Prova Finale di Reti Logiche

Progetto a cura di:
Francesco Rita, Davide Salonico
Codici Persona:
10794794, 10774487

Professore: Gianluca Palermo
Anno Accademico 2022/2023

Indice

1 | Introduzione

1.1. Scopo del Progetto

L'obiettivo del progetto è la realizzazione di un componente hardware descritto in VDHL, il quale, una volta forniti in input una codifica a 2 bit per la porta d'uscita e una codifica da 0 a 16 bit rappresentante un indirizzo di memoria, deve essere in grado di leggere il dato da una memoria esterna all'indirizzo ricevuto e di presentarlo sull'uscita selezionata.

1.2. Specifiche del Progetto

Il componente legge le informazioni di cui ha bisogno dall'ingresso sequenziale i_w, il quale è abilitato dal segnale i_start che rimane alto da 2 a 18 cicli di clock consecutivi. I bit sono in ordine dal più significativo al meno significativo e i primi due identificano l'uscita da aggiornare. Nel caso in cui non vengano forniti tutti i 16 bit che identificano l'indirizzo di memoria da cui bisogna leggere, bisognerà completarlo con degli zeri nei bit più significativi. Una volta terminata la fase di lettura, si hanno al massimo 20 cicli di clock per leggere il dato dalla memoria e mostrarlo sull'uscita corretta. La lettura delle uscite è abilitata dal segnale o_done che rimane alto per un solo ciclo di clock; quando questo segnale è basso il valore delle uscite è 0. Le quattro uscite del componente o_z0, o_z1, o_z2, o_z3 sono poste inizialmente a 0 (quando viene dato il segnale di reset) e ogni volta che o_done viene alzato, una delle uscite (quella selezionata) viene aggiornata, mentre le altre mostrano l'ultimo valore registrato. Per specifica, il segnale i_start deve attendere che o_done torni a 0 per potersi alzare.

1.3. Interfaccia del Componente

Il componente si presenta con 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;
```

Segue una breve descrizione delle funzioni dei vari ingressi e delle varie uscite.
Ingressi:
i_clk: segnale di CLOCK;
i_rst: segnale di RESET;
i_start: ingresso a 1 bit che fornisce informazioni circa la validità del segnale i_w;
i_w: ingresso sequenziale a 1 bit che specifica l'uscita dalla quale esporre il dato letto in memoria e l'indirizzo dello stesso;
i_mem_data: ingresso a 8 bit che fornisce il dato letto dalla memoria;
Uscite:
o_z0, o_z1, o_z2, o_z3: uscite a 8 bit che espongono, come da specifica, l'ultimo dato di competenza letto in memoria;
o_done: segnala l'effettiva validità dei segnali o_z0, o_z1, o_z2 e o_z3;
o_mem_addr: uscita a 16 bit dove viene specificato l'indirizzo di memoria a cui si vuole accedere;
o_mem_we: segnale per abilitare la scrittura in memoria;
o_mem_en: segnale per abilitare la lettura dalla memoria;

Di seguito si fornisce uno schema del componente project_reti_logiche con una visione a "scatola nera" al fine di evidenziare i terminali di ingresso e di uscita. In Fig 1.1 è rappresentata anche la memoria che però non fa parte del componente.

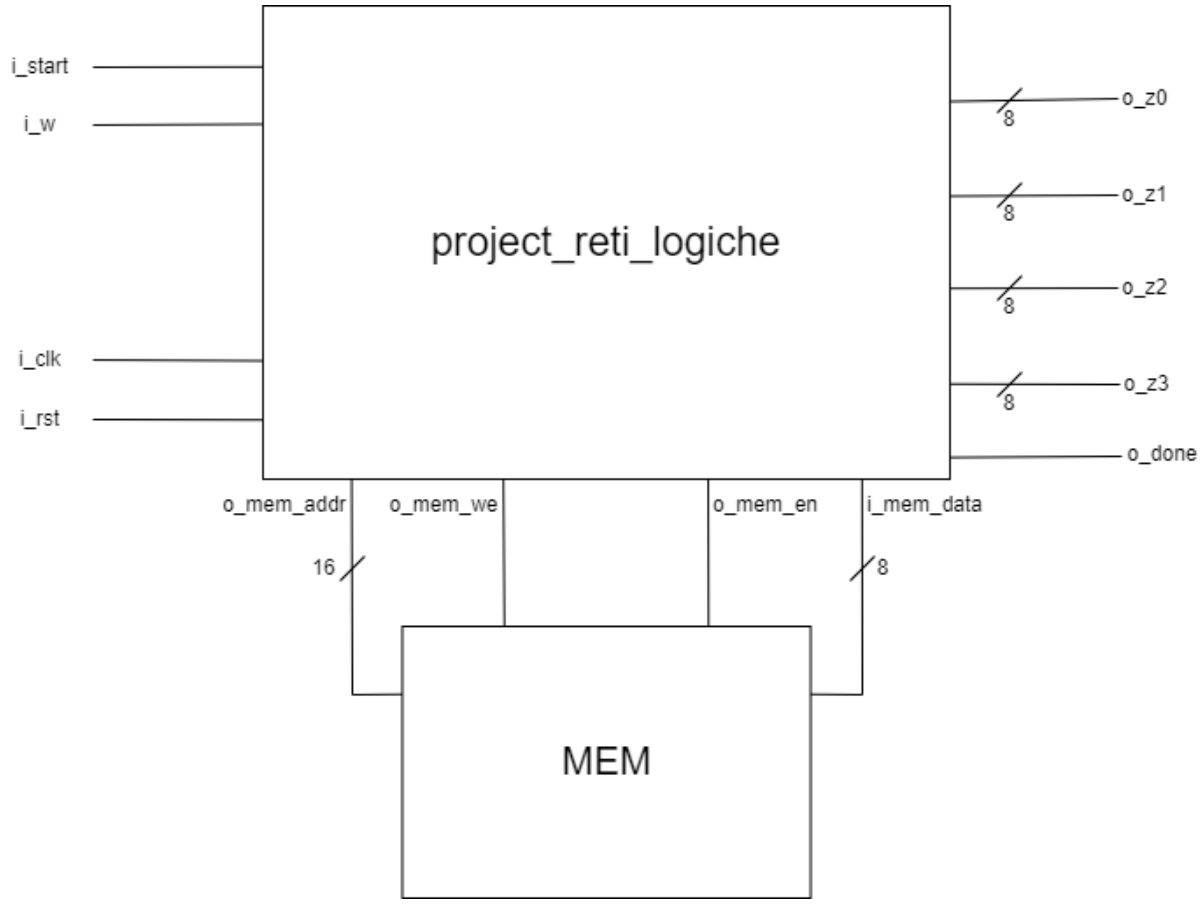


Figura 1.1: Visione del componente a scatola nera

2 | Architettura

Il componente è stato progettato come composizione di due moduli principali: il datapath (Fig 2.2, non dichiarato esplicitamente come entity in VHDL), contenente tutta la logica inerente alla gestione dei dati, e la FSM (Fig 2.1), il cui compito consiste nel generare i segnali che permettono il corretto funzionamento del primo modulo citato. Nel diagramma che illustra la struttura del datapath, al fine di ottenere maggior chiarezza e leggibilità, i segnali di CLOCK e RESET sono di colore grigio, mentre tutti i segnali generati dalla macchina a stati finiti sono colorati di blu.
Segue una breve descrizione del funzionamento del componente:
L'ingresso i_w viene portato in ingresso a entrambi i registri reg_out_index e reg_mem_addr, dove, tramite il segnale di enable generato dai segnali i_start e switch_in, viene garantito che nei due registri vengano salvati rispettivamente i primi due bit (il numero dell'uscita da aggiornare) e i restanti N bit (indirizzo di memoria da cui estrarre il dato). Terminata la fase di lettura, l'informazione in o_mem_addr è già pronta e viene abilitato il segnale o_mem_en, mentre il segnale o_mem_we rimane a 0 durante tutto il funzionamento del componente. Il ciclo di clock successivo viene posto a 1 il segnale reg_we, il quale, tramite un demux pilotato dal contenuto di reg_out_index, abilita la scrittura del dato in i_mem_data solo nel registro di uscita da aggiornare. Successivamente viene innalzato il segnale o_done, che permette di esporre i valori dei registri reg_out tramite dei multiplexer chiamati "show", i quali fanno in modo che alle varie uscite vengano mostrati vettori di tutti zeri quando il segnale o_done è basso. I segnali di CLOCK e RESET svolgono la loro funzione canonica per tutti gli elementi di memoria presenti. Il segnale o_done funge anche da reset per il registro reg_mem_addr, preparandolo per un'eventuale nuova fase di lettura.

Di seguito sono illustrate le varie entity definite in VHDL con un maggiore livello di dettaglio:

2.1. FSM

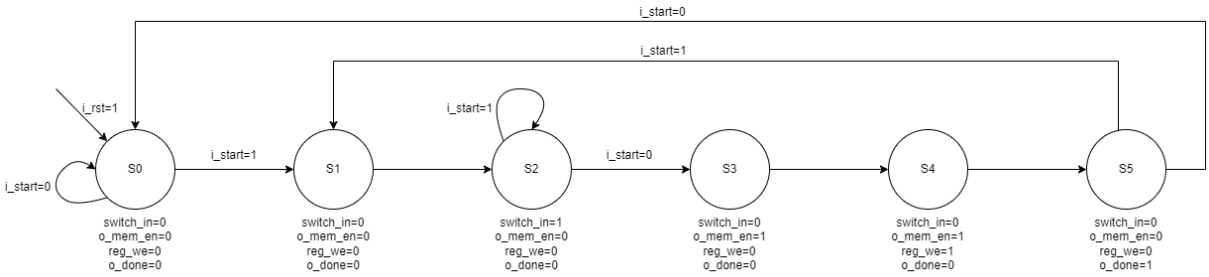


Figura 2.1: Macchina a Stati Finiti

Il modulo è una collezione di process che implementano la macchina a stati, la quale è composta dai seguenti 6 stati:

S0 E' lo stato di reset. La macchina a stati aspetta che il segnale di i_start venga alzato ad 1.

S1 i_start ha valore 1, è stato letto il primo bit che identifica l'uscita.

S2 E' stato letto il numero dell'uscita su cui scrivere il dato. Si leggono i bit che compongono l'indirizzo di memoria.

S3 La lettura dell'indirizzo è terminata, si abilita la memoria in lettura.

S4 Abilita l'enable del registro che salverà il dato letto dalla memoria.

S5 Espone il contenuto dei registri in uscita, preparandosi al contempo ad una nuova lettura.

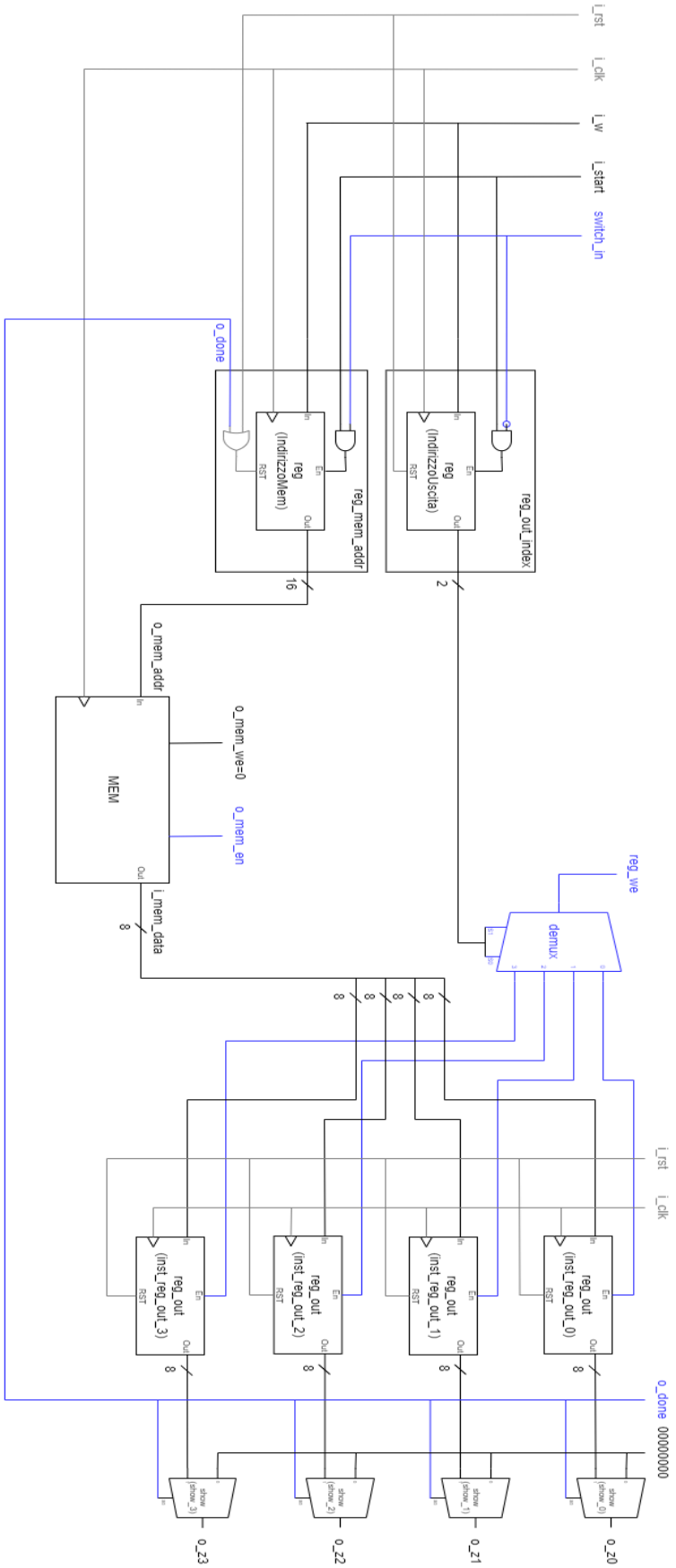


Figura 2.2: Datapath

2.2. Reg Out Index

Questo registro ha ingresso seriale e uscita in parallelo a due bit. Oltre agli ingressi di clk (clock) e rst (reset), ne sono presenti altri due, utili per abilitare la scrittura nel registro: start e switch_in. La descrizione del componente è mista dataflow e behavioural.

2.3. Reg Mem Addr

Questo registro si differenzia da reg_out_index solo per il numero dei bit che riesce a memorizzare (16 al posto di 2) e per la presenza di un ulteriore ingresso per il reset sincrono (routine_rst).

2.4. Reg Out

Questo registro ha sia ingresso che uscita in parallelo a 8 bit. Gli altri ingressi presenti, oltre a quello utilizzato per l'input, sono: clk (clock), rst (reset) e en (enable in scrittura). La descrizione del componente è di tipo behavioural.

2.5. Demultiplexer

Il modulo rappresenta un classico demultiplexer con quattro uscite, controllato da un segnale a 2 bit. La descrizione del componente è di tipo behavioural.

2.6. Show

Il modulo rappresenta un multiplexer con due ingressi, controllato da un segnale a 1 bit. L'ingresso abilitato dalla codifica 0 del segnale di controllo ha valore costante e corrisponde a un vettore di otto zeri. La descrizione del componente è di tipo behavioural.

3 Risutati sperimentali

3.1. Sintesi

Come è possibile notare dall'estratto del report di sintesi (Fig. 3.1), il componente non utilizza latch come elementi di memoria, ma solo flip flop. Da esso si evince anche che l'utilizzo della LUT è del 2%.

```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   25 |     0 |    134600 |  0.02 |
|   LUT as Logic          |   25 |     0 |    134600 |  0.02 |
|   LUT as Memory         |    0 |     0 |     46200 |  0.00 |
| Slice Registers         |   56 |     0 |    269200 |  0.02 |
|   Register as Flip Flop |   56 |     0 |    269200 |  0.02 |
|   Register as Latch     |    0 |     0 |    269200 |  0.00 |
| F7 Muxes                |    0 |     0 |     67300 |  0.00 |
| F8 Muxes                |    0 |     0 |     33650 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

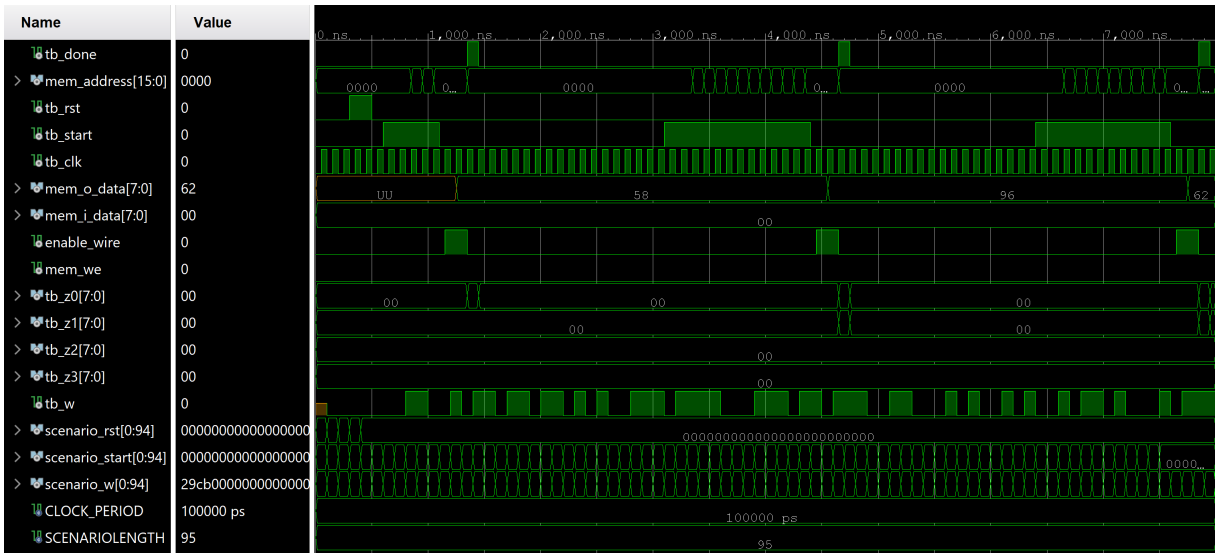
Figura 3.1: Report di Sintesi: Slide Logic

3.2. Simulazioni

Per testare il corretto funzionamento del componente sono stati effettuati numerosi test-bench; alcuni costruiti appositamente per testare i corner cases, altri generati automaticamente attraverso uno script. Tutti i test hanno avuto esito positivo in entrambe le simulazioni Behavioural e Post-Synthesis Functional.
Segue un elenco di brevi test che riteniamo particolarmente significativi:

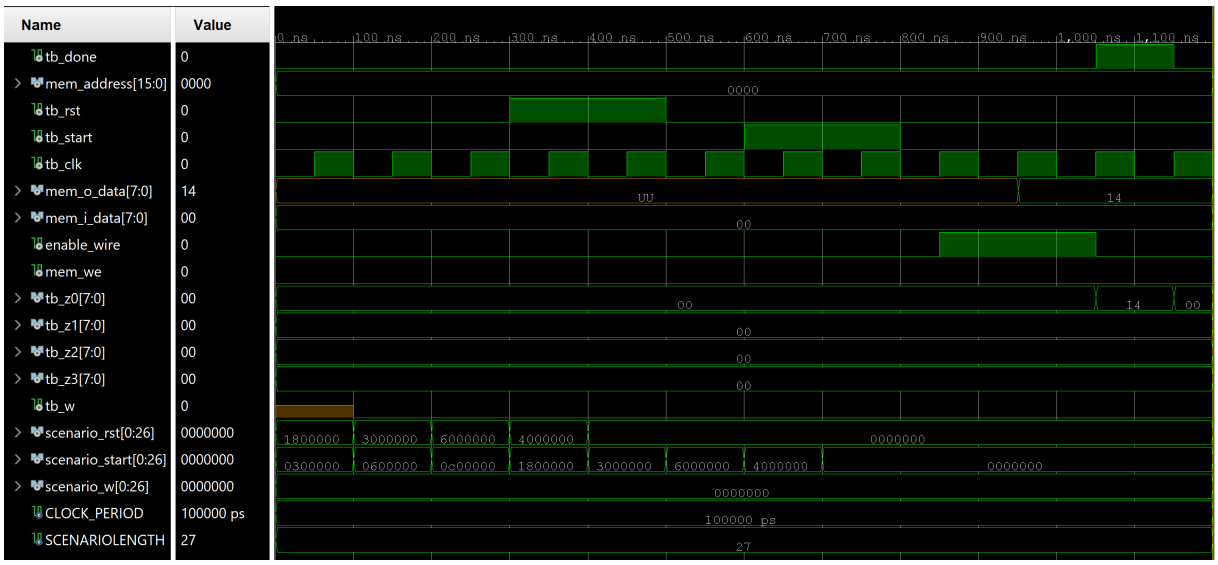
3.2.1. Testbench 1

Questo primo test verifica il corretto funzionamento del componente in condizioni standard, con indirizzi di memoria forniti di lunghezza strettamente compresa tra 0 e 16. In particolare, i_w fornisce degli input da ignorare quando i_start è a 0 e si controlla che l'uscita o_z0 venga aggiornata correttamente più volte.



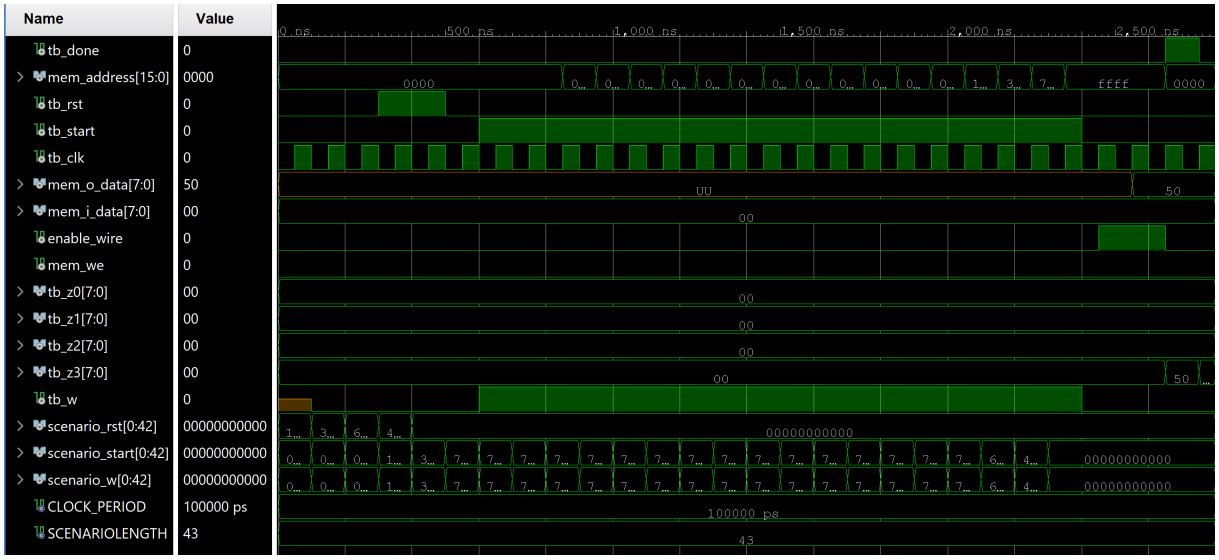
3.2.2. Testbench 2

Questo test verifica che il componente acceda all'indirizzo di memoria 0 (a 16 bit), nel caso limite in cui i_start sia posto a 1 per solo due cicli di clock consecutivi.



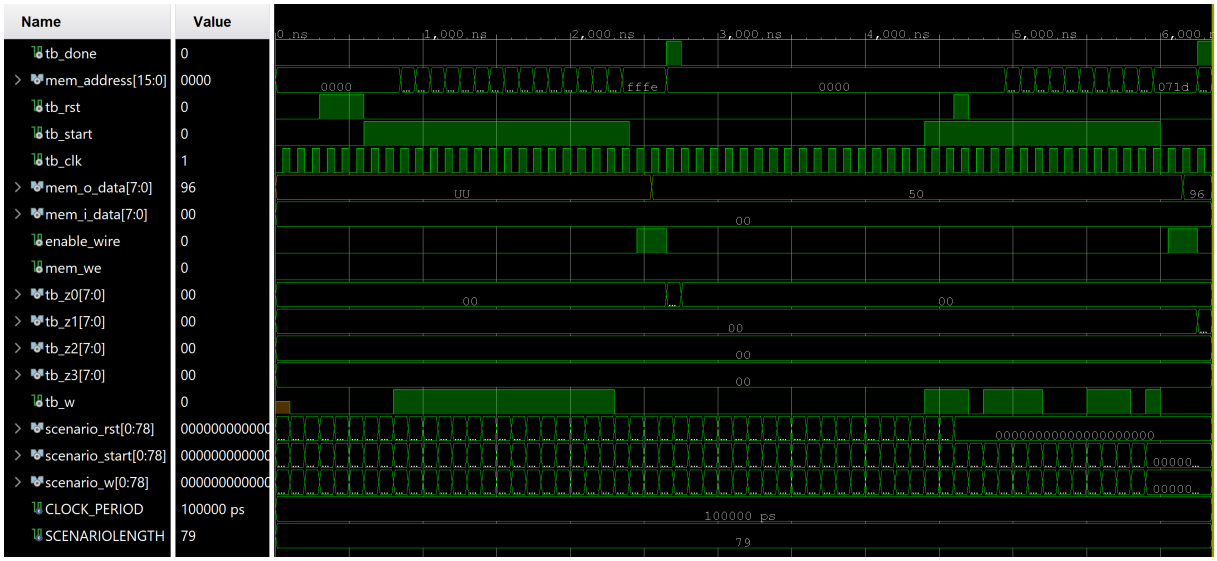
3.2.3. Testbench 3

Questo test verifica che il componente acceda all'ultimo indirizzo di memoria, nel caso limite in cui sia i_start che i_w siano posti a 1 per diciotto cicli di clock consecutivi.



3.2.4. Testbench 4

Questo test verifica il corretto funzionamento del componente nel caso limite in cui esso riceva un segnale di reset asincrono mentre i_start è a 1.



4 | Conclusioni

Per concludere, ci tenevamo a mostrare un'ottimizzazione da noi già implementata e a discutere alcune possibili ottimizzazioni future.

4.1. Ottimizzazioni

Nella FSM è presente una transizione da s5 a s1 con i_start=1 che, per come è strutturata la specifica del progetto, non verrà mai utilizzata, in quanto dopo la fase di lettura devono passare almeno 20 cicli di clock affinché i_start possa tornare a 1, mentre il componente da noi progettato impiega solo 3 cicli di clock per esporre i dati. Questa transizione è stata aggiunta nell'ottica di ottimizzare il componente nel momento in cui il vincolo dei 20 cicli di clock venga rimosso: senza di essa il componente avrebbe bisogno di 4 cicli di clock tra la terminazione di una fase di lettura e l'inizio della successiva, poiché dopo aver dato il segnale o_done è necessario riposizionarsi in s0 prima di poter iniziare a leggere nuovamente; con la transizione che va direttamente in s1, si guadagna facilmente un ciclo di clock.
Una possibile ottimizzazione che permetterebbe di risparmiare un altro ciclo di clock, eliminando di fatto lo stato s3 della FSM, consiste del mantenere o_mem_en sempre a 1, perché così facendo, appena l'indirizzo di memoria è completo si ha subito accesso al dato da prelevare. Si può fare ancora meglio, scendendo a un solo ciclo di clock per la fase di computazione, mostrando il dato aggiornato nello stesso ciclo di clock in cui viene scritto nel registro reg_out. Quest'ultima ottimizzazione ha però un costo, in quanto, per realizzarla, è necessario sdoppiare l'ingresso di ogni registro reg_out e sono necessari quattro ulteriori multiplexer (uno per ogni registro), che permettano di scegliere se mostrare il dato in uscita dal registro (dato vecchio) o il dato in ingresso al registro (dato aggiornato).