Relazione Progetto di Reti Logiche

Anno Accademico 2022/2023
Prof. Fabio Salice

Andrea Iodice, Sara Massarelli
Codici Persona: 10697363 (AI), 1074016 (SM)

POLITECNICO
MILANO 1863

Introduzione

Il modulo hardware da realizzare è un dispositivo che si interfaccia con una memoria.

Esso ha il compito di andare a interrogare l'unità di memoria, richiedendo il contenuto di una determinata cella, il cui indirizzo viene fornito in ingresso al dispositivo, e – una volta che la memoria ha fornito il contenuto richiesto – esso viene mandato in uscita su uno dei quattro canali disponibili appartenenti al dispositivo.

La memoria non era da realizzare, ma ci è già stata fornita mediante specifiche di progetto.

Interfaccia del modulo hardware

L'interfaccia viene riportata di seguito:

```
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
- `i_rst` è il segnale di reset in ingresso
- `i_start` è il segnale di start in ingresso
- `i_w` è il segnale che porta l'informazione riguardo all'indirizzo della memoria da interrogare e al canale di uscita da utilizzare
- `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
- `o_mem_we` è il segnale di write enable da dover mandare alla memoria: se vale 1 si seleziona la scrittura su memoria, se vale 0 si possono fare operazioni di lettura dalla memoria.

Di seguito, in Figura 1, le frecce tratteggiate rappresentano i segnali da 8 bit, le frecce continue fini rappresentano segnali da 1 bit, mentre quella continua e spessa rappresenta un segnale da 16 bit.

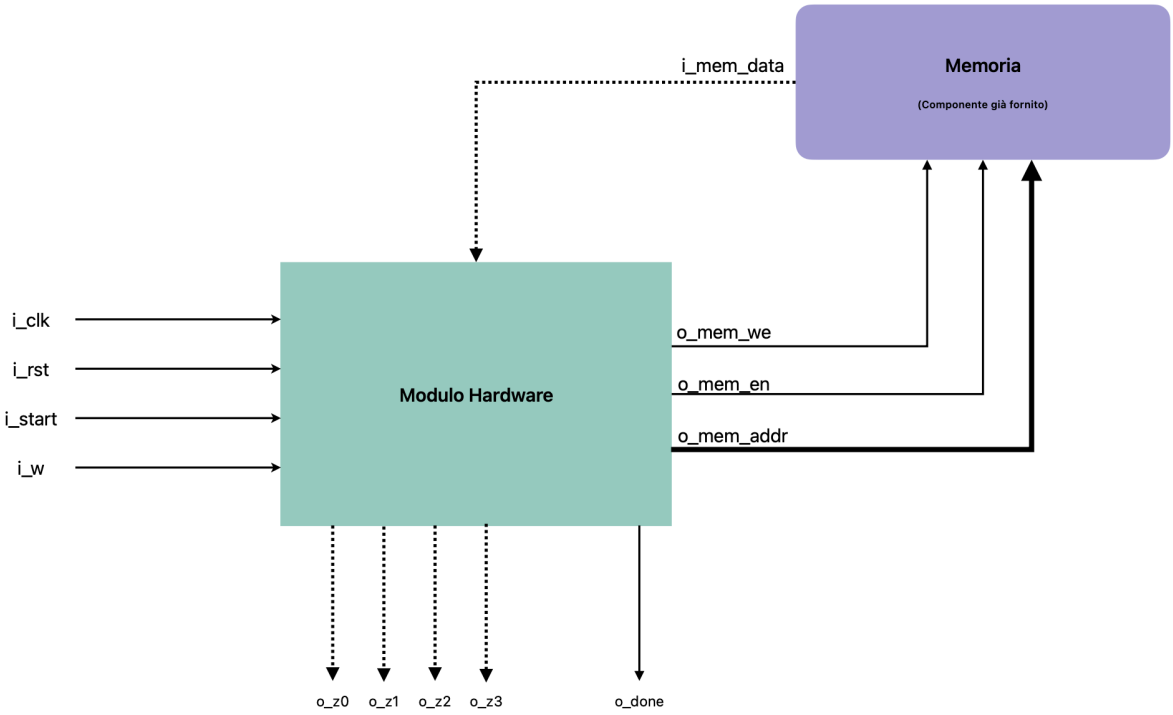

Figura 1

Funzionamento del modulo

All'istante iniziale, quando il modulo riceve il segnale di reset, le uscite hanno i seguenti valori: Z0, Z1, Z2 e Z3 sono 0000 0000, DONE è 0.

Una volta che il segnale `i_rst` diventa 0, il modulo resta in attesa che `i_start` diventi 1: a questo punto il modulo legge i dati in ingresso dal segnale `i_w`, che viene letto sul fronte di salita del clock.

Il segnale contiene informazioni sull'indirizzo della memoria da interrogare e sul canale di uscita selezionato, organizzate nel seguente modo:

- 2 bit per il canale di uscita, codificato in binario naturale (00 per z0, 01 per z1, e così via=
- Da 0 a 16 bit per l'indirizzo della memoria.

Nel caso in cui non vengano forniti tutti e 16 i bit per l'indirizzo è necessario estendere i bit nella parte più significativa fino a 16 con degli 0.

Ad esempio, se vengono forniti i seguenti 9 bit, l'indirizzo diventa:
111011000 → 0000000111011000, dove i bit in blu rappresentano l'estensione effettuata dal modulo.

Terminata la lettura da `i_w`, cioè quando `i_start` diventa 0, il modulo interroga la memoria e, una volta ricevuto il contenuto della cella desiderata, manda in uscita il dato appena ottenuto nel canale corrispondente.

Contemporaneamente, imposta il segnale `o_done` a 1 e mostra nei tre canali rimanenti i risultati delle precedenti interrogazioni, se disponibili, altrimenti mostrerà 16 bit a 0.

Il dispositivo mostra i valori sulle uscite per un ciclo di clock e successivamente imposta `o_done` a 0, così come i quattro canali, e si mette in attesa del successivo ciclo di computazione, cioè quando `i_start` tornerà a 1.

Qualora il modulo veda commutare il segnale `i_rst` da 0 a 1, esso procede a resettarsi.

In Figura 2 viene mostrato un esempio della sequenza di funzionamento del modulo realizzato.

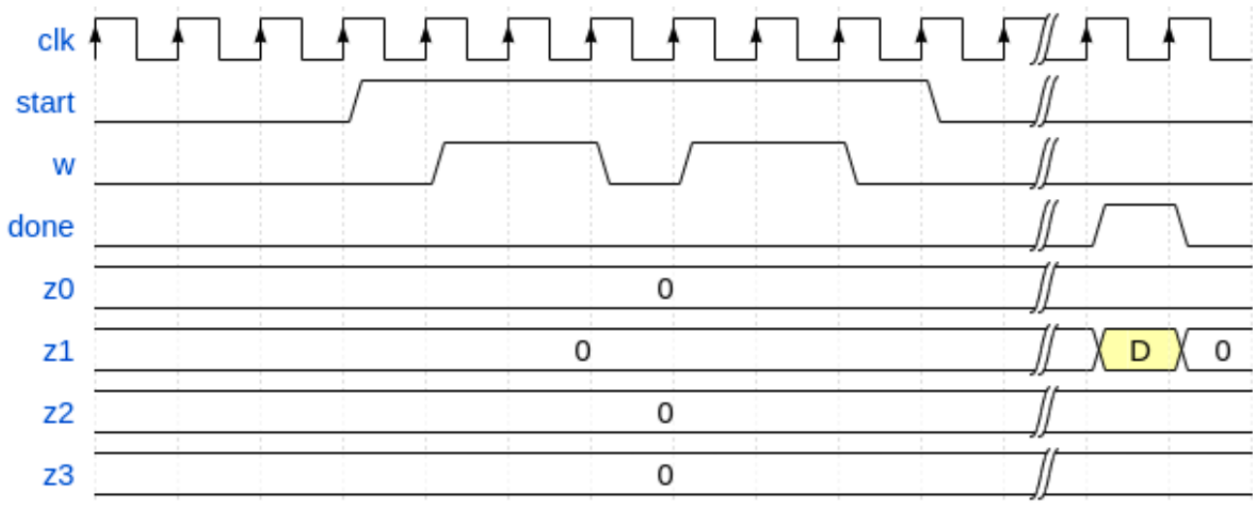

Figura 2

Architettura

Il componente hardware è stato implementato in linguaggio VHDL tramite un singolo process, che va a descrivere il comportamento di una Macchina a Stati Finiti, riportata in Figura 3.

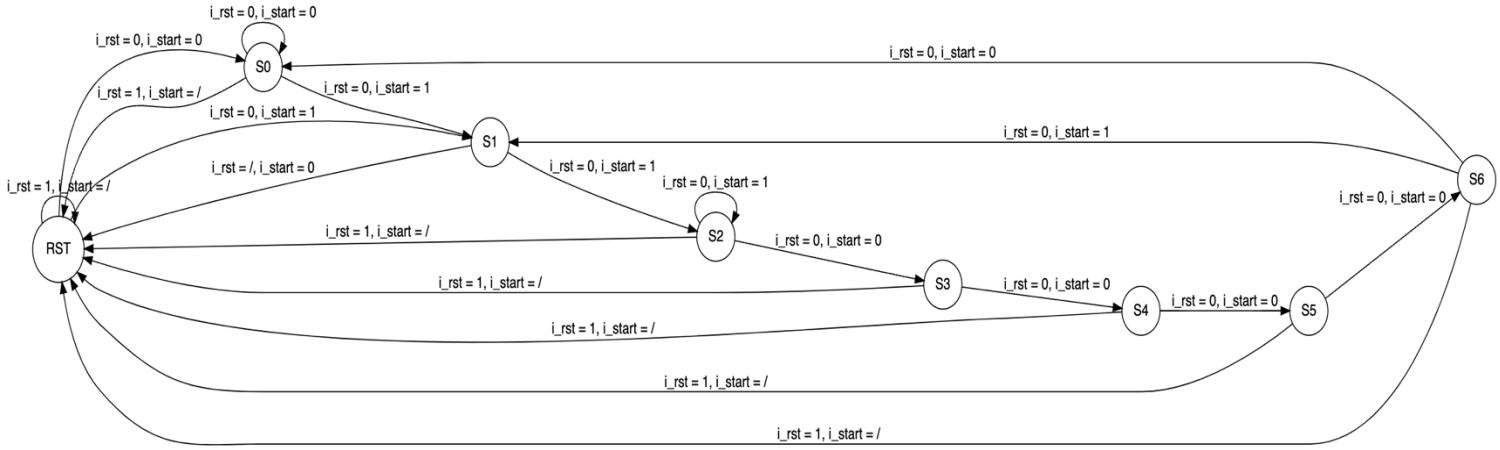

Figura 3

Di seguito è presente la descrizione di cosa accade in ogni stato.

→ *Stato RST:*
È lo stato iniziale della Macchina a Stati Finiti, dove viene eseguito il reset dei signal interni all'architettura e delle uscite del dispositivo. Ogni volta che il segnale `i_rst` vale 1, la macchina giunge a questo stato.

→ *Stato S_0:*
Dopo aver effettuato il reset, si giunge in questo stato di attesa: si rimane in S0 fino a quando `i_start` vale 0; non appena esso commuta da 0 a 1, si va nello stato S1 e si legge il primo bit da `i_w`.

→ *Stato S_1:*
Stato in cui si legge il secondo bit da `i_w`: a questo punto è stata effettuata la lettura del numero di canale di uscita da utilizzate in questo ciclo di computazione.

→ *Stato S_2:*
Dopo aver letto il numero del canale di uscita da usare, il modulo inizia a leggere da `i_w` i bit che costituiscono l'indirizzo della memoria al quale si vuole accedere: fino a quando `i_start` vale 1 si resta in questo stato e, ad ogni ciclo di clock, si legge un bit da `i_w`.
Quando `i_start` commuta da 1 a 0, si impostano le variabili `o_mem_addr` con l'indirizzo appena letto, `o_mem_we` a 0 – dato che stiamo cercando di leggere dalla memoria – e `o_mem_en` a 1, per accedere alla memoria.
Infine, si passa allo stato S3.

→ *Stato S_3:*
Stato in cui si attende che la memoria fornisca il risultato.

→ *Stato S_4:*
Stato in cui si legge il risultato prodotto dalla memoria – contenuto in `i_mem_data` – e si mostrano nei canali di uscita il valore richiesto e quelli dei cicli di computazione precedenti, se disponibili. Contemporaneamente, `o_done` viene impostato a 1.

→ *Stato S_5:*
Stato in cui si azzerano le uscite.

→ *Stato S_6:*
Stato finale del ciclo di computazione, dove vengono resettate alcuni segnali interni al modulo. Supponendo che `i_rst` valga 0,da qui si va in S0 nel caso in cui `i_start` valga 0, mentre si va in S1 leggendo il primo bit del canale di uscita da `i_w` se `i_start` vale 1 (in questo ultimo caso è iniziato un nuovo ciclo di computazione).
Se invece `i_rst` vale 1, si ritorna allo stato RST.

Scelte progettuali

All'interno dell'architettura abbiamo dichiarato vari segnali, alcuni usati per controllare il modulo, altri usati per memorizzare dati.

I segnali `b1` e `b0` memorizzano in codifica binaria naturale il numero del canale di output da utilizzare e sono rispettivamente il **MSB** e il **LSB** del numero di uscita.

I segnali `valoreZ0`, `valoreZ1`, `valoreZ2`, `valoreZ3` sono vettori che memorizzano i valori delle uscite dei cicli di computazione precedenti a quello attuale e vengono sovrascritti alla fine di ogni ciclo.

Il segnale `indirizzo` è usato come supporto per leggere i bit dell'indirizzo di memoria da `i_w` e impostare la variabile `o_mem_addr` di conseguenza. Ogni volta che posso leggere un bit dall'indirizzo da `i_w` facciamo uno shift left logical di una posizione e impostiamo il **LSB** di indirizzo con il valore letto da `i_w`.
Così facendo, non c'è bisogno di fare esplicitamente un'estensione di segno con gli 0 in modo esplicito ed è possibile usare un solo stato per leggere l'indirizzo da `i_w` (S2).

Inoltre, abbiamo usato quattro variabili dentro il process, `temp0`, `temp1`, `temp2`, `temp3`, dato che le variabili, a differenza dei signal, vengono aggiornate subito dentro un process.
Questa cosa è essenziale per poter leggere il risultato prodotto dalla memoria e mostrare i valori sulle uscite del dispositivo nello stesso ciclo di clock.

Risultati sperimentali: sintesi

Utilization report

Per il dispositivo abbiamo utilizzato una FPGA Artix-7 xc7a200tfbg484-1.
Il processo di sintesi non ha inferito latch ed ha portato all'utilizzo di 103 Flip-Flop e 75 Look-Up Table.

Di seguito, in Figura 4, sono presenti informazione dettagliate riguardo all'utilizzo della FPGA.

Resource	Utilization	Available	Utilization %
LUT	75	134600	0.06
FF	103	269200	0.04
IO	63	285	22.11

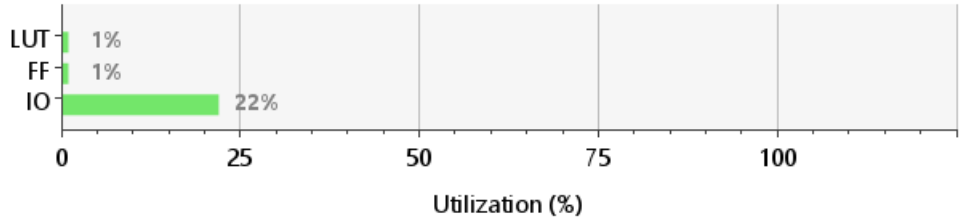

Figura 4

Timing report

Il dispositivo rispetta inoltre il time constraint imposto dalle specifiche: dalla Figura 5 è possibile intuirlo guardando nella sezione Setup la voce Worst Negative Slack (WNS).
Essa rappresenta il numero di nanosecondi rimanenti al completamento del ciclo di clock nel path peggiore di tutti.
Poiché il suo valore è maggiore di 0, il dispositivo rispetta il time constraint.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	96,942 ns	Worst Hold Slack (WHS):	0,139 ns	Worst Pulse Width Slack (WPWS):	49,500 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:	299	Total Number of Endpoints:	299	Total Number of Endpoints:	104

All user specified timing constraints are met.

Figura 5

Schema di sintesi del componente

Di seguito, in Figura 6, si riporta lo schema del componente sintetizzato.

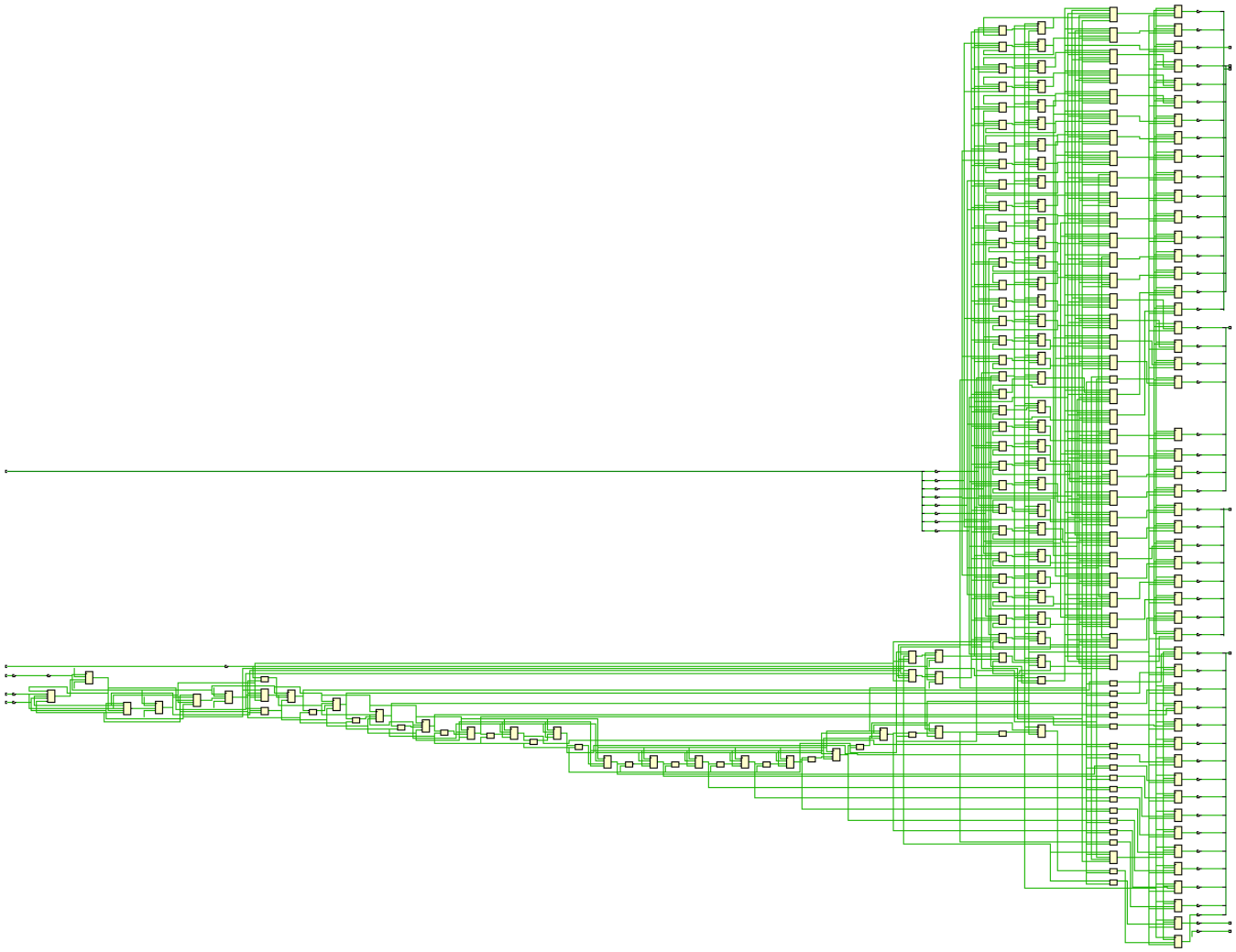

Figura 6

Simulazioni

Una volta passati i test bench forniti dai docenti sia in pre che post sintesi, abbiamo sottoposto il nostro elaborato a ulteriori prove da noi prodotte al fine di esaminare il comportamento del componente nei casi limite.

Di seguito i casi critici testati:

1. Indirizzo
    1.1. Indirizzo completo: verifica che il componente riconosca l'indirizzo massimo, ovvero quello composto da sedici bit 1, e l'indirizzo minimo, ovvero quello composto da sedici bit 0. Di seguito in figura 7 la waveform prodotta da questo test;

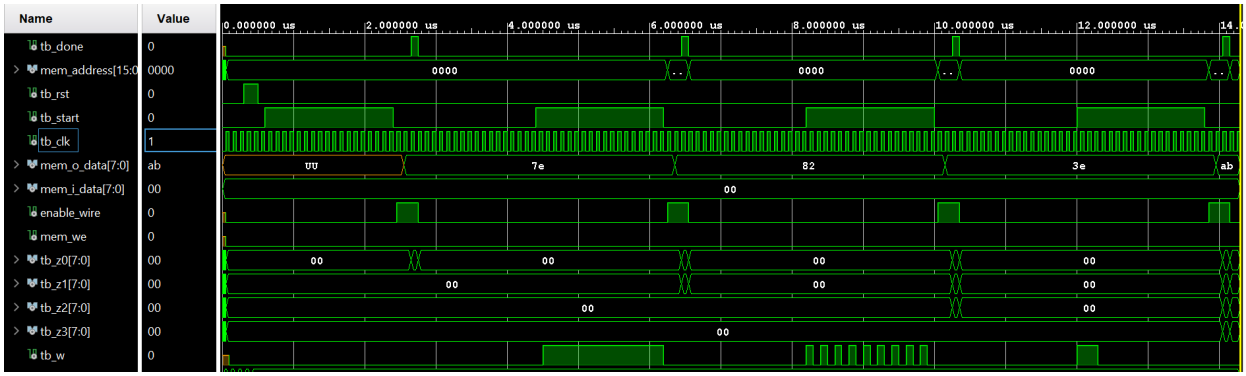

Figura 7

1.2. Indirizzo di lunghezza variabile: verifica che il componente riconosca indirizzi di varia lunghezza, tra cui anche l'indirizzo vuoto. Con indirizzo vuoto intendiamo l'indirizzo ottenuto tenendo il segnale di start attivo per soli due cicli di clock. A seguire la waveform prodotta da questo test;

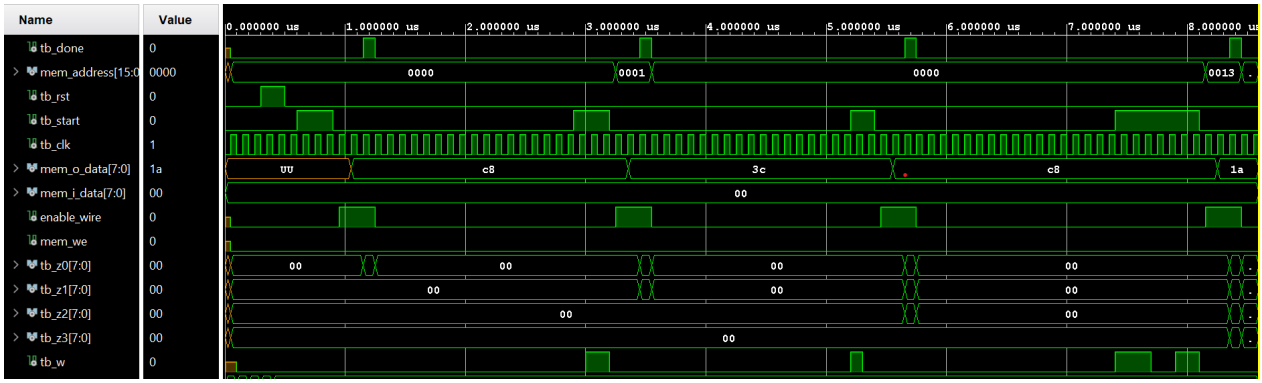

Figura 8

2. Uscita
    2.1. Due volte di seguito lo stesso indirizzo: verifica che a fronte di uno stesso indirizzo il componente imposti il segnale di "o_done" a "1" e ci mostri il contenuto dei canali di uscita z0, z1, z2, z3. A seguire in figura 9 un dettaglio della waveform pensata per questo caso limite: viene mandato in input due volte uno stesso indirizzo, la memoria restituisce quindi due volte 31, ma su due canali diversi (prima z3 e poi anche z2);

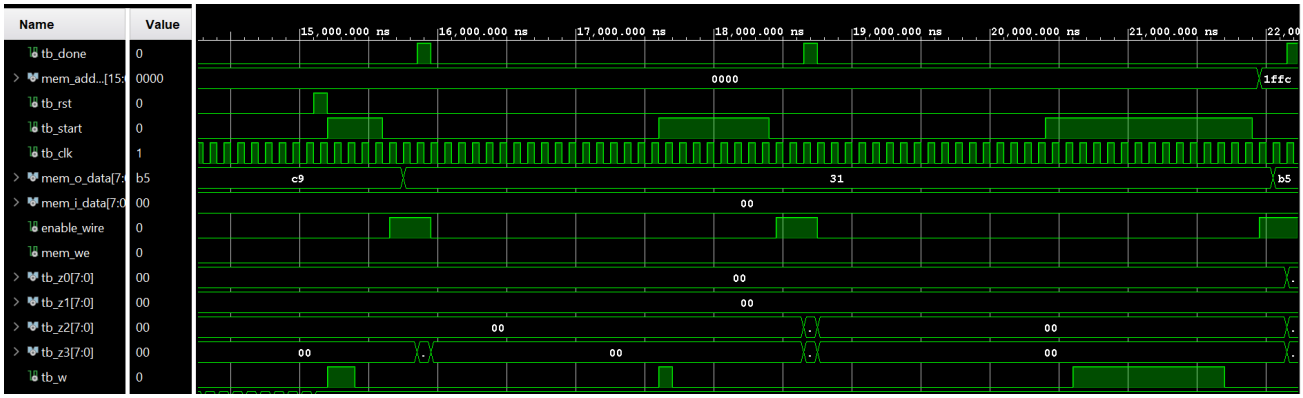

Figura 9

2.2. Sovra scrizione dei canali di uscita: verifica che se si scrive due volte su uno stesso canale il componente riesca a leggere e mostrare in uscita correttamente il valore più recente. Di seguito la waveform da cui si può notare che a partire dal quinto segnale di start attivo in poi il modulo è costretto a sovrascrivere uno dei canali e non risultano problemi;

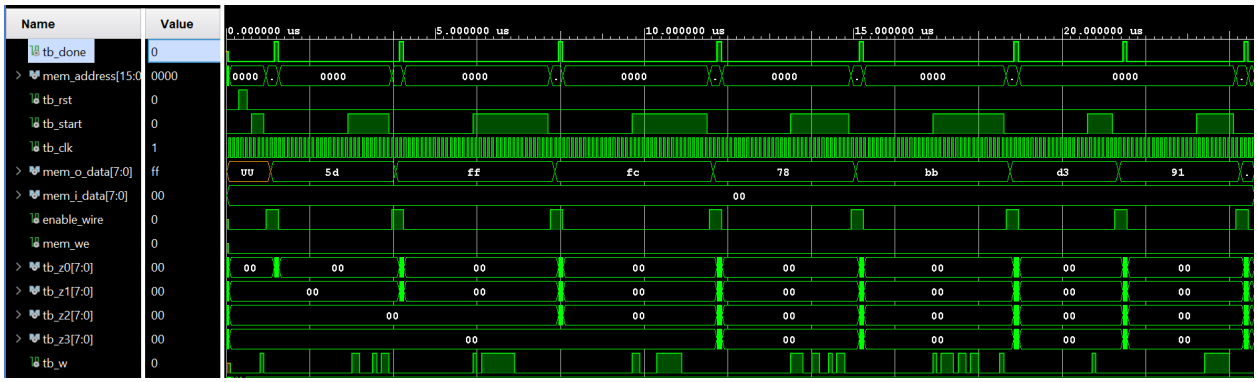

Figura 10

3. Reset

3.1. Reset dopo l'attivazione del segnale "o_done": verifica che, ricevuto un segnale di reset subito dopo il segnale di "o_done", tutti i valori siano stati azzerati. A seguire la waveform prodotta da questo test in figura 11;

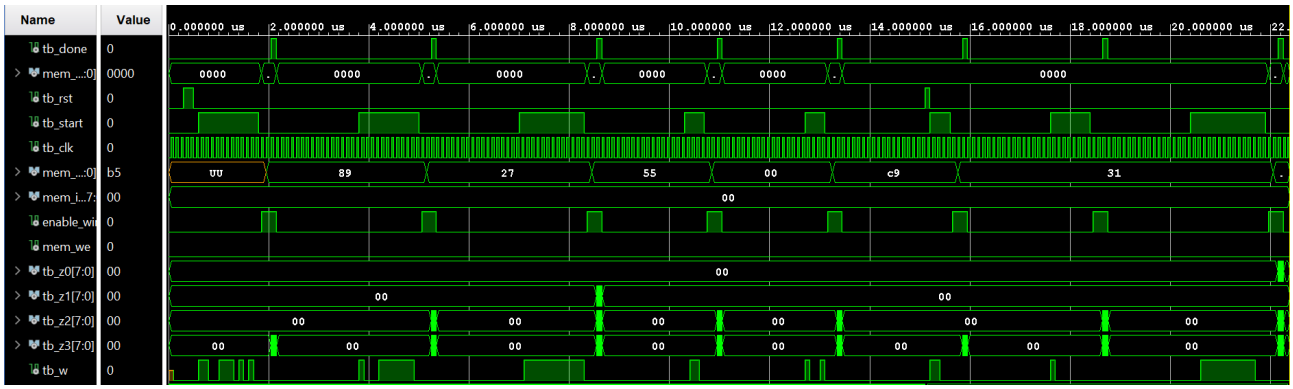

Figura 11

3.2. Reset durante elaborazione: verifica che il componente sia in grado, una volta ricevuto il valore dalla memoria, di mandare questo in uscita nonostante il segnale di reset un attimo prima dell'attivazione del segnale "o_done" . A seguire la waveform prodotta da questo test;

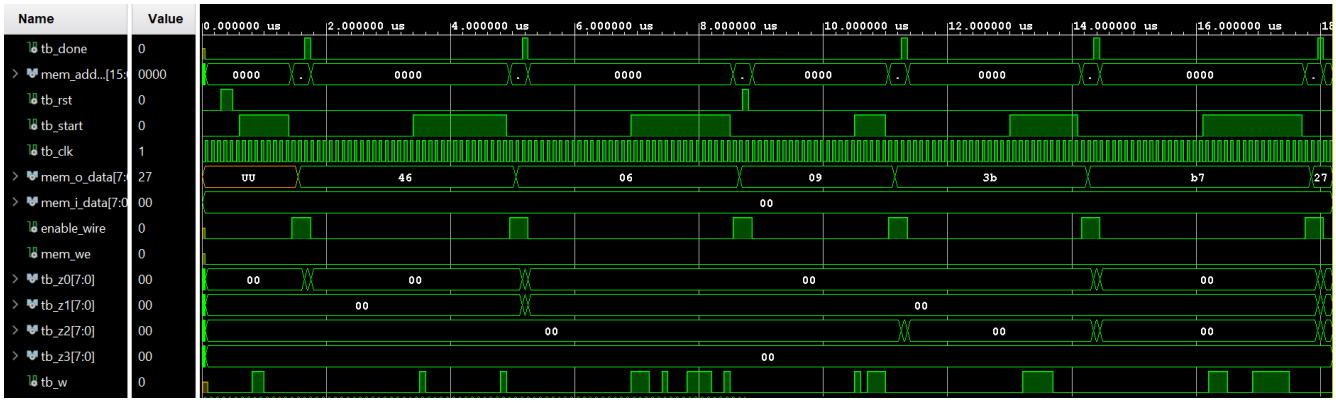

Figura 12

3.3. Reset quando il segnale “`i_start`” è attivo.
Questo test mira a verificare che il componente sia in grado di ricevere un segnale di reset quando è attivo il segnale di start, azzerare tutto e andare avanti. A seguire in figura 13 la waveform prodotta da questo caso di test;

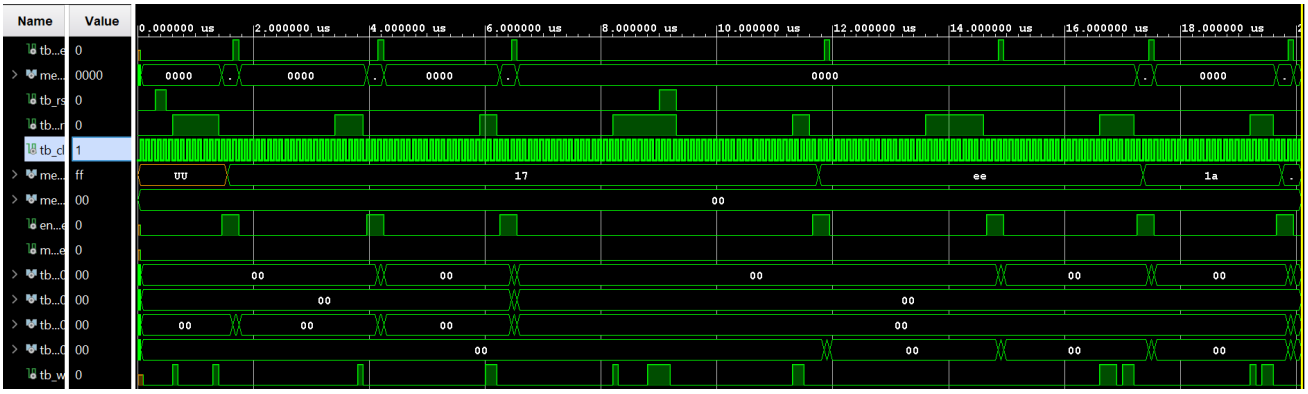

Figura 13

Inoltre, abbiamo sottoposto il nostro lavoro a una serie di test prodotti da un generatore casuale di input e indirizzi per assicurarcene la correttezza.

Conclusione

Dopo aver svolto vari test e analizzato a fondo i casi limite si giunge alla conclusione che il componente risponde correttamente alle specifiche di progetto.
Abbiamo fatto in modo poi che il modulo restituisse il contenuto della memoria il prima possibile, ovvero dopo soli due cicli di clock a prescindere dall’indirizzo scelto.
Inoltre, come si evince dal report di utilizzo esso sfrutta solamente Flip-Flop e Look Up Table in numero considerevolmente ridotto rispetto alle risorse disponibili, evitando l’utilizzo di Latch che possono creare problemi nella fase di sintesi. Anche dal punto di vista tempistico il componente ha una buona risposta in quanto si avvale di solo una piccola parte del tempo imposto dal constraint.
In conclusione, il componente realizzato si comporta come una sorta di puntatore. Esso riceve un canale di uscita e un indirizzo, completa quest’ultimo con i bit mancanti e accede alla memoria. Una volta ricavato il valore contenuto in questa, per un solo ciclo di clock mostra il contenuto di tutti i canali di uscita.