POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Prova Finale

Progetto di Reti Logiche
Ingegneria Informatica

Studente: **Yanfeng Li**

Codice Persona: 10650552
Prof: Prof. Fabio Salice, Prof. Cassano Luca Maria
Anno Accademico: 2021-2022

Contents

1 | Introduzione

1.1. Scopo del progetto

Lo scopo del progetto è quello di descrivere in VHDL e sintetizzare il componente HW che implementa la specifica richiesta, interfacciandosi con una memoria dove sono memorizzati i dati e dove andrà scritto il risultato finale.

1.2. Specifiche generali

Il componente da implementare deve ricevere in ingresso una sequenza continua di W parole, ognuna di 8 bit, e restituisce in uscita una sequenza continua di Z parole, ognuna da 8 bit. Ognuna delle parole di ingresso viene serializzata; in questo modo viene generato un flusso continuo U da 1 bit. Su questo flusso viene applicato il codice convoluzionale 1/2 (ogni bit viene codificato con 2 bit) secondo lo schema riportato in figura; questa operazione genera in uscita un flusso continuo Y. Il flusso Y è ottenuto come concatenamento alternato dei due bit di uscita. Utilizzando la notazione riportata in figura, il bit uk genera i bit p1k e p2k che sono poi concatenati per generare un flusso continuo yk (flusso da 1 bit). La sequenza d'uscita Z è la parallelizzazione, su 8 bit, del flusso continuo yk.

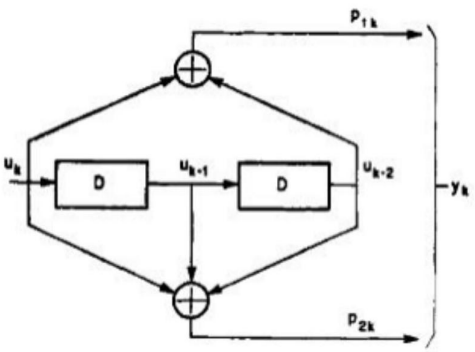

Figure 1.1: Codificatore convoluzionale con tasso di trasmissione 1/2

1.3. Interfaccia del componente

II componente da descrivere deve avere la seguente interfaccia.

```
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:

- il nome del modulo deve essere project_reti_logiche
- 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.

1.4. Dati e descrizione della memoria

I dati sia di input che di output sranno scritti su una memoria di tipo RAM con indirizzamento al byte e bus indirizzi di 16 bit.La sequenza di byte letta in ingresso è trasformata nella sequenza di bit U da elaborare. La quantità di parole W da codificare è memorizzata nell'indirizzo 0; il primo byte della sequenza W è memorizzato all'indirizzo 1. Lo stream di uscita Z deve essere memorizzato a partire dall'indirizzo 1000 (0000000111101000 in binario). La dimensione massima della sequenza di ingresso è 255 byte, cioè 255 parole da elaborare, in memoria cella 1 sarà scritta 255 (11111111 in binario).

1.5. Esempi

La seguente sequenza di numeri mostra un esempio del contenuto della memoria al termine di una elaborazione. I valori che qui sono rappresentati in decimale, sono memorizzati in memoria con l'equivalente codifica binaria su 8 bit senza segno.

W: 10100010 01001011
Z: 11010001 11001101 11110111 11010010

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	2	\\ Byte lunghezza sequenza di ingresso
1	162	\\ primo Byte sequenza da codificare
2	75	
[...]		
1000	209	\\ primo Byte sequenza di uscita
1001	205	
1002	247	
1003	210	

Figure 1.2: Sequenza lunghezza 2

W: 01110000 10100100 00101101
Z: 00111001 10110000 11010001 11110111 00001101 00101000

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	3	\\ Byte lunghezza sequenza di ingresso
1	112	\\ primo Byte sequenza da codificare
2	164	
3	45	
[...]		
1000	57	\\ primo Byte sequenza di uscita
1001	176	
1002	209	
1003	247	
1004	13	
1005	40	

Figure 1.3: Sequenza lunghezza 3

1.6. Deduzione dalle specifiche

P1(t) = U(t) xor U(t-2)
P2(t) = U(t) xor U(t-1) xor U(t-2)
Y(t) è la concatenazione tra P1(t) e P2(t)

Figure 1.4: Espressione logiche

U(t-2)	U(t-1)	U(t)	P1(t)	P2(t)
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	0
1	1	1	0	1

Figure 1.5: Tabella di verita

2 | Architettura

Quando il segnale **i_start** in ingresso viene portato a 1, il componente sviluppato inizia l'elaborazione spostandosi dallo stato **S0** al primo stato della computazione. Una volta terminata la computazione, dopo aver scritto il risultato in memoria, il componente alza il segnale **o_done**. La test bench risponde abbassando **i_start** e, di conseguenza il componente riporta 0 **o_done**. Il componente quindi ritorna nello stato **S0** in attesa che il segnale **i_start** ritorni alto. Il componente dispone inoltre di un segnale **i_rst** che, insieme agli altri elencati in capitolo precedente, ci hanno portato a definire una **FSM(D)**, macchina a stati finiti con data path, che combina una normale FSM con tipici circuiti sequenziali. Nelle seguenti sezioni troviamo infatti sia la descrizione della FSM sia la descrizione della parte sequenziale della macchina che permette la gestione dei registri utilizzati.

2.1. Datapath

Intera datapath è descritta da diversi componenti di base che sono, registri, multiplexer, sommatori, sottrattori. Complessivamente si potrebbero individuare 4 gruppi raggruppando i componenti in base alle loro funzioni. Che sono ciascunamente il componente per la lettura, il componente per il conteggio, il componente per la scrittura e il componente per indirizzamento. Di seguito verranno descritti in dettaglio.

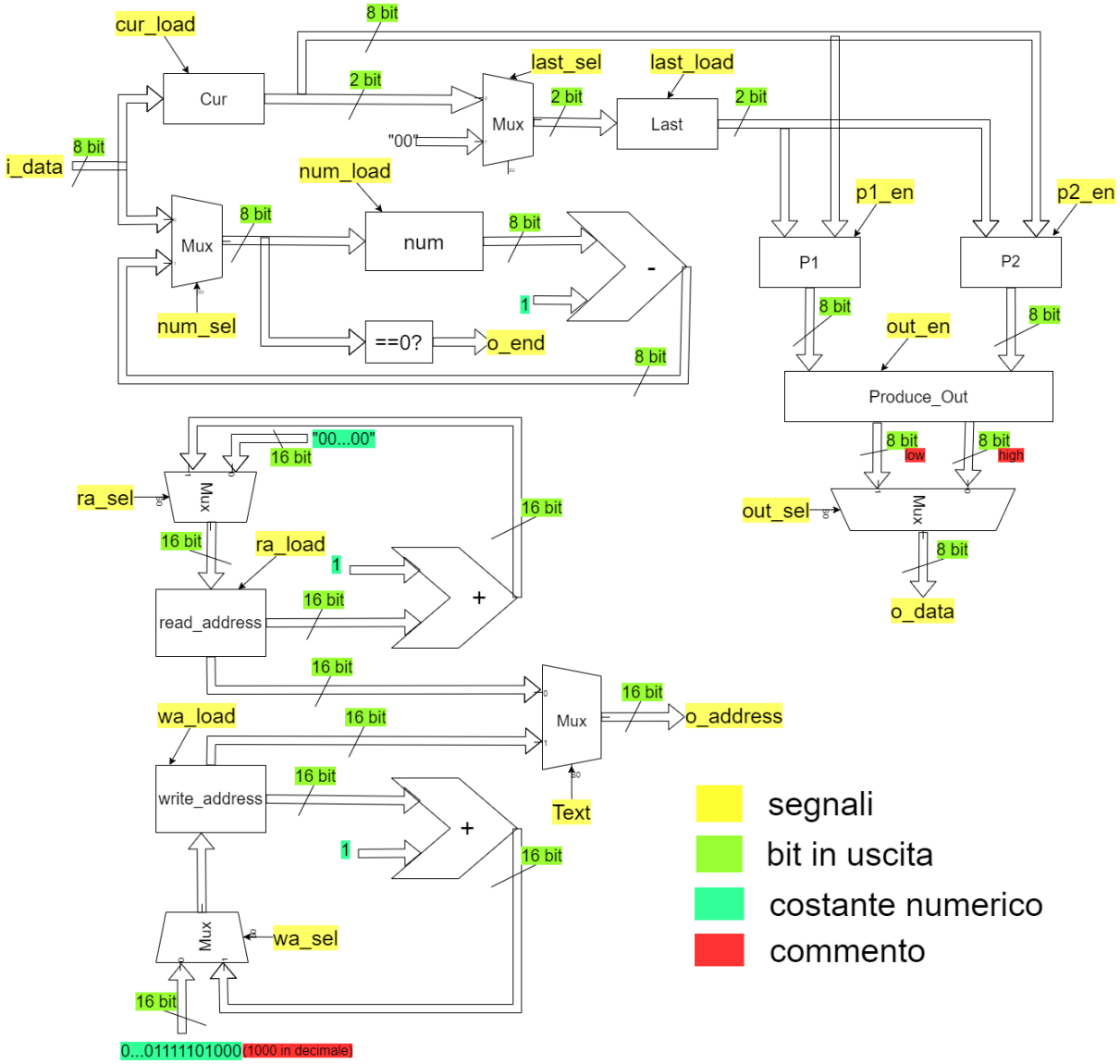

Figure 2.1: data path

2.1.1. Componente per la lettura

Il componente di lettura ha due registri e un multiplexer, in particolare

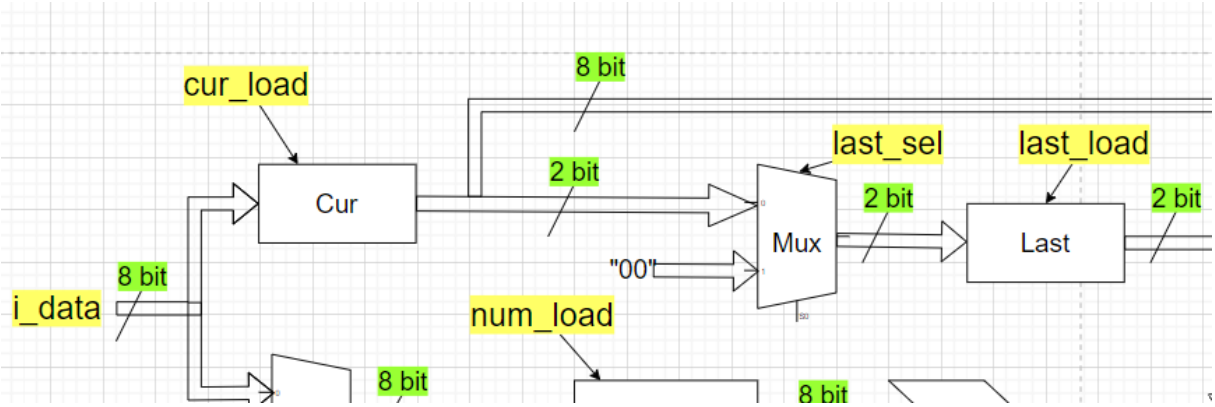

Figure 2.2: componente per la lettura

- il registro **cur** salva i dati in ingresso in istante corrente.
- il registro **last** salva inizialmente "00" binario successivamente le ultime due bit del contenuto che si trova in registro **cur**.
- il **multiplexer** con l'uscita collegato al registro last serve per la selezione del valore in ingresso.

2.1.2. Componente per il conteggio

Il componente per il conteggio è costruito da un registro, un multiplexer, un sottrattore e un comparatore.

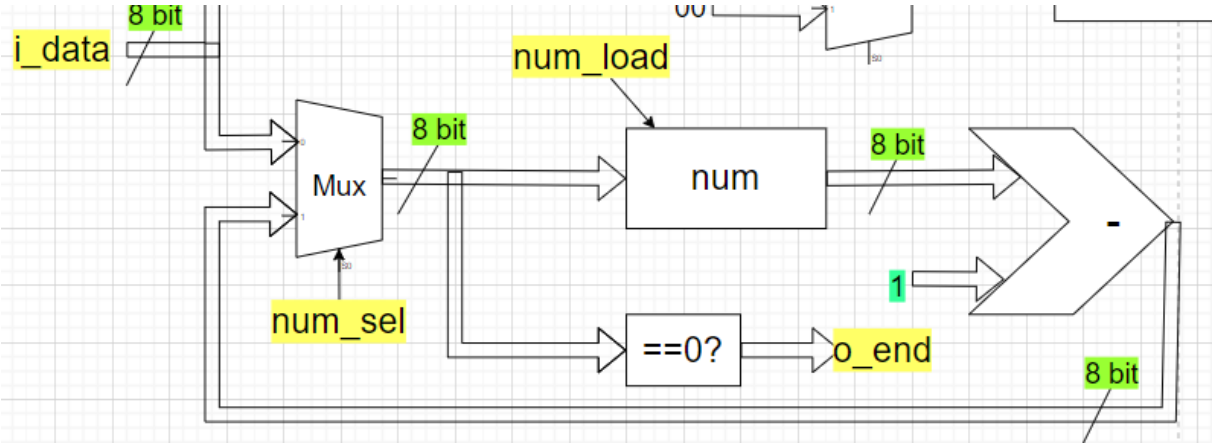

Figure 2.3: Componente per il conteggio

- il registro **num** salva il numero di parole che ci sono da leggere
- il **multiplexer** con l'uscita collegato al registro num seleziona ingresso da i_data o dal sottratore.

- il **sottratore** decrementa il conteggio salvato in registro di uno e lo restituisce indietro.
- il **comparatore** posizionato all'uscita del multiplexer cambia il valore del segnale **o_end** appenna rivela che output del mux sia 0.

2.1.3. Componente per la produzione e scrittura di output

Il componente per la produzione e scrittura di output è composto da 3 registri con algoritmo di calcolo di output incorporato, e un multiplexer.

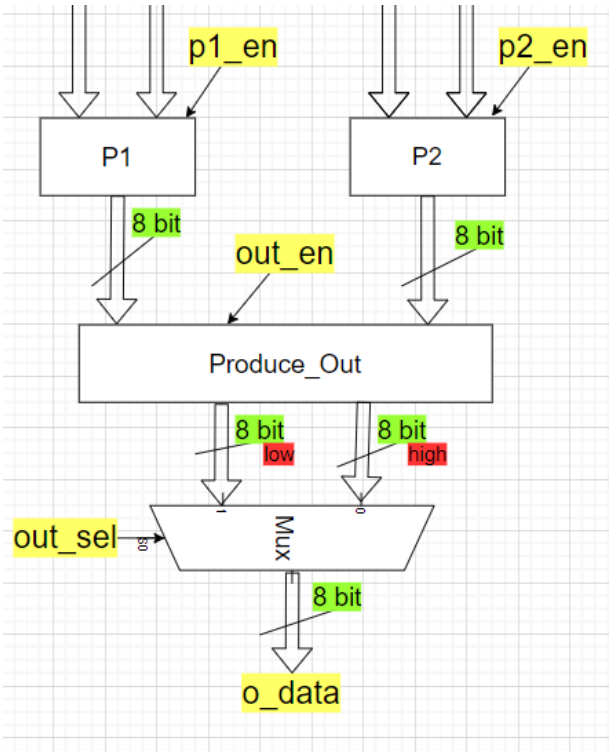

Figure 2.4: Componente per output

- il registro **P1** prende in ingresso input corrente e ultimi due bit del input precedente e ne genera il risultato seguendo il processo indicato nella specifica.

```
if(p1_en = '1') then
    temp(7) := o_cur(7) xor o_last(1);
    temp(6) := o_cur(6) xor o_last(0);
    for i in 5 downto 0 loop
        temp(i) := o_cur(i) xor o_cur(i+2);
    end loop;
    o_p1 <= temp;
```

Figure 2.5: P1(t) = U(t) xor U(t-2)

- il registro **P2** ha lo stesso ingresso di **P1**, genera però output diverso.

```
if(p2_en = '1') then
    temp(7) := (o_cur(7) xor o_last(0)) xor o_last(1);
    temp(6) := (o_cur(6) xor o_cur(7)) xor o_last(0);
    for i in 5 downto 0 loop
      temp(i) := (o_cur(i) xor o_cur(i+1)) xor o_cur(i+2);
    end loop;
    o_p2 <= temp;
```

Figure 2.6: P2(t) = U(t) xor U(t-1) xor U(t-2)

- il registro **Produce_out** riceve in ingresso gli output di **P1** e **P2** di dimensione 8 bit e lo concatena secondo il processo riportato nella specifica.

```
if(out_en = '1') then
    j := 15;
    for i in 7 downto 0 loop
        temp(j) := o_p1(i);
        temp(j-1) := o_p2(i);
        j := j-2;
    end loop;
    o_out <= temp;
```

Figure 2.7: OUT(t) = P1(t) concatena P2(t)

2.1.4. Componente per la gestione dell'indirizzamento

Il componente per la gestione dell'indirizzamento è costruito da 3 multiplexer, 2 registri e 2 sommatori. Il componente che alla sua volta può essere diviso in 2, uno per indirizzo di scrittura l'altra per indirizzo di lettura.

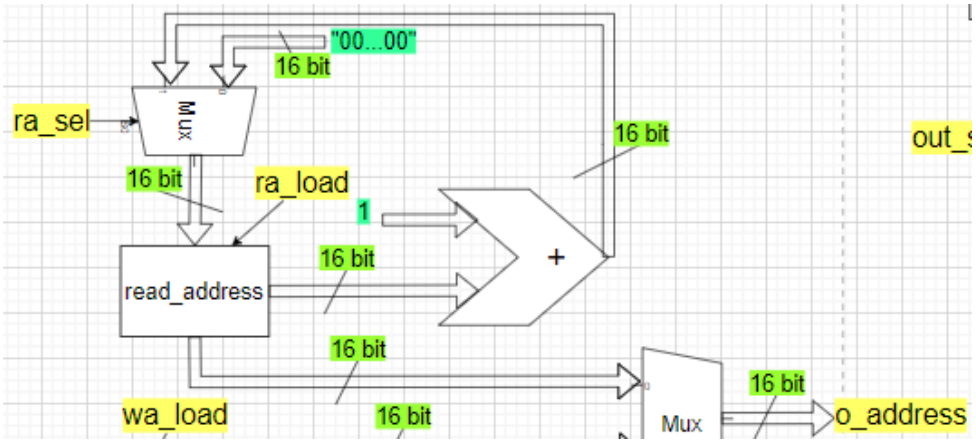

Figure 2.8: registro indirizzo lettura

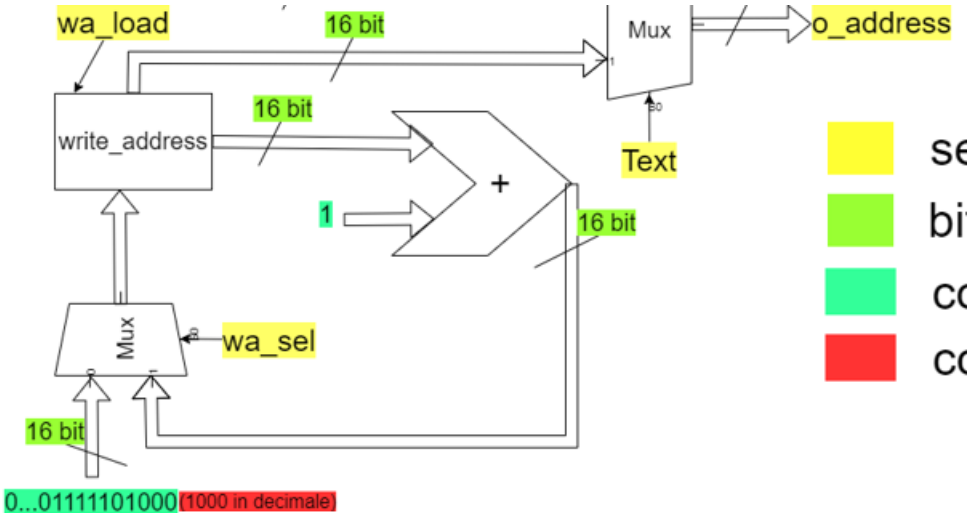

Figure 2.9: registro indirizzo scrittura

- il registro **read/write_address** salva indirizzo di lettura/scrittura al suo interno
- il **sommatore** incrementa indirizzo di lettura/scrittura di 1 e ne restituisce ai registri di lettura/scrittura.
- il **multiplexer** seleziona ingressi tra primo indirizzo e successivi per i registri di lettura/scrittura

2.2. Unita di controllo

L'unita di controllo gestisce i segnali in entrata ed in uscita di controllo del datapath per il giusto funzionamento del progetto. E' stato modellizzato come una FSM di Moore a 11 stati.

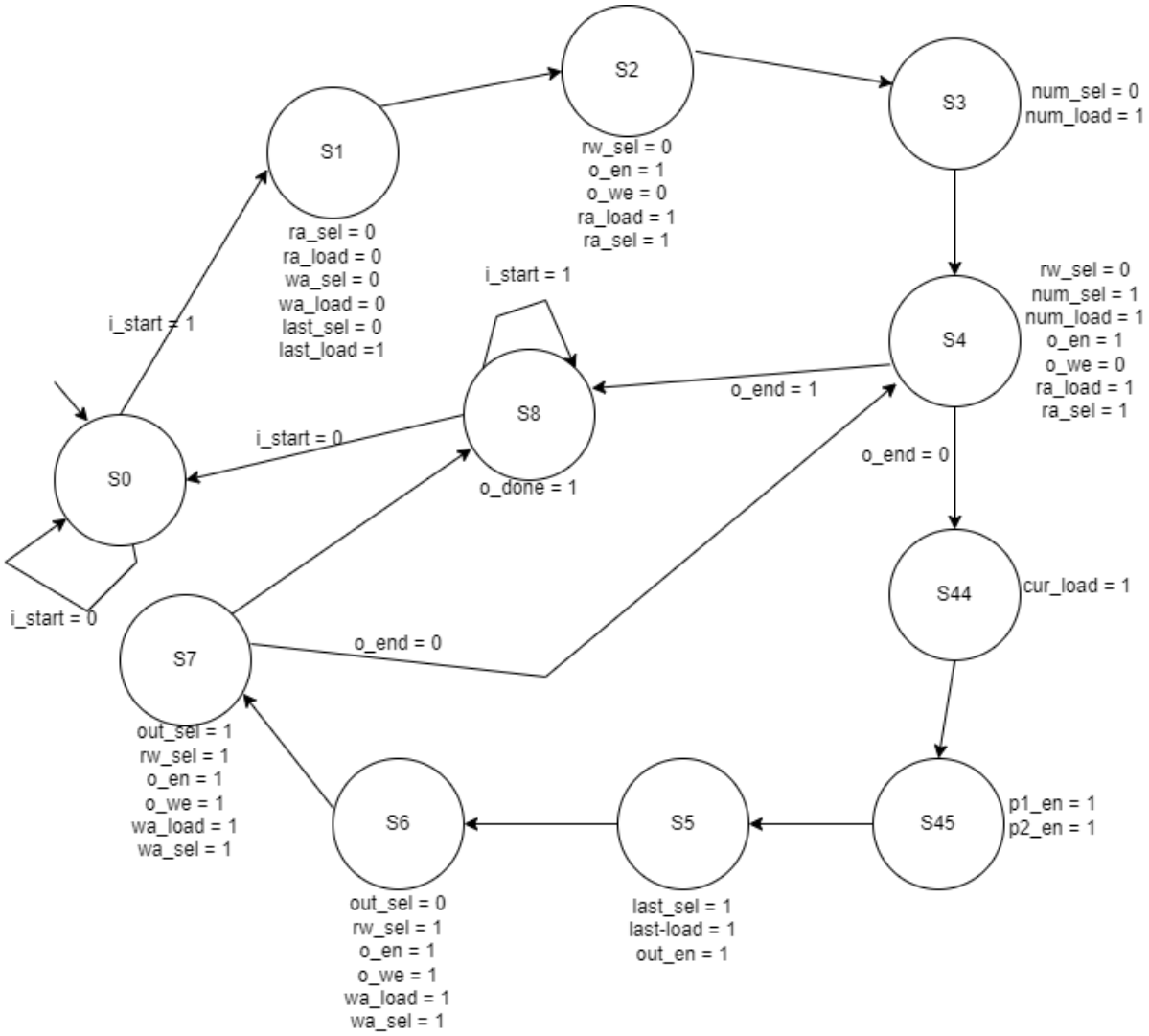

Figure 2.10: FSM

2.2.1. Lo stato S0

E' lo stato iniziale in cui la macchina attende il seganle di start.

2.2.2. Lo stato S1

Nello stato **S1** vengono inizializzati i valori iniziali ciascunamente in registri **read_address, write_address, last**

2.2.3. Lo stato S2

Abilita la memoria per la lettura, e incrementa indirizzo di lettura, il valore letto da i_data sarà pronto in istante di clock prossimo.

2.2.4. Lo stato S3

Abilita il registro **num** alla scrittura, il valore in ingresso sarà scritto, e sarà visibile a partire dall' istante di clock prossimo.

2.2.5. Lo stato S4

Nello stato **S4**, la memoria è abilitata per la lettura, il valore nel registro **num** sarà decrementato e quel di indirizzo di lettura incrementato. Inoltre, la transazione allo stato prossimo dipende dal segnale **o_end**, se qeusto è basso, **S4** andrà in **S44**, altrimenti in **S8**.

2.2.6. Lo stato S44

Abilita il registro **cur** alla scrittura, il valore in ingresso (i_data) sarà scritto, e sarà visibile a partire dall'istante di clock prossimo.

2.2.7. Lo stato S45

Abilita i registri **P1, P2** alla produzione e scrittura di semi-risultati.

2.2.8. Lo stato S5

Abilita il registro **produce_out** genera il risultato da trasmettere alla memoria e lo salva ricevendo dati dai registri **P1, P2**. Inoltre abilita il registro **last** alla scrittura e sposta il suo ingresso all'uscita di registro **cur** usando multiplexer.

2.2.9. Lo stato S6

Nello stato **S6** la memoria è abilitata per la scrittura, viene incrementato indirizzo di scrittura, la parte high del risultato generato verrà scritto in memoria.

2.2.10. Lo stato S7

Nello stato **S7** la memoria è abilitata per la scrittura, viene incrementato indirizzo di scrittura, la parte low del risultato generato verrà scritto in memoria. Inoltre la transazione allo stato prossimo dipende dal segnale **o_end**, se questo è basso, **S7** andrà in **S4**, altrimenti in **S8**.

2.2.11. Lo stato S8

Nello stato **S8**, il seganle **o_end** è alzato, e rimane in stato **S8** finchè il segnale **i_start** resta alto, altrimenti ritorna in stato **S0**.

3 | Risultati Dei Test

Per garantire il funzionamento corretto del componente sintetizzato, lo si sottopone ai vari test, in modo da cercare di coprire tutti i casi possibili che la macchina può terminare percorrendo diversi cammini. Di seguito viene riportato i report del post-sintesi e risultati dei test effettuati.

3.1. Sintesi

La post-sintesi è stata effettuata, il software utilizzato per questo progetto è Vivado, usando cme FPGA target il dispositivo xc7a200tfbg484-1 della Xilinx.

3.1.1. Utilization report

Dal utilization report si vede che il componente progettato è completamente sintetizzabile, senza la presenza di inferred latch.

Site Type	Used	Fixed	Available	Util%
Slice LUTs	92	0	134600	0.07
LUT as Logic	92	0	134600	0.07
LUT as Memory	0	0	46200	0.00
Slice Registers	86	0	269200	0.03
Register as Flip Flop	86	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

Figure 3.1: Utilization report

3.1.2. Timing report

Il Worst Negative Slack risulta circa 96ns, che indica per quanto tempo la macchina rimane al completamento del ciclo di clock nel caso peggiore. Che è anche al di sotto della richiesta di specifica, 100ns.

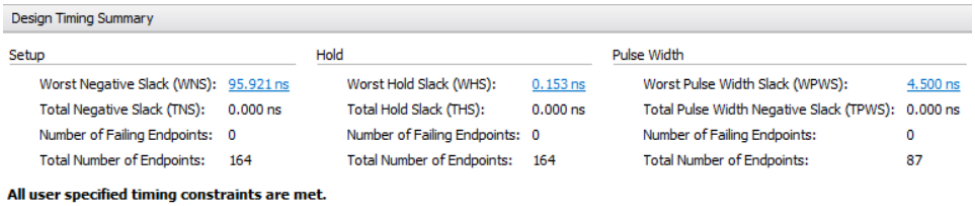

Figure 3.2: Timing report, sintesi, clock 100ns

3.2. Simulazioni

Tra vari test che sono stati effetuati, ci sono quelli per la verifica del comportamento normale e anche quelli per i casi limiti. Tutti i test fatto sul progetto ha avuto riscontro positivo sia in behavioral che in post-synthesis.

3.2.1. Test funzionamento

Il test fornisce sull'ingresso un intervallo ampio di parole cercando di coprire più casi possibile per la produzione di risultato.

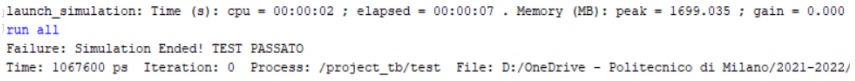

Figure 3.3: risultato

3.2.2. Test doppio uguale

In questo test bench sull'ingresso della memoria viene dato lo stesso input per due volte, l'intenzione è di quello di vedere se il componente progettato è in grado di elaborare sequenza uguali ottenendo risultati uguali.

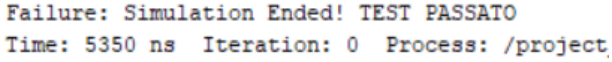

Figure 3.4: risultato

3.2.3. Test sequenza di lunghezza massima

Qui il progetto è sottoposto alla lunghezza massima di input, 255 parole.

```
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:08 . Memory (MB): peak = 1719.422 ; gain = 0.000
run all
Failure: Simulation Ended! TEST PASSATO
Time: 154050100 ps  Iteration: 0  Process: /project_tb/test  File: D:/OneDrive - Politecnico di Milano/2021-2022/
```

Figure 3.5: risultato

3.2.4. Test sequenza di lunghezza minima

Il progetto deve essere in grado di proccessare la seuquenza nulla, qui viene fornito 0 parola all'ingresso.

```
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:06 . Memory (MB): peak = 1726.328 ; gain = 0.000
run all
Failure: Simulation Ended! TEST PASSATO
Time: 1150100 ps  Iteration: 0  Process: /project_tb/test  File: D:/OneDrive - Politecnico di Milano/2021-2022
```

Figure 3.6: risultato

3.2.5. Test multi flussi con reset

Qui si vuole testare la capacita del progetto di elaborare diversi flussi di ingresso di lunghezza diversa, ognuna preceduto da un reset.

```
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:09 . Memory (MB): peak = 1719.422 ; gain = 0.000
run all
Failure: Simulation Ended! TEST PASSATO
Time: 11050100 ps  Iteration: 0  Process: /project_tb/test  File: D:/OneDrive - Politecnico di Milano/2021-2022
```

Figure 3.7: risultato

4 | Conclusione

Il componente ha superato tutti i test elencati in capitolo precedente secondo la richiesta. Qui nel conclusione si potrebbe aggiungere possibili ottimizzazione sul progetto già completato e funzionante. Un idea è questo: unire registri **cur, last** in un unico registro con 8+2 bit in uscita, in questo modo, ad ogni lettura di input basti che si fa un shift di 8 bit. In più, si poteva semplificare il processo di produzione di output in un unico processo.