POLITECNICO
MILANO 1863

Prova Finale Di Reti Logiche

Anno Accademico 2023/2024

Chen Jie
Codice Persona 10750777

Indice

1 Introduzione **2**
1.1 Specifiche del progetto 2
1.2 Regole del funzionamento 3
1.3 Interfaccia del Componente 3
1.4 Descrizione di memoria 4

2 Architettura **6**
2.1 Scelte Progettuali 6
2.1.1 Parte I - Elaborazione stringa 6
2.1.2 Parte II - Contattore e controllar di Done 7
2.1.3 Parte III - Segnali principali 7
2.2 Macchina a stati finiti 8
2.3 Gestione dei processi 10
2.4 Design finale dell'architettura 11

3 Risultati sperimentali **12**
3.1 Test Bench Example 12
3.1.1 Behavioural 12
3.1.2 Post-Synthesis 12
3.2 Test i casi limiti 13
3.2.1 Caso 1: solo zeri 13
3.2.2 Caso 2: tutti i numeri diversi da zeri 14
3.2.3 Caso 3: un numero non zero seguito da più di 31 zeri consecutivi 14
3.2.4 Caso 4: con i_K massimo 15
3.2.5 Caso 5: con i_K min 15
3.2.6 Caso 6: i_rst sale durante l'esecuzione 16
3.2.7 Caso 7: i_start consecutivi 16
3.2.8 Caso 8: scrive consecutivamente la stessa sequenza negli stessi indirizzi 17

4 Conclusioni **18**

1 Introduzione

1.1 Specifiche del progetto

La specifica richiede di implementare un modulo HW che è in grado di completare una sequenza di K parole, la quale viene memorizzata a partire dall'indirizzo ADD. Le regole da rispettare per trasformare una sequenza sono seguenti:

1. Le parole W di una sequenza sono salvate per ogni 2 byte di indirizzo (e.g. ADD, ADD+2, ..., ADD+2(K-1)), invece per il byte mancante si riempie con un valore di credibilità C.

2. Nel caso in cui W non è la prima parola della sequenza ed è uguale a zero, allora verrà sostituito con l'ultima W letta diverso da zero. In tutti altri casi W rimane invariata.

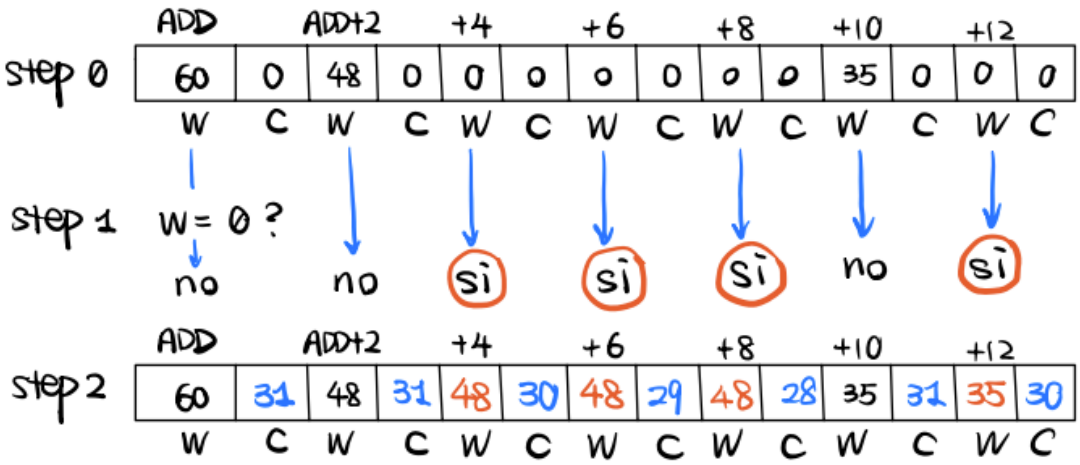
3. Se il valore di W è diverso da zero, allora il valore di credibilità C seguito a tale W deve essere uguale a 31, altrimenti:

   - Se il valore di C precedente è maggiore di zero, allora il valore C viene decrementato di 1 rispetto al valore C precedente, altrimenti il C uguale a zero.

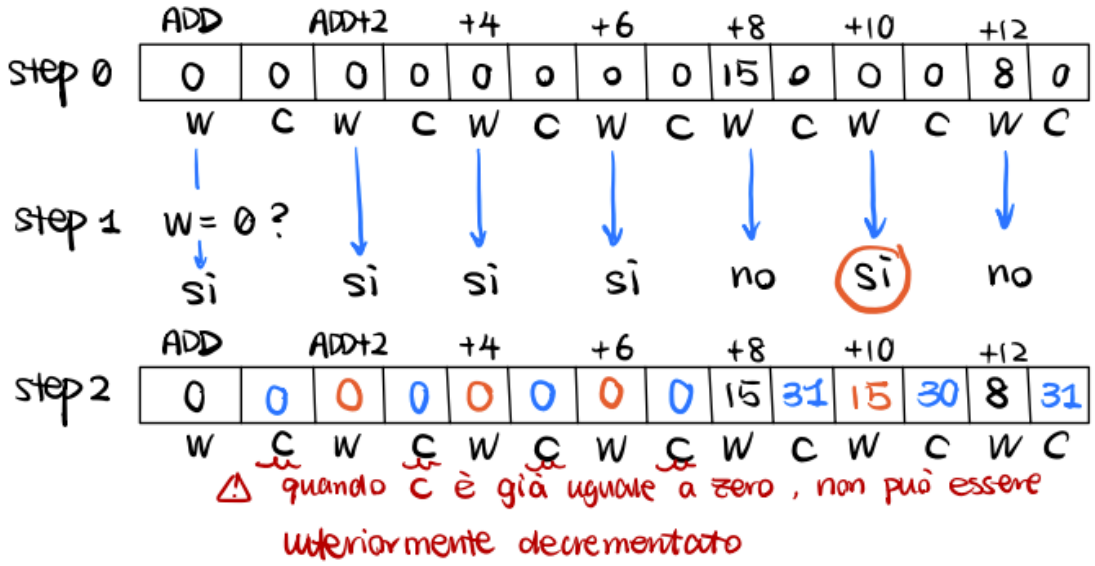
   Nota bene: nella fase di scrittura di credibilità, bisogna considerare il valore originale di W, cioè il valore di W prima della sostituzione.

Esempi:

iniziale:	60	0	48	0	0	0	0	0	0	0	35	0	0	0
finale:	60	31	48	31	48	30	48	29	48	28	35	31	35	30



iniziale:	0	0	0	0	0	0	0	0	15	0	0	0	8	0
finale:	0	0	0	0	0	0	0	0	15	31	15	30	8	31



1.2 Regole del funzionamento

Tutti i segnali, tranne il reset, sono sincroni e sono interpretati sul fronte di salita del clock:

- All'istante iniziale: RST=1, DONE=0;
- All'istante in cui reset torna a 0 e start alza a 1 indica l'inizio dell'elaborazione: START=1, DONE=0;
- All'istante in cui DONE alza a 1 indica il termine della computazione: START=0.

Considerazioni:

- Prima del primo START=1, deve esistere un RESET=1;
- Dopo il primo RESET=1, il segnare start una volta che è sceso a 0 può essere riportato a 1 senza aspettare che RESET rialza a 1;
- I valori ADD e K vengono posti sugli ingressi quando il START viene riportato ad alto e rimangono fissi per tutta la fase dell'elaborazione;
- Prima di alzare DONE, l'elaborazione della sequenza deve essere completata.

1.3 Interfaccia del Componente

Nei seguenti paragrafi, introduco i segnali collegati all'interfaccia(fig.1), classificando in segnali di input e segnali di output.
Segnali di input:

- i_clk è il segnale di CLOCK, generato da Test Bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere i_add e i_k all'ingresso quando start=1;

```
entity project_reti_logiche is
    port (
        i_clk   : in std_logic;
        i_rst   : in std_logic;
        i_start : in std_logic;
        i_add   : in std_logic_vector(15 downto 0);
        i_k     : in std_logic_vector(9 downto 0);

        o_done  : out std_logic;

        o_mem_addr : out std_logic_vector(15 downto 0);
        i_mem_data : in std_logic_vector(7 downto 0);
        o_mem_data : out std_logic_vector(7 downto 0);
        o_mem_we   : out std_logic;
        o_mem_en   : out std_logic
    );
end project_reti_logiche;
```

Figura 1: Interfaccia del componente

- i_start è il segnale di CLOCK, generato da Test Bench;
- i_k è il segnale(vettore) di CLOCK, generato da Test Bench, indica la lunghezza della sequenza.
- i_add è il segnale(vettore) di CLOCK, generato da Test Bench, indica l'indirizzo da cui partire la sequenza.
- i_mem_data è il segnale(vettore) che arriva dalla memoria e contiene il dato presente in memoria nell'indirizzo specificato.

Segnali di output:

- o_done è il segnale che comunica la fine dell'elaborazione;
- o_mem_addr è il segnale(vettore) che contiene l'indirizzo da effettuare l'operazione in memoria;
- o_mem_data è il segnale(vettore) che contiene il dato che verrà scritto successivamente in memoria;
- o_mem_en è il segnale di ENABLE: o_mem_en=1 sia in lettura che in scrittura;
- o_mem_we è il segnale di WRITE ENABLE: o_mem_we=1 in scrittura, o_mem_we=0 in lettura.

1.4 Descrizione di memoria

La memoria è già istanziata dal Test Bench, in questo paragrafo riassumo le caratteristiche principali del blocco RAM(fig.2).
L'entità RAM ha 5 segnali di input, tra quelli, clk, we, en servono per gestire l'operazione in memoria, mentre il segnale addr da 16 bit indica su quale indirizzo

vuole effettuare la lettura o la scrittura, per l'ultimo abbiamo il segnale da 8 bit che contiene i dati da scrivere in memoria. Invece il segnale di output do restituisce il valore letto da un certo indirizzo.
Dall'architettura, si osserva che la memoria fornita è sincronica, cioè tutte operazioni si svolgono sul fronte di salita del clock, inoltre sia la scrittura che la lettura di memoria sono fatte dopo 2 ns.

```
-- Single-Port Block RAM Write-First Mode (recommended template)
--
-- File: rams_02.vhd
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rams_sp_wf is
port(
  clk  : in  std_logic;
  we   : in  std_logic;
  en   : in  std_logic;
  addr : in  std_logic_vector(15 downto 0);
  di   : in  std_logic_vector(7 downto 0);
  do   : out std_logic_vector(7 downto 0)
);
end rams_sp_wf;

architecture syn of rams_sp_wf is
type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
signal RAM : ram_type;
begin
  process(clk)
    begin
    if clk'event and clk = '1' then
      if en = '1' then
        if we = '1' then
          RAM(conv_integer(addr)) <= di;
          do                      <= di after 2 ns;
        else
          do <= RAM(conv_integer(addr)) after 2 ns;
        end if;
      end if;
    end if;
  end process;
end syn;
```

Figura 2: Descrizione di RAM

2 Architettura

2.1 Scelte Progettuali

Dopo l'analisi precedente, ho deciso di classificare i componenti(fig.3 e fig.4) basandosi sulle diverse funzionalità che svolgono:

1. Una parte che occupa la conversione da una stringa in ingresso a una stringa elaborata in uscita.

2. Una parte che è costituita da un contattore decrementale e un controllore del segnale Done.

3. Una macchina a stati che gestisce i processi tramite i segnali trasmessi.

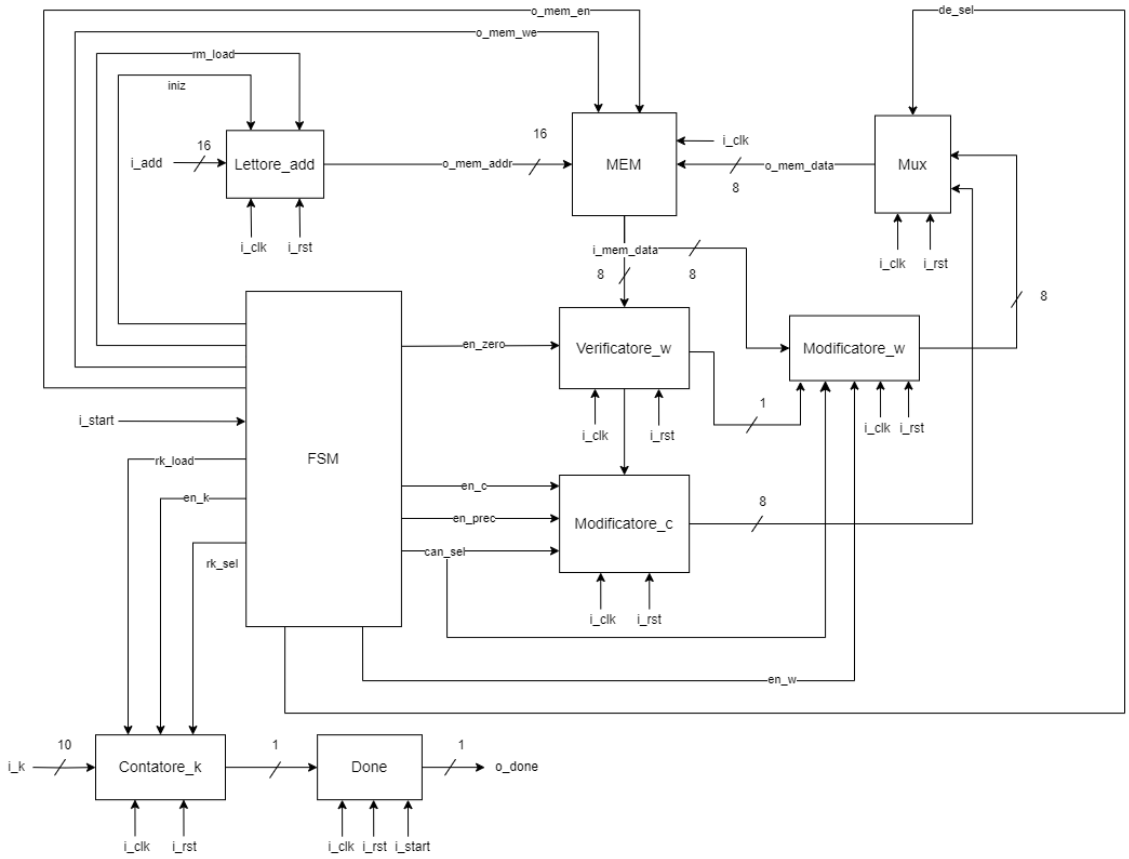


Figura 3: Design di architettura iniziale

2.1.1 Parte I - Elaborazione stringa

- Lettore_add: riceve in input i_add e come l'output restituisce il valore di o_mem_addr.

- Verificatore_W: verifica se l'input è uguale a zero o no e restituisce 0 se è falso, altrimenti restituisce 1.

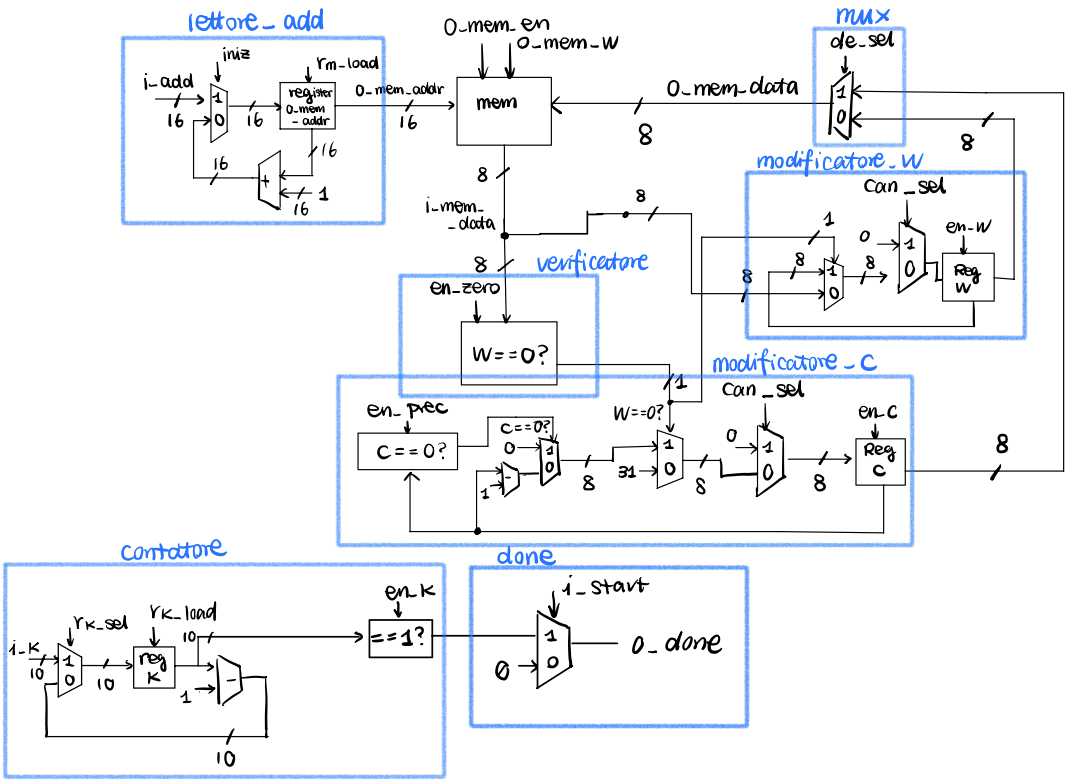


Figura 4: Design dettagliato iniziale

- Modificatore_W: gestisce la sostituzione della parola, soprattutto nel caso in cui W è uguale a zero, in input ha i_mem_data e in output ritorna il valore da scrivere in memoria.
- Modificatore_C: funziona analogamente come il Modificatore_W, ma nel suo interno contiene un verificatore di zeri di per sé in modo da evitare l'ulteriore decremento quando C precedente è uguale a zero.
- Mux: seleziona quale tra due segnali in input deve essere messo in output come il valore di o_mem_data.

2.1.2 Parte II - Contattore e controllar di Done

- Contatore_K: ha l'input come i_k, decrementa di 1 ogni qualvolta che finisca la scrittura di credibilità C.
- Done: collegato con il Contatore_K, serve per emettere o_done in modo corretto.

2.1.3 Parte III - Segnali principali

In questa sezione, presento solo i segnali gestiti dalla macchina a stati e approfondirò nella sezione successiva il funzionamento della macchina a stati.

- iniz: se iniz=1, allora legge i_add e indica che stiamo valutando la prima parola della stringa.

- rm_load: se rm_load=1, allora aggiorna il registro di indirizzo.
- en_zero: se en_zero=1 allora attiva il verificatore di zero per W.
- en_c: se en_c=1 allora attiva il registro per salvare C.
- en_prec: se en_prec=1 allora attiva il verificatore di C per verificare se il C salvato precedentemente è uguale a zero oppure no
- can_sel: se can_sel=1 allora azzera il registro di W e il registro di C.
- rk_load: se rk_load=1 allora salva il valore in ingresso nel registro K.
- rk_sel: se rk_sel=1 allora sceglie i_k altrimenti sceglie il valore decrementato.
- en_k: se en_k=1, attiva il verificatore per K, controlla se è uguale a 1 oppure no.
- en_w: se en_w=1 allora attiva il registro per salvare W.
- de_sel: se de_sel=1, allora scrive il valore salvato in registro C in memoria, altrimenti scrive il valore salvato in registro W in memoria.

2.2 Macchina a stati finiti

Per la parte di controllo, ho deciso di utilizzare una macchina di Moore(fig.5) in cui le uscite sono determinate in funzione dei soli stati correnti. Per aver una migliore visualizzazione, non sono state riportate in figura tutte le frecce per reset e i output generati sono presentati in un figura(fig.6) separata. Tutti i segnali sono inizializzati con gli zeri, quindi se non viene specificato il valore, per default le uscite hanno un valore uguale al zero.

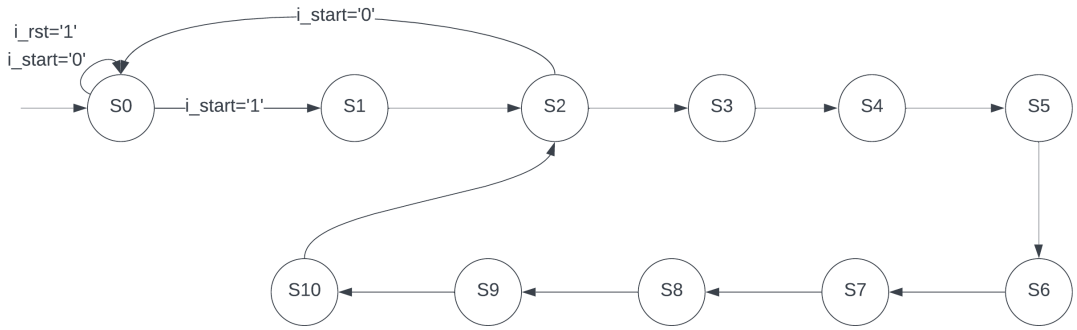


Figura 5: Diagramma di stati

- Stato 0:
  1. Stato di reset e di partenza, ogni volta che i_rst=1 o i_start=0 si ritorna allo stato 0.
- Stato 1:
  1. Attiva il lettore_add e svolge la lettura in memoria.
  2. Attiva il contatore_k, legge i_k in ingresso e salvarlo in registro.
  3. Inizializza il registro di W e il registro di C.

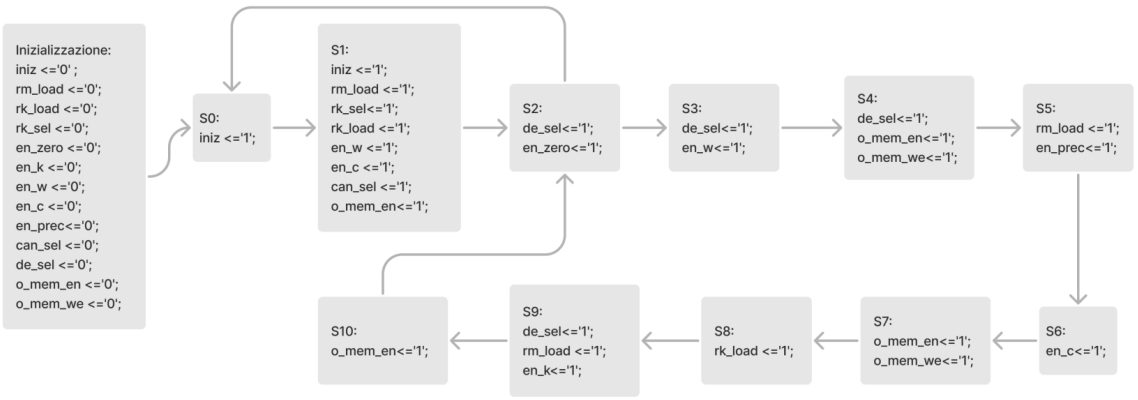


Figura 6: Output di stati

- Stato 2:
  1. Riceve il valore letto dalla memoria e attiva il verificatore di zero.
  2. Seleziona il blocco di modificatore_W. 3. Se i_stato=0, allora si ritorna allo stato S0;

- Stato 3:
  1. Salva il valore da scrivere nel registro di W.

- Stato 4:
  1. Scrive il valore in memoria.

- Stato 5:
  1. Inizia la fase di credibilità, incrementa l'indirizzo o_mem_data.
  2. Verifica se il valore salvato precedentemente in registro C è uguale a 0 oppure no.

- Stato 6:
  1. Salva il valore da scrivere in registro C.

- Stato 7:
  1. Scrive in memoria la credibilità C.

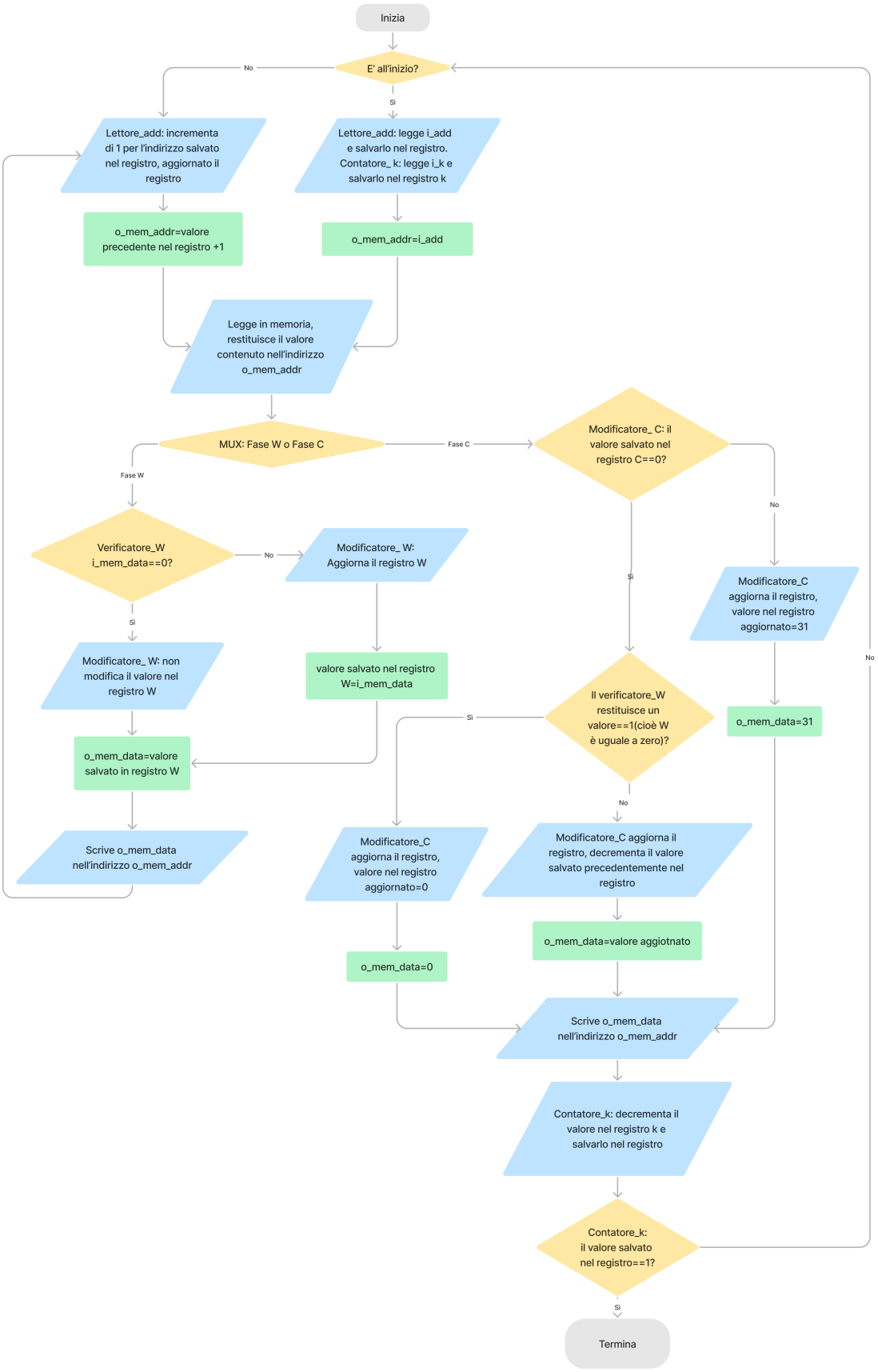
- Stato 8:
  1. Decrementa il valore i_k con il contatore di K

- Stato 9:
  1. Verifica se il valore aggiornato è uguale a 1 oppure no:
  - Se true, allora significa l'elaborazione di stringa è finito, avvisa il componente Done a portare o_done a 1.
  - Se false, allora prepara la lettura del prossimo indirizzo.

- Stato 10:
  - Se la verifica nello stato S9 ritorna true, allora allo stato S10 si realizza o_done=1 e i_start viene abbassato a 0, però il cambiamento del segnale i_start viene accorto solo un clock dopo allo stato successivo.
  - Se la verifica nello stato S9 ritorna false, allora allo stato S10 si inizia la lettura in memoria.

2.3 Gestione dei processi

Con il seguente diagramma di flusso sintetizzo come si genera l'output dall'input.



2.4 Design finale dell'architettura

Grazie al Vivado, tramite Open Elaborated Design è possibile ottenere la seguente figura dopo la fase di implementazione del codice. Confrontando il design iniziale (fig.3) e il design finale(fig.7) , possiamo notare la coerenza e la compatibilità tra loro.

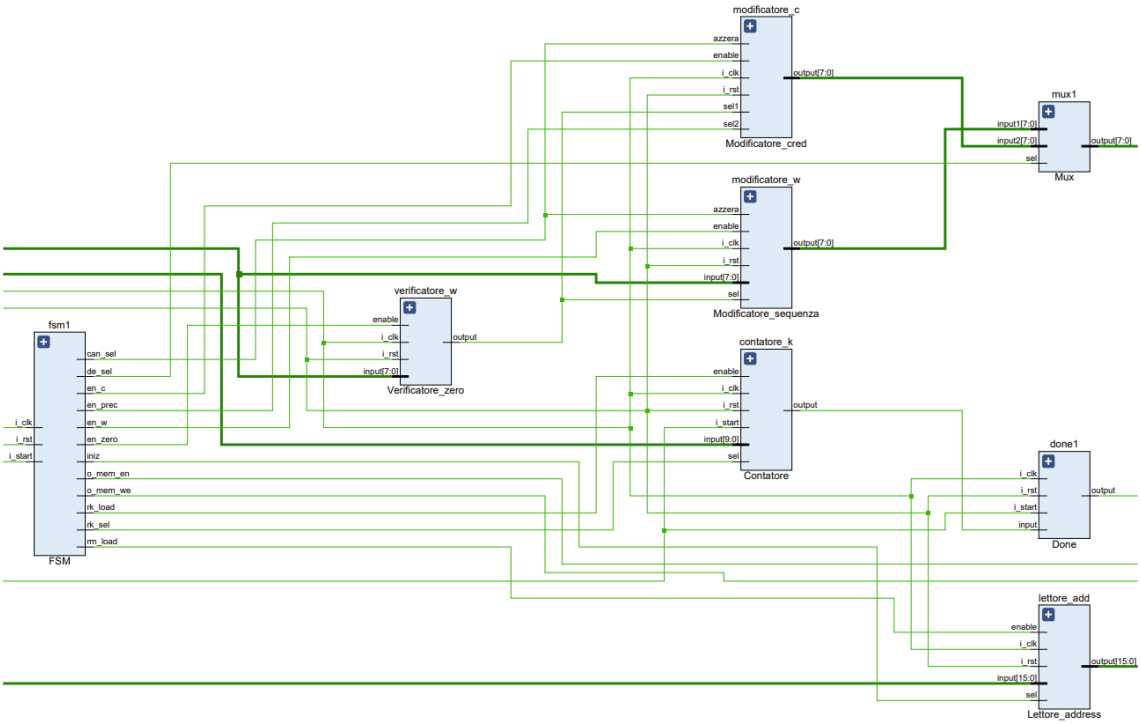


Figura 7: Architettura finale

3 Risultati sperimentali

In questa sezione, riporto i risultati della fase di testing: Test Bench Example e Test casi limiti.

3.1 Test Bench Example

Innazitutto, parto con l'analisi del comportamento del progetto con il Test Bench Example(scaricato da WeBeep), suddividendo in Behavioral Simulation e Post-Synthesis Simulation.

3.1.1 Behavioural

Osservando dal grafico(fig. 8) del risultato di test, è possibile vedere come si è comportato il mio progetto rispetto al Test Bench Example:

- Fase iniziale: o_done=0 dopo il reset e prima di i_start=1.
- Fase dell'elaborazione: o_done rimane a 0, i_start rimane a 1.
- Fase fine l'erabolazione: o_done sale a 1 e i_start scende a 0
- o_mem_en rimane a 0 dal clk in cui il progetto realizza o_done=1.
- Per l'input dato dal test, il progetto restituisce l'output desiderato in 3250 ns .

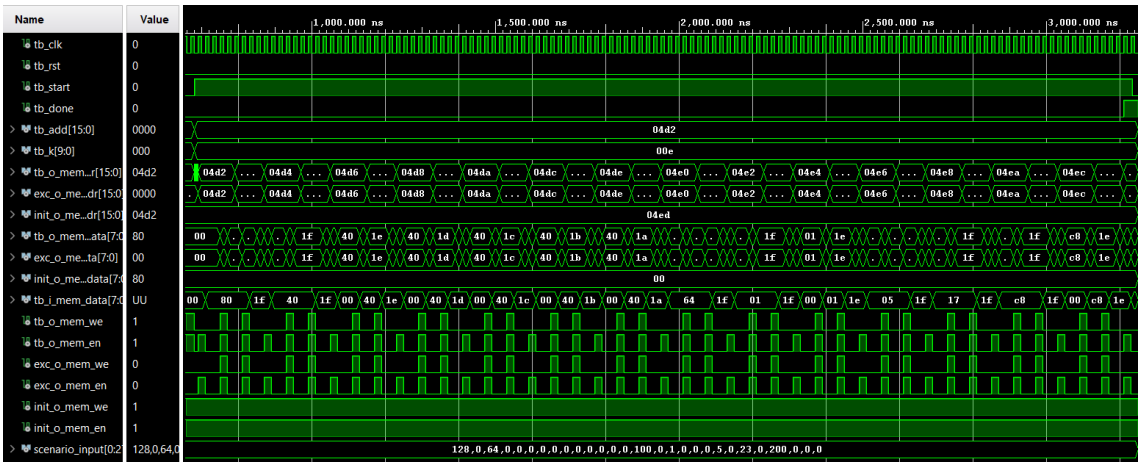


Figura 8: L'andamento dei segnali pre-sintesi

3.1.2 Post-Synthesis

Dai dati generati dal Report Utilization(fig.9), si nota che nel progetto sono stati utilizzati 54 Flip Flop e non è presente nessun Latch.
Invece per quando riguarda al Report Timing(fig.10), il progetto ha il slack da 17.557 ns che rappresenta la differenza tra il tempo necessario per produrre un output e il tempo di clock, cioè il progetto impiega solo 2.443 ns per commutare un segnale che è un valore abbastanza basso rispetto al 20 ns.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	67	0	0	134600	0.05
LUT as Logic	67	0	0	134600	0.05
LUT as Memory	0	0	0	46200	0.00
Slice Registers	54	0	0	269200	0.02
Register as Flip Flop	54	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Figura 9: Report Utilization

```
Timing Report

Slack (MET) :             17.557ns  (required time - arrival time)
  Source:                 fsm1/FSM_onehot_curr_state_reg[1]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@10.000ns period=20.000ns})
  Destination:            contatore_k/stored_value_reg[0]/CE
                            (rising edge-triggered cell FDRE clocked by clock  {rise@0.000ns fall@10.000ns period=20.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            20.000ns  (clock rise@20.000ns - clock rise@0.000ns)
  Data Path Delay:        2.021ns  (logic 0.489ns (24.196%)  route 1.532ns (75.804%))
  Logic Levels:           2  (LUT4=1 LUT6=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    1.860ns = ( 21.860 - 20.000 )
    Source Clock Delay      (SCD):    2.117ns
    Clock Pessimism Removal (CPR):    0.112ns
```

Figura 10: Report Timing

3.2 Test i casi limiti

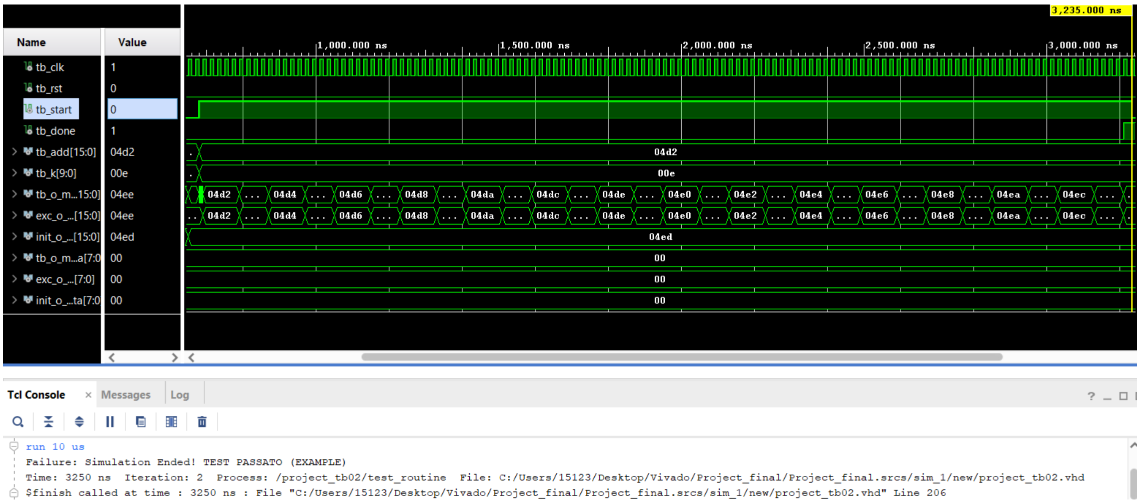
Una volta superato il Test Bench fornito, proseguo testing con gli esempi di input e output forniti dal file di specifica. Dopodiché nell'ultima fase di test si concentra di studiare i casi limiti del progetto.

3.2.1 Caso 1: solo zeri

La sequenza in ingresso è costituito solo dagli zeri, per testare se il mio progetto è in grado di gestire i valori di credibilità e i valori W della sequenza originale, dove i valori di C non devono essere sotto zero e in questo caso i valori finale di W devono essere uguali ai valori iniziali, cioè genera un output formato da tutti e soli zeri. Dallo screenshot seguente, si nota che il progetto passa il test in 3250 ns.

```
constant SCENARIO_LENGTH : integer := 14;
type scenario_type is array (0 to SCENARIO_LENGTH*2-1) of integer;

signal scenario_input : scenario_type := (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 );
signal scenario_full  : scenario_type := (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 );
```

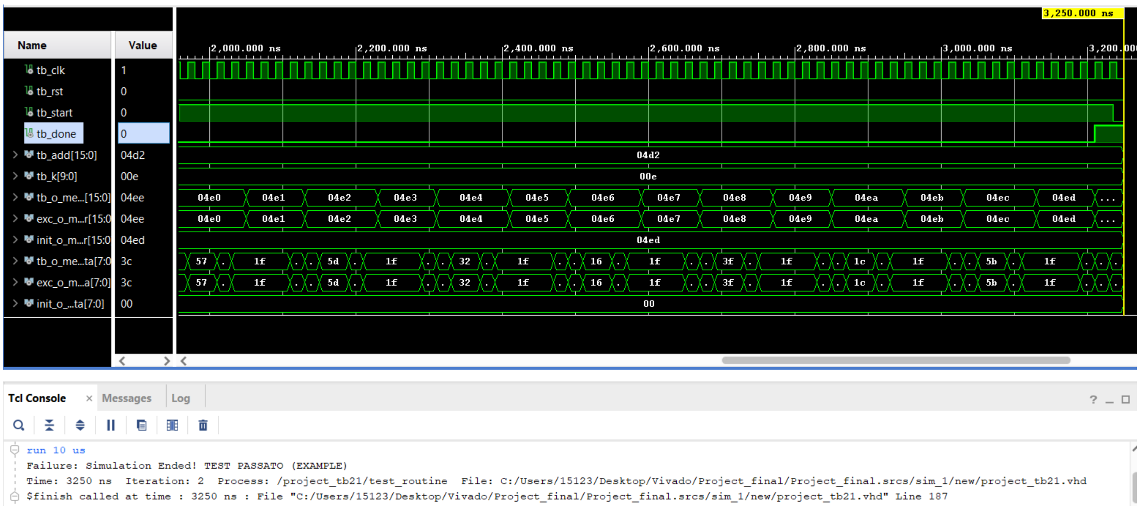


3.2.2 Caso 2: tutti i numeri diversi da zeri

Per testare come si comporta il mio progetto quando non bisogna incrementare mai la credibilità e fare la sostituzione della parola in sequenza, dovrebbe generare l'output con i valori di credibilità uguale a 31 e i valori di W invariati. Dallo screenshot seguente, si nota che il progetto passa il test in 3250 ns.

```
constant SCENARIO_LENGTH : integer := 14;
type scenario_type is array (0 to SCENARIO_LENGTH*2-1) of integer;

signal scenario_input : scenario_type := (84, 0, 87, 0, 78, 0, 16, 0, 94, 0, 36, 0, 87, 0, 93, 0, 50, 0, 22, 0, 63, 0, 28, 0, 91, 0, 60, 0 );
signal scenario_full  : scenario_type := (84, 31, 87, 31, 78, 31, 16, 31, 94, 31, 36, 31, 87, 31, 93, 31, 50, 31, 22, 31, 63, 31, 28, 31, 91, 31, 60, 31 );
```

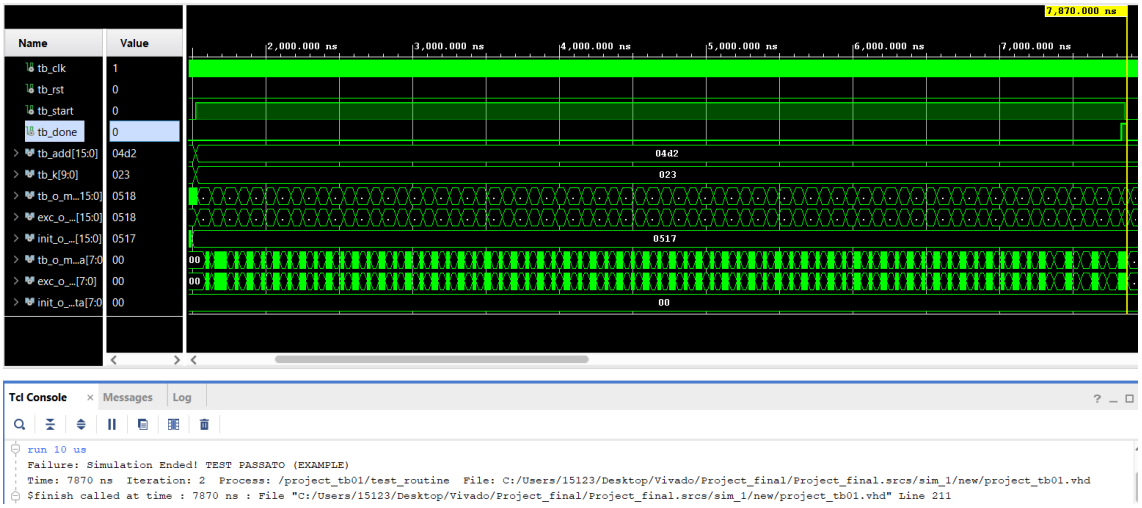


3.2.3 Caso 3: un numero non zero seguito da più di 31 zeri consecutivi

Per testare se il progetto riesce comportarsi in modo correttamente, cioè non si decrementa ulteriormente la credibilità quando il precedente credibilità è già uguale a zero. Dallo screenshot seguente, si nota che il progetto passa il test in 7870 ns.

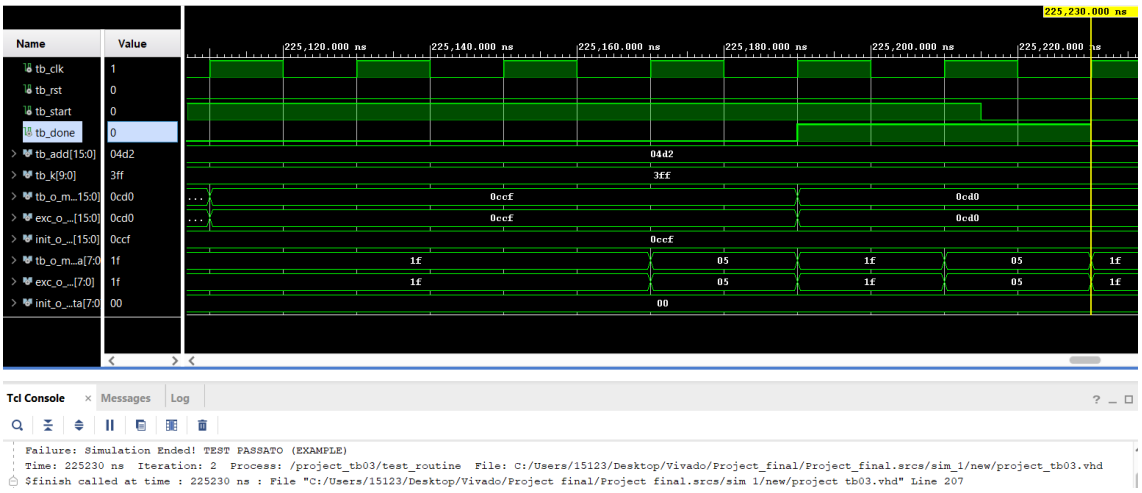
```
constant SCENARIO_LENGTH : integer := 35;
type scenario_type is array (0 to SCENARIO_LENGTH*2-1) of integer;

signal scenario_input : scenario_type := (128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 );
signal scenario_full  : scenario_type := (128, 31, 128, 30, 128, 29, 128, 28, 128, 27, 128, 26, 128, 25, 128, 24, 128, 23, 128, 22, 128, 21, 128, 20, 128,
19, 128, 18, 128, 17, 128, 16, 128, 15, 128, 14, 128, 13, 128, 12, 128, 11, 128, 10, 128, 9, 128, 8, 128, 7, 128, 6, 128, 5, 128, 4, 128, 3, 128, 2, 128,
1, 128, 0, 128, 0, 128, 0, 128, 0 );
```



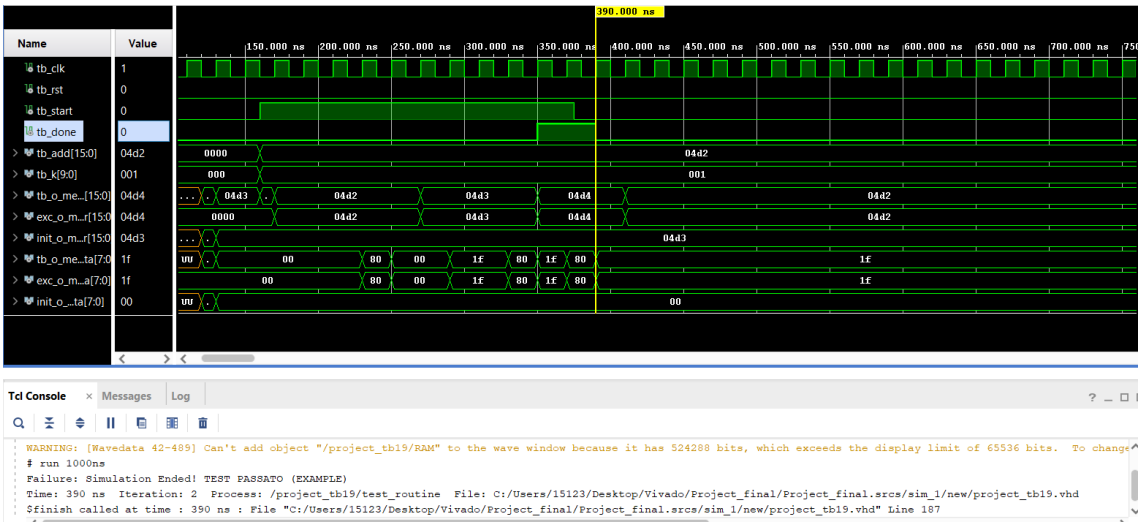
3.2.4 Caso 4: con i_K massimo

Per testare quando K aggiunge il valore massimo, come si comporta il progetto. Dallo screenshot seguente, si nota che il progetto passa il test in 225230 ns.



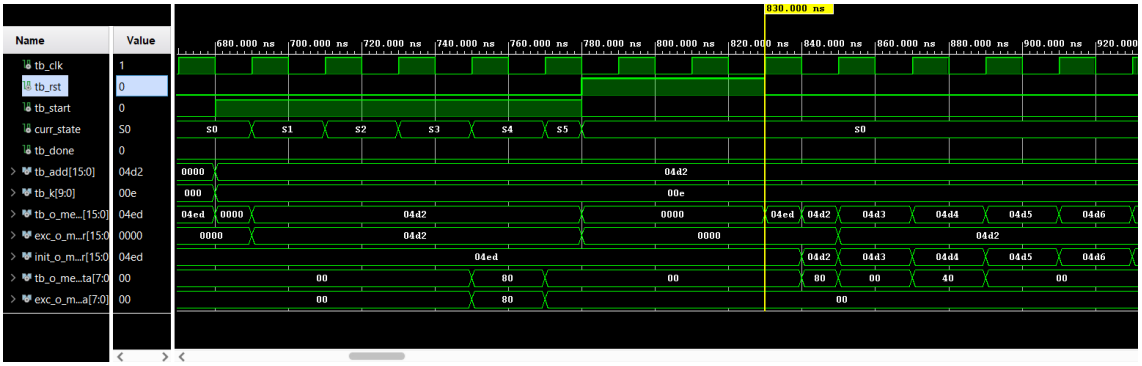
3.2.5 Caso 5: con i_K min

Per testare se il progetto funziona anche nel caso estremo ovvero elementare quando k è uguale a 1. Dallo screenshot seguente, si nota che il progetto passa il test in 390 ns.



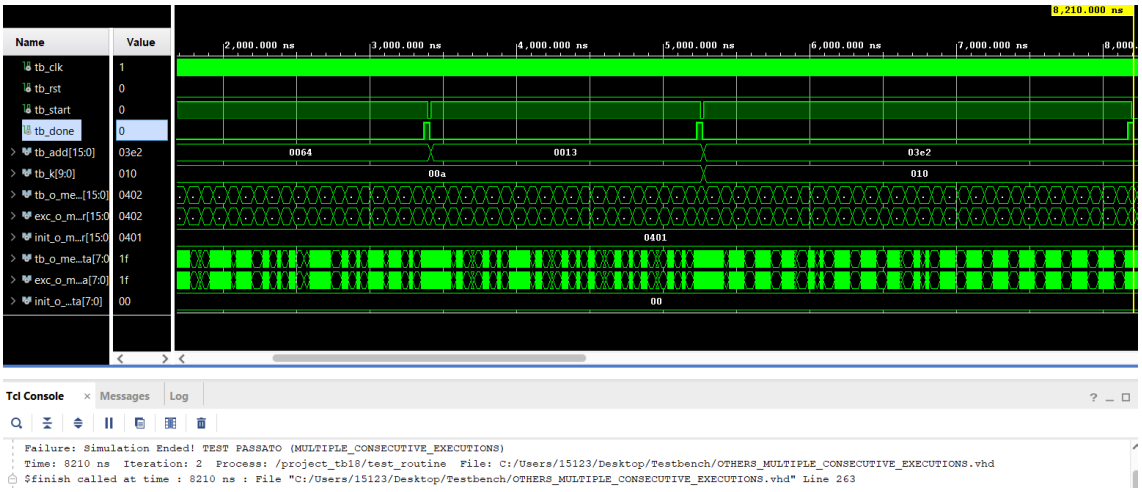
3.2.6 Caso 6: i_rst sale durante l'esecuzione

Per testare il comportamento del progetto quando sale il segnale di reset durante la fase dell'elaborazione. Analizzando i segnali portati dallo screenshot, è possibile osservare che in questo caso l'elaborazione è stata interrotta immediatamente quando il reset sale a 1 e la macchina a stati si ritorna allo stato iniziale S0, inoltre il progetto passa il test in 830 ns.



3.2.7 Caso 7: i_start consecutivi

Per testare come il mio progetto reagisce nel caso in cui dopo un start segue subito un'altro start di fronte a più sequenze consecutive. Dal screenshot è possibile osservare che il progetto funziona correttamente anche in questo caso specifico, i_done viene commutato più voltre e il test viene passato in 8210 ns.



3.2.8 Caso 8: scrive consecutivamente la stessa sequenza negli stessi indirizzi

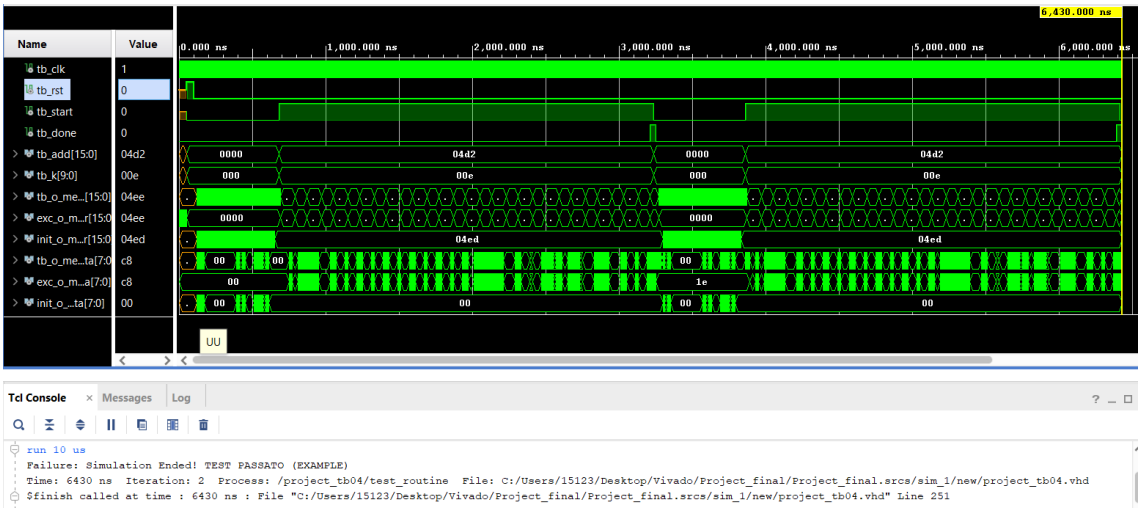
Per testare che cosa succederebbe quando la stessa sequenza viene scritta più volte negli stessi indirizzi. Si osserva che il progetto passa il test in 6430 ns e si comporta come se fosse nel caso in generale, cioè elabora la sequenza come se fosse una nuova sequenza diversa dalla sequenza passata precedentemente, rilegge la sequenza in ingresso e rielabora la sequenza.

```
constant SCENARIO_LENGTH : integer := 14;
constant SCENARIO_LENGTH_2 : integer := 14;

type scenario_type is array (0 to SCENARIO_LENGTH*2-1) of integer;
type scenario_type_2 is array (0 to SCENARIO_LENGTH_2*2-1) of integer;
signal scenario_input : scenario_type := (128, 0,  64, 0,   0,  0,  0,  0,  0,  0,  0,  0,  0, 100,  0, 1,  0, 0,  0, 5,  0, 23,  0, 200,  0,   0,  0 );
signal scenario_full  : scenario_type := (128, 31, 64, 31, 64, 30, 64, 29, 64, 28, 64, 27, 64, 26, 100, 31, 1, 31, 1, 30, 5, 31, 23, 31, 200, 31, 200, 30 );
signal scenario_input_2 : scenario_type_2 := (128, 0,  64, 0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 100,  0, 1,  0, 0,  0, 5,  0, 23,  0, 200,  0,   0,  0 )
signal scenario_full_2  : scenario_type_2 := (128, 31, 64, 31, 64, 30, 64, 29, 64, 28, 64, 27, 64, 26, 100, 31, 1, 31, 1, 30, 5, 31, 23, 31, 200, 31, 200, 30 )

signal memory_control : std_logic := '0';

constant SCENARIO_ADDRESS : integer := 1234;
constant SCENARIO_ADDRESS_2 : integer := 1234;
```



4 Conclusioni

Riassumendo le fasi di sviluppo: all'inizio mi trovavo un po' in difficoltà, soprattutto non sapevo da quale punto potevo partire perché è un progetto completamente diverso dal progetto di API che è un altro progetto per la laurea triennale. Per il primo passo, ho deciso di iniziare il design dell'architettura partendo da una bozza sulla carta, dopo una serie di prove ho utilizzato il piattaforma draw.io per disegnare il circuito. Prima di passare all'implementazione del codice ho stabilito anche una bozza per la macchina a stati per tenere sotto controllo dei prossi. Procedendo con le fasi successive e basandosi sulle richieste del progetto, ho dovuto apportare delle modifiche al design iniziale. E Una volta superato il Test Bench fornito, sono rivoltata allo studio dei casi limiti del progetto. Alla fine, sono riuscita a completare il progetto che è in grado di trasformare l'input in l'output seguendo le richieste. Complessivamente, si tratta di un'epserienza più che positiva, grazie alla quale ho imparato a sviluppare individualmente un progetto, ad accorgere gli errori durante la progettazione e a risolverli in modo razionale. Questa opportunità mi ha inoltre fornito una base solida per affrontare progetti futuri con maggiore competenza e sicurezza.