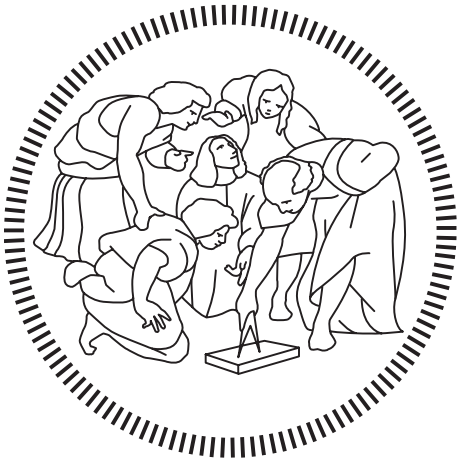
Politecnico di Milano

Prova Finale

Progetto di Reti Logiche



A.A. 2021-2022

Diego Pini

Codice persona 10668724

Matricola 932251

Indice

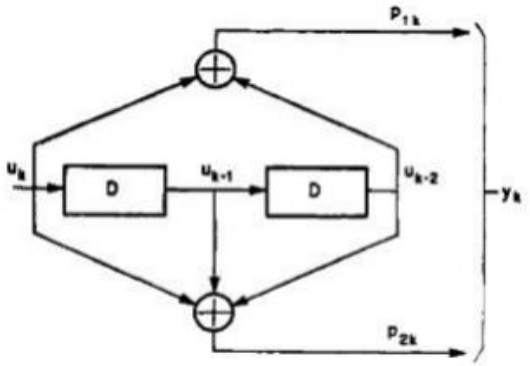
1 Introduzione

1.1 Specifica del progetto

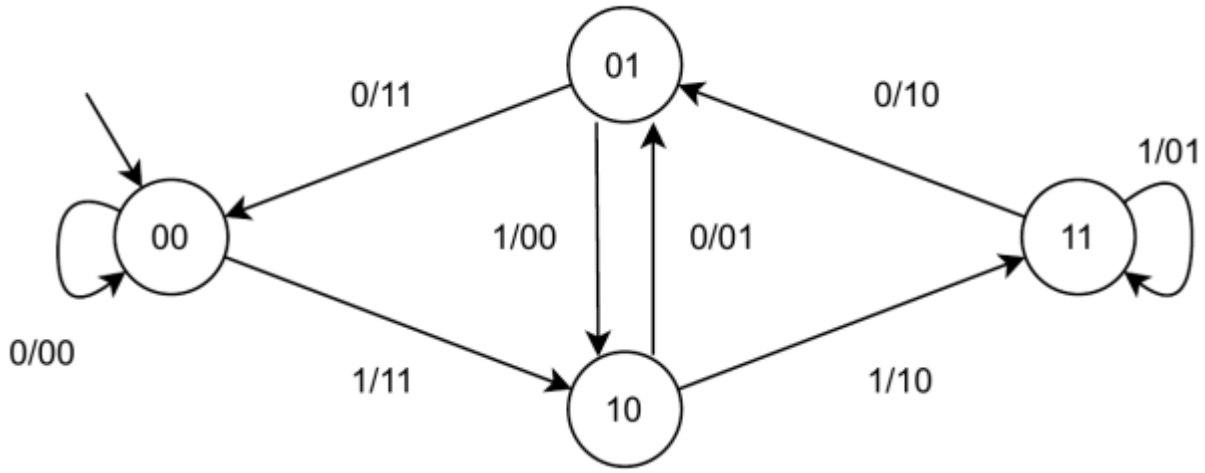
Il progetto richiesto consiste nell'implementazione in VHDL di un Codificatore convoluzionale con tasso di trasmissione ½. Il componente riceve un flusso continuo di W parole da 8 bit, e ne restituisce in uscita una sequenza di Z parole da 8 bit.
Essendo un Codificatore convoluzionale con tasso di trasmissione ½, per ogni bit in entrata ne vengono prodotti 2 in uscita, data una parola W difatti una volta finita la codifica si ottengono 2 parole Z, le quali una volta concatenate generano un flusso di uscita Y composto sempre da 2 parole da 8 bit ciascuna.

Nella figura seguente viene indicato come parola di ingresso U_k, mentre U_{k+1} e U_{k+2} indicano rispettivamente i bit di entrata sottoposti a due ritardi differenti.
Che nel nostro caso partendo da $U_k = (b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7)$ si ottiene:
$U_{k+1} = (b_{-1}\ b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6)$ e $U_{k+2} = (b_{-2}\ b_{-1}\ b_0\ b_1\ b_2\ b_3\ b_4\ b_5)$.
Nella lettura della prima parola da codificare, b_{-2} e b_{-1} non esistono, si suppone dunque che il componente sia spento e che quindi essi abbiano valore di 0.
Le uscite dell'operazione convoluzionale invece vengono indicati con P_{k1} e P_{k2}, essi sono ottenuti dalle operazioni logiche di XOR fra l'ingresso (U_k) e i suoi ritardi (U_{k+1} e U_{k+2}). La loro concatenazione genera y_k, l'output del nostro componente.

$P_{k1} = \{p_{10}\ p_{11}\ p_{12}\ p_{13}\ p_{14}\ p_{15}\ p_{16}\ p_{17}\}$
$P_{k2} = \{p_{20}\ p_{21}\ p_{22}\ p_{23}\ p_{24}\ p_{25}\ p_{26}\ p_{27}\}$
$y_k = \{p_{10}\ p_{20}\ p_{11}\ p_{21}\ p_{12}\ p_{22}\ p_{13}\ p_{23}\ p_{14}\ p_{24}\ p_{15}\ p_{25}\ p_{16}\ p_{26}\ p_{17}\ p_{27}\}$
Output1 = $\{p_{10}\ p_{20}\ p_{11}\ p_{21}\ p_{12}\ p_{22}\ p_{13}\ p_{23}\}$
Output2 = $\{p_{14}\ p_{24}\ p_{15}\ p_{25}\ p_{16}\ p_{26}\ p_{17}\ p_{27}\}$



Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$.



Machina di Mealy del codificatore convoluzionale con tasso di trasmissione ½

Un esempio pratico:

U_k = {10100010}

U_{k+1} = {01010001}

U_{k+2} = {00101000}

P_{k1} = U_k XOR U_{k+2} = {10001010}

P_{k2} = U_k XOR U_{k+1} XOR U_{k+2} = {11011011}

Concatenando P_{k1} e P_{k2} si ottengono:

Output1 = {11010001}

Output2 = {11001101}

1.2 Descrizione Memoria

Il modulo si interfaccia con una memoria RAM avente un indice di indirizzamento al byte per leggere e scrivere informazioni.

- Nell'indirizzo 0 è contenuto il numero di parole da codificare, con un limite fino a 255 parole.
- Dall'indirizzo 1 fino all'indirizzo specificato nel contenuto dell'indirizzo 0, vi sono le parole da codificare.
- Dopo aver codificato i bit con la codifica convoluzionale ½ le parole in uscita dal modulo vengono salvate a partire dall'indirizzo 1000.

Esempio:

W: 10100010 01001011
Z: 11010001 11001101 11110111 11010010

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	2	\\ Byte lunghezza sequenza di ingresso
1	162	\\ primo Byte sequenza da codificare
2	75	
[...]		
1000	209	\\ primo Byte sequenza di uscita
1001	205	
1002	247	
1003	210	

1.3 Specifiche del componente

```
entity project_reti_logiche is
      port (
            i_clk     : in std_logic;
            i_rst     : in std_logic;
            i_start   : in std_logic;
            i_data    : in std_logic_vector(7 downto 0);
            o_address : out std_logic_vector(15 downto 0);
            o_done    : out std_logic;
            o_en      : out std_logic;
            o_we      : out std_logic;
            o_data    : out std_logic_vector (7 downto 0)
      );
end project_reti_logiche;
```

- i_clk è il segnale di clock in ingresso al componente
- i_rst è il segnale di reset che inizializza la macchina
- i_start è il segnale che da inizio al processo
- i_data è il vettore di dati letto dalla memoria
- o_address è il vettore con cui il componente scrive e legge in memoria
- o_done il segnale con cui il componente comunica la fine del processo
- o_en è il segnale di Memory Enable che comunica la lettura in memoria
- o_we è il segnale di Write Enable che comunica la scrittura in memoria
- o_data è il vettore che contiene i dati da scrivere in memoria

2 Architettura

Per la realizzazione del progetto si è optato per la realizzazione di una macchina a stati finiti descritta da un solo processo, che ha come parametri nella sensitivity list il segnale di clock e di reset. Il primo risveglia il processo ad ogni sua variazione, mentre il secondo, ogni volta che viene portato a '1', re-inizializza le variabili del componente.

2.1 Descrizione ad Alto Livello

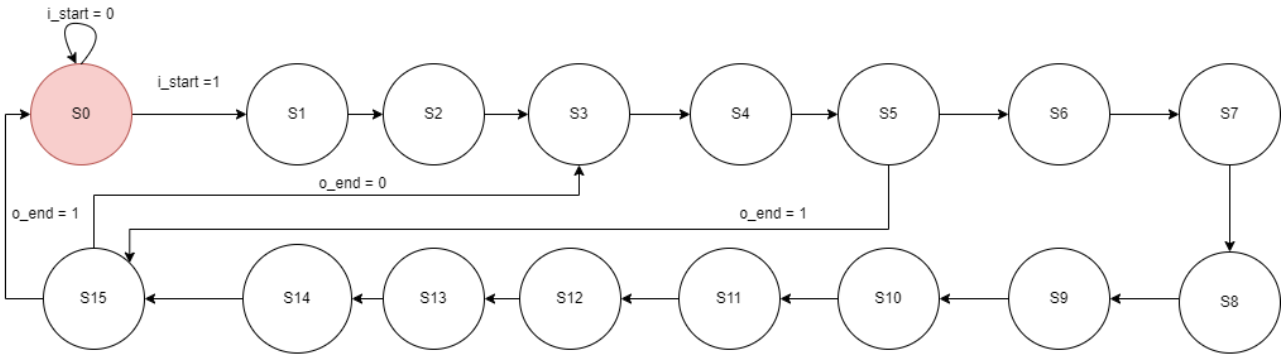
Essendo la macchina a stati finiti il componente centrale della specifica il suo compito è quello di gestire la codifica dei segnali in ingresso e in uscita del modulo, elaborandoli in modo corretto.

La macchina esegue le seguenti sequenze:

1. Legge nel primo indirizzo, l'indirizzo 0, il numero delle parole da elaborare.
2. Dall'indirizzo 1 in poi legge gli indirizzi da codificare.
3. Dopo aver letto la parola da elaborare esegue l'algoritmo convoluzionale.
4. In due stati diversi vengono scritti in output le 2 parole derivate dall'algoritmo.
5. Se non ci sono più parole da elaborare la codifica finisce, nel caso contrario la codifica riprende dal punto 2.
6. Finito il processo, il modulo rimane in attesa che il segnale di start salga a '1', per ricominciare la codifica e ripartire dal punto 1.

Inoltre, in ogni momento del processo il segnale di reset può salire a '1', in tal caso si ritorna al punto 1, iniziando così una nuova codifica.

2.2 Diagramma Macchina a Stati Finiti



2.3 Descrizione degli stati

- S0: Stato nel quale vengono inizializzati i segnali, e nel quale si attende che il segnale di start venga posto a 1 per iniziare la codifica.
- S1: Stato nel quale si porta a ‘1’ l’o_en permettendo al componente di leggere la memoria.
- S2: Stato nel quale viene letto il numero delle parole da codificare e dove si pone il segnale che permette di selezionare il valore del ritardo (rest_sel) a ‘0’.
- S3: In questo stato viene cambiato l’indirizzo facendolo passare a quello successivo dove si può trovare la parola da codificare.
- S4: Stato in cui si fa un controllo sul numero di parole da codificare rimanenti, se il numero è 0 si porta l’o_end a ‘1’ e si passa allo stato S15, altrimenti si passa a S5.
- S5: Stato nel quale viene salvata la parola da codificare e i suoi ultimi due bit.
- S6: Stato nel quale vengono impostati i ritardi, se il selettore del ritardo è uguale a ‘0’ allora il ritardo sarà di 0, se è ‘1’ il ritardo sarà ottenuto dagli ultimi due bit della parola precedente.
- S7: Prima parte delle operazioni di codifica.
- S8: Seconda parte dell’operazione di codifica.
- S9: Stato nel quale viene scritto in memoria la prima parola uscenti dalla codifica.
- S10: Stato nel quale viene sottratto 1 al numero delle parole da codificare, calcolando quindi le parole rimanenti da elaborare.
- S11: Stato nel quale viene scritto in memoria la seconda parola uscita dalla codifica.
- S12: Stato nel quale vengono portati a ‘0’ l’o_en e l’o_we, in modo tale da non scrivere e leggere in memoria.
- S13: Stato nel quale viene salvato il numero di parole rimanenti e in un registro temporaneo l’indirizzo di uscita successivo.
- S14: stato nel quale controlla se il numero di parole rimaste da codificare è 0, in caso positivo si porta l’o_end a ‘1’, in caso negativo rimane a ‘0’.
- S15: Stato nel quale si controlla l’o_end, se è ‘0’ il modulo salva gli ultimi 2 bit della parola attuale in un registro temporaneo e porta a ‘1’ il selettore del ritardo, riprendendo la codifica da S3, se è ‘1’ il modulo imposterà alto il segnale o_done indicando la terminazione.

- ## 2.4 Descrizione segnali utilizzati

o_end	Segnale che indica la terminazione della codifica
rest_sel	Segnale che permette di selezionare quale tipo di valore e da applicare a b_{-1} e b_{-2}, se è la prima parola si usa 0, dalla seconda in poi si usa old_rest
i_data_addr, i_data_addr_next	Sono registri temporanei che contengono l'indirizzo della parola successiva da codificare
o_reg1	Contiene la parola da codificare
mux_reg1	Contiene il numero delle parole da codificare
sub	Contiene il numero di parole da codificare mancanti subito dopo le operazioni logiche
rit1	Contiene la parola da codificare ritardata di 1
rit2	Contiene la parola da codificare ritardata di 2
ris1	È un registro temporaneo che contiene il risultato dell'operazione o_reg1 XOR rit2
ris2	È un registro temporaneo che contiene il risultato dell'operazione: temp XOR rit2
temp	È un registro temporaneo che contiene il risultato dell'operazione: o_reg1 XOR rit1,

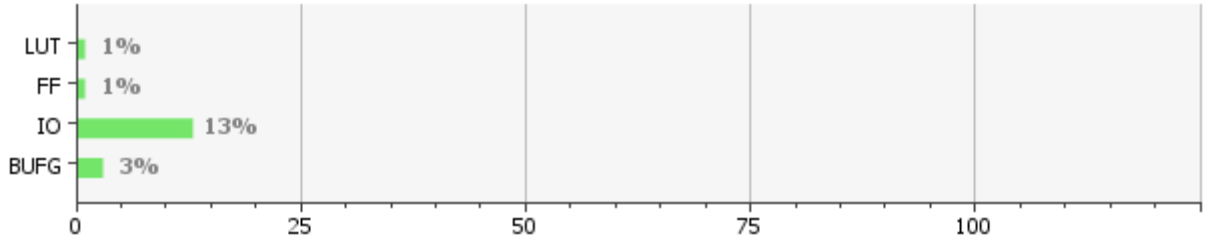
o_addr, o_addr_next	Registri temporanei che contengono il successivo indirizzo dell'output
rest	Segnale che contiene gli ultimi 2 bit della parola attuale
old_rest	Segnale che contiene gli ultimi 2 bit della parola precedente
curr_state, next_state	Segnali che permettono al modello di muoversi fra i vari stati

3 Sintesi

Per la sintesi del modulo si è utilizzato come software Xilinx Vivado 2016.4 e come FPGA si è utilizzata la xc7a200tfbg484-1.

Analizzando i report forniti da Vivado, in questo caso il “Report Utilization”, si può notare che le componenti utilizzate siano un numero molto inferiore rispetto alla disponibilità della FPGA

Resource	Estimation	Available	Utilization %
LUT	204	134600	0.15
FF	162	269200	0.06
IO	38	285	13.33
BUFG	1	32	3.13



Per quanto riguarda invece il “Timing Report”, analizzandolo si può ottenere il tempo effettivo di clock utilizzato per svolgere il lavoro. In questo caso con un periodo di clock nel test bench di 100 ns si è ottenuto un Worst Negative Slack pari a 96,849 ns, risultato che rientra nei limiti della consegna.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	96,849 ns	Worst Hold Slack (WHS):	0,134 ns	Worst Pulse Width Slack (WPWS):	49,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	314	Total Number of Endpoints:	314	Total Number of Endpoints:	163

All user specified timing constraints are met.

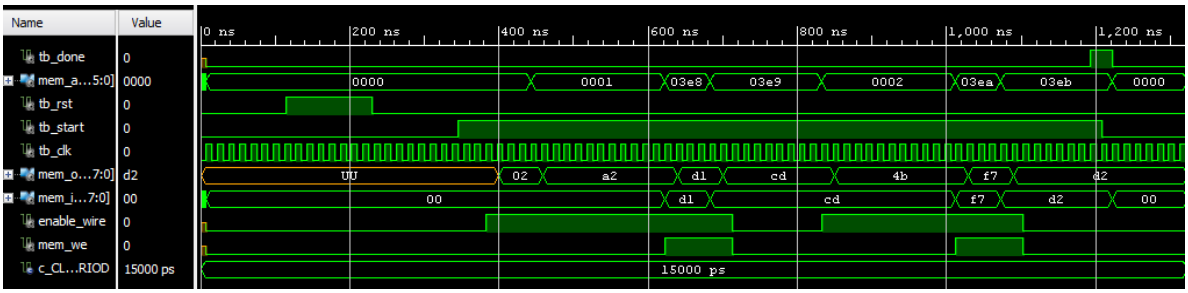
4 Simulazioni

Si è verificato il corretto funzionamento del componente utilizzando vari test bench, essi testano dei casi limite della nostra specifica, difatti si sono rilevati molto importanti nella fase di correzione e miglioramento del componente.

Per tutti i test presentati si è testato il modulo sia in pre-sintesi (behavioural simulation) che in post-sintesi (functional simulation), ottenendo sempre esiti positivi.

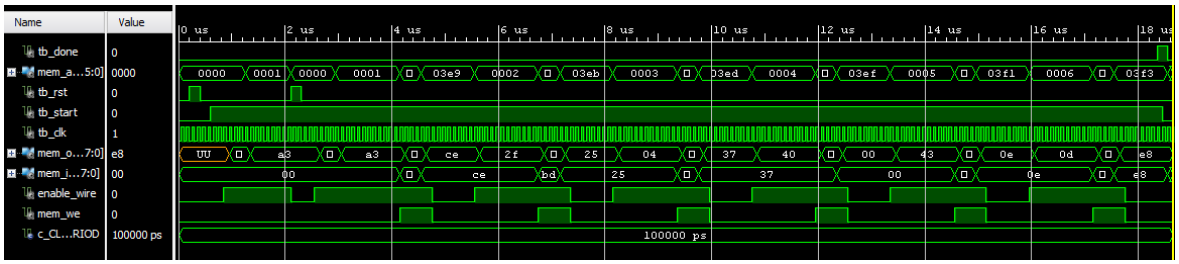
4.1 Risultati Test

- **Test Esempio_1:** questo test testa il funzionamento del programma su un input base di 2 parole



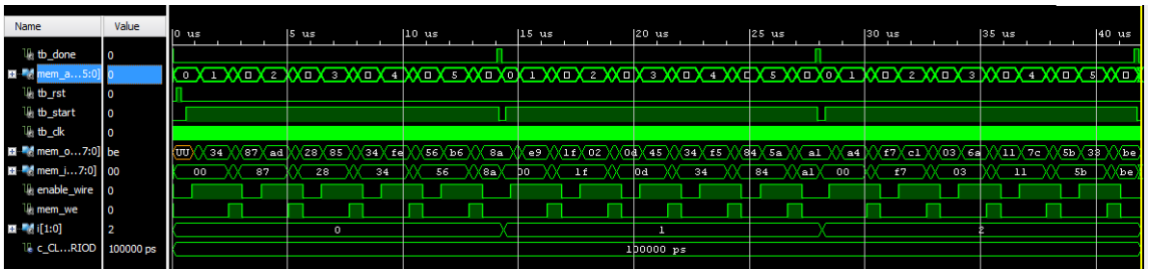
Tempo di esecuzione in Behavioural = 1322,500 ns
Tempo di esecuzione in Functional = 1322,600 ns

- **Test Reset:** questo test verifica la re-inizializzazione del componente tramite l'inserimento di un reset asincrono



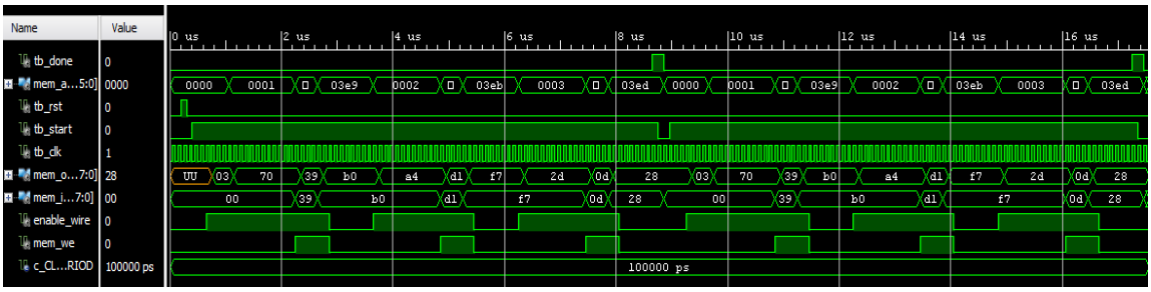
Tempo di esecuzione in Behavioural = 18650 ns
Tempo di esecuzione in Functional = 18650,100 ns

- **Test Re-Encode:** il test consiste nel codificare 3 serie di flussi diversi in successione



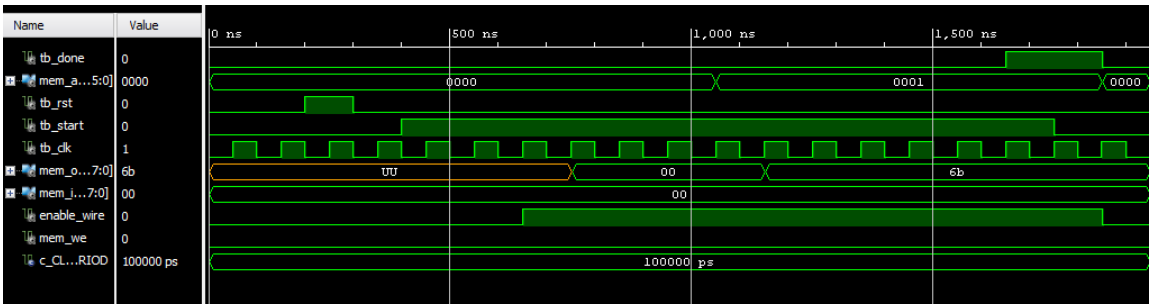
Tempo di esecuzione in Behavioural = 41950 ns
Tempo di esecuzione in Functional = 42150,100 ns

- **Test Doppio Uguale:** il test verifica che dato due volte lo stesso input, il risultato in output sia uguale



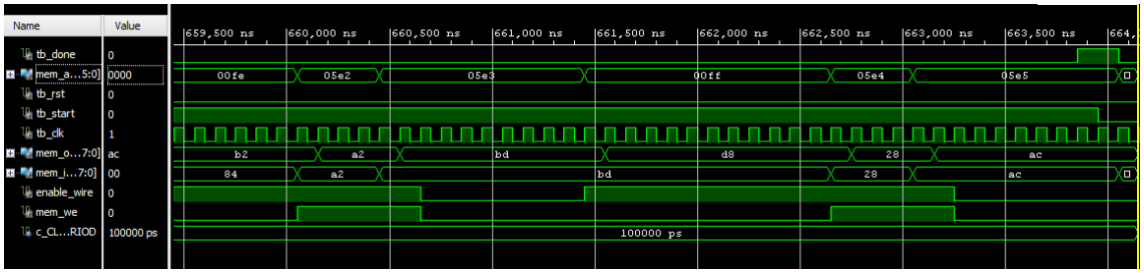
Tempo di esecuzione in Behavioural = 17450 ns
Tempo di esecuzione in Functional = 17550,100 ns

- **Test Sequenza Minima:** con questo test si studia il caso in cui ci siano 0 stringhe da codificare, si verifica di fatto che il componente non scriva in memoria e termini subito la codifica



Tempo di esecuzione in Behavioural = 1950 ns
Tempo di esecuzione in Functional = 1950,100 ns

- **Test Sequenza Massima:** con questo test si intende studiare l'inserimento del massimo numero di parole codificabili (255), il test vuole verificare la correttezza del componente sottoponendolo ad un numero elevato di parole da codificare



Tempo di esecuzione in Behavioural = 664150 ns
Tempo di esecuzione in Functional = 664150,100 ns

5 Conclusione

Per il raggiungimento della soluzione si è optato sin dal primo momento per l'utilizzo di una macchina a stati finiti, descrivendo il componente in modo tale da ottenere un singolo processo e un modulo compatto e ordinato con i vari stati chiari e ben definiti. Quest'ultimo infatti utilizza solamente una piccola parte del FPGA con un ritardo dell'esecuzione massimo inferiore al clock richiesto.

Dopo aver sottoposto il componente ai vari test, e corretto tramite essi le varie imprecisioni, sono arrivato ad un modulo corretto con le varie caratteristiche e vincoli dati rispettati. Inoltre, il componente è privo di latch, di fatti risulta funzionante correttamente anche in post sintesi.