POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Prova finale
Progetto di Reti Logiche

Francesco Ferlin

Codice persona: <redacted>

Matricola: <redacted>

Giacomo Orsenigo

Codice persona: <redacted>

Matricola: <redacted>

Corso di Reti Logiche

Anno accademico 2022/2023

Professore: Fabio Salice

Indice

1 Introduzione

1.1 Scopo

Lo scopo del progetto è quello di descrivere in VHDL e sintetizzare un componente hardware che si interfacci con una memoria. Il componente, tramite un ingresso seriale, riceve l'indirizzo di una locazione di memoria il cui contenuto dovrà essere indirizzato nel canale di uscita corretto. Il componente dovrà quindi innanzitutto separare i bit di uscita da quelli di indirizzo, dovrà poi convertire l'indirizzo da codifica seriale a parallela per interfacciarsi con la memoria e, infine, selezionare l'uscita corretta.

1.2 Interfaccia del componente

Il componente ha due ingressi a un bit (START e W), 4 uscite a 8 bit (da Z0 a Z3), e un'uscita a un bit (DONE). Sono presenti, inoltre, un segnale di start e uno di reset.

L'interfaccia del componente è quindi la seguente:

```
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;
```

Il segnale di reset viene posto a 1 per inizializzare il modulo prima di ricevere il primo segnale di START. Inoltre, ogni volta che il segnale di reset viene posto a 1, il modulo viene re-inizializzato.

1.3 Descrizione della memoria

Il componente sviluppato deve comunicare con una memoria. La memoria è descritta all'interno dei test bench ed è di tipo sincrono. Per leggere un dato è quindi necessario comunicare l'indirizzo tramite l'uscita `o_mem_addr` e porre a 1 l'uscita `o_mem_en`. Il dato richiesto sarà disponibile al ciclo di clock successivo sull'ingresso `i_mem_data`. L'uscita `o_mem_we` dovrà essere a 0 durante le letture e a 1 per effettuare delle scritture.

1.4 Codifica del segnale W

Quando il segnale `i_start` va a 1, inizia la lettura dell'ingresso seriale W. I primi due bit letti specificano il canale di uscita (da 00 a 11), mentre i bit successivi rappresentano l'indirizzo della memoria. I bit vengono letti a partire da quello più significativo. La memoria è indirizzata a 16 bit; nel caso in cui il numero di bit letti sia inferiore a 16, l'indirizzo viene esteso aggiungendo zeri prima del bit più significativo. La lunghezza del segnale W va quindi da un minimo di 2 bit a un massimo di 18, durante i quali il segnale `i_start` resta a 1.

Tutte le letture sono effettuate sul fronte di salita del clock.

1.4.1 Esempio

Se il segnale `i_start` resta a 1 per 7 cicli di clock e viene letto W = 0111000:

- I primi due bit (01) rappresentano il canale di uscita, che sarà quindi Z1.
- I 5 rimanenti (11000) sono i bit meno significativi dell'indirizzo di memoria. L'indirizzo completo a 16 bit sarà quindi 0000000000011000.

1.5 Le uscite

Tutte le uscite sono inizialmente a 0. Dopo che il segnale `i_start` è tornato a 0, sono disponibili 20 cicli di clock per prelevare il dato dalla memoria e indirizzarlo verso l'uscita corretta. Quando l'elaborazione è terminata, l'uscita `o_done` viene posta a 1 e contemporaneamente viene scritto il valore sul canale di uscita. Il segnale `o_done` resta a 1 per un solo ciclo di clock. I dati rimangono memorizzati finché non vengono sovrascritti con dei nuovi valori, ma sono visibili solo quando il segnale `o_done` è a 1. Quando `o_done` è a 0, tutte le uscite sono a 0.

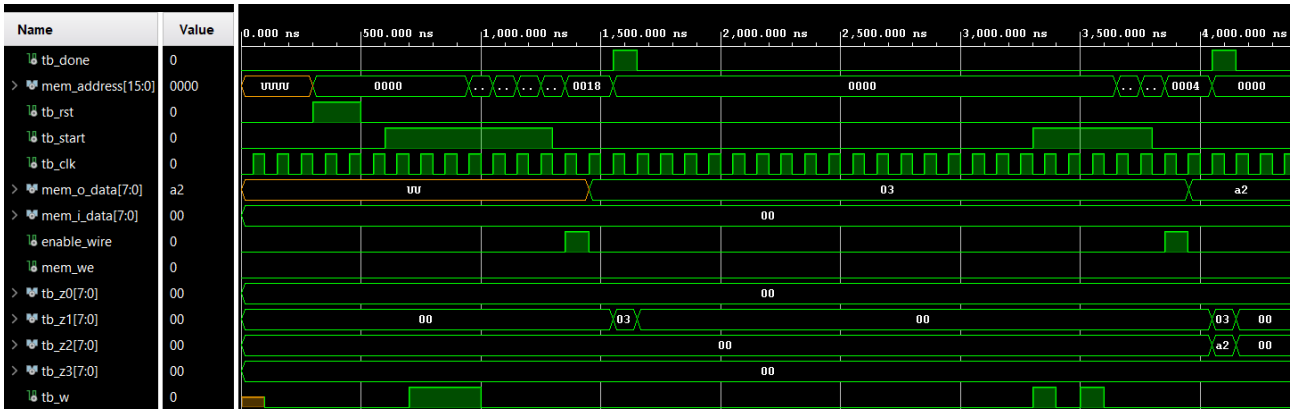
1.5.1 Esempio

Consideriamo la seguente memoria con i valori rappresentati in esadecimale:

…	
0000000000000100	a2
0000000000011000	03
…	

Ipotizziamo che il segnale `i_start` resti a 1 per 7 cicli di clock e venga letto W = 0111000. Entro i 20 cicli di clock successivi il segnale `o_done` viene posto a 1 e il valore "03", prelevato dalla memoria, viene scritto sul canale Z1. Il ciclo di clock successivo `o_done` e Z1 tornano a 0.

Successivamente il segnale `i_start` resta a 1 per 5 cicli di clock e viene letto W = 10100. Terminata l'elaborazione, il valore "a2", prelevato dalla memoria, viene scritto sul canale Z2 e il segnale `o_done` viene posto a 1 per un ciclo di clock. Mentre il segnale `o_done` è a 1, entrambi i valori sono visibili sui rispettivi canali di uscita.

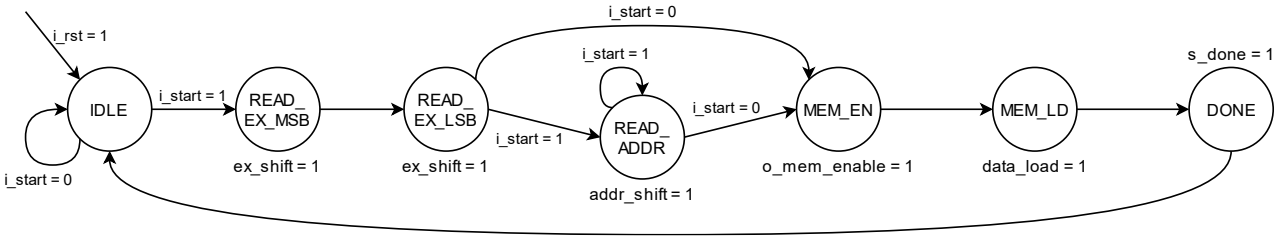


2 Architettura

Il componente è stato realizzato separando il datapath dalla macchina a stati finiti che lo controlla.

2.1 Macchina a stati finiti

La macchina a stati finiti genera i segnali che controllano il datapath. La rappresentazione della macchina è la seguente:



Per evitare di appesantire troppo il disegno, sono state omesse tutte le frecce di reset: da qualsiasi stato, in caso si riceva un segnale di reset, si ritorna allo stato IDLE. Inoltre, negli stati in cui non sono esplicitamente specificati, i segnali di controllo sono a 0.

Gli stati della macchina sono i seguenti:

2.1.1 IDLE

Viene raggiunto in seguito a un segnale di reset. La macchina viene inizializzata (tutti i segnali sono portati a 0) ed è pronta a ricevere il segnale `i_start`.

2.1.2 READ_EX_MSB

È stato ricevuto il primo bit di `i_start` a 1. Sull'ingresso `i_w` viene letto il bit più significativo del numero dell'uscita. Durante questo ciclo e il successivo, viene posto a 1 il segnale `ex_shift`, per attivare lo shift-register descritto in seguito.

2.1.3 READ EX_LSB

È stato ricevuto il secondo bit di `i_start` a 1. Sull'ingresso `i_w` viene letto il bit meno significativo del numero dell'uscita.

2.1.4 READ_ADDR

Si raggiunge questo stato al terzo bit di `i_start` a 1, e ci si rimane finché `i_start` non torna a 0. In questo stato vengono ricevuti su `i_w` i bit dell'indirizzo di memoria. Viene quindi posto a 1 il segnale `addr_shift`, per attivare lo shift-register descritto in seguito. Nel caso limite in cui i bit di indirizzo non siano presenti, questo stato viene saltato e si passa direttamente al successivo.

2.1.5 MEM_EN

Si raggiunge questo stato non appena `i_start` torna a 0. A questo punto l'indirizzo di memoria è stato correttamente ricevuto e viene attivato il segnale `o_mem_en` per effettuare la lettura.

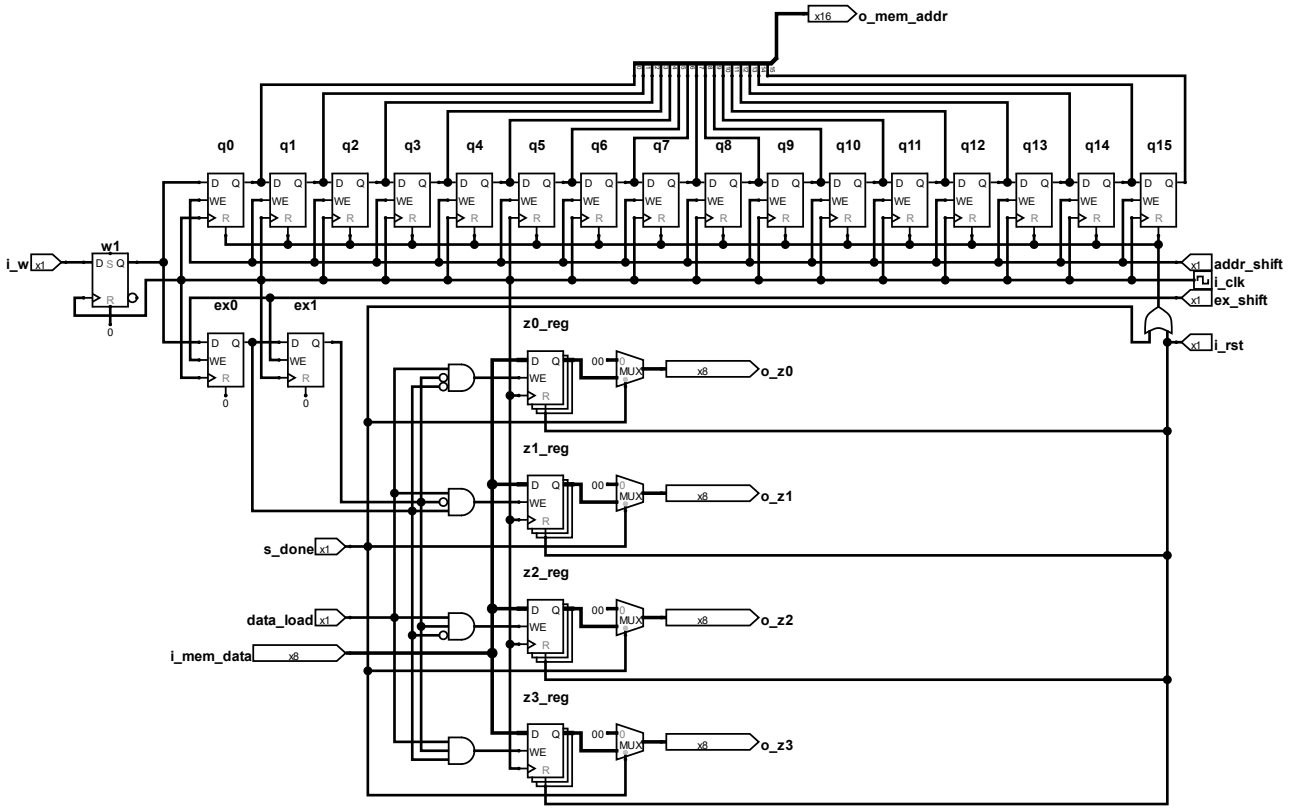
2.1.6 MEM_LD

Viene ricevuto dalla memoria il valore richiesto sull'ingresso `i_mem_data`. Portando a 1 il segnale `data_load`, viene scritto il valore letto nel registro corrispondente all'uscita precedentemente selezionata.

2.1.7 DONE

Il valore letto è pronto per essere visualizzato sul canale di uscita. Il segnale `o_done` viene portato a 1 e sui canali di uscita sono visibili i valori dei rispettivi registri. Successivamente si ritorna allo stato di IDLE.

2.2 Datapath



Il datapath è formato da tre componenti principali:

2.2.1 serial_to_parallel_2

Il componente ha lo scopo di leggere i primi due bit dell'ingresso seriale `w`, che selezionano il canale di uscita.

È realizzato tramite uno shift-register, composto da due flip-flop collegati in cascata. Tutti i flip-flop sono edge-triggered e commutano sul fronte di salita del clock. Hanno un ingresso di write-enable collegato al segnale `ex_shift`, che viene posto a 1 dalla macchina a stati durante la lettura del canale di uscita. Non è presente un ingresso di reset. Non è necessario, infatti, azzerare i due flip-flop, in quanto vengono sovrascritti a ogni computazione.

L'uscita del componente è quindi un segnale a due bit, che identifica il canale di uscita scelto.

2.2.2 serial_to_parallel_16

Lo scopo del componente è di leggere l'indirizzo della cella di memoria dall'ingresso seriale `w` e trasformarlo in un segnale parallelo.

Il componente è realizzato tramite uno shift-register da 16 flip-flop. Come nel componente precedente, tutti i flip-flop sono comandati dal segnale di clock. Hanno un ingresso di write-enable collegato al segnale `addr_shift`, che viene posto a 1 dalla macchina a stati durante la lettura dell'indirizzo. È presente, infine, un segnale di reset, che azzera tutti i flip-flop. Questo segnale non è direttamente connesso al reset proveniente dall'esterno, ma è posto in OR con il segnale `s_done`, in modo da resettare lo shifter dopo ogni computazione.

L'uscita del componente è quindi un segnale a 16 bit formato concatenando le uscite dei singoli flip-flop. Il bit meno significativo è l'uscita del primo flip-flop, mentre quello più significativo è l'uscita dell'ultimo flip-flop. In questo modo, nel caso il segnale `i_w` sia più corto di 18 bit, i bit più significativi dell'uscita rimangono a 0. L'uscita è connessa al segnale `o_mem_addr`.

2.2.3 out_reg

Il componente rappresenta un registro di uscita ed è istanziato quattro volte, una per ogni canale. Permette di memorizzare i dati inviati ai canali di uscita, per mantenerli anche nelle computazioni successive.

È formato da un registro a 8 bit e un mux. Il registro è edge-triggered ed ha un ingresso di write-enable che permette di abilitare la scrittura di un nuovo dato. A questo ingresso è connesso un segnale che, tramite un'opportuna porta AND mostrata in figura, quando il segnale `data_load` è a 1, attiva il registro relativo al canale di uscita scelto. È presente, inoltre, un segnale di reset, direttamente connesso al reset proveniente dall'esterno, che azzera il registro.

L'uscita del registro non è sempre visibile, ma è mascherata da un mux comandato dal segnale `s_done`. Solo quando il segnale `s_done` è a 1, il mux manderà in uscita il contenuto del registro, in alternativa manderà 8 bit a 0.

Collegati questi componenti, viene posto poi un flip flop all'ingresso di `i_w`, con lo scopo di leggerlo esattamente sul fronte di salita e ritardarlo di un ciclo di clock. Questo serve per dare il tempo alla macchina a stati finiti di comandare gli shift-register, che andranno quindi a leggere il segnale sul fronte di clock successivo.

3 Risultati sperimentali

3.1 Sintesi

3.1.1 Report Utilization

Analizzando il *Report Utilization* è possibile verificare che i componenti inseriti dal tool di sintesi siano quelli previsti.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	26	0	0	134600	0.02
LUT as Logic	26	0	0	134600	0.02
LUT as Memory	0	0	0	46200	0.00
Slice Registers	54	0	0	269200	0.02
Register as Flip Flop	54	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Si può notare infatti che il numero di flip-flop è coerente con quanto presente nel datapath, e che non sono stati inferiti dei latch, che avrebbero potuto causare problemi nelle simulazioni post-sintesi.

$$
\begin{aligned}
&+ 16 && \textit{per lo shift register da } 16 \\
&+ 2 && \textit{per lo shift register da } 2 \\
&+ 1 && \textit{per ritardare il segnale } w \\
&+ 8 * 4 && \textit{per i registri di uscita} \\
&+ \log_2 7 && \textit{per gli stati della FSM} \\
&= 54\ \textit{flip flop}
\end{aligned}
$$

3.1.2 Report Timing

Dal *Report Timing* invece, è possibile verificare che il componente sviluppato rispetti i requisiti di timing previsti.

In particolare, lo Slack Time è di 97.009ns e il Data Path Delay di 2.402ns, ampiamente inferiore al periodo di clock richiesto (100ns).

3.2 Simulazioni

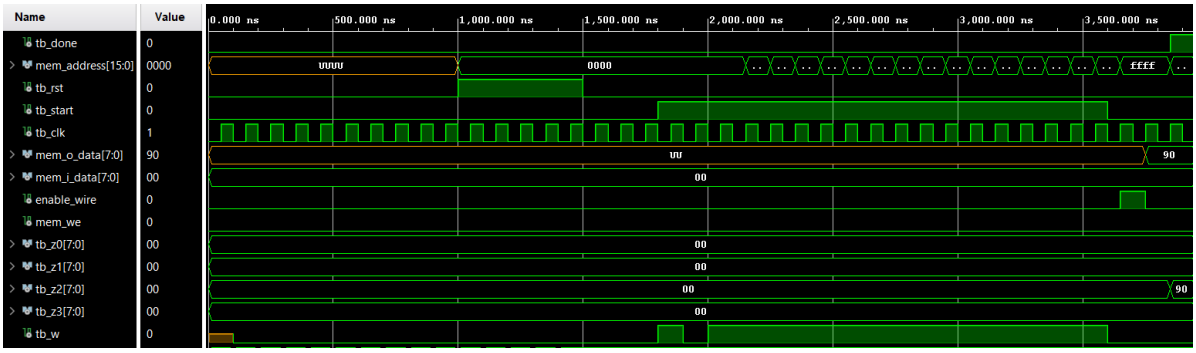
Per guidare lo sviluppo del componente e il suo corretto funzionamento, oltre ai test bench forniti come esempio, sono stati introdotti incrementalmente ulteriori test che verificassero casi limite.

Sono stati inoltre utilizzati vunit, un framework di behavioral testing automatizzato, e in seguito uno script Vivado in TCL per post synthesys testing, che hanno permesso di eseguire in ogni momento tutti i test raccolti in batch, potendo così individuare possibili regressioni durante lo sviluppo.

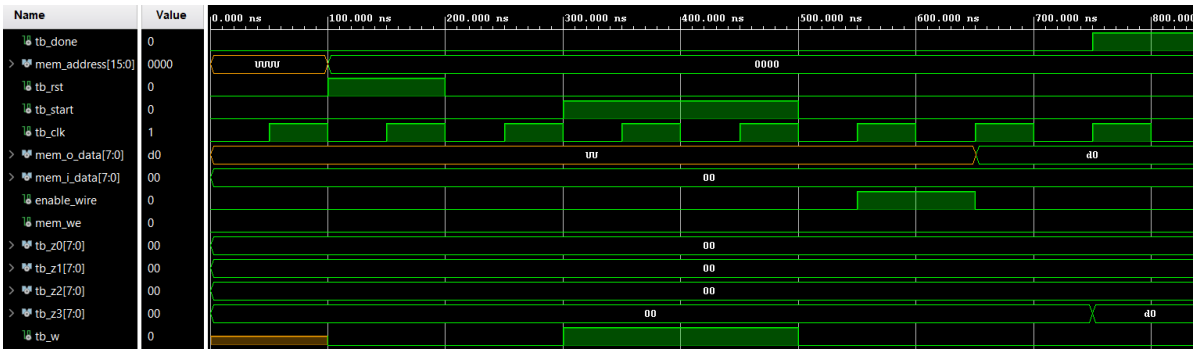
Si fornisce di seguito una breve descrizione dei test più significativi, con relativo screenshot per mostrare l'effettivo funzionamento.

Per la verifica dei corner case, sono stati individuati 3 test:

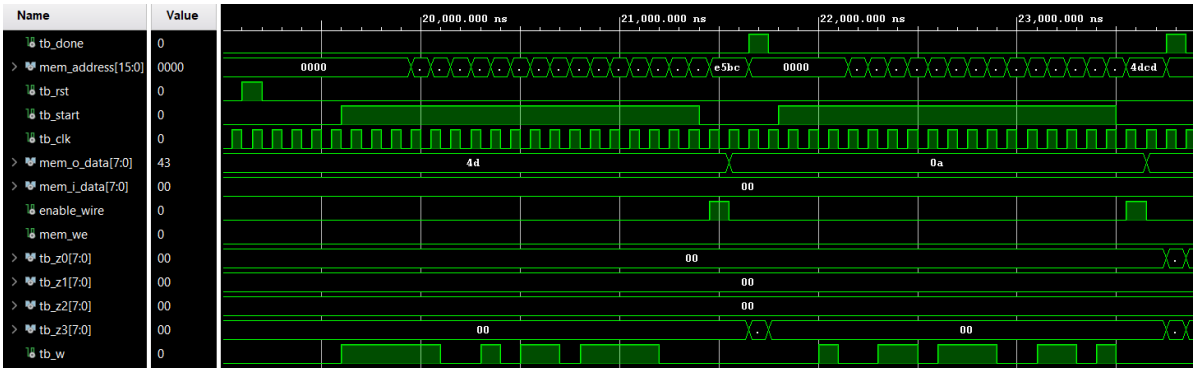
1. **Indirizzo in ingresso di lunghezza massima (1111111111111111)**: fornisce i due bit di indirizzo e sedici 1 consecutivi, verificando che anche nel caso di indirizzo più lungo la macchina rimanga nei cicli di computazione ad essa assegnati



2. **Indirizzo in ingresso di lunghezza minima (0000000000000000)**: fornisce solo i due bit dell'uscita, senza fornire alcun bit di indirizzo, verificando quindi che la macchina aggiunga correttamente il padding all'indirizzo

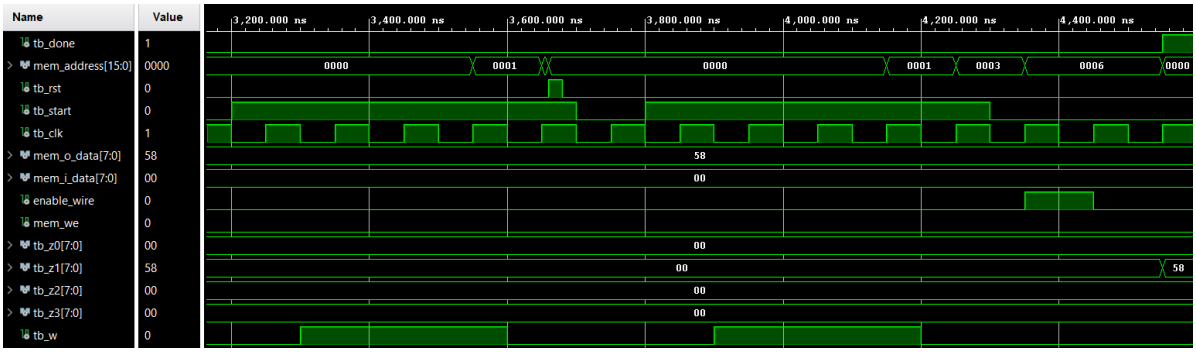


3. **Utilizzo del componente back-to-back**: fornisce numerosi messaggi uno dopo l'altro, senza aspettare che siano scaduti i 20 cicli di computazione, mandando invece il successivo messaggio appena viene abbassato o_done

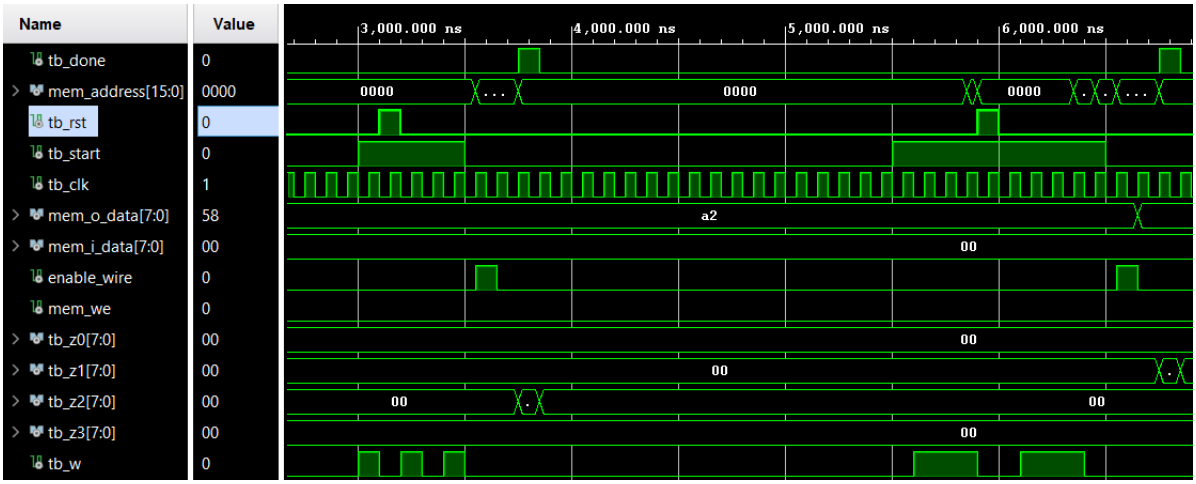


Sono poi stati utilizzati numerosi test per verificare il funzionamento corretto dopo un reset della macchina:

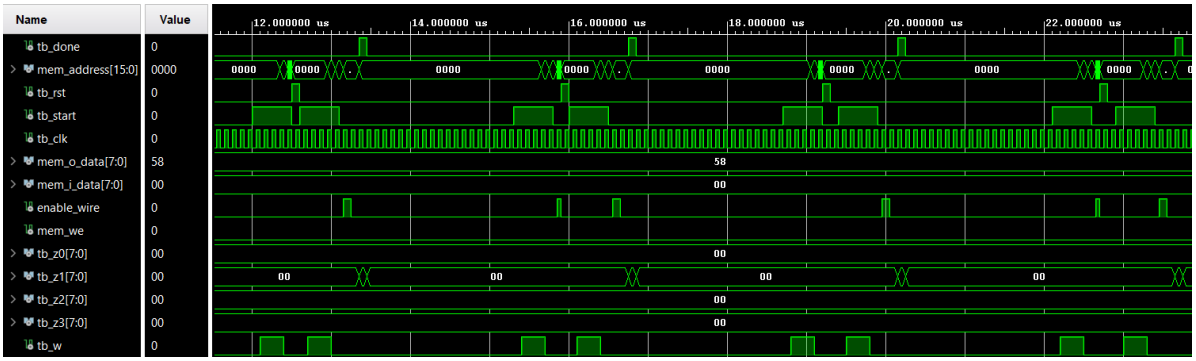
1. **Reset asincrono**: viene fornito un segnale di reset completamente asincrono per verificare che questo non comprometta il funzionamento della macchina



2. **Reset durante i bit di indirizzo**: viene fornito un segnale di reset mentre vengono letti i bit di uscita/indirizzo su w, verificando che la macchina sia in grado di riprendere a funzionare correttamente



3. **Reset durante i bit di computazione**: viene fornito un segnale di reset nel periodo in cui si aspetta che la macchina esegua la computazione, verificando che la macchina sia in grado di riprendere a funzionare correttamente



Infine, sono stati realizzati mediante uno script in Python degli stress test volti a individuare eventuali situazioni non previste. Lo script genera randomicamente i bit di uscita, gli indirizzi di memoria, il contenuto della memoria a tali indirizzi e i segnali di reset, utilizzando un elevato numero di iterazioni.

4 Conclusioni

Riteniamo che l'architettura progettata rispetti le specifiche date, cosa verificata dall'estensivo stress testing effettuato. Risulta inoltre ottimizzata anche dal punto di vista temporale, in quanto necessita di soli 4 dei 20 cicli di computazione e, come evidenziato dal report timing, ha uno slack time di molto inferiore ai 100ns richiesti.

È stata particolarmente utile la divisione in datapath e macchina a stati finiti: avere chiaro lo schema di funzionamento dell'architettura ha di molto facilitato la stesura del codice VHDL, dando anche una certa sicurezza che questo fosse sintetizzabile.

Infine, il regression testing automatizzato ci ha permesso di effettuare numerose modifiche e avere subito un feedback sul funzionamento di quanto cambiato, dandoci la certezza di non aver introdotto bug non previsti o già risolti in precedenza.