Progetto di Reti Logiche

Prof. Gianluca Palermo - Anno 2019/2020

Rigutti Luca [codice persona: 10558383]
Tortorelli Giuseppe [codice persona: 10582962]

Contents

1 Introduzione

1.1 Scopo del progetto

Il progetto di reti logiche dell'anno accademico 2019-2020 si basa sul metodo di codifica a bassa dissipazione di potenza detto "Working Zone". Il metodo Working Zone lavora sul Bus Indirizzi e si usa per codificare il valore di un indirizzo nel caso questo appartenga a certi intervalli noti: le Working Zone. Ci possono essere multiple Working Zone, ognuna delle quali parte da un indirizzo base e si estende per una dimensione fissa.

1.2 Specifiche generali

Vengono fornite 8 Working Zone e l'indirizzo da codificare. Ogni Working Zone parte dall'indirizzo base e si estende per una dimensione complessiva di 4 indirizzi (incluso quello base).
Si possono presentare due casi:

1. **Indirizzo non presente in nessuna Working Zone**
   In questo caso l'indirizzo codificato da restituire in output è così formato:

   WZ_BIT & **ADDR**

   - **WZ_BIT**: è il bit che indica se l'indirizzo appartiene o meno a qualche Working Zone e in questo caso vale 0.
   - **ADDR**: è l'indirizzo originale fornito in input.

2. **Indirizzo presente in una Working Zone**
   In questo caso l'indirizzo codificato da restituire in output è così formato:

   WZ_BIT & **WZ_NUM** & **WZ_OFFSET**

   - **WZ_BIT**: è il bit che indica se l'indirizzo appartiene o meno a qualche Working Zone e in questo caso vale 1.
   - **WZ_NUM**: è il numero della Working Zone a cui l'indirizzo appartiene.
   - **WZ_OFFSET**: è l'offset tra l'indirizzo base della Working Zone e l'indirizzo da codificare.

L'indirizzo da codificare è espresso su 7 bit, in modo tale da rappresentare tutti i valori che vanno da 0 a 127. Le Working Zone e l'indirizzo codificato sono espressi su 8 bit.

Poichè le Working Zone sono 8, WZ_NUM è espresso su 3 bit. Ne consegue che WZ_OFFSET è espresso su 4 bit.
In particolare WZ_OFFSET è codificato 1 hot così come segue:

- WZ_OFFSET = 0 è codificato come 0001;
- WZ_OFFSET = 1 è codificato come 0010;
- WZ_OFFSET = 2 è codificato come 0100;
- WZ_OFFSET = 3 è codificato come 1000;

1.3 Interfaccia del componente

```
entity poject_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;
```

- i_clk è il segnale di CLOCK;
- i_start è il segnale di START;
- i_rst è il segnale di RESET;
- i_data è il segnale che arriva dalla memoria in seguto ad una richiesta di lettura;
- o_address è il segnale di uscita che manda l'indirizzo alla memoria;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione
- o_en è il segnale di ENABLE da mandare alla memoria per abilitare la lettura
- o_we è il segnale di WRITE ENABLE da mandare alla memoria per abilitare la scritture
- o_data è il segnale di uscita che invia alla memoria l'indirizzo codificato

1.4 Dati e descrizione memoria

I dati, ciascuno di dimensione 8 bit (ADDR è esteso con uno 0 in posizione più significativa), sono memorizzati in una memoria RAM di 16 celle con indirizzamento al byte:

- Le cella di indirizzi dallo 0 al 7 contengono gli indirizzi base delle 8 Working Zone;
- La cella di indirizzo 8 contiene l'indirizzo da codificare;
- La cella di indirizzo 9 contiene l'indirizzo codificato che viene fornito in output;
- Le restanti celle sono inutilizzate;

Figure 1: schema della memoria

2 Design

L'esecuzione inzia con un segnale di i_rst posto a 1. Dopo l'abbassamento di i_rst, si attende che i_start diventi 1. Quest'ultimo rimmarrà alto fintanto che il segnale o_done è basso. Quindi, dopo aver portato a 1 il segnale o_en, si inizia con il prendere i dati dalla memoria. Successivamente si abilita il segnale di scrittura (o_we) e si cerca la Working Zone corrispondente all'indirizzo da codificare. A seconda che la Working Zone venga trovata o meno, si scrive sul seganle o_data l'indirizzo codificato nella maniera opportuna. Conclusa questa fase, si porta il segnale o_done a 1 per indicare di aver finito con l'esecuzione e in modo tale da poter far scendere prima i_start e di conseguenza ancora o_done. Quindi la macchina si pone in attesa di un nuovo segnale di inizio o di reset con la differenza che nel primo caso si procede a leggere la memoria solo nella posizione corrispondente all'indirizzo da codificare.
L'implementazione è stata sviluppata tramite un'unica architettura di tipo Behavioral. Di seguito sono illustrati i vari segnali interni utilizzati:

```
signal wz0 :  std_logic_vector(7 downto 0);
signal wz1 :  std_logic_vector(7 downto 0);
signal wz2 :  std_logic_vector(7 downto 0);
signal wz3 :  std_logic_vector(7 downto 0);
signal wz4 :  std_logic_vector(7 downto 0);
signal wz5 :  std_logic_vector(7 downto 0);
signal wz6 :  std_logic_vector(7 downto 0);
signal wz7 :  std_logic_vector(7 downto 0);
signal addr :  std_logic_vector(7 downto 0);
signal en_status :  std_logic;
signal we_status :  std_logic;
signal wz_found :  std_logic;
signal encode_status :  std_logic;
signal tmp_o_data :  std_logic_vector( 7 downto 0);
signal mem_counter :  integer;
signal status :  integer;

constant binary0 :  std_logic_vector(2 downto 0) := "000";
constant binary1 :  std_logic_vector(2 downto 0) := "001";
constant binary2 :  std_logic_vector(2 downto 0) := "010";
constant binary3 :  std_logic_vector(2 downto 0) := "011";
constant binary4 :  std_logic_vector(2 downto 0) := "100";
constant binary5 :  std_logic_vector(2 downto 0) := "101";
constant binary6 :  std_logic_vector(2 downto 0) := "110";
constant binary7 :  std_logic_vector(2 downto 0) := "111";
constant onehot0 :  std_logic_vector(3 downto 0) := "0001";
constant onehot1 :  std_logic_vector(3 downto 0) := "0010";
constant onehot2 :  std_logic_vector(3 downto 0) := "0100";
constant onehot3 :  std_logic_vector(3 downto 0) := "1000";
```

- wz0 : è utilizzato per memorizzare l'indirizzo base della prima Working Zone;
- wz1 : è utilizzato per memorizzare l'indirizzo base della seconda Working Zone;
- wz2 : è utilizzato per memorizzare l'indirizzo base della terza Working Zone;
- wz3 : è utilizzato per memorizzare l'indirizzo base della quarta Working Zone;
- wz4 : è utilizzato per memorizzare l'indirizzo base della quinta Working Zone;
- wz5 : è utilizzato per memorizzare l'indirizzo base della sensta Working Zone;

- `wz6` : è utilizzato per memorizzare l'indirizzo base della settima Working Zone;
- `wz7` : è utilizzato per memorizzare l'indirizzo base della ottava Working Zone;
- `addr` : è utilizzato per memorizzare l'indirizzo da codificare;
- `en_status` : è utilizzato per controllare il valore di `o_en`;
- `wn_status` : è utilizzato per controllare il valore di `o_we`;
- `wz_found` : è utilizzato per controllare se l'indirizzo è stato trovato in una delle Working Zone;
- `encode_status` : è utilizzato per controllare la fare si codifica;
- `tmp_o_data` : è utilizzato per memorizzare un valore temporaneo dell'indirizzo codificato;
- `mem_counter` : è utilizzato per realizzare il contatore che legge i valori dalla memoria;
- `status` : è utilizzato per distinguere le varie fasi di esecuzione della macchina;
- Le costanti sono state implementate escusivamente per una ragione di leggibilità del codice;

2.1 Stati della macchina

Le principali fasi di esecuzione sono scandite dal segnale `status`. Di seguito la descrizione precisa degli stati più interessanti della macchina.

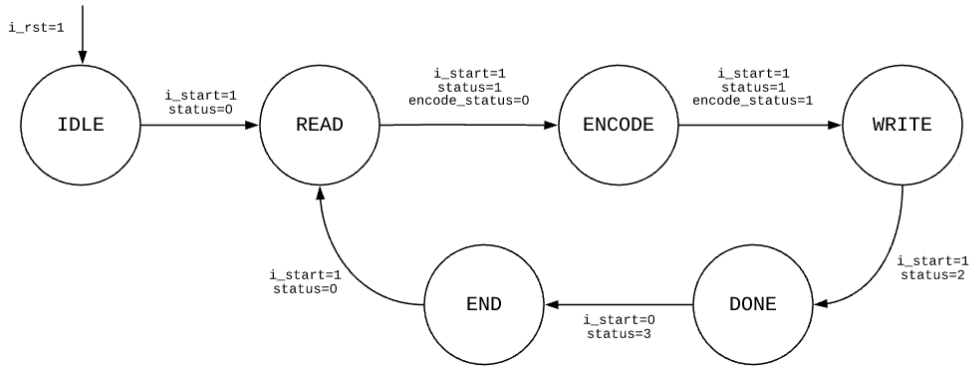

Figure 2: diagramma degli stati

2.1.1 IDLE: `i_rst` = 0

è lo stato si reset

2.1.2 READ: `i_start` = 1 e `status` = 0

Dopo un ciclo di clock utile per attivare la lettura tramite i segnali `en_status` e `o_en`, inizia il contatore che legge i dati dalla memoria. Ogni due cicli di clock viene posto in `o_address` l'indirizzo della memoria che contiene il valore che si vuole leggere al ciclo successivo. La scelta di leggere un dato ogni due cicli è stata presa al fine di evitare sfasamenti sulla lettura dei dati a causa di eventuali ritardi. Va precisato che se gli eventuali ritardi superano il periodo del clock la soluzione adottata non risulta efficace, ma dal momento che non è fornito alcun modo per verificare se il dato richiesto è stato effettivamente ricevuto, abbiamo assunto che tali ritardi siano frutto di un funzionamento non contemplato della macchina [usare solo come appunto per chiedere all'esercitatore].

2.1.3 ENCODE: `i_start` = 1, `status` = 1 e `encode_status` = 0

Dopo un ciclo di clock utile per attivare la scrittura tramite i segnali `we_status` e `o_we`, inizia la fase di codifica. Viene confrontato l'indirizzo da codificare con ogni set di Working Zone parallelamente e nel caso venga trovata una corrispondenza si scrive l'indirizzo codificato in `temp_o_data`. Il segnale `wz_found` serve per discriminare se la Working Zone è stata trovata o meno.

2.1.4 WRITE: `i_start` = 1, `status` = 1 e `encode_status` = 1

Questo è lo stato in cui viene scritto il risultato nella memoria. Grazie al segnale `wz_found` è possibile scrivere l'indirizzo codificato nella maniera opportuna.

2.1.5 DONE: `i_start` = 1 e `status` = 2

Finita l'elaborazione, si settano i vari segnali ai valori opportuni e si alva il segnale di `o_done` per notificare che l'esecuzione è stata completata.

2.1.6 END: `i_start` = 0 e `status` = 3

`i_start` è tornato a 0 quindi si riabbassa anche `o_done`.
I segnali `mem_counter` e `o_address` vengono settati in maniera tale entrare nel contatore nella posizione in cui viene letto dalla memoria l'indirizzo da codificare. Questo perchè gli indirizzi delle Working Zone non cambiano tra un segnale si start e un'altro ma solo quando viene resettata la macchina.

3 Risultati dei test

4 Conclusione

4.1 Risultati della sintesi

4.2 Ottimizzazioni

Una possibile ottimizzazione sarebbe quella di verificare che l'indirizzo appartenga ad una Working Zone man mano che si prende il dato dalla memoria in modo tale da interromere l'esecuzione tanto prima quanto viene trovata una corrispondenza. Il caso pessimo non cambia.