Prova Finale di Reti Logiche 2021/2022

Leonardo De Clara

A.A. 2021/2022

Matricola:	933527
Codice Persona:	10686418
Docente:	William Fornaciari

Indice

1 Introduzione

1.1 Descrizione Del Progetto

La Prova Finale di Reti Logiche 2021/2022 consiste nell'implementazione di un modulo hardware descritto in VHDL in grado di ricevere in ingresso una sequenza continue di parole di 8 bit su cui viene applicato il codice convoluzionale $\frac{1}{2}$, restituito successivamente in uscita come una sequenza di parole di 8 bit.
In particolare, ogni parola in ingresso viene serializzata, generando in questo modo un flusso continuo di singoli bit in ingresso al codificatore. Il convolutore dunque trasmette in uscita una sequenza ottenuta come concatenamento dei due bit ottenuti dall'applicazione della codifica al bit trasmesso in input, la quale verrà quindi memorizzata in memoria in parole da 8 bit.
Il codificatore convoluzionale è una macchina sequenziale sincrona con clock globale e segnale di reset descritto dal seguente diagramma degli stati.

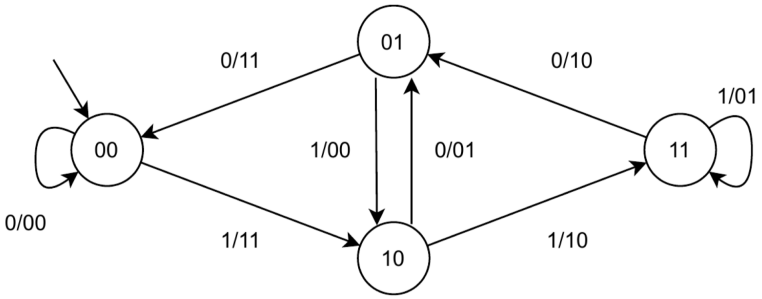


Figura 1: Diagramma a stati del codificatore convoluzionale

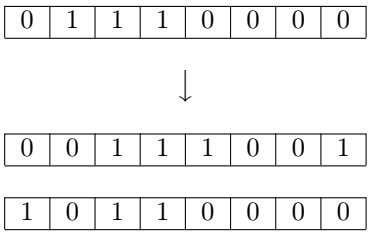


Figura 2: Esempio di applicazione del codice convoluzionale

Al componente viene richiesto di comunicare con una memoria sincrona con indirizzamento al byte per leggere il numero di parole da elaborare, registrato all'indirizzo 0, il contenuto della sequenza in ingresso, memorizzata a partire dall'indirizzo 1, e per scrivere la sequenza in uscita a partire dall'indirizzo 1000.

Numero parole	Indirizzo 0
Parola input 1	Indirizzo 1
Parola input 2	Indirizzo 2
..	..
Parola output 1	Indirizzo 1000
Parola output 2	Indirizzo 1001
Parola output 3	Indirizzo 1002
Parola output 4	Indirizzo 1003
..	..

Figura 3: Indirizzamento della memoria

Il modulo è stato progettato per poter codificare più flussi in ingresso: tra un'elaborazione e la successiva il codificatore convoluzionale verrà riportato nel suo stato iniziale, la quantità di parole si troverà nuovamente all'indirizzo 0 e la scrittura avverrà sempre a partire dell'indirizzo 1000.

Per la realizzazione del progetto è stata utilizzato lo strumento di sintesi Xilinx Vivado Webpack e la FPGA xc7a200tfbg484-1.

2 Architettura

2.1 Descrizione ad alto livello

Per la progettazione del componente si è scelto di utilizzare un'architettura modulare costituita da un modulo dedicato all'unità di elaborazione, il Datapath, e un modulo dedicato all'unità di controllo, la FSM.
L'implementazione esegue i seguenti passi:

1. Lettura del primo indirizzo di memoria e memorizzazione in un registro del numero di parole da elaborare.
2. Lettura del secondo indirizzo e memorizzazione in un registro della parola a cui sarà applicato il codice convoluzionale.
3. Selezione e conversione dei primi quattro bit della parola corrente.
4. Scrittura in memoria della parola generata a partire dai primi quattro bit della parola corrente.
5. Selezione e conversione dei quattri bit rimanenti della parola corrente.
6. Scrittura in memoria della parola generata a partire dai rimanenti quattro bit della parola corrente.
7. Se sono ancora presenti in memoria parole da convertire vengono ripetuti i passaggi dal 2 in poi.
8. Aggiornamento dei segnali di terminazione.

2.2 Datapath

Il componente Datapath è realizzato attraverso una collezione di process che si occupano di controllare, attraverso opportuni segnali, il corretto caricamento ed aggiornamento dei valori nei rispetti registri. Il suo funzionamento può essere diviso in tre moduli logici dedicati a specifiche operazioni:

1. Aggiornamento numero di parole da convertire.
2. Gestione indirizzi di lettura e scrittura.
3. Selezione e conversione della sequenza in ingresso.

Tutti i multiplexer sono realizzati utilizzando il costrutto `with-select`.

2.2.1 Aggiornamento numero parole

Questo modulo è composto da un registro `reg1` contenente il numero residuo di parole da convertire. Il contenuto in ingresso al registro viene selezionato da un multiplexer comandato dal segnale `r1_sel`, il quale permette di trasmettere il valore proveniente dalla memoria attraverso `i_data` oppure il contenuto di `reg1` decrementato di 1 attraverso un sottrattore collegato in uscita al registro. Il contenuto del registro in questione viene dato in ingresso ad un comparatore, il quale trasmette il segnale `o_end=1` nel caso il valore in uscita da `reg1` fosse pari a 0, `o_end=0` altrimenti.

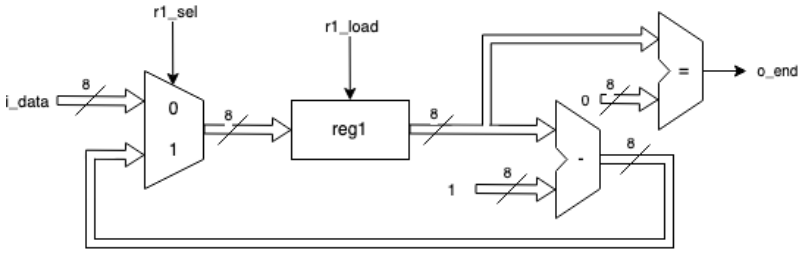


Figura 4: Schema rappresentativo del primo modulo

2.2.2 Gestione indirizzi

Gli indirizzi di lettura e scrittura sono memorizzati nei registri `reg4` e `reg5`. Il contenuto di ciascun registro viene selezionato dai multiplexer comandati rispettivamente da `r4_sel` e `r5_sel`: il primo multiplexer riceve in ingresso `i_addr`, pari a 0 su 16 bit, e il valore trasmesso in uscita da un sommatore collegato all'uscita di `reg4`, pari al valore del registro incrementato di 1; il secondo si occupa di selezionare un ingresso tra `wr_addr`, pari a 1000 su 16 bit, e il valore in uscita da un sommatore collegato all'uscita di `reg5`, pari al valore del registro incrementato di 1. I valori memorizzati nei due registri costituiscono inoltre gli ingressi del multiplexer comandato dal segnale `mem_sel`, il quale si occupa di trasmettere il valore di `o_address`, contenente l'indirizzo di memoria su cui scriverà o leggerà il componente.

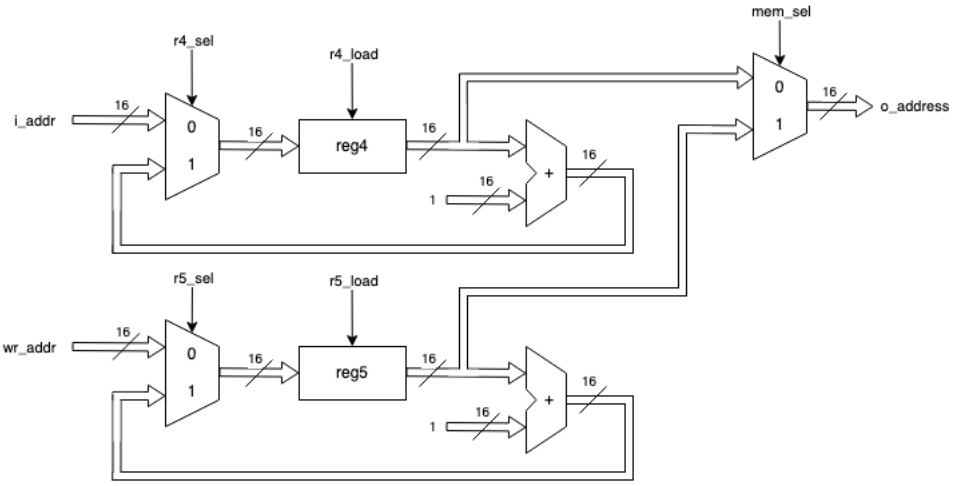


Figura 5: Schema rappresentativo del secondo modulo

2.2.3 Conversione

La gestione delle sequenze in ingresso con conseguente generazione dei valori in uscita è assegnata ad un modulo contenente la macchina a stati che descrive il codificatore convoluzionale. Il modulo riceve una parola da 8 bit dalla memoria attraverso il segnale `i_data`, il cui contenuto viene salvato nel registro `reg_2`. I singoli 8 bit salvati costituiscono gli ingressi ad un multiplexer che, attraverso il segnale di selezione a 3 bit `o_r2_sel`, restituisce in uscita al codificatore il bit da convertire. In base allo stato e al bit in ingresso il blocco convoluzionale trasmette in uscita l'output previsto costituito da due bit: nel caso il bit processato fosse il primo o il quinto relativamente alla parola corrente l'output del convolutore viene sommato a 8 bit pari a 0, selezionati attraverso un multiplexer comandato da `d_sel`. L'uscita del sommatore viene memorizzata nel registro `reg_3`: il dato memorizzato nel registro costituirà il contenuto di `o_data`, il segnale contenente il dato che scrivere in memoria. Questo dato viene trasmesso in ingresso ad un blocco `2_sll` responsabile per lo shift logico di due posizioni del suo contenuto. Questo segnale costituisce il secondo ingresso del multiplexer comandato da `d_sel`: il costrutto così descritto permette la concatenazione dei bit in uscita dal codificatore.

Il blocco convoluzionale è stato implementato attraverso una coppia di process CONV_STATE e CONV_LAMBDA, il primo responsabile l'aggiornamento del suo stato, il secondo per la funzione stato prossimo e per il calcolo dell'output.

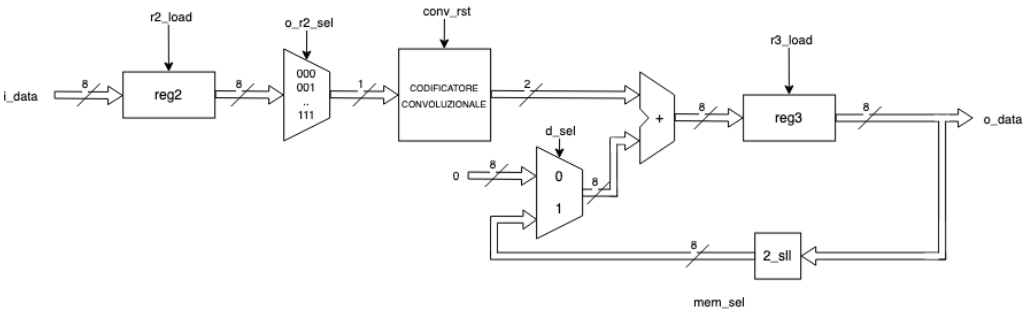
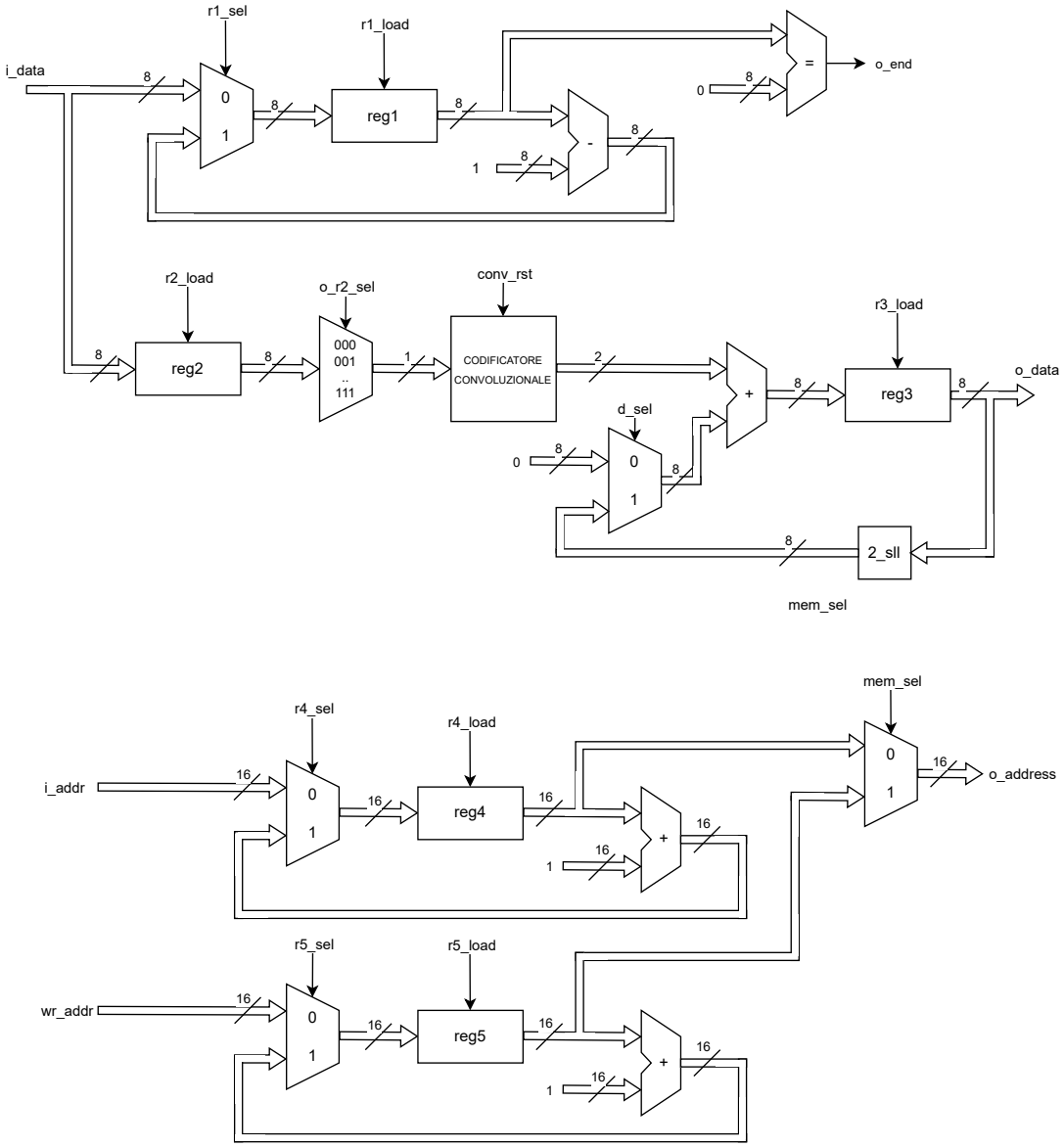


Figura 6: Schema rappresentativo del terzo modulo

Figura 7: Rappresentazione grafica del Datapath



2.3 Macchina a Stati Finiti

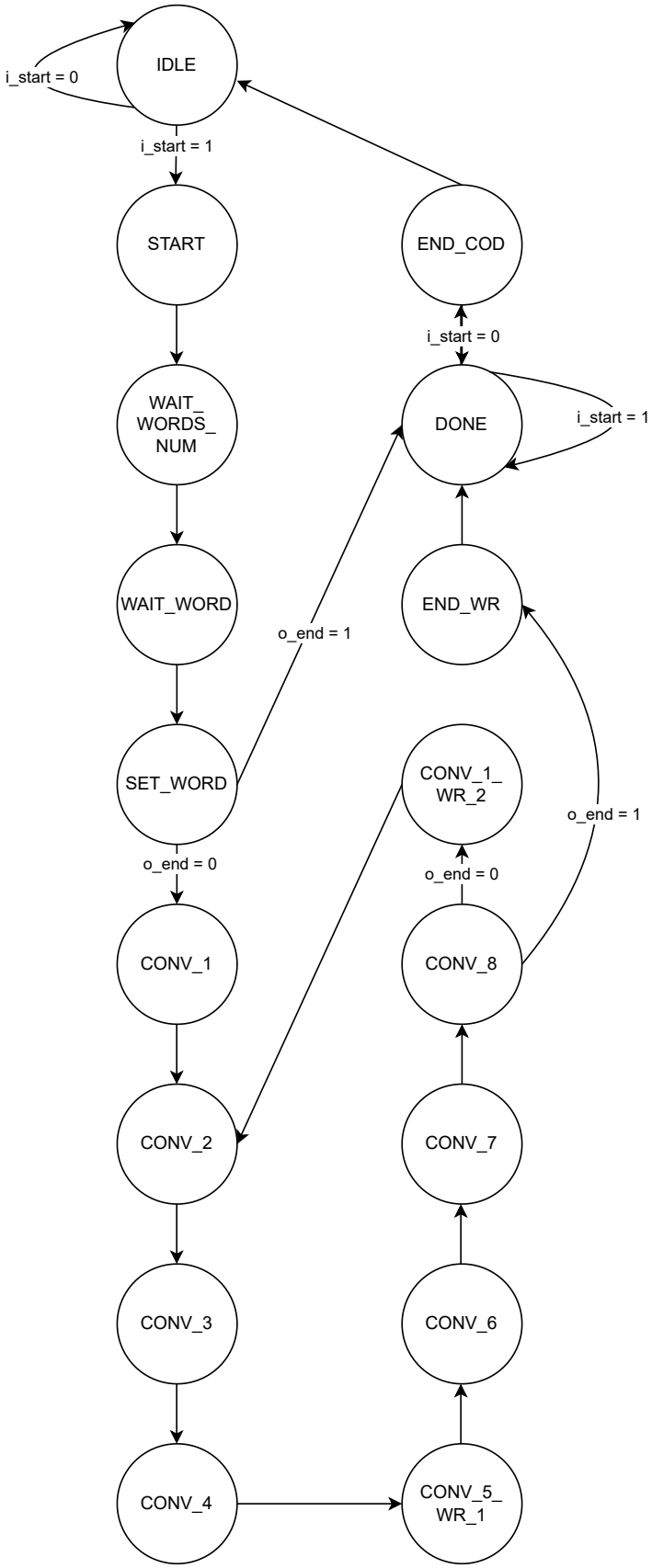
La Macchina a Stati Finiti (FSM) è realizzata con specifica Behavioral attraverso tre process:

1. DELTA: dedicato al calcolo della funzione stato prossimo, sensibile al signal `cur_state`, `i_start` e `o_end`.
2. STATE_SIGNALS: responsabile dell'aggiornamento dei segnali interni e delle uscite, sensibile al signal `cur_state` indicante lo stato corrente della FSM.
3. FSM_STATE: responsabile per l'aggiornamento dello stato della FSM, sensibile al segnale di reset `i_rst` e al clock `i_clk`.

Segue una descrizione degli stati della FSM:

- `IDLE`: Stato di idle in cui si trova la FSM in attesa che venga asserito il segnale `i_start`;
- `START`: Stato in cui viene asserito il segnale di reset al codificatore convoluzionale; inoltre vengono salvati nei rispettivi registri gli indirizzi di lettura e scrittura e iniziali;
- `WAIT_WORDS_NUM`: Stato in cui viene richiesto alla memoria il numero di parole da processare; viene aggiornato l'indirizzo della prossima cella di memoria in lettura;
- `WAIT_WORD`: Stato in cui viene salvato il numero di parole; viene richiesta alla memoria la prima parola salvata;
- `SET_WORD`: Stato in cui viene salvata nel registro dedicato la prima parola da processare; viene aggiornato l'indirizzo della prossima cella di memoria in lettura;
- `CONV_1`: Stato in cui viene processato il bit più significativo della prima parola e viene salvato in un registro il risultato restituito dal blocco convoluzionale;
- `CONV_2`: Stato in cui viene processato il secondo bit più significativo della parola corrente;
- `CONV_3`: Stato in cui viene processato il terzo bit più significativo della parola corrente;
- `CONV_4`: Stato in cui viene processato il quarto bit più significativo della parola corrente;
- `CONV_5_WR_1`: Stato in cui viene processato il quinto bit più significativo della parola corrente; viene scritta in memoria la parola ottenuta dal concatenamento degli ultimi quattro gruppi di 2 bit restituiti dal codificatore e viene aggiornato l'indirizzo della prossima cella di memoria in scrittura;
- `CONV_6`: Stato in cui viene processato il sesto bit più significativo della parola corrente;

- `CONV_7`: Stato in cui viene processato il settimo bit più significativo della parola corrente; viene richiesto alla memoria il contenuto dell'indirizzo della prossima parola e viene aggiornato il numero residuo di parole da processare;
- `CONV_8`: Stato in cui viene processato l'ultimo bit della parola corrente;
- `CONV_1_WR_2`: Stato in cui viene processato il bit più significativo della parola successiva; viene scritta in memoria la parola ottenuta dal concatenamento degli ultimi quattro gruppi di 2 bit restituiti dal codificatore e viene aggiornato l'indirizzo della prossima cella di memoria in scrittura;
- `END_WR`: Stato in cui viene scritta in memoria l'ultima parola ottenuta dal concatenamento degli ultimi quattro gruppi di 2 bit restituiti dal codificatore;
- `DONE`: Stato in cui viene asserito il segnale `o_done` e si attende che il segnale `i_start` venga posto uguale a 0;
- `END_COD`: Stato in cui viene posto `i_done=0`, riportandosi allo stato `IDLE`.


IDLE
i_start = 0
i_start = 1
START
WAIT_
WORDS_
NUM
WAIT_WORD
SET_WORD
o_end = 0
CONV_1
CONV_2
CONV_3
CONV_4
o_end = 1
END_COD
i_start = 0
DONE
i_start = 1
END_WR
CONV_1_
WR_2
o_end = 0
CONV_8
o_end = 1
CONV_7
CONV_6
CONV_5_
WR_1

3 Sintesi

3.1 Report di utilizzo

Il componente è stato correttamente sintetizzato con l'utilizzo di 84 LUT, 75 Flip Flop e 0 Latch, come mostra l'*Utilization Report*.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	84	0	0	134600	0.06
LUT as Logic	84	0	0	134600	0.06
LUT as Memory	0	0	0	46200	0.00
Slice Registers	75	0	0	269200	0.03
Register as Flip Flop	75	0	0	269200	0.03
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

3.2 Report di timing

Il *Timing Summary* mette in evidenza il rispetto del vincolo di durata del clock: il componente è in grado di funzionare con periodi di clock notevolmente al di sotto del limite previsto (100 ns), risultato dimostrato dal valore del *Worst Negative Slack* (WNS), definito come *required time - arrival time*.

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints
95.376	0.000	0	137

WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints
0.164	0.000	0	137

WPWS(ns)	TPWS(ns)	TPWS Failing Endpoints	TPWS Total Endpoints
4.500	0.000	0	82

4 Testing

I test effettuati sono stati progettati con l'obiettivo di rivelare criticità nell'implementazione del modello e di verificare possibili condizioni limite, in modo da guidare l'evoluzione del componente fino alla sua versione finale. In particolare sono stati effettuati test generati automaticamente e test mirati, volti ad accertare il correttamento funziona dell'architettura in situazioni d'interesse. Fanno parte di questa classe i seguenti test:

- Test 0 parole: l'indirizzo 0 contiene una parola pari al byte nullo, non viene effettuata alcuna codifica.
- Test 1 parola: la memoria contiene una sola parola da codificare.
- Test 2 parole: la memoria contiene due parole da codificare.
- Test reset asincrono: viene verificato il corretto comportamento nel caso venga asserito il segnale di reset del componente.
- Test reset tra computazioni: al termine di una computazione viene asserito il segnale di reset.
- Test multistart: viene verificata la capacità del componente di gestire in maniera corretta una sequenza di elaborazioni.

Il componente programmato è in grado correttamente tutti i test elencati in precedenza sia a livello *Behavioral* che *Post Synthesis*.

5 Conclusioni

È stato osservato che l'architettura progettata rispetta accuratamente la specifica fornita, valutazione confermata dai risultati ottenuti tramite estensivo testing. Si può quindi affermare che il componente prodotto è in grado di interagire in maniera corretta e precisa con una memoria avente indirizzamento al byte ed è in grado di eseguire la codifica prevista dal convolutore.
Dal punto di vista di design l'implementazione attraverso due moduli distinti - unità di elaborazione ed unità di controllo - è risultata funzionale e ha semplificato fortemente il processo di progettazione.
Possono essere realizzati eventuali perfezionamenti all'architettura progettata attraverso la riduzione del tempo di elaborazione e la diminuzione del numero di LUT e FF inferiti.