PROGETTO FINALE DI RETI LOGICHE

Yodahe Teshome Kumbi 10581027

Shaffaeet Mohammad 10328457

1.INTRODUZIONE

Il modulo da implementare riceve in ingresso una sequenza continua di parole di 8 bit, e restituisce in uscita una sequenza continua di parole da 8 bit dopo aver applicato a ogni bit un codice convoluzionale 1/2.

L'algoritmo da utilizzare per encoding è la seguente

- 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)

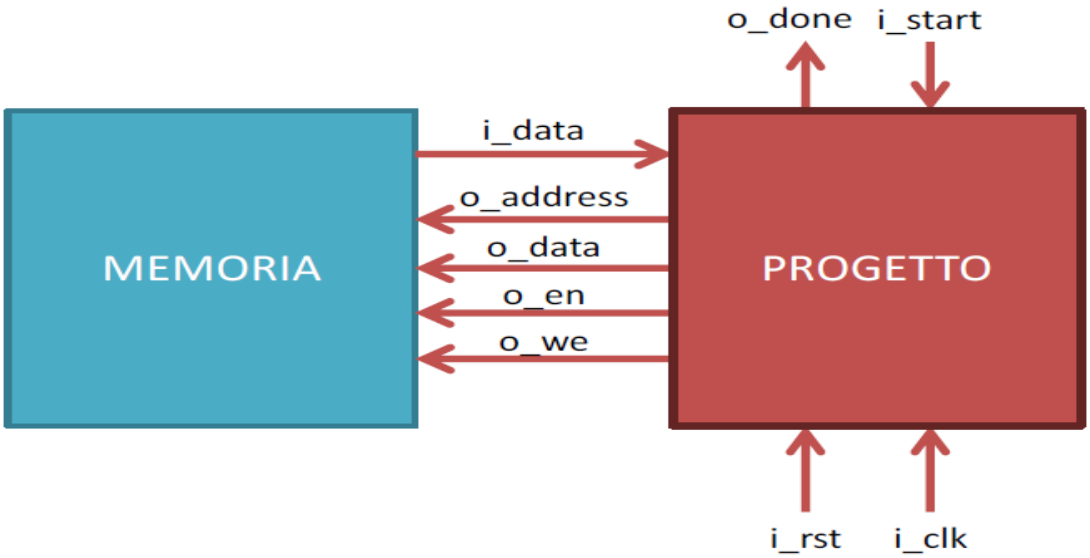
Il modulo legge da una memoria con indirizzamento al byte. La quantita di parole da codificare è da leggere dalla memoria all'indirizzo zero e il primo byte della sequenza da codificare si legge dall'indirizzo 1 dovra essere salvata all'indirizzo 1.

Le sequenze dei byte gia codificate devono essere memorizzate a partire dall'indirizzo 1000.

Il modulo comincia l'elaborazione quando un segnale START in ingresso verrà portato a 1. Il segnale di START rimarrà alto fino a che il segnale di DONE non verrà portato alto.

Al termine della computazione il modulo da progettare deve alzare il segnale che notifica la fine dellelaborazione

Il modulo deve essere progettato per poter codificare più flussi uno dopo l'altro e prima della prima codifica verrà sempre dato il RESET al modulo.

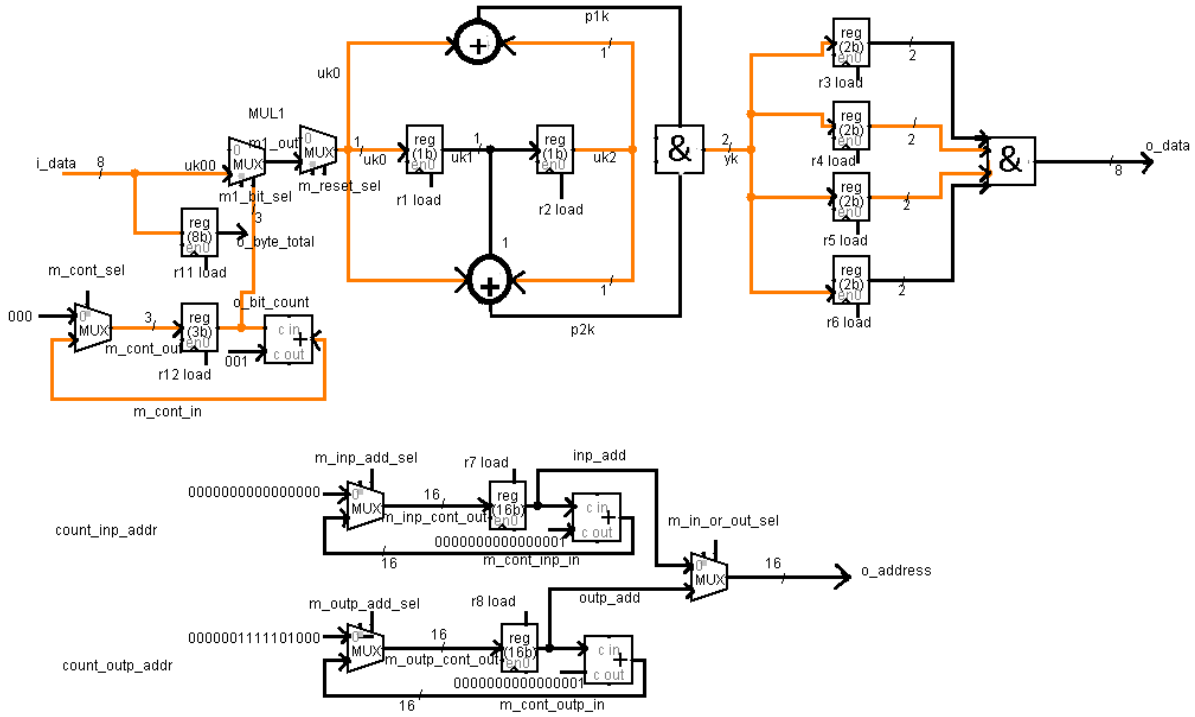


2.ARCHITETTURA

Si è stato scelto di dividere il progetto in due . Il modulo è una collezione di process che implementano la macchina a stati e il datapath .La macchina a stati pilotera il datapath tramite segnali ad ogni registro e multiplexer nel datapath.

Sono stati usati tre contatori , uno per tenere traccia e avanzare l'indirizzo di lettura , il secondo per tenere traccia dell'indirizzo di scrittura e il terzo contatore e stato usato per poter mandare segnale di selezione per un multiplexer che ha il compito di serializzare il byte di ingresso facendo passare un bit per volta.

Dopo di questo il bit in ingresso viene introdotto nell'area dove viene fatto lo XOR con i due bit precedenti e i 2 bit di output vengono concatenati e della codifica viene salvato in appositi registri .Quando tutti e quattro i registri contengono insieme 8 bit , questi vengono concatenati e scritti in memoria.



Funzionamento della macchina a stati

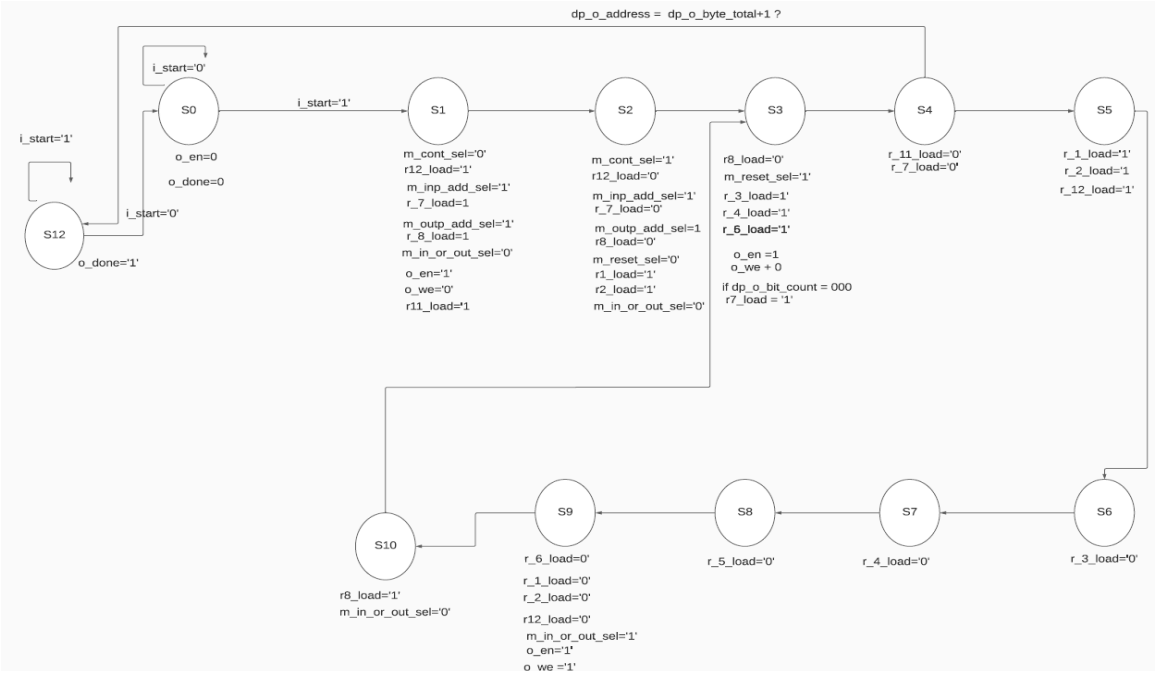
La macchina a stati ha 12 stati con il seguente funzionamento.

S0 : Stato di partenza dopo il segnale di reset . In questo stato mettiamo il segnale di o_done a 0 perchè un nuovo segnale start non può essere dato fin tanto che o_done non è stato riportato a zero.

S1: Qui resettiamo il contatore che usiamo per emettere il segnale usato dal multiplexer MUL1 per selezionare il bit da emettere in output , inizializziamo il multiplexer del contatore bit selector a zero. Inizializzaimo anche i multiplexer dei contatori degli indirizzi di input ed output a zero e salviamo gli indirizzi iniziali di input e output negli appositi registri r7 e r8.

S2:In questo stato settiamo l'indirizzo o_address a quello di lettura leggiamo dalla memoria all'indirizzo Zero la quantita di byte da codificare e lo scriviamo nel registro R11, chiudiamo la retroazione del contatore selezionatore dei bit , e dei contatori degli indirizzi di input e output . Disabilitiamo il i registri degli indirizzi di input e output cosi i dati non vengono modificati . Mettiamo a zero il multiplexer all'ingresso della sezione che si occupa della codifica convoluzionale e abilitiamo i

registri 1 e 2 in modo da poter salvare il valore iniziale a zero per i bit al tempo t-1 e t-2 utili per la codifica.
S3 :
IN questo stato parte il ciclo che va da S3 a S10 che fa la codifica di 4 bit per volta. In questo stato qui mettiamo a uno il selezionatore del multiplexer per lasciare passare un bit per ciclo di clock da codificare e se il contatore dei bit ha completato 2 giri (quindi se ha codificato 8 bit) facciamo avanzare di uno l'indirizzo di lettura.
S4:
Questo è lo stato in cui controlliamo se abbiamo finito di codificare tutti i byte di input nel quale caso procediamo allo stato finale , lo stato s12 . Nel caso contrario si procede allo stato s5
S5:
In questo stato abilitiamo i registri 1 e 2 in modo da poter salvare i valori dei bit al tempo t-1 e t-2 utili per la codifica del primo o quinto bit e salviamo nel registro r3 la prima coppia di bit ottenute dopo la prima codifica e facciamo avanzare il contatore del selezionatore dei bit.
S6:
In questo stato abilitiamo i registri 1 e 2 in modo da poter salvare i valori dei bit al tempo t-1 e t-2 utili per la codifica del secondo o sesto bit e salviamo nel registro r4 la seconda coppia di bit ottenuta dopo la codifica e facciamo avanzare il contatore del selezionatore dei bit.
S7:
In questo stato abilitiamo i registri 1 e 2 in modo da poter salvare i valori dei bit al tempo t-1 e t-2 utili per la codifica del terzo o settimo bit e salviamo nel registro r5 la terza coppia di bit ottenuta dopo la codifica e facciamo avanzare il contatore del selezionatore dei bit.
S8:
In questo stato abilitiamo i registri 1 e 2 in modo da poter salvare i valori dei bit al tempo t-1 e t-2 utili per la codifica del quarto o ottavo bit e salviamo nel registro r6 la quarta coppia di bit ottenuta dopo la codifica e facciamo avanzare il contatore del selezionatore dei bit.
S9:
In questo stato disabilito i registri r1 e r2 per salvare i valori dei 2 bit utili per la codifica del quinto bit in ingresso che avrà bisogno dei bit al tempo t-1 e t-2 e scrivo in memoria la concatenazione degli 8 bit nei registri r3, r4, r5 ed r6.
S10:Qui faccio avanzare il contatore degli indirizzi di output e vado allo stato s3 per codificare altri 4 bit.
S12 :Qui emetto il segnale che comunica che ho finito la codifica dei byte in ingresso.

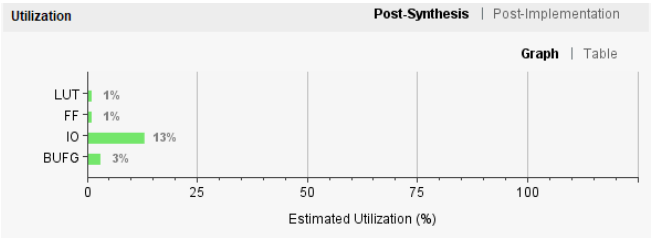


3. Risultati sperimentali:

a. Sintesi (Report di sintesi)

Sia la behavioural che la post-synthesis simulation son terminate senza latch inferiti.Sono stati utilizzati 97 FlipFlop e nessun latch .

Resource	Estimation	Available	Utilization %
LUT	78	134600	0.06
FF	97	269200	0.04
IO	38	285	13.33
BUFG	1	32	3.13




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

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   78 |     0 |    134600 |  0.06 |
|   LUT as Logic          |   78 |     0 |    134600 |  0.06 |
|   LUT as Memory         |    0 |     0 |     46200 |  0.00 |
| Slice Registers         |   97 |     0 |    269200 |  0.04 |
|   Register as Flip Flop |   97 |     0 |    269200 |  0.04 |
|   Register as Latch     |    0 |     0 |    269200 |  0.00 |
| F7 Muxes                |    1 |     0 |     67300 | <0.01 |
| F8 Muxes                |    0 |     0 |     33650 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

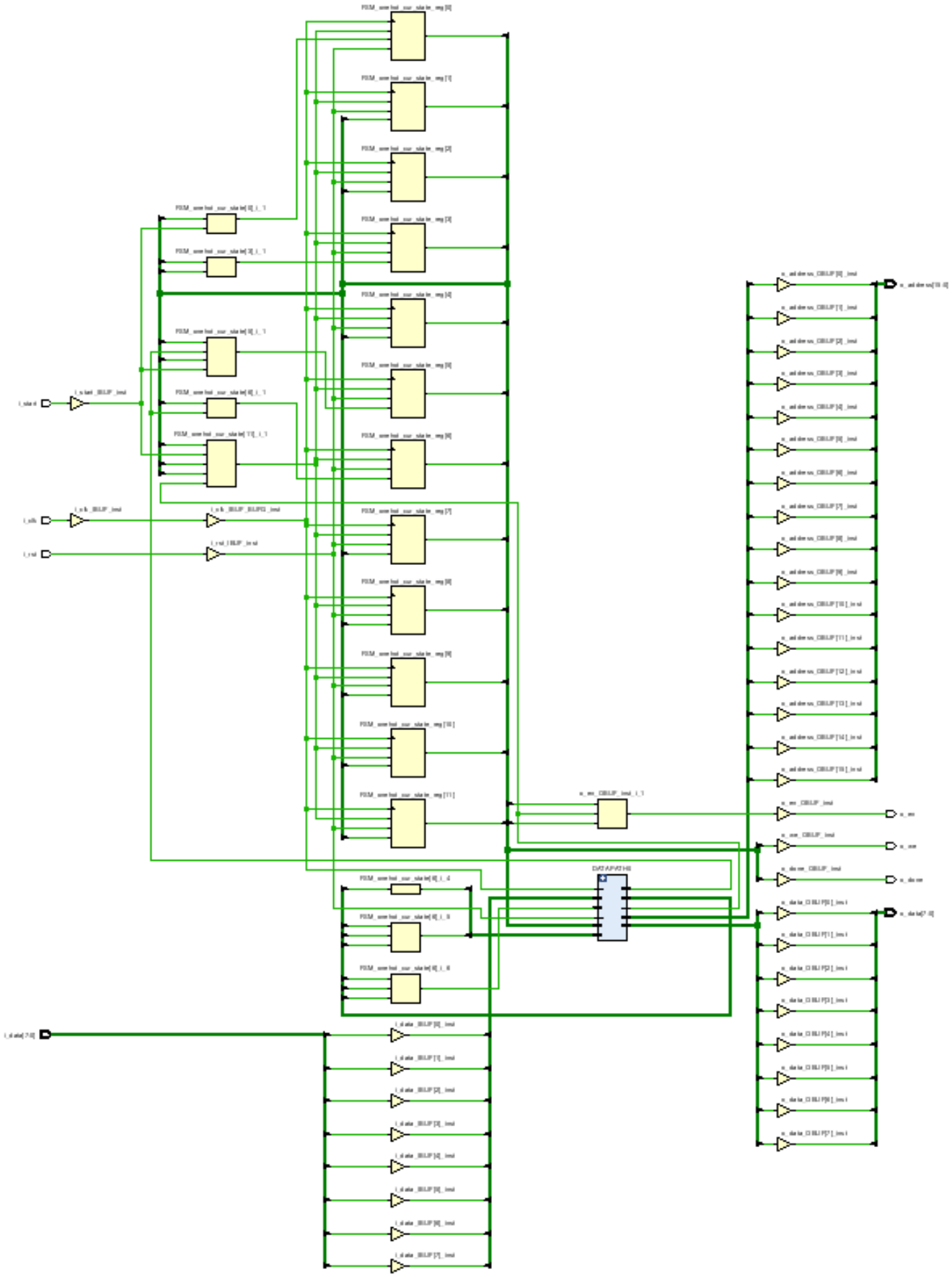
Come da tabella si puo notare che il Worst Negative Slack ottenuto e 93.034 ns , quindi il clock minimo applicabile al progetto è di 8.966 ns.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	93.034 ns	Worst Hold Slack (WHS):	0.168 ns	Worst Pulse Width Slack (WPWS):	49.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	149	Total Number of Endpoints:	149	Total Number of Endpoints:	98

All user specified timing constraints are met.

Schema di sintesi



b. Simulazioni

Di seguito troviamo alcuni dei test bench usati per le simulazioni behavioural e post sintesi. Tutti i test hanno avuto esito positivo.

i. tb_esempio1 Fornisce in input una sequenza di 2 parole da codificare e controlla la correttezza degli output

ii. tb_doppio_uguale Controlla se il modulo puo codificare piu flussi uno dopo l'altro e salvarlo nella stessa memoria . In questo caso fornisce 2 volte in input la stessa sequenza di parole .

iii. tb_seq_min Fornisce una sequenza di lunghezza nulla e controlla se il modulo emette il segnale di fine esecuzione senza fare le operazioni di codifica.

Iv. tb_seq_max controlla il condizione limite di massimo input , cioe una sequenza di lunghezza massima.

v.tb_re_encode controlla se il modulo riesce a codificare piu flussi uno dopo l'altro e se riesce a salvarli in 3 diverse memorie fornendo in input 3 sequenza di parole .

4. Conclusioni

Il modulo che è stato progettato soddisfa pienamente tutti i requisiti della specifica e infatti supera tutte le simuzioni con vari testbench e adempie anche ai vincoli di tempo forniti.