POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary

Progetto di Reti Logiche

Laurea Triennale in Computer Engineering - Ingegneria Informatica

Author: Matteo Romilio Pasqual

Academic year: 2023-2024

---

1. Introduzione

la Prova Finale di Reti Logiche consiste nell'implementazione di un modulo hardware descritto in VHDL che si interfacci con una memoria RAM e che esegua le seguenti istruzioni.
Il sistema deve leggere un messaggio costituito da una sequenza di K parole W, il cui valore è compreso tra 0 e 255. Il valore 0 all'interno della sequenza deve essere interpretato come "valore non specificato". La sequenza di K parole W da elaborare è memorizzata a partire da un indirizzo specificato (denominato ADD), ogni 2 byte (ad esempio, ADD, ADD+2, ADD+4, ..., ADD+2*(K-1)).
ESEMPIO: Sequenza di partenza: ADD=1024 K=16
0 0 **0** 0 **28** 0 **614** 0 **0** 0 **0** 0 **0** 0 **0** 0 **0** 0 **100** 0 **1** 0 **0** 0 **5** 0 **23** 0 **2** 0 **0** 0
Il byte mancante deve essere completato come descritto di seguito. Il modulo hardware deve completare la sequenza, sostituendo gli zeri con l'ultimo valore letto diverso da zero e inserendo un valore di "credibilità" C nel byte mancante per ogni valore della sequenza. La sostituzione degli zeri avviene copiando l'ultimo valore valido (non zero) letto precedentemente e appartenente alla sequenza. Il valore C associato ad ogni parola W viene memorizzato in memoria nel byte subito successivo (ad esempio, ADD+1 per W in ADD, ADD+3 per W in ADD+2, ...). Il valore C è sempre maggiore o uguale a 0 ed è reinizializzato a 31 ogni volta che si incontra un valore W diverso da zero. Quando C raggiunge il valore 0, non viene ulteriormente decrementato.
Il valore di credibilità C è pari a 31 ogni volta che il valore W della sequenza è non zero
ESEMPIO: in ADD=1024+4=1028
28 0 **614** 0 => **28** 31 **614** 31.
Il valore di credibilità C invece viene decrementato rispetto al valore precedente ogni volta che si incontra uno zero in W
ESEMPIO: in ADD=1024+6=1030
614 0 **0** 0 **0** 0 => **614** 31 **614** 30 **614** 29 .
Nota: Se il primo dato della sequenza è pari a zero, il suo valore rimane invariato e il valore di credibilità viene impostato a 0 (zero). Lo stesso vale fino al raggiungimento del primo dato della sequenza con un valore diverso da zero.
ESEMPIO: in ADD=1024
0 0 **0** 0 => **0** 0 **0** 0.

In accordo con il comportamento delineato, la sequenza conclusiva sarà caratterizzata dalla presenza delle informazioni di credibilità richieste.
ESEMPIO: Sequenza finale ADD=1024 K=16
0 0 **0** 0 **28** 31 **614** 31 **614** 30 **614** 29 **614** 28 **614** 27 **614** 26 **100** 31 **1** 31 **1** 30 **5** 31 **23** 31 **2** 31 **2** 30

1.1. Entity

L'entità del modulo da implementare è definita con tre ingressi primari: uno di 1 bit per il segnale START, uno di 16 bit per il segnale ADD e uno di 10 bit per il segnale K. Inoltre, il modulo dispone di un unico segnale di uscita primario di 1 bit, denominato DONE. Il modulo è sincrono rispetto al segnale di clock (CLK), che è unico per l'intero sistema, ed è interpretato sul fronte di salita del clock. L'unica eccezione è il segnale di reset (RESET), che è asincrono .

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.all;
4
5  --entity of project_reti_logiche
6  entity project_reti_logiche is
7      Port (
8          i_clk : in std_logic;       --in clock signal
9          i_rst : in std_logic;       --in rst signal
10         i_start : in std_logic;     --in start signal
11         i_add : in std_logic_vector(15 downto 0);   --in starting address
12         i_k   : in std_logic_vector(9 downto 0);    --in number of iteration
13
14         o_done : out std_logic;     --out done signal
15
16         o_mem_addr : out std_logic_vector(15 downto 0);     --set address in RAM
17         i_mem_data : in  std_logic_vector(7 downto 0);      --read data from RAM at address o_mem_addr
18         o_mem_data : out std_logic_vector(7 downto 0);      --write data in RAM at address o_mem_addr
19         o_mem_we   : out std_logic;     --RAM Write enable
20         o_mem_en   : out std_logic      --RAM memory enable
21     );
22 end project_reti_logiche;
23 --end entity of project_reti_logiche
```

Figure 1: Entity interface

1.2. Specifica dei segnali

Ogni segnale all'interno del sistema segue le seguenti regole:

- All'istante iniziale, corrispondente al reset del sistema, l'uscita DONE deve essere 0.
- Ogni volta che il segnale di RESET viene emesso (RESET = 1), il modulo viene reinizializzato.
- Quando RESET ritorna a zero, il modulo inizia l'elaborazione quando il segnale START in ingresso viene portato a 1 (Considerando che prima del primo START = 1 verrà sempre dato RESET = 1).
- Il segnale START rimane alto fino a quando il segnale DONE non viene portato alto;
- Al termine dell'elaborazione il modulo alza il segnale DONE a 1 per notificare la fine e il segnale rimane alto fino a quando il segnale START non viene riportato a 0 (Durante questo periodo, un nuovo segnale START non può essere emesso fintanto che DONE è alto).
- Quando il segnale di START viene impostato a 1 (e per tutto il periodo in cui rimane alto), i primi indirizzo e dimensione della sequenza da elaborare vengono impostati sugli ingressi ADD e K.
- Prima di alzare il segnale DONE, il modulo deve aggiornare la sequenza e i relativi valori di credibilità al valore opportuno, seguendo la descrizione generale del modulo.

2. Architettura

Una volta definite il comportamento desiderato, l'interfaccia dell'entity e la specifica dei segnali, si procede con la progettazione dell'architettura del componente hardware. Questo approccio si basa sulla costruzione di una macchina a stati (**FSM**) gestisce tutte le operazioni necessarie. Gli stati presenti

saranno: S_RST, S_ZERO, S_ZERO_READ, S_ZERO_CHOICE, S_READ_MEM, S_CHOICE, S_R2W_31, S_WRITE_MEM_CRED31, S_R2W_NUMPREC, S_WRITE_MEM_NUMPREC, S_R2W_CREDX, S_WRITE_MEM_CREDX, S_MOVE_ADD2, S_END. Ognuno di essi svolgerà una determinata attività e al fronte di salita del clock renderà effettive le modifiche passando poi allo stato successivo. L'unico stato con accesso asincrono è quello di reset.

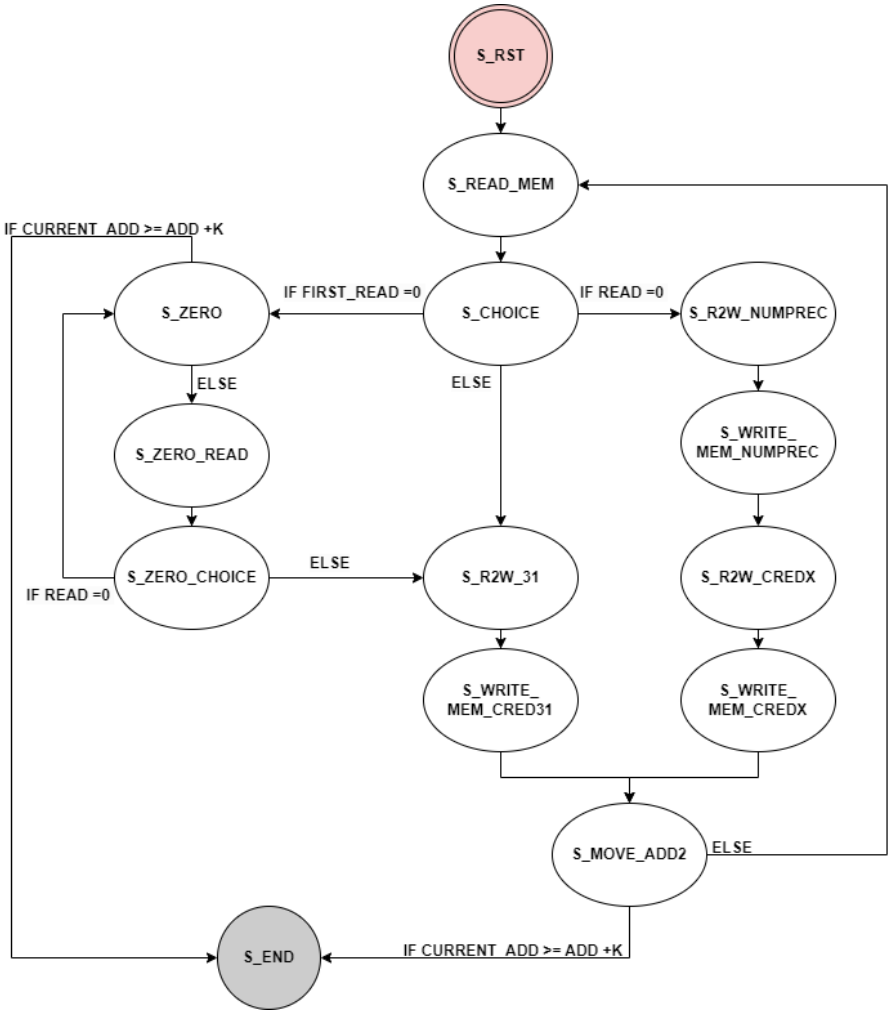

Figure 2: FSM

In VHDL il set degli stati verrà implementato attraverso un segnale current_state di tipo state_type e inizializzato di default allo stato di reset. State_type è un nuovo tipo di dato enumerativo contenente i nomi degli stati.

```
28    type state_type is (S_RST,      --reset state
29                        S_ZERO,S_ZERO_READ,S_ZERO_CHOICE,     --Starting 0
30                        S_READ_MEM,S_CHOICE,      --read state:read + choiche_on_read_data
31                        S_R2W_31,S_WRITE_MEM_CRED31,    --write 1 case state: ready_to_write + write31
32                        S_R2W_NUMPREC,S_WRITE_MEM_NUMPREC,  --write 2ndA case state: ready_to_write + write_current_word
33                        S_R2W_CREDX,S_WRITE_MEM_CREDX,      --write 2ndB case state: ready_to_write + write_current_credibility
34                        S_MOVE_ADD2,        --update address state + o_done = 1
35                        S_END        --end state: set o_done to 0 when i_start returns to 0
36                        );
37    signal current_state : state_type := S_RST;  --current state
```

Figure 3: State_type

Il progetto è stato sviluppato in un unico modulo contenente un solo processo con sensitivity list contenente i_cll e i_rst e per garantire il corretto funzionamento, è necessario definire delle variabili per memorizzare informazioni cruciali. Queste variabili includono:

- ‘current_credibility‘: memorizza il valore di credibilità corrente.
- ‘next_credibility‘: memorizza il valore di credibilità futuro.
- ‘current_word‘: memorizza l’ultima parola diversa da zero letta.
- ‘current_temp_add‘: memorizza l’indirizzo progressivo corrente.
- ‘next_temp_add‘: memorizza l’indirizzo successivo in cui ci si sposterà.

```
42        variable current_credibility : std_logic_vector (7 downto 0);  --current credibility
43        variable next_credibility : std_logic_vector (7 downto 0);     --next credibility = current credibility -1
44
45        variable current_word : std_logic_vector (7 downto 0);         --word read
46
47        variable current_temp_add : std_logic_vector(15 downto 0);     --current RAM address (from i_add to i_add + i_k)
48        variable next_temp_add : std_logic_vector(15 downto 0);        --next RAM address = current RAM address +2
```

Figure 4: Variables

Di seguito sarà descritto il comportamento di ciascuno stato, nonché gli stati successivi.

2.1. Reset state

È necessario distinguere tra segnale di reset e stato di reset. Durante l’esecuzione del programma, il segnale di reset viene innanzitutto emesso dal testbench. Il modulo reagisce portando a zero tutti i segnali di uscita, ponendo i valori delle variabili a zero 0 e infine posiziona il current_state nello stato di reset. Nello stato di reset (**S_RST**) vero e proprio, il modulo attende il segnale di inizio (i_start). Una volta ricevuto, avvia il processo di inizializzazione per la lettura in memoria e modifica il current_state assegnandolo allo stato di lettura.

```
52        if i_rst = '1' then
53        -- reset signal: all signals set at 0 and when i_start is setted to 1: (next_state <= S_R2R;)
54            o_done <= '0';
55            o_mem_en <= '0';
56            o_mem_we <= '0';
57            o_mem_addr <= (others => '0');
58            o_mem_data <= (others => '0');
59            current_credibility :=(others => '0');
60            current_word := (others => '0');
61            current_temp_add := (others => '0');
62            next_credibility := (others => '0');
63            next_temp_add := (others => '0');
64            current_state <= S_RST;
65
66        elsif rising_edge(i_clk) then
67            case current_state is
68
69                when S_RST =>
70                --RESET STATE: wait i_start then RAM memory Ready To Read
71                    if i_start = '1' then
72                        o_mem_addr <= i_add;
73                        current_temp_add := i_add;
74                        o_mem_en <= '1';
75                        o_mem_we <= '0';
76                        current_state <= S_READ_MEM;
77                    else current_state <= S_RST;
78                    end if;
```

Figure 5: Reset

2.2. Read and Choice state

Nello stato di lettura in memoria (**S_READ_MEM**), il modulo legge e porta gli enable della memoria a zero cosi da mantenere il dato per lo stato successivo infine modifica il current_state assegnandolo allo stato di decisione. Nello stato di decisione (**S_CHOICE**). il modulo decide in base alla parola letta cosa fare:

- se la parola è zero ed è la prima letta assegna il curren_state allo stato chiamato zero;

- se la parola è zero ma non è la prima letta assegna il current_state allo stato che scriverà la parola precedente diversa da zero e aggiorna il valore della credibilità corrente
- se la parola non è zero assegna il current_state allo stato che scriverà la credibilità 31, aggiorna credibilità corrente a 31 e credibilità successiva a 30

```
80              when S_READ_MEM =>
81              --RAM memory Read and freeze ram: (o_mem_en <= '0' + o_mem_we <= '0')
82                  o_mem_en <= '0';
83                  o_mem_we <= '0';
84                  current_state <= S_CHOICE;
85
86              when S_CHOICE =>
87              --RAM data available: make the choice
88                  if i_mem_data = "00000000" and current_temp_add = i_add then
89                  --sequence starts with 0
90                      current_state <= S_ZERO;
91                  elsif i_mem_data = "00000000" then
92                  --current word is 0
93                      current_state <= S_R2W_NUMPREC;
94                      current_credibility := next_credibility;
95                  else
96                  -- current word is a number
97                      current_state <= S_R2W_31;
98                      current_word := i_mem_data;
99                      next_credibility := "00011110";
100                     current_credibility := "00011111";
101                 end if;
```

Figure 6: Read and choice states

2.3. Write credibility 31 states

La scrittura della credibilità a 31 necessita di due stati:
Il primo (**S_R2W_31**) che inizializza la procedura alzando i due enable a uno, settando l'address prescelto per la scrittura e passando il current_state allo stato di scrittura effettivo.
Il secondo (**S_WRITE_MEM_CRED31**) invece scrive effettivamente la memoria, aggiorna il prossimo valore dell'address e passa il current_state allo stato di controllo dell'indirizzo.

```
103             when S_R2W_31 =>
104             --set memory read to write at ADD +1
105                 o_mem_addr <= std_logic_vector(SIGNED(current_temp_add) + 1);
106                 o_mem_en <= '1';
107                 o_mem_we <= '1';
108                 current_state <= S_WRITE_MEM_CRED31;
109
110             when S_WRITE_MEM_CRED31 =>
111             --write credibility = 31 and update next temp add
112                 o_mem_data <= "00011111";
113                 o_mem_en <= '1';
114                 o_mem_we <= '1';
115                 next_temp_add := std_logic_vector(SIGNED(current_temp_add) + 2);
116                 current_state <= S_MOVE_ADD2;
```

Figure 7: Write credibility 31

2.4. Write credibility X states

In questo set di stati prima si scrive la parola precedente poi si scrive la credibilità corrente X. La scrittura della parola necessita di due stati:
Il primo (**S_R2W_NUMPREC**) che inizializza la procedura alzando i due enable a uno, settando l'address prescelto per la scrittura e passando il current_state allo stato di scrittura effettivo.
Il secondo (**S_WRITE_MEM_NUMPREC**) invece scrive effettivamente la memoria e passa il current_state allo stato di controllo dell'indirizzo.
La scrittura della credibilità a X, come quella a 31, necessita di due stati:
Il primo (**S_R2W_CREDX**) che inizializza la procedura alzando i due enable a uno, settando l'address prescelto per la scrittura e passando il current_state allo stato di scrittura effettivo.
Il secondo (**S_WRITE_MEM_CREDX**) invece scrive effettivamente la memoria, aggiorna il prossimo valore dell'address e passa il current_state allo stato di controllo dell'indirizzo.

```
124             when S_WRITE_MEM_NUMPREC =>
125             --vrite the previous word
126                  o_mem_data <= current_word;
127                  o_mem_en <= '1';
128                  o_mem_we <= '1';
129                  next_temp_add := std_logic_vector(SIGNED(current_temp_add) + 2);
130                  current_state <= S_R2W_CREDX;
131
132             when S_R2W_CREDX =>
133             --set memory read to write at ADD +1
134                  o_mem_addr <= std_logic_vector(SIGNED(current_temp_add) + 1);
135                  o_mem_en <= '1';
136                  o_mem_we <= '1';
137                  current_state <= S_WRITE_MEM_CREDX;
138
139             when S_WRITE_MEM_CREDX =>
140             ----write credibility = 31 and update next temp add
141                  o_mem_data <= current_credibility;
142                  if current_credibility >= "00000001" then
143                     next_credibility := std_logic_vector(SIGNED(current_credibility) - 1);
144                  end if;
145                  o_mem_en <= '1';
146                  o_mem_we <= '1';
147                  current_state <= S_MOVE_ADD2;
```

Figure 8: Write credibility X

2.5. Zero states

Questa parte dell' FSM è formata da tre stati in loop e viene attivata solamente quando leggo uno zero all'inizio della sequenza

- **S_ZERO**: controlla se la sequenza è finita e assegna di conseguenza current_state allo stato di fine e o_done a 1 se la sequenza è effettivamente finita altrimenti prepara la memoria per una nuova lettura
- **S_ZERO_READ**: compie la nuova lettura
- **S_ZERO_CHOICE**: se il valore letto è ancora zero fa ripartire il loop altrimenti assegna di conseguenza current_state nello stato di scrittura 31.

2.6. Move ADD state

Nello stato di ADD (**S_MOVE_ADD2**) il modulo controlla se il vallore dell'current address è maggiore o uguale dell'address di partenza sommato al valore K.
In caso affermativo la sequenza è finita dunque assegna di conseguenza current_state allo stato di fine e o_done a 1. In caso negativo invece prepara la memoria per essere letta nuovamente e assegna di conseguenza current_state allo stato di lettura della memoria.

```
164                 when S_ZERO =>
165                 --Check if the sequence is finisshed
166                     if SIGNED(current_temp_add) >= 2*SIGNED(i_k) + SIGNED(i_add) -2  then
167                     --go to end state
168                         current_state <= S_END;
169                         o_done <= '1';
170                     else
171                     --ready to read a new value
172                         next_temp_add := std_logic_vector(SIGNED(current_temp_add) + 2);
173                         o_mem_addr <=next_temp_add;
174                         o_mem_en <= '1';
175                         o_mem_we <= '0';
176                         current_state <= S_ZERO_READ;
177                     end if;
178
179                 when S_ZERO_READ =>
180                 --read + freeze
181                     current_temp_add := next_temp_add;
182                     o_mem_en <= '0';
183                     o_mem_we <= '0';
184                     current_state <= S_ZERO_CHOICE;
185
186                 when S_ZERO_CHOICE =>
187                 --make the choice again
188                     if i_mem_data = "00000000" then
189                         current_state <= S_ZERO;
190                     else
191                         current_state <= S_R2W_31;
192                         current_word := i_mem_data;
193                         next_credibility := "00011110";
194                         current_credibility := "00011111";
195                    end if;
```

Figure 9: Zero states

```
149                  when S_MOVE_ADD2 =>
150                  --check if the sequence is finished ADD' >= ADD + K
151                      if SIGNED(next_temp_add) >= 2*SIGNED(i_k) + SIGNED(i_add)  then
152                      --set o_done to 1 and go to the end
153                         current_state <= S_END;
154                         o_mem_en <= '0';
155                         o_mem_we <= '0';
156                         o_done <= '1';
157                      else
158                      --restart with a new read
159                         current_temp_add := next_temp_add;
160                         o_mem_addr <=next_temp_add;
161                         o_mem_en <= '1';
162                         o_mem_we <= '0';
163                         current_state <= S_READ_MEM;
164                         o_done <= '0';
165                         end if;
```

Figure 10: Move ADD state

2.7. END state

Nello stato di END (**S_END**) l'analisi della sequenza è già terminata e il segnale di DONE è già stato portato a uno conformemente alle specifiche è richiesto di attendere finché il segnale di START non ritorna a uno. Una volta che ciò avviene, il segnale DONE deve essere abbassato a zero e il programma può considerarsi terminato. Inoltre il current_state è portato allo stato di reset per essere pronto ad

una nuova elaborazione
NB: il testbench termina solamente quando il segnale di done torna a zero.

```
167                 when S_END =>
168                 --wait that i_start coe back to 0 then reset o_done to 0
169                         if i_start <= '0' then
170                               o_done <= '0';
171                               current_state <= S_RST;
172                         end if;
```

Figure 11: END state

3. Risultati sperimentali

La sintesi e l'implementazione del modulo hardware sono state entrambi svolte con impostazioni di default (Vivado 2016) e con FPGA target xc7a200tsbg484-1 (Artix-7):

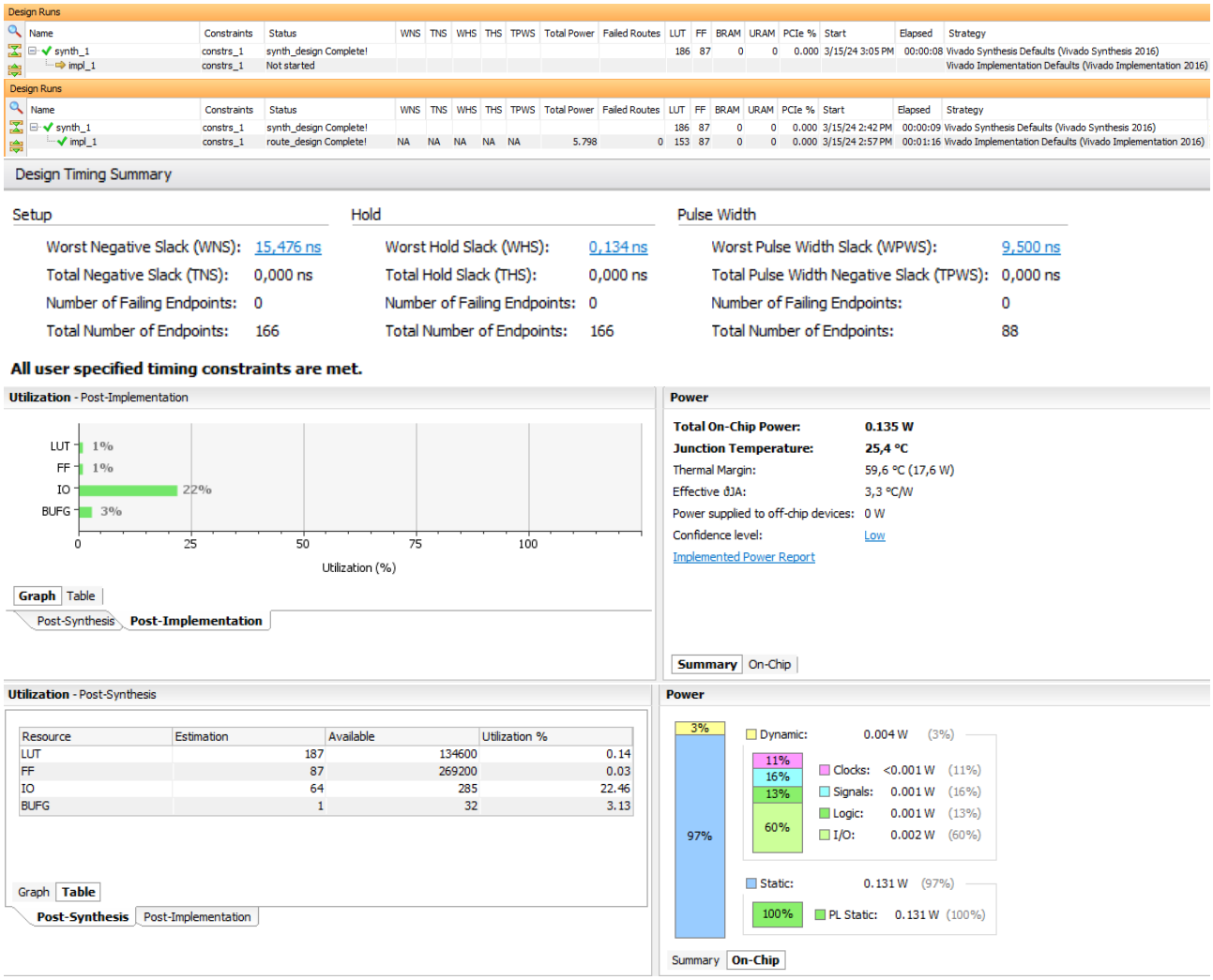

Figure 12: Synthesis report

Dai risultai di sintesi ed impleentazione si evince che:

- La sintesi è stata completata con successo entro un intervallo di tempo di 8 secondi, con l'utilizzo di 186 look-up table e 87 flip-flop.
- L'implementazione è stata completata con successo entro un intervallo di tempo di 1 minuto e 8 secondi, con l'utilizzo 153 look-up table e 87 flip-flop.

- Non ci sono failed route quindi tutti i collegamenti sono risultati efficaci.
- Non sono presenti LATCH.
- La percentuale di Look-Up Table, Flip Flop e BUFG risulta essere molto bassa, rispettivamente dello 0.11%, dello 0.03% e del 3.13%, conformemente alle specifiche del progetto.
- Lo slack time, ovvero il tempo rimanente ad ogni ciclo di clock dopo le elaborazioni, nel coso pessimo è di 15,47ns rispetto ad un clock di 20ns dunque ad ogni ciclo di clock rimane il 77% di tempo libero.
- Data la considerazione precedente saremmo in grado di dimezzare il clock a 10ns avanzando comunque del tempo per eventuali altre operazioni.
- La percentuale di IO al 22.46% è elevata, principalmente a causa delle dimensioni dei bus utilizzati. In particolare, la presenza di bus indirizzi da 15 bit contribuisce in modo significativo.
- Il consumo complessivo è ridotto, con particolare attenzione al consumo statico sotto 0.1W e il consumo dinamico praticamente assente (0.004W).

3.1. Post synthesis simulation

In questa sezione, verranno analizzati i testbench utilizzati per verificare il corretto funzionamento del modulo hardware progettato. I test verranno suddivisi in due sezioni: la prima analizzerà il corretto funzionamento con stringhe diverse, mentre la seconda controllerà le esecuzioni successive e il reset durante un'esecuzione. Tutti i test proposti superano con successo sia la behavioural simulation, sia la simulazione post-sintesi (timing e funzionale), che anche la simulazione post-implementazione (timing e funzionale).

TEST SU STRINGHE

TEST 1: DEFAULT
Il Test verifica le funzionalità di base del modulo. Test gia presente all'interno del testbench di default.
scenario_input = (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);
scenario_full = (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);
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
launch_simulation: Time (s): cpu = 00:00:01; elapsed = 00:00:10. Memory (MB): peak = 1713.980; gain = 0.000. $finish called at time : 2513 ns

TEST 2: CREDIBILITA' SEMPRE MAGGIORE DI 0
Il test verifica che la credibilità di una parola non possa mai essere minore di zero.
scenario_input = (1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
scenario_full = (1,31, 1,30, 1,29, 1,28, 1,27, 1,26, 1,25, 1,24, 1,23, 1,22, 1,21, 1,20, 1,19, 1,18, 1,17, 1,16, 1,15, 1,14, 1,13, 1,12, 1,11, 1,10, 1, 9, 1, 8, 1, 7, 1, 6, 1, 5, 1, 4, 1, 3, 1, 2, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0);
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
launch_simulation: Time (s): cpu = 00:00:25; elapsed = 00:00:31. Memory (MB): peak = 1713.980; gain = 909.480. $finish called at time : 7793 ns

TEST3: SEQUENZA INIZIALE DI 0
Il test verifica che se la sequenza iniziale di zero viene effettivamente ignorata.
scenario_input = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 5, 0, 23, 0, 200, 0, 0, 0);
scenario_full = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 31, 1, 30, 5, 31, 23, 31, 200, 31, 200, 30);
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
launch_simulation: Time (s): cpu = 00:00:18; elapsed = 00:00:34. Memory (MB): peak = 1687.879; gain = 895.016. $finish called at time : 2513 ns

TEST4: SEQUENZA DI 0
Il test verifica che se la sequenza è formata completamente da zero la ram non subisce variazioni.

scenario_input = (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);
scenario_full = (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);
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
launch_simulation: Time (s): cpu = 00:00:00; elapsed = 00:00:06. Memory (MB): peak = 1706.266; gain = 0.000. $finish called at time : 1670 ns

TEST SU SEGNALI

TEST 1: ESECUZIONI SUCCESSIVE
Il Test verifica la corretta esecuzione dell'analisi di due scenari consecutivi (il testbench è stato modificato per tenere conto di questa situazione). Come da specifica se un secondo start viene eseguito dopo che il done è tornato a zero il modulo inizierà una nuova esecuzione.
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)

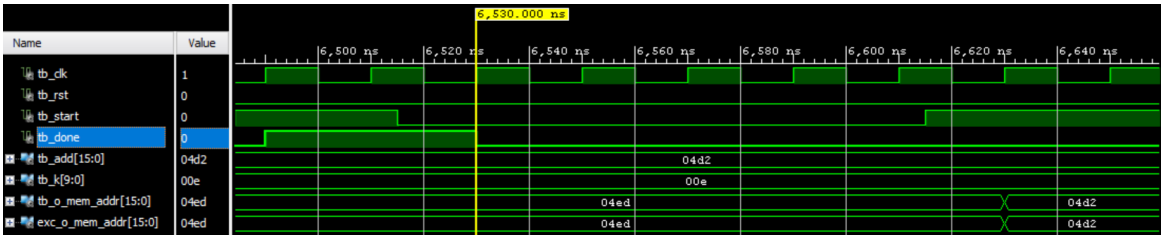

Figure 13: Esecuzioni successive

TEST 1: RESET INTERROMPENTE
Il Test verifica che un reset a metà esecuzione non comprometta la corretta uscita ma anzi faccia ripartire il procedimento da capo. Il risultato può variare nella parte di stringa già analizzata poichè gli zero sono già stati riscritti (il testbench è stato modificato per tenere conto di questa situazione).
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)

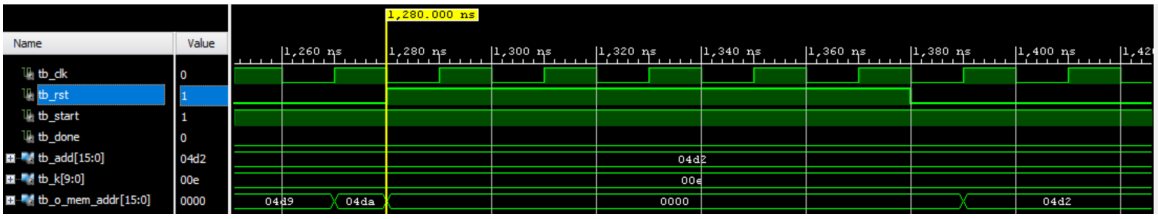

Figure 14: Reset interrompente

4. Conclusioni

La macchina a stati progettata è in grado di funzionare come da specifica manipolando stringhe e associando ad ogni valore una propria credibilità.
La corretta sintesi del modulo hardware sottolinea il successo complessivo delle varie fasi del processo di progettazione. In un secondo momento l'analisi dei dati ottenuti evidenzia un completamento adeguato impiegando risorse hardware in maniera ottimizzata. L'assenza di problematiche quali failed route e latch testimonia una corretta progettazione. La percentuale di utilizzo delle risorse hardware e il tempo di clock di 20ns risultano conformi alle linee guida del progetto.
Ulteriormente, i test condotti attraverso i testbench dimostrano la piena funzionalità del modulo hardware in diverse situazioni, confermando l'aderenza alle specifiche di progetto. Le condizioni particolari sono tutte state testate positivamente.
In sintesi, il modulo hardware progettato e sintetizzato emerge come una soluzione robusta, affidabile ed efficiente, capace di soddisfare appieno le esigenze del progetto.