Capitolo 2: Reti Sequenziali Elementari

Esercizio 6: Sistema di lettura-elaborazione-scrittura PO\_PC

Progetto e architettura

Immagine che contiene testo, diagramma, Piano, linea

Descrizione generata automaticamenteIl sistema richiesto può essere così schematizzato nel seguente modo.

Le componenti dell’architettura rappresentata, nel dettaglio, sono le seguenti:

* **UC**: unità di controllo il cui ruolo principale è quello di fornire i segnali di controllo per il contatore e le memorie
* **ROM**: memoria di sola lettura da cui vengono prelevati i dati di partenza
* **M**: macchina combinatoria che restituisce in uscita il segnale in ingresso ma negato
* **MEM**: memoria in cui vengono immagazzinati i dati finali
* **CONT**: contatore che scandisce le locazioni delle memorie da cui leggere e in cui scrivere

Il modulo **CU** implementa una macchina a stati finiti (FSM) con tre stati (**S0**, **S1**, **S2**) che gestisce il comportamento degli altri componenti. L'entity CU ha tre porte di input (**start**, **reset**, **clk**) e una porta di uscita (**output**), che è un vettore di 4 bit.

All’interno dell’architettura sono dichiarati e istanziati componenti di tipo ROM, M, MEM e cont\_mod\_16. Questi componenti sono collegati in modo appropriato alle porte di input/output di CU e sono interconnessi fra loro attraverso l'utilizzo di segnali interni appositamente creati, ovvero **read\_ROM**, **en\_cont**, **write\_MEM**, **address**, **ROM\_out** e **M\_out**.

Il file “CU.vhd”, quindi, rappresenta il sistema complessivo dato che contiene sia i due processi necessari a implementare il diagramma di stati dell’automa (descritto successivamente) sia tutte le istanze delle entità che sono controllate da questa macchina.

Immagine che contiene testo, diagramma, cerchio, Carattere

Descrizione generata automaticamente

* Stato : Questo è lo stato iniziale in cui si disabilita il contatore, la lettura dalla ROM e la scrittura nella MEM. Si può tornare in questo stato a partire da un qualsiasi altro con un reset e si rimanere in esso finché non si riceve un segnale di start.
* Stato : In questo stato, la macchina abilita il contatore e la lettura dalla ROM.
* Stato : In questo stato si abilita la scrittura nella MEM e poi si fa la transizione verso S1.

Il file "cont\_mod\_16.vhd" definisce un contatore modulo 16, caratterizzato da un segnale di clock (**clk**), un segnale di reset asincrono (**rst**), un segnale di abilitazione (**enable**), e produce un risultato a 4 bit denominato “**count”** che va in ingresso alle due memorie come “**address**”.

Il file “ROM.vhd” rappresenta una memoria di sola lettura (ROM) caratterizzata da una struttura di tipo array a 16 posizioni, ciascuna contenente un vettore di 8 bit. I terminali di ingresso sono il segnale di sincronismo (**clk**), il segnale di lettura (**read**), l'indirizzo in cui leggere (**address**); il segnale di uscita è il vettore a 8 bit dei dati letti (**d\_out**). La memoria in questione contiene valori binari che spaziano da "00000000" in posizione “0000” a "00001111" in posizione “1111”.

Il modulo M rappresenta una macchina puramente combinatoria. Quando essa riceve in ingresso un vettore di 8 bit (nell’architettura l’input è collegato all'uscita della ROM), produce immediatamente, a meno di ritardi di propagazione, un vettore di uscita di 4 bit. L’output corrisponde al negato dei 4 bit meno significativi del vettore in input (nell’architettura l’uscita è connessa all’ingresso della memoria MEM).

La MEM funge da banco di memoria a 16 posizioni, ciascuna contenente un vettore di 4 bit. L'entity presenta cinque porte: in ingresso si ha il segnale di tempificazione (**clk**), l'indirizzo di memoria (**address**), il dato di 4 bit da scrivere (**d\_in**) e il segnale di scrittura (**write**); in uscita si ha l’ultimo dato scritto (**d\_out**).

Implementazione

**CU.vhd**

library IEEE;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity CU is port (

      start : in std\_logic;

      reset: in std\_logic;

      clk : in std\_logic;

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

      );

end CU;

architecture Behavioral of CU is

    component ROM is

    port (

        clk     : in  std\_logic;

        read    : in  std\_logic;

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

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

    );

    end component;

    component M is port(

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

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

    );

    end component;

    component MEM is

    port (

        clk     : in  std\_logic;

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

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

        write   : in  std\_logic;

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

    );

    end component;

    component cont\_mod\_16

    port (

        clk     : in  std\_logic;

        rst     : in  std\_logic;

        enable  : in  std\_logic;

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

    );

    end component;

    signal read\_ROM : std\_logic :='0';

    signal en\_cont : std\_logic :='0';

    signal write\_MEM : std\_logic :='0';

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

    signal ROM\_out : std\_logic\_vector(7 downto 0) :=(others=>'0');

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

    type stato is (S0,S1,S2);

    signal stato\_corrente : stato := S0;

    signal stato\_prossimo : stato;

begin

    mem\_rom: ROM port map(

        clk => clk,

        read => read\_ROM,

        address => address,

        d\_out => ROM\_out

    );

    machine\_m: M port map(

        x => ROM\_out,

        y => M\_out

    );

    mem\_mem: MEM port map(

        clk => clk,

        address => address,

        d\_in => M\_out,

        write => write\_MEM,

        d\_out => output

    );

    contatore: cont\_mod\_16

        port map (

            clk => clk,

            rst => reset,

            enable => en\_cont,

            count => address);

    process\_1 : process(stato\_corrente,start,reset)

    begin

        case stato\_corrente is

            when S0 =>

                en\_cont <= '0';

                read\_ROM <= '0';

                write\_MEM <= '0';

                if reset='1' then

                    stato\_prossimo <= S0;

                elsif(start='1') then

                    stato\_prossimo <= S1;

                else stato\_prossimo <= S0;

                end if;

            when S1 =>

                 if(reset='1') then --

                    stato\_prossimo <= S0;

                 else en\_cont <= '1';

                    read\_ROM <= '1';

                    stato\_prossimo <= S2;

                 end if;

            when S2 =>

                write\_MEM <= '1';

                if(reset='1') then

                    stato\_prossimo <= S0;

                else stato\_prossimo <= S1;

                end if;

        end case;

    end process;

    process\_2 : process(clk)

    begin

        if rising\_edge(clk) then

            stato\_corrente <= stato\_prossimo;

        end if;

    end process;

end Behavioral;

**cont\_mod\_16.vhd**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity cont\_mod\_16 is

    port (

        clk     : in  std\_logic;

        rst     : in  std\_logic;

        enable  : in  std\_logic;

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

    );

end entity cont\_mod\_16;

architecture Behavioral of cont\_mod\_16 is

    signal counter\_value : unsigned(3 downto 0) := (others => '0');

begin

    process (clk, rst)

    begin

        if rst = '1' then

            counter\_value <= (others => '0');

        elsif rising\_edge(clk) and enable = '1' then

            if counter\_value = 15 then

                counter\_value <= (others => '0');

            else

                counter\_value <= counter\_value + 1;

            end if;

        end if;

    end process;

    count <= std\_logic\_vector(counter\_value);

end architecture Behavioral;

**ROM.vhd**

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;

        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"

    );

    signal data : std\_logic\_vector(7 downto 0) := (others => '0');

begin

    process (clk)

    begin

        if rising\_edge(clk) then

            if read='1' then

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

            end if;

        end if;

    end process;

    d\_out <= data;

end architecture RTL;

**M.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 Behavioral of M is

begin

    process(x)

    begin

        y <= (others => '0');

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

    end process;

end architecture behavioral;

**MEM.vhd**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity MEM is

    port (

        clk     : in  std\_logic;

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

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

        write   : in  std\_logic;

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

    );

end entity MEM;

architecture RTL of MEM is

    type MEMORY\_16\_4 is array (0 to 15) of std\_logic\_vector(3 downto 0);

    signal mem\_data : MEMORY\_16\_4 := (others => (others => '0'));

begin

    process (clk)

    begin

        if rising\_edge(clk) then

            if write = '1' then

                mem\_data(to\_integer(unsigned(address))) <= d\_in;

                d\_out <= d\_in;

            else

                d\_out <= "0000";

            end if;

        end if;

    end process;

end architecture RTL;

Simulazione

Immagine che contiene schermata, Policromia, Software multimediale, software

Descrizione generata automaticamenteImmagine che contiene schermata, Policromia, Software multimediale, software

Descrizione generata automaticamenteNella prima fase, al colpo di clock successivo all’alzarsi del segnale di start, vengono abilitati il contatore e la ROM. La macchina combinatoria M calcola il valore di uscita, invertendo il suo ingresso, e successivamente il dato ricavato viene scritto in memoria MEM abilitata tramite il segnale WRITE\_MEM (0🡪1).

Nella seconda fase invece, viene posto ad 1 il segnale di RESET. In questo caso, tutti i valori vengono portati a 0 e vengono “spente” le memorie e il contatore.

Immagine che contiene schermata

Descrizione generata automaticamenteImmagine che contiene schermata

Descrizione generata automaticamente

**CU\_tb.vhd**

library IEEE;

use ieee.std\_logic\_1164.all;

entity CU\_tb is

end entity CU\_tb;

architecture testbench of CU\_tb is

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

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

  component CU

    port (

      start  : in std\_logic;

      reset  : in std\_logic;

      clk    : in std\_logic;

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

    );

  end component;

begin

  clock: process

  begin

    while now < 1000 ns loop

      clk\_tb <= not clk\_tb after 2.5 ns;

      wait for 5 ns;

    end loop;

    wait;

  end process;

  stim : process

  begin

    wait for 10 ns;

    start\_tb <= '1';

    wait for 20 ns;

    wait for 80 ns;

    start\_tb <= '0';

    wait for 100 ns;

    start\_tb <= '1';

    wait for 20 ns;

    start\_tb <= '0';

    wait for 100 ns;

    reset\_tb <= '1';

    wait for 20 ns;

    reset\_tb <= '0';

    wait for 300 ns;

    start\_tb <= '1';

    wait for 20 ns;

    start\_tb <= '0';

    wait;

  end process;

  dut: CU

    port map (

      start  => start\_tb,

      reset  => reset\_tb,

      clk    => clk\_tb,

      output => output\_tb

    );

end architecture testbench;

Sintesi su board di sviluppo

Premendo il pustante **read** associato al bottone **M18** si fa partire l’elaborazione; il risultato del sistema ovvero il dato scritto in MEM viene riportato, in forma binaria, sui primi 4 **led** ovvero quelli associati ai pin che vanno da **H17** a **N14**. Il pulsante di **reset** **P17** è direttamente collegato al porto di ingresso "reset" della componente CU che è stata istanziata (**c\_u**). Infine, è da notare che per facilitare la visualizzazione dei vari valori sui LED, è stato generato, attraverso un process, un nuovo segnale di clock denominato "**clock**” il cui periodo è 0.5s (mezzo secondo) a partire da quello della scheda (**clk**) che ha T=10ns. Il segnale clock è quello che viene messo input a c\_u.

**CU\_on\_board.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity CU\_on\_board is Port (

    clk : in std\_logic;

    read : in std\_logic;

    reset : in std\_logic;

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

end CU\_on\_board;

architecture Behavioral of CU\_on\_board is

    component CU is port (

          start : in std\_logic;

          reset: in std\_logic;

          clk : in std\_logic;

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

          );

    end component;

    signal counter : integer := 0;

    signal clock : std\_logic := '0'; -- mezzo secondo

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

begin

    c\_u : CU port map(

        start => read,

        reset => reset,

        clk => clock,

        output => out\_put);

    led <= out\_put;

    process(clk)

    begin

        if rising\_edge(clk) then

            counter <= counter + 1;

            if counter = (50\_000\_000)/2 then

                counter <= 0;

                clock <= not clock;

            end if;

        end if;

    end process;

end Behavioral;

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

## 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]

##Buttons

set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { read }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { reset }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

Timing analysis

La timing analysis di un circuito digitale permette di valutare se il progetto rispetta i vincoli temporali richiesti per garantire un corretto funzionamento del sistema. I constraint temporali sono specificati sotto forma di comandi **TCL** (Tool Command Language) all’interno di un file di tipo .XDC (Xilinx Design Constraints).

Per stabilire vincoli sull’unico clock utilizzato in questo design sono stati scritti i seguenti comandi.

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

In questo caso, si è creato un “generated clock” (non virtuale) denominato **clk**, collegato al pin E3 della FPGA, il cui periodo (**-period**) è 10 nanosecondi e il duty cycle è del 50% (**-waveform**). Nel dettaglio “-waveform” consente di specificare gli istanti in ns in cui si verificano i fronti di salita e discesa all'interno di un singolo periodo del clock.

Questo segnale di sincronismo è proprio quello che viene messo in ingresso alle componenti cont\_mod\_16, CU, MEM e ROM quando si esegue la sintesi sul board tramite CU\_on\_board.

Dato questo vincolo, è possibile stimare approssimativamente la frequenza di funzionamento attuale a partire dal periodo del clock (T) e dal valore del **WNS** (Worst Negative Slack). Col termine "slack" ci si riferisce al margine temporale disponibile tra il ritardo richiesto e il ritardo effettivo in percorso all'interno del circuito. Il Worst Negative Slack rappresenta il valore minimo di questa misura nei percorsi critici del design. In altre parole, indica il delay massimo entro il quale il cammino critico deve essere “completato” affinché il sistema soddisfi i requisiti di temporizzazione; quindi, quando il WNS è negativo si potrebbero osservare comportamenti incorretti.

Il WNS può essere ottenuto consultando l'interfaccia "Design Timing Summary" nella sezione "Implementation" dell'applicazione Vivado.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

In questo momento il WNS è uguale a 6.179 ns; la frequenza di funzionamento del circuito è ricavata dall’esspressione . Per determinare la frequenza massima, si riduce gradualmente il periodo fino a quando il Worst Negative Slack (WNS) rimane positivo.

create\_clock -add -name sys\_clk\_pin -period 06.00 -waveform {0 3} [get\_ports {clk}];

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

create\_clock -add -name sys\_clk\_pin -period 04.00 -waveform {0 2} [get\_ports {clk}];

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

create\_clock -add -name sys\_clk\_pin -period 03.00 -waveform {0 1.5} [get\_ports {clk}];

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

create\_clock -add -name sys\_clk\_pin -period 03.50 -waveform {0 1.75} [get\_ports {clk}];

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

La massima frequenza di funzionamento del sistema digitale in questione è quindi