UNIVERSITÀ DEGLI STUDI DI SALERNO

Esame di Tecnologie Digitali
a.a. 2017-2018

Gruppo 40 - Traccia 6

Aliperta, Martusciello, Valitutto, Ventre

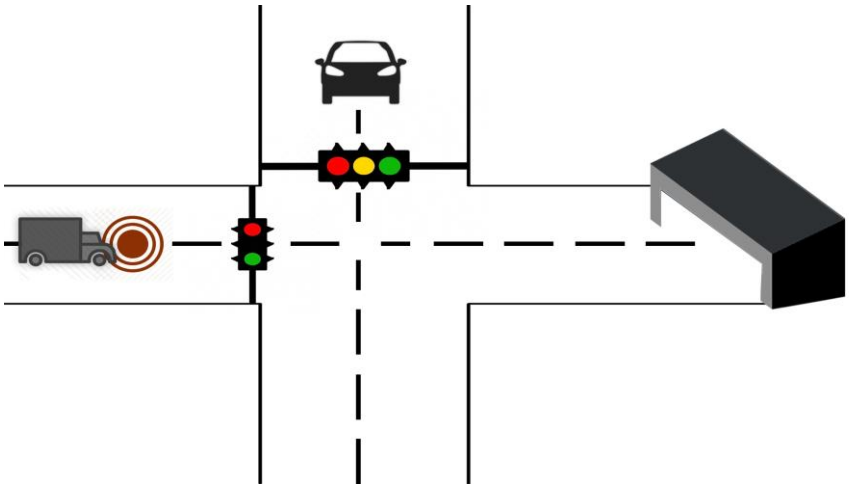
Introduzione

E' stata richiesta, per sostenere l'esame di Tecnologie Digitali, la relazione di un progetto attraverso lo svolgimento di una traccia assegnata.

La traccia, prevede la realizzazione di una macchina per il controllo di un semaforo (di seguito, controller). Il controller del semaforo è utilizzato per l'intersezione tra una strada principale veloce, ed una secondaria, che porta ad un deposito di tir dedicati a trasporti speciali a cui dare priorità.
E' presente un sensore sulla strada secondaria per rilevare la presenza di un veicolo. Se è così, il semaforo sulla via principale diventa GIALLO, quindi ROSSO in modo che i veicoli dalla strada secondaria possano attraversare la strada principale. Altrimenti, il semaforo sulla strada principale è sempre VERDE e il semaforo sulla strada secondaria è sempre ROSSO.
Il periodo di tempo è di 3 unità di tempo per la luce GIALLA e di 10 unità di tempo per la luce ROSSA. Si definisca una unità di tempo come un multiplo di cicli di clock (ossia, 5, 10, …).

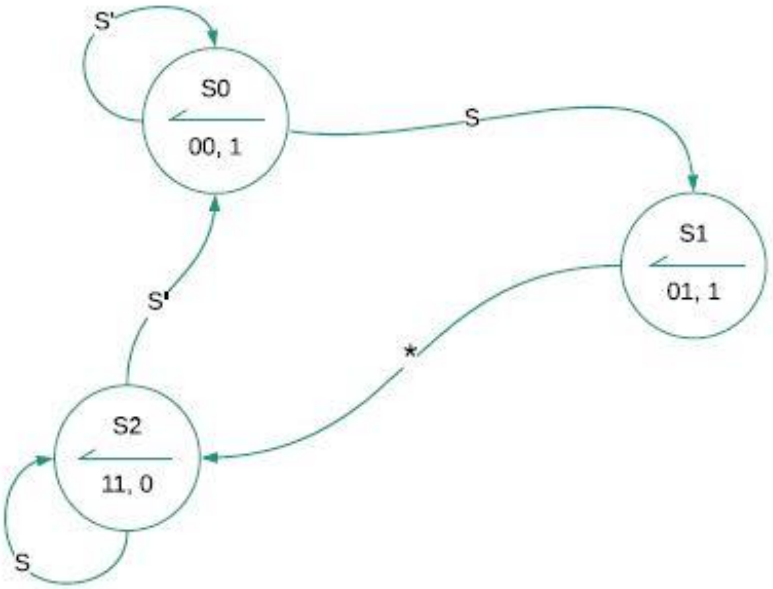
Tutti gli aspetti progettuali che non sono chiaramente descritti nelle proposte sono stati scelti liberamente dagli studenti che li hanno interpretati compatibilmente con il compito assegnato.

Caso Reale



Le due strade risultano intersecate perpendicolarmente e il TIR deve attraversare tale incrocio per giungere al deposito.
Alla strada principale è associata un semaforo (identificato nel controller dal vettore **sm1**) caratterizzato da 3 uscite: GREEN, YELLOW, RED.
Alla strada secondaria, invece il semaforo (identificato nel controller dal STD_LOGIC **sm2**), avrà le seguenti uscite: GREEN, RED.

Automa a stati finiti



Codifica Automa

Simbolo	Valore	Rappresenta	Descrizione
S0	00	Stato 0	Semaforo 1: Verde, Semaforo 2 Rosso
S1	01	Stato 1	Semaforo 1: Giallo, Semaforo 2 Rosso
S2	11	Stato 2	Semaforo 1: Rosso, Semaforo 2 Verde
S	1	Sensore	Acceso
S'	0	Sensore	Spento
*	1/0	Don't Care	Sia 1 che 0

Tabella di Flusso

	S	S'
S0	S1	S0
S1	S2	S2
S2	S2	S0

	1	0
00	01	00
01	11	11
11	11	00

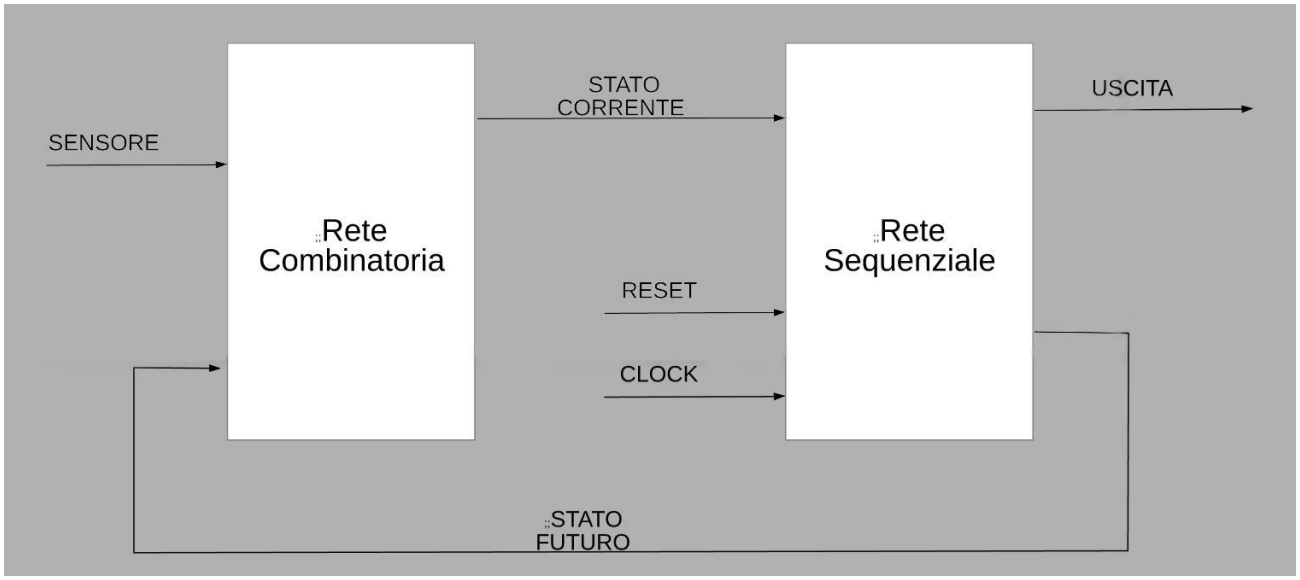
Comportamento

- Nello stato iniziale, S0, il semaforo principale (**sm1**) è posizionato sul verde*, "GREEN",* mentre il secondario (**sm2**) sul rosso, "*RED*".
- Ipotizziamo di installare sul manto della strada secondaria, un **sensore a pressione** che:
    - Garantisce la priorità alla strada secondaria.
    - Si attiva se e soltanto se si esercita su di esso una massa equivalente a quella di un TIR.
- Non appena il sensore (identificato nel controller come **sensor**) rileva la presenza di un TIR sulla strada secondaria:
    - Il semaforo principale (**sm1**) passa allo stato S1 "*YELLOW"*.
    - Dopo 3 unità di tempo passa allo stato S2 "*RED*" mentre il semaforo secondario (**sm2**) diventa verde.
- Passate altre 10 unità di tempo, possono verificarsi due casi:
    - Il sensore non rileva più nessun TIR e quindi si ritorna allo stato S0.
    - Il sensore ha rilevato, prima dello scadere delle 10 unità altri TIR passare, quindi prolunga il periodo di tempo di altre 10 unità e permane nello stato S2.

Specifiche

1. Il semaforo secondario (**sm2**) è caratterizzato solo e soltanto da 2 stati:
    - Verde: Passaggio
    - Rosso: Stop
2. L'unità di tempo scelta è di 5 fronti di salita (*Rising Edge*).
3. Il sensore è posto a una distanza tale da consentire il passaggio dei TIR all'arrivo di questi ultimi al semaforo.
4. Il sensore garantisce la priorità solo e soltanto ai TIR.
5. Il sensore garantisce che il semaforo principale sia sempre rosso finché tutti i TIR non hanno attraversato l'incrocio.
6. Le strade prese in considerazione, sono a senso unico di circolazione.

SCHEMA A BLOCCHI



XILINX

Entity

Per rispettare le specifiche della traccia assegnata, abbiamo ipotizzato di realizzare una *entity* basato sui seguenti segnali di **ingresso-uscita**:

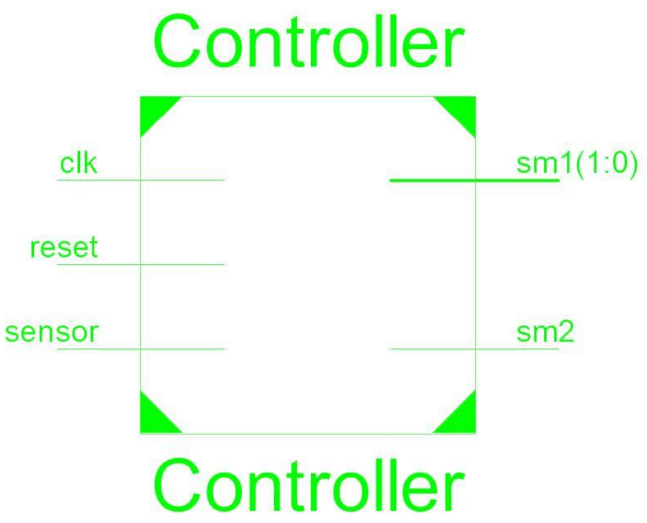
Input:

- Reset
  - 1: Porta il controller allo “stato iniziale”.
- Clock
  - Segnale di clock esterno
- Sensor
  - 1: Indica la presenza di un TIR sulla strada secondaria.
  - 0: Indica l’assenza di TIR sulla strada secondaria.

Output:

- sm1

    - Uscita vettoriale di 3 elementi codificata su due bit che identifica gli stati del semaforo della strada principale.
- sm2
    - Uscita che identifica gli stati del semaforo della strada secondaria.



Implementazione in linguaggio VHDL

Una possibile implementazione, in linguaggio VHDL, per la realizzazione del controller dei due semafori può essere la seguente:

Definizione dell'entità del controller:

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6  entity Controller is
7      Port ( clk, reset, sensor : in  STD_LOGIC;          -- Ingressi Controller
8             sm1 : out  STD_LOGIC_VECTOR (1 downto 0);    -- Output Semaforo strada principale
9             sm2 : out  STD_LOGIC);                       -- Outuput Semaforo strada secondaria
10 end Controller;
```

Abbiamo definito 3 ingressi di tipo ***STD_LOGIC*** che possono assumere solo valore 0 o 1 così come definito dalla logica binaria.
Le uscite sono due: un vettore di tipo ***STD_LOGIC*** che gestisce i segnali visivi del semaforo sulla strada principale, e uno di tipo ***STD_LOGIC*** anch'esso che gestisce i segnali visivi del semaforo sulla strada secondaria.

Definizione degli segnali interni dell’automa:

```
12 architecture Behavioral of Controller is
13
14   type state is (s0,s1,s2);                  -- Definiamo 3 stati dell'automa
15   signal current_state, next_state : state;  -- Variabili di tipo "state"
16
17   signal event_up : std_logic;               -- Segnale: se a '1' segnala la fine del "count"
18   constant UNIT_TIME: integer := 5;          -- Unità di tempo scelta come multiplo di cicli di clock
19   signal count, next_count : integer range 1 to 10*UNIT_TIME; -- Segnali di conteggio del periodo di tempo
20
21
22   begin
```

Subito prima del ***begin***, punto di partenza delle funzioni definite per il nostro *controller*, abbiamo definito dei segnali e una costante interna all’automa. Questi segnali infatti non vengono forniti dall’esterno, ma servono per la definizione e per l’evoluzione dell’automa.

Il tipo “***state***” è un tipo definito da noi che può assumere come valori i stati da noi definiti.
I segnali ***current_state*** e ***next_state***, sono due segnali di tipo “***state***” e servono per sapere in quale stato è il nostro automa e in quale stato andrà e deve andare.

Il segnale ***event_up*** è un flag che ci permette di sapere quando un particolare evento si è verificato.

La costante ***UNIT_TIME*** di tipo integer, e con valore 5, identifica l’unità di tempo scelta per il conteggio, necessaria al timer.

Definizione del processo del Clock:

```
23
24      synchronous : process(clk,reset)      -- Processo del clock
25      begin
26
27          if(reset = '1') then              -- Reset: se a '1' inizializza l'automa allo stato iniziale
28            current_state <= s0;            -- Impone lo stato iniziale S0
29            count <= 1;                     -- Inizializza il conteggio del timer
30          elsif(rising_edge(clk)) then      -- Commutazione sul fronte di salita del clock
31            count <= next_count;            -- Imposta il timer con il valore di next_count
32            current_state <= next_state;    -- Cambia lo stato attuale con quello futuro
33          end if;
34
35      end process;
```

Il processo *synchronous* permette l'aggiornamento continuo dello stato corrente e del conteggio del timer.
Il sistema risulta sincronizzato rispetto al fronte di salita, infatti, ogni rising edge, il sistema aggiorna stato e conteggio oppure, con il ***reset*** a valore 1, viene riportato allo stato iniziale; così come il segnale di incremento del timer.

Definizione del processo della Transizione degli Stati:

```
38      state_transiton : process
39      (current_state, sensor, event_up) -- Processo delle transizioni degli stati
40      begin
41
42        case (current_state) is
43
44          when s0 =>
45              if (sensor = '1') then
46                next_state <= s1; -- Il prossimo stato sarà s1
47              else
48                next_state <= current_state; -- Permane in S0
49              end if;
50
51          when s1 =>
52              if (event_up = '1') then
53                next_state <= s2; -- Il prossimo stato sarà s2
54              else
55                next_state <= current_state; -- Permane in S1
56              end if;
57
58          when s2 =>
59              if (event_up = '1' and sensor = '0') then
60                next_state <= s0; -- Il prossimo stato sarà s0
61              else
62                next_state <= current_state; -- Permane in S2
63              end if;
64
65        end case;
66      end process;
```

Lo ***state_transition*** process permette l'aggiornamento dello stato così come definito nella sezione Comportamento .

Definizione del Processo di output:

```
85      output : process(current_state) -- Processo di Output,
86      begin                           -- si attiva soltantanto se varia lo stato corrente
87
88        case (current_state) is
89
90          when s0 =>        -- Green State
91             sm1 <= "00";
92             sm2 <= '1';
93
94          when s1 =>        -- Yellow State
95             sm1  <= "01";
96             sm2 <= '1';
97
98          when s2 =>        -- Red State
99             sm1  <= "11";
100            sm2 <= '0';
101
102       end case;
103
104     end process;
```

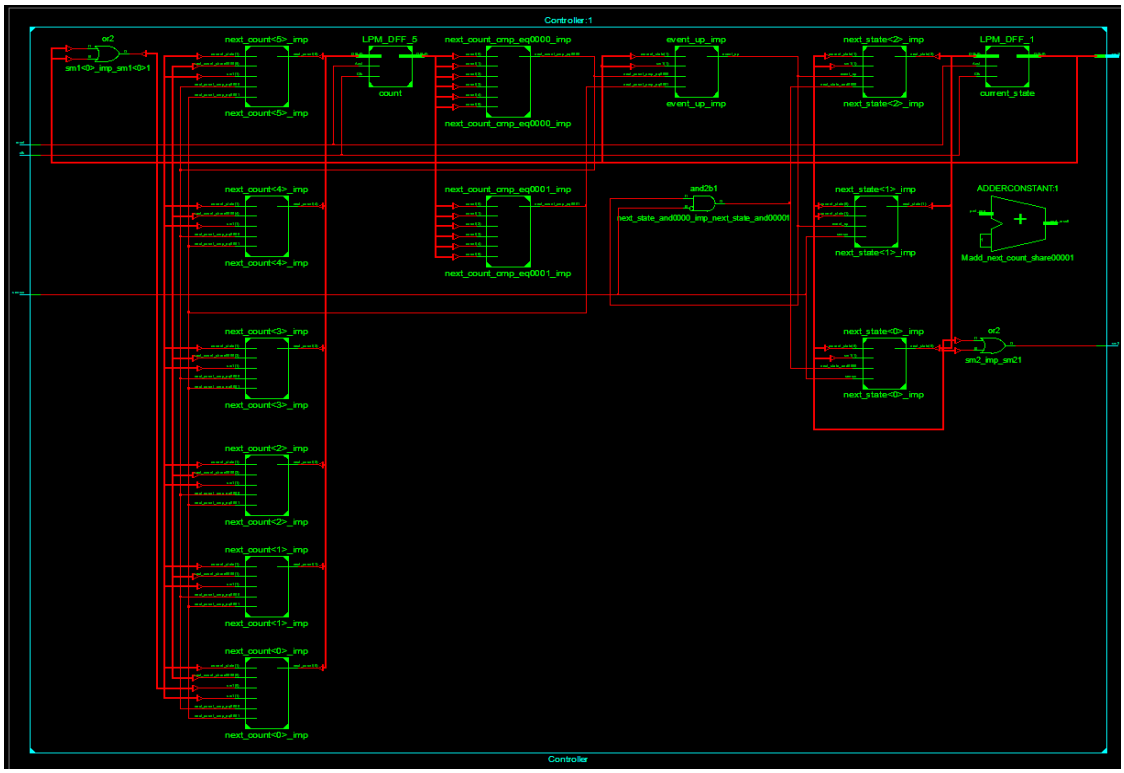
Gli output delle uscite saranno consequenziali alle scelte fatte in precedenza. Ovvero, fintantoché il semaforo sulla strada principale sia impegnato con il verde e con il conteggio del giallo, quello sulla secondaria, mostrerà al TIR, il rosso. Solamente quando ci sarà il conteggio del rosso sulla strada principale, al TIR verrà data la possibilità di attraversare l'incrocio, avendo sul suo semaforo il verde.

Definizione del Processo relativo al timer:

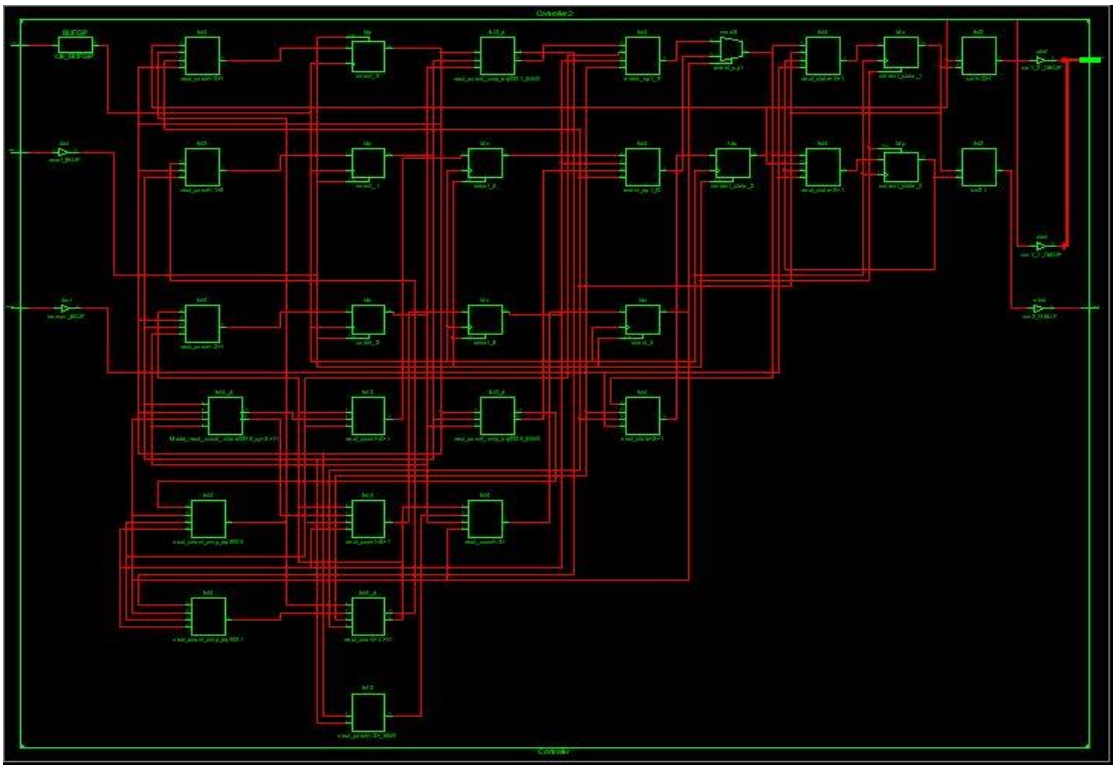
```
89      timer : process(current_state , count)  -- Processo relativo al Timer
90      begin
91
92      case current_state is
93
94        when s1 =>
95              if (count = 3*UNIT_TIME) then   -- In tale stato il periodo di tempo è di 3 unità
96                 next_count <= 1; -- Resetta il conteggio
97                 event_up <= '1'; -- Segnala la fine del tempo per la visualizzazione del giallo sul semaforo principale
98              else
99                 next_count<= count+1; -- Incrementa next_count che verrà assegnato a count
100                event_up <= '0'; -- Segnala che ancora nessun evento si è verificato
101             end if;
102
103          when s2 =>
104             if (count = 10*UNIT_TIME) then  -- In tale stato il periodo di tempo è di 10 unità
105                next_count <= 1; -- Resetta il conteggio
106                event_up <= '1'; -- Segnala la fine del tempo per la visualizzazione del rosso sul semaforo principale
107             else
108                next_count <= count+1; -- Incrementa next_count che verrà assegnato a count
109                event_up <= '0'; -- Segnala che ancora nessun evento si è verificato
110             end if;
111
112          when others => -- In tale stato viene azzerato il contatore
113             next_count <= 1; -- Resetta il conteggio
114             event_up <= '0'; -- Segnala che ancora nessun evento si è verificato
115
116       end case;
117
118     end process;
```

Tale processo risulterà impegnato nel conteggio delle unità di tempo per la luce *GIALLA* e la luce *ROSSA*, rispettando le specifiche assegnate.
Quando il conteggio risulterà terminato il segnale *event_up* sarà attivato.

RTL Schematic



Technology Schematic



Risultati

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	9	1,536	1%	
Number of 4 input LUTs	20	1,536	1%	
Number of occupied Slices	11	768	1%	
Number of Slices containing only related logic	11	11	100%	
Number of Slices containing unrelated logic	0	11	0%	
Total Number of 4 input LUTs	20	1,536	1%	
Number of bonded IOBs	6	63	9%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	3.95			

```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 6.401ns (Maximum Frequency: 156.226MHz)
   Minimum input arrival time before clock: 2.821ns
   Maximum output required time after clock: 9.121ns
   Maximum combinational path delay: No path found
```

Simulatore ISIM

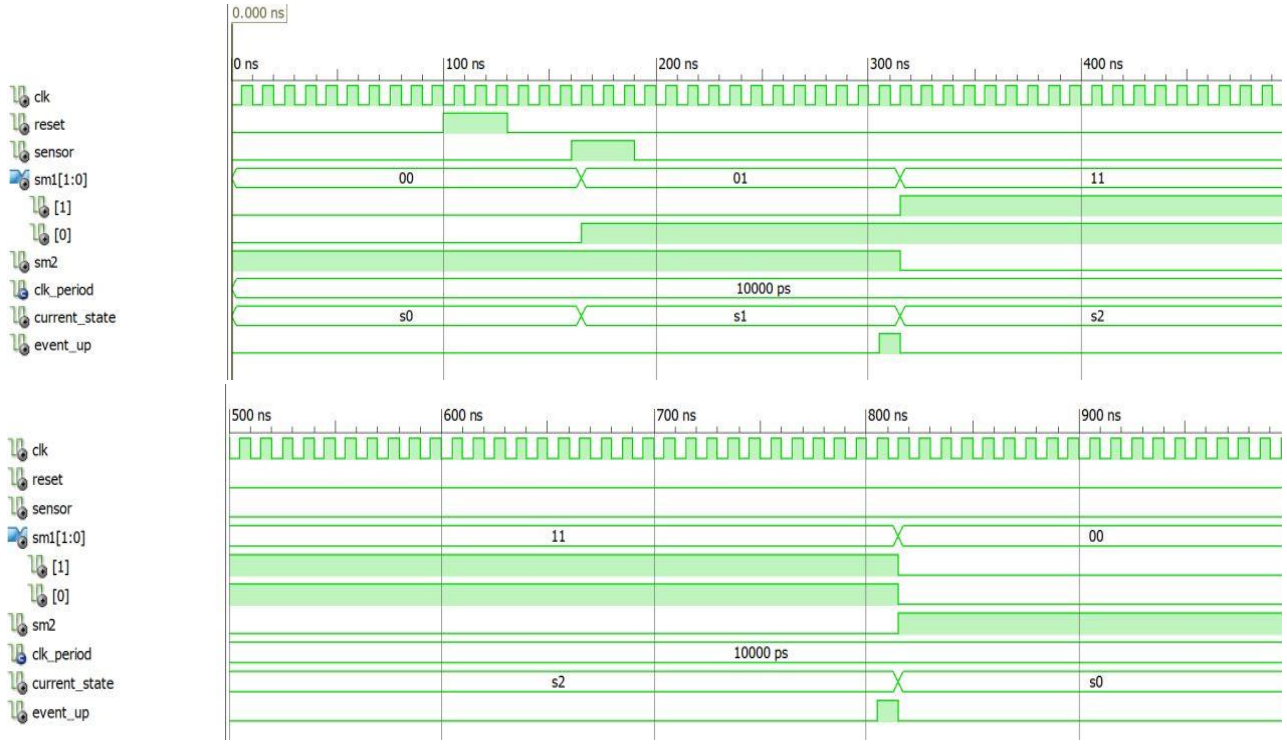
Per verificare che l'implementazione nel codice VHDL sia corretta, è stata eseguita una prova al simulatore, dove sono stati inseriti dei valori in ingresso, congruenti a quelli rilasciati dal report, in modo tale da essere sicuri che il sistema reagisca nel modo corretto.

Codice VHDL Test Bench

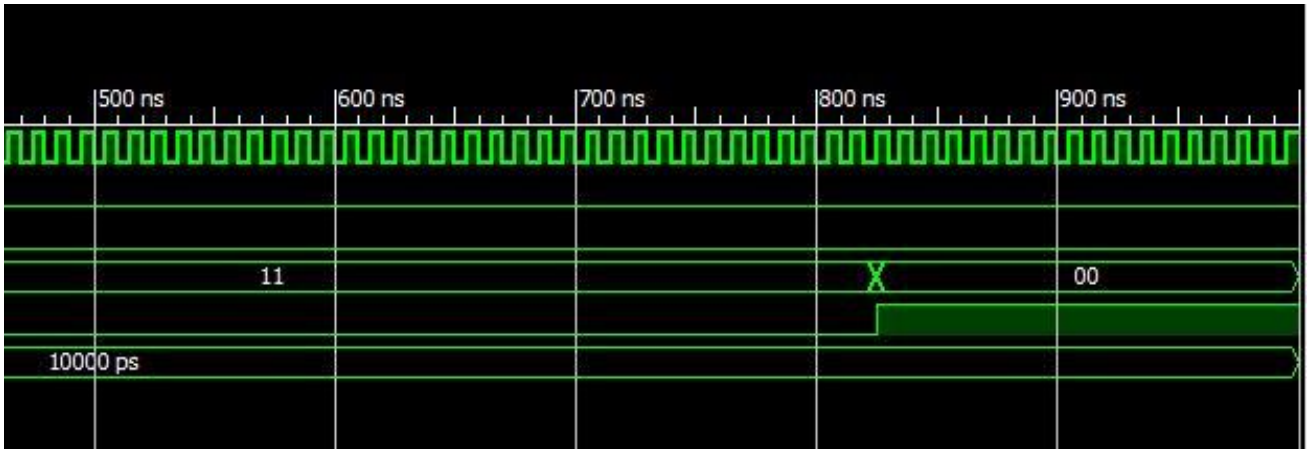
```
86      -- Stimulus process
87      stim_proc: process
88      begin
89         -- hold reset state for 100 ns.
90         wait for 100 ns;
91         -- insert stimulus here
92
93         reset <= '1';
94         wait for clk_period*3;
95
96         reset <= '0';
97         wait for clk_period*3;
98
99         sensor <= '1';
100        wait for clk_period*3;
101
102        sensor <= '0';
103
104        wait;
105     end process;
```

Andamento Temporale:

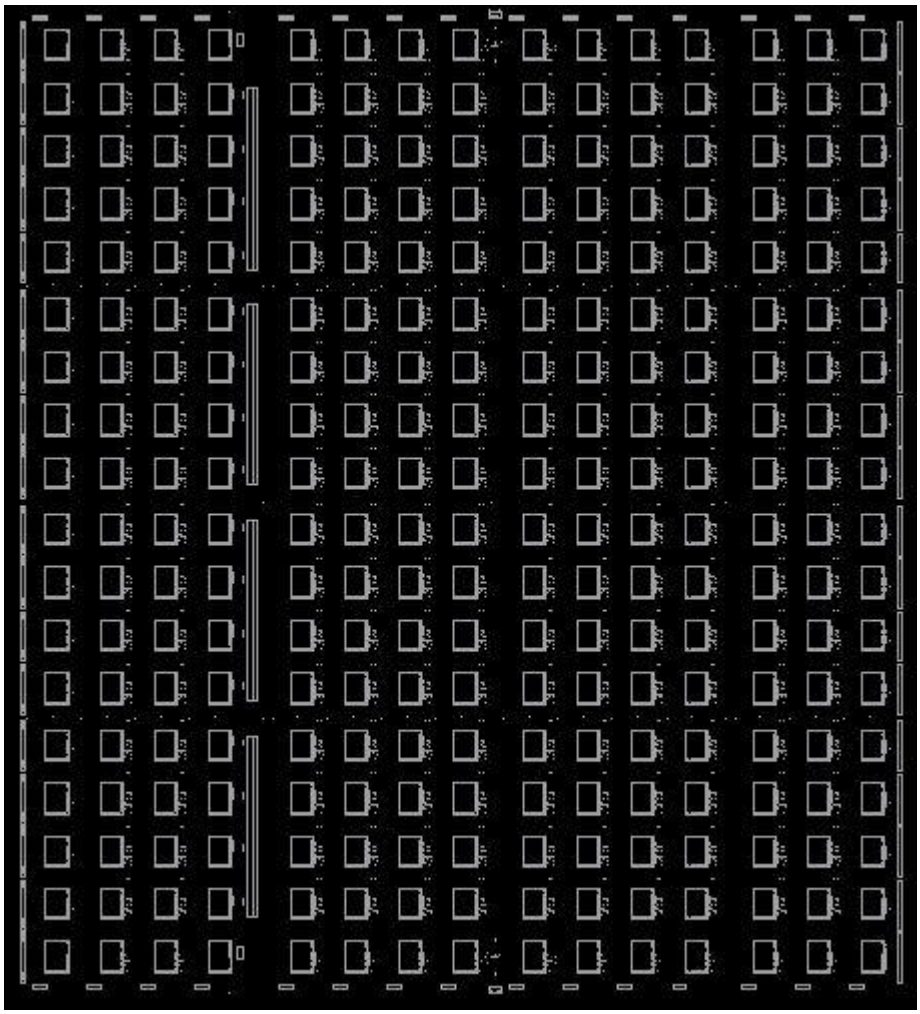
Behavioral Model Simulation :



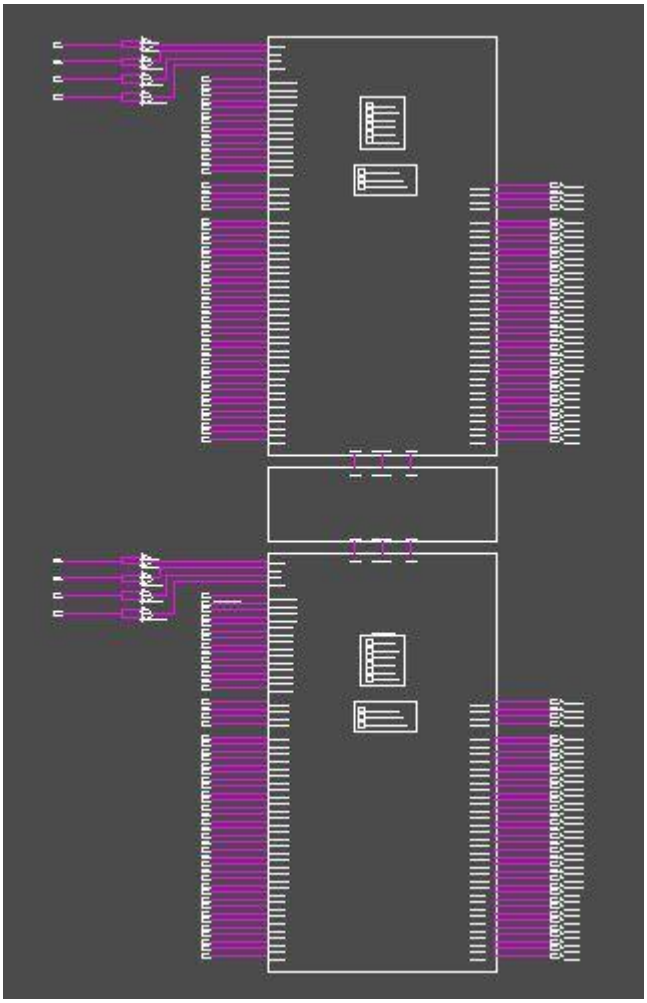
Post – Place & Route Model Simulation :



FPGA Placed Design:



FPGA Routed Design:



Conclusioni

Per concludere, tale relazione ha presentato una delle possibili soluzioni per il problema assegnatoci. Per tutti gli aspetti non chiaramente descritti, sono state fatte delle scelte in modo da rispettare le specifiche assegnate e in maniera compatibile con il progetto.