POLITECNICO
MILANO 1863

Progetto Finale di Reti Logiche

Luca Bresciani
Matricola 937107
Codice Persona 10692758

10 Maggio 2022

Contents

1 Specifiche

1.1 Introduzione

Il progetto ha come scopo la realizzazione della codifica convoluzionale con tasso di trasmissione $\frac{1}{2}$. Questa codifica, come gran parte di quelle che hanno come risultato finale un numero maggiore di bit in uscita rispetto al numero di quelli in entrata, viene spesso utilizzata per aggiungere della ridondanza di dati all'interno di un canale trasmissivo. Così facendo infatti, benché la banda diminuisca e si hanno più errori nella sequenza ricevuta, se la codifica avviene in maniera corretta il maggior numero di simboli sbagliati ricevuti viene compensato dalla capacità di recupero degli errori del decodificatore.

1.2 Tasso di trasmissione $\frac{1}{2}$

Il modulo hardware che la specifica di questo progetto richiede di implementare è un modulo che riceve in ingresso una sequenza continua W di parole direttamente dalla memoria con cui esso si interfaccia. Il numero di parole presenti in memoria è specificato all' indirizzo '0' e il modulo dovrà quindi iniziare a processare la prima parola dall'indirizzo '1'. Queste parole, ognuna composta da 8 bit, vengono serializzate per creare un flusso entrante continuo da 1 bit. Su questo flusso viene applicato il codice di codifica convoluzionale con tasso di trasmissione $\frac{1}{2}$ e l' effetto che questo passaggio ha, non è altro che codificare il flusso continuo in entrata in due flussi continui, sempre da 1 bit, in uscita P_1k e P_2k.

La logica con cui questa codifica avviene può essere dedotta dalla figura 1. Nello specifico, chiamando U(k) il bit in entrata al codificatore convoluzionale all'istante k, P_1k è calcolato come segue: $P_1k = U(k) \oplus U(k-2)$. Mentre P_2k è calcolato come segue: $P_2k = U(k) \oplus U(k-1) \oplus U(k-2)$ Una volta calcolati i flussi in uscita questi vengono concatenati in maniera alternata per creare due parole da 8 bit ciascuna. Il concatenamento avviene come di seguito. P_1k all'istante t, P_2k all'istante t, P_1k all'istante t + 1, P_2k all'istante t + 1, P_1k all'istante t + 2, P_2k all' istante t + 2 e così via. Come compito finale il modulo deve scrivere in memoria le stringhe processate a partire dall' indirizzo 1000.

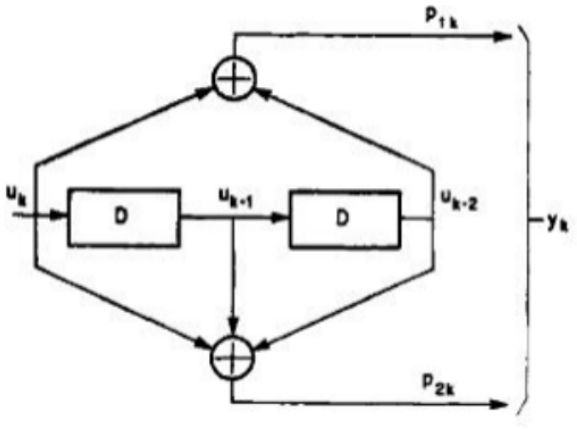

Figure 1: Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$.

1.3 Esempio

Di seguito viene fatto un esempio per migliorare la comprensione. La parola in entrata è il numero 112 che codificato in binario è 01110000. La successione dei bit sarà il flusso U(k) mentre i P_1k e P_2k risultanti sono mostrati in tabella 1.

T	0	1	2	3	4	5	6	7
U(k)	0	1	1	1	0	0	0	0
P_1k	0	1	1	0	1	1	0	0
P_2k	0	1	0	1	0	1	0	0

Table 1: Esempio codifica.

Risultato della concatenzaione = 0011100110110000. La prima parola sarà costituita dai primi 8 bit a partire da sinistra = 00111001 (57) mentre la seconda parola sarà costituita dai restanti bit = 10110000 (176).

2 Architettura

Per la realizzazione in VHDL si è deciso di usare due design sources. Il primo (project_reti_logiche) rappresenta la macchina a stati finiti che controlla i segnali in ingresso al secondo (datapath), che rappresenta il vero e proprio componente.

2.1 Datapath

In figura 2 viene riportata la rappresentazione del circuito implementato con i relativi segnali in ingresso e in uscita. Per la realizzazione del datapath si è deciso di dividere la parte di logica vera e propria dalle parti che si occupano di incrementare l'indirizzo di memoria al quale leggere e scrivere le parole. Per non complicare troppo la lettura sono stati omessi dalla rappresentazione il segnale di clock e il segnale di reset asincrono che tuttavia sono implementati correttamente nel codice VHDL.

2.1.1 Descrizione segnali

- fetch_load è il segnale per caricare il registro fetch_reg. Quando fetch_load è alto a '1' i_data viene scritto nel registro, mentre quando fetch_load è basso a '0' il registro si comporta da shift register.
- load_0_rst, load_1_rst, load_2_rst, load_3_rst, load_4_rst, load_5_rst, load_6_rst, load_7_rst sono i segnali di controllo ai multiplexer che permettono di resettare a '0' il contenuto dei registri che contengono i singoli bit.
- load_0, load_1, load_2, load_3, load_4, load_5, load_6, load_7 sono i segnali che permettono la scrittura all' interno dei registri che contengono i singoli bit.
- m1_sel, m2_sel, m3_sel sono i segnali di controllo ai tre multiplexer in uscita dai registri contenenti i bit e permettono di selezionare i giusti input per calcolare in maniera corretta P_1k e P_2k.
- out_reg_init è il segnale che permette di resettare a '0' il registro output_reg.

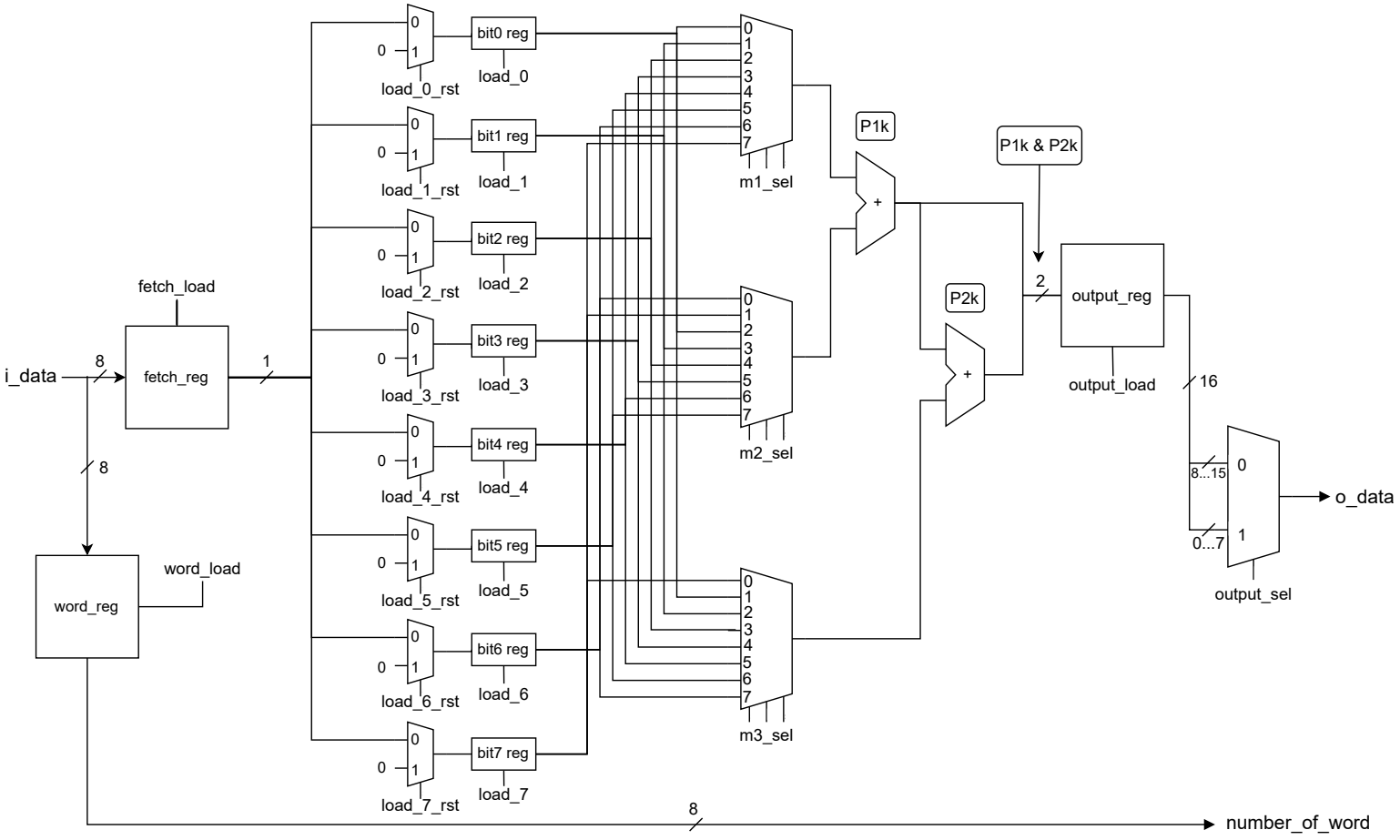

Figure 2: Datapath.

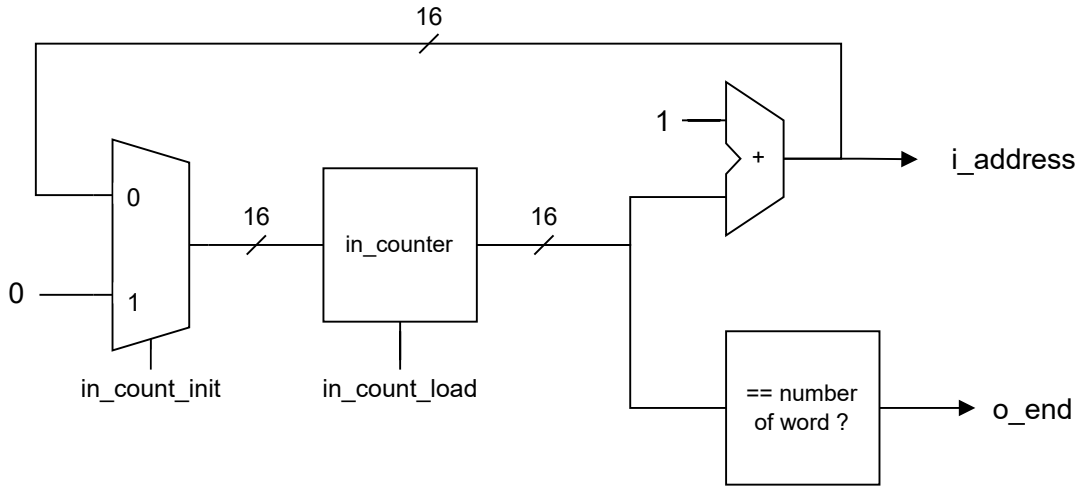

Figure 3: Contatore dell' indirizzo a cui leggere la parola.

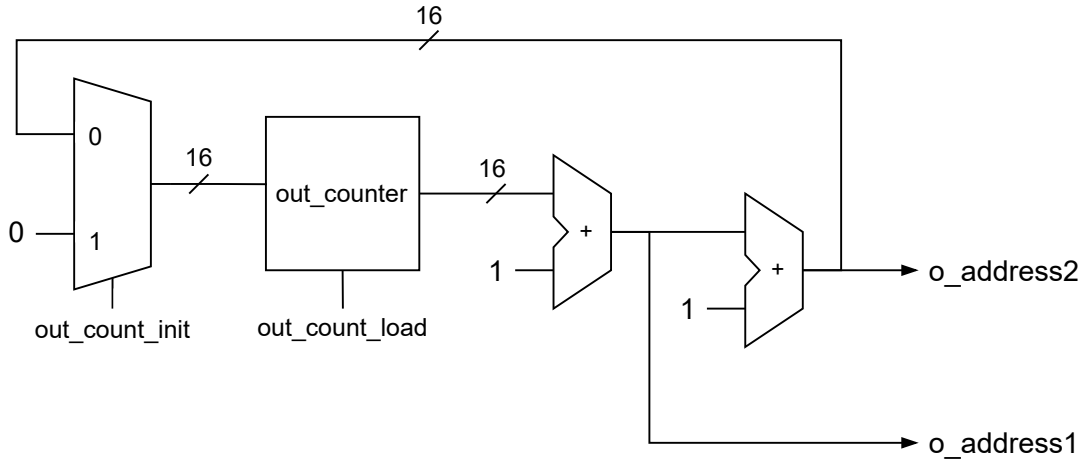

Figure 4: Contatore degli indirizzi a cui scrivere le parole.

- output_load è il segnale che permette la scrittura all'interno del registro output_reg che contiene la stringa da 16 bit ottenuta concatenando P_1k e P_2k.
- o_data è il segnale in uscita che andrà scritto in memoria.
- word_load è il segnale che permette scrittura nel registro word_reg.
- in_count_init è il segnale di comando al multiplexer che permette di resettare a '0' il registro in_counter_reg che si occupa di mantenere l'indirizzo di memoria a cui leggere le parole.
- in_count_load è il segnale che permette la scrittura nel registro in_count_reg.
- out_count_init è il segnale di comando al multiplexer che permette di resettare a '999' il registro out_counter_reg che si occupa di mantenere l'indirizzo di memoria in cui scrivere le parole.
- out_count_load è il segnale che permette la scrittura nel registro out_count_reg.
- o_end è il segnale che quando si trova alto a '1' indica la fine della lettura delle parole dalla memoria.
- i_address è il segnale che indica l'indirizzo di memoria al quale leggere le parole.
- o_address1 è il segnale che indica l'indirizzo di memoria al quale scrivere i primi 8 bit.
- o_address2 è il segnale che indica l'indirizzo di memoria al quale scrivere i secondi 8 bit.

2.2 Macchina a stati finiti

In figura 5 viene riportata la rappresentazione della macchina a stati finiti implementata con una descrizione dettagliata di ciascuno stato.

2.2.1 Reset

Questo stato rappresenta il reset del componente che riporta il modulo allo stato iniziale. La macchina rimane in questo stato fino a quando, come da specifica, il segnale di inizio i_start non viene portato a '1'.

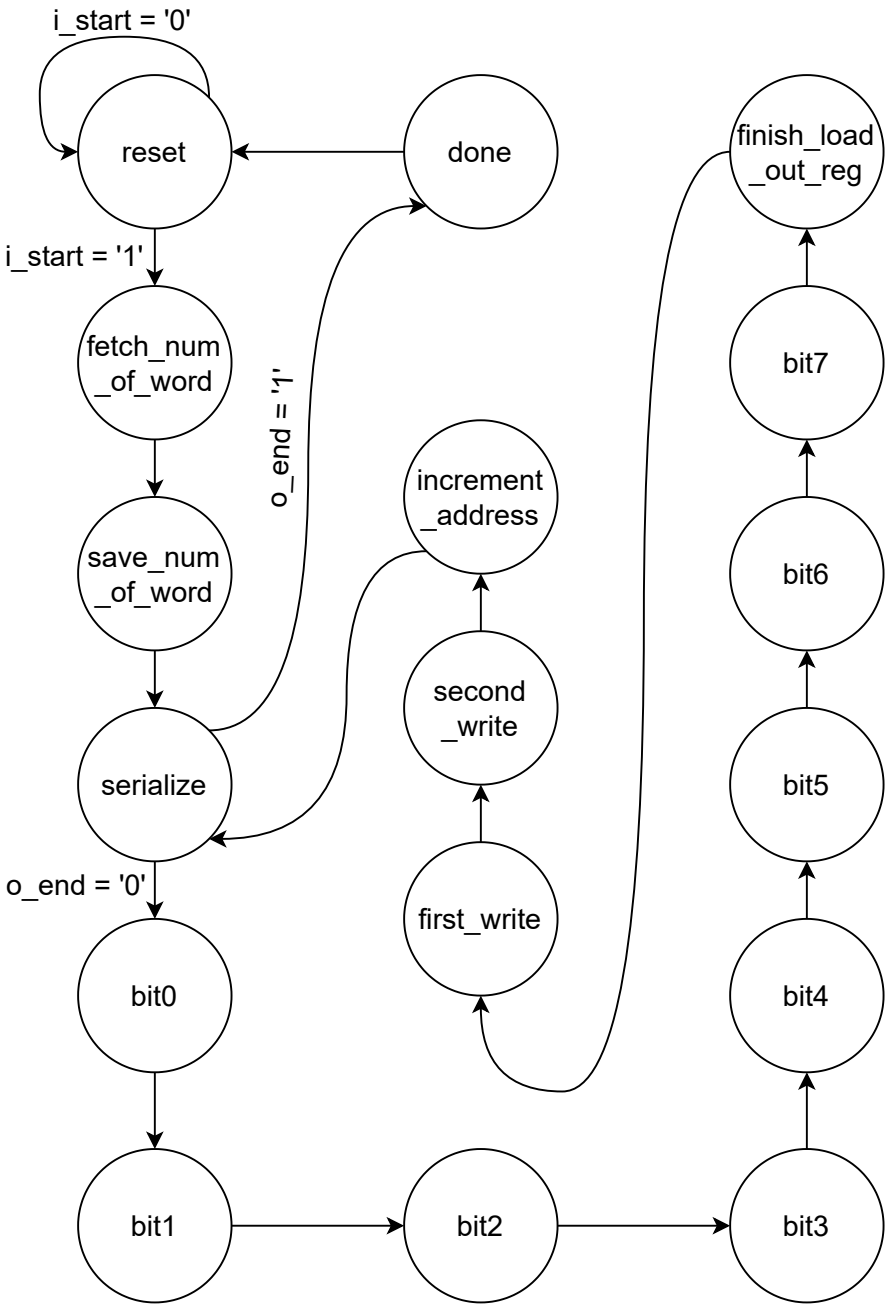

Figure 5: Macchina a stati finiti.

2.2.2 Fetch_num_of_word

In questo stato viene alzato il segnale o_en a '1' per interfacciarsi con la memoria e poter caricare il registro che si occupa di mantenere il numero di parole da processare.

2.2.3 Save_num_of_word

In questo stato viene alzato il segnale word_load a '1' per permettere di caricare il dato letto dalla memoria all'indirizzo '0' e salvare quindi il numero di parole da processare. All'interno di questo stato viene anche mantenuto alto il segnale o_en e viene assegnato il segnale i_address al segnale o_address. Questo per permettere nello stato successivo di poter leggere dalla memoria la prima parola contenuta all'indirizzo '1'.

2.2.4 Serialize

In questo stato viene alzato il segnale fetch_load a '1' per permettere di leggere dalla memoria la parola che si trova all'indirizzo dettato da o_address. Inoltre viene anche alzato a '1' il segnale in_count_load che permette la scrittura di i_address all'interno del registro che si occupa di mantenere l'indirizzo di memoria dal quale leggere le parole. Se il segnale o_end è alto la computazione termina e si passa allo stato DONE altrimenti la computazione continua e si passa allo stato BIT0.

2.2.5 Bit0

In questo stato viene alzato il segnale load_0 a '1' per permettere di caricare il registro che si occupa di mantenere il primo bit della parola processata.

2.2.6 Bit1 - Bit7

Per evitare una documentazione ripetitiva viene condensata la descrizione degli stati da BIT1 a BIT7 in questo sottoparagrafo. In questi stati infatti il segnale di load del rispettivo registro che si occupa di mantenere il bit della parola serializzata viene alzato a '1'. Inoltre viene alzato a '1' anche il segnale output_load per permettere di scrivere i dati processati nel registro che si occupa di mantenere la stringa da 16 bit risultante dalla codifica convoluzionale prima di scriverla effettivamente in memoria sotto forma di due

stringhe da 8 bit ciascuna. Per finire vengono impostati i valori corretti in entrata ai tre multiplexer che si occupano di selezionare il giusto segnale da propagare per calcolare in maniera corretta P_1k e P_2k.

2.2.7 Finish_load_out_reg

In questo stato vengono impostati i valori corretti in entrata ai tre multiplexer per calcolare gli ultimi due P_1k e P_2k e viene mantenuto alto il segnale output_load a '1' per permettere di scriverli all' interno del registro opportuno.

2.2.8 First_write

In questo stato vengono alzati a '1' i segnali o_en e o_we per permettere di interfacciarsi con la memoria in modalità scrittura. Inoltre viene impostato il segnale in ingresso al multiplexer che si occupa di selezionare i primi 8 bit o i secondi 8 bit della stringa da 16 bit calcolata in precedenza, in modo tale che vengano selezionati i primi 8. A o_address viene assegnato il valore di o_address1 che è il segnale in uscita dal datapath che rappresenta l'indirizzo di memoria dove scrivere la prima parola.

2.2.9 Second_write

In questo stato vengono mantenuti alti a '1' i segnali o_en e o_we. Il multiplexer in uscita dal modulo viene impostato per propagare i secondi 8 bit da scrivere in memoria e a o_address viene assegnato o_address2 che è il segnale in uscita dal datapath e rappresenta l'indirizzo di memoria dove scrivere la seconda parola. Inoltre viene alzato a '1' il segnale out_count_load per permettere la scrittura di o_address2 nel registro che si occupa di mantenere l' indirizzo al quale scrivere le parole in uscita.

2.2.10 Increment_address

In questo stato viene mantenuto alto a '1' il segnale o_en per permettere la lettura del dato nello stato successivo e viene assegnato i_address a o_address.

2.2.11 Done

Questo stato indica la fine della computazione. Il segnale o_done è portato a '1' e vengono inizializzati di nuovo tutti i registri che contengono i singoli bit, i registri dei contatori e il registro che mantiene la stringa da 16 bit computata in quanto, come da specifica, tra una computazione e un'altra non viene inviato un segnale di reset asincrono. Dopo questo stato la macchina è pronta per ripartire con una nuova codifica.

3 Risultati sperimentali

3.1 Sintesi

Il modulo è stato testato sulla versione di Vivado 2021.2 utilizzando come FPGA target xc7a200tfbg484-1. La sintesi ha prodotto i seguenti risultati.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	97	0	0	134600	0.07
LUT as Logic	97	0	0	134600	0.07
LUT as Memory	0	0	0	46200	0.00
Slice Registers	88	0	0	269200	0.03
Register as Flip Flop	88	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

Figure 6: Report utilization.

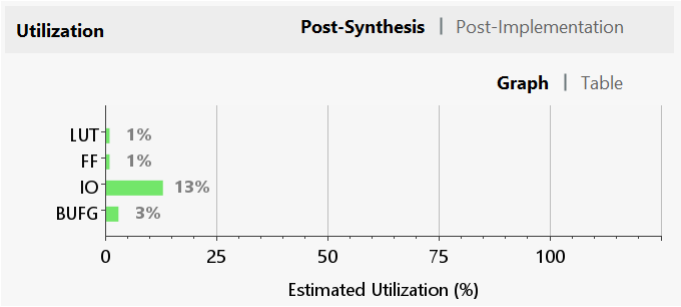

Figure 7: Percentuale di utilizzo.

```
Timing Report

Slack (MET) :             95.990ns  (required time - arrival time)
  Source:                 DATAPATH0/o_7_m1_reg/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Destination:            DATAPATH0/out_reg_reg[0]/D
                            (rising edge-triggered cell FDCE 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:        3.859ns  (logic 0.999ns (25.888%)  route 2.860ns (74.112%))
  Logic Levels:           3  (LUT5=1 LUT6=2)
  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
```

Figure 8: Report timing.

3.2 Simulazione

Il componente è stato simulato utilizzando diversi test benches. Di seguito viene riportata nella figura 9 la forma d'onda della simulazione ottenuta con il test bech fornitoci con il progetto che ha prodotto risultati positivi sia in behavioural simulation che in post synthesis functional simulation. Una descrizione degli altri test benches provati verrà fornita nel paragrafo successivo. Per semplicità di lettura i valori sono scritti in forma decimale e oltre alle forme d'onda dei segnali del test bench sono presenti anche quelle di altri segnali interessanti. Oltre alla forma d'onda completa vengono riportati anche due istanti particolari, ovvero quello in cui viene letta la prima parola in figura 10 e quello in cui viene letta la seconda in figura 11. In questo modo è possibile osservare tutti i valori.

3.3 Test Benches

Partendo dal test bench fornitoci ne sono stati scritti altri modificando i valori contenuti in memoria e i valori attesi in seguito alla codifica. Tuttavia una volta superato il primo test bench si è notato che, anche i successivi, i quali prendevano spunto da questo, venivano superati senza alcun problema. Di interesse sicuramente maggiore sono i test benches che testano i casi limite e alcune configurazioni particolari della memoria.

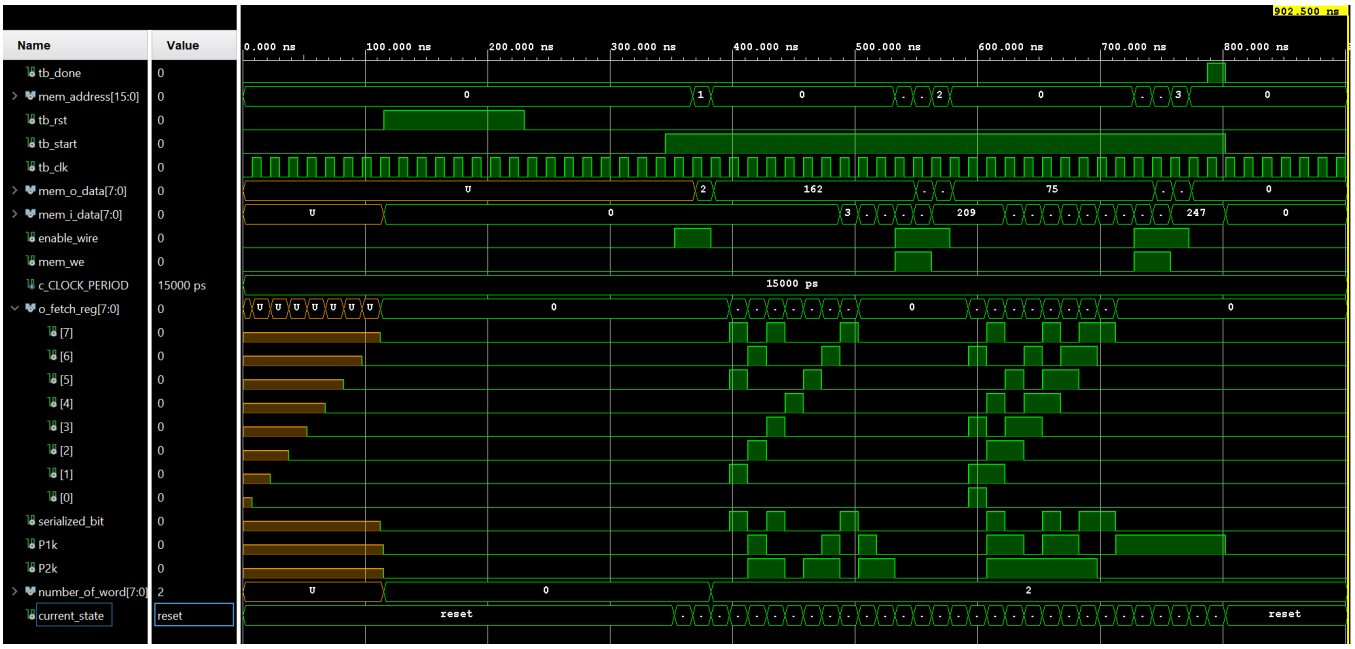

Figure 9: Forma d'onda completa testbench project_tb.

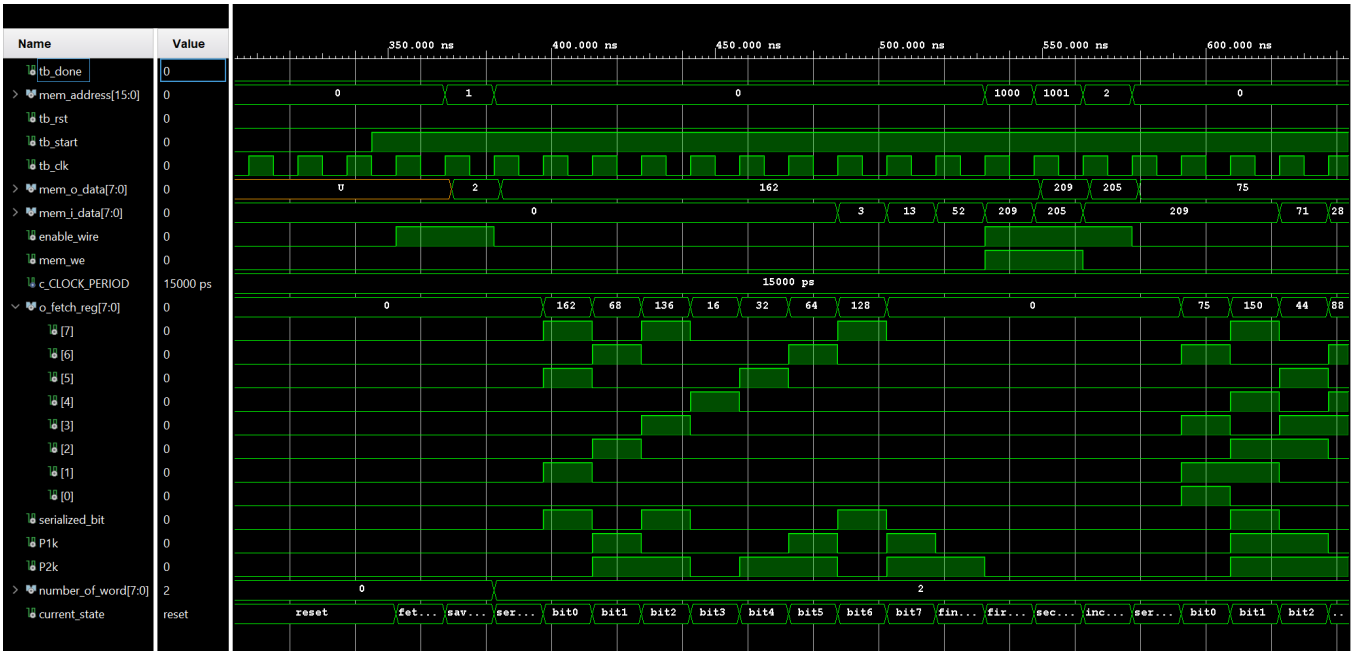

Figure 10: Forma d'onda zoomata sulla computazione di 162 da project_tb.

Figure 11: Forma d'onda zoomata sulla computazione di 75 da project_tb.

3.3.1 Casi limite

Sono stati testati e passati sia in behavioural simulation che in post synthesis functional simulation i seguenti casi di test:

- seq_min_tb. Memoria con 0 parole da processare (sequenza minima). Figura 12.
- seq_max_tb. Memoria con 255 parole da processare (sequenza massima).
- reset_tb. Al modulo arriva un segnale di reset asincrono durante la computazione. Figura 13.
- multi_reset_tb. Al modulo arrivano più segnali di reset asincorni durante la computazione. Figura 14

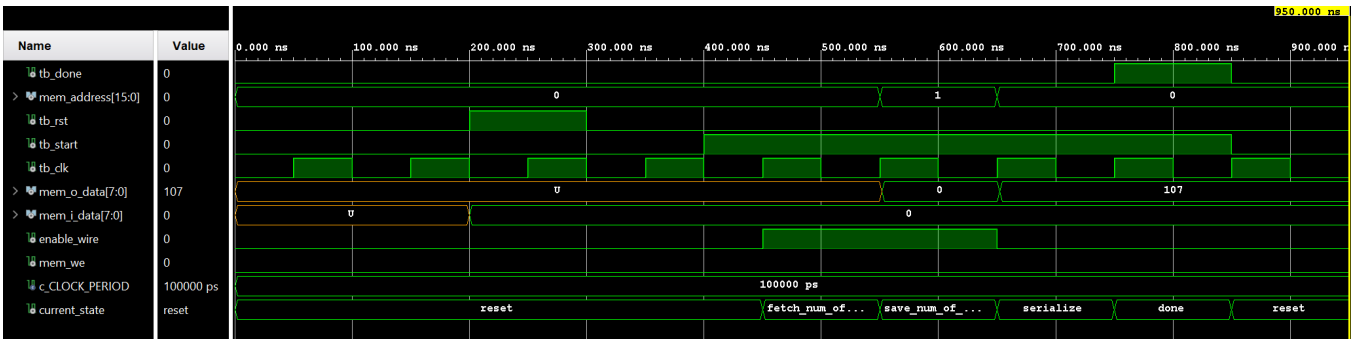

Figure 12: Forma d'onda completa con 0 parole da processare.

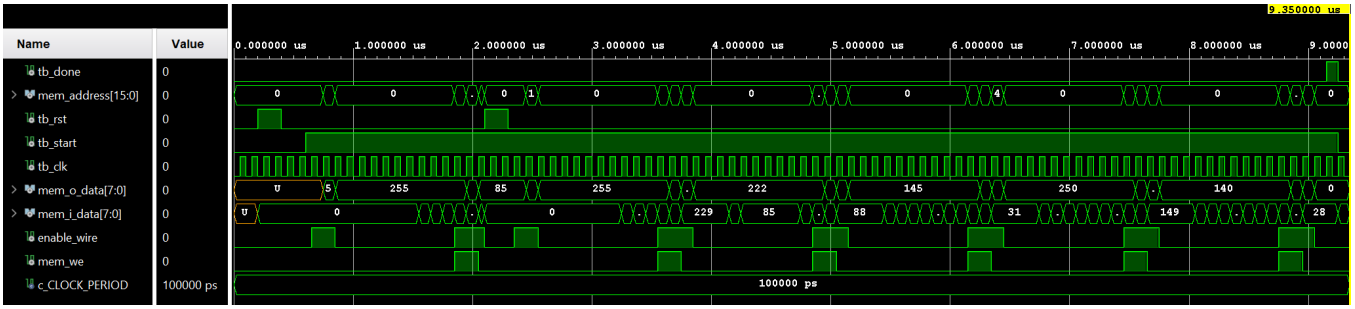

Figure 13: Forma d'onda completa con un reset asincrono.

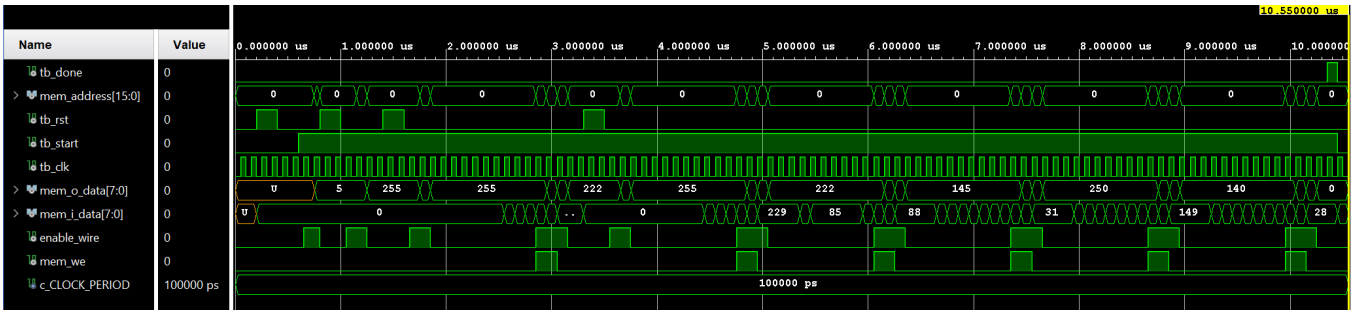

Figure 14: Forma d'onda completa con più reset asincroni.

4 Conclusioni

Il modulo ha superato senza problemi tutti i test a cui è stato sottoposto e funziona correttamente sia dal punto di vista comportamentale che dopo essere stato sintetizzato. Anche dopo il processo di sintesi non vengono inferiti latch nel circuito e la computazione viene svolta in un tempo nettamente al di sotto del vincolo imposto dalla specifica. Una possibile ottimizzazione potrebbe essere condensare la logica del calcolo degli indirizzi a cui scrivere le parole nella logica del calcolo dell'indirizzo a cui leggere la parola per avere meno segnali interni nel datapath. Tuttavia anche avere due diversi contatori non è un problema per questo progetto in quanto la percentuale di utilizzo dell'FPGA è di molto al di sotto delle sue capacità massime. Per lo svolgimento di questo progetto si è iniziato per prima cosa a ragionare sul design del datapath cercando di migliorarlo quanto il più possibile in modo tale che una volta terminata questa fase la scrittura del codice della macchina a stati fosse quasi automatica.