Prova Finale di Reti Logiche

Bonfanti Stefano

Anno 2021-22

Matricola:	935228
Codice Persona:	10670135
Docente:	Fornaciari Wiliam
Esercitatore:	Terraneo Federico

Contents

1 Introduzione

1.1 Scopo del progetto

Lo scopo del progetto è l'implementazione in VHDL di un modulo che realizza un codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$. I codificatori convoluzionali trovano una reale applicazione nell'ambito delle telecomunicazioni. La loro funzionalità è quella di trasformare una sequenza di bit fornita in ingresso, producendone una più lunga (nel nostro caso si ha rateo $\frac{1}{2}$, cioè per ogni bit in ingresso ne vengono prodotti due in uscita), ma rendendola correggibile in caso di eventuali errori. Il mittente di un messaggio fornirà i bit da inviare al codificatore che produrrà una nuova sequenza che verrà trasmessa al destinatario, il quale potrà verificare la presenza di errori, eventualmente correggerli ed infine decodificarlo, usando per esempio l'algoritmo di Viterbi.

1.2 Descrizione del modulo

Il modulo dovrà ripetere le stesse operazioni per ogni flusso di parole in ingresso:

1. Interfacciarsi con una memoria RAM dalla quale recupererà all'indirizzo 0 il numero totale di parole a cui applicare il codice convoluzionale.
2. Prelevare dalla memoria la parola all'indirizzo corrente, che verrà incrementato dopo ogni lettura fino ad arrivare al numero ricavato al punto precedente.
3. Serializzare la parola appena prelevata e fornire i singoli bit in ordine al codificatore convoluzionale, il quale restituirà una sequenza di bit da de-serializzare che mi formeranno due parole.
4. Scrivere in memoria, a partire dall'indirizzo 1000, le due parole ottenute in celle successive.
5. Se non si è ancora codificato tutte le parole bisognerà ripetere i passaggi 2, 3 e 4, una volta terminato bisognerà segnalare alla memoria l'avvenuta conclusione, innalzando il segnale *o_ done*.

Durante un'esecuzione se vi sono più flussi consecutivi quando si imposterà *o_ done* a 1 bisognerà anche resettare la macchina a stati del convolutore innalzando il segnale *rst* che la riporterà nello stato di partenza.

1.3 Memoria

La memoria ha un funzionamento sincrono. Per questo in lettura sarà necessario porre il segnale *o_ en* a 1 e indicare la cella di memoria da prelevare in *o_ address* il ciclo di clock precedente rispetto al ciclo in cui potremo leggere in *i_ data* il valore corrispondente. Invece, per la scrittura basterà innalzare i segnali di *o_ en* e *o_ we*, indicare il valore *o_ data* da scrivere all'indirizzo *o_ address* e nello stesso ciclo di clock potremo osservare il valore in memoria.

Il numero massimo di parole è di 255, che corrisponde anche alla dimensione massima di ogni singola parola (che equivalgono ad un singolo Byte).

1.4 Interpretazioni personali della specifica

Non viene contemplato l'abbassamento del segnale *i_start* nel mentre che il modulo sta computando, questo si potrà riportare a 0 solo in seguito alla conclusione della computazione e al conseguente innalzamento del valore *o_done* da parte del modulo.

Si può anche verificare che mentre il modulo sta procedendo nella computazione, venga alzato dalla memoria il segnale di reset: *i_rst*, che dovrà riportare il modulo allo stato di partenza e anche la macchina a stati del convolutore dovrà essere riportata allo stato iniziale, mentre in memoria rimarranno salvati i dati che ero già stati calcolati e scritti.

2 Architettura

L'architettura è composta da due moduli. Il modulo principale si occupa principalmente dell'interazione con la memoria RAM, prelevandone e salvandosi il numero totali di parole e le singole parole, serializzandole ed infine scrivendone i risultati della computazione che, invece, è effettuata da un altro modulo che simula il comportamento del convolutore.

2.1 Modulo principale

Il modulo è realizzato con una fsm che presenta due processi:

- Il processo `STATE_REG` è il processo che si occupa di cambiare lo stato e aggiornare i valori dei segnali sul fronte di salita del clock e di portare la macchina nello stato di reset quando il segnale *i_rst* è rilevato.
- Il processo `DELTA_FUNCTION` è invece il processo combinatorio che si occupa di modificare i valori dei segnali e decidere il prossimo stato della fsm.

La fsm presenta 10 stati che sono:

- `RESET`: Stato in cui la macchina è inizializzata, quando il segnale *i_rst* viene abbassato si resetta il convolutore e si può avanzare di stato.
- `START`: Stato in cui si attende l'innalzamento del segnale *i_start* e in tal caso si alza il valore *o_en* per permettere la lettura dalla RAM all'indirizzo 0 del numero di parole al ciclo successivo.
- `FETCH_W`: Stato in cui è disponibile nel segnale *i_data* il numero di parole che viene memorizzato, se è 0 si può andare direttamente allo stato di `DONE` altrimenti rimane alto il valore *o_en*, si pone a 1 *o_address* per permettere la lettura della parola al clock successivo e si inizializza il segnale che tiene il conteggio dell'indirizzo di scrittura.
- `FETCH_VAL`: Stato in cui si legge il valore della parola, si abbassa il valore di *o_en* e si aggiorna l'indirizzo di lettura della prossima parola.
- `CALC_FSM`: Stato in cui viene selezionato il bit della parola da passare al convolutore ponendolo in *ifsm_data* e alzando il segnale *ifsm_en*.
- `CALC_OUT_VAL`: Stato in cui si effettua uno shift a sinistra di due posizioni del segnale formato dalla sequenza di bit che compongono le due parole da scrivere in memoria e si mettono nelle posizioni meno significative i due bit prelevati dal segnale *o_fsm* prodotti dal convolutore.
- `PRINT_VAL_1`: Stato in cui si stampa la prima parola, cioè si prendono gli 8 bit più significativi della sequenza prodotta dal convolutore, si mettono nel segnale *o_data*, si innalzano i segnali *o_en* e *o_we* e viene incrementato il segnale che contiene l'indirizzo di stampa.

- `PRINT_VAL_2`: Stato in cui si stampa la seconda parola, cioè si prendono gli 8 bit meno significativi della sequenza prodotta dal convolutore, si mettono nel segnale *o_ data*, si innalzano i segnali *o_ en* e *o_ we* e viene incrementato il segnale che contiene l'indirizzo di stampa.

- `CHECK_END`: Stato in cui si controlla se sono state codificate tutte le parole, nel caso rimangano altre parole da codificare si alza il segnale *o_ en* per permettere la lettura di una nuova parola al prossimo ciclo di clock, nel caso tutte le parole siano già state codificate si innalza il segnale *o_ done*.

- `DONE`: Stato in cui si rimane con *o_ done* alto, in attesa che venga abbassato il segnale *i_ start*.

Alla pagina seguente è riportato il diagramma degli stati della fsm, ad ogni stato è omesso l'arco che riporta la fsm allo stato `RESET` quando viene alzato il segnale *i_ rst*.

2.2 Codificatore convoluzionale

Il convolutore è una macchina a stati finiti, che viene istanziata come un come un componente nella architettura del modulo principale. Questa fsm coincide con quella fornita nella specifica del progetto.

L'interfaccia di questo componente è:

```
entity fsm is
  port(
      clk : in std_logic;
      rst : in std_logic;
      en : in std_logic;
      i : in std_logic;
      o :  out std_logic_vector(1 downto 0)
  );
end fsm;
```

- `CLK` segnale di clock, che coinciderà con il clock utilizzato dal modulo principale.

- `RST` segnale di reset utilizzato per resettare la macchina a stati finiti e riportarla allo stato iniziale.

- `EN` segnale di enable per indicare quando la fsm deve funzionare.

- `I` segnale di input in cui viene passato il bit da codificare.

- `O` segnale di output contenente i bit prodotti dalla codifica dell'input.

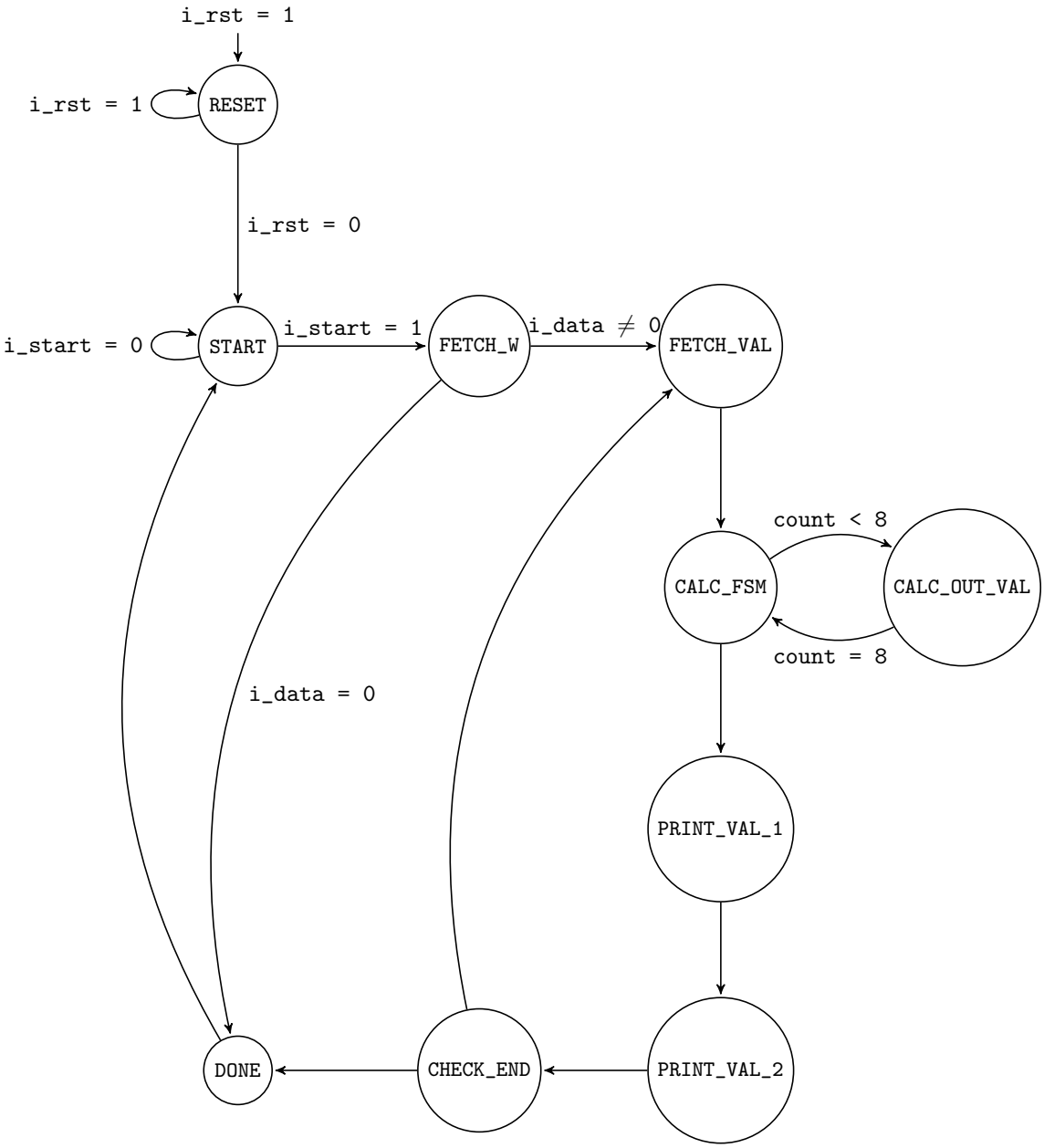

Diagramma degli stati della macchina a stati finiti

3 Risultati sperimentali

3.1 Report di Sintesi

Il componente è perfettamente sintetizzabile, ha un basso utilizzo delle risorse e non istanza alcun latch:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	81	0	134600	0.06
LUT as Logic	81	0	134600	0.06
LUT as Memory	0	0	46200	0.00
Slice Registers	70	0	269200	0.03
Register as Flip Flop	70	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Il vincolo sul tempo massimo di clock utilizzabile è rispettato, infatti avendo un data path delay di 3.276ns è utilizzabile anche con frequenze di un ordine di grandezza maggiore rispetto a quelle richieste.

3.2 Simulazioni

Per verificare il corretto funzionamento del componente, sia nella simulazione behavioural che post-synthesis functional, sono stati utilizzati vari Test Benches, ognuno rivolto a verificarne il comportamento in casi limite o in condizioni casuali, in particolare alcune di questi test sono:

- Numero di parole nullo, all'indirizzo 0 della memoria RAM è contenuto il valore 0, in questo caso il componente non deve leggere le celle successiva ma unicamente innalzare il segnale *o_done* che sarà mantenuto alto fino a quando il segnale *i_start* è riportato a 0.

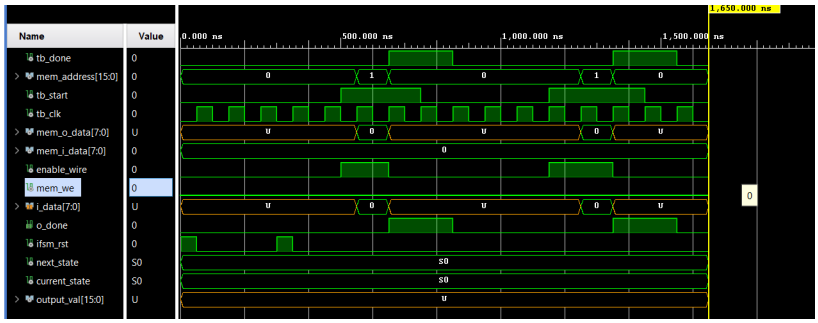

Figure 1: Test con numero di parole nullo

- Test con più flussi di parole: quando il componente finisce la computazione alza *o_done*, che rimane alto fino a quando *i_start* viene abbassato, a quel punto viene resettato il convolutore e si inizializzano nuovamente i segnali, quando *i_start* viene alzato nuovamente inizierà la computazione del nuovo flusso.

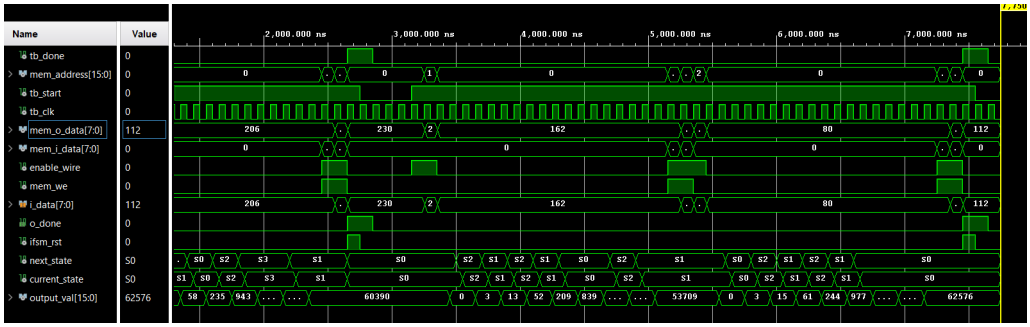

Figure 2: Test con più flussi di parole

- Reset mentre computo: mentre il componente sta ancora codificando le parole salvate nella memoria RAM, viene alzato il segnale *i_rst*, il componente dovrà resettare il convolutore, reinizializzare i segnali. Le parole già codificate saranno disponibili in memoria.

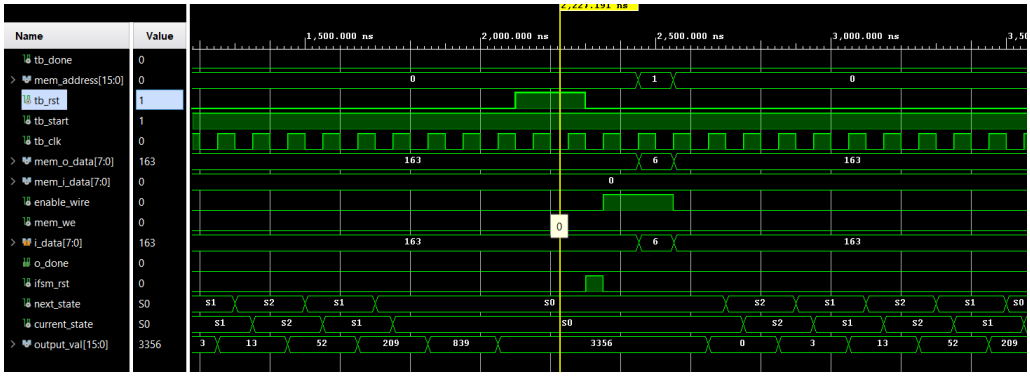

Figure 3: Test con reset mentre computo

- Massimo numero di parole, viene verificato il corretto funzionamento anche in un caso limite per cui ho all'indirizzo 0 della memoria RAM il valore 255.

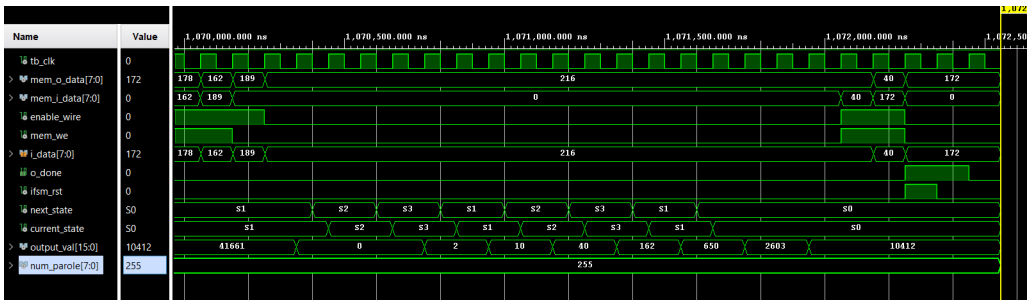

Figure 4: Test con il numero massimo di parole

4 Conclusioni

L'architettura realizzata implementa correttamente la funzionalità richiesta, infatti il comportamento è stato verificato sia con testbench casuali sia con test realizzati con lo scopo di testare delle casistiche limite e particolari sia con Behavioral simulation che in Post-Synthesis Functional simulation. Il vincolo sulla durata massima del periodo di clock della macchina è ampiamente verficato. Inoltre il modulo può essere realmente istanziato su una FPGA in quanto non presenta dei latch.