Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo – Anno 2019/20

Juri Sacchetta Cod. Matricola 890600

Indice

1. Introduzione
   1. Obiettivo del progetto……………………………………………………………………
   2. Specifiche…………………………………………………………………………………
   3. Interfaccia del componente……………………………………………………………...
   4. Dati e descrizione della memoria……………………………………………………….
2. Architettura
   1. Stati della macchina……………………………………………………………………...
      1. **IDLE** state………………………………………………………………………….
      2. **FETCH\_DATA** state…………………………………………..
      3. **WAIT\_DATA** state…………………………………………….
      4. **FETCH\_BASE\_WZ** state………………………………………………………...
      5. **WAIT\_BASE\_WZ** state…………………………………………………………..
      6. **ANALYZE\_WRITE** state…………………………………………………………..
      7. **DONE** state………………………………………………………………………..
   2. Scelte progettuali…………………………………………………………………………
3. Risultati sperimentali
   1. Risultati di sintesi
4. Risultati dei test bench
   1. Secondo start
   2. Reset asincrono
5. Conclusioni

**1 Introduzione**

1.1 Scopo del progetto

Il progetto si pone l’obiettivo di implementare un componente hardware, descritto in VHDL, che 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 un metodo pensato per il Bus Indirizzi ha lo scopo di trasformare il valore di un indirizzo quando viene trasmesso, se appartiene a certi intervalli (denominati Working Zone). Una Working Zone è definita come un intervallo di indirizzi di dimensione fissa (DIM\_WZ) e un indirizzo di base. All’interno dello schema di codifica possono esistere multiple Working Zone.

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 in ingresso;
* i\_start è il segnale di start che segna l’inizio della computazione;
* i\_rst è il segnale di reset che inizializza la macchina portandola nello stato iniziale;
* i\_data il segnale (vector) generato dalla memoria in seguito a una richiesta;
* o\_address è il segnale (vector) che indica alla memoria l’indirizzo al quale il componente è interessato;
* o\_done è il segnale in uscita che 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 il segnale (vector) in uscita utilizzato per comunicare i dato convertito;

1.4 Dati e descrizione della memoria

I dati utili al fine dell’elaborazione, ciascuno di 8 bit, sono memorizzati in memoria con indirizzamento al byte:

* Gli indirizzi dallo 0 al 7 sono usati per memorizzare le basi delle Working Zone;
* L’indirizzo 8 contiene il dato da convertire;
* L’indirizzo 9 è utilizzato per scrivere il dato convertito in uscita;

**2 Architettura**

2.1 Stati della macchina

La macchina è composta da sette stati. Di seguito viene fornita una breve descrizione per ognuno di essi:

* IDLE state
  + Stato iniziale nel quale si aspetta che il segnale i\_start salga a uno. Nel caso si riceva i\_rst alto si torna in questo stato portando le variabili e i segnali interni del componente ai valori di default.
* FETCH\_DATA
  + Stato nel quale viene inviata una richiesta alla memoria per conoscere il dato da convertire.
* WAIT\_DATA
  + Stato nel quale si attende la risposta della memoria e si salva il dato nel registro data\_to\_convert\_reg.
* FETCH\_BASE\_WZ
  + Stato nel quale si richiede alla memoria la base di una WZ. Si può passare molteplici volte in questo stato, ovvero finché non viene trovata la WZ corrispondente al dato o si esauriscono le WZ da controllare.
* WAIT\_BASE\_WZ
  + Stato nel quale si attende la risposta della memoria ovviamente si accede tante volte a questo stato tante quanti sono gli accessi allo stato FETCH\_BASE\_WZ.
* ANALYZE\_WRITE
  + Questo è il core del componente, questo stato ha il compito di verificare l’appartenenza del dato a una specifica WZ. Nel caso il dato risulti effettivamente appartenente a quella data WZ allora si definisce la codifica e viene mandata in scrittura alla memoria. In caso contrario si controlla se sono ancora disponibili WZ da controllare, in tal caso si torna nello stato FETCH\_BASE\_WZ. Infine, se non vi sono più WZ da controllare allora si trasmette il dato senza codifica.
* DONE
  + Stato che indica la fine dell’elaborazione e si resta in questo stato finché i\_start resta alto o viene ricevuto i\_rst alto.

2.2 Scelte architetturali

Le principali scelte progettuali effettuate sono state la definizione del numero di Working-Zone e la loro grandezze tramite delle constanti cosi da poter progettare il componente in maniera parametrica rispetto a questi due dati, e la definizione del componente tramite due processi:

1. Il primo rappresenta la parte sequenziale della macchina, gestisce il RT (Register transfert), reagisce al clock e al reset.
2. Il secondo implementa la logica interna di ogni stato, in questo vengono decisi gli output del componente, il valore prossimo delle variabili interne e lo stato prossimo

Immagine che contiene testo, mappa

Descrizione generata automaticamente

**3 Risultati sperimentali**

3.1 Warning

A seguito della sintesi Vivado avvisa di 13 warning che non vengono considerati in quanto 12 di questi riportano che il segnale o\_address ha i primi 12 bit a 0, questo è un comportamento voluto in quanto si accede a solo i primi 10 indirizzi di memoria. Il restante informa che non sono stati definiti dei *constraints* anche questo non viene considerato significativo.

3.2 Risultati di sintesi

Il componente sintetizzato supera correttamente tutti i test specificati nelle simulazioni:

*Behavioral* e *Post-Synthesis Functional.*

Qui di seguito è possibile vedere un confronto tra i tempi di simulazione dei due corner

case che portano la macchina verso la più breve e la più lunga simulazione:

* **1150ns** – tempo di simulazione (Behavioral) con dato appartenente alla prima Worzing-Zone
* **3150ns** – tempo di simulzione (Behavioral) con dato non appartenente a nessuna Worzing-Zone

**4 Risultati dei test**

Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con i due test bench forniti, sono stati definiti ulteriori test allo scopo mi massimizzare la copertura di tutti i possibili cammini che la macchina può effettuare durante la computazione.

Di seguito è fornita una breve descrizione dei test utilizzati e di alcuni viene anche mostrato l’effettivo funzionamento grazie allo *screenshot* della simulazione.

1. **Secondo start:** la macchina dopo aver eseguito già una computazione completa viene nuovamente stimolata senza che sia stato alzato il segnale di RESET.

Immagine che contiene monitor, schermo, nero, televisione

Descrizione generata automaticamente

1. **Reset asincrono:** Durante una computazione viene alzato il segnale di RESET in maniera asincrona e inseguito viene testato il funzionamento in una nuova computazione.

Immagine che contiene monitor, schermo, orologio, nero

Descrizione generata automaticamente

1. **Caso limite:** Il dato da convertire appartiene all’ultima WZ possibile: ADDR=127, WZ=124.
2. **Caso limite:** Il dato da convertire appartiene alla prima WZ: ADDR = 0, WZ = 0.

**5 Conclusioni**

5.1 Ottimizzazioni

Le ottimizzazioni che sono state effettuate sono principalmente nella riduzione del numero di stati. In prima battuta si era progettata una macchina che una volta analizzato il dato passasse in uno stato IN\_WZ O NOT\_IN\_WZ per effettuare la corretta codifica e uno stato WRITE\_MEM per scrivere il dato in memoria.

In seguito si è optato per lo schema finale così da rendere il componente più efficiente sia a livello temporale che architetturale.