Prova Finale (Progetto di Reti Logiche)

Professore: Palermo Gianluca – Anno Accademico 2019/2020

Terzi Eugenio
Zaffiretti Stefano

Indice

1 Introduzione

Il metodo di codifica a bassa dissipazione di potenza denominato “Working Zone”, è un metodo pensato per il Bus di indirizzi ed è utilizzato per codificare il valore di un indirizzo quando viene trasmesso, nel caso appartenga a certi intervalli detti appunto Working-zone.

1.1 Scopo del Progetto

Lo scopo del progetto è quello di realizzare un componente hardware, descritto in VHDL, che ricevuto in ingresso il valore di un indirizzo effettui una codifica per esplicitare la sua appartenenza o meno ad una Working zone.

1.2 Specifiche Generali

Una Working-zone è definita come un intervallo di inizi di dimensione fissa che parte da un indirizzo base. Lo schema di codifica implementato corrisponde alle seguenti specifiche:

- Se l’indirizzo da trasmettere non appartiene a nessuna Working zone, allora non verrà compiuta alcuna operazione, e il bit addizionale di indirizzamento **WZ_BIT** sarà posto a 0. La struttura dell’indirizzo scritto su memoria sarà: **WZ_BIT & ADDR**
- Se l’indirizzo da trasmettere appartiene ad una Working zone, il bit addizionale **WZ_BIT** è posto ad 1, mentre i sette bit di indirizzo vengono processati e divisi in due blocchi in questo modo: - 3 bit per il **WZ_NUM**
  - 4 bit per il **WZ_OFFSET** codificato in one-hot

  La struttura dell’indirizzo scritto su memoria sarà quindi: **WZ_BIT & WZ_NUM & WZ_OFFSET**

Il numero di Working zone da considerare è 8.
Il modulo da implementare legge gli 8 indirizzi base delle working zone da memoria, l’indirizzo da codificare e una volta effettuata la computazione produce l’indirizzo opportunamente codificato.

Base WZ 0	Indirizzo 0
Base WZ 1	Indirizzo 1
Base WZ 2	Indirizzo 2
Base WZ 3	Indirizzo 3
Base WZ 4	Indirizzo 4
Base WZ 5	Indirizzo 5
Base WZ 6	Indirizzo 6
Base WZ 7	Indirizzo 7
Indirizzo da Codificare	Indirizzo 8
Valore Codificato	Indirizzo 9

Figura 0: Legenda

4	Indirizzo 0
13	Indirizzo 1
22	Indirizzo 2
31	Indirizzo 3
37	Indirizzo 4
45	Indirizzo 5
77	Indirizzo 6
91	Indirizzo 7
33	Indirizzo 8
180 (1 - 011 - 0100)	Indirizzo 9

Figura 1: Esempio Hit

4	Indirizzo 0
13	Indirizzo 1
22	Indirizzo 2
31	Indirizzo 3
37	Indirizzo 4
45	Indirizzo 5
77	Indirizzo 6
91	Indirizzo 7
42	Indirizzo 8
42	Indirizzo 9

Figura 2: Esempio Miss

1.3 Interfaccia del componente

Il componente implementato ha la seguente interfaccia:

```
entity Components 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)
);
```

- i_clk è il segnale di CLOCK in ingresso
- i_start è il segnale di START in ingresso
- i_rst è il segnale di RESET che re-inizializza la macchina
- i_data è il segnale (vettore) che arriva alla memoria in seguito ad una richiesta di lettura
- o_address è il segnale (vettore) in uscita che contiene l'indirizzo da richiedere alla memoria
- o_done è il segnale DONE in uscita che segnala il termine della computazione
- o_en è il segnale di ENABLE in uscita mandato alla memoria per poter comunicare in lettura o scrittura
- o_we è il segnale di WRITE ENABLE in uscita mandato alla memoria per abilitare la scrittura
- o_data è il segnale (vettore) in uscita dal componente alla memoria che contiene l'indirizzo codificato

1.4 Dati e descrizione della memoria

La memoria e il suo protocollo sono istanziati nei test bench. Gli indirizzi della memoria, ciascuno di dimensione 8 bit, sono memorizzati con indirizzamento al Byte partendo dalla posizione 0. Anche l'indirizzo da controllare, di 7 bit, è memorizzato su 8 bit. Il valore dell'ottavo bit sarà sempre zero.

Gli elementi in memoria sono organizzati nel seguente modo:

- Posizioni in memoria da 0 - 7 sono usati per memorizzare gli otto indirizzi di base delle working-zone.
- La posizione di 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.

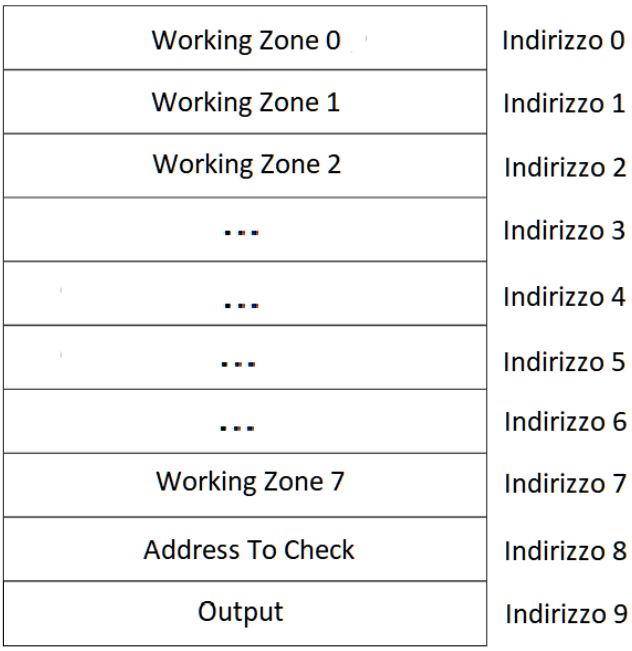

Figura 4: Struttura Memoria

2 Progettazione

2.1 Stati della Macchina

Il componente è distinto in due processi.

Il primo processo ha come parametri il segnale di CLOCK e il segnale di RESET, consentendoci di aggiornare i valori dei registri del componente.

Il secondo processo, invece, simula una FSM di 10 stati, riceve in ingresso i segnali di START e DATA, ed è strutturato in due cicli principali. Nel primo ciclo, quello di *loading*, richiede da memoria i dati delle working zone dagli indirizzi 0-7 e le carica nei propri registri locali, mentre il secondo ciclo, quello di *check,* richiede da memoria (indirizzo 8) l'indirizzo da controllare, verifica la sua appartenenza ad una working zone e scrive il risultato in memoria (indirizzo 9). A seguito di un nuovo segnale di START, se i registri locali sono vuoti, entra in fase di *loading*, altrimenti prosegue direttamente in fase di *check.*

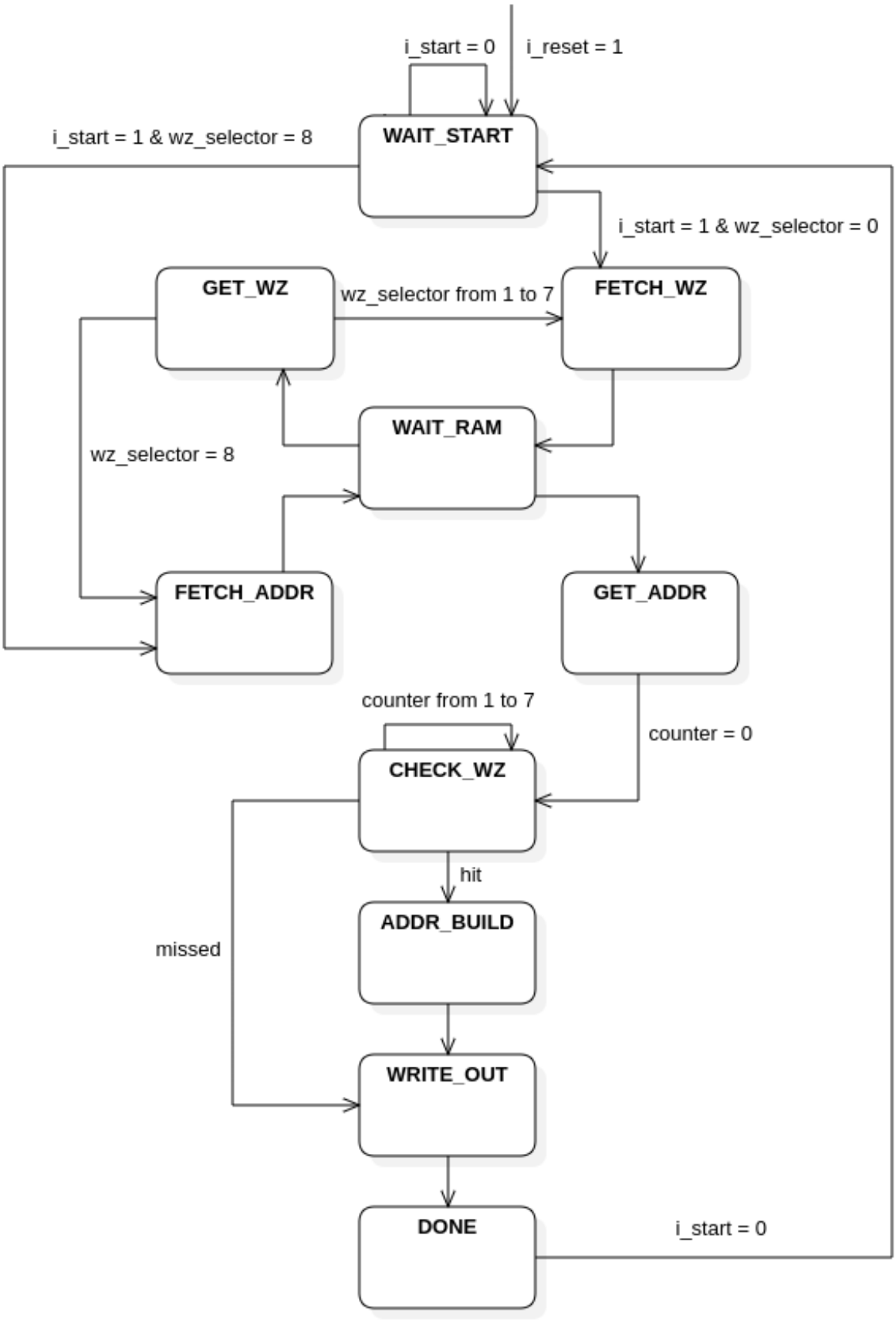

Figura 5 FlowChart FSM

2.1.1 WAIT_START

Stato iniziale del componente. Attende il segnale i_start per iniziare la computazione.
Si giunge in questo stato a termine di una computazione o a seguito dell'innalzamento del segnale i_rst.

2.1.2 FETCH_WZ

Stato in cui il componente richiede il contenuto (uno alla volta) dei primi otto indirizzi della memoria, in cui sono contenuti i valori delle working zone. Si entra in questo stato solo nel caso in cui i registri wz del componente non sono inizializzati.

2.1.3 WAIT_RAM

Stato di attesa del dato richiesto dalla memoria.

2.1.4 GET_WZ

Stato in cui il componente riceve il contenuto (uno alla volta) dei primi otto indirizzi della memoria e lo memorizza nei registri locali wz.

2.1.5 FETCH_ADDR

Stato in cui il componente richiede il contenuto del nono indirizzo di memoria, in cui vi è il valore dell'indirizzo da controllare.

2.1.6 GET_ADDR

Stato in cui il componente riceve il contenuto del nono indirizzo di memoria e lo memorizza del registro locale Address_ToCheck.

2.1.7 CHECK_WZ

Stato in cui il componente controlla l'appartenenza alla working zone dell'indirizzo.
Se l'indirizzo appartiene ad una working zone, si entra nello stato di ADDR_BUILD altrimenti non viene compiuta alcuna azione e si passa nello stato di WRITE_OUT.

2.1.8 ADDR_BUILD

Stato in cui il componente calcola il nuovo indirizzo, ponendo il WZ_BIT a 1, inserendo il WZ_NUM della wz hittata, e il WZ_OFFSET.

2.1.9 WRITE_OUT

Stato in cui viene scritto il dato in uscita sul decimo indirizzo di memoria.

2.1.10 DONE

Stato in cui si attende che la memoria abbassi i_start per poter abbassare o_done e tornare allo stato di WAIT_START.

2.2 Scelte progettuali

La scelta progettuale per la realizzazione del componente è stata di differenziarlo in due processi. Riteniamo questo modo ottimale, poiché ci consente con un processo di occuparsi del trasferimento e aggiornamento dei registri mentre il secondo, rappresentando la FSM, si occupa delle fasi di loading e controllo di appartenenza a Working zone analizzando i segnali in ingresso.

Per quanto riguarda l'algoritmo per determinare l'appartenenza alla Working Zone di un indirizzo, abbiamo deciso un approccio minimale, con un semplice confronto iterativo tra l'indirizzo da controllare e le varie working zone. Questo approccio risulta meno veloce a computare il risultato rispetto ad altre alternative, come ad esempio il parallelismo, ma consente una struttura hardware più minimale e di conseguenza low-power, che ci è sembrata più in linea con la filosofia "working zone".

3 Esito dei Test

Come primo test, si è verificato un corretto funzionamento con la TestBench fornita da specifiche.
In seguito, abbiamo creato una serie di test randomici con un breve script in Python. Lo script è stato realizzato in modo tale che i test realizzati rispecchiassero la filosofia Working Zone. Il singolo file di test conteneva un numero molto significativo di operazioni. Eseguendo queste simulazioni abbiamo appurato la robustezza e la correttezza del nostro componente.

La scelta di algoritmo di ricerca elementare, come la ricerca iterativa sui registri locali delle working zone, ci consente inoltre di superare le varie casistiche di test “critiche”. Nel caso di molteplici Working Zone uguali, l’hit avverrà alla prima Working Zone di quelle uguali, come nel caso ci possano essere delle Working Zone sovrapposte.

Per quanto riguarda il corretto funzionamento dei segnali, il numero di operazioni effettuate all’interno di un singolo test ci ha consentito di verificare il corretto funzionamento e timing della lettura e scrittura da memoria, nonché il passaggio dalla fase di DONE a WAIT_START.

4 Conclusione

Il componente sintetizzato supera correttamente tutti i testi specificati nelle simulazioni Behavioral e Post-Synthesis Functional. Nel caso migliore, ossia hit in prima Working Zone il componente impiega 3,650.000ns, nel caso pessimo, ossia l’hit in ottava Working Zone il componente impiega 4,350.000ns.

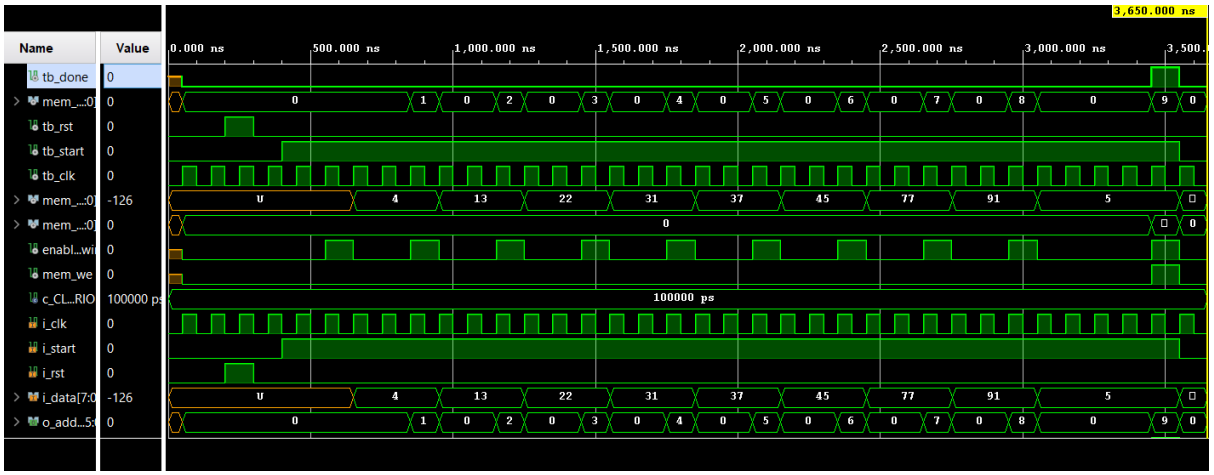

Figura 6: Caso Ottimo

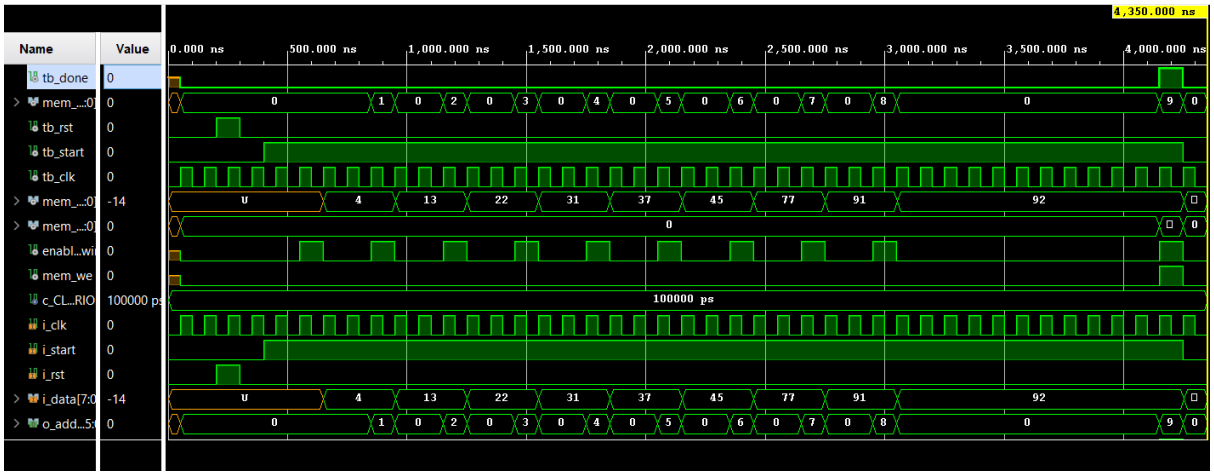

Figura 7: Caso Pessimo