Prova Finale (Progetto di reti logiche)

Politecnico di Milano

Anno scolastico 2021-2022

Documentazione

Federica Di Filippo

Matricola: 934669

Codice persona: 10666561

POLITECNICO
MILANO 1863

1 SOMMARIO

2 INTRODUZIONE

2.1 SCOPO DEL PROGETTO

L'obiettivo del progetto consiste nell'implementare un modulo hardware in VHDL che, data in ingresso una sequenza continua di W parole da 8 bit, la elabori e produca una sequenza di 2*W parole come output. Il modulo da implementare deve interfacciarsi con una memoria sincrona, da cui leggerà la sequenza di ingresso e su cui verrà trascritto l'output.

2.2 SPECIFICHE GENERALI

Il modulo riceve in ingresso la quantità di sequenze da analizzare, seguita dal flusso continuo di parole. Da ogni bit in ingresso ne vengono codificati due (PK1 e PK2), secondo l'algoritmo di convoluzione rappresentato dal seguente diagramma degli stati:

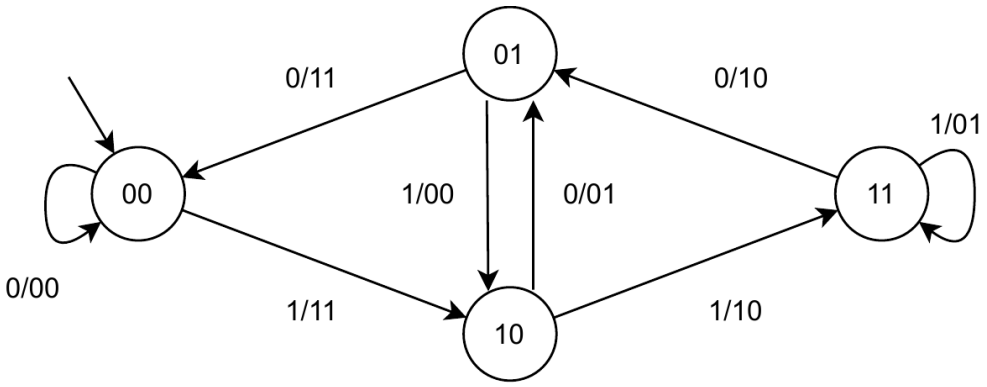

Figura 1: diagramma degli stati del convolutore

Il diagramma rappresenta una Macchina di Mealy quindi, conoscendo lo stato corrente e l'input, è possibile ricavare lo stato prossimo e la sequenza in uscita tramite le mappe di Karnot:

Tabella di verità:

Q0	Q1	U	Q0*	Q1*	PK1	PK2
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	1	1
0	1	1	1	0	0	0
1	0	0	0	1	0	1
1	0	1	1	1	1	0
1	1	0	0	1	1	0
1	1	1	1	1	0	1

Formule ricavate:

Q0* = U
Q1* = Q0
PK1 = U xor Q1
PK2 = Q0 xor (U xor Q1)

2.3 INTERFACCIA DEL COMPONENTE

Il componente da descrivere presenta la seguente interfaccia:

```
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;
```

Nello specifico:

- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- i_start è il segnale di START generato dal Test Bench;
- i_data è il segnale che arriva dalla memoria in seguito ad una richiesta di lettura - è un vettore da 8 bit che contiene la sequenza da elaborare;
- o_address è il segnale (vettore) di uscita che manda l'indirizzo alla memoria – viene utilizzato per leggere i dati dalla memoria e per scrivere l'output ;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- o_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_we è il segnale di WRITE ENABLE che viene inviato alla memoria: viene posto a '0' per la lettura, a '1' per la scrittura;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria

2.4 DESCRIZIONE DELLA MEMORIA

La memoria con cui si interfaccia il componente è sincrona, con indirizzamento al Byte. I dati, ciascuno di 8 bit, sono distribuiti nel modo seguente:

- l'indirizzo 0 è usato per memorizzare il numero di parole da codificare;
- gli indirizzi da 1 a 999 contengono le singole parole da 8 bit che verranno sottoposte al convolutore;
- dall'indirizzo 1000 in poi viene memorizzata la sequenza di output, ottenuta dall'elaborazione delle parole in input.

numero di parole	Indirizzo 0
primo Byte da codificare	Indirizzo 1
secondo Byte da codificare	Indirizzo 2
…	
primo Byte sequenza di output	Indirizzo 1000
secondo Byte sequenza di output	Indirizzo 1001
terzo Byte sequenza di output	Indirizzo 1002
quarto Byte sequenza di output	Indirizzo 1003
…	

3 ARCHITETTURA

In seguito all'inizializzazione delle variabili, il componente rimane in stato di attesa finché il segnale i_start non viene portato a '1'. Inizia quindi la computazione: ad ogni Byte in ingresso viene applicato l'algoritmo di avanzamento della macchina a stati per produrre le due sequenze di uscita (ciascuna da un Byte). Terminata la computazione, il segnale o_done viene portato a '1'. Di conseguenza il test bench porta i_start a '0' ed in seguito anche o_done viene azzerato, in attesa che i_start torni nuovamente a '1'. Il processo è rappresentato dalle seguenti fasi:

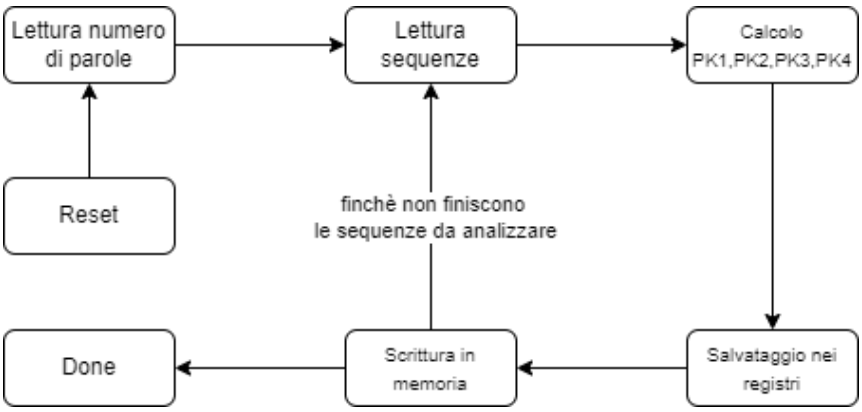

Figura 2: fasi del processo

3.1 AUTOMA A STATI FINITI

Ciascuna di queste fasi contiene diversi stati elencati di seguito:

- *Reset* – S0 : stato di attesa che i_start venga portato a ‘1’, si ritorna in questo stato ogni volta che i_reset viene alzato;

- *Lettura del numero di parole* – S1, S2: viene letto il numero di parole da elaborare:
  - Se il numero di parole è ‘0’, la computazione termina (lo stato successivo è S13);
  - Altrimenti lo stato successivo è S3;

- *Lettura sequenze* – S3, S4, S5, S6: sono gli stati in cui si ritorna ogni volta in cui è necessario leggere un nuovo bit della sequenza o una nuova sequenza (comprendono l’incremento dell’indirizzo di lettura quando necessario);

- *Calcolo valori* – S7: in questo stato a partire dal valore letto dalla sequenza vengono calcolati i bit di output PK1 e PK2 ed è stabilito il nuovo stato corrente (Q0,Q1);

- *Salvataggio nei registri* – S8, S9, S10, S11: in questi stati i valori precedentemente trovati vengono salvati in quattro registri (R0,R1,R2,R3) ciascuno da due bit;

- *Scrittura in memoria* – S12, S13: in questi stati viene aggiornato l’indirizzo di scrittura, il segnale o_we è posto a ‘1’ ed infine viene scritta in memoria la concatenazione dei 4 registri R0, R1, R2 e R3. Successivamente:
  - se la parola non è finita, si ritorna allo stato S3, per la lettura del bit successivo;
  - se la parola corrente è stata analizzata, ma le parole non sono ancora finite si ritorna allo stato S2, per la lettura della parola successiva;
  - altrimenti la computazione è terminata ed è possibile passare allo stato di done;

- *Done* - S13: in questo stato viene portato a ‘1’ il segnale o_done, in attesa che i_start venga posto nuovamente a ‘0'.

Di seguito è mostrata la rappresentazione completa della macchina a stati (Fig. 3) .

3.2 SCELTE PROGETTUALI

L’architettura del componente realizzato si compone di due moduli: una macchina a stati ed un datapath. La macchina a stati gestisce il flusso dell’elaborazione e il settaggio dei segnali fondamentali per il datapath, mentre il datapath contiene i processi di computazione dei dati e di interfacciamento con la memoria.

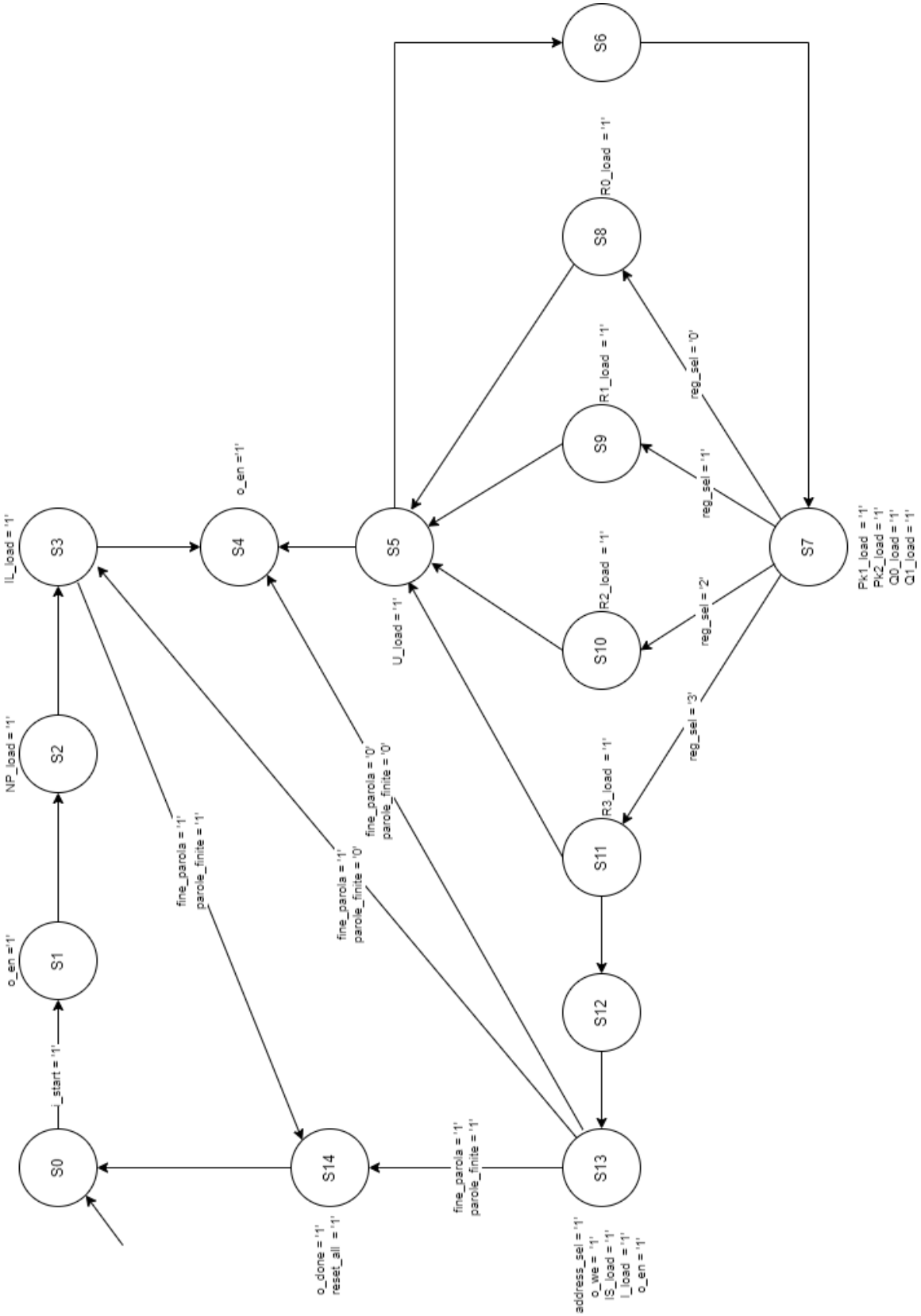

Figura 3: Macchina a stati

3.3 REGISTRI UTILIZZATI

L'algoritmo implementato utilizza diversi registri:

- Numero_Parole - std_logic_vector (7 downto 0): numero di parole da computare;
- U - std_logic : usato per prelevare dalla memoria ogni bit della sequenza da elaborare;
- Q0, Q1 - std_logic: contengono lo stato corrente della macchina a stati per il calcolo di Pk1,Pk2;
- Q0_temp, Q1_temp - std_logic: segnali temporanei per il calcolo di Q0,Q1;
- PK1,PK2 - std_logic: valori di uscita della macchina a stati;
- R0, R1, R2, R3 : std_logic_vector (1 downto 0): registri in cui viene salvato temporaneamente l'output, prima di essere scritto in memoria;
- Indice_Lettura - std_logic_vector (15 downto 0): indice per gli indirizzi di lettura, parte da 0 e viene incrementato per la lettura delle nuove parole;
- Indice_Scrittura - std_logic_vector (15 downto 0): indice per gli indirizzi di scrittura, parte da 1000 e viene incrementato prima di ogni scrittura in memoria;
- I - std_logic_vector(2 downto 0): contatore per i bit della sequenza da analizzare.

3.4 DATAPATH

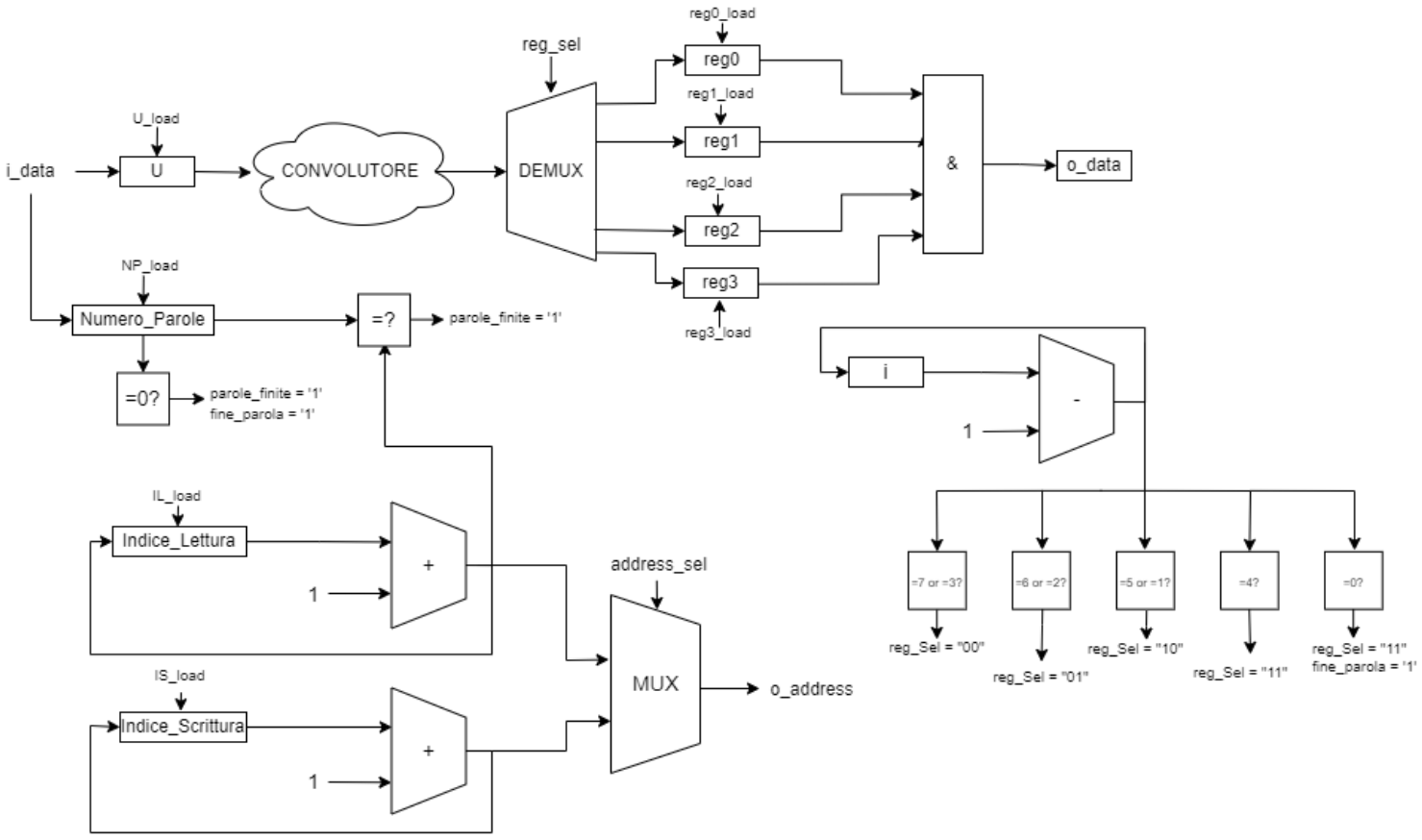

Figura 4: Datapath

4 RISULTATI SPERIMENTALI

4.1 SINTESI

4.1.1 Utilization report

Il componente è correttamente sintetizzabile ed implementabile e presenta 93 LUT, 70 Flip Flop e 0 Latch, come mostrato nell'immagine seguente.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	92	0	41000	0.22
LUT as Logic	92	0	41000	0.22
LUT as Memory	0	0	13400	0.00
Slice Registers	70	0	82000	0.09
Register as Flip Flop	70	0	82000	0.09
Register as Latch	0	0	82000	0.00
F7 Muxes	1	0	20500	<0.01
F8 Muxes	0	0	10250	0.00

Figura 5: Utilization report

4.1.2 Timing report

Il progetto presenta uno slack di 97.113 ns rispetto al periodo di clock richiesto di massimo 100 ns.

```
  (clock clock rise edge)    100.000   100.000 r
                               0.000   100.000 r  i_clk (IN)
  net (fo=0)                   0.000   100.000    i_clk
  IBUF (Prop_ibuf_I_O)         0.754   100.754 r  i_clk_IBUF_inst/O
  net (fo=1, unplaced)         0.554   101.308    i_clk_IBUF
  BUFG (Prop_bufg_I_O)         0.113   101.421 r  i_clk_IBUF_BUFG_inst/O
  net (fo=70, unplaced)        0.439   101.860    i_clk_IBUF_BUFG
  FDCE                                         r  FSM_sequential_cur_state_reg[0]/C
  clock pessimism              0.112   101.972
  clock uncertainty           -0.035   101.937
  FDCE (Setup_fdce_C_D)        0.037   101.974    FSM_sequential_cur_state_reg[0]
-------------------------------------------------
  required time                        101.974
  arrival time                          -4.861
-------------------------------------------------
  slack                                 97.113
```

Figura 6: Timing report

4.2 SIMULAZIONI

Il progetto è stato testato su numerose simulazioni e tutte hanno avuto successo, sia in Behavioral che in Post-Synthesis Functional, rispettando i vincoli imposti sul periodo di clock. Di seguito sono riportati gli esiti delle Behavioral Simulations di alcuni casi limite.

4.2.1 Reset

In questo testbench presenta un reset asincrono prima della fine dell'elaborazione.

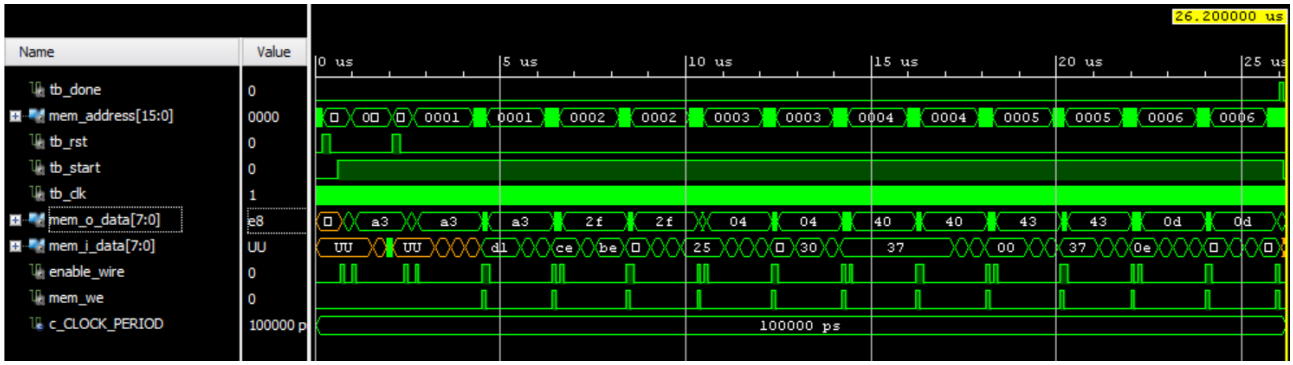

Figura 7: Test Reset

4.2.1 Sequenza min

Questo testbench presenta un numero di parole uguale a '0', ma sono comunque salvati in memoria dei valori, che però non devono essere analizzati.

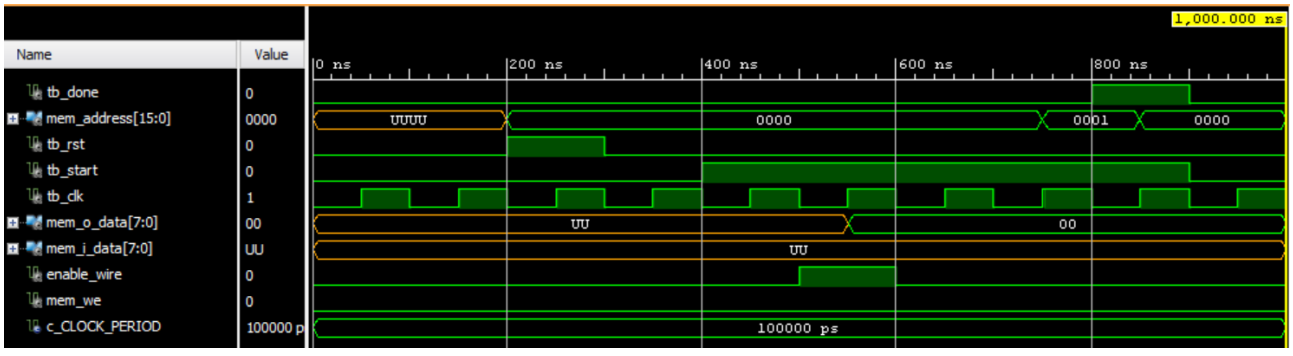

Figura 8: Test Sequenza minima

5 CONCLUSIONI

Il componente realizzato rispetta le specifiche (come si può vedere dai report) ed ha superato tutti i test effettuati, sia quelli forniti, che altri generati casualmente. Di conseguenza esso risulta funzionante ed in grado di soddisfare tutte le richieste.

La prima versione del componente prevedeva l'implementazione di un solo processo, ma per problemi di sincronizzazione è stata abbandonata a favore di una che prevedesse due moduli separati.