

# UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

---

*Corso di Laurea Magistrale in Ingegneria Informatica*

---



## ELABORATO DI ARCHITETTURA DEI SISTEMI DIGITALI

*Prof.ssa Alessandra De Benedictis*

a.a. 2024-25

Studenti:

ESPOSITO FABIO M63001806

FONTANELLA GIANLUCA M63001818

GRANDIOSO NICOLA M63001814

IOVINE GIOVANNA M63001821

## Sommario

Capitolo 1: reti combinatorie elementari .....	5
Esercizio 1: Rete di interconnessione .....	5
Esercizio 1.1 .....	5
Esercizio 1.2 .....	5
Progetto e architettura .....	5
Implementazione.....	6
Simulazione.....	9
Sintesi su board di sviluppo .....	10
Esercizio 1.3 .....	10
Esercizio 2: ROM+M.....	13
Esercizio 2.1 .....	13
Progetto e architettura .....	13
Implementazione.....	13
Simulazione.....	15
Sintesi su board di sviluppo .....	16
Esercizio 2.2 .....	16
Capitolo 2: Reti sequenziali elementari.....	18
Esercizio 3: Riconoscitore di sequenze .....	18
Esercizio 3.1 .....	18
Progetto e architettura .....	18
Implementazione.....	19
Simulazione.....	21
Esercizio 3.2 .....	24
Progetto e architettura .....	24
Implementazione.....	24
Sintesi su board di sviluppo .....	28
Timing Analysis .....	29
Esercizio 4: Shift Register.....	31
Progetto e Architettura .....	31
Implementazione.....	32
Simulazione.....	34
Esercizio 5 : Cronometro.....	36
Esercizio 5.1 : .....	36
Progetto e Architettura .....	36
Implementazione.....	37

Simulazione.....	39
Esercizio 5.2 : .....	40
Progetto e Architettura .....	40
Implementazione.....	42
Sintesi su board di sviluppo .....	46
Esercizio 6 : Sistema di lettura-elaborazione-scrittura PO_PC.....	48
Esercizio 6.1 : .....	48
Progetto e Architettura .....	48
Implementazione.....	50
Simulazione.....	54
Sintesi su board di sviluppo .....	55
Capitolo 3: Macchine aritmetiche .....	58
Esercizio 7.1 .....	58
Progetto e Architettura .....	58
Implementazione.....	59
Simulazione.....	62
Esercizio 7.2 .....	64
Progetto e architettura .....	64
Implementazione.....	64
Sintesi su board di sviluppo .....	66
Capitolo 4: Comunicazione con handshaking.....	67
Esercizio 8.1:.....	67
Progetto e Architettura .....	67
Implementazione.....	69
Simulazione.....	74
Capitolo 5: Processore .....	75
Esercizio 9.a .....	75
BIPUSH 0xA .....	75
ISTORE a .....	77
Esercizio 9.b .....	79
IADD .....	79
SWAP.....	80
Capitolo 6: Interfaccia Seriale .....	82
Esercizio 10 .....	82
Progetto e Architettura .....	82
Implementazione.....	84

Simulazione.....	89
Capitolo 7: Switch Multistadio.....	90
Esercizio 11 .....	90
Progetto e architettura .....	90
Implementazione.....	91
Simulazione.....	94
Capitolo 8: Prova Esame 19/12/2024.....	96
Esercizio 12:.....	96
Progetto e Architettura.....	96
Implementazione.....	98
Simulazione.....	104
Appendice.....	105
ROM.....	105
Progetto e architettura .....	105
Implementazione.....	105
MEMORIA .....	106
Progetto e architettura .....	106
Implementazione.....	106
CONTATORE .....	107
Progetto e architettura .....	107
Implementazione.....	107

# Capitolo 1: reti combinatorie elementari

## Esercizio 1: Rete di interconnessione

### Esercizio 1.1

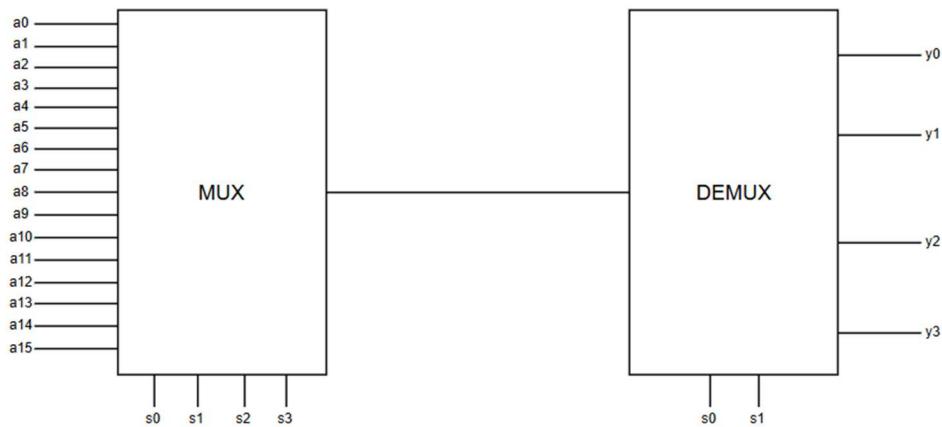
Progettare, implementare in VHDL e testare mediante simulazione un **multiplexer indirizzabile 16:1**, utilizzando un approccio di progettazione per composizione a partire da **multiplexer 4:1**.

### Esercizio 1.2

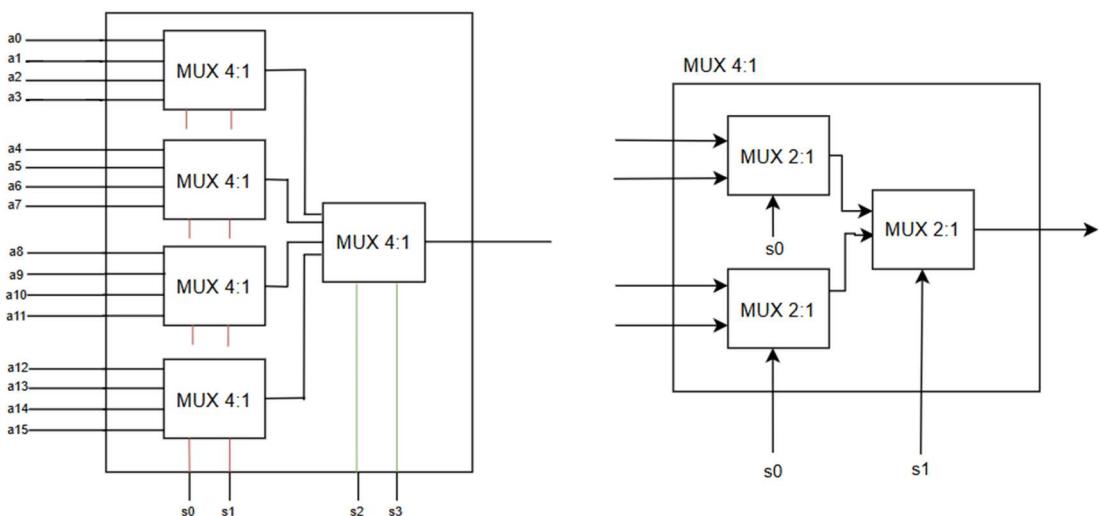
Utilizzando il componente sviluppato al punto precedente, progettare, implementare in VHDL e testare mediante simulazione una rete di interconnessione a 16 sorgenti e 4 destinazioni.

#### Progetto e architettura

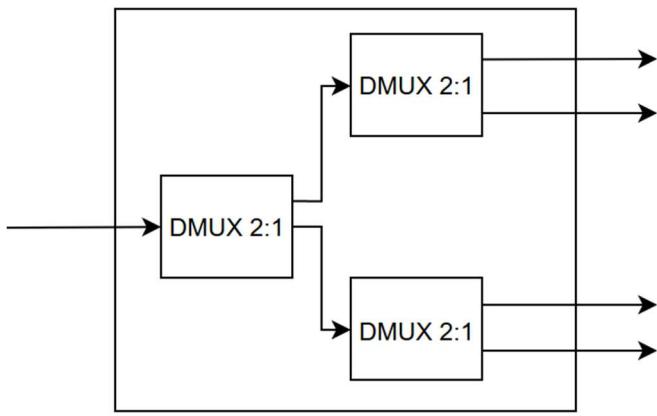
L'approccio di progetto utilizzato è quello strutturale.



La rete di interconnessione presenta 16 ingressi nel MUX, la selezione è codificata su 4 bit, l'uscita è convogliata in un DEMUX che presenta 4 possibili uscite e due ingressi di selezione. Sia il MUX che il DEMUX sono realizzati strutturalmente utilizzando i componenti MUX\_2\_1 (in appendice) e DEMUX\_1\_2 (in appendice).



Questo è il MUX 16:1 realizzato con MUX 4:1 a loro volta costruiti strutturalmente con MUX 2:1.



Si riporta anche la struttura del DEMUX 1:4.

*Implementazione*

### MUX 16:1 – STRUTTURALE (*mux\_16\_1.vhd*)

```

entity mux_16_1 is
  Port ( a0 : in STD_LOGIC;
         a1 : in STD_LOGIC;
         a2 : in STD_LOGIC;
         a3 : in STD_LOGIC;
         a4 : in STD_LOGIC;
         a5 : in STD_LOGIC;
         a6 : in STD_LOGIC;
         a7 : in STD_LOGIC;
         a8 : in STD_LOGIC;
         a9 : in STD_LOGIC;
         a10 : in STD_LOGIC;
         a11 : in STD_LOGIC;
         a12 : in STD_LOGIC;
         a13 : in STD_LOGIC;
         a14 : in STD_LOGIC;
         a15 : in STD_LOGIC;
         s0 : in STD_LOGIC;
         s1 : in STD_LOGIC;
         s2 : in STD_LOGIC;
         s3 : in STD_LOGIC;
         y : out STD_LOGIC);
end mux_16_1;

```

```

architecture structural of mux_16_1 is

signal u3 : std_logic := '0';
signal u2 : std_logic := '0';
signal u1 : std_logic := '0';
signal u0 : std_logic := '0';

component mux_4_1
  port(
    b0 : in STD_LOGIC;
    b1 : in STD_LOGIC;
    b2 : in STD_LOGIC;
    b3 : in STD_LOGIC;
    s0 : in STD_LOGIC;
    s1 : in STD_LOGIC;
    y0 : out STD_LOGIC
  );
end component;
begin

  mux_0: mux_4_1
    Port map(
      b0 => a0,
      b1 => a1,
      b2 => a2,
      b3 => a3,
      s0 => s0,
      s1 => s1,
      y0 => u3
    );
  end;

```

```

mux_1: mux_4_1
Port map(
  b0 => a4,
  b1 => a5,
  b2 => a6,
  b3 => a7,
  s0 => s0,
  s1 => s1,
  y0 => u2
);
mux_2: mux_4_1
Port map(
  b0 => a8,
  b1 => a9,
  b2 => a10,
  b3 => a11,
  s0 => s0,
  s1 => s1,
  y0 => u1
);
mux_3: mux_4_1
Port map(
  b0 => a12,
  b1 => a12,
  b2 => a14,
  b3 => a15,
  s0 => s0,
  s1 => s1,
  y0 => u0
);
mux_4: mux_4_1
Port map(
  b0 => u3,
  b1 => u2,
  b2 => u1,
  b3 => u0,
  s0 => s2,
  s1 => s3,
  y0 => y
);

```

## DEMUX 1:4 – STRUTTURALE (*demux\_1\_4.vhd*)

```

entity demux_1_4 is
  Port ( b0 : in STD_LOGIC;
         s1 : in STD_LOGIC;
         s0 : in STD_LOGIC;
         y0 : out STD_LOGIC;
         y1 : out STD_LOGIC;
         y2 : out STD_LOGIC;
         y3 : out STD_LOGIC);
end demux_1_4;

```

```

architecture Structural of demux_1_4 is

signal u0 : STD_LOGIC := '0';
signal u1 : STD_LOGIC := '0';

component DEMUX_1_2
Port ( a0 : in STD_LOGIC;
       s0 : in STD_LOGIC;
       y0 : out STD_LOGIC;
       y1 : out STD_LOGIC);
end component;

begin

demux_0: DEMUX_1_2
Port map(
    a0 => b0,
    s0 => s1,
    y0 => u0,
    y1 => u1
);

demux_1: DEMUX_1_2
Port map(
    a0 => u0,
    s0 => s0,
    y0 => y0,
    y1 => y1
);

end;

```

```

demux_2: DEMUX_1_2
Port map(
    a0 => u1,
    s0 => s0,
    y0 => y2,
    y1 => y3
);

end Structural;

```

## RETE DI INTERCONNESSIONE – STRUTTURALE (*rete\_di\_interconnessione.vhd*)

```

) entity rete_di_interconnessione is
  Port ( a0 : in STD_LOGIC;
         a1 : in STD_LOGIC;
         a2 : in STD_LOGIC;
         a3 : in STD_LOGIC;
         a4 : in STD_LOGIC;
         a5 : in STD_LOGIC;
         a6 : in STD_LOGIC;
         a7 : in STD_LOGIC;
         a8 : in STD_LOGIC;
         a9 : in STD_LOGIC;
         a10 : in STD_LOGIC;
         a11 : in STD_LOGIC;
         a12 : in STD_LOGIC;
         a13 : in STD_LOGIC;
         a14 : in STD_LOGIC;
         a15 : in STD_LOGIC;
         y0 : out STD_LOGIC;
         y1 : out STD_LOGIC;
         y2 : out STD_LOGIC;
         y3 : out STD_LOGIC;
         s0 : in STD_LOGIC;
         s1 : in STD_LOGIC;
         s2 : in STD_LOGIC;
         s3 : in STD_LOGIC;
         s4 : in STD_LOGIC;
         s5 : in STD_LOGIC);

) end rete_di_interconnessione;

```

```

    architecture Structural of rete_di_interconnessione is
    signal u0 : STD_LOGIC := '0';

    component demux_1_4
    Port ( b0 : in STD_LOGIC;
           s1 : in STD_LOGIC;
           s0 : in STD_LOGIC;
           y0 : out STD_LOGIC;
           y1 : out STD_LOGIC;
           y2 : out STD_LOGIC;
           y3 : out STD_LOGIC);
    end component;

    component mux_16_1
    Port ( a0 : in STD_LOGIC;
           a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           a3 : in STD_LOGIC;
           a4 : in STD_LOGIC;
           a5 : in STD_LOGIC;
           a6 : in STD_LOGIC;
           a7 : in STD_LOGIC;
           a8 : in STD_LOGIC;
           a9 : in STD_LOGIC;
           a10 : in STD_LOGIC;
           a11 : in STD_LOGIC;
           a12 : in STD_LOGIC;
           a13 : in STD_LOGIC;
           a14 : in STD_LOGIC;
           a15 : in STD_LOGIC;
           s0 : in STD_LOGIC;
           s1 : in STD_LOGIC;
           s2 : in STD_LOGIC;
           s3 : in STD_LOGIC;
           y : out STD_LOGIC);
    end component;

```

```

begin
    mux:mux_16_1
    Port map(
        a0 => a0,
        a1 => a1,
        a2 => a2,
        a3 => a3,
        a4 => a4,
        a5 => a5,
        a6 => a6,
        a7 => a7,
        a8 => a8,
        a9 => a9,
        a10 => a10,
        a11 => a11,
        a12 => a12,
        a13 => a13,
        a14 => a14,
        a15 => a15,
        s0 => s0,
        s1 => s1,
        s2 => s2,
        s3 => s3,
        y => u0
    );
    demux:demux_1_4
    Port map(
        b0 => u0,
        s0 => s4,
        s1 => s5,
        y0 => y0,
        y1 => y1,
        y2 => y2,
        y3 => y3
    );
end Structural;

```

### Simulazione

Per testare il sistema abbiamo posto dei valori di default in ogni ingresso e poi abbiamo selezionato mittente e destinatario tramite i bit di controllo per capire se i segnali venissero trasmessi nel modo giusto. Di seguito il process degli stimoli.

```

uut: rete_di_interconnessione port map ( a0  => input(0),
                                         a1  => input(1),
                                         a2  => input(2),
                                         a3  => input(3),
                                         a4  => input(4),
                                         a5  => input(5),
                                         a6  => input(6),
                                         a7  => input(7),
                                         a8  => input(8),
                                         a9  => input(9),
                                         a10 => input(10),
                                         a11 => input(11),
                                         a12 => input(12),
                                         a13 => input(13),
                                         a14 => input(14),
                                         a15 => input(15),
                                         y0  => output(0),
                                         y1  => output(1),
                                         y2  => output(2),
                                         y3  => output(3),
                                         s0  => control(0),
                                         s1  => control(1),
                                         s2  => control(2),
                                         s3  => control(3),
                                         s4  => control(4),
                                         s5  => control(5)
                                         );

```

```

stimulus: process
begin
    -- Put initialisation code here
    wait for 100 ns;

    input <= "0101111101011110";
    control <= "001100"; -- 0 PARLA CON 12
    wait for 10 ns;

    control <= "010000"; -- 1 PARLA CON 0
    wait for 10 ns;

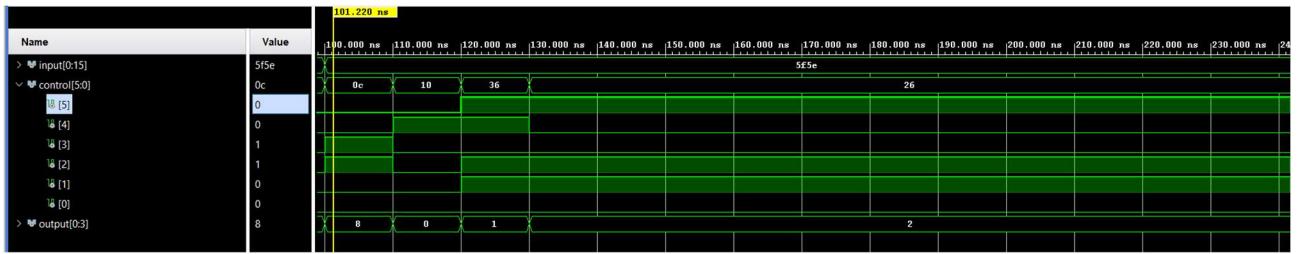
    control <= "110110"; -- 3 PARLA CON 6
    wait for 10 ns;

    control <= "100110"; -- 2 PARLA CON 1
    -- Put test bench stimulus code here
    wait;
end process;

```

Riportiamo, inoltre, i risultati principali della simulazione, possiamo osservare che ci sono 6 bit di controllo, i due più significativi per definire il ricevente, i quattro meno significativi per definire il trasmettente,

osservando in binario i valori dei numeri in esadecimale rappresentati nella simulazione si osserva che giungono i segnali giusti.



### Sintesi su board di sviluppo

#### Esercizio 1.3

Sintetizzare ed implementare su board il progetto della rete di interconnessione sviluppato al punto 1.2, utilizzando gli switch per fornire gli input di selezione e i led per visualizzare i 4 bit di uscita. Per quanto riguarda i 16 bit dato in input, essi devono essere immessi mediante switch, 8 bit alla volta, sviluppando un'apposita “rete di controllo” per l’acquisizione che utilizzi due bottoni della board per caricare rispettivamente la prima e la seconda metà del dato in ingresso.

Poiché sono necessari 16 bit da mandare in ingresso alla rete di interconnessione e 6 bit per la selezione di mittente e destinatario, avendo a disposizione solo 16 switch abbiamo supposto l’utilizzo di 2 bottoni, premendone uno si caricano i primi 8 bit di ingresso, mentre premendo l’altro si ottengono gli altri 8 bit, nel mentre 6 switch sono addetti alla selezione di mittente e destinatario.

#### CU – BEHAVIORAL (cu.vhd)

```

entity cu is
    Port ( clock : in STD_LOGIC;
            load_first_part : in STD_LOGIC;
            load_second_part : in STD_LOGIC;
            value16_out: out STD_LOGIC_VECTOR (0 to 15);
            value8_in: in STD_LOGIC_VECTOR(0 to 7));
end cu;

architecture Behavioral of cu is

signal reg_value : STD_LOGIC_VECTOR(0 to 15) := (others => '0');

begin

value16_out <= reg_value;

main: process(clock)
begin

if clock'event and clock = '1' then
    if load_first_part = '1' then
        reg_value(0 to 7) <= value8_in;
    elsif load_second_part = '1' then
        reg_value(8 to 15) <= value8_in;
    end if;
end if;
end process;
end Behavioral;
```

## RETE DI INTERCONNESSIONE ON BOARD – STRUCTURAL (*rete\_di\_interconnessione\_on\_board.vhd*)

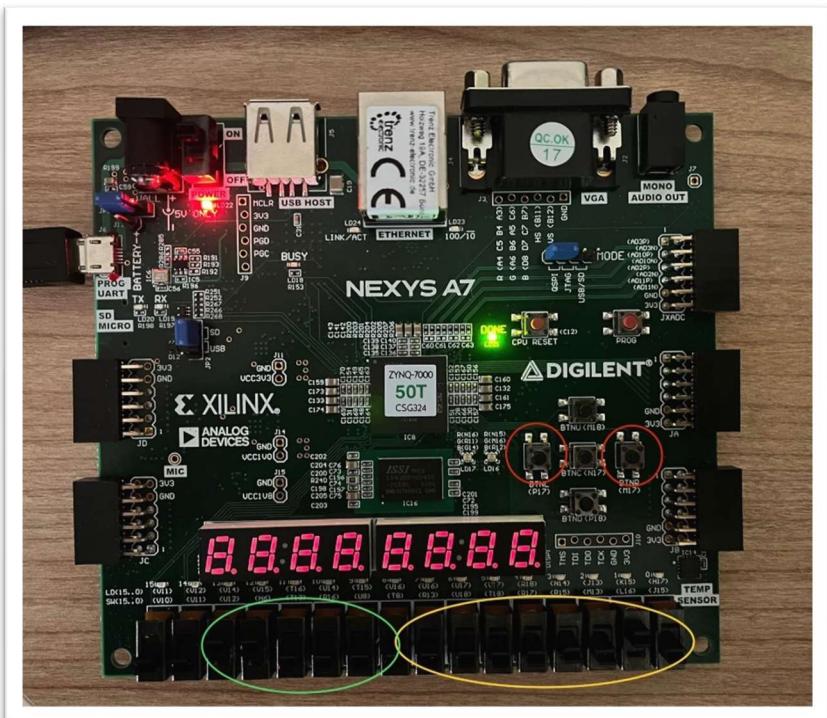
```

entity rete_di_interconnessione_on_board is
  Port ( clock : in STD_LOGIC;
         load_first_part : in STD_LOGIC;
         load_second_part : in STD_LOGIC;
         value8_in : in STD_LOGIC_VECTOR (0 to 7);
         value6_in : in STD_LOGIC_VECTOR (5 downto 0);
         value4_out : out STD_LOGIC_VECTOR (0 to 3));
end rete_di_interconnessione_on_board;

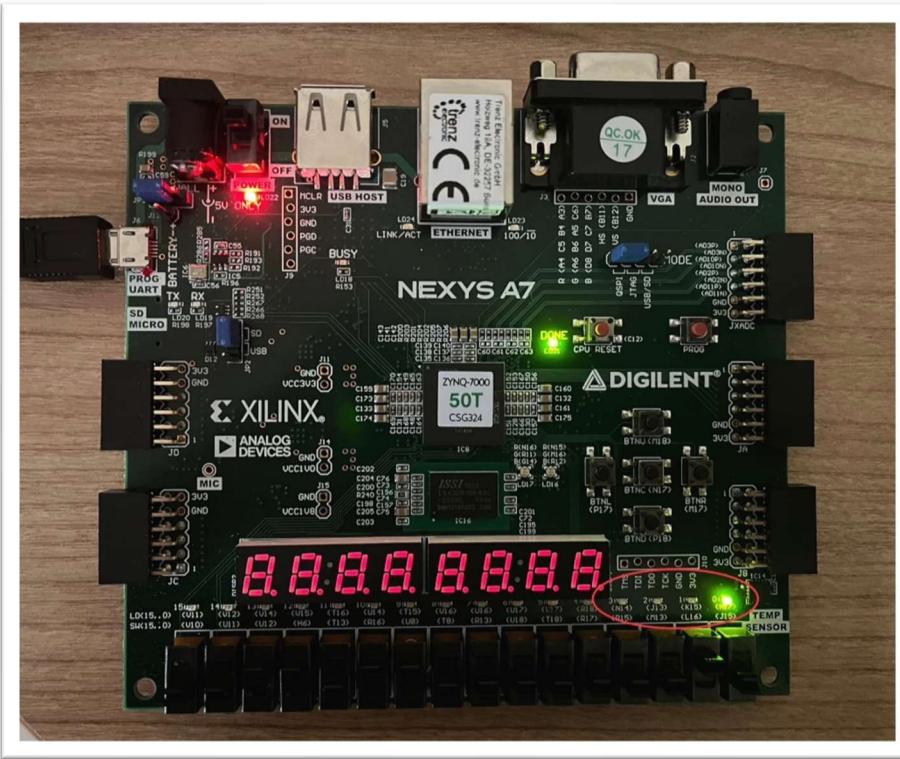
cui: cu port map(
  clock => clock,
  load_first_part => load_first_part,
  load_second_part => load_second_part,
  value8_in => value8_in,
  value16_out => cu_value
);

r1: rete_g1_interconnessione port map(
  a0 => cu_value(0),
  a1 => cu_value(1),
  a2 => cu_value(2),
  a3 => cu_value(3),
  a4 => cu_value(4),
  a5 => cu_value(5),
  a6 => cu_value(6),
  a7 => cu_value(7),
  a8 => cu_value(8),
  a9 => cu_value(9),
  a10 => cu_value(10),
  a11 => cu_value(11),
  a12 => cu_value(12),
  a13 => cu_value(13),
  a14 => cu_value(14),
  a15 => cu_value(15),
  y0 => value4_out(0),
  y1 => value4_out(1),
  y2 => value4_out(2),
  y3 => value4_out(3),
  s0 => value6_in(0),
  s1 => value6_in(1),
  s2 => value6_in(2),
  s3 => value6_in(3),
  s4 => value6_in(4),
  s5 => value6_in(5));
end Behavioral;

```



I due cerchi rossi indicano i due bottoni utilizzati per prelevare i bit dagli 8 switch cerchiati di giallo, nel test riportato nell'immagine ci limitiamo ad inserire per due volte la sequenza *00000100*. In verde osserviamo gli switch usati per effettuare la connessione tra sorgente e destinazione, i primi due più a sinistra prendono i bit per selezionare la destinazione, in questo caso *11*, mentre gli altri la sorgente, nell'immagine *0110*, il risultato viene mostrato sui quattro led cerchiati nella prossima immagine, dove indichiamo le 4 possibili destinazione da sinistra verso destra con *00*, *01*, *10* e *11*.



Osserviamo anche i collegamenti fatti sul file di constraints.

```
## Clock signal
set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { clock }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clock}]


##Switches
set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 } [get_ports { value8_in[7] }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 } [get_ports { value8_in[6] }]; #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sv[1]
set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMOS33 } [get_ports { value8_in[5] }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMOS33 } [get_ports { value8_in[4] }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
set_property -dict { PACKAGE_PIN R17      IOSTANDARD LVCMOS33 } [get_ports { value8_in[3] }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMOS33 } [get_ports { value8_in[2] }]; #IO_L7N_T1_D10_14 Sch=sv[5]
set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMOS33 } [get_ports { value8_in[1] }]; #IO_L17N_T2_A13_D29_14 Sch=sv[6]
set_property -dict { PACKAGE_PIN R13      IOSTANDARD LVCMOS33 } [get_ports { value8_in[0] }]; #IO_L5N_T0_D07_14 Sch=sv[7]
set_property -dict { PACKAGE_PIN T8       IOSTANDARD LVCMOS18 } [get_ports { value6_in[0] }]; #IO_L24N_T3_34 Sch=sv[8]
set_property -dict { PACKAGE_PIN U8       IOSTANDARD LVCMOS18 } [get_ports { value6_in[1] }]; #IO_25_34 Sch=sv[9]
set_property -dict { PACKAGE_PIN R16      IOSTANDARD LVCMOS33 } [get_ports { value6_in[2] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sv[10]
set_property -dict { PACKAGE_PIN T13      IOSTANDARD LVCMOS33 } [get_ports { value6_in[3] }]; #IO_L23P_T3_A03_D19_14 Sch=sv[11]
set_property -dict { PACKAGE_PIN H6       IOSTANDARD LVCMOS33 } [get_ports { value6_in[4] }]; #IO_L24P_T3_35 Sch=sv[12]
set_property -dict { PACKAGE_PIN U12      IOSTANDARD LVCMOS33 } [get_ports { value6_in[5] }]; #IO_L20P_T3_A08_D24_14 Sch=sv[13]
#set_property -dict { PACKAGE_PIN U11      IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sv[14]
#set_property -dict { PACKAGE_PIN V10      IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DOS_14 Sch=sv[15]
```

```

## LEDs
set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 } [get_ports { value4_out[3] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15    IOSTANDARD LVCMOS33 } [get_ports { value4_out[2] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13    IOSTANDARD LVCMOS33 } [get_ports { value4_out[1] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14    IOSTANDARD LVCMOS33 } [get_ports { value4_out[0] }]; #IO_L8P_T1_D11_14 Sch=led[3]
#set nognantrw -dict { PACKAGE_PIN D7M D12    IOSTANDARD LVCMOS33 } [get_ports { TDATA11 }]; #IO_T7D_T1_D0Q_14 Sch=led[4]

```

```

set_property -dict { PACKAGE_PIN P17    IOSTANDARD LVCMOS33 } [get_ports { load_second_part }]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17    IOSTANDARD LVCMOS33 } [get_ports { load_first_part }]; #IO_L10N_T1_D15_14 Sch=btnr

```

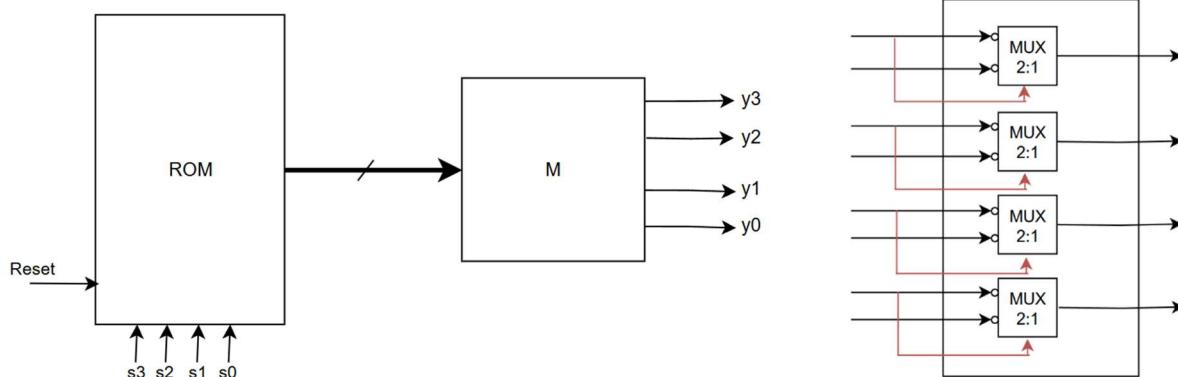
## Esercizio 2: ROM+M

### Esercizio 2.1

Progettare, implementare in VHDL e testare mediante simulazione un sistema S composto da una ROM puramente combinatoria di 16 locazioni da 8 bit ciascuna e da una macchina combinatoria M che opera come segue: fornito al sistema un indirizzo A di 4 bit, il sistema restituisce il valore contenuto nella ROM all'indirizzo A opportunamente “trasformato” attraverso la macchina M. Il comportamento della macchina M è totalmente a scelta dello studente, l'unico vincolo è che essa prenda in ingresso 8 bit e ne fornisca in uscita 4.

#### *Progetto e architettura*

In questo progetto abbiamo previsto di inserire una ROM combinatoria che in presenza di un indirizzo rilascia il contenuto della locazione indicata (espresso su 8 bit), questi entrano all'interno di una macchina M che a seguito di una semplice operazione combinatoria trasforma gli 8 bit in 4.



#### *Implementazione*

Essendo una ROM completamente combinatoria ed avendo usato negli altri progetti ROM sequenziali (in appendice) riportiamo l'implementazione anche della ROM.

## ROM – DATAFLOW (*rom.vhd*)

```
entity rom is
  Generic ( n: positive := 3; -- selezione
            l : positive := 15; -- numero di locazioni
            m : positive :=7 -- lunghezza dato
          );
  Port (
    reset : in STD_LOGIC;
    selezione : in STD_LOGIC_VECTOR (n downto 0);
    dato : out STD_LOGIC_VECTOR (m downto 0));
end rom;

architecture Dataflow of rom is

type rom_type is array (0 to 1) of std_logic_vector(m downto 0);

-- devo istanziare una ROM e inizializzarla

signal ROM : rom_type := (

X"2E",X"A4",X"02",X"35",
X"82",X"FE",X"AF",X"D2",
X"0B",X"A1",X"D6",X"AC",
X"C0",X"79",X"B6",X"11"
);

begin

dato <= X"FF" when reset = '1' else
  ROM(conv_integer(selezione));

end Dataflow;
```

Con il reset alzato l'uscita è FF in modo che l'elaborazione della macchina porti ad osservare tutti 0 come uscita totale.

## MACCHINA – STRUCTURAL (*machine.vhd*)

```
entity machine is
  Port ( data_in : in STD_LOGIC_VECTOR (7 downto 0);
         y : out STD_LOGIC_VECTOR (3 downto 0));
end machine;

architecture Structural of machine is

component mux_2_1

  port(  a0 : in STD_LOGIC;
         a1 : in STD_LOGIC;
         s : in STD_LOGIC;
         y : out STD_LOGIC
       );
end component;

signal not_data_in: std_logic_vector(7 downto 0);

begin

not_data_in <= not data_in;

mux_n_to_0: for n in 3 downto 0 generate

  mux_comp: mux_2_1 port map(
    a0 => not_data_in((2*n)+1),
    a1 => not_data_in(2*n),
    s => data_in(2*n+1),
    y => y(n)
  );

end generate mux_n_to_0;

end Structural;
```

Questa macchina riceve gli 8 bit in uscita dalla ROM, li inverte, poi usa un MUX 2:1 per ogni coppia di bit, in modo che il più significativo tra i due per ogni coppia degli 8 bit diventi selezione del MUX.

Ad esempio se arriva la sequenza **00100110**, la macchina inverte tutti i bit che diventano **11011001**, poi un MUX 2:1 riceve i primi due bit **11**, come selezione entra il bit più significativo tra i due con il valore che aveva prima di essere invertito, quindi **0**, per cui l'uscita del MUX sarà **1**.

## ROM+M – STRUCTURAL (*rom\_m.vhd*)

```
entity rom_m is
    Port ( s : in STD_LOGIC_VECTOR(3 downto 0);
            reset : in STD_LOGIC;
            y : out STD_LOGIC_VECTOR(3 downto 0));
end rom_m;
```

```
architecture Behavioral of rom_m is

component rom
Generic ( n: positive := 3; -- selezione
           l : positive := 15; -- numero di locazioni
           m : positive := 7 -- lunghezza dato
        );
Port (
            reset : in STD_LOGIC;
            selezione : in STD_LOGIC_VECTOR (3 downto 0);
            dato : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component machine
Port ( data_in : in STD_LOGIC_VECTOR (7 downto 0);
        y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal u0: std_logic_vector(7 downto 0);
```

```
begin
    mem: rom
    generic map(
        n => 3,
        l => 15,
        m => 7
    )
    port map(
        reset => reset,
        selezione => s,
        dato => u0
    );

    m: machine port map(
        data_in => u0,
        y => y
    );
end Behavioral;
```

## Simulazione

Per la simulazione si è utilizzato il seguente process di stimoli all'interno del testbench:

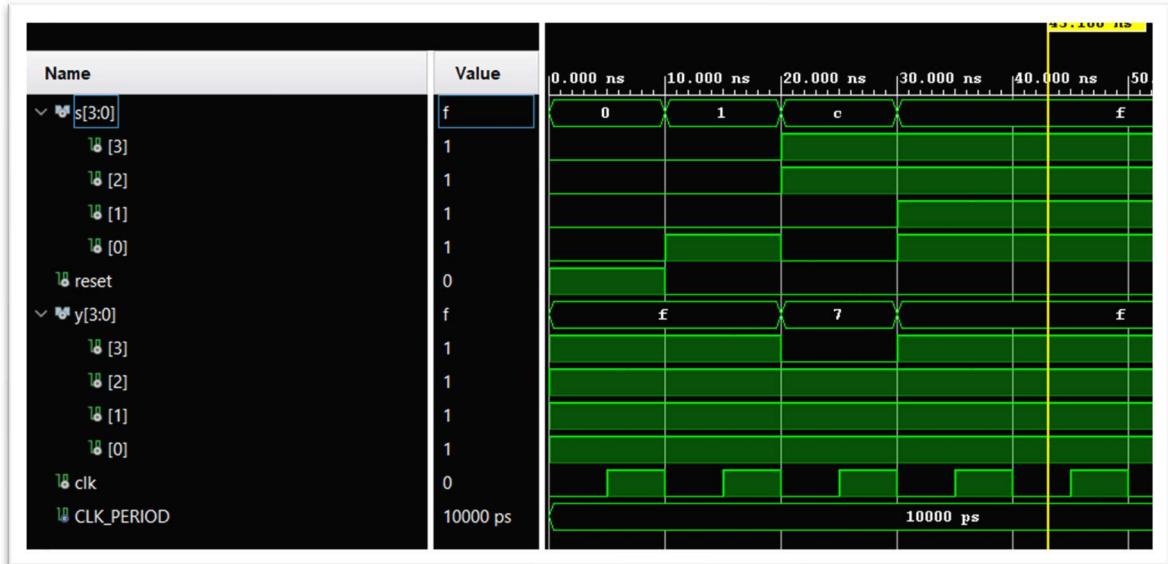
```
-- Stimulus process
stimulus_process: process
begin
    -- Step 1: Reset the system
    report "Starting simulation...";
    reset <= '1';
    wait for CLK_PERIOD;
    reset <= '0';

    report "Testing random values...";
    s <= "0001";
    wait for CLK_PERIOD;
    s <= "1100";
    wait for CLK_PERIOD;
    s <= "1111";
    wait for CLK_PERIOD;

    -- Simulation complete
    report "Simulation complete!";
    wait;
end process;
```

Ovvero sono stati forniti in ingresso al sistema 3 diversi indirizzi di memoria e si è osservato il valore in uscita. In particolare i valori contenuti negli indirizzi indicati sono 0xA4 (10100100), 0xAF (10101111) e 0x11 (00010001).

Considerata la funzione applicata da M, osserviamo che complementando ad esempio il primo ci aspettiamo il valore 01011011, i valori selezionati dal MUX in uscita saranno 1111, come possiamo apprezzare anche dalla simulazione:



### Sintesi su board di sviluppo

#### Esercizio 2.2

Sintetizzare ed implementare su board il progetto del sistema ROM+M sviluppato al punto 2.1, utilizzando gli switch per fornire l'indirizzo della ROM da cui leggere i valori da trasformare e i led per visualizzare i 4 bit di uscita.

Codice della rom e della macchina sulla board:

```

entity rom_on_board is
    Port ( s : in STD_LOGIC_VECTOR (3 downto 0);
           reset : in STD_LOGIC;
           y : out STD_LOGIC_VECTOR (3 downto 0));
end rom_on_board;

architecture Behavioral of rom_on_board is

component rom_m  Port ( s : in STD_LOGIC_VECTOR(3 downto 0);
                           reset : in STD_LOGIC;
                           y : out STD_LOGIC_VECTOR(3 downto 0));
end component;

signal selezioni: std_logic_vector(3 downto 0);
signal uscite: std_logic_vector(3 downto 0);
signal rst: std_logic;

begin

selezioni <= s;
rst <= reset;
y <= uscite;

romma: rom_m port map(
    s => selezioni,
    reset => rst,
    y => uscite
);

end Behavioral;

```

Riportiamo il file dei constraints:

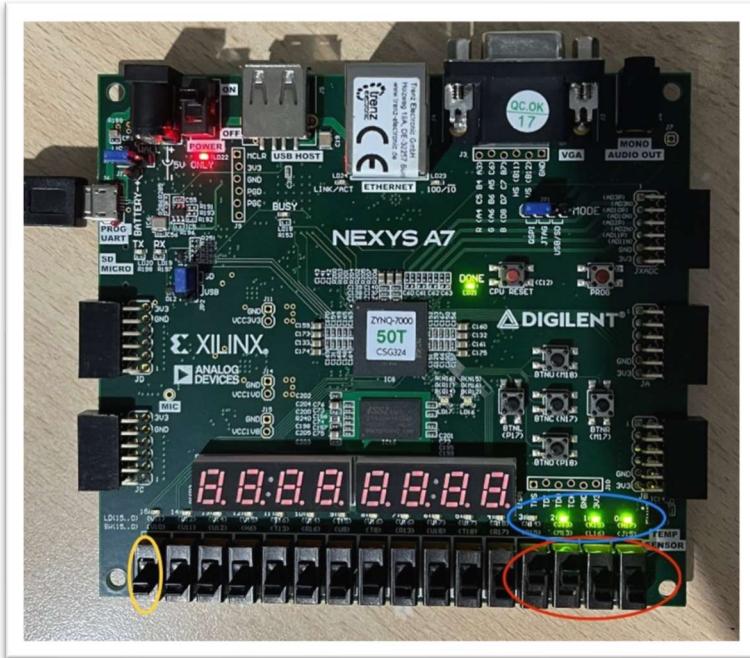
```

##Switches
set_property -dict { PACKAGE_PIN J15  IOSTANDARD LVCMS33 } [get_ports { s[0] }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
set_property -dict { PACKAGE_PIN L16  IOSTANDARD LVCMS33 } [get_ports { s[1] }]; #IO_L3N_T0_DQ5_EMCCCLK_14 Sch=sv[1]
set_property -dict { PACKAGE_PIN M13  IOSTANDARD LVCMS33 } [get_ports { s[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
set_property -dict { PACKAGE_PIN R15  IOSTANDARD LVCMS33 } [get_ports { s[3] }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
#set_property -dict { PACKAGE_PIN R17  IOSTANDARD LVCMS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
#set_property -dict { PACKAGE_PIN T18  IOSTANDARD LVCMS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sv[5]
#set_property -dict { PACKAGE_PIN U18  IOSTANDARD LVCMS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sv[6]
#set_property -dict { PACKAGE_PIN R13  IOSTANDARD LVCMS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sv[7]
#set_property -dict { PACKAGE_PIN T8   IOSTANDARD LVCMS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sv[8]
#set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sv[9]
#set_property -dict { PACKAGE_PIN R16  IOSTANDARD LVCMS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sv[10]
#set_property -dict { PACKAGE_PIN H6   IOSTANDARD LVCMS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sv[11]
#set_property -dict { PACKAGE_PIN T13  IOSTANDARD LVCMS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sv[12]
#set_property -dict { PACKAGE_PIN U12  IOSTANDARD LVCMS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sv[13]
#set_property -dict { PACKAGE_PIN U11  IOSTANDARD LVCMS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sv[14]
set_property -dict { PACKAGE_PIN V10  IOSTANDARD LVCMS33 } [get_ports { reset }]; #IO_L21P_T3_DQS_14 Sch=sv[15]

## LEDs
set_property -dict { PACKAGE_PIN M17  IOSTANDARD LVCMS33 } [get_ports { y[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15  IOSTANDARD LVCMS33 } [get_ports { y[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13  IOSTANDARD LVCMS33 } [get_ports { y[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14  IOSTANDARD LVCMS33 } [get_ports { v[3] }]; #IO_L6P_T1_D11_14 Sch=led[3]

```

Per sintetizzare il programma su board abbiamo utilizzato i primi 4 shift a destra (in rosso) per codificare l'indirizzo, il primo bit a sinistra per codificare il reset (in giallo) e i primi 4 led (in blu) per visualizzare l'uscita. In figura vediamo l'indirizzo in ingresso 0xC che dà in uscita il valore 0x7.



## Capitolo 2: Reti sequenziali elementari

### Esercizio 3: Riconoscitore di sequenze

#### Esercizio 3.1

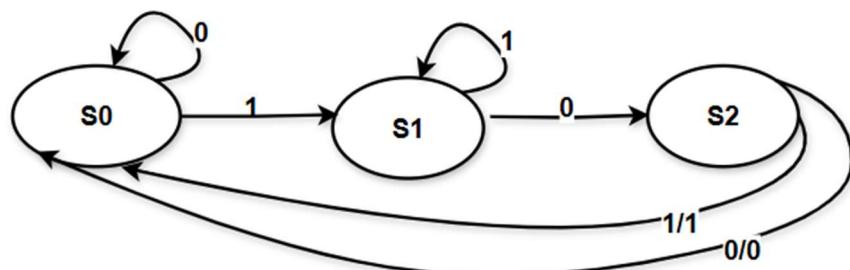
Progettare, implementare in VHDL e testare mediante simulazione una macchina in grado di riconoscere la sequenza 101. La macchina prende in ingresso un segnale binario  $i$  che rappresenta il dato, un segnale  $A$  di temporizzazione e un segnale  $M$  di modo, che ne disciplina il funzionamento, e fornisce un'uscita  $Y$  alta quando la sequenza viene riconosciuta. In particolare,

- se  $M=0$ , la macchina valuta i bit seriali in ingresso a gruppi di 3 (sequenze non sovrapposte),
- se  $M=1$ , la macchina valuta i bit seriali in ingresso uno alla volta, tornando allo stato iniziale ogni volta che la sequenza viene correttamente riconosciuta (sequenze parzialmente sovrapposte).

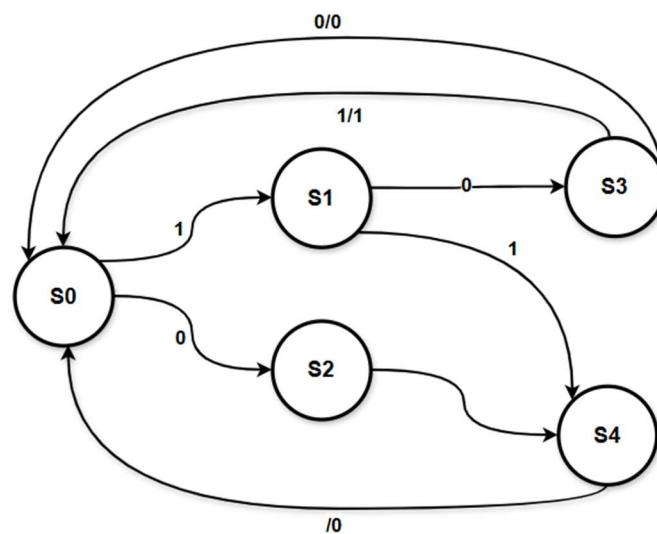
#### Progetto e architettura

Si vuole realizzare un riconoscitore di sequenze in grado di funzionare in due modalità. Si vuole un comportamento tale che al variare della modalità, la macchina inizi a riconoscere una nuova sequenza a partire dal dato corrente.

#### Automa del riconoscitore di sequenze 101 parzialmente sovrapposte



#### Automa del riconoscitore di sequenze 101 non sovrapposte



Vogliamo che a seconda se la modalità è 1, la macchina lavora come rappresentato dal primo automa, se la modalità è pari a 0, come dal secondo. Ogni volta che la modalità viene cambiata l'automa torna nello stato S0 e ricomincia l'elaborazione.

Il segnale che rappresenta la modalità dunque, funge da reset per una delle due modalità. Abbiamo poi un reset della macchina che invece fa da reset per entrambe.

#### *Implementazione*

Per implementare il comportamento sopra rappresentato tramite gli automi, abbiamo 3 process, uno per ciascun automa e uno per l'uscita.

Oltre al segnale di temporizzazione richiesto dalla specifica, si è ritenuto opportuno fornire anche un segnale di reset, in corrispondenza del quale, il riconoscitore riparte da capo e resta in attesa finché questo è alto. In questo modo si ha un reset della macchina complessiva, non solo di una delle due modalità.

### RICONOSCITORE – BEHAVIORAL (*RiconoscitoreDiSequenze.vhd*)

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

entity RiconoscitoreDiSequenze is
    Port ( A : in STD_LOGIC;
            reset: in STD_LOGIC;
            Mode : in STD_LOGIC;
            i : in STD_LOGIC;
            y : out STD_LOGIC);
end RiconoscitoreDiSequenze;

architecture Behavioral of RiconoscitoreDiSequenze is
type stato is (S0, S1, S2, S3, S4);
signal stato_corrente0 : stato := S0;
signal stato_corrente1 : stato := S0;
```

```

:9
:0 begin
:1 --RICONOSCITORE DI SEQUENZE PARZIALMENTE SOVRAPPOSTO 101 MODE1
:2 --Sul fronte di clock, vedo se il segnale di button è alto, in tal caso acquisisco l'ingresso e cambio stato
:3 stato_uscita_mem1: process(a)
:4 begin
:5 if(reset ='1') then
:6     stato_corrente1<=s0;
:7 elsif(a'event and a='1') then
:8     if(Mode = '0') then
:9         stato_corrente1 <= s0;
:0 elsif(Mode = '1') then
:1     case stato_corrente1 is
:2
:3         when s0 =>
:4             if( i = '0' ) then
:5                 stato_corrente1 <= s0;
:6             else
:7                 stato_corrente1 <= s1;
:8             end if;
:9         when s1 =>
:10            if( i = '0' ) then
:11                stato_corrente1 <= s2;
:12            else
:13                stato_corrente1 <= s1;
:14            end if;
:15         when s2 =>
:16             stato_corrente1 <= s0;
:17         when others =>
:18             stato_corrente1 <= s0;
:19         end case;
:20     end if;
:21 end if;
:22 end process;
:23

```

```

:~ :--RICONOSCITORE DI SEQUENZE NON SOVRAPPOSTE 101 MODE 0
:44 --Sul fronte di clock, vedo se il segnale di button è alto, in tal caso acquisisco l'ingresso e cambio stato
:45 stato_uscita_mem0: process(a)
:46 begin
:47 if(reset ='1') then
:48     stato_corrente0<=s0;
:49 elsif(a'event and a='1') then
:50     if( Mode = '1') then
:51         stato_corrente0 <= s0;
:52     elsif(Mode = '0') then
:53         case stato_corrente0 is
:54             when s0 =>
:55                 if( i = '0' ) then
:56                     stato_corrente0 <= s2;
:57                 else
:58                     stato_corrente0 <= s1;
:59                 end if;
:60             when s1 =>
:61                 if( i = '0' ) then
:62                     stato_corrente0 <= s3;
:63                 else
:64                     stato_corrente0 <= s4;
:65                 end if;
:66             when s2 =>
:67                 if( i = '0' ) then
:68                     stato_corrente0 <= s4;
:69                 else
:70                     stato_corrente0 <= s4;
:71                 end if;
:72             when s3 =>
:73                 if( i = '0' ) then
:74                     stato_corrente0 <= s0;
:75                 else
:76                     stato_corrente0 <= s0;
:77                 end if;
:78 end process;

```

```

        end if;
      when S4 =>
        if( i = '0' ) then
          stato_corrente0 <= S0;
        else
          stato_corrente0 <= S0;
        end if;
      when others =>
        stato_corrente0 <= S0;
    end case;
  end if;
end if;
end process;

```

```

process(a)
begin
  if rising_edge(a) then
    if(reset ='1') then
      Y <= '0';
    elsif( Mode = '1') then
      if(stato_corrente1 = S2 and i = '1') then
        Y <= '1';
      else
        Y <= '0';
      end if;
    else
      if(stato_corrente0 = S3 and i = '1') then
        Y <= '1';
      else
        Y <= '0';
      end if;
    end if;
  end if;
end process;

end Behavioral;

```

### *Simulazione*

Per la simulazione è stata scelta una sequenza tale che il comportamento del riconoscitore sia diverso per le due modalità. Si riporta di seguito il testbench.

```

3 | library ieee;
4 | use ieee.std_logic_1164.all;
5 |
6 | entity tb_RiconoscitoreDiSequenze is
7 | end tb_RiconoscitoreDiSequenze;
8 |
9 | architecture tb of tb_RiconoscitoreDiSequenze is
10 |
11 |     component RiconoscitoreDiSequenze
12 |         port (A      : in std_logic;
13 |                 reset : in std_logic;
14 |                 Mode   : in std_logic;
15 |                 i      : in std_logic;
16 |                 y      : out std_logic);
17 |     end component;
18 |
19 |     signal A      : std_logic;
20 |     signal reset : std_logic;
21 |     signal Mode   : std_logic;
22 |     signal i      : std_logic;
23 |     signal y      : std_logic;
24 |
25 |     constant TbPeriod : time := 10 ns; -- EDIT Put right period here
26 |     signal TbClock : std_logic := '0';
27 |     signal TbSimEnded : std_logic := '0';
28 |
29 |

```

```

begin

    dut : RiconoscitoreDiSequenze
    port map (A      => A,
              reset  => reset,
              Mode   => Mode,
              i      => i,
              y      => y);

    -- Clock generation
○    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';

    -- EDIT: Replace YOURCLOCKSIGNAL below by the name of your clock as I haven't guessed it
○    A <= TbClock;

    stimuli : process
    begin
        -- EDIT Adapt initialization as neededs
○        Mode <= '0';
○        i <= '1';

        -- Reset generation
        -- EDIT: Check that reset is really your reset signal
○        reset <= '1';
○        wait for 10 ns;
○        reset <= '0';
○        wait for 10 ns;

```

```

i<='0';
wait for 10 ns;
i<'1';
wait for 10 ns;
i<'1';
wait for 10 ns;
i<'1';
wait for 10 ns;
i<='0';
wait for 10 ns;
i<'1';
wait for 10 ns;

mode<='1';
i<='1';
wait for 10 ns;
i<='0';
wait for 10 ns;
i<'1';
wait for 10 ns;
i<'1';
wait for 10 ns;
i<='1';
wait for 10 ns;
i<='1';
wait for 10 ns;
i<='0';
wait for 10 ns;
i<'1';
wait for 10 ns;

-- EDIT Add stimuli here
wait for 100 * TbPeriod;

-- Stop the clock and hence terminate the simulation
TbsimEnded <= '1';
wait;
end process;

end tb;

```

Facendo una simulazione di tipo behavioral, notiamo come cambia l'uscita al variare della modalità rispetto alla sequenza in ingresso proposta.



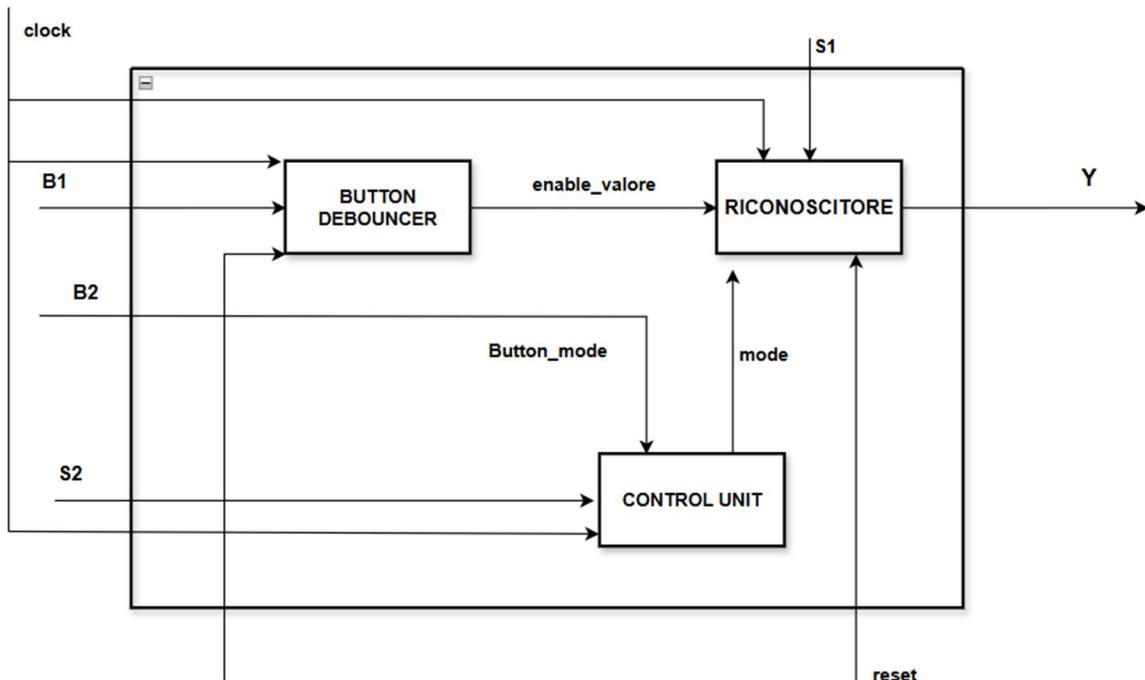
Quando mode=0, modalità non sovrapposta, l'uscita è alta solo una volta. Quando mode=1, invece considerando sequenze parzialmente sovrapposte, l'uscita è pari a 1 due volte.

### Esercizio 3.2

Sintetizzare e implementare su board la rete sviluppata al punto precedente, utilizzando uno switch S1 per codificare l'input i e uno switch S2 per codificare il modo M, in combinazione con due bottoni B1 e B2 utilizzati rispettivamente per acquisire l'input da S1 e S2 in sincronismo con il segnale di temporizzazione A, che deve essere ottenuto a partire dal clock della board. Infine, l'uscita Y può essere codificata utilizzando un led.

#### Progetto e architettura

Il segnale di ingresso al riconoscitore e la modalità di funzionamento vengono letti dagli switch in corrispondenza di un opportuno segnale proveniente dai bottoni. Mentre il segnale di abilitazione è dato dal clock della scheda. Possiamo usare il riconoscitore dell'esercizio 3.1 senza apportare alcuna modifica ad esso, fornendogli dei segnali i e mode, assumono il valore desiderato solo quando il bottone è attivo. Per fare ciò avremmo bisogno di un blocco che ci permetta di valutare ciò, si è pensato di dare tale responsabilità ad un'unità di controllo. Per fare in modo che il riconoscitore legga il valore del segnale in ingresso direttamente dallo switch, è stato fornito in ingresso ad esso un segnale di enable, che indica che il relativo bottone è premuto. Dunque quello che nel punto 3.1 era il segnale A, nell'implementazione su board è una and del segnale di clock della scheda, e del segnale in uscita al button debouncer di B1. Settare la modalità di funzionamento invece è una responsabilità che passa per l'unità di controllo. Vgliamo che il segnale di enable per leggere il valore di ingresso duri un ciclo di clock, in questo modo si evita di leggere più volte lo stesso ingresso. A tal fine è stato usato un button debouncer. Per quanto riguarda l'acquisizione del segnale relativo alla modalità, ciò non è stato ritenuto necessario, in quanto ci interessa solo quando la modalità cambia valore.



#### Implementazione

L'implementazione del riconoscitore è quella presentata al punto precedente, al netto dell'aggiunta del segnale di enable. Per semplicità si riportano di seguito le sole righe di codice modificate.

#### MODIFICHE AL RICONOSCITORE 3.1

```
32 end entity RiconoscitoreDiSequenze is
33   Port ( clock : in STD_LOGIC;
34         reset: in STD_LOGIC;
35         Enable: in STD_LOGIC;
36         Mode : in STD_LOGIC;
37         i : in STD_LOGIC;
38         y : out STD_LOGIC);
39 end RiconoscitoreDiSequenze;
```

```
52 begin
53   stato_uscita_mem1: process(clock)
54   begin
55     if(reset ='1') then
56       stato_corrente1<=S0;
57     elsif(clock'event and clock='1') then
58       if(Mode = '0') then
59         stato_corrente1 <= S0;
60       elsif(Enable ='1') then
61         case stato_corrente1 is
```

```
69 begin
70   stato_uscita_mem0: process(clock)
71   begin
72     if(reset ='1') then
73       stato_corrente0<=S0;
74     elsif(clock'event and clock='1') then
75       if( Mode = '1') then
76         stato_corrente0 <= S0;
77       elsif(Enable ='1') then
78         case stato_corrente0 is
```

```
14 begin
15   if rising_edge(clock) then
16     if reset = '1' then
17       y <= '0';
18
19     elsif(Enable='1') then
20       if( Mode = '1') then
21         if(stato_corrente1 = S2 and i = '1') then
22           y <= '1';
23         else
24           y <= '0';
25         end if;
```

## CONTROL UNIT – BEHAVIORAL (*control\_unit.vhd*)

```
1 entity control_unit is
2     Port ( clock : in STD_LOGIC;
3         B_mode : in STD_LOGIC;
4         S_mode : in STD_LOGIC;
5         mode : out STD_LOGIC
6     );
7 end control_unit;
8
9 architecture Behavioral of control_unit is
10
11 begin
12
13     main: process(clock)
14     begin
15         if clock'event and clock='1' then
16             if B_mode = '1' then
17                 mode <= S_mode;
18             end if;
19         end if;
20     end process;
21
22     . . .
23 
```

Il button debouncer è un componente riportato in appendice.

Si riporta di seguito il top module del progetto. Architettura strutturale, secondo lo schema sopra riportato, per l'implementazione su board.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4
5 --S1 switch per codificare valore, caricato con B1 -> primo switch a destra, bottone u
6 --S2 switch per codificare il modo, caricato con B2 -> secondo switch da destra, bottone o
7 entity riconoscitore_on_board is
8     Port ( clock_in : in STD_LOGIC;
9         reset : in STD_LOGIC;
10        B1 : in STD_LOGIC;
11        B2 : in STD_LOGIC;
12        S1 : in STD_LOGIC;
13        S2 : in STD_LOGIC;
14        LED : out STD_LOGIC);
15 end riconoscitore_on_board;
16
17 architecture structural of riconoscitore_on_board is
18
19 component control_unit
20     Port ( clock : in STD_LOGIC;
21         B_mode : in STD_LOGIC;
22         S_mode : in STD_LOGIC;
23         mode : out STD_LOGIC
24     );
25 end component.
```

```

56 component RiconoscitoreDiSequenze
57     Port ( clock : in STD_LOGIC;
58             reset: in STD_LOGIC;
59             Enable: in STD_LOGIC;
60             Mode : in STD_LOGIC;
61             i : in STD_LOGIC;
62             y : out STD_LOGIC);
63 end component;
64
65 component ButtonDebouncer
66 generic (
67     CLK_period: integer := 10; -- periodo del clock (della board) in nanosecondi
68     btn_noise_time: integer := 10000000 -- durata stimata dell'oscillazione del bottone in nanosecondi
69     -- il valore di default è 10 millisecondi
70 );
71     Port ( RST : in STD_LOGIC;
72             CLK : in STD_LOGIC;
73             BTN : in STD_LOGIC;
74             CLEARED_BTN : out STD_LOGIC);
75 end component;
76
77 signal enable_valore: std_logic;
78 signal mode: std_logic;
79 signal Reset_converted: std_logic;

```

```

81 begin
82     Reset_converted <= not reset; --visto che utilizzo il bottone CPU_reset della board, che è attivo-basso,
83     --devo convertire il segnale di reset
84
85 --VALORE
86 B1_clear: ButtonDebouncer
87 GENERIC MAP(
88     CLK_period => 10, -- periodo del clock della board pari a 10ns
89     btn_noise_time => 10000000 --intervallo di tempo in cui si ha l'oscillazione del bottone
90     --assumo che duri 10ms
91 )
92 PORT MAP ( RST => Reset_converted,
93             CLK => clock_in,
94             BTN => B1,
95             CLEARED_BTN => enable_valore
96 );
97
98 cu: control_unit PORT MAP(clock_in, B2, S2, mode);
99
00 riconoscitore: RiconoscitoreDiSequenze
01 PORT MAP(
02     clock => clock_in,
03     reset => Reset_converted,
04     Enable => enable_valore,
05     Mode => mode,
06     i => S1,
07     y => LED
08 );
09

```

## Sintesi su board di sviluppo

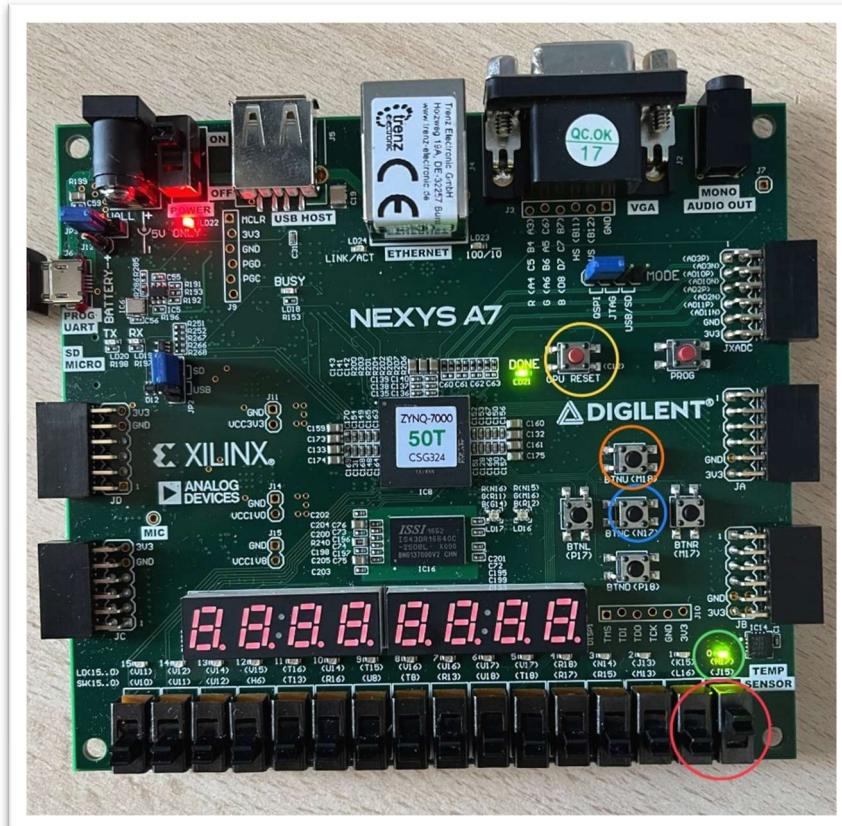
Per la sintesi sulla board sono state apportate alcune modifiche ai file dei constraints.

```
1 | ## CLOUD SIGNALS
2 | set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clock_in }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
3 | create_clock -add -name sys_clk_pin -period 10.00 -waveform { 0 5 } [get_ports { clock_in }];
4 |
5 |
6 | ##Switches
7 | set_property -dict { PACKAGE_PIN J15    IOSTANDARD LVCMOS33 } [get_ports { S1 }]; #IO_L24N_T3_RSO_15 Sch=sv[0]
8 | set_property -dict { PACKAGE_PIN L16    IOSTANDARD LVCMOS33 } [get_ports { S2 }]; #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sv[1]
9 | #set_property -dict { PACKAGE_PIN M13    IOSTANDARD LVCMOS33 } [get_ports { S2 }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
10 | #set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
11 |
12 |
13 |
14 |
15 |
16 |
17 |
18 |
19 |
20 |
21 |
22 |
23 |
24 |
25 |
26 |
27 |
28 |
29 |
30 | set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 } [get_ports { LED }]; #IO_L18P_T2_A24_15 Sch=led[0]
31 | #set_property -dict { PACKAGE_PIN K15    IOSTANDARD LVCMOS33 } [get_ports { LED[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
```

```
1 | ## PULLDOWN
2 | set_property -dict { PACKAGE_PIN C12    IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
3 | set_property -dict { PACKAGE_PIN N17    IOSTANDARD LVCMOS33 } [get_ports { B2 }]; #IO_L9P_T1_DQS_14 Sch=btnc
4 | #set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { R1_11 }]; #IO_L4N_T0_D05_1d Sch=btncat
5 | #set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { R1_11 }]; #IO_L4N_T0_D05_1d Sch=btncat
```

Poiché il segnale di reset è stato acquisito dal bottone di cpu\_reset della scheda, che è attivo basso, questo è stato convertito prima di essere posto in ingresso ai componenti mostrati sopra.

Si riporta infine l'immagine della board: in giallo indichiamo il pulsante di reset, in blu il pulsante per settare la modalità, in arancio quello per abilitare il riconoscitore a riconoscere, l'ultimo switch a destra si occupa di prelevare il bit i di ingresso al riconoscitore, mentre lo switch adiacente si occupa di inserire la modalità in cui deve lavorare il riconoscitore.



## Timing Analysis

Effettuare la **Timing Analysis** in Vivado ci consente di verificare che il design FPGA soddisfi i vincoli di temporizzazione e funzioni correttamente a una determinata frequenza di clock. Il tempo di setup e il tempo di hold definiscono l'intervallo in cui un segnale di input deve rimanere stabile rispetto al clock. Il tempo di setup, in particolare, influenza sulla massima frequenza operativa, poiché indica per quanto tempo l'input deve restare stabile prima del fronte attivo del clock, riducendo di conseguenza il tempo disponibile all'interno di un ciclo di clock.

Nel file constraints, mostrato al paragrafo precedente, è stato impostato un vincolo sul periodo di clock pari a 10 ns.

The screenshot shows the Vivado interface. On the left is the Constraints Editor window titled 'e(Behavioral) (Ricon...', containing a text editor with the following code:

```
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the
## Clock signal
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clock_in }]; #IO_L12P_T1_MRCC_3
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clock_in}];
#
##Switches
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { s1 }]; #IO_L24N_T3_RS0_15 Sch=s1
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { s2 }]; #IO_L3N_T0_DQS_EMCCCLK_14
#set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { S2 }]; #IO_L6N_T0_D08_VREF_14 .
#set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 .
#set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 .
#set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=d10
#set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29 .
```

The bottom part of the interface shows the 'Design Timing Summary' report with tabs for Design Runs, DRC, Methodology, Power, and Timing. The 'Timing' tab is selected, displaying the following data:

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 4,371 ns	Worst Hold Slack (WHS): 0,118 ns	Worst Pulse Width Slack (WPWS): 4,500 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWNS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 104	Total Number of Endpoints: 104	Total Number of Endpoints: 43

All user specified timing constraints are met.

A valle dell'implementazione, facendo una timing analysis, notiamo che le specifiche richieste sono state rispettate. In particolare che il minimo periodo di clock, affinchè questo avvenga è pari a 4.5 ns. Proviamo dunque a rifare l'analisi, imponendo un vincolo sul periodo di clock pari a 2ns.

```

6 ## Clock signal
7 set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clock_in }]; #IO_L12P_T1_MRCC_35
8 create_clock -add -name sys_clk_pin -period 2.00 -waveform {0 1} [get_ports {clock_in}];
9
10
11 ##Switches
12 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { S1 }]; #IO_L24N_T3_RS0_15 Sch=sw
13 set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { S2 }]; #IO_L3N_T0_DQS_EMCCCLK_14
14 #set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { S2 }]; #IO_L6N_T0_D08_VREF_14 S
15 #set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 S
16 #set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 S
17 #set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch
18 #set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_1
  
```

Design Runs DRC Methodology Power **Timing**

#### Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): <b>-1,637 ns</b>	Worst Hold Slack (WHS): <b>0,189 ns</b>	Worst Pulse Width Slack (WPWS): <b>-0,155 ns</b>
Total Negative Slack (TNS): <b>-102,912 ns</b>	Total Hold Slack (THS): <b>0,000 ns</b>	Total Pulse Width Negative Slack (TPWS): <b>-0,155 ns</b>
Number of Failing Endpoints: <b>97</b>	Number of Failing Endpoints: <b>0</b>	Number of Failing Endpoints: <b>1</b>
Total Number of Endpoints: <b>104</b>	Total Number of Endpoints: <b>104</b>	Total Number of Endpoints: <b>43</b>

Timing constraints are not met.

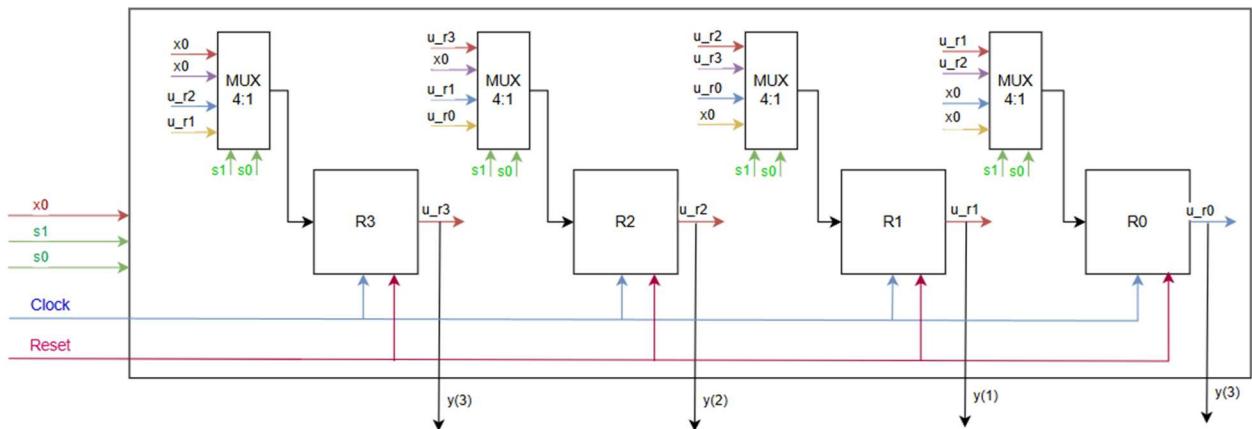
Vediamo che le specifiche di tempo richieste non sono state rispettate. Il worst negative slack, assume un valore negativo. Ciò significa che almeno un percorso critico non soddisfa i vincoli di temporizzazione, il segnale arriva troppo tardi per essere campionato correttamente.

## Esercizio 4: Shift Register

Progettare, implementare in VHDL e testare mediante simulazione un registro a scorrimento di N bit in grado di shiftare a destra o a sinistra di un numero Y variabile di posizioni a seconda di una opportuna selezione. In particolare, i valori possibili di Y sono 1 e 2. L'utente tramite selezione deve scegliere di quante posizioni shiftare. Il componente deve essere realizzato utilizzando sia un a) approccio comportamentale sia un b) approccio strutturale.

Nota: il numero di bit del registro deve essere implementato come un generic, e dall'esterno deve poter essere scelta la modalità di funzionamento mediante opportuni segnali di selezione.

### Progetto e Architettura



Per la progettazione dello shift register strutturale abbiamo utilizzato dei registri (appendice) per memorizzare i singoli bit e un MUX 4:1 (appendice) associato ad ogni flip-flop che in base alla selezione indicata effettua lo shift corretto, selezionando 00 andrà nel registro il valore proveniente dal registro di sinistra, con 01 ci sarà un doppio shift a destra, con 10 abbiamo uno shift a sinistra e con 11 un doppio shift a sinistra. Ovviamente per assicurarci il corretto funzionamento i registri agli estremi hanno bisogno di una configurazione particolare per la gestione degli  $x_0$  provenienti dall'esterno. Lo schema grafico mostra un progetto su 4 bit, ma il codice usa il costrutto generic che permette di usare anche più bit.

## Implementazione

L'implementazione behavioral di questo shift register si basa sull'utilizzo di un process che si attiva con il clock, sui cicli alti del clock si effettua lo shift.

### SHIFT REGISTER – COMPORTAMENTALE (*shift\_register.vhd*)

```
architecture Behavioral of shift_register is

    signal tmp: std_logic_vector(N-1 downto 0) := (others => '0');
    signal selezione: std_logic_vector(1 downto 0);
begin
    selezione(0) <= s0;
    selezione(1) <= s1;

    process(clock)
    begin
        if reset = '1' then
            tmp <= (others => '0'); -- Reset del registro
        elsif rising_edge(clock) then
            case selezione is
                when "00" => -- Shift a destra di 1
                    tmp(N-2 downto 0) <= tmp(N-1 downto 1);
                    tmp(N-1) <= x0;
                when "01" => -- Shift a destra di 2
                    tmp(N-3 downto 0) <= tmp(N-1 downto 2);
                    tmp(N-2) <= x0;
                    tmp(N-1) <= x0;
                when "10" => -- Shift a sinistra di 1
                    tmp(N-1 downto 1) <= tmp(N-2 downto 0);
                    tmp(0) <= x0;
                when "11" => -- Shift a sinistra di 2
                    tmp(N-1 downto 2) <= tmp(N-3 downto 0);
                    tmp(1) <= x0;
                    tmp(0) <= x0;
                when others =>
                    tmp <= (others => '0');
            end case;
        end if;
    end process;

    y <= tmp; -- Output del bit meno significativo
end Behavioral;
```

## SHIFT REGISTER – STRUTTURALE (*shift\_register.vhd*)

```

entity shift_register is
    Generic (n: positive:= 4);
    Port ( x0 : in STD_LOGIC;
            s0 : in STD_LOGIC;
            s1 : in STD_LOGIC;
            y : out STD_LOGIC_VECTOR (n-1 downto 0);
            clock : in STD_LOGIC;
            reset : in STD_LOGIC);
end shift_register;

```

```

architecture Structural of shift_register is

component registro is
    Port ( input : in STD_LOGIC;
            output : out STD_LOGIC;
            clock : in STD_LOGIC;
            reset : in STD_LOGIC);
end component;

component mux_4_1 is
    port( b0 : in STD_LOGIC;
          b1 : in STD_LOGIC;
          b2 : in STD_LOGIC;
          b3 : in STD_LOGIC;
          s0 : in STD_LOGIC;
          s1 : in STD_LOGIC;
          y0 : out STD_LOGIC
        );
end component;

signal temp: std_logic_vector(n-1 downto 0) := (others=>'0');
signal u_mux: std_logic_vector(n-1 downto 0) := (others=>'0');

```

```

begin
    ff_n_1: for i in 0 to n-1 generate
        ff_i: registro port map(
            input => u_mux(i),
            output => temp(i),
            clock => clock,
            reset => reset
        );
    end generate ff_n_1;

    mux_n_1: for i in 0 to n-1 generate
        IF_CLAUSE_1:if i = 0 generate
            mux_0: mux_4_1 port map(
                b0 => temp(1),
                b1 => temp(2),
                b2 => x0,
                b3 => x0,
                s0 => s0,
                s1 => s1,
                y0 => u_mux(i)
            );
        end generate IF_CLAUSE_1;

```

```

        IF_CLAUSE_2:if i = 1 generate
            mux_1: mux_4_1 port map(
                b0 => temp(i+1),
                b1 => temp(i+2),
                b2 => temp(i-1),
                b3 => x0,
                s0 => s0,
                s1 => s1,
                y0 => u_mux(i)
            );
        end generate IF_CLAUSE_2;

        IF_CLAUSE_PENULTIMO:if i = n-2 generate
            mux_n_2: mux_4_1 port map(
                b0 => temp(i+1),
                b1 => x0,
                b2 => temp(i-1),
                b3 => temp(i-2),
                s0 => s0,
                s1 => s1,
                y0 => u_mux(i)
            );
        end generate IF CLAUSE PENULTIMO;

```

```

IF_CLAUSE_ULTIMO:if i = N-1 generate
    mux_n_1: mux_4_1 port map(
        b0 => x0,
        b1 => x0,
        b2 => temp(i-1),
        b3 => temp(i-2),
        s0 => s0,
        s1 => s1,
        y0 => u_mux(i)
    );
end generate IF_CLAUSE_ULTIMO;

ELSE_CLAUSE :if i > 1 and i < n-2 generate
    mux_i: mux_4_1 port map(
        b0 => temp(i1),
        b1 => temp(i+2),
        b2 => temp(i-1),
        b3 => temp(i-2),
        s0 => s0,
        s1 => s1,
        y0 => u_mux(i)
    );
end generate ELSE_CLAUSE;
end generate mux_n_1;
y <= temp;
end Structural;

```

## Simulazione

```

----- r-----
begin

    reset <= '1';
    wait for 100 ns;
    reset <= '0';

    s0 <= '0';
    s1 <= '0';
    x0 <= '1';

    wait for 10 ns;
    x0 <= '0';
    wait for 10 ns;

    s0 <= '1';
    s1 <= '0';
    x0 <= '1';

    wait for 10 ns;
    x0 <= '1';
    wait for 10 ns;

    s0 <= '0';
    s1 <= '1';
    x0 <= '0';

    wait for 10 ns;

    s0 <= '1';
    s1 <= '1';
    x0 <= '0';

    wait for 10 ns;

    x0 <= '1';

    stop_the_clock <= true;
    wait;
end process;

```

Per la simulazione è stato usato un testbench con lo scopo di testare tutte e 4 le modalità disponibili.



Per l'implementazione behavioral abbiamo utilizzato un testbench differente per testare diverse successioni nella selezione delle modalità e nel valore del bit di ingresso.

```

stimulus: process
begin
    reset <= '1';
    wait for 100 ns;
    reset <= '0';

    s0 <= '1';
    s1 <= '0';
    x0 <= '1';

    wait for 10 ns;
    x0 <= '0';
    wait for 10 ns;

    s0 <= '1';
    s1 <= '1';
    x0 <= '1';

    wait for 10 ns;
    x0 <= '1';
    wait for 10 ns;

    s0 <= '0';
    s1 <= '0';
    x0 <= '0';

    wait for 10 ns;
    x0 <= '1';

    stop_the_clock <= true;
    wait;
end process;

```



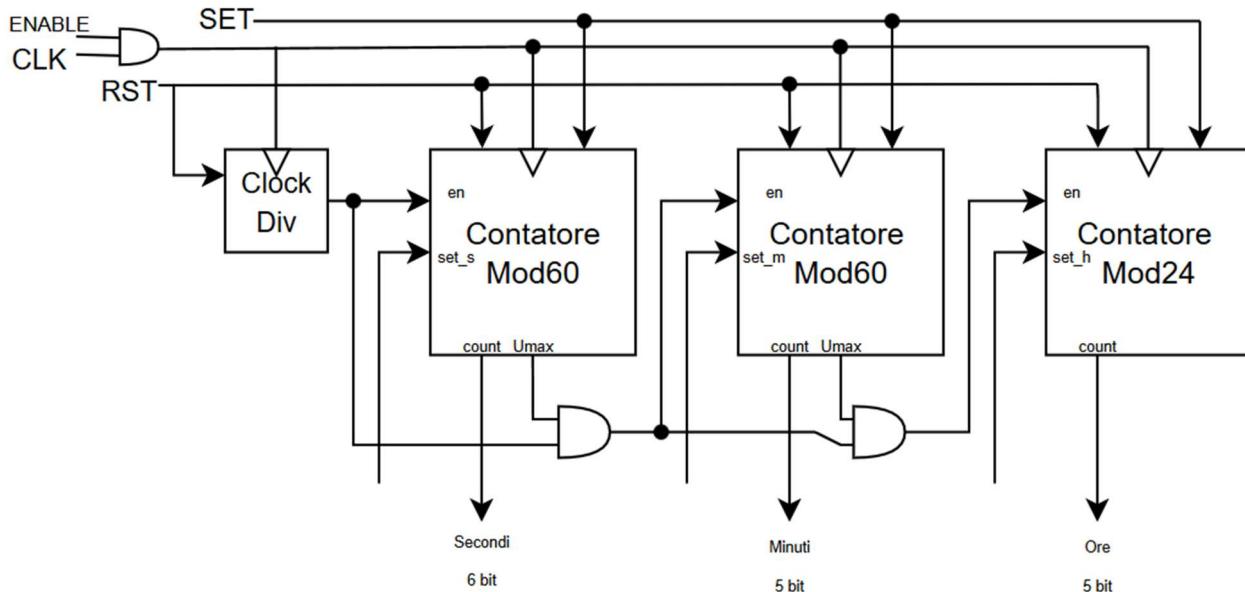
## Esercizio 5 : Cronometro

### Esercizio 5.1 :

Progettare, implementare in VHDL e testare mediante simulazione un cronometro, in grado di scandire secondi, minuti e ore a partire da una base dei tempi prefissata (es. si consideri il clock a disposizione sulla board). Il progetto deve prevedere la possibilità di inizializzare il cronometro con un valore iniziale, sempre espresso in termini di ore, minuti e secondi, mediante un opportuno ingresso di *set*, e deve prevedere un ingresso di *reset* per azzerare il tempo.

Il componente deve essere realizzato utilizzando un approccio strutturale, collegando opportunamente dei contatori secondo uno schema a scelta.

#### Progetto e Architettura



Si vuole realizzare un cronometro utilizzando un approccio strutturale: abbiamo utilizzato 3 contatori di cui due sono in modulo 60 per contare i secondi e i minuti e uno modulo 24 per contare le ore. L'architettura implementata per il cronometro è parallela, ciò significa che, a differenza di una architettura seriale, il clock è dato in ingresso a tutti i contatori che quindi saranno abilitati a contare solo nel momento in cui si presenta un fronte di salita del clock e risulta essere alta la Umax del contatore che lo precede. Per l'abilitazione del contatore dei secondi abbiamo utilizzato un clock divider che riduce la frequenza del clock della scheda in modo tale da avere un segnale di "enable" che ha un periodo di 1 s.

## Implementazione

### CRONOMETRO – STRUCTURAL (cronometro.vhd)

```
14 entity cronometro is
15     Port ( clock : in STD_LOGIC;
16             set : in STD_LOGIC;
17             reset : in STD_LOGIC;
18             set_value : in STD_LOGIC_VECTOR (16 downto 0);
19             h : out STD_LOGIC_VECTOR (4 downto 0);
20             m : out STD_LOGIC_VECTOR (5 downto 0);
21             s : out STD_LOGIC_VECTOR (5 downto 0);
22             enable : in STD_LOGIC);
23 end cronometro;
24
25 architecture Structural of cronometro is
26
27 component Divisore_freq is    generic (
28     f_in : integer := 100000000;
29     f_out : integer := 50000000
30 );
31     port (
32         reset : in std_logic;
33         clock : in std_logic;
34         div   : out std_logic
35     );
36 end component;
37
38 component cont_mod_24 is Port ( en : in STD_LOGIC;
39                                 clock : in STD_LOGIC;
40                                 set : in STD_LOGIC;
41                                 reset : in STD_LOGIC;
42                                 set_value : in STD_LOGIC_VECTOR (4 downto 0);
43                                 count : out STD_LOGIC_VECTOR (4 downto 0));
44 end component;
45
46 component cont_mod_60 is  Port ( en : in STD_LOGIC;
47                                 clock : in STD_LOGIC;
48                                 set : in STD_LOGIC;
49                                 reset : in STD_LOGIC;
50                                 set_value : in STD_LOGIC_VECTOR (5 downto 0);
51                                 count : out STD_LOGIC_VECTOR (5 downto 0);
52                                 U_max : out STD_LOGIC);
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
```

```
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
```

## DIVISORE DI FREQUENZA – BEHAVIORAL (*Divisore\_freq.vhd*)

```
34 □ entity Divisore_freq is
35     generic (
36         f_in : integer := 100000000;
37         f_out : integer := 1
38     );
39     port (
40         reset : in std_logic;
41         clock : in std_logic;
42         div : out std_logic
43     );
44 □ end Divisore_freq;
45
46 □ architecture Behavioral of Divisore_freq is
47
48     constant max_value : integer := (f_in/f_out)-1;
49
50     begin
51
52     process (clock)
53         variable conteggio : integer range 0 to max_value := 0;
54     begin
55
56         if (rising_edge(clock)) then
57             if (reset ='1') then
58                 conteggio := 0;
59                 div <= '0';
60             elsif (conteggio = max_value) then
61                 conteggio := 0;
62                 div <= '1';
63             else
64                 conteggio := conteggio+1;
65                 div <= '0';
66             end if;
67
68         end if;
69
70     end process;
71
72 □ end Behavioral.
```

## CONTATORE MODULO 60 – BEHAVIORAL (*cont\_mod\_60.vhd*)

```
14 □ entity cont_mod_60 is
15     Port ( en : in STD_LOGIC;
16             clock : in STD_LOGIC;
17             set : in STD_LOGIC;
18             reset : in STD_LOGIC;
19             set_value : in STD_LOGIC_VECTOR (5 downto 0);
20             count : out STD_LOGIC_VECTOR (5 downto 0);
21             U_max : out STD_LOGIC );
22 □ end cont_mod_60;
23
24 □ architecture Behavioral of cont_mod_60 is
25     signal counter: std_logic_vector(5 downto 0):= "000000";
26
27     begin
28         count <= counter;
29         --quando il contatore raggiunge il valore massimo (59), l'uscita U_max si deve alzare concomitamente
30         --ciò non deve essere fatto all'interno del process altrimenti ho ritardi dovuti alle assegnazioni nel process
31         --facendo ciò non appena arriva a 59 l'uscita si alza e viene vista dagli altri contatori
32         --che si aggiornano a loro volta.
33         U_max <= counter(5) and counter(4) and counter(3) and not counter(2) and counter(1) and counter(0);
34
35     process(clock)
36     begin
37         if rising_edge(clock)then
38
39             if reset = '1' then
40                 counter <= "000000";
41             elsif set = '1' then
42                 counter <= set_value;
43             elsif en = '1' then
44                 if counter = "111011" then
45                     counter<="000000";
46                 else
47                     counter <= std_logic_vector(unsigned(counter)+1);
48                 end if;
49             end if;
50
51         end if;
52
53     end if;
54
55     end process;
```

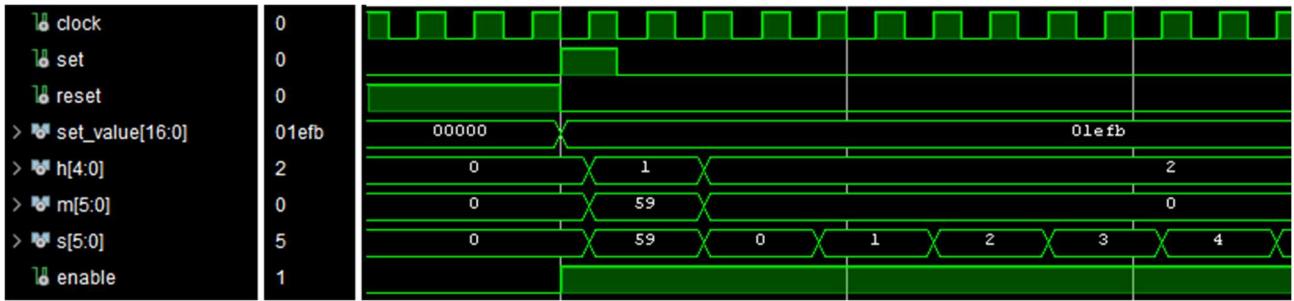
## CONTATORE MODULO 24 – BEHAVIORAL (*cont\_mod\_24.vhd*)

```
34 entity cont_mod_24 is
35     Port ( en : in STD_LOGIC;
36         clock : in STD_LOGIC;
37         set : in STD_LOGIC;
38         reset : in STD_LOGIC;
39         set_value : in STD_LOGIC_VECTOR (4 downto 0);
40         count : out STD_LOGIC_VECTOR (4 downto 0));
41 end cont_mod_24;
42
43 architecture Behavioral of cont_mod_24 is
44     signal counter: std_logic_vector (4 downto 0) := "00000";
45
46     begin
47         count <= counter;
48
49         process(clock)
50             begin
51                 if rising_edge(clock)then
52                     if reset = '1' then
53                         counter <= "00000";
54
55                     elsif set = '1' then
56                         counter <= set_value;
57                     elsif en = '1' then
58                         if counter = "10111" then
59                             counter<="00000";
60                         else
61                             counter <= std_logic_vector(unsigned(counter)+1);
62                         end if;
63                     end if;
64                 end if;
65             end process;
66
67         end Behavioral;
```

### Simulazione

Per il test di questo componente ci siamo posti in una situazione critica, abbiamo impostato il cronometro con i seguenti valori: 1 ora, 59 minuti, 59 secondi. In modo da mostrare l’incremento simultaneo dei secondi, delle ore e dei minuti.

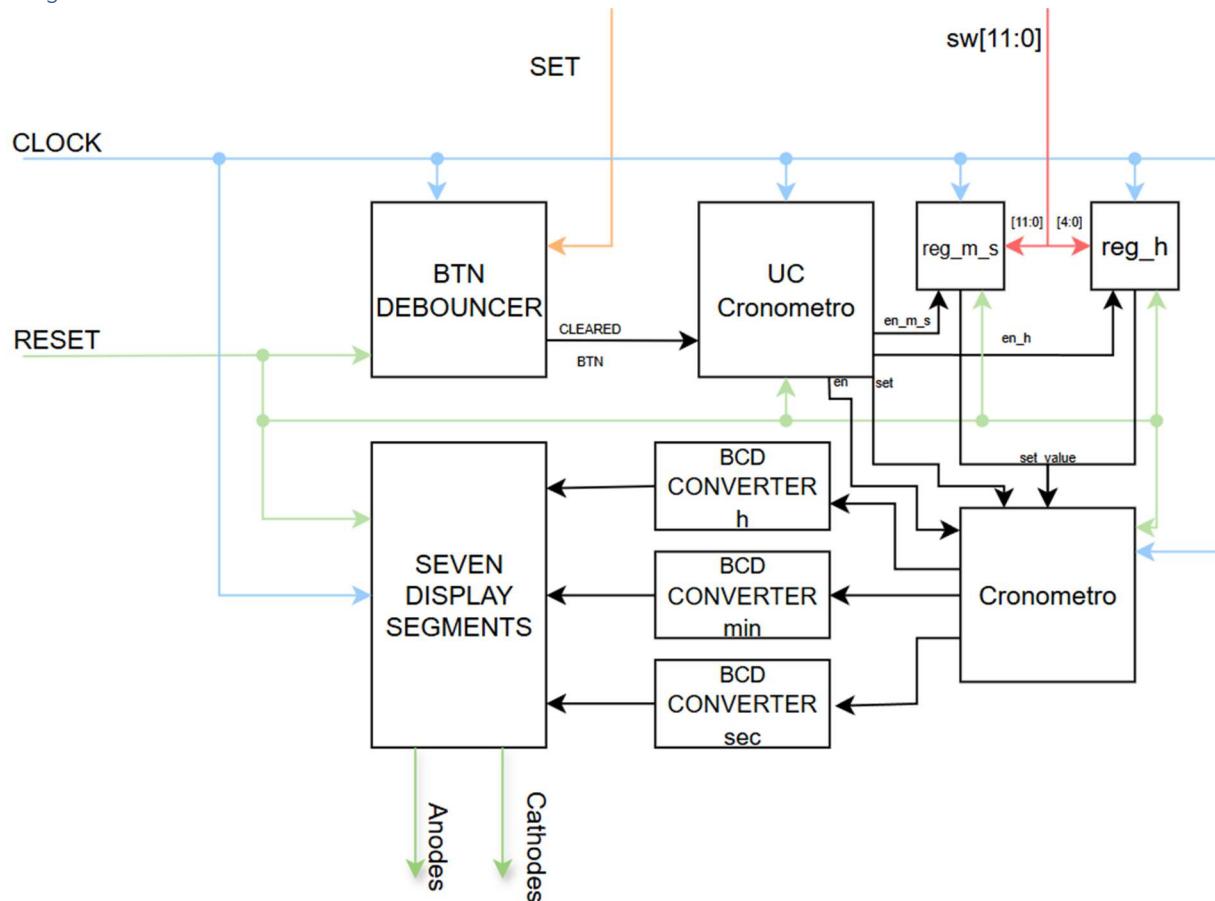
```
51 stimuli : process
52     begin
53         -- EDIT Adapt initialization as needed
54         set <= '0';
55         set_value <= (others => '0');
56         enable <= '0';
57         reset <= '1';
58         wait for 100 ns;
59         reset <= '0';
60         enable<= '1';
61         set_value <= "000011101111011";
62         --h= 01
63         --min = 59
64         --sec = 59
65         set<= '1';
66         wait for 10 ns;
67         set <= '0';
68
69         --10 secondi
70         wait for 10000 ns;
71         enable <= '0';
72
73         wait;
74     end process;
75
76 end tb;
```



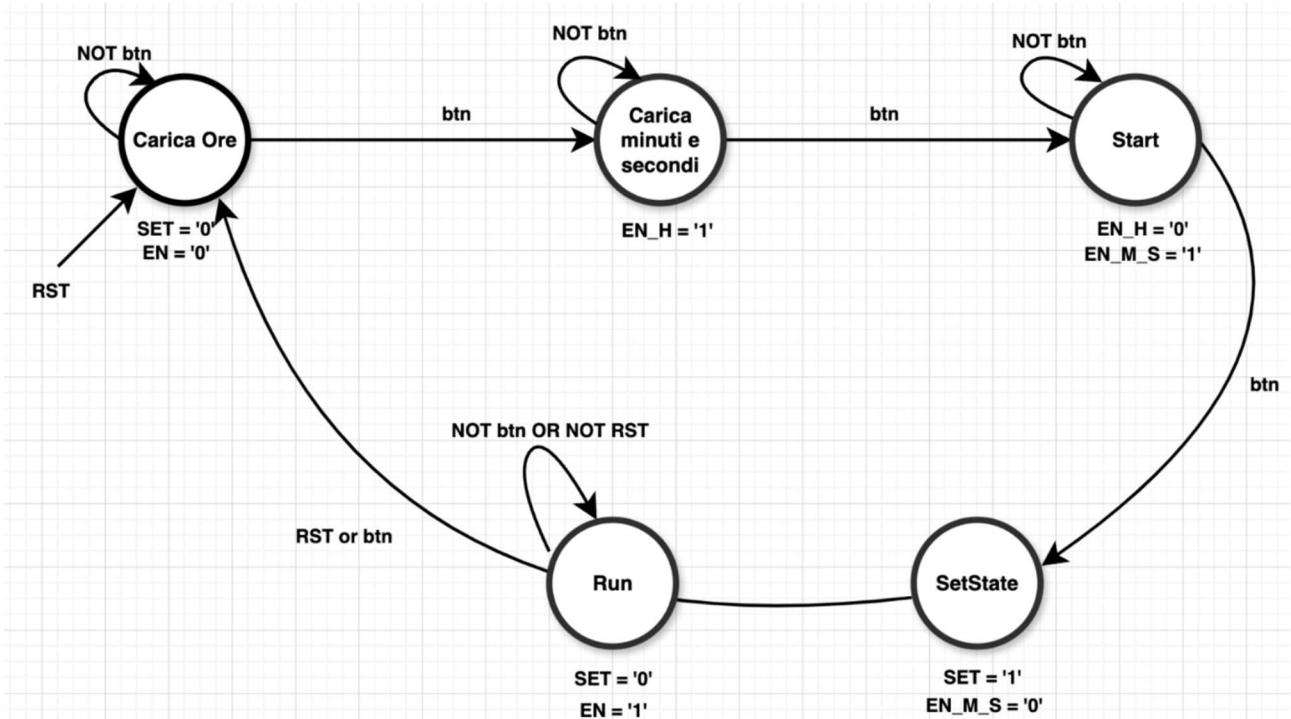
### Esercizio 5.2 :

Sintetizzare ed implementare su board il componente sviluppato al punto precedente, utilizzando i display a 7 segmenti per la visualizzazione dell'orario (o una combinazione di display e led nel caso in cui i display a disposizione siano in numero inferiore a quello necessario), gli switch per l'immissione dell'orario iniziale e due bottoni, uno per il set dell'orario e uno per il reset. Si utilizzi una codifica a scelta dello studente per la visualizzazione dell'orario sui display (esadecimale o decimale).

#### Progetto e Architettura



Si procede a descrivere il funzionamento del componente sviluppato su board tramite automa specifico. Il segnale "btn" è il segnale che viene dato in ingresso all'unità di controllo pulito tramite il componente "ButtonDebouncer". Il tasto fisico a cui corrisponde il segnale "btn", ossia il set, è il tasto N17 della board, mentre al segnale di RST, anch'esso opportunamente pulito tramite ButtonDebouncer corrisponde al tasto fisico CPU RESET.



La macchina alla partenza si trova nello stato "Carica Ore" dove si arriva anche premendo il bottone di RESET. Il cronometro inizialmente non è settato, quindi sul display troveremo tutti '0'. Posso caricare il valore binario delle ore attraverso i primi 5 switch a partire da destra (da SW0 a SW4). Premendo il bottone di set mi porto nello stato "Carica Minuti Secondi" in cui verrà alzato l'enable del registro che manterrà stabile il valore binario delle ore e qui posso settare tramite i primi 12 switch a partire da destra (SW0 a SW11) il valore dei secondi (SW0 a SW5) e dei minuti (SW6 a SW11). Premendo ancora il bottone di set vado nel successivo stato chiamato "Start" in cui abilito il registro per mantenere stabile il valore in binario di minuti e secondi; infine con un'ultima pressione del bottone di set arrivo nello stato "SET" in cui carico il preset nel contatore alzando il segnale di SET. Infine da questo stato la macchina al colpo di clock successivo si sposta subito nello stato "Run" che funge da stato di IDLE per il cronometro che va avanti a contare finchè non premo il bottone di RESET oppure il bottone di SET: nel primo caso la macchina torna nello stato "Carica Ore" in cui il cronometro si sarà azzerato, dal momento che il reset è il reset master e quindi coincide con il reset del cronometro mentre nel secondo caso torno sempre in "Carica Ore" ma il cronometro non risulterà azzerato, simulando il funzionamento di un vero cronometro che una volta stoppato mostra il tempo cronometrato. Posso quindi procedere a caricare nuovi valori delle ore, dei minuti e dei secondi come descritto in precedenza e questi saranno sovrascritti a quelli precedenti facendo ripartire la macchina col nuovo valore di preset.

## Implementazione

### CRONOMETRO\_ON\_BOARD – STRUCTURAL (*cronometro\_on\_board.vhd*)

```
--> -----
35     Port ( clock_in : in STD_LOGIC;
36             reset_in : in STD_LOGIC;
37             anodes_out : out STD_LOGIC_VECTOR (7 downto 0);
38             cathodes_out : out STD_LOGIC_VECTOR (7 downto 0);
39             set : in STD_LOGIC;
40             sv : in STD_LOGIC_VECTOR (11 downto 0));
41 end cronometro_on_board;
42
43 architecture Behavioral of cronometro_on_board is
44 component registro is generic ( N : integer:=8);
45     port( A: in std_logic_vector(N-1 downto 0);
46            clk, res, load: in std_logic;
47            B: out std_logic_vector(N-1 downto 0));
48 end component;
49 component display_seven_segments is
50     Generic(
51             CLKIN_freq : integer := 100000000;
52             CLKOUT_freq : integer := 500
53         );
54     Port ( CLK : in STD_LOGIC;
55             RST : in STD_LOGIC;
56             VALUE : in STD_LOGIC_VECTOR (31 downto 0);
57             ENABLE : in STD_LOGIC_VECTOR (7 downto 0); -- decide quali cifre abilitare
58             DOTS : in STD_LOGIC_VECTOR (7 downto 0); -- decide quali punti visualizzare
59             ANODES : out STD_LOGIC_VECTOR (7 downto 0);
60             CATHODES : out STD_LOGIC_VECTOR (7 downto 0));
61 end component;
62 component cronometro is
63     Port ( clock : in STD_LOGIC;
64             set : in STD_LOGIC;
65             reset : in STD_LOGIC;
66             set_value : in STD_LOGIC_VECTOR (16 downto 0);
67             h : out STD_LOGIC_VECTOR (4 downto 0);
68             m : out STD_LOGIC_VECTOR (5 downto 0);
69             s : out STD_LOGIC_VECTOR (5 downto 0);
70             enable : in STD_LOGIC);
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
```

```
component ButtonDebouncer is
    generic (
        CLK_period: integer := 10; -- periodo del clock (della board) in nanosecondi
        btn_noise_time: integer := 10000000 -- durata stimata dell'oscillazione del bottone in nanosecondi
                                         -- il valore di default è 10 millisecondi
    );
    Port ( RST : in STD_LOGIC;
           CLK : in STD_LOGIC;
           BTN : in STD_LOGIC;
           CLEARED_BTN : out STD_LOGIC);
end component;
component CU_set_cronom is
    Port ( CLK : in STD_LOGIC;
           RESET : in STD_LOGIC;
           BTN : in STD_LOGIC;
           EN: out STD_LOGIC;
           SET : out STD_LOGIC;
           en_h : out std_logic;
           en_m_s : out STD_LOGIC );
end component;
```

```

99 -- segnale tra registri e cronometro che memorizzano il valore
100 --delle ore, minuti e secondi da settare
101 signal set_value :std_logic_vector(16 downto 0);
102 --segnali abilitazioni registri ore e minuti-secondi
103 signal en_h:std_logic;
104 signal en_m_s:std_logic;
105 --segnale del bottone cleared (dopo il debouncer)
106 signal btn:std_logic;
107 --segnale che esce dalla CU e abilita il SET del contatore
108 signal set_CU:std_logic;
109 --segnale di reset collegato al bottone della board(ricorda 0 attivo)
110 signal reset:std_logic;
111 --segnale di abilitazione del contatore
112 signal en:std_logic;
113 -- segnale per le ore in uscita dal contatore
114 signal h :STD_LOGIC_VECTOR (4 downto 0);
115 -- segnale per i minuti in uscita dal contatore
116 signal m :STD_LOGIC_VECTOR (5 downto 0);
117 -- segnale per i secondi in uscita dal contatore
118 signal s :STD_LOGIC_VECTOR (5 downto 0);
119
120 --segnale per le ore convertito su 8 bit ( 4 bit per le unità e 4 per le decine)
121 signal h_x :STD_LOGIC_VECTOR (7 downto 0);
122 --segnale delle ore convertito su 6 bit ,
123 -- poichè essendo le ore su 5 bit le convertiamo su 6
124 --mettendo uno 0 di padding all'inizio in modo tale da poter utilizzare lo stesso convertitore
125 --(che accetta 6 bit in ingresso) sia per le ore che per i secondi/minuti
126 signal h_6:STD_LOGIC_VECTOR(5 downto 0);
127 --segnale per i minuti convertito su 8 bit ( 4 bit per le unità e 4 per le decine)
128 signal m_x :STD_LOGIC_VECTOR (7 downto 0);
129 --segnale per i minuti convertito su 8 bit ( 4 bit per le unità e 4 per le decine)
130 signal s_x :STD_LOGIC_VECTOR (7 downto 0);
131 --segnale dei valori da mostrare sul display(concatenazione ore min sec)
132 signal value_display : STD LOGIC VECTOR(31 downto 0);

```

```

133 begin
134
135 reset <= not reset_in;
136
137 --devo convertire secondi, min, h per visualizzarli sul display che accetta dati in esadecimale
138 converti_sec: bcd_converter port map(s,s_x);
139 converti_min:bcd_converter port map(m,m_x);
140 --devo convertire le or e un 6bit per utilizzare convertitore
141 h_6 <= '0' & h;
142 converti_hexbcd_converter port map(h_6,h_x);
143 value_display <= "00000000" & h_x & m_x & s_x;
144 --DISPLAY
145 dds: display_seven_segments port map(clock_in,reset,value_display,"11111111","00000000",anodes_out,cathodes_out);
146 --CRONOMETRO
147 crono: cronometro port map(clock_in,set_CU,reset,set_value,h,m,s,en);
148 --BTN DEBOUNCER
149 btn_dbc: ButtonDebouncer port map(reset, clock_in, set,btn);
150 --CU
151 cu: CU_set_cronom port map(clock_in,reset,btn,en,set_CU,en_h,en_m_s);
152 --Registro per le ore 5 bit
153 reg_h: registro generic map(5) port map(sw(4 downto 0),clock_in,reset,en_h,set_value(16 downto 12));
154 --Registro per i minuti e per i secondi 12 bit ( 6+6 bit)
155 reg_m_s: registro generic map(12) port map(sw(11 downto 0),clock_in,reset,en_m_s,set_value(11 downto 0));
156 end Behavioral;

```

## BCD\_CONVERTER– BEHAVIORAL (*bcd\_converter.vhd*)

Siccome in uscita dal cronometro i secondi e i minuti sono rappresentati su 6 bit e le ore su 5 bit, per visualizzarli sul display, che accetta per ogni cifra 4 bit (in esadecimale), abbiamo bisogno di un componente che effettui questa conversione.

Abbiamo quindi creato un componente unico per secondi, minuti e ore che accetta in ingresso uno std\_logic\_vector di 6 bit e restituisce uno std\_logic\_vector di 8 bit di cui i 4 più significativi per le decine ed i restanti 4 per le unità. Siccome in ingresso vuole uno std\_logic\_vector di 6 bit e le ore sono codificate su 5 bit dobbiamo aggiungere uno 0 di padding ( riga 141 del codice precedente).

```

14  library IEEE;
15  use IEEE.STD_LOGIC_1164.ALL;
16  use IEEE.NUMERIC_STD.ALL;
17
18  entity bcd_converter is
19      Port ( sec_6bit : in STD_LOGIC_VECTOR(5 downto 0); -- Secondi (0-59)
20            output      : out STD_LOGIC_VECTOR(7 downto 0)
21      );
22  end bcd_converter;
23
24  architecture Behavioral of bcd_converter is
25  begin
26      process(sec_6bit)
27          variable sec_int : integer range 0 to 59;
28          variable dec_int : integer range 0 to 5;
29          variable unit_int : integer range 0 to 9;
30      begin
31          -- Conversione da std_logic_vector a intero
32          sec_int := to_integer(unsigned(sec_6bit));
33
34          -- Calcolo decine e unità
35          dec_int := sec_int / 10; -- Parte intera della divisione
36          unit_int := sec_int mod 10; -- Resto della divisione
37
38          -- Conversione in std_logic_vector (BCD)
39          output <= std_logic_vector(to_unsigned(dec_int, 4)) & std_logic_vector(to_unsigned(unit_int, 4));
40      end process;
41  end Behavioral;

```

## CU\_set\_CRONOM– BEHAVIORAL (CU\_set\_cronom.vhd)

```

34  entity CU_set_cronom is
35      Port ( CLK : in STD_LOGIC;
36              RESET : in STD_LOGIC;
37              BTN : in STD_LOGIC;
38              EN: out STD_LOGIC;
39              SET : out STD_LOGIC;
40              en_h : out STD_LOGIC;
41              en_m_s : out STD_LOGIC );
42  end CU_set_cronom;
43
44  architecture Behavioral of CU_set_cronom is
45  type stato is (CaricaOre,CaricaMinutiSecondi,Start,SetState,Run);
46  signal stato_corrente:stato := CaricaOre;
47  begin
48
49
50
51  process(clk)
52      begin
53
54      if rising_edge(clk) then
55          if reset = '1' then
56              set <='0'; --segna che setta il cronometro
57              EN <= '1'; --segna che abilita cronometro
58              en_h<= '0'; --segna che abilita registro dove salvare valore degli switch
59              en_m_s<= '0';
60              stato_corrente <= CaricaOre;
61          else
62              case stato_corrente is
63                  when CaricaOre =>
64                      EN <= '0';
65                      set <= '0';
66                      en_h<= '0';
67                      en_m_s<= '0';
68                  if btn = '0' then
69                      stato_corrente <= CaricaOre;
70
71                  else
72                      stato_corrente <= CaricaMinutiSecondi;
73

```

```

74 when CaricaMinutiSecondi=>
75     EN <= '0';
76     set <= '0';
77     en_h<= '1';
78     en_m_s<= '0';
79     if btn = '0' then
80         stato_corrente <= CaricaMinutiSecondi;
81
82     else
83         stato_corrente <= Start;
84     end if;
85 when Start=>
86     EN <= '0';
87     set <= '0';
88     en_h<= '0';
89     en_m_s<= '1';
90
91     if btn = '0' then
92         stato_corrente <= Start;
93
94     else
95         stato_corrente <= SetState;
96     end if;
97 when SetState =>
98     EN <= '0';
99     set <= '1';
100    en_h<= '0';
101    en_m_s<= '0';
102    stato_corrente <= Run;
103 when Run =>
104     EN <= '1';
105     set <= '0';
106     en_h<= '0';
107     en_m_s<= '0';
108     if btn = '1' or reset = '1' then
109         stato_corrente <= CaricaOre;
110     else
111         stato_corrente <= Run;
112     end if;
113
114 end case;
115 end if;
end if;

```

## CONSTRAINTS

```

1: set_property -dict { PACKAGE_PIN Z0 IOSTANDARD LVCMOS33 } [get_ports { CLOCK_IN }]; #IO_Z0_I2C_BB_M00_05 SCH=M1K10VHIZ
2: create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clock_in}];
3:
4:
5: ##$Switches
6: set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
7: set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { sw[1] }]; #IO_L3N_T0_D05_EMCCLK_14 Sch=sv[1]
8: set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
9: set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
10: set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { sw[4] }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
11: set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { sw[5] }]; #IO_L7N_T1_D10_14 Sch=sv[5]
12: set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { sw[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sv[6]
13: set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { sw[7] }]; #IO_L5N_T0_D07_14 Sch=sv[7]
14: set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { sw[8] }]; #IO_L24N_T3_34 Sch=sv[8]
15: set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { sw[9] }]; #IO_25_34 Sch=sv[9]
16: set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { sw[10] }]; #IO_L15P_T2_D05_RDWR_B_14 Sch=sv[10]
17: set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { sw[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sv[11]

```

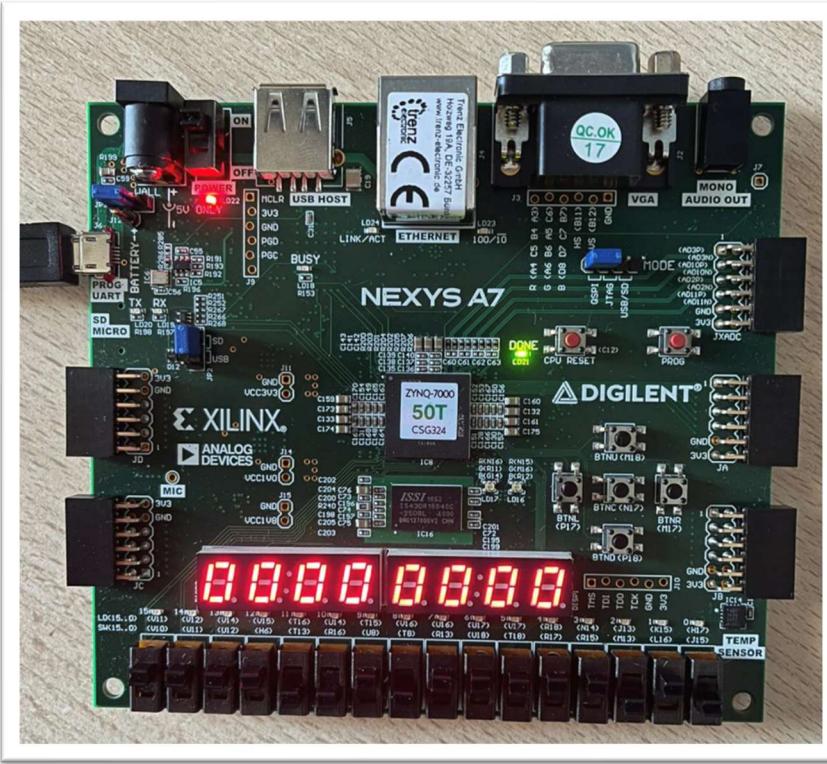
```

55: ##/ Segment display
56: set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { cathodes_out[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
57: set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { cathodes_out[1] }]; #IO_25_14 Sch=cb
58: set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { cathodes_out[2] }]; #IO_25_15 Sch=cc
59: set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { cathodes_out[3] }]; #IO_L17P_T2_A26_15 Sch=cd
60: set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { cathodes_out[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
61: set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { cathodes_out[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
62: set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 } [get_ports { cathodes_out[6] }]; #IO_L4P_T0_D04_14 Sch=cg
63: set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 } [get_ports { cathodes_out[7] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
64: set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { anodes_out[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
65: set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 } [get_ports { anodes_out[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
66: set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { anodes_out[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
67: set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { anodes_out[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
68: set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { anodes_out[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
69: set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { anodes_out[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
70: set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { anodes_out[6] }]; #IO_L23P_T3_35 Sch=an[6]
71: set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { anodes_out[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
72:
73: ##Buttons
74: set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVCMOS33 } [get_ports { reset_in }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
75: set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports { ear_11 }]; #IO_T0D_T1_DNC_12 Sch=button

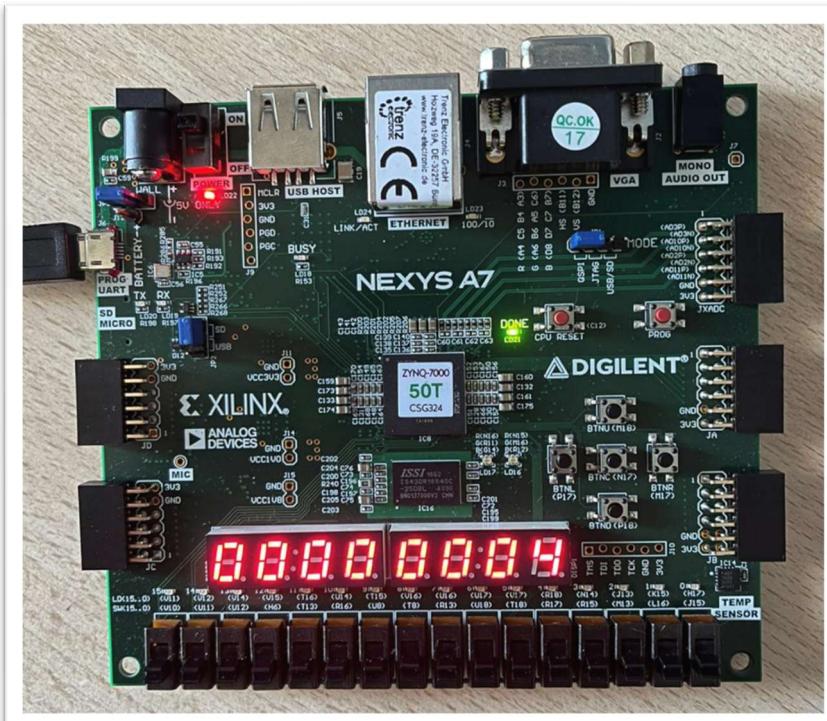
```

## Sintesi su board di sviluppo

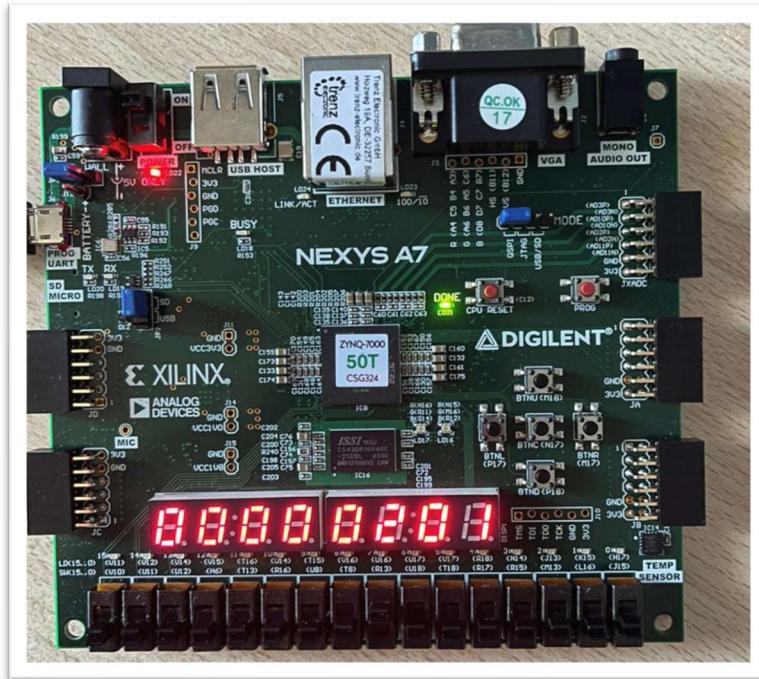
Il cronometro parte azzerato, stato in cui si trova inizialmente o in cui arriva dopo essere stato resettato:



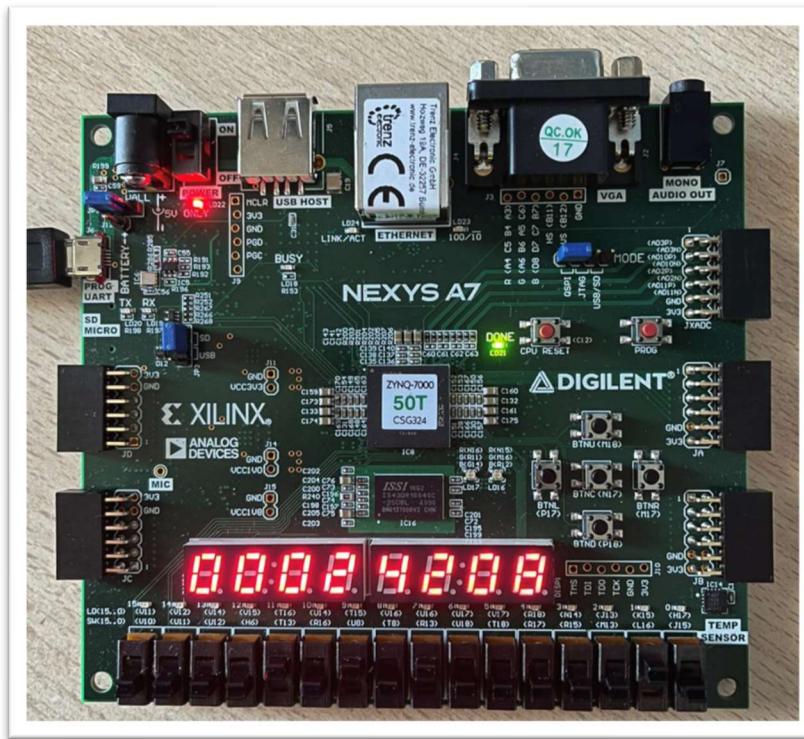
Tramite switch carichiamo il valore dei secondi, dei minuti e delle ore: carichiamo 0 minuti e 0 ore per evidenziare come i secondi vengano mostrati sui due display più a destra:



Premendo il bottone di set come descritto precedentemente il cronometro si ferma e possiamo caricare i minuti e i secondi e ancora una volta 0 ore; i minuti saranno mostrati sui 2 display a sinistra di quelli dei secondi.



Ora ripetiamo l'operazione caricando anche le ore diverse da 0 che saranno mostrate sui due display a sinistra di quelli dei minuti.



Da notare che sul display, grazie al componente convertitore che abbiamo implementato, il cui codice è riportato sopra, le cifre di ore, minuti e secondi vengono riportate in decimale sui display.

## Esercizio 6 : Sistema di lettura-elaborazione-scrittura PO\_PC

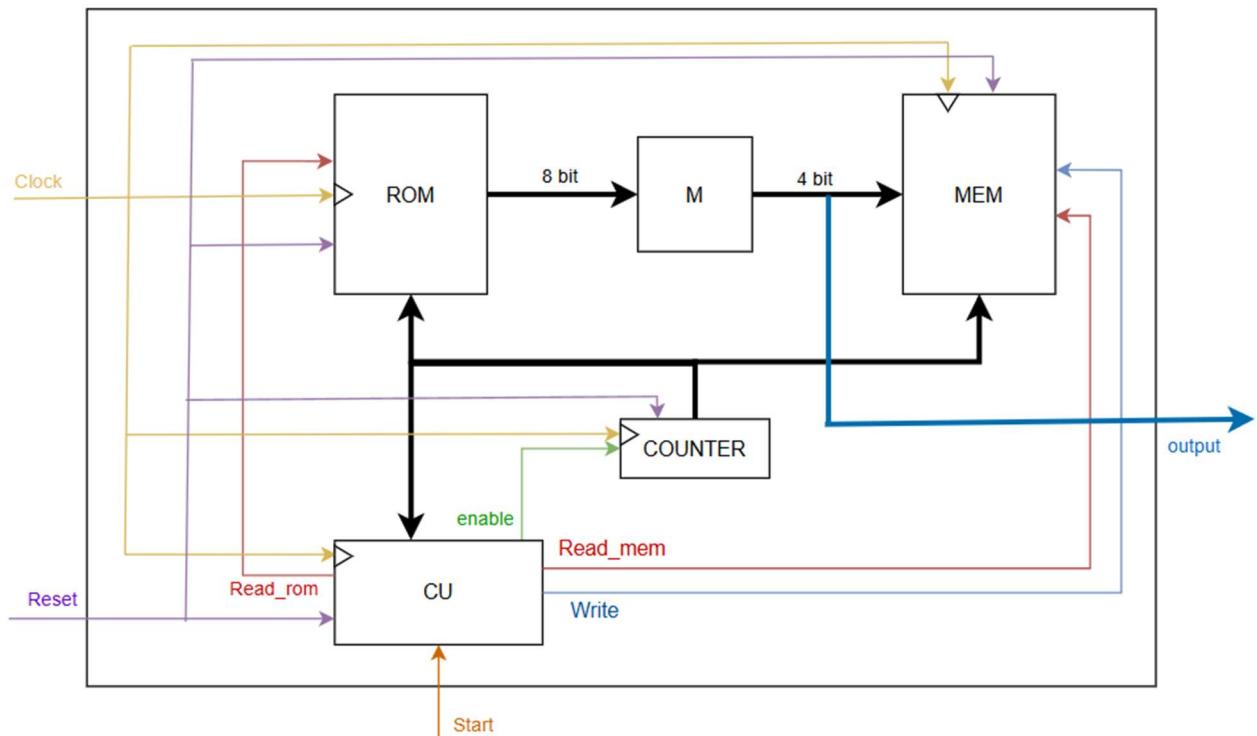
### Esercizio 6.1 :

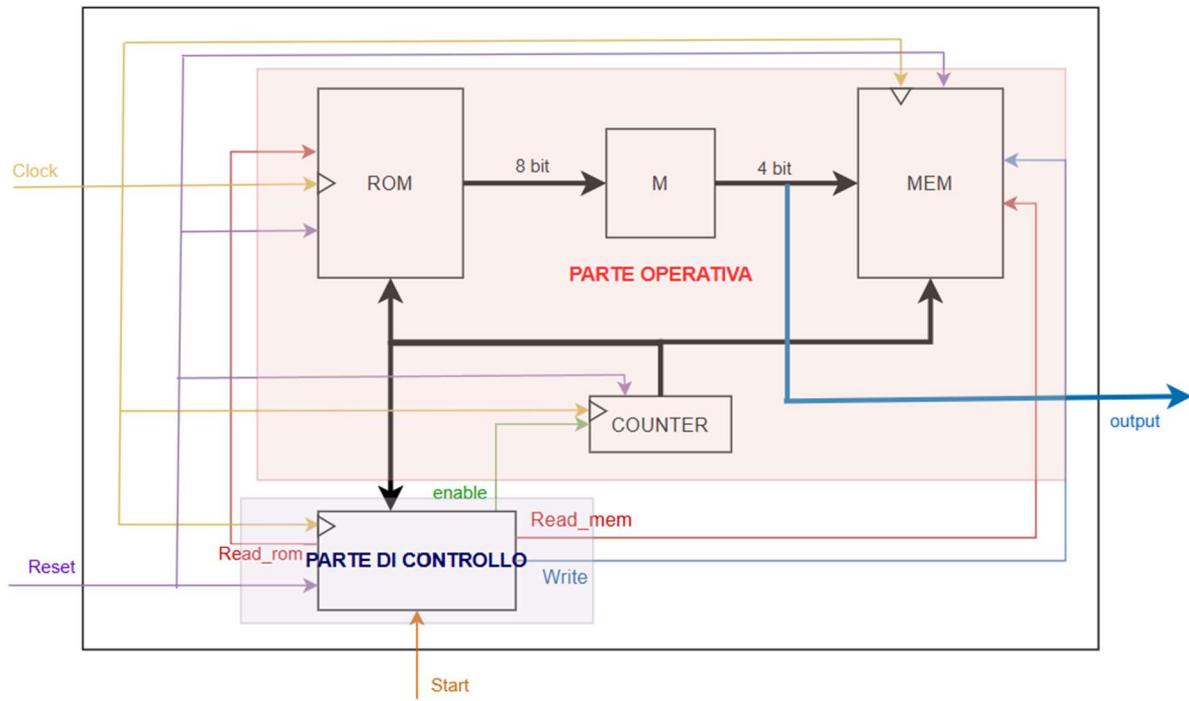
Progettare, implementare in VHDL e verificare mediante simulazione un sistema dotato di una memoria ROM di N locazioni da 8 bit ciascuna, una macchina combinatoria M in grado di trasformare (secondo una funzione a scelta dello studente) la stringa di 8 bit letta dalla ROM in una stringa di 4 bit, e una memoria MEM di N locazioni che memorizza la stringa in output da M.

Il sistema si avvia in corrispondenza di un segnale di START che viene fornito esternamente. Una volta avviato, tramite un'apposita unità di controllo che gestisce la temporizzazione del sistema, viene scandita una locazione alla volta della ROM e viene scritta la corrispondente locazione di MEM. Gli indirizzi di memoria sono forniti da un contatore. Le memorie ROM e MEM hanno rispettivamente un read e un write sincrono.

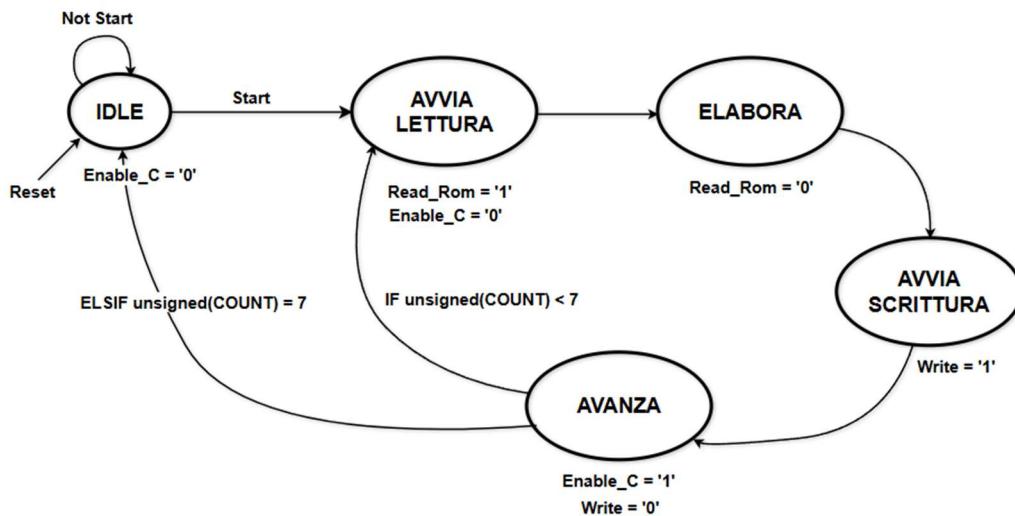
#### Progetto e Architettura

Riportiamo il progetto complessivo del sistema, inoltre si riporta anche uno schema in cui appare evidente la divisione tra parte operativa e parte di controllo.





L'automa dell'unità di controllo è il seguente.



Il sistema preleva un elemento dalla ROM, dopodiché aspetta che la macchina combinatoria termini l'elaborazione in presenza di eventuali ritardi, una volta ottenuto l'output avvia la scrittura sulla memoria. Al termine della scrittura abilita il conteggio del contatore, se abbiamo terminato il conteggio allora si ritorna allo stato di idle, altrimenti si procede leggendo la locazione di memoria successiva.

### *Implementazione*

Si riporta solo l'implementazione del sistema complessivo, della macchina M e dell'unità di controllo poichè l'implementazione della ROM, della memoria e del contatore si rifanno ai componenti base presenti in appendice.

### **SISTEMA LES – STRUCTURAL (*sistema\_les.vhd*)**

```
entity sistema_les is
    Port ( clock : in STD_LOGIC;
           start : in STD_LOGIC;
           reset : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (3 downto 0));
end sistema_les;

architecture Structural of sistema_les is

component rom_n is
    Generic (n : positive := 2; -- numero di bit necessari all'indirizzamento -1
             l: positive := 7; -- numero di locazioni - 1
             m: positive := 7); -- numero di bit per locazione -1
    Port ( read : in STD_LOGIC;
           clock : in STD_LOGIC;
           reset: in STD_LOGIC;
           address : in STD_LOGIC_VECTOR (n downto 0);
           output : out STD_LOGIC_VECTOR (m downto 0));
end component;

component memoria_n is
    Generic (n : positive := 2; -- numero di bit necessari all'indirizzamento -1
             l: positive := 7; -- numero di locazioni - 1
             m: positive := 7); -- numero di bit per locazione -1
    Port ( read : in STD_LOGIC;
           write : in STD_LOGIC;
           clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           input : in STD_LOGIC_VECTOR (m downto 0);
           output : out STD_LOGIC_VECTOR (m downto 0);
           address : in STD_LOGIC_VECTOR (n downto 0));
end component;
```

```

component macchina is
  Port (
    input : in STD_LOGIC_VECTOR (7 downto 0);
    output : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component uc is
  Port ( start : in STD_LOGIC;
         clock : in STD_LOGIC;
         reset : in STD_LOGIC;
         count : in STD_LOGIC_VECTOR(2 downto 0);
         read_rom : out STD_LOGIC;
         read_mem : out STD_LOGIC;
         write : out STD_LOGIC;
         enable_counter : out STD_LOGIC);
end component;

signal sig_read : std_logic;
signal sig_read2 : std_logic;
signal sig_write : std_logic;
signal sig_en : std_logic;
signal sig_address : std_logic_vector(2 downto 0);
signal sig_input : std_logic_vector(7 downto 0);
signal sig_output : std_logic_vector(3 downto 0);
signal sig_output_mem : std_logic_vector(3 downto 0);

```

```

begin
rom : rom_n Generic map (n => 2, l=> 7, m => 7)
  Port map ( read => sig_read,
             clock => clock,
             reset => reset,
             address => sig_address,
             output => sig_input );
mem : memoria_n
  Generic map( n=> 2, l=> 7, m=> 3)
  Port map (
    read => sig_read2,
    write => sig_write,
    clock => clock,
    reset => reset,
    input => sig_output,
    output => sig_output_mem,
    address  => sig_address);
counter : counter_n
  Generic map (n => 2)
  Port map( clock => clock,
             reset => reset,
             enable => sig_en,
             count => sig_address);
M :  macchina
  Port map(
    input => sig_input,
    output => sig_output);

control_unit: uc
  Port map ( start => start,
             clock => clock,
             reset => reset,
             count => sig_address,
             read_rom => sig_read,
             read_mem => sig_read2,
             write => sig_write,
             enable_counter => sig_en);

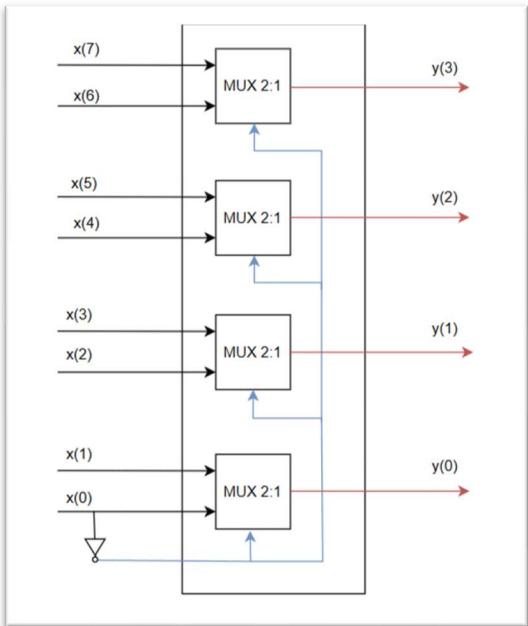
output <= sig_output;
end Structural;

```

## MACCHINA – STRUCTURAL (*macchina.vhd*)

```
entity macchina is
    Port (
        input : in STD_LOGIC_VECTOR (7 downto 0);
        output : out STD_LOGIC_VECTOR (3 downto 0));
end macchina;

architecture Dataflow of macchina is
begin
    begin
        output(0) <= input(0) when input(0) = '0' else
            input(1) when input(0) = '1'
            else '-';
        output(1) <= input(2) when input(0) = '0' else
            input(3) when input(0) = '1'
            else '-';
        output(2) <= input(4) when input(0) = '0' else
            input(5) when input(0) = '1'
            else '-';
        output(3) <= input(6) when input(0) = '0' else
            input(7) when input(0) = '1'
            else '-';
    end;
end Dataflow;
```



Se l'LSB è 0 mette in uscita i bit dell'ingresso che sono in posizione pari, altrimenti quelli in posizione dispari.

## UC – BEHAVIORAL (uc.vhd)

```
entity uc is
    Port ( start : in STD_LOGIC;
            clock : in STD_LOGIC;
            reset : in STD_LOGIC;
            count : in STD_LOGIC_VECTOR(2 downto 0);
            read_rom : out STD_LOGIC;
            read_mem : out STD_LOGIC;
            write : out STD_LOGIC;
            enable_counter : out STD_LOGIC);
end uc;

architecture Behavioral of uc is

type stato is (idle, avvia_lettura, lettura_effettuata, avvia_scrittura, scrittura_effettuata);

signal stato_corrente : stato := idle;
```

```
process(clock)
begin

if rising_edge(clock) then

    if reset = '1' then
        stato_corrente <= idle;
    else

        case stato_corrente is

            when idle =>
                enable_counter <= '0';
                if start = '1' then
                    stato_corrente <= avvia_lettura;
                else
                    stato_corrente <= idle;
                end if;

            when avvia_lettura =>
                read_rom <= '1';
                enable_counter <= '0';
                stato_corrente <= lettura_effettuata;

            when lettura_effettuata =>
                read_rom <= '0';
                stato_corrente <= avvia_scrittura;
```

```
when avvia_scrittura =>
    write <= '1';
    stato_corrente <= scrittura_effettuata;

when scrittura_effettuata =>
    write <= '0';
    enable_counter <= '1';
    stato_corrente <= idle;
    if unsigned(count)<7 then
        stato_corrente <= avvia_lettura;
    else
        stato_corrente <= idle;
    end if;
end case;
end if;

end process;
end Behavioral;
```

## Simulazione

Per quanto riguarda la simulazione gli stimoli si sono limitati a dare prima il segnale di reset e poi quello di start.

```
stimulus: process
begin

    -- Put initialisation code here
    reset <= '1';
    start <= '0';

    wait for 100 ns;

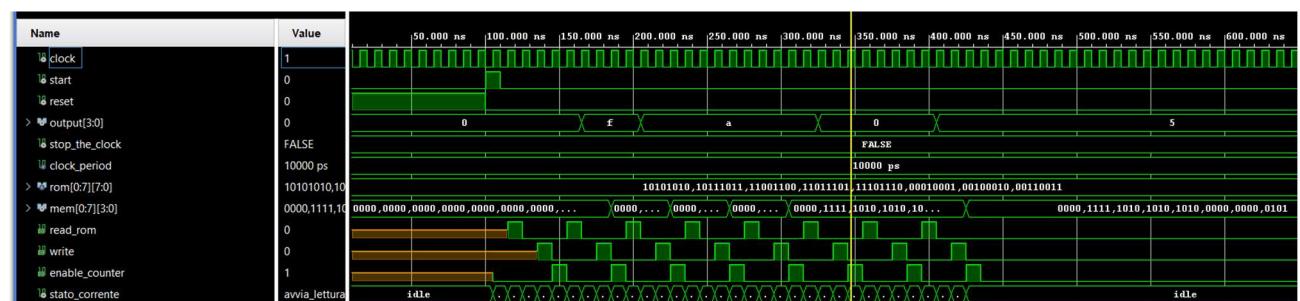
    reset <= '0';
    start <= '1';
    wait for 10 ns;
    start <= '0';

    wait for 1000 ns;

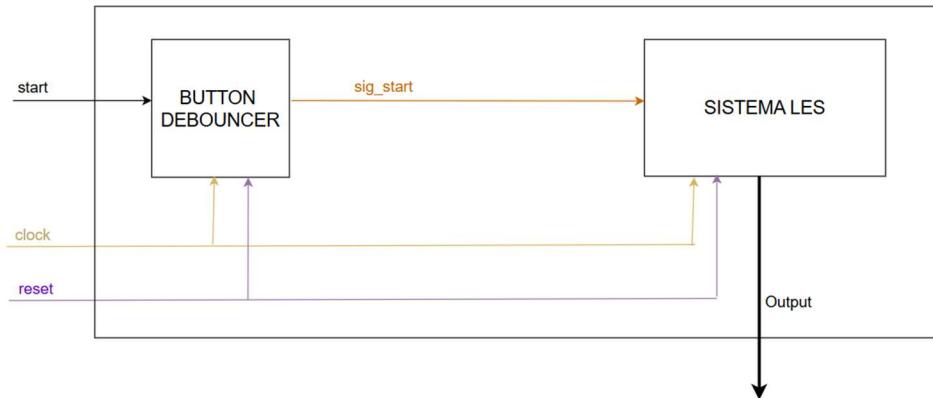
    -- Put test bench stimulus code here
    stop_the_clock <= true;
    wait;

end process;
```

Nella simulazione effettuata si può notare dai valori contenuti nella ROM e nella MEM alla fine dell'elaborazione che il tutto avviene correttamente.



## Sintesi su board di sviluppo



Per la sintesi su board si è usufruito di 4 led per visualizzare l'uscita della macchina memorizzata nella MEM, un tasto di start per avviare il sistema che è stato prima filtrato in un apposito button debouncer e un tasto per effettuare il reset, in questo caso non è stato necessario filtrare il segnale del bottone.

Per permettere di scorrere una locazione per volta si è resa necessaria la modifica dell'unità di controllo del sistema LES, in modo che scandisse una locazione di memoria ad ogni start ricevuto, si riporta il codice modificato.

## UNITA' DI CONTROLLO – BEHAVIORAL

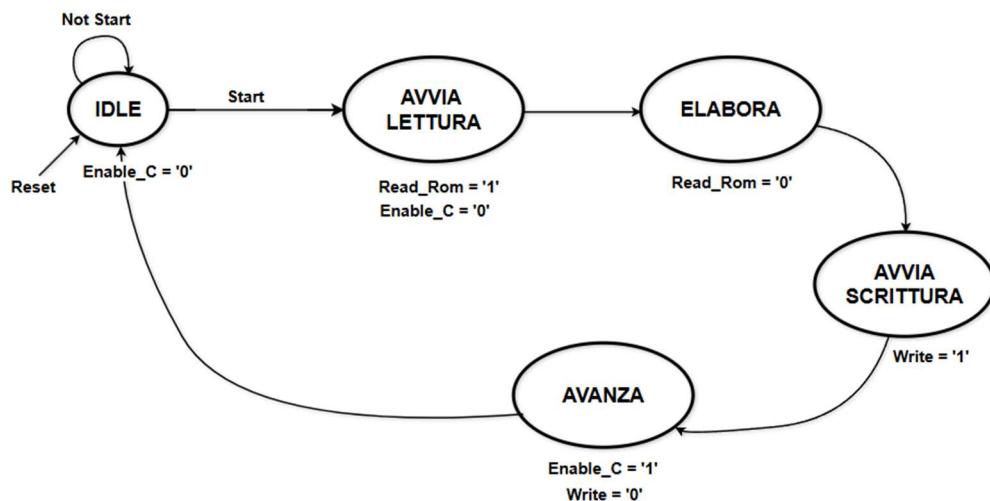
```

when scrittura_effettuata =>
  write <= '0';
  enable_counter <= '1';

  stato_corrente <= idle; -- MODIFICA PER LA SINTESI SU BOARD

  if unsigned(count)<7 then
    stato_corrente <= avvia lettura;
  else
    stato_corrente <= idle;
  end if;
end case;
end if;
  
```

Si riporta anche l'automa modificato.



## SISTEMA LES ON BOARD – STRUCTURAL (*sistema\_les\_on\_board.vhd*)

```
entity sistema_les_on_board is
    Port ( btn1 : in STD_LOGIC;
           clock_in : in STD_LOGIC;
           reset : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (3 downto 0));
end sistema_les_on_board;

architecture Structural of sistema_les_on_board is

component sistema_les is
    Port ( clock : in STD_LOGIC;
           start : in STD_LOGIC;
           reset : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component ButtonDebouncer is
    generic (
        CLK_period: integer := 10; -- periodo del clock (della board) in nanosecondi
        btn_noise_time: integer := 10000000 -- durata stimata dell'oscillazione del bottone in nanosecondi
                                         -- il valore di default è 10 millisecondi
    );
    Port ( RST : in STD_LOGIC;
           CLK : in STD_LOGIC;
           BTN : in STD_LOGIC;
           CLEARED_BTN : out STD_LOGIC);
end component;
```

```
begin
    btn: ButtonDebouncer
        generic map (
            CLK_period => 10, -- periodo del clock (della board) in nanosecondi
            btn_noise_time => 10000000 -- durata stimata dell'oscillazione del bottone in nanosecondi
                                         -- il valore di default è 10 millisecondi
        )
        Port map( RST => reset,
                  CLK => clock_in,
                  BTN => btn1,
                  CLEARED_BTN => btn_read);

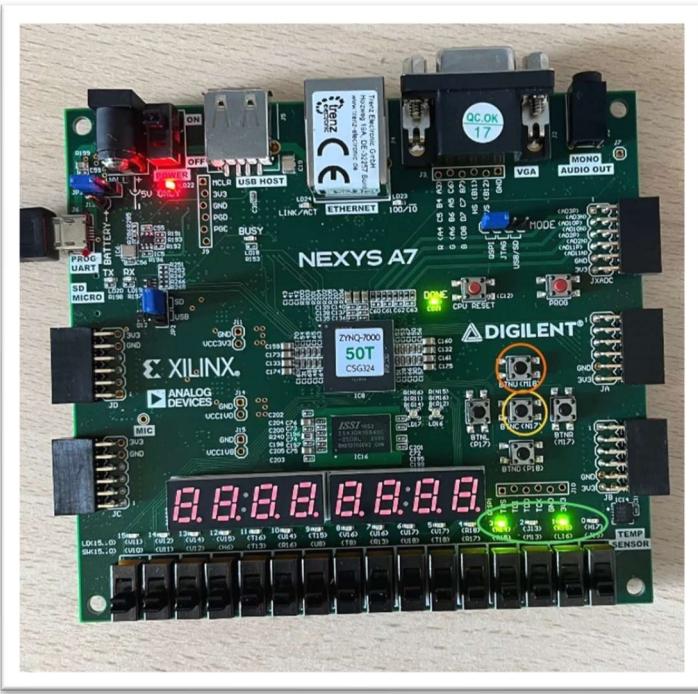
    sistema : sistema_les
        Port map( clock => clock_in,
                  start => btn_read,
                  reset => reset,
                  output => output);
end Structural;
```

Riportiamo i pin dei led e dei pulsanti utilizzati.

```
dict { PACKAGE_PIN M1 / IOSTANDARD LVCMS33 } [get_ports { output[0] }];
dict { PACKAGE_PIN K15 IOSTANDARD LVCMS33 } [get_ports { output[1] }];
dict { PACKAGE_PIN J13 IOSTANDARD LVCMS33 } [get_ports { output[2] }];
dict { PACKAGE_PIN N14 IOSTANDARD LVCMS33 } [get_ports { output[3] }];
```

```
dict { PACKAGE_PIN N17 IOSTANDARD LVCMS33 } [get_ports { reset }];
dict { PACKAGE_PIN M18 IOSTANDARD LVCMS33 } [get_ports { btn1 }];
```

Infine troviamo l'immagine della scheda con evidenziati i componenti utilizzati.

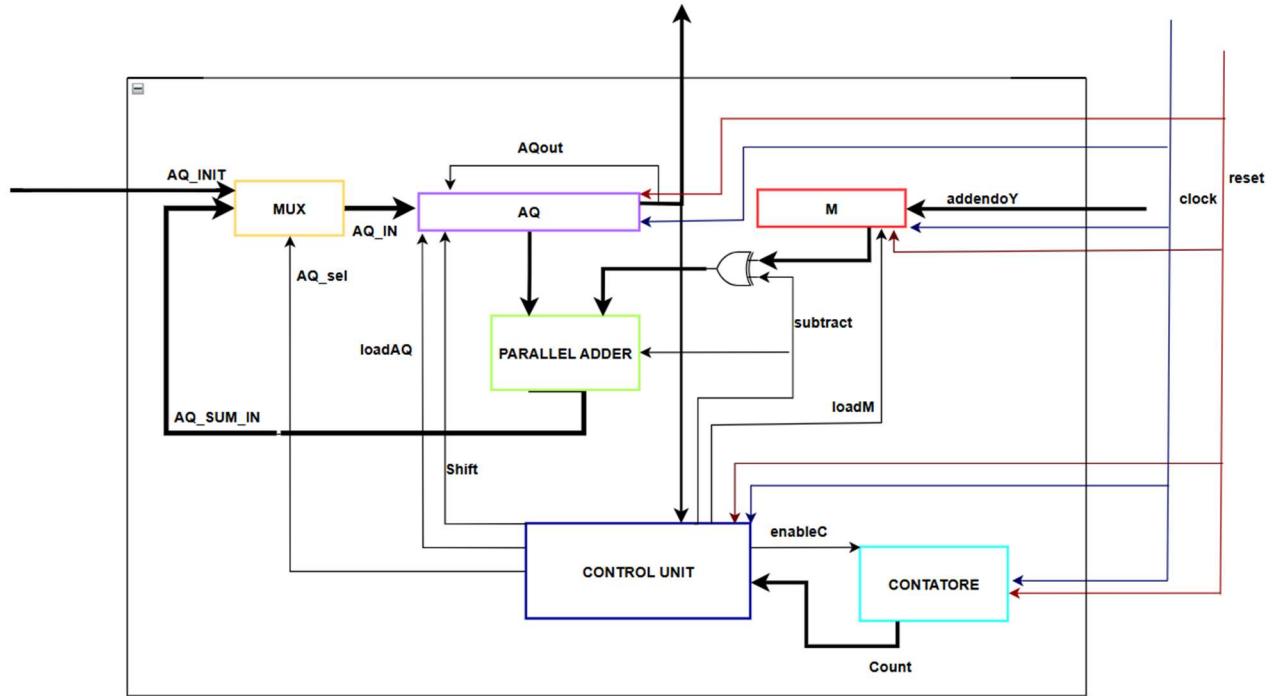


## Capitolo 3: Macchine aritmetiche

### Esercizio 7.1

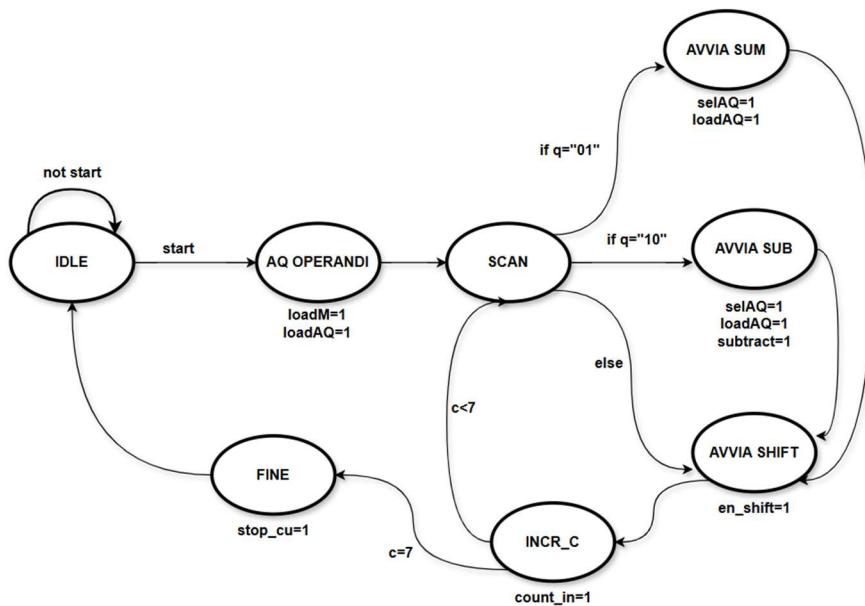
Progettare, implementare in VHDL e simulare una macchina moltiplicatore di Booth in grado di effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna

#### Progetto e Architettura



Si è scelto di utilizzare un solo registro a scorrimento da 17 bit per AQ. Per il resto l'architettura ricalca quella fornita.

#### Automa dell'unità di controllo



L'automa dell'UC parte nel momento in cui viene fornito un segnale di start dall'esterno, in tale momento si effettua l'acquisizione degli operandi di cui si deve calcolare il prodotto, considerando la giusta selezione

dello SR che sarà selAQ = '0', dopodiché si procede analizzando il valore degli ultimi due bit del registro AQ, in base al valore di essi si decide se effettuare la somma, la sottrazione o se effettuare un semplice shift, in ogni caso dopo l'operazione di somma e sottrazione comunque viene effettuato uno shift. Si prosegue in questo modo finché non termina il conteggio del contatore, momento in cui si ritorna nello stato di idle.

### *Implementazione*

Per realizzare il moltiplicatore di Booth sono stati utilizzati componenti base riportati in appendice. Si è scelto di seguire un approccio strutturale dividendo chiaramente unità operativa e unità di controllo.

## UNITA' OPERATIVA – STRUCTURAL (UO.vhd)

```

44 ; library IEEE;
45 use IEEE.STD_LOGIC_1164.ALL;
46
47 entity UO is
48     Port ( X : in STD_LOGIC_VECTOR (7 downto 0);
49             Y : in STD_LOGIC_VECTOR (7 downto 0);
50             enable_c: in STD_LOGIC;
51             count: out STD_LOGIC_VECTOR(2 downto 0);
52             subtract : in STD_LOGIC;
53             selAQ: in STD_LOGIC;
54             loadM: in STD_LOGIC;
55             loadAQ: in STD_LOGIC;
56             shift: in STD_LOGIC;
57             clock : in STD_LOGIC;
58             reset : in STD_LOGIC;
59             output : out STD_LOGIC_VECTOR (16 downto 0));
60 end UO;
61
62 architecture Behavioral of UO is
63
64     component cont_mod8 is
65         port( clock, reset: in std_logic;
66               count_in: in std_logic;
67               count: out std_logic_vector(2 downto 0));
68     end component;
69
70     component adder_sub is
71         port( X, Y: in std_logic_vector(7 downto 0);
72               cin: in std_logic;
73               Z: out std_logic_vector(7 downto 0);
74               cout: out std_logic);
75     end component;
76
77     component mux_21 is
78         generic (width : integer range 0 to 17 := 8);
79         port( x0, x1: in std_logic_vector(width-1 downto 0);
80               s: in std_logic;
81               y: out std_logic_vector(width-1 downto 0));
82 end architecture;

```

```

33 component registros is
34     port( A: in std_logic_vector(7 downto 0);
35         clk, res, load: in std_logic;
36         B: out std_logic_vector(7 downto 0));
37 end component;
38
39 component shift_register is
40     port( parallel_in: in std_logic_vector(16 downto 0);
41         serial_in: in std_logic;
42         clock, reset, load, shift: in std_logic;
43         parallel_out: out std_logic_vector(16 downto 0));
44 end component;
45
46 signal addendo_x : std_logic_vector(7 downto 0);
47 signal addendo_y : std_logic_vector(7 downto 0);
48 signal result_z: std_logic_vector(7 downto 0);
49 signal carry_out: std_logic;
50 signal AQ_init : std_logic_vector(16 downto 0);
51 signal AQ_sum_in: std_logic_vector(16 downto 0);
52 signal AQ_out: std_logic_vector(16 downto 0);
53 signal AQ_in: std_logic_vector(16 downto 0);
54 signal moltiplicando: std_logic_vector(7 downto 0);
55 signal Q16: std_logic;
56
57 begin
58
59     contatore : cont_mod8
60     port map( clock => clock, reset => reset, count_in => enable_c, count => count);
61
62     addizionatore: adder_sub
63     port map ( X => addendo_x, Y => addendo_y, cin => subtract,
64                Z => result_z,
65                cout => carry_out);
66
67     AQ_init <= "00000000"&X&'0';
68     AO sum in <= result z & AO out(8 downto 0);

```

```

01     generic map(width => 17)
02     port map( x0 => AQ_init ,
03                x1 => AQ_sum_in,
04                s => selAQ,
05                y => AQ_in);
06
07     molt : registro8
08     port map( A => Y,
09                clk => clock, res => reset, load => loadM,
10                B => addendo_y);
11
12     AQ: shift_register
13     port map( parallel_in => AQ_in,
14                serial_in => Q16,
15                clock => clock, reset => reset, load => loadAQ , shift => shift,
16                parallel_out => AQ_out);
17
18     Q16 <= AQ_out(16);
19
20     addendo_x <= AQ_out(16 downto 9);
21     output <= AQ_out(16 downto 0);
22

```

Costruiamo AQ\_init, segale di inizializzazione del registro AQ, concatenando una stringa di 8 zeri, l'operando X su 8 bit e un ultimo bit pari (Q[0]). AQ\_SUMIN invece è dato dalla concatenazione del risultato della somma e degli ultimi 9 bit del registro AQ. In questo modo è gestita la presenza di un unico registro a scorrimento.

## UNITA' DI CONTROLLO – BEHAVIORAL (*unita\_controllo.vhd*)

```
entity unita_controllo is
    port( clock, reset, start: in std_logic; --clock è il clock della board, clock_div viene dal divisore di freq
          count: in std_logic_vector(2 downto 0);
          q : in std_logic_vector(1 downto 0);
          loadM, count_in, loadAQ, en_shift: out std_logic;
          selAQ, subtract, stop_cu: out std_logic);
end entity unita_controllo;
```

```
architecture structural of unita_controllo is
type state is (idle, acquisisci_op, scan, avvia_sub, avvia_somma, avvia_shift, incr_count, fine);
signal current_state,next_state: state;

begin
reg_stato: process(clock)
begin
    if(clock'event and clock='1') then
        if(reset='1') then
            current_state <=idle;
        else
            current_state <=next_state;
        end if;
    end if;
end process;
```

```
comb: process(current_state, start, count,q)
begin
    count_in <='0';
    subtract <='0';
    selAQ <= '0';
    loadAQ <='0'; --carica nello shift register
    loadM <='0'; --carica il moltiplicando nel registro M
    stop_cu <='0';
    en_shift <='0'; --segnale che abilita lo shift durante le prime N-1 iterazioni

    CASE current_state is
        WHEN idle =>
            if(start='1') then
                next_state <= acquisisci_op;
            else
                next_state <= idle;
            end if;
        --fornisce i segnali di caricamento operandi
        WHEN acquisisci_op =>
            loadM <='1'; --abilita il caricamento del moltiplicando nel registro M
            loadAQ <='1'; --abilita il caricamento del moltiplicatore e degli 8 zeri in testa
                           --nello shift register A.Q (perchè selAQ=0)
            next_state <= scan;
```

```

WHEN scan =>
    if q = "01" then
        next_state <= avvia_somma;
    elsif q = "10" then
        next_state <= avvia_sub;
    else
        next_state <= avvia_shift;
    end if;
WHEN avvia_somma =>
    selAQ <= '1';
    loadAQ <= '1'; --fornisce il segnale di caricamento in A del risultato della somma
    next_state <= avvia_shift;

WHEN avvia_sub =>
    selAQ <= '1';
    loadAQ <= '1'; --fornisce il segnale di caricamento in A del risultato della somma
    subtract <= '1';
    next_state <= avvia_shift;

WHEN avvia_shift =>
    en_shift <='1';
    next_state <= incr_count;

WHEN incr_count =>
    count_in <= '1';
    if unsigned(count) < 7 then
        next_state <= scan;
    else
        next_state <= fine;
    end if;
WHEN fine =>
    stop_cu <='1';
    next_state <= idle;
end CASE;
end process;
end structural;

```

## *Simulazione*

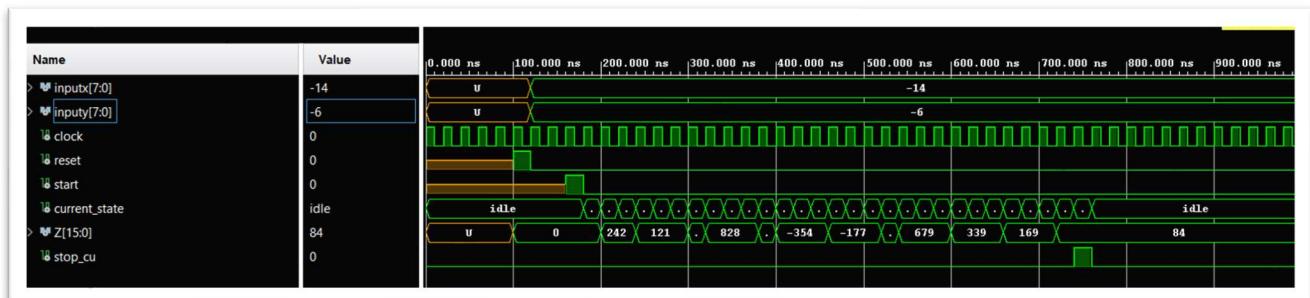
Per simulare il sistema è stato sviluppato un testbench, di seguito riportato.

```

3 CLK_PROCESS : process
4 begin
5   while (end_sim = '0') loop
6     clock<= '1';
7     wait for clk_period/2;
8     clock <= '0';
9     wait for clk_period/2;
0   end loop;
1   wait;
2 end process;
3
4 stimulus: process
5 begin
6
7   -- Put initialisation code here
8   wait for 100 ns;
9
0   reset<='1';
1   wait for 20 ns;
2   reset<='0';
3
4   -- 11110110
5   inputx<="11110010";
6   inputy<="11111010";
7
8   -- start deve essere visto da clk_div: poiché sarà generato dal button debouncer si aggiungerà anche il clk_div
9   -- al button debouncer e il segnale di start deve durare quanto il periodo del clk rallentato
0   wait for 40 ns;
1   start<='1';
2   wait for 20 ns;
3   start<='0';
4   -- aspetto fine operazione, ci vogliono circa 500ns
5   wait for 600ns;
6
7   wait;
8 end process;
9

```

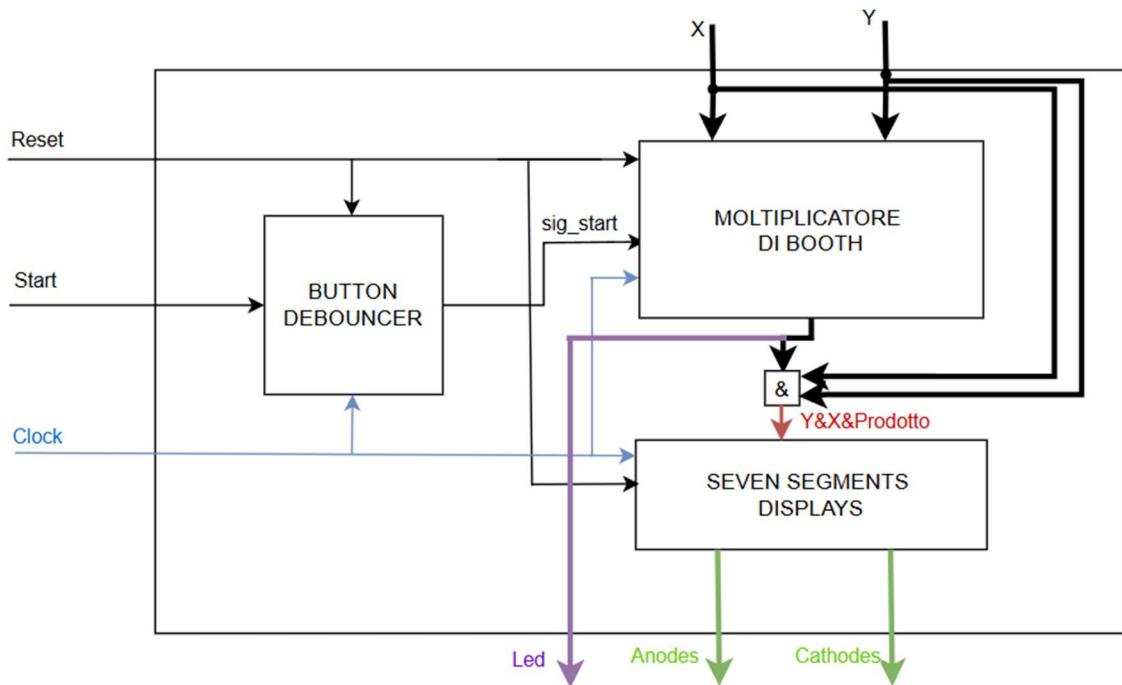
La simulazione behavioral mostra come la macchina evolve negli stati fino al termine dell'operazione richiesta.



## Esercizio 7.2

Sintetizzare il moltiplicatore implementato al punto 7.1 su FPGA e testarlo mediante l'utilizzo dei dispositivi di input/output (switch, bottoni, led, display) presenti sulla board di sviluppo in dotazione. La modalità di utilizzo degli stessi è a completa discrezione degli studenti.

### Progetto e architettura



Come si può vedere dallo schema a blocchi riportato si è deciso di inserire il segnale di start in un button debouncer che invia il segnale ripulito al moltiplicatore. Quest'ultimo riceve come input in ingresso moltiplicando(Y) e moltiplicatore(X), in uscita vedremo sui primi 2 display il valore Y, sui secondi due il valore X e sugli ultimi 4 il valore della moltiplicazione (tutto in esadecimale). Inoltre l'uscita del moltiplicatore è anche collegata ai led per permetterne la visualizzazione in binario.

### Implementazione

```
entity molt_booth_on_board is
  Port ( x : in STD_LOGIC_VECTOR (7 downto 0);
         y : in STD_LOGIC_VECTOR (7 downto 0);
         start : in STD_LOGIC;
         clock: in STD_LOGIC;
         z: out std_logic_vector(15 downto 0);
         anodes: out std_logic_vector(7 downto 0);
         cathodes: out std_logic_vector(7 downto 0);
         reset : in STD_LOGIC);
end molt_booth_on_board;
```

```

architecture Behavioral of molt_booth_on_board is
component display_seven_segments is
  Generic(
    clock_frequency_in : integer := 50000000;
    clock_frequency_out : integer := 5000000
  );
  Port ( clock : in STD_LOGIC;
         reset : in STD_LOGIC;
         value32_in : in STD_LOGIC_VECTOR (31 downto 0);
         enable : in STD_LOGIC_VECTOR (7 downto 0);
         dots : in STD_LOGIC_VECTOR (7 downto 0);
         anodes : out STD_LOGIC_VECTOR (7 downto 0);
         cathodes : out STD_LOGIC_VECTOR (7 downto 0));
  end component;

component ButtonDebouncer is
  generic (
    CLK_period: integer := 10; -- periodo del clock (della board) in nanosecondi
    btn_noise_time: integer := 10000000 -- durata stimata dell'oscillazione del bottone in nanosecondi
                                         -- il valore di default è 10 millisecondi
  );
  Port ( RST : in STD_LOGIC;
         CLK : in STD_LOGIC;
         BTN : in STD_LOGIC;
         CLEARED_BTN : out STD_LOGIC);
end component;

component moltiplicatore_booth is
  Port ( X : in STD_LOGIC_VECTOR (7 downto 0);
         Y : in STD_LOGIC_VECTOR (7 downto 0);
         start: in STD_LOGIC;
         clock : in STD_LOGIC;
         reset : in STD_LOGIC;
         Z : out STD_LOGIC_VECTOR (15 downto 0);
         stop_cu: out STD_LOGIC);
end component;

```

```

signal sig_start: std_logic;
signal prodotto: std_logic_vector(15 downto 0);
signal sig_value: std_logic_vector(31 downto 0);
signal stop_cu : std_logic;

begin

  btnStart: ButtonDebouncer generic map( CLK_period => 10, btn_noise_time => 10000000)
  port map(RST => reset,
            CLK => clock,
            BTN => start,
            CLEARED_BTN => sig_start);

  molt_booth: moltiplicatore_booth
  Port map(
    X => X,
    Y => Y,
    start => sig_start,
    clock => clock,
    reset => reset,
    Z => prodotto,
    stop_cu => stop_cu);

  sig_value <=Y&X&prodotto;
  z <= prodotto;

  display: display_seven_segments
  Generic map(
    clock_frequency_in => 100000000,
    clock_frequency_out =>500
  )
  Port map ( clock => clock,
             reset => reset,
             value32_in => sig_value,
             enable => "11111111",
             dots => "00000000",
             anodes => anodes,
             cathodes => cathodes);
end Behavioral;

```

## Sintesi su board di sviluppo

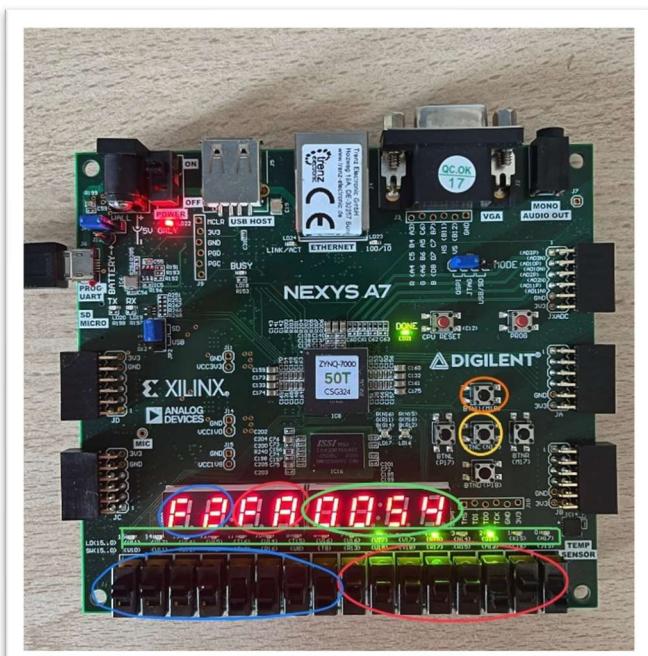
Si riporta di seguito il file di constraints.

```
## FUSES TO 0x0
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { x[0] }]; #IO_L24N_T3_B00_14 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { x[1] }]; #IO_L1H_T0_D00_EMCCLE_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { x[2] }]; #IO_L0H_T0_D00_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { x[3] }]; #IO_L13H_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { x[4] }]; #IO_L11N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { x[5] }]; #IO_L17H_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { x[6] }]; #IO_L17H_T2_A13_D28_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { x[7] }]; #IO_L5H_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS18 } [get_ports { y[0] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { y[1] }]; #IO_L25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS18 } [get_ports { y[2] }]; #IO_L15P_T2_D00_RWB_8_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS18 } [get_ports { y[3] }]; #IO_L12N_T2_D00_RWB_8_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS18 } [get_ports { y[4] }]; #IO_L24P_T2_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS18 } [get_ports { y[5] }]; #IO_L00P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS18 } [get_ports { y[6] }]; #IO_L18H_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS18 } [get_ports { y[7] }]; #IO_L21P_T3_D00_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS18 } [get_ports { z[0] }]; #IO_L18P_T2_A22_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS18 } [get_ports { z[1] }]; #IO_L24P_T3_R21_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS18 } [get_ports { z[2] }]; #IO_L17H_T2_A23_15 Sch=led[2]
set_property -dict { PACKAGE_PIN R11 IOSTANDARD LVCMOS18 } [get_ports { z[3] }]; #IO_L28P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS18 } [get_ports { z[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS18 } [get_ports { z[5] }]; #IO_L13H_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS18 } [get_ports { z[6] }]; #IO_L17P_T2_A14_D20_14 Sch=led[6]
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS18 } [get_ports { z[7] }]; #IO_L28P_T2_A12_D26_14 Sch=led[7]
set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS18 } [get_ports { z[8] }]; #IO_L14P_T2_A10_D25_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS18 } [get_ports { z[9] }]; #IO_L14H_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS18 } [get_ports { z[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS18 } [get_ports { z[11] }]; #IO_L15H_T2_D00_DOUT_C2Q_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS18 } [get_ports { z[12] }]; #IO_L16P_T2_C2I_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS18 } [get_ports { z[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS18 } [get_ports { z[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS18 } [get_ports { z[15] }]; #IO_L21N_T3_D05_A06_D22_14 Sch=led[15]

## Buttons
#set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L3P_T0_D00_A01P_15 Sch=mpu_reseetn
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L9P_T1_D00_14 Sch=bttn
```

Nell'immagine della board riportiamo in rosso gli switch e i led del moltiplicatore (X), in blu gli switch e i led del moltiplicando (Y) e in verde il risultato della moltiplicazione, in esadecimale sui 4 display e in binario sui led. Il bottone in giallo permette di effettuare il reset, quello in arancione da lo start al moltiplicatore.



## Capitolo 4: Comunicazione con handshaking

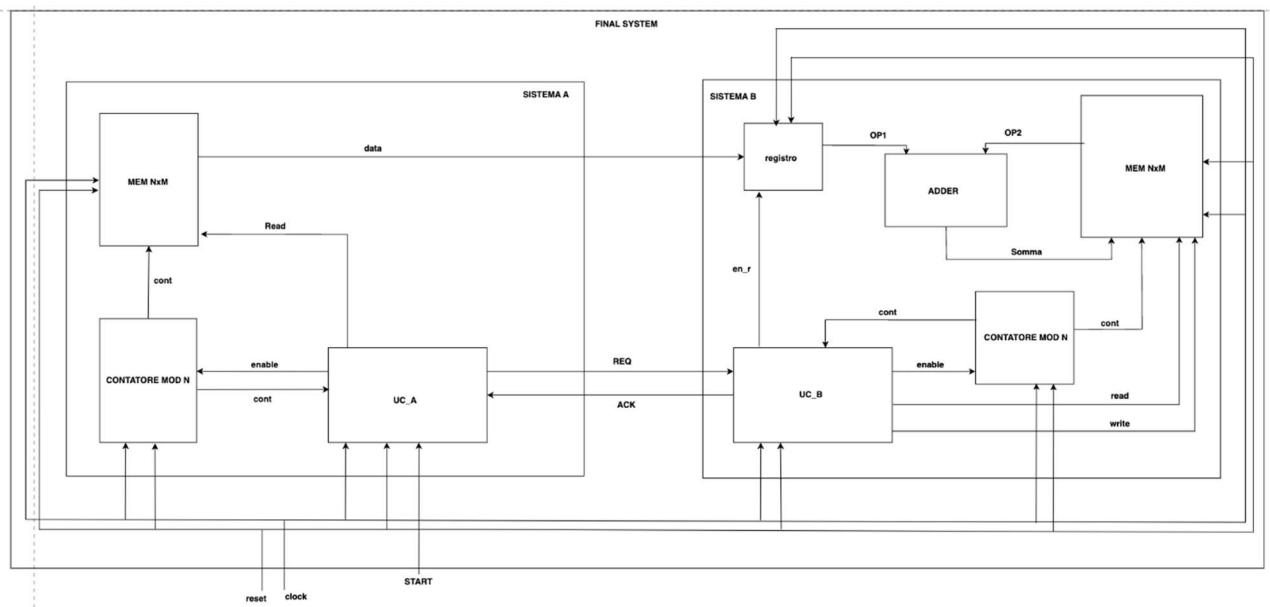
### Esercizio 8.1:

Progettare, implementare in VHDL e testare mediante simulazione un sistema composto da 2 nodi, A e B, che comunicano mediante un protocollo di handshaking. Il nodo A e il nodo B possiedono entrambi una memoria interna in cui sono memorizzate N stringhe di M bit, denominate X(i) e Y(i) rispettivamente ( $i=0,..,N-1$ ). Il nodo A trasmette a B ciascuna stringa X(i) utilizzando un protocollo di handshaking; B, ricevuta la stringa X(i), calcola  $S(i)=X(i)+Y(i)$  e immagazzina la somma in opportune locazioni della propria memoria interna.

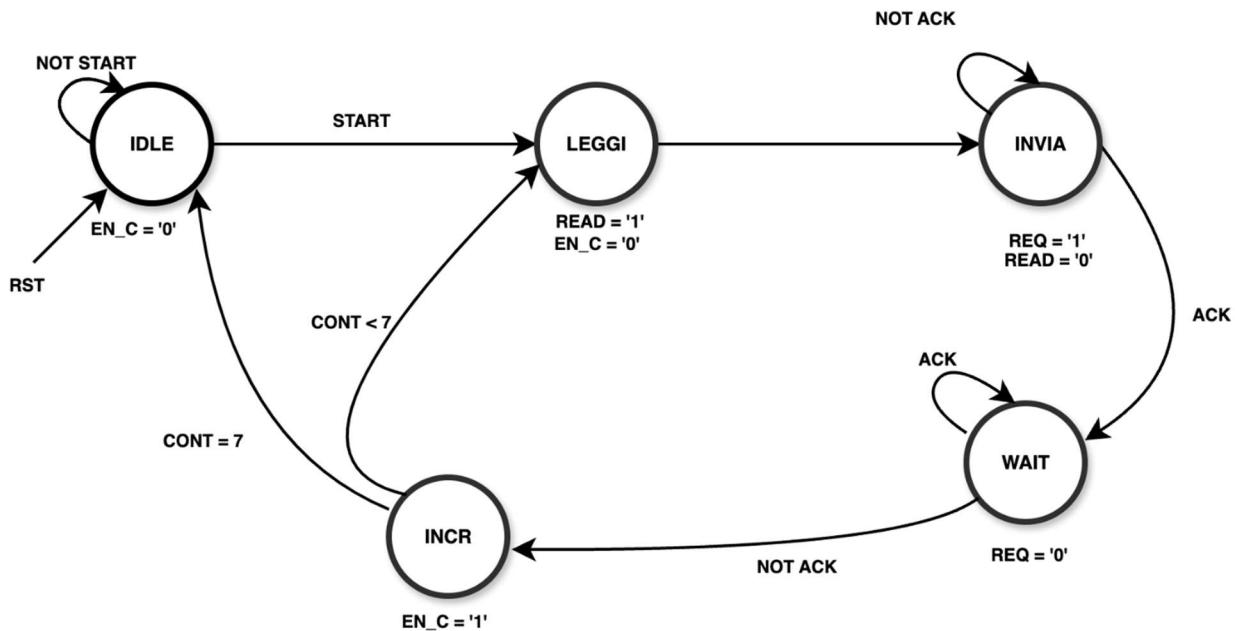
Per il progetto è possibile considerare una implementazione di tipo comportamentale per effettuare la somma, mentre è necessario prevedere esplicitamente un componente contatore sia nel sistema A sia nel sistema B per scandire la trasmissione/ricezione delle stringhe e per terminare la comunicazione.

#### Progetto e Architettura

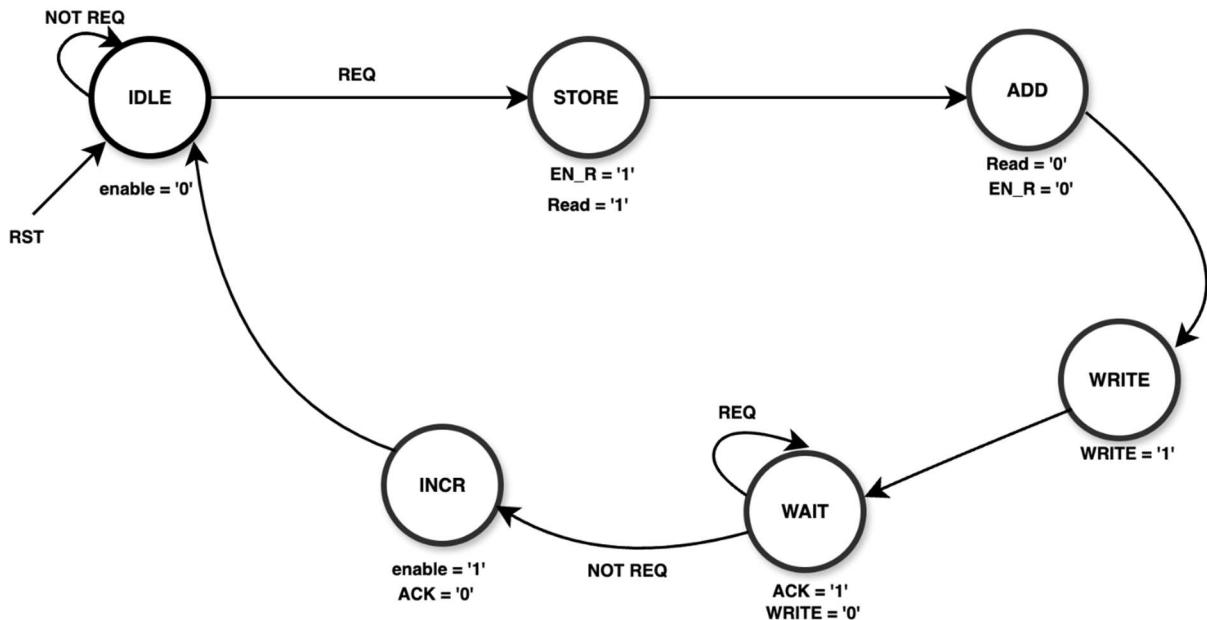
L'architettura di questo sistema prevede la comunicazione tramite handshaking semplice tra due nodi A e B.



All'interno dei due nodi troviamo i componenti che servono a implementare quanto richiesto dalla traccia: in A una ROM contiene le stringhe di bit che costituiranno l'operando 1 dell'adder nel nodo B, dato che sarà reso stabile tramite un registro di 8 bit inserito nel nodo B. I dati prelevati dalla memoria di B costituiscono il secondo operando di ogni operazione di somma e questo valore sarà poi scritto nella memoria alla stessa locazione indirizzata correntemente: dunque ogni operazione di somma è da intendersi come una sovrascrizione delle celle di memoria del nodo B con una stringa di 8 bit che è la somma dei due operandi. Gli indirizzi alle memorie sono dati tramite contatori modulo N. A ogni elemento sincrono dell'architettura è fornito il segnale di clock e di reset e alla UC di A è fornito anche un RESET poiché costituisce il trasmittente. Le due unità di controllo infine si scambiano i segnali di REQ e ACK per implementare l'handshaking (comunicazione sincrona).



**Unità di controllo di A.**



**Unità di controllo di B.**

L'handshaking che abbiamo utilizzato è implementato in modo tale da cauterarci dai casi in cui il clock di A sia più veloce del clock di B, o viceversa, e dai casi in cui questi possono risultare sfasati portando a problemi di tempificazione. In questo modo il trasmettitore alza per primo la REQ attendendo la ACK, dopodichè abbassa la REQ e prima di continuare l'esecuzione attende che l'ACK sia di nuovo bassa: questo ci cautela dalla possibilità di vedere come ACK alzato nel nuovo stato “INVIA” del trasmettitore l'ACK del ciclo precedente alzato dal ricevitore qualora questo fosse molto più lento del trasmettitore.

Allo stesso modo nel ricevitore una volta alzato l'ACK attendiamo che il REQ si abbassi per procedere con l'esecuzione ed abbassare l'ACK: questo ci cautela dalla possibilità che il ricevitore veda, tornato nello stato

“IDLE” un segnale di REQ alzato che appartiene alla comunicazione precedente, nel caso in cui il ricevitore fosse molto più veloce del trasmettitore.

NOTA: una modifica che si potrebbe effettuare nell’automa di B è di alzare l’ack dopo aver letto il dato, in modo tale da far predisporre il sistema A all’invio di un nuovo dato, senza dover attendere il termine dell’elaborazione di B.

#### Implementazione

Sono state omesse le implementazioni delle componenti basi come memoria, contatori, poiché riportati nell’appendice.

### FINAL SYSTEM – STRUCTURAL (*final\_system.vhd*)

```

54  entity finalsystem is
55  generic (
56      N : integer := 16; -- Numero di locazioni di memoria
57      addrN : integer := 4;
58      M : integer := 8    -- Larghezza di ogni locazione in bit
59  );
60  Port (clockA: in std_logic;
61         clockB: in std_logic;
62         resetA: in std_logic;
63         resetB: in std_logic;
64         start: in std_logic );
65 end finalsystem;
66
67 architecture structural of finalsystem is
68 component sistemaA is
69  generic (
70      N : integer := 16; -- Numero di locazioni di memoria
71      addrN : integer := 4;
72      M : integer := 8    -- Larghezza di ogni locazione in bit
73  );
74  Port (clock, reset: in std_logic;
75         start: in std_logic;
76         req: out std_logic;
77         data: out std_logic_vector(M-1 downto 0);
78         ack: in std_logic );
79 end component;
80
81 component sistemaB is
82  generic (
83      N : integer := 16; -- Numero di locazioni di memoria
84      addrN : integer := 4;
85      M : integer := 8    -- Larghezza di ogni locazione in bit
86  );
87  Port (clock, reset: in std_logic;
88         req: in std_logic;
89         data: in std_logic_vector(M-1 downto 0);
90         ack: out std_logic );
91 end component;

```

```

73 : signal req: std_logic;
74 : signal ack: std_logic;
75 : signal data: std_logic_vector(M-1 downto 0);
76 :
77 begin
78
79 A: sistemaA generic map(N, addrN, M) port map(clockA, resetA, start, req, data, ack);
80 B: sistemaB generic map(N, addrN, M) port map(clockB, resetB, req, data, ack);
81
82
83 end structural;

```

## SISTEMA A – STRUCTURAL (*sistemaA.vhd*)

```

34  entity sistemaA is
35      generic (
36          N : integer := 16;    -- Numero di locazioni di memoria
37          addrN : integer := 4;
38          M : integer := 8     -- Larghezza di ogni locazione in bit
39      );
40      Port (clock, reset: in std_logic;
41              start: in std_logic;
42              req: out std_logic;
43              data: out std_logic_vector(M-1 downto 0);
44              ack: in std_logic );
45  end sistemaA;
46
47  architecture structural of sistemaA is
48  component GenericMemory is
49      generic (
50          N : integer := 16;    -- Numero di locazioni di memoria
51          addrN : integer := 4;
52          M : integer := 8     -- Larghezza di ogni locazione in bit
53      );
54      port (
55          clk      : in  std_logic;
56          we       : in  std_logic;
57          re       : in  std_logic;
58          addr    : in  std_logic_vector(addrN-1 downto 0); -- Indirizzo di memoria
59          din     : in  std_logic_vector(M-1 downto 0);        -- Dato in ingresso (da scrivere)
60          dout   : out std_logic_vector(M-1 downto 0)           -- Dato in uscita (letto)
61      );
62  end component;
63
64  component GenericCounter is
65      generic (
66          addrN : integer := 4
67      );
68      port( clock, reset: in std_logic;
69             count_in: in std_logic;
70             count: out std_logic_vector(addrN-1 downto 0));
71  end component;

```

```

73  component UC_A is
74      generic (
75          N : integer := 16;    -- Numero di locazioni di memoria
76          addrN : integer := 4
77      );
78      Port (clock: in std_logic;
79              req: out std_logic;
80              ack: in std_logic;
81              start: in std_logic;
82              read: out std_logic;
83              enable: out std_logic;
84              c: in std_logic_vector(addrN-1 downto 0)
85      );
86  end component;
87
88  signal Read: std_logic;
89  signal Enable: std_logic;
90  signal addr: std_logic_vector(addrN-1 downto 0);
91 begin
92
93  MemA: GenericMemory generic map(N, addrN, M) port map(clock, '0', Read, addr, (others=> '0'), data);
94
95  ContmodN: GenericCounter generic map(addrN) port map(clock, reset, Enable, addr);
96
97  UC: UC_A generic map(N, addrN) port map(clock, req, ack, start, Read, Enable, addr);
98
99  end structural;

```

## UNITA' DI CONTROLLO A – BEHAVIORAL (UC\_A.vhd)

```

34  entity UC_A is
35      generic (
36          N : integer := 16; -- Numero di locazioni di memoria
37          addrN : integer := 4
38      );
39      Port (clock: in std_logic;
40             req: out std_logic;
41             ack: in std_logic;
42             start: in std_logic;
43             read: out std_logic;
44             enable: out std_logic;
45             c: in std_logic_vector(addrN-1 downto 0)
46         );
47 end UC_A;
48
49 architecture Behavioral of UC_A is
50
51 type stato is (IDLE, Leggi, Invia, Attesa, Incr);
52 signal stato_corrente : stato := IDLE;
53 signal stato_prossimo: stato;
54
55 begin
56
57 uscita:process(stato_corrente, start, ack, c)
58 begin
59     case stato_corrente is
60         when IDLE =>
61             enable <='0';
62             if(start='1') then
63                 stato_prossimo<=Leggi;
64             else
65                 stato_prossimo<=IDLE;
66             end if;
67         when Leggi =>
68             read<='1';
69             enable<='0';
70             stato_prossimo<=Invia;

```

```

71             when Invia=>
72                 req<='1';
73                 read<='0';
74                 if(ack='0') then
75                     stato_prossimo<=Invia;
76                 else
77                     stato_prossimo<=Attesa;
78                 end if;
79         when Attesa=>
80             req<='0';
81             if(ack='1') then
82                 stato_prossimo<=Attesa;
83             else
84                 stato_prossimo<=Incr;
85             end if;
86         when Incr=>
87             enable<='1';
88             if(to_integer(unsigned(c))<(N - 1)) then
89                 stato_prossimo<=Leggi;
90             else
91                 stato_prossimo<=IDLE;
92             end if;
93         when others =>
94             stato_prossimo<=IDLE;
95     end case;
96 end process;
97
98 mem:process(clock)
99 begin
100    if(clock'event and clock='1') then
101        stato_corrente<=stato_prossimo;
102    end if;
103 end process;
104
105 end Behavioral;

```

## SISTEMA B – STRUCTURAL (*sistemaB.vhd*)

```

4 ∵ entity sistemaB is
5     generic (
6         N : integer := 16; -- Numero di locazioni di memoria
7         addrN : integer := 4;
8         M : integer := 8 -- Larghezza di ogni locazione in bit
9     );
10    Port (clock, reset: in std_logic;
11           req: in std_logic;
12           data: in std_logic_vector(M-1 downto 0);
13           ack: out std_logic );
14 end sistemaB;
15
16 ∵ architecture structural of sistemaB is
17 ∵ component GenericMemory is
18     generic (
19         N : integer := 16; -- Numero di locazioni di memoria
20         addrN : integer := 4;
21         M : integer := 8 -- Larghezza di ogni locazione in bit
22     );
23     port (
24         clk      : in std_logic;
25         we       : in std_logic;
26         re       : in std_logic;
27         addr    : in std_logic_vector(addrN-1 downto 0); -- Indirizzo di memoria
28         din     : in std_logic_vector(M-1 downto 0);      -- Dato in ingresso (da scrivere)
29         dout   : out std_logic_vector(M-1 downto 0)        -- Dato in uscita (letto)
30     );
31 end component;
32
33 ∵ Component registro is generic ( N : integer:=8);
34    port( A: in std_logic_vector(N-1 downto 0);
35           clk, res, load: in std_logic;
36           B: out std_logic_vector(N-1 downto 0));
37 end component;
38
39 ∵ component GenericCounter is
40     generic (
41         addrN : integer := 4
42     );
43     port( clock, reset: in std_logic;
44           count_in: in std_logic;
45           count: out std_logic_vector(addrN-1 downto 0));
46

```

```

78 ∵ component UC_B is
79     generic (
80         N : integer := 16; -- Numero di locazioni di memoria
81         addrN : integer := 4
82     );
83     Port (clock: in std_logic;
84            reg: in std_logic;
85            ack: out std_logic;
86            read: out std_logic;
87            write: out std_logic;
88            enable: out std_logic;
89            en_r: out std_logic;
90            c: in std_logic_vector(addrN-1 downto 0)
91        );
92 end component;
93
94 ∵ component RCA_Nbit is
95     generic (N: natural range 0 to 32 := 8);
96     port(
97         OP_A_RCA: in std_logic_vector(N-1 downto 0);
98         OP_B_RCA: in std_logic_vector(N-1 downto 0);
99         CIN_RCA: in std_logic;
100
101        S_RCA: out std_logic_vector(N-1 downto 0);
102        COUT_RCA: out std_logic;
103        OV: out std_logic
104    );
105 end component;
106
107 signal Read: std_logic;
108 signal Write: std_logic;
109 signal Enable: std_logic;
110 signal addr: std_logic_vector(addrN-1 downto 0);
111 signal somma: std_logic_vector(M-1 downto 0);
112 signal secondoOP: std_logic_vector(M-1 downto 0);
113 signal ov: std_logic;
114 signal cout: std_logic;
115 signal primoOP: std_logic_vector(M-1 downto 0);
116 signal an: std_logic;

```

```

19 begin
20 reg_OP_A: registro port map(data,clock,reset,en_r,primoOP);
21
22 MemB: GenericMemory generic map(N, addrN, M) port map(clock, Write, Read, addr, somma, secondoOP);
23
24 ContmodN: GenericCounter generic map(addrN) port map(clock, reset, Enable, addr);
25
26 UC: UC_B generic map(N, addrN) port map(clock, req, ack, Read, Write, Enable,en_r, addr);
27
28 sommatore: RCA_Nbit generic map(M) port map(primoOP, secondoOP, '0', somma, cout, ovr);
29
30
31 end structural;

```

## UNITA' DI CONTROLLO B – BEHAVIORAL (*uc\_b.vhd*)

```

14 entity uc_b is
15   generic (
16     N : integer := 16; -- Numero di locazioni di memoria
17     addrN : integer := 4
18   );
19   Port (clock: in std_logic;
20         req: in std_logic;
21         ack: out std_logic;
22         read: out std_logic;
23         write: out std_logic;
24         enable: out std_logic;
25         en_r: out std_logic;
26         c: in std_logic_vector(addrN-1 downto 0)
27       );
28 end UC_B;
29
30 architecture Behavioral of UC_B is
31
32   type stato is (IDLE, Store, WAIT_STATE, ADD,WRITE_STATE, INCR);
33   signal stato_corrente : stato := IDLE;
34   signal stato_prossimo: stato;
35
36   begin
37
38   uscita:process(stato_corrente, req)
39     begin
40       case stato_corrente is
41         when IDLE =>
42           ack<='0';
43           enable <='0';
44           en_r <= '0';
45           if(req='1') then
46             stato_prossimo<=Store;
47           else
48             stato_prossimo<=IDLE;
49           end if;
50         when Store =>
51           en_r <= '1';
52           read<='1';
53           ----- .....

```

```

73    statoprossimo<=ADD;
74    when ADD=>
75        en_r <= '0';
76        write<='0';
77        read<='0';
78        statoprossimo<=WRITE_STATE;
79    when WRITE_STATE =>
80        en_r <= '0';
81        write<='1';
82        read<='0';
83        statoprossimo<=WAIT_STATE;
84    when WAIT_STATE=>
85        ack<='1';
86        write<='0';
87        if(req='0') then
88            statoprossimo<=INCR;
89        else
90            statoprossimo<=WAIT_STATE;
91        end if;
92    when INCR=>
93        enable<='1';
94        ack<='0';
95        statoprossimo<=IDLE;
96    when others =>
97        statoprossimo<=IDLE;
98    end case;
99 end process;
100
101 mem:process(clock)
102 begin
103     if(clock'event and clock='1') then
104         statocorrente<=statoprossimo;
105     end if;
106 end process;
107
108 end Behavioral.

```

## Simulazione

### TESTBENCH

```

stimuli : process
begin
    -- EDIT Adapt initialization as needed
    resetA <= '0';
    resetB <= '0';
    start <= '0';

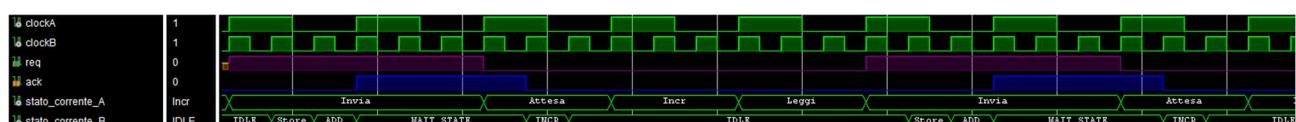
    -- EDIT Add stimuli here
    wait for 10 ns;
    start<='1';
    wait for TbPeriod;
    start <= '0';

    wait for 100000 ns;

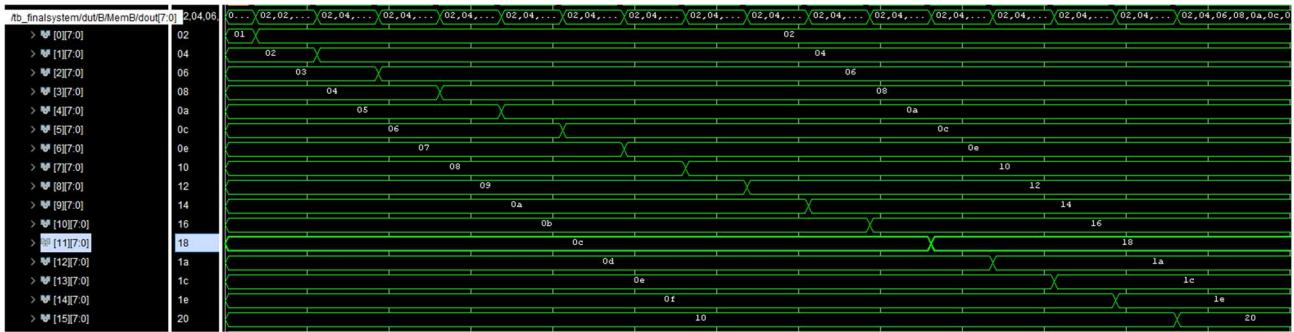
    -- Stop the clock and hence terminate the simulation
    TbSimEnded <= '1';
    wait;
end process;

```

Per testare il sistema, abbiamo impostato clock diversi per il sistema A e per il sistema B. La seguente simulazione mostra la corretta comunicazione che avviene tra i due i sistemi.



La successiva immagine mostra il corretto funzionamento del sistema. La memoria di B, infatti, presenta i valori aggiornati con la somma ottenuta dalla stringa ricevuta da A e la stringa di B.



## Capitolo 5: Processore

### Esercizio 9.a

Abbiamo deciso di analizzare le istruzioni BIPUSH e ISTORE. Attraverso la simulazione del processore con il tool GTKWave abbiamo analizzato i contenuti dei registri nel corso del tempo.

#### *BIPUSH 0xA*

L'istruzione analizzata è caratterizzata dal seguente codice:

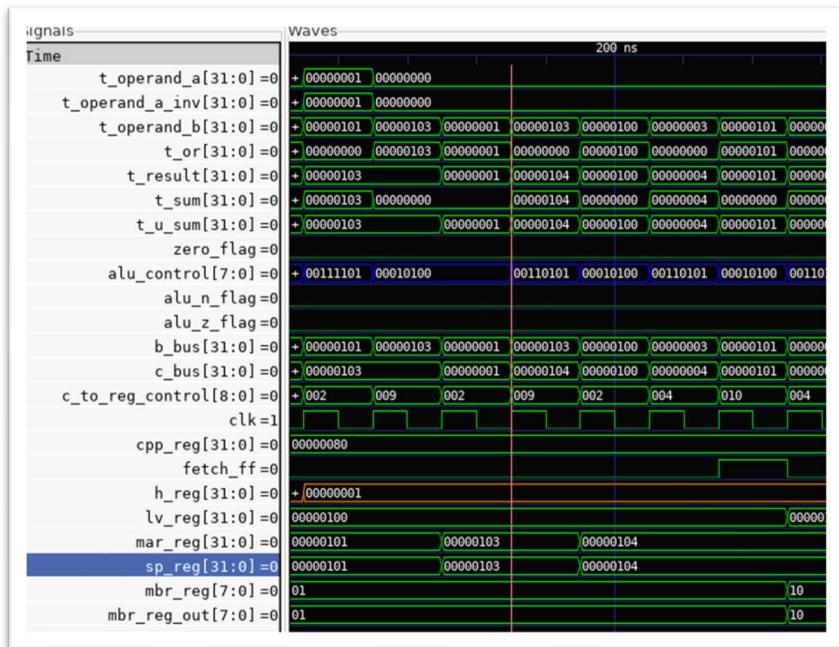
*bipush = 0x10:*

$$SP = MAR = SP + 1$$

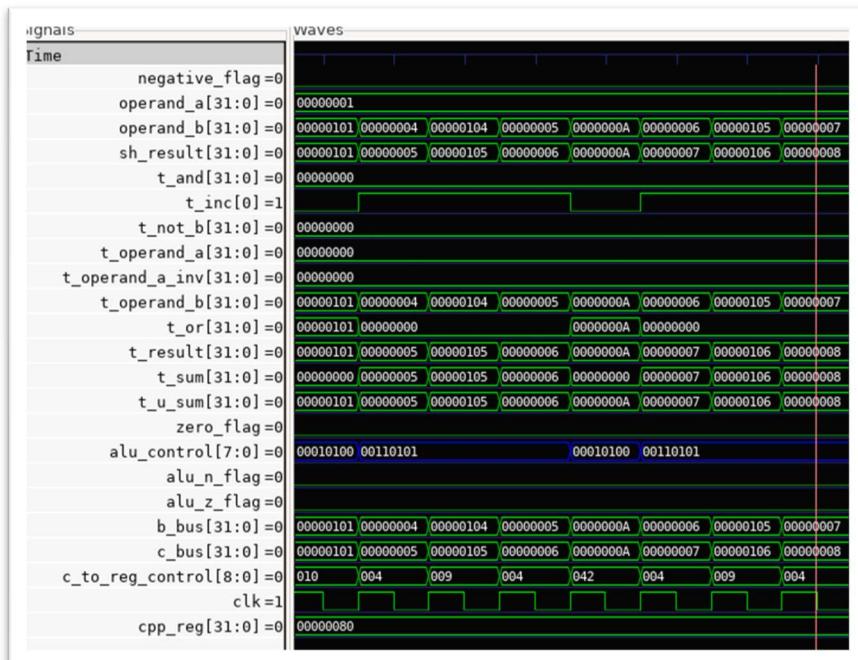
$$PC = PC + 1; \text{fetch}$$

$$MDR = TOS = MBR; wr; \text{goto main}$$

La prima cosa che va a fare tale è prendere il valore dello stack pointer corrente, incrementarlo di uno in modo da inserire il valore indicato come parametro dell'istruzione in cima allo stack. Per cui verrà inserito sul bus B il valore contenuto nel registro SP, che nella simulazione effettuata è pari inizialmente a *0X00000104*, verrà inserito nella ALU per poi aumentarne il valore di 1 con il segnale di controllo inviato alla ALU *00110101*. Il valore ottenuto viene memorizzato nello shift register che contiene il valore calcolato dalla ALU, al ciclo di clock successivo tale valore viene memorizzato in MAR e in SP. Nel primo poiché dobbiamo scrivere in quella locazione di memoria, mentre nel secondo perché dobbiamo mantenere il puntatore alla cima dello stack.



Grazie al fetch effettuato nel main in maniera concorrente al goto(0x10), nel MBR è già caricato il valore dell'operando (ovvero 0xa). Per memorizzare il valore in memoria è necessario mettere in scrittura il valore di MBR sul bus B, e fare in modo che la ALU faccia l'operazione di identità sull'operando B attraverso il segnale di controllo **00010100**. Il valore B viene memorizzato nello SR che contiene il risultato della ALU, dopodiché il valore viene memorizzato in TOS e in MDR, si attiva il segnale di WRITE e si scrive in memoria all'indirizzo contenuto in MAR. In contemporanea viene incrementato il PC ed effettuata un'operazione di fetch, in questo modo quando ritorniamo al main saltiamo all'istruzione giusta, questo è necessario poichè la bipush è un'istruzione composta da 2 byte, uno per il codice operativo e uno per l'operando.



<code>ndr_reg[31:0] = 0</code>	<code>00000100</code>	<code>0000000A</code>	
<code>tos_reg[31:0] = 0</code>	<code>00000101</code>	<code>0000000A</code>	
<code>wr ff=1</code>			

*ISTORE a*

*istore = 0x36:*

$H = LV$

$MAR = MBRU + H$

*istore\_cont:*

$MDR = TOS; wr$

$SP = MAR = SP - 1; rd$

$PC = PC + 1; fetch$

$TOS = MDR; goto main$

Rifacendoci al programma fornito insieme al codice del processore:

```
.main
.var
.endvar
```

*BIPUSH 0xA*

*BIPUSH 0xE*

*ISUB*

*ISTORE a*

*HALT*

*.endmethod*

```
-- RAM content
signal mem : dp_ar_ram_type := (
--BEGIN_WORDS_ENTRY
128 => "00000000000000000000000000000000",
0 => "00000001000000000000000010000000",
1 => "00001110000100000000101000010000",
2 => "10100111000000010011011001100101",
3 => "00000000000000000000000000000000",
others => (others => '0')
--END_WORDS_ENTRY
);
```

Per analizzare l'operazione di ISTORE abbiamo innanzitutto osservato quando il PC del programma puntava alla decima istruzione, quindi quando il valore era pari a 0X00000009 e osservato il valore contenuto nell'MBR (dopo il ciclo di clock necessario alla fetch) che era di fatto pari a 0x36, ovvero il codice operativo dell'istruzione in analisi. Però il salto vero è proprio all'istruzione viene fatto nel momento in cui si incrementa il PC, per cui abbiamo osservato i valori dei registri da quel momento in poi.

La prima operazione effettuata è l'inserimento del valore di LV (puntatore alla base dello stack delle variabili del programma) all'interno del registro H, per fare questa operazione il valore viene inserito come operando B nella ALU e viene eseguita l'operazione di identità con il codice operativo osservato già in precedenza.

<code>pc_reg[31:0] =0000000A</code>	<code>00000009</code>	<code>0000000A</code>
<code>h_reg[31:0] =0000000E</code>	<code>0000000E</code>	<code>00000101</code>
<code>lv_reg[31:0] =00000101</code>	<code>00000101</code>	

La ISTORE procede calcolando la posizione in memoria della variabile in cui memorizzare il valore presente sulla cima dello stack. Per cui la ALU somma il valore contenuto in MBR (che contiene il valore dell'offset della variabile rispetto al puntatore LV) il valore contenuto in H.

<code>fetch_ff=0</code>	<code>00000036</code>	<code>00000001</code>
<code>mbr_u[31:0] =00000001</code>		
<code>h_reg[31:0] =00000101</code>	<code>0000000E</code>	<code>00000101</code>
<code>lv_reg[31:0] =00000101</code>	<code>00000101</code>	
<code>operand_a[31:0] =00000101</code>	<code>0000000E</code>	<code>00000101</code>
<code>operand_b[31:0] =00000001</code>	<code>00000101</code>	<code>00000001</code>
<code>sh_result[31:0] =00000102</code>	<code>00000101</code>	<code>00000102</code>

Il valore ottenuto viene, poi, inserito nel MAR per poter accedere all'indirizzo di memoria in cui inserire il valore contenuto in TOS. Quest'ultimo deve essere inserito nel registro MDR. Ovviamente viene anche alzato il segnale di write per effettuare la scrittura vera e propria.

<code>mem_data_addr[31:0] =00000102</code>	<code>00000105</code>	<code>00000102</code>
<code>mem_data_out[31:0] =FFFFFFFC</code>	<code>FFFFFFFC</code>	
<code>wr_ff=1</code>		

Una volta memorizzato l'elemento, si decrementa lo SP, si legge il nuovo valore alla cima dello stack, si effettua l'incremento del PC e si effettua il fetch del prossimo codice operativo(poichè anche la ISTORE è su 2 byte), poi si legge il valore puntato da SP e tale valore viene inserito in TOS.

<code>pc_reg[31:0] =0000000A</code>	<code>0000000A</code>	<code>0000000B</code>
<code>sp_reg[31:0] =00000104</code>	<code>00000105</code>	<code>00000104</code>
<code>rd_ff=1</code>		
<code>tos_reg[31:0] =FFFFFFFC</code>	<code>FFFFFFFC</code>	
<code>mdr_reg[31:0] =FFFFFFFC</code>	<code>FFFFFFFC</code>	<code>00000100</code>

## Esercizio 9.b

Partendo sempre dal programma fornito, abbiamo deciso di modificare il comportamento della IADD, la quale invece che fornire la somma tra i primi due valori dello stack, fornisce la differenza, per tale ragione otteniamo il valore FFFFFFFC (differenza tra A ed E).

```
.main  
.var  
a  
.endvar  
BIPUSH 0xA  
BIPUSH 0xE  
IADD  
ISTORE a  
SWAP  
HALT  
.endmethod
```

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

La modifica effettuata si è limitata a sostituire l'operazione di sottrazione nella ALU, al posto dell'addizione, questo in binario si traduce al segnale di controllo 00111111.

operand_a[31:0]	000000E
operand_b[31:0]	000000A
sh_result[31:0]	FFFFFFFC
alu_control[7:0]	00111111

Riportiamo anche il codice binario della control store contenente le microistruzioni relative alla IADD, in cui si evidenziano i bit modificati.

Bit prima:

```
101 => "001100110000001101100000010010100100",  
102 => "00110011100000010100100000000000111",  
103 => "000000110000001111000010000101000000",
```

Bit dopo:

```
L01 => "001100110000001101100000010010100100",  
L02 => "0011001110000001010010000000000111",  
L03 => "000000110000001111110010000101000000",
```

Cambiano i bit di controllo inviati alla ALU (da *111100* a *111111*).

#### SWAP

Abbiamo anche deciso di modificare l'operazione di swap, questa inizialmente si limita ad effettuare lo scambio tra gli elementi nelle prime due posizioni dello stack. La nostra idea, invece, prevede di scambiare l'elemento in testa allo stack con la prima variabile dichiarata dal programma (questo implica che nel programma venga dichiarata una variabile per poter utilizzare tale istruzione, il che fa sì che il programma fornito si prestasse bene ad eseguire tale operazione). Inoltre il valore che dalla variabile viene memorizzato alla cima dello stack, prima di essere memorizzato viene anche negato, in modo da attivare anche un'operazione della ALU dettata dal segnale di controllo *111011*.

L'istruzione di partenza è la seguente:

*swap = 0x5F*:

*MAR = SP - 1; rd*

*MAR = SP*

*H = MDR; wr*

*MDR = TOS*

*MAR = SP - 1; wr*

*TOS = H; goto main*

Le modifiche apportate sono relative all'indirizzo da cui prelevare l'elemento da inserire alla cima dallo stack, infatti piuttosto che scambiare gli elementi presenti nelle prime due posizioni dello stack, prendiamo la prima variabile dichiarata dal programma, quindi partiamo da *LV* e incrementiamo di 1 tramite la ALU, il flusso di operazioni prosegue allo stesso modo, però quando andiamo a memorizzare il valore preso dalla variabile, prima di memorizzarlo lo neghiamo tramite l'istruzione NOT.

*swap = 0x5F*:

*MAR = LV + 1; rd*

*MAR = SP*

*H = NOT MDR; wr*

*MDR = TOS*

*MAR = LV + 1; wr*

*TOS = H; goto main*

Per osservare come avviene lo scambio riportiamo gli ingressi e le uscite della RAM, osservando che prima memorizza nella variabile il risultato della IADD (che fa la sottrazione), poi al momento dello swap memorizza il valore 0x00000100 che proviene dal TOS.

data_in_1[31:0] =0	FFFFFFFFFFFC	000000100	
data_out_1[31:0] =F	000000100	FFFFFFFFFFFC	

Dal punto di vista del registro TOS osserviamo che a valle dello swap riceve il valore proveniente dalla variabile a (FFFFFFFFFFC) però negato e quindi pari a 00000003.

mem_data_in[31:0] =F	FFFFFFFFFFFC	000000100	
tos_reg[31:0] =0	000000100	00000003	

Riportiamo anche l'operazione di negazione effettuata dalla ALU.

operand_b[31:0] =0	FFFFFFFFFFFC		
sh_result[31:0] =0	00000003		
alu_control[7:0] =0	00101100		

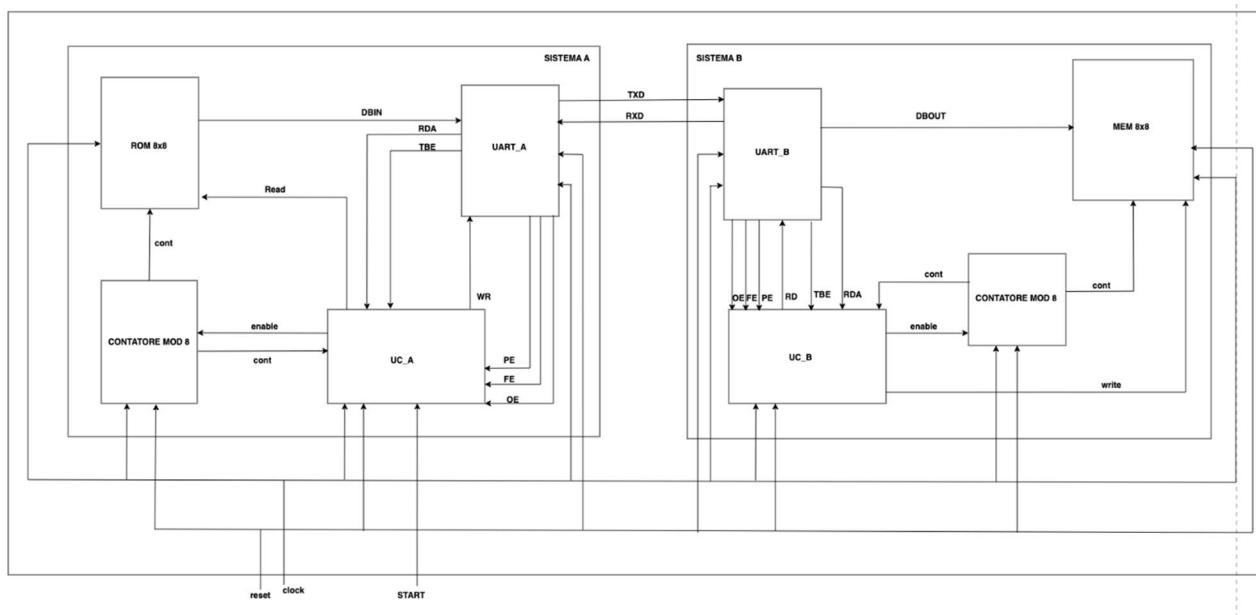
## Capitolo 6: Interfaccia Seriale

### Esercizio 10

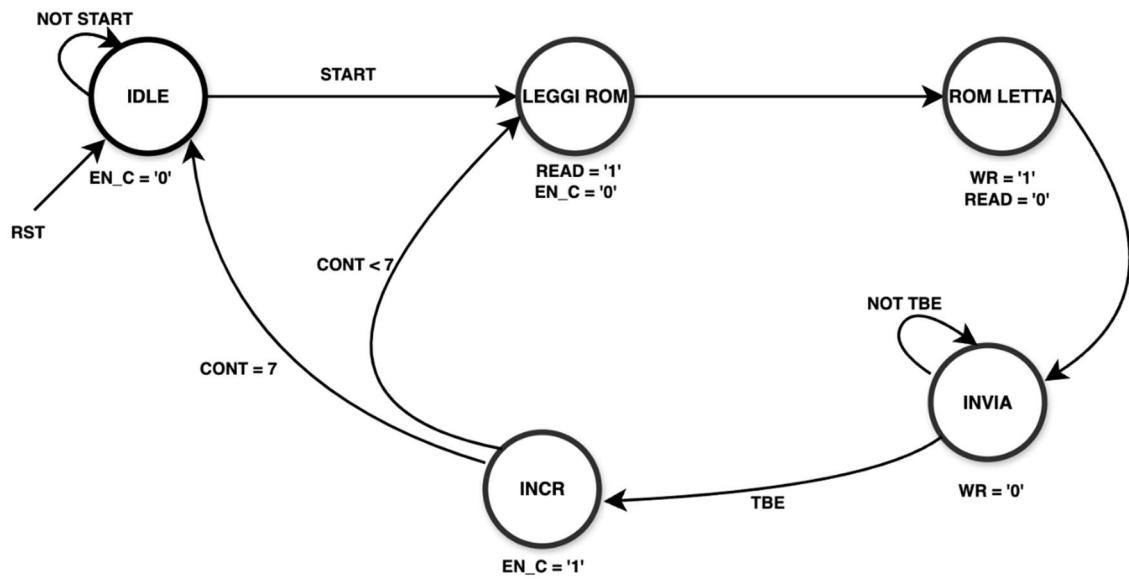
Partendo dall'implementazione fornita dalla Digilent di un dispositivo UART-RS232 (componente RS232RefComp.vhd), progettare, implementare e simulare in VHDL un sistema composto da 2 unità A e B che condividono lo stesso segnale di clock e comunicano tra loro mediante interfaccia seriale. Il sistema A contiene una ROM di 8 locazioni da 1 byte ciascuno, un contatore CONT\_A per scandire le locazioni della ROM e una UART\_A, mentre il sistema B contiene una memoria MEM di 8 locazioni da 1 byte ciascuno, un contatore CONT\_B per scandire le locazioni della MEM e una UART\_B. Quando un segnale WR viene asserito nell'unità A, viene prelevato un byte dalla ROM e inviato all'unità B, che dovrà riceverlo e salvarlo in MEM.

#### Progetto e Architettura

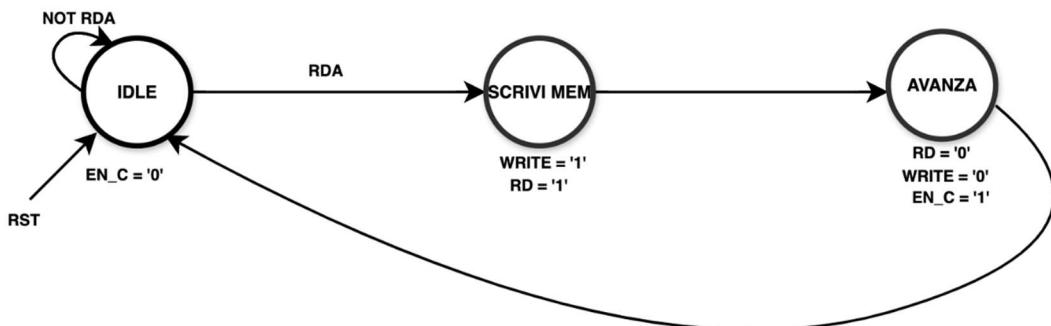
Si vuole realizzare una comunicazione asincrona seriale fra due nodi A e B utilizzando un componente di interfaccia seriale UART. Seguono gli automi a stati finiti delle due unità di controllo dei nodi A e B.



I segnali che servono alle due macchine per la comunicazione asincrona tramite UART sono: WR e TBE lato trasmettitore e RD e RDA lato ricevitore. I segnali WR e RD sono segnali di controllo in uscita dalle unità di controllo e in entrata nella UART mentre TBA e RDA sono segnali di stato in uscita dalla UART e in entrata nelle unità di controllo. Quando alzo WR significa che sto caricando lo shift register del trasmettitore con i dati da trasmettere, di conseguenza TBE mi darà la tempificazione necessaria a capire quando il trasmettitore ha terminato la trasmissione: se TBE = 1 lo shift-register è vuoto e di conseguenza posso scrivere e aspettare che si svuoti prima di inviare il dato successivo; WR=1 fa abbassare TBE e se TBE = 0 lo shift register è ancora pieno e non posso andare avanti a inviare la successiva cella di memoria. Mentre RDA = 1 indica che lo shift register del ricevitore è pieno e posso prelevare i dati alzando RD = 1. Alzare RD farà abbassare RDA; RDA = 0 significa che lo shift register non si è ancora riempito tutto.



Unità di controllo di A.



Unità di controllo di B.

## Implementazione

### MAIN- STRUCTURAL (*main.vhd*)

```
34 entity MAIN is
35     Port ( start : in STD_LOGIC;
36             clk : in STD_LOGIC;
37             reset : in STD_LOGIC);
38 end MAIN;
39
40 architecture Behavioral of MAIN is
41     component sistema_A is
42         Port ( CLK : in STD_LOGIC;
43                 START : in STD_LOGIC;
44                 RESET : in STD_LOGIC;
45                 TXD : out STD_LOGIC
46             );
47     end component;
48     component sistema_B is
49         Port ( CLK : in STD_LOGIC;
50                 RESET : in STD_LOGIC;
51                 START : in STD_LOGIC;
52                 RXD : in STD_LOGIC);
53     end component;
54
55     signal simplex:std_logic;
56
57     begin
58
59         A: sistema_A port map(clk,start,reset,simplex);
60         B:sistema_B port map (clk,reset,start,simplex);
61
62     end Behavioral
```

### SISTEMA A - STRUCTURAL (*sistema\_a.vhd*)

```
34 entity sistema_A is
35     Port ( CLK : in STD_LOGIC;
36             START : in STD_LOGIC;
37             RESET : in STD_LOGIC;
38             TXD : out STD_LOGIC
39         );
40 end sistema_A;
41
42 architecture Behavioral of sistema_A is
43     component UC_A is
44         port (
45             CLK, RST, START: in STD_LOGIC;
46             counter : in STD_LOGIC_VECTOR (2 downto 0);
47             count_en: out std_logic;
48             READ : out STD_LOGIC;
49             TBE: in STD_LOGIC;
50             WR : out STD_LOGIC
51         );
52
53     end component;
54     component ROM_N_M is generic(N:integer := 8; M:integer:=8);
55         Port ( read : in STD_LOGIC;
56                 clock : in STD_LOGIC;
57                 address : in STD_LOGIC_VECTOR (2 downto 0);
58                 cella_i_out : out STD_LOGIC_VECTOR (M-1 downto 0));
59     end component;
60     component cont_mod8 is
61         port( clock, reset: in std_logic;
62                 count_in: in std_logic;
63                 count: out std_logic_vector(2 downto 0));
64     end component;
```

```

65  component Rs232RefComp is
66    Port (
67      TXD : out std_logic := '1';
68      RXD : in std_logic;
69      CLK : in std_logic; --Master Clock
70      DBIN : in std_logic_vector (7 downto 0);--Data Bus in
71      DBOUT : out std_logic_vector (7 downto 0); --Data Bus out
72      RDA : inout std_logic; --Read Data Available(1 quando il dato Ã© disponibile nel registro rdReg)
73      TBE : inout std_logic := '1'; --Transfer Bus Empty(1 quando il dato da inviare Ã© stato caricato nello shift register)
74      RD : in std_logic; --Read Strobe(se 1 significa "leggi" --> fa abbassare RDA)
75      WR : in std_logic; --Write Strobe(se 1 significa "scrivi" --> fa abbassare TBE)
76      PE : out std_logic; --Parity Error Flag
77      FE : out std_logic; --Frame Error Flag
78      OE : out std_logic; --Overwrite Error Flag
79      RST : in std_logic := '0'); --Master Reset
80  end component;
81  --segnalet contatore -> CU per il conteggio e per l'indirizzo della ROM
82  signal count : std_logic_vector( 2 downto 0);
83  --segnalet contatore -> CU abilitazione contatore
84  signal en: std_logic;
85  --segnalet enable lettura rom
86  signal read: std_logic;
87  --segnalet UART -> CU
88  signal TBE: std_logic;
89  --segnalet CU -> UART
90  signal WR: std_logic;
91  --segnalet uscita ROM
92  signal DBIN: std logic vector(7 downto 0);

```

```

94  begin
95
96  --UnitÃ  di controllo
97  CU_A: UC_A port map(clk,reset,start,count,en,read,TBE,WR);
98
99  --ROM
100 ROM:ROM_N_M port map(read,clk,count, DBIN);
101
102 --Contatore
103 Cont: cont_mod8 port map(clk,reset,en,count);
104
105 --UART
106 Uart: Rs232RefComp port map(
107   TXD => TXD ,
108   RXD => '1', --sempre alta per indicare che non ricevo nulla
109   clk => clk,
110   DBIN => DBIN,
111   TBE => TBE,
112   RD => '0',
113   WR => WR,
114   RST => reset
115 );
116 end Behavioral;

```

UNITA DI CONTROLLO A – BEHAVIORAL (*uc\_a.vhd*)

```
entity UC_A is
  port (
    CLK, RST, START: in STD_LOGIC;
    counter : in STD_LOGIC_VECTOR (2 downto 0);
    count_en: out std_logic;
    READ : out STD_LOGIC;
    TBE: in STD_LOGIC;
    WR : out STD_LOGIC);
end UC_A;

architecture Behavioral of UC_A is
begin
  registri: process (CLK)
begin
  if rising_edge (CLK) then
    if ( RST = '1' ) then
      stato_corrente <= idle;
    else
      stato_corrente <= stato_successivo;
    end if;
  end if;
end process;
```

## SISTEMA B – STRUCTURAL (*sistema\_b.vhd*)

```

34 entity sistema_B is
35     Port ( CLK : in STD_LOGIC;
36             RESET : in STD_LOGIC;
37             START : in STD_LOGIC;
38             RXD : in STD_LOGIC);
39 end sistema_B;
40 architecture Behavioral of sistema_B is
41     component UC_B is
42         port (
43             clock, reset : in std_logic;
44             write: out std_logic;
45             en: out std_logic;
46             RDA : in STD_LOGIC;
47             RD : out STD_LOGIC
48         );
49     end component;
50     component MEM_N_M is generic(N:integer := 8; M:integer := 8);
51         Port ( write : in STD_LOGIC;
52                 clock : in STD_LOGIC;
53                 data_i_in : in STD_LOGIC_VECTOR (M-1 downto 0);
54                 address : in STD_LOGIC_VECTOR (2 downto 0));
55     end component;
56     component cont_mod8 is
57         port( clock, reset: in std_logic;
58                 count_in: in std_logic;
59                 count: out std_logic_vector(2 downto 0));
60     end component;
61     component Rs232RefComp is
62         Port (
63             TXD : out std_logic      := '1';
64             RXD : in std_logic;
65             CLK : in std_logic;          --Master Clock
66             DBIN : in std_logic_vector (7 downto 0);--Data Bus in
67             DBOUT : out std_logic_vector (7 downto 0); --Data Bus out
68             RDA : inout std_logic;        --Read Data Available(1 quando il dato Ã¢ disponibile nel registro rdReg)
69             TBE : inout std_logic := '1';    --Transfer Bus Empty(1 quando il dato da inviare Ã¢ stato caricato nello shift register)
70             RD : in std_logic;           --Read Strobe(se 1 significa "leggi" --> fa abbassare RDA)
71             WR : in std_logic;           --Write Strobe(se 1 significa "scrivi" --> fa abbassare TBE)
72             PE : out std_logic;          --Parity Error Flag
73             FE : out std_logic;          --Frame Error Flag
74             OE : out std_logic;          --Overwrite Error Flag
75             RST : in std_logic := '0';    --Master Reset
76         );
77     end architecture;

```

```

60     --segnale contatore --> CU per la conteggio e per la simulazione utile per
61     signal count : std_logic_vector( 2 downto 0);
62     --segnale contatore --> CU abilitazione contatore
63     signal en: std_logic;
64     --segnale enable lettura rom
65     signal write: std_logic;
66     --segnale UART -> CU
67     signal RDA: std_logic;
68     --segnale CU -> UART
69     signal RD: std_logic;
70     --segnale ingresso MEM
71     signal DBOUT: std_logic_vector(7 downto 0);
72     --segnali di errore
73     signal FE: std_logic;
74     signal OE: std_logic;
75     signal PE: std_logic;
76     begin
77
78     --Unità di controllo
79     CU_B: UC_B port map(clk,reset,write,en,RDA,RD);
80
81     --ROM
82     MEM:MEM_N_M port map(write,clk,DBOUT,count);
83
84     --Contatore
85     Cont: cont_mod8 port map(clk,reset,en,count);
86
87     --UART
88     Uart: Rs232RefComp port map(
89             RXD => RXD ,
90             clk => clk,
91             DBOUT => DBOUT,
92             RDA => RDA,
93             RD => RD,
94             WR => '0', -- se WR = 0 non trasmetto mai
95             RST => reset,
96             DBIN    => (others =>'0'),
97             FE => FE,
98             OE => OE,
99             PE => PE
100            );
101
102     end Behavioral;

```

## UNITA DI CONTROLLO B – BEHAVIORAL (*uc\_b.vhd*)

```
34 entity UC_B is
35     port (
36         clock, reset : in std_logic;
37         write: out std_logic;
38         en: out std_logic;
39         RDA : in STD_LOGIC;
40         RD : out STD_LOGIC
41     );
42 end UC_B;
43 architecture Behavioral of UC_B is
44     type stato is (idle, scrivi_su_mem, avanza);
45     signal stato_corrente, stato_successivo : stato := idle;
46 begin
47     registri: process (clock)
48     begin
49         if rising_edge(clock) then
50             if ( reset = '1' ) then
51                 stato_corrente <= idle;
52             else
53                 stato_corrente <= stato_successivo;
54             end if;
55         end if;
56     end process;
57     parte_combinatoria: process (stato_corrente, RDA)
58     begin
59         case stato_corrente is
60             when idle =>
61                 en <= '0';
62                 if ( RDA = '1' ) then
63                     stato_successivo <= scrivi_su_mem;
64                 else
65                     stato_successivo <= idle;
66                 end if;
67             when scrivi_su_mem =>
68                 RD <= '1';
69                 write <= '1';
70                 stato_successivo <= avanza;
71             when avanza =>
72                 RD <= '0';
73                 write <= '0';
74                 en <= '1';
75                 stato_successivo <= idle;
76         end case;

```

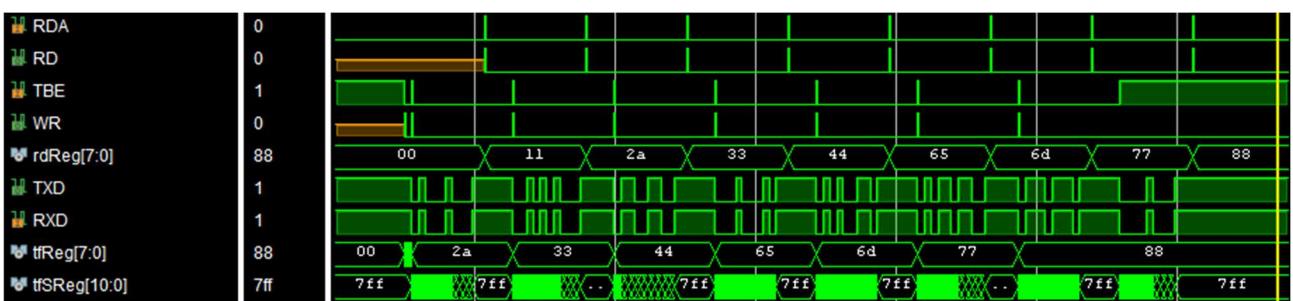
## Simulazione

Per la simulazione abbiamo fatto riferimento ad un testbench che si limita a dare un segnale di start impulsivo al sistema. Per permettere il corretto funzionamento della UART abbiamo impostato il clock su 20 ps.

Il risultato finale consiste nell'osservare la memoria di B che si riempie con i valori contenuti nella ROM A.



Questo viene effettuato gradualmente, si solleva il WR nel trasmettitore, che invia un bit per volta eseguendo uno shift dello shift register caricato con il valore inserito in DBin a cui vengono aggiunti i bit previsti dallo standard RS232. Al termine dell'invio di ogni frame si alza il segnale TBE che permette l'invio di un nuovo frame avanzando con gli indirizzi della ROM A grazie al contatore inserito. Possiamo vedere come vengono inviati i bit tra TXD e RXD.



Evoluzione degli stati nella uc\_a:



Evoluzione degli stati nella uc\_b:

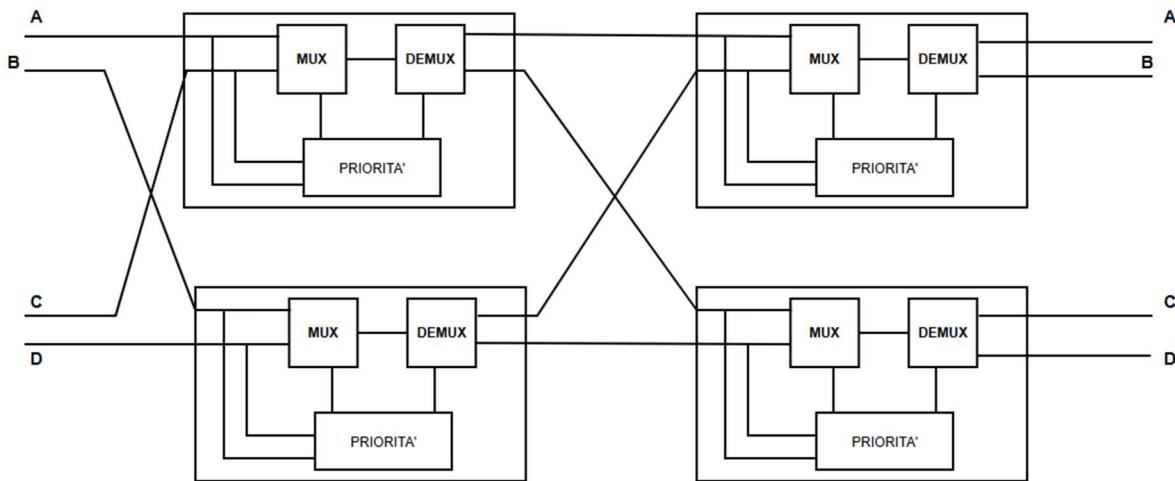


## Capitolo 7: Switch Multistadio

### Esercizio 11

Progettare ed implementare in VHDL uno switch multistadio secondo il modello omega network. Lo switch deve consentire lo scambio di messaggi di 2 bit ciascuno da un nodo sorgente a un nodo destinazione in una rete con 4 nodi, implementando uno schema a priorità fissa fra i nodi (es. nodo 1 più prioritario, con priorità decrescenti fino al nodo 4).

#### Progetto e architettura



Abbiamo 4 nodi, ciascuno dei quali può fungere da sorgente o da destinazione. Lo switch multistadio richiesto, deve consentire lo scambio di messaggi secondo il modello omega network, che per 4 nodi prevede due stadi identici interconnessi tra loro sulla base dell'algoritmo del perfect shuffling.

#### Switch

Per il progetto del singolo switch si è scelto di partire dal modello dello switch elementare che presenta due ingressi e due uscite tramite una semplice rete di interconnessione costituita da un multiplexer e un demultiplexer. La selezione del multiplexer è legata alla sorgente, mentre la selezione del demultiplexer è legata alla destinazione. È richiesto uno schema a priorità fissa tra i nodi, per cui all'interno di ogni switch è stato inserito un componente per la gestione delle priorità che sulla base degli ingressi fornisce le linee di selezione a multiplexer e demultiplexer.

Con lo switch elementare gestiamo i conflitti sulla sorgente, tuttavia, la rete omega network, presenta prestazioni migliori se questi venissero gestiti sulla destinazione. Per implementare un comportamento di questo tipo, si avrebbe bisogno di un altro tipo di switch.

#### Gestione priorità

Ogni switch, deve conoscere sorgente e destinazione del pacchetto di dati che sta inviando, per un corretto instradamento. Dunque, ai due bit data che vengono trasmessi, si è ritenuto opportuno aggiungere due bit per codificare la sorgente e due bit per codificare la destinazione (come una sorta di header). Un nodo a priorità più bassa può trasmettere solo se non sta trasmettendo attraverso lo stesso switch quello che ha priorità più alta. Per convenzione, si considera che un nodo non sta trasmettendo quando i suoi bit sorgente assumono valore unsigned. Scelto il trasmittente, la destinazione sarà quella relativa al messaggio da esso inviato. Se solo uno dei due sta trasmettendo, questo procede e non si hanno conflitti da gestire (l'altro è unsigned per cui la seconda condizione in or non viene valutata). Quando invece entrambe le sorgenti sono

valide, si ha un conflitto, allora trasmette il nodo a priorità maggiore (consideriamo la seconda condizione in or).

#### Implementazione

Multiplexer e Demultiplexer sono componenti base riportati in appendice. In particolare nel caso specifico, questi ricevono in ingresso e trasmettono in uscita segnali del tipo std\_logic\_vector(5 downto 0).

#### Gestione priorità – BEHAVIORAL (*priorita.vhd*)

```
22 | library IEEE;
23 | use IEEE.STD_LOGIC_1164.ALL;
24 |
25 |-- Uncomment the following library declaration if using
26 |-- arithmetic functions with Signed or Unsigned values
27 |use IEEE.NUMERIC_STD.ALL;
28 |
29 entity priorita is
30     Port (in1_val: in std_logic_vector(1 downto 0);
31            in2_val: in std_logic_vector(1 downto 0);
32            dest1: in std_logic;
33            dest2: in std_logic;
34            prior: out std_logic;
35            sel_demux: out std_logic );
36 end priorita;
37 |
38 architecture Behavioral of priorita is
39 begin
40
41     sel_demux<=dest2 when (in1_val="-->) or unsigned(in1_val)> unsigned(in2_val) else
42             dest1 when (in2_val="-->) or unsigned(in1_val) < unsigned(in2_val) else
43             '-';
44
45     prior <= '1' when (in1_val="-->) or unsigned(in1_val)> unsigned(in2_val) else
46             '0' when (in2_val="-->) or unsigned(in1_val) < unsigned(in2_val) else
47             '-';
48
49 end Behavioral;
```

## Switch – STRUCTURAL (*switch.vhd*)

```

12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14
15
16 --livello: livello della rete, parto a contare da 0
17 entity switch is
18     generic(livello: integer);
19     Port (in1: in std_logic_vector(5 downto 0);
20           in2: in std_logic_vector(5 downto 0);
21           out1: out std_logic_vector(5 downto 0);
22           out2: out std_logic_vector(5 downto 0) );
23 end switch;
24
25 architecture structural of switch is
26 component mux2_1 is
27     generic(N: integer);
28     port( a0 : in std_logic_vector(N-1 downto 0);
29           a1 : in std_logic_vector(N-1 downto 0);
30           s : in STD_LOGIC;
31           y : out std_logic_vector(N-1 downto 0)
32       );
33 end component;
34
35 component demux2_1 is
36     generic(N: integer);
37     Port ( a0 : in std_logic_vector(N-1 downto 0);
38           s0 : in std_logic;
39           y0 : out std_logic_vector(N-1 downto 0);
40           y1 : out std_logic_vector(N-1 downto 0));
41 end component;

```

```

12 component priorita is
13     Port (in1_val: in std_logic_vector(1 downto 0);
14           in2_val: in std_logic_vector(1 downto 0);
15           dest1: in std_logic;
16           dest2: in std_logic;
17           prior: out std_logic;
18           sel_demux: out std_logic );
19 end component;
20
21 signal sel_mux: std_logic;
22 signal sel_demux: std_logic ;
23 signal mux_demux: std_logic_vector(5 downto 0);
24 signal p: std_logic;
25
26 begin
27
28 gestione_priorita: priorita port map(in1(5 downto 4), in2(5 downto 4), in1(3-livello), in2(3-livello), sel_mux, sel_demux);
29
30 mux: mux2_1 generic map(6) port map(in1, in2, sel_mux, mux_demux);
31
32 demux: demux2_1 generic map(6) port map(mux_demux, sel_demux, out1, out2);
33
34 end structural;

```

Ad ogni livello di un'Omega Network, il bit i-esimo della stringa che codifica la destinazione è quello che funge da selezione per il demultiplexer. Per valutare tale informazione, a ciascuno switch è stata aggiunta la variabile *livello*. Questo ci consente di fornire in ingresso al componente per la gestione della priorità solo il bit necessario volta per volta.

**Codifica scelta per il messaggio:** Leggendo il vettore in ingresso dal bit meno significativo, i primi due bit rappresentano i dati da trasmettere, i successivi due la destinazione e gli ultimi due la sorgente.

## Switch Multistadio – STRUCTURAL (*rete4\_4.vhd*)

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4
5 entity rete4_4 is
6     Port (A_in: in std_logic_vector(5 downto 0);
7             B_in: in std_logic_vector(5 downto 0);
8             C_in: in std_logic_vector(5 downto 0);
9             D_in: in std_logic_vector(5 downto 0);
10            A_out: out std_logic_vector(5 downto 0);
11            B_out: out std_logic_vector(5 downto 0);
12            C_out: out std_logic_vector(5 downto 0);
13            D_out: out std_logic_vector(5 downto 0) );
14 end rete4_4;
15
16 architecture structural of rete4_4 is
17 component switch is
18 generic(livello: integer);
19     Port (in1: in std_logic_vector(5 downto 0);
20             in2: in std_logic_vector(5 downto 0);
21             out1: out std_logic_vector(5 downto 0);
22             out2: out std_logic_vector(5 downto 0) );
23 end component;
24
25 signal int1: std_logic_vector(5 downto 0);
26 signal int2: std_logic_vector(5 downto 0);
27 signal int3: std_logic_vector(5 downto 0);
28 signal int4: std_logic_vector(5 downto 0);
29
```

```
1 begin
2
3     s1_1: switch generic map(0) port map(A_in, C_in, int1, int2);
4     s1_2: switch generic map(0) port map(B_in, D_in, int3, int4);
5
6     s2_1: switch generic map(1) port map(int1, int3, A_out, B_out);
7     s2_2: switch generic map(1) port map(int2, int4, C_out, D_out);
8
9
10 end structural;
11
```

## Simulazione

Per simulare lo switch multistadio, sopra presentato, è stato usato il seguente testbench.

```
20 : library ieee;
21 : use ieee.std_logic_1164.all;
22 :
23 : entity tb_rete4_4 is
24 : end tb_rete4_4;
25 :
26 : architecture tb of tb_rete4_4 is
27 :
28 :     component rete4_4
29 :         port (A_in : in std_logic_vector (5 downto 0);
30 :               B_in : in std_logic_vector (5 downto 0);
31 :               C_in : in std_logic_vector (5 downto 0);
32 :               D_in : in std_logic_vector (5 downto 0);
33 :               A_out : out std_logic_vector (5 downto 0);
34 :               B_out : out std_logic_vector (5 downto 0);
35 :               C_out : out std_logic_vector (5 downto 0);
36 :               D_out : out std_logic_vector (5 downto 0));
37 :     end component;
38 :
39 :     signal A_in : std_logic_vector (5 downto 0);
40 :     signal B_in : std_logic_vector (5 downto 0);
41 :     signal C_in : std_logic_vector (5 downto 0);
42 :     signal D_in : std_logic_vector (5 downto 0);
43 :     signal A_out : std_logic_vector (5 downto 0);
44 :     signal B_out : std_logic_vector (5 downto 0);
45 :     signal C_out : std_logic_vector (5 downto 0);
46 :     signal D_out : std_logic_vector (5 downto 0);
47 :
```

```
53 :
54 : begin
55 :
56 :     dut : rete4_4
57 :     port map (A_in => A_in,
58 :               B_in => B_in,
59 :               C_in => C_in,
60 :               D_in => D_in,
61 :               A_out => A_out,
62 :               B_out => B_out,
63 :               C_out => C_out,
64 :               D_out => D_out);
-- :
```

```
6 :     stimuli : process
7 :     begin
8 :         -- EDIT Adapt initialization as needed
9 :         A_in <= "00" & "11" & "11";
10 :        B_in <= "01" & "00" & "11";
11 :        C_in <= (others=>'-');
12 :        D_in <= "11" & "01" & "10";
13 :
14 :        wait for 10 ns;
15 :
16 :        -- EDIT Add stimuli here
17 :        B_in <= (others=>'-');
18 :        D_in <= "11" & "01" & "10";
19 :
20 :        wait for 10 ns;
21 :
22 :        -- EDIT Add stimuli here
23 :        A_in <= (others=>'-');
24 :        B_in <= "01" & "11" & "11";
25 :        D_in <= (others=>'-');
26 :        C_in <= "10" & "11" & "11";
27 :
28 :
29 :        wait;
30 :    end process;
31 :
32 : end tb;
```

### TEST CASE 1

- Il nodo A invia al nodo D “11”.
- Il nodo B invia al nodo A “11”.
- Il nodo D invia al nodo B “10”.

B e D, trasmettono attraverso lo stesso nodo, per cui ci aspettiamo che solo la trasmissione a priorità maggiore, (ovvero quella proveniente da B) avvenga.

### TEST CASE 2:

- Il nodo A invia al nodo D “11”.
- Il nodo D invia al nodo B “10”.

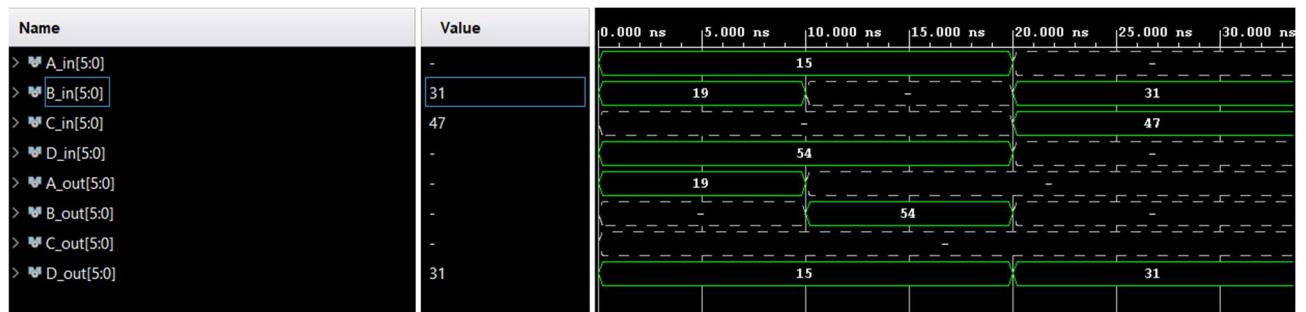
Poichè B non sta inviando nulla, stavolta, il messaggio di D viene correttamente trasmesso.

### TEST CASE 3:

- Il nodo B invia al nodo D “00”.
- Il nodo C invia al nodo D “11”.

La prima selezione del multiplexer è l'ingresso a priorità minore, la seconda quella a priorità maggiore. Vogliamo che il conflitto venga gestito correttamente e a D arrivi il messaggio inviato da B.

Di seguito l'esito della simulazione behavioral condotta.



Notiamo che il pacchetto inviato da A arriva correttamente a D, come il pacchetto inviato da B arriva ad A. Il pacchetto proveniente dal nodo D, invece, non viene inviato al nodo B. Però dopo 10 ns quando termina la trasmissione di B parte la trasmissione di D.

## Capitolo 8: Prova Esame 19/12/2024

### Esercizio 12:

Un sistema è composto da 2 nodi, A e B. A include una ROM (progettata come macchina sequenziale con READ sincrono) di 8 locazioni da 4 bit, mentre B include un sommatore parallelo in grado di effettuare la somma di 2 stringhe di 4 bit ciascuna e un registro R di 4 bit. Il sistema opera come segue: all'arrivo di un segnale di start, A inizia a prelevare gli elementi ROM[i] dalla propria memoria e li invia, uno alla volta, a B mediante handshaking. B somma progressivamente le stringhe ricevute utilizzando il sommatore e alla fine inserisce il risultato nel registro R.

1. Si disegni l'architettura complessiva del sistema tramite un diagramma a blocchi, identificando parte operativa e parte di controllo di ciascun nodo. Ogni nodo deve essere progettato seguendo un approccio strutturale, individuando tutti i componenti, le loro interfacce e le loro interconnessioni.
2. Si progettino le unità di controllo di A e B evidenziando gli stati, gli ingressi e le uscite negli automi risultanti. E' obbligatorio specificare la tempificazione che si intende dare alle macchine (fronte attivo del clock, tempificazione dei segnali di READ/WRITE su registri e memorie).
3. Si progetti il sommatore secondo un'architettura di tipo carry look ahead.
4. Si fornisca l'implementazione in VHDL dell'intero sistema e si proceda alla simulazione nel caso in cui il clock del sistema A e del sistema B siano diversi (A più lento e B più veloce)

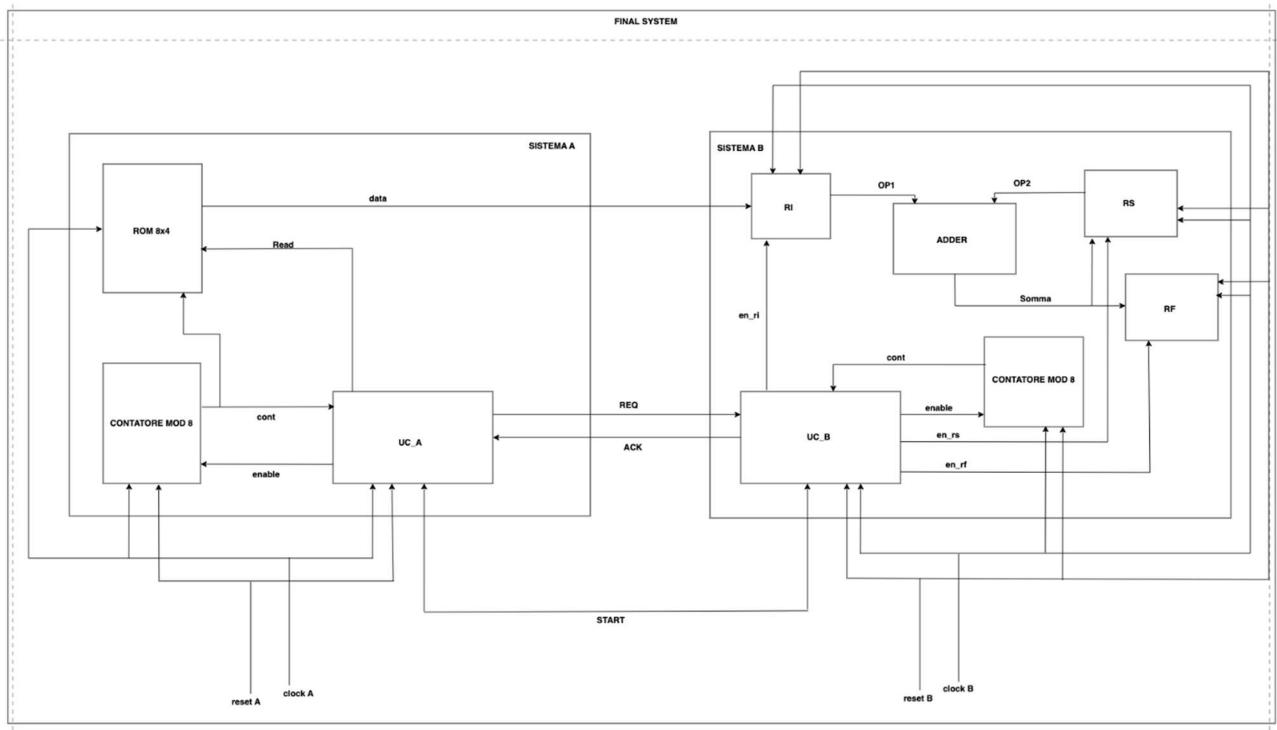
#### *Progetto e Architettura*

Dal momento che dobbiamo effettuare una somma progressiva utilizzando un Sommatore avremo bisogno di alcuni registri per rendere stabili gli operandi in ingresso alla macchina combinatoria ADDER: useremo un registro R1 per prelevare i dati in ingresso che arrivano dall'automa A e un registro RS che conterrà la somma parziale del sommatore a ogni ciclo. Al termine, la somma verrà inserita in un terzo registro RF.

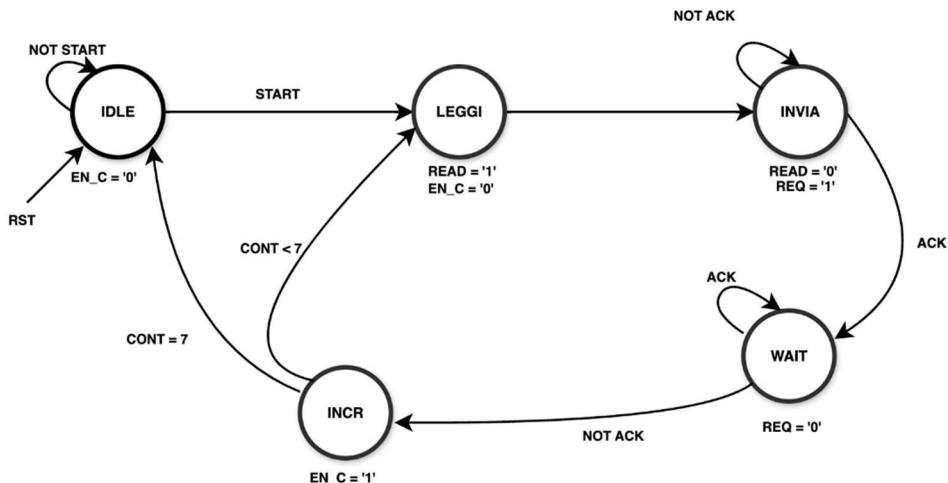
La somma finale sarà inserita in un registro a parte, chiamato RF, e per questo motivo avremo bisogno di un contatore modulo 8 all'interno della macchina B per capire quando la comunicazione è arrivata alla fine.

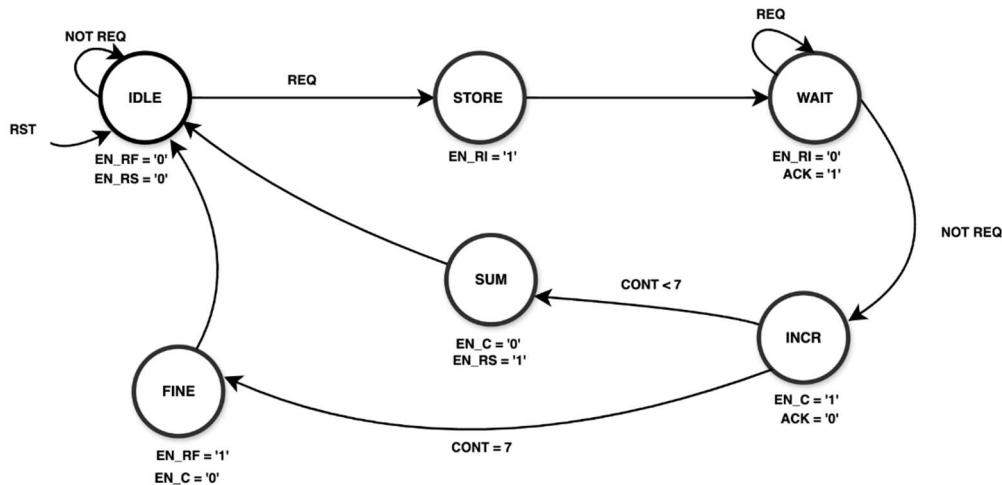
Segue l'architettura del sistema e gli automi delle due unità di controllo.

## Architettura complessiva del sistema



## Automa del nodo A.





Automa del nodo B.

Implementazione

Final System – Structural (*A\_hs\_B.vhd*)

```

14-->-->-->-->-->
15  Port (clockA: in std_logic;
16    resetA: in std_logic;
17    clockB: in std_logic;
18    resetB: in std_logic;
19    start: in std_logic
20  );
21 end A_hs_B;
22
23 architecture structural of A_hs_B is
24 component sistemaA is
25  Port (clock, reset: in std_logic;
26    data: out std_logic_vector(3 downto 0);
27    start: in std_logic;
28    req: out std_logic;
29    ack: in std_logic );
30 end component;
31
32 component sistemaB is
33  Port (clock, reset: in std_logic;
34    start: in std_logic;
35    req: in std_logic;
36    ack: out std_logic;
37    data: in std_logic_vector(3 downto 0) );
38 end component;
39
40 signal req: std_logic;
41 signal ack: std_logic;
42 signal data: std_logic_vector(3 downto 0);
43
44 begin
45
46 A: sistemaA port map(clockA, resetA, data, start, req, ack);
47
48 B: sistemaB port map(clockB, resetB,start, req, ack, data);
49
50
51 end structural;

```

## Sistema A – STRUCTURAL (*sistemaA.vhd*)

```
34 entity sistemaA is
35     Port (clock, reset: in std_logic;
36             data: out std_logic_vector(3 downto 0);
37             start: in std_logic;
38             req: out std_logic;
39             ack: in std_logic );
40 end sistemaA;
41
42 architecture structural of sistemaA is
43 component ROM is
44     Port (      clock : in STD_LOGIC;
45             address : in STD_LOGIC_VECTOR (2 downto 0);
46             read   : in STD_LOGIC;
47             data   : out STD_LOGIC_VECTOR (3 downto 0)
48         );
49 end component;
50 component cont_mod8 is
51     port( clock, reset: in std_logic;
52             count_in: in std_logic;
53             count: out std_logic_vector(2 downto 0));
54 end component;
55
56 component UC_A is
57     Port (clock: in std_logic;
58             start: in std_logic;
59             req: out std_logic;
60             ack: in std_logic;
61             read: out std_logic;
62             en: out std_logic;
63             c: in std_logic_vector(2 downto 0)
64         );
65 end component;
66 --indirizzo rom dato dal contatore
67 signal Cval: std_logic_vector(2 downto 0);
68 signal read: std_logic;
69 signal enable: std_logic;
70 begin
71
72     myROM: ROM port map(clock, Cval, read, data);
73     myCont: cont_mod8 port map(clock, reset, enable, Cval);
74     unita_controllo: UC_A port map(clock, start, req, ack, read, enable, Cval);
75 end structural;
```

## Unità di Controllo A – Behavioral (*UC\_A.vhd*)

```
34 entity UC_A is
35     Port (clock: in std_logic;
36             start: in std_logic;
37             req: out std_logic;
38             ack: in std_logic;
39             read: out std_logic;
40             en: out std_logic;
41             c: in std_logic_vector(2 downto 0)
42         );
43 end UC_A;
44
45 architecture Behavioral of UC_A is
46
47 type stato is (IDLE, Leggi, Invia, WaitState, Incr);
48 signal stato_corrente: stato := IDLE;
49 signal stato_prossimo: stato;
50
51 begin
52
53     uscita: process(stato_corrente, start, c, ack)
54     begin
55         case stato_corrente is
56             when IDLE =>
57                 en <= '0';
58                 Read <= '0';
59                 if(start='1') then
60                     stato_prossimo <= Leggi;
61                 else
62                     stato_prossimo <= IDLE;
63                 end if;
64             when Leggi =>
65                 Read <= '1';
66                 en <= '0';
67                 --stato_prossimo <= Invia;
```

```

50 :         when Invia =>
51 :             Req <= '1';
52 :             Read <= '0';
53 :             en <= '0';
54 :             if(Ack = '0') then
55 :                 stato_prossimo <= Invia;
56 :             else
57 :                 stato_prossimo <= WaitState;
58 :             end if;
59 :         when WaitState =>
60 :             Req <= '0';
61 :             en <= '0';
62 :             if(Ack = '1') then
63 :                 stato_prossimo <= WaitState;
64 :             else
65 :                 stato_prossimo <= Incr;
66 :             end if;
67 :         when Incr =>
68 :             en <= '1';
69 :             if(TO_INTEGER(unsigned(c))= 7) then
70 :                 stato_prossimo <= IDLE;
71 :             else
72 :                 stato_prossimo <= Leggi;
73 :             end if;
74 :         end case;
75 :     end process;
76 :
77 :     mem: process(clock)
78 :     begin
79 :         if(clock'event and clock='1') then
80 :             stato_corrente <=stato_prossimo;
81 :         end if;
82 :     end process;
83 :
84 : end Behavioral;

```

## Sistema B – Structural (*sistemaB.vhd*)

```

34 Y  entity sistemaB is
35 :     Port (clock, reset: in std_logic;
36 :             start: in std_logic;
37 :             req: in std_logic;
38 :             ack: out std_logic;
39 :             data: in std_logic_vector(3 downto 0) );
40 : end sistemaB;
41 :
42 : architecture structural of sistemaB is
43 : component registro4bit is
44 : port( A: in std_logic_vector(3 downto 0);
45 :           clk, res, load: in std_logic;
46 :           B: out std_logic_vector(3 downto 0));
47 : end component;
48 :
49 : component CLA_Adder is
50 : generic(N : integer := 4);
51 :     Port (A: in std_logic_vector(N-1 downto 0);
52 :             B: in std_logic_vector(N-1 downto 0);
53 :             c0: in std_logic;
54 :             carry: out std_logic;
55 :             Sum: out std_logic_vector(N-1 downto 0)
56 : );
57 : end component;
58 :
59 : component UC_B is
60 :     Port (clock: in std_logic;
61 :             start: in std_logic;
62 :             req: in std_logic;
63 :             en_c: out std_logic;
64 :             count : in std_logic_vector(2 downto 0);
65 :             ack: out std_logic;
66 :             enRI: out std_logic;
67 :             enRS: out std_logic;
68 :             enRF: out std_logic );
69 : end component;
70 : component cont_mod8 is
71 :     port( clock, reset: in std_logic;
72 :             count_in: in std_logic;
73 :             count: out std_logic_vector(2 downto 0));
74 : end component;

```

```

77 : --segnali interne e segnali di controllo
78 : signal enableRS: std_logic;
79 : signal enableRF: std_logic;
80 : signal dataRI: std_logic_vector(3 downto 0);
81 : signal dataRS: std_logic_vector(3 downto 0);
82 : signal dataRF: std_logic_vector(3 downto 0);
83 : signal somma: std_logic_vector(3 downto 0);
84 : signal carry: std_logic;
85 : signal count_in: std_logic;
86 : signal count: std_logic_vector(2 downto 0);
87 : begin
88 :   --contatore per riconoscere ultima somma
89 :   myCont: cont_mod8 port map (clock,reset,count_in,count);
90 :   --registro dati in ingresso
91 :   RI: registro4bit port map(data, clock, reset, enableRI, dataRI);
92 :   --registro su ramo di feedback
93 :   RS: registro4bit port map(somma, clock, reset, enableRS, dataRS);
94 :   --registro valore finale
95 :   RF: registro4bit port map(somma, clock, reset, enableRF, dataRF);
96 :   --sommatore CLA
97 :   adder: CLA_adder generic map(4) port map(dataRI, dataRS, '0', carry, somma);
98 :   unita_controllo: UC_B port map(clock,start, req, count_in, count, ack, enableRI, enableRS, enableRF)
99 :

```

## Unità di Controllo B – Behavioral (UC\_B.vhd)

```

14 entity UC_B is
15   Port (clock: in std_logic;
16         start: in std_logic;
17         req: in std_logic;
18         en_c: out std_logic;
19         count : in std_logic_vector(2 downto 0);
20         ack: out std_logic;
21         enRI: out std_logic;
22         enRS: out std_logic;
23         enRF: out std_logic );
24 end UC_B;
25
26 architecture Behavioral of UC_B is
27
28 type stato is (IDLE, Store,WaitState,Incr,Somma,Fine);
29 signal stato_corrente: stato := IDLE;
30 signal stato_prossimo: stato;
31
32 begin
33
34 uscita: process(stato_corrente, req)
35 begin
36   case stato_corrente is
37     when IDLE =>
38       ack <= '0';
39       enRI <= '0';
40       enRS <= '0';
41       enRF <= '0';
42       en_c <= '0';
43       if(req='1') then
44         stato_prossimo <= Store;
45       else
46         stato_prossimo <= IDLE;
47       end if;
48     when Store =>
49       enRI <= '1';
50       stato_prossimo <=WaitState;
51   end case;
52 end process;
53
54 end Behavioral;

```

```

71      when WaitState =>
72          enRI <= '0';
73          ack<='1';
74          if(req = '1') then
75              stato_prossimo <= WaitState;
76          else
77              stato_prossimo <=Incr;
78          end if;
79      when Incr =>
80          en_c <= '1';
81          if (TO_INTEGER(unsigned(count))= 7) then
82              stato_prossimo <= Fine;
83          else
84              stato_prossimo <= Somma;
85          end if;
86      when Somma =>
87          en_c <= '0';
88          enRS <= '1';
89          stato_prossimo <= IDLE;
90      when Fine =>
91          en_c <= '0';
92          enRF <= '1';
93          stato_prossimo <= IDLE;
94      end case;
95  end process;
96
97  mem: process(clock)
98  begin
99      if(clock'event and clock='1') then
100         stato_corrente <=stato_prossimo;
101     end if;
102  end process;
103
104
105
106 end Behavioral;

```

## CLA ADDER- Structural (CLA\_adder.vhd)

```

34  entity CLA_adder is
35      generic(N : integer := 4);
36          Port (A: in std_logic_vector(N-1 downto 0);
37                  B: in std_logic_vector(N-1 downto 0);
38                  c0: in std_logic;
39                  carry: out std_logic;
40                  Sum: out std_logic_vector(N-1 downto 0)
41                  );
42  end CLA_adder;
43
44  architecture structural of CLA_adder is
45  component myFA is
46  port(
47      OP_A: in std_logic;
48      OP_B: in std_logic;
49      CIN: in std_logic;
50
51      S: out std_logic;
52      COUT: out std_logic
53  );
54  end component;
55
56  component CLA is
57      generic(N : integer := 4);
58          Port (A: in std_logic_vector(N-1 downto 0);
59                  B: in std_logic_vector(N-1 downto 0);
60                  c0: in std_logic;
61                  C: out std_logic_vector(N-1 downto 0);
62                  carry: out std_logic
63                  );
64  end component;
65
66  signal FAcarry: std_logic_vector(N-1 downto 0);
67  signal C cla: std logic vector(N-1 downto 0);

```

```

65 begin
66
67 FA_0_to_N_1: for i in 0 to N-1 generate
68
69     FA: myFA port map(
70         OP_A => A(i),
71         OP_B => B(i),
72         CIN => C_cla(i),
73         S => Sum(i),
74         COUT => FAcarry(i)
75     );
76
77 end generate FA_0_to_N_1;
78
79 carry_look_ahead: CLA generic map(4) port map(A, B, c0, C_cla, carry);
80
81 --d -----
82
83
84
85

```

## CarryLookahead – Behavioral (*CLA.vhd*)

```

34 entity CLA is
35 generic(N : integer := 4);
36     Port (A: in std_logic_vector(N-1 downto 0);
37             B: in std_logic_vector(N-1 downto 0);
38             c0: in std_logic;
39             C: out std_logic_vector(N-1 downto 0);
40             carry: out std_logic
41         );
42 end CLA;
43
44 architecture Behavioral of CLA is
45 signal P: std_logic_vector(N-1 downto 0);
46 signal G: std_logic_vector(N-1 downto 0);
47 signal buffc: std_logic_vector(N-1 downto 0);
48
49 begin
50     P(0) <= A(0) xor B(0);
51     P(1) <= A(1) xor B(1);
52     P(2) <= A(2) xor B(2);
53     P(3) <= A(3) xor B(3);
54
55     G(0) <= A(0) and B(0);
56     G(1) <= A(1) and B(1);
57     G(2) <= A(2) and B(2);
58     G(3) <= A(3) and B(3);
59
60     buffc(0)<=c0;
61     buffc(1) <= G(0) or (P(0) and buffc(0));
62     buffc(2) <= G(1) or (P(1) and buffc(1));
63     buffc(3) <= G(2) or (P(2) and buffc(2));
64     Carry <= G(3) or (P(3) and buffc(3));
65
66     c<=buffc;
67
68 end Behavioral;

```

## FULL ADDER- DATAFLOW (*FA.vhd*)

```

44 entity myFA is
45     port(
46         OP_A: in std_logic;
47         OP_B: in std_logic;
48         CIN: in std_logic;
49
50         S: out std_logic;
51         COUT: out std_logic
52     );
53 end myFA;
54
55 architecture dataflow of myFA is
56 begin
57     S <= (OP_A xor OP_B) xor CIN;
58     COUT <= (OP_A and OP_B) or (CIN and(OP_A or OP_B));
59
60 end dataflow;

```

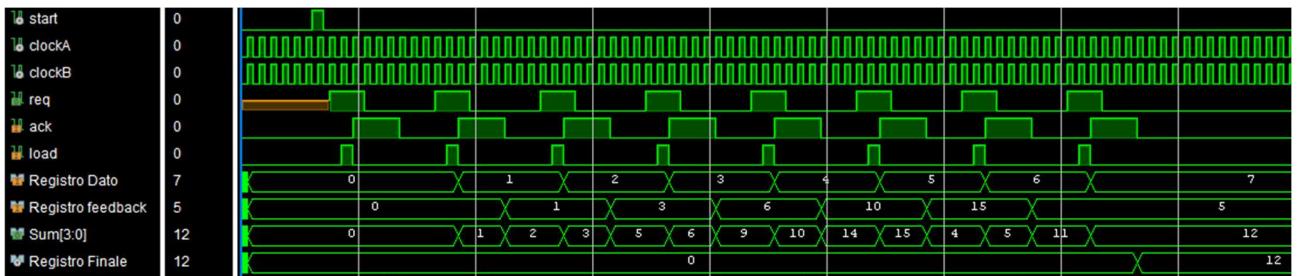
### Simulazione

Per testare il corretto funzionamento e la corretta comunicazione tra le componenti, durante la simulazione sono stati messi in evidenza i segnali di ACK, REQ, il registro dato della macchina B dove arrivano i dati, il registro di feedback per effettuare la somma e il registro finale dove verrà memorizzata l'ultima somma.

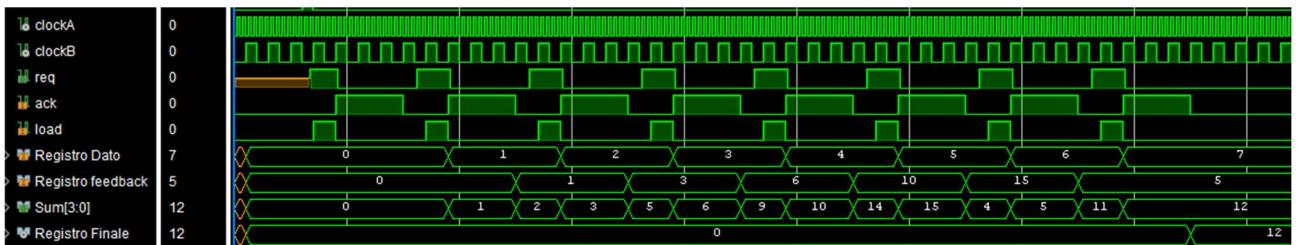
La simulazione è stata effettuata testando con lo stesso clock, clock di A più veloce e clock di A più lento; per tale motivo sono mostrati anche i clock di A e di B.

Stesso clock:

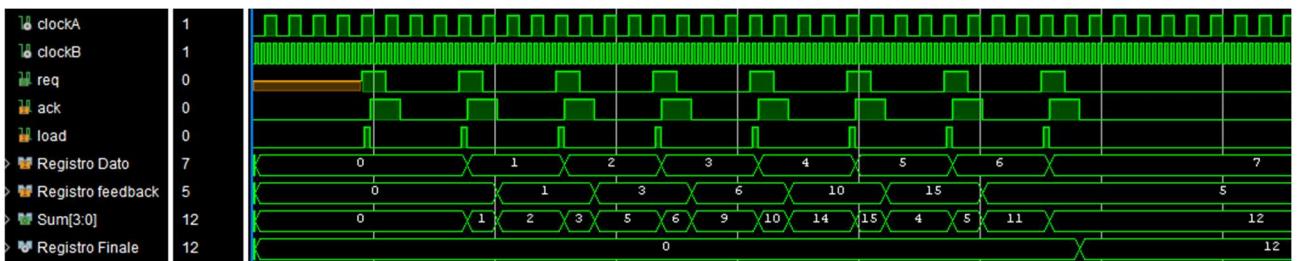
Nel Registro Dato arrivano i valori presenti nella Rom di A ossia valori da 0 a 7, per cui nel registro finale dovremo trovarci la somma di questi valori in modulo 16 (poiché il registro è di 4 bit e non teniamo conto dei riporti). ( $28 \bmod 16 = 12$ ).



Clock A più veloce:



Clock A più lento del Clock B:

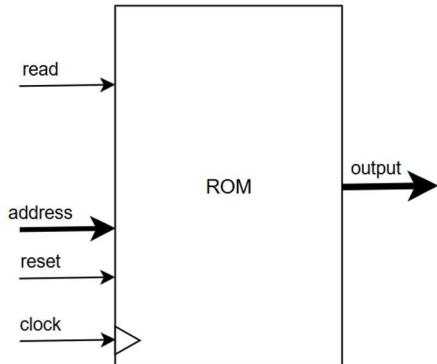


## Appendice

Di seguito si riportano i componenti base utilizzati nei progetti presentati.

### ROM

Progetto e architettura



### Implementazione

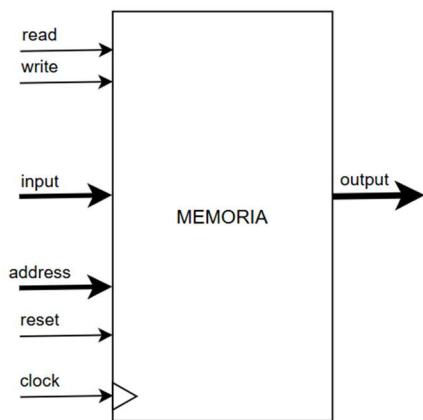
```
55 entity rom_n is
56     Generic (n : positive := 2; -- numero di bit necessari all'indirizzamento -1
57     l: positive := 7; -- numero di locazioni - 1
58     m: positive := 7); -- numero di bit per locazione -1
59     Port ( read : in STD_LOGIC;
60             clock : in STD_LOGIC;
61             reset: in STD_LOGIC;
62             address : in STD_LOGIC_VECTOR (n downto 0);
63             output : out STD_LOGIC_VECTOR (m downto 0));
64 end rom_n;
```

```
56 architecture behavioral of rom_n is
57
58     type rom_type is array (0 to 1) of std_logic_vector(m downto 0);
59
60     constant rom : rom_type := (
61         X"AA",
62         X"BB",
63         X"CC",
64         X"DD",
65         X"EE",
66         X"11",
67         X"22",
68         X"33"
69     );
```

```
61     begin
62
63         process(clock)
64         begin
65
66             if rising_edge(clock) then
67
68                 if reset = '1' then
69
70                     output <= "00000000";
71
72                 elsif read = '1' then
73
74                     output <= rom(conv_integer(address));
75
76             end if;
77
78         end if;
```

# MEMORIA

## Progetto e architettura

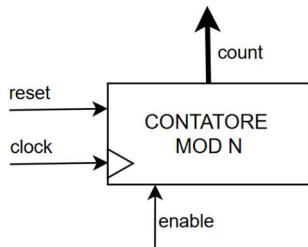


## Implementazione

```
35 entity memoria_n is
36     Generic (n : positive := 2; -- numero di bit necessari all'indirizzamento -1
37     l: positive := 7; -- numero di locazioni -1
38     m: positive := 7); -- numero di bit per locazione -1
39     Port ( read : in STD_LOGIC;
40             write : in STD_LOGIC;
41             clock : in STD_LOGIC;
42             reset : in STD_LOGIC;
43             input : in STD_LOGIC_VECTOR (m downto 0);
44             output : out STD_LOGIC_VECTOR (m downto 0);
45             address : in STD_LOGIC_VECTOR (n downto 0));
46 end memoria_n;
47
48 architecture Behavioral of memoria_n is
49
50     type mem_type is array (0 to 1) of std_logic_vector(m downto 0);
51     signal mem : mem_type := (others =>"0000");
52
53     begin
54
55         process(clock)
56         begin
57             if rising_edge(clock) then
58
59                 if reset = '1' then
60                     output <= mem(conv_integer("000"));
61                 elsif read = '1' then
62                     output <= mem(conv_integer(address));
63                 elsif write = '1' then
64                     mem(conv_integer(address)) <= input;
65             end if;
66         end if;
67     end process;
68
69 end Behavioral;
```

## CONTATORE

Progetto e architettura



Implementazione

```
34 entity counter_n is
35   Generic (n : positive := 2); -- numero di bit dell'indirizzo -1
36   Port ( clock : in STD_LOGIC;
37           reset : in STD_LOGIC;
38           enable : in STD_LOGIC;
39           count : out STD_LOGIC_VECTOR (n downto 0));
40 end counter_n;
41
42 architecture Behavioral of counter_n is
43
44   signal c : std_logic_vector(n downto 0) := (others => '0');
45   begin
46
47   process(clock)
48   begin
49     begin
50
51       if rising_edge(clock) then
52         if reset = '1' then
53           c <= (others => '0');
54         elsif enable = '1' then
55           c <= std_logic_vector(unsigned(c) + 1);
56         end if;
57       end if;
58     end if;
59   end process;
60
61   count <= c;
62
63 end Behavioral;
```

Gli altri componenti utilizzati come “componenti base”, sono stati ritenuti tali in quanto la relativa implementazione è stata fornita dai docenti del corso. Si fa riferimento in tal modo a:

- **MUX\_2\_1**
- **DEMUX\_1\_2**
- **REGISTRO**
- **BUTTON DEBOUNCER**
- **DISPLAY A 7 SEGMENTI**
- **RIPPLE CARRY ADDER**