Lab2b.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;




entity lab2b is  --lab2b entity is the 16-Node Toroid
        generic (W: integer := 8);
        port(   Din0, Din1, Din2, Din3, Din4: in STD_LOGIC_VECTOR(W-1 downto
0);
        Din5, Din6, Din7, Din8, Din9: in STD_LOGIC_VECTOR(W-1 downto 0);
        Din10, Din11, Din12, Din13, Din14, Din15: in STD_LOGIC_VECTOR(W-1
downto 0);
        address_2b: in STD_LOGIC_VECTOR(1 downto 0);
        clock, clear:           in  std_logic;
        Dout0, Dout1, Dout2, Dout3, Dout4: out STD_LOGIC_VECTOR(W-1 downto
0);
        Dout5, Dout6, Dout7, Dout8, Dout9: out STD_LOGIC_VECTOR(W-1 downto
0);
        Dout10, Dout11, Dout12, Dout13, Dout14, Dout15: out
STD_LOGIC_VECTOR(W-1 downto 0)
        );
end entity lab2b;



architecture toroid of lab2b is

component routerNode is
        generic (W: integer := 8);
        port(   in_N, in_E, in_S, in_W, in_D: in STD_LOGIC_VECTOR(W-1 downto
0);
        adrPort: in STD_LOGIC_VECTOR(1 downto 0);
        clock, clear:           in  std_logic;
        out_N, out_E, out_S, out_W, out_D : out STD_LOGIC_VECTOR(W-1 downto
0)
        );
end component routerNode;

component routerNode_at_first_col is
        generic (W: integer := 8);
        port(   in_N, in_E, in_S, in_W, in_D: in STD_LOGIC_VECTOR(W-1 downto
0);
        adrPort: in STD_LOGIC_VECTOR(1 downto 0);
        clock, clear:           in  std_logic;
        out_N, out_E, out_S, out_W, out_D : out STD_LOGIC_VECTOR(W-1 downto
0)
```

```
        );
end component routerNode_at_first_col;


signal nodein0_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein1_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein2_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein3_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein4_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein5_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein6_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein7_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein8_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein9_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein10_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein11_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein12_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein13_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein14_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein15_N: STD_LOGIC_VECTOR( 7 downto 0);

signal nodein0_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein1_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein2_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein3_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein4_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein5_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein6_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein7_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein8_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein9_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein10_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein11_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein12_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein13_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein14_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein15_E: STD_LOGIC_VECTOR( 7 downto 0);

signal nodein0_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein1_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein2_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein3_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein4_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein5_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein6_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein7_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein8_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein9_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein10_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein11_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein12_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein13_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein14_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein15_S: STD_LOGIC_VECTOR( 7 downto 0);

signal nodein0_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein1_W: STD_LOGIC_VECTOR( 7 downto 0);
```

```
signal nodein2_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein3_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein4_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein5_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein6_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein7_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein8_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein9_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein10_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein11_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein12_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein13_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein14_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodein15_W: STD_LOGIC_VECTOR( 7 downto 0);

signal nodeout0_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout1_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout2_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout3_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout4_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout5_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout6_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout7_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout8_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout9_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout10_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout11_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout12_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout13_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout14_N: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout15_N: STD_LOGIC_VECTOR( 7 downto 0);

signal nodeout0_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout1_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout2_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout3_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout4_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout5_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout6_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout7_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout8_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout9_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout10_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout11_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout12_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout13_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout14_E: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout15_E: STD_LOGIC_VECTOR( 7 downto 0);

signal nodeout0_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout1_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout2_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout3_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout4_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout5_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout6_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout7_S: STD_LOGIC_VECTOR( 7 downto 0);
```

```
signal nodeout8_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout9_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout10_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout11_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout12_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout13_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout14_S: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout15_S: STD_LOGIC_VECTOR( 7 downto 0);

signal nodeout0_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout1_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout2_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout3_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout4_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout5_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout6_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout7_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout8_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout9_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout10_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout11_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout12_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout13_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout14_W: STD_LOGIC_VECTOR( 7 downto 0);
signal nodeout15_W: STD_LOGIC_VECTOR( 7 downto 0);

begin
    node0: routerNode_at_first_col port map (in_N => nodeout12_S, in_E =>
nodeout1_W, in_S => nodeout4_N, in_W => nodeout3_E,
                                             out_N => nodeout0_N, out_E  =>
nodeout0_E, out_S  => nodeout0_S , out_W  => nodeout0_W,
                                             in_D => Din0, out_D => Dout0,
clock => clock, adrPort => address_2b, clear => clear
                                               );


    node1: routerNode port map (in_N => nodeout13_S, in_E => nodeout2_W, in_S
=> nodeout5_N, in_W => nodeout0_E,
                                            out_N => nodeout1_N, out_E  =>
nodeout1_E, out_S  => nodeout1_S , out_W  => nodeout1_W,
                                            in_D => Din1, out_D => Dout1,
clock => clock, adrPort => address_2b, clear => clear
                                            );

    node2: routerNode port map (in_N => nodeout14_S, in_E => nodeout3_W, in_S
=> nodeout6_N, in_W => nodeout1_E,
                                            out_N => nodeout2_N, out_E  =>
nodeout2_E, out_S  => nodeout2_S , out_W  => nodeout2_W,
                                            in_D => Din2, out_D => Dout2,
clock => clock, adrPort => address_2b, clear => clear
                                            );

    node3: routerNode port map (in_N => nodeout15_S, in_E => nodeout0_W, in_S
=> nodeout7_N, in_W => nodeout2_E,
                                            out_N => nodeout3_N, out_E  =>
nodeout3_E, out_S  => nodeout3_S , out_W  => nodeout3_W,
```

```
                                                in_D => Din3, out_D => Dout3,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node4: routerNode_at_first_col port map (in_N => nodeout0_S, in_E =>
nodeout5_W, in_S => nodeout8_N, in_W => nodeout7_E,
                                                out_N => nodeout4_N, out_E  =>
nodeout4_E, out_S  => nodeout4_S , out_W  => nodeout4_W,
                                                in_D => Din4, out_D => Dout4,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node5: routerNode port map (in_N => nodeout1_S, in_E => nodeout6_W, in_S
=> nodeout9_N, in_W => nodeout4_E,
                                                out_N => nodeout5_N, out_E  =>
nodeout5_E, out_S  => nodeout5_S , out_W  => nodeout5_W,
                                                in_D => Din5, out_D => Dout5,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node6: routerNode port map (in_N => nodeout2_S, in_E => nodeout7_W, in_S
=> nodeout10_N, in_W => nodeout5_E,
                                                out_N => nodeout6_N, out_E  =>
nodeout6_E, out_S  => nodeout6_S , out_W  => nodeout6_W,
                                                in_D => Din6, out_D => Dout6,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node7: routerNode port map (in_N => nodeout3_S, in_E => nodeout4_W, in_S
=> nodeout11_N, in_W => nodeout6_E,
                                                out_N => nodeout7_N, out_E  =>
nodeout7_E, out_S  => nodeout7_S , out_W  => nodeout7_W,
                                                in_D => Din7, out_D => Dout7,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node8: routerNode_at_first_col port map (in_N => nodeout4_S, in_E =>
nodeout9_W, in_S => nodeout12_N, in_W => nodeout11_E,
                                                out_N => nodeout8_N, out_E  =>
nodeout8_E, out_S  => nodeout8_S , out_W  => nodeout8_W,
                                                in_D => Din8, out_D => Dout8,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node9: routerNode port map (in_N => nodeout5_S, in_E => nodeout10_W, in_S
=> nodeout13_N, in_W => nodeout8_E,
                                                out_N => nodeout9_N, out_E  =>
nodeout9_E, out_S  => nodeout9_S , out_W  => nodeout9_W,
                                                in_D => Din9, out_D => Dout9,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node10: routerNode port map (in_N => nodeout6_S, in_E => nodeout11_W,
in_S => nodeout14_N, in_W => nodeout9_E,
                                                out_N => nodeout10_N, out_E  =>
nodeout10_E, out_S  => nodeout10_S , out_W  => nodeout10_W,
```

```
                                                in_D => Din10, out_D => Dout10,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node11: routerNode port map (in_N => nodeout7_S, in_E => nodeout8_W, in_S
=> nodeout15_N, in_W => nodeout10_E,
                                                out_N => nodeout11_N, out_E  =>
nodeout11_E, out_S  => nodeout11_S , out_W  => nodeout11_W,
                                                in_D => Din11, out_D => Dout11,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node12: routerNode_at_first_col port map (in_N => nodeout8_S, in_E =>
nodeout13_W, in_S => nodeout0_N, in_W => nodeout15_E,
                                                out_N => nodeout12_N, out_E  =>
nodeout12_E, out_S  => nodeout12_S , out_W  => nodeout12_W,
                                                in_D => Din12, out_D => Dout12,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node13: routerNode port map (in_N => nodeout9_S, in_E => nodeout14_W,
in_S => nodeout1_N, in_W => nodeout12_E,
                                                out_N => nodeout13_N, out_E  =>
nodeout13_E, out_S  => nodeout13_S , out_W  => nodeout13_W,
                                                in_D => Din13, out_D => Dout13,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node14: routerNode port map (in_N => nodeout10_S, in_E => nodeout15_W,
in_S => nodeout2_N, in_W => nodeout13_E,
                                                out_N => nodeout14_N, out_E  =>
nodeout14_E, out_S  => nodeout14_S , out_W  => nodeout14_W,
                                                in_D => Din14, out_D => Dout14,
clock => clock, adrPort => address_2b, clear => clear
                                                );

    node15: routerNode port map (in_N => nodeout11_S, in_E => nodeout12_W,
in_S => nodeout3_N, in_W => nodeout14_E,
                                                out_N => nodeout15_N, out_E  =>
nodeout15_E, out_S  => nodeout15_S , out_W  => nodeout15_W,
                                                in_D => Din15, out_D => Dout15,
clock => clock, adrPort => address_2b, clear => clear
                                                );



end architecture toroid;
```

LUT.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity LUT is
        port( --address: in STD_LOGIC_VECTOR(14 downto 0);
                --sel_LUT
                address : in STD_LOGIC_VECTOR (1 downto 0); -- 4 cases could
be test
                sel_LUT0,sel_LUT1,sel_LUT2,sel_LUT3,sel_LUT4: out
STD_LOGIC_VECTOR(2 downto 0)

            );
end entity LUT;



architecture selMuxBits of LUT is
begin

-- if (address = 00)
        -- we have default case
-- if address = 01
--      we have case for first col
-- if address = 10
--      we have the other case
-- if others , we dont care



            MUX0_LUT: process (address) --North out
            begin
                case address is
                    when "00" => sel_LUT0  <= "111";
                    --when "01" => sel_LUT0  <= "111";
                    when "01" => sel_LUT0  <= "111";
                    when others => sel_LUT0 <= "111";
                end case;
            end process;
            MUX1_LUT: process (address) --East out
            begin
                case address is
                    when "00" => sel_LUT1  <= "111";
                    --when "01" => sel_LUT1  <= "100";
                    when "01" => sel_LUT1  <= "100";
                    when others => sel_LUT1 <= "111";
                end case;
            end process;
            MUX2_LUT: process (address)  --South out
            begin
                case address is
                    when "00" => sel_LUT2  <= "111";
                    --when "01" => sel_LUT2  <= "011";
                    when "01" => sel_LUT2  <= "111";
                    when others => sel_LUT2  <= "111";
                end case;
```

```
            end process;
            MUX3_LUT: process (address)   --West out
            begin
                case address is
                    when "00" => sel_LUT3  <= "111";
                    --when "01" => sel_LUT3  <= "111";
                    when "01" => sel_LUT3  <= "111";
                    when others => sel_LUT3  <= "111";
                end case;
            end process;
            MUX4_LUT: process (address)   --Dout
            begin
                case address is
                    when "00" => sel_LUT4  <= "111";
                    --when "01" => sel_LUT4  <= "000";
                    when "01" => sel_LUT4  <= "011";
                    when others => sel_LUT4  <= "111";
                end case;
            end process;


end architecture selMuxBits;
```

LUT_for_first_col.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity LUT_for_first_col is
        port( --address: in STD_LOGIC_VECTOR(14 downto 0);
                --sel_LUT
                address : in STD_LOGIC_VECTOR (1 downto 0); -- 4 cases could
be test
                sel_LUT0,sel_LUT1,sel_LUT2,sel_LUT3,sel_LUT4: out
STD_LOGIC_VECTOR(2 downto 0)

            );
end entity LUT_for_first_col;



architecture selMuxBits of LUT_for_first_col is
begin

-- if (address = 00)
        -- we have default case
-- if address = 01
--      we have case for first col
-- if address = 10
--      we have the other case
-- if others , we dont care
```

```
    MUX0_LUT: process (address) --North out
    begin
        case address is
            when "00" => sel_LUT0  <= "111";
            when "01" => sel_LUT0  <= "111";
            --when "10" => sel_LUT0  <= "111";
            when others => sel_LUT0 <= "111";
        end case;
    end process;
    MUX1_LUT: process (address) --East out
    begin
        case address is
            when "00" => sel_LUT1  <= "111";
            when "01" => sel_LUT1  <= "100";
            --when "10" => sel_LUT1  <= "100";
            when others => sel_LUT1 <= "111";
        end case;
    end process;
    MUX2_LUT: process (address)  --South out
    begin
        case address is
            when "00" => sel_LUT2  <= "111";
            when "01" => sel_LUT2  <= "011";
            --when "10" => sel_LUT2  <= "111";
            when others => sel_LUT2  <= "111";
        end case;
    end process;
    MUX3_LUT: process (address)   --West out
    begin
        case address is
            when "00" => sel_LUT3  <= "111";
            when "01" => sel_LUT3  <= "111";
            --when "10" => sel_LUT3  <= "111";
            when others => sel_LUT3  <= "111";
        end case;
    end process;
    MUX4_LUT: process (address)   --Dout
    begin
        case address is
            when "00" => sel_LUT4  <= "111";
            when "01" => sel_LUT4  <= "000";
            --when "10" => sel_LUT4  <= "011";
            when others => sel_LUT4  <= "111";
        end case;
    end process;


end architecture selMuxBits;
```

routerNode.vhd

```
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;



entity routerNode is
        generic (W: integer := 8);
        port(   in_N, in_E, in_S, in_W, in_D: in STD_LOGIC_VECTOR(W-1 downto
0);
        adrPort: in STD_LOGIC_VECTOR(1 downto 0);
        clock, clear:           in  std_logic;
        out_N, out_E, out_S, out_W, out_D : out STD_LOGIC_VECTOR(W-1 downto
0)
        );
end entity routerNode;

architecture routerNode1 of routerNode is

signal in_Nr, in_Er, in_Sr, in_Wr, in_Dr :  STD_LOGIC_VECTOR(W-1 downto 0);
signal out0, out1, out2, out3, out4:   STD_LOGIC_VECTOR(W-1 downto 0);

component LUT is --include object LUT
        port( address: in STD_LOGIC_VECTOR(1 downto 0);
                sel_LUT0,sel_LUT1,sel_LUT2,sel_LUT3,sel_LUT4: out
STD_LOGIC_VECTOR(2 downto 0)
        );
end component;


signal sel: STD_LOGIC_VECTOR(14 downto 0);


begin
    --map the output from LUT to signal in switch
    assign_LUT_switch: LUT port map(sel_LUT0 => sel(2 downto 0), sel_LUT1 =>
sel(5 downto 3),
    sel_LUT2 => sel(8 downto 6), sel_LUT3 => sel(11 downto 9),sel_LUT4 =>
sel(14 downto 12), address => adrPort);


    Latch0: process (in_N, clock,clear) begin
        if (clear = '1') then
            for i in 0 to W-1 loop
                in_Nr(i) <= '0';
                -- in_Nr is the 'registered' or 'latched' version of in_N
            end loop;
        elsif (clock'EVENT and clock = '1') then
            in_Nr <= in_N;
        end if;
    end process;

    LATCH1:  process (in_E, clock, clear) begin
        if clear = '1' then
            for i in 0 to W-1 loop
                in_Er(i) <= '0';
            end loop;
```

```
    elsif clock'EVENT and clock = '1' then
        in_Er <= in_E;
    end if;
end process;

LATCH2:  process (in_S, clock, clear) begin
    if clear = '1' then
        for i in 0 to W-1 loop
            in_Sr(i) <= '0';
        end loop;
    elsif clock'EVENT and clock = '1' then
        in_Sr <= in_S;
    end if;
end process;

LATCH3:  process (in_W, clock, clear) begin
    if clear = '1' then
        for i in 0 to W-1 loop
            in_Wr(i) <= '0';
        end loop;
    elsif clock'EVENT and clock = '1' then
        in_Wr <= in_W;
    end if;
end process;

LATCH4:  process (in_D, clock, clear) begin
    if clear = '1' then
        for i in 0 to W-1 loop
            in_Dr(i) <= '0';
        end loop;
    elsif clock'EVENT and clock = '1' then
        in_Dr <= in_D;
    end if;
end process;
-------------------------------------------------------
my_MUX0: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr,sel(2 downto 0)) --sel0
begin
    case sel(2 downto 0) is
        when "000" => out0 <= in_Nr;
        when "001" => out0 <= in_Er;
        when "010" => out0 <= in_Sr;
        when "011" => out0 <= in_Wr;
        when "100" => out0 <= in_Dr;
        when others => null;
    end case;
end process;

my_MUX1: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr,sel(5 downto 3)) --sel1
begin
    case sel(5 downto 3) is
        when "000" => out1 <= in_Nr;
        when "001" => out1 <= in_Er;
        when "010" => out1 <= in_Sr;
        when "011" => out1 <= in_Wr;
        when "100" => out1 <= in_Dr;
        when others => null;
    end case;
```

```
    end process;

    my_MUX2: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr,sel(8 downto 6)) --sel2
    begin
        case sel(8 downto 6) is
            when "000" => out2 <= in_Nr;
            when "001" => out2 <= in_Er;
            when "010" => out2 <= in_Sr;
            when "011" => out2 <= in_Wr;
            when "100" => out2 <= in_Dr;
            when others => null;
        end case;
    end process;

    my_MUX3: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr,sel(11 downto 9)) --
sel3
    begin
        case sel(11 downto 9) is

            when "000" => out3 <= in_Nr;
            when "001" => out3 <= in_Er;
            when "010" => out3 <= in_Sr;
            when "011" => out3 <= in_Wr;
            when "100" => out3 <= in_Dr;
            when others => null;
        end case;
    end process;

    my_MUX4: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr, sel(14 downto 12)) --
sel4
    begin
        case sel(14 downto 12) is
            when "000" => out4 <= in_Nr;
            when "001" => out4 <= in_Er;
            when "010" => out4 <= in_Sr;
            when "011" => out4 <= in_Wr;
            when "100" => out4 <= in_Dr;
            when others => null;
        end case;
    end process;
    -------------------------------------------------------
    stage2_Latch0: process (out0, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_N(i) <= '0';
                -- out_N is the 'registered' or 'latched' version of out0
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_N <= out0;
        end if;
    end process;

    stage2_Latch1: process (out1, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_E(i) <= '0';
                -- out_N is the 'registered' or 'latched' version of out0
```

```
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_E <= out1;
        end if;
    end process;

    stage2_Latch2: process (out2, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_S(i) <= '0';
                -- out_N is the 'registered' or 'latched' version of out0
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_S <= out2;
        end if;
     end process;

    stage2_Latch3: process (out3, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_W(i) <= '0';
                -- out_N is the 'registered' or 'latched' version of out0
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_W <= out3;
        end if;
     end process;

     stage2_Latch4: process (out4, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_D(i) <= '0';
                -- out_N is the 'registered' or 'latched' version of out0
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_D <= out4;
        end if;
     end process;

end architecture routerNode1;
```

routerNode_at_first_col.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;



entity routerNode_at_first_col is
        generic (W: integer := 8);
        port(   in_N, in_E, in_S, in_W, in_D: in STD_LOGIC_VECTOR(W-1 downto
0);
```

```
        adrPort: in STD_LOGIC_VECTOR(1 downto 0);
        clock, clear:            in  std_logic;
        out_N, out_E, out_S, out_W, out_D : out STD_LOGIC_VECTOR(W-1 downto
0)
        );
end entity routerNode_at_first_col;

architecture rt of routerNode_at_first_col is

signal in_Nr, in_Er, in_Sr, in_Wr, in_Dr :  STD_LOGIC_VECTOR(W-1 downto 0);
signal out0, out1, out2, out3, out4:   STD_LOGIC_VECTOR(W-1 downto 0);

component LUT_for_first_col is --include object LUT
        port( address: in STD_LOGIC_VECTOR(1 downto 0);
                sel_LUT0,sel_LUT1,sel_LUT2,sel_LUT3,sel_LUT4: out
STD_LOGIC_VECTOR(2 downto 0)
        );
end component;


signal sel: STD_LOGIC_VECTOR(14 downto 0);


begin
    --map the output from LUT to signal in switch
    assign_LUT_switch: LUT_for_first_col port map(sel_LUT0 => sel(2 downto
0), sel_LUT1 => sel(5 downto 3),
    sel_LUT2 => sel(8 downto 6), sel_LUT3 => sel(11 downto 9),sel_LUT4 =>
sel(14 downto 12), address => adrPort);


    Latch0: process (in_N, clock,clear) begin
        if (clear = '1') then
            for i in 0 to W-1 loop
                in_Nr(i) <= '0';
                -- in_Nr is the 'registered' or 'latched' version of in_N
            end loop;
        elsif (clock'EVENT and clock = '1') then
            in_Nr <= in_N;
        end if;
    end process;

    LATCH1:  process (in_E, clock, clear) begin
        if clear = '1' then
            for i in 0 to W-1 loop
                in_Er(i) <= '0';
            end loop;
        elsif clock'EVENT and clock = '1' then
            in_Er <= in_E;
        end if;
    end process;

    LATCH2:  process (in_S, clock, clear) begin
        if clear = '1' then
            for i in 0 to W-1 loop
                in_Sr(i) <= '0';
            end loop;
```

```
    elsif clock'EVENT and clock = '1' then
        in_Sr <= in_S;
    end if;
end process;

LATCH3:  process (in_W, clock, clear) begin
    if clear = '1' then
        for i in 0 to W-1 loop
            in_Wr(i) <= '0';
        end loop;
    elsif clock'EVENT and clock = '1' then
        in_Wr <= in_W;
    end if;
end process;

LATCH4:  process (in_D, clock, clear) begin
    if clear = '1' then
        for i in 0 to W-1 loop
            in_Dr(i) <= '0';
        end loop;
    elsif clock'EVENT and clock = '1' then
        in_Dr <= in_D;
    end if;
end process;
-------------------------------------------------------------
my_MUX0: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr,sel(2 downto 0)) --sel0
begin
    case sel(2 downto 0) is
        when "000" => out0 <= in_Nr;
        when "001" => out0 <= in_Er;
        when "010" => out0 <= in_Sr;
        when "011" => out0 <= in_Wr;
        when "100" => out0 <= in_Dr;
        when others => null;
    end case;
end process;

my_MUX1: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr,sel(5 downto 3)) --sel1
begin
    case sel(5 downto 3) is
        when "000" => out1 <= in_Nr;
        when "001" => out1 <= in_Er;
        when "010" => out1 <= in_Sr;
        when "011" => out1 <= in_Wr;
        when "100" => out1 <= in_Dr;
        when others => null;
    end case;
end process;

my_MUX2: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr,sel(8 downto 6)) --sel2
begin
    case sel(8 downto 6) is
        when "000" => out2 <= in_Nr;
        when "001" => out2 <= in_Er;
        when "010" => out2 <= in_Sr;
        when "011" => out2 <= in_Wr;
        when "100" => out2 <= in_Dr;
```

```
            when others => null;
        end case;
    end process;

    my_MUX3: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr,sel(11 downto 9)) --
sel3
    begin
        case sel(11 downto 9) is

            when "000" => out3 <= in_Nr;
            when "001" => out3 <= in_Er;
            when "010" => out3 <= in_Sr;
            when "011" => out3 <= in_Wr;
            when "100" => out3 <= in_Dr;
            when others => null;
        end case;
    end process;

    my_MUX4: process(in_Nr, in_Er, in_Sr, in_Wr,in_Dr, sel(14 downto 12)) --
sel4
    begin
        case sel(14 downto 12) is
            when "000" => out4 <= in_Nr;
            when "001" => out4 <= in_Er;
            when "010" => out4 <= in_Sr;
            when "011" => out4 <= in_Wr;
            when "100" => out4 <= in_Dr;
            when others => null;
        end case;
    end process;
    ------------------------------------------------------
    stage2_Latch0: process (out0, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_N(i) <= '0';
                -- out_N is the 'registered' or 'latched' version of out0
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_N <= out0;
        end if;
    end process;

    stage2_Latch1: process (out1, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_E(i) <= '0';
                -- out_N is the 'registered' or 'latched' version of out0
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_E <= out1;
        end if;
    end process;

    stage2_Latch2: process (out2, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_S(i) <= '0';
```

```
                -- out_N is the 'registered' or 'latched' version of out0
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_S <= out2;
        end if;
     end process;

    stage2_Latch3: process (out3, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_W(i) <= '0';
                -- out_N is the 'registered' or 'latched' version of out0
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_W <= out3;
        end if;
     end process;

     stage2_Latch4: process (out4, clock,clear) begin
     if (clear = '1') then
            for i in 0 to W-1 loop
                out_D(i) <= '0';
                -- out_N is the 'registered' or 'latched' version of out0
            end loop;
        elsif (clock'EVENT and clock = '1') then
            out_D <= out4;
        end if;
     end process;

end architecture rt;
```

test_bench_2b.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std;

entity test_bench_2b is

end entity;

architecture test_behaviour of test_bench_2b is

Shared variable endsim : boolean := FALSE;
Shared variable W : integer := 8;

--components
component lab2b is  --lab2b entity is the 16-Node Toroid
        generic (W: integer := 8);
        port(   Din0, Din1, Din2, Din3, Din4: in STD_LOGIC_VECTOR(W-1 downto
0);
        Din5, Din6, Din7, Din8, Din9: in STD_LOGIC_VECTOR(W-1 downto 0);
```

```
        Din10, Din11, Din12, Din13, Din14, Din15: in STD_LOGIC_VECTOR(W-1
downto 0);
        address_2b: in STD_LOGIC_VECTOR(1 downto 0);
        clock, clear:            in  std_logic;
        Dout0, Dout1, Dout2, Dout3, Dout4: out STD_LOGIC_VECTOR(W-1 downto
0);
        Dout5, Dout6, Dout7, Dout8, Dout9: out STD_LOGIC_VECTOR(W-1 downto
0);
        Dout10, Dout11, Dout12, Dout13, Dout14, Dout15: out
STD_LOGIC_VECTOR(W-1 downto 0)
        );
end component lab2b;

------------------------
--define signals below



--signal in_N_test, in_E_test, in_S_test, in_W_test, in_D_test:
STD_LOGIC_VECTOR(W-1 downto 0);  --unsigned
--signal out_N_test, out_E_test, out_S_test, out_W_test, out_D_test:
STD_LOGIC_VECTOR(W-1 downto 0);
signal Din0test, Din1test, Din2test, Din3test, Din4test, Din5test, Din6test,
        Din7test, Din8test, Din9test, Din10test, Din11test, Din12test,
Din13test, Din14test, Din15test: STD_LOGIC_VECTOR(W-1 downto 0);

signal Dout0test, Dout1test, Dout2test, Dout3test, Dout4test, Dout5test,
Dout6test, Dout7test, Dout8test, Dout9test,
        Dout10test, Dout11test, Dout12test, Dout13test, Dout14test,
Dout15test: STD_LOGIC_VECTOR(W-1 downto 0);



signal clock_test : STD_LOGIC;
signal clear_test: STD_LOGIC;
signal address_test: STD_LOGIC_VECTOR(1 downto 0);


begin


 -- Instantiate the Unit Under Test (UUT)
   uut: lab2b generic map(w) PORT MAP (            --double check which
signals are to be kept
            clock=>clock_test, clear => clear_test,
--          in_N=>in_N_test, in_E=>in_E_test, in_S=>in_S_test,
in_W=>in_W_test, in_D=>in_D_test,
--          out_N=>out_N_test, out_E=>out_E_test, out_S=>out_S_test,
out_W=>out_W_test, out_D=>out_D_test,
            address_2b => address_test, Din0 => Din0test, Din1 => Din1test,
Din2 => Din2test, Din3 => Din3test, Din4 => Din4test,
            Din5 => Din5test, Din6 => Din6test, Din7 => Din7test, Din8 =>
Din8test, Din9 => Din9test, Din10 => Din10test, Din11 => Din11test,
            Din12 => Din12test, Din13 => Din13test, Din14 => Din14test, Din15
=> Din15test,
```

```
            Dout0 => Dout0test, Dout1 => Dout1test, Dout2 => Dout2test, Dout3
=> Dout3test, Dout4 => Dout4test,
            Dout5 => Dout5test, Dout6 => Dout6test, Dout7 => Dout7test, Dout8
=> Dout8test, Dout9 => Dout9test, Dout10 => Dout10test, Dout11 => Dout11test,
            Dout12 => Dout12test, Dout13 => Dout13test, Dout14 => Dout14test,
Dout15 => Dout15test
        );


clock_process :process  --50% duty cycle
begin
        if (endsim = FALSE) then
          clock_test <= '0';
        wait for 5ns;  --for 0.5 ns signal is '0'.
        clock_test <= '1';
        wait for 5ns;  --for next 0.5 ns signal is '1'.
          else
                wait;
            end if;
end process clock_process;

--Test Design
test_proc: process(clock_test)
variable clk_counter : integer := 0;
begin

        if(rising_edge(clock_test)) then
            Din0test <= "00000000";
            Din1test <= "00000001";
            Din2test<=  "00000010";
            Din3test<=  "00000011";
            Din4test<=  "00000100";
            Din5test<=  "00000101";
            Din6test<=  "00000110";
            Din7test<=  "00000111";
            Din8test<=  "00001000";
            Din9test<=  "00001001";
            Din10test<= "00001010";
            Din11test<= "00001011";
            Din12test<= "00001100";
            Din13test<= "00001101";
            Din14test<= "00001110";
            Din15test<= "00001111";

            clk_counter := clk_counter + 1;

            if (clk_counter = 1) then
                address_test <= "00";
            elsif (clk_counter = 5) then
                address_test <= "01";
            end if;

        end if;

  end process test_proc;
```

```
  KillClock: process (clock_test)
  variable counter: integer:=0;
  Begin
        if(RISING_EDGE(clock_test)) then
            If(counter <30) then            --after 30 CC kill simulation
                counter := counter + 1;
            else
                endsim := true;
                counter := 0;
            end if;
        else

        end if;
  end process;


end architecture test_behaviour;
```