Prove Finale del Progetto Reti Logiche
Prof. William Fornaciari

POLITECNICO
MILANO 1863

Alessandro Assini Codice Persona 10726342 - Matricola 960109
Riccardo Antonelli Codice Persona 10731730 - Matricola 955973

Anno Accademico 2022-23

Indice

Capitolo 1

Introduzione

1.1 Scopo del Progetto

Lo scopo del progetto è l'implementazione di un modulo hardware descritto in VHDL, che si interfaccia con una memoria per l'aquisizione di dati contenuti ad uno specifico indirizzo, con l'obiettivo di inviare questi dati in uscita ad uno dei quattro canali di uscita del modulo.

1.2 Specifiche Generali

Il segnale in arrivo al modulo viene ricevuto sequenzialmente, un bit alla volta, ed è composto in ordine da:

- 2 bit che fanno riferimento al canale di uscita desiderato
- N bit con cui viene ottenuto l'indirizzo della memoria.

Una volta ottenuti i dati dalla memoria, il modulo li invia sull'uscita desiderata.
La sequenza di bit in ingresso viene letta finché il segnale in ingresso START si trova sul fronte di salita (vale 1), e termina appena diventa 0. Per permettere il funzionamento corretto del componente START, una volta sul fronte di salita, vi rimarrà per almeno 2 cicli di clock, e non più di 18.
Il valore del segnale in uscita DONE monitora ciò che viene mostrato dal componente sulle sue 4 uscite. Quando si trova sul fronte di salita (vale 1) vengono mostrati i dati ottenuti fino a quel momento, altrimenti vengono mostrate delle sequenze di zeri.

Esempio:

Un esempio di processo è il seguente, con gli ingressi sequenziali per i rispettivi segnali elencati sotto:

i_w	0	0	1	**1**	**0**	**1**	0	0	1	0	0	1	1	1	1	0	0
i_start	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0
o_done	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
o_z0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	**00**	00	00
o_z1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	**00**	00	00
o_z2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	**dato**	00	00
o_z3	00	00	00	00	00	00	00	00	00	00	00	00	00	00	**00**	00	00

Appena START è uguale a 1, comincia la lettura di W, che assegnerà i primi due valori al canale di uscita (i valori evidenziati in arancione), che sarà "10" ovvero Z2, mentre il resto per l'indirizzo di memoria (i valori evidenziati in azzurro), che in questo caso sarà "0000000000000001".
Quando DONE è uguale a 1, viene rappresentato il "dato" in memoria cercato, al canale di uscita corrispondente.

1.3 Interfaccia del componente

Il componente da descrivere possiede la seguente interfaccia:

```
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;
```

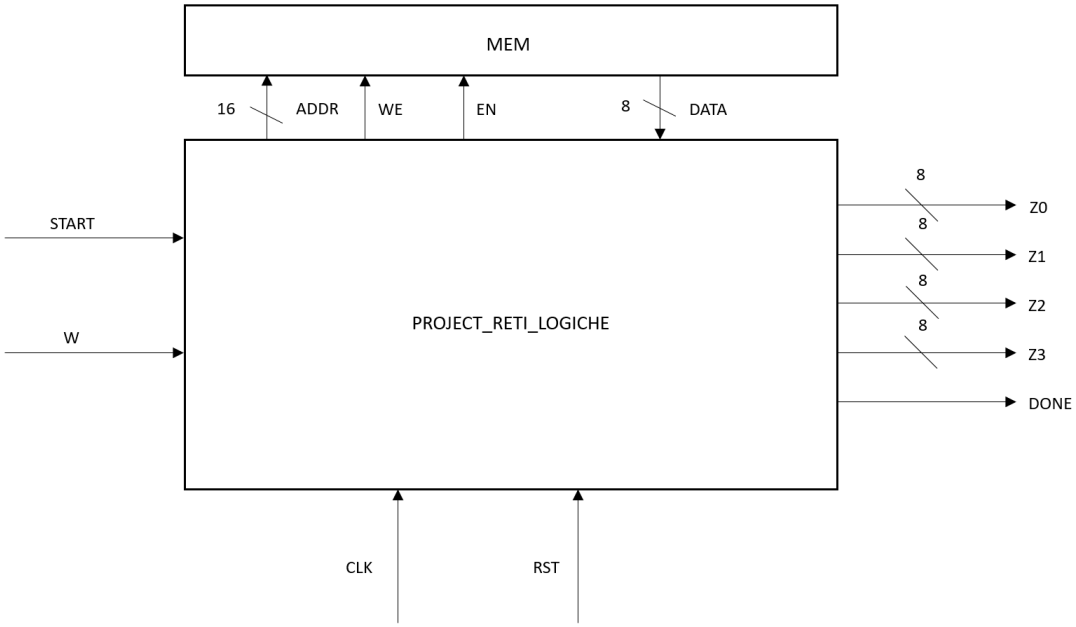
In particolare:

- **i_clk** è il segnale di CLOCK in ingresso generato dal Test Bench.
- **i_rst** è il segnale di RESET che inizializza il componente.
- **i_start** è il segnale di START generato dal Test Bench.
- **i_w** è il segnale "W" che corrisponde al bit in ingresso.
- **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 vettore di uscita che manda l'indirizzo alla memoria.
- **i_mem_data** è il vettore in ingresso che arriva dalla memoria con il dato richiesto in precedenza.
- **o_mem_en** è il segnale di Enable che permette di comunicare con la memoria.
- **o_mem_we** è il segnale di Write Enable che permette di scrivere sulla memoria.

Capitolo 2

Architettura

2.1 Schema del Componente



2.2 Descrizione Segnali Interni

All'interno del modulo sono stati inseriti i segnali:

```
signal curr_state : S;
signal bit_reg : std_logic_vector(1 downto 0);
signal n_bit_reg : std_logic_vector(15 downto 0);
signal z0_bit_reg : std_logic_vector(7 downto 0);
signal z1_bit_reg : std_logic_vector(7 downto 0);
signal z2_bit_reg : std_logic_vector(7 downto 0);
signal z3_bit_reg : std_logic_vector(7 downto 0);
signal inizio : std_logic;
```

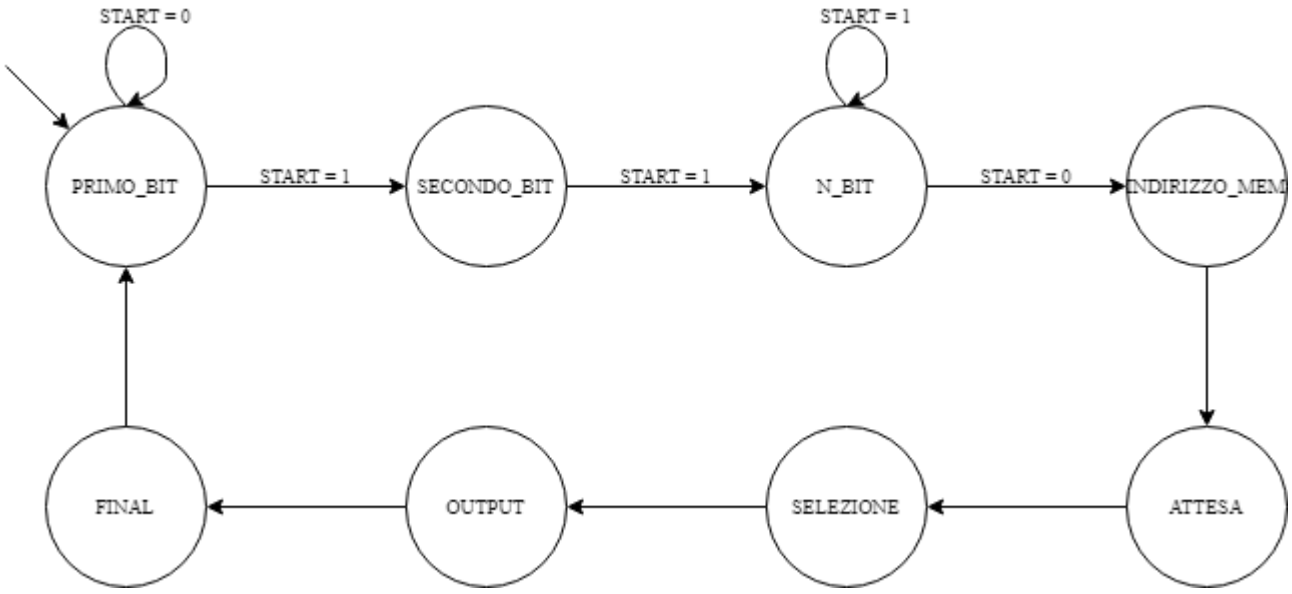
Rispettivamente:

- **S**: è il segnale utilizzato per lo sviluppo degli stati della FSM.
- **bit_reg**: è il segnale in cui salvo l'uscita desiderata all'inizio del processo. È un vettore inizializzato a "00" e durante la lettura dei primi due bit attraverso l'operatore "concatenazione" viene aggiornato.
- **n_bit_reg**: è il segnale in cui salvo l'indirizzo da passare alla memoria. È un vettore di 16 bit inizializzato a "0000000000000000" e dopo la lettura dei primi due bit, finché i_start = '1', viene aggiornato attraverso l'operatore "concatenazione".
- **z0_bit_reg**: è il segnale in cui viene salvato il valore all'uscita z0.
- **z1_bit_reg**: è il segnale in cui viene salvato il valore all'uscita z1.
- **z2_bit_reg**: è il segnale in cui viene salvato il valore all'uscita z2.
- **z3_bit_reg**: è il segnale in cui viene salvato il valore all'uscita z3.
- **inizio**: è il segnale necessario a mantenere corretta l'acquisizione dei dati dalla memoria. Viene posto a '1' e usato solo per inizializzare i segnali nella prima metà del primo ciclo di clock, per il resto del tempo viene settato uguale a '0'. (aggiunto per il superamento del "tb_example23_agg.vhd")

2.3 Scelte Progettuali

Il segnale **o_men_en** avrà sempre valore '1', ad eccezione del momento in cui si attiva i_rst, in cui passa a '0', e la stessa cosa vale anche per il segnale **o_mem_we**, inizialmente a '0'. La FSM gestisce il salvataggio dei dati provenienti dalla memoria in un momento specifico, evitando così errori nell'acquisizione e permettendo che il segnale **o_mem_en** rimanga a 1 senza creare problemi.
I segnali **z0_bit_reg**, **z1_bit_reg**, **z2_bit_reg**, **z3_bit_reg** sono vettori da 8 bit inizializzati a "00000000" utilizzati come registri e vengono aggiornati durante la selezione del canale di output.

2.4 Macchina a Stati



Descrizione degli stati:

- **PRIMO_BIT**: Stato in cui viene letto il primo bit se START = 1 altrimenti attendiamo sempre in questo stato che START vada a 1.
  Quando leggiamo il primo bit questo viene concatenato con l'ultimo bit del vettore interno "bit_reg".
- **SECONDO_BIT**: Stato in cui viene letto il secondo bit con le stesse condizioni dello stato precedente.
  Questo viene concatenato con l'ultimo bit del vettore "bit_reg", ottenendo l'uscita corrispondente.
- **N_BIT**: Stato in cui si continuano a leggere bit se START = 1, ogni bit viene concatenato con gli ultimi 15 bit del vettore "n_bit_reg", il quale in partenza è inizializzato con tutti zeri.
  Quando START = 0 inizia l'elaborazione.
- **INDIRIZZO_MEM**: Stato in cui al segnale "o_mem_address" in uscita dal componente si associa l'indirizzo in "n_bit_reg".
- **ATTESA**: Stato in cui si attende la risposta della memoria dopo l'invio dell'indirizzo nello stato precedente.
- **SELEZIONE**: Stato in cui viene inserito il dato ricevuto dalla memoria nel registro del canale scelto inizialmente tramite il segnale "bit_reg".
- **OUTPUT**: Stato in cui produco l'output mettendo il segnale "o_done" a 1.
  Ogni uscita produrrà il valore salvato nel registro corrispondente, e in caso non sia stato salvato nulla mostrerà un vettore di zeri.
- **FINAL**: Stato in cui si sistemano i segnali e ci si prepara per ricevere un nuovo input.

Capitolo 3

Risultati Sperimentali

3.1 Report di Sintesi

Il componente sintetizzato supera correttamente tutti i test specificati nelle due simulazioni: Behavioral, Post-Synthesis Functional.

Report Utilization:

```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   44 |     0 |          0 |    134600 |  0.03 |
|   LUT as Logic          |   44 |     0 |          0 |    134600 |  0.03 |
|   LUT as Memory         |    0 |     0 |          0 |     46200 |  0.00 |
| Slice Registers         |  103 |     0 |          0 |    269200 |  0.04 |
|   Register as Flip Flop |  103 |     0 |          0 |    269200 |  0.04 |
|   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 |
+-------------------------+------+-------+------------+-----------+-------+
```

Report Timing:

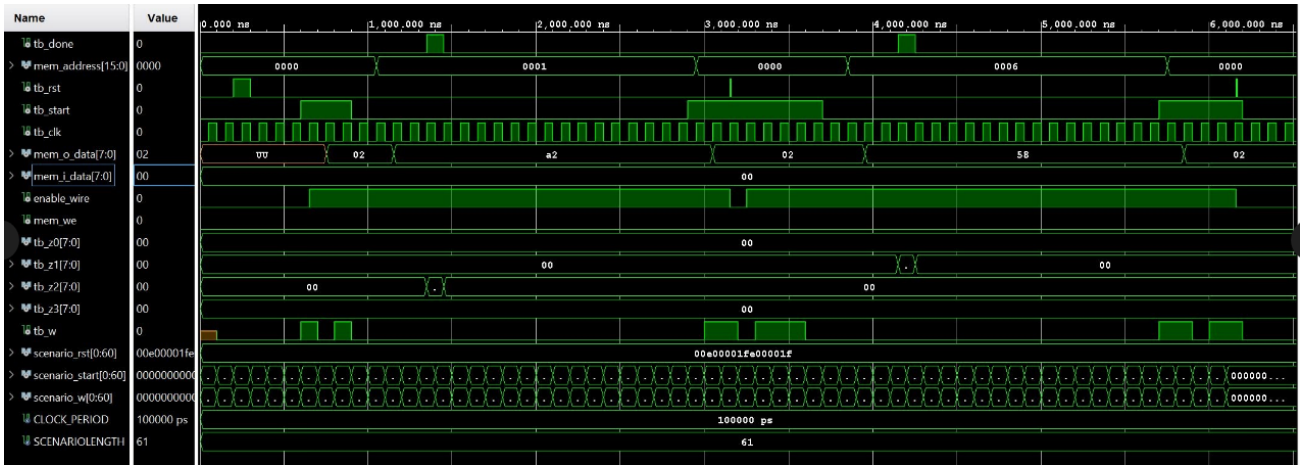
```
Timing Report

Slack (MET) :             97.462ns  (required time - arrival time)
  Source:                 FSM_sequential_curr_state_reg[2]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Destination:            z0_bit_reg_reg[0]/CE
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        2.156ns  (logic 0.751ns (34.833%)  route 1.405ns (65.167%))
  Logic Levels:           1  (LUT5=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 102.100 - 100.000 )
    Source Clock Delay      (SCD):    2.424ns
    Clock Pessimism Removal (CPR):    0.178ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

3.2 Simulazioni

Dopo aver testato il componente sintetizzato con alcuni test bench di esempio, abbiamo definito ulteriori test per verificare il suo corretto funzionamento durante i casi limite, in modo da coprire tutti i possibili cammini che il componente può seguire durante la sua computazione.

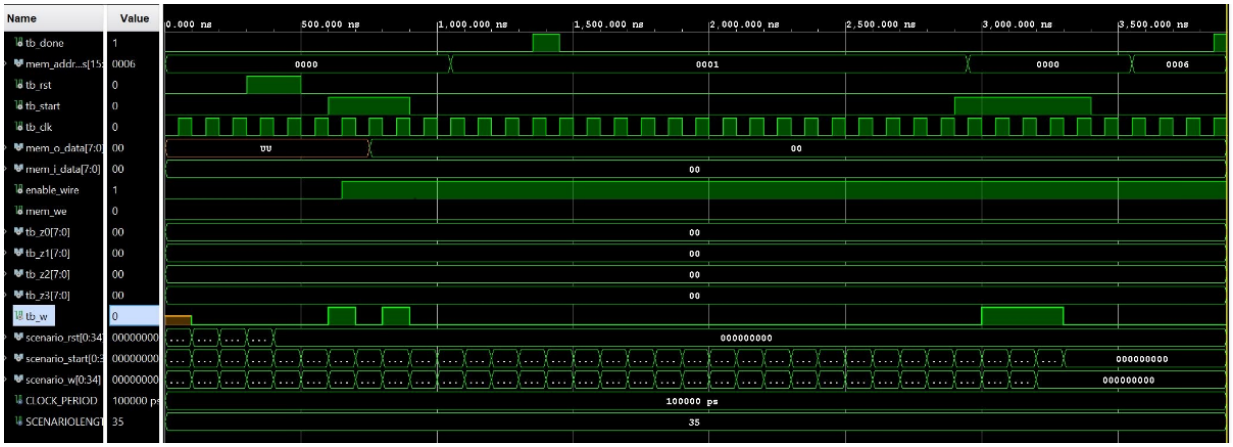
Di seguito c'è una breve descrizione dei test più significativi, accompagnati da immagini che mostrano l'andamento dei segnali durante la simulazione. In questo modo, sarà possibile avere una chiara rappresentazione del comportamento del componente durante i test effettuati.

Reset Asincrono:



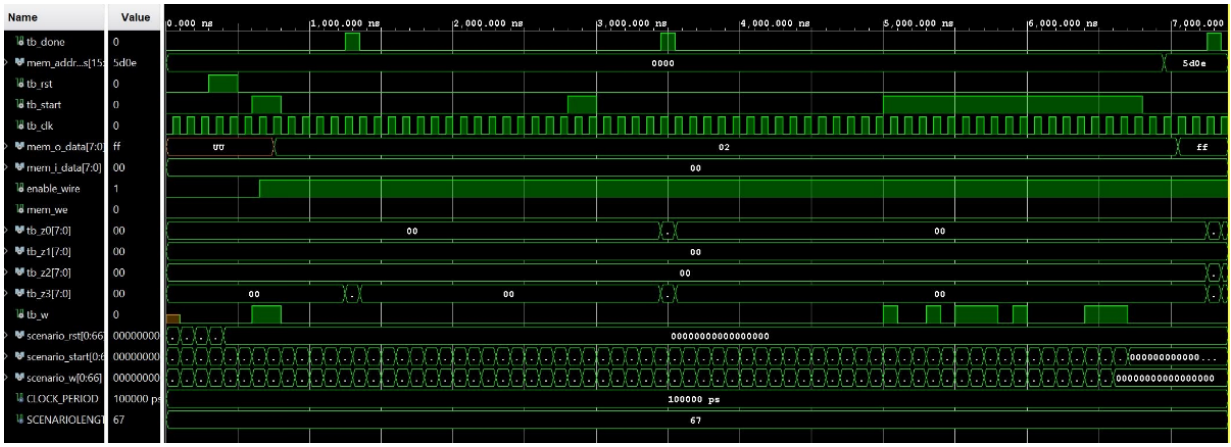
Il test verifica che un segnale di reset inaspettato, che si attiva in un qualunque momento del clock, non comprometta la computazione e che essa ricominci, facendo ritornare l'automa nello stato iniziale, resettandone i segnali.
Il successo di questo test garantisce che il componente che abbiamo realizzato riesce a sostenere cambi improvvisi nella logica in ingresso del reset, reagendo in maniera precisa e rapida senza errori.

Sequenza Zero:



Il test verifica che il componente si comporti nel modo corretto quando il segnale **mem_o_data** è sempre inizializzato come una sequenza di soli zeri, ovvero quando dalla memoria arriva sempre un dato in uscita con tutti zeri.

Sequenza Minimo e Massimo:

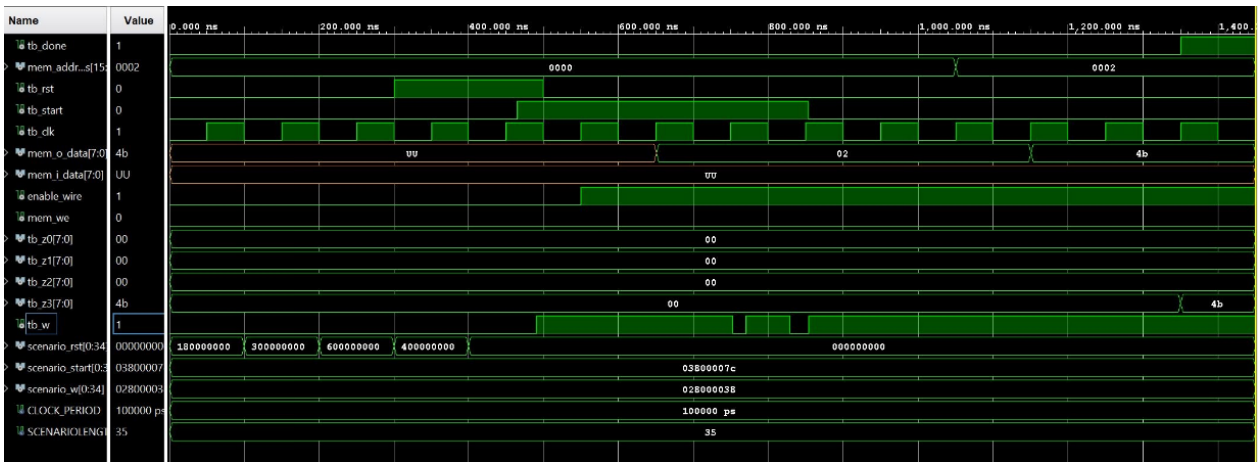


Il test ha lo scopo di verificare i casi limiti del segnale di START.
Controlla sia il numero minimo di clock per cui START è alzato, ovvero 2, sia per il numero massimo, ovvero 18.
Il successo di questo test ci conferma che

Caso Minimo: senza alcun dato esterno inserito nell'indirizzo da inviare alla memoria, viene comunque inviato il segnale con l'indirizzo corretto, ovvero tutti zeri.

Caso Massimo: raggiunge i 18 cicli di clock senza errori sull'indirizzo da inviare alla memoria, e resetta tutti i segnali correttamente al termine della lettura dei valori in input di **i_w**.

Ingresso Asincrono:



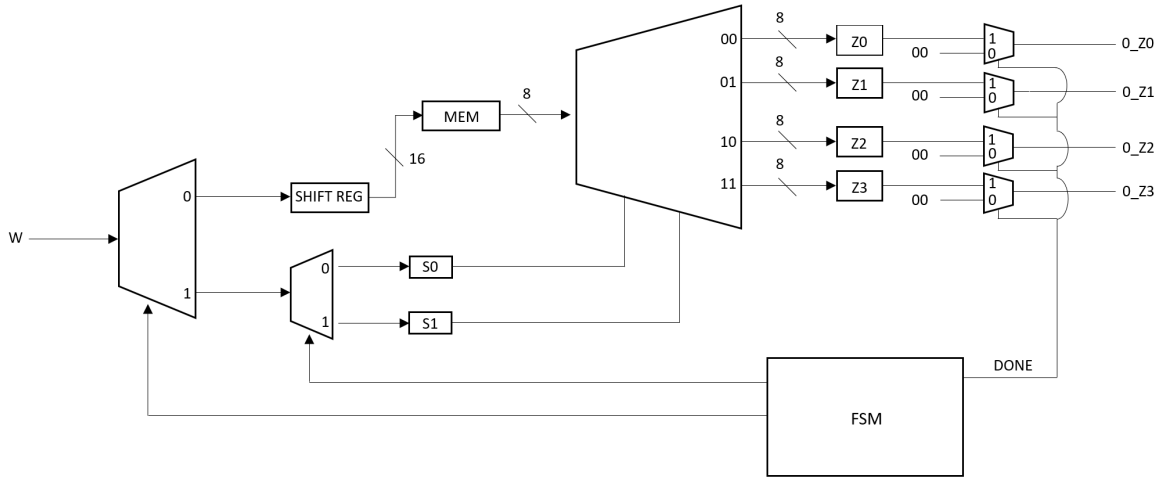
Il test verifica che il componente legga un input corretto anche quando questo non è sincronizzato con il ciclo di clock.
Il successo di questo test garantisce che un cambiamento improvviso sul valore in ingresso **i_w** durante il fronte di discesa del clock non avrà effetti sul processo di lettura.
Inoltre in caso di una variazione sul fronte di salita, non c'è la possibilità di avere un'incongruenza sui valori letti.

Capitolo 4

Conclusione

4.1 Sviluppo del Progetto

Il modulo hardware descritto in questo documento rappresenta la terza versione del nostro progetto.
Nella prima versione, erano presenti "component" come multiplexer e demultiplexer, tuttavia il codice risultava troppo lungo e il funzionamento non era del tutto corretto.
L'idea sulla struttura era la seguente:



Nella seconda versione, abbiamo sostituito questi component con dei process, in modo da organizzare tutti i registri e demultiplexer in processi separati. Questa versione funzionava meglio, ma presentava ancora alcuni problemi e non passava la sintesi.
Infine siamo giunti a questa terza versione che funziona correttamente, delegando in maniera compatta e più comprensibile tutti processi delle versioni precedenti alla FSM.

4.2 Osservazioni Personali

Nel primo sviluppo del progetto, con l'utilizzo dei "component", la codifica dei segnali tra i diversi moduli ci ha fatto comprendere molto meglio aspetti basilari riguardo il funzionamento degli elementi che stavamo considerando, e di come questi stessi elementi potessero essere adattati alle nostre necessità. Abbiamo usato proprio queste nuove conoscenze, apprese attraverso errori negli sviluppi precedenti, per sviluppare la soluzione basata sulla FSM, la quale, seppur in maniera più semplificata, utilizza la stessa logica iniziale.
Il progetto ci ha fornito una rappresentazione fisica del nostro codice e delle relazioni tra i componenti studiati durante il corso.
Pensiamo che sia stato fondamentale per il nostro apprendimento, ed il fatto di averlo svolto in coppia ha sicuramente aiutato a sviluppare la capacità di lavorare in squadra di cui avremo bisogno in futuro.