Capitolo 1: Reti Combinatorie Elementari

Esercizio 1: Multiplexer 16:1

Progetto e architettura

Per implementare il multiplexer indirizzabile 16:1, si è adottato un approccio di progettazione per composizione, sfruttando un multiplexer 4:1 come elemento base.

La realizzazione del multiplexer 4:1 ha coinvolto la definizione comportamentale di un modulo (**mux\_4\_1**) con sei ingressi (**D0**, **D1**, **D2**, **D3**, **S0**, **S1**) e un'uscita (**Y**). La logica di selezione è gestita in un processo comportamentale e la scelta viene fatta in base a due segnali di selezione in input (**S0**, **S1**). Ad esempio, se sia S0 che S1 sono alti, l'uscita Y sarà collegata a D3.

L’entity **mux\_16\_1** presenta, invece, due ingressi (**D**, **S**) e una singola uscita (**Y**). In particolare, D è un vettore di 16 bit che rappresenta le linee di dati in ingresso, mentre S è un vettore di 4 bit che funge da selezionatore. L'architettura del multiplexer 16:1 è implementata in modo strutturale: si impiegano cinque istanze del multiplexer 4:1 come “building block” per la costruzione del multiplexer indirizzabile 16:1.

Nell'immagine sottostante sono illustrati i collegamenti tra le uscite e gli ingressi del multiplexer 16:1 e le uscite e gli ingressi dei multiplexer 4:1. Questa rappresentazione offre una chiara visione dell'organizzazione gerarchica ad albero che si sviluppa da tali connessioni.

Immagine che contiene diagramma, testo, Piano, schematico

Descrizione generata automaticamente

Per implementare una rete di interconnessione a 16 sorgenti e 4 destinazioni (**interconnect\_16\_4**), si è adottato un approccio di progettazione strutturale. Il multiplexer indirizzabile 16:1 precedentemente sviluppato è un componente base che viene combinato in serie con un demultiplexer 1:4 (**demux\_1\_4**), sempre realizzato in modo “Behavioral”.

Il modulo **mux\_4\_to\_1** rappresenta un multiplexer con quattro ingressi (**D0**, **D1**, **D2**, **D3**) e un'uscita (**Y**). I segnali di selezione S0 e S1 controllano quale ingresso viene instradato all'uscita; ad esempio, se S0 e S1 sono entrambi bassi, l’output sarà collegata a D0.

Nel network, il multiplexer 16:1 è incaricato di selezionare l’input da instradare in base a linee di selezione della sorgente (**s\_in**). Contemporaneamente, il demultiplexer 1:4 indirizza il segnale prescelto verso la destinazione specificata attraverso segnali di selezione dedicati (**s\_out**).

Di seguito è riportata l'immagine dell'architettura del sistema di interconnessione completo.

Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Descrizione generata automaticamente

Implementazione

Sono successivamente riportate le descrizioni VHDL del multiplexer 4:1, del multiplexer 16:1, del demultiplexer 4:1 e della rete di interconnessione 16:4.

**mux\_4\_to\_1.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity mux\_4\_to\_1 is

    Port ( D0, D1, D2, D3 : in STD\_LOGIC;

           S0, S1 : in STD\_LOGIC;

           Y : out STD\_LOGIC);

end mux\_4\_to\_1;

architecture Behavioral of mux\_4\_to\_1 is

begin

    process (S0, S1, D0, D1, D2, D3)

    begin

        if (S0 = '0' and S1 = '0') then

            Y <= D0;

        elsif (S0 = '0' and S1 = '1') then

            Y <= D1;

        elsif (S0 = '1' and S1 = '0') then

            Y <= D2;

        elsif (S0 = '1' and S1 = '1') then

            Y <= D3;

        else

            Y <= '-';

        end if;

    end process;

end Behavioral;

**mux\_16\_1.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity mux\_16\_1 is

    Port ( D : in STD\_LOGIC\_VECTOR(15 downto 0);

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

           Y : out STD\_LOGIC);

end mux\_16\_1;

architecture Structural of mux\_16\_1 is

    signal intermediate\_signals : STD\_LOGIC\_VECTOR(3 downto 0);

    signal Y\_temp : STD\_LOGIC;

    component mux\_4\_to\_1

        Port ( D0, D1, D2, D3 : in STD\_LOGIC;

               S0, S1 : in STD\_LOGIC;

               Y : out STD\_LOGIC);

    end component;

begin

    MUX0: mux\_4\_to\_1 port map (

        D0 => D(0), D1 => D(1), D2 => D(2), D3 => D(3),

        S0 => S(1), S1 => S(0),

        Y => intermediate\_signals(0)

    );

    MUX1: mux\_4\_to\_1 port map (

        D0 => D(4), D1 => D(5), D2 => D(6), D3 => D(7),

        S0 => S(1), S1 => S(0),

        Y => intermediate\_signals(1)

    );

    MUX2: mux\_4\_to\_1 port map (

        D0 => D(8), D1 => D(9), D2 => D(10), D3 => D(11),

        S0 => S(1), S1 => S(0),

        Y => intermediate\_signals(2)

    );

    MUX3: mux\_4\_to\_1 port map (

        D0 => D(12), D1 => D(13), D2 => D(14), D3 => D(15),

        S0 => S(1), S1 => S(0),

        Y => intermediate\_signals(3)

    );

    MUX\_final: mux\_4\_to\_1 port map (

        D0 => intermediate\_signals(0),

        D1 => intermediate\_signals(1),

        D2 => intermediate\_signals(2),

        D3 => intermediate\_signals(3),

        S0 => S(3),

        S1 => S(2),

        Y => Y\_temp

    );

    Y <= Y\_temp;

end Structural;

**demux\_1\_4.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity demux\_1\_4 is

    Port ( D : in STD\_LOGIC;

           S : in STD\_LOGIC\_VECTOR(1 downto 0);

           Y0, Y1, Y2, Y3 : out STD\_LOGIC);

end demux\_1\_4;

architecture Behavioral of demux\_1\_4 is

begin

    process (D, S)

    begin

        case S is

            when "00" =>

                Y0 <= D;

                Y1 <= '0';

                Y2 <= '0';

                Y3 <= '0';

            when "01" =>

                Y0 <= '0';

                Y1 <= D;

                Y2 <= '0';

                Y3 <= '0';

            when "10" =>

                Y0 <= '0';

                Y1 <= '0';

                Y2 <= D;

                Y3 <= '0';

            when "11" =>

                Y0 <= '0';

                Y1 <= '0';

                Y2 <= '0';

                Y3 <= D;

            when others =>

                Y0 <= '-';

                Y1 <= '-';

                Y2 <= '-';

                Y3 <= '-';

        end case;

    end process;

end Behavioral;

**interconnect\_16\_4.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity interconnect\_16\_4 is

    Port (

        d\_in  : in  STD\_LOGIC\_VECTOR(15 downto 0);

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

        s\_out : in  STD\_LOGIC\_VECTOR(1 downto 0);

        d\_out : out STD\_LOGIC\_VECTOR(3 downto 0)

    );

end interconnect\_16\_4;

architecture Structural of interconnect\_16\_4 is

    component mux\_16\_1

        port (

            D : in  STD\_LOGIC\_VECTOR(15 downto 0);

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

            Y : out STD\_LOGIC

        );

    end component;

    component demux\_1\_4

        port (

            D  : in  STD\_LOGIC;

            S  : in  STD\_LOGIC\_VECTOR(1 downto 0);

            Y0 : out STD\_LOGIC;

            Y1 : out STD\_LOGIC;

            Y2 : out STD\_LOGIC;

            Y3 : out STD\_LOGIC

        );

    end component;

    signal mux\_out : STD\_LOGIC;

begin

    mux\_16\_inst : mux\_16\_1

        port map (

            D => d\_in,

            S => s\_in,

            Y => mux\_out

        );

    demux\_4\_inst : demux\_1\_4

        port map (

            D  => mux\_out,

            S  => s\_out,

            Y0 => d\_out(0),

            Y1 => d\_out(1),

            Y2 => d\_out(2),

            Y3 => d\_out(3)

        );

end Structural;

Simulazione

Sono stati sviluppati testbench per verificare il corretto funzionamento di ciascun modulo, tuttavia, ci si concentra su due in particolare.

Il testbench per il multiplexer 16:1 (**mux\_16\_1\_tb**) è stato progettato per simulare il multiplexer indirizzabile in diverse condizioni di input (**D\_input**) e selezione(**S\_input**). Lo scopo è verificare la corrispondenza tra l'output risultante (**Y\_output**) e quello atteso dato dalla specifica situazione in esame.

**mux\_16\_1\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity mux\_16\_1\_tb is

end mux\_16\_1\_tb;

architecture testbench of mux\_16\_1\_tb is

    signal D\_input : STD\_LOGIC\_VECTOR(15 downto 0) := "0000000000000000";

    signal S\_input : STD\_LOGIC\_VECTOR(3 downto 0) := "0000";

    signal Y\_output : STD\_LOGIC;

    component mux\_16\_1

        Port ( D : in STD\_LOGIC\_VECTOR(15 downto 0);

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

               Y : out STD\_LOGIC);

    end component;

begin

    MUX16to1 : mux\_16\_1

        port map (D => D\_input, S => S\_input, Y => Y\_output);

    stimulus : process

    begin

        D\_input <= "0000000000000001";

        S\_input <= "0000";

        wait for 10 ns;

        D\_input <= "1000000000000100";

        S\_input <= "1110";

        wait for 10 ns;

        D\_input <= "0000000000001000";

        S\_input <= "0011";

        wait for 10 ns;

        D\_input <= "0000000000010000";

        S\_input <= "0110";

        wait for 10 ns;

        D\_input <= "1000000100000000";

        S\_input <= "1111";

        wait for 10 ns;

        D\_input <= "0000000000000001";

        S\_input <= "0000";

        wait for 10 ns;

        D\_input <= "0000000000000100";

        S\_input <= "0010";

        wait for 10 ns;

        D\_input <= "1000000000000000";

        S\_input <= "1111";

        wait for 10 ns;

        D\_input <= "0100000000100000";

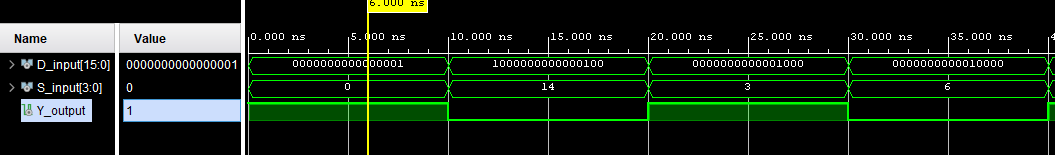
        S\_input <= "0101";

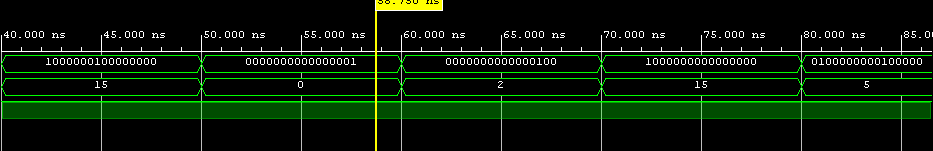
        wait for 10 ns;

        wait;

    end process stimulus;

end testbench;





Il testbench **interconnect\_16\_4\_tb** verifica il funzionamento dell'intera rete di interconnessione a 16 sorgenti e 4 uscite. Applicando sequenze di ingressi a 16 bit (**d\_in\_tb**) e segnali di selezione (**s\_in\_tb**, **s\_out\_tb**), il testbench valuta l'adeguato instradamento del segnale attraverso la rete, analizzando l'uscita ottenuta (**d\_out\_tb**).

**interconnect\_16\_4\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity interconnect\_16\_4\_tb is

end interconnect\_16\_4\_tb;

architecture testbench of interconnect\_16\_4\_tb is

    signal d\_in\_tb   : STD\_LOGIC\_VECTOR(15 downto 0);

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

    signal s\_out\_tb  : STD\_LOGIC\_VECTOR(1 downto 0);

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

    component interconnect\_16\_4

        port (

            d\_in  : in  STD\_LOGIC\_VECTOR(15 downto 0);

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

            s\_out : in  STD\_LOGIC\_VECTOR(1 downto 0);

            d\_out : out STD\_LOGIC\_VECTOR(3 downto 0)

        );

    end component;

begin

    uut : interconnect\_16\_4

        port map (

            d\_in  => d\_in\_tb,

            s\_in  => s\_in\_tb,

            s\_out => s\_out\_tb,

            d\_out => d\_out\_tb

        );

    stimuli : process

    begin

        d\_in\_tb  <= (others => '0');

        s\_in\_tb  <= "0000";

        s\_out\_tb <= "00";

        wait for 10 ns;

        d\_in\_tb  <= "1111111100000000";

        s\_in\_tb  <= "0010";

        s\_out\_tb <= "11";

        wait for 10 ns;

        d\_in\_tb  <= "1111111111111111";

        s\_in\_tb  <= "0001";

        s\_out\_tb <= "01";

        wait for 10 ns;

        d\_in\_tb  <= "0000000011111111";

        s\_in\_tb  <= "0011";

        s\_out\_tb <= "10";

        wait for 10 ns;

        wait;

    end process;

end testbench;

Immagine che contiene schermata, linea

Descrizione generata automaticamente

Sintesi su board di sviluppo

La sintesi sul board Nexys A7-100T richiede la scrittura di un modulo dedicato. L'entity **interconnect\_16\_4\_board** presenta diverse porte di input e output necessarie per interfacciarsi con il board di sviluppo:

* **value14\_in** è un vettore di 14 segnali collegati ai primi 14 switch, da **U12** a **J15**, i quali vengono utilizzati per inserire i dati in ingresso e la sorgente e la destinazione.
* **load\_sel** è il segnale collegato al pulsante **P18**; quando esso viene premuto si carica l’informazione della destinazione dagli switch **U12** e **H6** e quella della sorgente dagli switch **T13** a **T8**.
* **load\_first\_part** è il segnale collegato al bottone **P17**; quando esso viene premuto si carica la prima metà dei dati in ingresso, quella meno significativa, a partire dagli switch **R13** a **J15**.
* **load\_second\_part** è il segnale collegato al puntante **M17** il quale quando viene premuto carica la seconda metà dei dati, quelli più significativi, sempre dagli switch R13 a J15.
* **reset** è collegato al button **N17** e quando questo viene premuto tutte i segnali interni a interconnect\_16\_4\_board che contengono informazioni precedentemente inserite sono azzerati.
* **led** è un vettore di output a 4 bit collegato ai led da **N14** a **H17**, utilizzati per visualizzare l'output del sistema.

**interconnect\_16\_4\_board.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity interconnect\_16\_4\_board is

    Port (

        load\_first\_part : in  STD\_LOGIC;

        load\_second\_part : in  STD\_LOGIC;

        load\_sel : in STD\_LOGIC;

        reset : in STD\_LOGIC;

        value14\_in : in STD\_LOGIC\_VECTOR(13 downto 0);

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

    );

end interconnect\_16\_4\_board;

architecture Structural of interconnect\_16\_4\_board is

    component interconnect\_16\_4

        port (

            d\_in  : in  STD\_LOGIC\_VECTOR(15 downto 0);

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

            s\_out : in  STD\_LOGIC\_VECTOR(1 downto 0);

            d\_out : out STD\_LOGIC\_VECTOR(3 downto 0)

        );

    end component;

    signal inter\_out : STD\_LOGIC\_VECTOR(3 downto 0) := (others => '0');

    signal inter\_in : STD\_LOGIC\_VECTOR(15 downto 0) := (others => '0');

    signal sel\_in    : STD\_LOGIC\_VECTOR(3 downto 0) := (others => '0');

    signal sel\_out   : STD\_LOGIC\_VECTOR(1 downto 0) := (others => '0');

begin

    intconn\_16\_4 : interconnect\_16\_4

        port map (

            d\_in  => inter\_in,

            s\_in  => sel\_in,

            s\_out => sel\_out,

            d\_out => inter\_out

        );

    main : process (reset, load\_sel, load\_first\_part, load\_second\_part)

    begin

        if reset = '1' then

            inter\_in <= (others => '0');

        elsif load\_sel = '1' then

            sel\_in <= value14\_in(11 downto 8);

            sel\_out <= value14\_in(13 downto 12);

        elsif load\_first\_part = '1' then

            inter\_in(7 downto 0) <= value14\_in(7 downto 0);

        elsif load\_second\_part = '1' then

            inter\_in(15 downto 8) <= value14\_in(7 downto 0);

        end if;

    END PROCESS;

    led <= inter\_out;

end Structural;

Dopo sono riportate le linee che è stato necessario decommentare e opportunamente modificare nel file dei vincoli “Nexys-A7-100T-Master.xdc” per poter implementare quanto appena descritto.

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[4] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[5] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[6] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[7] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { value14\_in[8] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { value14\_in[9] }]; #IO\_25\_34 Sch=sw[9]

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[10] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[11] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[12] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { value14\_in[13] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

## 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 N17 IOSTANDARD LVCMOS33 } [get\_ports { reset }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

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

set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { load\_second\_part }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { load\_sel }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd