POLITECNICO
MILANO 1863

PROVA FINALE (PROGETTO DI RETI LOGICHE)

Prof. Palermo Gianluca - Anno Accademico 2023/2024

Rubagotti Andrea

(Cod. persona: 10780379 - Matricola: 982036)

Santagata Maria Concetta

(Cod. persona: 10799787- Matricola: 987495)

INDICE

1. INTRODUZIONE

Descrizione del progetto

Il progetto ha come scopo quello di implementare un componente hardware, mediante la stesura di un file in linguaggio VHDL, che sia in grado di svolgere il seguente compito.

Viene fornita una sequenza di un numero K di parole, il cui valore è compreso tra 0 e 255. I valori della sequenza non devono essere interpretati tutti allo stesso modo, in quanto si alternano valori numerici e valori di credibilità. Per questo motivo la sequenza di parole viene memorizzata a partire da un indirizzo specificato, ogni 2 byte.

Il byte compreso tra i due è quello di credibilità, e questo, al termine dell'elaborazione, verrà correttamente inserito in memoria.

Il valore di credibilità vale 31 successivamente ad ogni valore diverso da 0. Se invece la parola contiene uno 0, questo deve essere sostituito con l'ultimo valore letto, e la relativa credibilità deve essere decrementata di uno (se essa raggiunge il valore 0, allora non viene più decrementata).

Corner-case: se la sequenza comincia con uno zero, allora il relativo valore di credibilità non è 31, bensì rimane 0, fintanto che nella sequenza memorizzata non compare un valore diverso da 0.

Di seguito vengono forniti due esempi con valori decimali. I numeri in **verde** sono i valori numerici, gli altri sono invece i valori di credibilità. Nel secondo possiamo notare come venga correttamente trattato il corner-case descritto in precedenza.

ESEMPIO 1:

Sequenza iniziale: **3**, 0, **5**, 0, **0**, 0, **7**, 0, **0**, 0, **4**, 0, **2**, 0

Sequenza finale: **3**, 31, **5**, 31, **5**, 30, **7**, 31, **7**, 30, **4**, 31, **2**, 31

ESEMPIO 2:

Sequenza iniziale: **0**, 0, **0**, 0, **8**, 0, **0**, 0, **0**, 0, **0**, 0, **21**, 0

Sequenza finale: **0**, 0, **0**, 0, **8**, 31, **8**, 30, **8**, 29, **8**, 28, **21**, 31

Descrizione degli input e degli output

Il componente riceve in ingresso tre segnali primari: i_start, variando permette l'elaborazione o meno della stringa data (rispettivamente segnale ad 1 "alto" e a 0 "basso"), i_add, che indica l'indirizzo di memoria da cui partire per la lettura e la conseguente scrittura della stessa e, infine, i_k, che rappresenta il numero di parole da leggere. Inoltre, l'elaborazione della stringa è gestita dai segnali i_rst e i_clk.

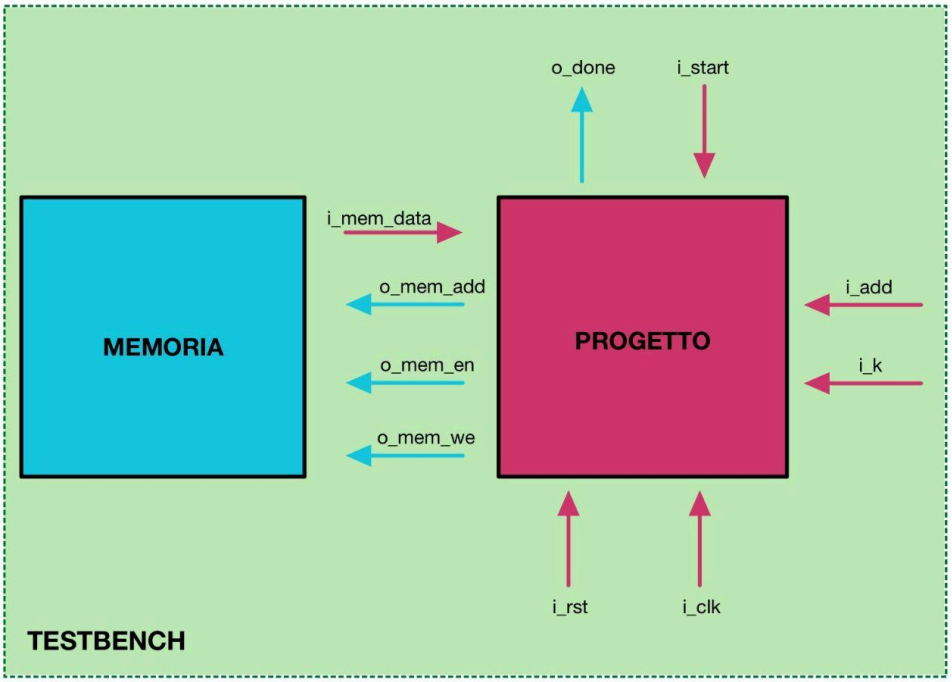
I segnali del componente sono tutti sincroni, quindi da interpretare sul fronte di salita del clock, eccezione fatta per i_rst (asincrono). Per quanto riguarda gli output, il componente è caratterizzato dal segnale o_done che si presenta alto al termine di ogni elaborazione e basso viceversa.

Descrizione della memoria

La memoria non è stata sintetizzata nel progetto, in quanto è già istanziata all'interno dei Test Bench (come si può osservare nell'immagine sottostante). Questa possiede un indirizzamento al byte.

Di seguito viene fornita una breve spiegazione dei segnali:

- i_mem_data è un byte (8 bit) di dati provenienti dalla memoria, dopo una richiesta di lettura;
- o_mem_add è l'indirizzo in cui si vuole leggere o scrivere, questo è composto da 2 byte (16 bit);
- o_mem_en è il segnale (1 bit) che permette di abilitare la memoria, sia alla scrittura sia alla lettura, quando esso vale 1;
- o_mem_we è il segnale (1 bit) che permette di abilitare la scrittura in memoria, nel momento in cui esso vale 1. Durante l'operazione di lettura, invece, questo segnale sarà posto a 0.



Descrizione dell'interfaccia del componente

Il componente utilizzato è caratterizzato dalla seguente interfaccia:

```
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;
```

Dunque, i segnali sono:

Input:

- i_clk: generato dal Test Bench, in base al quale interpreto tutti gli input/output sincronizzati;
- i_rst: unico segnale asincrono, che permette l'inizio dell'elaborazione di una nuova stringa di parole, inizializzando il componente;
- i_start: generato dal Test Bench, decisivo per permettere la lettura, con conseguente elaborazione, della stringa;
- i_add: generato dal Test Bench, indica l'indirizzo di memoria a partire dal quale è memorizzata la stringa;
- i_k: generato dal Test Bench, rappresenta il numero di valori nella sequenza (valori con annesse credibilità);
- i_mem_data: proviene dalla memoria, fornisce il dato letto.

Output:

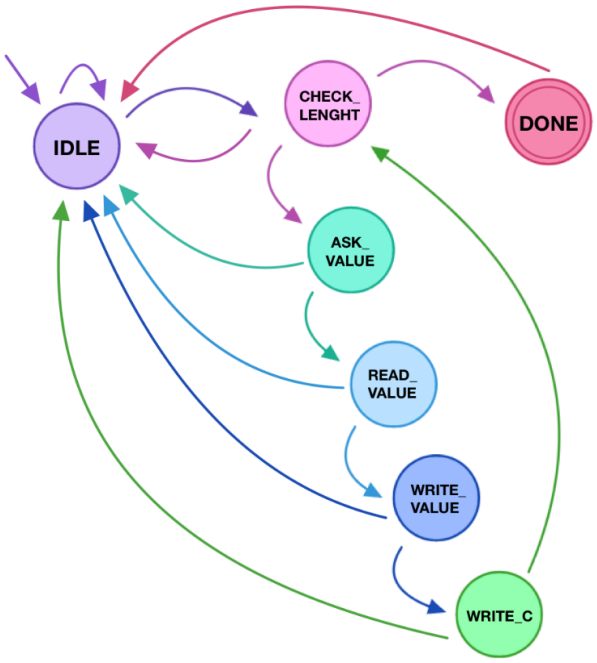
- o_done: indica se l'elaborazione sia terminata o meno;
- o_mem_addr: rappresenta l'indirizzo di memoria al quale viene scritto il valore o la credibilità;
- o_mem_data: è il dato da inserire allo specifico indirizzo di memoria;
- o_mem_en: indica se è possibile la lettura o la scrittura del dato, in combinazione con il segnale o_mem_we (in ambo i casi o_mem_en dev'essere pari ad 1, viceversa non avverrebbe nessuna delle due);
- o_mem_we: permette la scrittura nel caso in cui sia pari ad 1, la lettura altrimenti.

2. ARCHITETTURA

Descrizione della macchina a stati finiti (FSM)

La macchina a stati finiti possiede i seguenti stati:

- **IDLE**: è lo stato iniziale, in cui vengono inizializzati i segnali dei registri "next". Inoltre, lo stato IDLE funge anche da stato di reset, cioè, ogni qual volta, durante l'esecuzione del programma, venga dato in input i_rst = 1, allora il componente si re-inizializza e nel caso in cui non si abbia ancora i_start = 1, resta in questo stato. Nel momento in cui il segnale di i_start viene posto a 1, allora si passa allo stato CHECK_LENGHT;
- **CHECK_LENGHT**: in questo stato si verifica se siano stati controllati tutti i valori della stringa (con le relative credibilità), che sono pari al numero di i_k. In caso affermativo, allora la macchina passa allo stato DONE, altrimenti passa allo stato ASK_VALUE;
- **ASK_VALUE**: in questo stato la memoria viene abilitata per la lettura (ponendo i_mem_en a 1 e i_mem_we a 0), e viene passato alla memoria stessa l'indirizzo su cui effettuare la lettura (tramite il segnale o_mem_addr). Lo stato seguente è READ_VALUE;
- **READ_VALUE**: in questo stato viene letto dalla memoria il byte richiesto (tramite il segnale i_mem_data), il quale viene salvato solo se diverso da 0. Si aggiorna inoltre il valore di credibilità relativo, decrementandolo se esso è maggiore di 0, o ripristinandolo a 31 se il valore letto è diverso da 0. La credibilità verrà scritta nello stato WRITE_C. Lo stato seguente è WRITE_VALUE;
- **WRITE_VALUE**: in questo stato la memoria viene abilitata per la scrittura (ponendo sia i_mem_en che i_mem_we a 1) e la costante precedentemente memorizzata, viene scritta in memoria. Se ci si trova nella posizione della costante stessa, essa viene sovrascritta rimanendo invariata (in quanto è quella che era stata memorizzata al passo precedente), altrimenti se è presente uno 0 esso viene sovrascritto con l'ultimo valore memorizzato. Viene anche incrementato l'indirizzo successivo di o_mem_addr, per poter permettere la successiva scrittura della credibilità nella cella successiva. Lo stato seguente è WRITE_C;
- **WRITE_C**: in questo stato la memoria è ancora abilitata per la scrittura, in quanto non vengono modificati i valori di i_mem_en e i_mem_we rispetto allo stato precedente. Viene dunque scritto in memoria il valore di credibilità precedentemente stabilito. Inoltre viene aumentato di un byte l'indirizzo di memoria memorizzato, per poter permettere un'ulteriore lettura nell'eventuale parte rimanente della stringa. Lo stato seguente è CHECK_LENGHT;



- **DONE**: si giunge in questo stato nel momento in cui l'elaborazione della stringa è terminata ed essa è stata salvata con successo in memoria. Il segnale di o_done viene posto a 1 per notificare la fine della procedura.

Nota bene: da ogni stato è possibile raggiungere lo stato IDLE, questo avviene quando si ha in input un segnale i_rst pari ad 1 (componente re-inizializzato).

Descrizione dei segnali

Qui di seguito sono riportati i segnali che sono stati usati nel progetto, unitamente ad una loro breve descrizione:

- signal next_state, curr_state: state_type ⇒ indicano rispettivamente lo stato prossimo e il corrente;
- signal o_done_nx : std_logic ⇒ è il prossimo valore che deve assumere il segnale o_done;
- signal o_mem_addr_nx : std_logic_vector (15 downto 0) ⇒ è il prossimo valore che deve assumere il segnale o_mem_addr;
- signal o_mem_data_nx : std_logic_vector (7 downto 0) ⇒ è il prossimo valore che deve assumere il segnale o_mem_data;
- signal o_mem_en_nx : std_logic ⇒ è il prossimo valore che deve assumere il segnale o_mem_en;
- signal o_mem_we_nx : std_logic ⇒ è il prossimo valore che deve assumere il segnale o_mem_we;
- signal old_value, old_value_nx : std_logic_vector (7 DOWNTO 0) ⇒ rappresentano rispettivamente il valore letto e quello aggiornato;
- signal old_c, old_c_nx : std_logic_vector (7 DOWNTO 0) ⇒ rappresentano rispettivamente il valore di credibilità letto e quello aggiornato;
- signal visited_words, visited_words_nx : std_logic_vector(9 DOWNTO 0) ⇒ rappresentano rispettivamente le parole lette e il nuovo numero delle stesse;
- signal updated_add, updated_add_nx : std_logic_vector(15 DOWNTO 0) ⇒ rappresentano rispettivamente il valore vecchio e quello aggiornato dell'indirizzo di lettura/scrittura;
- signal first_nx, first: boolean ⇒ è un booleano utile per capire se il componente è stato inizializzato e quindi siamo nel primo passaggio dell'elaborazione della stringa; indicano valore aggiornato e vecchio del segnale.

Descrizione dei processi (VHDL processes) e scelte progettuali

Il componente possiede due processi concorrenziali differenti:

1. Il primo processo si occupa della parte sequenziale, gestisce il reset e aggiorna i segnali sul fronte di salita del clock, andando a memorizzare nel registro il valore salvato nel registro "next";
2. Il secondo processo, invece, si occupa della gestione della macchina a stati finiti. È quindi principalmente formato da "Case-When statements" che compiono determinate azioni in base allo stato attuale in cui ci si trova.

3. RISULTATI SPERIMENTALI

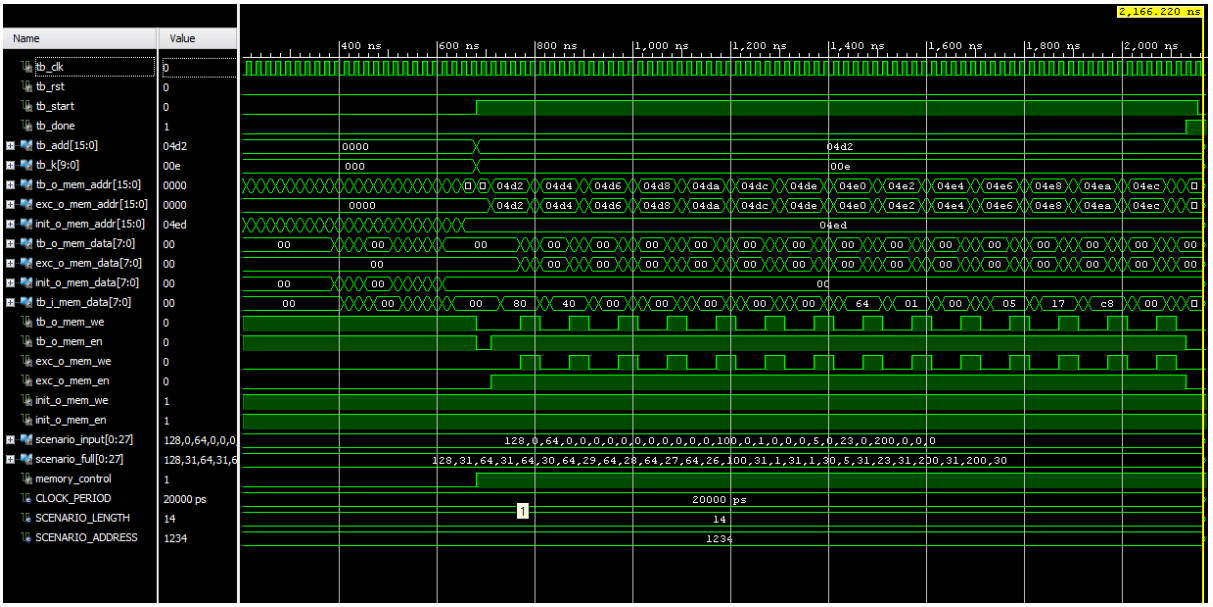
Uno step fondamentale per lo sviluppo del nostro progetto è stato quello riguardante il testing. Dopo aver correttamente superato il Test Bench fornito dal docente (come spiegato nel paragrafo qui di sotto), abbiamo deciso di creare noi alcuni Test Bench, che vadano a controllare determinate condizioni limite, che saranno spiegate di seguito.

Per ogni Test Bench sono date in allegato la foto del visualizzatore di onde (Waveform viewer) e anche il messaggio stampato dalla TCL console.

Infine, sono stati eseguite con successo anche le simulazioni post-sintesi, in modo tale da effettuare una verifica più approfondita della nostra implementazione.

Test Bench fornito dal docente (project_tb)

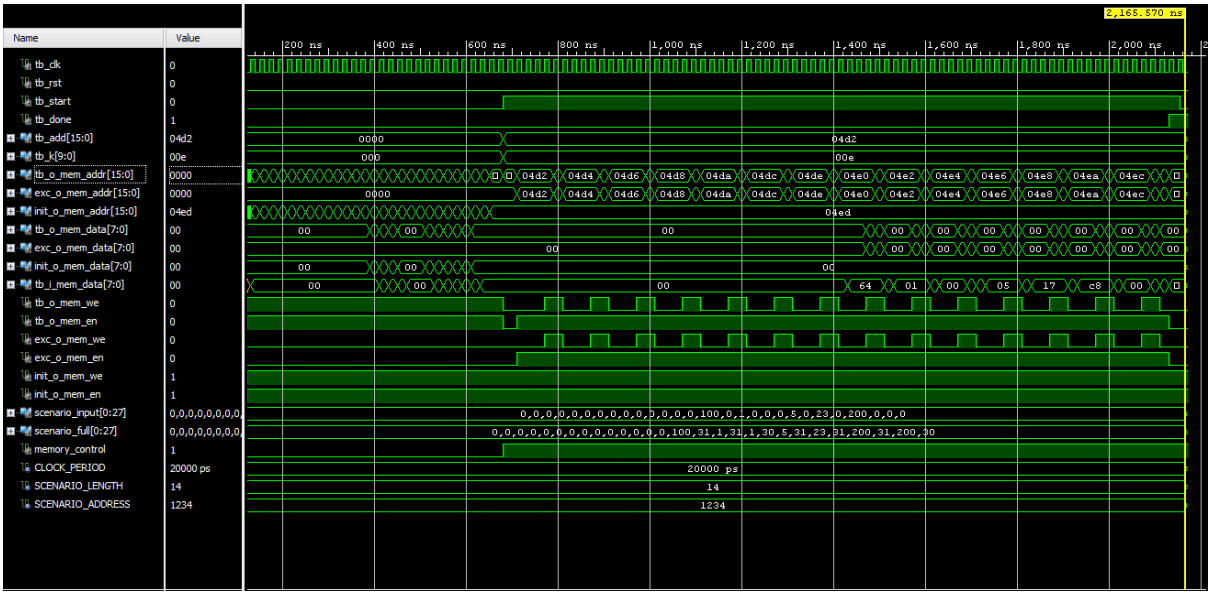
Questo Test Bench è risultato utile per verificare un generico caso base del nostro progetto, che si è rivelato funzionare correttamente.




```
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 2170 ns  Iteration: 1  Process: /project_tb/test_routine  File: D:/Download/project_tb.vhd
$finish called at time : 2170 ns : File "D:/Download/project_tb.vhd" Line 187
INFO: [USF-XSim-96] XSim completed. Design snapshot 'project_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 9000ns
launch_simulation: Time (s): cpu = 00:00:00 ; elapsed = 00:00:14 . Memory (MB): peak = 958.207 ; gain = 0.000
```

Test Bench “beginning_with_zero”

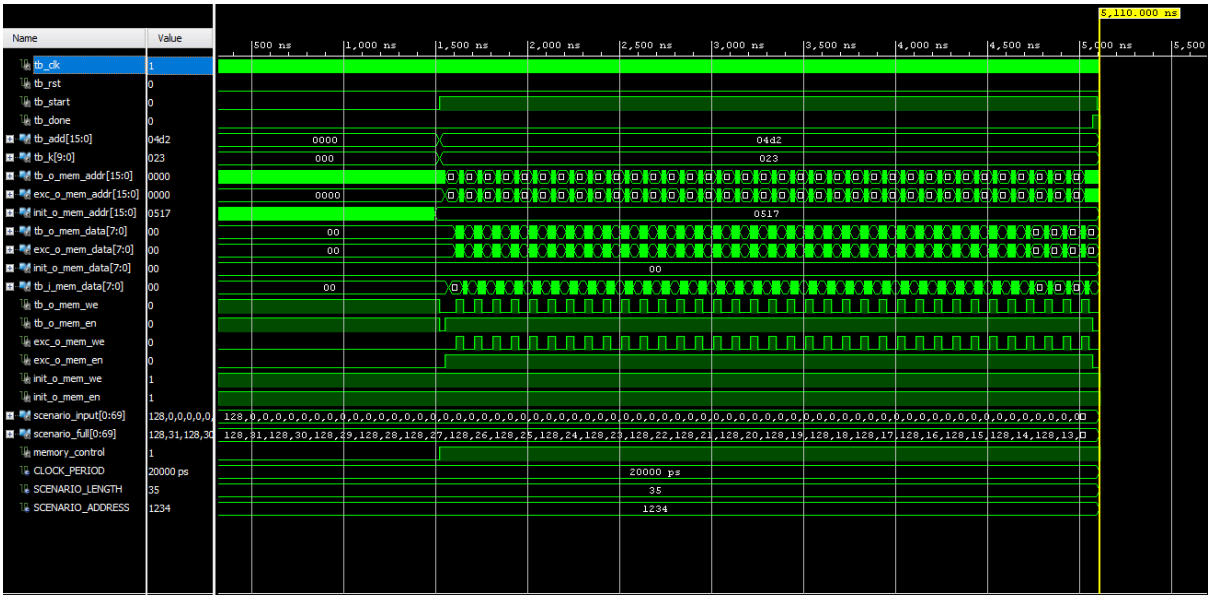
Il primo Test Bench da noi scritto va a verificare uno specifico corner-case, ovvero il caso in cui la sequenza iniziasse con un certo numero di zeri consecutivi. In questo caso i relativi valori di credibilità dovevano essere mantenuti costanti a 0 fintantoché la sequenza è composta da soli zero.




```
# run 1000000ns
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 2170 ns  Iteration: 1  Process: /tb_beginning_withzero/test_routine  File: C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/
$finish called at time : 2170 ns : File "C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/new/tb_beginning_withzero.vhd" Line 187
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_beginning_withzero_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000000ns
launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:10 . Memory (MB): peak = 1878.746 ; gain = 0.000
```

Test Bench “credibility_to_zero”

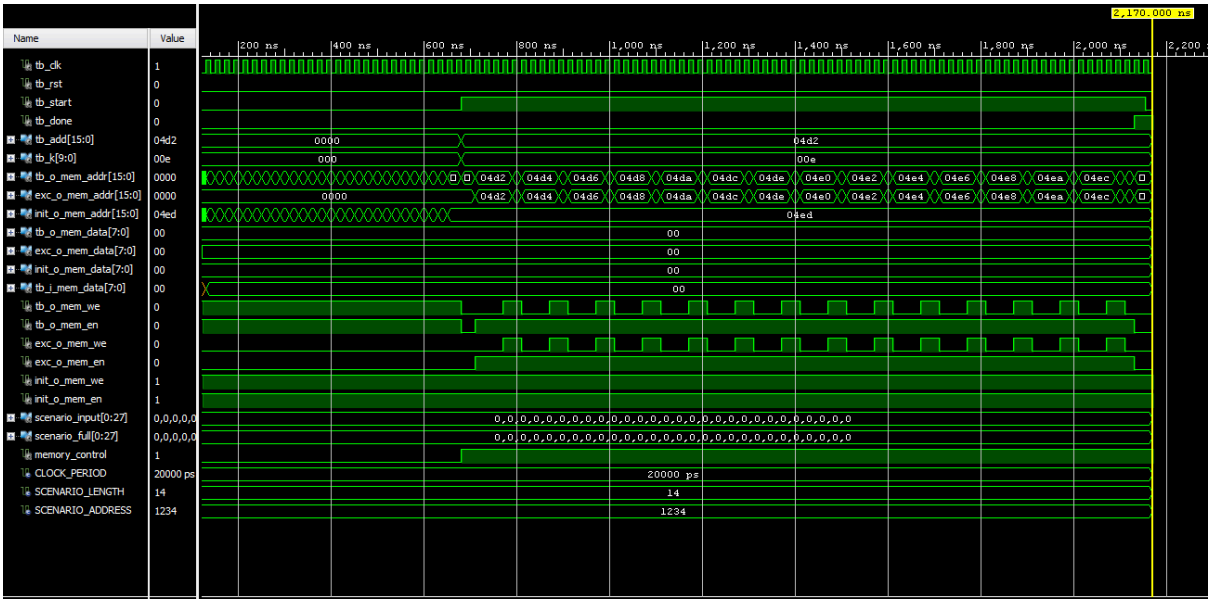
Il secondo Test Bench da noi scritto va a verificare un altro corner-case, che si verifica nel momento in cui ci sono svariati zeri consecutivi nel mezzo della sequenza. In questo modo la credibilità raggiunge il valore zero, e non dovrà essere decrementata ulteriormente.




```
# run 1000000ns
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 5110 ns  Iteration: 1  Process: /tb_credibility_to_zero/test_routine  File: C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/new/t
$finish called at time : 5110 ns : File "C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/new/tb_credibility_to_zero_arch.vhd" Line 185
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_credibility_to_zero_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000000ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:11 . Memory (MB): peak = 1878.746 ; gain = 0.000
```

Test Bench “only_zero”

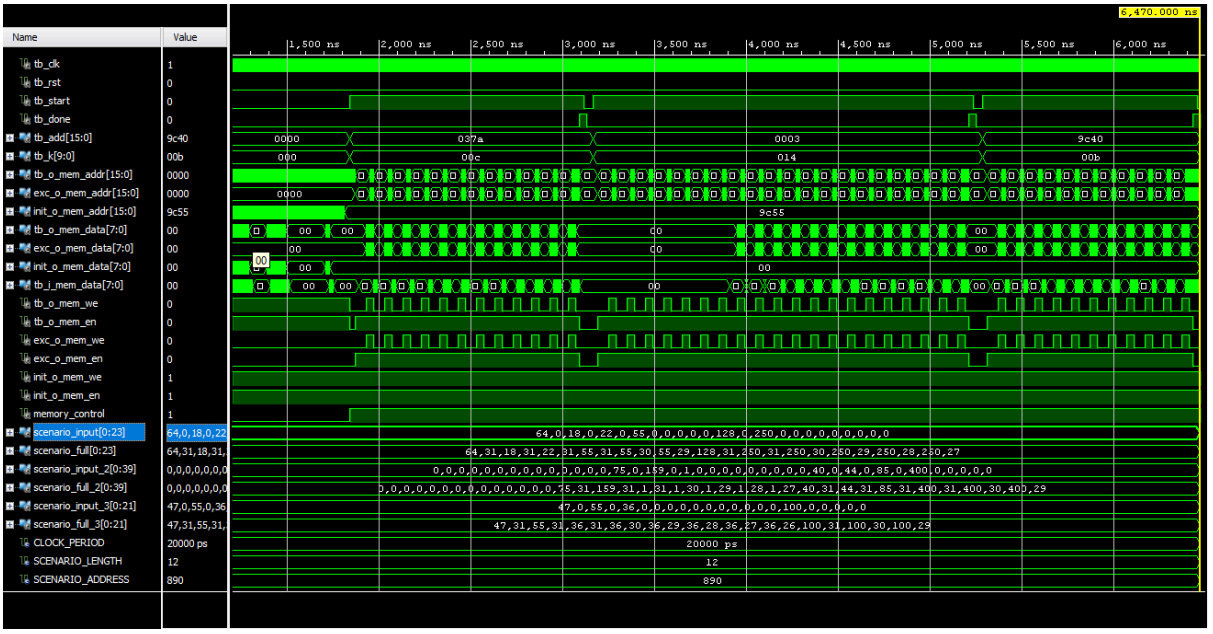
Il terzo Test Bench da noi scritto ha come finalità quella di verificare un corner-case simile al primo. Tuttavia in questo caso la sequenza è composta esclusivamente da zeri. Di conseguenza anche la sequenza di output dovrà essere formata solamente da zeri.




```
# run 1000000ns
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 2170 ns  Iteration: 1  Process: /tb_onlyzero/test_routine  File: C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/new/tb_onlyzero.vhd
$finish called at time : 2170 ns : File "C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/new/tb_onlyzero.vhd" Line 185
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_onlyzero_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000000ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:10 . Memory (MB): peak = 1878.746 ; gain = 0.000
```

Test Bench “multiple_sequences”

Il quarto Test Bench da noi scritto ha come finalità quella di verificare il caso in cui vengano fornite più sequenze da codificare, una dietro l’altra, senza dare alcun segnale di reset al modulo nel mezzo. Infatti il reset è assicurato solamente all’inizio, ovvero prima che parta la prima elaborazione.

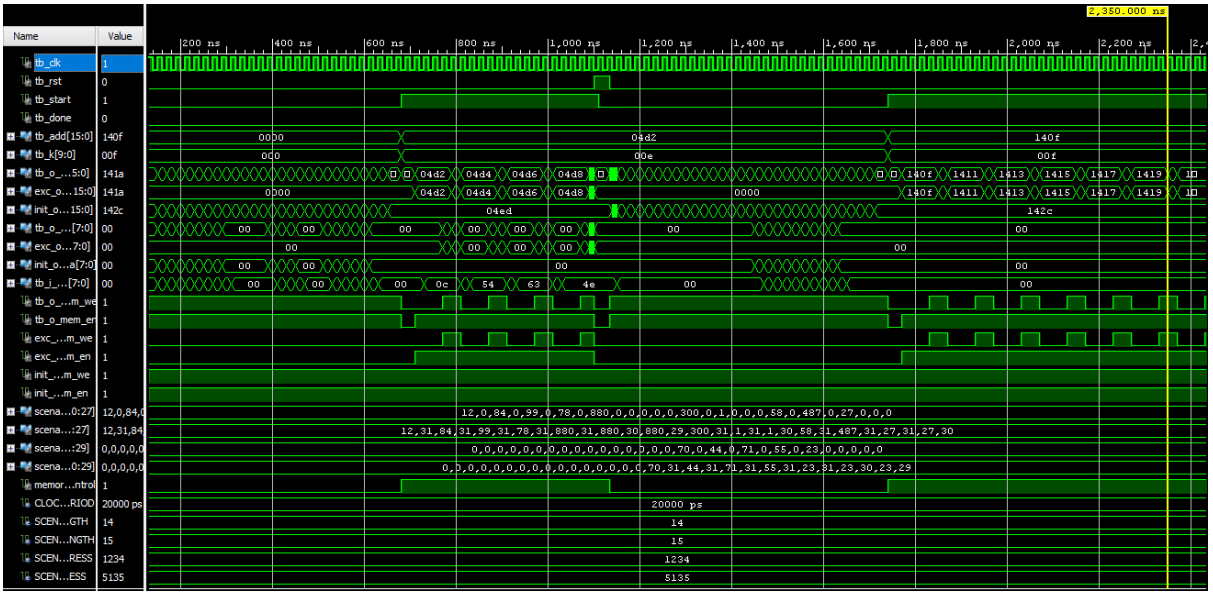



```
# run 1000000ns
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 6470 ns  Iteration: 1  Process: /tb_different_input/test_routine  File: C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/
$finish called at time : 6470 ns : File "C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/new/tb_different_input.vhd" Line 304
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_different_input_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000000ns
launch_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:11 . Memory (MB): peak = 1889.371 ; gain = 0.000
```

Test Bench “reset_during_elaboration”

Il quinto Test Bench da noi scritto è risultato essenziale per verificare un caso limite, che sussiste nel caso in cui al modulo venga fornito un segnale di reset alto (“1”) mentre il modulo sta compiendo un’elaborazione.

L’elaborazione in corso viene così interrotta, e ne viene poi iniziata una nuova (con un nuovo segnale di i_start).




```
# run 1000000ns
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 3330 ns  Iteration: 1  Process: /multiple_resets_during_elaboration/test_routine  File: C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/
$finish called at time : 3330 ns : File "C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.srcs/sim_1/new/multiple_resets_during_elaboration.vhd" Line 238
INFO: [USF-XSim-96] XSim completed. Design snapshot 'multiple_resets_during_elaboration_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000000ns
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:10 . Memory (MB): peak = 2043.273 ; gain = 0.000
```

REPORT

Qui, di seguito, sono riportati i report ottenuti dalla sintesi e dall'implementazione del nostro progetto.

Vivado Synthesis Report

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |     6|
|3     |LUT1   |    16|
|4     |LUT2   |    10|
|5     |LUT3   |    26|
|6     |LUT4   |    27|
|7     |LUT5   |    10|
|8     |LUT6   |    19|
|9     |FDCE   |    43|
|10    |FDPE   |     1|
|11    |FDRE   |    26|
|12    |LD     |    59|
|13    |IBUF   |    37|
|14    |OBUF   |    27|
+------+-------+------+

Report Instance Areas:
+------+---------+-------+------+
|      |Instance |Module |Cells |
+------+---------+-------+------+
|1     |top      |       |   308|
+------+---------+-------+------+
---------------------------------------------------------------------------------
Finished Writing Synthesis Report : Time (s): cpu = 00:00:37 ; elapsed = 00:00:47 . Memory (MB): peak = 660.926 ; gain = 451.426
---------------------------------------------------------------------------------

---------------------------------------------------------------------------------
Synthesis finished with 0 errors, 0 critical warnings and 15 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:00:20 ; elapsed = 00:00:28 . Memory (MB): peak = 660.926 ; gain = 109.797
Synthesis Optimization Complete : Time (s): cpu = 00:00:37 ; elapsed = 00:00:48 . Memory (MB): peak = 660.926 ; gain = 451.426
INFO: [Project 1-571] Translating synthesized netlist
INFO: [Netlist 29-17] Analyzing 102 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
  A total of 59 instances were transformed.
  LD => LDCE: 59 instances

INFO: [Common 17-83] Releasing license: Synthesis
18 Infos, 15 Warnings, 0 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
synth_design: Time (s): cpu = 00:00:36 ; elapsed = 00:00:41 . Memory (MB): peak = 660.926 ; gain = 451.426
INFO: [Common 17-1381] The checkpoint 'C:/Users/Lenovo/Desktop/project_reti_logiche/project_reti_logiche.runs/synth_2/project_reti_logiche.dcp' has been generated.
report_utilization: Time (s): cpu = 00:00:01 ; elapsed = 00:00:00.034 . Memory (MB): peak = 660.926 ; gain = 0.000
INFO: [Common 17-206] Exiting Vivado at Sun Aug 18 10:48:42 2024...
```

Utilization Report

2. Slice Logic Distribution

Site Type	Used	Fixed	Available	Util%
Slice	50	0	33450	0.15
SLICEL	36	0		
SLICEM	14	0		
LUT as Logic	88	0	133800	0.07
using O5 output only	0			
using O6 output only	83			
using O5 and O6	5			
LUT as Memory	0	0	46200	0.00
LUT as Distributed RAM	0	0		
LUT as Shift Register	0	0		
LUT Flip Flop Pairs	31	0	133800	0.02
fully used LUT-FF pairs	0			
LUT-FF pairs with one unused LUT output	31			
LUT-FF pairs with one unused Flip Flop	24			
Unique Control Sets	8			

Timing Summary Report

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             17.615ns  (required time - arrival time)
  Source:                 FSM_sequential_curr_state_reg[0]/C
                            (rising edge-triggered cell FDCE clocked by clk  {rise@0.000ns fall@10.000ns period=20.000ns})
  Destination:            o_mem_en_reg/D
                            (rising edge-triggered cell FDCE clocked by clk  {rise@0.000ns fall@10.000ns period=20.000ns})
  Path Group:             clk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            20.000ns  (clk rise@20.000ns - clk rise@0.000ns)
  Data Path Delay:        2.387ns  (logic 0.580ns (24.296%)  route 1.807ns (75.704%))
  Logic Levels:           1  (LUT2=1)
  Clock Path Skew:        -0.043ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    4.706ns = ( 24.706 - 20.000 )
    Source Clock Delay      (SCD):    5.076ns
    Clock Pessimism Removal (CPR):    0.326ns
  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
```

4. CONCLUSIONI

Attraverso la progettazione di questo componente abbiamo avuto modo di cimentarci in un nuovo tipo di programmazione, quello delle componenti logiche. Con l'utilizzo della logica a stati, abbiamo suddiviso il caso da risolvere in ulteriori sotto-problemi, in modo da renderne più semplice l'elaborazione. Infatti, ogni stato rappresenta uno step da svolgere per giungere alla risoluzione completa.

Il programma Vivado si è rivelato un ottimo software per la realizzazione del progetto, in quanto possiede funzionalità molto utili come la visualizzazione delle onde durante la Behavioral simulation. Infatti, per la prima volta, abbiamo potuto visualizzare in maniera quasi tangibile, il risultato ottenuto.

Sempre tramite il programma Vivado, per la verifica del progetto, sono state svolte molteplici simulazioni con differenti casistiche, dai corner-case alle più semplici. In tal modo, è stato possibile aver conferma del corretto funzionamento del progetto.