Prova Finale di Reti Logiche

Politecnico di Milano

Luca Quagliana

Matricola: 866468

Indice

1. Introduzione e specifiche 3
1.1. Descrizione del progetto 3
1.2. Contenuto della memoria 3
1.3. Entity VHDL del componente 4

2. Impostazione dell'implementazione 4
2.1. Strategia di risoluzione 4
2.2. Codice VHDL 4

3. Algoritmo e macchina a stati 5
3.1. Descrizione dell'algoritmo 5
3.2. Analisi degli stati 5
3.3. Schema della macchina a stati 7

4. Test del componente 8
4.1. Panoramica 8
4.2. Report di sintesi 8
4.3. Test effettuati 8

5. Conclusioni 9

1. Introduzione e specifiche

1.1 Descrizione del progetto

Una Working Zone è definita come un intervallo di indirizzi di dimensione fissa (Dwz) che parte da un indirizzo base. All'interno dello schema di codifica possono esistere multiple Working Zone (Nwz). Lo schema modificato 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 i n 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 i l valore da rappresentare è equivalente all'unico bit a 1 della codifica).

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

Il modulo implementato leggerà l'indirizzo da codificare e gli 8 indirizzi base delle Working Zone e dovrà produrre l'indirizzo opportunamente codificato.

1.2 Contenuto della memoria

I dati ciascuno di dimensione 8 bit sono memorizzati in una memoria con indirizzamento al Byte partendo dalla posizione 0. Anche l'indirizzo che è da specifica di 7 bit viene memorizzato su 8 bit. Il valore dell'ottavo bit sarà sempre zero.

Le posizioni in memoria da 0 a 7 sono usate per memorizzare gli otto indirizzi base delle Working Zone:

- 0 (Indirizzo Base WZ 0)
- 1 (Indirizzo Base WZ 1)
- 2 (Indirizzo Base WZ 2)
- 3 (Indirizzo Base WZ 3)
- 4 (Indirizzo Base WZ 4)
- 5 (Indirizzo Base WZ 5)
- 6 (Indirizzo Base WZ 6)
- 7 (Indirizzo Base WZ 7)

La posizione in memoria 8 avrà al suo interno il valore (indirizzo) da codificare (ADDR) e la posizione in memoria 9 è quella che deve essere usata per scrivere, alla fine, il valore codificato secondo le regole precedenti.

1.3 Entity VHDL 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;
```

Analisi dei segnali:

- i_clk è il clock di sistema generato dal testbench
- i_start è il segnale di start che inizia la computazione
- i_rst è il segnale di reset che porta la macchina allo stato iniziale
- i_data è un vettore contenente i dati letti dalla memoria
- o_address è un vettore in cui compare l'indirizzo da indicare alla memoria sia in ingresso che in uscita
- o_done è il segnale di terminazione della computazione
- o_en è il segnale di enable della memoria
- o_we è il segnale di enable della scrittura in memoria (se o_en=0 è ininfluente)
- o_data è un vettore contenente i dati da scrivere in memoria

2. Impostazione dell'implementazione

2.1 Strategia di risoluzione

Per la realizzazione del componente si sono seguite varie fasi:

- In primo luogo si sono analizzate le specifiche in modo da comprendere il problema nel dettaglio.
- Sono stati pensati vari algoritmi e si è scelto il più adatto.
- Individuato l'algoritmo, è stato suddiviso in modo da poterlo rappresentare tramite una macchina a stati.
- Successivamente si è sviluppato quanto descritto sopra in codice VHDL.
- Infine si è sottoposto il componente a vari test così da garantirne il corretto funzionamento.

2.2 Codice VHDL

Per lo sviluppo del codice VHDL si è deciso di descrivere il componente utilizzando il metodo Behavioral, visto il vantaggio di essere molto vicino alla soluzione algoritmica del problema.

È stato utilizzato un unico process per descrivere la macchina a stati, mentre per gestire le informazioni necessarie alla risoluzione del problema sono state utilizzate delle variabili di supporto interne al processo:

- Variabili di tipo vettore (8 bit) dedicate al valore delle Working Zone: WZ0, WZ1, WZ2, WZ3, WZ4, WZ5, WZ6, WZ7;
- Variabile di tipo vettore (8 bit) dedicata al valore da codificare: ADDR;
- Variabile WZBIT;
- Variabile di tipo vettore (3 bit): WZNUM;
- Variabile di tipo vettore (4 bit): WZOFFSET.

3. Algoritmo e macchina a stati

3.1 Descrizione dell'algoritmo

In primo luogo si immagazzinano i valori delle 8 Working Zone, per poi memorizzare anche il valore da codificare (ADDR).
Successivamente si confronta il valore di ADDR con tutti i valori contenuti nel vettore dedicato a ciascuna Working Zone, modificando di conseguenza le variabili interne al processo:

- WZBIT: è inizializzato a 0 e viene posto a 1 se ADDR risulta uguale, maggiorato di 1, maggiorato di 2 oppure maggiorato di 3 rispetto al valore di base di una Working Zone;
- WZNUM: se WZBIT ha valore 1, rappresenta il numero della Working Zone (WZ0->000, WZ1->001, WZ2->010, WZ3->011, WZ4->100, WZ5->101, WZ6->110, WZ7->111);
- WZOFFSET: se WZBIT ha valore 1, rappresenta il discostamento di ADDR dal valore di base della Working Zone ed è codificato come One-hot (se il valore di ADDR risulta uguale a quello contenuto nel vettore dedicato a una Working Zone, WZOFFSET= 0000; se risulta maggiorato di 1, WZOFFSET= 0010; se risulta maggiorato di 2, WZOFFSET= 0100; se risulta maggiorato di 3, WZOFFSET= 1000).

Successivamente si rileva il valore di WZBIT e nel caso in cui valga 0 (ADDR non è contenuto in nessuna Working Zone) si scrive in output il valore di ADDR, mentre se vale 1 si scrive in output la concatenazione WZBIT & WZNUM & WZOFFSET.
Arrivati alla fine della computazione si attende un segnale da parte del programma per stabilire se acquisire un nuovo valore da codificare mantenendo in memoria le precedenti Working Zone, oppure se ricominciare l'intero algoritmo.

3.2 Analisi degli stati

RST

Lo stato RST è lo stato iniziale e di reset. In questo stato tutte le variabili utilizzate nel processo sono inizializzate ai loro valori di partenza. Quando si è nello stato RTS, si attende che il segnale i_start diventi 1 per proseguire con la computazione.
RST può essere attivato in modo asincrono in ogni momento portando il segnale i_rst a 1.

REQUEST - WAIT - READ

L'acquisizione dei dati avviene mediante questa successione di stati:

- Nello stato REQUEST è inoltrata una richiesta di lettura alla memoria portando a 1 il segnale o_en e forzando sul segnale o_address l'indirizzo che si vuole leggere;
- Nello stato WAIT si attende un ciclo di clock in modo che la memoria abbia tempo di portare correttamente sul segnale i_data l'informazione richiesta;
- Nello stato WRITE viene letta dal segnale i_data l'informazione passata dalla memoria e viene impostato nuovamente a 0 il segnale o_en.

I dati coinvolti in questo processo sono gli indirizzi delle Working Zone e il valore da codificare.

VERIFY

Lo stato di VERIFY è il vero e proprio corpo dell'algoritmo risolutivo, in cui si assegnano i valori alle variabili di processo WZBIT, WZNUM, WZOFFSET (secondo quanto spiegato pocanzi).

CHOOSEOUT

Questo stato è di supporto alla computazione e serve per indirizzare l'algoritmo al corretto stato successivo. Viene rilevato il valore di WZBIT e: nel caso questo sia uguale a 0, lo stato prossimo sarà REQUESTOUT1; nel caso questo sia uguale a 1, lo stato prossimo sarà REQUESTOUT2.

REQUESTOUT - WAITOUT - WRITEOUT

Come per l'acquisizione dei dati in ingresso, anche per la scrittura in output si ricorre a un meccanismo simile:

- Nello stato REQUESTOUT viene inviata una richiesta di scrittura alla memoria ponendo sul segnale o_data l'informazione da scrivere, portando l'indirizzo interessato sul segnale o_address e ponendo a 1 i segnali o_en e o_we;
- Nello stato WAITOUT si attende un ciclo di clock in modo che la memoria abbia tempo di scrivere correttamente l'informazione contenuta in o_data;
- Nello stato WRITEOUT termina la fase di scrittura e si riportano a 1 i segnali o_en e o_we.

Sono presenti due successioni di questi tre stati in quanto una gestisce l'output nel caso in cui il valore di ADDR non sia contenuto in nessuna Working Zone (REQUESTOUT1-WAITOUT1-WRITEOUT1), mentre l'altra gestisce l'output nel caso in cui il valore di ADDR appartenga ad una Working Zone (REQUESTOUT2-WAITOUT2-WRITEOUT2).

DONE

Nello stato di DONE si porta a 1 il segnale o_done per segnalare al programma che la computazione è terminata correttamente. In seguito di passa allo stato START_WAIT.

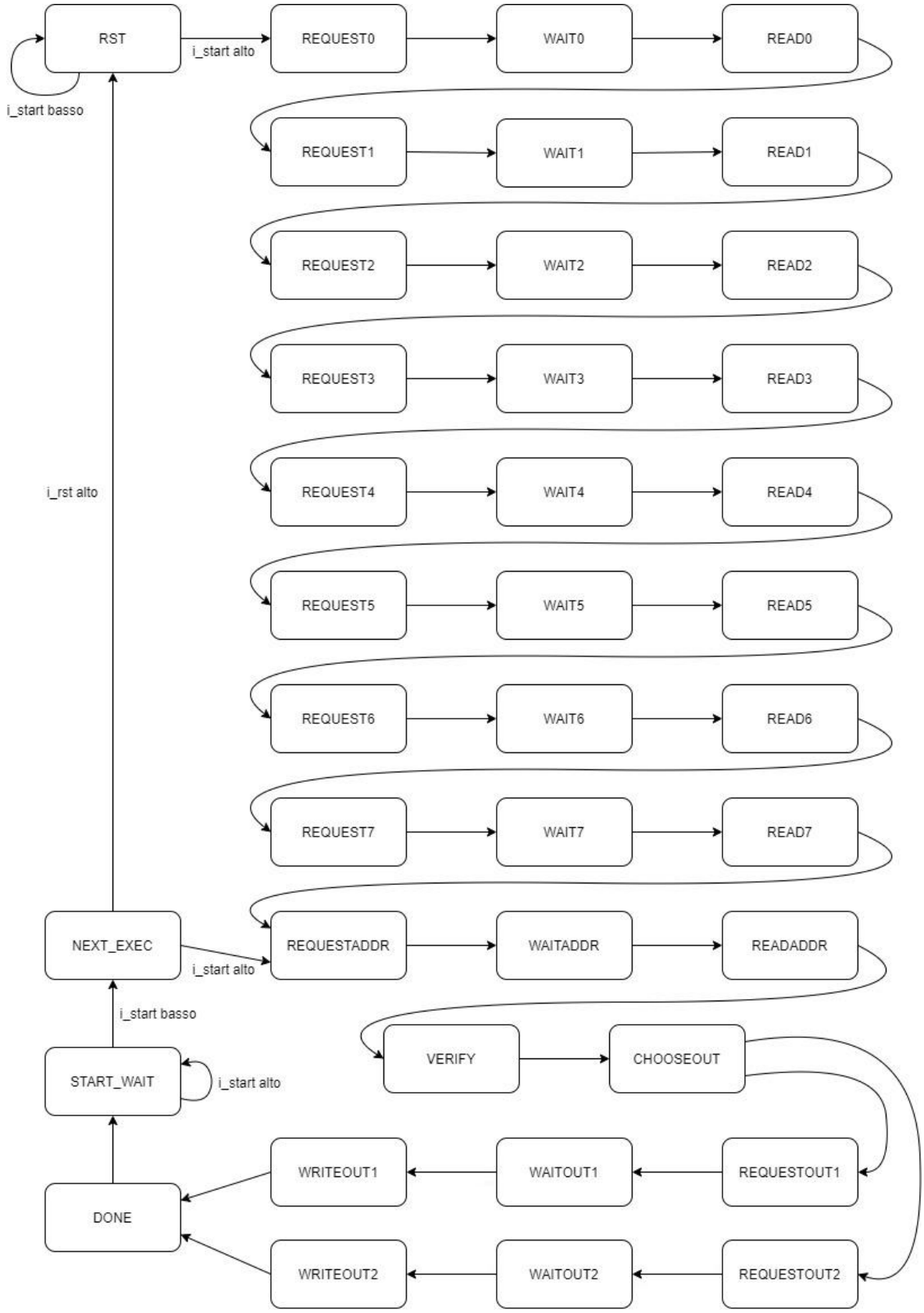
START_WAIT

Lo stato di START_WAIT è lo stato in cui si attende che il segnale i_start sia portato a 0 dal programma per poter riportare a 0 anche il segnale o_done. A questo punto l'algoritmo è giunto al termine e si passa allo stato NEXT_EXEC.

NEXT_EXEC

Questo stato è lo stato finale della macchina e attende un segnale dal programma: se viene portato a 1 il segnale di i_start, la computazione riprenderà dallo stato REQUESTADDR (si acquisisce un nuovo valore da codificare mantenendo le Working Zone già memorizzate); se viene portato a 1 il segnale di i_rst, si ritorna allo stato RST e si attende un nuovo segnale di i_start.

3.3 Schema della macchina a stati



Nota: per questione di ordine non sono state disegnate le frecce che portano ogni stato allo stato di reset.

4. Test del componente

4.1 Panoramica

Xilinx VIVADO consente di eseguire vari tipi di simulazione per testare un componente:

- Simulazione di tipo Behavioral: permette di verificare il funzionamento analizzando la correttezza della logica implementata senza basarsi su una sintesi virtuale del componente.
- Simulazione di tipo Post-Synthesis Functional: viene eseguita dopo la Synthesis del componente e permette di realizzarne un modello virtuale per poi testarlo come se fosse reale, ma senza considerare ritardi sulle logiche combinatorie o sequenziali.
- Simulazione di tipo Post-Synthesis Timing: è analoga alla simulazione Post-Synthesis Functional, ma considera i ritardi introdotti dalle logiche combinatorie e sequenziali reali del componente.

4.2 Report di sintesi

Report Cell Usage:

	Cell	Count
1	BUFG	1
2	LUT2	29
3	LUT3	36
4	LUT4	55
5	LUT5	77
6	LUT6	130
7	FDRE	134
8	IBUF	11
9	OBUF	27

Report Instance Areas:

	Instance	Module	Cells
1	top		500

Synthesis finished with 0 errors, 0 critical warnings and 0 warnings.

4.3 Test effettuati

Per ogni test effettuato sono state avviate simulazioni di tipo Behavioral e di tipo Post-Synthesis, sia Functional che Timing, e ognuna ha prodotto esiti positivi.

Inizialmente si è testato il componente quantitativamente, sottoponendolo a numerosi test generici in cui i valori di base delle Working Zone e il valore da codificare ADDR sono stati generati casualmente: in questi test il risultato ottenuto in output è sempre stato corretto sia nel caso di test con start singoli sia nel caso di test con start multipli.

Tra questi test si è verificato anche il corretto funzionamento del componente nel caso in cui esso sia sottoposto a computazioni con segnale di reset asincrono, ottenendo sempre feedback positivi.

Successivamente si sono cercati casi limite o critici a cui sottoporre il componente per verificare qualitativamente l'efficienza dell'algoritmo implementato: per esempio nel caso in cui il valore da codificare ADDR sia 127 (01111111) e appartenga ad una Working Zone che ha come valore di base il massimo consentito, cioè 124 (01111100).

Per il modo in cui è stato strutturato, anche i casi più anomali non sottopongono l'algoritmo a scelte ambigue: siccome da specifica il valore di base massimo di una Working Zone può essere 124 e l'output ha sempre come primo bit WZBIT, necessariamente il massimo valore ammissibile da codificare dovrà assumere valore massimo 127 (eccedendo questo valore si genererebbe un overflow).
In definitiva, ammettendo dati coerenti con quanto indicato nella disposizione del progetto, il componente supererà qualsiasi tipo di test senza generare errori.

5. Conclusioni

Nel codice sono stati inseriti commenti per rendere chiari tutti i passaggi effettuati e per garantire una corretta interpretazione delle scelte progettuali.
Nello sviluppo del componente si è scelto di prestare attenzione alla corretta gestione di tempi e ritardi, in modo da garantire il funzionamento anche per i test effettuati mediante simulazione Post-Synthesis Timing (motivo per cui sono stati utilizzati gli stati WAIT).