciamento tra le Butterfly di stadi adiacenti, è rappresentata pure la parte finale di esecuzione della prima Butterfly; il segnale di DONE di questa permette a una Butterfly del secondo stadio, le cui porte e segnali sono riportati in arancione e con l'indice (II), di entrare in funzione.

Si noti come la prima Butterfly, subito dopo aver asserito il segnale di DONE, non vada subito in stato di IDLE, ma piloti adeguatamente il MUX_OUT, in modo tale da fornire alla seconda Butterfly i dati corretti.

Per finire, il segnale di DONE dura un solo colpo di clock perché, nella fase di progetto in cui è stato realizzato il Timing Diagram riportato, non si era ancora deciso di differenziare il DONE della Modalità Continua da quello di fine esecuzione.

Per essere rigorosi, sarebbe stato più corretto far durare tale segnale 3 colpi di clock.

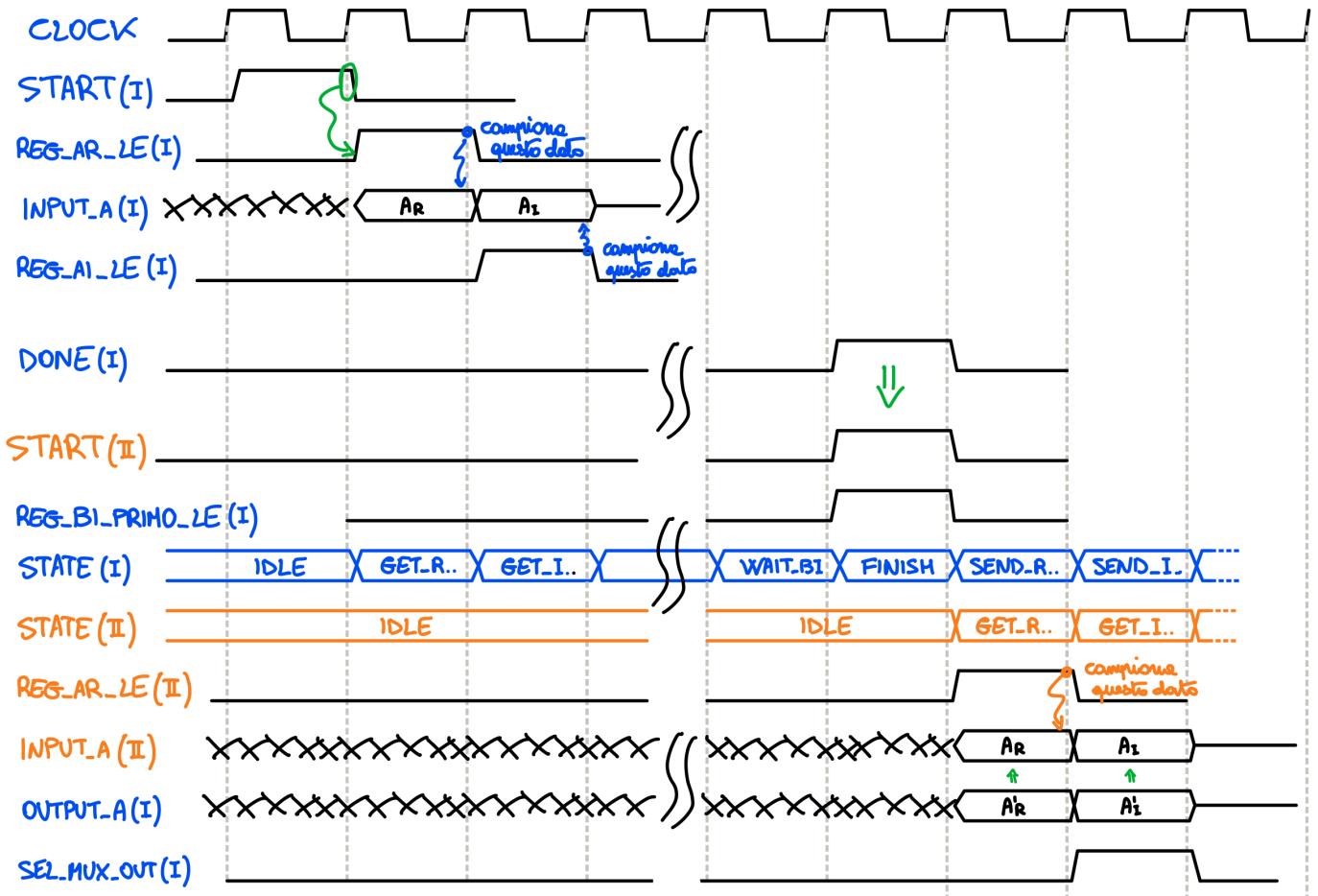


Figura 13: Timing Diagram

8 VHDL

In appendice, sono riportati tutti i codici VHDL sviluppati per il progetto. I componenti descritti, elencati in ordine gerarchico, sono i seguenti:

- **FFT_16x16:** Top-Level Entity del sistema; dispone 16 porte d'ingresso su 20 bit per i dati, di altrettante porte d'ingresso per le parti reali e immaginarie dei Twiddle Factors, nonché di 16 porte in uscita su 20 bit, dalle quali vengono forniti i dati nel dominio della frequenza. L'architettura è composta da 32 Butterfly, che sono opportunamente collegate tra loro.
- **Butterfly:** Processore che esegue operazioni aritmetiche sui due dati di ingresso; ogni Butterfly consta di 2 porte d'ingresso su 20 bit per i dati, di altrettante porte d'ingresso per le parti reali e immaginarie dei Twiddle Factors, nonché di 2 porte in uscita su 20 bit per i valori prodotti. A livello architettonico, la Butterfly utilizza 12 componenti, classificabili in due tipologie:
 - **Execution Unit:** è l'insieme dei componenti che si occupano della memorizzazione dei dati di ingresso, della loro rielaborazione aritmetica e dell'invio dei dati alla porta di uscita.
 - **Control Unit:** è l'insieme dei componenti che si occupano della corretta esecuzione dell'algoritmo; definisce lo stato della macchina, le sue evoluzioni e i comandi della EU.

Di seguito, sono elencati i componenti facenti parte della Execution Unit.

La struttura e il comportamento dei blocchi aritmetici sono stati già discussi in precedenza.

- **Multiplier_Shifter, Adder_Subtractor, Rounder**
- **Register_Nbit, MUX_2TO1_Nbit**

Di seguito, sono elencati i componenti facenti parte della Control Unit.

La struttura e il compito dei vari blocchi sono stati già discussi in precedenza.

- **Decoder_4to16, EVEN_ROM, ODD_ROM, LateStatus_PLA**
- **Register_Nbit_SLV, MUX_2TO1_Nbit_SLV, MUX_2TO1**

La stesura del codice VHDL ha portato alla luce diverse criticità nel sistema inizialmente progettato, tra le quali la necessità di aumentare il numero di porte di ingresso e uscite per permettere i giusti collegamenti tra le Butterfly, oltre al fatto di dover anche realizzare un'apposita struttura d'uscita (composta dai due MUX_OUT), che permettesse di mandare i dati sulla porta d'uscita in modo coerente con quanto succede alla porta d'ingresso.

9 Testbench

Per poter testare il funzionamento dell'intero sistema, è stato prodotto il file “TB_FFT_16x16.vhd”, dove è stato incluso come ”component” il sistema FFT_16x16 e sono stati descritti due processi esplicativi. Anche il codice del Testbench è riportato in Appendice.

Il primo processo prende il nome di ”**CLK**” e si occupa esclusivamente della generazione del clock, utilizzando una costante temporale chiamata ”TP”, dal valore di 100 ns. Il clock così generato ha una frequenza di 10 MHz.

Il secondo processo prende il nome di ”**Simulation**” e propone un esempio di utilizzo del sistema. In questo processo vengono dapprima definiti i valori dei coefficienti W, mandandoli nei rispettivi bus. Per fare ciò si è preferito numerare i bus da 1 a 8, rimanendo coerenti con quanto fatto con gli altri bus e le Butterfly stesse. In particolare sui bus saranno presenti i coefficienti nel seguente ordine:

$$\begin{aligned} \text{BUS W_1} &=> W^0, \text{BUS W_2} &=> W^4, \text{BUS W_3} &=> W^2, \text{BUS W_4} &=> W^6, \\ \text{BUS W_5} &=> W^1, \text{BUS W_6} &=> W^5, \text{BUS W_7} &=> W^3, \text{BUS W_8} &=> W^7 \end{aligned}$$

Dopodiché, viene eseguito un ciclo di inizializzazione della macchina, mandando un segnale di RESET. Viene poi attivato il segnale di START e, dopo un colpo di clock, vengono fornite le parti reali dei dati in ingresso ai bus; sono stati scritti tutti i vari vettori testati, in modo tale che sia possibile provarli uno alla volta, semplicemente rimuovendo il commento. Dopo un ulteriore colpo di clock, vengono mandate sul bus d’ingresso le parti immaginarie dei dati d’ingresso, sempre uguali a zero.

Al colpo di clock successivo, il bus viene mandato in alta impedenza, e ciò viene fatto sia per indicare che non c’è più bisogno di avere i dati sul bus, ma anche per distinguere visivamente questo istante temporale rispetto al caso iniziale (infatti, prima che i dati venissero mandati dall’utente, i valori presenti sui bus d’ingresso erano semplicemente non inizializzati).

In seguito, dopo altri 3 colpi di clock, è presente il codice che permette di testare due esecuzioni in modalità Continua. Se si volesse testare la modalità One Shot, basterebbe commentare i segnali di START.

Inoltre, è stata dichiarata una seconda costante temporale, chiamata ”**DELTA_DELAY**”, pari a 1fs. L’utilizzo di questa variabile è stato necessario perché abbiamo erroneamente disattivato il ritardo combinatorio che il simulatore dovrebbe inserire in automatico (pari alla risoluzione del simulatore stesso, e prende proprio il nome di Delta Delay) e, non sapendo più come riattivarlo, abbiamo escogitato questo simpatico ”stratagemma”, per permettere ai registri d’ingresso di campionare i dati correttamente.

Oltre al Testbench dell’intero sistema, sono stati simulati anche alcuni singoli componenti, ossia la Butterfly, il Multiplier_Shifter e l’Adder_Subtractor.

Nelle pagine seguenti, sono riportati degli estratti del Testbench. Per renderne più facile la fruizione, nelle immagini acquisite le porte del DUT hanno tutte il nome riportato in nero; in particolare, le forme d'onda dei segnali START, CLOCK, RESET e DONE sono pure di colore nero, mentre i colori delle forme d'onda dei bus dati assumono colori differenti, in base ai valori su essi presenti.

Inoltre, sono state aggiunte pure le porte START e DONE di 4 Butterfly, una per ogni stadio: primo stadio in nero, secondo in verde, terzo in rosso e quarto in blu.

Viene simulata un'esecuzione in modalità continua, con i seguenti vettori d'ingresso:

- 1) $X = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]/2$
- 2) $X = [-1 0 +1 0 -1 0 +1 0 -1 0 +1 0 -1 0 +1 0]/2$
- 3) $X = [+1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]/2$
- 4) $X = [-1 -1 +1 +1 -1 -1 +1 +1 -1 -1 +1 +1 -1 -1 +1 +1]/2$
- 5) $X = [+0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5]$
- 6) $X = [0 0 0 0 0 0 0 0 +0.75 0 0 0 0 0 0 0]/2$

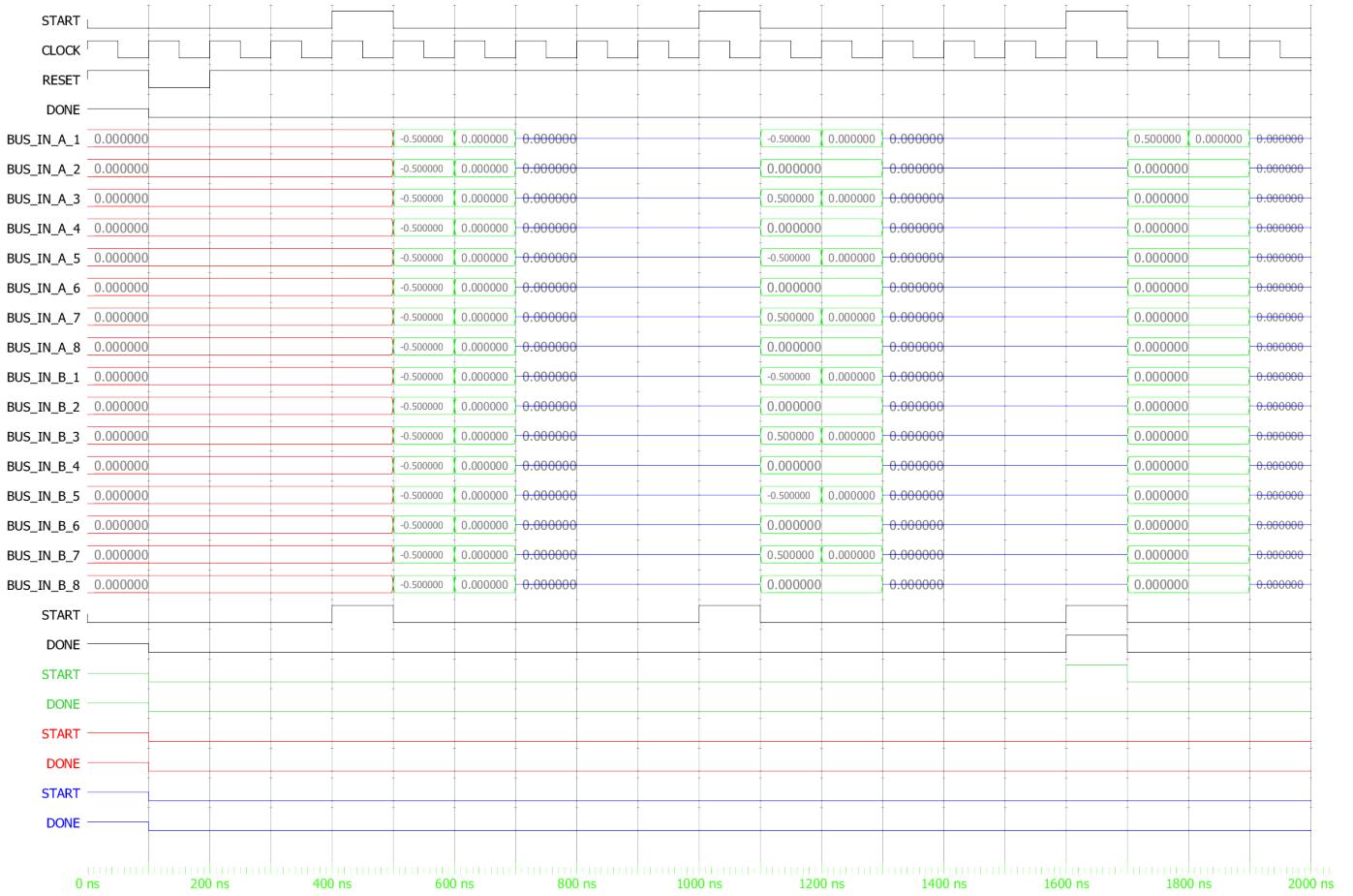


Figura 14: Accensione e inizio esecuzione. Vengono acquisiti i primi 3 vettori in ingresso. Dopo 1600 ns le Butterfly del primo stadio hanno prodotto dei risultati parziali, per cui entrano in gioco anche le Butterfly del secondo stadio



Figura 15: Valore dei W e delle uscite a inizio esecuzione. Si noti che i valori dei W sono fissi da inizio esecuzione, reso possibile dall'impiego di due bus separati per parte reale e parte immaginaria



Figura 16: Esecuzione intermedia tra 2000 ns e 4000 ns. Vengono acquisiti gli altri 3 vettori in ingresso. Le Butterfly del secondo stadio forniscono i primi dati, così anche il terzo stadio entra un funzione



Figura 17: Esecuzione intermedia tra 4800 ns e 6800 ns; le Butterfly del primo stadio hanno finito la loro esecuzione e vanno in stato di IDLE. Inoltre, a distanza di 6 colpi clock l'uno dall'altro, vengono forniti i primi 3 dati finali sui bus d'uscita: tutti i risultati prodotti sono corretti

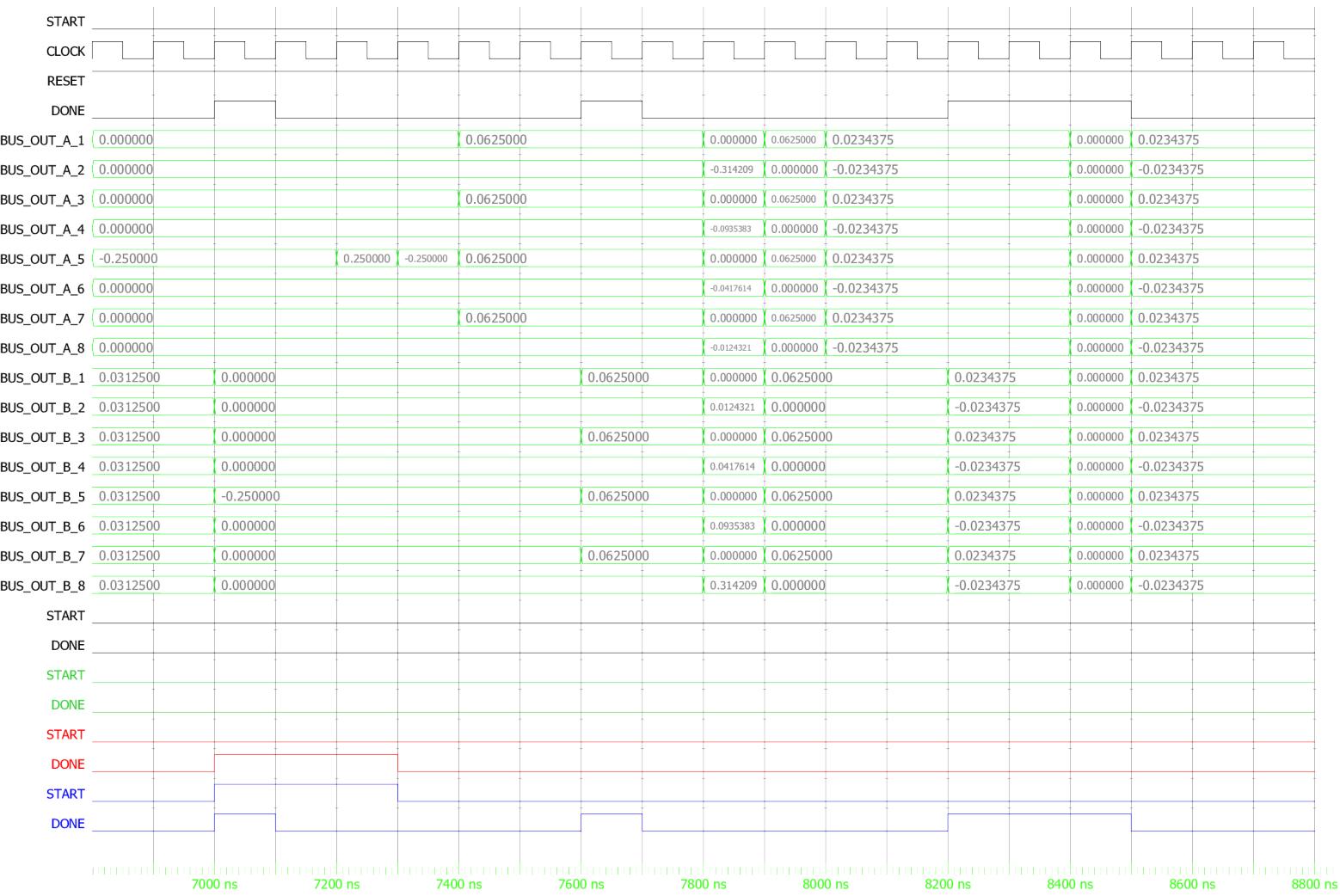


Figura 18: Esecuzione finale tra 6800 ns e 8800 ns; anche le Butterfly del secondo stadio hanno finito la loro esecuzione e vanno in stato di IDLE, seguite, dopo 6 colpi di clock, dalle Butterfly degli ultimi due stadi. Anche in questo caso, a distanza di 6 colpi clock l'uno dall'altro, vengono forniti gli ultimi 3 dati finali sui bus d'uscita: tutti i risultati prodotti sono corretti. Si noti come l'ultimo segnale di DONE duri 3 colpi di clock, a indicare che l'esecuzione complessiva è terminata e che l'intero dispositivo andrà in stato di IDLE

10 Prova Matlab

Al fine di testare la correttezza della Butterfly implementata, è stato scritto un codice Matlab che realizza le operazioni effettuate dalla Butterfly. Mediante delle variabili di appoggio e reiterati utilizzati della funzione implementata, è possibile effettuare una FFT 16x16 sui dati d'ingresso.

Per testare lo script, è sufficiente scrivere un vettore d'ingresso X con 16 componenti, dopodiché bisognerà copiare e incollare il contenuto del codice, dalla riga 18 (dove è presente il commento "PRIMO LIVELLO") fino alla fine, nella Command Window dell'ambiente Matlab.

Nel Workspace compariranno i vettori Y00-Y15, rappresentanti i campioni della variabile d'ingresso nel dominio della frequenza. Lo script implementato è riportato di seguito:

```

1 function [A_primo, B_primo] = butterfly(A, B, m)
2     A_fixed = fi(A, 1, 23, 19);
3     B_fixed = fi(B, 1, 23, 19);
4     W = fi(exp(-1i*2*pi*(m/16)), 1, 23, 19);
5     A_primo = fi((A_fixed + B_fixed.*W)/2, 1, 23, 19);
6     B_primo = fi((A_fixed - B_fixed.*W)/2, 1, 23, 19);
7 end
8
9 %{
10 X = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]./2;
11 X = [-1 0 +1 0 -1 0 +1 0 -1 0 +1 0 -1 0 +1 0]./2;
12 X = [+1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]./2;
13 X = [-1 -1 +1 +1 -1 -1 +1 +1 -1 -1 +1 +1 -1 +1 +1]./2;
14 X = [+0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5];
15 X = [0 0 0 0 0 0 0 +0.75 0 0 0 0 0 0 0 0]./2;
16
17 %PRIMO LIVELLO
18 [BF1_A, BF1_B] = butterfly(X(1), X(9), 0);
19 [BF2_A, BF2_B] = butterfly(X(2), X(10), 0);
20 [BF3_A, BF3_B] = butterfly(X(3), X(11), 0);
21 [BF4_A, BF4_B] = butterfly(X(4), X(12), 0);
22 [BF5_A, BF5_B] = butterfly(X(5), X(13), 0);
23 [BF6_A, BF6_B] = butterfly(X(6), X(14), 0);
24 [BF7_A, BF7_B] = butterfly(X(7), X(15), 0);
25 [BF8_A, BF8_B] = butterfly(X(8), X(16), 0);
26 %SECONDO LIVELLO
27 [BF9_A, BF9_B] = butterfly(BF1_A, BF5_A, 0);
28 [BF10_A, BF10_B] = butterfly(BF2_A, BF6_A, 0);
29 [BF11_A, BF11_B] = butterfly(BF3_A, BF7_A, 0);
30 [BF12_A, BF12_B] = butterfly(BF4_A, BF8_A, 0);
31 [BF13_A, BF13_B] = butterfly(BF1_B, BF5_B, 4);
32 [BF14_A, BF14_B] = butterfly(BF2_B, BF6_B, 4);
33 [BF15_A, BF15_B] = butterfly(BF3_B, BF7_B, 4);
34 [BF16_A, BF16_B] = butterfly(BF4_B, BF8_B, 4);
35 %TERZO LIVELLO
36 [BF17_A, BF17_B] = butterfly(BF9_A, BF11_A, 0);
37 [BF18_A, BF18_B] = butterfly(BF10_A, BF12_A, 0);
38 [BF19_A, BF19_B] = butterfly(BF9_B, BF11_B, 4);
39 [BF20_A, BF20_B] = butterfly(BF10_B, BF12_B, 4);
40 [BF21_A, BF21_B] = butterfly(BF13_A, BF15_A, 2);
41 [BF22_A, BF22_B] = butterfly(BF14_A, BF16_A, 2);

```

```

42 [BF23_A, BF23_B] = butterfly(BF13_B, BF15_B, 6);
43 [BF24_A, BF24_B] = butterfly(BF14_B, BF16_B, 6);
44 %QUARTO LIVELLO
45 [BF25_A, BF25_B] = butterfly(BF17_A, BF18_A, 0);
46 [BF26_A, BF26_B] = butterfly(BF17_B, BF18_B, 4);
47 [BF27_A, BF27_B] = butterfly(BF19_A, BF20_A, 2);
48 [BF28_A, BF28_B] = butterfly(BF19_B, BF20_B, 6);
49 [BF29_A, BF29_B] = butterfly(BF21_A, BF22_A, 1);
50 [BF30_A, BF30_B] = butterfly(BF21_B, BF22_B, 5);
51 [BF31_A, BF31_B] = butterfly(BF23_A, BF24_A, 3);
52 [BF32_A, BF32_B] = butterfly(BF23_B, BF24_B, 7);
53
54 Y_tmp = [BF25_A, BF29_A, BF27_A, BF31_A, BF26_A, BF30_A, BF28_A, BF32_A, BF25_B
      , BF29_B, BF27_B, BF31_B, BF26_B, BF30_B, BF28_B, BF32_B].*16;
55
56 Y00 = double(Y_tmp(1));
57 Y01 = double(Y_tmp(2));
58 Y02 = double(Y_tmp(3));
59 Y03 = double(Y_tmp(4));
60 Y04 = double(Y_tmp(5));
61 Y05 = double(Y_tmp(6));
62 Y06 = double(Y_tmp(7));
63 Y07 = double(Y_tmp(8));
64 Y08 = double(Y_tmp(9));
65 Y09 = double(Y_tmp(10));
66 Y10 = double(Y_tmp(11));
67 Y11 = double(Y_tmp(12));
68 Y12 = double(Y_tmp(13));
69 Y13 = double(Y_tmp(14));
70 Y14 = double(Y_tmp(15));
71 Y15 = double(Y_tmp(16));
72 %

```

Listing 1: Script MATLAB per il calcolo della FFT

11 Conclusioni

Per concludere, il sistema da noi realizzato risulta essere perfettamente funzionante, come testimoniato dalle simulazioni effettuate e riportate in precedenza. Le scelte progettuali da noi adottate sono frutto della ricerca del giusto equilibrio: ore e ore di studio ci hanno portato a tale soluzione, la quale ci sembra il miglior Trade-Off possibile tra il contenimento dei costi per l'Hardware e le prestazioni del sistema, rendendolo particolarmente appetibile a un eventuale acquirente. Nella fase finale del progetto sono state effettuate anche alcune ottimizzazioni che non hanno richiesto troppo tempo, tra cui la rimozione della porta di RESET dei registri del Datapath, in quanto ciò permette di utilizzare un componente più piccolo, diminuendo l'area occupata e, di conseguenza, anche i costi. Altre ottimizzazioni non sono state eseguite perché troppo dispendiose dal punto di vista del tempo da dedicare e non troppo vantaggiose dal punto di vista dei costi: una di queste, sarebbe la diminuzione del parallelismo delle μ ROM. In particolare, sarebbe possibile recuperare 4 bit, in quanto i comandi di Enable dei registri d'ingresso assumono gli stessi valori; sarebbe quindi possibile raggruppare i 6 comandi distinti in 2 soli comandi, uno per i registri dedicati alla parte reale dei dati, l'altro per i registri dedicati alla parte immaginaria. Si diminuirebbe, così, il parallelismo della μ ROM a 22 bit. Speriamo di poter fare ciò in una futura versione del sistema :)

12 Appendice: codici VHDL

12.1 TB_FFT_16x16.vhd

```

1  -- QUESTO CODICE E' STATO PRODOTTO DA:
2  -- Fichera Antonio, matricola: 337213
3  -- Vannelli Elisabetta, matricola: 346477
4
5 LIBRARY IEEE;
6 USE IEEE.STD_LOGIC_1164.ALL;
7 USE IEEE.NUMERIC_STD.ALL;
8
9 ENTITY TB_FFT_16x16 IS
10 END ENTITY;
11
12 ARCHITECTURE Behavioural OF TB_FFT_16x16 IS
13
14 COMPONENT FFT_16x16 IS
15   PORT(START : IN STD_LOGIC;
16         CLOCK : IN STD_LOGIC;
17         RESET : IN STD_LOGIC;
18         -- BUS A
19         BUS_IN_A_1 : IN SIGNED(19 DOWNTO 0);
20         BUS_IN_A_2 : IN SIGNED(19 DOWNTO 0);
21         BUS_IN_A_3 : IN SIGNED(19 DOWNTO 0);
22         BUS_IN_A_4 : IN SIGNED(19 DOWNTO 0);
23         BUS_IN_A_5 : IN SIGNED(19 DOWNTO 0);
24         BUS_IN_A_6 : IN SIGNED(19 DOWNTO 0);
25         BUS_IN_A_7 : IN SIGNED(19 DOWNTO 0);
26         BUS_IN_A_8 : IN SIGNED(19 DOWNTO 0);
27         -- BUS B
28         BUS_IN_B_1 : IN SIGNED(19 DOWNTO 0);
29         BUS_IN_B_2 : IN SIGNED(19 DOWNTO 0);
30         BUS_IN_B_3 : IN SIGNED(19 DOWNTO 0);
31         BUS_IN_B_4 : IN SIGNED(19 DOWNTO 0);
32         BUS_IN_B_5 : IN SIGNED(19 DOWNTO 0);
33         BUS_IN_B_6 : IN SIGNED(19 DOWNTO 0);
34         BUS_IN_B_7 : IN SIGNED(19 DOWNTO 0);
35         BUS_IN_B_8 : IN SIGNED(19 DOWNTO 0);
36         -- BUS WR
37         BUS_IN_WR_1 : IN SIGNED(19 DOWNTO 0);
38         BUS_IN_WR_2 : IN SIGNED(19 DOWNTO 0);
39         BUS_IN_WR_3 : IN SIGNED(19 DOWNTO 0);
40         BUS_IN_WR_4 : IN SIGNED(19 DOWNTO 0);
41         BUS_IN_WR_5 : IN SIGNED(19 DOWNTO 0);
42         BUS_IN_WR_6 : IN SIGNED(19 DOWNTO 0);
43         BUS_IN_WR_7 : IN SIGNED(19 DOWNTO 0);
44         BUS_IN_WR_8 : IN SIGNED(19 DOWNTO 0);
45         -- BUS WI
46         BUS_IN_WI_1 : IN SIGNED(19 DOWNTO 0);
47         BUS_IN_WI_2 : IN SIGNED(19 DOWNTO 0);
48         BUS_IN_WI_3 : IN SIGNED(19 DOWNTO 0);
49         BUS_IN_WI_4 : IN SIGNED(19 DOWNTO 0);

```

```

50      BUS_IN_WI_5 : IN SIGNED(19 DOWNTO 0);
51      BUS_IN_WI_6 : IN SIGNED(19 DOWNTO 0);
52      BUS_IN_WI_7 : IN SIGNED(19 DOWNTO 0);
53      BUS_IN_WI_8 : IN SIGNED(19 DOWNTO 0);
54      -- BUS A'
55      BUS_OUT_A_1 : OUT SIGNED(19 DOWNTO 0);
56      BUS_OUT_A_2 : OUT SIGNED(19 DOWNTO 0);
57      BUS_OUT_A_3 : OUT SIGNED(19 DOWNTO 0);
58      BUS_OUT_A_4 : OUT SIGNED(19 DOWNTO 0);
59      BUS_OUT_A_5 : OUT SIGNED(19 DOWNTO 0);
60      BUS_OUT_A_6 : OUT SIGNED(19 DOWNTO 0);
61      BUS_OUT_A_7 : OUT SIGNED(19 DOWNTO 0);
62      BUS_OUT_A_8 : OUT SIGNED(19 DOWNTO 0);
63      -- BUS B'
64      BUS_OUT_B_1 : OUT SIGNED(19 DOWNTO 0);
65      BUS_OUT_B_2 : OUT SIGNED(19 DOWNTO 0);
66      BUS_OUT_B_3 : OUT SIGNED(19 DOWNTO 0);
67      BUS_OUT_B_4 : OUT SIGNED(19 DOWNTO 0);
68      BUS_OUT_B_5 : OUT SIGNED(19 DOWNTO 0);
69      BUS_OUT_B_6 : OUT SIGNED(19 DOWNTO 0);
70      BUS_OUT_B_7 : OUT SIGNED(19 DOWNTO 0);
71      BUS_OUT_B_8 : OUT SIGNED(19 DOWNTO 0);
72      -- GLOBAL DONE
73      DONE : OUT STD_LOGIC);
74  END COMPONENT;
75
76  SIGNAL START_SIM, CLOCK_SIM, RESET_SIM, DONE_SIM : STD_LOGIC;
77  TYPE DATA_TYPE IS ARRAY(1 TO 8) OF SIGNED(19 DOWNTO 0);
78  SIGNAL INPUT_A_SIM, INPUT_B_SIM, INPUT_WR_SIM, INPUT_WI_SIM, OUTPUT_A_SIM, OUTPUT_B_SIM :
79    DATA_TYPE;
80  CONSTANT TP : TIME := 100 ns; -- frequenza del Clock = 10 MHz
81  CONSTANT DELTA_DELAY : TIME := 1 fs;
82
83  BEGIN
84    CLK : PROCESS
85    BEGIN
86      CLOCK_SIM <= '1';
87      WAIT FOR TP/2;
88      CLOCK_SIM <= '0';
89      WAIT FOR TP/2;
90    END PROCESS CLK;
91
92    Simulation : PROCESS
93    BEGIN
94      INPUT_A_SIM <= (OTHERS => (OTHERS => 'U'));
95      INPUT_B_SIM <= (OTHERS => (OTHERS => 'U'));
96
97      INPUT_WR_SIM(1) <= (19 => '0', OTHERS => '1');
98      INPUT_WI_SIM(1) <= (OTHERS => '0');
99
100     INPUT_WR_SIM(2) <= (OTHERS => '0');
101     INPUT_WI_SIM(2) <= (19 => '1', OTHERS => '0');
102

```

```

103 INPUT_WR_SIM(3) <= "01011010100000101000";
104 INPUT_WI_SIM(3) <= "10100101011111011000";
105
106 INPUT_WR_SIM(4) <= "10100101011111011000";
107 INPUT_WI_SIM(4) <= "10100101011111011000";
108
109 INPUT_WR_SIM(5) <= "01110110010000011011";
110 INPUT_WI_SIM(5) <= "11001111000001000100";
111
112 INPUT_WR_SIM(6) <= "11001111000001000100";
113 INPUT_WI_SIM(6) <= "10001001101111100101";
114
115 INPUT_WR_SIM(7) <= "00110000111110111100";
116 INPUT_WI_SIM(7) <= "10001001101111100101";
117
118 INPUT_WR_SIM(8) <= "10001001101111100101";
119 INPUT_WI_SIM(8) <= "11001111000001000100";
120
121
122 RESET_SIM <= '1';
123 START_SIM <= '0';
124 WAIT FOR TP;
125 RESET_SIM <= '0'; -- IL RESET E' ATTIVO BASSO, INIZIALMENTE LO ATTIVO PER PERMETTERE ALLA
126     MACCHINA DI ANDARE NELLO STATO DI IDLE
127 START_SIM <= '0';
128 WAIT FOR TP;
129 RESET_SIM <= '1'; -- DISATTIVO IL RESET PER FARE PARTIRE LA MACCHINA
130 WAIT FOR 2*TP;
131 START_SIM <= '1';
132 WAIT FOR TP;
133 START_SIM <= '0'; -- STATO GET_REAL
134
135 -- ENTRANO LE PARTI REALI DEI DATI
136 -- ****
137 -- 1) CASO X = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1] / 2
138 --INPUT_A_SIM <= (OTHERS => (19 => '1', 18 => '1', OTHERS => '0'));
139 --INPUT_B_SIM <= (OTHERS => (19 => '1', 18 => '1', OTHERS => '0'));
140 -- ****
141 -- ****
142 -- 2) CASO X = [-1 0 +1 0 -1 0 +1 0 -1 0 +1 0 -1 0 +1 0] / 2
143     INPUT_A_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
144         5 => (19 => '1', 18 => '1', OTHERS => '0'),
145         3 => (19 => '0', 18 => '1', OTHERS => '0'),
146         7 => (19 => '0', 18 => '1', OTHERS => '0'),
147             OTHERS => (OTHERS => '0'));
148     INPUT_B_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
149         5 => (19 => '1', 18 => '1', OTHERS => '0'),
150         3 => (19 => '0', 18 => '1', OTHERS => '0'),
151         7 => (19 => '0', 18 => '1', OTHERS => '0'),
152             OTHERS => (OTHERS => '0'));
153 -- ****
154 -- ****

```

```

156 -- 3) CASO X = [+1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]/2
157 -- INPUT_A_SIM <= (1 => (19 => '0', 18 => '1', OTHERS => '0'),
158 --     OTHERS => (OTHERS => '0'));
159 -- INPUT_B_SIM <= (OTHERS => (OTHERS => '0'));
160 -- ****
161 --
162 -- ****
163 -- 4) CASO X = [-1 -1 +1 +1 -1 -1 +1 +1 -1 -1 +1 +1]/2
164 -- INPUT_A_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
165 --     2 => (19 => '1', 18 => '1', OTHERS => '0'),
166 --     5 => (19 => '1', 18 => '1', OTHERS => '0'),
167 --     6 => (19 => '1', 18 => '1', OTHERS => '0'),
168 --     OTHERS => (19 => '0', 18 => '1', OTHERS => '0'));
169 -- INPUT_B_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
170 --     2 => (19 => '1', 18 => '1', OTHERS => '0'),
171 --     5 => (19 => '1', 18 => '1', OTHERS => '0'),
172 --     6 => (19 => '1', 18 => '1', OTHERS => '0'),
173 --     OTHERS => (19 => '0', 18 => '1', OTHERS => '0'));
174 -- ****
175 --
176 -- ****
177 -- 5) CASO X = [+0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5]/2
178 -- INPUT_A_SIM <= (OTHERS => (19 => '0', 18 => '0', 17 => '1', OTHERS => '0'));
179 -- INPUT_B_SIM <= (1 => (19 => '0', 18 => '0', 17 => '1', OTHERS => '0'),
180 --     OTHERS => (19 => '1', 18 => '1', 17 => '1', OTHERS => '0'));
181 --
182 -- STESSO CASO MA CON DINAMICA COMPLETA
183 -- INPUT_A_SIM <= (OTHERS => (19 => '0', 18 => '1', OTHERS => '0'));
184 -- INPUT_B_SIM <= (1 => (19 => '0', 18 => '1', OTHERS => '0'),
185 --     OTHERS => (19 => '1', 18 => '1', OTHERS => '0'));
186 -- ****
187 --
188 -- ****
189 -- 6) CASO X = [0 0 0 0 0 0 0 +0.75 0 0 0 0 0 0 0]/2
190 -- INPUT_A_SIM <= (OTHERS => (OTHERS => '0'));
191 -- INPUT_B_SIM <= (1 => (19 => '0', 18 => '0', 17 => '1', 16 => '1', OTHERS => '0'),
192 --     OTHERS => (OTHERS => '0'));
193 -- ****
194 --
195 --
196 WAIT FOR TP;
197 WAIT FOR DELTA_DELAY;
198 --
199 -- ENTRANO LE PARTI IMMAGINARIE DEI DATI -- > STATO GET_IMAG
200 INPUT_A_SIM <= (OTHERS => (OTHERS => '0'));
201 INPUT_B_SIM <= (OTHERS => (OTHERS => '0'));
202 --
203 WAIT FOR TP;
204 WAIT FOR DELTA_DELAY;
205 -- LIBERO IL BUS -- STATO DO_M3
206 --
207 INPUT_A_SIM <= (OTHERS => (OTHERS => 'Z'));
208 INPUT_B_SIM <= (OTHERS => (OTHERS => 'Z'));

```

```

209 WAIT FOR 3*TP;
210 WAIT FOR DELTA_DELAY;
211
212
213 -- ****
214 -- INIZIO MODALITA' CONTINUA, COMMENTARLA TUTTA SE SI VUOLE ESEGUIRE UNA ONE SHOT
215 -- ****
216
217 -- STATO DO_M5
218 START_SIM <= '1'; -- PER POTER ANDARE IN MODALITA' CONTINUA
219 WAIT FOR TP;
220 WAIT FOR DELTA_DELAY;
221 START_SIM <= '0'; -- STATO CONT_1
222
223 -- ENTRANO LE PARTI REALI DEI DATI
224 -- ****
225 -- 1) CASO X = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1] / 2
226 -- INPUT_A_SIM <= (OTHERS => (19 => '1', 18 => '1', OTHERS => '0'));
227 -- INPUT_B_SIM <= (OTHERS => (19 => '1', 18 => '1', OTHERS => '0'));
228 -- ****
229
230 -- ****
231 -- 2) CASO X = [-1 0 +1 0 -1 0 +1 0 -1 0 +1 0 -1 0 +1 0] / 2
232 -- INPUT_A_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
233 --      5 => (19 => '1', 18 => '1', OTHERS => '0'),
234 --      3 => (19 => '0', 18 => '1', OTHERS => '0'),
235 --      7 => (19 => '0', 18 => '1', OTHERS => '0'),
236 --      OTHERS => (OTHERS => '0'));
237 -- INPUT_B_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
238 --      5 => (19 => '1', 18 => '1', OTHERS => '0'),
239 --      3 => (19 => '0', 18 => '1', OTHERS => '0'),
240 --      7 => (19 => '0', 18 => '1', OTHERS => '0'),
241 --      OTHERS => (OTHERS => '0'));
242 -- ****
243
244 -- ****
245 -- 3) CASO X = [+1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] / 2
246 -- INPUT_A_SIM <= (1 => (19 => '0', 18 => '1', OTHERS => '0'),
247 --      OTHERS => (OTHERS => '0'));
248 -- INPUT_B_SIM <= (OTHERS => (OTHERS => '0'));
249 -- ****
250
251 -- ****
252 -- 4) CASO X = [-1 -1 +1 +1 -1 -1 +1 +1 -1 -1 +1 +1] / 2
253 -- INPUT_A_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
254 --      2 => (19 => '1', 18 => '1', OTHERS => '0'),
255 --      5 => (19 => '1', 18 => '1', OTHERS => '0'),
256 --      6 => (19 => '1', 18 => '1', OTHERS => '0'),
257 --      OTHERS => (19 => '0', 18 => '1', OTHERS => '0'));
258 -- INPUT_B_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
259 --      2 => (19 => '1', 18 => '1', OTHERS => '0'),
260 --      5 => (19 => '1', 18 => '1', OTHERS => '0'),
261 --      6 => (19 => '1', 18 => '1', OTHERS => '0'),
262 --      OTHERS => (19 => '0', 18 => '1', OTHERS => '0'));

```

```

263 -- ****
264
265 -- ****
266 -- 5) CASO X = [+0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5] /2
267 -- INPUT_A_SIM <= (OTHERS => (19 => '0', 18 => '0', 17 => '1', OTHERS => '0'));
268 -- INPUT_B_SIM <= (1 => (19 => '0', 18 => '0', 17 => '1', OTHERS => '0'),
269 -- OTHERS => (19 => '1', 18 => '1', 17 => '1', OTHERS => '0'));
270
271 -- STESSO CASO MA CON DINAMICA COMPLETA
272 INPUT_A_SIM <= (OTHERS => (19 => '0', 18 => '1', OTHERS => '0'));
273 INPUT_B_SIM <= (1 => (19 => '0', 18 => '1', OTHERS => '0'),
274 -- OTHERS => (19 => '1', 18 => '1', OTHERS => '0'));
275 -- ****
276
277 -- ****
278 -- 6) CASO X = [0 0 0 0 0 0 0 +0.75 0 0 0 0 0 0] /2
279 -- INPUT_A_SIM <= (OTHERS => (OTHERS => '0'));
280 -- INPUT_B_SIM <= (1 => (19 => '0', 18 => '0', 17 => '1', 16 => '1', OTHERS => '0'),
281 -- OTHERS => (OTHERS => '0'));
282 -- ****
283
284 WAIT FOR TP;
285 WAIT FOR DELTA_DELAY;
286
287 -- ENTRANO LE PARTI IMMAGINARIE DEI DATI -- > -- STATO CONT_2
288 INPUT_A_SIM <= (OTHERS => (OTHERS => '0'));
289 INPUT_B_SIM <= (OTHERS => (OTHERS => '0'));
290
291 WAIT FOR TP;
292 WAIT FOR DELTA_DELAY;
293
294 -- LIBERO IL BUS -- STATO CONT_3
295
296 INPUT_A_SIM <= (OTHERS => (OTHERS => 'Z'));
297 INPUT_B_SIM <= (OTHERS => (OTHERS => 'Z'));
298 WAIT FOR 3*TP;
299 WAIT FOR DELTA_DELAY;
300
301 -- STATO CONT_6
302 START_SIM <= '1'; -- PER POTER CONTINUARE AD ANDARE IN MODALITA' CONTINUA
303
304 WAIT FOR TP;
305 WAIT FOR DELTA_DELAY;
306 START_SIM <= '0'; -- STATO CONT_1
307
308 -- ENTRANO LE PARTI REALI DEI DATI
309 -- ****
310 -- 1) CASO X = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1] /2
311 -- INPUT_A_SIM <= (OTHERS => (19 => '1', 18 => '1', OTHERS => '0'));
312 -- INPUT_B_SIM <= (OTHERS => (19 => '1', 18 => '1', OTHERS => '0'));
313 -- ****
314
315 -- ****

```

```

316 -- 2) CASO X = [-1 0 +1 0 -1 0 +1 0 -1 0 +1 0 -1 0 +1 0]/2
317 -- INPUT_A_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
318 --      5 => (19 => '1', 18 => '1', OTHERS => '0'),
319 --      3 => (19 => '0', 18 => '1', OTHERS => '0'),
320 --      7 => (19 => '0', 18 => '1', OTHERS => '0'),
321 --      OTHERS => (OTHERS => '0'));
322 -- INPUT_B_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
323 --      5 => (19 => '1', 18 => '1', OTHERS => '0'),
324 --      3 => (19 => '0', 18 => '1', OTHERS => '0'),
325 --      7 => (19 => '0', 18 => '1', OTHERS => '0'),
326 --      OTHERS => (OTHERS => '0'));
327 -- ****
328
329 -- ****
330 -- 3) CASO X = [+1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]/2
331 -- INPUT_A_SIM <= (1 => (19 => '0', 18 => '1', OTHERS => '0'),
332 --      OTHERS => (OTHERS => '0'));
333 -- INPUT_B_SIM <= (OTHERS => (OTHERS => '0'));
334 -- ****
335
336 -- ****
337 -- 4) CASO X = [-1 -1 +1 +1 -1 -1 +1 +1 -1 -1 +1 +1 -1 -1 +1 +1]/2
338 -- INPUT_A_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
339 --      2 => (19 => '1', 18 => '1', OTHERS => '0'),
340 --      5 => (19 => '1', 18 => '1', OTHERS => '0'),
341 --      6 => (19 => '1', 18 => '1', OTHERS => '0'),
342 --      OTHERS => (19 => '0', 18 => '1', OTHERS => '0'));
343 -- INPUT_B_SIM <= (1 => (19 => '1', 18 => '1', OTHERS => '0'),
344 --      2 => (19 => '1', 18 => '1', OTHERS => '0'),
345 --      5 => (19 => '1', 18 => '1', OTHERS => '0'),
346 --      6 => (19 => '1', 18 => '1', OTHERS => '0'),
347 --      OTHERS => (19 => '0', 18 => '1', OTHERS => '0'));
348 -- ****
349
350 -- ****
351 -- 5) CASO X = [+0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 +0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5]/2
352 -- INPUT_A_SIM <= (OTHERS => (19 => '0', 18 => '0', 17 => '1', OTHERS => '0'));
353 -- INPUT_B_SIM <= (1 => (19 => '0', 18 => '0', 17 => '1', OTHERS => '0'),
354 --      OTHERS => (19 => '1', 18 => '1', 17 => '1', OTHERS => '0'));
355
356 -- STESSO CASO MA CON DINAMICA COMPLETA
357 -- INPUT_A_SIM <= (OTHERS => (19 => '0', 18 => '1', OTHERS => '0'));
358 -- INPUT_B_SIM <= (1 => (19 => '0', 18 => '1', OTHERS => '0'),
359 --      OTHERS => (19 => '1', 18 => '1', OTHERS => '0'));
360 -- ****
361
362 -- ****
363 -- 6) CASO X = [0 0 0 0 0 0 0 +0.75 0 0 0 0 0 0 0]/2
364 INPUT_A_SIM <= (OTHERS => (OTHERS => '0'));
365 INPUT_B_SIM <= (1 => (19 => '0', 18 => '0', 17 => '1', 16 => '1', OTHERS => '0'),
366 --      OTHERS => (OTHERS => '0'));
367 -- ****
368

```

```

369 WAIT FOR TP;
370 WAIT FOR DELTA_DELAY;
371
372 -- ENTRANO LE PARTI IMMAGINARIE DEI DATI -- > -- STATO CONT_2
373 INPUT_A_SIM <= (OTHERS => (OTHERS => '0'));
374 INPUT_B_SIM <= (OTHERS => (OTHERS => '0'));
375
376 WAIT FOR TP;
377 WAIT FOR DELTA_DELAY;
378
379 -- LIBERO IL BUS -- STATO CONT_3
380
381 INPUT_A_SIM <= (OTHERS => (OTHERS => 'Z'));
382 INPUT_B_SIM <= (OTHERS => (OTHERS => 'Z'));
383 WAIT FOR 3*TP;
384 WAIT FOR DELTA_DELAY;
385
386 -- STATO CONT_6
387 --START_SIM <= '1'; -- PER POTER CONTINUARE AD ANDARE IN MODALITA' CONTINUA, POI BISOGNA DARE
388     NUOVAMENTE GLI INPUT
389 -- PER USCIRE DA MODALITA' CONTINUA BASTA COMMENTARLO
390
391 -- *****
392 --      FINE MODALITA' CONTINUA
393 -- *****
394
395 WAIT;
396 END PROCESS Simulation;
397
398 DUT : FFT_16x16 PORT map
399   (START => START_SIM,
400    CLOCK => CLOCK_SIM,
401    RESET => RESET_SIM,
402    --OTHERS
403    -- BUS A
404    BUS_IN_A_1 => INPUT_A_SIM(1),
405    BUS_IN_A_2 => INPUT_A_SIM(2),
406    BUS_IN_A_3 => INPUT_A_SIM(3),
407    BUS_IN_A_4 => INPUT_A_SIM(4),
408    BUS_IN_A_5 => INPUT_A_SIM(5),
409    BUS_IN_A_6 => INPUT_A_SIM(6),
410    BUS_IN_A_7 => INPUT_A_SIM(7),
411    BUS_IN_A_8 => INPUT_A_SIM(8),
412    -- BUS B
413    BUS_IN_B_1 => INPUT_B_SIM(1),
414    BUS_IN_B_2 => INPUT_B_SIM(2),
415    BUS_IN_B_3 => INPUT_B_SIM(3),
416    BUS_IN_B_4 => INPUT_B_SIM(4),
417    BUS_IN_B_5 => INPUT_B_SIM(5),
418    BUS_IN_B_6 => INPUT_B_SIM(6),
419    BUS_IN_B_7 => INPUT_B_SIM(7),
420    BUS_IN_B_8 => INPUT_B_SIM(8),
421    -- BUS W
422    BUS_IN_WR_1 => INPUT_WR_SIM(1),

```

```

422    BUS_IN_WR_2 => INPUT_WR_SIM(2),
423    BUS_IN_WR_3 => INPUT_WR_SIM(3),
424    BUS_IN_WR_4 => INPUT_WR_SIM(4),
425    BUS_IN_WR_5 => INPUT_WR_SIM(5),
426    BUS_IN_WR_6 => INPUT_WR_SIM(6),
427    BUS_IN_WR_7 => INPUT_WR_SIM(7),
428    BUS_IN_WR_8 => INPUT_WR_SIM(8),
429    -- BUS WI
430    BUS_IN_WI_1 => INPUT_WI_SIM(1),
431    BUS_IN_WI_2 => INPUT_WI_SIM(2),
432    BUS_IN_WI_3 => INPUT_WI_SIM(3),
433    BUS_IN_WI_4 => INPUT_WI_SIM(4),
434    BUS_IN_WI_5 => INPUT_WI_SIM(5),
435    BUS_IN_WI_6 => INPUT_WI_SIM(6),
436    BUS_IN_WI_7 => INPUT_WI_SIM(7),
437    BUS_IN_WI_8 => INPUT_WI_SIM(8),
438    -- BUS A'
439    BUS_OUT_A_1 => OUTPUT_A_SIM(1),
440    BUS_OUT_A_2 => OUTPUT_A_SIM(2),
441    BUS_OUT_A_3 => OUTPUT_A_SIM(3),
442    BUS_OUT_A_4 => OUTPUT_A_SIM(4),
443    BUS_OUT_A_5 => OUTPUT_A_SIM(5),
444    BUS_OUT_A_6 => OUTPUT_A_SIM(6),
445    BUS_OUT_A_7 => OUTPUT_A_SIM(7),
446    BUS_OUT_A_8 => OUTPUT_A_SIM(8),
447    -- BUS B'
448    BUS_OUT_B_1 => OUTPUT_B_SIM(1),
449    BUS_OUT_B_2 => OUTPUT_B_SIM(2),
450    BUS_OUT_B_3 => OUTPUT_B_SIM(3),
451    BUS_OUT_B_4 => OUTPUT_B_SIM(4),
452    BUS_OUT_B_5 => OUTPUT_B_SIM(5),
453    BUS_OUT_B_6 => OUTPUT_B_SIM(6),
454    BUS_OUT_B_7 => OUTPUT_B_SIM(7),
455    BUS_OUT_B_8 => OUTPUT_B_SIM(8),
456    -- GLOBAL DONE
457    DONE => DONE_SIM;
458
459 END ARCHITECTURE;

```

12.2 FFT_16x16.vhd

```

1   -- QUESTO CODICE E' STATO PRODOTTO DA:
2   -- Fichera Antonio, matricola: 337213
3   -- Vannelli Elisabetta, matricola: 346477
4   LIBRARY IEEE;
5   USE IEEE.STD_LOGIC_1164.ALL;
6   USE IEEE.NUMERIC_STD.ALL;
7
8   ENTITY FFT_16x16 IS
9     PORT(START : IN STD_LOGIC;
10        CLOCK : IN STD_LOGIC;
11        RESET : IN STD_LOGIC;
12        -- BUS A

```

```

13   BUS_IN_A_1 : IN SIGNED(19 DOWNTO 0);
14   BUS_IN_A_2 : IN SIGNED(19 DOWNTO 0);
15   BUS_IN_A_3 : IN SIGNED(19 DOWNTO 0);
16   BUS_IN_A_4 : IN SIGNED(19 DOWNTO 0);
17   BUS_IN_A_5 : IN SIGNED(19 DOWNTO 0);
18   BUS_IN_A_6 : IN SIGNED(19 DOWNTO 0);
19   BUS_IN_A_7 : IN SIGNED(19 DOWNTO 0);
20   BUS_IN_A_8 : IN SIGNED(19 DOWNTO 0);
21   -- BUS B
22   BUS_IN_B_1 : IN SIGNED(19 DOWNTO 0);
23   BUS_IN_B_2 : IN SIGNED(19 DOWNTO 0);
24   BUS_IN_B_3 : IN SIGNED(19 DOWNTO 0);
25   BUS_IN_B_4 : IN SIGNED(19 DOWNTO 0);
26   BUS_IN_B_5 : IN SIGNED(19 DOWNTO 0);
27   BUS_IN_B_6 : IN SIGNED(19 DOWNTO 0);
28   BUS_IN_B_7 : IN SIGNED(19 DOWNTO 0);
29   BUS_IN_B_8 : IN SIGNED(19 DOWNTO 0);
30   -- BUS WR
31   BUS_IN_WR_1 : IN SIGNED(19 DOWNTO 0);
32   BUS_IN_WR_2 : IN SIGNED(19 DOWNTO 0);
33   BUS_IN_WR_3 : IN SIGNED(19 DOWNTO 0);
34   BUS_IN_WR_4 : IN SIGNED(19 DOWNTO 0);
35   BUS_IN_WR_5 : IN SIGNED(19 DOWNTO 0);
36   BUS_IN_WR_6 : IN SIGNED(19 DOWNTO 0);
37   BUS_IN_WR_7 : IN SIGNED(19 DOWNTO 0);
38   BUS_IN_WR_8 : IN SIGNED(19 DOWNTO 0);
39   -- BUS WI
40   BUS_IN_WI_1 : IN SIGNED(19 DOWNTO 0);
41   BUS_IN_WI_2 : IN SIGNED(19 DOWNTO 0);
42   BUS_IN_WI_3 : IN SIGNED(19 DOWNTO 0);
43   BUS_IN_WI_4 : IN SIGNED(19 DOWNTO 0);
44   BUS_IN_WI_5 : IN SIGNED(19 DOWNTO 0);
45   BUS_IN_WI_6 : IN SIGNED(19 DOWNTO 0);
46   BUS_IN_WI_7 : IN SIGNED(19 DOWNTO 0);
47   BUS_IN_WI_8 : IN SIGNED(19 DOWNTO 0);
48   -- BUS A'
49   BUS_OUT_A_1 : OUT SIGNED(19 DOWNTO 0);
50   BUS_OUT_A_2 : OUT SIGNED(19 DOWNTO 0);
51   BUS_OUT_A_3 : OUT SIGNED(19 DOWNTO 0);
52   BUS_OUT_A_4 : OUT SIGNED(19 DOWNTO 0);
53   BUS_OUT_A_5 : OUT SIGNED(19 DOWNTO 0);
54   BUS_OUT_A_6 : OUT SIGNED(19 DOWNTO 0);
55   BUS_OUT_A_7 : OUT SIGNED(19 DOWNTO 0);
56   BUS_OUT_A_8 : OUT SIGNED(19 DOWNTO 0);
57   -- BUS B'
58   BUS_OUT_B_1 : OUT SIGNED(19 DOWNTO 0);
59   BUS_OUT_B_2 : OUT SIGNED(19 DOWNTO 0);
60   BUS_OUT_B_3 : OUT SIGNED(19 DOWNTO 0);
61   BUS_OUT_B_4 : OUT SIGNED(19 DOWNTO 0);
62   BUS_OUT_B_5 : OUT SIGNED(19 DOWNTO 0);
63   BUS_OUT_B_6 : OUT SIGNED(19 DOWNTO 0);
64   BUS_OUT_B_7 : OUT SIGNED(19 DOWNTO 0);
65   BUS_OUT_B_8 : OUT SIGNED(19 DOWNTO 0);
66   -- GLOBAL DONE

```

```

67      DONE : OUT STD_LOGIC);
68  END ENTITY;
69
70 ARCHITECTURE Behavioural OF FFT_16x16 IS
71
72 COMPONENT Butterfly IS
73   PORT(START : IN STD_LOGIC;
74        CLOCK : IN STD_LOGIC;
75        RESET: IN STD_LOGIC;
76        INPUT_A : IN SIGNED(19 DOWNTO 0); -- A
77        INPUT_B : IN SIGNED(19 DOWNTO 0); -- B
78        INPUT_WR : IN SIGNED(19 DOWNTO 0); -- WR
79        INPUT_WI : IN SIGNED(19 DOWNTO 0); -- WI
80        OUTPUT_A : OUT SIGNED(19 DOWNTO 0); -- A'
81        OUTPUT_B : OUT SIGNED(19 DOWNTO 0); -- B'
82        DONE : OUT STD_LOGIC);
83  END COMPONENT;
84
85 TYPE BF_TYPE IS ARRAY(1 TO 24) OF SIGNED(19 DOWNTO 0); -- anzich 1 TO 32
86 SIGNAL BF_OUT_A, BF_OUT_B : BF_TYPE;
87 TYPE DONE_TYPE IS ARRAY(1 TO 32) OF STD_LOGIC;
88 SIGNAL DONE_BF : DONE_TYPE;
89
90 BEGIN
91
92 -- GLOBAL DONE : E' L'AND DI TUTTE LE BF DELL'ULTIMO LIVELLO. NON SAREBBE NECESSARIO, MA LO
93 -- FACCIO PER ESSERE SCRUPOLOSO
94 DONE <= DONE_BF(25) AND DONE_BF(26) AND DONE_BF(27) AND DONE_BF(28) AND DONE_BF(29) AND
95          DONE_BF(30) AND DONE_BF(31) AND DONE_BF(32);
96
97 --*****FIRST LEVEL BUTTERFLIES*****
98
99 BF1 : Butterfly
100  PORT MAP(START => START,
101            CLOCK => CLOCK,
102            RESET => RESET, -- IL RESET DELLE SINGOLE BFs COINCIDE CON QUELLO DELL'INTERO
103              SISTEMA DI FFT
104            INPUT_A => BUS_IN_A_1,
105            INPUT_B => BUS_IN_B_1,
106            INPUT_WR => BUS_IN_WR_1,
107            INPUT_WI => BUS_IN_WI_1,
108            OUTPUT_A => BF_OUT_A(1),
109            OUTPUT_B => BF_OUT_B(1),
110            DONE => DONE_BF(1));
111
112 BF2 : Butterfly
113  PORT MAP(START => START,
114            CLOCK => CLOCK,
115            RESET => RESET,
116            INPUT_A => BUS_IN_A_2,
117            INPUT_B => BUS_IN_B_2,
118            INPUT_WR => BUS_IN_WR_1,
119            INPUT_WI => BUS_IN_WI_2,
120            OUTPUT_A => BF_OUT_A(2),
121            OUTPUT_B => BF_OUT_B(2),
122            DONE => DONE_BF(2));
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
846
847
848
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
867
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
937
938
939
939
940
941
942
943
944
945
946
946
947
948
948
949
949
950
951
952
953
954
955
956
957
957
958
959
959
960
961
962
963
964
965
966
966
967
968
968
969
969
970
971
972
973
974
975
976
976
977
978
978
979
979
980
981
982
983
984
985
986
986
987
988
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1066
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1076
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1126
1127
1128
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1136
1137
1138
1138
1139
1139
1140
1141
1142
1143
1144
1145
1145
1146
1147
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1156
1157
1158
1158
1159
1159
1160
1161
1162
1163
1164
1165
1165
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1173
1174
1175
1175
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1204
1205
1206
1206
1207
1207
1208
1209
1209
1210
1210
1211
1212
1213
1214
1215
1215
1216
1217
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1225
1226
1227
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1255
1256
1257
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1265
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1294
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1304
1305
1306
1306
1307
1307
1308
1309
1309
1310
1310
1311
1312
1313
1314
1315
1315
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1325
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1335
1335
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1345
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1355
1356
1357
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1365
1365
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1375
1375
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1385
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1394
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1404
1405
1406
1406
1407
1407
1408
1409
1409
1410
1410
1411
1412
1413
1414
1415
1415
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1425
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1435
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1445
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1455
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1465
1465
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1475
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1485
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1494
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1504
1505
1506
1506
1507
1507
1508
1509
1509
1510
1510
1511
1512
1513
1514
1515
1515
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1525
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1535
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1545
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1555
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1565
1565
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1575
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1585
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1605
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1615
1615
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1625
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1635
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1645
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1655
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1665
1665
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1675
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1685
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1705
1706
1707
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1715
1715
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1725
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1735
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1745
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1755
1756
1757
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1765
1765
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1775
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1785
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1805
1806
1807
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1815
1815
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1825
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1835
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1845
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1855
1856
1857
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1865
1865
1866
1867
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1875
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1885
1886
1887
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1905
1906
1907
1907
1908
1908
1909
1909
1910
1911
1912
1913
1914
1915
1915
1916
1917
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1925
1926
1927
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1935
1936
1937
1937
1938
1938
1939
1939
1940
1941
1942
```

```
118     INPUT_WI => BUS_IN_WI_1,
119     OUTPUT_A => BF_OUT_A(2),
120     OUTPUT_B => BF_OUT_B(2),
121     DONE => DONE_BF(2));
122
123 BF3 : Butterfly
124     PORT MAP(START => START,
125                 CLOCK => CLOCK,
126                 RESET => RESET,
127                 INPUT_A => BUS_IN_A_3,
128                 INPUT_B => BUS_IN_B_3,
129                 INPUT_WR => BUS_IN_WR_1,
130                 INPUT_WI => BUS_IN_WI_1,
131                 OUTPUT_A => BF_OUT_A(3),
132                 OUTPUT_B => BF_OUT_B(3),
133                 DONE => DONE_BF(3));
134
135 BF4 : Butterfly
136     PORT MAP(START => START,
137                 CLOCK => CLOCK,
138                 RESET => RESET,
139                 INPUT_A => BUS_IN_A_4,
140                 INPUT_B => BUS_IN_B_4,
141                 INPUT_WR => BUS_IN_WR_1,
142                 INPUT_WI => BUS_IN_WI_1,
143                 OUTPUT_A => BF_OUT_A(4),
144                 OUTPUT_B => BF_OUT_B(4),
145                 DONE => DONE_BF(4));
146
147 BF5 : Butterfly
148     PORT MAP(START => START,
149                 CLOCK => CLOCK,
150                 RESET => RESET,
151                 INPUT_A => BUS_IN_A_5,
152                 INPUT_B => BUS_IN_B_5,
153                 INPUT_WR => BUS_IN_WR_1,
154                 INPUT_WI => BUS_IN_WI_1,
155                 OUTPUT_A => BF_OUT_A(5),
156                 OUTPUT_B => BF_OUT_B(5),
157                 DONE => DONE_BF(5));
158
159 BF6 : Butterfly
160     PORT MAP(START => START,
161                 CLOCK => CLOCK,
162                 RESET => RESET,
163                 INPUT_A => BUS_IN_A_6,
164                 INPUT_B => BUS_IN_B_6,
165                 INPUT_WR => BUS_IN_WR_1,
166                 INPUT_WI => BUS_IN_WI_1,
167                 OUTPUT_A => BF_OUT_A(6),
168                 OUTPUT_B => BF_OUT_B(6),
169                 DONE => DONE_BF(6));
170
171 BF7 : Butterfly
```

```

172 PORT MAP(START => START,
173     CLOCK => CLOCK,
174     RESET => RESET,
175     INPUT_A => BUS_IN_A_7,
176     INPUT_B => BUS_IN_B_7,
177     INPUT_WR => BUS_IN_WR_1,
178     INPUT_WI => BUS_IN_WI_1,
179     OUTPUT_A => BF_OUT_A(7),
180     OUTPUT_B => BF_OUT_B(7),
181     DONE => DONE_BF(7));
182
183 BF8 : Butterfly
184     PORT MAP(START => START,
185         CLOCK => CLOCK,
186         RESET => RESET,
187         INPUT_A => BUS_IN_A_8,
188         INPUT_B => BUS_IN_B_8,
189         INPUT_WR => BUS_IN_WR_1,
190         INPUT_WI => BUS_IN_WI_1,
191         OUTPUT_A => BF_OUT_A(8),
192         OUTPUT_B => BF_OUT_B(8),
193         DONE => DONE_BF(8));
194
195 ---*****SECOND LEVEL BUTTERFLIES*****
196 -- SECOND LEVEL BUTTERFLIES
197 ---*****SECOND LEVEL BUTTERFLIES*****
198
199 BF9 : Butterfly
200     PORT MAP(START => DONE_BF(1), -- DONE DELLA BF 1, USO LO STESSO PER TUTTE LE BF DEL
201         SECONDO LIVELLO, ESSENDO CHE LAVORANO IN PARALLELO
202         CLOCK => CLOCK,
203         RESET => RESET,
204         INPUT_A => BF_OUT_A(1),
205         INPUT_B => BF_OUT_A(5),
206         INPUT_WR => BUS_IN_WR_1,
207         INPUT_WI => BUS_IN_WI_1,
208         OUTPUT_A => BF_OUT_A(9),
209         OUTPUT_B => BF_OUT_B(9),
210         DONE => DONE_BF(9));
211
212 BF10 : Butterfly
213     PORT MAP(START => DONE_BF(1),
214         CLOCK => CLOCK,
215         RESET => RESET,
216         INPUT_A => BF_OUT_A(2),
217         INPUT_B => BF_OUT_A(6),
218         INPUT_WR => BUS_IN_WR_1,
219         INPUT_WI => BUS_IN_WI_1,
220         OUTPUT_A => BF_OUT_A(10),
221         OUTPUT_B => BF_OUT_B(10),
222         DONE => DONE_BF(10));
223
224 BF11 : Butterfly
225     PORT MAP(START => DONE_BF(1),

```

```

225     CLOCK => CLOCK,
226     RESET => RESET,
227     INPUT_A => BF_OUT_A(3),
228     INPUT_B => BF_OUT_A(7),
229     INPUT_WR => BUS_IN_WR_1,
230     INPUT_WI => BUS_IN_WI_1,
231     OUTPUT_A => BF_OUT_A(11),
232     OUTPUT_B => BF_OUT_B(11),
233     DONE => DONE_BF(11));
234
235 BF12 : Butterfly
236     PORT MAP(START => DONE_BF(1),
237                 CLOCK => CLOCK,
238                 RESET => RESET,
239                 INPUT_A => BF_OUT_A(4),
240                 INPUT_B => BF_OUT_A(8),
241                 INPUT_WR => BUS_IN_WR_1,
242                 INPUT_WI => BUS_IN_WI_1,
243                 OUTPUT_A => BF_OUT_A(12),
244                 OUTPUT_B => BF_OUT_B(12),
245                 DONE => DONE_BF(12));
246
247 BF13 : Butterfly
248     PORT MAP(START => DONE_BF(1),
249                 CLOCK => CLOCK,
250                 RESET => RESET,
251                 INPUT_A => BF_OUT_B(1),
252                 INPUT_B => BF_OUT_B(5),
253                 INPUT_WR => BUS_IN_WR_2,
254                 INPUT_WI => BUS_IN_WI_2,
255                 OUTPUT_A => BF_OUT_A(13),
256                 OUTPUT_B => BF_OUT_B(13),
257                 DONE => DONE_BF(13));
258
259 BF14 : Butterfly
260     PORT MAP(START => DONE_BF(1),
261                 CLOCK => CLOCK,
262                 RESET => RESET,
263                 INPUT_A => BF_OUT_B(2),
264                 INPUT_B => BF_OUT_B(6),
265                 INPUT_WR => BUS_IN_WR_2,
266                 INPUT_WI => BUS_IN_WI_2,
267                 OUTPUT_A => BF_OUT_A(14),
268                 OUTPUT_B => BF_OUT_B(14),
269                 DONE => DONE_BF(14));
270
271 BF15 : Butterfly
272     PORT MAP(START => DONE_BF(1),
273                 CLOCK => CLOCK,
274                 RESET => RESET,
275                 INPUT_A => BF_OUT_B(3),
276                 INPUT_B => BF_OUT_B(7),
277                 INPUT_WR => BUS_IN_WR_2,
278                 INPUT_WI => BUS_IN_WI_2,

```

```

279     OUTPUT_A => BF_OUT_A(15),
280     OUTPUT_B => BF_OUT_B(15),
281     DONE => DONE_BF(15));
282
283 BF16 : Butterfly
284     PORT MAP(START => DONE_BF(1),
285                 CLOCK => CLOCK,
286                 RESET => RESET,
287                 INPUT_A => BF_OUT_B(4),
288                 INPUT_B => BF_OUT_B(8),
289                 INPUT_WR => BUS_IN_WR_2,
290                 INPUT_WI => BUS_IN_WI_2,
291                 OUTPUT_A => BF_OUT_A(16),
292                 OUTPUT_B => BF_OUT_B(16),
293                 DONE => DONE_BF(16));
294
295 --*****
296 --      THIRD LEVEL BUTTERFLIES
297 --*****
298
299 BF17 : Butterfly
300     PORT MAP(START => DONE_BF(9),
301                 CLOCK => CLOCK,
302                 RESET => RESET,
303                 INPUT_A => BF_OUT_A(9),
304                 INPUT_B => BF_OUT_A(11),
305                 INPUT_WR => BUS_IN_WR_1,
306                 INPUT_WI => BUS_IN_WI_1,
307                 OUTPUT_A => BF_OUT_A(17),
308                 OUTPUT_B => BF_OUT_B(17),
309                 DONE => DONE_BF(17));
310
311 BF18 : Butterfly
312     PORT MAP(START => DONE_BF(9),
313                 CLOCK => CLOCK,
314                 RESET => RESET,
315                 INPUT_A => BF_OUT_A(10),
316                 INPUT_B => BF_OUT_A(12),
317                 INPUT_WR => BUS_IN_WR_1,
318                 INPUT_WI => BUS_IN_WI_1,
319                 OUTPUT_A => BF_OUT_A(18),
320                 OUTPUT_B => BF_OUT_B(18),
321                 DONE => DONE_BF(18));
322
323 BF19 : Butterfly
324     PORT MAP(START => DONE_BF(9),
325                 CLOCK => CLOCK,
326                 RESET => RESET,
327                 INPUT_A => BF_OUT_B(9),
328                 INPUT_B => BF_OUT_B(11),
329                 INPUT_WR => BUS_IN_WR_2,
330                 INPUT_WI => BUS_IN_WI_2,
331                 OUTPUT_A => BF_OUT_A(19),
332                 OUTPUT_B => BF_OUT_B(19),

```

```

333     DONE => DONE_BF(19));
334
335 BF20 : Butterfly
336     PORT MAP(START => DONE_BF(9),
337                 CLOCK => CLOCK,
338                 RESET => RESET,
339                 INPUT_A => BF_OUT_B(10),
340                 INPUT_B => BF_OUT_B(12),
341                 INPUT_WR => BUS_IN_WR_2,
342                 INPUT_WI => BUS_IN_WI_2,
343                 OUTPUT_A => BF_OUT_A(20),
344                 OUTPUT_B => BF_OUT_B(20),
345                 DONE => DONE_BF(20));
346
347 BF21 : Butterfly
348     PORT MAP(START => DONE_BF(9),
349                 CLOCK => CLOCK,
350                 RESET => RESET,
351                 INPUT_A => BF_OUT_A(13),
352                 INPUT_B => BF_OUT_A(15),
353                 INPUT_WR => BUS_IN_WR_3,
354                 INPUT_WI => BUS_IN_WI_3,
355                 OUTPUT_A => BF_OUT_A(21),
356                 OUTPUT_B => BF_OUT_B(21),
357                 DONE => DONE_BF(21));
358
359 BF22 : Butterfly
360     PORT MAP(START => DONE_BF(9),
361                 CLOCK => CLOCK,
362                 RESET => RESET,
363                 INPUT_A => BF_OUT_A(14),
364                 INPUT_B => BF_OUT_A(16),
365                 INPUT_WR => BUS_IN_WR_3,
366                 INPUT_WI => BUS_IN_WI_3,
367                 OUTPUT_A => BF_OUT_A(22),
368                 OUTPUT_B => BF_OUT_B(22),
369                 DONE => DONE_BF(22));
370
371 BF23 : Butterfly
372     PORT MAP(START => DONE_BF(9),
373                 CLOCK => CLOCK,
374                 RESET => RESET,
375                 INPUT_A => BF_OUT_B(13),
376                 INPUT_B => BF_OUT_B(15),
377                 INPUT_WR => BUS_IN_WR_4,
378                 INPUT_WI => BUS_IN_WI_4,
379                 OUTPUT_A => BF_OUT_A(23),
380                 OUTPUT_B => BF_OUT_B(23),
381                 DONE => DONE_BF(23));
382
383 BF24 : Butterfly
384     PORT MAP(START => DONE_BF(9),
385                 CLOCK => CLOCK,
386                 RESET => RESET,

```

```

387     INPUT_A => BF_OUT_B(14),
388     INPUT_B => BF_OUT_B(16),
389     INPUT_WR => BUS_IN_WR_4,
390     INPUT_WI => BUS_IN_WI_4,
391     OUTPUT_A => BF_OUT_A(24),
392     OUTPUT_B => BF_OUT_B(24),
393     DONE => DONE_BF(24));
394
395 ---*****FORTH LEVEL BUTTERFLIES*****
396 -- FORTH LEVEL BUTTERFLIES
397 ---*****FORTH LEVEL BUTTERFLIES*****
398
399 BF25 : Butterfly
400   PORT MAP(START => DONE_BF(17),
401             CLOCK => CLOCK,
402             RESET => RESET,
403             INPUT_A => BF_OUT_A(17),
404             INPUT_B => BF_OUT_A(18),
405             INPUT_WR => BUS_IN_WR_1,
406             INPUT_WI => BUS_IN_WI_1,
407             OUTPUT_A => BUS_OUT_A_1,
408             OUTPUT_B => BUS_OUT_B_1,
409             DONE => DONE_BF(25));
410
411 BF26 : Butterfly
412   PORT MAP(START => DONE_BF(17),
413             CLOCK => CLOCK,
414             RESET => RESET,
415             INPUT_A => BF_OUT_B(17),
416             INPUT_B => BF_OUT_B(18),
417             INPUT_WR => BUS_IN_WR_2,
418             INPUT_WI => BUS_IN_WI_2,
419             OUTPUT_A => BUS_OUT_A_5,
420             OUTPUT_B => BUS_OUT_B_5,
421             DONE => DONE_BF(26));
422
423 BF27 : Butterfly
424   PORT MAP(START => DONE_BF(17),
425             CLOCK => CLOCK,
426             RESET => RESET,
427             INPUT_A => BF_OUT_A(19),
428             INPUT_B => BF_OUT_A(20),
429             INPUT_WR => BUS_IN_WR_3,
430             INPUT_WI => BUS_IN_WI_3,
431             OUTPUT_A => BUS_OUT_A_3,
432             OUTPUT_B => BUS_OUT_B_3,
433             DONE => DONE_BF(27));
434
435 BF28 : Butterfly
436   PORT MAP(START => DONE_BF(17),
437             CLOCK => CLOCK,
438             RESET => RESET,
439             INPUT_A => BF_OUT_B(19),
440             INPUT_B => BF_OUT_B(20),

```

```

441     INPUT_WR => BUS_IN_WR_4,
442     INPUT_WI => BUS_IN_WI_4,
443     OUTPUT_A => BUS_OUT_A_7,
444     OUTPUT_B => BUS_OUT_B_7,
445     DONE => DONE_BF(28));
446
447 BF29 : Butterfly
448     PORT MAP(START => DONE_BF(17),
449                 CLOCK => CLOCK,
450                 RESET => RESET,
451                 INPUT_A => BF_OUT_A(21),
452                 INPUT_B => BF_OUT_A(22),
453                 INPUT_WR => BUS_IN_WR_5,
454                 INPUT_WI => BUS_IN_WI_5,
455                 OUTPUT_A => BUS_OUT_A_2,
456                 OUTPUT_B => BUS_OUT_B_2,
457                 DONE => DONE_BF(29));
458
459 BF30 : Butterfly
460     PORT MAP(START => DONE_BF(17),
461                 CLOCK => CLOCK,
462                 RESET => RESET,
463                 INPUT_A => BF_OUT_B(21),
464                 INPUT_B => BF_OUT_B(22),
465                 INPUT_WR => BUS_IN_WR_6,
466                 INPUT_WI => BUS_IN_WI_6,
467                 OUTPUT_A => BUS_OUT_A_6,
468                 OUTPUT_B => BUS_OUT_B_6,
469                 DONE => DONE_BF(30));
470
471 BF31 : Butterfly
472     PORT MAP(START => DONE_BF(17),
473                 CLOCK => CLOCK,
474                 RESET => RESET,
475                 INPUT_A => BF_OUT_A(23),
476                 INPUT_B => BF_OUT_A(24),
477                 INPUT_WR => BUS_IN_WR_7,
478                 INPUT_WI => BUS_IN_WI_7,
479                 OUTPUT_A => BUS_OUT_A_4,
480                 OUTPUT_B => BUS_OUT_B_4,
481                 DONE => DONE_BF(31));
482
483 BF32 : Butterfly
484     PORT MAP(START => DONE_BF(17),
485                 CLOCK => CLOCK,
486                 RESET => RESET,
487                 INPUT_A => BF_OUT_B(23),
488                 INPUT_B => BF_OUT_B(24),
489                 INPUT_WR => BUS_IN_WR_8,
490                 INPUT_WI => BUS_IN_WI_8,
491                 OUTPUT_A => BUS_OUT_A_8,
492                 OUTPUT_B => BUS_OUT_B_8,
493                 DONE => DONE_BF(32));
494

```

```

495 -----
496 --      END OF ALL BUTTERFLIES
497 -----
498
499 END ARCHITECTURE;

```

12.3 Butterfly.vhd

```

1  -- QUESTO CODICE E' STATO PRODOTTO DA:
2  -- Fichera Antonio, matricola: 337213
3  -- Vannelli Elisabetta, matricola: 346477
4  LIBRARY IEEE;
5  USE IEEE.STD_LOGIC_1164.ALL;
6  USE IEEE.NUMERIC_STD.ALL;
7
8  ENTITY Butterfly IS
9    PORT(START : IN STD_LOGIC;
10        CLOCK : IN STD_LOGIC;
11        RESET: IN STD_LOGIC;
12        INPUT_A : IN SIGNED(19 DOWNTO 0); -- A
13        INPUT_B : IN SIGNED(19 DOWNTO 0); -- B
14        INPUT_WR : IN SIGNED(19 DOWNTO 0); -- WR
15        INPUT_WI : IN SIGNED(19 DOWNTO 0); -- WI
16        OUTPUT_A : OUT SIGNED(19 DOWNTO 0); -- A'
17        OUTPUT_B : OUT SIGNED(19 DOWNTO 0); -- B'
18        DONE : OUT STD_LOGIC);
19  END ENTITY;
20
21  ARCHITECTURE Structural OF Butterfly IS
22
23  -----
24  --      COMPONENTI DATAPATH/EXECUTION UNIT
25  -----
26
27  COMPONENT Register_Nbit IS
28    GENERIC(N : POSITIVE);
29    PORT(CLOCK, ENABLE : IN STD_LOGIC;
30          R : IN SIGNED(N-1 DOWNTO 0);
31          Q : OUT SIGNED(N-1 DOWNTO 0));
32  END COMPONENT;
33
34  COMPONENT MUX_2TO1_Nbit IS
35    GENERIC(N : POSITIVE);
36    PORT(INPUT_0, INPUT_1 : IN SIGNED(N-1 DOWNTO 0);
37          SEL : IN STD_LOGIC;
38          MUX_OUT : OUT SIGNED(N-1 DOWNTO 0));
39  END COMPONENT;
40
41  COMPONENT Multiplier_Shifter IS
42    PORT(INPUT_SX, INPUT_DX : IN SIGNED(19 DOWNTO 0);
43          CLOCK, MPY_SHIFTN : IN STD_LOGIC;
44          MULT_OUT : OUT SIGNED(39 DOWNTO 0));
45  END COMPONENT;

```

```

46
47 COMPONENT Adder_Subtractor IS
48   PORT(INPUT_SX, INPUT_DX : IN SIGNED(41 DOWNTO 0);
49     SUB_ADDN_SX, SUB_ADDN_DX, CLOCK : IN STD_LOGIC;
50     SUM_OUT : OUT SIGNED(41 DOWNTO 0));
51 END COMPONENT;
52
53 COMPONENT Rounder IS
54   PORT(INPUT : IN SIGNED(41 DOWNTO 0);
55     CLOCK : IN STD_LOGIC;
56     OUTPUT : OUT SIGNED(19 DOWNTO 0));
57 END COMPONENT;
58
59 -- ****
60 -- COMPONENTI CONTROL UNIT
61 -- ****
62
63 COMPONENT Register_Nbit_SLV IS
64   GENERIC(N : POSITIVE);
65   PORT(CLOCK, ENABLE, RESET : IN STD_LOGIC;
66     R : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
67     Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0));
68 END COMPONENT;
69
70 COMPONENT MUX_2T01_Nbit_SLV IS
71   GENERIC(N : POSITIVE);
72   PORT(INPUT_0, INPUT_1 : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
73     SEL : IN STD_LOGIC;
74     MUX_OUT : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0));
75 END COMPONENT;
76
77 COMPONENT MUX_2T01 IS
78   PORT(INPUT_0, INPUT_1 : IN STD_LOGIC;
79     SEL : IN STD_LOGIC;
80     MUX_OUT : OUT STD_LOGIC);
81 END COMPONENT;
82
83 COMPONENT Decoder_4to16 IS
84   PORT(DEC_IN : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
85     DEC_OUT : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
86 END COMPONENT;
87
88 COMPONENT EVEN_ROM IS
89   PORT(ROW_SELECT : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
90     DATA_OUT : OUT STD_LOGIC_VECTOR(25 DOWNTO 0));
91 END COMPONENT;
92
93 COMPONENT ODD_ROM IS
94   PORT(ROW_SELECT : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
95     DATA_OUT : OUT STD_LOGIC_VECTOR(25 DOWNTO 0));
96 END COMPONENT;
97
98 COMPONENT LateStatus_PLA IS
99   PORT(STATUS, CC, LSB_IN : IN STD_LOGIC;

```

```

100      CC_Validation, LSB_OUT : OUT STD_LOGIC);
101  END COMPONENT;
102
103  -- ****
104  -- END OF ALL COMPONENTS
105  -- ****
106
107  -- ****
108  -- SIGNALS FOR DP PORT MAPPING
109  -- ****
110
111  -- REGISTRI D'INGRESSO
112  SIGNAL REG_AR_OUT, REG_AI_OUT, REG_BR_OUT, REG_BI_OUT, REG_WR_OUT, REG_WI_OUT : SIGNED(19
113    DOWNTO 0);
114  SIGNAL MUX_A_OUT, MUX_B_OUT, MUX_W_OUT, MUX_AB_OUT : SIGNED(19 DOWNTO 0);
115  -- OPERATORI ARITMETICI
116  SIGNAL MULT_OUT, REG_TMP_MPY_OUT : SIGNED(39 DOWNTO 0);
117  SIGNAL EXTENDED_SUM_A, EXTENDED_SUM_DX, MUX_SUM_OUT, SUM_OUT, REG_TMP_SUM_OUT : SIGNED(41
118    DOWNTO 0);
119  -- ROUNDER
120  SIGNAL ROUNDER_OUT : SIGNED(19 DOWNTO 0);
121  -- REGISTRI D'USCITA
122  SIGNAL REG_AR_PRIMO_OUT, REG_AI_PRIMO_OUT, REG_BR_PRIMO_OUT, REG_BI_PRIMO_OUT : SIGNED(19
123    DOWNTO 0);
124
125  -- ****
126  -- SIGNALS FOR CU PORT MAPPING
127  -- ****
128  SIGNAL NEXT_STATUS_BITS, uAR_OUT : STD_LOGIC_VECTOR(4 DOWNTO 0);
129  SIGNAL DEC_OUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
130  SIGNAL EVEN_ROM_OUT, ODD_ROM_OUT, MUX_ROM_OUT, uIR_OUT : STD_LOGIC_VECTOR(25 DOWNTO 0);
131  SIGNAL MUX_JUMP_OUT, CC_Validation_OUT, LSB_PLA_OUT, CLOCK_NEG : STD_LOGIC;
132
133  BEGIN
134
135
136  EXTENDED_SUM_A(41 DOWNTO 39) <= (OTHERS => MUX_A_OUT(19)); -- replica TRE volte l'MSB
137  EXTENDED_SUM_A(38 DOWNTO 19) <= MUX_A_OUT;
138  EXTENDED_SUM_A(18 DOWNTO 0) <= (OTHERS => '0');
139
140  EXTENDED_SUM_DX(41 DOWNTO 40) <= (OTHERS => REG_TMP_MPY_OUT(39));
141  EXTENDED_SUM_DX(39 DOWNTO 0) <= REG_TMP_MPY_OUT;
142
143  DONE <= uIR_OUT(25);
144
145  NEXT_STATUS_BITS <= uIR_OUT(4 DOWNTO 0);
146
147  CLOCK_NEG <= NOT CLOCK; -- SEGNALE AGGIUNTO PER NON AVERE ERRORE SU MODELSIM (DICEVA "NOT
148    GLOBALLY STATIC")
149  -- REGISTRI DI INGRESSO

```

```

150 REG_AR : Register_Nbit GENERIC MAP(N => 20)
151   PORT MAP(CLOCK => CLOCK,
152             ENABLE => uIR_OUT(23),
153             R => INPUT_A,
154             Q => REG_AR_OUT);
155
156 REG_AI : Register_Nbit GENERIC MAP(N => 20)
157   PORT MAP(CLOCK => CLOCK,
158             ENABLE => uIR_OUT(22),
159             R => INPUT_A,
160             Q => REG_AI_OUT);
161
162 REG_BR : Register_Nbit GENERIC MAP(N => 20)
163   PORT MAP(CLOCK => CLOCK,
164             ENABLE => uIR_OUT(21),
165             R => INPUT_B,
166             Q => REG_BR_OUT);
167
168 REG_BI : Register_Nbit GENERIC MAP(N => 20)
169   PORT MAP(CLOCK => CLOCK,
170             ENABLE => uIR_OUT(20),
171             R => INPUT_B,
172             Q => REG_BI_OUT);
173
174 REG_WR : Register_Nbit GENERIC MAP(N => 20)
175   PORT MAP(CLOCK => CLOCK,
176             ENABLE => uIR_OUT(19),
177             R => INPUT_WR,
178             Q => REG_WR_OUT);
179
180 REG_WI : Register_Nbit GENERIC MAP(N => 20)
181   PORT MAP(CLOCK => CLOCK,
182             ENABLE => uIR_OUT(18),
183             R => INPUT_WI,
184             Q => REG_WI_OUT);
185 -----
186 MUX_A : MUX_2TO1_Nbit GENERIC MAP(N => 20)
187   PORT MAP(INPUT_0 => REG_AR_OUT,
188             INPUT_1 => REG_AI_OUT,
189             SEL => uIR_OUT(17),
190             MUX_OUT => MUX_A_OUT);
191
192 MUX_B : MUX_2TO1_Nbit GENERIC MAP(N => 20)
193   PORT MAP(INPUT_0 => REG_BR_OUT,
194             INPUT_1 => REG_BI_OUT,
195             SEL => uIR_OUT(16),
196             MUX_OUT => MUX_B_OUT);
197
198 MUX_W : MUX_2TO1_Nbit GENERIC MAP(N => 20)
199   PORT MAP(INPUT_0 => REG_WR_OUT,
200             INPUT_1 => REG_WI_OUT,
201             SEL => uIR_OUT(15),
202             MUX_OUT => MUX_W_OUT);
203

```

```

204 MUX_AB : MUX_2TO1_Nbit GENERIC MAP(N => 20)
205   PORT MAP(INPUT_0 => MUX_A_OUT,
206             INPUT_1 => MUX_B_OUT,
207             SEL => uIR_OUT(14),
208             MUX_OUT => MUX_AB_OUT);
209
210 MULTIPLIER : Multiplier_Shifter
211   PORT MAP(INPUT_SX => MUX_AB_OUT,
212             INPUT_DX => MUX_W_OUT,
213             CLOCK => CLOCK, -- A CAUSA DEL REGISTRO DI PIPE INTERNO
214             MPY_SHIFTN => uIR_OUT(13),
215             MULT_OUT => MULT_OUT);
216
217 REG_TMP_MPY : Register_Nbit GENERIC MAP(N => 40)
218   PORT MAP(CLOCK => CLOCK,
219             ENABLE => '1',
220             R => MULT_OUT,
221             Q => REG_TMP_MPY_OUT);
222
223 MUX_SUM : MUX_2TO1_Nbit GENERIC MAP(N => 42)
224   PORT MAP(INPUT_0 => EXTENDED_SUM_A,
225             INPUT_1 => REG_TMP_SUM_OUT,
226             SEL => uIR_OUT(12),
227             MUX_OUT => MUX_SUM_OUT);
228
229 ADDER : Adder_Subtractor
230   PORT MAP(INPUT_SX => MUX_SUM_OUT,
231             INPUT_DX => EXTENDED_SUM_DX,
232             SUB_ADDN_SX => uIR_OUT(11),
233             SUB_ADDN_DX => uIR_OUT(10),
234             CLOCK => CLOCK, -- A CAUSA DEL REGISTRO DI PIPE INTERNO
235             SUM_OUT => SUM_OUT);
236
237 REG_TMP_SUM : Register_Nbit GENERIC MAP(N => 42)
238   PORT MAP(CLOCK => CLOCK,
239             ENABLE => '1',
240             R => SUM_OUT,
241             Q => REG_TMP_SUM_OUT);
242
243 Rounder_HU : Rounder
244   PORT MAP(INPUT => REG_TMP_SUM_OUT,
245             CLOCK => CLOCK, -- A CAUSA DEL REGISTRO DI PIPE INTERNO
246             OUTPUT => ROUNDER_OUT);
247
248 -- REGISTRI DI USCITA
249 REG_AR_PRIMO : Register_Nbit GENERIC MAP(N => 20)
250   PORT MAP(CLOCK => CLOCK,
251             ENABLE => uIR_OUT(8),
252             R => ROUNDER_OUT,
253             Q => REG_AR_PRIMO_OUT);
254
255 REG_AI_PRIMO : Register_Nbit GENERIC MAP(N => 20)
256   PORT MAP(CLOCK => CLOCK,
257             ENABLE => uIR_OUT(7),

```

```

258     R => ROUNDER_OUT,
259     Q => REG_AI_PRIMO_OUT);
260
261 REG_BR_PRIMO : Register_Nbit GENERIC MAP(N => 20)
262   PORT MAP(CLOCK => CLOCK,
263             ENABLE => uIR_OUT(6),
264             R => ROUNDER_OUT,
265             Q => REG_BR_PRIMO_OUT);
266
267 REG_BI_PRIMO : Register_Nbit GENERIC MAP(N => 20)
268   PORT MAP(CLOCK => CLOCK,
269             ENABLE => uIR_OUT(5),
270             R => ROUNDER_OUT,
271             Q => REG_BI_PRIMO_OUT);
272
273 MUX_OUT_A : MUX_2TO1_Nbit GENERIC MAP(N => 20)
274   PORT MAP(INPUT_0 => REG_AR_PRIMO_OUT,
275             INPUT_1 => REG_AI_PRIMO_OUT,
276             SEL => uIR_OUT(9),
277             MUX_OUT => OUTPUT_A);
278
279 MUX_OUT_B : MUX_2TO1_Nbit GENERIC MAP(N => 20)
280   PORT MAP(INPUT_0 => REG_BR_PRIMO_OUT,
281             INPUT_1 => REG_BI_PRIMO_OUT,
282             SEL => uIR_OUT(9),
283             MUX_OUT => OUTPUT_B);
284
285 -- ****
286 -- CONTROL UNIT PORT MAPPING
287 -- ****
288
289 uAR : Register_Nbit_SLV GENERIC MAP(N => 5)
290   PORT MAP(CLOCK => CLOCK_NEG, -- EVOLVE SUL FRONTE DI DISCESA DEL CLOCK
291             ENABLE => '1',
292             RESET => RESET, -- SE RESETTO IL uAR DALL'ESTERNO, SICURAMENTE RIPARTO DALLO
293             STATO DI IDLE
294             R(4 DOWNTO 1) => NEXT_STATUS_BITS(4 DOWNTO 1),
295             R(0) => LSB_PLA_OUT,
296             Q => uAR_OUT);
297
298 MUX_JUMP : MUX_2TO1
299   PORT MAP(INPUT_0 => uAR_OUT(0),
300             INPUT_1 => LSB_PLA_OUT,
301             SEL => CC_Validation_OUT,
302             MUX_OUT => MUX_JUMP_OUT);
303
304 Decoder : Decoder_4to16
305   PORT MAP(DEC_IN => uAR_OUT(4 DOWNTO 1),
306             DEC_OUT => DEC_OUT);
307
308 EVEN_ROM_EU : EVEN_ROM
309   PORT MAP(ROW_SELECT => DEC_OUT,
310             DATA_OUT => EVEN_ROM_OUT);
311

```

```

311 ODD_ROM_EU : ODD_ROM
312     PORT MAP(ROW_SELECT => DEC_OUT,
313                 DATA_OUT => ODD_ROM_OUT);
314
315 MUX_ROM : MUX_2TO1_Nbit_SLV GENERIC MAP(N => 26)
316     PORT MAP(INPUT_0 => EVEN_ROM_OUT,
317                 INPUT_1 => ODD_ROM_OUT,
318                 SEL => MUX_JUMP_OUT,
319                 MUX_OUT => MUX_ROM_OUT);
320
321 uIR : Register_Nbit_SLV GENERIC MAP(N => 26)
322     PORT MAP(CLOCK => CLOCK,
323                 ENABLE => '1',
324                 RESET => RESET, -- PURE IL uIR VIENE RESETTATO DALL'ESTERNO, IN MODO TALE DA
325                 -- AVERE TUTTI I COMANDI AL VALORE DI DEFAULT
326                 R => MUX_ROM_OUT,
327                 Q => uIR_OUT);
328
329 PLA : LateStatus_PLA
330     PORT MAP(STATUS => START,
331                 CC => uIR_OUT(24),
332                 LSB_IN => NEXT_STATUS_BITS(0),
333                 CC_Validation => CC_Validation_OUT,
334                 LSB_OUT => LSB_PLA_OUT);
335
336 END ARCHITECTURE;

```

12.4 Multiplier_Shifter.vhd

```

1  -- QUESTO CODICE E' STATO PRODOTTO DA:
2  -- Fichera Antonio, matricola: 337213
3  -- Vannelli Elisabetta, matricola: 346477
4  LIBRARY IEEE;
5  USE IEEE.STD_LOGIC_1164.ALL;
6  USE IEEE.NUMERIC_STD.ALL;
7
8  ENTITY Multiplier_Shifter IS
9      PORT (INPUT_SX, INPUT_DX : IN SIGNED(19 DOWNTO 0);
10             CLOCK, MPY_SHIFTN : IN STD_LOGIC;
11             MULT_OUT : OUT SIGNED(39 DOWNTO 0));
12  END ENTITY;
13
14  ARCHITECTURE Behavioural OF Multiplier_Shifter IS
15
16  COMPONENT MUX_2TO1_Nbit IS
17      GENERIC(N : POSITIVE);
18      PORT(INPUT_0, INPUT_1 : IN SIGNED(N-1 DOWNTO 0);
19             SEL : IN STD_LOGIC;
20             MUX_OUT : OUT SIGNED(N-1 DOWNTO 0));
21  END COMPONENT;
22
23  COMPONENT Register_Nbit IS
24      GENERIC(N : POSITIVE);

```

```

25 PORT(CLOCK, ENABLE : IN STD_LOGIC;
26     R : IN SIGNED(N-1 DOWNTO 0);
27     Q : OUT SIGNED(N-1 DOWNTO 0));
28 END COMPONENT;
29
30 SIGNAL MULT_OUT_TMP : SIGNED(39 DOWNTO 0);
31 SIGNAL SHIFTED_INPUT, MUX_OUTPUT : SIGNED(39 DOWNTO 0);
32
33 BEGIN
34
35 MULT_OUT_TMP <= (INPUT_SX * INPUT_DX);
36
37 SHIFTED_INPUT(39 DOWNTO 20) <= INPUT_SX(19 DOWNTO 0);
38 SHIFTED_INPUT(19 DOWNTO 0) <= (OTHERS => '0');
39
40 MUX : MUX_2TO1_Nbit GENERIC MAP(N => 40)
41     PORT MAP (INPUT_0 => SHIFTED_INPUT,
42                 INPUT_1 => MULT_OUT_TMP,
43                 SEL => MPY_SHIFTN,
44                 MUX_OUT => MUX_OUTPUT);
45
46 REG_PIPE : Register_Nbit GENERIC MAP(N => 40)
47     PORT MAP (CLOCK => CLOCK,
48                 ENABLE => '1',
49                 R => MUX_OUTPUT,
50                 Q => MULT_OUT);
51
52 END ARCHITECTURE;

```

12.5 Adder_Subtractor.vhd

```

1 -- QUESTO CODICE E' STATO PRODOTTO DA:
2 -- Fichera Antonio, matricola: 337213
3 -- Vannelli Elisabetta, matricola: 346477
4 LIBRARY IEEE;
5 USE IEEE.STD_LOGIC_1164.ALL;
6 USE IEEE.NUMERIC_STD.ALL;
7
8 ENTITY Adder_Subtractor IS
9     PORT (INPUT_SX, INPUT_DX : IN SIGNED(41 DOWNTO 0);
10            SUB_ADDN_SX, SUB_ADDN_DX, CLOCK : IN STD_LOGIC;
11            SUM_OUT : OUT SIGNED(41 DOWNTO 0));
12 END ENTITY;
13
14 ARCHITECTURE Behavioural OF Adder_Subtractor IS
15
16 COMPONENT Register_Nbit IS
17     GENERIC(N : POSITIVE);
18     PORT(CLOCK, ENABLE : IN STD_LOGIC;
19             R : IN SIGNED(N-1 DOWNTO 0);
20             Q : OUT SIGNED(N-1 DOWNTO 0));
21 END COMPONENT;
22

```

```

23 COMPONENT MUX_2TO1_Nbit IS
24   GENERIC(N : POSITIVE);
25     PORT(INPUT_0, INPUT_1 : IN SIGNED(N-1 DOWNTO 0);
26       SEL : IN STD_LOGIC;
27       MUX_OUT : OUT SIGNED(N-1 DOWNTO 0));
28   END COMPONENT;
29
30 SIGNAL MUX_SX_OUT, MUX_DX_OUT, SUM_OUT_TMP, SX_NEG, DX_NEG : SIGNED(41 DOWNTO 0);
31 SIGNAL SUB_ADDN : STD_LOGIC;
32 SIGNAL CIN_TMP : SIGNED(1 DOWNTO 0);
33
34 BEGIN
35
36 SUB_ADDN <= SUB_ADDN_SX OR SUB_ADDN_DX;
37 CIN_TMP <= '0' & SUB_ADDN; -- USO QUESTO SEGNALE TEMPORANEO PER POTER FARE LA SOMMA, NON
38   POTREI CONVERTIRE UNO STD_LOGIC IN SIGNED (NE UNSIGNED)
39
40 SUM_OUT_TMP <= (MUX_SX_OUT + MUX_DX_OUT + CIN_TMP);
41
42 -- SEGNALI AGGIUNTI PER NON AVERE ERRORE SU MODELSIM (DICEVA "NOT GLOBALLY STATIC")
43 SX_NEG <= NOT(INPUT_SX);
44 DX_NEG <= NOT(INPUT_DX);
45
46 MUX_SX : MUX_2TO1_Nbit GENERIC MAP(N => 42)
47   PORT MAP(INPUT_0 => INPUT_SX,
48             INPUT_1 => SX_NEG,
49             SEL => SUB_ADDN_SX,
50             MUX_OUT => MUX_SX_OUT);
51
52 MUX_DX : MUX_2TO1_Nbit GENERIC MAP(N => 42)
53   PORT MAP(INPUT_0 => INPUT_DX,
54             INPUT_1 => DX_NEG,
55             SEL => SUB_ADDN_DX,
56             MUX_OUT => MUX_DX_OUT);
57
58 REG_PIPE : Register_Nbit GENERIC MAP(N => 42)
59   PORT MAP(CLOCK => CLOCK,
60             ENABLE => '1',
61             R => SUM_OUT_TMP,
62             Q => SUM_OUT);
63
64 END ARCHITECTURE;

```

12.6 Rounder.vhd

```

1  -- QUESTO CODICE E' STATO PRODOTTO DA:
2  -- Fichera Antonio, matricola: 337213
3  -- Vannelli Elisabetta, matricola: 346477
4  LIBRARY IEEE;
5  USE IEEE.STD_LOGIC_1164.ALL;
6  USE IEEE.NUMERIC_STD.ALL;
7
8  ENTITY Rounder IS

```

```

9   PORT(INPUT : IN SIGNED(41 DOWNTO 0);
10  CLOCK : IN STD_LOGIC;
11  OUTPUT : OUT SIGNED(19 DOWNTO 0));
12 END ENTITY;
13
14 ARCHITECTURE Behavioural OF Rounder IS
15
16 COMPONENT Register_Nbit IS
17  GENERIC(N : POSITIVE);
18  PORT(CLOCK, ENABLE : IN STD_LOGIC;
19    R : IN SIGNED(N-1 DOWNTO 0);
20    Q : OUT SIGNED(N-1 DOWNTO 0));
21 END COMPONENT;
22
23 SIGNAL TRUNCATED_IN, ADDEND, SUM_TMP : SIGNED(40 DOWNTO 0);
24
25 BEGIN
26
27 TRUNCATED_IN <= INPUT(41 DOWNTO 1); -- SCALO DI 1 BIT PER RIPORTARE IL NUMERO NELLA DINAMICA
28   CORRETTA (STO DIVIDENDO PER 2)
29 ADDEND(40 DOWNTO 19) <= (OTHERS => '0');
30 ADDEND(17 DOWNTO 0) <= (OTHERS => '0');
31 ADDEND(18) <= '1'; -- 2 ** -20 = 1/2 LSB
32
33 SUM_TMP <= (TRUNCATED_IN + ADDEND); -- ARROTONDAMENTO HALF-UP
34
35 REG_PIPE : Register_Nbit GENERIC MAP(N => 20)
36   PORT MAP (CLOCK => CLOCK,
37             ENABLE => '1',
38             R => SUM_TMP(38 DOWNTO 19), -- TRONCAMENTO FINALE
39             Q => OUTPUT);
40
41 END ARCHITECTURE;

```

12.7 Register_Nbit.vhd

```

1  -- QUESTO CODICE E' STATO PRODOTTO DA:
2  -- Fichera Antonio, matricola: 337213
3  -- Vannelli Elisabetta, matricola: 346477
4  LIBRARY IEEE;
5  USE IEEE.STD_LOGIC_1164.ALL;
6  USE IEEE.NUMERIC_STD.ALL;
7
8  ENTITY Register_Nbit IS
9    GENERIC(N : POSITIVE);
10   PORT(CLOCK, ENABLE : IN STD_LOGIC;
11     R : IN SIGNED(N-1 DOWNTO 0);
12     Q : OUT SIGNED(N-1 DOWNTO 0));
13 END ENTITY;
14
15 ARCHITECTURE Behavioural OF Register_Nbit IS
16
17 BEGIN

```

```

18
19 PROCESS(CLOCK)
20 BEGIN
21 IF(CLOCK'EVENT AND CLOCK = '1') THEN
22     IF(ENABLE = '1') THEN
23         Q <= R;
24     END IF;
25 END IF;
26 END PROCESS;
27
28 END ARCHITECTURE;

```

12.8 MUX_2TO1_Nbit.vhd

```

1   -- QUESTO CODICE E' STATO PRODOTTO DA:
2   -- Fichera Antonio, matricola: 337213
3   -- Vannelli Elisabetta, matricola: 346477
4   LIBRARY IEEE;
5   USE IEEE.STD_LOGIC_1164.ALL;
6   USE IEEE.NUMERIC_STD.ALL;
7
8   ENTITY MUX_2TO1_Nbit IS
9       GENERIC(N : POSITIVE);
10      PORT(INPUT_0, INPUT_1 : IN SIGNED(N-1 DOWNTO 0);
11          SEL : IN STD_LOGIC;
12          MUX_OUT : OUT SIGNED(N-1 DOWNTO 0));
13      END ENTITY;
14
15  ARCHITECTURE Behavioural OF MUX_2TO1_Nbit IS
16
17  BEGIN
18
19  MUX_OUT <= INPUT_0 WHEN SEL <= '0' ELSE INPUT_1;
20
21  END ARCHITECTURE;

```

12.9 Decoder_4to16.vhd

```

1   -- QUESTO CODICE E' STATO PRODOTTO DA:
2   -- Fichera Antonio, matricola: 337213
3   -- Vannelli Elisabetta, matricola: 346477
4   LIBRARY IEEE;
5   USE IEEE.STD_LOGIC_1164.ALL;
6   USE IEEE.NUMERIC_STD.ALL;
7
8   ENTITY Decoder_4to16 IS
9       PORT(DEC_IN : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
10          DEC_OUT : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
11      END ENTITY;
12
13  ARCHITECTURE Behavioural OF Decoder_4to16 IS

```

```

14
15 BEGIN
16
17 PROCESS(DEC_IN)
18 BEGIN
19     DEC_OUT <= (OTHERS => '0');
20     DEC_OUT(TO_INTEGER(UNSIGNED(DEC_IN))) <= '1';
21 END PROCESS;
22
23 END ARCHITECTURE;

```

12.10 EVEN_ROM.vhd

```

1 -- QUESTO CODICE E' STATO PRODOTTO DA:
2 -- Fichera Antonio, matricola: 337213
3 -- Vannelli Elisabetta, matricola: 346477
4 LIBRARY IEEE;
5 USE IEEE.STD_LOGIC_1164.ALL;
6 USE IEEE.NUMERIC_STD.ALL;
7
8 ENTITY EVEN_ROM IS
9     PORT(ROW_SELECT : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
10         DATA_OUT : OUT STD_LOGIC_VECTOR(25 DOWNTO 0));
11 END ENTITY;
12
13 ARCHITECTURE Behavioural OF EVEN_ROM IS
14     TYPE ROM_TYPE IS ARRAY(0 TO 15) OF STD_LOGIC_VECTOR(25 DOWNTO 0);
15     CONSTANT ROM_DATA : ROM_TYPE := (-- struttura : DONE, REG_IN_LE, DP_CONTROLS, REG_OUT_LE,
16         NEXT_STATUS
17         "010000000000100000000000000000", -- locazione 0 --> STATUS = IDLE -> 00000
18         "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU", -- locazione 1 --> NON USATA
19         "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU", -- locazione 2 --> NON USATA
20         "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU", -- locazione 3 --> NON USATA
21         "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU", -- locazione 4 --> NON USATA
22         "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU", -- locazione 5 --> NON USATA
23         "00101010100001000000001110", -- locazione 6 --> STATUS = CONT_1 -> 01100
24         "00010101000111101000010000", -- locazione 7 --> STATUS = CONT_2 -> 01110
25         "000000000011111000100010010", -- locazione 8 --> STATUS = CONT_3 -> 10000
26         "000000000011110000010010100", -- locazione 9 --> STATUS = CONT_4 -> 10010
27         "0000000000110110000001010110", -- locazione 10 --> STATUS = CONT_5 -> 10100
28         "110000000000001010000101101", -- locazione 11 --> STATUS = CONT_6 -> 10110
29         "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU", -- locazione 12 --> NON USATA
30         "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU", -- locazione 13 --> NON USATA
31         "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU", -- locazione 14 --> NON USATA
32         "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU" -- locazione 15 --> NON USATA
33 );
34
35 BEGIN
36
37 PROCESS(ROW_SELECT)
38 VARIABLE INDEX : INTEGER := 0; -- POTREI PORLO UGUALE A -1 PER DISCRIMINARE IL CASO I CUI
        NESSUNA LOCAZIONE SIA SELEZIONATA
39 BEGIN

```

```

39
40 FOR i IN 0 TO 15 LOOP
41     IF(ROW_SELECT(i) = '1') THEN
42         INDEX := i;
43     END IF;
44 END LOOP;
45
46 DATA_OUT <= ROM_DATA(INDEX);
47
48 END PROCESS;
49
50 END ARCHITECTURE;

```

12.11 ODD_ROM.vhd

```

1 -- QUESTO CODICE E' STATO PRODOTTO DA:
2 -- Fichera Antonio, matricola: 337213
3 -- Vannelli Elisabetta, matricola: 346477
4 LIBRARY IEEE;
5 USE IEEE.STD_LOGIC_1164.ALL;
6 USE IEEE.NUMERIC_STD.ALL;
7
8 ENTITY ODD_ROM IS
9     PORT(ROW_SELECT : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
10          DATA_OUT : OUT STD_LOGIC_VECTOR(25 DOWNTO 0));
11 END ENTITY;
12
13 ARCHITECTURE Behavioural OF ODD_ROM IS
14 TYPE ROM_TYPE IS ARRAY(0 TO 15) OF STD_LOGIC_VECTOR(25 DOWNTO 0);
15 CONSTANT ROM_DATA : ROM_TYPE := (-- struttura : DONE, CC, REG_IN LE, DP_CONTROLS, REG_OUT LE
16 , NEXT_STATUS
17 "00101010000010000000000011", -- locazione 0 --> STATUS = GET_REAL -> 00001
18 "00010101000110000000000101", -- locazione 1 --> STATUS = GET_IMAG -> 00011
19 "00000000001110000000000111", -- locazione 2 --> STATUS = DO_M3 -> 00101
20 "000000000011110000000001001", -- locazione 3 --> STATUS = DO_M2 -> 00111
21 "000000000110110000000001011", -- locazione 4 --> STATUS = DO_M4 -> 01001
22 "01000000000001010000001101", -- locazione 5 --> STATUS = DO_M5 -> 01011
23 "000000000100001000000001111", -- locazione 6 --> STATUS = DO_M6 -> 01101
24 "000000000000011101000010001", -- locazione 7 --> STATUS = DO_S5 -> 01111
25 "000000000000011100100010011", -- locazione 8 --> STATUS = DO_S6 -> 10001
26 "000000000000010000010010101", -- locazione 9 --> STATUS = WAIT_BR -> 10011
27 "000000000000010000001010111", -- locazione 10 --> STATUS = WAIT_BI -> 10101
28 "100000000000010000000111001", -- locazione 11 --> STATUS = FINISH -> 10111
29 "100000000000010000000011011", -- locazione 12 --> STATUS = SEND_R -> 11001
30 "100000000000010001000000000", -- locazione 13 --> STATUS = SEND_I -> 11011
31 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUU", -- locazione 14 --> NON USATA
32 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUU" -- locazione 15 --> NON USATA
33 );
34 BEGIN
35
36 PROCESS(ROW_SELECT)

```

```

37  VARIABLE INDEX : INTEGER := 0; -- POTREI PORLO UGUALE A -1 PER DISCRIMINARE IL CASO IN CUI
38      NESSUNA LOCAZIONE SIA SELEZIONATA
39
40  BEGIN
41
42  FOR i IN 0 TO 15 LOOP
43      IF(ROW_SELECT(i) = '1') THEN
44          INDEX := i;
45      END IF;
46  END LOOP;
47
48  DATA_OUT <= ROM_DATA(INDEX);
49
50  END PROCESS;
51
52  END ARCHITECTURE;

```

12.12 LateStatus_PLA.vhd

```

1  -- QUESTO CODICE E' STATO PRODOTTO DA:
2  -- Fichera Antonio, matricola: 337213
3  -- Vannelli Elisabetta, matricola: 346477
4  LIBRARY IEEE;
5  USE IEEE.STD_LOGIC_1164.ALL;
6
7  ENTITY LateStatus_PLA IS
8      PORT(STATUS, CC, LSB_IN : IN STD_LOGIC;
9             CC_Validation, LSB_OUT : OUT STD_LOGIC);
10 END ENTITY;
11
12 ARCHITECTURE Behavioural OF LateStatus_PLA IS
13
14 COMPONENT MUX_2T01 IS
15     PORT(INPUT_0, INPUT_1 : IN STD_LOGIC;
16           SEL : IN STD_LOGIC;
17           MUX_OUT : OUT STD_LOGIC);
18 END COMPONENT;
19
20 SIGNAL NOT_LSB_IN, SEL_MUX : STD_LOGIC;
21
22 BEGIN
23
24 NOT_LSB_IN <= NOT(LSB_IN);
25 SEL_MUX <= CC AND STATUS;
26 CC_Validation <= SEL_MUX;
27
28 MUX : MUX_2T01 PORT MAP
29     (INPUT_0 => LSB_IN,
30      INPUT_1 => NOT_LSB_IN,
31      SEL => SEL_MUX,
32      MUX_OUT => LSB_OUT);
33
34 END ARCHITECTURE;

```

12.13 Register_Nbit_SLV.vhd

```

1  -- QUESTO CODICE E' STATO PRODOTTO DA:
2  -- Fichera Antonio, matricola: 337213
3  -- Vannelli Elisabetta, matricola: 346477
4  LIBRARY IEEE;
5  USE IEEE.STD_LOGIC_1164.ALL;
6
7  ENTITY Register_Nbit_SLV IS
8      GENERIC(N : POSITIVE);
9      PORT(CLOCK, ENABLE, RESET : IN STD_LOGIC;
10             R : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
11             Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0));
12 END ENTITY;
13
14 ARCHITECTURE Behavioural OF Register_Nbit_SLV IS
15
16 BEGIN
17
18 PROCESS(CLOCK, RESET)
19 BEGIN
20     IF(RESET = '0') THEN
21         Q <= (OTHERS => '0');
22     ELSIF(CLOCK'EVENT AND CLOCK = '1') THEN
23         IF(ENABLE = '1') THEN
24             Q <= R;
25         END IF;
26     END IF;
27 END PROCESS;
28
29 END ARCHITECTURE;

```

12.14 MUX_2TO1_Nbit_SLV.vhd

```

1  -- QUESTO CODICE E' STATO PRODOTTO DA:
2  -- Fichera Antonio, matricola: 337213
3  -- Vannelli Elisabetta, matricola: 346477
4  LIBRARY IEEE;
5  USE IEEE.STD_LOGIC_1164.ALL;
6
7  ENTITY MUX_2TO1_Nbit_SLV IS
8      GENERIC(N : POSITIVE);
9      PORT(INPUT_0, INPUT_1 : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
10             SEL : IN STD_LOGIC;
11             MUX_OUT : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0));
12 END ENTITY;
13
14 ARCHITECTURE Behavioural OF MUX_2TO1_Nbit_SLV IS
15
16 BEGIN
17
18 MUX_OUT <= INPUT_0 WHEN SEL <= '0' ELSE INPUT_1;
19

```

20 **END ARCHITECTURE;**

12.15 MUX_2TO1.vhd

```
1 -- QUESTO CODICE E' STATO PRODOTTO DA:  
2 -- Fichera Antonio, matricola: 337213  
3 -- Vannelli Elisabetta, matricola: 346477  
4 LIBRARY IEEE;  
5 USE IEEE.STD_LOGIC_1164.ALL;  
6  
7 ENTITY MUX_2TO1 IS  
8 PORT(INPUT_0, INPUT_1 : IN STD_LOGIC;  
9      SEL : IN STD_LOGIC;  
10     MUX_OUT : OUT STD_LOGIC);  
11 END ENTITY;  
12  
13 ARCHITECTURE Behavioural OF MUX_2TO1 IS  
14  
15 BEGIN  
16  
17 MUX_OUT <= INPUT_0 WHEN SEL <= '0' ELSE INPUT_1;  
18  
19 END ARCHITECTURE;
```