UNIVERSITÀ DI PISA

PROGETTAZIONE DI SISTEMI MICROELETTRONICI

TITOLO PROGETTO:

RICONOSCITORE DI SEQUENZA SICURA IN VHDL CON XILINX VIVADO

STUDENTE:

ANTONIO DI VITO

PROFESSORE:

LUCA FANUCCI

ANNO:

2018/2019

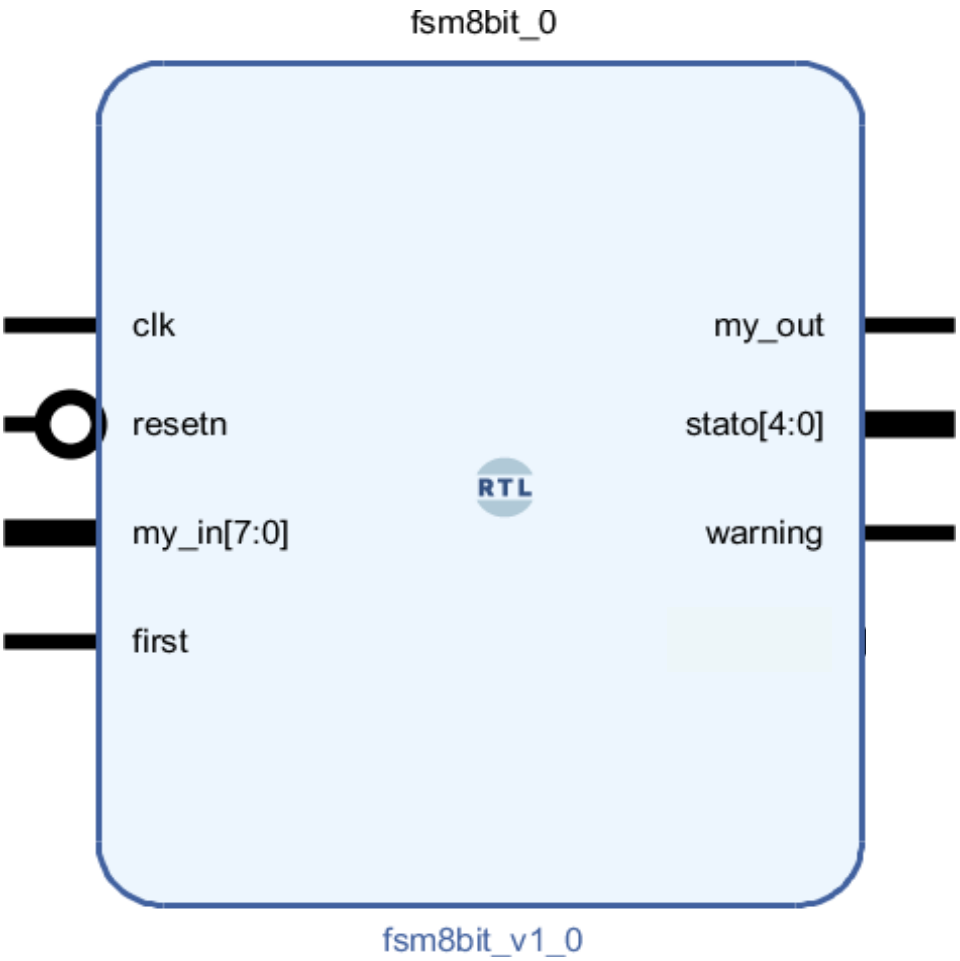
INTRODUZIONE

Il riconoscitore di sequenze è una macchina digitale che permette di verificare la corretta sequenza di un determinato numero di bit in ingresso restituendo in uscita un bit di controllo.

Questi riconoscitori posso essere implementati per vari scopi, tra i quali, in ambito di sicurezza, l'utilizzo di un codice digitale per aprire porte, casseforti e relativi sistemi di allarme che emettono un suono nel caso venga inserito un codice errato.

Vedremo in questa relazione come implementare questa struttura utilizzando il codice VHDL e i risultati della sintesi logica utilizzando la piattaforma Xilinx FPGA Zync.

Il sintetizzatore prodotto è il seguente:



Ora verranno descritte esplicitamente le porte utilizzate.

PORTE DI INGRESSO:

-***clk***: segnale di clock;

-***resetn***: ripristina tutti i registri e azzera i segnali interni della struttura;

-***my_in***: numero a 8-bit in ingresso;

-***first***: bit che indica quando iniziare il controllo della sequenza

PORTE DI USCITA:

-***my_out*** : bit che indica se la sequenza desiderata è stata riconosciuta con successo;

-***warning***: bit che controlla due principali situazioni

1) Va alto quando nella stessa sequenza da riconoscere vengono letti due segnali alti di first;
2) Quando risultano tre sequenze consecutive errate rimane fisso alto interrompendo il riconoscimento, ritorna basso quando si attiva il segnale di reset.

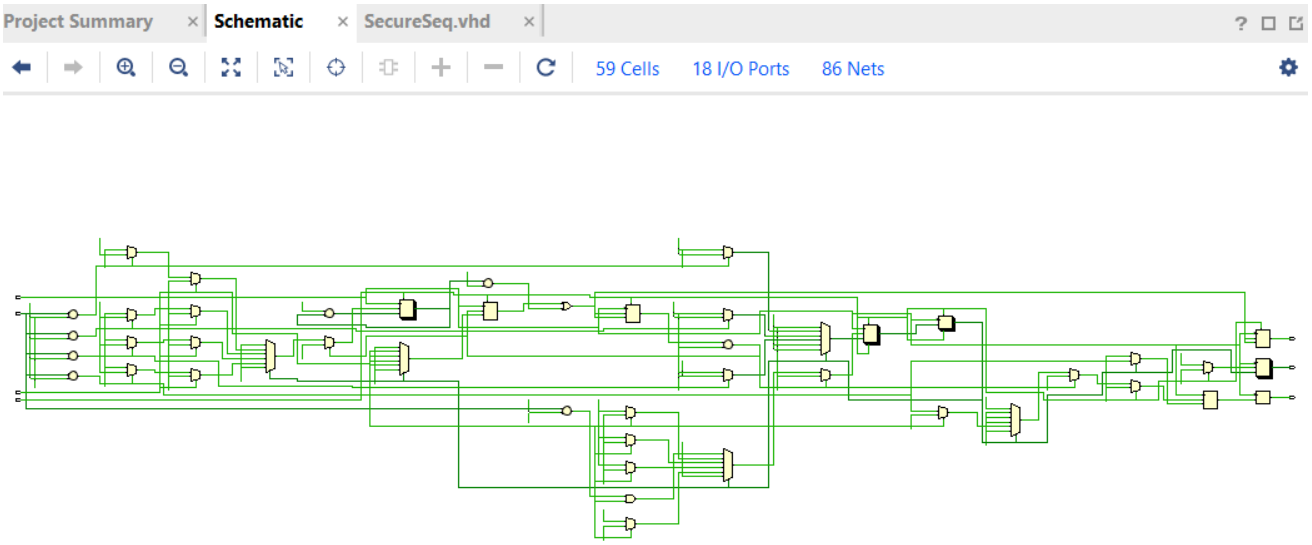
PORTE AGGIUNTIVE:

Al fine di una migliore implementazione e per favorire il debugging di eventuali problemi in caso di eventuale arresto improvviso, sia per il progettista che per l'utente stesso, è stata inserita una porta in uscita:

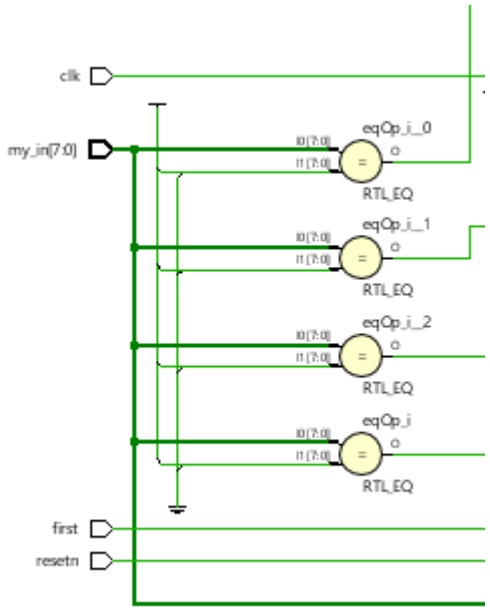
-***stato***: 5 bit che verificano lo stato della macchina a stati finiti.

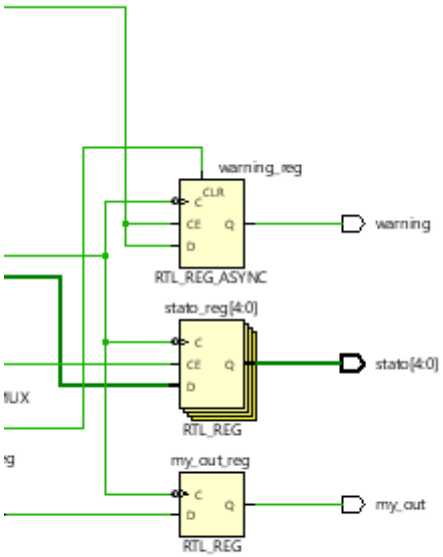
SCHEMA A BLOCCHI

Effettuando un’analisi RTL della struttura, possiamo ricavarne il diagramma a blocchi ed il risultato èmostrato nel seguente:



Si evidenziano quindi ingressi e uscite:





CODICE VHDL

Nel nostro caso il dispositivo deve riconoscere una sequenza di 5 valori ovvero 36, 19, 56, 101, 73. Avremo dunque la dichiarazione della struttura dove si introducono i vari segnali di ingresso e di uscita. All'interno dell'architettura sono stati dichiarati:
- gli stati come costanti necessari per lo sviluppo della macchina a stati finiti;
- i segnali utili per il corretto funzionamento del warning, out e per il conteggio dei cicli.

Avremo poi il blocco della macchina a stati finiti la quale modificherà i segnali interni di uscita e di warning a seconda se la sequenza di numeri in ingresso è quella desiderata (*next_state_logic*).
Infine, sono stati introdotti i blocchi rispettivamente di aggiornamento stato (*state_update*), riconoscimento output (*output_logic_p*) e warning (*warning_check*).

DICHIARAZIONE ENTITY:

```
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use ieee.STD_LOGIC_UNSIGNED.all;
6  use IEEE.NUMERIC_STD.all;
7
8  entity fsm8bit is
9      port (
10         clk     :in std_logic;
11         resetn  :in std_logic;
12         my_in   :in std_logic_vector (7 downto 0);   --seq ingresso
13         first   :in std_logic;                       --piedino di avvio riconoscimento
14         my_out  :out std_logic;                      --bit a 1 se ha riconosciuto
15         stato   :out std_logic_vector (4 downto 0);  --per visualizzare lo stato corrente, NON RICHIESTO NELLA TRACCIA MA MOLTO UTILE
16         warning :out std_logic--;                    --piedino warning
17 --      cycle   :out std_logic_vector (2 downto 0)   --per visualizzare l'inizio del ciclo, NON RICHIESTO MA MOLTO UTILE
18     );
19     end fsm8bit;
```

MACCHINA A STATI FINITI PER IL RICONOSCIMENTO DELLA SEQUENZA

```
architecture five_proc of fsm8bit is

    --per visualizzare lo stato attuale della macchina, hot coding
    constant stato1 : std_logic_vector(4 downto 0)     := "00001";    --iniziale
    constant stato2 : std_logic_vector(4 downto 0)     := "00010";
    constant stato3 : std_logic_vector(4 downto 0)     := "00100";
    constant stato4 : std_logic_vector(4 downto 0)     := "01000";
    constant stato5 : std_logic_vector(4 downto 0)     := "10000";    --finale

    signal recognized: boolean;                                       --collegato al pin di UNLOCK, cioè myout, nella traccia
    signal temp_loop: std_logic_vector(4 downto 0)     := "00001";   --per tenere traccia degli stati
    signal temp: std_logic_vector(4 downto 0)          := "00001";   --per tenere traccia degli stati

--  signal cycle_count: std_logic_vector(2 downto 0)  := "000";     --conta per 5 colpi di clock

    signal warning_loop: std_logic := '0';                            --warning dato da 2 first
    signal warning_count:std_logic_vector(1 downto 0) := "00";       --fino a 3 warning consecutivi, poi si blocca il programma (warning perenne a 1, quindi da reset
--    signal warning_imp: std_logic_vector(1 downto 0) := "00";      --warning impulsivo (2 bit per utilità)
    signal warning_temp: std_logic := '0';                            --buffer per l'out warning
-------------------------------------------------

begin

    next_state_logic_p : process (clk, resetn, temp, temp_loop, my_in, first, warning_temp)    --cycle_count
      begin
       if resetn = '1' then
           temp_loop <= stato1;
           warning_loop <= '0';
           warning_count <= "00";
       elsif resetn = '0' and warning_temp = '0' and rising_edge(clk) then
            warning_loop <= '0';
            case temp is
                when stato1 =>
                    if (my_in = "00100100" and first = '1') then     --deve avvenire a queste condizioni
                        temp_loop <= stato2;
                    end if;
                    recognized <= false;
                when stato2 =>
                    if first = '0' then
                       if (my_in = "00010011") then
                           temp_loop <= stato3;
                       else
                           temp_loop <=stato1;
                           warning_count <= warning_count +'1';
                       end if;
                   else
                    warning_loop <= '1';
                   end if;
                when stato3 =>
                    if first = '0' then
                       if (my_in = "00111000") then
                           temp_loop <= stato4;
                       else
                           temp_loop <= stato1;
                           warning_count <= warning_count +'1';
                       end if;
                    else
                       warning_loop <= '1';
                    end if;
                when stato4 =>
                    if first = '0' then
                       if (my_in = "01100101" ) then
                           temp_loop <= stato5;
                       else
                           temp_loop <= stato1;
                           warning_count <= warning_count +'1';
                       end if;
                    else
                       warning_loop <= '1';
                    end if;
                when stato5 =>
                    if first = '0' then
                       if (my_in = "01001001") then
                           recognized <= true;
                           temp_loop <= stato1;      --deve ritornare comunque all'inizio
                       else
                           temp_loop <= stato1;
                           warning_count <= warning_count +'1';
                       end if;
                    else
                       warning_loop <= '1';
                    end if;
                when others =>
                    temp_loop <= stato1;
            end case;
       end if;

    end process next_state_logic_p;
```

CODICE PER L'AGGIORNAMENTO DELLO STATO DELLA FSM

```
111
112     state_update : process (clk)                    --aggiornamento stato macchina
113         begin
114             if resetn = '1' then
115                 temp <= "00001";
116             elsif falling_edge(clk) then
117                 temp <= temp_loop ;
118                 stato <= temp;
119             end if;
120         end process state_update;
121
```

CONTROLLO DEL SEGNALE DI USCITA

```
123
124     output_logic_p : process (clk, recognized)    --se alto allora sequenza riconosciuta
125       begin
126          if falling_edge(clk) then
127              case recognized is
128                 when true =>
129                    my_out <= '1';
130                 when false =>
131                    my_out <= '0';
132              end case;
133          end if;
134     end process output_logic_p;
135
```

CONTROLLO DEL SEGNALE DI WARNING

```
137
138     warning_check : process (clk, resetn, warning_loop, warning_count)   --controllo del warning
139       begin
140          if resetn = '1' then
141              warning <= '0';
142              warning_temp <= '0';
143          elsif (falling_edge(clk) and resetn = '0') and (warning_loop = '1' or warning_count = "11") then
144              warning <= '1';
145              warning_temp <= '1';
146          end if;
147     end process warning_check;
```

Per il codice completo, si rimanda al file sorgente allegato.

SIMULAZIONI E RISULTATI CONCLUSIVI

Vediamo adesso come risponde il sintetizzatore esaminando tre differenti casistiche.

1) Controllo del valore di uscita una volta inserita la sequenza in ingresso ed il segnale di first nel primo valore.
2) Controllo del segnale di warning quando entrano due segnali alti di first durante la lettura di una sequenza impedendo di riconoscere la sequenza qualora fosse corretta.
3) Conteggio delle tre sequenze consecutive errate per mandare alto il warning fino a nuovo reset.

Naturalmente, in questa relazione vengono mostrati gli istanti salienti della simulazione completa. La restante parte di testbench è conveniente visualizzarla direttamente dal simulatore.

GENERAZIONE DEL TESTBENCH

In questa prima porzione di codice è stata dichiarata la struttura del rilevatore introducendo tutti i segnali che andremo poi ad esaminare nella simulazione.

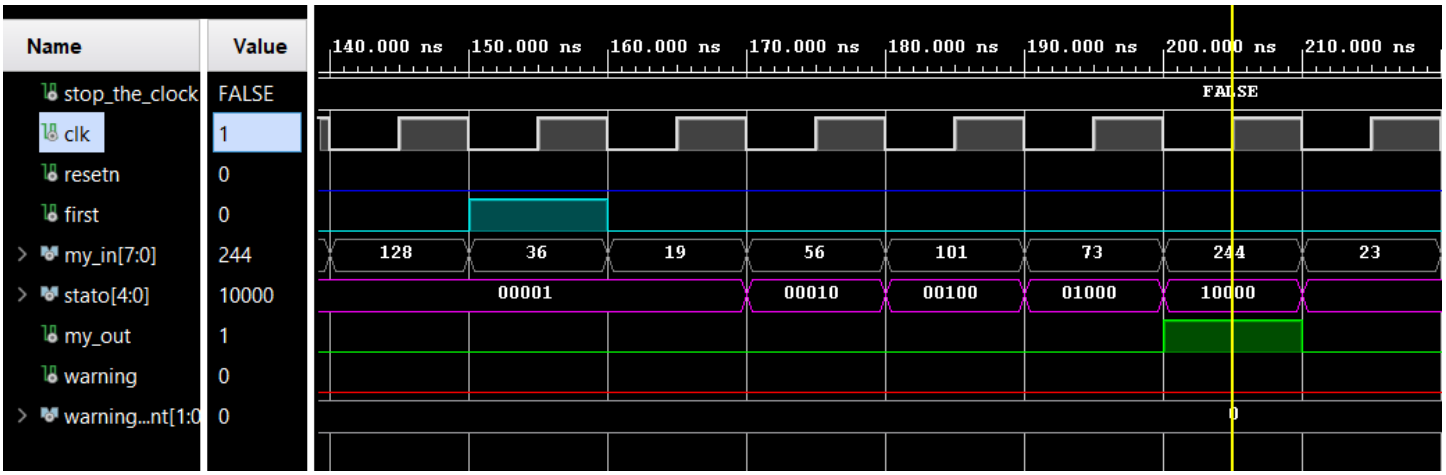
```
1  library IEEE;
2  use IEEE.Std_logic_1164.all;
3  USE IEEE.std_logic_unsigned.all;
4  use IEEE.Numeric_Std.all;
5
6  entity fsm8bit_tb is
7  end;
8
9  architecture bench of fsm8bit_tb is
10
11   component fsm8bit
12     port (
13         clk     :in std_logic;
14         resetn  :in std_logic;
15         my_in   :in std_logic_vector (7 downto 0);
16         first   :in std_logic;
17         my_out  :out std_logic;
18         stato   :out std_logic_vector (4 downto 0);     --eliminabile nel tb
19         warning :out std_logic--;
20 --          cycle   :out std_logic_vector (2 downto 0)    --eliminabile nel tb
21     );
22     end component;
23
24   signal clk:    std_logic;
25   signal resetn: std_logic;
26   signal my_in:  std_logic_vector (7 downto 0);
27   signal my_out: std_logic;
28   signal stato : std_logic_vector (4 downto 0);
29   signal first:  std_logic ;
30   signal warning:std_logic;
31 --  signal cycle  :std_logic_vector (2 downto 0);
33   constant clock_period: time := 10 ns;
34   signal stop_the_clock: boolean;
35
36 begin
37
38   uut: fsm8bit port map ( clk        => clk,
39                           resetn     => resetn,
40                           my_in      => my_in,
41                           my_out     => my_out,
42                           stato      => stato,
43                           first      => first ,
44                           warning    => warning--,
45 --                          cycle      => cycle
46                          );
47
48   stimulus: process
49   begin
```

SIMULAZIONE 1: SEGNALE DI USCITA CON SEQUENZA CORRETTA

Vediamo ora se il riconoscitore è in grado di rilevare una sequenza corretta. Inseriamo la sequenza con il segnale di first alto solo per il primo termine:

```
--correct flow (150-200 ns)
wait until falling_edge(clk);
first <= '1';
my_in <= "00100100" ;
wait until falling_edge(clk);
first <= '0';
my_in <= "00010011" ;
wait until falling_edge(clk);
my_in <= "00111000" ;
wait until falling_edge(clk);
my_in <= "01100101" ;
wait until falling_edge(clk);
my_in <= "01001001" ;
```

La simulazione restituisce questo risultato:



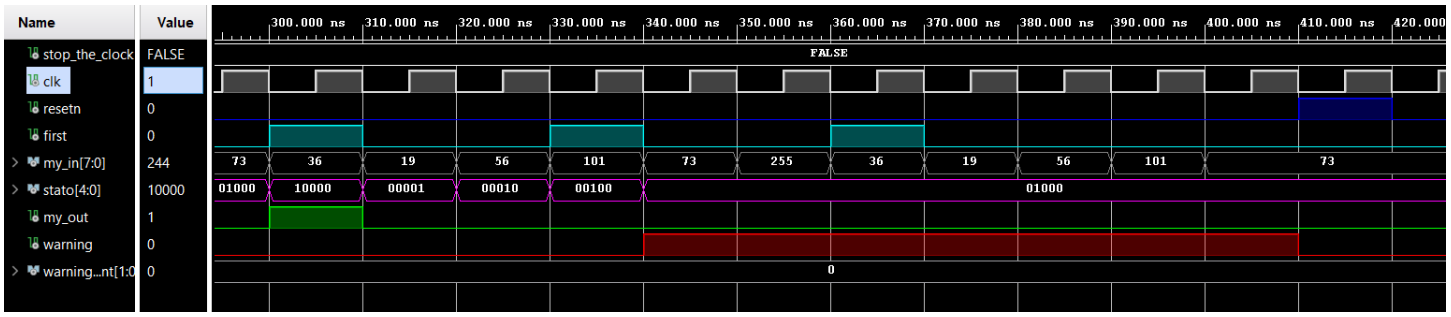
Dal momento in cui viene inserito il segnale di first la macchina a stati finiti inizia il controllo dei numeri in ingresso fino a restituire in uscita 1, per poi ritornare a 0 il ciclo successivo.

SIMULAZIONE 2: WARNING DATI 2 FIRST ALTI

Adesso verifichiamo il corretto funzionamento di warning dati 2 segnali di first alti nella stessa sequenza, come vedremo anche se risulta la sequenza corretta il warning impedisce che l'out si alzi:

```
--warning check caused by two firsts in same sequence (330-360 ns)
wait until falling_edge(clk);
first <= '1';
my_in <= "00100100" ;
wait until falling_edge(clk);
first <= '0';
my_in <= "00010011" ;
wait until falling_edge(clk);
my_in <= "00111000" ;
wait until falling_edge(clk);
first <= '1';
my_in <= "01100101" ;
wait until falling_edge(clk);
first <= '0';
my_in <= "01001001" ;
wait until falling_edge(clk);
my in <= "11111111" ;
```

In uscita risulterà:



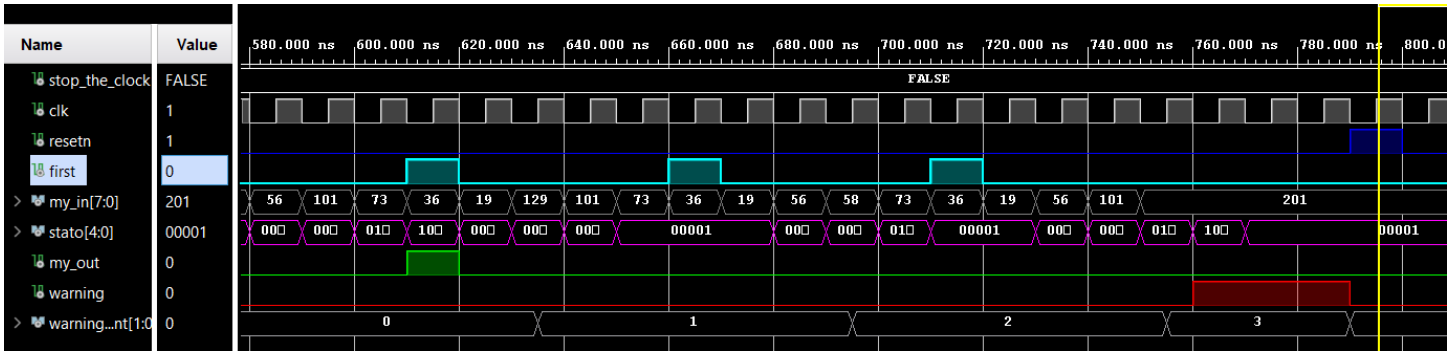
Anzitutto, vediamo come seppur la sequenza sia corretta ed inizializzata con *first*, poiché viene immesso un altro *first* poco dopo, immediatamente il segnale di warning andrà positivo. Anche immettendo un'altra sequenza giusta con relativo *first*, la macchina è bloccata fino al reset.

SIMULAZIONE 3: WARNING DATE 3 SEQUENZE ERRATE

Per vedere meglio questa funzione è stato utilizzato il *warning_count*, che conterà le sequenze consecutive errate per poi mandare alto il valore di warning. Il reset si occuperà di riavviare la macchina.

```
221     --check warning (always high) if the machine cant recognize for 3 times (610-780 ns)
222     wait until falling_edge(clk);
223     first <= '1';
224     my_in <= "00100100" ;
225     wait until falling_edge(clk);
226     first <= '0';
227     my_in <= "00010011" ;
228     wait until falling_edge(clk);
229     my_in <= "10000001" ;          --wrong number
230     wait until falling_edge(clk);
231     my_in <= "01100101" ;
232     wait until falling_edge(clk);
233     my_in <= "01001001" ;
234     wait until falling_edge(clk);
235     first <= '1';
236     my_in <= "00100100" ;
237     wait until falling_edge(clk);
238     first <= '0';
239     my_in <= "00010011" ;
240     wait until falling_edge(clk);
241     my_in <= "00111000" ;
242     wait until falling_edge(clk);
243     my_in <= "00111010" ;          --wrong number
244     wait until falling_edge(clk);
245     my_in <= "01001001" ;
246     wait until falling_edge(clk);
247     first <= '1';
248     my_in <= "00100100" ;
249     wait until falling_edge(clk);
250     first <= '0';
251     my_in <= "00010011" ;
```

Il risultato ben visibile è sull'ultima riga della simulazione:

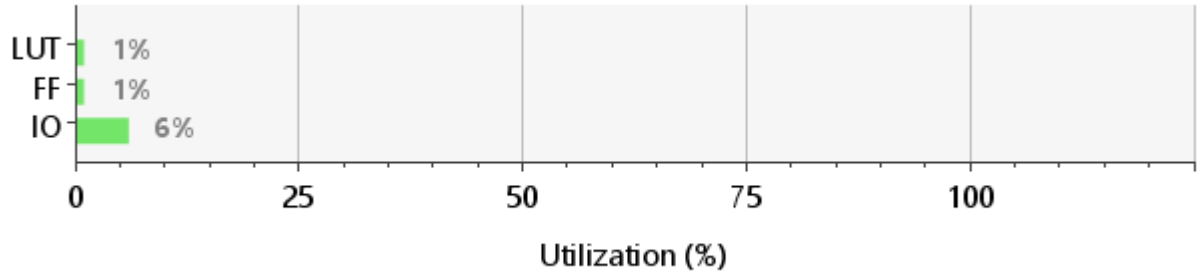


RISULTATI SINTESI LOGICA

Vediamo adesso le principali specifiche di progetto ricavate dal software Xilinx FPGA Vivado:

RISORSE UTILIZZATE

Resource	Utilization	Available	Utilization %
LUT	26	41000	0.06
FF	21	82000	0.03
IO	18	300	6.00



ANALISI NEL DOMINIO DEL TEMPO

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2,677 ns	Worst Hold Slack (WHS):	0,068 ns	Worst Pulse Width Slack (WPWS):	4,650 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	26	Total Number of Endpoints:	26	Total Number of Endpoints:	22

All user specified timing constraints are met.

Il *worst critical path* inficerà sulle massime prestazioni del sistema limitandole a:

$$f_{MAX} = \frac{1}{t_{CLK} - t_{SLACK}} = \frac{1}{10\ ns - 2.677 ns} = 135\ MHz$$

UTILIZZO DI POTENZA

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.084 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25,2°C**
Thermal Margin:	59,8°C (31,6 W)
Effective ϑJA:	1,9°C/W
Power supplied to off-chip devices:	0 W