Lab 1 - Logic

Lab Objectives:

1. Design and implement a circuit in hardware using integrated circuits.
2. Design and implement a circuit in hardware using VHDL.

Agenda:

1. Pre-lab.
2. In-class.
3. Assignment.

Collaboration:

Your instructor will inform you if you are allowed to work in pairs or not. For all assignments in this course, unless otherwise noted on the assignment, you may work with anyone. We expect all graded work, to include code, lab notebooks, and written reports, to be in your own work. Copying another person's work, with or without documentation, will result in NO academic credit. Furthermore, copying without attribution is dishonorable and will be dealt with as an honor code violation.

1 Purpose.

In this lab you will design a circuit that takes a month as a 4-bit binary input (e.g., January is equivalent to 0001 and December is equivalent to 1100). For each month that has 31 days, you should turn on the red LED (your output value, Y, is '1'). All other months and unused inputs should produce an output of '0'.

This circuit will first be implemented in hardware using integrated circuits and then implemented in hardware using VHDL.

2 Pre-lab. Due beginning of class on lesson 10 [25 Points]

1. **[5 Points]** Create a truth table that represents the conditions above.
2. **[5 Points]** Use a K-map to find the simplified equation for Y.
3. **[15]** Using the datasheets provided via Teams, draw **four** separate schematics that can implement your truth table. Make sure to reference data sheets for the components to ensure you have included every input they need to function (Hint: some of the chips must be enabled (Vcc or Gnd)). Pay attention to the order of the inputs to the MUX and encoder (which bit is the MSB)?
   (a) The first schematic will use an 8:1 MUX (74151) and inverter(s) (7404).
   (b) The second schematic will use a 4:16 "one-cold" (active low output) decoder (74154), inverter(s), and two-input OR gate(s) (7432). Using 2-input NANDs (7400) is also acceptable.
   (c) The third schematic will use only inverter(s), two-input OR gate(s), and three-input AND gate(s) (7411).
   (d) The fourth schematic will only use NAND gates with either two or four inputs.

☐ **[25 Points] Submit** all materials within the gradescope assignment, *Lab 1 - Prelab.*

3 In-class. [25 Points]

You will implement your first schematic (3.a 8:1 MUX (74151) and inverter(s) (7404)) in hardware using integrated circuits.

- Reference ICE1 if you need a refresher on how the breadboard functions and how to place your power converter. You can also reference the breadboard handout in the *Datasheets Library* to determine how the breadboard is connected.
- DO NOT leave floating inputs or control signals unconnected. All unused input pins need to be grounded or connected to Vcc.
- Double check all connections before turning on power.
- Power down your circuit before making any adjustments or changes!

☐ **[25 Points] Demo** your operational integrated circuit to an instructor.

4 Assignment. [50 Points]

You will implement your first schematic (3.a 8:1 MUX and inverter(s)) in hardware using VHDL.

4.1 Source Code

1. Download the VHDL templates from the 281 Team → Labs → Lab1
   (a) thirtyOneDayMonth.vhd
   (b) thirtyOneDayMonth_tb.vhd
   (c) Basys3Master_Lab1.xdc
2. Add your sources and constraints to Vivado. (Remember to leave "Copy sources into project" **unchecked**)
3. Edit the file headers as needed.

4.2 Entity (interface/ports)

Create your interface (ports) for the thirtyOneDayMonth according to figure 1. Figure 1 shows the thirtyOneDayMonth component entity interface in blue (think of this as the actual chip we used in the integrated circuit) and the architecture in purple (this is the circuitry inside the chip). The ports can be created in VHDL using the **std_logic** signal type (think of this as a single wire in our integrated circuit). For instance, i_A could be created in the **port** statement with the following:

```
i_A : in std_logic; -- one of four inputs
```

Remember, the "i_" follows the naming convention provided in the header that indicates the signal is an input.

1. **Use the above syntax to create the rest of the input ports within the entity.**

The output, "o_Y" is a single wire. This can be created in VHDL by using a **std_logic** signal type. For instance, "o_Y" can be created in the **port** statement with the following:

```
o_Y : out std_logic; -- output
```

2. **Use the above syntax to create the output port within the entity.**

You now have the entity portion of the thirtyOneDayMonth component complete. Next, you need to develop the internal logic, the architecture.

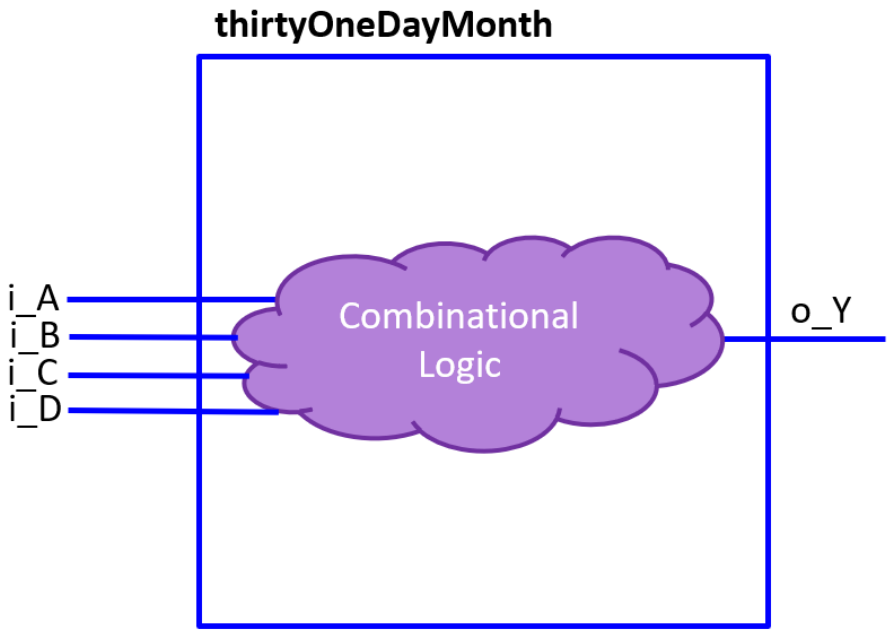

Figure 1: thirtyOneDayMonth component depicting the entity in blue and architecture in purple

4.3 Architecture

Create the logic to implement the internals of the thirtyOneDayMonth entity. The logic is *described* using VHDL, but the Basys3 board will *implement* the design using logic gates and other hardware components similar to what is inside the chip used during the integrated circuit portion of the lab.

As we saw in class, a multiplexer has 3 different signals: select, input, and output. We will connect three of our entity inputs to select bits and the fourth input bit to the input of the multiplexer. The entity output will be used as the multiplexer output. Since we already created the entity inputs and outputs, the only internal signal we need to implement is the select. This can be created using the **std_logic_vector** signal type as seen below. This is declared *before* **begin**:

```
signal w_sel : std_logic_vector(2 downto 0); -- MUX sel
```

1. **Replace the example code to include the multiplexer select signal within the architecture.**

In the CONCURRENT STATEMENTS section of the architecture you will need to connect your entity inputs to the appropriate wires of the `w_sel` vector. The individual wires of a vector can be accessed with indexes. To assign a value to the LSB of the vector you would do the following:

```
w_sel(0)<=i_D;  -- connect input D to the LSB of w_sel
```

2. **Repeat the signal assignment for `i_C` to index 1 and `i_B` to index 2.**
3. **Implement your prelab design using behavioral modeling as discussed in class.**

4.4 Testbench and Simulation

The testbench will be used to simulate your entity prior to implementing it on the Basys3 board. As you will learn, implementation takes a very long time and simulating your solution will save you a lot of headache.

The general concept of a testbench is to include the component you created and then test every input and observe if the output matches the truth table designed in the prelab.

As this is the first time using a testbench, a lot is filled out for you. Read through the provided template to see how it was set up. In the future you will have to write a lot of this code. There are three main sections that are complete to pay attention to:

- **Component:** declares the component of your top-level entity (this is the unit under test, UUT). This section will be an exact copy of your entity.
- **Additional components:** These are internal signals used to wire your simulated inputs and outputs to the entity. You will typically need a wire that corresponds to each port of your top-level entity.
- **Port Maps:** A port map wires your internal signals to the entity. There should be a port map for each entity you are testing.

A 4-bit test signal vector, `w_sw`, was created for you in the **Additional components** section to create your test cases. Instead of assigning each bit of the entity (A-D), you can assign all 4 inputs at once using hex. For example, you can use x"1" to represent the binary "0001" which would then assign a "0" to `i_A` - `i_C` and a "1" to `i_D`. Since we are simulating real hardware, we will have to create delays between each change in input. We can test an input and delay for 10 ns using the following:

```
w_sw <= x"0"; wait for 10 ns;
```

1. **Finish the test process. You will need to create 16 lines (2^4), similar to the one above, for each test case.**
2. **Edit the names of the `i_sw` bits as A, B, C, and D, to help match the results to your truth table. Click the ">" to expand the vector. Then right click on [3] and select rename. You can then change it to "i_A". Repeat for the other three signals.**
3. **Make sure the simulation results match your truth table.**

☐ **[15 Points] Submit** your waveform screenshot as an image in the gradescope assignment, *Lab1*.

4.5 Constraints File

1. **Open your Basys3Master_Lab1.xdc file.**

This time the constraints file is complete and you will not need to make changes to it. The `.xdc` file maps the signals in your entity to the physical pins on the board. As an example, `i_A` is mapped to pin V17 which is the 4th siwtch from the right. The other inputs are mapped similarly. The output, `o_Y` is mapped to pin U16 which corresponds to the LED on the far right. In the future, you will have to ensure the mapping is done correctly.

4.6 Implement

1. **Generate the bitstream (.bit) file and download it to your FPGA.**
2. **Verify that your design functions correctly.**

☐ **[25 Points] Demo** the operational VHDL circuit to an instructor.

☐ **[10 Points] Push** your `.vhd` and `.xdc` files using git.

5 Deliverables.

1. **[25 Points]** Prelab materials submitted to gradescope (Lab1 - Prelab).
2. **[25 Points]** Demo integrated circuit.
3. **[15 Points]** Simulation results waveform submitted to gradescope (Lab1).
4. **[25 Points]** Demo vhdl circuit.
5. **[10 Points]** Push your `.vhd` and `.xdc` files using git.