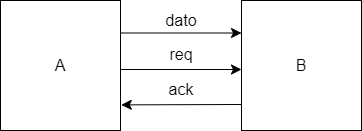
Capitolo 4: Comunicazione con Handshaking

Esercizio 8

Progetto e architettura

Sono stati progettati due sistemi, denominati A e B, che comunicano tramite un protocollo di handshaking semplice. In particolare, A deve trasmettere dati a B; quindi, i segnali di controllo che gestiscono la comunicazione sono due; **REQ** pilotato dal sistema A e **ACK** controllato da B; e questi sono “interlacciati” tra di loro nel seguente modo.

* **REQ = 0 🡪 1**: il sistema A ha scritto nuovi dati sui bus dedicati (**dato**), quindi, in maniera sincrona al proprio clock (**clkA**), alza il segnale REQ per avvisare B
* **ACK = 0 🡪 1**: Quando il sistema B rileva REQ alto, campionando rispetto al proprio clock **clkB**, legge i dati e alza il segnale ACK per informare A dell’avvenuta ricezione.
* **REQ = 1 🡪 0**: Quando A vede, su un fronte di attivazione di clkA, che il segnale di ACK è ‘1’ sa di poter terminare la comunicazione e abbassa REQ.
* **ACK = 0 🡪 1**:Quando B rileva la discesa di REQ (campionato su clkB) porta ACK a ‘0’.



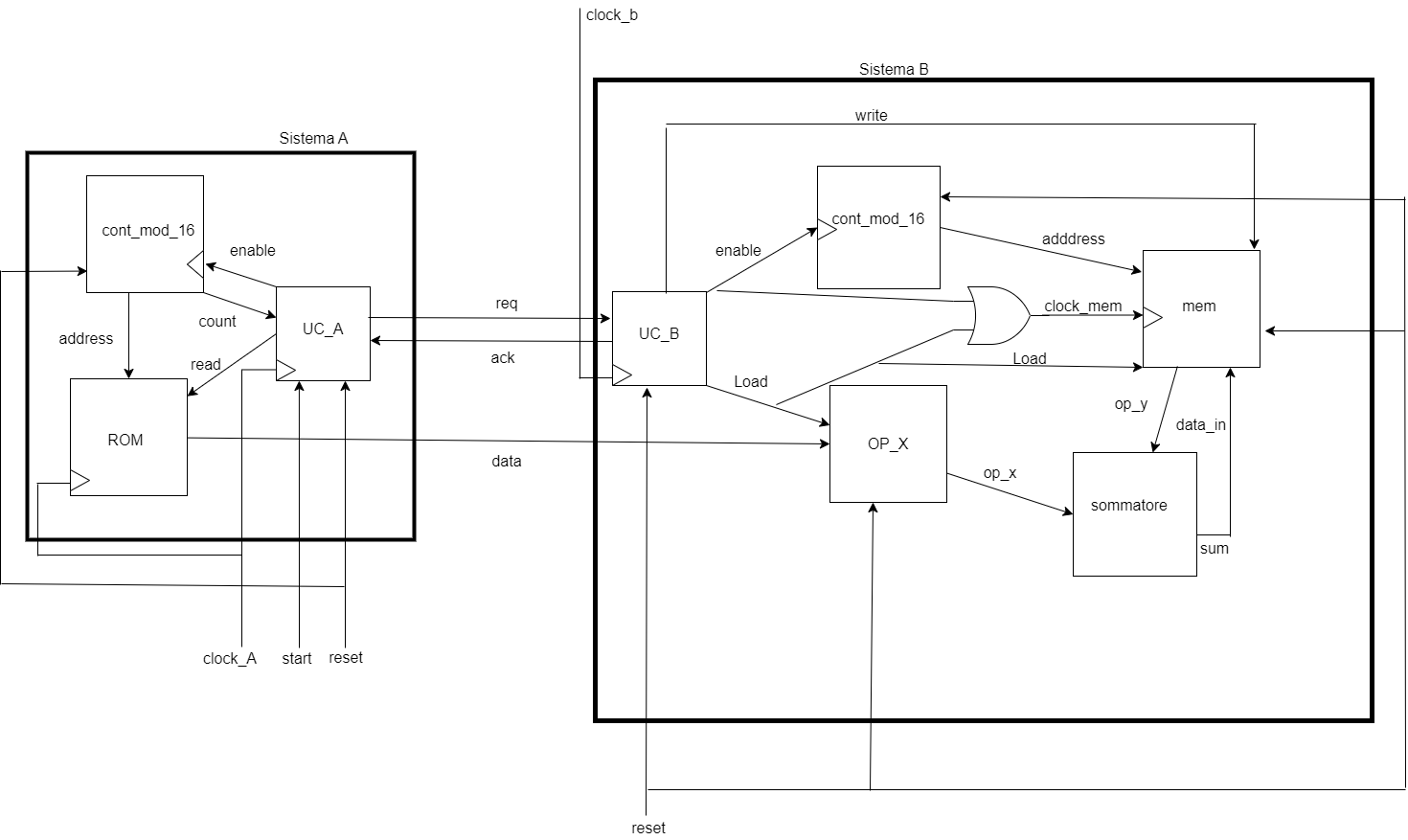
Il sistema A è composto da:

* Un'unità di controllo che comunica con il sistema B (inviando il segnale **req** e ricevendo il segnale **ack**) e fornisce agli altri componenti di A vari segnali di controllo, come il segnale **enable** per attivare il contatore e il segnale **read** alla ROM per ottenere il dato contenuto alla locazione **address**.
* Un contatore modulo 16 che riceve il segnale di abilitazione enable e fornisce il risultato del conteggio alla ROM, rappresentante l’indirizzo in cui leggere.
* Una ROM che riceve in ingresso una posizione e restituisce in uscita il dato attualmente presente in quel punto della memoria. Queste ultime informazioni sono quelle trasmesse a B.

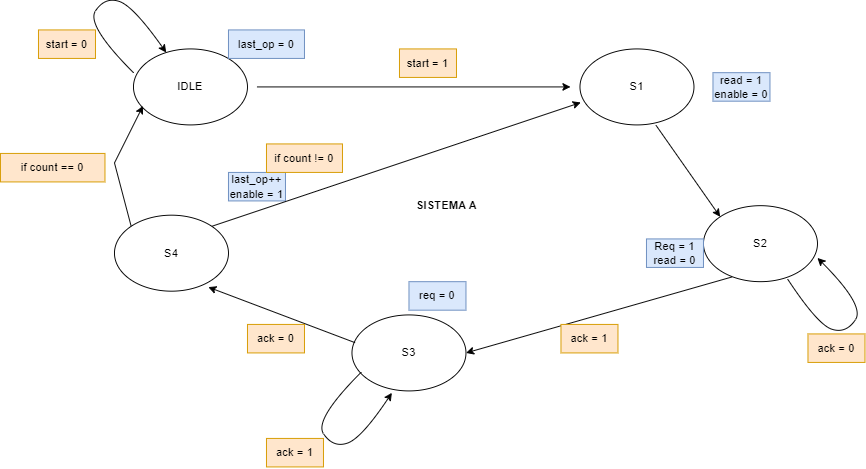
Il sistema B è composto da:

* Un'unità di controllo che comunica con l'unità di controllo del sistema A attraverso req e ack e che fornisce un segnale **enable** al proprio contatore, un segnale **load** al registro **op\_x** e un segnale **write** alla memoria.
* Un contatore mod-16 che riceve un segnale di abilitazione **enable** dall'unità di controllo e restituisce alla memoria il valore del conteggio, rappresentante l’indirizzo in cui leggere o scrivere.
* Una memoria che restituisce i valori scritti nelle sue locazioni come operando y al sommatore e riscrive nelle stesse posizioni il risultato fornito dal sommatore. Il clock della memoria è generato una OR tra i segnali enablee load**,** poiché deve attivarsi sia per le operazioni di scrittura che di lettura.
* Il registro op\_x riceve il segnale loaddall'unità di controllo e il dato dal sistema A che pone successivamente in ingresso sommatore come operando x.
* Il sommatore riceve due operandi: uno proveniente da op\_x e l'altro dalla memoria. Successivamente, il sommatore calcola, appunto, la somma di questi due operandi e mette il risultato in uscita. Per questo modulo è stata utilizzata una semplice architettura dataflow nella quale si descrivere il comportamento di un circuito digitale in termini del flusso dei dati.

Nel disegno successivo, viene illustrata l'architettura complessiva dei due sistemi.



Di seguito è possibile visualizzare il diagramma degli stati implementato dall’unità di controllo del sistema A.

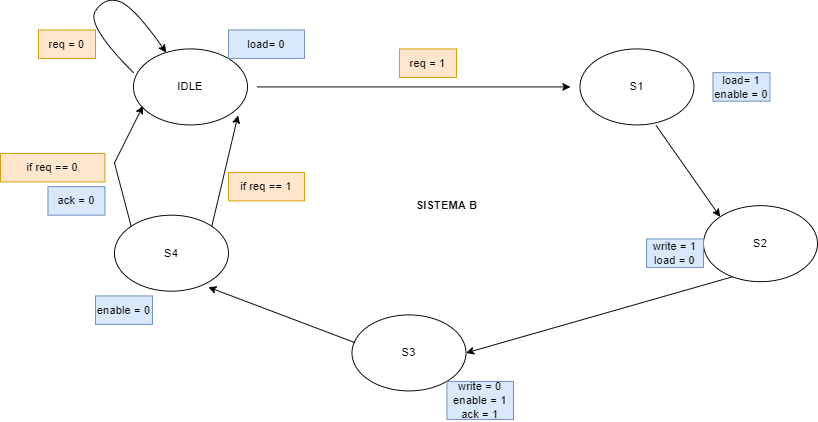


Si parte da uno stato **IDLE.** Il sistema non transita nello stato **S1** finché non riceve dall'esterno un segnale di **start**. Nello stato S1, viene abilitata la lettura (read) dalla ROM, che mette in uscita il dato conservato nella posizione data da address, mentre il segnale enable per il contatore è mantenuto a basso.

Nello stato **S2**, il segnale di read viene nuovamente abbassato e il segnale req viene alzato per avvisare il sistema B che il dato è stato inviato. Il sistema rimane in questo stato finché non osserva ack= 1 che sta a indicare che B ha ricevuto i dati e li sta elaborando.

Nello stato **S3**, viene abbassato il segnale req. Si passa allo stato **S4** se anche ack = 0; in questo stato, è possibile incrementare il contatore e ritornare allo stato **S1** per continuare la trasmissione, oppure, se il contatore si è di nuovo azzerato, si torna allo stato di IDLE.

L'immagine successiva rappresenta il diagramma degli stati finiti implementato dell’unità di controllo del sistema B.



Partendo dallo stato **IDLE**, il sistema mette loada 0 e rimane in questo stato fino a quando non vede il segnale req alto, che avvisa il sistema B del fatto che i dati ora scritti sui bus condivisi sono significativi.

Nello stato **S1**, il segnale loadviene impostato a ‘1’ (l'op\_x e la memoria caricano gli operandi in ingresso al sommatore) e enableè nuovamente azzerato.

Il sistema passa successivamente allo stato **S2**, dove writeè alzato (il sommatore scrive il risultato della somma nella stessa posizione da cui si era ottenuto il secondo operando, sovrascrivendolo).

Nello stato **S3**, writeviene abbassato e vengono alzati i segnali di enable (si incrementa il contatore) e di ack(si notifica A che B ha ricevuto correttamente i dati).

Nello stato **S4**, se req= 0 il segnale ackviene abbassato (il sistema B è di nuovo disponibile per nuove richieste) e si va in IDLE, se, invece, req =1 il sistema torna direttamente in IDLE senza aggiornare il segnale ack.

Implementazione

**sistema.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity sistema is -- complessivo

    Port (

        start, reset, clock : in std\_logic ); -- clock e’ il segnale di tempificazione del sistema A (a 10 ns)

end sistema;

architecture Structural of sistema is

    component sistema\_B is

      Port (

        clock : in std\_logic;

        reset : in std\_logic;

        req : in std\_logic;

        ack : out std\_logic;

        data : in std\_logic\_vector(3 downto 0));

    end component;

    component sistema\_A is

      Port (

        clock : in std\_logic;

        start : in std\_logic;

        reset : in std\_logic;

        req : out std\_logic;

        ack : in std\_logic;

        data : out std\_logic\_vector(3 downto 0));

    end component;

    signal ack, req : std\_logic;

    signal data : std\_logic\_vector(3 downto 0);

    signal count : integer range 0 to 1 := 0;

    signal clk\_B : STD\_LOGIC := '0'; -- clock e’ il segnale di tempificazione del sistema B (da 16 ns)

    constant CLK\_DIVISION\_FACTOR : integer := 2;

    constant COUNT\_MAX : integer := CLK\_DIVISION\_FACTOR - 1;

begin

    sis\_A : sistema\_A port map (clock => clock, start => start, reset => reset, req => req, ack => ack, data => data);

    sis\_B : sistema\_B port map (clock => clk\_B, reset => reset, req => req, ack => ack, data => data);

    process(clock, reset)

    begin

        if reset = '1' then

            count <= 0;

            clk\_B <= '0';

        elsif rising\_edge(clock) then

            if count = COUNT\_MAX then

                count <= 0;

                clk\_B <= not clk\_B;

            else

                count <= count + 1;

            end if;

        end if;

    end process;

end architecture Structural;

**sistema\_A.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

entity sistema\_A is

  Port (

    clock : in std\_logic;

    start : in std\_logic;

    reset : in std\_logic;

    req : out std\_logic;

    ack : in std\_logic;

    data : out std\_logic\_vector(3 downto 0));

end sistema\_A;

architecture Behavioral of sistema\_A is

    component ROM is port (

        clock : in STD\_LOGIC;

        reset : in STD\_LOGIC;

        read : in STD\_LOGIC;

        address : in STD\_LOGIC\_VECTOR(3 downto 0);

        data : out STD\_LOGIC\_VECTOR(3 downto 0));

    end component;

    component cont\_mod\_16 is

    generic (

        MAX\_VALUE : integer := 15

    );

    port (

        clock : in std\_logic;

        reset : in std\_logic;

        count : out integer range 0 to MAX\_VALUE

    );

    end component;

    component unita\_controllo\_A is

    Port ( clock : in STD\_LOGIC;

           start : in STD\_LOGIC;

           read : out STD\_LOGIC;

           reset : in STD\_LOGIC;

           enable : out STD\_LOGIC;

           count : in STD\_LOGIC\_VECTOR(3 downto 0);

           req : out std\_logic;

           ack : in std\_logic);

    end component;

    signal count : std\_logic\_vector(3 downto 0) := (others=>'0');

    signal enable : std\_logic := '0';

    signal read : std\_logic := '0';

    signal conteggio : integer := 0;

begin

    mem\_rom : ROM port map (clock => clock, reset => reset, read => read, address => count, data => data);

    cont : cont\_mod\_16 port map(clock => enable, reset => reset, count => conteggio);

    count <= std\_logic\_vector(to\_unsigned(conteggio, count'length));

    control\_unit : unita\_controllo\_A port map (clock => clock, read => read, start => start, reset => reset, enable => enable, count => count, req => req, ack => ack);

end Behavioral;

**unita\_controllo\_A.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

entity unita\_controllo\_A is

    Port ( clock : in STD\_LOGIC;

           start : in STD\_LOGIC;

           reset : in STD\_LOGIC;

           read : out STD\_LOGIC;

           enable : out STD\_LOGIC;

           count : in STD\_LOGIC\_VECTOR(3 downto 0);

           req : out std\_logic;

           ack : in std\_logic);

end unita\_controllo\_A;

architecture Behavioral of unita\_controllo\_A is

    type state is (idle, s1, s2, s3, s4);

    signal current\_state, next\_state: state;

begin

    reg\_stato: process(clock)

    begin

       if(clock'event and clock='1') then

           if(reset='1') then

              current\_state <= idle;

           else

              current\_state <= next\_state;

           end if;

       end if;

    end process;

    comb: process(current\_state, start, count, ack)

    variable last\_op : integer;

    begin

        case current\_state is

           when idle =>

              last\_op := 0;

              if(start='1') then

                next\_state <= s1;

              else

                next\_state <= idle;

              end if;

           when s1 =>

               read <= '1';

               enable <= '0';

               next\_state <= s2;

           when s2 =>

               req <= '1';

               read <= '0';

               if(ack/='1') then

                   next\_state <= s2;

               else next\_state <= s3;

               end if;

           when s3 =>

               req <= '0';

               if(ack='1') then

                   next\_state <= s3;

               else

                   next\_state <= s4;

               end if;

           when s4 =>

               if(count="0000" and last\_op/=0) then

                   next\_state <= idle;

               else

                  last\_op:=1;

                  enable <= '1';

                  next\_state <= s1;

               end if;

        end case;

    end process;

end Behavioral;

**ROM. vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

entity ROM is port (

    clock : in STD\_LOGIC;

    read : in STD\_LOGIC;

    reset : in STD\_LOGIC;

    address : in STD\_LOGIC\_VECTOR(3 downto 0);

    data : out STD\_LOGIC\_VECTOR(3 downto 0));

end ROM;

architecture Behavioral of ROM is

    type memory\_array is array (0 to 15) of STD\_LOGIC\_VECTOR(3 downto 0);

    constant rom\_data : memory\_array := (

        0 => "0000", 1 => "0001", 2 => "0010", 3 => "0011",

        4 => "0100", 5 => "0101", 6 => "0110", 7 => "0111",

        8 => "1000", 9 => "1001", 10 => "1010", 11 => "1011",

        12 => "1100", 13 => "1101", 14 => "1110", 15 => "1111"

    );

begin

    process(clock)

    begin

        if reset='1' then

            data <= (others=>'0');

        elsif(rising\_edge(clock) and read='1') then

            data <= rom\_data(to\_integer(unsigned(address)));

        end if;

    end process;

end Behavioral;

**cont\_mod\_16.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity cont\_mod\_16 is

    generic (

        MAX\_VALUE : integer := 15

    );

    port (

        clock : in std\_logic;

        reset : in std\_logic;

        count : out integer range 0 to MAX\_VALUE

    );

end entity;

architecture Behavioral of cont\_mod\_16 is

    signal counter : integer range 0 to MAX\_VALUE :=0;

begin

    process(clock, reset)

    begin

        if reset = '1' then

                counter <= 0;

        elsif rising\_edge(clock) then

            if counter < MAX\_VALUE then

                counter <= counter + 1;

            else

                counter <= 0;

            end if;

        end if;

    end process;

    count <= counter;

end architecture;

**sistema\_B.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

entity sistema\_B is

  Port (

    clock : in std\_logic;

    reset : in std\_logic;

    req : in std\_logic;

    ack : out std\_logic;

    data : in std\_logic\_vector(3 downto 0));

end sistema\_B;

architecture Structural of sistema\_B is

    component cont\_mod\_16 is

    generic (

        MAX\_VALUE : integer := 15

    );

    port (

        clock : in std\_logic;

        reset : in std\_logic;

        count : out integer range 0 to MAX\_VALUE

    );

    end component;

    component operando\_x is

    Port (

        clock : in std\_logic;

        reset : in std\_logic;

        dato\_in : in std\_logic\_vector(3 downto 0);

        dato\_out : out std\_logic\_vector(3 downto 0)

    );

    end component;

    component sommatore is

    Port ( A : in STD\_LOGIC\_VECTOR (3 downto 0);

           B : in STD\_LOGIC\_VECTOR (3 downto 0);

           sum : out STD\_LOGIC\_VECTOR (3 downto 0));

    end component;

    component unita\_controllo\_B is

    Port ( clock : in STD\_LOGIC;

           reset : in STD\_LOGIC;

           load : out STD\_LOGIC;

           write : out STD\_LOGIC;

           enable : out STD\_LOGIC;

           req : in std\_logic;

           ack : out std\_logic);

    end component;

    component memoria is

        Port (

            reset : in STD\_LOGIC;

            clock : in STD\_LOGIC;

            address : in STD\_LOGIC\_VECTOR (3 downto 0);

            data\_in : in STD\_LOGIC\_VECTOR (3 downto 0);

            write : in STD\_LOGIC;

            load : in STD\_LOGIC;

            data\_out : out STD\_LOGIC\_VECTOR (3 downto 0)

        );

    end component;

    signal load, write : std\_logic := '0';

    signal count : std\_logic\_vector(3 downto 0) := (others=>'0');

    signal enable : std\_logic := '0';

    signal conteggio : integer := 0;

    signal op\_x, op\_y, sum: std\_logic\_vector(3 downto 0) := (others=>'0');

    signal clock\_mem : std\_logic := '0';

begin

    control\_unit\_B : unita\_controllo\_B port map (clock => clock, enable => enable, reset => reset, load => load, write => write, req => req, ack => ack);

    cont\_B : cont\_mod\_16 port map(clock => enable, reset => reset, count => conteggio);

    count <= std\_logic\_vector(to\_unsigned(conteggio, count'length));

    operand\_x : operando\_x port map(clock => load, reset => reset, dato\_in => data, dato\_out => op\_x);

    somm : sommatore port map (A => op\_x, B => op\_y, sum => sum);

    clock\_mem <= enable or load;

    mem : memoria port map(reset => reset, clock => clock\_mem, address => count, data\_in => sum, write => write, load => load, data\_out => op\_y);

end Structural;

**unita\_controllo\_B.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

entity unita\_controllo\_B is

    Port ( clock : in STD\_LOGIC;

           reset : in STD\_LOGIC;

           enable : out STD\_LOGIC;

           load : out STD\_LOGIC;

           write : out STD\_LOGIC;

           req : in std\_logic;

           ack : out std\_logic);

end unita\_controllo\_B;

architecture Behavioral of unita\_controllo\_B is

    type state is (idle, s1, s2, s3, s4);

    signal current\_state: state;

begin

    process(clock)

    begin

        if rising\_edge(clock) then

            if reset = '1' then

                current\_state <= idle;

            else

                case current\_state is

                    when idle =>

                        load <= '0';

                        if req = '1' then

                            current\_state <= s1;

                        else

                            current\_state <= idle;

                        end if;

                    when s1 =>

                        load <= '1';

                        enable <= '0';

                        current\_state <= s2;

                    when s2 =>

                        load <= '0';

                        write <= '1';

                        current\_state <= s3;

                    when s3 =>

                        write <= '0';

                        enable <= '1';

                        ack <= '1';

                        current\_state <= s4;

                    when s4 =>

                        enable <= '0';

                        if req = '1' then

                            current\_state <= idle;

                        else

                            ack <= '0';

                            current\_state <= idle;

                        end if;

                end case;

            end if;

        end if;

    end process;

end Behavioral;

**operando\_x.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity operando\_x is

    Port (

        clock : in std\_logic;

        reset : in std\_logic;

        dato\_in : in std\_logic\_vector(3 downto 0);

        dato\_out : out std\_logic\_vector(3 downto 0)

    );

end operando\_x;

architecture Behavioral of operando\_x is

    signal op\_x : std\_logic\_vector(3 downto 0) := (others => '0');

begin

    process(clock, reset)

    begin

        if reset = '1' then

            op\_x <= (others => '0');

        elsif (clock'event and clock='1') then

            op\_x <= dato\_in;

        end if;

    end process;

    dato\_out <= op\_x;

end Behavioral;

**sommatore.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity sommatore is

    Port ( A : in STD\_LOGIC\_VECTOR (3 downto 0);

           B : in STD\_LOGIC\_VECTOR (3 downto 0);

           sum : out STD\_LOGIC\_VECTOR (3 downto 0));

end sommatore;

architecture Dataflow of sommatore is

begin

    sum <= std\_logic\_vector(unsigned(A) + unsigned(B));

end Dataflow;

**memoria.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

entity memoria is

    Port (

        reset : in STD\_LOGIC;

        clock : in STD\_LOGIC;

        address : in STD\_LOGIC\_VECTOR (3 downto 0);

        data\_in : in STD\_LOGIC\_VECTOR (3 downto 0);

        write : in STD\_LOGIC;

        load : in STD\_LOGIC;

        data\_out : out STD\_LOGIC\_VECTOR (3 downto 0)

    );

end memoria;

architecture Behavioral of memoria is

    type memory\_array is array (0 to 15) of STD\_LOGIC\_VECTOR (3 downto 0);

    signal memory : memory\_array := (

        0 => "0000", 1 => "0001", 2 => "0010", 3 => "0011",

        4 => "0100", 5 => "0101", 6 => "0110", 7 => "0111",

        8 => "1000", 9 => "1001", 10 => "1010", 11 => "1011",

        12 => "1100", 13 => "1101", 14 => "1110", 15 => "1111"

    );

begin

    process(clock, reset)

    begin

        if reset='1' then

            data\_out <= (others=>'0');

        elsif (clock'event and clock='1') then

            if write = '1' then

                memory(to\_integer(unsigned(address))) <= data\_in;

            elsif load = '1' then

                data\_out <= memory(to\_integer(unsigned(address)));

            end if;

        end if;

    end process;

end Behavioral;

Simulazione

Sono stati sviluppati tre testbench principali: uno dedicato alla simulazione del sistema A in isolamento (testbench\_A), uno per testare esclusivamente B, e infine, uno per valutare l'intero sistema composto sia da A che da B (testbench\_sis). In seguito, viene riportato sia il codice VHDL che i risultati ottenuti durante l’ultima simulazione menzionata.

**testbench\_sis.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity testbench\_sis is

end testbench\_sis;

architecture Behavioral of testbench\_sis is

    component sistema is

    Port (

        start, reset, clock : in std\_logic );

    end component;

    signal clk, rst,start : std\_logic  := '0';

begin

    clock\_gen: process

    begin

        wait for 2 ns;

        clk <= not clk;

    end process;

    sis : sistema port map (start=>start,reset=>rst,clock=>clk);

    process

    begin

        wait for 1 ns;

        start<='1';

        wait for 6 ns;

        start<='0';

        wait for 900 ns;

        wait;

    end process;

end Behavioral;

