Prova Finale (Progetto di Reti Logiche)

Politecnico di Milano
Anno Accademico 2021/2022

Prof. Fabio Salice

Riccardo Inghilleri (Codice Persona 10713236 - Matricola 937011)

Indice

1 Introduzione

1.1 Obiettivo del progetto

L'obiettivo della Prova Finale di Reti Logiche è quello di implementare un modulo in linguaggio VHDL in grado di leggere da una memoria una sequenza continua di W parole, elaborarle mediante un codice convoluzionale 1/2, e infine scrivere la sequenza d'uscita Z in memoria.

Un codice convoluzionale 1/2 è un tipo di codifica che a un bit in ingresso fa corrispondere 2 bit in uscita ed è utilizzato in diverse applicazioni con lo scopo di ottenere un trasferimento di dati molto affidabile come ad esempio nel video digitale, nella radio, nella telefonia mobile e nelle comunicazioni satellitari.

1.2 Specifica

Il modulo da implementare riceve in ingresso una sequenza continua di W parole da 8 bit ciascuna; ogni parola viene serializzata generando un flusso continuo di bit $U(8 * W)$ su cui viene applicato il codice convoluzionale 1/2. Il codificatore convoluzionale genera in uscita un flusso continuo di bit $Y(8 * W * 2)$ che viene parallelizzato, su 8 bit, generando la sequenza d'uscita $Z(2 * W)$.

Il convolutore utilizzato (rappresentato in figura 1) è una macchina sequenziale sincrona con un `clock` globale e un segnale di `reset`. Lo stato iniziale è 00 e ogni transizione rappresenta $Uk/P1k, P2k$. I bit $P1k$ e $P2k$ vengono poi concatenati per generare il flusso continuo Yk.

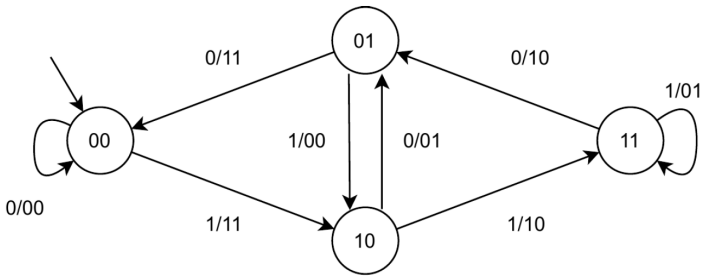

Figura 1: Convolutore

1.3 Interfaccia del componente

Il componente ha la seguente interfaccia:

```
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:

- il nome del modulo è `project_reti_logiche`;
- `i_clk` è il segnale di `CLOCK` in ingresso generato dal TestBench;
- `i_rst` è il segnale di `RESET` che inizializza la macchina, che a questo punto è pronta per ricevere il primo segnale di `START`;

- `i_start` è il segnale di `START` generato dal TestBench;
- `i_data` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_address` è il segnale (vettore) di uscita che rappresenta l'indirizzo della memoria da cui leggere o scrivere;
- `o_done` è il segnale di uscita che comunica la fine dell'elaborazione delle W parole;
- `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 poter scrivere e (=0) per poter leggere.
- `o_data` è il segnale (vettore) di uscita dal componente verso la memoria.

1.4 Descrizione della memoria

Le parole, ciascuna di dimensione 8 bit, sono memorizzate in memoria con indirizzamento al byte. Qui di seguito un esempio di una sequenza formata da 3 parole:

- W: 01110000 10100100 00101101
- Z: 00111001 10110000 11010001 11110111 00001101 00101000

Indirizzo di memoria	Valore binario	Valore decimale	Commento
0	00000011	3	Numero di parole in ingresso
1	01110000	112	Prima parola da codificare
2	10100100	164	Seconda parola da codificare
3	00101101	45	Terza parola da codificare
[...]			
1000	00111001	57	Codifica 1 della prima parola
1001	10110000	176	Codifica 2 della prima parola
1002	11010001	209	Codifica 1 della seconda parola
1003	11110111	247	Codifica 2 della seconda parola
1004	00001101	13	Codifica 1 della terza parola
1005	00101000	40	Codifica 2 della terza parola

Tabella 1: Descrizione della memoria

Nell'indirizzo 0 è memorizzata la quantità di parole W da codificare, le quali sono memorizzate a partire dall'indirizzo 1. Lo stream di uscita Z è memorizzato a partire dall'indirizzo 1000.

2 Architettura

2.1 Scelte implementative

Per l'implementazione del modulo sono state utilizzate due FSM (Finite State Machine) di Mealy, composte rispettivamente da 7 e da 4 stati. La prima macchina a stati finiti (figura 2) è quella principale e in un suo stato (`CONV`), tramite un segnale di `enable` abilita l'altra FSM, ovvero il convolutore (figura 1), il quale inizia la sua codifica convoluzionale.

Per descrivere le due FSM sono stati utilizzati 4 processi; potenzialmente si sarebbero potuti utilizzare 3 processi, inglobando i due processi (`state_reg`) che gestiscono le transizioni tra gli stati, ma si è preferito utilizzarne 4 per non ridurre la leggibilità del codice. I processi sono i seguenti:

- `fsm_state_reg:` processo che effettua la transizione tra gli stati della FSM principale, assegnando il valore del `next_state` al `current_state` a ogni fronte di salita del `clock` e resettando la macchina in caso di attivazione del segnale `i_rst`;
- `fsm_delta_lambda:` processo che gestisce sia la funzione d'uscita, cioè la logica di ogni stato e la manipolazione dei registri, sia la funzione di transizione, ovvero la determinazione dello stato prossimo della FSM in base allo stato corrente;
- `conv_state_reg:` processo che effettua la transizione tra gli stati della macchina convoluzionale, assegnando il valore del `next_state` al `current_state` a ogni fronte di salita del `clock`;
- `conv_delta_lambda:` processo che gestisce la funzione d'uscita e la funzione di transizione della macchina convoluzionale.

2.2 Funzionamento

Il modulo inizierà l'elaborazione quando il segnale `i_start` in ingresso verrà portato alto. L'inizio dell'elaborazione consiste nella lettura dell'indirizzo di memoria 0 in cui è salvato il numero di parole W da leggere e codificare. Successivamente verranno lette tutte le parole singolarmente a partire dall'indirizzo 1; ognuna di esse sarà serializzata, codificata mediante il convolutore e parallelizzata per poter essere mandata in uscita. Nel momento in cui termina la computazione, il segnale `o_done` sarà portato alto e rimarrà in questo stato finché non verrà abbassato il segnale `i_start`. Quando il segnale `o_done` sarà riportato a 0, la FSM terminerà oppure potrà essere dato un nuovo segnale `i_start` per far partire un'altra computazione.

Questo modulo è stato progettato per poter elaborare più di un flusso; per ogni nuovo flusso il convolutore viene riportato nello stato S00, che è il suo stato di reset.

2.3 Segnali e valori di default

Qui di seguito sono riportati i segnali utilizzati e i loro rispettivi valore di default:

Segnale	Valore di default	Commento
`next_state`	START	Stato prossimo della FSM principale
`current_state`	START	Stato corrente della FSM principale
`next_state_conv`	S00	Stato prossimo del convolutore
`current_state_conv`	S00	Stato corrente del convolutore
`conv_en`	0	Segnale di enable per il convolutore
`conv_rst`	0	Segnale di reset per il convolutore
`curr_addr`	0000000000000000	Indirizzo di memoria corrente
`read_addr`	0000000000000000	Indirizzo di lettura
`write_addr`	0000001111101000	Indirizzo di scrittura
`n_words`	00000000	Numero di parole da codificare
`counter`	0000	Contatore per i bit serializzati e codificati
`w`	00000000	Segnale per salvare ogni parola letta dalla memoria
`u`	0	Bit serializzato dato in input al convolutore
`z`	00000000	Output del convolutore

Tabella 2: Segnali e valori di default

2.4 Stati della FSM

La macchina a stati finiti principale (macchina di Mealy) in figura 2 ha 7 stati. Qui di seguito vi è una descrizione di ognuno di essi.

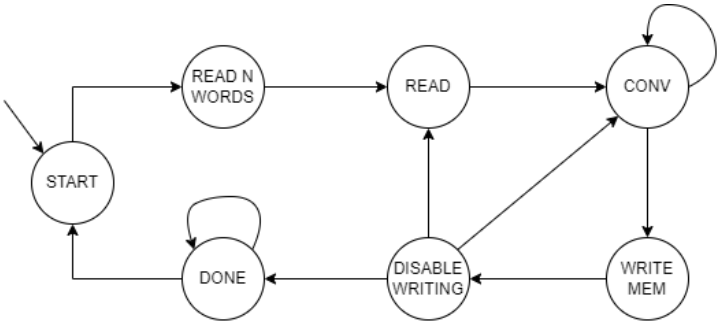

Figura 2: Macchina a stati finiti principale

2.4.1 START

Stato iniziale della FSM principale. In questo stato vengono resettati tutti i segnali. Quando il segnale in ingresso `i_start` viene portato alto, viene attivata la comunicazione con la memoria alzando il segnale `o_en` e viene settato il `next_state` a `READ_N_WORDS`. Questo stato è raggiungibile da ogni altro in caso di ricezione del segnale `i_rst`.

2.4.2 READ_N_WORDS

Stato in cui viene letto l'indirizzo 0 della memoria, viene incrementato il `curr_addr` e viene settato il `next_state` a `READ`.

2.4.3 READ

Il compito di questo stato è leggere le parole in ingresso dalla memoria; se il numero di parole è uguale a 0, il `next_state` viene settato direttamente a `DONE`.

2.4.4 CONV

In questo stato viene alzato il segnale di `enable` del convolutore, abilitandolo, e viene serializzata ogni parola. Grazie al segnale `counter`, che è un contatore, viene tenuta traccia dei bit serializzati e ogni volta che 4 bit vengono codificati dal convolutore la FSM va nello stato `WRITE_MEM`.

2.4.5 WRITE_MEM

Stato in cui viene scritto l'output in memoria a partire dall'indirizzo 1000; viene alzato il segnale `o_we` per poter scrivere in memoria e viene salvato in `write_addr` l'indirizzo di memoria in cui si dovrà scrivere la parola successiva.

2.4.6 DISABLE_WRITING

Questo stato serve per disabilitare la scrittura e per controllare se ci sono ancora parole da codificare.

2.4.7 DONE

Ultimo stato della macchina a stati finiti principale. In questo stato viene alzato il segnale `o_done` e viene interrotta la comunicazione con la memoria. Quando viene abbassato il segnale `i_start`, il convolutore viene resettato e la FSM viene portata nello stato `START`, che la resetta e la rende pronta a una nuova computazione nel caso in cui venisse rialzato il segnale `i_start`.

3 Risultati sperimentali

Per l'implementazione e la sintesi è stata utilizzata la seguente FPGA: Artix-7 xc7a200tffg1156-1, prodotto dall'azienda Xilinx.

3.1 Sintesi

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	95	0	134600	0.07
LUT as Logic	95	0	134600	0.07
LUT as Memory	0	0	46200	0.00
Slice Registers	100	0	269200	0.04
Register as Flip Flop	100	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

Figura 3: Utilization Report: Slice Logic

Come si può vedere in figura 3, è stata utilizzata solamente una piccolissima parte dell'FPGA: lo 0.07% di LUT as Logic e lo 0.04% di Flip Flop. Il codice è stato scritto in modo da evitare l'inferenza di Latch, sincronizzando il componente sul fronte di salita e di discesa del clock.

3.2 Simulazioni

Il modulo è stato sottoposto ai vari test benches forniti insieme alla specifica e sviluppati per testare i casi limite e il funzionamento dei segnali. Il componente passa con successo le simulazioni: Behavioural, Post-Synthesis Functional e Post-Synthesis Timing. Qui di seguito sono riportati i risultati ottenuti nei vari test benches.

3.2.1 Funzionamento dei segnali

1. **Reset Asincrono**

   - Behavioural Simulation: 11100 ns
   - Post-Synthesis Functional Simulation: 11200,1 ns
   - Post-Synthesis Timing Simulation: 11204,373 ns

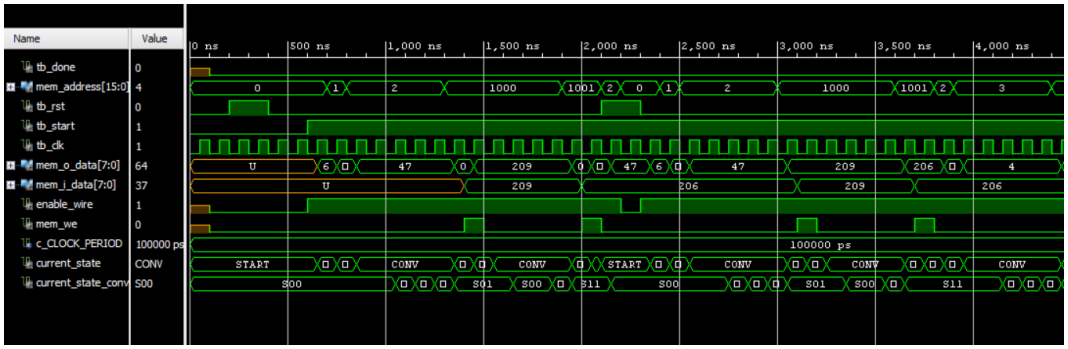

Figura 4: Simulazione con segnale di reset asincrono

Come si vede in figura 4, all'istante 2100 ns viene alzato il segnale di reset asincrono che porta la FSM principale nello stato di `START` e il convolutore nello stato `S00`.

2. **Computazioni Multiple**

   - Behavioural Simulation: 23000 ns
   - Post-Synthesis Functional Simulation: 23500,1 ns
   - Post-Synthesis Timing Simulation: 23504,373 ns

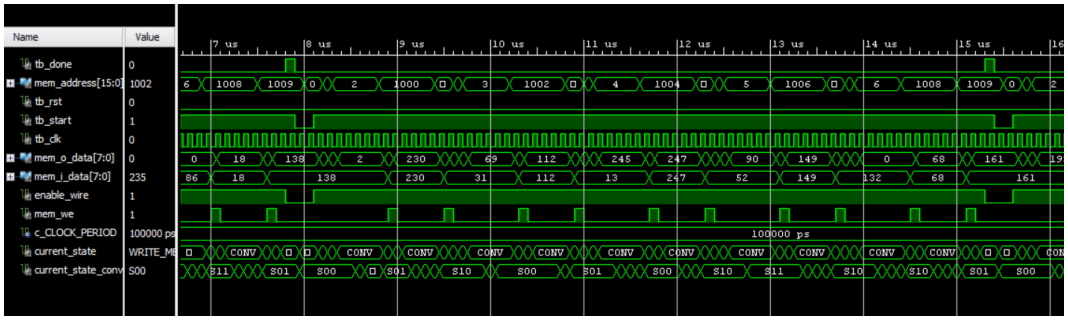

Figura 5: Simulazione con più flussi uno dopo l'altro

Come si vede in figura 5, all'istante 7800 ns viene terminata la computazione del primo flusso e il segnale `o_done` viene portato alto. Dopo 100 ns viene abbassato il segnale `i_start`. All'istante 7950 ns, sul fronte di salita del clock, la FSM principale viene portata nello stato `START` e il convolutore in `S00`.

3. **Doppia computazione sullo stesso flusso**

   - Behavioural Simulation: 9600 ns
   - Post-Synthesis Functional Simulation: 9900,1 ns
   - Post-Synthesis Timing Simulation: 9904,373 ns

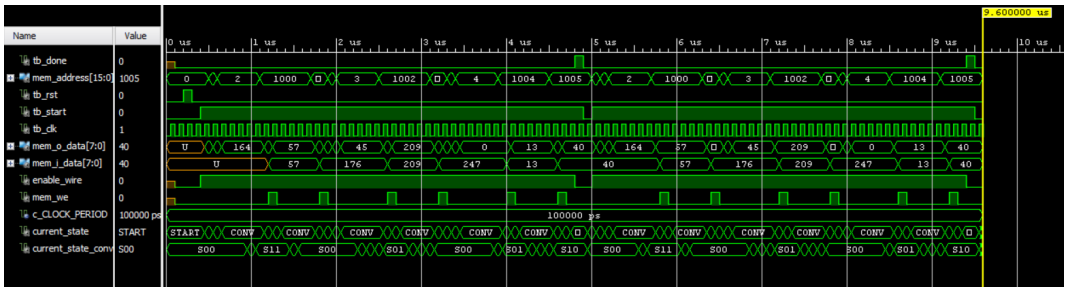

Figura 6: Simulazione con la codifica dello stesso flusso due volte

3.2.2 Casi limite

1. **Sequenza di parole nulla**

   - Behavioural Simulation: 900 ns
   - Post-Synthesis Functional Simulation: 1000,1 ns
   - Post-Synthesis Timing Simulation: 1004,373 ns

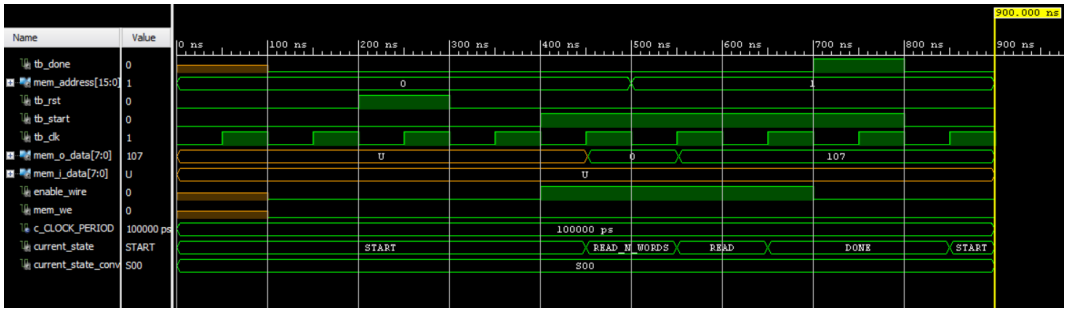

Figura 7: Simulazione con zero parole in ingresso

In figura 7 abbiamo 0 parole in input. All'istante 450 ns viene letto il numero di parole (0) e all'istante 650ns la FSM principale viene portata direttamente nello stato `DONE`.

2. **Sequenza di lunghezza massima**

   - Behavioural Simulation: 357800 ns
   - Post-Synthesis Functional Simulation: 357900,1 ns
   - Post-Synthesis Timing Simulation: 357904,373 ns

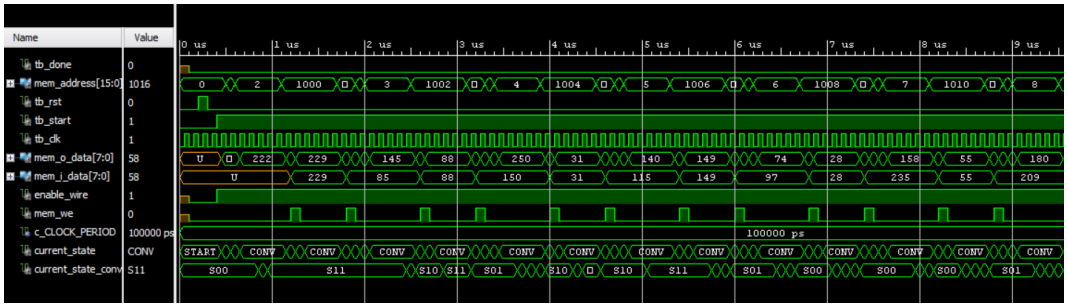

Figura 8: Simulazione con 255 parole in ingresso

3.2.3 Test casuali

Per verificare il corretto funzionamento del componente in modo più approfondito, si è deciso di sottoporlo a numerosi test casuali generati mediante un applicativo scritto in python. Questi ultimi sono stati passati tutti con successo.

4 Conclusioni

Il componente implementato e successivamente sintetizzato rispetta le specifiche proposte, rientrando largamente nel massimo tempo di clock consentito. Esso è in grado di leggere da 0 a 255 parole in input, di elaborlarle con un codice convoluzionale 1/2, e di scrivere in output fino a 510 parole.

Quindi si può affermare che il componente prodotto è in grado di soddisfare le specifiche richieste.