Lab 3: Standard Combinational Components

Purpose

In this lab you will implement several combinational circuits on the DE1 development board to test and verify their operations.

- Using a high-level description language to describe the operation of a circuit.
- Converting from a binary number to a decimal number.

Introduction

There are several standard combinational components that are used very often in building larger digital systems. These components are used as building blocks for larger systems. They are:

- Adder
- Multiplexer
- ALU
- Decoder
- Encoder
- Comparator

In this lab you will build some of them and verify their operations. These components will be used again in future labs, so save them in a library. For each component, create a symbol for it. To test it, open a new schematic drawing, add the symbol, and map the I/O pins to the symbol.

1. Multiplexer

a) Draw the complete circuit for a 4-bit wide 2-to-1 multiplexer circuit using only AND, OR, and NOT gates. This mux has two data inputs and one data output. The width of the two data input bus and data output bus is 4-bits wide. The select line *s* is only 1-bit wide. Refer to Section 4.2 of the textbook for a discussion of multiplexers.

b) Implement and verify the operation of your 4-bit 2-to-1 multiplexer.

c) Implement a 4-bit 4-to-1 multiplexer.

2. Arithmetic and Logic Unit (ALU)

Instead of manually designing a digital logic circuit, you can describe its operation using a high-level language and then have Quartus automatically synthesize the netlist (circuit) for you. There are currently two industry standard hardware description languages (HDL), VHDL and Verilog. You can use either language in Quartus. Below is the code for an 8-operation ALU written in both Verilog and VHDL. Notice the similarities between this code and C++ code.

The only difference between entering a schematic drawing and entering code is when you do a File | New, you select Verilog file or VHDL file instead of Block Diagram/Schematic File. Enter either the Verilog code or the VHDL code into Quartus and save it using the name **alu**. Create a symbol for this component, also called **alu**. Test it out on the DE1 board to make sure that it works correctly.

Verilog code for an 8-operation ALU:

```
module alu (
    input [2:0] S,
    input [n-1:0] A, B,
    output reg [n-1:0] F
);
    parameter n = 4;

    always @ (S or A or B) begin
        case (S)
        0:  F <= A;             // pass A through
        1:  F <= A & B;         // logical AND
        2:  F = A | B;          // logical OR
        3:  F = ~A;             // logical NOT A
        4:  F = A + B;          // add
        5:  F = A - B;          // subtract
        6:  F = A + 1;          // increment
        7:  F = A - 1;          // decrement
        endcase
    end
endmodule
```

VHDL code for an 8-operation ALU:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;     -- needed for doing + and -

ENTITY alu IS PORT (
   S: IN STD_LOGIC_VECTOR(2 DOWNTO 0);    -- select for operations
   A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- input operands
   F: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));  -- output
END alu;

ARCHITECTURE Behavioral OF alu IS
BEGIN
   PROCESS(S, A, B)
   BEGIN
      CASE S IS
      WHEN "000" =>  -- pass A through
        F <= A;
      WHEN "001" =>  -- logical AND
        F <= A AND B;
      WHEN "010" =>  -- logical OR
        F <= A OR B;
      WHEN "011" =>  -- logical NOT A
        F <= NOT A;
      WHEN "100" =>  -- add
        F <= A + B;
      WHEN "101" =>  -- subtract
        F <= A - B;
      WHEN "110" =>  -- increment
        F <= A + 1;
      WHEN OTHERS => -- decrement
        F <= A - 1;
      END CASE;
   END PROCESS;
END Behavioral;
```

3. 4-Bit Binary to 7-Segment Hex Digit Display Decoder

A 4-bit binary to 7-segment hex digit display decoder is a circuit that takes a 4-bit binary number and outputs a (hexadecimal) decimal digit on a single 7-segment display. The input to the decoder is a 4-bit binary value. The output from the decoder is a 7-bit value for turning on or off each of the seven LEDs in a 7-segment HEX display. For example, if the input is 0011, then it will turn on the corresponding LED segments to display the digit 3, i.e. segments *a*, *b*, *c*, *d* and *g* as shown next.

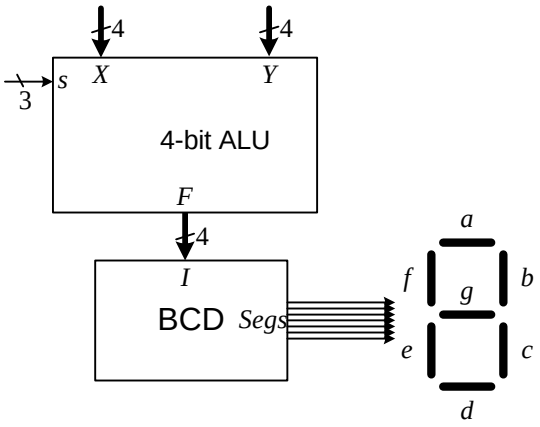
Enter either the Verilog code or the VHDL code into Quartus and save it using the name **BCD**. Create a symbol for this component, also called **BCD**. Test it out on the DE1 board to make sure that it works correctly.

```
// Verilog version

// BCD - 4-bit Binary to 7-segment hex digit
// active low output
// Copyright 2006 Enoch Hwang

module BCD (
  input [3:0] I,
  output reg [0:6] Segs
);

  always @ (I) begin
    case (I)
    // 0=on; 1=off
    0: Segs = 7'b0000001;     // 0
    1: Segs = 7'b1001111;     // 1
    2: Segs = 7'b0010010;     // 2
    3: Segs = 7'b0000110;     // 3
    4: Segs = 7'b1001100;     // 4
    5: Segs = 7'b0100100;     // 5
    6: Segs = 7'b0100000;     // 6
    7: Segs = 7'b0001111;     // 7
    8: Segs = 7'b0000000;     // 8
    9: Segs = 7'b0001100;     // 9
    10: Segs = 7'b0001000;    // A
    11: Segs = 7'b1100000;    // b
    12: Segs = 7'b0110001;    // C
    13: Segs = 7'b1000010;    // d
    14: Segs = 7'b0110000;    // E
    15: Segs = 7'b0111000;    // F
    default: Segs = 7'b1111111;      // all off
    endcase
  end

endmodule
```

```
-- VHDL Version

-- BCD - 4-bit Binary to 7-segment hex digit
-- active low output
-- Copyright 2006 Enoch Hwang

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY BCD IS PORT (
  I: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
  segs: OUT STD_LOGIC_VECTOR(0 TO 6));
END BCD;

ARCHITECTURE Behavioral OF BCD IS
BEGIN
PROCESS(I)
  BEGIN
    CASE I IS
    -- 0=on; 1=off
    WHEN "0000" => segs <= "0000001";     -- 0
    WHEN "0001" => Segs <= "1001111";     -- 1
    WHEN "0010" => Segs <= "0010010";     -- 2
    WHEN "0011" => Segs <= "0000110";     -- 3
    WHEN "0100" => Segs <= "1001100";     -- 4
    WHEN "0101" => Segs <= "0100100";     -- 5
    WHEN "0110" => Segs <= "0100000";     -- 6
    WHEN "0111" => Segs <= "0001111";     -- 7
    WHEN "1000" => Segs <= "0000000";     -- 8
    WHEN "1001" => Segs <= "0001100";     -- 9
    WHEN "1010" => Segs <= "0001000";     -- A
    WHEN "1011" => Segs <= "1100000";     -- b
    WHEN "1100" => Segs <= "0110001";     -- C
    WHEN "1101" => Segs <= "1000010";     -- d
    WHEN "1110" => Segs <= "0110000";     -- E
    WHEN "1111" => Segs <= "0111000";     -- F
    WHEN OTHERS => Segs <= "1111111";     -- all off
    END CASE;
  END PROCESS;
END Behavioral;
```

4. Complete ALU System

Create a logic symbol for both the 4-bit ALU and the 7-segment decoder circuits if you have not already done so from questions 2 and 3. Create a new schematic file. You will use this file as the top level for your project. Insert the two block symbols (ALU and BCD) and connect them together as shown below.



Map the I/O signals as follows:

y_3-y_0	SW[3]-SW[0]
x_3-x_0	SW[7]-SW[4]
s_2	KEY[0]
s_1-s_0	SW[9]-SW[8]
f_3-f_0	LEDR[3]-LEDR[0]
$Segs_0$-$Segs_6$	HEX0[0..6]

Test out your system on the DE1. The output of the ALU will now be in decimal instead of binary.

What happens when the result from the ALU has two decimal digits? Use the Verilog code below to solve this problem.

The code below will decode an 8-bit number to three 7-segment digits.

```
// convert an 8-bit binary number n
// to a 3-digit decimal number
// and display it on three 7-segment LEDs
// The signednumber parameter when set to 1 will interpret
// the 8-bit binary number as two's complement
module displayNumber
    #(parameter signednumber = 0)
    (
    input [7:0] n,
    output reg [0:6] HEX3, HEX2, HEX1, HEX0
  );

    reg [7:0] n2;
    reg [3:0] hundreth, tenth, unit;                    // binary coded decimals

    // convert an 8-bit number n to its hundreth, tenth, and unit decimal digits
    always @ (n) begin
        if ((signednumber == 1) & (n[7] == 1'b1)) begin
            n2 = ~n + 1;                                // two's complement; flip bits and add a 1
            HEX3 = ~7'b0000001;
        end else begin
            n2 = n;
            HEX3 = ~7'b0000000;
        end

        hundreth = n2/100;                              // calculate the hundreth digit
        n2 = n2 - hundreth*100;
        tenth = n2/10;                                  // calculate the tenth digit
        unit = n2%10;                                   // calculate the unit digit

        HEX2 = ~bcd(hundreth);                          // display hundreth digit on HEX2
        HEX1 = ~bcd(tenth);                             // display tenth digit on HEX1
        HEX0 = ~bcd(unit);                              // display unit digit on HEX0
    end

// function to convert BCD to 7-segment LED display
function [0:6] bcd
    (input [3:0] n);

    case (n)
    0: bcd = 7'b1111110;
    1: bcd = 7'b0110000;
    2: bcd = 7'b1101101;
    3: bcd = 7'b1111001;
    4: bcd = 7'b0110011;
    5: bcd = 7'b1011011;
    6: bcd = 7'b1011111;
    7: bcd = 7'b1110000;
    8: bcd = 7'b1111111;
    9: bcd = 7'b1110011;
    default: bcd = 7'b0000000;
    endcase
endfunction

endmodule
```