Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo – Anno 2019/20

Juri Sacchetta Cod. Matricola 890600

Indice

1. Introduzione

1.1 Scopo del progetto

Il progetto si pone l'obiettivo di pianificare un componente hardware, descritto in VHDL, che, avendo preso in ingresso un indirizzo di memoria, calcola una conversione ispirata al metodo di bassa dissipazione di potenza denominato "Working Zone".

1.2 Specifiche generali

Il metodo di codifica "Working Zone" è pensato per il Bus Indirizzi, e ha lo scopo di trasformare il valore di un indirizzo, qualora esso appartenga a determinati intervalli (denominati Working-Zone), quando viene trasmesso. Una Working Zone (da adesso in poi indicata con WZ) è un intervallo di indirizzi di dimensione fissa (DIM_WZ).
All'interno della memoria possono esistere multiple WZ, ognuna identificata univocamente dall'indirizzo di memoria dove è contenuto l'indirizzo della sua base (NUM_WZ). Definendo la dimensione della WZ e l'indirizzo di base, è possibile individuare tutti gli indirizzi che vi appartengono.
Da specifica del progetto, in questo caso, sono presenti in memoria otto WZ e hanno una dimensione di quattro indirizzi.
Se il dato appartiene a una data WZ, l'output del componente (il dato codificato) deve essere: 1 & NUM_WZ & offset: l'offset, espresso su quattro bit, è calcolato come sottrazione tra l'indirizzo da codificare e la base della WZ a cui appartiene, e viene espresso in codifica OneHot.
Nel caso in cui il dato non appartenga a nessuna WZ, il componente lo trasmette come l'ha ricevuto in ingresso (non codificato).

Esempio:

33
44
10
63
98
68
38
100

I numeri sono espressi in decimale per facilitarne la lettura

L'indirizzo 100 appartiene alla quinta WZ (indirizzo 4).
La codifica corretta dell'indirizzo, in questo caso, è 1-100-0100.

- Il primo bit è a uno ed indica che l'indirizzo è stato codificato.
- 100 è la codifica binaria del numero decimale 4.
- L'offset è 3 in codifica OneHot 0100.

1.3 Interfaccia del componente

```
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 della macchina;
- i_start è il segnale di inizio della computazione;
- i_rst è il segnale di reset che inizializza la macchina portandola nello stato iniziale;
- i_data è un segnale (vector) generato dalla memoria;
- o_address è un segnale (vector) che indica alla memoria l'indirizzo al quale il componente è interessato;
- o_done è il segnale in uscita che comunica la fine dell'elaborazione;
- o_en è il segnale di ENABLE che attiva la memoria, va usato in coppia con o_we;
- o_we è il segnale di ENABLE WRITE che abilità la memoria in modalità scrittura;
- o_data è un segnale (vector) in uscita utilizzato per comunicare il dato convertito;

1.4 Dati e descrizione della memoria

La memoria è formata da 2^{16} indirizzi contenenti dati a 8 bit; l'indirizzamento è al byte e si considerano validi solo gli indirizzi esprimibili su 7 bit (da 0 a 127).
Le informazioni utili al fine della progettazione sono che:

- Gli indirizzi delle basi delle WZ sono memorizzati negli indirizzi 0-7 della memoria;
- L'indirizzo 8 contiene il dato da convertire;
- L'indirizzo 9 è utilizzato per scrivere il dato convertito in uscita;

Indirizzo base WZ	Indirizzo 0
…….	
Indirizzo base WZ	Indirizzo 7
Dato da convertire	Indirizzo 8
Dato convertito in uscita	Indirizzo 9

2. Architettura

Per risolvere il problema si è scelto di implementare una macchina a stati finiti (FSM).

2.1 Stati della macchina

Tale macchina è composta da 6 stati brevemente descritti qui di seguito:

1) **IDLE state**
   Si aspetta che il segnale i_start salga ad uno; qualora si riceva i_rst alto si torna in questo stato, portando le variabili e i segnali interni del componente ai valori di default.
   (Tale stato è quello iniziale)

2) **FETCH_DATA**
   Viene inviata una richiesta alla memoria al fine di conoscere l'indirizzo da convertire.

3) **WAIT_DATA**
   Si attende la risposta della memoria, quindi si salva l'indirizzo nel registro data_to_convert_reg.

4) **FETCH_BASE_WZ**
   Si richiede alla memoria la base di una WZ. Finché non viene trovata la WZ alla quale appartiene l'indirizzo o finché non si esauriscono le WZ da controllare, si passa molteplici volte in questo stato.

5) **ANALYZE_WRITE**
   Nel caso in cui il dato risulti effettivamente appartenente alla WZ, richiesta nello stato precedente, si definisce la codifica e si scrive in output l'indirizzo codificato. In caso contrario si controlla se sono ancora disponibili WZ da testare, dunque si torna nello stato FETCH_BASE_WZ. Se invece non vi sono più WZ da controllare si scrive l'indirizzo in chiaro (senza codifica).
   (Questo stato ha il compito di verificare l'appartenenza del dato a una specifica WZ e scrivere in memoria, pertanto è il core del componente)

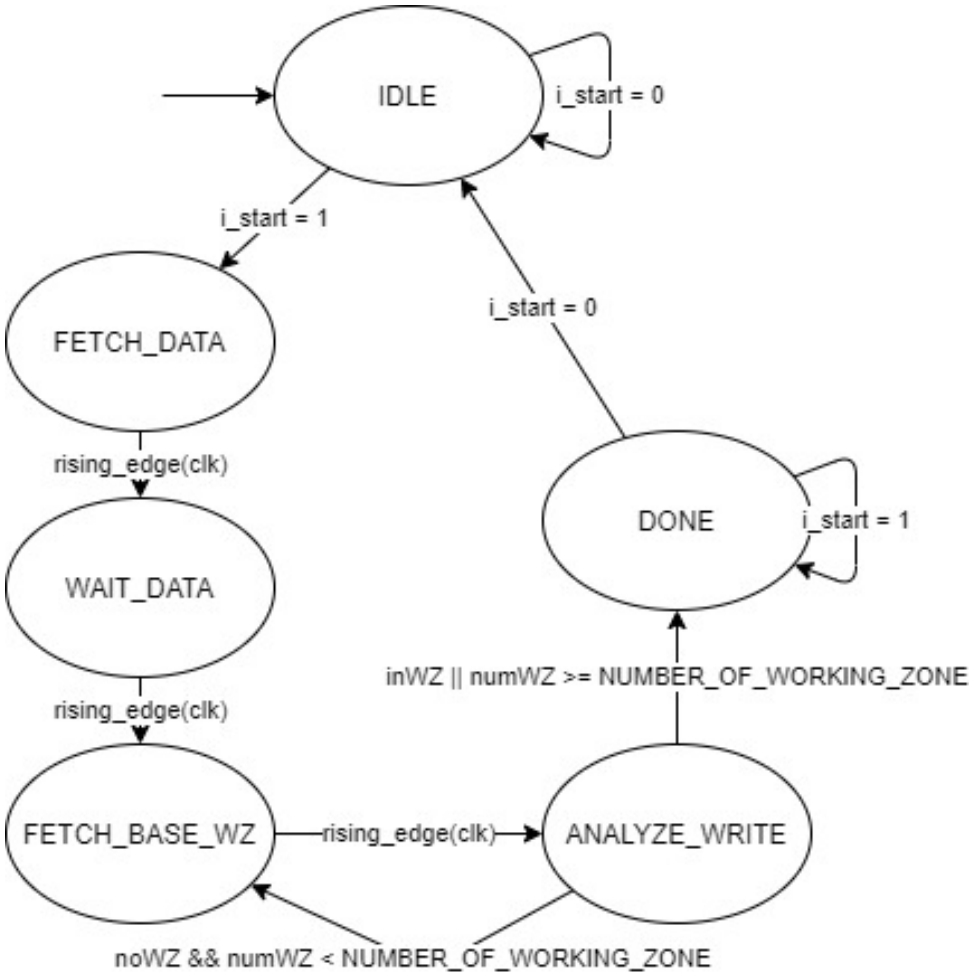
6) **DONE**
   Si resta in questo stato finché i_start rimane alto o viene ricevuto i_rst alto.
   (Tale stato indica la fine dell'elaborazione)

2.2 Scelte architetturali

Le principali scelte progettuali effettuate consistono nella definizione, tramite delle costanti, del numero di Working-Zone e della loro dimensione. In tal modo è stato possibile progettare il componente in maniera parametrica rispetto a questi due dati.
Inoltre, sono stati definiti due processi:

1) **State_reg**
   Gestisce il RT (Register transfert), reagisce al fronte di salita del clock e in maniera asincrona al reset.

2) **Delta**
   Implementa la logica interna di ogni stato: vengono decisi gli output del componente, il valore prossimo delle variabili interne e lo stato prossimo.



3. Risultati sperimentali

3.1 Risultati della simulazione

Il componente sintetizzato supera correttamente tutti i test a cui è stato sottoposto nelle simulazioni: *Behavioral* e *Post-Synthesis Functional.*
Qui di seguito è possibile vedere un confronto tra i tempi di simulazione dei due corner case, quello con il tempo di simulazione minore e quello con il tempo di simulazione maggiore:

- **1050ns** – tempo di simulazione (Behavioral) con dato appartenente alla prima WZ
- **2350ns** – tempo di simulazione (Behavioral) con dato non appartenente a nessuna WZ

3.2 Risultati di sintesi

All'interno del report di sintesi generato da Vivado si può constatare che sono stati utilizzati 18 Flip Flop e 0 Latch. Questo è il valore corretto, in quanto i dati memorizzati nei registri sono:

- **Lo stato -** 6 bit (perché Vivado li codifica in OneHot allo scopo di velocizzare le transizioni).
- **Il dato da convertire (ADDR)** - 8 bit come da specifica.
- **Il numero di WZ controllate -** 4 bit.

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |     2|
|3     |LUT2   |    15|
|4     |LUT3   |     5|
|5     |LUT4   |     7|
|6     |LUT5   |     4|
|7     |LUT6   |     1|
|8     |FDCE   |     9|
|9     |FDPE   |     1|
|10    |FDRE   |     8|
|11    |IBUF   |    11|
|12    |OBUF   |    27|
+------+-------+------+
```

```
Report Instance Areas:
+------+---------+-------+------
|      |Instance |Module |Cells
+------+---------+-------+------
|1     |top      |       |    91
+------+---------+-------+------
```

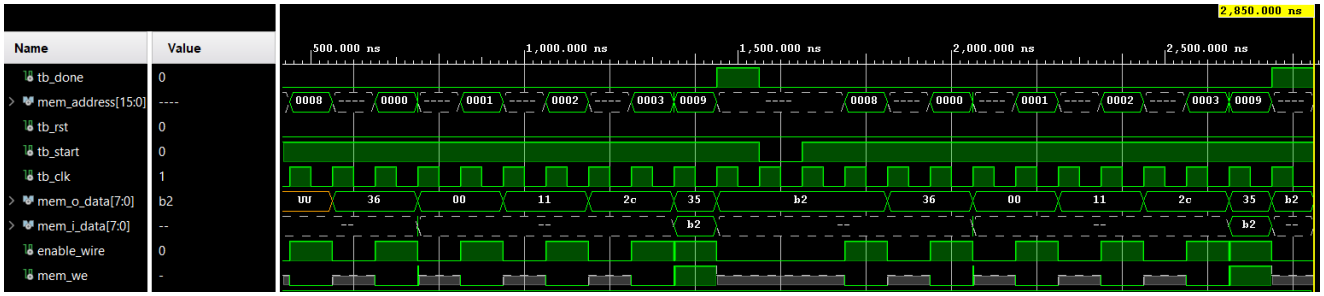
3.3 Warning

A seguito della sintesi, Vivado avvisa di 13 warning che non vengono considerati poiché 12 di questi riportano che il segnale o_address ha i primi 12 bit a 0. Tale comportamento è voluto, poiché si accede a solo i primi dieci indirizzi di memoria.
Il restante warning informa che non sono stati definiti dei *constraints*. In questo progetto tale warning non viene considerato significativo.

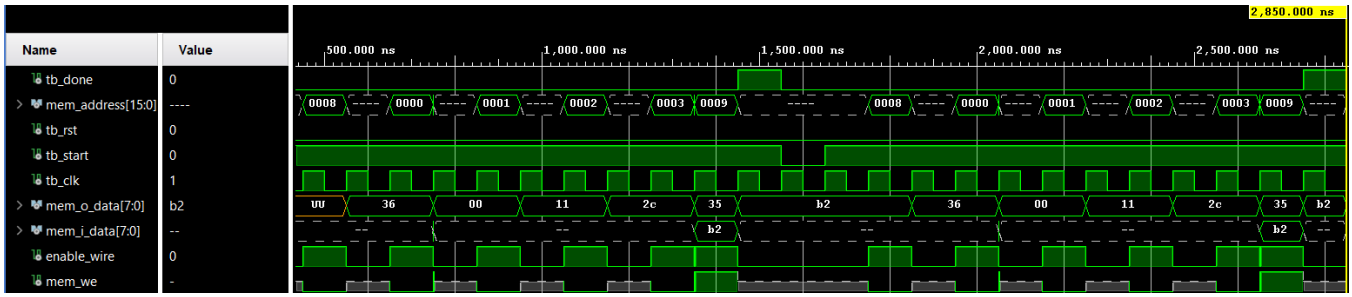
4. Risultati dei test

Dopo aver testato il componente progettato con i due test bench forniti, al fine di verificare il corretto funzionamento sono stati definiti ulteriori test, allo scopo di massimizzare la copertura di tutti i possibili cammini che la macchina può effettuare durante la computazione. Di seguito è fornita una breve descrizione di alcuni dei test utilizzati; dei due ritenuti più importanti viene anche mostrato l'effettivo funzionamento grazie allo *screenshot* della simulazione.

1. **Multi start:** la macchina dopo aver eseguito una computazione completa viene nuovamente stimolata senza che sia stato alzato il segnale i_rst.



2. **Reset asincrono:** Durante una computazione viene alzato il segnale i_rst in maniera asincrona e in seguito viene testato il funzionamento in una nuova computazione.



3. **Caso limite:** Il dato da convertire appartiene all'ultima WZ possibile: ADDR=127, WZ=124.

4. **Caso limite:** Il dato da convertire appartiene alla prima WZ possibile: ADDR = 0, WZ = 0. Dalla specifica del problema è noto che nei primi sette indirizzi di memoria sono contenuti gli indirizzi delle basi delle WZ, perciò in un reale caso d'interesse è improbabile che venga definita una WZ con indirizzo di base uguale a 0. Questo è però un caso d'interesse per il testing del componente quindi viene considerato nella progettazione.

5. Conclusioni

5.1 Ottimizzazioni

Le ottimizzazioni sono state focalizzate principalmente sulla riduzione del numero di stati. Inizialmente si progettata una macchina che, richiesto l'indirizzo della base di una WZ, passasse prima in uno stato WAIT_BASE_WZ e successivamente in uno stato ANALIZE, il quale aveva il compito di decidere se il dato appartenesse alla WZ ricevuta in precedenza.
A quest'ultimo stato potevano succedere due stati differenti (IN_WZ o NOT_IN_WZ) volti ad effettuare la corretta codifica, per poi passare nello stato WRITE_MEM, il quale aveva il compito di scrivere l'indirizzo in memoria.
In seguito, si è optato per lo schema finale, così da rendere il componente più efficiente sia a livello temporale che architetturale.
Sarebbe stato possibile diminuire ulteriormente il numero degli stati, utilizzandone soltanto uno per richiedere i dati alla memoria. Così facendo si sarebbe ridotto il numero degli stati, e ciò avrebbe portato a un minor numero di Flip Flop per il current_state_reg. Da ciò non sarebbero risultate migliorie a livello temporale mentre invece avrebbe portato a una architettura meno chiara, e a dei ripetuti controlli superflui all'interno di tale stato per decidere quale segnale o registro aggiornare.
Infine, la mancata rimozione dello stato WAIT_DATA è stata una scelta voluta dato il suo minimo impatto temporale (un ciclo di clock) sulle prestazioni. L'alternativa sarebbe stata quella di ricevere il dato da convertire nello stato FETCH_BASE_WZ e tramite un boolean decidere se aggiornare il dato o meno: anche in questo caso ciò avrebbe portato a uno schema meno chiaro della FSM e ad una ripetizione superflua del controllo.