Politecnico di Milano

PROVA FINALE
PROGETTO DI RETI LOGICHE

ANNO ACCADEMICO 2021/2022

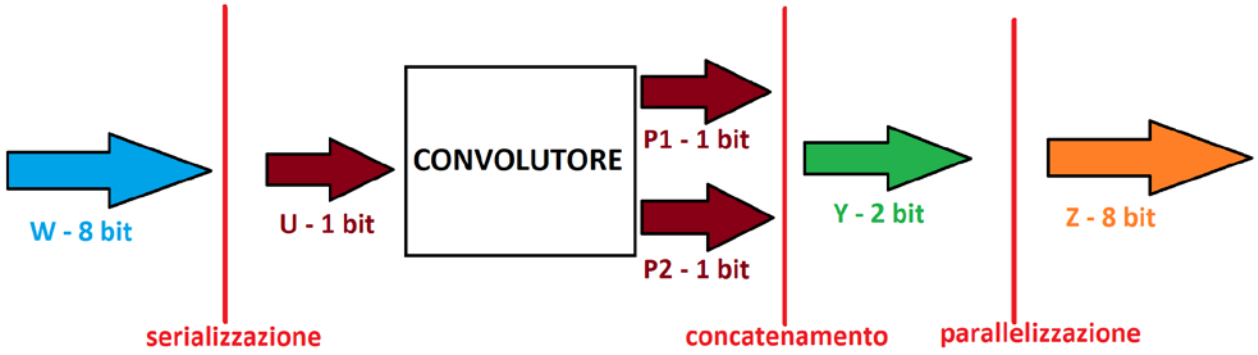
Angelo Attivissimo
10667094 - 935486

Sommario

INTRODUZIONE

1.1 Specifica di progetto

Si vuole descrivere, attraverso il linguaggio VHDL, un modulo hardware che riceve 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. Ciascuna delle parole di ingresso viene serializzata partendo dal suo MSB: in questo modo viene generato un flusso continuo U da 1 bit. Utilizzando un convolutore, ogni bit del flusso continuo U viene codificato con 2 bit, rispettivamente P_1 e P_2. Il flusso Y è ottenuto come concatenamento dei due bit di uscita, P_1 e P_2. La sequenza d'uscita Z è la parallelizzazione, su 8 bit, dei flussi continui Y_k.

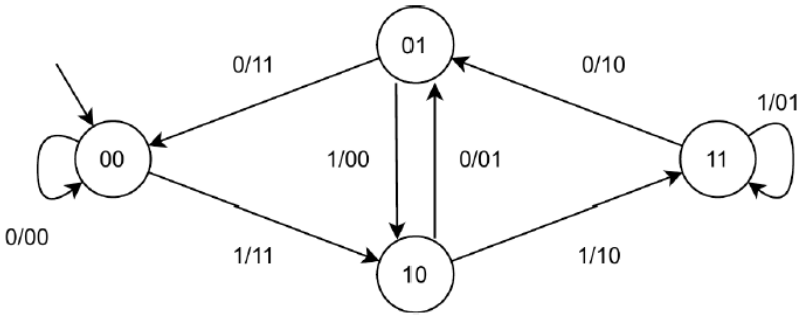


L'elaborazione del modulo partirà quando il segnale START in ingresso verrà portato a 1. Quando l'ultima sequenza Z è stata scritta in memoria, il segnale DONE viene portato a 1 per notificare la fine dell'elaborazione.

1.2 Convolutore

Il convolutore è una macchina sequenziale sincrona con un clock globale e un segnale di reset. Per ogni bit in ingresso vengono generati due bit in uscita. Ad ogni nuova elaborazione il convolutore viene portato nel suo stato di iniziale 00.

	0	1
00	00/00	10/11
01	00/11	10/00
10	01/01	11/10
11	01/10	11/01



1.3 Gestione della memoria

Il modulo da implementare deve leggere la sequenza da codificare da una memoria a 16 bit, con indirizzamento al Byte in cui è memorizzato; ogni singola parola di memoria è un byte. La quantità di

parole W da codificare è memorizzata nell'indirizzo 0; il primo byte della sequenza W (se esistente) è memorizzato all'indirizzo 1. Lo stream di uscita Z deve essere memorizzato a partire dall'indirizzo 1000.

INDIRIZZO MEMORIA	VALORE
0	2
1	162
2	75
[…]	[…]
1000	209
1001	205
1002	247
1003	210

In caso di una seconda elaborazione (START viene riportato a 1 dopo che DONE è stato alzato, ma non è stato mandato alcun segnale alto di RESET) oppure di un segnale alto di RESET mandato durante l'esecuzione, il modulo ricomincia l'elaborazione caricando la parola nella cella 0 e iniziando nuovamente a scrivere partendo dalla cella 1000, indipendentemente se in questa cella vi si era già scritto in precedenza. Inoltre anche il convolutore viene portato nel suo stato di iniziale 00.

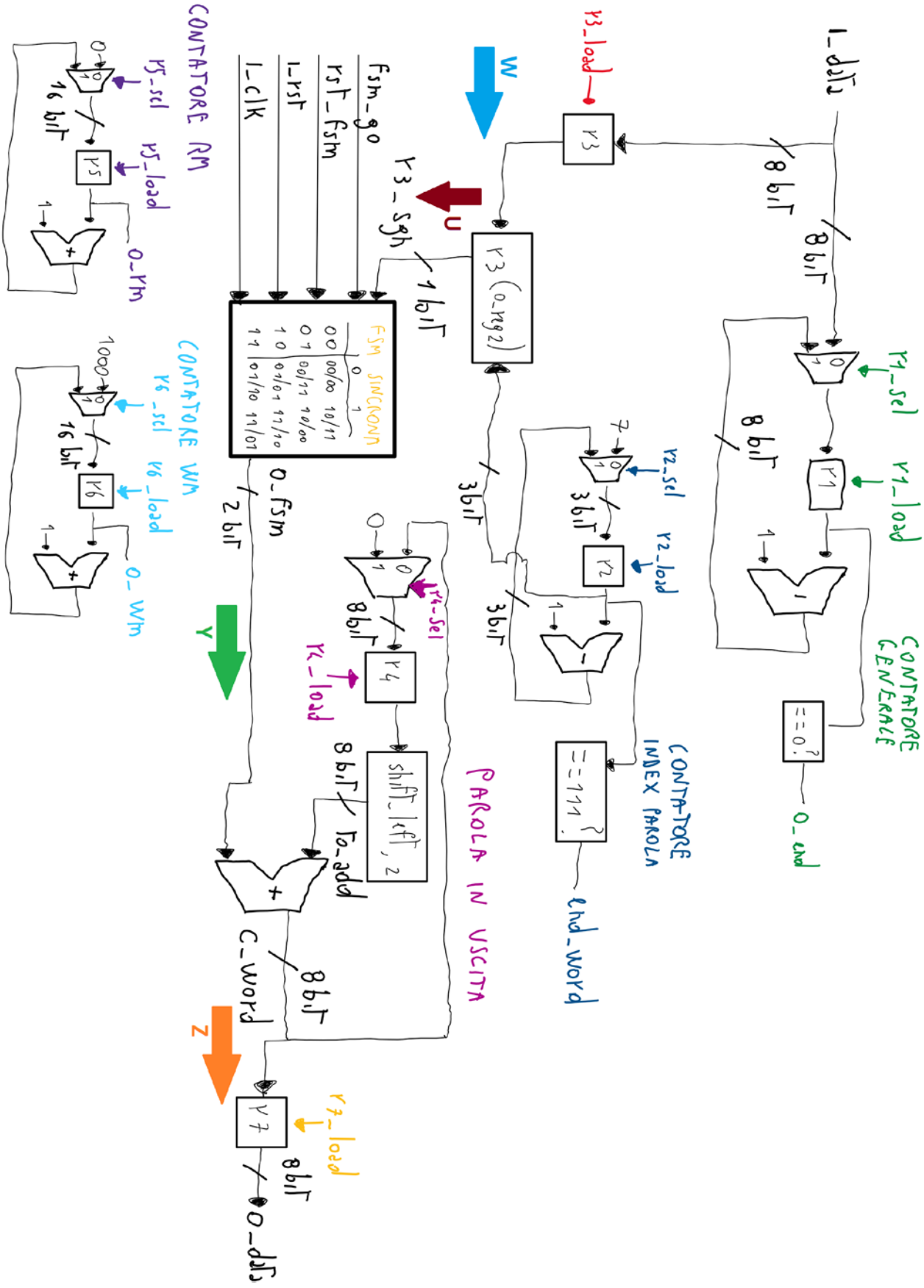
1.4 Implementazione

Il progetto utilizza una FPGA xc7a200tfbg484-1 con un tempo di clock di 100 ns.

Per la progettazione si è prima disegnato il datapath raffigurante le componenti per processare i dati secondo quanto le specifiche, in seguito è stata elaborata la macchina a stati per scandire il funzionamento del modulo nei vari cicli di clock. Appurato il funzionamento dei punti precedenti si è passati alla scrittura in linguaggio VHDL del comportamento del modulo.

ARCHITETTURA

2.1 Datapath



I segnali *i_clk* e *i_rst* influenzano l'intero datapath, ma per non appesantire troppo il disegno sono stati inseriti solo nella FSM sincrona.

2.1.1 Contatore generale

Il contatore generale carica in *r1* il numero di parole alla prima lettura da memoria, in seguito, ogni volta che una parola viene processata decrementa di 1 il valore positivo presente nel registro. Il segnale *o_end* è uguale a 1 quando il valore caricato in *r1* è uguale a 0.

2.1.2 Contatore index parola

Il contatore index parola è usato dal modulo per accedere alla posizione del vettore composto da 8 bit caricato in *r3* corrispondente al valore caricato in *r2*. In *r3* viene caricata la parola che deve essere codificata.

Il contatore index parola carica in *r2* il valore 7 e lo decrementa di 1 ogni volta che il bit di *o_reg3* corrispondente al valore caricato in *r2* è stato codificato dal convolutore. Il segnale *end_word* è uguale a 1 quando il valore caricato in *r2* è uguale a 7 (per maggiori chiarimenti vedi FSM).

2.1.3 Parola in uscita

Il modulo parola in uscita prende il valore presente in *r4*, ne esegue lo shift a sinistra per due posizioni, e lo somma al valore *o_fsm* in uscita dal convolutore. Questo valore viene ricaricato in *r4* e il ciclo viene ripetuto per quattro volte, fino a che non si è generata una nuova parola di 8 bit da caricare in *r7* (la parola Z).

La FSM sincrona, rappresentante il convolutore, riceve in ingresso il bit di *r3_sgn* (il flusso U), e restituisce in output un vettore *o_fsm* di 2 bit (il flusso Y). Questa FSM può funzionare (cambiare stato e generare output) se e solo se *fsm_go* è a 1. In caso *i_rst* o *rst_fsm* siano a 1 la FSM viene riportata allo stato iniziale 00, indipendentemente dal segnale *fsm_go*.

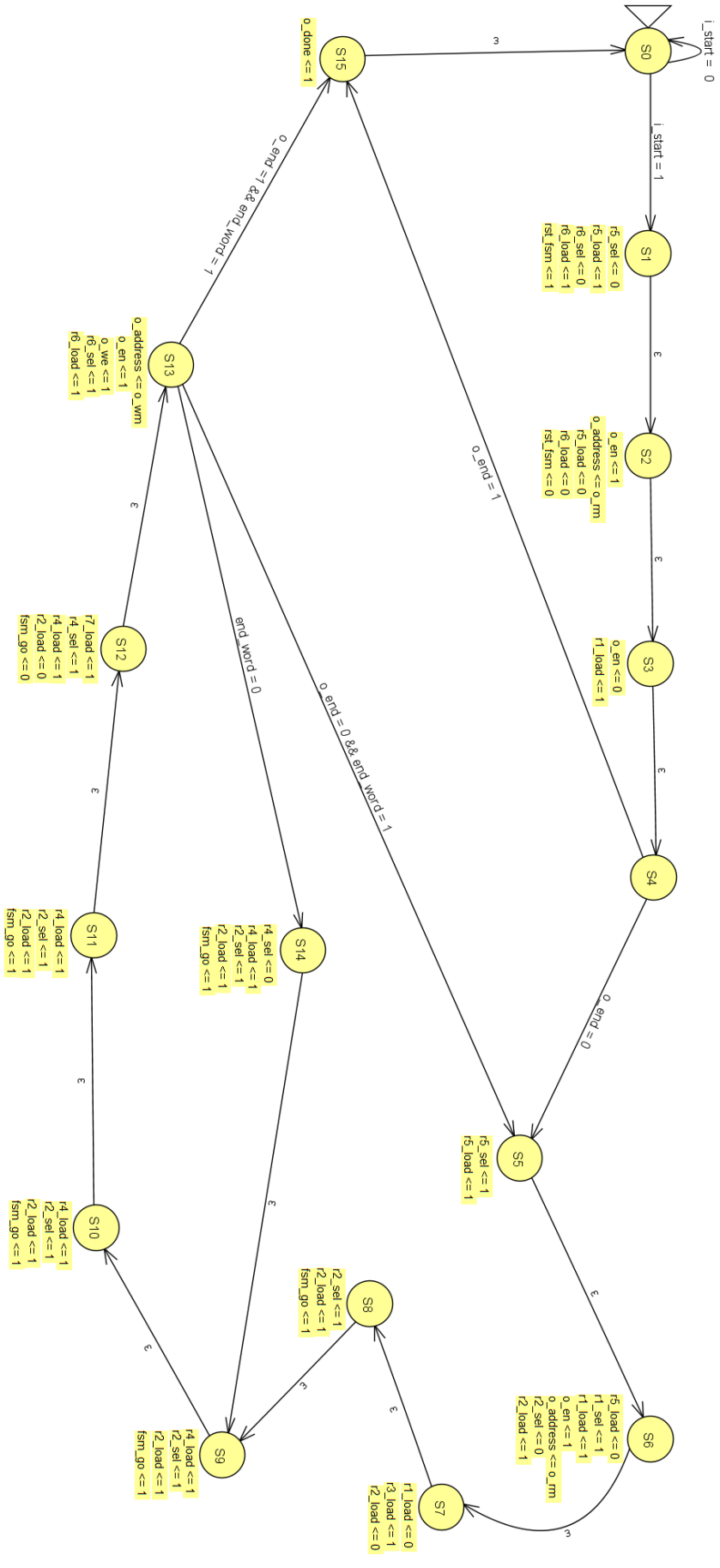
2.1.4 Contatore RM

Il contatore RM carica in *r5* il valore 0 e dopo avere eseguito una lettura in memoria ne incrementa di 1 il valore. Il segnale *o_rm* è un'identità con *o_reg5*, ma ne facilita la lettura.

2.1.5 Contatore WM

Il contatore RM carica in *r6* il valore 1000 e dopo avere eseguito una scrittura in memoria ne incrementa di 1 il valore. Il segnale *o_wm* è un'identità con *o_reg6*, ma ne facilita la lettura.

2.2 Macchina a stati



Per facilitare la lettura e non appesantire troppo il disegno, sono stati omessi tutti i segnali di RESET che portano, in qualsiasi momento, da qualunque stato a S0.

2.2.1 Descrizione degli stati

Tutti i segnali sono inizializzati a 0.

- S0: stato iniziale e anche stato di reset.
- S1: vengono inizializzati il *contatore RM* e il *contatore WM* e viene resettato il convolutore.
- S2: viene effettuato l'accesso all'indirizzo di memoria contenuto in *o_rm*.
- S3: viene inizializzato il *contatore generale* con il numero di parole da codificare.
- S4: è disponibile il numero di parole da codificare, se esso è diverso da 0 si prosegue verso S5 per l'inizio della codifica, altrimenti si passa a S15 per la conclusione dell'elaborazione.
- S5: il *contatore RM* passa al successivo indirizzo in memoria per la lettura.
- S6: il *contatore generale* decrementa il numero di parole da codificare rimanenti. Viene effettuato l'accesso all'indirizzo di memoria contenuto in *o_rm*. Viene inizializzato il *contatore index parola*.
- S7: il modulo *parola in uscita* carica la parola da codificare.
- S8: il *contatore index parola* passa al bit successivo da codificare della parola serializzata. Viene dato il segnale al convolutore per partire con la codifica. $^1/_4$ *step codifica*
- S9: il modulo *parola in uscita* carica la prima parallelizzazione del flusso Y in uscita. Il *contatore index parola* passa al bit successivo da codificare della parola serializzata. Viene dato il segnale al convolutore per continuare con la codifica. $^2/_4$ *step codifica*
- S10: come S9. $^3/_4$ *step codifica*
- S11: come S9. $^4/_4$ *step codifica*
- S12: Il modulo parola in uscita carica la parola Z in uscita pronta per essere scritta in memoria e resetta il registro usato per parallelizzare i flussi Y.
- S13: viene scritta in memoria all'indirizzo *o_wm* la parola Z in uscita. Il *contatore WM* passa al successivo indirizzo in memoria per la scrittura. Se le parole da codificare sono finite e la parola attualmente in codifica è terminata si passa a S15, se le parole da codificare non sono finite e la parola attualmente in codifica è terminata si passa a S5, se la parola attualmente in codifica non è terminata si passa a S14.
- S14: come S9. $^1/_4$ *step codifica (per la seconda semi parola)*
- S15: viene segnalata la fine dell'elaborazione.

RISULTATI SPERIMENTALI

3.1 Sintesi

- **Report di sintesi**

```
INFO: [Synth 8-802] inferred FSM for state register 'PS_reg' in module 'convolutore'
INFO: [Synth 8-802] inferred FSM for state register 'cur_state_reg' in module 'project_reti_logiche'
---------------------------------------------------------------------------------------------------
                   State |                     New Encoding |               Previous Encoding
---------------------------------------------------------------------------------------------------
                     s0 |                               00 |                               00
                     s2 |                               01 |                               10
                     s3 |                               10 |                               11
                     s1 |                               11 |                               01
---------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'PS_reg' using encoding 'sequential' in module 'convolutore'
---------------------------------------------------------------------------------------------------
                   State |                     New Encoding |               Previous Encoding
---------------------------------------------------------------------------------------------------
                     s0 |                 0000000000000001 |                               0000
                     s1 |                 0000000000000010 |                               0001
                     s2 |                 0000000000000100 |                               0010
                     s3 |                 0000000000001000 |                               0011
                     s4 |                 0000000000010000 |                               0100
                     s5 |                 0000000000100000 |                               0101
                     s6 |                 0000000001000000 |                               0110
                     s7 |                 0000000010000000 |                               0111
                     s8 |                 0000000100000000 |                               1000
                     s9 |                 0000001000000000 |                               1001
                    s10 |                 0000010000000000 |                               1010
                    s11 |                 0000100000000000 |                               1011
                    s12 |                 0001000000000000 |                               1100
                    s13 |                 0010000000000000 |                               1101
                    s14 |                 0100000000000000 |                               1110
                    s15 |                 1000000000000000 |                               1111
---------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'cur_state_reg' using encoding 'one-hot' in module 'project_reti_logiche'
---------------------------------------------------------------------------------
Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:15 ; elapsed = 00:00:16 . Memory (MB): peak = 1138.867 ; gain = 15.730
```

- **Slice Logic**

```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   82 |     0 |          0 |    134600 |  0.06 |
|   LUT as Logic          |   82 |     0 |          0 |    134600 |  0.06 |
|   LUT as Memory         |    0 |     0 |          0 |     46200 |  0.00 |
| Slice Registers         |   85 |     0 |          0 |    269200 |  0.03 |
|   Register as Flip Flop |   85 |     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 |
+-------------------------+------+-------+------------+-----------+-------+
```

- **Timing**

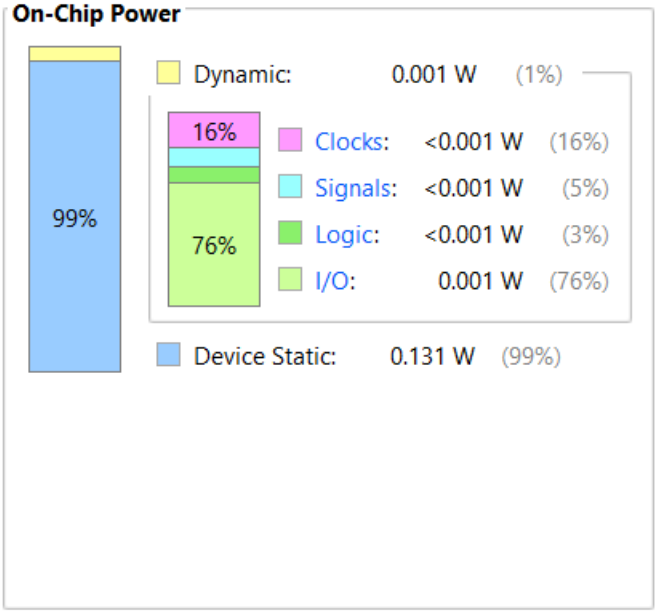
```
Timing Report

Slack (MET) :             96.775ns  (required time - arrival time)
  Source:                 FSM_onehot_cur_state_reg[6]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Destination:            FSM_onehot_cur_state_reg[0]/CE
                            (rising edge-triggered cell FDPE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        2.843ns  (logic 0.875ns (30.777%)  route 1.968ns (69.223%))
  Logic Levels:           2  (LUT2=1 LUT6=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 102.100 - 100.000 )
    Source Clock Delay      (SCD):    2.424ns
    Clock Pessimism Removal (CPR):    0.178ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

- **Power**

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

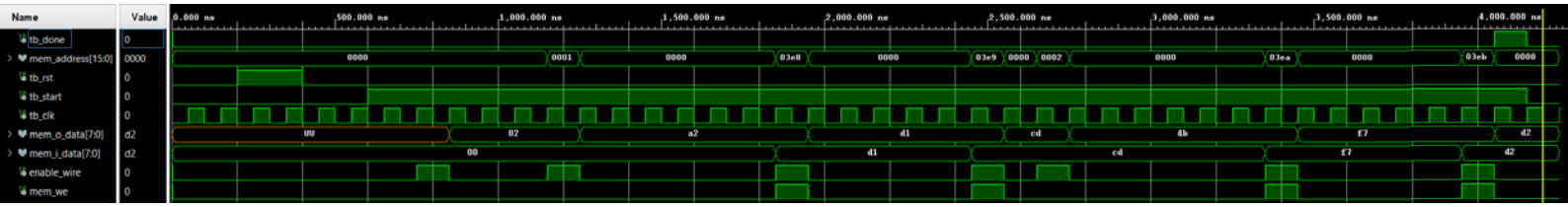
Total On-Chip Power:	**0.132 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.3°C**
Thermal Margin:	59.7°C (23.8 W)
Effective ϑJA:	2.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



3.2 Simulazioni

1) **tb_example**: è il test bench fornito come esempio. Simula il comportamento del modulo alla presenza di due parole in memoria.



2) **tb_zero_word**: questo test bench contiene nella cella di memoria all'indirizzo 0, la parola 0, ovvero non sono presenti parole da codificare.
3) **tb_seq_max**: questo test bench ha tutte le prime 256 celle della memoria riempite con parole da codificare, eccetto quella in posizione 0 che ne indica il numero.
4) **tb_all_three_examples**: questo test bench possiede tre diverse RAM, ognuna contenente le parole che si possono visionare nei tre esempi contenuti nelle specifiche. Il modulo deve gestire correttamente i dati e scrivere la parola in uscita nella RAM corrispondente.
5) **tb_double_processing**: questo test bench una volta finito il primo processo effettua un *double processing*, ovvero rialza il segnale di START e risottopone al modulo le stesse parole appena codificate, ma senza alzare il segnale di RESET.
6) **tb_reset**: questo test bench alza un segnale di RESET durante l'elaborazione.

Il modulo si comporta correttamente in tutti i casi sopra descritti sia in *Behavioral Simulation* che in *Post-Synthesis Functional Simulation*. È stato riportato lo schreenshot solo del primo test bench per non appesantire troppo il paragrafo.

CONCLUSIONI

4.1 Possibili ottimizzazioni

Non è escluso che l'architettura presentata nel capitolo 2 non sia ottimizzabile. Ad esempio i registri *r3* e *r7* del datapath possono essere rimossi, ottimizzando la complessità spaziale sulla FPGA, senza alterare il funzionamento del modulo, e in realtà così facendo si potrebbero eliminare anche due stati dalla macchina a stati, ottimizzando anche la complessità temporale. Tuttavia si è deciso di non modificare l'architettura del modulo per questioni di comodità: mantenendo *r3* non si ha una stretta dipendenza durante la codifica da *i_data*, in quanto la parola è stata clonata in *r3*, mentre mantenendo *r7* si evita di far oscillare *o_data* tra diversi valori, evitando così di far osservare ad un osservatore esterno al modulo, che si interfaccia ad esso tramite entity, i passaggi intermedi che portano alla composizione della parola Z in uscita.

4.2 Bibliografia e software utilizzati

- Introduzione al linguaggio VHDL – Brandolese
- Vivado
- VS Code - con estensione *VHDL for Professionals*
- JFLAP