Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo - Anno 2019/2020

Simone Sangeniti (Codice Persone 10615737 - Matricola 891082)
Massimo Valle (Codice Persona 10620441 - Matricola 912107)

Indice

1 Introduzione

1.1 Scopo del progetto

Implementare il metodo di codifica a bassa dissipazione di potenza denominata "Working Zone".

1.2 Specifiche generali

Il metodo di codifica Working Zone è un metodo pensato per il Bus Indirizzi che si usa per trasformare il valore di un indirizzo quando questo viene trasmesso, se appartiene a certi intervalli (detti appunto working-zone). Una working-zone 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);

- 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).

1.3 Interfaccia del componente

Il componente da descrivere ha 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 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 riceve il segnale di START;
- `i_data` è un segnale vettoriale che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_addr` è il segnale vettoriale 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 inviare alla memoria per poter poi mandare i segnali di lettura e scrittura;
- `o_we` è il segnale di WRITE ENABLE che messo a 1 viene mandato alla memoria nel momento in cui si ha la necessitò di doverci scrivere;
- `o_data` è il segnale vettoriale di uscita dal componente verso la memoria.

1.4 Dati e descrizione della memoria

I dati, ciascuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento a 2 byte (16 bit). Della memoria vengono utilizzate le sole prime 10 posizioni:

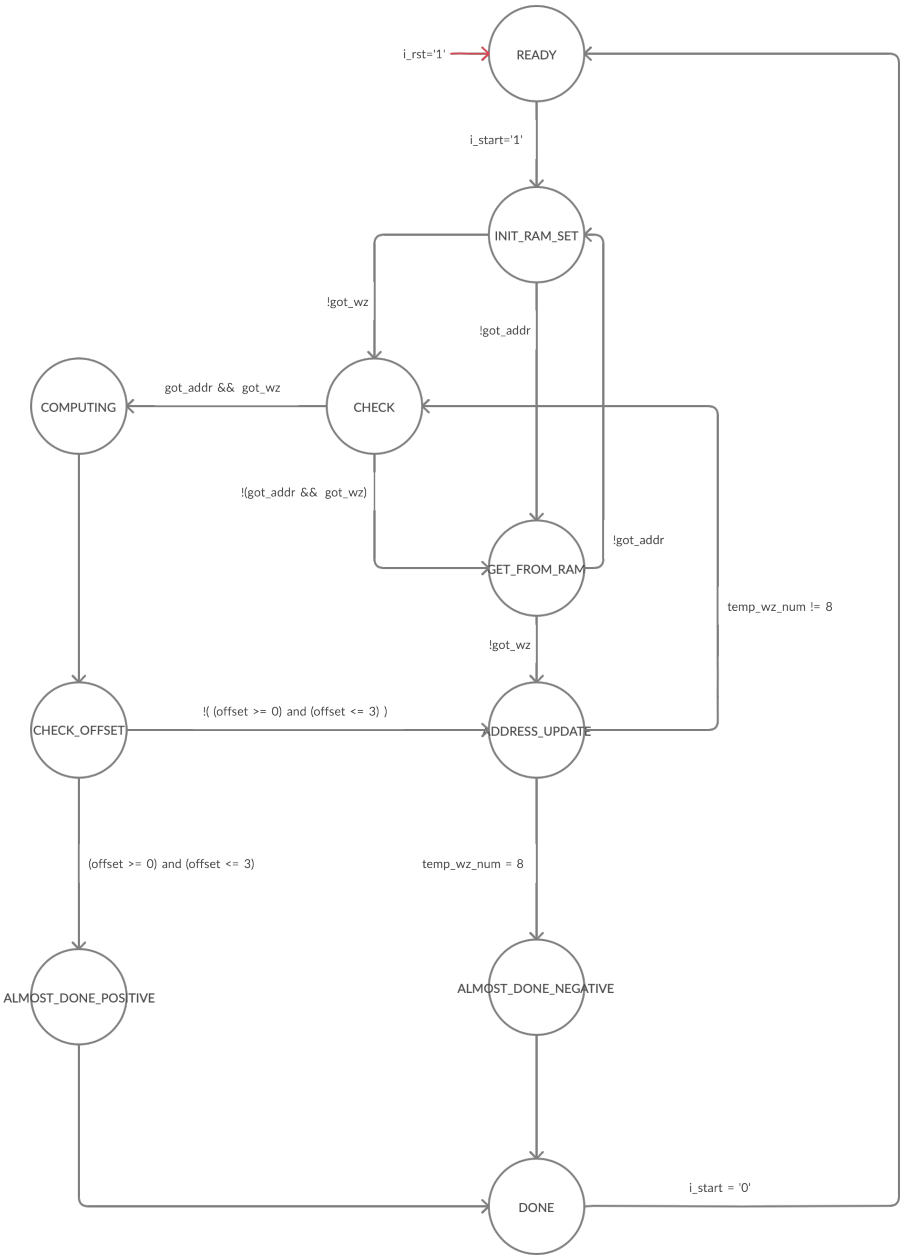
Indirizzo base della WZ0	Indirizzo 1
Indirizzo base della WZ1	indirizzo 2
Indirizzo base della WZ2	Indirizzo 3
Indirizzo base della WZ3	Indirizzo 4
Indirizzo base della WZ4	Indirizzo 5
Indirizzo base della WZ5	Indirizzo 6
Indirizzo base della WZ6	Indirizzo 7
Indirizzo base della WZ7	Indirizzo 8
ADDR da codificare	Indirizzo 9
Valore codificato in output	Indirizzo 10

- Nei primi 8 indirizzi vengono memorizzati gli indirizzi base delle working zone;
- L'indirizzo 9 è usato per memorizzato l'indirizzo da codificare;
- L'indirizzo 10 è usato per memorizzato il risultato dell'esecuzione del componente.

2 Design

2.1 Stati della macchina

La macchina è composta da 10 stati secondo il seguente schema:



In seguito vengono descritti i vari stati.

2.2.1 READY state

Stato iniziale in cui il componente attende il segnale `i_start`. In qualunque stato il componente si trovi, se viene messo a 1 il segnale `i_rst` il componente torna in questo stato. Qui vengono preparati i segnali per lo stato successivo.

2.2.2 INIT_RAM_SET state

Stato in cui viene richiesto l'indirizzo da codificare e successivamente il valore base della prima working zone.

2.2.3 CHECK state

Stato in cui viene controllato se a disposizione del nostro componente c'è sia l'indirizzo da codificare che una working zone. Altrimenti, in assenza di un indirizzo di una working zone, si va nello stato GET_FROM_RAM nel quale verrà preparato il componente per ricevere l'indirizzo della working zone (e successive).

2.2.4 GET_FROM_RAM state

Stato in cui viene effettivamente ricevuto il dato dalla ram relativo all'indirizzo precedentemente specificato. Nel primo caso sarà l'indirizzo da codificare mentre successivamente saranno gli indirizzi base delle working zone.

2.2.5 ADDRESS_UPDATE state

Stato in cui viene aggiornato il segnale di uscita che manda l'indirizzo dal quale prendere il prossimo valore della memoria. Una volta analizzate tutte ed 8 le working zone, se con esito negativo lo stato della macchina passerà in ALMOST_DONE_NEGATIVE.

2.2.6 COMPUTING state

Stato in cui viene calcolato l'offset tra la working zone e l'indirizzo da codificare.

2.2.7 CHECK_OFFSET state

Stato in cui viene verificata l'appartenenza dell'indirizzo da codificare alla working zone. Se appartiene viene messo il wz_bit a 1 e settato l'indirizzo della working zone (su 3 bit) a cui appartiene l'indirizzo da codificare. Se non appartiene verrà preparato l'indirizzo successivo dal quale leggere in memoria.

2.2.8 ALMOST_DONE_NEGATIVE state

Una volta verificata la **non** appartenenza dell'indirizzo da codificare a nessuna working zone, giunti in questo stato viene preparato il segnale di uscita del componente verso la memoria con il wz_bit a 0 e negli altri 7 bit l'indirizzo che non è stato codificato.

2.2.9 ALMOST_DONE_POSITIVE state

Una volta verificata l' appartenenza dell'indirizzo da codificare a una working zone, in questo stato viene messo nel segnale di uscita dal componente l'indirizzo codificato.

2.2.10 DONE state

La macchina persevera in questo stato fino a che il segnale di `i_start` non viene messo a 0 ed una volta avvenuto ciò, il componente ritornerà nello stato iniziale READY.

2.2 Scelte progettuali

La scelta progettuale è ricaduta sul modellare il componente con due processi:

1. Il primo processo rappresenta la parte sequenziale della macchina e serve per aggiornare i valori dei registri.

2. Il secondo processo, invece, rappresenta la FSM che sulla base dei segnali in ingesso e lo stato attuale del componente, deciderà come far evolvere il sistema.

L'algoritmo fa una sottrazione tra l'indirizzo base della working zone attualmente caricato e l'indirizzo da codificare.

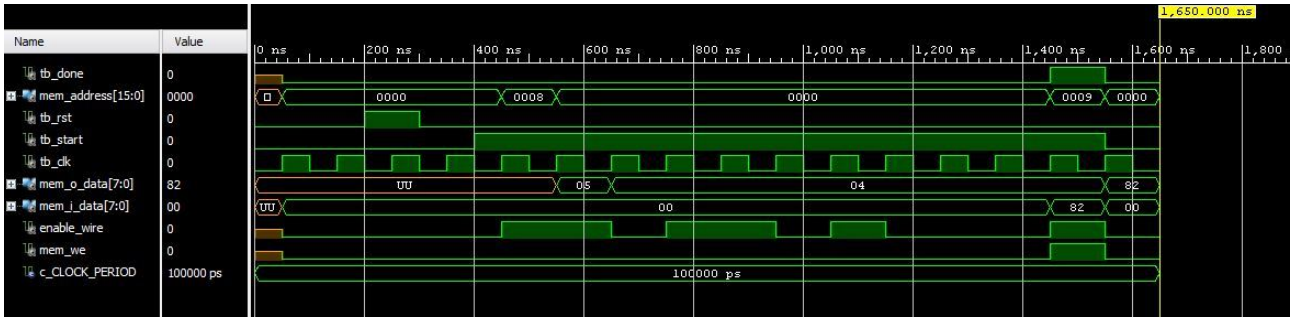
Abbiamo inoltre deciso di utilizzare un approccio che prevedesse di mantenere memorizzate meno informazioni possibili. Una volta verificata la non appartenenza dell'indirizzo da codificare alla working zone, quest'ultima viene sostituita dalla successiva in memoria per tutte le working zone. Questo approccio offre inoltre maggiore scalabilità in quanto, se bisognasse aumentare il numero di working zone da analizzare, non sarebbero necessarie modifiche al codice se non quella di cambiare le dimensioni dei vettori.

3 Risultati dei test

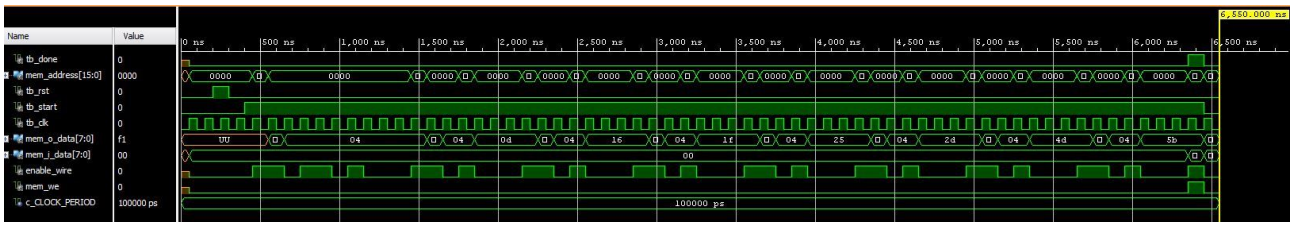
Per verificare il corretto funzionamento del componente sintetizzato l'abbiamo testato con il test bench di esempio e con altri test bench più completi in modo tale da cercare di massimizzare la copertura di tutti i possibili cammini che la macchina può effettuare durante l'esecuzione e i possibili dati in ingresso che potrebbero portarla a casi limite.

Di seguito sono riportati alcuni screenshot che mostrano l'andamento dei segnali durante la computazione dei test più significativi a cui abbiamo sottoposto la macchina:

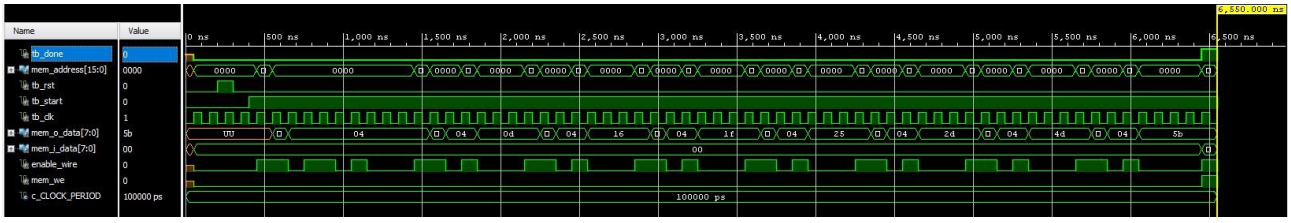
Il test bench in cui l'indirizzo appartiene alla prima working zone:



Il test bench in cui l'indirizzo appartiene all'ultima working zone:



Il test bench in cui l'indirizzo non appartiene a nessuna working zone:



4 Conclusioni

4.1 Risultati della sintesi

Il componente sintetizzato supera correttamente tutti i test specificati nelle 3 simulazioni: Behavioural, Post-Synthesis Functional e Post-Synthesis Timing.
In seguito vengono mostrati i tempi di simulazione dei casi più interessanti, ovvero se l'indirizzo appartiene alla prima working zone, all'ultima working zone e nel caso in cui non venga trovato:

- 1650ns nel caso in cui l'indirizzo è nella prima working zone, ovvero WZ0
- 6550ns nel caso in cui l'indirizzo si trova nell'ultima working zone, ovvero WZ7
- 6550ns nel caso in cui l'indirizzo non appartiene a nessuna working zone

4.2 Ottimizzazioni

Abbiamo cercato di ottimizzare la computazione riducendo il più possibile il numero di stati. Inizialmente caricavamo tutte le working zone assieme all'indirizzo per poi iniziare i confronti dalla prima all'ultima. Questo approccio però richiedeva più stati e più memoria rispetto a quello attuale.
Ora inizialmente carichiamo solo l'indirizzo e la prima working zone con cui fare la comparazione in modo tale che la computazione si può fermare subito nel caso in cui il confronto sia positivo, scrivendo in memoria l'esito.
Al contrario, nel caso in cui il confronto sia negativo, si procede caricando la working zone successiva per confrontarla con l'indirizzo e così via fino a che le working zone sono finite.