Prova finale di Reti Logiche

Prof. Gianluca Palermo – AA 2021/22

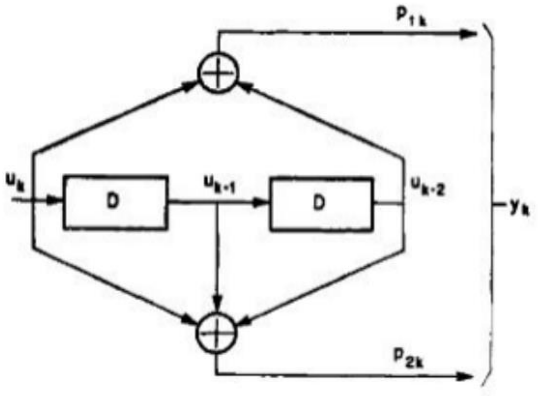
Michael Vitali

Sommario

Introduzione

Specifica del progetto

Lo scopo di questo progetto è l'implementazione in linguaggio VHDL di un componente hardware che si interfaccia con una memoria. Il componente riceve in ingresso un numero W di parole da 8 bit ciascuna e restituisce una sequenza di Z parole da 8 bit. Ognuna delle parole in ingresso viene serializzata; in questo modo viene generato un flusso continuo U da 1 bit. Su questo flusso viene applicato il convoluzionale ½ secondo lo schema in figura. Eseguendo questa operazione viene generato in uscita un flusso continuo Y. Tale flusso è ottenuto concatenando i due bit in uscita del convolutore.



Un esempio di funzionamento è il seguente dove il primo bit a sinistra è il primo da processare:

- Byte in ingresso: 10100011 (163)

Applicando l'algoritmo convoluzionale si ottiene la seguente serie di coppie in uscita:

T	0	1	2	3	4	5	6	7
Uk	1	0	1	0	0	0	1	1
P1k	1	0	0	0	1	0	1	1
P2k	1	1	0	1	1	0	1	0

Quindi dopo la concatenazione di Pk1 e Pk2 abbiamo in uscita i seguenti 2 Byte:

1) B1 = 11010001
2) B2 = 11001110

Quindi notiamo come il numero di Byte in uscita è il doppio delle parole in ingresso. Quindi la lunghezza del flusso U = 8*W, mentre la lunghezza del flusso Y = 2*U = 2*8*W.

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_data : in std_logic_vector(7 downto 0);
        o_address : out std_logic_vector(15 downto 0);
        o_done : out std_logic;
        o_en : out std_logic;
```

```
        o_we : out std_logic;
        o_data : out std_logic_vector (7 downto 0)
);
end project_reti_logiche;
```

Dove:

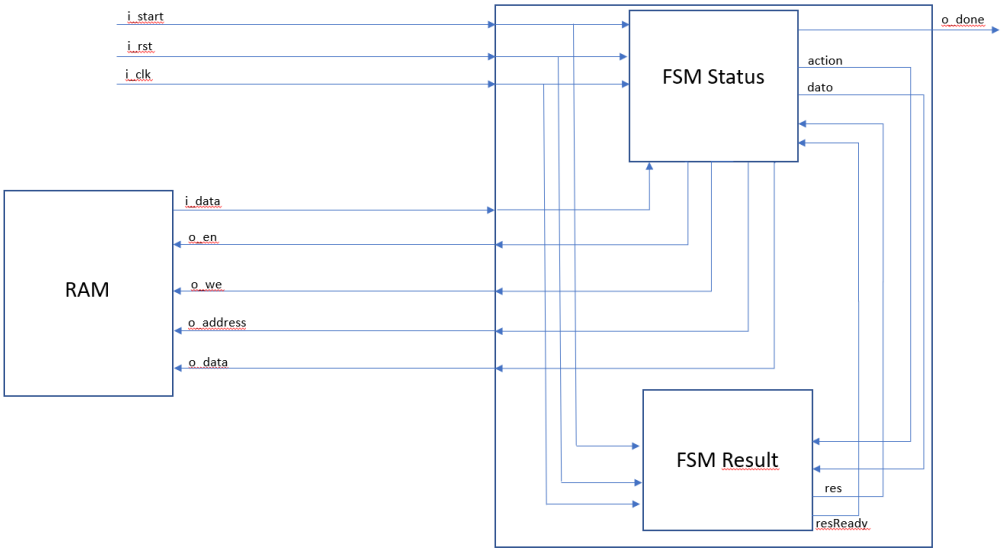
- **i_clk** rappresenta il segnale di **clock** in ingresso generato dal TestBench
- **i_rst** rappresenta il segnale di **reset** che inizializza la macchina pronta per ricevere il primo segnale di start.
- **i_start** rappresenta il segnale di **start** generato dal TestBench.
- **i_data** rappresenta il segnale contenente la parola che arriva dalla memoria in seguito ad una richiesta di lettura.
- **o_address** rappresenta il segnale di uscita che manda l'indirizzo alla memoria.
- **o_done** rappresenta il segnale di uscita che comunica la fine dell'elaborazione.
- **o_en** rappresenta il segnale di **enable** da dover mandare alla memoria per poter accedere ad essa. Deve essere mandato sia in caso di lettura che di scrittura.
- **o_we** rappresenta il segnale di **write enable** da dover mandare alla memoria. Nel caso in cui il suo valore è 1 scrivo nella memoria, altrimenti quando è 0 leggo dalla memoria.
- **o_data** rappresenta il segnale di uscita dal componente verso la memoria e contiene il dato che deve essere scritto in essa.

Dati e memoria

La memoria è istanziata all'interno del TestBench nel seguente modo:

- La quantità di parole W da codificare è memorizzata sempre nell'indirizzo 0. Il valore massimo che essa può assumere è 255 mentre il minimo è 0.
- Il primo byte della sequenza di W parole è sempre memorizzato all'indirizzo 1.
- Lo stream di uscita Z deve sempre essere memorizzato a partire dall'indirizzo 1000.

Architettura del componente



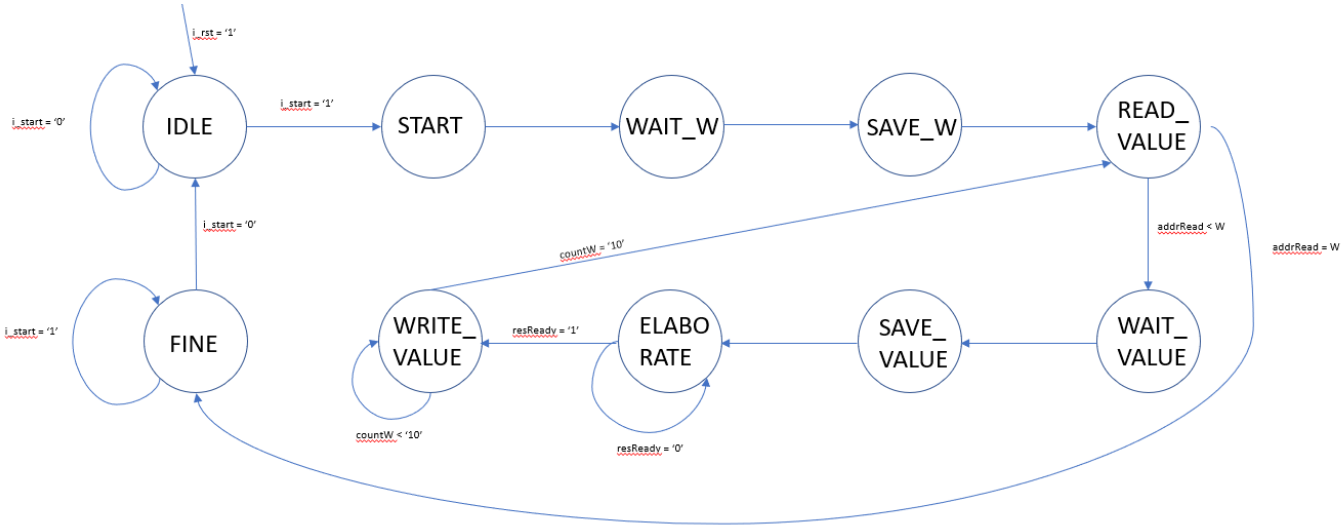
Il componente progettato ha al suo interno 2 moduli separati. Il primo modulo è una macchina a stati finiti che gestisce tutte le operazioni che il componente deve eseguire in modo sequenziale. Il secondo modulo è una macchina a stati finiti che riceve in ingresso il dato da elaborare, lo elabora e restituisce il relativo risultato.

Entrambi i moduli sono sincronizzati con il segnale di clock in ingresso e sono collegati tra di loro mediante 4 segnali interni. Il primo modulo passa al secondo modulo il dato che deve essere elaborato e segnala l'avvio dell'elaborazione tramite il segnale Action portandolo allo stato ELABORATE. Una volta ricevuto questo segnale il secondo modulo elabora il dato in ingresso ricevuto. Dopo che l'elaborazione è terminata il secondo modulo restituisce il risultato (res) e segnala al primo modulo che è terminata mediante il segnale resReady.

Il primo modulo si occupa anche di interfacciarsi con la memoria mettendo in uscita i segnali necessari per l'eventuale lettura o scrittura della stessa. Quando l'elaborazione di tutti i dati è terminata restituisce un segnale di done attendendo un nuovo segnale di start.

Il componente è progettato in modo da ricevere sempre un segnale di reset prima della prima lettura. Per le letture multiple attende solo che lo start sia messo ad 1, senza attendere un reset. Nel caso di reset durante l'elaborazione dei dati, il componente ricomincia l'elaborazione da capo non considerando i dati processati fino a quel momento.

Primo modulo – FSM Status



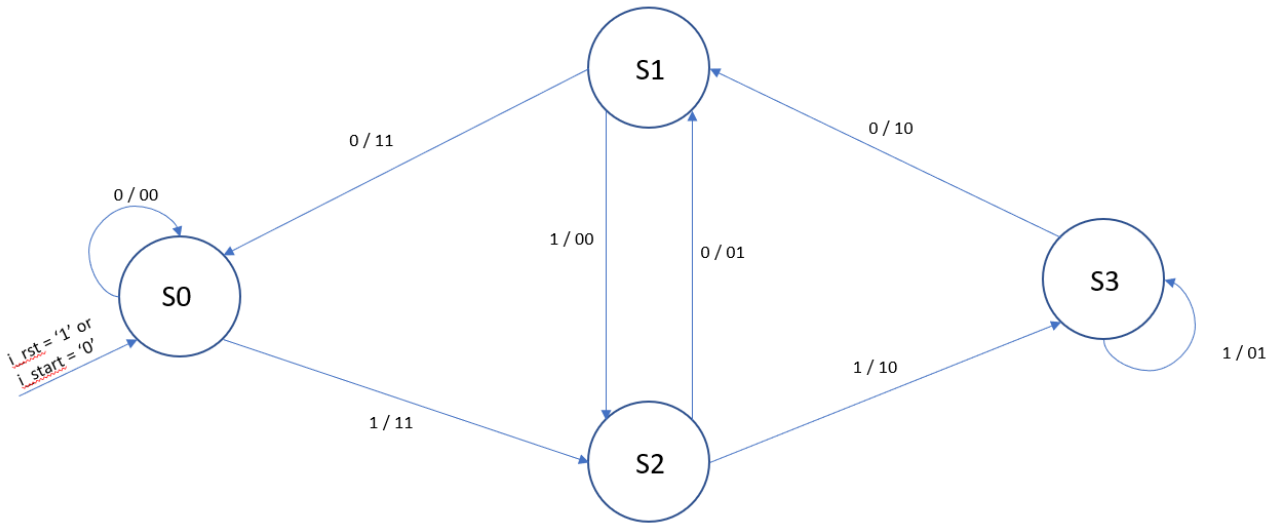
Il primo modulo si occupa di gestire le varie azioni (action) che il nostrocomponente deve svolgere; in particolare definisce gli indirizzi di memoria per le operazioni di scrittura e lettura, gestisce i segnali di write enable ed enable per abilitare la scrittura e la lettura dalla memoria e passa i dati necessari all'elaborazione al secondo modulo.

La macchina è composta da 10 stati. Di seguito una descrizione sintetica di questi ultimi:

- **IDLE:** stato nel quale il sistema si porta all'avvio del componente, quando viene chiamato il reset durante l'esecuzione oppure dopo che ha finito una elaborazione. Rimane in questo stato finché il segnale di start non diventa 1 e il reset vale 0. Imposta il valore dell'azione da eseguire a START così da iniziare la computazione dei dati.
- **START:** questo stato inizializza tutti i segnali interni per la gestione dell'elaborazione. Manda alla memoria l'indirizzo di memoria di W e abilita la lettura.
- **WAIT_W:** questo stato attende che il valore W restituito dalla memoria sia stabile. Disabilita la lettura della memoria.
- **SAVE_W:** questo stato salva il valore di W in input dalla memoria.

- **READ_VALUE:** questo stato si occupa della lettura delle W parole dalla memoria. Se le parole non sono ancora state lette tutte manda l'indirizzo relativo alla parola in memoria, abilita la memoria e la lettura. Se invece le parole da leggere sono terminate alza il segnale di done a 1 e si porta nello stato finale di attesa.
- **WAIT_VALUE:** questo stato attende che il segnale in input dalla memoria sia stabile.
- **SAVE_VALUE:** questo stato salva il dato in ingresso dalla memoria in una variabile interna e passa allo stato di elaborazione. Il segnale viene ricevuto anche dalla seconda macchina a stati che inizia ad elaborare il dato il ciclo di clock successivo.
- **ELABORATE:** in questo stato la macchina attende che la seconda macchina a stati abbia finito l'elaborazione. Dopo di che passa allo stato successivo di scrittura del risultato in memoria una volta ricevuto il segnale di resReady e il risultato dell'elaborazione.
- **WRITE_VALUE:** in questo stato la macchina si occupa di splittare il risultato prodotto dalla seconda macchina a stati e di salvarlo in memoria. Quindi manda alla memoria il relativo indirizzo di scrittura, abilita la scrittura e la memoria. Una volta terminata la scrittura del risultato torna a leggere il valore successivo passando allo stato di READ_VALUE.
- **FINE:** stato finale della macchina nel quale attende che il segnale di start venga portato a 0. Dopo di che pone il done a 0 e si porta allo stato IDLE dove attende un nuovo segnale di start.

Secondo modulo – FSM Result



Il secondo modulo si occupa di gestire l'elaborazione del dato ricevuto dal primo modulo per poi restituire il risultato elaborato che rappresenta rispettivamente la convoluzione ½ della parola. In questo caso la macchina a stati è formata da 4 stati e si attiva solo nel momento in cui il primo modulo segnala l'elaborazione del dato (action ELABORATE). Alla fine dell'elaborazione la macchina alza il segnale resReady, restituisce il risultato al primo modulo e attende il prossimo dato da elaborare.

La macchina si porta nello stato inziale in due occasioni:

1) Quando arriva in ingresso il segnale di reset alla macchina. Esso gestisce sia il caso di avvio del componente che il reset durante la computazione.
2) Quando la macchina finisce l'elaborazione del primo blocco di letture e si mette in attesa di una seconda computazione.

Gli altri stati sono auto-esplicativi e rappresentano i valori in uscita, che verranno salvati all'interno del risultato, in funzione di un determinato bit di input e il relativo stato prossimo nel quale si porta la macchina.

Risultati sperimentali

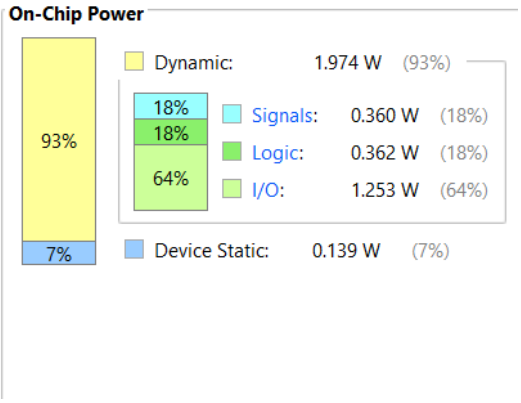
Risultati sintesi

Il componente è correttamente sintetizzabile e implementabile con un totale di 102 LTU e 123 FF. Il consumo totale di potenza è 2.113W.

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!									102	123	0	0	0
✓ impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA		NA	2.113	0	101	123	0	0	0

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**2.113 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**30,2°C**
Thermal Margin:	54,8°C (21,9 W)
Effective ϑJA:	2,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low



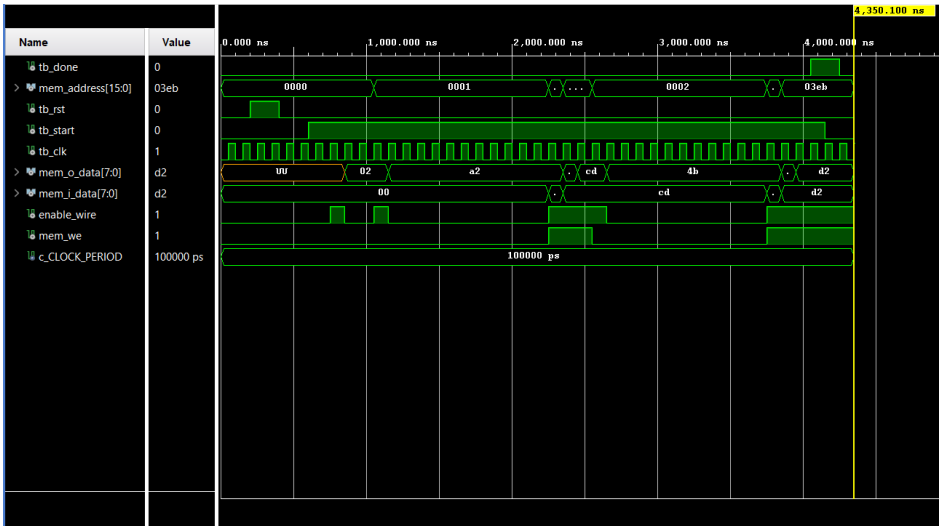
Simulazioni

Per quanto riguarda i test effettuati sul componente, si è deciso di testare sui seguenti casi particolari:

1) Caso generale di computazione
2) Multi start
3) Sequenza massima input, cioè W = 255
4) Sequenza minima, cioè W = 0
5) Multi reset

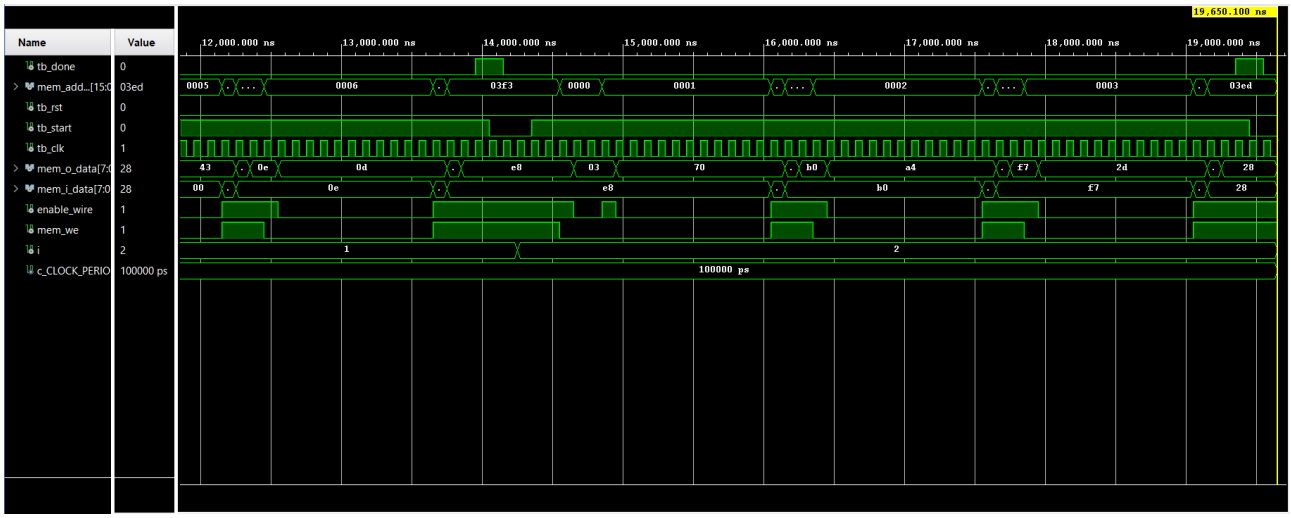
Inoltre, il componente è stato testato su un numero casuale di test e sono stati tutti superati.

- Caso generale fornito dal docente



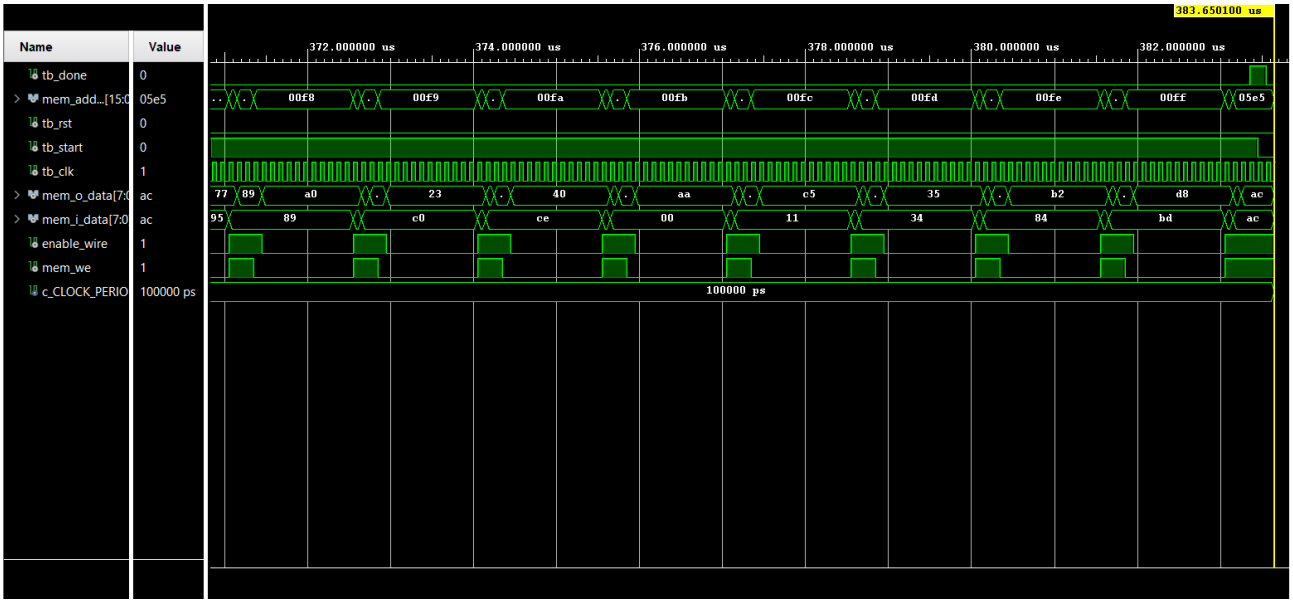
- Multi start

Questo test ha lo scopo di verificare il corretto funzionamento nel caso di più letture consecutive dalla memoria.



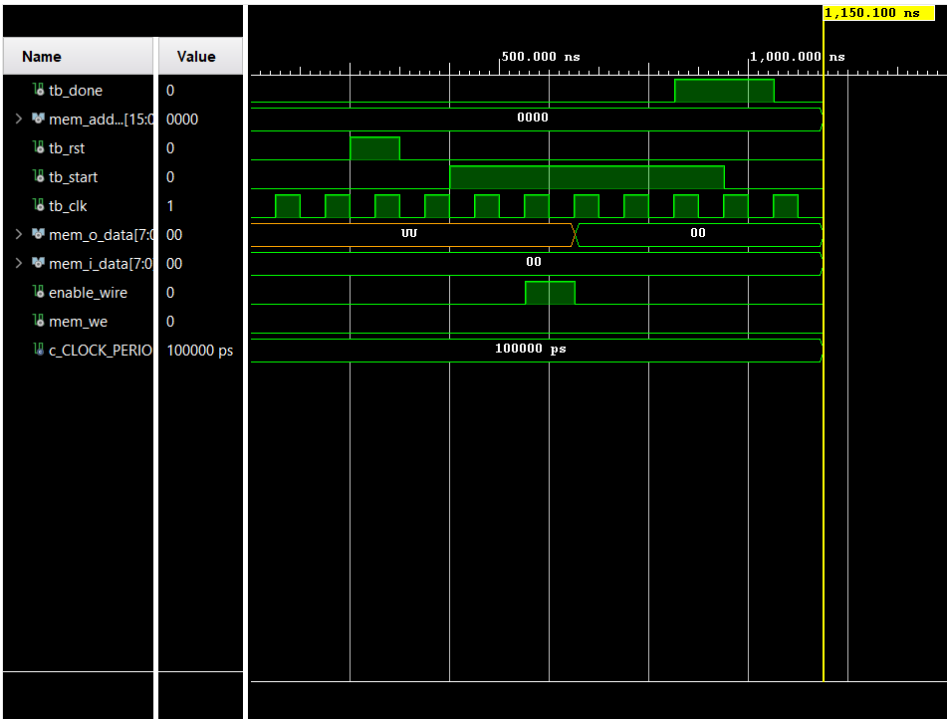
- Sequenza massima input

Questo test ha lo scopo di testare la specifica massima di 255 letture e ha lo scopo di evidenziare possibili errori quali insufficiente spazio allocato alle variabili di gestione della lettura.



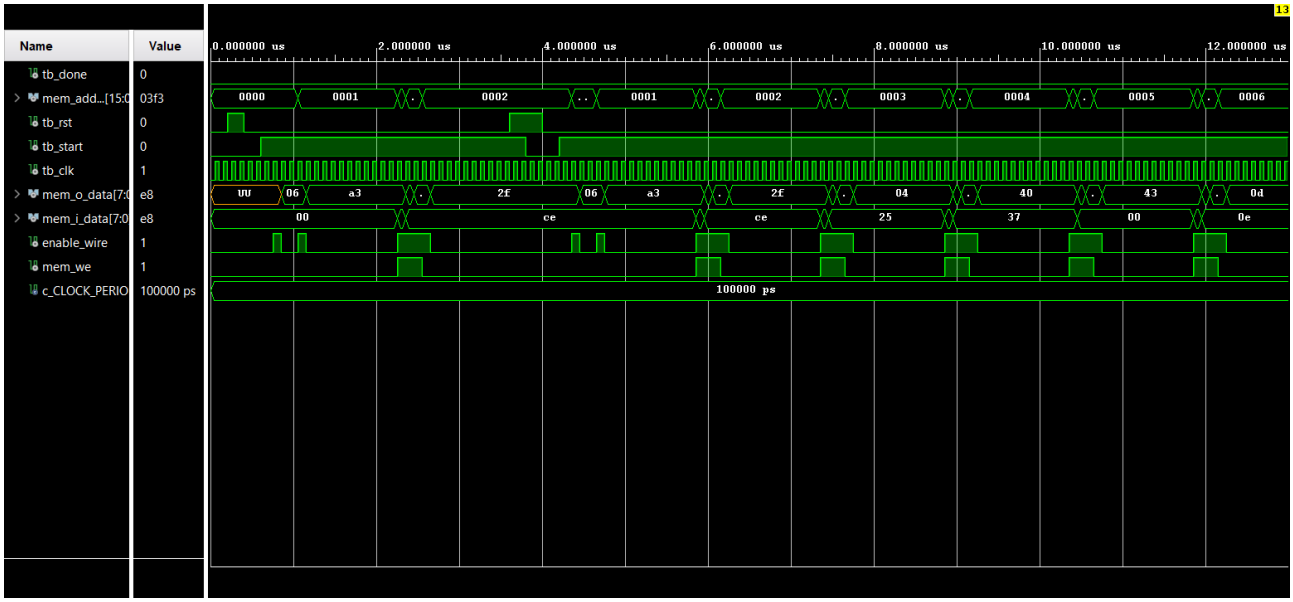
- Sequenza minima

Questo test ha lo scopo di verificare il funzionamento del componente nel caso in non ci sono valori in input da leggere. Quello che ci aspettiamo è che la computazione termini subito.



- Multi reset

Questo test ha lo scopo di testare l'arrivo di un reset asincrono durante la computazione. Quello che ci aspettiamo è che in questo caso il componente ricominci da capo la computazione non considerando i risultati fino a quel momento ottenuti.



Conclusioni

Il componente passa tutti i test precedentemente descritti sia eseguendo simulazioni comportamentali (Behavioral) che simulazioni funzionali (Post-Synthesis). In particolare, sono stati registrati i seguenti tempi di esecuzione:

	Behavioral (ns)	Functional (ns)
Caso generale	4.250	4.350,1
Multi start	19.150	19.650,1
Sequenza massima	383.550	383.650,1
Sequenza minima	1050	1.150,1
Multi reset	13.850	13.950,1

Da questi risultati notiamo come il tempo di esecuzione del componente sia direttamente proporzionale al numero di letture effettuate e al numero di valori in input letti ad ogni computazione.