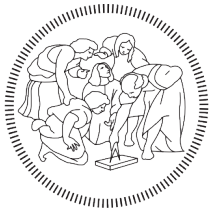
Prova Finale (Progetto di Reti Logiche)

Simone Callegarin (Codice Persona 10676880 - Matricola 932343)

Anno Accademico 2021/2022

Indice



POLITECNICO
MILANO 1863

1 Introduzione

1.1 Scopo del progetto

Sia data in ingresso una sequenza continua di parole da 8 bit ciascuna, lo scopo del progetto è di implementare un componente hardware descritto in VHDL che, serializzando ciascuna delle parole in ingresso in un flusso da singolo bit, applichi il codice convoluzionale $\frac{1}{2}$.
Parallelamente allo svolgimento di questa richiesta si è cercato di produrre un design che potesse risultare un buon compromesso tra facilità di manutenzione, leggibilità di codice e prestazioni sotto diversi aspetti.
Per avvicinarci quanto possibile a questi obiettivi sono state effettuate determinate scelte che verranno esposte e giustificate nelle successive sezioni.

1.2 Funzionamento in sintesi

Il modulo gestisce un flusso di parole da 8 bit, indicato con W come da specifica, questo viene poi sottoposto a diverse operazioni brevemente enunciate in tabella:

FLUSSO	**OPERAZIONI A CUI È STATO SOTTOPOSTO**	**CONTENUTO**	**LUNGHEZZA**
W	Lettura	words	$number\_of\_words$ (da 8 bit)
U	Serializzazione	serialized_words	$8 \times number\_of\_words$ (da 1 bit)
Y	Convoluzione	serialized_and_ convoluted_words	$2 \times 8 \times number\_of\_words$ (da 1 bit)
Z	Parallelizzazione e scrittura	convoluted_words	$2 \times number\_of\_words$ (da 8 bit)

In uscita sarà generato un flusso Z a partire dal flusso W a cui è stato applicato il codice convoluzionale $\frac{1}{2}$.
Tali operazioni sono riportate in modo sintetico anche dalla figura seguente, che riassume a grandi linee il funzionamento della macchina a stati in un modo semplificato:

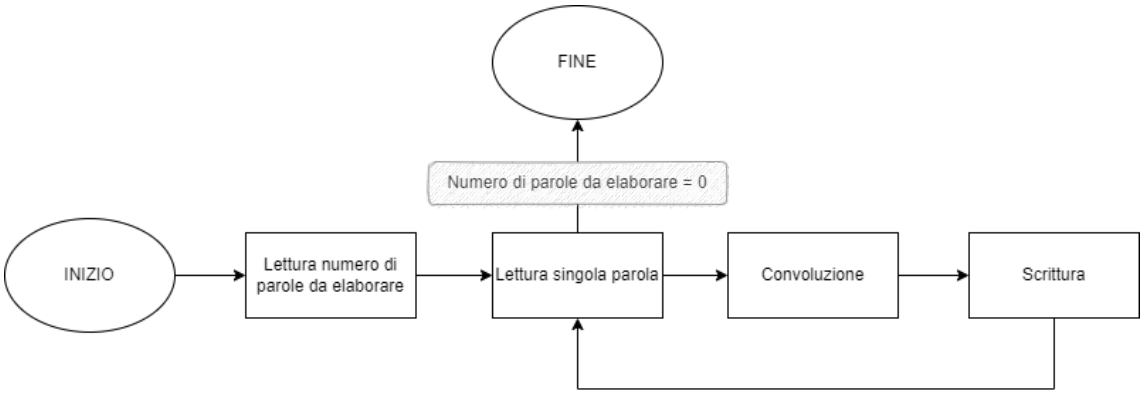


Figura 1: Funzionamento in sintesi

Il **codificatore convoluzionale** si compone di due XOR, uno a due ingressi e l'altro a tre, che a partire dal flusso U genereranno rispettivamente in uscita i bit p_{1k} e p_{2k}, che saranno poi concatenati al fine di produrre il flusso continuo y_k come mostrato da relativa figura:

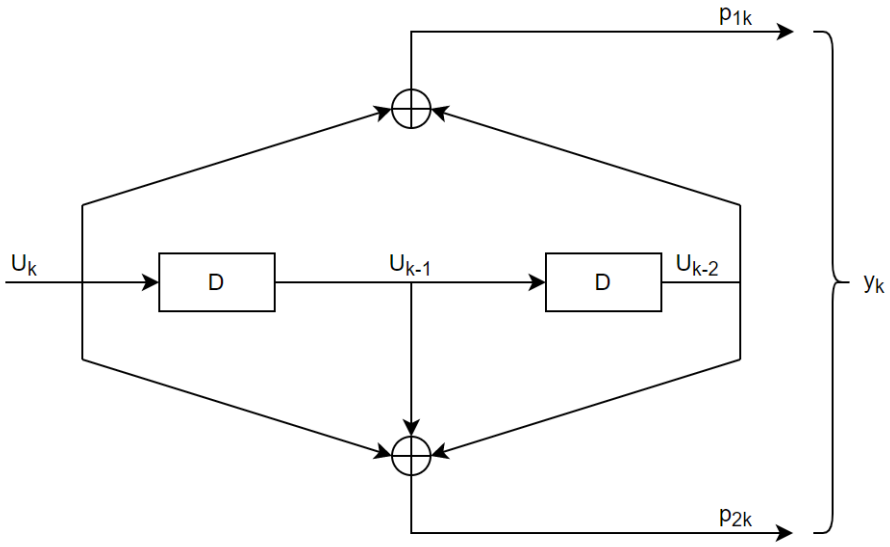


Figura 2: Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$

La seguente tabella mostra i bit generati dal codificatore convoluzionale a seconda dei valori assunti dal flusso U:

U_k	U_{k-1}	U_{k-2}	p_{1k} $(U_k \oplus U_{k-2})$	p_{2k} $(U_k \oplus U_{k-1} \oplus U_{k-2})$	y_k $(p_{1k}$ & $p_{2k})$
0	0	0	0	0	00
0	0	1	1	1	11
0	1	0	0	1	01
0	1	1	1	0	10
1	0	0	1	1	11
1	0	1	0	0	00
1	1	0	1	0	10
1	1	1	0	1	01

Il funzionamento del convolutore è inoltre riconducibile alla seguente macchina di Mealy (sequenziale sincrona con clock globale e segnale di reset che ha in 00 il suo stato iniziale):

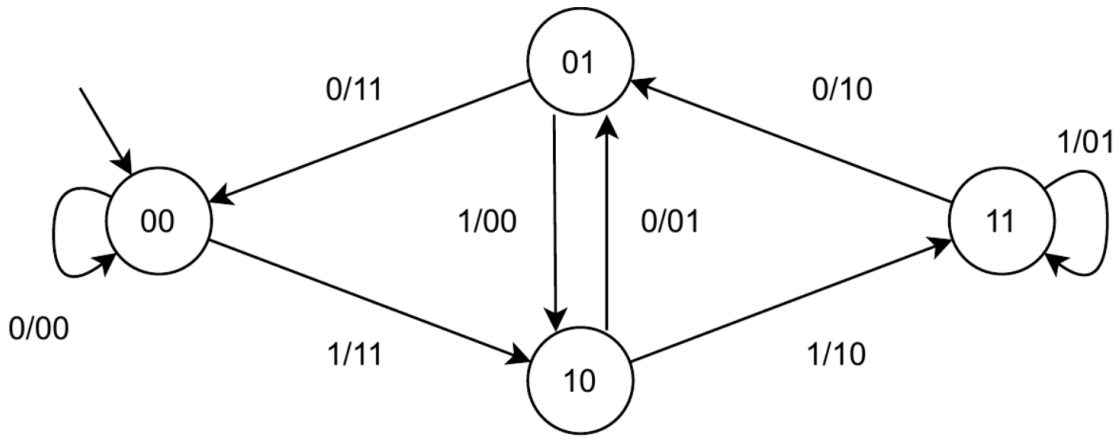


Figura 3: Convolutore

1.3 Interfaccia del componente

Il componente da descrivere ha un'interfaccia così definita:

```
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 pronta per ricevere il primo segnale di START;
- `i_start` è il segnale di `START` generato dal test bench;
- `i_data` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_address` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `o_done` è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- `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 poter scriverci. Per leggere da memoria esso deve essere 0;
- `o_data` è il segnale (vettore) di uscita dal componente verso la memoria.

1.4 Dati e descrizione memoria

I dati, ciascuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento al Byte a partire dall'indirizzo 0 secondo il seguente schema:

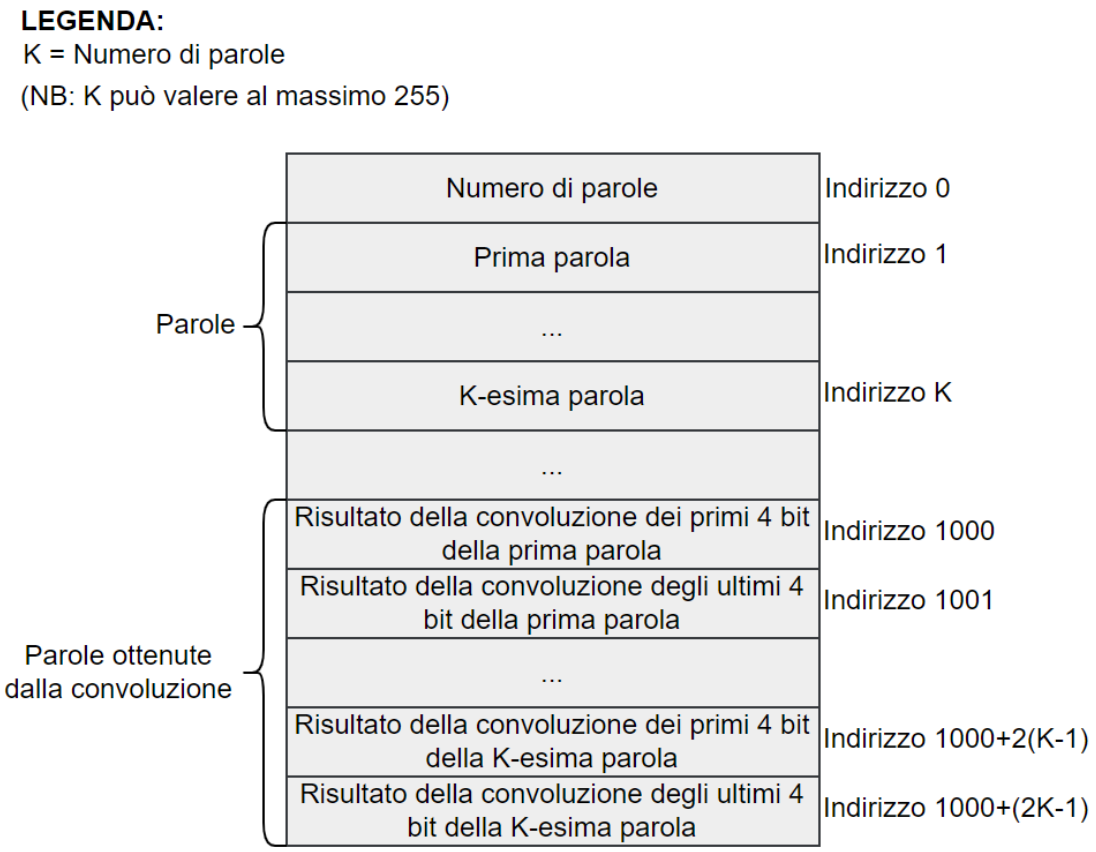


Figura 4: Rappresentazione indirizzi significativi della memoria

Riassumendo:

- All'indirizzo 0 avremo il numero di parole che verrà sottoposto al codice convoluzionale. Esso sarà un valore compreso tra 0 e 255 byte, che corrisponde alla dimensione massima della sequenza di ingresso.
- A partire dall'indirizzo 1 avremo i byte relativi al flusso W di parole, che non potranno superare mai l'indirizzo 255.
- Dall'indirizzo 1000 fino al massimo all'indirizzo 1509 saranno contenuti i byte relativi al flusso Z delle parole che sono state sottoposte al codice convoluzionale.

2 Architettura

2.1 Tabella dei signal interni

NOME	TIPO	VALORE INIZIALE	DESCRIZIONE
current_state	state_type	U	Memorizza stato corrente
next_state	state_type	U	Memorizza stato successivo
count_reg	integer (range 0 to 7)	0	Contatore del numero di bit ancora da serializzare
previous_convolution_state_reg	integer (range 0 to 3)	0	Mantiene un riferimento allo stato di uscita dalla convoluzione precedente
n_words_reg	std_logic_vector (7 downto 0)	00000000	Numero di parole ancora da leggere
word_reg	std_logic_vector (7 downto 0)	00000000	Parola letta da RAM
out_word_reg	std_logic_vector (15 downto 0)	0000000000000000	Parola ottenuta applicando l'algoritmo convoluzionale
input_address_reg	std_logic_vector (15 downto 0)	0000000000000000	Indirizzo per la lettura sequenziale da memoria
output_address_reg	std_logic_vector (15 downto 0)	0000000000000000	Indirizzo di memorizzazione stream di uscita
o_address_next	std_logic_vector (15 downto 0)	0000000000000000	Indirizzo mandato alla RAM
done	std_logic	0	Segnale attivato al termine della codifica
enable	std_logic	0	Attivazione della RAM
write	std_logic	0	Abilitazione scrittura sulla RAM
dout	std_logic_vector (7 downto 0)	00000000	Vettore per la scrittura su RAM

2.2 Macchina a stati finiti

La seguente macchina è composta da 13 stati, 4 dei quali riproducono il convolutore, mentre gli altri 9 svolgono funzioni di lettura e scrittura dati su RAM.
Al fine di produrre una macchina con un numero di stati il più possibile ridotto, mantenendo comunque un ottimo grado di leggibilità e facilità di comprensione del suo funzionamento, si è optato per l'utilizzo di una macchina di Mealy, con l'idea di sfruttare la macchina sequenziale sincrona fornitaci nel file "PFRL_Specifica_21_22".
La macchina è stata progettata con lo scopo di ridurre al minimo il numero di stati utilizzati, ad eccezione di **CONVOLUTION_END** che per praticità di scrittura e lettura di codice VHDL è stato aggiunto al termine del processo di convoluzione al fine di permettere la prima scrittura (di 8 bit) in memoria in una sola transizione di stato invece di quattro, questi sarebbe quindi omettibile ma per i motivi precedentemente indicati si è optato per la sua introduzione.

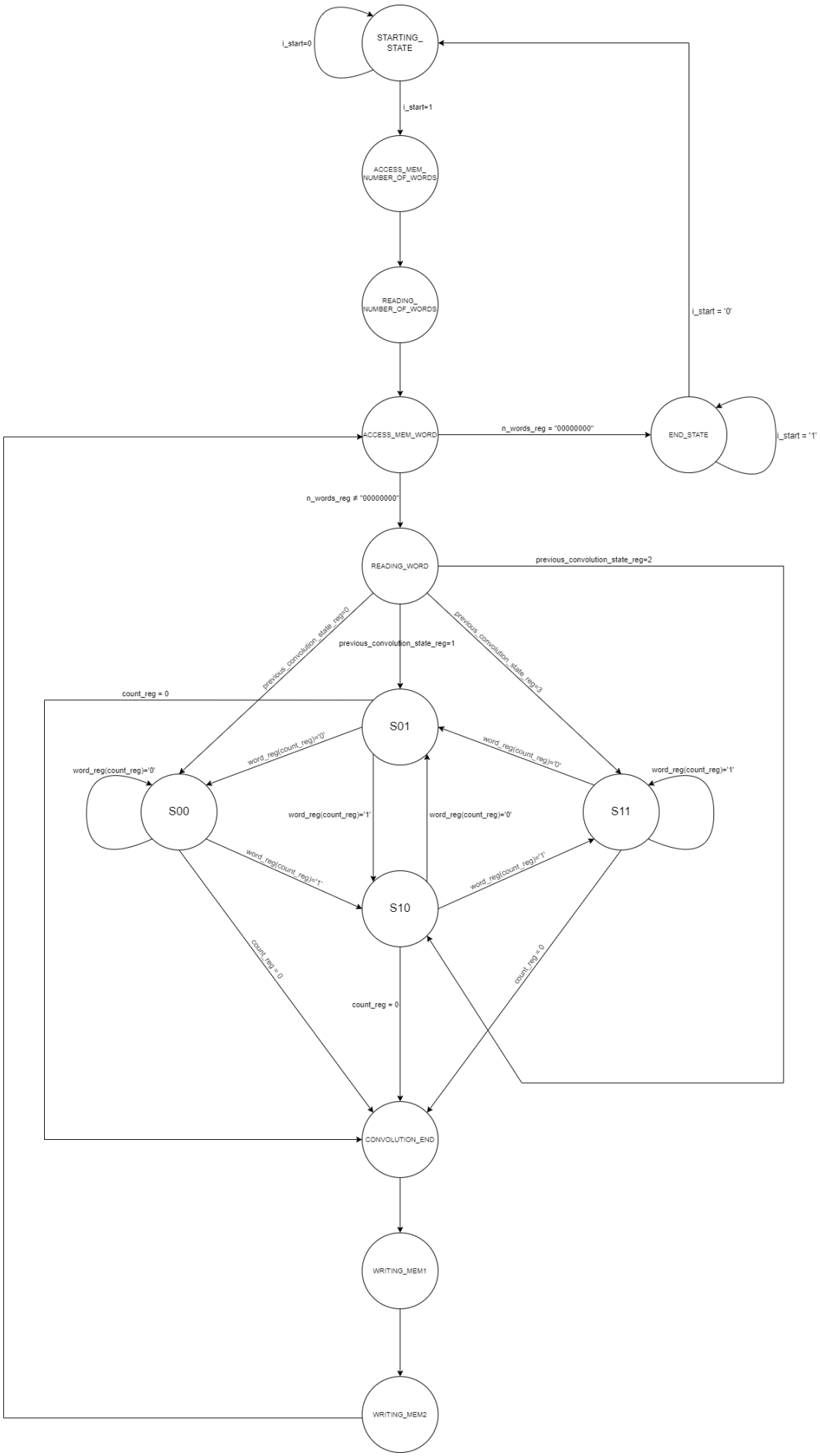


Figura 5: Macchina a stati con le condizioni di transizione.

Il modulo partirà nell'elaborazione quando un segnale `i_start` in ingresso verrà portato a **1**.
Questi rimarrà alto fino a che il segnale di `o_done` non verrà portato a **1**, e ciò accade al termine della computazione (una volta scritto il risultato in memoria), successivamente il segnale di `o_done` rimarrà alto fino a che il segnale di `i_start` non sarà riportato a **0**.
Una volta che il segnale di `i_start` è stato riportato a zero sarà possibile abbassare il segnale di DONE, e se a questo punto viene rialzato il segnale di `i_start` il modulo dovrà ripartire con la fase di codifica.

Segue una breve descrizione per ciascuno degli stati che compongono la macchina.

2.2.1 STARTING_STATE

Stato iniziale in cui si attende che il segnale di `i_start` venga portato alto.
In caso venga alzato il segnale `i_rst` si torna in questo stato al termine della elaborazione.
Quando il segnale di `i_start` è portato a 1 vengono settati `count_reg` a 7 e `output_address` a "0000001111101000" (1000 in numero decimale).

2.2.2 ACCESS_MEM_NUMBER_OF_WORDS

Stato in cui viene attivata la lettura della `RAM` all'indirizzo "0000000000000000"

2.2.3 READING_NUMBER_OF_WORDS

Stato in cui avviene l'effettiva lettura da `RAM` del numero di parole da elaborare, che sarà salvato in `n_words_reg`.

2.2.4 ACCESS_MEM_WORD

Nel caso in cui `n_words_reg` contenga il valore "00000000" la transizione porta a **END_STATE** e alza il segnale di `done`.
Altrimenti in questo stato sono attivate le letture della `RAM` a partire dall'indirizzo 1 fino all'indirizzo pari a $1000 + (2 \times numero\_di\_parole\_da\_leggere - 1)$, all'interno dei quali sono contenute le singole parole a cui applicare il codice convoluzionale.

2.2.5 READING_WORD

Stato in cui avviene la lettura da `RAM` della parola da elaborare, che sarà salvata in `word_reg`.
La transizione successiva coinvolge il registro `previous_convolution_state_reg` che indicherà in base al suo valore in quale stato della macchina che si occupa della convoluzione entrare (avrà valore 0 per riferirsi a **S00**, 1 per **S01**, 2 per **S10** e 3 per **S11**).

2.2.6 CONVOLUTORE (S00, S01, S10 e S11)

Per ciascuno stato appartenente al convolutore la transizione allo stato successivo dipende dal `count_reg`, quando questi è 0 essa notificherà la fine della convoluzione e terminerà in **CONVOLUTION_END**, altrimenti in base al bit serializzato ricevuto in ingresso (`word_reg(count_reg)`) sarà deciso lo stato successivo, seguendo a modello la macchina di Mealy fornita da specifica.
Ciascuno stato convolutore si occupa di shiftare a sinistra di due posizioni `out_word_reg` e aggiungere ad esso i seguenti valori in base a `word_reg(count_reg)`:

STATE	`word_reg(count_reg)='0'`	`word_reg(count_reg)='1'`
S00	"0000000000000000"	"0000000000000011"
S01	"0000000000000011"	"0000000000000000"
S10	"0000000000000001"	"0000000000000010"
S11	"0000000000000010"	"0000000000000001"

Inoltre per ogni transizione interna al convolutore viene decrementato `count_reg` di uno e viene memorizzato lo stato in cui si arriverà a seguito della transizione in `previous_convolution_state_reg`.

2.2.7 CONVOLUTION_END

Stato in cui termina la convoluzione, qui è ridotto il numero di parole ancora da leggere di 1 (`std_logic_vector(unsigned(n_words_reg)-1)`).

2.2.8 WRITING_MEM1

Stato in cui viene attivata e abilitata per la scrittura la `RAM` nella quale attraverso il registro `dout` riporto i bit dal 15 all' 8 di `out_word_reg`, che rappresentano i primi 4 bit della parola a cui è stato applicato il codice convoluzionale.
Viene inoltre portato `output_address_next` all'indirizzo successivo.

2.2.9 WRITING_MEM2

Stato in cui viene attivata e abilitata per la scrittura la `RAM` nella quale attraverso il registro `dout` riporto i bit dal 7 allo 0 di `out_word_reg`, che rappresentano gli ultimi 4 bit della parola a cui è stato applicato il codice convoluzionale.
Viene inoltre portato `output_address_next` all'indirizzo successivo ed è riportato a "00000000" il valore di `out_word_reg` a scrittura in memoria ultimata.

2.2.10 END_STATE

Stato di terminazione in cui si rimane fino a che `i_start` non è riportato a 0, in questo caso viene abbassato il segnale di `o_done` e vengono settati al valore iniziale diversi registri interni che sono stati utilizzati.

2.3 Scelte progettuali

La scelta progettuale effettuata è stata quella di utilizzare 3 processi:

1. **registers_process**, processo che ha nella propria sensitivity list clock, reset e n_words_reg. Utilizzato per controllare il fronte di salita del clock e i valori assunti dal reset e da n_words_reg a esecuzione ultimata (quando n_words_reg="00000000");

2. **operations_process**, processo che descrive il funzionamento della macchina di Mealy.
   Al suo interno sono contenute le operazione di lettura, scrittura e convoluzione.
   Presenta una sensitivity list vasta, infatti essa comprende ogni signal reg utilizzato nel codice, dato che al suo interno verranno utilizzati tutti i registri creati.

3. **next_state_process**, processo che gestisce il passaggio da uno stato all'altro.
   Ha come sensitivity list i signal coinvolti nelle condizioni di transizione.

3 Risultati sperimentali

Il componente sintetizzato supera correttamente tutti i test specificati nelle 3 simulazioni: `Behavioral`, `Post-Synthesis Functional` e `Post-Synthesis Timing`.

3.1 Sintesi (Report di sintesi)

3.1.1 Utilization report

"Report Utilization" fornisce indicazioni sull'area occupata dal design sintetizzato come segue:

Resource	Estimation	Available	Utilization %
LUT	124	134600	0.09
FF	100	269200	0.04
IO	38	285	13.33
BUFG	1	32	3.13

Figura 6: Report di utilizzo

Come rilevato dal report di sintesi, il componente è correttamente sintetizzabile con un totale di 124 LUT (Look Up Table) e 100 FF (Flip Flop).

3.1.2 Timing report

"Report Timing" permette di analizzare la velocità della computazione del componente rispetto un constraint di clock fornito.

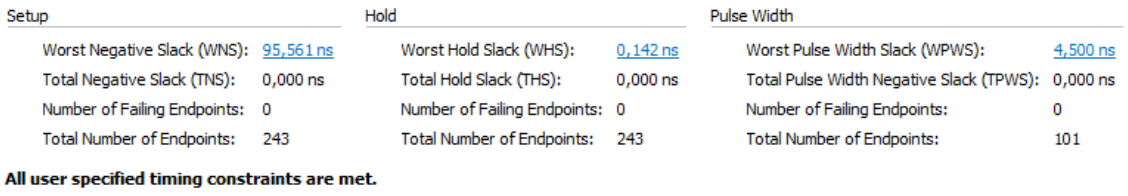


Figura 7: Timing Report

Il componente sintetizzato permette di stare notevolmente al di sotto del costraint di clock richiesto da specifica, rendendo possibile garantire il suo funzionamento anche per costraint fino a 10ns.
Si è ottenuto con il clock della specifica di 100ns un Worst Negative Slack pari a 95,561 ns.
Inoltre da "Report Timing Summary" si è ottenuto:

Paths	**Slack (MET) (arrival time - required time)**
Max Delay Paths	95.615
Min Delay Paths	1.087

3.2 Warnings Post-Synthesis

I warnings generati dal tool di sintesi durante lo sviluppo tra cui i warnings per latch inferiti e warnings per signal presenti nel processo ma non inseriti nella sensitivity list sono del tutto assenti.

3.3 Simulazioni

3.3.1 Test bench 0, example (fornito)

In questo test bench fornito dal prof. William Fornaciari viene provato un caso normale, senza casi limite.
La seguente tabella mostra un esempio del contenuto della memoria al termine dell' elaborazione. I valori che qui sono rappresentati in decimale, sono memorizzati in memoria con l'equivalente codifica binaria su 8 bit senza segno.

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	2	Numero di parole da elaborare
1	162	Prima parola da codificare
2	75	Seconda parola da codificare
[...]	...	...
1000	209	Primo Byte della sequenza di uscita
1001	205	Secondo Byte della sequenza di uscita
1002	247	Terzo Byte della sequenza di uscita
1003	210	Quarto Byte della sequenza di uscita

SEQUENZA IN INGRESSO	10100010		01001011	
SEQUENZA IN USCITA	11010001	11001101	11110111	11010010

Qui accanto viene riportata una breve legenda per permettere una più semplice lettura delle waveforms dei test bench:

OBJECT	color
CLOCK	Gray
ENTITY SIGNAL	Green
ADDRESS	Aqua
OUTPUT VALUE	Gold
FSM_STATE	olive
INTERNAL SIGNAL (INT)	Pink
INTERNAL SIGNAL (VECT)	Magenta

Di seguito è invece mostrata la waveform del suddetto test bench, che rappresenta il processo di convoluzione di 2 parole.
Il tutto funziona come richiesto da specifica e il test risulta essere passato con un tempo impiegato di 3850 ns.

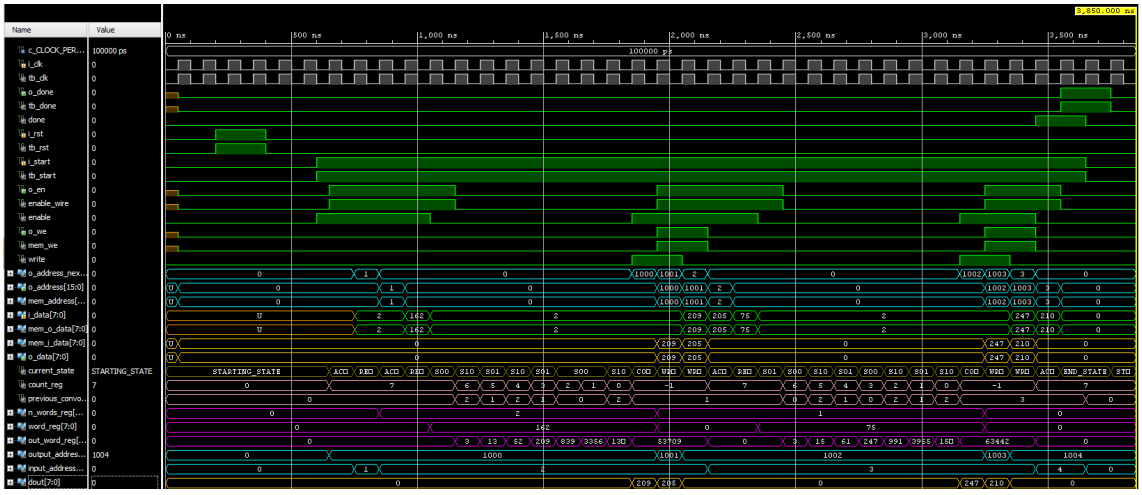


Figura 8: tb_example

3.3.2 Test bench custom

Di seguito tratteremo sinteticamente test bench custom, realizzati con lo scopo di analizzare casi limite e ogni possibile scenario particolare.
Sono stati effettuati anche altri test bench che trattavano casi generali, in particolare due sono stati prodotti a partire dagli esempi forniti nel file "PFRL_Specifica_21_22", questi sono stati omessi dalla relazione poichè facenti parte di casi non limite ma va tenuto presente che anch'essi sono stati portati a termine con esito positivo.

3.3.2.1 Test Bench 1, zero parole

Con questo test bench si è cercato di analizzare la situazione in cui venisse richiesto di elaborare 0 parole, andando così a verificare che la macchina a stati termini l'esecuzione passando direttamente da **ACCESS_MEM_WORD** a **END_STATE** senza mai entrare in uno stato di convoluzione o scrittura, così com'è stato previsto dal funzionamento ideato per la macchina.

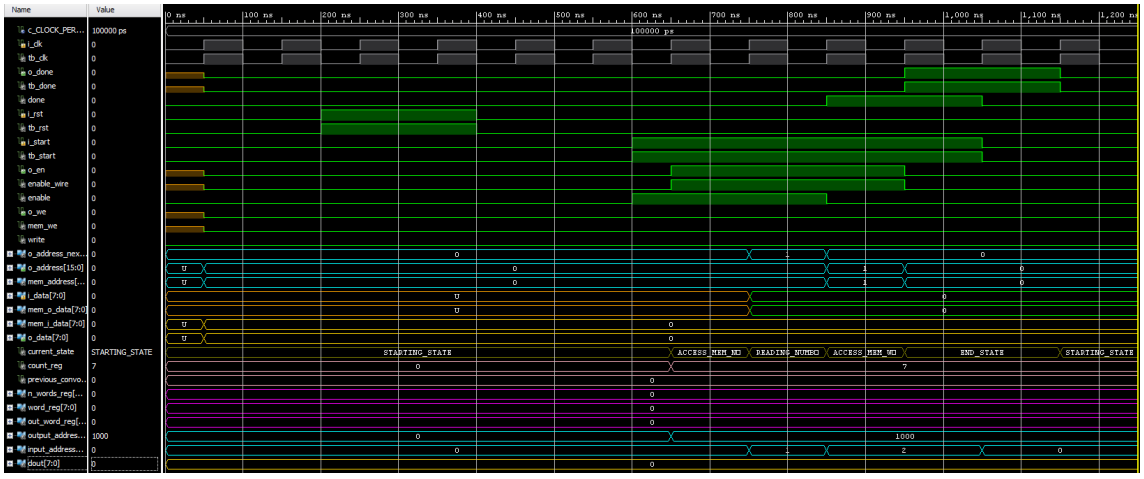


Figura 9: tb_zero_words

3.3.2.2 Test Bench 2, massima sequenza di parole

In questo test bench si è andati a verificare il caso limite in cui sia richiesta l'elaborazione di 255 parole, cioè la sequenza massima possibile in ingresso.
Per la scrittura di questo test bench si è sfruttato un semplice programma C che ha permesso di generare codice VHDL per l'analisi di 255 parole uguali del valore di 162.
Al fine di facilitare la lettura della wave form viene riportata solo la situazione a fine elaborazione, così che sia possibile vedere che essa viene portata a termine in modo corretto.

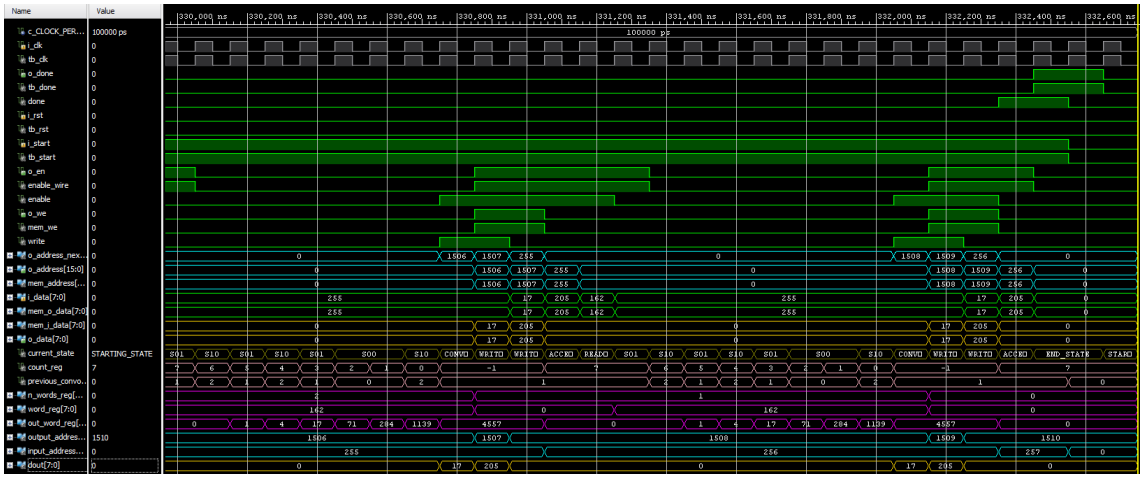


Figura 10: tb_zero_max_words

3.3.2.3 Test Bench 3, reset ed esecuzioni multiple

Il seguente test analizza il comportamento in presenza di 4 reset consecutivi (1 iniziale e 3 a inizio di ogni nuova elaborazione) e successivamente di altre 2 elaborazioni prive di un reset iniziale. Nella wave form sono stati evidenziati i segnali di reset e start rispettivamente in Red e Blue.

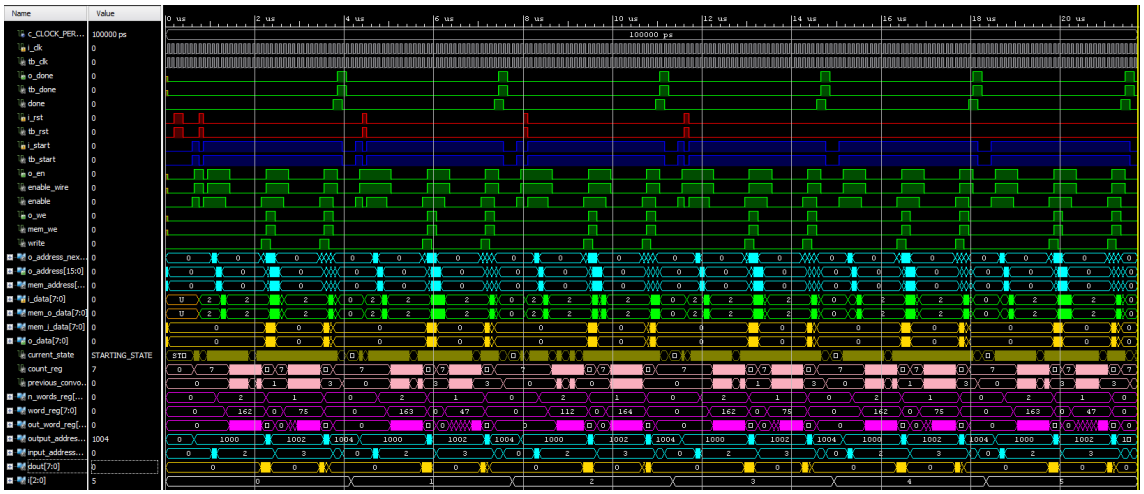


Figura 11: tb_resets_and_restarts

Qui di seguito è riportata un'immagine che ritrae le diversi fasi di inizio e fine di ciascuna delle 6 elaborazioni che sono state portate a termine.

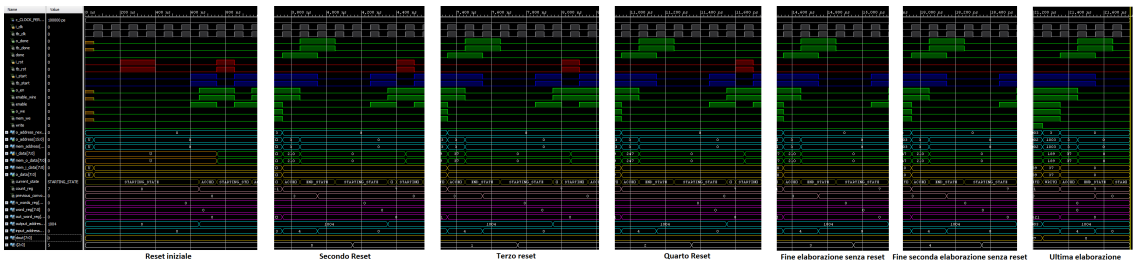


Figura 12: tb_resets_and_restarts_fasi

Il test ha dato esito positivo per ogni elaborazione conclusa ed è stato riportato un funzionamento coerente a quanto previsto.

4 Conclusioni

Si è prodotto un design che presenta le seguenti caratteristiche:

- Funzionante in `Behavioral`, `Post-Synthesis Functional` e `Post-Synthesis Timing`, privo di warning o latch.
- Numero di stati ridotto al minimo, con l'eccezione di **CONVOLUTION_END** per quanto specificato prima, che comunque non va a influire particolarmente sulle prestazioni (un ciclo di clock aggiuntivo per parola), permettendo comunque una ragionevole ottimizzazione e che in un contesto diverso da quello didattico risulterebbe facilmente omettibile ottenendo cosi una macchina a stati quanto più ottimizzata possibile e perfettamente funzionante.
- Lettura e scrittura della `RAM` ottimizzate in modo che siano eseguite solo quando strettamente necessario.
- Design che rispetti tutte le richieste fornite da specifica.
- Utilizzo di LUT pari a 0.09% e di FF pari a 0.04%.