Capitolo 1: Reti Combinatorie Elementari

Esercizio 2: Sistema ROM+M

Progetto e architettura

Il sistema totale S è realizzato per composizione di due componenti: una memoria di sola lettura di tipo ROM che contiene 16 parole a 8 bit e una macchina di tipo M che trasforma word a 8 bit in word a 4 bit secondo una specifica funzione.

Il sistema S ha un ingresso **address** che è un vettore di 4 bit e un’uscita **result** che è anch’essa un vettore di 4 bit. Tale sistema utilizza due segnali intermedi: il primo è **rom\_out** che è un vettore di 8 bit che rappresenta l’uscita della memoria ROM, mentre il secondo è **m\_out** che è un vettore di 4 bit.

Il vettore **address** va in input alla componente ROM che dà in output il vettore di 8 bit letto in questa posizione (**d\_out**), il quale è poi collegato all’ingresso **x** della macchina M. Essa è una macchina combinatoria elementare che prende i 4 bit meno significativi di x, li nega e li assegna all’uscita **y** da 4 bit. Il segnale m\_out connette l’output del blocco M con l’output dell’intero sistema S.

Immagine che contiene diagramma, schizzo, Disegno tecnico, linea

Descrizione generata automaticamente

Implementazione

L’entity ROM è realizzata attraverso un’architettura comportamentale; invece, la macchina M è progettata da un livello di astrazione dataflow. S presenta chiaramente un’architettura “Structural”.

**S.vhd**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity S is port(

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

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

    );

end entity S;

architecture S\_a of S is

    signal rom\_out  : std\_logic\_vector(7 downto 0);

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

begin

    memory\_rom : entity work.ROM port map(

            address => address,

            d\_out => rom\_out

        );

    machine\_m : entity work.M port map(

            x  => rom\_out,

            y => m\_out

        );

    result <= m\_out;

end architecture S\_a;

**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.numeric\_std.all;

entity M is port(

        x : in  std\_logic\_vector(7 downto 0);

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

    );

end entity M;

architecture Dataflow of M is

begin

    process(x)

    begin

        y(3 downto 0) <= not(x(3 downto 0));

    end process;

end architecture Dataflow;

Simulazione

**M\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity M\_tb is

end M\_tb;

architecture testbench of M\_tb is

    signal x\_tb : STD\_LOGIC\_VECTOR(7 downto 0);

    signal y\_tb : STD\_LOGIC\_VECTOR(3 downto 0);

    component M

        port (

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

            y : out STD\_LOGIC\_VECTOR(3 downto 0)

        );

    end component;

begin

    uut : M

        port map (

            x => x\_tb,

            y => y\_tb

        );

    stimulus : process

    begin

        wait for 10 ns;

        x\_tb <= "00000000";

        wait for 10 ns;

        x\_tb <= "11110000";

        wait for 10 ns;

        x\_tb <= "01010101";

        wait for 10 ns;

        x\_tb <= "11001100";

        wait for 10 ns;

        wait;

    end process stimulus;

end testbench;

Immagine che contiene schermata, linea, testo

Descrizione generata automaticamente

**ROM\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity ROM\_tb is

end ROM\_tb;

architecture testbench of ROM\_tb is

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

    signal d\_out\_tb   : STD\_LOGIC\_VECTOR(7 downto 0);

    component ROM

        port (

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

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

        );

    end component;

begin

    uut : ROM

        port map (

            address => address\_tb,

            d\_out   => d\_out\_tb

        );

    stimulus : process

    begin

        wait for 10 ns;

        address\_tb <= "0000";

        wait for 10 ns;

        address\_tb <= "0010";

        wait for 10 ns;

        address\_tb <= "1101";

        wait for 10 ns;

        address\_tb <= "1011";

        wait for 10 ns;

        wait;

    end process stimulus;

end testbench;

Immagine che contiene schermata, testo, linea, Carattere

Descrizione generata automaticamente

**S\_tb.vhd**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity S\_tb is

end entity S\_tb;

architecture testbench of S\_tb is

    signal address : std\_logic\_vector(3 downto 0) := "0000";

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

    component S port(

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

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

        );

    end component S;

begin

    uut: S port map(

            address => address,

            result  => result

        );

    stimuli : process

    begin

        address <= "0000";

        wait for 10 ns;

        address <= "0001";

        wait for 10 ns;

        address <= "1000";

        wait for 10 ns;

        address <= "1101";

        wait for 10 ns;

        address <= "1111";

        wait for 10 ns;

        wait;

    end process;

    process

    begin

        wait for 50 ns;

        assert result = "0000" report "Errore all'indirizzo 0" severity error;

        assert result = "0001" report "Errore all'indirizzo 1" severity error;

        assert result = "1000" report "Errore all'indirizzo 8" severity error;

        assert result = "1101" report "Errore all'indirizzo 13" severity error;

        assert result = "1111" report "Errore all'indirizzo 15" severity error;

        wait;

    end process;

end architecture testbench;

Immagine che contiene schermata, testo, linea

Descrizione generata automaticamente

Sintesi su board di sviluppo

I primi 4 interruttori (dal pin J15 al pin R15) della FPGA vengono usati per inserire l’indirizzo da cui leggere in memoria (**switch**), mentre i primi 4 led (da H17 a N14) sono impiegati per visualizzare il risultato (**led**), ossia la parola letta trasformata da M.

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { switch[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { switch[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { switch[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { switch[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

## 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]