ECE 385

Fall 2023

Lab 6: SOC with Microblaze in SystemVerilog

Alex Yu, Corey Yu
GL/ 13:45-14:00
Gene Lee

1. **Introduction**
    - **Summarize the basic functionality of the Microblaze processor running on the Spartan 7 FPGA.**
      The microblaze is a soft IP block that provides the functionality of a RISC harvard architecture processor. An IP block is akin to a software library (except in SystemVerilog), and "soft" refers to the fact the IP is implemented on the fpga fabric rather than having dedicated silicon for it. The microblaze processor can be customized with different configurations, supporting memory, interrupt, different timing frequency, and it is able to run software like C on Vitis.
    - **Briefly summarize the operation of the overall Week 2 design**
      The 6.2 design takes input from a USB keyboard through the MAX3421E chip. This chip communicates with the microblaze processor through SPI protocol through the quad SPI IP in the block design. A provided VGA controller in Vivado (SystemVerilog, not C) generates horizontal and vertical sync signals and drawX and drawY. The keycode data is fed into the provided "ball.sv" file which takes the keycode and vsync as a frame_clk to generate the motion for an object and check for edge collision, which is fed into a color mapper which actually assigns the pixels color to match the motion. The color mapper generates R, G, B signals which the VGA to HDMI takes to generate the corresponding HDMI signals.
2. **Written Description and Diagrams of Microblaze System**
    - **Module descriptions:**
        i. **Describe in words the hardware (e.g., every .SV module) including the provided .SV modules.**
            1. Mb_usb_hdmi_top.sv
                a. input logic Clk,
                b. input logic reset_rtl_0,
                c. input logic [0:0] gpio_usb_int_tri_i,
                d. input logic usb_spi_miso,
                e. input logic uart_rtl_0_rxd,
                f. output logic gpio_usb_rst_tri_o,
                g. output logic usb_spi_mosi,
                h. output logic usb_spi_sclk,
                i. output logic usb_spi_ss,
                j. output logic uart_rtl_0_txd,
                k. output logic hdmi_tmds_clk_n,
                l. output logic hdmi_tmds_clk_p,

    m. output logic [2:0]hdmi_tmds_data_n,
    n. output logic [2:0]hdmi_tmds_data_p,
    o. output logic [7:0] hex_segA,
    p. output logic [3:0] hex_gridA,
    q. output logic [7:0] hex_segB,
    r. output logic [3:0] hex_gridB
    s. Description: This is the top level file for the whole project. It calls the mb_block, hex_driver, clk_wizard, vga_controller, hdmi_tx_0 IP, the ball module, and the color mapper. It basically connects the mb_blocks with all the other modules that are needed to print the ball onto the screen.
    t. Purpose: The mb_block itself can’t do anything. It needs to be instantiated by the top level and connect to other modules.

2. Vga_controller.sv
    a. input logic pixel_clk, reset
    b. output logic hs, vs, active_nblank, sync
    c. output logic [9:0] drawX, drawY
    d. Description: This module controls the drawing logic of the VGA. THe logic includes what it should do when the reset is pressed, and how the VGA does the vertical sync and the horizontal sync.It takes care of the boundaries as well.
    e. Purpose: We need this module to make sure the VGA will print out correctly. If the sync or reset signal has wrong logic, the screen will not print out cracked screen.
3. Ball.sv
    a. input logic Reset, frame_clk,
    b. input logic [7:0] keycode,
    c. output logic [9:0] BallX, BallY, BallS
    d. Description: This module defines the properties of the ball, including the ball size, ball motion, and the ball logic to move on the screen. This module also takes in the input from the keyboard and will move the ball accordingly.
    e. Purpose: We need this module to control the ball properly. Without this sv file, we will not see there is a ball on the screen.

4. Color_Mapper.sv
    a. input logic [9:0] BallX, BallY, DrawX, DrawY, Ball_size,
    b. output logic [3:0] Red, Green, Blue
    c. Description: The color mapper has two main functions inside, one is basically drawing the circle on the screen , the other one is drawing the gradient background with the given RGB values.
    d. Purpose: While the other module controls the logic of the ball, the color_mapper actually puts the color on the screen by outputting the RGB values. We need it to print on the screen.
5. Hex.sv
    a. Inputs: Clk, reset, [3:0] in[4], [3:0] nibble
    b. Outputs: [7:0] hex_seg, [3:0] hex_grid, [7:0] hex
    c. Description: The hex driver takes in the data in the binary format and outputs the number to hex that can be shown on the LED segment display.
    d. Purpose: The LED segment display doesn't take in numbers in binary so we need this module to translate it to display.

ii. **Additionally for this lab, you created many modules which are encapsulated in a block design. Describe every component in the block design, some of them like the AXI interconnect, you will need to do some research. You can separately describe the 'core' components common to week 1 and 2 first, and then describe the additional components for week 2 separately.**

1. **MicroBlaze:** This is the soft IP block that provides the functionality of a RISC harvard architecture processor. As described in the introduction, this is the processor that responds and processes all commands.
2. **Clocking Wizard:** This is the block that takes in the clock signal input and input the signal to all the components connected, making sure that they are synchronized. It also takes in the 100MHz and outputs 2 clock signals 25MHz and 125MHz for the HDMI.
3. **AXI Interconnect:** The AXI interconnect module can connect one or more AXI masters to one or more AXI slaves. In this case, we have multiple slaves to connect to, so we need this module.

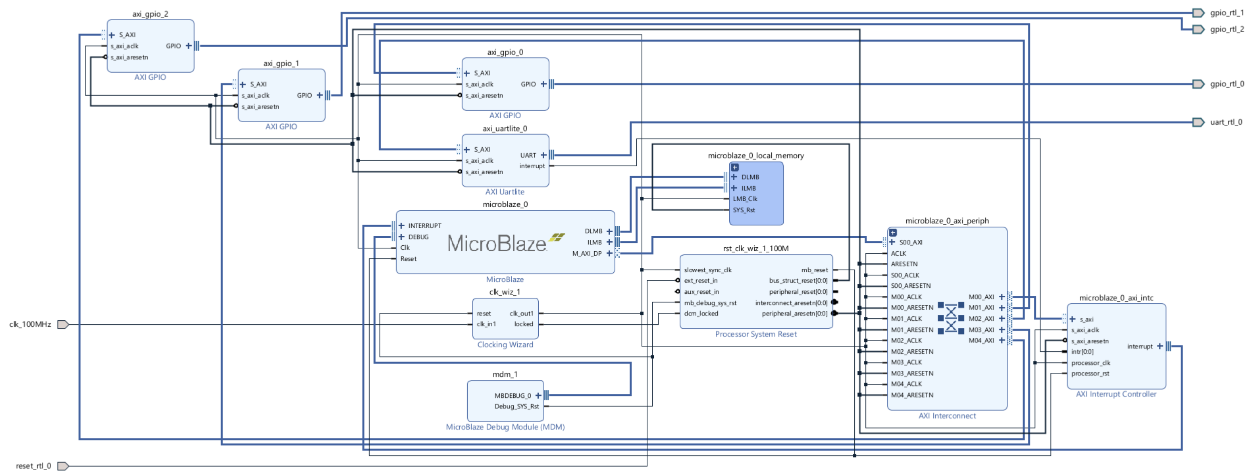
4. **AXI Quad SPI:** This module connects the AXI interface to the slaves that support quad SPI protocol.
5. **AXI Uartlite:** This module connects the processor and provides it with asynchronous serial data transfer.
6. **AXI Interrupt Controller:** This module concentrates multiple interrupt signals into one and sends it in the processor.
7. **Processor System Reset:** This module allows the user to set the parameter to enable/disable the feature. Like set it as active high reset or active low.
8. **Concat:** This block concatenates all the interrupt signals and feeds them in the interrupt controller.
9. **MicroBlaze Debug Module:** It is a core that enables JTAG-based debugging of the microblaze.
10. **MicroBlaze Local Memory:** This is the memory block of the MicroBlaze.
11. **AXI GPIO:** This is the block that connects to the GPIOs like USB, it is connected to the output pin.
12. **AXI Timer:** This is a timer/counter interface to the AXI-4 Lite that provides the signal of AXI_reset and AXI_clk.

- **Describe in Lab 6.1 how the I/O works.**
  IO is handled through GPIO IP blocks through memory-mapping. For each component (LEDs, switches), a GPIO IP was created of the appropriate width – e.g. in the tutorial only one LED was blinked so it had a width of one, whereas the GPIO block for the switches had a width of 16. Each GPIO corresponds to a struct, which is memory mapped by assigning a block of memory in microblaze RAM that corresponds to it.
  In the C portion volatile pointers are declared that correspond to the memory addresses of the GPIO blocks that are viewable in "Address Editor".
- **Describe in words how the MicroBlaze interacts with both the MAX3421E USB chip and the ball motion components.**
  The microblaze communicates to the MAX3241E via SPI. The quad SPI block uses 4 signals: save select, MOSI, MISO, and a Clk. The MAX3421E chip reads the keycode pressed and sends back the data through the MISO line. The keycode is sent through the AXI4 lite bus to the ball.sv module which implements the motion of the ball.
- **Describe in detail the VGA operation, and how your Ball, Color Mapper, and the VGA controller modules interact.**
  The VGA implements two nested counters, a horizontal one and vertical one, and outputs hsync, vsync, DrawX and DrawY. When the horizontal

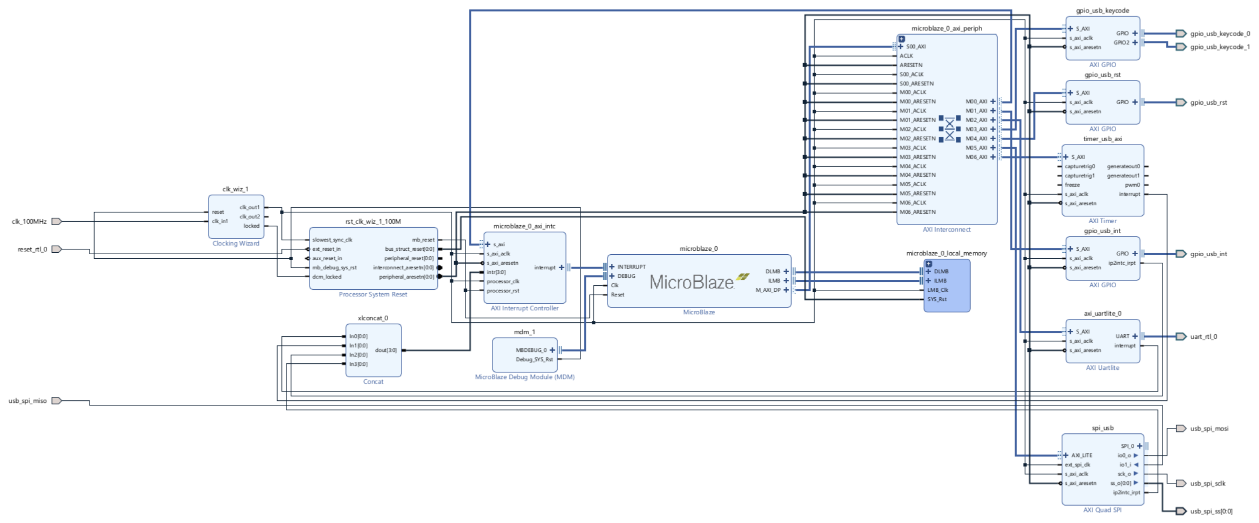
counter hits the maximum x hsync is set high, and the vertical counter increments. When the vertical counter hits the maximum y vsync is set to high. DrawX and DrawY are the counters current values. DrawX and DrawY are fed into the color mapper that implements the coloring of the moving ball.

- **Describe the VGA-HDMI IP, how does HDMI differ from VGA, how are they similar?**
  The VGA to HDMI IP takes in the RGB signals generated by the color mapper, horizontal and vertical syncs generated by VGA and a 25Mhz and 125 Mhz clock generated by the clocking wizard. It outputs the TMDS signals clk_p, clk_n, data_p, data_n which are differential signals as HDMI uses TMDS (transition minimized differential signaling).
  HDMI is a digital protocol that uses differential signaling to minimize noise and two clocks whereas VGA is an older analog protocol. Both display video output, though VGA is limited in resolution relative to HDMI.

3. **Top Level Block Diagram**
   - **a. This diagram should represent the placement of all your modules in the top level.Please only include the top-level diagram and not the RTL view of every module.**
   - **Lab6.1**



   - **Lab6.2**



4. **Describe in words the software component of the lab.**
   - **One of the INMB questions asks about the blinker code, but you must also describe your accumulator.**
     - For the accumulator, we use if conditions and a flag to achieve the accumulator. The if statement is used to determine if the button is pressed or not, and there is a flag to make sure every time the button is pressed, it will only be recorded once. Moreover, in our code, we also put in the overflow condition. If the overflow happens, the program will print out overflow on the serial terminal.
   - **Written description of the SPI protocol and how it operates in the context of the MAX3421E.**
     - For the SPI signal, there are 4 main signals: SCLK, MOSI (Master Out Slave In), MISO (Master In Slave Out), (SS)(Slave Select/Chip Select). The SCLK is the CLK signal of the SPI communication. It will constantly fluctuate between 1 and 0. For the other three signals, the slave select will go high when the SPI is going to start transmitting/receiving data. The first byte of data from the MOSI is always going to be the command byte, with data like the registers, the direction(read or write) and ACKSTAT. If it is writing(determined by the direction bit), the next byte of MOSI will contain the data it wants to write. And MISO will just be don't care. However, if it is read, the MISO will send the data bit from the second byte.
   - **Describe the purpose of each function you filled in in the C code (you do not need to describe the functions you did not modify).**
     - There are four functions we fill in in the C code:
     - void MAXreg_wr(BYTE reg, BYTE val)

        - This is a single host register write. It writes the register+val to the SPI, and it will get a status back to know if the write is successful.
    - BYTE* MAXbytes_wr(BYTE reg, BYTE nbytes, BYTE* data)
        - This is a multiple byte write. It writes the register + multiple values to the SPI, and it will return the pointer to the place that is last written.
    - BYTE MAXreg_rd(BYTE reg)
        - This is a single host read. It creates an array and stores the data read from the SPI to the array. It will also give an error status message if the read failed.
    - BYTE* MAXbytes_rd(BYTE reg, BYTE nbytes, BYTE* data)
        - This is a multiple-bytes register read that creates an arbitrary size of array, and it will store the result from SPI (including the register value as the first value) to the array. It will also give an error status message if the read failed.

5. **Answers to all INMB (italicized) & Post lab questions**
    a. **Select the "Microcontroller" Preset and then modify the "Local Memory" to 32KB (see Figure 2). You should do some research and figure out what are some primary differences between the various presets which are available.**
    The three presets are microcontroller, real-time and application. Microcontroller has the least features and is optimized to minimize area on the FPGA, it has no cache. Real time is optimized for performance and has small caches, it is used for tasks that require strict timing. Application is optimized for performance with large caches and an FPU (floating point unit), this is used in high demand tasks that are not latency sensitive.
    b. **Make sure you do a little bit of research regarding each component and can summarize their functionality in a sentence for your report. Note the bus connections coming from the MicroBlaze; is it a Von Neumann, "pure Harvard", or "modified Harvard" machine and why?**
    It is a modified Harvard architecture. The memory for instructions and data are the same physical place (as opposed to pure harvard) however there are separate lines for memory and data instructions (as opposed to Von Neuman).
    c. **What does the "asynchronous" in UART refer to regarding the data transmission method? What are some advantages and disadvantages of an asynchronous protocol vs. a synchronous protocol?**
    Asynchronous refers to the fact the protocol functions without a clock. This

means not all of the components are connected to the clock. The advantages of asynchronous protocols is that it is simpler to set-up; however they are often slower than synchronous protocols e.g. SPI. For the synchronous protocol, all the components have to be connected to the same clock, and they will change at the same time. (Same clock cycle.)

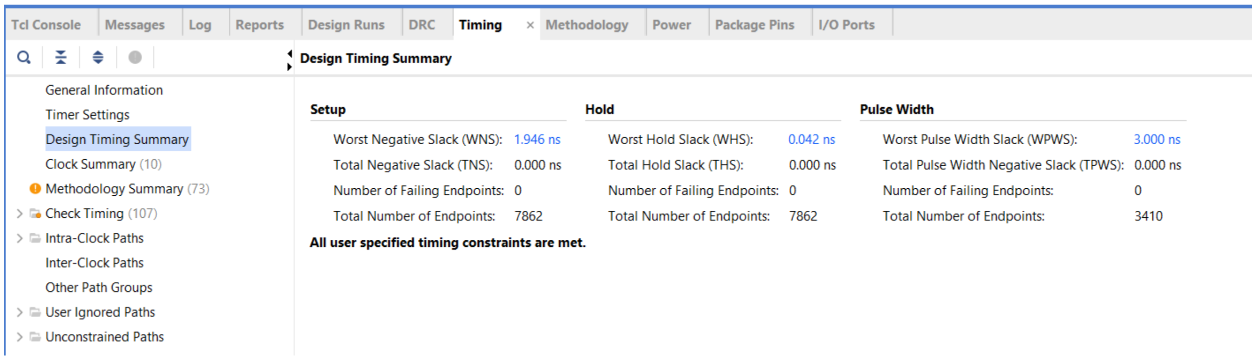
d. **You should have learned about interrupts in ECE 220, and it is obvious why interrupts are useful for inputs. However, even devices which transmit data benefit from interrupts; explain why. Hint: the UART takes a long time (relative to the CPU) to transmit a single byte.**
   Interrupts free the CPU from constantly polling for a ready signal resulting in increased efficiency. If the UART functioned through polling the CPU would have to poll for a ready bit, and then keep reading the UART and polling for a stop bit until it is done transmitting which is slow.
e. **Why are the UART and LED peripherals only connected to the data bus?**
   UART and LED peripherals are asynchronous, meaning they only need to send/receive data. There is no need to connect it elsewhere.
f. **You must be able to explain what each line of this (very short) program does to your TA. Specifically, you must be able to explain what the volatile keyword does (line 18), and how the set and clear functions work by working out an example on paper (lines 30 and 33).**
   The program declares a led_gpio uint32 pointer that is set to the base address of the LED GPIO block. It then OR's this pointer with x00000001 to force the data register (in position 0) high, then AND's the pointer with x11111110 to force the data register to low. Both operations do not modify the other registers of the struct. This results in a blinking LED. The volatile keyword means that this the variable can be modified by things external to its scope – this tells the CPU to not cache or otherwise optimize it. Set/clear was mentioned previously.
g. **Look at the various segments (text, data, bss), what does each segment mean?**
   Text contains the actual code of the program that is compiled/executed. The data section contains global or static variables that are initialized, and the bss contains uninitialized global/static variables.
h. **Why does the provided code, which does very little, take up so much program memory? Hint: try commenting out some lines of code and see how the size changes.**
   Printf links against a part of the C standard library, stdio.h which is

relatively large. When commented out the memory taken up is noticeably reduced.

i. **Make sure you understand the register map on page 10. If the base address is 0x40000000, how would you access GPIO2_DATA (for example?).**
GPIO2_DATA is accessed at x40000008 per the register map on page 10 as GPIO2_DATA is offset by x08 relative to the base address.

6. **Document the Design Resources and Statistics in table provided in the lab.**

	MicroBlaze
LUT	**2936**
DSP	**9**
Memory (BRAM)	**8**
Flip Flop	**3040**
Frequency	**0.1241MHz**
Static Power	**0.077W**
Dynamic Power	**0.385W**
Total Power	**0.461W**

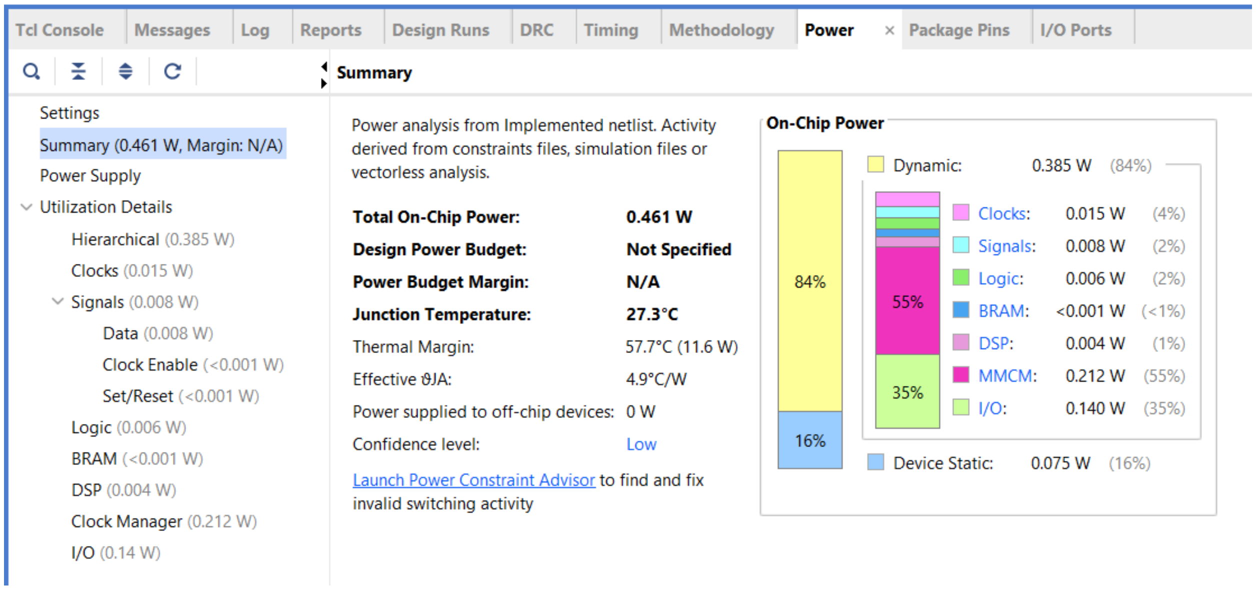

Tcl Console | Messages | Log | Reports | Design Runs | DRC | Timing | Methodology | Power | Package Pins | I/O Ports

Design Timing Summary

General Information
Timer Settings
Design Timing Summary
Clock Summary (10)
Methodology Summary (73)
Check Timing (107)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.946 ns	Worst Hold Slack (WHS):	0.042 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	7862	Total Number of Endpoints:	7862	Total Number of Endpoints:	3410

All user specified timing constraints are met.


Tcl Console | Messages | Log | Reports | Design Runs | DRC | Timing | Methodology | Power | Package Pins | I/O Ports

Summary

Settings
Summary (0.461 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (0.385 W)
Clocks (0.015 W)
Signals (0.008 W)
Data (0.008 W)
Clock Enable (<0.001 W)
Set/Reset (<0.001 W)
Logic (0.006 W)
BRAM (<0.001 W)
DSP (0.004 W)
Clock Manager (0.212 W)
I/O (0.14 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.461 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**27.3°C**
Thermal Margin:	57.7°C (11.6 W)
Effective ϑJA:	4.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

🟨	Dynamic:	0.385 W	(84%)
🟪	Clocks:	0.015 W	(4%)
🟦	Signals:	0.008 W	(2%)
🟩	Logic:	0.006 W	(2%)
🟦	BRAM:	<0.001 W	(<1%)
🟪	DSP:	0.004 W	(1%)
🟥	MMCM:	0.212 W	(55%)
🟩	I/O:	0.140 W	(35%)
🟦	Device Static:	0.075 W	(16%)

7. Conclusion

- **Discuss functionality of your design. If parts of your design did not work, discuss what could be done to fix it?**
  The 6.2 design takes input from a USB keyboard through the MAX3421E chip. This chip communicates with the microblaze processor through SPI protocol through the quad SPI IP in the block design. A VGA controller, ball file and color mapper generates R, G, B signals which the VGA to HDMI takes to generate the corresponding HDMI signals that output to the screen.
- **Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester?**
  N/A, nothing was super unclear.