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.

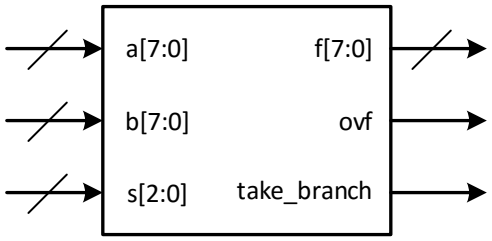

Figure 1: Complete ALU interface

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	$a << 1$ (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. As in Lab 1, you will test the lower half of the parital ALU first, followed by the upper half.

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.

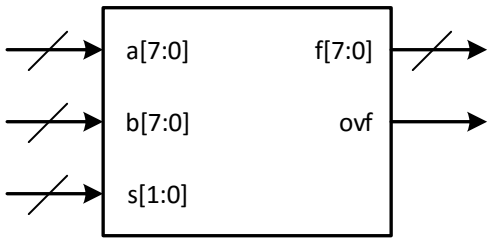

Figure 2: Partial ALU interface

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

3.1 Prelab

Download and complete the prelab assignment for this part from the course webpage. Remember that prelabs for all sections are due on Tuesdays.

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

3.2 Entering Your Design

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.

Create a new Verilog source called `eightbit_palu`.
You should have three input ports: *a[7:0]*, *b[7:0]*, and *s[1:0]*. Your output ports are *f[7:0]* and *ovf*.

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.

3.3 Simulating Your Design

Carefully determine what *test vectors* you will use to test all the functionality of your circuit.

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 XSIM. Make sure that you set the `Simulation Run Time` property to an appropriate value before running your simulation.

Save and print your waveform and show it to the TA when you submit this lab.

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. Open the file for edit, you will see that the top module is called: module palu_top. *Add your partial ALU driver to this module as sub modules.*

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.

Among other constraints, there are 10 constraint groups, 8 for switches and buttons and 4 for LEDs for interfacing the top level I/Os for having control over inputs to ALU, and showing the output on the LEDs.

The location of Switches and LEDs are based on the location map in Table 3.

Port	sel[0]	sel[1]	in1_ctrl[0]	in1_ctrl[1]	in2_ctrl[0]	in2_ctrl[1]
Device	SW0	SW1	BTN2	BTN3	BTN0	BTN1
LOC	M20	M19	L20	L19	D19	D20
Port	out_ctrl[0]	out_ctrl[1]	out_ctrl[2]	out_ctrl[3]	out_ctrl[4]	
Device	LD0	LD1	LD2	LD3	LD4	
LOC	R14	P14	N16	M14	L15	

Table 3: LOC Attributes for ALU Inputs

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 = 5'b00000, in1_ctrl[1], 1'b0, in1_ctrl[0]; assign palu_2nd_input = 5'b00001, in2_ctrl[1], 1'b0, in2_ctrl[0]; assign alu_operation_sel = sel; //assign the SW1-0 from the PYNQ board to the ALU operation select bus.
```

As you see, for both adder inputs 6 bits are set to a fixed pattern while other bits are connected to _ctrl signals. Because the inputs are 16 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 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 4 below shows which switches and buttons on the PYNQ board are connected to the inputs of your design.

For this design, switches 0 to 1 (the right-most two switches in the bank) are connected to the ALU's s input (operation select bus), buttons 2 to 3 are connected to bits 0 and 2 of the first ALU input (a), and buttons 0 to 1 are connected to bits 0 and 2 of the second ALU input (b). This will change when you test the upper bits of the partial ALU.

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.

SW1	SW0	BTN3	BTN2	BTN1	BTN0
$s[1]$	$s[0]$	$a[2]$	$a[0]$	$b[2]$	$b[0]$

Table 4: Hardware inputs

Once your hardware works, follow your TAs directions for submitting it.