

Lens Controller for Biomedical Cameras

By
Kevin Sha (ksha3)
Jihun Kim (jihunhk2)
Siddharth Sharma (sharma62)

TA: Zhicong Fan
Professor: Viktor Gruev

Final Report for ECE 445, Senior Design, Fall 2022

7 December 2022

Team 26

Abstract

This paper gives an overview of the design process and the results of a Lens Controller system, which aims to provide the ability to manipulate the lens remotely. This paper begins with describing the problem and the proposed solution to the problem as well as any approaches we have taken to reach the solution. The paper ends with the results of the project and future steps. Despite some failures, we managed to achieve most of our goals of the lens controller project as we were able to successfully control the lens.

Table of Contents

1. Introduction.....	1
1.1 Problem.....	1
1.2 Solution.....	1
1.3 Visual Aid.....	2
1.4 High-level Requirements List and Functionality.....	2
1.4.1 Aperture Functionality.....	2
1.4.2 Focus Functionality.....	2
1.4.3 Long Term Operation Reliability.....	3
1.4.4 Functionality	3
2. Design	4
2.1 Block Diagram.....	4
2.2 Subsystem Overview.....	4
2.2.1 FPGA	4
2.2.1.2 State Machine for SPI protocol.....	5
2.2.2 PCB Sub-Section	5
2.2.2.1 Rigid-Flex-Rigid PCB.....	6
2.2.2.2 Flexible PCB.....	8
2.2.3 Lens.....	10
2.2.4 Program.....	11
3. Cost and Schedule.....	12
3.1 Cost.....	12
3.2 Schedule.....	12
4. Verification.....	13
4.1 FPGA	13
4.2 Lens	14
4.3 PCBs.....	15
5. Conclusion	17
5.1 Successes.....	17
5.2 Failures	17
5.3 Future Works.....	17
5.4 Ethics and Safety	18
5.4.1 Safety	18
References.....	19
Appendix A State Machine Changes	21
Appendix B Modified Version of SPI protocol.....	23
Appendix C Lens Pin Ports.....	24
Appendix D Level Translation/Shift Circuit.....	25
Appendix E Python Code	26
Appendix F Requirements & Verification Tables	27
Appendix G Tolerance Analysis	30
G.1 FPGA	30
G.2 PCBs	30
Appendix H Cost of Parts Used.....	31
Appendix I Project Schedule.....	32
Appendix J Rigid-Flex-Rigid PCB Figures	34
Appendix K Flex PCB Figures.....	38

1. Introduction

1.1 Problem

In many operations, the margin for error is very slim. This is especially true for cancer treatment, where operation on tumors is considered one of, if not the only solution to cancer sickness. Operating on tumors requires a high degree of accuracy and so, the use of cameras to aid surgeons in the operating room would significantly reduce the risks associated with any mistakes involved in the removal of tumors. According to a study, incomplete tumor removal occurs in 25% of breast cancer patients, 35% of colon cancer patients and 40% of head and neck cancer patients [1]. From this, it can be seen that the problem is significant and requires a solution to this problem.

1.2 Solution

The solution to this problem is to develop a system where the lens of the camera can be adjusted based on a user input (A surgeon or surgery assistant) remotely, which would then help the surgeons in identifying any cancerous tumors and fully removing the tumors.

We are planning to use the FPGA to move the lens of the camera so that users can remotely control the lens from their computer. The PCBs will provide the connections between the FPGA and the lens due to the incompatible port assignments of the two components, which the PCBs will be responsible for correctly handling. We will be implementing a finite state machine and using the FPGA to control the overall operation of the camera. Users will be interacting with the movement of the camera using python code from their computer, by inputting their preferences for camera operation.

1.3 Visual Aid

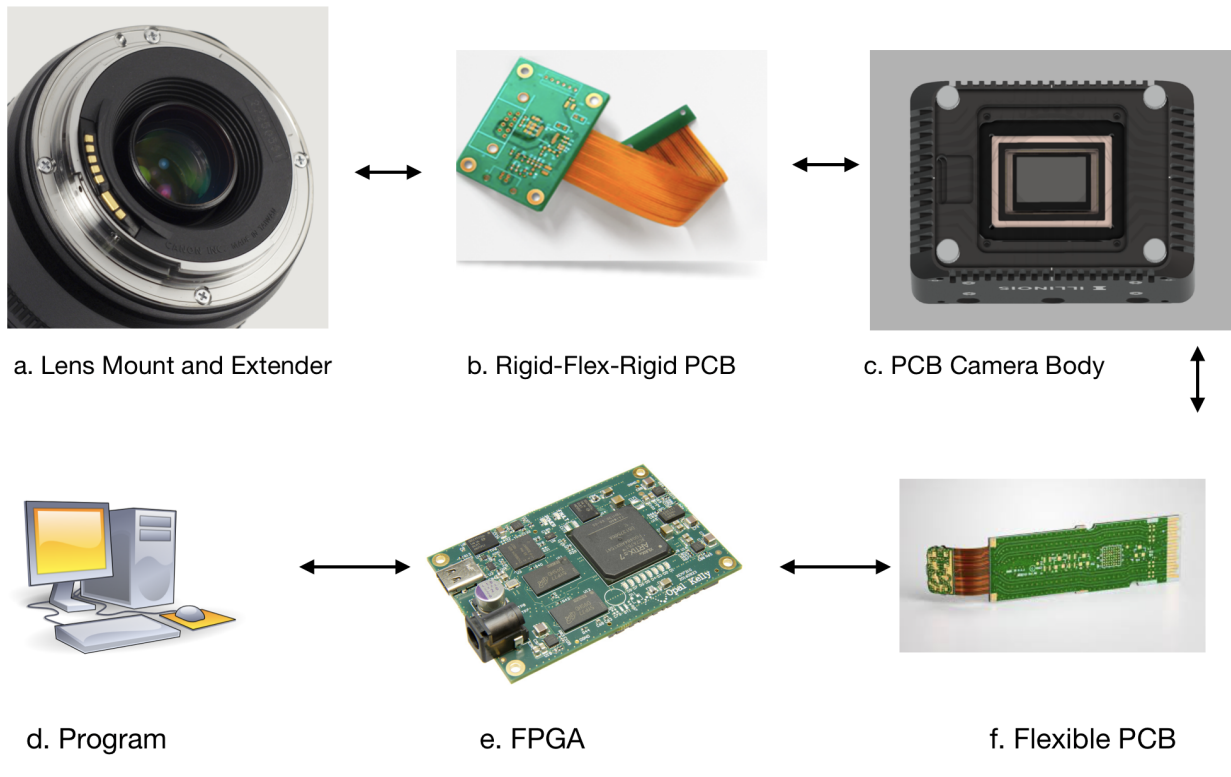


Figure 1: Visual Aid [2] [3] [4] [5] [6]

1.4 High-level Requirements List and Functionality

1.4.1 Aperture Functionality

Requirement : The lens must be able to change the aperture by the correct amount based on the command input by the user.

1.4.2 Focus Functionality

Requirement : The lens must be able to change the focus length by the correct amount on command input by the user.

The correct functionality of the camera lens will also be indicative of the correct mapping of the ports. This is important as it would indicate that the lens is receiving the correct signals from the FPGA and also outputting the correct signals back to the FPGA for user feedback.

1.4.3 Long Term Operation Reliability

Requirement : The system can run for at least 6 hours.

1.4.4 Functionality

Users will type in the command and any corresponding arguments required, into a Python program. Using the Spyder IDE, they will run the Python code and send the data to the FPGA. The FPGA will interpret the data using the Opal Kelly modules, which will be responsible for interpreting the command and arguments and storing them in the FPGA's registers. The FPGA will instantiate a clock cycle and will then send the command and arguments to the lens through the PCBs using the SPI protocol. After the lens has successfully carried out the command, the FPGA will instantiate another clock cycle to retrieve the response back from the lens, once again using the SPI protocol, which will be sent back to the PC for the users to view the response.

Depending on the command and arguments that the users input on the python program, the lens will execute the command accordingly. There are two main commands that we will focus on: changing focus and changing aperture. In addition to these commands, our design also supports a synchronization command as well.

We need to ensure that the camera can operate correctly for a long period of time. More specifically, the FPGA program should not be entering into any forbidden states during an operation. We decided to implement our system for at least 6 hours as that seems to be roughly the average time for cancer operations [7]. This is an important task as the camera lens needs to work in an operation-like setting and operations can last multiple hours.

2. Design

This section gives a description of each subsystem of the project.

2.1 Block Diagram

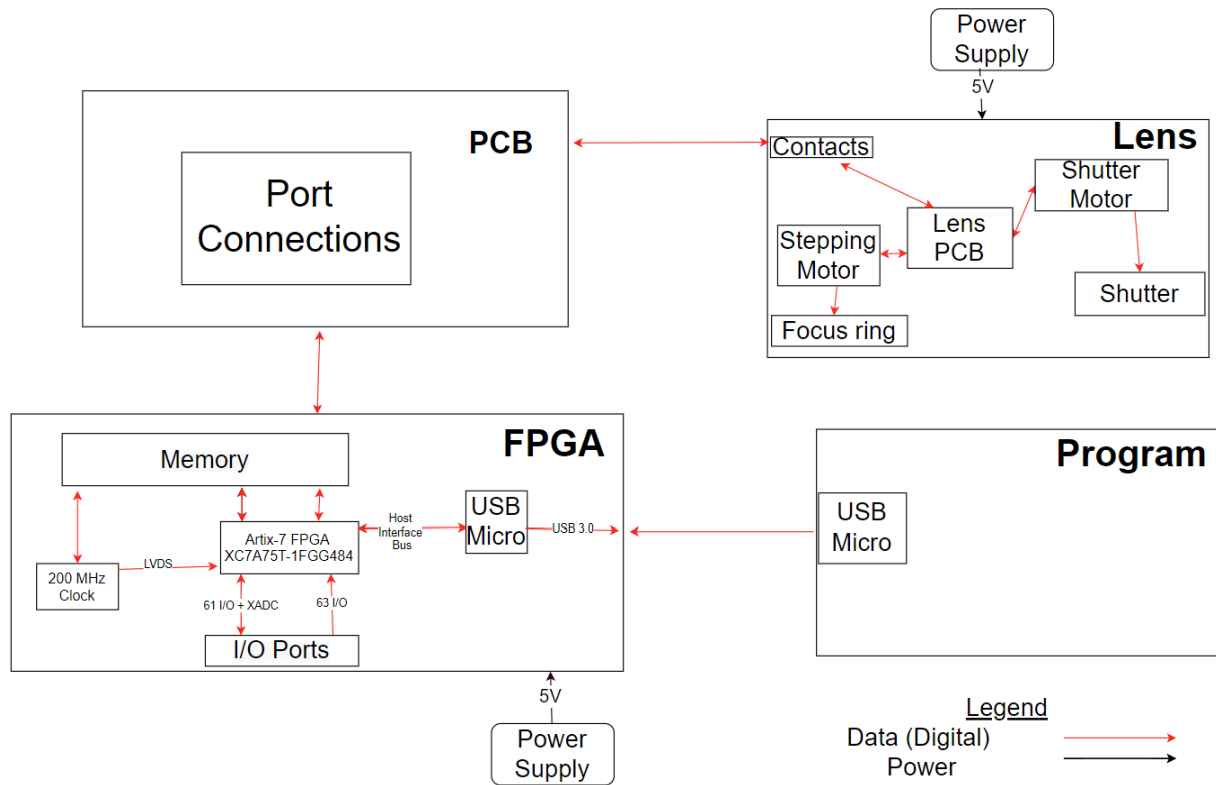


Figure 2: Block Diagram

2.2 Subsystem Overview

2.2.1 FPGA

The FPGA board component for our project is the XEM7310-A75 (for testing) and XEM7310-A200 (for actual design) by Opal Kelly. The FPGA will interact with the PC through the USB 3.0 port and control the lens using I/O pins through the flexible PCB board. The FPGA board will be receiving a power supply of 5 V from the computer through the USB 3.0 port.

2.2.1.1 ok modules

There are two Verilog modules that we will be using to enable the communication between the FPGA and the python code on the PC. The first one is the okWireIn module which allows us to input 32-bit data in the specified register location on the FPGA to the PC. The last one is the okWireOut module which sends 32-bit data from the registers on the FPGA to the PC.

Because of the interaction between the python code and the Verilog code, we need to ensure that the register location that we specify must be consistent between the two codes.

2.2.1.2 State Machine for SPI protocol

The SPI protocol is for the communication between the lens and the FPGA. After the FPGA receives the command, sometimes with the arguments, it will transfer those bits to the lens and receive response back from the lens using SPI protocol.

The state machine design underwent several design changes. First, we started off with a simple state machine where we have a start state and when users input a command, we move on to the command state. If the command requires additional arguments, it will move on to the argument1 and argument2 states, otherwise, it will return back to the start state. Each state will output an 8-bit signal which will be input to an SPI IP block which will do the job of SPI protocol. However, it was too difficult for us to instantiate the block and because the Canon lens is using a modified version of the SPI protocol, we had to implement it by ourselves.

The explanation for the modified version of SPI protocol is in Appendix B.

So in the new design, we decided to separate the single state into 8 different states and have each state output one bit of the command or argument starting with the most significant bit.

To prevent setup and hold time violations, we then replicated each state into 4 states and have each state control the clock signal that we will be using for our SPI protocol.

The state flow diagram of the overall state machine is in Appendix A.

2.2.2 PCB Sub-Section

For our finalized project, our PCBs ended up being vastly different from what we had originally planned. To start, we had a rigid-flex-rigid PCB that connected the preexisting camera PCB with the contact pads on the other side of the camera mount. This complicated flexible circuit would be the first of two connectors that brought everything together into one entity with properly transmitting signals. The second rigid section served as the contact through contact pads and pins to the second flexible PCB, which would then connect to the FPGA connector. As you can see, the use of these two PCBs carried out the work of what we thought one PCB could do when we were originally designing the project.

As for the PCBs themselves, they did not have any electrical circuit components besides wires and connecting contact pads. This is because all we were looking to do was properly transmit the signals to and from the camera and the FPGA, which had incompatible port mappings to begin with. It fell on us to figure out the proper schematic and wiring so that the corresponding signals from each component would line up with one another, and the commands

sent executed accordingly. However, the reasons for using the flexible PCBs remained the same: extremely restrictive mechanical constraints.

Our reasoning can be realized from our original design proposal: the contacts on the lens are very thin and also the port assignments between the FPGA and lens are incompatible. A flexible PCB would be better suited as it can handle multiple port assignments and through the specific sub-model of the flexible PCB known as a flat flexible cable, be able to perform the function of wires in transporting data signals. This will allow better connection with the fine contacts on the lens and allow the FPGA and lens to communicate through the reworked port assignments. Furthermore, this data will be transferred using an 8 bit modified SPI protocol which will be responsible for handling the seven data signals involved.

Some advantages of the flexible PCB include the fact that it is very flexible and thus can be used in a wider range of applications. In addition, there is very little wire connection which increases its reliability by preventing accidental shorting as with traditional PCBs which have that possibility. They take up less physical space and are much easier to use and transport, however their storage procedure is much more complicated and must be done properly. Going off those same lines, these flexible PCBs are very easily damaged and hard to repair, as they take a longer time to manufacture in the first place due to its complexity and eventual simplicity. The cost of resources is higher, but as such there can be greater circuit density on the boards.

In our implementation, we can use the concept of the flexible PCB in multiple ways. In one way, the flexible PCB would be implemented in its full capability where we would bend the circuit board in order to reduce space and to accommodate the lens' inconvenient port setup. This would involve calculating and designing the mechanical parameters of a flexible circuit board such as the bend angle, appropriate thickness, bend radius and the frequency of flexing [2]. This would be our ideal option as it would allow the camera lens to be used without any further changes.

2.2.2.1 Rigid-Flex-Rigid PCB

The design of the rigid-flex-rigid PCB had two aspects: Mechanical and electrical. The mechanical aspect involved analyzing the lens mount, adjusting our design to the mechanical constraints by determining any bending angles and regions that were required to ensure proper signal transport through the lens mount. The electrical aspect of the design was composed of routing and wire connection debugging and choosing the correct materials for the PCB.

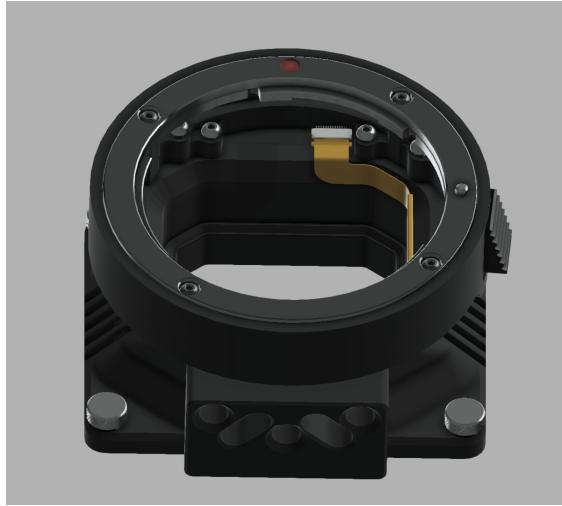


Figure 3: CAD Drawing of Lens Mount

The rigid-flex-rigid PCB would be placed in between the mounted screws as seen in Figure X. The thickness of the semicircular region, below the screws, is exactly 1 mm, which means that the maximum thickness of the rigid section would be 1 mm as well. The rigid section would also need to have a connector, which would connect the gold flexible PCB (As seen in Figure 3) to pass on the signals from the lens ports. As a result, the connector would have to be placed on the rigid region, further limiting space for making the relevant port connections.

The flex region would have to be bent in certain ways to navigate the lens mount. Shown below in Figure 4, is a bottom view of the lens mount which shows the contact pins of the lens mount.

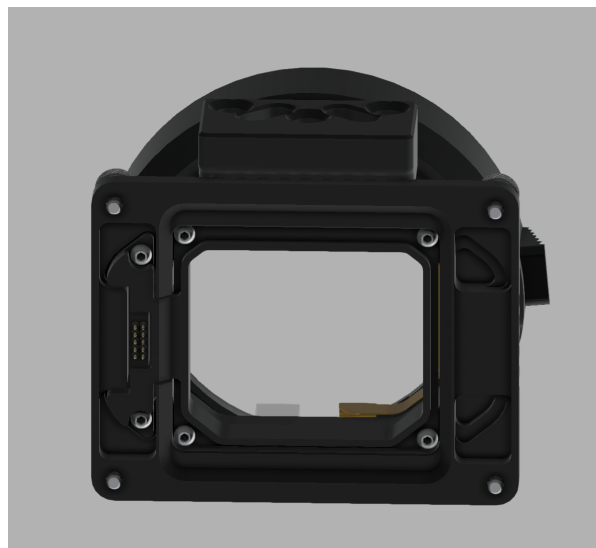


Figure 4: Bottom View of Lens Mount

From the connecting rigid PCB that takes the signals from the orange camera PCB, the flex region will have to extend outwards and essentially hug the wall of the lens mount so that it can reach the contact pad area. This means many 90 degree bends, going in many orientations. The flex region first bends downward, then to the side a few times, then back upward to go back into the second rigid PCB that has the contact pads. Through this process of bending the PCB in the designer, we realized that the bend radius was dependent on the bend angle: the bigger the angle, the bigger the bend. However this was not accurate enough for us, as we needed an exact replica of the 3D model. Thus we calculated the bend radius by going into a 3D modeling software and checking it from there. You can see the results in Appendix J as shown in our model. Finally, we were able to bend the PCB to their desired specifications.

The signals will travel from the gold flexible PCB, connect to the first rigid section of the rigid-flex-rigid and then flow across the flex region, reaching the second rigid section, which will have the contact pads to latch onto the contact pins. These contact pads will connect to the top of the contact pins, which will allow the signals to travel through the contact pins and out of the lens mount, thereby accomplishing its goal.

The electrical aspect of the design focused on optimizing signal integrity. This was done in two ways: routing the traces and considering the layer stackup. Routing was important because we had to take into consideration many factors. The first one was that there were constraints on the space between the wires, and we had to ensure that there was no potential wire breakdown where the PCB was going to be bent. The next factor was that we had to make the width of the power and ground traces bigger, again pushing the limits on the space between the wires. We also had to utilize vias to wire between different layers. This was crucial in solving our wiring problems as we were able to take the signals from the top/bottom layer, which would be where the connector was, and send the signals through the middle of the PCB, which was obviously the flex region. The routing schematic is shown in Appendix J.

The layer stackup was important as it allowed the rigid-flex-rigid PCB to function properly. As stated before, the flex layers emerge from the rigid layers, which means that the flex layers are consistent across all regions. The layer stackup used is seen in Appendix J, which shows Layers 2 and 3 as the flex layers, being identical across all the regions. The core layer provides the flexibility while the plane and signal layers carry the signals to the destination. As stated before, the rigid sections need to be less than 1 mm thick, which is accomplished in the layer stackup as the rigid sections are 0.8 mm.

2.2.2.2 Flexible PCB

The same design approach was used for the flexible PCB. This PCB was responsible for taking the signals from the contact pins of the lens mount and connecting it to the FPGA. It will start with connecting onto the contact pins on the bottom side of the lens mount through its contact pads (Identical in design as the ones used in the Rigid-Flex-Rigid PCB Subsection).

This is seen in Figure 5, which shows a top view of the camera body, which will attach to the lens mount. The small engraved area on the left will serve as the contact area, where one end of the flexible PCB (Mounted with the contact pads) will be placed. Once the lens mount is placed, the contact pins will make contact with the contact pads and the signals are ready to the FPGA.

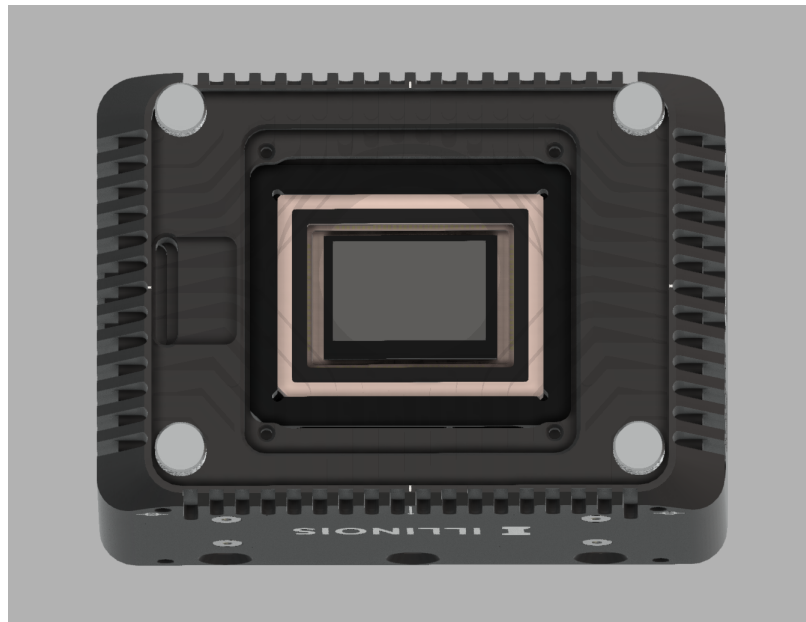


Figure 5: Top view of Camera Body

Now that the signals are ready to be transported, another challenge comes into play. The FPGA connector is directly beneath the contact area shown in Figure 5. This is shown in Figure 6, shown below, which displays a side view of the camera body. The FPGA connector is the white component shown in the figure.

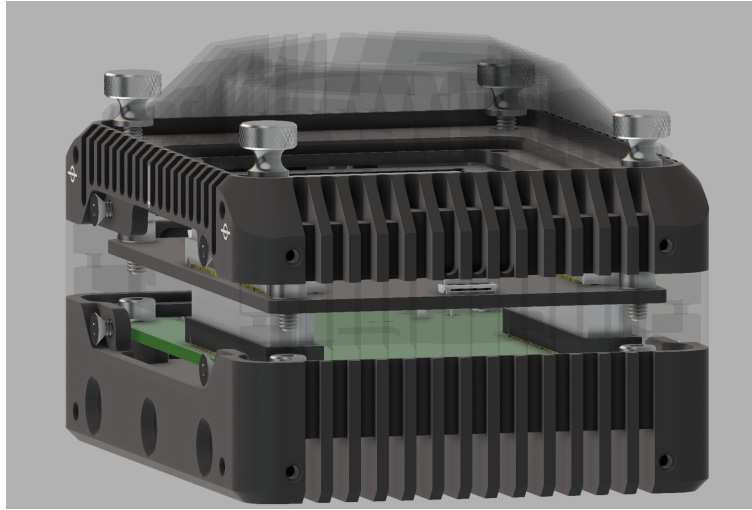


Figure 6: Side View of Camera Body

As a result, it was determined that the bend angle of the design would be 180 degrees, due to the FPGA connector being directly beneath the contact area on the top of the camera body.

The electrical aspect of this PCB was similar to the previous PCB, but with a few major changes. This PCB is a flex-only PCB and so only has two layers in its stackup. The materials used were the exact same as the flex region in the previous PCB, but had different thicknesses due to there being no rigid section. The overall thickness of the flex PCB is 0.13 mm. The layer stackup for this PCB is in Appendix K.

The edge that is being connected to the FPGA connector will be beveled and electroplated with gold to ensure better connection with the FPGA connector. This edge also had to be stiffened with FR-4 in order to ensure it stays inserted. On the other edge, contact pads were used but since this PCB is a flex PCB, it was more optimal to etch in contact pads with the same dimensions as those in the previous sub-section. The routing schematic is in Appendix K.

2.2.3 Lens

There are 5 main commands that our FPGA is able to send to the lens :

- 0x12 : Change focus + 2 argument
- 0x44 : Change aperture + 1 arguments
- 0x05 : Change focus to MAX
- 0x06 : Change focus to MIN
- 0x0A : Synchronization command

Each command and arguments will be 8-bit length data.

The lens will have an aperture motor which will control the aperture of the lens, and it will also have a focus motor which will change the focus length of the lens.

The change focus command and change aperture command will require additional arguments to tell the lens how much to open the aperture and how much to change the focus by. The change aperture command requires an 8-bit value, so we send an 8-bit argument whereas change focus command requires a 16-bit integer. Hence we send in two 8-bit arguments to the lens which will combine the two values to form a 16-bit integer.

The camera lens will have the focus mode switch set to manual so that the user can change the focus according to their own specifications.

The lens will interact with the power supply subsystem as it will be powered through a 6V supply and will also interact with the flexible PCB to receive instructions on how to adjust the appropriate camera features.

2.2.4 Program

The users are able to input command and argument values for the lens operation using python code on their PC. We are using a Spyder IDE due to its simplicity in the debugging process. There will be two main functions that we will be using that enables the PC to communicate with the FPGA.

1. SetWireInValue function

This function enables us to write data to one of the registers on the FPGA board.

2. SetWireOutValue function

This function enables us to read data from one of the registers on the FPGA board.

The program component will be implemented through the use of a personal computer or laptop. The PC will be connected to the FPGA using the USB port.

The program component will mainly be communicating with the FPGA subsystem to effectively operate the camera lens.

The sample python command is in Appendix E.

3. Cost and Schedule

3.1 Cost

See Appendix H for Parts Cost table

The average hourly salary of a graduate electrical engineer is \$38. [8]

We will be working approximately 13 hours per week for 10 weeks. Therefore:

$$\$38/\text{hour} \times 130 \text{ hours} = \$4,940 \text{ per person}$$

$$\text{Total Labor Cost} = \$4,940 * 3 * 2.5 = \$37,050$$

Our project does not require any labor from the machine shop.

Hence:

$$\text{Total Project Cost} = \text{Total Labor Cost} + \text{Total Parts Cost} = \$39,497.52$$

3.2 Schedule

See appendix C for the table.

4. Verification

4.1 FPGA

For the FPGA portion, we used two main tools for testing. The first one was the simulation tool in the Vivado program that helped us simulate the state machine before we can actually test on the FPGA board. We had to undergo several simulations and debugging to ensure that our state machine works as expected.

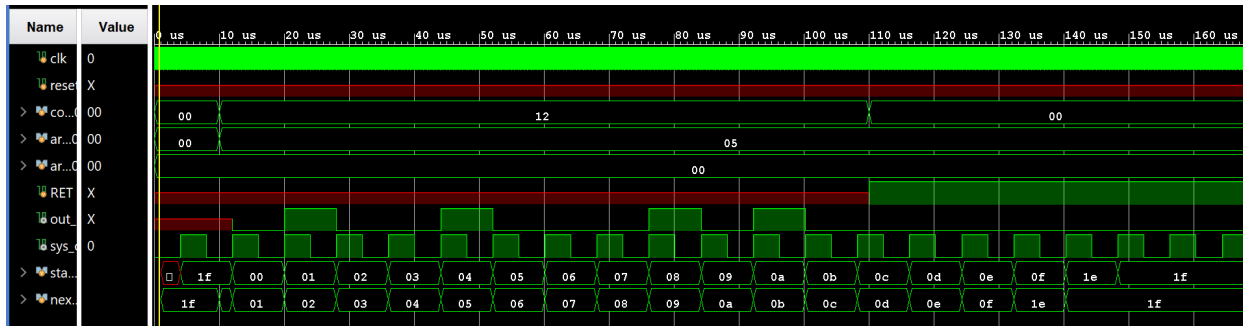


Figure 7: Using Vivado's simulation tool

We had to write a testbench that sets the inputs for the state machine. With each input set by the testbench, we need to check if the clock signals are correctly operating and if the output command and argument bits are the same as the input bits for the command and arguments.

As for the testing on the lens, we had to use an oscilloscope to test the signals coming from the FPGA breakout board. We also needed to build a level translation circuit to convert 3V signal to 5V signal as the lens only accepts 5V signals.

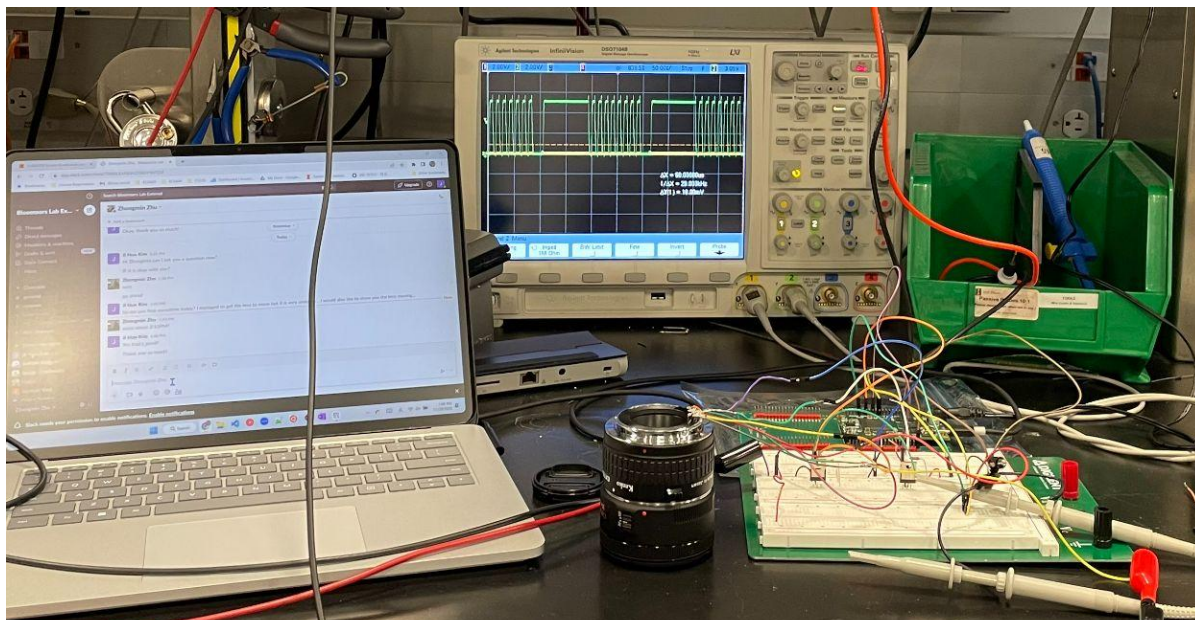


Figure 8: Observing signals using Oscilloscope

We initially tested signals coming out of the FPGA breakout board. After confirming that correct signals are captured, we made the signals coming out of the breakout board the input to the level translation circuit. The additional circuit will transform a 3.3 V signal coming out of the breakout board to 5 V signal. We had to build two level translation circuits, one for the clock and the other for the command signal. Unlike some level translation circuits, the one that we built is unidirectional. And because it is going through a MOSFET, we do not get a perfect square waves at the output of the circuit.

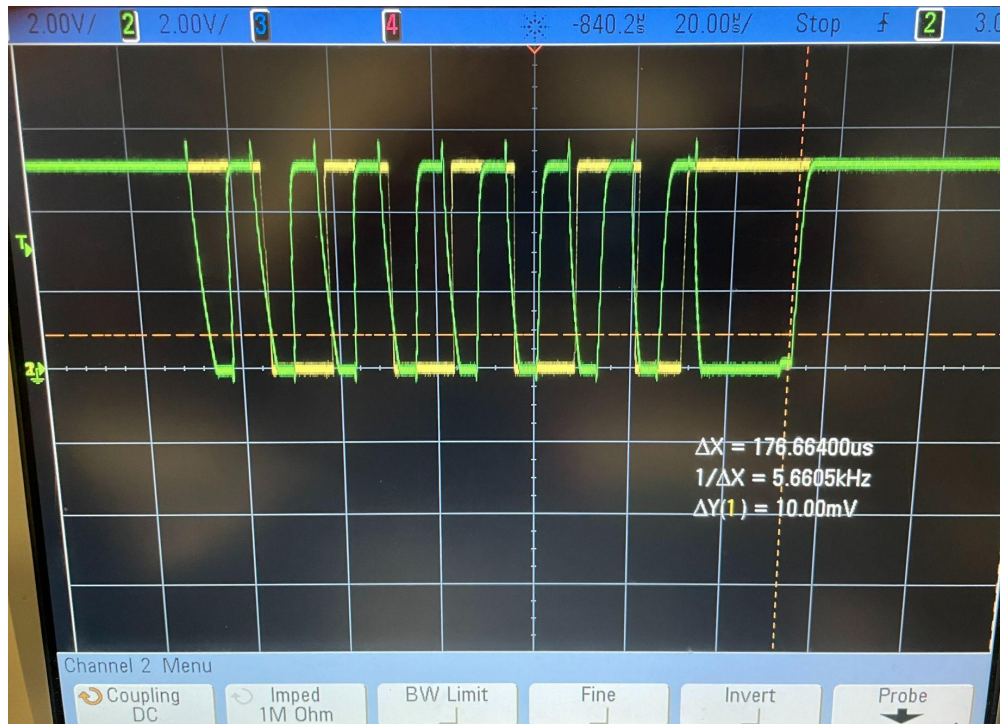


Figure 9: The command signal (yellow) on top of clock signal (green)

We had to check that the clock is always capturing the command signal when it is constant (either high or low) on the rising clock edge to prevent glitch. Using the scale, we also can verify that the clock period is at around 12.5 μs .

The circuit schematic for the level translation circuit is given in Appendix D.

4.2 Lens

We also used an oscilloscope to test if the lens is operating correctly. We used an oscilloscope to check if the lens is responding with the correct set of bits for the command that we are sending to the lens.

The same rule applies with the response bit. The response signal should be captured on the rising clock edge when the response signal is constant. Unlike the command and clock signals going into the lens, which are the outputs of the level shift circuits, the response signal is almost a perfect square wave.



Figure 10: Response signal (yellow) on top of clock signal(green)

We can verify if the response is correct by checking what command that we have sent. The response for the synchronization command is 0xAA, and for other commands, the response is the same as the command. So by checking the responses for each command that we sent, we have verified that the lens is operating correctly and responding with correct response bits.

The lens pin-out ports are shown in Appendix C.

4.3 PCBs

The PCBs' designs were verified while we were designing the PCBs. Altium Designer looks out for any design violations and promptly alerts the user if there are any issues. This was implemented both in designing the schematics and routing. Furthermore, the design constraints in our Tolerance Analysis would have to be accounted for and thoroughly verified before the PCBs were sent for fabrication. With the type of PCBs that are being used, the fabrication time is much longer and so, any errors in the design stage would not be easy to fix.

In the schematic portion of the design, the major errors were incorrect allocation of input or output status, short-circuit violations and broken netlists. For our finalized schematic designs, Altium did not detect any errors and thus, this verified that the schematic was implemented correctly.

In the routing and stackup stage, there was even more verification required. For the stackup, several checks had to be conducted. The flex layers had to be properly extended, the

rigid sections had to be at most 1 mm, the signal layers had to be placed correctly and for the flex region and flex PCBs, coverlay was required to act as the solder mask. Appendices J and K can be used to verify that these requirements for the design were met.

For the routing stage, there were some considerations that had to be fulfilled. The trace width for the camera signals had to be at least 8 mils and 20 mils for any power signals such as the 6V signal or VDD. Furthermore, the trace separation had to be at least 8 mils. The via diameters had to be at least 10 mils in order for safe transport between the layers. These were all checked by Altium as being fulfilled and so, can conclude that it was verified.

When implementing the bend angles and the bend radius, Altium also checked whether the bend angle was too severe or not. Altium checked our design for any mechanical violations and did not report back any, verifying that our needs were met. Throughout this entire process as described above, we learned about design rule management through the Constraints Editor within Altium, where we could change the tolerances and specifications of everything mentioned above and more to ensure that the PCB would pass the verification and file generation checks.

5. Conclusion

5.1 Successes

For the FPGA and PC subsection of the project, we managed to make the lens move according to the command input by the user on their PC using python codes. This was possible because we managed to implement the modified version of the SPI protocol that Canon lenses use using the state machine in Verilog code.

For the PCB side of the project, some of our key successes include: un-bending the CAD model & making it our board shape, using vias to ensure optimal signal routing, implementing different layer stackups, and application of rigid-flex and flex PCB knowledge.

5.2 Failures

One thing that we failed to do with the lens operation is that we did not manage to figure out the argument values that we can put in together with the change aperture and change focus commands. It seems like there are certain values that can be accepted by the commands depending on what state the aperture or the focus is at when sending the command to the lens.

On to the hardware components, our biggest setback was that we believed that the port mapping of the FPGA connector was the port mapping of the lens connector, and we did not anticipate the process of actually ordering the PCB.

5.3 Future Works

There are other lens commands that we did not implement such as extracting information regarding the lens such as aperture and focus states. In order to support these commands, we need to change the state machine. This is because the current state machine is only able to receive a byte of command. We need to modify the state machine in such a way that it is able to accept responses with varying responses coming from the lens as different commands have different lengths of the response. If our state machine is able to support this, it would be possible for us to figure out what arguments we can put in for the changing focus and aperture commands. These can be written as python functions to improve the usability of the users.

5.4 Ethics and Safety

5.4.1 Safety

As with any mechanical contraption, there is always a risk of the machine malfunctioning and worst case scenario, exploding. The PCB will need to be structured properly, as any mismatching connection can cause a short circuit. The camera breaking down during its intended operation will also pose a risk to the patient, and we most definitely want to avoid this. However, for the scope of our project, this will not be a main concern.

Regarding the camera lens, there are a few precautions that the manufacturer themselves have warned about in the manual. To quote the Canon manual “Whether it is attached to the camera or not, do not leave the lens under the sun without the lens cap attached . This is to prevent the lens from concentrating the sun’s rays, which could cause a fire. If the lens is taken from a cold environment into a warm one, condensation may develop on the lens surface and internal parts. To prevent condensation in this case, first put the lens into an airtight plastic bag before taking it from a cold to warm environment. Then take out the lens after it has warmed gradually. Do the same when taking the lens from a warm environment into a cold one. Do not leave the lens in excessive heat such as in a car in direct sunlight. High temperatures can cause the lens to malfunction.” [9]

Another potential risk that they have stated in their disclaimers is the fact that there is no guarantee that the interference will not occur in any particular installation. It is of course tested and compliant with part 15 of the FCC regulations for a class B digital device, designed to provide reasonable protection against harmful interference in a residential installation. As such, the camera can generate, use, and radiate radio frequency energy, and if not adhered to the proper usage regulations, may cause harmful interference to radio communications or television reception. Should this happen, there are instructions for the user to attempt and correct the complication.

References

- [1] “Hexachromatic bioinspired camera for image-guided cancer ... - science.” [Online]. Available: <https://www.science.org/doi/10.1126/scitranslmed.aaw7067>. [Accessed: 15-Sep-2022].
- [2] “Computer Graphics cliparts #2809421 (license: Personal use),” *animated picture of computer - Clip Art Library*. [Online]. Available: <http://clipart-library.com/clipart/621480.htm>. [Accessed: 07-Dec-2022].
- [3] A. Bahl, “Avoiding common flexible PCB errors,” *Sierra Circuits*, 19-Oct-2022. [Online]. Available: <https://www.protoexpress.com/blog/avoiding-common-flex-pcb-errors/>. [Accessed: 07-Dec-2022].
- [4] “Home,” *Flex PCBs | Rigid Flex PCBs | PCB Unlimited*. [Online]. Available: <https://www.pcbunlimited.com/products/rigid-flex-pcbs>. [Accessed: 07-Dec-2022].
- [5] Opalkelly.com, “XEM7310,” *Opal Kelly*, 16-Nov-2022. [Online]. Available: <https://opalkelly.com/products/xem7310/>. [Accessed: 07-Dec-2022].
- [6] “Canon EF lens mount,” *Wikipedia*, 22-Nov-2022. [Online]. Available: https://en.wikipedia.org/wiki/Canon_EF_lens_mount. [Accessed: 07-Dec-2022].
- [7] S. Watson, “Lung cancer surgery recovery time: How long does it take?,” *Healthline*, 11-Jan-2021. [Online]. Available: <https://www.healthline.com/health/lung-cancer/surgery-recovery-time?fbclid=IwAR1y1RB5a07RPHhE-MB3Av2rNQrBs9l45UcHLqTW0GCW6TspK91zgSEsjfY#surgery-length>. [Accessed: 07-Dec-2022].
- [8] Grainger Engineering Office of Marketing and Communications, “Salary averages,” *Electrical & Computer Engineering | UIUC*. [Online]. Available: <https://ece.illinois.edu/admissions/why-ece/salary-averages>. [Accessed: 28-Sep-2022].
- [9] “EF-S 18-55mm f/3.5-5.6 IS STM,” *User manual Canon EF-S 18-55mm f/3.5-5.6 IS STM (English - 14 pages)*. [Online]. Available: <https://www.manua.ls/canon/ef-s-18-55mm-f35-56-is-stm/manual>. [Accessed: 15-Sep-2022].

- [10] "Photo tech canon EOS-EF protocol - JP79DSFR.FREE.FR." [Online]. Available: http://jp79dsfr.free.fr/_Docs%20et%20infos/Photo%20Tech%20_%20Canon%20EOS-EF%20Protocol.pdf. [Accessed: 06-Dec-2022].
- [11] Y. Bando, "How to move Canon EF lenses ," *MIT Media Lab*. [Online]. Available: https://web.media.mit.edu/~bandy/invariant/move_lens.pdf. [Accessed: 07-Dec-2022].
- [12] "Electronic circuit design - MOSFET logic level shift," *MOSFET level shift - Electronics information from PenguinTutor*. [Online]. Available: <http://www.penguintutor.com/electronics/mosfet-levelshift>. [Accessed: 07-Dec-2022].

Appendix A State Machine Changes

The initial state machine design was simple. It starts at the start state. It then transitions to the command state. If the command requires some arguments, it moves to argument 1 state. It will move to argument 2 state if required. Each state will output 8-bit data which would go to the standard SPI core IP block which will carry out the SPI protocol and send the 8-bit values to the lens bit-by-bit.

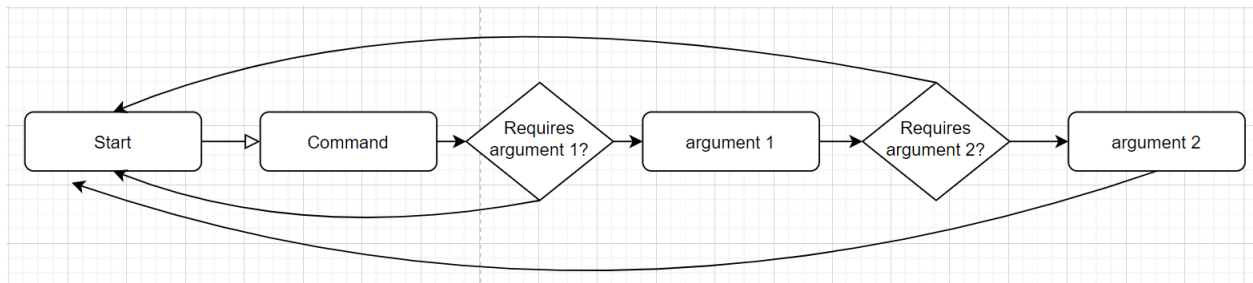


Figure 11: Initial State Machine Flow Chart

However, instantiating the block was a difficult process. So we decided to implement the SPI protocol by using the state machine. Instead of having a single state for the command, we separated that into eight different states that output one bit of the 8-bit data starting with the most significant bit.

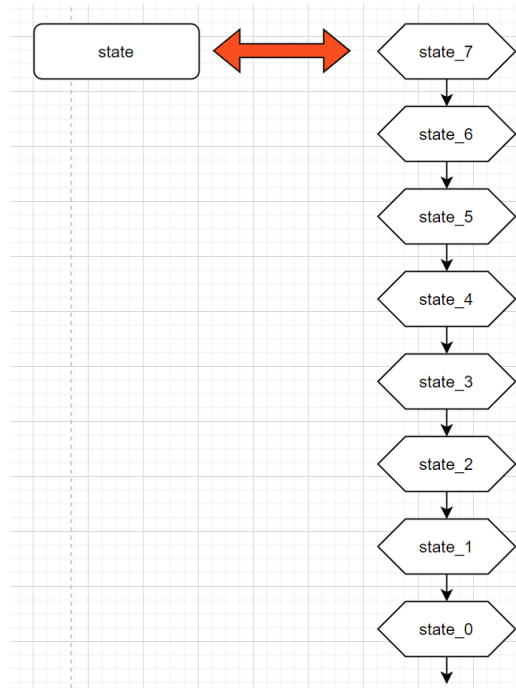


Figure 12: Changing a single state into 8 different states

Another change that we made was that in order to prevent setup time and hold-time violation, we needed to replicate the single command state to four states. And have the command bit to be constant throughout the four states. In addition, the command states will also control the clock signal that will be used for the SPI protocol.

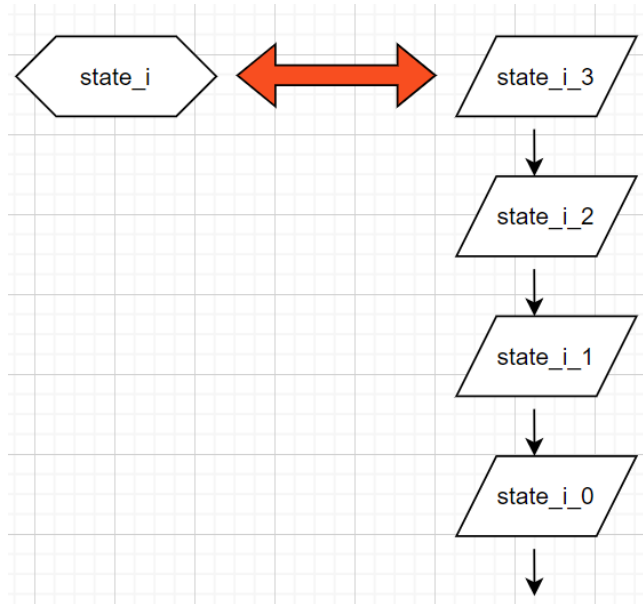


Figure 13: Replicating the state into 4 states

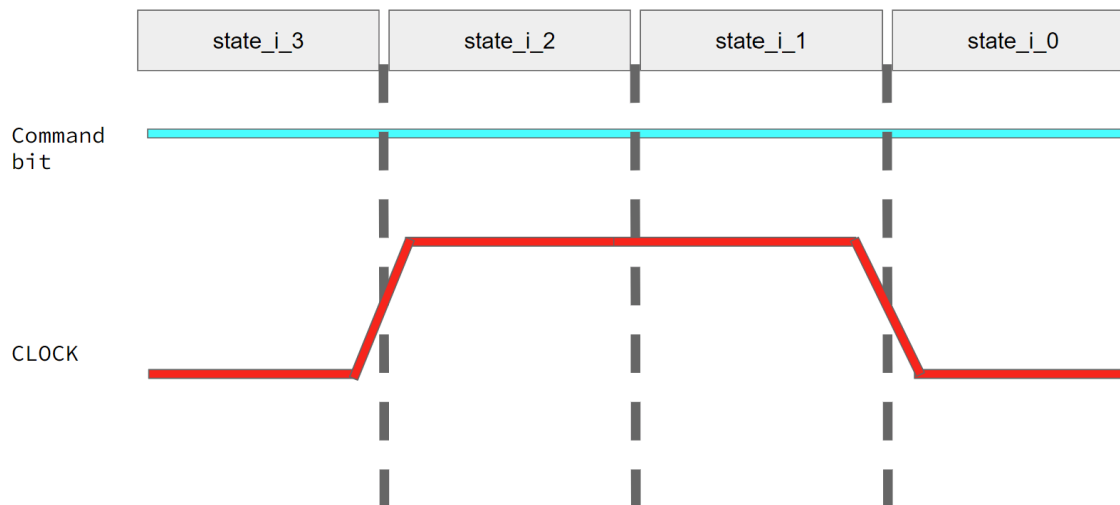


Figure 14: New state machine design to prevent setup and hold-time violation

Appendix B Modified Version of SPI protocol

The Canon lens that we are using uses a modified version of the SPI protocol. In addition to eight clock cycles, we need a ACK/busy state (acknowledgement cycle) which typically lasts for 15 μ s and the signal needs to stay high for another 110 μ s.

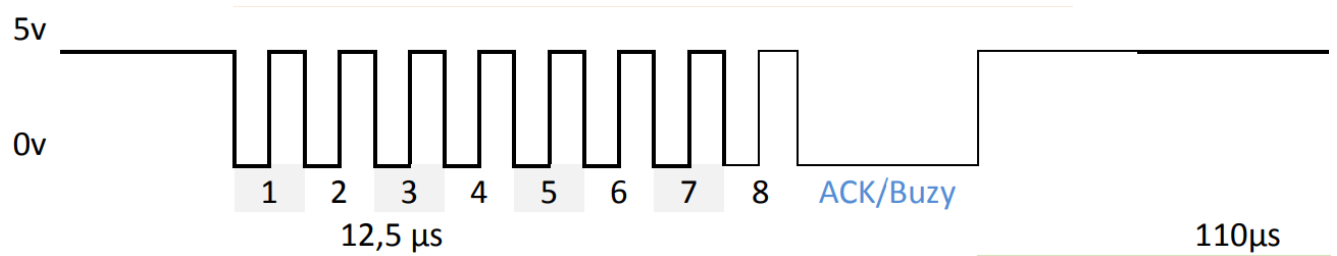
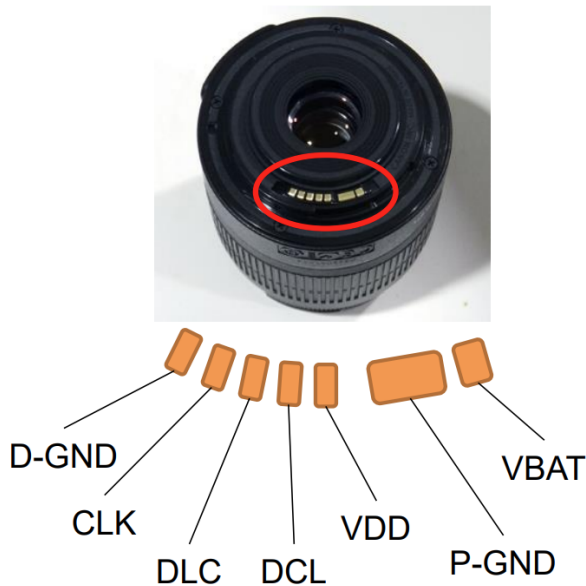


Figure 15: Modified version of the SPI protocol that Canon lens uses [10]

Appendix C Lens Pin Ports

Lens Pin-out

- Protocol: SPI (serial peripheral interface)
 - 8 data bits, 1 stop bit



VBAT: 6V power for lens motors
P-GND: Ground for lens motors

VDD: 5.5V power for digital logic
DCL: Data from camera to lens (MOSI)
DLC: Data from lens to camera (MISO)
CLK: Clock
D-GND: Ground for digital logic

MOSI: master output, slave input
MISO: master input, slave output
master: camera in this case
slave: lens in this case

Figure 16: Pin-outs on the lens ports [11]

Appendix D Level Translation/Shift Circuit

The level translation/shift circuit translates a logic from one level to another. For our circuit, the level is translated from 3.3 V to 5 V signal. The circuit schematic is shown below.

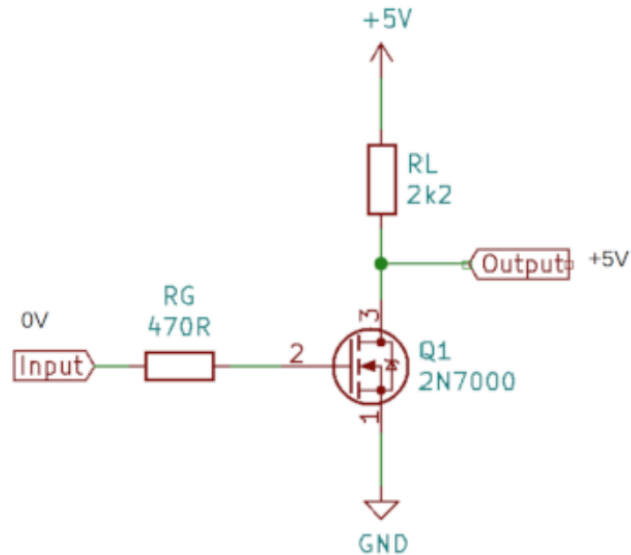


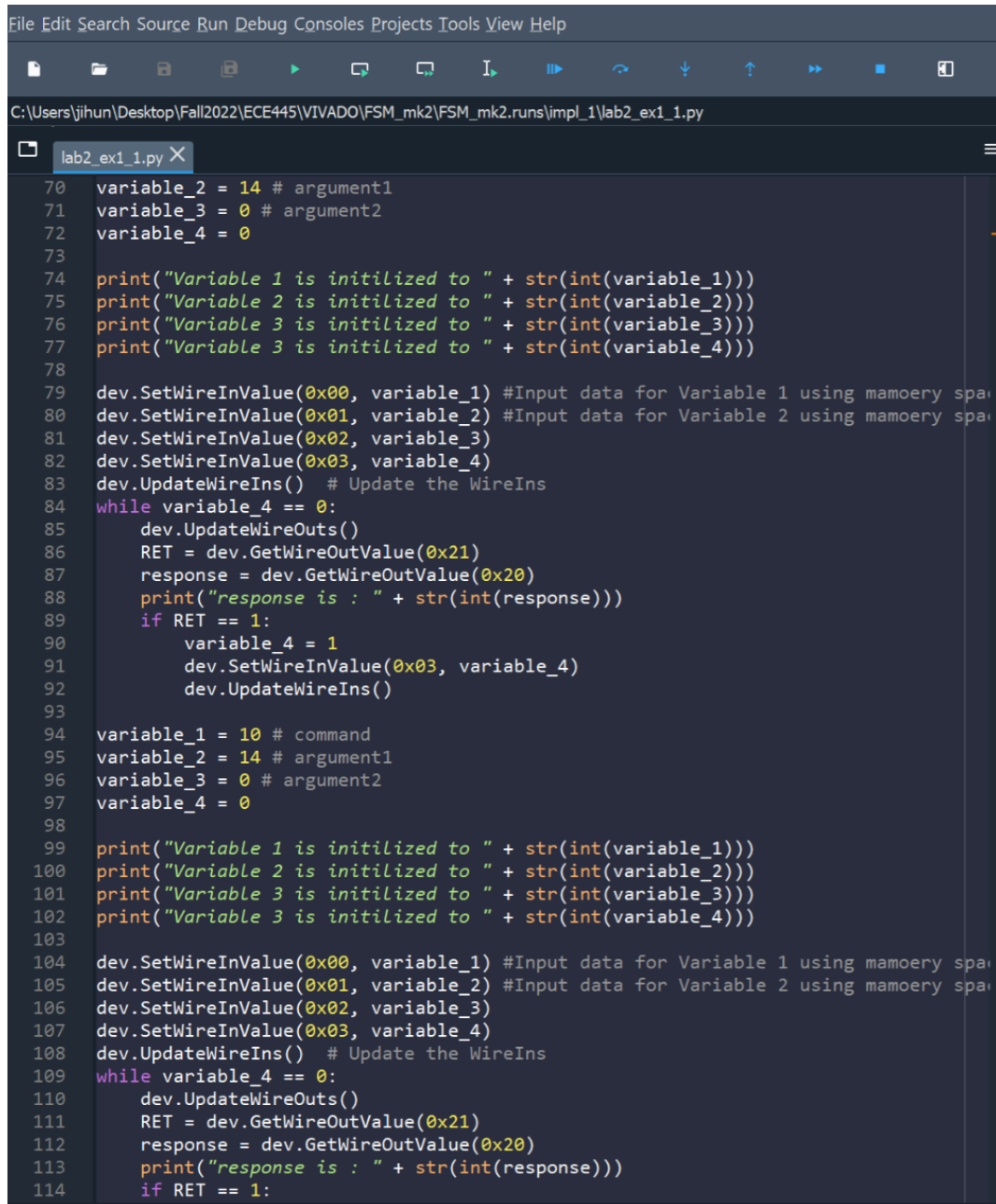
Figure 17: Circuit schematic for level translation circuit [12]

This is a one-directional MOSFET level shifter. The MOSFET is n-channel, the resistor RL is a pull-up resistor, and RG resistor is a resistor at the gate of MOSFET to prevent damage to the FPGA.

The output of the FPGA breakout board is fed into the input port of the level shifter, and the level shifter will output a 5 V signal. The signals that are fed into the level translation circuit are command and the clock signals.

We need a level shifter circuit in between the lens and the FPGA breakout board because the lens does not accept 3.3 V signals as it sees them as low. Translating the signals to 5 V signals ensures proper operation of the lens.

Appendix E Python Code



```
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\jihun\Desktop\Fall2022\ECE445\VIVADO\FSM_mk2\FSM_mk2.runs\impl_1\lab2_ex1_1.py

lab2_ex1_1.py X
70 variable_2 = 14 # argument1
71 variable_3 = 0 # argument2
72 variable_4 = 0
73
74 print("Variable 1 is initilized to " + str(int(variable_1)))
75 print("Variable 2 is initilized to " + str(int(variable_2)))
76 print("Variable 3 is initilized to " + str(int(variable_3)))
77 print("Variable 3 is initilized to " + str(int(variable_4)))
78
79 dev.SetWireInValue(0x00, variable_1) #Input data for Variable 1 using mammoery spa
80 dev.SetWireInValue(0x01, variable_2) #Input data for Variable 2 using mammoery spa
81 dev.SetWireInValue(0x02, variable_3)
82 dev.SetWireInValue(0x03, variable_4)
83 dev.UpdateWireIns() # Update the WireIns
84 while variable_4 == 0:
85     dev.UpdateWireOuts()
86     RET = dev.GetWireOutValue(0x21)
87     response = dev.GetWireOutValue(0x20)
88     print("response is : " + str(int(response)))
89     if RET == 1:
90         variable_4 = 1
91         dev.SetWireInValue(0x03, variable_4)
92         dev.UpdateWireIns()
93
94 variable_1 = 10 # command
95 variable_2 = 14 # argument1
96 variable_3 = 0 # argument2
97 variable_4 = 0
98
99 print("Variable 1 is initilized to " + str(int(variable_1)))
100 print("Variable 2 is initilized to " + str(int(variable_2)))
101 print("Variable 3 is initilized to " + str(int(variable_3)))
102 print("Variable 3 is initilized to " + str(int(variable_4)))
103
104 dev.SetWireInValue(0x00, variable_1) #Input data for Variable 1 using mammoery spa
105 dev.SetWireInValue(0x01, variable_2) #Input data for Variable 2 using mammoery spa
106 dev.SetWireInValue(0x02, variable_3)
107 dev.SetWireInValue(0x03, variable_4)
108 dev.UpdateWireIns() # Update the WireIns
109 while variable_4 == 0:
110     dev.UpdateWireOuts()
111     RET = dev.GetWireOutValue(0x21)
112     response = dev.GetWireOutValue(0x20)
113     print("response is : " + str(int(response)))
114     if RET == 1:
```

Figure 18: Python code we used in our project

The SetWireInValue function is the function that enables us to write into the register on the FPGA board. It requires two arguments; the address of the register and the value that we want to write with.

The GetWireOutvalue function enables us to read from the register on the FPGA board. It requires one argument, the address of the register on the FPGA that we want to read from.

Appendix F Requirements & Verification Tables

Table 1 FPGA R & V Table

<u>Requirement</u>	<u>Verification</u>
<ol style="list-style-type: none"> 1. The FPGA must be able to control the operation of the lens when the users type in commands in their PC 2. The FPGA must be able to communicate in between the camera lens using the 7-pin ports and the PC through the USB port. 	<ol style="list-style-type: none"> 1. Plug the FPGA board to the computer through the USB port. 2. Load the program to the FPGA board. 3. Send the command to the FPGA from the PC using python code. 4. Users will have to check whether the lens functions correspondingly.
<ol style="list-style-type: none"> 3. The FPGA must be able to maintain its operating temperature with the prolonged sustained work without overheating. (~6 hours) 	<ol style="list-style-type: none"> 1. Create a python code that runs the FPGA for a specified time. 2. Connect the FPGA to the PC and load the program. 3. Attach temperature sensor to the FPGA board 4. Run the program 5. Monitor the temperature of the FPGA
<ol style="list-style-type: none"> 4. The FPGA must not enter a forbidden state after prolonged operation. 	<ol style="list-style-type: none"> 1. Create a testbench in Verilog that contains loops (while or for) that runs the FPGA for a prolonged period of time. <ol style="list-style-type: none"> a. The testbench should be able to keep track of how many times it enters into each state and show how many times it did not enter the required states. 2. Run the testbench and check output generated by the testbench.

Table 2 PCB R and V Table

Requirements	Verifications
1. Series resistance	<ol style="list-style-type: none"> 1. Prepare the multimeter. 2. Use the probes of the multimeter to test two points on the PCB board to check the series resistance.
2. Capacitance between two lines	<ol style="list-style-type: none"> 1. Prepare the multimeter. 2. Use the probes of the multimeter to test two points on the PCB board to check the capacitance between the two lines.

Table 3 Lens R and V Table

Requirements	Verifications
<ol style="list-style-type: none"> 1. The lens can adjust focus to maximum and minimum values 2. The lens can adjust aperture to maximum to minimum values 	<ol style="list-style-type: none"> 1. Run the Verilog code on FPGA 2. Run Python code 3. Observe if the camera focuses or changes aperture according to the user input
<ol style="list-style-type: none"> 3. The lens shutter opens within one second of running code <p>The lens is able to execute the commands with less than 0.5s of latency.</p>	<ol style="list-style-type: none"> 1. Run the Verilog code on FPGA 2. Run Python code 3. Observe if the camera properly opens its shutter. Successful observation will be enough to pass this test
<ol style="list-style-type: none"> 4. All the instructions are executed reliably over 6 hours 	<ol style="list-style-type: none"> 1. Run the Verilog code on FPGA 2. Run Python code 3. Check the correct operation after running the program for 6 hours.

Table 4 Program R and V Table

Requirements	Verifications
The machine has to run for 6 hours	<ol style="list-style-type: none">1. Run the program on the PC for 6 hours.2. Check power usage, memory usage, and other metrics that a computer runs on.
The Python code has to be implemented correctly	<ol style="list-style-type: none">1. Run Python code2. Check visually to see if lens responds to commands
The machine has to power and run programs on the FPGA.	<ol style="list-style-type: none">1. Write a testbench to check if the Verilog code that we wrote is correct.2. Compile the program and run the program with testbench.3. Check the waveforms to see if we are going through the correct states and send correct signals to the lens4. Measure power drawn by the FPGA as the program is running to ensure it is within limits.

Appendix G Tolerance Analysis

G.1 FPGA

As our project involves a lot of programming, we need to ensure the reliability of the operation even after running the program for an extended period of time. An operation would be multiple hours in length, if not more and so we must make sure there are no unexpected crashes or errors. By doing this, we minimize the chance of ineffective tumor removal as discussed in the Introduction. In particular, the Finite State Machine, which is mainly responsible for the implementation of the various instructions put into the camera, has to continuously run for as long as the operations take, and we need to ensure that nothing breaks down in the middle of its functions.

This will be done by simulating the Finite State Machine to make sure that it steps through all the states effectively. Furthermore, the ability to simulate will also allow us to see the entire operating cycle of the Finite State Machine which will allow us to more effectively diagnose and solve any potential errors or faulty states.

G.2 PCBs

There will be many factors that will need to be considered across both PCBs. The rigid-flex-rigid PCB would have more constraints due to its placement in the lens mount.

For the stackup, there are several factors to consider. The flex layers need to be properly extended throughout all the regions so that the signals can reach the bottom of the lens mount. The other important consideration is the rigid sections having to be at most 1 mm, otherwise the PCB would not fit in the mount.

For the routing, the trace width for the camera signals has to be at least 8 mils and 20 mils for any power signals such as the 6V signal or VDD (Logical High) so that the impedance is lowered sufficiently such that it does not affect signal transport. Furthermore, the trace separation had to be at least 8 mils in order to make sure that there is no possibility of short-circuits between the traces. When utilizing the vias, the via diameter has to be at least 10 mils in order for safe transport between the layers.

Appendix H Cost of Parts Used

Table 5 The cost of components

Component	Manufacturer	Quantity	Price
XEM7310-A75	Opal Kelly	1	\$569.95
XEM7310-A200	Opal Kelly	1	\$734.95
BRK7010	Opal Kelly	1	\$49.95
Camera lens	Canon	1	\$199.99
857-10-010-10-002000	Milli Max	1	\$3.81
TF13BSA-SERIES (800)	Hirose	1	\$1.55
Rigid-Flex-Rigid PCB Fabrication Cost	PCBWay	1	\$686.35
Flex PCB Fabrication Cost	PCBWay	1	\$200.97
Total Parts Cost = \$2,447.52			

Appendix I Project Schedule

Table 6 Schedule Table

Week	Task	Person
Oct 3-7	<ol style="list-style-type: none"> 1. Start reviewing project equipment and documentation. 2. Disassemble camera to analyze lens ports for PCB design. 3. Start PCB ideation and design in preparation for ordering. 4. Start working on FSM and run through reviews with BioSensors lab staff 	Kevin: 1, 2 Jihun: 1, 3, 4 Sid: 1, 3, 4
Oct 10-14	<ol style="list-style-type: none"> 1. Oct 11th deadline to submit orders. 2. Continue working on FSM. 3. Complete Team Evaluations. 4. Continue working on PCB 	Kevin: 1, 3 Jihun: 2, 3 Sid: 2, 3
Oct 17-21	<ol style="list-style-type: none"> 1. Implement PCB (if it arrives) into the system to test functionality and synergy. 2. Simultaneously work on the flat PCB as a backup. 3. Start Python code to 	Kevin: 1 Jihun: 3 Sid: 2

	control user features and experiment with separate components.	
Oct 24-28	1. Integrate all systems together for an initial prototype of the final camera system.	Kevin: 1 Jihun: 1 Sid: 1
Oct 31-Nov 4	1. Nov 1st second deadline to submit orders if needed. 2. Finalize initial design details and refinement iterations.	Kevin: 1, 2 Jihun: 1, 2 Sid: 1, 2
Nov 7-11	1. End of initial design process.	Kevin: 1 Jihun: 1 Sid: 1
Nov 14-18	1. Mock demo. 2. Improve on the design based on the feedback from the mock demo.	Kevin: 1, 2 Jihun: 1, 2 Sid: 1, 2
Nov 28-Dec 2	1. Final demo.	Kevin: 1 Jihun: 1 Sid: 1
Dec 5-9	1. Final presentations + papers	Kevin: 1 Jihun: 1 Sid: 1

Appendix J Rigid-Flex-Rigid PCB Figures

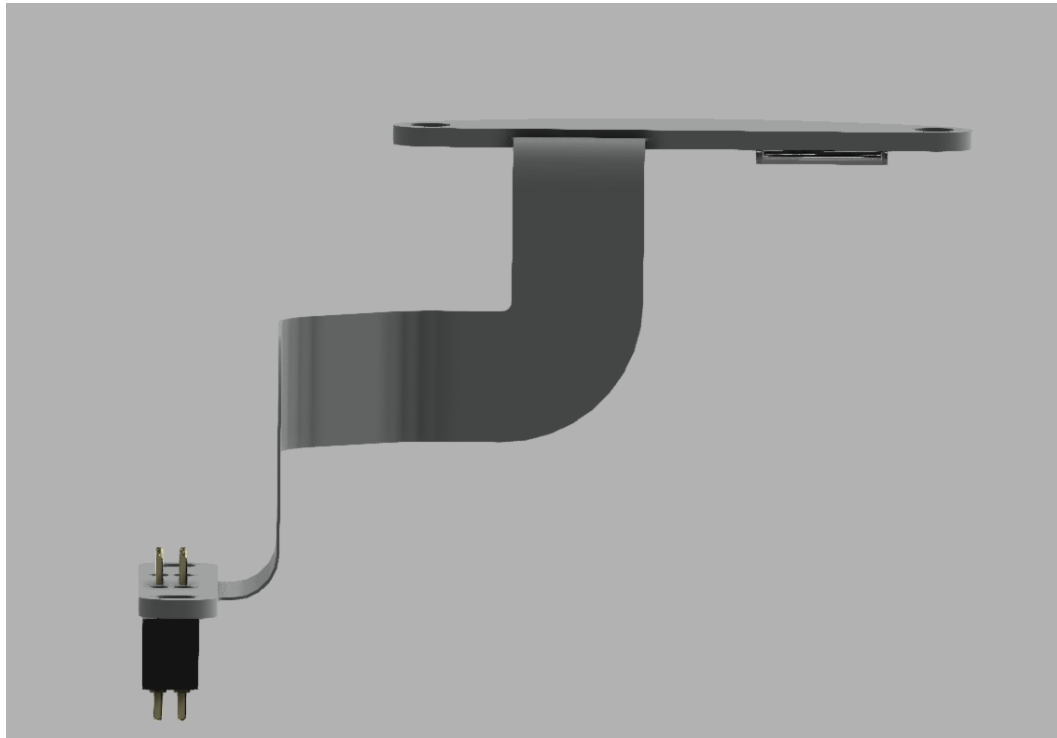


Figure 19: CAD Model for Rigid-Flex-Rigid PCB

Figure 19 is the 3D Model showing the Rigid-Flex-Rigid PCB and the bends that the flex region will have to navigate the lens mount. Furthermore, the contact pins can be seen protruding from the second section. This will be replaced with contact pads in our PCB.

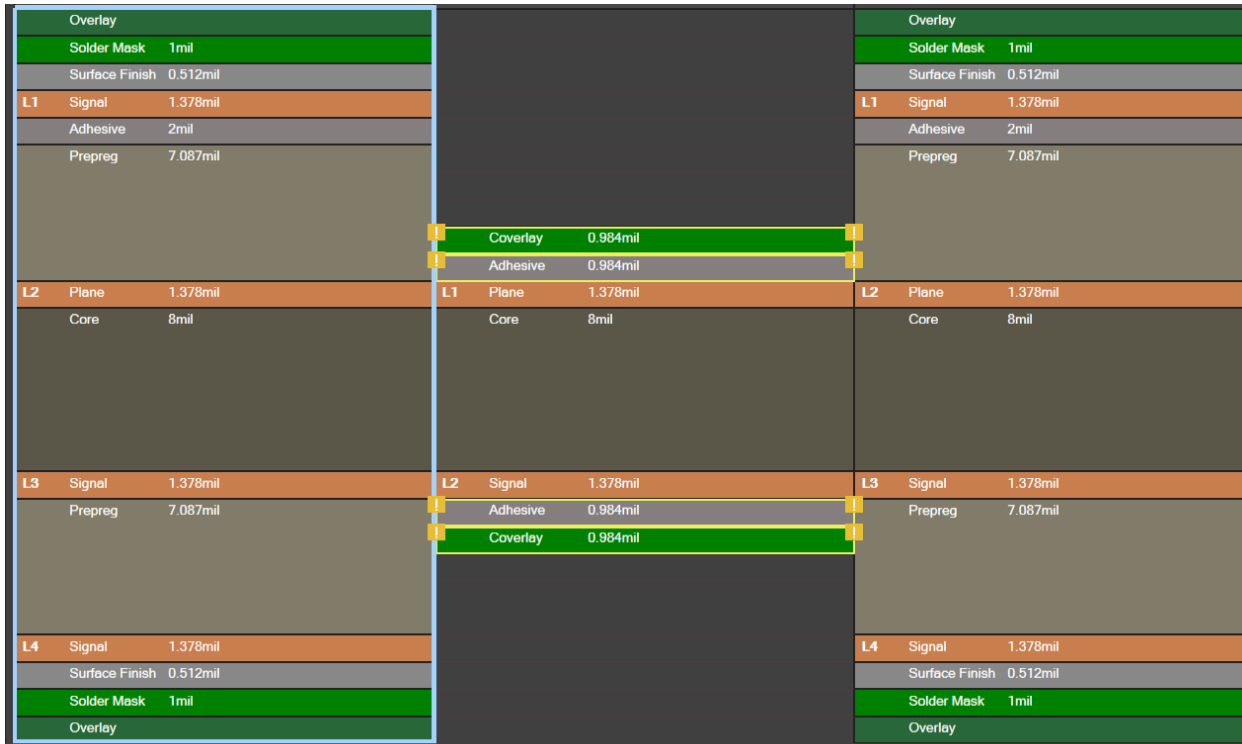


Figure 20: Layer Stackup for Rigid-Flex-Rigid PCB

Figure 20 shows the layer stackup for our Rigid-Flex-Rigid PCB. It has four rigid layers and two flex layers, which will emerge from the rigid layers and carry the signals using vias. The warning labels in the figure can be ignored as adhesive and coverlay will not be interacting with Prepreg layer in actual design, but in stackup, Altium cannot see the difference. The Thickness of the rigid section is 0.8 mm and the thickness of the flex section is 0.37 mm.

The signals will be transported using signal layers and it will begin on Layer 1, with the signals coming from the gold flex PCB. Using vias, it will be connected to Layer 3 as Layer 2 serves as a ground reference and flows to the second rigid section which will emerge back to Layer 4 using another set of vias.

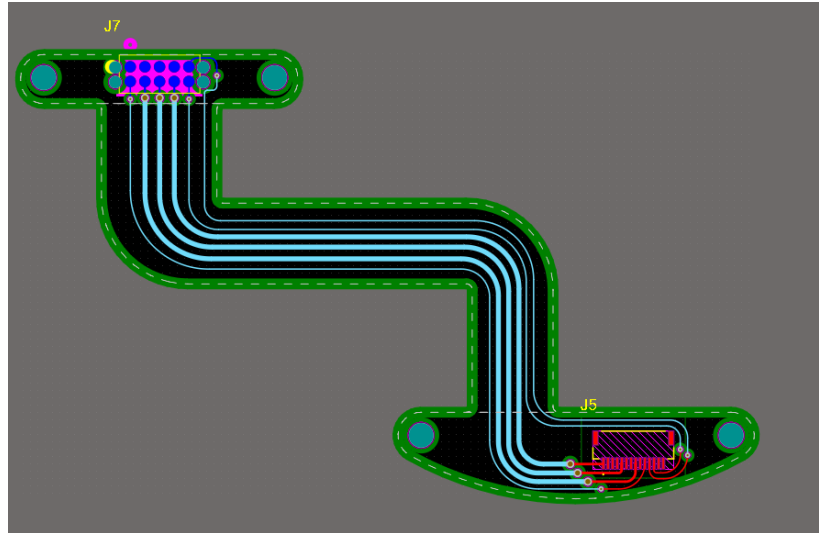


Figure 21: Routing Schematic for Rigid-Flex-Rigid PCB

Figure 21 shows the routing schematic of the Rigid-Flex-Rigid PCB. The contact pads are seen in the top left with the footprint having a pink and blue color scheme. The blue pads indicate that the pads are on the bottom layer (Layer 4), which makes sense since they would have to press down on the contact pins emerging from the lens mount. The light blue wires represent Layer 3, the signal layer in the flex region, responsible for transporting the signals through the lens mount.

As Figure 21 shows, some traces have different widths. The widths of the lens signals (DCL, DLC, LCLK) are 8 mils wide while the widths of the power signals (VDD, VBAT and D-GND) are 20 mils wide. This design choice was made since the lens signals would not draw much current and thus, for signal speed purposes, can be kept to be narrow. However, the power signals would draw too much current for an 8 mil trace and would risk breaking the PCB, due to the large impedance that the power signals would have to encounter. In order to reduce it, the traces are made wider.

The transportation of signals is made possible by vias, seen in the schematic as purple pads. Another visual aid is the traces suddenly changing colors: This indicates that the signals have changed layers. It was intentional to keep the vias directly away from any components if we could, since it is a hole in the PCB and could result in current leaking and causing short-circuit problems.

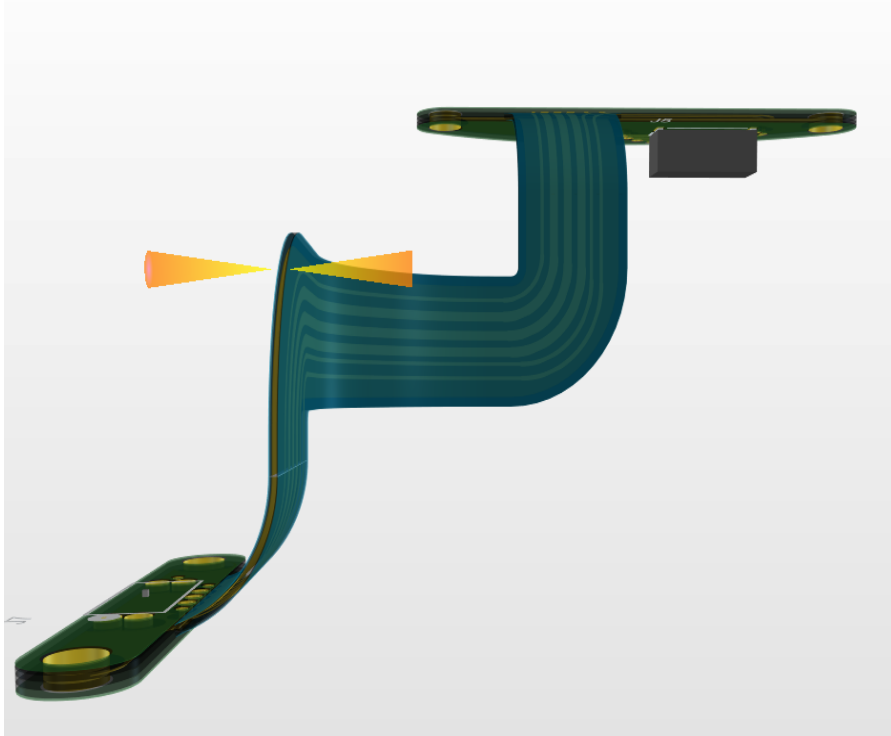
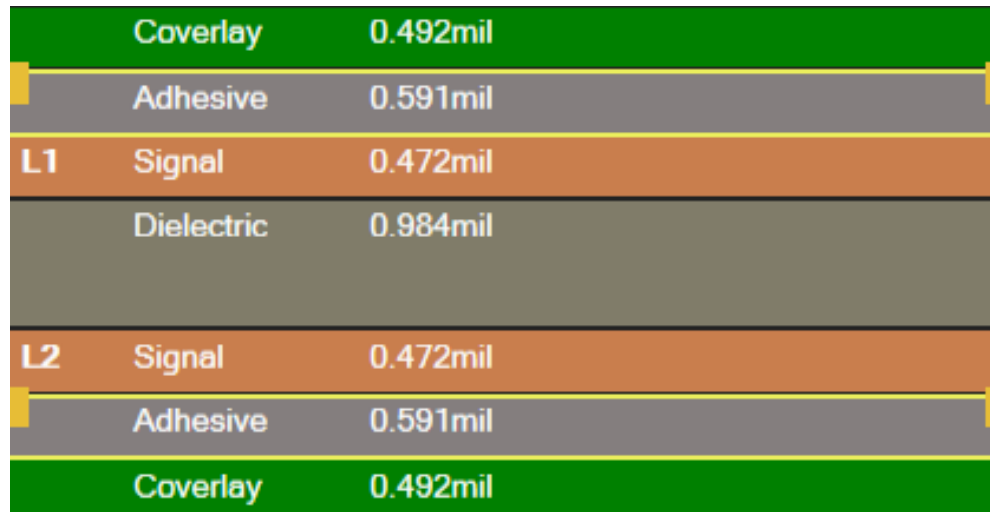


Figure 22: 3D View of Rigid-Flex-Rigid PCB

Figure 22 shows the final 3D view of the Rigid-Flex-Rigid PCB. In the top right corner, we can see the connector used to connect to the gold flex PCB. In the flex region (Coloured blue), we can see the signals being routed to the contact pads and the bends, they have to go through to navigate the lens mount. Finally, while the contact pads are themselves not visible, their outline can be in the bottom left corner. The signals are now at the base of the lens mount and need to be transported to the FPGA inside the camera body.

Appendix K Flex PCB Figures



	Coverlay	0.492mil
	Adhesive	0.591mil
L1	Signal	0.472mil
	Dielectric	0.984mil
L2	Signal	0.472mil
	Adhesive	0.591mil
	Coverlay	0.492mil

Figure 23: Layer Stack Up of Flex PCB

Figure 23 shows The layer stack up for the flex PCB. The materials used are virtually identical to the flex region in the Rigid-Flex-Rigid, but there are a few differences. Because this is a fully flex PCB, it is only composed of two flex layers. The thickness is also much lower, with it being 0.13 mm, compared to the flex region being 0.37 mm, and the reason that is the case, is that for a rigid-flex-rigid PCB, the flex layers needed to be consistent throughout all regions. Normally, a flex PCB has a much thinner core but if attached to a rigid section, like in our case, has to have a thicker core to maintain the layer consistency. With this constraint removed for this PCB, the thickness drops dramatically.

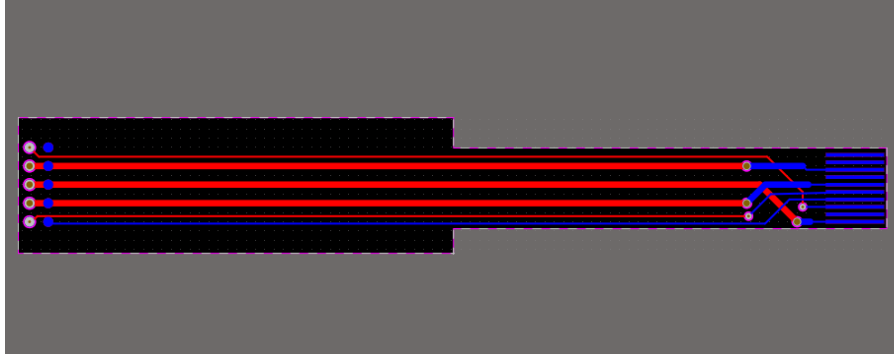


Figure 24: Routing Schematic for Flex PCB

Figure 24 shows the routing schematic for the flex PCB. On the left, we have the etched contact pads, which will latch onto the contact pins once the camera body and lens mount are attached. The pads have been designed to have the same specifications as the contact pads used in the rigid-flex-rigid and the reason why we did not just surface mount the same component in this case, was that we would have to dramatically increase the thickness of the flex PCB. A constraint that was present was that the FPGA connector has a thickness of 0.5 mm and while the thickness of the flex PCB is 0.13 mm, surface mounting the component would not guarantee, we can fulfill this constraint.

The same trace requirements are implemented as in the rigid-flex-rigid and the vias are used in a similar way to transport the signals. In the figure, the top layer (In red) was used to transport the signals to prevent short-circuits which would have been caused if only the bottom layer (In blue and where the contact pads and beveled edge are) was used.

The other side of the connector has a beveled edge, done to ensure proper insertion into the FPGA connector. The width of the beveled edge matches the width of the FPGA connector and that is why the PCB suddenly reduces its width. The edge is electroplated with gold to aid in conduction and ensure proper signal transport.

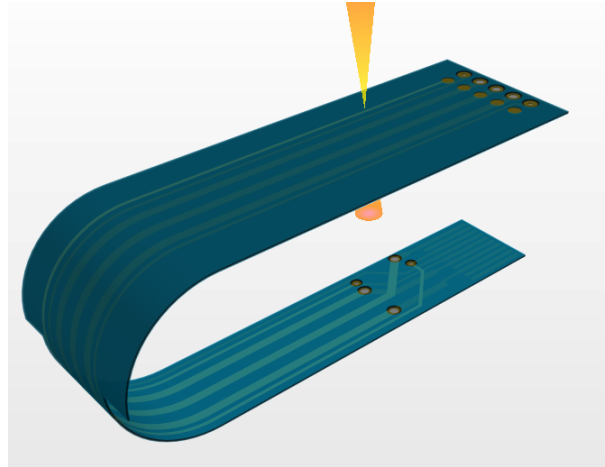


Figure 25: 3D View of Flex PCB

Figure 25 shows the final 3D view of the flex PCB with the contact pads being visible as the gold circles. As seen in the figure, the bend angle is 180 degrees due to the FPGA connector being directly beneath the contact area on the lens mount. Another design factor was that the PCB had to be long enough to maintain proper alignment between the FPGA connector and the contact area. Using CAD software, we determined that the PCB would have to be at least 47 mm. This measurement takes into account the arc of the PCB and thanks to the flex PCB's ability to bend ensures that the PCB can still function properly even if it exceeds the measurement. The design reason as to why the length would have to be exceeded is that the contact pins and the contact pads may slide against each other, causing subtle shifts in the PCB. If the PCB was fabricated with the exact length, it may slide out and sever the connection completely. That is why, the length of the PCB is actually 59 mm, with the extra 12 mm given as room for error.