Anno scolastico 2022/23

POLITECNICO
MILANO 1863

Relazione finale

(Progetto di reti logiche)

Prof. William Fornaciari

A cura di:

Antonio Di Paola Codice Persona: 10717589 Matricola: 956038

Indice

1 - Introduzione

Il progetto di quest'anno aveva l'obiettivo di implementare un componente hardware attraverso il codice di programmazione VHDL. Il componente consisteva in un modulo HW che sulla base dei segnali ricevuti in input andasse a leggere delle informazioni nella memoria con cui si interfaccia per poi restituirle in uno dei 4 possibili output.

1.1 Descrizione generale del componente

Il modulo ha due segnali in input principali (i_w e i_start) entrambi da un bit e cinque output primari: 4 da 8 bit (Z0, Z1, Z2 e Z3) e 1 da 1 bit (o_done). Inoltre il modulo riceve un segnale di clock i_clk e un segnale di reset i_reset.

I dati in ingresso ottenuti dal segnale seriale i_w sono validi fin tanto che il segnale d i_start è alto (si trova ad 1) e sono strutturati in questa maniera:

- I primi due bit indicano il canale di uscita sul quale andrà indirizzato il messaggio letto in memoria. Infatti ad ogni canale è associata una combinazione ordinata di due bit.
  - 00 per il canale Z0
  - 01 per il canale Z1
  - 10 per il canale Z2
  - 11 per il canale Z3
- I restanti, che possono variare da 0 a 16, andranno a costruire l'indirizzo in memoria nel quale andranno ad essere lette le informazioni

Il segnale di o_done è sempre basso eccetto nel ciclo di clock in cui il messaggio letto in memoria viene scritto in uno dei 4 canali di output e rimarrà alto solo durante questo ciclo.

I segnali di uscita sono inizialmente posti a 0 e rimangono immutati fin tanto che il segnale di DONE non venga posto a 1.

1.2 Interfaccia del componente

Il componente ha la seguente interfaccia (mostrata in *figura 1*) con in particolare:

- i_clk che è il segnale di CLOCK generato dal Test Bench
- i_rst che inizializza tutti i segnali al valore di reset
- i_start che è il segnale di START generato dal Test Bench
- i_w che è il segnale W precedentemente mostrato dal Test Bench
- o_z0, o_z1, o_z2, o_z3 che sono i 4 canali di uscita del componente
- o_done che il segnale che indica il termine della lettura dei dati in memoria e della scrittura sul canale
- o_mem_addr che è l'indirizzo da mandare in memoria
- i_mem_data che è il messaggio ricevuto dalla memoria

```
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;
```

Figura 1: interfaccia del componente

- o_mem_en che è il segnale di ENABLE che autorizza la comunicazione con la memoria
- o_mem_we che è il segnale di WRITE ENABLE che autorizza la scrittura in memoria

1.3 Funzionamento del componente

Il componente, in seguito al passaggio da basso ad alto del segnale i_start, inizierà la lettura del segnale seriale i_w. Questa procedura può avere una durata variabile compresa tra i 2 e i 18 cicli di clock, dipendentemente dalla lunghezza dell'indirizzo inviato in input.

Con i primi due bit del segnale i_w, il componente determina su quale canale dovrà essere inviato il messaggio ricevuto dalla memoria e si predispone per salvarlo e mostrarlo in output quando gli sarà chiesto. Con i restanti bit costruisce l'indirizzo da inviare in memoria, non appena il segnale i_start ritornerà basso.

L'indirizzo che il componente invia alla memoria ha lunghezza fissa di 16 bit, pertanto, nel caso in cui il numero di bit ricevuti siano minore, allora questo viene esteso con "0" sui bit più significativi.

Nel ciclo di clock successivo all'invio dell'indirizzo in memoria, il componente riceve da quest'ultima un messaggio dalla lunghezza di 8 bit che viene scritto nel canale preparato precedentemente. Contemporaneamente a questa scrittura viene portato ad alto il segnale di o_done e negli altri canali viene mostrato il messaggio precedentemente salvato.

2 - Architettura

2.1 Descrizione generale

Il componente funziona tramite una macchina a stati finiti (FSM), divisa in 6 differenti stati, che ne gestisce l'orchestrazione.

Il componente resta in attesa fino al passaggio ad alto del segnale i_start, quindi inizia la lettura del segnale di input.

Conclusa la lettura dell'indirizzo inviato in input, il segnale i_start torna basso; questo permette al componente di inviare l'indirizzo letto alla memoria che, in un ciclo di clock, elaborerà le informazioni ricevute e manderà indietro il messaggio corrispondente.

Ricevuto il segnale il componente, tramite un demultiplexer, provvederà a salvare il messaggio nell'opportuno canale.

Contemporaneamente al salvataggio in memoria il componente porta ad alto il segnale o_done, mostra in output il segnale ricevuto nel corretto canale e negli altri mostra i segnali precedentemente salvati.

Al termine del ciclo di clock tutti i canali di output ed il segnale di o_done ritornano a 0.

2.2 Scelte implementative

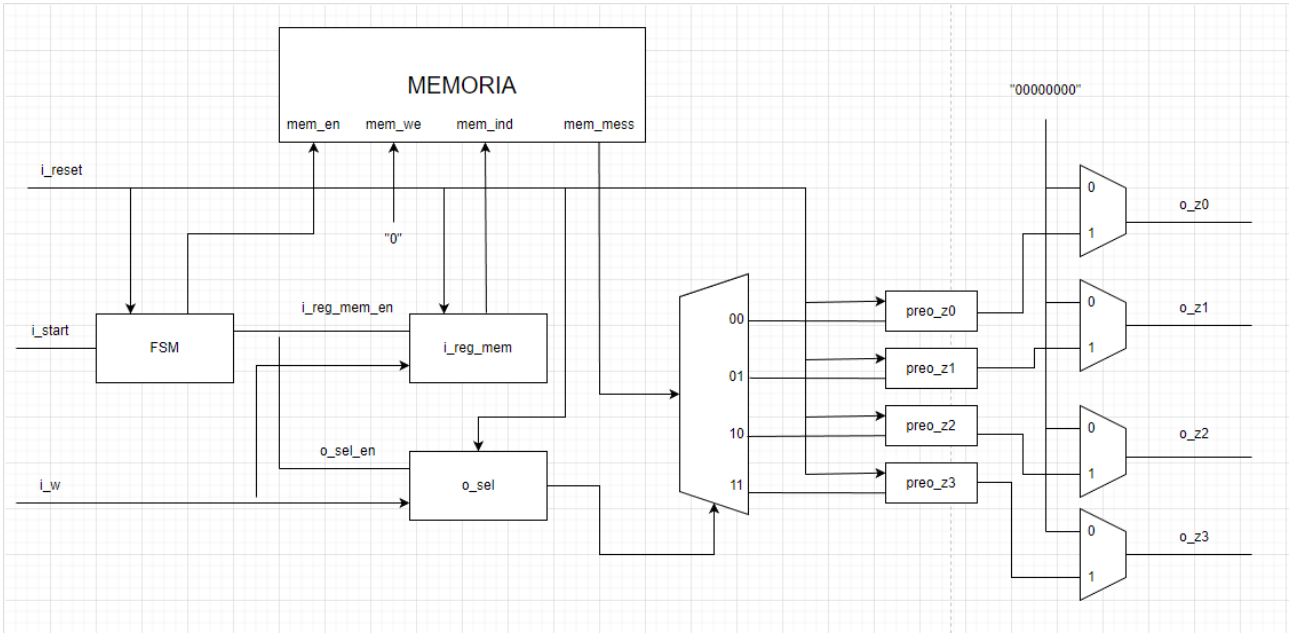


Figura 2: struttura del componente

La FSM gestisce tutta l'organizzazione delle informazioni all'interno del componente: i primi stati sono atti al salvataggio delle informazioni lette in input, mentre gli stati successivi si concentrano sul salvataggio dei dati provenienti dalla memoria e fanno in modo che tali dati vengano correttamente mostrati in output.

Per quanto riguarda la gestione dei dati letti, sia per l'indirizzo da mandare in memoria (i_reg_mem), che per il canale di scrittura(o_sel), vengono utilizzati dei registri di tipo SIPO (Serial In Parallel Out) così che essi siano in grado leggere i segnali bit a bit ma riescano anche ad inviarli in un unico ciclo di clock.

Per quanto riguarda, invece, la scelta del canale di output, al quale viene mandato il messaggio ricevuto in input dalla memoria, si utilizza un demultiplexer che riceve, come selettore, il segnale "o_sel" (nel quale erano stati salvati precedente i primi due bit letti in input).

Infine, per il salvataggio dei messaggi ricevuti dalla memoria si utilizzano dei registri di tipo PIPO (Parallel In Parallel Out) in modo tale da poter sia ricevere che inviare i messaggi in un unico ciclo di clock.

2.3 Stati della macchina a stati finiti

La macchina a stati finiti che gestisce il componente (mostrata in *figura 3*) è formata dai seguenti 6 stati:

1. **reset**: è lo stato di partenza dove viene inizializzato il componente e al quale esso ritorna ogni qual volta, o il segnale di reset viene portato a 1, o il segnale di done torna a 0. Il suo scopo è quello di attendere che il segnale di start scatti a 1 e quindi salvare in un registro il bit più significativo letto dal segnale i_w. Fatta la

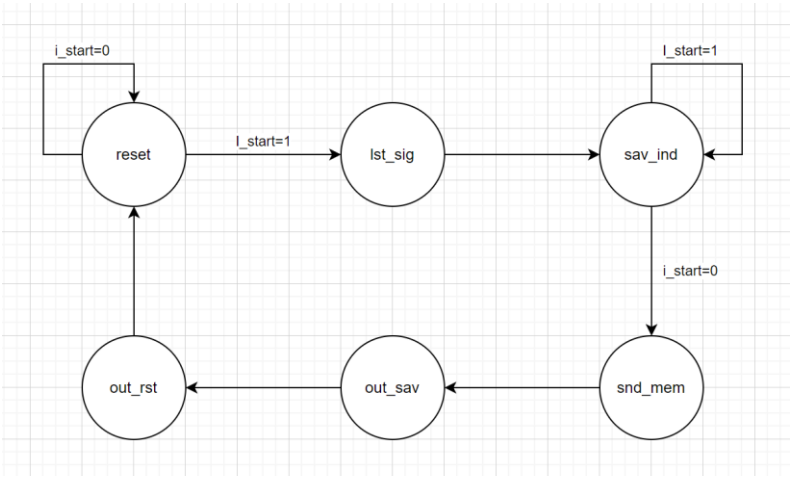


Figura 3:Macchina a stati finiti

lettura e il salvataggio, la macchina passa allo stato successivo.

2. **lst_sig:** questo stato serve per salvare il bit meno significativo dei due che identificano il canale di uscita del messaggio letto in memoria. Il componente resta in questo stato per un unico ciclo di clock e poi passa direttamente al successivo.
3. **sav_ind:** in questo stato la macchina si occupa della lettura e del salvataggio dell'indirizzo da mandare in memoria. La durata di questo stato può variare da 1 fino a 15 cicli di clock dipendentemente dalla lunghezza dell'indirizzo mandato. Il componente passa allo step successivo solo nel momento in cui il segnale di start viene riportato a 0.
4. **snd_mem:** questo stato ha lo scopo di inviare in memoria l'indirizzo salvato nello step precedente. Il componente cambierà stato sempre dopo uno ciclo di clock.
5. **out_sav:** in questo step il componente salva sul canale scelto precedentemente il messaggio ricevuto in memoria, parte il segnale o_done ad "1" e mostra in output nei canali i messaggi salvati. Il componente cambierà stato sempre dopo uno ciclo di clock.
6. **out_rst:** in questo step il componente riporta a "0" il segnale di o_done e tutti i canali di uscita e, alla fine del ciclo di clock, passa allo stato successivo.

3 - Risultati sperimentali

La fase di testing del componente si è svolta in tre fasi distinte: la prima si è concentrata su casi limite per verificare che rispettasse le condizioni limite, la seconda per le verifica della corretta risposta all'inserimento di un reset e, infine, uno stress test sottoponendolo a un serie più lunga di bit in ingresso.

Tutte i test che stanno per essere presentati sono stati eseguiti sui 3 tipi si simulazione: Behavioral, Post-Synthesis Functional, Post Synthesis Timing

3.1 Test dei casi limite

Per l'esecuzione dei test è stata utilizzata la seguente memoria:

```
TYPE ram_type IS ARRAY (65535 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RAM : ram_type := (  0 => STD_LOGIC_VECTOR(to_unsigned(2, 8)),
                            1 => STD_LOGIC_VECTOR(to_unsigned(162, 8)),
                            2 => STD_LOGIC_VECTOR(to_unsigned(75, 8)),
                            3 => STD_LOGIC_VECTOR(to_unsigned(175, 8)),
                            7 => STD_LOGIC_VECTOR(to_unsigned(88, 8)),
                            OTHERS => "00000000"-- (OTHERS => '0')
                         );
```

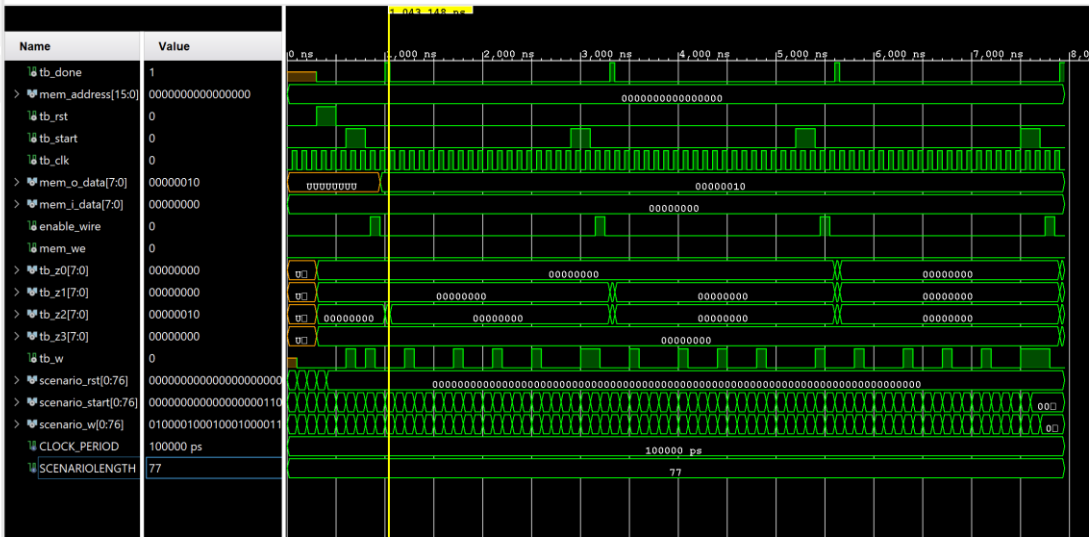
Lo scopo di questi test è di verificare che il componente risponda correttamente nel caso in cui venga sottoposto agli ingressi limite.

Test 1: Indirizzi di memoria non specificati

Il test è stato effettuato inserendo in ogni canale di uscita un indirizzo non specificato (pertanto il segnale i_start è stato ad "1" per soli 2 cicli di clock).

I segnali in ingresso al componente sono stati i seguenti:

```
CONSTANT SCENARIOLENG
SIGNAL scenario_rst :
    & "000" & "000000
SIGNAL scenario_start
    & "110" & "000000
SIGNAL scenario_w : u
    & "001" & "000100
```



E i risultati ottenuti sono i seguenti:

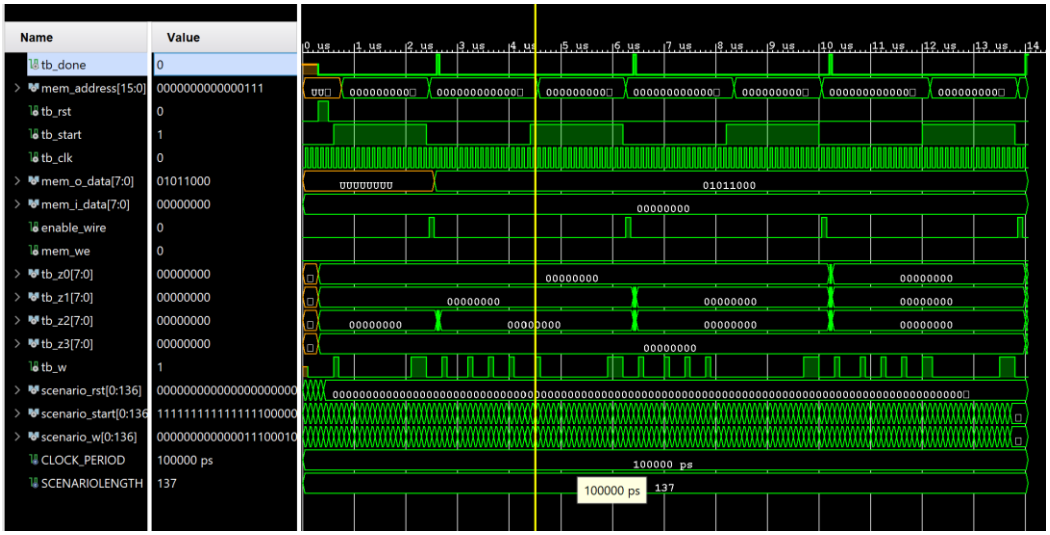
Test 2: Indirizzi di memoria pieni

Il test è stato effettuato utilizzando degli indirizzi di memoria che raggiungessero il numero di bit massimi (ovvero il segnale i_start rimane a “1” per 18 cicli di clock)

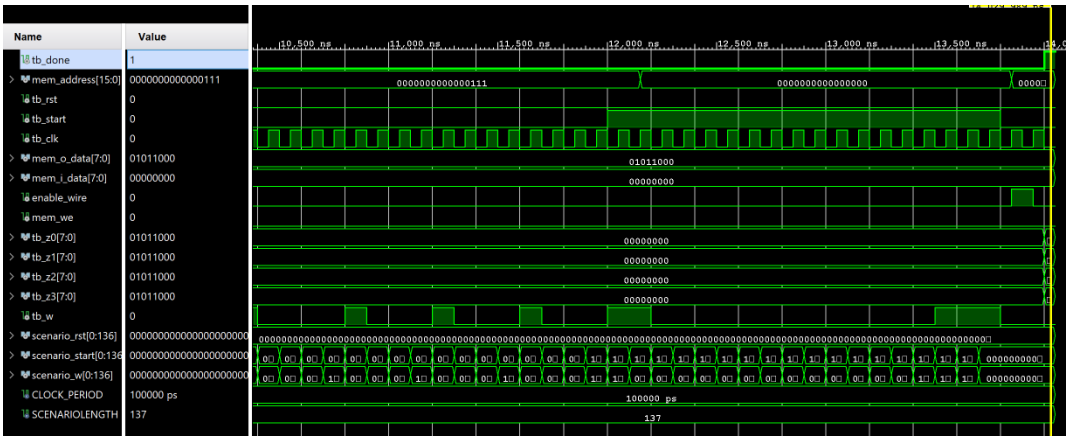
Quindi i segnali in ingresso al componente sono stati i seguenti:

```
CONSTANT SCENARIOLENGTH : INTEGER := 137;
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)    := "00110" & "00" & "0000000000000000" & "0000000000000000000" & "00" & "0000000000000000" &
    "0000000000000000000" & "00" & "0000000000000000" & "0000000000000000000" & "00" & "0000000000000000";
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1) := "00000" & "11" & "1111111111111111" & "0000000000000000000" & "11" & "1111111111111111" &
    "0000000000000000000" & "11" & "1111111111111111" & "0000000000000000000" & "11" & "1111111111111111";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)      := "00000" & "10" & "0000000000000111" & "0001000010001000" & "01" & "0000000000000111" &
    "0001000010001000" & "00" & "0000000000000111" & "0001000010001000" & "11" & "0000000000000111";
```

E i risultati ottenuti sono i seguenti:



E un focus particolare sulla parte finale dell'esecuzione:



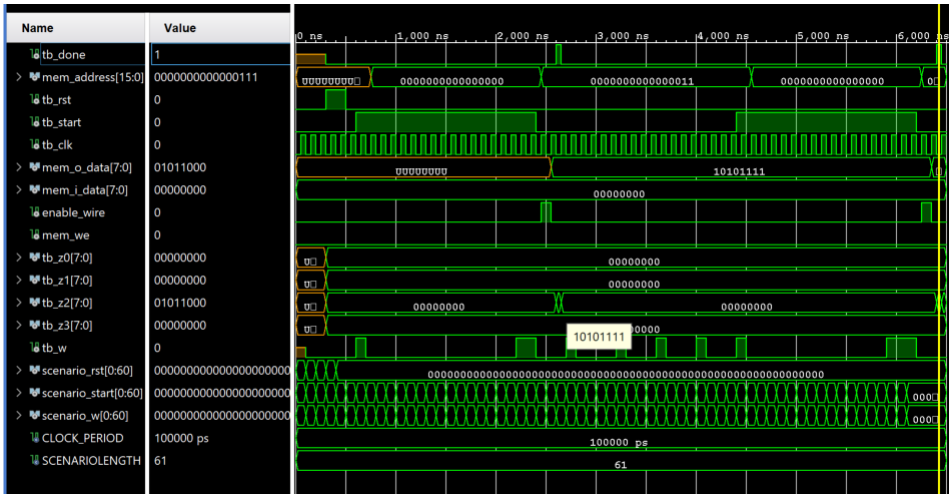
Test 3: Inserimento di un indirizzo e modifica del suo contenuto

Il test è stato effettuato inserendo un indirizzo di memoria in un canale e poi modificando in seguito il suo contenuto.

I segnali in ingresso al componente sono:

```
CONSTANT SCENARIOLENGTH : INTEGER := 61;
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)    := "00110" & "00" & "0000000000000000" & "0000000000000000000" & "00" & "0000000000000000";
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1) := "00000" & "11" & "1111111111111111" & "0000000000000000000" & "11" & "1111111111111111";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)      := "00000" & "10" & "0000000000000011" & "0001000010001000" & "10" & "0000000000000111";
```

E i risultati sono i seguenti:



ottenuti

Sono stati svolti test analoghi anche per gli altri 3 canali

3.2 Test dei casi con reset

La memoria utilizzata per questi casi è la medesima dei test precedenti.

I test sono effettuati per verificare che il componente reagisca correttamente al passaggio ad 1 del segnale di reset.

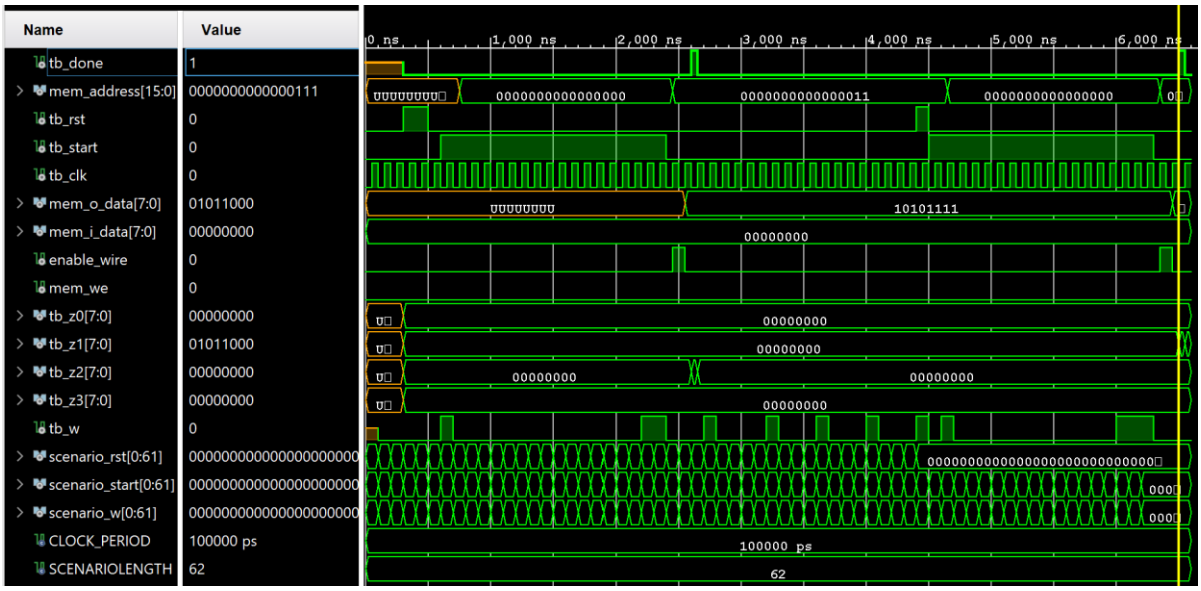
Test 1: Reset mentre la FSM si trova nello stato reset

Il test è stato effettuato mentre la FSM si trova in stato reset (ovvero mentre il componente attende che il segnale di start torni a "1")

I segnali in ingresso sono i seguenti:

```
CONSTANT SCENARIOLENGTH : INTEGER := 62;
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)    := "00110" & "00" & "0000000000000000" & "0000000000000000000" & "1" & "00" & "0000000000000000";
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1) := "00000" & "11" & "1111111111111111" & "0000000000000000000" & "0" & "11" & "1111111111111111";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)      := "00000" & "10" & "0000000000000011" & "0001000010001000" & "1" & "01" & "0000000000000111";
```

E i risultati sono i seguenti:



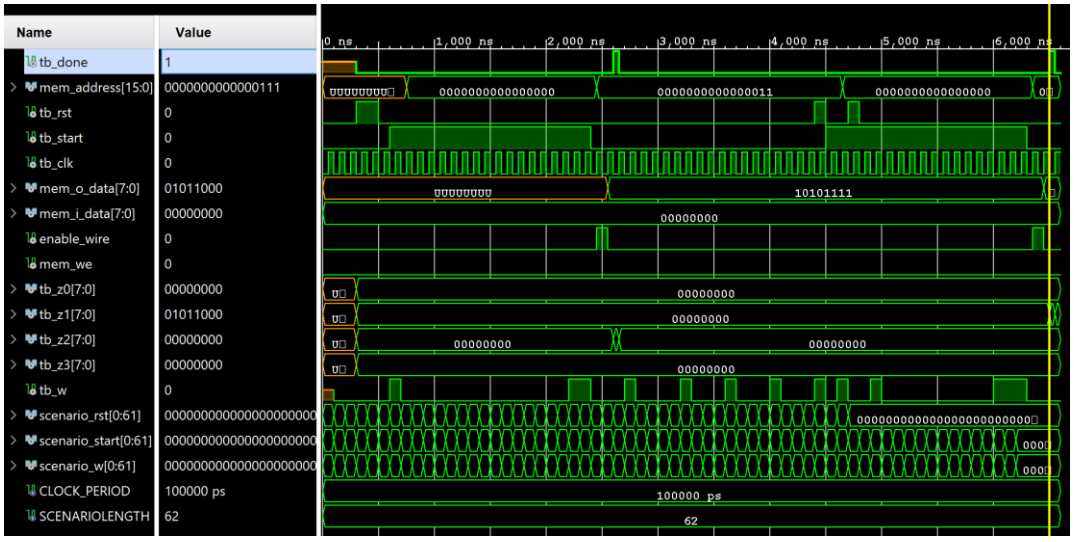
Test 2: Reset mentre la FSM si trova nello stato sav_ind

Il test è stato effettuato mentre la FSM si trova in stato sav_ind (ovvero mentre il segnale di start è “1” e il componente sta salvando l’indirizzo in memoria).

I segnali di ingresso sono i seguenti:

```
CONSTANT SCENARIOLENGTH : INTEGER := 62;
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)   := "00110" & "00" & "0000000000000000" & "0000000000000000000000000000000000000000000000000000000000" & "1" & "00" & "1" & "00" & "0000000000000";
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1) := "00000" & "11" & "1111111111111111" & "0000000000000000000000000000000000000000000000000000000000" & "0" & "11" & "1" & "11" & "1111111111111";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)     := "00000" & "10" & "0000000000000011" & "0001000010001000" & "1" & "01" & "0" & "01" & "0000000000111";
```

E i risultati sono i seguenti:



Sono stati svolti poi test analoghi con il reset in tutti i restanti stati della FSM con risultati analoghi.

3.3 Stress Test

La memoria per questi i test è la seguente:

```
TYPE ram_type IS ARRAY (65535 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RAM : ram_type := (  0 => STD_LOGIC_VECTOR(to_unsigned(2, 8)),
                            1 => STD_LOGIC_VECTOR(to_unsigned(162, 8)),
                            2 => STD_LOGIC_VECTOR(to_unsigned(75, 8)),
                            3 => STD_LOGIC_VECTOR(to_unsigned(175, 8)),
                            7=> STD_LOGIC_VECTOR(to_unsigned(88, 8)),
                            15=> STD_LOGIC_VECTOR(to_unsigned(32, 8)),
                            31=> STD_LOGIC_VECTOR(to_unsigned(76, 8)),
                            OTHERS => "00000000"-- (OTHERS => '0')
                         );
```

Lo scopo dei test è verificare come reagiscono in caso venga inserito un numero superiore di bit.

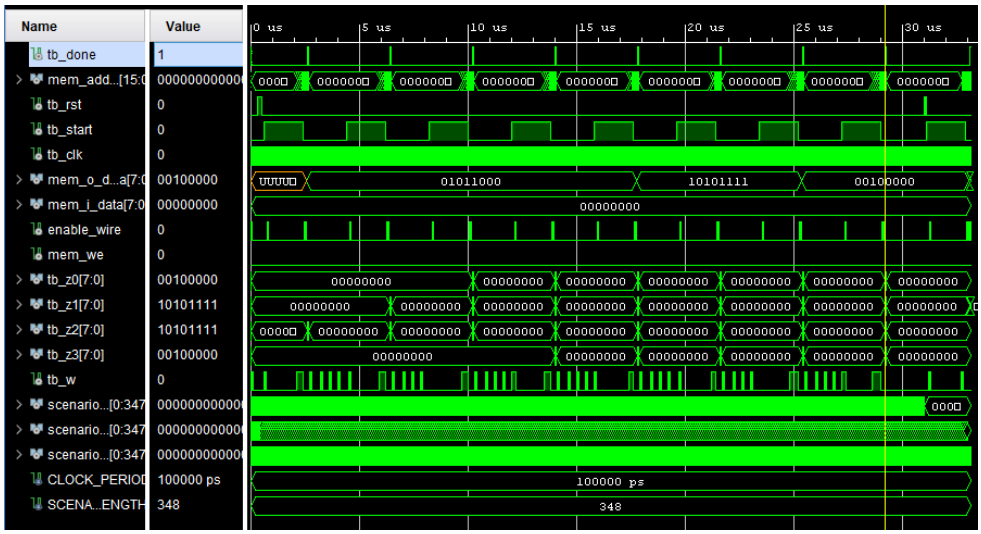
Test 1: Stress test su più canali

Il test è stato effettuato con un segnale d'ingresso con un numero di bit superiore ai 300 inserendo su tutti i canali, modificandoli e infine portando ad 1 il segnale di reset.

I segnali di ingresso sono i seguenti:

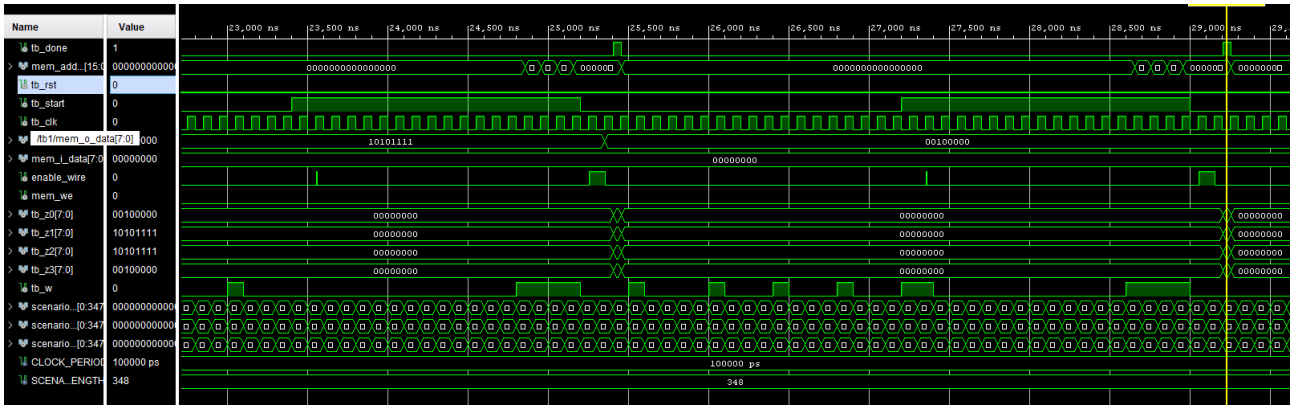
```
CONSTANT SCENARIOLENGTH : INTEGER := 348;
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)   := "00110" & "00" & "0000000000000000" & "0000000000000000000" & "00" & "0000000000000000" &
    "0000000000000000000" & "00" & "0000000000000000" & "0000000000000000000" & "00" & "0000000000000000" & "0000000000000000000" & "00" & "0000000000000000" &
    "0000000000000000000" & "00" & "0000000000000000" & "0000000000000000000" & "00" & "0000000000000000" & "0000000000000000000" & "00" & "0000000000000000" &
    "0000000000000000000" & "1" & "00" & "0000000000000000" & "0000000000000000000";
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1) := "00000" & "11" & "1111111111111111" & "0000000000000000000" & "11" & "1111111111111111" &
    "0000000000000000000" & "11" & "1111111111111111" & "0000000000000000000" & "11" & "1111111111111111" & "0000000000000000000" & "11" & "1111111111111111" &
    "0000000000000000000" & "11" & "1111111111111111" & "0000000000000000000" & "11" & "1111111111111111" & "0000000000000000000" & "11" & "1111111111111111" &
    "0000000000000000000" & "0" & "11" & "1111111111111111" & "0000000000000000000";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)     := "00000" & "10" & "0000000000000111" & "0001000010001001000" & "01" & "0000000000000111" &
    "0001000010001001000" & "00" & "0000000000000111" & "0001000010001001000" & "10" & "0000000000000011" &
    "0001000010001001000" & "01" & "0000000000000011" & "0001000010001001000" & "00" & "0000000000001111" & "0001000010001001000" & "11" & "0000000000001111" &
    "0000000000000000000" & "0" & "01" & "0000000000000010" & "0001000010001001000";
```

E i risultati sono seguenti:



i

Più nel dettaglio:



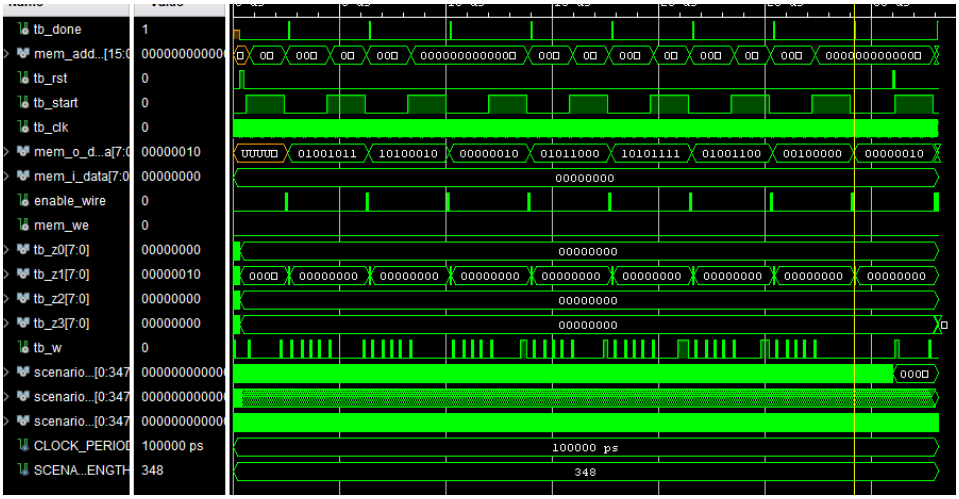
Test 2: Stress test su un singolo canale

Il test è stato effettuato con un segnale di ingresso di una lunghezza pari al test precedente ma modificando un unico canale eccetto l'ultimo dopo il segnale di reset che passa ad "1"

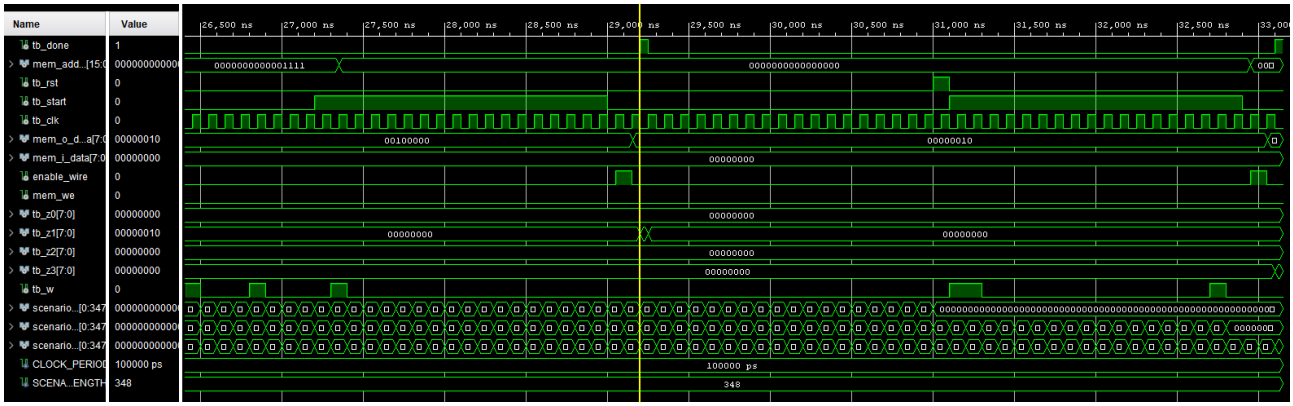
I segnali di ingresso sono i seguenti:

```
CONSTANT SCENARIOLENGTH : INTEGER := 348;
SIGNAL scenario_rst : unsigned(0 TO SCENARIOLENGTH - 1)    := "00110" & "00" & "0000000000000000" & "00000000000000000000" & "00" & "0000000000000000" &
    "00000000000000000000" & "00" & "0000000000000000" & "00000000000000000000" & "00" & "0000000000000000" & "00000000000000000000" & "00" & "0000000000000000" &
    "00000000000000000000" & "00" & "0000000000000000" & "00000000000000000000" & "00" & "0000000000000000" & "00000000000000000000" & "00" & "0000000000000000" &
    "00000000000000000000" & "1" & "00" & "0000000000000000" & "00000000000000000000";
SIGNAL scenario_start : unsigned(0 TO SCENARIOLENGTH - 1) := "00000" & "11" & "1111111111111111" & "00000000000000000000" & "11" & "1111111111111111" &
    "00000000000000000000" & "11" & "1111111111111111" & "00000000000000000000" & "11" & "1111111111111111" & "00000000000000000000" & "11" & "1111111111111111" &
    "00000000000000000000" & "11" & "1111111111111111" & "00000000000000000000" & "11" & "1111111111111111" & "00000000000000000000" & "11" & "1111111111111111" &
    "00000000000000000000" & "0" & "11" & "1111111111111111" & "00000000000000000000";
SIGNAL scenario_w : unsigned(0 TO SCENARIOLENGTH - 1)      := "00000" & "01" & "0000000000000010" & "0001000010001001000" & "01" & "0000000000000001" &
    "0001000010001001000" & "01" & "0000000000000000" & "0001000010001001000" & "01" & "0000000000000111" & "0001000010001001000" & "01" & "0000000000000011" &
    "0001000010001001000" & "01" & "0000000000011111" & "0001000010001001000" & "01" & "0000000000001111" & "0001000010001001000" & "01" & "0000000000000000" &
    "00000000000000000000" & "0" & "11" & "0000000000000010" & "0001000010001001000";
```

E i risultati sono i seguenti:



Più nel dettaglio:



4- Conclusioni

Il componente ha passato come appena mostrato tutti i test a cui è stato sottoposto in tutti i tipi di simulazioni.

Il componente presenta un totale di 86 flip flop e in completa assenza di latch

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	27	0	134600	0.02
LUT as Logic	27	0	134600	0.02
LUT as Memory	0	0	46200	0.00
Slice Registers	86	0	269200	0.03
Register as Flip Flop	86	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

4.1 Ottimizzazioni

Per cercare di ottimizzare le perfomance sono state applicate le seguenti modifiche:

- *Ottimizzazione numero di stati della FSM:* riduzione del numero di stati della FSM a 6 riducendo gli stati ponte allo stretto indispensabile per mantenere la macchina funzionante.

- *Scelta ottimale dei registri:* utilizzo dei registri SIPO per il salvataggio degli input letti in memoria così da ridurre il tempo di invio del loro contenuto
- *Eliminazione di tutti i latch:* modifica della struttura del codice per eliminare tutti i latch che si erano creati

4.2 Design Timing Summary

Per avere un parametro che possa quantificare concretamente la durata di esecuzione del programma allora si fornisce il Worst Negative Slack repocurato disponibile nella tabella Design Timing Summary del componente sintetizzato

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	47,013 ns	Worst Hold Slack (WHS):	0,132 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:	189	Total Number of Endpoints:	189	Total Number of Endpoints:	87