Elaborato di Architettura dei Sistemi di Elaborazione
Gruppo 21

Alessandro Placido Luise - Mat. M63/818 Giuseppina Tuoro - Mat. M63/811

18 gennaio 2020

Indice

Capitolo 1

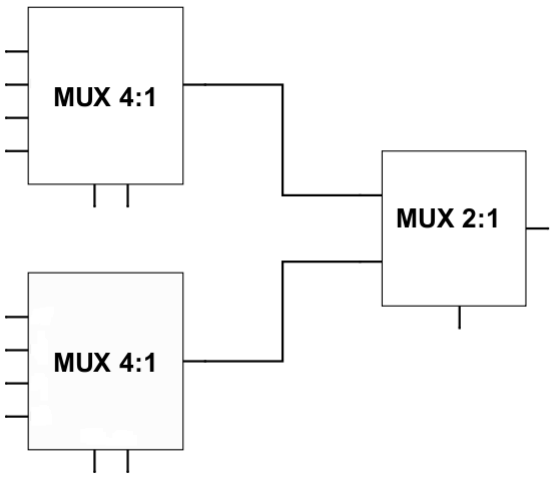
Esercizio 1

1.1 Traccia

Progettare ed implementare in VHDL un multiplexer 8:1 indirizzabile utilizzando una descrizione di tipo structural che componga opportunamente multiplexer più piccoli. Nota: il progetto deve fare uso di almeno un multiplexer 4:1, progettato con una tecnica a scelta dello studente.

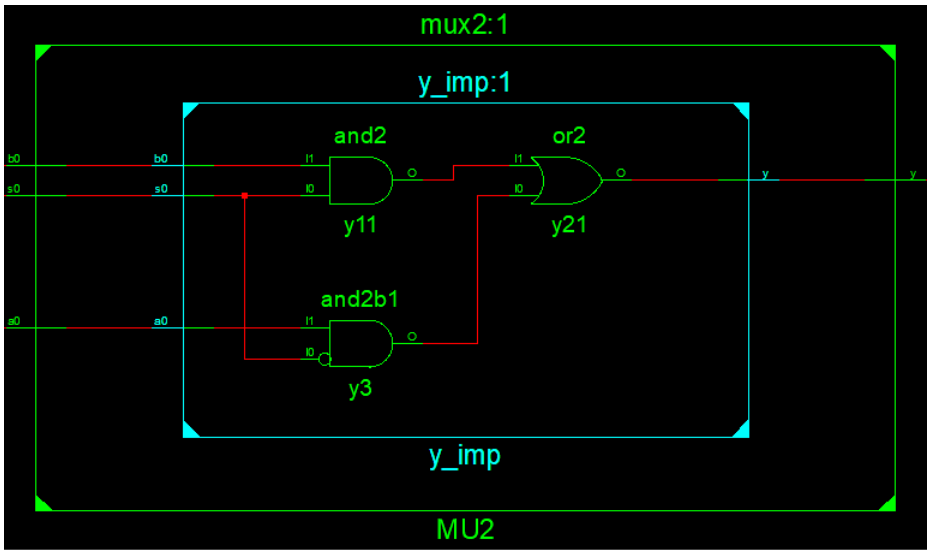
1.2 Soluzione

Si è scelto di implementare il multiplexer 8:1 utilizzando due multplexer 4:1 e uno 2:1 collegati in cascata. Lo schema è riportato nell'immagine seguente :

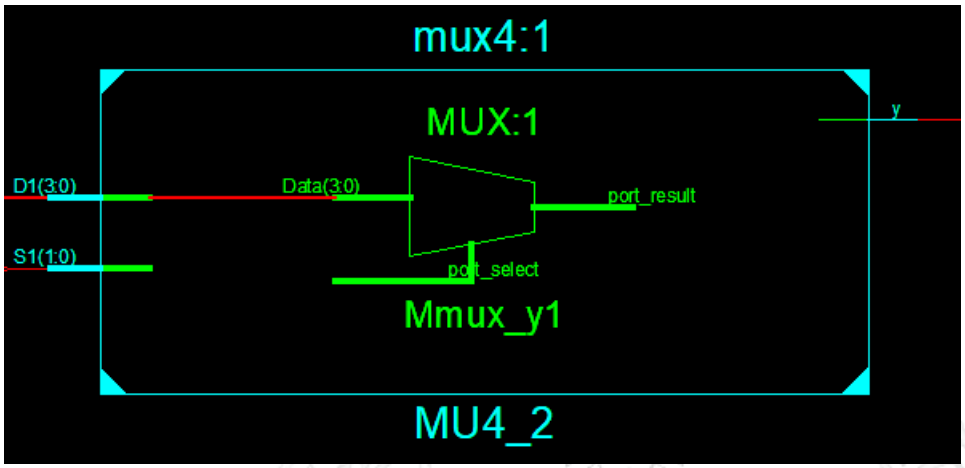


Il multiplexer da 8 è stato implementato utilizzando una descrizione strutturale, mentre per gli altri è stata utilizzata una descrizione di tipo DataFlow.

Di seguito è mostrato lo schematico del multiplexer 2:1 implementato.

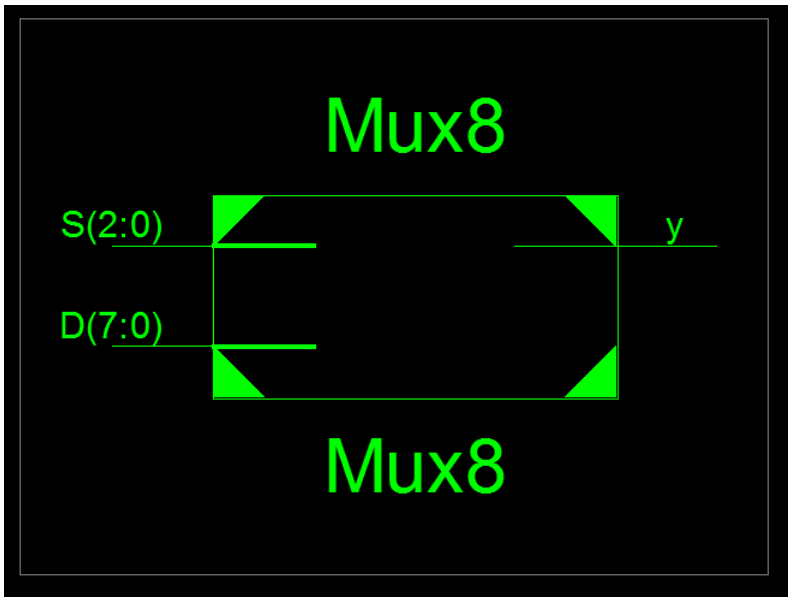


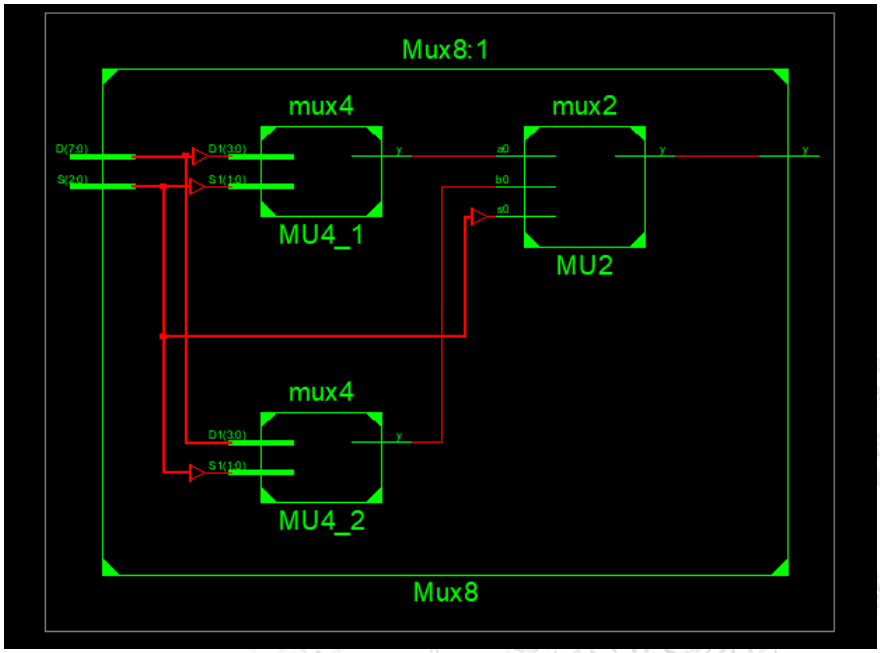
Il multiplexer 4:1 è stato sviluppato seguendo la stessa logica DataFlow.



1.2.1 Schematici

Lo schematico del mux 8:1 che ne segue è composizione dei due visti precedentemente.





1.2.2 Codice

1.2.2.1 Multiplexer 2:1

Il Mux 2:1 pone l'uscita pari ad a0 se s0 è pari a 0, a b0 se s0 è pari ad 1. In tutti gli altri casi l'uscita è un "don't care".

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity mux2 is
5   PORT (
6       s0 : in  std_logic;
```

```
        a0,b0: in  std_logic;
      y : out std_logic
      );
end mux2;

architecture DataFlow of mux2 is
begin
y<=  a0 when (s0 = '0') else
  b0 when (s0 = '1') else
  '-';
end DataFlow;
```

Codice Componente 1.1: Definizione del componente Bit String Comparator Generic

1.2.2.2 Multiplexer 4:1

Discorso simile per il Mux 4:1, che possiede invece un segnale di selezione a 2 bit.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux4 is
PORT (
  S1 : in  std_logic_vector(1 downto 0);
  D1 : in  std_logic_vector(3 downto 0);
  y : out std_logic );
end mux4;

architecture DataFlow of mux4 is
  begin
  y <= D1(0)  when S1 = "00" else
     D1(1)  when S1 = "01" else
     D1(2)  when S1 = "10" else
     D1(3)  when S1 = "11" else
     '-';
end DataFlow;
```

Codice Componente 1.2: Definizione del componente Bit String Comparator Generic

1.2.2.3 Multiplexer 8:1

Il Mux 8:1 viene descritto con una logica strutturale, come richiesto dalla traccia, ed utilizzando almeno un Mux 4:1.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux8 is
PORT (
```

```
7       S : in  std_logic_vector(2 downto 0);
8       D : in  std_logic_vector(7 downto 0);
9       y : out std_logic
10      );
11  end Mux8;
12
13  architecture Structural of Mux8 is
14
15  Component mux4
16    PORT (
17      S1 : in  std_logic_vector(1 downto 0);
18      D1 : in  std_logic_vector(3 downto 0);
19      y : out std_logic );
20  end Component;
21
22  Component mux2
23    PORT (
24      s0 : in  std_logic;
25      a0,b0: in  std_logic;
26      y : out std_logic );
27  end Component;
28
29  signal c0, c1 : std_logic :='0';
30
31  begin
32
33  MU4_1 : mux4 port map (
34      D1(0) => D(0),
35      D1(1) => D(1),
36      D1(2) => D(2),
37      D1(3) => D(3),
38      S1(0) => S(1),
39      S1(1) => S(0),
40      y  => c0
41      );
42  MU4_2 : mux4 port map (
43      D1(0) => D(4),
44      D1(1) => D(5),
45      D1(2) => D(6),
46      D1(3) => D(7),
47      S1(0) => S(1),
48      S1(1) => S(0),
49      y => c1
50      );
51  MU2 : mux2 port map (
52      a0 => c0,
53      b0 => c1,
54      s0 => S(2),
55      y => y
```

```
56      );
57
58  end Structural;
```

Codice Componente 1.3: Definizione del componente Bit String Comparator Generic

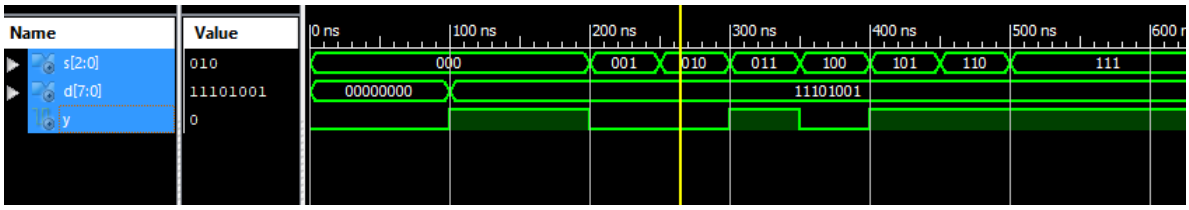
1.3 Simulazione

Nel seguente testbench il multiplexer prende in ingresso la stringa D di 8 bit. Ciò ci permette di osservare, al variare del segnale di selezione S, tutti i possibili valori di uscita y.

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   ENTITY test_mux IS
4   END test_mux;
5
6   ARCHITECTURE behavior OF test_mux IS
7
8   COMPONENT Mux8
9   PORT(
10    S : IN  std_logic_vector(2 downto 0);
11    D : IN  std_logic_vector(7 downto 0);
12    y : OUT  std_logic
13  );
14  END COMPONENT;
15
16    --Inputs
17    signal S : std_logic_vector(2 downto 0) := (others => '0');
18    signal D : std_logic_vector(7 downto 0) := (others => '0');
19    --Outputs
20    signal y : std_logic;
21
22  BEGIN
23    uut: Mux8 PORT MAP ( S => S, D => D, y => y );
24
25    stim_proc: process
26    begin
27      wait for 100 ns;
28    D <= "11101001";
29    S <= "000";
30    wait for 100 ns;
31    S <= "001";
32    wait for 50 ns;
33    S <= "010";
34    wait for 50 ns;
35    S <= "011";
36    wait for 50 ns;
37    S <= "100";
```

```
38    wait for 50 ns;
39    S <= "101";
40    wait for 50 ns;
41    S <= "110";
42    wait for 50 ns;
43    S <= "111";
44    wait for 50 ns;
45    wait;
46    end process;
47
48  END;
```

La simulazione ISim mostra che il multiplexer seleziona correttamente i valori di output, al variare del segnale di selezione, per la stringa data 11101001.



Capitolo 2

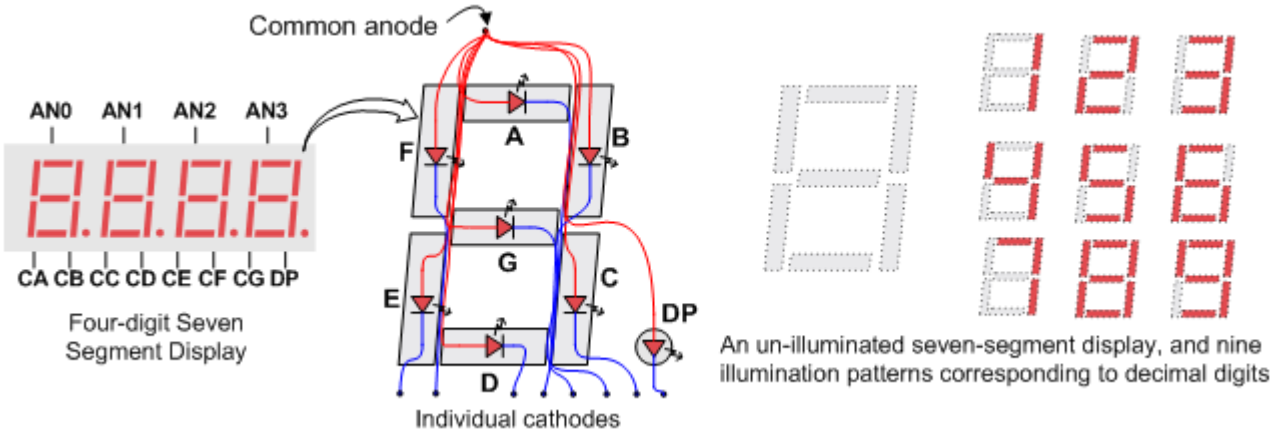
Esercizio 2

2.1 Traccia

Progettare un controller per un display a 7 segmenti che, data una stringa in ingresso di 4 bit che codifica un numero naturale fra 0 e 15, fornisca in uscita i segnali a,b,c,d,e,f,g (in figura) che consentono di rappresentare sul display il numero fornito in ingresso rappresentato nella codifica esadecimale (cifre 0..9 A.. F). Il controller deve essere sintetizzato sulla board e deve utilizzare gli switch per acquisire l'input e una cifra delle 4 disponibili per visualizzare l'output.

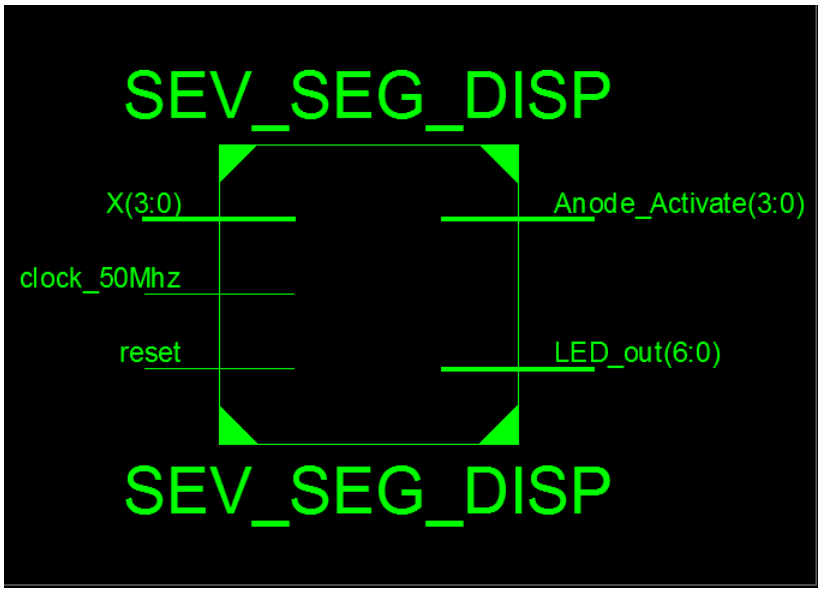
2.2 Soluzione

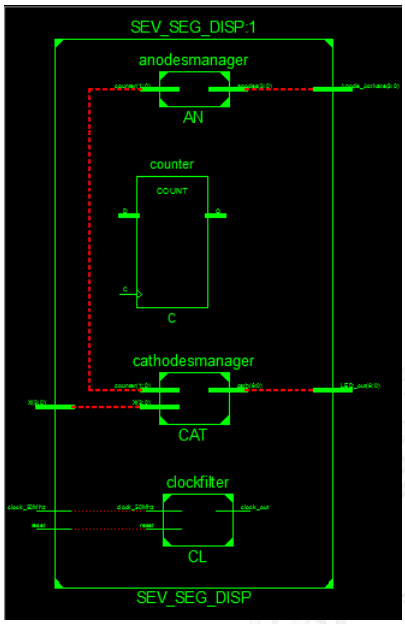
Il componente implementato consente di visualizzare su display a 7 segmenti, lettere e numeri che vanno da 0 a 9 e dalla A alla F.



2.2.1 Schematici

La macchina prende in ingresso il dato su 4 bit da visualizzare, il clock e un segnale di reset. In uscita ci sono i bit relativi agli anodi e ai catodi da accendere.





2.2.2 Codice

Il display è formato da diversi componenti principali : un gestore degli anodi, uno dei catodi, un clock filter e un contatore.

2.2.2.1 Display 7 Segmenti

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.std_logic_unsigned.all;
5
6
```

```
7  entity SEV_SEG_DISP is
8      Port ( clock_50Mhz : in STD_LOGIC;
9             reset : in STD_LOGIC;
10         X: in STD_LOGIC_VECTOR (3 downto 0);
11            Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
12            LED_out : out STD_LOGIC_VECTOR (6 downto 0)
13          );
14
15 end SEV_SEG_DISP;
16 architecture Behavioral of SEV_SEG_DISP is
17
18 Component counter
19 port(
20   clockfx,reset: in std_logic;
21   counter: out std_logic_vector(1 downto 0)
22 );
23 end component;
24
25
26 Component clockfilter
27 port(
28       clock_50Mhz : in STD_LOGIC;
29          reset : in STD_LOGIC;
30       clock_out:out std_logic
31
32 );
33 end component;
34
35 Component anodesmanager
36 port(
37     counter: in std_logic_vector(1 downto 0);
38     anodes: out STD_LOGIC_VECTOR (3 downto 0)
39
40
41 );
42 end component;
43
44
45 Component cathodesmanager
46 port(
47     counter:in std_logic_vector(1 downto 0);
48     X: in STD_LOGIC_VECTOR (3 downto 0);
49     cath : out STD_LOGIC_VECTOR (6 downto 0)
50
51 );
52 end component;
53
54 signal clk: std_logic:='0';
55 signal count: std_logic_vector(1 downto 0):=(others=>'0');
```

```
56
57 begin
58
59 C: counter port map(
60   clockfx=>clk,
61   reset=>reset,
62   counter=>count
63
64 );
65
66
67 AN: anodesmanager port map(
68
69   counter=>count,
70   anodes=>Anode_Activate
71
72
73 );
74
75
76 CAT: cathodesmanager port map(
77     counter=>count,
78     X=>X,
79     cath=> LED_out
80 );
81
82
83
84 CL: clockfilter port map(
85     clock_50Mhz=>clock_50Mhz,
86       reset=>reset,
87     clock_out=>clk
88
89 );
90
91
92 end Behavioral;
```

Codice Componente 2.1: Definizione del Display a 7 segmenti

2.2.2.2 Anodes Manager

Il ruolo del gestore degli anodi è quello di alternare l'attivazione dei 4 anodi disponibili tramite l'analisi dell'ingresso counter (corrispondente all'uscita del contatore). Tale compito è svolto dal process, che nel caso di valori differenti dai precedenti pone l'ingresso degli anodi a "1111".

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
```

```
4
5  entity anodesmanager is
6  port(
7      counter: in std_logic_vector(1 downto 0);
8      anodes: out STD_LOGIC_VECTOR (3 downto 0)
9
10 );
11 end anodesmanager;
12
13 architecture Behavioral of anodesmanager is
14
15 signal Anode_Activate: STD_LOGIC_VECTOR (3 downto 0):=(others=>'0');
16
17 begin
18
19
20 process(counter)
21 begin
22     case counter is
23     when "00" =>
24         Anode_Activate <= "0111";
25     when "01" =>
26         Anode_Activate <= "1011";
27     when "10" =>
28         Anode_Activate <= "1101";
29     when "11" =>
30         Anode_Activate <= "1110";
31     when others =>
32     Anode_Activate <= (others => '1');
33     end case;
34 end process;
35
36
37 anodes<= Anode_Activate;
38
39 end Behavioral;
```

Codice Componente 2.2: Definizione di Anodes Manager

2.2.2.3 Cathodes manager

Mentre il gestore degli anodi tiene attivi questi ultimi secondo la frequenza dettata dal contatore, il gestore dei catodi pone il valore dei bit in ingresso in LED_BCD. Inoltre controlla che, in base al valore di quest'ultimo, venga inserita la giusta combinazione di led accesi/spenti che consenta di visualizzare il dato corretto.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
```

```
4
5  entity cathodesmanager is
6  port(
7      counter:in STD_LOGIC_VECTOR (1 downto 0);
8      X: in STD_LOGIC_VECTOR (3 downto 0);
9      cath : out STD_LOGIC_VECTOR (6 downto 0)
10 );
11 end cathodesmanager;
12
13 architecture Behavioral of cathodesmanager is
14
15 signal LED_BCD: STD_LOGIC_VECTOR (3 downto 0):= (others=>'0');
16 signal LED_out: STD_LOGIC_VECTOR (6 downto 0):= (others=>'0');
17 signal displayed_number: STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
18 constant zero: STD_LOGIC_VECTOR (11 downto 0):= (others=>'0');
19
20 begin
21
22 process(counter,displayed_number)
23 begin
24     case counter is
25     when "00" =>
26         LED_BCD <= displayed_number(15 downto 12);
27     when "01" =>
28         LED_BCD <= displayed_number(11 downto 8);
29     when "10" =>
30         LED_BCD <= displayed_number(7 downto 4);
31     when "11" =>
32         LED_BCD <= displayed_number(3 downto 0);
33     when others =>
34       LED_BCD <= (others => '1');
35     end case;
36 end process;
37
38 displayed_number<= zero & X;
39
40
41
42 process(LED_BCD)
43 begin
44     case LED_BCD is
45     when "0000" => LED_out <= "0000001"; -- "0"
46     when "0001" => LED_out <= "1001111"; -- "1"
47     when "0010" => LED_out <= "0010010"; -- "2"
48     when "0011" => LED_out <= "0000110"; -- "3"
49     when "0100" => LED_out <= "1001100"; -- "4"
50     when "0101" => LED_out <= "0100100"; -- "5"
51     when "0110" => LED_out <= "0100000"; -- "6"
52     when "0111" => LED_out <= "0001111"; -- "7"
```

```
53      when "1000" => LED_out <= "0000000"; -- "8"
54      when "1001" => LED_out <= "0000100"; -- "9"
55      when "1010" => LED_out <= "0001000"; -- A
56      when "1011" => LED_out <= "1100000"; -- b
57      when "1100" => LED_out <= "0110001"; -- C
58      when "1101" => LED_out <= "1000010"; -- d
59      when "1110" => LED_out <= "0110000"; -- E
60      when "1111" => LED_out <= "0111000"; -- F
61      when others => LED_out <= "1111111";
62      end case;
63  end process;
64
65  cath<=LED_out;
66
67  end Behavioral;
```

Codice Componente 2.3: Definizione di Cathodes Manager

2.2.2.4 Counter

Il contatore permette di incrementare il segnale di uscita (counter) di 1, sul fronte di salita del clock clockfx. Il reset pone il contatore a 0.

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.NUMERIC_STD.ALL;
5
6   entity counter is
7   port(
8     clockfx,reset: in std_logic;
9     counter: out std_logic_vector(1 downto 0) -- 2 bit contatore mod 4
10  );
11  end counter;
12
13  architecture Behavioral of counter is
14
15  signal c: std_logic_vector(1 downto 0):=(others=>'0');
16
17  begin
18
19
20  process(clockfx,reset)
21  begin
22     if(reset='1') then
23          c <= (others => '0');
24      elsif(rising_edge(clockfx)) then
25          c <= std_logic_vector(unsigned(c) + 1);
26      end if;
```

```
27 end process;
28
29 counter<=c;
30
31 end Behavioral;
```

Codice Componente 2.4: Definizione del contatore

2.2.2.5 Clock Filter

Il clock filter permette di ridurre la frequenza del clock in ingresso. In questo caso abbiamo ottenuto in uscita un clock di 50 kHz.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity clockfilter is
6  port(
7        clock_50Mhz : in STD_LOGIC;
8              reset : in STD_LOGIC;
9        clock_out:out std_logic
10
11 );
12 end clockfilter;
13
14 architecture Behavioral of clockfilter is
15
16 signal clockfx: std_logic:='0';
17
18 begin
19
20 clkfilter: process(clock_50Mhz, reset)
21 variable counter : integer := 1;
22 begin
23
24  if (reset = '1') then
25   counter := 1;
26   clockfx <= '0';
27  elsif(rising_edge(clock_50Mhz)) then
28   if (counter = 1000) then
29     clockfx <= not clockfx;
30     counter := 1;
31   else
32     counter := counter + 1;
33   end if;
34
35  end if;
36
```

```
 end process;

clock_out<=clockfx;

end Behavioral;
```

Codice Componente 2.5: Definizione del Clock Filter

2.2.3 File UCF

Il file ucf utilizzato per la sintesi sull'FPGA Nexys 2 è riportato di seguito.

```

NET "LED_out<6>" LOC = "L18"; # Bank = 1, Pin name = IO_L10P_1, Type = I/O,
    Sch name = CA
NET "LED_out<5>" LOC = "F18"; # Bank = 1, Pin name = IO_L19P_1, Type = I/O,
    Sch name = CB
NET "LED_out<4>" LOC = "D17"; # Bank = 1, Pin name = IO_L23P_1/HDC, Type =
    DUAL, Sch name = CC
NET "LED_out<3>" LOC = "D16"; # Bank = 1, Pin name = IO_L23N_1/LDC0, Type =
    DUAL, Sch name = CD
NET "LED_out<2>" LOC = "G14"; # Bank = 1, Pin name = IO_L20P_1, Type = I/O,
    Sch name = CE
NET "LED_out<1>" LOC = "J17"; # Bank = 1, Pin name = IO_L13P_1/A6/RHCLK4/
    IRDY1, Type = RHCLK/DUAL, Sch name = CF
NET "LED_out<0>" LOC = "H14"; # Bank = 1, Pin name = IO_L17P_1, Type = I/O,
    Sch name = CG

NET "clock_50Mhz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
     = GCLK, Sch name = GCLK0
NET "Anode_Activate<0>" LOC = "F17"; # Bank = 1, Pin name = IO_L19N_1, Type
    = I/O, Sch name = AN0
NET "Anode_Activate<1>" LOC = "H17"; # Bank = 1, Pin name = IO_L16N_1/A0,
    Type = DUAL, Sch name = AN1
NET "Anode_Activate<2>" LOC = "C18"; # Bank = 1, Pin name = IO_L24P_1/LDC1,
    Type = DUAL, Sch name = AN2
NET "Anode_Activate<3>" LOC = "F15"; # Bank = 1, Pin name = IO_L21P_1, Type
    = I/O, Sch name = AN3

NET "X<0>"        LOC = "G18";    # Sch name = SW0
NET "X<1>"        LOC = "H18";    # Sch name = SW1
NET "X<2>"        LOC = "K18";    # Sch name = SW2
NET "X<3>"        LOC = "K17";    # Sch name = SW3
```

Codice Componente 2.6: ucf display a 7 segmenti

Capitolo 3

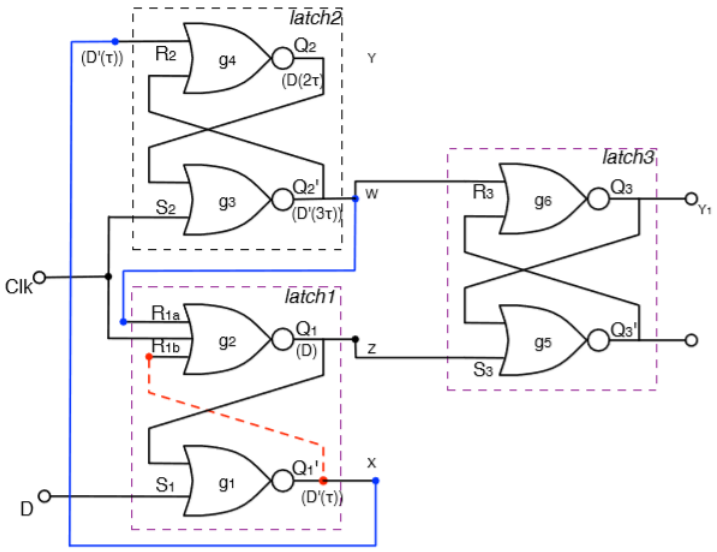
Esercizio 3

3.1 Traccia

Implementare in VHDL e simulare un Flip-flop D edge-triggered e master slave secondo i due differenti modelli visti a lezione (Mazzeo e Sami).

3.2 Flip Flop D Edge Triggered - Prof. Mazzeo

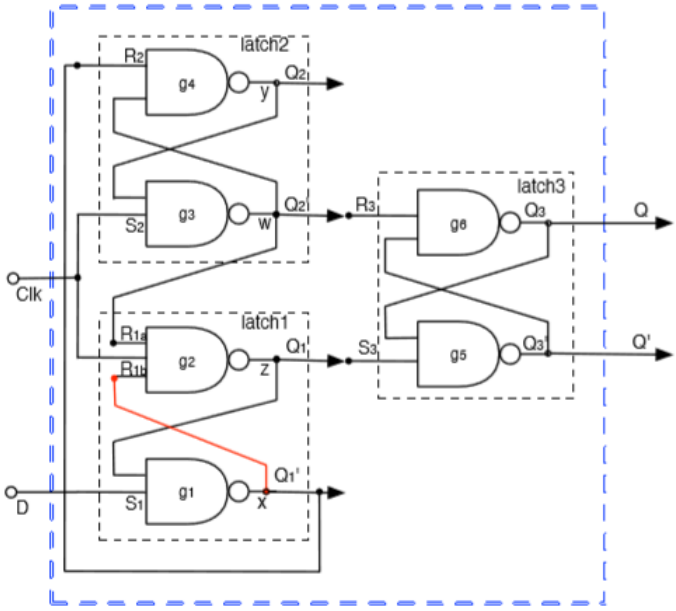
Il Flip Flop D Edge Triggered è un dispositivo bistabile con due ingressi (Dato e Clock) e due uscite (Q e Q negato). Il Flip Flop effettua una transizione di stato in corrispondenza del fronte di salita (o discesa) del segnale di Clock. Esso memorizza il dato ricevuto da D e fornisce attraverso l' uscita Q il dato stesso e attraverso Q negato il segnale invertito. Tale dispositivo è composto in questo caso da 3 Latch RS asincroni fondamentali, che possono essere realizzati sia con porte NAND (edge triggered attivo su fronte di salita) che con porte NOR (edge triggered attivo su fronte di discesa). Entrambe le soluzioni saranno mostrate.



L'immagine mostra il progetto di un Flip Flop D Edge Triggered sul fronte di discesa, realizzato mediante porte NOR. La NOR è una porta logica 0-attiva : presenta una uscita alta solo quando i suoi ingressi sono entrambi bassi, negli altri casi fornisce solo una uscita bassa.

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

Si riporta anche la versione del dispositivo su fronte di salita realizzato con porte NAND.

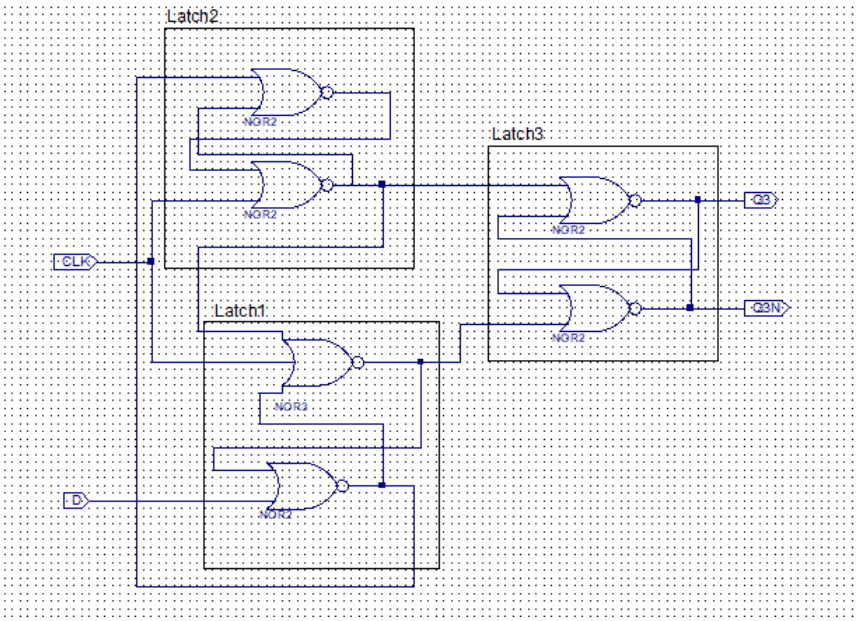


La porta NAND è 1-attiva. Essa ci consente di ottenere commutazione del dato sul fronte di salita del clock.

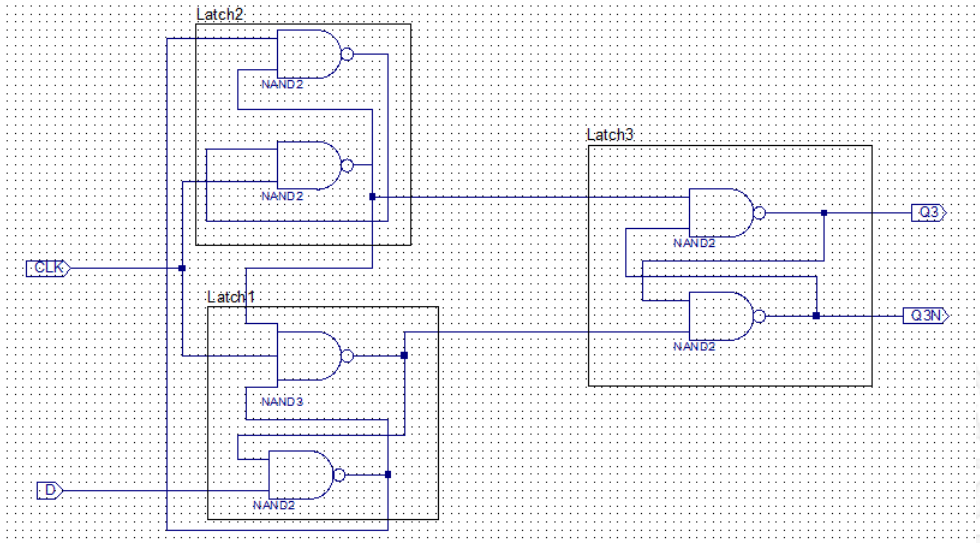
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

3.2.1 Schematici

Il flip flop edge triggered su fronte di discesa (così come quello su fronte di salita) è stato sviluppato col tool grafico di Xilnix ISE.



Segue implementazione con porte NAND della sua versione sul fronte di salita:



3.2.2 Codice

Il codice HDL functional model è autogenerato da Xilnix ISE e implementa la soluzione utilizzando un approccio di tipo strutturale

3.2.2.1 Flip Flop D Edge Triggered sul fronte di discesa

```
1
2 library ieee;
3 use ieee.std_logic_1164.ALL;
4 use ieee.numeric_std.ALL;
5 library UNISIM;
```

```
6  use UNISIM.Vcomponents.ALL;
7
8  entity FFD is
9    port ( CLK : in    std_logic;
10          D   : in    std_logic;
11          Q3  : out   std_logic;
12          Q3N : out   std_logic);
13 end FFD;
14
15 architecture BEHAVIORAL of FFD is
16    attribute BOX_TYPE   : string ;
17    signal Q1        : std_logic;
18    signal Q1N       : std_logic;
19    signal Q2        : std_logic;
20    signal Q2N       : std_logic;
21    signal Q3_DUMMY  : std_logic;
22    signal Q3N_DUMMY : std_logic;
23
24 component NOR2
25       port (
26         I0 : in    std_logic;
27               I1 : in    std_logic;
28               O  : out   std_logic);
29    end component;
30
31    attribute BOX_TYPE of NOR2 : component is "BLACK_BOX";
32        component NOR3       port (
33         I0 : in    std_logic;
34               I1 : in    std_logic;
35               I2 : in    std_logic;
36               O  : out   std_logic);
37    end component;
38
39    attribute BOX_TYPE of NOR3 : component is "BLACK_BOX";
40     begin
41   Q3 <= Q3_DUMMY;
42     Q3N <= Q3N_DUMMY;
43
44      g1 : NOR2
45       port map (I0=>D,
46                 I1=>Q1,
47                 O=>Q1N);
48      g2 : NOR3
49       port map (I0=>Q1N,
50                 I1=>CLK,
51                 I2=>Q2N,
52                 O=>Q1);
53      g3 : NOR2
54       port map (I0=>CLK,
```

```
55                 I1=>Q2,
56                 O=>Q2N);
57      g4 : NOR2
58       port map (I0=>Q2N,
59                 I1=>Q1N,
60                 O=>Q2);
61      g5 : NOR2
62       port map (I0=>Q1,
63                 I1=>Q3_DUMMY,
64                 O=>Q3N_DUMMY);
65      g6 : NOR2
66       port map (I0=>Q3N_DUMMY,
67                 I1=>Q2N,
68                 O=>Q3_DUMMY);
69
70
71  end BEHAVIORAL;
```

Codice Componente 3.1: Definizione del Flip Flop D Edge Triggered sul fronte di discesa

3.2.2.2 Flip Flop D Edge Triggered sul fronte di salita

```
1
2   library ieee;
3   use ieee.std_logic_1164.ALL;
4   use ieee.numeric_std.ALL;
5   library UNISIM;
6   use UNISIM.Vcomponents.ALL;
7
8   entity FFDMazzeoFronteSalita is
9      port ( CLK : in    std_logic;
10            D   : in    std_logic;
11            Q3  : out   std_logic;
12            Q3N : out   std_logic);
13  end FFDMazzeoFronteSalita;
14
15  architecture BEHAVIORAL of FFDMazzeoFronteSalita is
16     attribute BOX_TYPE   : string ;
17     signal Q1        : std_logic;
18     signal Q1N       : std_logic;
19     signal Q2        : std_logic;
20     signal Q2N       : std_logic;
21     signal Q3_DUMMY  : std_logic;
22     signal Q3N_DUMMY : std_logic;
23
24     component NAND2
25        port (  I0 : in    std_logic;
26                I1 : in    std_logic;
```

```
27              O  : out   std_logic);
28    end component;
29
30    attribute BOX_TYPE of NAND2 : component is "BLACK_BOX";
31    component NAND3
32       port ( I0 : in    std_logic;
33              I1 : in    std_logic;
34              I2 : in    std_logic;
35              O  : out   std_logic);
36   end component;
37
38   attribute BOX_TYPE of NAND3 : component is "BLACK_BOX";
39
40   begin
41   Q3 <= Q3_DUMMY;
42   Q3N <= Q3N_DUMMY;
43
44    g1 : NAND2
45       port map (I0=>D,
46                 I1=>Q1,
47                 O=>Q1N);
48   g2 : NAND3
49       port map (I0=>Q1N,
50                 I1=>CLK,
51                 I2=>Q2N,
52                 O=>Q1);
53      g3 : NAND2
54       port map (I0=>CLK,
55                 I1=>Q2,
56                 O=>Q2N);
57     g4 : NAND2
58      port map (I0=>Q2N,
59                I1=>Q1N,
60                O=>Q2);
61      g5 : NAND2
62      port map (I0=>Q1,
63                I1=>Q3_DUMMY,
64                O=>Q3N_DUMMY);
65      g6 : NAND2
66      port map (I0=>Q3N_DUMMY,
67                I1=>Q2N,
68                O=>Q3_DUMMY);
69
70
71 end BEHAVIORAL;
```

Codice Componente 3.2: Definizione del Flip Flop D Edge Triggered sul fronte di salita

3.3 Simulazione

3.3.0.1 TestBench Flip Flop Edge Triggered

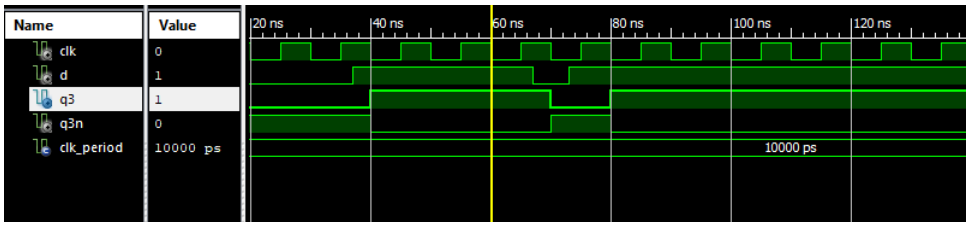
Il seguente codice è valido per entrambe le implementazioni, dunque ne viene riportato solamente uno.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY FFD_FFD_sch_tb1 IS
END FFD_FFD_sch_tb1;

ARCHITECTURE behavior OF FFD_FFD_sch_tb1 IS

    COMPONENT FFD
    PORT(
         CLK : IN  std_logic;
         D : IN  std_logic;
         Q3 : OUT  std_logic;
         Q3N : OUT  std_logic
       );
    END COMPONENT;

   signal CLK : std_logic := '0';
   signal D : std_logic := '0';

   signal Q3 : std_logic;
   signal Q3N : std_logic;

   constant CLK_period : time := 10 ns;
  BEGIN
     uut: FFD PORT MAP (
          CLK => CLK, D => D, Q3 => Q3, Q3N => Q3N );

   CLK_process :process
   begin
    CLK <= '0';
    wait for CLK_period/2;
    CLK <= '1';
    wait for CLK_period/2;
   end process;

   stim_proc: process

    begin
  wait for 20 ns;

  D<= '0', '1' after 17 ns,'0' after 47 ns, '1' after 53 ns;
  wait;
```

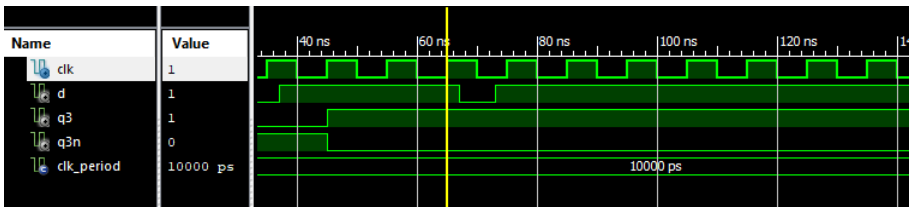
```
44
45      end process;
46  END;
```

Codice Componente 3.3: Definizione del testbench per ffd Mazzeo

Nell'implementazione con le porte NOR, vediamo come l'uscita commuta soltanto in occasione del fronte di discesa del clock:

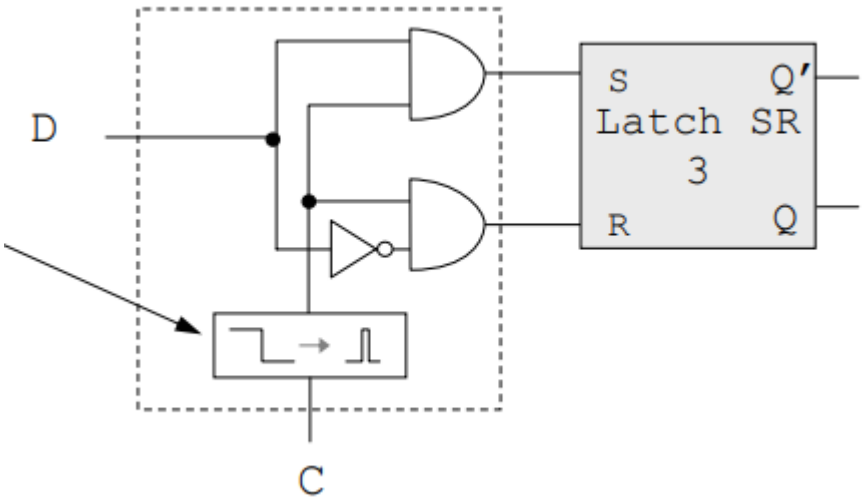


Nell'implementazione con le porte NAND, vediamo come l'uscita commuta soltanto in occasione del fronte di salita del clock:



3.4 Flip Flop D Edge Triggered - Prof. Sami

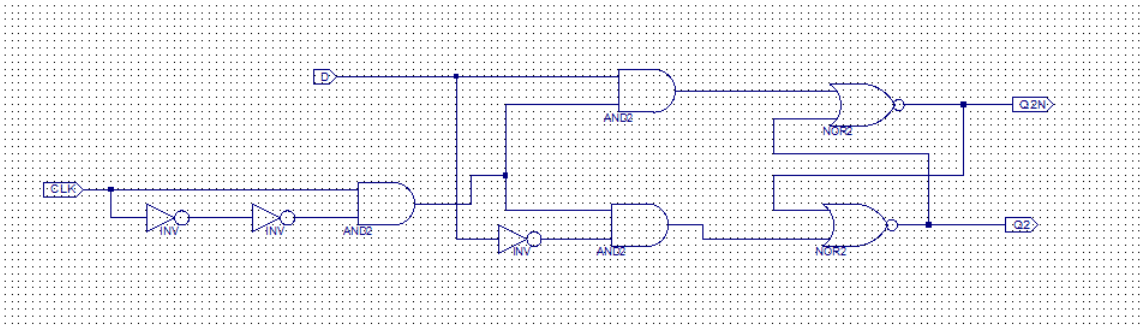
Le dispense di Sami propongono un approccio differente da quelle del paragrafo precedente. Viene utilizzato un blocco di controllo direttamente connesso a un semplice Latch SR e si considera la derivata del segnale di clock, generando un impulso in corrispondenza del fronte di salita o di discesa.

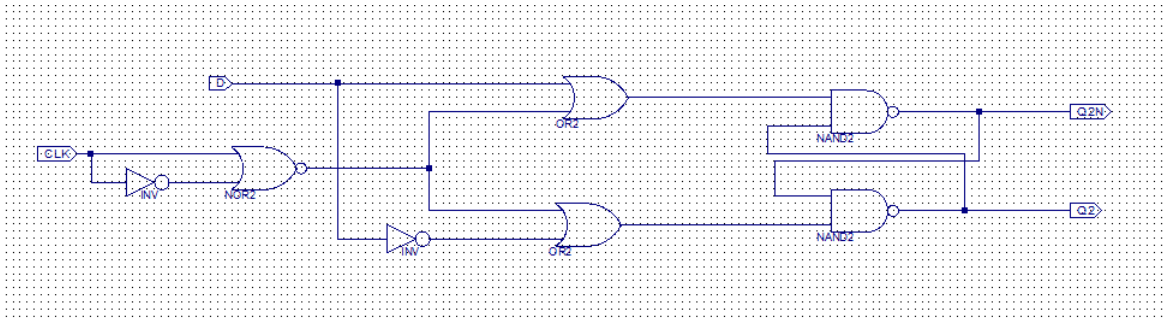


Il latch SR verrà realizzato con porte NOR per il fronte di salita. Per simulare la transizione di stato sul fronte di discesa bisogna fare ulteriori accorgimenti : le due porte AND del blocco di sinistra devono essere sostituite con delle porte OR, mentre il Latch SR va realizzato con le porte NAND.

3.4.1 Schematici

Il primo schematico è relativo al flip flop D Edge triggered sul fronte di salita, mentre il secondo commuta sul fronte di discesa.





3.4.2 Codice

Il codice HDL functional model è autogenerato da Xilnix ISE e implementa la soluzione utilizzando un approccio di tipo strutturale.

3.4.2.1 Flip Flop D Edge Triggered sul fronte di salita

```
1
2  library ieee;
3  use ieee.std_logic_1164.ALL;
4  use ieee.numeric_std.ALL;
5  library UNISIM;
6  use UNISIM.Vcomponents.ALL;
7
8  entity SamiFronteSalita is
9    port ( CLK : in    std_logic;
10          D   : in    std_logic;
11          Q2  : out   std_logic;
```

```
12            Q2N : out   std_logic);
13 end SamiFronteSalita;
14
15 architecture BEHAVIORAL of SamiFronteSalita is
16    attribute BOX_TYPE   : string ;
17    signal Q         : std_logic;
18    signal Q1        : std_logic;
19    signal XLXN_5    : std_logic;
20    signal XLXN_42   : std_logic;
21    signal XLXN_68   : std_logic;
22    signal XLXN_69   : std_logic;
23    signal Q2_DUMMY  : std_logic;
24    signal Q2N_DUMMY : std_logic;
25
26    component AND2
27     port ( I0 : in    std_logic;
28            I1 : in    std_logic;
29            O  : out   std_logic);
30   end component;
31
32   attribute BOX_TYPE of AND2 : component is "BLACK_BOX";
33      component NOR2
34      port ( I0 : in    std_logic;
35             I1 : in    std_logic;
36             O  : out   std_logic);
37  end component;
38   attribute BOX_TYPE of NOR2 : component is "BLACK_BOX";
39
40     component INV
41     port ( I : in    std_logic;
42            O : out   std_logic);
43    end component;
44
45    attribute BOX_TYPE of INV : component is "BLACK_BOX";
46    begin
47
48    Q2 <= Q2_DUMMY;
49    Q2N <= Q2N_DUMMY;
50
51      a1 : AND2
52      port map (I0=>XLXN_69,
53                I1=>CLK,
54                O=>XLXN_42);
55      a2 : AND2
56      port map (I0=>XLXN_5,
57                I1=>XLXN_42,
58                O=>Q1);
59      a3 : AND2
60      port map (I0=>XLXN_42,
```

```
61                I1=>D,
62                O=>Q);
63      g2 : NOR2
64      port map (I0=>Q1,
65                I1=>Q2N_DUMMY,
66                O=>Q2_DUMMY);
67    g3 : NOR2
68      port map (I0=>Q2_DUMMY,
69                I1=>Q,
70                O=>Q2N_DUMMY);
71      INV1 : INV
72      port map (I=>CLK,
73                O=>XLXN_68);
74      INV2 : INV
75      port map (I=>XLXN_68,
76                O=>XLXN_69);
77   INV3 : INV
78      port map (I=>D,
79                O=>XLXN_5);
80
81
82 end BEHAVIORAL;
```

Codice Componente 3.4: Definizione del Flip Flop D Edge Triggered sul fronte di salita Sami

3.4.2.2 Flip Flop D Edge Triggered sul fronte di discesa

```
1  library ieee;
2  use ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.ALL;
4  library UNISIM;
5  use UNISIM.Vcomponents.ALL;
6
7  entity SamiFronteDiscesa is
8     port  ( CLK : in    std_logic;
9             D   : in    std_logic;
10            Q2  : out   std_logic;
11            Q2N : out   std_logic);
12 end SamiFronteDiscesa;
13
14 architecture BEHAVIORAL of SamiFronteDiscesa is
15    attribute BOX_TYPE   : string ;
16    signal Q         : std_logic;
17    signal Q1        : std_logic;
18    signal XLXN_10   : std_logic;
19    signal XLXN_23   : std_logic;
20    signal XLXN_26   : std_logic;
21    signal Q2_DUMMY  : std_logic;
```

```
22    signal Q2N_DUMMY : std_logic;
23    component NAND2
24       port ( I0 : in    std_logic;
25              I1 : in    std_logic;
26              O  : out   std_logic);
27    end component;
28
29 attribute BOX_TYPE of NAND2 : component is "BLACK_BOX";
30    component NOR2       port ( I0 : in    std_logic;
31            I1 : in    std_logic;
32               O  : out   std_logic);
33    end component;
34
35 attribute BOX_TYPE of NOR2 : component is "BLACK_BOX";
36    component OR2
37       port ( I0 : in    std_logic;
38              I1 : in    std_logic;
39              O  : out   std_logic);
40    end component;
41
42 attribute BOX_TYPE of OR2 : component is "BLACK_BOX";
43
44    component INV
45       port ( I : in    std_logic;
46              O : out   std_logic);
47    end component;
48
49 attribute BOX_TYPE of INV : component is "BLACK_BOX";
50
51 begin
52
53    Q2 <= Q2_DUMMY;
54    Q2N <= Q2N_DUMMY;
55
56       a1 : NAND2
57       port map (I0=>Q1,
58                 I1=>Q2N_DUMMY,
59                 O=>Q2_DUMMY);
60       a2 : NAND2
61       port map (I0=>Q2_DUMMY,
62                 I1=>Q,
63                O=>Q2N_DUMMY);
64       g1 : NOR2
65       port map (I0=>XLXN_26,
66                 I1=>CLK,
67                 O=>XLXN_23);
68       g2 : OR2
69       port map (I0=>XLXN_10,
70                 I1=>XLXN_23,
```

```
71                  O=>Q1);
72       g3 : OR2
73       port map (I0=>XLXN_23,
74                 I1=>D,
75                 O=>Q);
76       INV1 : INV
77    port map (I=>CLK,
78                 O=>XLXN_26);
79       INV2 : INV
80       port map (I=>D,
81                 O=>XLXN_10);
82
83 end BEHAVIORAL;
```

Codice Componente 3.5: Definizione del Flip Flop D Edge Triggered sul fronte di discesa Sami

3.5 Simulazione

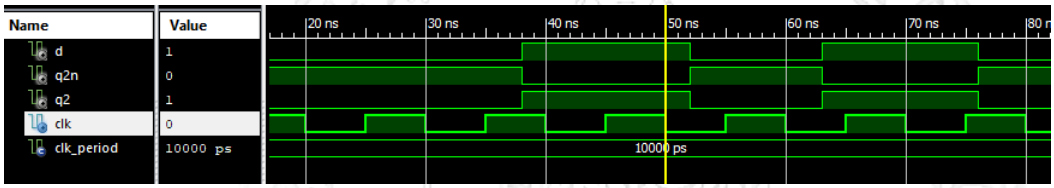
La seguente simulazione behavioral mostra come cambia lo stato del flip flop, a seconda della variazione del dato D, in corrispondenza del fronte del clock. Un unico testbench è presentato per entrambe le soluzioni (rising e falling edge).

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4  USE ieee.numeric_std.ALL;
5  LIBRARY UNISIM;
6  USE UNISIM.Vcomponents.ALL;
7  ENTITY SamiFronteDiscesa_SamiFronteDiscesa_sch_tb IS
8  END SamiFronteDiscesa_SamiFronteDiscesa_sch_tb;
9
10 ARCHITECTURE behavioral OF SamiFronteDiscesa_SamiFronteDiscesa_sch_tb IS
11
12    COMPONENT SamiFronteDiscesa
13    PORT(    D : IN STD_LOGIC;
14             Q2N : OUT STD_LOGIC;
15             Q2 : OUT STD_LOGIC;
16             CLK : IN STD_LOGIC);
17    END COMPONENT;
18
19    SIGNAL D :STD_LOGIC :='0';
20    SIGNAL Q2N :STD_LOGIC;
21    SIGNAL Q2 :STD_LOGIC;
22    SIGNAL CLK :STD_LOGIC;
23    constant CLK_period : time := 10 ns;
24
25
26 BEGIN
```

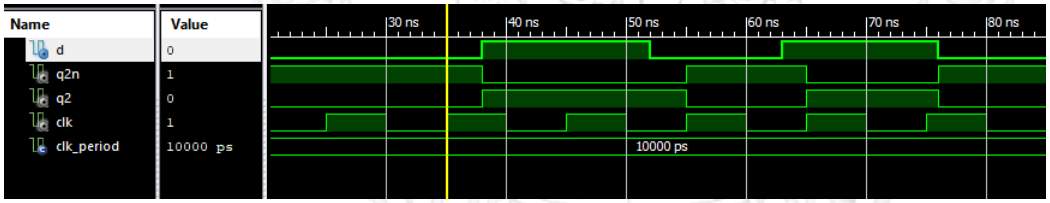
```
27    UUT: SamiFronteDiscesa PORT MAP(
28      D => D,
29     Q2N => Q2N,
30     Q2 => Q2,
31     CLK => CLK
32     );
33
34
35    CLK_process :process
36    BEGIN
37     CLK<= '0';
38     wait for CLK_period/2;
39     CLK <= '1';
40     wait for CLK_period/2;
41    end process;
42
43    tb : PROCESS
44    BEGIN
45   wait for 10 ns;
46
47   D<= '0', '1' after 28 ns,'0' after 42 ns, '1' after 53 ns,'0' after 66 ns;
48
49     WAIT;
50    END PROCESS;
51
52 END;
```

Codice Componente 3.6: Definizione del testbench per ffd sami etd

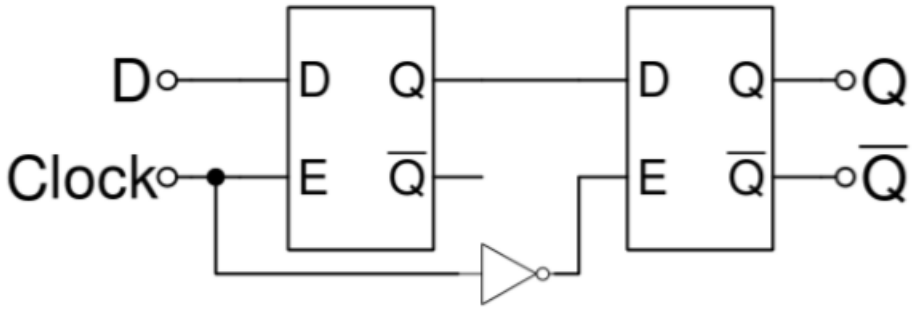
Simulazione behavioral del flip flop edge triggered su fronte di discesa:



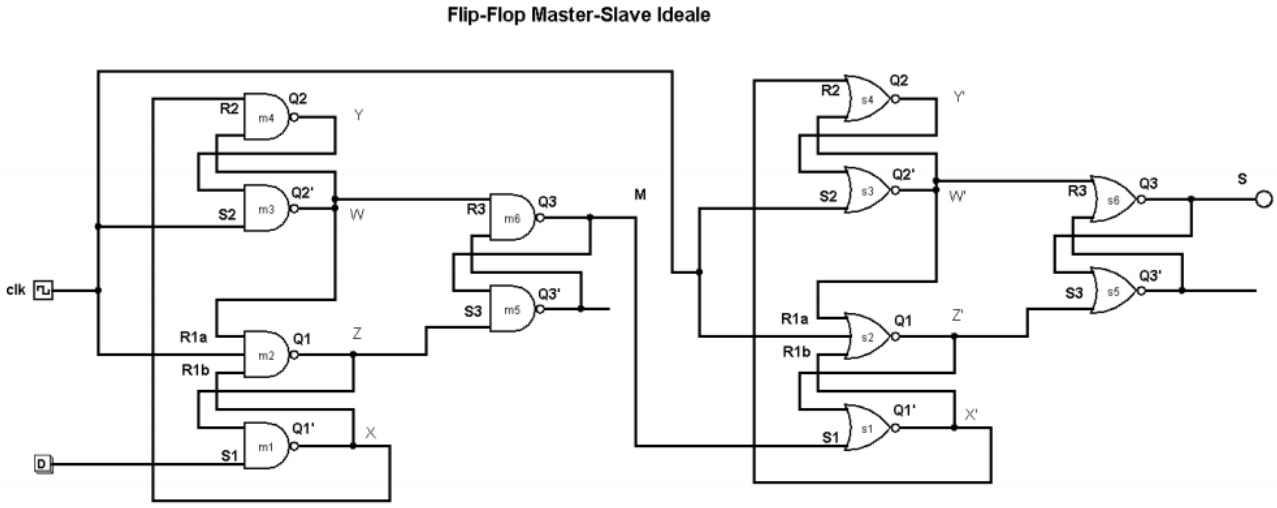
Simulazione behavioral del flip flop edge triggered su fronte di salita:



3.6 Flip Flop D Master Slave - Prof. Antonino Mazzeo



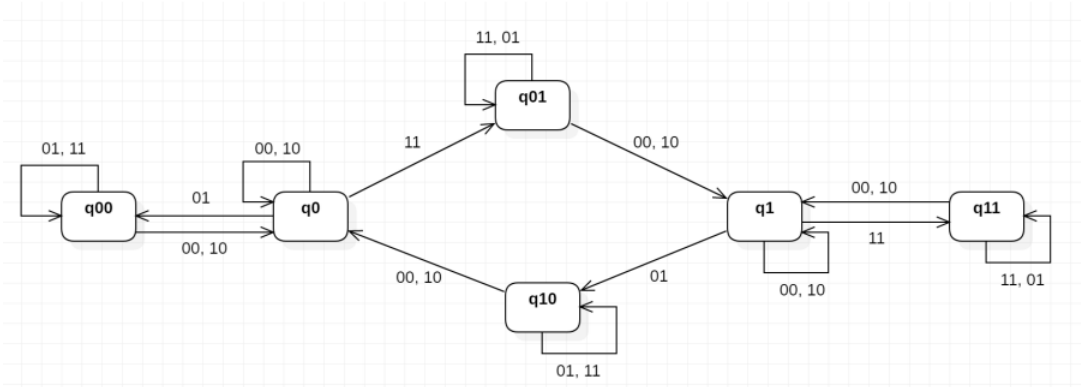
La soluzione prevede di collegare in cascata due flip flop D: un master, attivo sul fronte di salita del clock, e uno slave, attivo sul fronte di discesa. Quando il clock si alza, il master campiona il dato D in ingresso e lo presenta in uscita dopo un certo ritardo;il flip flop slave riceve in ingresso il dato campionato dal master ma non campiona e presenta il dato fino a quando il segnale di clock non si abbassa.



Di seguito è possibile osservare l'automa a stati finiti del flip flop master-slave ideale, i cui stati sono:

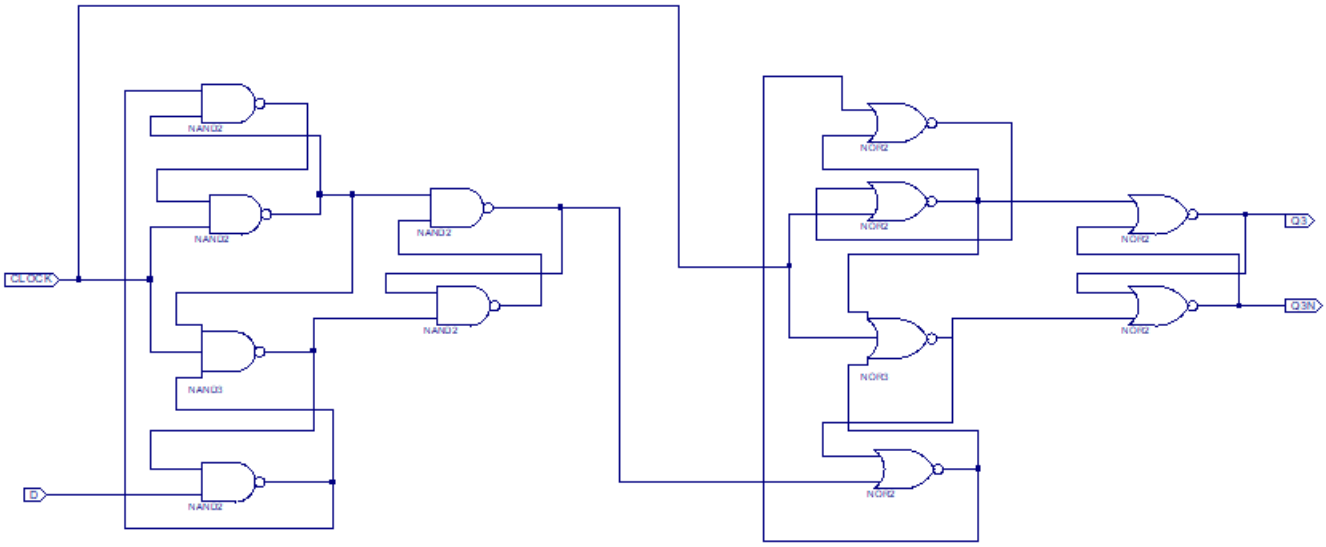
- q0 e q1 in cui il flip flop presenta rispettivamente 0 e 1;
- q00 e q11 in cui, all'alzarsi del segnale di abilitazione, campiona lo stesso dato che sta già presentando;

- q01 e q10 in cui è stato campionato un valore diverso da quello che sta correntemente presentando.



3.6.1 Schematici

Si è scelto di sviluppare il progetto direttamente tramite schematic in Xilnix ISE, usando il tool grafico e collegando manualmente le porte logiche.



Come si può osservare, la struttura deve essre realizzata con un flip flop edge triggered a porte NAND per il master e uno a porte NOR per lo slave, per poter campionare sul fronte di salita e presentare sul fronte di discesa del segnale di abilitazione.

3.6.2 Codice

Il codice HDL functional model è autogenerato da Xilnix ISE e implementa la soluzione utilizzando un approccio di tipo strutturale.

```
1 library ieee;
```

```
2  use ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.ALL;
4  library UNISIM;
5  use UNISIM.Vcomponents.ALL;
6
7  entity MazzeoMS is
8     port ( CLOCK : in    std_logic;
9            D     : in    std_logic;
10           Q3    : out   std_logic;
11           Q3N   : out   std_logic);
12 end MazzeoMS;
13
14 architecture Structural of MazzeoMS is
15    attribute BOX_TYPE   : string ;
16    signal M          : std_logic;
17    signal W          : std_logic;
18    signal W1         : std_logic;
19    signal X          : std_logic;
20    signal XLXN_86    : std_logic;
21    signal X1         : std_logic;
22    signal Y          : std_logic;
23    signal Y1         : std_logic;
24    signal Z          : std_logic;
25    signal Z1         : std_logic;
26    signal Q3_DUMMY   : std_logic;
27    signal Q3N_DUMMY  : std_logic;
28    component NAND2
29       port ( I0 : in    std_logic;
30              I1 : in    std_logic;
31              O  : out   std_logic);
32    end component;
33    attribute BOX_TYPE of NAND2 : component is "BLACK_BOX";
34        component NAND3
35       port ( I0 : in    std_logic;
36              I1 : in    std_logic;
37              I2 : in    std_logic;
38              O  : out   std_logic);
39    end component;
40    attribute BOX_TYPE of NAND3 : component is "BLACK_BOX";
41       component NOR2
42       port ( I0 : in    std_logic;
43              I1 : in    std_logic;
44              O  : out   std_logic);
45    end component;
46    attribute BOX_TYPE of NOR2 : component is "BLACK_BOX";
47        component NOR3
48       port ( I0 : in    std_logic;
49              I1 : in    std_logic;
50              I2 : in    std_logic;
```

```
51              O   : out    std_logic);
52  end component;
53  attribute BOX_TYPE of NOR3 : component is "BLACK_BOX";
54    begin
55  Q3 <= Q3_DUMMY;
56   Q3N <= Q3N_DUMMY;
57
58    m1 : NAND2
59      port map (I0=>D,
60                I1=>Z,
61               O=>X);
62    m2 : NAND3
63      port map (I0=>X,
64                I1=>CLOCK,
65                I2=>W,
66                O=>Z);
67       m3 : NAND2
68      port map( I0=>CLOCK,
69                I1=>Y,
70                O=>W);
71       m4 : NAND2
72      port map (I0=>W,
73                I1=>X,
74                O=>Y);
75       m5 : NAND2
76      port map (I0=>Z,
77                I1=>M,
78                O=>XLXN_86);
79       m6 : NAND2
80      port map (I0=>XLXN_86,
81                I1=>W,
82                O=>M);
83       s1 : NOR2
84      port map (I0=>M,
85                I1=>Z1,
86                O=>X1);
87       s2 : NOR3
88      port map (I0=>X1,
89                I1=>CLOCK,
90                I2=>W1,
91                O=>Z1);
92       s3 : NOR2
93      port map (I0=>CLOCK,
94                I1=>Y1,
95                O=>W1);
96       s4 : NOR2
97      port map (I0=>W1,
98                I1=>X1,
99                O=>Y1);
```

```
100         s5 : NOR2
101       port map (I0=>Z1,
102                 I1=>Q3_DUMMY,
103                 O=>Q3N_DUMMY);
104         s6 : NOR2
105       port map (I0=>Q3N_DUMMY,
106                 I1=>W1,
107                 O=>Q3_DUMMY);
108     end Structural;
```

Codice Componente 3.7: Definizione del componente Arbitro 2 su 3

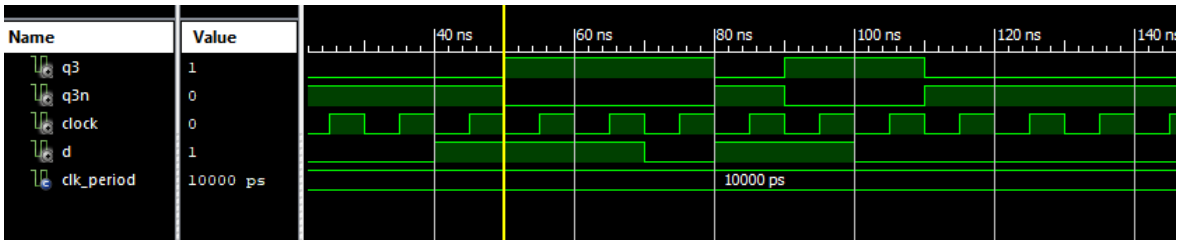
3.7 Simulazione

Il testbench ci conferma quanto già discusso in via teorica. Il flip flop master slave campiona sul fronte di salita e presenta il valore campionato sul fronte di discesa del clock.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  LIBRARY UNISIM;
5  USE UNISIM.Vcomponents.ALL;
6  ENTITY MazzeoMS_MazzeoMS_sch_tb IS END MazzeoMS_MazzeoMS_sch_tb;
7  ARCHITECTURE behavioral OF MazzeoMS_MazzeoMS_sch_tb IS
8
9  COMPONENT MazzeoMS
10    PORT(   CLOCK: IN STD_LOGIC;
11       Q3: OUT STD_LOGIC;
12            Q3N: OUT STD_LOGIC;
13            D: IN STD_LOGIC
14      );
15 END COMPONENT;
16
17    SIGNAL Q3: STD_LOGIC :='0';
18    SIGNAL Q3N: STD_LOGIC;
19    SIGNAL CLOCK: STD_LOGIC;
20    SIGNAL D: STD_LOGIC:='0';
21    constant CLK_period : time := 10 ns;
22
23 BEGIN
24    UUT: MazzeoMS PORT MAP( Q3 => Q3, Q3N => Q3N, CLOCK => CLOCK, D => D );
25
26 CLOCK_process :process
27    begin
28     CLOCK <= '0';
29     wait for CLK_period/2;
30     CLOCK <= '1';
31     wait for CLK_period/2;
```

```
32 end process;
33
34    stim_proc: process
35    begin
36   wait for 20 ns;
37   D<= '0', '1' after 20 ns,'0' after 50 ns, '1' after 60 ns,'0' after 80 ns;
38   wait;
39    end process;
40 END;
```

La simulazione in ISim conferma le aspettative mostrando che al variare di D, l'uscita del flip flop commuta solamente sul fronte di discesa del clock.



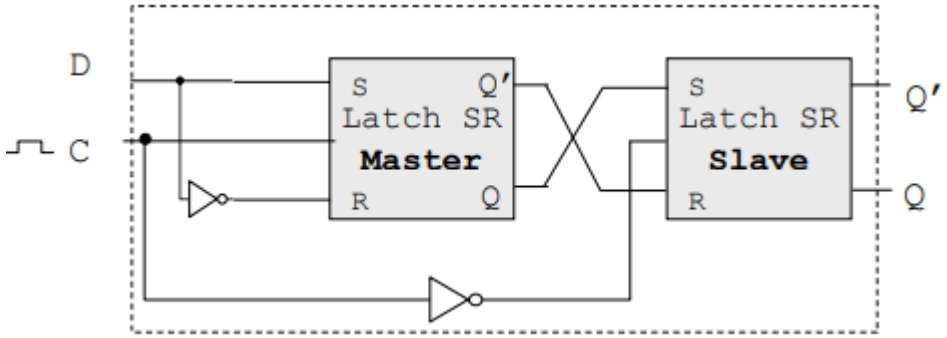
3.8 Flip Flop D Master Slave - Prof. Sami

Il progetto illustrato nelle dispense ci mostra un tipo di implementazione differente da quello visto in precedenza. Innanzitutto esso utilizza due Latch SR (un Master e uno Slave), le cui uscite Q e Q negato del Master sono collegate rispettivamente agli ingressi di Set e Reset dello Slave. Il Latch SR segue la logica retroazionale vista precedentemente nel Flip Flop del Prof. Mazzeo e può essere implementato sia con porte NAND (attivo basso) sia con porte NOR (attivo alto). Nella sua versione attivo alto, che è quella scelta, lo stato di memoria del Latch SR si ottiene quando entrambi gli ingressi sono bassi. La funzione di Reset avviene quando R è alto ed S è basso, come illustrato nella seguente tabella di verità :

S	R
0	0
0	1
1	0
1	1

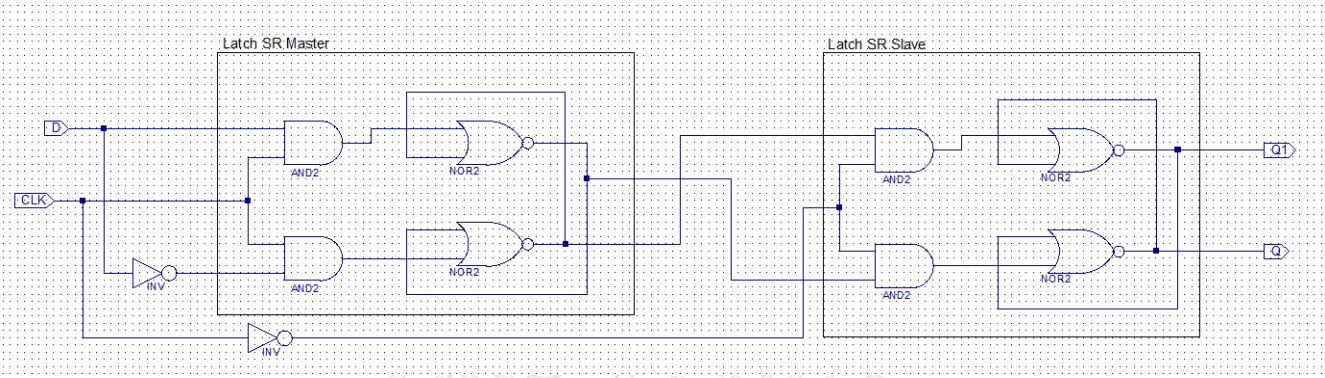
"00" Memorizzazione. "01" Reset. "10" Set. "11" non ammesso.

Partendo da questo si può costruire un flip flop D Master Slave :



3.8.1 Schematici

Lo schematico in figura sottostante, realizzato col tool grafico di Xilnix, mostra i collegamenti tra i dispositivi e in particolare i latch che sono costruiti nella versione attivo alta (quindi con la parte di destra formata dal Latch SR asincrono e le due porte AND).



3.8.2 Codice

Il codice HDL functional model è autogenerato da Xilnix ISE e implementa la soluzione utilizzando un approccio di tipo strutturale.

```
1  library ieee;
2  use ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.ALL;
4  library UNISIM;
5  use UNISIM.Vcomponents.ALL;
6
7  entity SamiMasterSlave is
8     port ( CLK : in    std_logic;
9            D   : in    std_logic;
10           Q   : out   std_logic;
11           Q1  : out   std_logic);
12 end SamiMasterSlave;
13
14 architecture BEHAVIORAL of SamiMasterSlave is
15    attribute BOX_TYPE   : string ;
16    signal XLXN_1   : std_logic;
```

```
17    signal XLXN_2   : std_logic;
18    signal XLXN_4   : std_logic;
19    signal XLXN_5   : std_logic;
20    signal XLXN_27  : std_logic;
21    signal XLXN_28  : std_logic;
22    signal XLXN_29  : std_logic;
23    signal XLXN_41  : std_logic;
24    signal Q1_DUMMY : std_logic;
25    signal Q_DUMMY  : std_logic;
26
27    component AND2
28       port (  I0 : in    std_logic;
29               I1 : in    std_logic;
30               O  : out   std_logic);
31    end component;
32
33   attribute BOX_TYPE of AND2 : component is "BLACK_BOX";
34   component NOR2
35     port ( I0 : in    std_logic;
36                I1 : in    std_logic;
37                O  : out   std_logic);
38   end component;
39   attribute BOX_TYPE of NOR2 : component is "BLACK_BOX";
40
41   component INV
42       port (  I : in    std_logic;
43               O : out   std_logic);
44   end component;
45   attribute BOX_TYPE of INV : component is "BLACK_BOX";
46   begin
47    Q <= Q_DUMMY;
48    Q1 <= Q1_DUMMY;
49
50     a1 : AND2
51      port map (I0=>XLXN_27,
52                I1=>CLK,
53                O=>XLXN_4);
54
55       a2 : AND2
56      port map (I0=>CLK,
57                I1=>D,
58                O=>XLXN_5);
59
60       a3 : AND2
61      port map (I0=>XLXN_1,
62                I1=>XLXN_41,
63                O=>XLXN_28);
64
65       a4 : AND2
```

```
66      port map (I0=>XLXN_41,
67                I1=>XLXN_2,
68                O=>XLXN_29);
69
70        g1 : NOR2
71       port map (I0=>XLXN_4,
72                 I1=>XLXN_1,
73                 O=>XLXN_2);
74
75        g2 : NOR2
76       port map (I0=>XLXN_2,
77                 I1=>XLXN_5,
78                 O=>XLXN_1);
79
80        g3 : NOR2
81      port map  (I0=>XLXN_28,
82                 I1=>Q1_DUMMY,
83                 O=>Q_DUMMY);
84
85        g4 : NOR2
86       port map (I0=>Q_DUMMY,
87                 I1=>XLXN_29,
88                 O=>Q1_DUMMY);
89
90        not1 : INV
91       port map (I=>CLK,
92                 O=>XLXN_41);
93
94        not2 : INV
95       port map (I=>D,
96                 O=>XLXN_27);
97
98    end BEHAVIORAL;
```

Codice Componente 3.8: Definizione del componente Arbitro 2 su 3

3.9 Simulazione

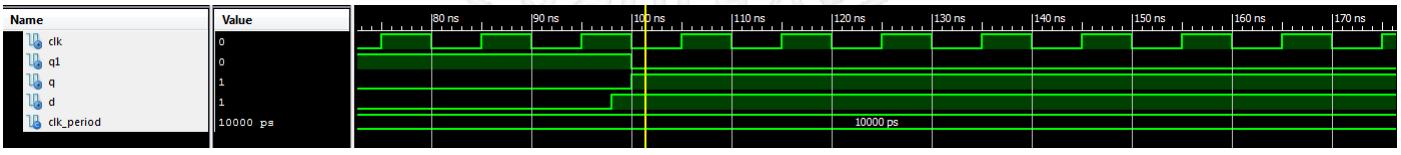
Il codice del testbench è riportato di seguito.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  LIBRARY UNISIM;
5  USE UNISIM.Vcomponents.ALL;
6  ENTITY MasterSlave_MasterSlave_sch_tb IS
7  END MasterSlave_MasterSlave_sch_tb;
8  ARCHITECTURE behavioral OF MasterSlave_MasterSlave_sch_tb IS
```

```
9
10    COMPONENT MasterSlave
11    PORT( CLK  : IN  STD_LOGIC;
12          Q1   : OUT STD_LOGIC;
13          Q : OUT STD_LOGIC;
14          D : IN  STD_LOGIC);
15    END COMPONENT;
16
17    SIGNAL CLK : STD_LOGIC;
18    SIGNAL Q1  : STD_LOGIC;
19    SIGNAL Q : STD_LOGIC;
20    SIGNAL D : STD_LOGIC;
21
22   constant CLK_period : time := 10 ns;
23
24 BEGIN
25
26    UUT: MasterSlave PORT MAP(
27     CLK => CLK,
28     Q1 => Q1,
29     Q => Q,
30     D => D
31    );
32
33   CLK_process :process
34     begin
35       CLK <= '0';
36       wait for CLK_period/2;
37       CLK <= '1';
38       wait for CLK_period/2;
39     end process;
40
41   stim_proc: process
42   begin
43     D <= '0';
44     wait for 98 ns;
45     D <= '1';
46     wait;
47   end process;
48 END;
```

Codice Componente 3.9: Definizione del testbench per SamiMS

I risultati del test sono quelli attesi. Il dato contenuto nel registro commuta al variare di D e soltanto sul fronte di discesa del clock:



Capitolo 4

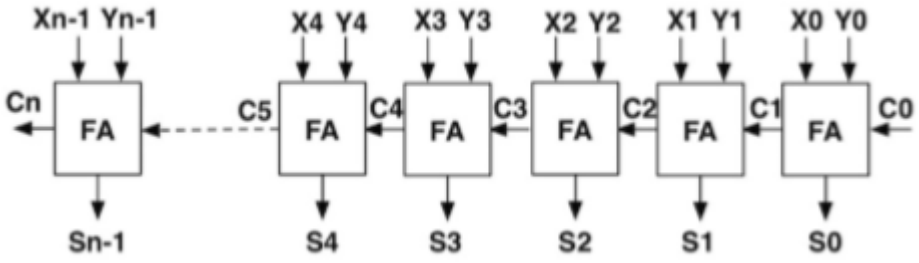
Esercizio 4

4.1 Traccia

Progettare ed implementare in VHDL un sommatore a propagazione dei riporti per stringhe di 4 bit, ed utilizzarlo successivamente per realizzare un sommatore di stringhe di 8 bit. Sintetizzare sulla board il sommatore di stringhe di 8 bit, utilizzando gli switch e i bottoni per inserire le stringhe di input, due cifre del display a 7 segmenti per visualizzare l'output su 8 bit, e un led per segnalare la condizione di overflow. NOTA: i due addendi devono essere acquisiti in due tempi diversi, secondo una tecnica a scelta dello studente.

4.2 Soluzione

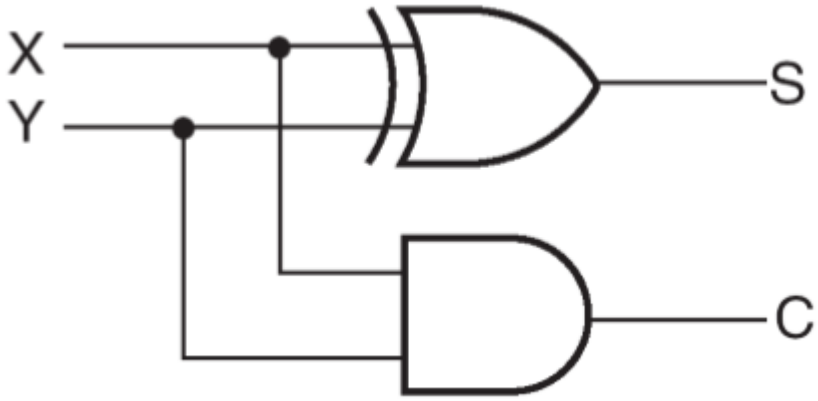
Il sommatore ad 8 bit viene realizzato per composizione di due sommatori a 4 bit. Il sommatore a 4 bit viene realizzato tramite cascata di full adder (4 in questo caso). Il singolo full adder presenta due ingressi per gli operandi da sommare e uno per il valore di riporto. Quest'ultimo nel primo full adder della cascata è normalmente 0. Le uscite sono la somma dei due operandi e il riporto (0,1) che viene collegato in ingresso al full adder immediatamente successivo.



Per realizzare il full adder si è fatto uso di due half adder, le cui uscite vengono messe in xor per ottenere il bit di somma.

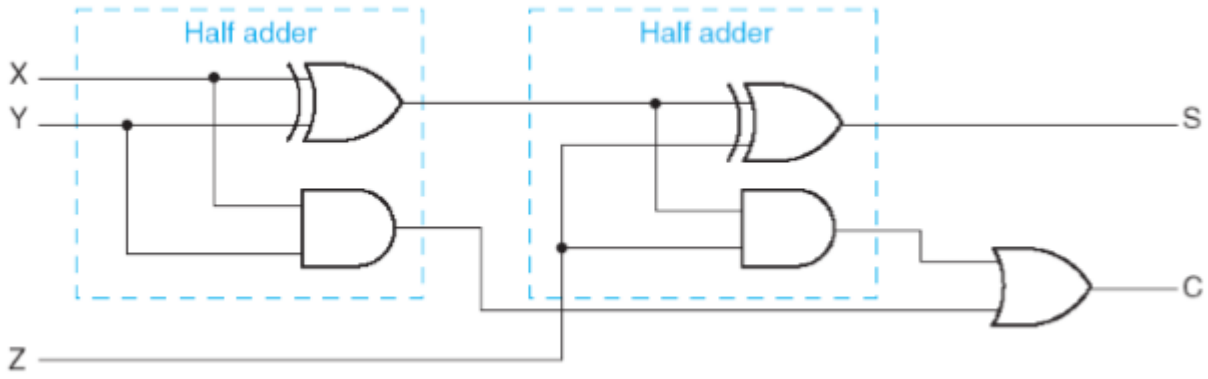
Per quest'ultimo componente, si riporta la tabella di verità (input a sinistra e output a destra):

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1



Di seguito si riporta la tabella di verità del full adder, realizzato come composizione di due half adder. X,Y e Z sono gli ingressi; S è il bit di somma e C quello di riporto.

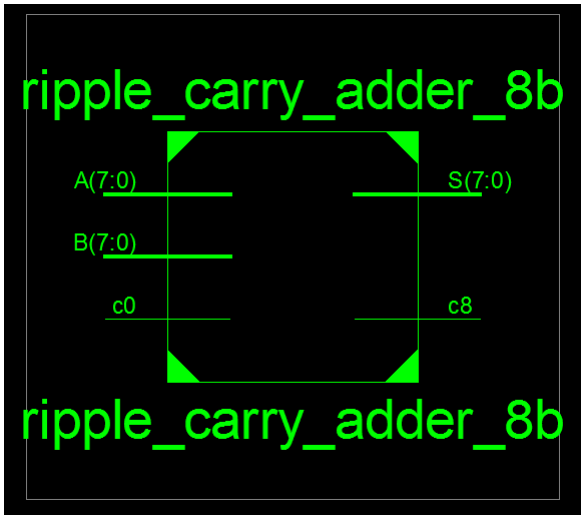
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	1	1	1



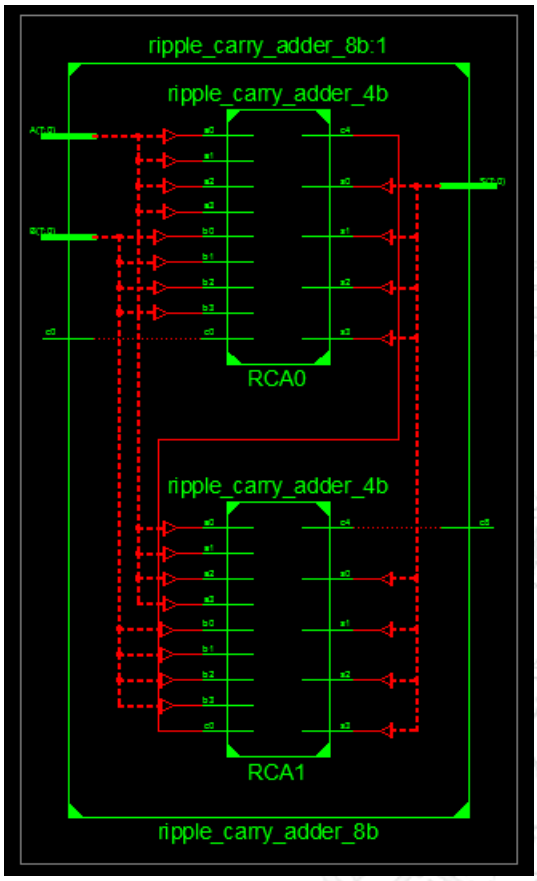
4.2.1 Schematici

Gli schematici sono mostrati a partire da quello del full adder a 8 bit.

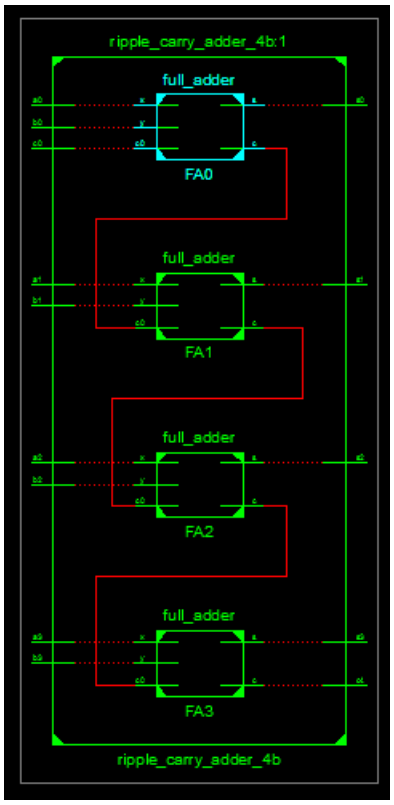
(Ripple Carry Adder 8 bit)



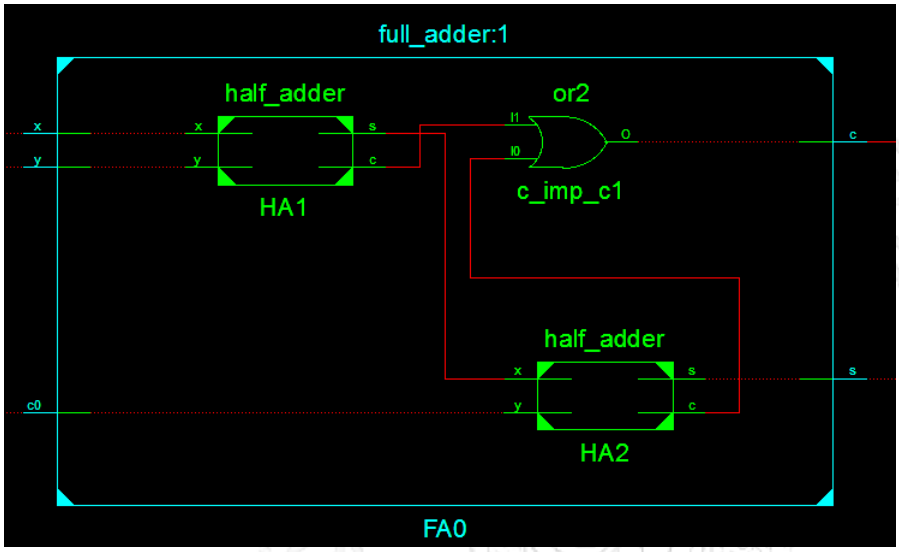
(Ripple Carry Adder 8 bit - Composizione)



(Ripple Carry Adder 4 bit - Composizione)



(Full Adder - Composizione)



4.2.2 Codice

4.2.2.1 Ripple Carry Adder 8 bit

Tramite descrizione strutturale si usano i due sommatori a propagazione di riporti di 4 bit per crearne uno da 8 bit. Tale componente può essere usato per comporre eventualmente sommatori più grandi (16, 32, 64 bit).

L'ingresso c0 del sommatore a 8 bit corrisponderà al riporto del primo da 4 bit ed è '0' in questo caso.

```
entity ripple_carry_adder_8b is

PORT (
      A : in  std_logic_vector (7 downto 0);
      B : in  std_logic_vector(7 downto 0);
      c0 : in std_logic;
      c8 : out std_logic;
      S: out std_logic_vector (7 downto 0)
      );

end ripple_carry_adder_8b;

architecture Structural of ripple_carry_adder_8b is

Component ripple_carry_adder_4b
  PORT (
      a0, a1, a2, a3 : in  std_logic;
      b0, b1, b2, b3 : in  std_logic;
      c0 : in std_logic;
      c4 : out std_logic;
      s0, s1, s2, s3: out std_logic  );
end Component;

signal c : std_logic;

begin
RCA0 : ripple_carry_adder_4b port map (
    a0 => A(0),
    a1 => A(1),
    a2 => A(2),
    a3 => A(3),
    b0 => B(0),
    b1 => B(1),
    b2 => B(2),
    b3 => B(3),
    c0 => c0,
    s0 => S(0),
    s1 => S(1),
    s2 => S(2),
    s3 => S(3),
    c4 => c
    );
RCA1 : ripple_carry_adder_4b port map (
    a0 => A(4),
    a1 => A(5),
    a2 => A(6),
```

```
    a3 => A(7),
    b0 => B(4),
    b1 => B(5),
    b2 => B(6),
    b3 => B(7),
    c0 => c,
    s0 => S(4),
    s1 => S(5),
    s2 => S(6),
    s3 => S(7),
    c4 => c8
    );
end Structural;
```

Codice Componente 4.1: Definizione del componente Bit String Comparator Generic

4.2.2.2 Ripple Carry Adder 4 bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ripple_carry_adder_4b is
PORT (
      a0, a1, a2, a3 : in  std_logic;
      b0, b1, b2, b3 : in  std_logic;
      c0 : in std_logic;
      c4 : out std_logic;
      s0, s1, s2, s3: out std_logic
      );

end ripple_carry_adder_4b;

architecture Structural of ripple_carry_adder_4b is

Component full_adder
  PORT (
      x  : in  std_logic;
      y : in std_logic;
      c0 : in std_logic;
      s : out std_logic;
      c : out std_logic );
end Component;

signal c1, c2, c3 : std_logic;

begin
FA0 : full_adder port map (
    x => a0,
```

```
31      y => b0,
32      c0 => c0,
33      s => s0,
34      c => c1
35      );
36
37  FA1 : full_adder port map (
38      x => a1,
39      y => b1,
40      c0 => c1,
41      s => s1,
42      c => c2
43      );
44
45  FA2 : full_adder port map (
46      x => a2,
47      y => b2,
48      c0 => c2,
49      s => s2,
50      c => c3
51      );
52  FA3 : full_adder port map (
53      x => a3,
54      y => b3,
55      c0 => c3,
56      s => s3,
57      c => c4
58      );
59
60  end Structural;
```

Codice Componente 4.2: Definizione del componente Bit String Comparator Generic

4.2.2.3 Full Adder

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity full_adder is
5     PORT (
6         x  : in  std_logic;
7         y : in std_logic;
8         c0 : in std_logic;
9         s : out std_logic;
10        c : out std_logic );
11
12  end full_adder;
13
```

```
14 architecture structural of full_adder is
15
16 Component half_adder
17   PORT ( x : in std_logic;
18        y : in std_logic;
19        s : out std_logic;
20        c : out std_logic);
21 end Component;
22
23 signal s1, c1, c2 : std_logic;
24
25 begin
26
27 HA1 : half_adder port map (
28     x => x,
29     y => y,
30     s => s1,
31     c => c1);
32 HA2 : half_adder port map(
33     x => s1,
34     y => c0,
35     s => s,
36     c => c2);
37
38 c <= c2 or c1;
39
40
41 end structural;
```

Codice Componente 4.3: Definizione del componente Bit String Comparator Generic

4.2.2.4 Half Adder

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity half_adder is
5    PORT ( x : in std_logic;
6         y : in std_logic;
7         s : out std_logic;
8         c : out std_logic);
9  end half_adder;
10
11 architecture DataFlow of half_adder is
12
13 begin
14
15 s <= x xor y;
```

```
16 c <=  x and y;
17
18 end DataFlow;
```

Codice Componente 4.4: Definizione del componente Bit String Comparator Generic

4.3 Simulazione

Sono stati analizzati alcuni casi di particolare interesse con riporto 0 e 1 :

Riporto 0

A	B	C0	OUT	C_OUT
01010110	00011000	0	01101110	0
00000000	00000000	0	00000000	0
11111110	00000001	0	11111111	0
11111111	00000001	0	00000000	1
10000000	10000000	0	00000000	1

Riporto 1

A	B	C0	OUT	C_OUT
01010110	00011000	1	01101111	0
00000000	00000000	1	00000001	0
11111110	00000001	1	00000000	1
11111111	00000001	1	00000001	1
10000000	10000000	1	00000001	1

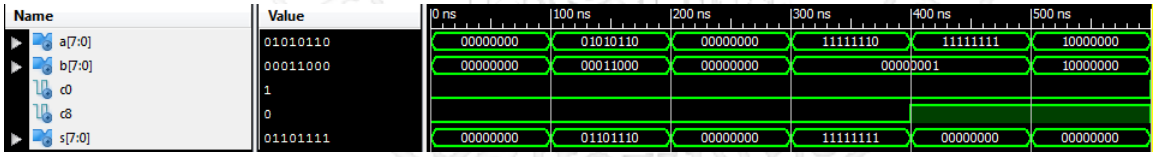
```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  -- Uncomment the following library declaration if using
6  -- arithmetic functions with Signed or Unsigned values
7  --USE ieee.numeric_std.ALL;
8
9  ENTITY tests IS
10 END tests;
11
12 ARCHITECTURE behavior OF tests IS
13
14     COMPONENT ripple_carry_adder_8b
15     PORT(
16          A : IN  std_logic_vector(7 downto 0);
17          B : IN  std_logic_vector(7 downto 0);
18          c0 : IN  std_logic;
19          c8 : OUT  std_logic;
20          S : OUT  std_logic_vector(7 downto 0)
```

```
21          );
22      END COMPONENT;
23
24
25     --Inputs
26     signal A : std_logic_vector(7 downto 0) := (others => '0');
27     signal B : std_logic_vector(7 downto 0) := (others => '0');
28     signal c0 : std_logic := '0';
29
30    --Outputs
31     signal c8 : std_logic;
32     signal S : std_logic_vector(7 downto 0);
33
34
35
36 BEGIN
37
38     uut: ripple_carry_adder_8b PORT MAP (
39             A => A,
40             B => B,
41             c0 => c0,
42             c8 => c8,
43             S => S
44          );
45
46
47     -- Stimulus process
48     stim_proc: process
49     begin
50          wait for 100 ns;
51
52      A <= "01010110";
53      B <= "00011000";
54      c0 <= '0';
55      wait for 100 ns;
56
57      A <= "00000000";
58      B <= "00000000";
59      c0 <= '0';
60      wait for 100 ns;
61
62      A <= "11111110";
63      B <= "00000001";
64      c0 <= '0';
65      wait for 100 ns;
66
67      A <= "11111111";
68      B <= "00000001";
69      c0 <= '0';
```

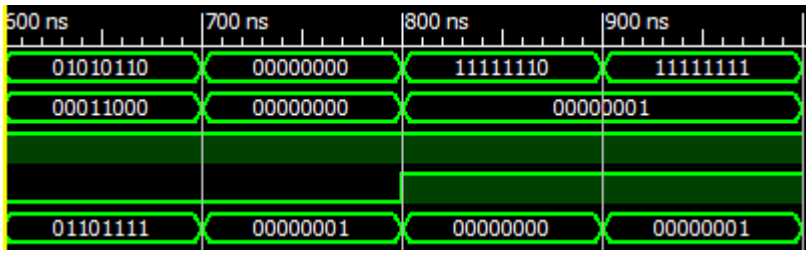
```
70      wait for 100 ns;
71
72      A <= "10000000";
73      B <= "10000000";
74      c0 <= '0';
75      wait for 100 ns;
76
77      A <= "01010110";
78      B <= "00011000";
79      c0 <= '1';
80      wait for 100 ns;
81
82      A <= "00000000";
83      B <= "00000000";
84      c0 <= '1';
85      wait for 100 ns;
86
87      A <= "11111110";
88      B <= "00000001";
89      c0 <= '1';
90      wait for 100 ns;
91
92      A <= "11111111";
93      B <= "00000001";
94      c0 <= '1';
95      wait for 100 ns;
96
97      A <= "10000000";
98      B <= "10000000";
99      c0 <= '1';
100
101
102       wait;
103    end process;
104
105 END;
```

Codice Componente 4.5: Testbench ripple carry adder

Risultati del test nel caso riporto 0 :

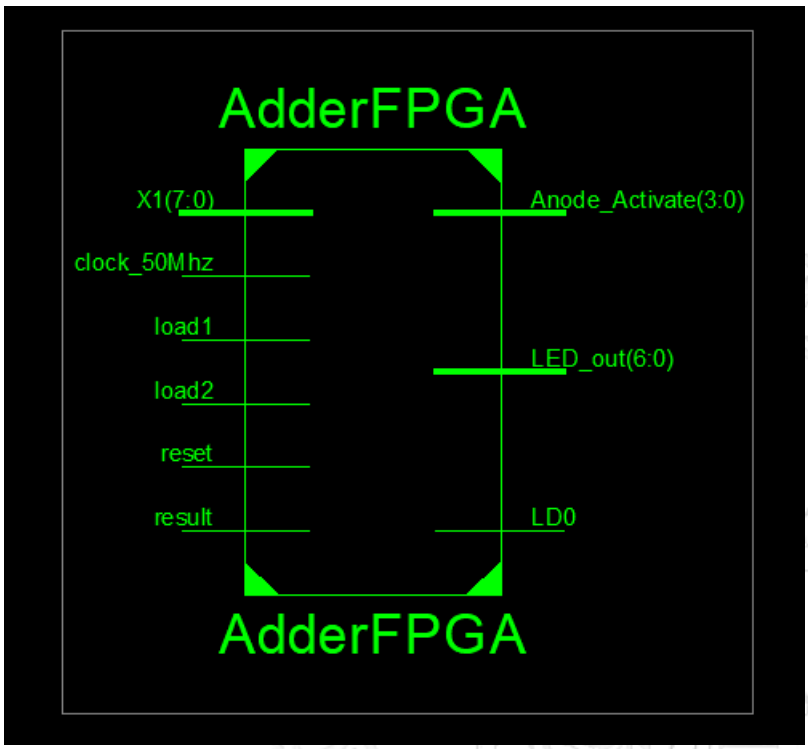


Risultati del test nel caso riporto 1 :



4.4 Sintesi su board FPGA

La sintesi del ripple carry adder a 8 bit su FPGA Nexys2 ha richiesto la realizzazione di componenti aggiuntivi.
Due bottoni (load 1 e load2) vengono utilizzati per l'acquisizione del valore dei due operandi, attraverso gli switch. Viene effettuata la somma e con la pressione di un terzo pulsante (result) viene mostrata su display a 7 segmenti. Il quarto ed ultimo pulsante viene invece usato come reset. La condizione di overflow è mostrata attraverso l'accensione del led LD0.



Il display a 7 segmenti utilizzato è quello mostrato nei capitoli precendenti. Di seguito sono presenti i file vhd e ucf relativi alla sintesi del componente su FPGA.

4.4.1 File UCF

```
1
```

```
2  NET "LED_out<6>" LOC = "L18"; # Bank = 1, Pin name = IO_L10P_1, Type = I/O,
       Sch name = CA
3  NET "LED_out<5>" LOC = "F18"; # Bank = 1, Pin name = IO_L19P_1, Type = I/O,
       Sch name = CB
4  NET "LED_out<4>" LOC = "D17"; # Bank = 1, Pin name = IO_L23P_1/HDC, Type =
       DUAL, Sch name = CC
5  NET "LED_out<3>" LOC = "D16"; # Bank = 1, Pin name = IO_L23N_1/LDC0, Type =
       DUAL, Sch name = CD
6  NET "LED_out<2>" LOC = "G14"; # Bank = 1, Pin name = IO_L20P_1, Type = I/O,
       Sch name = CE
7  NET "LED_out<1>" LOC = "J17"; # Bank = 1, Pin name = IO_L13P_1/A6/RHCLK4/
       IRDY1, Type = RHCLK/DUAL, Sch name = CF
8  NET "LED_out<0>" LOC = "H14"; # Bank = 1, Pin name = IO_L17P_1, Type = I/O,
       Sch name = CG
9
10 NET "clock_50Mhz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
       = GCLK, Sch name = GCLK0
11 NET "Anode_Activate<0>" LOC = "F17"; # Bank = 1, Pin name = IO_L19N_1, Type
       = I/O, Sch name = AN0
12 NET "Anode_Activate<1>" LOC = "H17"; # Bank = 1, Pin name = IO_L16N_1/A0,
       Type = DUAL, Sch name = AN1
13 NET "Anode_Activate<2>" LOC = "C18"; # Bank = 1, Pin name = IO_L24P_1/LDC1,
       Type = DUAL, Sch name = AN2
14 NET "Anode_Activate<3>" LOC = "F15"; # Bank = 1, Pin name = IO_L21P_1, Type
       = I/O, Sch name = AN3
15
16 NET "X1<0>"     LOC = "G18";    # Sch name = SW0
17 NET "X1<1>"     LOC = "H18";    # Sch name = SW1
18 NET "X1<2>"     LOC = "K18";    # Sch name = SW2
19 NET "X1<3>"     LOC = "K17";    # Sch name = SW3
20 NET "X1<4>" LOC = "L14";
21 NET "X1<5>" LOC = "L13";
22 NET "X1<6>" LOC = "N17";
23 NET "X1<7>" LOC = "R17";
24 NET "load1" LOC = "B18";
25 NET "load2"  LOC = "D18";
26 NET "reset"  LOC = "H13";
27 NET "result"  LOC = "E18";
28 NET "LD0" LOC = "J14";
```

4.4.2 Sintesi finale

Un process regola l'immissione dei dati di input e il reset. Vengono utilizzati due flag booleani OP_1 ed OP_2 per verificare che siano stati immessi entrambi i valori. Quando vengono inseriti entrambi gli operandi, la pressione del terzo pulsante (result) pone in ingresso al segnale value_to_display il valore della somma e permette di far accendere il led della scheda se si ha overflow. Tale condizione si verifica quando gli addendi sono positivi e si ottiene un risultato negativo o quando gli addendi sono negativi e si ottiene un risultato positivo.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.std_logic_signed.all;
5
6  entity AdderFPGA is
7
8   Port (   clock_50Mhz : in STD_LOGIC;
9            reset : in STD_LOGIC;
10       X1 : in STD_LOGIC_VECTOR (7 downto 0);
11        load1,load2,result : in STD_LOGIC;
12           Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
13           LED_out : out STD_LOGIC_VECTOR (6 downto 0);
14       LD0 : out STD_LOGIC );
15
16 end AdderFPGA;
17
18 architecture Behavioural of AdderFPGA is
19
20 signal OP_1, OP_2 : boolean := false;
21 signal op1,op2,sum : std_logic_vector (7 downto 0) := (others => '0') ;
22 signal value_to_display : std_logic_vector (7 downto 0):= (others => '0');
23 signal L1: std_logic:='0';
24
25 component ripple_carry_adder_8b is PORT (
26       A,B : in  std_logic_vector (7 downto 0);
27         c0 : in std_logic;
28       c8: out std_logic;
29       S: out std_logic_vector (7 downto 0)
30       );
31 end component;
32
33 component SEV_SEG_DISP is
34     Port ( clock_50Mhz : in STD_LOGIC;
35            reset : in STD_LOGIC;
36       X: in STD_LOGIC_VECTOR (7 downto 0);
37            Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
38            LED_out : out STD_LOGIC_VECTOR (6 downto 0)
39       );
40 end component;
41
42
43
44 begin
45
46 FA8 : ripple_carry_adder_8b port map(
47     A => op1,
48     B => op2,
```

```
49      c0 => '0',
50      S => sum
51  );
52
53  SSD : SEV_SEG_DISP port map(
54      clock_50Mhz => clock_50Mhz,
55      reset => reset,
56      Anode_Activate => Anode_Activate,
57      LED_out => LED_out,
58      X => value_to_display
59  );
60
61
62
63  process (clock_50Mhz, reset)
64  begin
65
66    if (reset = '1') then
67      -- Reset
68      op1 <= (others => '0');
69      op2 <= (others => '0');
70      OP_1 <= false;
71      OP_2 <= false;
72    elsif (rising_edge(clock_50Mhz)) then
73
74      -- Immissione dati input
75      if load1 = '1' then
76          op1 <= X1;
77          OP_1 <= true;
78          OP_2 <= false;
79          LD0<='0';
80      elsif load2 = '1' then
81          op2 <= X1;
82          OP_2 <= true;
83      end if;
84
85      if (OP_1=true and OP_2=true) then
86        -- Display del risultato
87        if (result = '1') then
88          value_to_display <=sum;
89
90        if ((op1(7)='0' and op2(7)='0' and sum(7)='1') or (op1(7)='1' and op2
              (7)='1' and sum(7)='0')) then
91          LD0<='1';
92        else
93          LD0<='0';
94        end if;
95
96        end if;
```

```
97      else
98        -- Display dell'operando
99        value_to_display <= X1;
100       LD0<='0';
101     end if;
102   end if;
103
104 end process;
105 end Behavioural;
```

Capitolo 5

Esercizio 5

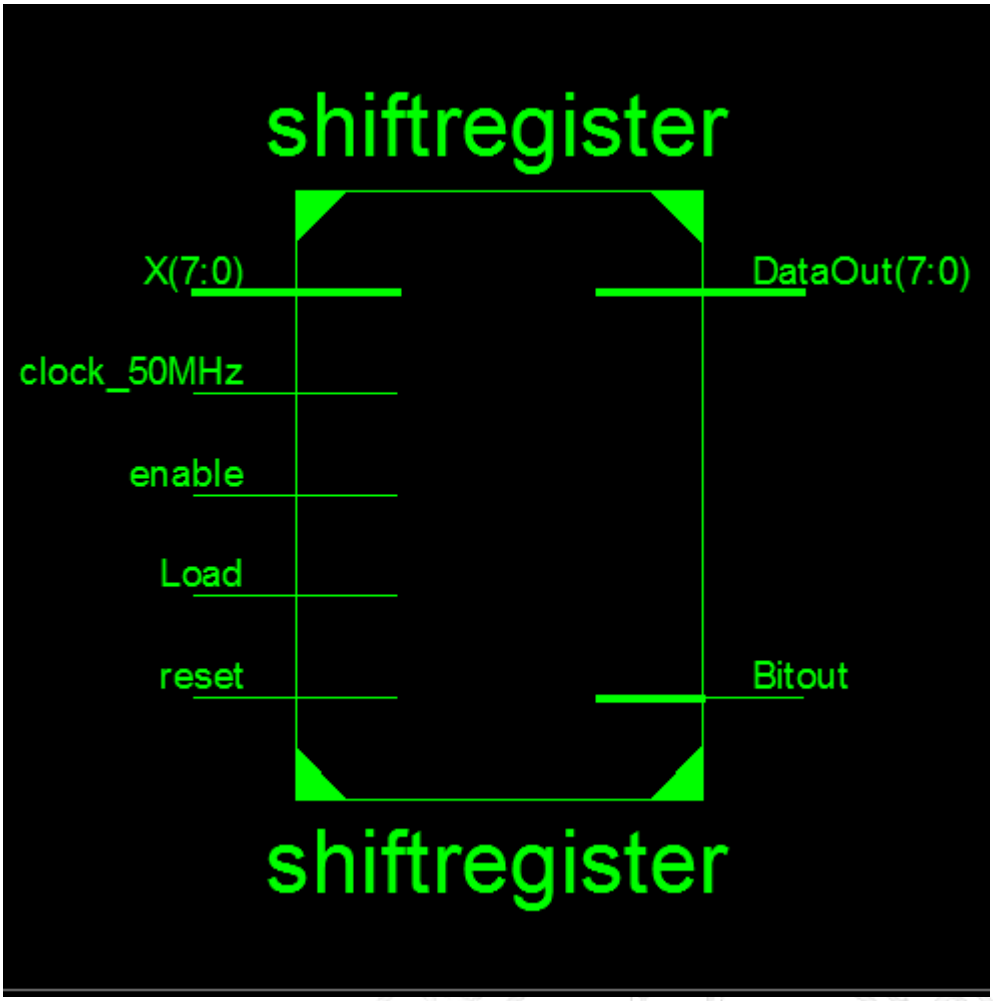
5.1 Traccia

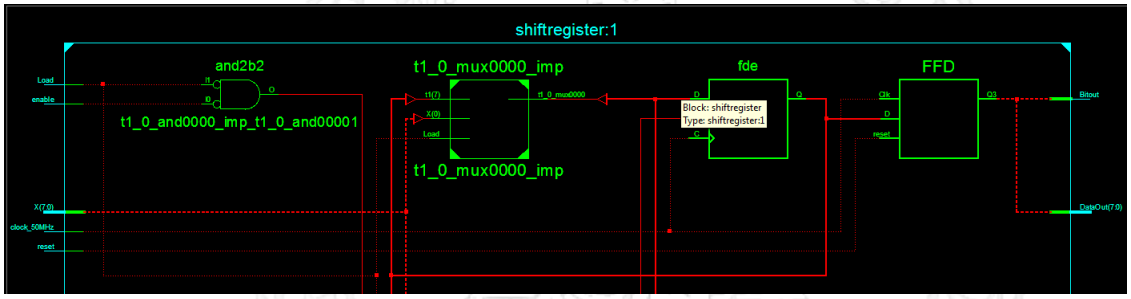
Progettare ed implementare in VHDL un registro a scorrimento circolare di 8 bit utilizzando i flip-flop D edge triggered implementati all'esercizio 3 (si scelga una delle realizzazioni). Sintetizzare sulla board il componente utilizzando gli switch per acquisire il dato iniziale, un bottone per acquisire il segnale di shift e i led per visualizzare il contenuto del registro in ogni istante.

5.2 Soluzione

Lo shift register implementato prende in ingresso una stringa di 8 bit X, che può essere caricata nei flip flop (D edge triggered su fronte di discesa) in parallelo attraverso la pressione del pulsante load. Lo shift viene eseguito nel momento in cui il segnale di enable (pulsante enable) è pari ad 1. In questo caso, il valore contenuto nel flip flop i-esimo verrà propagato al flip flop (i+1)-esimo ogni qualvolta viene premuto il pulsante. Il segnale di reset pone a 0 il contenuto dei flip flop.
In output è possibile visualizzare il valore del bit meno significativo tramite uscita Bitout e il valore corrente di tutti i registri con DataOut.

5.2.1 Schematici





5.2.2 Codice

5.2.2.1 Registro a scorrimento circolare

Un process regola il caricamento dei dati in parallelo e l'abilitazione dell'operazione di shift, mentre i flip flop sono tra loro collegati per consentire la propagazione circolare del dato.

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
```

```
4
5  entity shiftregister is
6      Port (
7          X : in  STD_LOGIC_VECTOR (7 downto 0);
8          clock_50MHz, reset,Load,enable : in  STD_LOGIC;
9             Bitout : out STD_LOGIC:='0';
10         DataOut : out  STD_LOGIC_VECTOR (7 downto 0)
11         );
12 end shiftregister;
13
14 architecture Structural of shiftregister is
15
16 COMPONENT FFD
17     PORT(
18         D : IN std_logic;
19         Clk : IN std_logic;
20         reset : IN std_logic;
21         Q3 : OUT std_logic
22         );
23     END COMPONENT;
24
25 signal t1, t2 : std_logic_vector(7 downto 0):=(others=>'0');
26
27 begin
28
29
30
31 FFD0: FFD PORT MAP(
32         D => t1(0),
33         Q3 => t2(0),
34         Clk => clock_50MHz,
35         reset => reset
36     );
37
38 FFD1: FFD PORT MAP(
39         D => t1(1),
40         Q3 => t2(1),
41         Clk => clock_50MHz,
42         reset => reset
43     );
44
45 FFD2: FFD PORT MAP(
46         D => t1(2),
47         Q3 => t2(2),
48         Clk => clock_50MHz,
49         reset => reset
50     );
51
52
```

```
53
54  FFD3: FFD PORT MAP(
55          D => t1(3),
56          Q3 => t2(3),
57          Clk => clock_50MHz,
58          reset => reset
59      );
60
61
62  FFD4: FFD PORT MAP(
63          D => t1(4),
64          Q3 => t2(4),
65          Clk => clock_50MHz,
66          reset => reset
67      );
68
69
70  FFD5: FFD PORT MAP(
71          D => t1(5),
72          Q3 => t2(5),
73          Clk => clock_50MHz,
74          reset => reset
75      );
76
77
78  FFD6: FFD PORT MAP(
79          D => t1(6),
80          Q3 => t2(6),
81          Clk =>clock_50MHz,
82          reset => reset
83      );
84
85  FFD7: FFD PORT MAP(
86          D => t1(7),
87          Q3 => t2(7),
88          Clk => clock_50MHz,
89          reset => reset
90      );
91
92
93
94  DataOut <= t2;
95  BitOut <= t2(7);
96
97  p: process(clock_50MHz)
98
99  begin
100
101
```

```
102 if(rising_edge(clock_50MHz)) then
103
104   if (Load='1')then
105     t1<=X;
106
107   elsif (enable='1') then
108
109     t1(7 downto 1)<=t1(6 downto 0);
110     t1(0)<=t1(7);
111   end if;
112
113 end if;
114 end process;
115
116 end Structural;
```

Codice Componente 5.1: Definizione del componente Registro a scorrimento circolare

5.2.2.2 FFD sul fronte di discesa con reset

Per l'implementazione del registro circolare sono stati utilizzati i seguenti flip flop edge triggered attivi su fronte di discesa (con aggiunta del reset):

```
1
2  library ieee;
3  use ieee.std_logic_1164.ALL;
4  use ieee.numeric_std.ALL;
5  library UNISIM;
6  use UNISIM.Vcomponents.ALL;
7
8  entity FFD is
9     port ( CLK    : in    std_logic:='0';
10           D      : in    std_logic:='0';
11           reset  : in    std_logic:='0';
12           Q3     : out   std_logic :='0'
13         );
14 end FFD;
15
16 architecture BEHAVIORAL of FFD is
17    attribute BOX_TYPE   : string ;
18    signal A12       : std_logic:='0';
19    signal inverted  : std_logic:='0';
20    signal Q1        : std_logic:='0';
21    signal Q1N       : std_logic:='0';
22    signal Q2        : std_logic:='0';
23    signal Q2N       : std_logic:='0';
24    signal Q3_DUMMY  : std_logic:='0';
25    signal Q3N_DUMMY : std_logic:='0';
26    component AND2
```

```
27      port ( I0 : in    std_logic;
28             I1 : in    std_logic;
29             O  : out   std_logic);
30    end component;
31    attribute BOX_TYPE of AND2 : component is "BLACK_BOX";
32
33    component NOR2
34      port ( I0 : in    std_logic;
35             I1 : in    std_logic;
36             O  : out   std_logic);
37    end component;
38    attribute BOX_TYPE of NOR2 : component is "BLACK_BOX";
39
40    component NOR3
41      port ( I0 : in    std_logic;
42             I1 : in    std_logic;
43             I2 : in    std_logic;
44             O  : out   std_logic);
45    end component;
46    attribute BOX_TYPE of NOR3 : component is "BLACK_BOX";
47
48    component INV
49      port ( I : in    std_logic;
50             O : out   std_logic);
51    end component;
52    attribute BOX_TYPE of INV : component is "BLACK_BOX";
53
54    begin
55    Q3 <= Q3_DUMMY;
56
57
58    a1 : AND2
59      port map (I0=>inverted,
60                I1=>D,
61                O=>A12);
62
63    g1 : NOR2
64      port map (I0=>A12,
65                I1=>Q1,
66                O=>Q1N);
67
68    g2 : NOR3
69      port map (I0=>Q1N,
70                I1=>CLK,
71                I2=>Q2N,
72                O=>Q1);
73
74    g3 : NOR2
75      port map (I0=>CLK,
```

```
76                  I1=>Q2,
77                  O=>Q2N);
78
79     g4 : NOR2
80        port map (I0=>Q2N,
81                  I1=>Q1N,
82                  O=>Q2);
83
84     g5 : NOR2
85        port map (I0=>Q1,
86                  I1=>Q3_DUMMY,
87                  O=>Q3N_DUMMY);
88
89     g6 : NOR2
90        port map (I0=>Q3N_DUMMY,
91                  I1=>Q2N,
92                  O=>Q3_DUMMY);
93
94     INV1 : INV
95        port map (I=>reset,
96                  O=>inverted);
97
98 end BEHAVIORAL;
```

Codice Componente 5.2: Definizione del componente FFD con reset

5.3 Simulazione

Per verificare il corretto funzionamento del registro circolare, sono stati eseguiti questi test :

1 - Inizializzazione dei registri a 0. Inserimento dei valori di X pari a 01001111, load ed enable pari ad '1';

2 - Test di funzionamento del segnale di reset;

3 - Inserimento di nuovi valori dopo il reset.

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  ENTITY testb IS
6  END testb;
7
8  ARCHITECTURE behavior OF testb IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11
12     COMPONENT shiftregister
13     PORT(
14          X : IN  std_logic_vector(7 downto 0);
```

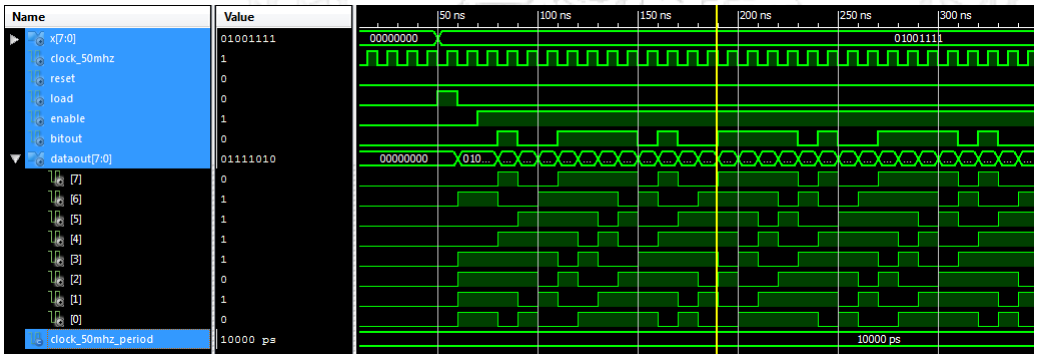
```
15          clock_50MHz : IN  std_logic;
16          reset : IN  std_logic;
17          Load,enable : IN  std_logic;
18          Bitout : OUT  std_logic;
19          DataOut : OUT  std_logic_vector(7 downto 0)
20         );
21     END COMPONENT;
22
23
24    --Inputs
25    signal X : std_logic_vector(7 downto 0) := (others => '0');
26    signal clock_50MHz : std_logic := '0';
27    signal reset : std_logic := '0';
28    signal Load,enable : std_logic := '0';
29
30   --Outputs
31    signal Bitout : std_logic:= '0';
32    signal DataOut : std_logic_vector(7 downto 0):= (others => '0');
33
34    -- Clock period definitions
35    constant clock_50MHz_period : time := 10 ns;
36
37 BEGIN
38
39   -- Instantiate the Unit Under Test (UUT)
40    uut: shiftregister PORT MAP (
41           X => X,
42           clock_50MHz => clock_50MHz,
43           reset => reset,
44           Load => Load,
45         enable=>enable,
46           Bitout => Bitout,
47           DataOut => DataOut
48         );
49
50    -- Clock process definitions
51    clock_50MHz_process :process
52    begin
53     clock_50MHz <= '0';
54     wait for clock_50MHz_period/2;
55     clock_50MHz <= '1';
56     wait for clock_50MHz_period/2;
57    end process;
58
59
60    -- Stimulus process
61    stim_proc: process
62    begin
63       -- hold reset state for 100 ns.
```

```
64        wait for 50 ns;
65
66      -- TEST 1
67      X<="01001111";
68      Load<='1';
69      wait for 10 ns;
70      Load <= '0';
71      wait for 20 ns;
72      enable <= '1';
73      wait for 300 ns;
74
75      -- TEST 2
76      reset <= '1';
77      wait for 50 ns;
78      reset <= '0';
79      enable <= '0';
80      wait for 150 ns;
81
82      -- TEST 3
83      X<="01000000";
84      Load<='1';
85      wait for 10 ns;
86      Load <= '0';
87      wait for 20 ns;
88      enable <= '1';
89
90        wait;
91    end process;
92
93  END;
```

Codice Componente 5.3: Definizione del testbench per Registro a scorrimento circolare

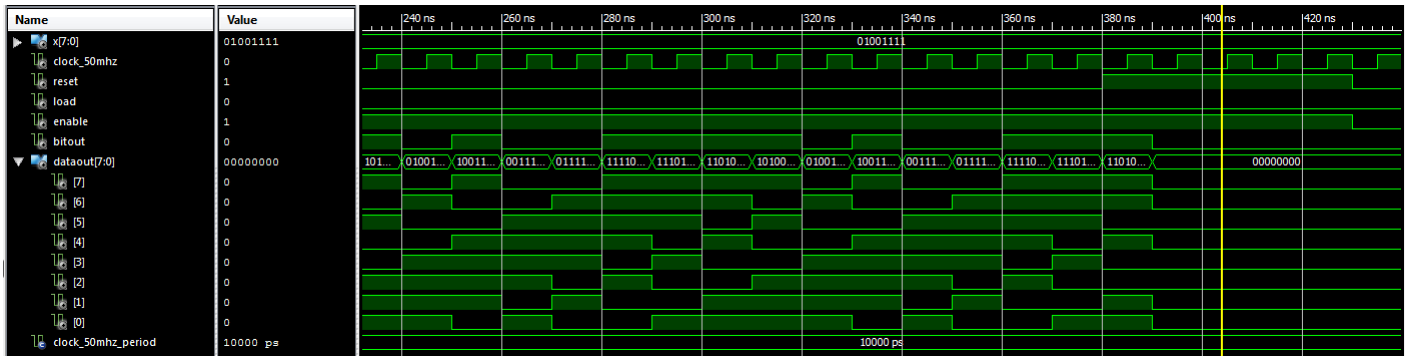
5.3.0.1 Test 1

Vengono caricati i bit in parallelo su tutti i flip flop. I bit vengono propagati per tutta la durata della simulazione in corrispondenza del fronte di discesa del clock.



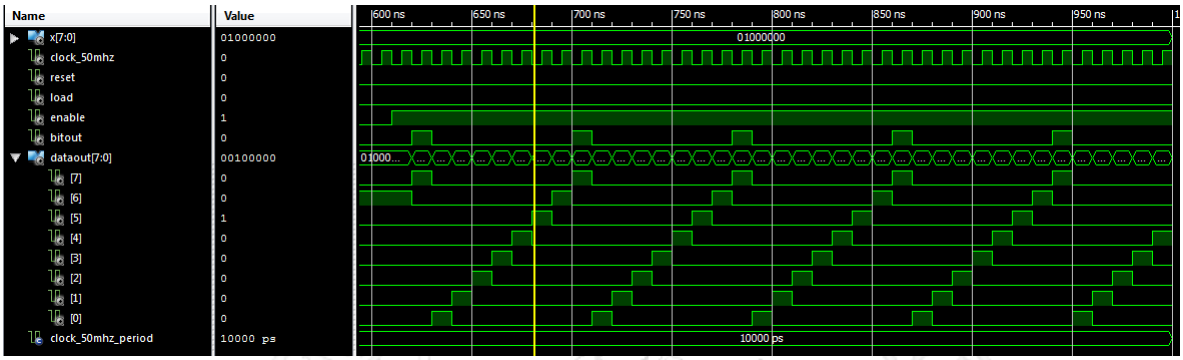
5.3.0.2 Test 2

Osserviamo che il reset funziona correttamente. L'attivazione del segnale di reset pone direttamente a 0 il contenuto dei registri. Enable viene posto a 0 e lo shift register termina lo scorrimento.



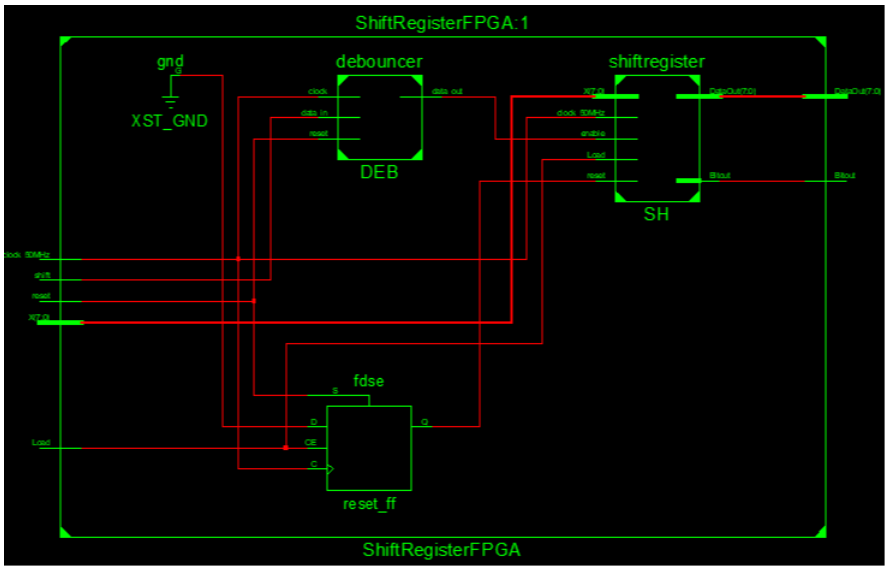
5.3.0.3 Test 3

Dopo il segnale di reset viene caricato un nuovo dato (01000000), che viene propagato fra tutti i flip flop per tutta la durata della simulazione.



5.4 Sintesi su board FPGA

Adattare l'implementazione per il funzionamento su FPGA è semplice in questo caso, poichè essendo il caricamento effettuato in parallelo è possibile collegare l'input X con gli switch e l'output con i led. Load, reset e shift corrispondono a 3 dei bottoni della board. Load carica in X i dati attualmente presenti sugli switch, reset pone a 0 il contenuto dei flip flop e shift esegue un'unica operazione di shift circolare visualizzabile in output tramite i led.




```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  -- Uncomment the following library declaration if using
6  -- arithmetic functions with Signed or Unsigned values
7  --use IEEE.NUMERIC_STD.ALL;
8
9  -- Uncomment the following library declaration if instantiating
10 -- any Xilinx primitives in this code.
11 --library UNISIM;
12 --use UNISIM.VComponents.all;
13
14 entity ShiftRegisterFPGA is
15     Port (
16         X : in  STD_LOGIC_VECTOR (7 downto 0);
17         clock_50MHz, reset : in  STD_LOGIC;
18            Load,shift :in STD_LOGIC;
19            Bitout : out STD_LOGIC;
20         DataOut : out  STD_LOGIC_VECTOR (7 downto 0)
21         );
22 end ShiftRegisterFPGA;
23
24 architecture Structural of ShiftRegisterFPGA is
25
26
27 COMPONENT shiftregister
28   PORT(
29         X : in  STD_LOGIC_VECTOR (7 downto 0);
30         clock_50MHz, reset,Load,enable : in  STD_LOGIC;
31            Bitout : out STD_LOGIC:='0';
32         DataOut : out  STD_LOGIC_VECTOR (7 downto 0)
```

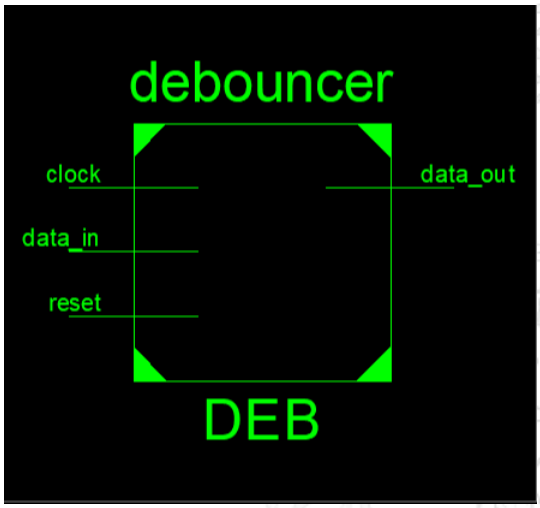
```
33
34    );
35  END COMPONENT;
36
37  COMPONENT debouncer
38      PORT(
39      clock,reset: in std_logic;
40      data_in: in std_logic;
41      data_out: out std_logic
42          );
43
44  END COMPONENT;
45
46
47  signal t1, t2 : std_logic_vector(7 downto 0);
48  signal reset_ff, c: std_logic :='0';
49
50
51  begin
52
53  DEB: debouncer PORT MAP(
54
55    clock=>clock_50MHz,
56    reset=>reset,
57    data_in=>shift,
58    data_out=>c
59
60
61
62  );
63
64
65  SH: shiftregister PORT MAP(
66    X=>X,
67    clock_50MHz=>clock_50MHz,
68    reset=>reset_ff,
69    enable=>c,
70    Load=>Load,
71    Bitout=>Bitout,
72    DataOut=>DataOut
73
74  );
75
76
77  p: process(clock_50MHz)
78
79  begin
80
81
```

```
82 if(rising_edge(clock_50MHz)) then
83   if (reset='1') then
84     reset_ff<='1';
85
86   elsif( Load='1') then
87     reset_ff<='0';
88
89
90   end if;
91
92 end if;
93 end process;
94
95 end Structural;
```

Codice Componente 5.4: Definizione del file UCF

5.4.0.1 Debouncer

Il componente debouncer permette di generare un unico impulso di durata pari al periodo del segnale di clock che esso prende in ingresso. Esso ci permette, alla pressione di un determinato bottone, di considerare quella pressione come un unico impulso in ingresso al nostro sistema (data l'elevata frequenza del clock). In questo caso è utilizzato per il bottone di shift, che corrisponde all' input data_in.




```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use ieee.std_logic_unsigned.all;
5
6 entity debouncer is
7   port(
8   clock,reset: in std_logic;
```

```
  data_in: in std_logic;
  data_out: out std_logic
  );

end debouncer;

architecture Behavioral of debouncer is
signal s1,s2,slow_clk_enable: std_logic :='0';
signal counter: integer:=1;

begin

proc1: process(clock,reset)
begin
 if(reset='1') then
    counter <= 1;
    slow_clk_enable <= '0';
 elsif(rising_edge(clock)) then
          if(counter =1000000) then
            counter <=  1;
            slow_clk_enable <= '1';
          else
            slow_clk_enable <= '0';
            counter <= counter + 1;
          end if;
 end if;

end process;


proc2: process (clock,reset)
begin
  if (reset='1') then
    s1<='0';
    s2<='0';
  elsif (rising_edge(clock)) then
    if(slow_clk_enable='1') then
      s1<= data_in;
      s2<=s1;
    end if;
  end if;


end process;

data_out<= s1 and (not s2) and slow_clk_enable;

end Behavioral;
```

Codice Componente 5.5: Definizione del componente FFD con reset

5.4.0.2 File UCF

```
1
2
3  NET "clock_50MHz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
       = GCLK, Sch name = GCLK0
4
5  NET "X<0>"        LOC = "G18";    # Sch name = SW0
6  NET "X<1>"        LOC = "H18";    # Sch name = SW1
7  NET "X<2>"        LOC = "K18";    # Sch name = SW2
8  NET "X<3>"        LOC = "K17";    # Sch name = SW3
9  NET "X<4>"       LOC = "L14";
10 NET "X<5>"       LOC = "L13";
11 NET "X<6>"       LOC = "N17";
12 NET "X<7>"       LOC = "R17";
13
14
15 NET "Load" LOC = "H13";
16 NET "reset" LOC = "E18";
17 NET "shift" LOC ="D18";
18
19 NET "DataOut<0>" LOC = "J14";
20 NET "DataOut<1>" LOC = "J15";
21 NET "DataOut<2>" LOC = "K15";
22 NET "DataOut<3>" LOC = "K14";
23 NET "DataOut<4>" LOC = "E16";
24 NET "DataOut<5>" LOC = "P16";
25 NET "DataOut<6>" LOC = "E4";
26 NET "DataOut<7>" LOC = "P4";
```

Codice Componente 5.6: Definizione del file UCF

Capitolo 6

Esercizio 6

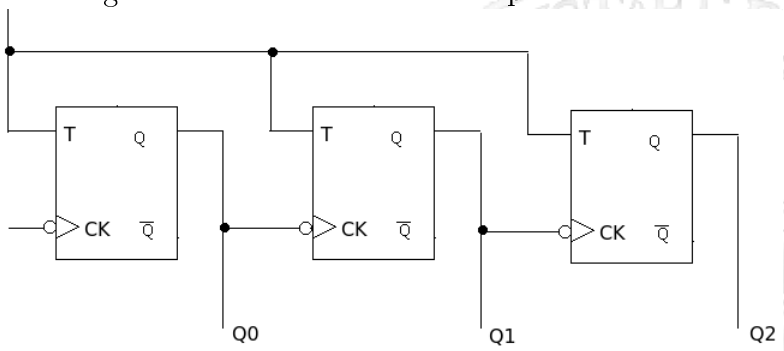
6.1 Traccia

Progettare ed implementare in VHDL un contatore mod-16 nella modalità serie e parallelo. Sintetizzare sulla board il componente (nella sola modalità serie) utilizzando un bottone per acquisire il segnale di conteggio e 2 cifre del display per visualizzare il contenuto del registro.

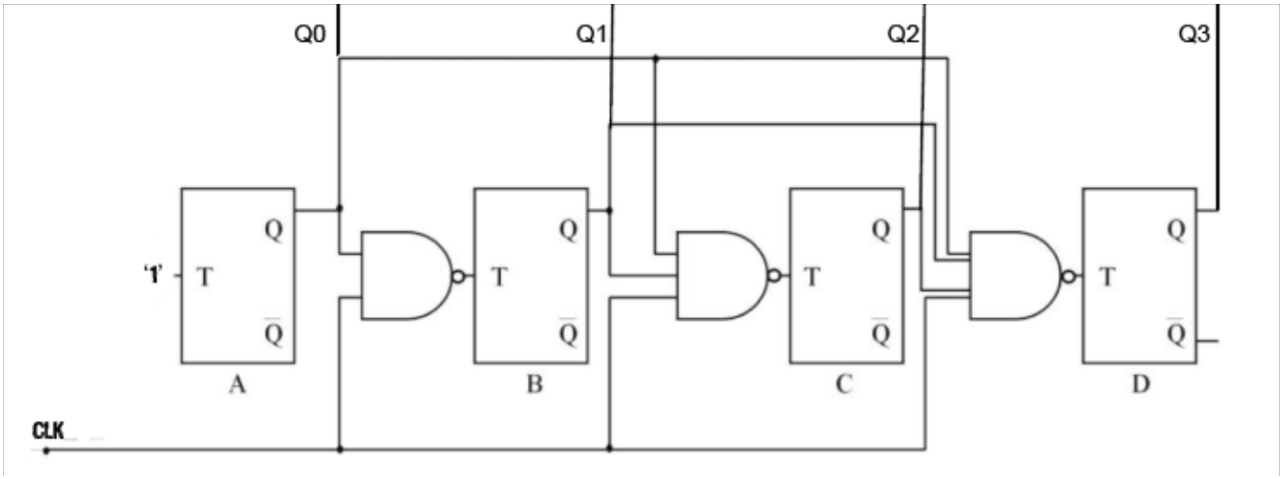
6.2 Soluzione

La soluzione proposta mostra i contatori in parallelo e serie realizzati con flip flop di tipo T (fronte di discesa), dotati di ingresso T, CLK e Reset e di uscita Q.

Di seguito viene mostrato un esempio di contatore seriale.



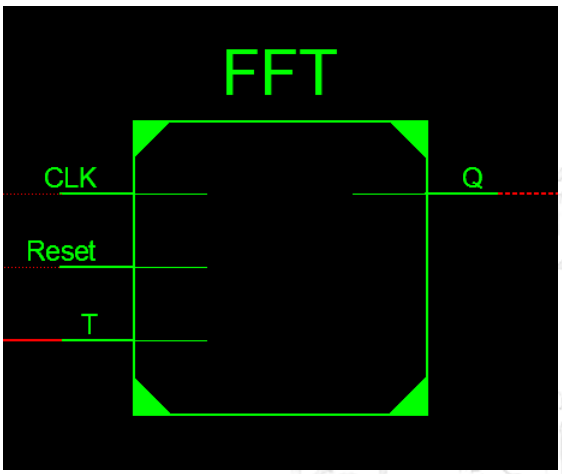
Tale soluzione sfrutta le proprietà del flip flop T per costruire un contatore in serie. Nel nostro caso i flip flop utilizzati saranno 4 poichè è richiesto un contatore modulo 16. Come si può osservare, il clock viene dato in ingresso solamente al primo flip flop. Tutti gli altri avranno clock pari all'uscita del precedente e tutti avranno ingresso T pari al valore logico '1'.
Per quanto riguarda il contatore parallelo invece abbiamo una situazione di questo tipo:

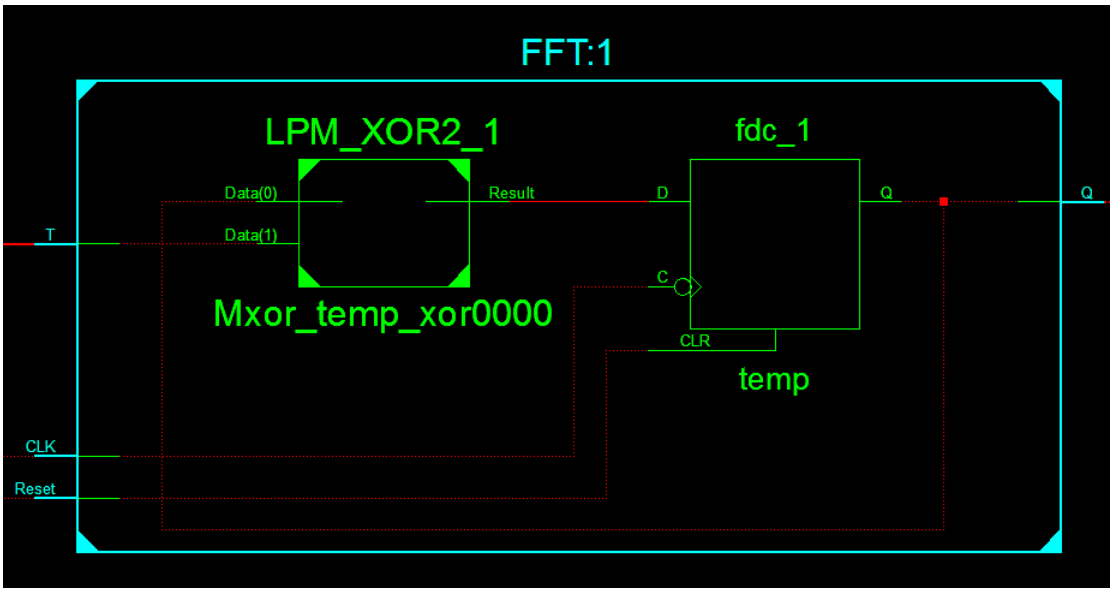


Lo stesso clock va in ingresso a tutti i flip flop della catena. Un flip flop commuta quando i precedenti hanno commutato ed il clock è alto.

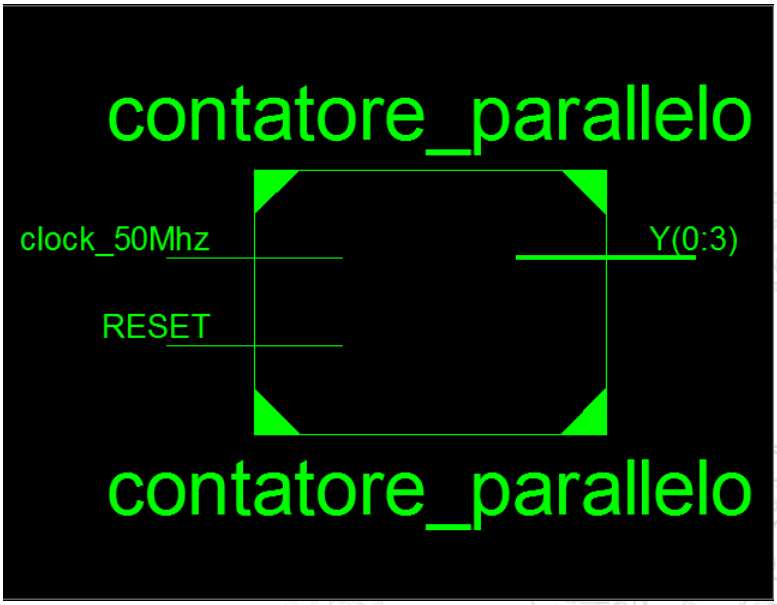
6.2.1 Schematici

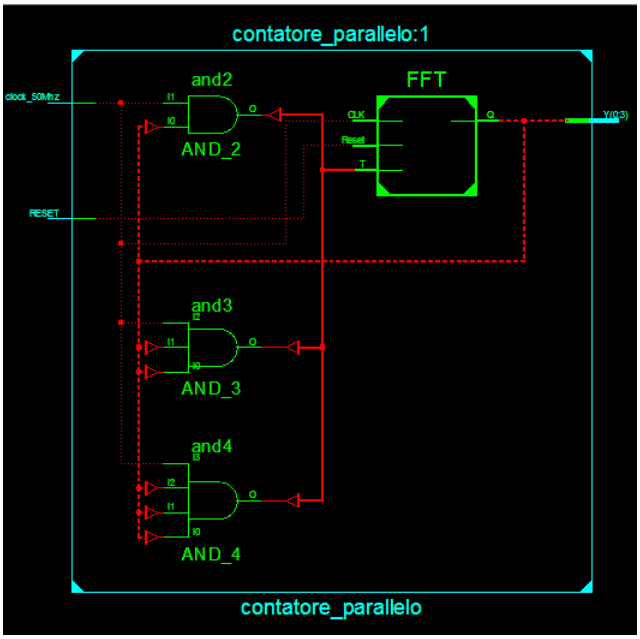
6.2.1.1 Flip Flop T



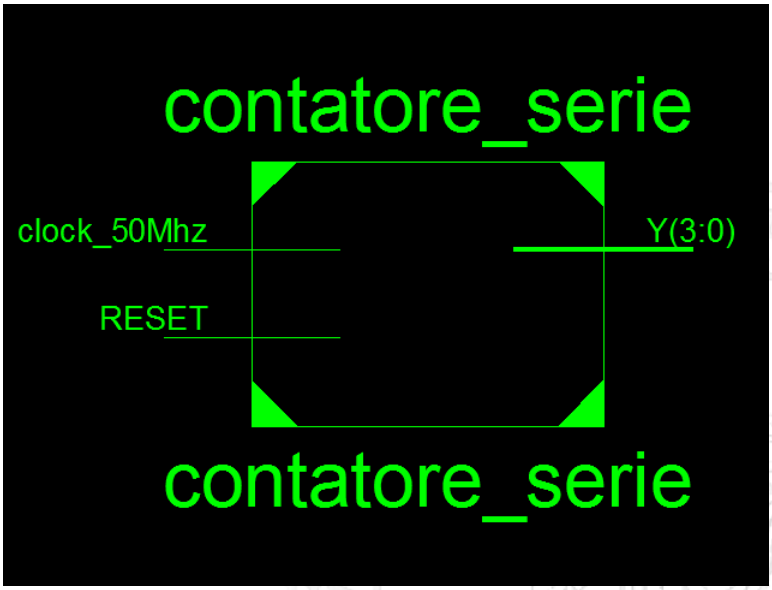


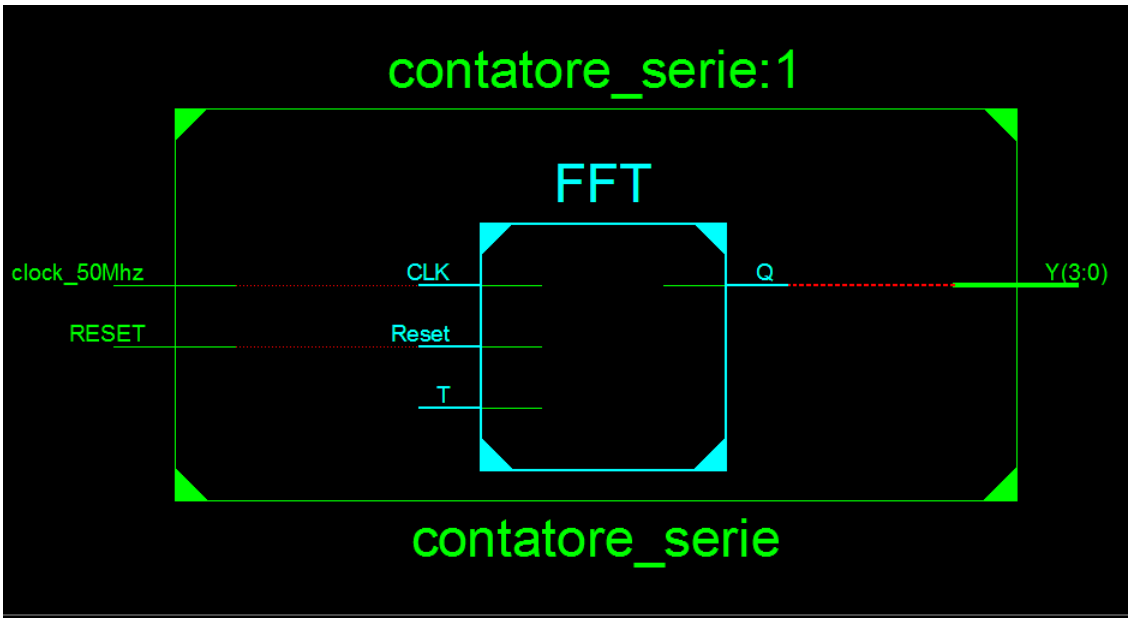
6.2.1.2 Contatore Parallelo Mod-16





6.2.1.3 Contatore Seriale Mod-16





6.2.2 Codice

Di seguito è riportato il codice dei flip flop utilizzati e dei contatori in parallelo e serie, questi ultimi realizzati con una descrizione di tipo structural.

6.2.2.1 Flip Flop T

I flip flop utilizzati per l'implementazione dei contatori sono flip flop di tipo T sensibili al fronte di discesa del clock. Ad ogni colpo di clock l'uscita è calcolata attraverso una XOR tra il segnale di ingresso T e il segnale temporaneo calcolato precedentemente,che viene posto a 0 in caso di reset.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity FFT is port (
6
7     T,Reset,CLK: in std_logic;
8     Q: out std_logic
9  );
10 end FFT;
11
12 architecture Behavioral of FFT is
13 signal  temp: std_logic := '0';
14
15 begin
16 process (Reset,CLK)
17
18 begin
19 if Reset='1' then
20     temp <= '0';
```

```
21
22 elsif (falling_edge(CLK)) then
23     temp <= T xor temp;
24 end if;
25
26 end process;
27
28 Q <= temp;
29 end Behavioral;
```

Codice Componente 6.1: Definizione del flip flop T

6.2.2.2 Contatore Parallelo Mod-16

```
1
2
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5   use ieee.std_logic_unsigned.all;
6   library UNISIM;
7  use UNISIM.Vcomponents.ALL;
8
9  entity contatore_parallelo is
10
11 port (
12     clock_50Mhz : in std_logic;
13     RESET : in std_logic;
14     Y : out std_logic_vector(0 to 3)
15     );
16
17 end contatore_parallelo;
18
19 architecture Structural of contatore_parallelo is
20
21
22
23 Component FFT
24      port ( CLK : in    std_logic;
25           T   : in    std_logic;
26        reset : in std_logic;
27           Q   : out   std_logic
28
29
30        );
31 end Component;
32
33 signal t:  std_logic_vector (6 downto 0) := (others => '0') ;
34
```

```
35
36 begin
37
38
39 FFT0 : FFT port map (
40     T => '1',
41     CLK => clock_50Mhz,
42     Q => t(0),
43     reset => reset
44     );
45
46 AND_2 : AND2
47     port map (I0=>t(0),
48                 I1=>clock_50Mhz,
49                 O=>t(1));
50
51
52 FFT1 : FFT port map (
53     T => t(1),
54     CLK => clock_50Mhz,
55     Q => t(2),
56
57     reset => reset
58     );
59
60 AND_3 : AND3
61     port map (I0=>t(0),
62                 I1=>t(2),
63             I2 =>clock_50Mhz,
64                 O=>t(3));
65
66
67
68 FFT2 : FFT port map (
69     T => t(3),
70     CLK => clock_50Mhz,
71     Q => t(4),
72     reset => reset
73     );
74
75 AND_4 : AND4
76     port map (I0=>t(0),
77                 I1=>t(2),
78             I2 =>t(4),
79             I3 =>clock_50Mhz,
80                 O=>t(5));
81
82 FFT3 : FFT port map (
83     T => t(5),
```

```
84      CLK => clock_50Mhz,
85      Q => t(6),
86      reset => reset
87      );
88
89  Y <= t(6) & t(4) & t(2) & t(0);
90
91  end Structural;
```

Codice Componente 6.2: Definizione del contatore parallelo

6.2.2.3 Contatore in serie Mod-16

```
1
2
3
4   library IEEE;
5   use IEEE.STD_LOGIC_1164.ALL;
6    use ieee.std_logic_unsigned.all;
7    library UNISIM;
8   use UNISIM.Vcomponents.ALL;
9
10  entity contatore_serie is
11
12  port (
13      clock_50Mhz : in std_logic;
14      RESET : in std_logic;
15      Y : out std_logic_vector(3 downto 0)
16      );
17
18  end contatore_serie;
19
20  architecture Structural of contatore_serie is
21
22  Component FFT
23       port ( CLK : in    std_logic;
24            T   : in    std_logic;
25         reset : in std_logic;
26            Q   : out   std_logic
27         );
28  end Component;
29
30  signal t:  std_logic_vector (3 downto 0) := (others => '0') ;
31
32
33  begin
34
35
```

```
36 FFT0 : FFT port map (
37     T => '1',
38     CLK => clock_50Mhz,
39     Q => t(0),
40     reset => reset
41     );
42
43 FFT1 : FFT port map (
44     T => '1',
45     CLK => t(0),
46     Q => t(1),
47     reset => reset
48     );
49
50 FFT2 : FFT port map (
51     T => '1',
52     CLK => t(1),
53     Q => t(2),
54
55     reset => reset
56     );
57
58 FFT3 : FFT port map (
59     T => '1',
60     CLK => t(2),
61     Q => t(3),
62     reset => reset
63     );
64
65 Y <= t;
66
67 end Structural;
```

Codice Componente 6.3: Definizione del contatore in serie

6.3 Simulazione

È stato eseguito lo stesso testbench per entrambi i contatori. Si riporta di seguito quello relativo al contatore parallelo. Il conteggio viene fatto partire per 100 ns e poi viene testato il reset.

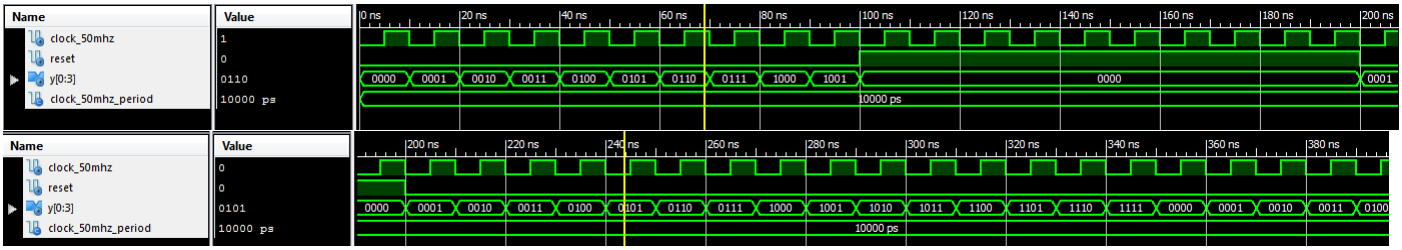
```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3
4
5 ENTITY test IS
6 END test;
7
8 ARCHITECTURE behavior OF test IS
```

```
9
10
11      COMPONENT contatore_parallelo
12      PORT(
13           clock_50Mhz : IN  std_logic;
14           RESET : IN  std_logic;
15           Y : OUT  std_logic_vector(0 to 3)
16          );
17      END COMPONENT;
18
19
20     --Inputs
21     signal clock_50Mhz : std_logic := '0';
22     signal RESET : std_logic := '0';
23
24    --Outputs
25     signal Y : std_logic_vector(0 to 3);
26
27     -- Clock period definitions
28     constant clock_50Mhz_period : time := 10 ns;
29
30  BEGIN
31
32     uut: contatore_parallelo PORT MAP (
33            clock_50Mhz => clock_50Mhz,
34            RESET => RESET,
35            Y => Y
36           );
37
38     -- Clock process definitions
39     clock_50Mhz_process :process
40     begin
41      clock_50Mhz <= '0';
42      wait for clock_50Mhz_period/2;
43      clock_50Mhz <= '1';
44      wait for clock_50Mhz_period/2;
45     end process;
46
47
48     stim_proc: process
49     begin
50
51          wait for 100 ns;
52      reset <= '1';
53      wait for 100 ns;
54      reset <= '0';
55
56        wait;
57     end process;
```

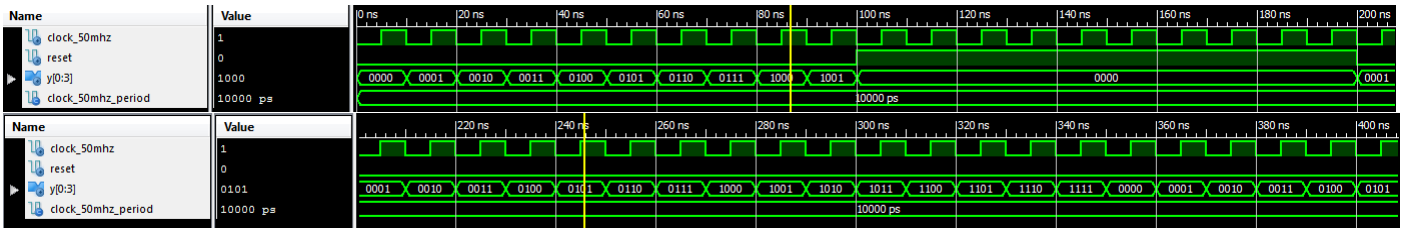
```
58
59 END;
```

Codice Componente 6.4: Definizione del testbench contatore parallelo

6.3.0.1 Test Contatore Parallelo



6.3.0.2 Test Contatore Serie

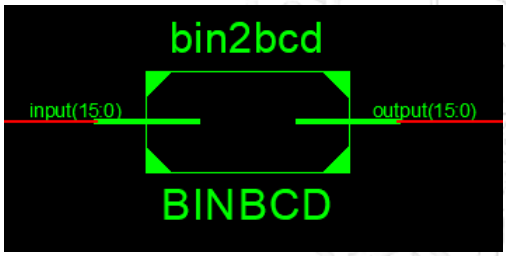


6.4 Sintesi su board FPGA

Per la sintesi su FPGA del contatore in serie, sono stati importati diversi componenti, tra cui: display a 7 segmenti, debouncer e componente bin2bcd.

6.4.1 Bin2bcd

Il componente bint2bcd permette di ottenere, a partire da un input binario, un output che può essere mostrato sul display a 7 segmenti in digitale.



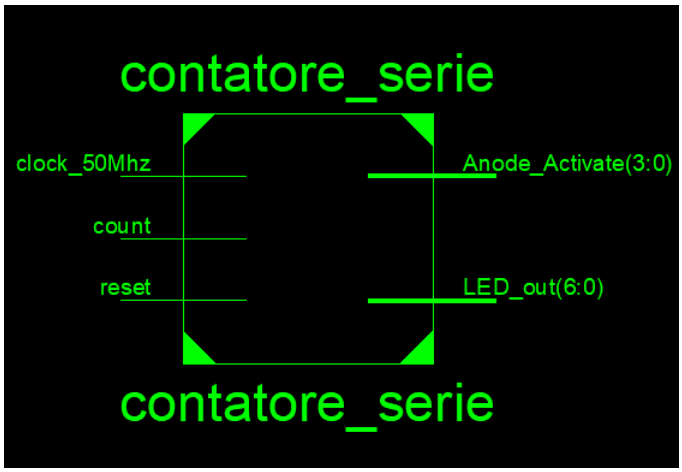

```
1
2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.numeric_std.all;
5 use IEEE.std_logic_unsigned.all;
```

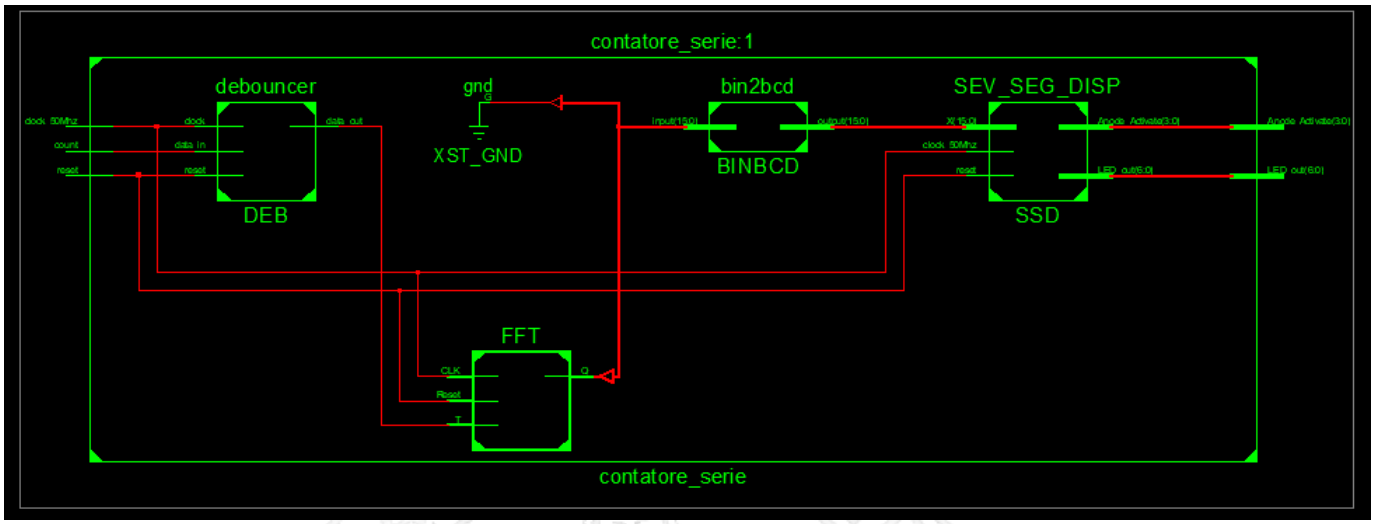
```
6
7  entity bin2bcd is
8      port (  input:      in   std_logic_vector (15 downto 0);
9          output: out std_logic_vector (15 downto 0)
10 );
11 end entity;
12
13 architecture behavioural of bin2bcd is
14     alias data: std_logic_vector (15 downto 0) is input;
15
16 begin
17    process (data)
18         variable bcd:   std_logic_vector (15 downto 0);
19         variable bint:  std_logic_vector (13 downto 0);
20     begin
21         bcd := (others => '0');
22         bint := data (13 downto 0);
23
24         for i in 0 to 13 loop
25             bcd(15 downto 1) := bcd(14 downto 0);
26             bcd(0) := bint(13);
27             bint(13 downto 1) := bint(12 downto 0);
28             bint(0) := '0';
29
30             if i < 13 and bcd(3 downto 0) > "0100" then
31                bcd(3 downto 0) :=  (bcd(3 downto 0)) + 3;
32             end if;
33             if i < 13 and bcd(7 downto 4) > "0100" then
34                bcd(7 downto 4) := (bcd(7 downto 4)) + 3;
35             end if;
36             if i < 13 and bcd(11 downto 8) > "0100" then
37                bcd(11 downto 8) := (bcd(11 downto 8)) + 3;
38           end if;
39             if i < 13 and bcd(15 downto 12) > "0100" then
40                 bcd(11 downto 8) := (bcd(15 downto 12)) + 3;
41            end if;
42         end loop;
43
44     output<= bcd;
45
46 end process ;
47 end architecture;
```

Codice Componente 6.5: Definizione bin2bcd

6.4.2 Sintesi finale

Il contatore presenta in ingresso il clock, il reset e il segnale di conteggio; le uscite sono quelle del display a 7 segmenti.






```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use ieee.std_logic_unsigned.all;
5  library UNISIM;
6  use UNISIM.Vcomponents.ALL;
7
8  entity contatore_serie is
9
10 port (
11     clock_50Mhz : in std_logic;
12     reset : in std_logic;
13     count: in std_logic;
14     Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
15         LED_out : out STD_LOGIC_VECTOR (6 downto 0)
16     );
17 end contatore_serie;
18
```

```
19 architecture Structural of contatore_serie is
20
21 Component FFT
22      port ( CLK : in    std_logic;
23               T   : in    std_logic;
24         reset : in std_logic;
25               Q   : out   std_logic
26 );
27 end Component;
28
29 Component debouncer
30     port (  clock, reset: in std_logic;
31         data_in: in std_logic;
32         data_out: out std_logic
33 );
34 end Component;
35
36 component bin2bcd is
37     port (
38         input: in  std_logic_vector (15 downto 0);
39         output: out std_logic_vector (15 downto 0)
40 );
41 end component;
42
43 component SEV_SEG_DISP is
44     Port ( clock_50Mhz : in STD_LOGIC;
45            reset : in STD_LOGIC;
46       X : in STD_LOGIC_VECTOR (15 downto 0);
47            Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
48            LED_out : out STD_LOGIC_VECTOR (6 downto 0)
49 );
50 end component;
51
52 signal t:  std_logic_vector (3 downto 0) := (others => '0');
53 signal value_to_display : std_logic_vector (15 downto 0);
54 signal value_converted : std_logic_vector (15 downto 0);
55 signal c: std_logic :='0';
56 constant ZERO : std_logic_vector(11 downto 0) := (others => '0');
57
58 begin
59
60 FFT0 : FFT port map (
61     T => c,
62     CLK =>clock_50Mhz,
63     Q => t(0),
64     reset => reset
65     );
66
67 FFT1 : FFT port map (
```

```
68      T => '1',
69      CLK => t(0),
70      Q => t(1),
71      reset => reset
72      );
73
74  FFT2 : FFT port map (
75      T => '1',
76      CLK => t(1),
77      Q => t(2),
78      reset => reset
79      );
80
81  FFT3 : FFT port map (
82      T => '1',
83      CLK => t(2),
84      Q => t(3),
85      reset => reset
86      );
87
88  BINBCD : bin2bcd port map(
89      input => value_to_display,
90      output => value_converted);
91
92  SSD : SEV_SEG_DISP port map(
93      clock_50Mhz => clock_50Mhz,
94      reset => reset,
95      Anode_Activate => Anode_Activate,
96      LED_out => LED_out,
97      X => value_converted );
98
99  DEB: DEBOUNCER port map(
100     clock=>clock_50Mhz,
101     reset=>reset,
102     data_in=>count,
103     data_out=>c
104 );
105
106 value_to_display <= ZERO & t;
107 end Structural;
```

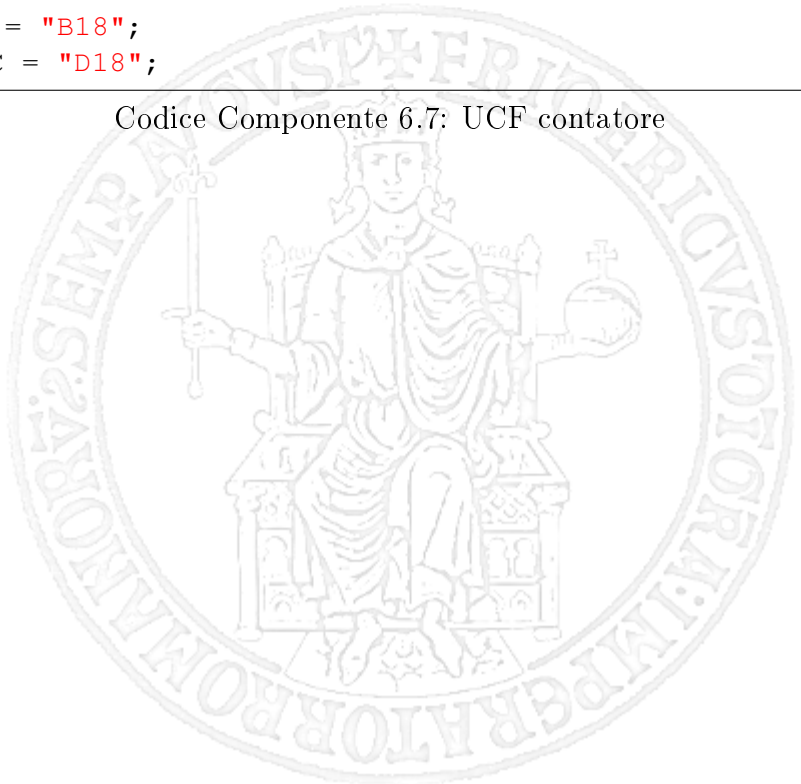
Codice Componente 6.6: Sintesi del contatore in serie

6.4.3 File UCF

```
1
2   NET "LED_out<6>" LOC = "L18"; # Bank = 1, Pin name = IO_L10P_1, Type = I/O,
        Sch name = CA
```

```
3  NET "LED_out<5>" LOC = "F18"; # Bank = 1, Pin name = IO_L19P_1, Type = I/O,
       Sch name = CB
4  NET "LED_out<4>" LOC = "D17"; # Bank = 1, Pin name = IO_L23P_1/HDC, Type =
       DUAL, Sch name = CC
5  NET "LED_out<3>" LOC = "D16"; # Bank = 1, Pin name = IO_L23N_1/LDC0, Type =
       DUAL, Sch name = CD
6  NET "LED_out<2>" LOC = "G14"; # Bank = 1, Pin name = IO_L20P_1, Type = I/O,
       Sch name = CE
7  NET "LED_out<1>" LOC = "J17"; # Bank = 1, Pin name = IO_L13P_1/A6/RHCLK4/
       IRDY1, Type = RHCLK/DUAL, Sch name = CF
8  NET "LED_out<0>" LOC = "H14"; # Bank = 1, Pin name = IO_L17P_1, Type = I/O,
       Sch name = CG
9
10 NET "clock_50Mhz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
        = GCLK, Sch name = GCLK0
11 NET "Anode_Activate<0>" LOC = "F17"; # Bank = 1, Pin name = IO_L19N_1, Type
       = I/O, Sch name = AN0
12 NET "Anode_Activate<1>" LOC = "H17"; # Bank = 1, Pin name = IO_L16N_1/A0,
       Type = DUAL, Sch name = AN1
13 NET "Anode_Activate<2>" LOC = "C18"; # Bank = 1, Pin name = IO_L24P_1/LDC1,
       Type = DUAL, Sch name = AN2
14 NET "Anode_Activate<3>" LOC = "F15"; # Bank = 1, Pin name = IO_L21P_1, Type
       = I/O, Sch name = AN3
15
16
17 NET "count" LOC = "B18";
18 NET "reset"  LOC = "D18";
```

Codice Componente 6.7: UCF contatore



Capitolo 7

Esercizio 7

7.1 Traccia

Progettare ed implementare in VHDL un "arbitro 2 su 3", ossia un componente che, presi due input binari in ingresso, fornisce in uscita un valore binario pari a quello che compare almeno 2 volte su 3 in ingresso. Sintetizzare sulla board il componente utilizzando gli switch per acquisire gli ingressi e un led per visualizzare il risultato.

7.2 Soluzione

Il funzionamento dell'arbitro 2 su 3 da implementare è descritto nella seguente tabella di verità

A0	A1	A2	U
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

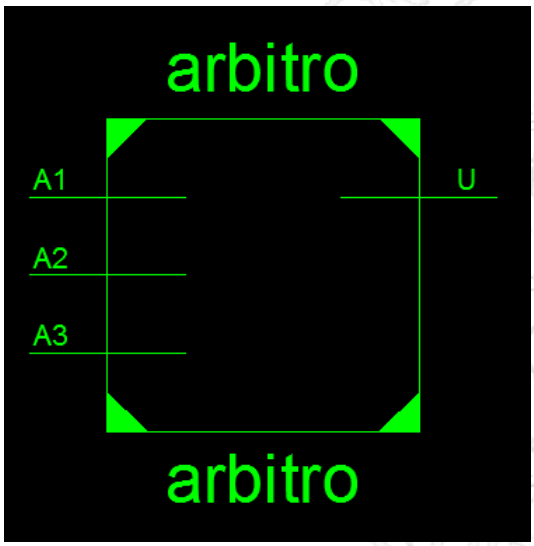
Il calcolo della funzione minimizzata tramite mappa di Karnaugh risulta essere :

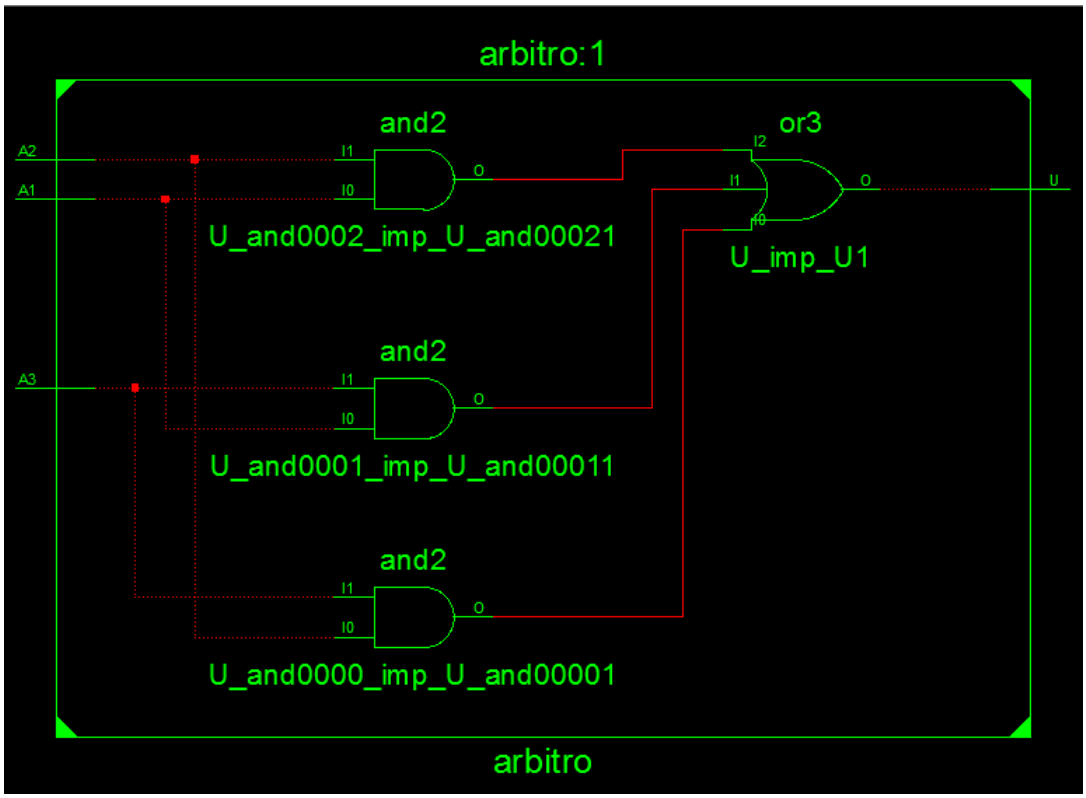
$U = A_2A_3 \vee A_1A_3 \vee A_1A_2$
É stato utilizzato il tool sis per la verifica dell' espressione ottenuta:

```
UC Berkeley, SIS 1.2 (compiled May 21 2002 09:23:42)
sis> read_blif arbitro.blif
sis> write_blif
.model arbitro
.inputs A1 A2 A3
.outputs U
.names A1 A2 A3 U
111 1
011 1
101 1
110 1
.end
sis> write_eqn
INORDER = A1 A2 A3;
OUTORDER = U;
U = A1*A2*!A3 + A1*!A2*A3 + !A1*A2*A3 + A1*A2*A3;
sis> print_stats
arbitro  pi= 3  po= 1  node=  1  latch= 0  lits(sop)=  12  lits(ff)=   9
sis> simplify
sis> write_eqn
INORDER = A1 A2 A3;
OUTORDER = U;
U = A2*A3 + A1*A3 + A1*A2;
sis> print_stats
arbitro  pi= 3  po= 1  node=  1  latch= 0  lits(sop)=   6  lits(ff)=   5
sis>
```

7.2.1 Schematici

Lo schematico presenta 3 ingressi A1, A2 ed A3 ed un'unica uscita U, che è alta se vi sono almeno due bit pari a '1' in ingresso.





7.2.2 Codice

L'espressione minimizzata è stata tradotta con una descrizione dataflow.

7.2.2.1 Arbitro 2 su 3

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity arbitro is
5    Port ( A1 : in STD_LOGIC;
6           A2 : in STD_LOGIC;
7           A3 : in STD_LOGIC;
8           U : out STD_LOGIC
9    );
10 end arbitro;
11
12 architecture Dataflow of arbitro is
13
14 begin
15
16 U <= (A2 AND A3) OR (A1 AND A3) OR (A1 AND A2);
17
18 end Dataflow;
```

Codice Componente 7.1: Definizione del componente Arbitro 2 su 3

7.3 Simulazione

Nel caso in esame, essendo 8 le combinazioni totali in input da verificare possiamo fornire un testbench completo. I risultati attesi corrispondono a quelli della tabella di verità mostrata nel paragrafo precedente.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY test IS
5  END test;
6
7  ARCHITECTURE behavior OF test IS
8
9      COMPONENT arbitro
10     PORT(
11          A1 : IN  std_logic;
12          A2 : IN  std_logic;
13          A3 : IN  std_logic;
14          U : OUT  std_logic
15         );
16     END COMPONENT;
17
18
19    --Inputs
20    signal A1 : std_logic := '0';
21    signal A2 : std_logic := '0';
22    signal A3 : std_logic := '0';
23
24   --Outputs
25    signal U : std_logic;
26
27
28 BEGIN
29
30    uut: arbitro PORT MAP (
31           A1 => A1,
32           A2 => A2,
33           A3 => A3,
34           U => U
35         );
36
37
38    stim_proc: process
39    begin
40       -- hold reset state for 100 ns.
41       wait for 100 ns;
42     A1 <= '0';
43     A2 <= '0';
```

```
44      A3 <= '0';
45      wait for 100 ns;
46      A1 <= '0';
47      A2 <= '0';
48      A3 <= '1';
49      wait for 100 ns;
50      A1 <= '0';
51      A2 <= '1';
52      A3 <= '0';
53      wait for 100 ns;
54      A1 <= '0';
55      A2 <= '1';
56      A3 <= '1';
57      wait for 100 ns;
58      A1 <= '1';
59      A2 <= '0';
60      A3 <= '0';
61      wait for 100 ns;
62      A1 <= '1';
63      A2 <= '0';
64      A3 <= '1';
65      wait for 100 ns;
66      A1 <= '1';
67      A2 <= '1';
68      A3 <= '0';
69      wait for 100 ns;
70      A1 <= '1';
71      A2 <= '1';
72      A3 <= '1';
73
74        wait;
75    end process;
76
77  END;
```

Codice Componente 7.2: Definizione del testbench per Arbitro 2 su 3

I risultati del test mostrano come varia U rispetto agli ingressi A1,A2,A3. Da una semplice analisi si evince che la macchina si comporta da arbitro 2 su 3.

7.4 Sintesi su board FPGA

Per sintetizzare la macchina su schedino NEXYS 2 è sufficiente mappare i segnali A1,A2,A3 con gli switch SW0,SW1,SW2 e l'uscita U col led LED0.

```
1 NET "A1"        LOC = "G18";    # Sch name = SW0
2 NET "A2"        LOC = "H18";    # Sch name = SW1
3 NET "A3"        LOC = "K18";    # Sch name = SW2
4
5 NET "U" LOC = "J14"; #Sch name LD0
```

Codice Componente 7.3: Definizione del file UCF

Capitolo 8

Esercizio 8

8.1 Traccia

Progettare ed implementare un orologio che, a partire da un clock di riferimento che opera da base dei tempi di adeguata precisione, genera mediante uso di contatori il secondo, il minuto e l'ora. L' orologio deve essere sintetizzato su FPGA e la visualizzazione dell'ora deve sfruttare le 4 cifre del display e i led messi a disposizione dalla board di sviluppo, secondo la seguente modalità:

• i minuti (da 1 a 60) e le ore (da 1 a 24) sono visualizzati in formato 8-4-2-1 mediante le due cifre rispettivamente meno e più significative del display a 4 cifre a sette segmenti;

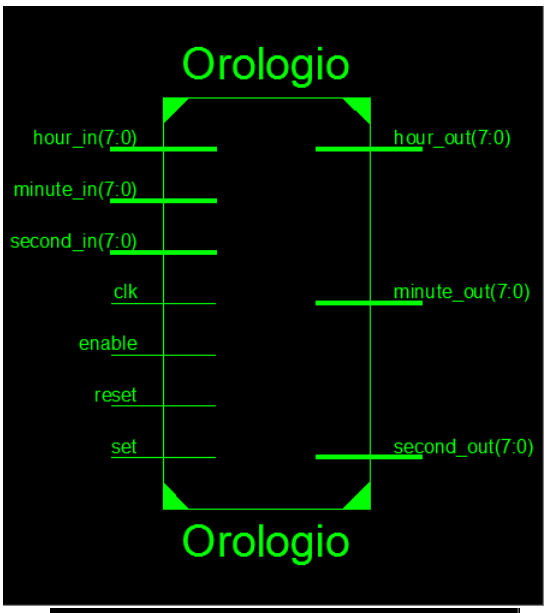
• i secondi (da 1 a 60) sono visualizzati utilizzando i quattro led di peso meno significativo dell'array di led presenti nella scheda

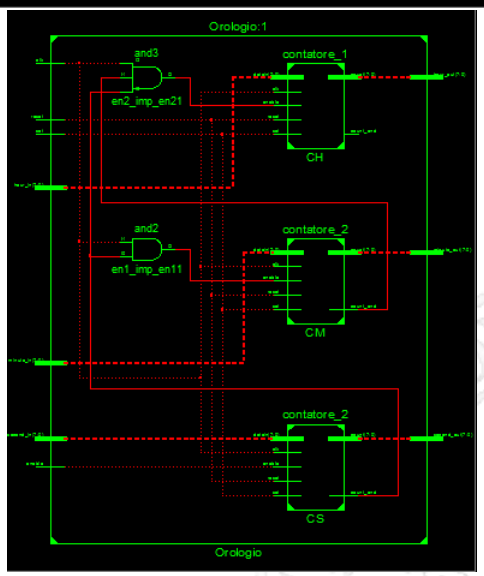
Il valore del tempo deve poter essere inizializzato acquisendo, in sequenza e tramite gli switch, i valori dell'ora, del minuti e dei secondi.

8.2 Soluzione e sintesi

L'orologio viene implementato con 3 contatori collegati in parallelo. Essi contano ore, minuti e secondi (rappresentati su 8 bit) e vengono inizializzati tramite il segnale logico di set. In particolare set stabilisce il valore di conteggio corrente del singolo contatore e, quando il segnale di enable sarà alto, partirà poi a contare fino al valore impostato tramite interfaccia generic. Al raggiungimento di questo valore, viene generato un segnale di fine conteggio e scatta il contatore successivo. Il segnale di fine conteggio (count_end) uscente dal contatore viene posto in AND col clock di base per consentire la sincronizzazione tra i vari componenti.

8.2.1 Schematici





8.2.2 Codice

Il componente principale Orologio connette tra loro i tre contatori e imposta il valore limite di conteggio tramite la loro interfaccia generic. Come già detto essi sono tra loro connessi tramite porte logiche AND2 e AND3, che ne gestitscono la sincronizzazione.

8.2.2.1 Orologio

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use ieee.numeric_std.all;
5 use IEEE.std_logic_unsigned.all;
```

```
6
7
8  entity Orologio is
9    Port ( hour_in, minute_in, second_in : in std_logic_vector (7 downto 0);
10          clk,enable,reset,set : in STD_LOGIC;
11         hour_out, minute_out, second_out : out std_logic_vector (7 downto 0)
12   );
13 end Orologio;
14
15 architecture Behavioral of Orologio is
16
17 Component contatore
18 Generic (N: integer);
19 port(
20
21     reset : in STD_LOGIC;
22     clk,enable,set : in STD_LOGIC;
23     datain: in STD_LOGIC_VECTOR(7 downto 0);
24     count : out STD_LOGIC_VECTOR(7 downto 0);
25     count_end : out STD_LOGIC
26
27 );
28 end component;
29
30 signal second :std_logic_vector (7 downto 0) := (others=>'0');
31 signal minute :std_logic_vector (7 downto 0) := (others=>'0');
32 signal hour :std_logic_vector (7 downto 0) := (others=>'0');
33 signal en1,en2,cc1,cc2,cc3: std_logic:='0';
34
35 begin
36
37
38 CH:contatore generic map(24) port map(
39   reset=>reset,
40   clk=>clk,
41   enable=>en2,
42   set=>set,
43   datain=>hour_in,
44   count=>hour,
45   count_end=>cc3
46
47 );
48
49
50 CM:contatore generic map(60) port map(
51   reset=>reset,
52   clk=>clk,
53   enable=>en1,
54   set=>set,
```

```
55    datain=>minute_in,
56    count=>minute,
57    count_end=>cc2
58
59  );
60
61
62
63  CS:contatore generic map(60) port map(
64    reset=>reset,
65    clk=>clk,
66    enable=>enable,
67    set=>set,
68    datain=>second_in,
69    count=>second,
70    count_end=>cc1
71
72  );
73
74
75  en1<=cc1 and clk;
76  en2<=cc1 and cc2 and clk;
77  second_out <= second;
78  minute_out <= minute;
79  hour_out <= hour;
80
81
82  end Behavioral;
```

Codice Componente 8.1: Definizione del componente Orologio

8.2.2.2 Contatore

Il contatore prende in ingresso diversi segnali, tra cui enable per iniziare il conteggio e set per impostarne il valore di partenza preso in ingresso da datain. L'interfaccia generic ci consente di stabilire un limite di conteggio, dopodichè il contatore riparte da 0. Il segnale di reset pone a 0 il conteggio e il valore di count_end.

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.NUMERIC_STD.ALL;
5   use IEEE.std_logic_arith. all;
6   use IEEE.MATH_REAL. all;
7
8
9   entity contatore is
10  Generic (N: integer);
11  Port (
```

```
12      reset : in STD_LOGIC;
13      clk,enable,set : in STD_LOGIC;
14      datain: in STD_LOGIC_VECTOR(7 downto 0);
15      count : out STD_LOGIC_VECTOR(7 downto 0);
16      count_end : out STD_LOGIC
17  );
18  end contatore;
19
20  architecture Behavioral of contatore is
21
22  begin
23
24  process(clk,reset,enable)
25  variable cnt: integer range 0 to N-1:=0;
26
27  begin
28
29  if (reset='1') then
30    cnt:=0;
31    count_end<='0';
32
33  elsif (set='1') then
34    cnt:= TO_INTEGER(unsigned(datain));
35
36  elsif falling_edge(clk) then
37
38    if(enable='1')then
39      if (cnt=N-1) then
40      cnt:=0;
41      count_end<='1';
42      else
43      cnt:= cnt+1;
44      count_end<='0';
45      end if;
46
47    end if;
48
49  end if;
50
51
52  count<=CONV_STD_LOGIC_VECTOR (cnt,8);
53
54  end process;
55
56  end Behavioral;
```

Codice Componente 8.2: Definizione del componente counter

8.2.2.3 Orologio FPGA

I componenti usati per la sintetizzazione sono : Display a 7 segmenti, convertitore bin to bcd, il counter visto precedentemente e il divisore di clock. I segnali vengono inseriti tramite gli switch (codificati su 8 bit) e acquisiti tramite i bottoni della board. La pressione finale del pulsante pone il segnale di enable a 1 e dà inizio al conteggio, partendo dai valori che sono stati immessi.
Il divisore di clock, il cui codice è mostrato in questa sezione, prende in ingresso il clock a 50 Mhz della board e restituisce in uscita un clock le cui finestre temporali sono pari a 1 secondo. Tale delay viene creato tramite un segnale temporaneo che, quando il contatore arriva a 25'000'000, viene negato. Tale valore limite di conteggio viene calcolato attraverso il fattore di scala :

$$Scale = \frac{f_{in}}{f_{out}}$$

La macchina a stati implementata nel process è composta da 4 stati (A, B, C, D). I primi 3 permettono di inserire rispettivamente ore, minuti e secondi usando gli switch della board e premendo l'apposito bottone di load (BTN_NEXT). Successivamente lo stato D pone ad '1' il segnale Enable_out del componente counter che da inizio al conteggio. Sono stati usati due bin2bcd che prendono in ingresso i valori di output del componente counter (ore e minuti) e li convertono. Questi valori sono poi posti in ingresso al display a 7 segmenti. I secondi vengono semplicemente mostrati sui led. Il segnale di reset pone a 0 Enable_out, ore, minuti e secondi e riporta la macchina nello stato A.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity orologioFPGA is
6    Port ( BTN_NEXT : in STD_LOGIC;
7           BTN_RESET : in STD_LOGIC;
8          clock_50Mhz : in STD_LOGIC;
9          X : in STD_LOGIC_VECTOR(7 downto 0);
10         Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);-- 4 Anode signals
11            LED_out : out STD_LOGIC_VECTOR (6 downto 0);
12         LDS  : out STD_LOGIC_VECTOR (3 downto 0)
13
14   );
15 end orologioFPGA;
16
17 architecture Behavioral of orologioFPGA is
18
19 type State_type is (A, B, C, D);  -- Define the states
20 signal State : State_Type := A;
21
22
23 signal hour_t, minute_t, second_t : STD_LOGIC_VECTOR(7 downto 0):= (others
       =>'0');
24 signal enable_out_t : STD_LOGIC := '0';
25 signal set_t : STD_LOGIC := '1';
```

```
26 signal clock_div, c1 : STD_LOGIC := '0';
27
28 signal h,m,s : std_logic_vector (7 downto 0) := (others =>'0');
29
30 signal hour_conv,min_conv :  STD_LOGIC_VECTOR(11 downto 0 ) := (others
       =>'0');
31
32 component Orologio is
33   Port ( hour_in, minute_in,second_in  : in std_logic_vector (7 downto 0);
34          clk, enable,reset,set : in STD_LOGIC;
35        hour_out, minute_out, second_out: out std_logic_vector (7 downto 0)
36   );
37 end component;
38
39
40 Component debouncer
41   port(
42   clock, reset: in std_logic;
43   data_in: in std_logic;
44   data_out: out std_logic
45
46   );
47 end Component;
48
49 Component Clock_Divider
50 port (
51   clk,reset: in std_logic;
52   clock_out: out std_logic
53 );
54 end Component;
55
56 component bin2bcd is
57     port (
58         bin:    in  std_logic_vector (7 downto 0);
59         bcd:    out std_logic_vector (11 downto 0)
60     );
61 end component;
62
63 component SEV_SEG_DISP is
64     Port ( clock_50Mhz : in STD_LOGIC;
65            reset : in STD_LOGIC; -- reset
66            Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);-- 4 Anode
           signals
67            LED_out : out STD_LOGIC_VECTOR (6 downto 0);
68         X : in STD_LOGIC_VECTOR (15 downto 0)
69         );
70
71 end component;
72
```

```
73
74
75  begin
76
77
78  CT : Orologio port map(
79    hour_in => hour_t,
80    minute_in => minute_t,
81    second_in => second_t,
82    enable => enable_out_t,
83    set=> set_t,
84    clk => clock_div,
85    reset => BTN_RESET,
86
87    hour_out => h,
88    minute_out => m,
89    second_out => s
90  );
91
92  CD : Clock_Divider port map (
93    clk => clock_50Mhz,
94    clock_out => clock_div,
95    reset => BTN_RESET
96
97  );
98
99
100 DEB: debouncer port map(
101   clock=> clock_50MHz,
102   reset=> BTN_RESET,
103   data_in=>BTN_NEXT,
104   data_out=>c1
105
106 );
107
108
109 B2BCD : bin2bcd port map (
110   bin => h,
111   bcd => hour_conv
112 );
113
114
115 B2BCD2 : bin2bcd port map (
116   bin => m,
117   bcd => min_conv
118 );
119
120 SEG : SEV_SEG_DISP port map (
121   clock_50Mhz => clock_50Mhz,
```

```
122   reset => BTN_RESET,
123   Anode_Activate => Anode_Activate,
124   LED_out => LED_out,
125 
126   X(15) => hour_conv(7),
127   X(14) => hour_conv(6),
128   X(13) => hour_conv(5),
129   X(12) => hour_conv(4),
130   X(11) => hour_conv(3),
131   X(10) => hour_conv(2),
132   X(9) => hour_conv(1),
133   X(8) => hour_conv(0),
134 
135   X(7) => min_conv(7),
136   X(6) => min_conv(6),
137   X(5) => min_conv(5),
138   X(4) => min_conv(4),
139   X(3) => min_conv(3),
140   X(2) => min_conv(2),
141   X(1) => min_conv(1),
142   X(0) =>min_conv(0)
143 
144 );
145 
146 process (clock_div, BTN_RESET)
147 begin
148 
149 
150 
151  if (BTN_RESET='1') then
152         hour_t <= (others =>'0');
153         minute_t <= (others =>'0');
154         second_t <=(others =>'0');
155         enable_out_t <= '0';
156         set_t<='1';
157         State<=A;
158  elsif (rising_edge(clock_50MHz)) then
159   case State is
160       when A =>
161         if (c1='1') then
162          if (X > "00010111") then
163           State<=A;
164          else
165           hour_t <= X;
166           State<=B;
167          end if;
168 
169         end if;
170       when B=>
```

```
171
172         if (c1='1') then
173
174         if (X > "00111011") then
175           State<=B;
176          else
177           minute_t <= X;
178           State<=C;
179          end if;
180
181
182         end if;
183       when C=>
184         if (c1='1') then
185
186         if (X > "00111011") then
187           State<=C;
188          else
189           second_t <= X;
190           State<=D;
191          end if;
192
193
194         end if;
195       when D=>
196         if (c1='1') then
197           set_t<='0';
198           enable_out_t <= '1';
199         end if;
200
201   end case;
202 end if;
203   LDS <= s(3 downto 0);
204
205 end process;
206
207
208
209
210 end Behavioral;
```

Codice Componente 8.3: Definizione della sintesi dell'orologio su FPGA

8.2.2.4 Clock Divider

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
```

```
4  use IEEE.numeric_std.ALL;
5
6  entity Clock_Divider is
7  port (
8    clk,reset: in std_logic;
9    clock_out: out std_logic);
10 end Clock_Divider;
11
12 architecture bhv of Clock_Divider is
13
14 signal count: integer:=1;
15 signal tmp : std_logic := '0';
16
17 begin
18
19 process(clk,reset)
20 begin
21   if(reset='1') then
22     count<=1;
23     tmp<='0';
24 elsif(clk'event and clk='1') then
25     count <=count+1;
26 if (count = 25000000) then --  vedremo i valori in 1 secondo
27     tmp <= NOT tmp;
28     count <= 1;
29 end if;
30 end if;
31     clock_out <= tmp;
32 end process;
33
34 end bhv;
```

Codice Componente 8.4: Definizione del file Clock Divider

8.2.2.5 Test UCF

Vengono impostati gli switch, i 4 led per i secondi, il display a 7 segmenti e i bottoni.

```
1
2  NET "LED_out<6>" LOC = "L18"; # Bank = 1, Pin name = IO_L10P_1, Type = I/O,
       Sch name = CA
3  NET "LED_out<5>" LOC = "F18"; # Bank = 1, Pin name = IO_L19P_1, Type = I/O,
       Sch name = CB
4  NET "LED_out<4>" LOC = "D17"; # Bank = 1, Pin name = IO_L23P_1/HDC, Type =
       DUAL, Sch name = CC
5  NET "LED_out<3>" LOC = "D16"; # Bank = 1, Pin name = IO_L23N_1/LDC0, Type =
       DUAL, Sch name = CD
6  NET "LED_out<2>" LOC = "G14"; # Bank = 1, Pin name = IO_L20P_1, Type = I/O,
       Sch name = CE
```

```
7  NET "LED_out<1>" LOC = "J17"; # Bank = 1, Pin name = IO_L13P_1/A6/RHCLK4/
       IRDY1, Type = RHCLK/DUAL, Sch name = CF
8  NET "LED_out<0>" LOC = "H14"; # Bank = 1, Pin name = IO_L17P_1, Type = I/O,
       Sch name = CG
9
10 NET "clock_50Mhz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
       = GCLK, Sch name = GCLK0
11 NET "Anode_Activate<0>" LOC = "F17"; # Bank = 1, Pin name = IO_L19N_1, Type
       = I/O, Sch name = AN0
12 NET "Anode_Activate<1>" LOC = "H17"; # Bank = 1, Pin name = IO_L16N_1/A0,
       Type = DUAL, Sch name = AN1
13 NET "Anode_Activate<2>" LOC = "C18"; # Bank = 1, Pin name = IO_L24P_1/LDC1,
       Type = DUAL, Sch name = AN2
14 NET "Anode_Activate<3>" LOC = "F15"; # Bank = 1, Pin name = IO_L21P_1, Type
       = I/O, Sch name = AN3
15 NET "X<0>"        LOC = "G18";    # Sch name = SW0
16 NET "X<1>"        LOC = "H18";    # Sch name = SW1
17 NET "X<2>"        LOC = "K18";    # Sch name = SW2
18 NET "X<3>"        LOC = "K17";    # Sch name = SW3
19 NET "X<4>"       LOC = "L14";
20 NET "X<5>"       LOC = "L13";
21 NET "X<6>"       LOC = "N17";
22 NET "X<7>"       LOC = "R17";
23
24
25 NET "BTN_NEXT" LOC = "B18";
26 NET "BTN_RESET"  LOC = "D18";
27
28 NET "LDS<0>" LOC = "J14";
29 NET "LDS<1>" LOC = "J15";
30 NET "LDS<2>" LOC = "K15";
31 NET "LDS<3>" LOC = "K14";
```

Codice Componente 8.5: Definizione del file UCF

8.3 Simulazione

Per l'orologio verranno presentati i casi di test più significativi. Nel primo caso si verifica che allo scadere del minuto 23:59:00, il contatore riprende il conteggio dalle ore 00:00:00. Nel secondo caso si mostra che posto il segnale di enable a 0 durante il conteggio, esso si blocca. Nel terzo caso viene mostrato il funzionamento del reset quando il segnale di enable è alto.

Ore	Minuti	Secondo	Enable_out	Reset
00010111 (23)	00111011 (59)	00000001 (1)	1	0
00010111(23)	00111011 (59)	00000001 (1)	1 -> 0	0
00001010(10)	00101011(43)	00001011(11)	1	1->0

Risultato atteso
Allo scadere del minuto, il contatore riparte da 00:00:00
Posto enable a 0, il conteggio si blocca
Posto reset a 0, il conteggio riparte

Il seguente codice mostra il testbench utilizzato. Si sceglie di non mostrare il codice per tutti i casi analizzati, poichè molto simili tra loro (basta modificare le variabili in ingresso nel process).

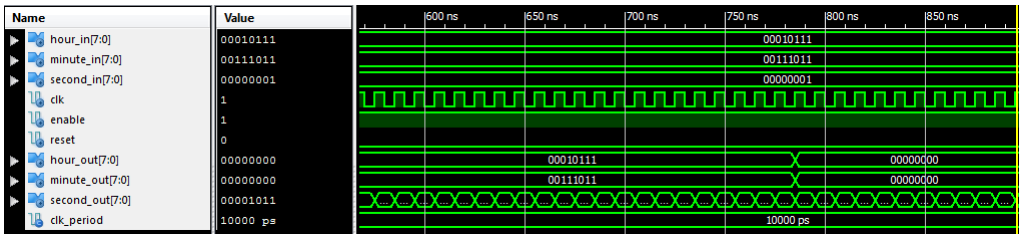
```
1  LIBRARY ieee;
2
3  LIBRARY ieee;
4  USE ieee.std_logic_1164.ALL;
5
6  ENTITY testc IS
7  END testc;
8
9  ARCHITECTURE behavior OF testc IS
10
11     -- Component Declaration for the Unit Under Test (UUT)
12
13     COMPONENT counter
14     PORT(
15          hour_in : IN  std_logic_vector(7 downto 0);
16          minute_in : IN  std_logic_vector(7 downto 0);
17          second_in : IN  std_logic_vector(7 downto 0);
18          clk : IN  std_logic;
19          enable : IN  std_logic;
20          reset : IN  std_logic;
21          hour_out : OUT  std_logic_vector(7 downto 0);
22          minute_out : OUT  std_logic_vector(7 downto 0);
23          second_out : OUT  std_logic_vector(7 downto 0)
24         );
25     END COMPONENT;
26
27
28    --Inputs
29    signal hour_in : std_logic_vector(7 downto 0) := (others => '0');
30    signal minute_in : std_logic_vector(7 downto 0) := (others => '0');
31    signal second_in : std_logic_vector(7 downto 0) := (others => '0');
32    signal clk : std_logic := '0';
33    signal enable : std_logic := '0';
34    signal reset : std_logic := '0';
35
36   --Outputs
37    signal hour_out : std_logic_vector(7 downto 0);
38    signal minute_out : std_logic_vector(7 downto 0);
39    signal second_out : std_logic_vector(7 downto 0);
40
41    -- Clock period definitions
```

```
42    constant clk_period : time := 10 ns;
43
44  BEGIN
45
46    -- Instantiate the Unit Under Test (UUT)
47    uut: counter PORT MAP (
48            hour_in => hour_in,
49            minute_in => minute_in,
50            second_in => second_in,
51            clk => clk,
52            enable => enable,
53            reset => reset,
54            hour_out => hour_out,
55            minute_out => minute_out,
56            second_out => second_out
57          );
58
59    -- Clock process definitions
60    clk_process :process
61    begin
62     clk <= '0';
63     wait for clk_period/2;
64     clk <= '1';
65     wait for clk_period/2;
66    end process;
67
68
69    -- Stimulus process
70    stim_proc: process
71    begin
72       -- hold reset state for 100 ns.
73       wait for 100 ns;
74
75
76     hour_in <= "00010111";
77     minute_in <= "00111011";
78     second_in <= "00000001";
79     wait for 100 ns;
80     enable <= '1';
81     wait for 300 ns;
82       enable <= '0';
83     wait;
84    end process;
85
86  END;
```

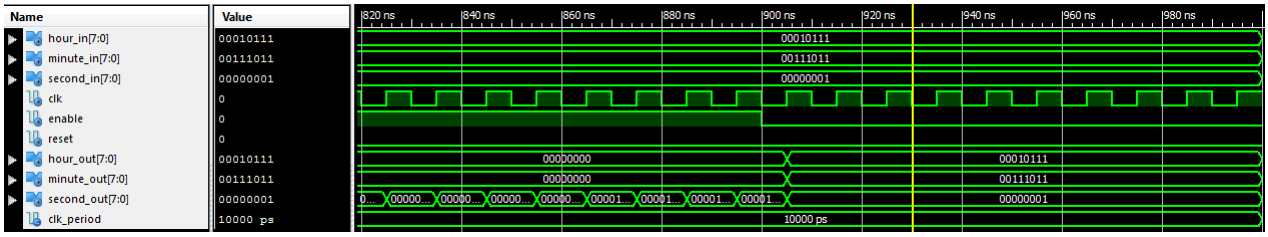
Codice Componente 8.6: Definizione del testbench per counter.vhd

Test 1

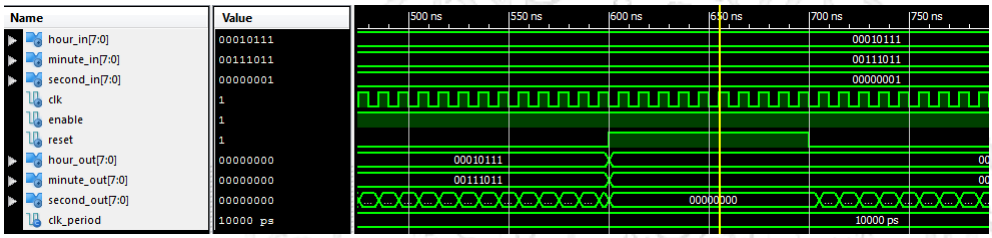
Si attende 1 minuto per verificare che il conteggio riparta da 0. Appena i secondi arrivano a 59 i valori di ore, minuti e secondi tornano a 0 e il conteggio riparte. Il test è superato correttamente e l'orologio si resetta allo scadere della mezzanotte.



Test 2
Si abbassa il valore di enable e si verifica che il conteggio si blocca al successivo colpo di clock (fronte di salita).



Test 3
Il reset funziona correttamente, impostato a 1 il valore di reset e dopo 100ns a 0 (con enable alto) i valori vengono resettati e il conteggio riparte immediatamente dopo.



Capitolo 9

Esercizio 9

9.1 Traccia

Progettare ed implementare in VHDL una macchina aritmetica combinatoria a scelta fra le seguenti:

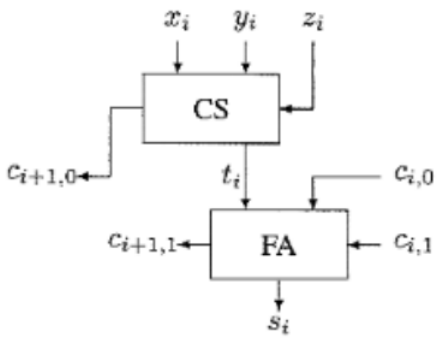
- adder carry look ahead, per effettuare la somma di 2 stringhe A e B da 8 bit ciascuna;
- carry save adder, per effettuare la somma di 3 stringhe A, B e C da 8 bit ciascuna;
- carry select adder, per effettuare la somma di 2 stringhe A e B da 16 bit ciascuna;
- moltiplicatore con somma per righe, per effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna;
- moltiplicatore con somma per diagonali, per effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna;
- moltiplicatore con somma per colonne, per effettuare il prodotto di 2 stringhe A e B da 6 bit ciascuna;
- moltiplicatore a celle MAC, per effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna.

In ogni caso, la macchina implementata deve essere sintetizzata su FPGA e deve poter essere testata mediante l'utilizzo dei dispositivi di input/output (switch, bottoni, led, display) presenti sulla board di sviluppo in dotazione al gruppo. La modalità di utilizzo degli stessi è a completa discrezione degli studenti.

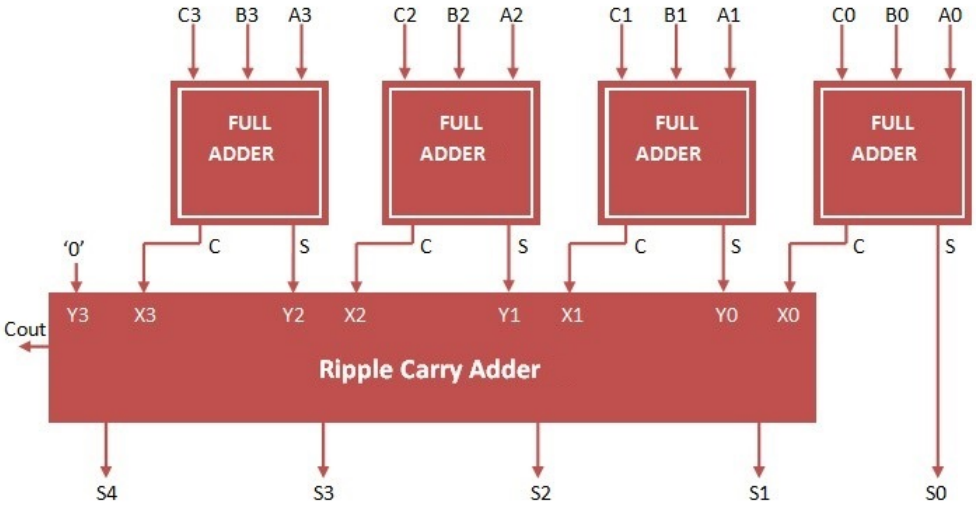
9.2 Soluzione

Il carry save adder è una macchina aritmetica costituita da blocchi carry save, che sommano i bit relativi agli operandi, e full adder. I full adder a valle prendono in ingresso i riporti generati nel livello superiore della somma dei bit i-esimi. I blocchi CS non sono soggetti alla propagazione del riporto, mentre i blocchi FA sono connessi a formare un sommatore ripple-carry. In tal modo è possibile realizzare la somma di un numero elevato di operandi, riducendo il ritardo totale complessivo.

$$s_i = (x_i + y_i + z_i) + c_{i,0} + c_{i,1}$$



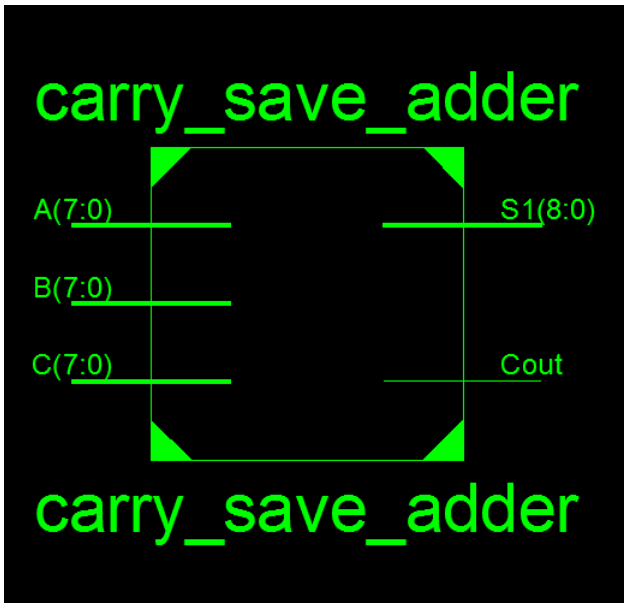
Per l'implementazione del carry save adder, è stato seguito il seguente modello:

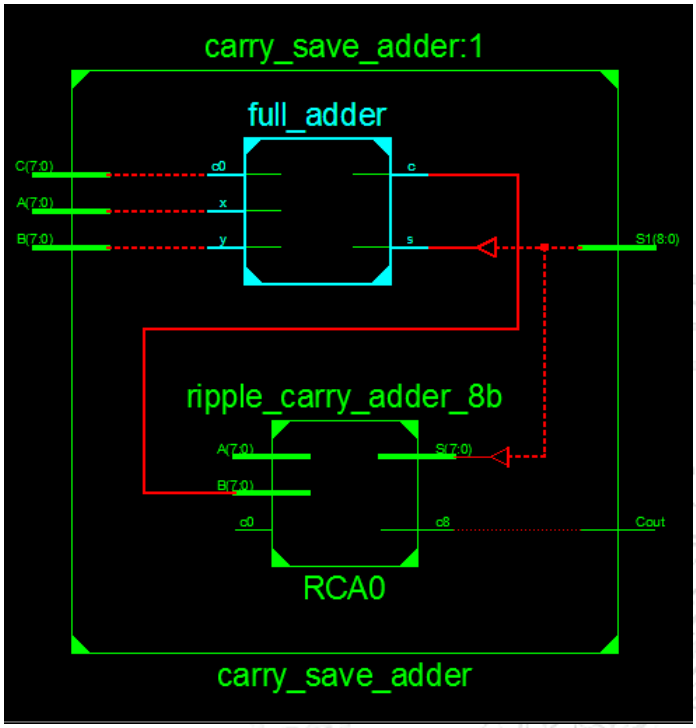


Dopo aver sommato i bit relativi alle 3 stringhe (A,B e C), le somme e i carry ottenuti dai full adder vengono messo in ingresso ad un ripple carry adder, ad eccezione della prima somma S0 che rappresenta il primo bit del risultato finale.

9.2.1 Schematici

Il carry save adder è composto da 8 full adder, che costituiscono la carry save logic e permettono di sommare gli 8 bit dei 3 operandi considerati, e un ripple carry adder su 8 bit, che restituisce somma e riporto finali.





9.2.2 Codice

Il codice prodotto di seguito mostra l'implementazione strutturale del Carry Save Adder, importando nel progetto i full adder e il ripple carry adder a 8 bit sviluppato nel capitolo 4.

9.2.2.1 Carry Save Adder

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity carry_save_adder is
6  Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
7  B : in STD_LOGIC_VECTOR (7 downto 0);
8  C : in STD_LOGIC_VECTOR (7 downto 0);
9  S1 : OUT STD_LOGIC_VECTOR (8 downto 0);
10 Cout : OUT STD_LOGIC);
11 end carry_save_adder;
12
13 architecture Behavioral of carry_save_adder is
14
15 component ripple_carry_adder_8b
16 PORT (
17       A : in  std_logic_vector (7 downto 0);
18       B : in  std_logic_vector(7 downto 0);
19       c0 : in std_logic;
20       c8 : out std_logic;
21       S: out std_logic_vector (7 downto 0)
22       );
23
24 end component;
25
26
27 component full_adder is
28   PORT (
29       x  : in  std_logic;
30       y : in std_logic;
31       c0 : in std_logic;
32       s : out std_logic;
33       c : out std_logic );
34
35 end component;
36
37
38
39 -- Intermediate signal
40 signal X,Y: STD_LOGIC_VECTOR(3 downto 0);
41 signal C1,C2,C3: STD_LOGIC;
42
43 signal carry : STD_LOGIC_VECTOR(7 downto 0);
44 signal sum : STD_LOGIC_VECTOR(7 downto 0);
45
46 begin
47
48 -- 8 full adder
```

```
49 FA0 : full_adder port map ( x => A(0), y => B(0), c0 => C(0), s => S1(0), c
       => carry(0) );
50 FA1 : full_adder port map ( x => A(1), y => B(1), c0 => C(1), s => sum(0), c
       => carry(1) );
51 FA2 : full_adder port map ( x => A(2), y => B(2), c0 => C(2), s => sum(1), c
       => carry(2) );
52 FA3 : full_adder port map ( x => A(3), y => B(3), c0 => C(3), s => sum(2), c
       => carry(3) );
53 FA4 : full_adder port map ( x => A(4), y => B(4), c0 => C(4), s => sum(3), c
       => carry(4) );
54 FA5 : full_adder port map ( x => A(5), y => B(5), c0 => C(5), s => sum(4), c
       => carry(5) );
55 FA6 : full_adder port map ( x => A(6), y => B(6), c0 => C(6), s => sum(5), c
       => carry(6) );
56 FA7 : full_adder port map ( x => A(7), y => B(7), c0 => C(7), s => sum(6), c
       => carry(7) );
57
58 -- 1 ripple carry adder a 8 bit
59 RCA0 : ripple_carry_adder_8b port map (
60     A(0) => sum(0),
61     A(1) => sum(1),
62     A(2) => sum(2),
63     A(3) => sum(3),
64     A(4) => sum(4),
65     A(5) => sum(5),
66     A(6) => sum(6),
67     A(7) => '0',
68     B => carry,
69     c0 => '0',
70     c8 => cout,
71     S(0) => S1(1),
72     S(1) => S1(2),
73     S(2) => S1(3),
74     S(3) => S1(4),
75     S(4) => S1(5),
76     S(5) => S1(6),
77     S(6) => S1(7),
78     S(7) => S1(8)
79     );
80
81
82 end Behavioral;
```

Codice Componente 9.1: Definizione del componente Carry Save Adder

9.3 Simulazione

Prima di sintetizzare il componente ne verifichiamo il corretto funzionamento attraverso dei test. A, B, C sono i 3 operandi in ingresso, SUM è la loro somma e Cout è l'eventuale valore di riporto. La seguente tabella mostra i casi di test analizzati e l'output atteso per ognuno di essi.

A	B	C	Cout	SUM
00000000	00000000	00000000	0	000000000
11111111	11111111	11111111	1	011111101
00000000	0000001	10101101	0	010101110
10101101	00000000	0000001	0	010101110
00101010	10000010	10101010	0	101010110

In decimale i test corrispondono a :

A	B	C	SUM
0	0	0	0
255	255	255	765
0	1	173	174
173	0	1	174
42	130	170	342

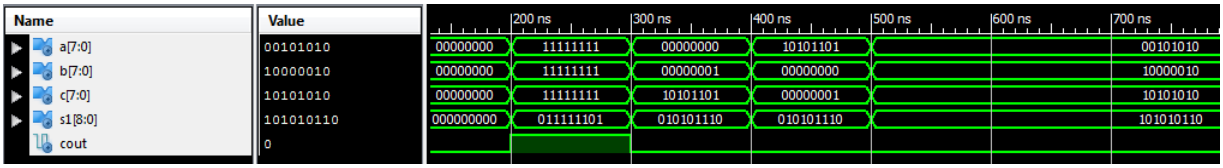
```
1
2
3  LIBRARY ieee;
4  USE ieee.std_logic_1164.ALL;
5
6  ENTITY testcsa IS
7  END testcsa;
8
9  ARCHITECTURE behavior OF testcsa IS
10
11     -- Component Declaration for the Unit Under Test (UUT)
12
13     COMPONENT carry_save_adder
14     PORT(
15          A : IN  std_logic_vector(7 downto 0);
16          B : IN  std_logic_vector(7 downto 0);
17          C : IN  std_logic_vector(7 downto 0);
18          S1 : OUT  std_logic_vector(8 downto 0);
19          Cout : OUT  std_logic
20         );
21     END COMPONENT;
22
23
24    --Inputs
25    signal A : std_logic_vector(7 downto 0) := (others => '0');
26    signal B : std_logic_vector(7 downto 0) := (others => '0');
```

```
27    signal C : std_logic_vector(7 downto 0) := (others => '0');
28
29   --Outputs
30    signal S1 : std_logic_vector(8 downto 0);
31    signal Cout : std_logic;
32    -- No clocks detected in port list. Replace <clock> below with
33    -- appropriate port name
34
35
36
37  BEGIN
38
39   -- Instantiate the Unit Under Test (UUT)
40    uut: carry_save_adder PORT MAP (
41           A => A,
42           B => B,
43           C => C,
44           S1 => S1,
45           Cout => Cout
46         );
47
48
49
50    -- Stimulus process
51    stim_proc: process
52    begin
53       -- hold reset state for 100 ns.
54
55     wait for 100 ns;
56     A <= "00000000";
57     B <= "00000000";
58     C <= "00000000";
59     wait for 100 ns;
60     A <= "11111111";
61     B <= "11111111";
62     C <= "11111111";
63     wait for 100 ns;
64     A <= "00000000";
65     B <= "00000001";
66     C <= "10101101";
67     wait for 100 ns;
68     A <= "10101101";
69     B <= "00000000";
70     C <= "00000001";
71     wait for 100 ns;
72     A <= "00101010";
73     B <= "10000010";
74     C <= "10101010";
75       wait for 100 ns;
```

```
76        wait;
77    end process;
78
79 END;
```

Codice Componente 9.2: Definizione del testbench per Carry Save Adder

I risultati dei test in ISim confermano i risultati attesi.



9.4 Sintesi su board FPGA

La sintesi su board per il carry save adder è simile a quella implementata per il semplice full adder. Ci serviamo del solito componente display a 7 segmenti (per mostrare i dati) e del carry save precedentemente sviluppato. Il processo di controllo prevede che l'utente inserisca in ordine i valori da sommare (codificati su 8 bit) attraverso gli switch. Questi poi sono caricati con la pressione dei bottoni presenti sulla board. Ogni volta che un valore viene immesso, il sistema controlla se sono stati inseriti anche gli altri 2 e calcola il risultato, che viene mostrato premendo il bottone di result. Durante tutta l'operazione i valori degli operandi vengono mostrati in tempo reale sul display a 7 segmenti.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use ieee.numeric_std.all;
5  use IEEE.std_logic_signed.all;
6
7
8  entity CarrySaveFPGA is
9
10  Port (   clock_50Mhz : in STD_LOGIC;
11             load1, load2,load3,result: in STD_LOGIC;
12        X : in STD_LOGIC_VECTOR (7 downto 0);
13             Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
14             LED_out : out STD_LOGIC_VECTOR (6 downto 0)
15 );
16
17 end CarrySaveFPGA;
18
19 architecture Behavioural of CarrySaveFPGA is
20
21
22 signal OP_1, OP_2, OP_3: boolean := false;
23
```

```
signal op1, op2,op3 : std_logic_vector (7 downto 0) := (others => '0') ;
signal sum : std_logic_vector (8 downto 0) := (others => '0') ;
signal zero :std_logic_vector (7 downto 0):= (others => '0') ;

signal cout : std_logic;

signal value_to_display: std_logic_vector (8 downto 0);



component carry_save_adder is
  Port (   A,B,C : in STD_LOGIC_VECTOR (7 downto 0);
        S1 : OUT STD_LOGIC_VECTOR (8 downto 0);
        Cout : OUT STD_LOGIC
);
end component;



component SEV_SEG_DISP is
    Port (
        X : in STD_LOGIC_VECTOR(8 downto 0);
        clock_50Mhz : in STD_LOGIC;
              reset : in STD_LOGIC;
              Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
              LED_out : out STD_LOGIC_VECTOR (6 downto 0)
);

end component;


begin

CSA : carry_save_adder port map(
    A => op1,
    B => op2,
    C => op3,
    Cout => cout,
    S1 => sum
);


SSD : SEV_SEG_DISP port map(
    X => value_to_display,
    clock_50Mhz => clock_50Mhz,
    reset => '0',
    Anode_Activate => Anode_Activate,
    LED_out => LED_out

```

```
);


process (clock_50Mhz)

begin

  if (rising_edge(clock_50Mhz)) then
    if load1 = '1' then
    -- Primo operando
        op1 <= X;
        OP_1 <= true;
        OP_2 <= false;
        OP_3 <= false;
    elsif load2 = '1' then
    -- Secondo operando
        op2 <= X;
        OP_2 <= true;
        OP_3 <= false;
    elsif load3 = '1' then
    -- Terzo operando
        op3 <= X;
        OP_3 <= true;
    end if;

    if (OP_1 = true and OP_2 = true and OP_3 = true) then
      if result = '1' then
      -- Se sono stati inseriti 3 operandi e se il bottone apposito viene
         premuto
        value_to_display <= sum;
      end if;
    else
      -- mostra in tempo reale i valori degli operandi inseriti
      value_to_display <= '0' & X;
    end if;

  end if;

end process;

end Behavioural;
```

Codice Componente 9.3: Sintesi carry save

9.4.1 File UCF

```

```

```
2  NET "LED_out<6>" LOC = "L18"; # Bank = 1, Pin name = IO_L10P_1, Type = I/O,
       Sch name = CA
3  NET "LED_out<5>" LOC = "F18"; # Bank = 1, Pin name = IO_L19P_1, Type = I/O,
       Sch name = CB
4  NET "LED_out<4>" LOC = "D17"; # Bank = 1, Pin name = IO_L23P_1/HDC, Type =
       DUAL, Sch name = CC
5  NET "LED_out<3>" LOC = "D16"; # Bank = 1, Pin name = IO_L23N_1/LDC0, Type =
       DUAL, Sch name = CD
6  NET "LED_out<2>" LOC = "G14"; # Bank = 1, Pin name = IO_L20P_1, Type = I/O,
       Sch name = CE
7  NET "LED_out<1>" LOC = "J17"; # Bank = 1, Pin name = IO_L13P_1/A6/RHCLK4/
       IRDY1, Type = RHCLK/DUAL, Sch name = CF
8  NET "LED_out<0>" LOC = "H14"; # Bank = 1, Pin name = IO_L17P_1, Type = I/O,
       Sch name = CG
9
10 NET "clock_50Mhz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
       = GCLK, Sch name = GCLK0
11 NET "Anode_Activate<0>" LOC = "F17"; # Bank = 1, Pin name = IO_L19N_1, Type
       = I/O, Sch name = AN0
12 NET "Anode_Activate<1>" LOC = "H17"; # Bank = 1, Pin name = IO_L16N_1/A0,
       Type = DUAL, Sch name = AN1
13 NET "Anode_Activate<2>" LOC = "C18"; # Bank = 1, Pin name = IO_L24P_1/LDC1,
       Type = DUAL, Sch name = AN2
14 NET "Anode_Activate<3>" LOC = "F15"; # Bank = 1, Pin name = IO_L21P_1, Type
       = I/O, Sch name = AN3
15
16 NET "X<0>"        LOC = "G18";    # Sch name = SW0
17 NET "X<1>"        LOC = "H18";    # Sch name = SW1
18 NET "X<2>"        LOC = "K18";    # Sch name = SW2
19 NET "X<3>"        LOC = "K17";    # Sch name = SW3
20 NET "X<4>"       LOC = "L14";
21 NET "X<5>"       LOC = "L13";
22 NET "X<6>"       LOC = "N17";
23 NET "X<7>"       LOC = "R17";
24
25 NET "load1" LOC = "B18";
26 NET "load2"  LOC = "D18";
27 NET "load3"  LOC = "E18";
28 NET "result"  LOC = "H13";
```

Codice Componente 9.4: F

Capitolo 10

Esercizio 10

10.1 Traccia

Progettare ed implementare in VHDL una macchina aritmetica sequenziale a scelta fra le seguenti:

- moltiplicatore di Robertson, per effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna;
- moltiplicatore di Booth, per effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna;
- divisore non-restoring, per effettuare la divisione intera fra due stringhe A e B di 4 bit ciascuna;
- divisore restoring, per effettuare la divisione intera fra due stringhe A e B di 4 bit ciascuna;

In ogni caso, la macchina implementata deve essere sintetizzata su FPGA e deve poter essere testata mediante l'utilizzo dei dispositivi di input/output (switch, bottoni, led, display) presenti sulla board di sviluppo in dotazione al gruppo. La modalità di utilizzo degli stessi è a completa discrezione degli studenti.

10.2 Soluzione

Il moltiplicatore di Robertson è un moltiplicatore sequenziale, che riceve in ingresso due operandi ad N bit codificati in complementi a due e restituisce un prodotto su 2N bit. Tale moltiplicatore sfrutta questa notazione:

$$X = -2^{n-1}x^{n-1} + \sum_{i=0}^{n-2} 2^i x_i$$

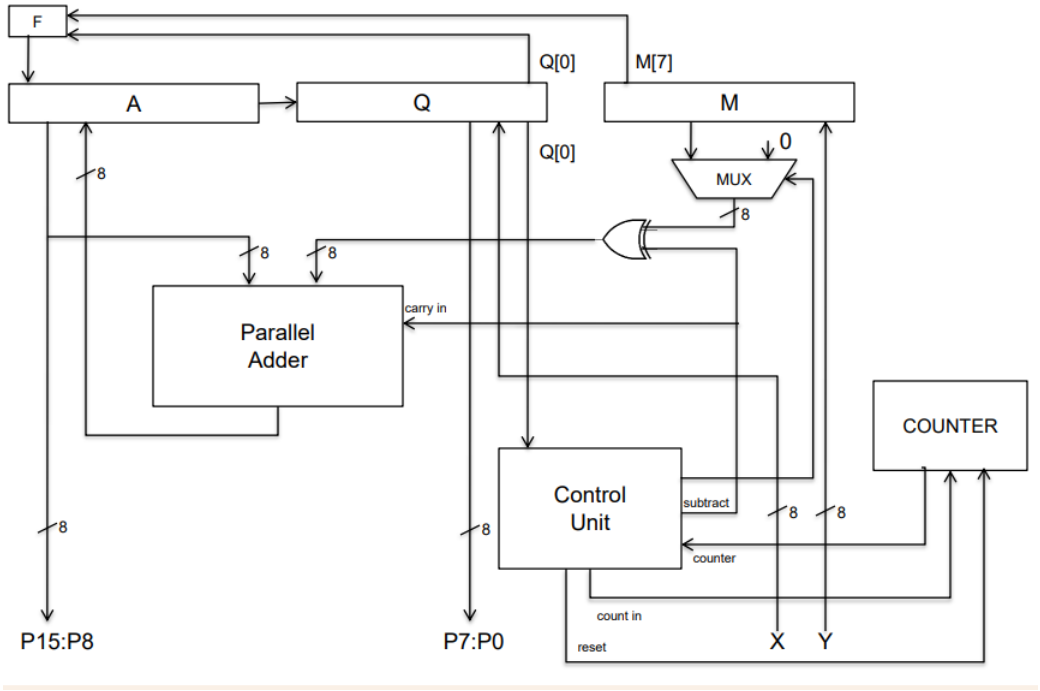
Per un numero positivo viene assegnato peso 2^{n-1} al bit di segno x_{n-1} : poiché è nullo il suo contributo allo 0 sulla cifra più significativa. Per un numero negativo viene assegnato peso -2^{n-1} al bit di segno x_{n-1}: poiché esso vale 1 il suo contributo al numero è -1 sulla cifra più significativa. Tale metodo ci consente di utilizzare una tecnica unsigned facendo attenzione ad effettuare una sottrazione invece di una addizione quando si incontra un segno negativo. A tal scopo è stato implementato un componente che effettua sia l'addizione che la sottrazione di due operandi a seconda del valore di un bit (0,1) in ingresso.
Riassumiamo dunque i 4 casi possibili :

- X $>$ 0 e Y $>$ 0 : caso standard, moltiplicazione tra unsigned fatta con addizione e shift.
- X $>$ 0 e Y $<$ 0 : quando si moltiplica Y per il j-esimo valore di X diverso da 0, il prodotto parziale è negativo e quindi il bit più significativo dell'accumulatore A diventa 1.
- X $<$ 0 e Y $>$ 0 : per l'ultimo prodotto va effettuato un ulteriore operazione di correzione A - M.
- X $<$ 0 e Y $<$ 0 : anche in tal caso è necessario per l'ultimo prodotto l'operazione di correzione A - M. Il bit più significativo di A resta 0 e diventa 1 quando il j-esimo valore di X è 1.

L'algoritmo è mostrato di seguito :

```
2CMultiplier:   (in:INBUS; OUT:OUTBUS)
                register A[7:0],M[7:0],Q[7:0],COUNT[2:0],F;
                bus INBUS[7:0],OUTBUS[7:0];

BEGIN:          A:=0,COUNT:=0,F:=0,
INPUT:          M:=INBUS;Q:=INBUS;

ADD:            A[7:0]:= A[7:0] + M [7:0] x Q[0],
                F:= (M[7] and Q[0]) or F;
RSHIFT:         A[7]:= F,   A[6:0].Q:= A.Q[7:1],
INCREMENT:      COUNT:=COUNT+1

TEST:           if COUNT<7 then go to ADD;

SUBTRACT:       A[7:0]:=A[7:0]-M[7:0]xQ[0];      {l'ultima op è sempre SUB}
RSHIFT:         A[7]:= A[7],   A[6:0].Q:= A.Q[7:1];

OUTPUT:         OUTBUS;=Q; OUTBUS:=A;
END 2CMultiplier;
```

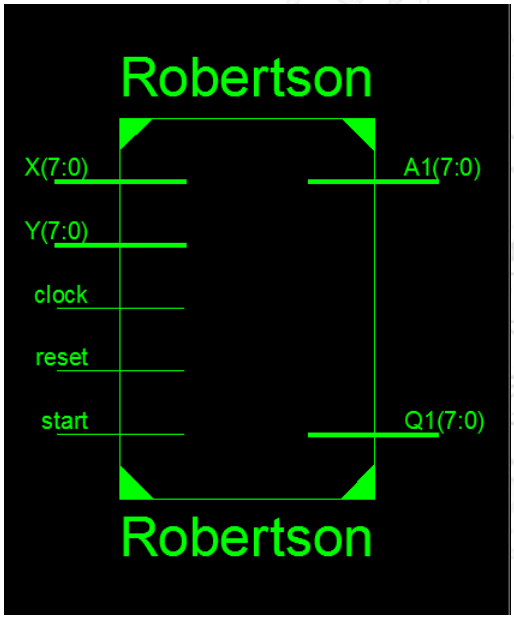
L'algoritmo inizia ponendo in M e Q i due operandi X e Y (inbus) e inizializzando il registro A (accumulatore), il contatore e il registro F.
L'addizione si esegue tra A ed M e si pone il risultato nel registro A. Successivamente viene effettuata l'operazione F =(M(7) and Q(0)) or F, che serve a gestire il caso 2 (moltiplicando negativo e moltiplicatore positivo). Il valore di F viene posto nel bit più significativo di A e il contenuto di quest'ultimo viene shiftato a destra in modo tale che il bit Q(1) si trovi in Q(0) per la prossima operazione. Il contatore delle operazioni viene incrementato e fin quando rimane minore di 7 le operazioni vanno avanti. Lo stato di correzione viene richiamato quando il contatore si ferma e viene fatta la correzione tramite sottrazione di A ed M.



La control unit ha il compito di abilitare e disabilitare i componenti adibiti alle operazioni in base allo stato corrente. La parte di controllo dunque implementa una macchina a stati che gestisce il comportamento dei registri.

10.2.1 Schematici

Lo schematico mostra i due ingressi X e Y (moltiplicatore e moltiplicando) il clock, il reset e il segnale di start che abilita la periferica ad effettuare l'operazione. L'uscita di 16 bit è suddivisa in due segnali A1 e Q1 rappresentanti gli 8 bit rispettivamente più e meno significativi.



10.2.2 Codice

10.2.2.1 Moltiplicatore di Robertson

Tutti i componenti vengono collegati tra loro, come da figura, nel codice sottostante. Il flop flop D è il registro F mostrato in figura, nel segnale tD viene messa la and tra M(7) e Q(0). t_sr_a è il primo bit più significativo shiftato in A e prende il contenuto di F oppure A(7) nell'ultimo passaggio (per la correzione finale). t_mux_sel è il segnale del multiplexer che serve a prendere il contenuto del moltiplicando quando bisogna effettuare addizione o sottrazione e il bit Q(0) è pari ad 1. t_shift abilita lo shift di A e Q a destra nei registri a scorrimento.

```
1
2
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6  use ieee.numeric_std.all;
7  use IEEE.std_logic_unsigned.all;
8
9
10 entity Robertson is
11   Port (
12     clock : in  STD_LOGIC;
13     X,Y : in  STD_LOGIC_VECTOR (7 downto 0);
14     start,reset : in STD_LOGIC;
15     A1,Q1: out STD_LOGIC_VECTOR (7 downto 0)
16
17   );
18 end Robertson;
19
20 architecture Structural of Robertson is
21
22
23 component PC
24 Port (
25   clock : in STD_LOGIC;
26   start,reset : in STD_LOGIC;
27   count: in STD_LOGIC_VECTOR(2 downto 0);
28   Q0 : in STD_LOGIC;
29   mux_en,reg_en,shift_en,fshift_en : out STD_LOGIC;
30   sub : out STD_LOGIC;
31   A_en : out STD_LOGIC;
32   enablecount: out STD_LOGIC;
33   resetcount: out STD_LOGIC
34 );
35 end component;
36
37
38 Component FlipFlopD
39 port(
```

```
40      D: in STD_LOGIC;
41      clock: in STD_LOGIC;
42      enable,reset : in STD_LOGIC;
43      Q,Qn : out STD_LOGIC
44
45  );
46  end Component;
47
48
49  Component addsub
50  port(
51      X,Y : in STD_LOGIC_VECTOR (7 downto 0);
52      subtract : in STD_LOGIC;
53      cout : out STD_LOGIC;
54      s : out STD_LOGIC_VECTOR (7 downto 0);
55      ov : out STD_LOGIC
56
57  );
58  end Component;
59
60
61  Component shiftregister
62  port(
63
64    clock : in STD_LOGIC;
65    reset : in STD_LOGIC;
66    shift : in STD_LOGIC;
67    load : in STD_LOGIC;
68    serial_in : in STD_LOGIC;
69    parallel_in : in STD_LOGIC_VECTOR (7 downto 0);
70    serial_out : out STD_LOGIC;
71    parallel_out : out STD_LOGIC_VECTOR (7 downto 0)
72
73  );
74  end Component;
75
76
77  Component registro
78  port(
79
80    clock : in STD_LOGIC;
81    reset : in STD_LOGIC;
82    enable: in STD_LOGIC;
83    datain : in STD_LOGIC_VECTOR (7 downto 0);
84    dataout : out STD_LOGIC_VECTOR (7 downto 0)
85
86  );
87  end Component;
88
```

```
89
90  Component muxb
91  port(
92
93        a0,b0: in  std_logic_vector(7 downto 0);
94        s0 : in  std_logic;
95        y : out std_logic_vector(7 downto 0)
96
97  );
98  end Component;
99
100
101 Component counter
102 port(
103   clock : in STD_LOGIC;
104   enable : in STD_LOGIC;
105   reset: in STD_LOGIC;
106   count : out STD_LOGIC_VECTOR (2 downto 0)
107 );
108 end component;
109
110
111 signal tQ, t_a_in, uscita_a, tM, uscita_mux : std_logic_vector(7 downto 0)
      :=(others=>'0');
112 signal mux_enable, reg_enable, t_shift_en, t_fshift_en, sub_enable,
      t_Aenable :std_logic :='0';
113 signal resetcount,enablecount,t_mux_sel, tD, tF, tFn, tshift, t_load, t_sr_a
      , t_sr_q :std_logic :='0';
114 signal cnt : std_logic_vector(2 downto 0):=(others=>'0');
115
116
117 begin
118
119 tshift <= t_shift_en or t_fshift_en;
120 t_load <= t_Aenable or sub_enable;
121 t_mux_sel <= mux_enable and (sub_enable or t_Aenable);
122
123 t_sr_a <= tF when t_shift_en = '1' else
124         '0' when (t_fshift_en='1' and tD='1') else
125         uscita_a(7) when t_fshift_en ='1' else
126         '0';
127
128 tD <= (tM(7) and tQ(0));
129
130 A1<= uscita_a;
131 Q1<= tQ;
132
133
134 cu: PC port map(clock,start,reset,cnt,tQ(0),mux_enable, reg_enable,
```

```
    t_shift_en, t_fshift_en, sub_enable, t_Aenable, enablecount, resetcount)
    ;
135
136 adder1: addsub port map(uscita_a,uscita_mux,sub_enable, open, t_a_in, open);
137
138 mux1: muxb  port map ("00000000", tM, t_mux_sel, uscita_mux);
139
140 ff1 : FlipFlopD port map(tD, clock, tFn , reset, tF, tFn);
141
142 regM: registro port map(clock, reset, reg_enable, Y, tM);
143
144 regA: shiftregister port map(clock, reset, tshift, t_load,t_sr_a, t_a_in,
    t_sr_q, uscita_a);
145
146 regQ: shiftregister port map(clock, reset, tshift, reg_enable, t_sr_q, X,
    open, tQ);
147
148 contatore: counter port map (clock,enablecount,resetcount,cnt);
149
150 end Structural;
```

Codice Componente 10.1: Definizione del moltiplicatore di Robertson

10.2.2.2 PC

La parte di controllo implementa la macchina a stati di cui si è discusso in precedenza. Gli stati in tutto sono 6 : Idle, Add, Shift, Sub, Finale(ultimo shift) ed Output. Lo stato di Idle abilita i registri e resetta il contatore delle operazioni. Tutti gli altri componenti restano disabilitati fino a quando lo stato di start non è alto. Quando start è alto, se il primo bit di Q(0) è 1, si effettua una addizione; altrimenti si fa semplicemente lo shift. Nello stato di add si abilita il registro A (accumulatore) per memorizzare il risultato e si passa allo stato di shift per ottenere il nuovo Q(0). Lo stato di shift abilita i registri a scorrimento e il contatore aggiunge 1 al conteggio delle operazioni. Se il contatore è arrivato a 7 (maggiore di 110), lo stato successivo è substate che effettua la correzione e porta allo stato finale e di output, che mostra poi il risultato. Se invece il contatore non è ancora arrivato al limite massimo, si torna allo stato di add.

```
1
2 --
    ----------------------------------------------------------------------------
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5
6 -- Uncomment the following library declaration if using
7 -- arithmetic functions with Signed or Unsigned values
8 --use IEEE.NUMERIC_STD.ALL;
9
10 -- Uncomment the following library declaration if instantiating
11 -- any Xilinx primitives in this code.
```

```
12 --library UNISIM;
13 --use UNISIM.VComponents.all;
14
15 entity PC is
16 port(
17   clock : in STD_LOGIC;
18   start,reset : in STD_LOGIC;
19   count: in STD_LOGIC_VECTOR(2 downto 0);
20   Q0 : in STD_LOGIC;
21   mux_en,reg_en,shift_en,fshift_en : out STD_LOGIC;
22   sub : out STD_LOGIC;
23   A_en : out STD_LOGIC;
24   enablecount: out STD_LOGIC;
25   resetcount: out STD_LOGIC
26 );
27 end PC;
28
29 architecture Behavioral of PC is
30
31
32 type STATE_type is (idle, add, shift, substate, finale, output);
33 signal State: STATE_type := idle;
34
35 begin
36
37 mux_en <= Q0;
38
39
40
41 process (clock,start,reset,Q0,State,count)
42 begin
43
44 if rising_edge(clock) then
45 case State is
46
47 when idle =>
48   resetcount <= '1';
49   reg_en <= '1';
50   shift_en <= '0';
51   fshift_en <= '0';
52   A_en <= '0';
53   sub <= '0';
54   enablecount <='0';
55
56   if (start = '1' and Q0 = '1') then
57     State <= add;
58   elsif (start = '1' and Q0 = '0') then
59     State <= shift;
60   else
```

```
61      State <= idle;
62    end if;
63
64  when add =>
65    reg_en <= '0';
66    shift_en <= '0';
67    fshift_en <= '0';
68    A_en <= '1';
69    resetcount <= '0';
70    sub <= '0';
71    enablecount <='0';
72
73    if reset = '1' then
74      State <= idle;
75    else
76      State <= shift;
77    end if;
78
79  when shift =>
80    reg_en <= '0';
81    shift_en <= '1';
82    fshift_en <= '0';
83    A_en <= '0';
84    resetcount <= '0';
85    sub <= '0';
86    enablecount <= '1';
87
88    if reset = '1' then
89      State <=idle;
90    elsif count = "110" then
91      State <= substate;
92    else
93      State <= add;
94    end if;
95
96  when substate =>
97    reg_en <= '0';
98    shift_en <= '0';
99    fshift_en <= '0';
100   A_en <= '0';
101   resetcount <= '0';
102
103   sub <= '1';
104   enablecount <= '0';
105
106   if reset = '1' then
107     State <=idle;
108   else
109     State <= finale;
```

```
110   end if;
111
112
113 when finale =>
114   shift_en <= '0';
115   fshift_en <= '1';
116   reg_en <= '0';
117   A_en <= '0';
118   resetcount <= '0';
119   sub <= '0';
120   enablecount <= '0';
121
122   if reset = '1' then
123     State <=idle;
124   else
125     State <= output;
126   end if;
127
128
129 when output =>
130   resetcount <= '0';
131   shift_en <= '0';
132   fshift_en <= '0';
133   reg_en <= '0';
134   A_en <= '0';
135   sub <= '0';
136   enablecount <= '0';
137
138   if (reset = '1' or start='0') then
139     State <=idle;
140   else
141     State <= output;
142   end if;
143
144
145 end case;
146
147
148 end if;
149 end process;
150
151
152 end Behavioral;
```

Codice Componente 10.2: Definizione della Parte di Controllo

10.2.2.3 Registro

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164. ALL;
4
5  entity registro is
6
7  Port(
8    clock : in STD_LOGIC;
9    reset : in STD_LOGIC;
10   enable: in STD_LOGIC;
11   datain : in STD_LOGIC_VECTOR (7 downto 0);
12   dataout : out STD_LOGIC_VECTOR (7 downto 0)
13   );
14 end registro;
15
16 architecture behavioral of registro is
17
18 begin
19
20 process (clock, reset, enable)
21
22 variable d: STD_LOGIC_VECTOR (7 downto 0):=(others => '0');
23
24 begin
25
26 if (reset = '1') then
27   d := (others => '0');
28
29 elsif (enable ='1' and rising_edge(clock)) then
30   d := datain;
31
32 else
33   d := d;
34 end if;
35
36 dataout <= d;
37
38 end process;
39
40 end behavioral;
```

Codice Componente 10.3: Definizione del registro

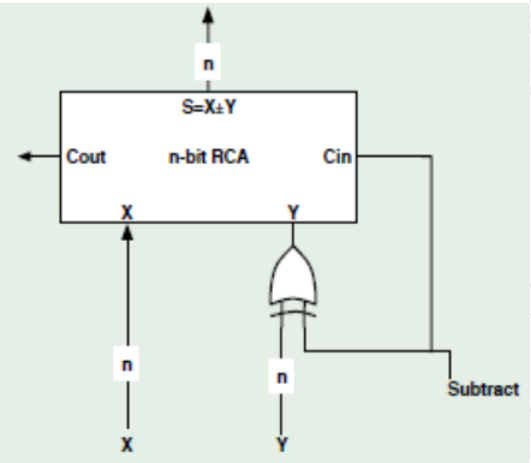
10.2.2.4 Mux Bus

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
```

```
4
5
6  entity muxb is
7    PORT (
8        a0,b0: in  std_logic_vector(7 downto 0);
9        s0 : in  std_logic;
10       y : out std_logic_vector(7 downto 0)
11 );
12 end muxb;
13
14
15 architecture dataflow of muxb is
16
17 begin
18
19 y<= a0 when (s0 = '0') else
20    b0 when (s0 = '1') else
21    "00000000";
22
23 end dataflow;
```

Codice Componente 10.4: Definizione del mux bus

10.2.2.5 Adder - Subtractor



Il componente Adder-Subtractor, dati due operandi, ci permette di realizzare sia un' addizione che una sottrazione.
Quando il segnale subtract è '1' l'addizionatore prende in ingresso X e Y negato. Mettendo '1' in C0 al ripple carry adder (implementazione nei precedenti capitoli) si ottiene l'operazione X-Y.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164. ALL;
4
5
```

```
6  entity addsub is
7  Port (
8      X,Y : in STD_LOGIC_VECTOR (7 downto 0);
9      subtract : in STD_LOGIC;
10     cout : out STD_LOGIC;
11     s : out STD_LOGIC_VECTOR (7 downto 0);
12     ov : out STD_LOGIC
13     );
14 end addsub;
15
16 architecture Structural of addsub is
17
18 component ripple_carry_adder_8b is
19 Port(
20
21     A,B : in  std_logic_vector (7 downto 0);
22     c0 : in std_logic;
23     c8 : out std_logic;
24     S: out std_logic_vector (7 downto 0)
25
26 );
27 end component;
28
29 signal btemp,stemp : STD_LOGIC_VECTOR (7 downto 0);
30
31 begin
32
33
34 process (Y, subtract)
35 begin
36
37   for i in 0 to 7 loop
38   btemp(i) <= Y(i) xor subtract;
39   end loop;
40
41 end process;
42
43 adder: ripple_carry_adder_8b port map (X,btemp,subtract,cout,stemp);
44
45 ov <= '1' when (X(7)='0' and btemp(7)='0' and stemp(7)='1') or (X(7)='1' and
       btemp(7)='1' and stemp(7)='0') else
46       '0';
47 s <= stemp;
48
49 end Structural;
```

Codice Componente 10.5: Definizione del Adder - Subtractor

10.3 Simulazione

Il testbench istanzia il componente moltiplicatore di Robertson e, dopo aver caricato i due operandi, l'operazione viene avviata mettendo ad 1 il segnale di start.

X	Y	Output atteso
00000011	00001000	0000000000011000
11111110	00001000	1111111111110000
00001000	11111110	1111111111110000
11111100	11111101	0000000000001100
111111111	111111111	0000000000000001
10000000	10000000	0100000000000000

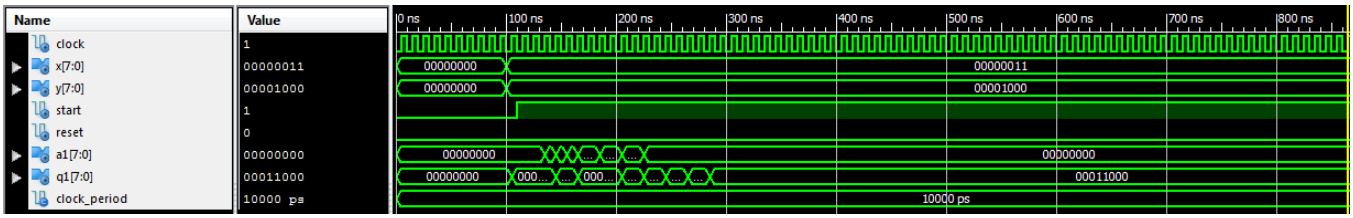
I dati sono stati raccolti sostituendo di volta in volta i valori di X ed Y.

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  ENTITY robertest IS
6  END robertest;
7
8  ARCHITECTURE behavior OF robertest IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11
12     COMPONENT Robertson
13     PORT(
14          clock : IN  std_logic;
15          X : IN  std_logic_vector(7 downto 0);
16          Y : IN  std_logic_vector(7 downto 0);
17          start : IN  std_logic;
18          reset : IN  std_logic;
19          A1 : OUT  std_logic_vector(7 downto 0);
20          Q1 : OUT  std_logic_vector(7 downto 0)
21         );
22     END COMPONENT;
23
24
25    --Inputs
26    signal clock : std_logic := '0';
27    signal X : std_logic_vector(7 downto 0) := (others => '0');
28    signal Y : std_logic_vector(7 downto 0) := (others => '0');
29    signal start : std_logic := '0';
30    signal reset : std_logic := '0';
31
32   --Outputs
33    signal A1 : std_logic_vector(7 downto 0);
34    signal Q1 : std_logic_vector(7 downto 0);
```

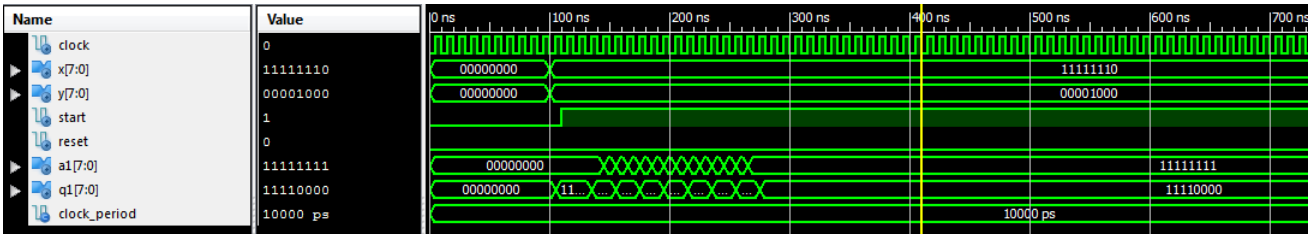
```
35
36    -- Clock period definitions
37    constant clock_period : time := 10 ns;
38
39  BEGIN
40
41    -- Instantiate the Unit Under Test (UUT)
42    uut: Robertson PORT MAP (
43            clock => clock,
44            X => X,
45            Y => Y,
46            start => start,
47            reset => reset,
48            A1 => A1,
49            Q1 => Q1
50          );
51
52    -- Clock process definitions
53    clock_process :process
54    begin
55     clock <= '0';
56     wait for clock_period/2;
57     clock <= '1';
58     wait for clock_period/2;
59    end process;
60
61
62     -- Stimulus process
63     stim_proc: process
64     begin
65
66
67     X <= "00000000";
68     Y <= "00000000";
69         wait for 100 ns;
70
71
72     X <= "00000001";
73     Y <= "00000001";
74
75     wait for 10 ns;
76     start <= '1';
77     --wait for 100ns;
78     --start <= '0';
79
80       wait;
81    end process;
82
83  END;
```

Codice Componente 10.6: Definizione del testbench per Moltiplicatore di Robertson

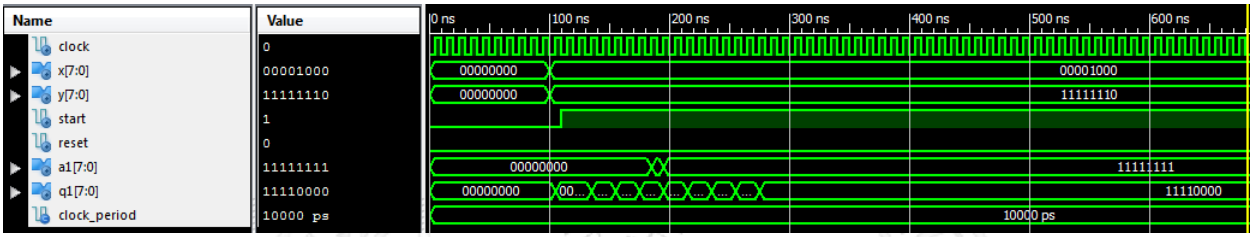
Tutti i risultati dei test corrispondono a quelli attesi.
Test 1



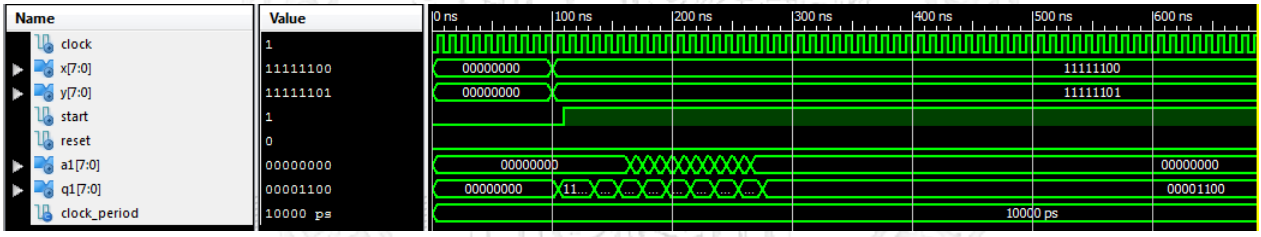
Test 2



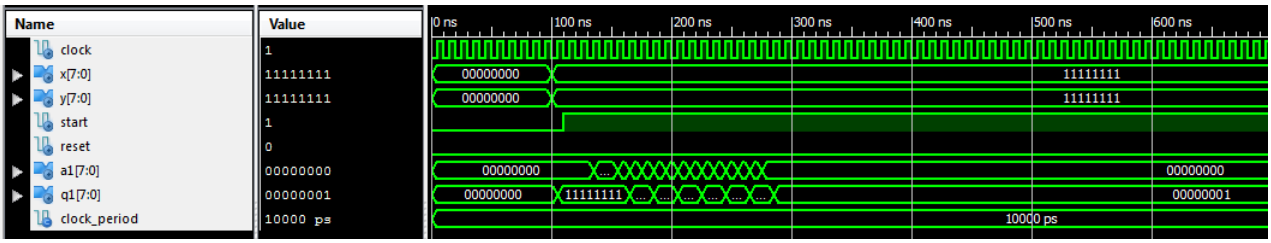
Test 3



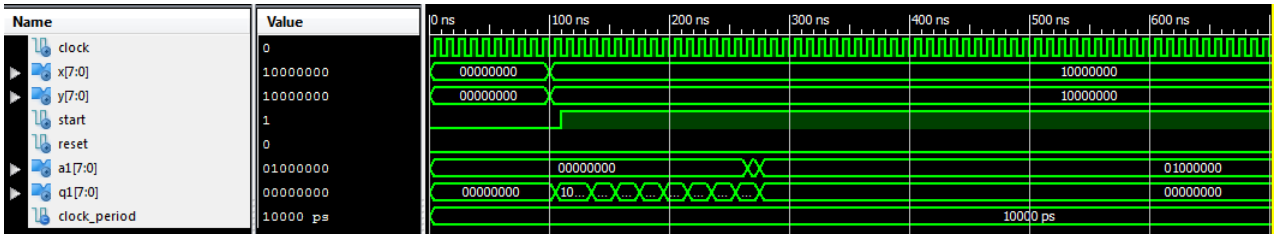
Test 4



Test 5



Test 6



10.4 Sintesi su board FPGA

10.4.1 Sintesi Robertson

Al moltiplicatore di Robertson vengono aggiunti un componente display a 7 segmenti per mostrare operandi e risultato. É presente un process per regolare l'immissione degli input (switch e bottoni) e dare lo start al moltiplicatore per calcolare il risultato. Dopo la pressione del bottone result, al segnale value_to_display, direttamente collegato col display a 7 segmenti, vengono assegnati tramite l'operatore & i due segnali di uscita A e Q, che insieme rappresentano il risultato su 16 bit della moltiplicazione.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6  entity RobertsonFPGA is
7  Port (
8      clock_50Mhz : in STD_LOGIC;
9      X : in STD_LOGIC_VECTOR (7 downto 0);
10     load1,load2,result,reset  : in STD_LOGIC;
11         Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
12         LED_out : out STD_LOGIC_VECTOR (6 downto 0)
13  );
14
15 end RobertsonFPGA;
16
17 architecture Behavioral of RobertsonFPGA is
18
19 signal OP_1,OP_2 : boolean := false;
```

```
20 signal op1,op2: std_logic_vector (7 downto 0) := (others => '0') ;
21
22 signal start: std_logic := '0';
23 constant zero: std_logic_vector (7 downto 0) := (others => '0') ;
24 signal A1,Q1 : std_logic_vector (7 downto 0) := (others => '0') ;
25
26 signal value_to_display : std_logic_vector (15 downto 0);
27
28
29 component Robertson is
30   Port (
31     clock : in  STD_LOGIC;
32     X,Y : in  STD_LOGIC_VECTOR (7 downto 0);
33     start,reset : in STD_LOGIC;
34     A1, Q1: out STD_LOGIC_VECTOR (7 downto 0)
35
36   );
37 end component;
38
39
40 component SEV_SEG_DISP is
41     Port ( clock_50Mhz : in STD_LOGIC;
42          X : in STD_LOGIC_VECTOR (15 downto 0);
43             reset : in STD_LOGIC;
44             Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
45             LED_out : out STD_LOGIC_VECTOR (6 downto 0)
46          );
47
48 end component;
49
50
51 begin
52
53
54 ROB : Robertson port map (
55     clock => clock_50Mhz,
56     X => op1,
57     Y => op2,
58     start => start,
59     reset => reset,
60     A1 => A1,
61     Q1 => Q1
62 );
63
64
65
66 SSD : SEV_SEG_DISP port map(
67     clock_50Mhz => clock_50Mhz,
68     X => value_to_display,
```

```
69      reset => reset,
70      Anode_Activate => Anode_Activate,
71      LED_out => LED_out
72  );
73
74
75  p: process (clock_50Mhz, reset)
76
77  begin
78
79    if (reset= '1') then
80      OP_1 <= false;
81      OP_2 <= false;
82      op1<= (others=>'0');
83      op2<= (others=>'0');
84      start <= '0';
85    elsif (rising_edge(clock_50Mhz)) then
86      if load1 = '1' then
87          start <= '0';
88          op1 <= X;
89          OP_1 <= true;
90          OP_2 <= false;
91
92      elsif load2 = '1' then
93          op2 <= X;
94          OP_2 <= true;
95      end if;
96
97      if (OP_1 = true and OP_2 = true) then
98        if result = '1' then
99          start <= '1';
100         value_to_display <= A1 & Q1;
101       end if;
102     else
103       value_to_display <= zero & X;
104     end if;
105
106   end if;
107
108
109 end process;
110
111 end Behavioral;
```

Codice Componente 10.7: Sintesi Robertson

10.4.2 File UCF

```
1
2  NET "LED_out<6>" LOC = "L18"; # Bank = 1, Pin name = IO_L10P_1, Type = I/O,
       Sch name = CA
3  NET "LED_out<5>" LOC = "F18"; # Bank = 1, Pin name = IO_L19P_1, Type = I/O,
       Sch name = CB
4  NET "LED_out<4>" LOC = "D17"; # Bank = 1, Pin name = IO_L23P_1/HDC, Type =
       DUAL, Sch name = CC
5  NET "LED_out<3>" LOC = "D16"; # Bank = 1, Pin name = IO_L23N_1/LDC0, Type =
       DUAL, Sch name = CD
6  NET "LED_out<2>" LOC = "G14"; # Bank = 1, Pin name = IO_L20P_1, Type = I/O,
       Sch name = CE
7  NET "LED_out<1>" LOC = "J17"; # Bank = 1, Pin name = IO_L13P_1/A6/RHCLK4/
       IRDY1, Type = RHCLK/DUAL, Sch name = CF
8  NET "LED_out<0>" LOC = "H14"; # Bank = 1, Pin name = IO_L17P_1, Type = I/O,
       Sch name = CG
9
10 NET "clock_50Mhz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
       = GCLK, Sch name = GCLK0
11 NET "Anode_Activate<0>" LOC = "F17"; # Bank = 1, Pin name = IO_L19N_1, Type
       = I/O, Sch name = AN0
12 NET "Anode_Activate<1>" LOC = "H17"; # Bank = 1, Pin name = IO_L16N_1/A0,
       Type = DUAL, Sch name = AN1
13 NET "Anode_Activate<2>" LOC = "C18"; # Bank = 1, Pin name = IO_L24P_1/LDC1,
       Type = DUAL, Sch name = AN2
14 NET "Anode_Activate<3>" LOC = "F15"; # Bank = 1, Pin name = IO_L21P_1, Type
       = I/O, Sch name = AN3
15
16
17 NET "X<0>"        LOC = "G18";    # Sch name = SW0
18 NET "X<1>"        LOC = "H18";    # Sch name = SW1
19 NET "X<2>"        LOC = "K18";    # Sch name = SW2
20 NET "X<3>"        LOC = "K17";    # Sch name = SW3
21 NET "X<4>"      LOC = "L14";
22 NET "X<5>"      LOC = "L13";
23 NET "X<6>"      LOC = "N17";
24 NET "X<7>"      LOC = "R17";
25
26
27 NET "load1" LOC = "B18";
28 NET "load2"  LOC = "D18";
29 NET "result"  LOC = "E18";
30 NET "reset"  LOC = "H13";
```

Codice Componente 10.8: Definizione del file UCF

Capitolo 11

Esercizio 11

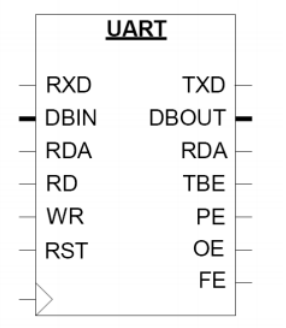
11.1 Traccia

Sfruttando l'implementazione fornita dalla Digilent di un dispositivo UART (RS232RefComp.vhd), progettare ed implementare in VHDL i seguenti componenti:

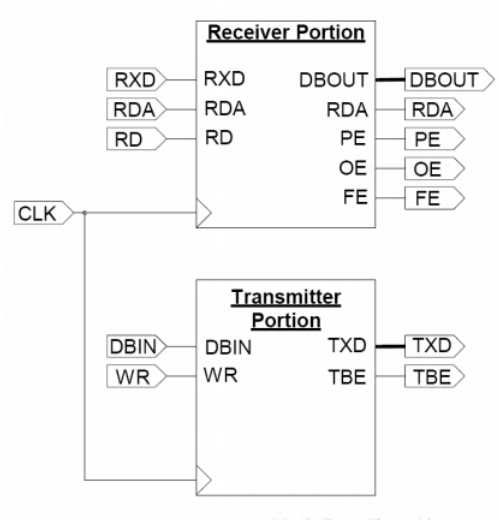
- a) UART_TAPPO: il componente acquisisce una stringa di 8 bit (fornita attraverso gli switch della board di sviluppo) e la serializza tramite la sezione di trasmissione del dispositivo UART; l'output seriale della UART viene re-inviato in ingresso alla sezione di ricezione dello stesso dispositivo (configurazione a tappo), e il dato deserializzato viene visualizzato sui led della board di sviluppo.
- b) 2_UART: il componente acquisisce una stringa di 8 bit (fornita dall'utente tramite gli switch della board di sviluppo), la serializza tramite la sezione di trasmissione di un primo dispositivo UART, la deserializza tramite la sezione di ricezione di un secondo dispositivo UART collegato a valle del primo, e mostra le stringa led della board di sviluppo.
- c) UART_PC (facoltativo): il componente realizza la comunicazione fra la board di sviluppo e un terminale seriale in esecuzione su PC (es. Termite), previa connessione di PC e board tramite dispositivo fisico RS232 (uno degli endpoint di comunicazione è rappresentato dal PC). Il componente deve poter acquisire una stringa di 8 bit che rappresenta un carattere in codifica ASCII (fornita attraverso gli switch della board di sviluppo), ed inviarla tramite il dispositivo UART al terminale in esecuzione sul PC, in cui il carattere viene visualizzato. Allo stesso modo, il componente deve essere in grado di ricevere attraverso lo stesso dispositivo UART (oppure una seconda UART) un carattere trasmesso dal terminale e mostrarlo sui led.

11.2 UART RS232

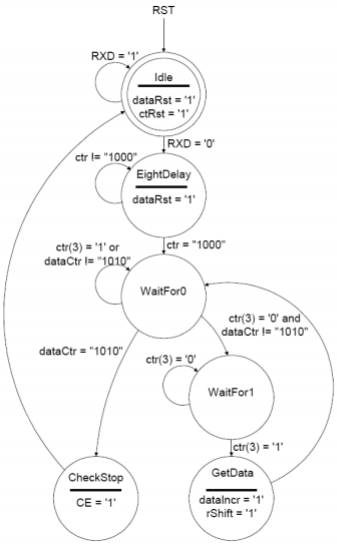
L'UART RS232 è un dispositivo di comunicazione asincrona che permette di serializzare il dato in trasmissione, di parallelizzare il dato in ricezione, di gestire il protocollo asincrono verso l'interlocutore collegato e di controllare e segnalare errori.



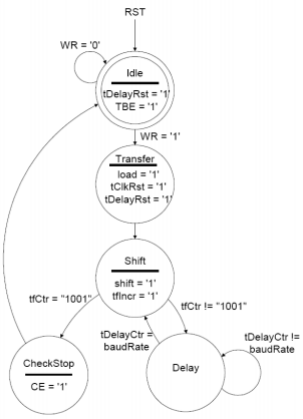
Il dispositivo possiede due componenti principali : uno per ricevere informazioni seriali e uno per trasmetterle. Il ricevitore prende in ingresso sulla porta RXD un byte in forma seriale e lo converte in una informazione parallela, che viene posta in uscita sulla porta DBOUT. Il blocco trasmettitore prende in ingresso l'uscita di DBOUT e converte il byte in seriale, ritrasmettendolo in uscita su TXD.



Il comportamento della UART in ricezione e in tramsissione viene descritto tramite dei diagrammi a stati finiti. Il primo che viene mostrato è quello relativo alla ricezione.



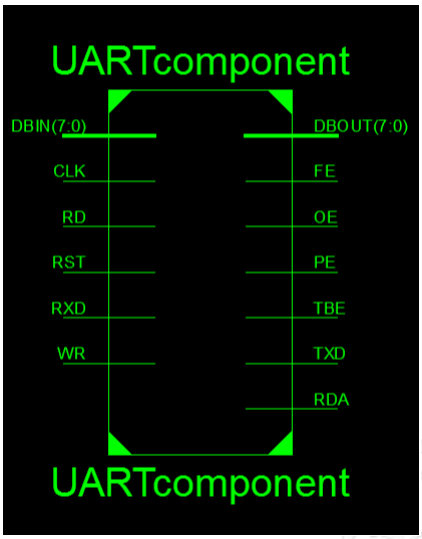
Il ricevitore include un controllore seriale, due contatori usati per la sincronizzazione, uno shift register e un controllo dell'errore sui bit. Lo shift register è usato per memorizzare i dati provenienti dalla porta RXD. Quando un ciclo di lettura è chiuso, lo shift register è pronto per acquisire il byte ricevuto. Inoltre un controllore è necessario per sincronizzare la fase di acquisizione.
Il sistema rimane nello stato di idle fin quando RXD è 1. Quando RXD è 0, si passa allo stato di EightDelay, che viene usato per consentire ai dati di essere letti da RXD nel mezzo della trasmissione. Un contatore ctr conta fino a 8 per acquisire il byte in input. Quando il contatore arriva ad 8 si passa allo stato WaitFor0 e successivamente a WaitFor1. Entrambi questi stati controllano il bit più significativo di ctr. Essi inoltre consentono alla macchina di attendere per il periodo di tempo necessario a leggere il segnale proveniente da RXD nel mezzo della prossima trasmissione. Lo stato successivo, Get Data, usa lo shift register per memorizzare i dati provenienti da RXD e usa un contatore DataCtr per tenere traccia di quanti bit sono stati shiftati. Quando il contatore arriva a 10 (8 bit di dati, 1 bit di parità e 1 bit di stop) si passa allo stato CheckStop, che abilita il controllo sull'errore, e una volta terminato torna allo stato di idle ponendo il sistema in attesa di nuovi dati.
Di seguito si mostra il diagramma a stati finiti per quando riguarda la trasmissione.



Il trasmettitore comprende un controllore per la trasmissione, due contatori per la sincronizzazione

e uno shift register di trasferimento. Esso accetta in ingresso i byte provenienti dalla porta DBIN e li trasmette come dati seriali sulla porta TXD. Similmente a come visto per la precedente macchina a stati, il trasmettitore rimane in uno stato di idle finchè WR è basso. Quando WR è alto, passa allo stato di Transfer che prepara lo shift register al trasferimento. Ponendo Load = 1, lo shift register è caricato con un bit di start, il byte in DBIN, un bit di parità e un bit di stop. I due segnali di reset vengono posti ad 1 per i due contatori di sincronizzazione.
Il prossimo stato è ShiftState, che pone il segnale di shift a 1 per effettuare l'operazione di shift a destra e tfIncr viene anch'esso posto ad 1 per incrementare il contatore dei dati. Quando questo contatore è pari a 9 se non tutti i dati sono stati trasferiti si va nello stato di Delay, da cui si ritorna quando il tDelayCtr è uguale al baudRate.
Una volta entrati nello stato di WaitWrite, la trasmissione è completata. Questo stato è necessario ad assicurarsi che il segnale WR sia stato mantenuto alto durante il processo di trasferimento.

11.2.1 Schematici



11.2.2 Codice

Viene mostrata l'implementazione fornita del RS232RefComp così come descritta.

11.2.2.1 RS232RefComp2

```
1
2  ----------------------------------------------------------------------
3  -- uartcomponent.vhd
4  ----------------------------------------------------------------------
5  -- Author:  Dan Pederson
6  --          Copyright 2004 Digilent, Inc.
7  ----------------------------------------------------------------------
8  -- Description: This file defines a UART which transfers data to and
9  --        from serial and parallel information.  It requires two
10 --        major processes:  receiving and transferring.  The
11 --        receiving portion reads serially transmitted data, and
```

```
12 --        converts it into parallel data, while the transferring
13 --        portion reads parallel data, and transmits it as serial
14 --        data.  There are three error signals provided with this
15 --        UART.  They are frame error, parity error, and overwrite
16 --        error signals.  This UART is configured to use an ODD
17 --        parity bit at a baud rate of 9600.
18 --
19 ----------------------------------------------------------------------
20 -- Revision History:
21 --    07/15/04 (DanP) Created
22 --    05/24/05 (DanP) Updated commenting style
23 --    06/06/05 (DanP) Synchronized state machines to fix timing bug
24 ----------------------------------------------------------------------
25
26 library IEEE;
27 use IEEE.STD_LOGIC_1164.ALL;
28 use IEEE.STD_LOGIC_ARITH.ALL;
29 use IEEE.STD_LOGIC_UNSIGNED.ALL;
30
31 ----------------------------------------------------------------------
32 --
33 --Title:  UARTcomponent entity
34 --
35 --Inputs: 7 : RXD
36 --          CLK
37 --          DBIN
38 --          RDA
39 --          RD
40 --          WR
41 --          RST
42 --
43 --Outputs:  7 : TXD
44 --          DBOUT
45 --          RDA
46 --          TBE
47 --          PE
48 --          FE
49 --          OE
50 --
51 --Description:  This describes the UART component entity.  The inputs are
52 --        the Pegasus 50 MHz clock, a reset button, The RXD from
53 --        the serial cable, an 8-bit data bus from the parallel
54 --        port, and Read Data Available (RDA)and Transfer Buffer
55 --        Empty(TBE) handshaking signals.  The outputs are the TXD
56 --        signal for the serial port, an 8-bit data bus for the
57 --        parallel port, RDA and TBE handshaking signals, and three
58 --        error signals for parity, frame, and overwrite errors.
59 --
60 ----------------------------------------------------------------------
```

```
61 entity UARTcomponent is
62   Generic (
63     --@48MHz
64 --    BAUD_DIVIDE_G : integer := 26;  --115200 baud
65 --    BAUD_RATE_G   : integer := 417
66
67     --@26.6MHz
68     BAUD_DIVIDE_G : integer := 14;  --115200 baud
69     BAUD_RATE_G   : integer := 231
70   );
71   Port (
72     TXD   : out   std_logic   := '1';         -- Transmitted serial data
          output
73     RXD   : in    std_logic;              -- Received serial data input
74     CLK   : in    std_logic;              -- Clock signal
75     DBIN  : in    std_logic_vector (7 downto 0);    -- Input parallel data
          to be transmitted
76     DBOUT   : out   std_logic_vector (7 downto 0);    -- Recevived parallel
          data output
77     RDA   : inout  std_logic;             -- Read Data Available
78     TBE   : out   std_logic   := '1';         -- Transfer Buffer Emty
79     RD    : in    std_logic;              -- Read Strobe
80     WR    : in    std_logic;              -- Write Strobe
81     PE    : out   std_logic;              -- Parity error
82     FE    : out   std_logic;              -- Frame error
83     OE    : out   std_logic;              -- Overwrite error
84     RST   : in    std_logic := '0');      -- Reset signal
85
86 end UARTcomponent;
87
88 architecture Behavioral of UARTcomponent is
89
90 -----------------------------------------------------------------------
91 -- Local Type and Signal Declarations
92 -----------------------------------------------------------------------
93
94 -----------------------------------------------------------------------
95 --Title:  Local Type Declarations
96 --
97 --Description:  There are two state machines used in this entity.  The
98 --        rstate is used to synchronize the receiving portion of
99 --        the UART, and the tstate is used to synchronize the
100 --        sending portion of the UART.
101 --
102 -----------------------------------------------------------------------
103   type rstate is (
104     strIdle,
105     strEightDelay,
106     strGetData,
```

```
107     strWaitFor0,
108     strWaitFor1,
109     strCheckStop
110   );
111 
112   type tstate is (
113     sttIdle,
114     sttTransfer,
115     sttShift,
116     sttDelay,
117     sttWaitWrite
118     );
119 
120 ---------------------------------------------------------------------------
121 --
122 --Title:  Local Signal Declarations
123 --
124 --Description:  The constants and signals used by this entity are
125 --         described below:
126 --
127 --         -baudRate : This is the Baud Rate constant used to
128 --                 synchronize the Pegasus 50 MHz clock with a
129 --                 baud rate of 9600.  To get this number, divide
130 --                 50MHz by 9600.
131 --         -baudDivide :   This is the Baud Rate divider used to safely
132 --                 read data transmitted at a baud rate of 9600.
133 --                 It is simply the above described baudRate
134 --                 constant divided by 16.
135 --
136 --         -rdReg    : this is the receive holding register
137 --         -rdSReg   : this is the receive shift register
138 --         -tfReg    : this is the transfer holding register
139 --         -tfSReg   : this is the transfer shifting register
140 --         -clkDiv   : counter used to get rClk
141 --         -ctr    : used for delay times
142 --         -tfCtr    : used to delay in the transfer process
143 --         -dataCtr  : counts the number of read data bits
144 --         -parError : parity error bit
145 --         -frameError : frame error bit
146 --         -CE     : clock enable bit for the writing latch
147 --         -ctRst      : reset for the ctr
148 --         -load   : load signal used to load the transfer shift
149 --                 register
150 --         -shift    : shift signal used to unload the transfer
151 --                 shift register
152 --         -par    : represents the parity in the transfer
153 --                 holding register
154 --         -tClkRST  : reset for the tfCtr
155 --         -rShift   : shift signal used to load the receive shift
```

```
156 --                register
157 --        -dataRST  : reset for the dataCtr
158 --        -dataIncr : signal to increment the dataCtr
159 --        -tfIncr   : signal to increment the tfCtr
160 --        -tDelayCtr  : counter used to delay the transfer state
161 --                machine.
162 --        -tDelayRst  : reset signal for the tDelayCtr counter.
163 --
164 --        The following signals are used by the two state machines
165 --        for state control:
166 --        -Receive State Machine  : strCur, strNext
167 --        -Transfer State Machine : sttCur, sttNext
168 --
169 ---------------------------------------------------------------------------
170
171 --  @26.7MHz
172 --  constant baudRate : std_logic_vector(12 downto 0) := "1 0100 0101 1000";
173 --  constant baudRate : std_logic_vector(12 downto 0) :=
    conv_std_logic_vector(1406,13); -- 19200
174 --  constant baudRate : std_logic_vector(12 downto 0) :=
    conv_std_logic_vector(703,13);  -- 38400
175 --  constant baudRate : std_logic_vector(12 downto 0) :=
    conv_std_logic_vector(469,13);  -- 57600
176 --   constant baudRate  : std_logic_vector(12 downto 0) :=
    conv_std_logic_vector(417,13);  --115200
177
178 --  @26.7MHz
179 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(1,9);  -- Used for simulation
180 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(88,9); -- Used for  19 200 baud
181 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(44,9); -- Used for  38 400 baud
182 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(29,9); -- Used for  57 600 baud
183 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(26,9); -- Used for 115 200 baud
184
185   constant baudRate : std_logic_vector(12 downto 0)   :=
        conv_std_logic_vector(BAUD_RATE_G,13);  --115200
186   constant baudDivide :   std_logic_vector(8 downto 0)  :=
        conv_std_logic_vector(BAUD_DIVIDE_G-1,9); -- Used for 115 200 baud
187
188   signal rdReg    :   std_logic_vector(7 downto 0)  := "00000000";
189   signal rdSReg   :   std_logic_vector(9 downto 0)  := "1111111111";
190   signal tfReg    :   std_logic_vector(7 downto 0);
191   signal tfSReg     :   std_logic_vector(10 downto 0)   := "11111111111";
192   signal clkDiv   :   std_logic_vector(9 downto 0)    := "0000000000";
193   signal ctr      :   std_logic_vector(3 downto 0)    := "0000";
```

```
194   signal tfCtr    :   std_logic_vector(3 downto 0)    := "0000";
195   signal dataCtr    :   std_logic_vector(3 downto 0)    := "0000";
196   signal parError   :   std_logic;
197   signal frameError :   std_logic;
198   signal CE     : std_logic;
199   signal ctRst    :   std_logic   := '0';
200   signal load     :   std_logic   := '0';
201   signal shift    :   std_logic   := '0';
202   signal par      :   std_logic;
203    signal tClkRST   :   std_logic   := '0';
204   signal rShift   :   std_logic   := '0';
205   signal dataRST    : std_logic   := '0';
206   signal dataIncr   :   std_logic   := '0';
207   signal tfIncr   : std_logic := '0';
208   signal tDelayCtr  : std_logic_vector (12 downto 0);
209   signal tDelayRst  : std_logic := '0';
210
211   signal strCur   :  rstate := strIdle;
212   signal strNext    :  rstate;
213   signal sttCur     :  tstate   := sttIdle;
214   signal sttNext    :  tstate;
215
216 ------------------------------------------------------------------------
217 -- Module Implementation
218 ------------------------------------------------------------------------
219 begin
220 ------------------------------------------------------------------------
221 --
222 --Title:  Initial signal definitions
223 --
224 --Description:  The following lines of code define 4 internal and 1
225 --        external signal.  The most significant bit of the rdSReg
226 --        signifies the frame error bit, so frameError is tied to
227 --        that signal.  The parError is high if there is a parity
228 --        error, so it is set equal to the inverse of rdSReg(8)
229 --        XOR-ed with the data bits.  In this manner, it can
230 --        determine if the parity bit found in rdSReg(8) matches
231 --        the data bits.  The parallel information output is equal
232 --        to rdReg, so DBOUT is set equal to rdReg.  Likewise, the
233 --        input parallel information is equal to DBIN, so tfReg is
234 --        set equal to DBIN.  Because the tfSReg is used to shift
235 --        out transmitted data, the TXD port is set equal to the
236 --        first bit of tfsReg.  Finally, the par signal represents
237 --        the parity of the data, so par is set to the inverse of
238 --        the data bits XOR-ed together.  This UART can be changed
239 --        to use EVEN parity if the "not" is omitted from the par
240 --        definition.
241 --
242 ------------------------------------------------------------------------
```

```
243   frameError <= not rdSReg(9);
244   parError <= not ( rdSReg(8) xor (((rdSReg(0) xor rdSReg(1)) xor
245     (rdSReg(2) xor rdSReg(3))) xor ((rdSReg(4) xor rdSReg(5)) xor
246     (rdSReg(6) xor rdSReg(7)))) );
247   DBOUT <= rdReg;
248   tfReg <= DBIN;
249   TXD <= tfsReg(0);
250   par <=  not ( ((tfReg(0) xor tfReg(1)) xor (tfReg(2) xor tfReg(3))) xor
251     ((tfReg(4) xor tfReg(5)) xor (tfReg(6) xor tfReg(7))) );
252 ------------------------------------------------------------------------
253 --
254 --Title: Clock Divide counter
255 --
256 --Description:  This process defines clkDiv as a signal that increments
257 --        with the clock up until it is either reset by ctRst, or
258 --        equals baudDivide.  This signal is used to define a
259 --        counter called ctr that increments at the rate of the
260 --        divided baud rate.
261 --
262 ------------------------------------------------------------------------
263   process (CLK, clkDiv)
264     begin
265       if (CLK = '1' and CLK'event) then
266         if (clkDiv = baudDivide or ctRst = '1') then
267           clkDiv <= "0000000000";
268         else
269           clkDiv <= clkDiv +1;
270         end if;
271       end if;
272     end process;
273 ------------------------------------------------------------------------
274 --
275 --Title: Transfer delay counter
276 --
277 --Description:  This process defines tDelayCtr as a counter that runs
278 --        until it equals baudRate, or until it is reset by
279 --        tDelayRst.  This counter is used to measure delay times
280 --        when sending data out on the TXD signal.  When the
281 --        counter is equal to baudRate, or is reset, it is set
282 --        equal to 0.
283 --
284 ------------------------------------------------------------------------
285   process (CLK, tDelayCtr)
286     begin
287       if (CLK = '1' and CLK'event) then
288         if (tDelayCtr = baudRate or tDelayRst = '1') then
289           tDelayCtr <= "0000000000000";
290         else
291           tDelayCtr <= tDelayCtr+1;
```

```
292         end if;
293       end if;
294     end process;
295 ---------------------------------------------------------------------------
296 --
297 --Title: ctr set up
298 --
299 --Description:  This process sets up ctr, which uses clkDiv to count
300 --        increase at a rate needed to properly receive data in
301 --        from RXD.  If ctRst is strobed, the counter is reset.  If
302 --        clkDiv is equal to baudDivide, then ctr is incremented
303 --        once.  This signal is used by the receiving state machine
304 --        to measure delay times between RXD reads.
305 --
306 ---------------------------------------------------------------------------
307   process (CLK)
308     begin
309       if CLK = '1' and CLK'Event then
310         if ctRst = '1' then
311           ctr <= "0000";
312         elsif clkDiv = baudDivide then
313           ctr <= ctr + 1;
314         else
315           ctr <= ctr;
316         end if;
317       end if;
318     end process;
319 ---------------------------------------------------------------------------
320 --
321 --Title: transfer counter
322 --
323 --Description:  This process makes tfCtr increment whenever the tfIncr
324 --        signal is strobed high.  If the tClkRst signal is strobed
325 --        high, the tfCtr is reset to "0000."  This counter is used
326 --        to keep track of how many data bits have been
327 --        transmitted.
328 --
329 ---------------------------------------------------------------------------
330   process (CLK, tClkRST)
331     begin
332       if (CLK = '1' and CLK'event) then
333         if tClkRST = '1' then
334           tfCtr <= "0000";
335         elsif tfIncr = '1' then
336           tfCtr <= tfCtr +1;
337         end if;
338       end if;
339     end process;
340 ---------------------------------------------------------------------------
```

```
341 --
342 --Title: Error and RDA flag controller
343 --
344 --Description:  This process controls the error flags FE, OE, and PE, as
345 --        well as the Read Data Available (RDA) flag.  When CE goes
346 --        high, it means that data has been read into the rdSReg.
347 --        This process then analyzes the read data for errors, sets
348 --        rdReg equal to the eight data bits in rdSReg, and flags
349 --        RDA to indicate that new data is present in rdReg.  FE
350 --        and PE are simply equal to the frameError and parError
351 --        signals.  OE is flagged high if RDA is already high when
352 --        CE is strobed.  This means that unread data was still in
353 --        the rdReg when it was written over with the new data.
354 --
355 -------------------------------------------------------------------------
356   process (CLK, RST, RD, CE)
357     begin
358       if RD = '1' or RST = '1' then
359         FE <= '0';
360         OE <= '0';
361         RDA <= '0';
362         PE <= '0';
363       elsif CLK = '1' and CLK'event then
364         if CE = '1' then
365           FE <= frameError;
366           PE <= parError;
367           rdReg(7 downto 0) <= rdSReg (7 downto 0);
368           if RDA = '1' then
369             OE <= '1';
370           else
371             OE <= '0';
372             RDA <= '1';
373           end if;
374         end if;
375       end if;
376     end process;
377 -------------------------------------------------------------------------
378 --
379 --Title: Receiving shift register
380 --
381 --Description:  This process controls the receiving shift register
382 --        (rdSReg).  Whenever rShift is high, implying that data
383 --        needs to be shifted in, rdSReg is shifts in RXD to the
384 --        most significant bit, while shifting its existing data
385 --        right.
386 --
387 -------------------------------------------------------------------------
388   process (CLK, rShift)
389     begin
```

```
390       if CLK = '1' and CLK'Event then
391         if rShift = '1' then
392           rdSReg <= (RXD & rdSReg(9 downto 1));
393         end if;
394       end if;
395     end process;
396 -------------------------------------------------------------------------
397 --
398 --Title: Incoming Data counter
399 --
400 --Description:  This process controls the dataCtr to keep track of
401 --         shifted values into the rdSReg.  The dataCtr signal is
402 --         incremented once every time dataIncr is strobed high.
403 --
404 -------------------------------------------------------------------------
405
406   process (CLK, dataRST)
407     begin
408       if (CLK = '1' and CLK'event) then
409         if dataRST = '1' then
410           dataCtr <= "0000";
411         elsif dataIncr = '1' then
412           dataCtr <= dataCtr +1;
413         end if;
414       end if;
415     end process;
416 -------------------------------------------------------------------------
417 --
418 --Title: Receiving State Machine controller
419 --
420 --Description:  This process takes care of the Receiving state machine
421 --         movement.  It causes the next state to be evaluated on
422 --         each rising edge of CLK.  If the RST signal is strobed,
423 --         the state is changed to the default starting state,
424 --         which is strIdle
425 --
426 -------------------------------------------------------------------------
427   process (CLK, RST)
428     begin
429       if CLK = '1' and CLK'Event then
430         if RST = '1' then -- najj
431           strCur <= strIdle;
432         else
433           strCur <= strNext;
434         end if;
435       end if;
436     end process;
437 -------------------------------------------------------------------------
438 --
```

```
439 --Title: Receiving State Machine
440 --
441 --Description:  This process contains all of the next state logic for the
442 --        Receiving state machine.
443 --
444 -------------------------------------------------------------------------
445   process (strCur, ctr, RXD, dataCtr)
446     begin
447       case strCur is
448 -------------------------------------------------------------------------
449 --
450 --Title: strIdle state
451 --
452 --Description:  This state is the idle and startup default stage for the
453 --        Receiving state machine.  The machine stays in this state
454 --        until the RXD signal goes low.  When this occurs, the
455 --        ctRst signal is strobed to reset ctr for the next state,
456 --        which is strEightDelay.
457 --
458 -------------------------------------------------------------------------
459         when strIdle =>
460           dataIncr <= '0';
461           rShift <= '0';
462           dataRst <= '1';
463           CE <= '0';
464           ctRst <= '1';
465 
466           if RXD = '0' then
467             strNext <= strEightDelay;
468           else
469             strNext <= strIdle;
470           end if;
471 -------------------------------------------------------------------------
472 --
473 --Title: strEightDelay state
474 --
475 --Description:  This state simply delays the state machine for eight clock
476 --        cycles.  This is needed so that the incoming RXD data
477 --        signal is read in the middle of each data emission.  This
478 --        ensures an accurate RXD signal reading.  ctr counts from
479 --        0 to 8 to keep track of rClk cycles.  When it equals 8
480 --        (1000) the next state, strWaitFor0, is loaded.  During
481 --        this state, the dataRst signal is strobed high to reset
482 --        the shift-in data counter (dataCtr).
483 --
484 -------------------------------------------------------------------------
485         when strEightDelay =>
486           dataIncr <= '0';
487           rShift <= '0';
```

```
488           dataRst <= '1';
489           CE <= '0';
490           ctRst <= '0';
491 
492           if ctr(3 downto 0) = "1000" then
493             strNext <= strWaitFor0;
494           else
495             strNext <= strEightDelay;
496           end if;
497 ---------------------------------------------------------------------------
498 --
499 --Title: strGetData state
500 --
501 --Description:  In this state, the dataIncr and rShift signals are
502 --        strobed high for one clock cycle.  By doing this, the
503 --        rdSReg shift register shifts in RXD once, while the
504 --        dataCtr is incremented by one.  This state simply
505 --        captures the incoming data on RXD into the rdSReg shift
506 --        register.  The next state loaded is strWaitFor0, which
507 --        starts the two delay states needed between data shifts.
508 --
509 ---------------------------------------------------------------------------
510         when strGetData =>
511           CE <= '0';
512           dataRst <= '0';
513           ctRst <= '0';
514           dataIncr <= '1';
515           rShift <= '1';
516 
517           strNext <= strWaitFor0;
518 ---------------------------------------------------------------------------
519 --
520 --Title: strWaitFor0 state
521 --
522 --Description:  This state is a delay state, which delays the receive
523 --        state machine if not all of the incoming serial data has
524 --        not been shifted in yet.  If dataCtr does not equal 10
525 --        (1010), the state is stayed in until the fourth bit of
526 --        ctr is equal to 1.  When this happens, half of the delay
527 --        has been achieved, and the second delay state is loaded,
528 --        which is strWaitFor1.  If dataCtr does equal 10 (1010),
529 --        all of the needed data has been acquired, so the
530 --        strCheckStop state is loaded to check for errors and
531 --        reset the receive state machine.
532 --
533 ---------------------------------------------------------------------------
534         when strWaitFor0 =>
535           CE <= '0';
536           dataRst <= '0';
```

```
537           ctRst <= '0';
538           dataIncr <= '0';
539           rShift <= '0';
540 
541           if dataCtr = "1010" then
542             strNext <= strCheckStop;
543           elsif ctr(3) = '0' then
544             strNext <= strWaitFor1;
545           else
546             strNext <= strWaitFor0;
547           end if;
548 ---------------------------------------------------------------------
549 --
550 --Title: strEightDelay state
551 --
552 --Description:  This state is much like strWaitFor0, except it waits for
553 --        the fourth bit of ctr to equal 1.  Once this occurs, the
554 --        strGetData state is loaded in order to shift in the next
555 --        data bit from RXD.  Because strWaitFor0 is the only state
556 --        that calls this state, no other signals need to be
557 --        checked.
558 --
559 ---------------------------------------------------------------------
560         when strWaitFor1 =>
561           CE <= '0';
562           dataRst <= '0';
563           ctRst <= '0';
564           dataIncr <= '0';
565           rShift <= '0';
566 
567           if ctr(3) = '0' then
568             strNext <= strWaitFor1;
569           else
570             strNext <= strGetData;
571           end if;
572 ---------------------------------------------------------------------
573 --
574 --Title: strCheckStop state
575 --
576 --Description:  This state allows the newly acquired data to be checked
577 --        for errors.  The CE flag is strobed to start the
578 --        previously defined error checking process.  This state is
579 --        passed straight through to the strIdle state.
580 --
581 ---------------------------------------------------------------------
582         when strCheckStop =>
583           dataIncr <= '0';
584           rShift <= '0';
585           dataRst <= '0';
```

```
586           ctRst <= '0';
587           CE <= '1';
588           strNext <= strIdle;
589       end case;
590     end process;
591 ------------------------------------------------------------------------
592 --
593 --Title: Transfer shift register controller
594 --
595 --Description:  This process uses the load, shift, and clk signals to
596 --        control the transfer shift register (tfSReg).  Once load
597 --        is equal to '1', the tfSReg gets a '1', the parity bit,
598 --        the data bits found in tfReg, and a '0'.  Under this
599 --        format, the shift register can be used to shift out the
600 --        appropriate signal to serially transfer the data.  The
601 --        data is shifted out of the tfSReg whenever shift = '1'.
602 --
603 ------------------------------------------------------------------------
604   process (load, shift, CLK, tfSReg)
605     begin
606       if CLK = '1' and CLK'Event then
607         if load = '1' then
608           tfSReg (10 downto 0) <= ('1' & par & tfReg(7 downto 0) &'0');
609         elsif shift = '1' then
610           tfSReg (10 downto 0) <= ('1' & tfSReg(10 downto 1));
611         end if;
612       end if;
613     end process;
614 ------------------------------------------------------------------------
615 --
616 --Title: Transfer State Machine controller
617 --
618 --Description:  This process takes care of the Transfer state machine
619 --        movement.  It causes the next state to be evaluated on
620 --        each rising edge of CLK.  If the RST signal is strobed,
621 --        the state is changed to the default starting state, which
622 --        is sttIdle.
623 --
624 ------------------------------------------------------------------------
625   process (CLK, RST)
626     begin
627       if (CLK = '1' and CLK'Event) then
628         if RST = '1' then
629           sttCur <= sttIdle;
630         else
631           sttCur <= sttNext;
632         end if;
633       end if;
634     end process;
```

```
635 ------------------------------------------------------------------------
636 --
637 --Title: Transfer State Machine
638 --
639 --Description:  This process controls the next state logic in the
640 --        transfer state machine.  The transfer state machine
641 --        controls the shift and load signals that are used to load
642 --        and transmit the parallel data in a serial form.  It also
643 --        controls the Transmit Buffer Empty (TBE) signal that
644 --        indicates if the transmit buffer (tfSReg) is in use or
645 --        not.
646 --
647 ------------------------------------------------------------------------
648   process (sttCur, tfCtr, WR, tDelayCtr)
649     begin
650       case sttCur is
651 ------------------------------------------------------------------------
652 --
653 --Title: sttIdle state
654 --
655 --Description:  This state is the idle and startup default stage for the
656 --        transfer state machine.  The state is stayed in until
657 --        the WR signal goes high.  Once it goes high, the
658 --        sttTransfer state is loaded.  The load and shift signals
659 --        are held low in the sttIdle state, while the TBE signal
660 --        is held high to indicate that the transmit buffer is not
661 --        currently in use.  Once the idle state is left, the TBE
662 --        signal is held low to indicate that the transfer state
663 --        machine is using the transmit buffer.
664 --
665 ------------------------------------------------------------------------
666         when sttIdle =>
667           TBE <= '1';
668           tClkRST <= '0';
669           tfIncr <= '0';
670           shift <= '0';
671           load <= '0';
672           tDelayRst <= '1';
673 
674           if WR = '0' then
675             sttNext <= sttIdle;
676           else
677             sttNext <= sttTransfer;
678           end if;
679 ------------------------------------------------------------------------
680 --
681 --Title: sttTransfer state
682 --
683 --Description:  This state sets the load, tClkRST, and tDelayRst signals
```

```
684 --        high, while setting the TBE signal low.  The load signal
685 --        is set high to load the transfer shift register with the
686 --        appropriate data, while the tClkRST and tDelayRst signals
687 --        are strobed to reset the tfCtr and tDelayCtr.  The next
688 --        state loaded is the sttDelay state.
689 --
690 -------------------------------------------------------------------------
691         when sttTransfer =>
692           TBE <= '0';
693           shift <= '0';
694           load <= '1';
695           tClkRST <= '1';
696           tfIncr <= '0';
697           tDelayRst <= '1';
698 
699           sttNext <= sttDelay;
700 -------------------------------------------------------------------------
701 --
702 --Title: sttShift state
703 --
704 --Description:  This state strobes the shift and tfIncr signals high, and
705 --        checks the tfCtr to see if enough data has been
706 --        transmitted.  By strobing the shift and tfIncr signals
707 --        high, the tfSReg is shifted, and the tfCtr is incremented
708 --        once.  If tfCtr does not equal 9 (1001), then not all of
709 --        the bits have been transmitted, so the next state loaded
710 --        is the sttDelay state.  If tfCtr does equal 9, the final
711 --        state, sttWaitWrite, is loaded.
712 --
713 -------------------------------------------------------------------------
714         when sttShift =>
715           TBE <= '0';
716           shift <= '1';
717           load <= '0';
718           tfIncr <= '1';
719           tClkRST <= '0';
720           tDelayRst <= '0';
721 
722           if tfCtr = "1010" then
723             sttNext <= sttWaitWrite;
724           else
725             sttNext <= sttDelay;
726           end if;
727 -------------------------------------------------------------------------
728 --
729 --Title: sttDelay state
730 --
731 --Description:  This state is responsible for delaying the transfer state
732 --        machine between transmissions.  All signals are held low
```

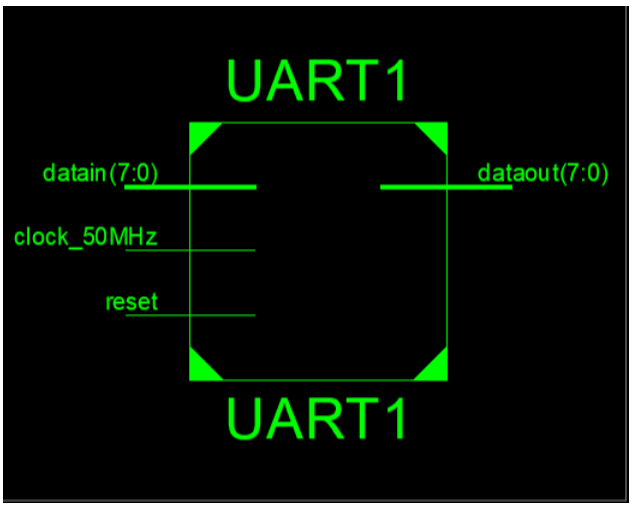
```
733 --        while the tDelayCtr is tested.  Once tDelayCtr is equal
734 --        to baudRate, the sttShift state is loaded.
735 --
736 ---------------------------------------------------------------------------
737         when sttDelay =>
738           TBE <= '0';
739           shift <= '0';
740           load <= '0';
741           tClkRst <= '0';
742           tfIncr <= '0';
743           tDelayRst <= '0';
744 
745           if tDelayCtr = baudRate then
746             sttNext <= sttShift;
747           else
748             sttNext <= sttDelay;
749           end if;
750 ---------------------------------------------------------------------------
751 --
752 --Title: sttWaitWrite state
753 --
754 --Description:  This state checks to make sure that the initial WR signal
755 --        that triggered the transfer state machine has been
756 --        brought back low.  Without this state, a write signal
757 --        that is held high for a long time will result in multiple
758 --        transmissions.  Once the WR signal is low, the sttIdle
759 --        state is loaded to reset the transfer state machine.
760 --
761 ---------------------------------------------------------------------------
762         when sttWaitWrite =>
763           TBE <= '0';
764           shift <= '0';
765           load <= '0';
766           tClkRst <= '0';
767           tfIncr <= '0';
768           tDelayRst <= '0';
769 
770           if WR = '1' then
771             sttNext <= sttWaitWrite;
772           else
773             sttNext <= sttIdle;
774           end if;
775       end case;
776     end process;
777 end Behavioral;
```

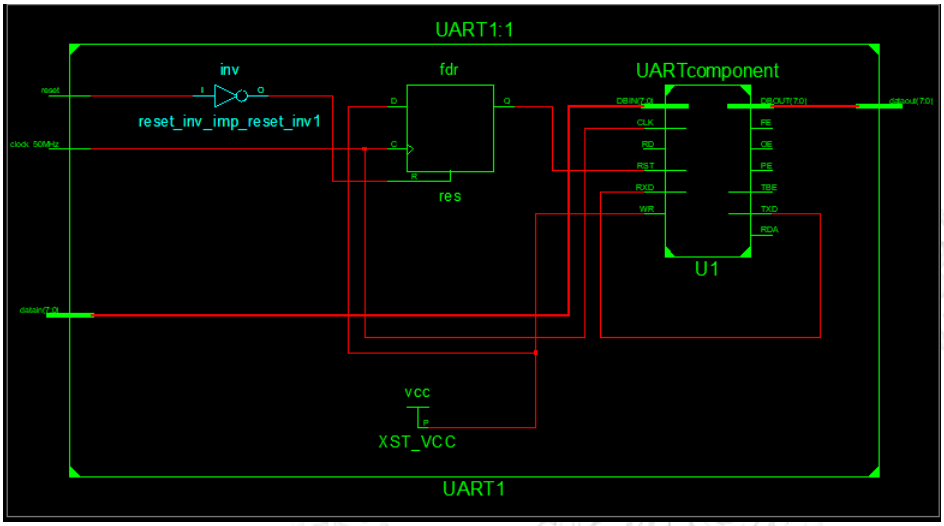
Codice Componente 11.1: RS232RefComp2.vhd

11.3 Soluzione a) Uart Tappo

In ingresso al sistema abbiamo il clock, il byte che deve essere acquisito attraverso gli switch e il reset; l'uscita coincide invece con l'uscita in parallelo del componente UART.

11.3.1 Schematici





11.3.2 Codice

L'uscita TXD del componente UART viene collegata direttamente all'ingresso RXD in retroazione, in questo modo dati ricevuti saranno pari a quelli trasmessi. Il byte acquisito in parallelo attraverso gli switch viene posto in ingresso a DBIN, l'uscita in parallelo DBOUT è collegata all'uscita dataout del sistema e viene visualizzata attraverso l'accensione dei led della scheda.

11.3.2.1 UART Tappo

```
1
2 library IEEE;
```

```
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity UART1 is
6   Port (
7        clock_50MHz : in STD_LOGIC;
8        datain : in STD_LOGIC_VECTOR (7 downto 0);
9        reset : in std_logic;
10       dataout : out STD_LOGIC_VECTOR (7 downto 0):= (others=>'0')
11
12 );
13
14 end UART1;
15
16 architecture Behavioral of UART1 is
17
18 Component UARTcomponent is
19
20   Port (
21     TXD    : out    std_logic    := '1';          -- Transmitted serial data
          output
22     RXD    : in     std_logic;                -- Received serial data input
23     CLK    : in     std_logic;                -- Clock signal
24     DBIN   : in     std_logic_vector (7 downto 0);    -- Input parallel data
          to be transmitted
25     DBOUT    : out    std_logic_vector (7 downto 0);    -- Recevived parallel
          data output
26     RDA    : inout  std_logic;                -- Read Data Available
27     TBE    : out    std_logic    := '1';          -- Transfer Buffer Emty
28     RD     : in     std_logic;                -- Read Strobe
29     WR     : in     std_logic;                -- Write Strobe
30     PE     : out    std_logic;                -- Parity error
31     FE     : out    std_logic;                -- Frame error
32     OE     : out    std_logic;                -- Overwrite error
33     RST    : in     std_logic := '0'
34     );
35 end Component;
36
37 signal din,dout,rd,td :std_logic:= '0';
38
39 signal t1,t_rda,t_tbe,t_pe,t_fe,t_oe: std_logic:= '0';
40
41 begin
42
43
44 U1: UARTcomponent PORT MAP(
45   TXD => t1,
46   RXD => t1,
47   CLK => clock_50MHz,
48   DBIN => datain,
```

```
49    DBOUT => dataout,
50    RDA => t_rda,
51    TBE => t_tbe,
52    RD => '0',
53    WR => '1',
54    PE => t_pe,
55    FE => t_fe,
56    OE => t_oe,
57    RST =>reset
58  );
59
60
61  end Behavioral;
```

Codice Componente 11.2: Definizione del componente UART Tappo

11.4 Sintesi su board FPGA

I segnali in ingresso e in uscita sono mappati sulla board Nexys2 tramite switch e led. La UART si comporta da tappo e il segnale inserito tramite switch viene visualizzato correttamente sui led.

```
1
2
3   ##Leds
4
5   NET "dataout<0>"  LOC = "J14"; # Bank = 1, Pin name = IO_L14N_1/A3/RHCLK7,
        Type = RHCLK/DUAL, Sch name = JD10/LD0
6   NET "dataout<1>"  LOC = "J15"; # Bank = 1, Pin name = IO_L14P_1/A4/RHCLK6,
        Type = RHCLK/DUAL, Sch name = JD9/LD1
7   NET "dataout<2>"  LOC = "K15"; # Bank = 1, Pin name = IO_L12P_1/A8/RHCLK2,
        Type = RHCLK/DUAL, Sch name = JD8/LD2
8   NET "dataout<3>"  LOC = "K14"; # Bank = 1, Pin name = IO_L12N_1/A7/RHCLK3/
        TRDY1, Type = RHCLK/DUAL, Sch name = JD7/LD3
9   NET "dataout<4>" LOC = "E16"; # Bank = 1, Pin name = N.C., Type = N.C., Sch
        name = LD4? other than s3e500
10  NET "dataout<5>" LOC = "P16"; # Bank = 1, Pin name = N.C., Type = N.C., Sch
        name = LD5? other than s3e500
11  NET "dataout<6>" LOC = "E4";  # Bank = 3, Pin name = N.C., Type = N.C., Sch
        name = LD6? other than s3e500
12  NET "dataout<7>" LOC = "P4";  # Bank = 3, Pin name = N.C., Type = N.C., Sch
        name = LD7? other than s3e500
13
14  NET "clock_50MHz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
        = GCLK, Sch name = GCLK0
15
16
17  ##Switch
18  NET "datain<0>"       LOC = "G18";    # Sch name = SW0
```

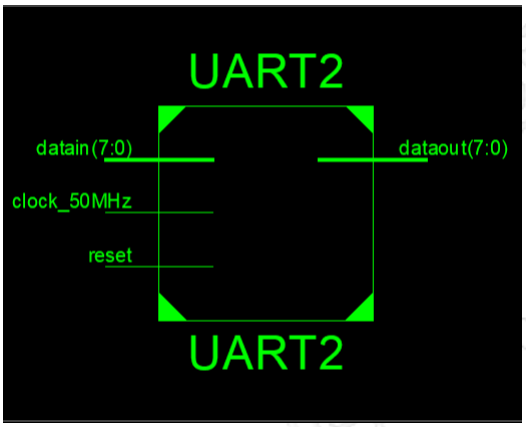
```
19 NET "datain<1>"        LOC = "H18";    # Sch name = SW1
20 NET "datain<2>"        LOC = "K18";    # Sch name = SW2
21 NET "datain<3>"        LOC = "K17";    # Sch name = SW3
22 NET "datain<4>"        LOC = "L14";    # Sch name = SW4
23 NET "datain<5>"        LOC = "L13";    # Sch name = SW5
24 NET "datain<6>"        LOC = "N17";    # Sch name = SW6
25 NET "datain<7>"        LOC = "R17";    # Sch name = SW7
26
27 ## Buttons
28 #
29 NET "reset" LOC = "B18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name =
       BTN0
```

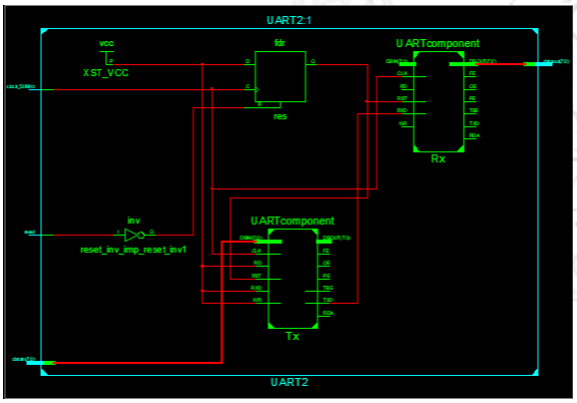
Codice Componente 11.3: Definizione del file UCF

11.5 Soluzione b) 2 UART

La stringa datain di 8 bit viene serializzata tramite il primo dispositivo UART: il segnale viene posto in ingresso a DBIN, viene serializzato dal blocco trasmettitore della prima UART e l'uscita TXD viene posta in ingresso al blocco ricevitore della seconda UART. Il segnale parallelizzato DBOUT ottenuto dalla seconda UART viene collegato all'uscita dataout.

11.5.1 Schematici





11.5.2 Codice

L'entità UART2 contiene due componenti UART dello stesso tipo, RS232. Per consentire il trasferimento dei dati, nel blocco trasmetttore della prima UART vengono posti in ingresso WR e RD entrambi pari ad '1', mentre nella seconda UART (in ricezione) entrambi pari a 0.

11.5.2.1 2UART

```
1
2
3  entity UART2 is
4   Port (
5        reset : in STD_LOGIC;
6        clock_50MHz : in STD_LOGIC;
7        datain : in STD_LOGIC_VECTOR (7 downto 0);
8        dataout : out STD_LOGIC_VECTOR (7 downto 0)
9        );
10
11 end UART2;
12
13 architecture Behavioral of UART2 is
14
15
16 Component UARTcomponent is
17
18   Port (
19      TXD    : out    std_logic    := '1';          -- Transmitted serial data
          output
20      RXD    : in     std_logic;                    -- Received serial data input
21      CLK    : in     std_logic;                    -- Clock signal
22      DBIN   : in     std_logic_vector (7 downto 0);    -- Input parallel data
          to be transmitted
23      DBOUT    : out    std_logic_vector (7 downto 0);    -- Recevived parallel
          data output
24      RDA    : inout  std_logic;                    -- Read Data Available
25      TBE    : out    std_logic    := '1';          -- Transfer Buffer Emty
26      RD     : in     std_logic;                    -- Read Strobe
27      WR     : in     std_logic;                    -- Write Strobe
28      PE     : out    std_logic;                    -- Parity error
29      FE     : out    std_logic;                    -- Frame error
30      OE     : out    std_logic;                    -- Overwrite error
31      RST    : in     std_logic := '0'
32      );
33 end Component;
34
35 signal t_DBout, t_DBin : std_logic_vector(7 downto 0);
36 signal t,t_rda1,t_tbe1,t_rda2,t_tbe2,t_pe1,t_fe1,t_oe1,t_pe2,t_fe2,t_oe2,
      t_WR2,t_RD2,t_txd2: std_logic:= '0';
37
```

```
38
39 begin
40
41
42 Tx: UARTcomponent PORT MAP(
43   TXD => t,
44   RXD => '1',
45   CLK => clock_50MHz,
46   DBIN => datain,
47   DBOUT => t_DBout,
48   RDA => t_rda1,
49   TBE => t_tbe1,
50   RD => '1',
51   WR => '1',
52   PE => t_pe1,
53   FE => t_fe1,
54   OE => t_oe1,
55   RST => reset
56 );
57
58 Rx: UARTcomponent PORT MAP(
59   TXD => t_txd2,
60   RXD => t,
61   CLK => clock_50MHz,
62   DBIN => t_DBin,
63   DBOUT => dataout,
64   RDA => t_rda2,
65   TBE => t_tbe2,
66   RD => '0',
67   WR => '0',
68   PE => t_pe2,
69   FE => t_fe2,
70   OE => t_oe2,
71   RST => reset
72 );
73
74
75 end Behavioral;
```

Codice Componente 11.4: Definizione del componente 2 UART

11.6 Sintesi su board FPGA

Per sintetizzare le UART su dispositivo NEXYS2 è sufficiente mappare i segnali su led e switch. Il segnale di ingresso DBIN viene inserito tramite gli switch ed è possibile visualizzare l'uscita corretta tramite i led (su cui sono mappati i bit del segnale dataout). Il segnale di reset viene invece collegato ad uno dei 4 bottoni disponibili.

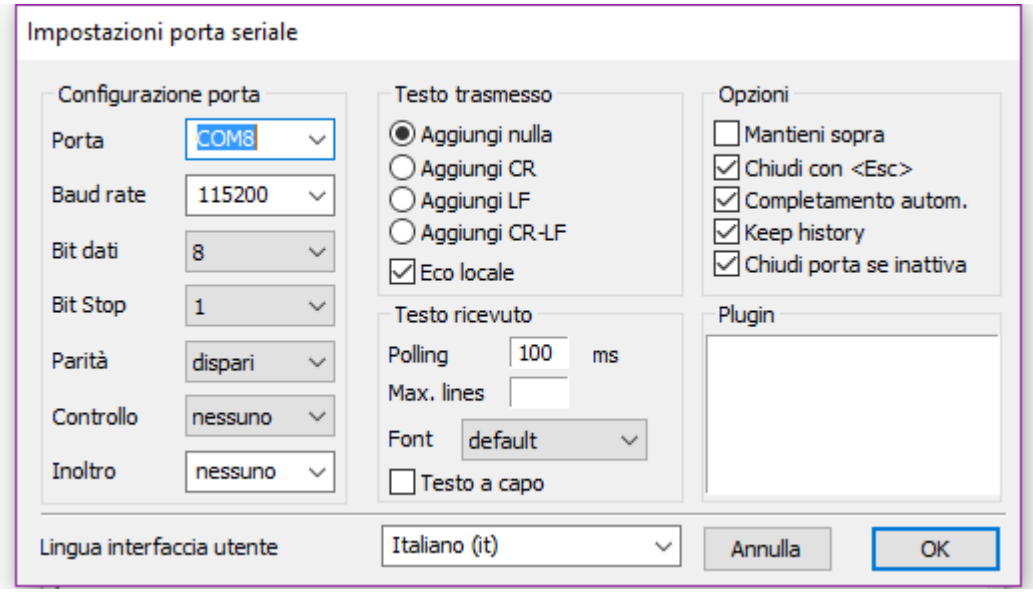
```
1
2  ##Leds
3
4  NET "dataout<0>"  LOC = "J14"; # Bank = 1, Pin name = IO_L14N_1/A3/RHCLK7,
       Type = RHCLK/DUAL, Sch name = JD10/LD0
5  NET "dataout<1>"  LOC = "J15"; # Bank = 1, Pin name = IO_L14P_1/A4/RHCLK6,
       Type = RHCLK/DUAL, Sch name = JD9/LD1
6  NET "dataout<2>"  LOC = "K15"; # Bank = 1, Pin name = IO_L12P_1/A8/RHCLK2,
       Type = RHCLK/DUAL, Sch name = JD8/LD2
7  NET "dataout<3>"  LOC = "K14"; # Bank = 1, Pin name = IO_L12N_1/A7/RHCLK3/
       TRDY1, Type = RHCLK/DUAL, Sch name = JD7/LD3
8  NET "dataout<4>" LOC = "E16"; # Bank = 1, Pin name = N.C., Type = N.C., Sch
       name = LD4? other than s3e500
9  NET "dataout<5>" LOC = "P16"; # Bank = 1, Pin name = N.C., Type = N.C., Sch
       name = LD5? other than s3e500
10 NET "dataout<6>" LOC = "E4";  # Bank = 3, Pin name = N.C., Type = N.C., Sch
       name = LD6? other than s3e500
11 NET "dataout<7>" LOC = "P4";  # Bank = 3, Pin name = N.C., Type = N.C., Sch
       name = LD7? other than s3e500
12
13 NET "clock_50MHz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
       = GCLK, Sch name = GCLK0
14
15
16 ##Switch
17 NET "datain<0>"       LOC = "G18";   # Sch name = SW0
18 NET "datain<1>"       LOC = "H18";   # Sch name = SW1
19 NET "datain<2>"       LOC = "K18";   # Sch name = SW2
20 NET "datain<3>"       LOC = "K17";   # Sch name = SW3
21 NET "datain<4>"       LOC = "L14";   # Sch name = SW4
22 NET "datain<5>"       LOC = "L13";   # Sch name = SW5
23 NET "datain<6>"       LOC = "N17";   # Sch name = SW6
24 NET "datain<7>"       LOC = "R17";   # Sch name = SW7
25
26
27 NET "reset" LOC = "B18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name =
       BTN0
```

Codice Componente 11.5: Definizione del file UCF

11.7 Soluzione c) UART_PC (facoltativo)

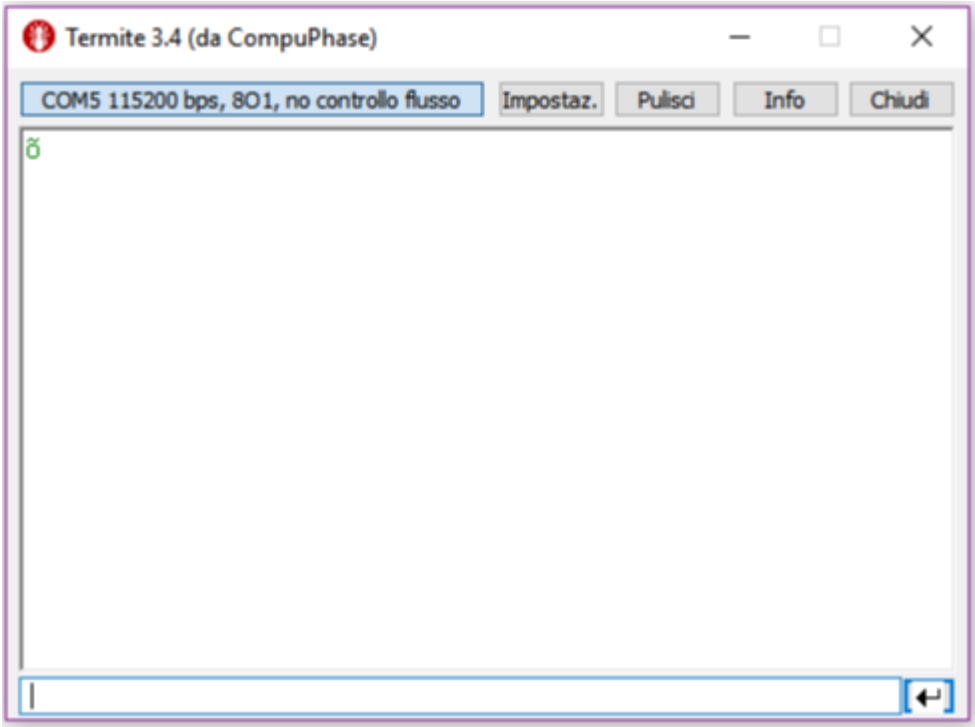
La comunicazione tra PC e UART presente sullo schedino è stata gestita collegando un cavo seriale USB a RS232 alla scheda Nexys2. Il programma utilizzato per osservare i valori codificati in ASCII (e inseriti tramite switch) è Termite, un tool provvisto di terminale in grado di mostrare valori in ingresso e uscita. In configurazione porta si pone la porta COM8 (a cui è stato collegato fisicamente il cavo) e il baud rate a 115200, lo stesso definito per la scheda Nexys2. In questo caso

abbiamo 8 bit di dati da trasferire e 1 bit di stop, parità dispari.

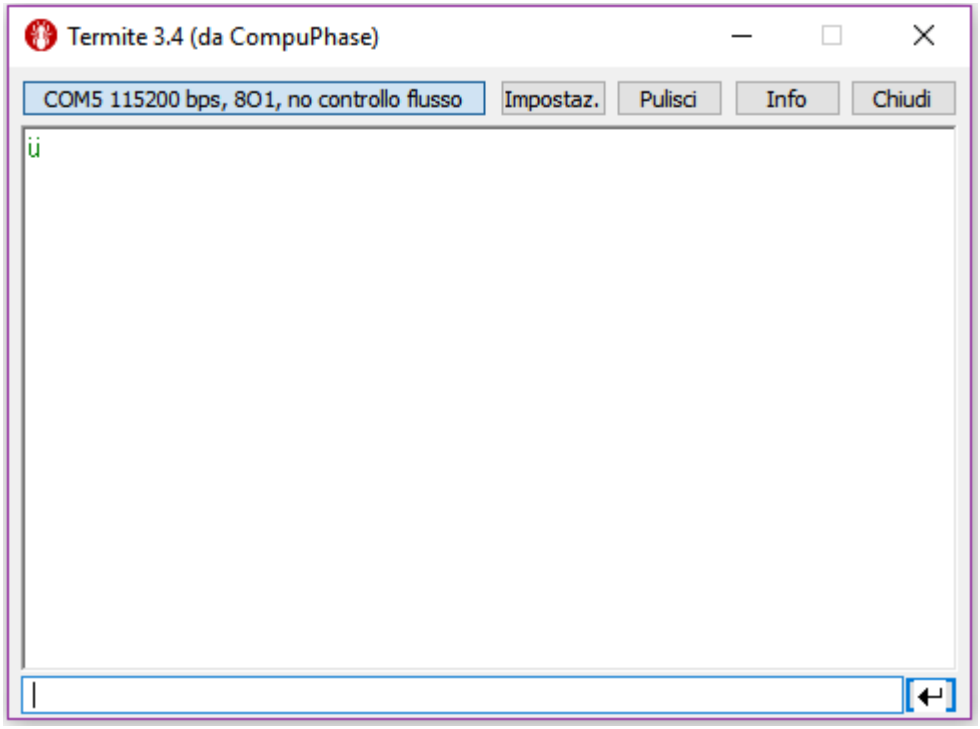


Come esempio si mostrano alcuni dati casuali immessi tramite Nexys2, 11110101 e 11111100 che vengono ricevuti e convertiti in ASCII.

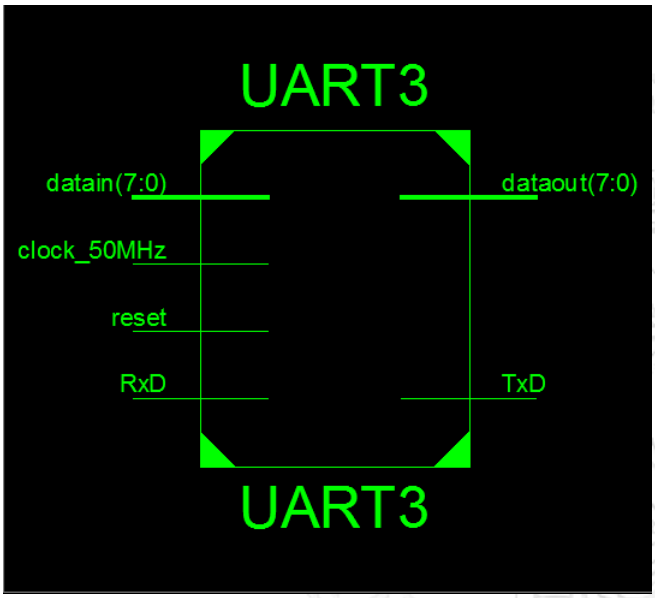
CASO 11110101

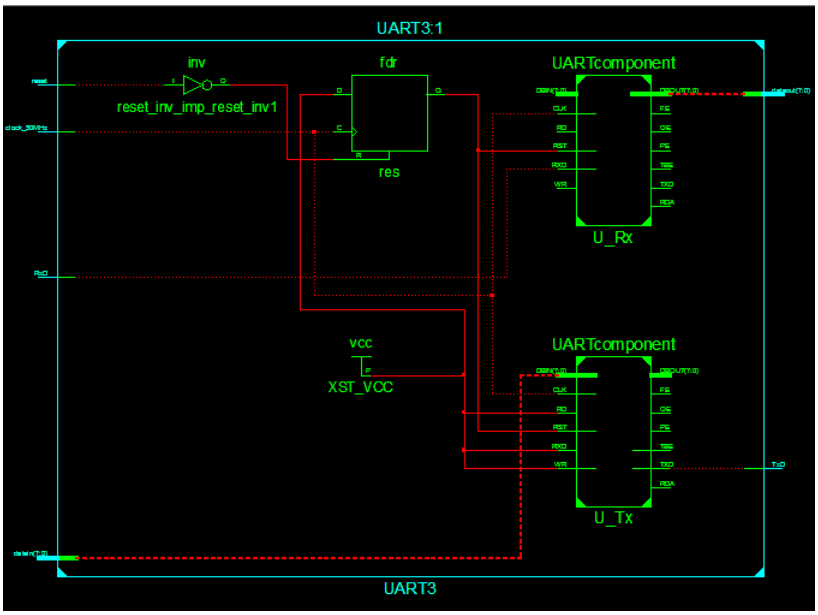


CASO 11111100



11.7.1 Schematici





11.7.2 Codice

La soluzione viene implementata usando una logica di tipo strutturale. Il componente U_Tx trasmette il dato al pc e viene letto da Termite. Il componente U_Rx riceve i dati sulla scheda che vengono visualizzati tramite i led.

11.7.2.1 UART_PC

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity UART3 is
6  port(
7      clock_50MHz: in std_logic;
8      reset: in std_logic;
9      TxD: out std_logic;
10     RxD: in std_logic;
11     datain: in std_logic_vector(7 downto 0);
12     dataout: out std_logic_vector(7 downto 0)
13
14 );
15 end UART3;
16
17 architecture Behavioral of UART3 is
18
19 Component UARTcomponent is
20 port(
21
```

```
22      TXD    : out    std_logic    := '1';          -- Transmitted serial data
           output
23      RXD    : in     std_logic;               -- Received serial data input
24      CLK    : in     std_logic;               -- Clock signal
25      DBIN   : in     std_logic_vector (7 downto 0);    -- Input parallel data
           to be transmitted
26      DBOUT    : out    std_logic_vector (7 downto 0);    -- Recevived parallel
           data output
27      RDA    : inout  std_logic;               -- Read Data Available
28      TBE    : out    std_logic    := '1';          -- Transfer Buffer Emty
29      RD     : in     std_logic;               -- Read Strobe
30      WR     : in     std_logic;               -- Write Strobe
31      PE     : out    std_logic;               -- Parity error
32      FE     : out    std_logic;               -- Frame error
33      OE     : out    std_logic;               -- Overwrite error
34      RST    : in     std_logic := '0'
35
36  );
37  end Component;
38
39  signal t_DBout, t_DBin : std_logic_vector(7 downto 0);
40  signal t,t_rda1,t_tbe1,t_rda2,t_tbe2,t_pe1,t_fe1,t_oe1,t_pe2,t_fe2,t_oe2,
        t_WR2,t_RD2,t_txd2: std_logic:= '0';
41
42
43  begin
44
45
46  U_Tx: UARTcomponent PORT MAP(
47    TXD => TxD,
48    RXD => '1',
49    CLK => clock_50MHz,
50    DBIN => datain,
51    DBOUT => t_DBout,
52    RDA => t_rda1,
53    TBE => t_tbe1,
54    RD => '1',
55    WR => '1',
56    PE => t_pe1,
57    FE => t_fe1,
58    OE => t_oe1,
59    RST => reset
60  );
61
62
63  U_Rx: UARTcomponent PORT MAP(
64    TXD => t_txd2,
65    RXD => RxD,
66    CLK => clock_50MHz,
```

```
67    DBIN => t_DBin,
68    DBOUT => dataout,
69    RDA => t_rda2,
70    TBE => t_tbe2,
71    RD => '0',
72    WR => '0',
73    PE => t_pe2,
74    FE => t_fe2,
75    OE => t_oe2,
76    RST => reset
77  );
78
79
80  end Behavioral;
```

Codice Componente 11.6: Definizione del componente UART$_PC$

11.7.2.2 File UCF

```
1
2
3   ##Leds
4
5   NET "dataout<0>"  LOC = "J14"; # Bank = 1, Pin name = IO_L14N_1/A3/RHCLK7,
        Type = RHCLK/DUAL, Sch name = JD10/LD0
6   NET "dataout<1>"  LOC = "J15"; # Bank = 1, Pin name = IO_L14P_1/A4/RHCLK6,
        Type = RHCLK/DUAL, Sch name = JD9/LD1
7   NET "dataout<2>"  LOC = "K15"; # Bank = 1, Pin name = IO_L12P_1/A8/RHCLK2,
        Type = RHCLK/DUAL, Sch name = JD8/LD2
8   NET "dataout<3>"  LOC = "K14"; # Bank = 1, Pin name = IO_L12N_1/A7/RHCLK3/
        TRDY1, Type = RHCLK/DUAL, Sch name = JD7/LD3
9   NET "dataout<4>" LOC = "E16"; # Bank = 1, Pin name = N.C., Type = N.C., Sch
        name = LD4? other than s3e500
10  NET "dataout<5>" LOC = "P16"; # Bank = 1, Pin name = N.C., Type = N.C., Sch
        name = LD5? other than s3e500
11  NET "dataout<6>" LOC = "E4";  # Bank = 3, Pin name = N.C., Type = N.C., Sch
        name = LD6? other than s3e500
12  NET "dataout<7>" LOC = "P4";  # Bank = 3, Pin name = N.C., Type = N.C., Sch
        name = LD7? other than s3e500
13
14  NET "clock_50MHz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
        = GCLK, Sch name = GCLK0
15
16
17  ##Switch
18  NET "datain<0>"       LOC = "G18";   # Sch name = SW0
19  NET "datain<1>"       LOC = "H18";   # Sch name = SW1
20  NET "datain<2>"       LOC = "K18";   # Sch name = SW2
```

```
21 NET "datain<3>"        LOC = "K17";    # Sch name = SW3
22 NET "datain<4>"        LOC = "L14";    # Sch name = SW4
23 NET "datain<5>"        LOC = "L13";    # Sch name = SW5
24 NET "datain<6>"        LOC = "N17";    # Sch name = SW6
25 NET "datain<7>"        LOC = "R17";    # Sch name = SW7
26
27
28
29 NET "reset" LOC = "B18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name =
       BTN0
30
31
32 ## RS232 connector
33 NET "RxD" LOC = "U6"; # Bank = 2, Pin name = IP, Type = INPUT, Sch name = RS
      -RX
34 NET "TxD" LOC = "P9"; # Bank = 2, Pin name = IO, Type = I/O, Sch name = RS-
      TX
```

Codice Componente 11.7: Definizione del file UCF

Capitolo 12

Esercizio 12

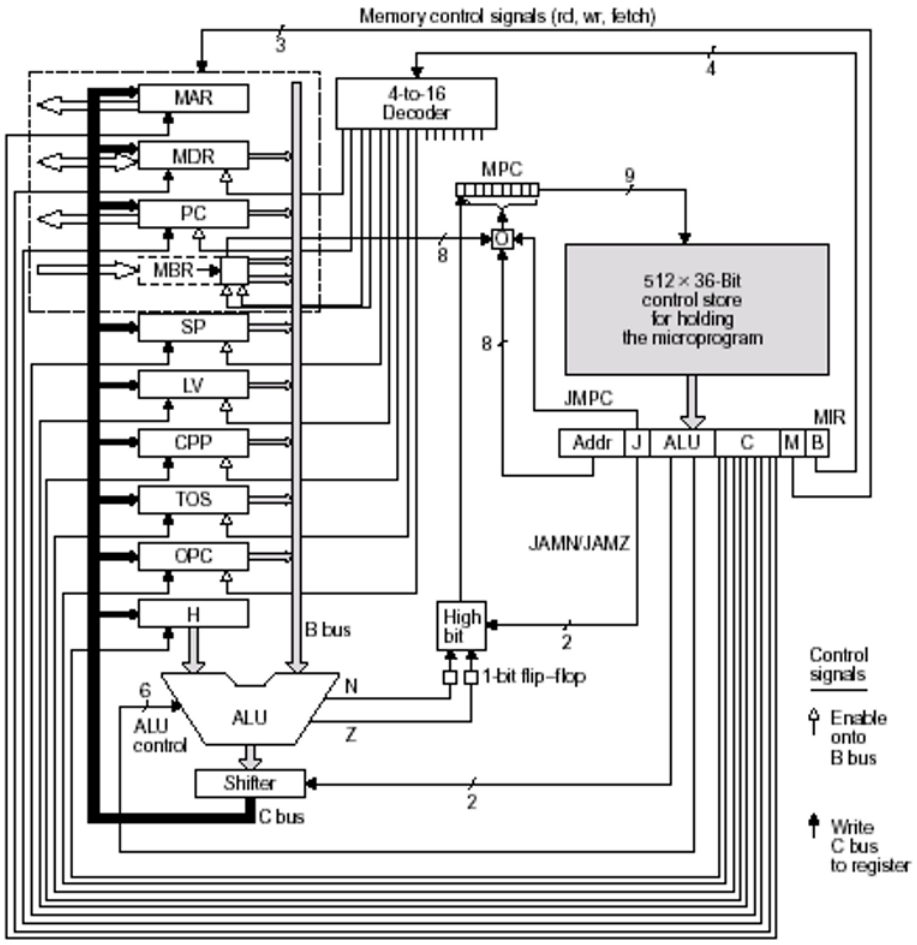
12.1 Traccia

A partire dall'implementazione fornita di un processore operante secondo il modello IJVM,

- a) si proceda all'analisi dell'architettura mediante simulazione e si approfondisca lo studio del suo funzionamento per due istruzioni a scelta,
- b) si modifichi un codice operativo a scelta, documentando tutte le modifiche effettuate,
- c) (opzionale) si descriva il funzionamento del processore in merito alle istruzioni di input/output,
- d) (solo ove possibile) si sintetizzi il processore su FPGA.

12.2 Soluzione a)

Il MIC-1 è un processore che opera secondo logica microprogrammata. Consiste di una unità di controllo che esegue microcodice prelevato da una memoria, detta control store, di 512 word. Il datapath è ottimizzato per il particolare set di istruzioni ISA ed è controllato da una microistruzione per ogni ciclo. Una lista di microistruzioni forma le microprocedure del programma. Nell'immagine successiva si mostra l'architettura MIC-1 completa di tutti i suoi componenti.



Il datapath contiene registri a 32 bit accessibili solo a livello microarchitetturale e 2 bus, B e C, per lettura e scrittura. L'unità logico aritmetica esegue le operazioni su due operandi : uno contenuto appunto nel registro H e l'altro proveniente dal bus B. Per eseguire una intera istruzione ISA, il control store deve essere in grado di calcolare l'indirizzo della microistruzione successiva da eseguire, che verrà inserita nell' MPC. Quando la ALU termina l'operazione, N e Z vengono salvati e utilizzati per determinare l'istruzione successiva :

1 - Il contenuto dei 9 bit NextAddress viene copiato in MPC

2 - Se JAM = "000", MPC = NextAddress altrimenti viene calcolato MPC;

3 - Se JAMN è settato ad "1" -> il bit più significativo di MPC viene messo in OR con il flag N. Se JAMZ è settato ad "1" -> il bit più significativo di MPC viene messo in OR con il flag Z. Se entrambi sono settati ad "1" -> il bit più significativo di MPC viene messo in OR con entrambi (F = (JAMN AND N) OR (JAMZ AND Z) OR bitNEXT_ADDRESS[8]). Se JAMC è settato ad "1" -> gli 8 bit di MBR sono messi in OR bit a bit con gli 8 bits meno significativi di MIR.

Le istruzioni scelte per la simulazione del processore del MIC1 vengono mostrate nel seguente codice IJVM :

```
1
2 .main
3 .var
4 a
```

```
.endvar
BIPUSH 0x56
ISTORE a
.endmethod
```

Codice Componente 12.1: Istruzioni

Il programma fa il push sullo stack del valore esadecimale 0x56,che viene successivamente memorizzato nella variabile "a".

Per poterlo tradurre in un linguaggio interpretabile dal processore MIC1 si usa il tool fornito su piattaforma Ubuntu lanciando il comando "make create_ram", che produce il risultato seguente:

```
@128
00000000000000000000000000000000
@0
00000001000000000000000100000000
00000001001101100101011000010000
```

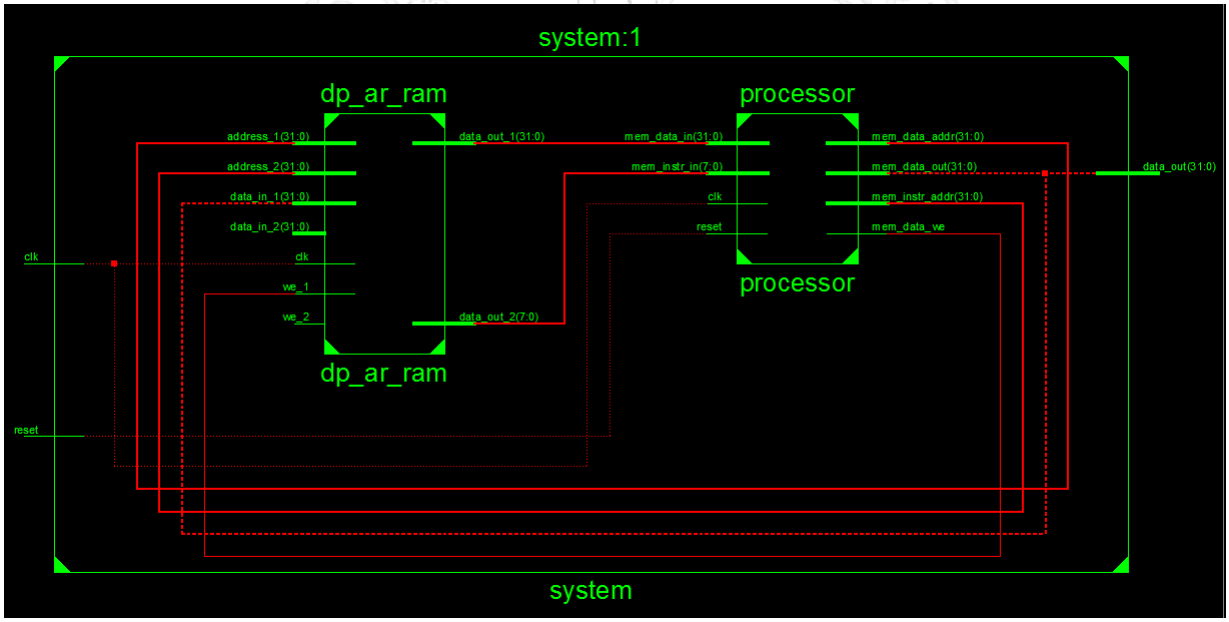
Codice Componente 12.2: Istruzioni

All'indirizzo 128 abbiamo la costant pool e le istruzioni all'indirizzo 0 e 1 (word da 32 bit). I dati vengono letti dal processore da destra verso sinistra a 8 bit alla volta tramite i registri dato e indirizzo.

La RAM viene successivamente caricata con tali valori e viene fatta partire la simulazione.

12.2.1 Schematici

Si mostra lo schematico relativo ai collegamenti tra RAM e processore MIC1. I due componenti si scambiano dati ed indirizzi permettendo al processore MIC1 di poter elaborare le istruzioni ed eseguirle, fornendo il dato in uscita tramite porta di output data_out.



12.2.2 Codice

Soltanto il file dp_ar_ram.vhd è stato modificato e il programma caricato nel blocco contenente le word.

```
1
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  use work.common_defs.all;
8
9  --! Data memory for amic-0 based systems
10
11 --! Dual port, asynchronous read RAM for amic-0 based systems.
12 entity dp_ar_ram is
13   port (
14     --! Clock
15     clk        : in  std_logic;
16     --! Write enable 1
17     we_1       : in  std_logic;
18     --! Port for memory write 1
19     data_in_1  : in  reg_data_type;
20     --! Port for memory read 1
21     data_out_1 : out reg_data_type;
22     --! Address for memory operations 1
23     address_1  : in  reg_data_type;
24     --! Write enable 2
25     we_2       : in  std_logic;
26     --! Port for memory write 2
27     data_in_2  : in  reg_data_type;
28     --! Port for memory read 2
29     data_out_2 : out mbr_data_type;
30     --! Address for memory operations 2
31     address_2  : in  reg_data_type
32     );
33 end entity dp_ar_ram;
34
35 --! Dataflow architecture for the control store
36 architecture behavioral of dp_ar_ram is
37
38   -- Signals
39   signal t_address_1  : integer := 0;
40   signal t_address_2  : integer := 0;
41   signal wa_address_2 : reg_data_type;
42   signal t_data_out_2 : reg_data_type;
43
44   -- RAM content
45   signal mem : dp_ar_ram_type := (
```

```
46 --BEGIN_WORDS_ENTRY
47 128 => "00000000000000000000000000000000",
48 0 => "00000001000000000000000100000000",
49 1 => "00000001001101100101011000010000",
50 others => (others => '0')
51 --END_WORDS_ENTRY
52     );
53
54
55
56
57 begin  -- architecture behavioral
58
59   wa_address_2 <= "00" & address_2(reg_data_type'high downto 2);
60   t_address_1 <= to_integer(unsigned(address_1));
61   t_address_2 <= to_integer(unsigned(wa_address_2));
62
63   mem_proc : process(clk) is
64   begin
65     if (rising_edge(clk)) then
66       if (we_1 = '1') then
67         mem(t_address_1) <= data_in_1;
68       elsif (we_2 = '1') then
69         mem(t_address_2) <= data_in_2;
70       end if;
71     end if;
72   end process;
73
74   data_out_1   <= mem(t_address_1);
75   t_data_out_2 <= mem(t_address_2);
76
77   with address_2(1 downto 0) select data_out_2 <=
78     t_data_out_2(7 downto 0)   when "00",
79     t_data_out_2(15 downto 8)  when "01",
80     t_data_out_2(23 downto 16) when "10",
81     t_data_out_2(31 downto 24) when "11",
82     (others => '0')            when others;
83
84 end architecture behavioral;
```

Codice Componente 12.3: RAM modificata col nuovo programma

12.3 Simulazione

Sono stati modificati i valori all'interno del process wavegen_proc del testbench processor_tb.

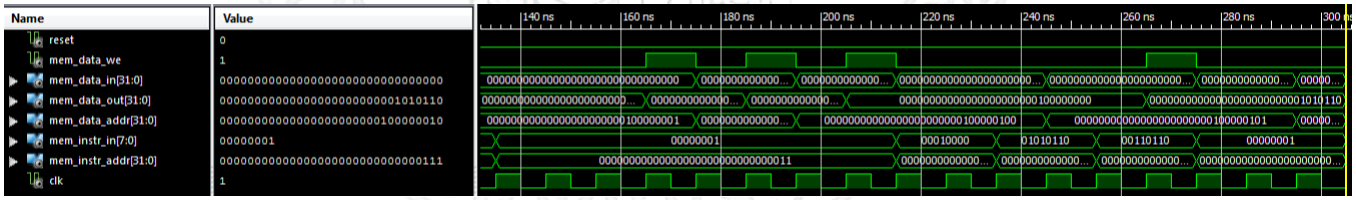
```
1
2 library ieee;
```

```
3  use ieee.std_logic_1164.all;
4
5  use work.common_defs.all;
6
7  --! Empty entity for the testbench
8  entity processor_tb is
9  end entity processor_tb;
10
11 --! Behavioral architecture for the testbench
12 architecture behavioral of processor_tb is
13
14   -- Component ports
15   signal reset          : std_logic;
16   signal mem_data_we    : std_logic;
17   signal mem_data_in    : reg_data_type;
18   signal mem_data_out   : reg_data_type;
19   signal mem_data_addr  : reg_data_type;
20   signal mem_instr_in   : mbr_data_type;
21   signal mem_instr_addr : reg_data_type;
22
23   -- Clock
24   signal clk : std_logic := '1';
25
26   -- Variables
27   shared variable end_run : boolean := false;
28
29 begin  -- architecture behavioral
30
31   -- Component instantiation
32   dut : entity work.processor
33     port map (
34       clk           => clk,
35       reset         => reset,
36       mem_data_we   => mem_data_we,
37       mem_data_in   => mem_data_in,
38       mem_data_out  => mem_data_out,
39       mem_data_addr => mem_data_addr,
40       mem_instr_in  => mem_instr_in,
41       mem_instr_addr => mem_instr_addr);
42
43   dp_ar_ram : entity work.dp_ar_ram
44     port map (
45       clk        => clk,
46       we_1       => mem_data_we,
47       data_in_1  => mem_data_out,
48       data_out_1 => mem_data_in,
49       address_1  => mem_data_addr,
50       we_2       => '0',
51       data_in_2  => (others => '0'),
```

```
52        data_out_2 => mem_instr_in,
53        address_2  => mem_instr_addr);
54
55    -- Clock generation
56    clk_proc : process
57    begin
58      while end_run = false loop
59        clk <= not clk;
60        wait for 5 ns;
61      end loop;
62
63      wait;
64    end process clk_proc;
65
66    -- Waveform generation
67    wavegen_proc: process
68    begin
69      wait until clk = '1';
70      wait for 2 ns;
71
72      reset <= '1';
73      wait for 10 ns;
74      reset <= '0';
75
76      wait until mem_instr_addr = x"00000007" and mem_data_we = '1';
77      assert mem_data_out = x"00000056" report "Bad calculated value" severity
            failure; -- Se il valore è diverso da 0x56
78
79
80      end_run := true;
81      wait;
82    end process wavegen_proc;
83
84  end architecture behavioral;
```

Codice Componente 12.4: Simulazione delle due istruzioni IJVM

Le istruzioni vengono correttamente elaborate e il valore in uscita a mem_data_out è la conversione in binario del valore esadecimale 0x56.



12.4 Soluzione b) IADD modificata

La prima modifica eseguita nei codici operativi IJVM è stata sulla IADD, che viene modificata in modo che essa funzioni come una sottrazione ISUB. Per far ciò è sufficiente modificare l'ultima istruzione, in modo che la ALU esegua una sottrazione invece che un' addizione tra i due operandi H e Memory Data Register.

```
1
2 iadd = 0x65:
3         MAR = SP = SP - 1; rd
4         H = TOS
5         MDR = TOS = MDR - H; wr; goto main
```

Codice Componente 12.5: Istruzione IADD modificata

Come possiamo notare, l'istruzione IADD adesso si comporta esattamente come una ISUB.

```
1
2 isub = 0x5C:
3         MAR = SP = SP - 1; rd
4         H = TOS
5         MDR = TOS = MDR - H; wr; goto main
```

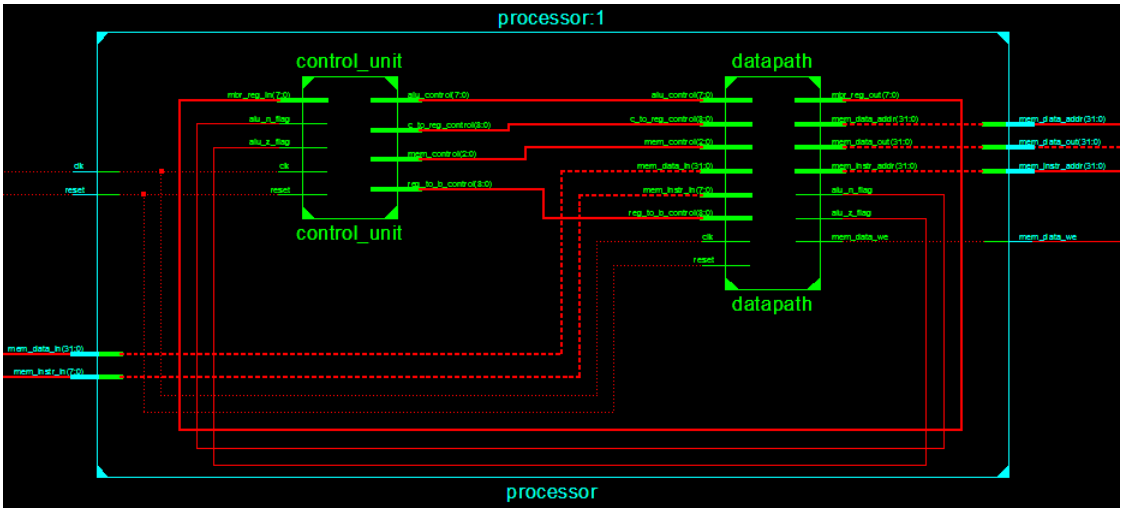
Codice Componente 12.6: Istruzione ISUB

Si testa il corretto funzionamento della IADD modificata facendo il push sullo stack di due valori esadecimali (entrambi 0x56) e memorizzandoli in due variabili a e b. Successivamente tali variabili vengono caricate e si richiama la nuova IADD che effettuerà la sottrazione tra i due elementi, il cui risultato previsto è 0. Il risultato della sottrazione viene memorizzato poi in b.

```
1  .main
2  .var
3  a
4  b
5  .endvar
6  BIPUSH   0x56
7  ISTORE   a
8  BIPUSH   0x56
9  ISTORE   b
10
11 ILOAD    a
12 ILOAD    b
13 IADD
14 ISTORE   b
15
16 .endmethod
```

Codice Componente 12.7: Programma assembler

12.4.1 Schematici



12.4.2 Codice

Il componente del MIC-1 Control Store contiene le definizioni di tutte le istruzioni IJVM composte da un codice operativo e in alcuni casi da un operando. Utilizzando i tool forniti e lanciando il comando "make create_control_store" tali istruzioni vengono ridefinite aggiungendo la nuova IADD. Il control store nell'implementazione fornita del MIC-1 è composto da 512 word da 32 bit che codificano le 112 istruzioni IJVM. Ne riportiamo una parte:

```
1
2  0 => "00000011000000000000000000000001001",
3  1 => "01011110000000000000000000000001001",
4  2 => "00000000000000000000000000000000000",
5  3 => "00000000000000000000000000000000000",
6  4 => "00000000000000000000000000000000000",
7  5 => "00000000000000000000000000000000000",
8  6 => "00000000010000110101000000100010001",
9  7 => "00000100000000110101000000100000001",
10 8 => "00000100100000110101000000100010001",
11 9 => "00000011000000000000000000000001001",
12 10 => "00000000000000000000000000000000000",
13 11 => "00000000000000000000000000000000000",
14 12 => "00000000000000000000000000000000000",
15 13 => "00000000000000000000000000000000000",
16 14 => "00000000000000000000000000000000000",
17 15 => "00000000000000000000000000000000000",
18 16 => "00001000100000110101000001001000100",
19 17 => "00001001000000011010100000010000010001",
20 18 => "00000011000000001010000100001010000010",
21 19 => "00000000000000000000000000000000000",
22 20 => "00000000000000000000000000000000000",
23 21 => "00001011000000001010010000000000101",
24 22 => "00001011100000011110000000000010100011",
```

```
23 => "00001100000000011010100000100100000100",
24 => "00001100100000011010100000010010100001",
25 => "00000011000000010100001000000000000000",
26 => "00000000000000000000000000000000000000",
27 => "00000000000000000000000000000000000000",
28 => "00000000000000000000000000000000000000",
29 => "00000000000000000000000000000000000000",
30 => "00000000000000000000000000000000000000",
31 => "00000000000000000000000000000000000000",
32 => "00010000100000110101000000100001000100",
33 => "00010001000010010100100000000000000011",
33 => "00010001000010010100100000000000000011",
34 => "00010001100000011100100000000000000011",
35 => "00001011100000111100000000000100100110",
36 => "00000000000000000000000000000000000000",
37 => "00000000000000000000000000000000000000",
38 => "00000000000000000000000000000000000000",
39 => "00000000000000000000000000000000000000",
40 => "00000000000000000000000000000000000000",
41 => "00000000000000000000000000000000000000",
42 => "00000000000000000000000000000000000000",
43 => "00000000000000000000000000000000000000",
44 => "00000000000000000000000000000000000000",
45 => "00000000000000000000000000000000000000",
46 => "00000000000000000000000000000000000000",
47 => "00000000000000000000000000000000000000",
48 => "00000000000000000000000000000000000000",
49 => "00000000000000000000000000000000000000",
50 => "00000000000000000000000000000000000000",
51 => "00000000000000000000000000000000000000",
52 => "00000000000000000000000000000000000000",
53 => "00000000000000000000000000000000000000",
54 => "00011011100000010100100000000000000101",
55 => "00011100000000111100000000010000000011",
56 => "00011100100000010100000000001010001111",
57 => "00011101000000110110000001001010010100",
58 => "00011101100000110101000000100001000100",
59 => "00000011000000010100001000000000000000",
60 => "00000000000000000000000000000000000000",
61 => "00000000000000000000000000000000000000",
62 => "00000000000000000000000000000000000000",
63 => "00000000000000000000000000000000000000",
64 => "00000000000000000000000000000000000000",
65 => "00000000000000000000000000000000000000",
66 => "00000000000000000000000000000000000000",
67 => "00000000000000000000000000000000000000",
68 => "00000000000000000000000000000000000000",
69 => "00000000000000000000000000000000000000",
70 => "00000000000000000000000000000000000000",
```

```
71 => "00000000000000000000000000000000",
72 => "00000000000000000000000000000000",
73 => "00000000000000000000000000000000",
74 => "00000000000000000000000000000000",
75 => "00000000000000000000000000000000",
76 => "00000000000000000000000000000000",
77 => "00000000000000000000000000000000",
78 => "00000000000000000000000000000000",
79 => "00000000000000000000000000000000",
80 => "00000000000000000000000000000000",
81 => "00000000000000000000000000000000",
82 => "00000000000000000000000000000000",
83 => "00000000000000000000000000000000",
84 => "00000000000000000000000000000000",
85 => "00000000000000000000000000000000",
86 => "00000000000000000000000000000000",
87 => "00101100000000110101000001001000100",
88 => "00000011000000010100000000001010000111",
89 => "00101101000000011011000000100101001000",
90 => "00101101100000000000000000000001001",
91 => "00000011000000010100001000000000000",
92 => "00101110100000110110000001001010010000",
93 => "00101111000000010100100000000000111",
94 => "00000011000000011111100100001010000000",
95 => "00110000000000110110000000001010010000",
96 => "00110000100000010100000000001000010000",
97 => "00110001000000010100100000000010000000",
98 => "00110001100000010100000000010000000111",
99 => "00110010000000110110000000001100010000",
100 => "00000011000000011000001000000001001",
101 => "00110011000000110110000001001010010000",
102 => "00110011100000010100100000000000111",
103 => "00000011000000011111100100001010000000",
104 => "00000000000000000000000000000000",
105 => "00000000000000000000000000000000",
106 => "00000000000000000000000000000000",
107 => "00000000000000000000000000000000",
108 => "00000000000000000000000000000000",
109 => "00000000000000000000000000000000",
110 => "00000000000000000000000000000000",
111 => "00000000000000000000000000000000",
112 => "00000000000000000000000000000000",
113 => "00000000000000000000000000000000",
114 => "00000000000000000000000000000000",
115 => "00000000000000000000000000000000",
116 => "00000000000000000000000000000000",
117 => "00000000000000000000000000000000",
118 => "00000000000000000000000000000000",
119 => "00000000000000000000000000000000",
```

```
120 => "00000000000000000000000000000000",
121 => "00000000000000000000000000000000",
122 => "00000000000000000000000000000000",
123 =>"00000000000000000000000000000000",
124 => "00000000000000000000000000000000",
125 => "00000000000000000000000000000000",
126 => "00111111100000110110000010010100100",
127 => "01000000000000010100100000000000111",
128 => "00000011000000001100001000010100000",
129 => "00000000000000000000000000000000",
130 => "00000000000000000000000000000000",
131 => "00000000000000000000000000000000",
132 => "01000010100000010100100000000000101",
133 => "01000011000000011110000000000010100011",
134 => "01000011100000011010100000010000010001",
135 => "01000100000000010100100000000000000000",
136 => "01000100100000011010100000010000010001",
137 => "00000011000000011110000000000101000010",
138 => "00000000000000000000000000000000",
139 => "00000000000000000000000000000000",
140 => "00000000000000000000000000000000",
141 => "00000000000000000000000000000000",
142 => "00000000000000000000000000000000",
143 => "00000000000000000000000000000000",
144 => "00000000000000000000000000000000",
145 => "00000000000000000000000000000000",
146 => "00000000000000000000000000000000",
147 => "00000000000000000000000000000000",
148 => "00000000000000000000000000000000",
149 => "00000000000000000000000000000000",
150 => "00000000000000000000000000000000",

...
```

Codice Componente 12.8: Nuovo Control$_{store}$

Il programma infine è stato tradotto e caricato in RAM come visto nel paragrafo precedente.

```


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.common_defs.all;

--! Data memory for amic-0 based systems

--! Dual port, asynchronous read RAM for amic-0 based systems.
entity dp_ar_ram is
```

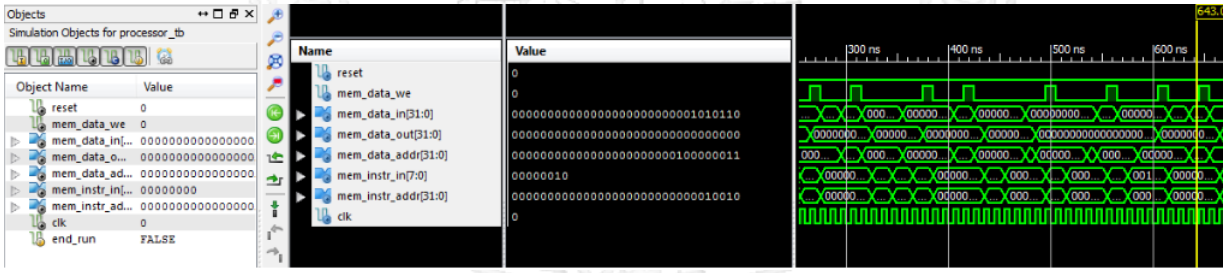
```
13    port (
14      --! Clock
15      clk        : in  std_logic;
16      --! Write enable 1
17      we_1       : in  std_logic;
18      --! Port for memory write 1
19      data_in_1  : in  reg_data_type;
20      --! Port for memory read 1
21      data_out_1 : out reg_data_type;
22      --! Address for memory operations 1
23      address_1  : in  reg_data_type;
24      --! Write enable 2
25      we_2       : in  std_logic;
26      --! Port for memory write 2
27      data_in_2  : in  reg_data_type;
28      --! Port for memory read 2
29      data_out_2 : out mbr_data_type;
30      --! Address for memory operations 2
31      address_2  : in  reg_data_type
32      );
33  end entity dp_ar_ram;
34
35  --! Dataflow architecture for the control store
36  architecture behavioral of dp_ar_ram is
37
38    -- Signals
39    signal t_address_1  : integer := 0;
40    signal t_address_2  : integer := 0;
41    signal wa_address_2 : reg_data_type;
42    signal t_data_out_2 : reg_data_type;
43
44    -- RAM content
45    signal mem : dp_ar_ram_type := (
46  --BEGIN_WORDS_ENTRY
47
48  128 => "00000000000000000000000000000000",
49  0 => "00000010000000000000000100000000",
50  1 => "00000001001101100101011000010000",
51  2 => "00000010001101100101011000010000",
52  3 => "00000010000101010000000100010101",
53  4 => "00000000000000100011011001100101",
54
55  others => (others => '0')
56  --END_WORDS_ENTRY
57      );
58
59
60
61  begin  -- architecture behavioral
```

```
62
63    wa_address_2 <= "00" & address_2(reg_data_type'high downto 2);
64    t_address_1 <= to_integer(unsigned(address_1));
65    t_address_2 <= to_integer(unsigned(wa_address_2));
66
67    mem_proc : process(clk) is
68    begin
69      if (rising_edge(clk)) then
70        if (we_1 = '1') then
71          mem(t_address_1) <= data_in_1;
72        elsif (we_2 = '1') then
73          mem(t_address_2) <= data_in_2;
74        end if;
75      end if;
76    end process;
77
78    data_out_1   <= mem(t_address_1);
79    t_data_out_2 <= mem(t_address_2);
80
81    with address_2(1 downto 0) select data_out_2 <=
82      t_data_out_2(7 downto 0)   when "00",
83      t_data_out_2(15 downto 8)  when "01",
84      t_data_out_2(23 downto 16) when "10",
85      t_data_out_2(31 downto 24) when "11",
86      (others => '0')            when others;
87
88  end architecture behavioral;
```

Codice Componente 12.9: dp$_a r_r am.vhd$

12.5 Simulazione

Il testbench utilizzato è processor_tb, che simula il comportamento del processore e della RAM. Esso è del tutto simile a quello del paragrafo precedente e per questo non viene mostrato. E' sufficiente osservare direttamente il comportamento delle forme d'onda, una volta lanciata la simulazione con ISim.



Dopo aver eseguito tutte le istruzioni IJVM del programma, il dato in uscita da mem_data_out è 0, come atteso.

12.6 Soluzione b) ISUB modificata

La ISUB è stata modificata in questo modo : i due operandi vengono immessi nello stack e viene effettuata la sottrazione, tuttavia essi non vengono rimossi e il risultato finale dell'operazione viene posto in cima allo stack. Per far ciò si è incrementato il puntatore SP di 1 subito dopo aver messo il contenuto di SP-1 in MAR per la lettura dell'operando.

Poichè le istruzioni IJVM sono memorizzate in una porzione di memoria limitata bisogna modificare gli indirizzi di partenza. Lanciare il comando senza effettuare le modifiche agli indirizzi porterà ad un errore di "Invalid control store region". Nel nostro caso l'istruzione IJVM immediatamente successiva alla ISUB è la swap, che è stata rimossa .

```
1
2 isub = 0x5C:
3         MAR = SP - 1; rd
4     MAR = SP = SP + 1
5         H = TOS
6         MDR = TOS = MDR - H; wr; goto main
```

Codice Componente 12.10: Istruzione ISUB modificata

Si effettua il push sullo stack dei valori esadecimali 0x56 e 0x55. La sottrazione tra essi darà come risultato 1. I valori vengono inseriti nello stack e vengono memorizzati in a e b. La ISUB modificata fa in modo tale da non rimuoverli dallo stack e posizionare il risultato in cima.

```
1  .main
2  .var
3  a
4  b
5  .endvar
6
7  BIPUSH   0x56
8  ISTORE   a
9  BIPUSH   0x55
10 ISTORE   b
11
12 ILOAD    a
13 ILOAD    b
14 ISUB
15
16 .endmethod
```

Codice Componente 12.11: Programma assembler

12.6.1 Codice

Come nel precedente esempio sono state ricreate le word del Control Store e il file control_store.vhd è stato modificato.

```
1
2 library ieee;
```

```
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  use work.common_defs.all;
7
8  --! Processor control store
9
10 --! The control store is a ROM used to store the processor microprogram.
11 entity control_store is
12   port (
13     --! Address of the desired word
14     address : in  ctrl_str_addr_type;
15     --! Content of the addressed word
16     word    : out ctrl_str_word_type
17     );
18 end entity control_store;
19
20 --! Dataflow architecture for the control store
21 architecture dataflow of control_store is
22
23   -- Constants
24   constant words : ctrl_str_type := (
25 --BEGIN_WORDS_ENTRY0 => "00000011000000000000000000000001001",
26
27
28 0 => "00000100100000000000000000000001001",
29 1 => "01011110000000000000000000000001001",
30 2 => "00000000000000000000000000000000000",
31 3 => "00000000000000000000000000000000000",
32 4 => "00000000000000000000000000000000000",
33 5 => "00000000000000000000000000000000000",
34 6 => "00000011100000110101000000100000001",
35 7 => "00000100000000110101000000100001001",
36 8 => "00000100100000000000000000000001001",
37 9 => "00000000010000110101000000100001001",
38 10 => "00000000000000000000000000000000000",
39 11 => "00000000000000000000000000000000000",
40 12 => "00000000000000000000000000000000000",
41 13 => "00000000000000000000000000000000000",
42 14 => "00000000000000000000000000000000000",
43 15 => "00000000000000000000000000000000000",
44 16 => "00001000100000110101000001001000100",
45 17 => "00000100100000011010100000010000100",
46 18 => "00000100100000010100001000010100010",
47 19 => "00000000000000000000000000000000000",
48 20 => "00000000000000000000000000000000000",
49 21 => "00001011000000010100100000000000101",
50 22 => "00001011100000111100000000010100011",
51 23 => "00001100000000110101000001001000100",
```

```
24 => "00001100100000110101000000100101000 1",
25 => "00000100100000010100001000000000000",
26 => "00000000000000000000000000000000000",
27 => "00000000000000000000000000000000000",
28 => "00000000000000000000000000000000000",
29 => "00000000000000000000000000000000000",
30 => "00000000000000000000000000000000000",
31 => "00000000000000000000000000000000000",
32 => "00010000100000110101000000100001000 1",
33 => "00010001000010010100100000000000011",
34 => "00010001100000011100100000000000011",
35 => "00001011100000111100000000010100110",
36 => "00000000000000000000000000000000000",
37 => "00000000000000000000000000000000000",
38 => "00000000000000000000000000000000000",
39 => "00000000000000000000000000000000000",
40 => "00000000000000000000000000000000000",
41 => "00000000000000000000000000000000000",
42 => "00000000000000000000000000000000000",
43 => "00000000000000000000000000000000000",
44 => "00000000000000000000000000000000000",
45 => "00000000000000000000000000000000000",
46 => "00000000000000000000000000000000000",
47 => "00000000000000000000000000000000000",
48 => "00000000000000000000000000000000000",
49 => "00000000000000000000000000000000000",
50 => "00000000000000000000000000000000000",
51 => "00000000000000000000000000000000000",
52 => "00000000000000000000000000000000000",
53 => "00000000000000000000000000000000000",
54 => "00011011100000010100100000000000101",
55 => "00011100000000111100000000010000011",
56 => "00011100100000010100000000001010001 11",
57 => "00011101000000110110000001001010010 0",
58 => "00011101100000110101000000100001000 1",
59 => "00000100100000010100001000000000000",
60 => "00000000000000000000000000000000000",
61 => "00000000000000000000000000000000000",
62 => "00000000000000000000000000000000000",
63 => "00000000000000000000000000000000000",
64 => "00000000000000000000000000000000000",
65 => "00000000000000000000000000000000000",
66 => "00000000000000000000000000000000000",
67 => "00000000000000000000000000000000000",
68 => "00000000000000000000000000000000000",
69 => "00000000000000000000000000000000000",
70 => "00000000000000000000000000000000000",
71 => "00000000000000000000000000000000000",
72 => "00000000000000000000000000000000000",
```

```
73 => "00000000000000000000000000000000000000",
74 => "00000000000000000000000000000000000000",
75 => "00000000000000000000000000000000000000",
76 => "00000000000000000000000000000000000000",
77 => "00000000000000000000000000000000000000",
78 => "00000000000000000000000000000000000000",
79 => "00000000000000000000000000000000000000",
80 => "00000000000000000000000000000000000000",
81 => "00000000000000000000000000000000000000",
82 => "00000000000000000000000000000000000000",
83 => "00000000000000000000000000000000000000",
84 => "00000000000000000000000000000000000000",
85 => "00000000000000000000000000000000000000",
86 => "00000000000000000000000000000000000000",
87 => "00101100000000110101000001001000010000",
88 => "00000100100000001010000000001010000111",
89 => "00101101000000110110000001001010010000",
90 => "00101101100000000000000000000000001001",
91 => "00000100100000001010000100000000000000",
92 => "00101110100000110110000000001010010000",
93 => "00101111000000110101000001001000010000",
94 => "00101111100000001010010000000000000111",
95 => "00000100100000011111100100001010000000",
96 => "00000000000000000000000000000000000000",
97 => "00000000000000000000000000000000000000",
98 => "00000000000000000000000000000000000000",
99 => "00000000000000000000000000000000000000",
100 => "00000000000000000000000000000000000000",
101 => "00110011000000110110000001001010010000",
102 => "00110011100000010100100000000000000111",
103 => "00000100100000011110000100001010000000",
104 => "00000000000000000000000000000000000000",
105 => "00000000000000000000000000000000000000",
106 => "00000000000000000000000000000000000000",
107 => "00000000000000000000000000000000000000",
108 => "00000000000000000000000000000000000000",
109 => "00000000000000000000000000000000000000",
110 => "00000000000000000000000000000000000000",
111 => "00000000000000000000000000000000000000",
112 => "00000000000000000000000000000000000000",
113 => "00000000000000000000000000000000000000",
114 => "00000000000000000000000000000000000000",
115 => "00000000000000000000000000000000000000",
116 => "00000000000000000000000000000000000000",
117 => "00000000000000000000000000000000000000",
118 => "00000000000000000000000000000000000000",
119 => "00000000000000000000000000000000000000",
120 => "00000000000000000000000000000000000000",
121 => "00000000000000000000000000000000000000",
```

```
122 => "00000000000000000000000000000000",
123 => "00000000000000000000000000000000",
124 => "00000000000000000000000000000000",
125 => "00000000000000000000000000000000",
126 => "00111111100000110110000010010100100",
127 => "01000000000000010100100000000000111",
128 => "00000100100000001100001000010100000",
129 => "00000000000000000000000000000000",
130 => "00000000000000000000000000000000",
131 => "00000000000000000000000000000000",
132 => "01000010100000010100100000000000101",
133 => "01000011000000011110000000000010100011",
134 => "01000011100000011010100000010000010001",
135 => "01000100000000010100100000000000000000",
136 => "01000100100000011010100000010000010001",
137 => "00000100100000011110000000001010000010",
138 => "00000000000000000000000000000000",
139 => "00000000000000000000000000000000",
140 => "00000000000000000000000000000000",
141 => "00000000000000000000000000000000",
142 => "00000000000000000000000000000000",
143 => "00000000000000000000000000000000",
144 => "00000000000000000000000000000000",
145 => "00000000000000000000000000000000",
146 => "00000000000000000000000000000000",
147 => "00000000000000000000000000000000",
148 => "00000000000000000000000000000000",
149 => "00000000000000000000000000000000",
150 => "00000000000000000000000000000000",
151 => "00000000000000000000000000000000",
152 => "00000000000000000000000000000000",
153 => "01001101000000110110000001001010100",
154 => "01001101100000010100010000000000111",
155 => "01001110000000010100001000000000000",
156 => "00000011000100010100000000000001000",
157 => "01001111000000110110000001001010100",
158 => "01001111100000010100010000000000111",
159 => "01010000000000010100001000000000000",
160 => "00000011001000010100000000000001000",
161 => "01010001000000110110000001001010100",
162 => "01010001100000110110000001001010000100",
163 => "01010010000000010100100000000000100000",
164 => "01010010100000010100010000000000111",
165 => "01010011000000010100001000000000000",
166 => "00000011000100111111100000000000001000",
167 => "01010100000000110110010000000000000001",
168 => "01010100100000110101000000010000010001",
169 => "01010101000010010100100000000000000010",
170 => "01010101100000011100100000000000000011",
```

```
171 => "01010110000000111100000000010000011000",
172 => "00000100100000000000000000000000001001",
173 => "01010111000000010100000000100101001010",
174 => "01010111100000000000000000000000001001",
175 => "01011000000000001010000001000101000000",
176 => "01011000100000110101000000000100000101",
177 => "01011001000000010100000000010001100000",
178 => "01011001100000010100000000000100000100",
179 => "01011010000000001010000001000000000000",
180 => "01011010100000010100000000001010001111",
181 => "00000100100100110110000000000000000001",
182 => "01011011100000110110000000100101001000",
183 => "01011100000000001010010000000000000111",
184 => "00000100100000011100001000010100000000",
185 => "01011101000000110101000000010000100001",
186 => "01011101100010010100100000000000000011",
187 => "01011110000000011100100000000000000011",
188 => "01011110100000111100000000000101001100",
189 => "01011111000000110101010000000000000001",
190 => "01011111100000010100000000010000100000",
191 => "01100000000000110101000000010000100001",
192 => "01100000100010010100100000000000000011",
193 => "01100001000000011100100000000000000011",
194 => "01100001100000110101000000010000100001",
195 => "01100010000000011111100100000000000100",
196 => "01100010100000110101001000001000001111",
197 => "01100011000000110101000000010000100001",
198 => "01100011100010010100100000000000000011",
199 => "01100100000000011100100000000000000011",
200 => "01100100100000111101000000010100010000",
201 => "01100101000000010100000000100100000000",
202 => "01100101100000010100000000001010010000",
203 => "01100110000000110101000001001000010000",
204 => "01100110100000010100000000001010001010",
205 => "01100111000000110101000000010000100001",
206 => "00000100100000010100000010000000000111",
207 => "10000000010000110101000000010000100001",
208 => "00000000000000000000000000000000000000",
209 => "00000000000000000000000000000000000000",
210 => "00000000000000000000000000000000000000",
211 => "00000000000000000000000000000000000000",
212 => "00000000000000000000000000000000000000",
213 => "00000000000000000000000000000000000000",
214 => "00000000000000000000000000000000000000",
215 => "00000000000000000000000000000000000000",
216 => "00000000000000000000000000000000000000",
217 => "00000000000000000000000000000000000000",
218 => "00000000000000000000000000000000000000",
219 => "00000000000000000000000000000000000000",
```

```
220 => "00000000000000000000000000000000",
221 => "00000000000000000000000000000000",
222 => "00000000000000000000000000000000",
223 => "00000000000000000000000000000000",
224 => "00000000000000000000000000000000",
225 => "00000000000000000000000000000000",
226 => "00000000000000000000000000000000",
227 => "00000000000000000000000000000000",
228 => "00000000000000000000000000000000",
229 => "00000000000000000000000000000000",
230 => "00000000000000000000000000000000",
231 => "00000000000000000000000000000000",
232 => "00000000000000000000000000000000",
233 => "00000000000000000000000000000000",
234 => "00000000000000000000000000000000",
235 => "00000000000000000000000000000000",
236 => "00000000000000000000000000000000",
237 => "00000000000000000000000000000000",
238 => "00000000000000000000000000000000",
239 => "00000000000000000000000000000000",
240 => "00000000000000000000000000000000",
241 => "00000000000000000000000000000000",
242 => "00000000000000000000000000000000",
243 => "00000000000000000000000000000000",
244 => "00000000000000000000000000000000",
245 => "00000000000000000000000000000000",
246 => "00000000000000000000000000000000",
247 => "00000000000000000000000000000000",
248 => "00000000000000000000000000000000",
249 => "00000000000000000000000000000000",
250 => "00000000000000000000000000000000",
251 => "00000000000000000000000000000000",
252 => "00000000000000000000000000000000",
253 => "00000000000000000000000000000000",
254 => "00000000000000000000000000000000",
255 => "00000000000000000000000000000000",
256 => "00000000000000000000000000000000",
257 => "00000000000000000000000000000000",
258 => "00000000000000000000000000000000",
259 => "00000000000000000000000000000000",
260 => "00000000000000000000000000000000",
261 => "00000000000000000000000000000000",
262 => "01010100000000110110010000000010001",
263 => "00000000000000000000000000000000",
264 => "00000000000000000000000000000000",
265 => "10000100100000000000000000000001001",
266 => "00000000000000000000000000000000",
267 => "00000000000000000000000000000000",
268 => "00000000000000000000000000000000",
```

```
297 | 269 => "00000000000000000000000000000000",
298 | 270 => "00000000000000000000000000000000",
299 | 271 => "00000000000000000000000000000000",
300 | 272 => "00000000000000000000000000000000",
301 | 273 => "00000000000000000000000000000000",
302 | 274 => "00000000000000000000000000000000",
303 | 275 => "00000000000000000000000000000000",
304 | 276 => "00000000000000000000000000000000",
305 | 277 => "10001011000000110101000000100010001",
306 | 278 => "10001011100010010100100000000000011",
307 | 279 => "10001100000000011100100000000000011",
308 | 280 => "00001011100000111100000000010100101",
309 | 281 => "00000000000000000000000000000000",
310 | 282 => "00000000000000000000000000000000",
311 | 283 => "00000000000000000000000000000000",
312 | 284 => "00000000000000000000000000000000",
313 | 285 => "00000000000000000000000000000000",
314 | 286 => "00000000000000000000000000000000",
315 | 287 => "00000000000000000000000000000000",
316 | 288 => "00000000000000000000000000000000",
317 | 289 => "00000000000000000000000000000000",
318 | 290 => "00000000000000000000000000000000",
319 | 291 => "00000000000000000000000000000000",
320 | 292 => "00000000000000000000000000000000",
321 | 293 => "00000000000000000000000000000000",
322 | 294 => "00000000000000000000000000000000",
323 | 295 => "00000000000000000000000000000000",
324 | 296 => "00000000000000000000000000000000",
325 | 297 => "00000000000000000000000000000000",
326 | 298 => "00000000000000000000000000000000",
327 | 299 => "00000000000000000000000000000000",
328 | 300 => "00000000000000000000000000000000",
329 | 301 => "00000000000000000000000000000000",
330 | 302 => "00000000000000000000000000000000",
331 | 303 => "00000000000000000000000000000000",
332 | 304 => "00000000000000000000000000000000",
333 | 305 => "00000000000000000000000000000000",
334 | 306 => "00000000000000000000000000000000",
335 | 307 => "00000000000000000000000000000000",
336 | 308 => "00000000000000000000000000000000",
337 | 309 => "00000000000000000000000000000000",
338 | 310 => "10011011100000110101000000100010001",
339 | 311 => "10011100000010010100100000000000011",
340 | 312 => "10011100100000011100100000000000011",
341 | 313 => "00011100000000011110000000010000101",
342 | 314 => "00000000000000000000000000000000",
343 | 315 => "00000000000000000000000000000000",
344 | 316 => "00000000000000000000000000000000",
345 | 317 => "00000000000000000000000000000000",
```

```
346 318 => "00000000000000000000000000000000",
347 319 => "00000000000000000000000000000000",
348 320 => "00000000000000000000000000000000",
349
350 ...
351
352 others => (others => '0')
353 --END_WORDS_ENTRY
354     );
355
356 begin  -- architecture dataflow
357
358   word <= words(to_integer(unsigned(address)));
359
360 end architecture dataflow;
```

Codice Componente 12.12: Nuovo Control*store*

Il programma infine è stato tradotto e caricato in RAM come visto nel paragrafo precedente.

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  use work.common_defs.all;
7
8  --! Data memory for amic-0 based systems
9
10 --! Dual port, asynchronous read RAM for amic-0 based systems.
11 entity dp_ar_ram is
12   port (
13     --! Clock
14     clk        : in  std_logic;
15     --! Write enable 1
16     we_1       : in  std_logic;
17     --! Port for memory write 1
18     data_in_1  : in  reg_data_type;
19     --! Port for memory read 1
20     data_out_1 : out reg_data_type;
21     --! Address for memory operations 1
22     address_1  : in  reg_data_type;
23     --! Write enable 2
24     we_2       : in  std_logic;
25     --! Port for memory write 2
26     data_in_2  : in  reg_data_type;
27     --! Port for memory read 2
28     data_out_2 : out mbr_data_type;
29     --! Address for memory operations 2
30     address_2  : in  reg_data_type
```

```
31      );
32  end entity dp_ar_ram;
33
34  --! Dataflow architecture for the control store
35  architecture behavioral of dp_ar_ram is
36
37    -- Signals
38    signal t_address_1  : integer := 0;
39    signal t_address_2  : integer := 0;
40    signal wa_address_2 : reg_data_type;
41    signal t_data_out_2 : reg_data_type;
42
43    -- RAM content
44    signal mem : dp_ar_ram_type := (
45  --BEGIN_WORDS_ENTRY
46
47  128 => "00000000000000000000000000000000",
48  0 => "00000010000000000000000100000000",
49  1 => "00000001001101100101011000010000",
50  2 => "00000010001101100101010100010000",
51  3 => "00000010000101010000000100010101",
52  4 => "00000000000000000000000000011100",
53
54  others => (others => '0')
55  --END_WORDS_ENTRY
56      );
57
58  begin  -- architecture behavioral
59
60    wa_address_2 <= "00" & address_2(reg_data_type'high downto 2);
61    t_address_1 <= to_integer(unsigned(address_1));
62    t_address_2 <= to_integer(unsigned(wa_address_2));
63
64    mem_proc : process(clk) is
65    begin
66      if (rising_edge(clk)) then
67        if (we_1 = '1') then
68          mem(t_address_1) <= data_in_1;
69        elsif (we_2 = '1') then
70          mem(t_address_2) <= data_in_2;
71        end if;
72      end if;
73    end process;
74
75    data_out_1   <= mem(t_address_1);
76    t_data_out_2 <= mem(t_address_2);
77
78    with address_2(1 downto 0) select data_out_2 <=
79      t_data_out_2(7 downto 0)   when "00",
```

```
80      t_data_out_2(15 downto 8)  when "01",
81      t_data_out_2(23 downto 16) when "10",
82      t_data_out_2(31 downto 24) when "11",
83      (others => '0')            when others;
84
85  end architecture behavioral;
```

Codice Componente 12.13: Istruzioni

12.7 Simulazione

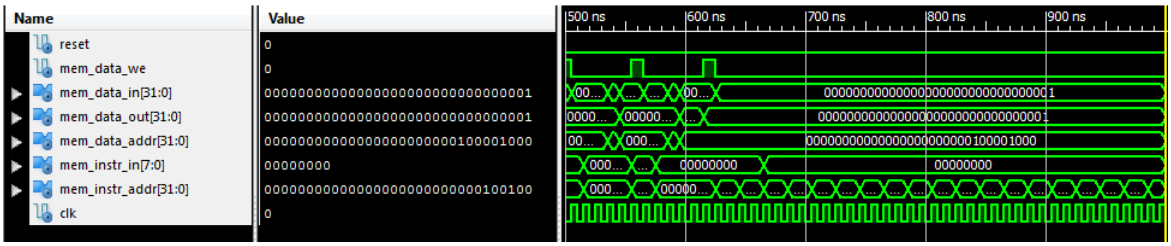
Il testbench utilizzato è processor_tb, simile al precedente, resetta il processore che elabora il programma caricato nella memoria RAM.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   use work.common_defs.all;
5
6   --! Empty entity for the testbench
7   entity processor_tb is
8   end entity processor_tb;
9
10  --! Behavioral architecture for the testbench
11  architecture behavioral of processor_tb is
12
13    -- Component ports
14    signal reset          : std_logic;
15    signal mem_data_we    : std_logic;
16    signal mem_data_in    : reg_data_type;
17    signal mem_data_out   : reg_data_type;
18    signal mem_data_addr  : reg_data_type;
19    signal mem_instr_in   : mbr_data_type;
20    signal mem_instr_addr : reg_data_type;
21
22    -- Clock
23    signal clk : std_logic := '1';
24
25    -- Variables
26    shared variable end_run : boolean := false;
27
28  begin  -- architecture behavioral
29
30    -- Component instantiation
31    dut : entity work.processor
32      port map (
33        clk            => clk,
34        reset          => reset,
```

```
35        mem_data_we    => mem_data_we,
36        mem_data_in    => mem_data_in,
37        mem_data_out   => mem_data_out,
38        mem_data_addr  => mem_data_addr,
39        mem_instr_in   => mem_instr_in,
40        mem_instr_addr => mem_instr_addr);
41
42    dp_ar_ram : entity work.dp_ar_ram
43      port map (
44        clk        => clk,
45        we_1       => mem_data_we,
46        data_in_1  => mem_data_out,
47        data_out_1 => mem_data_in,
48        address_1  => mem_data_addr,
49        we_2       => '0',
50        data_in_2  => (others => '0'),
51        data_out_2 => mem_instr_in,
52        address_2  => mem_instr_addr);
53
54    -- Clock generation
55    clk_proc : process
56    begin
57      while end_run = false loop
58        clk <= not clk;
59        wait for 5 ns;
60      end loop;
61
62      wait;
63    end process clk_proc;
64
65    -- Waveform generation
66    wavegen_proc: process
67    begin
68      wait until clk = '1';
69      wait for 2 ns;
70
71      reset <= '1';
72      wait for 10 ns;
73      reset <= '0';
74
75      wait;
76    end process wavegen_proc;
77
78  end architecture behavioral;
```

Codice Componente 12.14: Simulazione delle due istruzioni IJVM

Alla fine dell'elaborazione il processore fornisce in output (mem_data_out) il valore di sottrazione di 0x56 e 0x55, cioè 1.


Name
Value
reset
mem_data_we
mem_data_in[31:0]
mem_data_out[31:0]
mem_data_addr[31:0]
mem_instr_in[7:0]
mem_instr_addr[31:0]
clk
0
0
00000000000000000000000000000001
00000000000000000000000000000001
00000000000000000000000100001000
00000000
00000000000000000000000100100
0
500 ns
600 ns
700 ns
800 ns
900 ns

Capitolo 13

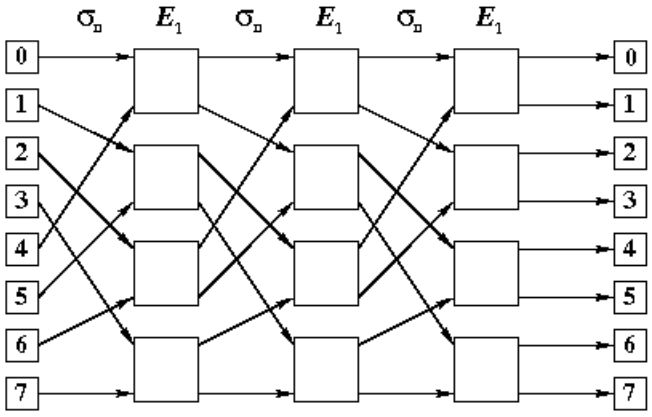
Esercizio 13

13.1 Traccia

Progettare ed implementare in VHDL uno switch multistadio secondo il modello omega network, descritto nelle dispense fornite nella cartella "SWITCH". Lo switch progettato deve consentire lo scambio di messaggi di 2 bit ciascuno fra 4 nodi diversi, tra i quali deve essere realizzato un handshaking semplice regolato da una coppia di segnali (pronto a inviare/pronto a ricevere). L'indirizzo del nodo destinazione, espresso su 2 bit, viene inviato insieme ai 2 bit di informazione per consentire l'avanzamento dei messaggi. Qualora fosse necessario gestire possibili conflitti, ilsistema può operare perdendo uno dei messaggi in conflitto (sistema con perdita).

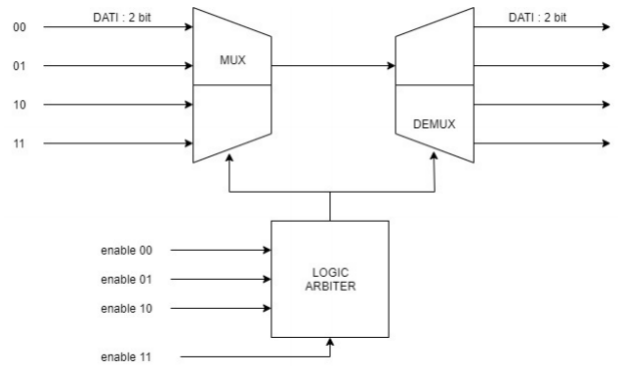
13.2 Soluzione

In un'architettura multicomputer la logica a n-stadi viene progettata utilizzando degli switch a due ingressi e due uscite in grado di deviare il traffico proveniente da una sorgente verso una destinazione. L'immagine sottostante mostra un esempio di nodi collegati gli uni agli altri attraverso 12 switch differenti.

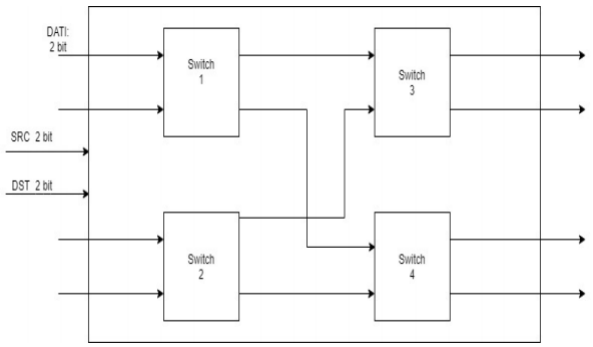


Nel nostro caso avremo 4 nodi codificati con 00, 01, 10, 11. Ognuno di essi può essere sorgente o destinazione di un messaggio di 2 bit che viene inviato attraverso la rete di switch. Si è scelto

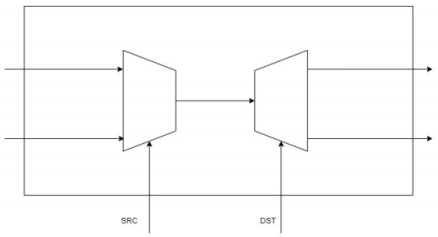
di utilizzare una rete a 4 switch (profondità 2) per la rete da implementare. Il progetto presenta una parte di controllo (PC) e una parte operativa (PO)



La parte di controllo in figura è composta da un multiplexer e un demultiplexer regolati da un logic arbiter, che ha il compito di stabilire la priorità tra i vari nodi. In tal modo all'interno della rete passerà sempre un solo dato alla volta e viene evitato alcun tipo di collisione tra i pacchetti di 2 bit inviati. Il segnale di selezione passato al demultiplexer indica quale dei dati entra nella rete.



La parte operativa è formata dalla rete e prende in ingresso i 2 bit di dati, sorgente e destinazione. Questi ultimi servono allo switch per scegliere una delle due direzioni di uscita.

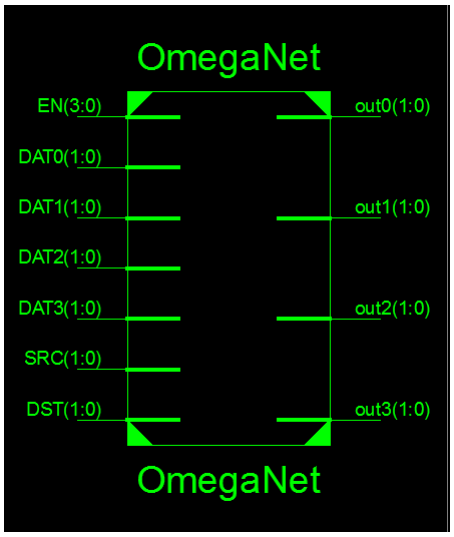


Gli switch sono composti da un multiplexer e un demultiplexer collegati tra loro e i cui segnali di selezione corrispondono al primo o al secondo bit di SRC e DST a seconda del livello a cui si trova il pacchetto. Ad esempio se avessimo SRC = '00' e DST = '01' il multiplexer dello switch selezionerà i dati provenienti dalla prima linea di input (SRC(0) = 0) e il demultiplexer li porrà

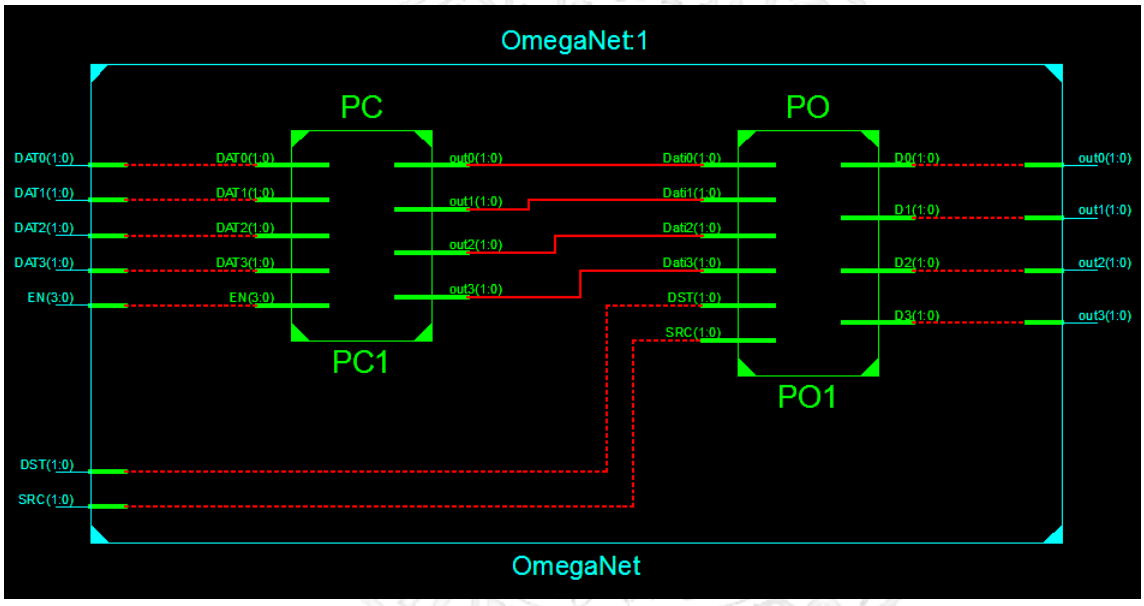
in uscita sulla prima porta di output (DST(1) = 1). Dallo switch1, quindi, passiamo allo switch3 e quest'ultimo selezionerà la seconda uscita (DST(0) = '0') consegnando il dato alla destinazione '01'.

13.2.1 Schematici

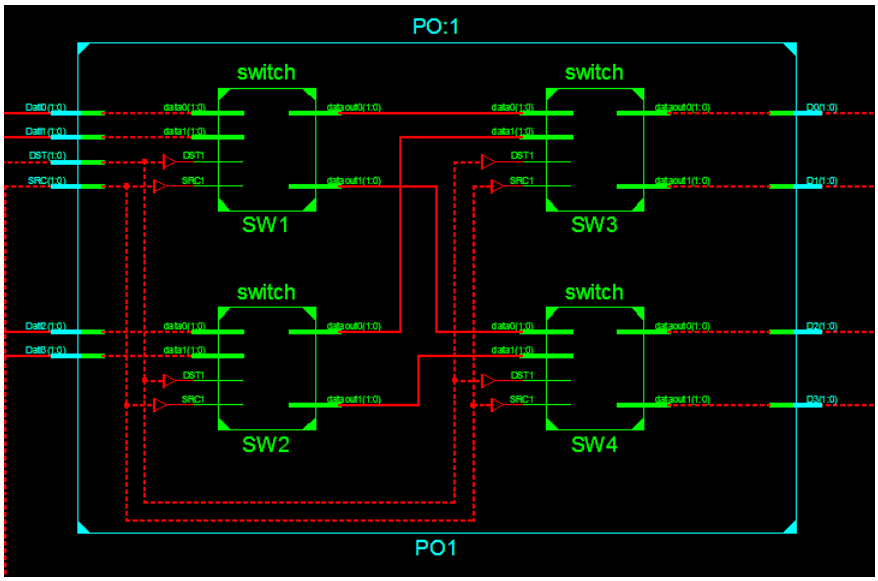
Come descritto, un ingresso enable da 4 bit, 4 linee dati da 2 bit, sorgente e destinazione e 4 linee dati di uscita.



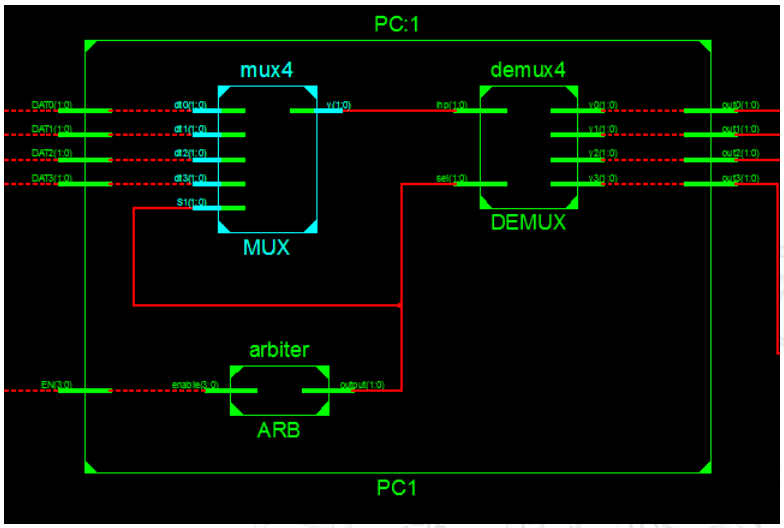
La parte di controllo è direttamente collegata con la parte operativa:



La parte di controllo è costituita da 4 switch per i percorsi dei dati:



Parte di controllo costituita da un mux, un demux e un arbitro:



13.2.2 Codice

Il tutto è stato implementato con una logica di tipo strutturale. Parte operativa e parte di controllo sono racchiuse in OmegaNetwork che prende in ingresso le 4 linee di dati, il segnale di abilitazione EN, sorgente e destinazione. In uscita vi è il dato che attraversa la rete. Sono stati definiti i componenti PC e PO e le uscite del demultiplexer del PC sono collegate agli ingressi dati del PO. Come già detto gli switch sono composti da un multiplexer 2:1 e un demultiplexer 1:2 e i segnali di selezione sno i bit di SRC e DST. Il demultiplexer pone il segnale in ingresso su una delle due uscite in base al valore di SEL.

13.2.2.1 OmegaNetwork

```
1
```

```
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity OmegaNet is
6
7  port(
8
9    EN: in std_logic_vector(3 downto 0);
10   DAT0,DAT1,DAT2,DAT3: in std_logic_vector (1 downto 0);
11   out0,out1,out2,out3: out std_logic_vector (1 downto 0);
12   SRC, DST: in std_logic_vector(1 downto 0)
13
14 );
15
16
17 end OmegaNet;
18
19 architecture Structural of OmegaNet is
20
21 Component PC
22 port(
23
24   EN: in std_logic_vector(3 downto 0);
25   DAT0,DAT1,DAT2,DAT3: in std_logic_vector (1 downto 0);
26   out0,out1,out2,out3: out std_logic_vector (1 downto 0)
27
28 );
29 end component;
30
31
32 Component PO is
33 port(
34     Dati0,Dati1,Dati2,Dati3 : in std_logic_vector(1 downto 0);
35     SRC, DST: in std_logic_vector(1 downto 0);
36     D0,D1,D2,D3 : out std_logic_vector(1 downto 0)
37
38 );
39 end component;
40
41
42 signal t_out0,t_out1,t_out2,t_out3:  std_logic_vector (1 downto 0);
43
44 begin
45
46 PC1: PC port map(
47
48   EN=>EN,
49   DAT0=> DAT0,
50   DAT1=>DAT1,
```

```
51    DAT2=>DAT2,
52    DAT3=>DAT3,
53    out0 => t_out0,
54    out1 => t_out1,
55    out2 => t_out2,
56    out3 => t_out3
57
58  );
59
60  PO1: PO port map(
61
62    Dati0 => t_out0,
63    Dati1 => t_out1,
64    Dati2 => t_out2,
65    Dati3 => t_out3,
66    SRC => SRC,
67    DST => DST,
68    D0 => out0,
69    D1 => out1,
70    D2 => out2,
71    D3 => out3
72
73  );
74
75
76
77  end Structural;
```

Codice Componente 13.1: Definizione del componente OmegaNetwork

13.2.2.2 PO (Parte Operativa)

```
1
2
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6
7
8   entity PO is
9   port(
10      Dati0,Dati1,Dati2,Dati3 : in std_logic_vector(1 downto 0);
11      SRC, DST: in std_logic_vector(1 downto 0);
12      D0,D1,D2,D3 : out std_logic_vector(1 downto 0)
13
14  );
15  end PO;
16
```

```
17 architecture STRUCTURAL of PO is
18
19 Component switch
20 port(
21     data0,data1: in std_logic_vector(1 downto 0);
22     SRC1, DST1: in std_logic;
23     dataout0,dataout1: out std_logic_vector(1 downto 0)
24 );
25 end component;
26
27 signal t0,t1,t2,t3: std_logic_vector(1 downto 0):=(others=>'0');
28
29 begin
30
31
32 SW1: switch port map(
33
34   data0=> Dati0,
35   data1=> Dati1,
36   SRC1=> SRC(0),
37   DST1=> DST(1),
38   dataout0=> t0,
39   dataout1=>t1
40
41 );
42
43 SW2: switch port map(
44
45   data0=> Dati2,
46   data1=> Dati3,
47   SRC1=>  SRC(0),
48   DST1=> DST(1),
49   dataout0=> t2,
50   dataout1=>t3
51
52 );
53
54
55 SW3: switch port map(
56
57   data0=> t0,
58   data1=> t2,
59   SRC1=> SRC(1),
60   DST1=> DST(0),
61   dataout0=> D0,
62   dataout1=>D1
63
64
65 );
```

```
66
67
68 SW4: switch port map(
69
70   data0=> t1,
71   data1=> t3,
72   SRC1=> SRC(1),
73   DST1=> DST(0),
74   dataout0=> D2,
75   dataout1=>D3
76
77 );
78
79 end STRUCTURAL;
```

Codice Componente 13.2: Definizione del componente PO

13.2.2.3 Switch

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity switch is
6  port(
7      data0,data1: in std_logic_vector(1 downto 0);
8      SRC1, DST1: in std_logic;
9      dataout0,dataout1: out std_logic_vector(1 downto 0)
10 );
11 end switch;
12
13 architecture structural of switch is
14
15 Component MUX2
16   PORT (
17        s0 : in  std_logic;
18        a0,b0: in  std_logic_vector(1 downto 0);
19        y : out std_logic_vector(1 downto 0)
20 );
21 end component;
22
23 Component DEMUX2
24   PORT (
25        X : in  std_logic_vector(1 downto 0);
26        s1 : in  std_logic;
27        A1,A2: out  std_logic_vector(1 downto 0)
28        );
29 end component;
```

```
30
31 signal t: std_logic_vector(1 downto 0) :=(others=>'0');
32
33 begin
34
35 MUX_1: mux2 port map(
36   s0=>SRC1,
37   a0=>data0,
38   b0=>data1,
39   y=>t
40
41 );
42
43 DEMUX_1: demux2 port map(
44
45   X=>t,
46   s1=>DST1,
47   A1=>dataout0,
48   A2=>dataout1
49
50 );
51
52 end structural;
```

Codice Componente 13.3: Definizione del componente Switch

13.2.2.4 Demultiplexer 1:2

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity DEMUX2 is
6    PORT (
7        X : in  std_logic_vector(1 downto 0);
8        s1 : in  std_logic;
9        A1,A2: out  std_logic_vector(1 downto 0)
10       );
11 end DEMUX2;
12
13 architecture dataflow of DEMUX2 is
14
15 begin
16
17 A1<= X when s1='0' else "--";
18 A2<= X when s1='1' else "--";
19
20 end dataflow;
```

Codice Componente 13.4: Definizione del componente Demux 1:2

13.2.2.5 PC (Parte di Controllo)

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity PC is
6
7  port(
8    EN: in std_logic_vector(3 downto 0);
9    DAT0,DAT1,DAT2,DAT3: in std_logic_vector (1 downto 0);
10   out0,out1,out2,out3: out std_logic_vector (1 downto 0)
11
12 );
13
14 end PC;
15
16 architecture structural of PC is
17
18 Component arbiter
19 port(
20     enable: in STD_LOGIC_VECTOR(3 downto 0);
21     output: out STD_LOGIC_VECTOR(1 downto 0)
22
23 );
24 end component;
25
26 Component mux4
27 port(
28       S1 : in  std_logic_vector(1 downto 0);
29       dt0,dt1,dt2,dt3 : in  std_logic_vector(1 downto 0);
30       y : out std_logic_vector(1 downto 0)
31 );
32 end component;
33
34
35 Component demux4
36 port(
37   inp, sel: in std_logic_vector(1 downto 0);
38   y0,y1,y2,y3: out std_logic_vector(1 downto 0)
39 );
40 end component;
41
42 signal f1, t1: std_logic_vector(1 downto 0);
43
```

```
44 begin
45
46
47 MUX: mux4 port map(
48
49   S1=>f1,
50   dt0=> DAT0,
51   dt1=>DAT1,
52   dt2=>DAT2,
53   dt3=>DAT3,
54   y=>t1
55
56
57 );
58
59
60 DEMUX: demux4 port map(
61   inp=>t1,
62   sel=>f1,
63   y0=>out0,
64   y1=>out1,
65   y2=>out2,
66   y3=>out3
67
68 );
69
70 ARB: arbiter port map(
71
72   enable=>EN,
73   output=>f1
74
75 );
76
77
78
79 end structural;
```

Codice Componente 13.5: Definizione del componente PC

13.2.2.6 Arbiter

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity arbiter is
6 port(
7     enable: in STD_LOGIC_VECTOR(3 downto 0);
```

```
    output: out STD_LOGIC_VECTOR(1 downto 0));

end arbiter;

architecture dataflow of arbiter is

begin

output <=    "00" when enable(3) = '1' else
       "01" when (enable(3) = '0' and enable(2)='1') else
       "10" when (enable(3) = '0' and enable(2)='0' and enable(1)='1') else
       "11" when (enable(3) = '0' and enable(2)='0' and enable(1)='0' and
          enable(0)='1') else
       "--";

end dataflow;
```

Codice Componente 13.6: Definizione del componente Arbiter

13.2.2.7 Demultiplexer 1:4

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity demux4 is
port(
  inp, sel: in std_logic_vector(1 downto 0);
  y0,y1,y2,y3: out std_logic_vector(1 downto 0)

);

end demux4;

architecture dataflow of demux4 is

begin

y0<= inp when sel="00" else (others=>'-');
y1<= inp when sel="01" else (others=>'-');
y2<= inp when sel="10" else (others=>'-');
y3<= inp when sel="11" else (others=>'-');

end dataflow;
```

Codice Componente 13.7: Definizione del componente Demux 1:4

13.3 Simulazione

I test sono stati effettuati secondo i valori presenti nella tabella sottostante (a sinistra gli input e a destra gli output attesi).

EN	D0	D1	D2	D3	SRC	DST	OUT0	OUT1	OUT2	OUT3
1000	01	11	10	11	00	10	–	–	01	–
1000	11	11	10	11	00	10	–	–	11	–
0100	00	01	10	11	01	00	01	–	–	–
0001	00	01	10	11	11	01	–	11	–	–
1100	11	00	10	01	00	10	–	–	11	–

13.3.1 Codice

Viene mostrato il codice per un singolo caso di test. Per tutti gli altri si modificano i valori nello stim_process.

```
1
2
3  LIBRARY ieee;
4  USE ieee.std_logic_1164.ALL;
5
6  -- Uncomment the following library declaration if using
7  -- arithmetic functions with Signed or Unsigned values
8  --USE ieee.numeric_std.ALL;
9
10 ENTITY test_omega IS
11 END test_omega;
12
13 ARCHITECTURE behavior OF test_omega IS
14
15
16     COMPONENT OmegaNet
17     PORT(
18          EN : IN  std_logic_vector(3 downto 0);
19          DAT0 : IN  std_logic_vector(1 downto 0);
20          DAT1 : IN  std_logic_vector(1 downto 0);
21          DAT2 : IN  std_logic_vector(1 downto 0);
22          DAT3 : IN  std_logic_vector(1 downto 0);
23          out0 : OUT  std_logic_vector(1 downto 0);
24          out1 : OUT  std_logic_vector(1 downto 0);
25          out2 : OUT  std_logic_vector(1 downto 0);
26          out3 : OUT  std_logic_vector(1 downto 0);
27          SRC : IN  std_logic_vector(1 downto 0);
28          DST : IN  std_logic_vector(1 downto 0)
29         );
30     END COMPONENT;
31
32
```

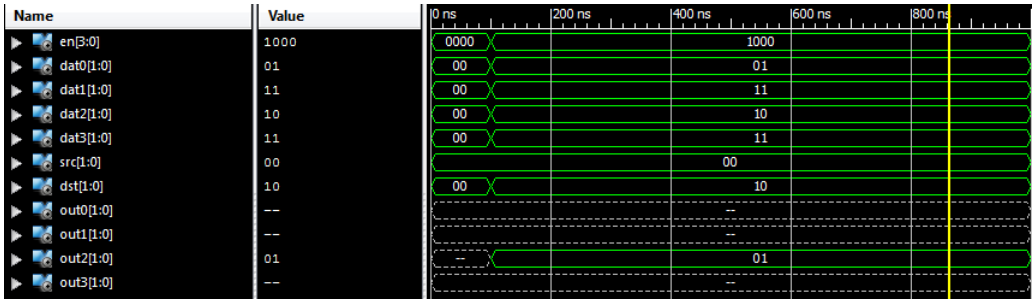
```
33    --Inputs
34    signal EN : std_logic_vector(3 downto 0) := (others => '0');
35    signal DAT0 : std_logic_vector(1 downto 0) := (others => '0');
36    signal DAT1 : std_logic_vector(1 downto 0) := (others => '0');
37    signal DAT2 : std_logic_vector(1 downto 0) := (others => '0');
38    signal DAT3 : std_logic_vector(1 downto 0) := (others => '0');
39    signal SRC : std_logic_vector(1 downto 0) := (others => '0');
40    signal DST : std_logic_vector(1 downto 0) := (others => '0');
41
42   --Outputs
43    signal out0 : std_logic_vector(1 downto 0);
44    signal out1 : std_logic_vector(1 downto 0);
45    signal out2 : std_logic_vector(1 downto 0);
46    signal out3 : std_logic_vector(1 downto 0);
47
48
49  BEGIN
50
51   -- Instantiate the Unit Under Test (UUT)
52    uut: OmegaNet PORT MAP (
53            EN => EN,
54            DAT0 => DAT0,
55            DAT1 => DAT1,
56            DAT2 => DAT2,
57            DAT3 => DAT3,
58            out0 => out0,
59            out1 => out1,
60            out2 => out2,
61            out3 => out3,
62            SRC => SRC,
63            DST => DST
64          );
65
66
67    -- Stimulus process
68    stim_proc: process
69    begin
70         wait for 100 ns;
71
72     EN <= "1000";
73     DAT0 <= "01";
74     DAT1 <= "11";
75     DAT2 <= "10";
76     DAT3 <= "11";
77     SRC <= "00";
78     DST <= "10";
79
80
81       wait;
```

```
82      end process;
83
84  END;
```

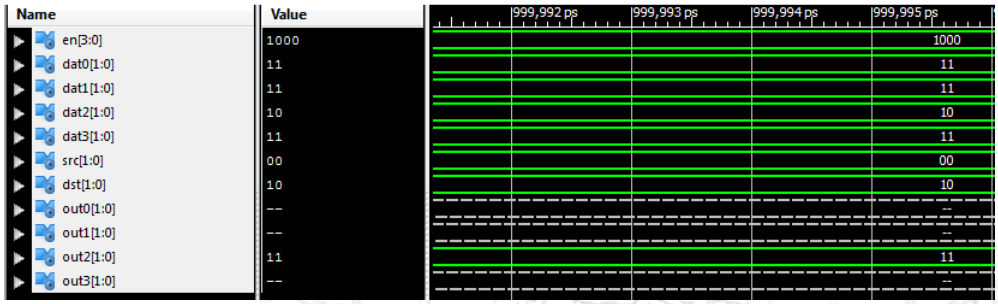
Codice Componente 13.8: Definizione del testbench per OmegaNetwork

13.3.2 Risultati

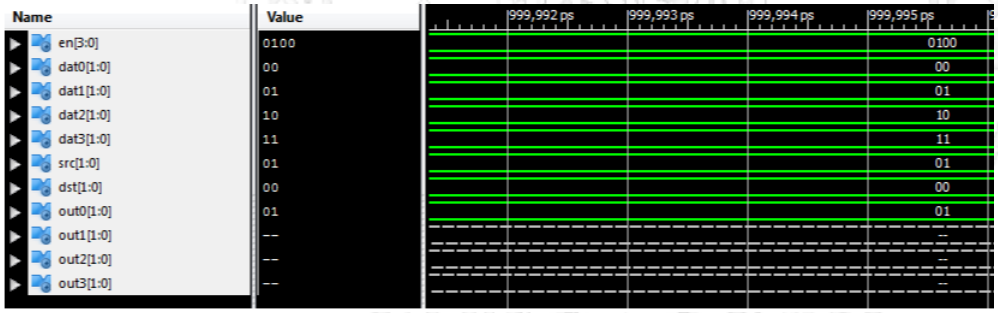
Test 1 : L'output ottenuto corrisponde a quello atteso. L'informazione proveniente dal nodo 1 (00) viene consegnata correttamente al nodo 3 (10).



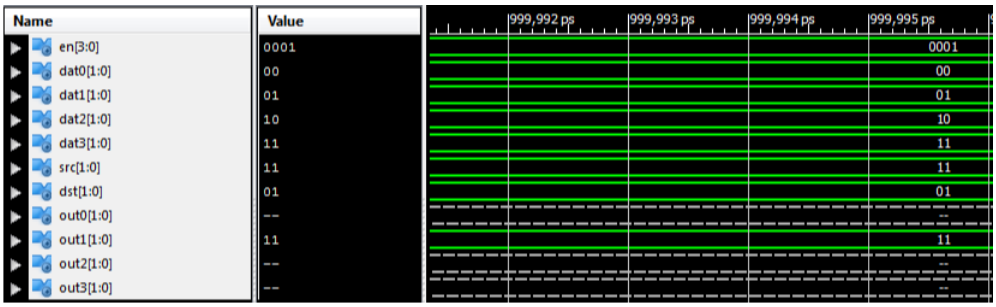
Test 2 : L'output ottenuto corrisponde a quello atteso. L'informazione proveniente dal nodo 1 (00) viene consegnata correttamente al nodo 3 (10).



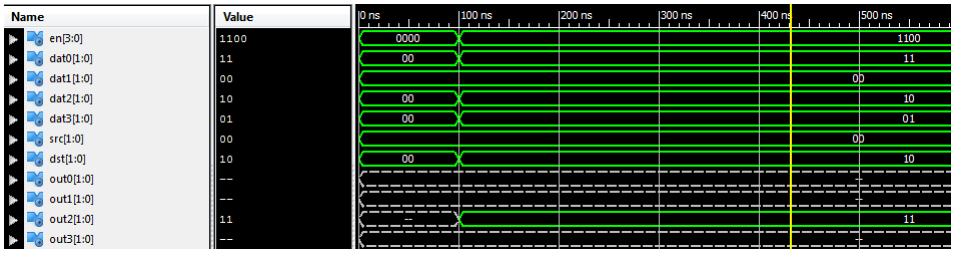
Test 3 : Il dato viene consegnato dal secondo nodo al primo nodo.



Test 4 : Il dato viene consegnato al secondo nodo dal quarto nodo.



Test 5 : La priorità viene rispettata e viene inviato il dato 11 (dat0) alla terza uscita out2.



13.4 Sintesi su board FPGA

Due componenti sono stati aggiunti alla OmegaNetwork per la sintesi su FPGA Nexys2 : il debouncer e il display a 7 segmenti per indicare l'input da inserire. Due process regolano il comportamento della macchina : uno per la macchina a stati finiti e uno per l'inserimento dei bit in input sulla corretta linea dati. Gli stati sono Enable, Source, Dest, Dati e Fine. Tutti questi stati regolano l'immissione dei dati tramite gli switch e il bottone Load in ingresso al debouncer (che ha uscita c1). Nello stato di Enable si inserisce il valore di ingresso dell'arbiter. Successivamente si passa allo stato di Source in cui si inserisce la sorgente del dato e così via con destinazione e dati. Lo stato di Fine permette di visualizzare sui led l'uscita della omega netowork.

```
1
2  entity FPGAOmega is
3  port(
4      clock_50MHz: in std_logic;
5      Load,reset: in std_logic;
6      SW:in std_logic_vector (3 downto 0);
7      LED: out std_logic_vector (7 downto 0):= (others=>'0');
8      Anode_Activate: out STD_LOGIC_VECTOR (3 downto 0);
9      LED_out : out STD_LOGIC_VECTOR (6 downto 0)
10
11 );
12 end FPGAOmega;
13
14 architecture Behavioural of FPGAOmega is
15
16
17 signal t0,t1,t2,t3,s0,s1,s2,s3: std_logic_vector (1 downto 0):= (others
       =>'0');
```

```
18 signal c1: std_logic:='0';
19 signal O1,O2,O3,O4 :std_logic_vector (3 downto 0):= (others=>'0');
20 signal st :std_logic_vector (15 downto 0):= (others=>'0');
21
22
23 type State_type is (En,Source,Dest,Dati,Fine);
24 signal State: State_type:= En;
25
26 Component OmegaNet is
27 port(
28
29   EN: in std_logic_vector(3 downto 0);
30   DAT0,DAT1,DAT2,DAT3: in std_logic_vector (1 downto 0);
31   out0,out1,out2,out3: out std_logic_vector (1 downto 0);
32   SRC, DST: in std_logic_vector(1 downto 0)
33
34 );
35 end Component;
36
37 Component debouncer is
38 port(
39   clock, reset: in std_logic;
40   data_in: in std_logic;
41   data_out: out std_logic
42
43
44 );
45 end Component;
46
47 Component SEV_SEG_DISP is
48 port(
49   clock_50Mhz : in STD_LOGIC;
50    reset : in STD_LOGIC;
51   X: in STD_LOGIC_VECTOR (15 downto 0);
52    Anode_Activate : out STD_LOGIC_VECTOR (3 downto 0);
53    LED_out : out STD_LOGIC_VECTOR (6 downto 0)
54
55
56 );
57
58
59 end Component;
60
61 begin
62
63 O: OmegaNet port map(
64     EN=>O1,
65     DAT0=>t0,
66     DAT1=>t1,
```

```
67      DAT2=>t2,
68      DAT3=>t3,
69      SRC(1)=> O2(1),
70      SRC(0)=>O2(0),
71      DST(1)=>O3(1),
72      DST(0)=>O3(0),
73      out0=>s0,
74      out1=>s1,
75      out2=>s2,
76      out3=>s3
77
78
79  );
80
81
82  DB: debouncer port map(
83      clock=>clock_50MHz,
84      reset=>reset,
85      data_in=>Load,
86      data_out=>c1
87
88
89  );
90
91  DISP:SEV_SEG_DISP port map(
92
93    clock_50MHz=>clock_50MHz,
94    reset=>reset,
95    X=>st,
96    Anode_Activate=>Anode_Activate,
97    LED_out=>LED_out
98
99  );
100
101 p: process(clock_50MHz)
102
103 begin
104
105 if (rising_edge(clock_50MHz)) then
106
107 if (O2="--00") then
108     t0(1)<=O4(1);
109     t0(0)<=O4(0);
110
111 elsif(O2="--01") then
112     t1(1)<=O4(1);
113     t1(0)<=O4(0);
114
115 elsif (O2="--10") then
```

```
116     t2(1)<=O4(1);
117     t2(0)<=O4(0);
118 
119 elsif (O2="--11") then
120     t3(1)<=O4(1);
121     t3(0)<=O4(0);
122 
123 end if;
124 end if;
125 
126 end process;
127 
128 
129 
130 
131 p1:process(clock_50MHz)
132 begin
133 
134 
135 if (reset='1') then
136     State<=En;
137     O1<=(others=>'0');
138     O2<=(others=>'0');
139     O3<=(others=>'0');
140     O4<=(others=>'0');
141     LED<=(others=>'0');
142 
143 elsif (rising_edge(clock_50MHz)) then
144 
145 case State is
146 
147       when En =>
148       --INSERIMENTO ENABLE
149           st<="1110100110101011"; --Sul display viene visualizzata la
                scritta Enable
150         if (c1='1') then
151           O1 <= SW;
152           State<=Source;
153          end if;
154 
155 
156       when Source=>
157       --INSERIMENTO SORGENTE
158         st<="0101000001000110";   --Sul display viene visualizzata la
              scritta SORG
159         if (c1='1') then
160           O2 <= SW;
161           State<=Dest;
162          end if;
```

```
163
164
165         when Dest=>
166         --INSERIMENTO DESTINAZIONE
167         st<="0000111001010011";      --Sul display viene visualizzata la
                scritta Dest
168           if (c1='1') then
169             O3 <= SW;
170             State<=Dati;
171            end if;
172
173
174         when Dati=>
175         --INSERIMENTO DATI
176         st<="0000101000110010";      --Sul display viene visualizzata la
                scritta Dati
177           if (c1='1') then
178             O4 <= SW;
179             State<=Fine;
180           end if;
181
182
183         when Fine=>
184           st<="1111001010011000";    --Sul display viene visualizzata la
                 scritta Fine
185           LED<= s3 & s2 & s1 & s0;
186
187     end case;
188
189 end if;
190
191 end process;
192
193 end Behavioural;
```

Codice Componente 13.9: Definizione del componente FPGAOmega

13.4.1 File UCF

Il file UCF permette di mappare led, display, switch e bottoni per l'immissione degli input e la visualizzazione degli output.

```
1
2 NET "LED_out<6>" LOC = "L18"; # Bank = 1, Pin name = IO_L10P_1, Type = I/O,
      Sch name = CA
3 NET "LED_out<5>" LOC = "F18"; # Bank = 1, Pin name = IO_L19P_1, Type = I/O,
      Sch name = CB
4 NET "LED_out<4>" LOC = "D17"; # Bank = 1, Pin name = IO_L23P_1/HDC, Type =
      DUAL, Sch name = CC
```

```
5  NET "LED_out<3>" LOC = "D16"; # Bank = 1, Pin name = IO_L23N_1/LDC0, Type =
       DUAL, Sch name = CD
6  NET "LED_out<2>" LOC = "G14"; # Bank = 1, Pin name = IO_L20P_1, Type = I/O,
       Sch name = CE
7  NET "LED_out<1>" LOC = "J17"; # Bank = 1, Pin name = IO_L13P_1/A6/RHCLK4/
       IRDY1, Type = RHCLK/DUAL, Sch name = CF
8  NET "LED_out<0>" LOC = "H14"; # Bank = 1, Pin name = IO_L17P_1, Type = I/O,
       Sch name = CG
9  NET "clock_50Mhz"   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type
        = GCLK, Sch name = GCLK0
10 NET "Anode_Activate<0>" LOC = "F17"; # Bank = 1, Pin name = IO_L19N_1, Type
       = I/O, Sch name = AN0
11 NET "Anode_Activate<1>" LOC = "H17"; # Bank = 1, Pin name = IO_L16N_1/A0,
       Type = DUAL, Sch name = AN1
12 NET "Anode_Activate<2>" LOC = "C18"; # Bank = 1, Pin name = IO_L24P_1/LDC1,
       Type = DUAL, Sch name = AN2
13 NET "Anode_Activate<3>" LOC = "F15"; # Bank = 1, Pin name = IO_L21P_1, Type
       = I/O, Sch name = AN3
14
15
16 NET "SW<0>"        LOC = "G18";    # Sch name = SW0
17 NET "SW<1>"        LOC = "H18";    # Sch name = SW1
18 NET "SW<2>"        LOC = "K18";    # Sch name = SW2
19 NET "SW<3>"        LOC = "K17";    # Sch name = SW3
20
21 NET "Load" LOC = "B18";
22 NET "reset"  LOC = "D18";
23
24
25 NET "LED<0>" LOC = "J14";
26 NET "LED<1>" LOC = "J15";
27 NET "LED<2>" LOC = "K15";
28 NET "LED<3>" LOC = "K14";
29 NET "LED<4>" LOC = "E16";
30 NET "LED<5>" LOC = "P16";
31 NET "LED<6>" LOC = "E4";
32 NET "LED<7>" LOC = "P4";
```

Codice Componente 13.10: Definizione del file UCF