

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

Studente:

ALEONSO MOGGIO M63001772

Sommario

Capitolo 1: Reti Combinatorie Elementari.....	5
Esercizio 1: Multiplexer 16:1	5
Esercizio 1.1	5
Progetto e architettura	5
Implementazione.....	6
Simulazione.....	7
Esercizio 1.2	11
Progetto e architettura	11
Implementazione.....	11
Simulazione.....	13
Esercizio 1.3	15
Sintesi su board di sviluppo	15
Esercizio 2: Sistema ROM + M	20
Esercizio 2.1	20
Progetto e architettura	20
Implementazione.....	21
Simulazione.....	23
Esercizio 2.2	24
Sintesi su board di sviluppo	24
Capitolo 2: Reti Sequenziali Elementari.....	26
Esercizio 3: Riconoscitore di sequenze	26
Esercizio 3.1	26
Progetto e architettura	26
Implementazione.....	27
Simulazione.....	29
Esercizio 3.2	32
Sintesi su board di sviluppo	32
Esercizio 4: Shift Register.....	36
Esercizio 4.1	36
Progetto e architettura	36
Implementazione.....	36
Esercizio 5: Cronometro	40
Esercizio 5.1	40
Progetto e architettura	40
Implementazione.....	41
Simulazione.....	44
Esercizio 5.2	46
Sintesi su board di sviluppo	46

Esercizio 6: Sistema di lettura-elaborazione-scrittura PO_PC	47
Esercizio 6.1	47
Progetto e architettura	47
Implementazione	49
Esercizio 6.2	54
Sintesi su board di sviluppo	54
Capitolo 3: Esercizi sulle macchine aritmetiche	55
Esercizio 7: Moltiplicatore di Booth.....	55
Esercizio 7.1	55
Progetto e architettura	55
Implementazione.....	56
Esercizio 7.2	56
Progetto e architettura	56
Implementazione.....	56
Sintesi su board di sviluppo	57
Capitolo 4: Handshaking	58
Esercizio 8: Comunicazione con Handshaking	58
Esercizio 8.1	58
Progetto e architettura	58
Implementazione.....	60
Simulazione.....	73
Capitolo 5: Processore	75
Esercizio 9	75
Progetto e architettura	75
Approfondimento Istruzioni a scelta	75
Capitolo 6: Interfaccia Seriale.....	78
Esercizio 10	78
Progetto e architettura	78
Implementazione.....	80
Simulazione.....	91
Capitolo 7: Switch Multistadio.....	93
Esercizio 11: Switch Multistadio	93
Esercizio 11.1	93
Progetto e architettura	93
Implementazione.....	93
Capitolo 8: Prova Dicembre	94
Esercizio 12	94
Progetto e architettura	94
Implementazione.....	95
Simulazione.....	106

Appendice.....	109
Multiplexer 2:1	109
Progetto e architettura.....	109
Implementazione.....	109
Multiplexer 4:1	110
Progetto e architettura.....	110
Implementazione.....	110
Demultiplexer 1:4	112
Progetto e architettura.....	112
Implementazione.....	112
Contatore.....	114
Progetto e architettura.....	114
Implementazione.....	114
Registro ad N bit.....	116
Progetto e architettura.....	116
Implementazione.....	116
Full Adder	118
Progetto e architettura.....	118
Implementazione.....	118
Carry Look-Ahead	119
Progetto e architettura.....	119
Implementazione.....	120
Ripple Carry Adder.....	123
Progetto e architettura.....	123

Capitolo 1: Reti Combinatorie Elementari

Esercizio 1: Multiplexer 16:1

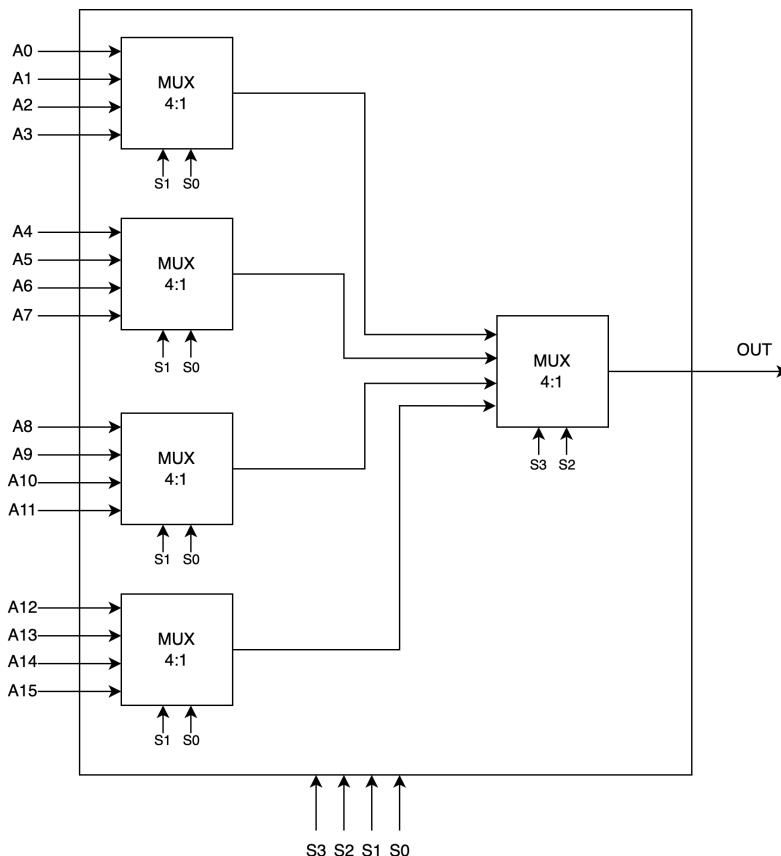
Esercizio 1.1

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

Ai fini del progetto, si è innanzitutto importato gli elementi fondamentali per realizzare il multiplexer 16:1, ovvero il multiplexer 4:1 (la cui descrizione è riportata nell'appendice) e, di conseguenza, il multiplexer 2:1 (anch'esso in appendice). A questo punto, partendo dai mux 4:1, si è proceduto all'implementazione, tramite composizione, del multiplexer 16:1.

Di seguito viene riportato il disegno strutturale:



Dal punto di vista funzionale, abbiamo che ognuno dei 16 ingressi viene inviato all'uscita in base ai bit di selezione (di numero 4). L'output finale sarà il risultato dell'uscita di uno dei 4 multiplexer 4:1, selezionato in base ai 4 bit di indirizzo di selezione.

L'implementazione prevede che i 4 bit di selezione siano divisi in due gruppi:

- I primi 2 bit determinano quale multiplexer 4:1 utilizzare (ovvero quale “gruppo” di 4 ingressi scegliere).
- I successivi 2 bit sono usati per selezionare l'ingresso all'interno del mux 4:1 scelto.

Implementazione

Di seguito viene riportato il codice del multiplexer 16:1 :

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

entity mux_16_1 is
    port ( b0 : in STD_LOGIC;
           b1 : in STD_LOGIC;
           b2 : in STD_LOGIC;
           b3 : in STD_LOGIC;
           b4 : in STD_LOGIC;
           b5 : in STD_LOGIC;
           b6 : in STD_LOGIC;
           b7 : in STD_LOGIC;
           b8 : in STD_LOGIC;
           b9 : in STD_LOGIC;
           b10 : in STD_LOGIC;
           b11 : in STD_LOGIC;
           b12 : in STD_LOGIC;
           b13 : in STD_LOGIC;
           b14 : in STD_LOGIC;
           b15 : in STD_LOGIC;
           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;
           s1 : in STD_LOGIC;
           y : out STD_LOGIC
);
end component;

begin
    mux0: mux_4_1
        Port map ( a0 => b0,
                   a1 => b1,
```

```

        a2 => b2,
        a3 => b3,
        s0 => s0,
        s1 => s1,
        y  => u0
    );
mux1: mux_4_1
    Port map (
        a0 => b4,
        a1 => b5,
        a2 => b6,
        a3 => b7,
        s0 => s0,
        s1 => s1,
        y  => u1
    );
mux2: mux_4_1
    Port map (
        a0 => b8,
        a1 => b9,
        a2 => b10,
        a3 => b11,
        s0 => s0,
        s1 => s1,
        y  => u2
    );
mux3: mux_4_1
    Port map (
        a0 => b12,
        a1 => b13,
        a2 => b14,
        a3 => b15,
        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

Il seguente testbench serve a verificare il funzionamento del multiplexer 16:1. Per farlo, viene prima dichiarato il componente da testare, che ha 16 ingressi, 4 segnali di selezione e un'uscita.

Successivamente, vengono definiti i segnali di test: un vettore di ingressi, un vettore di controllo per selezionare quale ingresso instradare verso l'uscita, e un segnale per l'uscita. Questi segnali vengono poi collegati ai corrispondenti ingressi e uscite del multiplexer tramite l'istanza uut.

Nel processo di stimolo, dopo un breve ritardo iniziale, viene assegnato un valore agli ingressi. A questo punto, vengono cambiati i segnali di selezione per verificare che l'uscita del multiplexer corrisponda effettivamente al valore dell'ingresso selezionato. Dopo alcuni test, la simulazione viene interrotta.

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

entity tb_mux_16_1 is
end tb_mux_16_1;

architecture tb of tb_mux_16_1 is

-- Dichiarazione del componente mux_16_1
component mux_16_1
    port (
        b0 : in std_logic;
        b1 : in std_logic;
        b2 : in std_logic;
        b3 : in std_logic;
        b4 : in std_logic;
        b5 : in std_logic;
        b6 : in std_logic;
        b7 : in std_logic;
        b8 : in std_logic;
        b9 : in std_logic;
        b10 : in std_logic;
        b11 : in std_logic;
        b12 : in std_logic;
        b13 : in std_logic;
        b14 : in std_logic;
        b15 : in std_logic;
        s0 : in std_logic;
        s1 : in std_logic;
        s2 : in std_logic;
        s3 : in std_logic;
        y0 : out std_logic
    );
end component;

-- Segnali di test
signal input : std_logic_vector(15 downto 0) := (others => '0');
```

```

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

begin

-- Istanza del multiplexer 16:1
uut: mux_16_1
port map (
    b0 => input(0),
    b1 => input(1),
    b2 => input(2),
    b3 => input(3),
    b4 => input(4),
    b5 => input(5),
    b6 => input(6),
    b7 => input(7),
    b8 => input(8),
    b9 => input(9),
    b10 => input(10),
    b11 => input(11),
    b12 => input(12),
    b13 => input(13),
    b14 => input(14),
    b15 => input(15),
    s0 => control(0),
    s1 => control(1),
    s2 => control(2),
    s3 => control(3),
    y0 => output
);
;

-- Processo di stimolo
stim_proc: process
begin
    wait for 100 ns;
    -- Imposta i valori di input e selezione
    input <= "1010101010101010"; -- Alterna ingressi 1 e 0

    -- Test per selezioni
    control <= "0000"; -- Seleziona b0 (1)
    wait for 10 ns;

    control <= "0001"; -- Seleziona b1 (0)
    wait for 10 ns;

    control <= "0010"; -- Seleziona b2 (1)
    wait for 10 ns;

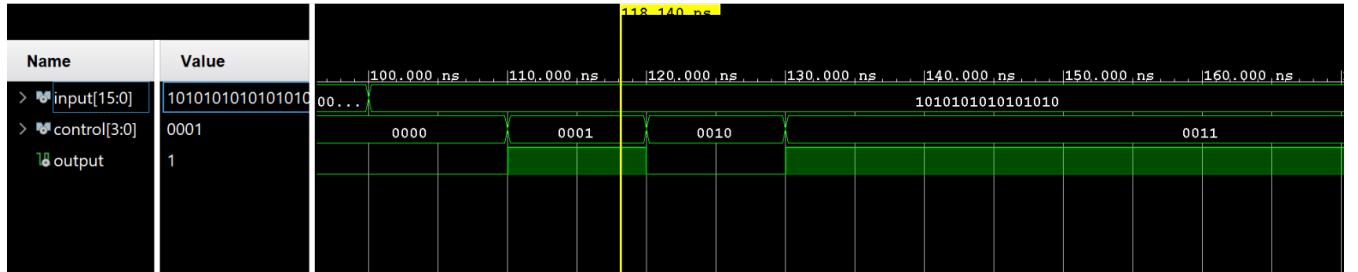
    control <= "0011"; -- Seleziona b3 (0)
    wait for 10 ns;

    wait; -- Termina la simulazione

```

```
end process;  
  
end tb;
```

Di seguito è riportato il risultato grafico della simulazione:



Il risultato ottenuto è conforme alle aspettative. Ad esempio, con l'ingresso impostato a 1010101010101010 e il segnale di selezione 0001, l'uscita risulta alta.

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**.

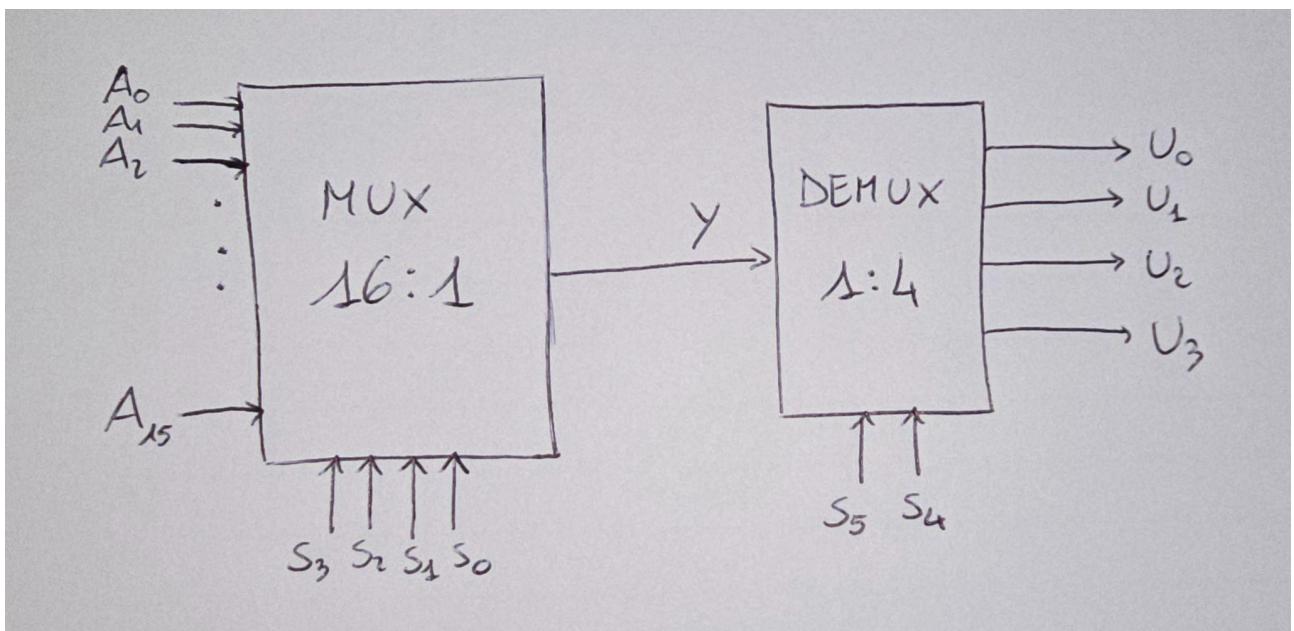
Progetto e architettura

Per la realizzazione della rete di interconnessione a 16 sorgenti e 4 destinazioni, l'approccio adottato è basato sull'utilizzo di componenti modulari riutilizzabili, integrati secondo un'architettura strutturale. In particolare, la rete è stata progettata con l'impiego di un multiplexer (MUX) a 16:1 e di un demultiplexer (DEMUX) a 1:4.

L'architettura si basa su una logica di suddivisione e selezione:

- Il multiplexer riceve come ingressi i 16 segnali provenienti dalle sorgenti e, mediante un segnale di selezione a 4 bit, instrada uno solo di questi verso l'uscita.
- L'uscita del multiplexer è poi instradata al demultiplexer, che attraverso un segnale di selezione a 2 bit, ridirige il segnale selezionato verso una delle 4 destinazioni finali.

Di seguito viene riportato il disegno:



Implementazione

Di seguito viene riportato il codice VHDL dell'interconnessione (con il codice del demux riportato invece nell'appendice):

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

entity interconnection is
Port (
    sorgenti : in STD_LOGIC_VECTOR(15 downto 0);
    sorgenti_sel : in STD_LOGIC_VECTOR(3 downto 0);
```

```

destinazioni_sel : in STD_LOGIC_VECTOR(1 downto 0);
destinazioni   : out STD_LOGIC_VECTOR(3 downto 0)
);
end interconnection;

-- LOGICA DI INTERCONNESSIONE:
architecture structural of interconnection is
    -- Segnale interno tra MUX e DEMUX
    signal mux_output : STD_LOGIC;

    component mux_16_1 is
        port(
            b0, b1, b2, b3, b4, b5, b6, b7,
            b8, b9, b10, b11, b12, b13, b14, b15 : in STD_LOGIC;
            s0, s1, s2, s3 : in STD_LOGIC;
            y0 : out STD_LOGIC
        );
    end component;

    component dmux_1_4 is
        port(
            c0 : in STD_LOGIC;
            s4 : in STD_LOGIC;
            s5 : in STD_LOGIC;
            d0 : out STD_LOGIC;
            d1 : out STD_LOGIC;
            d2 : out STD_LOGIC;
            d3 : out STD_LOGIC
        );
    end component;

begin
    -- Istanza del MUX
    MUX: mux_16_1
        port map(
            b0 => sorgenti(0),
            b1 => sorgenti(1),
            b2 => sorgenti(2),
            b3 => sorgenti(3),
            b4 => sorgenti(4),
            b5 => sorgenti(5),
            b6 => sorgenti(6),
            b7 => sorgenti(7),
            b8 => sorgenti(8),
            b9 => sorgenti(9),
            b10 => sorgenti(10),
            b11 => sorgenti(11),
            b12 => sorgenti(12),
            b13 => sorgenti(13),
            b14 => sorgenti(14),
            b15 => sorgenti(15),

```

```

s0 => sorgenti_sel(0),
s1 => sorgenti_sel(1),
s2 => sorgenti_sel(2),
s3 => sorgenti_sel(3),
y0 => mux_output
);

-- Istanza del DMUX
DMUX: dmux_1_4
port map(
    c0 => mux_output,
    s4 => destinazioni_sel(0),
    s5 => destinazioni_sel(1),
    d0 => destinazioni(0),
    d1 => destinazioni(1),
    d2 => destinazioni(2),
    d3 => destinazioni(3)
);
end structural;

```

Simulazione

Il seguente testbench serve a verificare il corretto funzionamento del modulo interconnection, che collega una sorgente selezionata tra 16 ingressi a una delle 4 destinazioni disponibili.

All'inizio, i segnali vengono inizializzati con valori predefiniti: le sorgenti sono impostate a 10101010101010, mentre le selezioni di sorgenti_sel e destinazioni_sel partono da zero.

Nel processo di stimolo, i segnali di selezione vengono modificati per testare diverse combinazioni, verificando che la sorgente selezionata venga correttamente instradata alla destinazione desiderata. La simulazione termina dopo l'ultimo test.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_interconnection is
end tb_interconnection;

architecture tb of tb_interconnection is

component interconnection
port (
    sorgenti      : in std_logic_vector (15 downto 0);
    sorgenti_sel  : in std_logic_vector (3 downto 0);
    destinazioni_sel : in std_logic_vector (1 downto 0);
    destinazioni  : out std_logic_vector (3 downto 0)
);
end component;

```

```

signal sorgenti      : std_logic_vector (15 downto 0) := (others => '0');
signal sorgenti_sel  : std_logic_vector (3 downto 0) := (others => '0');
signal destinazioni_sel : std_logic_vector (1 downto 0) := (others => '0');
signal destinazioni   : std_logic_vector (3 downto 0);

begin
  uut : interconnection
    port map (
      sorgenti      => sorgenti,
      sorgenti_sel  => sorgenti_sel,
      destinazioni_sel => destinazioni_sel,
      destinazioni   => destinazioni
    );

  stim_proc: process
    begin
      -- Inizializzazione dei segnali
      sorgenti      <= "1010101010101010"; -- Valori iniziali per sorgenti
      sorgenti_sel  <= "0000"; -- Selezione iniziale delle sorgenti
      destinazioni_sel <= "00"; -- Selezione iniziale delle destinazioni
      wait for 100 ns;

      -- Stimoli per testare il comportamento
      sorgenti_sel <= "0101";
      destinazioni_sel <= "10";
      wait for 200 ns;

      sorgenti_sel  <= "0000";
      destinazioni_sel <= "00";
      wait for 100 ns;

      sorgenti_sel <= "0001";
      destinazioni_sel <= "01";
      wait for 200 ns;

      sorgenti_sel  <= "0000";
      destinazioni_sel <= "00";
      wait for 100 ns;

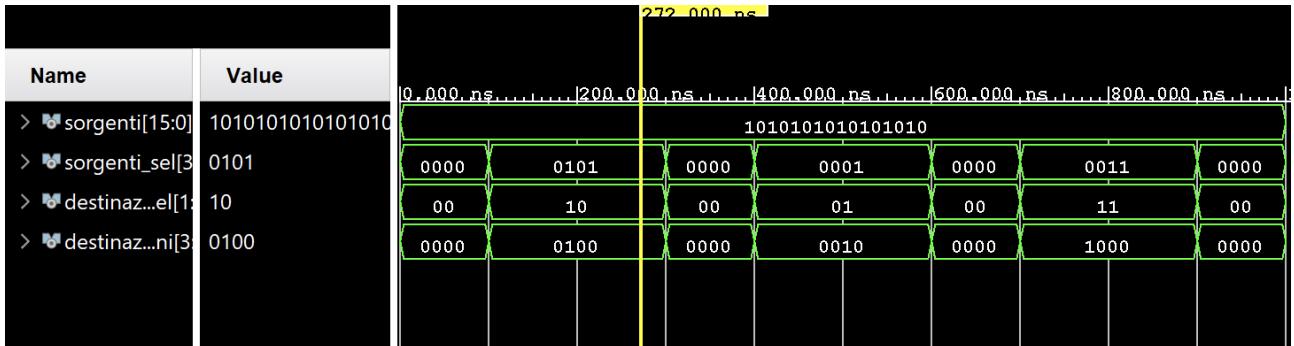
      sorgenti_sel <= "0011";
      destinazioni_sel <= "11";
      wait for 200 ns;

      sorgenti_sel  <= "0000";
      destinazioni_sel <= "00";
      wait for 100 ns;
      wait;
    end process;

end tb;

```

Come possiamo osservare, il risultato della simulazione (riportato di seguito), è conforme alle aspettative. Ad esempio, dato l'ingresso del mux 16:1 1010101010101010, la selezione sorgente 0101 e la selezione destinazione 10, possiamo osservare come viene riportato il bit 1 sull'uscita 3 del demux 1:4.



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 implementare su board la rete di interconnessione sviluppata nel punto 1.2, è stata progettata un'unità di controllo per l’acquisizione dei 16 bit di input tramite gli 8 switch disponibili sulla scheda. Il funzionamento prevede l’inserimento degli 8 bit meno significativi, seguito dalla pressione del pulsante sinistro (BTNL), e successivamente degli 8 bit più significativi, confermati con il pulsante destro (BTNR). Inoltre, sono stati predisposti 6 switch aggiuntivi per l’inserimento del segnale di selezione della rete di interconnessione, la cui acquisizione avviene premendo il pulsante centrale (BTNC). Infine, il reset del sistema è gestito tramite il pulsante BTND.

Di seguito è riportato il codice dell’unità di controllo, del top level e del file di constraint:

- Unità di controllo

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

entity control_unit is
  Port (
    clock : in STD_LOGIC;
    reset : in STD_LOGIC;
    load_first_part : in STD_LOGIC;
    load_second_part : in STD_LOGIC;
    load_sel : in STD_LOGIC;
    value8_in : in STD_LOGIC_VECTOR(7 downto 0); --valore acquisito mediante 8 switch
    valuesel_in : in STD_LOGIC_VECTOR(5 downto 0);
    valuesel_out : out STD_LOGIC_VECTOR(5 downto 0);
  );
end entity;
```

```

    value16_out : out STD_LOGIC_VECTOR(15 downto 0) --valore su 16 bit, formato dai due valori da 8
bit acquisiti
);
end control_unit;

architecture Behavioral of control_unit is

signal reg_value : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
signal reg_valuesel_out : STD_LOGIC_VECTOR(5 downto 0) := (others => '0');

begin

value16_out <= reg_value;
valuesel_out <= reg_valuesel_out;

main: process(clock)
begin

if clock'event and clock = '1' then
  if reset = '1' then
    reg_value <= (others => '0');
  else
    if load_first_part = '1' then
      reg_value(7 downto 0) <= value8_in;
    elsif load_second_part = '1' then
      reg_value(15 downto 8) <= value8_in;
    elsif load_sel = '1' then
      reg_valuesel_out <= valuesel_in;
    end if;
  end if;
end if;

end process;
end Behavioral;

```

- Interconnection on board (top level)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity interconnection_on_board is
  Port (
    clock : in STD_LOGIC; --clock board
    reset : in STD_LOGIC; --reset, associato a un bottone
    load_first_part : in STD_LOGIC; --comando di caricamento 8bit meno significativi, associato a un
bottone
    load_second_part : in STD_LOGIC; --comando di caricamento 8bit più significativi, associato a un
bottone
    load_sel : in STD_LOGIC;
    value8_in : in STD_LOGIC_VECTOR(7 downto 0);

```

```

valuesel_in : in STD_LOGIC_VECTOR(5 downto 0);
led : out STD_LOGIC_VECTOR(3 downto 0)
);
end interconnection_on_board;

architecture Structural of interconnection_on_board is

COMPONENT control_unit
PORT(
    clock : in STD_LOGIC;
    reset : in STD_LOGIC;
    load_first_part : in STD_LOGIC;
    load_second_part : in STD_LOGIC;
    load_sel : in STD_LOGIC;
    value8_in : in STD_LOGIC_VECTOR(7 downto 0); --valore acquisito mediante 8 switch
    valuesel_in : in STD_LOGIC_VECTOR(5 downto 0);
    valuesel_out : out STD_LOGIC_VECTOR(5 downto 0);

    value16_out : out STD_LOGIC_VECTOR(15 downto 0) --valore su 16 bit, formato dai due valori da 8
bit acquisiti
);
END COMPONENT;


COMPONENT interconnection
port(
    -- 16 sorgenti per il MUX + 4 selezioni
    sorgenti : in STD_LOGIC_VECTOR(15 downto 0);
    sorgenti_sel : in STD_LOGIC_VECTOR(3 downto 0);
    -- 2 selezione per il DEMUX + 4 output
    destinazioni_sel : in STD_LOGIC_VECTOR(1 downto 0);
    destinazioni : out STD_LOGIC_VECTOR(3 downto 0)

);
END COMPONENT;


signal cu_value : std_logic_vector(15 downto 0);
signal cu_valuesel : std_logic_vector(5 downto 0);

begin

cu: control_unit PORT MAP(
    clock => clock,
    reset => reset,
    load_first_part => load_first_part,
    load_second_part => load_second_part,
    load_sel => load_sel,
    value8_in => value8_in,
    valuesel_in => valuesel_in,
    valuesel_out => cu_valuesel,
    value16_out => cu_value
);

```

```

i : interconnection port map(
    sorgenti => cu_value,
    sorgenti_sel => cu_valuesel (5 downto 2),
    destinazioni_sel => cu_valuesel (1 downto 0),
    destinazioni => led
);

```

end Structural;

- Constraint

```

## This file is a general .xdc for the Nexys A7-50T
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
#
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVC MOS33 } [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 LVC MOS33 } [get_ports { value8_in[0] }];
#IO_L24N_T3_RSO_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVC MOS33 } [get_ports { value8_in[1] }];
#IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVC MOS33 } [get_ports { value8_in[2] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVC MOS33 } [get_ports { value8_in[3] }];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVC MOS33 } [get_ports { value8_in[4] }];
#IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVC MOS33 } [get_ports { value8_in[5] }];
#IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVC MOS33 } [get_ports { value8_in[6] }];
#IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVC MOS33 } [get_ports { value8_in[7] }];
#IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVC MOS18 } [get_ports { valuesel_in[0] }]; #IO_L24N_T3_34
Sch=sw[8]
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVC MOS18 } [get_ports { valuesel_in[1] }]; #IO_25_34
Sch=sw[9]
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVC MOS33 } [get_ports { valuesel_in[2] }];
#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVC MOS33 } [get_ports { valuesel_in[3] }];
#IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVC MOS33 } [get_ports { valuesel_in[4] }]; #IO_L24P_T3_35
Sch=sw[12]
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVC MOS33 } [get_ports { valuesel_in[5] }];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVC MOS33 } [get_ports { SW[14] }];
#IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVC MOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14

```

```

Sch=sw[15]

## 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]
#set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14
Sch=led[4]
#set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { LED[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { LED[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
#set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { LED[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { LED[8] }];
#IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14
Sch=led[9]
#set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { LED[10] }];
#IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { LED[11] }];
#IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
#set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { LED[12] }]; #IO_L16P_T2_CSI_B_14
Sch=led[12]
#set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { LED[13] }];
#IO_L22N_T3_A04_D20_14 Sch=led[13]
#set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { LED[14] }];
#IO_L20N_T3_A07_D23_14 Sch=led[14]
#set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { LED[15] }];
#IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

##Buttons
#set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports { load_sel }]; #IO_L9P_T1_DQS_14
Sch=btnc
#set_property -dict { PACKAGE_PIN M18 IOSTANDARD LVCMOS33 } [get_ports { BTNU }]; #IO_L4N_T0_D05_14
Sch=btnu
set_property -dict { PACKAGE_PIN P17 IOSTANDARD LVCMOS33 } [get_ports { load_first_part }];
#IO_L12P_T1_MRCC_14 Sch=btln
set_property -dict { PACKAGE_PIN M17 IOSTANDARD LVCMOS33 } [get_ports { load_second_part }];
#IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18 IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L9N_T1_DQS_D13_14
Sch=btnd

```

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

La progettazione del sistema S si basa sulla composizione di due moduli principali:

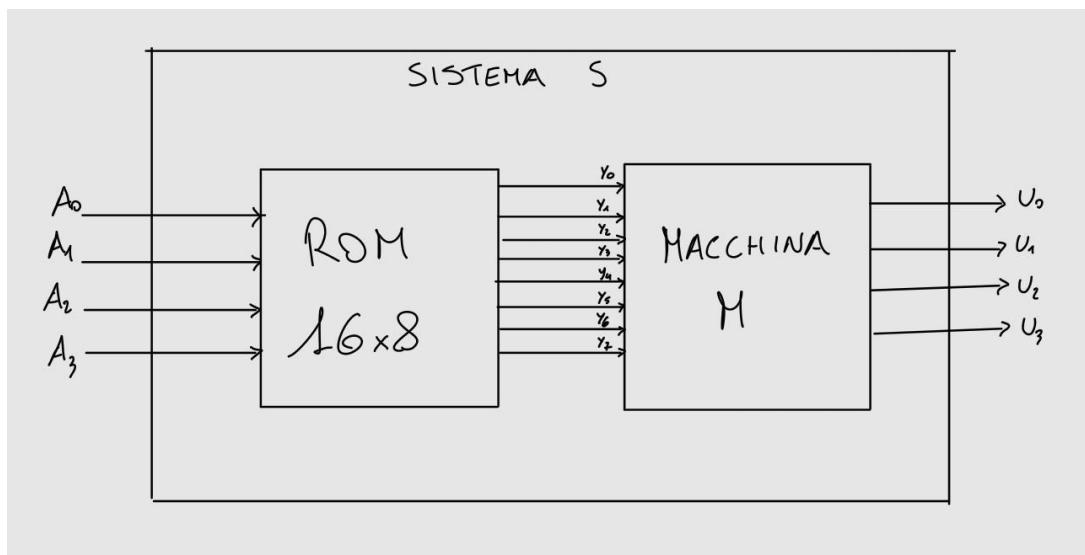
1. ROM (Read-Only Memory):

Un'unità combinatoria che fornisce un valore di 8 bit corrispondente all'indirizzo di ingresso a 4 bit. È stata implementata usando il costrutto `with...select` per mappare ciascun indirizzo (16 locazioni totali) al valore desiderato.

2. Macchina combinatoria M:

Essendo a scelta, è stata utilizzata un'unità combinatoria che prende in ingresso 8 bit dalla ROM e restituisce i 4 bit più significativi (MSB) dell'input come risultato. Il comportamento di questa macchina è progettato per estrarre e trasformare selettivamente i dati prodotti dalla ROM, fornendo così la riduzione desiderata.

Di seguito viene riportato il disegno:



In pratica, il sistema S opera nel seguente modo:

• Indirizzamento della ROM:

- Fornendo un indirizzo addr a 4 bit, il modulo ROM restituisce un valore a 8 bit che è predefinito per ciascuna delle sue 16 locazioni.
- La ROM è definita come una memoria puramente combinatoria, ossia senza clock, rendendola immediata nel rispondere agli ingressi.

- **Elaborazione del dato dalla macchina M:**
 - L'output a 8 bit della ROM viene fornito come input alla macchina combinatoria M.
 - La macchina M estrae i 4 bit più significativi (MSB) del dato proveniente dalla ROM e li restituisce come output finale del sistema.

Implementazione

Di seguito viene riportato il codice VHDL:

- Macchina M

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Definizione Macchina M
entity M is
  Port(
    x : in STD_LOGIC_VECTOR(7 downto 0); -- Ingresso a 8 bit
    y : out STD_LOGIC_VECTOR(3 downto 0) -- Uscita a 4 bit
  );
end M;

architecture Behavioral of M is -- Mette in uscita solo i 4 MSB

begin
  y <= x(7 downto 4);

end Behavioral;

```

- ROM 16x8

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Definizione ROM 16x8:
entity ROM is
  Port(
    addr: in STD_LOGIC_VECTOR(3 downto 0); -- Indirizzo di 4 bit in ingresso (su 16 locazioni)
    data: out STD_LOGIC_VECTOR(7 downto 0) -- Uscita su 8 bit
  );
end ROM;

architecture Behavioral of ROM is
begin
  -- Mappatura dei 4 bit sulle 16 celle, per poi produrre l'uscita a 8 bit
  with addr select
    data <= "00000001" when "0000", -- Locazione 0
      "00000010" when "0001", -- Locazione 1
      "00000100" when "0010", -- Locazione 2
      others => "00000000";
end Behavioral;

```

```

    "00001000" when "0011", -- Locazione 3
    "00010000" when "0100", -- Locazione 4
    "00100000" when "0101", -- Locazione 5
    "01000000" when "0110", -- Locazione 6
    "10000000" when "0111", -- Locazione 7
    "10000001" when "1000", -- Locazione 8
    "10000010" when "1001", -- Locazione 9
    "10000100" when "1010", -- Locazione 10
    "10001000" when "1011", -- Locazione 11
    "10010000" when "1100", -- Locazione 12
    "10100000" when "1101", -- Locazione 13
    "11000000" when "1110", -- Locazione 14
    "10100000" when "1111", -- Locazione 15
    "-----" when others; -- Valore di default
end Behavioral;

```

- Sistema S (top level)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- DEFINIZIONE SISTEMA S:
entity S is
  Port(
    addr: in STD_LOGIC_VECTOR(3 downto 0);
    out_final: out STD_LOGIC_VECTOR(3 downto 0);
  );
end S;

architecture Behavioral of S is
  -- Segnale di interconnessione
  signal data_rom: STD_LOGIC_VECTOR(7 downto 0);

begin
  -- Dichirazione oggetti:
  ROM1: entity work.ROM
    port map(
      addr => addr,
      data => data_rom
    );

  ENC1: entity work.M
    port map(
      x => data_rom,
      y => out_final
    );
end Behavioral;

```

Simulazione

Il seguente testbench verifica il funzionamento del **Sistema S**, che prende in ingresso un indirizzo a 4 bit e fornisce un'uscita a 4 bit.

Il processo di test genera diversi valori per addr, applicandoli con un intervallo di tempo definito (100 ns iniziali e poi 10 ns per ogni variazione). In questo modo, si osserva come cambia l'output in risposta agli input, permettendo di valutare il comportamento del sistema.

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

entity tb_Sistema_S is
end tb_Sistema_S;

architecture tb of tb_Sistema_S is
component Sistema_S
port (
    addr : in std_logic_vector (3 downto 0);
    out_final : out std_logic_vector (3 downto 0)
);
end component;

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

begin
uut : Sistema_S
port map (
    addr => addr,
    out_final => out_final
);

stim_proc: process
begin
    -- Inizializzazione dei segnali
    addr <= "0000"; -- Indirizzo iniziale

    wait for 100 ns;

    -- Stimoli per testare il comportamento
    addr <= "0001"; -- Indirizzo 1
    wait for 10 ns;

    addr <= "0110"; -- Indirizzo 2
    wait for 10 ns;

    addr <= "1100"; -- Indirizzo 3
    wait for 10 ns;

    addr <= "1101"; -- Indirizzo 4
    wait for 10 ns;
```

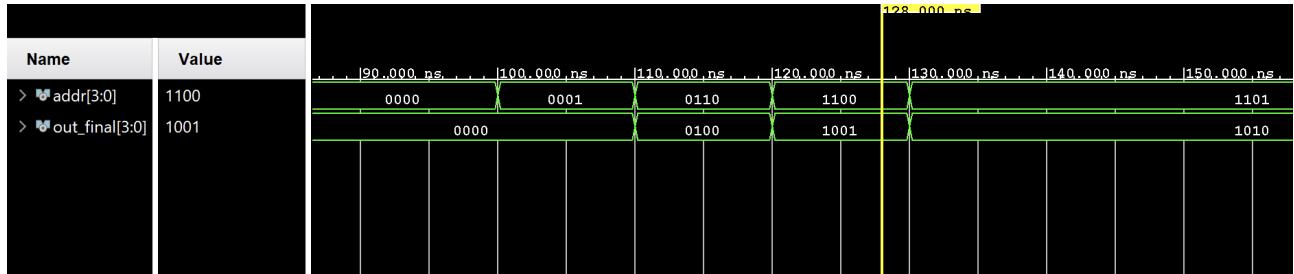
```

wait;
end process;

end tb;

```

Dall'analisi dei risultati della simulazione, si può verificare che il sistema si comporta come previsto. In particolare, osservando l'indirizzo di ingresso 1100, l'uscita restituisce i bit 1001, che corrispondono ai MSB dell'indirizzo 10010000.



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

Di seguito viene riportato direttamente il file di constraint (visto che non sono state necessarie modifiche ulteriori), con i valori degli switch e dei led opportunamente settati:

```

##Switches
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMS3 } [get_ports { addr[0] }]; #IO_L24N_T3_RS0_15
Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMS3 } [get_ports { addr[1] }];
#IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMS3 } [get_ports { addr[2] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMS3 } [get_ports { addr[3] }]; #IO_L13N_T2_MRCC_14
Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMS3 } [get_ports { X[4] }]; #IO_L12N_T1_MRCC_14
Sch=sw[4]
#set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMS3 } [get_ports { X[5] }]; #IO_L7N_T1_D10_14
Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMS3 } [get_ports { X[6] }]; #IO_L17N_T2_A13_D29_14
Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMS3 } [get_ports { X[7] }]; #IO_L5N_T0_D07_14
Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMS18 } [get_ports { X[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMS18 } [get_ports { X[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMS3 } [get_ports { SW[10] }];
#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMS3 } [get_ports { SW[11] }];

```

```

#IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35
Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { SW[13] }];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { SW[14] }];
#IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14
Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { out_final[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { out_final[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { out_final[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { out_final[3] }]; #IO_L8P_T1_D11_14
Sch=led[3]
#set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14
Sch=led[4]
#set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { LED[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { LED[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
#set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { LED[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { LED[8] }];
#IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14
Sch=led[9]
#set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { LED[10] }];
#IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { LED[11] }];
#IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
#set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { LED[12] }]; #IO_L16P_T2_CSI_B_14
Sch=led[12]
#set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { LED[13] }];
#IO_L22N_T3_A04_D20_14 Sch=led[13]
#set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { LED[14] }];
#IO_L20N_T3_A07_D23_14 Sch=led[14]
#set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { LED[15] }];
#IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

```

Capitolo 2: Reti Sequentziali 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 che rappresenta il dato, un segnale A di tempificazione 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

Dal punto di vista progettuale, il riconoscitore è stato progettato con un'architettura composta da due parti fondamentali:

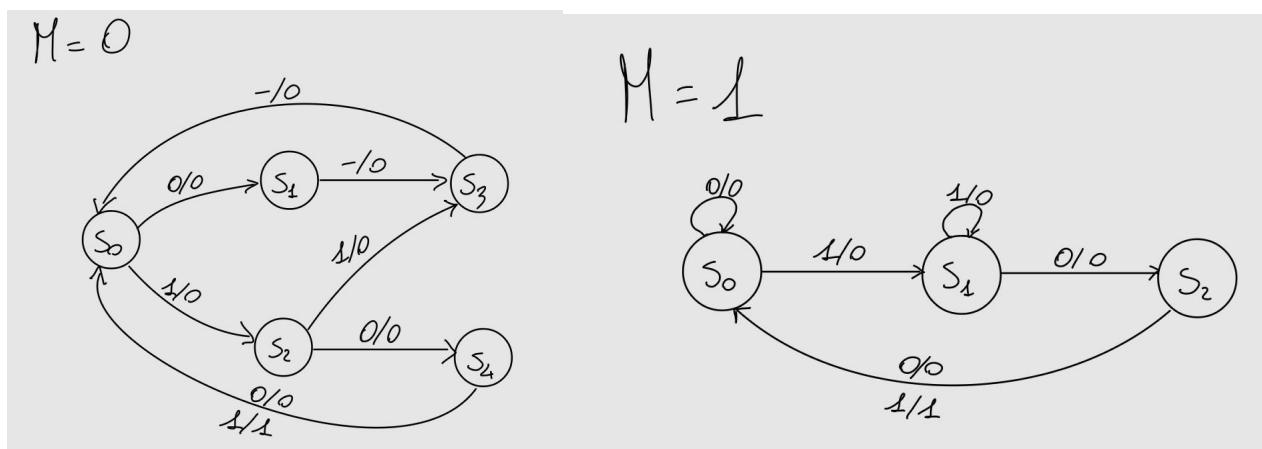
1. Parte Sequentiale:

Questa componente è responsabile del mantenimento dello stato corrente, che viene aggiornato sul fronte di salita del segnale di tempificazione (A). In caso di segnale di reset (RST), la macchina torna allo stato iniziale S0.

2. Parte Combinatoria:

Questa componente determina lo stato successivo e produce l'uscita in base allo stato corrente, all'input binario e alla modalità di funzionamento. La logica combinatoria implementa le regole di transizione tra gli stati e l'emissione del segnale di riconoscimento della sequenza.

Di seguito viene riportato il disegno degli automi nelle due configurazioni richieste dalla traccia:



La macchina funziona come segue:

1. **Modalità non sovrapposta (M=0):** La macchina riconosce solo sequenze complete e non sovrapposte, valutando i bit in ingresso a gruppi di 3. Ad esempio, in una stringa di input 010101100, riconoscerà solo la sequenza evidenziata.
2. **Modalità parzialmente sovrapposta (M=1):** Consente di riconoscere sequenze parzialmente sovrapposte, valutando i bit in ingresso singolarmente. Ad esempio, con input 0101001, la macchina riconoscerà la sequenza evidenziata.

Implementazione

Di seguito è riportato il codice VHDL:

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

entity riconoscitore_101 is
    port(
        i: in STD_LOGIC;
        A: in STD_LOGIC; -- Segnale di temporizzazione
        -- CLK: in STD_LOGIC;
        M: in STD_LOGIC;
        RST: in std_logic;
        Y: out STD_LOGIC
    );
end riconoscitore_101;
```

```
architecture Behavioral of riconoscitore_101 is
```

```
    type stato is (S0, S1, S2, S3, S4);
    signal stato_corrente : stato:=S0;
    signal stato_prossimo : stato;
```

```
-- PARTE COMBINATORIA:
```

```
begin
```

```
    stato_uscita: process(stato_corrente, i, M) begin
        Y <= '0'; -- Default output
        if(M = '1') then
            case stato_corrente is
                when S0 =>
                    if (i='1') then
                        stato_prossimo <= S1;
                        Y <= '0';
                    else
                        stato_prossimo <= S0;
                        Y <= '0';
                    end if;
                when S1 =>
                    if( i = '0') then
                        stato_prossimo <= S2;
                        Y <= '0';
                    end if;
                when S2 =>
                    if( i = '0') then
                        stato_prossimo <= S3;
                        Y <= '0';
                    end if;
                when S3 =>
                    if( i = '0') then
                        stato_prossimo <= S4;
                        Y <= '0';
                    end if;
                when S4 =>
                    if( i = '0') then
                        stato_prossimo <= S0;
                        Y <= '0';
                    end if;
            end case;
        end if;
    end process;
    Y <= stato_prossimo;
end begin;
```

```

else
    stato_prossimo <= S1;
    Y <= '0';
end if;
when S2 =>
    if (i = '1') then
        stato_prossimo <= S0;
        Y <= '1';
    else
        stato_prossimo <= S0;
        Y <= '0';
    end if;
when others => stato_prossimo <= S0;
end case;
elsif(M = '0') then
    case stato_corrente is
        when S0 =>
            if (i='1') then
                stato_prossimo <= S2;
                Y <= '0';
            else
                stato_prossimo <= S1;
                Y <= '0';
            end if;
        when S1 =>
            stato_prossimo <= S3;
            Y <= '0';
        when S2 =>
            if (i='0') then
                stato_prossimo <= S4;
                Y <= '0';
            else
                stato_prossimo <= S3;
                Y <= '0';
            end if;
        when S3 =>
            stato_prossimo <= S0;
            Y <= '0';
        when S4 =>
            if (i='1') then
                stato_prossimo <= S0;
                Y <= '1';
            else
                stato_prossimo <= S0;
                Y <= '0';
            end if;
        end case;
    end if;
end process;

```

-- PARTE SEQUENZIALE -> stato corrente aggiornato solo sul fronte di salita del clock:
mem: **process**(A)

```

begin
  if( A'event and A = '1' ) then
    if ( RST = '1' ) then
      stato_corrente <= S0;
    else
      stato_corrente <= stato_prossimo;
    end if;
  end if;
end process;
end Behavioral;

```

Simulazione

Il seguente testbench simula il comportamento del riconoscitore di sequenza 101. Al suo interno viene generato un segnale periodico (A) che simula il clock, necessario per la sincronizzazione della macchina a stati.

Nella fase di test vero e proprio, vengono applicati diversi valori all'ingresso i, rappresentando una sequenza di bit che il riconoscitore deve analizzare. Inizialmente, il test viene eseguito con M = 1, modalità in cui il riconoscitore valuta la sequenza in modo parzialmente sovrapposto, ripartendo ogni volta che rileva 101. Dopo un breve intervallo, la modalità viene cambiata in M = 0, e il test prosegue verificando il comportamento in cui le sequenze vengono analizzate senza sovrapposizione.

L'obiettivo è osservare se il segnale di uscita Y si attiva nei momenti previsti, confermando che il riconoscitore passa correttamente tra gli stati e identifica la sequenza desiderata nelle due modalità operative.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use std.textio.ALL;

entity tb_riconoscitore_101 is
end tb_riconoscitore_101;

architecture testbench of tb_riconoscitore_101 is

component riconoscitore_101
port(
  i: in STD_LOGIC;
  A: in STD_LOGIC;
  M: in STD_LOGIC;
  RST: in std_logic;
  Y: out STD_LOGIC
);
end component;

--inputs
signal i, A, M, RST: STD_LOGIC := '0';

--outputs

```

```

signal Y: STD_LOGIC;
--signal A period definition
constant A_period : time := 10 ns;

begin
    -- Istanziazione del DUT (Device Under Test)
    uut: riconoscitore_101 port map (
        i => i,
        A => A,
        M => M,
        RST => RST,
        Y => Y
    );
    -- A process definitions
    A_process :process
        begin
            A <= '0';
            wait for A_period/2;
            A <= '1';
            wait for A_period/2;
        end process;
    stim_proc: process
        begin
            -- insert stimulus here
            -- Modalità parzialmente sovrapposta
            M<='1';
            i<='0';
            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;
            i<='0';
            wait for 10 ns;
            i<='1';
            wait for 10 ns;
            i<='0';
            wait for 30 ns;
            -- Modalità non sovrapposta
            M<='0';
            i<='0';
            wait for 10 ns;

```

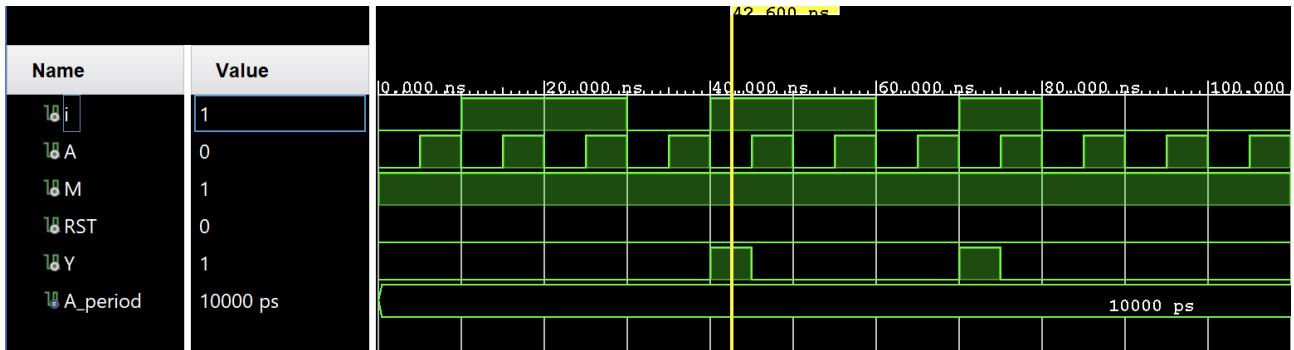
```

i<='1';
wait for 10 ns;
i<='0';
wait for 10 ns;
i<='1';
wait for 10 ns;
i<='0';
wait for 10 ns;
i<='1';
wait for 10 ns;
i<='0';
wait for 10 ns;
i<='1';
wait for 30 ns;

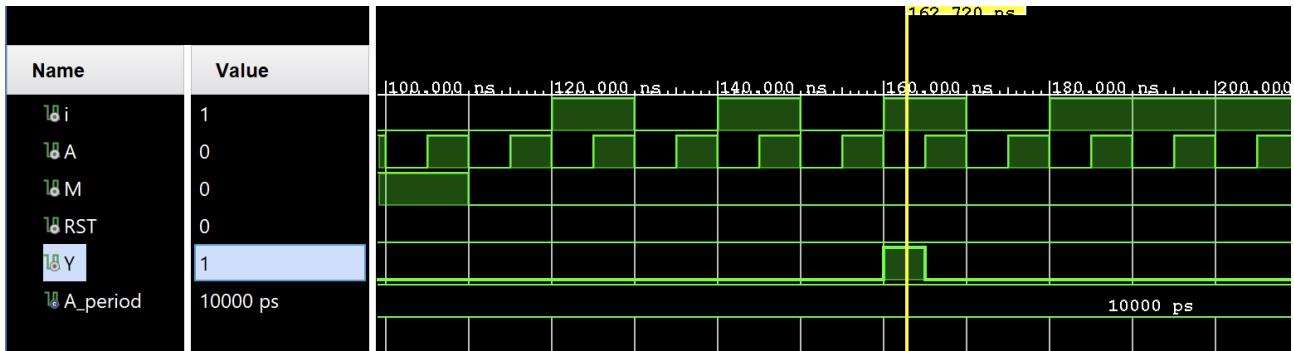
wait;
end process;
end testbench;

```

- Risultato della simulazione nella modalità parzialmente sovrapposta ($M = 1$)



- Risultato della simulazione nella modalità non sovrapposta ($M = 0$)



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

Per implementare su board la rete sviluppata al punto precedente, è stata progettata un'unità di controllo per la gestione dell'input e del segnale di modo M, tramite due appositi switch disponibili sulla scheda.

Di seguito viene riportato il codice VHDL dell'unità di controllo, del top level e del file di constraint:

- Top Level

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

entity top_level_ric is
  Port (
    i: in STD_LOGIC;
    sel_i : in STD_LOGIC;
    A: in STD_LOGIC; -- Segnale di temporizzazione
    M: in STD_LOGIC;
    sel_m : in STD_LOGIC;
    RST: in std_logic;
    Y: out STD_LOGIC
  );
end top_level_ric;

architecture structural of top_level_ric is
  signal i_out_tmp : std_logic := '0';
  signal m_out_tmp : std_logic := '0';

component riconoscitore_101
  port (
    i : in STD_LOGIC;
    A : in STD_LOGIC; -- Segnale di temporizzazione
    M : in STD_LOGIC;
    RST : in std_logic;
    Y : out STD_LOGIC
  );
end component;

component control_unit
  port (
    i : in STD_LOGIC;
    sel_i : in STD_LOGIC;
    A : in STD_LOGIC;
    M : in STD_LOGIC;
    sel_m : in STD_LOGIC;
```

```

    i_out : out STD_LOGIC;
    M_out : out STD_LOGIC
);
end component;

begin

ric : riconoscitore_101 port map(
    i => i_out_tmp,
    A => A,
    M => m_out_tmp,
    RST => RST,
    Y => Y
);

u_c : control_unit port map(
    i => i,
    sel_i => sel_i,
    A => A,
    M => M,
    sel_m => sel_M,
    i_out => i_out_tmp,
    M_out => m_out_tmp
);
end structural;

```

- Unità di controllo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity control_unit is
Port (
    i: in STD_LOGIC;
    sel_i : in STD_LOGIC;
    A: in STD_LOGIC;
    M: in STD_LOGIC;
    sel_m : in STD_LOGIC;
    i_out : out STD_LOGIC;
    M_out : out STD_LOGIC
);
end control_unit;

architecture Behavioral of control_unit is

signal reg_i, reg_m : std_logic;

begin
process (A) begin

```

```

if (A'event and A='1') then
    if (sel_i = '1') then
        reg_i <= i;
    elsif (sel_M = '1') then
        reg_M <= M;
    end if;
end if;
end process;

i_out <= reg_i;
M_out <= reg_m;

end Behavioral;

```

- Constraint

```

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

##Switches
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { i }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { M }]; #IO_L3N_T0_DQS_EMCCCLK_14
Sch=sw[1]
#set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { X[2] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
#set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { X[3] }]; #IO_L13N_T2_MRCC_14
Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { X[4] }]; #IO_L12N_T1_MRCC_14
Sch=sw[4]
#set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { X[5] }]; #IO_L7N_T1_D10_14
Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { X[6] }]; #IO_L17N_T2_A13_D29_14
Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { X[7] }]; #IO_L5N_T0_D07_14
Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { X[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { X[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { SW[10] }];
#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { SW[11] }];
#IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35
Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { SW[13] }];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { SW[14] }];
#IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14
Sch=sw[15]

```

```

## LEDs
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { Y }]; #IO_L18P_T2_A24_15
Sch=led[0]
#set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { Y[1] }]; #IO_L24P_T3_RS1_15
Sch=led[1]
#set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { Y[2] }]; #IO_L17N_T2_A25_15
Sch=led[2]
#set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { Y[3] }]; #IO_L8P_T1_D11_14
Sch=led[3]
#set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14
Sch=led[4]
#set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { LED[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { LED[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
#set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { LED[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { LED[8] }];
#IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14
Sch=led[9]
#set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { LED[10] }];
#IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { LED[11] }];
#IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
#set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { LED[12] }]; #IO_L16P_T2_CSI_B_14
Sch=led[12]
#set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { LED[13] }];
#IO_L22N_T3_A04_D20_14 Sch=led[13]
#set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { LED[14] }];
#IO_L20N_T3_A07_D23_14 Sch=led[14]
#set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { LED[15] }];
#IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

##Buttons
#set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports { RST }]; #IO_L9P_T1_DQS_14
Sch=btnc
#set_property -dict { PACKAGE_PIN M18 IOSTANDARD LVCMOS33 } [get_ports { BTNU }]; #IO_L4N_T0_D05_14
Sch=btnu
set_property -dict { PACKAGE_PIN P17 IOSTANDARD LVCMOS33 } [get_ports { sel_m }]; #IO_L12P_T1_MRCC_14
Sch=btln
set_property -dict { PACKAGE_PIN M17 IOSTANDARD LVCMOS33 } [get_ports { sel_i }]; #IO_L10N_T1_D15_14
Sch=btnr
#set_property -dict { PACKAGE_PIN P18 IOSTANDARD LVCMOS33 } [get_ports { BTND }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd

```

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.

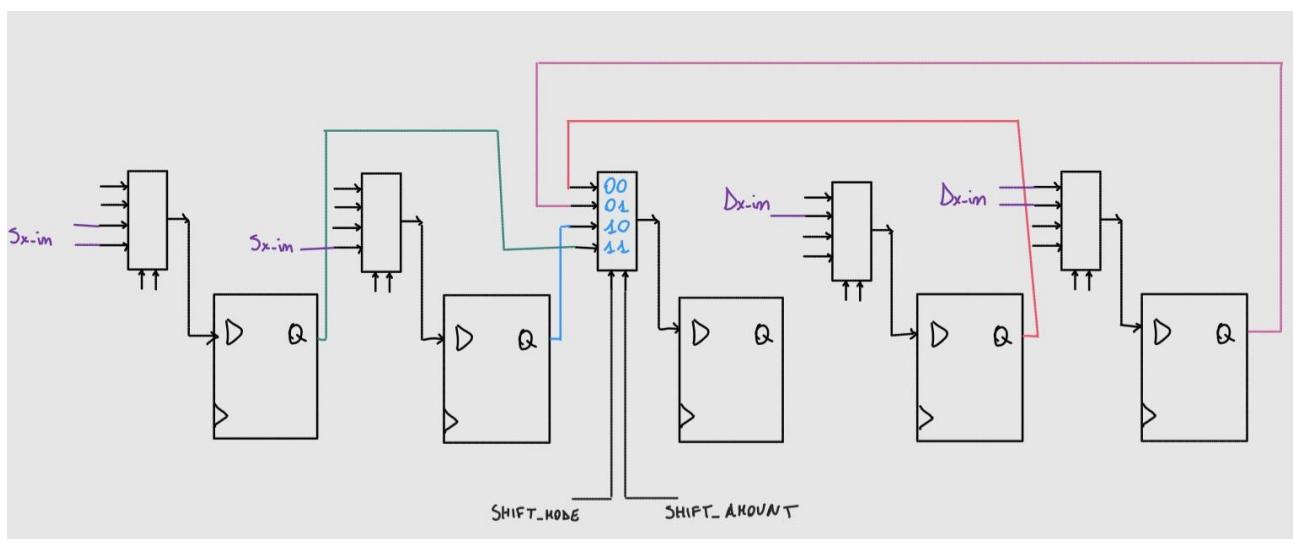
Nota: il numero di bit del registro deve essere implementato come un generic, e dall'esterno deve poter essere scelta la modalità di funzionamento mediante opportuni segnali di selezione.

Progetto e architettura

Come espresso dai requisiti progettuali, l'architettura del registro a scorrimento è stata definita in 2 modi: comportamentale e strutturale. In entrambi i casi si è tenuto traccia degli N bit del registro usando una variabile '**generic**'. Inoltre, si sono utilizzate 2 variabili per, rispettivamente, indicare di quante posizioni shiftare e soprattutto in quale direzione (in quest'ultimo caso, con lo '0' si indica lo shift a sinistra, mentre con '1' lo shift a destra).

Architetturalmente, invece, il registro è stato progettato in base ad una logica '**SIFO**' (*Serial Input - Parallel Output*), ovvero con il segnale in ingresso fornito serialmente, e l'output che invece viene generato in parallelo.

Di seguito viene riportato il disegno dello shift register, in grado di shiftare in entrambe le direzioni (implementato con, ad esempio, N = 5 bit):



Per mantenere il disegno più chiaro e leggibile, viene mostrato un esempio di collegamento relativo al solo registro centrale ed i casi particolari di ingresso ai registri 0,1, N-2, N-1, mentre sono stati omessi il segnale di clock e le uscite parallele.

Implementazione

Viene qui riportato il codice VHDL prima utilizzando un approccio comportamentale, e poi strutturale (il codice del multiplexer 4:1 è riportato in appendice):

a) Approccio comportamentale

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

entity shift_register_behavioral is
    -- Numero di bit del registro, 8 di def.
    generic (
        N: integer := 8
    );
    port(
        SI: in STD_LOGIC;
        RST: in STD_LOGIC;
        CLK: in STD_LOGIC;
        SHIFT_AMOUNT: in integer range 0 to 1; -- Y = 1 o 2 posizioni shiftabili;
        SHIFT_MODE: in STD_LOGIC; -- '0' shift a destra, e '1' shift a sinistra
        SO: out STD_LOGIC_VECTOR(N-1 downto 0)
    );
end shift_register_behavioral;

architecture behavioral of shift_register_behavioral is
    signal tmp_reg: STD_LOGIC_VECTOR(N-1 downto 0) := (others => '0');
begin
    process(CLK)
    begin
        if(CLK'event and CLK='1') then
            if(RST='1') then
                tmp_reg <= (others => '0');
            else
                if SHIFT_MODE = '1' then -- Shift a destra
                    if SHIFT_AMOUNT = 1 then
                        tmp_reg <= SI & tmp_reg(N-1 downto 1); -- Shift di 1 posizione
                    elsif SHIFT_AMOUNT = 2 then
                        tmp_reg <= SI & SI & tmp_reg(N-1 downto 2); -- Shift di 2 posizioni
                    end if;
                else -- Shift a sinistra
                    if SHIFT_AMOUNT = 1 then
                        tmp_reg <= tmp_reg(N-2 downto 0) & SI; -- Shift di 1 posizione
                    elsif SHIFT_AMOUNT = 2 then
                        tmp_reg <= tmp_reg(N-3 downto 0) & SI & SI; -- Shift di 2 posizioni
                    end if;
                end if;
            end if;
        end if;
    end process;
    SO <= tmp_reg;
end behavioral;
```

b) Approccio strutturale

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

entity shift_register_structural is
Generic (N : integer := 5);

Port (
    Sx_in : in STD_LOGIC;
    Dx_in : in STD_LOGIC;
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    SHIFT_MODE : in STD_LOGIC;
    SHIFT_AMOUNT : in STD_LOGIC;
    Q : out STD_LOGIC_VECTOR(N-1 downto 0)

);
end shift_register_structural;

architecture structural of shift_register_structural is
signal Q_temp : STD_LOGIC_VECTOR(N-1 downto 0) := (others => '0');
signal D_temp : STD_LOGIC_VECTOR(N-1 downto 0) := (others => '0');

component FF
port (
    D : in STD_LOGIC;
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    Q : out STD_LOGIC
);
end component;

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;
        s1 : in STD_LOGIC;
        y : out STD_LOGIC
);
end component;

begin
    -- Generazione registri
    gen_reg : for i in 0 to N-1 generate
```

```

regi : FF port map(
    D => D_temp(i),
    CLK => CLK,
    RST => RST,
    Q => Q_temp(i)
);
end generate;

-- Generazione multiplexer
MUXN_1 : mux_4_1 port map(
    a0 => Q_temp(N-2),
    a1 => Q_temp(N-3),
    a2 => Sx_in,
    a3 => Sx_in,
    s0 => SHIFT_AMOUNT,
    s1 => SHIFT_MODE,
    y => D_temp(N-1)
);
MUXN_2 : mux_4_1 port map(
    a0 => Q_temp(N-3),
    a1 => Q_temp(N-4),
    a2 => Q_temp(N-1),
    a3 => Sx_in,
    s0 => SHIFT_AMOUNT,
    s1 => SHIFT_MODE,
    y => D_temp(N-2)
);

gen_mux : for index in N-3 downto 2 generate
    MUXi : mux_4_1 port map(
        a0 => Q_temp(index-1),
        a1 => Q_temp(index-2),
        a2 => Q_temp(index+1),
        a3 => Q_temp(index+2),
        s0 => SHIFT_AMOUNT,
        s1 => SHIFT_MODE,
        y => D_temp(index)
    );
end generate;

MUX1 : mux_4_1 port map(
    a0 => Q_temp(0),
    a1 => Dx_in,
    a2 => Q_temp(2),
    a3 => Q_temp(3),
    s0 => SHIFT_AMOUNT,
    s1 => SHIFT_MODE,
    y => D_temp(1)
);

MUX0 : mux_4_1 port map(

```

```

    a0 => Dx_in,
    a1 => Dx_in,
    a2 => Q_temp(1),
    a3 => Q_temp(2),
    s0 => SHIFT_AMOUNT,
    s1 => SHIFT_MODE,
    y => D_temp(0)
);
Q <= Q_temp;
end structural;

```

- FF

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FF is
  Port (
    D : in STD_LOGIC;
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    Q : out STD_LOGIC
  );
end FF;

architecture Behavioral of FF is
  signal Q_reg : STD_LOGIC := '0';
begin
  process (CLK)
  begin
    if (CLK'event and CLK='1') then
      if (RST = '1') then
        Q_reg <= '0';
      else
        Q_reg <= D;
      end if;
    end if;
  end process;
  Q <= Q_reg;
end Behavioral;

```

Simulazione

Per simulare il comportamento del registro a scorrimento, sia nel caso del componente realizzato con approccio comportamentale che strutturale, sono stati prodotti i seguenti testbench, nei quali, dopo un reset iniziale, vengono verificate le diverse modalità di shift:

- Testbench Shift Register Behavioral

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

entity tb_shift_register is
end tb_shift_register;

architecture testbench of tb_shift_register is
constant N : integer := 8;
signal SI      : STD_LOGIC := '0';
signal RST     : STD_LOGIC := '0';
signal CLK     : STD_LOGIC := '0';
signal SHIFT_AMOUNT : integer range 0 to 1 := 0;
signal SHIFT_MODE : STD_LOGIC := '0';
signal SO      : STD_LOGIC_VECTOR(N-1 downto 0);

constant clk_period : time := 20 ns;

component shift_register_behavioral
generic (N : integer := 8);
port (
    SI      : in STD_LOGIC;
    RST     : in STD_LOGIC;
    CLK     : in STD_LOGIC;
    SHIFT_AMOUNT : in integer range 0 to 1;
    SHIFT_MODE : in STD_LOGIC;
    SO      : out STD_LOGIC_VECTOR(N-1 downto 0)
);
end component;

begin
clk_process : process
begin
    CLK <= '0';
    wait for clk_period/2;
    CLK <= '1';
    wait for clk_period/2;
end process;

uut: shift_register_behavioral
generic map (N => N)
port map (
    SI => SI,
    RST => RST,
    CLK => CLK,
```

```

SHIFT_AMOUNT => SHIFT_AMOUNT,
SHIFT_MODE => SHIFT_MODE,
SO => SO
);

process
begin
-- Reset iniziale
RST <= '1';
wait for 10 ns;
RST <= '0';
wait for 10 ns;

-- Shift a sinistra di 1 posizione
SHIFT_MODE <= '1';
SHIFT_AMOUNT <= 1;
SI <= '1';
wait for 20 ns;

-- Shift a destra di 1 posizione
SHIFT_MODE <= '0';
SHIFT_AMOUNT <= 1;
SI <= '1';
wait for 20 ns;

-- Shift a destra di 2 posizioni
SHIFT_MODE <= '0';
SHIFT_AMOUNT <= 2;
SI <= '1';
wait for 20 ns;

-- Shift a sinistra di 2 posizioni
SHIFT_MODE <= '1';
SHIFT_AMOUNT <= 2;
SI <= '1';
wait for 20 ns;

wait;
end process;
end testbench;

```

- Testbench Shift Register Structural

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity shift_register_tb is
end shift_register_tb;

```

```

architecture test of shift_register_tb is

    constant N : integer := 5;
    signal Sx_in, Dx_in, CLK, RST, SHIFT_MODE, SHIFT_AMOUNT : STD_LOGIC;
    signal Q : STD_LOGIC_VECTOR(N-1 downto 0);

    component shift_register_structural
        Generic (N : integer := 5);
        Port (
            Sx_in : in STD_LOGIC;
            Dx_in : in STD_LOGIC;
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            SHIFT_MODE : in STD_LOGIC;
            SHIFT_AMOUNT : in STD_LOGIC;
            Q : out STD_LOGIC_VECTOR(N-1 downto 0)
        );
    end component;

begin

    uut: shift_register_structural
        generic map (N => N)
        port map (
            Sx_in => Sx_in,
            Dx_in => Dx_in,
            CLK => CLK,
            RST => RST,
            SHIFT_MODE => SHIFT_MODE,
            SHIFT_AMOUNT => SHIFT_AMOUNT,
            Q => Q
        );

    -- Clock process
    process
    begin
        CLK <= '0';
        wait for 10 ns;
        CLK <= '1';
        wait for 10 ns;
    end process;

    -- Stimulus process
    process
    begin
        -- Reset
        RST <= '1';
        Sx_in <= '0';
        Dx_in <= '0';
        SHIFT_MODE <= '0';
        SHIFT_AMOUNT <= '0';
    
```

```

wait for 20 ns;
RST <= '0';
wait for 20 ns;

-- Shift Right by 1
Sx_in <= '1';
SHIFT_MODE <= '1';
SHIFT_AMOUNT <= '0';
wait for 20 ns;

-- Shift Right by 2
Sx_in <= '0';
SHIFT_AMOUNT <= '1';
wait for 20 ns;

-- Shift Left by 1
Dx_in <= '1';
SHIFT_MODE <= '0';
SHIFT_AMOUNT <= '0';
wait for 20 ns;

-- Shift Left by 2
Dx_in <= '0';
SHIFT_AMOUNT <= '1';
wait for 20 ns;

-- Stop Simulation
wait;
end process;

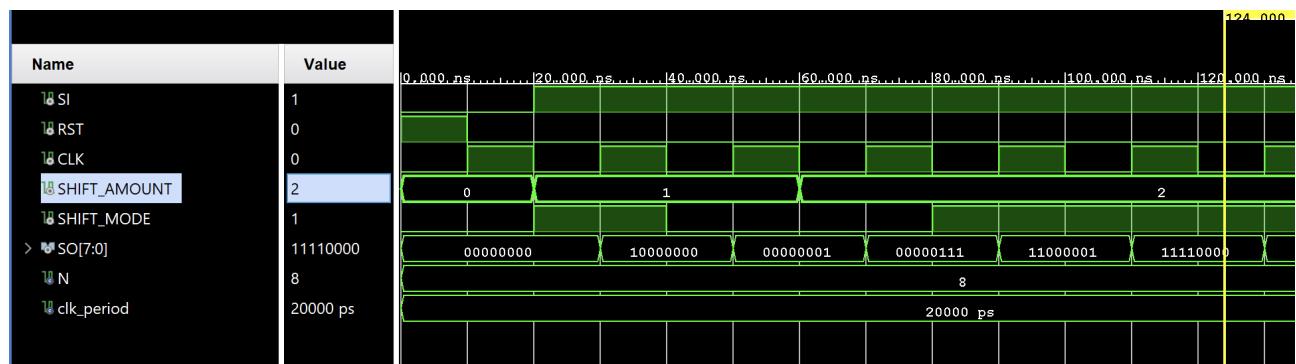
end test;

```

Come si evince dai risultati delle simulazioni, il sistema si comporta come previsto; in particolare:

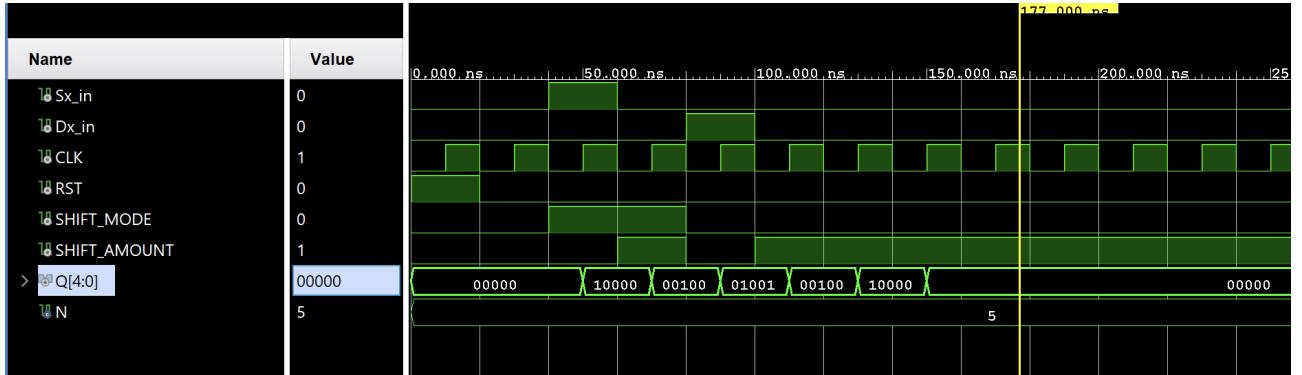
A. Approccio Comportamentale

1. Shift a destra di 1 posizione
2. Shift a sinistra di 1 posizione
3. Shift a sinistra di 2 posizioni
4. Shift a destra di 2 posizioni



B. Approccio Strutturale

5. Shift a destra di 1 posizione, Sx_in = 1
6. Shift a destra di 2 posizioni, Sx_in = 0
7. Shift a sinistra di 1 posizione, Dx_in = 1
8. Shift a sinistra di 2 posizioni, Dx_in = 0



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.

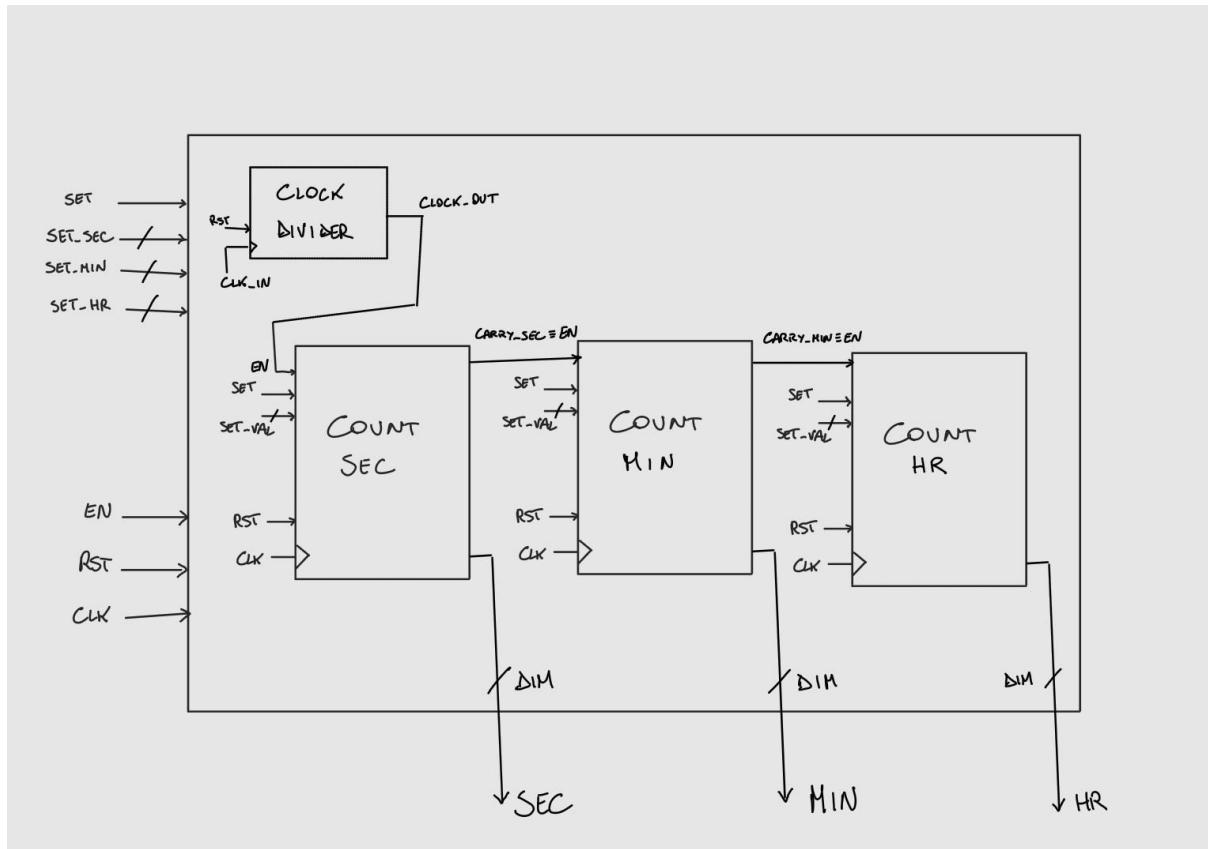
Progetto e architettura

Il cronometro in questione è progettato per scandire il tempo in secondi, minuti e ore a partire da un clock di base, con:

- **Impostazione iniziale:** È possibile inizializzare il cronometro con ore, minuti e secondi tramite segnali di set.
- **Reset:** Un segnale dedicato permette di riportare il cronometro a 0.
- **Struttura modulare:** Implementato in modo strutturale, utilizzando il componente contatore (il cui codice è riportato in appendice) per contare secondi, minuti e ore.

Si nota poi che è stato aggiunto un **Clock Divider** (per permettere un funzionamento corretto sulla board), il cui scopo è quello di abilitare il contatore dei secondi ogni secondo (visto che il clock della board è a 100MHz).

Di seguito è riportato il disegno dell'architettura del cronometro:



Implementazione

Di seguito viene riportato il codice VHDL per l'implementazione del **cronometro** (il codice del componente contatore è riportato in appendice):

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

entity cronometro is
    Port (
        CLK : in STD_LOGIC;
        RST : in STD_LOGIC;
        SET : in STD_LOGIC;
        SET_SEC : in STD_LOGIC_VECTOR(5 downto 0);
        SET_MIN : in STD_LOGIC_VECTOR(5 downto 0);
        SET_HR : in STD_LOGIC_VECTOR(4 downto 0);
        SEC : out STD_LOGIC_VECTOR(5 downto 0);
        MIN : out STD_LOGIC_VECTOR(5 downto 0);
        HR : out STD_LOGIC_VECTOR(4 downto 0)
    );
end cronometro;

architecture Structural of cronometro is
    signal carry_sec, carry_min, enable_sec : STD_LOGIC;

    component counter
        Generic(MAX_VALUE : integer; DIM : integer);
        Port(
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            EN : in STD_LOGIC;
            SET : in STD_LOGIC;
            SET_VALUE : in STD_LOGIC_VECTOR(DIM-1 downto 0);
            COUNT : out STD_LOGIC_VECTOR(DIM-1 downto 0);
            CARRY : out STD_LOGIC
        );
    end component;

    component clock_divider
        Generic(
            clock_frequency_in : integer;
            clock_frequency_out : integer
        );
        Port (
            clock_in : in STD_LOGIC;
            reset : in STD_LOGIC;
            clock_out : out STD_LOGIC
        );
    end component;
begin
    --Divisore di frequenza, per permettere al contatore dei secondi di effettuare un conteggio al secondo
    --La frequenza in uscita viene lasciata inalterata per permettere la visualizzazione del risultato in simulazione

```

```

div : clock_divider
  Generic Map(clock_frequency_in => 1000000000, clock_frequency_out => 1000000000)
  Port Map(
    clock_in => CLK,
    reset => RST,
    clock_out => enable_sec
  );

--Contatore secondi
count_sec : counter
  Generic Map(MAX_VALUE => 59, DIM => 6)
  Port Map(
    CLK => CLK,
    RST => RST,
    EN => enable_sec,
    SET => SET,
    SET_VALUE => SET_SEC,
    COUNT => SEC,
    CARRY => carry_sec
  );

--Contatore minuti
count_min : counter
  Generic Map(MAX_VALUE => 59, DIM => 6)
  Port Map(
    CLK => CLK,
    RST => RST,
    EN => carry_sec,
    SET => SET,
    SET_VALUE => SET_MIN,
    COUNT => MIN,
    CARRY => carry_min
  );

--Contatore ore
count_hr : counter
  Generic Map(MAX_VALUE => 23, DIM => 5)
  Port Map(
    CLK => CLK,
    RST => RST,
    EN => carry_min,
    SET => SET,
    SET_VALUE => SET_HR,
    COUNT => HR,
    CARRY => open
  );

end Structural;

```

- Clock Divider

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- il clock_divider genera un impulso a frequenza inferiore rispetto a quella del clock

entity clock_divider is
  generic(
    clock_frequency_in : integer := 100000000;--100MHz
    clock_frequency_out : integer := 500 --500 Hz
  );
  Port ( clock_in : in STD_LOGIC;
         reset : in STD_LOGIC;
         clock_out : out STD_LOGIC);
end clock_divider;

architecture Behavioral of clock_divider is

signal clockfx: std_logic := '0';

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

begin

clock_out <= clockfx;

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

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

end process;

end Behavioral;

```

Simulazione

Il testbench seguente simula il funzionamento del cronometro generando un segnale di clock e applicando diversi stimoli. Inizialmente, viene eseguito un reset, poi si imposta l'orario iniziale a 12:30:45 attivando il segnale SET. Successivamente, si lascia che il cronometro conti per un certo periodo di tempo. L'obiettivo è osservare il comportamento del modulo e verificare che i contatori di secondi, minuti e ore funzionino correttamente.

Di seguito è riportato il codice:

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

ENTITY tb_cronometro IS
END tb_cronometro;

ARCHITECTURE testbench OF tb_cronometro IS

-- Component declaration for the Unit Under Test (UUT)
COMPONENT cronometro
PORT(
    CLK : IN STD_LOGIC;
    RST : IN STD_LOGIC;
    SET : IN STD_LOGIC;
    SET_SEC : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
    SET_MIN : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
    SET_HR : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
    SEC : OUT STD_LOGIC_VECTOR(5 DOWNTO 0);
    MIN : OUT STD_LOGIC_VECTOR(5 DOWNTO 0);
    HR : OUT STD_LOGIC_VECTOR(4 DOWNTO 0)
);
END COMPONENT;

-- Signals
SIGNAL CLK : STD_LOGIC := '0';
SIGNAL RST : STD_LOGIC := '0';
SIGNAL SET : STD_LOGIC := '0';
SIGNAL SET_SEC : STD_LOGIC_VECTOR(5 DOWNTO 0) := (OTHERS => '0');
SIGNAL SET_MIN : STD_LOGIC_VECTOR(5 DOWNTO 0) := (OTHERS => '0');
SIGNAL SET_HR : STD_LOGIC_VECTOR(4 DOWNTO 0) := (OTHERS => '0');
SIGNAL SEC : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL MIN : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL HR : STD_LOGIC_VECTOR(4 DOWNTO 0);

-- Clock period definition
CONSTANT CLK_PERIOD : TIME := 10 ns;

BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut: cronometro PORT MAP (
        CLK => CLK,
```

```

RST => RST,
SET => SET,
SET_SEC => SET_SEC,
SET_MIN => SET_MIN,
SET_HR => SET_HR,
SEC => SEC,
MIN => MIN,
HR => HR
);

-- Clock process definitions
CLK_process : PROCESS
BEGIN
  WHILE TRUE LOOP
    CLK <= '0';
    WAIT FOR CLK_PERIOD/2;
    CLK <= '1';
    WAIT FOR CLK_PERIOD/2;
  END LOOP;
END PROCESS;

-- Stimulus process
stim_proc: PROCESS
BEGIN
  -- Reset the system
  RST <= '1';
  WAIT FOR 20 ns;
  RST <= '0';
  WAIT FOR 100 ns;

  -- Set the initial time
  SET <= '1';
  SET_SEC <= "101101"; -- 45 seconds
  SET_MIN <= "011110"; -- 30 minutes
  SET_HR <= "01100"; -- 12 hours
  WAIT FOR 20 ns;
  SET <= '0';

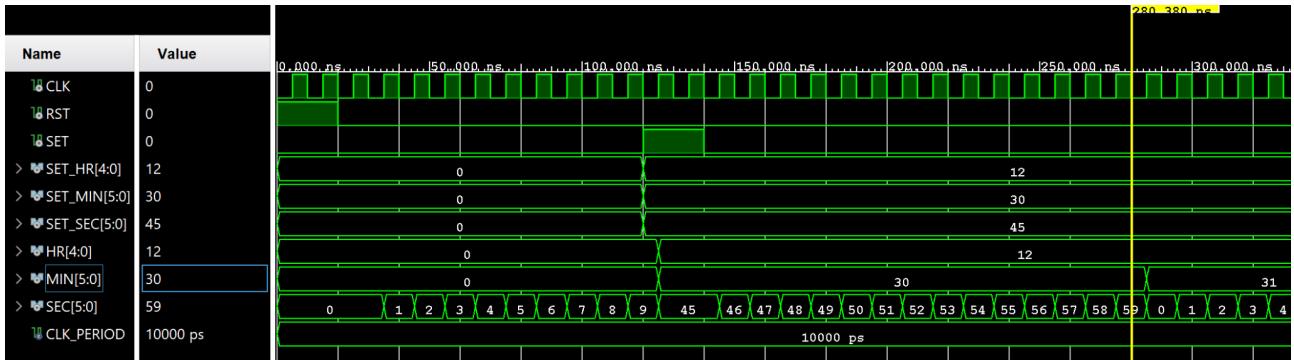
  -- Run simulation for some time
  WAIT FOR 500 ns;

  -- End simulation
  WAIT;
END PROCESS;
END testbench;

```

Dai risultati della simulazione, di seguito riportata, si può vedere che il sistema rispetta le specifiche richieste. Dopo il reset iniziale, il cronometro assume correttamente il valore impostato di 12:30:45. Possiamo osservare che i secondi iniziano a contare fino a 59, per poi azzerarsi e incrementare il valore dei minuti, che passa da 30 a 31. Questo conferma che il carry dei secondi è gestito correttamente. Le ore

rimangono stabili a 12, come previsto in questa fase della simulazione. Il comportamento osservato dimostra che il cronometro segue la logica di conteggio attesa.



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

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

Il sistema è progettato per leggere dati da una memoria ROM, elaborarli tramite una macchina combinatoria M, e memorizzare i risultati in una memoria MEM. L'intero processo viene coordinato da un'unità di controllo, che lo gestisce in modo sequenziale tramite un contatore.

L'esecuzione inizia quando il segnale di START viene attivato. A ogni ciclo di clock, il sistema:

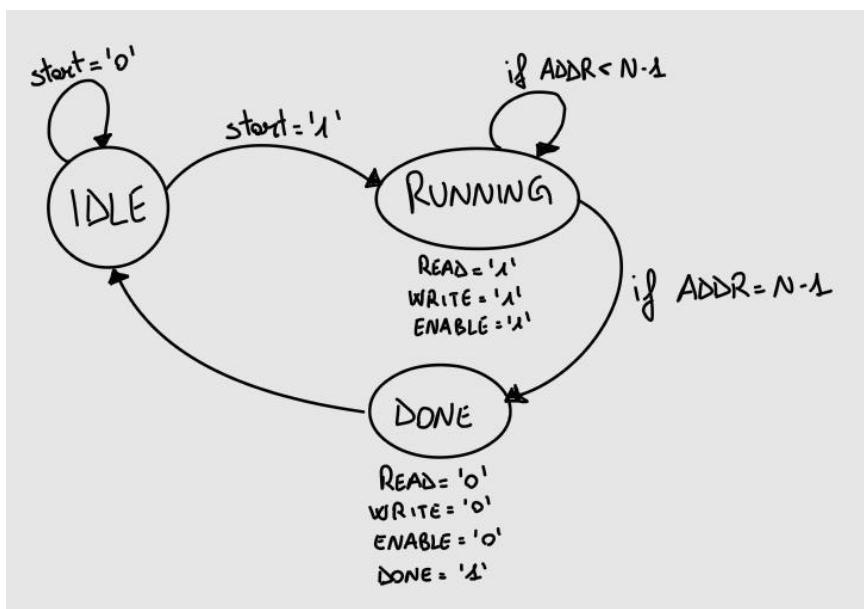
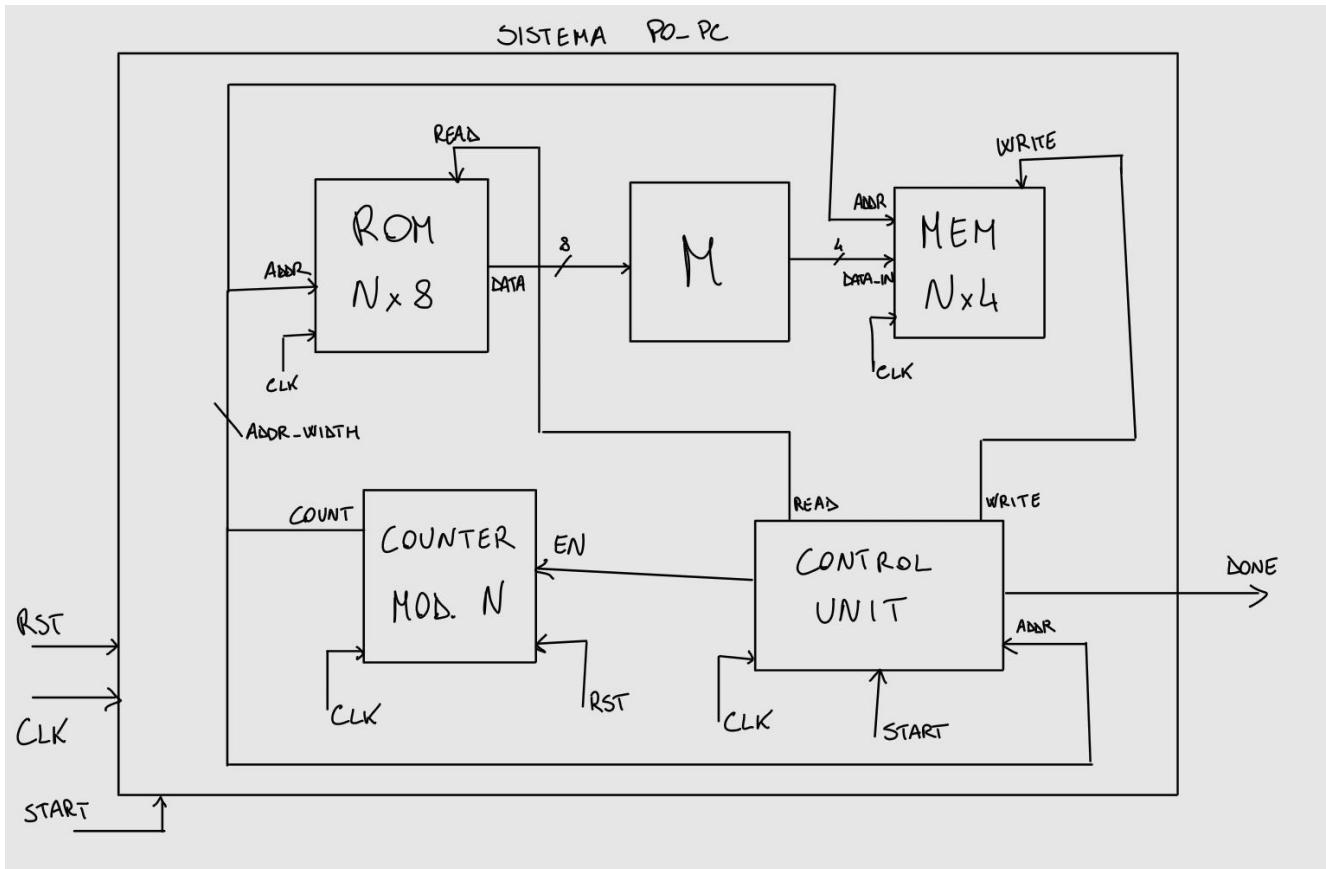
1. Legge un valore a 8 bit dalla ROM.
2. Trasforma il valore letto in una stringa a 4 bit tramite la macchina combinatoria M.
3. Scrive il valore trasformato nella memoria MEM alla stessa locazione di memoria.
4. Incrementa l'indirizzo di lettura e scrittura, finché tutte le locazioni non sono state processate.

Il sistema utilizza una memoria ROM sincrona per la lettura e una MEM sincrona per la scrittura. Il contatore degli indirizzi gestisce invece la scansione sequenziale delle memorie.

Per quanto riguarda il contatore (il cui scopo è generare gli indirizzi per leggere la ROM e scrivere nella MEM), è utile sottolineare che la dimensione dell'indirizzo dipende dal numero di locazioni N ($\log_2(N)$ bit). Invece, per quanto riguarda la macchina combinatoria M, si è scelto di implementare una funzione che permette di portare in uscita i 4 MSB del segnale in ingresso.

Infine, l'unità di controllo attiva il contatore quando il segnale di START è alto ed abilita la lettura dalla ROM e la conseguente scrittura sulla MEM. Essa termina l'elaborazione quando poi tutte le locazioni sono state processate.

Di seguito sono riportati il disegno del sistema e dell'automa a stati finiti:



Implementazione

Di seguito viene riportato il codice VHDL del sistema complessivo (l'implementazione del contatore è riportata in appendice, mentre quella della macchina M è la medesima di quella riportata in un esercizio precedente):

- Sistema_PO_PC (top level)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity sistema_PO_PC is
  Generic (
    N : integer := 8;
    ADDR_WIDTH : integer := 3
  );
  Port (
    CLK : in STD_LOGIC;
    RESET : in STD_LOGIC;
    START : in STD_LOGIC;
    DONE : out STD_LOGIC
  );
end sistema_PO_PC;

architecture Structural of sistema_PO_PC is
  signal ADDR : STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
  signal DATA_ROM : STD_LOGIC_VECTOR(7 downto 0);
  signal DATA_M, DATA_MEM_OUT : STD_LOGIC_VECTOR(3 downto 0);
  signal EN, READ, WRITE : STD_LOGIC;

  component ROM
    Generic (N : integer; ADDR_WIDTH : integer);
    Port(
      CLK : in STD_LOGIC;
      READ : in STD_LOGIC;
      ADDR : in STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
      DATA : out STD_LOGIC_VECTOR(7 downto 0)
    );
  end component;

  component M
    Port(
      x : in STD_LOGIC_VECTOR(7 downto 0); -- Ingresso a 8 bit
      y : out STD_LOGIC_VECTOR(3 downto 0) -- Uscita a 4 bit
    );
  end component;

  component MEM
    Generic (N : integer; ADDR_WIDTH : integer);
    Port (
      CLK : in STD_LOGIC;
      WRITE : in STD_LOGIC;
      ADDR : in STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
      DATA_IN : in STD_LOGIC_VECTOR(3 downto 0);
      DATA_OUT : out STD_LOGIC_VECTOR(3 downto 0)
    );
  end component;

```

```

end component;

component counter
  Generic ( MAX_VALUE : integer; DIM : integer);
  Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    SET : in STD_LOGIC;
    SET_VALUE : in STD_LOGIC_VECTOR(DIM-1 downto 0);
    COUNT : out STD_LOGIC_VECTOR(DIM-1 downto 0);
    CARRY : out STD_LOGIC
  );
end component;

component control_unit
  Generic ( N : integer);
  Port (
    CLK : in STD_LOGIC;
    START : in STD_LOGIC;
    RST : in STD_LOGIC;
    ADDR : in integer range 0 to N-1;
    EN : out STD_LOGIC;
    READ : out STD_LOGIC;
    WRITE : out STD_LOGIC;
    DONE : out STD_LOGIC
  );
end component;

begin
  memoria_ROM : ROM
    Generic Map(N => N, ADDR_WIDTH => ADDR_WIDTH)
    Port Map(
      CLK => CLK,
      READ => READ,
      ADDR => ADDR,
      DATA => DATA_ROM
    );
  macchina_M : M
    Port Map(
      x => DATA_ROM,
      y => DATA_M
    );
  memoria_MEM : MEM
    Generic Map(N => N, ADDR_WIDTH => ADDR_WIDTH)
    Port Map(
      CLK => CLK,
      WRITE => WRITE,
      ADDR => ADDR,
      DATA_IN => DATA_M,

```

```

        DATA_OUT => DATA_MEM_OUT
    );
}

contatore : counter
Generic Map(MAX_VALUE => N-1, DIM => ADDR_WIDTH)
Port Map(
    CLK => CLK,
    RST => RESET,
    EN => EN,
    SET => '0',
    SET_VALUE => (others => '0'),
    COUNT => ADDR,
    CARRY => OPEN
);
u_c : control_unit
Generic Map(N => N)
Port Map(
    CLK => CLK,
    START => START,
    RST => RESET,
    ADDR => to_integer(unsigned(ADDR)),
    EN => EN,
    READ => READ,
    WRITE => WRITE,
    DONE => DONE
);

```

end Structural;

- ROM

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity ROM is
    Generic (N : integer := 8; ADDR_WIDTH : integer := 3);
    Port (
        CLK : in STD_LOGIC;
        READ : in STD_LOGIC;
        ADDR : in STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
        DATA : out STD_LOGIC_VECTOR(7 downto 0)
    );
end ROM;

architecture Behavioral of ROM is
type rom_type is array (0 to N-1) of std_logic_vector(7 downto 0);

```

```

signal ROM : rom_type := (
    "00000000",
    "00010000",
    "00100000",
    "00110000",
    "01000000",
    "01010000",
    "01110000",
    "10000000"
);
attribute rom_style : string;
attribute rom_style of ROM : signal is "block";

begin
    process(CLK) begin
        if rising_edge(CLK) then
            if(READ = '1') then
                DATA <= ROM(to_integer(unsigned(ADDR)));
            end if;
        end if;
    end process;
end Behavioral;

```

- MEM

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

entity MEM is
    Generic (N : integer := 8; ADDR_WIDTH : integer := 3);
    Port (
        CLK : in STD_LOGIC;
        WRITE : in STD_LOGIC;
        ADDR : in STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
        DATA_IN : in STD_LOGIC_VECTOR(3 downto 0);
        DATA_OUT : out STD_LOGIC_VECTOR(3 downto 0)
    );
end MEM;

architecture Behavioral of MEM is
    type mem_type is array (0 to N-1) of std_logic_vector(3 downto 0);
    signal MEM : mem_type := (others => (others => '0'));-- Inizializza tutta la memoria a 0
begin
    process(CLK) begin
        if rising_edge(CLK) then
            if(WRITE = '1') then
                MEM(to_integer(unsigned(ADDR))) <= DATA_IN;
            end if;
    end process;

```

```

    end if;
end process;

DATA_OUT <= MEM(conv_integer(ADDR)); -- lettura asincrona

end Behavioral;

```

- Unità di controllo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity control_unit is
Generic (
    N : integer := 8 -- numero di locazioni di memoria
);
Port (
    CLK : in STD_LOGIC;
    START : in STD_LOGIC;
    RST : in STD_LOGIC;
    ADDR : in integer range 0 to N-1; -- indirizzo corrente del contatore
    EN : out STD_LOGIC;
    READ : out STD_LOGIC;
    WRITE : out STD_LOGIC;
    DONE : out STD_LOGIC
);
end control_unit;

architecture Structural of control_unit is
type state is (IDLE, RUNNING, DONE_STATE);
signal current_state,next_state : state;
begin

reg_stato: process(CLK)
begin
if(CLK'event and CLK='1') then
    if(RST='1') then
        current_state <= IDLE;
    else
        current_state <= next_state;
    end if;
end if;
end process;

comb : process(current_state, start) begin
case current_state is

when IDLE =>
    EN <= '0';
    READ <= '0';

```

```

WRITE <= '0';
DONE <= '0';

if START = '1' then
    next_state <= RUNNING;
else
    next_state <= IDLE;
end if;

when RUNNING =>
    EN <= '1';
    READ <= '1';
    WRITE <= '1';
    DONE <= '0';

    if (ADDR = N-1) then
        next_state <= DONE_STATE;
    else
        next_state <= RUNNING;
    end if;

when DONE_STATE =>
    EN <= '0';
    READ <= '0';
    WRITE <= '0';
    DONE <= '1';

    next_state <= IDLE;

end case;
end process;
end Structural;

```

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: Esercizi sulle 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

Il moltiplicatore di Booth appartiene alla famiglia dei moltiplicatori sequenziali. Esso si basa sulla codifica di uno dei due operandi e si presta bene con numeri signed. Lo scopo di questa codifica è quello di ridurre i termini da sommare fra loro e, dunque, ridurre i tempi di propagazione.

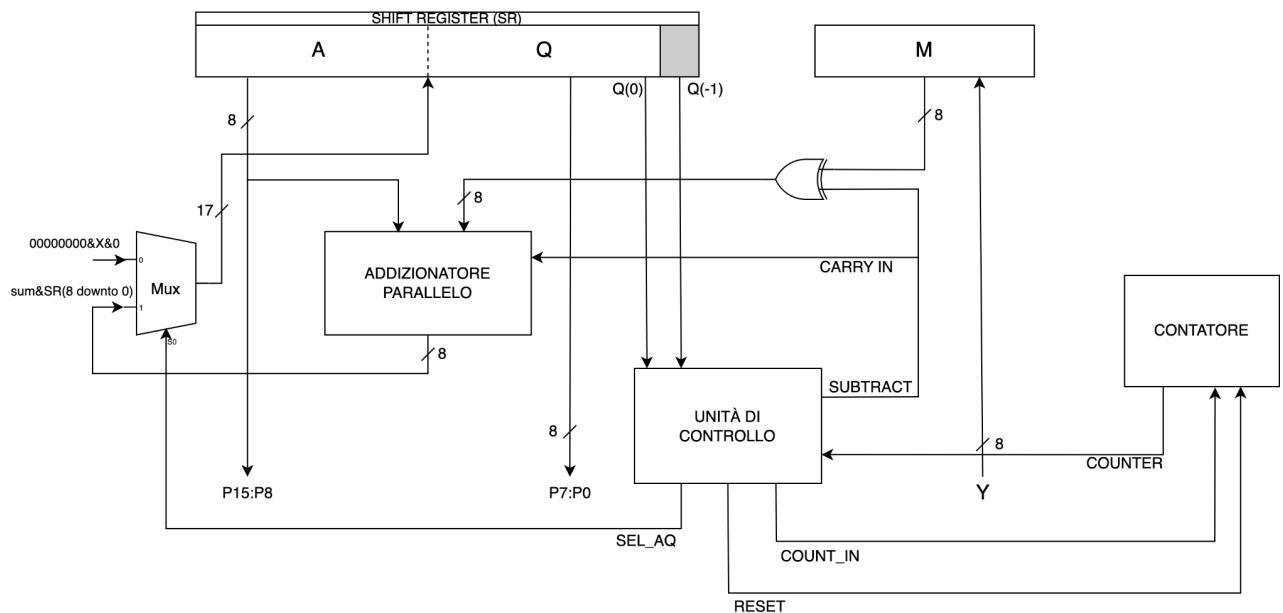
Dati due numeri X (moltiplicatore) e Y (moltiplicando), la rappresentazione di Booth può essere facilmente ottenuta sostituendo ciascuna coppia di bit adiacenti di X con un valore in {-1, 0, 1}, secondo la corrispondenza in tabella:

$x_j \ x_{j-1}$	codifica
00/11	0
01	+1
10	-1

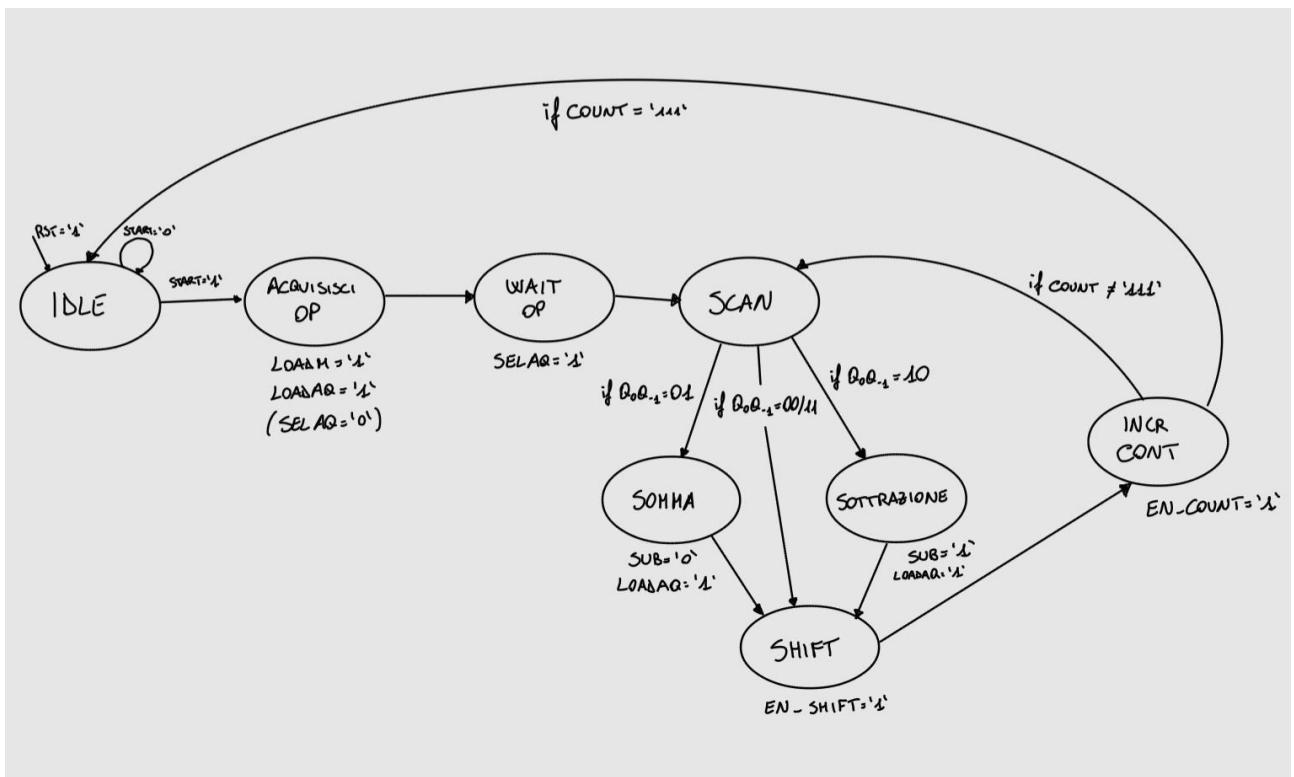
Poiché ogni cifra della rappresentazione Booth del moltiplicatore appartiene a {-1, 0, 1}, a seconda della cifra coinvolta nel prodotto verrà effettuata un'operazione di somma o di differenza (o nessuna delle due) prima dello shift, seguendo lo schema mostrato in basso:

$x_j \ x_{j-1}$	operazione
00/11	non viene effettuata né la somma né la sottrazione, ma solo lo shift
01	Y viene aggiunto al prodotto parziale corrente
10	Y viene sottratto dal prodotto parziale corrente

Di seguito è riportato il disegno dell'architettura del moltiplicatore:



Inoltre, è riportato in basso l'automa a stati finiti:



Implementazione

Di seguito vengono riportati i codici VHDL per la parte operativa e la parte di controllo del moltiplicatore (il codice dell'adder subtractor, del registro8, del multiplexer e del contatore è riportato in appendice):

- MOLTIPLICATORE DI BOOTH (top level)

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

entity molt_Booth is
    Port (
        clock, reset, start : in STD_LOGIC;
        X, Y: in std_logic_vector(7 downto 0);
        P: out std_logic_vector(15 downto 0)
    );
end molt_Booth;

architecture structural of molt_Booth is

component unita_operativa
    Port (
        X, Y : in STD_LOGIC_VECTOR(7 downto 0);
        clock, reset : in STD_LOGIC;
        loadAQ, shiftAQ, loadM, sub, selAQ, count_in : in STD_LOGIC;
        COUNT : out STD_LOGIC_VECTOR(2 downto 0);
        q0, q_1 : out STD_LOGIC;
        P : out STD_LOGIC_VECTOR(15 downto 0)
    );
end component;

component unita_controllo
    Port (
        clock, reset, start : in STD_LOGIC;
        COUNT : in STD_LOGIC_VECTOR(2 downto 0);
        q0, q_1 : in STD_LOGIC;
        loadAQ, shiftAQ, loadM, sub, selAQ, count_in : out STD_LOGIC
    );
end component;

signal tmp_loadAQ, tmp_shiftAQ, tmp_loadM, tmp_sub, tmp_selAQ, tmp_count_in : STD_LOGIC;
signal tmp_count : STD_LOGIC_VECTOR(2 downto 0);
signal tmp_q0, tmp_q_1 : STD_LOGIC;
signal temp_p: std_logic_vector(15 downto 0);

begin

    UO : unita_operativa
        port map(
            X => X,
            Y => Y,
            clock => clock,
            reset => reset,
```

```

loadAQ => tmp_loadAQ,
shiftAQ => tmp_shiftAQ,
loadM => tmp_loadM,
sub => tmp_sub,
selAQ => tmp_selAQ,
count_in => tmp_count_in,
COUNT => tmp_count,
q0 => tmp_q0,
q_1 => tmp_q_1,
P => temp_P
);

UC : unita_controllo
port map(
    clock => clock,
    reset => reset,
    start => start,
    COUNT => tmp_count,
    q0 => tmp_q0,
    q_1 => tmp_q_1,
    loadAQ => tmp_loadAQ,
    shiftAQ => tmp_shiftAQ,
    loadM => tmp_loadM,
    sub => tmp_sub,
    selAQ => tmp_selAQ,
    count_in => tmp_count_in
);
P<=temp_p;

end structural;

```

- UNITÀ OPERATIVA

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity unita_operativa is
    Port (
        X, Y : in STD_LOGIC_VECTOR(7 downto 0);
        clock, reset : in STD_LOGIC;
        loadAQ, shiftAQ, loadM, sub, selAQ, count_in : in STD_LOGIC;
        COUNT : out STD_LOGIC_VECTOR(2 downto 0);
        q0, q_1 : out STD_LOGIC;
        P : out STD_LOGIC_VECTOR(15 downto 0)
    );
end unita_operativa;

architecture structural of unita_operativa is

```

```

component shift_register
  port( parallel_in: in std_logic_vector(16 downto 0);
    clock, reset, load, shift: in std_logic;
    q0, q_1 : out std_logic;
    parallel_out: out std_logic_vector(16 downto 0));
end component;
```



```

component registro8
  port( A: in std_logic_vector(7 downto 0);
    clk, res, load: in std_logic;
    B: out std_logic_vector(7 downto 0));
end component;
```



```

component cont_mod8
  port( clock, reset: in std_logic;
    count_in: in std_logic;
    count: out std_logic_vector(2 downto 0));
end component;
```



```

component adder_sub
  port( X, Y: in std_logic_vector(7 downto 0);
    cin: in std_logic;
    Z: out std_logic_vector(7 downto 0);
    cout: out std_logic);
end component;
```



```

component mux_21 is
  generic (width : integer);
  port( x0, x1: in std_logic_vector(width-1 downto 0);
    s: in std_logic;
    y: out std_logic_vector(width-1 downto 0));
end component;
```



```

signal tmp_out_M : STD_LOGIC_VECTOR(7 downto 0);
signal AQ_init: std_logic_vector(16 downto 0); --segnale in input all'SR
signal sum: std_logic_vector(7 downto 0); --uscita del parallel adder
signal AQ_sum_in : std_logic_vector(16 downto 0);
signal AQ_out: std_logic_vector(16 downto 0); --segnale temporaneo uscita dell'SR
signal AQ_in: std_logic_vector(16 downto 0); --segnale in input all'SR
signal riporto: std_logic; -- riporto in uscita dell'adder che non utilizziamo
```



```

begin
  -- 1) predisposizione del secondo operando della somma:
  M : registro8
  port map(
    A => Y,
    clk => clock,
    res => reset,
    load => loadM,
    B => tmp_out_M
  );

```

```
-- 2) predisposizione del primo operando della somma:
```

```
--stringa da 17 bit da inserire nello shift register A.Q durante la fase di inizializzazione:
```

```
--è ottenuta concatenando 00000000 con il moltiplicatore X e con uno 0 finale
```

```
AQ_init <= "00000000" & X & '0'; --valore da inserire all'inizio nello shift register
```

```
--stringa di 17 bit da inserire nello shift register A.Q durante la fase operativa dopo aver effettuato la somma
```

```
AQ_sum_in <= sum & AQ_out(8 downto 0);
```

```
-- mux per selezionare l'ingresso parallelo dello shift register: valore iniziale AQ_init
```

```
-- oppure uscita dell'adder AQ_sum_in
```

```
MUX_SR_parallel_in : mux_21
```

```
generic map (width => 17)
```

```
port map(
```

```
  x0 => AQ_init,
```

```
  x1 => AQ_sum_in,
```

```
  s => selAQ,
```

```
  y => AQ_in
```

```
);
```

```
-- 3) shift register A.Q
```

```
AQ : shift_register
```

```
port map(
```

```
  parallel_in => AQ_in,
```

```
  clock => clock,
```

```
  reset => reset,
```

```
  load => loadAQ,
```

```
  shift => shiftAQ,
```

```
  q0 => q0,
```

```
  q_1 => q_1,
```

```
  parallel_out => AQ_out
```

```
);
```

```
-- 4) sommatore
```

```
ADD_SUB : adder_sub
```

```
port map(
```

```
  X => AQ_out(16 downto 9),
```

```
  Y => tmp_out_M,
```

```
  cin => sub,
```

```
  Z => sum,
```

```
  cout => riporto
```

```
);
```

```
--5) contatore
```

```
CONT : cont_mod8
```

```
port map(
```

```
  clock => clock,
```

```
  reset => reset,
```

```
  count_in => count_in,
```

```
  count => count
```

```
};
```

--6) uscita del moltiplicatore, corrispondente al valore contenuto nello shift register

```
P<=AQ_out(16 downto 1);
```

```
end structural;
```

- Shift Register

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

---shift register che contiene inizialmente una stringa di 8 zeri e il moltiplicatore X
---al termine dell'operazione di moltiplicazione conterrà il risultato A+Q

```
entity shift_register is
  port( parallel_in: in std_logic_vector(16 downto 0);
        clock, reset, load, shift: in std_logic;
        q0, q_1 : out STD_LOGIC;
        parallel_out: out std_logic_vector(16 downto 0));
end shift_register;

architecture behavioural of shift_register is

  signal temp: std_logic_vector(16 downto 0);

begin

SR: process(clock)
begin
  if(clock'event and clock='1') then
    if(reset='1') then
      temp <=(others=>'0');
    else
      if(load='1') then --caricamento iniziale del moltiplicatore
        temp(16 downto 0) <= parallel_in;
        --temp(0) <= '0';
      elsif(shift='1') then
        temp(16) <= temp(16);
        temp(15 downto 0) <= temp(16 downto 1);
      end if;
    end if;

  end if;
end process;
```

```
parallel_out <= temp;
q0 <= temp(1);
q_1 <= temp(0);
```

```
end behavioural;
```

- UNITÀ DI CONTROLLO

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

entity unita_controllo is
  Port (
    clock, reset, start : in STD_LOGIC;
    COUNT : in STD_LOGIC_VECTOR(2 downto 0);
    q0, q_1 : in STD_LOGIC;
    loadAQ, shiftAQ, loadM, sub, selAQ, count_in : out STD_LOGIC
  );
end unita_controllo;

architecture Behavioral of unita_controllo is

  type state is (idle, acquisisci_op, scan, somma, sottrazione, shift, incr_cont, wait_op);
  signal current_state : state := IDLE;
  signal next_state : state;

begin

  parte_comb : process(current_state, start, count)
  begin
    loadAQ <= '0';
    shiftAQ <= '0';
    loadM <= '0';
    sub <= '0';
    count_in <= '0';

    case current_state is

      when idle =>
        selAQ <= '0';
        if (start = '0') then
          next_state <= idle;
        else
          next_state <= acquisisci_op;
        end if;

      when acquisisci_op =>
        loadM <= '1';
        loadAQ <= '1';
        next_state <= wait_op;

      when wait_op =>
        selAQ <= '1';
        next_state <= scan;

      when others =>
        null;
    end case;
  end process;
end;
```

```

when scan =>
    --selAQ <= '1';
    if (q0 = '0' and q_1 = '1') then
        next_state <= somma;
    elsif (q0 = '1' and q_1 = '0') then
        next_state <= sottrazione;
    else
        next_state <= shift;
    end if;

when somma =>
    sub <= '0';
    loadAQ <= '1';
    next_state <= shift;

when sottrazione =>
    sub <= '1';
    loadAQ <= '1';
    next_state <= shift;

when shift =>
    shiftAQ <= '1';
    next_state <= incr_cont;

when incr_cont =>
    count_in <= '1';
    if (count /= "111") then
        next_state <= scan;
    else
        next_state <= idle;
    end if;
end case;
end process;

```

```

parte_sequenziale : 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;

end Behavioral;

```

Simulazione

Il seguente testbench verifica il funzionamento del moltiplicatore di Booth. Esso genera un segnale di clock con periodo di 20 ns e gestisce il reset del sistema. Vengono testati tre casi di moltiplicazione:

1. 3×2 , entrambi numeri positivi.
2. -4×3 , con un operando negativo.
3. -5×-6 , entrambi gli operandi negativi.

Ogni test inizia con un reset, seguito dall'assegnazione dei valori agli ingressi X e Y. Dopo un breve ritardo, il segnale start viene attivato per avviare la moltiplicazione. Il testbench attende quindi un tempo sufficiente per il completamento del calcolo prima di passare al test successivo.

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

ENTITY tb_molt_Booth IS
END tb_molt_Booth;

ARCHITECTURE testbench OF tb_molt_Booth IS

COMPONENT molt_Booth
PORT (
    clock, reset, start : IN STD_LOGIC;
    X, Y : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    P : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
);
END COMPONENT;

SIGNAL clock, reset, start : STD_LOGIC := '0';
SIGNAL X, Y : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL P : STD_LOGIC_VECTOR(15 DOWNTO 0);

CONSTANT clk_period : TIME := 20 ns;

BEGIN

DUT : molt_Booth
PORT MAP (
    clock => clock,
    reset => reset,
    start => start,
    X => X,
    Y => Y,
    P => P
);

clock_process : PROCESS
BEGIN
```

```

WHILE TRUE LOOP
    clock <= '0';
    WAIT FOR clk_period / 2;
    clock <= '1';
    WAIT FOR clk_period / 2;
END LOOP;
WAIT;
END PROCESS;

stim_proc: PROCESS
BEGIN
    reset <= '1';
    WAIT FOR 20 ns;
    reset <= '0';
    WAIT FOR 20 ns;

    -- Test case 1: 3 * 2
    X <= "00000011"; -- 3
    Y <= "00000010"; -- 2
    WAIT FOR 40 ns;
    start <= '1';
    WAIT FOR 20 ns;
    start <= '0';
    WAIT FOR 700 ns;

    reset <= '1';
    WAIT FOR 20 ns;
    reset <= '0';
    WAIT FOR 20 ns;

    -- Test case 2: -4 * 3
    X <= "11111100"; -- -4 (Two's complement)
    Y <= "00000011"; -- 3
    WAIT FOR 40 ns;
    start <= '1';
    WAIT FOR 20 ns;
    start <= '0';
    WAIT FOR 700 ns;

    reset <= '1';
    WAIT FOR 20 ns;
    reset <= '0';
    WAIT FOR 20 ns;

    -- Test case 3: -5 * -6
    X <= "11111011"; --- -5
    Y <= "11111010"; --- -6
    WAIT FOR 40 ns;
    start <= '1';
    WAIT FOR 20 ns;
    start <= '0';
    WAIT FOR 700 ns;

```

```

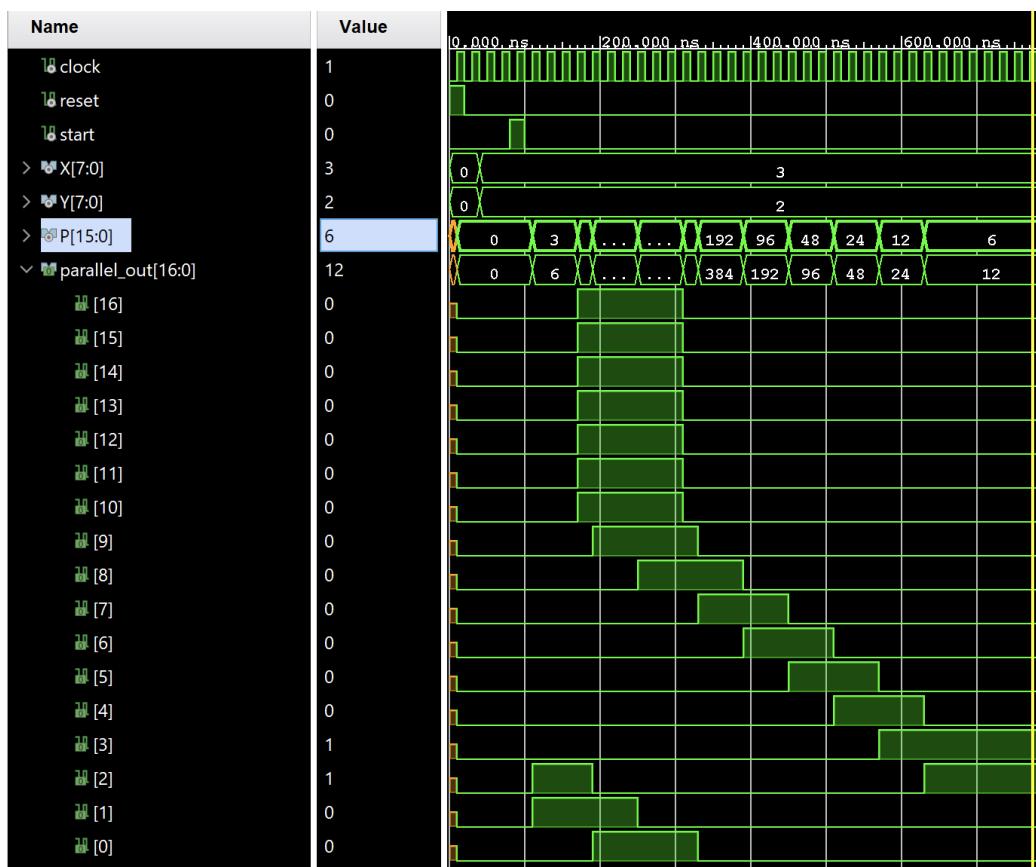
WAIT;
END PROCESS;

END testbench;

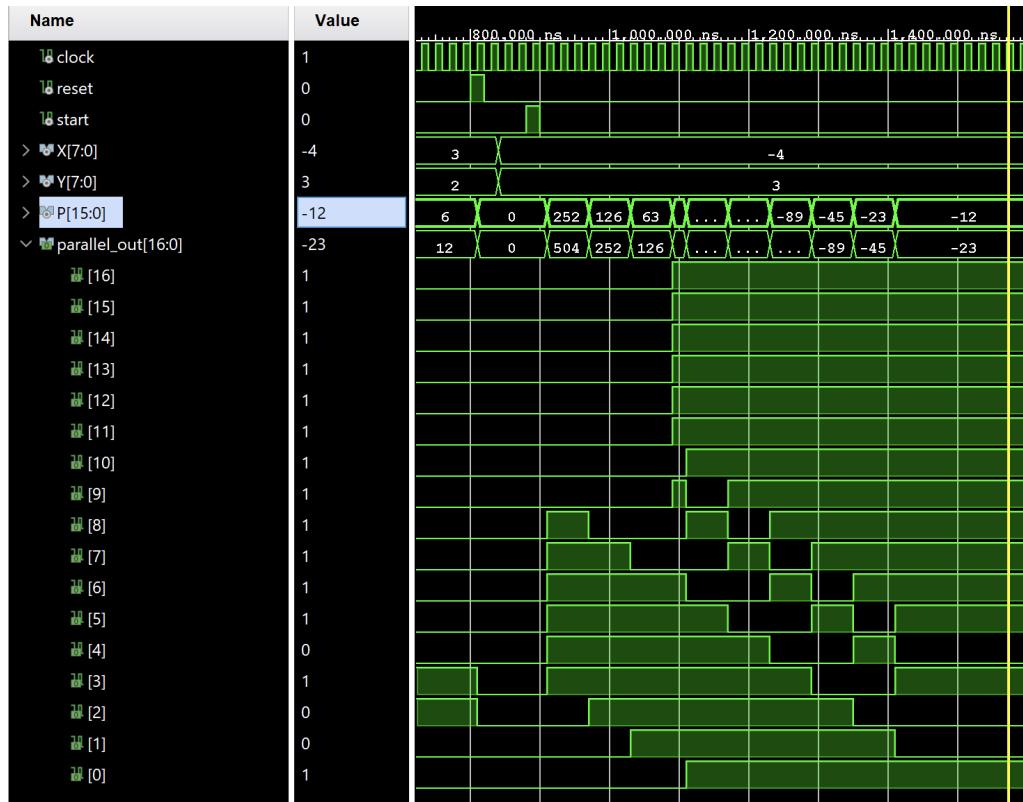
```

Al termine della simulazione si ottengono i seguenti risultati, in cui è riportata anche la waveform del registro a scorrimento (`parallel_out`) per mostrarne il funzionamento:

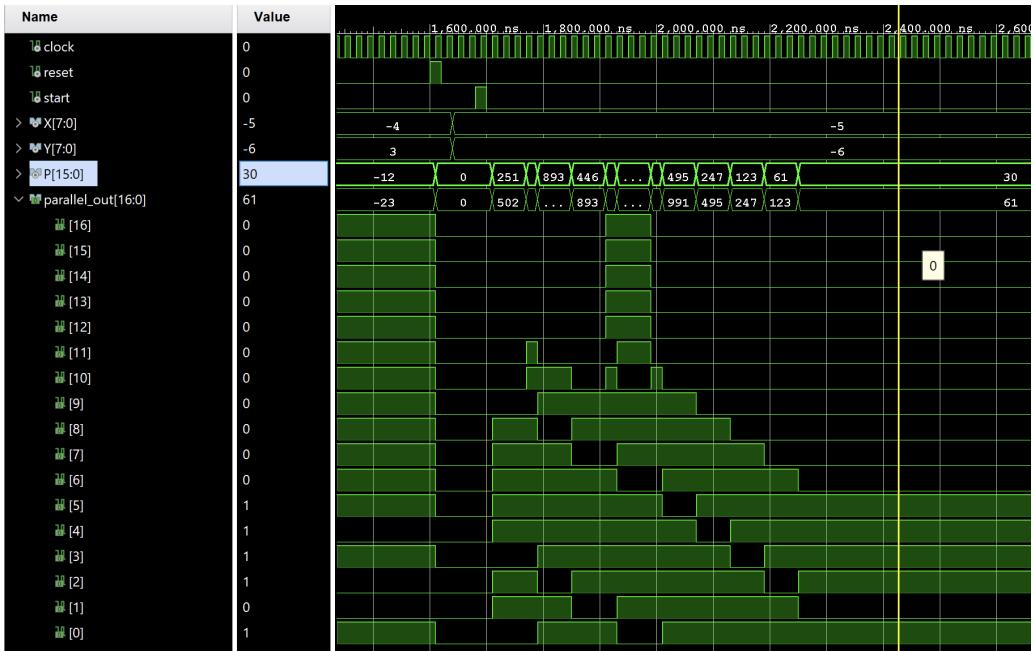
1. Risultato del prodotto di due numeri positivi, 3 x 2:



2. Risultato del prodotto di un numero negativo ed uno positivo, -4 x 3:



3. Risultato del prodotto di due numeri negativi, -5 x -6:



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

Per la realizzazione della sintesi del moltiplicatore su FPGA è stato aggiunto il componente ButtonDebouncer. Esso prende in input il segnale proveniente dal bottone e genera un segnale "ripulito" che presenta un impulso della durata di un colpo di clock per segnalare l'avvenuta pressione del bottone. Inoltre, si è deciso di mappare il segnale di ingresso di start sul bottone BTNC ed il segnale di reset sul bottone BTND. L'inserimento dei due operandi avviene tramite l'utilizzo degli switch presenti sulla board, mentre l'output viene mostrato sui led.

Di seguito è riportato il codice aggiornato con l'aggiunta del componente button debouncer ed il file di constraint:

- Moltiplicatore di Booth (top level)

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

entity molt_Booth is
  Port (
    clock, reset, start : in STD_LOGIC;
    X, Y: in std_logic_vector(7 downto 0);
    P: out std_logic_vector(15 downto 0)
  );
end molt_Booth;

architecture structural of molt_Booth is

component unita_operativa
  Port (
    X, Y : in STD_LOGIC_VECTOR(7 downto 0);
    clock, reset : in STD_LOGIC;
    loadAQ, shiftAQ, loadM, sub, selAQ, count_in : in STD_LOGIC;
    COUNT : out STD_LOGIC_VECTOR(2 downto 0);
    q0, q_1 : out STD_LOGIC;
    P : out STD_LOGIC_VECTOR(15 downto 0)
  );
end component;

component unita_controllo
  Port (
    clock, reset, start : in STD_LOGIC;
    COUNT : in STD_LOGIC_VECTOR(2 downto 0);
    q0, q_1 : in STD_LOGIC;
    loadAQ, shiftAQ, loadM, sub, selAQ, count_in : out STD_LOGIC
  );
end component;

component ButtonDebouncer
  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;
```

```

signal tmp_loadAQ, tmp_shiftAQ, tmp_loadM, tmp_sub, tmp_selAQ, tmp_count_in : STD_LOGIC;
signal tmp_count : STD_LOGIC_VECTOR(2 downto 0);
signal tmp_q0, tmp_q_1 : STD_LOGIC;
signal temp_p: std_logic_vector(15 downto 0);
signal cleared_start : STD_LOGIC;
```

```

begin
```

```

BD : ButtonDebouncer
    generic map (CLK_period => 10, btn_noise_time => 10000000)
    port map(
```

```

        RST => reset,
        CLK => clock,
        BTN => start,
        CLEARED_BTN => cleared_start
    );
```

```

UO : unita_operativa
    port map(
```

```

        X => X,
        Y => Y,
        clock => clock,
        reset => reset,
        loadAQ => tmp_loadAQ,
        shiftAQ => tmp_shiftAQ,
        loadM => tmp_loadM,
        sub => tmp_sub,
        selAQ => tmp_selAQ,
        count_in => tmp_count_in,
        COUNT => tmp_count,
        q0 => tmp_q0,
        q_1 => tmp_q_1,
        P => temp_P
    );
```

```

UC : unita_controllo
    port map(
```

```

        clock => clock,
        reset => reset,
        start => cleared_start,
        COUNT => tmp_count,
        q0 => tmp_q0,
```

```

    q_1 => tmp_q_1,
    loadAQ => tmp_loadAQ,
    shiftAQ => tmp_shiftAQ,
    loadM => tmp_loadM,
    sub => tmp_sub,
    selAQ => tmp_selAQ,
    count_in => tmp_count_in
);

P<=temp_p;

end structural;

```

- Constraint

```

## 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 { X[0] }]; #IO_L24N_T3_RS0_15
Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { X[1] }];
#IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { X[2] }]; #IO_L6N_T0_D08_VREF_14
Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { X[3] }]; #IO_L13N_T2_MRCC_14
Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { X[4] }]; #IO_L12N_T1_MRCC_14
Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { X[5] }]; #IO_L7N_T1_D10_14
Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { X[6] }]; #IO_L17N_T2_A13_D29_14
Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { X[7] }]; #IO_L5N_T0_D07_14
Sch=sw[7]
set_property -dict { PACKAGE_PIN T8 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_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { Y[2] }];
#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { Y[3] }]; #IO_L23P_T3_A03_D19_14
Sch=sw[11]
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { Y[4] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { Y[5] }]; #IO_L20P_T3_A08_D24_14
Sch=sw[13]
set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { Y[6] }];
#IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { Y[7] }]; #IO_L21P_T3_DQS_14
Sch=sw[15]

## LEDs

```

```

set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { P[0] }]; #IO_L18P_T2_A24_15
Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { P[1] }]; #IO_L24P_T3_RS1_15
Sch=led[1]
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { P[2] }]; #IO_L17N_T2_A25_15
Sch=led[2]
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { P[3] }]; #IO_L8P_T1_D11_14
Sch=led[3]
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { P[4] }]; #IO_L7P_T1_D09_14
Sch=led[4]
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { P[5] }]; #IO_L18N_T2_A11_D27_14
Sch=led[5]
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { P[6] }]; #IO_L17P_T2_A14_D30_14
Sch=led[6]
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { P[7] }]; #IO_L18P_T2_A12_D28_14
Sch=led[7]
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { P[8] }]; #IO_L16N_T2_A15_D31_14
Sch=led[8]
set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { P[9] }]; #IO_L14N_T2_SRCC_14
Sch=led[9]
set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { P[10] }]; #IO_L22P_T3_A05_D21_14
Sch=led[10]
set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { P[11] }];
#IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { P[12] }]; #IO_L16P_T2_CSI_B_14
Sch=led[12]
set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { P[13] }]; #IO_L22N_T3_A04_D20_14
Sch=led[13]
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { P[14] }]; #IO_L20N_T3_A07_D23_14
Sch=led[14]
set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { P[15] }];
#IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

##Buttons
#set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports { start }]; #IO_L9P_T1_DQS_14
Sch=btnc
#set_property -dict { PACKAGE_PIN M18 IOSTANDARD LVCMOS33 } [get_ports { BTNU }]; #IO_L4N_T0_D05_14
Sch=btnu
#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
set_property -dict { PACKAGE_PIN P18 IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L9N_T1_DQS_D13_14
Sch=btnd

```

Capitolo 4: Handshaking

Esercizio 8: Comunicazione con Handshaking

Esercizio 8.1

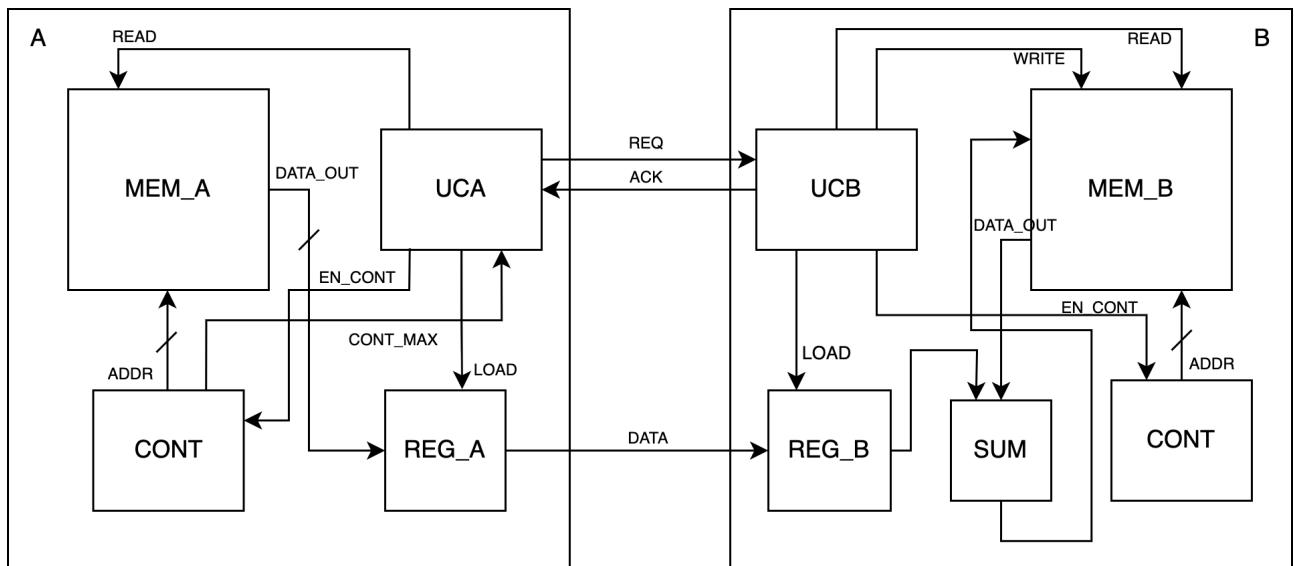
Progettare, implementare in VHDL e testare mediante simulazione un sistema composto da 2 nodi, A e B, che comunicano mediante un protocollo di handshaking. Il nodo A e il nodo B possiedono entrambi una memoria interna in cui sono memorizzate N stringhe di M bit, denominate X(i) e Y(i) rispettivamente ($i=0,..,N-1$). 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

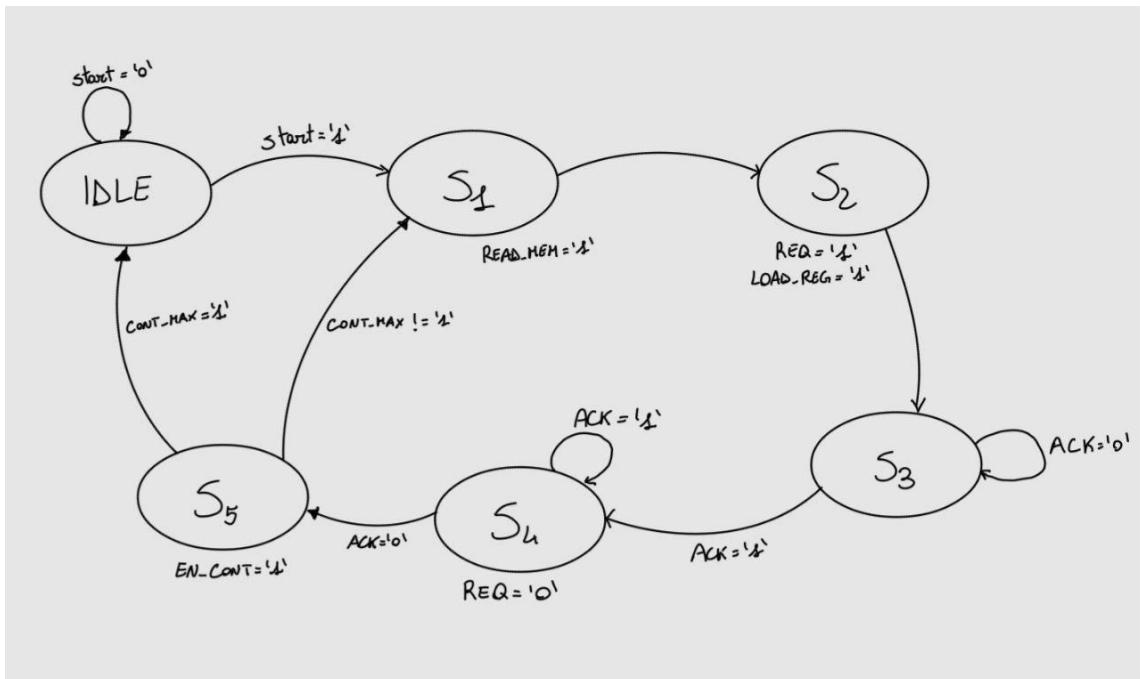
Per realizzare il sistema composto dai due nodi, A e B, che comunicano tramite protocollo di handshaking, sono stati integrati diversi componenti: una memoria interna, un contatore ed un'unità di controllo per la gestione del flusso di controllo per ciascun nodo, registri per il trasferimento dei dati e un componente dedicato all'operazione di somma all'interno del nodo B.

Di seguito è riportato un disegno dell'architettura (per non appesantire troppo il disegno e renderlo più leggibile sono stati omessi i segnali di clock, reset e start):

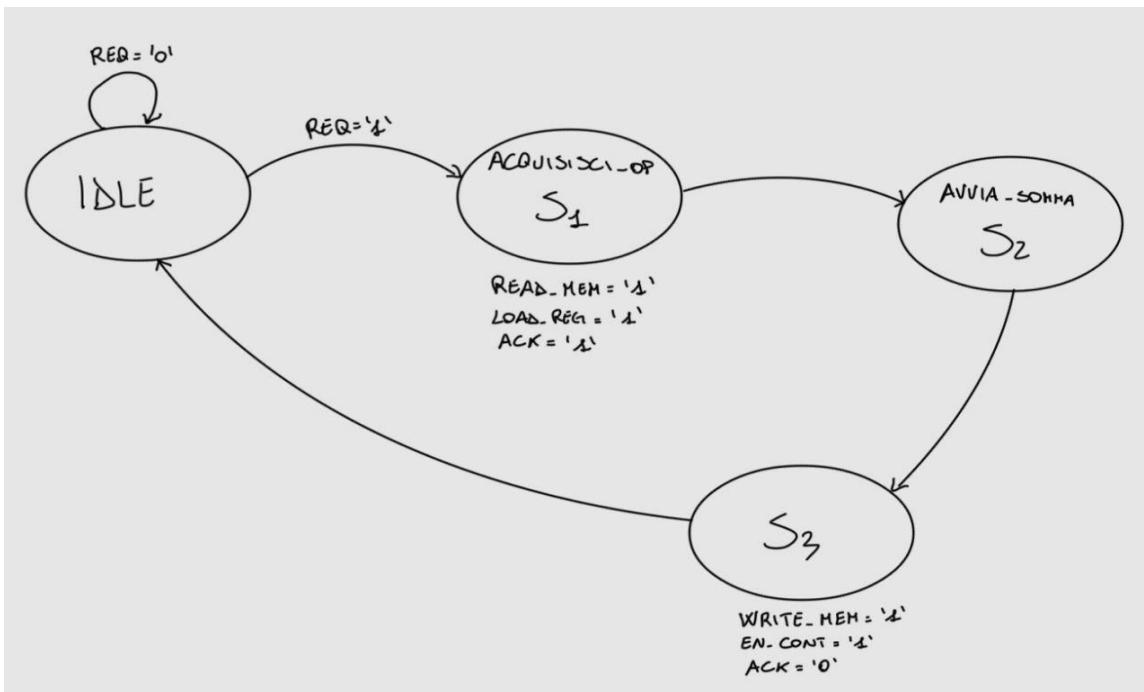


Inoltre, sono riportati gli automi dei due nodi A e B:

- **NODO A**



- **NODO B**



Implementazione

Di seguito viene riportato il codice VHDL dei vari componenti (il codice del contatore è riportato in appendice, mentre quello relativo al registro del nodo B è stato omesso, poiché uguale a quello del nodo A):

- Handshaking (top level)

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

entity Handshaking is
  Port (
    CLK_A : in STD_LOGIC;
    CLK_B : in STD_LOGIC;
    START : in STD_LOGIC;
    RST : in STD_LOGIC;
    DATA_OUT : out STD_LOGIC_VECTOR(3 downto 0)
  );
end Handshaking;

architecture Structural of Handshaking is

  signal tmp_req, tmp_ack : STD_LOGIC;
  signal data_outA : STD_LOGIC_VECTOR(3 downto 0);

  component NodoA
    Port (
      START : in STD_LOGIC;
      CLK_A : in STD_LOGIC;
      RST : in STD_LOGIC;
      ACK : in STD_LOGIC;
      REQ : out STD_LOGIC;
      DATA_OUT : out STD_LOGIC_VECTOR(3 downto 0)
    );
  end component;

  component NodoB
    Port (
      CLK_B : in STD_LOGIC;
      RST : in STD_LOGIC;
      DATA_IN : in STD_LOGIC_VECTOR(3 downto 0);
      REQ : in STD_LOGIC;
      ACK : out STD_LOGIC
    );
  end component;

begin

  A : NodoA
    Port Map(
      START => START,
      CLK_A => CLK_A,
```

```

    RST => RST,
    ACK => tmp_ack,
    REQ => tmp_req,
    DATA_OUT => data_outA
);

B : NodoB
Port Map(
    CLK_B => CLK_B,
    RST => RST,
    DATA_IN => data_outA,
    REQ => tmp_req,
    ACK => tmp_ack
);
end Structural;

```

- Nodo A

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NodoA is
Port (
    START : in STD_LOGIC;
    CLK_A : in STD_LOGIC;
    RST : in STD_LOGIC;
    ACK : in STD_LOGIC;
    REQ : out STD_LOGIC;
    DATA_OUT : out STD_LOGIC_VECTOR(3 downto 0)
);
end NodoA;

architecture Structural of NodoA is
    signal tmp_read_mem, tmp_en_count, tmp_load_reg, carry : STD_LOGIC;
    signal tmp_addr : STD_LOGIC_VECTOR(2 downto 0);
    signal mem_out : STD_LOGIC_VECTOR(3 downto 0);

component ROM
    Generic (
        N : integer;
        DIM : integer;
        M : integer
    );
    Port(
        clk : in STD_LOGIC;
        read : in STD_LOGIC;
        addr : in STD_LOGIC_VECTOR(DIM-1 downto 0);
        data : out STD_LOGIC_VECTOR(M-1 downto 0)
    );
end component;

```

```

component counter
  Generic (
    MAX_VALUE : integer;
    DIM : integer
  );
  Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    SET : in STD_LOGIC;
    SET_VALUE : in STD_LOGIC_VECTOR(DIM-1 downto 0);
    COUNT : out STD_LOGIC_VECTOR(DIM-1 downto 0);
    CARRY : out STD_LOGIC
  );
end component;

```

```

component Registro
  Port (
    D : in STD_LOGIC_VECTOR(3 downto 0);
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    Q : out STD_LOGIC_VECTOR(3 downto 0)
  );
end component;

```

```

component UCA
  Port (
    CLK_A : in STD_LOGIC;
    RST : in STD_LOGIC;
    START : in STD_LOGIC;
    ACK : in STD_LOGIC;
    CONT_MAX : in STD_LOGIC;
    REQ : out STD_LOGIC;
    READ_MEM : out STD_LOGIC;
    EN_COUNT : out STD_LOGIC;
    LOAD_REG : out STD_LOGIC
  );
end component;

```

begin

```

MEM : ROM
  Generic Map(N => 8, DIM => 3, M => 4)
  Port Map(
    clk => CLK_A,
    read => tmp_read_mem,
    addr => tmp_addr,
    data => mem_out
  );

```

COUNT : counter

```

Generic Map(MAX_VALUE => 7, DIM => 3)
Port Map(
    CLK => CLK_A,
    RST => RST,
    EN => tmp_en_count,
    SET => '0',
    SET_VALUE => "000",
    COUNT => tmp_addr,
    CARRY => carry
);

REG : Registro
Port Map(
    D => mem_out,
    CLK => CLK_A,
    RST => RST,
    EN => tmp_load_reg,
    Q => DATA_OUT
);

control_unit : UCA
Port Map(
    CLK_A => CLK_A,
    RST => RST,
    START => START,
    ACK => ACK,
    CONT_MAX => carry,
    REQ => REQ,
    READ_MEM => tmp_read_mem,
    EN_COUNT => tmp_en_count,
    LOAD_REG => tmp_load_reg
);

end Structural;

```

- MEM_A

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Memoria composta da N stringhe di M bit
entity ROM is
    Generic ( N : integer := 8;
        DIM : integer := 3;
        M : integer := 4
    );
    Port(
        clk : in STD_LOGIC;
        read : in STD_LOGIC;
        addr : in STD_LOGIC_VECTOR(DIM-1 downto 0);

```

```

    data : out STD_LOGIC_VECTOR(M-1 downto 0)
);
end ROM;

architecture Behavioral of ROM is
    signal data_out : STD_LOGIC_VECTOR(M-1 downto 0);
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if read = '1' then
                case addr is
                    when "000" => data_out <= "0001"; -- Locazione 0
                    when "001" => data_out <= "0010"; -- Locazione 1
                    when "010" => data_out <= "0100"; -- Locazione 2
                    when "011" => data_out <= "1000"; -- Locazione 3
                    when "100" => data_out <= "1001"; -- Locazione 4
                    when "101" => data_out <= "1010"; -- Locazione 5
                    when "110" => data_out <= "1100"; -- Locazione 6
                    when "111" => data_out <= "1111"; -- Locazione 7
                    when others => data_out <= "0000";
                end case;
            end if;
        end if;
    end process;
    data <= data_out;
end Behavioral;

```

- REG_A

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Registro is
    Port (
        D : in STD_LOGIC_VECTOR(3 downto 0);
        CLK : in STD_LOGIC;
        RST : in STD_LOGIC;
        EN : in STD_LOGIC;
        Q : out STD_LOGIC_VECTOR(3 downto 0)
    );
end Registro;

architecture Behavioral of Registro is
    signal Q_reg : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
begin
    process (CLK)
    begin
        if (CLK'event and CLK='1') then
            if (RST = '1') then

```

```

    Q_reg <= (others => '0');
  elsif (EN = '1') then
    Q_reg <= D;
  end if;
end if;
end process;

Q <= Q_reg;

end Behavioral;

```

- UCA

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity UCA is
  Port (
    CLK_A : in STD_LOGIC;
    RST : in STD_LOGIC;
    START : in STD_LOGIC;
    ACK : in STD_LOGIC;
    CONT_MAX : in STD_LOGIC;
    REQ: out STD_LOGIC;
    READ_MEM : out STD_LOGIC;
    EN_COUNT : out STD_LOGIC;
    LOAD_REG : out STD_LOGIC
  );
end UCA;

architecture Behavioral of UCA is
  type stato is (IDLE, S1, S2, S3, S4, S5);

  signal stato_corrente : stato := IDLE;
  signal stato_prossimo : stato;

begin
  -- Parte combinatoria
  comb: process(stato_corrente, START, ACK, CONT_MAX)
  begin
    REQ <= '0';
    READ_MEM <= '0';
    EN_COUNT <= '0';
    LOAD_REG <= '0';

    case stato_corrente is
      when IDLE =>
        if(start = '0') then
          stato_prossimo <= IDLE;
        else

```

```

elsif(start = '1') then
    stato_prossimo <= S1;
end if;

when S1 =>
    READ_MEM <= '1';
    stato_prossimo <= S2;

when S2 =>
    REQ <= '1';
    LOAD_REG <= '1';
    stato_prossimo <= S3;

when S3 =>
    REQ <= '1';
    if (ACK = '0') then
        stato_prossimo <= S3;
    else
        stato_prossimo <= S4;
    end if;

when S4 =>
    if (ACK = '1') then
        stato_prossimo <= S4;
    else
        stato_prossimo <= S5;
    end if;

when S5 =>
    EN_COUNT <= '1';
    if (CONT_MAX /= '1') then
        stato_prossimo <= S1;
    else
        stato_prossimo <= IDLE;
    end if;

end case;
end process;

```

```

-- Parte Sequenziale
mem: process (CLK_A)
begin
    if( CLK_A'event and CLK_A = '1' ) then
        if( RST = '1' ) then
            stato_corrente <= IDLE;
        else
            stato_corrente <= stato_prossimo;
        end if;
    end if;
end process;

```

```
end Behavioral;
```

- NODO B

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

entity NodoB is
  Port (
    CLK_B : in STD_LOGIC;
    RST : in STD_LOGIC;
    DATA_IN : in STD_LOGIC_VECTOR(3 downto 0);
    REQ : in STD_LOGIC;
    ACK : out STD_LOGIC
  );
end NodoB;

architecture Structural of NodoB is
  signal tmp_read_mem, tmp_write_mem, tmp_en_count, tmp_load_reg, en_sum : STD_LOGIC;
  signal tmp_addr : STD_LOGIC_VECTOR(2 downto 0);
  signal mem_out, tmp_reg_out, tmp_sum_out : STD_LOGIC_VECTOR(3 downto 0);

component MEM
  Generic(
    N : integer;
    DIM : integer;
    M : integer
  );
  Port(
    clk : in STD_LOGIC;
    read : in STD_LOGIC;
    write : in STD_LOGIC;
    data_in : in STD_LOGIC_VECTOR(M-1 downto 0);
    addr : in STD_LOGIC_VECTOR(DIM-1 downto 0);
    data_out : out STD_LOGIC_VECTOR(M-1 downto 0)
  );
end component;

component counter
  Generic (
    MAX_VALUE : integer;
    DIM : integer
  );
  Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    SET : in STD_LOGIC;
    SET_VALUE : in STD_LOGIC_VECTOR(DIM-1 downto 0);
    COUNT : out STD_LOGIC_VECTOR(DIM-1 downto 0);
  );
end component;
```

```

CARRY : out STD_LOGIC
);
end component;

component Registro
Port (
  D : in STD_LOGIC_VECTOR(3 downto 0);
  CLK : in STD_LOGIC;
  RST : in STD_LOGIC;
  EN : in STD_LOGIC;
  Q : out STD_LOGIC_VECTOR(3 downto 0)
);
end component;

component UCB
Port (
  CLK_B : in STD_LOGIC;
  RST : in STD_LOGIC;
  REQ : in STD_LOGIC;
  COUNT : in STD_LOGIC_VECTOR(2 downto 0);
  ACK: out STD_LOGIC;
  READ_MEM : out STD_LOGIC;
  WRITE_MEM : out STD_LOGIC;
  EN_COUNT : out STD_LOGIC;
  EN_SUM : out STD_LOGIC;
  LOAD_REG : out STD_LOGIC
);
end component;

component SUM
Generic (
  M : integer -- Numero di bit dei numeri da sommare
);
Port (
  clk : in STD_LOGIC; -- Clock
  A : in STD_LOGIC_VECTOR(M-1 downto 0); -- Primo operando
  B : in STD_LOGIC_VECTOR(M-1 downto 0); -- Secondo operando
  SUM : out STD_LOGIC_VECTOR(M-1 downto 0) -- Risultato della somma
);
end component;

begin

  MEMORIA : MEM
  Generic Map(N => 8, DIM => 3, M => 4)
  Port Map(
    clk => CLK_B,
    read => tmp_read_mem,
    write => tmp_write_mem,
    data_in => tmp_sum_out,
    addr => tmp_addr,

```

```

    data_out => mem_out
);

SOMMA : SUM
Generic Map(
    M => 4
)
Port Map(
    clk => CLK_B,
    A => tmp_reg_out,
    B => mem_out,
    SUM => tmp_sum_out
);
;

COUNT : counter
Generic Map(MAX_VALUE => 7, DIM => 3)
Port Map(
    CLK => CLK_B,
    RST => RST,
    EN => tmp_en_count,
    SET => '0',
    SET_VALUE => "000",
    COUNT => tmp_addr,
    CARRY => open
);
;

REG : Registro
Port Map(
    D => DATA_IN,
    CLK => CLK_B,
    RST => RST,
    EN => tmp_load_reg,
    Q => tmp_reg_out
);
;

control_unit : UCB
Port Map(
    CLK_B => CLK_B,
    RST => RST,
    REQ => REQ,
    COUNT => tmp_addr,
    ACK => ACK,
    READ_MEM => tmp_read_mem,
    WRITE_MEM => tmp_write_mem,
    EN_COUNT => tmp_en_count,
    LOAD_REG => tmp_load_reg
);
;

end Structural;

```

- MEM_B

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MEM is
  Generic(
    N : integer := 8;
    DIM : integer := 3;
    M : integer := 4
  );
  Port(
    clk : in STD_LOGIC;
    read : in STD_LOGIC;
    write : in STD_LOGIC;
    data_in : in STD_LOGIC_VECTOR(3 downto 0);
    addr : in STD_LOGIC_VECTOR(2 downto 0);
    data_out : out STD_LOGIC_VECTOR(3 downto 0)
  );
end MEM;

architecture Behavioral of MEM is
  -- Definizione della memoria come array di N locazioni di M bit
  type memory_type is array (0 to N-1) of STD_LOGIC_VECTOR(M-1 downto 0);

  -- Inizializzazione
  signal mem : memory_type := (
    "0001", "0010", "0100", "1000",
    "1001", "1010", "1100", "1111"
  );

begin
  process(clk)
  begin
    if rising_edge(clk) then
      if write = '1' then
        -- Scrittura: aggiorna la memoria all'indirizzo specificato
        mem(CONV_INTEGER(addr)) <= data_in;
      elsif read = '1' then
        -- Lettura: assegna il valore della memoria a data_out
        data_out <= mem(CONV_INTEGER(addr));
      end if;
    end if;
  end process;
end Behavioral;

```

- SUM

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SUM is
  Generic (
    M : integer := 4 -- Numero di bit dei numeri da sommare
  );
  Port (
    clk : in STD_LOGIC; -- Clock
    A : in STD_LOGIC_VECTOR(M-1 downto 0); -- Primo operando
    B : in STD_LOGIC_VECTOR(M-1 downto 0); -- Secondo operando
    SUM : out STD_LOGIC_VECTOR(M-1 downto 0) -- Risultato della somma
  );
end SUM;

architecture Behavioral of SUM is

begin
  process(clk)
  begin
    if rising_edge(clk) then
      SUM <= A + B;
    end if;
  end process;
end Behavioral;

```

- UCB

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity UCB is
  Port (
    CLK_B : in STD_LOGIC;
    RST : in STD_LOGIC;
    REQ : in STD_LOGIC;
    COUNT : in STD_LOGIC_VECTOR(2 downto 0);
    ACK: out STD_LOGIC;
    READ_MEM : out STD_LOGIC;
    WRITE_MEM : out STD_LOGIC;
    EN_COUNT : out STD_LOGIC;
    EN_SUM : out STD_LOGIC;
    LOAD_REG : out STD_LOGIC
  );
end UCB;

```

```

end UCB;

architecture Behavioral of UCB is

    type stato is (IDLE, S1, S2, S3);
    signal stato_corrente : stato := IDLE;
    signal stato_prossimo : stato;

    begin

        -- Parte combinatoria
        comb : process (stato_corrente, req, count) begin
            EN_COUNT <= '0';
            WRITE_MEM <= '0';

            case stato_corrente is
                when IDLE =>
                    if (REQ = '0') then
                        stato_prossimo <= IDLE;
                    else
                        stato_prossimo <= S1;
                    end if;

                when S1 =>
                    ACK <= '1';
                    READ_MEM <= '1';
                    LOAD_REG <= '1';
                    stato_prossimo <= S2;

                when S2 =>
                    stato_prossimo <= S3;

                when S3 =>
                    WRITE_MEM <= '1';
                    EN_COUNT <= '1';
                    ACK <= '0';
                    stato_prossimo <= IDLE;
            end case;
        end process;

        -- Parte sequenziale
        mem : process(CLK_B) begin
            if rising_edge(CLK_B) then
                if (RST = '1') then
                    stato_corrente <= IDLE;
                else
                    stato_corrente <= stato_prossimo;
                end if;
            end if;
        end process;

    end Behavioral;

```

Simulazione

Il seguente testbench verifica il funzionamento del modulo Handshaking. Genera un segnale di clock per ogni nodo e simula il comportamento del sistema. Inizialmente, viene applicato un reset, poi viene attivato il segnale START per avviare la comunicazione tra i due nodi. Dopo un periodo di attesa per consentire il completamento della trasmissione, la simulazione termina.

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

entity Handshaking_tb is
end Handshaking_tb;

architecture testbench of Handshaking_tb is
    signal CLK_A, CLK_B, START, RST : STD_LOGIC := '0';
    constant CLK_PERIOD_A : time := 10 ns;
    constant CLK_PERIOD_B : time := 15 ns;

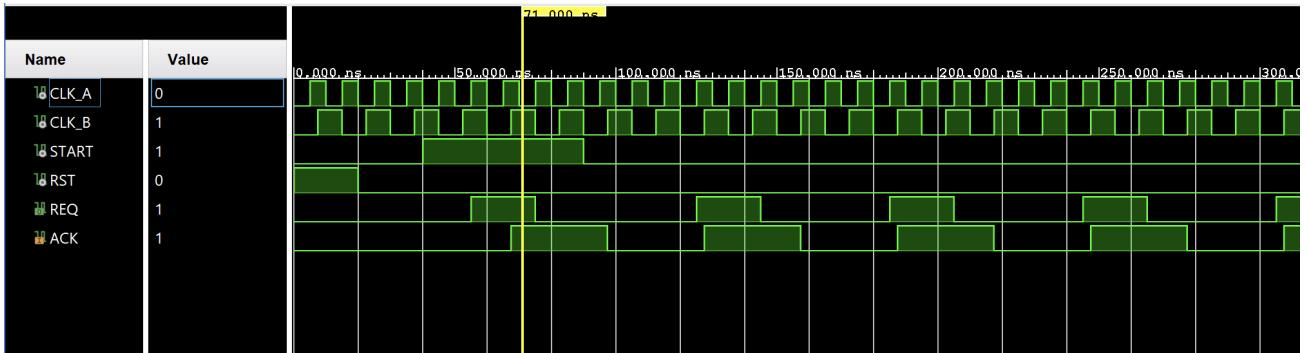
    component Handshaking
        Port (
            CLK_A : in STD_LOGIC;
            CLK_B : in STD_LOGIC;
            START : in STD_LOGIC;
            RST : in STD_LOGIC
        );
    end component;

begin
    -- Instanza del DUT (Device Under Test)
    DUT: Handshaking
        port map (
            CLK_A => CLK_A,
            CLK_B => CLK_B,
            START => START,
            RST => RST
        );
    -- Processo per generare il clock A
    process
    begin
        while true loop
            CLK_A <= '0';
            wait for CLK_PERIOD_A/2;
            CLK_A <= '1';
            wait for CLK_PERIOD_A/2;
        end loop;
    end process;
    -- Processo per generare il clock B
    process
```

```

begin
  while true loop
    CLK_B <= '0';
    wait for CLK_PERIOD_B/2;
    CLK_B <= '1';
    wait for CLK_PERIOD_B/2;
  end loop;
end process;
-- Processo di test
process
begin
  -- Reset iniziale
  RST <= '1';
  START <= '0';
  wait for 20 ns;
  RST <= '0';
  wait for 20 ns;
  -- Avvio della comunicazione
  START <= '1';
  wait for 50 ns;
  START <= '0';
  -- Attendere il completamento della trasmissione
  wait for 200 ns;
  -- Fine simulazione
  wait;
end process;
end testbench;

```



Objects		Protocol Instances
Name	Value	
clk	1	
read	1	
write	0	
> data_in[3:0]	0001	
> addr[2:0]	000	
> data_out[3:0]	0010	
mem[0:7][3:0]	0010,0100,1000,0000,0010,0100,0000,0000	
> [0][3:0]	0010	
> [1][3:0]	0100	
> [2][3:0]	1000	
> [3][3:0]	0000	
> [4][3:0]	0010	
> [5][3:0]	0100	
> [6][3:0]	1000	
> [7][3:0]	1110	

Dai risultati della simulazione (mostrati in figura), il sistema si comporta correttamente secondo le aspettative. Al termine dell'esecuzione, la memoria del nodo B contiene i valori risultanti dalle operazioni di somma, confermando il corretto funzionamento del protocollo di handshaking e dell'intero processo di elaborazione.

Capitolo 5: Processore

Esercizio 9

A partire dall'implementazione fornita del processore operante

secondo il modello IJVM,

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

Progetto e architettura

Iniziamo dalla simulazione dell'architettura IJVM del processore, effettuata simulando il seguente programma:

```
.main
.var
a
.endvar
BIPUSH 0xA // Carica 10 sullo stack
BIPUSH 0xE // Carica 14 sullo stack
IADD // Esegue 14 + 10 = 24
ISTORE a // Salva il risultato nella variabile 'a'
HALT
.endmethod
```

Di seguito, vengono riportati i risultati della simulazione:

[Riporta come vengono eseguite le istruzioni, il comportamento dello stack, della memoria, dei registri e del Program counter, e come l'architettura IJVM gestisce il ciclo fetch-decode-execute per ogni istruzione].

Approfondimento Istruzioni a scelta

Prima di vedere nello specifico le istruzioni scelte, ricordiamo che ogni microistruzione è formata da 3 fasi:

1. **FETCH**: il processore legge l'opcode dell'istruzione dalla memoria.
2. **DECODE**: l'unità di controllo riconosce dunque l'istruzione dall'opcode e attiva una serie di segnali di controllo necessari.
3. **EXECUTE**: il processore effettua (nel caso, ad esempio, dell'IADD) il pop del primo valore dallo stack, poi il pop del secondo; l'ALU esegue l'addizione e poi viene effettuato il push del risultato sullo stack.

Le istruzioni dell'architettura MIC-1 scelte che andremo ad approfondire sono:

- **IADD** (opcode: 0x65)
- **BIPUSH** (opcode: 0x10)

Iniziamo dalla **IADD** → il microprogramma è il seguente:

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

Analizziamo ogni riga, in modo da capire il funzionamento dell'istruzione:

- **MAR = SP = SP - 1; rd** → lo stack pointer (**SP**) viene decrementato di 1, in modo da accedere al penultimo valore dello stack. Questo valore verrà poi caricato in memoria all'indirizzo specificato dal MAR (*Memory Address Register*), e infine letto (**rd**).
- **H = TOS** → il valore in cima allo stack (**TOS**) viene salvato nel registro **H** (ovvero viene salvato il primo operando da sommare); questo consente di mantenerlo temporaneamente per l'operazione aritmetica.
- **MDR = TOS = MDR + H** → (N.B.: il secondo operando è nel MDR) si esegue poi la somma tra il valore appena letto (**MDR**) e quello salvato nel registro **H**. Il risultato dell'addizione viene poi salvato nel **TOS**.
- **wr; goto main** → il risultato viene scritto in memoria, e si ritorna al ciclo principale dell'interprete IJVM.

Vediamo adesso la **BIPUSH**:

```
bipush = 0x10:  
SP = MAR = SP + 1  
PC = PC + 1; fetch
```

```
MDR = TOS = MBR; wr; goto main
```

Come prima, analizziamo ogni riga per capire come funziona:

- $SP = MAR = SP + 1 \rightarrow$ si incrementa lo stack pointer per creare il nuovo valore da inserire; l'indirizzo aggiornato dello stack (SP) viene poi caricato nel MAR.
- $PC = PC + 1;$ fetch \rightarrow il program counter viene a questo punto incrementato, per leggere l'operando immediato dell'istruzione. Con 'fetch', il byte viene poi caricato nel MBR (Memory Buffer Register).
- $MDR = TOS = MBR; wr \rightarrow$ il valore letto (MBR) viene memorizzato come nuovo valore sulla cima dello stack e poi scritto in memoria.
- goto main \rightarrow ritorno al ciclo principale per eseguire l'istruzione successiva.

Approfondiamo adesso il punto **b)** dell'esercizio, e supponiamo di andare a sostituire l'addizione con la sottrazione, ovvero:

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

La modifica effettuata consente di analizzare come l'architettura gestisce operazioni aritmetiche diverse (sottrazione al posto della somma).

Il microprogramma per ISUB è simile a IADD, ma esegue una sottrazione:

```
isub = 0x5C:
MAR = SP = SP - 1; rd
H = TOS
MDR = TOS = MDR - H; wr; goto main
```

Capitolo 6: Interfaccia Seriale

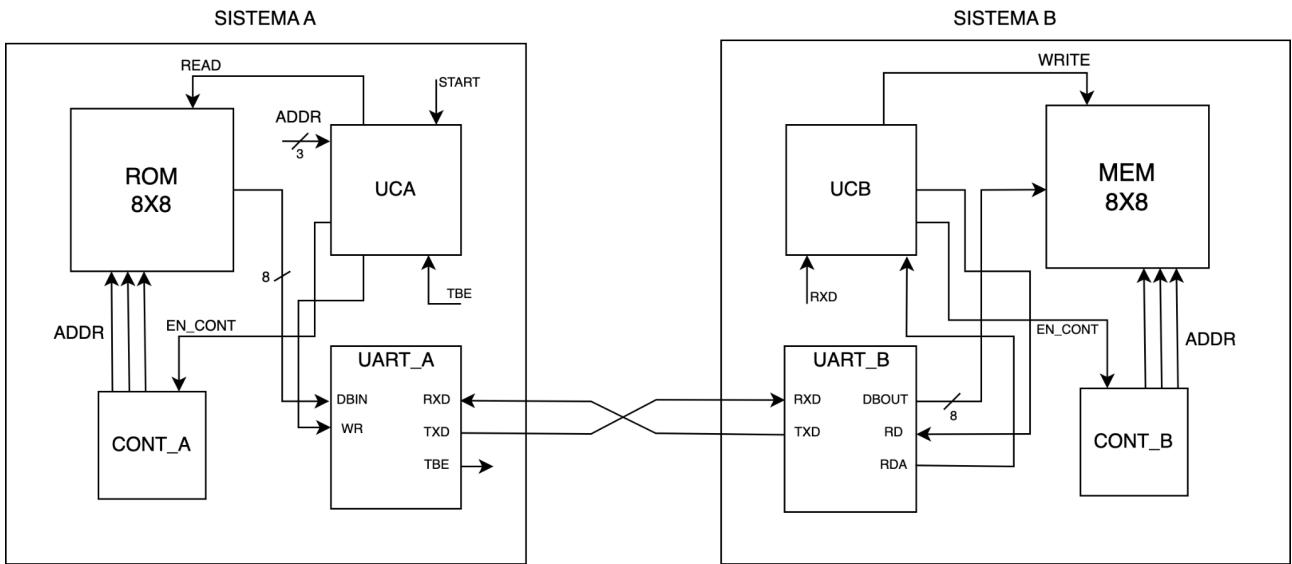
Esercizio 10

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

Per la realizzazione del sistema complessivo è stato utilizzato un approccio strutturale. Esso si compone di due moduli, sistema A e sistema B, che comunicano tramite interfaccia UART. Entrambi i sistemi sono composti da una memoria (nel sistema A avviene una lettura mentre nel sistema B una scrittura), un contatore per scandire le locazioni della memoria, una UART per permettere ai sistemi di comunicare ed infine un'unità di controllo.

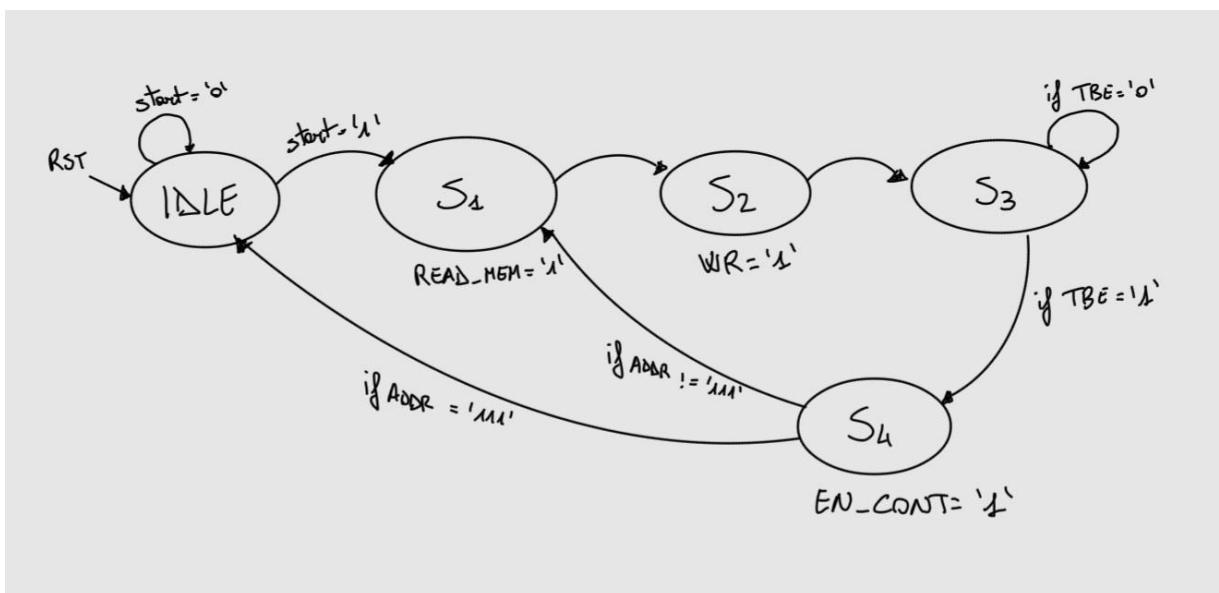
In basso è riportato il disegno dell'architettura:



Inoltre, sono riportati i disegni degli automi dei due sistemi A e B:

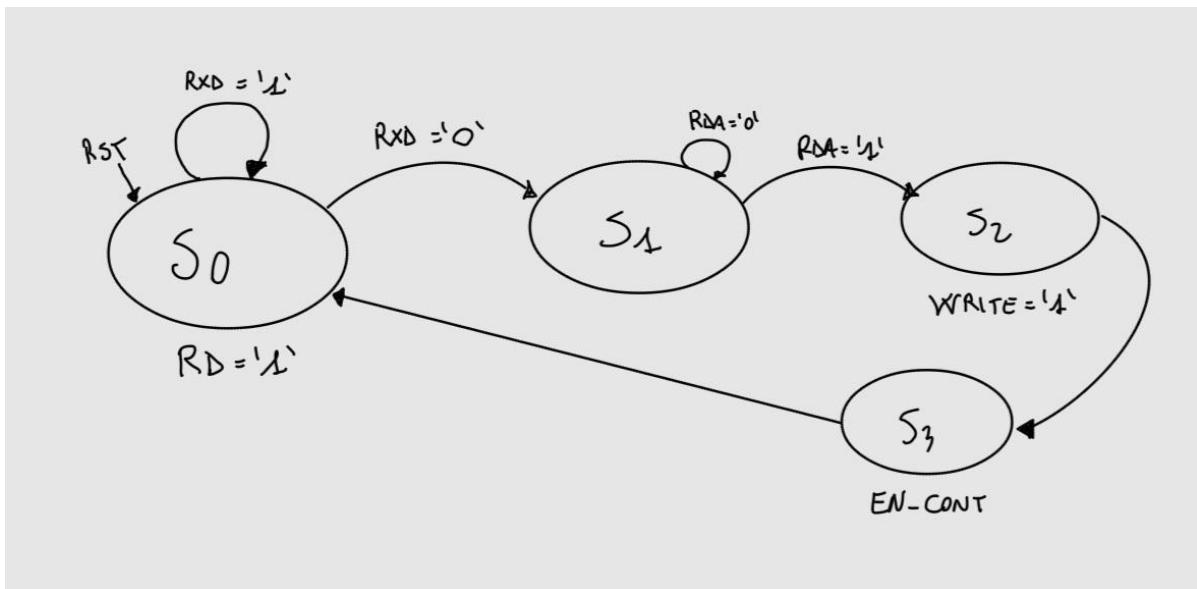
- **Automa del sistema A**

Il sistema rimane nello stato IDLE fino all'assegnazione del segnale START. Successivamente, viene letta la stringa dalla memoria e attivato il segnale WR, consentendo la scrittura del dato nel registro dell'UART. L'abilitazione di WR provoca l'abbassamento di TBE, quindi il sistema attende che quest'ultimo torni alto, indicando che il dato è stato caricato nello shift register. Infine, il contatore viene abilitato per permettere la lettura della successiva locazione di memoria.



- **Automa del sistema B**

Il sistema attende la ricezione dello start bit (valore logico basso). L'abilitazione di RD provoca l'abbassamento del segnale RDA, quindi si attende che quest'ultimo ritorni alto, indicando che il dato è disponibile nel registro rdReg. Una volta ricevuta la stringa, questa viene salvata nella memoria del sistema B e, infine, viene data l'abilitazione al contatore per permettere la scrittura nella prossima locazione.



Implementazione

Di seguito è riportato il codice VHDL (il codice dell'UART-RS232 non è riportato essendo fornito dalla Digilent, mentre quello del contatore è presente in appendice):

- **Sistema_UART (top level)**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Sistema_UART is
Port (
    START : in STD_LOGIC;
    RST : in STD_LOGIC;
    CLK : in STD_LOGIC
);
end Sistema_UART;

architecture Structural of Sistema_UART is

signal internal_tx : STD_LOGIC;

component Sistema_A
Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    START : in STD_LOGIC;
    
```

```

        TXD : out STD_LOGIC
    );
end component;

component Sistema_B
Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    RXD : in STD_LOGIC
);
end component;

begin

A : Sistema_A
port map(
    CLK => CLK,
    RST => RST,
    START => START,
    TXD => internal_tx
);

B : Sistema_B
port map(
    CLK => CLK,
    RST => RST,
    RXD => internal_tx
);

end Structural;

```

- Sistema_A

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Sistema_A is
Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    START : in STD_LOGIC;
    TXD : out STD_LOGIC
);
end Sistema_A;

architecture structural of Sistema_A is
    signal en_count, tmp_read : STD_LOGIC;
    signal tmp_count : STD_LOGIC_VECTOR(2 downto 0);
    signal tmp_data : STD_LOGIC_VECTOR(7 downto 0);
    signal dabout_tmp : STD_LOGIC_VECTOR(7 downto 0);

```

```

signal UART_TBE : STD_LOGIC;
signal UART_WR : STD_LOGIC;

component ROM
  Port(
    clk : in STD_LOGIC;
    read : in STD_LOGIC;
    addr : in STD_LOGIC_VECTOR(2 downto 0);
    data : out STD_LOGIC_VECTOR(7 downto 0)
  );
end component;

component counter
  Generic (
    MAX_VALUE : integer;
    DIM : integer
  );
  Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    SET : in STD_LOGIC;
    SET_VALUE : in STD_LOGIC_VECTOR(2 downto 0);
    COUNT : out STD_LOGIC_VECTOR(2 downto 0);
    CARRY : out STD_LOGIC
  );
end component;

component Rs232RefComp
  port (
    TXD : out STD_LOGIC := '1';
    RXD : in STD_LOGIC;
    CLK : in STD_LOGIC;
    DBIN : in STD_LOGIC_VECTOR (7 downto 0);
    DBOUT : out STD_LOGIC_VECTOR (7 downto 0);
    RDA : inout STD_LOGIC;
    TBE : inout STD_LOGIC := '1';
    RD : in STD_LOGIC;
    WR : in STD_LOGIC;
    PE : out STD_LOGIC;
    FE : out STD_LOGIC;
    OE : out STD_LOGIC;
    RST : in STD_LOGIC := '0'
  );
end component;

component UCA
  port(
    start: in STD_LOGIC;
    clk: in STD_LOGIC;
    rst: in STD_LOGIC;
  
```

```

addr: in STD_LOGIC_VECTOR(2 downto 0);
tbe: in STD_LOGIC;
read: out STD_LOGIC;
wr: out STD_LOGIC;
en_cont: out STD_LOGIC
);
end component;
begin

count : counter
Generic Map(
  MAX_VALUE => 7,
  DIM => 3
)
Port Map(
  CLK => CLK,
  RST => RST,
  EN => en_count,
  SET => '0',
  SET_VALUE => "000",
  COUNT => tmp_count,
  CARRY => OPEN
);
memoria : ROM
Port Map(
  clk => CLK,
  read => tmp_read,
  addr => tmp_count,
  data => tmp_data
);

UART_A: Rs232RefComp
port map (
  TXD => TXD,
  RXD => '0',
  CLK => CLK,
  DBIN => tmp_data,
  DBOUT => dbout_tmp,
  TBE => UART_TBE,
  RD => '0',
  WR => UART_WR,
  RST => RST
);
control_unit : UCA
port map(
  start => START,
  clk => CLK,
  rst => RST,
  addr => tmp_count,
  tbe => UART_TBE,

```

```

    read => tmp_read,
    wr => UART_WR,
    en_cont => en_count
  );

```

end structural;

- ROM

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ROM is
  Port(
    clk : in STD_LOGIC;
    read : in STD_LOGIC;
    addr : in STD_LOGIC_VECTOR(2 downto 0);
    data : out STD_LOGIC_VECTOR(7 downto 0)
  );
end ROM;

architecture Behavioral of ROM is
  signal data_reg : STD_LOGIC_VECTOR(7 downto 0);

  type rom_type is array (0 to 7) of std_logic_vector(7 downto 0);
  constant ROM : rom_type := (
    "00000001",
    "00000010",
    "00000011",
    "00000100",
    "00000101",
    "00000110",
    "00000111",
    "00001000");

begin
  process(clk)
  begin
    if rising_edge(clk) then
      if READ = '1' then
        -- Legge il valore dalla ROM all'indirizzo specificato
        data_reg <= ROM(to_integer(unsigned(addr)));
      end if;
    end if;
  end process;

  data <= data_reg;
end Behavioral;

```

- UCA

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity UCA is
  port(
    start: in STD_LOGIC;
    clk: in STD_LOGIC;
    rst: in STD_LOGIC;
    addr: in STD_LOGIC_VECTOR(2 downto 0);
    tbe: in STD_LOGIC;
    read: out STD_LOGIC;
    wr: out STD_LOGIC;
    en_cont: out STD_LOGIC
  );
end UCA;

architecture Behavioral of UCA is
  type stato is (IDLE, S1, S2, S3, S4);

  signal stato_corrente : stato := IDLE;
  signal stato_prossimo : stato;

begin
  -- PARTE COMBINATORIA:
  stato_uscita : process(stato_corrente, start, addr, tbe)
  begin
    read <= '0';
    wr <= '0';
    en_cont <= '0';

    case stato_corrente is
      when IDLE =>
        if(start = '0') then
          stato_prossimo <= IDLE;
        else
          stato_prossimo <= S1;
        end if;

      when S1 =>
        read <= '1';
        stato_prossimo <= S2;

      when S2 =>
        wr <= '1';
        stato_prossimo <= S3;
    end case;
  end process;
end;

```

```

when S3 =>
  if(tbe = '0') then
    stato_prossimo <= S3;
  else
    stato_prossimo <= S4;
  end if;

when S4 =>
  en_cont <= '1';
  if (addr /= "111") then
    stato_prossimo <= S1;
  else
    stato_prossimo <= IDLE;
  end if;
end case;
end process;

```

-- PARTE SEQUENZIALE:

```

mem: process(clk)
begin
  if(clk'event AND clk = '1') then
    if(rst = '1') then
      stato_corrente <= IDLE;
    else
      stato_corrente <= stato_prossimo;
    end if;
  end if;
  end process;
end behavioral;

```

- Sistema B

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Sistema_B is
  Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    RXD : in STD_LOGIC
  );
end Sistema_B;

architecture structural of Sistema_B is
  signal en_count, tmp_write : STD_LOGIC;
  signal tmp_count : STD_LOGIC_VECTOR(2 downto 0);
  signal tmp_data : STD_LOGIC_VECTOR(7 downto 0);
  signal txd_temp : STD_LOGIC := '0';

```

```

signal UART_RDA : STD_LOGIC;
signal UART_RD : STD_LOGIC;

component counter
  Generic (
    MAX_VALUE : integer;
    DIM : integer
  );
  Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    SET : in STD_LOGIC;
    SET_VALUE : in STD_LOGIC_VECTOR(2 downto 0);
    COUNT : out STD_LOGIC_VECTOR(2 downto 0);
    CARRY : out STD_LOGIC
  );
end component;

component MEM
  Port(
    clk : in STD_LOGIC;
    read : in STD_LOGIC;
    write : in STD_LOGIC;
    data_in : in STD_LOGIC_VECTOR(7 downto 0);
    addr : in STD_LOGIC_VECTOR(2 downto 0);
    data_out : out STD_LOGIC_VECTOR(7 downto 0)
  );
end component;

component Rs232RefComp
  port (
    TxD : out STD_LOGIC := '1';
    RxD : in STD_LOGIC;
    CLK : in STD_LOGIC;
    DBIN : in STD_LOGIC_VECTOR (7 downto 0);
    DBOUT : out STD_LOGIC_VECTOR (7 downto 0);
    RDA : inout STD_LOGIC;
    TBE : inout STD_LOGIC := '1';
    RD : in STD_LOGIC;
    WR : in STD_LOGIC;
    PE : out STD_LOGIC;
    FE : out STD_LOGIC;
    OE : out STD_LOGIC;
    RST : in STD_LOGIC := '0'
  );
end component;

component UCB
  Port (
    clk: in STD_LOGIC;
    rst: in STD_LOGIC;
  
```

```

    rda: in STD_LOGIC;
    rxd: in STD_LOGIC;
    write: out STD_LOGIC;
    rd: out STD_LOGIC;
    en_cont: out STD_LOGIC
);
end component;

```

begin

```

control_unit : UCB
Port Map(
    clk => CLK,
    rst => RST,
    rda => UART_RDA,
    rxd => RXD,
    write => tmp_write,
    rd => UART_RD,
    en_cont => en_count
);

```

UART_B: Rs232RefComp

```

port map(
    TxD => txd_temp,
    RxD => RXD,
    CLK => CLK,
    DBIN => (others => '0'),
    DBOUT => tmp_data,
    RDA => UART_RDA,
    RD => UART_RD,
    WR => '0',
    RST => RST
);

```

memoria : MEM

```

Port Map(
    clk => CLK,
    read => '0',
    write => tmp_write,
    data_in => tmp_data,
    addr => tmp_count,
    data_out => open
);

```

count : counter

```

Generic Map(
    MAX_VALUE => 7,
    DIM => 3
)
Port Map(
    CLK => CLK,
    RST => RST,

```

```

EN => en_count,
SET => '0',
SET_VALUE => "000",
COUNT => tmp_count,
CARRY => OPEN
);
end structural;

```

- MEM

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MEM is
Port(
    clk : in STD_LOGIC;
    read : in STD_LOGIC;
    write : in STD_LOGIC;
    data_in : in STD_LOGIC_VECTOR(7 downto 0);
    addr : in STD_LOGIC_VECTOR(2 downto 0);
    data_out : out STD_LOGIC_VECTOR(7 downto 0)
);
end MEM;

architecture Behavioral of MEM is
-- Definizione della memoria come array di N locazioni di M bit
type memory_type is array (0 to 7) of STD_LOGIC_VECTOR(7 downto 0);

signal mem : memory_type := (others => "00000000"); -- Inizializzazione della memoria

begin
process(clk)
begin
    if rising_edge(clk) then
        if write = '1' then
            mem(CONV_INTEGER(addr)) <= data_in;
        elsif read = '1' then
            data_out <= mem(CONV_INTEGER(addr));
        end if;
    end if;
end process;
end Behavioral;

```

- UCB

```
library IEEE;
```

```

use IEEE.STD_LOGIC_1164.ALL;

entity UCB is
  Port (
    clk: in STD_LOGIC;
    rst: in STD_LOGIC;
    rda: in STD_LOGIC;
    rxd: in STD_LOGIC;
    write: out STD_LOGIC;
    rd: out STD_LOGIC;
    en_cont: out STD_LOGIC
  );
end UCB;

architecture Behavioral of UCB is
  type stato is (S0, S1, S2, S3);

  signal stato_corrente : stato := S0;
  signal stato_prossimo : stato;

begin
  -- PARTE COMBINATORIA:
  stato_uscita : process(stato_corrente, rda, rxd)
  begin
    write <= '0';
    en_cont <= '0';
    rd <= '0';

    case stato_corrente is
      when S0 =>
        rd <= '1';
        if (rxd = '1') then
          stato_prossimo <= S0;
        else
          stato_prossimo <= S1;
        end if;
      when S1 =>
        if (RDA = '0') then
          stato_prossimo <= S1;
        else
          stato_prossimo <= S2;
        end if;
      when S2 =>
        write <= '1';
        stato_prossimo <= S3;
      when S3 =>
        en_cont <= '1';
        stato_prossimo <= S0;
    end case;
  end process;

  -- PARTE SEQUENZIALE:

```

```

mem: process(clk)
begin
  if(clk'event AND clk = '1') then
    if(rst = '1') then
      stato_corrente <= S0;
    else
      stato_corrente <= stato_prossimo;
    end if;
  end if;
end process;

end Behavioral;

```

Simulazione

Il seguente testbench simula il funzionamento del modulo Sistema_UART, generando un segnale di clock periodico e applicando stimoli ai segnali START e RST. Dopo un reset iniziale, il segnale START viene attivato e poi disattivato per verificare il comportamento del sistema.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity Sistema_UART_tb is
end Sistema_UART_tb;

architecture testbench of Sistema_UART_tb is

-- Component under test (CUT)
component Sistema_UART
  Port (
    START : in STD_LOGIC;
    RST : in STD_LOGIC;
    CLK : in STD_LOGIC
  );
end component;

-- Signal declarations
signal START_tb : STD_LOGIC := '0';
signal RST_tb : STD_LOGIC := '0';
signal CLK_tb : STD_LOGIC := '0';

-- Clock period definition

```

```

constant CLK_PERIOD : time := 1 ns;

begin

    -- Instantiate the CUT
    uut: Sistema_UART
    port map (
        START => START_tb,
        RST => RST_tb,
        CLK => CLK_tb
    );

    -- Clock process
    CLK_process: process
    begin
        while true loop
            CLK_tb <= '0';
            wait for CLK_PERIOD / 2;
            CLK_tb <= '1';
            wait for CLK_PERIOD / 2;
        end loop;
        wait;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        -- Reset the system
        RST_tb <= '1';
        wait for 20 ns;
        RST_tb <= '0';

        -- Apply START signal
        wait for 30 ns;
        START_tb <= '1';
        wait for 40 ns;
        START_tb <= '0';

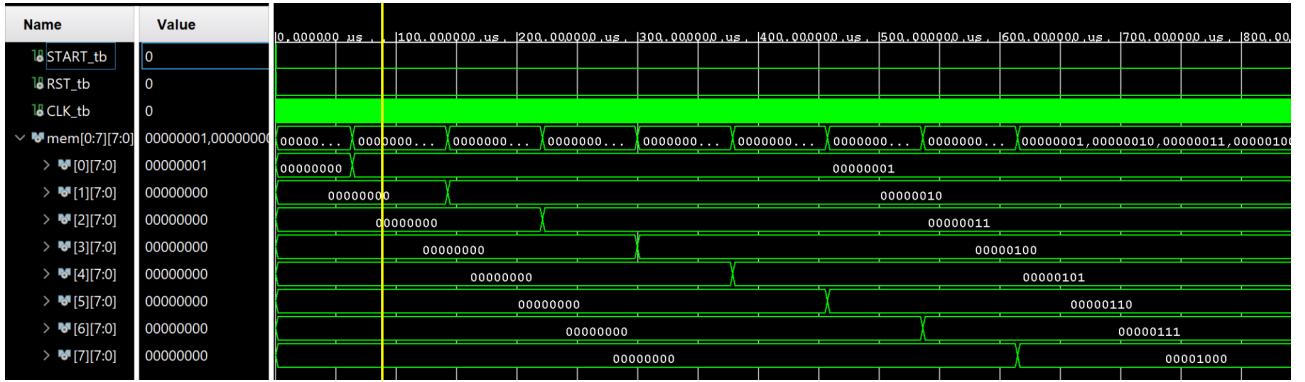
        -- More test cases can be added here

        wait;
    end process;

```

end testbench;

L'immagine sottostante mostra il risultato della simulazione, evidenziando come i dati ricevuti vengano progressivamente scritti nella memoria del sistema ricevente.



Capitolo 7: Switch Multistadio

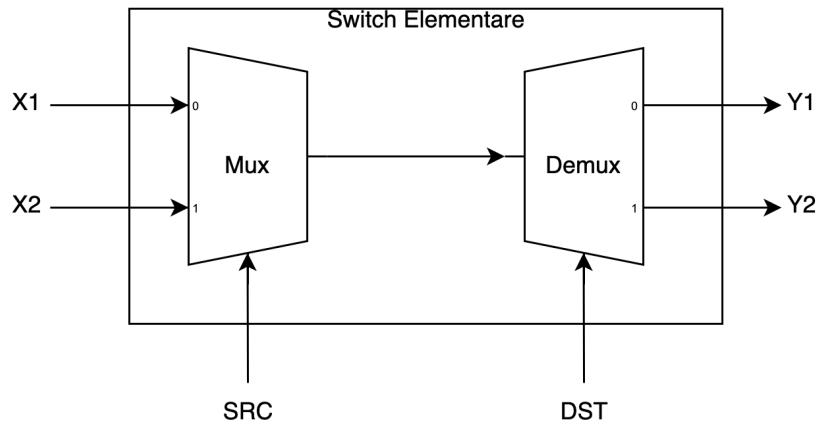
Esercizio 11: Switch Multistadio

Esercizio 11.1

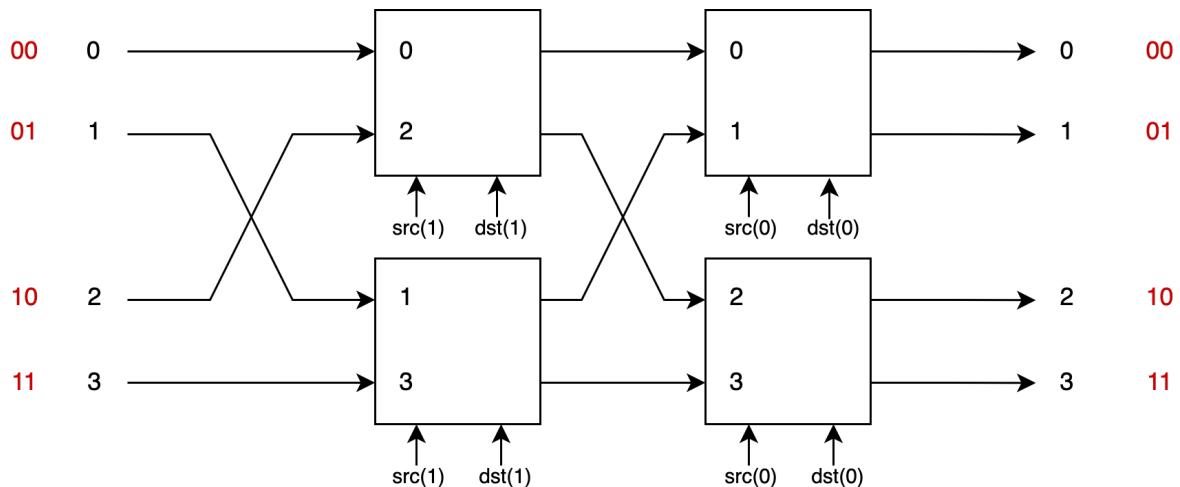
Progettare ed implementare in VHDL uno switch multistadio secondo il modello omega network. Lo switch deve consentire lo scambio di messaggi di 2 bit ciascuno da un nodo sorgente a un nodo destinazione in una rete con 4 nodi, implementando uno schema a priorità fissa fra i nodi (es. nodo 1 più prioritario,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

Progetto e architettura

Per la realizzazione dello switch multistadio basato sul modello Omega Network, è stato seguito un approccio modulare, sviluppando inizialmente i componenti base necessari per la costruzione della rete di interconnessione. In particolare, sono stati implementati un multiplexer 2:1 ed un demultiplexer 1:2 per realizzare lo switch elementare, che rappresenta il blocco fondamentale della rete. La rappresentazione grafica dell'architettura dello switch elementare è riportata in basso:



La rete è stata quindi costruita collegando più switch elementari in modo da garantire il corretto instradamento dei messaggi tra i nodi sorgente e destinazione, seguendo lo schema in basso:



Implementazione

Capitolo 8: Prova Dicembre

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

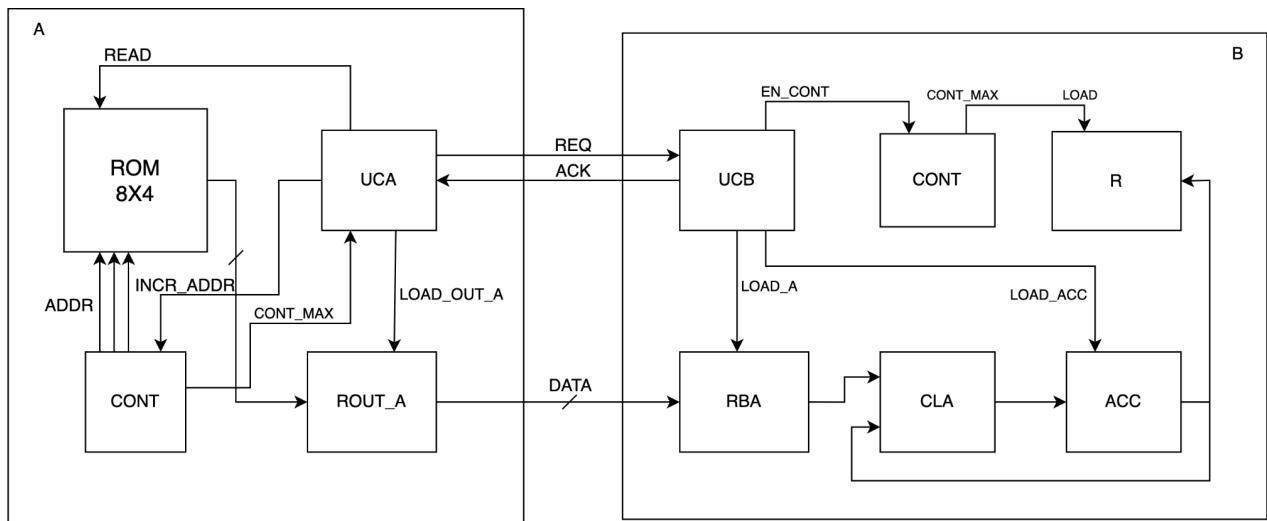
operativa e parte di controllo di ciascun nodo. Ogni nodo deve essere progettato seguendo un approccio strutturale, individuando tutti i componenti, le loro interfacce e le loro interconnessioni.

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

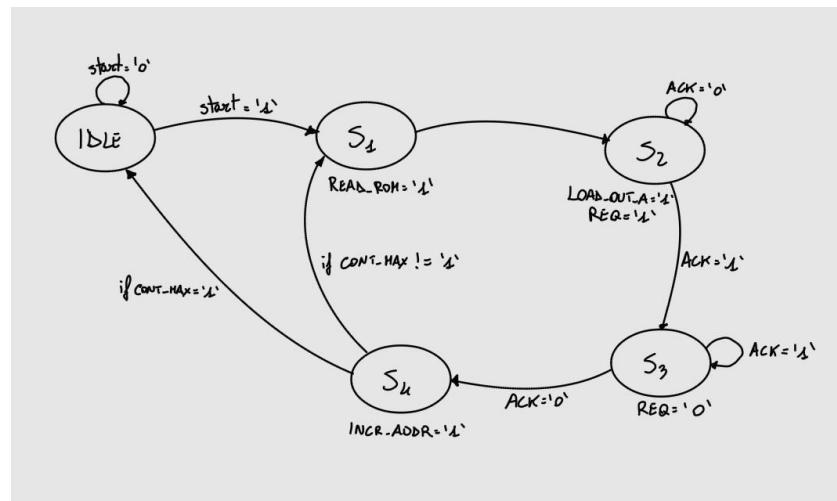
Riportiamo innanzitutto l'architettura complessiva del sistema, ottenuta usando un diagramma a blocchi come da specifica. Si noti che '**UCA**' e '**UCB**' rappresentano, rispettivamente, la parte di controllo del nodo A e del nodo B:



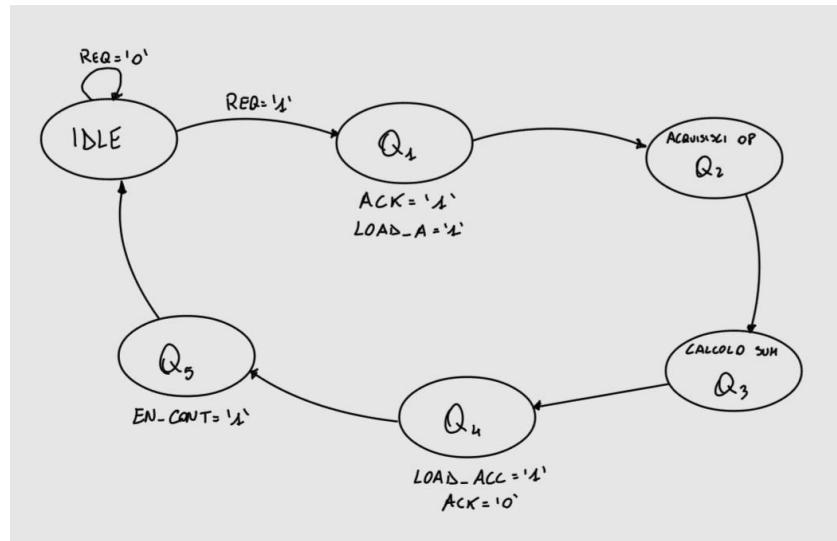
L'architettura del sommatore utilizzato dal nodo B, implementato mediante Carry Look Ahead (come da specifica) è riportata in appendice.

Di seguito sono invece riportati gli automi per le unità di controllo dei nodi, ipotizzando che le macchine lavorino sul fronte di salita del clock ed il READ/LOAD siano sincroni:

- AUTOMA DI A



- AUTOMA DI B



Implementazione

Di seguito viene riportato il codice VHDL del sistema complessivo (il codice del sommatore CLA, del contatore e dei registri è riportato in appendice):

- Sistema Complessivo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Sistema is
  Port (
    CLK_A : in STD_LOGIC;
    CLK_B : in STD_LOGIC;
    RST : in STD_LOGIC;
    START : in STD_LOGIC;
    DATA_OUT : out STD_LOGIC_VECTOR(3 downto 0)
  );
end Sistema;

architecture Structural of Sistema is

  signal tmp_ack, tmp_req : STD_LOGIC;
  signal data_out_A : STD_LOGIC_VECTOR(3 downto 0);

  component NodoA
    Port (
      START : in STD_LOGIC;
      CLK_A : in STD_LOGIC;
      RST : in STD_LOGIC;
      ACK : in STD_LOGIC;
      REQ : out STD_LOGIC;
      DATA : out STD_LOGIC_VECTOR(3 downto 0)
    );
  end component;

```

```

end component;

component NodoB
  Port (
    CLK_B : in STD_LOGIC;
    RST : in STD_LOGIC;
    REQ : in STD_LOGIC;
    DATA_IN : in STD_LOGIC_VECTOR(3 downto 0);
    ACK : out STD_LOGIC;
    DATA_OUT : out STD_LOGIC_VECTOR(3 downto 0)
  );
end component;

begin

  A : NodoA
    Port Map(
      START => START,
      CLK_A => CLK_A,
      RST => RST,
      ACK => tmp_ack,
      REQ => tmp_req,
      DATA => data_out_A
    );

  B : NodoB
    Port Map(
      CLK_B => CLK_B,
      RST => RST,
      REQ => tmp_req,
      DATA_IN => data_out_A,
      ACK => tmp_ack,
      DATA_OUT => DATA_OUT
    );

end Structural;

```

- Nodo A

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NodoA is
  Port (
    START : in STD_LOGIC;
    CLK_A : in STD_LOGIC;
    RST : in STD_LOGIC;
    ACK : in STD_LOGIC;
    REQ : out STD_LOGIC;
    DATA : out STD_LOGIC_VECTOR(3 downto 0)
  )

```

```

);
end NodoA;

architecture Structural of NodoA is

signal en_count, tmp_read, tmp_load, cont_max : STD_LOGIC;
signal tmp_count : STD_LOGIC_VECTOR(2 downto 0);
signal tmp_data : STD_LOGIC_VECTOR(3 downto 0);

component ROM
Port(
    clk : in STD_LOGIC;
    read : in STD_LOGIC;
    addr : in STD_LOGIC_VECTOR(2 downto 0);
    data : out STD_LOGIC_VECTOR(3 downto 0)
);
end component;

component counter
Generic (
    MAX_VALUE : integer;
    DIM : integer
);
Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    SET : in STD_LOGIC;
    SET_VALUE : in STD_LOGIC_VECTOR(2 downto 0);
    COUNT : out STD_LOGIC_VECTOR(2 downto 0);
    CARRY : out STD_LOGIC
);
end component;

component Registro
Port (
    D : in STD_LOGIC_VECTOR(3 downto 0);
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    Q : out STD_LOGIC_VECTOR(3 downto 0)
);
end component;

component UCA
Port (
    START : in STD_LOGIC;
    ACK : in STD_LOGIC;
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    CONT_MAX : in STD_LOGIC;
    REQ : out STD_LOGIC;

```

```

READ : out STD_LOGIC;
INCR_ADDR : out STD_LOGIC;
LOAD_OUT_A : out STD_LOGIC
);
end component;

begin

control_unit : UCA
  Port Map(
    START => START,
    ACK => ACK,
    CLK => CLK_A,
    RST => RST,
    CONT_MAX => cont_max,
    REQ => REQ,
    READ => tmp_read,
    INCR_ADDR => en_count,
    LOAD_OUT_A => tmp_load
  );

count : counter
  Generic Map(
    MAX_VALUE => 7,
    DIM => 3
  )
  Port Map(
    CLK => CLK_A,
    RST => RST,
    EN => en_count,
    SET => '0',
    SET_VALUE => "000",
    COUNT => tmp_count,
    CARRY => cont_max
  );

memoria : ROM
  Port Map(
    clk => CLK_A,
    read => tmp_read,
    addr => tmp_count,
    data => tmp_data
  );

reg : Registro
  Port Map(
    D => tmp_data,
    CLK => CLK_A,
    RST => RST,
    EN => tmp_load,
    Q => DATA
  );

```

end Structural;

- UCA

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

entity UCA is
  Port (
    START : in STD_LOGIC;
    ACK : in STD_LOGIC;
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    CONT_MAX : in STD_LOGIC;
    REQ : out STD_LOGIC;
    READ : out STD_LOGIC;
    INCR_ADDR : out STD_LOGIC;
    LOAD_OUT_A : out STD_LOGIC
  );
end UCA;

architecture Behavioral of UCA is
  type stato is (IDLE, S1, S2, S3, S4);

  signal stato_corrente : stato := IDLE;
  signal stato_prossimo : stato;

begin

  -- Parte Combinatoria
  stato_uscita : process(stato_corrente, START, CONT_MAX, ACK)
  begin
    REQ <= '0';
    READ <= '0';
    INCR_ADDR <= '0';
    LOAD_OUT_A <= '0';

    case stato_corrente is
      when IDLE =>
        if (START = '0') then
          stato_prossimo <= IDLE;
        else
          stato_prossimo <= S1;
        end if;

      when S1 =>
        READ <= '1';

      when S2 =>
        READ <= '0';
        INCR_ADDR <= '1';
        LOAD_OUT_A <= '1';

      when S3 =>
        READ <= '0';
        INCR_ADDR <= '0';
        LOAD_OUT_A <= '0';

      when S4 =>
        READ <= '0';
        INCR_ADDR <= '0';
        LOAD_OUT_A <= '0';
    end case;
  end process;
end Behavioral;
```

```

stato_prossimo <= S2;

when S2 =>
  REQ <= '1';
  LOAD_OUT_A <= '1';
  if (ACK = '0') then
    stato_prossimo <= S2;
  else
    stato_prossimo <= S3;
  end if;

when S3 =>
  if (ACK = '1') then
    stato_prossimo <= S3;
  else
    stato_prossimo <= S4;
  end if;

when S4 =>
  INCR_ADDR <= '1';
  if (CONT_MAX /= '1') then
    stato_prossimo <= S1;
  else
    stato_prossimo <= IDLE;
  end if;
end case;
end process;

-- Parte Sequenziale
mem: process (CLK)
begin
  if( CLK'event and CLK = '1' ) then
    if( RST = '1' ) then
      stato_corrente <= IDLE;
    else
      stato_corrente <= stato_prossimo;
    end if;
  end if;
end process;
end Behavioral;

```

- ROM

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ROM is
  Port(
    clk : in STD_LOGIC;
    read : in STD_LOGIC;

```

```

addr : in STD_LOGIC_VECTOR(2 downto 0);
data : out STD_LOGIC_VECTOR(3 downto 0)
);
end ROM;

architecture Behavioral of ROM is
  signal data_out : STD_LOGIC_VECTOR(3 downto 0);
begin
  process(clk)
  begin
    if rising_edge(clk) then
      if read = '1' then
        case addr is
          when "000" => data_out <= "0001"; -- Locazione 0
          when "001" => data_out <= "0010"; -- Locazione 1
          when "010" => data_out <= "0100"; -- Locazione 2
          when "011" => data_out <= "1000"; -- Locazione 3
          when "100" => data_out <= "1001"; -- Locazione 4
          when "101" => data_out <= "1010"; -- Locazione 5
          when "110" => data_out <= "1100"; -- Locazione 6
          when "111" => data_out <= "1111"; -- Locazione 7
          when others => data_out <= "0000";
        end case;
      end if;
    end if;
  end process;
  data <= data_out;
end Behavioral;

```

- Nodo B

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NodoB is
  Port (
    CLK_B : in STD_LOGIC;
    RST : in STD_LOGIC;
    REQ : in STD_LOGIC;
    DATA_IN : in STD_LOGIC_VECTOR(3 downto 0);
    ACK : out STD_LOGIC;
    DATA_OUT : out STD_LOGIC_VECTOR(3 downto 0)
  );
end NodoB;

architecture Structural of NodoB is

  signal tmp_load_A, tmp_load_acc, tmp_en_count, tmp_count_max : STD_LOGIC;
  signal reg_a_out, CLA_out, acc_out : STD_LOGIC_VECTOR(3 downto 0);

```

```

component Registro
  Port (
    D : in STD_LOGIC_VECTOR(3 downto 0);
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    Q : out STD_LOGIC_VECTOR(3 downto 0)
  );
end component;

component counter
  Generic (
    MAX_VALUE : integer;
    DIM : integer
  );
  Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    SET : in STD_LOGIC;
    SET_VALUE : in STD_LOGIC_VECTOR(2 downto 0);
    COUNT : out STD_LOGIC_VECTOR(2 downto 0);
    CARRY : out STD_LOGIC
  );
end component;

component SOMMATORE_CLA
  Port (
    X : in STD_LOGIC_VECTOR(3 downto 0);
    Y : in STD_LOGIC_VECTOR(3 downto 0);
    CIN : in STD_LOGIC;
    S : out STD_LOGIC_VECTOR(3 downto 0);
    COUT : out STD_LOGIC
  );
end component;

component UCB
  Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    REQ : in STD_LOGIC;
    COUNT_MAX : in STD_LOGIC;
    ACK : out STD_LOGIC;
    EN_COUNT : out STD_LOGIC;
    LOAD_ACC : out STD_LOGIC;
    LOAD_A : out STD_LOGIC
  );
end component;

begin

```

```

control_unit : UCB
Port Map(
    CLK => CLK_B,
    RST => RST,
    REQ => REQ,
    COUNT_MAX => tmp_count_max,
    ACK => ACK,
    EN_COUNT => tmp_en_count,
    LOAD_ACC => tmp_load_acc,
    LOAD_A => tmp_load_A
);

```

```

Sum : SOMMATORE_CLA
Port Map(
    X => reg_a_out,
    Y => acc_out,
    CIN => '0',
    S => CLA_out,
    COUT => OPEN
);

```

```

cont : counter
Generic Map(
    MAX_VALUE => 7,
    DIM => 3
)
Port Map(
    CLK => CLK_B,
    RST => RST,
    EN => tmp_en_count,
    SET => '0',
    SET_VALUE => "000",
    COUNT => OPEN,
    CARRY => tmp_count_max
);

```

```

Rba : Registro
Port Map(
    D => Data_in,
    CLK => CLK_B,
    RST => RST,
    EN => tmp_load_A,
    Q=> reg_a_out
);

```

```

ACC : Registro
Port Map(
    D => CLA_out,
    CLK => CLK_B,
    RST => RST,
    EN => tmp_load_acc,
    Q=> acc_out
);

```

```

);
R : Registro
Port Map(
    D => acc_out,
    CLK => CLK_B,
    RST => RST,
    EN => tmp_count_max,
    Q => Data_out
);
end Structural;

```

- UCB

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity UCB is
Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    REQ : in STD_LOGIC;
    COUNT_MAX : in STD_LOGIC;
    ACK : out STD_LOGIC;
    EN_COUNT : out STD_LOGIC;
    LOAD_ACC : out STD_LOGIC;
    LOAD_A : out STD_LOGIC
);
end UCB;

architecture Behavioral of UCB is

    type stato is (IDLE, Q1, Q2, Q3, Q4, Q5);

    signal stato_corrente : stato := IDLE;
    signal stato_prossimo : stato;

    begin

        -- Parte Combinatoria
        stato_uscita : process(stato_corrente, req, count_max)
        begin
            ACK <= '0';
            EN_COUNT <= '0';
            LOAD_ACC <= '0';
            LOAD_A <= '0';

            case stato_corrente is

```

```

when IDLE =>
  if (REQ = '0') then
    stato_prossimo <= IDLE;
  else
    stato_prossimo <= Q1;
  end if;

when Q1 =>
  LOAD_A <= '1';
  ACK <= '1';
  stato_prossimo <= Q2;

when Q2 =>
  --acquisizione operandi
  ACK <= '1';
  LOAD_A <= '1';
  stato_prossimo <= Q3;

when Q3 =>
  --calcolo somma
  ACK <= '1';
  stato_prossimo <= Q4;

when Q4 =>
  LOAD_ACC <= '1';
  stato_prossimo <= Q5;

when Q5 =>
  EN_COUNT <= '1';
  stato_prossimo <= IDLE;
end case;
end process;

-- Parte Sequenziale
mem: process (CLK)
begin
  if( CLK'event and CLK = '1' ) then
    if( RST = '1' ) then
      stato_corrente <= IDLE;
    else
      stato_corrente <= stato_prossimo;
    end if;
  end if;
end process;

end Behavioral;

```

Simulazione

Il seguente testbench verifica il funzionamento del sistema. Vengono generati due segnali di clock differenti per il nodo A ed il nodo B. Inizialmente, viene applicato un reset, poi viene attivato il segnale START per avviare la comunicazione tra i due nodi. Dopo un periodo di attesa per consentire il completamento della trasmissione, la simulazione termina.

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

entity Sistema_tb is
end Sistema_tb;

architecture testbench of Sistema_tb is

signal CLK_A : STD_LOGIC := '0';
signal CLK_B : STD_LOGIC := '0';
signal RST : STD_LOGIC := '1';
signal START : STD_LOGIC := '0';
signal DATA_OUT : STD_LOGIC_VECTOR(3 downto 0);

constant CLK_A_PERIOD : time := 20 ns;
constant CLK_B_PERIOD : time := 10 ns;

component Sistema
Port (
    CLK_A : in STD_LOGIC;
    CLK_B : in STD_LOGIC;
    RST : in STD_LOGIC;
    START : in STD_LOGIC;
    DATA_OUT : out STD_LOGIC_VECTOR(3 downto 0)
);
end component;

begin

UUT: Sistema
port map (
    CLK_A => CLK_A,
    CLK_B => CLK_B,
    RST => RST,
    START => START,
    DATA_OUT => DATA_OUT
);

-- Processo per generare il clock A
process
begin
    while true loop
        CLK_A <= '0';
        wait for CLK_A_PERIOD / 2;
        CLK_A <= '1';
        wait for CLK_B_PERIOD / 2;
    end loop;
end process;

end;
```

```

CLK_A <= '1';
wait for CLK_A_PERIOD / 2;
end loop;
end process;

-- Processo per generare il clock B
process
begin
while true loop
  CLK_B <= '0';
  wait for CLK_B_PERIOD / 2;
  CLK_B <= '1';
  wait for CLK_B_PERIOD / 2;
end loop;
end process;

-- Stimolo del testbench
process
begin
  -- Reset iniziale
  RST <= '1';
  wait for 50 ns;
  RST <= '0';

  -- Attivazione del segnale START
  wait for 20 ns;
  START <= '1';
  wait for 20 ns;
  START <= '0';

  -- Attendere per osservare il comportamento
  wait for 500 ns;

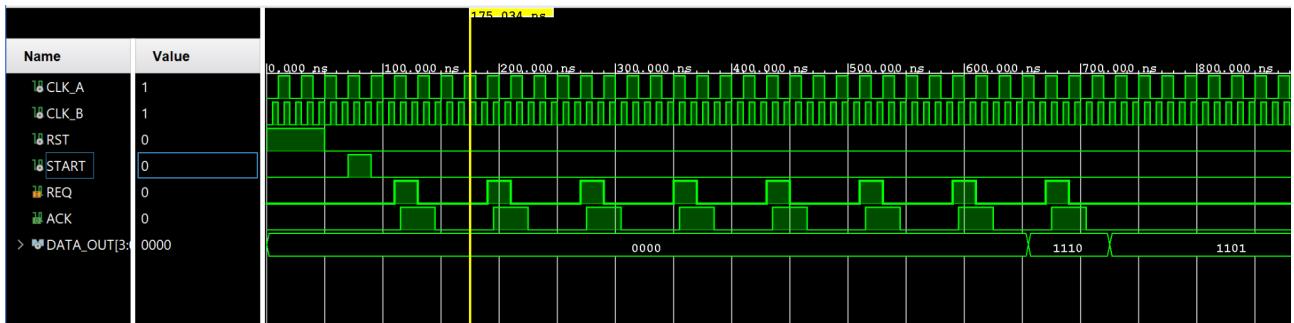
  -- Fine della simulazione
  wait;
end process;

end testbench;

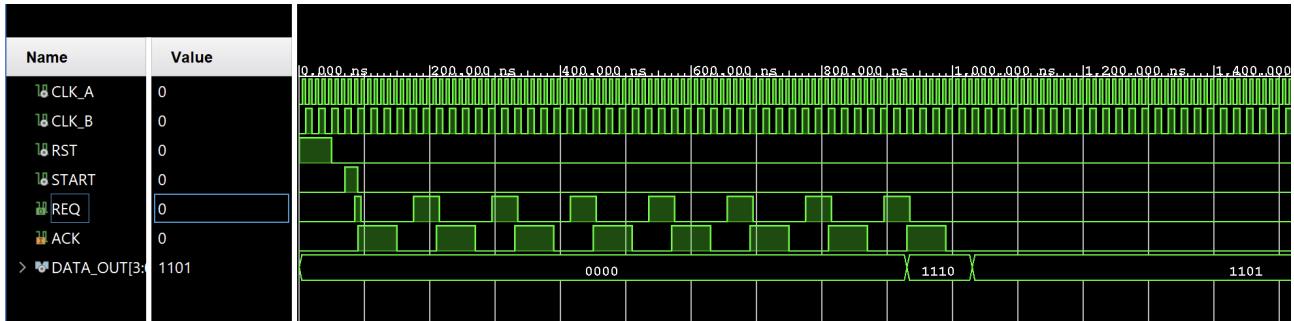
```

Come richiesto dalla traccia è stata effettuata la simulazione del sistema sia nel caso in cui il clock del nodo A sia più veloce di quello del nodo B, sia il caso inverso. I risultati della simulazione sono i seguenti:

- A più lento



- A più veloce



Il valore scritto nel registro R corrisponde al valore atteso, dato dalla somma degli elementi contenuti nella ROM ma rappresentato soltanto con i 4 bit meno significativi.

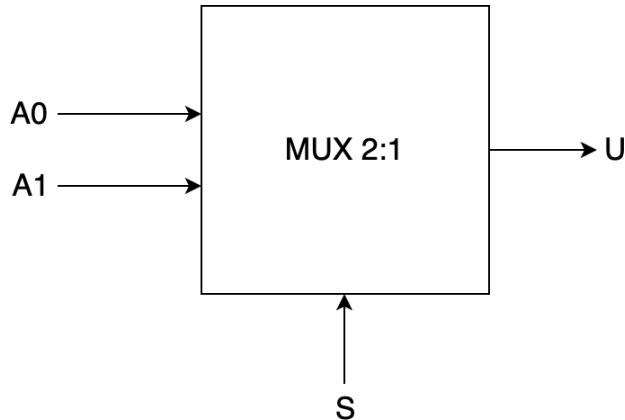
Appendice

Multiplexer 2:1

Progetto e architettura

Il multiplexer 2:1 è un circuito digitale che seleziona uno dei due ingressi, basandosi sul valore di un singolo bit di selezione. L'approccio utilizzato è basato su un'architettura di tipo “dataflow”, in cui l'uscita del sistema viene determinata in modo combinatorio a partire dai segnali di ingresso.

Di seguito viene riportato il disegno del componente:



Dal punto di vista funzionale, osserviamo che:

- Se il segnale di selezione s è '0', l'uscita U prende il valore dell'ingresso a0.
- Se il segnale di selezione s è '1', l'uscita U prende il valore dell'ingresso a1.
- Se il segnale di selezione s ha un valore indefinito (diverso da '0' o '1'), l'uscita U verrà impostata a un valore indefinito, rappresentato dal carattere '-'.

Implementazione

```
entity mux_2_1 is
  port(  a0  : in STD_LOGIC;
         a1  : in STD_LOGIC;
         s   : in STD_LOGIC;
         y   : out STD_LOGIC
      );
end mux_2_1;

architecture dataflow_v1 of mux_2_1 is

begin
  y  <=  a0 when s = '0' else
         a1 when s = '1' else
         '-'; --specifico cosa succede quando s non assume valore 0 o 1
end dataflow_v1;
```

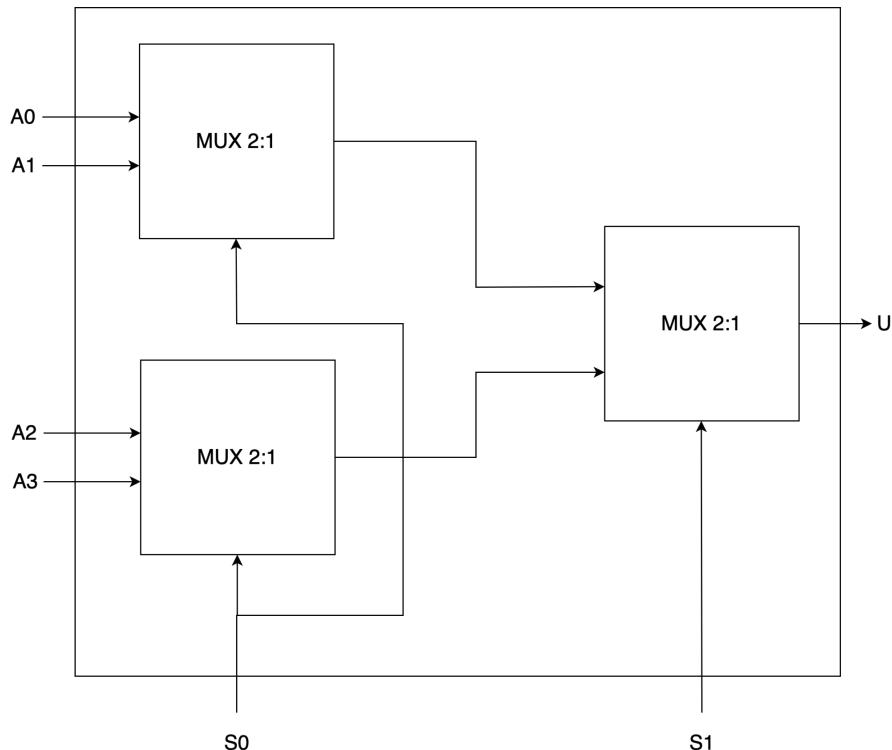
Multiplexer 4:1

Progetto e architettura

Il **multiplexer 4:1** ha la funzionalità di selezionare uno tra quattro ingressi (a_0, a_1, a_2, a_3) in base a due segnali di selezione (s_0 e s_1). L'uscita finale U dipende dai valori di questi segnali di selezione.

In questo caso, l'approccio utilizzato è stato strutturale: il mux è realizzato usando 3 mux 2:1, collegati opportunamente tra di loro.

Di seguito viene riportato il disegno della struttura:



Dal punto di vista funzionale, osserviamo che:

- Quando $s_0 = 0$ e $s_1 = 0$, l'uscita U deve essere uguale a a_0 .
- Quando $s_0 = 1$ e $s_1 = 0$, l'uscita U deve essere uguale a a_1 .
- Quando $s_0 = 0$ e $s_1 = 1$, l'uscita U deve essere uguale a a_2 .
- Quando $s_0 = 1$ e $s_1 = 1$, l'uscita U deve essere uguale a a_3 .

Implementazione

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

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;
```

```

end mux_4_1;

architecture structural of mux_4_1 is
  signal u0 : STD_LOGIC := '0';
  signal u1 : STD_LOGIC := '0';
  component mux_2_1
    port( a0 : in STD_LOGIC;
            a1 : in STD_LOGIC;
            s : in STD_LOGIC;
            y : out STD_LOGIC
    );
  end component;

begin

  mux0: mux_2_1
    Port map( a0 => a0,
                a1 => a1,
                s => s0,
                y => u0
    );
  mux1: mux_2_1
    Port map( a0 => a2,
                a1 => a3,
                s => s0,
                y => u1
    );
  mux2: mux_2_1
    Port map( a0 => u0,
                a1 => u1,
                s => s1,
                y => y
    );
end structural;

```

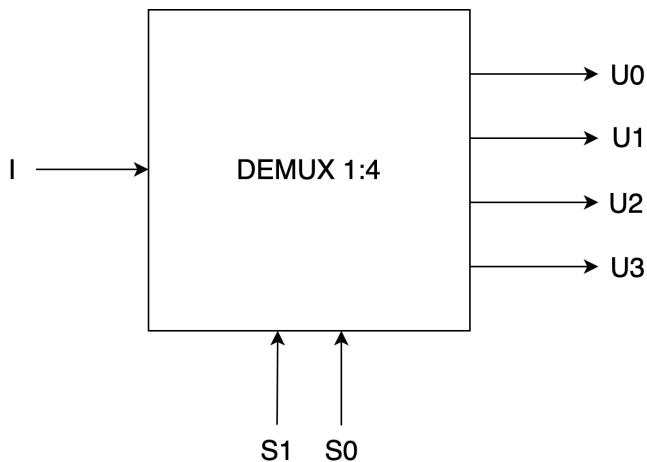
Demultiplexer 1:4

Progetto e architettura

Per la progettazione del demultiplexer (DEMUX) 1:4 è stato adottato un approccio comportamentale, mediante cui possiamo modellare il comportamento del circuito, specificandone il funzionamento in termini di risposta a ingressi e selettori. L'approccio utilizzato prevede:

- La concatenazione dei due segnali di selezione (s_0 e s_1) per formare un vettore a 2 bit (sel), che rappresenta direttamente l'indirizzo di selezione.
- Un processo sensibile al segnale di ingresso (I) e ai selettori (s_4 e s_5), in cui un'istruzione case determina a quale uscita instradare il segnale, azzerando le altre.

Di seguito viene riportato il disegno:



Implementazione

Di seguito viene riportato il codice VHDL del componente:

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

entity dmux_1_4 is
  Port ( c0 : in STD_LOGIC;
         s4 : in STD_LOGIC;
         s5 : in STD_LOGIC;
         d0 : out STD_LOGIC;
         d1 : out STD_LOGIC;
         d2 : out STD_LOGIC;
         d3 : out STD_LOGIC
      );
end dmux_1_4;

architecture Behavioral of dmux_1_4 is
  signal sel: std_logic_vector(1 downto 0);
begin
  sel <= s5 & s4;
  process(c0, s4, s5)
  begin
    d0 <= '0';
    if sel = "10" then
      d1 <= '1';
    elsif sel = "01" then
      d2 <= '1';
    elsif sel = "11" then
      d3 <= '1';
    end if;
  end process;
end Behavioral;
```

```

d1 <= '0';
d2 <= '0';
d3 <= '0';

case sel is
  when "00" =>
    d0 <= c0;
  when "01" =>
    d1 <= c0;
  when "10" =>
    d2 <= c0;
  when "11" =>
    d3 <= c0;
  when others =>
    d0 <= '0';
    d1 <= '0';
    d2 <= '0';
    d3 <= '0';
end case;

end process;

end Behavioral;

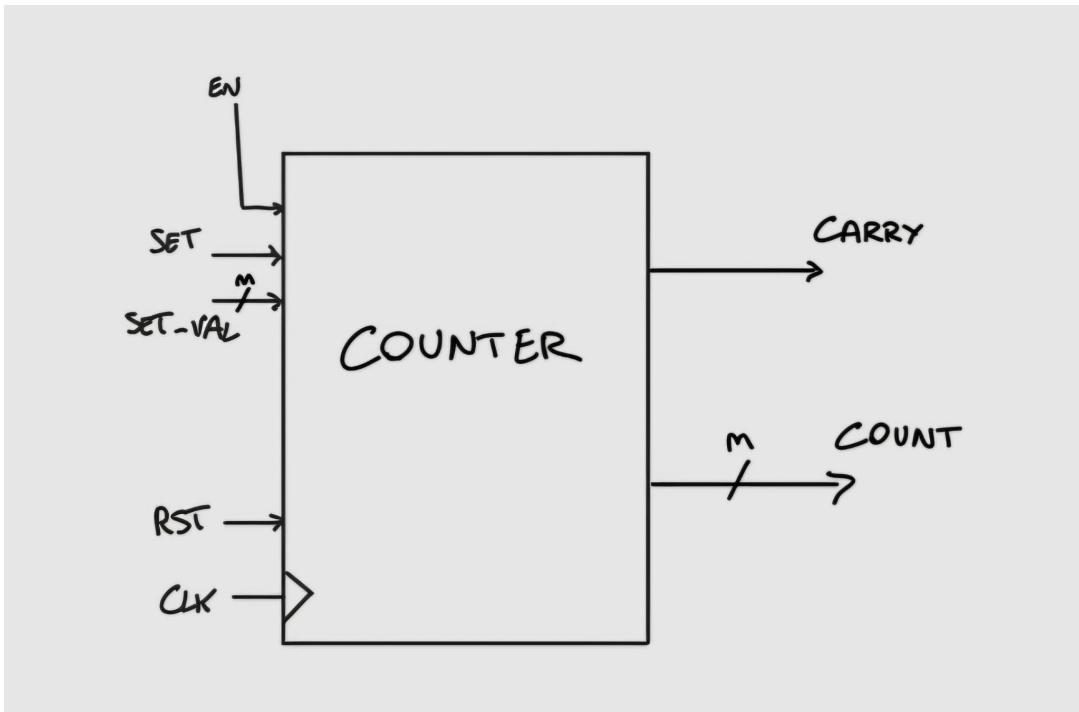
```

Progetto e architettura

L'architettura del contatore è basata su un registro che memorizza il valore corrente e una logica di incremento che permette di avanzare nel conteggio. Il contatore possiede le seguenti caratteristiche:

- **Conteggio solo in avanti:** il valore viene incrementato a ogni impulso di clock.
- **Inizializzazione programmabile:** è possibile impostare un valore iniziale tramite un segnale di ingresso.
- **Reset sincrono:** consente di riportare il valore del conteggio a zero.
- **Carry:** il contatore si resetta automaticamente e propaga il riporto quando raggiunge il valore massimo.

Di seguito viene riportato il disegno del componente:



Implementazione

Di seguito viene riportato il codice VHDL del componente:

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

entity counter is
  Generic (
    MAX_VALUE : integer := 59;
    DIM : integer := 6
  );
  Port (
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
```

```

EN :in STD_LOGIC;
SET :in STD_LOGIC;
SET_VALUE :in STD_LOGIC_VECTOR(DIM-1 downto 0);
COUNT :out STD_LOGIC_VECTOR(DIM-1 downto 0);
CARRY :out STD_LOGIC
);
end counter;

architecture Behavioral of counter is
signal count_tmp :STD_LOGIC_VECTOR(DIM-1 downto 0) := (others => '0');
begin
process(CLK) begin
if (RST = '1') then
count_tmp <= (others => '0');
elsif (CLK'event and CLK='1') then
if (SET='1') then
count_tmp <= SET_VALUE;
elsif (EN='1') then
if(count_tmp = CONV_STD_LOGIC_VECTOR(MAX_VALUE, DIM)) then
count_tmp <= (others => '0');
else
count_tmp <= count_tmp + 1;
end if;
end if;
end if;
end process;

COUNT <= count_tmp;
CARRY <= '1' when (count_tmp = CONV_STD_LOGIC_VECTOR(MAX_VALUE, DIM)) else '0';

end Behavioral;

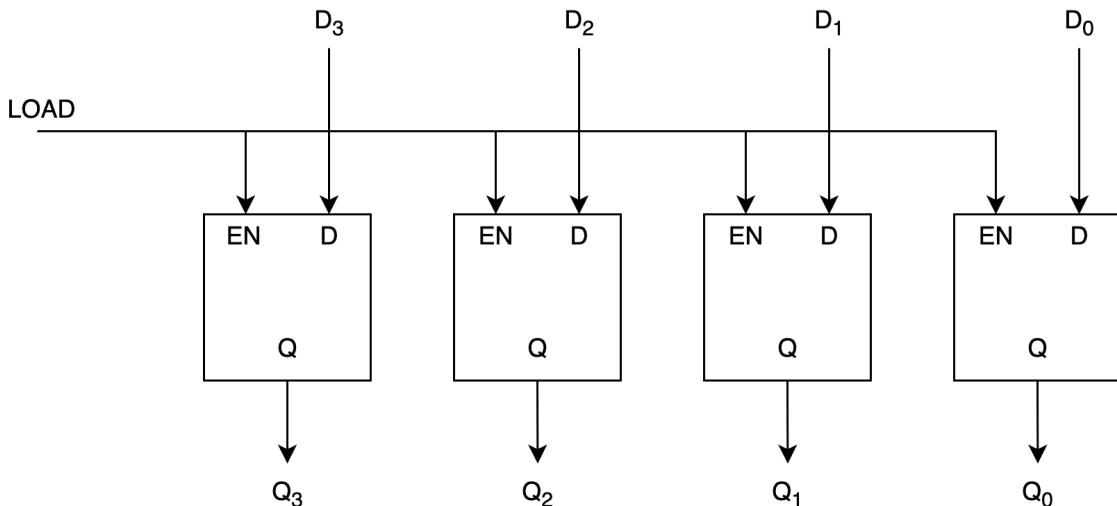
```

Registro ad N bit

Progetto e architettura

Il registro acquisisce il valore presente in ingresso (D) su un fronte di salita del clock (CLK), a condizione che il segnale di abilitazione (EN) sia attivo. Se il segnale di reset (RST) è alto, il registro viene azzerato indipendentemente dallo stato degli altri segnali. L'output del registro (Q) riflette il valore memorizzato.

In basso è rappresentato il disegno dell'architettura di un registro a 4 bit:



Implementazione

Di seguito viene riportato il codice VHDL del componente:

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

entity Registro is
  Port (
    D : in STD_LOGIC_VECTOR(3 downto 0);
    CLK : in STD_LOGIC;
    RST : in STD_LOGIC;
    EN : in STD_LOGIC;
    Q : out STD_LOGIC_VECTOR(3 downto 0)
  );
end Registro;

architecture Behavioral of Registro is
  signal Q_reg : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
begin
  begin
    process (CLK)
    begin
      if (CLK'event and CLK='1') then
        if (RST = '1') then
          Q_reg <= (others => '0');
        elsif (EN = '1') then
          Q_reg <= D;
        end if;
      end if;
    end process;
  end begin;
end Behavioral;
```

```
end process;  
  
Q <= Q_reg;  
  
end Behavioral;
```

Full Adder

Progetto e architettura

Il Full Adder è un circuito combinatorio che esegue l'addizione di due bit e un riporto in ingresso, producendo un bit di somma e un riporto in uscita.

Queste sono le equazioni logiche per il calcolo della somma e del riporto in uscita:

- Somma (s): $s = a \oplus b \oplus cin = a \oplus b \oplus cin$

- Riporto (cout): $\text{cout} = (\text{a} \cdot \text{b}) + (\text{cin} \cdot (\text{a} \oplus \text{b}))$

Implementazione

Di seguito viene riportato il codice VHDL del componente:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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;

```

Carry Look-Ahead

Progetto e architettura

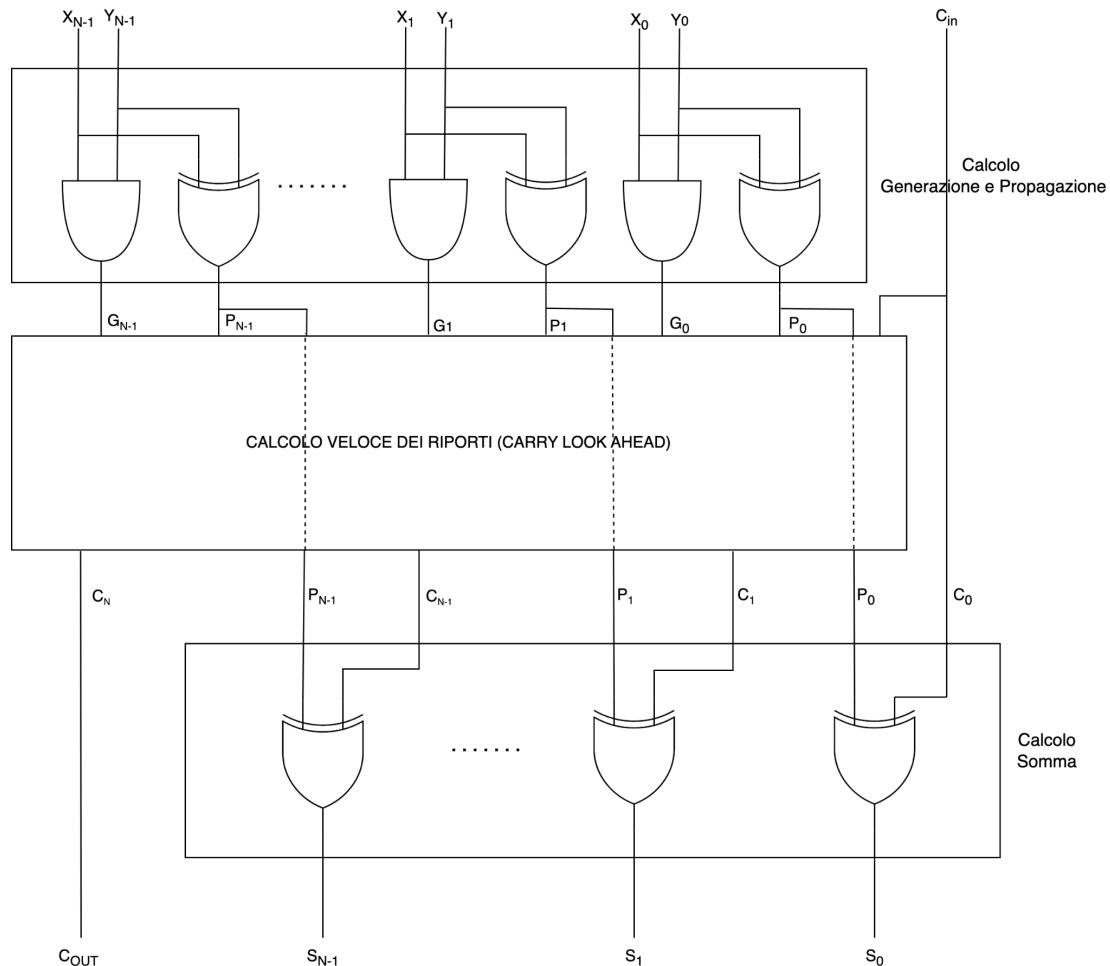
Il Carry Look Ahead Adder (CLA) è un sommatore progettato per eseguire operazioni di somma binaria in modo efficiente, riducendo il tempo di propagazione del riporto rispetto ai sommatori a propagazione di riporto classici.

L'idea principale è quella di precalcolare i riporti in base ai bit di input, invece di propagarli sequenzialmente.

Ipotizzando di avere in ingresso 2 stringhe da N bit ciascuna, indichiamo con **Cin** il carry in ingresso e con **Cout** il riporto finale. In sostanza, il sommatore si comporta in questo modo:

- Calcolo dei segnali di propagazione (P) e generazione (G).
- Determinazione dei riporti (C) direttamente invece di propagareli sequenzialmente.
- Calcolo della somma S usando P e C.

In basso è riportato un disegno della struttura del sommatore:



Implementazione

Di seguito viene riportato il codice VHDL (nel caso specifico con $N = 4$):

- CLA (top level)

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

```

entity SOMMATORE_CLA is
  Port (
    X : in STD_LOGIC_VECTOR(3 downto 0);
    Y : in STD_LOGIC_VECTOR(3 downto 0);
    CIN : in STD_LOGIC;
    S : out STD_LOGIC_VECTOR(3 downto 0);
    COUT : out STD_LOGIC
  );
end SOMMATORE_CLA;

architecture Structural of SOMMATORE_CLA is

  signal P_interno, G_interno : STD_LOGIC_VECTOR(3 downto 0);
  signal C_interno : STD_LOGIC_VECTOR(4 downto 0);

  component Calcola_G_P is
    Port (
      X : in STD_LOGIC_VECTOR(3 downto 0);
      Y : in STD_LOGIC_VECTOR(3 downto 0);
      P : out STD_LOGIC_VECTOR(3 downto 0);
      G : out STD_LOGIC_VECTOR(3 downto 0)
    );
  end component;

  component Calcola_Riporti is
    Port (
      G : in STD_LOGIC_VECTOR(3 downto 0);
      P : in STD_LOGIC_VECTOR(3 downto 0);
      Cin : in STD_LOGIC;
      C : out STD_LOGIC_VECTOR(4 downto 0)
    );
  end component;

  component Calcola_Somme is
    Port (
      P : in STD_LOGIC_VECTOR(3 downto 0);
      C : in STD_LOGIC_VECTOR(3 downto 0);
      S : out STD_LOGIC_VECTOR(3 downto 0)
    );
  end component;

begin

  Generate_Propagate : Calcola_G_P
    Port Map (
      X => X,
      Y => Y,
      P => P_interno,
      G => G_interno
    );

```

```
Riporti : Calcola_Riporti
```

```
Port Map (
```

```
    G => G_interno,  
    P => P_interno,  
    Cin => CIN,  
    C => C_interno  
);
```

```
Somme : Calcola_Somme
```

```
Port Map (
```

```
    P => P_interno,  
    C => C_interno(3 downto 0),  
    S => S  
);
```

```
COUT <= c_interno(4);
```

```
end Structural;
```

- Calcolo Generate e Propagate

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
  
entity Calcola_G_P is  
Port (  
    X : in STD_LOGIC_VECTOR(3 downto 0);  
    Y : in STD_LOGIC_VECTOR(3 downto 0);  
    P : out STD_LOGIC_VECTOR(3 downto 0);  
    G : out STD_LOGIC_VECTOR(3 downto 0)  
);  
end Calcola_G_P;
```

```
architecture Behavioral of Calcola_G_P is
```

```
begin
```

```
    G <= X and Y;  
    P <= X xor Y;
```

```
end Behavioral;
```

- Calcolo Riporti

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

```
entity Calcola_Riporti is  
Port (
```

```

G : in STD_LOGIC_VECTOR(3 downto 0);
P : in STD_LOGIC_VECTOR(3 downto 0);
Cin : in STD_LOGIC;
C : out STD_LOGIC_VECTOR(4 downto 0)
 );
end Calcola_Riporti;

architecture Behavioral of Calcola_Riporti is

begin

C(0) <= Cin;
C(1) <= G(0) or (P(0) and Cin);
C(2) <= G(1) or (P(1) and G(0)) or (P(1) and P(0) and Cin);
C(3) <= G(2) or (P(2) and G(1)) or (P(2) and P(1) and G(0)) or (P(2) and P(1) and P(0) and Cin);
C(4) <= G(3) or (P(3) and G(2)) or (P(3) and P(2) and G(1)) or (P(3) and P(2) and P(1) and G(0)) or (P(3)
and P(2) and P(1) and P(0) and Cin);

end Behavioral;

```

- Calcolo Somme

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Calcola_Somme is
Port (
    P : in STD_LOGIC_VECTOR(3 downto 0);
    C : in STD_LOGIC_VECTOR(3 downto 0);
    S : out STD_LOGIC_VECTOR(3 downto 0)
 );
end Calcola_Somme;

architecture Behavioral of Calcola_Somme is
begin

S <= P xor C;

end Behavioral;

```

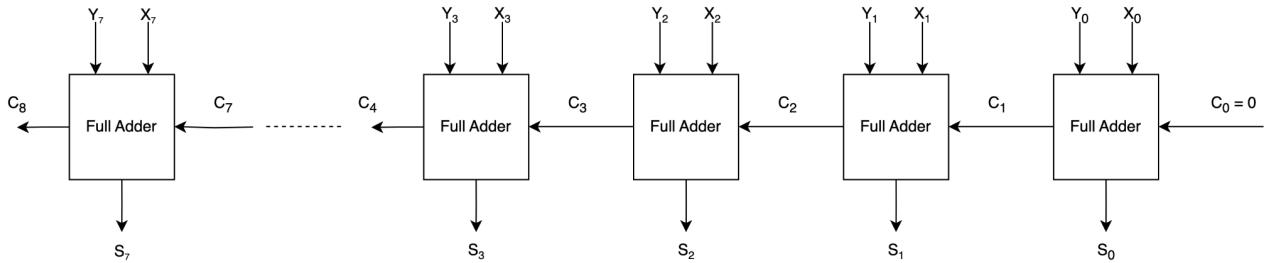
Ripple Carry Adder

Progetto e architettura

Il Ripple Carry Adder è un sommatore ad N bit costruito concatenando più full adder in cascata, dove ogni full adder somma due bit corrispondenti di X e Y, insieme al riporto del full adder precedente. Il riporto di

un full adder viene passato come carry-in al full adder successivo. Il ritardo totale dipende dal tempo impiegato per propagare il riporto attraverso tutti gli N full adder.

Il disegno seguente rappresenta un Ripple Carry Adder ad 8 bit:



Implementazione

Di seguito viene riportato il codice VHDL del componente:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ripple_carry is
  port( X, Y: in std_logic_vector(7 downto 0);
        c_in: in std_logic;
        c_out: out std_logic;
        Z: out std_logic_vector(7 downto 0));
end ripple_carry;

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

  signal temp: std_logic_vector(7 downto 0);

begin
  RA0: full_adder port map(X(0), Y(0), c_in, temp(0), Z(0));

  RA1to6: FOR i IN 1 TO 6 GENERATE
    RA: full_adder port map(X(i), Y(i), temp(i-1), temp(i), Z(i));
  END GENERATE;

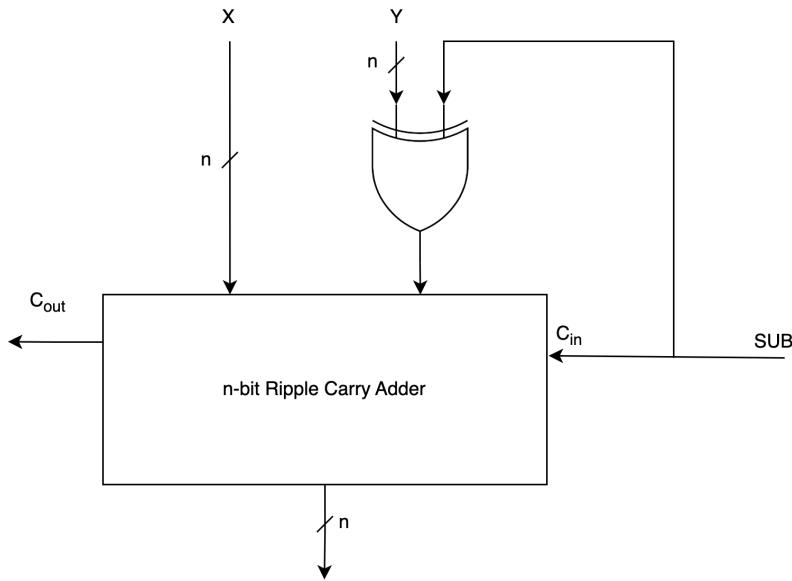
  RA7: full_adder port map(X(7), Y(7), temp(6), c_out, Z(7));
end structural;
  
```

Adder Subtractor

Progetto e architettura

Il componente Adder Subtractor viene utilizzato per poter svolgere all'occorrenza le operazioni di somma o sottrazione fra due numeri, sfruttando la proprietà per cui la sottrazione viene effettuata complementando ed aggiungendo un 1 al numero da sottrarre.

Il disegno dell'adder subtractor è il seguente:



Implementazione

Di seguito viene riportato il codice VHDL del componente:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity adder_sub is
    port( X, Y: in std_logic_vector(7 downto 0);
          cin: in std_logic;
          Z: out std_logic_vector(7 downto 0);
          cout: out std_logic);
end adder_sub;

architecture structural of adder_sub is
    component ripple_carry is
        port( X, Y: in std_logic_vector(7 downto 0);
              c_in: in std_logic;
              c_out: out std_logic;
              Z: out std_logic_vector(7 downto 0));
    end component;

    signal complementoy: std_logic_vector(7 downto 0);

begin
    complemento_y: FOR i IN 0 TO 7 GENERATE
        complementoy(i)<=Y(i) xor cin;
    END GENERATE;

    RA: ripple_carry port map(X, complementoy, cin, cout, Z);

end structural;

```

