# Tesina di Architettura dei Sistemi di Elaborazione
# Gruppo 50

Mario Pace - Mat. M63/0988
Felice Santaniello - Mat. M63/1046
Anna Piscitelli - Mat. M63/1066
Andrea Nuzzolo - Mat. M63/1008

10 ottobre 2020

# Indice

# Capitolo 1

# Esercizio 1

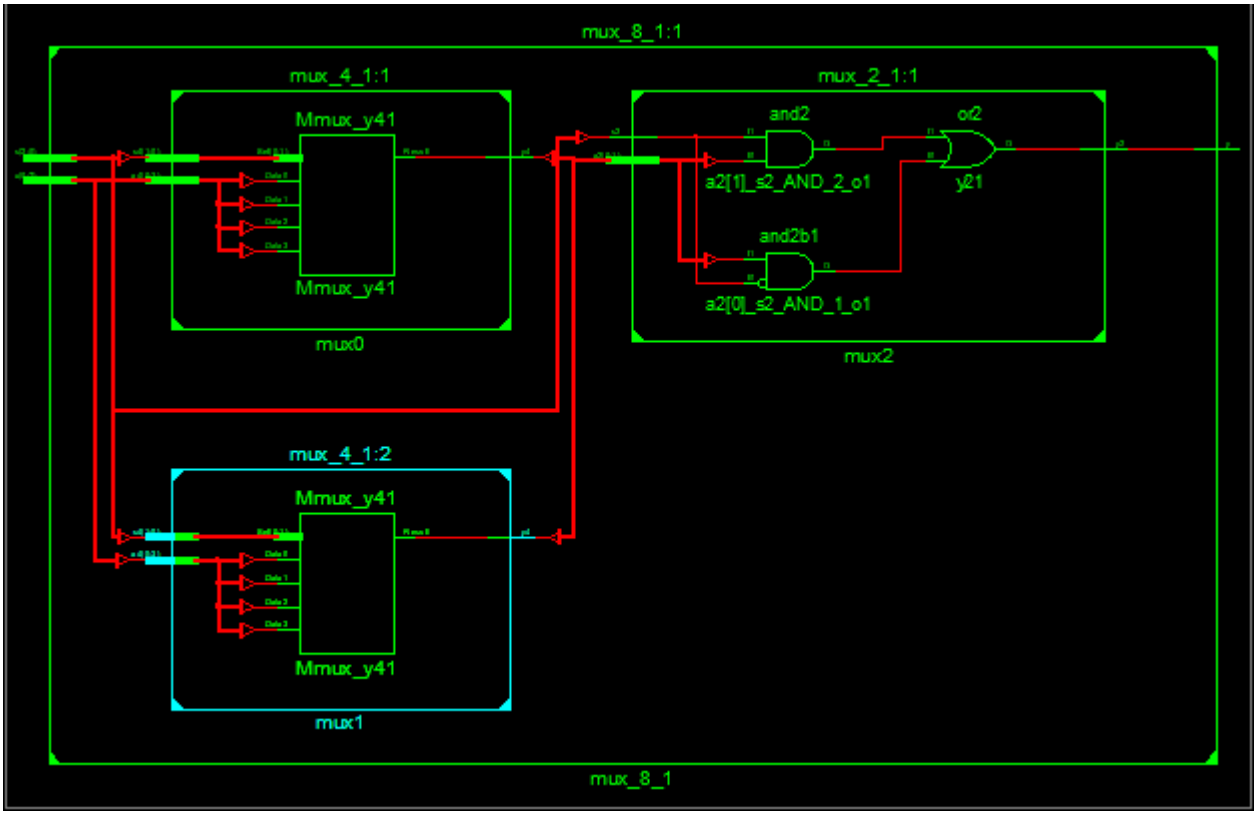
## 1.1 Traccia

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

Per la soluzione si è deciso di utilizzare due multiplexer a 4 ingressi e un multiplexer da 2 ingressi descritti con la tecnica dataflow.

### 1.2.1 Schematici




I due multiplexer 4_1 hanno in ingresso rispettivamente i 4 bit più significativi e i 4 bit meno significativi della stringa di ingresso da 8 bit.

Ragionando come se la stringa di ingresso fosse codificata su 3 bit, i segnali di selezione dei due multiplexer 4_1 valutano i due bit meno significativi, mentre il bit più significativo è valutato dal multiplexer 2_1 che si trova a valle e prende in ingresso le uscite dei due multiplexer a monte.

### 1.2.2 Codice

#### 1.2.2.1 Multiplexer 8_1

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  entity mux_2_1 is
5    Port ( a2 : in  STD_LOGIC_VECTOR(0 to 1);     -- due bit in ingresso, sono
       le uscite dei due mux 4_1
6         s2 : in  STD_LOGIC;                -- segnale di selezione
7             y2 : out  STD_LOGIC);              -- uscita, che nella
                rappresentazione strutturale coincide con l'uscita del mux 8
                _1 complessivo
8  end mux_2_1;
9
10 architecture dataflow of mux_2_1 is
11
```

```
12 begin
13
14 y2 <= ((a2(0) AND (NOT s2)) OR (a2(1) AND s2));  -- descrizione dataflow,
       utilizzo la classica equazione per il mux a due ingressi
15
16 end dataflow;
```

Codice Componente 1.1: Definizione del componente Multiplexer $2_1$

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  entity mux_4_1 is
5      Port ( a4 : in  STD_LOGIC_VECTOR(0 to 3);       -- ingresso
6         s4 : in  STD_LOGIC_VECTOR(1 downto 0);        -- selezione
7             y4 : out  STD_LOGIC);  -- uscita
8  end mux_4_1;
9  architecture dataflow of mux_4_1 is
10 begin
11 y4  <=  a4(0) when s4="00" else
12           a4(1) when s4="01" else
13             a4(2) when s4="10" else
14               a4(3) when s4="11" else       -- in questo caso l'uscita è
                     calcolata tramite un costrutto di assegnazione
                     condizionata
15                   '-';
16 end dataflow;
```

Codice Componente 1.2: Definizione del componente Multiplexer $4_1$

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  entity mux_8_1 is
5    Port ( x : in  STD_LOGIC_VECTOR (0 to 7);       -- stringa di ingresso
         iniziale
6           s : in  STD_LOGIC_VECTOR (2 downto 0);  -- segnale di selezione,
              utilizzo downto e non to perchè più adatto alla natura del
              segnale
7         y : out  STD_LOGIC);   -- uscita finale
8  end mux_8_1;
9  architecture Structural of mux_8_1 is
10 signal u : STD_LOGIC_VECTOR (0 to 1) := (others => '0');
11 -- segnale temporaneo, usato per collegare le uscite dei due mux 4_1 all'
       ingresso del mux 8_1
12 -- definizione dei component per una descrizione strutturale:
13 component mux_2_1
14   Port(
15     a2  : in STD_LOGIC_VECTOR (0 to 1);
16     s2  : in STD_LOGIC;
```

```
17      y2   : out STD_LOGIC      );
18    end component;
19  component mux_4_1
20    Port(
21      a4   : in STD_LOGIC_VECTOR (0 to 3);
22      s4   : in STD_LOGIC_VECTOR (1 downto 0);
23      y4   : out STD_LOGIC      );
24    end component;
25  begin
26  -- port map dei componenti per associare ingressi e uscite della rete
       combinatoria complessiva
27  mux0: mux_4_1
28      Port map(
29        a4 => x(0 to 3),
30        s4=> s(1 downto 0),
31        y4 => u(0)     );
32
33  mux1: mux_4_1
34      Port map(
35        a4 => x(4 to 7),
36        s4 => s(1 downto 0),
37        y4 => u(1)     );
38
39  mux2: mux_2_1
40      Port map(
41        a2 => u(0 to 1),
42        s2 => s(2),
43        y2 => y      );
44
45  end Structural;
```

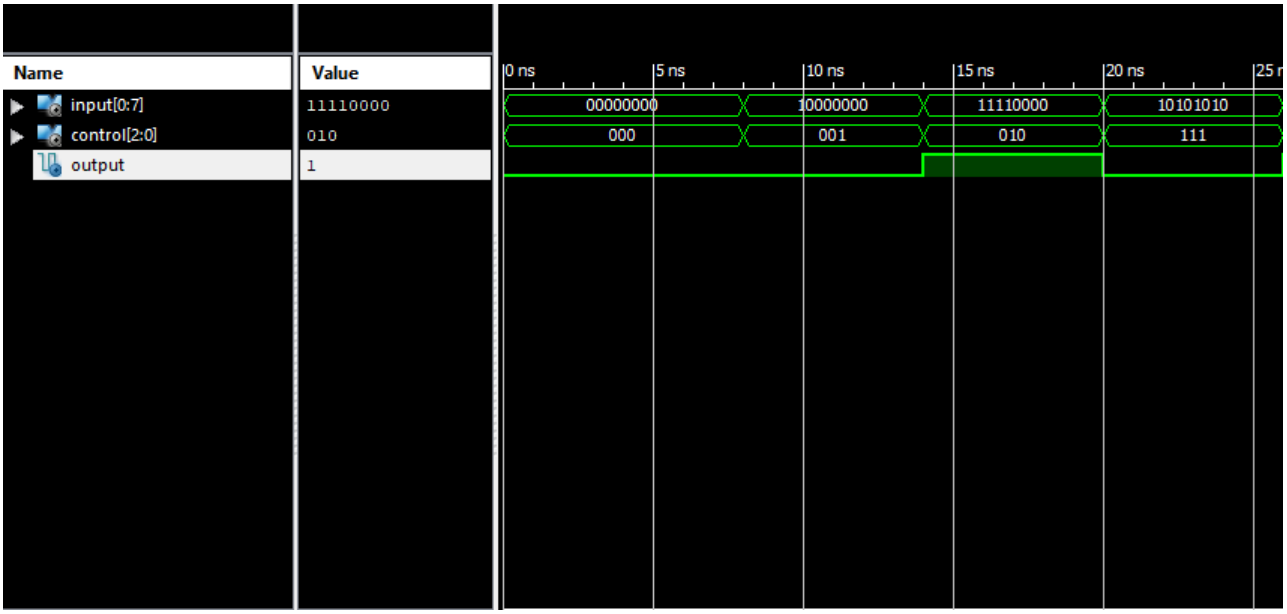
Codice Componente 1.3: Definizione del componente Multiplexer $8_1$

```
1
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.ALL;
4   ENTITY mux_8_1_tb IS
5   END mux_8_1_tb;
6   ARCHITECTURE behavior OF mux_8_1_tb IS
7          COMPONENT mux_8_1
8       PORT(
9         x : IN  std_logic_vector(0 to 7);
10            s : IN  std_logic_vector(2 downto 0);
11            y : OUT  std_logic          );
12      END COMPONENT;
13
14     signal input : std_logic_vector(0 to 7) := (others => '0');
15     signal control : std_logic_vector(2 downto 0) := (others => '0');
16
```

```
17    signal output : std_logic;
18
19  BEGIN
20     uut: mux_8_1 PORT MAP (
21          x => input,
22          s => control,
23          y => output
24     );
25    stim_proc: process
26    begin
27    wait for 8 ns;
28    input <= "10000000";     -- 001 seleziona il secondo bit da sinistra avendo
          definito l'ingresso come (0 to 7) e il segnale di selezione
29    control <= "001" ;        -- come (2 downto 0)
30    wait for 6 ns;
31    assert output = '0'     -- utilizzo un assert per far controllare al
         simulatore integrato in ISE se l'output è corretto
32      report "errore1"
33      severity failure;
34
35      input <= "11110000";
36      control <= "010" ;
37      wait for 6 ns;
38    assert output = '1'
39      report "errore2"
40      severity failure;
41      input <= "10101010";
42        control <= "111" ;
43        wait for 6 ns;
44    assert output = '0'
45      report "errore3"
46      severity failure;
47          wait;
48      end process;
49  END;
```

Codice Componente 1.4: Definizione del componente Multiplexer $8_1$*testbench*

## 1.3 Simulazione




Simulazione del componente finale mux_8_1: notiamo come l'output sia alto in corrispondenza della coppia di ingressi input="11110000" e control="010" perchè viene selezionato il terzo bit da sinistra che è alto, mentre negli altri casi viene selezionato sempre un bit basso.

La simulazione si comporta come desiderato, infatti non compare nessun errore a schermo causato da una violazione dell'assert inserito nel testbench.

## 1.4 Sintesi su board FPGA

Non richiesta dalla traccia dell'esercizio.

# Capitolo 2

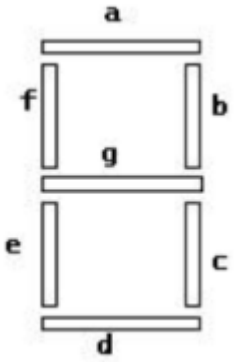
# Esercizio 2

## 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). NOTA: per consentire la visualizzazione di tutte le cifre in maniera univoca è possibile riprodurre le lettere A, C, E ed F in

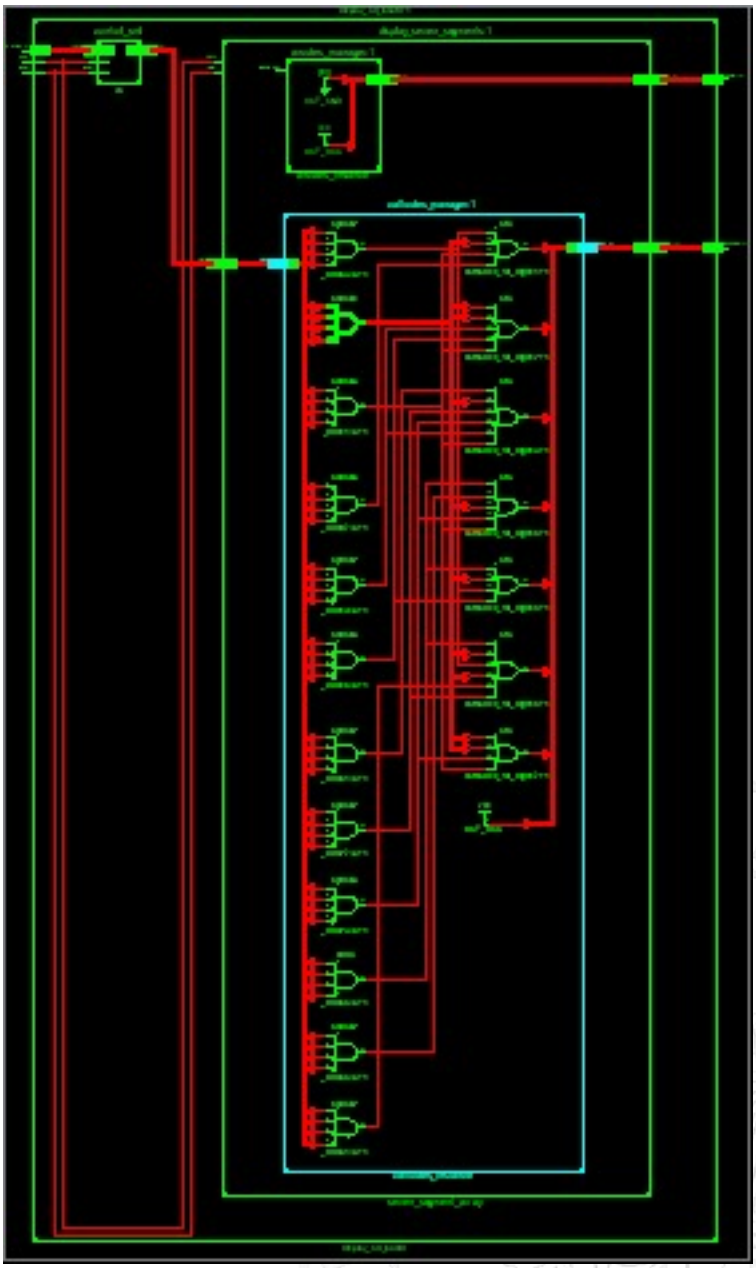
maiuscolo e le lettere B e D in minuscolo. Il controller deve essere sintetizzato sulla board e deve utilizzare gli switch per acquisire l'input e una cifra delle 4 disponibili per visualizzare

l'output.



## 2.2 Soluzione

### 2.2.1 Schematici



Dallo schematico si può notare come la control unit piloti il cathodes_manager per la selezione di quali segmenti del display accendere e quali spegnere a seconda del numero che deve essere mostrato.

L'anodes manager invece non è collegato alla control unit e in questo caso neanche ad un contatore per mostrare più di una cifra contemporaneamente, poichè nella traccia è richiesto l'utilizzo di una sola delle 4 cifre del display a 7 segmenti. Notiamo come l'anodo corrispondente alla prima cifra sia messo a massa (logica negata, quindi attivo), mentre gli anodi corrispondenti alle restanti cifre sono collegati all'alimentazione VDD in modo da tenerli spenti.

## 2.2.2 Codice

### 2.2.2.1 Display 7 segmenti

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity anodes_manager is
5      Port ( enable_digit : in  STD_LOGIC; -- ingresso inserito solo per
           leggibilità del codice, in realtà verrà associato a 1 per attivare
           tutti gli anodi che vi sono collegati.
6             anodes : out  STD_LOGIC_VECTOR (7 downto 0) -- in uscita fornisce
                  le cifre da accendere
7          );
8  end anodes_manager;
9
10 architecture Behavioral of anodes_manager is
11
12 begin
13
14 anodes(0) <= not enable_digit; -- logica negata, quindi nego l'abilitazione
15 anodes ( 7 downto 1) <= (others => '1'); -- tengo spenti gli altri anodi
16
17 end Behavioral;
```

Codice Componente 2.1: Definizione del componente anodes manager

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  use IEEE.NUMERIC_STD.ALL;
5
6  entity cathodes_manager is
7      Port ( value : in  STD_LOGIC_VECTOR (3 downto 0); -- value codificato su
           16 bit, è il valore esadecimale che dovrà essere mostrato in uscita
           sul display
8             cathodes : out  STD_LOGIC_VECTOR (7 downto 0)); -- catodi che
                 dovranno essere attivati per mostrare l'uscita
9  end cathodes_manager;
10
11 architecture Behavioral of cathodes_manager is
12 -- definisco delle costanti corrispondenti ai segmenti da attivare per
      ciascun numero esadecimale da 0 a F
13 constant zero   : std_logic_vector(6 downto 0) := "1000000";
14 constant one    : std_logic_vector(6 downto 0) := "1111001";
15 constant two    : std_logic_vector(6 downto 0) := "0100100";
16 constant three  : std_logic_vector(6 downto 0) := "0110000";
17 constant four   : std_logic_vector(6 downto 0) := "0011001";
18 constant five   : std_logic_vector(6 downto 0) := "0010010";
19 constant six    : std_logic_vector(6 downto 0) := "0000010";
```

```
20 constant seven  : std_logic_vector(6 downto 0) := "1111000";
21 constant eight  : std_logic_vector(6 downto 0) := "0000000";
22 constant nine   : std_logic_vector(6 downto 0) := "0010000";
23 constant a     : std_logic_vector(6 downto 0) := "0001000";
24 constant b      : std_logic_vector(6 downto 0) := "0000011";
25 constant c      : std_logic_vector(6 downto 0) := "1000110";
26 constant d      : std_logic_vector(6 downto 0) := "0100001";
27 constant e      : std_logic_vector(6 downto 0) := "0000110";
28 constant f      : std_logic_vector(6 downto 0) := "0001110";
29
30
31 signal cathodes_for_digit : std_logic_Vector(6 downto 0) := (others => '0');
       -- segnale temporaneo che verrà utilizzato per il calcolo dell'uscita
   effettiva
32
33 begin
34
35 seven_segment_decoder_process: process(value) -- il process ha nella
   sensitivity list soltanto l'ingresso value, pochè deve modificare l'
   uscita soltanto al variare dell'ingresso
36   begin
37     case value is  -- costrutto case per assegnare a cathodes_for_digit i
         giusti segmenti da attivare a seconda dell'ingresso
38       when "0000" => cathodes_for_digit <= zero;
39       when "0001" => cathodes_for_digit <= one;
40       when "0010" => cathodes_for_digit <= two;
41       when "0011" => cathodes_for_digit <= three;
42       when "0100" => cathodes_for_digit <= four;
43       when "0101" => cathodes_for_digit <= five;
44       when "0110" => cathodes_for_digit <= six;
45       when "0111" => cathodes_for_digit <= seven;
46       when "1000" => cathodes_for_digit <= eight;
47       when "1001" => cathodes_for_digit <= nine;
48       when "1010" => cathodes_for_digit <= a;
49       when "1011" => cathodes_for_digit <= b;
50       when "1100" => cathodes_for_digit <= c;
51       when "1101" => cathodes_for_digit <= d;
52       when "1110" => cathodes_for_digit <= e;
53       when "1111" => cathodes_for_digit <= f;
54     when others => cathodes_for_digit <= (others => '0');
55     end case;
56   end process seven_segment_decoder_process;
57
58 cathodes (6 downto 0) <= cathodes_for_digit; -- associo ai primi 7 catodi l'
   uscita corrispondente
59 cathodes(7) <= '1'; -- non utilizzo i dots, quindi li tengo inattivi
60
61 end Behavioral;
```

Codice Componente 2.2: Definizione del componente cathodes manager

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

entity control_unit is
    Port ( clock : in  STD_LOGIC;
           reset : in  STD_LOGIC;
       in_byte : in STD_LOGIC_VECTOR(3 downto 0); -- valore in ingresso che verrà poi associato agli switch della board
           value : out  STD_LOGIC_VECTOR(3 downto 0)           -- valore passato al cathodes_manager
         );

end control_unit;

architecture Behavioral of control_unit is

signal reg_value : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');

begin

value <= reg_value;

main: process(clock, reset) -- process con reset asincrono, quindi presente nella sensitivity list

begin

  if reset = '1' then
    reg_value <= (others => '0'); -- in caso di reset il valore di uscita è 0
  elsif clock'event and clock = '1' then
    reg_value <= in_byte(3 downto 0); -- sul fronte di salita del clock aggiorno il valore dell'uscita con quello dell'ingresso fornito
    end if;

end process;

end Behavioral;
```

Codice Componente 2.3: Definizione del componente control unit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- componente che ha il compito di connettere control unit e display_seven_segment per far si che il valore di uscita sia pilotato correttamente
```

```
4  entity display_on_board is
5    Port(
6        clock : in  STD_LOGIC;
7        reset : in  STD_LOGIC;
8        in_byte : in STD_LOGIC_VECTOR(3 downto 0);
9        anodes : out  STD_LOGIC_VECTOR (7 downto 0);
10       cathodes : out  STD_LOGIC_VECTOR (7 downto 0)
11         );
12
13 end display_on_board;
14
15 architecture Structural of display_on_board is
16
17 COMPONENT display_seven_segments
18   PORT(
19     clock : IN std_logic;
20     reset : IN std_logic;
21     value : IN std_logic_vector(3 downto 0);--4 nibble da mostrare
22     anodes : OUT std_logic_vector(7 downto 0);
23     cathodes : OUT std_logic_vector(7 downto 0)
24     );
25 END COMPONENT;
26
27 COMPONENT control_unit
28   PORT(
29     clock : IN std_logic;
30     reset : IN std_logic;
31     in_byte : IN std_logic_vector(3 downto 0);
32     value : OUT std_logic_vector(3 downto 0)
33     );
34 END COMPONENT;
35
36 signal cu_value : std_logic_vector(3 downto 0); -- non posso leggere da una
     variabile di uscita, quindi utilizzo un segnale temporaneo per la
     connessione
37
38 begin
39
40 seven_segment_array: display_seven_segments
41   PORT MAP(
42     clock => clock,
43     reset => reset,
44     value => cu_value,    -- associo il value uscita della control_unit al
         value ingresso del display
45     anodes => anodes,
46     cathodes => cathodes
47 );
48
49 cu: control_unit PORT MAP(
```

```
50      clock => clock,
51      reset => reset,
52      in_byte => in_byte,
53      value => cu_value
54
55    );
56
57  end Structural;
```

Codice Componente 2.4: Definizione del componente display on board

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   -- componente che connette i due gestori di anodi e catodi
4   entity display_seven_segments is
5      Port ( clock : in  STD_LOGIC;
6             reset : in  STD_LOGIC;
7             value : in  STD_LOGIC_VECTOR (3 downto 0);
8             anodes : out  STD_LOGIC_VECTOR (7 downto 0);
9             cathodes : out  STD_LOGIC_VECTOR (7 downto 0));
10  end display_seven_segments;
11
12  architecture Structural of display_seven_segments is
13
14  COMPONENT cathodes_manager
15    PORT(
16      value : IN std_logic_vector(3 downto 0);
17      cathodes : OUT std_logic_vector(7 downto 0)
18      );
19  END COMPONENT;
20
21  COMPONENT anodes_manager
22    PORT(
23      enable_digit : IN std_logic;
24      anodes : OUT std_logic_vector(7 downto 0)
25      );
26  END COMPONENT;
27
28
29  begin
30
31  cathodes_instance: cathodes_manager port map(
32    value => value,
33    cathodes => cathodes
34  );
35
36  anodes_instance: anodes_manager port map(
37    enable_digit => '1', -- abilitazione sempre presente, nella soluzione
        fornita non c'è bisogno di un segnale di enable, l'uscita varia su
```

```
      ogni fronte di salita del clock se cambio l'ingresso
  anodes => anodes
);

end Structural;
```

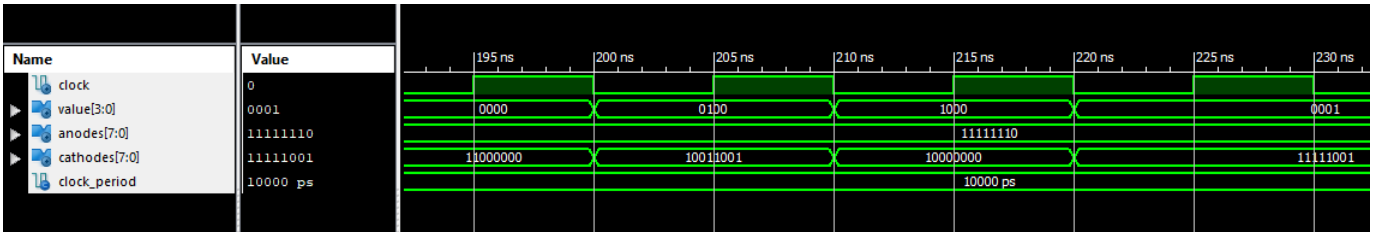
Codice Componente 2.5: Definizione del componente display seven segments

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

ENTITY tb_display IS
END tb_display;

ARCHITECTURE behavior OF tb_display IS

    COMPONENT display_seven_segments
    PORT(
         clock : IN  std_logic;

         value : IN  std_logic_vector(3 downto 0);

         anodes : OUT  std_logic_vector(7 downto 0);
         cathodes : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clock : std_logic := '0';

   signal value : std_logic_vector(3 downto 0) := (others => '0');


  --Outputs
   signal anodes : std_logic_vector(7 downto 0);
   signal cathodes : std_logic_vector(7 downto 0);

   -- Clock period definitions
   constant clock_period : time := 10 ns; -- periodo di clock per la
      simulazione

BEGIN

  -- Instantiate the Unit Under Test (UUT)
   uut: display_seven_segments PORT MAP (
          clock => clock,

          value => value,
```

```
41
42          anodes => anodes,
43          cathodes => cathodes
44        );
45
46    -- Clock process definitions
47    clock_process :process  -- process per la definizione del clock per la
          simulazione di periodo clock_period costante inizializzato
          precedentemente
48    begin
49     clock <= '0';
50     wait for clock_period/2;
51     clock <= '1';
52     wait for clock_period/2;
53    end process;
54
55
56    -- Stimulus process
57    stim_proc: process
58    begin
59       -- hold reset state for 100 ns.
60       wait for 100 ns;
61
62       wait for clock_period*10;
63
64     -- inserisco vari valori per testare se le stringhe di uscita di anodi e
          catodi corrispondono alla rappresentazione dei numeri inseriti in
          ingresso
65     value <= "0100";
66
67     wait for 10 ns;
68
69     value <= "1000";
70
71     wait for 10 ns;
72
73     value <= "0001";
74
75     wait for 10 ns;
76
77
78
79
80       wait;
81    end process;
82
83  END;
```

Codice Componente 2.6: Definizione del componente $\text{tb}_{display}$

## 2.3 Simulazione




Dalla simulazione notiamo come sui fronti di discesa del clock, al variare dell'ingresso value, l'uscita cathodes assume la combinazione di bit necessaria per l'illuminazione dei giusti segmenti del display.

L'uscita anodes rimane costante perchè non è presente nessuna abilitazione, e corrisponde all'attivazione solo della cifra meno significativa del display (messa a 0), mentre le altre sono lasciate spente.

## 2.4 Sintesi su board FPGA

Per interfacciare il design realizzato con l'I/O di board sono state utilizzate le seguenti locazioni:

Per il clock si è utilizzato semplicemente il clock della scheda, senza l'aggiunta di eventuali filtri per abbassare la frequenza perchè non necessari.

```
## Clock signal
NET "clock"    LOC = "E3"   | IOSTANDARD = "LVCMOS33";
```

Per l'ingresso sono stati utilizzati i 4 switch più a destra nella board.

```
NET "in_byte<0>"          LOC=J15 | IOSTANDARD=LVCMOS33;
NET "in_byte<1>"          LOC=L16 | IOSTANDARD=LVCMOS33;
NET "in_byte<2>"          LOC=M13 | IOSTANDARD=LVCMOS33;
NET "in_byte<3>"          LOC=R15 | IOSTANDARD=LVCMOS33;
```

Per il display sono stati utilizzati tutti gli anodi e i catodi a disposizione: questa soluzione potrebbe sembrare insensata, poichè bastava soltando un anodo per illuminare la cifra meno significativa del display della board, ma se non vengono disattivati esplicitamente gli anodi non utilizzati collegandoli all'alimentazione (logica negata) questi assumono valori casuali poichè non inizializzati e la soluzione è meno chiara.

```
## 7 segment display

NET "cathodes<0>"              LOC=T10 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_A00_D16_14
NET "cathodes<1>"              LOC=R10 | IOSTANDARD=LVCMOS33; #IO_25_14
NET "cathodes<2>"              LOC=K16 | IOSTANDARD=LVCMOS33; #IO_25_15
NET "cathodes<3>"              LOC=K13 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A26_15
NET "cathodes<4>"              LOC=P15 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_14
NET "cathodes<5>"              LOC=T11 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A10_D26_14
NET "cathodes<6>"              LOC=L18 | IOSTANDARD=LVCMOS33; #IO_L4P_T0_D04_14
NET "cathodes<7>"              LOC=H15 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_A21_VREF_15

NET "anodes<0>"          LOC=J17 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_FOE_B_15
NET "anodes<1>"          LOC=J18 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_FWE_B_15
NET "anodes<2>"          LOC=T9 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_A01_D17_14
NET "anodes<3>"          LOC=J14 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A22_15
NET "anodes<4>"          LOC=P14 | IOSTANDARD=LVCMOS33; #IO_L8N_T1_D12_14
NET "anodes<5>"          LOC=T14 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_14
NET "anodes<6>"          LOC=K2 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_35
NET "anodes<7>"          LOC=U13 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_A02_D18_14
```

# 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 Soluzione

Secondo la soluzione adottata da Sami, un flip-flop D edge triggered attivo sul fronte di discesa può essere realizzato mediante un latch D il cui segnale di clock è idealmente impulsivo. Per rendere il clock impulsivo si è posto tale segnale in ingresso ad un blocco derivatore. È stato quindi realizzato un flip-flop D edge-triggered attivo sul fronte di discesa, seguendo un'architettura multi-livello: il primo livello è costituito dal Latch_RS; il secondo dal Latch_D e dal Derivatore; il terzo ed ultimo, ETd_Sami, costituisce il top-level realizzato per composizione dei precedenti componenti.

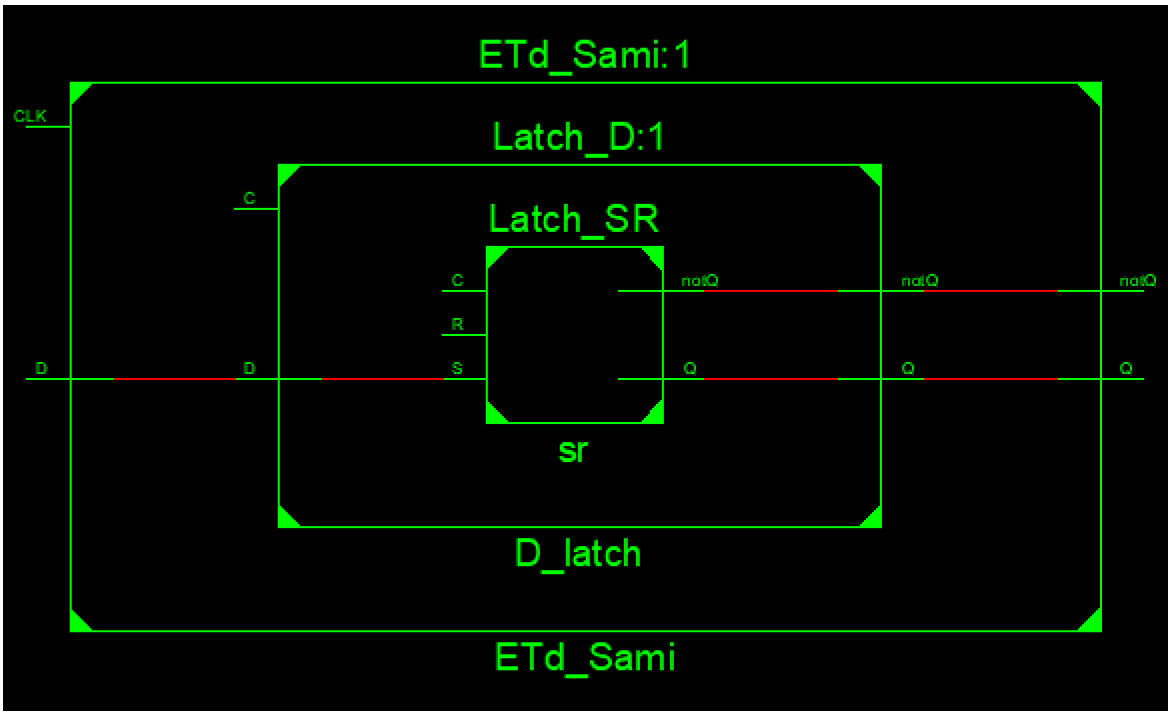
Secondo la soluzione adottata da Mazzeo un flip-flop D edge-triggered attivo sul fronte di discesa si ottiene per composizione di 3 latch RS non abilitati realizzati con porte NOR.

Secondo la soluzione adottata da Mazzeo e Sami, un flip-flop D master-slave viene realizzato utilizzando due latch in cascata, il cui segnale di sincronismo è in contrapposizione di fase. Entrambe le soluzioni realizzano un master-slave non ideale.

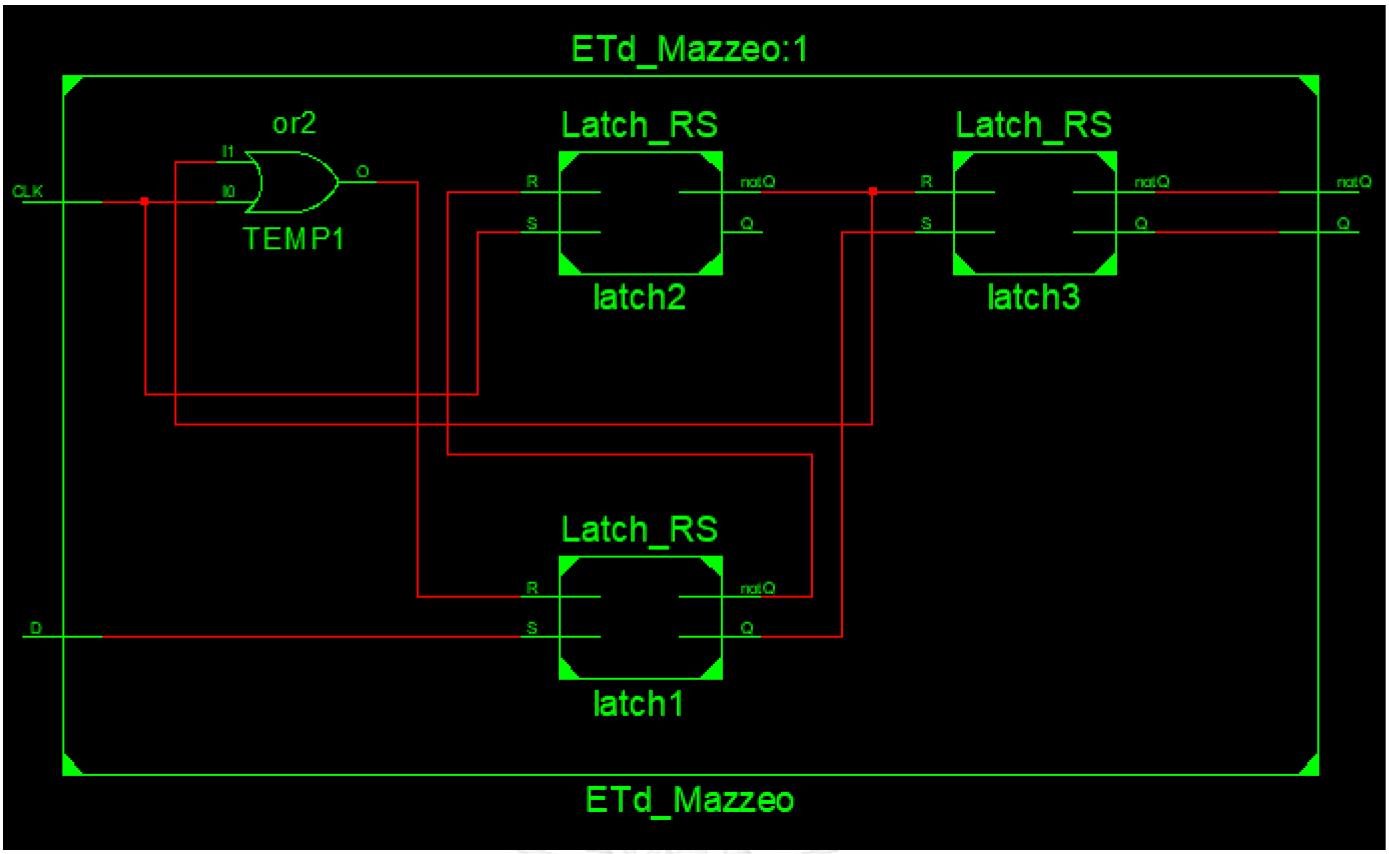
Infine si è riportato anche una soluzione che realizza un flip flop master-slave ideale che utilizza in cascata un flip flop edge triggered attivo sul fronte di salita ed un flip flop edge triggered attivo sul fronte di discesa.

### 3.2.1 Schematici

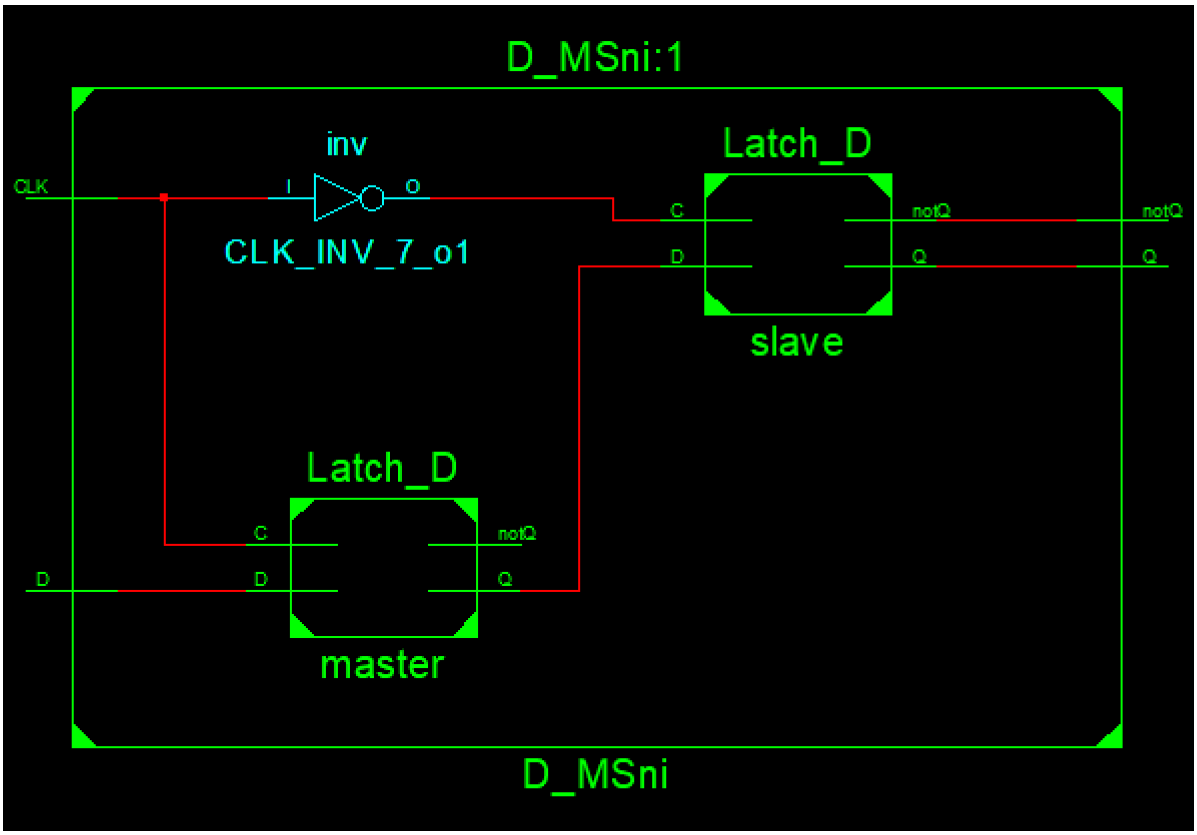
- Schematico del flip flop D edge triggered secondo il modello di Sami:



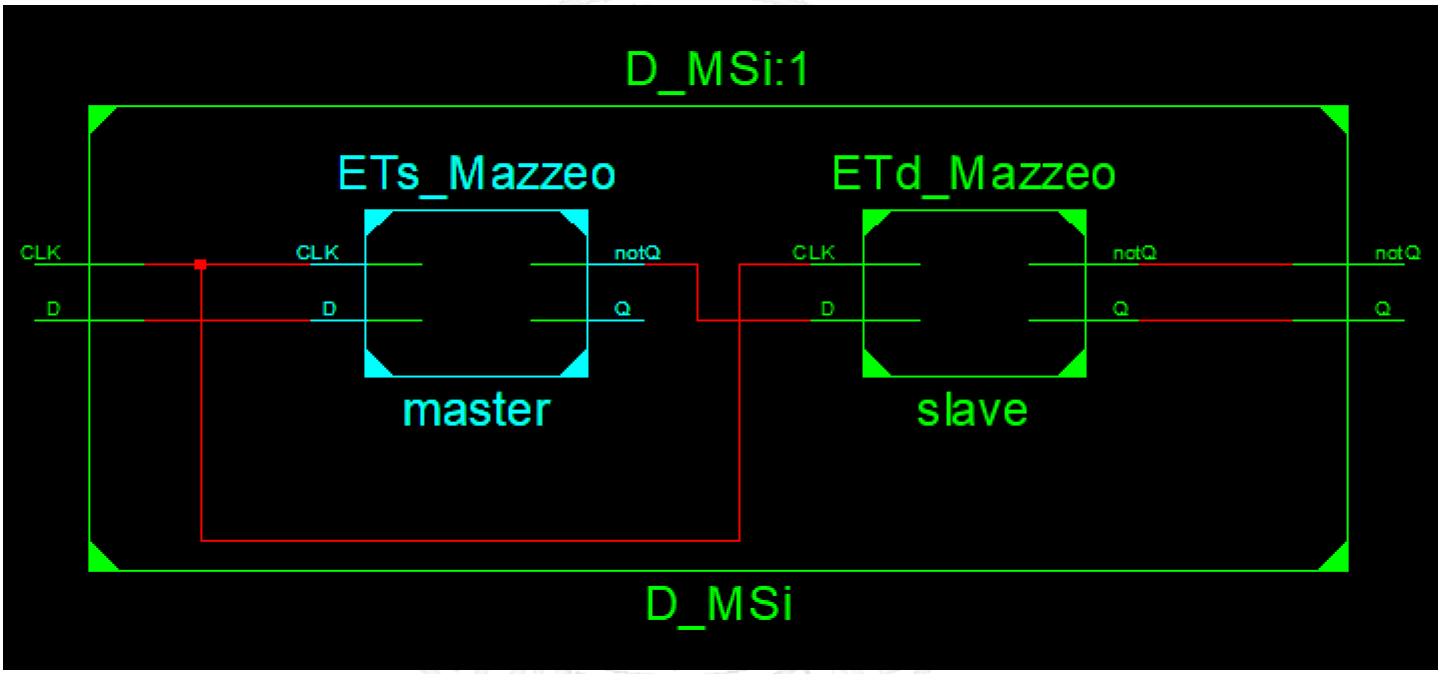

- Schematico del flip flop D edge triggered secondo il modello di Mazzeo:



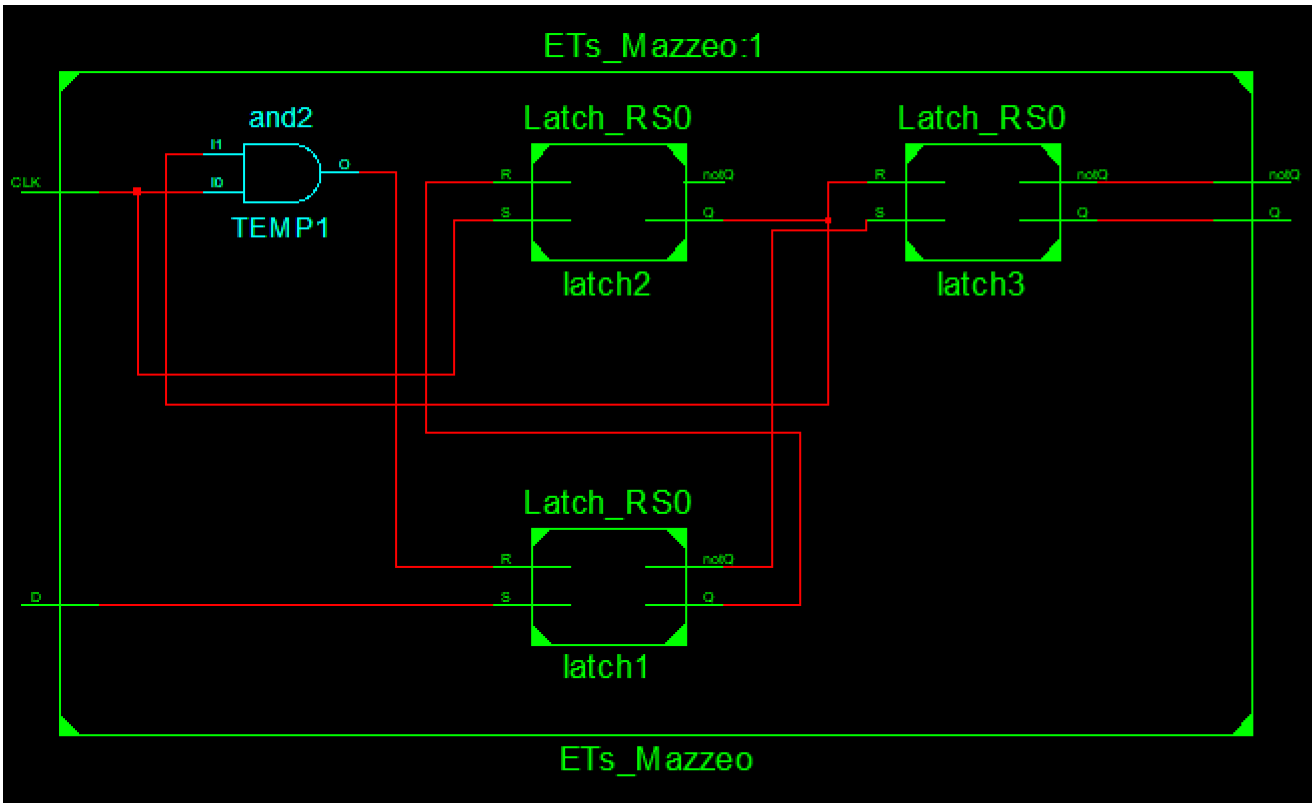

- Schematico del flip flop master slave (non ideale) secondo il modello di Sami e Mazzeo:




- Schematico del flip flop master slave (ideale):




- Schematico del flip flop D edge triggered attivo sul fronte di salita secondo il modello di Mazzeo:




### 3.2.2 Codice

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Latch_SR is
5      Port ( S : in  STD_LOGIC;
6             R : in  STD_LOGIC;
7             C : in  STD_LOGIC;  -- Abilitazione
8             Q : out  STD_LOGIC;
9             notQ : out  STD_LOGIC);
10 end Latch_SR;
11
12 architecture Behavioral of Latch_SR is
13
14 signal TEMP : STD_LOGIC;
15
16 begin
17
18 latch: process(R, S, C)
19 begin
20
21 if(C = '1') then
22   if(S = '1' AND R = '0') then
23     TEMP <= '1';
```

```
  elsif(S = '0' AND R = '1') then
    TEMP <= '0';
  elsif(S = '0' AND R = '0') then
    null;
  elsif(S = '1' AND R = '1') then
    TEMP <= 'X';
  end if;
end if;

end process;

Q <= TEMP;
notQ <= NOT TEMP;

end Behavioral;
```

Codice Componente 3.1: Definizione del componente Latch$_SR$

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

entity Latch_D is
    Port ( D : in  STD_LOGIC;
           C : in  STD_LOGIC;
           Q : out  STD_LOGIC;
           notQ : out  STD_LOGIC);
end Latch_D;

architecture Structural of Latch_D is

component Latch_SR
    Port ( S : in  STD_LOGIC;
           R : in  STD_LOGIC;
           C : in  STD_LOGIC;
           Q : out  STD_LOGIC;
           notQ : out  STD_LOGIC);
end component;

signal notD : STD_LOGIC;

begin

notD <= not D;

sr : Latch_SR
Port Map(  S => D,
           R => notD,
           C => C,
           Q => Q,
```

```
32              notQ => notQ);
33
34  end Structural;
```

Codice Componente 3.2: Definizione del componente Latch$_D$

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity Derivatore is
5   generic( delay : TIME := 1 ns);
6   port( C : in STD_LOGIC;
7       Controllo : out STD_LOGIC);
8   end Derivatore;
9
10  architecture dataflow of Derivatore is
11
12  signal B : STD_LOGIC;
13
14  begin
15
16  B <= NOT C after delay;
17  Controllo <= NOT (C OR B);
18
19  end dataflow;
```

Codice Componente 3.3: Definizione del componente Derivatore

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3
4   entity ETd_Sami is
5   port( D : in STD_LOGIC;
6       CLK : in STD_LOGIC;
7       Q : out STD_LOGIC;
8       notQ : out STD_LOGIC);
9   end ETd_Sami;
10
11  architecture Structural of ETd_Sami is
12
13  component Latch_D
14  port(   D : in STD_LOGIC;
15      C : in STD_LOGIC;
16      Q : out STD_LOGIC;
17      notQ : out STD_LOGIC);
18  end component;
19
20  component Derivatore
21  port( C : in STD_LOGIC;
22      Controllo : out STD_LOGIC);
```

```
23 end component;
24
25 signal TEMP : STD_LOGIC;
26
27 begin
28
29 D_latch : Latch_D
30 port map( D => D,
31       C => TEMP,
32      Q => Q,
33      notQ => notQ);
34
35 Der : Derivatore
36 port map( C => CLK,
37       Controllo => TEMP);
38
39 end Structural;
```

Codice Componente 3.4: Definizione del componente ETd$_{S}ami$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY ETd_Sami_tb IS
5  END ETd_Sami_tb;
6
7  ARCHITECTURE behavior OF ETd_Sami_tb IS
8
9      -- Component Declaration for the Unit Under Test (UUT)
10
11     COMPONENT ETd_Sami
12     PORT(
13          D : IN  std_logic;
14          CLK : IN  std_logic;
15          Q : OUT  std_logic;
16          notQ : OUT  std_logic
17         );
18     END COMPONENT;
19
20
21    --Inputs
22    signal D : std_logic := '0';
23    signal CLK : std_logic := '0';
24
25   --Outputs
26    signal Q : std_logic;
27    signal notQ : std_logic;
28
29    -- Clock period definitions
```

```
30    constant CLK_period : time := 10 ns;
31
32  BEGIN
33
34    -- Instantiate the Unit Under Test (UUT)
35    uut: ETd_Sami PORT MAP (
36            D => D,
37            CLK => CLK,
38            Q => Q,
39            notQ => notQ
40          );
41
42    -- Clock process definitions
43    CLK_process :process
44    begin
45     CLK <= '0';
46     wait for CLK_period/2;
47     CLK <= '1';
48     wait for CLK_period/2;
49    end process;
50
51
52    -- Stimulus process
53    stim_proc: process
54    begin
55
56     wait for CLK_period/4;
57
58     D <= '1';
59     wait until(CLK'EVENT AND CLK = '0');
60
61     wait for 1 ns;
62     assert Q = '1' AND notQ = '0'
63     report "errore0"
64     severity failure;
65
66     D <= '0';
67     wait until(CLK'EVENT AND CLK = '0');
68
69     wait for 1 ns;
70
71     assert Q = '0' AND notQ = '1'
72     report "errore1"
73     severity failure;
74
75     D <= '1';
76     wait until(CLK'EVENT AND CLK = '0');
77
78     wait for 0.5 ns;  -- Attesa inferiore ad 1 ns dopo il fronte di discesa
```

```
        di CLK
    assert Q = '1' AND notQ = '0'
    report "errore2"
    severity failure;

    D <= '0';
    wait until(CLK'EVENT AND CLK = '0');

    wait for 1 ns;
    assert Q = '0' AND notQ = '1' -- Errore dovuto alla non idealita'
    report "errore2"
    severity failure;

      wait;
   end process;

END;
```

Codice Componente 3.5: Definizione del componente ETd$_S ami_t b$

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

entity Latch_RS is
    Port ( R : in  STD_LOGIC;
           S : in  STD_LOGIC;
           Q : out  STD_LOGIC;
           notQ : out  STD_LOGIC);
end Latch_RS;

architecture Behavioral of Latch_RS is

signal TEMP : STD_LOGIC;

begin

latch: process(R, S)
begin

  if(S = '1' AND R = '0') then
    Q <= '1';
    notQ <= '0';
  elsif(S = '0' AND R = '1') then
    Q <= '0';
    notQ <= '1';
  elsif(S = '0' AND R = '0') then
    null;
  elsif(S = '1' AND R = '1') then
    Q <= '0';
```

```
30      notQ <= '0';
31    end if;
32
33  end process;
34
35  end Behavioral;
```

Codice Componente 3.6: Definizione del componente Latch$_R S$

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity ETd_Mazzeo is
5       Port ( D : in  STD_LOGIC;
6              CLK : in  STD_LOGIC;
7              Q : out  STD_LOGIC;
8              notQ : out  STD_LOGIC);
9   end ETd_Mazzeo;
10
11  architecture Structural of ETd_Mazzeo is
12
13  component Latch_RS
14      Port ( R : in  STD_LOGIC;
15             S : in  STD_LOGIC;
16             Q : out  STD_LOGIC;
17             notQ : out  STD_LOGIC);
18  end component;
19
20  signal TEMP : STD_LOGIC;
21  signal R3 : STD_LOGIC;
22  signal S3 : STD_LOGIC;
23  signal RIS1 : STD_LOGIC;
24
25  begin
26
27  TEMP <= CLK or R3;
28
29  latch1 : Latch_RS
30  port map( R => TEMP,
31        S => D,
32       Q => S3,
33       notQ => RIS1);
34
35  latch2 : Latch_RS
36  port map( R => RIS1,
37        S => CLK,
38       notQ => R3);
39
40  latch3 : Latch_RS
```

```
41 port map( R => R3,
42      S => S3,
43      Q => Q,
44      notQ => notQ);
45
46
47 end Structural;
```

Codice Componente 3.7: Definizione del componente ETd$_{M}azzeo$

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY ETd_Mazzeo_tb IS
5  END ETd_Mazzeo_tb;
6
7  ARCHITECTURE behavior OF ETd_Mazzeo_tb IS
8
9      -- Component Declaration for the Unit Under Test (UUT)
10
11     COMPONENT ETd_Mazzeo
12     PORT(
13          D : IN  std_logic;
14          CLK : IN  std_logic;
15          Q : OUT  std_logic;
16          notQ : OUT  std_logic
17         );
18     END COMPONENT;
19
20
21    --Inputs
22    signal D : std_logic := '0';
23    signal CLK : std_logic := '0';
24
25   --Outputs
26    signal Q : std_logic;
27    signal notQ : std_logic;
28
29    -- Clock period definitions
30    constant CLK_period : time := 10 ns;
31
32 BEGIN
33
34   -- Instantiate the Unit Under Test (UUT)
35    uut: ETd_Mazzeo PORT MAP (
36           D => D,
37           CLK => CLK,
38           Q => Q,
39           notQ => notQ
```

```
40          );
41
42     -- Clock process definitions
43     CLK_process :process
44     begin
45      CLK <= '0';
46      wait for CLK_period/2;
47      CLK <= '1';
48      wait for CLK_period/2;
49     end process;
50
51
52     -- Stimulus process
53     stim_proc: process
54     begin
55
56        wait for CLK_period*10;
57
58      wait for 3 ns;
59
60        -- insert stimulus here
61
62      D <= '1';
63      wait for CLK_period;
64
65      D <= '0';
66      wait for 1 ns;
67
68      D <= '1';
69      wait for 3 ns;
70
71      D <= '1';
72      wait for 4 ns;
73
74      D <= '0';
75
76        wait;
77     end process;
78
79 END;
```

Codice Componente 3.8: Definizione del componente ETd$_M azzeo_t b$

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  entity D_MSni is
5      Port ( D : in  STD_LOGIC;
6             CLK : in  STD_LOGIC;
```

```
7             Q : out  STD_LOGIC;
8             notQ : out  STD_LOGIC);
9   end D_MSni;
10
11  architecture Structural of D_MSni is
12
13  component Latch_D
14      Port ( D : in  STD_LOGIC;
15             C : in  STD_LOGIC;
16             Q : out  STD_LOGIC;
17             notQ : out  STD_LOGIC);
18  end component;
19
20  signal TEMP : STD_LOGIC;
21
22  begin
23
24  master : Latch_D
25  Port Map( D => D,
26            C => CLK,
27            Q => TEMP);
28
29  slave : Latch_D
30  Port Map( D => TEMP,
31            C => not CLK,
32            Q => Q,
33            notQ => notQ);
34
35  end Structural;
```

Codice Componente 3.9: Definizione del componente $D_MSni$

```
1
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.ALL;
4
5   ENTITY D_MSni_tb IS
6   END D_MSni_tb;
7
8   ARCHITECTURE behavior OF D_MSni_tb IS
9
10      -- Component Declaration for the Unit Under Test (UUT)
11
12      COMPONENT D_MSni
13      PORT(
14           D : IN  std_logic;
15           CLK : IN  std_logic;
16           Q : OUT  std_logic;
17           notQ : OUT  std_logic
```

```
18          );
19      END COMPONENT;
20
21
22     --Inputs
23     signal D : std_logic := '0';
24     signal CLK : std_logic := '0';
25
26    --Outputs
27     signal Q : std_logic;
28     signal notQ : std_logic;
29
30     -- Clock period definitions
31     constant CLK_period : time := 10 ns;
32
33  BEGIN
34
35    -- Instantiate the Unit Under Test (UUT)
36     uut: D_MSni PORT MAP (
37            D => D,
38            CLK => CLK,
39            Q => Q,
40            notQ => notQ
41          );
42
43     -- Clock process definitions
44     CLK_process :process
45     begin
46      CLK <= '0';
47      wait for CLK_period/2;
48      CLK <= '1';
49      wait for CLK_period/2;
50     end process;
51
52
53     -- Stimulus process
54     stim_proc: process
55     begin
56
57        wait for CLK_period/4;
58
59      D <= '1';
60      wait until(CLK'EVENT AND CLK = '0');
61
62      wait for 1 ns;
63
64      assert Q = '1' AND notQ = '0'
65      report "errore0"
66      severity failure;
```

```
67
68      D <= '0';
69      wait until(CLK'EVENT AND CLK = '0');
70
71      wait for 1 ns;
72
73      assert Q = '0' AND notQ = '1'
74      report "errore1"
75      severity failure;
76
77      D <= '1';
78      wait until(CLK'EVENT AND CLK = '0');
79
80      wait for 1 ns;
81
82      assert Q = '1' AND notQ = '0'
83      report "errore2"
84      severity failure;
85
86      D <= '0';
87      wait until(CLK'EVENT AND CLK = '1'); -- Il flip-flop campiona 0 sul
            fronte di salita
88      wait for 1 ns;
89
90      D <= '1'; -- Il dato varia mentre CLK e' alto
91      wait until(CLK'EVENT AND CLK = '0');
92
93      wait for 1 ns;
94
95      assert Q = '1' AND notQ = '0' -- Non idealita': il flip-flop presenta 1
            sul fronte di discesa
96      report "errore2"
97      severity failure;
98
99      wait;
100
101   end process;
102
103 END;
```

Codice Componente 3.10: Definizione del componente $\mathrm{D}_M Sni_t b$

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  -- Latch_RS 0-attivo (Realizzzione con porte NAND)
6
7  entity Latch_RS0 is
```

```
    Port ( R : in  STD_LOGIC;
           S : in  STD_LOGIC;
           Q : out  STD_LOGIC;
           notQ : out  STD_LOGIC);
end Latch_RS0;

architecture Behavioral of Latch_RS0 is

begin

latch: process(R, S)
begin

  if(S = '1' AND R = '0') then
    Q <= '0';
    notQ <= '1';
  elsif(S = '0' AND R = '1') then
    Q <= '1';
    notQ <= '0';
  elsif(S = '1' AND R = '1') then
    null;
  elsif(S = '0' AND R = '0') then
    Q <= '1';
    notQ <= '1';
  end if;

end process;

end Behavioral;
```

Codice Componente 3.11: Definizione del componente Latch$_RS0$

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity ETs_Mazzeo is
Port ( D : in  STD_LOGIC;
       CLK : in  STD_LOGIC;
       Q : out  STD_LOGIC;
       notQ : out  STD_LOGIC);
end ETs_Mazzeo;

architecture Structural of ETs_Mazzeo is

component Latch_RS0
    Port ( R : in  STD_LOGIC;
           S : in  STD_LOGIC;
           Q : out  STD_LOGIC;
```

```
18            notQ : out  STD_LOGIC);
19 end component;
20
21 signal TEMP : STD_LOGIC;
22 signal R3 : STD_LOGIC;
23 signal S3 : STD_LOGIC;
24 signal RIS1 : STD_LOGIC;
25
26 begin
27
28 TEMP <= CLK and R3;
29
30 latch1 : Latch_RS0
31 port map( R => TEMP,
32        S => D,
33        Q => RIS1,
34        notQ => S3);
35
36 latch2 : Latch_RS0
37 port map( R => RIS1,
38        S => CLK,
39        Q => R3);
40
41 latch3 : Latch_RS0
42 port map( R => R3,
43        S => S3,
44        Q => Q,
45        notQ => notQ);
46
47 end Structural;
```

Codice Componente 3.12: Definizione del componente ETs$_{M}azzeo$

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4
5  entity D_MSi is
6  Port ( D : in  STD_LOGIC;
7             CLK : in  STD_LOGIC;
8             Q : out  STD_LOGIC;
9             notQ : out  STD_LOGIC);
10 end D_MSi;
11
12 architecture Structural of D_MSi is
13
14 component ETd_Mazzeo
15     Port ( D : in  STD_LOGIC;
16            CLK : in  STD_LOGIC;
```

```
17              Q : out  STD_LOGIC;
18              notQ : out  STD_LOGIC);
19  end component;
20
21  component ETs_Mazzeo
22      Port ( D : in  STD_LOGIC;
23             CLK : in  STD_LOGIC;
24             Q : out  STD_LOGIC;
25             notQ : out  STD_LOGIC);
26  end component;
27
28  signal TEMP : STD_LOGIC;
29
30  begin
31
32  master : ETs_Mazzeo
33  Port Map( D => D,
34            CLK => CLK,
35            notQ => TEMP);
36
37  slave : ETd_Mazzeo
38  Port Map( D => TEMP,
39            CLK => CLK,
40            Q => Q,
41            notQ => notQ);
42
43  end Structural;
```

Codice Componente 3.13: Definizione del componente $D_MSi$

```
1
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.ALL;
4
5   ENTITY D_MSi_tb IS
6   END D_MSi_tb;
7
8   ARCHITECTURE behavior OF D_MSi_tb IS
9
10      -- Component Declaration for the Unit Under Test (UUT)
11
12      COMPONENT D_MSi
13      PORT(
14           D : IN  std_logic;
15           CLK : IN  std_logic;
16           Q : OUT  std_logic;
17           notQ : OUT  std_logic
18          );
19      END COMPONENT;
```

```
20
21
22    --Inputs
23    signal D : std_logic := '0';
24    signal CLK : std_logic := '0';
25
26   --Outputs
27    signal Q : std_logic;
28    signal notQ : std_logic;
29
30    -- Clock period definitions
31    constant CLK_period : time := 10 ns;
32
33  BEGIN
34
35   -- Instantiate the Unit Under Test (UUT)
36    uut: D_MSi PORT MAP (
37            D => D,
38            CLK => CLK,
39            Q => Q,
40            notQ => notQ
41          );
42
43    -- Clock process definitions
44    CLK_process :process
45    begin
46     CLK <= '0';
47     wait for CLK_period/2;
48     CLK <= '1';
49     wait for CLK_period/2;
50    end process;
51
52
53    -- Stimulus process
54    stim_proc: process
55    begin
56       -- hold reset state for 100 ns.
57       wait for CLK_period/4;
58
59     D <= '1';
60     wait until(CLK'EVENT AND CLK = '0');
61
62     wait for 1 ns;
63
64     assert Q = '1' AND notQ = '0'
65     report "errore0"
66     severity failure;
67
68     D <= '0';
```

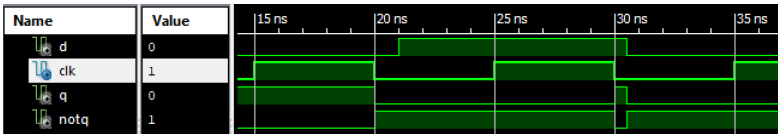
```
69      wait until(CLK'EVENT AND CLK = '0');
70
71      wait for 1 ns;
72
73      assert Q = '0' AND notQ = '1'
74      report "errore1"
75      severity failure;
76
77      D <= '1';
78      wait until(CLK'EVENT AND CLK = '0');
79
80      wait for 1 ns;
81
82      assert Q = '1' AND notQ = '0'
83      report "errore2"
84      severity failure;
85
86      D <= '0';
87      wait until(CLK'EVENT AND CLK = '1'); -- Il flip-flop campiona
88                                           -- 0 sul fronte di salita
89      wait for 1 ns;
90
91      D <= '1'; -- Il dato varia mentre CLK e' alto
92      wait until(CLK'EVENT AND CLK = '0');
93
94      wait for 1 ns;
95
96      assert Q = '0' AND notQ = '1' -- Idealita': il flip-flop
97                                    -- presenta 0 sul fronte di discesa
98      report "errore2"
99      severity failure;
100
101     wait;
102
103       wait;
104   end process;
105
106 END;
```

Codice Componente 3.14: Definizione del componente $D_M Si_t b$

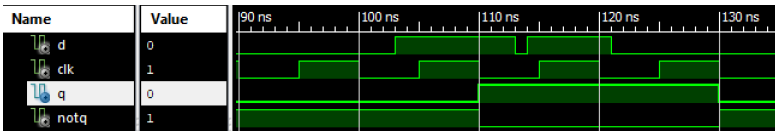
## 3.3 Simulazione

- Nel testbench del componente ETd_Sami è stato evidenziato un caso in cui si manifesta il comportamento non ideale della soluzione proposta da Sami. Va infatti considerato che, non potendo generare fisicamente un impulso ideale, il segnale in uscita dal derivatore è comunque un impulso rettangolare avente una propria durata T (nel nostro caso specifico 1 ns). Durante la durata dell'impulso il dispositivo sarà quindi trasparente, seguendo il comportamento di

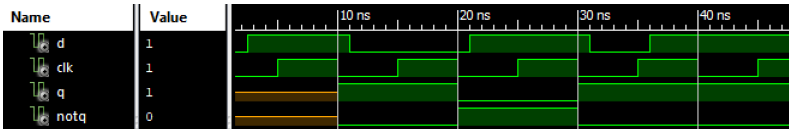
un latch, piuttosto che di un flip-flop edge-triggered. Per evidenziare tale anomalia, si è fatto quindi variare il dato D da 1 a 0 dopo 0.5 T dal fronte di discesa del clock. Secondo il comportamento ideale, il flip-flop dovrebbe ignorare la variazione del dato e continuare a presentare in uscita il valore 1, campionato sul fronte di discesa del clock. Tuttavia, come riportato in figura, l'uscita del dispositivo segue la variazione del dato, comportandosi di fatto come un latch.




- Simulazione del flip flop edge triggered attivo sul fronte di discesa secondo il modello di Mazzeo:

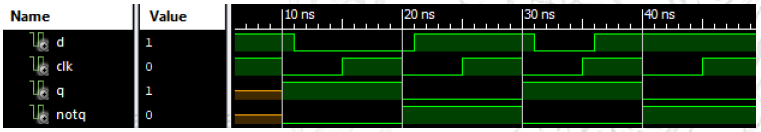



- Simulazione del flip flop master slave non ideale:




Nel testbench è stato evidenziato un caso in cui si manifesta il comportamento non ideale presente nella soluzione realizzata con due latch D. Per evidenziare tale anomalia, si è fatto quindi variare il dato D da 0 a 1 mentre il segnale di clock è alto. Secondo il comportamento del flip-flop master slave ideale, il dispositivo dovrebbe ignorare la variazione del dato e presentare sul fronte di discesa in uscita il valore 0, campionato sul fronte di salita. Tuttavia, come riportato in figura, l'uscita del dispositivo segue la variazione del dato, comportandosi di fatto come un latch nella frazione di tempo in cui il clock è alto.

- Simulazione del flip flop master slave ideale:




## 3.4 Sintesi su board FPGA

Non richiesta dalla traccia.

# Capitolo 4

# Esercizio 4

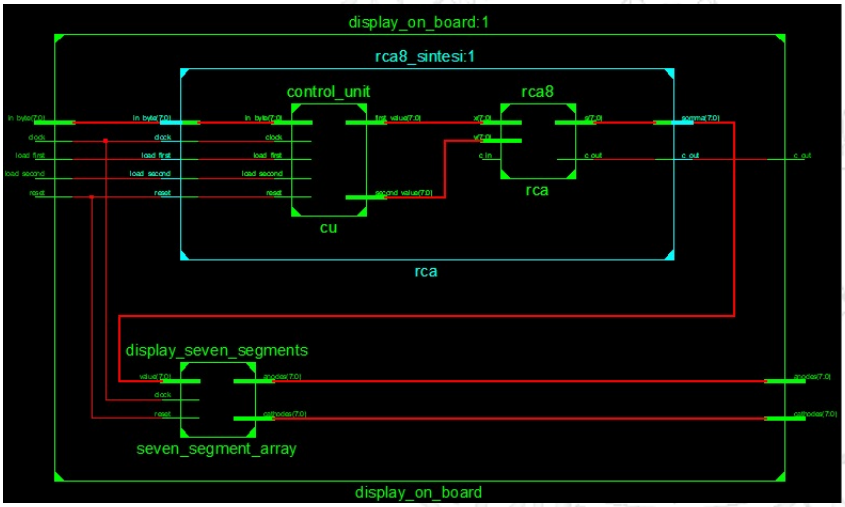
## 4.1 Traccia

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

### 4.2.1 Schematici

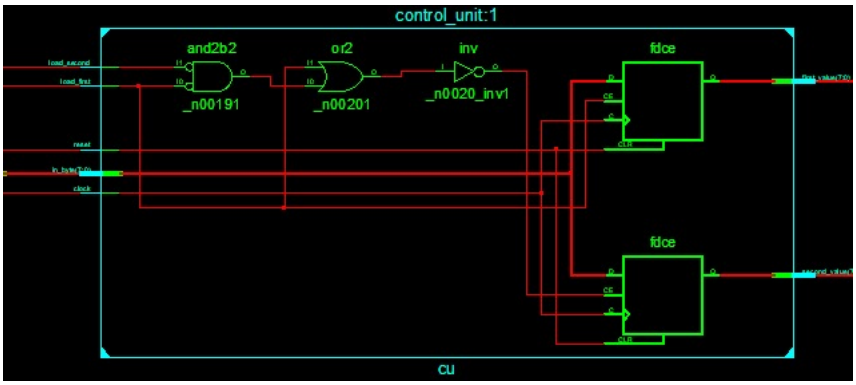
- Schematico generale



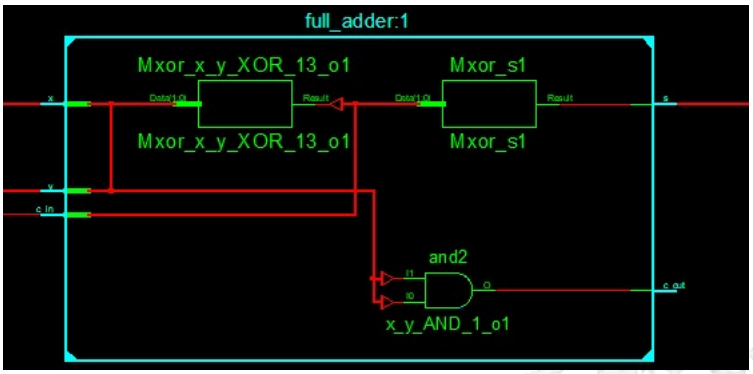

Il ripple carry adder è collegato alla control_unit per poter ricevere gli ingressi dagli switch della board tramite un componente chiamato rca_8_sintesi.

Quest'ultimo è poi connesso al display_seven_segments per permettere che l'uscita sia visualizzata sul display della board.

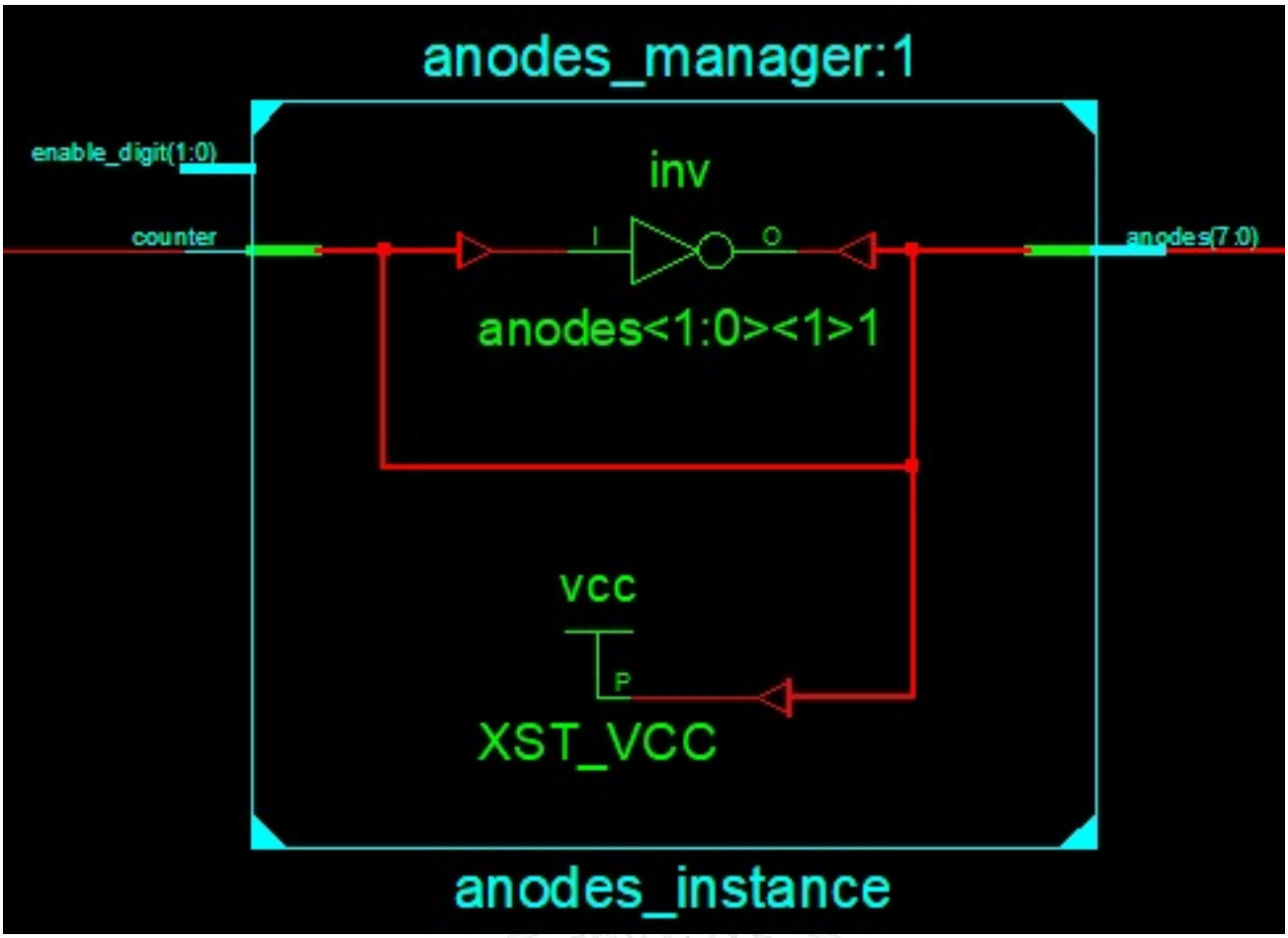
- Schematico control_unit



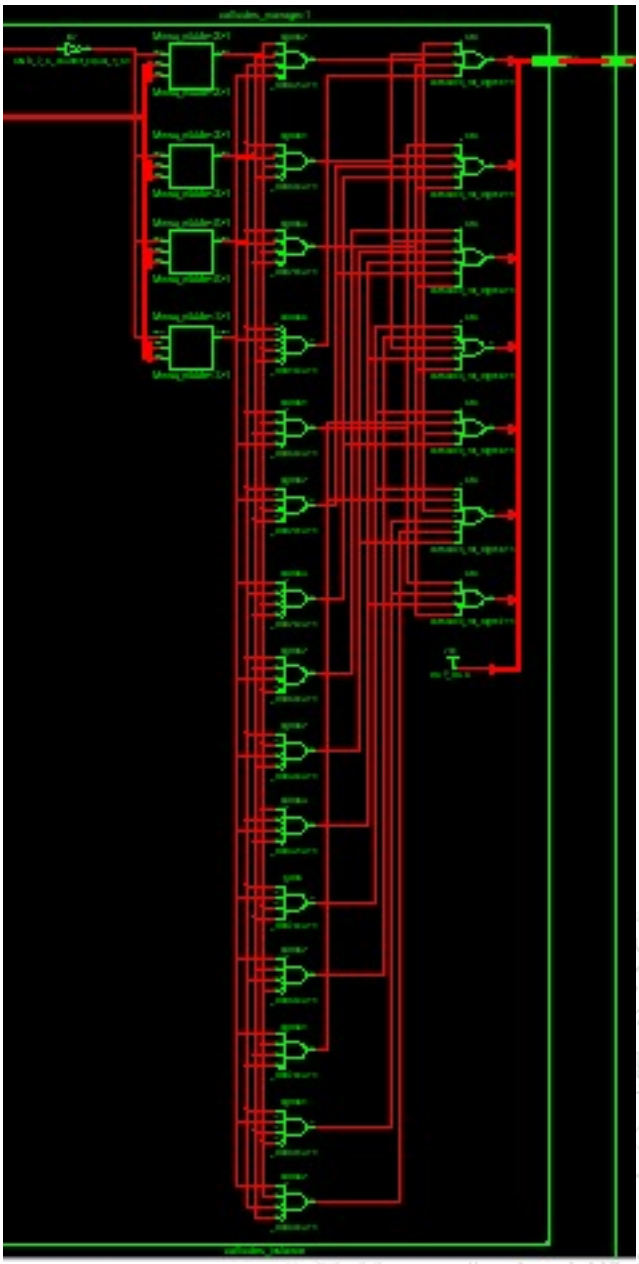

- Schematico full_adder



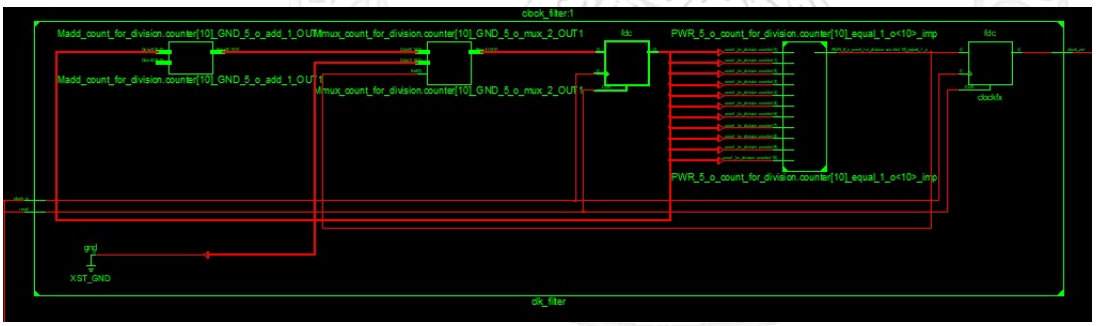

- Schematico anodes_manager



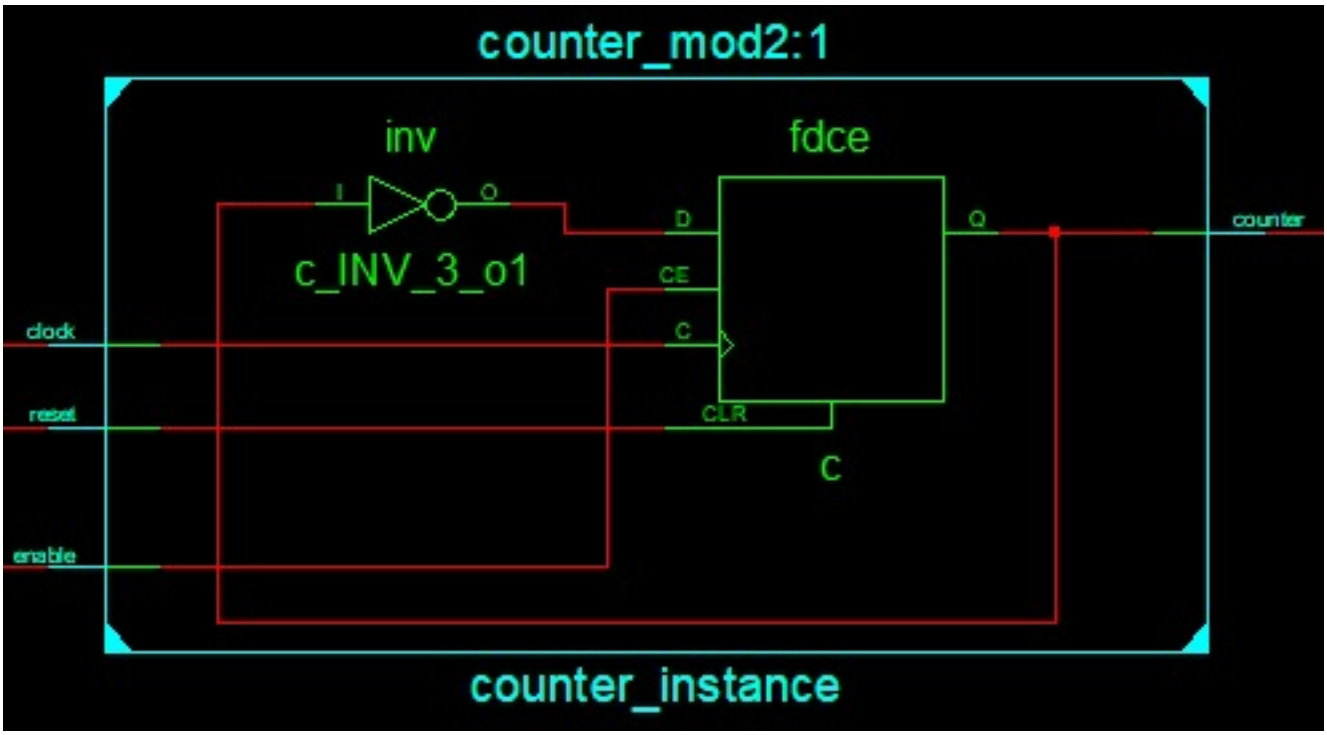

- Schematico cathodes_manager



- Schematico clock_filter




- Schematico counter




### 4.2.2 Codice

#### 4.2.2.1 Ripple_Carry_Adder

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity full_adder is
5      Port ( x : in  STD_LOGIC;
6             y : in  STD_LOGIC;
7             c_in : in  STD_LOGIC;
8             s : out  STD_LOGIC;
9             c_out : out  STD_LOGIC);
10 end full_adder;
11
12 architecture DataFlow of full_adder is
13
14 begin
15
16     s <= (x xor y xor c_in);
17     c_out <= ((x and y) or (c_in and (x xor y)));  --Semplice descrizione
           dataflow di un full adder: posso usare xor e non or nella seconda
           parentesi perchè tanto escluderei solo l'1 associato alla
           configurazione x=y=1 che però è già assicurata dalla x and y della
           prima parentesi. Così facendo ho una struttura più lineare poichè
           posso creare un full adder partendo da 2 half adder e da una porta
           or.
```

```
18
19 end DataFlow;
```

Codice Componente 4.1: Definizione del componente full$_a$*dder*

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity rca4 is -- rca a 4 ingressi, descrizione strutturale a partire dal
       full adder, mi servirà per arrivare all'rca da 8 ingressi
5      Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
6             y : in  STD_LOGIC_VECTOR (3 downto 0);
7             c_in : in  STD_LOGIC;
8             s : out  STD_LOGIC_VECTOR (3 downto 0);
9             c_out : out  STD_LOGIC);
10 end rca4;
11
12 architecture Structural of rca4 is
13
14 component full_adder
15     Port ( x : in  STD_LOGIC;
16            y : in  STD_LOGIC;
17            c_in : in  STD_LOGIC;
18            s : out  STD_LOGIC;
19            c_out : out  STD_LOGIC);
20 end component;
21
22 signal c1, c2, c3 : STD_LOGIC;
23
24 begin
25
26 fa0 : full_adder
27   port map (
28     x => x(0),
29     y => y(0),
30     c_in => c_in,
31     s => s(0),
32     c_out => c1
33   );
34
35 fa1 : full_adder
36   port map (
37     x => x(1),
38     y => y(1),
39     c_in => c1,
40     s => s(1),
41     c_out => c2
42   );
43
```

```
44 fa2 : full_adder
45   port map (
46     x => x(2),
47     y => y(2),
48     c_in => c2,
49     s => s(2),
50     c_out => c3
51   );
52
53 fa3 : full_adder
54   port map (
55     x => x(3),
56     y => y(3),
57     c_in => c3,
58     s => s(3),
59     c_out => c_out
60   );
61
62 end Structural;
```

Codice Componente 4.2: Definizione del componente rca4

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity rca8 is
6      Port ( x : in  STD_LOGIC_VECTOR (7 downto 0);
7             y : in  STD_LOGIC_VECTOR (7 downto 0);
8             c_in : in  STD_LOGIC;
9             s : out  STD_LOGIC_VECTOR (7 downto 0);
10            c_out : out  STD_LOGIC);
11 end rca8;
12
13 architecture Structural of rca8 is
14
15 component rca4
16     Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
17            y : in  STD_LOGIC_VECTOR (3 downto 0);
18            c_in : in  STD_LOGIC;
19            s : out  STD_LOGIC_VECTOR (3 downto 0);
20            c_out : out  STD_LOGIC);
21 end component;
22
23 signal c : STD_LOGIC;
24
25 begin
26
27 rca4_0 : rca4
```

```
28    port map(
29      x => x(3 downto 0),
30      y => y(3 downto 0),
31      c_in => c_in,
32      s => s(3 downto 0),
33      c_out => c
34    );
35
36  rca4_1 : rca4
37    port map(
38      x => x(7 downto 4),
39      y => y(7 downto 4),
40      c_in => c,
41      s => s(7 downto 4),
42      c_out => c_out
43    );
44
45  end Structural;
```

Codice Componente 4.3: Definizione del componente rca8

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity control_unit is
6       Port ( clock : in  STD_LOGIC;
7              reset : in  STD_LOGIC;
8              load_first : in  STD_LOGIC;
9              load_second : in  STD_LOGIC;
10             in_byte : in  STD_LOGIC_VECTOR (7 downto 0);
11             first_value : out  STD_LOGIC_VECTOR (7 downto 0);
12             second_value : out  STD_LOGIC_VECTOR (7 downto 0));
13  end control_unit;
14
15  architecture Behavioral of control_unit is
16
17  signal reg_first : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
18  signal reg_second : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
19
20  begin
21
22  first_value <= reg_first;
23  second_value <= reg_second;
24
25  main : process(clock, reset)
26  begin
27
28    if reset = '1' then
```

```
29      reg_first <= (others => '0');
30      reg_second <= (others => '0');
31    elsif clock'event and clock = '1' then
32      if load_first = '1' then
33        reg_first(7 downto 0) <= in_byte;
34      elsif load_second = '1' then
35        reg_second(7 downto 0) <= in_byte;
36      end if;
37    end if;
38
39
40  end process;
41
42  end Behavioral;
```

Codice Componente 4.4: Definizione del componente control$_unit$

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity rca8_sintesi is
6       Port ( clock : in  STD_LOGIC;
7              reset : in  STD_LOGIC;
8              in_byte : in  STD_LOGIC_VECTOR (7 downto 0);
9              load_first : in  STD_LOGIC;
10             load_second : in  STD_LOGIC;
11             somma : out  STD_LOGIC_VECTOR (7 downto 0);
12             c_out : out  STD_LOGIC);
13  end rca8_sintesi;
14
15  architecture Structural of rca8_sintesi is
16
17  component control_unit
18     Port ( clock : in  STD_LOGIC;
19             reset : in  STD_LOGIC;
20             load_first : in  STD_LOGIC;
21             load_second : in  STD_LOGIC;
22             in_byte : in  STD_LOGIC_VECTOR (7 downto 0);
23             first_value : out  STD_LOGIC_VECTOR (7 downto 0);
24             second_value : out  STD_LOGIC_VECTOR (7 downto 0));
25  end component;
26
27  component rca8
28      Port ( x : in  STD_LOGIC_VECTOR (7 downto 0);
29             y : in  STD_LOGIC_VECTOR (7 downto 0);
30             c_in : in  STD_LOGIC;
31             s : out  STD_LOGIC_VECTOR (7 downto 0);
32             c_out : out  STD_LOGIC);
```

```
33 end component;
34
35 signal first : STD_LOGIC_VECTOR (7 downto 0);
36 signal second : STD_LOGIC_VECTOR (7 downto 0);
37
38 -- segnali per associare gli ingressi dell'rca8 a quelli pilotati dalla
       control unit
39
40 begin
41
42 cu : control_unit
43   port map (
44     clock => clock,
45     reset => reset,
46     load_first => load_first,
47     load_second => load_second,
48     in_byte => in_byte,
49     first_value => first,
50     second_value => second
51     );
52
53 rca : rca8
54   port map (
55     x => first,
56     y => second,
57     c_in => '0',
58     s => somma,
59     c_out => c_out
60     );
61
62 end Structural;
```

Codice Componente 4.5: Definizione del componente $\text{rca8}_{s}intesi$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity anodes_manager is
8      Port (  counter : in STD_LOGIC;  -- necessario per utilizzare 2 cifre
           del display
9             enable_digit : in  STD_LOGIC_VECTOR (1 downto 0);
10              anodes : out  STD_LOGIC_VECTOR (7 downto 0)
11          );
12 end anodes_manager;
13
14 architecture Behavioral of anodes_manager is
```

```
15
16 signal anodes_switching : std_logic_vector(1 downto 0) := (others => '0');
17
18 begin
19
20 anodes(1 downto 0) <= not enable_digit or not anodes_switching;
21 anodes ( 7 downto 2) <= (others => '1');
22
23 anodes_process: process(counter, enable_digit)
24 begin
25   --a seconda del valore di counter le cifre si illuminano una alla volta da
       destra a sinistra
26   case counter is
27     when '0' =>
28       anodes_switching <= "01";
29     when '1' =>
30       anodes_switching <= "10";
31     when others =>
32       anodes_switching <= (others => '0');
33   end case;
34
35 end process;
36
37
38 end Behavioral;
```

Codice Componente 4.6: Definizione del componente anodes$_m$*anager*

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  use IEEE.NUMERIC_STD.ALL;
5
6  entity cathodes_manager is
7      Port (
8           counter : in STD_LOGIC; -- necessario per capire ogni cifra quale
               valore deve assumere
9           value : in  STD_LOGIC_VECTOR (7 downto 0);
10           cathodes : out  STD_LOGIC_VECTOR (7 downto 0));
11 end cathodes_manager;
12
13 architecture Behavioral of cathodes_manager is
14
15 constant zero   : std_logic_vector(6 downto 0) := "1000000";
16 constant one    : std_logic_vector(6 downto 0) := "1111001";
17 constant two    : std_logic_vector(6 downto 0) := "0100100";
18 constant three  : std_logic_vector(6 downto 0) := "0110000";
19 constant four   : std_logic_vector(6 downto 0) := "0011001";
20 constant five   : std_logic_vector(6 downto 0) := "0010010";
```

```
21 constant six    : std_logic_vector(6 downto 0) := "0000010";
22 constant seven  : std_logic_vector(6 downto 0) := "1111000";
23 constant eight  : std_logic_vector(6 downto 0) := "0000000";
24 constant nine   : std_logic_vector(6 downto 0) := "0010000";
25 constant a     : std_logic_vector(6 downto 0) := "0001000";
26 constant b      : std_logic_vector(6 downto 0) := "0000011";
27 constant c      : std_logic_vector(6 downto 0) := "1000110";
28 constant d      : std_logic_vector(6 downto 0) := "0100001";
29 constant e      : std_logic_vector(6 downto 0) := "0000110";
30 constant f      : std_logic_vector(6 downto 0) := "0001110";
31
32 alias digit_0 is value (3 downto 0);  -- alias per la cifra esadecimale meno
       significativa
33 alias digit_1 is value (7 downto 4);  -- alias per la cifra esadecimale più
       significativa
34
35 signal cathodes_for_digit : std_logic_Vector(6 downto 0) := (others => '0');
36
37 signal nibble :std_logic_vector(3 downto 0) := (others => '0'); -- segnale
       di appoggio per associare a ciascun alias il giusto valore
38
39 begin
40
41 digit_switching: process(counter, value)
42
43 begin
44   case counter is
45     when '0' =>
46       nibble <= digit_0;   -- quando il contatore è 0 la cifra attiva è la
           prima da destra quindi il valore del nibble è associato a digit_0
47     when '1' =>
48       nibble <= digit_1;
49     when others =>
50       nibble <= (others => '0');
51   end case;
52 end process;
53
54
55 seven_segment_decoder_process: process(nibble)
56   begin
57     case nibble is
58       when "0000" => cathodes_for_digit <= zero;
59       when "0001" => cathodes_for_digit <= one;
60       when "0010" => cathodes_for_digit <= two;
61       when "0011" => cathodes_for_digit <= three;
62       when "0100" => cathodes_for_digit <= four;
63       when "0101" => cathodes_for_digit <= five;
64       when "0110" => cathodes_for_digit <= six;
65       when "0111" => cathodes_for_digit <= seven;
```

```
      when "1000" => cathodes_for_digit <= eight;
      when "1001" => cathodes_for_digit <= nine;
      when "1010" => cathodes_for_digit <= a;
      when "1011" => cathodes_for_digit <= b;
      when "1100" => cathodes_for_digit <= c;
      when "1101" => cathodes_for_digit <= d;
      when "1110" => cathodes_for_digit <= e;
      when "1111" => cathodes_for_digit <= f;
    when others => cathodes_for_digit <= (others => '0');
    end case;
  end process seven_segment_decoder_process;

cathodes (6 downto 0) <= cathodes_for_digit;
cathodes(7) <= '1';

end Behavioral;
```

Codice Componente 4.7: Definizione del componente cathodes$_m$*anager*

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

entity clock_filter is  -- necessario un clock filter quando vengono mostrate più cifre per evitare che la frequenza sia troppa elevata per permettere l'aggiornamento dell'uscita dopo il calcolo
   generic(
        clock_frequency_in : integer := 100000000;
        clock_frequency_out : integer := 50000  -- frequenza abbassata ma non troppo per non permettere all'occhio umano di accorgersene
        );
    Port ( clock_in : in  STD_LOGIC;
        reset : in STD_LOGIC;
           clock_out : out  STD_LOGIC);
end clock_filter;

architecture Behavioral of clock_filter is

signal clockfx : STD_LOGIC := '0';

constant count_max_value : integer := clock_frequency_in/(clock_frequency_out)-1;

begin

clock_out <= clockfx;

count_for_division: process(clock_in, reset)
variable counter : integer range 0 to count_max_value := 0;
begin
```

```
27
28    if reset = '1' then
29      counter := 0;
30      clockfx <= '0';
31    elsif clock_in'event and clock_in = '1' then
32      if counter = count_max_value then
33        clockfx <=  '1';
34        counter := 0;
35      else
36        clockfx <=  '0';
37        counter := counter + 1;
38      end if;
39    end if;
40
41  end process;
42
43
44  end Behavioral;
```

Codice Componente 4.8: Definizione del componente clock$_filter$

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity counter_mod2 is
6       Port ( clock : in  STD_LOGIC;
7              reset : in  STD_LOGIC;
8           enable : in STD_LOGIC;
9              counter : out  STD_LOGIC
10          );
11  end counter_mod2;
12
13  architecture Behavioral of counter_mod2 is
14
15  signal c : std_logic := '0';
16
17  begin
18
19  counter <= c;
20
21  counter_process: process(clock, reset, c, enable)
22  begin
23
24    if reset = '1' then
25      c <= '0';
26    elsif clock'event AND clock = '1' AND enable = '1' then
27      c <= not c;
28    end if;
```

```
29
30 end process;
31
32 end Behavioral;
```

Codice Componente 4.9: Definizione del componente counter$_m od 2$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity display_seven_segments is
5  Generic(
6          clock_frequency_in : integer := 100000000; --parametri da
               customizzare, questo è il valore di default
7          clock_frequency_out : integer := 50000
8          );
9     Port ( clock : in  STD_LOGIC;
10              reset : in  STD_LOGIC;
11              value : in  STD_LOGIC_VECTOR (7 downto 0);
12           anodes : out  STD_LOGIC_VECTOR (7 downto 0);
13              cathodes : out  STD_LOGIC_VECTOR (7 downto 0));
14 end display_seven_segments;
15
16 architecture Structural of display_seven_segments is
17
18
19 signal counter : std_logic;
20 signal clock_fx : std_logic := '0';
21
22
23 COMPONENT cathodes_manager
24   Port (
25           counter : in STD_LOGIC;
26           value : in  STD_LOGIC_VECTOR (7 downto 0);
27            cathodes : out  STD_LOGIC_VECTOR (7 downto 0)
28         );
29 END COMPONENT;
30
31 COMPONENT anodes_manager
32    Port (
33            counter : in STD_LOGIC;
34            enable_digit : in  STD_LOGIC_VECTOR (1 downto 0);
35             anodes : out  STD_LOGIC_VECTOR (7 downto 0)
36          );
37 END COMPONENT;
38
39 COMPONENT clock_filter
40   GENERIC(
41          clock_frequency_in : integer := 100000000;
```

```
42          clock_frequency_out : integer := 50000
43        );
44    PORT(
45          clock_in : IN std_logic;
46           reset : in  STD_LOGIC;
47         clock_out : OUT std_logic
48      );
49  END COMPONENT;
50
51  COMPONENT counter_mod2
52    PORT(
53      clock : in  STD_LOGIC;
54        reset : in  STD_LOGIC;
55      enable : in STD_LOGIC;
56        counter : out  STD_LOGIC
57      );
58  END COMPONENT;
59
60  begin
61
62  clk_filter: clock_filter GENERIC MAP(
63    clock_frequency_in => clock_frequency_in,
64    clock_frequency_out => clock_frequency_out
65    )
66    PORT MAP(
67      clock_in => clock,
68      reset => reset,
69      clock_out => clock_fx
70    );
71
72  cathodes_instance: cathodes_manager port map(
73    value => value,
74    cathodes => cathodes,
75    counter => counter
76  );
77  counter_instance: counter_mod2 port map(
78    clock => clock,
79    enable => clock_fx,
80    reset => reset,
81    counter => counter
82  );
83
84
85  anodes_instance: anodes_manager port map(
86    enable_digit => (others => '1'),
87    anodes => anodes,
88    counter => counter
89  );
90
```

```
91 end Structural;
```

Codice Componente 4.10: Definizione del componente $display_seven_segment$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity display_on_board is
5    Port(
6        clock : in  STD_LOGIC;
7        reset : in  STD_LOGIC;
8        load_first : in  STD_LOGIC;
9          load_second : in  STD_LOGIC;
10       in_byte : in STD_LOGIC_VECTOR(7 downto 0);
11       anodes : out  STD_LOGIC_VECTOR (7 downto 0);
12       cathodes : out  STD_LOGIC_VECTOR (7 downto 0);
13       c_out : out STD_LOGIC
14        );
15
16 end display_on_board;
17
18 architecture Structural of display_on_board is
19
20 COMPONENT display_seven_segments
21 GENERIC(
22         clock_frequency_in : integer := 100000000;
23         clock_frequency_out : integer := 50000
24         );
25   PORT(
26     clock : IN std_logic;
27     reset : IN std_logic;
28     value : IN std_logic_vector(7 downto 0);--4 nibble da mostrare
29     anodes : OUT std_logic_vector(7 downto 0);
30     cathodes : OUT std_logic_vector(7 downto 0)
31     );
32 END COMPONENT;
33
34 COMPONENT rca8_sintesi
35    Port ( clock : in  STD_LOGIC;
36           reset : in  STD_LOGIC;
37         load_first : in  STD_LOGIC;
38           load_second : in  STD_LOGIC;
39         in_byte : in STD_LOGIC_VECTOR(7 downto 0);
40           somma : out  STD_LOGIC_VECTOR (7 downto 0);
41           c_out : out  STD_LOGIC);
42 end component;
43
44 signal cu_value : std_logic_vector(7 downto 0);
45
```

```
46 begin
47
48 seven_segment_array: display_seven_segments
49 GENERIC MAP(
50   clock_frequency_in => 100000000,
51   clock_frequency_out => 50000
52   )
53   PORT MAP(
54     clock => clock,
55     reset => reset,
56     value => cu_value,
57     anodes => anodes,
58     cathodes => cathodes
59 );
60
61 rca: rca8_sintesi PORT MAP(
62     clock => clock,
63     reset => reset,
64     load_first => load_first,
65     load_second => load_second,
66     in_byte => in_byte,
67     somma => cu_value,
68     c_out => c_out
69
70   );
71
72 end Structural;
```

Codice Componente 4.11: Definizione del componente display$_o n_b oard$

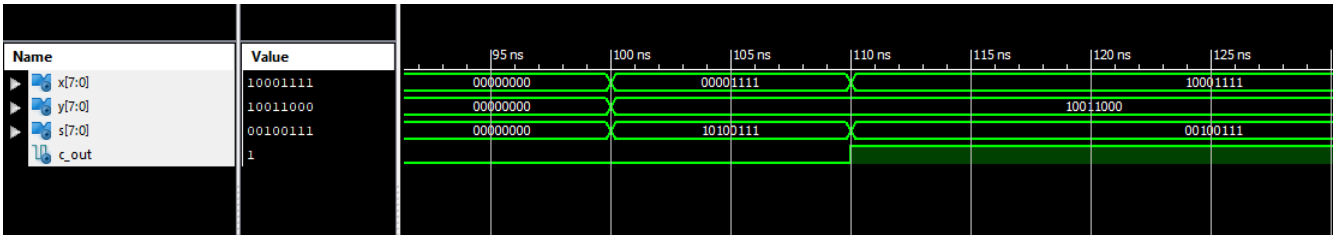
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  -- Uncomment the following library declaration if using
5  -- arithmetic functions with Signed or Unsigned values
6  --USE ieee.numeric_std.ALL;
7
8  ENTITY rca_8_tb IS
9  END rca_8_tb;
10
11 ARCHITECTURE behavior OF rca_8_tb IS
12
13     -- Component Declaration for the Unit Under Test (UUT)
14
15     COMPONENT rca_8
16     PORT(
17          x : IN  std_logic_vector(7 downto 0);
18          y : IN  std_logic_vector(7 downto 0);
19          s : OUT  std_logic_vector(7 downto 0);
```

```
20          c_out : OUT  std_logic
21        );
22     END COMPONENT;
23
24
25    --Inputs
26    signal x : std_logic_vector(7 downto 0) := (others => '0');
27    signal y : std_logic_vector(7 downto 0) := (others => '0');
28
29   --Outputs
30    signal s : std_logic_vector(7 downto 0);
31    signal c_out : std_logic;
32    -- No clocks detected in port list. Replace <clock> below with
33    -- appropriate port name
34
35 --    constant <clock>_period : time := 10 ns;
36
37 BEGIN
38
39   -- Instantiate the Unit Under Test (UUT)
40    uut: rca_8 PORT MAP (
41           x => x,
42           y => y,
43           s => s,
44           c_out => c_out
45         );
46
47    -- Clock process definitions
48    --<clock>_process :process
49    --begin
50     --<clock> <= '0';
51     --wait for <clock>_period/2;
52     --<clock> <= '1';
53     --wait for <clock>_period/2;
54    --end process;
55
56
57    -- Stimulus process
58    stim_proc: process
59    begin
60       -- hold reset state for 100 ns.
61       wait for 100 ns;
62
63       --wait for <clock>_period*10;
64
65       -- insert stimulus here (casi di test che verranno valutati in
          simulazione, non uso assert)
66
67     x <= "00001111";
```

```
68      y <= "10011000";
69
70      wait for 10 ns;
71
72        x <= "10001111";
73      y <= "10011000";
74        wait;
75     end process;
76  END;
```

Codice Componente 4.12: Definizione del componente $\text{rca}_{8t}b$

## 4.3 Simulazione




Nella simulazione sono presenti due casi di test che verificano il corretto funzionamento del ripple carry adder.

## 4.4 Sintesi su board FPGA

Per interfacciare il design realizzato con l'I/O di board sono state utilizzate le seguenti locazioni:

Per il clock associo il clock della board al segnale "clock", che però non è quello utilizzato perchè c'è in aggiunta un clock_filter che genera un clock_fx a minore frequenza.

```
## Clock signal
NET "clock"    LOC = "E3"   | IOSTANDARD = "LVCMOS33";
```

I due addendi vengono presi dagli switch, quindi è necessario utilizzare 8 switch della board, in questo caso vengono scelti i primi 8 a partire da destra.

```
## Switches
NET "in_byte<0>"          LOC=J15 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_RS0_15
NET "in_byte<1>"          LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "in_byte<2>"          LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
NET "in_byte<3>"          LOC=R15 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_14
NET "in_byte<4>"          LOC=R17 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_14
NET "in_byte<5>"          LOC=T18 | IOSTANDARD=LVCMOS33; #IO_L7N_T1_D10_14
NET "in_byte<6>"          LOC=U18 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A13_D29_14
NET "in_byte<7>"          LOC=R13 | IOSTANDARD=LVCMOS33; #IO_L5N_T0_D07_14
```

Per il reset e per il caricamento degli addendi si utilizzano i 3 seguenti bottoni presenti sulla board.

```
## Buttons
NET "reset"           LOC=M17 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_D15_14
NET "load_second"           LOC=M18 | IOSTANDARD=LVCMOS33; #IO_L4N_T0_D05_14
NET "load_first"          LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14
```

Viene utilizzato inoltre un led che si accende nel caso si verifichi una condizione di overflow, in questo caso il primo a partire da destra.

```
## LEDs
NET "c_out"         LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
```

Per il display valgono le considerazioni del precedente esercizio, quindi utilizzo tutti gli anodi e tutti i catodi.

```
## 7 segment display
NET "cathodes<0>"             LOC=T10 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_A00_D16_14
NET "cathodes<1>"             LOC=R10 | IOSTANDARD=LVCMOS33; #IO_25_14
NET "cathodes<2>"             LOC=K16 | IOSTANDARD=LVCMOS33; #IO_25_15
NET "cathodes<3>"             LOC=K13 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A26_15
NET "cathodes<4>"             LOC=P15 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_14
NET "cathodes<5>"             LOC=T11 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A10_D26_14
NET "cathodes<6>"             LOC=L18 | IOSTANDARD=LVCMOS33; #IO_L4P_T0_D04_14
NET "cathodes<7>"             LOC=H15 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_A21_VREF_15

NET "anodes<0>"          LOC=J17 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_FOE_B_15
NET "anodes<1>"          LOC=J18 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_FWE_B_15
NET "anodes<2>"          LOC=T9 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_A01_D17_14
NET "anodes<3>"          LOC=J14 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A22_15
NET "anodes<4>"          LOC=P14 | IOSTANDARD=LVCMOS33; #IO_L8N_T1_D12_14
NET "anodes<5>"          LOC=T14 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_14
NET "anodes<6>"          LOC=K2 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_35
NET "anodes<7>"          LOC=U13 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_A02_D18_14
```

# Capitolo 5

# Esercizio 5

## 5.1 Traccia

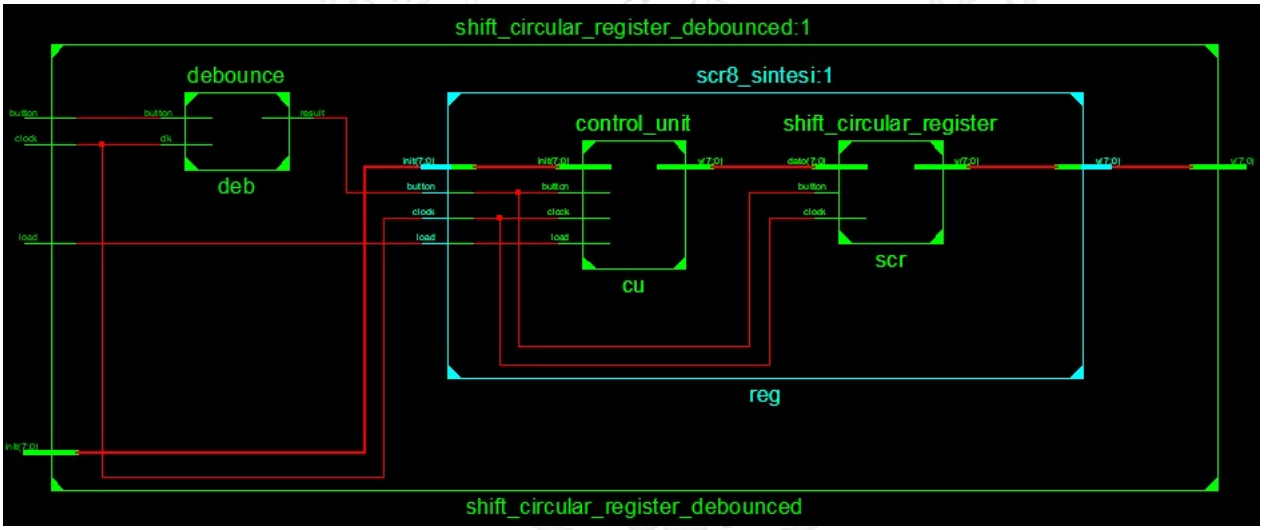
Progettare ed implementare in VHDL un registro a scorrimento circolare di 8 bit utilizzando i flip-flop D edge triggered implementati all'esercizio 3 (si scelga una delle realizzazioni). Sintetizzare sulla board il componente utilizzando gli switch per acquisire il dato iniziale, un bottone per acquisire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

## 5.2 Soluzione

Per la soluzione si è deciso di utilizzare il flip flop edge triggered con il modello utilizzato dal professore Mazzeo.

### 5.2.1 Schematici

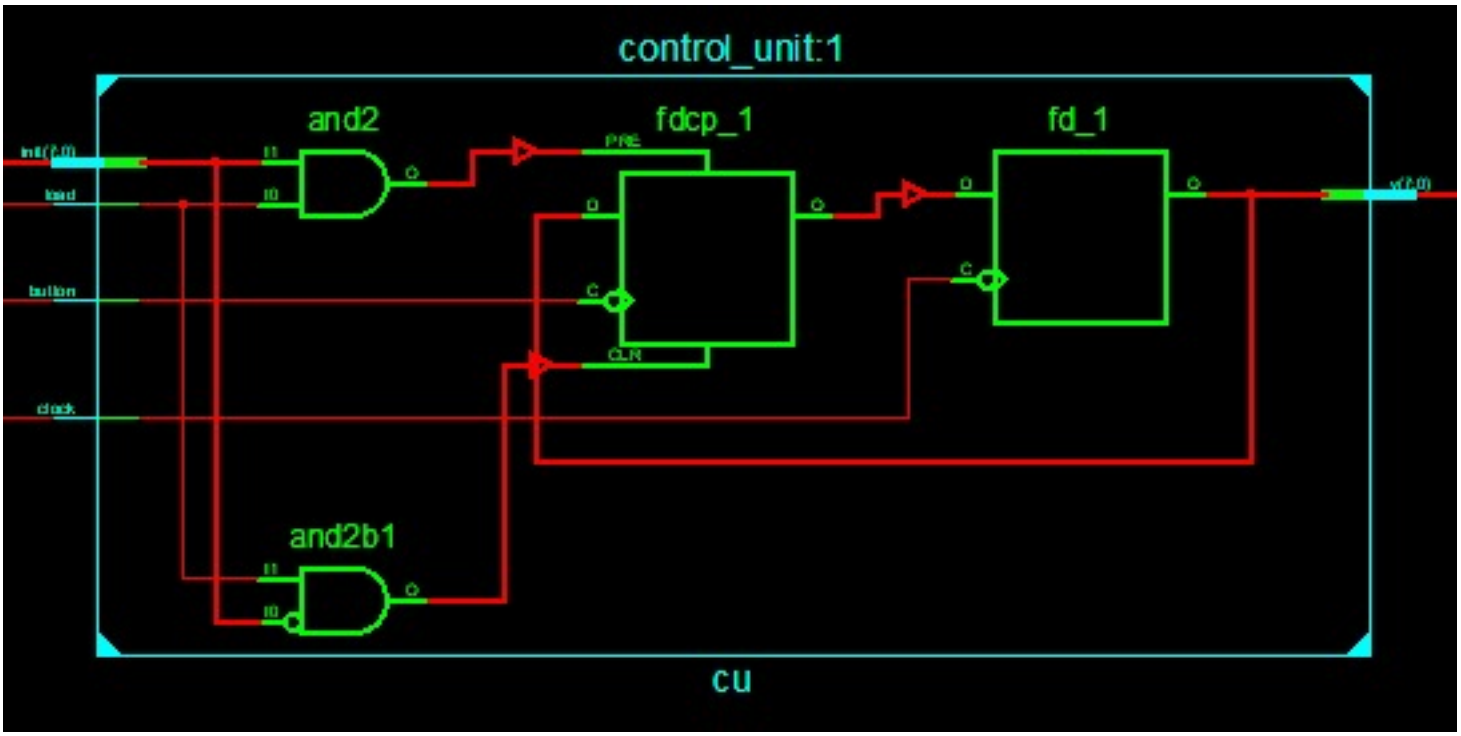
- Schematico generale



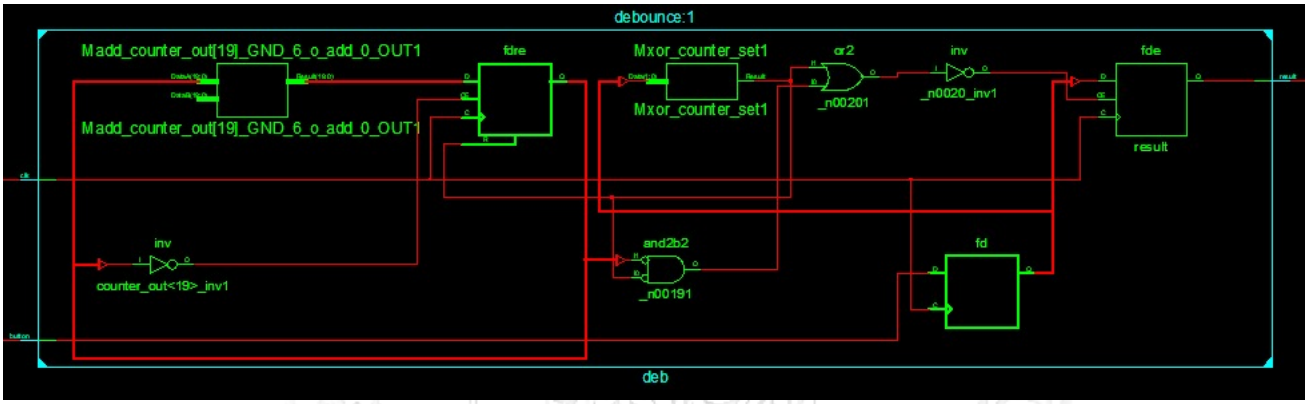

Si nota la presenza di un componente, chiamato debounce, necessario per permettere il corretto funzionamento del bottone della board.

L'et_mazzeo ha il reset a massa, perchè non viene mai utilizzato in questo progetto.

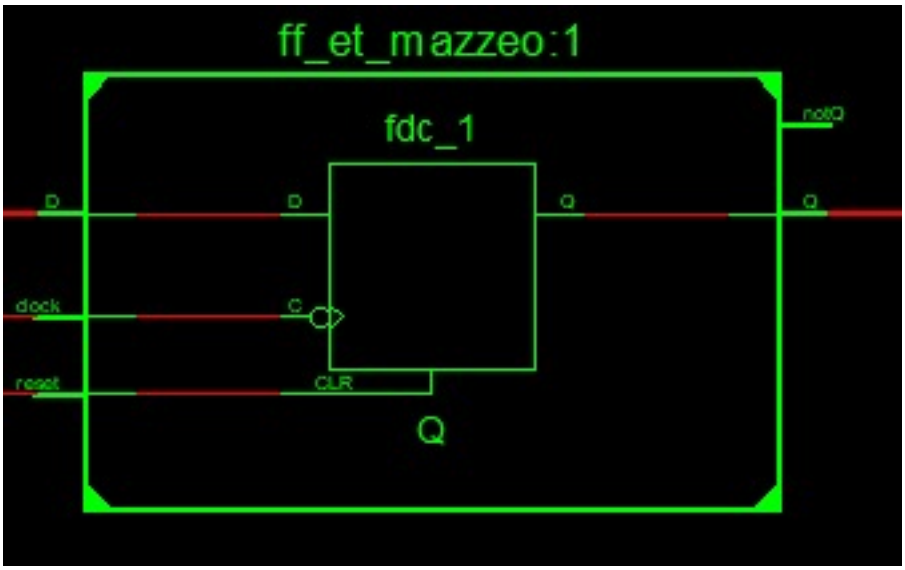
- Schematico control unit




- Schematico debounce




- Schematico et_mazzeo




### 5.2.2 Codice

#### 5.2.2.1 Shift_Circular_Register

```
1 et_mazzeo - traccia n° 3
```

Codice Componente 5.1: Definizione del componente et$_m azzeo$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity shift_circular_register is
5      Port ( clock : in  STD_LOGIC;
6             button : in  STD_LOGIC;
7             dato : in  STD_LOGIC_VECTOR (7 downto 0);
8             y : out  STD_LOGIC_VECTOR (7 downto 0));
9  end shift_circular_register;
10
11 architecture Structural of shift_circular_register is
12
13 component ff_et_mazzeo
14     Port ( clock : in  STD_LOGIC;
15            reset : in  STD_LOGIC;
16            D : in  STD_LOGIC;
17            Q : out  STD_LOGIC;
18            notQ : out  STD_LOGIC);
19 end component;
20
21 signal t : std_logic_vector (7 downto 0);
22 signal x : std_logic_vector (7 downto 0) := (others => '0');
23
24 begin
```

```
25
26 ff0 : ff_et_mazzeo
27     PORT MAP( clock => clock,
28               reset => '0',
29               D => dato(0),
30               Q => y(0),
31               notQ => x(0)
32             );
33
34 ff1 : ff_et_mazzeo
35     Port Map( clock => clock,
36               reset => '0',
37               D => dato(1),
38               Q => y(1),
39               notQ => x(1)
40             );
41
42 ff2 : ff_et_mazzeo
43     Port Map( clock => clock,
44               reset => '0',
45               D => dato(2),
46               Q => y(2),
47               notQ => x(2)
48             );
49
50 ff3 : ff_et_mazzeo
51     Port Map( clock => clock,
52               reset => '0',
53               D => dato(3),
54               Q => y(3),
55               notQ => x(3)
56             );
57
58 ff4 : ff_et_mazzeo
59     Port Map( clock => clock,
60               reset => '0',
61               D => dato(4),
62               Q => y(4),
63               notQ => x(4)
64             );
65
66 ff5 : ff_et_mazzeo
67     Port Map( clock => clock,
68               reset => '0',
69               D => dato(5),
70               Q => y(5),
71               notQ => x(5)
72             );
73
```

```
74 ff6 : ff_et_mazzeo
75     Port Map( clock => clock,
76               reset => '0',
77               D => dato(6),
78               Q => y(6),
79               notQ => x(6)
80             );
81 
82 ff7 : ff_et_mazzeo
83     Port Map( clock => clock,
84               reset => '0',
85               D => dato(7),
86               Q => y(7),
87               notQ => x(7)
88             );
89 
90 
91 end Structural;
```

Codice Componente 5.2: Definizione del componente $\text{shift}_c ircular_r egister$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  
4  entity control_unit is
5      Port ( load : in  STD_LOGIC;
6             clock : in  STD_LOGIC;
7             button : in  STD_LOGIC;
8             init : in  STD_LOGIC_VECTOR (7 downto 0);
9             y : out  STD_LOGIC_VECTOR (7 downto 0));
10 end control_unit;
11 
12 architecture Behavioral of control_unit is
13 
14 signal y_temp : STD_LOGIC_VECTOR (7 downto 0);
15 
16 begin
17 
18 main : process(clock)
19 begin
20 
21 y <= y_temp;
22 
23 if(clock'event and clock = '0') then
24   if(load = '1') then
25     y_temp <= init;
26   elsif (button'event and button = '0') then
27     y_temp(1) <= y_temp(0);
28     y_temp(2) <= y_temp(1);
```

```
29      y_temp(3) <= y_temp(2);
30      y_temp(4) <= y_temp(3);
31      y_temp(5) <= y_temp(4);
32      y_temp(6) <= y_temp(5);
33      y_temp(7) <= y_temp(6);
34      y_temp(0) <= y_temp(7);
35    end if;
36  end if;
37
38  end process;
39
40  end Behavioral;
```

Codice Componente 5.3: Definizione del componente control$_unit$

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity scr8_sintesi is
6       Port ( clock : in  STD_LOGIC;
7              button : in  STD_LOGIC;
8              init : in  STD_LOGIC_VECTOR (7 downto 0);
9              load : in  STD_LOGIC;
10             y : out  STD_LOGIC_VECTOR (7 downto 0));
11  end scr8_sintesi;
12
13  architecture Structural of scr8_sintesi is
14
15  component control_unit
16      Port ( load : in  STD_LOGIC;
17             clock : in  STD_LOGIC;
18             button : in  STD_LOGIC;
19             init : in  STD_LOGIC_VECTOR (7 downto 0);
20             y : out  STD_LOGIC_VECTOR (7 downto 0));
21  end component;
22
23  component shift_circular_register
24      Port ( clock : in  STD_LOGIC;
25             button : in  STD_LOGIC;
26             dato : in  STD_LOGIC_VECTOR (7 downto 0);
27             y : out  STD_LOGIC_VECTOR (7 downto 0));
28  end component;
29
30  signal temp : STD_LOGIC_VECTOR (7 downto 0);
31
32  begin
33
34  cu : control_unit
```

```
35      PORT MAP( load => load,
36                clock => clock,
37                button => button,
38                init => init,
39                y => temp
40              );
41
42  scr : shift_circular_register
43      PORT MAP( clock => clock,
44                button => button,
45                dato => temp,
46                y => y
47              );
48
49  end Structural;
```

Codice Componente 5.4: Definizione del scr8$_s$*intesi*

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.std_logic_unsigned.all;
4
5   ENTITY debounce IS
6     GENERIC(
7       counter_size  :  INTEGER := 19); --counter size (19 bits gives 10.5ms
          with 50MHz clock)
8     PORT(
9       clk     : IN  STD_LOGIC;  --input clock
10      button  : IN  STD_LOGIC;  --input signal to be debounced
11      result  : OUT STD_LOGIC); --debounced signal
12  END debounce;
13
14  ARCHITECTURE logic OF debounce IS
15    SIGNAL flipflops   : STD_LOGIC_VECTOR(1 DOWNTO 0); --input flip flops
16    SIGNAL counter_set : STD_LOGIC;                    --sync reset to zero
17    SIGNAL counter_out : STD_LOGIC_VECTOR(counter_size DOWNTO 0) := (OTHERS =>
        '0'); --counter output
18  BEGIN
19
20    counter_set <= flipflops(0) xor flipflops(1);   --determine when to start/
        reset counter
21
22    PROCESS(clk)
23    BEGIN
24      IF(clk'EVENT and clk = '1') THEN
25        flipflops(0) <= button;
26        flipflops(1) <= flipflops(0);
27        If(counter_set = '1') THEN                  --reset counter because
            input is changing
```

```
28          counter_out <= (OTHERS => '0');
29        ELSIF(counter_out(counter_size) = '0') THEN --stable input time is not
              yet met
30          counter_out <= counter_out + 1;
31        ELSE                                        --stable input time is met
32          result <= flipflops(1);
33        END IF;
34      END IF;
35    END PROCESS;
36  END logic;
```

Codice Componente 5.5: Definizione del componente debounce

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity shift_circular_register_debounced is
5      Port ( clock : in  STD_LOGIC;
6             button : in  STD_LOGIC;
7             load : in  STD_LOGIC;
8             init : in  STD_LOGIC_VECTOR (7 downto 0);
9             y : out  STD_LOGIC_VECTOR (7 downto 0));
10 end shift_circular_register_debounced;
11
12 architecture Behavioral of shift_circular_register_debounced is
13
14 component debounce
15 GENERIC(counter_size : INTEGER := 19); --counter size (19 bits gives 10.5ms
       with 50MHz clock)
16   Port(
17       clk : IN STD_LOGIC; --input clock
18       button : IN STD_LOGIC; --input signal to be debounced
19       result : OUT STD_LOGIC); --debounced signal
20 end component;
21
22 component scr8_sintesi
23     Port ( clock : in  STD_LOGIC;
24            button : in  STD_LOGIC;
25            init : in  STD_LOGIC_VECTOR (7 downto 0);
26            load : in  STD_LOGIC;
27            y : out  STD_LOGIC_VECTOR (7 downto 0));
28 end component;
29
30 signal b : STD_LOGIC;
31
32 begin
33
34 deb : debounce
35   Port Map(
```

```
36            clk => clock,
37           button => button,
38           result => b
39           );
40
41 reg : scr8_sintesi
42     Port Map(
43             clock => clock,
44               button => b,
45               init => init,
46               load => load,
47               y => y
48           );
49
50 end Behavioral;
```

Codice Componente 5.6: Definizione del componente $\text{shift}_c ircular_r egister_d ebounced$

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY shift_circular_register_tb IS
5  END shift_circular_register_tb;
6
7  ARCHITECTURE behavior OF shift_circular_register_tb IS
8
9      -- Component Declaration for the Unit Under Test (UUT)
10
11     COMPONENT shift_circular_register
12     PORT(
13          clk : IN  std_logic;
14          button : IN  std_logic;
15          init : IN  std_logic_vector(0 to 7);
16          y : OUT  std_logic_vector(0 to 7)
17         );
18     END COMPONENT;
19
20
21    --Inputs
22    signal clk : std_logic := '0';
23    signal button : std_logic := '0';
24    signal init : std_logic_vector(0 to 7) := (others => '0');
25
26   --Outputs
27    signal y : std_logic_vector(0 to 7);
28
29    -- Clock period definitions
30    constant clk_period : time := 1 ns;
31
```

```
32 BEGIN
33
34   -- Instantiate the Unit Under Test (UUT)
35    uut: shift_circular_register PORT MAP (
36           clk => clk,
37           button => button,
38           init => init,
39           y => y
40         );
41
42    -- Clock process definitions
43    clk_process :process
44    begin
45     clk <= '0';
46     wait for clk_period/2;
47     clk <= '1';
48     wait for clk_period/2;
49    end process;
50
51
52    -- Stimulus process
53    stim_proc: process
54    begin
55       -- hold reset state for 100 ns.
56       wait for 100 ns;
57
58       wait for clk_period*10;
59
60       -- insert stimulus here
61
62       button <= '1';
63
64       wait for 1 ns;
65
66       button <= '0';
67
68       wait for 4 ns;
69       button <= '1';
70
71       wait for 1 ns;
72
73       button <= '0';
74
75       wait for 4 ns;
76       button <= '1';
77
78       wait for 1 ns;
79
80       button <= '0';
```

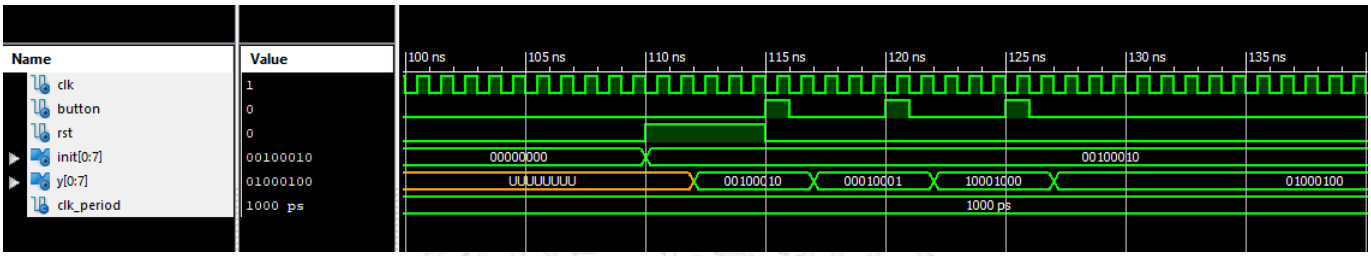
```
81
82          wait for 4 ns;
83
84
85
86
87
88          wait;
89      end process;
90
91  END;
```

Codice Componente 5.7: Definizione del componente shift$_c ircular_r egister_t b$

```
1  et_mazzeo - traccia n° 3
```

Codice Componente 5.8: Definizione del componente et$_m azzeo$

## 5.3 Simulazione




Si nota come su ogni fronte di discesa del clock successivo all'innalzamento dell'ingresso button lo shift register, se il reset è basso, shifta le cifre verso destra.

## 5.4 Sintesi su board FPGA

Per interfacciare il design realizzato con l'I/O di board sono state utilizzate le seguenti locazioni:

Per il clock si è utilizzato il clock della scheda

```
## Clock signal
NET "clock"    LOC = "E3"   | IOSTANDARD = "LVCMOS33";
```

Per l'inizializzazione e per l'ingresso di shift sono stati utilizzati due dei bottoni presenti sulla scheda

```
## Buttons
NET "load"            LOC=M17 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_D15_14
NET "button"            LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14
```

Per mostrare l'evolversi dell'uscita dello shift register sono stati utilizzati gli 8 led più a destra sulla board

```
## LEDs
NET "y<0>"          LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
NET "y<1>"          LOC=K15 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_RS1_15
NET "y<2>"          LOC=J13 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A25_15
NET "y<3>"          LOC=N14 | IOSTANDARD=LVCMOS33; #IO_L8P_T1_D11_14
NET "y<4>"          LOC=R18 | IOSTANDARD=LVCMOS33; #IO_L7P_T1_D09_14
NET "y<5>"          LOC=V17 | IOSTANDARD=LVCMOS33; #IO_L18N_T2_A11_D27_14
NET "y<6>"          LOC=U17 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A14_D30_14
NET "y<7>"          LOC=U16 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A12_D28_14
```

Per inizializzare il registro sono stati utilizzati gli 8 switch più a destra della board

```
## Switches
NET "init<0>"           LOC=J15 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_RS0_15
NET "init<1>"           LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "init<2>"           LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
NET "init<3>"           LOC=R15 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_14
NET "init<4>"           LOC=R17 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_14
NET "init<5>"           LOC=T18 | IOSTANDARD=LVCMOS33; #IO_L7N_T1_D10_14
NET "init<6>"           LOC=U18 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A13_D29_14
NET "init<7>"           LOC=R13 | IOSTANDARD=LVCMOS33; #IO_L5N_T0_D07_14
```

# Capitolo 6

# Esercizio 6

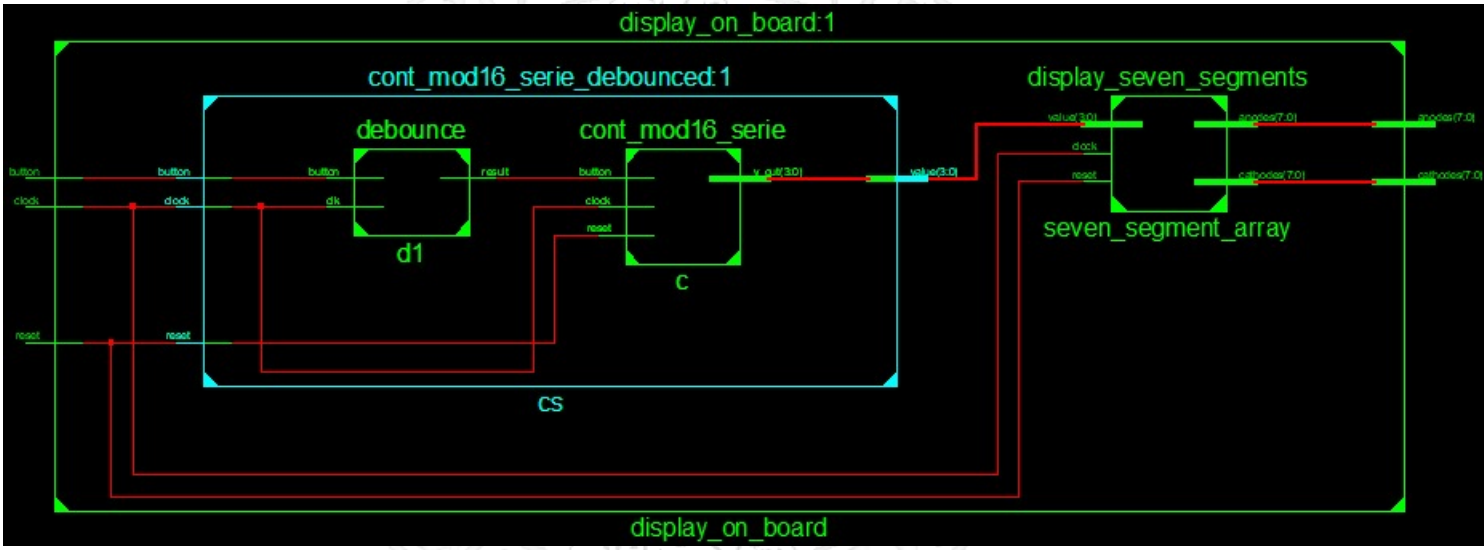
## 6.1 Traccia

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 1 cifra del display per visualizzare il contenuto del registro. Si noti che il segnale corrispondente al bottone deve essere acquisito in maniera sincrona con un clock.

## 6.2 Soluzione

### 6.2.1 Schematici

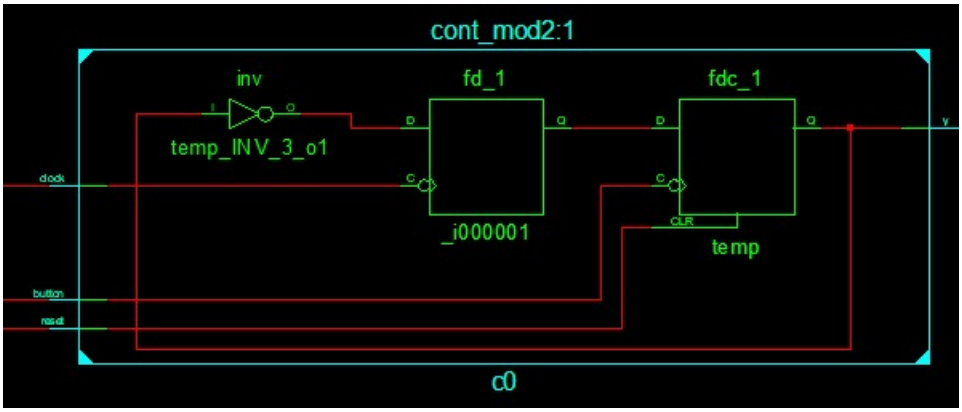
- Schematico generale




Il contatore è progettato utilizzando due componenti che verranno successivamente analizzati nel dettaglio chiamati debounce, che hanno il compito di eliminare il debouncing del bottone nel momento in cui questo passa dal valore basso a quello alto (che corrisponde alla pressione del bottone sull'FPGA). Questo viene fatto appositamente per evitare che una pressione del bottone possa contare più di un valore.

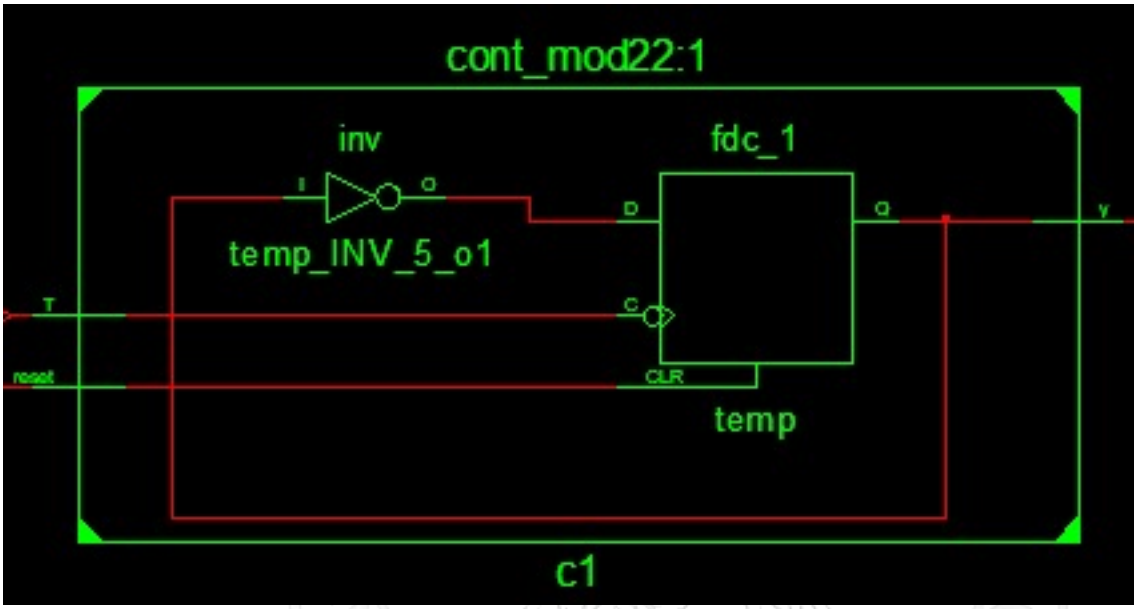
Il sistema completo è cosi composto da un contatore modulo 16 nella configurazione serie, 2 componenti debounce e il display_seven_segment necessario per inizializzare anodi e catodi per mostrare sul display della scheda il risultato del conteggio ad ogni passo.

- Schematico cont_mod2




Classico contatore modulo 2, progettato col funzionamento di un flip flop T a commutazione.

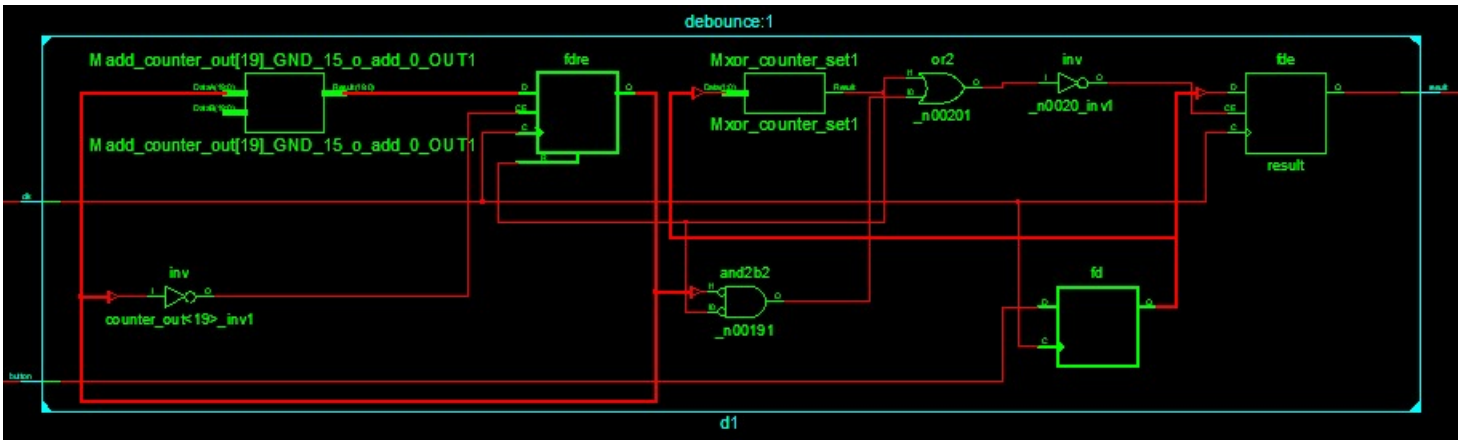
- Schematico cont_mod22




Questo componente differisce dal precedente perchè non ha il clock in ingresso, è utilizzato per i 3 contatori a valle del primo nella struttura composta da 4 contatori modulo 2 per formare un modulo 16.

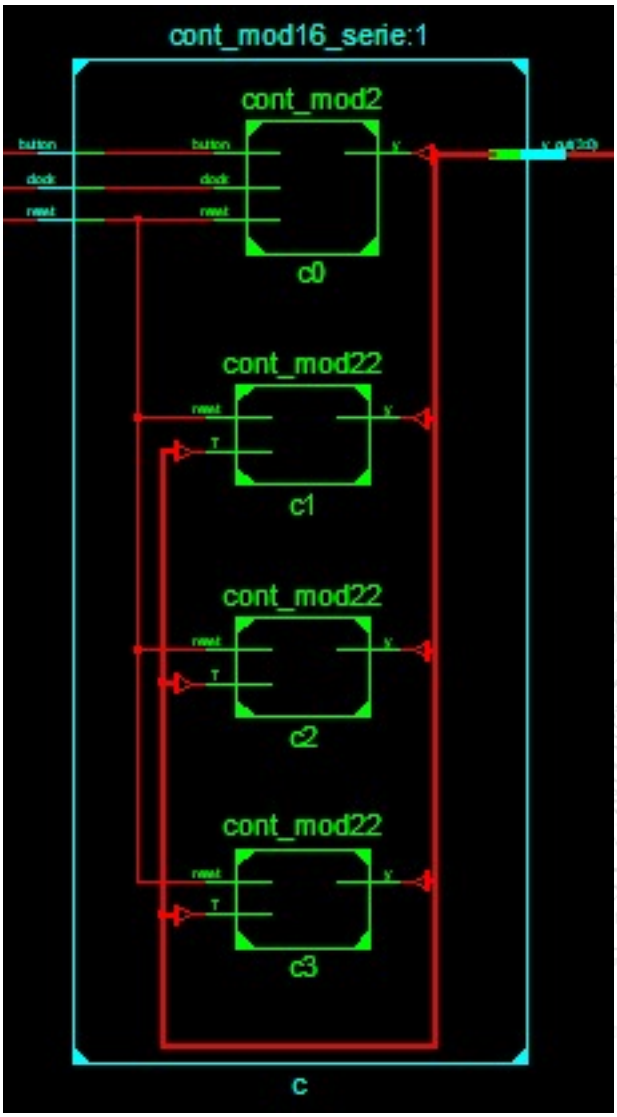
La sua uscita varia soltanto al variare dell'ingresso t, che sarà collegato all'uscita del contatore a monte.

- Schematico debounce




Il cuore di questo componente è un contatore, che ha il compito di fare da divisore di frequenza per far si che il segnale di conteggio venga modificato soltanto dopo che l'effetto causato dal debouncing sia terminato.

- Schematico cont_mod16_serie



Progettato a partire da un contatoremod_2 e 3 contatori_mod22, poichè il clock va in ingresso soltanto al primo componente, e l'uscita del componente precedente diventa il segnale di abilitazione per il successivo.

## 6.2.2 Codice

### 6.2.2.1 Contatore_mod_16

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity cont_mod2 is
5      Port ( button : in  STD_LOGIC;
6             clock : in  STD_LOGIC;
7             reset : in  STD_LOGIC;
8             y : out  STD_LOGIC);
9  end cont_mod2;
10
11 architecture Behavioral of cont_mod2 is
12
13 signal temp : STD_LOGIC := '0';
14
15 begin
16
17 y <= temp;
18
19 c : process(clock, reset, button)
20 begin
21
22 if reset = '1' then
23   temp <= '0';
24 elsif (button'event and button = '0') then
25     if (clock'event and clock = '0') then
26     temp <= not temp;
27     end if;
28 end if;
29
30 end process;
31
32 end Behavioral;
```

Codice Componente 6.1: Definizione del componente $cont_mod_2$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity cont_mod22 is
5      Port ( T : in  STD_LOGIC;
6             reset : in  STD_LOGIC;
```

```
7           y : out  STD_LOGIC);
8  end cont_mod22;
9
10 architecture Behavioral of cont_mod22 is
11
12 signal temp : STD_LOGIC := '0';
13
14 begin
15
16 y <= temp;
17
18 c : process(T, reset)
19 begin
20
21 if reset = '1' then
22   temp <= '0';
23 elsif T'event and T = '0' then
24     temp <= not temp;
25 end if;
26
27 end process;
28
29 end Behavioral;
```

Codice Componente 6.2: Definizione del componente cont$_m$od22

```
1
2  entity cont_mod16_serie is
3      Port ( clock : in  STD_LOGIC;
4             reset : in  STD_LOGIC;
5             button : in  STD_LOGIC;
6             y_out : out  STD_LOGIC_VECTOR (3 downto 0));
7  end cont_mod16_serie;
8
9  architecture Structural of cont_mod16_serie is
10
11 component cont_mod2
12     Port ( button : in  STD_LOGIC;
13            clock : in  STD_LOGIC;
14            reset : in  STD_LOGIC;
15            y : out  STD_LOGIC);
16 end component;
17
18 component cont_mod22
19     Port ( T : in  STD_LOGIC;
20            reset : in  STD_LOGIC;
21            y : out  STD_LOGIC);
22 end component;
23
```

```
24 signal y_temp : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
25
26 begin
27
28 c0 : cont_mod2
29 Port Map ( button => button,
30            clock => clock,
31            reset => reset,
32            y => y_temp(0));
33
34 c1 : cont_mod22
35 Port Map ( T => y_temp(0),
36            reset => reset,
37            y => y_temp(1));
38
39 c2 : cont_mod22
40 Port Map ( T => y_temp(1),
41            reset => reset,
42            y => y_temp(2));
43
44 c3 : cont_mod22
45 Port Map ( T => y_temp(2),
46            reset => reset,
47            y => y_temp(3));
48
49 y_out <= y_temp;
50
51 end Structural;
```

Codice Componente 6.3: Definizione del componente $\text{cont}_m od16_s erie$

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity cont_mod16_serie_debounced is
6      Port ( clock : in  STD_LOGIC;
7             reset : in  STD_LOGIC;
8             button : in  STD_LOGIC;
9             value : out  STD_LOGIC_VECTOR (3 downto 0));
10 end cont_mod16_serie_debounced;
11
12 architecture Structural of cont_mod16_serie_debounced is
13
14 component cont_mod16_serie
15     Port ( clock : in  STD_LOGIC;
16            reset : in  STD_LOGIC;
17            button : in  STD_LOGIC;
18            y_out : out  STD_LOGIC_VECTOR (3 downto 0));
```

```
19 end component;
20
21 component debounce
22 GENERIC(counter_size : INTEGER := 19); --counter size (19 bits gives 10.5ms
       with 50MHz clock)
23 Port( clk : IN STD_LOGIC; --input clock
24      button : IN STD_LOGIC; --input signal to be debounced
25      result : OUT STD_LOGIC); --debounced signal
26 end component;
27
28 signal b : STD_LOGIC;
29 signal r : STD_LOGIC;
30
31 begin
32
33 c : cont_mod16_serie
34 Port Map ( clock => clock,
35         reset => reset,
36         button => b,
37         y_out => value);
38
39 d1 : debounce
40 Port Map ( clk => clock,
41         button => button,
42         result => b);
43
44 end Structural;
```

Codice Componente 6.4: Definizione del componente $\text{cont}_m od16_s erie_d ebounced$

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity cont_mod16_parallelo is
6     Port ( clock : in  STD_LOGIC;
7            reset : in  STD_LOGIC;
8            button : in  STD_LOGIC;
9            y_out : out  STD_LOGIC_VECTOR (3 downto 0));
10 end cont_mod16_parallelo;
11
12 architecture Structural of cont_mod16_parallelo is
13
14 component cont_mod2
15     Port ( button : in  STD_LOGIC;
16            clock : in  STD_LOGIC;
17            reset : in  STD_LOGIC;
18            y : out  STD_LOGIC);
19 end component;
```

```
20
21  component cont_mod2_parallelo
22      Port ( button : in  STD_LOGIC;
23             clock : in  STD_LOGIC;
24             reset : in  STD_LOGIC;
25           x : in STD_LOGIC;
26             y : out  STD_LOGIC);
27  end component;
28
29  signal y_temp : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
30  signal a : STD_LOGIC;
31  signal b : STD_LOGIC;
32
33  begin
34
35  a <= y_temp(0) and y_temp(1);
36  b <= a and y_temp(2);
37
38  c0 : cont_mod2
39  Port Map ( button => button,
40             clock => clock,
41             reset => reset,
42             y => y_temp(0));
43
44  c1 : cont_mod2_parallelo
45  Port Map ( button => button,
46         clock => clock,
47             reset => reset,
48         x => y_temp(0),
49             y => y_temp(1));
50
51  c2 : cont_mod2_parallelo
52  Port Map ( button => button,
53         clock => clock,
54             reset => reset,
55         x => a,
56             y => y_temp(2));
57
58  c3 : cont_mod2_parallelo
59  Port Map ( button => button,
60         clock => clock,
61             reset => reset,
62         x => b,
63             y => y_temp(3));
64
65  y_out <= y_temp;
66
67  end Structural;
```

Codice Componente 6.5: Definizione del componente $cont_mod16_parallelo$

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

entity cont_mod16_parallelo_debounced is
    Port ( clock : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           button : in  STD_LOGIC;
           value : out  STD_LOGIC_VECTOR (3 downto 0));
end cont_mod16_parallelo_debounced;

architecture Structural of cont_mod16_parallelo_debounced is

component cont_mod16_parallelo
    Port ( clock : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           button : in  STD_LOGIC;
           y_out : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

component debounce
GENERIC(counter_size : INTEGER := 19); --counter size (19 bits gives 10.5ms
    with 50MHz clock)
Port(  clk : IN STD_LOGIC; --input clock
     button : IN STD_LOGIC; --input signal to be debounced
     result : OUT STD_LOGIC); --debounced signal
end component;

signal b : STD_LOGIC;

begin

c : cont_mod16_parallelo
Port Map ( clock => clock,
       reset => reset,
       button => b,
       y_out => value);

d1 : debounce
Port Map ( clk => clock,
       button => button,
       result => b);

end Structural;
```

Codice Componente 6.6: Definizione del componente $cont_mod16_parallelo_debounced$

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

entity anodes_manager is
    Port ( enable_digit : in  STD_LOGIC;
           anodes : out  STD_LOGIC_VECTOR (7 downto 0)
         );
end anodes_manager;

architecture Behavioral of anodes_manager is

begin

anodes(0) <= enable_digit; -- mi basta una sola cifra esadecimale per
   mostrare l'output modulo 16
anodes ( 7 downto 1) <= (others => '1');

end Behavioral;
```

Codice Componente 6.7: Definizione del componente $\text{anodes}_{m}anager$

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

use IEEE.NUMERIC_STD.ALL;

entity cathodes_manager is
    Port ( value : in  STD_LOGIC_VECTOR (3 downto 0); -- valore di conteggio
        rappresentato su 4 bit
           cathodes : out  STD_LOGIC_VECTOR (7 downto 0));
end cathodes_manager;

architecture Behavioral of cathodes_manager is

constant zero   : std_logic_vector(6 downto 0) := "1000000";
constant one    : std_logic_vector(6 downto 0) := "1111001";
constant two    : std_logic_vector(6 downto 0) := "0100100";
constant three  : std_logic_vector(6 downto 0) := "0110000";
constant four   : std_logic_vector(6 downto 0) := "0011001";
constant five   : std_logic_vector(6 downto 0) := "0010010";
constant six    : std_logic_vector(6 downto 0) := "0000010";
constant seven  : std_logic_vector(6 downto 0) := "1111000";
constant eight  : std_logic_vector(6 downto 0) := "0000000";
constant nine   : std_logic_vector(6 downto 0) := "0010000";
constant a     : std_logic_vector(6 downto 0) := "0001000";
constant b      : std_logic_vector(6 downto 0) := "0000011";
constant c      : std_logic_vector(6 downto 0) := "1000110";
constant d      : std_logic_vector(6 downto 0) := "0100001";
constant e      : std_logic_vector(6 downto 0) := "0000110";
```

```
28 constant f      : std_logic_vector(6 downto 0) := "0001110";
29
30
31 signal cathodes_for_digit : std_logic_Vector(6 downto 0) := (others => '0');
32
33 begin
34
35 seven_segment_decoder_process: process(value)
36   begin
37     case value is
38       when "0000" => cathodes_for_digit <= zero;
39       when "0001" => cathodes_for_digit <= one;
40       when "0010" => cathodes_for_digit <= two;
41       when "0011" => cathodes_for_digit <= three;
42       when "0100" => cathodes_for_digit <= four;
43       when "0101" => cathodes_for_digit <= five;
44       when "0110" => cathodes_for_digit <= six;
45       when "0111" => cathodes_for_digit <= seven;
46       when "1000" => cathodes_for_digit <= eight;
47       when "1001" => cathodes_for_digit <= nine;
48       when "1010" => cathodes_for_digit <= a;
49       when "1011" => cathodes_for_digit <= b;
50       when "1100" => cathodes_for_digit <= c;
51       when "1101" => cathodes_for_digit <= d;
52       when "1110" => cathodes_for_digit <= e;
53       when "1111" => cathodes_for_digit <= f;
54     when others => cathodes_for_digit <= (others => '0');
55     end case;
56   end process seven_segment_decoder_process;
57
58 cathodes (6 downto 0) <= cathodes_for_digit;
59 cathodes(7) <= '1';
60
61 end Behavioral;
```

Codice Componente 6.8: Definizione del componente cathodes$_m$*anager*

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity display_seven_segments is
5     Port ( clock : in  STD_LOGIC;
6            reset : in  STD_LOGIC;
7            value : in  STD_LOGIC_VECTOR (3 downto 0);
8            anodes : out  STD_LOGIC_VECTOR (7 downto 0);
9            cathodes : out  STD_LOGIC_VECTOR (7 downto 0));
10 end display_seven_segments;
11
12 architecture Structural of display_seven_segments is
```

```
13
14
15 COMPONENT cathodes_manager
16   PORT(
17     value : IN std_logic_vector(3 downto 0);
18     cathodes : OUT std_logic_vector(7 downto 0)
19     );
20 END COMPONENT;
21
22 COMPONENT anodes_manager
23   PORT(
24     enable_digit : IN std_logic;
25     anodes : OUT std_logic_vector(7 downto 0)
26     );
27 END COMPONENT;
28
29 begin
30
31
32 cathodes_instance: cathodes_manager port map(
33   value => value,
34   cathodes => cathodes
35 );
36
37 anodes_instance: anodes_manager port map(
38   enable_digit => '0',
39   anodes => anodes
40 );
41
42 end Structural;
```

Codice Componente 6.9: Definizione del componente $display_seven_segment$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity display_on_board is
5    Port(
6        clock : in  STD_LOGIC;
7        reset : in  STD_LOGIC;
8        button : in  STD_LOGIC;
9        anodes : out  STD_LOGIC_VECTOR (7 downto 0);
10       cathodes : out  STD_LOGIC_VECTOR (7 downto 0)
11        );
12
13 end display_on_board;
14
15 architecture Structural of display_on_board is
16
```

```
17 COMPONENT display_seven_segments
18   PORT(
19     clock : IN std_logic;
20     reset : IN std_logic;
21     value : IN std_logic_vector(3 downto 0);
22     anodes : OUT std_logic_vector(7 downto 0);
23     cathodes : OUT std_logic_vector(7 downto 0)
24     );
25 END COMPONENT;
26 
27 component cont_mod16_serie_debounced
28     Port ( clock : in  STD_LOGIC;
29            reset : in  STD_LOGIC;
30            button : in  STD_LOGIC;
31            value : out  STD_LOGIC_VECTOR (3 downto 0));
32 end component;
33 
34 signal cu_value : std_logic_vector(3 downto 0);
35 
36 begin
37 
38 seven_segment_array: display_seven_segments
39   PORT MAP(
40     clock => clock,
41     reset => reset,
42     value => cu_value,
43     anodes => anodes,
44     cathodes => cathodes
45 );
46 
47 cs: cont_mod16_serie_debounced
48 Port Map(  clock => clock,
49            reset => reset,
50            button => button,
51            value => cu_value
52       );
53 
54 end Structural;
```

Codice Componente 6.10: Definizione del componente display_onboard

```
1 debounce - traccia n° 5
```

Codice Componente 6.11: Definizione del componente debounce

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 
4 
5 ENTITY cont_mod_16_parallelo_tb IS
```

```
6  END cont_mod_16_parallelo_tb;
7
8  ARCHITECTURE behavior OF cont_mod_16_parallelo_tb IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11
12     COMPONENT cont_mod_16_parallelo
13     PORT(
14          clk : IN  std_logic;
15          rst : IN  std_logic;
16          button : IN  std_logic;
17          y : OUT  std_logic_vector(3 downto 0)
18         );
19     END COMPONENT;
20
21
22    --Inputs
23    signal clk : std_logic := '0';
24    signal rst : std_logic := '0';
25    signal button : std_logic := '0';
26   --signal u : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
27 --signal x : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
28
29   --Outputs
30    signal y : std_logic_vector(3 downto 0);
31
32    -- Clock period definitions
33    constant clk_period : time := 0.5 ns;
34
35 BEGIN
36
37   -- Instantiate the Unit Under Test (UUT)
38    uut: cont_mod_16_parallelo PORT MAP (
39           clk => clk,
40           rst => rst,
41           button => button,
42           y => y
43         );
44
45    -- Clock process definitions
46    clk_process :process
47    begin
48     clk <= '0';
49     wait for clk_period/2;
50     clk <= '1';
51     wait for clk_period/2;
52    end process;
53
54
```

```
55    -- Stimulus process
56    stim_proc: process
57    begin
58
59   --x(0) <= u(0);
60  --x(1) <= u(0) and u(1);
61  --x(2) <= u(0) and u(1) and u(2);
62
63        -- hold reset state for 100 ns.
64        wait for 100 ns;
65
66        wait for clk_period*10;
67
68        -- insert stimulus here
69
70      button <= '1';
71        wait for 1 ns;
72
73        button <= '0';
74        wait for 4 ns;
75
76        button <= '1';
77        wait for 1 ns;
78
79        button <= '0';
80        wait for 4 ns;
81
82        rst <= '1';
83
84        button <= '1';
85        wait for 1 ns;button <= '0';
86        wait for 4 ns;
87
88        button <= '1';
89        wait for 1 ns;button <= '0';
90        wait for 4 ns;
91
92        button <= '1';
93        wait for 1 ns;button <= '0';
94        wait for 4 ns;
95
96        button <= '1';
97        wait for 1 ns;button <= '0';
98        wait for 4 ns;
99
100       button <= '1';
101       wait for 1 ns;button <= '0';
102       wait for 4 ns;
103
```

```
104       button <= '1';
105       wait for 1 ns;button <= '0';
106       wait for 4 ns;
107 
108       button <= '1';
109       wait for 1 ns;button <= '0';
110       wait for 4 ns;
111 
112       button <= '1';
113       wait for 1 ns;
114 
115       wait;
116    end process;
117 
118 END;
```

Codice Componente 6.12: Definizione del componente $\text{cont}_m od_1 6_p arallelo_t b$

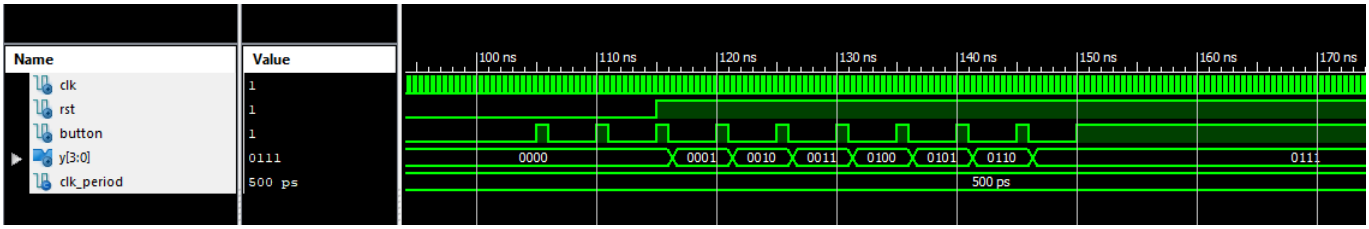
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  
4  ENTITY cont_mod_16_serie_tb IS
5  END cont_mod_16_serie_tb;
6  
7  ARCHITECTURE behavior OF cont_mod_16_serie_tb IS
8  
9      -- Component Declaration for the Unit Under Test (UUT)
10 
11     COMPONENT cont_mod_16_serie
12     PORT(
13          clk : IN  std_logic;
14          rst : IN  std_logic;
15          button : IN  std_logic;
16          y : OUT  std_logic_vector(3 downto 0)
17         );
18     END COMPONENT;
19 
20 
21    --Inputs
22    signal clk : std_logic := '0';
23    signal rst : std_logic := '0';
24    signal button : std_logic := '0';
25 
26   --Outputs
27    signal y : std_logic_vector(3 downto 0);
28 
29    -- Clock period definitions
30    constant clk_period : time := 1 ns;
31 
```

```
32 BEGIN
33
34   -- Instantiate the Unit Under Test (UUT)
35    uut: cont_mod_16_serie PORT MAP (
36           clk => clk,
37           rst => rst,
38           button => button,
39           y => y
40         );
41
42    -- Clock process definitions
43    clk_process :process
44    begin
45     clk <= '0';
46     wait for clk_period/2;
47     clk <= '1';
48     wait for clk_period/2;
49    end process;
50
51
52    -- Stimulus process
53    stim_proc: process
54    begin
55       -- hold reset state for 100 ns.
56       wait for 100 ns;
57
58       wait for clk_period*10;
59
60       -- insert stimulus here
61
62
63     button <= '1';
64       wait for 10 ns;
65
66       button <= '0';
67       wait for 5 ns;
68
69       button <= '1';
70       wait for 5 ns;
71
72       button <= '0';
73       wait for 5 ns;
74
75       rst <= '1';
76
77       button <= '1';
78       wait for 5 ns;button <= '0';
79       wait for 5 ns;
80
```

```
81        button <= '1';
82        wait for 5 ns;button <= '0';
83        wait for 5 ns;
84
85        button <= '1';
86        wait for 5 ns;button <= '0';
87        wait for 5 ns;
88
89        button <= '1';
90        wait for 5 ns;
91
92        button <= '0';
93        wait for 5 ns;
94
95        button <= '1';
96        wait for 5 ns;
97
98        button <= '0';
99        wait for 5 ns;
100
101       button <= '1';
102       wait for 5 ns;
103
104       button <= '0';
105       wait for 5 ns;
106
107       button <= '1';
108       wait for 5 ns;
109
110       button <= '0';
111       wait for 5 ns;
112
113       button <= '1';
114       wait for 5 ns;
115
116       wait;
117    end process;
118
119 END;
```

Codice Componente 6.13: Definizione del componente $\text{cont}_m od_1 6_s erie_t b$

## 6.3 Simulazione




Viene proposta la simulazione soltanto nella configurazione parallelo, essendo input e output analoghi.

Si nota come, quando il segnale di reset si alza (in logica negata), il conteggio avanza ad ogni fronte di discesa del bottone.

## 6.4 Sintesi su board FPGA

Per interfacciare il design realizzato con l'I/O di board sono state fatte le seguenti scelte:

Per il clock si è deciso di utilizzare il clock della scheda.

```
## Clock signal
NET "clock"    LOC = "E3"   | IOSTANDARD = "LVCMOS33";
```

Per il conteggio è stato utilizzato un bottone "button" per l'avanzamento dell'output del contatore, e un bottone "reset" per riportare il valore dell'uscita a 0.

```
## Buttons
NET "button"          LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14
NET "reset"          LOC=M17 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_D15_14
```

Per il display sono stati utilizzati tutti gli anodi e i catodi.

```
## 7 segment display
NET "cathodes<0>"             LOC=T10 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_A00_D16_14
NET "cathodes<1>"             LOC=R10 | IOSTANDARD=LVCMOS33; #IO_25_14
NET "cathodes<2>"             LOC=K16 | IOSTANDARD=LVCMOS33; #IO_25_15
NET "cathodes<3>"             LOC=K13 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A26_15
NET "cathodes<4>"             LOC=P15 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_14
NET "cathodes<5>"             LOC=T11 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A10_D26_14
NET "cathodes<6>"             LOC=L18 | IOSTANDARD=LVCMOS33; #IO_L4P_T0_D04_14
NET "cathodes<7>"             LOC=H15 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_A21_VREF_15

NET "anodes<0>"          LOC=J17 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_FOE_B_15
NET "anodes<1>"          LOC=J18 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_FWE_B_15
NET "anodes<2>"          LOC=T9 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_A01_D17_14
NET "anodes<3>"          LOC=J14 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A22_15
NET "anodes<4>"          LOC=P14 | IOSTANDARD=LVCMOS33; #IO_L8N_T1_D12_14
NET "anodes<5>"          LOC=T14 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_14
NET "anodes<6>"          LOC=K2 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_35
NET "anodes<7>"          LOC=U13 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_A02_D18_14
```

# Capitolo 7

# Esercizio 7

## 7.1 Traccia

Progettare ed implementare in VHDL un "arbitro 2 su 3", ossia un componente che, presi tre input binari in ingresso A1, A2 e A3, fornisce in uscita un valore binario U 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.

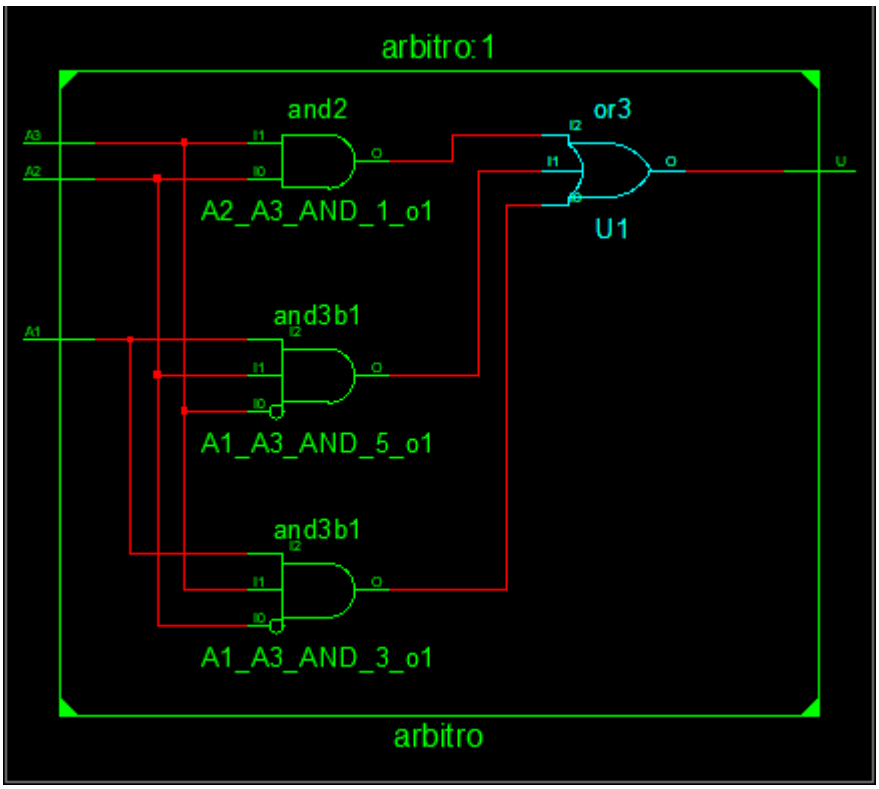
## 7.2 Soluzione

In allegato la tabella di verità per la soluzione del problema.

| A0 | A1 | A2 | Y |
|---|---|---|---|
| 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 |

Y=A0'A1A2+A0A1'A2+A0A1A2'+A0A1A2=
=A1A2(A0+A0')+A0A1'A2+A0A1A2'=
=A1A2+A0A1'A2+A0A1A2'

### 7.2.1 Schematici




Lo schematico implementa l'equazione dell'uscita Y calcolata a partire dalla tabella di verità precedentemente.

Rappresentazione puramente dataflow, quindi la sintesi è fatta con le porte logiche definite dal codice.

### 7.2.2 Codice

#### 7.2.2.1 Arbitro 2 su 3

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

```
17 U <= (A2 AND A3) OR (A1 AND (NOT A2) AND A3) OR (A1 AND A2 AND (NOT A3)); --
       equazione ricavata precedentemente
18
19
20 end dataflow;
```

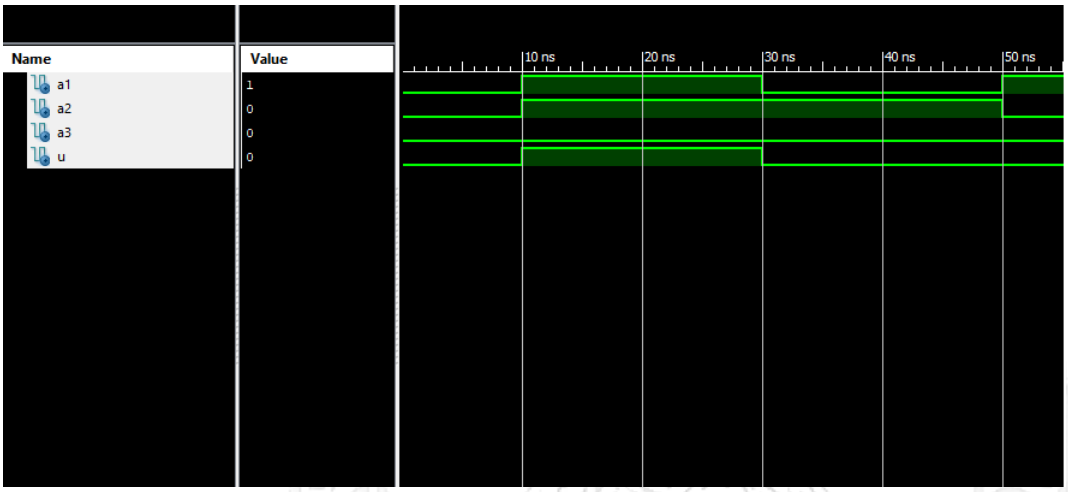
Codice Componente 7.1: Definizione del componente arbitro

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.all;
4
5  entity arbitro_testbench is
6  end arbitro_testbench;
7
8  architecture behavioral of arbitro_testbench is
9
10   component arbitro is
11   Port( 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 signal a1 : STD_LOGIC:='0';
19 signal a2 :  STD_LOGIC:='0';
20 signal a3 :  STD_LOGIC:='0';
21 signal u:  STD_LOGIC:='0';
22
23 begin
24
25   uut: arbitro Port map(a1,a2,a3,u);
26
27   stim_proc: process
28   begin
29
30   wait for 10 ns;
31   a1 <= '1';
32     a2 <= '1';
33     a3 <='0';
34   wait for 20 ns;
35
36     assert u='1'
37     report "errore"
38     severity failure;
39
40     a1 <= '0';
41     a2 <= '1';
```

```
42      a3 <='0';
43    wait for 20 ns;
44
45      assert u='0'
46      report "errore"
47      severity failure;
48
49      a1 <= '1';
50      a2 <= '0';
51      a3 <='0';
52    wait for 20 ns;
53
54      assert u='0'
55      report "errore"
56      severity failure;
57    wait;
58    end process;
59  end;
```

Codice Componente 7.2: Definizione del componente arbitro testbench

## 7.3 Simulazione




Dalla simulazione si nota come l'uscita sia alta quando almeno due ingressi sono alti (es. da 10 a 30 ns a1=a2=1 e a3=0), mentre si abbassa quando sono in maggior numero gli ingressi bassi (da 30 a 50 ns a1=a3=0 e a2=1).

## 7.4 Sintesi su board FPGA

Ai fini dello svolgimento dell'esercizio, sono stati utilizzati i seguenti pin di I/O presenti sulla board.

Per gli ingressi sono stati utilizzati i 3 switch più a destra della board.

```
## Switches

NET "A1"          LOC=J15 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_RS0_15
NET "A2"          LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "A3"          LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
```

Per l'uscita è stato utilizzato il led più a destra della board, acceso nel caso l'uscita è alta e spento in caso contrario.

```
## LEDs

 NET "U"          LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
```

# Capitolo 8

# Esercizio 8

## 8.1 Traccia

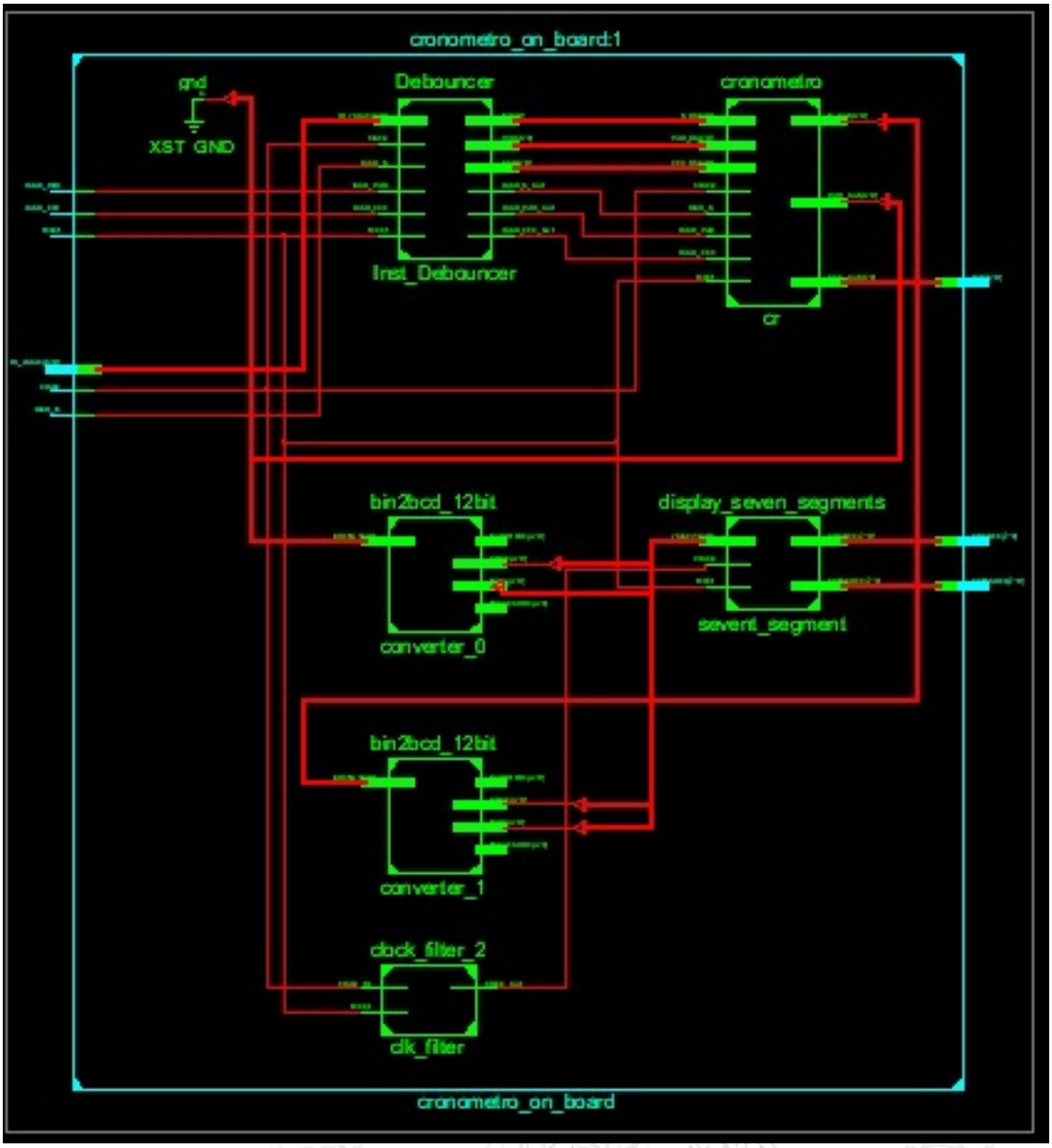
Progettare ed implementare un orologio che, a partire da un clock di riferimento che opera da base dei tempi di adeguata precisione, genera mediante uso di contatori il secondo, il minuto e l'ora. L' orologio deve essere sintetizzato su FPGA e la visualizzazione dell'ora deve sfruttare le 4 cifre del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

È stato realizzato un orologio seguendo un'architettura multi-livello. Al livello più alto si trova il top-level-module, ovvero il modulo cronometro (structural). Tale componente riceve in ingresso, oltre ai segnali di clock e reset, tre segnali di caricamento e tre valori, associati rispettivamente a secondi, minuti ed ore. In questo modo è possibile impostare l'orario desiderato. In uscita sono invece riportati i valori correnti di secondi, minuti ed ore. I componenti appartenenti al livello inferiore sono: clock_filter (behavioural): componente responsabile della divisione in frequenza del segnale di clock. Tale componente viene utilizzato come base dei tempi, fornendo in uscita un segnale avente un periodo pari ad 1 secondo. Counter_mod_M (behavioural): componente in grado di effettuare il conteggio modulo M con caricamento. All'interno del top-level-module sono istanziati tre contatori, in modo che i primi due, associati a secondi e minuti, contino modulo 60, mentre l'ultimo, associato alle ore, conti modulo 24. I tre contatori sono organizzati secondo un'architettura seriale. Il contatore dei secondi incrementa il conteggio sul fronte di salita del segnale in uscita dal clock_filter e, raggiunto il valore massimo, fornisce in uscita un segnale alto. I contatori di minuti e ore incrementano invece il conteggio sul fronte di salita del segnale di uscita del contatore precedente.

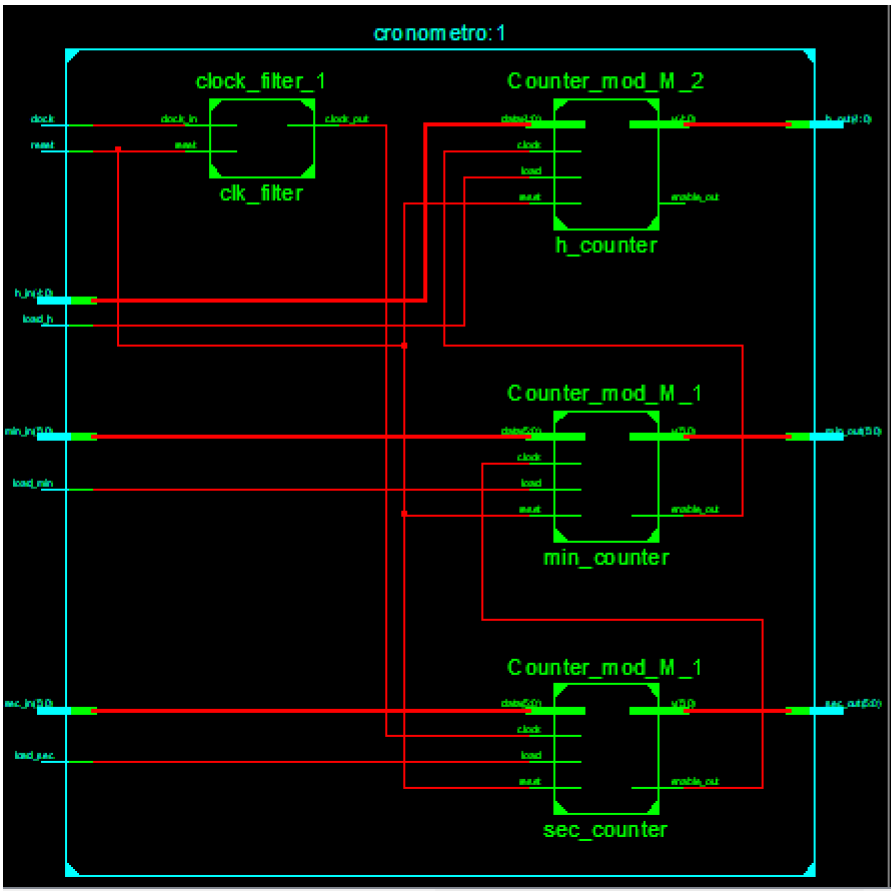
### 8.2.1 Schematici

- Schematico generale




Lo schematico generale è composto dal cronometro, da un debouncer, da convertitori e da un display_seven_segment per mostrare correttamente l'output in uscita sul display della board.

- Schematico cronometro




Il cronometro è composto da un clock_filter_crono, cha ha il compito di far avanzare il tempo nel modo corretto, e da 3 contatori con modulo coerente per la rappresentazione di ore, minuti e secondi.

## 8.2.2 Codice

### 8.2.2.1 Cronometro

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6
7  entity Counter_mod_M is
8  generic( M : integer := 60;
9        N : integer := 6
10 );
11 port( clock : in STD_LOGIC;
12     reset : in STD_LOGIC;
13     load : in STD_LOGIC;
14     data : in STD_LOGIC_VECTOR (N-1 downto 0);
15     y : out STD_LOGIC_VECTOR (N-1 downto 0);
16     enable_out : out STD_LOGIC
```

```
17 );
18 end Counter_mod_M;
19
20 architecture Behavioral of Counter_mod_M is
21
22 signal ty : STD_LOGIC_VECTOR (N-1 downto 0);
23
24 begin
25
26 count: process(clock, reset, load, data)
27 begin
28
29 if(reset = '1') then
30   ty <= (others => '0');
31   enable_out <= '0';
32 elsif(load = '1') then
33   if(conv_integer(data) > M-1) then
34     ty <= std_logic_vector(to_unsigned(M-1, ty' length));
35   else
36     ty <= data;
37   end if;
38   enable_out <= '0';
39 elsif(rising_edge(clock)) then
40   if(ty = std_logic_vector(to_unsigned(M-1, ty' length))) then
41     ty <= (others => '0');
42     enable_out <= '1';
43   else
44     ty <= ty + "1";
45     enable_out <= '0';
46   end if;
47 end if;
48
49 end process;
50
51 y <= ty;
52
53 end Behavioral;
```

Codice Componente 8.1: Definizione del componente Counter$_m od_M$

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity clock_filter is
5 generic(
6 clock_frequency_in : integer := 50000000;
7 clock_frequency_out : integer := 500
8 );
9 Port ( clock_in : in STD_LOGIC;
```

```
10      reset : in STD_LOGIC;
11      clock_out : out STD_LOGIC);
12  end clock_filter;
13
14  architecture Behavioral of clock_filter is
15
16  signal clockfx : std_logic := '0';
17
18  constant count_max_value : integer := clock_frequency_in/(
        clock_frequency_out)-1;
19
20  begin
21  clock_out <= clockfx;
22  count_for_division: process(clock_in, reset)
23
24  variable counter : integer range 0 to count_max_value := 0;
25
26  begin
27
28  if reset = '1' then
29    counter := 0;
30    clockfx <= '0';
31  elsif clock_in'event and clock_in = '1' then
32    if counter = count_max_value then
33      clockfx <= '1';
34      counter := 0;
35    else
36      clockfx <= '0';
37      counter := counter + 1;
38    end if;
39  end if;
40
41  end process;
42
43  end Behavioral;
```

Codice Componente 8.2: Definizione del componente $clock_filter$

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity cronometro is
5   Generic( clock_frequency_in : integer := 50000000;
6         clock_frequency_out : integer := 1
7       );
8
9   Port ( clock : in STD_LOGIC;
10        reset : in STD_LOGIC;
11        load_sec : in STD_LOGIC;
```

```
12      load_min : in STD_LOGIC;
13      load_h : in STD_LOGIC;
14      sec_in : in STD_LOGIC_VECTOR (5 downto 0);
15      min_in : in STD_LOGIC_VECTOR (5 downto 0);
16      h_in : in STD_LOGIC_VECTOR (4 downto 0);
17      sec_out : out STD_LOGIC_VECTOR (5 downto 0);
18      min_out : out STD_LOGIC_VECTOR (5 downto 0);
19      h_out : out STD_LOGIC_VECTOR (4 downto 0));
20  end cronometro;
21
22  architecture Structural of cronometro is
23
24  component Counter_mod_M
25  generic( M : integer;
26       N : integer
27  );
28  port( clock : in STD_LOGIC;
29      reset : in STD_LOGIC;
30      load : in STD_LOGIC;
31      data : in STD_LOGIC_VECTOR (N-1 downto 0);
32      y : out STD_LOGIC_VECTOR (N-1 downto 0);
33      enable_out : out STD_LOGIC
34  );
35  end component;
36
37  COMPONENT clock_filter
38  GENERIC(
39  clock_frequency_in : integer;
40  clock_frequency_out : integer
41  );
42  PORT( clock_in : IN std_logic;
43  reset : in STD_LOGIC;
44  clock_out : OUT std_logic
45  );
46  END COMPONENT;
47  signal enable_sec : STD_LOGIC;
48  signal enable_min : STD_LOGIC;
49  signal clock_min : STD_LOGIC;
50  signal clock_h : STD_LOGIC;
51  signal clock_fx : STD_LOGIC;
52  begin
53  clk_filter: clock_filter
54  GENERIC MAP(
55  clock_frequency_in => clock_frequency_in,
56  clock_frequency_out => clock_frequency_out
57  )
58  PORT MAP(
59      clock_in => clock,
60      reset => reset,
```

```
61      clock_out => clock_fx
62  );
63
64  clock_min <= enable_sec;
65  clock_h <= enable_min;
66
67  sec_counter: Counter_mod_M
68  generic map( M => 60,
69         N => 6
70  )
71  port map( clock => clock_fx,
72        reset => reset,
73        load => load_sec,
74        data => sec_in,
75        y => sec_out,
76        enable_out => enable_sec
77  );
78
79  min_counter: Counter_mod_M
80  generic map( M => 60,
81         N => 6
82  )
83  port map( clock => clock_min,
84       reset => reset,
85        load => load_min,
86        data => min_in,
87        y => min_out,
88        enable_out => enable_min
89  );
90
91  h_counter: Counter_mod_M
92  generic map( M => 24,
93         N => 5
94  )
95  port map( clock => clock_h,
96        reset => reset,
97       load => load_h,
98       data => h_in,
99       y => h_out
100 );
101
102 end Structural;
```

Codice Componente 8.3: Definizione del componente cronometro

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity Debouncer is
```

```
Port(   clock : in STD_LOGIC;
    reset : in STD_LOGIC;
    load_sec : in STD_LOGIC;
    load_min : in STD_LOGIC;
    load_h : in STD_LOGIC;
    in_value : in STD_LOGIC_VECTOR(5 downto 0);
    sec : out STD_LOGIC_VECTOR (5 downto 0);
    min : out STD_LOGIC_VECTOR (5 downto 0);
    h : out STD_LOGIC_VECTOR (4 downto 0);
    load_sec_out : out STD_LOGIC;
    load_min_out : out STD_LOGIC;
    load_h_out : out STD_LOGIC
);
end Debouncer;

architecture behavioral of Debouncer is

signal temp_sec : STD_LOGIC_VECTOR (5 downto 0) := (others => '0');
signal temp_min : STD_LOGIC_VECTOR (5 downto 0) := (others => '0');
signal temp_h : STD_LOGIC_VECTOR (4 downto 0) := (others => '0');

begin

sec <= temp_sec;
min <= temp_min;
h <= temp_h;

main: process(clock, reset)
begin
if reset = '1' then
  temp_sec <= (others => '0');
  temp_min <= (others => '0');
  temp_h <= (others => '0');
elsif clock'event and clock = '1' then
  if load_sec = '1' then
    temp_sec <= in_value;
    load_sec_out <= '1';
  elsif load_min = '1' then
    temp_min <= in_value;
    load_min_out <= '1';
  elsif load_h = '1' then
    temp_h <= in_value(4 downto 0);
    load_h_out <= '1';
  else
    load_sec_out <= '0';
    load_min_out <= '0';
    load_h_out <= '0';
  end if;
end if;
```

```
54
55 end process;
56
57 end Behavioral;
```

Codice Componente 8.4: Definizione del componente Debouncer

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6  use IEEE.numeric_std.all;
7
8  entity bin2bcd_12bit is
9  Port(   binIN : in STD_LOGIC_VECTOR (11 downto 0);
10     ones : out STD_LOGIC_VECTOR (3 downto 0);
11     tens : out STD_LOGIC_VECTOR (3 downto 0);
12     hundreds : out STD_LOGIC_VECTOR (3 downto 0);
13     thousands : out STD_LOGIC_VECTOR (3 downto 0)
14 );
15 end bin2bcd_12bit;
16
17 architecture Behavioral of bin2bcd_12bit is
18 begin
19
20 bcd1: process(binIN)
21 -- temporary variable
22 variable temp : STD_LOGIC_VECTOR (11 downto 0);
23
24 variable bcd : UNSIGNED (15 downto 0) := (others => '0');
25
26 begin
27 -- zero the bcd variable
28 bcd := (others => '0');
29 -- read input into temp variable
30 temp(11 downto 0) := binIN;
31 -- cycle 12 times as we have 12 input bits
32 -- this could be optimized, we do not need to check and add 3 for the
33 -- first 3 iterations as the number can never be >4
34 for i in 0 to 11 loop
35   if bcd(3 downto 0) > 4 then
36     bcd(3 downto 0) := bcd(3 downto 0) + 3;
37 end if;
38
39   if bcd(7 downto 4) > 4 then
40     bcd(7 downto 4) := bcd(7 downto 4) + 3;
41   end if;
42
```

```
43    if bcd(11 downto 8) > 4 then
44      bcd(11 downto 8) := bcd(11 downto 8) + 3;
45    end if;
46  -- thousands can't be >4 for a 12-bit input number
47  -- so don't need to do anything to upper 4 bits of bcd
48  -- shift bcd left by 1 bit, copy MSB of temp into LSB of bcd
49    bcd := bcd(14 downto 0) & temp(11);
50  -- shift temp left by 1 bit
51    temp := temp(10 downto 0) & '0';
52  end loop;
53  -- set outputs
54  ones <= STD_LOGIC_VECTOR(bcd(3 downto 0));
55  tens <= STD_LOGIC_VECTOR(bcd(7 downto 4));
56  hundreds <= STD_LOGIC_VECTOR(bcd(11 downto 8));
57  thousands <= STD_LOGIC_VECTOR(bcd(15 downto 12));
58
59  end process bcd1;
60
61  end Behavioral;
```

Codice Componente 8.5: Definizione del componente $\text{bin2bcd}_1 2bit$

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity anodes_manager is
5   Port(   counter : in STD_LOGIC_VECTOR (1 downto 0);
6       anodes : out STD_LOGIC_VECTOR (7 downto 0)
7   );
8   end anodes_manager;
9
10  architecture Behavioral of anodes_manager is
11
12  signal anodes_switching : std_logic_vector(7 downto 0) := (others => '0');
13
14  begin
15  anodes <= not anodes_switching OR not "11111111";
16  anodes_process: process(counter)
17
18  begin
19
20  case counter is
21
22  when "00" =>
23    anodes_switching <= "00000001";
24
25  when "01" =>
26    anodes_switching <= "00000010";
27
```

```
28  when "10" =>
29    anodes_switching <= "00000100";
30
31  when "11" =>
32    anodes_switching <= "00001000";
33
34  when others =>
35    anodes_switching <= (others => '0');
36
37  end case;
38
39  end process;
40
41  end Behavioral;
```

Codice Componente 8.6: Definizione del componente anodes$_m$*anager*

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity cathodes_manager is
5   Port( counter : in STD_LOGIC_VECTOR (1 downto 0);
6       value : in STD_LOGIC_VECTOR (15 downto 0);
7       cathodes : out STD_LOGIC_VECTOR (7 downto 0));
8   end cathodes_manager;
9
10  architecture Behavioral of cathodes_manager is
11
12  constant zero : std_logic_vector(6 downto 0) := "1000000";
13  constant one : std_logic_vector(6 downto 0) := "1111001";
14  constant two : std_logic_vector(6 downto 0) := "0100100";
15  constant three : std_logic_vector(6 downto 0) := "0110000";
16  constant four : std_logic_vector(6 downto 0) := "0011001";
17  constant five : std_logic_vector(6 downto 0) := "0010010";
18  constant six : std_logic_vector(6 downto 0) := "0000010";
19  constant seven : std_logic_vector(6 downto 0) := "1111000";
20  constant eight : std_logic_vector(6 downto 0) := "0000000";
21  constant nine : std_logic_vector(6 downto 0) := "0010000";
22  constant a : std_logic_vector(6 downto 0) := "0001000";
23  constant b : std_logic_vector(6 downto 0) := "0000011";
24  constant c : std_logic_vector(6 downto 0) := "1000110";
25  constant d : std_logic_vector(6 downto 0) := "0100001";
26  constant e : std_logic_vector(6 downto 0) := "0000110";
27  constant f : std_logic_vector(6 downto 0) := "0001110";
28
29  alias digit_0 is value (3 downto 0);
30  alias digit_1 is value (7 downto 4);
31  alias digit_2 is value (11 downto 8);
32  alias digit_3 is value (15 downto 12);
```

```
33
34 signal DOT : STD_LOGIC;
35 signal cathodes_for_digit : std_logic_Vector(6 downto 0) := (others => '0');
36 signal nibble :std_logic_vector(3 downto 0) := (others => '0');
37
38 begin
39
40 digit_switching: process(counter, value)
41 begin
42
43 case counter is
44
45 when "00" =>
46   nibble <= digit_0;
47   DOT <= '0';
48
49 when "01" =>
50   nibble <= digit_1;
51   DOT <= '0';
52
53 when "10" =>
54   nibble <= digit_2;
55   DOT <= '1';
56
57 when "11" =>
58   nibble <= digit_3;
59   DOT <= '0';
60
61 when others =>
62   nibble <= (others => '0');
63   DOT <= '0';
64
65 end case;
66
67 end process;
68
69 seven_segment_decoder_process: process(nibble)
70 begin
71
72 case nibble is
73
74 when "0000" => cathodes_for_digit <= zero;
75 when "0001" => cathodes_for_digit <= one;
76 when "0010" => cathodes_for_digit <= two;
77 when "0011" => cathodes_for_digit <= three;
78 when "0100" => cathodes_for_digit <= four;
79 when "0101" => cathodes_for_digit <= five;
80 when "0110" => cathodes_for_digit <= six;
81 when "0111" => cathodes_for_digit <= seven;
```

```
82 when "1000" => cathodes_for_digit <= eight;
83 when "1001" => cathodes_for_digit <= nine;
84 when "1010" => cathodes_for_digit <= a;
85 when "1011" => cathodes_for_digit <= b;
86 when "1100" => cathodes_for_digit <= c;
87 when "1101" => cathodes_for_digit <= d;
88 when "1110" => cathodes_for_digit <= e;
89 when "1111" => cathodes_for_digit <= f;
90 when others => cathodes_for_digit <= (others => '0');
91
92 end case;
93
94 end process seven_segment_decoder_process;
95
96 cathodes <= not(DOT) & cathodes_for_digit;
97
98 end Behavioral;
```

Codice Componente 8.7: Definizione del componente cathodes$_m$*anager*

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity counter_mod4 is
6  Port( clock : in STD_LOGIC;
7      reset : in STD_LOGIC;
8      counter : out STD_LOGIC_VECTOR (1 downto 0));
9  end counter_mod4;
10
11 architecture Behavioral of counter_mod4 is
12
13 signal c : std_logic_vector (1 downto 0) := (others => '0');
14
15 begin
16
17 counter <= c;
18 counter_process: process(clock, reset)
19
20 begin
21 if reset = '1' then
22   c <= (others => '0');
23 elsif rising_edge(clock) then
24   c <= std_logic_vector(unsigned(c) + 1);
25 end if;
26
27 end process;
28
29 end Behavioral;
```

Codice Componente 8.8: Definizione del componente counter$_m od4$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity display_seven_segments is
5  Port( clock : in STD_LOGIC;
6      reset : in STD_LOGIC;
7      value : in STD_LOGIC_VECTOR (15 downto 0);
8      anodes : out STD_LOGIC_VECTOR (7 downto 0);
9      cathodes : out STD_LOGIC_VECTOR (7 downto 0));
10 end display_seven_segments;
11
12 architecture Structural of display_seven_segments is
13
14 signal counter : std_logic_vector(1 downto 0);
15
16 COMPONENT counter_mod4
17 PORT(
18     clock : in STD_LOGIC;
19     reset : in STD_LOGIC;
20     counter : out STD_LOGIC_VECTOR (1 downto 0)
21 );
22 END COMPONENT;
23
24 COMPONENT cathodes_manager
25 PORT(
26     counter : IN std_logic_vector(1 downto 0);
27     value : IN std_logic_vector(15 downto 0);
28     cathodes : OUT std_logic_vector(7 downto 0)
29 );
30 END COMPONENT;
31
32 COMPONENT anodes_manager
33 PORT(
34     counter : IN std_logic_vector(1 downto 0);
35     anodes : OUT std_logic_vector(7 downto 0)
36 );
37 END COMPONENT;
38
39 begin
40
41 counter_instance: counter_mod4 port map(
42   clock => clock,
43   reset => reset,
44   counter => counter
45 );
46
```

```
47 cathodes_instance: cathodes_manager port map(
48   counter => counter,
49   value => value,
50   cathodes => cathodes
51 );
52
53 anodes_instance: anodes_manager port map(
54   counter => counter,
55   anodes => anodes
56 );
57
58 end Structural;
```

Codice Componente 8.9: Definizione del componente display$_s$even$_s$egments

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  entity cronometro_on_board is
5  Port( clock : in STD_LOGIC;
6      reset : in STD_LOGIC;
7      load_sec : in STD_LOGIC;
8      load_min : in STD_LOGIC;
9      load_h : in STD_LOGIC;
10     in_value : in STD_LOGIC_VECTOR (5 downto 0);
11     led : out STD_LOGIC_VECTOR (5 downto 0);
12     anodes : out STD_LOGIC_VECTOR (7 downto 0);
13     cathodes : out STD_LOGIC_VECTOR (7 downto 0)
14 );
15 end cronometro_on_board;
16
17 architecture Behavioral of cronometro_on_board is
18 component display_seven_segments
19 Port( clock : in STD_LOGIC;
20     reset : in STD_LOGIC;
21     value : in STD_LOGIC_VECTOR (15 downto 0);
22     anodes : out STD_LOGIC_VECTOR (7 downto 0);
23     cathodes : out STD_LOGIC_VECTOR (7 downto 0)
24 );
25 end component;
26
27 COMPONENT Debouncer
28 PORT(
29     clock : IN std_logic;
30     reset : IN std_logic;
31     load_sec : IN std_logic;
32     load_min : IN std_logic;
33     load_h : IN std_logic;
34     in_value : IN std_logic_vector(5 downto 0);
```

```
35      sec : OUT std_logic_vector(5 downto 0);
36      min : OUT std_logic_vector(5 downto 0);
37      h : OUT std_logic_vector(4 downto 0);
38      load_sec_out : OUT std_logic;
39      load_min_out : OUT std_logic;
40      load_h_out : OUT std_logic
41  );
42  END COMPONENT;
43
44  component cronometro is
45  Generic( clock_frequency_in : integer;
46           clock_frequency_out : integer
47  );
48  Port( clock : in STD_LOGIC;
49      reset : in STD_LOGIC;
50      load_sec : in STD_LOGIC;
51      load_min : in STD_LOGIC;
52      load_h : in STD_LOGIC;
53      sec_in : in STD_LOGIC_VECTOR (5 downto 0);
54      min_in : in STD_LOGIC_VECTOR (5 downto 0);
55      h_in : in STD_LOGIC_VECTOR (4 downto 0);
56      sec_out : out STD_LOGIC_VECTOR (5 downto 0);
57      min_out : out STD_LOGIC_VECTOR (5 downto 0);
58      h_out : out STD_LOGIC_VECTOR (4 downto 0)
59  );
60  end component;
61
62  COMPONENT clock_filter
63  GENERIC(
64  clock_frequency_in : integer;
65  clock_frequency_out : integer
66  );
67  PORT(
68      clock_in : IN std_logic;
69      reset : in STD_LOGIC;
70      clock_out : OUT std_logic
71  );
72  END COMPONENT;
73
74  component bin2bcd_12bit is
75  Port( binIN : in STD_LOGIC_VECTOR (11 downto 0);
76      ones : out STD_LOGIC_VECTOR (3 downto 0);
77      tens : out STD_LOGIC_VECTOR (3 downto 0);
78      hundreds : out STD_LOGIC_VECTOR (3 downto 0);
79      thousands : out STD_LOGIC_VECTOR (3 downto 0)
80  );
81  end component;
82
83  signal clock_fx : STD_LOGIC;
```

```
84  signal value : STD_LOGIC_VECTOR (15 downto 0);
85  signal temp_sec_in : STD_LOGIC_VECTOR (5 downto 0);
86  signal temp_min_in : STD_LOGIC_VECTOR (5 downto 0);
87  signal temp_h_in : STD_LOGIC_VECTOR (4 downto 0);
88
89  signal temp_load_sec : STD_LOGIC;
90  signal temp_load_min : STD_LOGIC;
91  signal temp_load_h : STD_LOGIC;
92  signal temp_sec_out : STD_LOGIC_VECTOR (5 downto 0);
93  signal temp_min_out : STD_LOGIC_VECTOR (5 downto 0);
94  signal temp_h_out : STD_LOGIC_VECTOR (4 downto 0);
95  signal temp_binIN_0 : STD_LOGIC_VECTOR (11 downto 0);
96  signal temp_binIN_1 : STD_LOGIC_VECTOR (11 downto 0);
97
98  begin
99  sevent_segment: display_seven_segments
100 port map(
101       clock => clock_fx,
102       reset => reset,
103       value => value,
104       anodes => anodes,
105       cathodes => cathodes
106 );
107
108 Inst_Debouncer: Debouncer PORT MAP(
109       clock => clock,
110       reset => reset,
111       load_sec => load_sec,
112       load_min => load_min,
113       load_h => load_h,
114       in_value => in_value,
115       sec => temp_sec_in,
116       min => temp_min_in,
117       h => temp_h_in,
118       load_sec_out => temp_load_sec,
119       load_min_out => temp_load_min,
120       load_h_out=> temp_load_h
121 );
122
123 cr: cronometro
124 generic map(
125 clock_frequency_in => 100000000,
126 clock_frequency_out => 1
127 )
128 port map(
129       clock => clock,
130       reset => reset,
131       load_sec => temp_load_sec,
132       load_min => temp_load_min,
```

```
133       load_h => temp_load_h,
134       sec_in => temp_sec_in,
135       min_in => temp_min_in,
136       h_in => temp_h_in,
137       sec_out => temp_sec_out,
138       min_out => temp_min_out,
139       h_out => temp_h_out
140 );
141 
142 clk_filter: clock_filter
143 generic map(
144 clock_frequency_in => 100000000,
145 clock_frequency_out => 1000
146 )
147 PORT MAP(
148       clock_in => clock,
149       reset => reset,
150       clock_out => clock_fx
151 );
152 
153 temp_binIN_1 <= "0000000" & temp_h_out;
154 
155 -- Si concatenano 6 bit '0' essendo il
156 -- convertitore binario - BCD a 12 bit
157 
158 converter_1: bin2bcd_12bit
159 port map(
160       binIN => temp_binIN_1,
161       ones => value(11 downto 8),
162       tens => value(15 downto 12)
163 );
164 
165 temp_binIN_0 <= "000000" & temp_min_out;
166 
167 converter_0: bin2bcd_12bit
168 port map(
169       binIN => temp_binIN_0,
170       ones => value(3 downto 0),
171       tens => value(7 downto 4)
172       );
173 
174 led <= temp_sec_out;
175 
176 end Behavioral;
```

Codice Componente 8.10: Definizione del componente cronometro$_o n_b oard$

```
1 
2 LIBRARY ieee;
```

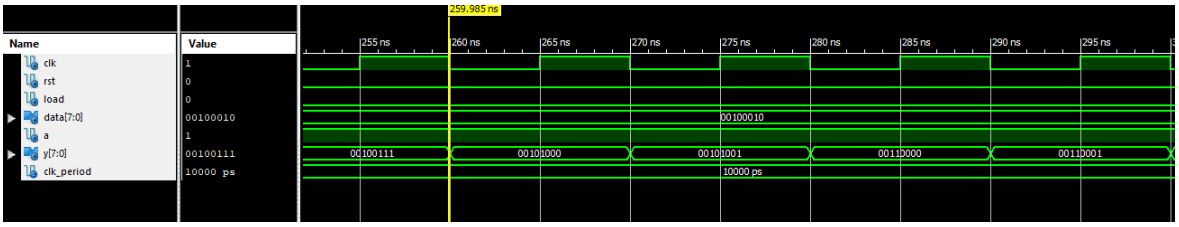
```
3  USE ieee.std_logic_1164.ALL;
4
5  ENTITY cronometro_tb IS
6  END cronometro_tb;
7
8  ARCHITECTURE behavior OF cronometro_tb IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11
12     COMPONENT cronometro
13     PORT(
14          clock : IN  std_logic;
15          reset : IN  std_logic;
16          load_sec : IN  std_logic;
17          load_min : IN  std_logic;
18          load_h : IN  std_logic;
19          sec_in : IN  std_logic_vector(5 downto 0);
20          min_in : IN  std_logic_vector(5 downto 0);
21          h_in : IN  std_logic_vector(4 downto 0);
22          sec_out : OUT  std_logic_vector(5 downto 0);
23          min_out : OUT  std_logic_vector(5 downto 0);
24          h_out : OUT  std_logic_vector(4 downto 0)
25         );
26     END COMPONENT;
27
28
29    --Inputs
30    signal clock : std_logic := '0';
31    signal reset : std_logic := '0';
32    signal load_sec : std_logic := '0';
33    signal load_min : std_logic := '0';
34    signal load_h : std_logic := '0';
35    signal sec_in : std_logic_vector(5 downto 0) := (others => '0');
36    signal min_in : std_logic_vector(5 downto 0) := (others => '0');
37    signal h_in : std_logic_vector(4 downto 0) := (others => '0');
38
39   --Outputs
40    signal sec_out : std_logic_vector(5 downto 0);
41    signal min_out : std_logic_vector(5 downto 0);
42    signal h_out : std_logic_vector(4 downto 0);
43
44    -- Clock period definitions
45    constant clock_period : time := 10 ns;
46
47 BEGIN
48
49   -- Instantiate the Unit Under Test (UUT)
50    uut: cronometro PORT MAP (
51           clock => clock,
```

```
52            reset => reset,
53            load_sec => load_sec,
54            load_min => load_min,
55            load_h => load_h,
56            sec_in => sec_in,
57            min_in => min_in,
58            h_in => h_in,
59            sec_out => sec_out,
60            min_out => min_out,
61            h_out => h_out
62          );
63
64    -- Clock process definitions
65    clock_process :process
66    begin
67     clock <= '0';
68     wait for clock_period/2;
69     clock <= '1';
70     wait for clock_period/2;
71    end process;
72
73
74    -- Stimulus process
75    stim_proc: process
76    begin
77       -- hold reset state for 100 ns.
78       wait for 100 ns;
79
80       wait for clock_period*10;
81
82       -- insert stimulus here
83
84     reset <= '1';
85
86     wait for 5 ns;
87
88     reset <= '0';
89
90     wait for clock_period*50;
91
92     sec_in <= "111011";
93     min_in <= "111011";
94     h_in <= "10111";
95     load_sec <= '1';
96     load_min <= '1';
97     load_h <= '1';
98
99     wait for 5 ns;
100
```

```
101     load_sec <= '0';
102     load_min <= '0';
103     load_h <= '0';
104 
105       wait;
106    end process;
107 
108 END;
```

Codice Componente 8.11: Definizione del componente cronometro_t*b*

## 8.3 Simulazione




Vediamo nel dettaglio la simulazione soltanto per il componente dei secondi : vediamo come ad ogni colpo di clock questo viene incrementato secondo la notazione utilizzata per unità e decine.

## 8.4 Sintesi su board FPGA

Per interfacciare il design realizzato con l'I/O di board è stata adottata la seguente soluzione:

Per il clock si è associato alla variabile clock quello della scheda, poi opportunamente filtrato per ottenere una base dei tempi corretta.

```
## Clock signal
NET "clock"   LOC = "E3"  | IOSTANDARD = "LVCMOS33";
```

Per inizializzare ore, minuti e secondi sono stati utilizzati i 6 switch più a destra della scheda

```
## Switches
NET "in_value<0>"          LOC=J15 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_RS0_15
NET "in_value<1>"          LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "in_value<2>"          LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
NET "in_value<3>"          LOC=R15 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_14
NET "in_value<4>"          LOC=R17 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_14
NET "in_value<5>"          LOC=T18 | IOSTANDARD=LVCMOS33; #IO_L7N_T1_D10_14
```

I bottoni sono stati utilizzati per resettare il conteggio ("reset") e per acquisire in tempi diversi i valori di inizializzazione di ore, minuti e secondi.

```
## Buttons
NET "load_sec"           LOC=M17 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_D15_14
NET "load_min"           LOC=M18 | IOSTANDARD=LVCMOS33; #IO_L4N_T0_D05_14
NET "load_h"           LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14

#NET "RESET"     LOC=C12 | IOSTANDARD=LVCMOS33; #IO_L3P_T0_DQS_AD1P_15

NET "reset"           LOC=P18 | IOSTANDARD=LVCMOS33; #IO_L9N_T1_DQS_D13_14
```

L'uscita relativa ai secondi è mostrata sui 6 led più a destra.

```
 ## LEDs
NET "led<0>"          LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
NET "led<1>"          LOC=K15 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_RS1_15
NET "led<2>"          LOC=J13 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A25_15
NET "led<3>"          LOC=N14 | IOSTANDARD=LVCMOS33; #IO_L8P_T1_D11_14
NET "led<4>"          LOC=R18 | IOSTANDARD=LVCMOS33; #IO_L7P_T1_D09_14
NET "led<5>"          LOC=V17 | IOSTANDARD=LVCMOS33; #IO_L18N_T2_A11_D27_14
```

# 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

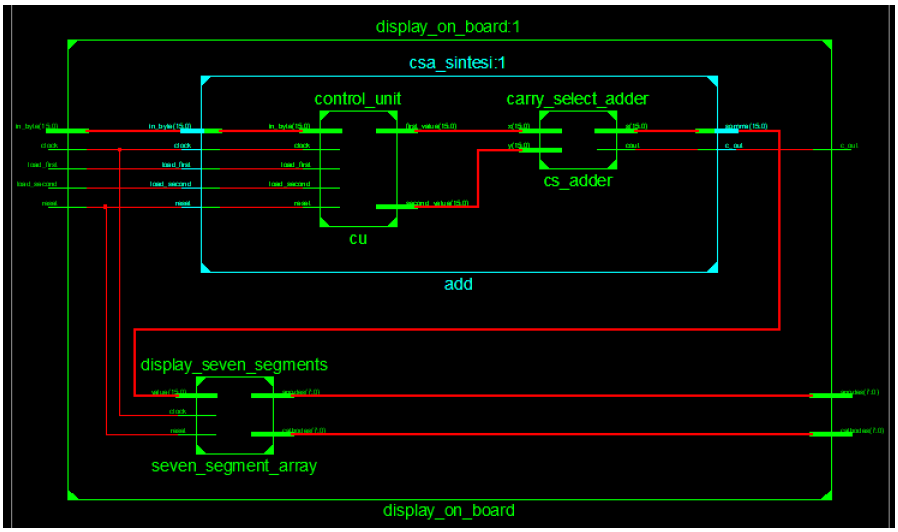
mediante l'utilizzo dei dispositivi di input/output (switch, bottoni, led, display) presenti sulla board di

sviluppo in dotazione al gruppo. La modalità di utilizzo degli stessi è a completa discrezione degli studenti.

## 9.2 Soluzione

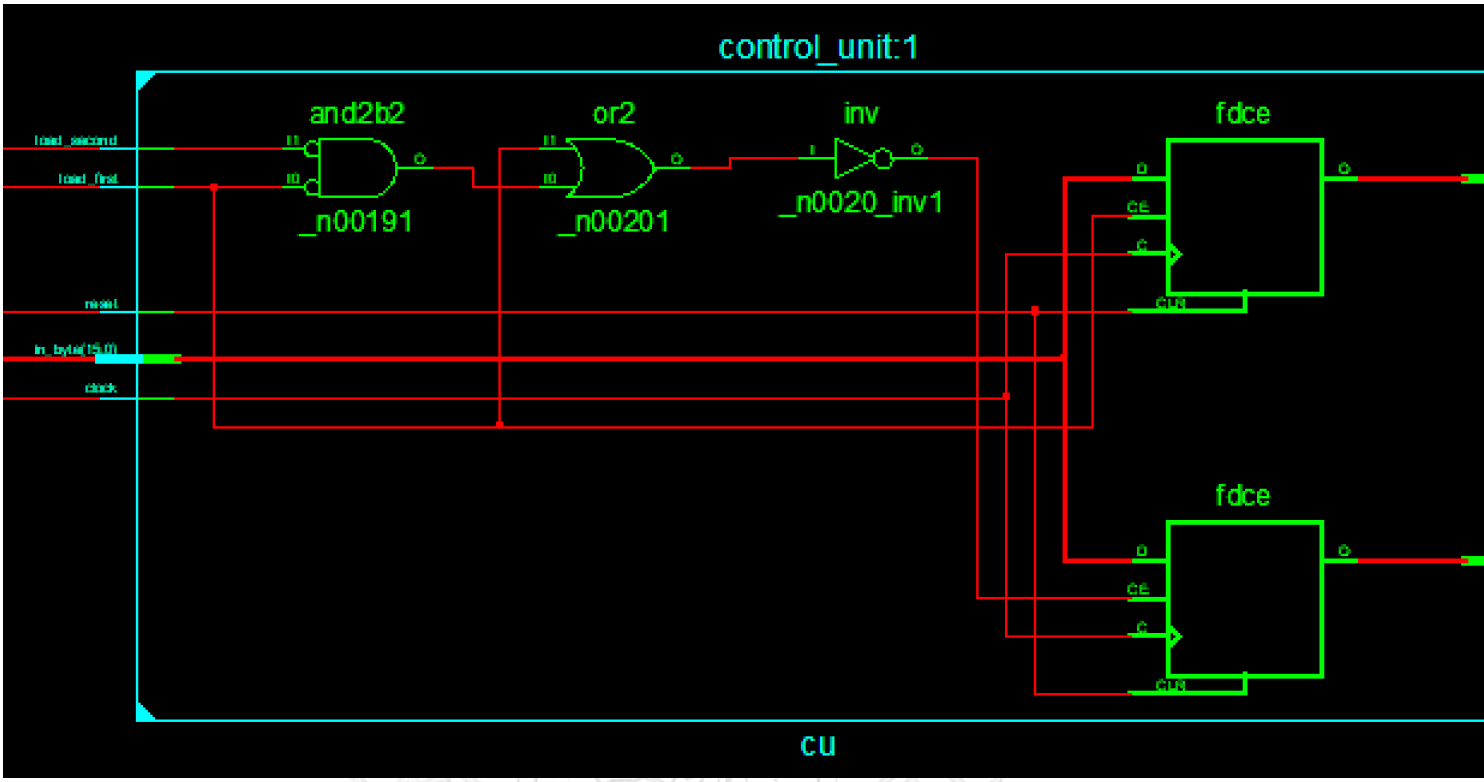
### 9.2.1 Schematici

- Schematico generale



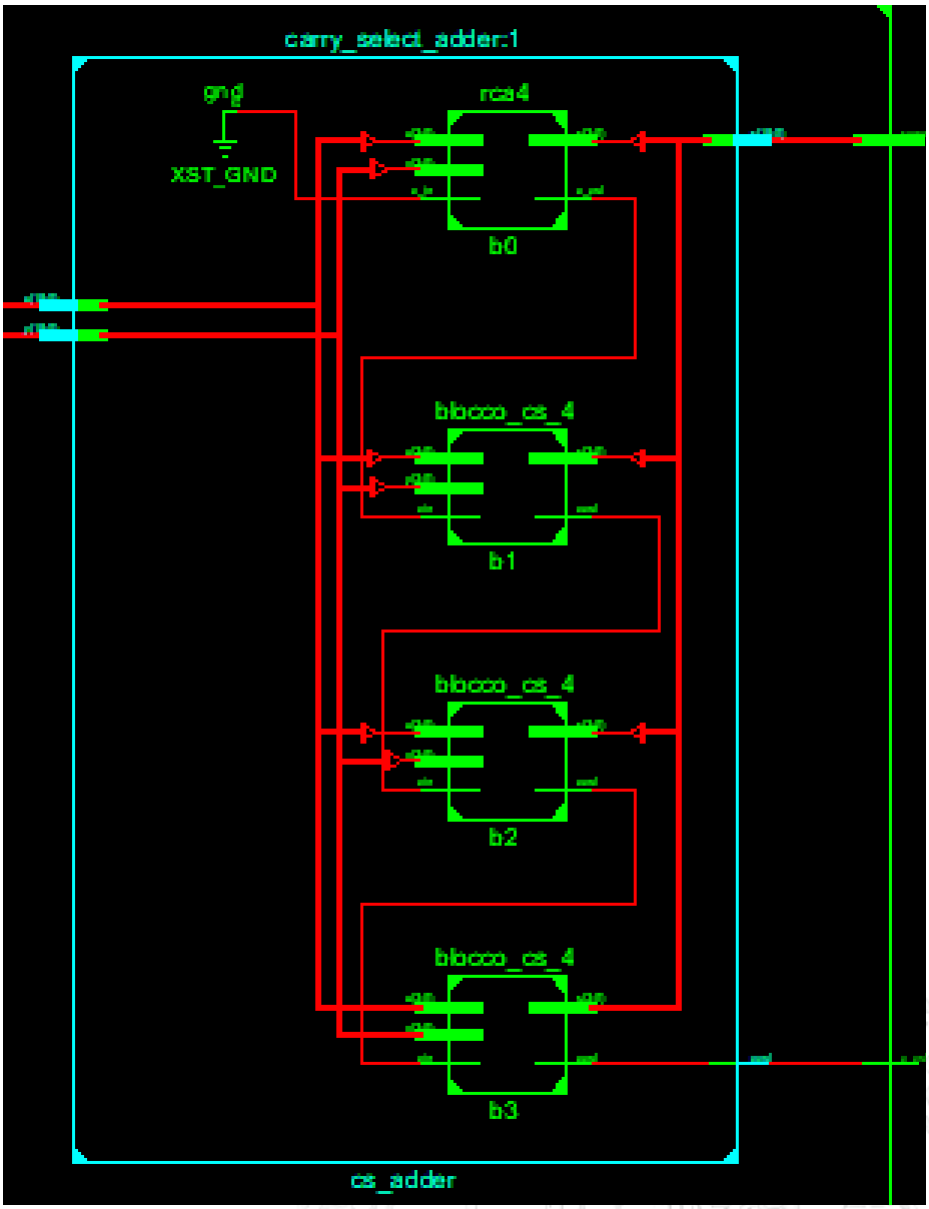

Lo schematico generale è costituito dal carry select adder, dalla control unit per inserire i due addendi in istanti diversi e dal display_seven_segment per la visualizzazione del risultato sul display della board.

- Schematico control_unit



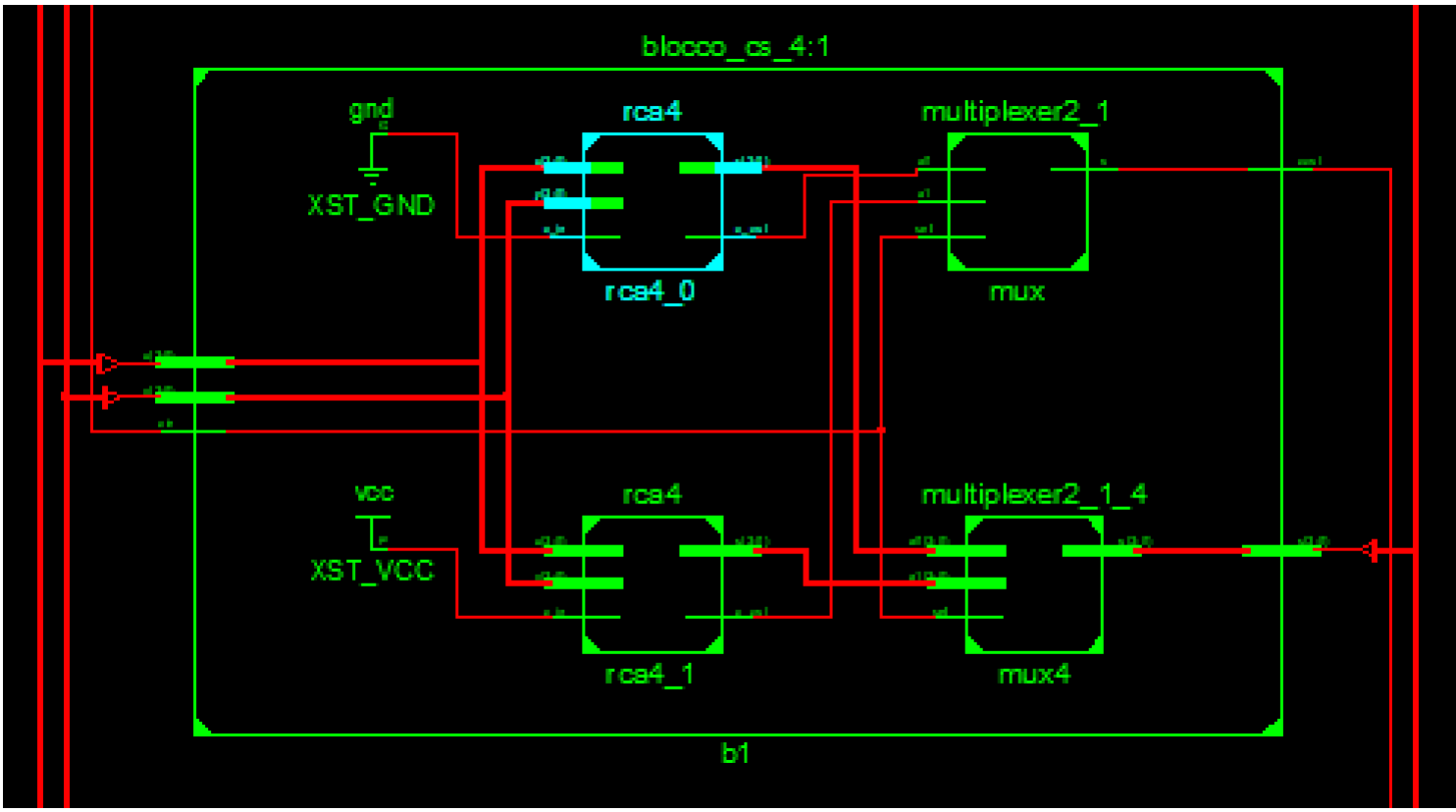

Schema classico per la control_unit, che a partire dai segnali load_first e load_second assegna correttamente i valori ai registri del primo e del secondo operando.

- Schematico Carry Select Adder




Carry Select Adder è composto da un rca a 4 bit e da 4 blocchi esplicitati nella figura successiva.

- Schematico Blocco Carry Select




Tale blocco è composto da 2 rca a 4 bit dei quali uno calcola somma e riporto uscente con carry entrante pari ad 1 e l'altro pari a 0. Successivamente a seconda dal valore di selezione del multiplexer si sceglieranno le opportune uscite.

## 9.2.2 Codice

### 9.2.2.1 Carry_Select_Adder

```
1
2 rca4 - traccia n°4
```

Codice Componente 9.1: Definizione del componente rca4

```
1
2 multiplexer2_1 - traccia n°1
```

Codice Componente 9.2: Definizione del componente multiplexer$2_1$

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity multiplexer2_1_4 is
6     Port ( a0 : in  STD_LOGIC_VECTOR (3 downto 0);
7            a1 : in  STD_LOGIC_VECTOR (3 downto 0);
8            sel : in  STD_LOGIC;
```

```
9             u : out  STD_LOGIC_VECTOR (3 downto 0));
10 end multiplexer2_1_4;
11 
12 architecture rtl of multiplexer2_1_4 is
13 
14 begin
15 
16  u <= a0 when sel = '0' else
17     a1 when sel = '1' else
18     "----";
19 
20 end rtl;
```

Codice Componente 9.3: Definizione del componente multiplexer2$_{14}$

```
1 
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4 
5  entity blocco_cs_4 is
6      Port ( cin : in  STD_LOGIC;
7             x : in  STD_LOGIC_VECTOR (3 downto 0);
8             y : in  STD_LOGIC_VECTOR (3 downto 0);
9             s : out  STD_LOGIC_VECTOR (3 downto 0);
10            cout : out  STD_LOGIC);
11 end blocco_cs_4;
12 
13 architecture Structural of blocco_cs_4 is
14 
15 component multiplexer2_1
16     Port ( a0 : in  STD_LOGIC;
17            a1 : in  STD_LOGIC;
18            sel : in  STD_LOGIC;
19            u : out  STD_LOGIC);
20 end component;
21 
22 component multiplexer2_1_4
23     Port ( a0 : in  STD_LOGIC_VECTOR (3 downto 0);
24            a1 : in  STD_LOGIC_VECTOR (3 downto 0);
25            sel : in  STD_LOGIC;
26            u : out  STD_LOGIC_VECTOR (3 downto 0));
27 end component;
28 
29 component rca4
30     Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
31            y : in  STD_LOGIC_VECTOR (3 downto 0);
32            c_in : in  STD_LOGIC;
33            s : out  STD_LOGIC_VECTOR (3 downto 0);
34            c_out : out  STD_LOGIC);
```

```
35 end component;
36
37 signal c0 : STD_LOGIC;
38 signal c1 : STD_LOGIC;
39 signal s0 : STD_LOGIC_VECTOR (3 downto 0);
40 signal s1 : STD_LOGIC_VECTOR (3 downto 0);
41
42 begin
43
44 rca4_0 : rca4
45 Port Map ( x => x,
46         y => y,
47         c_in => '0',
48         s => s0,
49         c_out => c0
50       );
51
52 rca4_1 : rca4
53 Port Map ( x => x,
54         y => y,
55         c_in => '1',
56         s => s1,
57         c_out => c1
58       );
59
60 mux : multiplexer2_1
61 Port Map ( a0 => c0,
62         a1 => c1,
63         sel => cin,
64         u => cout
65       );
66
67 mux4 : multiplexer2_1_4
68 Port Map ( a0 => s0,
69         a1 => s1,
70         sel => cin,
71         u => s
72       );
73
74 end Structural;
```

Codice Componente 9.4: Definizione del componente blocco$_cs_4$

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity carry_select_adder is
6     Port ( x : in  STD_LOGIC_VECTOR (15 downto 0);
```

```
7           y : in  STD_LOGIC_VECTOR (15 downto 0);
8           --cin : in  STD_LOGIC;
9           s : out  STD_LOGIC_VECTOR (15 downto 0);
10          cout : out  STD_LOGIC);
11 end carry_select_adder;
12
13 architecture Structural of carry_select_adder is
14
15 component rca4
16     Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
17            y : in  STD_LOGIC_VECTOR (3 downto 0);
18            c_in : in  STD_LOGIC;
19            s : out  STD_LOGIC_VECTOR (3 downto 0);
20            c_out : out  STD_LOGIC);
21 end component;
22
23 component blocco_cs_4
24     Port ( cin : in  STD_LOGIC;
25            x : in  STD_LOGIC_VECTOR (3 downto 0);
26            y : in  STD_LOGIC_VECTOR (3 downto 0);
27            s : out  STD_LOGIC_VECTOR (3 downto 0);
28            cout : out  STD_LOGIC);
29 end component;
30
31 signal c0 : STD_LOGIC;
32 signal c1 : STD_LOGIC;
33 signal c2 : STD_LOGIC;
34
35 begin
36
37 b0 : rca4
38 Port Map ( x => x(3 downto 0),
39         y => y(3 downto 0),
40         c_in => '0',
41         s => s(3 downto 0),
42         c_out => c0
43       );
44
45 b1 : blocco_cs_4
46 Port Map ( x => x(7 downto 4),
47         y => y(7 downto 4),
48         cin => c0,
49         s => s(7 downto 4),
50         cout => c1
51       );
52
53 b2 : blocco_cs_4
54 Port Map ( x => x(11 downto 8),
55         y => y(11 downto 8),
```

```
56          cin => c1,
57          s => s(11 downto 8),
58          cout => c2
59        );
60
61  b3 : blocco_cs_4
62  Port Map ( x => x(15 downto 12),
63          y => y(15 downto 12),
64          cin => c2,
65          s => s(15 downto 12),
66          cout => cout
67        );
68
69  end Structural;
```

Codice Componente 9.5: Definizione del componente $carry_select_adder$

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity control_unit is
6       Port ( clock : in  STD_LOGIC;
7              reset : in  STD_LOGIC;
8              load_first : in  STD_LOGIC;
9              load_second : in  STD_LOGIC;
10             in_byte : in  STD_LOGIC_VECTOR (15 downto 0);
11             first_value : out  STD_LOGIC_VECTOR (15 downto 0);
12             second_value : out  STD_LOGIC_VECTOR (15 downto 0));
13  end control_unit;
14
15  architecture Behavioral of control_unit is
16
17  signal reg_first : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
18  signal reg_second : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
19
20  begin
21
22  first_value <= reg_first;
23  second_value <= reg_second;
24
25  main : process(clock, reset)
26  begin
27
28    if reset = '1' then
29      reg_first <= (others => '0');
30      reg_second <= (others => '0');
31    elsif clock'event and clock = '1' then
32      if load_first = '1' then
```

```
33        reg_first(15 downto 0) <= in_byte;
34      elsif load_second = '1' then
35        reg_second(15 downto 0) <= in_byte;
36      end if;
37    end if;
38
39  end process;
40
41  end Behavioral;
```

Codice Componente 9.6: Definizione del componente control$_unit$

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity csa_sintesi is
6       Port ( clock : in  STD_LOGIC;
7              reset : in  STD_LOGIC;
8              in_byte : in  STD_LOGIC_VECTOR (15 downto 0);
9              load_first : in  STD_LOGIC;
10             load_second : in  STD_LOGIC;
11             somma : out  STD_LOGIC_VECTOR (15 downto 0);
12             c_out : out  STD_LOGIC);
13  end csa_sintesi;
14
15  architecture Structural of csa_sintesi is
16
17  component control_unit
18     Port ( clock : in  STD_LOGIC;
19             reset : in  STD_LOGIC;
20             load_first : in  STD_LOGIC;
21             load_second : in  STD_LOGIC;
22             in_byte : in  STD_LOGIC_VECTOR (15 downto 0);
23             first_value : out  STD_LOGIC_VECTOR (15 downto 0);
24             second_value : out  STD_LOGIC_VECTOR (15 downto 0));
25  end component;
26
27  component carry_select_adder
28      Port ( x : in  STD_LOGIC_VECTOR (15 downto 0);
29             y : in  STD_LOGIC_VECTOR (15 downto 0);
30             --cin : in  STD_LOGIC;
31             s : out  STD_LOGIC_VECTOR (15 downto 0);
32             cout : out  STD_LOGIC);
33  end component;
34
35  signal first : STD_LOGIC_VECTOR (15 downto 0);
36  signal second : STD_LOGIC_VECTOR (15 downto 0);
37
```

```
38 begin
39
40 cu : control_unit
41   port map (
42     clock => clock,
43     reset => reset,
44     load_first => load_first,
45     load_second => load_second,
46     in_byte => in_byte,
47     first_value => first,
48     second_value => second
49     );
50
51 cs_adder : carry_select_adder
52   port map (
53     x => first,
54     y => second,
55     s => somma,
56     --cin => '0',
57     cout => c_out
58     );
59
60 end Structural;
```

Codice Componente 9.7: Definizione del componente csa$_s$*intesi*

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity anodes_manager is
6      Port ( counter : in  STD_LOGIC_VECTOR (1 downto 0);
7             enable_digit : in  STD_LOGIC_VECTOR (3 downto 0);
8             anodes : out  STD_LOGIC_VECTOR (7 downto 0)
9           );
10 end anodes_manager;
11
12 architecture Behavioral of anodes_manager is
13
14 signal anodes_switching : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
15
16 begin
17
18 anodes(3 downto 0) <= not enable_digit or not anodes_switching;
19 anodes ( 7 downto 4) <= (others => '1');
20
21 anodes_process: process(counter, enable_digit)
22 begin
```

```
23    --a seconda del valore di counter le cifre si illuminano una alla volta da
          destra a sinistra
24    case counter is
25      when "00" =>
26        anodes_switching <= x"1";
27      when "01" =>
28        anodes_switching <= x"2";
29      when "10" =>
30        anodes_switching <= x"4";
31      when "11" =>
32        anodes_switching <= x"8";
33      when others =>
34        anodes_switching <= (others => '0');
35    end case;
36
37  end process;
38
39
40  end Behavioral;
```

Codice Componente 9.8: Definizione del componente anodes$_m$*anager*

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity cathodes_manager is
6       Port ( counter : in  STD_LOGIC_VECTOR (1 downto 0);
7              value : in  STD_LOGIC_VECTOR (15 downto 0);
8              cathodes : out  STD_LOGIC_VECTOR (7 downto 0));
9   end cathodes_manager;
10
11  architecture Behavioral of cathodes_manager is
12
13  constant zero   : std_logic_vector(6 downto 0) := "1000000";
14  constant one    : std_logic_vector(6 downto 0) := "1111001";
15  constant two    : std_logic_vector(6 downto 0) := "0100100";
16  constant three  : std_logic_vector(6 downto 0) := "0110000";
17  constant four   : std_logic_vector(6 downto 0) := "0011001";
18  constant five   : std_logic_vector(6 downto 0) := "0010010";
19  constant six    : std_logic_vector(6 downto 0) := "0000010";
20  constant seven  : std_logic_vector(6 downto 0) := "1111000";
21  constant eight  : std_logic_vector(6 downto 0) := "0000000";
22  constant nine   : std_logic_vector(6 downto 0) := "0010000";
23  constant a     : std_logic_vector(6 downto 0) := "0001000";
24  constant b      : std_logic_vector(6 downto 0) := "0000011";
25  constant c      : std_logic_vector(6 downto 0) := "1000110";
26  constant d      : std_logic_vector(6 downto 0) := "0100001";
27  constant e      : std_logic_vector(6 downto 0) := "0000110";
```

```
28 constant f      : std_logic_vector(6 downto 0) := "0001110";
29
30 alias digit_0 is value (3 downto 0);
31 alias digit_1 is value (7 downto 4);
32 alias digit_2 is value (11 downto 8);
33 alias digit_3 is value (15 downto 12);
34
35 signal cathodes_for_digit : std_logic_Vector(6 downto 0) := (others => '0');
36 signal nibble :std_logic_vector(3 downto 0) := (others => '0');
37
38 begin
39
40 digit_switching: process(counter, value)
41
42 begin
43   case counter is
44     when "00" =>
45       nibble <= digit_0;
46     when "01" =>
47       nibble <= digit_1;
48     when "10" =>
49       nibble <= digit_2;
50     when "11" =>
51       nibble <= digit_3;
52     when others =>
53       nibble <= (others => '0');
54   end case;
55 end process;
56
57 seven_segment_decoder_process: process(nibble)
58   begin
59     case nibble is
60       when "0000" => cathodes_for_digit <= zero;
61       when "0001" => cathodes_for_digit <= one;
62       when "0010" => cathodes_for_digit <= two;
63       when "0011" => cathodes_for_digit <= three;
64       when "0100" => cathodes_for_digit <= four;
65       when "0101" => cathodes_for_digit <= five;
66       when "0110" => cathodes_for_digit <= six;
67       when "0111" => cathodes_for_digit <= seven;
68       when "1000" => cathodes_for_digit <= eight;
69       when "1001" => cathodes_for_digit <= nine;
70       when "1010" => cathodes_for_digit <= a;
71       when "1011" => cathodes_for_digit <= b;
72       when "1100" => cathodes_for_digit <= c;
73       when "1101" => cathodes_for_digit <= d;
74       when "1110" => cathodes_for_digit <= e;
75       when "1111" => cathodes_for_digit <= f;
76     when others => cathodes_for_digit <= (others => '0');
```

```
77      end case;
78    end process seven_segment_decoder_process;
79
80  cathodes <= '1' & cathodes_for_digit;
81
82  end Behavioral;
```

Codice Componente 9.9: Definizione del componente cathodes$_m$*anager*

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity counter_mod4 is
    Port ( clock : in  STD_LOGIC;
           reset : in  STD_LOGIC;
          enable : in STD_LOGIC;
           counter : out  STD_LOGIC_VECTOR (1 downto 0));
end counter_mod4;

architecture Behavioral of counter_mod4 is

signal c : std_logic_vector (1 downto 0) := (others => '0');

begin

counter <= c;

counter_process: process(clock, reset, c)
begin

  if reset = '1' then
    c <= (others => '0');
  elsif clock'event AND clock = '1' AND enable = '1' then
    c <= std_logic_vector(unsigned(c) + 1);
  end if;

end process;

end Behavioral;
```

Codice Componente 9.10: Definizione del componente counter$_m$*od*4

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clock_filter is
   generic(
        clock_frequency_in : integer := 50000000;
```

```
        clock_frequency_out : integer := 5000000
        );
    Port ( clock_in : in  STD_LOGIC;
        reset : in STD_LOGIC;
           clock_out : out  STD_LOGIC);
end clock_filter;

architecture Behavioral of clock_filter is

signal clockfx : std_logic := '0';

constant count_max_value : integer := clock_frequency_in/(
   clock_frequency_out)-1;

begin

clock_out <= clockfx;

count_for_division: process(clock_in, reset)
variable counter : integer range 0 to count_max_value := 0;
begin

  if reset = '1' then
    counter := 0;
    clockfx <= '0';
  elsif clock_in'event and clock_in = '1' then
    if counter = count_max_value then
      clockfx <=  '1';
      counter := 0;
    else
      clockfx <=  '0';
      counter := counter + 1;
    end if;
  end if;

end process;


end Behavioral;
```

Codice Componente 9.11: Definizione del componente $clock_filter$

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity display_seven_segments is
  Generic(
```

```
7         clock_frequency_in : integer := 100000000; --parametri da
              customizzare, questo è il valore di default
8         clock_frequency_out : integer := 50000
9         );
10    Port ( clock : in  STD_LOGIC;
11           reset : in  STD_LOGIC;
12           value : in  STD_LOGIC_VECTOR (15 downto 0);
13           anodes : out  STD_LOGIC_VECTOR (7 downto 0);
14           cathodes : out  STD_LOGIC_VECTOR (7 downto 0));
15 end display_seven_segments;
16
17 architecture Structural of display_seven_segments is
18
19 signal counter : std_logic_vector(1 downto 0);
20 signal clock_fx : std_logic := '0';
21
22 COMPONENT counter_mod4
23   PORT(
24     clock : in  STD_LOGIC;
25       reset : in  STD_LOGIC;
26     enable : in STD_LOGIC;
27       counter : out  STD_LOGIC_VECTOR (1 downto 0)
28     );
29 END COMPONENT;
30
31 COMPONENT cathodes_manager
32   PORT(
33     counter : IN std_logic_vector(1 downto 0);
34     value : IN std_logic_vector(15 downto 0);
35     cathodes : OUT std_logic_vector(7 downto 0)
36     );
37 END COMPONENT;
38
39 COMPONENT anodes_manager
40   PORT(
41
42     counter : IN std_logic_vector(1 downto 0);
43     enable_digit : IN std_logic_vector(3 downto 0);
44     anodes : OUT std_logic_vector(7 downto 0)
45     );
46 END COMPONENT;
47
48 COMPONENT clock_filter
49   GENERIC(
50         clock_frequency_in : integer := 100000000;
51         clock_frequency_out : integer := 50000
52         );
53   PORT(
54     clock_in : IN std_logic;
```

```
55        reset : in  STD_LOGIC;
56      clock_out : OUT std_logic
57      );
58  END COMPONENT;
59  begin
60  --il clock filter genera un segnale di abilitazione per il contatore mod4
        che viene usato
61  --come segnale di conteggio e quindi di fatto fornisce la frequenza con cui
        viene modificata
62  --la cifra da mostrare
63
64  clk_filter: clock_filter GENERIC MAP(
65    clock_frequency_in => clock_frequency_in,
66    clock_frequency_out => clock_frequency_out
67    )
68    PORT MAP(
69      clock_in => clock,
70      reset => reset,
71      clock_out => clock_fx
72    );
73
74  counter_instance: counter_mod4 port map(
75    clock => clock,
76    enable => clock_fx,
77    reset => reset,
78    counter => counter
79  );
80  --il valore di conteggio viene usato dal gestore dei catodi e degli anodi
        per
81  --selezionare l'anodo da accendere e il suo rispettivo valore
82  cathodes_instance: cathodes_manager port map(
83    counter => counter,
84    value => value,
85    cathodes => cathodes
86  );
87
88  anodes_instance: anodes_manager port map(
89    counter => counter,
90    enable_digit => (others => '1'),
91    anodes => anodes
92  );
93
94
95  end Structural;
```

Codice Componente 9.12: Definizione del componente $display_seven_segments$

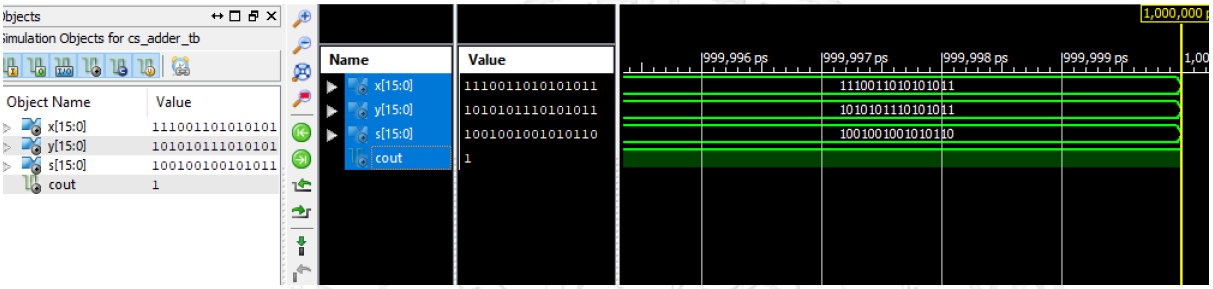
```
1
2  library IEEE;
```

```
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity display_on_board is
6    Port(
7        clock : in  STD_LOGIC;
8        reset : in  STD_LOGIC;
9        load_first : in  STD_LOGIC;
10         load_second : in  STD_LOGIC;
11       in_byte : in STD_LOGIC_VECTOR(15 downto 0);
12       anodes : out  STD_LOGIC_VECTOR (7 downto 0);
13       cathodes : out  STD_LOGIC_VECTOR (7 downto 0);
14       c_out : out STD_LOGIC
15         );
16 end display_on_board;
17
18 architecture Structural of display_on_board is
19
20 COMPONENT display_seven_segments
21   GENERIC(
22         clock_frequency_in : integer := 100000000;
23         clock_frequency_out : integer := 50000
24         );
25   PORT(
26     clock : IN std_logic;
27     reset : IN std_logic;
28     value : IN std_logic_vector(15 downto 0);--4 nibble da mostrare
29     anodes : OUT std_logic_vector(7 downto 0);
30     cathodes : OUT std_logic_vector(7 downto 0)
31     );
32 END COMPONENT;
33
34 COMPONENT csa_sintesi
35     Port ( clock : in  STD_LOGIC;
36            reset : in  STD_LOGIC;
37            in_byte : in  STD_LOGIC_VECTOR (15 downto 0);
38            load_first : in  STD_LOGIC;
39            load_second : in  STD_LOGIC;
40            somma : out  STD_LOGIC_VECTOR (15 downto 0);
41            c_out : out  STD_LOGIC);
42 END COMPONENT;
43
44 signal cu_value : std_logic_vector(15 downto 0);
45 signal cu_enable : std_logic_vector(3 downto 0);
46
47 begin
48
49 seven_segment_array: display_seven_segments GENERIC MAP(
50   clock_frequency_in => 100000000,
51   clock_frequency_out => 50000
```

```
52    )
53    PORT MAP(
54      clock => clock,
55      reset => reset,
56      value => cu_value,
57      anodes => anodes,
58      cathodes => cathodes
59  );
60
61  add: csa_sintesi PORT MAP(
62      clock => clock,
63      reset => reset,
64      load_first => load_first,
65      load_second => load_second,
66      in_byte => in_byte,
67      somma => cu_value,
68      c_out => c_out
69    );
70
71  end Structural;
```

Codice Componente 9.13: Definizione del componente display_onboard

## 9.3 Simulazione




Esempio di prodotto tra due valori x e y con rispettivo output s.

## 9.4 Sintesi su board FPGA

Per interfacciare il design realizzato con l'I/O di board è stata adottata la seguente soluzione:

Per il clock si è utilizzato il clock della scheda:

```
## Clock signal
NET "clock"    LOC = "E3"   | IOSTANDARD = "LVCMOS33";
```

Per gli ingressi sono stati utilizzati gli 8 switch più a destra della board:

```
## Switches
NET "in_byte<0>"        LOC=J15 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_RS0_15
NET "in_byte<1>"        LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "in_byte<2>"        LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
NET "in_byte<3>"        LOC=R15 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_14
NET "in_byte<4>"        LOC=R17 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_14
NET "in_byte<5>"        LOC=T18 | IOSTANDARD=LVCMOS33; #IO_L7N_T1_D10_14
NET "in_byte<6>"        LOC=U18 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A13_D29_14
NET "in_byte<7>"        LOC=R13 | IOSTANDARD=LVCMOS33; #IO_L5N_T0_D07_14

NET "in_byte<8>"        LOC=T8 | IOSTANDARD=LVCMOS18; #IO_L24N_T3_34
NET "in_byte<9>"        LOC=U8 | IOSTANDARD=LVCMOS18; #IO_25_34
NET "in_byte<10>"       LOC=R16 | IOSTANDARD=LVCMOS33; #IO_L15P_T2_DQS_RDWR_B_14
NET "in_byte<11>"       LOC=T13 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_A03_D19_14
NET "in_byte<12>"       LOC=H6 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_35
NET "in_byte<13>"       LOC=U12 | IOSTANDARD=LVCMOS33; #IO_L20P_T3_A08_D24_14
NET "in_byte<14>"       LOC=U11 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_A09_D25_VREF_14
NET "in_byte<15>"       LOC=V10 | IOSTANDARD=LVCMOS33; #IO_L21P_T3_DQS_14
```

Per il reset del moltiplicatore e per inserire in istanti diversi i due fattori sono stati utilizzati i seguenti bottoni:

```
## Buttons
NET "reset"             LOC=M17 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_D15_14
NET "load_second"         LOC=M18 | IOSTANDARD=LVCMOS33; #IO_L4N_T0_D05_14
NET "load_first"          LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14
```

Per mostrare il risultato su display sono stati utilizzati tutti gli anodi e i catodi presenti sulla board ed un led per l'eventuale riporto:

```
## 7 segment display
NET "cathodes<0>"           LOC=T10 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_A00_D16_14
NET "cathodes<1>"           LOC=R10 | IOSTANDARD=LVCMOS33; #IO_25_14
NET "cathodes<2>"           LOC=K16 | IOSTANDARD=LVCMOS33; #IO_25_15
NET "cathodes<3>"           LOC=K13 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A26_15
NET "cathodes<4>"           LOC=P15 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_14
NET "cathodes<5>"           LOC=T11 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A10_D26_14
NET "cathodes<6>"           LOC=L18 | IOSTANDARD=LVCMOS33; #IO_L4P_T0_D04_14
NET "cathodes<7>"           LOC=H15 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_A21_VREF_15

NET "anodes<0>"         LOC=J17 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_FOE_B_15
NET "anodes<1>"         LOC=J18 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_FWE_B_15
NET "anodes<2>"         LOC=T9 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_A01_D17_14
NET "anodes<3>"         LOC=J14 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A22_15
NET "anodes<4>"         LOC=P14 | IOSTANDARD=LVCMOS33; #IO_L8N_T1_D12_14
NET "anodes<5>"         LOC=T14 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_14
NET "anodes<6>"         LOC=K2 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_35
NET "anodes<7>"         LOC=U13 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_A02_D18_14
```

# 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

Tra gli esercizi proposti, si è scelto di realizzare il divisore restoring. Data la complessità dell'esercizio, è stato necessario suddividere l'architettura del componente in due parti: unità operativa e unità di controllo. Le due unità cooperano in modo da realizzare l'algoritmo di divisione restoring. In particolare, i segnali di uscita dell'unità di controllo pilotano l'unità operativa. Per risolvere problemi di tempificazione, si è scelto di abilitare l'unità di controllo sul fronte di salita del clock, mentre i componenti dell'unità operativa sul fronte di discesa. In questo modo, non solo l'unità operativa viene abilitata quando i segnali generati dall'unità di controllo sono ormai stabili, ma anche l'unità di controllo valuta i segnali di uscita dell'unità operativa solo dopo che essi si sono stabilizzati.

L'unità operativa, spesso indicata anche con il termine datapath, prevede i seguenti componenti: registri SAQ e M, un'unità aritmetica adder-subtractor, in grado di effettuare somma e sottrazione, ed un contatore modulo 4.

Il componente RegisterSAQ è un registro di 9 bit attivo sul fronte di discesa del segnale di clock. Dal punto di vista logico può essere scomposto in tre sotto-registri:

• Registro S: registro di 1 bit contenente l'informazione relativa al segno del risultato delle operazioni aritmetiche effettuate dall'adder-subtracter.

• Registro A: registro di 4 bit contenente inizialmente i 4 bit pari a zero. Al termine dell'algoritmo conterrà il resto della divisione.

• Registro Q: registro di 4 bit contenente inizialmente i 4 bit del dividendo. Al termine dell'algoritmo conterrà il quoziente della divisione.

Si è scelto di descrivere il registro attraverso il livello di astrazione behavioural, in modo da gestirne la complessità. Difatti, per poter implementare correttamente l'algoritmo di divisone restoring, il registro SAQ può operare in diverse modalità. La modalità di funzionamento è stabilita dai seguenti segnali di ingresso:

• INIT: consente l'inizializzazione del registro al valore X_INIT.

• LOAD_SA: consente il caricamento dei bit dei sotto-registri S ed A, mediante il valore di input X_SA.

• LOAD_Q: consente il caricamento del bit meno significato del sotto-registro Q, mediante il valore di input X_Q.

• SHIFT: consente di utilizzare il registro SAQ come un registro a scorrimento effettuando il left-shift di un bit del contenuto.

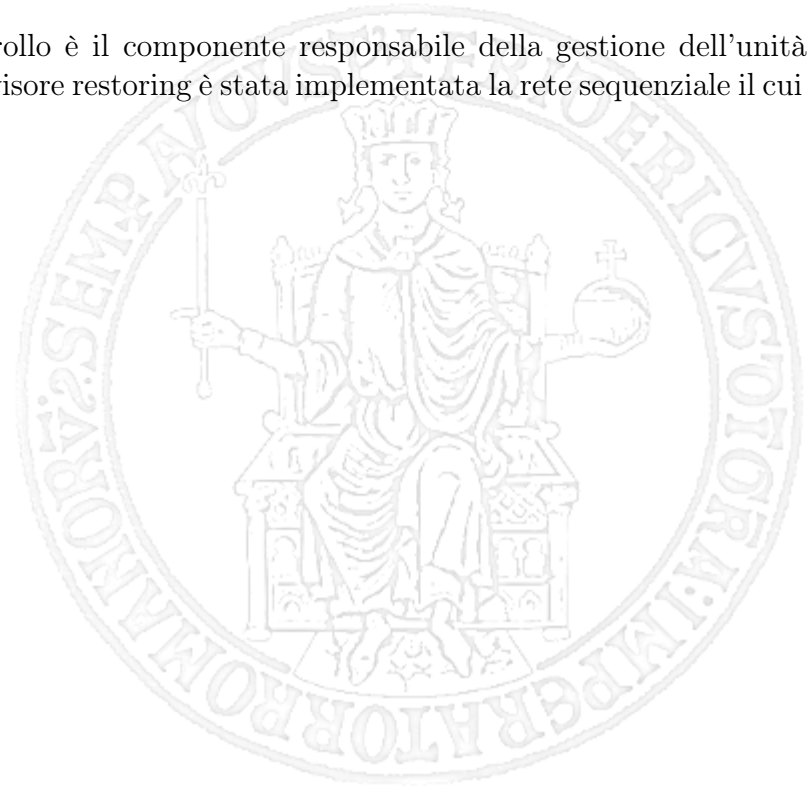
Il componente RegisterM (behavioural) è un registro parallelo-parallelo di 4 bit attivo sul fronte di discesa del segnale di clock, contenente il divisore dell'operazione.

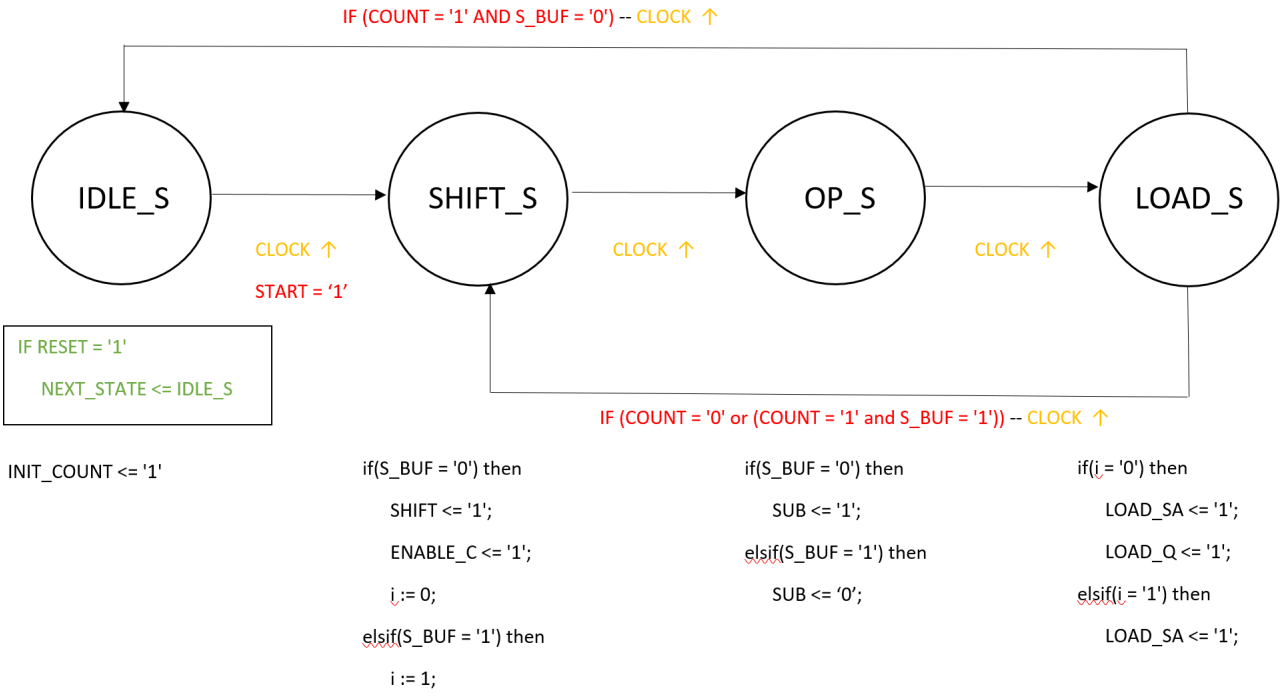
Il componente Adder_Subtractor_Nbit (structural) rappresenta l'unità aritmetica della parte operativa. È in grado di effettuare somma o sottrazione tra due operandi rappresentati in complemento a due, sulla base del segnale SUB, generato dall'unità di controllo. Se tale segnale è basso, viene eseguita la somma, viceversa, viene eseguita la sottrazione.

Il componente Counter (behavioural) rappresenta un contatore modulo 4, il cui scopo è quello di tenere traccia del numero di operazioni aritmetiche effettuate dall'addersubtracter. Quando il segnale di conteggio raggiunge il valore massimo, viene segnalata la terminazione dell'algoritmo di divisione, alzando l'opportuno segnale di uscita COUNT.

.

L'unità di controllo è il componente responsabile della gestione dell'unità operativa. Per la realizzazione del divisore restoring è stata implementata la rete sequenziale il cui automa è riportato in figura.






L'automa evolve dunque attraverso i seguenti stati:

• IDLE_S: stato di inattività della macchina.

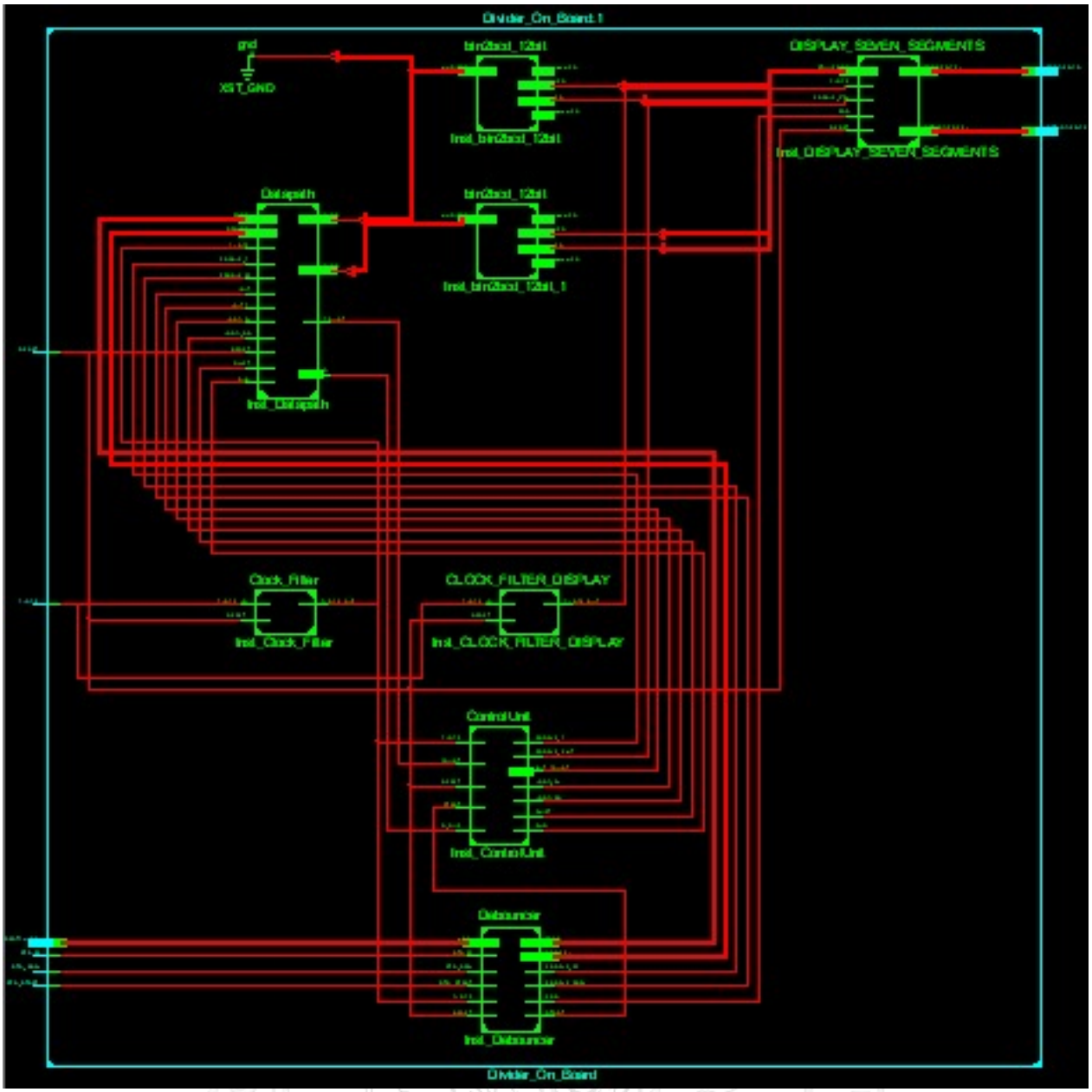
• SHIFT_S: stato in cui viene abilitato il conteggio e lo shift del registro SAQ.

• OP_S: stato in cui viene impostato il valore del segnale SUB, a seconda del bit più significativo del sotto-registro S. Se tale bit è basso viene effettuate la sottrazione, viceversa viene effettuata la somma.

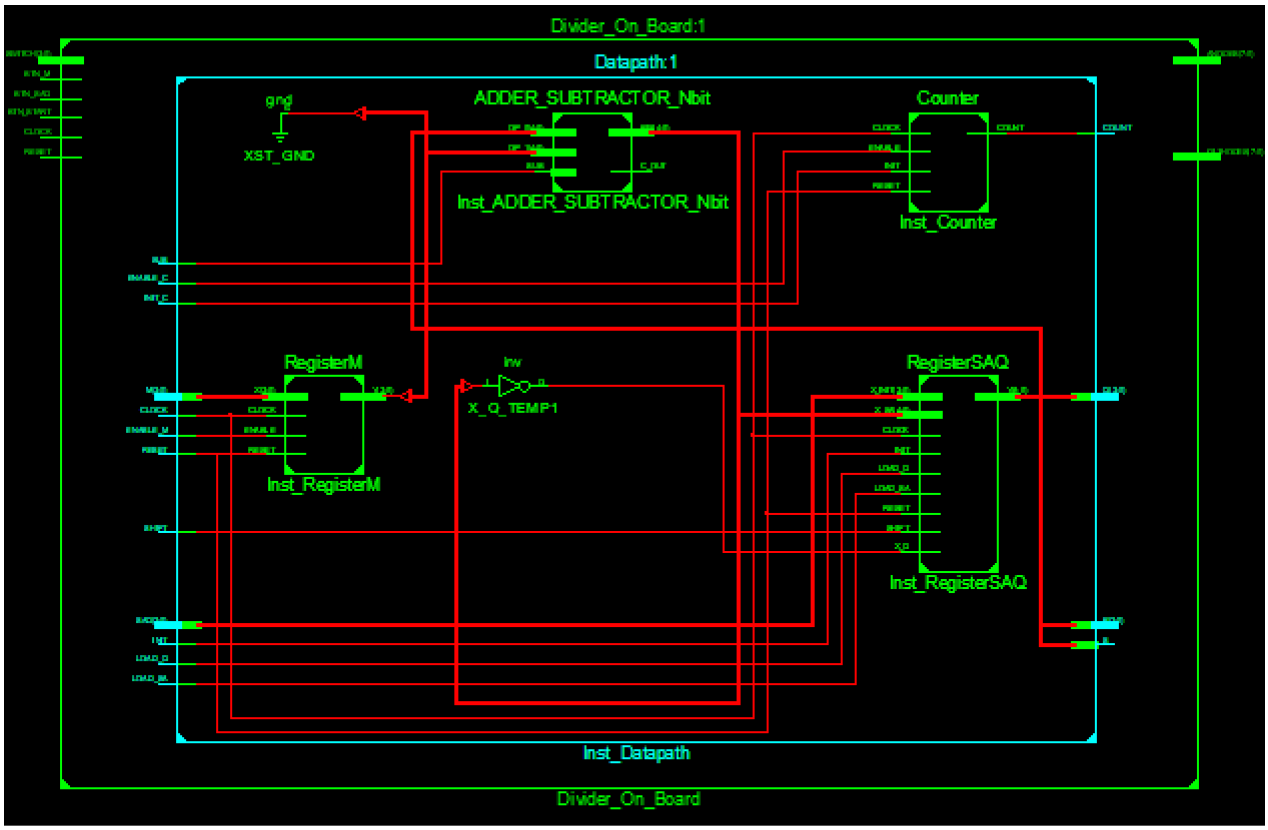
• LOAD_S: stato in cui viene abilitato il caricamento dei sotto-registri S, A e Q. In particolare, in S e A viene caricato il risultato dell'operazione aritmetica eseguita, mentre in Q viene caricato l'opposto del bit di segno del risultato. In tale stato viene inoltre valutata la condizione di terminazione dell'algoritmo (COUNT = '1'). Se la condizione è verificata, si passa allo stato di uscita, altrimenti si torna allo stato di shift.

### 10.2.1 Schematici

Schematico generale composto da: Datapath, Control Unit, Clock Filter, Convertitori e Display Seven Segments.




Schematico del componente Datapath.




## 10.2.2 Codice

### 10.2.2.1 Divisore_restoring

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity RegisterSAQ is
5    generic( N   : integer := 4);
6    port(   CLOCK      : in    STD_LOGIC;  -- Clock
7         RESET      : in    STD_LOGIC;  -- Reset
8         X_INIT     : in    STD_LOGIC_VECTOR(N-1 downto 0); -- Ingresso
              inizializzazione
9         X_SA       : in    STD_LOGIC_VECTOR(N downto 0); -- Ingresso SA
10        X_Q      : in    STD_LOGIC;  -- Ingresso Q
11        INIT       : in    STD_LOGIC;  -- Segnale inizializzazione
12        LOAD_SA    : in    STD_LOGIC;  -- Segnale caricamento SA
13        LOAD_Q     : in    STD_LOGIC;  -- Segnale caricamento Q(0)
14        SHIFT      : in    STD_LOGIC;  -- Segnale shift
15        Y        : out STD_LOGIC_VECTOR(2*N downto 0));  -- Contenuto del
              registro
16 end RegisterSAQ;
17
18 architecture Behavioral of RegisterSAQ is
19
```

```
20    signal Y_TEMP : STD_LOGIC_VECTOR(2*N downto 0);
21
22    begin
23      REG_SAQ : process(CLOCK, RESET)
24      begin
25        if(RESET = '1') then
26          Y_TEMP  <= (others => '0');
27        elsif(falling_edge(CLOCK)) then
28          if(INIT = '1') then -- Inizializzazione
29            Y_TEMP <= "00000" & X_INIT;
30          end if;
31          if(LOAD_SA = '1') then  -- Caricamento S e A
32            Y_TEMP(2*N downto N) <= X_SA;
33          end if;
34          if(LOAD_Q = '1') then -- Caricamento Q(0)
35            Y_TEMP(0) <= X_Q;
36          end if;
37          if(SHIFT = '1') then  -- Left shift
38  --          Y_TEMP(0) <= '0';
39            Y_TEMP(2*N-1 downto 1) <= Y_TEMP(2*N-2 downto 0);
40          end if;
41        end if;
42      end process REG_SAQ;
43
44      Y <= Y_TEMP;
45
46  end Behavioral;
```

Codice Componente 10.1: Definizione del componente RegisterSAQ

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity RegisterM is
5     generic( N  : integer := 4);
6     port( CLOCK : in STD_LOGIC;
7         RESET : in STD_LOGIC;
8         ENABLE : in STD_LOGIC;
9         X : in STD_LOGIC_VECTOR(N-1 downto 0);
10        Y : out STD_LOGIC_VECTOR(N-1 downto 0));
11  end RegisterM;
12
13  architecture Behavioral of RegisterM is
14
15    begin
16      REG_M : process(CLOCK, RESET)
17      begin
18        if (RESET='1') then
19          Y<=(others =>'0');
```

```
20        elsif (falling_edge(CLOCK)) then
21          if(ENABLE='1') then
22            Y <=  X;
23          end if;
24        end if;
25      end process REG_M;
26
27  end Behavioral;
```

Codice Componente 10.2: Definizione del componente RegisterM

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity RCA_Nbit is
5     generic(  n : POSITIVE := 4);
6     port( A   : in  STD_LOGIC_VECTOR (n-1 downto 0);
7         B   : in  STD_LOGIC_VECTOR (n-1 downto 0);
8         CIN : in  STD_LOGIC;
9         S   : out STD_LOGIC_VECTOR (n-1 downto 0);
10        COUT  : out STD_LOGIC);
11  end RCA_Nbit;
12
13  architecture Structural of RCA_Nbit is
14
15    component FULL_ADDER
16      port( X   : in STD_LOGIC;
17          Y   : in STD_LOGIC;
18          C_IN  : in STD_LOGIC;
19          Z   : out STD_LOGIC;
20          C_OUT : out STD_LOGIC);
21    end component;
22
23    signal C : STD_LOGIC_VECTOR (n-2 downto 0);
24
25    begin
26      FULL_ADDER_ALL  : for i in 0 to n-1 generate
27        LEAST: if i = 0 generate
28          l: FULL_ADDER
29            port map( X   => A(i),
30                    Y     => B(i),
31                    C_IN  => CIN,
32                    C_OUT => C(i),
33                    Z   => S(i));
34          end generate;
35
36        REST: if (i > 0 and i < n-1) generate
37          r: FULL_ADDER
38            port map( X   => A(i),
```

```
39                    Y    => B(i),
40                    C_IN  => C(i-1),
41                    C_OUT => C(i),
42                    Z    => S(i));
43          end generate;
44
45        MOST: if i = n-1 generate
46          M: FULL_ADDER
47            port map( X    => A(i),
48                    Y    => B(i),
49                    C_IN  => C(i-1),
50                    C_OUT => COUT,
51                    Z    => S(i));
52            end generate;
53        end generate;
54
55  end Structural;
```

Codice Componente 10.3: Definizione del componente RCA$_N bit$

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity ADDER_SUBTRACTOR_Nbit is
5     generic(  N : POSITIVE := 5);
6     port(   OP_0     : in STD_LOGIC_VECTOR (N-1 downto 0);
7         OP_1     : in STD_LOGIC_VECTOR (N-1 downto 0);
8         SUB    : in STD_LOGIC;
9         RIS    : out STD_LOGIC_VECTOR (N-1 downto 0);
10        C_OUT    : out STD_LOGIC
11        );
12  end ADDER_SUBTRACTOR_Nbit;
13
14  architecture structural of ADDER_SUBTRACTOR_Nbit is
15
16    component RCA_Nbit
17      generic( N : positive);
18      port(   A      : in STD_LOGIC_VECTOR(N-1 downto 0);
19          B      : in STD_LOGIC_VECTOR(N-1 downto 0);
20          CIN    : in STD_LOGIC;
21          S      : out STD_LOGIC_VECTOR(N-1 downto 0);
22          COUT   : out STD_LOGIC);
23    end component;
24
25    signal OP_1_xor : std_logic_vector(N-1 downto 0);
26
27    begin
28
29      loo : process (OP_1, SUB)
```

```
30      begin
31        for i in 0 to N-1 loop
32          OP_1_xor(i) <= OP_1(i) xor SUB;
33        end loop;
34      end process;
35
36
37      Inst_RCA_Nbit: RCA_Nbit
38        generic map( N => N)
39        port map(    A    => OP_0,
40                     B     => OP_1_xor,
41                     S     => RIS,
42                     CIN => SUB,
43                     COUT  => C_OUT);
44
45  end Structural;
```

Codice Componente 10.4: Definizione del componente Adder$_S ubtractor_N bit$

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5   use IEEE.NUMERIC_STD.ALL;
6
7   entity Counter is
8     generic( M : integer := 4;
9           N : integer := 3);
10
11    port( CLOCK : in STD_LOGIC;
12        RESET : in STD_LOGIC;
13        ENABLE: in STD_LOGIC;
14        INIT  : in STD_LOGIC;
15        COUNT : out STD_LOGIC
16        );
17
18  end Counter;
19
20  architecture Behavioral of Counter is
21
22    signal COUNT_TEMP : STD_LOGIC_VECTOR(N-1 downto 0);
23
24    begin
25      COUNT_PROC : process(CLOCK, RESET)
26      begin
27        if(RESET = '1') then
28          COUNT_TEMP <= (others => '0');
29          COUNT <= '0';
30        elsif (falling_edge(CLOCK)) then
```

```
31          if(INIT = '1') then
32            COUNT_TEMP <= (others => '0');
33          elsif (ENABLE = '1') then
34            if (COUNT_TEMP = std_logic_vector(to_unsigned(M-1, COUNT_TEMP'
                  length))) then
35              COUNT_TEMP <= (others => '0');
36              COUNT <= '1';
37            else
38              COUNT_TEMP <= COUNT_TEMP + "1";
39              COUNT <= '0';
40            end if;
41          end if;
42        end if;
43      end process COUNT_PROC;
44
45  end Behavioral;
```

Codice Componente 10.5: Definizione del componente Counter

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity Datapath is
5     generic(N : integer := 4);
6     port(
7       CLOCK     : IN std_logic;
8       RESET     : IN std_logic;
9       SAQ     : IN std_logic_vector(N-1 downto 0);
10      M       : IN std_logic_vector(N-1 downto 0);
11      INIT      : IN std_logic;
12      ENABLE_M  : IN std_logic;
13      ENABLE_C  : IN std_logic;
14      INIT_C    : IN std_logic;
15      LOAD_SA     : IN std_logic;
16      LOAD_Q    : IN std_logic;
17      SHIFT     : IN std_logic;
18      SUB     : IN std_logic;
19      Q       : OUT std_logic_vector(N-1 downto 0);
20      R       : OUT std_logic_vector(N-1 downto 0);
21      S         : OUT std_logic;
22      COUNT     : OUT std_logic
23      );
24  end Datapath;
25
26  architecture Structural of Datapath is
27
28    COMPONENT RegisterM
29    generic( N: integer);
30    PORT(
```

```
31      CLOCK : IN std_logic;
32      RESET : IN std_logic;
33      ENABLE : IN std_logic;
34      X : IN std_logic_vector(N-1 downto 0);
35      Y : OUT std_logic_vector(N-1 downto 0)
36      );
37    END COMPONENT;
38
39    COMPONENT RegisterSAQ
40    generic( N  : integer);
41    PORT(
42      CLOCK : IN std_logic;
43      RESET : IN std_logic;
44      X_INIT : IN std_logic_vector(N-1 downto 0);
45      X_SA : IN std_logic_vector(N downto 0);
46      X_Q : IN std_logic;
47      INIT : IN std_logic;
48      LOAD_SA : IN std_logic;
49      LOAD_Q : IN std_logic;
50      SHIFT : IN std_logic;
51      Y : OUT std_logic_vector(2*N downto 0)
52      );
53    END COMPONENT;
54
55
56    COMPONENT ADDER_SUBTRACTOR_Nbit
57    generic( N  : integer);
58    PORT(
59      OP_0 : IN std_logic_vector(N-1 downto 0);
60      OP_1 : IN std_logic_vector(N-1 downto 0);
61      SUB : IN std_logic;
62      RIS : OUT std_logic_vector(N-1 downto 0);
63      C_OUT : OUT std_logic
64      );
65    END COMPONENT;
66
67    COMPONENT Counter
68    generic( M : integer;
69          N : integer);
70    PORT(
71      CLOCK : IN std_logic;
72      RESET : IN std_logic;
73      INIT  : in STD_LOGIC;
74      ENABLE : IN std_logic;
75      COUNT : OUT std_logic
76      );
77    END COMPONENT;
78
79    signal Y_TEMP : std_logic_vector(N*2 downto 0);
```

```
80    signal M_TEMP : std_logic_vector(N-1 downto 0);
81    signal OP_1_TEMP : std_logic_vector(N downto 0);
82    signal RIS_TEMP : std_logic_vector(N downto 0);
83    signal X_Q_TEMP : std_logic;
84
85  begin
86    Inst_RegisterM: RegisterM
87    generic map(N => N)
88    PORT MAP(
89      CLOCK => CLOCK,
90      RESET => RESET,
91      ENABLE => ENABLE_M,
92      X => M,
93      Y => M_TEMP
94    );
95
96    X_Q_TEMP <= not(RIS_TEMP(N));
97
98    Inst_RegisterSAQ: RegisterSAQ
99    generic map(N => N)
100   PORT MAP(
101     CLOCK => CLOCK,
102     RESET => RESET,
103     X_INIT => SAQ,
104     X_SA => RIS_TEMP,
105     X_Q => X_Q_TEMP,
106     INIT => INIT,
107     LOAD_SA => LOAD_SA,
108     LOAD_Q => LOAD_Q,
109     SHIFT => SHIFT,
110     Y => Y_TEMP(N*2 downto 0)
111   );
112
113   OP_1_TEMP <= "0" & M_TEMP;
114   Inst_ADDER_SUBTRACTOR_Nbit: ADDER_SUBTRACTOR_Nbit
115   generic map(N => N+1)
116   PORT MAP(
117     OP_0 => Y_TEMP(N*2 downto N),
118     OP_1 => OP_1_TEMP,
119     SUB => SUB,
120     RIS => RIS_TEMP
121   );
122
123   Inst_Counter: Counter
124   GENERIC MAP(M => 4,
125         N => 3)
126   PORT MAP(
127     CLOCK => CLOCK,
128     RESET => RESET,
```

```
129     ENABLE => ENABLE_C,
130     COUNT => COUNT,
131     INIT => INIT_C
132   );
133
134   Q <= Y_TEMP(N-1 downto 0);
135   R <= Y_TEMP(2*N-1 downto N);
136   S <= Y_TEMP(2*N);
137
138 end Structural;
```

Codice Componente 10.6: Definizione del componente Datapath

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity ControlUnit is
5    port( CLOCK       : in STD_LOGIC;
6        RESET       : in STD_LOGIC;
7        START       : in STD_LOGIC;
8        S_BUF       : in STD_LOGIC;
9        COUNT       : in STD_LOGIC;
10       LOAD_SA       : out STD_LOGIC;
11       LOAD_Q      : out STD_LOGIC;
12       SHIFT       : out STD_LOGIC;
13       ENABLE_C      : out STD_LOGIC;
14       ENABLE_EXIT   : out STD_LOGIC;
15       SUB       : out STD_LOGIC;
16       INIT_COUNT    : out STD_LOGIC
17   );
18 end ControlUnit;
19
20 architecture Behavioral of ControlUnit is
21
22   type STATE_TYPE is(
23     IDLE_S,
24     SHIFT_S,
25     OP_S,
26     LOAD_S
27   );
28
29 -- SEGNALI TEMPORANEI
30   signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
31
32   begin
33     CURR_STATE_PROC : process(CLOCK,RESET)
34     begin
35       if(RESET = '1') then
36         CURRENT_STATE <= IDLE_S;
```

```
37        elsif(rising_edge(CLOCK)) then
38          if(START = '1' and CURRENT_STATE = IDLE_S) then
39            CURRENT_STATE <= SHIFT_S;
40          else CURRENT_STATE <= NEXT_STATE;
41          end if;
42        end if;
43      end process CURR_STATE_PROC;
44
45  NEXT_STATE_PROC : process(CURRENT_STATE, S_BUF, COUNT)
46
47    variable i : integer := 0;
48
49  begin
50    LOAD_SA <= '0';
51    LOAD_Q <= '0';
52    SHIFT <= '0';
53    ENABLE_C <= '0';
54    ENABLE_EXIT <= '0';
55    INIT_COUNT <= '0';
56
57  case CURRENT_STATE is
58    when IDLE_S =>
59      INIT_COUNT <= '1';
60      NEXT_STATE <= IDLE_S;
61
62    when SHIFT_S =>
63      if(S_BUF = '0') then
64        SHIFT <= '1';
65        ENABLE_C <= '1';
66        i := 0;
67      elsif(S_BUF = '1') then
68        i := 1;
69      end if;
70      NEXT_STATE <= OP_S;
71
72    when OP_S =>
73      if(S_BUF = '0') then
74        SUB <= '1';
75      elsif(S_BUF = '1') then
76        SUB <= '0';
77      end if;
78      NEXT_STATE <= LOAD_S;
79
80    when LOAD_S =>
81      if(i=0) then
82        LOAD_SA <='1';
83        LOAD_Q <= '1';
84      elsif(i=1) then
85        LOAD_SA <='1';
```

```
86      end if;
87
88      if(COUNT = '0' or (COUNT = '1' and S_BUF = '1')) then
89        NEXT_STATE <= SHIFT_S;
90      elsif(COUNT = '1' and S_BUF = '0') then
91        --NEXT_STATE <= EXIT_S;
92        ENABLE_EXIT <= '1';
93        NEXT_STATE <= IDLE_S;
94      end if;
95
96  --  when EXIT_S =>
97  --    ENABLE_EXIT <= '1';
98  --    NEXT_STATE <= IDLE_S;
99
100 end case;
101     end process NEXT_STATE_PROC;
102 end Behavioral;
```

Codice Componente 10.7: Definizione del componente control$_unit$

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Debouncer is
6    generic (N : integer := 4);
7    port (CLOCK : in STD_LOGIC;
8        RESET : in STD_LOGIC;
9        X : in STD_LOGIC_VECTOR(N-1 downto 0); --switch
10       BTN_SAQ : in STD_LOGIC;
11       BTN_M : in STD_LOGIC;
12       BTN_START : in STD_LOGIC;
13       SAQ : out STD_LOGIC_VECTOR(N-1 downto 0);  -- dividendo
14       M : out STD_LOGIC_VECTOR(N-1 downto 0);  -- divisore
15       START : out STD_LOGIC;
16       ENABLE_SAQ : out STD_LOGIC;
17       ENABLE_M : out STD_LOGIC;
18       ERR : out STD_LOGIC
19       );
20 end Debouncer;
21
22 architecture Behavioral of Debouncer is
23
24   signal ERR_TEMP : STD_LOGIC;
25
26   constant CONST : integer := 0;
27   begin
28     DEB_PROC : process (CLOCK, RESET )
29     variable i : integer := 0;
```

```
30      begin
31  --      ENABLE_SAQ <= '0';
32  --      ENABLE_M <= '0';
33  --
34        if(RESET = '1') then
35          SAQ <= (others =>'0');
36          M <=(others =>'0');
37          ENABLE_SAQ <= '1';
38          ENABLE_M <= '1';
39          ERR_TEMP <= '0';
40        elsif(rising_edge(CLOCK)) then
41          if(BTN_SAQ='1') then
42            SAQ <= X;
43            ENABLE_SAQ <= '1';
44            i := 1;
45          elsif(BTN_M = '1') then
46            if (X(N-1 downto 0) = std_logic_vector(to_unsigned(CONST, X(N-1
                  downto 0)' length))) then
47              ERR_TEMP <= '1';
48            else
49              M <= X(N-1 downto 0);
50              ENABLE_M <= '1';
51              ERR_TEMP <= '0';
52            end if;
53          elsif(BTN_START = '1' and i = 1) then
54            if (ERR_TEMP = '0') then
55              START<= '1';
56              i := 0;
57            end if;
58            ENABLE_SAQ <= '0';
59            ENABLE_M <= '0';
60          else
61            ENABLE_SAQ <= '0';
62            ENABLE_M <= '0';
63            START <= '0';
64          end if;
65        end if;
66      end process DEB_PROC;
67
68      ERR <= ERR_TEMP;
69
70  end Behavioral;
```

Codice Componente 10.8: Definizione del componente Debouncer

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity ANODES_MANAGER is
```

```
    Port ( COUNTER : in  STD_LOGIC_VECTOR (1 downto 0);
           ANODES : out  STD_LOGIC_VECTOR (7 downto 0));
end ANODES_MANAGER;

architecture behavioral of ANODES_MANAGER is

  signal ANODES_SWITCHING : std_logic_vector(7 downto 0) := (others => '0');

  begin
    ANODES <= not ANODES_SWITCHING OR not "11111111";

    ANODES_PROCESS: process(COUNTER)
    begin
      case COUNTER is
        when "00" =>
          ANODES_SWITCHING <= "00000001";
        when "01" =>
          ANODES_SWITCHING <= "00000010";
        when "10" =>
          ANODES_SWITCHING <= "00000100";
        when "11" =>
          ANODES_SWITCHING <= "00001000";
        when others =>
          ANODES_SWITCHING <= (others => '0');
      end case;
    end process;

end behavioral;
```

Codice Componente 10.9: Definizione del componente $anodes_manager$

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

entity CATHODES_MANAGER is
    port ( COUNTER  : in  STD_LOGIC_VECTOR (1 downto 0);
        ERR     : in  STD_LOGIC;
           VALUE    : in  STD_LOGIC_VECTOR (15 downto 0);
           CATHODES : out  STD_LOGIC_VECTOR (7 downto 0));
end CATHODES_MANAGER;

architecture Behavioral of CATHODES_MANAGER is

  constant zero   : STD_LOGIC_VECTOR(6 downto 0) := "1000000";
  constant one    : STD_LOGIC_VECTOR(6 downto 0) := "1111001";
  constant two    : STD_LOGIC_VECTOR(6 downto 0) := "0100100";
  constant three  : STD_LOGIC_VECTOR(6 downto 0) := "0110000";
  constant four   : STD_LOGIC_VECTOR(6 downto 0) := "0011001";
  constant five   : STD_LOGIC_VECTOR(6 downto 0) := "0010010";
```

```
19  constant six    : STD_LOGIC_VECTOR(6 downto 0) := "0000010";
20  constant seven  : STD_LOGIC_VECTOR(6 downto 0) := "1111000";
21  constant eight  : STD_LOGIC_VECTOR(6 downto 0) := "0000000";
22  constant nine   : STD_LOGIC_VECTOR(6 downto 0) := "0010000";
23  constant a     : STD_LOGIC_VECTOR(6 downto 0) := "0001000";
24  constant b      : STD_LOGIC_VECTOR(6 downto 0) := "0000011";
25  constant c      : STD_LOGIC_VECTOR(6 downto 0) := "1000110";
26  constant d      : STD_LOGIC_VECTOR(6 downto 0) := "0100001";
27  constant e      : STD_LOGIC_VECTOR(6 downto 0) := "0000110";
28  constant f      : STD_LOGIC_VECTOR(6 downto 0) := "0001110";
29  constant r      : STD_LOGIC_VECTOR(6 downto 0) := "0101111";
30  constant empty  : STD_LOGIC_VECTOR(6 downto 0) := "1111111";
31
32  alias DIGIT_0 is VALUE (3 downto 0);
33  alias DIGIT_1 is VALUE (7 downto 4);
34  alias DIGIT_2 is VALUE (11 downto 8);
35  alias DIGIT_3 is VALUE (15 downto 12);
36
37  signal CATHODES_FOR_DIGIT   : STD_LOGIC_VECTOR(6 downto 0) := (others =>
        '0');
38  signal NIBBLE           : STD_LOGIC_VECTOR(4 downto 0) := (others => '0');
39  signal DOT          : STD_LOGIC;
40
41  begin
42
43    DIGIT_SWITCHING: process(COUNTER, VALUE, ERR)
44    begin
45      case COUNTER is
46        when "00" =>
47          if(ERR = '0') then
48            NIBBLE <= '0' & DIGIT_0;
49            DOT <= '0';
50          elsif(ERR = '1') then
51            NIBBLE <= "11101";
52            DOT <= '0';
53          end if;
54        when "01" =>
55          if(ERR = '0') then
56            NIBBLE <= '0' & DIGIT_1;
57            DOT <= '0';
58          elsif(ERR = '1') then
59            NIBBLE <= "11110";
60            DOT <= '1';
61          end if;
62        when "10" =>
63          if(ERR = '0') then
64            NIBBLE <= '0' & DIGIT_2;
65            DOT <= '1';
66          elsif(ERR = '1') then
```

```
67              NIBBLE <= "11110";
68              DOT <= '0';
69            end if;
70          when "11" =>
71            if(ERR = '0') then
72              NIBBLE <= '0' & DIGIT_3;
73              DOT <= '0';
74            elsif(ERR = '1') then
75              NIBBLE <= "11111";
76              DOT <= '0';
77            end if;
78          when others =>
79            NIBBLE <= (others => '0');
80        end case;
81      end process;
82
83      SEVENT_SEGMENT_DECODER_PROCESS: process(nibble)
84        begin
85         case NIBBLE is
86          when "00000" => CATHODES_FOR_DIGIT <= zero;
87          when "00001" => CATHODES_FOR_DIGIT <= one;
88          when "00010" => CATHODES_FOR_DIGIT <= two;
89          when "00011" => CATHODES_FOR_DIGIT <= three;
90          when "00100" => CATHODES_FOR_DIGIT <= four;
91          when "00101" => CATHODES_FOR_DIGIT <= five;
92          when "00110" => CATHODES_FOR_DIGIT <= six;
93          when "00111" => CATHODES_FOR_DIGIT <= seven;
94          when "01000" => CATHODES_FOR_DIGIT <= eight;
95          when "01001" => CATHODES_FOR_DIGIT <= nine;
96          when "01010" => CATHODES_FOR_DIGIT <= a;
97          when "01011" => CATHODES_FOR_DIGIT <= b;
98          when "01100" => CATHODES_FOR_DIGIT <= c;
99          when "01101" => CATHODES_FOR_DIGIT <= d;
100         when "01110" => CATHODES_FOR_DIGIT <= e;
101         when "01111" => CATHODES_FOR_DIGIT <= f;
102         when "11111" => CATHODES_FOR_DIGIT <= e;
103         when "11110" => CATHODES_FOR_DIGIT <= r;
104         when "11101" => CATHODES_FOR_DIGIT <= empty;
105         when others => CATHODES_FOR_DIGIT <= (others => '0');
106        end case;
107       end process SEVENT_SEGMENT_DECODER_PROCESS;
108
109
110     CATHODES <= not(DOT) & CATHODES_FOR_DIGIT;
111
112 end Behavioral;
```

Codice Componente 10.10: Definizione del componente cathodes$_m$*anager*

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity DISPLAY_SEVEN_SEGMENTS is
5      port( CLOCK    : in  STD_LOGIC;
6            RESET    : in  STD_LOGIC;
7            ENABLE_VAL : in STD_LOGIC;
8            ERR    : in  STD_LOGIC;
9            VALUE    : in  STD_LOGIC_VECTOR (15 downto 0);
10           ANODES   : out STD_LOGIC_VECTOR (7 downto 0);
11           CATHODES : out  STD_LOGIC_VECTOR (7 downto 0));
12 end DISPLAY_SEVEN_SEGMENTS;
13
14 architecture Structural of DISPLAY_SEVEN_SEGMENTS is
15
16   signal COUNTER : STD_LOGIC_VECTOR (1 downto 0);
17   signal TEMP_VAL : std_logic_vector(15 downto 0);
18   signal ERR_OUT : std_logic;
19
20   component COUNTER_MOD4
21     port( CLOCK    : in  STD_LOGIC;
22           RESET : in  STD_LOGIC;
23           COUNTER : out  STD_LOGIC_VECTOR (1 downto 0));
24   end component;
25
26   component CATHODES_MANAGER
27     port( COUNTER : IN STD_LOGIC_VECTOR (1 downto 0);
28           ERR      : in  STD_LOGIC;
29           VALUE : IN std_logic_vector(15 downto 0);
30           CATHODES : OUT std_logic_vector(7 downto 0));
31   end component;
32
33   component ANODES_MANAGER
34     port( COUNTER : IN  STD_LOGIC_VECTOR (1 downto 0);
35           ANODES : OUT std_logic_vector(7 downto 0));
36   end component;
37
38   begin
39
40   COUNTER_INSTANCE: COUNTER_MOD4 port map(
41     CLOCK   => CLOCK,
42     RESET => RESET,
43     COUNTER => COUNTER
44   );
45
46   CATHODES_INSTANCE: CATHODES_MANAGER port map(
47     COUNTER => COUNTER,
48     ERR => ERR_OUT,
49     VALUE   => TEMP_VAL,
```

```
50      CATHODES => CATHODES
51    );
52
53    ANODES_INSTANCE: ANODES_MANAGER port map(
54      COUNTER => COUNTER,
55      ANODES  => ANODES
56    );
57
58    VAL_PROC : process(CLOCK, RESET)
59    begin
60      if(RESET = '1') then
61        TEMP_VAL <= (others => '0');
62        ERR_OUT <= '0';
63      elsif(falling_edge(CLOCK)) then
64        if(ERR = '1') then
65          ERR_OUT <= '1';
66        elsif(ENABLE_VAL = '1') then
67          ERR_OUT <= '0';
68          TEMP_VAL <= VALUE;
69        end if;
70      end if;
71    end process VAL_PROC;
72
73  end Structural;
```

Codice Componente 10.11: Definizione del componente display$_s$even$_s$egments

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity Divider_On_Board is
5     GENERIC(  N : integer := 4);
6     port(   CLOCK : in  STD_LOGIC;
7         RESET : in STD_LOGIC;
8         SWITCH : in  std_logic_vector(3 downto 0);
9         BTN_SAQ : in STD_LOGIC;
10        BTN_M : in STD_LOGIC;
11        BTN_START : in STD_LOGIC;
12        ANODES : out std_logic_vector(7 downto 0);
13        CATHODES : out std_logic_vector(7 downto 0));
14  end Divider_On_Board;
15
16  architecture Structural of Divider_On_Board is
17    COMPONENT Datapath
18    GENERIC(N : integer);
19    PORT(
20      CLOCK     : IN std_logic;
21      RESET     : IN std_logic;
22      SAQ     : IN std_logic_vector(N-1 downto 0);
```

```
23      M       : IN std_logic_vector(N-1 downto 0);
24      INIT      : IN std_logic;
25      ENABLE_M  : IN std_logic;
26      ENABLE_C  : IN std_logic;
27      INIT_C    : IN std_logic;
28      LOAD_SA     : IN std_logic;
29      LOAD_Q    : IN std_logic;
30      SHIFT     : IN std_logic;
31      SUB     : IN std_logic;
32      Q       : OUT std_logic_vector(N-1 downto 0);
33      R       : OUT std_logic_vector(N-1 downto 0);
34      S         : OUT std_logic;
35      COUNT     : OUT std_logic
36      );
37    END COMPONENT;
38
39    COMPONENT ControlUnit
40    PORT(
41      CLOCK : IN std_logic;
42      RESET : IN std_logic;
43      START : IN std_logic;
44      S_BUF : IN std_logic;
45      COUNT : IN std_logic;
46      LOAD_SA : OUT std_logic;
47      LOAD_Q : OUT std_logic;
48      SHIFT : OUT std_logic;
49      ENABLE_C : OUT std_logic;
50      ENABLE_EXIT : OUT std_logic;
51      SUB : OUT std_logic;
52      INIT_COUNT    : out STD_LOGIC
53      );
54    END COMPONENT;
55
56    COMPONENT Clock_Filter
57    generic(  CLOCK_FREQUENCY_IN : integer;
58          CLOCK_FREQUENCY_OUT : integer);
59    PORT(
60      CLOCK_IN : IN std_logic;
61      RESET : IN std_logic;
62      CLOCK_OUT : OUT std_logic
63      );
64    END COMPONENT;
65
66    COMPONENT CLOCK_FILTER_DISPLAY
67    generic(  CLOCK_FREQUENCY_IN : integer;
68          CLOCK_FREQUENCY_OUT : integer);
69    PORT(
70      CLOCK_IN : IN std_logic;
71      RESET : IN std_logic;
```

```
72      CLOCK_OUT : OUT std_logic
73      );
74    END COMPONENT;
75
76    COMPONENT DISPLAY_SEVEN_SEGMENTS
77    PORT(
78      CLOCK : IN std_logic;
79      RESET : IN std_logic;
80      ERR   : in  STD_LOGIC;
81      ENABLE_VAL : IN std_logic;
82      VALUE : IN std_logic_vector(15 downto 0);
83      ANODES : OUT std_logic_vector(7 downto 0);
84      CATHODES : OUT std_logic_vector(7 downto 0)
85      );
86    END COMPONENT;
87
88
89    COMPONENT bin2bcd_12bit
90    PORT(
91      binIN : IN std_logic_vector(11 downto 0);
92      ones : OUT std_logic_vector(3 downto 0);
93      tens : OUT std_logic_vector(3 downto 0);
94      hundreds : OUT std_logic_vector(3 downto 0);
95      thousands : OUT std_logic_vector(3 downto 0)
96      );
97    END COMPONENT;
98
99    COMPONENT Debouncer
100   generic (N : integer);
101   PORT(
102     CLOCK : IN std_logic;
103     RESET : IN std_logic;
104     X : IN std_logic_vector(N-1 downto 0);
105     BTN_SAQ : IN std_logic;
106     BTN_M : IN std_logic;
107     BTN_START : IN std_logic;
108     SAQ : OUT std_logic_vector(N-1 downto 0);
109     M : OUT std_logic_vector(N-1 downto 0);
110     START : OUT std_logic;
111     ENABLE_SAQ : OUT std_logic;
112     ENABLE_M : OUT std_logic;
113     ERR : out STD_LOGIC
114     );
115   END COMPONENT;
116
117   signal CLOCK_FX : std_logic;
118   signal SAQ_TEMP : std_logic_vector(N-1 downto 0);
119   signal M_TEMP : std_logic_vector(N-1 downto 0);
120   signal INIT_TEMP : std_logic;
```

```
121   signal ENABLE_M_TEMP : std_logic;
122   signal LOAD_SA_TEMP : std_logic;
123   signal LOAD_Q_TEMP : std_logic;
124   signal SHIFT_TEMP : std_logic;
125   signal SUB_TEMP : std_logic;
126   signal START_TEMP : std_logic;
127   signal ENABLE_C_TEMP : std_logic;
128   signal COUNT_TEMP : std_logic;
129   signal S_BUF_TEMP : std_logic := '0';
130   signal Q_TEMP : std_logic_vector(N-1 downto 0);
131   signal R_TEMP : std_logic_vector(N-1 downto 0);
132   signal Q_CONV : std_logic_vector(11 downto 0);
133   signal R_CONV : std_logic_vector(11 downto 0);
134   signal VALUE  : std_logic_vector(15 downto 0);
135   signal ENABLE_EXIT : std_logic;
136   signal INIT_COUNT_TEMP  : STD_LOGIC;
137   signal ERR_TEMP : STD_LOGIC;
138
139
140 begin
141   Inst_Datapath: Datapath
142   GENERIC MAP(N => 4)
143   PORT MAP(
144     CLOCK => CLOCK_FX,
145     RESET => RESET,
146     SAQ => SAQ_TEMP,
147     M => M_TEMP,
148     INIT => INIT_TEMP,
149     ENABLE_M => ENABLE_M_TEMP,
150     LOAD_SA => LOAD_SA_TEMP,
151     LOAD_Q => LOAD_Q_TEMP,
152     SHIFT => SHIFT_TEMP,
153     SUB => SUB_TEMP,
154     ENABLE_C => ENABLE_C_TEMP,
155     INIT_C => INIT_COUNT_TEMP,
156     Q => Q_TEMP,
157     R => R_TEMP,
158     S => S_BUF_TEMP,
159     COUNT => COUNT_TEMP
160   );
161
162   Inst_ControlUnit: ControlUnit
163   PORT MAP(
164     CLOCK => CLOCK_FX,
165     RESET => RESET,
166     START => START_TEMP,
167     S_BUF => S_BUF_TEMP,
168     COUNT => COUNT_TEMP,
169     LOAD_SA => LOAD_SA_TEMP,
```

```
170     LOAD_Q => LOAD_Q_TEMP,
171     SHIFT => SHIFT_TEMP,
172     ENABLE_C => ENABLE_C_TEMP,
173     ENABLE_EXIT => ENABLE_EXIT,
174     SUB => SUB_TEMP,
175     INIT_COUNT => INIT_COUNT_TEMP
176   );
177
178   Inst_Clock_Filter: Clock_Filter
179   GENERIC MAP(
180     CLOCK_FREQUENCY_IN => 50000000,
181     CLOCK_FREQUENCY_OUT => 500
182   )
183   PORT MAP(
184     CLOCK_IN => CLOCK,
185     RESET => RESET,
186     CLOCK_OUT => CLOCK_FX
187   );
188
189
190   Inst_Debouncer: Debouncer
191   GENERIC MAP(N => 4)
192   PORT MAP(
193     CLOCK => CLOCK_FX,
194     RESET => RESET,
195     X => SWITCH,
196     BTN_SAQ => BTN_SAQ,
197     BTN_M => BTN_M,
198     BTN_START => BTN_START,
199     SAQ => SAQ_TEMP,
200     M => M_TEMP,
201     START => START_TEMP,
202     ENABLE_SAQ => INIT_TEMP,
203     ENABLE_M => ENABLE_M_TEMP,
204     ERR => ERR_TEMP
205   );
206
207   Inst_CLOCK_FILTER_DISPLAY: CLOCK_FILTER_DISPLAY
208   GENERIC MAP(
209     CLOCK_FREQUENCY_IN => 50000000,
210     CLOCK_FREQUENCY_OUT => 500
211   )
212   PORT MAP(
213     CLOCK_IN => CLOCK,
214     RESET => RESET,
215     CLOCK_OUT => CLOCK_FX_1
216   );
217
218   Inst_DISPLAY_SEVEN_SEGMENTS: DISPLAY_SEVEN_SEGMENTS PORT MAP(
```

```
219     CLOCK => CLOCK_FX_1,
220     RESET => RESET,
221     ERR => ERR_TEMP,
222     ENABLE_VAL => ENABLE_EXIT,
223     VALUE => VALUE,
224     ANODES => ANODES,
225     CATHODES => CATHODES
226   );
227
228   Q_CONV <= "00000000" & Q_TEMP;
229   Inst_bin2bcd_12bit: bin2bcd_12bit PORT MAP(
230     binIN => Q_CONV,
231     ones => value(11 downto 8),
232     tens => value(15 downto 12)
233   );
234
235   R_CONV <= "00000000" & R_TEMP;
236   Inst_bin2bcd_12bit_1: bin2bcd_12bit PORT MAP(
237     binIN => R_CONV,
238     ones => value(3 downto 0),
239     tens => value(7 downto 4)
240   );
241
242 end Structural;
```

Codice Componente 10.12: Definizione del componente Divider$_{O}n_{B}oard$

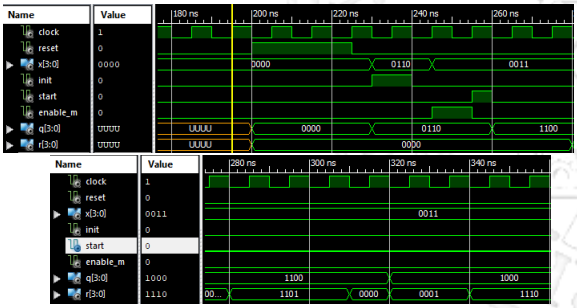
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY divider_tb IS
5  END divider_tb;
6
7  ARCHITECTURE behavior OF divider_tb IS
8
9      -- Component Declaration for the Unit Under Test (UUT)
10
11     COMPONENT divider
12     PORT(
13          CLOCK : IN  std_logic;
14          RESET : IN  std_logic;
15          X : IN  std_logic_vector(3 downto 0);
16          INIT : IN  std_logic;
17          START : IN  std_logic;
18          ENABLE_M : IN  std_logic;
19          Q : OUT  std_logic_vector(3 downto 0);
20          R : OUT  std_logic_vector(3 downto 0)
21         );
22     END COMPONENT;
```

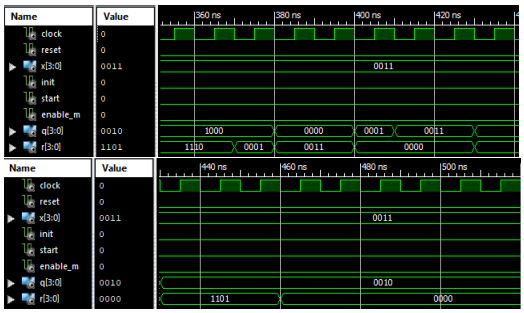
```
23
24
25      --Inputs
26      signal CLOCK : std_logic := '0';
27      signal RESET : std_logic := '0';
28      signal X : std_logic_vector(3 downto 0) := (others => '0');
29      signal INIT : std_logic := '0';
30      signal START : std_logic := '0';
31      signal ENABLE_M : std_logic := '0';
32
33     --Outputs
34      signal Q : std_logic_vector(3 downto 0);
35      signal R : std_logic_vector(3 downto 0);
36
37      -- Clock period definitions
38      constant CLOCK_period : time := 10 ns;
39
40  BEGIN
41
42     -- Instantiate the Unit Under Test (UUT)
43      uut: divider PORT MAP (
44              CLOCK => CLOCK,
45              RESET => RESET,
46              X => X,
47              INIT => INIT,
48              START => START,
49              ENABLE_M => ENABLE_M,
50              Q => Q,
51              R => R
52            );
53
54      -- Clock process definitions
55      CLOCK_process :process
56      begin
57       CLOCK <= '0';
58       wait for CLOCK_period/2;
59       CLOCK <= '1';
60       wait for CLOCK_period/2;
61      end process;
62
63
64      -- Stimulus process
65      stim_proc: process
66      begin
67         -- hold reset state for 100 ns.
68         wait for 100 ns;
69
70         wait for CLOCK_period*10;
71
```

```
72        -- insert stimulus here
73
74      reset<='1';
75      wait for 25 ns;
76      reset<='0';
77
78      wait for 5 ns;
79      x<="0110";
80      init<='1';
81      wait for 10 ns;
82
83      init<='0';
84
85      wait for 5 ns;
86
87      x<="0011";
88      enable_m<='1';
89
90      wait for 10 ns;
91
92      enable_m<='0';
93      start<='1';
94
95      wait for 5 ns;
96
97      start<='0';
98
99        wait;
100    end process;
101
102 END;
```

Codice Componente 10.13: Definizione del componente divider$_t b$

## 10.3 Simulazione






Attraverso le varie figure possiamo notare passo dopo passo, a seguito di operazioni di shift, add e sub, i valori assunti dai registri contenenti resto e quoziente.

## 10.4 Sintesi su board FPGA

Per interfacciare il design realizzato con l'I/O di board si è scelta la seguente soluzione:

Per il clock si è deciso di utilizzare il clock della scheda, opportunamente filtrato poi per permettere il corretto svolgimento di tutte le operazioni.

```
## Clock signal
NET "clock"   LOC = "E3"  | IOSTANDARD = "LVCMOS33";
```

Per gli ingressi sono stati utilizzati i 4 switch più a destra nella board.

```
## Switches
NET "SWITCH<0>"          LOC=J15 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_RS0_15
NET "SWITCH<1>"          LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "SWITCH<2>"          LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
NET "SWITCH<3>"          LOC=R15 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_14
```

Per resettare l'uscita e per inserire in istanti diversi dividendo e divisore sono stati utilizzati i seguenti bottoni.

```
## Buttons
NET "BTN_SAQ"           LOC=M17 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_D15_14
NET "BTN_M"           LOC=M18 | IOSTANDARD=LVCMOS33; #IO_L4N_T0_D05_14
NET "BTN_START"           LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14

#NET "RESET"     LOC=C12 | IOSTANDARD=LVCMOS33; #IO_L3P_T0_DQS_AD1P_15

NET "RESET"           LOC=P18 | IOSTANDARD=LVCMOS33; #IO_L9N_T1_DQS_D13_14
#NET "btnl"          LOC=P17 | IOSTANDARD=LVCMOS33; #IO_L12P_T1_MRCC_14
```

Per mostrare l'output vengono utilizzati tutti gli anodi e i catodi del display.

```
## 7 segment display
NET "cathodes<0>"              LOC=T10 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_A00_D16_14
NET "cathodes<1>"              LOC=R10 | IOSTANDARD=LVCMOS33; #IO_25_14
NET "cathodes<2>"              LOC=K16 | IOSTANDARD=LVCMOS33; #IO_25_15
NET "cathodes<3>"              LOC=K13 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A26_15
NET "cathodes<4>"              LOC=P15 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_14
NET "cathodes<5>"              LOC=T11 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A10_D26_14
NET "cathodes<6>"              LOC=L18 | IOSTANDARD=LVCMOS33; #IO_L4P_T0_D04_14
NET "cathodes<7>"              LOC=H15 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_A21_VREF_15

NET "anodes<0>"          LOC=J17 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_FOE_B_15
NET "anodes<1>"          LOC=J18 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_FWE_B_15
NET "anodes<2>"          LOC=T9 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_A01_D17_14
NET "anodes<3>"          LOC=J14 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A22_15
NET "anodes<4>"          LOC=P14 | IOSTANDARD=LVCMOS33; #IO_L8N_T1_D12_14
NET "anodes<5>"          LOC=T14 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_14
NET "anodes<6>"          LOC=K2 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_35
NET "anodes<7>"          LOC=U13 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_A02_D18_14
```

# Capitolo 11

# Esercizio 11

## 11.1 Traccia

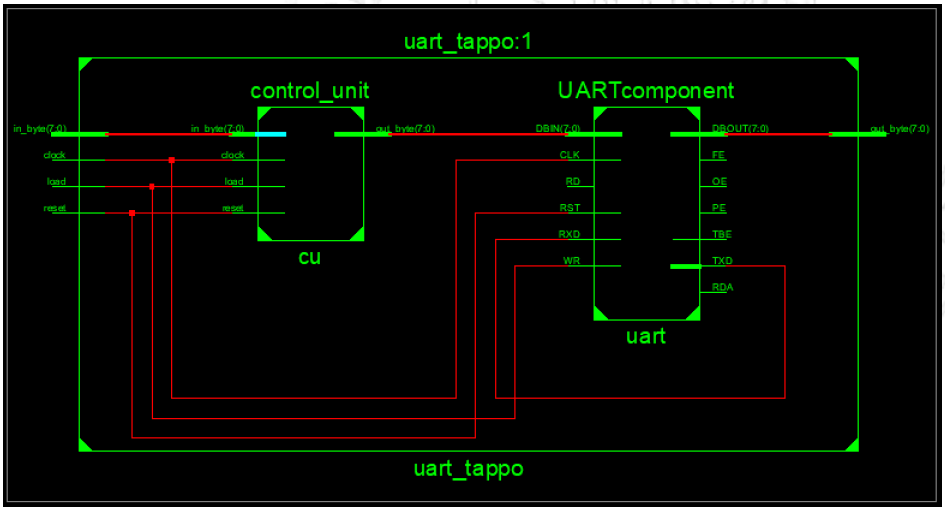
Sfruttando l'implementazione fornita dalla Digilent di un dispositivo UART (componente RS232RefComp.vhd), progettare ed implementare in VHDL i seguenti componenti:

- 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.

## 11.2 Soluzione

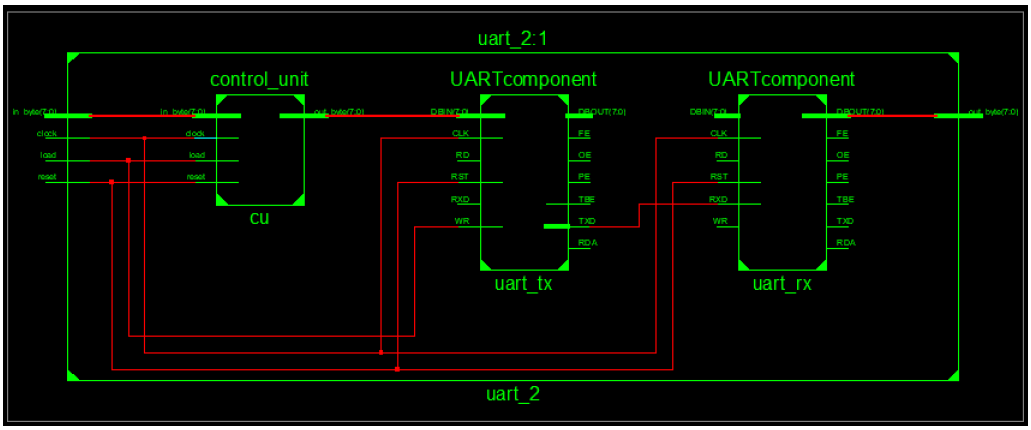
### 11.2.1 Schematici

- Schematico componente uart_tappo



L'UART_TAPPO è composto da una control unit, che ha il compito di inviare il dato parallelo da serializzare all'ingresso del lato trasmettitore dell'UARTcomponent, e dall'UARTcomponent stesso che ha l'uscita seriale del trasmettitore TXD collegata all'ingresso seriale del ricevitore RXD nella cosiddetta configurazione a tappo.

- Schematico 2_uart




L'UART_2 è anch'esso composto da una control_unit identica a quella del componente precedente, ma da 2 UARTcomponent, dove u1 ha il compito di trasmettere il dato e U2 quello di riceverlo.

## 11.2.2 Codice

### 11.2.2.1 UART

```
1  ----------------------------------------------------------------------
2  -- uartcomponent.vhd
3  ----------------------------------------------------------------------
4  -- Author:  Dan Pederson
5  --          Copyright 2004 Digilent, Inc.
6  ----------------------------------------------------------------------
7  -- Description: This file defines a UART which transfers data to and
8  --        from serial and parallel information.  It requires two
9  --        major processes:  receiving and transferring.  The
10 --        receiving portion reads serially transmitted data, and
11 --        converts it into parallel data, while the transferring
12 --        portion reads parallel data, and transmits it as serial
13 --        data.  There are three error signals provided with this
14 --        UART.  They are frame error, parity error, and overwrite
15 --        error signals.  This UART is configured to use an ODD
16 --        parity bit at a baud rate of 9600.
17 --
18 ----------------------------------------------------------------------
19 -- Revision History:
20 --    07/15/04 (DanP) Created
21 --    05/24/05 (DanP) Updated commenting style
22 --    06/06/05 (DanP) Synchronized state machines to fix timing bug
```

```
23 ----------------------------------------------------------------------------
24
25 library IEEE;
26 use IEEE.STD_LOGIC_1164.ALL;
27 use IEEE.STD_LOGIC_ARITH.ALL;
28 use IEEE.STD_LOGIC_UNSIGNED.ALL;
29
30 ----------------------------------------------------------------------------
31 --
32 --Title:  UARTcomponent entity
33 --
34 --Inputs: 7 : RXD porta che trasmette dati seriali al ricevitore
35 --          CLK
36 --          DBIN porta che trasmette byte parallelo al trasmettitore
37 --          RDA
38 --          RD
39 --          WR
40 --          RST
41 --
42 --Outputs:  7 : TXD byte seriale trasmesso dal trasmettitore
43 --          DBOUT byte parallelo trasmesso dal ricevitore
44 --          RDA
45 --          TBE
46 --          PE
47 --          FE
48 --          OE
49 --
50 --Description:  This describes the UART component entity.  The inputs are
51 --        the Pegasus 50 MHz clock, a reset button, The RXD from
52 --        the serial cable, an 8-bit data bus from the parallel
53 --        port, and Read Data Available (RDA)and Transfer Buffer
54 --        Empty(TBE) handshaking signals.  The outputs are the TXD
55 --        signal for the serial port, an 8-bit data bus for the
56 --        parallel port, RDA and TBE handshaking signals, and three
57 --        error signals for parity, frame, and overwrite errors.
58 --
59 ----------------------------------------------------------------------------
60 entity UARTcomponent is
61   Generic (
62     --@48MHz
63 --    BAUD_DIVIDE_G : integer := 26;  --115200 baud
64 --    BAUD_RATE_G   : integer := 417
65
66     --@26.6MHz
67     BAUD_DIVIDE_G : integer := 14;  --115200 baud
68     BAUD_RATE_G   : integer := 231
69   );
70   Port (
```

```
71      TXD    : out    std_logic    := '1';          -- Transmitted serial data
           output
72      RXD    : in     std_logic;               -- Received serial data input
73      CLK    : in     std_logic;               -- Clock signal
74      DBIN   : in     std_logic_vector (7 downto 0);    -- Input parallel data
           to be transmitted
75      DBOUT    : out    std_logic_vector (7 downto 0);    -- Recevived parallel
           data output
76      RDA    : inout  std_logic;               -- Read Data Available, dato
           disponibile in lettura
77      TBE    : out    std_logic    := '1';          -- Transfer Buffer Empty
78      RD     : in     std_logic;               -- Read Strobe, abilito la lettura
79      WR     : in     std_logic;               -- Write Strobe, abilito la
           scrittura
80      PE     : out    std_logic;               -- Parity error
81      FE     : out    std_logic;               -- Frame error
82      OE     : out    std_logic;               -- Overwrite error
83      RST    : in     std_logic := '0');         -- Reset signal
84
85  end UARTcomponent;
86
87  architecture Behavioral of UARTcomponent is
88
89  ---------------------------------------------------------------------
90  -- Local Type and Signal Declarations
91  ---------------------------------------------------------------------
92
93  ---------------------------------------------------------------------
94  --Title:  Local Type Declarations
95  --
96  --Description:  There are two state machines used in this entity.  The
97  --         rstate is used to synchronize the receiving portion of
98  --         the UART, and the tstate is used to synchronize the
99  --         sending portion of the UART.
100 --
101 ---------------------------------------------------------------------
102   type rstate is (
103     strIdle,
104     strEightDelay,
105     strGetData,
106     strWaitFor0,
107     strWaitFor1,
108     strCheckStop
109   );
110
111   type tstate is (
112     sttIdle,
113     sttTransfer,
114     sttShift,
```

```
115     sttDelay,
116     sttWaitWrite
117     );
118 
119 ----------------------------------------------------------------------
120 --
121 --Title:  Local Signal Declarations
122 --
123 --Description:  The constants and signals used by this entity are
124 --        described below:
125 --
126 --        -baudRate : This is the Baud Rate constant used to
127 --                synchronize the Pegasus 50 MHz clock with a
128 --                baud rate of 9600.  To get this number, divide
129 --                50MHz by 9600.
130 --        -baudDivide :   This is the Baud Rate divider used to safely
131 --                read data transmitted at a baud rate of 9600.
132 --                It is simply the above described baudRate
133 --                constant divided by 16.
134 --
135 --        -rdReg    : this is the receive holding register
136 --        -rdSReg   : this is the receive shift register
137 --        -tfReg    : this is the transfer holding register
138 --        -tfSReg   : this is the transfer shifting register
139 --        -clkDiv   : counter used to get rClk
140 --        -ctr    : used for delay times
141 --        -tfCtr    : used to delay in the transfer process
142 --        -dataCtr  : counts the number of read data bits
143 --        -parError : parity error bit
144 --        -frameError : frame error bit
145 --        -CE     : clock enable bit for the writing latch
146 --        -ctRst      : reset for the ctr
147 --        -load   : load signal used to load the transfer shift
148 --                register
149 --        -shift    : shift signal used to unload the transfer
150 --                shift register
151 --        -par    : represents the parity in the transfer
152 --                holding register
153 --        -tClkRST  : reset for the tfCtr
154 --        -rShift   : shift signal used to load the receive shift
155 --                register
156 --        -dataRST  : reset for the dataCtr
157 --        -dataIncr : signal to increment the dataCtr
158 --        -tfIncr   : signal to increment the tfCtr
159 --        -tDelayCtr  : counter used to delay the transfer state
160 --                machine.
161 --        -tDelayRst  : reset signal for the tDelayCtr counter.
162 --
163 --        The following signals are used by the two state machines
```

```
164 --        for state control:
165 --        -Receive State Machine  : strCur, strNext
166 --        -Transfer State Machine : sttCur, sttNext
167 --
168 ---------------------------------------------------------------------------
169
170 --  @26.7MHz
171 --  constant baudRate : std_logic_vector(12 downto 0) := "1 0100 0101 1000";
172 --  constant baudRate : std_logic_vector(12 downto 0) :=
    conv_std_logic_vector(1406,13); -- 19200
173 --  constant baudRate : std_logic_vector(12 downto 0) :=
    conv_std_logic_vector(703,13);  -- 38400
174 --  constant baudRate : std_logic_vector(12 downto 0) :=
    conv_std_logic_vector(469,13);  -- 57600
175 --   constant baudRate  : std_logic_vector(12 downto 0) :=
    conv_std_logic_vector(417,13);  --115200
176
177 --  @26.7MHz
178 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(1,9);  -- Used for simulation
179 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(88,9); -- Used for  19 200 baud
180 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(44,9); -- Used for  38 400 baud
181 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(29,9); -- Used for  57 600 baud
182 --  constant baudDivide :   std_logic_vector(8 downto 0)  :=
    conv_std_logic_vector(26,9); -- Used for 115 200 baud
183
184   constant baudRate : std_logic_vector(12 downto 0)   :=
        conv_std_logic_vector(BAUD_RATE_G,13);  --115200
185   constant baudDivide :   std_logic_vector(8 downto 0)  :=
        conv_std_logic_vector(BAUD_DIVIDE_G-1,9); -- Used for 115 200 baud
186
187   signal rdReg    :   std_logic_vector(7 downto 0)  := "00000000";
188   signal rdSReg   :   std_logic_vector(9 downto 0)  := "1111111111";
189   signal tfReg    :   std_logic_vector(7 downto 0);
190   signal tfSReg     :   std_logic_vector(10 downto 0)   := "11111111111";
191   signal clkDiv   :   std_logic_vector(9 downto 0)    := "0000000000";
192   signal ctr      :   std_logic_vector(3 downto 0)    := "0000";
193   signal tfCtr    :   std_logic_vector(3 downto 0)    := "0000";

194   signal dataCtr    :   std_logic_vector(3 downto 0)    := "0000";
195   signal parError   :   std_logic;
196   signal frameError :   std_logic;
197   signal CE     : std_logic;
198   signal ctRst    :   std_logic   := '0';
199   signal load     :   std_logic   := '0';
200   signal shift    :   std_logic   := '0';
```

```
201   signal par      :   std_logic;
202    signal tClkRST   :   std_logic   := '0';
203   signal rShift    :   std_logic   := '0';
204   signal dataRST   : std_logic   := '0';
205   signal dataIncr  :   std_logic   := '0';
206   signal tfIncr    : std_logic := '0';
207   signal tDelayCtr  : std_logic_vector (12 downto 0);
208   signal tDelayRst  : std_logic := '0';
209
210   signal strCur    :  rstate := strIdle;
211   signal strNext    :  rstate;
212   signal sttCur     :  tstate   := sttIdle;
213   signal sttNext    :  tstate;
214
215 ----------------------------------------------------------------------
216 -- Module Implementation
217 ----------------------------------------------------------------------
218 begin
219 ----------------------------------------------------------------------
220 --
221 --Title:  Initial signal definitions
222 --
223 --Description:  The following lines of code define 4 internal and 1
224 --        external signal.  The most significant bit of the rdSReg
225 --        signifies the frame error bit, so frameError is tied to
226 --        that signal.  The parError is high if there is a parity
227 --        error, so it is set equal to the inverse of rdSReg(8)
228 --        XOR-ed with the data bits.  In this manner, it can
229 --        determine if the parity bit found in rdSReg(8) matches
230 --        the data bits.  The parallel information output is equal
231 --        to rdReg, so DBOUT is set equal to rdReg.  Likewise, the
232 --        input parallel information is equal to DBIN, so tfReg is
233 --        set equal to DBIN.  Because the tfSReg is used to shift
234 --        out transmitted data, the TXD port is set equal to the
235 --        first bit of tfsReg.  Finally, the par signal represents
236 --        the parity of the data, so par is set to the inverse of
237 --        the data bits XOR-ed together.  This UART can be changed
238 --        to use EVEN parity if the "not" is omitted from the par
239 --        definition.
240 --
241 ----------------------------------------------------------------------
242   frameError <= not rdSReg(9);
243   parError <= not ( rdSReg(8) xor (((rdSReg(0) xor rdSReg(1)) xor
244     (rdSReg(2) xor rdSReg(3))) xor ((rdSReg(4) xor rdSReg(5)) xor
245     (rdSReg(6) xor rdSReg(7)))) );
246   DBOUT <= rdReg;
247   tfReg <= DBIN;
248   TXD <= tfsReg(0);
249   par <=  not ( ((tfReg(0) xor tfReg(1)) xor (tfReg(2) xor tfReg(3))) xor
```

```
250     ((tfReg(4) xor tfReg(5)) xor (tfReg(6) xor tfReg(7))) );
251 ----------------------------------------------------------------------
252 --
253 --Title: Clock Divide counter
254 --
255 --Description:  This process defines clkDiv as a signal that increments
256 --        with the clock up until it is either reset by ctRst, or
257 --        equals baudDivide.  This signal is used to define a
258 --        counter called ctr that increments at the rate of the
259 --        divided baud rate.
260 --
261 ----------------------------------------------------------------------
262   process (CLK, clkDiv)
263     begin
264       if (CLK = '1' and CLK'event) then
265         if (clkDiv = baudDivide or ctRst = '1') then
266           clkDiv <= "0000000000";
267         else
268           clkDiv <= clkDiv +1;
269         end if;
270       end if;
271     end process;
272 ----------------------------------------------------------------------
273 --
274 --Title: Transfer delay counter
275 --
276 --Description:  This process defines tDelayCtr as a counter that runs
277 --        until it equals baudRate, or until it is reset by
278 --        tDelayRst.  This counter is used to measure delay times
279 --        when sending data out on the TXD signal.  When the
280 --        counter is equal to baudRate, or is reset, it is set
281 --        equal to 0.
282 --
283 ----------------------------------------------------------------------
284   process (CLK, tDelayCtr)
285     begin
286       if (CLK = '1' and CLK'event) then
287         if (tDelayCtr = baudRate or tDelayRst = '1') then
288           tDelayCtr <= "0000000000000";
289         else
290           tDelayCtr <= tDelayCtr+1;
291         end if;
292       end if;
293     end process;
294 ----------------------------------------------------------------------
295 --
296 --Title: ctr set up
297 --
298 --Description:  This process sets up ctr, which uses clkDiv to count
```

```
299 --        increase at a rate needed to properly receive data in
300 --        from RXD.  If ctRst is strobed, the counter is reset.  If
301 --        clkDiv is equal to baudDivide, then ctr is incremented
302 --        once.  This signal is used by the receiving state machine
303 --        to measure delay times between RXD reads.
304 --
305 -------------------------------------------------------------------------
306   process (CLK)
307     begin
308       if CLK = '1' and CLK'Event then
309         if ctRst = '1' then
310           ctr <= "0000";
311         elsif clkDiv = baudDivide then
312           ctr <= ctr + 1;
313         else
314           ctr <= ctr;
315         end if;
316       end if;
317     end process;
318 -------------------------------------------------------------------------
319 --
320 --Title: transfer counter
321 --
322 --Description:  This process makes tfCtr increment whenever the tfIncr
323 --        signal is strobed high.  If the tClkRst signal is strobed
324 --        high, the tfCtr is reset to "0000."  This counter is used
325 --        to keep track of how many data bits have been
326 --        transmitted.
327 --
328 -------------------------------------------------------------------------
329   process (CLK, tClkRST)
330     begin
331       if (CLK = '1' and CLK'event) then
332         if tClkRST = '1' then
333           tfCtr <= "0000";
334         elsif tfIncr = '1' then
335           tfCtr <= tfCtr +1;
336         end if;
337       end if;
338     end process;
339 -------------------------------------------------------------------------
340 --
341 --Title: Error and RDA flag controller
342 --
343 --Description:  This process controls the error flags FE, OE, and PE, as
344 --        well as the Read Data Available (RDA) flag.  When CE goes
345 --        high, it means that data has been read into the rdSReg.
346 --        This process then analyzes the read data for errors, sets
347 --        rdReg equal to the eight data bits in rdSReg, and flags
```

```
348 --        RDA to indicate that new data is present in rdReg.  FE
349 --        and PE are simply equal to the frameError and parError
350 --        signals.  OE is flagged high if RDA is already high when
351 --        CE is strobed.  This means that unread data was still in
352 --        the rdReg when it was written over with the new data.
353 --
354 -------------------------------------------------------------------------
355   process (CLK, RST, RD, CE)
356     begin
357       if RD = '1' or RST = '1' then
358         FE <= '0';
359         OE <= '0';
360         RDA <= '0';
361         PE <= '0';
362       elsif CLK = '1' and CLK'event then
363         if CE = '1' then
364           FE <= frameError;
365           PE <= parError;
366           rdReg(7 downto 0) <= rdSReg (7 downto 0);
367           if RDA = '1' then
368             OE <= '1';
369           else
370             OE <= '0';
371             RDA <= '1';
372           end if;
373         end if;
374       end if;
375     end process;
376 -------------------------------------------------------------------------
377 --
378 --Title: Receiving shift register
379 --
380 --Description:  This process controls the receiving shift register
381 --        (rdSReg).  Whenever rShift is high, implying that data
382 --        needs to be shifted in, rdSReg is shifts in RXD to the
383 --        most significant bit, while shifting its existing data
384 --        right.
385 --
386 -------------------------------------------------------------------------
387   process (CLK, rShift)
388     begin
389       if CLK = '1' and CLK'Event then
390         if rShift = '1' then
391           rdSReg <= (RXD & rdSReg(9 downto 1));
392         end if;
393       end if;
394     end process;
395 -------------------------------------------------------------------------
396 --
```

```
397 --Title: Incoming Data counter
398 --
399 --Description:  This process controls the dataCtr to keep track of
400 --        shifted values into the rdSReg.  The dataCtr signal is
401 --        incremented once every time dataIncr is strobed high.
402 --
403 -------------------------------------------------------------------------

404
405   process (CLK, dataRST)
406     begin
407       if (CLK = '1' and CLK'event) then
408         if dataRST = '1' then
409           dataCtr <= "0000";
410         elsif dataIncr = '1' then
411           dataCtr <= dataCtr +1;
412         end if;
413       end if;
414     end process;
415 -------------------------------------------------------------------------
416 --
417 --Title: Receiving State Machine controller
418 --
419 --Description:  This process takes care of the Receiving state machine
420 --        movement.  It causes the next state to be evaluated on
421 --        each rising edge of CLK.  If the RST signal is strobed,
422 --        the state is changed to the default starting state,
423 --        which is strIdle
424 --
425 -------------------------------------------------------------------------
426   process (CLK, RST)
427     begin
428       if CLK = '1' and CLK'Event then
429         if RST = '1' then -- najj
430           strCur <= strIdle;
431         else
432           strCur <= strNext;
433         end if;
434       end if;
435     end process;
436 -------------------------------------------------------------------------
437 --
438 --Title: Receiving State Machine
439 --
440 --Description:  This process contains all of the next state logic for the
441 --        Receiving state machine.
442 --
443 -------------------------------------------------------------------------

444   process (strCur, ctr, RXD, dataCtr)
```

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

484         when strEightDelay =>
485           dataIncr <= '0';
486           rShift <= '0';
487           dataRst <= '1';
488           CE <= '0';
489           ctRst <= '0';
490 
491           if ctr(3 downto 0) = "1000" then
492             strNext <= strWaitFor0;
```

```
493           else
494             strNext <= strEightDelay;
495           end if;
496 ---------------------------------------------------------------------------
497 --
498 --Title: strGetData state
499 --
500 --Description:  In this state, the dataIncr and rShift signals are
501 --        strobed high for one clock cycle.  By doing this, the
502 --        rdSReg shift register shifts in RXD once, while the
503 --        dataCtr is incremented by one.  This state simply
504 --        captures the incoming data on RXD into the rdSReg shift
505 --        register.  The next state loaded is strWaitFor0, which
506 --        starts the two delay states needed between data shifts.
507 --
508 ---------------------------------------------------------------------------
509         when strGetData =>
510           CE <= '0';
511           dataRst <= '0';
512           ctRst <= '0';
513           dataIncr <= '1';
514           rShift <= '1';
515 
516           strNext <= strWaitFor0;
517 ---------------------------------------------------------------------------
518 --
519 --Title: strWaitFor0 state
520 --
521 --Description:  This state is a delay state, which delays the receive
522 --        state machine if not all of the incoming serial data has
523 --        not been shifted in yet.  If dataCtr does not equal 10
524 --        (1010), the state is stayed in until the fourth bit of
525 --        ctr is equal to 1.  When this happens, half of the delay
526 --        has been achieved, and the second delay state is loaded,
527 --        which is strWaitFor1.  If dataCtr does equal 10 (1010),
528 --        all of the needed data has been acquired, so the
529 --        strCheckStop state is loaded to check for errors and
530 --        reset the receive state machine.
531 --
532 ---------------------------------------------------------------------------
533         when strWaitFor0 =>
534           CE <= '0';
535           dataRst <= '0';
536           ctRst <= '0';
537           dataIncr <= '0';
538           rShift <= '0';
539 
540           if dataCtr = "1010" then
541             strNext <= strCheckStop;
```

```
542           elsif ctr(3) = '0' then
543             strNext <= strWaitFor1;
544           else
545             strNext <= strWaitFor0;
546           end if;
547 ---------------------------------------------------------------------
548 --
549 --Title: strEightDelay state
550 --
551 --Description:  This state is much like strWaitFor0, except it waits for
552 --        the fourth bit of ctr to equal 1.  Once this occurs, the
553 --        strGetData state is loaded in order to shift in the next
554 --        data bit from RXD.  Because strWaitFor0 is the only state
555 --        that calls this state, no other signals need to be
556 --        checked.
557 --
558 ---------------------------------------------------------------------

559         when strWaitFor1 =>
560           CE <= '0';
561           dataRst <= '0';
562           ctRst <= '0';
563           dataIncr <= '0';
564           rShift <= '0';
565 
566           if ctr(3) = '0' then
567             strNext <= strWaitFor1;
568           else
569             strNext <= strGetData;
570           end if;
571 ---------------------------------------------------------------------
572 --
573 --Title: strCheckStop state
574 --
575 --Description:  This state allows the newly acquired data to be checked
576 --        for errors.  The CE flag is strobed to start the
577 --        previously defined error checking process.  This state is
578 --        passed straight through to the strIdle state.
579 --
580 ---------------------------------------------------------------------

581         when strCheckStop =>
582           dataIncr <= '0';
583           rShift <= '0';
584           dataRst <= '0';
585           ctRst <= '0';
586           CE <= '1';
587           strNext <= strIdle;
588       end case;
```

```
589     end process;
590 -------------------------------------------------------------------------
591 --
592 --Title: Transfer shift register controller
593 --
594 --Description:  This process uses the load, shift, and clk signals to
595 --        control the transfer shift register (tfSReg).  Once load
596 --        is equal to '1', the tfSReg gets a '1', the parity bit,
597 --        the data bits found in tfReg, and a '0'.  Under this
598 --        format, the shift register can be used to shift out the
599 --        appropriate signal to serially transfer the data.  The
600 --        data is shifted out of the tfSReg whenever shift = '1'.
601 --
602 -------------------------------------------------------------------------
603   process (load, shift, CLK, tfSReg)
604     begin
605       if CLK = '1' and CLK'Event then
606         if load = '1' then
607           tfSReg (10 downto 0) <= ('1' & par & tfReg(7 downto 0) &'0');
608         elsif shift = '1' then
609           tfSReg (10 downto 0) <= ('1' & tfSReg(10 downto 1));
610         end if;
611       end if;
612     end process;
613 -------------------------------------------------------------------------
614 --
615 --Title: Transfer State Machine controller
616 --
617 --Description:  This process takes care of the Transfer state machine
618 --        movement.  It causes the next state to be evaluated on
619 --        each rising edge of CLK.  If the RST signal is strobed,
620 --        the state is changed to the default starting state, which
621 --        is sttIdle.
622 --
623 -------------------------------------------------------------------------
624   process (CLK, RST)
625     begin
626       if (CLK = '1' and CLK'Event) then
627         if RST = '1' then
628           sttCur <= sttIdle;
629         else
630           sttCur <= sttNext;
631         end if;
632       end if;
633     end process;
634 -------------------------------------------------------------------------
635 --
636 --Title: Transfer State Machine
637 --
```

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

```
687 --         state loaded is the sttDelay state.
688 --
689 -------------------------------------------------------------------------
690         when sttTransfer =>
691           TBE <= '0';
692           shift <= '0';
693           load <= '1';
694           tClkRST <= '1';
695           tfIncr <= '0';
696           tDelayRst <= '1';
697 
698           sttNext <= sttDelay;
699 -------------------------------------------------------------------------
700 --
701 --Title: sttShift state
702 --
703 --Description:  This state strobes the shift and tfIncr signals high, and
704 --         checks the tfCtr to see if enough data has been
705 --         transmitted.  By strobing the shift and tfIncr signals
706 --         high, the tfSReg is shifted, and the tfCtr is incremented
707 --         once.  If tfCtr does not equal 9 (1001), then not all of
708 --         the bits have been transmitted, so the next state loaded
709 --         is the sttDelay state.  If tfCtr does equal 9, the final
710 --         state, sttWaitWrite, is loaded.
711 --
712 -------------------------------------------------------------------------
713         when sttShift =>
714           TBE <= '0';
715           shift <= '1';
716           load <= '0';
717           tfIncr <= '1';
718           tClkRST <= '0';
719           tDelayRst <= '0';
720 
721           if tfCtr = "1010" then
722             sttNext <= sttWaitWrite;
723           else
724             sttNext <= sttDelay;
725           end if;
726 -------------------------------------------------------------------------
727 --
728 --Title: sttDelay state
729 --
730 --Description:  This state is responsible for delaying the transfer state
731 --         machine between transmissions.  All signals are held low
732 --         while the tDelayCtr is tested.  Once tDelayCtr is equal
733 --         to baudRate, the sttShift state is loaded.
734 --
735 -------------------------------------------------------------------------
```

```
736         when sttDelay =>
737           TBE <= '0';
738           shift <= '0';
739           load <= '0';
740           tClkRst <= '0';
741           tfIncr <= '0';
742           tDelayRst <= '0';
743 
744           if tDelayCtr = baudRate then
745             sttNext <= sttShift;
746           else
747             sttNext <= sttDelay;
748           end if;
749 ----------------------------------------------------------------------
750 --
751 --Title: sttWaitWrite state
752 --
753 --Description:  This state checks to make sure that the initial WR signal
754 --        that triggered the transfer state machine has been
755 --        brought back low.  Without this state, a write signal
756 --        that is held high for a long time will result in multiple
757 --        transmissions.  Once the WR signal is low, the sttIdle
758 --        state is loaded to reset the transfer state machine.
759 --
760 ----------------------------------------------------------------------

761         when sttWaitWrite =>
762           TBE <= '0';
763           shift <= '0';
764           load <= '0';
765           tClkRst <= '0';
766           tfIncr <= '0';
767           tDelayRst <= '0';
768 
769           if WR = '1' then
770             sttNext <= sttWaitWrite;
771           else
772             sttNext <= sttIdle;
773           end if;
774       end case;
775     end process;
776 end Behavioral;
```

Codice Componente 11.1: Definizione del componente UART$_component$

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 
4 entity control_unit is
```

```
    Port ( in_byte : in  STD_LOGIC_VECTOR (7 downto 0);
           reset : in  STD_LOGIC;
           load : in  STD_LOGIC;
           clock : in  STD_LOGIC;
           out_byte : out  STD_LOGIC_VECTOR (7 downto 0));
end control_unit;

architecture Behavioral of control_unit is

signal reg : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');

begin

out_byte <= reg;

main : process(clock, reset)
begin

if reset = '1' then
  reg <= (others => '0');
elsif (clock'event and  clock = '1') then
  if load = '1' then
    reg(7 downto 0) <= in_byte;
  end if;
end if;

end process;

end Behavioral;
```

Codice Componente 11.2: Definizione del componente control$_unit$

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


entity uart_tappo is
    Port ( in_byte : in  STD_LOGIC_VECTOR (7 downto 0);
           clock : in  STD_LOGIC;
           reset : in  STD_LOGIC;
       load : in STD_LOGIC;
           out_byte : out  STD_LOGIC_VECTOR (7 downto 0));
end uart_tappo;

architecture Structural of uart_tappo is

component control_unit
    Port ( in_byte : in  STD_LOGIC_VECTOR (7 downto 0);
           reset : in  STD_LOGIC;
```

```
18            load : in  STD_LOGIC;
19            clock : in  STD_LOGIC;
20            out_byte : out  STD_LOGIC_VECTOR (7 downto 0));
21  end component;
22
23  component UARTcomponent
24    Generic (
25      --@48MHz
26  --    BAUD_DIVIDE_G : integer := 26;  --115200 baud
27  --    BAUD_RATE_G   : integer := 417
28
29      --@26.6MHz
30      BAUD_DIVIDE_G : integer := 14;  --115200 baud
31      BAUD_RATE_G   : integer := 231
32    );
33    Port (
34      TXD   : out   std_logic   := '1';          -- Transmitted serial data
          output
35      RXD   : in    std_logic;                -- Received serial data input
36      CLK   : in    std_logic;                -- Clock signal
37      DBIN  : in    std_logic_vector (7 downto 0);    -- Input parallel data
          to be transmitted
38      DBOUT   : out   std_logic_vector (7 downto 0);    -- Recevived parallel
          data output
39      RDA   : inout  std_logic;               -- Read Data Available, dato
          disponibile in lettura
40      TBE   : out   std_logic   := '1';          -- Transfer Buffer Empty
41      RD    : in    std_logic;                -- Read Strobe, abilito la lettura
42      WR    : in    std_logic;                -- Write Strobe, abilito la
          scrittura
43      PE    : out   std_logic;                -- Parity error
44      FE    : out   std_logic;                -- Frame error
45      OE    : out   std_logic;                -- Overwrite error
46      RST   : in    std_logic := '0');          -- Reset signal
47  end component;
48
49  signal data : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
50  signal temp : STD_LOGIC := '1';
51
52  begin
53
54  cu : control_unit
55  Port Map ( in_byte => in_byte,
56             reset => reset,
57             load => load,
58             clock => clock,
59             out_byte => data
60        );
61
```

```
62 uart : UARTcomponent
63   Generic Map (
64     --@48MHz
65 --    BAUD_DIVIDE_G : integer := 26;  --115200 baud
66 --    BAUD_RATE_G   : integer := 417
67
68     --@26.6MHz
69     BAUD_DIVIDE_G => 14,  --115200 baud
70     BAUD_RATE_G => 231
71   )
72 Port Map ( TXD => temp,
73            RXD => temp,
74            CLK => clock,
75            DBIN => data,
76            DBOUT => out_byte,
77         RST => reset,
78         WR => load,
79         RD => '0'
80       );
81
82 end Structural;
```

Codice Componente 11.3: Definizione del componente uart_*t*appo

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity uart_2 is
5      Port ( in_byte : in  STD_LOGIC_VECTOR (7 downto 0);
6             clock : in  STD_LOGIC;
7             reset : in  STD_LOGIC;
8         load : in STD_LOGIC;
9             out_byte : out  STD_LOGIC_VECTOR (7 downto 0));
10 end uart_2;
11
12 architecture Structural of uart_2 is
13
14 component control_unit
15     Port ( in_byte : in  STD_LOGIC_VECTOR (7 downto 0);
16            reset : in  STD_LOGIC;
17            load : in  STD_LOGIC;
18            clock : in  STD_LOGIC;
19            out_byte : out  STD_LOGIC_VECTOR (7 downto 0));
20 end component;
21
22 component UARTcomponent
23   Generic (
24     --@48MHz
25 --    BAUD_DIVIDE_G : integer := 26;  --115200 baud
```

```
--    BAUD_RATE_G   : integer := 417

    --@26.6MHz
    BAUD_DIVIDE_G : integer := 14;  --115200 baud
    BAUD_RATE_G   : integer := 231
  );
  Port (
    TXD   : out   std_logic   := '1';         -- Transmitted serial data output
    RXD   : in    std_logic;                  -- Received serial data input
    CLK   : in    std_logic;                  -- Clock signal
    DBIN  : in    std_logic_vector (7 downto 0);    -- Input parallel data to be transmitted
    DBOUT   : out   std_logic_vector (7 downto 0);    -- Recevived parallel data output
    RDA   : inout  std_logic;                 -- Read Data Available, dato disponibile in lettura
    TBE   : out   std_logic   := '1';         -- Transfer Buffer Empty
    RD    : in    std_logic;                  -- Read Strobe, abilito la lettura
    WR    : in    std_logic;                  -- Write Strobe, abilito la scrittura
    PE    : out   std_logic;                  -- Parity error
    FE    : out   std_logic;                  -- Frame error
    OE    : out   std_logic;                  -- Overwrite error
    RST   : in    std_logic := '0');          -- Reset signal
end component;

signal data : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal temp : STD_LOGIC := '1';

begin

cu : control_unit
Port Map ( in_byte => in_byte,
           reset => reset,
           load => load,
           clock => clock,
           out_byte => data
      );

uart_tx : UARTcomponent
  Generic Map (
    --@48MHz
--    BAUD_DIVIDE_G : integer := 26;  --115200 baud
--    BAUD_RATE_G   : integer := 417

    --@26.6MHz
    BAUD_DIVIDE_G => 14,  --115200 baud
    BAUD_RATE_G => 231
```

```
70    )
71  Port Map ( TXD => temp,
72           RXD => '1',
73              CLK => clock,
74              DBIN => data,
75           RST => reset,
76           WR => load,
77           RD => '1'
78         );
79
80
81  uart_rx : UARTcomponent
82    Generic Map (
83      --@48MHz
84  --    BAUD_DIVIDE_G : integer := 26;  --115200 baud
85  --    BAUD_RATE_G   : integer := 417
86
87      --@26.6MHz
88      BAUD_DIVIDE_G => 14,  --115200 baud
89      BAUD_RATE_G => 231
90    )
91  Port Map ( RXD => temp,
92              CLK => clock,
93           DBIN => (others => '0'),
94              DBOUT => out_byte,
95           RST => reset,
96           WR => '0',
97           RD => '0'
98         );
99
100 end Structural;
```

Codice Componente 11.4: Definizione del componente $2_uart$

## 11.3 Simulazione

## 11.4 Sintesi su board FPGA

Per interfacciare il design realizzato con l'I/O di board è stata utilizzata la seguente soluzione:

Per il clock si è utilizzato il clock della scheda.

```
## Clock signal
NET "clock"   LOC = "E3"  | IOSTANDARD = "LVCMOS33";
```

Per il dato da trasmettere sono stati utilizzati gli 8 switch più a destra della board.

```
## Switches
NET "in_byte<0>"          LOC=J15 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_RS0_15
NET "in_byte<1>"          LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "in_byte<2>"          LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
NET "in_byte<3>"          LOC=R15 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_14
NET "in_byte<4>"          LOC=R17 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_14
NET "in_byte<5>"          LOC=T18 | IOSTANDARD=LVCMOS33; #IO_L7N_T1_D10_14
NET "in_byte<6>"          LOC=U18 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A13_D29_14
NET "in_byte<7>"          LOC=R13 | IOSTANDARD=LVCMOS33; #IO_L5N_T0_D07_14
```

Per il reset della trasmissione e per l'avvio della trasmissione sono stati utilizzati i seguenti bottoni.

```
## Buttons
NET "reset"           LOC=M17 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_D15_14
NET "load"            LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14
```

Per mostrare l'output, quindi il dato parallelo ricevuto, sono stati utilizzati gli 8 led più a destra della scheda.

```
## LEDs
NET "out_byte<0>"         LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
NET "out_byte<1>"         LOC=K15 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_RS1_15
NET "out_byte<2>"         LOC=J13 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A25_15
NET "out_byte<3>"         LOC=N14 | IOSTANDARD=LVCMOS33; #IO_L8P_T1_D11_14
NET "out_byte<4>"         LOC=R18 | IOSTANDARD=LVCMOS33; #IO_L7P_T1_D09_14
NET "out_byte<5>"         LOC=V17 | IOSTANDARD=LVCMOS33; #IO_L18N_T2_A11_D27_14
NET "out_byte<6>"         LOC=U17 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A14_D30_14
NET "out_byte<7>"         LOC=U16 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A12_D28_14
```

# 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

### 12.2.1 Toolchain

Dopo aver installato il microassemblatore mal e l'assemblatore ajvm e i diversi control system necessari per permettere il corretto funzionamento dei tools, per lanciare microassemblatore e assemblatore per aggiornare il contenuto del control_store e della ram sono necessarie le seguenti istruzioni da bash UBUNTU:

- Istruzioni per modificare il control_store:

```
poba@DESKTOP-1526FLJ:~/esercitazione_mic$ cd amic-0
poba@DESKTOP-1526FLJ:~/esercitazione_mic/amic-0$ cd build
poba@DESKTOP-1526FLJ:~/esercitazione_mic/amic-0/build$ cmake ..
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/common_defs.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/alu.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/control_unit.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/processor.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/datapath.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/control_store.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/dp_ar_ram.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/system.vhd
-- Adding VHDL Test: /home/poba/esercitazione_mic/amic-0/src/test/vhdl/alu_tb.vhd
-- Adding VHDL Test: /home/poba/esercitazione_mic/amic-0/src/test/vhdl/control_unit_tb.vhd
-- Adding VHDL Test: /home/poba/esercitazione_mic/amic-0/src/test/vhdl/datapath_tb.vhd
-- Adding VHDL Test: /home/poba/esercitazione_mic/amic-0/src/test/vhdl/processor_tb.vhd
-- Configuring done
-- Generating done
-- Build files have been written to: /home/poba/esercitazione_mic/amic-0/build
poba@DESKTOP-1526FLJ:~/esercitazione_mic/amic-0/build$ make create_control_store
```

- Istruzioni per modificare la ram:

```
poba@DESKTOP-1526FLJ:~/esercitazione_mic/amic-0/build$ cmake ..
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/common_defs.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/alu.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/control_unit.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/processor.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/datapath.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/control_store.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/dp_ar_ram.vhd
-- Adding VHDL Source: /home/poba/esercitazione_mic/amic-0/src/main/vhdl/system.vhd
-- Adding VHDL Test: /home/poba/esercitazione_mic/amic-0/src/test/vhdl/alu_tb.vhd
-- Adding VHDL Test: /home/poba/esercitazione_mic/amic-0/src/test/vhdl/control_unit_tb.vhd
-- Adding VHDL Test: /home/poba/esercitazione_mic/amic-0/src/test/vhdl/datapath_tb.vhd
-- Adding VHDL Test: /home/poba/esercitazione_mic/amic-0/src/test/vhdl/processor_tb.vhd
-- Configuring done
-- Generating done
-- Build files have been written to: /home/poba/esercitazione_mic/amic-0/build
poba@DESKTOP-1526FLJ:~/esercitazione_mic/amic-0/build$ make create_ram
Built target assemble_program
Built target create_ram
poba@DESKTOP-1526FLJ:~/esercitazione_mic/amic-0/build$
```

## 12.2.2 Codice

### 12.2.2.1 Processore Mic

Sono riportati i codici dei componenti forniti nel materiale didattico per la realizzazione del mic con qualche commento per chiarirne il funzionamento, e i codici dei componenti utilizzati per tentare la sintesi su FPGA

```
1 ----------------------------------------------------------------------------
    VHDL--
2 -- Copyright (C) 2019 Alberto Moriconi
3 --
4 -- This program is free software: you can redistribute it and/or modify it
    under
5 -- the terms of the GNU General Public License as published by the Free
    Software
```

```
6  -- Foundation, either version 3 of the License, or (at your option) any
       later
7  -- version.
8  --
9  -- This program is distributed in the hope that it will be useful, but
       WITHOUT
10 -- ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
       FITNESS
11 -- FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
12 -- details.
13 --
14 -- You should have received a copy of the GNU General Public License along
       with
15 -- this program. If not, see <http://www.gnu.org/licenses/>.
16 --
       ----------------------------------------------------------------------------

17 --! @file alu.vhd
18 --! @author Alberto Moriconi
19 --! @date 2019-04-26
20 --! @brief Arithmetic logic unit based on MIC-1 ALU
21 --
       ----------------------------------------------------------------------------

22
23 library ieee;
24 use ieee.std_logic_1164.all;
25 use ieee.numeric_std.all;
26
27 use work.common_defs.all;
28
29 --! Arithmetic logic unit based on MIC-1 ALU
30
31 --! The ALU operates on 2 operands (A, B); its operation is controlled by
       the 8
32 --! control bits:
33 --!
34 --! SLL8 | SRA1 | F_0 | F_1 | EN_A | EN_B | INV_A | INC
35 --!   ^                                           ^
36 --!  MSB                                         LSB
37 --!
38 --! The arithmetical/logical operation is selected by the 2 MSBs (F_0, F_1):
39 --!
40 --!  - 00 : Logical AND (A AND B)
41 --!  - 01 : Logical OR (A OR B)
42 --!  - 10 : Logical negation (NOT B)
43 --!  - 11 : Arithmetical sum (A + B)
44 --!
45 --! The ALU also provides separate enables for A and B and an invert signal
```

```
    for
46 --! A. The INC bit is used only by the arithmetical sum and has the effect
    of
47 --! adding an LSB to the sum.
48 --!
49 --! The zero flag (Z) is high when all bits of the result are 0; the
    negative
50 --! flag (N) is high when the operation is the arithmetical sum and the MSB
    of
51 --! the result is 1.
52 --!
53 --! When SLL8 is high, the result of the operation is shifted left 8 bits
    and
54 --! the 8 LSBs are zero filled; when SRA1 is high, the input is shifter
    right
55 --! 1 bit and the MSB is sign extended; when both bits are low, the input is
56 --! taken to the output unmodified; when both are high, the output is
    undefined.
57 --!
58 --! The shifted result goes to the output.
59 entity alu is
60   port (
61     --! ALU control
62     control       : in  alu_ctrl_type;
63     --! ALU operand A
64     operand_a     : in  reg_data_type;
65     --! ALU operand B
66     operand_b     : in  reg_data_type;
67     --! ALU result
68     sh_result     : out reg_data_type;
69     --! Negative flag
70     negative_flag : out std_logic;
71     --! Zero flag
72     zero_flag     : out std_logic
73     );
74 end entity alu;
75
76 --! Behavioral architecture for the ALU
77 architecture behavioral of alu is
78
79   -- Alias
80   alias sh    : alu_sh_type is control(alu_sh_type'range);
81   alias fn    : alu_fn_type is control(alu_fn_type'range);
82   alias en_a  : std_logic is control(alu_ctrl_en_a);
83   alias en_b  : std_logic is control(alu_ctrl_en_b);
84   alias inv_a : std_logic is control(alu_ctrl_inv_a);
85   alias inc   : std_logic is control(alu_ctrl_inc);
86
87   -- Signals
```

```
88    signal t_operand_a     : reg_data_type;
89    signal t_operand_a_inv : reg_data_type;
90    signal t_operand_b     : reg_data_type;
91    signal t_inc           : std_logic_vector(0 downto 0);
92    signal t_u_sum         : unsigned(reg_data_type'range);
93    signal t_and           : reg_data_type;
94    signal t_or            : reg_data_type;
95    signal t_not_b         : reg_data_type;
96    signal t_sum           : reg_data_type;
97    signal t_result        : reg_data_type;
98
99  begin  -- architecture behavioral
100
101   -- Inputs
102   t_operand_a     <= operand_a       when en_a = '1'  else (others => '0');
103   t_operand_a_inv <= not t_operand_a when inv_a = '1' else t_operand_a;
104   t_operand_b     <= operand_b       when en_b = '1'  else (others => '0');
105   t_inc(0)        <= inc;
106   t_u_sum         <= unsigned(t_operand_a_inv) + unsigned(t_operand_b) +
          unsigned(t_inc);
107
108   -- ALU function
109   t_and   <= t_operand_a_inv and t_operand_b when fn = alu_fn_and   else (
          others => '0');
110   t_or    <= t_operand_a_inv or t_operand_b  when fn = alu_fn_or    else (
          others => '0');
111   t_not_b <= not t_operand_b                 when fn = alu_fn_not_b else (
          others => '0');
112   t_sum   <= reg_data_type(t_u_sum)          when fn = alu_fn_sum   else (
          others => '0');
113
114   with fn select t_result <=
115     t_and   when alu_fn_and,
116     t_or    when alu_fn_or,
117     t_not_b when alu_fn_not_b,
118     t_sum   when others;
119
120   -- ALU flags
121   negative_flag <= t_sum(31);
122   zero_flag     <= '1' when t_result = x"00000000" else '0';
123
124   -- Shifter
125   with sh select sh_result <=
126     t_result(23 downto 0) & x"00"       when alu_sh_sll8,
127     t_result(31) & t_result(31 downto 1) when alu_sh_sra1,
128     t_result                            when others;
129
130 end architecture behavioral;
```

Codice Componente 12.1: Definizione del componente alu

```
1  ----------------------------------------------------------------------------
   VHDL--
2  -- Copyright (C) 2019 Alberto Moriconi
3  --
4  -- This program is free software: you can redistribute it and/or modify it
   under
5  -- the terms of the GNU General Public License as published by the Free
   Software
6  -- Foundation, either version 3 of the License, or (at your option) any
   later
7  -- version.
8  --
9  -- This program is distributed in the hope that it will be useful, but
   WITHOUT
10 -- ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
   FITNESS
11 -- FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
12 -- details.
13 --
14 -- You should have received a copy of the GNU General Public License along
   with
15 -- this program. If not, see <http://www.gnu.org/licenses/>.
16 --
   ----------------------------------------------------------------------------

17 --! @file common_defs.vhd
18 --! @author Alberto Moriconi
19 --! @date 2019-05-13
20 --! @brief Common definitions for amic-0 design
21 --
   ----------------------------------------------------------------------------

22
23 library ieee;
24 use ieee.std_logic_1164.all;
25
26 --! Common definitions for amic-0 design
27
28 --! Contains common definitions for the processor design.
29 package common_defs is
30
31   -- Data widths
32   --! Processor register data width
33   constant reg_data_width      : positive := 32;
34   --! MBR register data width
35   constant mbr_data_width      : positive := 8;
```

```
36  --! ALU control width
37  constant alu_ctrl_width      : positive := 8;
38  --! C bus control width
39  constant c_ctrl_width        : positive := 9;
40  --! Memory control width
41  constant mem_ctrl_width      : positive := 3;
42  --! B bus control width
43  constant b_ctrl_width        : positive := 9;
44  --! Control store address width
45  constant ctrl_str_addr_width : positive := 9;
46  --! Control store word width
47  constant ctrl_str_word_width : positive := 36;
48  --! Control store size
49  constant ctrl_str_words      : positive := 512;
50  --! RAM size in 32 bit words (16 KB)
51  constant dp_ar_ram_size      : positive := 512;
52
53  -- Subtypes
54  --! Processor register data
55  subtype reg_data_type is std_logic_vector(reg_data_width - 1 downto 0);
56  --! MBR register data
57  subtype mbr_data_type is std_logic_vector(mbr_data_width - 1 downto 0);
58  --! ALU control type
59  subtype alu_ctrl_type is std_logic_vector(alu_ctrl_width - 1 downto 0);
60  --! C bus control type
61  subtype c_ctrl_type is std_logic_vector(c_ctrl_width -1 downto 0);
62  --! Memory control type
63  subtype mem_ctrl_type is std_logic_vector(mem_ctrl_width - 1 downto 0);
64  --! B bus control type
65  subtype b_ctrl_type is std_logic_vector(b_ctrl_width - 1 downto 0);
66  --! Shifter control input
67  subtype alu_sh_type is std_logic_vector(7 downto 6);
68  --! ALU function field
69  subtype alu_fn_type is std_logic_vector(5 downto 4);
70  --! Control store address
71  subtype ctrl_str_addr_type is std_logic_vector(ctrl_str_addr_width - 1
        downto 0);
72  --! Control store word
73  subtype ctrl_str_word_type is std_logic_vector(ctrl_str_word_width - 1
        downto 0);
74  --! Control store word next address field
75  subtype ctrl_nxt_addr_type is std_logic_vector(35 downto 27);
76  --! Control store word next address field without MSB
77  subtype ctrl_nxt_addr_no_msb_type is std_logic_vector(34 downto 27);
78  --! Control store word ALU field
79  subtype ctrl_alu is std_logic_vector(23 downto 16);
80  --! Control store word C bus field
81  subtype ctrl_c is std_logic_vector(15 downto 7);
82  --! Control store word memory field
```

```
83  subtype ctrl_mem is std_logic_vector(6 downto 4);
84  --! Control store word B bus field
85  subtype ctrl_b is std_logic_vector(3 downto 0);
86  --! MBR signed extension range
87  subtype mbr_s_ext is std_logic_vector(reg_data_width - 1 downto
        mbr_data_width);
88
89  -- Types
90  --! Control store content
91  type ctrl_str_type is array (ctrl_str_words - 1 downto 0) of
        ctrl_str_word_type;
92  --! RAM content
93  type dp_ar_ram_type is array (dp_ar_ram_size - 1 downto 0) of
        reg_data_type;
94
95  -- Fields
96  --! ALU control EN_A bit
97  constant alu_ctrl_en_a     : natural := 3;
98  --! ALU control EN_B bit
99  constant alu_ctrl_en_b     : natural := 2;
100 --! ALU control INV_A bit
101 constant alu_ctrl_inv_a    : natural := 1;
102 --! ALU control INV bit
103 constant alu_ctrl_inc      : natural := 0;
104 --! Control store word next address MSB
105 constant ctrl_nxt_addr_msb : natural := 35;
106 --! Control store word JMPC bit
107 constant ctrl_jmpc         : natural := 26;
108 --! Control store word JAMN bit
109 constant ctrl_jamn         : natural := 25;
110 --! Control store word JAMZ bit
111 constant ctrl_jamz         : natural := 24;
112 --! C control MAR bit
113 constant c_ctrl_mar        : natural := 0;
114 --! C control MDR bit
115 constant c_ctrl_mdr        : natural := 1;
116 --! C control PC bit
117 constant c_ctrl_pc         : natural := 2;
118 --! C control SP bit
119 constant c_ctrl_sp         : natural := 3;
120 --! C control LV bit
121 constant c_ctrl_lv         : natural := 4;
122 --! C control CPP bit
123 constant c_ctrl_cpp        : natural := 5;
124 --! C control TOS bit
125 constant c_ctrl_tos        : natural := 6;
126 --! C control OPX bit
127 constant c_ctrl_opc        : natural := 7;
128 --! C control H bit
```

```
129   constant c_ctrl_h          : natural := 8;
130   --! Memory control fetch bit
131   constant mem_ctrl_fetch    : natural := 0;
132   --! Memory control read bit
133   constant mem_ctrl_read     : natural := 1;
134   --! Memory control write bit
135   constant mem_ctrl_write    : natural := 2;
136
137   --! Constants
138   --! ALU function logical AND
139   constant alu_fn_and   : alu_fn_type := "00";
140   --! ALU function logical OR
141   constant alu_fn_or    : alu_fn_type := "01";
142   --! ALU function logical NOT B
143   constant alu_fn_not_b : alu_fn_type := "10";
144   --! ALU function arithmetical sum
145   constant alu_fn_sum   : alu_fn_type := "11";
146   --! Shifter control shift left logical 8 bit
147   constant alu_sh_sll8  : alu_sh_type := "10";
148   --! Shifter control shift right arithmetical 1 bit
149   constant alu_sh_sra1  : alu_sh_type := "01";
150   --! Shifter control NOP
151   constant alu_sh_nop   : alu_sh_type := "00";
152   --! Shifter control invalid
153   constant alu_sh_err   : alu_sh_type := "11";
154   --! B control MDR mask
155   constant b_ctrl_mdr   : b_ctrl_type := "000000001";
156   --! B control PC mask
157   constant b_ctrl_pc    : b_ctrl_type := "000000010";
158   --! B control MBR mask
159   constant b_ctrl_mbr   : b_ctrl_type := "000000100";
160   --! B control MBRU mask
161   constant b_ctrl_mbru  : b_ctrl_type := "000001000";
162   --! B control SP mask
163   constant b_ctrl_sp    : b_ctrl_type := "000010000";
164   --! B control LV mask
165   constant b_ctrl_lv    : b_ctrl_type := "000100000";
166   --! B control CPP mask
167   constant b_ctrl_cpp   : b_ctrl_type := "001000000";
168   --! B control TOS mask
169   constant b_ctrl_tos   : b_ctrl_type := "010000000";
170   --! B control OPC mask
171   constant b_ctrl_opc   : b_ctrl_type := "100000000";
172
173 end package common_defs;
```

Codice Componente 12.2: Definizione del componente $common_defs$

```
1  ----------------------------------------------------------------------------
   VHDL--
2  -- Copyright (C) 2019 Alberto Moriconi
3  --
4  -- This program is free software: you can redistribute it and/or modify it
   under
5  -- the terms of the GNU General Public License as published by the Free
   Software
6  -- Foundation, either version 3 of the License, or (at your option) any
   later
7  -- version.
8  --
9  -- This program is distributed in the hope that it will be useful, but
   WITHOUT
10 -- ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
   FITNESS
11 -- FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
12 -- details.
13 --
14 -- You should have received a copy of the GNU General Public License along
   with
15 -- this program. If not, see <http://www.gnu.org/licenses/>.
16 --
   ----------------------------------------------------------------------------

17 --! @file control_store.vhd
18 --! @author Alberto Moriconi
19 --! @date 2019-05-11
20 --! @brief Processor control store
21 --
   ----------------------------------------------------------------------------

22
23 library ieee;
24 use ieee.std_logic_1164.all;
25 use ieee.numeric_std.all;
26
27 use work.common_defs.all;
28
29 --! Processor control store
30
31 --! The control store is a ROM used to store the processor microprogram.
32 entity control_store is
33   port (
34     --! Address of the desired word
35     address : in  ctrl_str_addr_type;
36     --! Content of the addressed word
37     word    : out ctrl_str_word_type
38     );
```

```
39 end entity control_store;
40
41 --! Dataflow architecture for the control store
42 architecture dataflow of control_store is
43
44   -- Constants
45   constant words : ctrl_str_type := (
46 -- indirizzi che vengono modificati dall'istruzione make
     create_control_store
47 --BEGIN_WORDS_ENTRY
48 0 => "00000011000000000000000000000001001",
49 1 => "01011110000000000000000000000001001",
50 2 => "00000000000000000000000000000000000",
51 3 => "00000000000000000000000000000000000",
52 4 => "00000000000000000000000000000000000",
53 5 => "00000000000000000000000000000000000",
54 6 => "00000000010000110101000000100001001",
55 7 => "00000100000000110101000000100000001",
56 8 => "00000100100000110101000000100001001",
57 9 => "00000011000000000000000000000001001",
58 10 => "00000000000000000000000000000000000",
59 11 => "00000000000000000000000000000000000",
60 12 => "00000000000000000000000000000000000",
61 13 => "00000000000000000000000000000000000",
62 14 => "00000000000000000000000000000000000",
63 15 => "00000000000000000000000000000000000",
64 16 => "00001000100000110101000001001000100",
65 17 => "00001001000000011010100000010000100",
66 18 => "00000011000000010100001000010100010",
67 19 => "00000000000000000000000000000000000",
68 20 => "00000000000000000000000000000000000",
69 21 => "00001011000000010100100000000000101",
70 22 => "00001011100000011110000000000010100011",
71 23 => "00001100000000110101000001001000100",
72 24 => "00001100100000110101000000100100010001",
73 25 => "00000011000000010100001000000000000",
74 26 => "00000000000000000000000000000000000",
75 27 => "00000000000000000000000000000000000",
76 28 => "00000000000000000000000000000000000",
77 29 => "00000000000000000000000000000000000",
78 30 => "00000000000000000000000000000000000",
79 31 => "00000000000000000000000000000000000",
80 32 => "00010000100000110101000000100001001",
81 33 => "00010001000010010100100000000000011",
82 34 => "00010001100000011110010000000000011",
83 35 => "00001011100000011110000000000010100110",
84 36 => "00000000000000000000000000000000000",
85 37 => "00000000000000000000000000000000000",
86 38 => "00000000000000000000000000000000000",
```

```
87  39 => "00000000000000000000000000000000",
88  40 => "00000000000000000000000000000000",
89  41 => "00000000000000000000000000000000",
90  42 => "00000000000000000000000000000000",
91  43 => "00000000000000000000000000000000",
92  44 => "00000000000000000000000000000000",
93  45 => "00000000000000000000000000000000",
94  46 => "00000000000000000000000000000000",
95  47 => "00000000000000000000000000000000",
96  48 => "00000000000000000000000000000000",
97  49 => "00000000000000000000000000000000",
98  50 => "00000000000000000000000000000000",
99  51 => "00000000000000000000000000000000",
100 52 => "00000000000000000000000000000000",
101 53 => "00000000000000000000000000000000",
102 54 => "00011011100000010100100000000000101",
103 55 => "00011100000000111100000000010000011",
104 56 => "00011100100000010100000000101000111",
105 57 => "00011101000000110110000001001010100",
106 58 => "00011101100000110101000000100001001",
107 59 => "00000011000000010100001000000000000",
108 60 => "00000000000000000000000000000000",
109 61 => "00000000000000000000000000000000",
110 62 => "00000000000000000000000000000000",
111 63 => "00000000000000000000000000000000",
112 64 => "00000000000000000000000000000000",
113 65 => "00000000000000000000000000000000",
114 66 => "00000000000000000000000000000000",
115 67 => "00000000000000000000000000000000",
116 68 => "00000000000000000000000000000000",
117 69 => "00000000000000000000000000000000",
118 70 => "00000000000000000000000000000000",
119 71 => "00000000000000000000000000000000",
120 72 => "00000000000000000000000000000000",
121 73 => "00000000000000000000000000000000",
122 74 => "00000000000000000000000000000000",
123 75 => "00000000000000000000000000000000",
124 76 => "00000000000000000000000000000000",
125 77 => "00000000000000000000000000000000",
126 78 => "00000000000000000000000000000000",
127 79 => "00000000000000000000000000000000",
128 80 => "00000000000000000000000000000000",
129 81 => "00000000000000000000000000000000",
130 82 => "00000000000000000000000000000000",
131 83 => "00000000000000000000000000000000",
132 84 => "00000000000000000000000000000000",
133 85 => "00000000000000000000000000000000",
134 86 => "00000000000000000000000000000000",
135 87 => "00101100000000110101000010010000100",
```

```
88 => "00000011000000010100000000001010001111",
89 => "00101101000000110110000001001010010100",
90 => "00101101100000000000000000000000001001",
91 => "00000011000000010100001000000000000000",
92 => "00101110100000110110000001001010010100",
93 => "00101111000000010100100000000000000111",
94 => "00000011000000011111100100001010000000",
95 => "00110000000000110110000000001010010100",
96 => "00110000100000010100000000001000000100",
97 => "00110001000000010100100000000001000000",
98 => "00110001100000010100000000001000001111",
99 => "00110010000000110110000000000110001100",
100 => "00000011000000011000001000000000001001",
101 => "00110011000000110110000001001010010100",
102 => "00110011100000010100100000000000000111",
103 => "00000011000000011110000100001010000000",
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 => "00000000000000000000000000000000000000",
123 => "00000000000000000000000000000000000000",
124 => "00000000000000000000000000000000000000",
125 => "00000000000000000000000000000000000000",
126 => "00111111100000110110000001001010010100",
127 => "01000000000000010100100000000000000111",
128 => "00000011000000001100001000001010000000",
129 => "00000000000000000000000000000000000000",
130 => "00000000000000000000000000000000000000",
131 => "00000000000000000000000000000000000000",
132 => "01000010100000010100100000000000000101",
133 => "01000011000000011110000000000010100011",
134 => "01000011100000110101000000010000010001",
135 => "01000100000000010100100000000000000000",
136 => "01000100100000110101000000010000010001",
```

```
137 => "00000011000000111100000000101000010",
138 => "00000000000000000000000000000000000",
139 => "00000000000000000000000000000000000",
140 => "00000000000000000000000000000000000",
141 => "00000000000000000000000000000000000",
142 => "00000000000000000000000000000000000",
143 => "00000000000000000000000000000000000",
144 => "00000000000000000000000000000000000",
145 => "00000000000000000000000000000000000",
146 => "00000000000000000000000000000000000",
147 => "00000000000000000000000000000000000",
148 => "00000000000000000000000000000000000",
149 => "00000000000000000000000000000000000",
150 => "00000000000000000000000000000000000",
151 => "00000000000000000000000000000000000",
152 => "00000000000000000000000000000000000",
153 => "01001101000000110110000001001010010",
154 => "01001101100000010100010000000000111",
155 => "01001110000000010100001000000000000",
156 => "00000011100100010100000000000001000",
157 => "01001111000000110110000001001010010",
158 => "01001111100000010100010000000000111",
159 => "01010000000000010100001000000000000",
160 => "00000011101000010100000000000001000",
161 => "01010001000000110110000001001010010",
162 => "01010001100000110110000001001000010",
163 => "01010010000000010100100000000100000",
164 => "01010010100000010100010000000000111",
165 => "01010011000000010100001000000000000",
166 => "00000011100100111111000000000001000",
167 => "01010100000000110110010000000000001",
168 => "01010100100000110101000000100001001",
169 => "01010101000010010100100000000000010",
170 => "01010101100000011100100000000000011",
171 => "01010110000000111100000000100001100",
172 => "00000011000000000000000000000001001",
173 => "01010111000000010100000001001010010",
174 => "01010111100000000000000000000001001",
175 => "01011000000000010100000010001010000",
176 => "01011000100000110101000000010000101",
177 => "01011001000000010100000000100011000",
178 => "01011001100000010100000000001000100",
179 => "01011010000000010100000100000000000",
180 => "01011010100000010100000000001010001",
181 => "00000011000100110110000000000000001",
182 => "01011011100000110110000001001010010",
183 => "01011100000000010100100000000000111",
184 => "00000011000000011100001000010100000",
185 => "01011101000000110101000000100001001",
```

```
234 186 => "01011101100010010100100000000000011",
235 187 => "01011110000000011100100000000000011",
236 188 => "01011110100000111100000000010100110",
237 189 => "01011111000000110101010000000000001",
238 190 => "01011111100000010100000001000010000",
239 191 => "01100000000000110101000001000010001",
240 192 => "01100000100010010100100000000000011",
241 193 => "01100001000000011100100000000000011",
242 194 => "01100001100000110101000001000010001",
243 195 => "01100010000000111111001000000000100",
244 196 => "01100010100000110101001000010000111",
245 197 => "01100011000000110101000001000010001",
246 198 => "01100011100010010100100000000000011",
247 199 => "01100100000000011100100000000000011",
248 200 => "01100100100000111101000000101000100",
249 201 => "01100101000000010100000001001010000000",
250 202 => "01100101100000010100000000001010001000",
251 203 => "01100110000000110101000001001010000100",
252 204 => "01100110100000010100000000001010000101",
253 205 => "01100111000000110101000001000010001",
254 206 => "00000011000000010100000100000000111",
255 207 => "10000000010000110101000001000010001",
256 208 => "00000000000000000000000000000000000000",
257 209 => "00000000000000000000000000000000000000",
258 210 => "00000000000000000000000000000000000000",
259 211 => "00000000000000000000000000000000000000",
260 212 => "00000000000000000000000000000000000000",
261 213 => "00000000000000000000000000000000000000",
262 214 => "00000000000000000000000000000000000000",
263 215 => "00000000000000000000000000000000000000",
264 216 => "00000000000000000000000000000000000000",
265 217 => "00000000000000000000000000000000000000",
266 218 => "00000000000000000000000000000000000000",
267 219 => "00000000000000000000000000000000000000",
268 220 => "00000000000000000000000000000000000000",
269 221 => "00000000000000000000000000000000000000",
270 222 => "00000000000000000000000000000000000000",
271 223 => "00000000000000000000000000000000000000",
272 224 => "00000000000000000000000000000000000000",
273 225 => "00000000000000000000000000000000000000",
274 226 => "00000000000000000000000000000000000000",
275 227 => "00000000000000000000000000000000000000",
276 228 => "00000000000000000000000000000000000000",
277 229 => "00000000000000000000000000000000000000",
278 230 => "00000000000000000000000000000000000000",
279 231 => "00000000000000000000000000000000000000",
280 232 => "00000000000000000000000000000000000000",
281 233 => "00000000000000000000000000000000000000",
282 234 => "00000000000000000000000000000000000000",
```

```
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 => "10000011000000000000000000001001",
263 => "01010100000000110110010000000010001",
264 => "00000000000000000000000000000000",
265 => "00000000000000000000000000000000",
266 => "00000000000000000000000000000000",
267 => "00000000000000000000000000000000",
268 => "00000000000000000000000000000000",
269 => "00000000000000000000000000000000",
270 => "00000000000000000000000000000000",
271 => "00000000000000000000000000000000",
272 => "00000000000000000000000000000000",
273 => "00000000000000000000000000000000",
274 => "00000000000000000000000000000000",
275 => "00000000000000000000000000000000",
276 => "00000000000000000000000000000000",
277 => "10001011000000110101000000100010001",
278 => "10001011100010010100100000000000011",
279 => "10001100000000011100100000000000011",
280 => "00001011100000111100000000010100101",
281 => "00000000000000000000000000000000",
282 => "00000000000000000000000000000000",
283 => "00000000000000000000000000000000",
```

```
284 => "00000000000000000000000000000000",
285 => "00000000000000000000000000000000",
286 => "00000000000000000000000000000000",
287 => "00000000000000000000000000000000",
288 => "00000000000000000000000000000000",
289 => "00000000000000000000000000000000",
290 => "00000000000000000000000000000000",
291 => "00000000000000000000000000000000",
292 => "00000000000000000000000000000000",
293 => "00000000000000000000000000000000",
294 => "00000000000000000000000000000000",
295 => "00000000000000000000000000000000",
296 => "00000000000000000000000000000000",
297 => "00000000000000000000000000000000",
298 => "00000000000000000000000000000000",
299 => "00000000000000000000000000000000",
300 => "00000000000000000000000000000000",
301 => "00000000000000000000000000000000",
302 => "00000000000000000000000000000000",
303 => "00000000000000000000000000000000",
304 => "00000000000000000000000000000000",
305 => "00000000000000000000000000000000",
306 => "00000000000000000000000000000000",
307 => "00000000000000000000000000000000",
308 => "00000000000000000000000000000000",
309 => "00000000000000000000000000000000",
310 => "10011011100000110101000000100010001",
311 => "10011100000010010100100000000000011",
312 => "10011100100000011100100000000000011",
313 => "00011100000000111100000000010000101",
314 => "00000000000000000000000000000000",
315 => "00000000000000000000000000000000",
316 => "00000000000000000000000000000000",
317 => "00000000000000000000000000000000",
318 => "00000000000000000000000000000000",
319 => "00000000000000000000000000000000",
320 => "00000000000000000000000000000000",
321 => "00000000000000000000000000000000",
322 => "00000000000000000000000000000000",
323 => "00000000000000000000000000000000",
324 => "00000000000000000000000000000000",
325 => "00000000000000000000000000000000",
326 => "00000000000000000000000000000000",
327 => "00000000000000000000000000000000",
328 => "00000000000000000000000000000000",
329 => "00000000000000000000000000000000",
330 => "00000000000000000000000000000000",
331 => "00000000000000000000000000000000",
332 => "00000000000000000000000000000000",
```

```
381 333 => "00000000000000000000000000000000",
382 334 => "00000000000000000000000000000000",
383 335 => "00000000000000000000000000000000",
384 336 => "00000000000000000000000000000000",
385 337 => "00000000000000000000000000000000",
386 338 => "00000000000000000000000000000000",
387 339 => "00000000000000000000000000000000",
388 340 => "00000000000000000000000000000000",
389 341 => "00000000000000000000000000000000",
390 342 => "00000000000000000000000000000000",
391 343 => "00000000000000000000000000000000",
392 344 => "00000000000000000000000000000000",
393 345 => "00000000000000000000000000000000",
394 346 => "00000000000000000000000000000000",
395 347 => "00000000000000000000000000000000",
396 348 => "00000000000000000000000000000000",
397 349 => "00000000000000000000000000000000",
398 350 => "00000000000000000000000000000000",
399 351 => "00000000000000000000000000000000",
400 352 => "00000000000000000000000000000000",
401 353 => "00000000000000000000000000000000",
402 354 => "00000000000000000000000000000000",
403 355 => "00000000000000000000000000000000",
404 356 => "00000000000000000000000000000000",
405 357 => "00000000000000000000000000000000",
406 358 => "00000000000000000000000000000000",
407 359 => "00000000000000000000000000000000",
408 360 => "00000000000000000000000000000000",
409 361 => "00000000000000000000000000000000",
410 362 => "00000000000000000000000000000000",
411 363 => "00000000000000000000000000000000",
412 364 => "00000000000000000000000000000000",
413 365 => "00000000000000000000000000000000",
414 366 => "00000000000000000000000000000000",
415 367 => "00000000000000000000000000000000",
416 368 => "00000000000000000000000000000000",
417 369 => "00000000000000000000000000000000",
418 370 => "00000000000000000000000000000000",
419 371 => "00000000000000000000000000000000",
420 372 => "00000000000000000000000000000000",
421 373 => "00000000000000000000000000000000",
422 374 => "00000000000000000000000000000000",
423 375 => "00000000000000000000000000000000",
424 376 => "00000000000000000000000000000000",
425 377 => "00000000000000000000000000000000",
426 378 => "00000000000000000000000000000000",
427 379 => "00000000000000000000000000000000",
428 380 => "00000000000000000000000000000000",
429 381 => "00000000000000000000000000000000",
```

```
430 382 => "00000000000000000000000000000000",
431 383 => "00000000000000000000000000000000",
432 384 => "00000000000000000000000000000000",
433 385 => "00000000000000000000000000000000",
434 386 => "00000000000000000000000000000000",
435 387 => "00000000000000000000000000000000",
436 388 => "00000000000000000000000000000000",
437 389 => "00000000000000000000000000000000",
438 390 => "00000000000000000000000000000000",
439 391 => "00000000000000000000000000000000",
440 392 => "00000000000000000000000000000000",
441 393 => "00000000000000000000000000000000",
442 394 => "00000000000000000000000000000000",
443 395 => "00000000000000000000000000000000",
444 396 => "00000000000000000000000000000000",
445 397 => "00000000000000000000000000000000",
446 398 => "00000000000000000000000000000000",
447 399 => "00000000000000000000000000000000",
448 400 => "00000000000000000000000000000000",
449 401 => "00000000000000000000000000000000",
450 402 => "00000000000000000000000000000000",
451 403 => "00000000000000000000000000000000",
452 404 => "00000000000000000000000000000000",
453 405 => "00000000000000000000000000000000",
454 406 => "00000000000000000000000000000000",
455 407 => "00000000000000000000000000000000",
456 408 => "00000000000000000000000000000000",
457 409 => "00000000000000000000000000000000",
458 410 => "00000000000000000000000000000000",
459 411 => "00000000000000000000000000000000",
460 412 => "00000000000000000000000000000000",
461 413 => "00000000000000000000000000000000",
462 414 => "00000000000000000000000000000000",
463 415 => "00000000000000000000000000000000",
464 416 => "00000000000000000000000000000000",
465 417 => "00000000000000000000000000000000",
466 418 => "00000000000000000000000000000000",
467 419 => "00000000000000000000000000000000",
468 420 => "00000000000000000000000000000000",
469 421 => "00000000000000000000000000000000",
470 422 => "00000000000000000000000000000000",
471 423 => "00000000000000000000000000000000",
472 424 => "00000000000000000000000000000000",
473 425 => "00000000000000000000000000000000",
474 426 => "00000000000000000000000000000000",
475 427 => "00000000000000000000000000000000",
476 428 => "00000000000000000000000000000000",
477 429 => "00000000000000000000000000000000",
478 430 => "00000000000000000000000000000000",
```

```
479 | 431 => "00000000000000000000000000000000",
480 | 432 => "00000000000000000000000000000000",
481 | 433 => "00000000000000000000000000000000",
482 | 434 => "00000000000000000000000000000000",
483 | 435 => "00000000000000000000000000000000",
484 | 436 => "00000000000000000000000000000000",
485 | 437 => "00000000000000000000000000000000",
486 | 438 => "00000000000000000000000000000000",
487 | 439 => "00000000000000000000000000000000",
488 | 440 => "00000000000000000000000000000000",
489 | 441 => "00000000000000000000000000000000",
490 | 442 => "00000000000000000000000000000000",
491 | 443 => "00000000000000000000000000000000",
492 | 444 => "00000000000000000000000000000000",
493 | 445 => "00000000000000000000000000000000",
494 | 446 => "00000000000000000000000000000000",
495 | 447 => "00000000000000000000000000000000",
496 | 448 => "00000000000000000000000000000000",
497 | 449 => "00000000000000000000000000000000",
498 | 450 => "00000000000000000000000000000000",
499 | 451 => "00000000000000000000000000000000",
500 | 452 => "00000000000000000000000000000000",
501 | 453 => "00000000000000000000000000000000",
502 | 454 => "00000000000000000000000000000000",
503 | 455 => "00000000000000000000000000000000",
504 | 456 => "00000000000000000000000000000000",
505 | 457 => "00000000000000000000000000000000",
506 | 458 => "00000000000000000000000000000000",
507 | 459 => "00000000000000000000000000000000",
508 | 460 => "00000000000000000000000000000000",
509 | 461 => "00000000000000000000000000000000",
510 | 462 => "00000000000000000000000000000000",
511 | 463 => "00000000000000000000000000000000",
512 | 464 => "00000000000000000000000000000000",
513 | 465 => "00000000000000000000000000000000",
514 | 466 => "00000000000000000000000000000000",
515 | 467 => "00000000000000000000000000000000",
516 | 468 => "00000000000000000000000000000000",
517 | 469 => "00000000000000000000000000000000",
518 | 470 => "00000000000000000000000000000000",
519 | 471 => "00000000000000000000000000000000",
520 | 472 => "00000000000000000000000000000000",
521 | 473 => "00000000000000000000000000000000",
522 | 474 => "00000000000000000000000000000000",
523 | 475 => "00000000000000000000000000000000",
524 | 476 => "00000000000000000000000000000000",
525 | 477 => "00000000000000000000000000000000",
526 | 478 => "00000000000000000000000000000000",
527 | 479 => "00000000000000000000000000000000",
```

```
480 => "00000000000000000000000000000000",
481 => "00000000000000000000000000000000",
482 => "00000000000000000000000000000000",
483 => "00000000000000000000000000000000",
484 => "00000000000000000000000000000000",
485 => "00000000000000000000000000000000",
486 => "00000000000000000000000000000000",
487 => "00000000000000000000000000000000",
488 => "00000000000000000000000000000000",
489 => "00000000000000000000000000000000",
490 => "00000000000000000000000000000000",
491 => "00000000000000000000000000000000",
492 => "00000000000000000000000000000000",
493 => "00000000000000000000000000000000",
494 => "00000000000000000000000000000000",
495 => "00000000000000000000000000000000",
496 => "00000000000000000000000000000000",
497 => "00000000000000000000000000000000",
498 => "00000000000000000000000000000000",
499 => "00000000000000000000000000000000",
500 => "00000000000000000000000000000000",
501 => "00000000000000000000000000000000",
502 => "00000000000000000000000000000000",
503 => "00000000000000000000000000000000",
504 => "00000000000000000000000000000000",
505 => "00000000000000000000000000000000",
506 => "00000000000000000000000000000000",
507 => "00000000000000000000000000000000",
508 => "00000000000000000000000000000000",
509 => "00000000000000000000000000000000",
510 => "00000000000000000000000000000000",
511 => "00000000000000000000000000000000",
others => (others => '0')
--END_WORDS_ENTRY
    );

begin  -- architecture dataflow

  word <= words(to_integer(unsigned(address)));

end architecture dataflow;
```

Codice Componente 12.3: Definizione del componente control_*store*

```
--------------------------------------------------------------------------------
    VHDL--
-- Copyright (C) 2019 Alberto Moriconi
--
```

```
4  -- This program is free software: you can redistribute it and/or modify it
       under
5  -- the terms of the GNU General Public License as published by the Free
       Software
6  -- Foundation, either version 3 of the License, or (at your option) any
       later
7  -- version.
8  --
9  -- This program is distributed in the hope that it will be useful, but
       WITHOUT
10 -- ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
       FITNESS
11 -- FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
12 -- details.
13 --
14 -- You should have received a copy of the GNU General Public License along
       with
15 -- this program. If not, see <http://www.gnu.org/licenses/>.
16 --
       ----------------------------------------------------------------------------

17 --! @file control_unit.vhd
18 --! @author Alberto Moriconi
19 --! @date 2019-05-11
20 --! @brief Processor control unit
21 --
       ----------------------------------------------------------------------------

22
23 library ieee;
24 use ieee.std_logic_1164.all;
25 use ieee.numeric_std.all;
26
27 use work.common_defs.all;
28
29 --! Processor control unit
30
31 --! The control unit provides the control signals to the ALU based on the
       stored
32 --! microprogram.
33 --!
34 --! It receives the content of the MBR register and the N/Z ALU flags from
       the
35 --! datapath and produces control signals for ALU, C and B bus and memory
36 --! operations based on the addressed microinstruction format.
37 entity control_unit is
38   port (
39     --! Clock
40     clk              : in  std_logic;
```

```
41      --! Synchronous active-high reset
42      reset            : in  std_logic;
43      --! Content of the MBR register
44      mbr_reg_in       : in  mbr_data_type;
45      --! ALU negative flag
46      alu_n_flag       : in  std_logic;
47      --! ALU zero flag
48      alu_z_flag       : in  std_logic;
49      --! Control signals for the ALU
50      alu_control      : out alu_ctrl_type;
51      --! Control signals for the C bus
52      c_to_reg_control : out c_ctrl_type;
53      --! Control signals for memory operations
54      mem_control      : out mem_ctrl_type;
55      --! Control signals for the B bus
56      reg_to_b_control : out b_ctrl_type
57      );
58  end entity control_unit;
59
60  --! Behavioral architecture for the control unit
61  architecture behavioral of control_unit is
62
63    -- Registers
64    signal mpc_virtual_reg : ctrl_str_addr_type;
65    signal mir_reg         : ctrl_str_word_type;
66    signal n_ff            : std_logic;
67    signal z_ff            : std_logic;
68
69    -- Signals
70    signal control_store_word   : ctrl_str_word_type;
71    signal ctrl_nxt_addr_no_msb : mbr_data_type;
72    signal jmpc_addr            : mbr_data_type;
73    signal high_bit             : std_logic;
74    signal reg_to_b_decoder_out : b_ctrl_type;
75
76  begin  -- architecture behavioral
77
78    -- Control store instantiation
79    control_store : entity work.control_store
80      port map (
81        address => mpc_virtual_reg,
82        word    => control_store_word);
83
84    -- Registers
85    reg_proc : process(clk) is
86    begin
87      if rising_edge(clk) then
88        if reset = '1' then
89          mir_reg <= "00110010100000000000000000000001001"  -- Entry point
```

```
            may change in future versions

      n_ff    <= '0';
        z_ff    <= '0';
      else
        mir_reg <= control_store_word;
        n_ff    <= alu_n_flag;
        z_ff    <= alu_z_flag;
      end if;
    end if;
  end process reg_proc;

  -- MPC virtual register
  ctrl_nxt_addr_no_msb <= mir_reg(ctrl_nxt_addr_no_msb_type'range);
  jmpc_addr            <= ctrl_nxt_addr_no_msb or mbr_reg_in
              when mir_reg(ctrl_jmpc) = '1' else ctrl_nxt_addr_no_msb;
  high_bit        <= (alu_n_flag and mir_reg(ctrl_jamn)) or (alu_z_flag and
     mir_reg(ctrl_jamz));
  mpc_virtual_reg <= (mir_reg(ctrl_nxt_addr_msb) or high_bit) & jmpc_addr;

  -- B_BUS control decoder
  reg_to_b_decoder : process(mir_reg(ctrl_b'range)) is
  begin
    reg_to_b_decoder_out <= (others => '0');

    if unsigned(mir_reg(ctrl_b'range)) < b_ctrl_width then
      reg_to_b_decoder_out(to_integer(unsigned(mir_reg(ctrl_b'range)))) <=
         '1';
    end if;
  end process reg_to_b_decoder;

  -- Output to datapath
  alu_control      <= mir_reg(ctrl_alu'range);
  c_to_reg_control <= mir_reg(ctrl_c'range);
  mem_control      <= mir_reg(ctrl_mem'range);
  reg_to_b_control <= reg_to_b_decoder_out;

end architecture behavioral;
```

Codice Componente 12.4: Definizione del componente control$_unit$

```
------------------------------------------------------------------------
    VHDL--
-- Copyright (C) 2019 Alberto Moriconi
--
-- This program is free software: you can redistribute it and/or modify it
    under
-- the terms of the GNU General Public License as published by the Free
    Software
```

```
6  -- Foundation, either version 3 of the License, or (at your option) any
       later
7  -- version.
8  --
9  -- This program is distributed in the hope that it will be useful, but
       WITHOUT
10 -- ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
       FITNESS
11 -- FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
12 -- details.
13 --
14 -- You should have received a copy of the GNU General Public License along
       with
15 -- this program. If not, see <http://www.gnu.org/licenses/>.
16 --
       ----------------------------------------------------------------------------

17 --! @file datapath.vhd
18 --! @author Alberto Moriconi
19 --! @date 2019-05-01
20 --! @brief Processor datapath
21 --
       ----------------------------------------------------------------------------

22
23 library ieee;
24 use ieee.std_logic_1164.all;
25
26 use work.common_defs.all;
27
28 --! Processor datapath
29
30 --! # Inputs
31 --!
32 --! The datapath contains the ALU, the processor registers and the memory
       interface.
33 --!
34 --! It provides to the control unit the content of the MBR register and the
       N/Z
35 --! ALU flags, and receives control signals for ALU, C and B bus and memory.
36 --! It also produces the signals that operate the memory (addresses, data
       and enables).
37 entity datapath is
38   port (
39     --! Clock
40     clk              : in  std_logic;
41     --! Synchronous active-high reset
42     reset            : in  std_logic;
43     --! Control signals for the ALU
```

```
44      alu_control      : in  alu_ctrl_type;
45      --! Control signals for the C bus
46      c_to_reg_control : in  c_ctrl_type;
47      --! Control signals for memory operations
48      mem_control      : in  mem_ctrl_type;
49      --! Control signals for the B bus
50      reg_to_b_control : in  b_ctrl_type;
51      --! Content of the MBR register
52      mbr_reg_out      : out mbr_data_type;
53      --! ALU negative flag
54      alu_n_flag       : out std_logic;
55      --! ALU zero flag
56      alu_z_flag       : out std_logic;
57      --! Memory data write enable
58      mem_data_we      : out std_logic;
59      --! Port for memory data read
60      mem_data_in      : in  reg_data_type;
61      --! Port for memory data write
62      mem_data_out     : out reg_data_type;
63      --! Memory address for memory data operations
64      mem_data_addr    : out reg_data_type;
65      --! Port for memory instruction read
66      mem_instr_in     : in  mbr_data_type;
67      --! Memory address for memory instruction fetch
68      mem_instr_addr   : out reg_data_type
69      );
70  end entity datapath;
71
72  --! Structural architecture for the datapath
73  architecture behavioral of datapath is
74
75    -- Registers
76    signal sp_reg   : reg_data_type;
77    signal lv_reg   : reg_data_type;
78    signal cpp_reg  : reg_data_type;
79    signal tos_reg  : reg_data_type;
80    signal opc_reg  : reg_data_type;
81    signal h_reg    : reg_data_type;
82    signal mar_reg  : reg_data_type;
83    signal mdr_reg  : reg_data_type;
84    signal pc_reg   : reg_data_type;
85    signal mbr_reg  : mbr_data_type;
86    signal rd_ff    : std_logic;
87    signal fetch_ff : std_logic;
88    signal wr_ff    : std_logic;
89
90    -- Signals
91    signal a_bus : reg_data_type;
92    signal b_bus : reg_data_type;
```

```
93    signal c_bus : reg_data_type;
94
95    signal mbr_u : reg_data_type;
96    signal mbr_s : reg_data_type;
97
98  begin  -- architecture structural
99
100   -- ALU instantiation
101   alu : entity work.alu
102     port map (
103       control       => alu_control,
104       operand_a     => a_bus,
105       operand_b     => b_bus,
106       sh_result     => c_bus,
107       negative_flag => alu_n_flag,
108       zero_flag     => alu_z_flag);
109
110   -- Processor registers
111   reg_proc : process(clk) is
112   begin
113     if rising_edge(clk) then
114       if reset = '1' then
115         sp_reg  <= x"00000101";
116         lv_reg  <= x"00000100";
117         cpp_reg <= x"00000080";
118         tos_reg <= (others => '0');
119         opc_reg <= (others => '0');
120         h_reg   <= (others => '0');
121         mar_reg <= (others => '0');
122         mdr_reg <= (others => '0');
123         pc_reg  <= (others => '0');
124         mbr_reg <= (others => '0');
125
126         rd_ff    <= '0';
127         fetch_ff <= '0';
128         wr_ff    <= '0';
129       else
130         if c_to_reg_control(c_ctrl_mar) = '1' then
131           mar_reg <= c_bus;
132         end if;
133         if c_to_reg_control(c_ctrl_pc) = '1' then
134           pc_reg <= c_bus;
135         end if;
136         if c_to_reg_control(c_ctrl_sp) = '1' then
137           sp_reg <= c_bus;
138         end if;
139         if c_to_reg_control(c_ctrl_lv) = '1' then
140           lv_reg <= c_bus;
141         end if;
```

```
142         if c_to_reg_control(c_ctrl_tos) = '1' then
143           tos_reg <= c_bus;
144         end if;
145         if c_to_reg_control(c_ctrl_cpp) = '1' then
146           cpp_reg <= c_bus;
147         end if;
148         if c_to_reg_control(c_ctrl_h) = '1' then
149           h_reg <= c_bus;
150         end if;
151         if c_to_reg_control(c_ctrl_opc) = '1' then
152           opc_reg <= c_bus;
153         end if;
154
155         -- MDR can also receive data from memory
156         if c_to_reg_control(c_ctrl_mdr) = '1' then
157           mdr_reg <= c_bus;
158         elsif rd_ff = '1' then
159           mdr_reg <= mem_data_in;
160         end if;
161
162         -- MBR can't be written from C bus
163         if fetch_ff = '1' then
164           mbr_reg <= mem_instr_in;
165         end if;
166
167         -- Effects on regs on next clock cycle
168         rd_ff    <= mem_control(mem_ctrl_read);
169         fetch_ff <= mem_control(mem_ctrl_fetch);
170         wr_ff    <= mem_control(mem_ctrl_write);
171       end if;
172     end if;
173   end process reg_proc;
174
175   -- A bus is reserved for register H
176   a_bus <= h_reg;
177
178   -- B bus MUX
179   mbr_u(mbr_s_ext'range)      <= (others => '0');
180   mbr_u(mbr_data_type'range)  <= mbr_reg;
181   mbr_s(mbr_s_ext'range)      <= (others => mbr_reg(mbr_data_type'high));
182   mbr_s(mbr_data_type'range)  <= mbr_reg;
183
184   with reg_to_b_control select b_bus <=
185     mdr_reg          when b_ctrl_mdr,
186     pc_reg           when b_ctrl_pc,
187     mbr_s            when b_ctrl_mbr,
188     mbr_u            when b_ctrl_mbru,
189     sp_reg           when b_ctrl_sp,
190     lv_reg           when b_ctrl_lv,
```

```
191     cpp_reg          when b_ctrl_cpp,
192     tos_reg          when b_ctrl_tos,
193     opc_reg          when b_ctrl_opc,
194     (others => '0') when others;
195
196   -- Output
197   mbr_reg_out    <= mbr_reg;
198   mem_data_out   <= mdr_reg;
199   mem_data_addr  <= mar_reg;
200   mem_data_we    <= wr_ff;
201   mem_instr_addr <= pc_reg;
202
203 end architecture behavioral;
```

Codice Componente 12.5: Definizione del componente datapath

```
1  ----------------------------------------------------------------------------
   VHDL--
2  -- Copyright (C) 2019 Alberto Moriconi
3  --
4  -- This program is free software: you can redistribute it and/or modify it
   under
5  -- the terms of the GNU General Public License as published by the Free
   Software
6  -- Foundation, either version 3 of the License, or (at your option) any
   later
7  -- version.
8  --
9  -- This program is distributed in the hope that it will be useful, but
   WITHOUT
10 -- ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
   FITNESS
11 -- FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
12 -- details.
13 --
14 -- You should have received a copy of the GNU General Public License along
   with
15 -- this program. If not, see <http://www.gnu.org/licenses/>.
16 --
   ----------------------------------------------------------------------------

17 --! @file dp_ar_ram.vhd
18 --! @author Alberto Moriconi
19 --! @date 2019-05-20
20 --! @brief Dual port, asynchronous read RAM for amic-0 based systems.
21 --
   ----------------------------------------------------------------------------

22
```

```
23 library ieee;
24 use ieee.std_logic_1164.all;
25 use ieee.numeric_std.all;
26
27 use work.common_defs.all;
28
29 --! Data memory for amic-0 based systems
30
31 --! Dual port, asynchronous read RAM for amic-0 based systems.
32 entity dp_ar_ram is
33   port (
34     --! Clock
35     clk        : in  std_logic;
36     --! Write enable 1
37     we_1       : in  std_logic;
38     --! Port for memory write 1
39     data_in_1  : in  reg_data_type;
40     --! Port for memory read 1
41     data_out_1 : out reg_data_type;
42     --! Address for memory operations 1
43     address_1  : in  reg_data_type;
44     --! Write enable 2
45     we_2       : in  std_logic;
46     --! Port for memory write 2
47     data_in_2  : in  reg_data_type;
48     --! Port for memory read 2
49     data_out_2 : out mbr_data_type;
50     --! Address for memory operations 2
51     address_2  : in  reg_data_type
52     );
53 end entity dp_ar_ram;
54
55 --! Dataflow architecture for the control store
56 architecture behavioral of dp_ar_ram is
57
58   -- Signals
59   signal t_address_1  : integer := 0;
60   signal t_address_2  : integer := 0;
61   signal wa_address_2 : reg_data_type;
62   signal t_data_out_2 : reg_data_type;
63
64   -- RAM content
65   signal mem : dp_ar_ram_type := (
66 -indirizzi che vengono modificati dall'istruzione make create_ram
67 --BEGIN_WORDS_ENTRY
68 --128 => "00000000000000000000000000000000",
69 --129 => "00000000000000000000000001010110",
70 --130 => "00000000000000000000000000000011",
71 --131 => "00000000000000000000000000100010",
```

```
0 => "00000010000000000000000100000000",
1 => "00000001001101100101011000010000",
--2 => "00010101000000010000000000100000",
--3 => "00000110000000001010000100000001",
--4 => "00010000000011110000000010100111",
--5 => "00100000000000010001010100000000",
--6 => "00000000101110010000001000000000",
--7 => "10100111000000100011011000000011",
--8 => "00000011000000000000000000000000",
--9 => "00000001000101010000000000000000",
--10 => "00001111000100000000001000010101",
--11 => "00000000101011010110010101100101",
others => (others => '0')
--END_WORDS_ENTRY
    );

begin  -- architecture behavioral

  wa_address_2 <= "00" & address_2(reg_data_type'high downto 2);
  t_address_1 <= to_integer(unsigned(address_1));
  t_address_2 <= to_integer(unsigned(wa_address_2));

  mem_proc : process(clk) is
  begin
    if (rising_edge(clk)) then
      if (we_1 = '1') then
        mem(t_address_1) <= data_in_1;
      elsif (we_2 = '1') then
        mem(t_address_2) <= data_in_2;
      end if;
    end if;
  end process;

  data_out_1   <= mem(t_address_1);
  t_data_out_2 <= mem(t_address_2);

  with address_2(1 downto 0) select data_out_2 <=
    t_data_out_2(7 downto 0)   when "00",
    t_data_out_2(15 downto 8)  when "01",
    t_data_out_2(23 downto 16) when "10",
    t_data_out_2(31 downto 24) when "11",
    (others => '0')            when others;

end architecture behavioral;
```

Codice Componente 12.6: Definizione del componente $\mathrm{dp}_a r_r am$

```
--------------------------------------------------------------------------
    VHDL--
```

```
2  -- Copyright (C) 2019 Alberto Moriconi
3  --
4  -- This program is free software: you can redistribute it and/or modify it
       under
5  -- the terms of the GNU General Public License as published by the Free
       Software
6  -- Foundation, either version 3 of the License, or (at your option) any
       later
7  -- version.
8  --
9  -- This program is distributed in the hope that it will be useful, but
       WITHOUT
10 -- ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
       FITNESS
11 -- FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
12 -- details.
13 --
14 -- You should have received a copy of the GNU General Public License along
       with
15 -- this program. If not, see <http://www.gnu.org/licenses/>.
16 --
       ----------------------------------------------------------------------------

17 --! @file processor.vhd
18 --! @author Alberto Moriconi
19 --! @date 2019-05-11
20 --! @brief The amic-0 processor
21 --
       ----------------------------------------------------------------------------

22
23 library ieee;
24 use ieee.std_logic_1164.all;
25
26 use work.common_defs.all;
27
28 --! The amic-0 processor
29
30 --! The processor exposes the memory interface defined in the datapath.
31 entity processor is
32   port (
33     --! Clock
34     clk            : in  std_logic;
35     --! Synchronous active-high reset
36     reset          : in  std_logic;
37     --! Memory data write enable
38     mem_data_we    : out std_logic;
39     --! Port for memory data read
40     mem_data_in    : in  reg_data_type;
```

```
41      --! Port for memory data write
42      mem_data_out    : out reg_data_type;
43      --! Memory address for memory data operations
44      mem_data_addr   : out reg_data_type;
45      --! Port for memory instruction read
46      mem_instr_in    : in  mbr_data_type;
47      --! Memory address for memory instruction fetch
48      mem_instr_addr : out reg_data_type
49      );
50  end entity processor;
51
52  -- Structural architecture for the processor
53  architecture structural of processor is
54
55    -- Signals
56    signal alu_control_t      : alu_ctrl_type;
57    signal c_to_reg_control_t : c_ctrl_type;
58    signal reg_to_b_control_t : b_ctrl_type;
59    signal mem_control_t      : mem_ctrl_type;
60    signal mbr_reg_t          : mbr_data_type;
61    signal alu_n_flag_t       : std_logic;
62    signal alu_z_flag_t       : std_logic;
63
64  begin  -- architecture structural
65
66    -- Control unit instantiation
67    control_unit : entity work.control_unit
68      port map (
69        clk              => clk,
70        reset            => reset,
71        mbr_reg_in       => mbr_reg_t,
72        alu_n_flag       => alu_n_flag_t,
73        alu_z_flag       => alu_z_flag_t,
74        alu_control      => alu_control_t,
75        c_to_reg_control => c_to_reg_control_t,
76        mem_control      => mem_control_t,
77        reg_to_b_control => reg_to_b_control_t);
78
79    -- Datapath instantiation
80    datapath : entity work.datapath
81      port map (
82        clk              => clk,
83        reset            => reset,
84        alu_control      => alu_control_t,
85        c_to_reg_control => c_to_reg_control_t,
86        mem_control      => mem_control_t,
87        reg_to_b_control => reg_to_b_control_t,
88        mbr_reg_out      => mbr_reg_t,
89        alu_n_flag       => alu_n_flag_t,
```

```
90        alu_z_flag        => alu_z_flag_t,
91        mem_data_we       => mem_data_we,
92        mem_data_in       => mem_data_in,
93        mem_data_out      => mem_data_out,
94        mem_data_addr     => mem_data_addr,
95        mem_instr_in      => mem_instr_in,
96        mem_instr_addr    => mem_instr_addr);
97
98  end architecture structural;
```

Codice Componente 12.7: Definizione del componente processor

```
1  ------------------------------------------------------------------------
   VHDL--
2  -- Copyright (C) 2019 Alberto Moriconi
3  --
4  -- This program is free software: you can redistribute it and/or modify it
   under
5  -- the terms of the GNU General Public License as published by the Free
   Software
6  -- Foundation, either version 3 of the License, or (at your option) any
   later
7  -- version.
8  --
9  -- This program is distributed in the hope that it will be useful, but
   WITHOUT
10 -- ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
   FITNESS
11 -- FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
12 -- details.
13 --
14 -- You should have received a copy of the GNU General Public License along
   with
15 -- this program. If not, see <http://www.gnu.org/licenses/>.
16 --
   ------------------------------------------------------------------------

17 --! @file system.vhd
18 --! @author Alberto Moriconi
19 --! @date 2019-05-32
20 --! @brief A sample system based on amic-0
21 --
   ------------------------------------------------------------------------

22
23 library ieee;
24 use ieee.std_logic_1164.all;
25
26 use work.common_defs.all;
```

```
27
28 --! A sample system based on amic-0
29
30 --! The system contains the processor and a RAM
31 entity system is
32   port (
33     --! Clock
34     clk            : in  std_logic;
35     --! Synchronous active-high reset
36     reset    : in  std_logic;
37     --! Memory data output
38     data_out : out reg_data_type
39     );
40 end entity system;
41
42 -- Structural architecture for the system
43 architecture structural of system is
44
45   signal mem_data_we    : std_logic;
46   signal mem_data_in    : reg_data_type;
47   signal mem_data_out   : reg_data_type;
48   signal mem_data_addr  : reg_data_type;
49   signal mem_instr_in   : mbr_data_type;
50   signal mem_instr_addr : reg_data_type;
51
52 begin -- architecture structural
53
54   -- Processor instantiation
55   processor : entity work.processor
56     port map (
57       clk            => clk,
58       reset          => reset,
59       mem_data_we    => mem_data_we,
60       mem_data_in    => mem_data_in,
61       mem_data_out   => mem_data_out,
62       mem_data_addr  => mem_data_addr,
63       mem_instr_in   => mem_instr_in,
64       mem_instr_addr => mem_instr_addr);
65
66   -- RAM instantiation
67   dp_ar_ram : entity work.dp_ar_ram
68     port map (
69       clk        => clk,
70       we_1       => mem_data_we,
71       data_in_1  => mem_data_out,
72       data_out_1 => mem_data_in,
73       address_1  => mem_data_addr,
74       we_2       => '0',
75       data_in_2  => (others => '0'),
```

```
76        data_out_2 => mem_instr_in,
77        address_2  => mem_instr_addr);
78
79    -- Output
80    data_out <= mem_data_out;
81
82  end architecture structural;
```

Codice Componente 12.8: Definizione del componente system

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity anodes_manager is
5       Port ( counter : in  STD_LOGIC_VECTOR (2 downto 0);
6              enable_digit : in  STD_LOGIC_VECTOR (7 downto 0);
7              anodes : out  STD_LOGIC_VECTOR (7 downto 0)
8            );
9   end anodes_manager;
10
11  architecture Behavioral of anodes_manager is
12
13  signal anodes_switching : std_logic_vector(7 downto 0) := (others => '0');
14
15  begin
16
17  anodes(7 downto 0) <= not anodes_switching OR not enable_digit;
18
19
20  anodes_process: process(counter, enable_digit)
21  begin
22    --a seconda del valore di caunter le cifre si illuminano una alla volta da
          destra a sinistra
23    case counter is -- devo utilizzare tutte le 8 cifre del display per poter
         visualizzare il valore del registro a 32 bit della ram
24      when "000" =>
25        anodes_switching <= x"01";
26      when "001" =>
27        anodes_switching <= x"02";
28      when "010" =>
29        anodes_switching <= x"04";
30      when "011" =>
31        anodes_switching <= x"08";
32      when "100" =>
33        anodes_switching <= "00010000";
34      when "101" =>
35        anodes_switching <= "00100000";
36      when "110" =>
37        anodes_switching <= "01000000";
```

```
38      when "111" =>
39        anodes_switching <= "10000000";
40      when others =>
41        anodes_switching <= (others => '0');
42    end case;
43
44  end process;
45
46
47  end Behavioral;
```

Codice Componente 12.9: Definizione del componente anodes$_m$*anager*

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   use IEEE.NUMERIC_STD.ALL;
5
6   entity cathodes_manager is
7       Port ( counter : in  STD_LOGIC_VECTOR (2 downto 0);
8              value : in  STD_LOGIC_VECTOR (31 downto 0);
9              cathodes : out  STD_LOGIC_VECTOR (7 downto 0));
10  end cathodes_manager;
11
12  architecture Behavioral of cathodes_manager is
13
14  constant zero   : std_logic_vector(6 downto 0) := "1000000";
15  constant one    : std_logic_vector(6 downto 0) := "1111001";
16  constant two    : std_logic_vector(6 downto 0) := "0100100";
17  constant three  : std_logic_vector(6 downto 0) := "0110000";
18  constant four   : std_logic_vector(6 downto 0) := "0011001";
19  constant five   : std_logic_vector(6 downto 0) := "0010010";
20  constant six    : std_logic_vector(6 downto 0) := "0000010";
21  constant seven  : std_logic_vector(6 downto 0) := "1111000";
22  constant eight  : std_logic_vector(6 downto 0) := "0000000";
23  constant nine   : std_logic_vector(6 downto 0) := "0010000";
24  constant a     : std_logic_vector(6 downto 0) := "0001000";
25  constant b      : std_logic_vector(6 downto 0) := "0000011";
26  constant c      : std_logic_vector(6 downto 0) := "1000110";
27  constant d      : std_logic_vector(6 downto 0) := "0100001";
28  constant e      : std_logic_vector(6 downto 0) := "0000110";
29  constant f      : std_logic_vector(6 downto 0) := "0001110";
30
31  alias digit_0 is value (3 downto 0);
32  alias digit_1 is value (7 downto 4);
33  alias digit_2 is value (11 downto 8);
34  alias digit_3 is value (15 downto 12);
35  alias digit_4 is value (19 downto 16);
36  alias digit_5 is value (23 downto 20);
```

```
37 alias digit_6 is value (27 downto 24);
38 alias digit_7 is value (31 downto 28);
39
40
41 signal cathodes_for_digit : std_logic_Vector(6 downto 0) := (others => '0');
42 signal nibble :std_logic_vector(3 downto 0) := (others => '0');
43
44 begin
45
46 digit_switching: process(counter, value)
47
48 begin
49   case counter is
50     when "000" =>
51       nibble <= digit_0;
52     when "001" =>
53       nibble <= digit_1;
54     when "010" =>
55       nibble <= digit_2;
56     when "011" =>
57       nibble <= digit_3;
58     when "100" =>
59       nibble <= digit_4;
60     when "101" =>
61       nibble <= digit_5;
62     when "110" =>
63       nibble <= digit_6;
64     when "111" =>
65       nibble <= digit_7;
66     when others =>
67       nibble <= (others => '0');
68   end case;
69 end process;
70
71 seven_segment_decoder_process: process(nibble)
72   begin
73     case nibble is
74       when "0000" => cathodes_for_digit <= zero;
75       when "0001" => cathodes_for_digit <= one;
76       when "0010" => cathodes_for_digit <= two;
77       when "0011" => cathodes_for_digit <= three;
78       when "0100" => cathodes_for_digit <= four;
79       when "0101" => cathodes_for_digit <= five;
80       when "0110" => cathodes_for_digit <= six;
81       when "0111" => cathodes_for_digit <= seven;
82       when "1000" => cathodes_for_digit <= eight;
83       when "1001" => cathodes_for_digit <= nine;
84       when "1010" => cathodes_for_digit <= a;
85       when "1011" => cathodes_for_digit <= b;
```

```
86        when "1100" => cathodes_for_digit <= c;
87        when "1101" => cathodes_for_digit <= d;
88        when "1110" => cathodes_for_digit <= e;
89        when "1111" => cathodes_for_digit <= f;
90      when others => cathodes_for_digit <= (others => '0');
91      end case;
92    end process seven_segment_decoder_process;
93
94  cathodes (6 downto 0)<= cathodes_for_digit;
95  cathodes (7) <= '1';
96
97  end Behavioral;
```

Codice Componente 12.10: Definizione del componente cathodes$_m$*anager*

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity clock_filter is
5      generic(
6           clock_frequency_in : integer := 100000000;
7           clock_frequency_out : integer := 50000
8           );
9       Port ( clock_in : in  STD_LOGIC;
10          reset_n : in STD_LOGIC;
11             clock_out : out  STD_LOGIC);
12  end clock_filter;
13
14  architecture Behavioral of clock_filter is
15
16  signal clockfx, reset : std_logic := '0';
17
18  constant count_max_value : integer := clock_frequency_in/(
        clock_frequency_out)-1;
19
20  begin
21
22  clock_out <= clockfx;
23  reset <= not reset_n;
24
25  count_for_division: process(clock_in, reset)
26  variable counter : integer range 0 to count_max_value := 0;
27  begin
28
29    if reset = '1' then
30      counter := 0;
31      clockfx <= '0';
32    elsif clock_in'event and clock_in = '1' then
33      if counter = count_max_value then
```

```
34        clockfx <=  '1';
35        counter := 0;
36      else
37        clockfx <=  '0';
38        counter := counter + 1;
39      end if;
40    end if;
41
42  end process;
43
44
45  end Behavioral;
```

Codice Componente 12.11: Definizione del componente clock$_filter$

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity counter_mod4 is -- in realtà è modulo 8, errore di battitura
6       Port ( clock : in  STD_LOGIC;
7              reset_n : in  STD_LOGIC;
8           enable : in STD_LOGIC;
9              counter : out  STD_LOGIC_VECTOR (2 downto 0));
10  end counter_mod4;
11
12  architecture Behavioral of counter_mod4 is
13
14  signal c : std_logic_vector (2 downto 0) := (others => '0');
15  signal reset : std_logic;
16
17  begin
18
19  counter <= c;
20
21  reset <= not reset_n;
22
23  counter_process: process(clock, reset, c)
24  begin
25
26    if reset = '1' then
27      c <= (others => '0');
28    elsif clock'event AND clock = '1' AND enable = '1' then
29      c <= std_logic_vector(unsigned(c) + 1);
30    end if;
31
32  end process;
33
34  end Behavioral;
```

Codice Componente 12.12: Definizione del componente counter$_m od 8$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use work.common_defs.all;
4
5  entity display_on_board is
6    Port(
7        clock : in  STD_LOGIC;
8        reset : in  STD_LOGIC;
9        anodes : out  STD_LOGIC_VECTOR (7 downto 0);
10       cathodes : out  STD_LOGIC_VECTOR (7 downto 0)
11         );
12
13 end display_on_board;
14
15 architecture Structural of display_on_board is
16
17 --signal clock_fx : std_logic := '0';
18
19 COMPONENT display_seven_segments
20 GENERIC(
21         clock_frequency_in : integer := 100000000;
22         clock_frequency_out : integer := 50000
23         );
24   PORT(
25     clock : IN std_logic;
26     reset_n : IN std_logic;
27     value : IN std_logic_vector(31 downto 0);--4 nibble da mostrare
28     anodes : OUT std_logic_vector(7 downto 0);
29     cathodes : OUT std_logic_vector(7 downto 0)
30     );
31 END COMPONENT;
32
33 COMPONENT  sistema -- viene utilizzato un componente sistema che unisce il
       system fornito dal materiale didattico a un clock filter per abbassare
       la frequenza e permettere il corretto svolgimento di tutte le
       operazioni durante i periodi del clock.
34    port (
35     --! Clock
36     clk            : in  std_logic;
37     --! Synchronous active-high reset
38     reset     : in  std_logic;
39     --! Memory data output
40     data_out : out reg_data_type
41     );
42 end component;
43
```

```
44
45
46  signal reset_n : std_logic;
47  signal cu_value : std_logic_vector(31 downto 0);
48
49  begin
50
51  reset_n <= not reset;
52
53  seven_segment_array: display_seven_segments
54  GENERIC MAP(
55    clock_frequency_in => 100000000,
56    clock_frequency_out => 50000
57    )
58    PORT MAP(
59      clock => clock,
60      reset_n => reset_n,
61      value => cu_value,
62      anodes => anodes,
63      cathodes => cathodes
64  );
65
66
67
68  s: sistema PORT MAP(
69      clk => clock,
70      reset => reset,
71      data_out => cu_value
72    );
73
74
75
76
77
78
79  end Structural;
```

Codice Componente 12.13: Definizione del componente display$_o n_b oard$

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity display_seven_segments is
5     Generic(
6           clock_frequency_in : integer := 100000000;
7           clock_frequency_out : integer := 50000
8           );
9       Port ( clock : in  STD_LOGIC;
10            reset_n : in  STD_LOGIC;
```

```
11            value : in  STD_LOGIC_VECTOR (31 downto 0);
12            anodes : out  STD_LOGIC_VECTOR (7 downto 0);
13            cathodes : out  STD_LOGIC_VECTOR (7 downto 0));
14  end display_seven_segments;
15
16  architecture Structural of display_seven_segments is
17
18  signal counter : std_logic_vector(2 downto 0);
19  signal clock_fx : std_logic := '0';
20
21  COMPONENT counter_mod4
22    PORT(
23      clock : in  STD_LOGIC;
24        reset_n : in  STD_LOGIC;
25      enable : in STD_LOGIC;
26        counter : out  STD_LOGIC_VECTOR (2 downto 0)
27      );
28  END COMPONENT;
29
30  COMPONENT cathodes_manager
31    PORT(
32      counter : IN std_logic_vector(2 downto 0);
33      value : IN std_logic_vector(31 downto 0);
34      cathodes : OUT std_logic_vector(7 downto 0)
35      );
36  END COMPONENT;
37
38  COMPONENT anodes_manager
39    PORT(
40
41      counter : IN std_logic_vector(2 downto 0);
42      enable_digit : IN std_logic_vector(7 downto 0);
43      anodes : OUT std_logic_vector(7 downto 0)
44      );
45  END COMPONENT;
46
47  COMPONENT clock_filter
48    GENERIC(
49          clock_frequency_in : integer := 100000000;
50          clock_frequency_out : integer := 50000
51          );
52    PORT(
53      clock_in : IN std_logic;
54        reset_n : in  STD_LOGIC;
55      clock_out : OUT std_logic
56      );
57  END COMPONENT;
58  begin
59  --il clock filter genera un segnale di abilitazione per il contatore mod4
```

```
    che viene usato
--come segnale di conteggio e quindi di fatto fornisce la frequenza con cui
    viene modificata
--la cifra da mostrare

clk_filter: clock_filter GENERIC MAP(
  clock_frequency_in => clock_frequency_in,
  clock_frequency_out => clock_frequency_out
  )
  PORT MAP(
    clock_in => clock,
    reset_n => reset_n,
    clock_out => clock_fx
  );

counter_instance: counter_mod4 port map(
  clock => clock,
  enable => clock_fx,
  reset_n => reset_n,
  counter => counter
);
--il valore di conteggio viene usato dal gestore dei catodi e degli anodi
    per
--selezionare l'anodo da accendere e il suo rispettivo valore
cathodes_instance: cathodes_manager port map(
  counter => counter,
  value => value,
  cathodes => cathodes
);

anodes_instance: anodes_manager port map(
  counter => counter,
  enable_digit => (others => '1'),
  anodes => anodes
);


end Structural;
```

Codice Componente 12.14: Definizione del componente display$_s$even$_s$egment

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

use work.common_defs.all;


entity sistema is
 port (
```

```
9     --! Clock
10    clk             : in  std_logic;
11    --! Synchronous active-high reset
12    reset     : in  std_logic;
13    --! Memory data output
14    data_out : out reg_data_type
15    );
16
17 end sistema;
18
19 architecture structural of sistema is
20
21 signal reset_n : std_logic;
22 signal clk_fx : std_logic;
23
24
25 component system
26  port (
27    --! Clock
28    clk             : in  std_logic;
29    --! Synchronous active-high reset
30    reset     : in  std_logic;
31    --! Memory data output
32    data_out : out reg_data_type
33    );
34 end component;
35
36 component clock_filter
37 generic(
38         clock_frequency_in : integer := 100000000;
39         clock_frequency_out : integer := 50000
40         );
41     Port ( clock_in : in  STD_LOGIC;
42         reset_n : in STD_LOGIC;
43            clock_out : out  STD_LOGIC);
44 end component;
45
46
47
48 begin
49
50 reset_n <= not reset;
51
52 c : clock_filter
53 generic map (
54 clock_frequency_in => 100000000,
55         clock_frequency_out => 50000
56 )
57 port map (
```

```
58 clock_in => clk,
59 reset_n => reset_n,
60 clock_out => clk_fx
61 );
62
63 s : system
64 port map (
65 clk => clk_fx, -- collego il clk del sistema a quello filtrato
66 reset => reset,
67 data_out => data_out
68 );
69
70 end structural;
```

Codice Componente 12.15: Definizione del componente sistema

## 12.3 Simulazione

- IADD

La prima istruzione che viene analizzata nel dettaglio è la IADD, che ha il compito di sommare i primi due valori presenti dello stack.

Valutiamo nel dettaglio il codice scritto in linguaggio mal:

```
iadd = 0x65:
        MAR = SP = SP - 1; rd
        H = TOS
        MDR = TOS = MDR + H; wr; goto main
```

L'indirizzo che viene assegnato all'istruzione ijvm IADD nel control store è 0x65, che deve essere uguale alla sua codifica in linguaggio macchina prodotta dall'assemblatore ijvm, in modo tale che quando la microistruzione main esegue il fetch del byte 0x65, l'MPC è settato a 0x65 e il microcodice relativo all'istruzione IADD viene eseguito.

Nel dettaglio l'istruzione come prima cosa decrementa lo stack pointer e lo salva sia nello SP stesso sia in MAR, in modo tale da ottenere al successivo ciclo di clock su MDR il secondo elemento dello stack, che sarà il nostro secondo operando. Successivamente viene copiato il valore attuale di TOS ( che per l'invariante è sempre il valore dell'elemento in testa allo stack, quindi il primo operando) nel registro H, in modo tale da prepararlo per la somma. Nell'ultimo ciclo viene effettuata la somma tra MDR (che ora contiene il secondo operando) e H (che contiene il primo operando), e il risultato viene salvato sia in MDR (per permettere la scrittura in memoria) sia in TOS (per mantenere la veridicità dell'invariante).

Per testare l'istruzione sul processore è necessario scrivere il codice in linguaggio assembly ajvm aggiungendo anche il push dei due operandi:

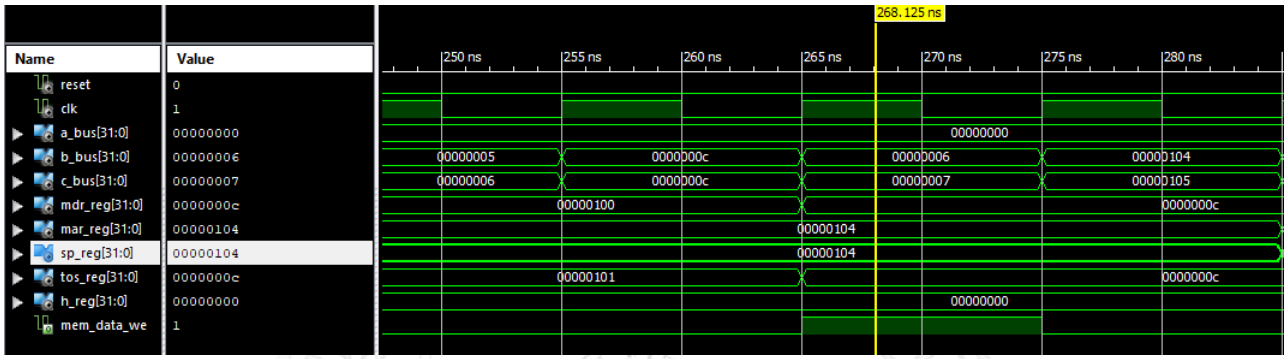
```
  .main
  BIPUSH 0xC
 BIPUSH 0xB
IADD
HALT
  .endmethod
```

Dopo aver scritto questo codice nel file program.ajvm, è necessario lanciare il comando "make create_ram" che va a modificare il contenuto della ram nel seguente modo:

```
 -- RAM content
 signal mem : dp_ar_ram_type := (
--BEGIN_WORDS_ENTRY
128 => "00000000000000000000000000000000",
0 => "00000000000000000000000100000000",
1 => "00001011000100000000110000010000",
2 => "00000000000000001010011101100101",
others => (others => '0')
--END_WORDS_ENTRY
   );
```

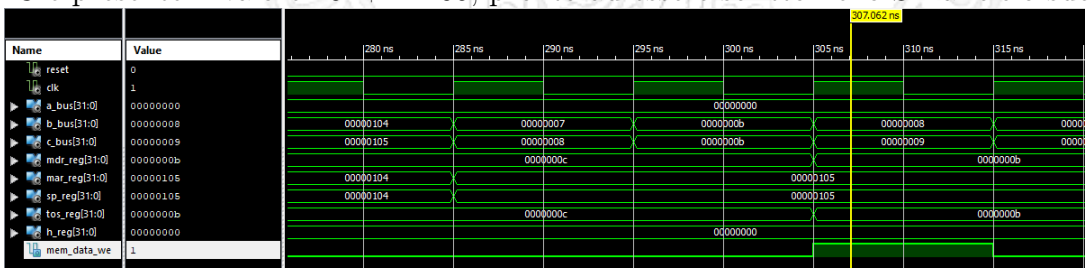
Valutiamo ora come opera la IADD in simulazione:

I primi 20 cicli di clock circa sono necessari per l'inizializzazione del processore, e quindi vengono ignorati nella trattazione.



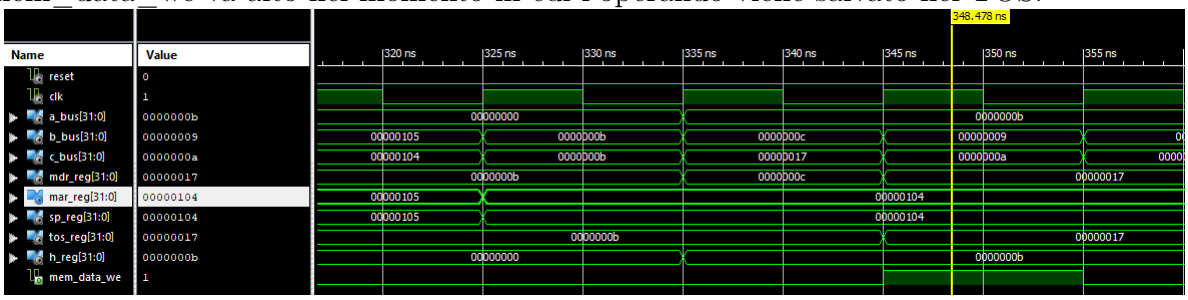

Il primo ciclo di clock rilevante è quello al ns 255: dopo che lo SP è stato incrementato al fronte di clock precedente (ns 245, involontariamente tagliato nello screen, ma presente dell'esempio successivo della IOR), viene effettuato il push del primo operando dopo aver scritto il valore dello SP nel registro MAR, in modo che al successivo ciclo otteniamo sul registro MDR l'operando che viene prima letto dal bus B, passa senza modifiche per l'alu e arriva al bus C, che lo scrive sul registro MDR e sul registro TOS al ns 265.

Al ns 275 viene poi letto dal bus b il valore dello SP e viene incrementato dall'ALU, infatti sul bus C è presente il valore 104+1=105, pronto ad essere scritto nello SP al ciclo successivo.




Al ciclo di clock successivo vediamo come lo SP venga aggiornato al valore incrementato presente

sul bus C, in modo da poter fare il push anche del secondo operando con le stesse operazioni gia descritte al passo precedente. Durante il push è necessario fare una scrittura in memoria, quindi il mem_data_we va alto nel momento in cui l'operando viene salvato nel TOS.




Nella parte finale iniziano le operazioni necessarie per la IADD: lo SP viene prima decrementato (ns 325, sempre dopo aver iniziato l'operazione al ciclo precedente con lettura da bus B e scrittura sul bus C dell'uscita dell'ALU che ha decrementato), poi lo SP decrementato dal bus C viene scritto sia sullo SP sia sul MAR, in modo tale che al ciclo dopo sul registro MDR sia presente il secondo operando. Nel ciclo successivo (ns 335) viene copiato il registro TOS nel registro H, e nel ciclo ancora dopo (ns 345) viene completata la somma e il risultato viene scritto sia nel registro TOS per l'invariante sia nel registro MDR per permetterne la scrittura in memoria (infatti si alza il mem_data_we).

- IOR

La seconda istruzione che viene analizzata nel dettaglio è la IOR, che fa la or bit a bit dei primi due valori presenti dello stack.

Valutiamo nel dettaglio il codice scritto in linguaggio mal:

```
ior = 0xB6:
        MAR = SP = SP - 1; rd
        H = TOS
        MDR = TOS = MDR OR H; wr; goto main
```

L'indirizzo che viene assegnato all'istruzione ijvm IOR nel control store è 0xB6, che deve essere uguale alla sua codifica in linguaggio macchina prodotta dall'assemblatore ijvm, in modo tale che quando la microistruzione main esegue il fetch del byte 0xB6, l'MPC è settato a 0xB6 e il microcodice relativo all'istruzione IOR viene eseguito.

Nel dettaglio le prime due righe di codice sono perfettamente analoghe a quelle dell'operazione IADD, per questo non vengono approfondite. Nell'ultimo ciclo viene effettuata la or bit a bit tra MDR (che ora contiene il secondo operando) e H (che contiene il primo operando), e il risultato viene salvato sia in MDR (per permettere la scrittura in memoria) sia in TOS (per mantenere la veridicità dell'invariante).

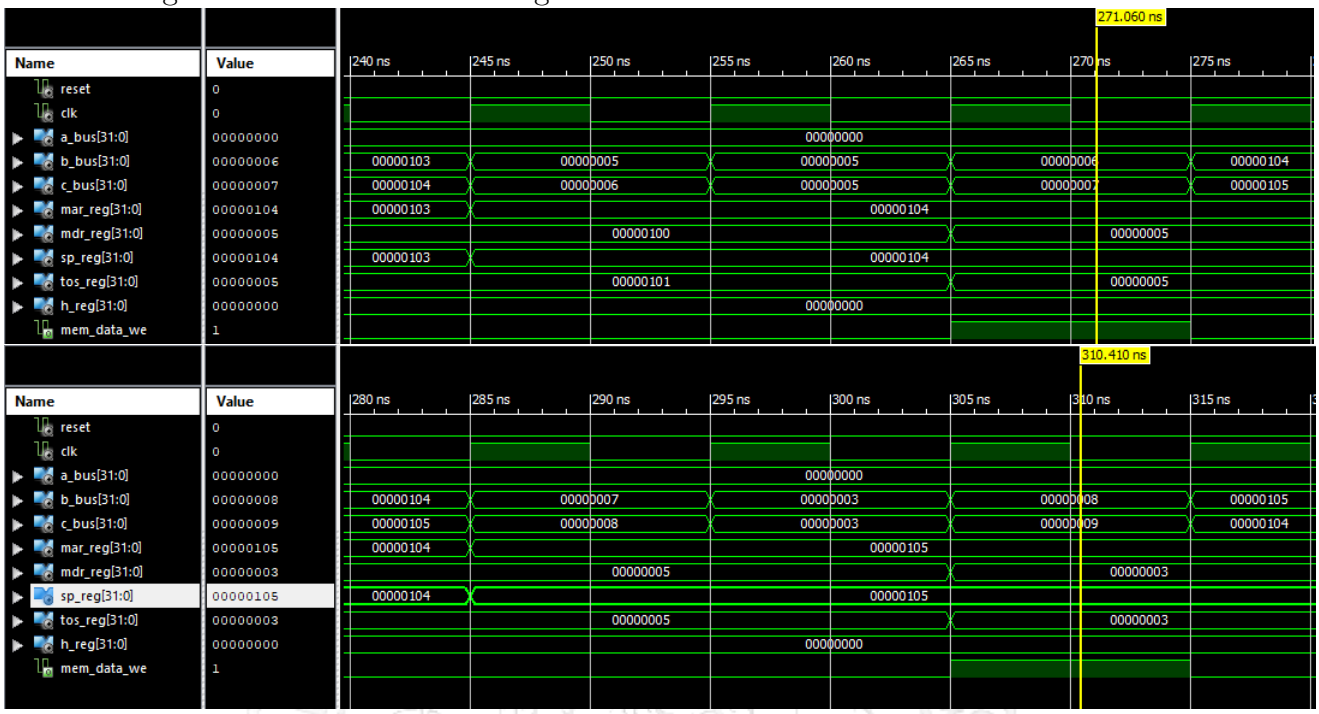
Per testare l'istruzione sul processore è necessario scrivere il codice in linguaggio assembly ajvm aggiungendo anche il push dei due operandi:

```
  .main
  BIPUSH 0x5
 BIPUSH 0x3
IOR
HALT
  .endmethod
```

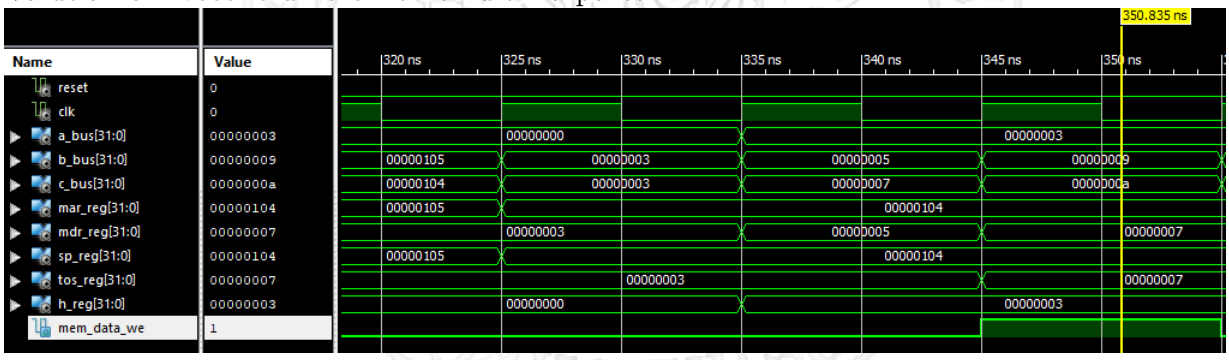
Dopo aver scritto questo codice nel file program.ajvm, è necessario lanciare il comando "make create_ram" che va a modificare il contenuto della ram nel seguente modo:

```
 -- RAM content
signal mem : dp_ar_ram_type := (
--BEGIN_WORDS_ENTRY
128 => "00000000000000000000000000000000",
0 => "00000000000000000000000100000000",
1 => "00000011000100000000010100010000",
2 => "00000000000000001010011110110110",
others => (others => '0')
--END_WORDS_ENTRY
    );
```

Per la simulazione le due schermate che seguono ripetono le stesse operazioni della IADD, per questo non vengono commentate nel dettaglio:




Valutiamo invece le differenze nell'ultima parte:




L'unica vera differenza sta nell'operazione che viene fatta dall'ALU, che non è più una somma (F0=1 F1=1, non presenti nella simulazione) ma una or (F0=0 F1=1).

I registri sui quali vengono fatte lettura e scrittura del dato sono gli stessi del caso della IADD.

- IADD con codice operativo modificato

Vediamo ora come procedere alla modifica di un codice operativo, per esempio quello della IADD gia studiata in precedenza:

```
iadd = 0x65:
        MAR = SP = SP - 1; rd
        H = TOS
        MDR = TOS = MDR + H; wr; goto main
```

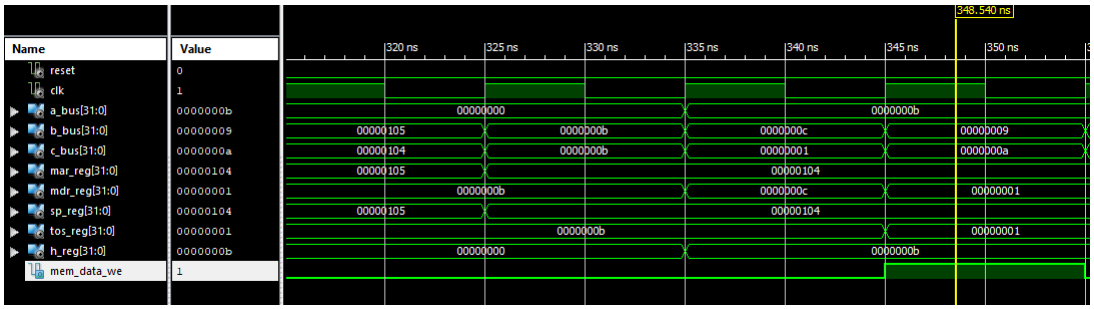
Si è deciso di andare a modificare l'ultimo step della somma, trasformandola in una sottrazione:

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

Questa modifica, dopo aver lanciato il comando make create_control_store, genera le seguenti modifiche nel codice della control store:

```
101 => "00101110100000110110000001001010010O",
102 => "00101111000000001010010000000000000111",
103 => "00000011000000011111100100001010000000",
```

Vediamo le modifiche effettuate anche in simulazione:




Al passo finale, il risultato dell'alu che viene salvato nel MDR e nel TOS non è più C+B=17, ma C-B=1.

## 12.4 Sintesi su board FPGA

Per la sintesi si è tentato di mostrare sul display il valore che viene scritto in memoria dopo un'istruzione, nello specifico nel codice si è considerata la IADD.

Per il clock si è utilizzato il clock della scheda, poi filtrato per permettere lo svolgimento di tutte le operazioni necessarie in ogni ciclo.

```
## Clock signal
NET "clock"    LOC = "E3"   | IOSTANDARD = "LVCMOS33";
```

Si è utilizzato poi un bottone di reset, per far ripartire dall'entry point (istruzione main) il processore.

```
## Buttons
NET "reset"           LOC=M17 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_D15_14
```

L'uscita è mostrata sul display utilizzando tutti gli anodi e tutti i catodi.

```
## 7 segment display
NET "cathodes<0>"             LOC=T10 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_A00_D16_14
NET "cathodes<1>"             LOC=R10 | IOSTANDARD=LVCMOS33; #IO_25_14
NET "cathodes<2>"             LOC=K16 | IOSTANDARD=LVCMOS33; #IO_25_15
NET "cathodes<3>"             LOC=K13 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A26_15
NET "cathodes<4>"             LOC=P15 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_14
NET "cathodes<5>"             LOC=T11 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A10_D26_14
NET "cathodes<6>"             LOC=L18 | IOSTANDARD=LVCMOS33; #IO_L4P_T0_D04_14
NET "cathodes<7>"             LOC=H15 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_A21_VREF_15

NET "anodes<0>"          LOC=J17 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_FOE_B_15
NET "anodes<1>"          LOC=J18 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_FWE_B_15
NET "anodes<2>"          LOC=T9 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_A01_D17_14
NET "anodes<3>"          LOC=J14 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A22_15
NET "anodes<4>"          LOC=P14 | IOSTANDARD=LVCMOS33; #IO_L8N_T1_D12_14
NET "anodes<5>"          LOC=T14 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_14
NET "anodes<6>"          LOC=K2 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_35
NET "anodes<7>"          LOC=U13 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_A02_D18_14
```

# Capitolo 13

# Esercizio 13

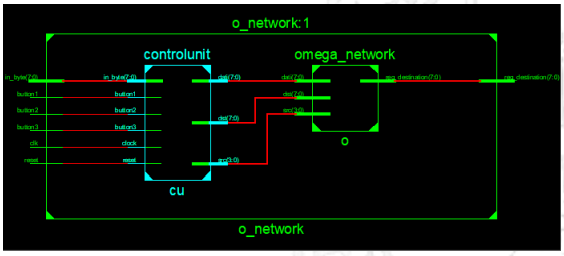
## 13.1 Traccia

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, il sistema può operare perdendo uno dei messaggi in conflitto (sistema con perdita)

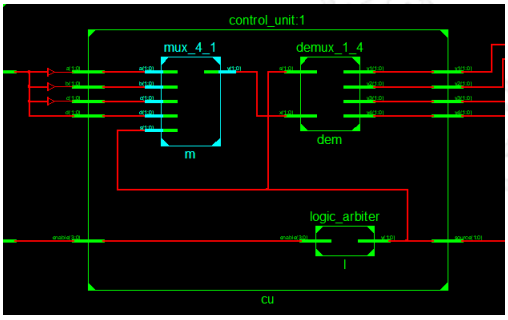
## 13.2 Soluzione

### 13.2.1 Schematici

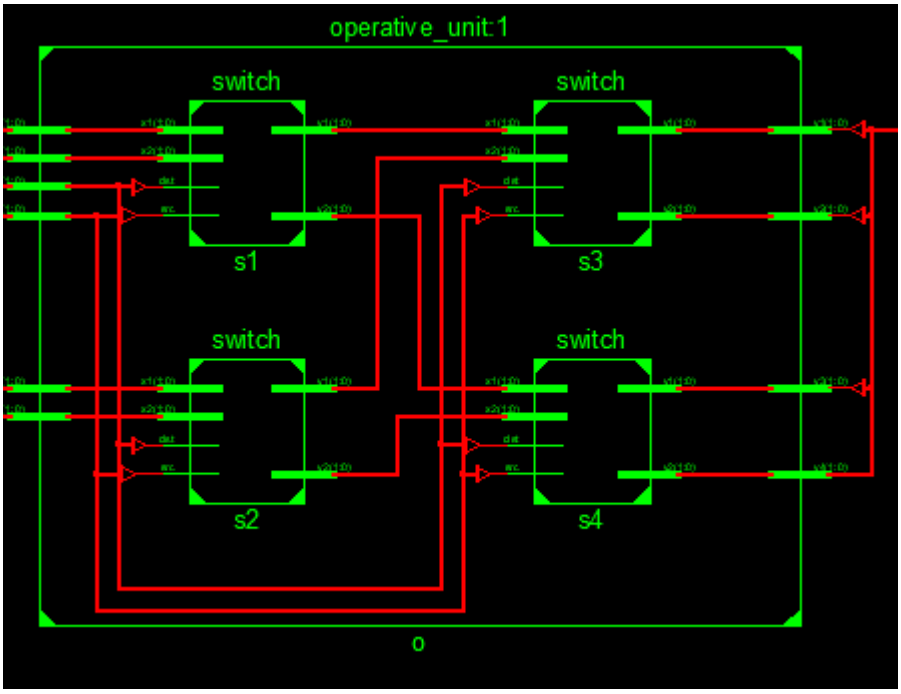
- Schematico generale




- Schematico control_unit

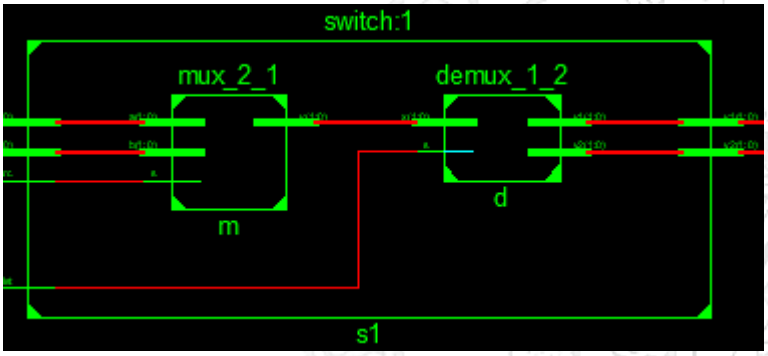


- Schematico operative_unit




Nella operative_unit ho 4 switch presenti e che prendono in ingresso ciascuno due linee dati di 2 bit ciascuna. In ingresso alla parte operativa ho due bit SRC del processore che vuole parlare e due bit DST del processore che deve ricevere. La parte operativa deve andare a mettere l'ingresso in uscita, se il processore 00 vuole parlare con 00 seguo il percorso dallo switch 1 a quello 3, se vuole parlare con processore 11 passa dallo switch 1 a quello 4.

- Schematico switch




Lo switch è fatto da un MUX e un DEMUX. Il mux seleziona tramite un bit del processore che vuole parlare (SRC) una delle due linee che viene passata al demux, che agendo sul bit di destinazione pone in uscita una delle due linee a seconda del valore del bit DST.

## 13.2.2 Codice

### 13.2.2.1 Omega_network

```
1
2 library IEEE;
```

```
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity o_network is
6      Port ( button1 : in  STD_LOGIC;
7             button2 : in  STD_LOGIC;
8             button3 : in  STD_LOGIC;
9           reset : in STD_LOGIC;
10          clk : in STD_LOGIC;
11             in_byte : in  STD_LOGIC_VECTOR (7 downto 0);
12             reg_destination : out  STD_LOGIC_VECTOR (7 downto 0));
13 end o_network;
14
15 architecture Structural of o_network is
16
17 component controlunit
18     Port ( button1 : in  STD_LOGIC;
19          button2 : in  STD_LOGIC;
20          button3 : in  STD_LOGIC;
21          reset : in  STD_LOGIC;
22          clock : in STD_LOGIC;
23             src : out  STD_LOGIC_VECTOR (3 downto 0);
24             dst : out  STD_LOGIC_VECTOR (7 downto 0);
25             dati : out  STD_LOGIC_VECTOR (7 downto 0);
26             in_byte : in  STD_LOGIC_VECTOR (7 downto 0));
27 end component;
28
29 component omega_network
30     Port ( src : in STD_LOGIC_VECTOR (3 downto 0);
31          dst : in STD_LOGIC_VECTOR (7 downto 0);
32          dati : in STD_LOGIC_VECTOR (7 downto 0);
33             --reg_source : out  STD_LOGIC_VECTOR (7 downto 0);
34             reg_destination : out  STD_LOGIC_VECTOR (7 downto 0));
35 end component;
36
37 signal s_temp : STD_LOGIC_VECTOR (3 downto 0);
38 signal d_temp : STD_LOGIC_VECTOR (7 downto 0);
39 signal dati_temp : STD_LOGIC_VECTOR (7 downto 0);
40
41 begin
42
43 cu : controlunit
44  Port Map( button1 => button1,
45          button2 => button2,
46          button3 => button3,
47          reset => reset,
48          clock => clk,
49             src => s_temp,
50             dst => d_temp,
51             dati => dati_temp,
```

```
52            in_byte => in_byte);
53
54 o : omega_network
55  Port Map( src => s_temp,
56         dst => d_temp,
57         dati => dati_temp,
58         reg_destination => reg_destination);
59
60
61 end Structural;
```

Codice Componente 13.1: Definizione del componente o$_n$*etwork*

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity controlunit is
6      Port ( button1 : in  STD_LOGIC;
7           button2 : in  STD_LOGIC;
8           button3 : in  STD_LOGIC;
9           reset : in  STD_LOGIC;
10          clock : in STD_LOGIC;
11             src : out  STD_LOGIC_VECTOR (3 downto 0);
12             dst : out  STD_LOGIC_VECTOR (7 downto 0);
13             dati : out  STD_LOGIC_VECTOR (7 downto 0);
14             in_byte : in  STD_LOGIC_VECTOR (7 downto 0));
15 end controlunit;
16
17 architecture Behavioral of controlunit is
18
19 signal reg_temp1 : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
20 signal reg_temp2 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
21 signal reg_temp3 : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
22
23 begin
24
25 main : process(reset, clock)
26 begin
27
28 if (clock'event and clock = '1') then
29 if reset = '1' then
30   reg_temp1 <= (others => '0');
31   reg_temp2 <= (others => '0');
32   reg_temp3 <= (others => '0');
33 end if;
34 end if;
35
36 if (clock'event and clock = '1') then
```

```
37 if button1 = '1' then
38   reg_temp1 <= in_byte(3 downto 0);
39 end if;
40
41 if button2 = '1' then
42   reg_temp2 <= in_byte;
43 end if;
44
45 if button3 = '1' then
46   reg_temp3 <= in_byte;
47 end if;
48
49 end if;
50
51 src <= reg_temp1;
52 dst <= reg_temp2;
53 dati <= reg_temp3;
54
55 end process;
56
57 end Behavioral;
```

Codice Componente 13.2: Definizione del componente controlunit

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity control_unit is
8      Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
9             b : in  STD_LOGIC_VECTOR (1 downto 0);
10            c : in  STD_LOGIC_VECTOR (1 downto 0);
11            d : in  STD_LOGIC_VECTOR (1 downto 0);
12            enable : in  STD_LOGIC_VECTOR (3 downto 0);
13         source : out STD_LOGIC_VECTOR (1 downto 0);
14            y1 : out  STD_LOGIC_VECTOR (1 downto 0);
15            y2 : out  STD_LOGIC_VECTOR (1 downto 0);
16            y3 : out  STD_LOGIC_VECTOR (1 downto 0);
17            y4 : out  STD_LOGIC_VECTOR (1 downto 0));
18 end control_unit;
19
20 architecture structural of control_unit is
21
22
23 signal temp : STD_LOGIC_VECTOR(1 downto 0);
24 signal en : STD_LOGIC_VECTOR (1 downto 0);
25
```

```
26 component mux_4_1
27 Port ( a : in  STD_LOGIC_VECTOR(1 downto 0);
28         b : in STD_LOGIC_VECTOR(1 downto 0);
29         c : in  STD_LOGIC_VECTOR(1 downto 0);
30         d : in STD_LOGIC_VECTOR(1 downto 0);
31            s : in  STD_LOGIC_VECTOR(1 downto 0);
32            y : out  STD_LOGIC_VECTOR(1 downto 0));
33 end component;
34
35 component demux_1_4
36  Port ( x : in  STD_LOGIC_VECTOR (1 downto 0);
37            y1 : out  STD_LOGIC_VECTOR (1 downto 0);
38         y2 : out  STD_LOGIC_VECTOR (1 downto 0);
39         y3 : out  STD_LOGIC_VECTOR (1 downto 0);
40         y4 : out  STD_LOGIC_VECTOR (1 downto 0);
41            s : in  STD_LOGIC_VECTOR (1 downto 0));
42 end component;
43
44 component logic_arbiter
45 Port ( enable : in  STD_LOGIC_VECTOR (3 downto 0);
46            y : out  STD_LOGIC_VECTOR (1 downto 0));
47 end component;
48
49 begin
50
51 m : mux_4_1
52 port map (
53 a => a,
54 b => b,
55 c => c,
56 d => d,
57 s => en,
58 y => temp
59 );
60
61 dem: demux_1_4
62 port map (
63 x => temp,
64 y1 => y1,
65 y2 => y2,
66 y3 => y3,
67 y4 => y4,
68 s => en
69 );
70
71 l: logic_arbiter
72 port map (
73 enable => enable,
74 y => en
```

```
75 );
76
77 source <= en;
78
79
80 end structural;
```

Codice Componente 13.3: Definizione del componente control$_unit$

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

Codice Componente 13.4: Definizione del componente decoder$_{24}$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity demux_1_2 is
5      Port ( x : in  STD_LOGIC_VECTOR ( 1 downto 0);
6             y1 : out  STD_LOGIC_VECTOR (1 downto 0);
7           y2 : out  STD_LOGIC_VECTOR (1 downto 0);
8             s : in  STD_LOGIC);
9  end demux_1_2;
10
11 architecture Behavioral of demux_1_2 is
12
13 begin
14
15 process (x,s)
16 begin
17
18 if s = '0' then y1 <= x;
19 else y2 <= x;
20 end if;
21 end process;
22
23
24 end Behavioral;
```

Codice Componente 13.5: Definizione del componente demux$_{12}$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity demux_1_4 is
6      Port ( x : in  STD_LOGIC_VECTOR (1 downto 0);
7             y1 : out  STD_LOGIC_VECTOR (1 downto 0);
8           y2 : out  STD_LOGIC_VECTOR (1 downto 0);
9           y3 : out  STD_LOGIC_VECTOR (1 downto 0);
```

```
10          y4 : out  STD_LOGIC_VECTOR (1 downto 0);
11             s : in  STD_LOGIC_VECTOR (1 downto 0));
12 end demux_1_4;
13
14 architecture Behavioral of demux_1_4 is
15
16 begin
17
18 process (x,s)
19 begin
20
21 if s = "00" then y1 <= x;
22 elsif s = "01" then y2 <= x;
23 elsif s = "10" then y3 <= x;
24 else y4 <= x;
25 end if;
26 end process;
27
28
29 end Behavioral;
```

Codice Componente 13.6: Definizione del componente $\text{demux}_{14}$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity logic_arbiter is
6      Port ( enable : in  STD_LOGIC_VECTOR (3 downto 0);
7             y : out  STD_LOGIC_VECTOR (1 downto 0));
8  end logic_arbiter;
9
10 architecture Behavioral of logic_arbiter is
11
12 begin
13
14 process (enable)
15 begin
16 -- il valore delle uscite dipende da quale dei 4 bit dell'ingresso enable è
       alto
17 if enable(0) = '1' then y <= "00";
18 elsif enable(1) = '1' then y <= "01";
19 elsif enable(2) = '1' then y <= "10";
20 elsif enable(3) = '1' then y <= "11";
21 else y <= "UU";
22 end if;
23 end process;
24
25
```

```
end Behavioral;
```

Codice Componente 13.7: Definizione del componente $\text{logic}_arbiter$

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


entity mux_2_1 is
    Port ( a : in  STD_LOGIC_VECTOR(1 downto 0);
        b : in STD_LOGIC_VECTOR(1 downto 0);
           s : in  STD_LOGIC;
           y : out  STD_LOGIC_VECTOR(1 downto 0));
end mux_2_1;

architecture dataflow of mux_2_1 is

begin

y   <=  a when s='0' else
      b when s='1'  else

        "--";
end dataflow;
```

Codice Componente 13.8: Definizione del componente $\text{mux}_{21}$

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


entity mux_4_1 is
    Port (  a : in  STD_LOGIC_VECTOR(1 downto 0);
        b : in  STD_LOGIC_VECTOR(1 downto 0);
        c : in  STD_LOGIC_VECTOR(1 downto 0);
        d : in  STD_LOGIC_VECTOR(1 downto 0);
           s : in  STD_LOGIC_VECTOR(1 downto 0);
           y : out  STD_LOGIC_VECTOR(1 downto 0));
end mux_4_1;

architecture dataflow of mux_4_1 is

begin

y   <=  a when s="00" else
      b when s="01" else
      c when s="10" else
      d when s="11" else
        "--";

```

```
24 end dataflow;
```

Codice Componente 13.9: Definizione del componente $\text{mux}_{41}$

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity omega_network is
6      Port ( src : in STD_LOGIC_VECTOR (3 downto 0);
7           dst : in STD_LOGIC_VECTOR (7 downto 0);
8           dati : in STD_LOGIC_VECTOR (7 downto 0);
9              reg_destination : out  STD_LOGIC_VECTOR (7 downto 0));
10 end omega_network;
11
12 architecture structural of omega_network is
13
14 -- collego tra loro control_unit e operative_unit utilizzando anche il
       decoder per ottenere il sistema complessivo
15
16 component control_unit
17  Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
18             b : in  STD_LOGIC_VECTOR (1 downto 0);
19             c : in  STD_LOGIC_VECTOR (1 downto 0);
20             d : in  STD_LOGIC_VECTOR (1 downto 0);
21             enable : in  STD_LOGIC_VECTOR (3 downto 0);
22         source : out STD_LOGIC_VECTOR (1 downto 0);
23             y1 : out  STD_LOGIC_VECTOR (1 downto 0);
24             y2 : out  STD_LOGIC_VECTOR (1 downto 0);
25             y3 : out  STD_LOGIC_VECTOR (1 downto 0);
26             y4 : out  STD_LOGIC_VECTOR (1 downto 0));
27 end component;
28
29 component operative_unit
30 Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
31             b : in  STD_LOGIC_VECTOR (1 downto 0);
32             c : in  STD_LOGIC_VECTOR (1 downto 0);
33             d : in  STD_LOGIC_VECTOR (1 downto 0);
34             src : in STD_LOGIC_VECTOR (1 downto 0);
35          dst : in STD_LOGIC_VECTOR (1 downto 0);
36             y1 : out  STD_LOGIC_VECTOR (1 downto 0);
37             y2 : out  STD_LOGIC_VECTOR (1 downto 0);
38             y3 : out  STD_LOGIC_VECTOR (1 downto 0);
39             y4 : out  STD_LOGIC_VECTOR (1 downto 0));
40 end component;
41
42 component mux4_1
43 Port ( x0 : in  STD_LOGIC_VECTOR (1 downto 0);
44      x1 : in  STD_LOGIC_VECTOR (1 downto 0);
```

```
45      x2 : in  STD_LOGIC_VECTOR (1 downto 0);
46      x3 : in  STD_LOGIC_VECTOR (1 downto 0);
47      s : in  STD_LOGIC_VECTOR (1 downto 0);
48        y : out  STD_LOGIC_VECTOR (1 downto 0));
49  end component;
50
51  signal en : std_logic_vector (3 downto 0);
52  signal destination : std_logic_vector (1 downto 0);
53  signal temp1 : std_logic_vector (1 downto 0);
54  signal temp2 : std_logic_vector (1 downto 0);
55  signal temp3 : std_logic_vector (1 downto 0);
56  signal temp4 : std_logic_vector (1 downto 0);
57
58
59  begin
60
61  c : control_unit
62  port map (
63  a => dati(1 downto 0),
64  b => dati(3 downto 2),
65  c => dati(5 downto 4),
66  d => dati(7 downto 6),
67  enable => src,
68  source => en,
69  y1 => temp1,
70  y2 => temp2,
71  y3 => temp3,
72  y4 => temp4
73  );
74
75
76  o : operative_unit
77  port map (
78  a => temp1,
79  b => temp2,
80  c => temp3,
81  d => temp4,
82  src => en,
83  dst => destination,
84  y1 => reg_destination(1 downto 0),
85  y2 => reg_destination(3 downto 2),
86  y3 => reg_destination(5 downto 4),
87  y4 => reg_destination(7 downto 6)
88  );
89
90
91  mux : mux4_1
92  port map (
93  x0 => dst(1 downto 0),
```

```
94  x1 => dst(3 downto 2),
95  x2 => dst(5 downto 4),
96  x3 => dst(7 downto 6),
97  s => en,
98  y => destination
99  );
100
101 end structural;
```

Codice Componente 13.10: Definizione del componente omega$_n$*etwork*

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   -- unità operativa composta da 4 switch
5
6   entity operative_unit is
7    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
8              b : in  STD_LOGIC_VECTOR (1 downto 0);
9              c : in  STD_LOGIC_VECTOR (1 downto 0);
10             d : in  STD_LOGIC_VECTOR (1 downto 0);
11             src : in STD_LOGIC_VECTOR (1 downto 0);
12          dst : in STD_LOGIC_VECTOR (1 downto 0);
13             y1 : out  STD_LOGIC_VECTOR (1 downto 0);
14             y2 : out  STD_LOGIC_VECTOR (1 downto 0);
15             y3 : out  STD_LOGIC_VECTOR (1 downto 0);
16             y4 : out  STD_LOGIC_VECTOR (1 downto 0));
17  end operative_unit;
18
19  architecture structural of operative_unit is
20
21  component switch
22   Port ( x1 : in  STD_LOGIC_VECTOR (1 downto 0);
23             x2 : in  STD_LOGIC_VECTOR (1 downto 0);
24             src : in  STD_LOGIC;
25             dst : in  STD_LOGIC;
26             y1 : out  STD_LOGIC_VECTOR (1 downto 0);
27             y2 : out  STD_LOGIC_VECTOR (1 downto 0));
28  end component;
29
30  signal temp1 : std_logic_vector (1 downto 0);
31  signal temp2 : std_logic_vector (1 downto 0);
32  signal temp3 : std_logic_vector (1 downto 0);
33  signal temp4 : std_logic_vector (1 downto 0);
34
35  begin
36
37  -- i bit della sorgente sono valutati dal meno significativo al più
       significativo, quindi i primi due switch hanno in ingresso src(0) e gli
```

```
    ultimi 2 hanno in ingresso src(1). Ragionamento opposto nel caso della
    destinazione.
38
39  s1 : switch
40  port map (
41  x1 => a,
42  x2 => b,
43  src => src(0),
44  dst => dst(1),
45  y1 => temp1,
46  y2 => temp2
47  );
48
49  s2 : switch
50  port map (
51  x1 => c,
52  x2 => d,
53  src => src(0),
54  dst => dst(1),
55  y1 => temp3,
56  y2 => temp4
57  );
58
59  s3 : switch
60  port map (
61  x1 => temp1,
62  x2 => temp3,
63  src => src(1),
64  dst => dst(0),
65  y1 => y1,
66  y2 => y2
67  );
68
69  s4 : switch
70  port map (
71  x1 => temp2,
72  x2 => temp4,
73  src => src(1),
74  dst => dst(0),
75  y1 => y3,
76  y2 => y4
77  );
78
79  end structural;
```

Codice Componente 13.11: Definizione del componente operative$_unit$

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

```
3
4
5  entity switch is
6      Port ( x1 : in  STD_LOGIC_VECTOR (1 downto 0);
7             x2 : in  STD_LOGIC_VECTOR (1 downto 0);
8             src : in  STD_LOGIC;
9             dst : in  STD_LOGIC;
10            y1 : out  STD_LOGIC_VECTOR (1 downto 0);
11            y2 : out  STD_LOGIC_VECTOR (1 downto 0));
12 end switch;
13
14 architecture structural of switch is
15
16 component mux_2_1
17 Port ( a : in  STD_LOGIC_VECTOR(1 downto 0);
18         b : in STD_LOGIC_VECTOR(1 downto 0);
19            s : in  STD_LOGIC;
20            y : out  STD_LOGIC_VECTOR(1 downto 0));
21 end component;
22
23 component demux_1_2
24  Port ( x : in  STD_LOGIC_VECTOR ( 1 downto 0);
25            y1 : out  STD_LOGIC_VECTOR (1 downto 0);
26         y2 : out  STD_LOGIC_VECTOR (1 downto 0);
27            s : in  STD_LOGIC);
28 end component;
29
30
31 signal temp : std_logic_vector (1 downto 0);
32
33 begin
34
35 -- switch banalmente costituito da un demux a valle di un mux, con segnali
       di selezione src per il mux e dst per il demux
36
37 m : mux_2_1
38 port map(
39 a => x1,
40 b => x2,
41 s => src,
42 y => temp
43 );
44
45 d: demux_1_2
46 port map (
47 x => temp,
48 y1 => y1,
49 y2 => y2,
50 s => dst
```

```
51 );
52
53 end structural;
```

Codice Componente 13.12: Definizione del componente switch

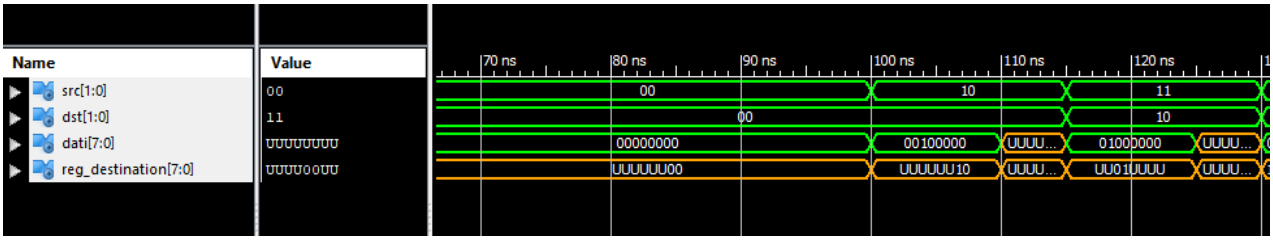
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4
5  ENTITY omega_network_tb IS
6  END omega_network_tb;
7
8  ARCHITECTURE behavior OF omega_network_tb IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11
12     COMPONENT omega_network
13     PORT(
14          src : IN  std_logic_vector(1 downto 0);
15          dst : IN  std_logic_vector(1 downto 0);
16          dati : IN  std_logic_vector(7 downto 0);
17          reg_destination : OUT  std_logic_vector(7 downto 0)
18         );
19     END COMPONENT;
20
21
22    --Inputs
23    signal src : std_logic_vector(1 downto 0) := (others => '0');
24    signal dst : std_logic_vector(1 downto 0) := (others => '0');
25    signal dati : std_logic_vector(7 downto 0) := (others => '0');
26
27   --Outputs
28    signal reg_destination : std_logic_vector(7 downto 0);
29    -- No clocks detected in port list. Replace <clock> below with
30    -- appropriate port name
31
32
33 BEGIN
34
35   -- Instantiate the Unit Under Test (UUT)
36    uut: omega_network PORT MAP (
37           src => src,
38           dst => dst,
39           dati => dati,
40           reg_destination => reg_destination
41         );
42
43
```

```
44    -- Stimulus process
45    stim_proc: process
46    begin
47       -- hold reset state for 100 ns.
48       wait for 100 ns;
49
50
51       -- insert stimulus here
52
53     dati <= "00100000";
54
55     src <= "10";
56     dst <= "00";
57
58     wait for 10 ns;
59
60     dati <= "UUUUUUUU";
61
62     wait for 5 ns;
63     dati <= "01000000";
64     src <= "11";
65     dst <= "10";
66
67     wait for 10 ns;
68
69     dati <= "UUUUUUUU";
70
71     wait for 5 ns;
72     dati <= "00000011";
73     src <= "00";
74     dst <= "11";
75
76     wait for 5 ns;
77
78     dati <= "UUUUUUUU";
79
80  wait for 10 ns;
81 dati <= "00110000";
82   src <= "10";
83     dst <= "00";
84
85     wait for 10 ns;
86
87     dati <= "UUUUUUUU";
88
89     wait for 5 ns;
90     dati <= "01000000";
91     src <= "11";
92     dst <= "10";
```

```
93
94          wait;
95      end process;
96
97  END;
```

Codice Componente 13.13: Definizione del componente omega$_n etwork_t b$

## 13.3 Simulazione




La simulazione mostra il funzionamento dello switch: tra 100 e 115 ns ho che la sorgente è 10, quindi essendo il vettore dati definito come 7 downto 0 il valore 10 corrisponde alla seconda coppia di bit (ossia 10), che viene mostrata in uscita nell'ultima coppia di bit essendo l'ingresso dst = "11".

Nel secondo esempio, tra 115 e 130 ns, ho invece che src="11" e dst="10", quindi la prima coppia di bit dati (in questo caso 01) viene mostrata in uscita come seconda coppia del reg_destination.

## 13.4 Sintesi su board FPGA

Per interfacciare il design realizzato con l'I/O di board si è adottata la seguente soluzione:

Per il clock si è utilizzato il clock della scheda.

```
## Clock signal
NET "clk"   LOC = "E3" | IOSTANDARD = "LVCMOS33";
```

Poichè sono stati utilizzati 12 switch per gli ingressi, è stato necessario aggiungere il seguente constraint per permettere la sintesi e rimuovere alcuni errori in fase di mapping.

```
NET "src<0>" CLOCK_DEDICATED_ROUTE = FALSE;

NET "src<1>" CLOCK_DEDICATED_ROUTE = FALSE;

NET "dst<0>" CLOCK_DEDICATED_ROUTE = FALSE;

NET "dst<1>" CLOCK_DEDICATED_ROUTE = FALSE;
```

Per gli ingressi sono stati utilizzati i 12 switch della scheda più a destra: i primi 8 per il dato, poi da destra verso sinistra 2 per la destinazione e 2 per la sorgente.

```
## Switches
NET "dati<0>"          LOC=J15 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_RS0_15
NET "dati<1>"          LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "dati<2>"          LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
NET "dati<3>"          LOC=R15 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_14
NET "dati<4>"          LOC=R17 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_14
NET "dati<5>"          LOC=T18 | IOSTANDARD=LVCMOS33; #IO_L7N_T1_D10_14
NET "dati<6>"          LOC=U18 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A13_D29_14
NET "dati<7>"          LOC=R13 | IOSTANDARD=LVCMOS33; #IO_L5N_T0_D07_14
NET "dst<0>"           LOC=T8 | IOSTANDARD=LVCMOS18; #IO_L24N_T3_34
NET "dst<1>"           LOC=U8 | IOSTANDARD=LVCMOS18; #IO_25_34
NET "src<0>"          LOC=R16 | IOSTANDARD=LVCMOS33; #IO_L15P_T2_DQS_RDWR_B_14
NET "src<1>"          LOC=T13 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_A03_D19_14
```

L'uscita è mostrata sugli 8 led più a destra della board.

```
## LEDs
NET "reg_destination<0>"          LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
NET "reg_destination<1>"          LOC=K15 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_RS1_15
NET "reg_destination<2>"          LOC=J13 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A25_15
NET "reg_destination<3>"          LOC=N14 | IOSTANDARD=LVCMOS33; #IO_L8P_T1_D11_14
NET "reg_destination<4>"          LOC=R18 | IOSTANDARD=LVCMOS33; #IO_L7P_T1_D09_14
NET "reg_destination<5>"          LOC=V17 | IOSTANDARD=LVCMOS33; #IO_L18N_T2_A11_D27_14
NET "reg_destination<6>"          LOC=U17 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A14_D30_14
NET "reg_destination<7>"          LOC=U16 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A12_D28_14
```