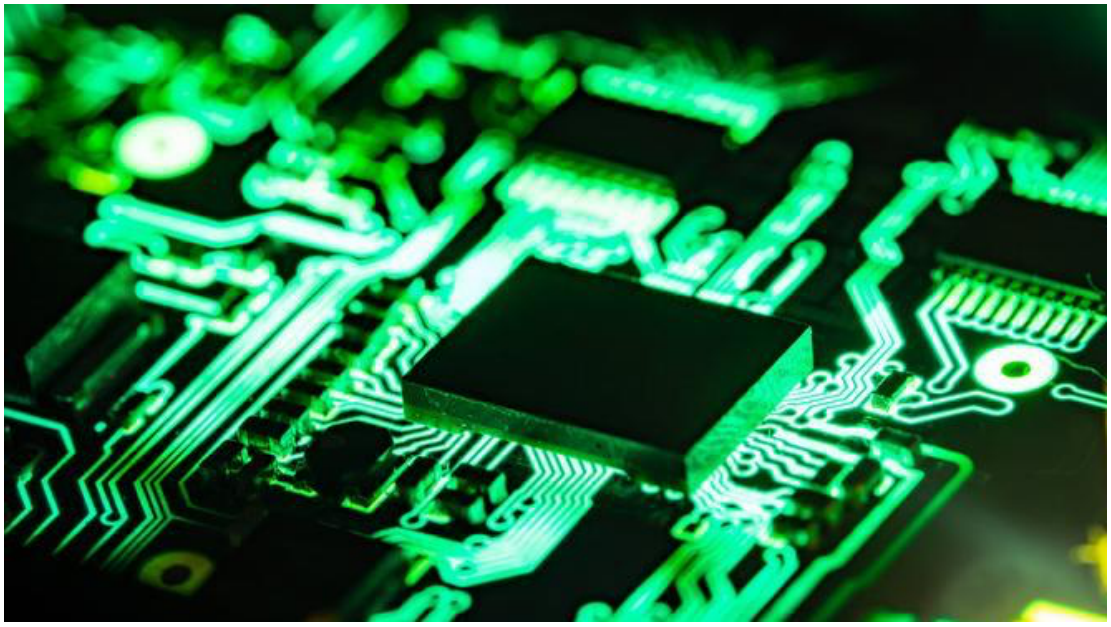
PROVA FINALE

(PROGETTO DI RETI LOGICHE)

PROF. FABIO SALICE – ANNO 2022/2023



Yang Hao Mao

(Codice Persona: 10705881)
(Matricola: 956410)

Indice

1. INTRODUZIONE

1.1 SCOPO DEL PROGETTO

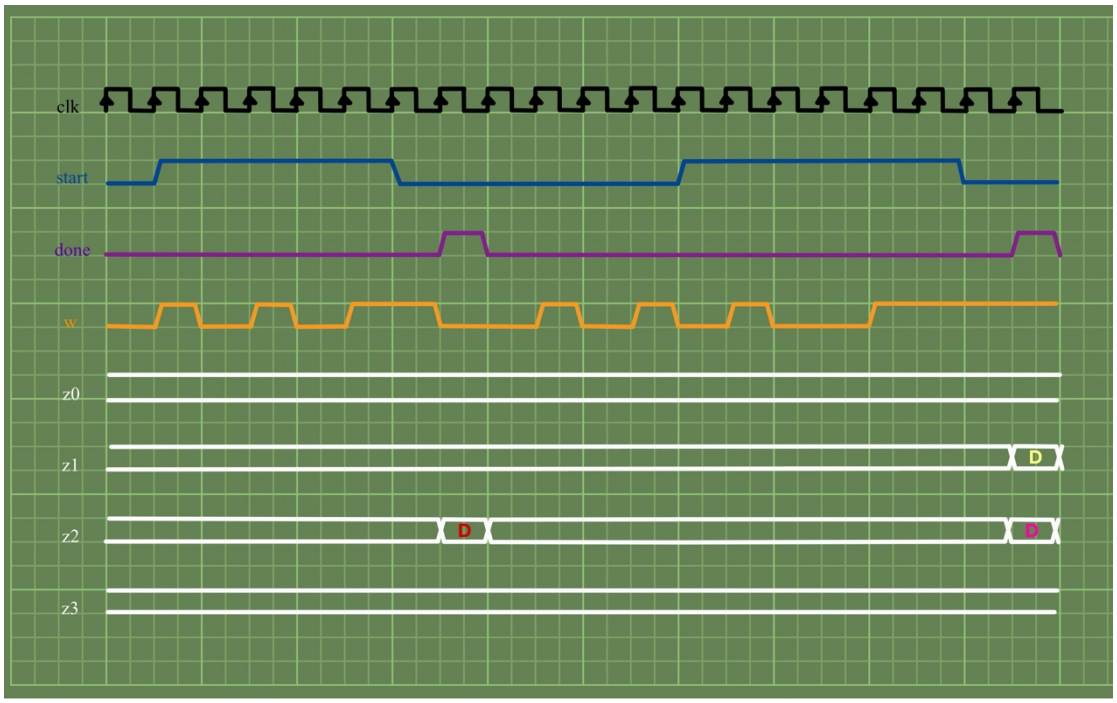
Lo scopo del progetto è di implementare un componente hardware descritto in VHDL, che il sistema riceve mediante un ingresso seriale da un bit, ed estrae il contenuto indirizzato da una locazione di memoria verso uno o più tra i quattro canali di uscita disponibili in modo parallelo.

1.2 SPECIFICHE GENERALI

Specificamente, all'istante iniziale, le uscite z0, z1, z2 e z3 sono 00, 00, 00 e 00 e DONE è 0; gli ingressi START e W vengono letti dal test bench con lunghezza dei bit complessivi compreso tra 2 e 18 bit, dove i primi 2 bit di intestazione identificano il canale di uscita (00 identifica z0, 01 identifica z1, 10 identifica z2, 11 identifica z3), e i restanti che indicano gli indirizzi di memoria di 16 bit (il numero di bit inferiore a 16 viene esteso con 0 sui bit più significativi). Tutti i bit su W validi devono essere letti sul fronte di salita del START e termina quando è basso. Le uscite z0, z1, z2 e z3 tutti sono 00 quando il segnale DONE è 0, e quando quest'ultimo è 1 (rimane per un solo ciclo di clock), il valore associato al rispettivo canale è visibili, e gli altri canali mostreranno l'ultimo valore trasmesso.

Il segnale START è garantito rimanere a 0 fino a che il segnale DONE non è tornato a 0 e il tempo massimo per produrre il risultato è inferiore a 20 cicli di clock. Ogni qual volta viene dato il segnale di RESET (uguale a 1), il modulo viene re-inizializzato.

Esempio:



I primi W utile è 10101, tra cui 10 indica il canale d'uscita z2, 101 (esteso in 16 bit: 0000 0000 0000 0101) indica l'indirizzo di memoria. Seguito dal primo DONE=1, il canale z2 scrive il DATO.

Si riscatta il STRAT=1, con W utile 010011, tra cui 01 indica il canale z1 e 0011 (esteso in 16 bit: 0000 0000 0000 0011) il relativo indirizzo.

Concluso con l'ultimo DONE=1, i dati vengono scritti nel canale z1 e z2 contemporaneamente.

1.3 INTERFACCIA DEL COMPONENTE

Il componente da descrivere deve avere 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_w     : in std_logic;
          o_z0    : out std_logic_vector(7 downto 0);
          o_z1    : out std_logic_vector(7 downto 0);
          o_z2    : out std_logic_vector(7 downto 0);
          o_z3    : out std_logic_vector(7 downto 0);
          o_done  : out std_logic;
          o_mem_addr : out std_logic_vector(15 downto 0);
          i_mem_data : in std_logic_vector(7 downto 0);
          o_mem_we   : out std_logic;
          o_mem_en   : out std_logic
      );
  end project_reti_logiche;
```

In particolare:

- `i_clk` è il segnale di CLOCK in ingresso generato dal Test Bench;
- `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_w` è il segnale W precedentemente descritto e generato dal Test Bench;
- `o_z0`, `o_z1`, `o_z2`, `o_z3` sono i quattro canali di uscita;
- `o_done` è il segnale di uscita che comunica la fine dell'elaborazione;
- `o_mem_addr` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `i_mem_data` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;

- `o_mem_en` è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- `o_mem_we` è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0.

1.4 DATI E DESCRIZIONE DELLA MEMORIA

I dati sono di dimensione 8 bit, memorizzati in una memoria data.

2. DESIGN

2.1 STATI DELLA MACCHINA

- **s. IDLE**

Stato iniziale in cui si attende la risposta della memoria in seguito alla richiesta del primo dato i_w "utile".

- **s. GET_CAN**

Stato in cui si richiede il secondo i_w "utile" per il completamento dell'acquisizione del canale d'uscita.

- **s. GET_ADDR**

Stato in cui viene richiesto il bit dell'indirizzo di memoria. Il tutto viene fatto al massimo 16 volte, poiché l'indirizzo di memoria è composto da 16 bit; e al minimo 1 volta ovvero 0 bit dell'indirizzo di memoria, e quindi richiede l'accesso alla memoria.

- **s. ASK_MEM**

Stato in cui viene richiesto e inseguito la scrittura del dato sul (sui) registro di canale (canali) di uscita dedicato (dedicati) dopo lo scatto di o_mem_en.

- **s. WRITE_OUT**

Stato in cui viene effettivamente stampato (stampati) il (i) dato (dati) sul (sui) apposito (appositi) canale (canali) e alza il segnale o_done.

- **s. DONE**

Stato in cui il componente completa il ciclo di lettura e scrittura per poi di tornare allo stato IDLE.

- Il componente è dotato di due processi:
  - **Il primo rappresenta la parte sequenziale e la gestione dei registri**

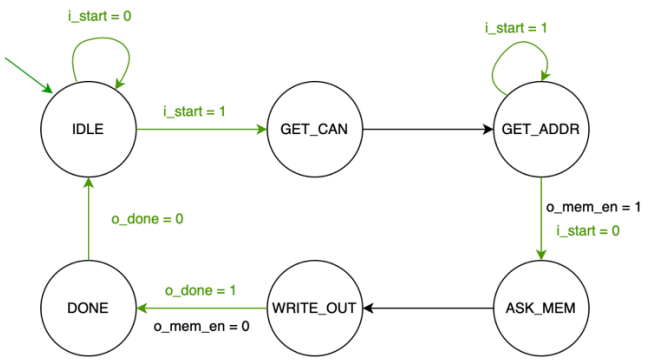
Condizione con cui il reset si alza => tutti i segnali ritornano allo stato di default Condizione con cui il clock si abbassa => tutti i segnali vengono aggiornati dai segnali precedentemente elaborati

  - **Il secondo rappresenta la FSM che in base allo stato corrente determina il prossimo stato in cui si troverà il componente**

Pre-eseguimento (prima che esegue la FSM): tutti i segnali di output assegna in valori di default, e i segnali di "x_next" ai segnali attuali. (dove x sono i vari segnali attuali) Esecuzione della FSM: rielabora i segnali in base allo stato corrente e determina lo stato prossimo.

- La FSM esegue sequenzialmente, dove i primi due stati corrisponde all'acquisizione del canale di uscita ovvero i primi due bit dell'intestazione riconosciuto dal segnale di i_start; e i successivi bit (da 0 a 16 bit) acquisiti singolarmente nello stato GET_ADDR; quindi uno stato che legge i dati dalla memoria che termina con il primo i_start = 0; dopo di che nello stato ASK_MEM appunto richiede i dati dalla memoria seguito da uno stato WRITE_OUT per il completamento della scrittura dei dati sugli appositi output dai registri attivati (dal canale di uscita attivati e precedentemente attivati: en_zN = true, dove N è il numero intero corrispondete al numero di canale: 0, 1, 2, 3); il ciclo si conclude con l'ultimo stato DONE che ritorna allo stato IDLE iniziale.

Nota: nello stato GET_ADDR ho utilizzato UNSIGNED_VECTOR da 16 bit per lo SHIFT ed AND per incrementare ovvero calcolo dell'indirizzo di memoria richiesto, e il CAST a quest'ultimo per la correttezza del STD_LOGIC_VECTOR.

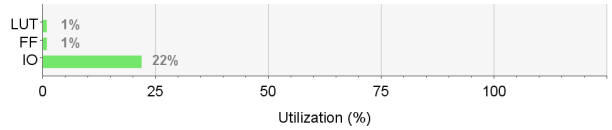


3.1 SINTESI

- **Utilization Report**:

  il dispositivo progettato è perfettamente sintetizzabile (*FPGA* usato: xc7a200tfbg484-1) senza *inferred latch*, 49 *Look Up Table* e 107 *Flip Flop.*

Resource	Utilization	Available	Utilization %
LUT	49	134600	0.04
FF	107	269200	0.04
IO	63	285	22.11



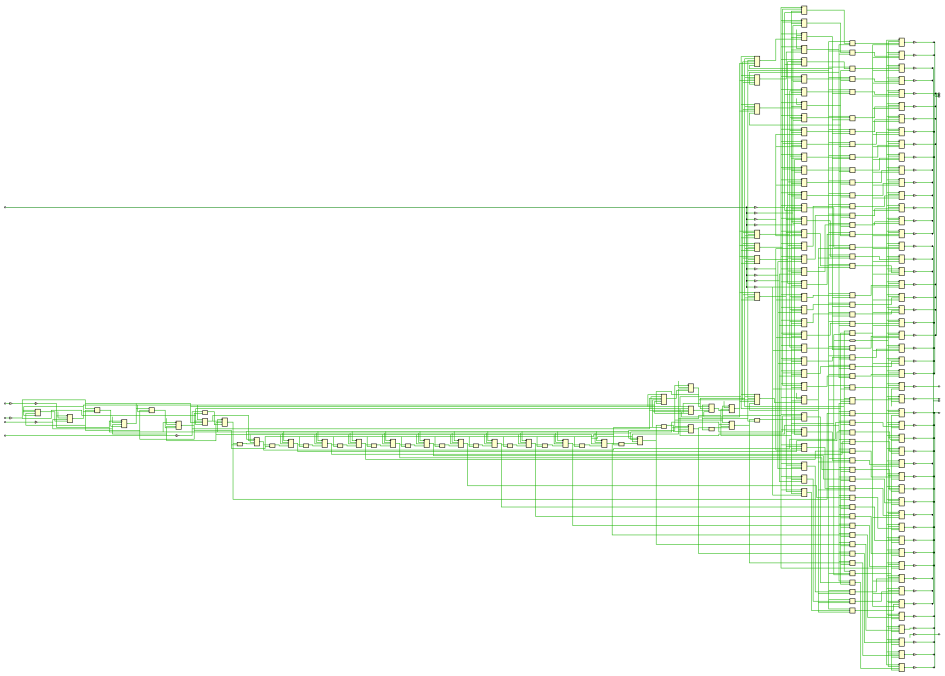
- **Timing Report**:

il tempo esecuzione Worst Negative Slack ci dice che abbiamo ancora 96,989ns tempo al comportamento del ciclo di clock nel peggiore dei paths, con un ciclo di clock di 100 ns abbiamo i seguenti risultati:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	96.989 ns	Worst Hold Slack (WHS):	0.149 ns	Worst Pulse Width Slack (WPWS):	4.500 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:	171	Total Number of Endpoints:	171	Total Number of Endpoints:	108

All user specified timing constraints are met.

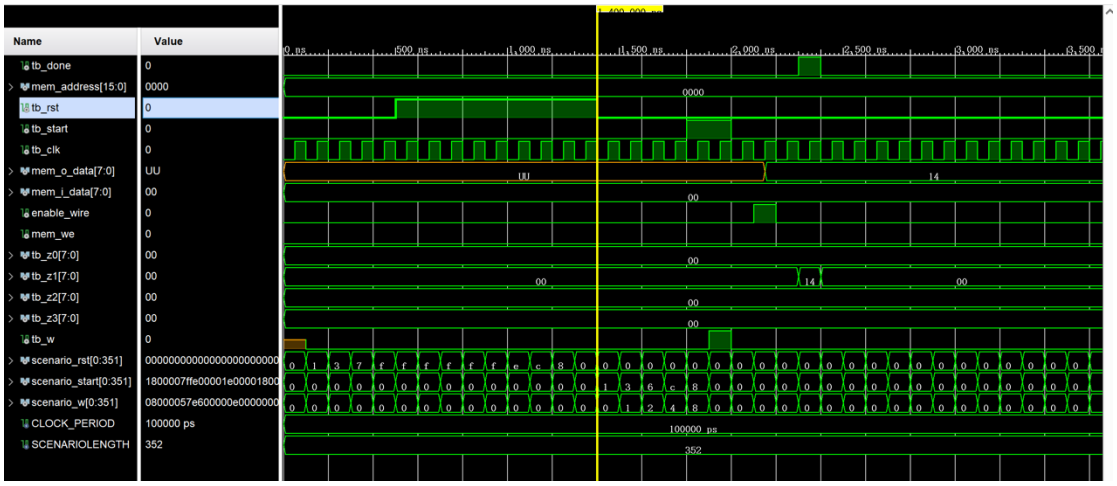
- **Schema del componente**:



Tutti i *Test Bench* forniti sono andati a buon fine superando la *Behavioral Simulation* e la *Post-Synthesis Functional Simulation*.

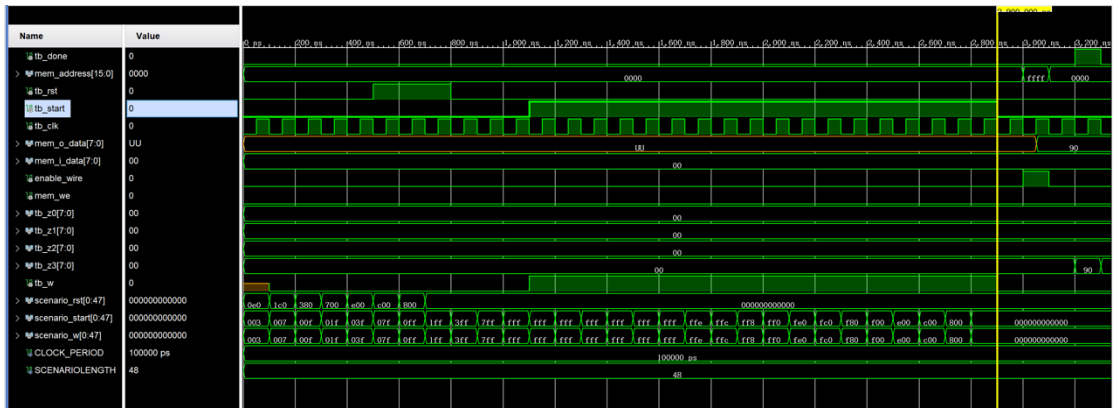
- **Reset con durata più di 1 clock (Test_Bench_4)**:

  il test verifica il comportamento del componente quando scatta un reset di lungo periodo durante l' abbassamento di o_done e l'alzamento del i_start successivo:



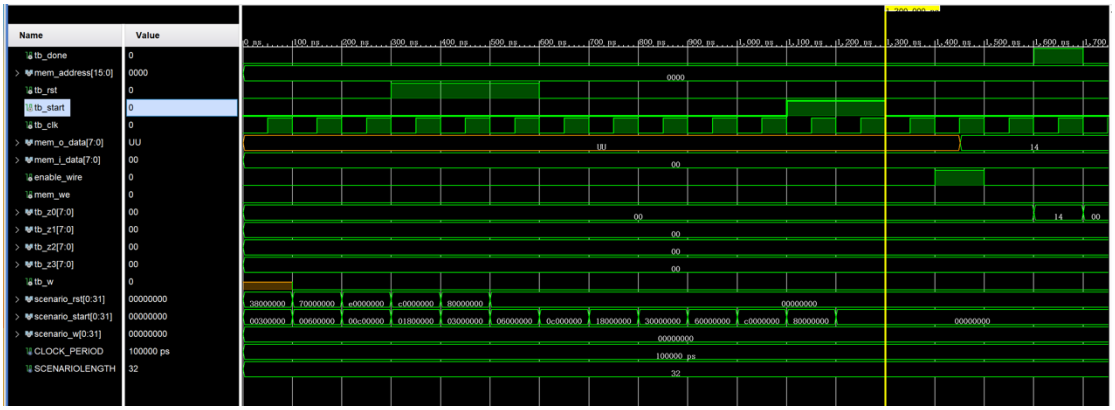
- **Max read (Test_Bench_6)**:

  il test verifica la correttezza della lettura di tutti i 18 bit utili (i_start = 1) con lo stesso valore di i_w:



- **Min read (Test_Bench_7)**:

il test verifica la correttezza della lettura di tutti e solo 2 bit utili(i_start = 1) con lo stesso valore di i_w:

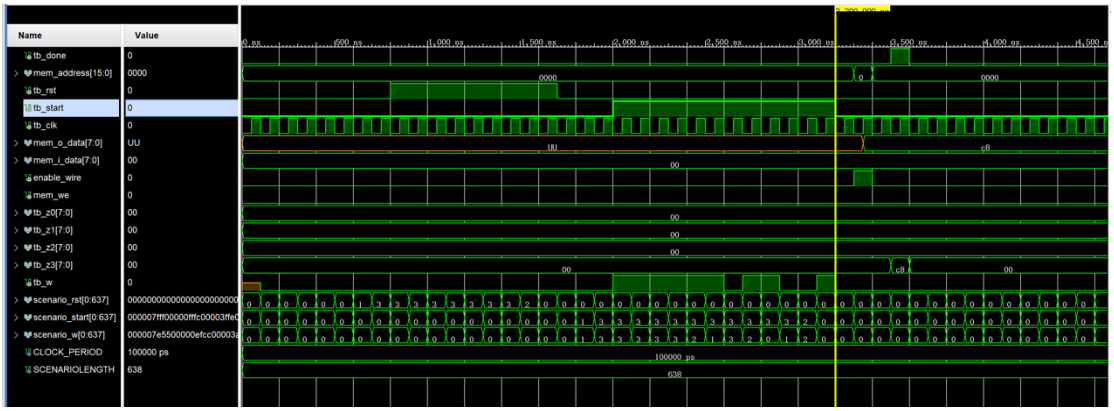


Il test simula la lettura solo dei bit di canale, con default di indirizzo uguale a 0.

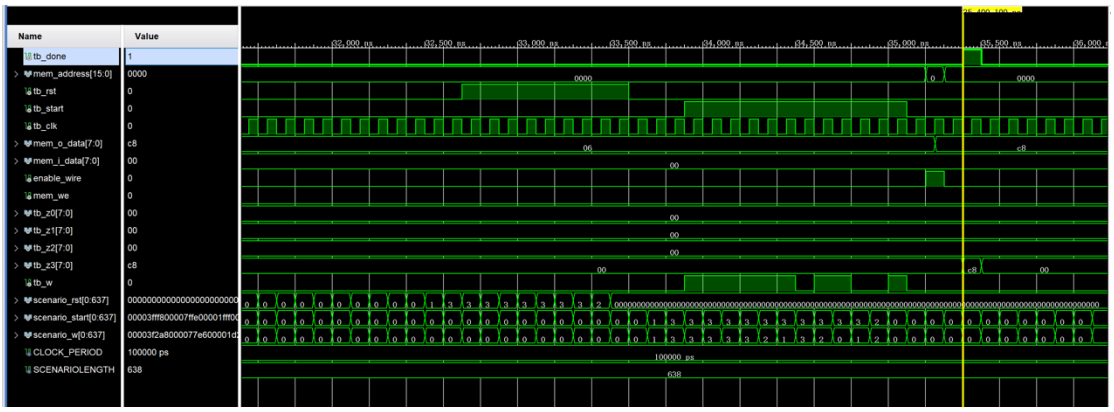
- **Ri-lettura dei dati (Test_Bench_5)**:

il test di verifica esegue nuovamente la lettura degli stessi dati letti all'istante del reset successivo:

(istante iniziale del test bench 5)



(istante 32600 ns del test bench 5)



4. CONCLUSIONI

La realizzazione finale del progetto soddisfa gli obiettivi che è stato posto, il progetto inizialmente è stato difficile a capire del codice VHDL e l'implementazione del modulo, sono riuscito a capire il lavoro e procedere l'implementazione del codice a partire dalla realizzazione dell'automa a stati finiti, la scrittura del codice, la fase di debug e la correzione degli errori trovati. Dopo tutto ciò sono riuscito a far funzionare il modulo correttamente mi ha permesso di ottenere subito un componente funzionante in post-sintesi

È stato utile e interessante a svolgere questo progetto che mi ha fatto capire l'implementazione delle reti sequenziali in modo dettagliato .