Prova Finale
Progetto Di Reti Logiche

Prof. Gianluca Palermo - Anno 2019/2020

Riccardo Pezzoni
(Codice Persona 10575577 - Matricola 888572)

1. Specifica

1.1 Descrizione

La specifica della Prova finale (Progetto di Reti Logiche) 2019 è ispirata al metodo di 1 codifica a bassa dissipazione di potenza denominato “Working Zone” .
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)
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.

1.2 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(7 downto 0);
                o_address : in std_logic(15 downto 0);
                o_done : in std_logic;
                o_en : in std_logic;
                o_we : in std_logic;
                o_data : in std_logic(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 Test Bench;
- 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.

2. Scelte progettuali

Già da una prima analisi della specifica si nota come le scelte progettuali si concentrino nel metodo utilizzato per gestire gli start multipli che non prevedono un cambio di indirizzi di working zone. Sulla base di ciò si delineano tre diversi approcci possibili:

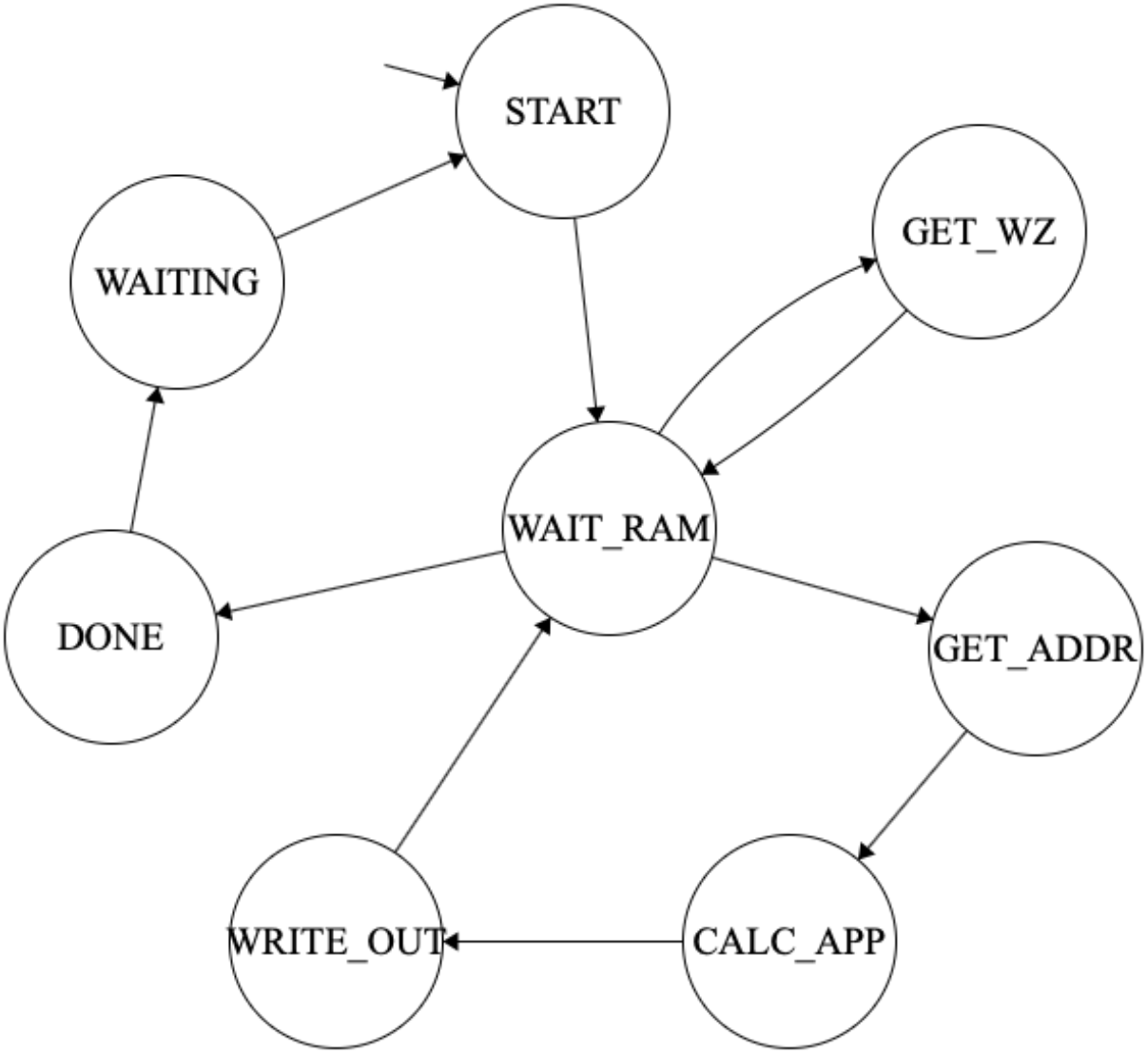
1) Ricaricare le working zone ogni volta

2) Caricare le working zone solo alla prima esecuzione o in caso di reset

3) Controllare una working zone alla volta così da non caricare quelle non necessarie

La prima strada è sicuramente la più facile da implementare ma allo stesso tempo la peggiore in termini di tempo di computazione. Esclusa questa, restano quindi le ultime due tra cui la scelta è più complicata.
Da un lato (nella seconda) non si hanno benefici nei casi in cui *ADDR* appartenga ad una delle working zone contenute nei primi indirizzi, nel secondo caso si entra più volte nello stato *CALC_APP*.
Da un'analisi di complessità asintotica (per avere un risultato più preciso servirebbero dei test) la terza opzione sembra di poco migliore della seconda in un set ristretto di computazioni.
La scelta ricade quindi sulla seconda opzione, avendo essa prestazioni simili ma semplicità di implementazione decisamente maggiore.

2.1 Schema macchina a stati finiti



2.2 Stati

START

Stato iniziale in cui si attende *i_start*. Alla prima esecuzione o dopo il reset
viene fatto il set dell'indirizzo di memoria in cui è contenuto l'indirizzo della prima working zone e si attiva la ram, nel caso si tratti di un secondo start si fa invece il set dell'indirizzo di memoria in cui è contenuto *ADDR* . Da start si va in wait_ram.

WAIT_RAM

Stato visitato sempre successivamente alla richiesta di un dato, necessario a garantire il tempo di completamento.
Lo stato successivo viene ricavato dallo stato di provenienza, che ad ogni passo viene salvato come *P_STATE*.

GET_WZ

Visitato ciclicamente per salvare il valore della woking zone richiesta precedentemente e richiedere il successivo. Salvato l'ultimo valore di working zone, infine, richiede il valore di *ADDR* da codificare.

GET_ADDR

Assegna ad *ADDR* il valore prelevato dalla memoria.

CALC_APP

Controlla una per una le working zone verificando se *ADDR* appartiene a una di esse.
Nel caso, setta a true *appartiene*;

WRITE_OUT

Scrive in memoria la codifica corretta in base all'appartenenza servendosi del numero di wz (*wz_num*) e l'offset (*wz_offset*) ricavati in precedenza.

DONE

Pone o_done a 1.

WAITING

Azzera le variabili utilizzate, fatta eccezione per quelle relative alle working zone e permette il ritorno in start per attendere una nuova invocazione.

3. Test

3.1 Test svolti

I test effettuati sono stati di diversi tipi:
1.Test sul risultato

1.1 mirati: assegnamento di valori specifici per verificare il corretto funzionamento in stati limite

1.2 randomici su la larga scala: set randomici di grandi dimensioni

2. Test sul comportamento:

2.1 controllo del corretto funzionamento in caso di reset

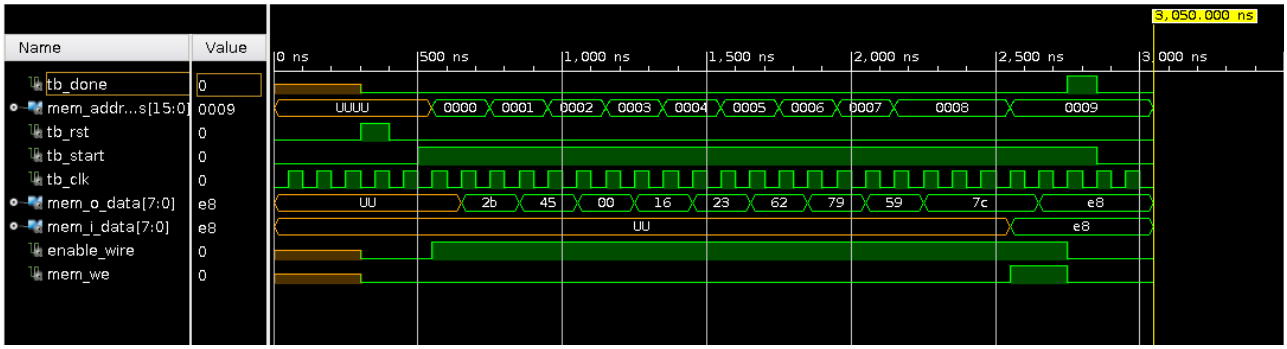
2.2 controllo del corretto funzionamento in caso di start successivi

3.2 Risultati dei test

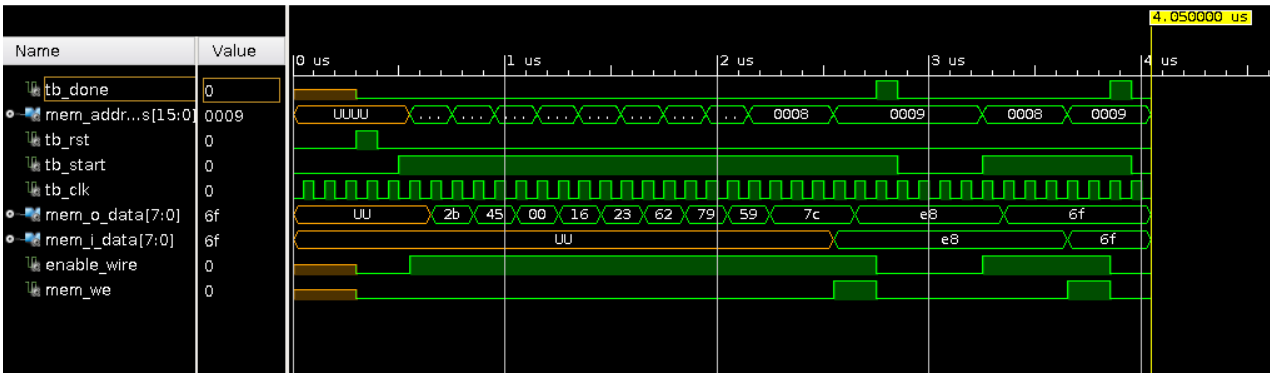
I test effettuati, tutti superati correttamente in Behavioral, Post-Synthesis Functional e Post-Synthesis Timing, hanno mostrato il corretto funzionamento del programma VHDL e hanno evidenziato le peculiarità dell'implementazione scelta.
I tempi migliori vengono registrati nei casi in cui il si hanno start consecutivi, poiché il risparmio di tempo nel non dover ricaricare ogni working zone è considerevole.
Allo stesso modo il tempo di computazione è pressoché equivalente se *ADDR* appartiene o no ad una working zone così come se appartenga alla prima o all'ultima working zone, esattamente come ci aspettiamo.

Due esempi a dimostrazione:

- Uno start, addr appartiene
  wz: [43, 69, 0, 22, 35, 98, 121, 89] addr: 124 result: 232



- Due start senza reset, addr appartiene nel primo e non nel secondo
  wz: [43, 69, 0, 22, 35, 98, 121, 89] addr: 124 result: 232
  wz: [43, 69, 0, 22, 35, 98, 121, 89] addr: 111 result: 111



4. Limiti della programma

Al fine di risparmiare computazioni poco utili, non vengono effettuati controlli su nessuna condizione non richiesta, in particolare:

1. working zone sovrapposte
2. *ADDR* fuori range
3. Nel caso di non appartenenza bisognerebbe salvare in memoria 0&ADDR, in realtà *ADDR* è massimo 127.

5. Conclusioni

Sebbene siano possibili altre migliorie per risparmiare alcuni cicli di clock oppure eliminare o raggruppare alcuni stati, dai risultati ottenuti si può considerare di aver ottenuto un buon compromesso tra prestazioni e leggibilità del codice.