ECE 5727

Lab 3: ThunderBird Taillights

Due Wednesday, April 7
Total Points: 40

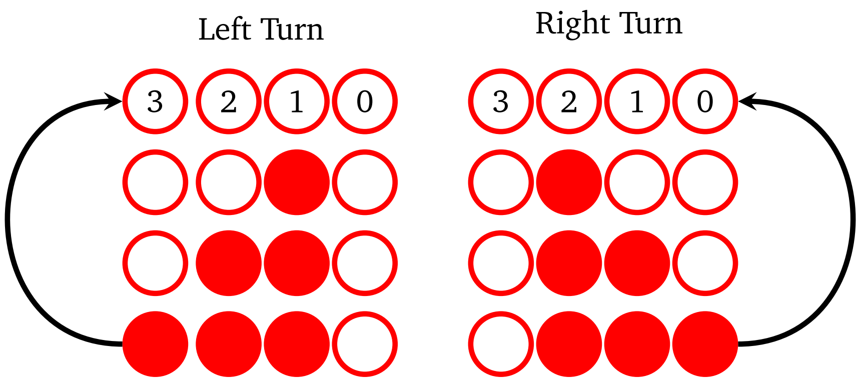
Objectives

In this lab you will:

1. Design and build a synchronous state machine.
2. Gain further familiarity with Xilinx Vivado and simulation
3. Learn how to load a design onto the Zybo dev-board

Lab Overview

The 1965-67 Ford Thunderbird had three left and three right tail lights which flashed on in sequence to indicate left and right turns. Unfortunately, the Zybo only has 4 LEDs, so we will be using the middle two for both right and left turns.



Your task is to design the FSM controller for these 4 lights.

To mimic the turn indicator for the car, which is normally a tri-state switch, we will be using two of the switches on the board. One will act as the **Left** indicator, and the other will indicate a **Right** turn. You can assume that both switches will never be on at the same time. We will also be using one of the buttons for input, to re-create the functionality of **Hazard** lights being turned on. To simplify the design, assume this button will never be pressed when a turn signal is enabled.

- When **Left** is asserted (switched on), the lights flash in the following pattern: all off; LED[1] on; LED[2:1] on; LED[3:1] on; and then the sequence repeats.
- When **Right** is asserted, the sequence is similar, using LEDS[2:0].
- If a switch is turned off in the middle of a turning sequence, the sequence should complete (not go straight to all-off). Thus, in the middle of a **Left** or **Right** sequence, there is no need to continue checking the input until the state machine returns to its idle/all-off state.
- If the **Hazard** button is pressed, the lights should enter hazard mode, and all four lights flash off and on in unison. The controller should remain in hazard mode with lights flashing until the button is pressed again.

Instructions

Setup

1. Install the Digilent Board files for the Zybo Z7-10. This can be done in two ways.
   (a) Follow the instructions provided by Digilent in Section 3: Installing Digilent Board Files:
   https://reference.digilentinc.com/vivado/installing-vivado/v2019.2
   (b) Open Vivado and in the TCL window run the following command:
   ```
   xhub::install [xhub::get_xitems digilentinc.com:xilinx_board_store:zybo-z7-10:1.0]
   ```
2. Download the Lab 3 skeleton project from either the course GitHub or Canvas and open it in Vivado.

Design

1. Design an FSM for this system. Draw out the state diagram. The typical design requires 8 states: one idle/off state, one for the hazards/all on, and three each for left and right turns. Note: if you use 9 states (2 for hazard) that is OK.
2. Within the *tbird.sv* code provided, implement your FSM.
   - An enum with potential state names is provided. Feel free to change them.

- You may use either a 1 or 2-process statement machine. The shell code contains the start of a 1-process machine. I have also included commented out code to use for a 2-process machine.
- Use the provided leftSwitch and rightSwitch signals for your switch inputs to the FSM. Notice that the switches are being double-flop synchronized.
- For the hazard button, you can use either the hazardPushed or hazardEnable signals. The former will generate a single pulse when the button is first pressed. The latter indicates the button has been pressed once to enter hazard mode.
- Take a look at the debouncer module code if you'd like. This module has a SIM_ONLY parameter. This is often added to modules to allow for shortcut code to be used during simulation. In this case, its to reduce the amount of clock cycles required to debounce the hazard button so simulation doesn't take so long to run.
- There is a clocking IP Core being used. Double clicking it (clk_gen : clk_gen.xci) will open the IP Core configuration GUI.

Test & Implement

1. *tbird_tb.v* is the testbench file for the project. Run a behavioral simulation. As you can see, the testbench currently is set up to only test the left turn signal and a single press of the hazard button. Modify it to fully test your design. *Submit a screenshot of your simulation waveform showing the design working correctly*.

2. Take a look at the constraints file - *tbird.xdc*. This file informs Vivado which package pins to connect to the input/output signals of the top-level module. Each pin is described by a line such as this:

   ```
   set_property -dict {PACKAGE_PIN W13 IOSTANDARD LVCMOS33}[get_ports {sw[0]}];
   ```

   - Above, we are specifying that we are referring to pin W13 (row W, column 13) on the FPGA package, and that it should be internally configured to use the CMOS 3.3V standard. The `get_ports` command tells Vivado the name of the module port we want connected to the pin - sw[0] in this example. As this is the right-most switch I have already renamed this to `RIGHT` on line 13.
   - I have also already picked which switch to use for `LEFT` and which button to use for `HAZARD_BTN`. You can pick other ones if you'd prefer. Just uncomment the appropriate line and update the port name.
   - The `LEDS` and `SYS_CLK` are taken care of. Leave those alone.

3. Run implementation and then open the implemented design. It should open with the `Device` tab selected, where you can now see all the logic placed into the FPGA fabric.

4. Take a look at the timing for the design. You can see this summarized in the Project Summary. From the `Timing` tab you have interactive access to the full timing report. Find the path with WNS and double click it to open its timing analysis. The top of the analysis is a summary, including a `Data Path` field which will tell you how much the combinational logic in the path is contributing compared to the routing of the signal between all those logical elements on the chip.

Taking the Lab Further

If you'd like to do more with the lab here are some ideas to try out:

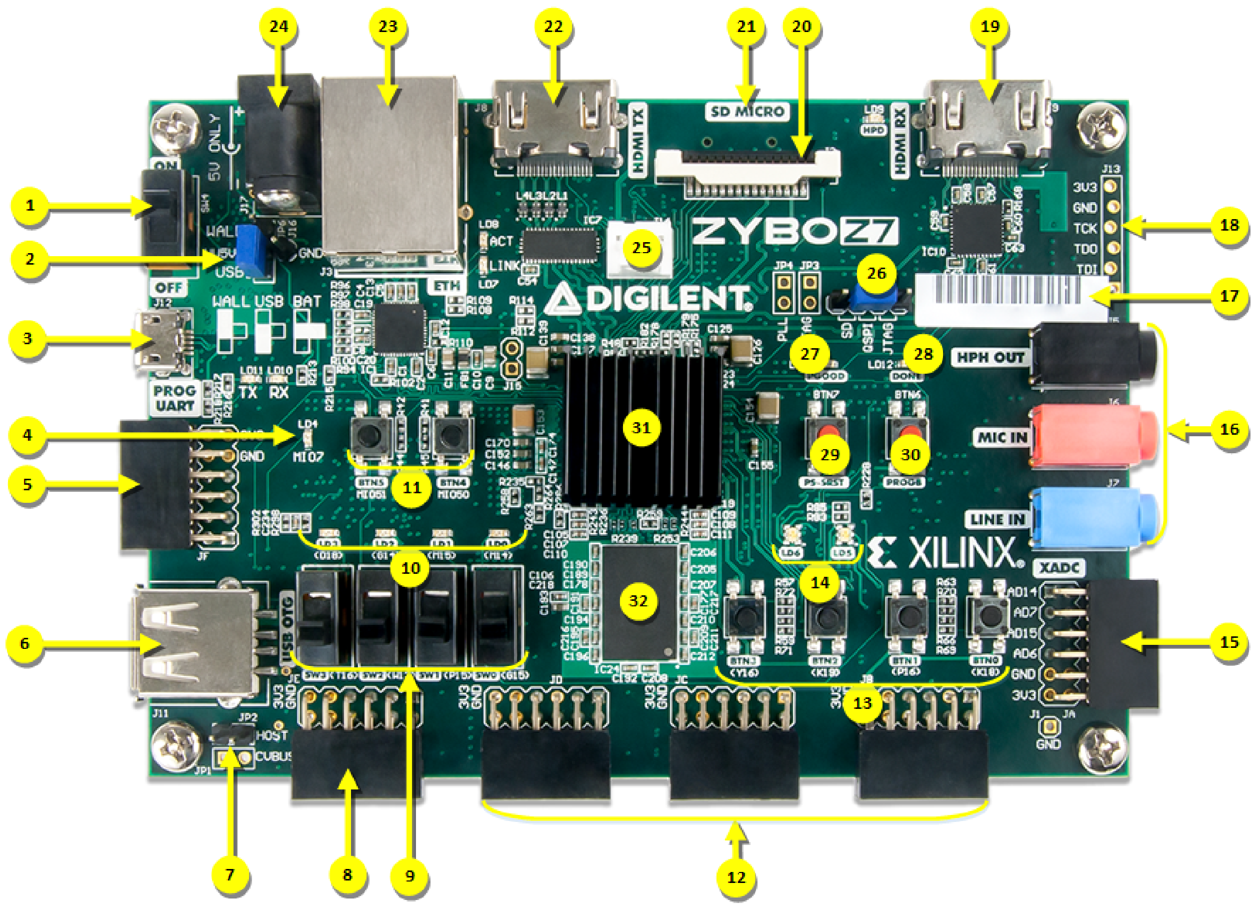
- Make the FSM more robust: handle the hazard button being pressed at any time (including during a turn); make it so that whichever turn was in progress when the second switch is flipped on remains running until one of the switches is flipped off.
- Make the FSM a 2-process Mealy machine if you didn't already do it that way. If you did, then make a 2-process Moore machine. To make these functionally the same, do you end up with any benefit of choosing one style over the other? What about if you make 1-process versions?
- Anything else you can think of!

Running on the Zybo Z7

If you have access to one of the Zybo dev boards and would like to run the design on it, you can follow these instructions to do so.

1. Once implementation is complete, `Generate Bitstream` from the Flow Navigator pane
2. Connect your computer to the Zybo via a micro-USB cable.
3. Make sure the jumpers are positioned appropriately, and then turn on the power switch.
   - The jumper (#2) next to the power switch should be connecting the lower two pins vertically to select the micro-USB as the power source.
   - The jumper (#26) above the two red buttons should be positioned to the far right to select `JTAG` as the programming mode.
4. From the Flow Navigator pane select **Open Hardware Manager -> Open Target -> Auto Connect**
5. Finally, select `Program Device` from the green bar near the top of Vivado. This will bring up an option to specify a bitstream and debug file. We don't have a debug file, so its fine to leave that blank. The bitstream should already be pointed to. Click OK to program the board!

For a (better) set of instructions with pictures take a look at the section on using the Hardware Manager at https://reference.digilentinc.com/reference/programmable-logic/guides/getting-started-with-vivado



Lab Submission

1. (*10 points*) Diagram of your FSM - take a picture or scan it if you drew it by hand.
2. (*each 10 points*) Completed *tbird.sv* and *tbird_tb.sv* files.
3. (*5 points*) Screenshot of testbench waveform
4. (*5 points*) What was the path with the WNS, and what was the value? Is the logic or the routing causing most of the delay?