POLITECNICO
MILANO 1863

Progetto Finale di Reti Logiche

Chiara Thien Thao Nguyen Ba - Codice Persona 10727985
Flavia Nicotri - Codice Persona 10751801

Anno Accademico 2022-2023
Prof. Fabio Salice

Indice

1 Introduzione

1.1 Scopo del progetto

Dato un ingresso seriale e quattro canali di uscita, lo scopo del progetto è quello di implementare un modulo hardware tramite codice VHDL che si interfacci con una memoria. L'ingresso seriale fornisce il canale di uscita e l'indirizzo di memoria a cui accedere per leggere il dato da presentare in uscita.

1.2 Specifiche generali

All'istante iniziale, quello corrispondente a `i_rst` uguale a `1`, i quattro canali di uscita sono inizializzati a `00000000` e `o_done` è a `0`.
L'elaborazione inizia quando `i_start` è portato a `1`, a partire da questo momento viene letto sequenzialmente `i_w` fintanto che `i_start` rimane a `1`.
Il segnale `i_w` ha una lunghezza variabile da un minimo di 2 bit ad un massimo di 18 bit. I primi due bit di `i_w` corrispondono ad uno dei quattro canali di uscita, mentre i restanti N bit corrispondono all'indirizzo di memoria da cui leggere il dato. Poiché la lunghezza N è variabile, per identificare l'indirizzo di memoria da 16 bit, se è necessario, bisogna estendere l'indirizzo letto aggiungendo 16-N `0` nei bit più significativi dell'indirizzo.

Le uscite rimangono a `0` durante tutta l'elaborazione. Al termine dell'elaborazione, quando viene portato `o_done` alto a `1`, viene mostrato il dato appena letto nell'uscita corrispondente al canale selezionato e nelle uscite degli altri canali vengono mostrati gli ultimi dati precedentemente letti e salvati (nel caso in cui si ha la prima elaborazione queste uscite sono a `0`).

Il segnale `i_start` può essere portato a `1` solo dopo che `o_done` viene riportato a `0`. In questo modo è possibile effettuare una nuova elaborazione.

1.3 Interfaccia del componente

Di seguito si riporta l'interfaccia del componente:

```
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;
```

Dove:

- `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 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_address` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `i_mem_address` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_mem_enable` 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 Esempio

Un esempio è riportato qui di seguito in figura. Nella prima elaborazione, `i_start` è alto per 5 cicli di clock, per cui vengono letti i 5 bit di `i_w` corrispondenti (`10110`). I primi due bit di `i_w`, in questo caso `10`, indicano che il canale di uscita scelto è Z2. I restanti bit di `i_w`, in questo caso `110`, indicano l'indirizzo di memoria, che verrà esteso a `0000000000000110`, da cui leggere il dato. In seguito, quando `o_done` viene portato a `1` viene visualizzato in uscita dal canale Z2 il dato letto dall'indirizzo di memoria.

Nella successiva elaborazione, `i_start` è alto per 2 cicli di clock, per cui vengono letti i 2 bit di `i_w` corrispondenti. In questo caso il canale selezionato è Z1, poichè vengono letti i bit `01`, mentre l'indirizzo di memoria è `0000000000000000`. In seguito, quando `o_done` viene portato a `1` viene visualizzato in uscita dal canale Z1 il dato letto dalla memoria e anche il dato letto nell'elaborazione precedente sul canale di uscita Z2.

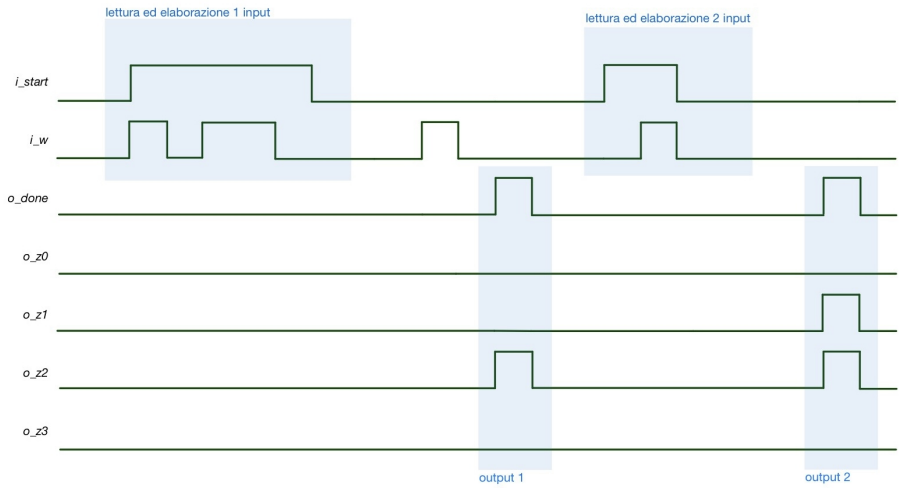

Figura 1: Esempio di funzionamento

2 Architettura

L'architettura del modulo da noi implementato prevede tre distinti processi:

1. Processo di gestione dei segnali `i_clk` e `i_rst`;
2. Processo per identificare lo stato prossimo nella macchina a stati finiti (FSM);
3. Processo di gestione dei segnali nella macchina a stati finiti (FSM);

2.1 `i_clk` e `i_rst` process

È il processo che gestisce l'inizializzazione dei registri e il passaggio da uno stato all'altro nella FSM.

Quando il segnale `i_rst` è portato alto a `1`, tutti i registri vengono inizializzati. Il reset del modulo è asincrono, quindi può avvenire in qualsiasi momento della computazione.

Quando il segnale `i_clk` commuta dal fronte basso al fronte alto, la FSM passa allo stato successivo ed esegue le istruzioni associate allo stato corrente.

2.2 Next state process

È il processo che stabilisce qual è lo stato prossimo della FSM sulla base dello stato corrente. In alcuni casi, la scelta dello stato prossimo è determinata dal valore assunto dal segnale `i_start`.

2.3 Process per la gestione dei segnali

È il processo che gestisce i diversi segnali, inizializzandoli a `0` e portandoli a `1`, quando necessario, per il corretto funzionamento della FSM.

Il processo è descritto da una FSM composta da 7 stati, riportati qui di seguito con la propria descrizione. In seguito, è riportata la figura che rappresenta la FSM; nello specifico sono evidenziati i segnali alti, omettendo gli altri segnali che di default sono da intendersi a `0`.

`IDLE`

È lo stato iniziale che raggiungo quando `i_rst` è portato a `1` e nel quale rimango fintanto che `i_start` è uguale a `0`. Non appena `i_start` passa a `1`, viene salvato il primo bit del canale di uscita scelto nel registro `regCh`. Inoltre, il segnale `clear_addr` è portato a `1` perchè viene inizializzato il registro `regAddr` in cui verrà salvato l'indirizzo di memoria.

`CH`

È lo stato che raggiungo al primo `i_start` uguale a `1`. Viene letto il secondo bit di `i_w` corrispondente al canale di uscita su cui scrivere il dato letto da memoria.

In questo modo, i due bit del canale vengono salvati nell'apposito registro `regCh`.

READ

È lo stato che raggiungo se dopo la lettura del canale ho ancora `i_start` uguale a `1` e vi rimango fintanto che questa condizione è verificata. Vengono letti i bit di `i_w` corrispondenti all'indirizzo di memoria da cui estrarre il dato. Ogni bit letto viene salvato nel registro `regAddr`.

ADDR

È lo stato che raggiungo non appena `i_start` è portato a `0`. Avendo salvato tutto l'indirizzo di memoria nel registro `regAddr`, `o_mem_en` è portato a `1` e `o_mem_we` a `0` per poter leggere il dato contenuto nell'indirizzo di memoria letto.

MEM

È lo stato in cui viene salvato il dato letto dalla memoria nel registro `regIN`.

LOAD

È lo stato in cui viene scelto il canale di uscita sulla base del valore precedentemente salvato nel registro `regCh`. Il dato salvato nel registro `regIN` viene caricato nel registro dell'uscita selezionata.

DONE

È lo stato in cui termina l'elaborazione e vengono visualizzati i dati salvati nei registri delle uscite nei rispettivi canali di uscita.

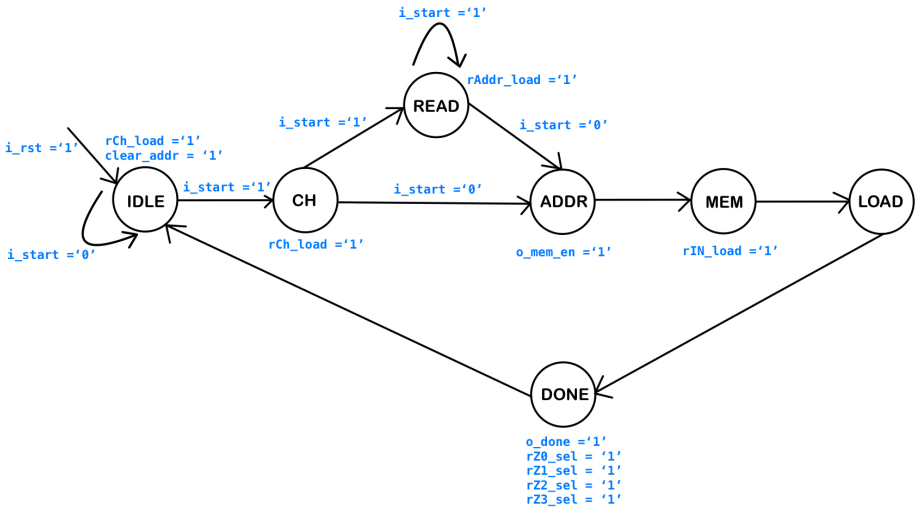

Figura 2: FSM del modulo implementato

2.4 Scelte progettuali

Per la realizzazione del progetto siamo partite dalla costruzione del datapath, individuando i componenti necessari per il funzionamento del circuito.

Per il salvataggio del canale e dell'indirizzo di memoria ricavati da `i_w` abbiamo utilizzato due shift-register, in modo tale da leggere in ingresso il dato seriale `i_w` e produrlo in uscita in parallelo (registro Serial IN Parallel OUT). In particolare, per l'indirizzo di memoria abbiamo aggiunto il segnale `clear_addr` in modo tale da inizializzare il registro all'inizio di ogni elaborazione, così da ottenere in uscita l'indirizzo corretto già esteso sui 16 bit.

Per il dato letto dalla memoria e per il salvataggio dei dati nei canali di uscita abbiamo usato dei registri paralleli da 8 bit.

Per la scelta del canale di uscita abbiamo usato un demultiplexer 1-4, che usa come ingresso di selezione il contenuto del registro `regCh` e salva il dato letto da memoria nel registro corrispondente all'uscita selezionata.

Infine, per distinguere quale dato mandare alle uscite abbiamo usato per ogni canale di uscita un multiplexer 2-1 che seleziona il dato salvato nel registro se è selezionato `1`, oppure `00000000` se è selezionato `0`.

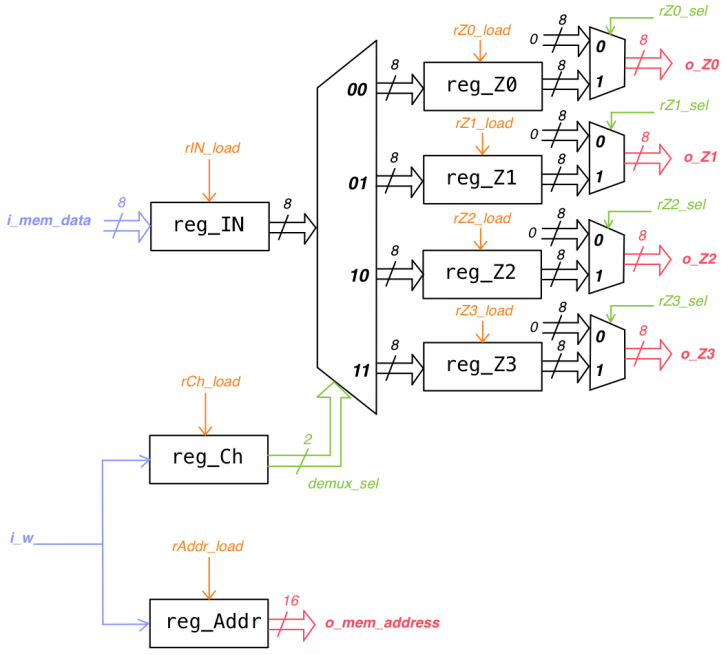

Figura 3: Datapath

3 Sintesi

3.1 Utilization report

Il dispositivo è perfettamente sintetizzabile senza inferred latch e utilizza 26 Look Up Table (LUT) e 65 Flip Flop.

Resource	Utilization	Available	Utilization %
LUT	26	134600	0.02
FF	65	269200	0.02
IO	63	285	22.11

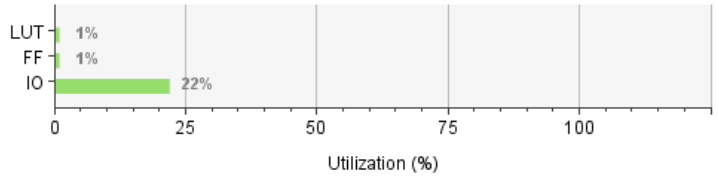

Figura 4: Utilization report

3.2 Timing report

Analizzando il tempo di esecuzione, ci siamo focalizzate sul Worst Negative Slack, ovvero l'indice che mostra quanto tempo rimane al completamento di un ciclo di clock nel peggiore dei casi. Nel nostro progetto si verifica che in un ciclo di clock di 100ns, esso è pari a 97.491ns.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	97,491 ns	Worst Hold Slack (WHS):	0,142 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:	128	Total Number of Endpoints:	128	Total Number of Endpoints:	66

Figura 5: Timing report

3.3 Schema di sintesi

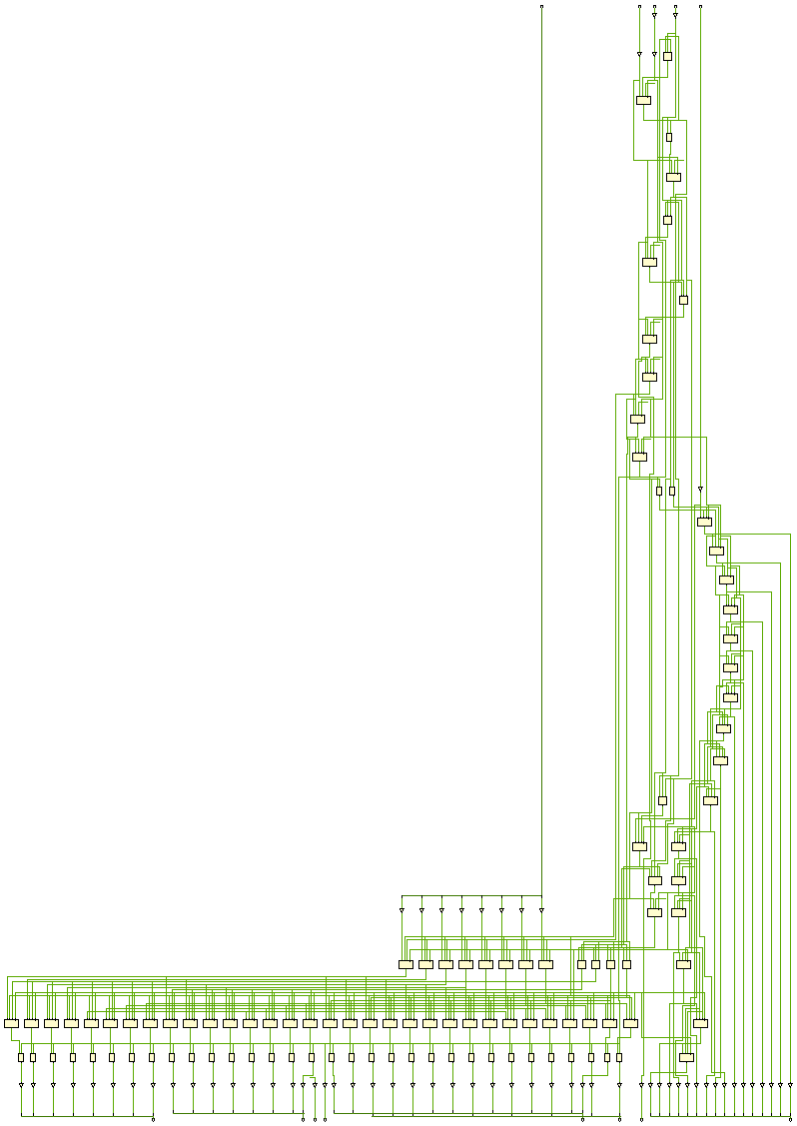

Figura 6: Schema di sintesi del componente

4 Simulazioni

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

Il modulo ha passato anche i test generati da noi per valutare delle condizioni limite che si potrebbero verificare. Qui di seguito riportiamo alcuni dei casi più rilevanti che abbiamo analizzato.

4.1 Scrittura sui diversi canali di uscita

È un test che verifica il corretto funzionamento dei registri delle uscite in una situazione standard, ovvero nel caso in cui si verificano varie elaborazioni successive che interessano i diversi canali di uscita. Inoltre, si verifica il caso in cui il segnale di reset viene portato a `1` alla fine dell'elaborazione (reinizializzando tutti i registri delle uscite a `0`). Questo verifica che il segnale di reset lavora in modo asincrono.

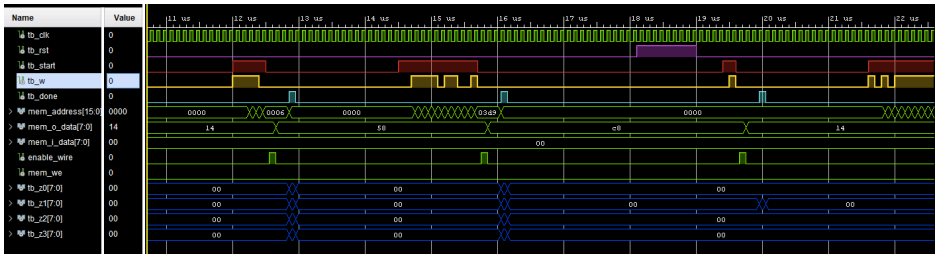

Figura 7: Test Bench 1

4.2 Input `i_w` su 18 bit

È un test che verifica il corretto funzionamento dei registri delle uscite nel caso limite in cui il segnale `i_w` è formato da 18 bit, quindi l'indirizzo di memoria è completamente dato (16 bit).

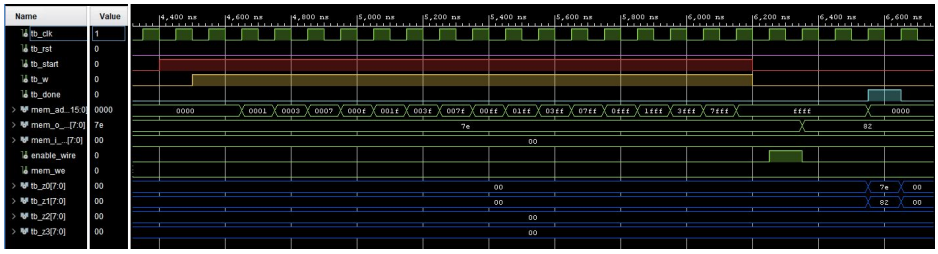

Figura 8: Test Bench 2

4.3 Input `i_w` su 2 bit

È un test che verifica il corretto funzionamento dei registri delle uscite nel caso limite in cui il segnale `i_w` è formato da 2 bit, quindi l'indirizzo di memoria è `0000000000000000`.

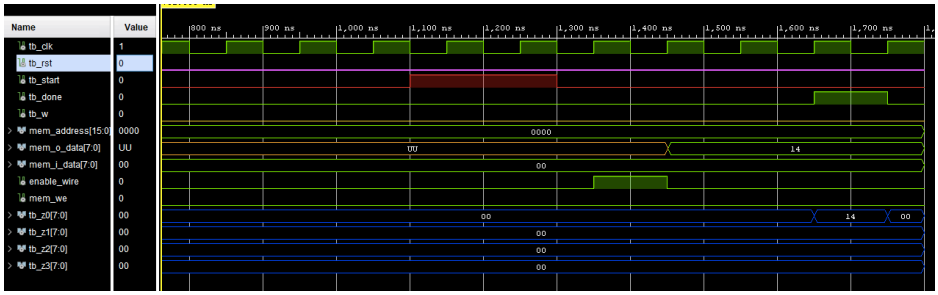

Figura 9: Test Bench 3

4.4 Scrittura sullo stesso canale di uscita

È un test che verifica il corretto aggiornamento dell'uscita nel caso limite in cui il canale di uscita selezionato è sempre lo stesso (in questo caso Z2). Di conseguenza, viene mostrato il corretto caricamento del dato letto da memoria nell'uscita selezionata.

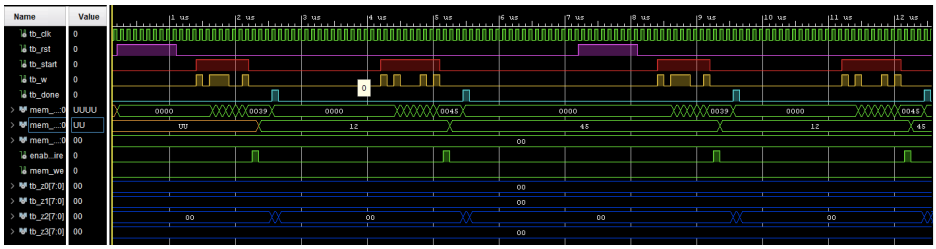

Figura 10: Test Bench 4

4.5 Reset alto durante la lettura di `i_w`

È un test che verifica il corretto funzionamento del circuito nel caso limite in cui il segnale `i_rst` è portato a `1` quando ancora il segnale `i_start` è alto a `1`. Di conseguenza, l'elaborazione si interrompe poichè il modulo viene reinizializzato. Anche in questo caso si verifica che il segnale di reset è asincrono e funziona correttamente in situazioni borderline.

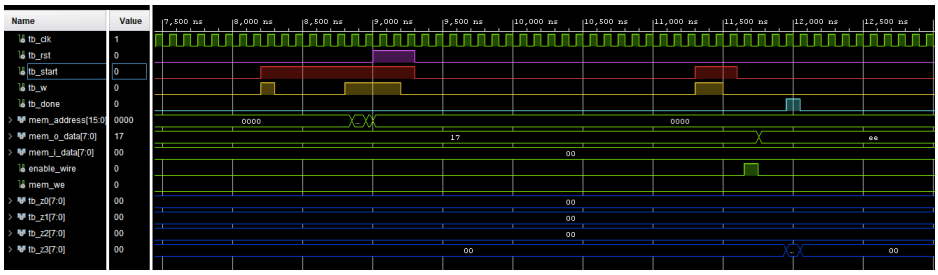

Figura 11: Test Bench 5

5 Conclusioni

Per la realizzazione di questo progetto, inizialmente è stato difficile capire il funzionamento di VHDL e di Vivado.
Non avendo mai utilizzato questi strumenti, abbiamo seguito le indicazioni fornite durante l'esercitazione. Abbiamo quindi abbozzato su carta il circuito, individuando i componenti che ci sarebbero serviti e in seguito abbiamo elaborato una FSM che descriveva il comportamento dei segnali ad ogni passo. Questo ci ha permesso di avere una visione più chiara delle richieste della specifica e di come implementare le nostre idee tramite VHDL.

Inizialmente il comportamento del circuito analizzando i Test Bench non era quello richiesto, di conseguenza, analizzando le forme d'onda e gli ASSERT abbiamo corretto gli errori, riuscendo quindi a passare tutti i Test Bench forniti.

Siamo complessivamente soddisfatte del lavoro svolto, perchè ci ha permesso di affinare con la pratica quanto appreso durante il corso. Inoltre, è stato gratificante vedere come quello che inizialmente era solo un'idea, prendere forma in un progetto completo e funzionante.