POLITECNICO
MILANO 1863

Ingegneria Informatica
Corso di Laurea Triennale — Milano Leonardo
A.A. 2019/2020

Prova Finale di Reti Logiche
Working Zone

Professore:
Prof. Gianluca Palermo

Studente:
Mirko Usuelli
matr. 888170
cod. 10570238

Milano — 1 Aprile, 2020

Contents

1 Introduzione

1.1 Scopo del progetto

Lo scopo del progetto è la realizzazione di un componente hardware descritto in VHDL attinente al metodo di codifica a bassa dissipazione di potenza denominato *"Working Zone"*; tale componente prenderà il nome `project_reti_logiche`. Dati 8 indirizzi base — referenti le *Working Zone* disponibili — si codifichi l'indirizzo indicato a seconda della sua appartenenza ad una di esse.

1.2 Specifiche progettuali

1.2.1 Working Zone

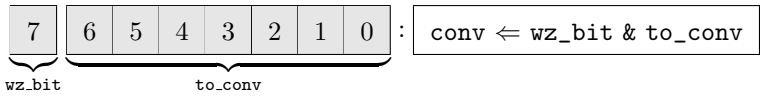
Una *Working Zone* è un'area di memoria avente un proprio indirizzo, un valore base (`wz` con `wz_offset = 0`) e ulteriori 3 valori discostanti da quello base attraverso un offset unitario progressivo.
Per un totale di 8 *Working Zone* — ognuna delle quali è enumerata univocamente in maniera sequenziale (`wz_num`) — risultano esserci 32 valori coperti.

`0x----` : indirizzo	`wz` : valore base	`wz_offset = 0` : base
		`wz_offset = 1` : base + 1
		`wz_offset = 2` : base + 2
		`wz_offset = 3` : base + 3

1.2.2 Codifica

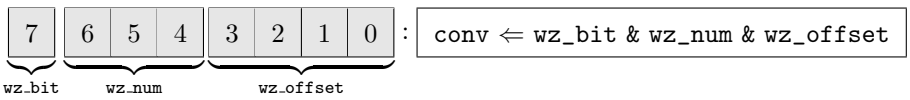
L'elaborazione di codifica consiste nell'ottenere un valore codificato (`conv`) verificando l'appartenenza di un dato (`to_conv`) ad una delle possibili *Working Zone*; indicando con il simbolo "`&`" la *concatenazione* tra i signal, possiamo distinguere due scenari in cui il *byte* del signal `conv` assume codifiche diverse:

- **Indirizzo non appartenente a nessuna *Working Zone*** :



  - `wz_bit` : uguale a 0, indicante la non appartenenza (1 bit).
  - `to_conv` : valore da codificare (7 bit).

- **Indirizzo appartenente ad una *Working Zone*** :



  - `wz_bit` : uguale a 1, indicante l'appartenenza (1 bit).
  - `wz_num` : identificativo numerico della *Working Zone* (3 bit).

– `wz_offset` : discostamento dall'indirizzo base della *Working Zone* (4 bit).

1.2.3 Descrizione della memoria

Gli indirizzi della memoria RAM — a cui si interfaccia `project_reti_logiche` — sono composti da 16 bit, mentre il valore per ogni cella è di 8 bit.
Una *Working Zone* può assumere un valore massimo pari a 124 (avendo l'offset di 4 indirizzi che porta fino al valore limite consentito di 127).
Pertanto risulta che il MSB è acceso per il solo indirizzo `0x0009` nel momento in cui il valore nella cella `0x0000` appartenga ad una delle *Working Zone* comprese tra l'indirizzo `0x0001` e `0x0007` .

Indirizzo 0	0x0000	to_conv
Indirizzo 1	0x0001	wz_num = 000
Indirizzo 2	0x0002	wz_num = 001
Indirizzo 3	0x0003	wz_num = 010
Indirizzo 4	0x0004	wz_num = 011
Indirizzo 5	0x0005	wz_num = 100
Indirizzo 6	0x0006	wz_num = 101
Indirizzo 7	0x0007	wz_num = 110
Indirizzo 8	0x0008	wz_num = 111
Indirizzo 9	0x0009	conv

Working Zone

1.3 Scelte progettuali

1.3.1 Funzionamento

La strategia scelta consiste nel prelevare `to_conv` e caricare di volta in volta una *Working Zone* da analizzare nei suoi `wz_offset`.

Una volta trovata l'appartenenza si *bypasseranno* i controlli successivi e si procederà con la fase di codifica dedicata per poi passare con la scrittura in memoria di `conv`.
In caso di non appartenenza verrano effettuati tutti i confronti possibili e si procederà con l'usuale procedura di codifica e scrittura in memoria.

1.3.2 Signals principali

- `to_conv` : valore da convertire secondo la codifica richiesta.
  - **Dimensione** : 8 bit
  - **Dominio** : `to_conv` $\in [0, 127]$
  - **Indirizzamento** : `0x0000`

- `wz` : valore effettivo referente all'indirizzo di base della *Working Zone* presa in considerazione.
  - **Dimensione** : 8 bit
  - **Dominio** : `wz` $\in [0, 124]$
  - **Indirizzamento** : `0x0001` — `0x0008`

- `conv` : valore convertito al termine dell'elaborazione.
  - **Dimensione** : 8 bit
  - **Dominio** : `to_conv` $\in [1, 248]$
  - **Indirizzamento** : `0x0009`

- `wz_bit` : bit indicante l'appartenenza (=1) o meno (=0) ad una *Working Zone*.
  - **Dimensione** : 1 bit
  - **Dominio** : `wz_bit` $\in \{0, 1\}$

- `wz_num` (Nwz) : enumerativo che identifica la *working-zone* (address : 0x01 — 0x08).
  - **Dimensione** : 3 bit
  - **Dominio** : `wz_num` $\in [0, 7]$

- `wz_offset` (Dwz) : ogni *Working Zone* possiede 4 indirizzi — compreso quello identificativo — e il suo offset da esso è rappresentato attraverso la codifica `ONE-HOT` come segue.

`wz_offset` $= 0$	$\longrightarrow$	`0001`	(indirizzo base)
`wz_offset` $= 1$	$\longrightarrow$	`0010`	
`wz_offset` $= 2$	$\longrightarrow$	`0100`	
`wz_offset` $= 3$	$\longrightarrow$	`1000`	

  - **Dimensione** : 4 bit
  - **Dominio** : `wz_offset` $\in \{1, 2, 4, 8\}$

1.4 Interfaccia del componente

Il progetto è stato interamente basato sulla scheda `xc7a200tfbg484-1` e sviluppato mediante `Vivado 2019.2`.

```
entity project_reti_logiche is
    port (
        i_clk       : in std_logic;
        i_start     : in std_logic;
        i_rst       : 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;
```

- `i_clk` è il segnale di `CLOCK` in ingresso generato dal Test Bench;
- `i_start` è il segnale di `START` generato dal Test Bench;
- `i_rst` è il segnale di `RESET` che inizializza la macchina pronta per ricevere i l primo segnale di `START`;
- `i_data` è i l segnale (vettore) che arriva dalla memoria i n seguito ad una richiesta di lettura;
- `o_address` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `o_done` è i l segnale di uscita che comunica l a fine dell'elaborazione e i l dato di uscita scritto in memoria;
- `o_en` è i l segnale di `ENABLE` da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- `o_we` è i l segnale di `WRITE ENABLE` da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- `o_data` è il segnale (vettore) di uscita dal componente verso la memoria.

2 Architettura

2.1 Schema funzionale

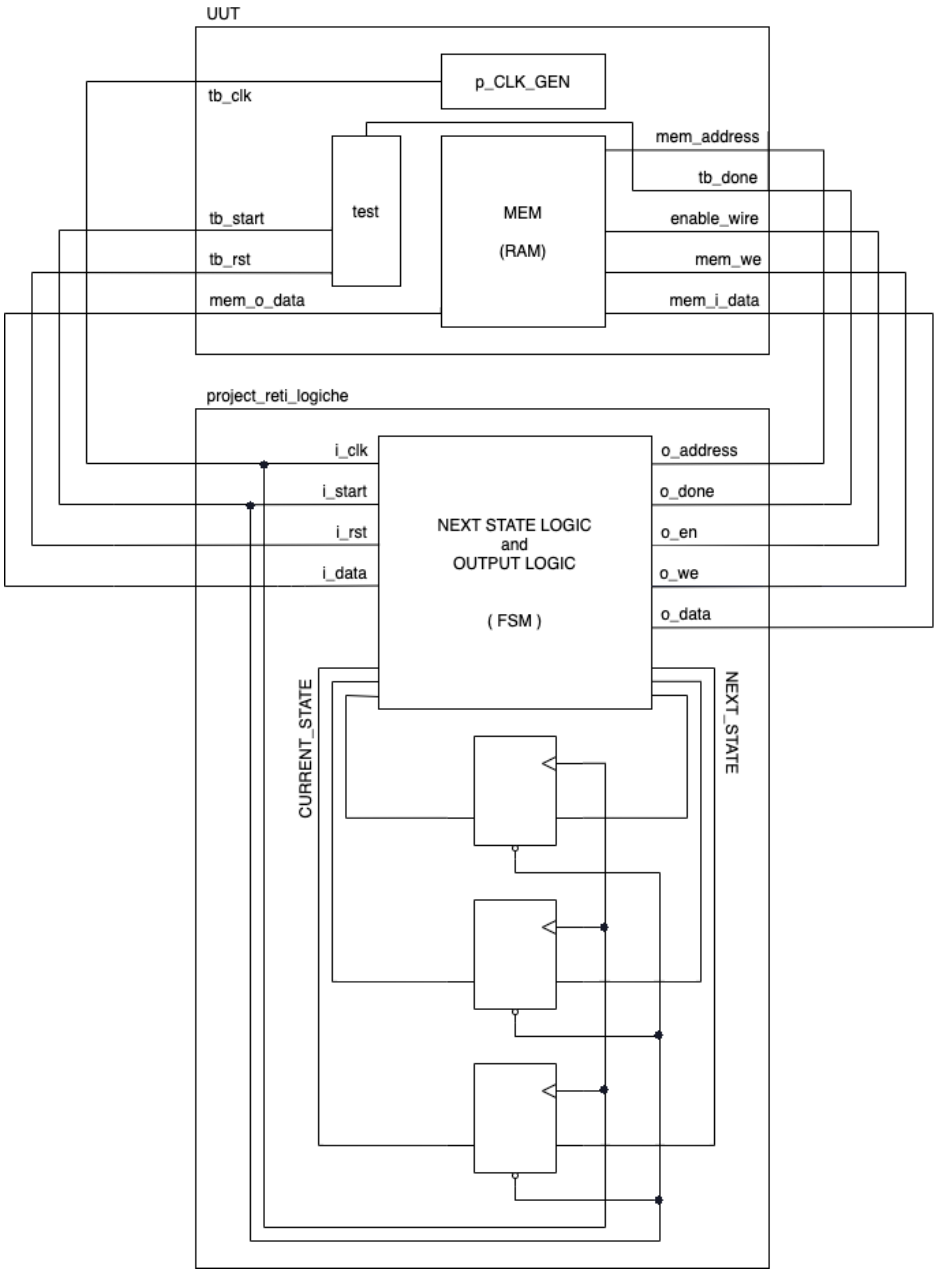


Figure 1: Schema in moduli

L'architettura segue il paradigma della *Macchina di Mealy* — dove l'uscita dipende dalla transizione eseguita — con 2 processi logici distinti (*Next State Logic* e *Output Logic*). La struttura presenta 3 *flip-flop* che mantengono lo stato corrente e accolgono quello successivo, disponendo di 8 possibili stati diversi tra loro descritti nel dettaglio in successione.

Lo schema sopra riportato descrive come *black-box* i moduli omettendo i collegamenti *intra-modulo* per semplicità di lettura.

2.1.1 Next State Logic

Nome *process* nel codice : `NEXT_STATE`

In questa unità viene implementata la logica che cambia lo stato corrente a seconda delle condizioni che si verificano nel corso dell'esecuzione di uno stato. Questo modulo può solo leggere i signal di input e di output.

2.1.2 Output Logic

Nome *process* nel codice : `CURRENT_STATE`

In questa unità viene implementata la logica che stabilisce l'uscita verso lo stato indicato ed elaborato dal modulo di *Next State Logic*.
Questo modulo può sia leggere i signal di input che scrivere quelli di output.

2.1.3 Corrispondenza I/O

Di seguito viene riportata in maniera più chiara la corrispondenza che intercorre tra il pilotaggio dell'unità di *test bench* e il componente `project_reti_logiche`.

Come da disegno gli output del componente di *Test Bench* diventano gli input di `project_reti_logiche` e viceversa.

```
i_clk      :   tb_clk
i_start    :   tb_start
i_rst      :   tb_rst
i_data     :   mem_o_data
o_address  :   mem_address
o_done     :   tb_done
o_en       :   enable_wire
o_we       :   mem_we
o_data     :   mem_i_data
```

2.2 Macchina a stati

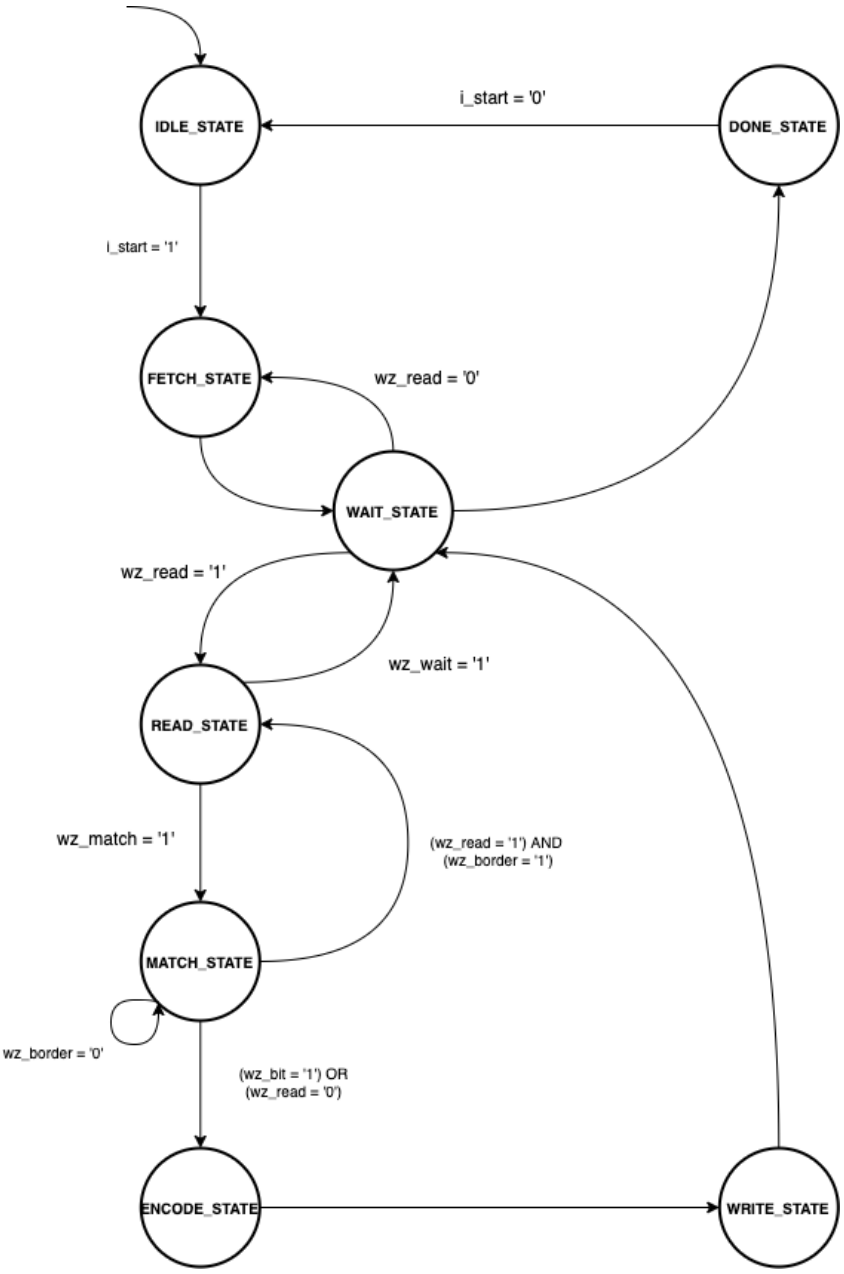


Figure 2: Macchina a stati

Attenzione : la FSM, in qualsiasi stato si trovi, andrà sempre in `IDLE_STATE` al ciclo di clock successivo se si è in presenza del signal `i_rst = '1'`.

2.2.1 `IDLE_STATE`

Stato di reset utilizzato per l'avvio e la ripetizione di una elaborazione della FSM *ex novo*: una volta giunta al termine in `DONE_STATE` ottenendo `o_done = '1'` e dopo aver ricevuto `i_start = '0'` .

2.2.2 `FETCH_STATE`

Identifica il caricamento del signal `to_conv` posto all'indirizzo `0x0000`; per farlo sfrutta la transizione in uno stato di `WAIT_STATE` in modo tale da caricare l'indirizzo in `o_address` nel primo ciclo di clock, mentre nel secondo di leggerne il contenuto attraverso `i_data`.

2.2.3 `WAIT_STATE`

Stato di intermezzo utilizzato da `FETCH_STATE`, `READ_STATE` e `WRITE_STATE` in modo tale da sopperire all'esigenza di lettura e scrittura in memoria richiedente l'esecuzione di più fronti di clock.

2.2.4 `READ_STATE`

Una volta giunti nello stato corrente avremo il valore da confrontare in `to_conv`. Qui verrà letto uno per volta il valore fondamentale di ogni *Working Zone* con `wz_offset = 0001` attraverso l'ausilio dello stato `WAIT_STATE`, fino a quando non si verificherà la condizione di appartenenza.
Inoltre qui vengono resettati alcuni signal utili per il `MATCH_STATE` , ad esempio `wz_border`.

2.2.5 `MATCH_STATE`

L'algoritmo implementato consiste nel confrontare `to_conv` con i rispettivi `wz_offset` di ciascuna *Working Zone*, in maniera ciclica sui fronti di salita del clock.

Se `wz_border = '0'` significa che lo stato sta ancora confrontando i `wz_offset` rispetto alla *Working Zone* analizzata; pertanto la FSM permane nel suo stato di `MATCH_STATE`.
Se in fase di confronto verrà verificata l'appartenenza ad una *Working Zone*, verrà settato `wz_bit = '1'` in modo tale da ottenere una transizione verso `ENCODE_STATE` in maniera anticipata all'iterazione di *matching* in corso.
Se sono stati terminati i confronti possibili arrivando a `wz_offset = 1000` significa che è stata raggiunta la condizione di *bordo* (`wz_border = 1`) e se `wz_num` $<$ `"111"` allora si potrà continuare a leggere (`wz_read = '1'`) tornando a `READ_STATE`, altrimenti se `wz_num = "111"` non si potrà continuare a leggere ulteriormente (`wz_read = '0'`) andando in stato di `ENCODE_STATE`.

2.2.6 ENCODE_STATE

Dopo MATCH_STATE verrà utilizzato il risultato ottenuto in wz_bit per stabilire quale tipo di codifica è opportuna per il valore letto in to_conv.

2.2.7 WRITE_STATE

Impostando i signal o_en = '1' e o_we = '1' verrà effettuata la scrittura in memoria all'indirizzo o_address = 0x0009 e per farlo la FSM si appoggerà a WAIT_STATE, per poi procedere a DONE_STATE una volta che la scrittura sarà avvenuta con successo e con conseguente impostazione del signal o_done = '1'.

2.2.8 DONE_STATE

Giunti al termine dell'elaborazione la FSM rimarrà in questo stato fino a quando non verrà ricevuto il signal i_start = '0'.

3 Risultati sperimentali

3.1 Sintesi

Il processo di sintesi termina con successo per ciascuno dei casi effettuati sottostante; di seguito, infatti, vengono mostrati i relativi report.

L'unico warning che compare risulta essere : `No constraints selected for write.` Del tutto prevedibile e ininfluente al fine del funzionamento complessivo.

3.2 Simulazioni

Avendo testato con successo l'efficacia di appartenenza per ogni wz_num e per ogni wz_offset al dominio possibile, di seguito vengono elencati i report di due casi limite:

a. **Limite inferiore** :
   Appartenenza con wz_num = "000" e wz_offset = "0001". In questo caso la codifica avviene nel minor tempo possibile e raggiungibile dalla FSM, ovvero al primo slot.

b. **Limite superiore** :
   Appartenenza con wz_num = "111" e wz_offset = "1000". In termini di tempistiche questo caso somiglia alla non appartenenza del *test bench* assegnato, cosa del tutto in linea con le previsioni, con l'unica eccezione in fase di codifica.

c. **Reset** : Come da definizione il componente risponde al segnale di interrupt i_rst tornando allo stato IDLE_STATE in qualsiasi stato corrente si trovi al termine della transizione.

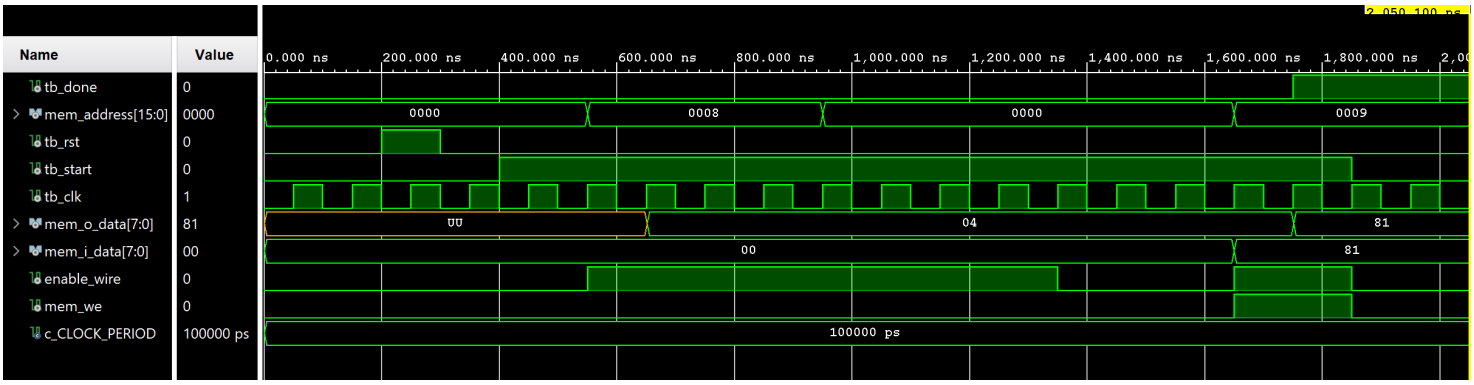


Figure 3: Caso (a) — Tempo minimo

- *Behavioural Simulation* : 1,950,000 ps
- *Post-Synthesis Functional Simulation* : 2,050,100 ps
- *Post-Synthesis Timing Simulation* : 2,053,737 ps

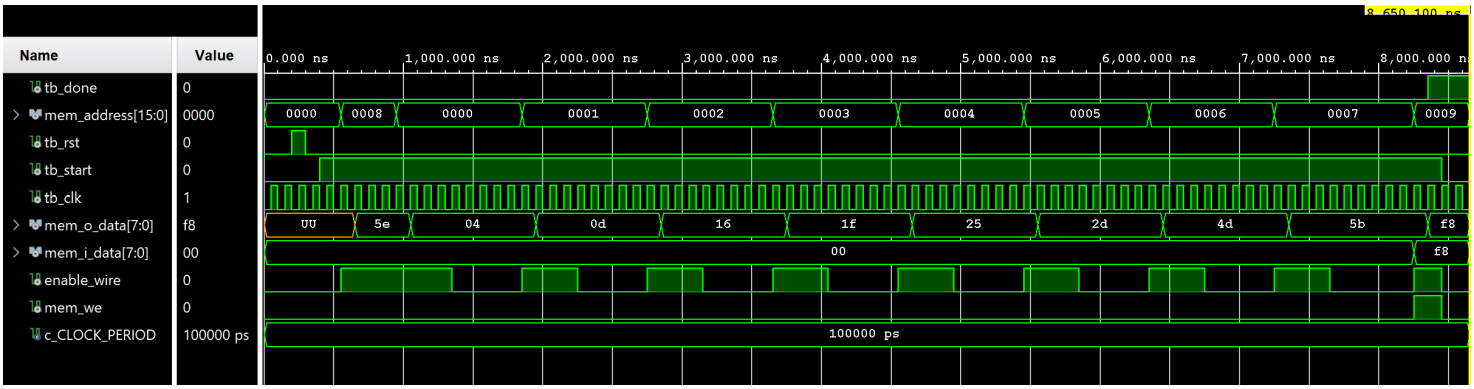


Figure 4: Caso (b) — Tempo massimo

- *Behavioural Simulation* : 8,550,000 ps
- *Post-Synthesis Functional Simulation* : 8,650,100 ps
- *Post-Synthesis Timing Simulation* : 8,653,737 ps

4 Test Bench

Di seguito vengono riportati i risultati dei due *test bench* assegnati di appartenenza e di non appartenenza in seguito alla sintesi — *post-sinthesys* — in modalità funzionale.

4.1 Valore non presente in nessuna Working Zone

Il seguente *test-bench* verifica la condizione di non appartenenza controllando per ogni `wz_num` delle *Working Zone* presenti, ciascun `wz_offset`.

E' possibile notare come in memoria vengano richiesti tutti gli 8 indirizzi base delle *Working Zone* in seguito al caricamento del valore iniziale in `to_conv = "0x2A"`, per poi riscrivere in memoria — al termine degli esiti negativi di tutti i confronti sopra elencati — all'indirizzo `0x0009` lo stesso valore invariato.

Quindi verrà ricavato:

```
conv <= wz_bit & to_conv
conv <= '0' & "0101010"
conv <= "00101010"
ovvero : bin 00101010 = dec 42 = hex 2A
```

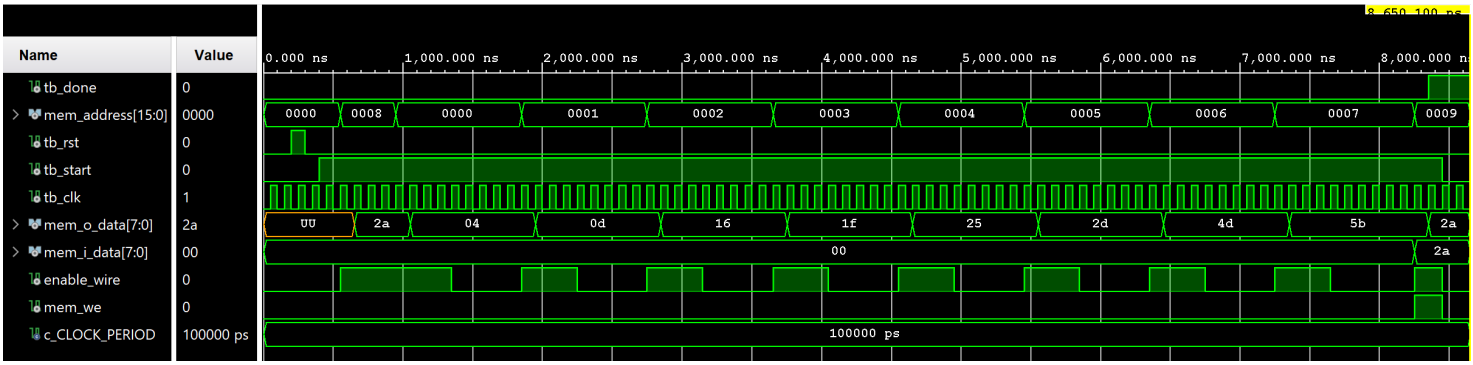


Figure 5: Caption

- *Behavioural Simulation* : 8,500,000 ps
- *Post-Synthesis Functional Simulation* : 8,650,100 ps
- *Post-Synthesis Timing Simulation* : 8,653,737 ps

4.2 Valore presente in una Working Zone

Il seguente *test bench* verifica la condizione di appartenenza controllando fino a `wz_num = "011"` con `wz_offset = "0100"`; ovvero caricando il valore `hex 21 = dec 33` in `to_conv` e riscontrando l'appartenenza con la terza *Working Zone* con shift "+ 2" rispetto al valore base di `hex 1F = dec 31`.

Quindi verrà ricavato:

```
conv <= wz_bit & wz_num & wz_offset
conv <= '1' & "011" & "0100"
conv <= "10110100"
```
ovvero : `bin 10110100 = dec 180 = hex B4`

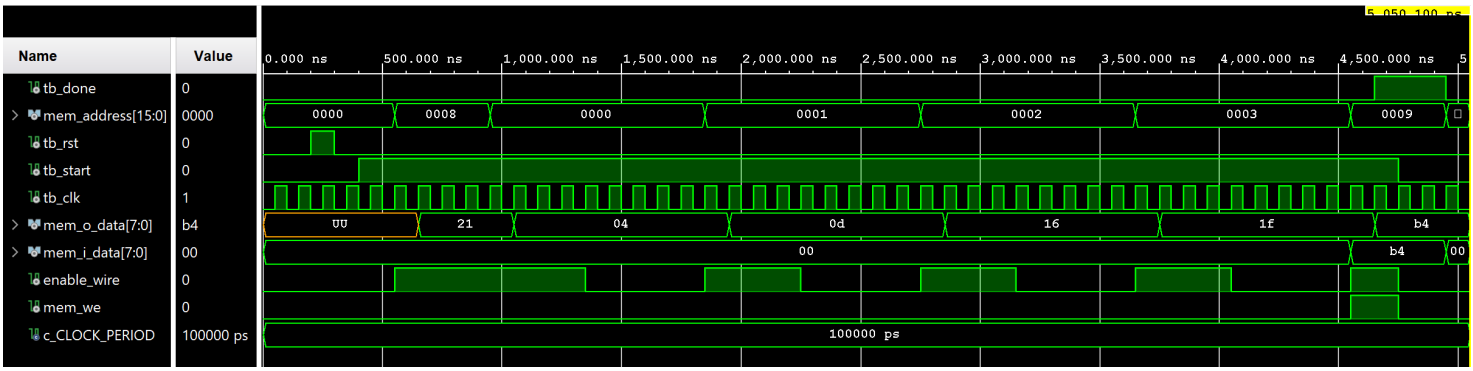


Figure 6: Caption

- *Behavioural Simulation* : 4,950,000 ps
- *Post-Synthesis Functional Simulation* : 5,050,100 ps
- *Post-Synthesis Timing Simulation* : 5,053,737 ps

5 Conclusioni

In fase di sviluppo sono sorte diverse idee di approccio nello stabilire la FSM ottimale: inizialmente la soluzione prevedeva il caricamento di tutte e 8 le *Working Zone* in una sorta di memoria interna al componente `project_reti_logiche`. Strada poi abbandonata per far spazio al *mono-caricamento* di una *Working Zone* per volta; tuttavia all'inizio questo approccio è stato utile per capire meglio come interfacciarsi con la memoria.
Questa soluzione, infatti, ha presentato notevoli miglioramenti in termini di prestazioni — temporali e spaziali — nei casi in cui l'appartenenza fosse stata verificata anzitempo il controllo totale delle *Working Zone*.

Infine è stato aggiunto come ultima ottimizzazione lo stato `WAIT_STATE` utile in fase di lettura e scrittura in memoria per la gestione dei clock necessari alla finalizzazione stessa delle operazioni, cosa che prima veniva gestita negli stati di `FETCH_STATE`, `READ_STATE`, `WRITE_STATE` attraverso dei contatori.
Ritengo, quindi, che la sua introduzione apporti, oltre che maggior robustezza, anche migliore chiarezza e modularità al componente finale progettato.