Progetto di Reti Logiche
Anno 2021/2022

Michael Fiano - Matricola 937942 -
Federico Lamperti - Matricola 935353 -

29 Aprile 2022

Indice

Capitolo 1

Introduzione

1.1 Scopo del progetto

Sintetizzare in linguaggio VHDL, un modulo che ricevuta una sequenza di byte da una memoria RAM, restituisce una sequenza di byte, ognuna ottenuta a seguito di una rielaborazione tramite un codice convoluzionale. Questa sequenza opportunamente elaborata, viene poi memorizzata nella stessa memoria RAM, negli indirizzi a partire dal millesimo.

1.2 Interfaccia Componente

```
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 pronta per ricevere il primo segnale di START;

- i_start è il segnale di START generato dal Test Bench;
- 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 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 poter scriverci. Per leggere da memoria esso deve essere 0;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria

1.3 Elaborazione dei dati

Il modulo riceve una sequenza continua di parole dalla memoria che verranno trattate una alla volta. Ogni parola viene nuovamente suddivisa in modo da creare un flusso di bit singoli. Questi vengono sottoposti ad una rielaborazione tramite codice convoluzionale 1/2 (1 bit in entrata corrisponde a 2 in uscita).

Viene prodotta quindi una sequenza di 16 bit, ottenuta concatenando le uscite del convolutore, che viene poi divisa a metà e viene memorizzata in due indirizzi adiacenti.

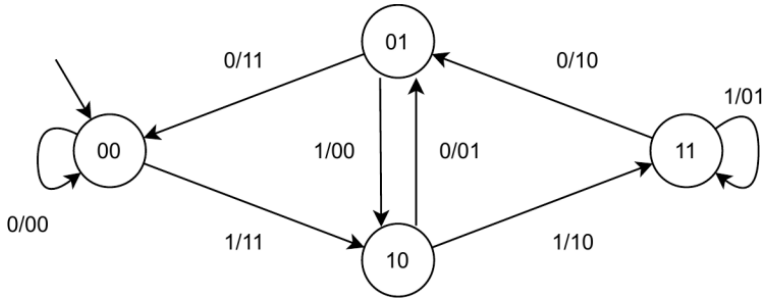

Figura 1.1: Convolutore (Macchina a stati finiti)

1.4 Gestione della memoria

I dati, rappresentati in parole da 1 byte (8 bit), sono allocati in memoria nella seguente maniera:

- l'indirizzo 0 indica il numero di parole effettivamente presenti in memoria e che vanno in seguito processate;
- dall'indirizzo 1 al 255 sono contenute le parole da elaborare e trasformare;
- Dall'indirizzo 1000 in poi sono presenti gli indirizzi dedicati alla scrittura del flusso di parole generato dal modulo.

Numero parole da processare	Indirizzo 0
Prima parola da elaborare	Indirizzo 1
...	
Ultimo indirizzo valido di lettura	indirizzo 255
...	
Primo indirizzo di scrittura	Indrizzo 1000
...	
Ultimo indirizzo valido di scrittura	Indirizzo 1509

Tabella 1.1: Rappresentazione indirizzi di memoria

1.5 Esempio di Test

1.5.1 Ingressi e Uscite

Sequenza di lunghezza 2:
W: 01111000 00001011
Z: 00111001 01101100 00000000 11010010

Indirizzo memoria	Valore
0	2
1	120
2	11
...	...
1000	57
1001	108
1002	0
1003	210

1.5.2 Passi algoritmo convoluzionale

Valutando la parola in ingresso: 01111000

t	0	1	2	3	4	5	6	7
U_k	0	1	1	1	1	0	0	0
P_{k1}	0	1	1	0	0	1	1	0
P_{k2}	0	1	0	1	1	0	1	0

Il concatenamento dei valori P_{k1} e P_{k2} per produrre Z segue il seguente schema: P_{k1} al tempo t, P_{k2} al tempo t, P_{k1} al tempo t+1 P_{k2} al tempo t+1, P_{k1} al tempo t+2 P_{k2} al tempo t+2, e così fino a t+7 ottenendo Z: 0011100101101100

Capitolo 2

Architettura

La nostra scelta progettuale è stata quella di procedere implementando un automa a stati finiti mediante l'utilizzo di stati con compiti il più possibile specifici. Il modulo funziona in modo sincrono grazie all'utilizzo di un clock. Per utilizzare il modulo occorre un segnale iniziale di reset in ingresso. Successivamente, tramite un segnale in ingresso di start, è possibile avviare il processo. Questo segnale viene mantenuto alto fino alla fine dell'elaborazione. Il modulo, completato il trattamento dei dati, provvede ad "alzare" il segnale di terminazione che rimane attivo finché il segnale di start rimane alto. Una volta che il segnale di start si pone a 0 e che anche il segnale di terminazione si pone a 0, il processo termina. Il segnale di reset può essere portato ad 1 anche prima che il modulo termini i suoi compiti. Ciò comporta un reset sia dello stato della macchina che di ogni segnale e variabile.

2.1 Macchina a Stati

La nostra architettura prevede quindi una moltitudine di stati che sono in seguito descritti (Figura 2.1).

2.1.1 IDLE

Stato iniziale della macchina, nonché lo stato di reset. Qui infatti si arriva in seguito ad ogni segnale di reset ricevuto in ingresso, compreso il primo, e si attende che il segnale i_start venga alzato. In questo stato ogni segnale e ogni variabile viene posta al valore di default.

2.1.2 MEM_REQUEST

Stato in cui vengono richiesti alla memoria i dati da leggere, aggiornando il segnale di enable e l'indirizzo di lettura.

2.1.3 WAIT_MEM

Stato in cui si attende il dato in arrivo dalla memoria.

2.1.4 READ_MEM

Stato in cui si leggono le parole in ingresso dalla memoria. Alla lettura del primo indirizzo di memoria si salva il numero di parole su cui applicare il ciclo convoluzionale, qualora questo numero fosse 0 l'elaborazione termina.

2.1.5 SERIALIZER

Stato in cui si disabilita la possibilità di lettura in memoria e in cui ogni parola letta viene trattata come un vettore. Ogni cella del vettore viene trasmessa al convolutore una alla volta.

2.1.6 CONVOLVER

Stato in cui ogni bit in ingresso viene trasformato in due bit attraverso un algoritmo convoluzionale. Questo stato è organizzato a sua volta come un'automa a stati finiti.

2.1.7 PARALLELIZER

Stato in cui i bit elaborati dal convolutore vengono salvati in una vettore di lunghezza di 16 bit, uno in seguito all'altro.

2.1.8 MEM_WRITE_REQUEST

Stato di richiesta di scrittura in memoria, in cui si alzano i segnali di enable e write-enable e si aggiorna l'indirizzo di salvataggio.

2.1.9 WRITE_MEM

Stato di scrittura in due blocchi contigui di memoria del vettore precedentemente salvato.

2.1.10 CHECK

Stato in cui si abbassano i segnali di enable e di write-enable e, successivamente, si valuta se ogni parola da leggere sia stata effettivamente elaborata. Nel caso in cui la macchina non abbia svolto il suo compito fino alla fine, si ritorna allo stato di lettura della memoria, in caso contrario si procede.

2.1.11 DONE

Stato in cui si alza il segnale o_done e si attende che il segnale i_start venga portato a 0. In seguito si pone a 0 anche o_done, questo indica la fine di tutta l'elaborazione eseguita dal modulo tornando allo stato IDLE.

2.2 Scelte Progettuali

Abbiamo pensato che fosse più chiaro e più comodo descrivere il comportamento del modulo utilizzando un solo Process che valuta costantemente il valore del segnale di reset e, ad ogni fronte di risalita del Clock, aggiorna lo stato ed i segnali. Abbiamo inoltre scelto di utilizzare meno segnali possibili in favore di variabili il cui utilizzo rende il modulo più efficiente e rapido nello svolgere il suo compito. Per la gestione delle celle dei vettori sono stati usati dei segnali che fungono da contatore, infatti essi vengono decrementati opportunamente quando si passa alla cella successiva.

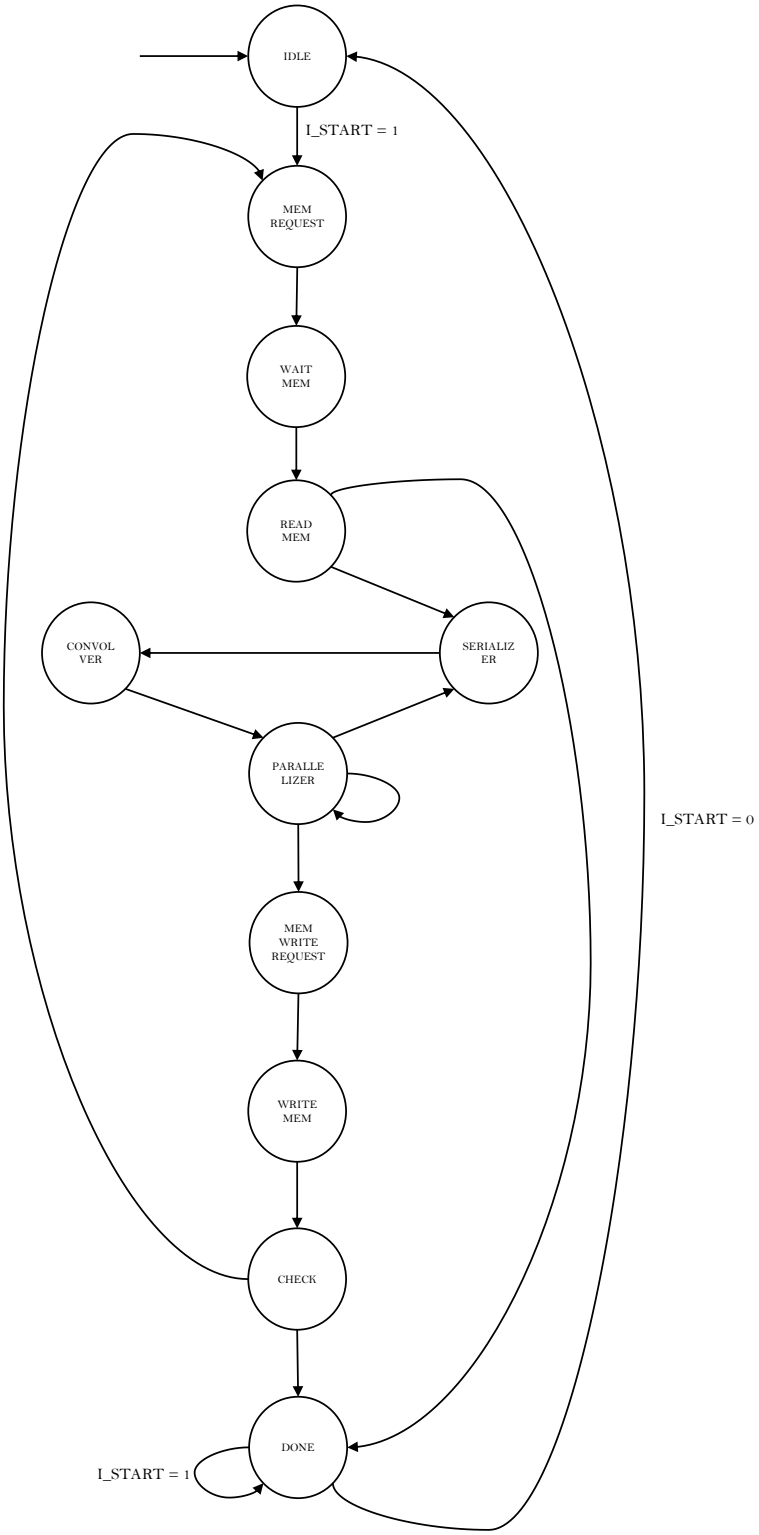

Figura 2.1: Macchina a stati finiti

Capitolo 3

Risultati sperimentali

3.1 Sintesi

Lo schematico della sintesi in Figura 3.1 rappresenta l'FPGA fisico attraverso 638 Cells, 38 Ports I/O e 872 Nets.

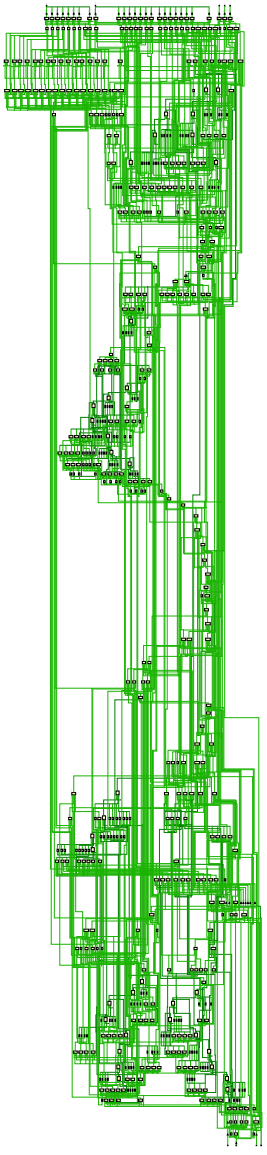

Figura 3.1: schematico della sintesi

3.2 Simulazioni

Per valutare la correttezza del nostro codice ci sono stati forniti svariati test ognuno con uno scopo differente, di alcuni ne forniamo una breve spiegazione a seguito.

3.2.1 seq_min

La prima parola da leggere in memoria è 0. Lo scopo del test è quello di verificare che il modulo, una volta letto il primo indirizzo, termini immediamente senza fare ulteriori operazioni.

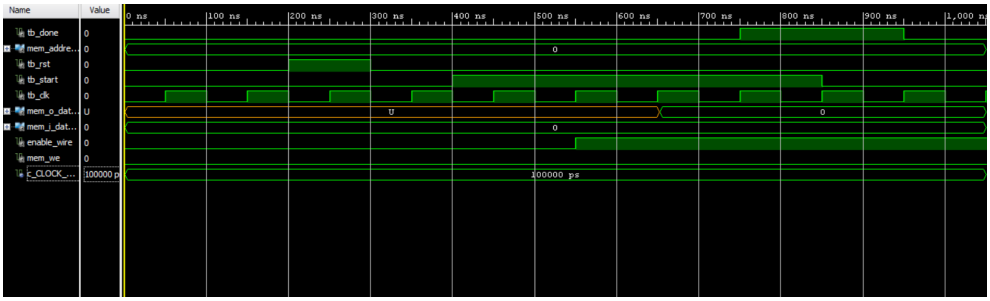

Figura 3.2: Segnali simulazione seq_min

3.2.2 esempio_Test

Test che valuta un processo con due letture. È un test modificato da noi, seguendo il codice del testBench esempio_1, ma modificando i dati in ingresso e di conseguenza quelli in uscita. Questo è il test che abbiamo svolto più volte a causa della sua generalità, poiché, gestione del reset asincrono esclusa, testa il processo nella sua interezza. Grazie a questo test abbiamo riscritto più volte il codice arrivando poi alla versione finale.

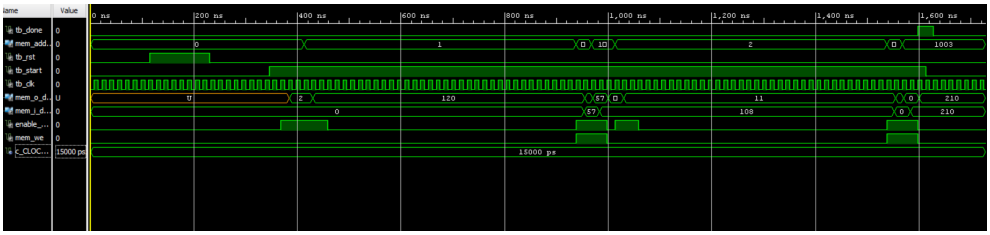

Figura 3.3: Segnali simulazione esempio_Test

3.2.3 reset

Lo scopo del test è quello di valutare l'effetto di un segnale di reset asincrono. Infatti una volta inviato il segnale di reset, il processo deve ricominciare, il modulo deve ritornare nello stato di IDLE e ciò non deve compromettere la successiva elaborazione.

3.2.4 seq_max

Test con 255 indirizzi in lettura da processare (lunghezza massima ammissibile). Questo test è stato anche utile per verificare la corretta gestione dei segnali di start, reset e terminazione.

3.2.5 tre_bis

Test il cui scopo è valutare il funzionamento dell'elaborazione di 3 flussi differenti uno dopo l'altro.

Capitolo 4

Conclusioni

4.1 Ottimizzazioni

Tramite i test-bench ci siamo resi conto di aver posto degli stati di attesa superflui che rallentavano l'uscita del modulo ed una volta trovati li abbiamo eliminati. Inoltre abbiamo cercato di minimizzare il numero di variabili e segnali in modo da ridurre il più possibile l'utilizzo della memoria.

4.2 Considerazioni personali

È stato molto interessante poter lavorare con un programma ed un linguaggio di programmazione mai usati prima. Abbiamo entrambi compreso meglio l'interazione del componente con una memoria esterna di tipo RAM, l'utilizzo dei segnali interni e soprattutto come affrontare un problema complesso in un mondo, come quello dei linguaggi di descrizione hardware, a noi quasi del tutto estraneo. Fondamentali sono stati il confronto tra colleghi e l'integrazione di idee spesso differenti.