Elettronica dei Sistemi Digitali
Lab 05

FSM

Gruppo: A11

Pronesti, Massimiliano	S245831
Oropallo, Maria Vittoria	S245999
Nicolicchia, Riccardo	S244728
Miceli, Michelangelo	S245478

22 aprile 2020

Indice

1 One-Hot Finite state machine **2**
- 1.1 Spiegazione teorica . . . 2
- 1.2 Procedimento . . . 2
- 1.3 Risultati . . . 3
- 1.4 Appendice . . . 3

2 Modified One-Hot Finite state machine **8**
- 2.1 Spiegazione teorica . . . 8
- 2.2 Procedimento . . . 8
- 2.3 Risultati . . . 9
- 2.4 Appendice . . . 9

3 Two-process FSM **13**
- 3.1 Spiegazione teorica . . . 13
- 3.2 Procedimento . . . 13
- 3.3 Risultati . . . 14
- 3.4 Appendice . . . 15

4 "HELLO" FSM **19**
- 4.1 Spiegazione teorica . . . 19
- 4.2 Procedimento . . . 19
- 4.3 Risultati . . . 21
- 4.4 Risultati . . . 21
- 4.5 Appendice . . . 22

Capitolo 1

One-Hot Finite state machine

1.1 Spiegazione teorica

Ci si propone di realizzare una FSM a codifica One Hot per il riconoscimento di sequenze di 4 cifre consecutive uguali attraverso l'uso di 9 Flip-Flop di tipo D dotati, inoltre, di un ingresso set, nel quale ogni ingresso è denominato next_state, e ogni uscita current_state.

1.2 Procedimento

Si costruisce la FSM mediante connessione di 9 D-FF in cascata. Si procede per ispezione, facendo riferimento alla tabella mostrata in **figura 1.1**, la cui traduzione comportamentale

state	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
A	0	0	0	0	0	0	0	0	1
B	0	0	0	0	0	0	0	1	0
C	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	1	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0
H	0	1	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0

Figura 1.1: one-hot FSM table

degli stati è descritta nel pallogramma presente nel testo dell'esercitazione.

1.3 Risultati

I risultati prodotti dalla simulazione e dalla sintesi del circuito sono riportati di seguito

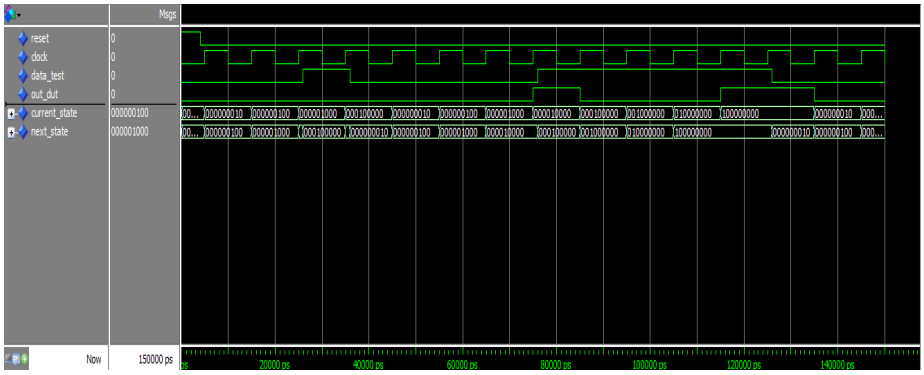

Figura 1.2: wave exercise no. 1

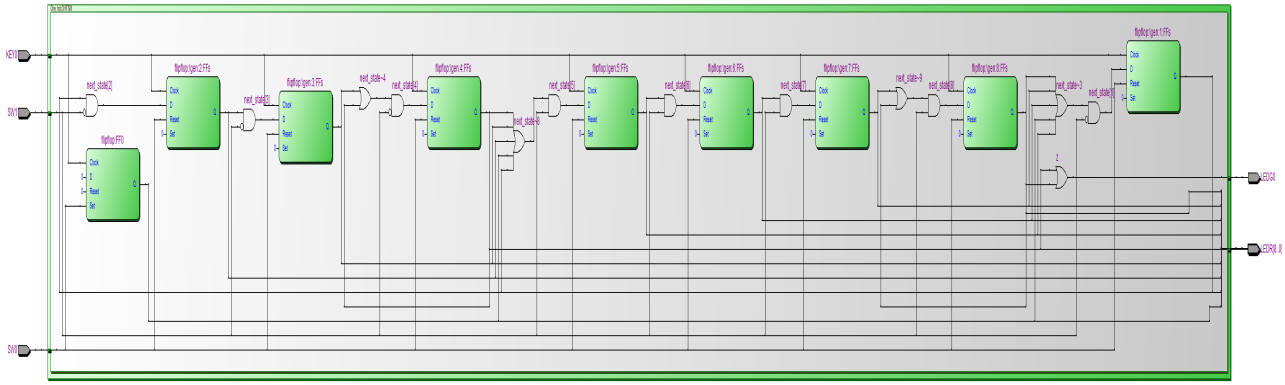

Figura 1.3: RTL exercise no. 1

1.4 Appendice

```
1 library ieee;
```

```
2  use ieee.std_logic_1164.all;
3
4
5  --SW1->w,KEY0->clk,SW0->rst,
6  --LEDR(8 DOWNTO 0)->current_state,
7  --LEDG0->z
8  entity one_hot_synth is
9    port(
10       SW1  : in std_logic;
11       KEY0 : in std_logic;
12       SW0  : in std_logic;
13       LEDR : buffer std_logic_vector (8 DOWNTO 0);
14       LEDG0: out std_logic
15   );
16 end one_hot_synth;
17
18 architecture structure of one_hot_synth is
19
20   component one_hot is
21   port(
22         w: in std_logic;
23         clk: in std_logic;
24         rst: in std_logic;
25         current_state :buffer std_logic_vector(8 downto 0);
26         z: out std_logic
27   );
28   end component;
29   begin
30
31   OHFSM: one_hot port map (SW1, KEY0, SW0, LEDR(8 downto 0), LEDG0);
32 end architecture;
```

Code 1.1: one_hot_synth.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity One_hot is
5    port(
6        w  : in std_logic;
7        clk: in std_logic;
8        rst: in std_logic;
9        z  : out std_logic;
10       current_state :buffer std_logic_vector(8 downto 0)
11   );
12   end One_hot;
13
14   architecture behavior of One_hot is
15
```

```
16     component flipflop is
17       port (
18            D, Clock, Reset,Set : in std_logic;
19            Q : out std_logic
20       );
21       end component;
22
23    signal next_state: std_logic_vector(8 downto 0);
24       begin
25       FF0: flipflop port map (next_state(0),clk,'0',rst,
             current_state(0));
26         gen: for i in 1 to 8 generate
27               FFs: flipflop port map (next_state(i),clk,Rst,'0',
                     current_state(i));
28         end generate;
29
30
31         next_state(0)<='0';
32         next_state(1)<= (current_state(0) or current_state(5) or
               current_state(6) or current_state(7) or current_state(8))
               and not w;
33         next_state(2)<= current_state(1) and not w ;
34         next_state(3)<= current_state(2) and not w;
35         next_state(4)<= (current_state(3) or current_state(4)) and not w;
36         next_state(5)<= (current_state(0) or current_state(1) or
               current_state(2) or current_state(3) or current_state(4)
               )and w;
37         next_state(6)<= current_state(5) and w;
38         next_state(7)<= current_state(6) and w;
39         next_state(8)<= (current_state(7) or current_state(8)) and w;
40         z<= current_state(8) or current_state(4);
41
42  end architecture;
```

Code 1.2: one_hot.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4     entity one_hot_tb is
5     end one_hot_tb;
6
7     architecture DUT of one_hot_tb is
8         component one_hot is
9           port(
10             w  : in std_logic;
11             clk: in std_logic;
12             rst: in std_logic;
13             current_state :buffer std_logic_vector(8 downto 0);
```

```
14            z: out std_logic
15          );
16       end component;
17
18     signal reset,clock,data_test:std_logic;
19     signal out_dut: std_logic;
20
21    begin
22       reset<='1','0' after 4 ns;
23
24       CLK_PR : process begin
25         clock <= '0';
26         wait for 5 ns;
27         clock <= '1';
28         wait for 5 ns;
29       end process;
30
31
32       data_test<='0',
33            '1' after 26 ns,
34            '0' after 36 ns,
35            '1' after 76 ns,
36            '0' after 126 ns;
37
38   DUT: One_hot port map (w=>data_test,clk=>clock,rst=>reset,z=>out_dut);
39
40 end architecture;
```

Code 1.3: one_hot_tb.vhd

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4
5
6  entity flipflop is
7    port (
8      D, Clock, Reset,Set : in std_logic;
9     Q : out std_logic
10   );
11 end flipflop;
12
13
14 architecture Behavior of flipflop is
15 begin
16
17   process (Clock, Reset,Set)
18     begin
19       if (Reset = '1') then -- asynchronous clear
```

```
20          Q <= '0';
21        elsif Set = '1' then -- asynchronous set
22          Q <= '1';
23        elsif (Clock'EVENT and Clock = '1') then
24          Q <= D;
25        end if;
26      end process;
27
28  end Behavior;
```

Code 1.4: FlipFlop.vhd

Capitolo 2

Modified One-Hot Finite state machine

2.1 Spiegazione teorica

Si modifica la precedente FSM in modo che lo stato A diventi uno stato di Reset.

2.2 Procedimento

Si apportano le lievi modifiche alla descrizione mostrata in **Code 1.2** secondo la tabella mostrata in **figura 2.1**.

state	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
A	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	1	1
C	0	0	0	0	0	0	1	0	1
D	0	0	0	0	0	1	0	0	1
E	0	0	0	0	1	0	0	0	1
F	0	0	0	1	0	0	0	0	1
G	0	0	1	0	0	0	0	0	1
H	0	1	0	0	0	0	0	0	1
I	1	0	0	0	0	0	0	0	1

Figura 2.1: one-hot modified FSM table

2.3 Risultati

Di seguito sono riportati i risultati prodotti dalla simulazione e dalla sintesi.

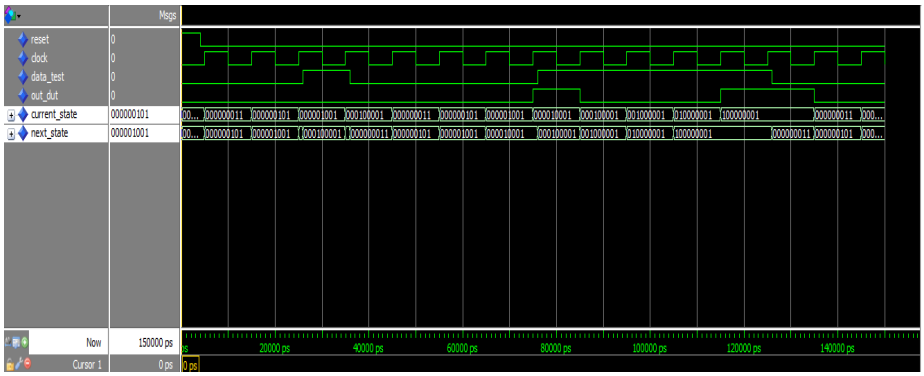

Figura 2.2: wave exercise no. 2

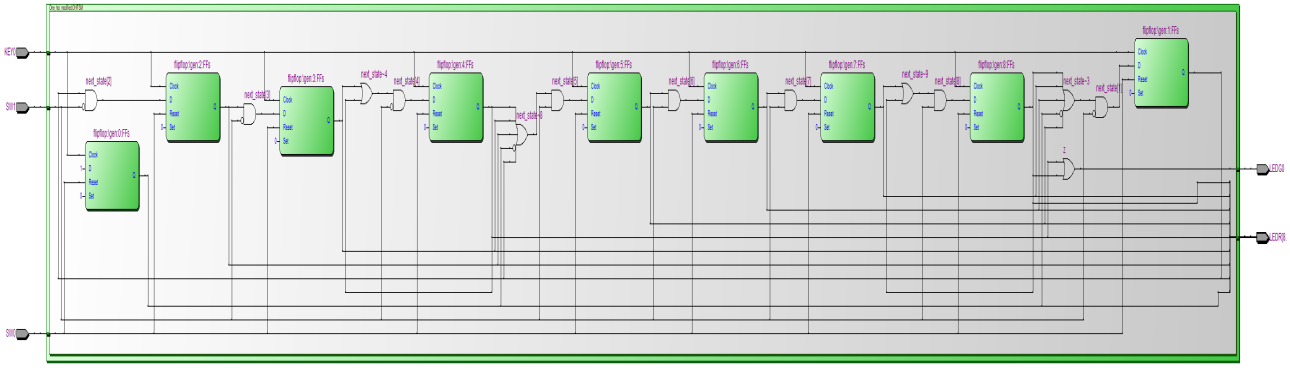

Figura 2.3: RTL exercise no. 2

2.4 Appendice

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  --SW1->w,KEY0->clk,SW0->rst,
5  --LEDR(8 DOWNTO 0)->current_state,
6  --LEDG0->z
```

```
7  entity one_hot_modified_synth is
8    port(
9        SW1  : in std_logic;
10       KEY0 : in std_logic;
11       SW0  : in std_logic;
12       LEDR : buffer std_logic_vector (8 DOWNTO 0);
13       LEDG0: out std_logic
14    );
15 end one_hot_modified_synth;
16
17 architecture structure of One_hot_modified_synth is
18
19   component One_hot_modified is
20     port(
21         w: in std_logic;
22         clk: in std_logic;
23         rst: in std_logic;
24          current_state :buffer std_logic_vector(8 downto 0);
25        z: out std_logic
26      );
27     end component;
28   begin
29
30   OHFSM: one_hot_modified port map (SW1,KEY0,SW0,LEDR(8 downto 0),LEDG0);
31   end architecture;
```

Code 2.1: one_hot_modified_synth.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity one_hot_modified is
5    port(
6        w: in std_logic;
7        clk: in std_logic;
8        rst: in std_logic;
9        current_state :buffer std_logic_vector(8 downto 0);
10       z: out std_logic
11   );
12 end one_hot_modified;
13
14 architecture behavior of One_hot_modified is
15
16    component flipflop is
17      port (
18           D, Clock, Reset,Set : in std_logic;
19           Q : out std_logic
20      );
21      end component;
```

```
22
23      signal next_state: std_logic_vector(8 downto 0);
24       begin
25
26          -- FF instanciation
27          gen: for i in 0 to 8 generate
28              FFs: flipflop port map (next_state(i),clk,Rst,'0',
                    current_state(i));
29          end generate;
30
31
32        -- state table
33          next_state(0) <= '1';
34          next_state(1) <= ( not current_state(0) or current_state(5) or
                current_state(6) or current_state(7) or current_state(8))
                and not w;
35          next_state(2) <= current_state(1) and not w ;
36          next_state(3) <= current_state(2) and not w;
37          next_state(4) <= (current_state(3) or current_state(4)) and not
                w;
38          next_state(5) <= (not current_state(0) or current_state(1) or
                current_state(2) or current_state(3) or current_state(4)
                )and w;
39          next_state(6) <= current_state(5) and w;
40          next_state(7) <= current_state(6) and w;
41          next_state(8) <= (current_state(7) or current_state(8)) and w;
42          z <= current_state(8) or current_state(4) ;
43
44  end architecture;
```

Code 2.2: on_hot_modified.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity one_hot_modified_tb is
5   end one_hot_modified_tb;
6
7   architecture DUT of one_hot_modified_tb is
8
9      component One_hot_modified is
10       port(
11          w: in std_logic;
12          clk: in std_logic;
13          rst: in std_logic;
14          current_state :buffer std_logic_vector(8 downto 0);
15       z: out std_logic
16       );
17      end component;
```

```
18
19      signal reset,clock,data_test:std_logic;
20      signal out_dut: std_logic;
21
22    begin
23        reset<='1','0' after 4 ns;
24
25        -- clock process
26       CLK_PR : process begin
27          clock <= '0';
28          wait for 5 ns;
29          clock <= '1';
30          wait for 5 ns;
31       end process;
32
33
34       -- input testing
35         data_test<='0',
36               '1' after 26 ns,
37               '0' after 36 ns,
38               '1' after 76 ns,
39               '0' after 126 ns;
40
41    DUT: One_hot_modified port map
          (w=>data_test,clk=>clock,rst=>reset,z=>out_dut);
42
43  end architecture;
```

Code 2.3: one_hot_modified_tb.vhd

Capitolo 3

Two-process FSM

3.1 Spiegazione teorica

Si implementa la FSM descritta tramite il pallogramma presente nel testo dell'esercitazione utilizzando il tipico stile descritto di una macchina a stati in linguaggio VHDL, i.e. tramite **case** statements inseriti in **process** blocks.

3.2 Procedimento

Si opta per una descrizione dell'hardware secondo tre process:

- **STATE_TRANSITION**, nel quale viene specificato lo stato futuro della macchina in funzione dello stato presente e del valore di 'w';
- **FFs**, nel quale si gestisce il reset sincrono e, ad ogni fronte positivo del clock, si ha il passaggio dallo stato presente a quello futuro;
- **OUT_DEC**, il quale gestisce il valore dell'uscita z.

In funzione dello stato presente, sono assegnati i valori relativi ai LEDRs. Si è, infine, usato un file di sintesi per collegare le entrate e le uscite della macchina agli switches ed ai led della DE2.

3.3 Risultati

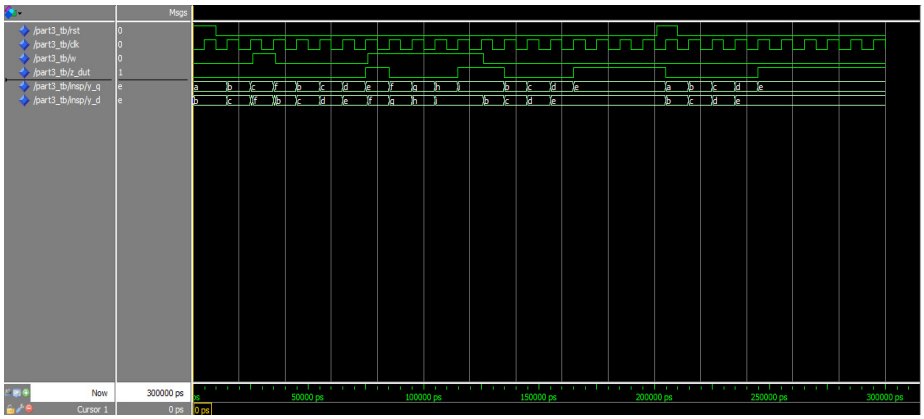

Figura 3.1: wave exercise no. 3

	Name	y_Q.I	y_Q.H	y_Q.G	y_Q.F	y_Q.E	y_Q.D	y_Q.C	y_Q.B	y_Q.A
1	y_Q.A	0	0	0	0	0	0	0	0	0
2	y_Q.B	0	0	0	0	0	0	0	1	1
3	y_Q.C	0	0	0	0	0	0	1	0	1
4	y_Q.D	0	0	0	0	0	1	0	0	1
5	y_Q.E	0	0	0	0	1	0	0	0	1
6	y_Q.F	0	0	0	1	0	0	0	0	1
7	y_Q.G	0	0	1	0	0	0	0	0	1
8	y_Q.H	0	1	0	0	0	0	0	0	1
9	y_Q.I	1	0	0	0	0	0	0	0	1

Figura 3.2: One hot state table

	Name	y_Q.state_bit_3	y_Q.state_bit_2	y_Q.state_bit_1	y_Q.state_bit_0
1	y_Q.A	0	0	0	0
2	y_Q.B	0	0	0	1
3	y_Q.C	0	0	1	1
4	y_Q.D	0	1	0	0
5	y_Q.E	0	1	0	1
6	y_Q.F	0	0	1	0
7	y_Q.G	0	1	1	0
8	y_Q.H	0	1	1	1
9	y_Q.I	1	0	0	0

Figura 3.3: Minimal bit state

Si osserva che nella codifica 'One-hot' ciascuno stato è rappresentato su 8 bit, mentre nella codifica 'Minimal Bits' sono necessari solo 4 bit per stato.

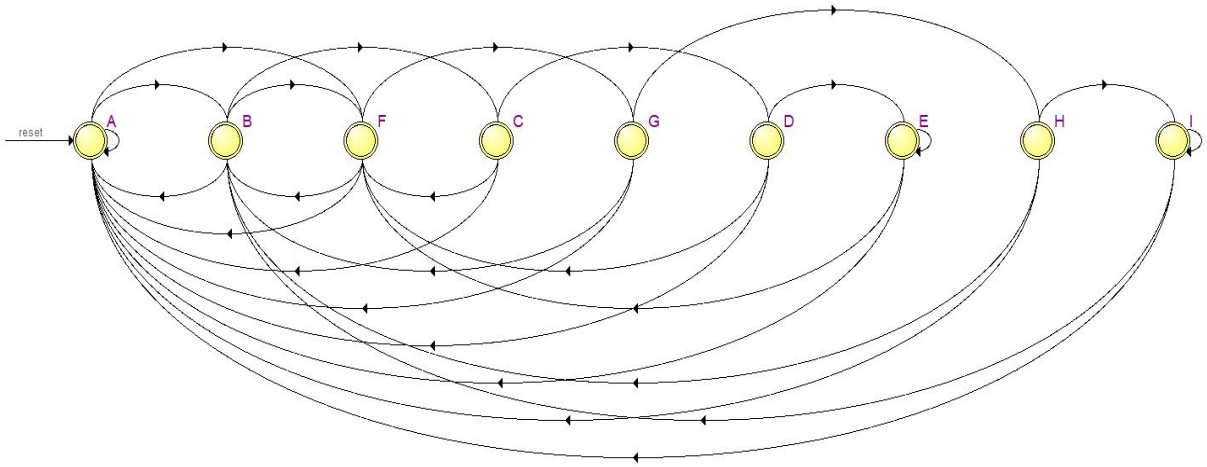

Figura 3.4: State diagram exercise no. 3

3.4 Appendice

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity FSM_synth is
5    port ( SW : in std_logic_vector(0 to 1);
6           KEY0 : in std_logic;
7           LEDR : out std_logic_vector(8 downto 0);
8           LEDG0 : out std_logic
9         );
10 end FSM_synth;
11
12
13
14 architecture gate of FSM_synth is
15
16   component part3
17     port (
18          Rst,clk,w : in std_logic;
19          CS : out std_logic_vector ( 8 downto 0);
20          z : out std_logic
```

```
21        );
22    end component;
23
24
25    begin
26     FSM: part3 port map(Rst=>SW(0), clk=>KEY0,
           w=>SW(1),CS=>LEDR,z=>LEDG0);
27
28  end architecture;
```

Code 3.1: FSM_synth.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity part3 is
5     port ( Rst,clk,w : in std_logic;
6            CS : out std_logic_vector ( 8 downto 0);
7            z : out std_logic
8          );
9    end part3;
10
11
12  architecture beh of part3 is
13
14    Type state_type is (A,B,C,D,E,F,G,H,I);
15    signal y_Q : state_type; -- counter state
16    signal Y_D : state_type; -- next state
17
18
19  begin
20
21    STATE_TRANSITION: process (w,y_Q)
22    begin
23      case y_Q is
24       when A =>
25         if w = '0' then Y_D <= B; else Y_D <= F; end if;
26        when B =>
27         if w = '0' then Y_D <= C; else Y_D <= F; end if;
28        when C =>
29         if w = '0' then Y_D <= D; else Y_D <= F; end if;
30        when D =>
31         if w = '0' then Y_D <= E; else Y_D <= F; end if;
32        when E =>
33         if w = '0' then Y_D <= E; else Y_D <= F; end if;
34        when F =>
35         if w = '0' then Y_D <= B; else Y_D <= G; end if;
36        when G =>
37         if w = '0' then Y_D <= B; else Y_D <= H; end if;
```

```
38        when H =>
39         if w = '0' then Y_D <= B; else Y_D <= I; end if;
40        when I =>
41         if w = '0' then Y_D <= B; else Y_D <= I; end if;
42        when others =>
43          Y_D <= A; --return on reset if unknown states
44      end case;
45    end process;
46
47
48    FFs: process (clk)
49
50    begin
51      if clk'event and clk = '1' then
52        if Rst = '1' then
53          y_Q <= A;
54        else
55          y_Q <= Y_D;
56        end if;
57      end if;
58    end process;
59
60    OUT_DEC: process (y_Q)
61
62    begin
63      z <= '0';
64      if (y_Q = E or y_Q = I) then
65        z <= '1';
66      end if;
67    end process;
68
69    --CS out assignment
70    with y_Q select
71       CS <=
72        "000000001" when A,
73        "000000010" when B,
74        "000000100" when C,
75        "000001000" when D,
76        "000010000" when E,
77        "000100000" when F,
78        "001000000" when G,
79        "010000000" when H,
80        "100000000" when I,
81        "000000000" when others;
82
83
84  end architecture;
```

Code 3.2: part3.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;


entity part3_tb is
end part3_tb;


architecture struct of part3_tb is

  component part3
    port ( Rst,clk,w : in std_logic;
         CS : out std_logic_vector ( 8 downto 0);
         z : out std_logic
       );
  end component;

  signal Rst,clk,w,z_dut : std_logic;

begin

  CLK_GEN: process
  begin
    clk <= '0','1' after 5 ns;
    wait for 10 ns;
  end process CLK_GEN;

  Rst <= '1', '0' after 10 ns, '1' after 201 ns,'0' after 210 ns;

  w <= '0','1' after 26 ns,'0' after 36 ns,'1' after 76 ns,'0' after 126
      ns;

  INSP: part3 port map (Rst=>Rst,clk=>clk,w=>w,z=>z_dut);

end architecture;
```

Code 3.3: part3_tb.vhd

Capitolo 4

"HELLO" FSM

4.1 Spiegazione teorica

In quest'utimo capitolo, ci si è occupati dell'implementazione di una macchina a stati finiti che permette, dopo una preliminare fase di load, di visualizzare la parola "HELLO " (inclusi i 3 blanks) ad intervalli di un secondo in un loop infinito.

4.2 Procedimento

Si implementa lo schema a blocchi mostrato in **figura 4.2** che fa riferimento al pallogramma mostrato in **figura 4.1** che può essere ripartito in due macro gruppi : stati di load che si occupano dell'inizializzazione della macchina, adempiendo alla fase 1 delineata nel testo dell'esercitazione, e stato di SCROLL che, sfruttando gli enable dei registri ed il collegamento ciclico, permette lo scorrimento della parola.
In merito all'implementazione in linguaggio VHDL, si osservi, che, come suggerito a lezione, non è stata fatta alcuna ottimizzazione nella descrizione dei "case" al fine di preservarne la leggibilità.

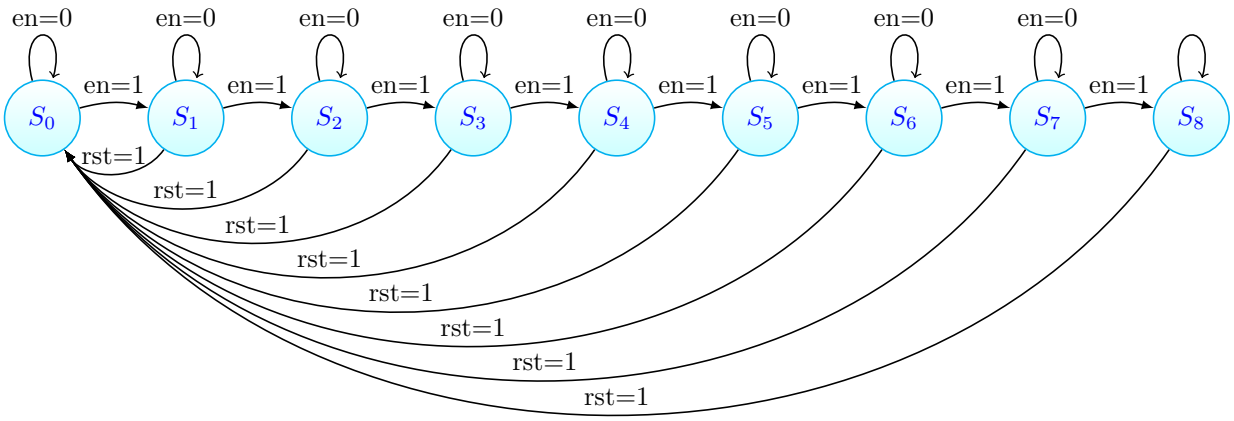

Figura 4.1: Expected state diagram

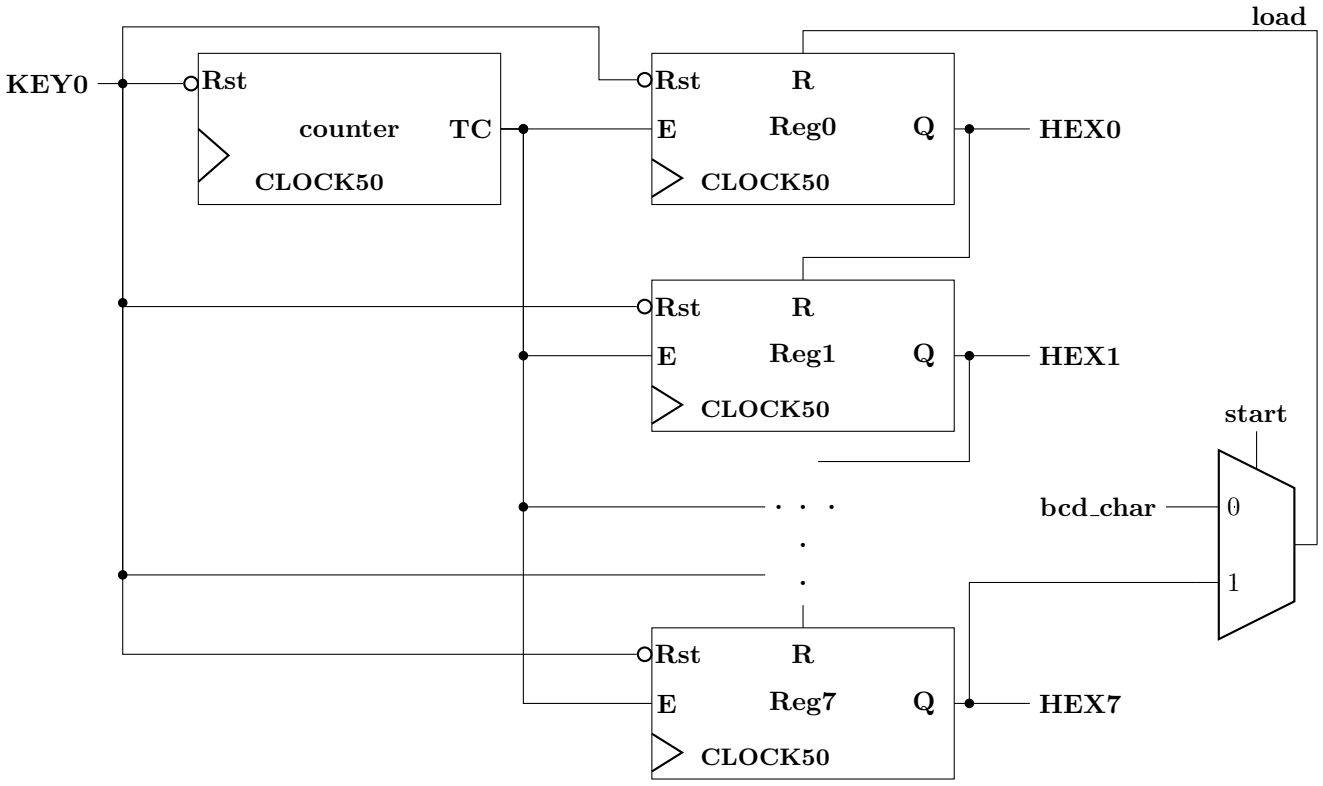

Figura 4.2: Block Scheme

4.3 Risultati

4.4 Risultati

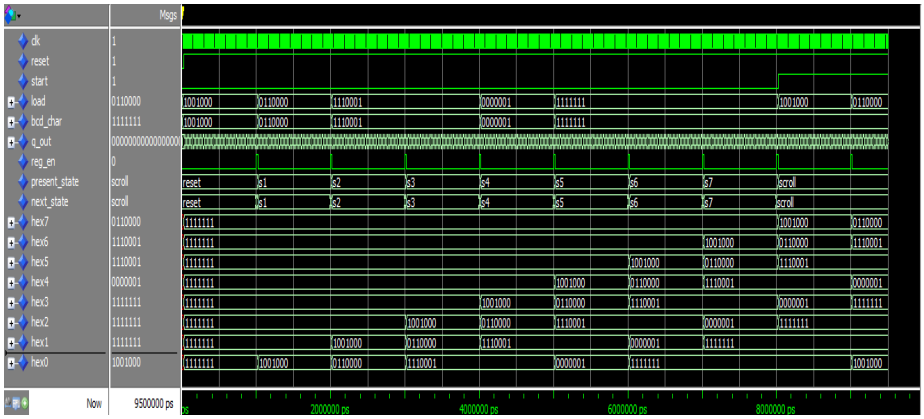

Figura 4.3: wave exercise no. 4

Si osservi che il parametro FREQ è stato ridotto per apprezzare i risultati della simulazione.

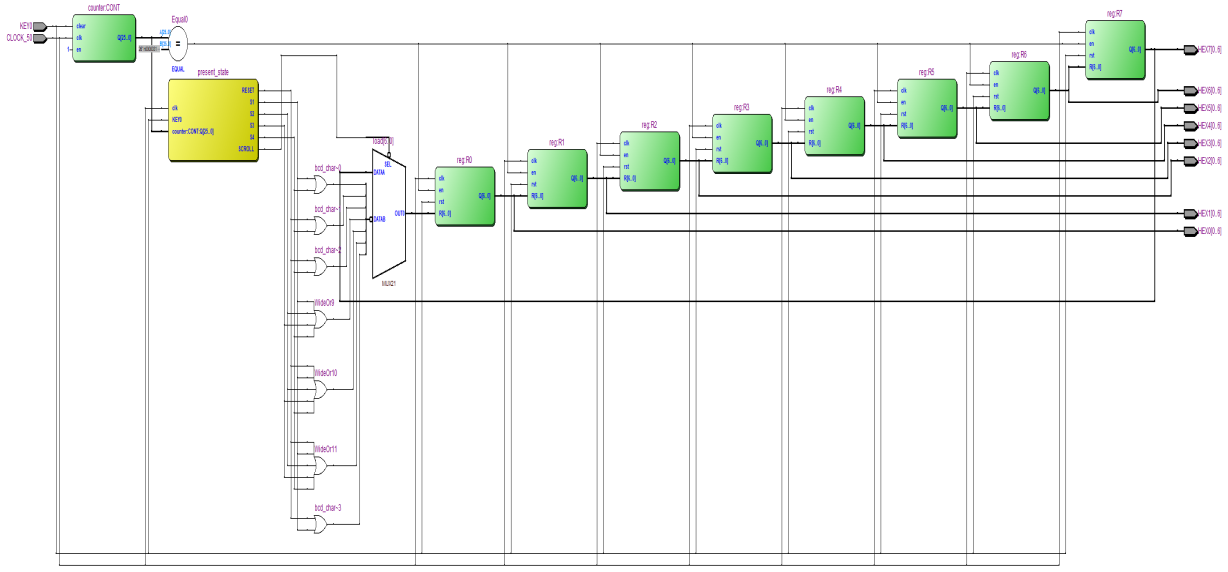

Figura 4.4: RTL exercise no. 4

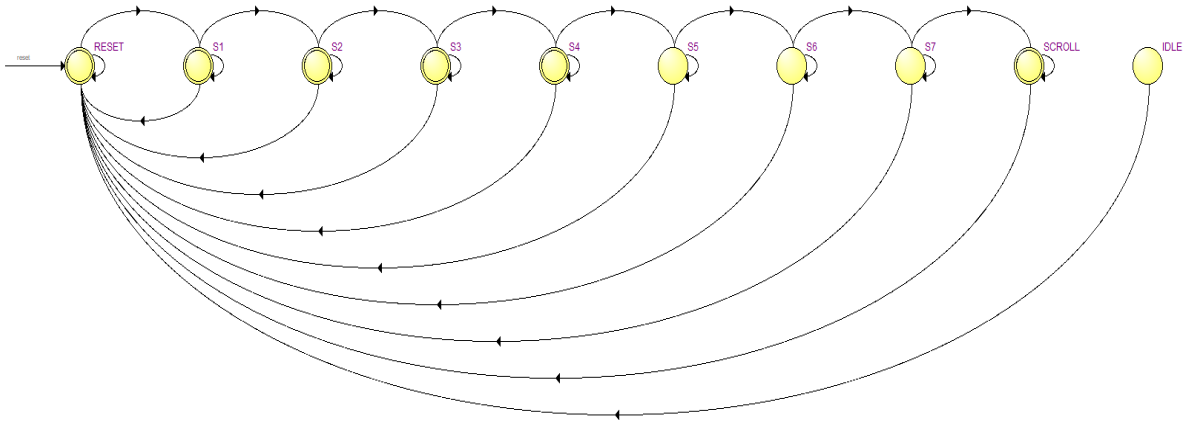

Figura 4.5: State diagram exercise no. 3

4.5 Appendice

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity hello_fsm is
6    port(
7      KEY0                : in std_logic; -- reset
8      CLOCK_50            : in std_logic; -- clock
9      HEX7, HEX6, HEX5, HEX4,
10     HEX3, HEX2, HEX1, HEX0 : out std_logic_vector(0 to 6)
11   );
12 end entity;
13
14
15 architecture behavioral of hello_fsm is
16
17   component reg is
18     port(
19       clk, rst, en : in std_logic;
20       R            : in std_logic_vector(6 downto 0);
21       Q            : out std_logic_vector(6 downto 0)
22     );
23   end component;
24
25   component counter is
26     generic (N : integer := 16; UP_TO : integer := 2**16 - 1);
27     port(
```

```
28          en, clk, clear : in std_logic;
29          Q : out unsigned (N-1 downto 0)
30      );
31    end component;
32
33     -- states definition
34    type  STATE is (RESET, S1, S2, S3, S4, S5, S6, S7, SCROLL);
35    signal present_state : STATE;
36    signal next_state   : STATE;
37
38    signal bcd_char    : std_logic_vector ( 0 to 6);
39
40    signal start : std_logic; -- flag signal to start scrolling loop
41    signal load : std_logic_vector( 6 downto 0);
42
43    signal Q_out : unsigned (25 downto 0); -- counter output signal
44    signal reg_en : std_logic; -- register enable;
45
46    -- tmp signals
47    signal H0, H1, H2, H3,
48           H4, H5, H6, H7 : std_logic_vector (0 to 6);
49
50
51    -- constants to improve readability
52    constant H    : std_logic_vector(6 downto 0) := "1001000" ;
53    constant E    : std_logic_vector(6 downto 0) := "0110000" ;
54    constant L    : std_logic_vector(6 downto 0) := "1110001" ;
55    constant O    : std_logic_vector(6 downto 0) := "0000001" ;
56    constant BLANK : std_logic_vector(6 downto 0) := "1111111" ;
57    constant FREQ : integer := 50000000;
58
59    begin
60      -- counter instantiation
61      CONT : counter generic map (26, FREQ - 1)
62             port map ('1', CLOCK_50, KEY0, Q_out);
63
64
65      reg_en <= '1' when Q_out = FREQ - 1 else '0';
66
67      -- main
68      TRANSIT_PROC : process (present_state, reg_en) begin
69
70        case present_state is
71          when RESET => if reg_en = '1' then next_state <= S1;
72                        else next_state <= RESET;
73                        end if;
74          when S1 => if reg_en = '1' then next_state <= S2;
75                     else next_state <= S1;
76                        end if;
77          when S2 => if reg_en = '1' then next_state <= S3;
```

```
78                  else next_state <= S2;
79                  end if;
80          when S3 => if reg_en = '1' then next_state <= S4;
81                  else next_state <= S3;
82                  end if;
83          when S4 => if reg_en = '1' then next_state <= S5;
84                  else next_state <= S4;
85                  end if;
86          when S5 => if reg_en = '1' then next_state <= S6;
87                  else next_state <= S5;
88                  end if;
89          when S6 => if reg_en = '1' then next_state <= S7;
90                  else next_state <= S6;
91                  end if;
92          when S7 => if reg_en = '1' then next_state <= SCROLL;
93                  else next_state <= S7;
94                  end if;
95          when SCROLL => next_state <= SCROLL;
96          when others => next_state <= RESET; -- back to reset state
97        end case;
98      end process;
99
100
101       REG_PROC : process (CLOCK_50) begin
102         if CLOCK_50'event and CLOCK_50 = '1' then
103           if KEY0 = '0' then -- syn reset
104             present_state <= RESET;
105           else
106             present_state <= next_state;
107           end if;
108         end if;
109       end process;
110
111
112     -- loading correct char in the first reg
113     OUT_PROC : process (present_state) begin
114       start <= '0'; bcd_char <= BLANK;
115       case present_state is
116         when RESET => bcd_char <= H;
117         when S1 => bcd_char <= E;
118         when S2 => bcd_char <= L;
119         when S3 => bcd_char <= L;
120         when S4 => bcd_char <= O;
121         when S5 => bcd_char <= BLANK;
122         when S6 => bcd_char <= BLANK;
123         when S7 => bcd_char <= BLANK;
124         when SCROLL => start <= '1';
125         when others => bcd_char <= BLANK;
126       end case;
127     end process;
```

```
128
129     load <= bcd_char when (start = '0') else H7;
130
131     -- registers instantiation
132     R0 : reg port map (CLOCK_50, KEY0, reg_en, load, H0);
133     R1 : reg port map (CLOCK_50, KEY0, reg_en, H0, H1);
134     R2 : reg port map (CLOCK_50, KEY0, reg_en, H1, H2);
135     R3 : reg port map (CLOCK_50, KEY0, reg_en, H2, H3);
136     R4 : reg port map (CLOCK_50, KEY0, reg_en, H3, H4);
137     R5 : reg port map (CLOCK_50, KEY0, reg_en, H4, H5);
138     R6 : reg port map (CLOCK_50, KEY0, reg_en, H5, H6);
139     R7 : reg port map (CLOCK_50, KEY0, reg_en, H6, H7);
140
141     HEX7 <= H7;
142     HEX6 <= H6;
143     HEX5 <= H5;
144     HEX4 <= H4;
145     HEX3 <= H3;
146     HEX2 <= H2;
147     HEX1 <= H1;
148     HEX0 <= H0;
149 end architecture;
```

Code 4.1: hello_fsm.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity hello_fsm_tb is
5  end entity;
6
7  architecture testing of hello_fsm_tb is
8    component hello_fsm is
9    port(
10     KEY0                 : in std_logic; -- reset
11     CLOCK_50             : in std_logic; -- clock
12     HEX7, HEX6, HEX5, HEX4,
13     HEX3, HEX2, HEX1, HEX0 : out std_logic_vector(6 downto 0)
14   );
15   end component;
16
17   signal clk   : std_logic := '1';
18   signal reset : std_logic := '0';
19   signal H0, H1, H2, H3,
20        H4, H5, H6, H7 : std_logic_vector(6 downto 0);
21
22   constant Ts : time := 20 ns;
23
24   begin
```

```
25      FSM_M : hello_fsm port map (reset, clk, H7, H6, H5, H4, H3, H2, H1,
            H0);
26
27      process begin
28        clk <= '1';
29        wait for Ts/2;
30        clk <= '0';
31        wait for Ts/2;
32      end process;
33
34    reset <= '0', '1' after 25 ns;
35
36  end architecture;
```

Code 4.2: hello_fsm_tb.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity counter is
6     generic (N : integer := 16; UP_TO : integer := 2**16 - 1);
7     port(
8         en, clk, clear : in std_logic;
9         Q : out unsigned (N-1 downto 0)
10    );
11  end counter;
12
13  architecture behavior of counter is
14    signal cnt : integer range 0 to 2**N - 1;
15
16    begin
17      process(clear, clk) begin
18
19        if clk'event and clk = '1' then
20          if (clear='0') then -- synch clear
21             cnt <= 0;
22          elsif (en='1') then
23            if cnt = UP_TO then -- range check
24              cnt <= 0;
25            else
26              cnt <= cnt + 1;
27            end if;
28          end if;
29        end if;
30
31      end process;
32
33      Q <= to_unsigned(cnt, N);
```

```
34 end architecture;
```

Code 4.3: counter.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4
5  entity reg is
6    port(
7      clk, rst,en : in std_logic;
8      R           : in std_logic_vector(6 downto 0);
9      Q           : out std_logic_vector(6 downto 0)
10   );
11 end reg;
12
13
14
15 architecture behavioral of reg is
16  begin
17   process (clk) begin
18       if clk'event and clk = '1' then
19         if rst = '0' then     -- synch reset
20           Q <= (others => '1'); -- BLANK init
21         elsif en = '1' then
22           Q<= R;
23         end if;
24       end if;
25     end process;
26
27 end architecture;
```

Code 4.4: register.vhd