Prova Finale di Reti Logiche

Lampis Andrea - Matricola n. 888390

Anno Accademico 2019/2020

Politecnico di Milano

Indice

1 Introduzione

Nel presente documento viene riportata la relazione e documentazione relativa al progetto di Reti Logiche A.A. 2019/2020. Verrà innanzitutto presentato un riassunto della specifica, per poi passare alle scelte progettuali prese durante l'implementazione, ed infine arrivare alla presentazione del report dei test utilizzati al fine di verificare il corretto funzionamento.

1.1 Specifica

La specifica della Prova finale (Progetto di Reti Logiche) 2020 è ispirata al metodo di codifica a bassa dissipazione di potenza denominato "Working Zone". Essa è un metodo di codifica utilizzato per trasformare il valore di un indirizzo quando questo viene trasmesso sul Bus Indirizzi, se appartiene a certi intervalli (detti appunto Working Zone, d'ora in avanti anche abbreviate WZ). Una WZ è definita come un intervallo di indirizzi di dimensione fissa (Dwz) che parte da un indirizzo base. All'interno dello schema di codifica possono esistere multiple working-zone (Nwz).

Lo schema modificato di codifica da implementare è il seguente:

- se l'indirizzo da trasmettere (`ADDR`) non appartiene a nessuna Working Zone, esso viene trasmesso così come è, e un bit addizionale rispetto ai bit di indirizzamento (`WZ_BIT`) viene messo a 0. In pratica dato `ADDR`, verrà trasmesso `WZ_BIT=0` concatenato ad `ADDR` (`WZ_BIT & ADDR`, dove `&` è il simbolo di concatenazione);

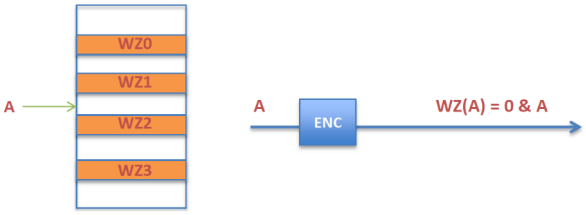

Figura 1: ADDR non appartenente a nessuna Working Zone

- se l'indirizzo da trasmettere (`ADDR`) appartiene ad una Working Zone, il bit addizionale `WZ_BIT` è posto a 1, mentre i bit di indirizzo vengono divisi in 2 sotto campi rappresentanti:
  - Il numero della working-zone al quale l'indirizzo appartiene `WZ_NUM`, che sarà codificato in binario
  - L'offset rispetto all'indirizzo di base della working zone `WZ_OFFSET`, codificato come one-hot (cioè il valore da rappresentare è equivalente all'unico bit a 1 della codifica).

  In pratica dato `ADDR`, verrà trasmesso `WZ_BIT=1` concatenato ad `WZ_NUM` e `WZ_OFFSET` (`WZ_BIT & WZ_NUM & WZ_OFFSET`, dove `&` è il simbolo di concatenazione);

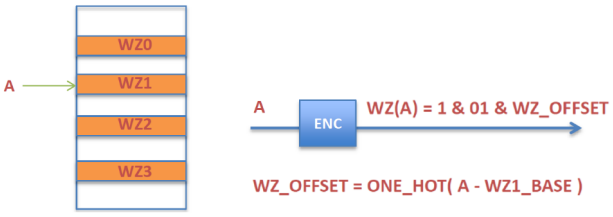

Figura 2: ADDR appartenente a una Working Zone

Nella versione da implementare il numero di bit da considerare per l'indirizzo da codificare è 7. Il che definisce come indirizzi validi quelli da 0 a 127. Il numero di working-zone è 8 (`Nwz=8`) mentre la dimensione della working-zone è 4 indirizzi incluso quello base (`Dwz=4`). Questo comporta che l'indirizzo codificato sarà composto da 8 bit: 1 bit per `WZ_BIT` + 7 bit per `ADDR`, oppure 1 bit per `WZ_BIT`, 3 bit per codificare in binario a quale tra le 8 working zone l'indirizzo appartiene, e 4 bit per codificare one hot il valore dell'offset di `ADDR` rispetto all'indirizzo base. Il modulo da implementare leggerà l'indirizzo da codificare e gli 8 indirizzi base delle working-zone e dovrà produrre l'indirizzo opportunamente codificato.

Esempio:
La seguente sequenza di numeri mostra un esempio del contenuto della memoria al termine di una elaborazione. I valori che qui sono rappresentati in decimale, sono memorizzati in memoria con l'equivalente codifica binaria su 8 bit senza segno.

Caso 1[1]		**Caso 2**[2]		Commento
Indirizzo Memoria	Valore	Indirizzo Memoria	Valore	
0	4	0	4	Indirizzo base WZ 0
1	13	1	13	Indirizzo base WZ 1
2	22	2	22	Indirizzo base WZ 2
3	31	3	31	Indirizzo base WZ 3
4	37	4	37	Indirizzo base WZ 4
5	45	5	45	Indirizzo base WZ 5
6	77	6	77	Indirizzo base WZ 6
7	91	7	91	Indirizzo base WZ 7
8	42	8	33	ADDR da codificare
9	42	9	180[3]	Valore codificato in output

[1] con valore non presente in nessuna working zone

[2] con valore presente in una working zone

[3] 1 - 011 - 0100

2 Architettura

La specifica progettuale si traduce nel realizzare un'implementazione in grado di leggere i valori contenuti nelle working zone e l'indirizzo da codificare, quindi verificare se quest'ultimo è presente o meno in una WZ ed effettuare di conseguenza l'opportuna codifica.

2.1 Macchina a Stati Finiti

Il funzionamento alla base del componente è stato implementato attraverso una FSM che evolve di stato in stato in modo sequenziale ad ogni ciclo di clock (-) oppure al variare dei segnali di ingresso `i_start` e `i_rst`. Ciò che ogni stato rappresenta è spiegato più nel dettaglio in tabella 1.

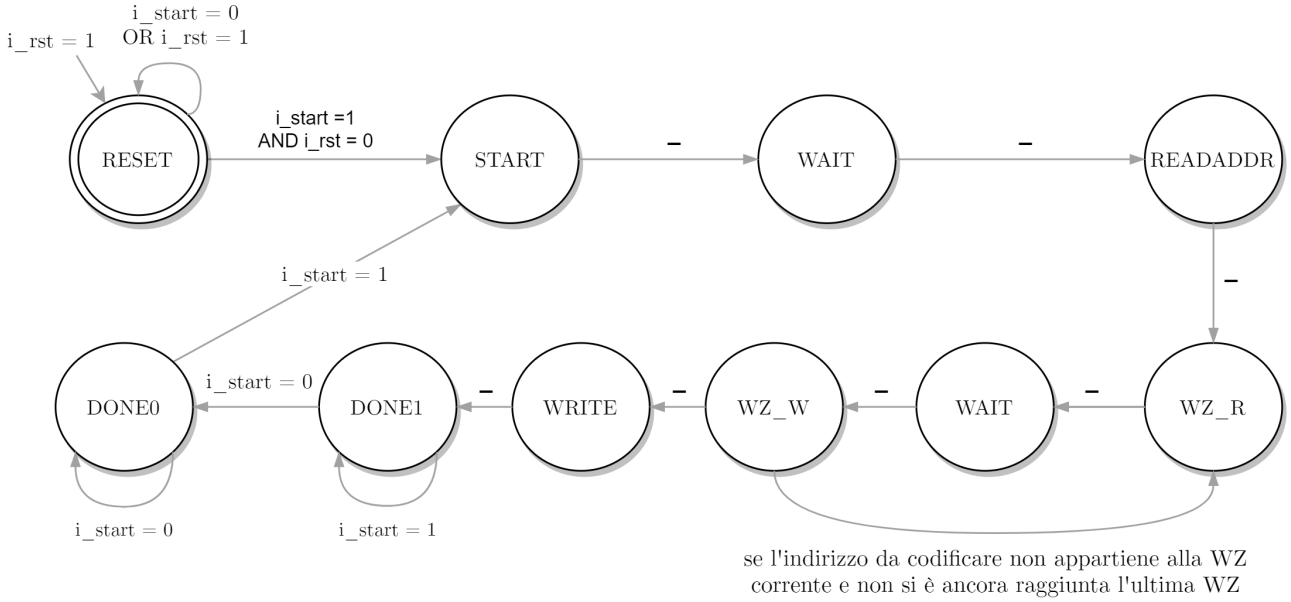

Figura 3: Macchina a Stati Finiti implementata

Tabella 1: Stati della FSM

`RESET`	Stato di partenza della FSM e stato in cui si andrà in presenza di un segnale `i_rst` (sincrono o asincrono). Alla ricezione di un segnale `i_start` si passa allo stato `START`.
`START`	Stato iniziale. In questo stato viene richiesta la lettura dalla RAM dell'indirizzo da codificare.
`WAIT`	Stato di attesa per garantire alla memoria il tempo di inviare il dato richiesto.
`READADDR`	Stato in cui il componente legge (riceve) e salva in un registro l'indirizzo da codificare (ADDR).
`WZ_R`	Stato nel quale viene richiesta la lettura dalla RAM dell'indirizzo contenuto nella WZ corrente.
`WZ_W`	Stato nel quale avviene il confronto tra valore da codificare (ADDR) e valore contenuto nella WZ corrente. Se: • ADDR appartiene alla WZ, effettua la codifica e passa allo stato `WRITE` • ADDR non appartiene alla WZ e si è raggiunta l'ultima WZ, effettua la codifica per indirizzi non appartenenti ad una WZ • ADDR non appartiene alla WZ, passa alla WZ successiva
`WRITE`	Stato nel quale viene scritto in output il valore codificato.
`DONE1`	Stato in cui si segnala che il risultato è stato scritto in RAM: `o_done` è portato ad '1'. Alla ricezione di `i_start` uguale a '0' si porta la macchina in `DONE0` per una possibile successiva elaborazione.
`DONE1`	Stato nel quale si rimane in attesa di una possibile successiva elaborazione: `o_done` è portato a '0'. Alla ricezione di `i_start` uguale a '1' si riporta la macchina in `START`.

2.2 Implementazione

A livello implementativo si è optato per una FSM *single-process*, in modo da realizzare un algoritmo di semplice stesura, interpretazione e manutenzione, evitando così anche eventuali problemi di sincronizzazione tra più processi.

2.2.1 Interfaccia

L'interfaccia del componente è stata presentata nelle specifiche nel seguente modo:

```
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;
```

Figura 4: Interfaccia

Tale componente si interfaccia, inoltre, con un chip RAM in cui sono caricati tutti i dati necessari all'elaborazione.

2.2.2 Segnali

Sono stati definiti i seguenti segnali:

```
type state_type is (RESET_STATE,      -- Stato di reset della FSM
                    START_STATE,      -- Stato iniziale della FSM
                    WAIT_STATE,       -- Stato di attesa per il corretto caricamento dei dati in memoria
                    READADDR_STATE,   -- Stato nel quale avviene la lettura dell'indirizzo da codificare
                    WZ_R_STATE,       -- Stato nel quale viene richiesta la Working Zone desiderata (incrementato con un contatore)
                    WZ_W_STATE,       -- Stato nel quale si riceve il dato contenuto nella WZ richiesta. Se necessario, effettua la codifica
                    WRITE_STATE,      -- Stato nel quale viene scritto in uscita l'indirizzo codificato
                    DONE1_STATE,      -- Stato nel quale viene segnalato il termine della codifica
                    DONE0_STATE);     -- Stato nel quale ci si trova al termine della codifica. La FSM è pronta per iniziare una nuova codifica
signal state : state_type;

type p_state_type is (START_STATE, WZ_R_STATE); -- previous_state_type: utilizzati quando è necessario aspettare un ciclo di clock
signal p_state : p_state_type;
```

Figura 5: Segnali

I segnali definiti sono quindi utilizzati per la definizione degli stati della macchina a stati.

2.2.3 Variabili

Poiché la FSM realizzata è mono-processo, si è fatto un discreto utilizzo delle variabili:

```
variable count : integer range 0 to 7 := 0;          -- memorizza lo stato di avanzamento delle WZ
variable addr : integer range 0 to 127;              -- memorizza l'indirizzo da codificare
variable offset : integer range 0 to 3 := 0;         -- memorizza (come intero) l'offset dell'indirizzo codificato
variable WZ_offset : std_logic_vector(3 downto 0);   -- memorizza (come vettore) l'offset dell'indirizzo codificato
variable WZ_num : std_logic_vector(2 downto 0);      -- memorizza (come vettore) il numero della WZ di appartenenza di ADDR
variable i_data_int : integer range 0 to 127;        -- memorizza (come intero) il valore letto dalla WZ corrente
```

Figura 6: Variabili

2.2.4 Algoritmo

Gli stati della FSM e il loro avanzamento sequenziale sono già stati descritti nella sezione "Macchina a Stati Finiti" a pagina 4.

Vediamo ora un riassunto dell'algoritmo di esecuzione:

1. Si accende la macchina e ci si pone subito nello stato di RESET, nel quale si rimane finché non viene ricevuto un segnale di start
2. Nello stato di START viene attivata la comunicazione con la memoria e viene richiesto il dato contenuto in RAM(8), ovvero l'indirizzo ADDR da codificare
3. Viene ricevuto e salvato in una variabile il dato richiesto (ADDR)
4. Per ogni working zone:
   (a) viene richiesto il dato contenuto nella working zone corrente
   (b) viene ricevuto il dato richiesto
   (c) viene confrontato tale dato con ADDR per capire se quest'ultimo appartiene o meno a tale WZ
   (d) se appartiene, viene codificato come da specifica. Altrimenti si passa alla WZ successiva.
   In caso si sia raggiunta l'ultima WZ, ADDR viene codificato come indirizzo fuori working-zone
5. Quando ADDR viene codificato, si interrompe il ciclo di lettura delle WZ (se non si è ancora raggiunta l'ultima) e si procede a scrivere in output su RAM(9) il valore codificato
6. Viene quindi segnalato che il dato è stato scritto e che è possibile iniziare una eventuale nuova codifica

N.B: in caso di ricezione del segnale di reset, si ritorna immediatamente nello stato RESET interrompendo l'elaborazione corrente.

Vediamo ora nel dettaglio cosa accade quando viene effettuato il confronto tra ADDR e la WZ corrente (ciò avviene nello stato `WZ_W_STATE`):

```
1  when WZ_W_STATE =>
2
3    i_data_int := to_integer(unsigned(i_data));
4    if (addr >= i_data_int) and (addr < i_data_int+4) then
5      offset := (addr - i_data_int);
6      WZ_num := std_logic_vector(to_unsigned(count, 3));
7      WZ_offset := (others => '0');
8      WZ_offset(offset) := '1';
9      o_data <= '1' & WZ_num & WZ_offset;
10     o_address <= std_logic_vector(to_unsigned(9 , 16));
11     state <= WRITE_STATE;
12
13   elsif count = 7 then
14     o_data <= std_logic_vector(to_unsigned(addr, 8));
15     o_address <= std_logic_vector(to_unsigned(9 , 16));
16     state <= WRITE_STATE;
17
18   else
19       count := (count + 1);
20       state <= WZ_R_STATE;
21   end if;
```

Figura 7: WZ_W_STATE

Linea 3. Lettura, conversione ad intero e salvataggio del valore contenuto nella WZ corrente

Linea 4. Viene controllato se ADDR appartiene alla WZ corrente, sfruttando il fatto che, da specifica, la dimensione di una Working Zone è 4 indirizzi (`Dwz=4`). Se appartiene...

- Linea 5. Viene calcolato l'offset (intero) come sottrazione tra il valore di ADDR l'indirizzo base della WZ
- Linea 6. Viene convertito il numero della WZ da intero a vettore logico di 3 bit, utilizzando la variabile `count` che tiene memorizzato il numero della WZ corrente
- Linea 7. Viene inizializzato a 0 il vettore logico dell'offset
- Linea 8. Il bit in posizione `offset`-esima viene posto a 1. Questo realizza la codifica one-hot dell'offset
- Linea 9. Viene costruito l'indirizzo codificato, come concatenazione dei vettori logici riportati (come da specifica)
- Linea 10. Viene indicato su quale indirizzo della RAM andare a scrivere il valore codificato
- Linea 11. Ci si sposta nello stato `WRITE_STATE`

Linea 13. Se si è raggiunta l'ultima WZ e non si è ancora trovato un riscontro con le WZ...

- Linea 14. Viene posto in output ADDR così com'è stato letto, senza alcuna codifica ulteriore
- Linea 15. Viene indicato su quale indirizzo della RAM andare a scrivere il valore in output
- Linea 16. Ci si sposta nello stato `WRITE_STATE`

Linea 18. Se non si è trovato un riscontro con la WZ corrente e non si è ancora raggiunta l'ultima WZ...

- Linea 19. Si incrementa di uno il valore della variabile `count`
- Linea 20. Si passa allo stato `WZ_R_STATE`, nel quale si andrà a richiedere la lettura della WZ successiva, per poi ritornare nuovamente in questo stato (`WZ_W_STATE`)

---

3 Sintesi

3.1 Report di sintesi

Analizzando il "Vivado Synthesis Report" troviamo che la FSM è stata codificata in one-hot, per migliorarne l'efficienza:

```
-----------------------------------------------------------------------------------------------------------
                State |                     New Encoding |               Previous Encoding
-----------------------------------------------------------------------------------------------------------
          reset_state |                        000000001 |                         0000
          start_state |                        000000010 |                         0001
           wait_state |                        000000100 |                         0010
      readaddr_state |                        000001000 |                         0011
           wz_w_state |                        000010000 |                         0101
          write_state |                        000100000 |                         0110
          done1_state |                        001000000 |                         0111
          done0_state |                        010000000 |                         1000
           wz_r_state |                        100000000 |                         0100
-----------------------------------------------------------------------------------------------------------
```

Dal report di sintesi possono essere inoltre estratte le seguenti informazioni riguardo i registri e i mux:

```
+---Adders :
   3 Input      9 Bit       Adders := 1
   2 Input      3 Bit       Adders := 1
   3 Input      2 Bit       Adders := 1

+---Registers :
   16 Bit    Registers := 1
   8 Bit     Registers := 1
   7 Bit     Registers := 1
   3 Bit     Registers := 1
   1 Bit     Registers := 4

+---Muxes :
   9 Input     16 Bit        Muxes := 1
   9 Input      9 Bit        Muxes := 1
   2 Input      9 Bit        Muxes := 6
   2 Input      8 Bit        Muxes := 1
   9 Input      3 Bit        Muxes := 1
   2 Input      1 Bit        Muxes := 1
   3 Input      1 Bit        Muxes := 1
   9 Input      1 Bit        Muxes := 8
```

3.2 Area occupata

Eseguendo un "Report Utilization" vediamo ora l'area occupata dal design sintetizzato.
E' da notare che si è voluta ottimizzare l'area occupata dal componente, non tanto per limiti di disponibilità della FPGA utilizzata[1] (che come possiamo vedere, i valori di utilizzo hanno svariati ordini di grandezza in meno rispetto alla disponibilità della FPGA), ma in previsione di poter utilizzare tale algoritmo su FPGA molto più piccole.

Risorsa	**Utilizzo**	**Disponibilità**	**Utilizzo in %**
Look Up Table	41	134600	0.03%
Flip Flop	35	269200	0.01%

Tabella 2: Report di utlizzo

3.3 Report di timing

Analizzando il report di timing si può vedere quanto è veloce in un singolo ciclo di clock il design sintetizzato. Si è ottenuto con il clock della specifica di 100ns un Worst Negative Slack pari a 95,354ns. Da questo valore, sapendo anche il ritardo di riposta della RAM (T_{RAM}), possiamo calcolare il periodo minimo applicabile al design creato:

$$T_{min} = T_{curr} - WNS + T_{RAM} = 100ns - 95.354ns + 1ns = 5.646ns$$

Il design creato ha quindi una massima frequenza di clock pari a: $f_{max} = 1/T_{min} \approx 177.1Mhz$.

[1] xc7a200tfbg484-1

4 Simulazioni

Per verificare il corretto funzionamento del design, sono stati creati dei test bench appositi al fine di testare il componente sia nei normali casi di utilizzo sia nei casi limite. Di seguito sono riportati i test bench più significativi.

4.1 Test Bench 0 (forniti con la specifica)

In questi test bench viene verificato il corretto funzionamento nei due casi base, ovvero l'appartenenza o meno dell'indirizzo da codificare all'interno di una working-zone.

4.1.1 Behavioral Simulation

In questo primo test, l'indirizzo da codificare è presente nella Working Zone 3. Come possiamo vedere dalla waveform, per prima cosa viene letto l'indirizzo da codificare e successivamente si passa alla lettura delle WZ. Arrivati alla terza WZ, viene individuato un riscontro e pertanto viene effettuata la codifica e la scrittura, quindi viene segnalato il termine dell'elaborazione.

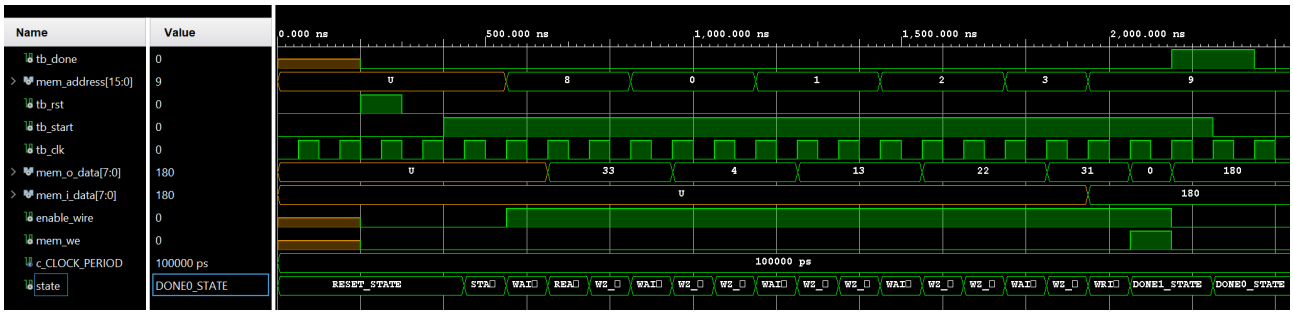

Figura 8: L'indirizzo da codificare è presente nella Working Zone 3

Nel secondo test, l'indirizzo da codificare (ADDR) non è presente in alcuna Working Zone. Pertanto, vengono lette tutte le WZ e quindi ADDR viene scritto in output così com'è stato letto (come da specifica).

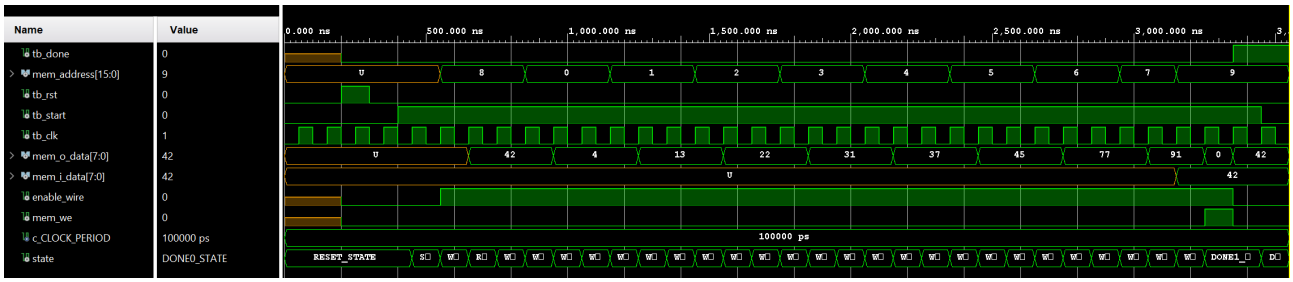

Figura 9: L'indirizzo da codificare non è presenta in alcuna Working Zone

4.1.2 Post-Synthesis Functional Simulation

Come possiamo vedere, ripetendo in post-sintesi il test bench rappresentato dalla Figura 8, i valori che prima risultavano non inizializzati ("U") ora spariscono e vengono rimpiazzati da valori reali, questi valori sono ininfluenti e sono dati dalle ottimizzazioni fatte dal tool di sintesi.

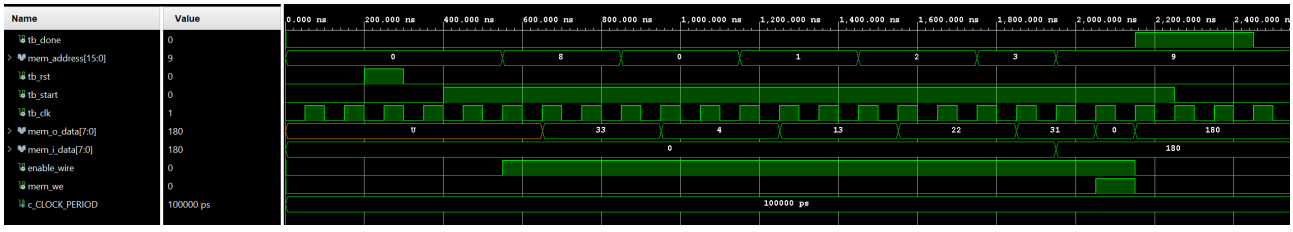

Figura 10: Waveform dei segnali in Post-Synthesis Timing Simulation. Test bench della Figura 8

4.2 Test Bench 1: Multi Start

Si è poi effettuato un test per verificare il corretto funzionamento nel caso di conversioni in sequenza, senza reset delle Working Zone.

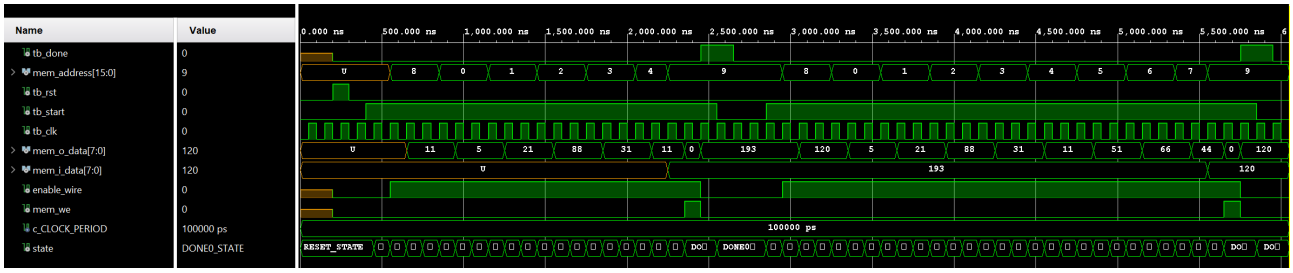

Figura 11: Segnali di start multipli

In questo caso, nel primo start viene trovato un riscontro tra ADDR e la WZ numero 3, mentre nel secondo start non viene trovato alcun riscontro.

4.3 Test Bench 2: Multi Reset

Un ulteriore test effettuato è quello nel caso in cui si verifichi un reset al termine della prima elaborazione, prima del secondo start. Da specifica, solo tramite reset si può verificare un cambiamento delle WZ. Questo test verifica la corretta lettura delle nuove Working Zone.

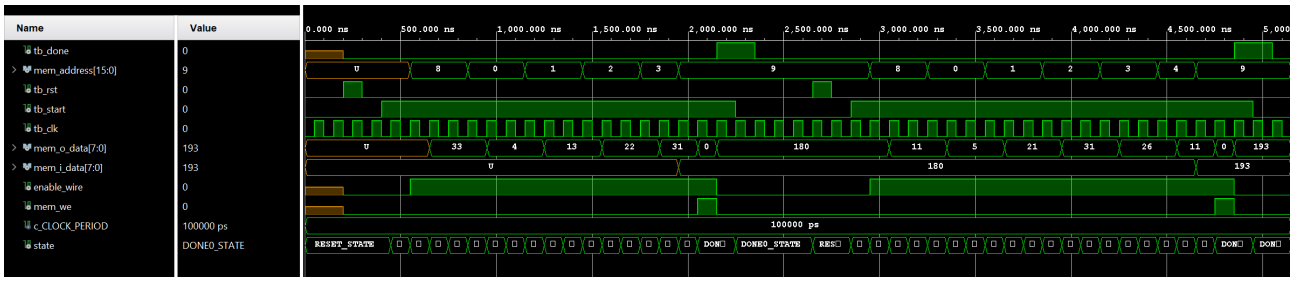

Figura 12: Segnali di reset multipli

In questo caso, in entrambi gli start viene trovato un riscontro tra ADDR e una WZ, ma prima del secondo start è stato effettuato un reset, con conseguente cambio delle WZ.

4.4 Test Bench 3: Reset Asincrono

Un particolare caso limite è quello che si genera quando viene ricevuto un segnale di reset in modo asincrono durante una elaborazione in corso, iniziata a seguito di uno start.
Come possiamo vedere dal test, il componente ha inizialmente avviato la normale elaborazione, leggendo prima ADDR e poi ciclando sulle varie WZ. Alla ricezione del segnale di reset, il componente è ritornato nello stato di `RESET` per poi iniziare da capo l'elaborazione, in quanto il segnale di start è nuovamente alto.

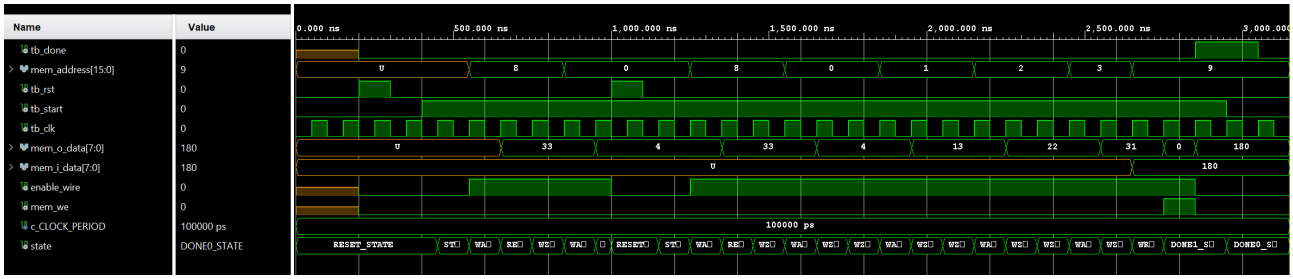

Figura 13: Segnali di reset asincrono

4.5 Altri test

Sono inoltre stati effettuati i seguenti test:

- Working Zone adiacenti, per verificare che il componente riconosca la WZ corretta
- Reset Asincroni multipli, di breve durata, per verificare il corretto comportamento anche in condizioni di stress
- Unione di tutti i test precedenti con centinaia di elaborazioni (WZ generate in modo randomico), per verificare contemporaneamente tutte le casistiche durante un'elaborazione prolungata

---

5 Conclusione

Riassumendo, si è creato un design con le seguenti caratteristiche:

- Funzionante in pre e post-sintesi (Functional e Timing).
- Ottimizzato in modo da ridurre il più possibile l'area occupata.
- Frequenza massima di clock impostabile a 177.1Mhz.
- Utilizzo di LUT pari al 0.03%.
- Utilizzo di FF pari al 0.01%

5.1 Nota aggiuntiva sull'ottimizzazione

Come già indicato più volte, si è deciso di seguire la scelta progettuale di ottimizzare l'area occupata dai componenti di memoria, nell'ottica di rendere possibile l'utilizzo di tale componente su FPGA di dimensioni ben più ridotte di quella utilizzata in fase di progettazione.

Si è però consapevoli che un'altra ottimizzazione possibile fosse quella di migliorare il componente dal punto di vista delle prestazioni, a discapito dell'area occupata. Infatti, il componente realizzato non mantiene salvate in memoria le Working Zone lette durante l'elaborazione, ma le rilegge ad ogni nuovo start. Poiché da specifica le WZ possono cambiare solo a seguito di un reset, sarebbe stato possibile salvarle alla prima elaborazione e quindi riutilizzarle ad ogni nuova elaborazione, dovendo così leggere dalla RAM solamente l'indirizzo ADDR da codificare. In caso di reset, queste sarebbero state rilette e salvate nuovamente.
A scopo didattico, si è provato a realizzare anche questa seconda implementazione, ottenendo un componente di area quadrupla rispetto alla prima implementazione, con un leggero miglioramento di prestazioni in caso di migliaia di elaborazioni senza reset.