Prova finale di Reti Logiche

Matteo Laini
Matteo Macaluso

A.A. 2021-22

Matricole:	935359 - 933782
Codici Persona:	10683821 - 10656537
Docente:	Fabio Salice

POLITECNICO
MILANO 1863

Indice

1 Introduzione

1.1 Requisiti del progetto

La specifica della Prova Finale (Progetto di Reti Logiche) 2022 chiede di implementare un modulo hardware (sviluppato in VHDL) che riceva in ingresso una sequenza continua di W parole da 8 bit e che restituisca in uscita una sequenza di Z parole (sempre da 8 bit), utilizzando un codificatore convoluzionale.
Il componente deve essere in grado di leggere e serializzare ognuna delle parole in ingresso in un flusso continuo da 1 bit. Dopo aver applicato il codice convoluzionale 1/2, deve generare un flusso ottenuto come concatenamento alternato dei due bit di uscita del convolutore. Infine deve parallelizzare su 8 bit il flusso ottenuto e salvare il risultato in memoria.

1.2 Ipotesi progettuali

La memoria utilizzata ha un indirizzamento al Byte. All'indirizzo 0 è salvato il numero di parole di cui è richiesta l'elaborazione. Il contenuto delle parole si trova nell'indirizzo 1 e nei successivi. Il salvataggio delle parole in uscita è richiesto che avvenga a partire dall'indirizzo 1000. In caso di elaborazioni consecutive l'indirizzo di uscita non viene ripristinato al valore 1000.
La dimensione massima della sequenza di ingresso è 255 byte.

1.3 Interfaccia del componente

È richiesto che il componente rispetti la seguente interfaccia standard VHDL:

```
entity project_reti_logiche is
    port(
        i_clk : in std_logic;
        i_rst : in std_logic;
        i_start : 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 TestBench;
- `i_rst` è il segnale di **Reset** che inizializza la macchina prima del segnale di Start;
- `i_start` è il segnale di **Start** generato dal TestBench;
- `i_data` è il segnale che arriva dalla memoria in seguito alla 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 l'avvenuta scrittura in memoria dei dati;
- `o_en` è il segnale di **Enable** da mandare alla memoria per poter comunicare, posto a 1 sia per leggere che per scrivere;
- `o_we` è il segnale di **Write Enable** da mandare alla memoria per leggere (=0) o per scrivere (=1);
- `o_data` è il segnale di uscita dal componente alla memoria.

1.4 Esempio

Di seguito un esempio con il contenuto della memoria in seguito a una elaborazione. Si ricordi che i valori, qui rappresentati in decimale per chiarezza, sono memorizzati in codifica binaria 8 bit senza segno.
L'esempio è tratto da una simulazione fornita dal docente (tb_esempio_3.vhd).

Si noti che all'indirizzo 0 è contenuto il numero di parole da codificare, mentre dall'indirizzo 1000 troviamo l'output dell'elaborazione.

Indirizzo di memoria	Contenuto
0	3
1	112
2	164
3	45
...	...
1000	57
1001	176
1002	209
1003	247
1004	13
1005	40

Tabella 1: Contenuto della memoria dopo un'elaborazione

2 Implementazione

Si è scelto di implementare un modulo *single-process* tramite architettura *behavioral* (comportamentale) in linguaggio VHDL.

2.1 Descrizione ad Alto Livello

Da un'ottica di alto livello, l'implementazione esegue i seguenti passi:

- Legge il primo indirizzo e salva il numero di parole da codificare;
- Verifica il valore numerico: se 0 termina, altrimenti procede al passo successivo;
- Legge il secondo indirizzo e salva la prima parola da codificare;
- Converte la prima parola generando due sequenze separate;
- Salva i due risultati della codifica negli indirizzi corretti;
- Verifica che l'elaborazione sia terminata e lo comunica con i relativi bit di comando, altrimenti si prepara a leggere una nuova parola dalla memoria;
- Termina l'elaborazione e si prepara a ricevere un input di una nuova elaborazione.

Per questo algoritmo si è scelta un'implementazione costituita da una macchina a stati finiti, che gestisce tutto il processo di lettura, codifica e scrittura.

2.2 Macchina a Stati Finiti

Segue una descrizione degli stati formali della Macchina a Stati Finiti:

- `START`: Stato di avvio in cui la macchina attende che il segnale i_start venga posto a 1. Vengono anche settati i parametri di lettura e il primo indirizzo di lettura.
- `PREPARE_READ_NUMBER`: Stato in cui si attende il contenuto della memoria.
- `READ_WORDS_NUMBER`: Stato in cui viene salvato il contenuto della memoria. Viene anche inizializzato il contatore a 0.
- `CHECK_WORDS_NUMBER`: Stato in cui si controlla il numero salvato. Se 0 l'elaborazione termina, altrimenti si passa al prossimo stato.
- `START_WORD_READ`: Stato in cui viene incrementato il contatore di 1.
- `SET_WORD_ADDRESS`: Stato in cui viene settato l'indirizzo di lettura della parola da codificare.
- `PREPARE_READ_WORD`: Stato in cui si attende il contenuto della memoria.
- `READ_WORD`: Stato in cui viene salvato il contenuto della memoria.
- `CONVERTER`: Stato in cui viene codificata la parola letta attraverso il convolutore 1/2. Viene anche settato l'indirizzo di salvataggio.
- `SAVE_FIRST`: Stato in cui viene salvata la prima parte della codifica.
- `PREPARE_ADDRESS`: Stato in cui si attende che il segnale o_data si aggiorni.
- `SAVE_SECOND`: Stato in cui viene salvata la seconda parte della codifica. Viene anche incrementato il contatore di 1.
- `CHECK_COUNTER`: Stato in cui viene controllato il valore del contatore. Se quest'ultimo è maggiore del numero di parole da codificare (salvato nello stato `READ_WORDS_NUMBER`) l'elaborazione termina. Altrimenti si procede a leggere la parola successiva.

- `END_ELABORATION`: Stato in cui viene portato a 1 il segnale o_done, che comunica la fine dell'elaborazione.
- `FINAL`: Stato di terminazione. Se il segnale i_start è posto a 1, la macchina riparte dallo stato `START`.

Nella pagina seguente è riportato il disegno dell'automa. Sono state usate le seguenti abbreviazioni degli stati:

START:	START
PREPARE_READ_NUMBER:	P_RD_N
READ_WORDS_NUMBER:	RD_W_N
CHECK_WORDS_NUMBER:	C_W_N
START_WORD_READ:	S_W_RD
SET_WORD_ADDRESS:	S_W_ADD
PREPARE_READ_WORD:	P_RD_W
READ_WORD:	RD_W
CONVERTER:	CONV
SAVE_FIRST:	SAVE1
PREPARE_ADDRESS:	P_ADD
SAVE_SECOND:	SAVE2
CHECK_COUNTER:	C_COUNT
END_ELABORATION:	END_E
FINAL:	FINAL

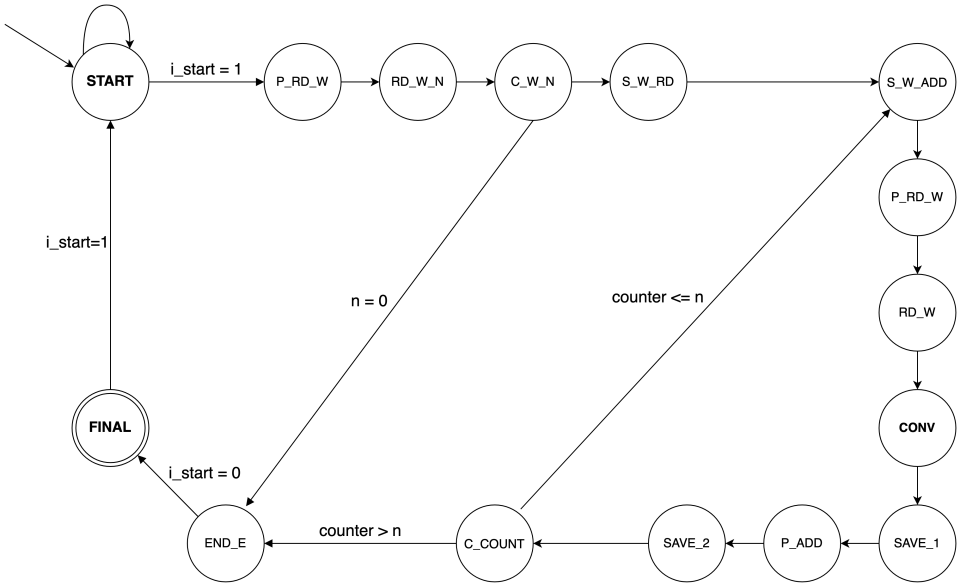

Figura 1: Rappresentazione della Macchina a Stati Finiti

2.3 Schema dell'implementazione

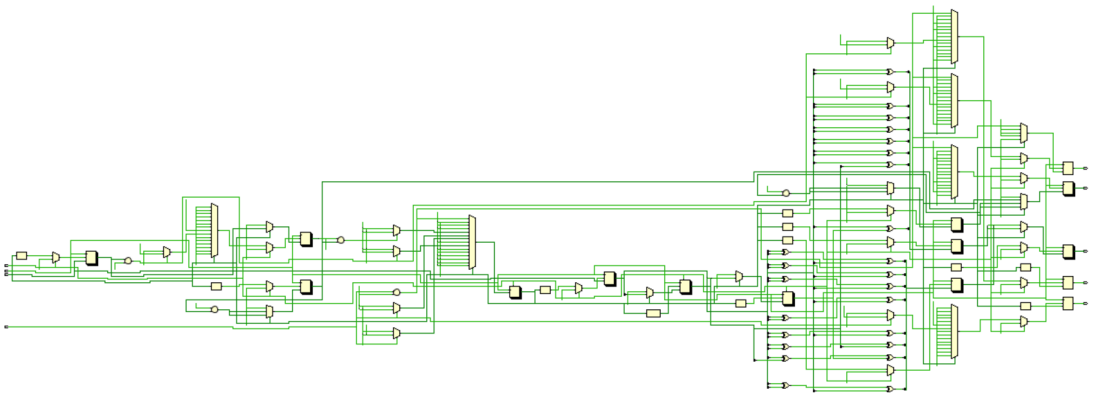

Figura 2: Schema dell'implementazione

3 Risultati sperimentali

3.1 Report di Sintesi

Dopo l'analisi del report di sintesi, si è verificato l'utilizzo dei seguenti componenti:

- LUT: 123
- Flip Flop: 105

Tutti i registri utilizzati sono Flip Flop e non vi è alcun Latch, che avrebbe potuto causare problemi nei test post sintesi.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	123	0	134600	0.09
LUT as Logic	123	0	134600	0.09
LUT as Memory	0	0	46200	0.00
Slice Registers	105	0	269200	0.04
Register as Flip Flop	105	0	269200	0.04
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

3.2 Simulazioni

Si è verificato il corretto funzionamento del componente tramite diversi Test-Bench forniti dal docente e altri generati grazie a un tool fornito da colleghi. Le simulazioni sono state utili ai fini di verificare i casi limite del progetto e hanno avuto tutte esito positivo.

Di seguito vengono riportati i tempi di esecuzione per i test più significativi (in behavioral e in post-synthesis):

- Tre Reset: quattro run consecutive con reset sul primo
  - Behavioral: 14250000ps
  - Post-Synthesis: 14950100ps

- Seq Min: sequenza di lunghezza nulla
  - Behavioral: 1050000ps
  - Post-Synthesis: 1150100ps
- Seq Max: sequenza di lunghezza massima
  - Behavioral: 205150000ps
  - PostSynthesis: 205250100ps

4 Conclusioni

Si ritiene che l'architettura rispetti le specifiche di progetto assegnate, in quanto porta a termine senza problemi sia i test forniti dal professore, che quelli da noi generati. Inoltre l'architettura è stata pensata sulla base di una Macchina a Stati Finiti, evitando la generazione di Latch, che avrebbero potuto creare l'instaurarsi di cicli infiniti.