Capitolo 7: switch multistadio

Esercizio 11: switch multistadio

Progetto e architettura

Il circuito digitale complessivo è formato da due componenti principali: **gestione\_priorità** e **rete\_di\_interconnesione**.

Il sistema ha quattro ingressi di abilitazione da un bit (**en0**, **en1**, **en2**, **en3**) e quattro ingressi di dati da 6 bit (**in0**, **in1**, **in2**, **in3**). Inoltre, ci sono quattro uscite di dati (**out0**, **out1**, **out2**, **out3**) di larghezza 2 bit ciascuna.

L'architettura del sistema, ossia come i vari blocchi sono collegati alle porte di ingresso e uscita e interconnessi tra loro attraverso segnali interni, è illustrata nel diagramma sottostante.

Immagine che contiene diagramma, schizzo, testo, Disegno tecnico

Descrizione generata automaticamente

Il modulo gestione\_priorita è stato progettato per selezionare la “trasmissione” con la priorità più alta tra quattro possibilità (**A\_1**, **A\_2**, **A\_3**, **A\_4**). I primi due bit di ciascun ingresso indicano la sorgente della trasmissione, i successivi due la destinazione, e gli ultimi due contengono i dati da trasmettere. In base ai quattro segnali di abilitazione in ingresso (**en\_1**, **en\_2**, **en\_3**, **en\_4**), il modulo “seleziona” uno tra gli input disponibili definendo i valori degli output. Le uscite di gestione\_priorita sono i segnali di sorgente (**s**) e destinazione (**d**) e i quattro segnali di dati (**B\_1**, **B\_2**, **B\_3**, **B\_4**), tutti da due bit, che sono in entrata alla rete di interconnessione.

In particolare, all’interno dell’architettura “Behavioral” viene creato il segnale “**k**” che rappresenta la priorità decrescente. I segnali s, d, B\_1, B\_2, B\_3 e B\_4 sono definiti proprio in base a questa k. In dettaglio, il valore di k è "00" quando è attivo il segnale di abilitazione en\_1. Nel caso in cui en\_1 sia basso, k è “01" se è attivo en\_2. Se sia en\_1 che en\_2 sono bassi, k assume il valore "10" se è attivo en\_3, altrimenti è uguale a "11" se è attivo en\_4. Infine, se nessuna delle condizioni precedenti è verificata, il valore di k è impostato su "--" (don't care). Assegnazioni di questo tipo sono rese possibili da costrutto **when…else** (assegnazione concorrente condizionale).

La rete di interconnessione non è altro che una "omega network". Questa rete multistadio è composta da quattro switch elementari (**switch\_elem**) opportunamente collegati tra di loro, ciascuno avente due ingressi (**X\_1**, **X\_2**) e due uscite (**Y\_1**, **Y\_2**). L'obiettivo è quello di instradare un segnale a due bit dalla una delle quattro sorgenti (**S\_1**, **S\_2**, **S\_3**, **S\_4**) a una delle quattro destinazioni (**D\_1**, **D\_2**, **D\_3**, **D\_4**) in base ai valori di due segnali di selezione (**sorgente**, **destinazione**) di lunghezza 2.

I quattro switch sono collegati tra di loro e ai porti di I/O della rete in modo conforme all’algoritmo di “**perfect shuffling**”, ovvero le interconnessioni risultati prevedono due livelli di switch e seguono uno schema specifico che alterna gli indirizzi sorgente/destinazione. In altre parole, gli ingressi vengono instradati attraverso due switch iniziali, i cui risultati intermedi vengono poi opportunamente inoltrati attraverso altri due switch per raggiungere le uscite desiderate. La specifica disposizione delle connessioni tra gli switch elementari può essere osservata nell'immagine successiva.

Immagine che contiene schizzo, diagramma, disegno, Disegno tecnico

Descrizione generata automaticamente

Lo switch elementare (switch\_elem) è realizzato combinando un multiplexer 2 a 1 (**mux\_2\_1**) e un demultiplexer 1 a 2 (**demux\_1\_2**) in serie. L'entity definisce quattro ingressi (**X\_1**, **X\_2**, **src**, **dest**) e due uscite (**Y\_1**, **Y\_2**): X\_1 e X\_2 sono i bit provenienti da due possibili sorgenti, src è il segnale di selezione della sorgente, dest della destinazione e Y\_1 e Y\_2 sono singoli segnali connessi alle due potenziali destinazioni. Il segnale interno mux\_out\_demux\_in mette in collegamento l’uscita del multiplexer con l’ingresso del demultiplexer. La struttura completa di switch\_elem è illustrata di seguito.

Immagine che contiene diagramma, linea, Disegno tecnico, schizzo

Descrizione generata automaticamente

Implementazione

**mux\_2\_1.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity mux\_2\_1 is

    Port (d\_in\_1 : in std\_logic\_vector(1 downto 0);

        d\_in\_2 : in std\_logic\_vector(1 downto 0);

        sel : in std\_logic;

        d\_out : out std\_logic\_vector(1 downto 0));

end mux\_2\_1;

architecture Dataflow of mux\_2\_1 is

begin

    d\_out <= d\_in\_1 when sel='0' else

            d\_in\_2 when sel='1' else

            "--";

end Dataflow;

**demux\_1\_2.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity demux\_1\_2 is

    Port (d\_in : in std\_logic\_vector(1 downto 0);

        sel : in std\_logic;

        d\_out\_1 : out std\_logic\_vector(1 downto 0);

        d\_out\_2 : out std\_logic\_vector(1 downto 0));

end demux\_1\_2;

architecture Dataflow of demux\_1\_2 is

begin

    d\_out\_1 <= d\_in when sel='0' else "--";

    d\_out\_2 <= d\_in when sel='1' else "--";

end Dataflow;

**switch\_elem.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity switch\_elem is

    Port (X\_1, X\_2 : in std\_logic\_vector(1 downto 0);

        src, dest : in std\_logic;

        Y\_1, Y\_2 : out std\_logic\_vector(1 downto 0));

end switch\_elem;

architecture Structural of switch\_elem is

    component mux\_2\_1 is

    Port (d\_in\_1 : in std\_logic\_vector(1 downto 0);

        d\_in\_2 : in std\_logic\_vector(1 downto 0);

        sel : in std\_logic;

        d\_out : out std\_logic\_vector(1 downto 0));

    end component;

    component demux\_1\_2 is

    Port (d\_in : in std\_logic\_vector(1 downto 0);

        sel : in std\_logic;

        d\_out\_1 : out std\_logic\_vector(1 downto 0);

        d\_out\_2 : out std\_logic\_vector(1 downto 0));

    end component;

    signal mux\_out\_demux\_in : std\_logic\_vector(1 downto 0);

begin

    multiplexer\_2\_1 : mux\_2\_1 port map(d\_in\_1 => X\_1, d\_in\_2 => X\_2, sel => src, d\_out => mux\_out\_demux\_in);

    demultiplexer\_1\_2 : demux\_1\_2 port map(d\_in => mux\_out\_demux\_in, sel => dest, d\_out\_1 => Y\_1, d\_out\_2 => Y\_2);

end Structural;

**gestione\_priorita.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity gestione\_priorita is

    Port (A\_1, A\_2, A\_3, A\_4 : in std\_logic\_vector(5 downto 0); -- i primi due bit determinano la sorgente, successivi due la destinazione, gli ultimi due i dati da trasmettere

    en\_1, en\_2, en\_3, en\_4 : in std\_logic;

    s, d : out std\_logic\_vector(1 downto 0);

    B\_1, B\_2, B\_3, B\_4 : out std\_logic\_vector(1 downto 0));

end gestione\_priorita;

architecture Behavioral of gestione\_priorita is

    signal k : std\_logic\_vector(1 downto 0); -- priorita’

    signal temp : std\_logic\_vector(1 downto 0);

begin

    k <= "00" when en\_1='1' else

        "01" when en\_2='1' else

        "10" when en\_3='1' else

        "11" when en\_4='1' else

        "--";

    s <= A\_1 (5 downto 4) when k ="00" else

        A\_2 (5 downto 4) when k ="01" else

        A\_3 (5 downto 4) when k ="10" else

        A\_4 (5 downto 4) when k ="11" else

        "--";

    d <= A\_1 (3 downto 2) when k ="00" else

        A\_2 (3 downto 2) when k ="01" else

        A\_3 (3 downto 2) when k ="10" else

        A\_4 (3 downto 2) when k ="11" else

        "--";

    temp <= A\_1(1 downto 0) when k="00" else

         A\_2(1 downto 0) when k="01" else

         A\_3(1 downto 0) when k="10" else

         A\_4(1 downto 0) when k="11" else

         "--";

    B\_1 <= temp when k = "00" else "--";

    B\_2 <= temp when k = "01" else "--";

    B\_3 <= temp when k = "10" else "--";

    B\_4 <= temp when k = "11" else "--";

end Behavioral;

**rete\_di\_interconnesione.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity rete\_di\_interconnesione is -- omega network

    Port (S\_1, S\_2, S\_3, S\_4 : in std\_logic\_vector(1 downto 0);

        D\_1, D\_2, D\_3, D\_4 : out std\_logic\_vector(1 downto 0);

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

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

end rete\_di\_interconnesione;

architecture Behavioral of rete\_di\_interconnesione is

    component switch\_elem is

    Port (X\_1, X\_2 : in std\_logic\_vector(1 downto 0);

        src, dest : in std\_logic;

        Y\_1, Y\_2 : out std\_logic\_vector(1 downto 0));

    end component;

    signal temp\_1, temp\_2, temp\_3, temp\_4 : std\_logic\_vector(1 downto 0);

begin

    switch\_1 : switch\_elem port map (

        X\_1 => S\_1,

        X\_2 => S\_2,

        src => sorgente(0),

        dest => destinazione(1),

        Y\_1 => temp\_1,

        Y\_2 => temp\_2

    );

    switch\_2 : switch\_elem port map (

        X\_1 => S\_3,

        X\_2 => S\_4,

        src => sorgente(0),

        dest => destinazione(1),

        Y\_1 => temp\_3,

        Y\_2 => temp\_4

    );

    switch\_3 : switch\_elem port map (

        X\_1 => temp\_1,

        X\_2 => temp\_3,

        src => sorgente(1),

        dest => destinazione(0),

        Y\_1 => D\_1,

        Y\_2 => D\_2

    );

    switch\_4 : switch\_elem port map (

        X\_1 => temp\_2,

        X\_2 => temp\_4,

        src => sorgente(1),

        dest => destinazione(0),

        Y\_1 => D\_3,

        Y\_2 => D\_4

    );

end Behavioral;

**sistema\_totale.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity sistema\_totale is

    Port (en0, en1, en2, en3 : in std\_logic;

        in0, in1, in2, in3 : in std\_logic\_vector(5 downto 0);

        out0, out1, out2, out3 : out std\_logic\_vector(1 downto 0));

end sistema\_totale;

architecture Behavioral of sistema\_totale is

    component rete\_di\_interconnesione is

    Port (S\_1, S\_2, S\_3, S\_4 : in std\_logic\_vector(1 downto 0);

        D\_1, D\_2, D\_3, D\_4 : out std\_logic\_vector(1 downto 0);

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

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

    end component;

    component gestione\_priorita is

    Port (A\_1, A\_2, A\_3, A\_4 : in std\_logic\_vector(5 downto 0);

    en\_1, en\_2, en\_3, en\_4 : in std\_logic;

    s, d : out std\_logic\_vector(1 downto 0);

    B\_1, B\_2, B\_3, B\_4 : out std\_logic\_vector(1 downto 0));

    end component;

    signal temp1, temp2, temp3, temp4 : std\_logic\_vector(1 downto 0);

    signal source, destination : std\_logic\_vector(1 downto 0);

begin

    g\_p : gestione\_priorita port map (

        A\_1 => in0,

        A\_2 => in1,

        A\_3 => in2,

        A\_4 => in3,

        en\_1 => en0,

        en\_2 => en1,

        en\_3 => en2,

        en\_4 => en3,

        s => source,

        d => destination,

        B\_1 => temp1,

        B\_2 => temp2,

        B\_3 => temp3,

        B\_4 => temp4);

    r\_d\_i : rete\_di\_interconnesione port map (

        S\_1 => temp1,

        S\_2 => temp2,

        S\_3 => temp3,

        S\_4 => temp4,

        sorgente => source,

        destinazione => destination,

        D\_1 => out0,

        D\_2 => out1,

        D\_3 => out2,

        D\_4 => out3

    );

end Behavioral;

Simulazione

**switch\_elem\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity switch\_elem\_tb is

end switch\_elem\_tb;

architecture testbench of switch\_elem\_tb is

    signal X\_1, X\_2, Y\_1, Y\_2: std\_logic\_vector(1 downto 0);

    signal src, dest: std\_logic := '0';

    constant CLOCK\_PERIOD: time := 10 ns;

begin

    uut : entity work.switch\_elem

        port map(

            X\_1 => X\_1,

            X\_2 => X\_2,

            src => src,

            dest => dest,

            Y\_1 => Y\_1,

            Y\_2 => Y\_2

        );

    process

    begin

        while now < 1000 ns loop

            wait for CLOCK\_PERIOD / 2;

            wait for CLOCK\_PERIOD / 2;

        end loop;

        wait;

    end process;

    process

    begin

        X\_1 <= "00";

        X\_2 <= "11";

        src <= '0';

        dest <= '0';

        wait for 20 ns;

        assert Y\_1 = "00" report "Error for src = '0', dest = '0'" severity error;

        assert Y\_2 = "--" report "Error for src = '0', dest = '0'" severity error;

        X\_1 <= "01";

        X\_2 <= "10";

        src <= '1';

        dest <= '1';

        wait for 20 ns;

        assert Y\_1 = "--" report "Error for src = '1', dest = '1'" severity error;

        assert Y\_2 = "10" report "Error for src = '1', dest = '1'" severity error;

        X\_1 <= "10";

        X\_2 <= "01";

        src <= '0';

        dest <= '0';

        wait for 20 ns;

        assert Y\_1 = "10" report "Error for src = '0', dest = '0'" severity error;

        assert Y\_2 = "--" report "Error for src = '0', dest = '0'" severity error;

        X\_1 <= "11";

        X\_2 <= "00";

        src <= '1';

        dest <= '1';

        wait for 20 ns;

        assert Y\_1 = "--" report "Error for src = '1', dest = '1'" severity error;

        assert Y\_2 = "00" report "Error for src = '1', dest = '1'" severity error;

        wait;

    end process;

end testbench;

**Immagine che contiene schermata, linea, software

Descrizione generata automaticamente**

**rete\_di\_interconnesione\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity rete\_di\_interconnesione\_tb is

end rete\_di\_interconnesione\_tb;

architecture testbench of rete\_di\_interconnesione\_tb is

    signal S\_1, S\_2, S\_3, S\_4, D\_1, D\_2, D\_3, D\_4, sorgente, destinazione: std\_logic\_vector(1 downto 0);

    constant CLOCK\_PERIOD: time := 10 ns;

begin

    dut : entity work.rete\_di\_interconnesione

        port map(

            S\_1 => S\_1,

            S\_2 => S\_2,

            S\_3 => S\_3,

            S\_4 => S\_4,

            D\_1 => D\_1,

            D\_2 => D\_2,

            D\_3 => D\_3,

            D\_4 => D\_4,

            sorgente => sorgente,

            destinazione => destinazione

        );

    process

    begin

        while now < 1000 ns loop

            wait for CLOCK\_PERIOD / 2;

            wait for CLOCK\_PERIOD / 2;

        end loop;

        wait;

    end process;

    process

    begin

        S\_1 <= "00";

        S\_2 <= "11";

        S\_3 <= "10";

        S\_4 <= "01";

        sorgente <= "01";

        destinazione <= "10";

        wait for 20 ns;

        assert D\_1 = "10" report "Error for Test 1" severity error;

        assert D\_2 = "11" report "Error for Test 1" severity error;

        assert D\_3 = "--" report "Error for Test 1" severity error;

        assert D\_4 = "--" report "Error for Test 1" severity error;

        sorgente <= "10";

        destinazione <= "01";

        wait for 20 ns;

        assert D\_1 = "--" report "Error for Test 2" severity error;

        assert D\_2 = "--" report "Error for Test 2" severity error;

        assert D\_3 = "00" report "Error for Test 2" severity error;

        assert D\_4 = "11" report "Error for Test 2" severity error;

        sorgente <= "00";

        destinazione <= "11";

        wait for 20 ns;

        assert D\_1 = "--" report "Error for Test 3" severity error;

        assert D\_2 = "--" report "Error for Test 3" severity error;

        assert D\_3 = "10" report "Error for Test 3" severity error;

        assert D\_4 = "01" report "Error for Test 3" severity error;

        sorgente <= "01";

        destinazione <= "01";

        wait for 20 ns;

        assert D\_1 = "01" report "Error for Test 4" severity error;

        assert D\_2 = "--" report "Error for Test 4" severity error;

        assert D\_3 = "--" report "Error for Test 4" severity error;

        assert D\_4 = "--" report "Error for Test 4" severity error;

        sorgente <= "10";

        destinazione <= "10";

        wait for 20 ns;

        assert D\_1 = "--" report "Error for Test 5" severity error;

        assert D\_2 = "10" report "Error for Test 5" severity error;

        assert D\_3 = "--" report "Error for Test 5" severity error;

        assert D\_4 = "--" report "Error for Test 5" severity error;

        sorgente <= "11";

        destinazione <= "00";

        wait for 20 ns;

        assert D\_1 = "--" report "Error for Test 6" severity error;

        assert D\_2 = "--" report "Error for Test 6" severity error;

        assert D\_3 = "00" report "Error for Test 6" severity error;

        assert D\_4 = "--" report "Error for Test 6" severity error;

        wait;

    end process;

end testbench;

**Immagine che contiene schermata, testo, linea

Descrizione generata automaticamente**

**tb\_sistema\_totale.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_sistema\_totale is

end tb\_sistema\_totale;

architecture tb\_architecture of tb\_sistema\_totale is

    signal en0, en1, en2, en3 : std\_logic;

    signal in0, in1, in2, in3 : std\_logic\_vector(5 downto 0);

    signal out0, out1, out2, out3 : std\_logic\_vector(1 downto 0);

    component sistema\_totale

        Port (en0, en1, en2, en3 : in std\_logic;

              in0, in1, in2, in3 : in std\_logic\_vector(5 downto 0);

              out0, out1, out2, out3 : out std\_logic\_vector(1 downto 0));

    end component;

begin

    uut: sistema\_totale port map (

        en0 => en0,

        en1 => en1,

        en2 => en2,

        en3 => en3,

        in0 => in0,

        in1 => in1,

        in2 => in2,

        in3 => in3,

        out0 => out0,

        out1 => out1,

        out2 => out2,

        out3 => out3

    );

    stim\_process: process

    begin

        wait for 10 ns;

        en0 <= '1';

        en1 <= '1';

        en2 <= '1';

        en3 <= '1';

        in0 <="001100"; --11 a 2

        in1 <="011001"; --11 a 3

        in2 <="100110"; --10 a 0

        in3 <="110011"; --00 a 1

        wait for 60 ns;

        en0 <= '0';

        wait for 60ns;

        en1 <= '0';

        wait for 60ns;

        en2 <= '0';

        wait;

    end process stim\_process;

end tb\_architecture;

**Immagine che contiene schermata

Descrizione generata automaticamente**