Esercizio esame ASDi

Raffaele Russo M63001325

19 Ottobre 2022

Indice

1 Traccia

Progettare, implementare in VHDL e simulare un sistema composto da due nodi, A e B, operanti come segue. Il nodo A acquisisce, mediante un protocollo definito dallo studente, fino a un massimo di 16 stringhe di 4 bit attraverso un'unità di input esterna. Ciascuna stringa viene trasmessa da A a B serialmente, un bit alla volta. All'interno di B, per ogni gruppo di 4 bit ricevuto, un riconoscitore di sequenze (senza sovrapposizione) fornisce uscita alta se la sequenza ricevuta è uguale a 1010. Ogni volta che tale condizione risulta verificata, un contatore modulo 16 si incrementa di un'unità: se il numero di sequenze riconosciute raggiunge il valore 10, il nodo B invia un segnale di stop ad A (codificato a scelta dello studente), che termina immediatamente l'acquisizione

2 Approccio utilizzato

Per la progettazione del sistema complessivo ho individuato le entità del mio problema, scomponendo dove necessario i vari sottosistemi in unità operativa e di controllo.

Le entità coinvolte sono:

- Nodo di input
- Sistema
  - Nodo A
  - Nodo B

L'unità di input esterna considerata è un nodo che dispone di una ROM 16×4, contenente le 16 sequenze di 4 bit da trasmettere al nodo A.

Per la scansione delle stringhe ho utilizzato un contatore modulo 16 mentre per l'acquisizione dell'input da parte di A ho deciso di implementare un protocollo di handshaking non interlacciato. La scelta è stata fatta per ragioni di semplicità. L'introduzione di un segnale e di stati aggiuntivi, per sbloccare eventualmente l'unità di input per fare altre operazioni, non è stata ritenuta conveniente.

Il funzionamento ad alto livello è il seguente: il nodo di input avvia la fase di handshaking con il nodo A, il quale riceverà in parallelo i 4 bit provenienti dalla ROM, per poi memorizzarli in un registro a scorrimento a ingresso parallelo e uscita seriale. Fino a quando il nodo A non trasmetterà al nodo B i 4 bit, il nodo di input resterà in attesa del segnale CTS.

L'unità di controllo di A dovrà tempificare opportunamente l'abilitazione dello shift register, l'invio dei 4 bit al nodo B e un contatore modulo 4. Quest'ultimo ci consente di effettuare esattamente 4 shift e di indicare all'unità di controllo di A la fine della trasmissione della sequenza. Il nodo B possiede un riconoscitore di sequenza, che analizza i bit in maniera sequenziale. A tal proposito oltre al bit della sequenza, A invierà a B anche il segnale di abilitazione,

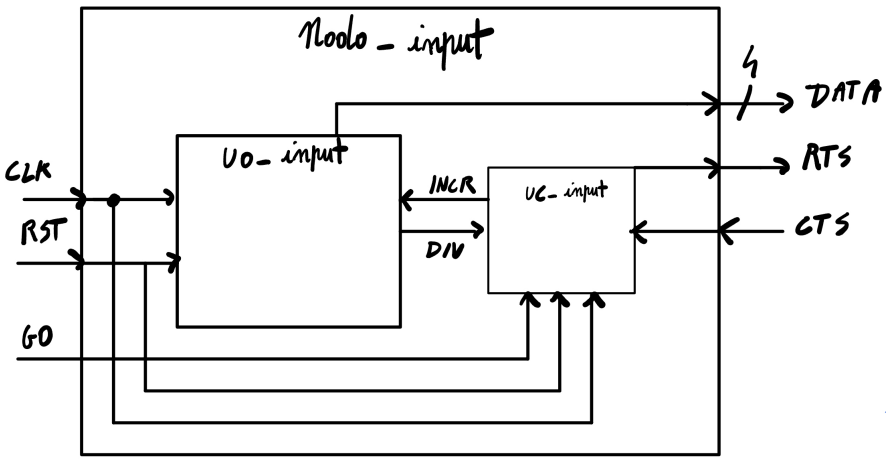

Figura 1: Schema nodo di input

a partire dal quale il riconoscitore inizierà a elaborare l'input a ogni colpo di clock.

L'uscita del riconoscitore andrà a pilotare un contatore modulo 10, il cui segnale DIV si alzerà alla ricezione di 10 stringhe pari a 1010. Tale segnale ha la funzione di stoppare l'acquisizione dei bit dall'unità di input esterna. Questo viene ottenuto grazie all'unità di controllo di A, che sulla base di questo segnale di stop, decide se chiudere l'handshaking con l'unità di controllo del nodo di input o se interrompere l'acquisizione. Si tenga a mente che per gestire la tempificazione del sistema è stato necessario sfruttare entrambi i fronti del clock.

3 Nodo di input

Entity e architecture

Per la realizzazione del nodo di input è stato adottato un approccio strutturale, in particolare si è scelto di suddividere l'entità in unità operativa e di controllo. In figura 1 è riportata l'architettura del nodo di input.

```
entity Nodo_input is
    port(
        CLK : IN STD_LOGIC;
        RST : IN STD_LOGIC;
        rts : out std_logic;
        cts: in std_logic;
        DATA : out std_logic_vector(3 downto 0);
        go : in std_logic);
```

```
end Nodo_input;

architecture structural of Nodo_input is
 component uc_input is
 port(
      CLK : IN STD_LOGIC;
      RST : IN STD_LOGIC;
      en_cnt : out std_logic;
      rts : out std_logic;
      cts: in std_logic;
      div : in std_logic;
      go : in std_logic);
 end component;

 component uo_input is
 port(
      CLK : in std_logic;
      RST : in std_logic;
      incr: in std_logic;
      div : out std_logic;
      DATA : out std_logic_vector(3 downto 0));
 end component;

signal en_cnt_0: std_logic := '0';
signal div_0: std_logic := '0';

begin
 ucA : uc_input
 port map(
  CLK => CLK,
  RST => RST,
  en_cnt => en_cnt_0,
  rts => rts,
  cts => cts,
  div => div_0,
  go => go
  );

 uoA: uo_input
 port map(
  CLK => CLK,
  RST => RST,
  incr => en_cnt_0,
  DATA => DATA,
  div => div_0
 );
```

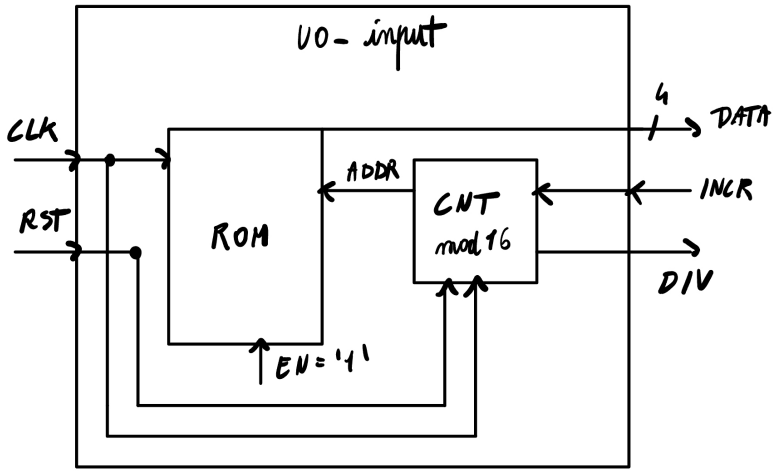

Figura 2: Unità operativa nodo di input

```
end structural;
```

3.1 Unità operativa

Entity e architecture

In figura 2 è riportato lo schema complessivo dell'unità operativa del nodo di input.

```
entity uo_input is
 port(
  CLK : in std_logic;
  RST : in std_logic;
  incr: in std_logic;
  div : out std_logic;
  DATA : out std_logic_vector(3 downto 0)
  );
end uo_input;

architecture structural of uo_input is
 component ROM is
  port(
   CLK : in std_logic;
   RST : in std_logic;
   ADDR : in std_logic_vector(3 downto 0);
   DATA : out std_logic_vector(3 downto 0);
   en : in std_logic
```

```
   );
 end component;

 component cont16 is
  port(
    CLK: in std_logic;
    RST: in std_logic;
    en: in std_logic;
    count: out std_logic_vector(3 downto 0);
    div : out std_logic);
end component;

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

begin
 rom_0 : ROM
 port map(
   CLK => CLK,
   RST => '0',
   ADDR => addr_rom,
   DATA => data,
   en => '1');

 cnt_0 : cont16
  port map(
    CLK => CLK,
    RST => RST,
    en => incr,
    count => addr_rom,
    div => div
    );
end structural;
```

3.1.1 ROM

Entity e architecture

```
entity ROM is
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ADDR : in std_logic_vector(3 downto 0);
        DATA : out std_logic_vector(3 downto 0);
        en : in std_logic);
end ROM;
```

```
architecture behavioral of ROM is
 type rom_type is array (15 downto 0) of std_logic_vector(3 downto 0);
 signal ROM : rom_type := (
        X"5",
        X"f",
        X"f",
        X"f",
        X"f",
        X"5",
        X"f",
        X"5",
        X"5",
        X"f",
        X"f",
        X"f",
        X"f",
        X"5",
        X"5",
        X"f");

 attribute rom_style : string;
 attribute rom_style of ROM : signal is "block";
 begin
 process(CLK)
  begin
  if rising_edge(CLK) then
   if (RST = '1') then
    DATA <= ROM(conv_integer(X"0"));
   elsif (en = '1') then
    DATA <= ROM(conv_integer(ADDR));
   end if;
  end if;
 end process;
end behavioral;
```

3.1.2 Contatore mod16

Entity e architecture

```
entity cont16 is
 port(
    CLK, RST: in std_logic;
    en: in std_logic;
    count: out std_logic_vector(3 downto 0);
    div : out std_logic);
```

```
end cont16;

architecture behavioural of cont16 is
 signal c: std_logic_vector(3 downto 0) := (others => '0');
 begin
 CM16: process(CLK,RST)
 begin
  if(RST = '1') then
   c <= (others=>'0');
   div <= '0';
  elsif(falling_edge(CLK))then
   if (en = '1') then
     c <= std_logic_vector(unsigned(c) + 1);
     if  (c = "1110") then
      div <='1';
     else
      div <= '0';
     end if;
   end if;
  end if;
  end process;
 count<= c;
end behavioural;
```

3.2 Unità di controllo

Entity e architecture

In figura 3 è riportato l'automa dell'unità di controllo del nodo di input.

```
    entity uc_A is
        port(
          CLK : IN STD_LOGIC;
          RST : IN STD_LOGIC;
          en_cnt : out std_logic;
          rts : out std_logic;
          cts: in std_logic;
          div : in std_logic;
          stop : in std_logic;
          load_sr : out std_logic;
          start : out std_logic;
          shift : out std_logic;
          go : in std_logic);
end uc_A;

architecture structural of uc_A is
 type state is (idle,wait_cts, upload,wait_div,q_done);
```

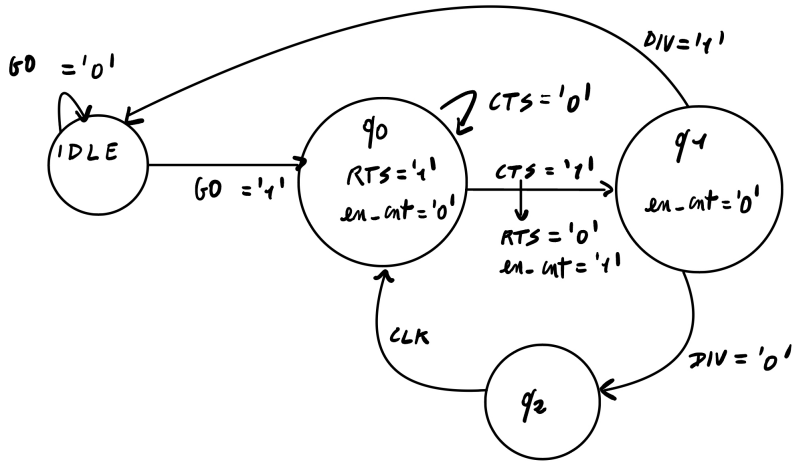

Figura 3: Automa unità di controllo

```
signal current_state : state := idle;
begin
reg_stato: process(CLK)
  begin
   if(rising_edge(CLK)) then
       if rst = '1' then
  current_state<=idle;
   else
   CASE current_state is
   WHEN idle =>
       if(go= '1') then
      current_state<=wait_cts;
     end if;
   WHEN wait_cts =>
     RTS <= '0';

     if(CTS = '1') then
      current_state <= upload;
     end if;

  WHEN upload =>
    current_state <= wait_div;
    load_sr <= '1';
    start <= '1';
    en_cnt <= '1';
    shift <= '1';
    if (stop = '1') then
```

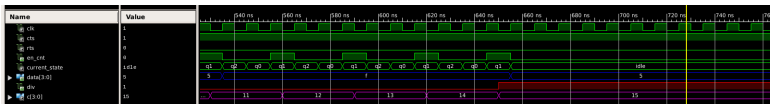

Figura 4: Simulazione nodo di input

```
      current_state <= idle;
      shift <= '0';
      en_cnt <= '0';
      start <= '0';
      load_sr <= '0';
    end if;

 WHEN wait_div =>
   load_sr <= '0';
   if ( DIV = '1') then
    current_state <= q_done;
   end if;

 when q_done =>
  shift <= '0';
  en_cnt <= '0';
  start <= '0';
  RTS <= '1';
  current_state <= wait_cts;

   end CASE;
  end if;
end if;
   end process;
   end structural;
```

3.3 Simulazione

In figura 4 è riportata la simulazione del nodo di input.

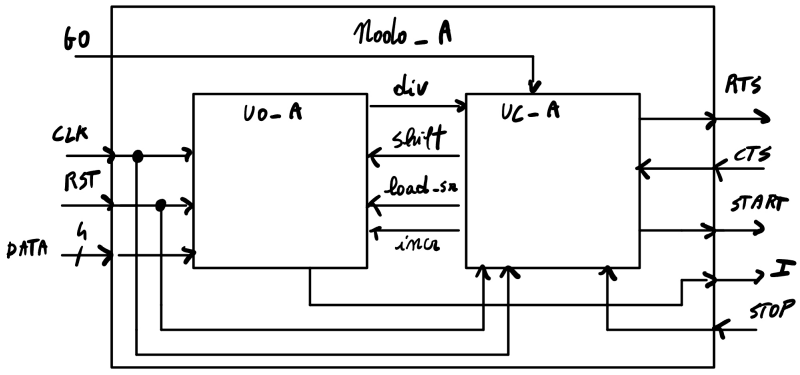

Figura 5: Schema nodo A

4 Nodo A

Entity e architecture

In figura 5 è riportata l'architettura del nodo A.

```
entity Nodo_A is
port(
  CLK : IN STD_LOGIC;
  RST : IN STD_LOGIC;
  rts : out std_logic;
  cts: in std_logic;
  DATA : in std_logic_vector(3 downto 0);
  I : out std_logic;
  start : out std_logic;
  stop : in std_logic;
  go : in std_logic
 );
end Nodo_A;

architecture structural of Nodo_A is
 component uc_A is
 port(
  CLK : IN STD_LOGIC;
  RST : IN STD_LOGIC;
  en_cnt : out std_logic;
  rts : out std_logic;
  cts: in std_logic;
  div : in std_logic;
  stop : in std_logic;
  load_sr : out std_logic;
```

```
 start : out std_logic;
 shift : out std_logic;
 go : in std_logic
 );
end component;

component uo_A is
port(
 CLK : in std_logic;
 RST : in std_logic;
 DATA :in std_logic_vector(3 downto 0);
 I : out std_logic;
 shift : in std_logic;
 load_sr : in std_logic;
 en_cnt : in std_logic;
 div : out std_logic
);
end component;

signal en_cnt_0: std_logic := '0';
signal div_0: std_logic := '0';
signal load_sr_0: std_logic := '0';
signal shift_0: std_logic := '0';

begin

ucA : uc_A
 port map(
 CLK => CLK,
 RST  => RST,
 en_cnt => en_cnt_0,
 rts => rts,
 cts => cts,
 div  => div_0,
 stop => stop,
 start => start,
 load_sr => load_sr_0,
 shift => shift_0,
 go => go
 );

uoA: uo_A
port map(
 CLK  => CLK,
 RST => RST,
 DATA => DATA,
```

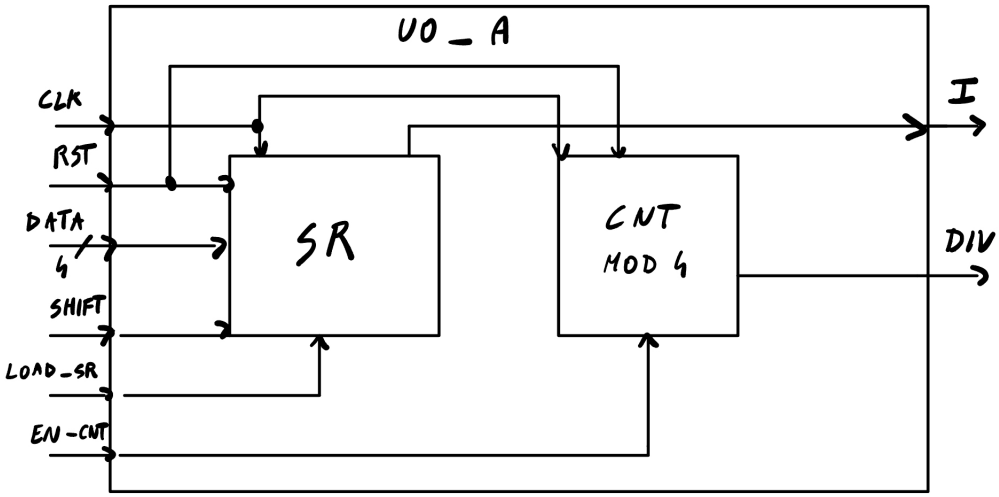

Figura 6: Unità operativa nodo A

```
  I => I,
  shift => shift_0,
  load_sr => load_sr_0,
  en_cnt => en_cnt_0,
  div => div_0
 );


end structural;
```

4.1 Unità operativa

Entity e architecture

In figura 6 è rappresentata l'unità operativa del nodo A.

```
entity uo_A is
 port(
  CLK : in std_logic;
  RST : in std_logic;
  DATA :in std_logic_vector(3 downto 0);
  I : out std_logic;
  shift : in std_logic;
```

```
  load_sr : in std_logic;
  en_cnt : in std_logic;
  div : out std_logic
 );
end uo_A;

architecture Structural of uo_A is
 component sr_par is
  port(
  CLK: in std_logic;
  RST: in std_logic;
  X: in std_logic_vector(3 downto 0);
  load : in std_logic;
  en_sr : in std_logic;
  Y: out std_logic
 );
  end component;

  component cnt4 is
  port( CLK, RST: in std_logic;
    en: in std_logic;
    count: out std_logic_vector(1 downto 0);
    div : out std_logic
    );
  end component;

  begin
  sr_par_0 : sr_par
   port map(
    CLK => CLK,
    RST => RST,
    X => Data,
    load => load_sr,
    en_sr => shift,
    Y => I
    );

   cnt_4_0 : cnt4
  port map(
   CLK => CLK,
   RST => RST,
   en => en_cnt,
   div => div
    );

end Structural;
```

4.1.1 Shift register

Entity e architecture

```
entity sr_par is
 port(
 CLK: in std_logic;
 RST: in std_logic;
 X: in std_logic_vector(3 downto 0);
 load : in std_logic;
 en_sr : in std_logic;
 Y: out std_logic
);
 end sr_par;

architecture rtl of sr_par is
 signal T: std_logic_vector(3 downto 0);
begin
 reg: process( CLK, RST )
   begin
   if( RST = '1' ) then
    T <= "0000";
   elsif( CLK'event and CLK = '0' ) then
    if (load = '1') then
     T <= X;
    elsif(en_sr = '1') then
     T(3) <= '0';
     T(2 downto 0) <= T(3 downto 1);

    end if;
   end if;
   end process;

   Y <= T(0);
end rtl;
```

4.1.2 Contatore mod4

Entity e architecture

```
entity cnt4 is
 port( CLK, RST: in std_logic;
    en: in std_logic;
    count: out std_logic_vector(1 downto 0);
```

```
    div : out std_logic);
end cnt4;

architecture behavioural of cnt4 is
 signal c: std_logic_vector(1 downto 0) := (others => '0');

 begin
 CM4: process(CLK,RST)
 begin
  if(RST = '1') then
   c <= (others=>'0');
   div <= '0';
  elsif(falling_edge(CLK))then
   if (en = '1') then
     c <= std_logic_vector(unsigned(c) + 1);
     if  (c = "10") then
      div <='1';
     else
      div <= '0';
     end if;
   end if;

  end if;
  end process;

 count<= c;
end behavioural;
```

4.2 Unità di controllo

Entity e architecture

In figura 7 è rappresentato l'automa dell'unità di controllo del nodo A.

```
    entity uc_A is
        port(
      CLK : IN STD_LOGIC;
      RST : IN STD_LOGIC;
      en_cnt : out std_logic;
      rts : out std_logic;
      cts: in std_logic;
      div : in std_logic;
      stop : in std_logic;
      load_sr : out std_logic;
```

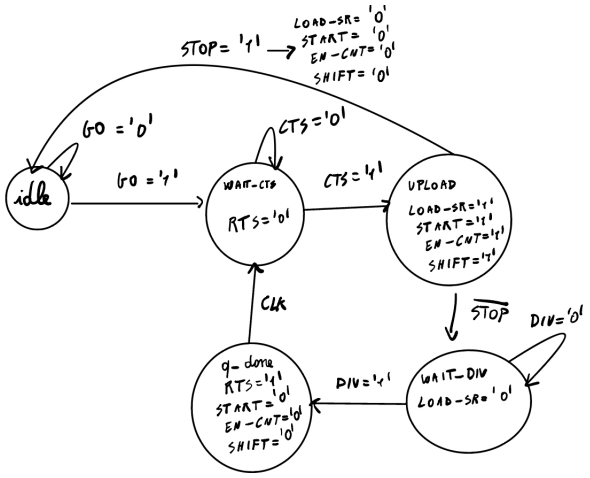

Figura 7: Unità di controllo A

```
      start : out std_logic;
      shift : out std_logic;
      go : in std_logic);
end uc_A;

architecture structural of uc_A is
 type state is (idle,wait_cts, upload,wait_div,q_done);
 signal current_state : state := idle;
 begin
 reg_stato: process(CLK)
   begin
    if(rising_edge(CLK)) then
    if rst = '1' then
   current_state<=idle;
    else
    CASE current_state is
    WHEN idle =>
        if(go= '1') then
       current_state<=wait_cts;
      end if;

        WHEN wait_cts =>
      RTS <= '0';

      if(CTS = '1') then
       current_state <= upload;
      end if;
```

```
 WHEN upload =>
   current_state <= wait_div;
   load_sr <= '1';
   start <= '1';
   en_cnt <= '1';
   shift <= '1';
   if (stop = '1') then
     current_state <= idle;
     shift <= '0';
     en_cnt <= '0';
     start <= '0';
     load_sr <= '0';
   end if;

WHEN wait_div =>
  load_sr <= '0';
  if ( DIV = '1') then
   current_state <= q_done;
  end if;

WHEN q_done =>
 shift <= '0';
 en_cnt <= '0';
 start <= '0';
 RTS <= '1';
 current_state <= wait_cts;

  end CASE;
 end if;
 end if;
  end process;
  end structural;
```

5 Nodo B

Entity e architecture

```
  entity Nodo_B is
          port(
            CLK   : in std_logic;
            START : in std_logic;
            rst : in std_logic;
            stop : out std_logic;
            I : in std_logic;
```

```
              Z : out std_logic);
end Nodo_B;

architecture Structural of Nodo_B is
 component cnt_10 is
 port(
            CLK : in std_logic;
            RST: in std_logic;
            en: in std_logic;
            count: out std_logic_vector(3 downto 0);
            div : out std_logic);
    end component;

component riconoscitore_sequenza is
    port(
            I : in std_logic :='0';
            CLK : in std_logic := '0';
            EN : in std_logic := '0';
            Y : out std_logic := '0');
 end component;
 signal Y_0 : std_logic := '0';
begin
Z <= Y_0 ;

cnt_10_is : cnt_10
  port map(
   CLK => CLK,
    RST => RST,
    en => Y_0,
    div  =>stop
    );

ric: riconoscitore_sequenza
 port map(
           I => I,
           CLK => clk,
           EN => start,
           Y => Y_0);
end Structural;
```

5.1 Riconoscitore di sequenza

Entity e architecture

A seguire è riportata la descrizione vhdl del riconoscitore e il relativo automa in figura 8.

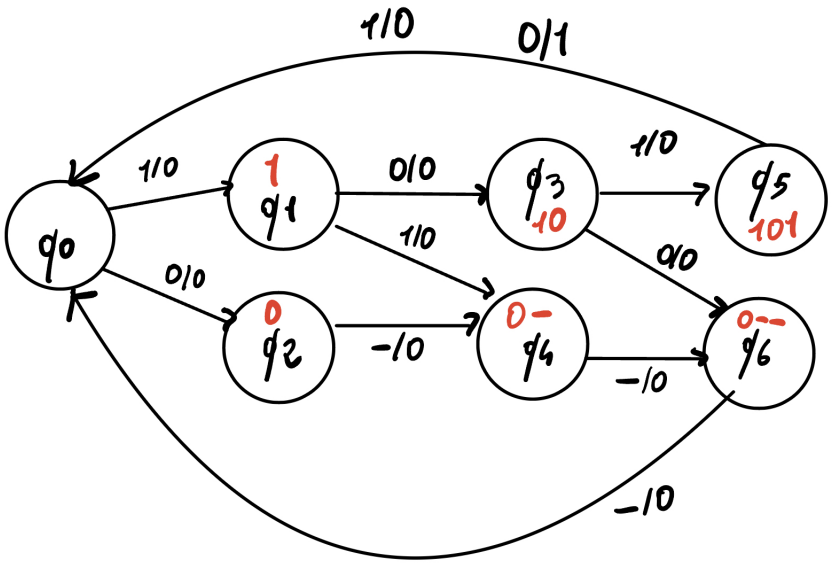

Figura 8: Automa riconoscitore

```
    entity riconoscitore_sequenza is
    port(
           I : in std_logic :='0';
           CLK : in std_logic := '0';
           EN : in std_logic := '0';
           Y : out std_logic := '0');
end riconoscitore_sequenza;

architecture Behavioral of riconoscitore_sequenza is
 type stato is (Q0, Q1, Q2, Q3, Q4, Q5, Q6);
 signal stato_corrente : stato := Q0;
    begin
reg : process(CLK)
 begin
 if (rising_edge(CLK)) then
  if (EN = '1') then
   case stato_corrente iS
    when Q0 =>
    y <= '0';
        if(I = '0') then
      stato_corrente <= Q2;
        else
      stato_corrente <= Q1;
        end if;
```

```
    when Q1 =>
     if(I = '0') then
      stato_corrente <= Q3;
     else
      stato_corrente <= Q4;
     end if;

    when Q2 =>
     stato_corrente <= Q4;

    when Q3 =>
     if(I = '0') then
      stato_corrente <= Q6;
     else
      stato_corrente <= Q5;
     end if;
    when Q4 =>
     stato_corrente <= Q6;

    when Q5 =>
     if(I = '0') then
      stato_corrente <= Q0;
      y <='1';
     else
      stato_corrente <= Q0;
      y <= '0';
     end if;

    when Q6 =>
     stato_corrente <= Q0;
     y <='0';
   end case;
  else
   y <='0';
   end if;
  end if;
 end process;
end Behavioral;
```

5.2 Contatore mod10

Entity e architecture

```
    entity cnt_10 is
 port( CLK, RST: in std_logic;
```

```
    en: in std_logic;
    count: out std_logic_vector(3 downto 0);
    div : out std_logic);
end cnt_10;

architecture behavioural of cnt_10 is
 signal c: std_logic_vector(3 downto 0) := (others => '0');
 begin
 CM16: process(CLK,RST)
 begin
  if(RST = '1') then
   c <= (others=>'0');
   div <= '0';
  elsif(falling_edge(CLK))then
   if  (c = "1010") then
     div <='1';
   elsif (en = '1') then
     c <= std_logic_vector(unsigned(c) + 1);
     div <= '0';
   else div <='0';
   end if;
  end if;
  end process;

 count<= c;
end behavioural;
```

6 Sistema

Entity e architecture

In figura 9 è rappresentato il sistema composto dai nodi A e B.

```
    entity system is
  port(
            CLK : IN STD_LOGIC;
            RST : IN STD_LOGIC;
            rts : out std_logic;
            cts: in std_logic;
            DATA : in std_logic_vector(3 downto 0);
            Z : out std_logic;
            go  : in std_logic);
end system;

architecture Structural of system is
component Nodo_A is
```

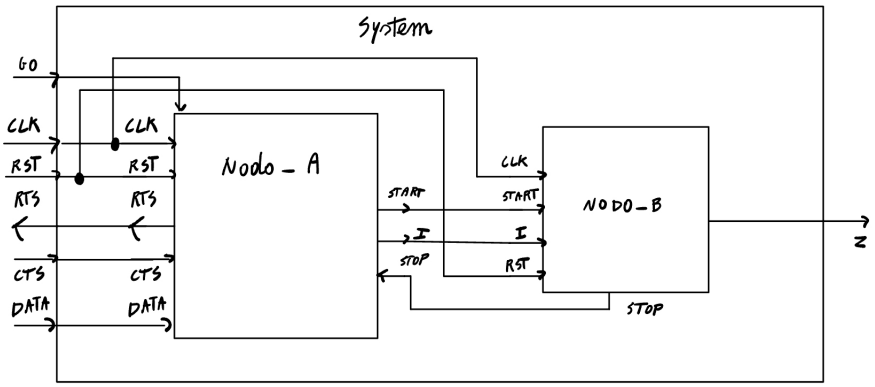

Figura 9: Schema system A e B

```
port(
  CLK : IN STD_LOGIC;
  RST : IN STD_LOGIC;
  rts : out std_logic;
  cts: in std_logic;
  DATA : in std_logic_vector(3 downto 0);
  I : out std_logic;
  start : out std_logic;
  stop : in std_logic;
  go : in std_logic);
end component;

component Nodo_B is
 port(
  CLK  : in std_logic;
  START : in std_logic;
  rst : in std_logic;
  stop : out std_logic;
  I : in std_logic;
  Z : out std_logic);
 end component;

signal start_0 : std_logic:= '0';
signal stop_0 : std_logic:= '0';
signal I_0 : std_logic:= '0';

begin
 nodoA : Nodo_A
  port map(
```

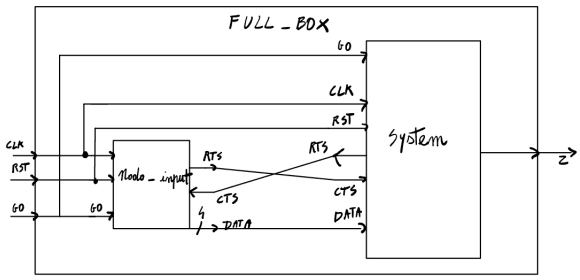

Figura 10: Schema Full Box

```
  CLK => CLK,
  RST => RST,
  rts => rts,
  cts => cts,
  DATA => data,
  I => I_0,
  start => start_0,
  stop => stop_0,
  go => go);

 nodoB : Nodo_B
 port map(
  CLK => CLK,
  start => start_0,
  rst => rst,
  stop => stop_0,
  I => I_0,
  Z => Z);
end Structural;
```

7 Full box

Entity e architecture

In figura 10 è rappresentata l'architettura complessiva.

```
    entity full_box is
  port(
     CLK : in std_logic;
     RST : in std_logic;
     go : in std_logic;
     Z : out std_logic);
end full_box;
```

```
architecture Structural of full_box is
component system is
  port(
            CLK : IN STD_LOGIC;
      RST : IN STD_LOGIC;
      rts : out std_logic;
      cts: in std_logic;
      DATA : in std_logic_vector(3 downto 0);
      Z : out std_logic;
      go : in std_logic);
end component;

component Nodo_input is
    port(
  CLK : IN STD_LOGIC;
  RST : IN STD_LOGIC;
  rts : out std_logic;
  cts: in std_logic;
  DATA : out std_logic_vector(3 downto 0);
  go : in std_logic);
 end component;

signal rts_0 : std_logic :='0';
signal cts_0 : std_logic :='0';
signal DATA_0 :  std_logic_vector(3 downto 0);

begin
 sys : system
  port map(
            CLK  => CLK,
      RST => RST,
      rts  => rts_0,
      cts => cts_0,
      DATA => data_0,
      Z => Z,
      go => go);

 N_in :  Nodo_input
 port map(
  CLK  => CLK,
  RST  => RST,
  rts  => cts_0,
  cts  => rts_0,
  DATA => data_0,
  go => go);
```

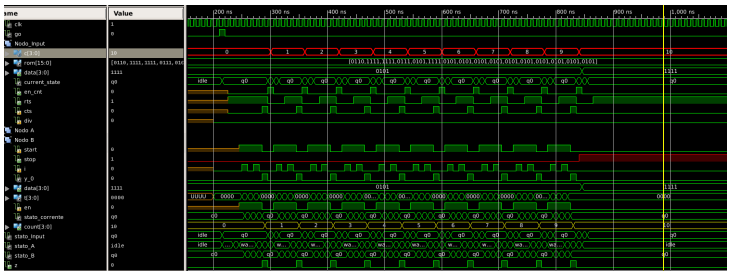

Figura 11: Simulazione finale

```
end Structural;
```

In figura 11 è mostrato un esempio di simulazione del sistema complessivo.