POLITECNICO
MILANO 1863

Prova Finale Di Reti Logiche

Anno Accademico 2023/2024

Chen Jie
Codice Persona 10750777

Indice

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.

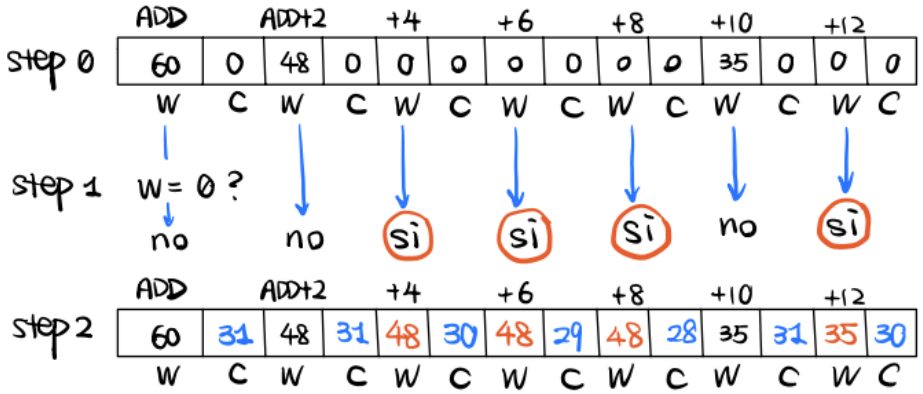
3. Se il valore di W è diverso da zero, allora il valore di credibilità C seguito a tale W deve essere uguale a 31, altrimenti il valore C viene decrementato rispetto al valore C precedente se C è maggiore o uguale di zero, nel caso in cui C precedente è uguale a zero, non verrà ulteriormente decrementato.

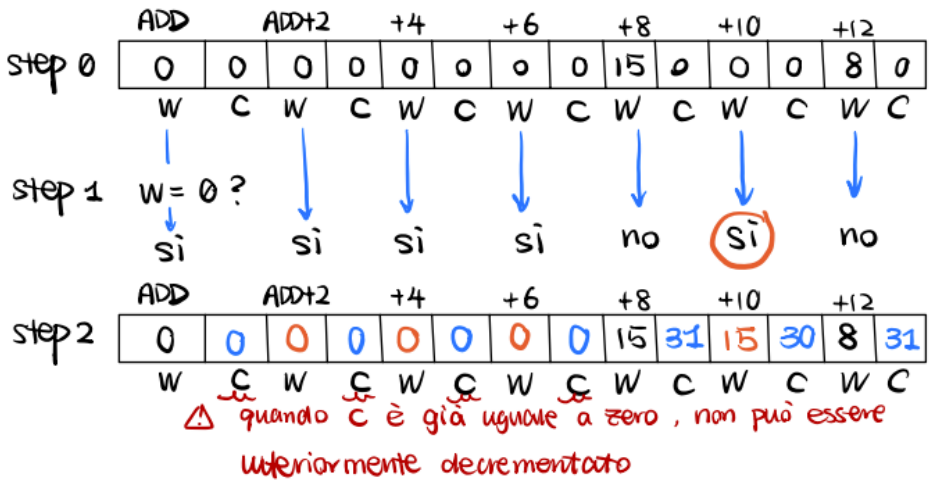
   Attenzione: nella fase di scrittura di credibilità, si considera il valore originale di W cioè il valore prima della sostituzione.

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.
Possiamo suddividere in casi seguenti:

- 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, c'è sempre il RESET=1;
- Dopodiché una volta che START riabbassa a zero, può essere riportato a 1 senza aspettare che RESET rialzarsi 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, introduciamo i segnali collegati alla nostra interfaccia, 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;
- i_start è il segnale di CLOCK, generato da Test Bench;

```
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_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;

Nota bene:

1. Il nome del modulo deve essere project_reti_logiche per poter eseguire correttamente il Test Bench Example e deve essere presente una sola architettura per ogni entità.

1.4 Descrizione di memoria

La memoria è già istanziata dal Test Bench, in questo paragrafo cerchiamo di riassumere le caratteristiche principali del blocco RAM, vedi figura 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 di di 8 bit che rappresenta i dati da scrivere in memoria. Invece il segnale di output do restituisce il valore letto da un certo indirizzo. Dall'architettura, si nota che la memoria fornita

```
-- 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

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.

2 Architettura

2.1 Scelte Progettuali

Dopo l'analisi precedente ho deciso di separare le funzionalità in più parti:

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

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

3. Una macchina a stati controlla il corretto funzionamento di ogni componente tramite i segnali trasmessi.

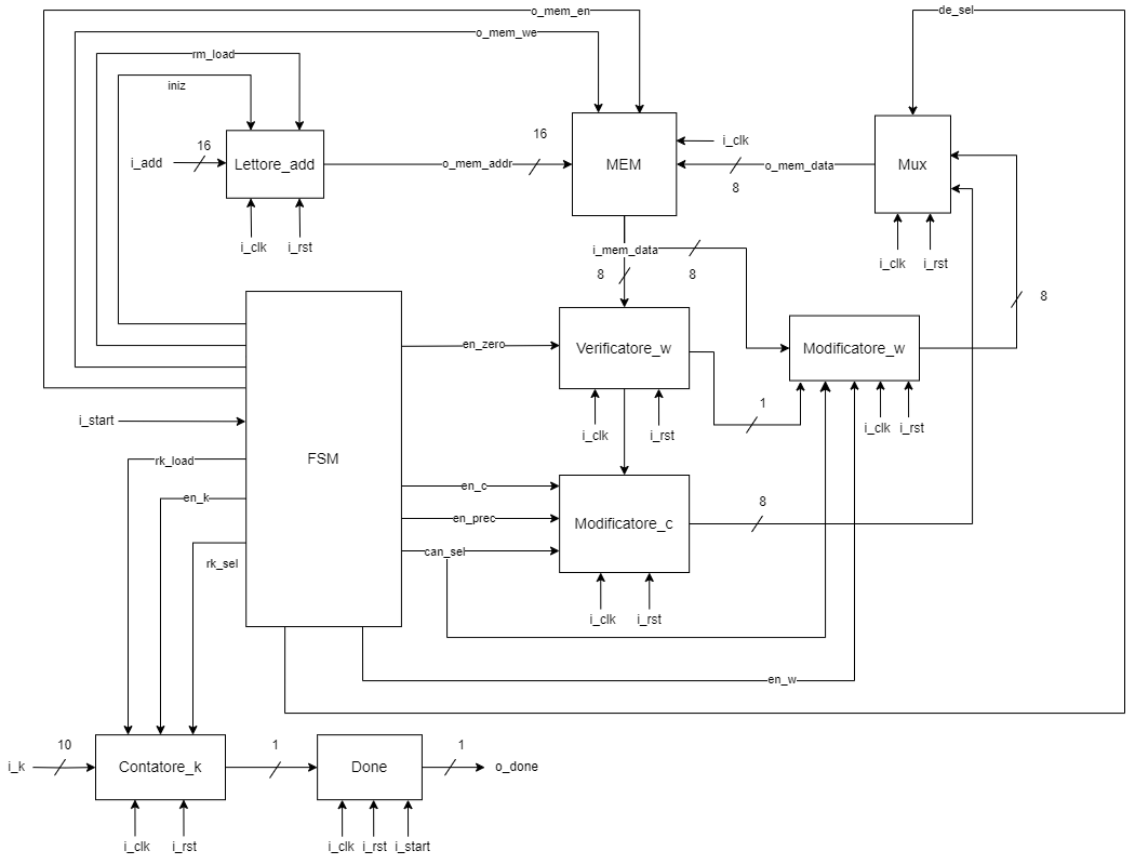


Figura 3: Design di architettura iniziale

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.

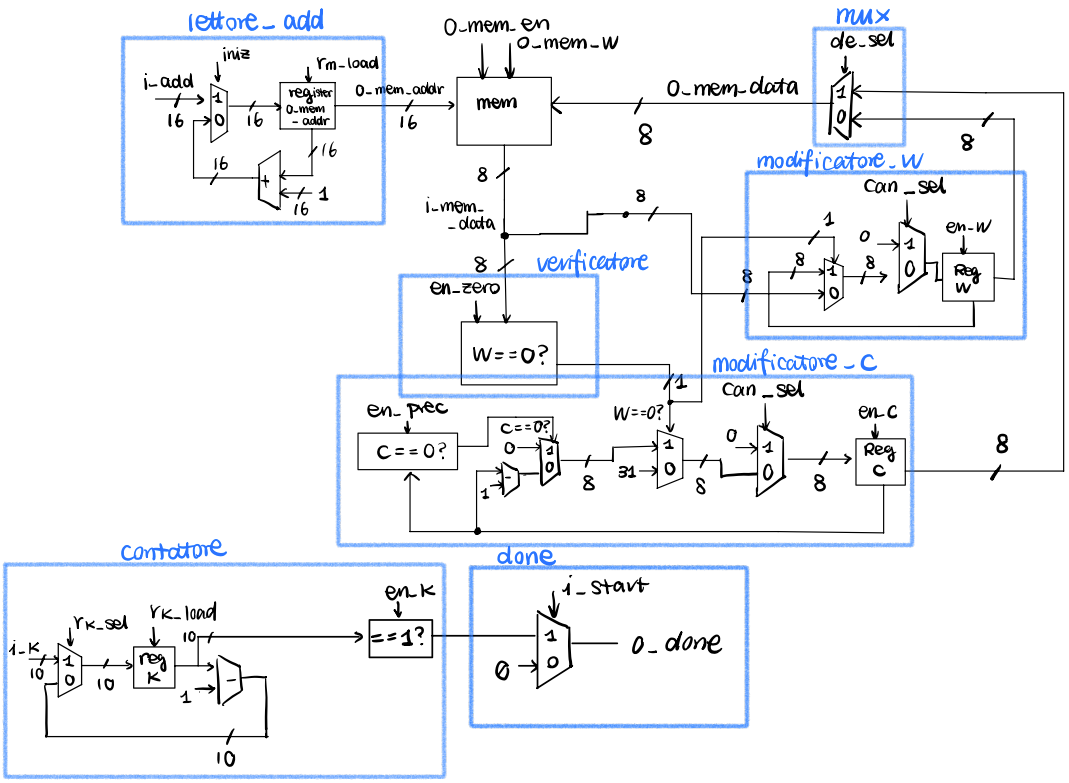


Figura 4: Design dettagliato iniziale

- 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 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 finisce la scrittura di credibilità C.
- Done: collegato con il Contatore_K, serve per gestire l'output o_done.

2.1.3 Parte III - segnali principali

In questa sezione, descriviamo solo i segnali gestiti dalla macchina a stati e approfondiremo nella sezione successiva in modo dettagliato tale 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

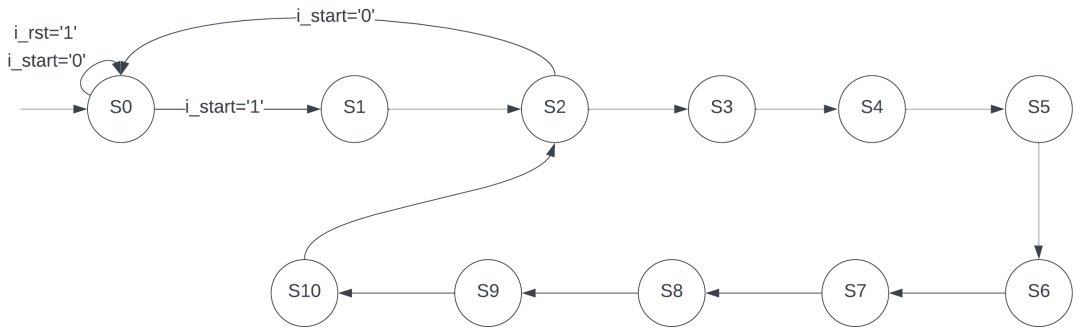


Figura 5: Diagrammi di stati

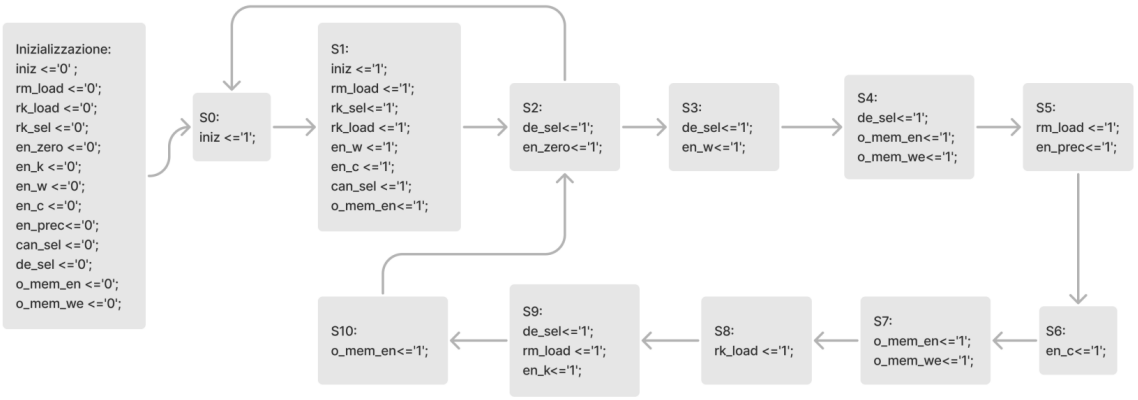


Figura 6: Output di stati

Per parte di controllo, ho deciso di utilizzare una macchina di Moore in cui le uscite sono determinate in funzione dei soli stati correnti. Per migliorare la visualizzazione, non sono state disegnate tutte le frecce per reset e i output sono riportati in un grafico 6separato.
Le funzionalità che svolgono in vari stati:

Tutti i segnali sono inizializzati con zero, quindi se non viene specificato il valore, per default le uscite hanno un valore uguale al zero.

- 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.

- Stato 2:
  1. Riceve il valore letto dalla memoria e attiva il verificatore di zero.
  2. Seleziona il blocco di modificatore_W.

- 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 il valore salvato precedentemente in registro C è uguale a zero oppure no.

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

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

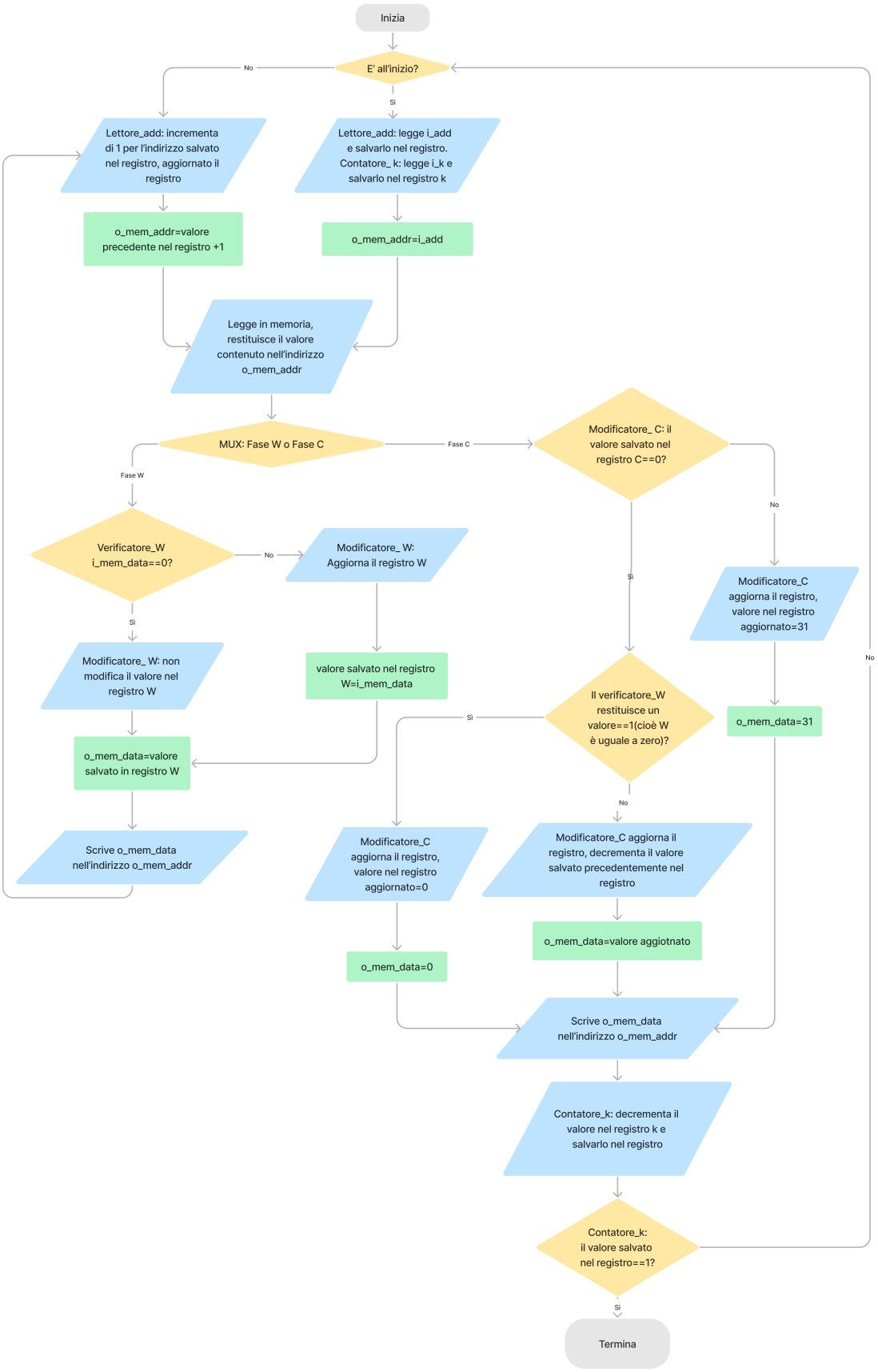
- Stato 8:
  1. Decrementa il valore i_k con il contatore.

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

- Stato 10:
  1.1 Stato realizza o_done=1 e i_start si abbassa a 0 ma viene accorta solo un clock dopo quindi quando arriva allo stato 2 si ritorna allo stato 0.
  1.2 Inizia la lettura in memoria.

2.3 Gestione dei processi

Per descrivere i processi, ho progettato un digramma di flusso:



2.4 Risultato finale dell'architettura

Grazie al comando Open Elaborated Design, siamo riusciti ad ottenere la schermata seguente che è generata automaticamente da Vivado.

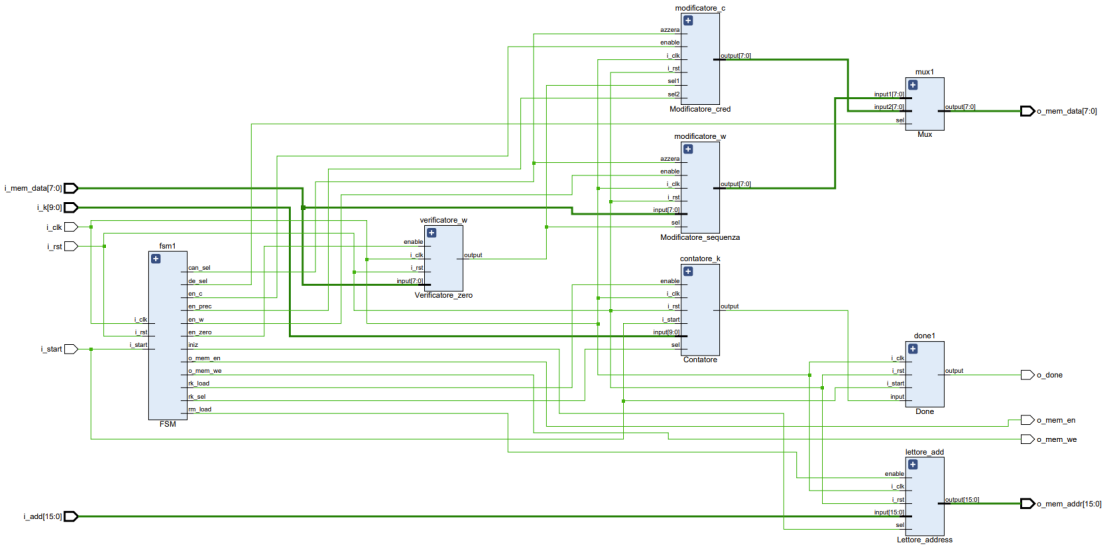


Figura 7: Architettura finale

Si può notare la compatibilità con il design iniziale.

3 Risultati sperimentali

In seguendo, presentiamo i risultati ottenuti usando i Test Bench.

3.1 Test Bench Example

Innazitutto, iniziamo con l'analisi del comportamento del progetto con il Test Bench fornito, suddividendo in Behavioral e Post-Synthesis.

3.1.1 Behavioural

Analizzando la figura 8 del risultato di test, possiamo concludere che il progetto soddisfa le seguenti richieste:

1. Durante reset: o_done=0.
2. Fase iniziale: o_done=0 dopo reset prima di start=1.
3. Fase dell'elaborazione: o_done rimane a 0.
4. Fase fine l'erabolazione: o_done sale a 1 e start scende nel fronte di salita di i_clk quando scopre che o_done=1, e nel prossimo fronte di salita o_done si riabassa a 0.
5. O_mem_en=0 quando done=1.
6. Restituisce l'output desiderato.

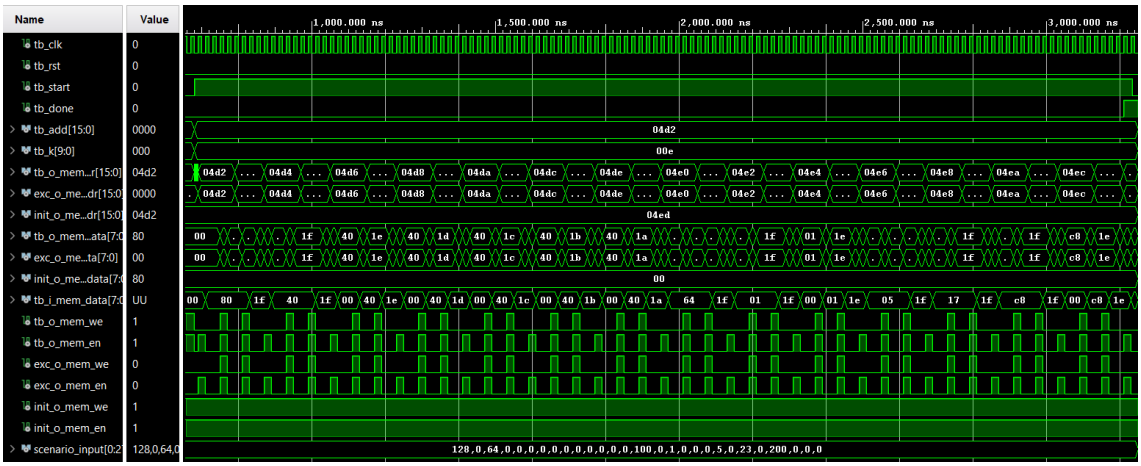


Figura 8: L'andamento dei segnali pre-sintesi

3.1.2 Post-Synthesis

Dai dati generati dal Report Utilization, si nota che nel progetto sono stati utilizzati 54 Flip Flop e non è presente nessun Latch.

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

Invece per quando riguarda al Report Timing(fig.10), il progetto ha il slack da 16.467 ns che rappresenta la differenza tra il tempo necessario per produrre un output e il tempo di clock, cioè il progetto impiega solo 3.533 ns per ottenere l'output desiderato che è un valore abbastanza basso rispetto al 20 ns.

```
Timing Report

Slack (MET) :             16.467ns  (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:        3.151ns  (logic 0.901ns (28.594%)  route 2.250ns (71.406%))
  Logic Levels:           2  (LUT4=1 LUT6=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 22.100 - 20.000 )
    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
```

Figura 10: Report Timing

3.2 Test i casi limiti

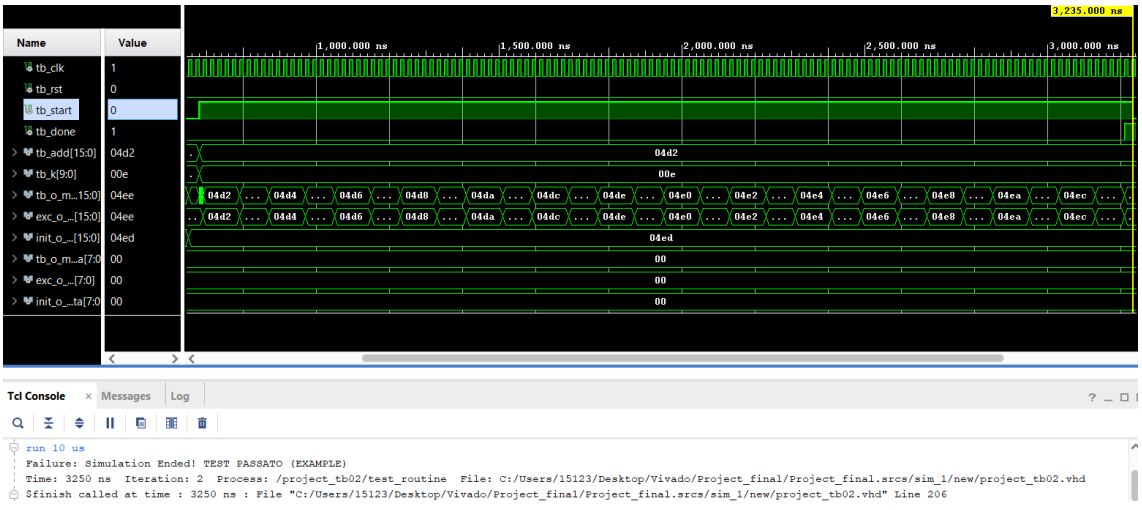
Una volta superato il Test Bench fornito, proseguo con la fase di test con i Test Bench generati con gli esempi forniti dalla specifica. Dopodiché con l'ultima fase di test si considerano i casi limiti del progetto.

3.2.1 Caso 1: solo zeri

Il caso in cui la sequenza di stringa è fatta solo dagli zeri, per testare i valori di credibilità non andranno sotto gli zeri e i valori di stringa devono rimanere invariati, inoltre in questo caso dovrebbe generare come l'output formati da tutti e soli zeri.

```
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 );
```

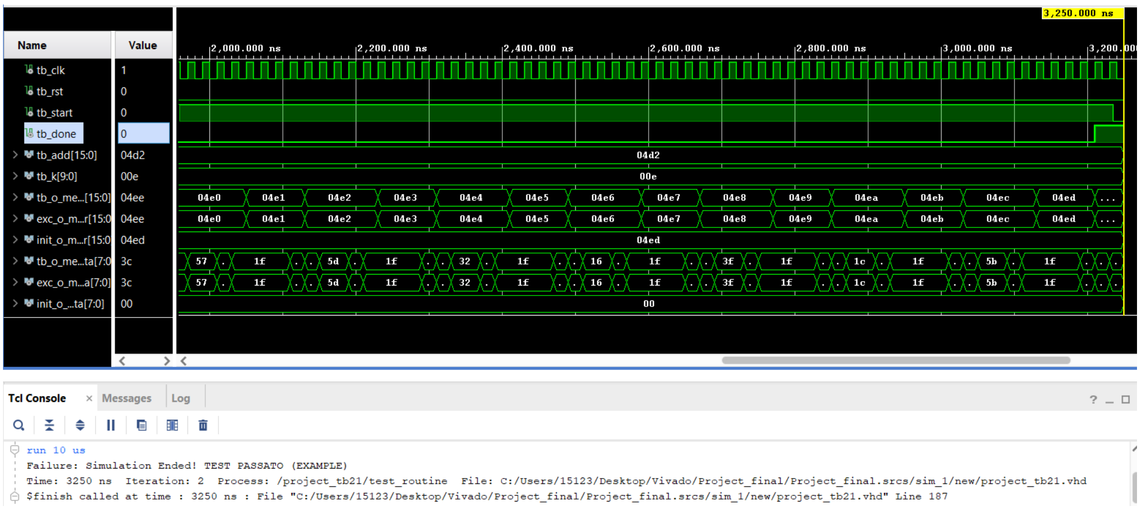


3.2.2 Caso 2: tutti i numeri diversi da zeri

Per testare come si comporta il progetto di fronte al caso quando non bisogna incrementare mai la credibilità e fare la sostituzione della parola in sequenza.

```
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 );
```

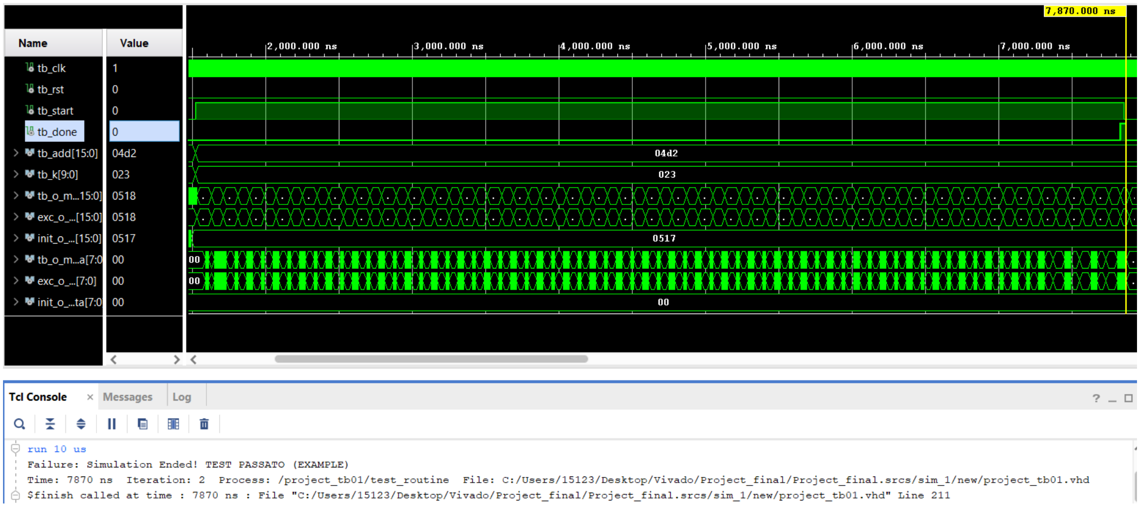


3.2.3 Caso 3: un numero non zero seguito da più di 31 zeri consecutivi

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.

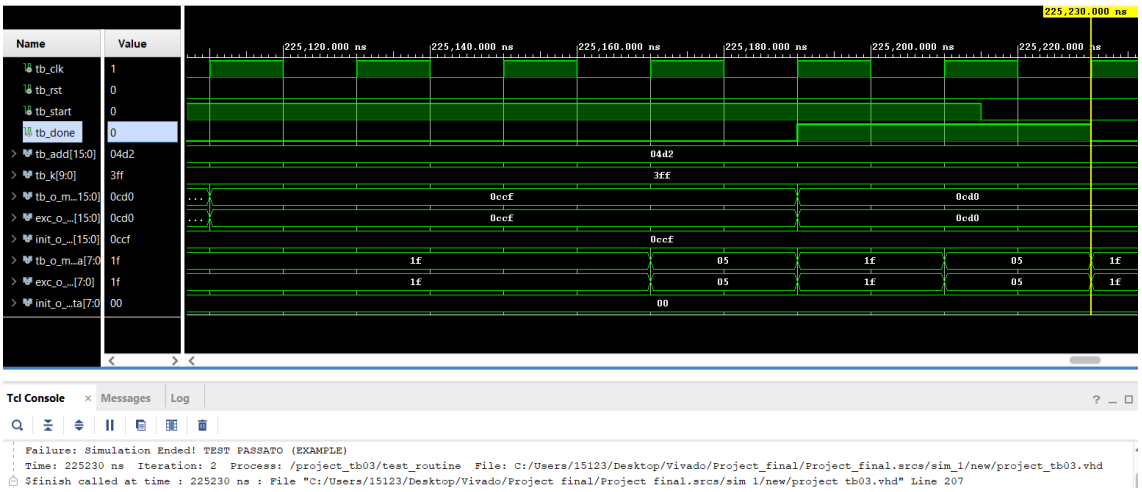
```
constant SCENARIO_LENGTH : integer := 35;
type scenario_type is array (0 to SCENARIO_LENGTH*2-1) of integer;

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 );
```



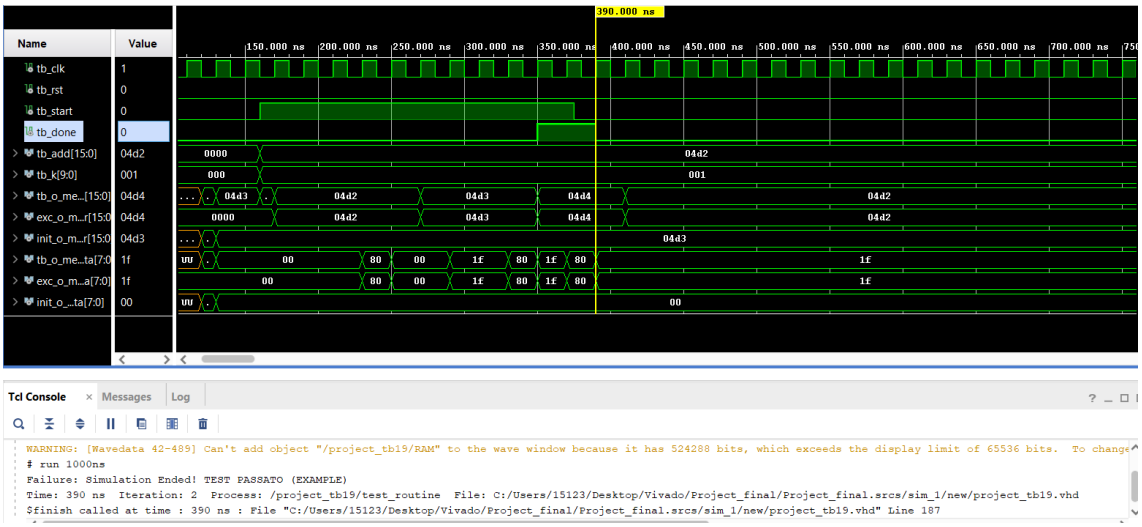
3.2.4 Caso 4: con i_K massimo

Per testare quando K aggiunge il valore massimo, come si comporta il progetto.



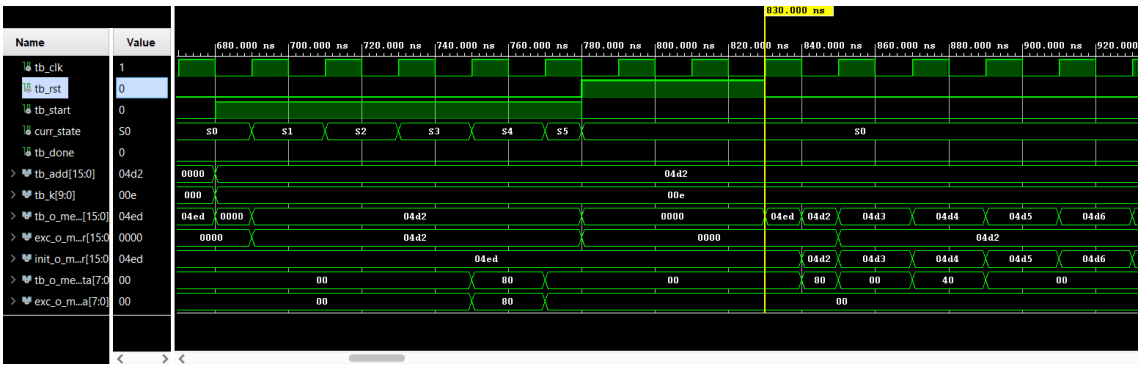
3.2.5 Caso 5: con i_K min

Per testare se il progetto funziona anche nel caso estremo ovvero elementare quando k è uguale a 1.



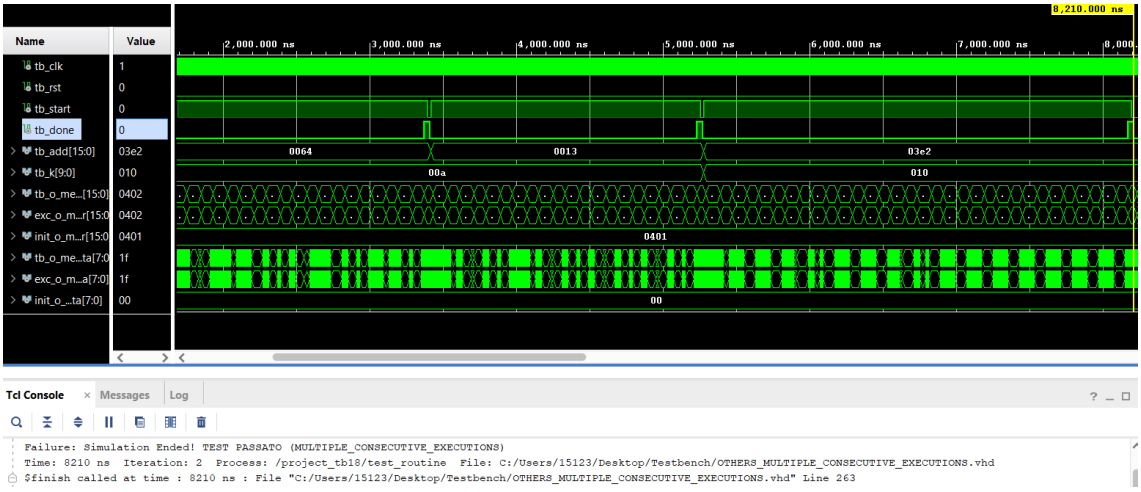
3.2.6 Caso 6: i_rst sale durante l'esecuzione

Per testare il comportamento del progetto quando sale il segnale di reset durante la fase dell'elaborazione.



3.2.7 Caso 7: i_start consecutivi

Per vedere come il progetto reagisce quando bisogna gestire in modo sequenziale le stringhe in ingresso.



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

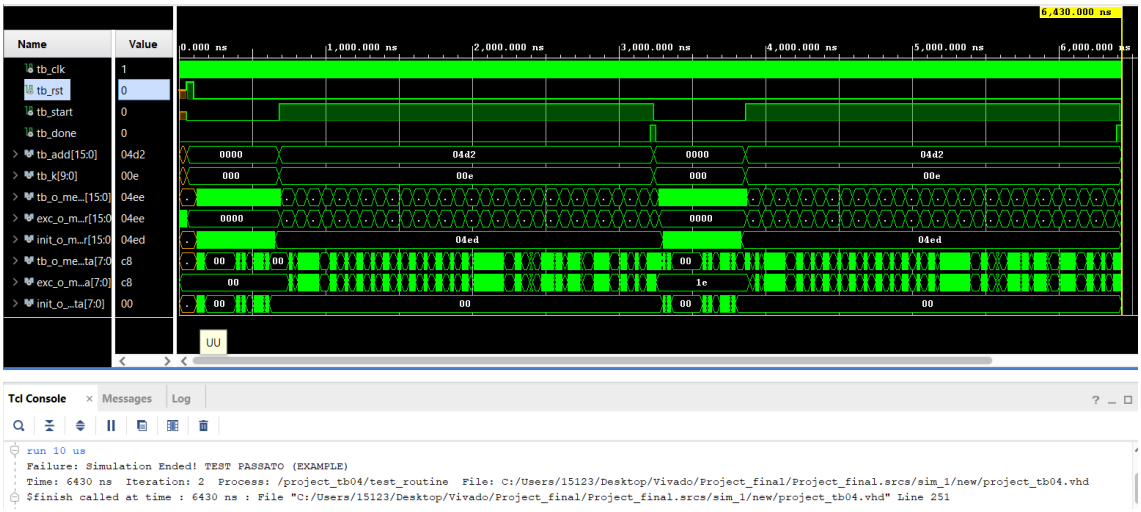
Per testare che cosa succederebbe quando vengono scritto più volte consecutivamente negli stessi indirizzi con la stessa sequenza di ingresso.

```
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, 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 il processo di sviluppo, all'inizio mi trovavo un po' in difficoltà di orientarmi, 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. Ho deciso di iniziare la fase di design disegnando i componenti sulla carta, dopo una serie di prove riporto tutte le idee usando il piattaforma draw.io, inoltre prima di passare all'implementazione del codice ho stabilito anche una bozza per la macchina a stati in modo da poter mantenere un flusso chiare e ragionevole per le fasi successive. Durante la fase di coding sono dovuta a modificare un ben po' design iniziale per rendere il progetto che soddisfi le richieste descritto nella specifica. Una volta superato il Test Bench fornito, si prosegue con la valutazione per i casi limiti. Alla fine, sono riuscita a completare il progetto che è in grado di trasformare l'input in l'output corretto in un tempo adeguato. Si tratta di un'epserienza più che positiva, grazie alla quale ho imparato di sviluppare individualmente un progetto, pensare in modo più razionale per risolvere gli eventuali problemi incontrati.