Progetto di Reti Logiche

Simone Bevilacqua (10720775) - Luca Barda (10697790)
Politecnico di Milano
Corso: Prova Finale (Progetto di Reti Logiche)
Docente: William Fornaciari
Anno accademico: 2022 / 2023

Indice

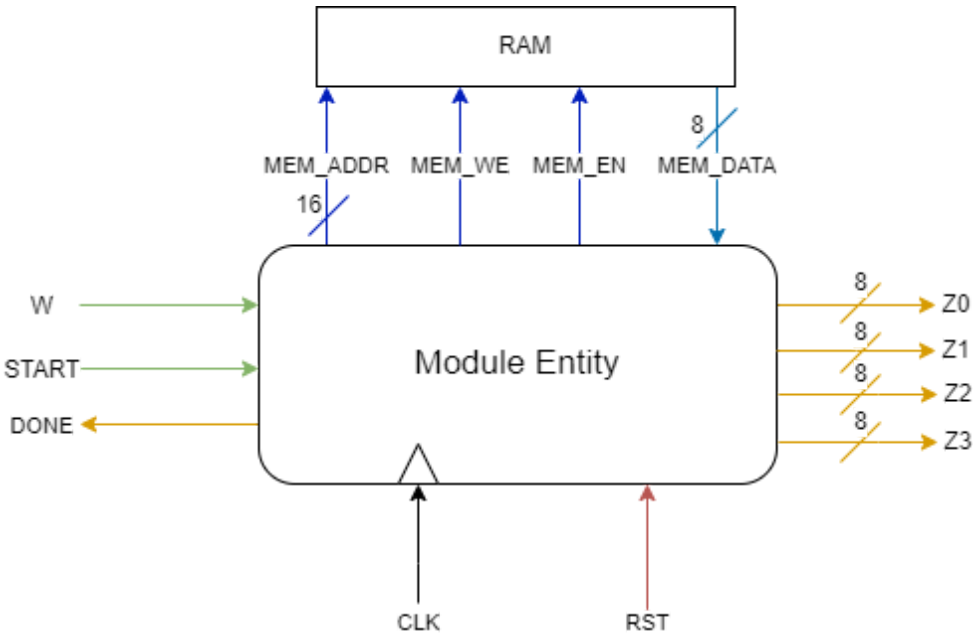
1. Introduzione

1.1 Obiettivo

L'obiettivo è l'ideazione e la progettazione di un modulo HW descritto in VHDL in grado di leggere dei dati da una memoria esterna ed esporli sulle sue uscite.

1.2 Interfaccia modulo

- 2 ingressi primari da 1-bit: **i_w** e **i_start**;
- 5 uscite primarie: **o_z0**, **o_z1**, **o_z2**, **o_z3** a 8-bit e **o_done** a 1-bit;
- 1 ingresso dalla memoria: **i_mem_data**;
- 3 uscite verso la memoria: **o_mem_addr**, **o_mem_we**, **o_mem_en**;
- 1 segnale di clock: **i_clk**;
- 1 segnale di reset asincrono: **i_rst**.



1.3 Specifiche

In input al modulo sono forniti 2<=N<=18 bit sull'ingresso seriale i_w, considerati utili solo durante il periodo in cui i_start = '1'. I primi 2 bit rappresentano il canale d'uscita ("00"=>o_z0, "01"=>o_z1, "10"=>o_z2, "11"=>o_z3), mentre i restanti rappresentano i

bit dell'indirizzo di memoria da cui leggere il dato, dal più significativo al meno significativo. L'indirizzo della memoria andrà esteso con un numero sufficiente di leading zeros nel caso in cui N<18 fino a raggiungere 16-bit.
Una volta scritto il dato sul registro di uscita corretto si deve porre o_done = '1' per esattamente un ciclo di clock. In generale si ha o_z0=o_z1=o_z2=o_z3 = "0000 0000" per tutto il tempo in cui o_done = '0', mentre quando o_done = '1' o_z0, o_z1, o_z2 e o_z3 mostrano l'ultimo dato letto dalla memoria salvato sul rispettivo registro ("0000 0000" se non è ancora stato esposto alcun dato sull'uscita).
i_clk ha un periodo di 100ns e una nuova richiesta di lettura dalla memoria (i_start = '1') deve attendere il fronte di discesa del segnale o_done o al più 20 cicli di clock.
Allo stato di reset tutti i registri di uscita vengono azzerati e viene annullata l'eventuale richiesta non conclusa in corso prima del segnale di reset asincrono.

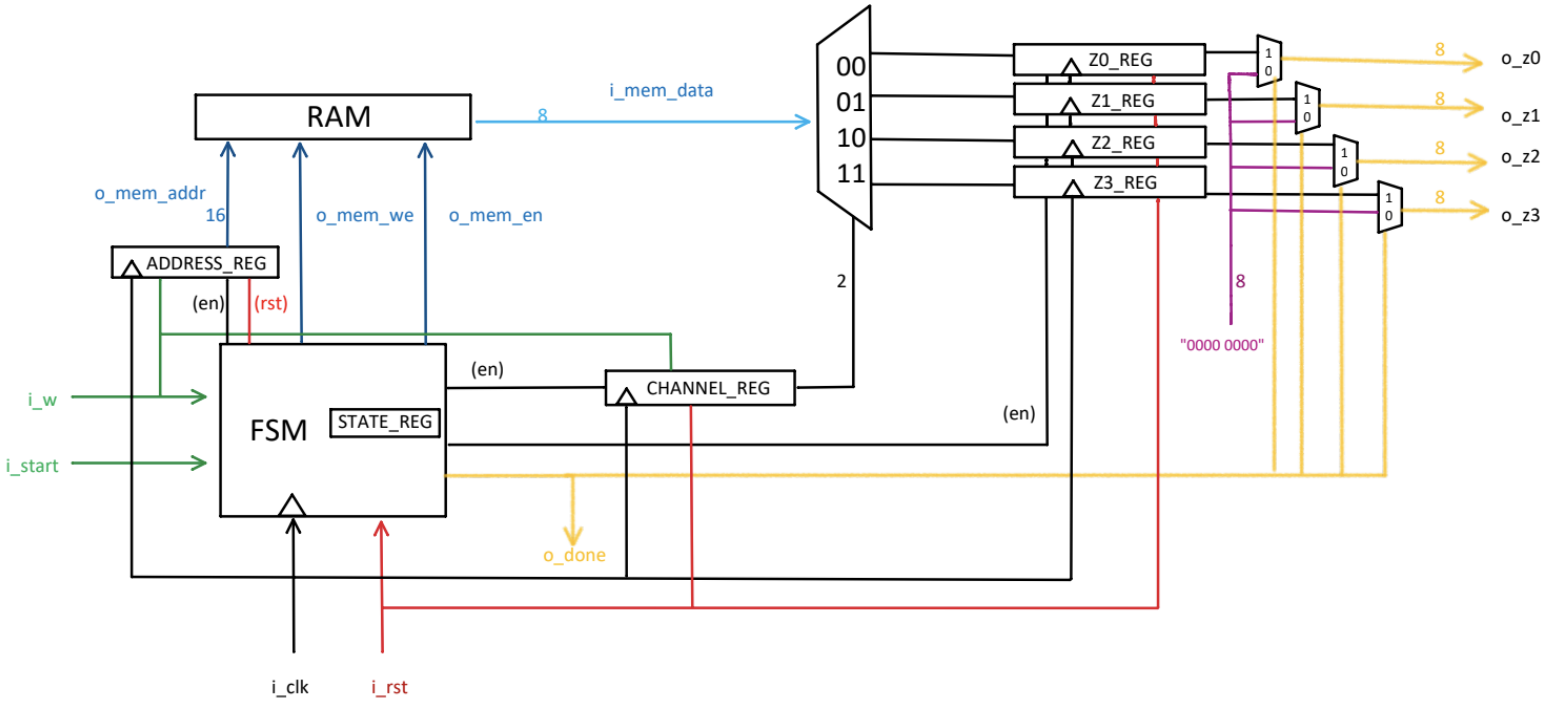
1.4 Descrizione memoria

La memoria con cui il modulo si interfaccia è a 16 bit con parole di 8 bit. Il modulo può leggere un dato dalla memoria ponendo il segnale o_mem_we = '0', il segnale o_mem_en = '1' e l'indirizzo richiesto su o_mem_addr. La memoria garantisce di rispondere con il dato salvato all'indirizzo richiesto su i_mem_data sempre entro 2ns (e quindi sicuramente entro il successivo ciclo di clock).

2. Architettura

2.1 Datapath

Il modulo centrale del progetto è rappresentato dalla macchina a stati FSM che svolge la funzione di controllore dell'entità. La FSM comunica con la memoria per richiedere i dati desiderati e con i registri per gestire il salvataggio dei dati tramite i segnali di enable (en). Lo shift-register ADDRESS_REG viene resettato dalla FSM ogni volta che si raggiunge lo stato iniziale, gli altri registri vengono invece resettati dal segnale i_rst.



2.1.1 Segnali e registri

Per la gestione dei componenti sono stati definiti i seguenti segnali:

- **state_reg**: rappresenta lo stato corrente della FSM;
- **address_reg**: salva l'indirizzo di memoria esteso su 16-bit;
- **channel_reg**: salva il codice binario relativo all'uscita dove esporre il dato;
- **z0_reg**: salva l'ultimo dato visualizzato sull'uscita corrispondente (o_z0);
- **z1_reg**: salva l'ultimo dato visualizzato sull'uscita corrispondente (o_z1);

- **z2_reg**: salva l'ultimo dato visualizzato sull'uscita corrispondente (o_z2);
- **z3_reg**: salva l'ultimo dato visualizzato sull'uscita corrispondente (o_z3).

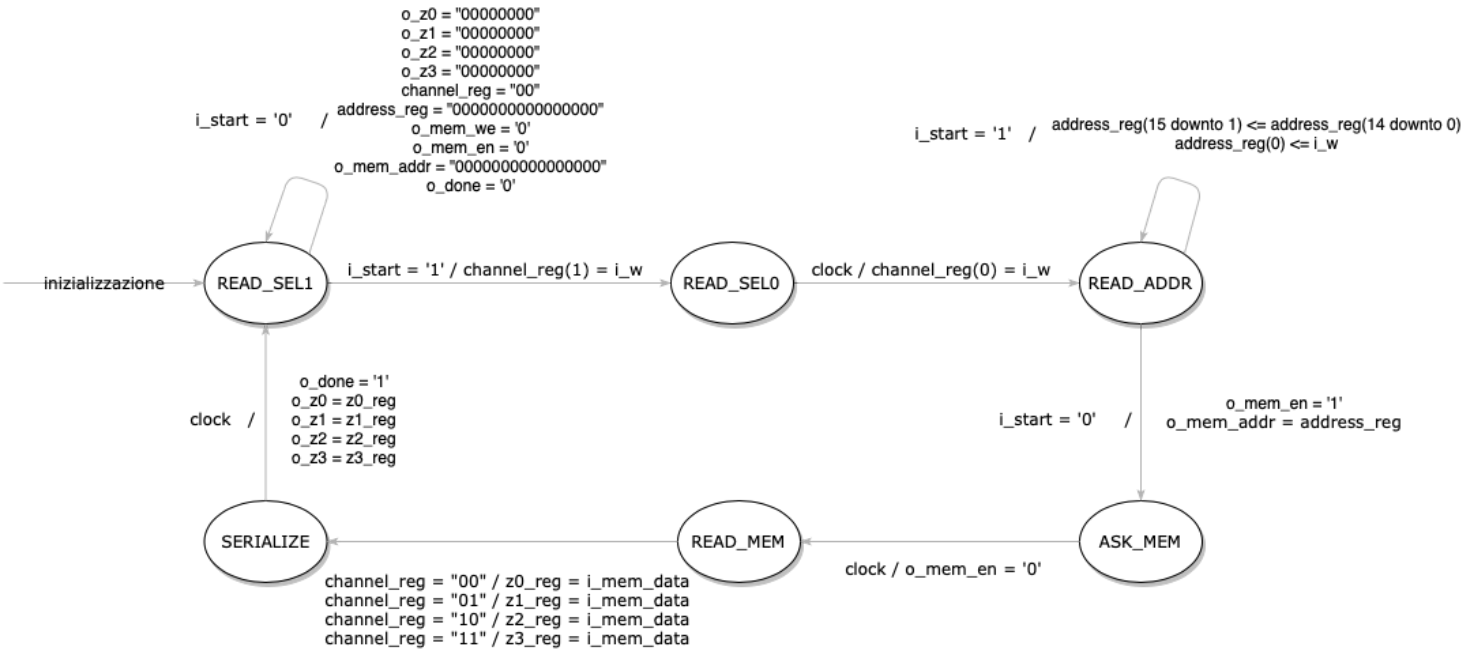
Ogni registro è anche connesso ai seguenti segnali:

- **i_clk**: segnale di clock per salvare i dati in modo sincrono sul fronte di salita;
- **I_rst**: per gestire il reset dei registri in modo asincrono.

2.2 FSM

2.2.1 Schema

Il diagramma degli stati del controllore dell'entità (FSM) è rappresentato nella seguente figura.



2.2.2 Descrizione stati

La FSM realizzata è una macchina di Mealy composta dai seguenti stati:

- **READ_SEL1**: stato iniziale della FSM (stato di reset). Pone il controllore in attesa del MSB relativo al codice binario a 2-bit che identifica la porta di uscita sulla quale verrà mostrato il dato letto dalla memoria;

- **READ_SEL0**: pone il controllore in attesa del LSB relativo al codice binario a 2-bit che identifica la porta di uscita sulla quale verrà mostrato il dato letto dalla memoria;
- **READ_ADDR**: pone il controllore in attesa degli eventuali bit (da 0 a 16) che compongono l'indirizzo di memoria dove è salvato il dato da mostrare in uscita;
- **ASK_MEM**: richiede alla memoria il dato all'indirizzo esteso su 16-bit computato precedentemente;
- **READ_MEM**: legge dalla memoria il dato richiesto;
- **SERIALIZE**: imposta il segnale o_done = '1', mostrando sulle 4 uscite i valori letti dalla memoria (i dati letti in precedenza assieme a quello letto a seguito dell'ultima richiesta alla memoria).

2.3 Scelte progettuali

Abbiamo scelto di usare un solo processo che implementa le funzioni di transizione di stato, transizione di uscita e aggiornamento dei registri interni.

Per quanto riguarda il salvataggio della porta di uscita utilizziamo un registro a 2-bit dedicato.

Per il salvataggio dell'indirizzo di memoria utilizziamo uno shift register a 16-bit che permette, ad ogni ciclo di clock, di salvare il nuovo bit letto dal segnale i_w, mantenendo allo stesso momento l'indirizzo temporaneo esteso su 16-bit pronto per essere fornito in ingresso alla memoria.

Abbiamo inoltre inizializzato il segnale o_mem_we al valore '0' senza mai modificarlo in quanto il modulo non ha mai la necessità di scrivere in memoria.

3. Sintesi

3.1 Report Utilization

Analizzando il report del progetto fornito da Vivado si vede che la sintesi del componente è stata effettuata correttamente senza generare errori o warning. Inoltre possiamo vedere che per la sintesi della FSM a 6 stati è stata correttamente utilizzata una codifica binaria a 3-bit.

```
Report Check Netlist:
+------+------------------+-------+---------+-------+------------------+
|      |Item              |Errors |Warnings |Status |Description       |
+------+------------------+-------+---------+-------+------------------+
|1     |multi_driven_nets |      0|        0|Passed |Multi driven nets |
+------+------------------+-------+---------+-------+------------------+
```

```
-------------------------------------------------------------------------------------
      State |                     New Encoding |                Previous Encoding
-------------------------------------------------------------------------------------
     iSTATE |                              000 |                               000
    iSTATE0 |                              001 |                               001
    iSTATE1 |                              010 |                               010
    iSTATE2 |                              011 |                               011
    iSTATE3 |                              100 |                               100
    iSTATE4 |                              101 |                               101
```

Per quanto riguarda i componenti effettivamente sintetizzati per la realizzazione del modulo la seguente sezione del report ne riporta un elenco dettagliato mostrando i registri e i mux utilizzati.

```
Detailed RTL Component Info :
+---Registers :
                   16 Bit    Registers := 2
                    8 Bit    Registers := 8
                    2 Bit    Registers := 1
                    1 Bit    Registers := 3
+---Muxes :
       2 Input     16 Bit         Muxes := 2
       2 Input      8 Bit         Muxes := 4
       8 Input      3 Bit         Muxes := 1
       2 Input      2 Bit         Muxes := 3
       2 Input      1 Bit         Muxes := 9
       4 Input      1 Bit         Muxes := 4
       5 Input      1 Bit         Muxes := 1
       3 Input      1 Bit         Muxes := 1
       6 Input      1 Bit         Muxes := 7
```

Analizzando il “Synthesis Report” fornito da Vivado in seguito alla sintesi possiamo osservare che sono state utilizzate 64 LUT (Look-Up Tables), 103 FF (Flip Flops) e nessun latch.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	64	0	134600	0.05
LUT as Logic	64	0	134600	0.05
LUT as Memory	0	0	46200	0.00
Slice Registers	103	0	269200	0.04
Register as Flip Flop	103	0	269200	0.04
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

3.2 Timing Report

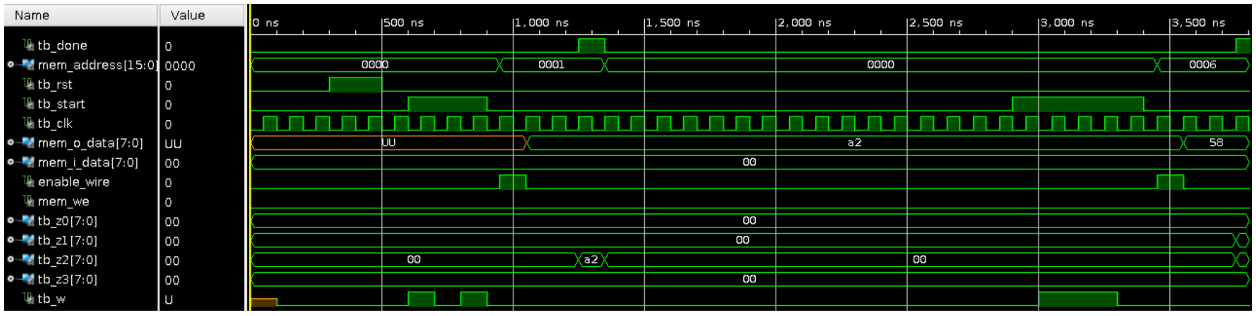
Analizzando il “Timing Report” fornito sempre da Vivado al termine della sintesi del modulo, il dato interessante da osservare è lo Slack, ovvero il tempo inutilizzato all’interno del ciclo di clock in cui l’elaborazione è durata di più.
Nel nostro caso lo Slack è pari a 97,460ns, il che indica che durante un ciclo di clock, al massimo l’elaborazione dura 2,540ns (= 100ns - 97,460ns).
Tenendo conto che la memoria utilizzata per questo progetto ha un tempo di risposta pari a 2ns, possiamo stabilire che, per funzionare correttamente, il componente sintetizzato necessita di un periodo di clock non inferiore a 4,540ns (= 2,540ns + 2ns) che corrisponde ad una frequenza di clock massima di circa 220 GHz.

4. Simulazioni

In seguito alla sintesi ci siamo dedicati al testing per valutare il corretto funzionamento del componente.

4.1 TestBench standard

Questo è il test bench d'esempio che ci è stato fornito. Testa le funzionalità di base, ovvero la scrittura di due dati provenienti da due indirizzi di memoria diversi su due porte di uscita differenti.

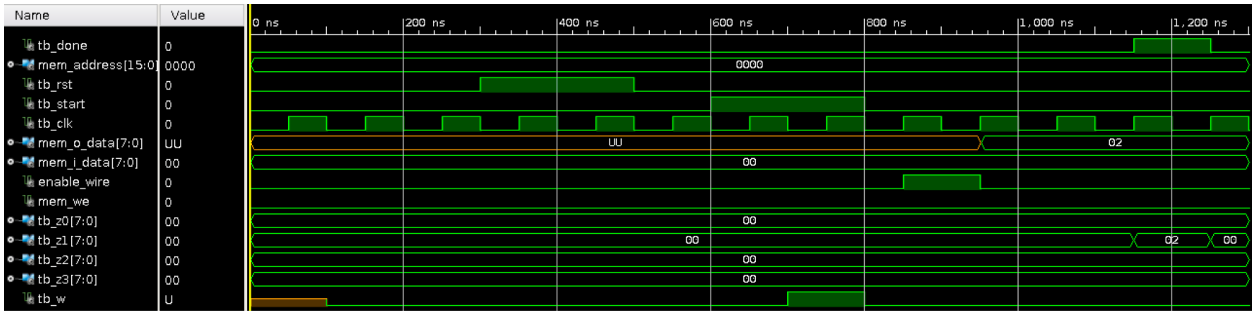


4.2 TestBench casi limite

Durante la creazione di ulteriori test bench abbiamo prediletto le situazioni che coprono casi limite "scomodi" per il funzionamento del componente.

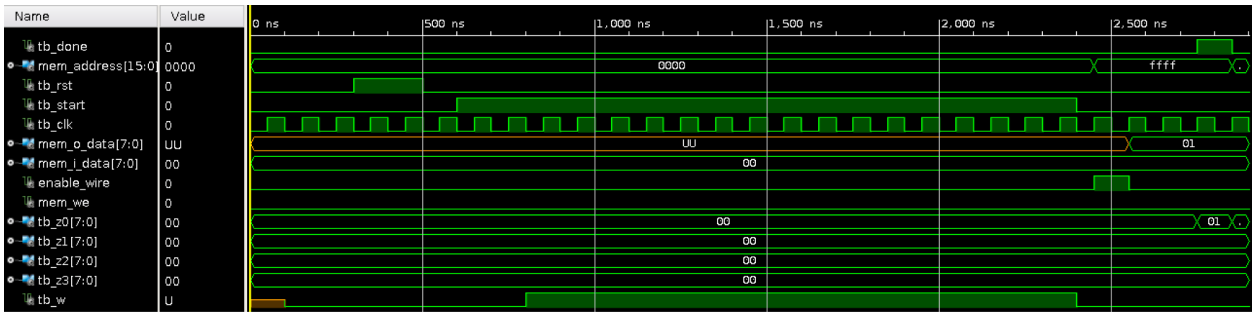
4.2.1 TestBench con indirizzo a 0-bit

Questo test bench testa il funzionamento del componente nel caso in cui il segnale i_start rimanga attivo per soli 2 cicli di clock (numero minimo). Per definizione l'indirizzo di memoria dove andare a leggere il dato è "0000 0000 0000 0000".



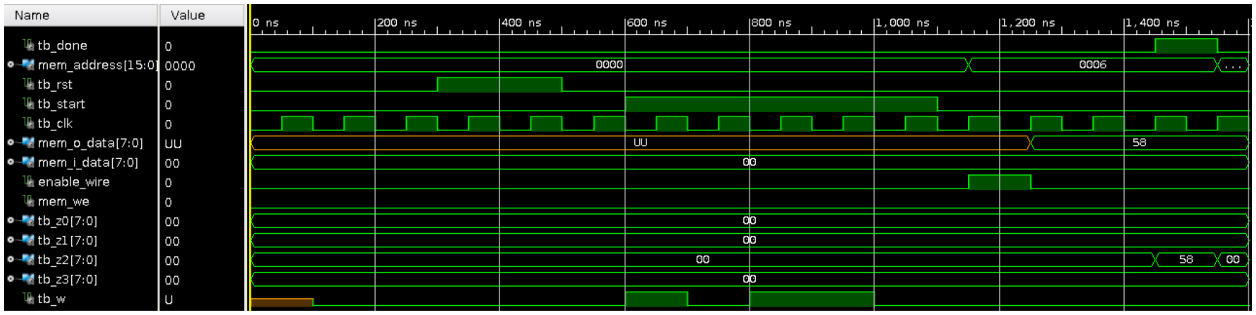
4.2.2 TestBench con indirizzo a 16-bit

Questo test bench testa il funzionamento del componente nel caso in cui il segnale i_start rimanga attivo per 18 cicli di clock (numero massimo). In particolare vogliamo testare se lo shift register funzioni correttamente in casi di capienza massima (16-bit).



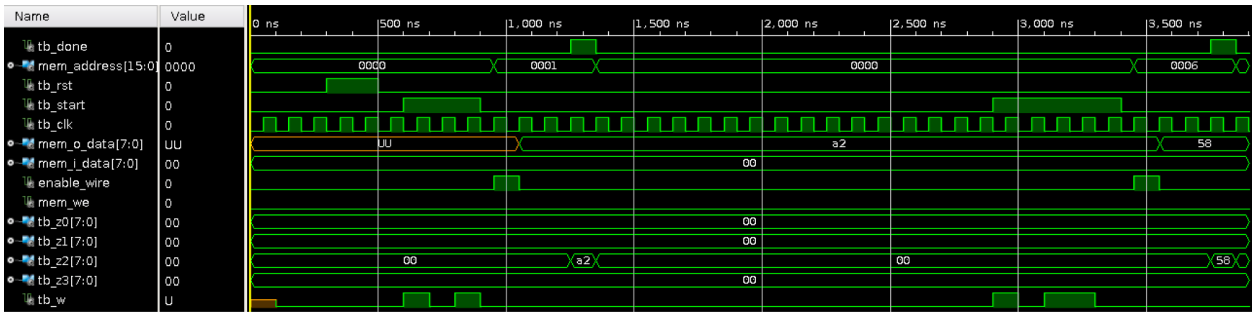
4.2.3 TestBench con indirizzo generico esteso su 16-bit

Questo test bench testa il funzionamento del componente nel caso in cui il segnale i_start rimanga attivo per un numero di cicli di clock generico compreso tra 2 (numero minimo) e 16 (numero massimo). In particolare vogliamo testare se lo shift register esegua correttamente l'estensione a 16-bit dell'indirizzo.



4.2.4 TestBench con riscrittura su porta di uscita

Questo test bench testa il funzionamento del componente nel caso in cui, in due esecuzioni successive, il dato letto dalla memoria venga mostrato sulla stessa porta di uscita. In particolare vogliamo testare se i registri di uscita gestiscono correttamente la riscrittura di un nuovo dato.

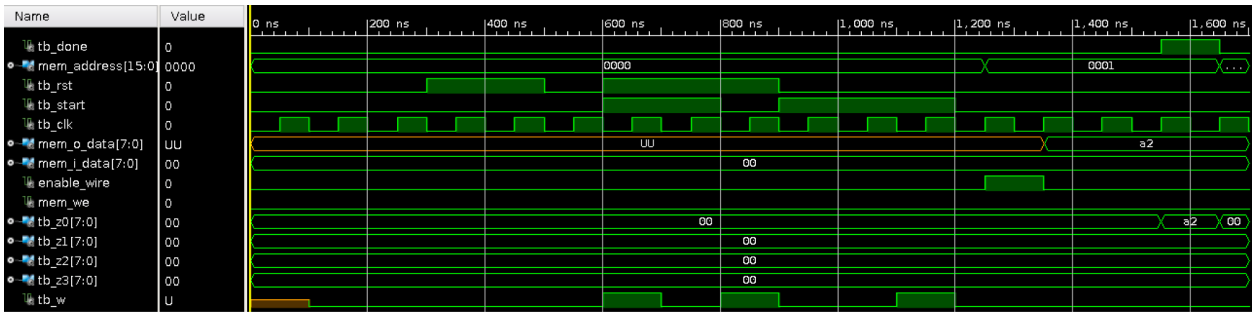


4.3 TestBench reset asincrono

Un segnale del componente che non viene testato dai precedenti test bench è il segnale di reset (i_rst). Questo segnale è particolare rispetto agli altri visto che funziona il modo asincrono.

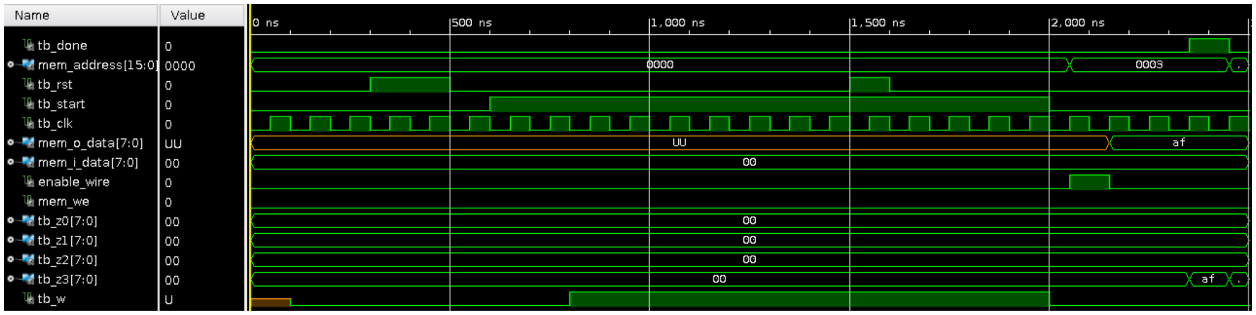
4.3.1 TestBench reset asincrono ad inizio lettura

Questo test bench testa il funzionamento del componente nel caso in cui il segnale i_rst sia alto all'inizio di una nuova lettura, sia in corrispondenza di i_start = '1' sia quando i_start = '0'. In particolare vogliamo testare il reset dei vari registri coinvolti.



4.3.2 TestBench reset asincrono durante la lettura

Questo test bench testa il funzionamento del componente nel caso in cui il segnale i_rst sia alto durante una lettura, quindi quando lo shift register e gli altri registri hanno uno stato “intermedio” (hanno già salvato alcuni dati). In particolare vogliamo testare il reset dei vari registri coinvolti e la capacità del componente di interpretare correttamente i bit letti su i_w: successivamente al segnale di reset i primi due bit saranno riferiti alla porta di uscita e i possibili successivi all’indirizzo di memoria.



5. Conclusioni

La realizzazione del progetto è iniziata analizzando la specifica del componente e individuando le fasi operative principali. Siamo quindi passati alla realizzazione del datapath scegliendo i segnali interni e la tipologia di registri da utilizzare.

Abbiamo anche discusso di come estendere su 16-bit l'indirizzo, pensando dapprima ad un semplice registro a 16-bit da estendere correttamente solamente al momento di i_start = '0', scegliendo infine uno shift register che minimizza i cicli di clock di "elaborazione" tra i_start = '0' e o_done = '1' e rende disponibile l'indirizzo temporaneo correttamente esteso in ogni momento.

Successivamente ci siamo dedicati alla stesura di una FSM con un numero maggiore di stati rispetto a quella finale. Dopo un'ulteriore analisi l'abbiamo riprogettata riducendo il numero di stati e a questo punto la scrittura del codice è risultata abbastanza diretta e veloce.

Dopo la scrittura del codice ci siamo dedicati al debug del componente risolvendo alcuni problemi di inizializzazione dei segnali.

Come ultimo passo abbiamo controllato nuovamente tutto il codice cercando possibili ottimizzazioni come, ad esempio, lasciare il segnale o_mem_en = '1' solo i cicli di clock strettamente necessari (1 ciclo di clock) per evitare di tenere occupata la memoria esterna più del necessario, nell'ipotesi che ci possano essere altri moduli che interagiscono in lettura e/o scrittura con essa.