Scarlett Jones
CSCE 3730
Homework #2
Due: 10/04/2016 11:55 PM

1. Discuss in detail the architecture of a FPGA.
    a. Logic Elements
        i. These are the combinational functions and registers. Logic elements use SRAM as a lookup table for combinational functions. They are programmable for input connections and internal functions. Logic Elements typically have 4 to 6 inputs and include a register.
    b. Interconnections
        i. Interconnections are used to connect the different logic elements. They may overlap. These connections are done with programmable wiring. The wires are organized into channels (channels have many wires). Not all wires will be used. There will be short wires for local LE connections, global wires for long distance/buffered communication, and special wires for clocks.
    c. Input/Output pins
        i. These are located on the sides of the FPGA so that there is room on the top of the device for the LE's and interconnections.
2. Briefly explain the different FPGA programming technologies.
    a. SRAM
        i. Can be programmed many times, but must be programmed at power up. It cannot hold memory once the power is off.
    b. Antifuse
        i. This can only be programmed once. Antifuse makes connections with electrical signals.
    c. Flash
        i. This can also be programmed many times, but it can hold its memory in the flash even when the power is not on. Flash also allows reprogramming without boot-up procedure.
3. Explain in detail with an example how a function is implemented in a SRAM LUT of a FPGA.
    a. A function of 2^n input combinations can be implemented in an SRAM N-input LUT of an FPGA.
    b. Say you have a function $F = \bar{A}B + C$. The truth table will have 8 input combinations because 2^3=8. This will use a 3 input LUT.
4. What is the difference between configuration and programming? Why a configuration ROM is used?
    a. In FPGA configuration, bits stay at the device they program. The bits don't need to be recalled from somewhere else. A configuration bit controls a switch or a logic bit to perform an operation.
    b. In CPU programming, instructions must be fetched from a memory source. Instructions written select complex operations.
    c. A configuration ROM is used to save the logic elements apart from the FPGA. If the power goes out, the SRAM's lose their memory. Using a ROM will allow you to have a backup of your configuration.
5. What are the resources in a CLB and a Slice?

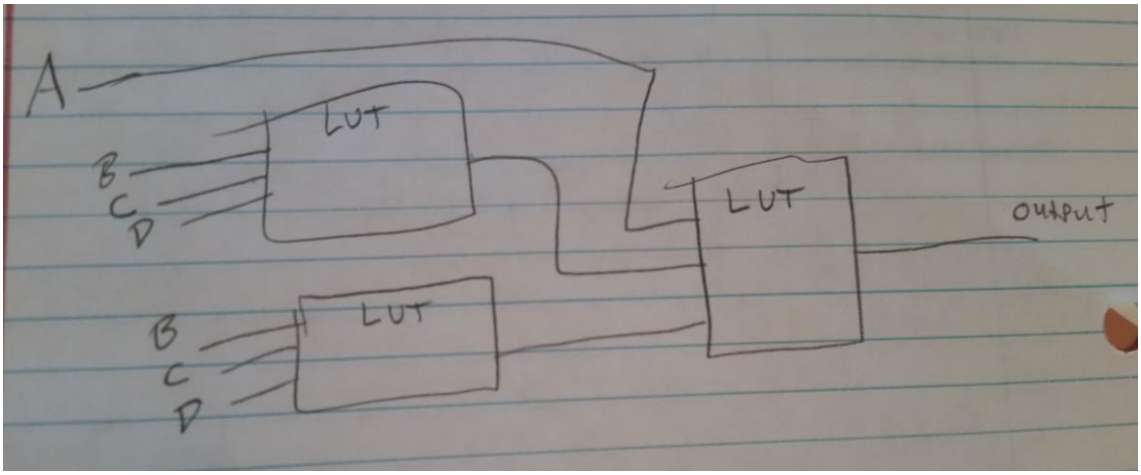
Scarlett Jones
CSCE 3730
Homework #2
Due: 10/04/2016 11:55 PM

a. Configurable Logic Blocks (CLB) have slices (SLICEL and SLICEM), 6-input LUTs, distributed RAM, shift register, and flip-flops.
b. Each slice has four 6-input LUTs, eight flip-flops, multiplexers, and arithmetic carry logic.

6. Map the function F1 (a, b, c, d) = ab' + b'd + bc using 3-input LUTs. (Hint: Derive the truth table for the function and map the function on to 3-input LUTs.)
   a. Truth Table

A	B	C	D	F1(A,B,C,D) = $A\bar{B}+\bar{B}D$+BC
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

   b. 3 Input LUTS



7. Explain the steps involved in a combinational design process?
   a. Step 1: Capture the function

i. Create a truth table or equations, depending on what is easier for the given problem, to describe the desired behavior of the combinational logic.

b. Step 2: Convert to equations
   i. If you chose to describe the function with a truth table in step 1, then turn it into an equation. To do this, OR all of the minterms for that output. Simplify equations if desired. If you started with an equation in step 1, there is nothing to do for this step.

c. Step 3: Implement as a gate based or LUT based (FPGA) circuit
   i. For each output, create a circuit corresponding to the output's equation. LUT-based or FPGA implementation is done using a Hardware Description Language (HDL).

8. Explain the different VHDL modeling styles with an example. (Do not use the full adder example given in the lecture notes as an example for this question.)
   a. Behavioral Model: Uses process to exactly describe the functionality of the block.
      i. Describe the function z = x + y with the behavioral model.

```
LIBRARY ieee ;

USE ieee.std_logic_1164.all ;

ENTITY or_func IS

        PORT ( x, y : IN STD_LOGIC ;

                z : OUT STD_LOGIC) ;

END or_func ;

ARCHITECTURE Behavior OF or_func IS

        signal input : STD_LOGIC_VECTOR(1 DOWNTO 0);

BEGIN


        PROCESS(x, y) -- Sensitivity list, triggers when A, B, or C value changes

                BEGIN

                        input <= x&y;

                        WITH input SELECT

                                                z <=    '0' when "00",

                                                        '1' when "01",

                                                        '1' when "10",

                                                        '1' when "11",
```

```
                                                    '0' when OTHERS;

        END PROCESS;

END Behavior;
```

b. Structural Model: Uses lower level blocks to build the design.
    i. Describe the function z = x + y with the structural model.

```
LIBRARY ieee ;

USE ieee.std_logic_1164.all ;

USE ieee.std_logic_signed.all ;

ENTITY or_func IS

        PORT (x, y : IN STD_LOGIC ;

                z : OUT STD_LOGIC) ;

END or_func ;

ARCHITECTURE structural OF or_func IS

BEGIN

        z = x OR y;

END structural ;
```

c. Data Flow Model: Uses concurrent statements.
    i. Describe the function z = x + y with the data flow model.

```
LIBRARY ieee ;

USE ieee.std_logic_1164.all ;

USE ieee.std_logic_signed.all ;

ENTITY or_func IS

        PORT ( x, y : IN STD_LOGIC ;

                z : OUT STD_LOGIC) ;

END or_func ;

ARCHITECTURE DataFlow OF or_func IS

BEGIN

        z <=        '0' WHEN x='0' AND y='0' ELSE

                    '1' WHEN x='0' AND y='1' ELSE
```

‘1’ WHEN x=’1’ AND y=’0’ ELSE

‘1’ WHEN x=’1’ AND y=’1’;

z = x OR y;

END DataFlow ;

9. Simplify the function and write a behavioral VHDL code to implement the function F2 (a, b, c) = ab’ + abc’ + a’b’ + acb’. Write a test bench to test the VHDL implementation.
   a. K-MAP to simplify function

	$\bar{C}$	C
$\bar{A}\bar{B}$	1	1
$\bar{A}B$	0	0
AB	1	0
$A\bar{B}$	1	1

$$F2(A,B,C) = \bar{B} + A\bar{C}$$

   b. Behavioral VHDL code

```
LIBRARY ieee ;

USE ieee.std_logic_1164.all ;

ENTITY problem9 IS

        PORT ( a, b, c : IN STD_LOGIC ;

                f2 : OUT STD_LOGIC) ;

END problem9 ;

ARCHITECTURE Behavior OF problem9 IS

        signal input : STD_LOGIC_VECTOR(2 DOWNTO 0);

BEGIN


        PROCESS(a, b, c) -- Sensitivity list, triggers when A, B, or C value changes

                BEGIN

                        input <= a&b&c;

                        WITH input SELECT

                                                f2 <=     ‘1’ when “000”,

                                                        ‘1’ when “001”,
```

```
                                        '0' when "010",
                                        '0' when "011",
                                        '1' when "100",
                                        '1' when "101",
                                        '1' when "110",
                                        '0' when "111",
                                        '0' when OTHERS;
        END PROCESS;
END Behavior;
```

c. Test Bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY problem9_tb IS
END problem9_tb;
ARCHITECTURE behavioral OF problem9_tb IS
        COMPONENT f2
        PORT( a, b, c : IN STD_LOGIC;
                f2 : OUT STD_LOGIC);
        END COMPONENT;
        SIGNAL a, b, c, f2 : STD_LOGIC;
BEGIN
        UUT: f2 PORT MAP(
                                        a=>a,
                                        b => b,
                                        c => c,
```

```
                                        f2 => f2);

-- *** Test Bench - User Defined Section ***

        tb : PROCESS

        BEGIN

                a<='0'; b='0'; c='0';

                wait for 10 ms;

                a<='0'; b='0'; c='1';

                wait for 10 ms;

                a<='0'; b='1'; c='0';

                wait for 10 ms;

                a<='0'; b='1'; c='1';

                wait for 10 ms;

                a<='1'; b='0'; c='0';

                wait for 10 ms;

                a<='1'; b='0'; c='1';

                wait for 10 ms;

                a<='1'; b='1'; c='0';

                wait for 10 ms;

                a<='1'; b='1'; c='1';

                wait for 10 ms;

        WAIT; -- will wait forever

        END PROCESS;

-- *** End Test Bench - User Defined Section ***

END behavioral;
```