Prova finale Progetto Reti Logiche

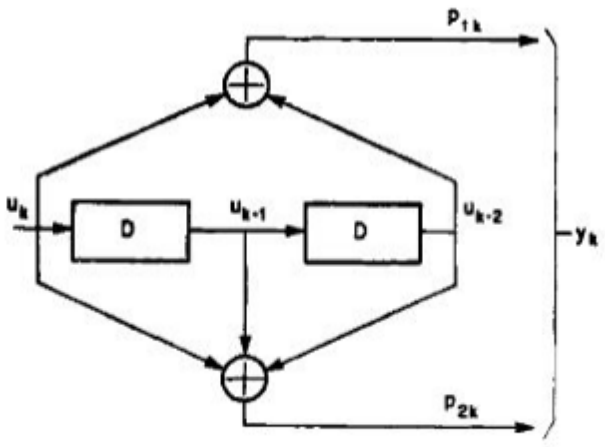
Anno accademico 2021/2022

Tommaso Aiello
Codice persona: 10687571

Indice

Introduzione

La specifica del progetto consiste nella sintesi di un modulo HW scritto in VHDL che si interfacci con una memoria RAM e che implementi un codificatore convoluzionale a tasso di trasmissione ½.



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

Tale codificatore può essere visto sia come una macchina a stati sia come tre segnali salvati in tre registri e due porte logiche XOR.

Il modulo da sintetizzare ha come compito quello di interfacciarsi con la memoria RAM, leggere il numero di Byte da dover processare all'indirizzo zero della memoria RAM, trasformare la sequenza di Byte in un flusso U di bit, il quale dopo essere stato processato dal codificatore diventerà il flusso Z di bit di lunghezza doppia rispetto al flusso U.
Il flusso Z dovrà poi essere salvato in memoria a partire dall'indirizzo 1000.

Quindi ogni parola da 8 bit che compone il flusso U verrà codificata in due parole da 8 bit ciascuna secondo la seguente logica: il codificatore convoluzionale all'istante di tempo k (Tk)
prende in ingresso un bit Uk, il risultato dello XOR tra Uk e UkMeno2 produce il bit Pk1, mentre il risultato dello XOR a tre ingressi tra Uk, UkMeno1 e UkMeno2 produce il bit Pk2.
Successivamente i due bit sono concatenati per poi formare il flusso Z.
Al ciclo di clock successivo il registro UkMeno1 prende il valore di Uk e il registro UkMeno2 prende il valore di UkMeno1 e un nuovo bit può essere processato.
Quando viene attivato un segnale di reset il valore dei due registri UkMeno1 e UkMeno2 è riportato a 0.

Esempio: con byte "10100011" e codificatore allo stato iniziale con uk1 e uk2 a zero.

	T0	T1	T2	T3	T4	T5	T6	T7
Uk	1	0	1	0	0	0	1	1
Pk1	1	0	0	0	1	0	1	1
Pk2	1	1	0	1	1	0	1	0

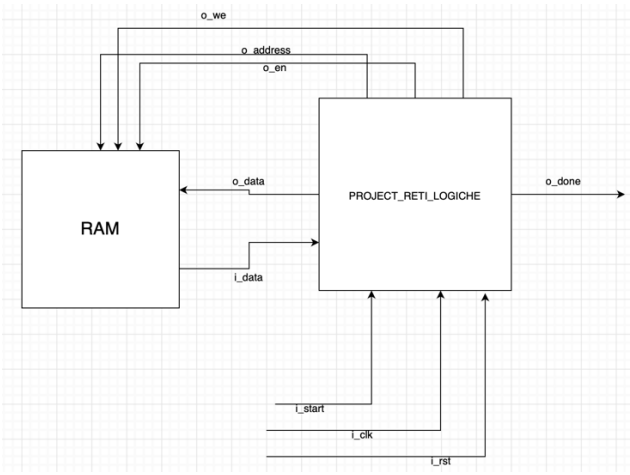
Di conseguenza il flusso K sarà 1101000111001110 il quale sarà poi diviso in due byte per essere salvato in memoria nei byte 11010001 e 11001110.

In breve il codice VHDL che è stato prodotto si comporta nel modo seguente:

- Dopo un segnale di reset e il seguente segnale di start inizia l'elaborazione.
- Assegna ai registri il loro valore di default.
- Legge il numero di byte da processare dalla memoria all'indirizzo di memoria.
- Salva in un registro la parola da processare, la serializza e fornisce l'input al codificatore un bit alla volta.
- Il codificatore calcola i due bit di uscita e li concatena.
- Scrive in memoria le due parole prodotte.
- Se le parole da processare sono terminate entra in uno stato dove pone il segnale di DONE alto e aspetta che il segnale di START si abbassi prima di riportare il segnale DONE basso così da prepararsi ad una nuova elaborazione.

Architettura

Data la semplicità del processo, è stato scelto di optare per una soluzione monomodulare, con un singolo process che gestisce sia l'evoluzione della macchina a stati sia le operazioni per giungere alla definizione del flusso di bit Z.



Segnali

Descrizione dei segnali che devono essere intesi come registri:

- **current_state** : il tipo è **STATE**. Usato per l'avanzamento della macchina a stati. Usato per il case statement che si trova nel processo della FSM.
  Valore default: S1.
- **dimension**: **std_logic_vector** da 8 bit. Ha il compito di salvare il numero di parole W che devono essere elaborate. Viene usato per salvare quindi il contenuto che si trova all'indirizzo

0 della RAM. Usato anche in confronto con il segnale count_words per indicare alla FSM quando l'elaborazione può terminare perché le parole da lavorare sono finite.
Valore default: "00000000".

- **count_words**: **unsigned** da 16 bit. Serve per tenere il conto delle parole lavorate. Sebbene non ci possano essere più di 255 parole da lavorare (8bit), ho deciso comunque di usare 16 bit perché tale segnale è usato anche per calcolare o_address che ha 16 bit.
  Valore default: "0000000000000000".

- **var_words**: **unsigned** da 16 bit. Segnale di supporto, usato sia da count_words sia da write_address.
  Valore default: "0000000000000000".

- **write_address**: **unsigned** da 16 bit. Il segnale ha come valore di default 1000 cioè l'indirizzo di base per scrivere in memoria. Il suo compito è quello di tenere traccia di quale sia il prossimo indirizzo disponibile per la scrittura in memoria.
  Valore default: "0000001111101000".

- **u**: **std_logic_vector** da 8 bit. Usato per salvare il byte in memoria da elaborare. Sfruttando il fatto che è un array, è stata usata la possibilità di riferirsi ai singoli bit dell'array per serializzare il byte da processare.
  Valore default: "00000000".

- **uk**: **std_logic**. E' il bit di ingresso al codificatore convoluzionale. Usato dalle porte XOR per calcolare sia il bit pk1 sia il bit pk2.
  Valore default: '0'.

- **uk1**: **std_logic**. E' il bit al tempo (t-1) del codificatore convoluzionale. E' uno dei tre ingressi dello XOR che genera il bit pk2.
  Valore di default 0.

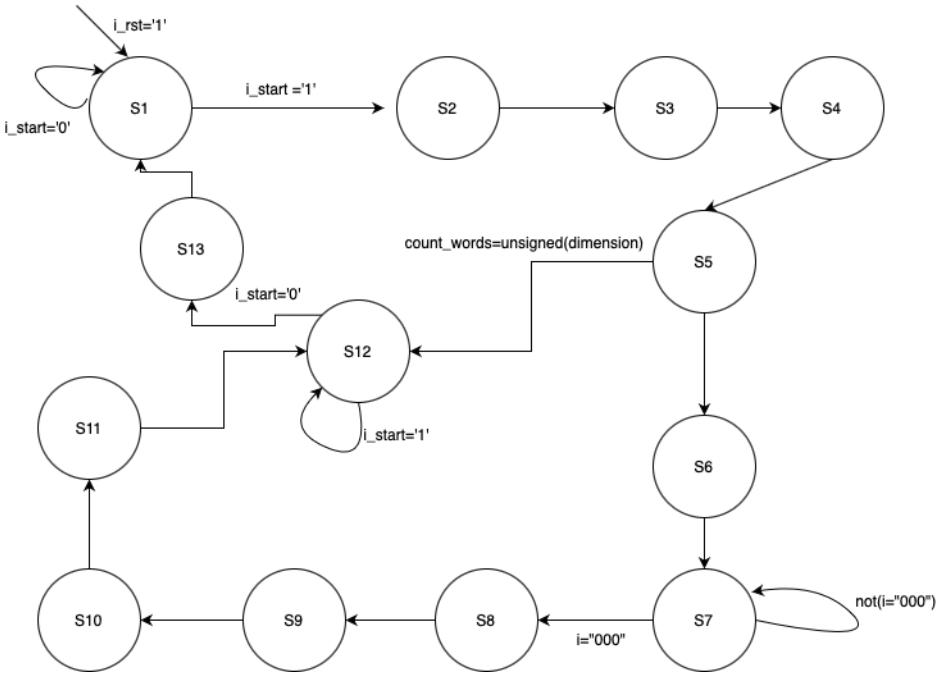
- **uk2**: **std_logic**. E' il bit al tempo (t-2) del codificatore convoluzionale. E' uno dei bit di ingresso delle porte XOR che generano i bit pk2 e pk1.
  Valore di default 0.

- **z**: **std_logic_vector** da 16 bit. E' il vettore che tiene conto delle elaborazioni del codificatore. Al suo interno sono salvati i bit pk1 e pk2 concatenati per ogni istante di tempo. I primi 8 bit (15 downto 8) saranno scritti in memoria all'indirizzo write_address mentre i bit (7 downto 0) saranno scritti in memoria all'indirizzo write_address +1.
  Valore di default: "0000000000000000".

- **i, vari**: **unsigned** 3 bit. Segnali che vengono usati per calcolare l'indice del bit che si vuole scegliere dal vettore u. Usati due segnali per poter codificare ogni bit in un ciclo di clock.
  Valore di default "111".

- **j, varj**: **unsigned** 4 bit. Segnali che vengono usati per calcolare l'indice del bit che si vuole scrivere nel vettore z. Usati due segnali per poter codificare ogni bit in un ciclo di clock.
  Valore di default "1111".

Macchina a Stati



Descrizione degli stati.

- **S1 :** lo stato di reset. Quando viene dato il segnale di reset='1', allora il current_state diventa S1. Fino a che non viene dato il segnale di Start ALTO, la macchina a stati rimane in S1, una volta che il segnale i_start = '1' allora current_state <= S2. In questo stato tutti i registri sono inizializzati al loro valore di default.

- **S2 :** stato in cui viene alzato il segnale di enable della memoria, o_address è già all'indirizzo zero, quindi alla fine di questo stato la macchina è pronta a leggere il contenuto della memoria.

- **S3 :** stato usato per aspettare che il segnale i_data sia disponibile. Alto il segnale di enable per leggere il dato.

- **S4 :** il contenuto della memoria è assegnato al segnale dimension. In più è incrementato il valore di o_address ed è alzato o_en così da essere pronti a leggere i_data in seguito, cioè il valore del primo byte da processare.

- **S5:** questo stato è usato per confrontare dimension con count_words così da stabilire se l'elaborazione può giungere al termine oppure deve proseguire. Alla fine di questo stato sarà disponibile i_data che verrà assegnato poi al registro u successivmente.

- **S6:** Stato in cui si assegna i_data al registro u. Alla fine di questo stato, il registro u contiene il dato presente in RAM(o_address). [o_address in questo caso inteso come intero].
  U contiene quindi il valore del byte da elaborare.

- **S7:** stato in cui si assegna al segnale uk il valore di u(7), così che nel ciclo di clock successivo si possa iniziare il ciclo che porta alla definizione del flusso Z.

- **S8:** questo stato si comporta in maniera differente in base al valore di i.

Se not(i=”000), allora ci sono ancora bit da dover lavorare quindi si assegna a uk il valore del bit successivo uk <= u(i-1), si assegna a uk1 il valore corrente di uk, e a uk2 il valore di uk1. Sostanzialmente si fanno scorrere i bit. Poi sono presenti le due porte XOR implementate con un costrutto if/elsif che porta alla definizione dei due bit pk1 e pk2 che vengono salvati nel vettore z all’indice corretto grazie all’utilizzo di un contatore j. Poi sono aggiornati i valori di i e j che dovranno essere usati al ciclo successivo.
Se invece i=”000” allora vuol dire che quello che viene lavorato è l’ultimo bit della parola u, quindi non serve assegnare il nuovo uk, mentre è necessario assegnare i nuovi valori di uk1 e uk2 e di pk1 e pk2. Occorre anche resettare i valori di i,j,vari,varj per averli nuovamente a disposizione in caso di necessità di una nuova elaborazione, prima di passare allo stato 9.

- **S9:** Viene alzato sia il segnale di enable, sia quello di write enable così di poter scrivere in memoria al ciclo di clock successivo. Viene inoltre aggiornato il segnale o_address grazie al segnale write_address, così da averlo pronto per essere usato al ciclo di clock successivo. Assegnato anche o_data con i bit di z che vanno da 15 a 8. Viene usato anche il segnale var_words a cui è assegnato il numero di parole processate.

- **S10:** Viene tenuto alto sia o_en che o_we per poter scrivere al ciclo di clock successivo anche il byte del vettore z(7 downto 0). Viene aumentato il valore di o_address e viene usato var_words per aumentare il conto delle parole elaborate. Poi si assegna a var_words il valore di write_address così da poter incrementare write_address.

- **S11:** Viene alzato nuovamente o_en, mentre è tenuto basso o_we, perché si vuole iniziare una nuova elaborazione, quindi anche o_address è riportato grazie a count_words al valore dell’indirizzo della parola W successiva. Write_address è incrementato di 2 così da poter essere pronto successivamente.

- **S12:** Questo stato è uno dei stati che precede la fine dell’elaborazione. o_done<=’1’ . Fino a che il segnale di i_start non è abbassato allora la macchina a stati rimane in questo stato. Se i_start =’0’ allora la macchina a stati va nello stato 13, stato finale.

- **S13:** stato finale in cui viene abbassato il segnale di o_done e si ritorna nella stato iniziale dove si può attendere un nuovo segnale di i_start così da iniziare una nuova elaborazione.

Analisi Post Sintesi

Il codice VHDL prodotto è sintetizzabile e si possono notare le seguenti caratteristiche ottenute tramite il comando 'report_utilization' :

- Sono usati 139 Slice LUTs e ne sono disponibili 134600, il che significa che è stato utilizzato il 0.10%

- Sono usati 137 Slice Registers e ne sono disponibili 269200, il che significa che è stato utilizzato lo 0.05%. **Di questi 137 Slice Registers 0 sono dei Latch.**

Tramite il comando report_timing:

- Slack (MET) : 95.984 ns. Il clock richiesto è di 100ns mentre il design sintetizzato produce un Data Path Delay di appena 3,865 ns. Tmin = Tclk – Slack + Tram = 6.016 ns .
  Quindi la frequenza massima a cui può andare il design costruito è:
  fmax= 1/Tmin = 166,2 MHz

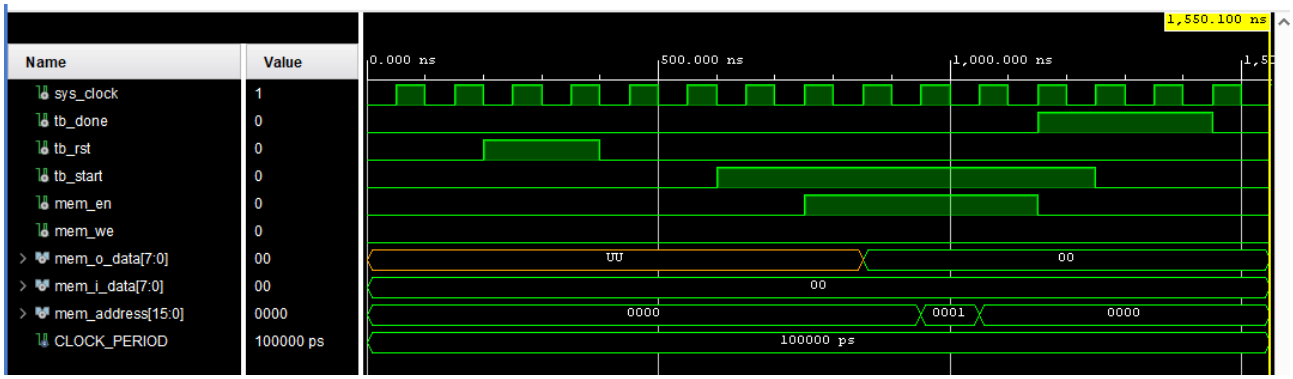
Risultati Simulazioni

Test 1:

Test con numero di parole 0. RAM inizializzata con tutti vettori che sono a zero.
RAM(0) <= "00000000";
Il Test è stato pensato per vedere se il modulo sintetizzato gestisce il corner case per cui non ci sono parole da elaborare.
Assert superati:

- assert RAM(1000) = "00000000"

Time: 1550100 ps.

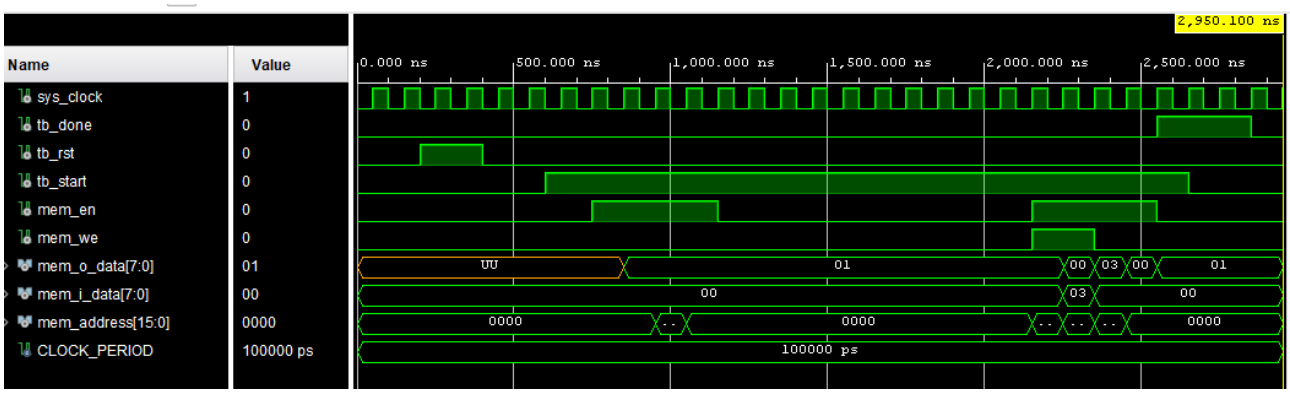


Test 2:

Test con dimensione 1. Input stream = [1] Output stream=[0,3].
Assert superati:

- assert RAM(1000) = std_logic_vector(to_unsigned(0,8))
- assert RAM(1001) = std_logic_vector(to_unsigned(3,8))
- assert RAM(1002) = std_logic_vector(to_unsigned(0,8))
- assert RAM(1003) = std_logic_vector(to_unsigned(0,8))
- assert RAM(1004) = std_logic_vector(to_unsigned(0,8))

Time: 2950100 ps;

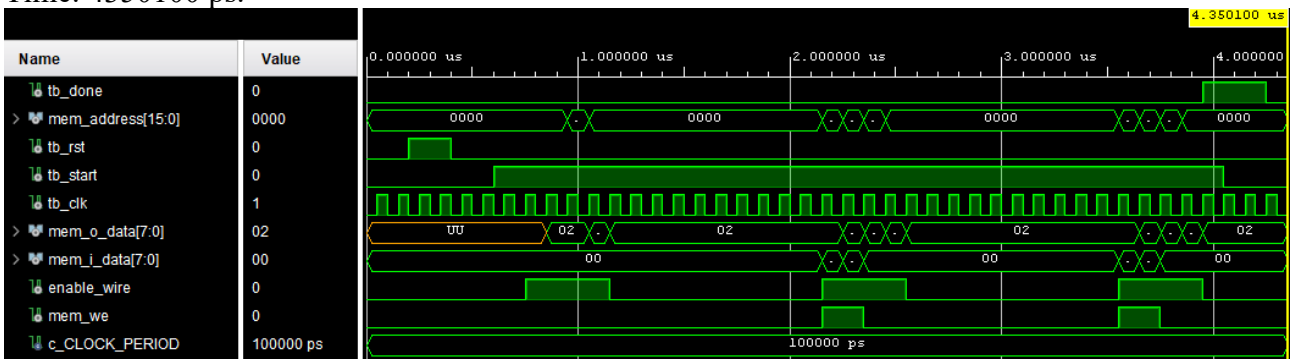


Test 3:

Testbench di esempio.
Dimensione 2, Input Stream = [162, 75] , Output Stream = [209, 205,247,210]
Assert superati:

- assert RAM(1000) = std_logic_vector(to_unsigned(209,8))
- assert RAM(1001) = std_logic_vector(to_unsigned(205,8))
- assert RAM(1002) = std_logic_vector(to_unsigned(247,8))
- assert RAM(1003) = std_logic_vector(to_unsigned(210,8))
- assert RAM(1004) = std_logic_vector(to_unsigned(0,8))

Time: 4350100 ps.

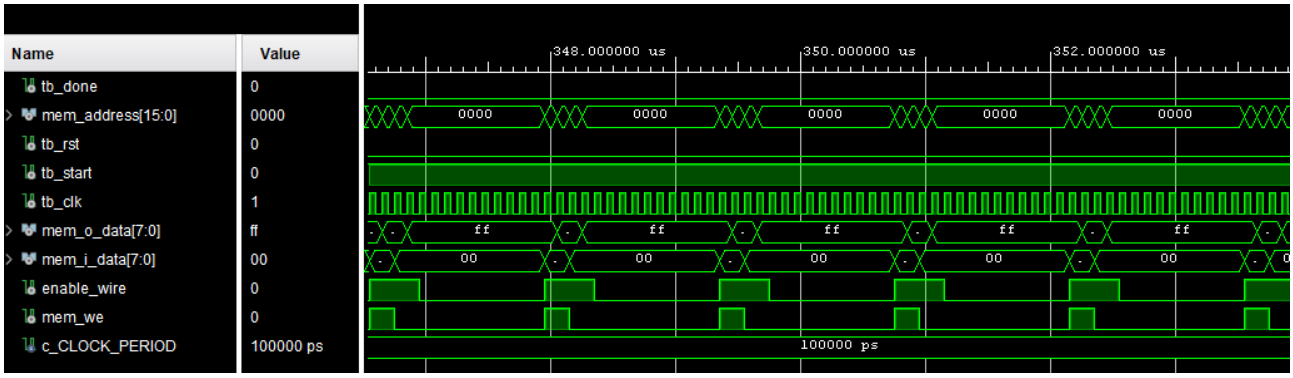


Test 4:

Dimensione massima 255. Input Stream tutti zero. Output Stream tutti zero.
Test pensato per vedere se il sistema riesce a gestire la dimensione massima di byte da elaborare e il tempo che viene impiegato.
Assert superati:

- assert RAM(1000) = std_logic_vector(to_unsigned(0,8))
- assert RAM(1001) = std_logic_vector(to_unsigned(0,8))
- assert RAM(1002) = std_logic_vector(to_unsigned(0,8))
- assert RAM(1003) = std_logic_vector(to_unsigned(0,8))
- assert RAM(1054) = std_logic_vector(to_unsigned(0,8))
- assert RAM(1254) = std_logic_vector(to_unsigned(0,8))
- assert RAM(1504) = std_logic_vector(to_unsigned(0,8))

Time : 358550100 ps.

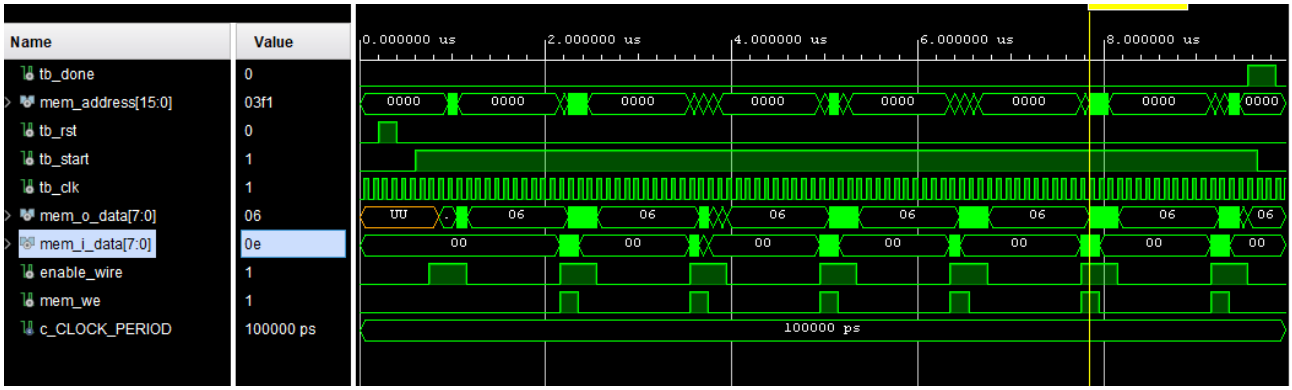


Test 5:

Dimensione 6.
Input Stream = [163,47,4,64,67,13]
Output Stream = [209,206,189,37,176,55,55,0,55,14,176,232]

Superati tutti gli assert per ogni cella da 1000 a 1011.

Time: 9950100 ps.



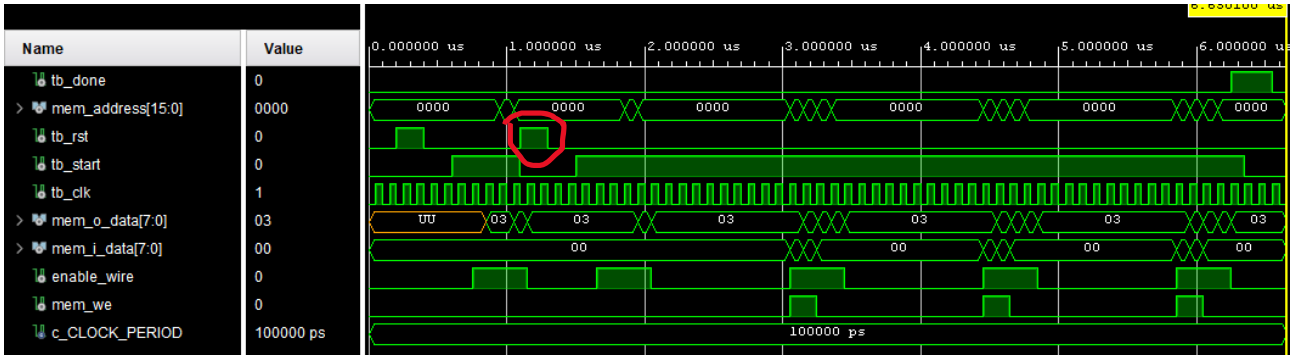
Test 6:

Dimensione 3.
Input Stream = [112,164,45]
Output Stream = [57, 176, 209, 247, 13, 40]
Particolarità di questo test è che durante l'esecuzione è stato dato il segnale di reset='1' il che ha portato il modulo a ripartire. Segnalato nell'immagine da un cerchio rosso.

Assert superati:

- assert RAM(1000) = std_logic_vector(to_unsigned(57,8))
- assert RAM(1001) = std_logic_vector(to_unsigned(176,8))
- assert RAM(1002) = std_logic_vector(to_unsigned(209,8))
- assert RAM(1003) = std_logic_vector(to_unsigned(247,8))
- assert RAM(1004) = std_logic_vector(to_unsigned(13,8))
- assert RAM(1005) = std_logic_vector(to_unsigned(40,8))

Time: 6650100 ps.

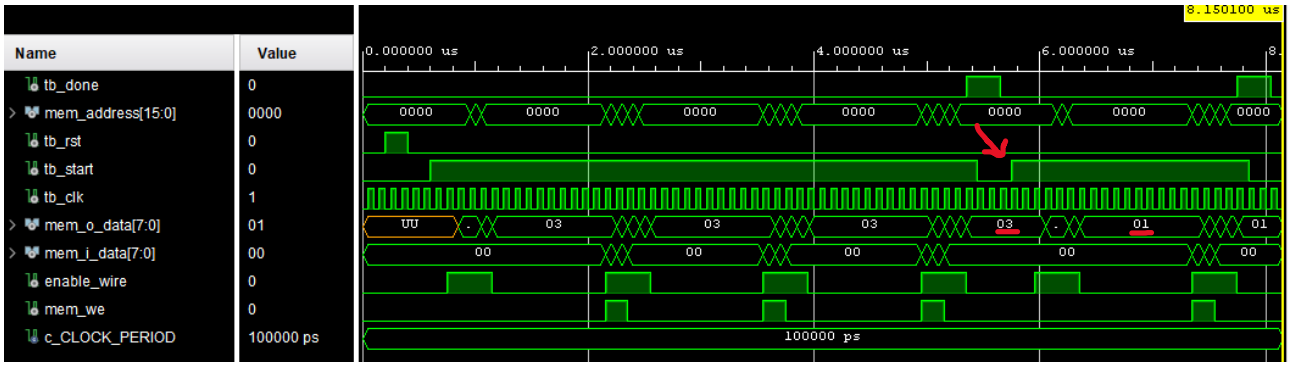


Test 7:

Test per vedere se elaborazione riparte correttamente dopo una elaborazione e il successivo stato di start.
E' modificato il processo che gestisce la RAM in modo tale che nel momento in cui il primo processo termina, quindi in corrispondenza di un segnale di o_done='1', venga modificato il contenuto della RAM e in seguito di un nuovo segnale di start il modulo può elaborare dati nuovi.

```
MEM : process(tb_clk)
begin
    if tb_clk'event and tb_clk = '1' then
        if enable_wire = '1' then
            if mem_we = '1' then
                RAM(conv_integer(mem_address))  <= mem_i_data;
                mem_o_data                      <= mem_i_data after 1 ns;
            else
                mem_o_data <= RAM(conv_integer(mem_address)) after 1 ns;
            end if;
        end if;
        if(tb_done ='1') then
            RAM(0) <= "00000001";
            RAM(1) <= "10100010";
            RAM(2) <= std_logic_vector(to_unsigned(  0  , 8));
            RAM(3) <= std_logic_vector(to_unsigned(  0  , 8));
        END IF;
    end if;
end process;
```

Prima elaborazione: dimensione 3, Input [112,164,45], Output [57,176,209,247,13,40]
Seconda elaborazione: dimensione 1, Input [“10100010”] Output [209, 205]
Time: 8150100 ps.
Evidenziato in rosso il momento in cui viene dato un nuovo segnale di start e il cambiamento del contenuto dell’indirizzo 0 della memoria RAM, dal valore 3 al valore 1.



Conclusioni

In conclusione il design progettato ha le seguenti caratteristiche:

- Funziona in pre e post sintesi.
- La frequenza massima del segnale di clock è di 166,2 MHz
- Utilizzo di LUT pari al 0.10%
- Utilizzo di Flip Flop pari al 0.05%
- Assenza di Latch