Course Name: EMBEDDED SYSTEMS I / III

Course Number and Section: **14:332:493:03**

Year: Spring 2021

Assignment: Final Project

Lab Instructor: Philip Southard

Student Name and RUID: Jonathan Ely, 204006108

Date Submitted: May 5, 2021

GitHub Link:

https://github.com/JonathanEly/EmbeddedSystems_FinalProject_Sp2021

<u>Purpose/Objective:</u>

To turn the Zybo in a digital level using the GYRO and ENC PMODs.

Using the position outputs from the GYRO, we can store those values into a shift register and take the average of each signal to produce a stabilized position signal. We can then compare those stabilized signals to an internal register to indicate whether or not the Zybo is level. Then, by taking the difference from that comparison, we attribute a new number which controls the brightness of a corresponding LED. The LEDs indicate tilting in the positive and negative x and y axes; the further the Zybo tilts in one direction, the brighter the corresponding LED lights. At the zeroed GYRO position, the LEDs should all be off. Using the rotary encoder module, we can turn it to adjust the maximum brightness shown by the LEDs. Pressing the button on the encoder will update the internal registers holding the "zeroing" comparison positions with the current GYRO positions. By flipping the switch on the encoder, we change the button's behavior to reset the internal registers back to zero so that the GYRO is reset to be level.

<u>Theory of Operation:</u>

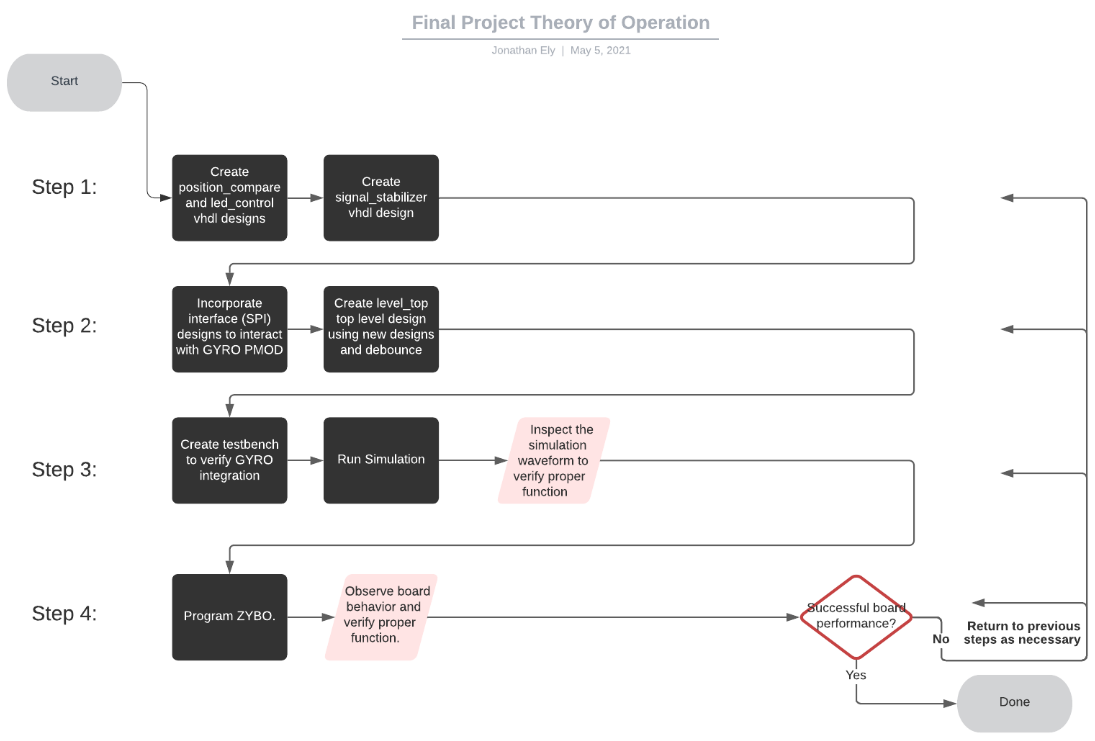
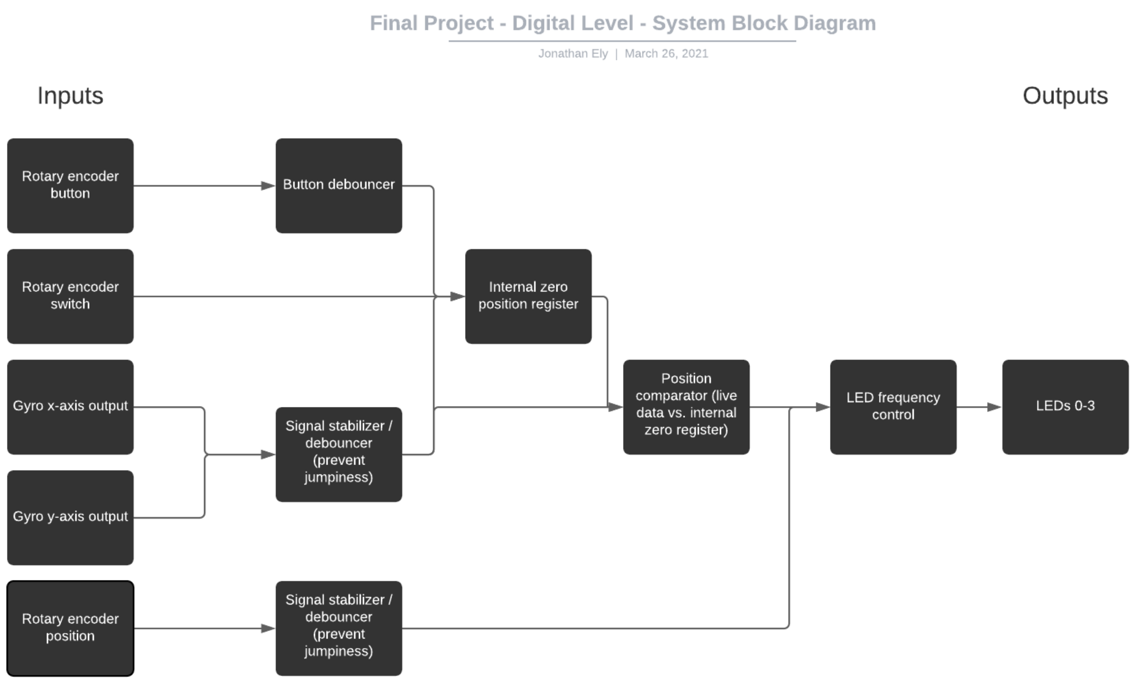

Final Project Theory of Operation
Jonathan Ely | May 5, 2021
Start
Step 1:
Create position_compare and led_control vhdl designs
Create signal_stabilizer vhdl design
Step 2:
Incorporate interface (SPI) designs to interact with GYRO PMOD
Create level_top top level design using new designs and debounce
Step 3:
Create testbench to verify GYRO integration
Run Simulation
Inspect the simulation waveform to verify proper function
Step 4:
Program ZYBO.
Observe board behavior and verify proper function.
Successful board performance?
No
Return to previous steps as necessary
Yes
Done

Figure 1: Theory of Operation.

<u>System Block Diagram:</u>

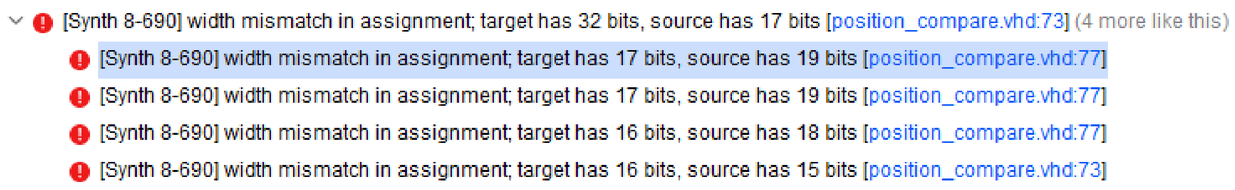


Final Project - Digital Level - System Block Diagram
Jonathan Ely | March 26, 2021
Inputs
Outputs
Rotary encoder button
Button debouncer
Rotary encoder switch
Internal zero position register
Gyro x-axis output
Signal stabilizer / debouncer (prevent jumpiness)
Gyro y-axis output
Position comparator (live data vs. internal zero register)
LED frequency control
LEDs 0-3
Rotary encoder position
Signal stabilizer / debouncer (prevent jumpiness)

Figure 2: System Block Diagram.

Simulation Waveforms:

I do not have simulation waveforms or other schematics because I encountered an issue with inputs and outputs between designs where two 16 bit signals are somehow generating width mismatch errors.



```
[Synth 8-690] width mismatch in assignment; target has 32 bits, source has 17 bits [position_compare.vhd:73] (4 more like this)
    [Synth 8-690] width mismatch in assignment; target has 17 bits, source has 19 bits [position_compare.vhd:77]
    [Synth 8-690] width mismatch in assignment; target has 17 bits, source has 19 bits [position_compare.vhd:77]
    [Synth 8-690] width mismatch in assignment; target has 16 bits, source has 18 bits [position_compare.vhd:77]
    [Synth 8-690] width mismatch in assignment; target has 16 bits, source has 15 bits [position_compare.vhd:73]
```

All of these signals should be of the same bit length. The error persisted even after I changed the inputs and outputs to constant values of the same bit length, so it is not clear to me where the error is.

Vivado Schematics:

a) Vivado Elaboration Schematic

b) Vivado Synthesis Schematic

c) Post- Synthesis Utilization Table

d) On-Chip Power Graphs

The constraints file needs to include the regular clock pin as well as the pin assignments for the four board LEDs, and the GYRO and ENC PMODs. The four LEDs are wired as outputs as we have in labs throughout the semester. The GYRO PMOD has 12 pins and can be used in either SPI or I2C busses. Of the 12 pins, two are GND, two are VCC, and two have no connection. The remaining 6 pins are Master Out/Slave In, Master In/Slave Out, Serial Clock, Chip Select, and two interrupts.

The ENC PMOD has 6 pins and uses the GPIO bus. There is a pin for GND, VCC, the on-board switch, the on-board button, and A and B pins which are used for the rotary encoder's position.

Answers to Additional Questions and Extra Credit:

The final project does not outline specific additional questions or extra credit opportunities.

Conclusion:

I was not able to complete this project. My time spent on this project consisted partly of research into how to read from the GYRO PMOD on the Zybo board using SPI. The available documentation through the Digilent website and the Vivado Xilinx Help pages were helpful, but I ran into the same issue as my attempt at Lab 6 where I could not understand exactly how the SPI interfacing should work and I ran out of time.

The two main issues holding up progress were the SPI interface for the GYRO PMOD and an internal error where a signal width mismatch was occurring. As mentioned earlier in this report, there was an unexpected error involving signal width mismatches.

Follow Up:

Conceptually, the project is relatively simple, but completing it required time that was lost to me. If more time were available to me during the end of this semester, I would have been able to dedicate more time to thoroughly understanding the SPI concepts required to connect the GYRO PMOD.

I understand how the designs need to be implemented, but I got snagged on creating this implementation. I spent a lot of time trying to understand how to use the SPI interface to get the axis data from the GYRO PMOD, but was unable to get it operational. From the output of the gyro, the x- and y-axis position data should be passed through a signal stabilizer to ensure the processed signals would be "smoother" and would prevent as much flickering in the LEDs on the output end of the project. The stabilizer is made using a simple shift register and then using an internal register to take the average of the signals and output said average.

The button input from the encoder PMOD also required a debounce module, which was reused from previous labs. The switch input from the encoder PMOD was simple to implement because that PMOD uses a GPIO bus, so direct binary signals can be easily used.

The encoder position input is determined using the A and B binary signals that come from the module. Those signals do not need to be stabilized, but are passed through a Finite State Machine. The encoder works such that when it is turned one click to one side, the A and B signals flip off and on quickly. It is sent in a pattern of A off, B off, both on. Alternatively, if the encoder is turned in the opposite direction, the signals are sent B off, A off, both on. The signals are on by default if the rotary encoder is stationary. So by using a FSM, we can track these changes in signals to check which pattern is being sent and then increment or decrement an internal register appropriately. We can create an internal register of arbitrary size/length and adjust maximum and minimum values in the FSM. We can also make it so that if you continue to increment or decrement past the maximum or minimum values, respectively, the internal register will overflow or underflow. For the purpose of this design, I did not enable this functionality because the encoder controls the maximum LED brightness. It would be frustrating in operation to have the internal register overflow when trying to turn the dial all the way in one direction to achieve the maximum or minimum brightness easily.

Using the stabilized outputs from the gyro, we can set an internal register in the design to hold said axis positions. By comparing the current gyro data to the stored data, we can change the LED output indicators to indicate how off-level the gyro currently is from the stored orientation. Similarly, we can set these internal registers to zero in order to reset the level to interpret axis tilt compared to zero.

This functionality is implemented using the switch and button on the encoder. If the button is pressed and the switch is off, the internal register stores the current value of the gyro output. If the button is pressed and the switch is on, the internal register resets to zero. The internal register is used in processing later to compare to the current gyro data.

In processing, when the gyro data is being compared to the internal register storing axis position values, a frequency value is associated with various differences between these two values. The greater the difference, the greater the frequency. This frequency is then used in a clock-divider-type process, similar to our first Blinker Lab. This controls the frequency and duty cycle of the LED. With greater duty cycle, the LED appears brighter. The frequency is increased if the difference between the current data and stored data is greater, so more tilting results in brighter LED displays.

The positions from the encoder change the maximum brightness the LEDs are allowed to display. The gyro positions control when the LED turns on and the encoder position controls when the LED turns off. With a lower internal counter threshold, the LEDs turn off sooner, therefore resulting in a lower duty cycle percentage, and therefore dimmer LEDs. With higher internal counter thresholds, the LEDS turn off later, therefore resulting in a proportionately higher duty cycle, and therefore brighter LEDs (up to a point where the change in brightness becomes imperceivable).

The frequency controls of the LEDs output directly to the LEDs, resulting in a visual display of how far tilted the gyro is in the x- and y-axis directions. Implementation can also modified to include the z-axis or exchange the axes.

While I was able to get most of the design completed, I did not have the time to finish implementing the SPI interfacing, which is arguably the most important part of the project because it allows us to use gyro data. With that, the unexpected signal width mismatching, and other complications, I was not able to complete this project’s implementation.