POLITECNICO
MILANO 1863

Prova finale di Reti Logiche
Anno accademico 2019/2020
Professore: Salice Fabio

Leoni Luca, 889638, 10574709
Locarno Silvia, 889442, 10560587

1. Introduzione

Il seguente documento rappresenta il lavoro svolto per la risoluzione della Prova finale di Reti Logiche dell'anno accademico 2019/2020, la quale prevede la realizzazione in VHDL di un componente hardware.
L'implementazione del codice e la stesura della documentazione sono stati svolti da Leoni Luca (codice persona 10574709) e Locarno Silvia (codice persona 10560587).
Per la realizzazione del componente sono stati utilizzati:
- XILINX VIVADO WEBPACK;
- Target FPGA xc7a200tfbg484-1.

2. Specifica

La specifica della Prova finale è ispirata al metodo di codifica a bassa dissipazione di potenza denominato "Working Zone". Dato in input un indirizzo di 7 bit e letti gli indirizzi delle working-zone[1], il componente hardware dovrà trasmettere l'indirizzo, correttamente codificato, secondo il seguente schema:
- se l'indirizzo da trasmettere (ADDR) non appartiene a nessuna working-zone, esso verrà trasmesso così come è, e un bit addizionale rispetto ai bit di indirizzamento (WZ_BIT) verrà posto a 0. L'output fornito sarà WZ_BIT & ADDR (dove & è il simbolo di concatenazione);
- 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 due sotto campi rappresentanti:
  - Il numero della working-zone alla quale appartiene WZ_NUM, codificato in binario.
  - L'offset rispetto all'indirizzo base della working-zone WZ_OFFSET, codificato come one-hot.
- L'output fornito sarà dunque WZ_BIT & WZ_NUM & WZ_OFFSET (dove & è il simbolo di concatenazione).

Essendo ADDR un indirizzo composto da 7 bit, esso sarà un numero compreso tra 0 e 127. Il numero delle working-zone è 8, mentre la dimensione di ciascuna WZ è 4 indirizzi (incluso quello base). L'indirizzo codificato sarà dunque composto da 8 bit, uno per WZ_BIT, tre per WZ_NUM e quattro per WZ_OFFSET.

I dati, ciascuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento al Byte partendo dalla posizione 0. Anche l'indirizzo da trasmettere, che da specifica è di 7 bit, viene memorizzato su 8 bit. Il valore dell'ottavo bit sarà sempre 0.
Le posizioni in memoria da 0 a 7 sono usate per memorizzare gli otto indirizzi base delle working-zone:

- 0 - Indirizzo Base WZ 0
- 1 - Indirizzo Base WZ 1
- ...
- 7 - Indirizzo Base WZ 7

La posizione in memoria 8 avrà al suo interno il valore (indirizzo) da codificare (ADDR).

La posizione in memoria 9 è quella che deve essere usata per scrivere, alla fine, il valore codificato secondo le regole precedenti.

[1] Una working-zone è definita come un intervallo di dimensione fissa che parte da un indirizzo base.

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 CON VALORE NON PRESENTE IN NESSUNA WORKING-ZONE

Indirizzo Memoria Valore Commento

0 4 // Indirizzo Base WZ 0
1 13 // Indirizzo Base WZ 1
2 22 // Indirizzo Base WZ 2
3 31 // Indirizzo Base WZ 3
4 37 // Indirizzo Base WZ 4
5 45 // Indirizzo Base WZ 5
6 77 // Indirizzo Base WZ 6
7 91 // Indirizzo Base WZ 7
8 42 // ADDR da codificare
9 42 // valore codificato

CASO 2 CON VALORE PRESENTE IN UNA WORKING-ZONE

Indirizzo Memoria Valore Commento

0 4 // Indirizzo Base WZ 0
1 13 // Indirizzo Base WZ 1
2 22 // Indirizzo Base WZ 2
3 31 // Indirizzo Base WZ 3
4 37 // Indirizzo Base WZ 4
5 45 // Indirizzo Base WZ 5
6 77 // Indirizzo Base WZ 6
7 91 // Indirizzo Base WZ 7
8 33 // ADDR da codificare
9 180 // Valore codificato con in OUTPUT (1 - 011 - 0100)

INTERFACCIA DEL COMPONENTE

Il componente da descrivere deve avere la seguente interfaccia.

```
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;
```

In particolare:

- `i_clk` è il segnale di CLOCK in ingresso generato dal TestBench;
- `i_start` è il segnale di START generato dal TestBench;
- `i_rst` è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- `i_data` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_address` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `o_done` è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- `o_en` è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- `o_we` è il 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.

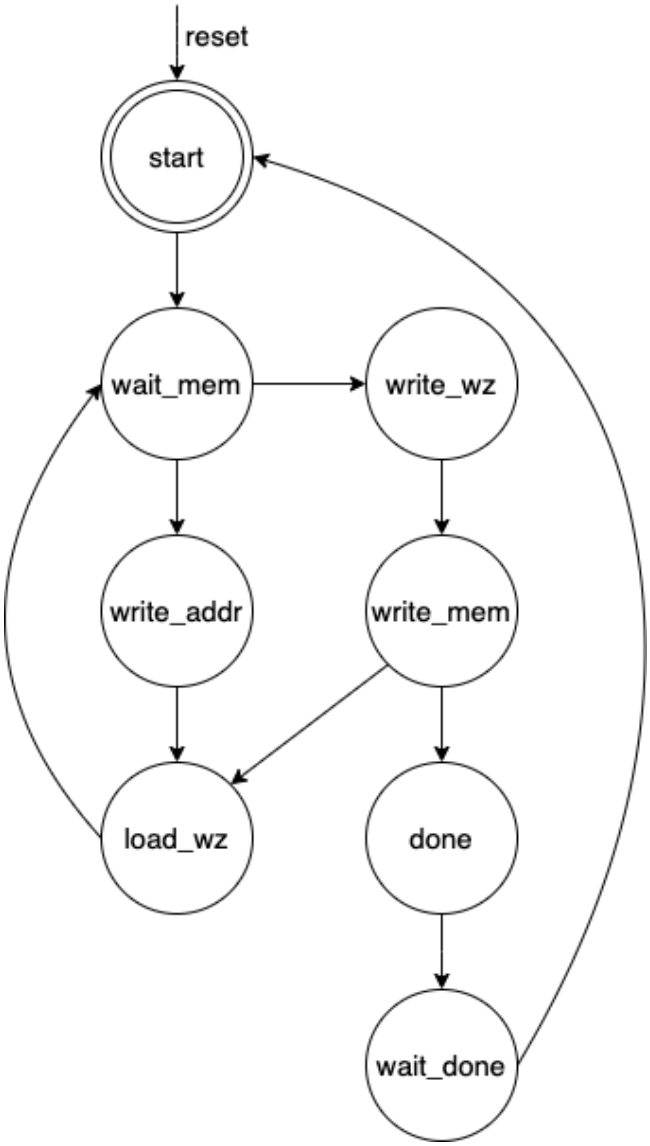
3. ARCHITETTURA

L'approccio scelto è quello di una macchina a stati finiti. La scelta è condizionata dalla presenza del segnale di RESET e dalla natura sequenziale del processo.
Gli stati utilizzati sono in totale 8.

La macchina preleva dalla cella otto della memoria l'indirizzo da codificare e da scrivere successivamente nella cella nove. L'indirizzo da codificare viene salvato all'interno di un vettore MI, confrontato con gli indirizzi delle working-zone così da poter identificare, nel caso in cui il confronto dia esito positivo, la working-zone corrispondente. Gli indirizzi delle working-zone sono salvati in un vettore MO. Individuata la working-zone corrispondente, l'indirizzo viene codificato, secondo quanto indicato nella specifica, e scritto nella cella nove della memoria.

Gli stati rappresentati nella figura a lato svolgono le seguenti funzioni:

- **start**: stato iniziale dove si attende che il segnale `i_start` venga alzato e successivamente si carica l'indirizzo da codificare; nel caso vengo alzato il segnale di reset `i_rst` si torna in questo stato, qualunque sia lo stato di provenienza. In questo stato viene abilitata la lettura.
- **wait_mem**: stato che intermedia gli stati che necessitano di ricevere dati dalla memoria.
- **write_addr**: stato in cui viene memorizzato l'indirizzo da codificare all'interno del vettore MI.
- **load_wz**: stato in cui viene caricato l'indirizzo di una working-zone.
- **write_wz**: stato in cui viene memorizzato l'indirizzo di una working-zone all'interno del vettore MO.
- **write_mem**: stato in cui si verifica se l'indirizzo appartiene a una qualche working-zone e viene scritto il valore dell'indirizzo codificato nella cella nove della memoria. In questo stato viene abilitata la scrittura.
- **done**: stato in cui viene a portato a '1' il segnale `o_done` e vengono disabilitate lettura e scrittura.
- **wait_done**: stato in cui si attende che il segnale di start `i_start` venga riportato a '0' per riportarci allo stato **start**.



4. TEST

Per verificare l'effettivo funzione del componente, esso è stato testato con alcuni test benches, due forniti nella specifica e altri quattro che coprono i casi più critici. Qui di seguito si fornisce una breve descrizione e lo screenshot dell'andamento dei segnali durante la simulazione.

TEST DI ESEMPIO

In questi due test viene verificato il funzionamento nei due casi base: l'appartenenza dell'indirizzo da codificare all'interno di una working-zone e la sua assenza.

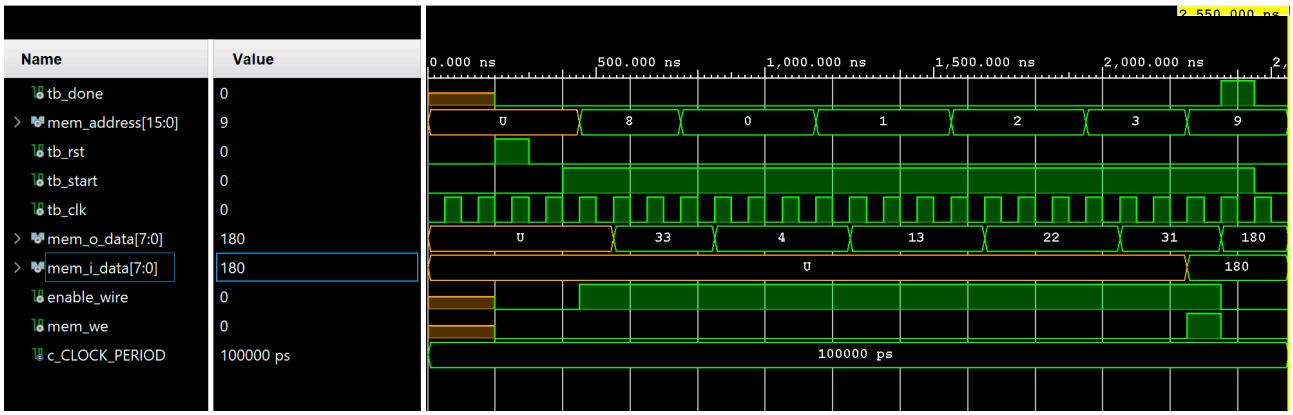


Figura 1: l'indirizzo da codificare è presente nella working-zone 3

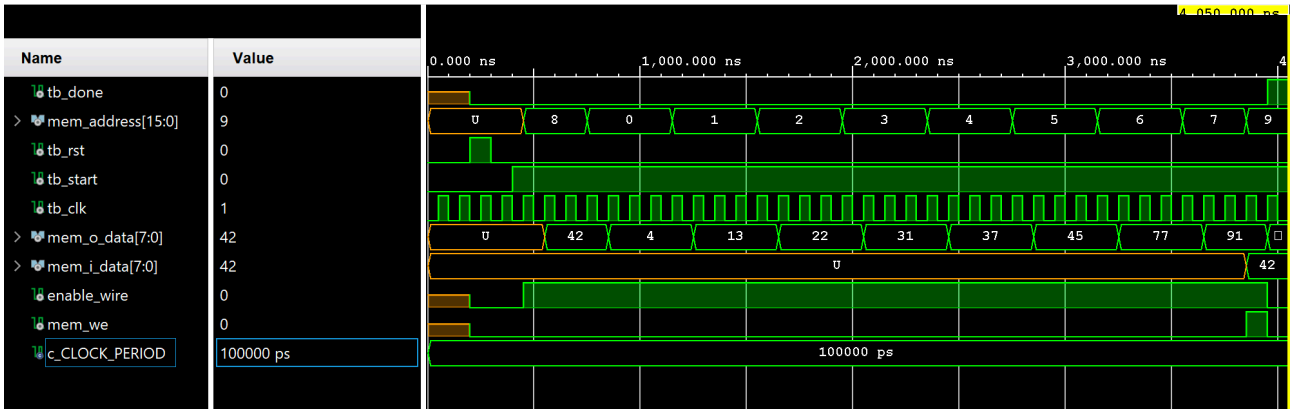


Figura 2: l'indirizzo da codificare non è presente in nessuna working-zone

WORKING-ZONE ADIACENTI

Il test verifica che la presenza di due working-zone adiacenti non comprometta la computazione e che quindi venga individuata la working-zone di appartenenza corretta.

In questo preciso caso, riportato in figura 3, l'indirizzo da codificare era 33 e le working-zone adiacenti avevamo come indirizzo base rispettivamente 31 e 35.

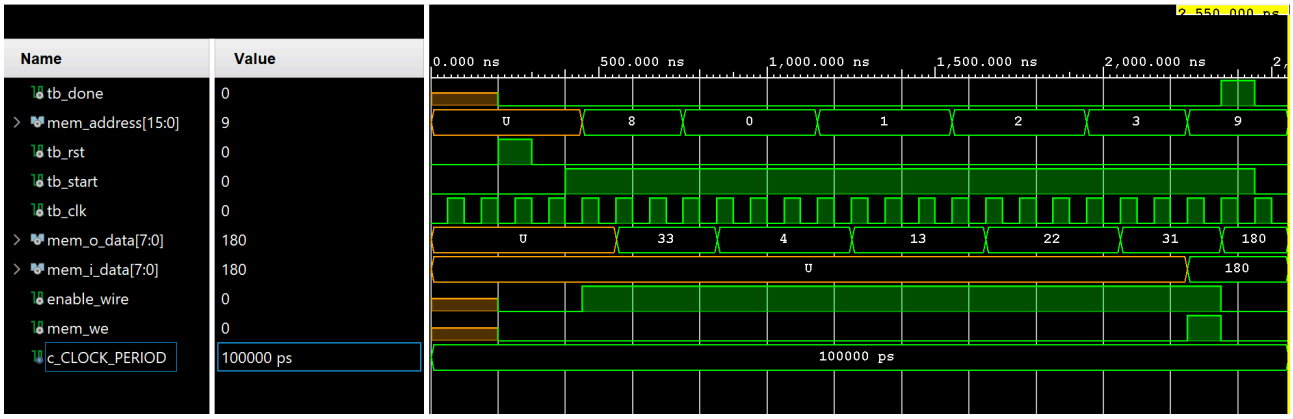


Figura 3: Working-zone adiacenti

MULTISTART

Il test verifica la corretta computazione nel caso in cui all'interno dello stesso test ci siano due segnali di start, senza cambiamenti nelle working-zone, ma con il cambiamento dell'indirizzo da codificare.

Nel caso riportato in figura 4, il primo indirizzo appartiene ad una delle working-zone mentre il secondo no.

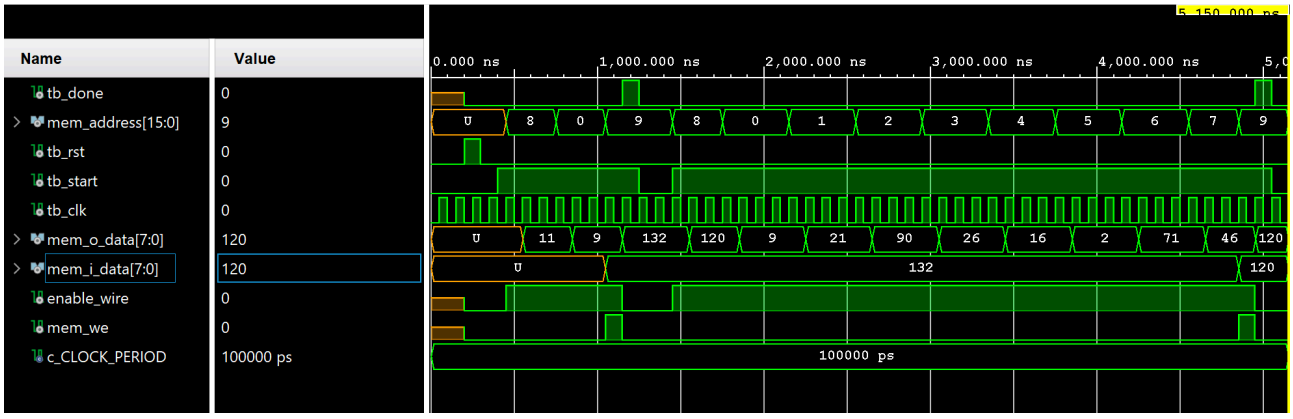


Figura 4: MultiStart

MULTIRESET

Il test verifica la corretta computazione nel caso in cui all'interno dello stesso test ci siano due segnali di reset. Questo provoca la ridefinizione degli indirizzi delle working-zone e dello stesso indirizzo da codificare.

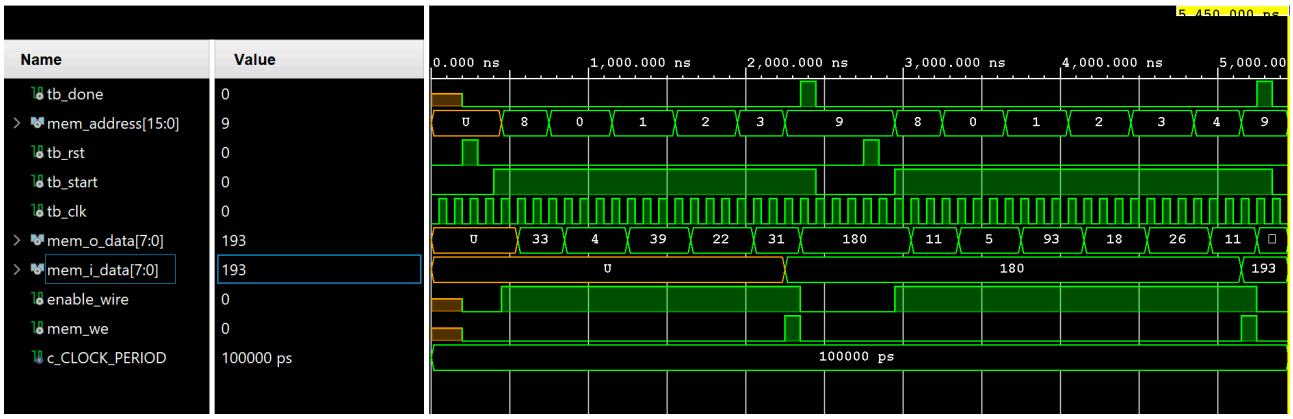


Figura 5: MultiReset

RESET ASINCRONO

Il test verifica che la presenza di un segnale di reset asincrono non comprometta la computazione e che questa ricominci facendo tornare la macchina nello stato iniziale **START**.

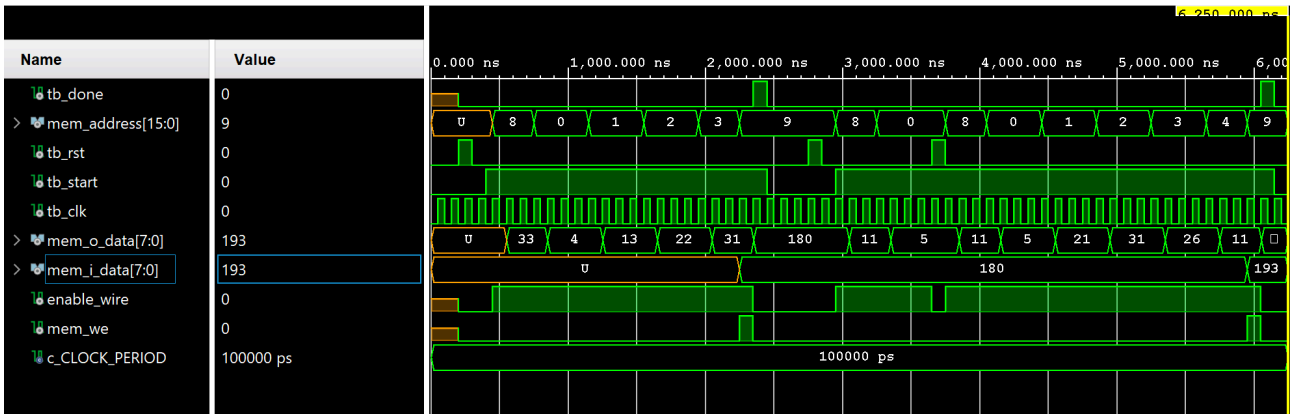


Figura 6: Reset Asincrono

5. CONSIDERAZIONI FINALI

RISULTATI SPERIMENTALI

Il componente realizzato ha dimostrato di superare correttamente, con tutti i test sopra descritti, la simulazione *Behavioral*, *Post-Synthesis Functional* e *Post-Synthesis Timing*.
La sintesi del componente ha prodotto il seguente *Schematic*:

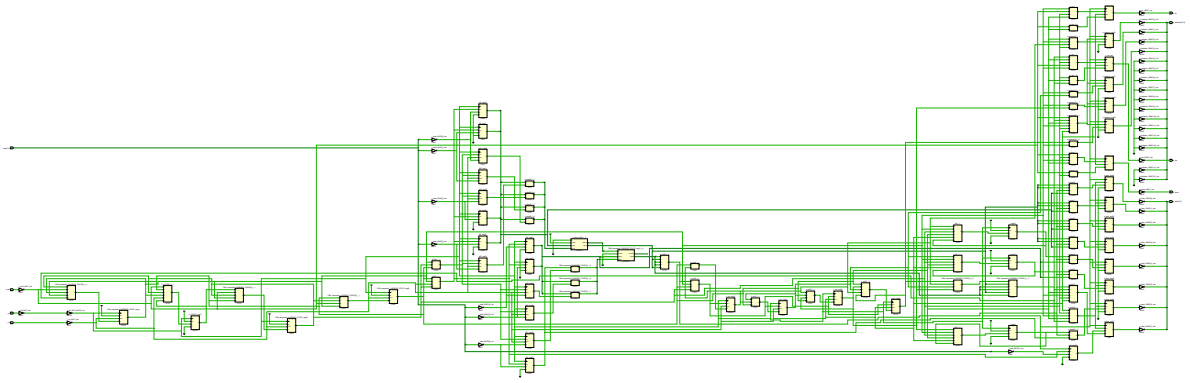


Figura 7: Schematico del componente (Elaborated Design)

OTTIMIZZAZIONI

Dopo una prima stesura è stato possibile ottimizzare il componente grazie alla riduzione degli stati. Esso inizialmente era composto da più stati, predisposti all'individuazione della corretta posizione dell'indirizzo da codificare, in ciascuno dei quali era svolta una singola operazione. É stato poi possibile raggruppare tali operazioni all'interno di un unico stato: **write_mem**.