| 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.

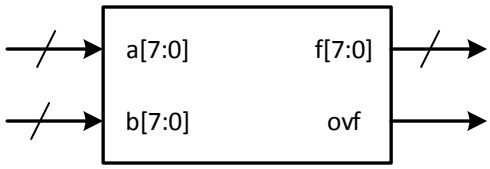


Figure 1: 8-bit Adder

*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 - Complete before coming to the lab

After reading Sections 1 and 2 above, answer the following questions. Include your responses with your final lab report submission.

1. Complete the table below for the expected input and output values for the 8 bit adder circuit you are describing in lab 1. Note that the syntax used for constant values is from Verilog; a,b and f are two's complement number.

| **a** | **b** | **f** | **ovf** |
|---|---|---|---|
| 8'd0 | 8'd0 | | |
| 8'd12 | 8'd34 | | |
| -8'd12 | -8'd34 | | |
| 8'd100 | -8'd50 | | |
| -8'd100 | 8'd50 | | |
| 8'd100 | 8'd100 | | |
| -8'd100 | -8'd100 | | |

   Note: `8'` indicates 8 bits in verilog and d indicates the value is decimal.

2. Write a Boolean or verilog equation for the overflow output `ovf` based on input values `a` and `b`. Your equation can also incorporate output `f`.

3. A good simulation testbench tests that every input and output bit can take the values zero and one. For the values in question 1 answer the following questions:

   (a) The adder has 16 bits of input and 9 bits of output. Do the values in question 1 taken together set every input bit to both one and zero and every output bit to both one and zero? Explain your answer. Identify which bits are not tested in this manner.

   (b) Add additional input ( `a` and `b`) values that test every bit in both the inputs and outputs so that every bit takes either a zero or a one value in your test cases.

========== END Prelab ==========

### 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 Vivado Simulator (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. If you right click any input and select "Radix" you can change the input and output data type to binary, signed decimal, or any desired type.**

**Include your waveform output with your final report submission.**

==========
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 and the included add-on board.

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 and outputs 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 and study it. Your `eightbit_adder` module has already been instantiated for you. You do not need to do anything. Just verify that the input and output port names match what's in your module.**

## 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 output of the adder. These are specified in the constraints file. There are 23 constraint groups, 10 for switches, 4 for buttons, and 9 for LEDs for interfacing the top level I/Os. You will need to have the included **add-on board** plugged in to the PYNQ board as shown in the picture below. No additional hardware setup needed.

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

### 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.**

```
##LEDs
## Add-on board LEDs
set_property -dict { PACKAGE_PIN B20   IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; # LDA
set_property -dict { PACKAGE_PIN W8    IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; # LDB
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; # LDC
set_property -dict { PACKAGE_PIN W6    IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; # LDD
set_property -dict { PACKAGE_PIN Y7    IOSTANDARD LVCMOS33 } [get_ports { led[4] }]; # LDE
set_property -dict { PACKAGE_PIN F20   IOSTANDARD LVCMOS33 } [get_ports { led[5] }]; # LDF

## PYNQ-Z2 board LEDs
set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { led[6] }]; # LD0
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { led[7] }]; # LD1
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { led[8] }]; # LD2 (not used)
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { wire ovf_ctrl }]; # LD3

##Switches
## Add-on board switches
set_property -dict { PACKAGE_PIN V6    IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; # SWA
set_property -dict { PACKAGE_PIN Y6    IOSTANDARD LVCMOS33 } [get_ports { sw[1] }]; # SWB
set_property -dict { PACKAGE_PIN B19   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; # SWC
set_property -dict { PACKAGE_PIN U7    IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; # SWD
set_property -dict { PACKAGE_PIN C20   IOSTANDARD LVCMOS33 } [get_ports { sw[4] }]; # SWE
set_property -dict { PACKAGE_PIN Y8    IOSTANDARD LVCMOS33 } [get_ports { sw[5] }]; # SWF
set_property -dict { PACKAGE_PIN A20   IOSTANDARD LVCMOS33 } [get_ports { sw[6] }]; # SWG
set_property -dict { PACKAGE_PIN W9    IOSTANDARD LVCMOS33 } [get_ports { sw[7] }]; # SWH


## PYNQ-Z2 board switches
set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS33 } [get_ports { sw[8] }]; # SW0
set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports { sw[9] }]; # SW1

## PYNQ-Z2 Buttons
set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }]; # BTN0
set_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; # BTN1
set_property -dict { PACKAGE_PIN L20   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; # BTN2
set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; # BTN3
```

Figure 2: PYNQ Switches, Buttons, and LEDs in the Constraint File

### 4.2.3 Testing your Design in Hardware

The two 8-bit inputs of the 8-bit adder are controlled as follows:
1. Input $a$ is controlled by the 8 switches on the add-on board ($sw[7:0]$).
2. The most significant 6 bits of the second input, b, are controlled by the two switches and four push buttons on the pynq board, while setting the remaining two bits to 0 as shown below:

| input_b | b[7] | b[6] | b[5] | b[4] | b[3] | b[2] | b[1] | b[0] |
|---|---|---|---|---|---|---|---|---|
| Device | SW1 | SW0 | BTN3 | BTN2 | BTN1 | BTN0 | 0 | 0 |
| LOC | M19 | M20 | L19 | L20 | D20 | D19 | – | – |

Table 1: LOC Attributes for Adder Input $b$

3. The output of the adder (`f[7:0]` and `ovf`) are displayed on the LEDs on the pynq board and add-on board as show below:

| output | *ovf* | f[7] | f[6] | f[5] | f[4] | f[3] | f[2] | f[1] | f[0] |
|---|---|---|---|---|---|---|---|---|---|
| Device | LD3 | LD1 | LD0 | LDF | LDE | LDD | LDC | LDB | LDA |
| LOC | M14 | P14 | R14 | F20 | Y7 | W6 | U8 | W8 | B20 |

Table 2: LOC Attributes for Adder Output $f$ and $ovf$

The *green* box indicates the switches used for input $a$ of the 8-bit adder. The *blue* box indicates the buttons and switches used for input $b$, and the *yellow* box indicates the LEDs used for the output $f$ of the adder. The overflow bit $ovf$ is displayed on the *white* LED.

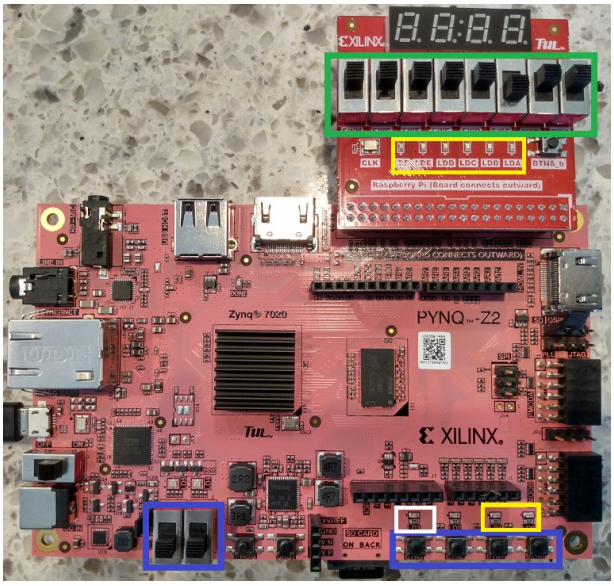


Figure 3: PYNQ Switches and LEDs

**Validate the correct operation of your design with various inputs.**

## 5 What to submit:

1. Demonstrate your working design to the TA.
2. A single PDF report (see format and rubric under "Reference Documents") containing a) Pre-lab responses including; screen capture of your Verilog module and test-bench, and the completed simulation waveforms, questions asked, etc. and b) summary of hardware implementation.