PROVA FINALE DI RETI LOGICHE

Andrea Bricchi 10660408
Eliahu Cohen 10704321

1 Aprile 2023

1. INTRODUZIONE

1.1 Descrizione del Progetto

L'obiettivo del progetto è di scrivere un programma in linguaggio VHDL che prende in ingresso prima 2 bit di intestazione, i quali identificano il canale d'uscita (*Z0,Z1,Z2,Z3*) del componente, e successivamente N bit di indirizzo della memoria che contengono il messaggio, quest'ultimo di 8 bit, che verrà poi stampato sul canale d'uscita definito dai 2 bit di intestazione. L'indirizzo di memoria può variare da 0 fino a 16 bit e, in caso abbia lunghezza inferiore a 16, questo viene esteso con '0' sui bit più significativi. Per fare un esempio:
N = 16 : 1010111000110011 -> 1010111000110011
N = 9 : 111000011 -> 0000000111000011

1.2 Interfaccia del Componente

Il componente da descrivere ha 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_address       : 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.

2. ARCHITETTURA

2.1 Modello della FSM

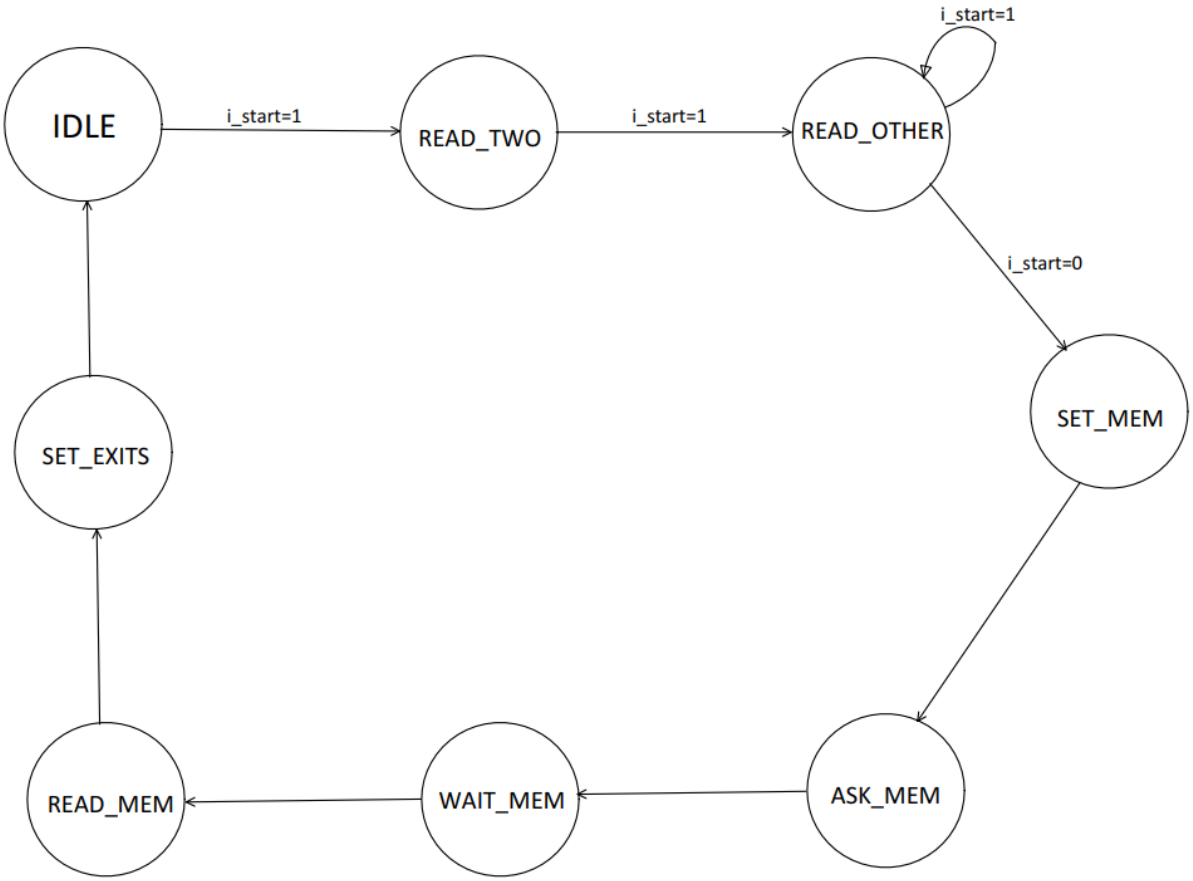

Fig. 1: Modello della FSM.

La FSM è composta da un ciclo unico che include la fase di lettura iniziale e la fase di scrittura finale, infine essa torna allo stato inziale per prepararsi così ad un'eventuale nuova lettura.

2.2 Descrizione degli Stati

La macchina a stati è composta da 8 stati *(IDLE, READ_TWO, READ_OTHER, SET_MEM, ASK_MEM, WAIT_MEM, READ_MEM e SET_EXITS*), nel dettaglio:

- *IDLE*: stato iniziale in cui vengono inizializzate le variabili, nel momento in cui *i_start* = 1 viene salvato il primo di intestazione nel segnale *uscita*.
- *READ_TWO*: stato in cui viene letto e salvato il secondo bit di intestazione nel segnale *uscita*.
- *READ_OTHER*: stato in cui, mentre *i_start* = 1, vengono letti e successivamente salvati gli N bit dell'indirizzo di memoria nel segnale *indirizz*. Quando invece *i_start* = 0 la macchina passa allo stato successivo.
- *SET_MEM*: stato in cui viene letto e successivamente salvato l'indirizzo di memoria nella variabile *o_mem_addr* che verrà poi passato alla memoria.
- *ASK_MEM*: stato in cui la macchina attende il segnale *o_mem_en* = 1 per passare l'indirizzo di memoria alla memoria.
- WAIT_MEM: stato di attesa, della durata di un ciclo di clock, in cui l'FSM attende che la memoria riceva correttamente l'indirizzo di memoria.
- *READ_MEM*: stato in cui viene letto il dato dalla memoria e settato sull'uscita corretta.
- *SET_EXITS*: stato in vengono settati i dati salvati precedentemente sulle rispettive uscite e poi successivamente stampati a schermo.
- La macchina è pilotata dal segnale di clock *i_clk* e dal segnale di reset *i_rst* e ogni azione di questa viene eseguita ogni volta che si verifica un fronte di salita del segnale di clock. L'FSM inoltre utilizza il segnale *i_start* per determinare l'inizio della fase di lettura e il segnale *i_w* per leggere i dati ricevuti in input. Alla fine del processo di lettura, l'FSM imposta il segnale *o_done* = 1 per indicare la fase di scrittura, dalla durata di un ciclo di clock, sulle uscite dei rispettivi dati precedentemente salvati.

2.3 Scelte Progettuali

La macchina è stata creata implementando una'architettura di tipo behavioural a processo singolo. Ogni stato è determinato dalla variabile *curr_state* di tipo S e il cambiamento di questo avviene sempre in corrispondenza del fronte di salita del ciclo di clock e con segnale di reset *i_rst* = 0; se invece *i_rst* = 1 la FSM torna nello stato di *IDLE* e resetta tutti i segnali. Si è scelto di utilizzare dei segnali di supporto per alcune operazioni, nel dettaglio:

- *last_z0, last_z1, last_z2, last_z3*: segnali che contengono i dati salvati precedentemente sulle uscite del componente;
- *indirizz*: segnale su cui vengono salvati gli N bit passati in input in fase di lettura;
- *uscita* : segnale su cui vengono salvati i 2 bit di intestazione letti in input in fase di lettura;

2.4 Ottimizzazione

Il codice è stato implementato in modo da ottimizzare il tempo di esecuzione del programma usando il numero minore di stati possibile. Si è cercato inoltre di assegnare a ogni stato parti specifiche di entrambe le fasi di lettura e scrittura, aggiungengo anche uno stato ausiliario *WAIT_MEM* che permettesse una corretta esecuzione del processo senza perdere dati durante la sua esecuzione.

3. Risultati Sperimentali

3.1 Sintesi

Il componente è sintetizzabile ed è il seguente:

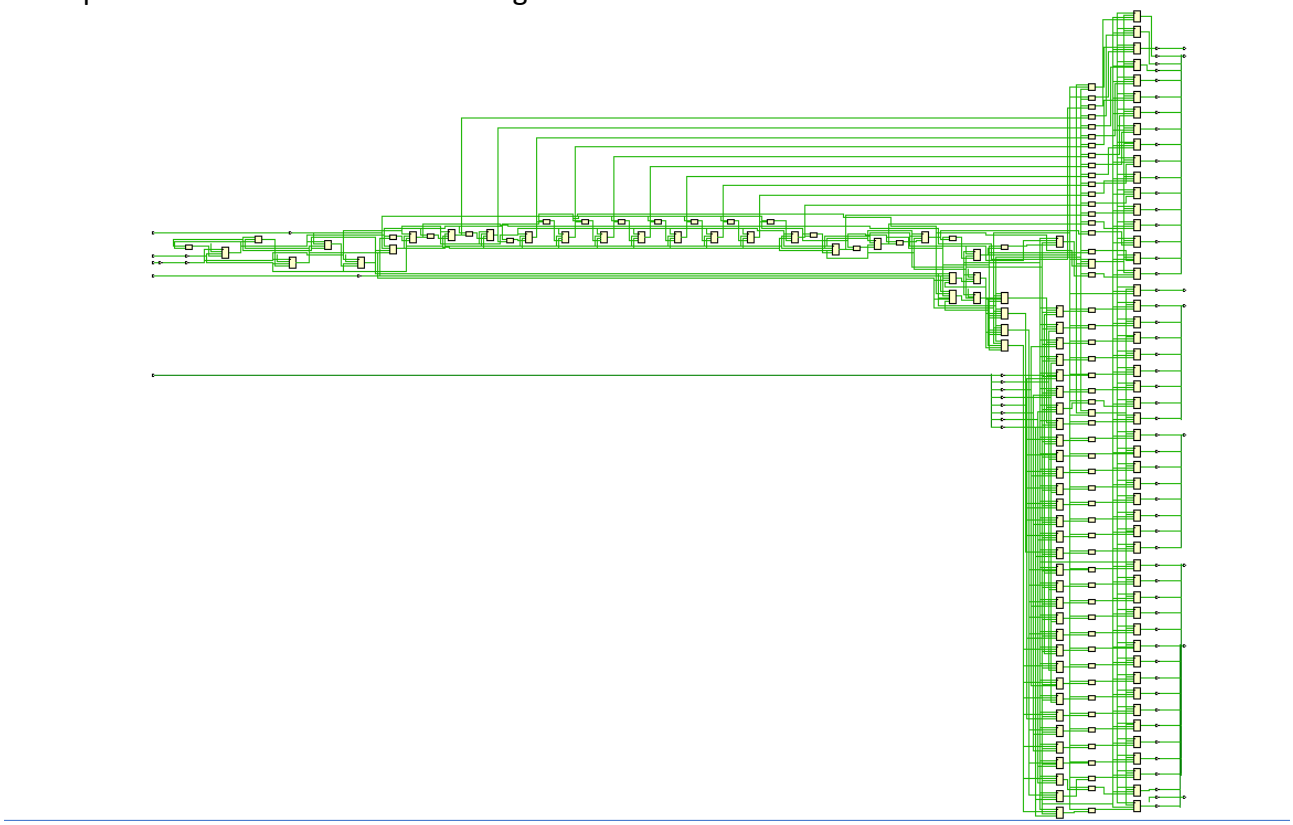

Fig. 2: Schema componente sintetizzato.

I risultati del report di utilizzo:

Resource	Utilization	Available	Utilization %
LUT	44	134600	0.03
FF	103	269200	0.04
IO	63	285	22.11

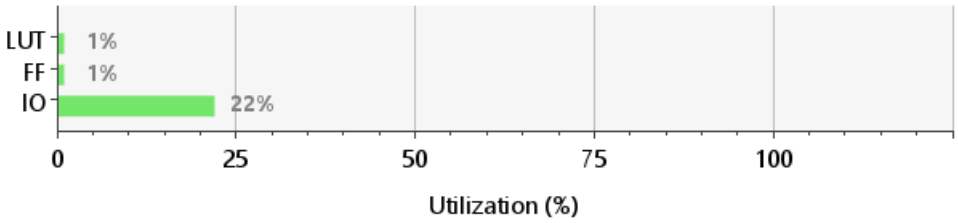

Fig. 3: Risultati del report.

3.2 Simulazioni

Per verificare il corretto funzionamento del componente sono stati effettuati vari casi di test, tutti terminati con esito positivo sia in pre-sintesi che in post-sintesi. Ecco un elenco dei test eseguiti:

- Test Completi
  In questi test viene verificato il corretto funzionamento del componente nel caso più generale. In fig. 4 e fig. 5 vengono illustrati due test creati da noi, mentre in fig. 6 viene illustrato il test reso a disposizione dal professore.

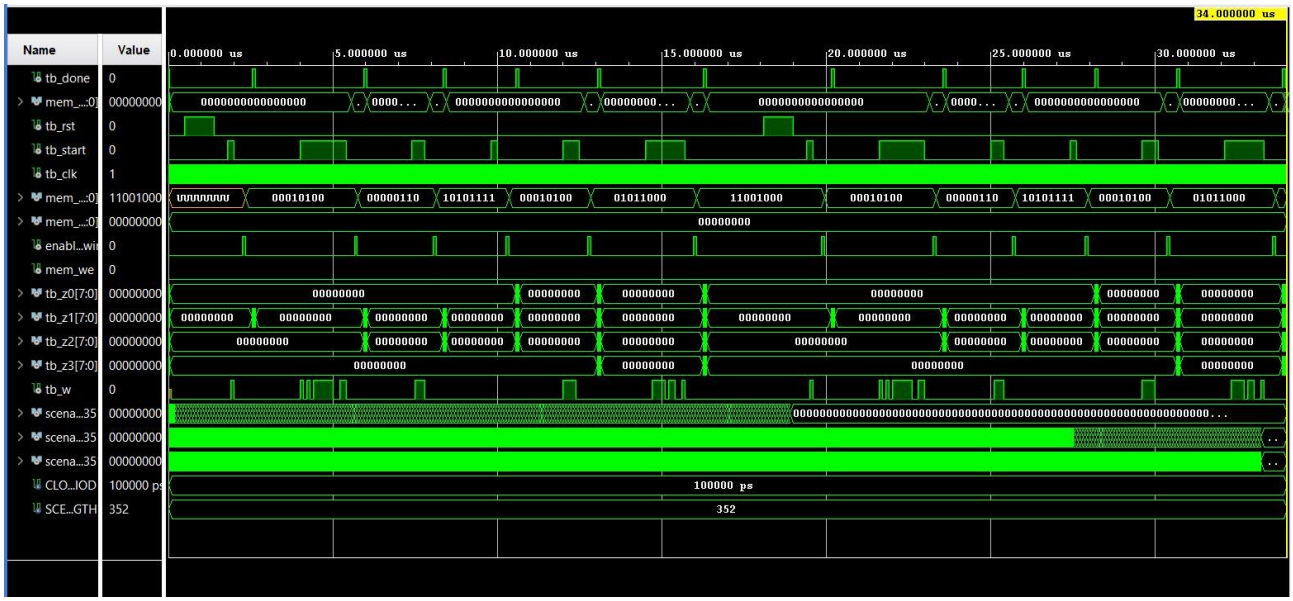

Fig. 4: Test completo 1.

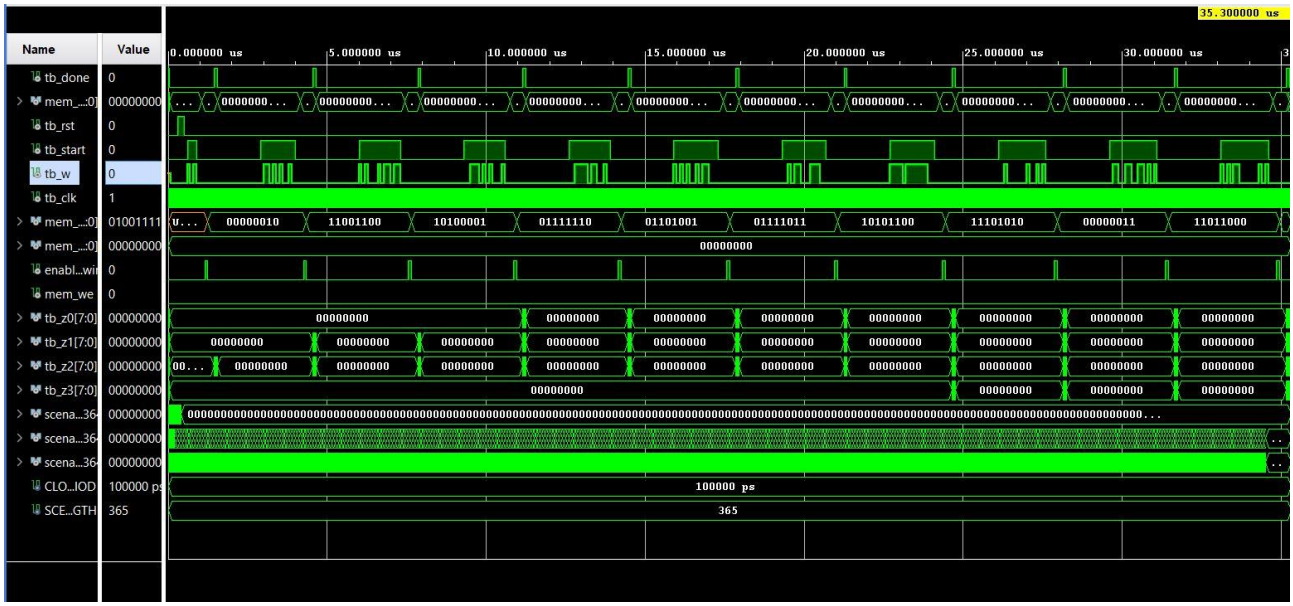

Fig. 5: Test completo 2.

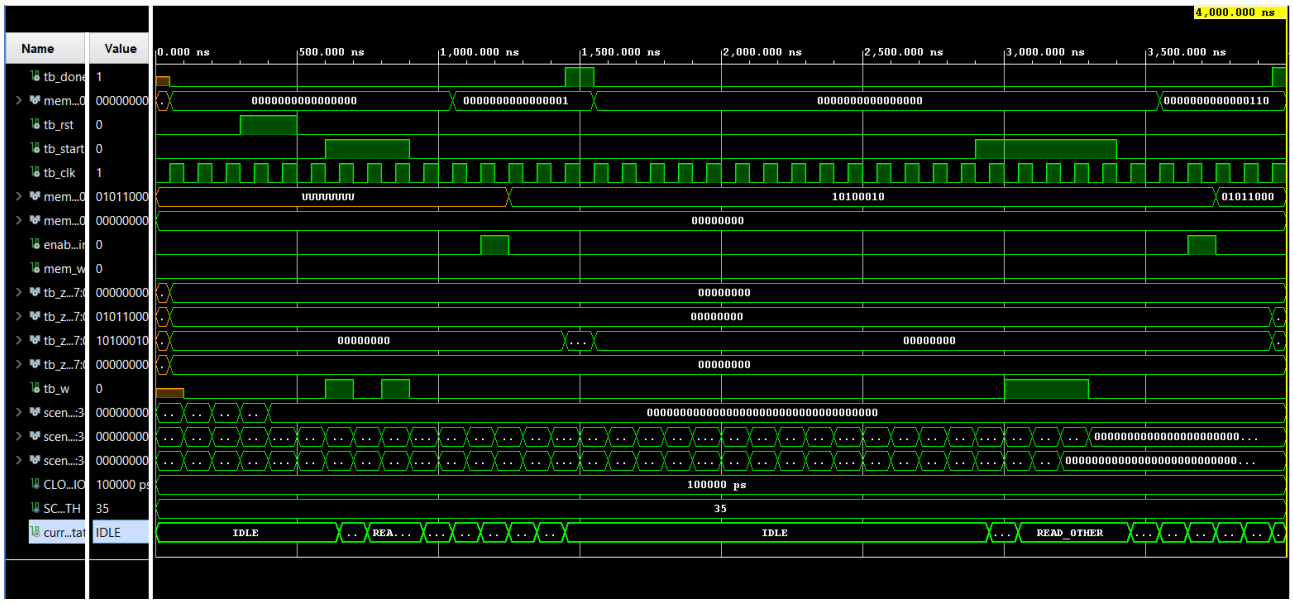

Fig. 6: Test fornito dal professore.

- Reset durante lo Start

  In questo test è stato attivato il segnale di reset mentre il segnale di start è alto.

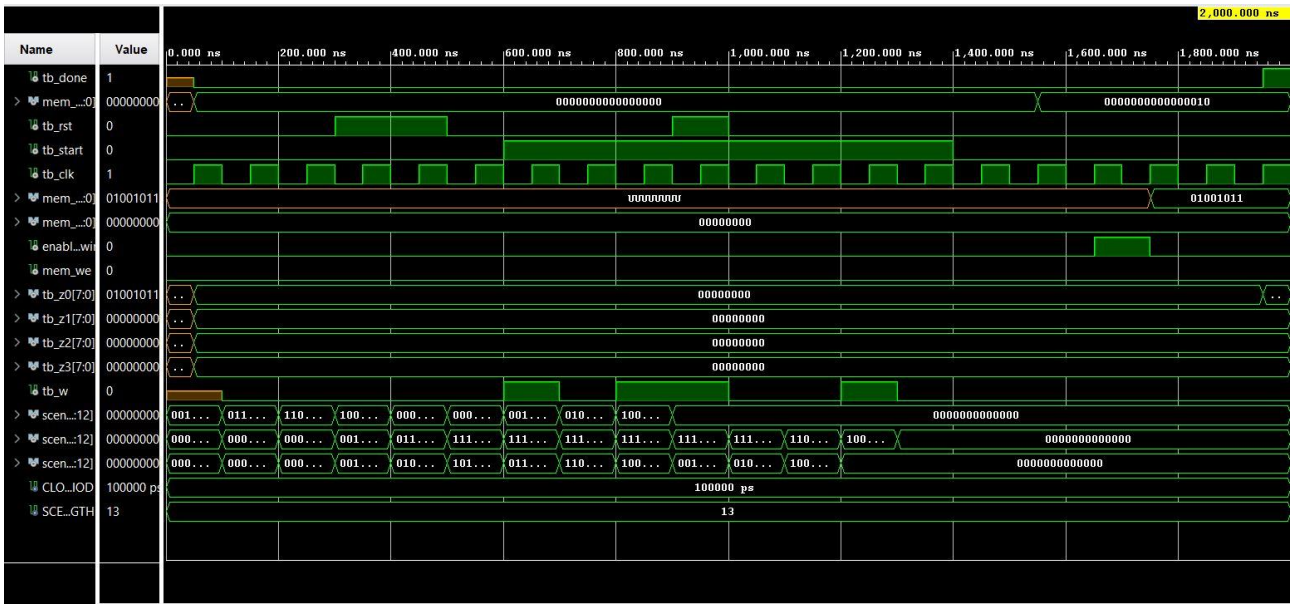

Fig. 7: Test Reset durante il segnale di Start.

- Reset nello stato *ASK MEM*

  In questo test è stato attivato il segnale di reset mentre la FSM è nello stato *ASK MEM*.

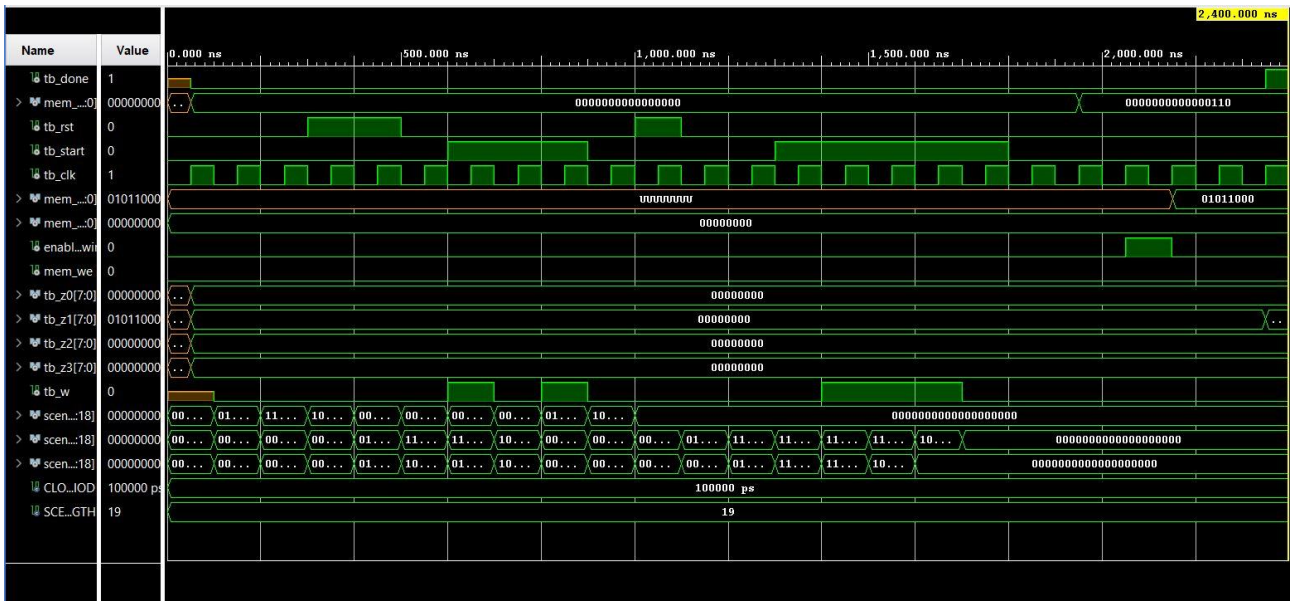

Fig. 8: Test Reset nello stato *ASK MEM*.

- Reset nello stato *READ MEM*

  In questo test è stato attivato il segnale di reset mentre la FSM è nello stato *READ MEM*.

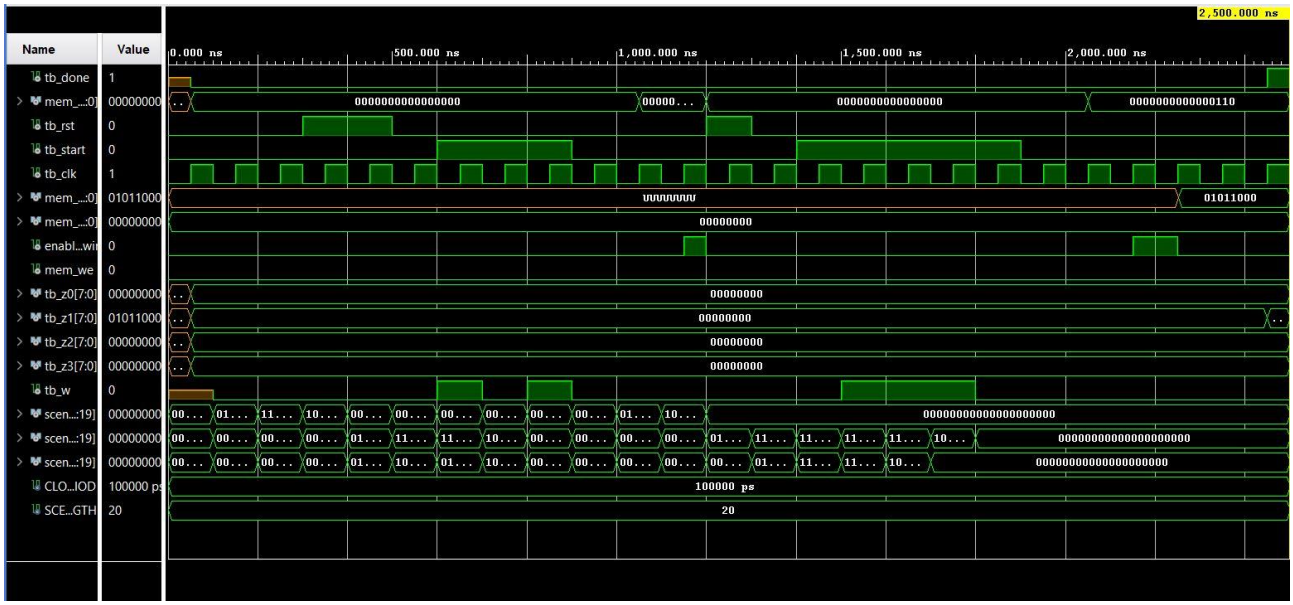

Fig. 9: Test Reset nello stato *READ MEM.*

- Reset dopo il segnale di Start

  In questo test è stato attivato il segnale di reset dopo che il segnale di Start è passato da alto a basso.

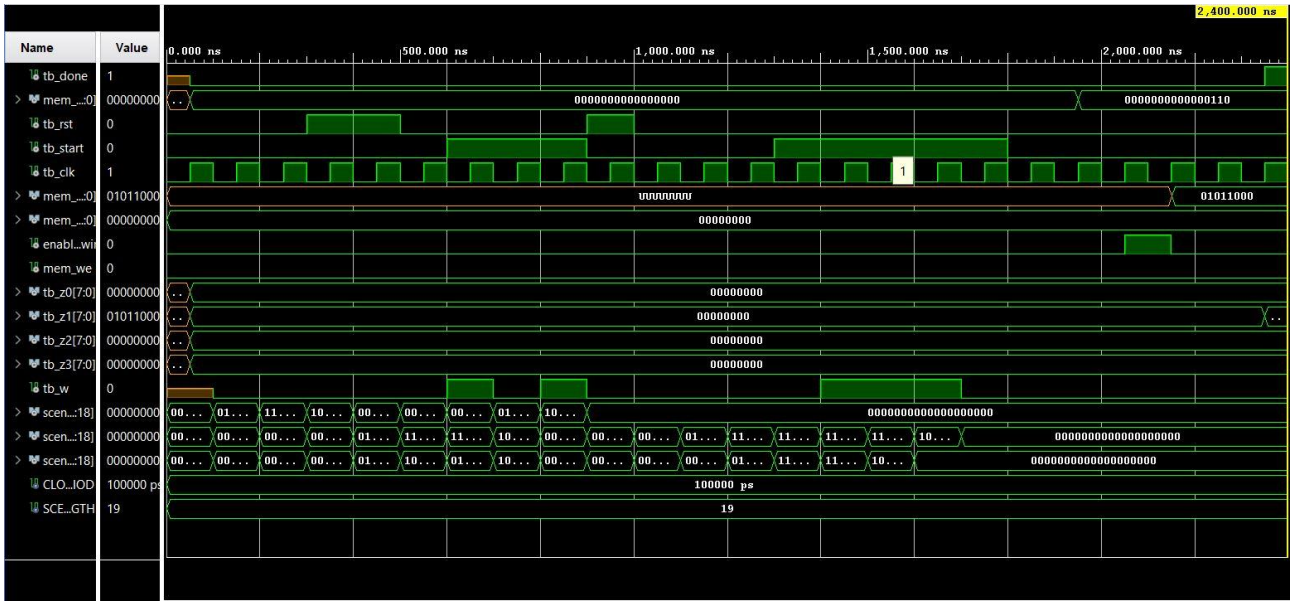

Fig. 10: Test Reset dopo il segnale di Start

4. CONCLUSIONI

In conclusione, si ritiene eseguita correttamente l'implementazione dell'algoritmo descritto nelle specifiche, in quanto il risultato è un componente sintetizzabile e simulabile in post-sintesi utilizzando 44 *Look Up Tables* (utilization del 0.03%) e 103 *Flip Flop* (utilization del 0.04%), tramite una FSM a 8 stati.