Progetto di Reti Logiche

Antonio Ercolani - 10621728
Riccardo Nannini - 10626268

Prof Fabio Salice - Anno Accademico 2019/2020

Indice

1 Introduzione **2**
1.1 Il metodo di codifica Working Zone . . . 2
1.2 Esempi di funzionamento . . . 3
1.3 L'interfaccia del componente . . . 4

2 Architettura **5**
2.1 Moduli . . . 5
2.1.1 State manager . . . 5
2.1.2 Memoria macchina a stati . . . 6
2.1.3 Difference calculator . . . 6
2.1.4 Difference evaluator . . . 6
2.1.5 Memoria ADDR . . . 6
2.2 Scelte progettuali . . . 6

3 Risultati sperimentali **7**

4 Simulazioni **7**
4.1 Casi limite . . . 7
4.2 Reset asincrono . . . 7
4.3 Esecuzioni multiple . . . 7
4.4 Test casuali . . . 7

5 Conclusioni **7**

1 Introduzione

Il progetto prevede la realizzazione di un componente hardware, tramite il linguaggio VHDL.

1.1 Il metodo di codifica Working Zone

Il progetto ruota attorno ai dati presenti in una memoria, con indirizzamento al byte, che si interfaccia con il componente da realizzare. Di seguito lo schema della memoria:

Indirizzo Memoria	Valore
0	Indirizzo Base WZ0
1	Indirizzo Base WZ1
2	Indirizzo Base WZ2
3	Indirizzo Base WZ3
4	Indirizzo Base WZ4
5	Indirizzo Base WZ5
6	Indirizzo Base WZ6
7	Indirizzo Base WZ7
8	Indirizzo da codificare
9	Indirizzo codificato

Figura 1: La memoria

Le prime 8 celle contengono gli indirizzi base delle **working-zone**. Una working-zone è un intervallo di indirizzi di dimensione fissa che parte dall'indirizzo base. Nel progetto in questione, il numero di bit degli indirizzi è 7, di conseguenza il range di indirizzi validi per una working-zone va da 0 a 127. Gli intervalli, invece, sono composti da 4 indirizzi, indirizzo base compreso. L'ottava cella contiene l'**indirizzo da codificare** (ADDR).

Lo scopo del progetto è quello di realizzare un componente che riesca a valutare l'appartenenza o meno di ADDR a una delle working zone. Questo viene fatto attraverso la codifica di un nuovo indirizzo, composto da 8 bit, che verrà scritto nella cella numero 9 della memoria.
La codifica segue il seguente schema:

ADDR NON appartiene a nessuna working-zone In questo caso la codifica è la seguente:

0	ADDR

Figura 2: Codifica indirizzo che NON appartiene alle working-zone

ADDR appartiene a una delle working-zone In questo caso invece la codifica è più complessa:

1	WZ_NUM	WZ_OFFSET

Figura 3: Codifica indirizzo che appartiene a una delle working-zone

In particolare:

- WZ_NUM (3 bit) indica il numero della working-zone, in binario naturale;
- WZ_OFFSET (4 bit) indica l'offset rispetto all'indirizzo base della working-zone, in codifica one-hot.

1.2 Esempi di funzionamento

Sono qui riportati due esempi di funzionamento con le relative spiegazioni.

INDIRIZZO MEMORIA	VALORE	//
0	5	WZ0
1	12	WZ1
2	77	WZ2
3	90	WZ3
4	24	WZ4
5	36	WZ5
6	120	WZ6
7	124	WZ7
8	27	ADDR
9	$1\ 100\ 1000_{bin}$	Indirizzo codificato

(a) ADDR appartiene alla working-zone 4

INDIRIZZO MEMORIA	VALORE	//
0	0	WZ0
1	121	WZ1
2	30	WZ2
3	40	WZ3
4	11	WZ4
5	92	WZ5
6	103	WZ6
7	7	WZ7
8	52	ADDR
9	$0\ 0110100_{bin}$	Indirizzo codificato

(b) ADDR non appartiene alle working-zone

Figura 4: Due esempi di funzionamento del metodo Working Zone

Esempio (a) In questo caso ADDR è uguale al quarto indirizzo della quarta working-zone. L'indirizzo codificato quindi sarà uguale a: **0** concatenato a **100** (working-zone 4, in binario naturale) concatenato a **1000** (codifica one hot indicante il quarto indirizzo dell'intervallo).

Esempio (b) Nell'esempio b, invece, ADDR non è contenuto in nessuna working-zone. Di conseguenza l'indirizzo codificato sarà composto da uno **0** concatenato a **0110100** (ADDR in binario naturale).

1.3 L'interfaccia del componente

Il componente da realizzare dovrà 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_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_clock** è il segnale di clock in ingresso;
- **i_start** è il segnale che dà inizio alla computazione;
- **i_rst** è il segnale di reset;
- **i_data** è il segnale (vettore di 8 bit) attraverso cui la memoria invia il dato richiesto, in lettura, al componente;
- **o_address** è il segnale (vettore di 16 bit) che manda l'indirizzo alla memoria;
- **o_done** è il segnale che comunica la fine della computazione e la scrittura del dato in memoria;
- **o_en** è il segnale che abilita la memoria alla lettura/scrittura;
- **o_we** è il segnale che comunica alla memoria che la stiamo accedendo in SCRITTURA (o_we = 1), diversamente, settando o_we = 0 l'accesso viene fatto in LETTURA;
- **o_data** è il segnale (vettore di 8 bit) attraverso cui il componente invia il dato da scrivere in memoria.

Note sulla gestione dei segnali
La computazione da parte del componente parte quando il segnale *i_start* viene alzato a 1. Fino a quando *o_done* non viene portato a 1, segnalando quindi la fine dell'esecuzione, *i_start* rimarrà alto. Successivamente quando *i_start* verrà messo a 0 anche *o_done* potrà essere messo a 0. Una volta fatto questo, *i_start* potrà essere rialzato a 1, facendo ripartire la codifica.

2 Architettura

Si è deciso di implementare una macchina a stati la cui logica di funzionamento è spiegata di seguito.
Quando il segnale di `i_start` viene portato ad 1, il componente passa dallo stato `READY` a quello `ADDRESS` in cui viene eseguita la lettura dell'indirizzo da codificare dalla RAM. Nei successivi cicli di clock il componente passa in successione gli stati chiamati `WZ0,WZ1,...,WZ7` in cui viene eseguito il fetch della working zone da RAM e il confronto con l'indirizzo. Questo fino a quando le working zones non finiscono o una corrispondenza viene trovata.

In ambedue i casi la codifica viene caricata sul segnale `o_data`, `o_done` viene portato ad 1 ed i rimanenti segnali per la scrittura adeguatamente configurati.

Il componente passa quindi allo stato di `WAIT` in cui aspetta che il segnale `i_start` venga portato a 0; non appena ciò accade viene abbassato a 0 il segnale `o_done` e la computazione può riprendere dallo stato `READY` non appena `i_start` viene asserito di nuovo.

Segue un'illustrazione dell'architettura del componente descritto nei **moduli** che lo compongono.

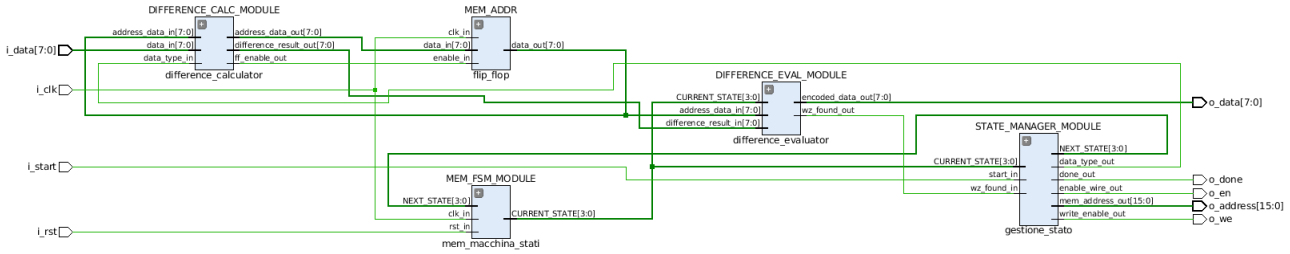

Figura 5: Moduli del componente

2.1 Moduli

2.1.1 State manager

Il modulo è un process che implementa la macchina a stati. In particolare si occupa della decisione dello stato futuro e del settaggio dei segnali in uscita relativi a lettura e scrittura sulla RAM in base al risultato del confronto tra indirizzo e working zone corrente.

Il diagramma della macchina ha 11 stati qui riportati:

- `READY`

  Stato iniziale in cui si aspetta un segnale di `i_start` per poter iniziare la computazione; è anche lo stato in cui si torna a fronte di un segnale `i_rst`.

- `ADDR`

  Stato che si occupa del fetch e salvataggio dell'***indirizzo*** da codificare.

- `WZ0,WZ1,...,WZ6`

  Stati che si occupano di recuperare l'indirizzo dell'n-esima ***working zone*** e effettuare il relativo confronto con l'indirizzo da codificare; in caso di successo viene prodotta la codifica in output, altrimenti avviene il passaggio alla working zone successiva.

- `WZ7`

  Simile agli altri stati WZ ma con la differenza che in caso di risultato negativo del confronto tra working zone e indirizzo verrà prodotto in output l'indirizzo originale preceduto dal `wz_bit` a 0.

- `WAIT`

  Stato raggiunto alla fine della computazione, indipendentemente dall'esito, in cui il componente aspetta che il segnale di `i_start` venga abbassato per poter tornare nello stato `READY` ed essere pronto ad una nuova computazione.

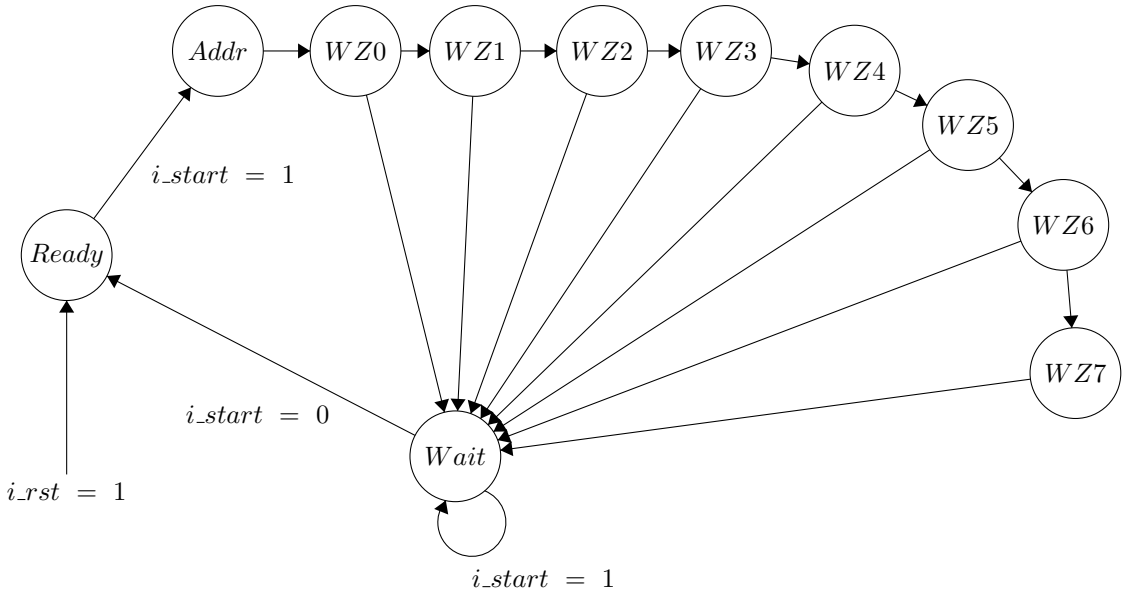

Figure 2: Diagramma della macchina a stati

2.1.2 Memoria macchina a stati

Il modulo è costituito da un process che gestisce la memoria della macchina a stati assegnando lo stato prossimo a quello corrente a meno di un reset, il quale fa portare lo stato a `READY`.

2.1.3 Difference calculator

Il modulo ha due funzioni principali: quando il componente si trova nello stato di `ADDRESS` il compito del modulo è quello di salvare l'indirizzo da codificare nel flip flop adibito a tale scopo. Quando invece lo stato corrente è uno tra `WZ0,WZ1,...,WZ7` il compito del modulo è quello di computare la **differenza** tra indirizzo da codificare e working zone corrente.

2.1.4 Difference evaluator

Il modulo, realizzato con un process, riceve la differenza calcolata da *difference calculator* e ne esegue una valutazione. Se la differenza è compresa tra 0 e 3 (l'indirizzo appartiene quindi alla working zone corrente), il modulo computa la codifica appropriata sull'apposito segnale di output. Altrimenti è l'indirizzo originario (preceduto da un bit a 0) a venire posto sul segnale di output; tale segnale verrà effettivamente scritto solamente se tutti i confronti con le working zone non sono andate a buon fine.

É anche compito del modulo segnalare allo *state manager* l'esito del confronto corrente.

2.1.5 Memoria ADDR

Modulo che implementa un flip flop standard in cui viene immagazzinato l'indirizzo da codificare (ADDR) per la computazione corrente.

2.2 Scelte progettuali

Il design ottenuto è frutto di scelte volte ad ottimizzare il più possibile l' **area occupata** dal componente.
Non vi è infatti alcuna memoria in cui vengono salvate le working zones; esse vengono lette dal componente ad ogni diversa computazione, anche se non sono state effettivamente cambiate a fronte di un reset.

Risulta evidente come il componente sia ottimizzato per situazioni in cui i reset sono molti e frequenti e dove le working zones cambiano frequentemente; al contrario perde in prestazioni rispetto ad un componente con memoria delle working zones in un contesto in cui quest'ultime cambiano di rado.

3 Risultati sperimentali

Una volta sintetizzato il componente possiede:

- **33 LUTs**
- **12 Flip Flops**

  Di cui 8 per salvare l'indirizzo da codificare e 4 per la memoria dello stato.

I risultati sono coerenti con le scelte progettuali fatte volte a minimizzare la superficie occupata e conseguentemente il numero di componenti.

4 Simulazioni

Per verificare il corretto funzionamento del componente abbiamo predisposto una serie di test, di diversa natura. Alcuni insistono su casi limite del metodo *Working Zone*, mentre altri sono di natura più "casuale".
Di seguito i test effettuati.

4.1 Casi limite

I primi test insistono su due casi limite del metodo, ovvero:

- ADDR "minimo" (0) che appartiene alla prima working zone, con WZ-OFFSET = $0001_{\text{one-hot}}$;

0	12	45	100	27	51	112	120	**0**
WZ0	WZ1	WZ2	WZ3	WZ4	WZ5	WZ6	WZ7	**ADDR**

Figura 6: Caso Limite 1

- ADDR "massimo" (127) che appartiene all'ultima working zone, con WZ-OFFSET = $1000_{\text{one-hot}}$.

0	12	45	100	27	51	112	**124**	**127**
WZ0	WZ1	WZ2	WZ3	WZ4	WZ5	WZ6	**WZ7**	**ADDR**

Figura 7: Caso Limite 2

4.2 Reset asincrono

Test nei quali viene inviato al componente un segnale di reset **durante** la valutazione di appartenenza alle working zone di ADDR.

4.3 Esecuzioni multiple

In questi test viene valutata la capacità del modulo di effettuare esecuzioni multiple con la possibilità di un cambiamento del valore di ADDR da un esecuzione all'altra.

4.4 Test casuali

Abbiamo, infine, sottoposto il componente a diversi test generati casualmente, per una maggiore copertura di casi, tramite uno script java. In questi test poteva variare sia il contenuto della memoria (valori di ADDR e working zone) che la frequenza dei vari segnali di controllo dell'esecuzione (*i_start* e *i_rst*).

5 Conclusioni

Il progetto è stato sviluppato rispettando le specifiche e coerentemente alle scelte progettuali che ci siamo imposti. La realizzazione del componente attraverso moduli separati ha permesso di dividere in sottoproblemi il progetto e, una volta specificate le interfacce, ci ha permesso di lavorare separatamente sui diversi moduli per poi integrare e testare il tutto assieme in un secondo momento.