Progetto Finale di Reti Logiche 2021/2022

Tomaso Stefanizzi
Codice Persona: 10713635
Matricola: 936695

28 novembre 2022

Indice

1 Introduzione

Qui di seguito verranno descritti nel dettaglio il problema da risolvere, le scelte progettuali relative alla macchina, i test creati e i relativi risultati e infine i risultati della sintesi.

1.1 Specifiche del progetto

La consegna di questa prova finale, richiede di progettare un encoder convoluzionale. Il modulo Hardware in questione deve essere descritto in linguaggio VHDL, e si deve interfacciare con una memoria. Il modulo riceve in ingresso una sequenza continua di parole, ognuna di 8 bit, alla quale viene applicato il codice convoluzionale. $\frac{1}{2}$ (numero di bit in ingresso / numero di bit in uscita); infine, il componente scrive in memoria la sequenza in uscita.

1.2 Il Convolutore

Il codificatore convoluzionale (o convolutore) è una macchina sequenziale sincrona con un clock globale e un segnale di reset. Lo scopo di un convolutore è quello di aggiungere dei bit di ridondanza, per favorire una trasmissione di un flusso di dati più robusta. Il funzionamento del convolutore da progettare, è descritto dal diagramma a stati in *Figura 1* , dove "00" è lo stato iniziale, e ogni uscita è rappresentata come $u_k/p1_k p2_k$, dove u_k è il k-esimo bit in ingresso, e di conseguenza $p1_k$ e $p2_k$ sono i bit in uscita.

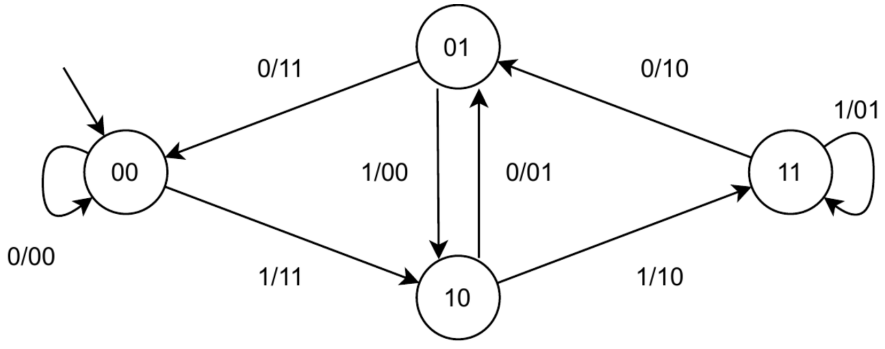


Figura 1: diagramma a stati del convolutore

1.3 Interfaccia del componente

Il componente da descrivere deve avere un'interfaccia composta dai seguenti segnali:

- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- i_start è il segnale di START generato dal Test Bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- 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.

1.4 La RAM

L'architettura della RAM non viene istanziata nel convolutore, dato che è già presente nel Test Bench. In particolare, la memoria sarà utilizzata in questo modo:

- cella 0: contiene il numero di parole in ingresso
- celle 1-255 : contengono le parole da leggere
- celle >= 1000 : conterranno il risultato della computazione

1.5 Il Clock

Il convolutore deve funzionare con un clock di almeno 100 ns.

1.6 Esempio di Funzionamento

in prima battuta, bisogna leggere il numero di parole da processare, presente all'indirizzo "0" della RAM. Successivamente, viene letta, un bit alla volta, una parola di 1 byte. A ogni lettura di un bit, il convolutore genera 2 bit; di conseguenza, a termine della codifica di ogni parola, verranno generate 2 parole di 1 byte ciascuna. Infine, se il numero di parole processate è pari al numero presente all'indirizzo 0 della RAM, la codifica sarà completata. Viene di seguito fornito un esempio di ciò che accade al termine della computazione.

Siano "I" il flusso di parole in input, e "O" il flusso di bit in output:

cella	contenuto
0	2
1	162
2	75
[...]	...
1000	209
1001	205
1002	247
1003	210

Tabella 1: contenuto nella RAM

- **I**: 0000010 10100010 01001011
- **O**: 1010001 11001101 11110111 11010010

Come si può notare dalla *Tabella 1*, all'indirizzo "0" viene memorizzato il numero delle parole (che corrisponde alla prima parola in ingresso, proposta in codifica binaria nel flusso "I"), dall'indirizzo "1" all'indirizzo "2" sono invece presenti le parole da codificare, infine dall' indirizzo "1000" all'indirizzo "1003" è presente l'ouptut della computazione (corrispondente al flusso di bit rappresentato in "O").

2 Scelte progettuali

Per la risoluzione del problema assegnato, ho ritenuto di procedere con l'implementazione di una macchina a stati finiti tramite un singolo processo sensibile al segnale di clock.

2.1 Segnali

per prima cosa è stato definito il segnale current_state, il quale mantiene memoria dello stato attuale della macchina a stati. successivamente sono stati definiti read_address e write_address, inizializzati rispettivamente a 0 e a 1000, per poter poi eseguire le operazioni di lettura e scrittura in maniera più flessibile. Infine, word_buffer è il vettore designato per ricevere in input la parola da processare, mentre output contiene l'intera codifica di output di 16 bit, che verrà divisa in 2 parti uguali al fine di scrivere le parole da 1 byte in uscita in sequenza; ogni volta che una parola viene assegnata a word_buffer, viene posto a true il flag booleano ok_number.

```
    signal current_state : STATE_TYPE := INIT;

    -- Input/Output addresses --
    signal read_address : STD_LOGIC_VECTOR(15 downto 0)
    signal write_address : STD_LOGIC_VECTOR(15 downto 0)


    -- Number of words to process --
    signal number_of_words : STD_LOGIC_VECTOR(7 downto 0)

    -- Flags --
    signal ok_number : boolean

    -- Tempourary buffers --
    signal word_buffer : std_logic_vector(7 downto 0)
    signal output : STD_LOGIC_VECTOR(15 downto 0)
```

2.2 Stati

La macchina, rappresentata dal diagramma in *Figura 2* è composta da 15 stati, che sono:

- INIT: stato iniziale della macchina: inizializza i segnali ai valori di default.
- READY: stato che si occupa di "connettere" la memoria al componente hardware per la lettura di dati in ingresso.
- READY_NEXT: funziona da stato di ridondanza per attendere la corretta lettura dei dati. Inoltre, se il numero di parole è già stato letto, indirizza la macchina allo stato SWITCH, senza passare da READ_NUMBER.
- READ_NUMBER: stato che si occupa di leggere il numero di parole da processare.

- WAIT_STATE: funziona da stato di ridondanza per attendere la corretta lettura dei dati.

- SWITCH: stato che ha duplice funzione: decide, se il numero di parole rilevato allo stato READ_NUMBER è 0, e quindi di alzare a 1 il segnale o_done e quindi indirizzare la macchina allo stato DONE, altrimenti di indirizzare la macchina allo stato READ.

- READ: stato che si occupa della lettura della parola da codificare (che verrà salvata nel segnale "word_buffer").

- CONV_START: imposta "o_en" a 0, affinchè non vi siano perdite di dati in input.

- CONVOLVER: è lo stato fulcro del convolutore: contiene infatti la logica dell'intero codificatore. Questa viene fatta tramite l'utilizzo di 2 variabili, definite ad'inizio processo, ossia "i", un semplice intero, e "STATE", un vettore di 2 bit che modella lo stato corrente del codificatore in *Figura 1.* All'interno di questo stato, è presente un ciclo while, che permette di svolgere l'intera computazione in un solo stato, dal momento che l'assegnamento alle variabili avviene immediatamente, a differenza dei segnali che si aggiornano solo a fine processo. L'assegnazione al segnale "output" dei bit in uscita, risulta sempre sincronizzato con il segnale di clock.

- WRITE: stato designato per la scrittura della prima parola (output(15 to 8)); inoltre, "collega" la macchina alla memoria e imposta "o_we = 1", in modo da permettere la scrittura nella RAM.

- WAIT_WRITE: funziona da stato di ridondanza per attendere la corretta scrittura dei dati.

- WAIT_NEXT: stato designato per la scrittura della prima parola (output(7 to 0)); inoltre, se il numero di parole processate è pari al numero salvato in "number_of_words", indirizza la macchina allo stato DONE, alzando a 1 il segnale "o_done".

- OK: imposta "o_we = 1", disabilitando la funzione di scrittura in memoria.

- WAIT_READ: funziona da stato di ridondanza per attendere la corretta lettura dei dati.

- DONE: attende che venga abbassato a o il segnale "i_start", per poter reiniziare una nuova codifica, e di conseguenza, abbassare a 0 il segnale "o_done".

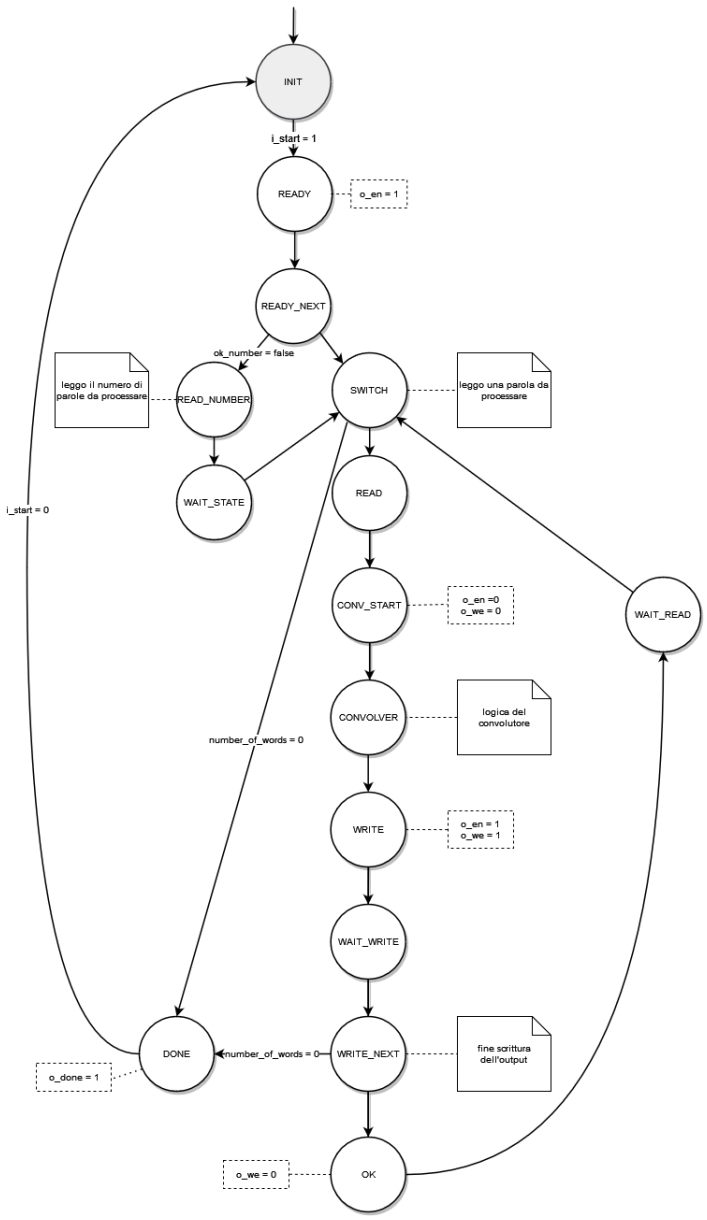


Figura 2: macchina a stati

2.3 Processi

La macchina utilizza un unico processo, main, sincronizzato con il segnale di clock; in questo modo, la logica della macchina e la propagazione dei segnali non vengono separati. In qualsiasi stato della macchina, in presenza di un segnale di reset, la macchina viene ripristinata alle condizioni iniziali, ossia alle condizioni impostate nello stato INIT.

3 Test e Risultati

3.1 Test utilizzati

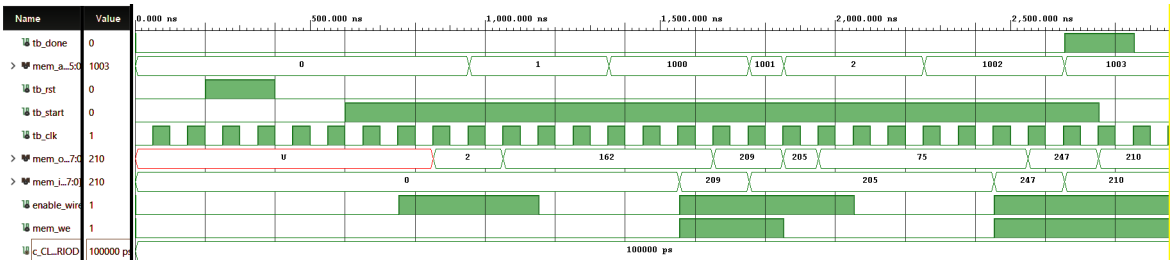
Per verificare la corretta progettazione del componente, ho utilizzato 6 test bench, ognuno con caratteristiche significative e/o con casi limite:

1. **tb_example**: test fornito dal docente.
2. **tb_seq_min**: il numero di parole da codificare è 0.
3. **tb_seq_max**: il numero di parole da codificare è molto grande.
4. **tb_reset**: test con reset asincrono.
5. **tb_re_encode**: test con 3 runs consecutive.
6. **tb_tre_reset**: test con 3 runs consecutive, reset sulla prima run.

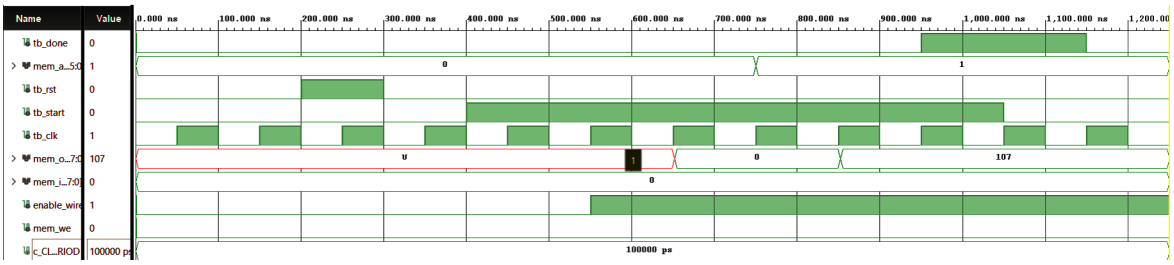
3.2 Risultati ottenuti

Di seguito sono riportati le schermate delle waveform ottenute e della stampa "TEST PASSATO" nella console al termine della "Run All" del progetto in Post Synthesis Functional Simulation, per ogni test bench usato.

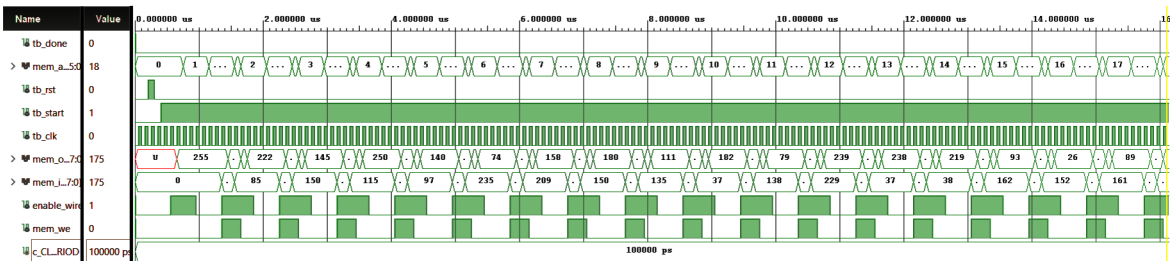
1. **tb_esempio**:



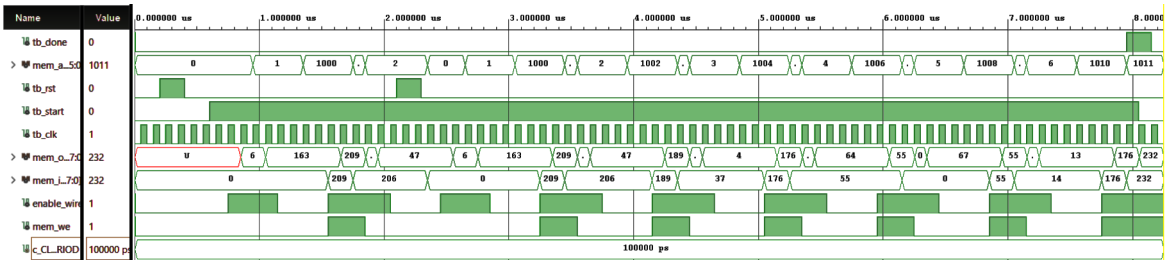
2. **tb_seq_min**:



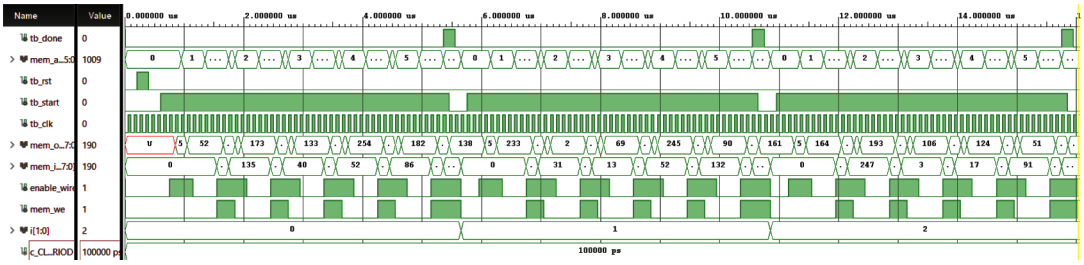
3. **tb_seq_max**:



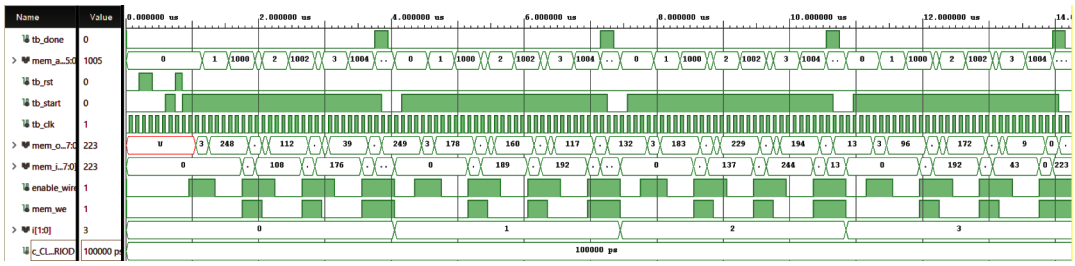
4. **tb_reset**:



5. **tb_re_encode**:



6. **tb_tre_reset**:



Vengono riportati di seguito i tempi di esecuzione per ciascuna simulazione, in pre e post-sintesi.

test	behavioural	post-synthesis
tb_example	2850 ns	2950,1 ns
tb_seq_min	1150 ns	1250,1 ns
tb_seq_max	230350 ns	230450,1 ns
tb_reset	8150 ns	8250,1 ns
tb_e_encode	15550 ns	16050,1 ns
tb_tre_reset	13550 ns	14250,1 ns

Tabella 2: risultati test con timing

4 Risultati sintesi

Il componente descritto viene sintetizzato correttamente, e non sono presenti latches, come indicato nella *Figura 3*, quindi il suo comportamento risulta completamente prevedibile. Si può nuotare inoltre che il componente viene realizzato con 67 LUTs e 98 Flip Flops. Il WNS, o *Worst Negative Slack* è di 96.439 ns, pertanto la richiesta di funzionamento del componente entro i 100 ms di clock viene ampiamente rispettato.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	67	0	0	134600	0.05
LUT as Logic	67	0	0	134600	0.05
LUT as Memory	0	0	0	46200	0.00
Slice Registers	98	0	0	269200	0.04
Register as Flip Flop	98	0	0	269200	0.04
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

Figura 3: output comando report_utilization in Tcl console

Viene di seguito riportato lo schematico della rete ottenuta tramite la sintesi

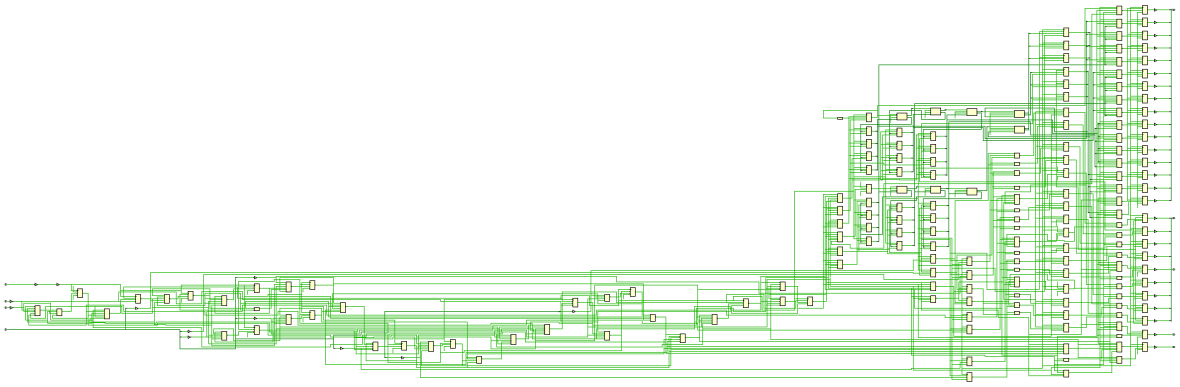


Figura 4: schematico della rete

5 Conclusioni

Nella sua prima versione, ho descritto il componente hardware facendo uso di due processi: uno sensibile al segnale di clock, che gestiva esclusivamente l'aggiornamento dei registri; infatti inizialmente ogni signal aveva un altro segnale next per gestire l'aggiornamento di questo. Il secondo processo, invece, era sensibile ai segnali i_data, next_state e i_start. Tuttavia, a seguito della sintesi, erano presenti del Latches, che mi hanno fatto decidere di convertire il progetto a singolo processo, rendendo inoltre il codice meno verboso e diminuendo sensibilmente la quantità di segnali utilizzati, pur mantenendo quasi del tutto invariata la struttura della macchina progettata. In conclusione, il componente hardware descritto rispetta in pieno tutti i requisiti forniti dal docente, ed è stato testato correttamente sia in pre-sintesi che in post-sintesi, con esito positivo.