Relazione della Prova Finale di Reti Logiche

Tommaso Bonetti

1 settembre 2022

Indice

1 Introduzione **2**

2 Architettura **3**
2.1 Macchina a stati finiti 3
2.2 Convolutore 5
2.3 Datapath 6

3 Risultati sperimentali **8**
3.1 Sintesi 8
3.2 Simulazioni 8
3.2.1 Comportamento standard 8
3.2.2 Corner case 8

4 Conclusioni **9**

1 Introduzione

Il progetto di prova finale consiste nell'implementazione in VHDL di un modulo hardware che, ricevuta in ingresso una sequenza U di parole di 8 bit, genera in uscita un'altra sequenza Y di parole di 8 bit, generate tramite il codificatore convoluzionale rappresentato in figura.

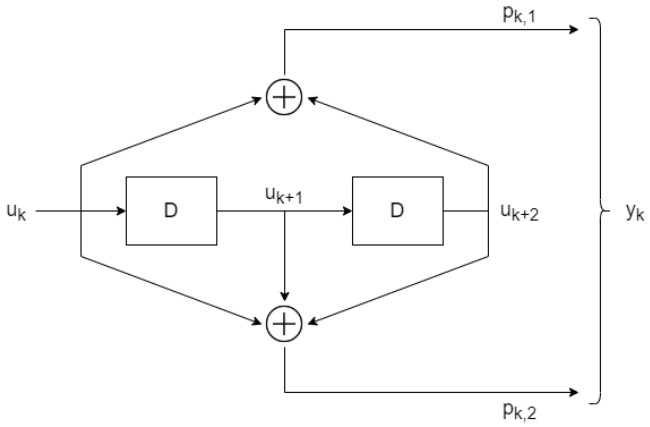

Figura 1: il codificatore convoluzionale da implementare.

Si può osservare che il convolutore ha tasso di trasmissione $\frac{1}{2}$, ossia ogni bit di input viene codificato con 2 bit di output: ne consegue che il numero di parole in output Z sarà il doppio del numero di parole in input W. Inoltre, il codificatore non elabora parole intere, bensì un flusso di bit serializzati u_k, generando un flusso di output $y_k = p_{k,1} \| p_{k,2}$ che viene concatenato per ottenere le parole da 8 bit che vengono poi scritte in memoria.

È importante notare come l'elaborazione in sequenza dei bit che compongono le parole dell'input fa sì che l'output della codifica di una singola parola sia, in generale, variabile, in quanto dipenderà anche dalle parole che sono state codificate prima di essa.

A partire da queste premesse è quindi possibile rappresentare il convolutore come una macchina sequenziale sincrona di Mealy, in cui l'output (2 bit) dipende sia dallo stato che dall'input (1 bit). La macchina risultante ha 4 stati e si presenta come nella figura sotto.

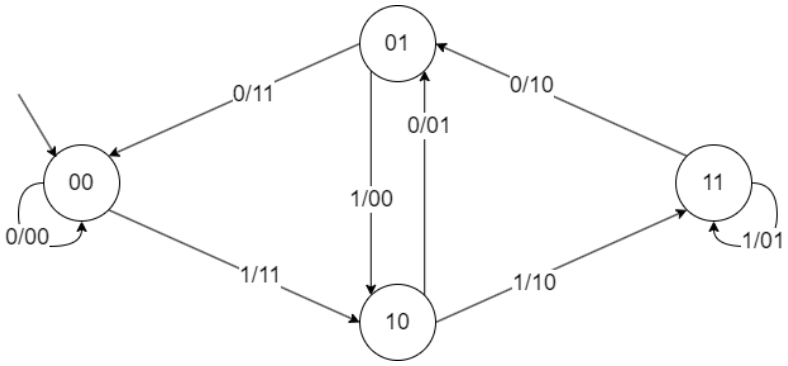

Figura 2: il convolutore come macchina a stati finiti di Mealy.

Esempio di funzionamento Sia $U = 00100101$, dove il primo bit codificato sarà il Most Significant Bit (in questo caso 0). Allora, a partire dall'istante $t = 0$ e ricordando l'ordine di concatenazione $y_k = p_{k,1} \| p_{k,2}$, il flusso y_k sarà quello illustrato in tabella.

t	0	1	2	3	4	5	6	7
u_k	0	0	1	0	0	1	0	1
$p_{1,k}$	0	0	1	0	1	1	0	0
$p_{2,k}$	0	0	1	1	1	1	1	0
y_k	00	00	11	01	11	11	01	00

Tabella 1: i valori di input e di output per ogni istante $0 \leq t \leq 7$.

Concatenando i bit di y_k si ottiene dunque la sequenza $Y = 00001101\ 11110100$.

2 Architettura

Il progetto è implementato utilizzando tre componenti: una macchina a stati finiti, il codificatore convoluzionale e il datapath.

2.1 Macchina a stati finiti

Si tratta del componente principale, che si interfaccia con il test bench e la memoria gestendo l'elaborazione delle sequenze di parole e occupandosi anche di coordinare gli altri due componenti e gestire la serializzazione e deserializzazione di input e output. Nel codice VHDL è rappresentato dall'entity `project_reti_logiche`.

Porte Come già detto in precedenza questo componente è l'interfaccia verso il test bench, motivo per cui le porte sono quelle definite nelle specifiche. Tra esse si segnalano:

- il clock `i_clk`;
- i segnali di start e reset `i_start` e `i_rst`;
- i vettori di input e output in memoria, rispettivamente `i_data` di 8 bit e `o_data` di 16 bit;
- i segnali di enable e write enable della memoria `o_en` e `o_we`;
- il segnale di fine elaborazione `o_done`.

Architettura L'architettura si interfaccia con i due component `convolutor` e `datapath` e descrive due processi, uno sincronizzato e uno combinatorio.

Il primo, sensibile ai soli segnali di clock e reset, è preposto ad aggiornare lo stato della macchina sui fronti di salita del clock e a deserializzare l'output. Si noti che il vettore `o_data` viene scritto due bit alla volta, per cui non sarebbe possibile assegnare a esso un valore di default in un processo combinatorio senza sovrascrivere i dati già codificati: deserializzare in un processo combinatorio causerebbe quindi l'inferenza di latch portando ad effetti indesiderati, specialmente in post sintesi. Per scongiurare questo rischio risulta quindi necessario effettuare la deserializzazione in un processo sincronizzato.

Il secondo processo, che viene invocato dopo l'aggiornamento dello stato della macchina, consiste invece nel serializzare l'input da passare al convolutore e nell'aggiornare lo stato prossimo e i segnali necessari per comunicare con il test bench e con gli altri moduli.

Tra i segnali si evidenziano:

- `curr_state` e `next_state`, che rappresentano lo stato attuale e lo stato prossimo della macchina;
- `dp_rst` e `conv_rst`, che danno il segnale di reset rispettivamente a datapath e convolutore;
- `dp_done`, che viene alzato a 1 dal datapath quando l'elaborazione è terminata e segnala alla macchina a stati che il segnale `o_done` deve essere alzato.

Il significato degli altri segnali verrà spiegato in dettaglio nelle descrizioni degli altri moduli.

Stati La macchina che implementa il modulo ha 17 stati, da $s0$ a $s16$, così organizzati:

- lo stato $s0$ è lo stato iniziale, a cui la macchina viene riportata a ogni reset e su cui rimane finché il segnale `i_start` si alza, indicando l'inizio di una nuova sequenza di parole da codificare;
- lo stato $s1$ legge il numero di parole nella sequenza U dalla prima cella di memoria e porta la macchina nel ciclo di codifica di una singola parola;

- gli stati $s2$ e $s3$ leggono la successiva parola da codificare;
- gli stati da $s5$ a $s12$ eseguono la serializzazione dell'input e la deserializzazione dell'output ricevuto dal convolutore;
- gli stati da $s13$ a $s15$ scrivono in memoria le due parole di output corrispondenti alla codifica dell'input attuale;
- lo stato $s16$ riporta la macchina in $s0$ se la sequenza di parole da codificare è terminata, altrimenti la porta in $s2$ e ricomincia il ciclo di codifica.

I segnali aggiornati da ogni stato sono illustrati in figura.

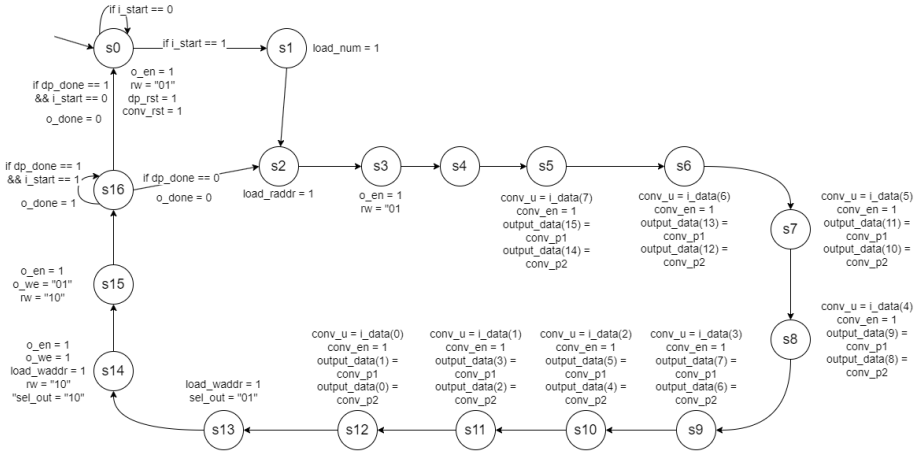

Figura 3: la macchina a stati finiti e i segnali aggiornati da ogni stato.

2.2 Convolutore

Questo componente implementa il codificatore convoluzionale, che riceve l'input serializzato e, per ogni bit, produce due bit di output. Nel codice VHDL è rappresentato dall'entity `conv`.

Porte Ricordando che il convolutore è una macchina sequenziale sincrona, tra le porte si trovano:

- il clock `i_clk`;
- il segnale di reset `i_rst`, che riporta il convolutore allo stato iniziale;
- il segnale di enable `i_en`, che permette al convolutore di cambiare stato solo quando è alto;
- l'ingresso `i_u` e le uscite `o_p1` e `o_p2`.

Architettura L'architettura, alla stregua di quella della macchina a stati di `project_reti_logiche`, descrive un processo sincronizzato e uno combinatorio.

Il primo, sensibile ai soli segnali di clock e reset, aggiorna lo stato del convolutore sui fronti di salita del clock quando il segnale `i_en` è alto e riporta il convolutore nello stato iniziale quando `i_rst` è alto.

Il secondo processo, invocato dopo l'aggiornamento dello stato del convolutore, aggiorna lo stato prossimo e le uscite. Si nota che il codificatore è una macchina a stati finiti di Mealy, quindi il valore delle uscite dipende sia dallo stato attuale che dall'ingresso.

La macchina ha quattro stati, da $s0$ a $s3$, e il suo funzionamento è illustrato in figura 2 (v. pagina 3).

2.3 Datapath

Questo componente ha il compito di calcolare i corretti indirizzi di memoria per la lettura e scrittura dei dati e di indicare quando tutte le parole della sequenza di input U sono state codificate e l'elaborazione è terminata. Nel codice VHDL è rappresentato dall'entity `dp`.

Porte Le porte del modulo comprendono:

- i segnali di clock e reset `i_clk` e `i_rst`;
- i vettori della parola letta dalla memoria, `i_data`, e della parola da scrivere in memoria, `o_data`, entrambi da 8 bit;
- il vettore da 16 bit `data_long`, contenente l'output da scrivere in memoria;
- il vettore da 16 bit `o_address`, che indica l'indirizzo da cui leggere o su cui scrivere in memoria;
- il segnale `o_done`, che indica che tutte le parole sono state codificate e l'elaborazione è terminata;
- i segnali `load_raddr` e `load_waddr`, che quando sono alti indicano, rispettivamente, di caricare l'indirizzo di lettura o quello di scrittura sul vettore `o_address`;
- il vettore `sel_out`, che indica quale delle due parole dell'output `data_long` deve essere scritta su `o_data`: quando vale 00 non c'è output, quando vale 01 viene scritto il primo byte e quando vale 10 viene scritto il secondo;
- il vettore `rw`, che indica il tipo di accesso alla memoria: quando vale 00 non c'è accesso, quando vale 01 c'è accesso in lettura e quando vale 10 c'è accesso in scrittura.

Architettura L'architettura fa uso di una serie di segnali, che rappresentano diversi dati di interesse. Alcuni vengono aggiornati con processi sincronizzati

sul fronte di salita del clock, mentre altri sono costantemente aggiornati; il significato di questi ultimi è spiegato in maggior dettaglio di seguito:

- `sum_r`, vettore di 16 bit, indica il successivo indirizzo di memoria da cui leggere;
- `mux_data`, vettore di 8 bit, contiene la successiva parola da scrivere in memoria[1];
- `mux_waddr`, vettore di 16 bit, indica il successivo indirizzo di memoria su cui scrivere[2];
- `sub`, vettore di 16 bit, indica il numero di parole rimaste da codificare.

I segnali che vengono aggiornati con processi sincronizzati sono invece i seguenti:

- `read_addr`, che indica l'indirizzo di memoria da cui leggere;
- `write_addr`, che indica l'indirizzo di memoria su cui scrivere;
- `num_words`, che indica il numero complessivo di parole da codificare.

Infine, in base ai valori dei segnali interni e a quelli dei segnali in input ricevuti dal modulo `project_reti_logiche`, vengono aggiornati i seguenti segnali in output:

- `o_data`, che vale 0 se `sel_out` vale 00 e contiene la parola da scrivere in memoria se `sel_out` vale 01 o 10;
- `o_address`, che vale 0 se `rw` vale 00, contiene l'indirizzo di lettura se `rw` vale 01 e contiene l'indirizzo di scrittura se `rw` vale 10;
- `o_done`, che vale 1 solo se non ci sono più parole da codificare, 0 altrimenti.

[1] Si ricorda che, per ogni parola in input, l'output prodotto dal convolutore ha dimensione 16 bit e deve quindi essere diviso in due parole di memoria.

[2] Per ogni parola codificata è necessario scrivere due parole in due indirizzi consecutivi, motivo per cui si utilizza questo multiplexer la cui uscita corrisponde a uno tra i segnali `sum_w1` e `sum_w2`.

3 Risultati sperimentali

3.1 Sintesi

Report utilization Di seguito si riporta la tabella Slice Logic. In particolare è possibile notare come tutti i registri utilizzati per sintetizzare il modulo siano flip flop.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	109	0	0	134600	0.08
LUT as Logic	109	0	0	134600	0.08
LUT as Memory	0	0	0	46200	0.00
Slice Registers	83	0	0	269200	0.03
Register as Flip Flop	83	0	0	269200	0.03
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Tabella 2: la tabella Slice Logic.

Report timing Il report sul timing evidenzia uno slack di 94.847 ns rispetto a un clock con periodo 100 ns.

3.2 Simulazioni

3.2.1 Comportamento standard

Test bench default Questo test bench contiene due parole da codificare e ha lo scopo di verificare la corretta funzionalità del modulo. Questo significa che il modulo deve codificare le parole come previsto dalla definizione del convolutore, scrivendo i risultati nelle corrette celle di memoria, e gestire il segnale `o_done` come richiesto dalla specifica, alzandolo a 1 a fine elaborazione e abbassandolo a 0 quando il segnale `i_start` in ingresso dal test bench viene anch'esso abbassato.

Test bench 1 – funzionalità standard Questo test bench contiene sei parole da codificare e ha lo scopo di verificare la corretta funzionalità del modulo, alla stregua del precedente.

Test bench 2 – funzionalità standard Questo test bench contiene tre parole da codificare e ha lo scopo di verificare la corretta funzionalità del modulo, alla stregua dei precedenti.

3.2.2 Corner case

Test bench 3 – sequenza minima Questo test bench contiene zero parole da codificare: ha lo scopo di verificare la capacità del modulo di rispondere correttamente a questo input, terminando immediatamente l'elaborazione.

Test bench 4 – sequenza massima Questo test bench contiene 255 parole da codificare: ha lo scopo di verificare la capacità del modulo di rispondere correttamente a questo input, codificando correttamente ogni parola della sequenza.

Test bench 5 – doppia elaborazione Questo test bench contiene tre parole da codificare per due volte consecutive: ha lo scopo di verificare la capacità del modulo di elaborare più sequenze di parole senza ricevere il segnale di reset tra esse.

Test bench 6 – elaborazione multipla Questo test bench contiene tre sequenze di cinque parole l'una da codificare consecutivamente: ha lo scopo di verificare la capacità del modulo di elaborare più sequenze di parole senza ricevere il segnale di reset tra esse.

Test bench 7 – elaborazione multipla Questo test bench contiene quattro sequenze di tre parole l'una da codificare consecutivamente: ha lo scopo di verificare la capacità del modulo di elaborare più sequenze di parole senza ricevere il segnale di reset tra esse.

Test bench 8 – reset asincrono Questo test bench contiene sei parole da codificare e invia un segnale di reset asincrono prima dell'inizio dell'elaborazione: ha lo scopo di verificare la capacità del modulo di rispondere correttamente a questo stimolo, iniziando e completando correttamente l'elaborazione.

4 Conclusioni

Il lavoro svolto ha permesso di realizzare un modulo hardware descritto in VHDL in grado di codificare una sequenza di parole di 8 bit secondo il codice convoluzionale definito nelle specifiche.

Il prodotto finale è un modulo diviso in tre componenti: la macchina sequenziale sincrona, che gestisce la serializzazione e deserializzazione di input e output; il convolutore, che codifica il flusso serializzato di bit in input; infine, il datapath, che gestisce l'accesso alla memoria e monitora lo stato di avanzamento dell'elaborazione.

Nell'implementazione è stata posta particolare attenzione all'inferenza di latch da parte del tool di sintesi di Vivado: per evitare questo comportamento la deserializzazione dell'output del convolutore viene eseguita in un processo sincrono.

Il modulo è stato validato con diversi test bench, che mirano a verificarne la funzionalità in condizioni di utilizzo standard e a controllare che il suo comportamento rimanga coerente con le specifiche anche nei possibili corner case. Tutti i test bench svolti sono stati superati con successo, sia nelle simulazioni behavioral che nelle simulazioni funzionali in post sintesi.