Progetto di Reti Logiche (Prova Finale – CFU 1)

Prof. Fabio Salice - Anno 2019/2020

Lorenzo Giovanni Lacchini
Luca Minotti

Indice

1. Introduzione

1.1 Scopo del progetto

Siano dati 8 indirizzi base appartenenti ad un intervallo di indirizzi di dimensione fissa detta Working Zone, lo scopo del progetto è di implementare un componente hardware, descritto in VHDL, che preso in ingresso un indirizzo fornisca in output l'indirizzo opportunamente codificato mediante il metodo di codifica a bassa dissipazione di potenza denominato "*Working Zone*".

1.2 Specifiche del progetto

Il metodo di codifica "*Working Zone*" è un metodo pensato per il Bus Indirizzi che si una per trasformare il valore di un indirizzo quando questo viene trasmesso, se appartiene a certi intervalli (detti appunto working zone). Lo schema di codifica da implementare è il seguente:

- se l'indirizzo da trasmettere (ADDR) non appartiene a nessuna Working Zone, esso viene trasmesso così come è, e un bit addizionale rispetto ai bit di indirizzamento (WZ_BIT) viene messo a 0. In pratica dato ADDR, verrà trasmesso WZ_BIT=0 concatenato ad ADDR (WZ_BIT & ADDR, dove & è il simbolo di concatenazione);

- se l'indirizzo da trasmettere (ADDR) appartiene ad una Working Zone, il bit addizionale WZ_BIT è posto a 1, mentre i bit di indirizzo vengono divisi in 2 sotto campi rappresentanti:

    - Il numero della working-zone al quale l'indirizzo appartiene WZ_NUM, che sarà codificato in binario;
    - L'offset rispetto all'indirizzo di base della working zone WZ_OFFSET, codificato come one-hot (cioè il valore da rappresentare è equivalente all'unico bit a 1 della codifica).

In pratica dato ADDR, verrà trasmesso WZ_BIT=1 concatenato ad WZ_NUM e WZ_OFFSET (WZ_BIT & WZ_NUM & WZ_OFFSET, dove & è il simbolo di concatenazione).

Per comprendere meglio il funzionamento dell'algoritmo di codifica, sono stati riportati nella tabella sottostante due casistiche di funzionamento:

- *Caso 1*: il valore da codificare non è presente nella working zone e quindi in output avremo l'indirizzo ADDR concatenato a 0.
- *Caso 2*: il valore da codificare è presente nella working zone e quindi in output avremo la seguente concatenazione:

1	0	1	1	0	1	0	0
WZ_BIT	WZ_NUM			WZ_OFFSET (one hot)			

Caso 1 : valore non presente in WZ		Caso 2: valore presente in WZ		
Indirizzo Mem	**Valore**	**Indirizzo Mem**	**Valore**	**Commento**
0……0000	**00000100**	**0……0000**	**00000100**	Indirizzo base WZ 0
0……0001	**00001101**	**0……0001**	**00001101**	Indirizzo base WZ 1
0……0010	**00010110**	**0……0010**	**00010110**	Indirizzo base WZ 2
0……0011	**00011111**	**0……0011**	**00011111**	Indirizzo base WZ 3
0……0100	**00100101**	**0……0100**	**00100101**	Indirizzo base WZ 4
0……0101	**00101101**	**0……0101**	**00101101**	Indirizzo base WZ 5
0……0110	**01001101**	**0……0110**	**01001101**	Indirizzo base WZ 6
0……0111	**01011011**	**0……0111**	**01011011**	Indirizzo base WZ 7
0……0100	**00101010**	**0……0100**	**00100001**	ADDR da codificare
0……0101	**00101010**	**0……0101**	**10110100**	Valore in OUTPUT
16 bit	**8 bit**	**16 bit**	**8 bit**	

1.3 Interfaccia del componente

Il componente da descrivere ha un'interfaccia così definita:

```
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 generato dal test bench;
- i_start è il segnale START generato dal test bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- i_data è il segnale che arriva dalla memoria in seguito 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 e il dato di uscita scritto in memoria;

- o_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura)

- o_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poterci scrivere. Per leggere da memoria esso deve essere 0;

- o_data è il segnale di uscita dal componente verso la memoria.

Tale componente, inoltre, si interfaccia con un chip RAM in cui sono caricati tutti i dati necessari ad eseguire la codifica dell'indirizzo.

1.4 Dati e descrizione memoria

I dati, ciascuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento al byte:

- le posizioni in memoria da 0 a 7 sono usati per memorizzare gli otto indirizzi base delle working zone;
- la posizione 8 in memoria conterrà l'indirizzo da codificare (ADDR);
- la posizione 9 in memoria è usato per memorizzare, alla fine, il valore codificato secondo il metodo descritto nella specifica.

Indirizzo base WZ 0	Indirizzo 0
Indirizzo base WZ 1	Indirizzo 1
Indirizzo base WZ 2	Indirizzo 2
Indirizzo base WZ 3	Indirizzo 3
Indirizzo base WZ 4	Indirizzo 4
Indirizzo base WZ 5	Indirizzo 5
Indirizzo base WZ 6	Indirizzo 6
Indirizzo base WZ 7	Indirizzo 7
ADDR	Indirizzo 8
Indirizzo codificato	Indirizzo 9

Osservazione:
Anche l'indirizzo che è da specifica di 7 bit viene memorizzato su 8 bit ponendo sempre l'ottavo bit pari a zero.

2. Design

Quando il segnale i_start in ingresso viene portato a 1, il componente sviluppato inizia l'elaborazione spostandosi dallo stato IDLE al primo stato della computazione. Una volta terminata la computazione, e dopo aver scritto il risultato nella memoria (posizione 9), il componente alza il segnale o_done. Il test bench risponde abbassando i_start e successivamente il componente riporta a 0 il segnale o_done. Quindi il componente ritorna nello stato IDLE in attesa che il segnale i_start torni al valore alto.

2.1 Stati della macchina

La macchina costruita è composta da 7 stati (una versione semplificata dalla macchina è illustrata in *Figura 1*). Qui di seguito è fornita una breve descrizione di ciascuno di essi.

2.1.1 IDLE **state**

Stato iniziale in cui si attende il segnale i_start, ricevuto il quale viene richiesta la lettura della cella di memoria (posizione 8) dove è contenuto il valore (indirizzo) da codificare. In caso venga alzato il segnale i_rst si torna in questo stato.

2.1.2 WAIT_ADDR **state**

Stato in cui si attende (al ciclo di clock successivo) la lettura dell'indirizzo da codificare.

2.1.3 READ_ADDR **state**

Stato in cui avviene la lettura dell'indirizzo da codificare ed è richiesta la lettura dell'indirizzo base della prima working zone.

2.1.4 WAIT_WZ **state**

Stato in cui si attende (al ciclo di clock successivo) la lettura dell'indirizzo base della working zone.

2.1.5 READ_WZ **state**

Stato in cui è letto l'indirizzo base della working zone ed è controllata l'appartenenza dell'indirizzo da codificare a tale WZ. Se la verifica ha esito positivo si procede eseguendo la codifica, in caso contrario lo stato richiede la lettura dell'indirizzo base della successiva working zone (se presente). Nel caso in cui le working zone siano terminate, si applica la codifica che concatena all'indirizzo da codificare uno 0.

2.1.6 WRITE_RESULT **state**

Stato in cui viene scritto l'indirizzo codificato (mediante il metodo "Working Zone") all'indirizzo della memoria 9 e viene alzato il segnale o_done.

2.1.7 DONE **state**

Stato in cui si attende che la memoria abbassi i_start per poter abbassare o_done e tornare nello stato IDLE.

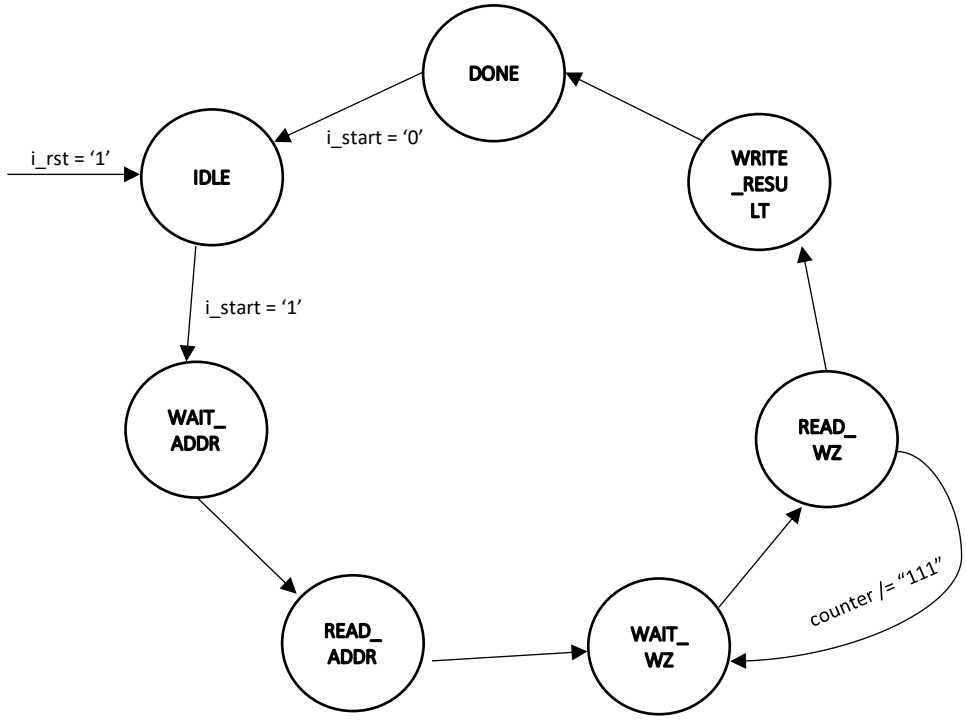


Figura 1: Finite state machine con le principali transizioni

2.2 Scelte progettuali

La principale scelta progettuale effettuata è stata quella di descrivere il componente con due processi:

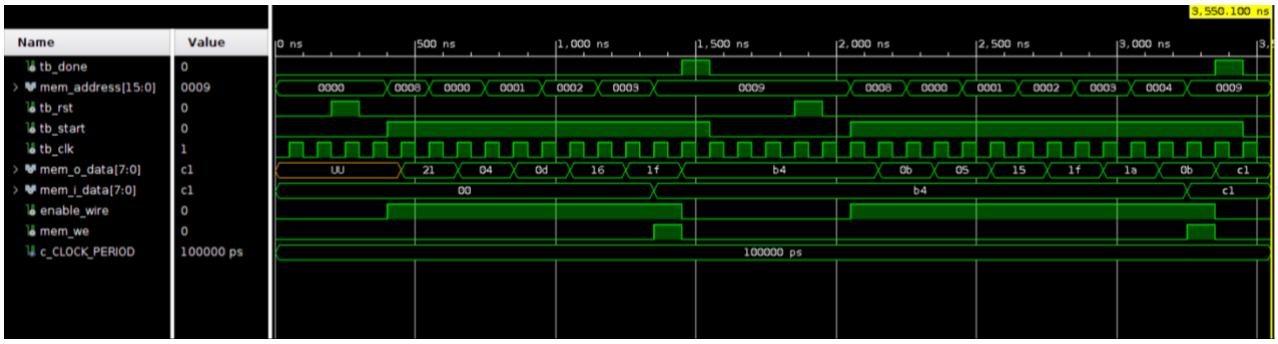
1. il primo processo (controller) rappresenta la parte sequenziale della macchina;
2. Il secondo (fsm) rappresenta la *finite state machine* che analizza i segnali in ingresso e lo stato corrente per determinare il prossimo stato in cui evolverà il sistema (data dalla relazione Q x I → Q).

Inoltre abbiamo assunto che la frequenza di aggiornamento degli indirizzi di base della working zone sia sufficientemente elevata da rendere svantaggiosa alcuna ottimizzazione in tal senso ad esempio la memorizzazione interna degli indirizzi base delle working zone. Non avendo, da specifica, restrizioni per tempo di esecuzione e area utilizzata, abbiamo cercato di focalizzarci nella realizzazione di una soluzione semplice a livello di algoritmo.

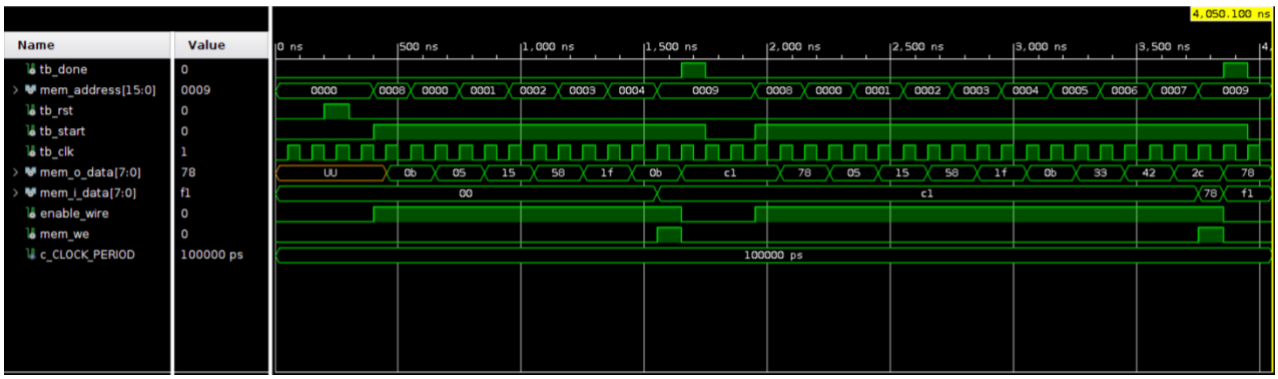
3. Risultati dei test

Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con i test bench fornitoci come esempio, abbiamo creato altri test in modo da cercare di massimizzare la copertura di tutti i possibili cammini che la macchina può effetturare durante la computazione.
A seguire è fornita una breve descrizione dei test utilizzati e per alcuni è presente il relativo screenshot che mostra l'andamento dei segnali durante la simulazione:

- ***Multi-reset test*** *: esegue un reset preliminare e successivamente conversioni ripetute separate da un segnale di reset che cambia il contenuto della ram;*



- ***Multi-start test*** *: esegue in successione delle conversioni mantenendo gli stessi indirizzzi base della working zone.*



- ***Indirizzi adiacenti*** : verifica la correttezza della codifica in caso di indirizzi di working zone adiacenti tra loro;

- ***Indirizzi limite*** : verifica la correttezza della codifica nel caso in cui:
  - l'indirizzo da convertire abbia offset minimo (WZ_OFFSET 0) dalla prima WZ (WZ0);
  - l'indirizzo da convertire abbia offset massimo (WZ_OFFSET 4) dall'ultima WZ (WZ7).

Oltre ai test appena descritti, abbiamo deciso di simulare anche numerosi test randomici per testare ulteriormente il nostro componente. Attraverso un software in C abbiamo generato una test bench con un gran numero di casi e li abbiamo simulati in modo automatizzato utilizzando uno script TCL fornito da Vivado in modalità batch. L'output dello script viene poi salvato in un file di testo e attraverso un comando di ricerca si controlla che i messaggi delle assertions di ogni simulazione siano di superamento per ciascun test.

4. Conclusioni

4.1 Risultati della sintesi

Il componente sintetizzato supera correttamente tutti i test specificati nelle due simulazioni: *Behavioral* e *Post-Synthesis Functional*.

Di seguito è possibile vedere un confronto tra i tempi di simulazione dei due test descritti nella sezione precedente:

- ***3550 ns*** : multi-reset;
- ***4050 ns***: multi-start.

4.2 Ottimizzazioni

Le ottimizzazioni che abbiamo attuato sono state concentrate principalmente nella riduzione del numero di stati della *finite state machine.*