EECE2323 **Digital Logic Design Lab** **Lab 2**

Partial Arithmetic and Logic Unit

1 Objective

In this and the next lab you will design a combinatorial circuit known as an *Arithmetic and Logic Unit* (ALU) that performs certain calculations on binary inputs.

For this lab you will implement the partial arithmetic and logic unit responsible for addition, Bitwise NOT (inversion), bitwise AND, and bitwise OR. Then you will display the partial ALU operation results on the PYNQ board's LEDs.

In the next lab, you will create a complete ALU by adding the shift and branch operations to the partial ALU.

2 Overview

Arithmetic and Logic Units (ALUs) are an important part of every computer and calculator. They usually have two input operands for data and some inputs for control. Depending on the value of the control inputs, one of several arithmetic or logical operations is performed on the data. The ALU you will design has two 8-bit data inputs, one 8-bit data output, one overflow flag output and a 3-bit selector for choosing among the following operations: ADD(i), INV, AND(i) ($\cdot$), OR(i) ($|$), arithmetic shift right ($>>>$), logical shift left ($<<$), branch if equal (beq) and branch if not equal (bne).

Figure 1 shows the complete ALU interface.

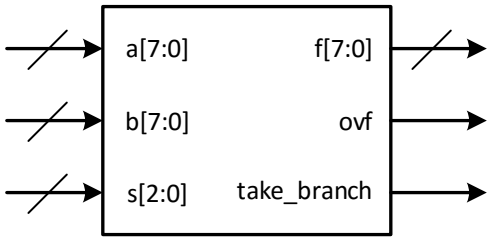

Figure 1: Complete ALU interface

a(7:0) and *b(7:0)* are the data inputs. Each input is eight bits wide and is interpreted as a signed number (2's complement). *s(2:0)* is the operation-select control input. The function performed can be changed by setting s to a different value, as shown in Table 1. 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. Note that logic operations cannot overflow, so the *ovf* output is always set to 0 in these operations. The *take_branch* output indicates the result of the branch operations. We will discuss this output in lab 3.

In all lab experiments, we will be working with signed numbers represented in 2's complement.

s[2:0]	f[7:0]	ovf	Description
0 0 0	$a + b$ (add)	overflow	a plus b
0 0 1	b (inv)	0	Bitwise inversion of b
0 1 0	$a \cdot b$ (and)	0	Bitwise AND of a and b
0 1 1	$a \mid b$ (or)	0	Bitwise OR of a and b
1 0 0	$a >>> 1$ (sra)	0	Arithmetic shift right
1 0 1	$a << b$ (sll)	0	logical shift left
1 1 0	0	0	Branch if Equal
1 1 1	0	0	Branch if not Equal

Table 1: ALU operations

The ALU represents the first piece of the *datapath* of our calculator. The datapath consists of circuits that store, move, and operate on data. For instance, an ALU performs arithmetic and logic operations. A *register file* acts as a memory and stores registers until we need them. The next several labs deal with designing circuits that will make up the complete datapath for our calculator.

The final part of this lab is to show the results of the partial ALU operations on the LEDs on the PYNQ board. Similar to Lab 1, you will connect the ports of your ALU to the top module of the parital ALU, then the driver will take care of displaying the result using the LEDs.

3 Partial Arithmetic and Logic Unit

The first part of this lab assignment is to create a *partial* ALU circuit using the Verilog HDL. Figure 2 shows the partial ALU interface. Note that it closely resembles the complete ALU interface (Figure 1), except that the operation-select control input, s, is 2-bits here, and there is no *take_branch* output.

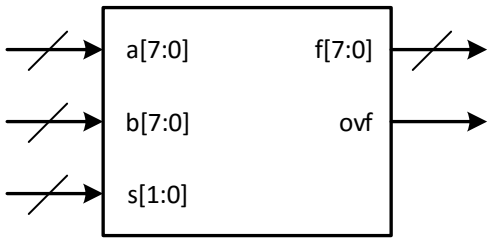

Figure 2: Partial ALU interface

The partial ALU operations and their descriptions are shown in Table 2. This is a subset of Table 1.

s[1:0]	f[7:0]	ovf	Description
0 0	$a + b$ (add)	overflow	a plus b
0 1	b (inv)	0	Bitwise inversion of b
1 0	$a \cdot b$ (and)	0	Bitwise AND of a and b
1 1	$a \mid b$ (or)	0	Bitwise OR of a and b

Table 2: Partial ALU operations

==========

3.1 Prelab - Complete before coming to the lab

Begin by reading Sections 1, 2, and 3 above. Then answer the following questions.

Start Vivado IDE. Create a new project called `lab2`. 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.

3.1.1 Prelab Part 1: Design the Partial ALU

Create a Verilog module called `eightbit_palu` which has two 8-bit inputs, a and b, and one 2-bit input, sel s. The outputs of this module are an 8-bit signal f, and a 1-bit signal ovf. The value of these outputs should change based on the *sel* signal value which determines the operation.

The partial ALU operations and their descriptions are shown in Table 2 above. Use the case statement in Verilog to describe your ALU. Be sure to provide a default value and be sure that the outputs are always defined no matter what the select inputs are.

3.1.2 Prelab Part 2: Create Test Vectors

Similar to what you did in prelab 1, create a set of test vectors that could test all of the bits in inputs, i.e. a, b and s, and outputs, i.e. f and ovf, of the partial ALU. **Create a table** to show your test vectors.

3.1.3 Prelab Part 3: Testbench and Simulation

Create a new testbench module called `palu_unit_tb`.
Enter the test vectors that you have chosen into the testbench, save the testbench, and run it in Vivado Simulator. Make sure that you set the `Simulation Run Time` property to an appropriate value before running your simulation.

Save and print your waveform and include it in your submission.

3.1.4 Prelab: What to Submit

Your should include, in the prelab section of your report, your Verilog code for your partial ALU, your Verilog code for your testbench, a table of your test vectors, and a screen shot showing your simulation working.

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

4 Displaying the Results on the LEDs

Connect the ports of your partial ALU to the top module. The driver will take care of displaying the operands, operation and results, similar to Lab 1.

4.1 Entering Your Design

Start Project Navigator and open lab2 if it is not already open.

The top level code file is called `palu_top.v`. You should get this file from the course Canvas site. Add the file to your project. Open the file for edit, you will see that the top module is called: `module palu_top`. *Instantiate your partial ALU driver* (`eightbit_palu`) *in this module if not done so already.* You should have three input ports: *a[7:0], b[7:0],* and *s[1:0]*; and output ports *f[7:0]* and *ovf* in your `eightbit_palu`.

4.2 Testing The Partial ALU 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. The `eightbit_palu.xdc` is provided on Canvas.

Among other constraints, there are 23 constraint groups, 10 for switches, 4 for buttons, and 9 for LEDs for interfacing the top level I/Os for having control over inputs to the ALU, and showing the output on the LEDs.

The two 8-bit inputs a and b of the partial ALU, and the 2-bit selector input s, are controlled as follows:
1. Input a is controlled by the 8 switches on the add-on board ($sw[7:0]$).
2. The least significant 4 bits of the second input, $b[3:0]$, are controlled four push buttons on the pynq board, while setting the remaining 4 most significant bits, $b[7:4]$ to $4'b0001$ as shown below.
3. The 2-bit selector input $s[1:0]$ are controlled by the two switches `SW0` and `SW1` on the Pynq board.

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

Table 3: LOC Attributes for Adder Input b

4. 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 4: LOC Attributes for Adder Output f and ovf

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_palu.xdc`.

Open the toplevel, `palu_top.v` file and find the code representing the inputs to the Partial ALU.

```
wire [7:0] palu_1st_input, palu_2nd_input;
wire [7:0] palu_output;
wire [1:0] alu_operation_sel;

assign palu_1st_input = sw[7:0]; // 8 add-on board switches.
assign palu_2nd_input = {4'b0001, btn[3:0]}; // constant values and pynq 4 btns.
assign alu_operation_sel = sw[9:8]; //assign the SW1-0 from the pynq board to the
ALU operation select bus.
```

Since input b is 8 bits while we have only 4 buttons on the PYNQ board and we also want to control the 2-bit select bus using switches, we can only have control over 4 bits of b at the same time using buttons.

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.

4.2.3 Testing your Design in Hardware

Test the inputs to the ALU and the different functions by assigning different sets of numbers to the inputs by pressing different buttons. Table 3 above, and the picture below, shows which switches and buttons on the PYNQ board are connected to the inputs of your design.

Using the switches and buttons, run the test vectors that you simulated. If you cannot use the same values, use new test vectors. Make sure that you toggle every output bit and that you verify the results on the LEDs.

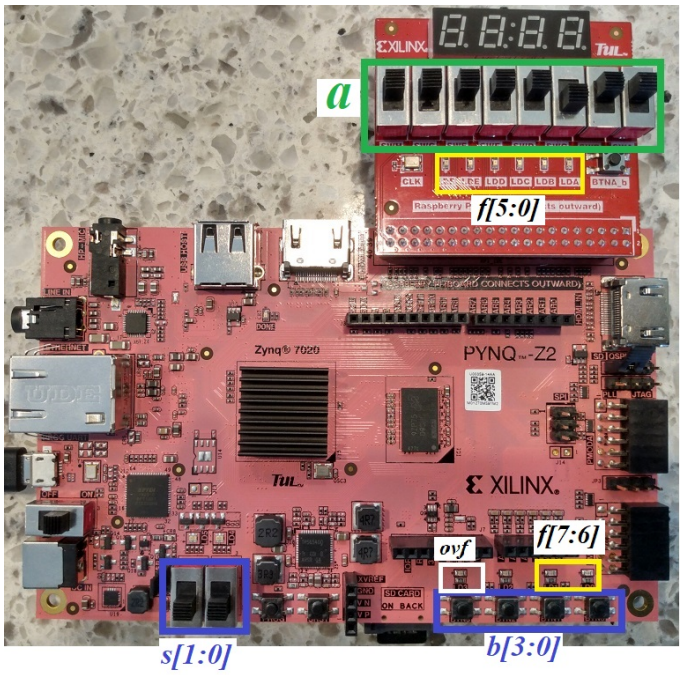

Figure 3: PYNQ Switches, Buttons and LEDs

5 What to submit on Canvas:

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/design implementation as outlined in the rubric.