EECE2323	Digital Logic Design Lab	Lab 1

8-bit Adder

1 Objective

In this lab you will create an 8-bit adder with overflow flag using Verilog. This lab is divided into two parts; in the first part you will design the adder in Verilog and simulate it. In the second part of the lab, you will implement the adder on the TUL PYNQ board and display the results on the LEDs.

2 Overview

The *8-bit Adder* that you will design has two 8-bit data inputs, one 8-bit data output and one overflow flag output.

Figure 1 shows the 8-bit adder interface.

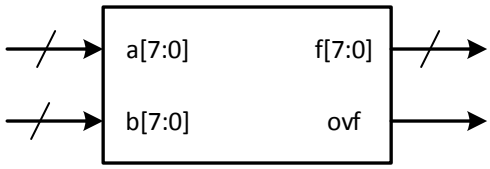

Figure 1: 8-bit Adder

a(7:0) and *b(7:0)* are the data inputs. Each input is eight bits wide and is interpreted as a signed, 2's complement number. The result of the operation is produced on *f(7:0)*, which is also an 8-bit signed number. A flag bit indicating *overflow* is produced on the *ovf* output.

In this lab we will be working with signed numbers represented in 2's complement. Since your program treats the input vectors as 2's complement signed values, you must watch to see if your arithmetic sets the overflow flag, indicating that the result is wrong.

==========

Remember that you are not dealing with a "carry" flag. When working with signed, 2's complement math, you have an overflow flag. The carry flag is only relevant for unsigned numbers.

==========

The second part of this lab is to show the results of the add operation on the TUL PYNQ board. You will connect the ports of your adder to a top module `adder8_top` which will take care of setting the operands, operation and result for you.

3 8-bit Adder

The first part of this lab assignment is to create an *8-bit Adder* using the Verilog HDL. This unit will implement the *a plus b* operation.

3.1 Prelab 1

Download the prelab assignment for this part from the course webpage. Keep in mind that the TAs may post different assignments in different semesters.

3.2 Entering Your Design

Start Vivado IDE. Create a new project called `lab1`. Make sure that the device type is set to xc7z020clg400-1 which is the Xilinx FPGA part on the PYNQ board you are using in this lab.

Create a new Verilog source called `eightbit_adder`.
You should have two input ports: *a[7:0]* and *b[7:0]*. Your output ports are *f[7:0]* and *ovf*.

Write Verilog code for the adder that generates the output and overflow flags.

3.3 Simulating Your Design

The 8-bit adder circuit has sixteen bits of input: two eight-bit data inputs. For the 8-bit adder there are $2^{16} = 65536$ possible vector combinations, which are a lot of combinations to check, even for this simple design. Instead of exhaustively testing all inputs, you should generate a set of test vectors that *cover* the inputs and outputs. These test vectors should check that all the inputs and outputs can assume both a zero and a one value. You should have come up with these test vectors in your prelab.

Write Verilog code for a testbench for your adder that includes the test vectors you developed in your prelab.

Create a new testbench module called `adder8_tb`.
Enter the test vectors that you have chosen into the testbench, save the testbench, and run it in XSIM. Make sure that you set the `Simulation Run Time` property to an appropriate value before running your simulation. Check that you are getting the output values that you expect.

Show your waveform to a TA for a grade for first of this lab.

==========
If your simulation is not working properly, the first thing to do is to check the Transcript pane in Vivado, especially the TCL Console and Messages tabs. These should give you some ideas for debugging.
==========

4 Displaying the Results on the LEDs

The second part of this lab is to show the results of the add operation on the LEDs that are available on the PYNQ board.

Because the PYNQ board is small, we will be testing the adder in halves. First, you will test the lowest 4 bits. Then you will test the highest 4 bits along with the overflow flag.

In this lab you are going to connect the ports of your 8-bit adder to the top module in order to help handle the inputs from the hardware.

4.1 Entering Your Design

Start Project Navigator and open lab1 if it is not already open.

The top level module file is called adder8_top.v. You should get this file from the course web site. Open the file for edit, you will see that the top module is called: adder8_top.

Tip for coding: name each file exactly the same as the module name.

Add your 8-bit Adder to this module as a sub-module.

If you add the `adder8_top.v` file to your Vivado project as a design module, the other file will automatically be treated as sub-module by Vivado.

Open the adder8_top.v file. The port definition of this module should be exactly the same as those in the `eightbit_adder` module. Note that you do not need to interface to these ports. This has been done for you.

4.2 Testing the 8-bit adder in Hardware

The last step is to test your design in hardware. Remember you need to have a constraints file for synthesis and for the placer.

Add the XDC file to your project by clicking on Add Source under the Project Manager section in the Flow Navigator pane and choosing `Add or Create Constraints` from the list. The name of the constraints file is eightbit_adder.xdc. This file is provided on the course web page.

We are using switches and buttons on the board to control some of the inputs to the adder, and LEDs on the board to show the state of the switches. These are specified in the constraints file. There are 14 constraint groups, 2 for switches, 4 for buttons, and 5 for LEDs for interfacing the top level I/Os.

Open the top level, adder8_top.v file and find the code representing the inputs to the adder on lines 38-43.

As you can see, the first test will be for the bottom four bits. You will later change these lines to test the highest four bits.

4.2.1 Implementing a Design with Vivado Synthesizer

Click on `Generate Bitstream` under the Program and Debug section in the Flow Navigator pane to run the synthesis process. Vivado IDE will run the Synthesis and Implementation processes automatically.

4.2.2 Programming the FPGA with Hardware Manager

To program the Zynq's PL section you will use the Vivado Hardware Manager. After the bitstream is generated turn on the board and open the Hardware Manager either by clicking on the option that will pop-up at the end of bitstream generation or by clicking on `Open Hardware Manager` under the Program and Debug section in the Flow Navigator pane. In the Hardware Manager toolbar click on the `Open target` the choose `Auto Connect` from the list. In the same toolbar click on the `Program device` and choose xc7z020-1 from the list. Keep the settings unchanged in the window and press `Program`. Now you are ready to test your design in hardware.

```
## Switches
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {in1_ctrl[1]}]
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {in1_ctrl[2]}]

## Buttons
set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports {in2_ctrl[0]}]
set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports {in2_ctrl[1]}]
set_property -dict {PACKAGE_PIN L20 IOSTANDARD LVCMOS33} [get_ports {in2_ctrl[2]}]
set_property -dict {PACKAGE_PIN L19 IOSTANDARD LVCMOS33} [get_ports {in1_ctrl[0]}]
```

Figure 2: PYNQ Switches and LEDs in the Constraint File

4.2.3 Testing your Design in Hardware

The location of Switches and LEDs are based on the location map in Table 1. The out_ctrl bits in the table are the bits when testing the lower 4 bits of the adder.

Port	in1_ctrl[0]	in1_ctrl[1]	in1_ctrl[2]	in2_ctrl[0]	in2_ctrl[1]	in2_ctrl[2]
Device	BTN3	SW0	SW1	BTN0	BTN1	BTN2
LOC	L19	M20	M19	D19	D20	L20
Port	out_ctrl[0]	out_ctrl[1]	out_ctrl[2]	out_ctrl[3]		
Device	LD0	LD1	LD2	LD3		
LOC	R14	P14	N16	M14		

Table 1: LOC Attributes for Adder Inputs

Using the buttons and the switches, test that your adder works. The picture below shows which buttons correspond to which input. The purple box indicates bits 1-3 of input 1 (1 on the right). The orange box indicates bits 1-3 of input 2 (1 on the right). The blue shows the LEDs that will light up for the output (LD3-0) and the overflow flag (LD4). The overflow flag is only used when testing the upper 4 bits.

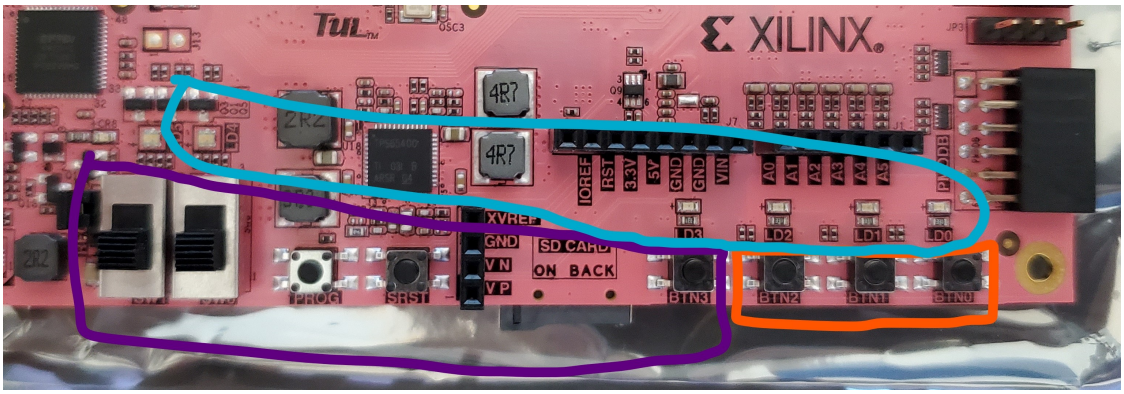

Figure 3: PYNQ Switches and LEDs

Once you have tested the lowest 4 bits of the adder, comment out the lines indicated in the top module and constraint file. Then uncomment the lines to test the higher bits. You will need to regenerate the bitstream to use the new constraint file. Once you have reuploaded to the board, test the higher bits as well as the overflow flag.

If your hardware works, demonstrate the tests for the TA. This is the second part of your lab grade.