2022

Università degli Studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base

Esame ASDI

SIMONE D'ORTA

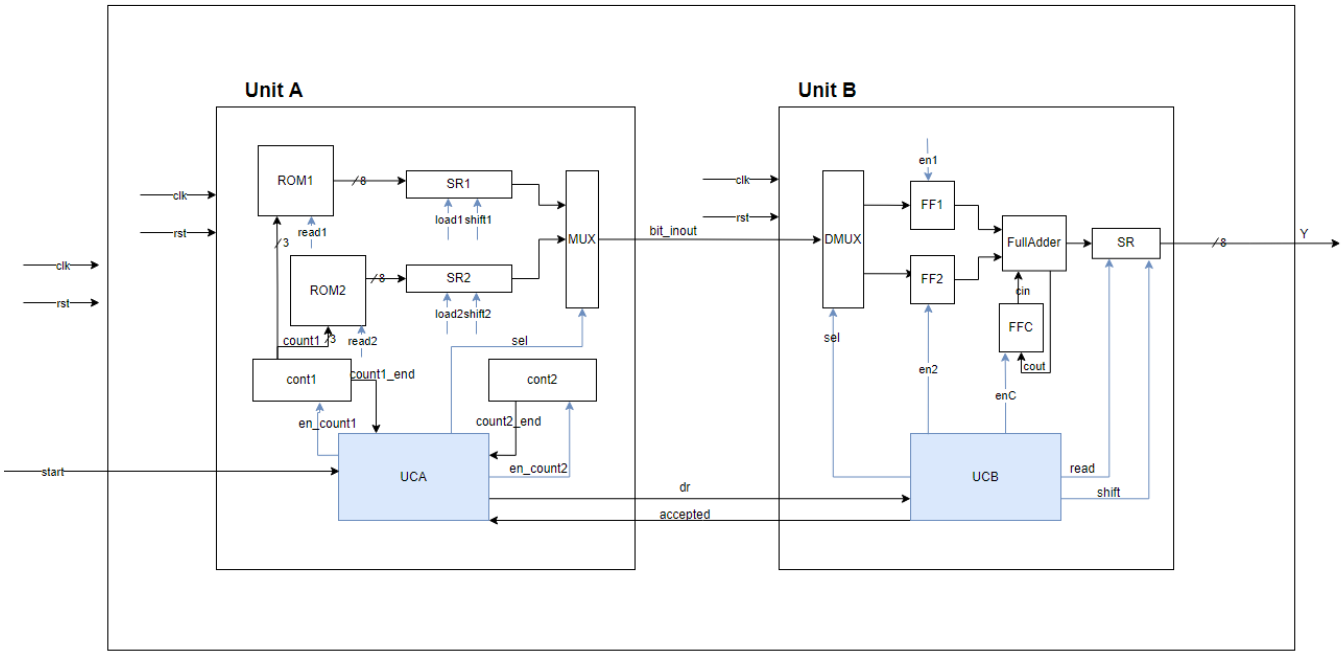
Sommario

1 Traccia

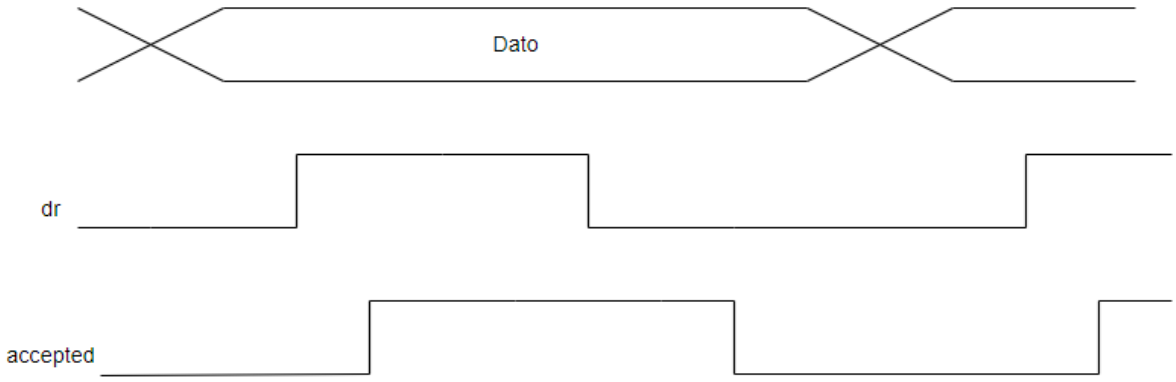
Un'unità A invia a un'unità B N coppie di byte da addizionare, prelevando il primo byte Xi di ciascuna coppia da una memoria ROM1 e l'altro Yi da una seconda memoria ROM2. I byte vengono trasmessi a B mediante un protocollo di handshaking effettuato bit a bit, inviando prima un bit di Xi e poi uno di Yi. Per ogni coppia di bit ricevuti, l'unità B li passa ad un sommatore seriale.
Si progetti ed implementi in VHDL il sistema complessivo, avendo cura di includere due componenti contatore in A (uno che conta le coppie inviate e l'altro utilizzato per la trasmissione bit a bit) e un componente contatore e uno sommatore in B.

2 Soluzione

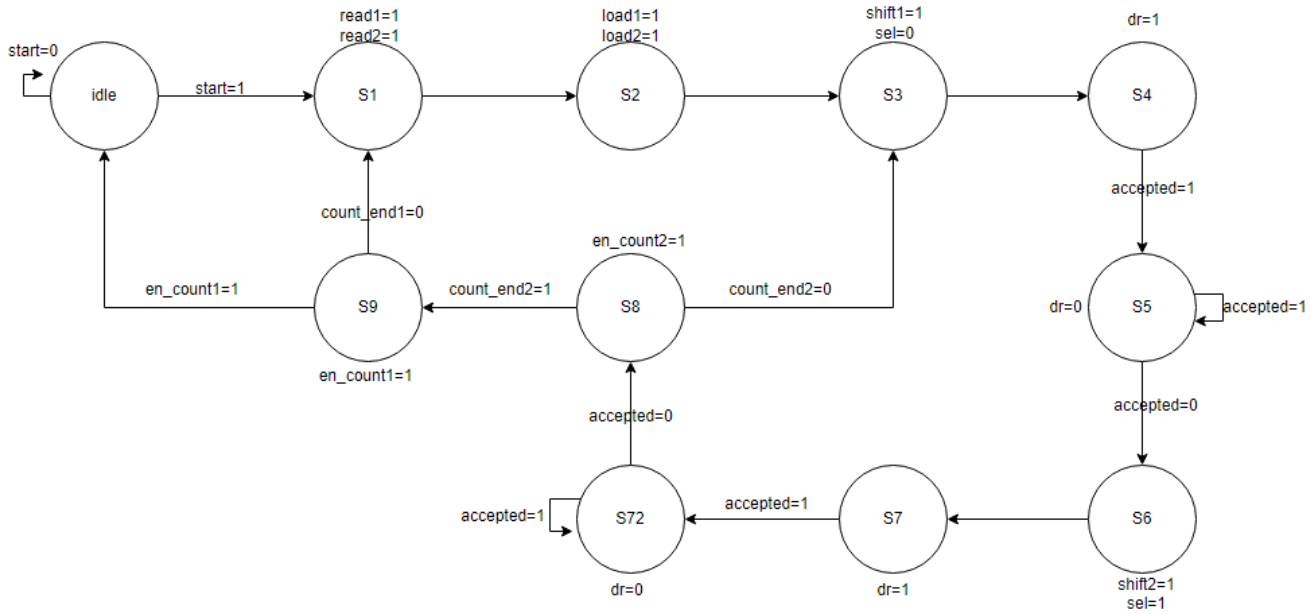
Ogni singola unità è stata decomposta in unità operativa e unità di controllo.
Le due unità sono state poi collegate all'interno di un top module.



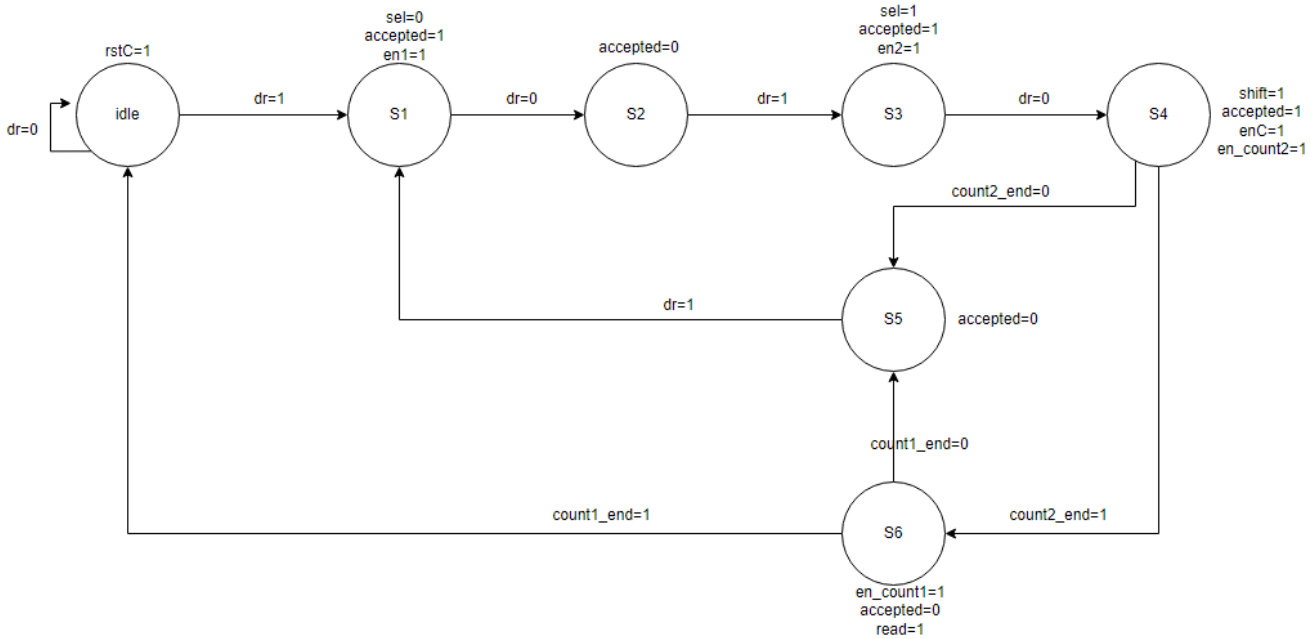
Le due unità di controllo, oltre a gestire le unità operative, implementano un protocollo di handshake tramite i segnali dr (data ready) e accepted.



AUTOMA CONTROLLER A



AUTOMA CONTROLLER B



3 Codice

3.1 Unità A

```
entity unitA is
    Port (
        clk, rst, start: in std_logic;
        accepted: in std_logic;
        dr, bit_out: out std_logic
     );
end unitA;

architecture structural of unitA is

component Cont8 is
    Port (
        clock, reset: in std_logic;
        count_in: in std_logic;
        count_end: out std_logic;
        count: out std_logic_vector(2 downto 0)
    );
end component;

component ROM is
port(
    CLK : in std_logic; -- clock della board
    RST : in std_logic;
    READ : in std_logic; -- segnale che abilita la lettura
    ADDR : in std_logic_vector(2 downto 0); --3 bit di ind
    DATA : out std_logic_vector(7 downto 0) -- dato su 8 b
    );
end component;

component ROM2 is
port(
    CLK : in std_logic; -- clock della board
    RST : in std_logic;
    READ : in std_logic; -- segnale che abilita la lettura
    ADDR : in std_logic_vector(2 downto 0); --3 bit di ind
    DATA : out std_logic_vector(7 downto 0) -- dato su 8 b
    );
end component;
```

```
component parallel_in_serial_out is
     port(
         clk : in STD_LOGIC;
         reset : in STD_LOGIC;
         load, shift : in STD_LOGIC;
         din : in STD_LOGIC_VECTOR(7 downto 0);
         dout : out STD_LOGIC
         );
end component;

component mux2_1 is
    Port ( x0 : in STD_LOGIC;
           x1 : in STD_LOGIC;
           sel : in STD_LOGIC;
           y0 : out STD_LOGIC);
end component;

component control_unitA is
    Port (
        clk, rst, start: in std_logic;
        count_end1, count_end2: in std_logic;
        accepted: in std_logic;
        read1, read2, load1, load2, shift1, shift2: out std_logic;
        en_count1, en_count2: out std_logic;
        sel, dr: out std_logic
     );
end component;
signal sr1_out, sr2_out, sel: std_logic;
signal rom1_out, rom2_out: std_logic_vector(7 downto 0);
signal count1, count2: std_logic_vector(2 downto 0);
signal en_count1, en_count2, read1, read2: std_logic;
signal load1, load2, shift1, shift2: std_logic;
signal count_end1, count_end2: std_logic;

begin
mux: mux2_1 port map(sr1_out, sr2_out, sel, bit_out);
r1: ROM port map(clk, rst, read1, count1, rom1_out);
r2: ROM2 port map(clk, rst, read2, count1, rom2_out);
sr1: parallel_in_serial_out port map(clk, rst, load1, shift1, rom1_out, sr1_out);
sr2: parallel_in_serial_out port map(clk, rst, load2, shift2, rom2_out, sr2_out);
c1: cont8 port map(clk, rst, en_count1, count_end1, count1);
c2: cont8 port map(clk, rst, en_count2, count_end2, count2);
cu: control_unitA port map(clk, rst, start, count_end1, count_end2, accepted, read1, read2, load1, load2,
                           shift1, shift2, en_count1, en_count2, sel, dr);
```

3.1.1 Unità operativa

Mux:

```
entity mux2_1 is
    Port ( x0 : in STD_LOGIC;
           x1 : in STD_LOGIC;
           sel : in STD_LOGIC;
           y0 : out STD_LOGIC);
end mux2_1;

architecture rtl of mux2_1 is

begin
    y0 <= x0 when sel = '0' else
        x1 when sel = '1' else
        '-';

end rtl;
```

ROM:

```
entity ROM is
port(
    CLK : in std_logic; -- clock della board
    RST : in std_logic;
    READ : in std_logic; -- segnale che abilita la lettura, inserito
    ADDR : in std_logic_vector(2 downto 0); --3 bit di indirizzo per
    DATA : out std_logic_vector(7 downto 0) -- dato su 8 bit letto da
    );
end ROM;
-- creo una ROM di 8 elementi da 8 bit ciascuno
architecture behavioral of ROM is
type rom_type is array (7 downto 0) of std_logic_vector(7 downto 0);
signal ROM : rom_type := (
X"4A",
X"49",
X"01",
X"DD",
X"34",
X"02",
X"1E",
X"26");

attribute rom_style : string;
attribute rom_style of ROM : signal is "block";-- block dice al tool
                                               -- distributed di usar
begin

process(CLK)
  begin
    if rising_edge(CLK) then
        if (RST = '1') then
            DATA <= ROM(conv_integer("000"));
        elsif (READ = '1') then
            DATA <= ROM(conv_integer(ADDR));
        end if;
    end if;
end process;
end behavioral;
```

Shift register PISO:

```
entity parallel_in_serial_out is
     port(
         clk : in STD_LOGIC;
         reset : in STD_LOGIC;
         load, shift : in STD_LOGIC;
         din : in STD_LOGIC_VECTOR(7 downto 0);
         dout : out STD_LOGIC
         );
end parallel_in_serial_out;

architecture piso_arc of parallel_in_serial_out is

signal temp: std_logic_vector(7 downto 0);

begin

    piso : process (clk,reset) is
    begin
        if (reset='1') then
            temp <= (others=>'0');
        elsif (clk'event and clk='1') then
            if (load='1') then
                temp <= din ;
            elsif (shift='1') then
                dout <= temp(0);
                temp <= '0'&temp(7 downto 1);
        end if;
        end if;
    end process piso;

end piso_arc;
```

Contatore modulo 8:

```
entity Cont8 is
    Port (
        clock, reset: in std_logic;
        count_in: in std_logic;
        count_end: out std_logic;
        count: out std_logic_vector(2 downto 0)
    );
end Cont8;

architecture Behavioral of Cont8 is

signal c: std_logic_vector(2 downto 0):="000";

begin

CM8: process(clock)
begin
    if(clock'event and clock='1') then
        if(reset='1') then
            c<=(others=>'0');
            count_end<='0';
        elsif(count_in='1') then
            c<=std_logic_vector(unsigned(c)+1);
            if(c="111") then    count_end<='1';
            else    count_end<='0';
            end if;
        end if;
    end if;
end process;
count <= c;
end Behavioral;
```

3.1.2 Unità di controllo

```
entity control_unitA is
    Port (
        clk, rst, start: in std_logic;
        count_end1, count_end2: in std_logic;
        accepted: in std_logic;
        read1, read2, load1, load2, shift1, shift2: out std_logic;
        en_count1, en_count2: out std_logic;
        sel, dr: out std_logic
     );
end control_unitA;

architecture automa of control_unitA is

type state is(idle, S1, S2, S3, S4, S5, S6, S7, S72, S8, S9);
signal current_state: state := idle;
signal next_state: state;

begin

reg:process(clk)
begin
if (clk'event and clk='0') then
    if(rst='1') then
        current_state<=idle;
    else
    current_state<=next_state;
    end if;
end if;
end process;

comb: process(start, accepted, count_end1, count_end2, current_state)
begin
    en_count1<='0';
    en_count2<='0';
    read1<='0';
    read2<='0';
    load1<='0';
    load2<='0';
    shift1<='0';
    shift2<='0';
    dr<='0';
    case current_state is
        when idle =>
                    if(start='1') then
```

```
                    next_state<=S1;
                else   next_state<=idle;
                end if;
    when S1 =>  read1<='1';
                read2<='1';
                next_state<=S2;
    when S2 =>  load1<='1';
                load2<='1';
                next_state<=S3;
    when S3 =>  shift1<='1';
                sel<='0';
                next_state<=S4;
    when S4 =>  dr<='1';
                if(accepted='1') then
                    next_state<=S5;
                end if;
    when S5 =>  dr<='0';
                if(accepted='0') then
                    next_state<=S6;
                end if;
    when S6 =>  shift2<='1';
                sel<='1';
                next_state<=S7;
    when S7 =>  dr<='1';
                if(accepted='1') then
                    next_state<=S72;
                end if;
    when S72 => dr<='0';
                if(accepted='0') then
                    next_state<=S8;
                end if;
    when S8 =>  en_count2<='1';
                if(count_end2='1') then
                    next_state<=S9;
                elsif (count_end2='0') then
                    next_state<=S3;
                end if;
    when S9 =>  en_count1<='1';
                if(count_end1='1') then
                    next_state<=idle;
                elsif (count_end1='0') then
                    next_state<=S1;
                end if;
end case;
```

3.2 Unità B

```
entity unitB is
    Port (
        clk, rst: in std_logic;
        dr, bit_in: in std_logic;
        accepted: out std_logic;
        Y: out std_logic_vector(7 downto 0)
     );
end unitB;

architecture structural of unitB is

component serial_in_parallel_out is
     port(
         clk : in STD_LOGIC;
         reset : in STD_LOGIC;
         read, shift : in STD_LOGIC;
         din : in STD_LOGIC;
         dout : out STD_LOGIC_VECTOR(7 downto 0)
         );
end component;

component full_adder is
    port(
    a,b: in std_logic;
    cin: in std_logic;
    cout, s: out std_logic);
end component;

component demux_12 is
port ( X: in std_logic;
       s: in std_logic; --ingresso di selezione
       Y0, Y1: out std_logic
       );
end component;

component FFD is
   port(
      Q : out std_logic;
      clk, rst :in std_logic;
      D, en :in  std_logic
   );
end component;
```

```
component Cont8 is
    Port (
        clock, reset: in std_logic;
        count_in: in std_logic;
        count_end: out std_logic;
        count: out std_logic_vector(2 downto 0)
    );
end component;

component control_unitB is
    Port (
        clk, rst: in std_logic;
        dr, count1_end, count2_end: in std_logic;
        rstC, accepted, en1, en2, enC: out std_logic;
        shift, read, en_count1, en_count2, sel: out std_logic
     );
end component;

signal sel, en1, en2, enC, a, b, rstC: std_logic;
signal y1, y2, cin, cout, s, count1_end, count2_end: std_logic;
signal count1, count2: std_logic_vector(2 downto 0);
signal en_count1, en_count2, shift, read: std_logic;

begin
ff1: FFD port map(a, clk, rst, y1, en1);
ff2: FFD port map(b, clk, rst, y2, en2);
ffC: FFD port map(cin, clk, rstC, cout, enC);
demux: demux_12 port map(bit_in, sel, y1, y2);
fa: full_adder port map(a, b, cin, cout, s);
c1: cont8 port map(clk, rst, en_count1, count1_end, count1);
c2: cont8 port map(clk, rst, en_count2, count2_end, count2);
sr: serial_in_parallel_out port map(clk, rst, read, shift, s, Y);
cu: control_unitB port map(clk, rst, dr, count1_end, count2_end, rstC, accepted, en1,
                           en2, enC, shift, read, en_count1, en_count2, sel);

end structural;
```

3.2.1 Unità operativa

Flip Flop D:

```
entity FFD is
   port(
      Q : out std_logic;
      clk, rst :in std_logic;
      D, en :in  std_logic
   );
end FFD;
architecture Behavioral of FFD is
begin
 process(clk)
 begin
    if(rising_edge(clk)) then
        if(rst='1') then
            Q <= '0';
        elsif(en='1') then
            Q <= D;
        end if;
    end if;
 end process;
end Behavioral;
```

Demultiplexer:

```
entity demux_12 is
port ( X: in std_logic;
       s: in std_logic; --ingresso di selezione
       Y0, Y1: out std_logic
       );
end demux_12;

architecture Behavioral of demux_12 is

begin

process(X,s)
begin
if (s = '0')then
   Y0 <= X;
   Y1 <= '0';
elsif (s = '1') then
   Y0 <= '0';
   Y1 <= X;
end if;
end process;
end Behavioral;
```

Full adder:

```
entity full_adder is
    port(
    a,b: in std_logic;
    cin: in std_logic;
    cout, s: out std_logic);
end full_adder;


architecture rtl of full_adder is

    begin

    s<= a xor b xor cin;
    cout<= (a and b) or (cin and (a xor b));

    end rtl;
```

Shift register SIPO:

```
entity serial_in_parallel_out is
     port(
          clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          read, shift : in STD_LOGIC;
          din : in STD_LOGIC;
          dout : out STD_LOGIC_VECTOR(7 downto 0)
          );
end serial_in_parallel_out;

architecture behavioral of serial_in_parallel_out is

signal temp: std_logic_vector(7 downto 0):=X"00";

begin

    process (clk,reset) is
    begin
        if (reset='1') then
            temp <= (others=>'0');
            dout <= (others=>'0');
        elsif (clk'event and clk='1') then
            if (read='1') then
                dout <= temp ;
            elsif (shift='1') then
                temp <= din&temp(7 downto 1);
        end if;
        end if;
    end process piso;

end behavioral;
```

3.2.2 Unità di controllo

```
entity control_unitB is
    Port (
        clk, rst: in std_logic;
        dr, count1_end, count2_end: in std_logic;
        rstC, accepted, en1, en2, enC: out std_logic;
        shift, read, en_count1, en_count2, sel: out std_logic
     );
end control_unitB;

architecture automa of control_unitB is

type state is(idle, S1, S2, S3, S4, S5, S6);
signal current_state: state := idle;
signal next_state: state;

begin

reg:process(clk)
begin
if (clk'event and clk='0') then
    if(rst='1') then
        current_state<=idle;
    else
    current_state<=next_state;
    end if;
end if;
end process;

comb: process(dr, count1_end, count2_end, current_state)
begin
    en_count1<='0';
    en_count2<='0';
    shift<='0';
    accepted<='0';
    en1<='0';
    en2<='0';
    enC<='0';
    rstC<='0';
    sel<='0';
    read<='0';
    case current_state is
        when idle => rstC<='1';
                     if(dr='1') then
                        next_state<=S1;
```

```
                    else
                       next_state<=idle;
                    end if;
        when S1 =>  en1<='1';
                    accepted<='1';
                    sel<='0';
                    if(dr='0') then
                       next_state<=S2;
                    end if;
        when S2 =>  accepted<='0';
                    if(dr='1') then
                       next_state<=S3;
                    end if;
        when S3 =>  sel<='1';
                    accepted<='1';
                    en2<='1';
                    if(dr='0') then
                       next_state<=S4;
                    end if;
        when S4 =>  shift<='1';
                    enC<='1';
                    en_count2<='1';
                    accepted<='1';
                    if(count2_end='1') then
                       next_state<=S6;
                    elsif(count2_end='0') then
                       next_state<=S5;
                    end if;
        when S5 =>  if(dr='1') then
                       next_state<=S1;
                    end if;
        when S6 =>  en_count1<='1';
                    read<='1';
                    if(count1_end='1') then
                       next_state<=idle;
                    elsif(count1_end='0') then
                       next_state<=S5;
                    end if;
    end case;
end process;

end automa;
```

3.3 Top Module

```
entity TopModule is
    Port (
        clk, rst, start: in std_logic;
        Y: out std_logic_vector(7 downto 0)
     );
end TopModule;

architecture structural of TopModule is

component unitA is
    Port (
        clk, rst, start: in std_logic;
        accepted: in std_logic;
        dr, bit_out: out std_logic
     );
end component;

component unitB is
    Port (
        clk, rst: in std_logic;
        dr, bit_in: in std_logic;
        accepted: out std_logic;
        Y: out std_logic_vector(7 downto 0)
     );
end component;

signal accepted, dr, bit_inout: std_logic;

begin
A: unitA port map(clk, rst, start, accepted, dr, bit_inout);
B: unitB port map(clk, rst, dr, bit_inout, accepted, Y);
end structural;
```

4 Simulazione

ROM1 = 26, 1E, 02, 34, DD, 01, 49, 41

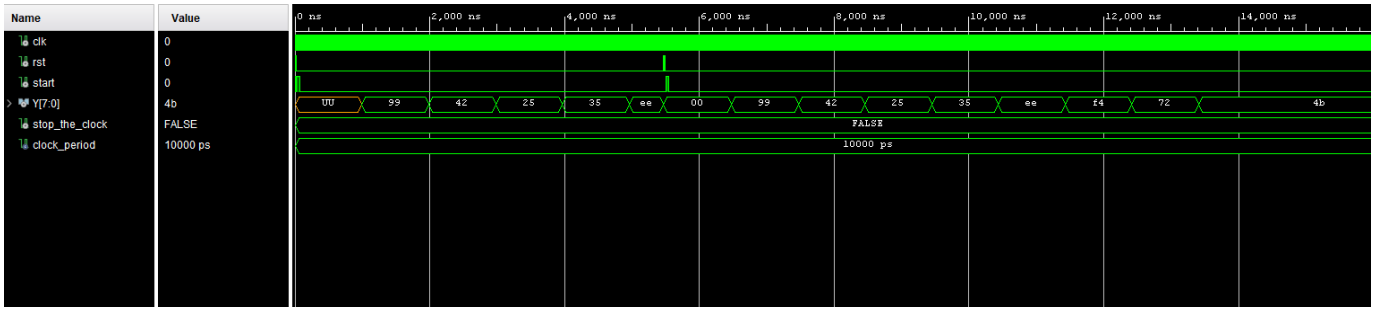
ROM2 = 73, 24, 23, 01, 11, F3, 20, 01

Y = 99, 42, 25, 35, EE, F4, 72, 4B

```
stimulus: process
begin

  wait for 20ns;
  rst<='0';
  start<='1';
  wait for 50ns;
  start<='0';
  wait for 5400ns;
  rst<='1';
  wait for 20ns;
  rst<='0';
  wait for 10ns;
  start<='1';
  wait for 50ns;
  start<='0';

  wait;
end process;
```



Dettaglio della singola iterazione: