

# 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:

Alessandro Campanella M63001697

Cesare Pulcrano M63001243

## Sommario

<b>Capitolo 1: Reti combinatorie elementari</b>	1
Esercizio 1: Multiplexer 16:1	1
Esercizio 1.1	1
Progetto e architettura	1
Implementazione	1
Simulazione	1
Esercizio 1.2	2
Progetto e architettura	4
Implementazione	4
Simulazione	4
Esercizio 1.3	6
Sintesi su board di sviluppo	6
Esercizio 2: Sistema ROM + M	7
Esercizio 2.1	9
Progetto e architettura	9
Implementazione	9
Simulazione	9
Esercizio 2.2	9
Sintesi su board di sviluppo	9
<b>Capitolo 2: Reti sequenziali elementari</b>	10
Esercizio 3: Riconoscitore di sequenze	10
Esercizio 3.1	10
Progetto e architettura	10
Implementazione	10
Simulazione	10
Esercizio 3.2	10
Sintesi su board di sviluppo	10
Esercizio 4: Shift register	10
Esercizio 4.1	10
Progetto e architettura	11
Implementazione	11
Simulazione	11
Esercizio 5: Cronometro	11
Esercizio 5.1	11
Progetto e architettura	11

Implementazione .....	11
Simulazione.....	11
Esercizio 5.2 .....	11
Sintesi su board di sviluppo .....	11
Esercizio 5.3 (solo 9 CFU).....	12
Sintesi su board di sviluppo .....	12
Esercizio 6: Sistema di lettura-elaborazione-scrittura PO_PC.....	12
Esercizio 6.1 .....	12
Progetto e architettura .....	12
Implementazione .....	12
Simulazione.....	12
Esercizio 6.2 .....	12
Sintesi su board di sviluppo .....	12
<b>Capitolo 3: Macchine aritmetiche .....</b>	<b>13</b>
Esercizio 7: Moltiplicatore di Booth .....	13
Esercizio 7.1 .....	13
Progetto e architettura .....	13
Implementazione .....	13
Simulazione.....	13
Esercizio 7.2 .....	13
Sintesi su board di sviluppo .....	13
Esercizio 7BIS: Divisore Non-Restoring (solo 9 CFU) .....	13
Esercizio 7BIS.1 .....	13
Progetto e architettura .....	13
Implementazione .....	13
Simulazione.....	13
Esercizio 7BIS.2 .....	14
Sintesi su board di sviluppo .....	14
<b>Capitolo 4: Comunicazione con handshaking .....</b>	<b>15</b>
Esercizio 8: Comunicazione con handshaking .....	15
Esercizio 8.1 .....	15
Progetto e architettura .....	15
Implementazione .....	15
Simulazione.....	15
<b>Capitolo 5: Processore.....</b>	<b>16</b>
Esercizio 9: Processore IJVM.....	16

Progetto e architettura .....	16
Implementazione .....	16
Simulazione.....	16
<b>Capitolo 6: Interfaccia seriale .....</b>	<b>17</b>
Esercizio 10: Interfaccia UART .....	17
Progetto e architettura .....	17
Implementazione .....	17
Simulazione.....	17
Esercizio 10BIS: Interfaccia UART (solo 9 CFU) .....	17
Sintesi su board di sviluppo .....	17
<b>Capitolo 7: Switch multistadio.....</b>	<b>18</b>
Esercizio 11: Switch multistadio .....	18
Esercizio 11.1 .....	18
Progetto e architettura .....	18
Implementazione .....	18
Simulazione (?).....	18
Esercizio 11.2 (solo 9 CFU).....	18
Progetto e architettura .....	18
Implementazione .....	18
Simulazione (?).....	18
Esercizio 11.3 (solo 9 CFU).....	18
Progetto e architettura .....	18
Implementazione .....	19
Simulazione (?).....	19
<b>Capitolo 8: Esercizio prova di esame dicembre 2024 .....</b>	<b>20</b>
Esercizio 12: Prova di esame del 19 dicembre 2024 .....	20
Progetto e architettura .....	20
Implementazione .....	20
Simulazione (?).....	20
<b>Appendice .....</b>	<b>21</b>
Multiplexer 4:1.....	21
Progetto e architettura .....	21
Implementazione .....	21
Button Debouncer.....	21
Progetto e architettura .....	21
Implementazione .....	22

Divisore di frequenza .....	22
Progetto e architettura .....	23
Implementazione .....	23

# Capitolo 1: Reti combinatorie elementari

## Esercizio 1: Multiplexer 16:1

### 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**.

#### Progetto e architettura

Per realizzare un multiplexer indirizzabile 16:1, sono stati combinati cinque multiplexer 4:1 ([vedi appendice](#)) seguendo un approccio **Structural**.

Un MUX 16:1 convoglia una delle 16 linee di ingresso verso un'unica uscita. Seguendo un'architettura strutturale *ad albero*, si è reso necessario utilizzare quattro MUX 4:1 al primo livello della struttura, in modo da realizzare i 16 input, e un MUX 4:1 al secondo livello per raccogliere le quattro uscite del livello precedente e convogliare solo una di esse verso l'uscita finale.

[screen linter]

#### Implementazione

Gli ingressi sono stati formattati come uno `std_logic_vector` di 16 locazioni, le selezioni e l'output invece sono dei semplici input `std_logic`.

```
entity mux_16_1 is
  Port ( x   : in   STD_LOGIC_VECTOR (15 downto 0);
        s0  : in   STD_LOGIC;
        s1  : in   STD_LOGIC;
        s2  : in   STD_LOGIC;
        s3  : in   STD_LOGIC;
        y0  : out  STD_LOGIC
  );
end mux_16_1;

architecture Structural of mux_16_1 is

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

  component mux_4_1
    port( a0 : in  STD_LOGIC;
          a1 : in  STD_LOGIC;
          a2 : in  STD_LOGIC;
          a3 : in  STD_LOGIC;
          s0 : in  STD_LOGIC; -- selezione
          s1 : in  STD_LOGIC; -- selezione
          y  : out STD_LOGIC
    );
  end component;

begin

  mux0 : mux_4_1
    Port map( a0 => x(0),
              a1 => x(1),
              a2 => x(2),
              a3 => x(3),
```

```

        s0 => s0,
        s1 => s1,
        y => u0
    );

mux1 : mux_4_1
    Port map(
        a0 => x(4),
        a1 => x(5),
        a2 => x(6),
        a3 => x(7),
        s0 => s0,
        s1 => s1,
        y => u1
    );

mux2 : mux_4_1
    Port map(
        a0 => x(8),
        a1 => x(9),
        a2 => x(10),
        a3 => x(11),
        s0 => s0,
        s1 => s1,
        y => u2
    );

mux3 : mux_4_1
    Port map(
        a0 => x(12),
        a1 => x(13),
        a2 => x(14),
        a3 => x(15),
        s0 => s0,
        s1 => s1,
        y => u3
    );

mux4 : mux_4_1
    Port map(
        a0 => u0,
        a1 => u1,
        a2 => u2,
        a3 => u3,
        s0 => s2,
        s1 => s3,
        y => y0
    );
end Structural;

```

### Simulazione

Per simulare il componente, è bastato realizzare un vettore di input e uno di controllo attraverso dei signal. Aggiornando i segnali nella testbench, si è potuto osservare, in un diagramma temporale di simulazione, che il MUX 16:1 strutturale porta in output il dato richiesto dalle linee di selezione.

```

entity mux_16_1_tb is
end mux_16_1_tb;

architecture behavioral of mux_16_1_tb is

    component mux_16_1

        port(
            x    : in  std_logic_vector (15 downto 0);
            s0   : in   STD_LOGIC;
            s1   : in   STD_LOGIC;
            s2   : in   STD_LOGIC;
            s3   : in   STD_LOGIC;
            y0   : out  STD_LOGIC
        );
    end component;

```

```

signal input      : std_logic_vector (15 downto 0) := (others => 'U');
signal control    : std_logic_vector (3 downto 0) := (others => 'U');
signal output     : std_logic := 'U';

begin

  uut: entity work.mux_16_1(Structural)
    port map(
      x => input,
      s0 => control(0),
      s1 => control(1),
      s2 => control(2),
      s3 => control(3),
      y0 => output
    );

  stim_proc: process
  begin

    wait for 100 ns;

    input <= "1000000010000000";

    wait for 50 ns;
    control <= "0000";

    wait for 50 ns;
    control <= "1111";
    --control <= "1111"; -- 13 in binario reverse

    wait for 50 ns;
    control <= "0111";

    wait for 50 ns;

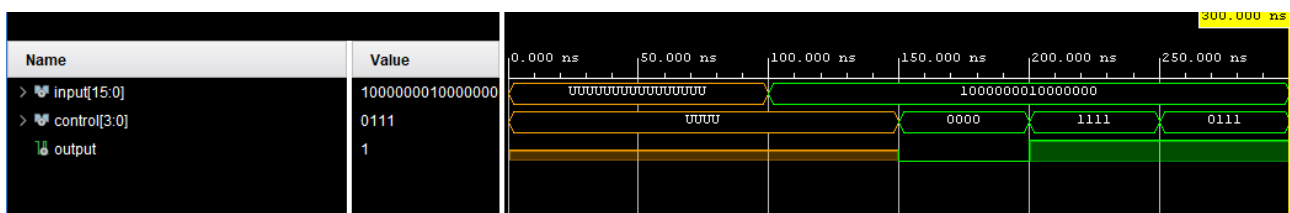
    assert output = '0'
    report "error"
    severity failure;

    wait;
  end process;
end;

```

I risultati della simulazione sono riportati di seguito: l'output diventa alto solo nell'istante in cui si richiede al componente di convogliare il dato alla linea di input "1111" (15), sulla quale è presente un segnale alto [200.000 ns]. Analogo comportamento avviene per il dato alla linea "0111" (8) [250.000 ns].

Nel momento in cui, invece, si richiede il dato alla linea "0000" (0), l'uscita è bassa, in quanto il dato trasportato in output è proprio il valore logico "0" [150.000 ns].





## 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

Per trasportare un dato, scelto tra 16 sorgenti lungo una delle 4 destinazioni, è necessario realizzare un'architettura **Structural** che unisca un MUX 16:1, realizzato [al punto precedente](#), e un DEMUX 1:4, un nuovo componente.

Un **demultiplexer indirizzabile 1:4** è un componente puramente combinatorio che, a differenza del MUX, convoglia l'unico dato disponibile in input lungo una delle quattro uscite, selezionate opportunamente mediante un ingresso di selezione.

La rete di interconnessione, dunque, è formata da un MUX 16:1 (16 input e 4 linee di selezione) e un DEMUX 1:4 (4 output e 2 linee di selezione). L'uscita del MUX è collegata all'ingresso del DEMUX.

### [screen linter]

### Implementazione

Il demultiplexer indirizzabile è stato realizzato seguendo un approccio **Behavioral**: **[output U e non -]**

```
entity demux_1_4 is
  port(
    x   : in  std_logic;
    s0  : in  std_logic;
    s1  : in  std_logic;
    y   : out std_logic_vector (3 downto 0)
  );
end demux_1_4;

architecture Behavioral of demux_1_4 is
  begin
    process(x,s1,s0)
    begin
      y<="0000";
      if(s1='0' and s0 = '0') then
        y(0)<=x;
      elsif(s1='0' and s0 = '1') then
        y(1)<=x;
      elsif(s1='1' and s0 = '0') then
        y(2)<=x;
      elsif(s1='1' and s0 = '1') then
        y(3)<=x;
      else
        y<="----";
      end if;
    end process;
  end Behavioral;
```

Nel componente principale, invece (approccio **Structural**), tutti gli ingressi, le selezioni e le uscite sono state realizzate come `std_logic_vector`. È stato realizzato, inoltre, un signal `demux_in` per collegare fisicamente l'uscita del multiplexer 16:1 all'ingresso del demultiplexer 1:4.

```
entity net_16_4 is
  Port (
    input       : in  std_logic_vector(15 downto 0);
    input_control : in  std_logic_vector (3 downto 0);
    output_control : in  std_logic_vector (1 downto 0);
    output      : out std_logic_vector (3 downto 0)
  );
end net_16_4;
```

```

architecture Structural of net_16_4 is
    signal demux_in : std_logic := '0';

    component mux_16_1
        port(
            x : in std_logic_vector (15 downto 0);
            s0 : in std_logic;
            s1 : in std_logic;
            s2 : in std_logic;
            s3 : in std_logic;
            y0 : out std_logic
        );
    end component;

    component demux_1_4
        port(
            x : in std_logic;
            s0 : in std_logic;
            s1 : in std_logic;
            y : out std_logic_vector (3 downto 0)
        );
    end component;

begin

    mux0: mux_16_1
        Port map(
            x=>input,
            s0=>input_control(0),
            s1=>input_control(1),
            s2=>input_control(2),
            s3=>input_control(3),
            y0=>demux_in
        );

    demux0: demux_1_4
        Port map(
            x=>demux_in,
            s0=>output_control(0),
            s1=>output_control(1),
            y=>output
        );

end Structural;

```

### Simulazione

Per simulare il componente, si è proceduto in maniera analoga all'[esercizio precedente](#): sono stati creati dei segnali per simulare ingressi, selezioni e uscite.

```

entity net_16_4_tb is
end net_16_4_tb;

architecture Behavioral of net_16_4_tb is

    component net_16_4
        Port (
            input : in std_logic_vector (15 downto 0);
            input_control : in std_logic_vector (3 downto 0);
            output_control : in std_logic_vector (1 downto 0);
            output : out std_logic_vector (3 downto 0)
        );
    end component;

    signal input_test : std_logic_vector (15 downto 0) := (others=>'U');
    signal input_control_test : std_logic_vector (3 downto 0) := (others=>'U');
    signal output_control_test : std_logic_vector (1 downto 0) := (others=>'U');
    signal output_test : std_logic_vector (3 downto 0) := (others=>'U');

begin

    uut: entity work.net_16_4(Structural)
        Port map(
            input=>input_test,

```

```

        input_control=>input_control_test,
        output_control=>output_control_test,
        output=>output_test
    );

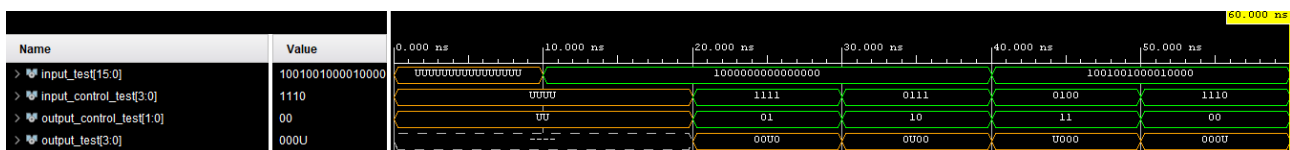
stim_proc: process
begin

    wait for 10 ns;

    input_test<="1000000000000000";
    wait for 10 ns;
    input_control_test<="1111";
    output_control_test<="01";
    wait for 10 ns;
    input_control_test<="0111";
    output_control_test<="10";
    wait for 10 ns;
    input_test<="1001001000010000";
    input_control_test<="0100";
    output_control_test<="11";
    wait for 10 ns;
    input_control_test<="1110";
    output_control_test<="00";
    wait for 10 ns;
    assert output_test="0000";
    report "errore"
    severity failure;
end process;
end Behavioral;

```

[rifai]



### 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.

#### Sintesi su board di sviluppo

Per realizzare il componente sulla board FPGA fornita, è necessario fornire, in aggiunta ai segnali visti [al punto precedente](#), il **segnale di clock** della board. Infatti, a differenza di switch e LED che possono essere codificati anche in una logica puramente combinatoria, i bottoni hanno bisogno di un fronte del clock (salita o discesa) al fine di riconoscere il livello logico (basso/alto) del bottone.

La gestione del riconoscimento del livello del bottone è affidata a un Button Debouncer, discusso nell'[appendice](#).

Utilizzando i primi 8 switch, si procede come segue:

- al clic di BTNL (bottone sinistro) i valori degli switch verranno salvati come i primi 8 LSB della stringa in input alla rete di interconnessione;
- al clic di BTNR (bottone destro) i valori degli switch verranno salvati come gli ultimi 8 MSB della stringa in input alla rete di interconnessione;

Per fare ciò, è stato pensato un componente **Behavioral** switch\_8\_capture, mostrato di seguito:

```
entity switch_8_capture is
    Port (
        CLOCK      : in std_logic; -- Clock signal
        BTN         : in std_logic; -- Button input
        SWITCH_INPUT : in std_logic_vector(7 downto 0); -- 8 input
        OUTPUT      : out std_logic_vector(7 downto 0) -- Input + selezioni
    );
end switch_8_capture;

architecture Behavioral of switch_8_capture is
begin
    process(CLOCK, BTN, SWITCH_INPUT)
    begin
        if (rising_edge(CLOCK)) then
            if(BTN = '1') then -- Click del bottone
                OUTPUT(7 downto 0) <= SWITCH_INPUT; -- Input
            end if;
        end if;
    end process;
end Behavioral;
```

Il componente completo, invece, è realizzato secondo un approccio **Structural**, come segue:

```
entity net_16_4_fpga is
    Port (
        CLOCK      : in std_logic; -- Clock signal
        BTNL        : in std_logic; -- Button input 1
        BTNR        : in std_logic; -- Button input 2
        SWITCH_INPUT : in std_logic_vector(7 downto 0); -- 16 input
        SWITCH_SEL_IN : in std_logic_vector(3 downto 0); -- 4 selezioni input
        SWITCH_SEL_OUT : in std_logic_vector(1 downto 0); -- 2 selezioni output
        LED         : out std_logic_vector(3 downto 0) -- 4 LED
    );
end net_16_4_fpga;

architecture Structural of net_16_4_fpga is
    signal switch_to_net : std_logic_vector(15 downto 0);

    component switch_8_capture
        Port (
            CLOCK      : in std_logic; -- Clock signal
            BTN         : in std_logic; -- Button input
            SWITCH_INPUT : in std_logic_vector(7 downto 0); -- 8 input
            OUTPUT      : out std_logic_vector(7 downto 0) -- Input
        );
    end component;

    component net_16_4 is
        Port (
            input      : in std_logic_vector(15 downto 0);
            input_control : in std_logic_vector(3 downto 0);
            output_control : in std_logic_vector(1 downto 0);
            output      : out std_logic_vector(3 downto 0)
        );
    end component;

begin
    switch_0 : switch_8_capture
        Port map(
            CLOCK => CLOCK,
            BTN => BTNL, -- Button input 1
            SWITCH_INPUT => SWITCH_INPUT, -- 8 input
            OUTPUT => switch_to_net(7 downto 0) -- Input
        );

    switch_1 : switch_8_capture
        Port map(
            CLOCK => CLOCK,
            BTN => BTNR, -- Button input 2
            SWITCH_INPUT => SWITCH_INPUT, -- 8 input
            OUTPUT => switch_to_net(15 downto 8) -- Input
        );
```

```

net : net_16_4
  Port map(
    input => switch_to_net,
    input_control => SWITCH_SEL_IN,
    output_control => SWITCH_SEL_OUT,
    output => LED
  );
end Structural;

```

Come si può osservare, per realizzare gli ingressi di selezione sono stati utilizzati gli altri switch disponibili sulla board.

I *constraints* del progetto sono i seguenti:

```

## 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 { SWITCH_INPUT[0] }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_INPUT[1] }];
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_INPUT[2] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_INPUT[3] }];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_INPUT[4] }];
#IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_INPUT[5] }];
#IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_INPUT[6] }];
#IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_INPUT[7] }];
#IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8       IOSTANDARD LVCMOS18 } [get_ports { SWITCH_SEL_IN[0] }];
#IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8       IOSTANDARD LVCMOS18 } [get_ports { SWITCH_SEL_IN[1] }];
#IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_SEL_IN[2] }];
#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_SEL_IN[3] }];
#IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6       IOSTANDARD LVCMOS33 } [get_ports { SWITCH_SEL_OUT[0] }];
#IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12      IOSTANDARD LVCMOS33 } [get_ports { SWITCH_SEL_OUT[1] }];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]

## LEDs
set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 } [get_ports { LED[0] }]; #IO_L18P_T2_A24_15
Sch=led[0]
set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMOS33 } [get_ports { LED[1] }]; #IO_L24P_T3_RS1_15
Sch=led[1]
set_property -dict { PACKAGE_PIN J13      IOSTANDARD LVCMOS33 } [get_ports { LED[2] }]; #IO_L17N_T2_A25_15
Sch=led[2]
set_property -dict { PACKAGE_PIN N14      IOSTANDARD LVCMOS33 } [get_ports { LED[3] }]; #IO_L8P_T1_D11_14
Sch=led[3]

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

```

## Esercizio 2: Sistema 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*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

#### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

#### *Simulazione*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### 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.

#### *Sintesi su board di sviluppo*

<se richiesto: descrizione dell'architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell'I/O); file di constraint utilizzato per il progetto>

## 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

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

#### Implementazione

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

#### Simulazione

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

#### 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.

#### Sintesi su board di sviluppo

<se richiesto: descrizione dell'architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell'I/O); file di constraint utilizzato per il progetto>

### Esercizio 4: Shift register

#### Esercizio 4.1

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*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

#### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

#### *Simulazione*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### 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*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

#### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

#### *Simulazione*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

#### 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).

#### *Sintesi su board di sviluppo*

<se richiesto: descrizione dell'architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell'I/O); file di constraint utilizzato per il progetto>



### Esercizio 5.3 (solo 9 CFU)

Estendere il componente sviluppato ai punti precedenti in modo che sia in grado di acquisire e memorizzare internamente fino ad un numero N di intertempi in corrispondenza di un ingresso di stop. Opzionalmente, il componente può prevedere una modalità di visualizzazione in cui, alla pressione di un bottone, vengano visualizzati sui display gli intertempi memorizzati (uno per ogni pressione).

#### *Sintesi su board di sviluppo*

<se richiesto: descrizione dell'architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell'I/O); file di constraint utilizzato per il progetto>

## 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 tempificazione 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*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

#### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

#### *Simulazione*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### Esercizio 6.2

Sintetizzare ed implementare su board il componente sviluppato al punto precedente, utilizzando due bottoni per i segnali di read e reset rispettivamente e i led per la visualizzazione delle uscite della macchina istante per istante.

#### *Sintesi su board di sviluppo*

<se richiesto: descrizione dell'architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell'I/O); file di constraint utilizzato per il progetto>

## Capitolo 3: Macchine aritmetiche

### Esercizio 7: Moltiplicatore di Booth

#### 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*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

##### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

##### *Simulazione*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

#### 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.

##### *Sintesi su board di sviluppo*

<se richiesto: descrizione dell'architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell'I/O); file di constraint utilizzato per il progetto>

### Esercizio 7BIS: Divisore Non-Restoring (solo 9 CFU)

#### Esercizio 7BIS.1

Progettare, implementare in VHDL e simulare una macchina divisore (modalità non-restoring) in grado di effettuare la divisione intera fra due stringhe A e B di 4 bit ciascuna.

##### *Progetto e architettura*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

##### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

##### *Simulazione*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### Esercizio 7BIS.2

Sintetizzare il divisore implementato al punto 7BIS.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.

#### *Sintesi su board di sviluppo*

<se richiesto: descrizione dell'architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell'I/O); file di constraint utilizzato per il progetto>

## Capitolo 4: Comunicazione con handshaking

### Esercizio 8: 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 ( $i=0, \dots, N-1$ ). 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*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

#### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

#### *Simulazione*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

## Capitolo 5: Processore

### Esercizio 9: Processore IJVM

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

- a) si proceda all'analisi dell'architettura mediante simulazione e si approfondisca lo studio del suo funzionamento per due istruzioni a scelta,
- b) si modifichi un codice operativo a scelta, documentando tutte le modifiche effettuate.

#### *Progetto e architettura*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

#### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

#### *Simulazione*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

## Capitolo 6: Interfaccia seriale

### Esercizio 10: Interfaccia UART

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*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

#### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

#### *Simulazione*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### Esercizio 10BIS: Interfaccia UART (solo 9 CFU)

Dopo aver simulato il comportamento del sistema, lo si implementi su board usando un bottone per il segnale di WR in A, un bottone per il segnale di RD in B, e i display per la visualizzazione del dato correntemente trasmesso e memorizzato in MEM. Si testi l'errore di overrun.

#### *Sintesi su board di sviluppo*

<se richiesto: descrizione dell'architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell'I/O); file di constraint utilizzato per il progetto>

## Capitolo 7: Switch multistadio

### Esercizio 11: Switch multistadio

#### Esercizio 11.1

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*

<descrizione dell'approccio di progetto utilizzato, disegno architetture del sistema e dei suoi componenti, descrizione delle funzionalità>

##### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

##### *Simulazione (?)*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

#### Esercizio 11.2 (solo 9 CFU)

Rimuovendo l'ipotesi di lavorare secondo uno schema a priorità fissa fra i nodi e considerando una rete di 8 nodi, lo switch deve gestire eventuali conflitti generati da collisioni con un meccanismo a scelta dello studente.

##### *Progetto e architettura*

<descrizione dell'approccio di progetto utilizzato, disegno architetture del sistema e dei suoi componenti, descrizione delle funzionalità>

##### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

##### *Simulazione (?)*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

#### Esercizio 11.3 (solo 9 CFU)

Si implementi un protocollo di handshaking semplice regolato da una coppia di segnali (pronto a inviare/pronto a ricevere) per l'invio di ciascun messaggio fra due nodi.

##### *Progetto e architettura*

<descrizione dell'approccio di progetto utilizzato, disegno architetture del sistema e dei suoi componenti, descrizione delle funzionalità>

### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

### *Simulazione (?)*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>



## Capitolo 8: Esercizio prova di esame dicembre 2024

### Esercizio 12: Prova di esame del 19 dicembre 2024

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. È 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 A più veloce).

#### *Progetto e architettura*

<descrizione dell'approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

#### *Implementazione*

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>

#### *Simulazione (?)*

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

## Appendice

### Multiplexer 4:1

#### Progetto e architettura

Il **Multiplexer 4:1** è un componente puramente combinatorio che convoglia, attraverso opportune linee di selezione, una tra le quattro linee d'ingresso verso un'unica uscita.

In particolare, il multiplexer realizzato è un **multiplexer indirizzabile**, ovvero un componente nel quale le linee di selezione sono opportunamente decodificate da un decoder. Le linee di selezione realizzate in tale architettura, dunque, non sono pari al numero delle linee di ingresso (come avviene nella versione lineare del multiplexer), bensì  $\log_2 N$ , ovvero il numero di bit necessari a codificare i numeri che vanno da 0 a  $N$ .

Per selezionare un'opportuna linea, dunque, è stata realizzata un'architettura di tipo **Dataflow** basata sugli ingressi e sulle selezioni.

#### Implementazione

Nel **Port** sono stati definiti quattro ingressi  $a_i$ , due selezioni  $s_j$  e un'uscita  $y$ . L'architecture definisce il flusso dati secondo la funzione logica del multiplexer indirizzabile.

```
entity mux_4_1 is
  Port (
    a0 : in STD_LOGIC;
    a1 : in STD_LOGIC;
    a2 : in STD_LOGIC;
    a3 : in STD_LOGIC;
    s0 : in STD_LOGIC;
    s1 : in STD_LOGIC;
    y  : out STD_LOGIC
  );
end mux_4_1;

architecture Dataflow of mux_4_1 is
  begin
    y <= ((a0 AND NOT(s1) AND NOT(s0)) OR (a1 AND NOT(s1) AND s0) OR (a2 AND s1 AND NOT(s0)) OR
(a3 AND s1 AND s0));
  end Dataflow;
```

### Button Debouncer

#### Progetto e architettura

Questo componente serve per evitare che il segnale logico del bottone venga rilevato in maniera non corretta a causa di possibili oscillazioni e rumori. L'obiettivo è assicurarsi che il clic abbia una durata simile a quella di un plausibile clic umano; i rumori hanno durate nettamente inferiori a un vero clic. Se non inserissimo tale componente, si rischierebbe di valutare un'oscillazione temporanea e indesiderata come valore desiderato per il funzionamento di un componente che presenti un bottone. Inoltre, anche se tenessimo

Il Button Debouncer prende in input il segnale proveniente dal bottone, e genera un segnale "ripulito" della durata di un colpo di clock per segnalare l'avvenuta pressione del bottone.

Per farlo, si implementa un automa a 4 stati:

- **NOT\_PRESSED**: il bottone non è stato cliccato (stato iniziale);
- **CHK\_PRESSED**: possibile clic del bottone (BTN='1' rilevato);
- **PRESSED**: clic riconosciuto (se BTN è ancora '1' dopo un certo tempo);
- **CHK\_NOT\_PRESSED**: stato intermedio in cui si attende un certo tempo per superare eventuali oscillazioni. Se dopo tale tempo il segnale logico del bottone è ancora alto, si va in PRESSED; altrimenti, si torna in NOT\_PRESSED.

In tal modo, se si mantiene il bottone premuto, non vengono generati più impulsi in uscita.

### Implementazione

```
entity 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
  );
  Port ( RST : in STD_LOGIC;
        CLK : in STD_LOGIC;
        BTN : in STD_LOGIC;
        CLEARED_BTN : out STD_LOGIC);
end ButtonDebouncer;

architecture Behavioral of ButtonDebouncer is

  type stato is (NOT_PRESSED, CHK_PRESSED, PRESSED, CHK_NOT_PRESSED);
  signal BTN_state : stato := NOT_PRESSED;

  constant max_count : integer := btn_noise_time/CLK_period; -- 10000000/10= conto 1000000 colpi di clock

begin

  deb: process (CLK)
    variable count: integer := 0;

  begin
    if rising_edge(CLK) then

      if( RST = '1') then
        BTN_state <= NOT_PRESSED;
        CLEARED_BTN <= '0';
      else
        case BTN_state is
          when NOT_PRESSED =>
            if( BTN = '1' ) then
              BTN_state <= CHK_PRESSED;
            else
              BTN_state <= NOT_PRESSED;
            end if;
          when CHK_PRESSED =>
            if(count = max_count -1) then
              if(BTN = '1') then --se arrivo a count max ed è ancora alto vuol dire che non era
un bounce, devo alzare CLEARED_BTN
                count:=0;
                CLEARED_BTN <= '1';
                BTN_state <= PRESSED;
              else
                count:=0;
                BTN_state <= NOT_PRESSED;
              end if;
            else
              count:= count+1;
              BTN_state <= CHK_PRESSED;
            end if;
          end if;
        end if;
      end if;
    end if;
  end if;
end if;
```

```

        when PRESSED =>
            CLEARED_BTN<= '0'; -- questo lo metto per fare in modo che il segnale sia alto per un
solo impulso di clock
            if(BTN = '0') then
                BTN_state <= CHK_NOT_PRESSED;
            else
                BTN_state <= PRESSED;
            end if;

            when CHK_NOT_PRESSED =>
                if(count = max_count -1) then
                    if(BTN = '0') then -- se arrivo a count max ed è ancora basso vuol dire che non era
un bounce e il bottone è stato rilasciato
                        count:=0;
                        BTN_state <= NOT_PRESSED;
                    else
                        count:=0;
                        BTN_state <= PRESSED;
                    end if;
                else
                    count:= count+1;
                    BTN_state <= CHK_NOT_PRESSED;
                end if;
            when others =>
                BTN_state <= NOT_PRESSED;
            end case;
        end if;
    end if;
end process;

end Behavioral;

```

## Divisore di frequenza

### Progetto e architettura

<descrizione dell'approccio di progetto utilizzato, disegno architettuale del sistema e dei suoi componenti, descrizione delle funzionalità>

### Implementazione

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un'appendice>