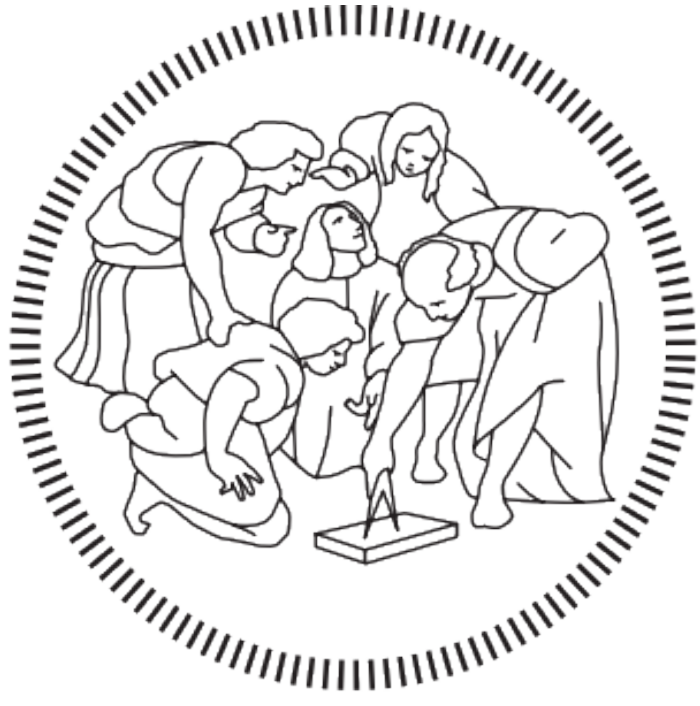
ANNO ACCADEMICO 2021/22

PROF. GIANLUCA PALERMO

MATTEO SPREAFICO
10669138
932096

LUDOVICA TASSINI
10663137
938238



POLITECNICO
MILANO 1863

PROVA FINALE

(PROGETTO DI RETI LOGICHE)

1. INTRODUZIONE AL PROGETTO

1.1 COMPOSIZIONE DEL GRUPPO

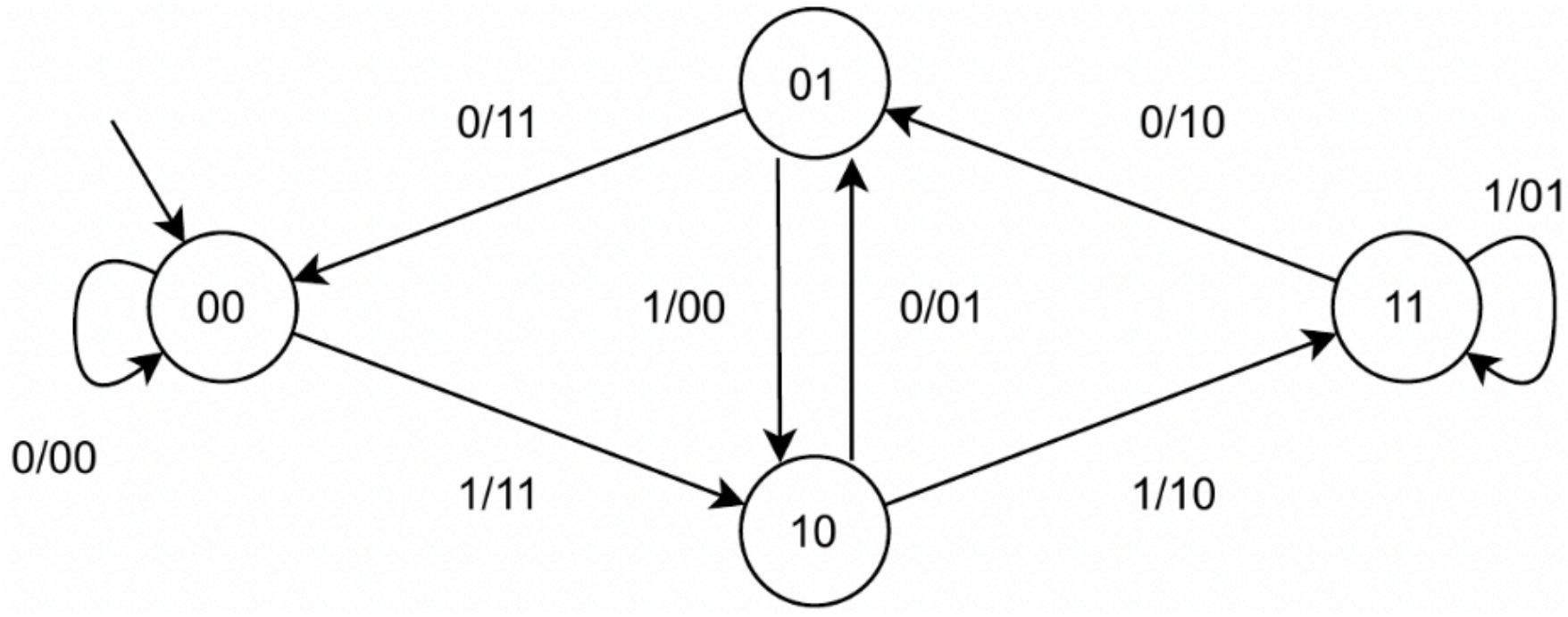
Tutto il materiale consegnato, ovvero il file VHDL e la seguente relazione, è stato realizzato dagli studenti Matteo Spreafico (codice persona 10669138, matricola 932096) e Ludovica Tassini (codice persona 10663137, matricola 938238).

1.2 SPECIFICA DEL PROGETTO

La specifica del progetto chiede di descrivere in linguaggio VHDL un componente hardware che si interfacci con una memoria e che, una volta serializzata la parola (di dimensione pari a un byte, cioè 8 bit) in input, applichi su tale flusso di bit il *codice convoluzionale 1/2* (cioè, ogni bit viene codificato con due bit). Dopo tale operazione, il componente parallelizza, su 8 bit, il flusso continuo in uscita dal codificatore convoluzionale e lo scrive in memoria. Il componente, inoltre, deve funzionare con un periodo di clock di almeno 100 ns.

1.3 CODIFICATORE CONVOLUZIONALE

Il *codificatore convoluzionale* (o, più semplicemente, *convolutore*) è una macchina sequenziale sincrona con un clock globale e un segnale di reset, il cui diagramma degli stati è il seguente:



Si noti che “00” è il suo stato iniziale, e che ogni transazione è annotata come $u_k/p_{1k}p_{2k}$, dove u_k è il k-esimo bit in ingresso e $p_{1k}p_{2k}$ sono i k-esimi bit in uscita.

1.4 STRUTTURA DELLA MEMORIA

Per la lettura e la scrittura delle parole, il componente si interfaccia con una memoria con indirizzamento al byte in cui le parole sono memorizzate. Ai fini dell'interazione, le celle della memoria sono divise in questo modo:

- la cella 0 contiene il numero di parole che il modulo deve leggere. Tale numero, come da indicazione, non può superare 255.
- le celle dalla 1 alla 255 contengono le parole da leggere.
- le celle dalla 1000 alla 1509 conterranno le parole in uscita dal convolutore.

1.5 INTERFACCIA DEL COMPONENTE

Il componente da descrivere ha la seguente interfaccia:

```
entity project_reti_logiche is
    port (
        -- Inputs
        i_clk : in STD_LOGIC;                           -- Clock signal
        i_rst : in STD_LOGIC;                           -- Reset signal
        i_start : in STD_LOGIC;                         -- Start signal
        i_data : in STD_LOGIC_VECTOR(7 downto 0);       -- Input data
        -- Outputs
        o_address : out STD_LOGIC_VECTOR(15 downto 0);  -- Output data address
        o_done : out STD_LOGIC;                         -- Done signal
        o_en : out STD_LOGIC;                           -- Memory enable signal
        o_we : out STD_LOGIC;                           -- Memory mode signal: 0 read, 1 write
        o_data : out STD_LOGIC_VECTOR(7 downto 0)       -- Output data
    );
end project_reti_logiche;
```

In particolare:

- `i_clk` è il segnale di clock in ingresso.
- `i_rst` è il segnale che inizializza la macchina, affinché sia pronta a ricevere il primo segnale di start.
- `i_start` è il segnale che avvia le operazioni di lettura.
- `i_data` è il segnale (vettore) che arriva dalla memoria in seguito a una richiesta di lettura.
- `o_address` è il segnale (vettore) di uscita che invia l'indirizzo alla memoria.
- `o_done` è il segnale di uscita che comunica la fine delle operazioni di scrittura dei dati elaborati.

- `o_en` è il segnale da dover inviare alla memoria per poter avviare la comunicazione, sia in lettura che in scrittura.
- `o_we` è il segnale da dover alzare a 1 e da dover inviare alla memoria per permettere le operazioni di scrittura. Quando si desidera leggere dalla memoria, tale segnale deve essere a 0.
- `o_data` è il segnale (vettore) di uscita dal componente verso la memoria.

1.6 DETTAGLI SULL'UTILIZZO DEL COMPONENTE

Il componente comincia l'elaborazione dei dati quando il segnale `i_start` viene portato a 1, e termina quando il segnale `o_done` viene portato a 1. Quest'ultimo segnale viene alzato una volta terminata la scrittura in memoria, e rimane alto fino a quando il segnale `i_start` viene riportato a 0. Il segnale `i_start` non può essere riportato a 1 finché `o_done` non viene riportato a 0. Una volta che tale condizione è soddisfatta, rialzare il segnale `i_start` significa far ripartire l'elaborazione.

Il componente è progettato per poter codificare più flussi, uno dopo l'altro. Ad ogni nuova elaborazione, il convolutore viene riportato nel suo stato iniziale "00" (che è anche quello di reset). La quantità di parole da codificare sarà sempre memorizzata all'indirizzo 0, le (eventuali) parole da leggere saranno sempre memorizzate dall'indirizzo 1 fino al massimo all'indirizzo 255 e l'uscita deve sempre essere memorizzata a partire dall'indirizzo 1000.

Il componente è stato progettato considerando che prima della prima codifica verrà sempre dato il segnale di reset al modulo. Tuttavia, una seconda elaborazione non dovrà attendere il reset del modulo, ma solo la terminazione della prima elaborazione.

1.7 ESEMPIO DI FUNZIONAMENTO DEL COMPONENTE

In generale, ad ogni elaborazione regolare, la macchina deve:

1. leggere quante parole deve codificare.
2. leggere una parola da codificare.
3. applicare l'algoritmo di convoluzione su tale parola, producendone due in output.
4. scrivere le due parole in memoria.
5. verificare se restano altre parole da codificare. Se sì, la sequenza di azioni riparte dal punto 2; altrimenti, l'elaborazione è terminata.

Per illustrare meglio il funzionamento del componente, si analizzerà ora il testbench d'esempio. Tale testbench descrive una memoria RAM che inizialmente è così formata:

INDIRIZZO CELLA DI MEMORIA	CONTENUTO
0	2
1	162
2	75
da 3 a 65535	0

La macchina, innanzitutto, legge il numero 2, dunque apprende che deve leggere dalla memoria due parole. Legge perciò la prima parola, la salva in un registro interno e passa al convolutore un bit alla volta.

La prima parola letta è 162, in binario 10100010. La macchina a stati del convolutore ha come stato di partenza "00"; dato che il primo bit a partire da destra della parola letta è 1, il prossimo stato raggiunto dal convolutore è "10", mentre i primi due bit di output sono 11. Il secondo bit della parola è 0, dunque il prossimo stato che il convolutore raggiunge è "01", e i prossimi due bit in uscita sono 01. Il processo si ripete fino a quando ogni bit della parola non è stato codificato. L'output finale è lungo 16 bit ed è 1101000111001101; dividendolo in due metà, si ottengono le due parole da scrivere in memoria, ovverosia 11010001 e 11001101 (in decimale, rispettivamente, 209 e 205), nelle celle di indirizzo 1000 e 1001.

Una volta scritte, la macchina sa che deve ancora leggere una parola, pertanto esegue una nuova richiesta di lettura alla memoria e legge la seconda parola, che è 75 (in binario 01001011). Il convolutore non partirà più necessariamente dallo stato di partenza "00", bensì dallo stato che avrebbe dovuto raggiungere se all'ultimo bit codificato ne fosse seguito un altro. Il processo convoluzionale di una parola, infatti, tiene conto del (eventuale) processo precedente, dunque - per permettere la scrittura in memoria dei risultati - è necessario "congelare" tale processo e tenere traccia dello stato da cui ripartire. Nel caso specifico, lo stato da cui riparte la codifica è lo "01". Il nuovo output viene diviso secondo la medesima regola del numero precedente, e le parole così ottenute 11110111 e 11010010 (in decimale, rispettivamente, 247 e 210) vengono scritte in memoria agli indirizzi 1002 e 1003.

Una volta lette due parole, la macchina ignora qualsiasi parola eventualmente scritta dalla cella 3 in poi, e termina l'elaborazione. La memoria RAM appare ora come segue:

INDIRIZZO CELLA DI MEMORIA	CONTENUTO
0	2
1	162
2	75
da 3 a 999	0

1000	209
1001	205
1002	247
1003	210
da 1003 a 65535	0

2. ARCHITETTURA DEL COMPONENTE

2.1 SCELTE PROGETTUALI

Il gruppo ha deciso di descrivere il componente mediante una macchina a stati finiti. La soluzione scelta prevede un'architettura monomodulare composta da un singolo *process* sensibile al segnale di clock. La macchina a stati che descrive l'algoritmo convoluzionale è stata integrata, per via della sua complessità ridotta, nella macchina a stati principale dell'architettura.

La macchina a stati è stata sviluppata gradualmente. Inizialmente, si è verificato che la macchina leggesse correttamente dalla memoria, mediante un testbench creato ad hoc. In questa prima fase, sono stati creati e testati gli stati `START`, `ENABLE_MEMORY`, `READ_SIZE`, `DONE` ed uno stato di attesa. In seconda battuta, dopo aver aggiunto lo stato `READ_WORD` ed averne controllato il corretto funzionamento, è stato creato lo stato necessario per scrivere in memoria (`WRITE_WORD`). In questa fase, sono state scritte le parole lette dalla memoria, senza alcun tipo di elaborazione. Infine, è stato aggiunto lo stato `CONV_CONTROLLER`, che controlla il processo convoluzionale, e gli stati `CONVOLUTION_00`, `CONVOLUTION_01`, `CONVOLUTION_10`, `CONVOLUTION_11`, che si occupano dell'effettiva produzione dell'output. Questo viene poi passato allo stato `WRITE_WORD`, il quale si occupa di dividerlo correttamente in due parole da 8 bit ciascuna e di scriverle in memoria.

2.2 STATI DELLA MACCHINA

La macchina è composta da 12 stati:

1. `START`. È lo stato da cui la macchina parte per effettuare l'elaborazione. In esso, vengono resettati tutti i suoi registri. Quando il modulo riceve il segnale di `i_rst`, infatti, torna a questo stato.
2. `ENABLE_MEMORY`. È lo stato in cui la macchina decide, in base ai valori dei registri `has_words_number` e `words_number`, se richiedere la lettura da memoria o saltare direttamente allo stato finale (questo è il caso particolare in cui l'elaborazione parte ma non vi sono parole da elaborare, cioè il modulo legge 0 alla cella 0 della memoria). Se la lettura viene richiesta, aggiorna opportunamente l'indirizzo di memoria da cui leggere.
3. `WAITING_STATE`. È lo stato di attesa della macchina. Lo stato raggiunto al ciclo di clock successivo dipende dal valore contenuto in alcuni registri:
   a. se il booleano `word_switch` contiene `false`, significa che si vuole attendere la risposta della memoria in seguito a una richiesta di lettura. Se il registro

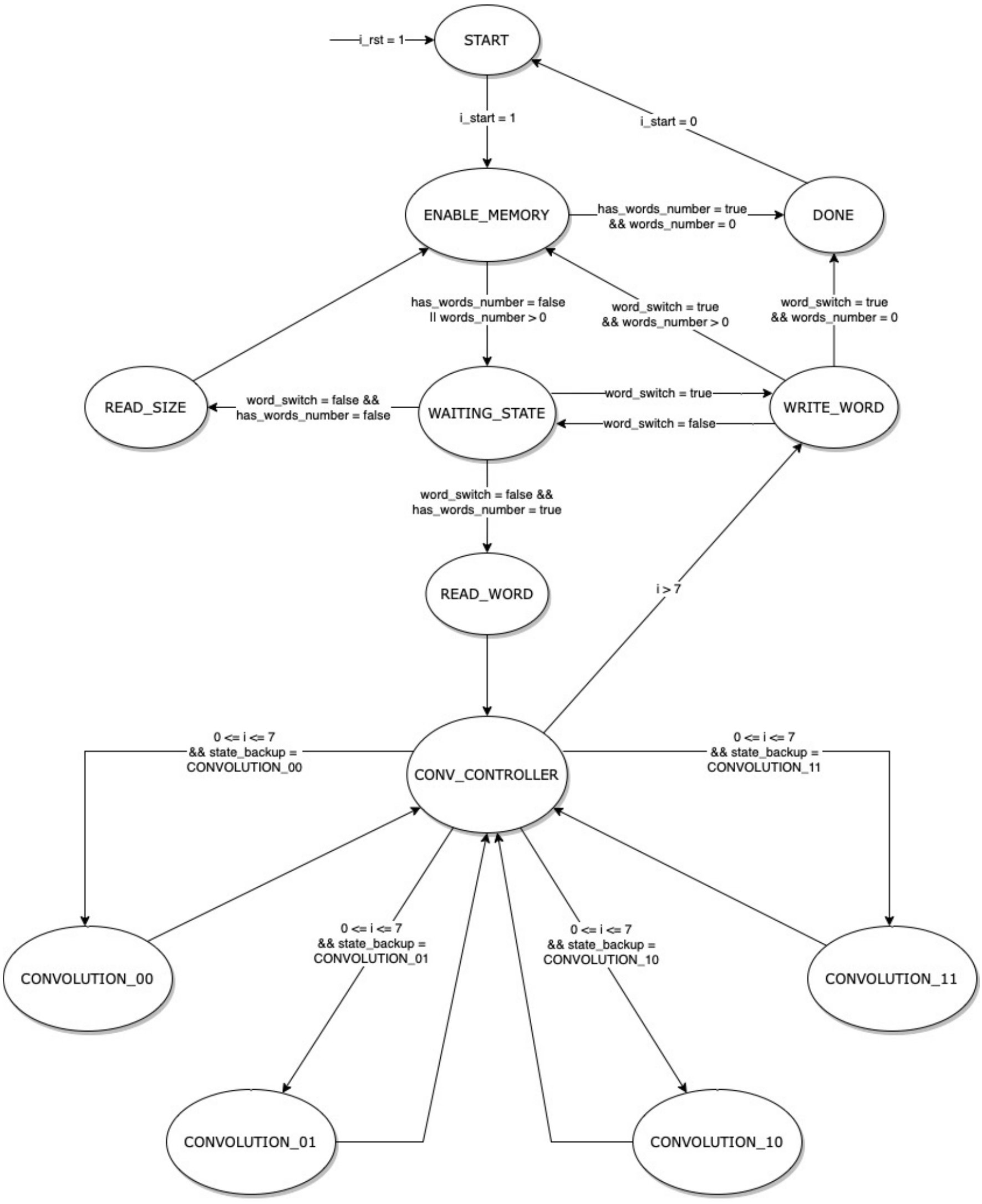
has_words_number contiene il valore false, si procede verso lo stato READ_SIZE (dunque si sta per leggere la quantità di parole da elaborare), altrimenti si procede verso lo stato READ_WORD (dunque si sta leggendo una parola da elaborare).

   b. se il booleano word_switch contiene true, significa che si sta attendendo che la memoria sia pronta per la scrittura di una nuova parola, e dunque si procede verso WRITE_WORD.

4. READ_SIZE. È lo stato in cui la macchina legge alla cella 0 della memoria quante parole dovranno essere processate; aggiorna di conseguenza il registro words_number.
5. READ_WORD. È lo stato in cui la macchina legge dalla memoria le parole da processare, salvandole progressivamente nel registro current_word.
6. CONV_CONTROLLER. È uno stato in cui, in base al valore dell'iteratore i, la macchina decide, se la parola letta non è stata ancora completamente processata (cioè, i è minore o uguale a 7), di proseguire con la convoluzione, oppure (i maggiore di 7) di passare allo stato WRITE_WORD.
7. CONVOLUTION_00. È uno degli stati del codificatore convoluzionale, come mostrato nella sezione 1.3 della documentazione.
8. CONVOLUTION_01. È uno degli stati del codificatore convoluzionale, come mostrato nella sezione 1.3 della documentazione.
9. CONVOLUTION_10. È uno degli stati del codificatore convoluzionale, come mostrato nella sezione 1.3 della documentazione.
10. CONVOLUTION_11. È uno degli stati del codificatore convoluzionale, come mostrato nella sezione 1.3 della documentazione.
11. WRITE_WORD. È lo stato in cui la macchina scrive in memoria il risultato della convoluzione. Se ha inviato in uscita la prima delle due parole prodotte dall'elaborazione, deve attendere che essa venga scritta in memoria, dunque procede verso WAITING_STATE cambiando (dopo aver superato il controllo word_switch = false) il valore di word_switch in true. Altrimenti, se ha inviato in uscita la seconda parola, riporta word_switch a false e controlla il registro words_number: se è maggiore di 0 (ovverosia, ci sono altre parole da processare), torna in ENABLE_MEMORY, altrimenti passa allo stato DONE.
12. DONE. È lo stato in cui la macchina si trova quando ha terminato l'elaborazione.

Di seguito, viene riportato il grafo che rappresenta la macchina a stati descritta; vengono anche riportati i valori dei registri che permettono di entrare nei rami if che consentono il passaggio della macchina da un dato stato a un altro ad esso collegato. Per ulteriore

chiarezza del grafo, si ribadisce che, in qualsiasi stato ci si trovi, all'arrivo del segnale di reset (`i_rst` = 1) la macchina torna allo stato `START`.



2.3 REGISTRI DELLA MACCHINA

La macchina possiede 11 registri che mantengono informazioni necessarie per il corretto svolgimento dell'elaborazione:

```
-- Current state register
signal state : STATE_TYPE := START; -- contains the machine's current state

-- I/O stream address registers
signal i_stream_address : STD_LOGIC_VECTOR(15 downto 0) := (others => '0'); -- address to read from
signal o_stream_address : STD_LOGIC_VECTOR(15 downto 0) := "0000001111101000"; -- address to write to

-- Words registers
signal has_words_number : BOOLEAN := false;
signal words_number : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal current_word : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');

-- Convolution registers
signal i : INTEGER range 0 to 8 := 0;
signal u : STD_LOGIC := '0';
signal Z : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
signal state_backup : STATE_TYPE := CONVOLUTION_00;

-- Writing process register
signal word_switch : BOOLEAN := false;
```

In particolare:

- `state` indica in quale stato si trova la macchina.
- `i_stream_address` contiene l'indirizzo della cella di memoria da cui leggere, e viene aggiornato e comunicato alla memoria opportunamente.
- `o_stream_address` contiene l'indirizzo della cella di memoria su cui scrivere, e viene aggiornato e comunicato alla memoria opportunamente.
- `has_words_number` indica se la macchina conosce il numero di parole che deve elaborare.
- `words_number` indica quante parole devono essere ancora elaborate dalla macchina (è al suo valore massimo quando la macchina si trova nello stato `READ_SIZE`, e viene decrementato man mano che le parole vengono effettivamente lette nello stato `READ_WORD`).
- `current_word` contiene la parola che la macchina sta elaborando.
- `i` è un iteratore che permette alla macchina di prendere da `current_word` il giusto bit da inviare all'algoritmo di convoluzione.
- `u` contiene l'`i`-esimo bit di `current_word`, contati a partire dal bit più significativo, e rappresenta l'ingresso della macchina a stati del convolutore (descritta nella sezione 1.3).

- `Z` contiene l'intero output (di dimensione 16 bit) del convolutore.
- `state_backup` contiene lo stato dal quale far ripartire il processo di convoluzione, qualora l'elaborazione di una parola finisca ma rimangano in memoria delle parole da processare.
- `word_switch` indica se la macchina sta salvando in memoria i primi 8 bit o gli ultimi 8 bit di `Z`.

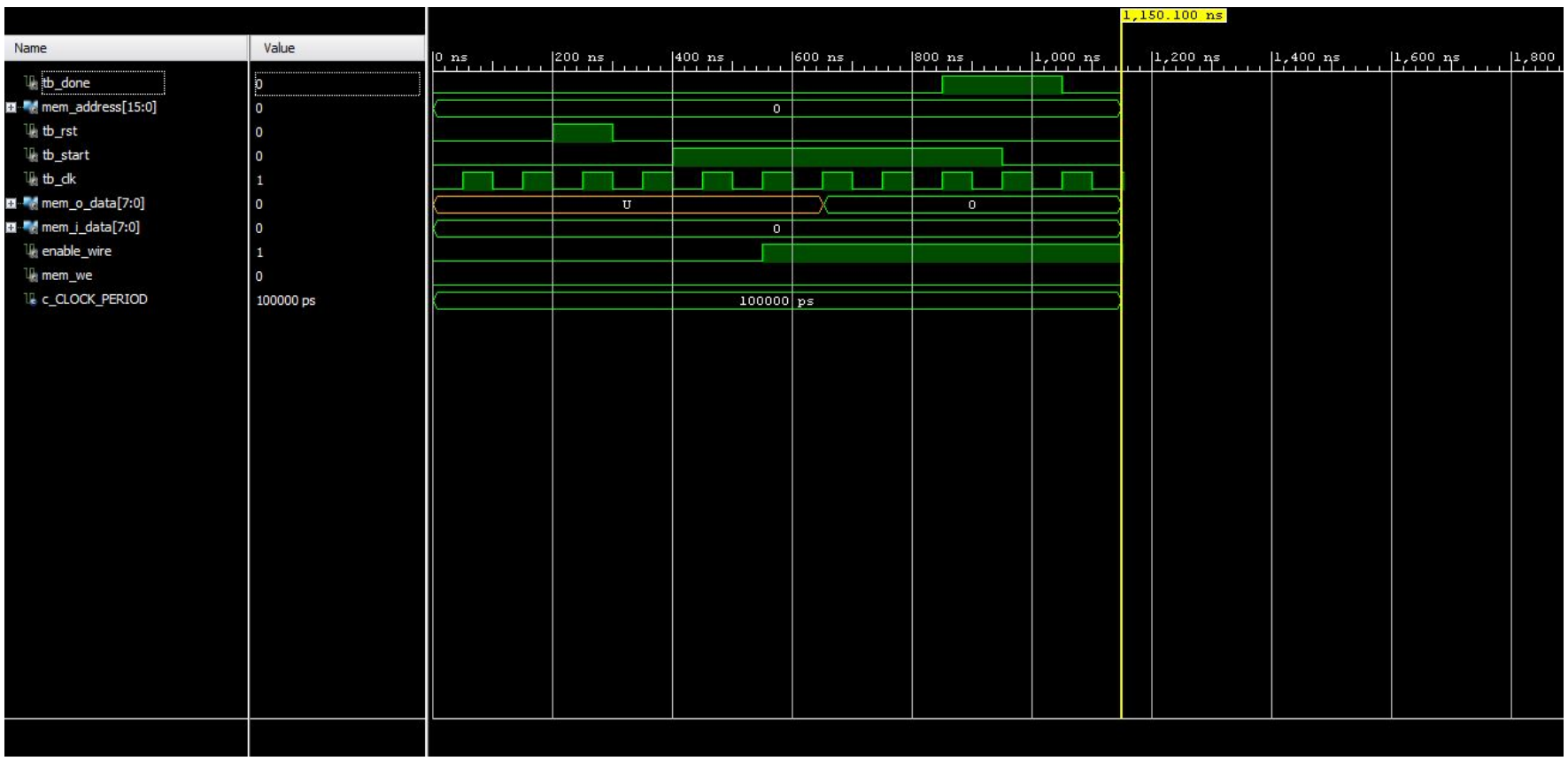
3. RISULTATI SPERIMENTALI

3.1 SIMULAZIONI SIGNIFICATIVE

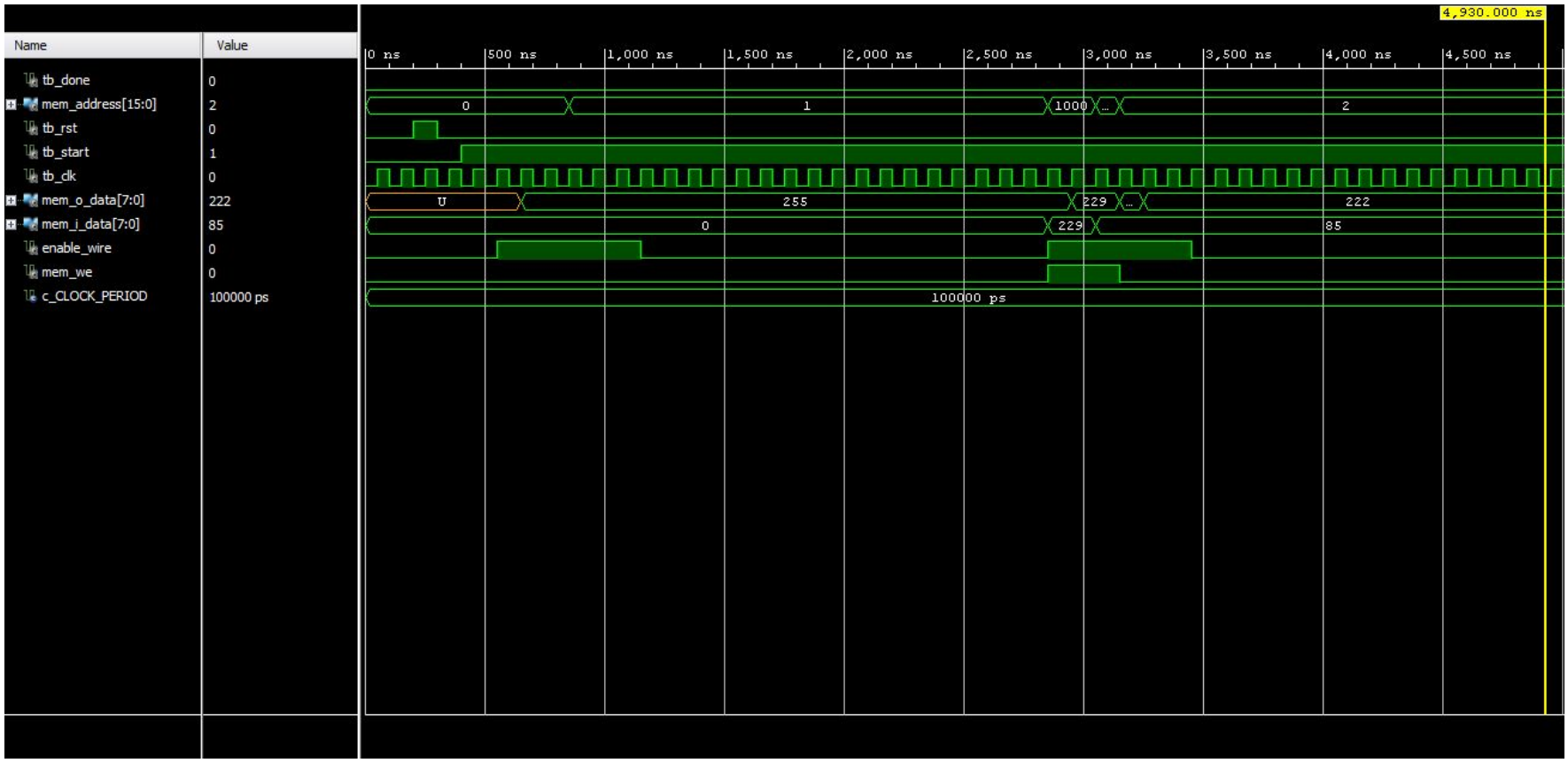
Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con il testbench di esempio, abbiamo definito altri casi di test in modo da cercare di massimizzare la copertura di tutti i possibili cammini che la macchina può effettuare durante la computazione. Di seguito è fornita una breve descrizione di alcuni dei test più significativi utilizzati, e per quelli più rilevanti viene anche mostrato l'effettivo funzionamento del componente grazie allo screenshot dell'andamento dei segnali durante la simulazione.

I cinque test che il gruppo ha ritenuto più significativi sono:

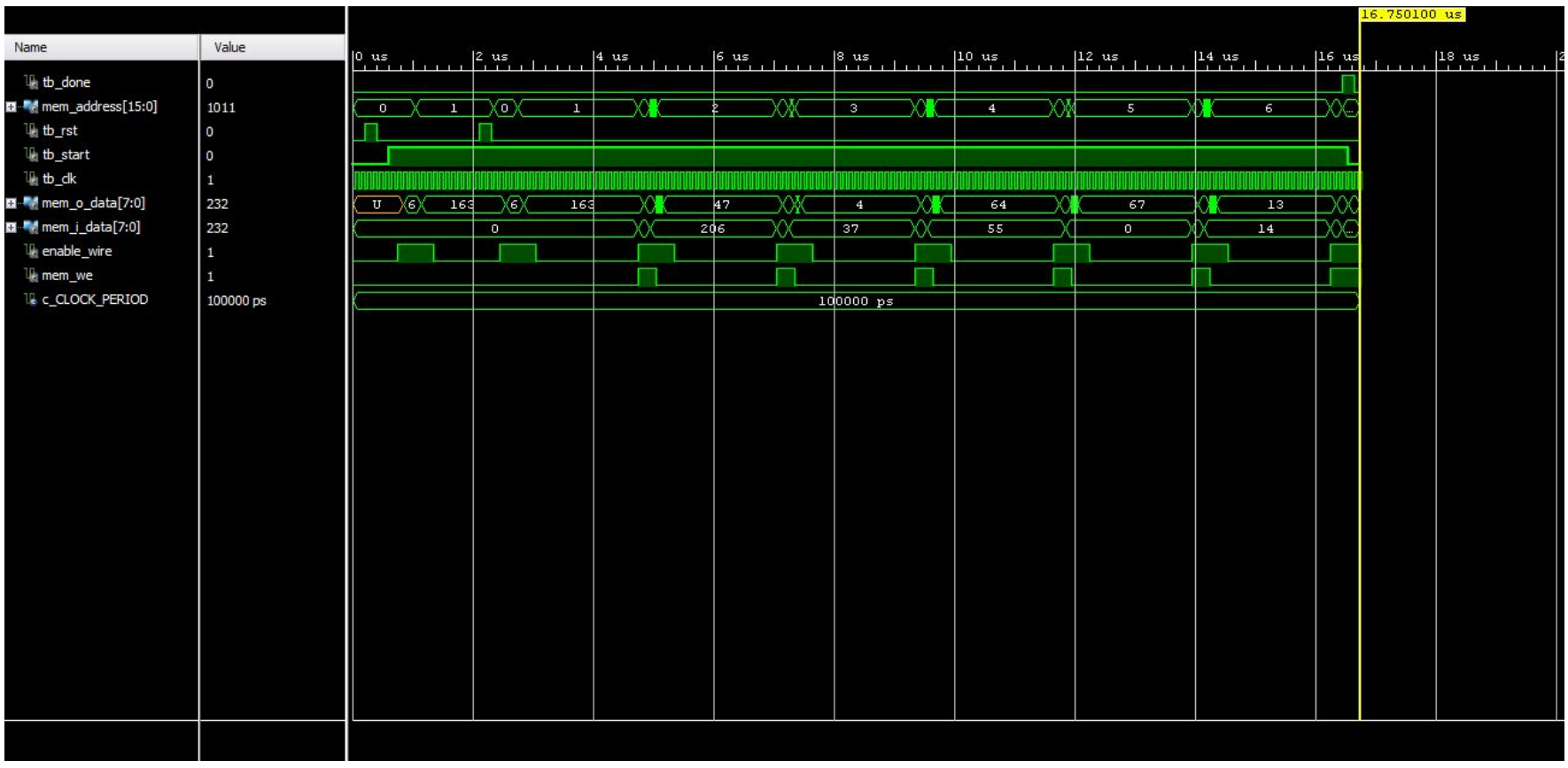
1. **SEQUENZA DI INGRESSO MINIMA:** il numero di parole da leggere è pari a zero. Il test verifica che la macchina, dopo aver letto il numero di parole da leggere, termini immediatamente l'elaborazione.



2. **SEQUENZA DI INGRESSO MASSIMA:** il numero di parole da leggere è il massimo. Il test verifica che la macchina, dopo aver letto il numero di parole da leggere (255), prosegua nella lettura e nella convoluzione per tutte le parole, fino a giungere allo stato DONE.



3. **RESET ASINCRONO:** il test verifica che il trigger asincrono del segnale di reset non comprometta la computazione e che questa ricominci facendo ritornare la macchina nello stato iniziale di START.



4. **DOPPIA COMPUTAZIONE**: il test verifica la corretta sincronizzazione dei segnali `i_start`, `i_rst`, `o_done` andando a simulare due volte di fila la stessa memoria.
5. **MEMORIE MULTIPLE**: quest'ultimo test verifica che la macchina riesca a gestire più flussi provenienti da memorie diverse, uno dopo l'altro, scrivendo i risultati sulle corrispondenti memorie.

3.2 RISULTATI DELLA SINTESI

Il componente è risultato correttamente sintetizzabile e implementabile dal tool *Xilinx Vivado Webpack* con un totale di 184 lookup table e 105 flip-flop.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF
synth_1	constrs_1	synth_design Complete!							184	105
impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	0	184	105

Il componente sintetizzato supera correttamente tutti i test specificati nelle simulazioni *Behavioral* e *Post-Synthesis Functional*. Di seguito, è possibile vedere un confronto tra i tempi di simulazione dei due *corner case* che portano la macchina verso la più breve e la più lunga computazione:

	BEHAVIORAL	POST-SYNTHESIS FUNCTIONAL
SEQUENZA MINIMA	1050 ns	1150 ns
SEQUENZA MASSIMA	587450 ns	587550 ns

4. CONCLUSIONI

4.1 OTTIMIZZAZIONI

In un primo momento, fra gli stati `CONV_CONTROLLER` e `WRITE_WORD` era presente uno stato `CREATE_WORD`, dove l'output del convolutore veniva suddiviso in due parole da 8 bit ciascuna, che venivano poi alternativamente salvate in un registro, da cui lo stato `WRITE_WORD` leggeva per scrivere in memoria il corretto valore. Tutte queste operazioni sono ora svolte direttamente nello stato `WRITE_WORD`, senza passare per il suddetto registro (che è dunque stato rimosso per rendere la macchina più snella). Inoltre, il segnale `i`, utilizzato per indicizzare il segnale `current_word` e dichiarato come `INTEGER`, assume valori (nel contesto dell'elaborazione) compresi unicamente fra 0 e 8. Per evitare inutili sprechi, la dichiarazione del segnale è stata arricchita con l'aggiunta del `range` appropriato.

4.2 RAPPRESENTAZIONE A BLOCCHI DEL COMPONENTE

Infine, il modulo sintetizzato può essere anche descritto dal seguente diagramma a blocchi: