Lab 5: Summing up Programmable Logic

Name: _____ Lab Partner: _____

Until now, we have treated FPGAs as collections of gates the computer can connect together. FPGAs are actually constructed from a large number of Configurable Logic Blocks (CLBs) connected in rows and columns on the chip, connected with a switching matrix to route signals between the CLBs. Configurable logic blocks in a Xilinx Spartan 3 have the following structure, taken from *Spartan-3 FPGA Family: Complete Data Sheet.*

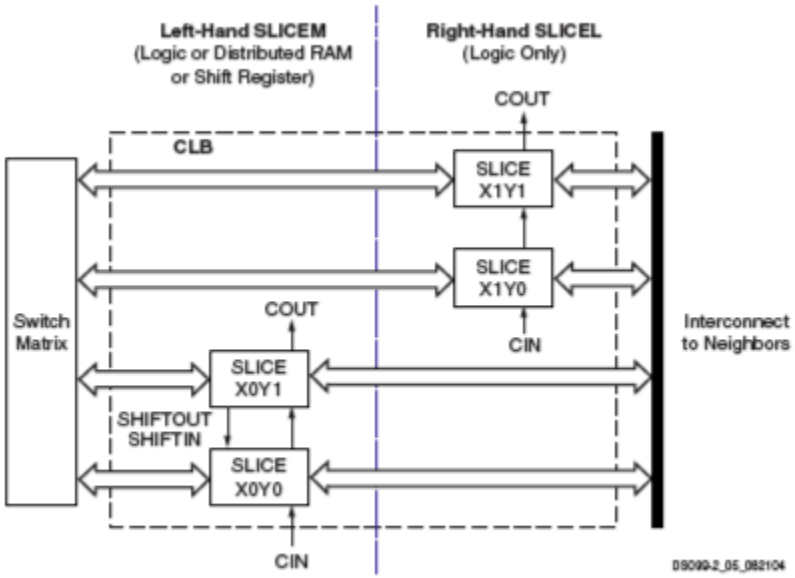
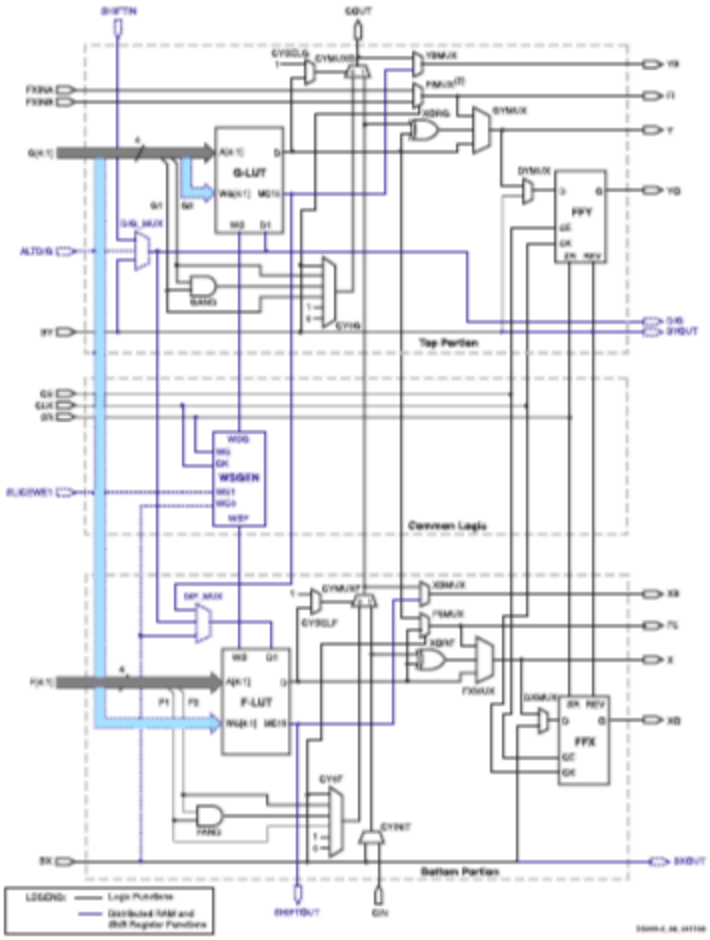


Figure 5: **Arrangement of Slices within the CLB**

Inputs to the CLB come from the switch matrix and are routed to one or more of four slices of the CLB. The slices are also connected in columns so they can be cascaded. The cascading inputs and outputs, labeled `CIN` and `COUT` respectively, are used to implement cascading circuits. A cascading circuit is made up of stages. Each stage has direct inputs, cascading inputs, direct outputs, and cascading outputs. The direct normal inputs and outputs are inputs and outputs of the circuit that are visible from the outside world. The cascading outputs of one stage connect to the cascading inputs of the next stage, passing a result along the length of the circuit. The ripple carry adder is an example of a cascading circuit with the addends being direct inputs, the sum being a direct output, the carry in to each stage being a cascading input and the carry out of each stage being cascading outputs.

The logic inside each of the slices takes the following form, also taken from *Spartan-3 FPGA Family: Complete Data Sheet*



Notes:
1. Options to invert signal polarity as well as other options that enable lines for various functions are not shown.
2. The index i can be 6, 7, or 8, depending on the slice. In this position, the upper right-hand slice has an F8MUX, and the upper left-hand slice has an F7MUX. The lower right-hand and left-hand slices both have an F6MUX.

Figure 6: **Simplified Diagram of the Left-Hand SLICEM**

For combinatorial circuits our biggest concern will be the LookUp Tables (LUTs), though there are a few other resources in the slice (including an XOR gate, which is handy for addition, and a MUX). The LUTs in the Spartan-3 take up to four inputs and have a single output. A LUT takes some number of inputs and outputs a preprogrammed value. It is essentially a programmable truth table; any function of four variables is equally expensive to program into a LUT.

The important thing to infer from your understanding of FPGA structure is that different functions will take different numbers of Slices to implement, and thus different amounts of time for signals to propagate through the circuit. The tools in ISE provide details on both when you synthesize and implement your design.

Information about size and timing for a design can be found in the Synthesis Report under the headers “Timing Summary” and “Logic Utilization”. The reports will look like the following, in which the relevant lines are in bold.

```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 19.776ns

Logic Utilization:
  Number of 4 input LUTs:              16 out of   3,840    1%
Logic Distribution:
  Number of occupied Slices:                12 out of   1,920    1%
    Number of Slices containing only related logic: 12 out of 12
     100%
    Number of Slices containing unrelated logic: 0 out of  12  0%
      *See NOTES below for an explanation of the effects of unrelated
     logic
Total Number of 4 input LUTs:          16 out of   3,840    1%
  Number of bonded IOBs:               26 out of     173   15%

Total equivalent gate count for design:  96
```

Verilog Test Fixtures

One of the major benefits of designing in Hardware Description Languages is that testing can be automated by writing sections of Verilog code to generate test inputs, instead of clicking in a waveform. This practice is known as procedural testbenches, and is standard practice on larger projects. A procedural testbench can be added to a project in ISE much the same way as a verilog file, but as a “Verilog Test Fixture” from the new file menu. It will automatically be filled out with an instance of your top level module if it is created this way, but will not contain code to generate inputs. An example test bench, which meets the requirements, to test a 3x8 encoder implemented in two different coding styles is included.

Verilog Behavioral Coding

Another primary benefit of designing in Hardware Description Languages is that we typically can

describe our modules using high-level coding constructs, such as addition, subtraction, branching and choice using case statements, among other things. When we use such programming techniques for synthesis and the subsequent configuration of hardware, we are in essence leaving the low level details up to the software. Such code is called *behavioral* because it describes what the circuit does, rather than how it is implemented. Our seven segment display code is an example behavioral coding, while using gate-level primitives is structural coding (there is a place for this style of coding usually in optimization).

Now let us revisit the case statement (introduced in the lab 3 description). It is useful for representing many forms of combinational logic (the other form of logic involves memory – next week). In particular, they basically resolve the problem's specification into a form of truth table form, where we map each valuation of the input directly to values of the output(s). Using the case statement correctly insures that there is no ambiguity in the assignment of the outputs. Ambiguity in the mapping may be misinterpreted by the synthesis and configuration software and lead to errors or weird implementations. While the if-else behavioral construct is useful, large nested if-else statements can leave certain valuation of the inputs unspecified and this can lead to real problems.

As an example, consider the following 4 input/2output system. This module takes two 2-bit unsigned integers and compares them, if x > y then f = 1(ie.2'b01) , if y>x then f=2(ie.2'b10) , else f = 0 (2'b00, they are equal).

```
module testMod1(input [1:0] x, input [1:0]y, output [1:0] f);
reg [1:0] f;    // required if value of output , (or wire) , is determined in an always block, a variable
                // may only have it's value assigned in one always block.  Otherwise it generates a
                // "multisource" error.
        always @(x,y)
        begin
                case({x,y})  // {x,y} concanates x,y. treated as 1 4-bit #
                        4'b0000:  f = 0;
                        4'b0001:  f = 2;
                        4'b0010:  f = 2;  // 0<2, x<y f=2
                        4'b0011:  f = 2;
                        4'b0100:  f = 1;
                        4'b0101:  f = 0;
                        4'b0110:  f = 2;
                        4'b0111:  f = 2;
                        4'b1000:  f = 1;
                        4'b1001:  f = 1;
                        4'b1010:  f = 0;
                        4'b1011:  f = 2;
                        4'b1100:  f = 1;
                        4'b1101:  f = 1;
                        4'b1110:  f = 1;
                        4'b1111:  f = 0;
                        default: f = 0;  // not needed here but very important
                                         // when all valuations of input aren't
                                         // considered. Eliminates ambiguity.
                endcase
        end
endmodule
```

THE LAB: Rock, Paper, Scissors

Over the next two labs, we will implement a simple Rock, Paper, Scissors game where one player plays against the FPFA. In this first part we are going to design, simulate, and test the module (logic block) that will make the choice of a winner between the player (***input [1:0] p;***), and the fpga(***input [1:0] c;***). The output for this module is a 2-bit value (***output [1:0] win;***) . A choice of rock is defined as 00(2-bit binary value for either p or c), a choice of paper is defined as 01, and scissors is defined as 10 (11 is unused and should never occur). The choice of a winner if defined as 00 – draw, 01 -player wins, 10-fpga(computer) wins, 11 is unused. Your first task is to code a module that outputs the correct value for winner based on the four inputs Rock, Paper, Scissors and the rules for the game: rock beats scissors, scissors beats paper, and paper beats rock. For example, if p = 00(player picks rock) and c=01 (computer picks paper) then win=10 (computer wins). A module very similar to the one shown on the previous page can be used to implement this. Hint: use ***default: win=0;*** for the default case and you will not need to include cases that should never occur. Then design a simple test fixture(exhaustive but no error signal required) and simulate.

For the second part of this lab design, code and simulate a simple 2-bit input, 8 bit output seven-segment display module. The eigth bit is for the decimal point. This module will take as input the win output of the first module and output to a seven segment display. The display will output d (draw – elements b,c,d,e,g on the display) if input is 00 in binary, h(player wins – elements c,e,f,g) if the input is 01 in binary, and c(computer wins – elements d,e,g) if the input is 11. This is very similar to the R,P,S display of lab 3 but it will need to be modified.

Prelab Questions

1. Sketch out you Verilog code for both modules described above. This time handwritten code is fine.
2. Sketch out the test fixtures you will use for these modules. Use the provided code as a template.

Experiments

A big part of this lab is to better understand the place of simulation and analysis in design. Make sure to record the Timing Summary and Logic Utilization sections of the synthesis report for each design, Also save copies of the RTL and Technology schematics (produced in the synthesis section in ISE) for your designs to include in your report. These diagrams show how the hardware is logically configured (equivalent gates), as well as how it is actually mapped to the hardware on the board (LUTs, etc.).

1. Design, develop, and test the Rock, Paper, Scissors decision logic block. Simulate it exhaustively using the test fixture you develop.

TA signature for part 1: _____

1. Design, develop, and test the Rock, Paper, Scissors decision logic block. Simulate it exhaustively using the test fixture you develop.

TA signature for part 2: _____

2. Now integrate these modules into a new top module (and project – import files); then program this module to the Spartan3 board. Use switches for the inputs, and display the result (including carry-out) on a seven segment display.

TA signature for part 3: _____