Lab 5
Arithmetic Circuits in VHDL

Name_____ Class _____ Date _____

OBJECTIVES:

Upon completion of this laboratory exercise, you should be able to:

- Create and simulate a full adder in VHDL, assign pins to the design, and test it on a FPGA circuit board.
- Use VHDL Testbenches in ModelSim to verify the correctness of the circuit design.
- Use a VHDL full adder as a component in an 8-bit two's complement adder/subtractor.
- Create a VHDL hierarchical design, including components for full adders and seven-segment decoders, without using the Quartus II Graphic Editor.
- Design an overflow detector for use in a VHDL two's complement adder/subtractor.

REFERENCE READING:

Making a VHDL Testbench for a Full Adder

Dueck, Robert K., *Digital Design with CPLD Applications and VHDL*:
Chapter 7: Digital Arithmetic and Arithmetic Circuits
7.1 Digital Arithmetic
7.2 Representing Signed Binary Numbers
7.3 Signed Binary Arithmetic
7.6 Binary Adders and Subtractors

EQUIPMENT REQUIRED:

Altera DE1 or DE2 Circuit Board with USB Blaster Download Cable
Quartus II and ModelSim Software
Anti-static wrist strap

EXPERIMENTAL NOTES:

Note: *The designs in this laboratory exercise are to be done entirely in VHDL, without using the Quartus II Graphic Editor for any portion of the design.*

Arithmetic Circuits

Circuits for performing binary arithmetic are based on half adders, which add two bits and produce a sum and carry, and full adders, which also account for a carry added from a less-significant bit (pp. 367-370, *Digital Design with CPLD Applications and VHDL*). Full adders can be grouped together to make a parallel binary adder, with *n* full adders allowing two *n*-bit numbers to be added, generating an *n*-bit sum and a carry output (pp. 370-380, *Digital Design with CPLD Applications and VHDL*).

Lab 5
Arithmetic Circuits in VHDL

A parallel adder can be converted to a two's complement adder/subtractor by including XOR functions on the inputs of one set of operand bits, say input B, allowing the operations A+B or A–B to be performed (pp.380-388, *Digital Design with CPLD Applications and VHDL*). A control input, SUB (for SUBtract), causes the XORs to invert the B bits if HIGH, producing the one's complement of B. SUB will not invert B if LOW, transferring B to the parallel adder without modification. If SUB is also tied to the carry input of the parallel adder, the result is (A + B + 0 = A + B) when SUB = 0 and ($A + \overline{B} + 1 = A - B$) when SUB = 1, where $\overline{B}$ is the one's complement of B and ($\overline{B}$ +1) is its two's complement.

Sign-bit overflow occurs when a two's complement sum or difference exceeds the permissible range of numbers for a given bit size (pp. 357-360, *Digital Design with CPLD Applications and VHDL*). This can be detected by an SOP circuit that compares the operand and result sign bits of a parallel adder, or by an XOR gate that compares carry into and out of the MSB (pp. 386-388, *Digital Design with CPLD Applications and VHDL*).

Using Components in VHDL

VHDL designs can be created using a hierarchy of design entities. For example, a parallel adder can be designed as the top level of a hierarchy that contains several VHDL full adder components. An advantage of this approach is that certain functions, such as full adders, seven-segment decoders, and so on can be designed once and used many times in different design projects. The method of using components in a VHDL file is called **structural** design, in contrast to **dataflow** design, which uses signal assignment statements and similar constructs or **behavioral** design, which is based on descriptions of circuit operation.

Structural design requires:

1. One or more complete VHDL design files that can be used as components. These are separate from, but used by, the top level of the design hierarchy.

2. Component *declaration* in the top-level VHDL file, similar to the entity declaration of the component.

3. Component *instantiation* in the top-level VHDL file, which maps the inputs and outputs of the component to the ports and signals of the top-level design.

Lab 5
Arithmetic Circuits in VHDL

The general form of a top-level design entity using components is:

```
ENTITY entity_name IS
    PORT ( input and output definitions);
END entity_name;

ARCHITECTURE arch_name OF entity_name IS
    component declaration(s);
    signal declaration(s);
BEGIN
    Component instantiation(s);
    Other statements;
END arch_name;
```

The components in the above file are complete designs in their own right, defined in different files. As long as the folder containing the component is on a recognizable library path, the top-level file will compile, using the component files.

Device and pin assignments can be made directly in the VHDL files, without requiring the components to be inserted as symbols in a QUARTUS II graphic design file.

Structural design of parallel adders is discussed in more detail on pp. 373-380 of *Digital Design with CPLD Applications and VHDL*.

PROCEDURE:

Full Adders

1. The logic diagram for a full adder is shown in Figure 1. Use this diagram to write a VHDL file called **full_adder.vhd**. ***Do not use the graphic editor.***

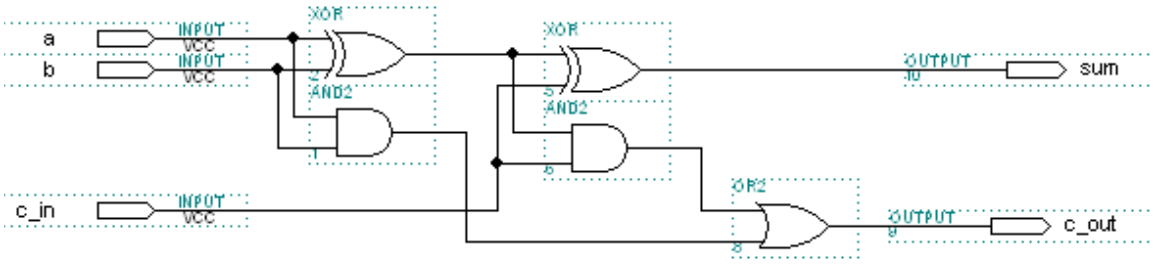

Figure 1 Full Adder

Set the project to the current file and save the design. Assign pin numbers to the design, as shown in Table 1. This can be done directly from the Assign menu in the Quartus II text editor.

Function	I/O Device	Pin (DE1)	Pin (DE2)
a	SW[2]	M22	P25
b	SW[1]	L21	N26
c_in	SW[0]	L22	N25
c_out	LEDR[1]	R19	AF23
sum	LEDR[0]	R20	AE23

Table 1 Pin Assignments for a Full Adder

2. Compile the file.

3. Refer to *Making a VHDL Testbench for a Full Adder* (separate document). Follow the steps to make a testbench and simulation for the full adder.

Instructor's initials _____

4. Download the full adder design to the DE1/DE2 board. Take the truth table of the full adder to verify its operation. Show the result to your instructor.

Instructor's initials _____

Lab 5
Arithmetic Circuits in VHDL

Parallel Adder

1. Create an 8-bit parallel adder in VHDL, using a GENERATE statement and the component **full_adder.vhd** from the previous section. ***Do not use a graphic design file.*** Compile the file.

2. Test the parallel adder with the following testbench file. Show the simulation to your instructor.

Instructor's initials _____

```
-- parallel_adder_8bit_tb.vhd
-- test bench for parallel_adder_8bit.vhd
-- simulation criteria:
--      9-bit output = sum of two 8-bit inputs and a carry
--      Test by applying 16 left-shifted 1s, then 16 0s through the inputs
--      Refer to pages 383-385 in textbook for further explanation
library ieee;
use ieee.std_logic_1164.all;

entity parallel_adder_8bit_tb is
end parallel_adder_8bit_tb;

architecture test_bench of parallel_adder_8bit_tb is
    component parallel_adder_8bit
        port(
            a, b: instd_logic_vector(8 downto 1);
            c_in: instd_logic;
            c_out : out  std_logic;
            sum : out   std_logic_vector(8 downto 1)
            );
    end component;

    signal operands  : std_logic_vector(16 downto 1);
    signal result        : std_logic_vector(8 downto 1);
    signal c0, c8        : std_logic;
begin
    -- instantiate full adder with test signals
    parallel_add: parallel_adder_8bit
                                port map(a      => operands(16 downto 9),
                                         b      => operands(8 downto 1),
                                         c_in   => c0,
                                         c_out  => c8,
                                         sum    => result);

    -- set input carry LOW
    c0 <= '0';
    -- stimulus process (generates simulation input waveforms)
    stim_proc: process
    begin
        -- Set all operand bits to '0' for 100 ns
        operands <= (others => '0');
        wait for 100 ns;

        -- Count from 1 to 16 (shift in '1' each time)
        for i in 1 to 16 loop
            operands <= operands(15 downto 1) & '1';
            -- Set time interval for each input state
            wait for 100 ns;
        end loop;
```

```
        -- Count from 1 to 16 (shift in '0' each time)
        for i in 1 to 16 loop
            operands <= operands(15 downto 1) & '0';
            -- Set time interval for each input state
            wait for 100 ns;
        end loop;

        wait;       -- stall here (simulation done)
    end process;
end test_bench;
```

(This looks harder to understand than it really is. The generated pattern allows easy testing of the circuit on the DE1/DE2 board. All you have to do is set all DIP switches LOW, then make them HIGH in sequence, from b[1] to a[8], then go back and make them LOW in the same sequence. Very easy to test.)

4. Assign pins to the 8-bit adder as shown in Table 3. Refer to the DE1/DE2 User's Guide for pin numbers.

Function	Device	Pin
a8	SW[16]	
a7	SW[15]	
a6	SW[14]	
a5	SW[13]	
a4	SW[12]	
a3	SW[11]	
a2	SW[10]	
a1	SW[9]	
b8	SW[8]	
b7	SW[7]	
b6	SW[6]	
b5	SW[5]	
b4	SW[4]	
b3	SW[3]	
b2	SW[2]	
b1	SW[1]	
c0	SW[0]	
c8	LEDR[0]	
sum8	LEDG[7]	
sum7	LEDG[6]	
sum6	LEDG[5]	
sum5	LEDG[4]	
sum4	LEDG[3]	
sum3	LEDG[2]	
sum2	LEDG[1]	
sum1	LEDG[0]	

Table 2 Pin Assignments for an 8-bit Parallel Adder

5. Compile the file and download the design for the 8-bit parallel adder to the DE1/DE2 board. Test the operation of the 8-bit parallel adder by applying the combinations of inputs A and B listed in Table 2. Show the results to your instructor.

Instructor's initials _____

Two's Complement Adder/Subtractor (50% Bonus)

1. Modify the 8-bit adder you created in the previous section to make an 8-bit two's complement adder/subtractor. Include an overflow detector that will turn on an LED when the output of the adder subtractor overflows beyond the permissible range of values for an 8-bit signed number. Display the sum outputs on LEDs, as with the previous 8-bit adder, but also add a pair of seven-segment decoders to display the result numerically on the board's seven-segment display.

Make the design entirely in VHDL, using components where required. Do not use a graphic file.

2. Create a simulation based on the testbench for the 8-bit adder, with appropriate modifications to include the ADD/SUBTRACT functions. The simulation should contain all combinations with the **SUB** input LOW, then again with the **SUB** input HIGH. The simulation should also account for the overflow output.

3. Leave pin assignments the same as for the 8-bit adder, except for the following changes:

Function	Device	Pin
sub	SW[0]	
overflow	LEDG[8]	
7 Segment (MSD)	HEX5[0..7]	
7 Segment (LSD)	HEX4[0..7]	

Table 4 Pin Assignment Changes for 8-bit Adder/Subtractor

4. Compile the file and download it to the FPGA board. Test the operation of the circuit by applying the A and B inputs from simulation test data. Show the results to your instructor.

Instructor's initials _____