Prova Finale Progetto di Reti Logiche

Politecnico di Milano AA 2021/2022
Prof. Fabio Salice

Alessandro Franzini (Codice Persona 10690276 - Matricola 913663)

Indice

1 Introduzione

1.1 Scopo del progetto

L'obiettivo del progetto è l'implementazione di un modulo HW descritto in VHDL che, ricevuta in ingresso una sequenza continua di W parole, ognuna di 8 bit, restituisca in uscita una sequenza continua di Z parole, ognuna da 8 bit, applicando l'algoritmo di Viterbi.

1.2 Algoritmo di Viterbi

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 $\frac{1}{2}$ (ogni bit viene codificato con 2 bit) secondo lo schema fornito nelle specifiche che genera in uscita un flusso continuo `Y` (ottenuto come concatenamento alternato dei due bit di uscita). Utilizzando la notazione riportata in Figura 1, 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. La lunghezza del flusso U è 8*W, mentre la lunghezza del flusso Y è 8*W*2 (Z=2*W).

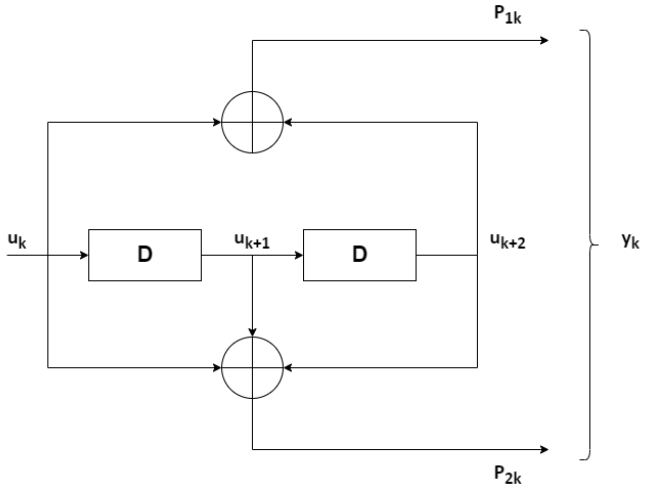


Figura 1: Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$

Il convolutore è una macchina sequenziale sincrona con un clock globale e un segnale di reset con il seguente diagramma degli stati che ha nel suo 00 lo stato iniziale, con uscite in ordine P1K, P2K (ogni transizione è annotata come `Uk/p1k, p2k`).

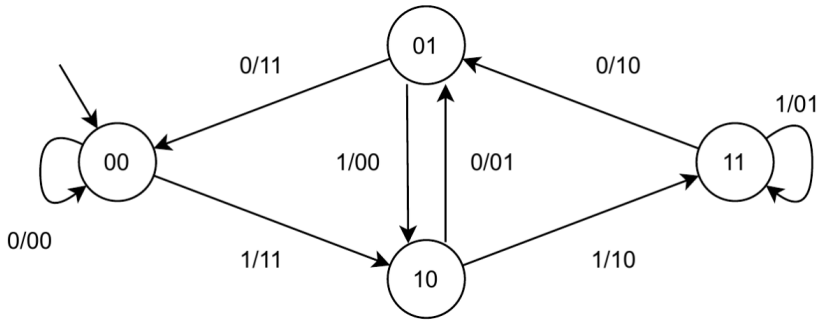


Figura 2: Diagramma degli stati per l'agloritmo di Viterbi

1.2.1 Esempio

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 = 11100101 (229 in decimale), viene serializzata come 1 al tempo t, 1 al tempo t+1, 1 al tempo t+2, 0 al tempo t+3, 0 al tempo t+4, 1 al tempo t+5, 0 la tempo t+6 e 1 al tempo t+7

Applicando l'algoritmo convoluzionale, mostrato in Figura 2, si ottiene la seguente serie di coppie di bit:

T	0	1	2	3	4	5	6	7
Uk	1	1	1	0	0	1	0	1
P1k	1	1	0	1	1	1	0	0
P2k	1	0	1	0	1	1	1	0

Il concatenamento dei valori Pk1 e Pk2 per produrre Z segue il seguente schema:
$Pk1(t) \rightarrow Pk2(t) \rightarrow Pk1(t+1) \rightarrow Pk2(t+1) \rightarrow, ..., \rightarrow Pk1(t+7) \rightarrow Pk2(t+7)$, cioè:

$$Z = 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0$$

- BYTE IN USCITA = 11100110 e 11110100 (230 e 244 in decimale)

NOTA: ogni byte di ingresso W ne genera due in uscita (Z)

1.3 Interfaccia componenti

1.3.1 Interfaccia controllore

Il componente presenta 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;
```

Nello specifico:

- `i_clk` è il segnale di `CLOCK` in ingresso;
- `i_rst` è è il segnale di `RESET` che inizializza la macchina;
- `i_start` è il segnale di `START`;
- `i_data` è il segnale che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_address` è il segnale di uscita che manda l'indirizzo alla memoria;
- `o_done` è il segnale di fine elaborazione;
- `o_en` è il segnale di `ENABLE` da inviare alla memoria per poter comunicare sia in lettura che in scrittura;
- `o_we` è il segnale di `WRITE ENABLE` per abilitare la scrittura in memoria;
- `o_data` è il segnale di uscita dal componente verso la memoria.

Il controllore si interfaccia con un componente (il convolutore) che si occupa dell'esecuzione dell'algoritmo di Viterbi.

1.3.2 Interfaccia convolutore

Il convolutore presenta la seguente interfaccia:

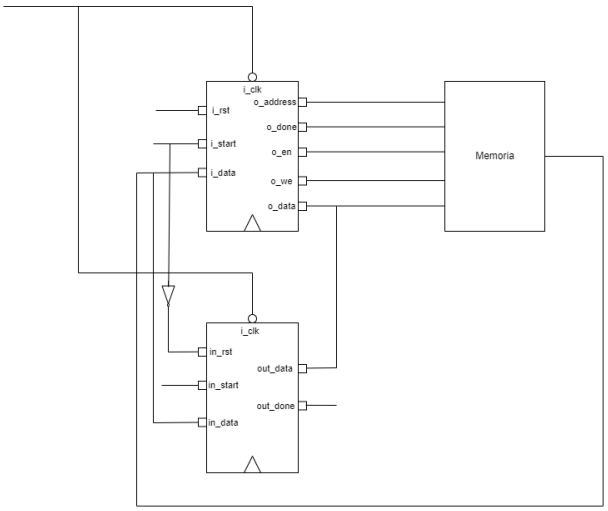
```
entity convolutore is
    port (
    in_clk : in std_logic;
    in_rst : in std_logic;
    in_start : in std_logic;
    in_data : in std_logic_vector(7 downto 0);
    out_data : out std_logic_vector(15 downto 0);
    out_done : out std_logic := '0'
    );
end convolutore;
```

Nello specifico:

- `in_clk` è il segnale di `CLOCK` in ingresso;
- `in_rst` è è il segnale di `RESET` che inizializza la macchina;
- `in_start` è il segnale di `START`;
- `in_data` è il segnale che arriva dal controllore che corrisponde al BYTE W a cui applicare l'algoritmo;
- `out_data` è il segnale di uscita dal convolutore verso il componente e contiene Z (i due BYTE generati).
- `out_done` è il segnale di fine elaborazione del convolutore;

1.4 Collegamento componenti

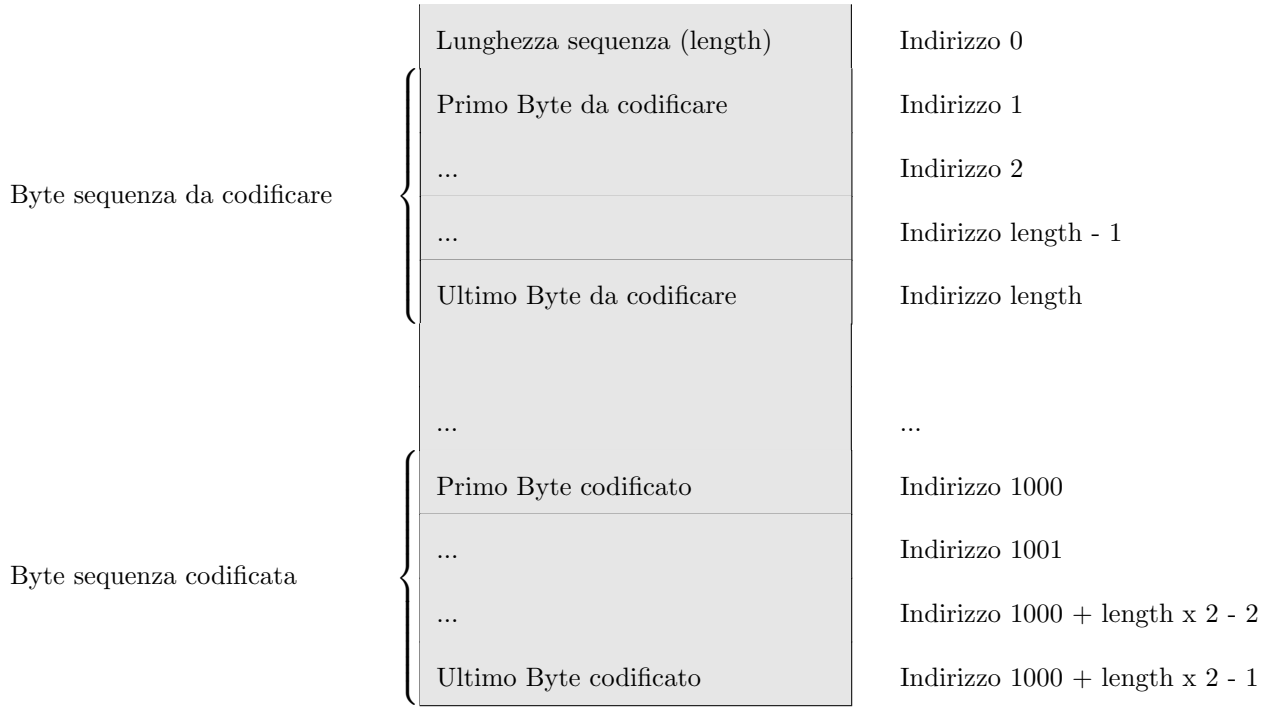
Controllore, convolutore e memoria sono collegati come mostrato di seguito:



NOTA: lo schema intende mostrare i collegamenti tra i componenti in modo semplificato, la descrizione dettagliata del funzionamento sarà affrontata in seguito.

1.5 Memoria

I dati, ciascuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento al byte nel seguente modo:



All'indirizzo 0 è allocato il numero che rappresenta la lunghezza della sequenza di ingresso da codificare.
I Byte da codificare sono memorizzati in memoria a partire dall'indirizzo 1, mentre i Byte codificati (in uscita dalla macchina dopo aver applicato l'algoritmo di Viterbi), vengono salvati in memoria a partire dall'indirizzo 1000.
La dimensione massima della sequenza di ingresso è 255 Byte.

1.6 Specifiche per inizio e fine computazione

Quando il segnale di ingresso `i_start` viene portato a 1, il componente progettato inizia la computazione spostandosi nello stato `WAITING` in cui inizia a richiedere dati in memoria (il primo è la lunghezza della sequenza memorizzata all'indirizzo 0). Il segnale `i_start` rimmarrà alto fino a quando `o_done` non verrà portato alto.
Dopo aver scritto l'intero risultato in memoria, il componente alza a 1 il segnale di `o_done`, per segnalare la fine dell'elaborazione. Il segnale `o_done` rimane alto fino a quando il segnale di `i_start` non viene riportato a 0; dopodichè anche `o_done` viene abbassato. Nell'intervallo di tempo durante cui `i_start` = 0 e `o_done` = 1 non può essere dato nessun nuovo segnale di `START`. Se a questo punto viene rialzato il segnale di `START`, il modulo riparte con la fase di codifica.

1.7 Specifiche per accesso alla memoria

Durante la computazione posso accedere in memoria all'indirizzo indicato in `o_address` sia in lettura che in scrittura. In particolare:

- Se `o_en`=1 e `o_we`=0 leggo dalla memoria il dato contenuto nel segnale `i_data`
- Se `o_en`=1 e `o_we`=1 scrivo in memoria il dato contenuto nel segnale `o_data`

2 Architettura

Per la progettazione del componente è stata scelta l'implementazione tramite due macchine a stati finiti (FSM) di Mealy.

2.1 FSM Controllore

La macchina principale è composta da 7 stati. Di seguito è fornita una breve descrizione qualitativa del loro funzionamento.

2.1.1 IDLE state

Stato iniziale e di default della macchina, in cui si attende il segnale di `i_start` e in cui si torna in caso di ricezione di un segnale di reset.

2.1.2 WAITING state

Stato in cui si attende la risposta della memoria in seguito alla richiesta di un dato o la fine dell'elaborazione da parte della FSM Convolutore segnalata alzando a 1 il segnale `conv_done`.

2.1.3 GET_LENGTH state

Stato in cui si legge la prima cella di memoria (indirizzo 0), contenente la lunghezza della sequenza da codificare.

2.1.4 READ state

Stato in cui si leggono (uno per volta) i Byte da codificare (`Uk`) passandoli in input alla FSM Convolutore che si occupa dell'esecuzione dell'algoritmo di Viterbi.

2.1.5 WRITE state

Stato in cui vengono salvati in memoria (uno per volta) i valori della sequenza codificata (`Z`).

2.1.6 UPDATE state

Stato in cui, prima che avvenga la transizione verso DONE, vengono settati tutti i segnali e le uscite ai valori di default.

2.1.7 DONE state

Stato per la terminazione di un'istanza di computazione che riporta allo stato di IDLE per rendere il modulo disponibile a nuove computazioni.

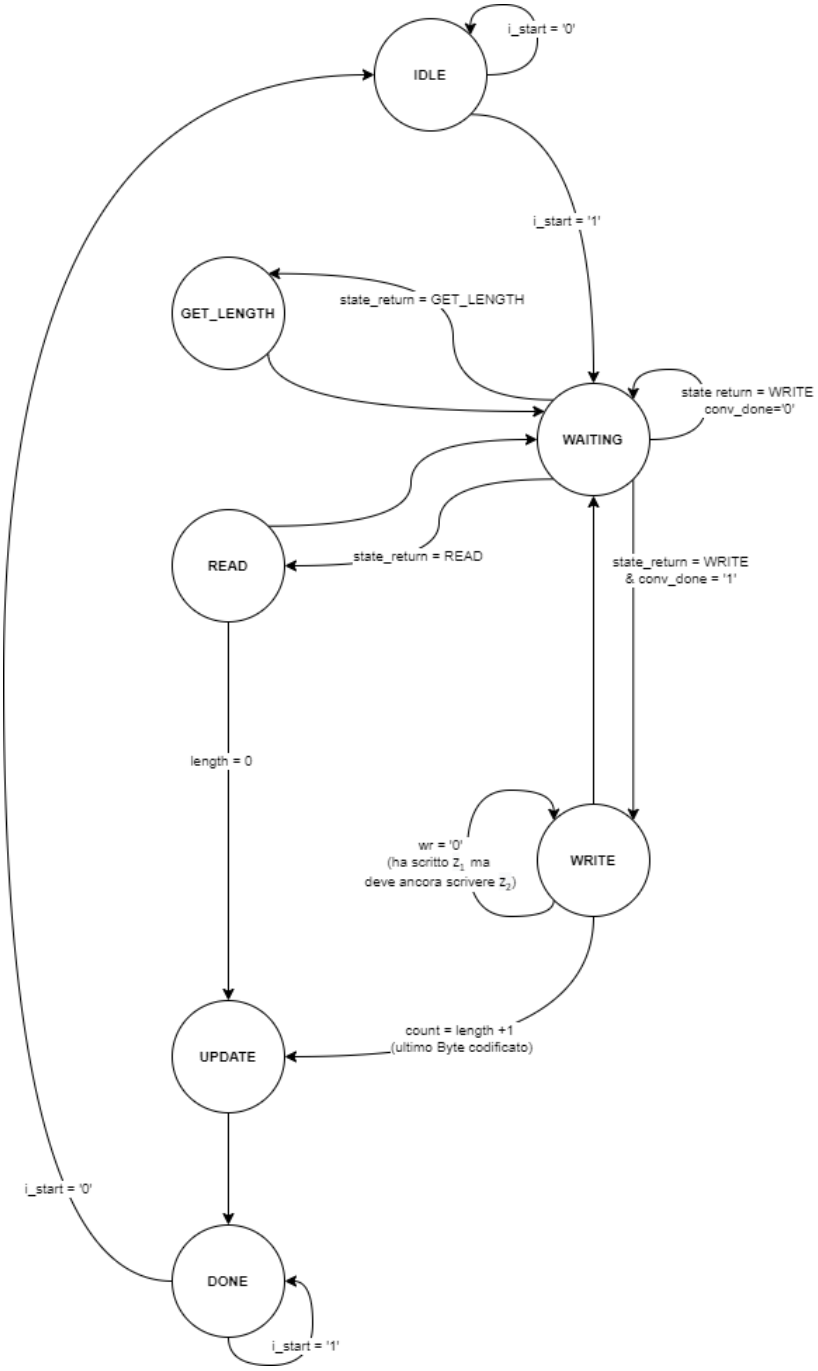


Figura 3: Diagramma degli stati FSM Controllore

2.2 FSM Convolutore

La macchina del convolutore è composta da 4 stati: il suo diagramma degli stati è stato mostrato in precedenza, in Figura 2.
Ogni Byte da codificare letto in memoria dalla macchina controllore viene passato al convolutore che, per ogni bit applica l'algoritmo di Viterbi. Il convolutore segnala la fine della sua elaborazione alzando a 1 il segnale `out_done` che corrisponde al segnale `conv_done` del controllore, che può quindi scrivere il risultato (in uscita dal vettore `out_data` del convolutore) in memoria.

NOTA: Siccome il modulo deve poter codificare più flussi uno dopo l'altro, il segnale di `in_rst` del convolutore corrisponde al segnale `i_start` del controllore, così che quando `i_start`=1 (ad ogni nuova elaborazione) il convolutore viene portato nel suo stato di iniziale 00 (che è anche quello di reset).

2.3 Port map

Di seguito viene riportato il mapping delle porte tra convolutore e controllore:

```
port map(
    in_clk => i_clk,
    in_rst => i_start,
    in_start => conv_start,
    in_data => uk,
    out_data => result,
    out_done => conv_done
);
```

Il segnale `in_start` del convolutore viene attivato dal controllore dopo la lettura dalla memoria del Byte da codificare, mentre viene portato a 0 dopo aver finito la computazione (cioè quando `out_done`='1'). La scelta di inserire questo segnale che ha la funzione di start&stop è dovuta a una maggiore sincronizzazione tra convolutore e controllore, così da essere sicuri che la macchina convolutore sia ferma fino a quando il controllore non legge il nuovo Byte.

2.4 Valori di default

Di seguito sono riportati i signals e le variabili utilizzate e i rispettivi valori di default a esse assegnati.

FSM	Nome	Uso	Default	Motivazione
CNV	`current_state`	Contiene lo stato attuale	S0	Scelta arbitraria
CNV	`z`	Contiene i 16 bit codificati	others $\Rightarrow' 0'$	Scelta arbitraria
CNV	`r_count`	Helper per scorrere il vettore `uk` da codificare	7	Uk è un Byte quindi l'ultimo bit è in posizione 7
CNV	`i`	Helper per scorrere il vettore `z`	15	Z è un vettore di due Byte quindi l'ultimo bit è in posizione 15
CTR	`current_state`	Contiene lo stato attuale	IDLE	Scelta arbitraria
CTR	`length`	Contiene la lunghezza della sequenza	0	Scelta arbitraria
CTR	`count`	Helper per scorrere la memoria in lettura	1	Il primo Byte da codificare è all'indirizzo 1
CTR	`uk`	Contiene il Byte da codificare letto in memoria	others $\Rightarrow' 0'$	Scelta arbitraria
CTR	`result`	Contiene i due Byte codificati	others $\Rightarrow' 0'$	Scelta arbitraria
CTR	`write_address`	Contiene l'indirizzo di memoria dove scrivere i dati	1000	Il primo Byte codificato va scritto all'indirizzo 1000 (da specifica)

NOTA: CNV : FSM Convolutore, CTR : FSM Controllore

3 Sintesi

3.1 Report Utilization

Di seguito viene mostrato il Report Utilization fornito da Vivado dopo la sintesi:

Risorsa	Utilizzo	Disponibilità	Utilizzo %
LUT	134	134 600	0.10
FF	143	269 200	0.05

In particolare, la FSM Convolutore utilizza 82 LUT (dei 134) e 69 FF (dei 143).

3.2 Report Timing

La specifica richiede che il componente funzioni con un periodo di clock $T_{clock} \leq 100ns$. Per la verifica della soddisfacibilità del requisito è stato aggiunto post sintesi un vincolo temporale a `i_clk` che setta il periodo a 100ns.
I risultati sono mostrati nel Report Timing Summary qui riportato:

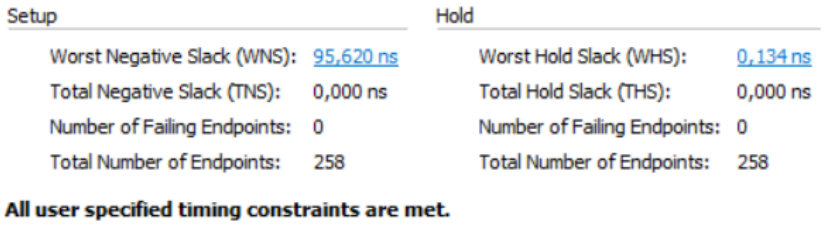


Figura 4: Timing Summary con $T_{clock} = 100ns$

NOTA: Lo Slack (= required time - arrival time) rappresenta la differenza tra il tempo di arrivo del segnale richiesto affinchè il vincolo sia rispettato e il tempo di arrivo effettivo.
Dalla Figura 4 si può notare che il WNS (Worst Negative Slack) è positivo sia per il tempo di Setup che per quello di Hold, dunque il vincolo temporale imposto è rispettato.

3.3 Note aggiuntive sulla sintesi

Tutti i warning per latch inferiti e warning per segnali presenti nel processo ma non inseriti nella sensivity list generati dal tool di sintesi durante lo sviluppo sono stati risolti.
Non è quindi presente alcun warning problematico nella versione finale del componente.

4 Simulazioni

4.1 Test casi limite

4.1.1 Elaborazioni multiple

Il modulo è progettato per poter codificare più flussi uno dopo l'altro. Questo test verifica il funzionamento quando, dopo aver finito una computazione e scritto i risultati, viene alzato nuovamente il segnale `i_start` senza prima fare il RESET della macchina.

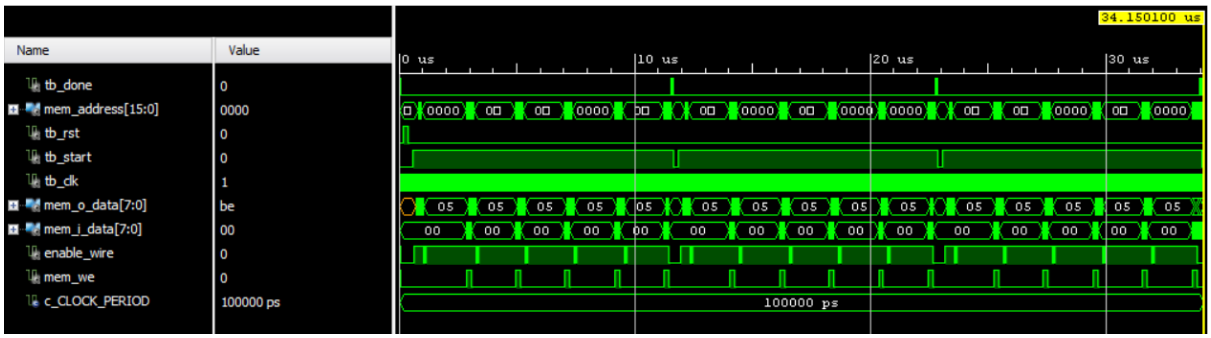


Figura 5: Simulazione test con 3 computazioni di seguito

4.1.2 Reset

In questo test viene inviato un segnale di RESET asincrono durante l'elaborazione. Il componente si comporta correttamente resettando tutti i segnali coi valori di default e riportando la macchina nello stato di IDLE. Quando il segnale `i_rst` torna a 0 la computazione procede correttamente e il risultato finale è quello previsto.

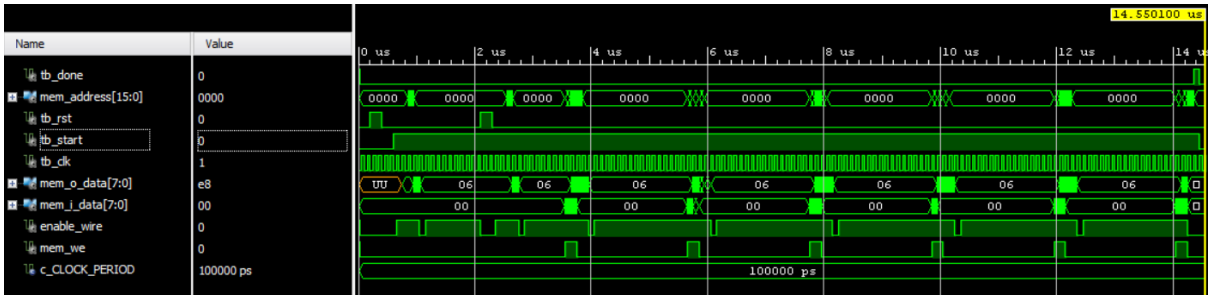


Figura 6: Simulazione test con RESET attivo durante elaborazione

4.1.3 Sequenza minima (0 Byte)

Questo test copre il caso in cui la lunghezza della sequenza da codificare sia 0, ovvero `length`=0.

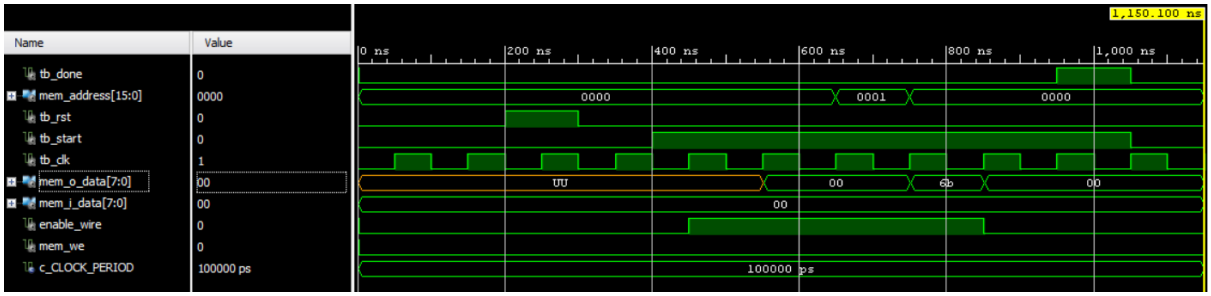


Figura 7: Simulazione test con sequenza minima

4.1.4 Sequenza massima (255 Byte)

Questo test copre il caso in cui la lunghezza della sequenza da codificare sia 255, ovvero `length`=255. La computazione termina con il risultato previsto come si può notare da `mem_o_address` che ha come ultimo valore significativo (prima di venire settato al suo valore di default) $05E5_{16} = 1509_{10}$ che è esattamente 1000 + `length` x 2 - 1

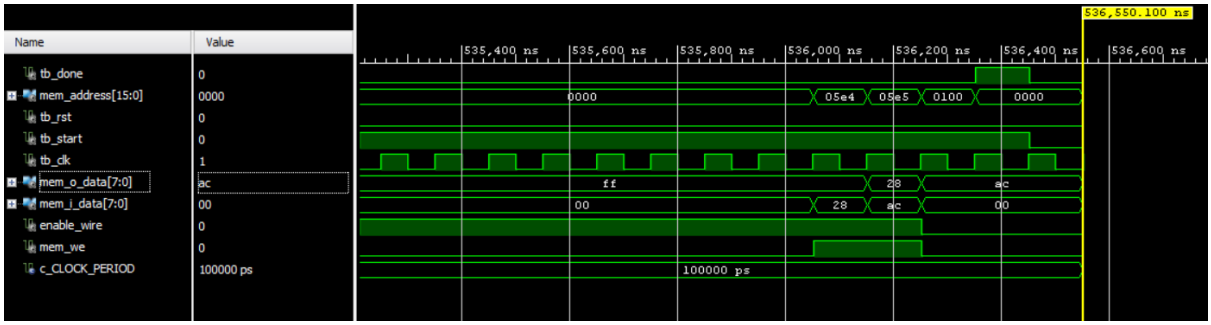


Figura 8: Simulazione test con sequenza massima

4.2 Test casuali

Oltre che a test sui casi limite, il componente è stato sottoposto a diversi TestBench casuali, superati sia a livello *Behavioural* che *Post-Synthesis*.

5 Conclusioni

Riassumendo si è realizzato un componente con queste caratteristiche:

- Funzionante in pre e post-sintesi (*Behavioural Simulation* e *Post-Synthesis Functional Simulation*).
- Funzionante con un periodo di clock $T_{clock} \leq 100ns$.
- Utilizzo di LUT pari al 0.10 %.
- Utilizzo di FF pari al 0.05 %
- *Worst Negative Slack (WNS)* pari a 95.620ns

Il componente è in grado di interfacciarsi con una memoria avente indirizzamento al Byte, di applicare correttamente l'algoritmo di Viterbi per qualunque sequenza di lunghezza compresa tra 0 e 255, come da specifica.
Per la progettazione si è utilizzato il tool "Vivado 2016.4 WebPACK Edition".