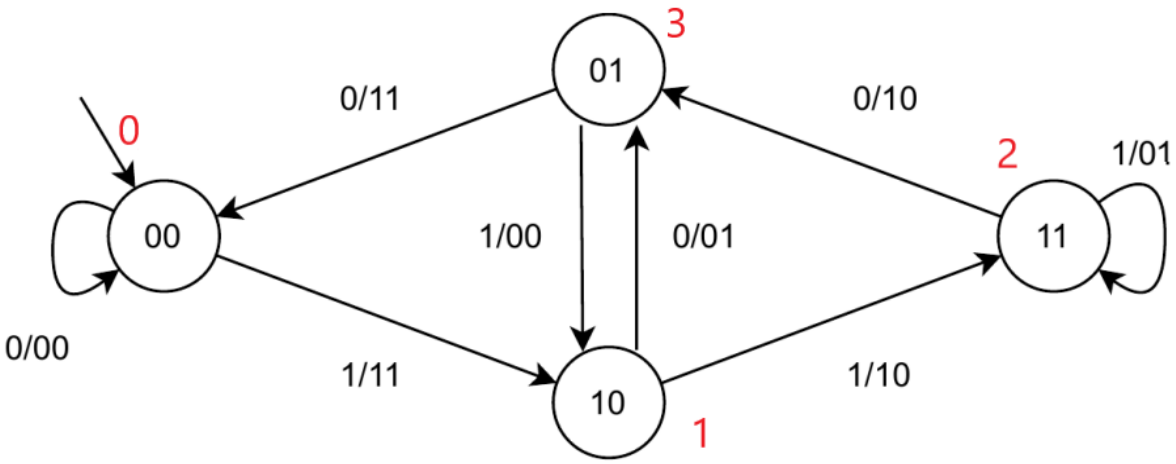
Progetto di Reti Logiche
Prof. William Fornaciari – Anno Accademico 2021/2022
Nicolò Avarino (Codice Persona 10667140 - Matricola 916653)

Ingegneria Informatica
Corso di Laurea Triennale - Milano Leonardo

Introduzione

L'obiettivo della "Prova Finale (Progetto di Reti Logiche)" 2021/2022 è quello di implementare un modulo HW (descritto in VHDL) che si interfacci con una memoria e che, ricevuta in ingresso una sequenza di W parole restituisca un flusso di Y parole generato da un convolutore.

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 (nodo 0) lo stato iniziale e con uscite in ordine PK1, PK2 le quali, poi, concatenando i bit di uscita genereranno il flusso Y.



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 U = 10100010
(viene serializzata 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 la tempo t+6 e 0 al tempo t+7)
Applicando l'algoritmo convoluzionale si ottiene la seguente serie di coppie di bit:

Time	0 1 2 3 4 5 6 7
U	1 0 1 0 0 0 1 0
P1k	1 0 0 0 1 0 1 0
P2k	1 1 0 1 1 0 1 1

Nello specifico il primo bit di U , in questo caso 1, entra nel nodo 0 del convolutore.
Seguendo il diagramma quindi p1k e p2k sono entrambi 1 e ci si sposta nel nodo 1.
In questo caso il bit di U analizzato è 0. P1k e p2k saranno quindi 0 e 1 e ci spostiamo nel nodo 3 e così via.
Importante ricordare che al termine della parola U lo stato del convolutore va salvato perché qualora dovesse presentarsi una seconda parola, bisognerà ripartire dallo stato finale della macchina e non dal nodo 0 iniziale.

Architettura

Interfaccia del componente

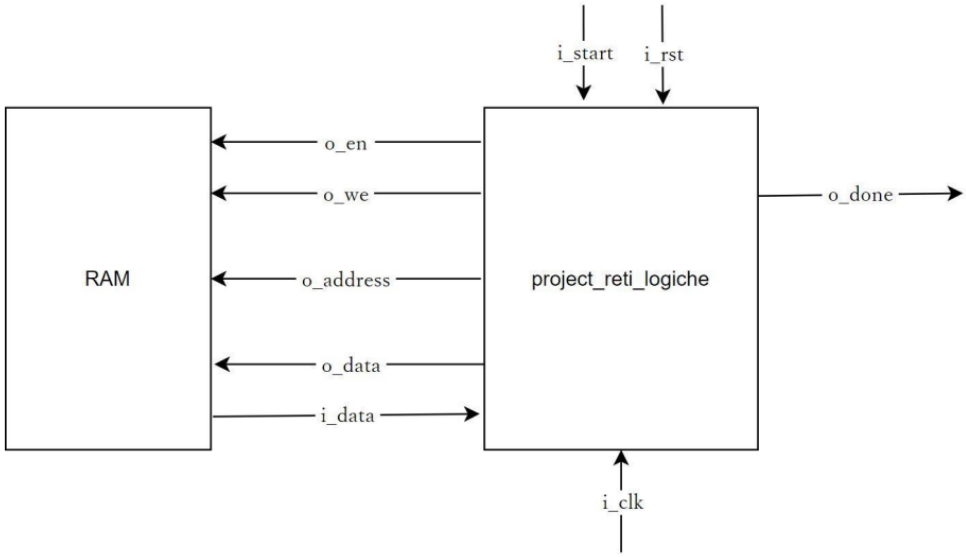
L'interfaccia del componente è la seguente :

```
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 :

Segnale	Descrizione
i_clk	Segnale di clock fornito in ingresso dal test bench
i_rst	Segnale di reset per preparare la macchina a ricevere la prima immagine
i_start	Segnale di start per iniziare ad eseguire le operazioni
i_data	Segnale di input che contiene i dati letti dalla memoria
o_address	Segnale di output che specifica l'indirizzo della memoria
o_done	Segnale di output per avvisare della fine dell'elaborazione
o_en	Segnale di enable per poter comunicare con la memoria
o_we	Segnale di write enable che indica l'operazione da svolgere in memoria: o_we = 0 ⇒ lettura o_we = 1 ⇒ scrittura
o_data	Segnale di uscita che contiene i dati inviati alla memoria

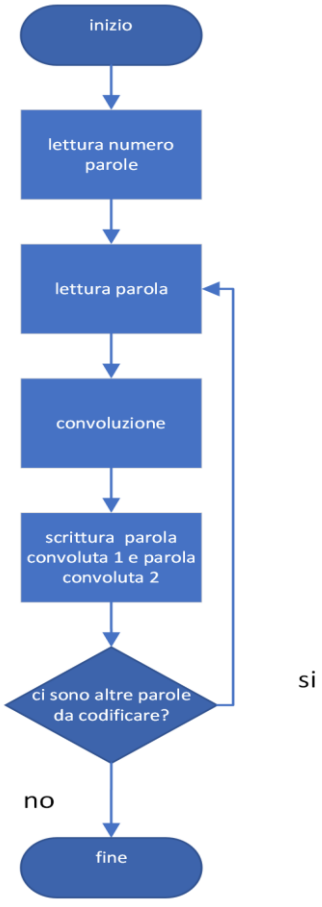
Si fornisce inoltre uno schema che illustra l'iterazione tra la RAM ed il modulo implementato attraverso i segnali precedentemente descritti :



Algoritmo

L'algoritmo utilizzato, raffigurato nel diagramma di flusso sottostante, si sviluppa secondo i seguenti passaggi:

- Inizializzazione delle variabili e successiva lettura del numero di parole che prenderà il nome di number_of_words.
- Lettura parola e convoluzione di essa. Ogni parola è un array binario. Viene quindi applicato un ciclo che a ogni iterazione prende un bit e ne applica la convoluzione generandone due in uscita e salvandoli in word_convoluted (segnale da 16 bit)
- Word_convoluted viene tagliata in due e fatta uscire nell'apposito indirizzo di memoria. Si decrementa number_of_words.
- Nel caso in cui ci fossero altre parole da codificare, l'algoritmo riprenderà a fare la convoluzione sulla nuova parola. Il controllo viene fatto verificando se number_of_words sia diverso da 0. In caso contrario termina l'algoritmo



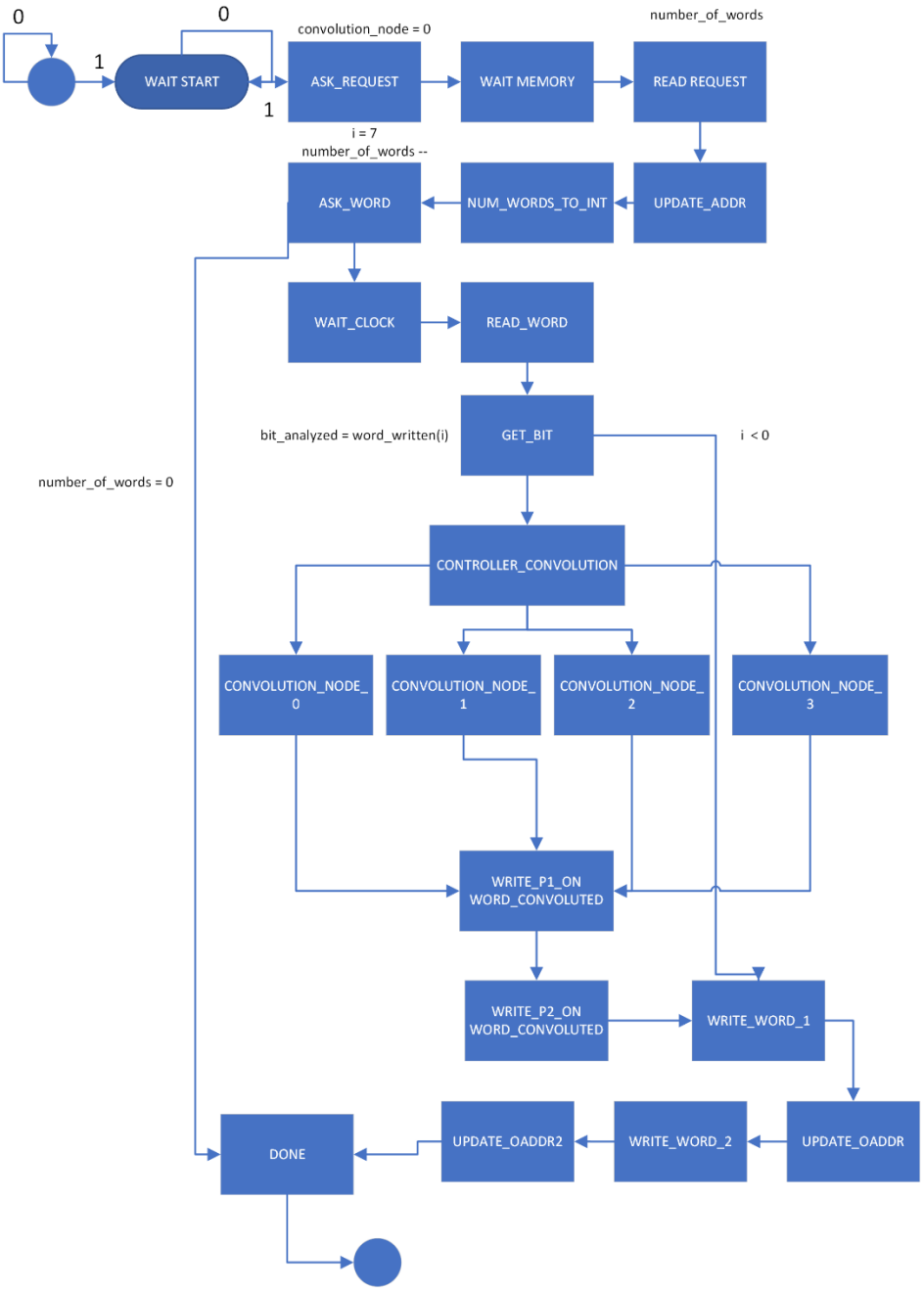
Si è deciso di sviluppare il componente richiesto come una macchina a stati finiti seguendo l'algoritmo descritto.
Nella descrizione di ogni stato verrà fatto riferimento ai seguenti segnali utilizzati:

```
SIGNAL next_state          : stato;
    SIGNAL curr_iaddress       : std_logic_vector(15 DOWNTO 0);
    SIGNAL curr_oaddress       : std_logic_vector(15 DOWNTO 0);
    SIGNAL number_of_words_std,word_written        : std_logic_vector(7 DOWNTO 0);
    SIGNAL word_convoluted     : std_logic_vector(15 DOWNTO 0);
    SIGNAL number_of_words, i,k,convolution_node : integer range 0 to 255;
    SIGNAL p1k             :  std_logic ;
    SIGNAL p2k             :  std_logic ;
    SIGNAL bit_analyzed    :  std_logic ;
```

Macchina a stati finiti

- *Stato iniziale:* attesa ricevimento del segnale i_rst = 1
- *WAIT_START* : attesa del segnale i_start = 1 per ricevere il numero di parole
- *ASK_REQUEST:* richiesta lettura della cella di memoria contentente il numero di parole
- *WAIT_MEMORY :* aspetto un ciclo di clock
- *READ_REQUEST* : lettura della cella di memoria contentente il numero di parole che salverò nel segnale num_of_words
- *UPDATE_ADDR :* incremento segnale indirizzo di memoria di input curr_iaddress
- *NUM_WORDS_TO_INT:* conversione del segnale num_of_words in un intero da decrementare a ogni lettura di parola.
- *ASK_WORD :* richiesta lettura della cella di memoria contentente parola
- *WAIT_CLOCK: aspetto un ciclo di clock*
- *READ_WORD:* lettura della cella di memoria contentente parola che salverò nel segnale word_written
- *GET_BIT:* salvo in un segnale bit_analyzed da word_written(i) con i inizializzato precedentemente a 7, se i < 0 vuol dire che ho finito di codificare la parola e devo scriverla.
-
- *CONTROLLER_CONVOLUTION:* stato di controllo della convoluzione, in base al valore del segnale intero convolution_node, che può assumere valori da 0 a 3, il prossimo stato sarà il corrispondente nodo di convoluzione.
- *CONVOLUTION_0*: assegna valori ai segnali: pk1 ,pk2 in base al valore di bit_analyzed, poi aggiorna il segnale convolution_node.
- *CONVOLUTION_1:* assegna valori ai segnali: pk1 ,pk2 in base al valore di bit_analyzed, poi aggiorna il segnale convolution_node
- *CONVOLUTION_2:* assegna valori ai segnali: pk1 ,pk2 in base al valore di bit_analyzed, poi aggiorna il segnale convolution_node
- *CONVOLUTION_3:* assegna valori ai segnali: pk1 ,pk2 in base al valore di bit_analyzed, poi aggiorna il segnale convolution_node

- *WRITE_P1k_ON_WORD_CONVOLUTED:* scrivo il valore di p1k sul segnale word_convoluted.
- *WRITE_P2k_ON_WORD_CONVOLUTED*: scrivo il valore di p1k sul segnale word_convoluted e passo al bit successivo tornando allo stato GET_BIT
- *WRITE_WORD_1*: scrivo la parola 1 prendendone gli ultimi 8 bit.
- *UPDATE_OADDR:* incremento segnale indirizzo di memoria di input curr_oaddress
- *WRITE_WORD_2:* scrivo la parola 2 prendendone i primi 8 bit.
- *UPDATE_OADDR2:* incremento segnale indirizzo di memoria di input curr_oaddress
- *DONE: stato in cui si porta o_done a 1 e si attende che il segnale i_start venga riportato a 0 per riportare a sua volta o_done a 0;*



Risultati sperimentali

Risultati Sintesi

Il comando report utilization evidenzia l'assenza di Latch

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	95	0	134600	0.07
LUT as Logic	95	0	134600	0.07
LUT as Memory	0	0	46200	0.00
Slice Registers	70	0	269200	0.03
Register as Flip Flop	70	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	1	0	67300	<0.01
F8 Muxes	0	0	33650	0.00

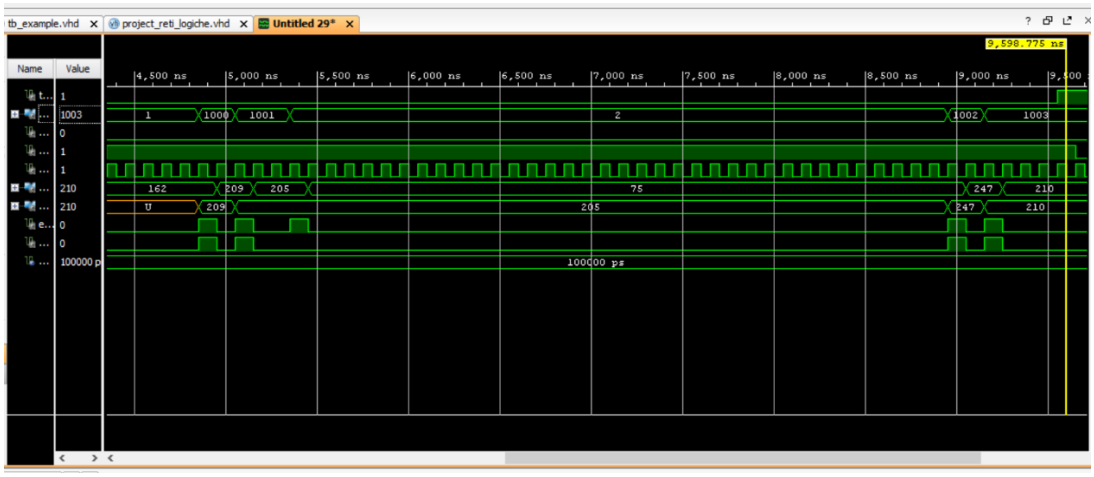
Il timing report invece mostra come il componente venga eseguito in circa 3,769 ns con quindi un periodo di clock inferiore ai 100 ns, come richiesto dalla specifica.

```
Timing Report

Slack (MET) :             95.618ns  (required time - arrival time)
  Source:                 number_of_words_reg[2]/C
                            (rising edge-triggered cell FDRE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Destination:            o_address_reg[0]/R
                            (rising edge-triggered cell FDRE clocked by clock  {rise@0.000ns fall@50.000ns 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:        3.769ns  (logic 0.999ns (26.506%)  route 2.770ns (73.494%))
  Logic Levels:           3  (LUT3=1 LUT6=2)
  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

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
```

Risultati Simulazioni :

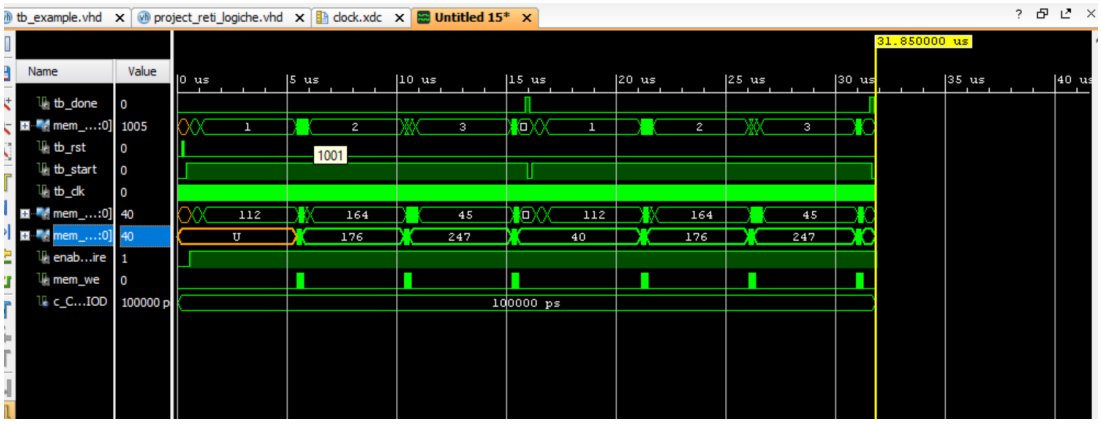
Il componente è stato testato in tutte le situazioni possibili. Purtroppo i test post sintesi non è stato possibile verificarne il superamento a causa di un warning sul constraint del clock e di warning su alcuni segnali (gli interi e i segnali indirizzi di memoria) che la sintesi definiva inutilizzabili e quindi rimuoveva dal modulo.
Cercando sul forum di Xilinx alcuni utenti hanno lamentato lo stesso problema con la versione di Vivado 2016.4 dicendo fosse un bug della sintesi.
Sono stati comunque eseguiti test diversi (incluso quello fornitoci) al fine di "stressare" al massimo l'architettura progettata e coprirne tutti i casi limite.

Test 1
Questo primo test base verifica semplicemente la corretta funzionalità del modulo.
Vengono date in input 2 parole (162 e 75) che verranno, come si vede in foto, correttamente convolute e fatte uscire negli indirizzi di memoria expected.



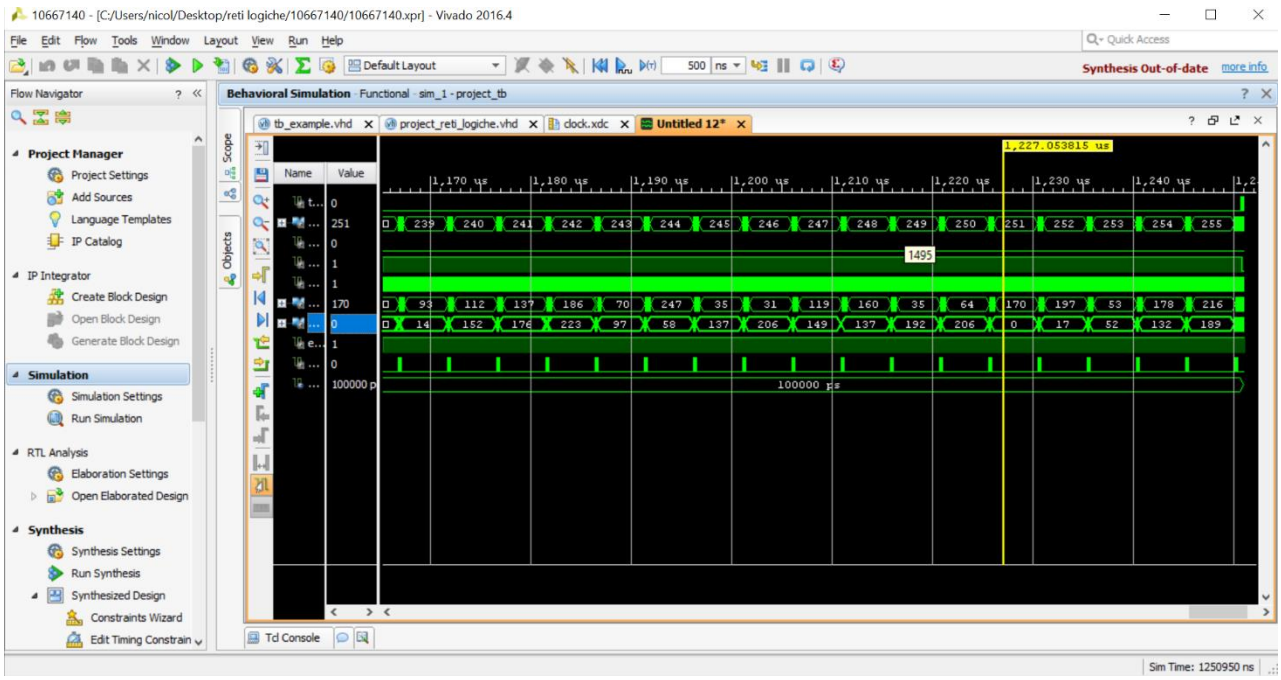
Test 2

Questo Test verifica un double processing sulla stessa ram.
Verifica quindi la robustezza del modulo facendogli fare inizialmente una convoluzione normale di 3 parole, alla fine di esse il segnale done (primo segnale in foto) passa a 1 e start a 0. In seguito viene rieseguita l'operazione precedente sugli stessi indirizzi di memoria di input e di output.



Test 3

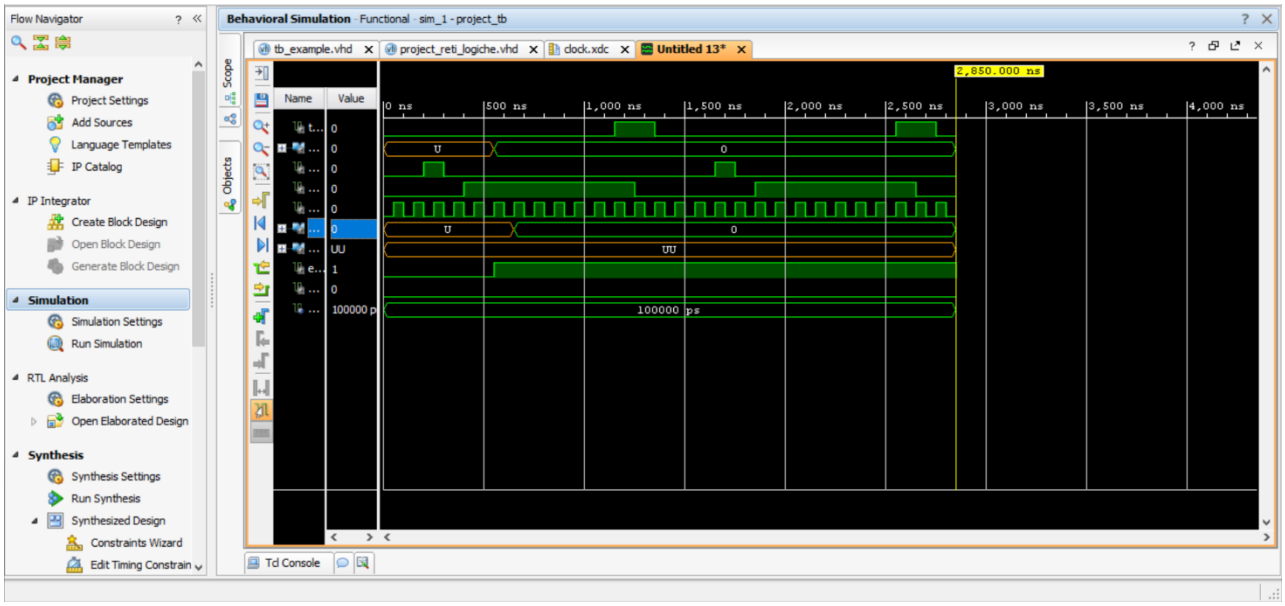
Questo test è stato fatto come test limite, infatti vengono date in input 255 parole ovvero la sequenza di lunghezza massima, cioè ram(0) = ‘11111111’ che corrisponde appunto a 255. Come mostrato in figura dal secondo segnale mem_address (secondo segnale), l’inidirizzo 255 viene effettivamente usato e il modulo risponde in maniera corretta al test.



Test 4

Questo test verifica un altro caso limite.

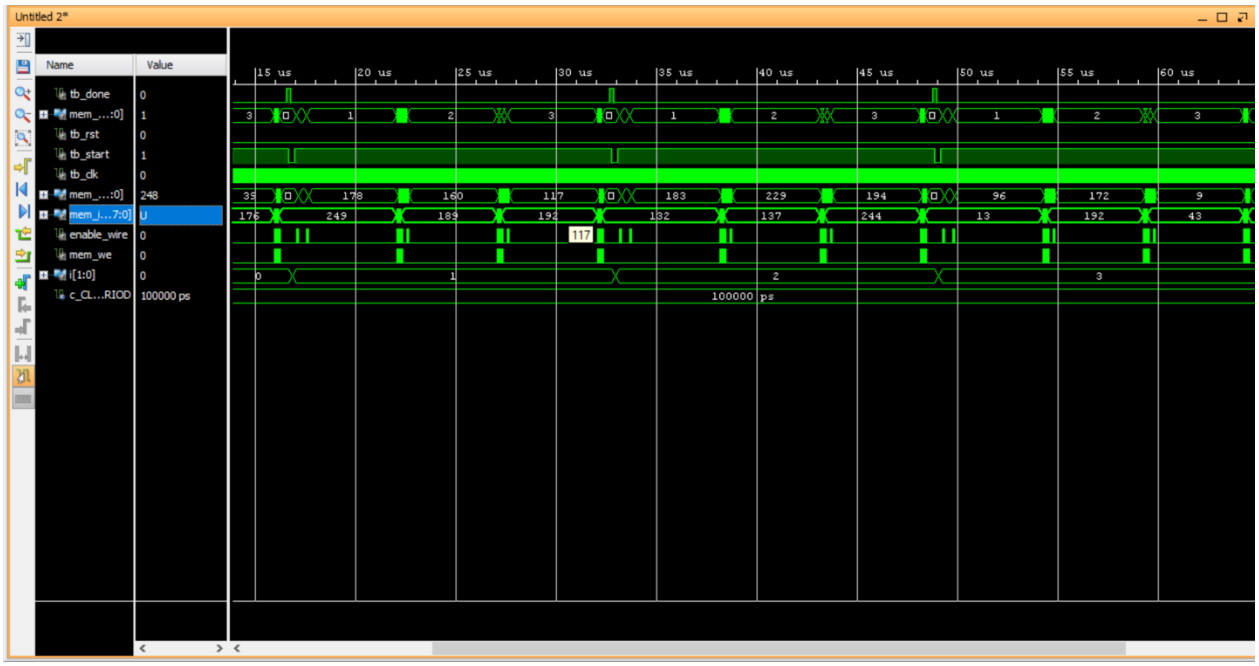
La prima parola che il modulo riceve in ingresso indica il numero di parole che dovranno essere commutate. In questo caso riceve 0 come prima parola e in seguito riceve altre 3 parole da commutare. Il modulo si comporta risponde correttamente al test non commutando le parole e rimanendo al mem_address 0.



Test 5

Questo test “stressa” al massimo il modulo andando a fare 3 run con reset per ognuna e convoluzione a ogni run di 3 parole diverse.

Anche in questo caso il modulo risponde in maniera corretta.



Conclusioni

Si ritiene che l'architettura progettata rispetti anzitutto le specifiche, fatto che è stato verificato mediante estensivo testing sia casuale, che con test benches manualmente generati.
Le tempistiche rispettano l'unico vincolo fornito dalla specifica riguardante il tempo minimo di clock pari a 100 ns.
Si ritiene quindi di aver descritto un componente hardware conforme alle specifiche in grado di rispondere correttamente ad ogni tipo di richiesta fornitagli.