POLITECNICO
MILANO 1863

Progetto di Reti Logiche

Autore
Christian Confalonieri

Docente
William Fornaciari

2021/2022

Indice

1 Introduzione

1.1 Obiettivo

Data una sequenza di N parole, ognuna di 8 bit, e la dimensione N, lo scopo del progetto è quello di implementare un modulo hardware, descritto in VHDL, che si interfacci con una memoria e che fornisca le 2N parole calcolate tramite un codificatore convoluzionale.

1.2 Specifiche

Il modulo riceve in ingresso una sequenza continua di W parole, ognuna di 8 bit, e restituisce in uscita una sequenza continua di Z parole, ognuna di 8 bit. Ciascuna delle parole in ingresso viene serializzata, in questo modo viene generato un flusso continuo U da 1 bit. Su questo flusso viene applicato il codice convoluzionale $\frac{1}{2}$ (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 in uscita.
La sequenza d'uscita Z è la parallelizzazione, su 8 bit, del flusso continuo Y.

1.2.1 Codificatore Convoluzionale

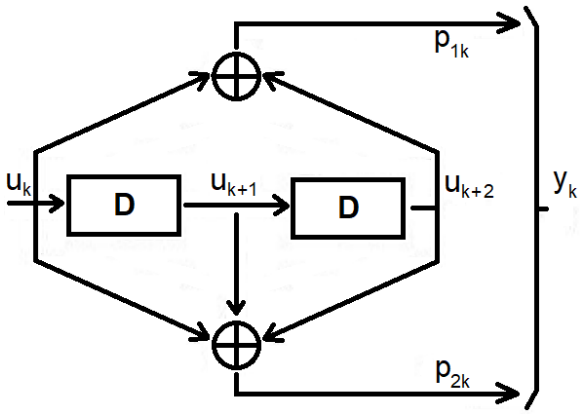


Figura 1: Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$

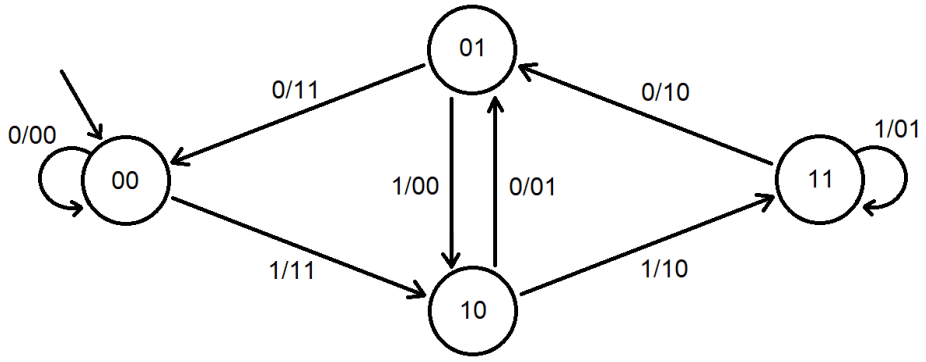


Figura 2: Macchina a stati finiti del codificatore convoluzionale

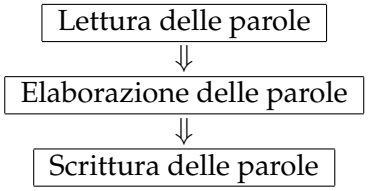
Un esempio di funzionamento è il seguente, dove il primo bit a sinistra, il più significativo del byte, è il primo bit seriale da processare:
- Byte in ingresso = 10100010 (viene serializzato come 1 al tempo t, 0 al tempo t+1, 1 al tempo t+2, 0 al tempo t+3, 0 al tempo t+4, 0 al tempo t+5, 1 al tempo t+6 e 0 al tempo t+7)
Applicando l'algoritmo convoluzionale si ottiene la seguente serie di coppie di bit:
T 0 1 2 3 4 5 6 7
Uk 1 0 1 0 0 0 1 0
P_{1k} 1 0 0 0 1 0 1 0
P_{2k} 1 1 0 1 1 0 1 1
Il concatenamento dei valori p_{1k} e p_{2k} per produrre Z segue lo schema: p_{1k} al tempo t, p_{2k} al tempo t, p_{1k} al tempo t+1, p_{2k} al tempo t+1, p_{1k} al tempo t+2, p_{2k} al tempo t+2, ... cioè
Z: 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0 1
- Byte in uscita = 11010001 e 11001101
Nota: ogni byte in ingresso ne genera due in uscita.

1.2.2 Struttura della Memoria

Il modulo da implementare deve leggere la sequenza da codificare da una memoria con indirizzamento al byte; ogni singola parola di memoria è un byte. La sequenza di byte è trasformata nella sequenza di bit U da elaborare.
Le celle di memoria sono suddivise in questo modo:

- la cella (0) contiene il numero di parole da leggere;
- le celle (1),...,(N) (max: N=255) contengono le parole da leggere;
- le celle (1000),(1001),...,(M) (max: M=1509) conterranno le parole che verranno calcolate dal componente.

	Memoria
0	Numero di parole
1	Input 1
2	Input 2
...	...
255	Input 255
...	...
1000	Output 1a
1001	Output 1b
1002	Output 2a
1003	Output 2b
...	...
1508	Output 255a
1509	Output 255b



Nota: i risultati ottenuti dall'applicazione dell'algoritmo convoluzionale (output) verranno scritti in memoria al termine della computazione di ogni parola (input).

1.2.3 Interfaccia del Componente

```
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:

- *i_clk* è il segnale di *clock* in ingresso generato dal Test Bench;
- *i_rst* è il segnale di *reset* che inizializza la macchina in modo che sia pronta per ricevere il primo segnale di *start*;
- *i_start* è il segnale di *start* generato dal Test Bench;
- *i_data* è il segnale, composto da un vettore di bit, che arriva dalla memoria in seguito ad una richiesta di lettura;
- *o_address* è il segnale, composto da un vettore di bit, in uscita che manda l'indirizzo alla memoria;
- *o_done* è il segnale in uscita che comunica la fine dell'elaborazione;
- *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 scrivere in memoria;
  - '0' per leggere da memoria;
- *o_data* è il segnale, composto da un vettore di bit, in uscita dal componente verso la memoria.

1.2.4 Panoramica e Utilizzo

Il modulo partirà nell'elaborazione quando un segnale *start* in ingresso verrà portato a 1. Il segnale *start* rimarrà alto fino a che il segnale *done* non verrà alzato; al termine della computazione, e una volta scritto il risultato in memoria, il modulo da progettare deve portare a 1 il segnale *done* che notifica la fine dell'elaborazione. Il segnale *done* deve rimanere alto fino a che il segnale *start* non è riportato a 0.
Un nuovo segnale *start* non può essere mandato fintanto che *done* non è stato riportato

basso. Se a questo punto viene rialzato il segnale *start* il modulo deve ripartire con la fase di codifica.

Il modulo è stato progettato per poter codificare più flussi uno dopo l'altro. Ad ogni nuova elaborazione, quando *start* viene riportato alto a seguito del *done* basso, il convolutore viene portato nel suo stato iniziale 00, che è anche quello di *reset*. La quantità di parole da codificare sarà sempre memorizzata all'indirizzo 0 e l'uscita deve essere sempre memorizzata a partire dall'indirizzo 1000.

Il modulo è stato progettato considerando che antecedentemente alla prima codifica verrà sempre dato il *reset*. Invece, come descritto nel protocollo precedente, una seconda elaborazione non dovrà attendere il *reset* del modulo ma solo la terminazione.

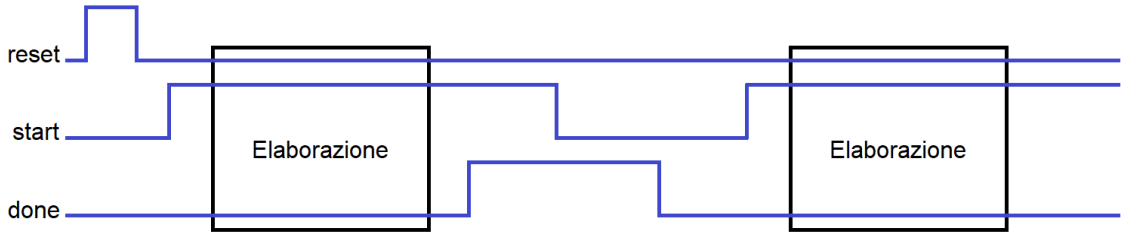


Figura 3: Segnali di reset, start e done durante la computazione (il segnale di reset può essere riportato a 1 ma non impatterà elaborazioni successive)

2 Architettura

Il modulo è stato diviso in due architetture:

- datapath: descrive l'insieme di registri e unità funzionali necessarie a implementare ogni singola istruzione;
- macchina a stati finiti: controlla il funzionamento del datapath.

2.1 Datapath

Il datapath è costituito da:

- 5 registri (*reg1*,*reg2*,*reg3*,*reg4*,*reg5*) e 4 flip flop (*ff1*,*ff2*,*ff3*,*ff4*);
- 8 multiplexer;
- 3 sommatori e 2 sottrattori;
- 1 "convolutore" (il cui funzionamento verrà spiegato in seguito).

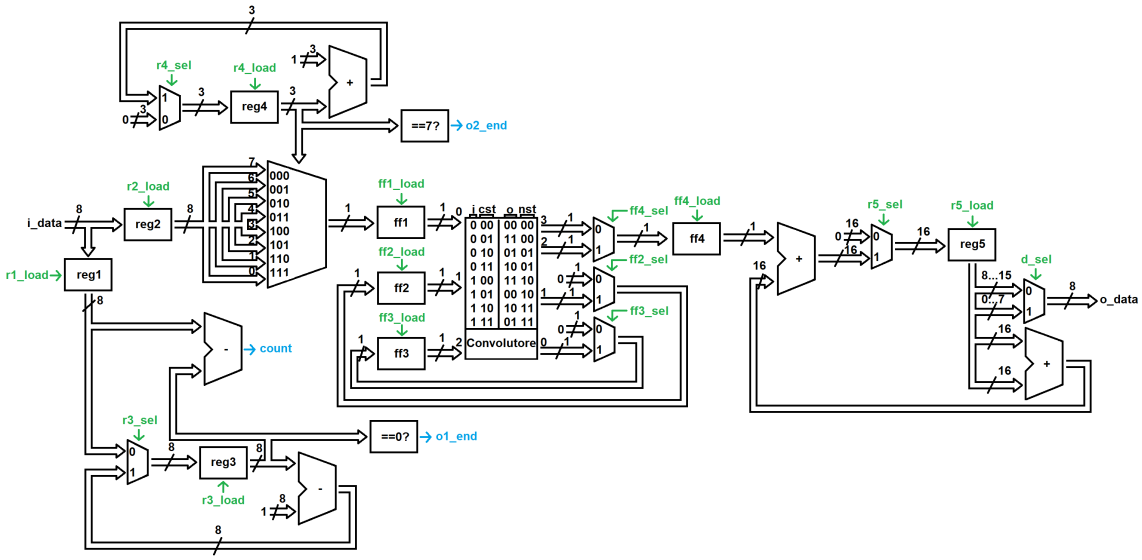


Figura 4: Datapath del modulo implementato

2.1.1 Funzionamento del Datapath

La parte collocata in basso a sinistra del datapath (figura 5), tramite *i_data*, salva in *reg1* il valore contenuto nella cella di memoria 0. Questo valore verrà caricato in *reg3* a inizio computazione.
Per ogni parola letta il valore contenuto in *reg3* verrà decrementato di 1 tramite il sottrattore.
Vengono quindi prodotte due uscite che saranno lette dalla macchina a stati finiti:

- *count* = *reg1* - *reg3* [8 bit]: verrà utilizzato per consentire l'avanzamento degli indirizzi di memoria (con reg1 e reg3, in questo caso, si intendono i valori contenuti in essi);
- *o1_end* [1 bit]: verrà posto a 1 dopo aver letto e computato tutte le parole in ingresso.

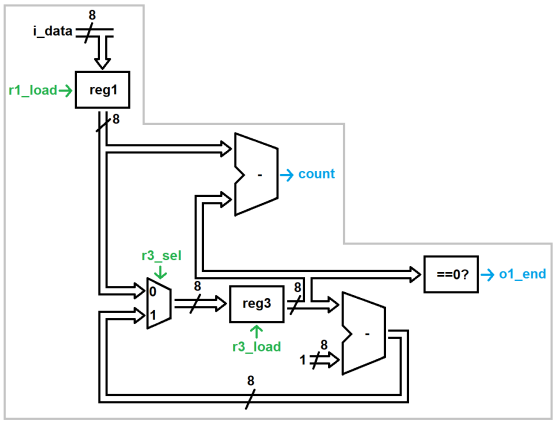


Figura 5: Parte in basso a sinistra del datapath

La parte collocata in alto a sinistra del datapath (figura 6), tramite *i_data*, salva in *reg2* il valore contenuto nella cella di memoria *count* (che alla prima computazione sarà uguale a 1).
Dopo aver caricato tale valore in *reg2*, verrà caricato il valore 0 in *reg4*.
Ad ogni bit computato dal modulo il valore contenuto in *reg4* verrà incrementato di 1 tramite il sommatore.
Viene prodotta un'uscita che sarà letta dalla macchina a stati finiti:

- *o2_end* [1 bit]: verrà posto a 1 dopo aver letto e computato tutti i bit della parola corrente in ingresso.

Il valore contenuto in *reg4* viene anche utilizzato come input per il controllo del multiplexer che sarà così in grado di scorrere i bit della parola.
Ad ogni iterazione il bit corrente viene salvato in *ff1*.

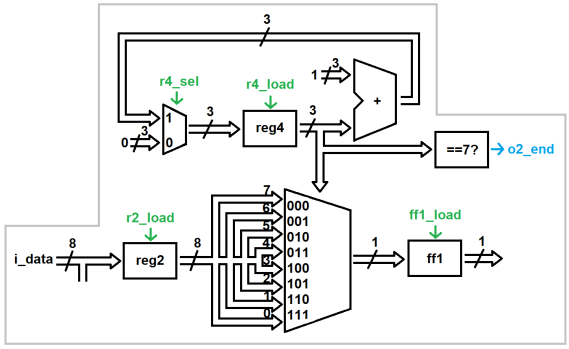


Figura 6: Parte in alto a sinistra del datapath

La parte centrale del datapath (figura 7) è quella che gestisce l'algoritmo di convoluzione.
Il "convolutore" raffigurato riceve 3 bit in ingresso:

- il primo salvato in *ff1*: è il bit da computare tramite l'algoritmo convoluzionale;
- il secondo salvato in *ff2*: è il primo bit dello stato corrente;
- il terzo salvato in *ff3*: è il secondo bit dello stato corrente.

Seguendo la tabella mostrata all'interno del convolutore vengono prodotti 4 bit in uscita:

- il primo: è il primo bit ottenuto tramite l'algoritmo convoluzionale che verrà salvato in *ff4* e successivamente gestito;
- il secondo: è il secondo bit ottenuto tramite l'algoritmo convoluzionale che verrà salvato in *ff4* e successivamente gestito;
- il terzo: è il primo bit dello stato prossimo; una volta salvato in *ff2* sarà il primo bit dello stato corrente;
- il quarto: è il secondo bit dello stato prossimo; una volta salvato in *ff3* sarà il secondo bit dello stato corrente.

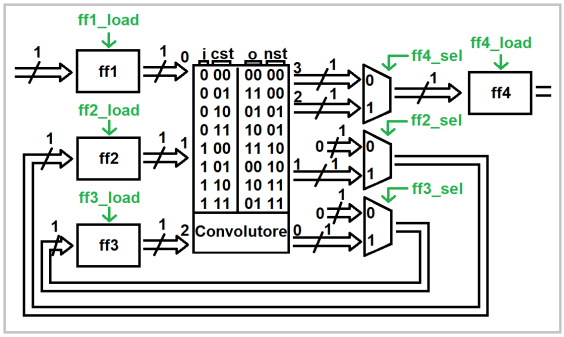


Figura 7: Parte centrale del datapath

La parte a destra del datapath (figura 8) è quella che gestisce l'aggiunta del nuovo bit alla parola risultante dalla computazione.
All'inizio del calcolo di ogni nuova parola il valore in *reg5* viene inizializzato a 0.
Ogni bit salvato in *ff4* verrà posto nella posizione meno significativa della parola salvata in *reg5* (la parola verrà spostata a sinistra di una posizione e il bit verrà sommato).
Quando la parola in ingresso sarà stata completamente computata, e quindi *reg5* conterrà il risultato finale, a 16 bit, della computazione, verrà diviso in 2 parole che verranno caricate nei rispettivi indirizzi:

- gli 8 bit più significativi verranno caricati all'indirizzo: 998 + *count* + *count* (alla prima computazione sarà uguale a 1000);
- gli 8 bit meno significativi verranno caricati all'indirizzo: 999 + *count* + *count* (alla prima computazione sarà uguale a 1001).

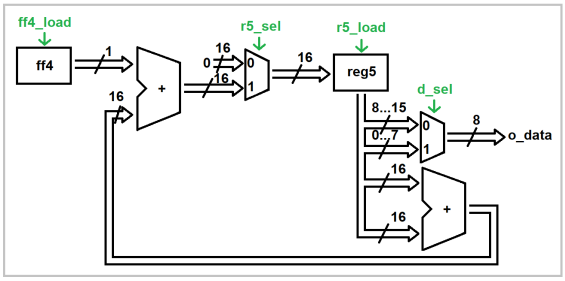


Figura 8: Parte a destra del datapath

2.2 Macchina a Stati Finiti

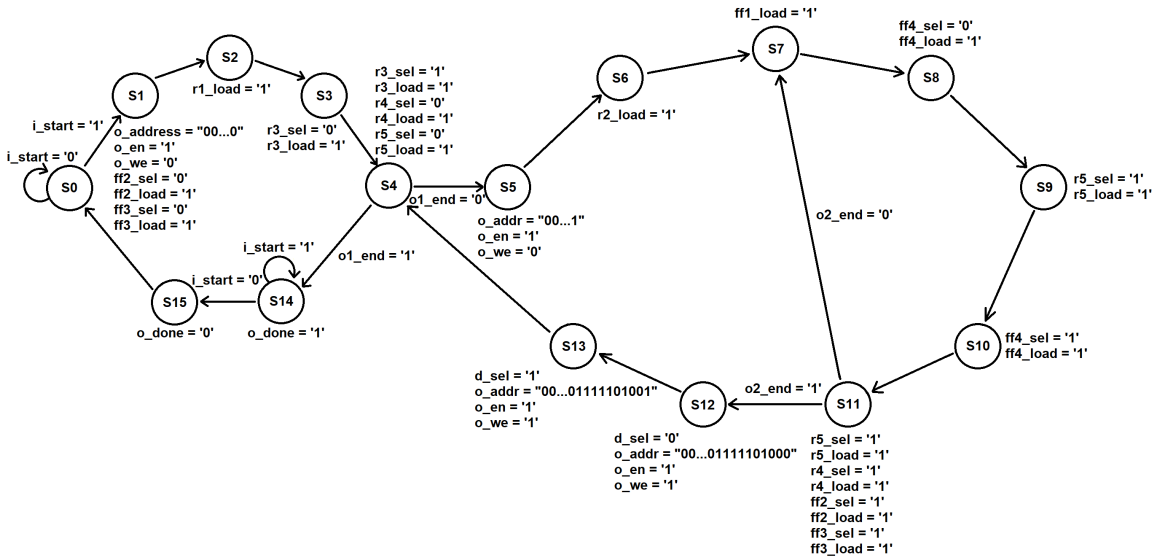


Figura 9: Macchina a stati finiti del modulo implementato

La macchina a stati finiti è costituita da 16 stati.
Ha 3 segnali di input:

- *i_start* [1 bit]: è il segnale di *start* generato dal Test Bench;
- *o1_end* [1 bit]: è il segnale che verrà posto a 1 dal datapath dopo aver letto e computato tutte le parole in ingresso;
- *o2_end* [1 bit]: è il segnale che verrà posto a 1 dal datapath dopo aver letto e computato tutti i bit della parola corrente in ingresso.

Ha 20 segnali di output che vengono riportati solo quando il loro valore è significativo per il funzionamento del modulo; se un segnale di output non viene riportato in una transizione si sottintende uguale a 0:

- *o_address* [16 bit]: è il segnale, composto da un vettore di bit, in uscita che manda l'indirizzo alla memoria;
- *o_en* [1 bit]: è il segnale di *enable* da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- *o_we* [1 bit]: è il segnale di *write enable* da dover mandare alla memoria:
  - '1' per scrivere in memoria;
  - '0' per leggere da memoria;

Nota: nella macchina a stati finiti mostrata in figura 9 sono stati scritti gli indirizzi relativi alla prima computazione (indirizzo 1, indirizzo 1000 e indirizzo 1001), in realtà, tali indirizzi, vengono incrementati, tramite *count*, ad ogni parola letta. (Vedasi 2.1.1)

- *o_done* [1 bit]: è il segnale di uscita che comunica la fine dell'elaborazione;
- *d_sel* [1 bit]: è il segnale di comando del multiplexer incaricato di selezionare le 2 parole risultanti dalla computazione di una singola parola;
- *r3_sel* [1 bit]: è il segnale di comando del multiplexer incaricato di selezionare tra il valore contenuto in *reg1* e il risultato prodotto dal rispettivo sottrattore;
- *r4_sel* [1 bit]: è il segnale di comando del multiplexer incaricato di selezionare tra 0 e il risultato prodotto dal rispettivo sommatore;
- *r5_sel* [1 bit]: è il segnale di comando del multiplexer incaricato di selezionare tra 0 e il risultato prodotto dal rispettivo sommatore;
- *ff2_sel* [1 bit]: è il segnale di comando del multiplexer incaricato di selezionare tra 0 e il terzo bit del risultato prodotto dal "convolutore";
- *ff3_sel* [1 bit]: è il segnale di comando del multiplexer incaricato di selezionare tra 0 e il quarto bit del risultato prodotto dal "convolutore";
- *ff4_sel* [1 bit]: è il segnale di comando del multiplexer incaricato di selezionare tra il primo e il secondo bit del risultato prodotto dal "convolutore";
- *r1_load* [1 bit]: è il segnale che comanda il caricamento di *reg1*;
- *r2_load* [1 bit]: è il segnale che comanda il caricamento di *reg2*;
- *r3_load* [1 bit]: è il segnale che comanda il caricamento di *reg3*;
- *r4_load* [1 bit]: è il segnale che comanda il caricamento di *reg4*;
- *r5_load* [1 bit]: è il segnale che comanda il caricamento di *reg5*;
- *ff1_load* [1 bit]: è il segnale che comanda il caricamento di *ff1*;
- *ff2_load* [1 bit]: è il segnale che comanda il caricamento di *ff2*;
- *ff3_load* [1 bit]: è il segnale che comanda il caricamento di *ff3*;
- *ff4_load* [1 bit]: è il segnale che comanda il caricamento di *ff4*.

2.2.1 Funzionamento della Macchina a Stati Finiti

Dal grafo mostrato in figura 9, oltre al ciclo generale di computazione se ne possono notare altri 2 interni:

- Il primo più grande che ad ogni iterazione computa una parola letta dalla memoria;
- Il secondo più piccolo, posto all'interno dell'altro, che ad ogni iterazione computa i 2 bit prodotti dall'algoritmo convoluzionale.

Descrizione degli stati:

- *S0*: stato iniziale in cui si attende che *i_start* venga portato a 1;
- *S1*: stato in cui viene letto il valore contenuto all'indirizzo 0 (sarà poi disponibile al ciclo di *clock* successivo dato che la memoria è sincrona). Inoltre viene richiesto il caricamento di 0 in *ff2* e *ff3*;
- *S2*: stato in cui viene richiesto il caricamento in *reg1* del valore letto allo stato precedente;
- *S3*: stato in cui viene richiesto il caricamento in *reg3* del valore contenuto in *reg1*;
- *S4*: stato in cui viene richiesto di decrementare il valore contenuto in *reg3* e viene richiesto il caricamento di 0 in *reg4* e *reg5*;
- *S5*: stato in cui viene letto il valore contenuto all'indirizzo *count* (sarà poi disponibile al ciclo di *clock* successivo dato che la memoria è sincrona);
- *S6*: stato in cui viene richiesto il caricamento in *reg2* del valore letto allo stato precedente;
- *S7*: stato in cui viene richiesto il caricamento in *ff1* del valore selezionato dal rispettivo multiplexer;
- *S8*: stato in cui viene richiesto il caricamento in *ff4* del primo bit prodotto dal "convolutore";
- *S9*: stato in cui viene richiesto l'aggiornamento del nuovo valore contenuto in *reg5*;
- *S10*: stato in cui viene richiesto il caricamento in *ff4* del secondo bit prodotto dal "convolutore";
- *S11*: stato in cui viene richiesto l'aggiornamento del nuovo valore contenuto in *reg5*, di incrementare il valore contenuto in *reg4* e di caricare il terzo e quarto bit prodotti dal "convolutore" rispettivamente in *ff2* e *ff3*;
- *S12*: stato in cui vengono scritti all'indirizzo "998+*count*+*count*" della memoria gli 8 bit più significativi del valore contenuto in *reg5*;
- *S13*: stato in cui vengono scritti all'indirizzo "999+*count*+*count*" della memoria gli 8 bit meno significativi del valore contenuto in *reg5*;
- *S14*: stato in cui viene posto il segnale *o_done* a 1; si rimarrà in tale stato fintanto che il valore del segnale *i_start* non sarà portato a 0;
- *S15*: stato in cui viene posto il segnale *o_done* a 0.

3 Risultati Sperimentali

3.1 Sintesi

3.1.1 Utilization Report

Il componente è correttamente sintetizzabile ed implementabile dal tool con un totale di 65 LUT e 63 FF.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	65	0	0	134600	0.05
LUT as Logic	65	0	0	134600	0.05
LUT as Memory	0	0	0	46200	0.00
Slice Registers	63	0	0	269200	0.02
Register as Flip Flop	63	0	0	269200	0.02
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 10: Utilization Report

3.1.2 Timing Report

Un dato importante che riguarda il tempo di esecuzione è lo *slack*, che indica il tempo rimanente al completamento del ciclo di *clock*.
Con un ciclo di *clock* di 100ns si ottengono i seguenti risultati:

```
Slack (MET) :             96.462ns  (required time - arrival time)
  Source:                 FSM_onehot_cur_state_reg[7]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns
fall@5.000ns period=100.000ns})
  Destination:            FSM_onehot_cur_state_reg[0]/CE
                            (rising edge-triggered cell FDPE clocked by clock  {rise@0.000ns
fall@5.000ns 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:        3.156ns  (logic 0.875ns (27.725%)  route 2.281ns (72.275%))
  Logic Levels:           2  (LUT5=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
```

Figura 11: Timing Report

3.1.3 Schematic

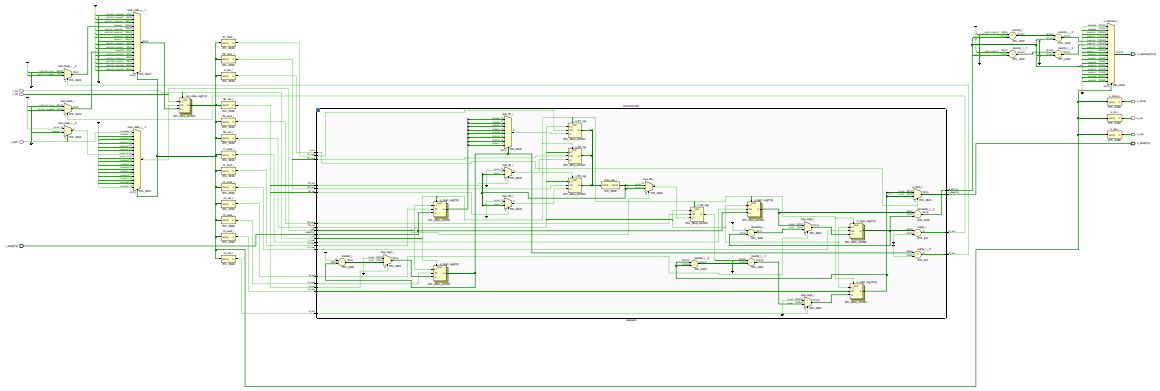


Figura 12: Schematic

3.2 Simulazioni

Tutti i Test Bench sono andati a buon fine superando la Behavioral Simulation e la Post-Synthesis Functional Simulation.
Di seguito sono riportati alcuni risultati di test dei casi limite.

3.2.1 Sequenza Minima

Test che verifica il funzionamento del modulo in caso debba computare la sequenza minima di 0 parole.

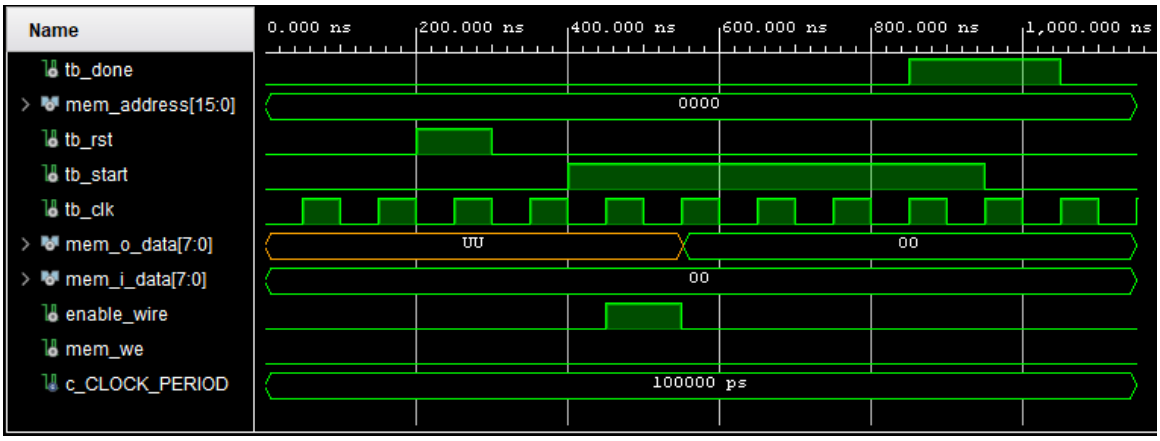


Figura 13: Sequenza Minima

3.2.2 Sequenza Massima

Test che verifica il funzionamento del modulo in caso debba computare la sequenza massima di 255 parole.

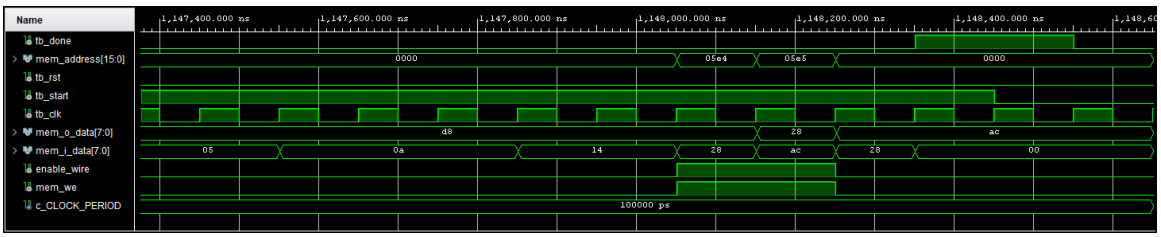


Figura 14: Sequenza Massima

3.2.3 Re-Encode

Test che verifica il funzionamento del modulo in caso debba computare più sequenze consecutivamente. In questo caso viene testata la lettura di 3 sequenze di parole.

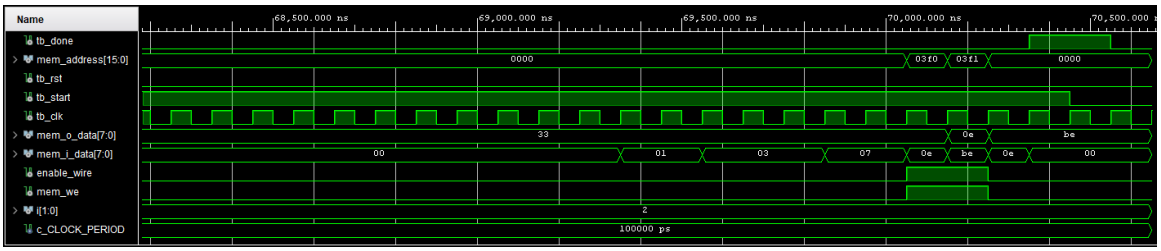


Figura 15: Re-Encode

3.2.4 Reset

Test che verifica il funzionamento del modulo in caso venga portato a 1 il segnale di reset durante la computazione.

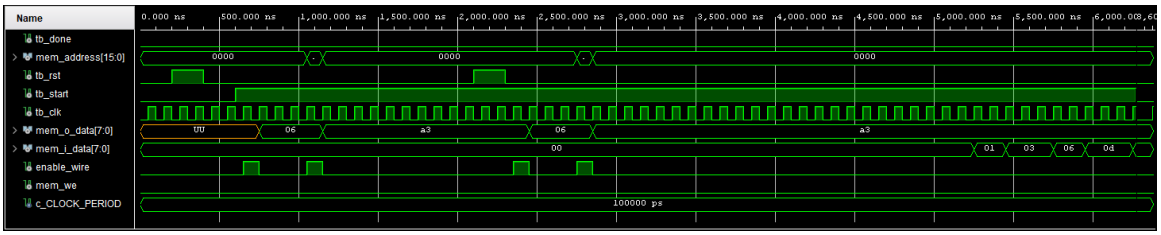


Figura 16: Reset

3.2.5 Altri

Il modulo ha anche passato tutti gli altri Test Bench non riportati tra cui quelli creati da un generatore riadattato per questo progetto.

4 Conclusioni

Si ritiene che il modulo progettato rispetti le specifiche, fatto che è stato ampiamente verificato mediante estensivo testing sia casuale, tramite un generatore di test, che manuale, con Test Bench di casi limite e generici.
Oltre a ciò si è preferito dividere la progettazione in due parti, datapath e macchina a stati finiti, come consigliato durante le lezioni, perché ritenuta una procedura generica e standardizzata che può essere utilizzata per pensare ed implementare qualsiasi progetto di questo tipo, anche di grosse dimensioni.
Si è preferito non ottimizzare ulteriormente il modulo per garantire una maggiore chiarezza logica e funzionale.