Capitolo 6: Interfaccia Seriale

Esercizio 10: RS232

Progetto e architettura

Il progetto prevede un sistema composto da due unità (che condividono lo stesso segnale di clock) comunicanti tramite interfaccia seriale, tale che, dato un segnale **start,** venga prelevato un byte dalla ROM del sistema A e inviato al sistema B per essere scritto in una memoria.

Il dispositivo utilizzato per la comunicazione seriale asincrona è la **UART** (Universal Asynchronus Receiver Transmitter) nel quale la comunicazione avviene secondo un preciso protocollo. Lo standard impiegato è l’**RS-232** (Recommended Standard 232) che è uno standard di interfaccia seriale che specifica la connessione elettrica, i segnali e le regole per la trasmissione seriale dei dati. Per quanto riguarda le specifiche elettriche, l’intervallo di tensione [+3 V, +15 V] è usato per rappresentare il valore logico basso e l’intervallo [-3 V, -15 V] per il valore logico basso; questa soluzione permette di tollerare meglio il rumore.

Immagine che contiene testo, diagramma, Carattere, schermata

Descrizione generata automaticamente

Dal punto di vista architetturale la componente è formata da una porzione dedicata alla ricezione e una alla trasmissione, ognuna delle quali contiene:

* Un registro per la serializzazione/deserializzazione dei bit inviati/ricevuti (PISO per il trasmettitore e SIPO per il ricevitore).
* Un contatore per scandire i bit del frame.
* Un contatore per gestire l’invio dei singoli bit in base al baud rate scelto.
* Un'unità di controllo.

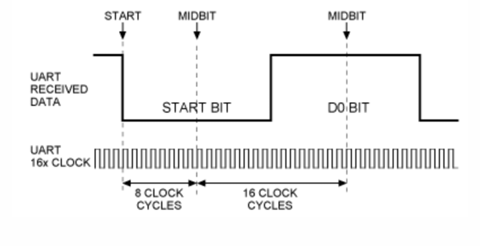
I segnali di ingresso e uscita del componente RS232 sono:

* **TXD** per la trasmissione dei dati (di default ha valore 1)
* **RXD** per la ricezione dei dati
* **CLK** (Clock)
* **DBIN** (Data Bus In) contiene l’intero vettore di bit che deve essere trasmesso
* **DBOUT** (Data Bus Out) contiene il vettore di bit ricevuto nella sua interezza
* **RDA** (Read Data Available) informa che si è conclusa la ricezione dei dati
* **TBE** (Transfer Bus Empty) informa che si è pronti a inviare nuovi dati
* **RD** (Read Strobe) avvisa il ricevitore di leggere i dati (se RD = 1 allora RDA=0)
* **WR** (Write Strobe) avvisa il trasmettitore di inviare i dati (se WR = 1 allora TBE= 0)
* **PE** (Parity Error Flag)
* **FE** (Frame Error Flag)
* **OE** (Overwrite Error Flag)
* **RST** (Reset)

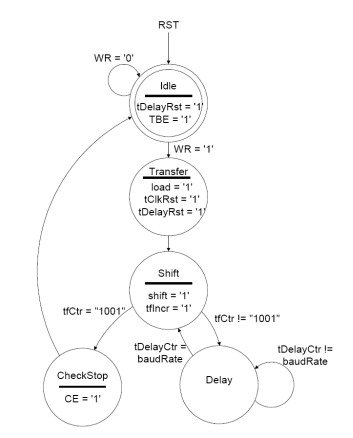
Il componente usato usa un baud rate, una velocità, di 9600, ovvero trasmette e riceve 9600 bit al secondo. La frequenza di campionamento dell’interconnessione deve però essere diversa tra le porzioni di trasmissione e ricezione, dove il ricevitore deve presentare una frequenza 16 o 64 volte maggiore.

Il codice VHDL contenuto in “RS232RefComp.vhd”genera, a partire da CLK, un clock in ricezione (**rCLK**) con duty cycle del 50% e da quest’ultimo viene ricavato il clock di trasmissione (**tCLK**). rCLK è ottenuto dal segnale **clkDiv** nel seguente modo: clkDiv è incrementato ogni colpo di clock (CLK), quando clkDiv raggiunge il valore massimo **baudDivide**, ossia 163, si azzera e viene invertito rClk. Inoltre, ogni volta che si osserva un fronte di salita di rClk, il segnale **rClkDiv**, a quattro bit, viene incrementato. tClk non è altro che l’ultimo bit di **rClkDiv** e quindi esso commuta ogni otto colpi di rClk.

In questo modo si ottiene una frequenza di clock per il ricevitore 16 volte quella del trasmettitore. Appena viene riconosciuto lo start bit, il ricevitore si posiziona al “centro” del bit contando un certo numero di fronti del suo clock, in questo caso 8; successivamente campionerà il filo collegato a RXD ogni 16 colpi così da posizionarsi sempre nel mezzo.



Di seguito è riportato il diagramma degli stati del controller del “Transmitter Portion”del componente UART. Partendo da **IDLE** sirimane in questo stato finché non si alza il segnale esterno WR**,** che indica l’inizio della trasmissione. Si passa, quindi, allo stato successivo di **Transfer** in cui si alza sia **load** (viene caricata la stringa nello Shift Register) che **tClkRst** (viene resettato il contatore di bit inviati). Nella fase successiva di **Shift**, il bit uscente dallo shift register viene trasmesso. Si transita nello stato di **Delay**, dove si rimane per un certo intervallo di tempo in accordo con il bound rate per poi ritornare in Shift. Questo ciclo si ripete finché il frame non è stato completamente trasmesso ovvero quando **tfCtr**, il qualeconta quanti bit sono stati inviati,è uguale a 12 (11 bit del frame più 1 per separare la trasmissione di due frame successivi); a questo punto di transiziona nello stato di IDLE.

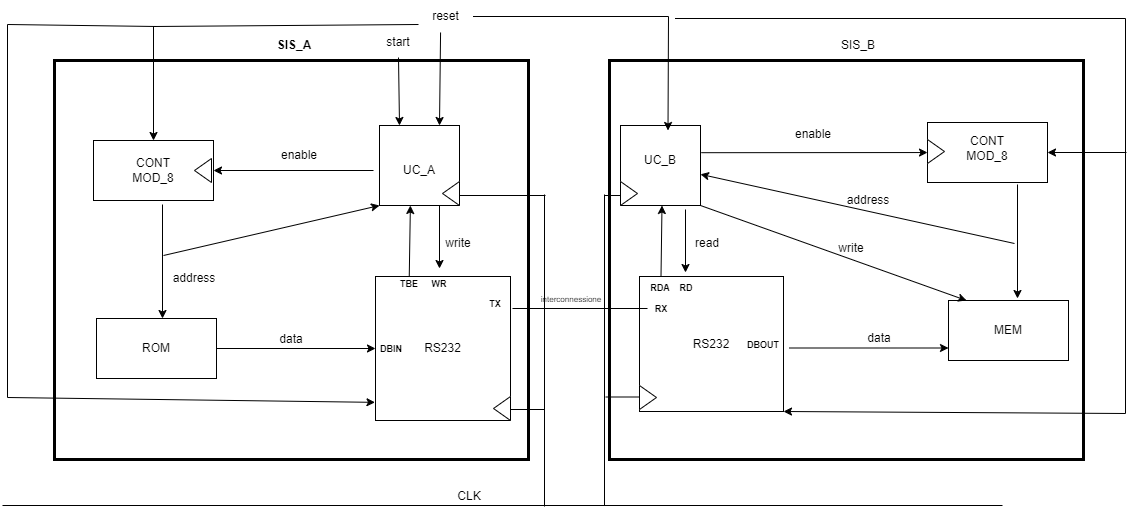


Successivamente è illustrato il diagramma a stati dell’unità di controllo del RXdel componente UART. A partire dallo stato **IDLE**, si fa una transizione allo stato successivo se RXDsi abbassa, evento che indica l'inizio di una comunicazione. Si rimane nello stato **EightDelay** il tempo necessario per contare 8 impulsi, in modo tale da posizionarsi al centro del bit di start. Appena **ctr** arriva ad 8, si passa allo stato successivo **WaitFor0**;gli stati WaitFor0e **WaitFor1** sono strutturati in maniera tale che la lettura di RXD avviene sempre al centro della trasmissione. In altre parole, il loro funzionamento combinato permette di contare 16 colpi di clock: partendo da WaitFor0quando **ctr**(3)=0 si sono contati 8 impulsi, si va quindi nello stato **WaitFor1** e qui si rimane finchè ctr(3)=1 ovvero si sono contati altri 8 impulsi, alla fine si fa la transizione in **GetData.** Nello stato GetDatasi incrementa **dataCtr**, il contatore dei bit di dato ricevuti, si fornisce il segnale di shift e si torna in WaitFor0. Si ripete in loop fino a quando **datCtr** non diventa10 (8 bit di dati, 1 di parità e 1 di stop), a questo punto ci si sposta nello stato **CheckStop** che abilita il controllo degli errori alzando il bit CE. Infine si transiziona di nuovo nello stato di riposo.

Immagine che contiene testo, diagramma, cerchio, Carattere

Descrizione generata automaticamente

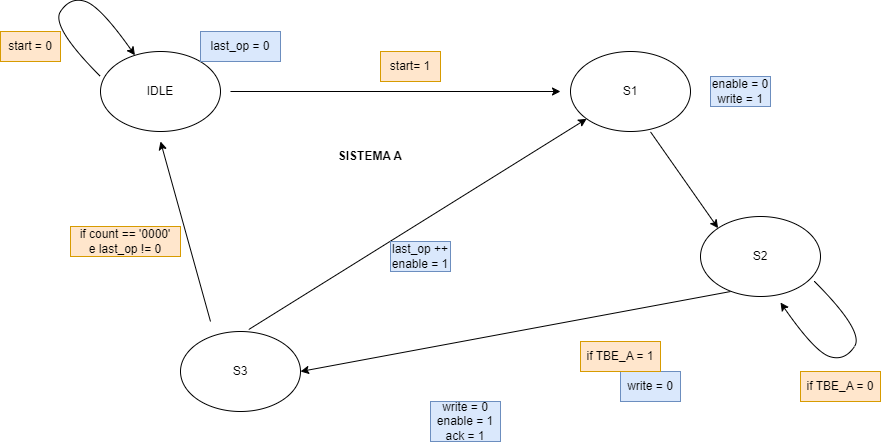
Di seguito è disegnata l’architettura dei due sistemi comunicanti. Si osservi che il segnale di tempificazione, il clock, è condiviso.



Il sistema A è composto da un’unita di controllo, una componente UART RS232 con TXconnesso a RXdel ricevitore, un contatore modulo 8 e una memoria ROM. Un segnale di startfornito dall’ esternoal sistema dà il via alla trasmissione seriale, a intervalli regolari, dei dati contenuti nella ROM (l’indirizzo è fornito dal contatore) al sistema B, tramite UART.

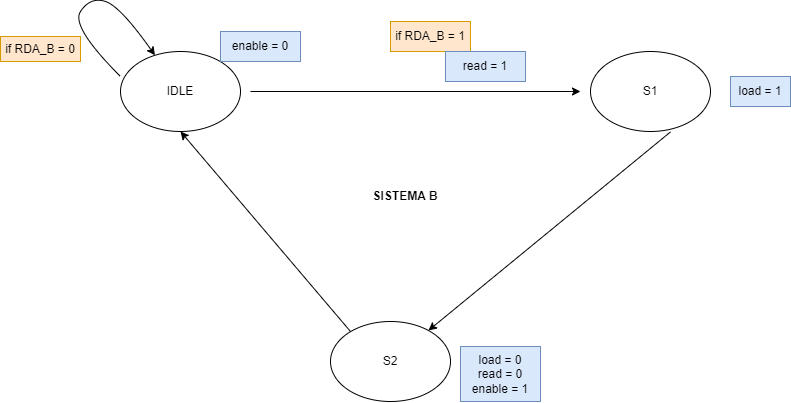
Il diagramma degli stati finiti del sistema A è descritto successivamente.

Per determinare lo stato in un certo istante l’unità di controllo riceve dal componente RS232 il segnale TBEche ha di default un valore logico alto. Nel momento in cui WRsi alza e si inizia una trasmissione, TBEdi conseguenza si abbassa. Solo nel momento in cui è finita una trasmissione TBEè di nuovo 1 quindi si torna allo stato di partenza.



Si parte da uno stato iniziale **IDLE**. Se viene fornito dall’esterno un segnale startsi passa a uno stato **S1**, dove **enable** è abbassato e **write** è impostato ad 1 (writeva in ingresso al componente RS232 come segnale WR che dà inizio alla trasmissione). Successivamente si passa ad uno stato **S2** in cui si permane finché il segnale **TBE\_A**, che l’unità di controllo riceve dall’UART, si alza; in questo caso viene abbassato write. Se il contatore mod-8 ha raggiunto il valore massimo, si azzera e si ritorna allo stato IDLE, altrimenti si abilita **enable** per incrementare il contatore e si transiziona nello stato **S1**.

Il sistema B è composto, anch’esso da un’unità di controllo, un componente RS232 con **RX** connesso al **TX** del trasmettitore, un contatore mod-8 e una memoria. Il diagramma a stati dell’unità di controllo di B sfrutta il segnale **RDA** in uscita al componente RS232 per determinare quando un frame è stato completamente ricevuto.



Partendo da uno stato **IDLE,** il sistemapermane qui fino a quando **RDA\_B** non assume valore logico alto. Si fa, quindi, una transizione allo stato **S1** in cuisi abilita il segnale **read** per la lettura del dato, che andrà in ingresso al componente RS232 come RD, e si alza **load** per il caricamento dei dati ricevuti nella memoria. Successivamente nello stato **S3** vengono abbassati i segnali di **load** e **read,** mentre l’**enable** del contatore viene abilitato. Infine, si torna ad **IDLE**.

Implementazione

Per l’unità di controllo del sistema A il diagramma degli stati finiti è stato implementato mediante due process. Invece, per l’unità di controllo di B è stata utilizzato un unico process.

**sistema\_complessivo.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity sistema\_complessivo is

    Port (

    clock, start, reset : in std\_logic

     );

end sistema\_complessivo;

architecture Behavioral of sistema\_complessivo is

    component sistema\_A is

    Port (clock, start, reset: in std\_logic;

        interconnessione : out std\_logic

     );

    end component;

    component sistema\_B is

    Port (clock, reset, interconnessione : in std\_logic );

    end component;

    signal interconnessione : STD\_LOGIC;

begin

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

    sis\_B : sistema\_B port map (clock => clock, reset => reset, interconnessione => interconnessione);

end Behavioral;

**sistema\_A.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity sistema\_A is

    Port (clock, start, reset: in std\_logic;

        interconnessione : out std\_logic

     );

end sistema\_A;

architecture Behavioral of sistema\_A is

    component ROM is port(

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

            d\_out    : out std\_logic\_vector(7 downto 0)

        );

    end component;

    component cont\_mod\_8 is

    generic (

        MAX\_VALUE : integer := 7

    );

    port (

        clock : in std\_logic;

        reset : in std\_logic;

        count : out std\_logic\_vector(3 downto 0)

    );

    end component;

    component Rs232RefComp is

    Port (

        TXD     : out std\_logic     := '1';

        RXD     : in  std\_logic;

        CLK     : in  std\_logic;                    --Master Clock

        DBIN    : in  std\_logic\_vector (7 downto 0);--Data Bus in

        DBOUT : out std\_logic\_vector (7 downto 0);  --Data Bus out

        RDA : inout std\_logic;                      --Read Data Available(1 quando il dato è disponibile nel registro rdReg)

        TBE : inout std\_logic   := '1';             --Transfer Bus Empty(1 quando il dato da inviare è stato caricato nello shift register)

        RD      : in  std\_logic;                    --Read Strobe(se 1 significa "leggi" --> fa abbassare RDA)

        WR      : in  std\_logic;                    --Write Strobe(se 1 significa "scrivi" --> fa abbassare TBE)

        PE      : out std\_logic;                    --Parity Error Flag

        FE      : out std\_logic;                    --Frame Error Flag

        OE      : out std\_logic;                    --Overwrite Error Flag

        RST     : in  std\_logic := '0');            --Master Reset

    end component;

    component control\_unit\_A is

    Port ( clock : in STD\_LOGIC;

           start : in STD\_LOGIC;

           reset : in STD\_LOGIC;

           write : out STD\_LOGIC;

           enable : out STD\_LOGIC;

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

           TBE\_A : inout STD\_LOGIC

     );

    end component;

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

    signal write, -- della UART

        TBE\_A,

        enable : std\_logic; -- del contatore

    signal data, DBIN, DBOUT : std\_logic\_vector(7 downto 0); -- DBOUT non usato

    signal RDA, PE, FE, OE : std\_logic :='0'; -- non usati

begin

    mem\_rom : ROM port map (address => address, d\_out => data);

    cont\_A : cont\_mod\_8 port map(clock => enable, reset => reset, count => address);

    contr\_unit\_A : control\_unit\_A port map(clock => clock,

        start => start,

        reset => reset,

        write => write,

        enable => enable,

        count => address,

        TBE\_A => TBE\_A);

    UART\_A : Rs232RefComp port map(

         TXD => interconnessione,

         RXD => '0',

         CLK => clock,

         DBIN => data,

         DBOUT => DBOUT,

         RDA => RDA,

         TBE => TBE\_A,

         RD => '0',

         WR => write,

         PE => PE,

         FE => FE,

         OE => OE,

         RST => reset);

end Behavioral;

**control\_unit\_A.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

entity control\_unit\_A is

    Port ( clock : in STD\_LOGIC;

           start : in STD\_LOGIC;

           reset : in STD\_LOGIC;

           write : out STD\_LOGIC;

           enable : out STD\_LOGIC;

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

           TBE\_A : in STD\_LOGIC

     );

end control\_unit\_A;

architecture Behavioral of control\_unit\_A is

    type state is (idle, s1, s2, s3);

    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, TBE\_A)

    variable last\_op : integer :=0;

    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 =>

               enable <= '0';

               write <= '1';

               next\_state <= s2;

           when s2 =>

               if(TBE\_A='0') then

                   next\_state <= s2;

               else

                   write <= '0';

                   next\_state <= s3;

               end if;

           when s3 =>

               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;

**sistema\_B.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity sistema\_B is

    Port (clock, reset, interconnessione : in std\_logic);

end sistema\_B;

architecture Behavioral of sistema\_B is

    component MEM is

        Port (

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

            data\_in : in STD\_LOGIC\_VECTOR(7 downto 0);

            write : in STD\_LOGIC;

            read : in STD\_LOGIC;

            data\_out : out STD\_LOGIC\_VECTOR(7 downto 0)

        );

    end component;

    component cont\_mod\_8 is

    generic (

        MAX\_VALUE : integer := 7

    );

    port (

        clock : in std\_logic;

        reset : in std\_logic;

        count : out std\_logic\_vector(3 downto 0)

    );

    end component;

    component control\_unit\_B is

    Port ( clock : in STD\_LOGIC;

           reset : in STD\_LOGIC;

           enable : out STD\_LOGIC;

           load : out STD\_LOGIC;

           read : out STD\_LOGIC;

           RDA\_B : in STD\_LOGIC

           );

    end component;

    component Rs232RefComp is

    Port (

        TXD     : out std\_logic     := '1';

        RXD     : in  std\_logic;

        CLK     : in  std\_logic;                    --Master Clock

        DBIN    : in  std\_logic\_vector (7 downto 0);--Data Bus in

        DBOUT : out std\_logic\_vector (7 downto 0);  --Data Bus out

        RDA : inout std\_logic;                      --Read Data Available(1 quando il dato è disponibile nel registro rdReg)

        TBE : inout std\_logic   := '1';             --Transfer Bus Empty(1 quando il dato da inviare è stato caricato nello shift register)

        RD      : in  std\_logic;                    --Read Strobe(se 1 significa "leggi" --> fa abbassare RDA)

        WR      : in  std\_logic;                    --Write Strobe(se 1 significa "scrivi" --> fa abbassare TBE)

        PE      : out std\_logic;                    --Parity Error Flag

        FE      : out std\_logic;                    --Frame Error Flag

        OE      : out std\_logic;                    --Overwrite Error Flag

        RST     : in  std\_logic := '0');            --Master Reset

    end component;

    signal data, data\_out : std\_logic\_vector(7 downto 0); -- data\_ount non utilizzato

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

    signal enable, -- del contatore

        load, -- della memoria

        read: std\_logic; -- della UART

    signal TXD, TBE, PE, FE, OE : std\_logic; -- non usati

    signal DBIN, DBOUT : STD\_LOGIC\_VECTOR(7 DOWNTO 0); -- DBIN non usati

    signal RDA\_B : std\_logic :='0';

begin

    mem\_data : MEM port map(address => address, data\_in => data, write => load, read => '0', data\_out => data\_out);

    cont\_B : cont\_mod\_8 port map(clock => enable, reset => reset, count => address);

    contr\_unit\_B : control\_unit\_B port map(clock => clock, reset => reset, read => read, enable => enable, load => load, RDA\_B => RDA\_B);

    UART\_B : Rs232RefComp port map(

        TXD => TXD,

        RXD => interconnessione,

        CLK => clock,

        DBIN => "00000000",

        DBOUT => data,

        RDA => RDA\_B,

        TBE => TBE,

        RD => read,

        WR => '0',

        PE => PE,

        FE => FE,

        OE => OE,

        RST => reset);

end Behavioral;

**control\_unit\_B .vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all;

entity control\_unit\_B is

    Port ( clock : in STD\_LOGIC;

           reset : in STD\_LOGIC;

           enable : out STD\_LOGIC;

           load : out STD\_LOGIC;

           read : out STD\_LOGIC;

           RDA\_B : in STD\_LOGIC

           );

end control\_unit\_B;

architecture Behavioral of control\_unit\_B is

    type state is (idle, s1, s2);

    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 =>

                        enable <= '0';

                        if RDA\_B = '1' then

                            read <= '1';

                            current\_state <= s1;

                        else

                            current\_state <= idle;

                        end if;

                    when s1 =>

                        load <= '1';

                        current\_state <= s2;

                    when s2 =>

                        load <= '0';

                        read <= '0';

                        enable <= '1';

                        current\_state <= idle;

                end case;

            end if;

        end if;

    end process;

end Behavioral;

**cont\_mod\_8.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use ieee.numeric\_std.all;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity cont\_mod\_8 is

    generic (

        MAX\_VALUE : integer := 7

    );

    port (

        clock : in std\_logic;

        reset : in std\_logic;

        count : out std\_logic\_vector(3 downto 0) -- std\_logic\_vector di 4 bit

    );

end entity cont\_mod\_8;

architecture Behavioral of cont\_mod\_8 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 <= std\_logic\_vector(to\_unsigned(counter, count'length));

end architecture Behavioral;

**ROM.vhd**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity ROM is port(

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

        d\_out    : out std\_logic\_vector(7 downto 0)

    );

end entity ROM;

architecture RTL of ROM is

    type MEMORY\_16\_8 is array (0 to 15) of std\_logic\_vector(7 downto 0);

    constant ROM\_16\_8 : MEMORY\_16\_8 := (

        "00000000",

        "00000001",

        "00000010",

        "00000011",

        "00000100",

        "00000101",

        "00000110",

        "00000111",

        "00001000",

        "00001001",

        "00001010",

        "00001011",

        "00001100",

        "00001101",

        "00001110",

        "00001111"

    );

begin

    main : process(address)

    begin

        d\_out <= ROM\_16\_8(to\_integer(unsigned(address)));

    end process main;

end architecture RTL;

**MEM.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity MEM is

    Port (

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

        data\_in : in STD\_LOGIC\_VECTOR(7 downto 0);

        write : in STD\_LOGIC;

        read : in STD\_LOGIC;

        data\_out : out STD\_LOGIC\_VECTOR(7 downto 0)

    );

end MEM;

architecture Behavioral of MEM is

    type Memory\_Array is array (0 to 7) of STD\_LOGIC\_VECTOR(7 downto 0);

    signal mem : Memory\_Array := (others => "00000000");

begin

    process(address, data\_in, write, read)

    begin

        if write = '1' then

            mem(conv\_integer(address)) <= data\_in;

        elsif read = '1' then

            data\_out <= mem(conv\_integer(address));

        else

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

        end if;

    end process;

end Behavioral;

Simulazione

**tb\_sistema\_complessivo.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity tb\_sistema\_complessivo is

end entity tb\_sistema\_complessivo;

architecture Behavioral of tb\_sistema\_complessivo is

    signal clock : std\_logic := '0';

    signal start : std\_logic := '0';

    signal reset : std\_logic := '0';

begin

    sis\_tot: entity work.sistema\_complessivo

    port map (

        clock => clock,

        start => start,

        reset => reset

    );

    clock <= not clock after 1 ps;

    stimulus: process

    begin

        wait for 2 ps;

        start <= '1';

        wait for 2 ps;

        start <= '0';

        wait;

    end process stimulus;

end architecture Behavioral;

