Prova Finale di Reti Logiche

Montanari Tommaso e Negri Riccardo

15 Maggio 2022

Docente:	Salice Fabio		
Studente 1:	Montanari Tommaso	10661941	932673
Studente 2:	Negri Riccardo	10729927	936820

1 Introduzione

La prova prevede l'implementazione in VHDL di un componente che opera su una memoria e svolge la seguente operazione: il componente deve per prima cosa leggere il primo byte dalla memoria che identifica il numero di parole che sono state fornite come input, questa informazione è importante per capire quando la macchina deve terminare la lettura.
Dopodiché ogni parola successiva viene tradotta in due parole che vengono scritte progressivamente in un'altra parte della memoria.
Le parole di memoria sono interpretate come un flusso continuo e l'uscita è un flusso di lunghezza doppia. Visto che i byte vengono tradotti come un unico flusso l'output di una certa parola dipende anche da quelle precedenti e non possono essere processate separatamente.

1.1 Esempio

Indirizzo	Valore	Codifica binaria
0	2	0000 0010
1	35	0010 0011
2	161	1010 0001

Questo stato della memoria si traduce nell'input [35, 161] e in questo caso la lunghezza dell'input W=2 quindi mi aspetto una lunghezza dell'output Z=4.

Indirizzo	Valore	Codifica binaria
1000	13	0000 1101
1001	206	1100 1110
1002	97	0110 0001
1003	195	1100 0011

Rappresenta l'output [13, 206, 97, 195] dove la codifica dei numeri [13, 206]

è l'output prodotto dai bit dal 35 in ingresso. Mentre [97, 195] sono il risultato dei bit presenti nella la codifica di 161.

1.2 Ipotesi Progettuali

- Si utilizza la scheda Artix-7 FPGA xc7a200tfbg484-1
- Ogni byte può contenere numeri da 0 a 255.
- La quantità di numeri in ingresso (W) è contenuta in una parola da un byte quindi anche il numero massimo di parole da tradurre è 255.
- Dato che l'input occupa al massimo 256 byte posso scrivere sui byte successivi quindi l'output parte sempre dal millesimo indirizzo di memoria che sicuramente non contiene l'input

2 Architetttura

L'implementazione è stata realizzata in un solo modulo per sempllicità, si possono distinguere due parti all'interno della macchina che cooperano per raggiungere il risultato. È presente una macchina che simula il funzionamento dell'automa a stati finiti fornito nella specifica, questa è responsabile della traduzione uno a due dei bit. Inoltre è presente una seconda macchina principale che si occupa della lettura e scrittura dei flussi di bit e tratta la prima come una scatola nera. Ciascuna macchina ha una variabile separata che contiene il suo stato.

2.1 Segnali

- **i_clk:** Segnale di clock
- **i_rst:** Segnale di reset, deve riportare la macchina allo stato iniziale.
- **i_start:** Quando viene portato a 1 la macchina comincia ad operare e prima di terminare aspetta che sia riportato a 0 (handshake).
- **i_data:** Vettore da 8 bit che contiene il la parola di memoria che è stata letta.
- **o_address:** Un vettore di 16 bit che contiene l'indirizzo della parola da scrivere o da leggere.
- **o_done:** Quando la macchina ha finito l'esecuzione viene portato a 1 e aspetta che i_start sia 0 poi anche 0_done è riportato a 0 (handshake).
- **o_en:** Se settato a 1 richiede l'operazione di lettura o scrittura.
- **o_we:** Se settato a 1 l'operazione richiesta da o_en è scrittura altrimenti è lettura.
- **o_data:** Vettore da 8 bit che contiene il la parola di memoria da scrivere.

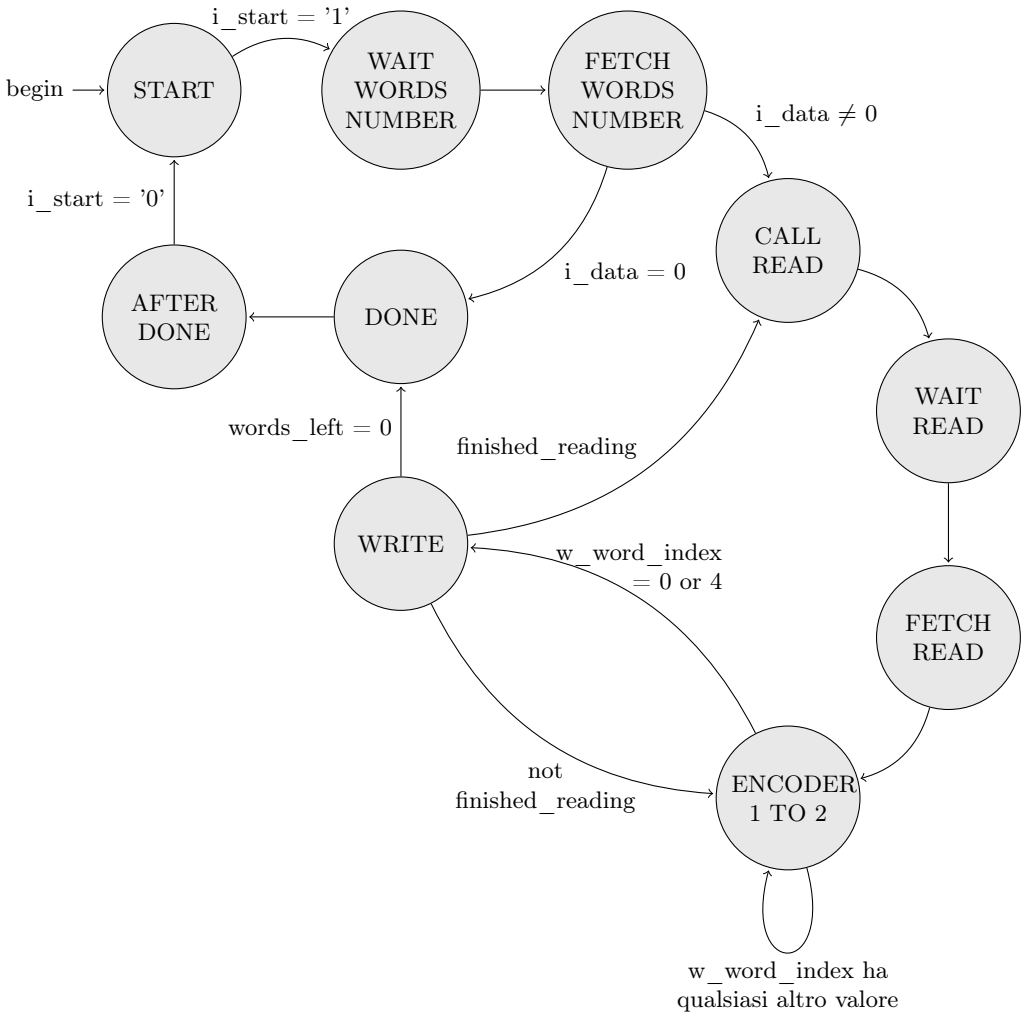
2.2 Descrizione ad alto livello

Questa è la descrizione a parole delle fasi che l'esecuzione segue. All'interno del programma ciascuna di queste diverse fasi corrisponde ad uno stato (o a più stati).

1. Inizio:
   (a) Legge il primo byte di memoria contenente la lunghezza dell'input e salva il valore in una variabile.
   (b) Inizializza il punto di lettura al secondo byte e quello di scrittura al millesimo.
2. Traduzione:
   (a) Legge un byte e sposta il punto di lettura.
   (b) Simula un passo dell'automa dando in ingresso un bit dal byte appena letto. (Ripetuto 4 volte)
   (c) L'output dell'automa ora contiene 8 bit quindi viene salvato in una parola di memoria e si incrementa l'indirizzo di scrittura.
   (d) Se sono rimasti bit da tradurre nel byte letto torna al punto 2.b
   (e) Altrimenti se sono rimasti byte da leggere torna al punto 2.a
   (f) Altrimenti termina.

Ogni operazione di lettura nella memoria si scompone in più stati nel programma: richiesta della lettura, attesa della fine dell'operazione, utilizzo del dato letto o salvataggio in una variabile.

2.3 Diagramma degli stati



- **START:** Lo stato in cui la macchina è all'inizio dell'esecuzione del programma. In presenza del segnale i_state = '1' imposta valori iniziali e passa allo stato successivo.
- **WAIT_WORDS_NUMBER:** Un ciclo di clock dopo che la lettura è stata richiesta riporta a 0 il segnale di lettura.
- **FETCH_WORD_NUMBER:** Dopo la fine della lettura salva il numero di byte in input nella variabile words_left. Nel caso particolare in

cui ci sono 0 byte in ingresso passa direttamente allo stato Done altrimenti incomincia con la lettura del primo byte.

- **CALL_READ:** Manda segnale di richiesta di lettura e imposta l'indirizzo corretto.
- **WAIT_READ:** Un ciclo di clock dopo che la lettura è stata richiesta riporta a 0 il segnale di lettura.
- **FETCH_READ:** Dopo che la lettura è terminata salva il valore letto in w_word.
- **ENCODER_1_TO_2:** In questo stato viene simulato l'automa che effettua la traduzione, lo stato di tale automa è salvato in una variabile separata chiamata internal_state. Ogni 4 bit tradotti z_word, ovvero la parola da un byte in uscita, è piena e quindi si passa alla scrittura di tale parola in memoria.
- **WRITE:** Scrive z_word in memoria. Dopodiché se rimangono dei bit da leggere in w_word torna allo stato dell'encoder. Se la parola è finita ma ci sono altre parole in input da leggere torna a Call Reader, se tutte le parole sono state tradotte allora passa a Done.
- **DONE:** Manda il segnale o_done a 1.
- **AFTER_DONE:** Prima di tornare allo stato Start si assicura che il segnale i_start che era stato dato all'inizio sia stato riportato a 0. Inotre riporta a 0 il segnale o_done.

2.4 Macchina a stati finiti?

3 Risultati sperimentali

3.1 Sintesi

Il componente sviluppato è sintetizzabile ed implementabile.

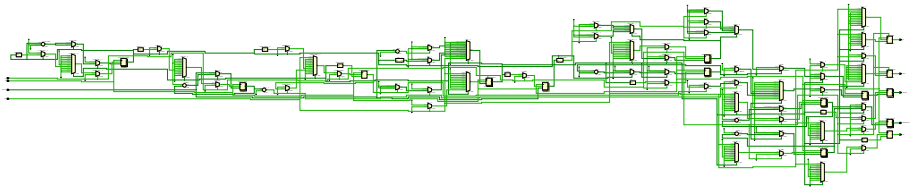


Figure 1: Schema post-sintesi

Nella seguente tabella sono indicati i componenti utilizzati nella sintesi.

Site Type	Used	Available	Utilization%
LUT as Logic	117	134600	0.09
Register as Flip Flop	95	269200	0.04
F7 Muxes	1	67300	<0.01

3.2 Simulazioni

Al fine di verificare il corretto funzionamento del componente sono stati eseguiti diversi test bench in simulazioni Behavioural, Post-synthesis functional e Post-synthesis timing. Di seguito sono riportati i test significativi effettuati con relativa spiegazione, tempi ottenuti e grafico d'onda.

3.2.1 Flussi successivi

Tramite questo test si verifica la correttezza nel caso di codifica di più flussi uno dopo l'altro. Nel testbench usato vengono codificati tre flussi (senza reset dopo ogni flusso).

Behavioural 21850 ns

Post-synthesis functional 22350100 ps

Post-synthesis timing 22353714 ps

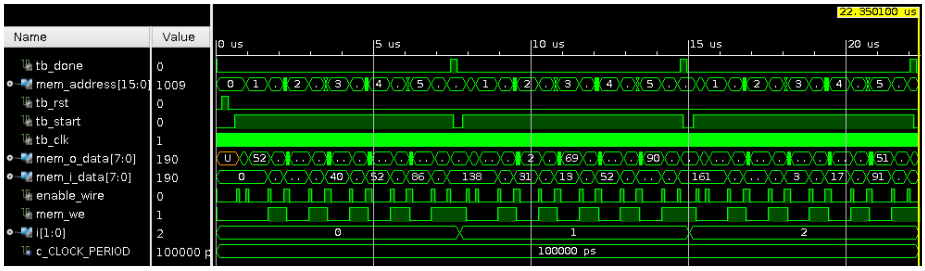


Figure 2: Post-synthesis functional simulation waveform

3.2.2 Reset asincrono

Tramite questo test si verifica il corretto funzionamento del reset asincrono.

Behavioural 10650 ns

Post-synthesis functional 10750100 ps

Post-synthesis timing 10753714 ps

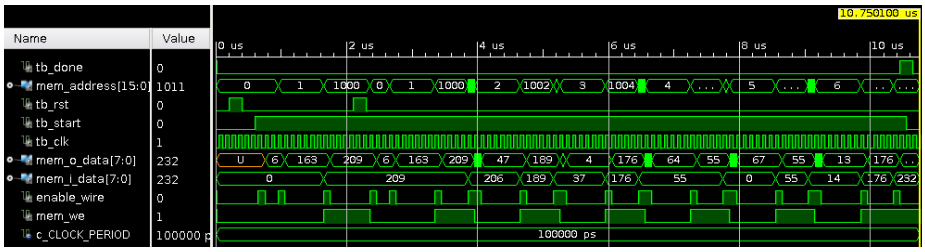


Figure 3: Post-synthesis functional simulation waveform

3.2.3 Sequenza di lunghezza massima

Tramite questo test si verifica il corretto funzionamento nel caso di sequenza di ingresso di lunghezza massima: 255 byte.

Behavioural 332450 ns

Post-synthesis functional 332550100 ps

Post-synthesis timing 332553714 ps

In questo caso non si riporta grafico della forma d'onda perché non è possibile distinguere i valori che i segnali assumono nella vista "fit".

3.2.4 Sequenza di lunghezza nulla

Tramite questo test si verifica il corretto funzionamento nel caso di sequenza di ingresso di lunghezza minima: 0 byte.

Behavioural 950 ns

Post-synthesis functional 1050100 ps

Post-synthesis timing 1053714 ps

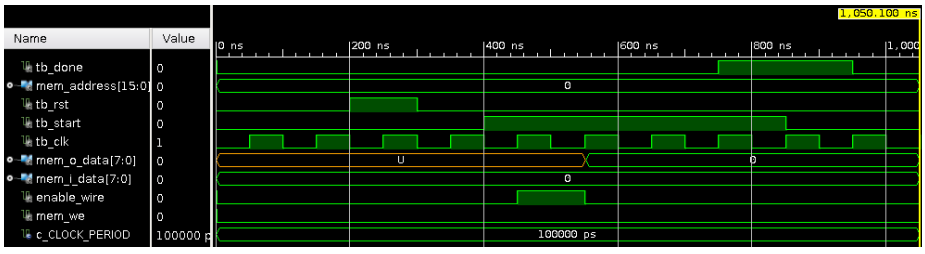


Figure 4: Post-synthesis functional simulation waveform

3.2.5 Double processing sulla stessa RAM

Tramite questo test si verifica il corretto funzionamento nel caso di scrittura sulla stessa RAM.

Behavioural 9250 ns

Post-synthesis functional 9550100 ps

Post-synthesis timing 9553714 ps

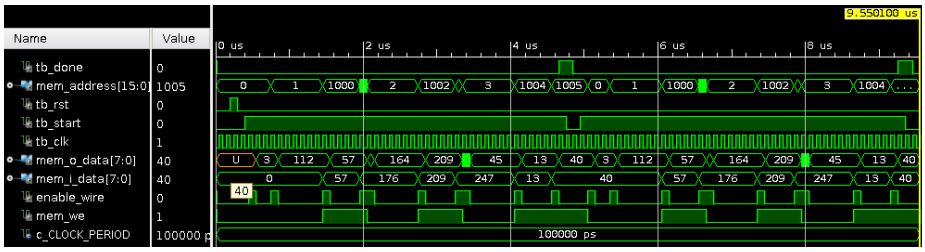


Figure 5: Post-synthesis functional simulation waveform

3.2.6 Tests con reset

Test bench, con reset dopo ogni test, che effettua 1000 test (generati casualmente con uno script Python).

Behavioural 172743250 ns

Post-synthesis functional 172943150100 ps

Post-synthesis timing 172943153714 ps

In questo caso non si riporta grafico della forma d'onda perché non è possibile distinguere i valori che i segnali assumono nella vista "fit".

3.2.7 Tests senza reset

Test bench, senza reset dopo ogni test, che effettua 1000 test (generati casualmente con uno script Python).

Behavioural 172543450 ns

Post-synthesis functional 172743350100 ps

Post-synthesis timing 172743353714 ps

In questo caso non si riporta grafico della forma d'onda perché non è possibile distinguere i valori che i segnali assumono nella vista "fit".

4 Conclusioni