Capitolo 2: Reti Sequenziali

Esercizio 4: Shift Register

Progetto e architettura

Il progetto prevede di implementare un registro a scorrimento di N bit in grado di shiftare il vettore a destra o sinistra di 1 o 2 posizioni. Si realizza il design sia da un livello di astrazione comportamentale che strutturale.

I parametri in ingresso al sistema includono, oltre al clock e al reset, il segnale **dir,** che indica se deve essere eseguito un left o un right shift, e il segnale **Y,** che specifica se il vettore in ingresso da N bit (**din**) deve essere caricato nello shift register in modalità parallela o se deve essere eseguito uno shift di 1 o 2 posizioni. Il generic **N** specifica la dimensione dello shift register; l’impiego di questo generic consente di parametrizzare il modulo in modo che esso possa essere riutilizzato per dati di diverse dimensioni. In ogni istante, i valori memorizzati nello shift register possono essere letti in parallelo tramite l’uscita **dout** da N bit.

Nell'architettura **Behavioral**, il comportamento del registro è descritto attraverso un processo sensibile al fronte di salita del segnale di clock (**rising\_edge(clk)**) e al segnale di reset (**rst**). Il reset è quindi asincrono.

L’operazione da eseguire è definita da una combinazione dei valori degli input Y e dir:

* Y = “00”: caricamento parallelo
* Y = “01”: shift di 1 posizione
  + dir = “0”: shift a destra
  + dir = “1”: shift a sinistra
* Y = “10” shift di 2 posizioni.
  + dir = “0”: shift a destra
  + dir = “1”: shift a sinistra

Per implementare ciò, vengono utilizzati sia il costrutto if...then...else che il costrutto case.

Nell'architettura **Structural**, il registro viene implementato per composizione di componenti di tipo **flip\_flop\_d** e **mux\_8\_1**. Il modulo flip\_flop\_d rappresenta un flip-flop D edge-triggered sul fronte di salita, mentremux\_8\_1 rappresenta un multiplexer 8 a 1. Il progetto prevede l'uso di N multiplexer e flip-flop opportunamente interconnessi tra loro: ogni multiplexer, in base a un segnale di selezione comune a tutti, imposta uno specifico bit in ingresso al flip-flop che lo segue.

Nella prossima immagine sono stati evidenziati soltanto i tipici collegamenti di un i-esimo multiplexer/flip-flop**,** con “i” compreso tra 2 e N-3. Grazie alla forte ripetizione, l’istanziazione di queste componenti nello shifter è svolta attraverso costrutti **for…generate**. Invece la realizzazione dei primi due stadi (i=0 e i=1) e degli ultimi due (i=N-2 e i=N-1) deve essere fatta a parte. Lo shift register così progettato richiede, quindi, che N sia come minimo 4.

A seconda del segnale di selezione condiviso **ctr**, il cui valore dipende da rst, Y e dir, l’ingresso del flip flop D i-esimo è uno dei seguenti:

* **din(i)** (nel caso delcaricamento parallelo)
* **flip\_flop\_out(i+1)** (nel caso delLeft Shift di 1 posizione)
* **flip\_flop\_out(i-1)** (nel caso del Right Shift di 1 posizione)
* **flip\_flop\_out(i+2)** (nel caso del Left Shift di 2 posizioni)
* **flip\_flop\_out(i-2)** (nel caso delRight Shift di 2 posizioni)

Immagine che contiene diagramma, Piano, linea, Disegno tecnico

Descrizione generata automaticamente

I porti dell'entity **mux\_8\_1** sono rappresentati da 8 bit in ingresso (da **input\_0** a **input\_7**), un segnale di controllo da 3 bit (**control**), sempre in input, e da un bit in uscita (**output**). L’architettura è comportamentale e include un processo sensibile ai cambiamenti degli input. Utilizzando una struttura case, il processo seleziona l'output in base al valore del segnale di controllo; ad esempio, se il segnale di controllo è "000", l'output sarà input\_0, e così via.

L'entity **flip\_flop\_d** presenta come ingressi i segnali **clk**, **reset** e **d** e come uscita il bit **q**. Nell'architecture “Behavioral”, c'è un processo la cui sensitivity list è composta dal segnale di clock (clk) e da quello di reset (reset). Se il segnale di reset è '1', l'uscita viene impostata a '0'; in caso contrario, all'evento di fronte di salita (**rising\_edge**) di clk, l'uscita assume il valore dell'input d. Il flip-flop D edge-triggered rappresenta uno degli elementi di memoria (1 bit) più comunemente usati nei circuiti digitali.

Implementazione

**shift\_register.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity shift\_register is

    generic (

        N : integer := 8

    );

    port (

        clk      : in  std\_logic;

        rst      : in  std\_logic;

        dir      : in  std\_logic; -- 0: destra, 1: sinistra

        Y        : in  std\_logic\_vector(1 downto 0); -- 00: load, 01: shifta di 1 posizione, 10: shifta di 2 posizioni

        din      : in  std\_logic\_vector(N-1 downto 0);

        dout     : out std\_logic\_vector(N-1 downto 0)

    );

end entity shift\_register;

architecture Behavioral of shift\_register is

    signal reg : std\_logic\_vector(N-1 downto 0);

begin

    process(clk, rst)

    begin

        if rst = '1' then

            reg <= (others => '0');

        elsif rising\_edge(clk) then

            case Y is

                when "00" => -- load

                    reg <= din;

                when "01" => -- una posizione

                    case dir is

                        when '0' => reg <= '0' & reg(N-1 downto 1); -- destra

                        when '1' => reg <= reg(N-2 downto 0) & '0' ; -- sinistra

                        when others => reg <= (others=>'0');

                    end case;

                when "10" => -- due posizioni

                    case dir is

                        when '0' => reg <= "00" & reg(N-1 downto 2); -- destra

                        when '1' => reg <= reg(N-3 downto 0) & "00"; -- sinistra

                        when others => reg <= (others=>'0');

                    end case;

                when others =>

                    reg <= (others=>'0');

            end case;

        end if;

    end process;

    dout <= reg;

end architecture Behavioral;

architecture Structural of shift\_register is

    component flip\_flop\_d

    port ( clk    : in STD\_LOGIC;

           reset  : in STD\_LOGIC;

           d      : in STD\_LOGIC;

           q      : out STD\_LOGIC);

    end component;

    component mux\_8\_1

    port (

        input\_0 : in STD\_LOGIC;

        input\_1 : in STD\_LOGIC;

        input\_2 : in STD\_LOGIC;

        input\_3 : in STD\_LOGIC;

        input\_4 : in STD\_LOGIC;

        input\_5 : in STD\_LOGIC;

        input\_6 : in STD\_LOGIC;

        input\_7 : in STD\_LOGIC;

        control : in STD\_LOGIC\_VECTOR(2 downto 0);

        output : out STD\_LOGIC

    );

    end component;

    signal mux\_out: std\_logic\_vector(N-1 downto 0) ;

    signal flip\_flop\_out: std\_logic\_vector(N-1 downto 0);

    signal ctr: std\_logic\_vector(2 downto 0);

    signal dir\_new: std\_logic;

begin

    ctr <= "101" when rst='1' else

           "000" when Y="00" else

           "001" when dir='0' and Y="01" else

           "010" when dir='1' and Y="01" else

           "011" when dir='0' and Y="10" else

           "100" when dir='1' and Y="10";

    flip\_flop\_gen: for i in 0 to N-1 generate

        flip\_flop: flip\_flop\_d port map(

            clk => clk,

            reset => rst,

            d => mux\_out(i),

            q => flip\_flop\_out(i)

        );

    end generate flip\_flop\_gen;

    mux\_0: mux\_8\_1 port map(

        input\_0 =>  din(0), -- caricamento

        input\_1 => flip\_flop\_out(1), -- shift di una posizione a sinistra

        input\_2 => '0', -- shift di una posizione a destra

        input\_3 => flip\_flop\_out(2), -- shift di due posizioni a sinistra

        input\_4 => '0', -- shift di due posizioni a destra

        input\_5 => '0', -- reset

        input\_6 => '0', -- non usato

        input\_7 => '0', -- non usato

        control => ctr,

        output => mux\_out(0)

    );

    mux\_1: mux\_8\_1 port map(

        input\_0 =>  din(1),

        input\_1 => flip\_flop\_out(2),

        input\_2 => flip\_flop\_out(0),

        input\_3 => flip\_flop\_out(3),

        input\_4 => '0',

        input\_5 => '0',

        input\_6 => '0',

        input\_7 => '0',

        control => ctr,

        output => mux\_out(1)

    );

    mux\_N\_1: mux\_8\_1 port map(

        input\_0 =>  din(N-1),

        input\_1 => '0',

        input\_2 => flip\_flop\_out(N-2),

        input\_3 => '0',

        input\_4 => flip\_flop\_out(N-3),

        input\_5 => '0',

        input\_6 => '0',

        input\_7 => '0',

        control => ctr,

        output => mux\_out(N-1)

    );

    mux\_N\_2: mux\_8\_1 port map(

        input\_0 =>  din(N-2),

        input\_1 => flip\_flop\_out(N-1),

        input\_2 => flip\_flop\_out(N-3),

        input\_3 => '0',

        input\_4 => flip\_flop\_out(N-4),

        input\_5 => '0',

        input\_6 => '0',

        input\_7 => '0',

        control => ctr,

        output => mux\_out(N-2)

    );

    mux\_gen: for i in 2 to N-3 generate

        mux\_interni: mux\_8\_1 port map(

            input\_0 =>  din(i),

            input\_1 => flip\_flop\_out(i+1),

            input\_2 => flip\_flop\_out(i-1),

            input\_3 => flip\_flop\_out(i+2),

            input\_4 => flip\_flop\_out(i-2),

            input\_5 => '0',

            input\_6 => '0',

            input\_7 => '0',

            control => ctr,

            output => mux\_out(i)

        );

    end generate mux\_gen;

    dout <= flip\_flop\_out;

end Structural;

**mux\_8\_1.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity mux\_8\_1 is

    Port (

        input\_0 : in STD\_LOGIC;

        input\_1 : in STD\_LOGIC;

        input\_2 : in STD\_LOGIC;

        input\_3 : in STD\_LOGIC;

        input\_4 : in STD\_LOGIC;

        input\_5 : in STD\_LOGIC;

        input\_6 : in STD\_LOGIC;

        input\_7 : in STD\_LOGIC;

        control : in STD\_LOGIC\_VECTOR(2 downto 0);

        output : out STD\_LOGIC

    );

end mux\_8\_1;

architecture Behavioral of mux\_8\_1 is

begin

    process (input\_0, input\_1, input\_2, input\_3, input\_4, input\_5, input\_6, input\_7, control)

    begin

        case control is

            when "000" =>

                output <= input\_0;

            when "001" =>

                output <= input\_1;

            when "010" =>

                output <= input\_2;

            when "011" =>

                output <= input\_3;

            when "100" =>

                output <= input\_4;

            when "101" =>

                output <= input\_5;

            when "110" =>

                output <= input\_6;

            when "111" =>

                output <= input\_7;

            when others =>

                output <= '0';

        end case;

    end process;

end Behavioral;

**flip\_flop\_d.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity flip\_flop\_d is Port (

    clk    : in STD\_LOGIC;

    reset  : in STD\_LOGIC;

    d      : in STD\_LOGIC;

    q      : out STD\_LOGIC);

end flip\_flop\_d;

architecture Behavioral of flip\_flop\_d is

    signal q\_out : STD\_LOGIC;

begin

    process(clk, reset)

    begin

        if reset = '1' then

            q\_out <= '0';

        elsif rising\_edge(clk) then

            q\_out <= d;

        end if;

    end process;

    q <= q\_out;

end Behavioral;

Simulazione

Si sono creati due testbench, uno dedicato alla simulazione dello shift register realizzato a partire dall'architecture comportamentale (**shift\_register\_behavioral \_tb**) e uno per testare l'architettura strutturale (**shift\_register\_structural \_tb**). Sono stati scelti due diversi valori per il parametro N (8 nel primo caso, 12 nel secondo).

**shift\_register\_structural \_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity shift\_register\_structural\_tb is

end entity shift\_register\_structural\_tb;

architecture testbench of shift\_register\_structural\_tb is

    constant N : integer := 12;

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

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

    signal shift\_dir: std\_logic := '0';

    signal shift\_amp: std\_logic\_vector(1 downto 0) := "00";

    signal din      : std\_logic\_vector(N-1 downto 0) := (others => '0');

    signal dout     : std\_logic\_vector(N-1 downto 0) := (others => '0');

    constant clk\_period : time := 10 ns;

    component shift\_register

        generic (

            N : integer := N

        );

        port (

            clk      : in  std\_logic;

            rst      : in  std\_logic;

            shift\_dir: in  std\_logic;

            shift\_amp: in  std\_logic\_vector(1 downto 0);

            din      : in  std\_logic\_vector(N-1 downto 0);

            dout     : out std\_logic\_vector(N-1 downto 0)

        );

    end component;

begin

    dut: entity work.shift\_register(Structural)

        generic map (N=>N)

        port map (

            clk      => clk,

            rst      => rst,

            dir      => shift\_dir,

            Y        => shift\_amp,

            din      => din,

            dout     => dout

        );

    CLK\_process :process

    begin

        clk <= '0';

        wait for clk\_period/2;

        clk <= '1';

        wait for clk\_period/2;

    end process;

    process

    begin

        din <= "110101100111";

        shift\_amp <= "00"; -- load

        wait for 10 ns;

        shift\_dir <= '0'; -- shift a destra di...

        shift\_amp <= "01"; -- ...1 posizione

        wait for 30 ns;

        shift\_amp <= "10"; -- ...2 posizioni

        wait for 30 ns;

        din <= "010101110010";

        shift\_amp <= "00"; -- load

        wait for 10 ns;

        shift\_dir <= '1'; -- shift a sinistra di...

        shift\_amp <= "01"; -- ...1 posizione

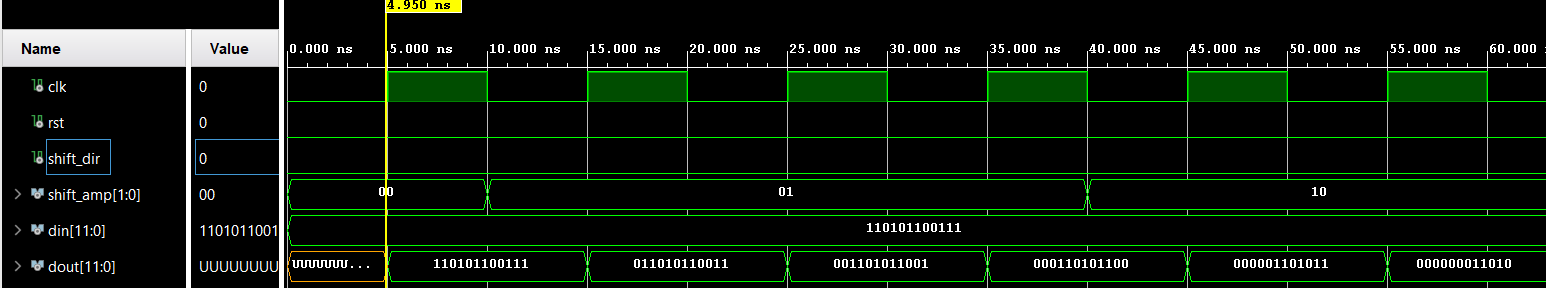
        wait for 30 ns;

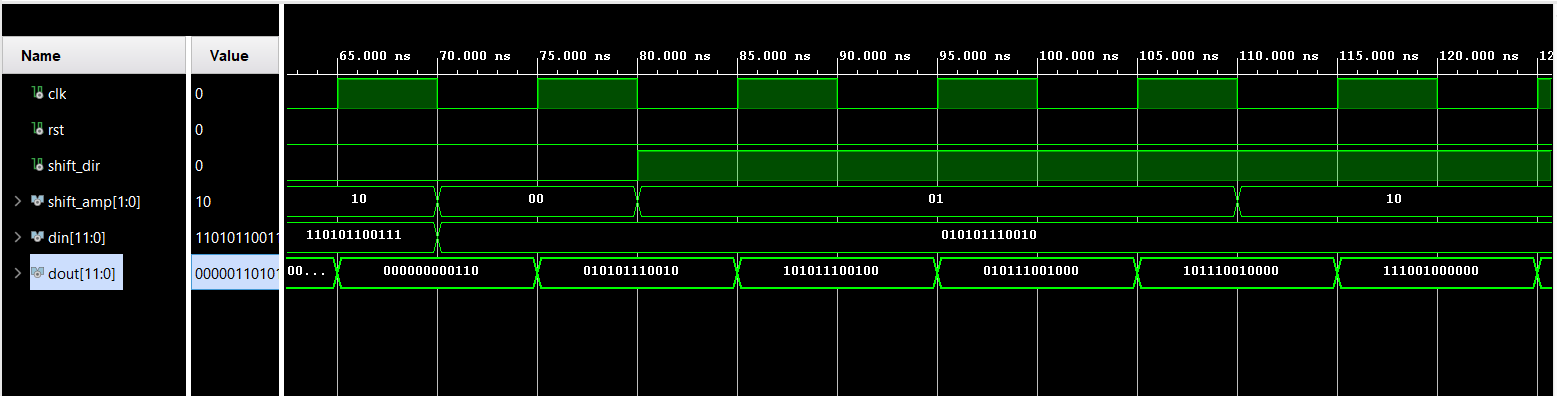
        shift\_amp <= "10"; -- ...2 posizioni

        wait;

    end process;

end architecture testbench;





**shift\_register\_behavioral\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity shift\_register\_behavioral\_tb is

end entity shift\_register\_behavioral\_tb;

architecture testbench of shift\_register\_behavioral\_tb is

    constant N : integer := 8;

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

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

    signal shift\_dir: std\_logic := '0';

    signal shift\_amp: std\_logic\_vector(1 downto 0) := "00";

    signal din      : std\_logic\_vector(N-1 downto 0) := (others => '0');

    signal dout     : std\_logic\_vector(N-1 downto 0) := (others => '0');

    constant clk\_period : time := 10 ns;

    component shift\_register

        generic (

            N : integer := N

        );

        port (

            clk      : in  std\_logic;

            rst      : in  std\_logic;

            shift\_dir: in  std\_logic;

            shift\_amp: in  std\_logic\_vector(1 downto 0);

            din      : in  std\_logic\_vector(N-1 downto 0);

            dout     : out std\_logic\_vector(N-1 downto 0)

        );

    end component;

begin

    dut: entity work.shift\_register(Behavioral)

        generic map (N=>N)

        port map (

            clk      => clk,

            rst      => rst,

            dir      => shift\_dir,

            Y        => shift\_amp,

            din      => din,

            dout     => dout

        );

    CLK\_process :process

    begin

        clk <= '0';

        wait for clk\_period/2;

        clk <= '1';

        wait for clk\_period/2;

    end process;

    process

    begin

        din <= "11010110";

        shift\_amp <= "00"; -- load

        wait for 10 ns;

        shift\_dir <= '0'; -- shift a destra di...

        shift\_amp <= "01"; -- ...1 posizione

        wait for 30 ns;

        shift\_amp <= "10"; -- ...2 posizioni

        wait for 30 ns;

        din <= "01010111";

        shift\_amp <= "00"; -- load

        wait for 10 ns;

        shift\_dir <= '1'; -- shift a sinistra di...

        shift\_amp <= "01"; -- ...1 posizione

        wait for 30 ns;

        shift\_amp <= "10"; -- ...2 posizioni

        wait;

    end process;

end architecture testbench;

