ECE 152A LABORATORY 2

Objectives:

1. Understand the trade-off between time- and space-efficiency in the design of adders. In this lab, adders operate on unsigned numbers.

2. Learn how to write Verilog code for different adder implementations. Apply the principles of hierarchical design. Verify each design by simulation.

3. On an FPGA, implement a 4-bit adder that is made up of two 2-bit adders. The adder operates on unsigned numbers and should be able to detect overflow.

Deliverables and Due Dates:

Lab # 2 Prelab is due at the beginning of your lab section

Please hand in written responses to the following questions (only 1 submission per group)

In addition, you must come to the lab having read the rest of this document. If you have also written the Verilog code for the two different implementations, you will have maximum time to work on the physical implementation. *(Please note that we take the Honor Code very seriously. You must write the Verilog code with only your own team.)*

Part A: Worst-case delay and # of gates cost analysis of different implementations.

In this step, we will use analytical techniques and the simple gate delay model to compare three different implementations of the 4-bit adder with respect to their worst- case delay and number of gates performance. (We will see more sophisticated adders such as the carry look-ahead adder later in the course. But the design of those adders use the same ideas of time- and space-efficiency.)

In all of the steps below, Your adder design must be able to detect overflow (EG it has a carry-out), and it should have a 1-bit carry-in. This will allow you to cascade them to build larger adder structures.

Step A.1. In this step, we implement the 4-bit adder as four full-adders in cascade. This is called a "ripple carry adder" because the carry bit has to ripple from one element to the next. We expect that the worst-case delay of the ripple carry adder will be dominated by the carry chain. In this exercise, we want to quantify such intuitions.

(a) Write the truth table and gate implementation for the full adder.

(b) Using the simple gate delay model (which assumes that each gate, regardless of the type and number of inputs, has the same delay, D), find an expression for the maximum (i.e. worst-case) delay of the 4-bit ripple carry adder. In addition, find a "critical input transition" that will result in this maximum delay.

(c) What is the total number of gates this implementation uses?

Step A.2. Now, we want to estimate the delay and the space-complexity of the 4- bit adder when it is designed as a combinational logic block from the truth table directly and then minimized using Karnaugh maps to arrive at a minimum sum-of-products expression for each output.

The problem is that the truth table for this 4-bit adder is large. (How many rows does it have?) So, in this section, instead of finding the explicit expressions, we will try to estimate the delay and space-complexity.

(a) Write out the logic expressions for each of the sum bits in terms of the internal sums and carry bits

(b) Now, back substitute your expression for S0 into S1. Do you see what will happen for the next 2 bits? What input variables will sum[2] and sum[3] depend on?

(c) How many logic levels would you expect this circuit to have? Use this to draw an approximate schematic of the network.

(d) Estimate the number of gates this circuit would use. Identify the critical path and estimate the delay.

Step A.3. Now, we implement the 4-bit adder by using two 2-bit adders. Each of the 2-bit adders is designed from a truth table and using logic minimization.

(a) Write down the truth table for a 2-bit adder. Then, find minimum sum-of-products expressions for the outputs. Draw a block diagram schematic to show how to exactly cascade two 2-bit adders to get a 4-bit adder.

(b) Compute the worst-case delay and find a critical transition.

(c) Compute the number of gates of this implementation.

Step A.4. Now, compare quantitatively the time- and space-efficiencies of the implementations in the previous steps. (For A.2, we have only estimates from our analysis.) What is your conclusion? When should we prefer which implementation?

Lab #2 is due at the end of lab section

Please check off your completed circuit with your TA before the end of lab section. This comprises completing all of the remaining steps in this lab and demonstrating the correct functionality of your simulations and the final implementation in FPGA.

The grades for Part B are based on demonstration of functionality for each step. For each step in Part B, partial credit will NOT be given.

Design Strategy (Outline):

We will write the Verilog code for the implementations of the ripple carry, and cascade of 2-Bit adders. For the implementation of the ripple carry, do the following steps:

Step 1: Write the design in Verilog.

Step 2: Simulate the design using ModelSim and verify operation.

Next, we will implement the cascade of 2-Bit adders on the FPGA

Step 3: Compile and download to FPGA.

Step 4: Test the FPGA implementation and verify operation.

Your design is to be demonstrated to your TA.

Functional Simulation, FPGA download and testing for functionality

In this part of the lab, we will write two of the implementations of the previous part in Verilog and write a testbench to verify their operation.

Then, for simplicity, we will download to the FPGA only one of these 4-bit adders: the one that is comprised of two 2-bit adders in cascade. Then, we will test the correctness of this implementation in FPGA directly.

Step 1 for Ripple Carry:

In order to facilitate the design process, we use "hierarchical design". This is a term you are likely to hear many times throughout your engineering career. To implement the 4-bit ripple carry adder, you will implement a full-adder module in Verilog and use these full-adder modules to implement a 4-bit ripple-carry adder. We consider the full-adder to be the “lowest level” of the hierarchy and the 4-bit ripple-carry adder the “top level.” First, you should write down the module declarations for the full-adder and the ripple-carry adder. (The comments that begin with "Students:" is a comment for you, not a comment that should appear in your final program.)

```
//Students: Always start your module declarations with a description
of //what the module does:
//1: The module "fourBitAdder_FourByOne implements a 4-bit adder
//using four 1-bit full adders in cascade. The phrase "FourByOne"
refers to the fact that we have four 1-bit adders in cascade.
//Students: write down the inputs and outputs, do not forget overflow
//detection.
module fourBitAdder_FourByOne (....);
input ...; output ...;
//Students: implementation goes in here.
endmodule
```

```
//Students: write down the inputs and outputs module fullAdder
(.....);
input ...; output ...; //Students: implementation goes in here.
endmodule
```

Next, fill in the code for the fourBitAdder_FourByOne in Verilog assuming that you have the fullAdder module. Note that you will instantiate the full adder module inside the fourBitAdder_FourByOne module.

Then, fill in the Verilog code for the fullAdder using only the gate-level primitives (e.g. and, or, not gates).

Run the Modelsim compiler and check that it compiles.

Step 1 for 2-bit cascade:

Write the module for the implementation, just as we did for the ripple carry implementation.

You MUST use the following name for this module: fourBitAdder_TwoByTwo (This refers to the fact that we have two 2-bit adders in cascade.)

Step 2: In this step, simulate both 4-bit adder designs using Modelsim. Set up a Verilog test bench and simulate your design. From your simulations, verify that both of your implementations are operating correctly. Refer to Lab 1 if you need a reference for testbenches.

Step 3: In this step, we compile and download the Verilog design to the FPGA. You will use Quartus II to compile and create an .rbf file. You will then use FPGAtool.exe to download the .rbf file to your board, exactly as you did last week in Lab 1. You can find the documentation regarding the FPGA tools here: https://www.ece.ucsb.edu/courses/ECE152/digilab-fpga/

Step 4: In this step, we test the design on the FPGA. For external logic, you have to use 5 LEDs as your outputs – 4 LEDs for "sum0", "sum1", "sum2", "sum3", and one LED for "overflow". You will use 9 DIP switches as your inputs – four of them for A0, A1, A2, A3; four of them for B0, B1, B2, B3; and one of them for Cin. Note that you will need either pull-up or pull-down resistors for your DIP switches.

Now you are ready to test various combinations of inputs and observe the output. Verify that the outputs (LEDs) are functioning correctly.

Supplies

- **Digilab FPGA (**https://www.ece.ucsb.edu/courses/ECE152/digilab-fpga/
- **5 Red or Green through-hole LED's**
- **Dip switch with at least 9 switches**
- **Breadboard**
- **Jumper Wire**
- **Cables for using the DMM**