**EE 326: Electronic System Design  
Project Report**

Author: Michael Jenz Collaborator: Michael Kim

**INTRODUCTION**

The project for Electrical Engineering 326: Electronic System Design was to build a web camera from scratch. A user of this web camera should be able to connect to the video stream via a website, hosted locally on the ESP32 chip, and see a stream of images taken from the camera. The purpose of this project is to learn about the steps involved in the design of an electronic system like a webcam and become capable of taking on a project like this with less guidance in the future. Therefore, the expected performance of this web camera is not expected to be fast.

The design process for this project spanned a quarter at Northwestern totalling 10 weeks, with progress goals for each week. Each week, we would learn a new aspect of the content in class and then work on implementing it into our project. The design process began with creating breakout boards for the chips being used, creating a new PCB design, writing the firmware, creating the website, and designing an enclosure for the web camera. We will dive deep into the design process in the next section.

The chips that we were to use were chosen by the professor. The microcontroller unit (MCU) that we would use is the SAM4S8B from Microchip. This chip is the brain of the system and helps interface the camera peripheral with the wifi peripheral to transfer images to the website. The camera we used was the OV2640 and we used the ESP32 chip to host an internet connection. These three main components were what made up the system. In this project, we have integrated them together using various communication protocols to take pictures, transfer them, and upload them to our website to make a functioning web camera.

**DESIGN PROCESS**

*PCB Design:*

One of the earlier tasks in this project was to design a new printed circuit board (PCB) for the web camera project. In essence, this would reduce the rats nest of wires that made up our breakout board setup into a well packaged final design. The breakout board design process that happened before this was an important step that helped us plan the pin connections that we would need for the PCB and acted as a prototyping platform as we waited for our PCB order to arrive from the manufacturer. We used Eagle from Autodesk for our PCB design, as it was recommended by the professor.

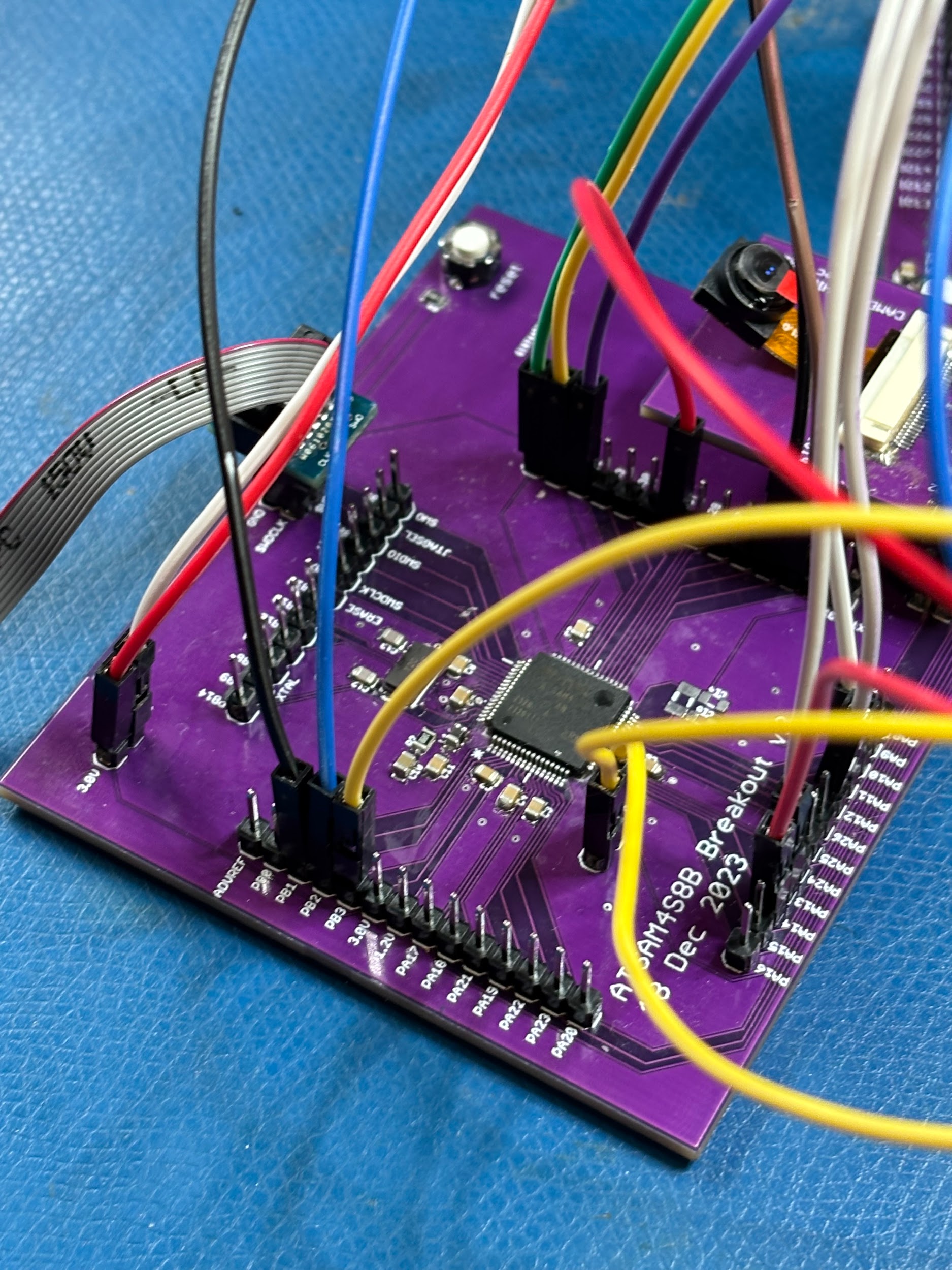
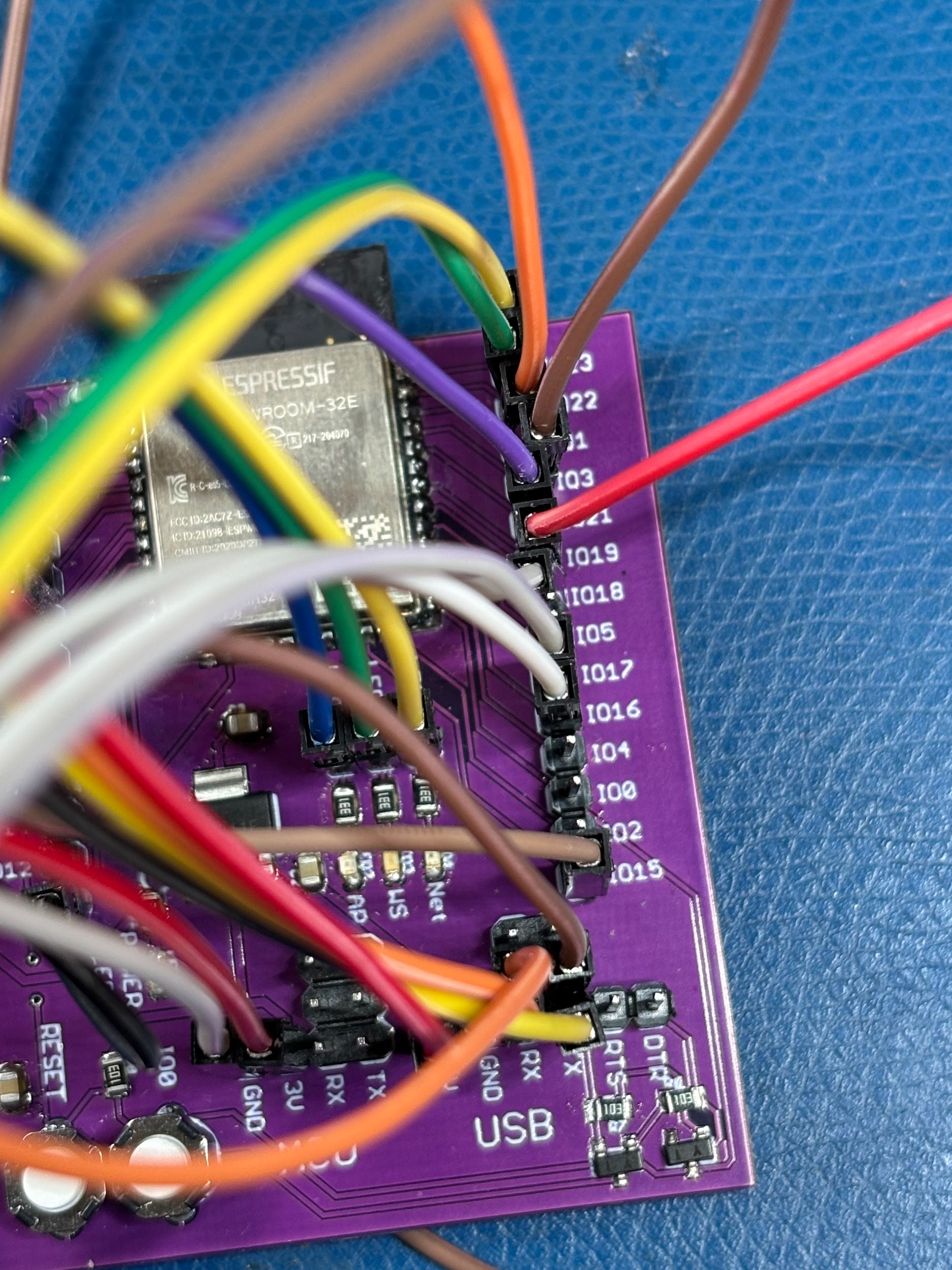


Figure 1: Breakout boards

The general process for designing a PCB is to first create a schematic diagram, and then to map out the actual wiring on the board. The schematic is a symbolic representation of the connections made between all components and chips on the board. In order to create the schematic we needed to consult the breakout board schematics provided by the professor, as well as our wiring between the breakout boards themselves that we designed.

First, we made sure to copy all the connections made on our breakout boards already. For example, the breakout board camera setup does not require any wiring by us because the pin setup matches the camera breakout board. This allows us to use the MCU breakout board as a reference to make sure that we are matching the pins appropriately. See Figure 2 below which shows the camera and MCU breakout board setup.

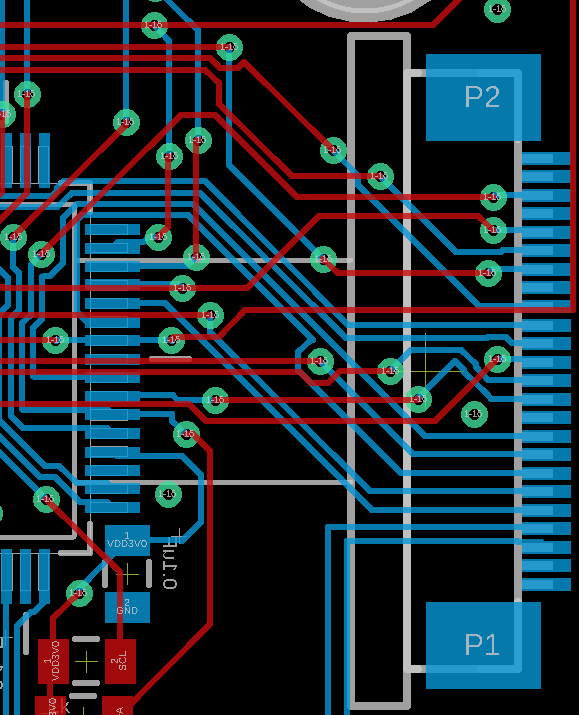
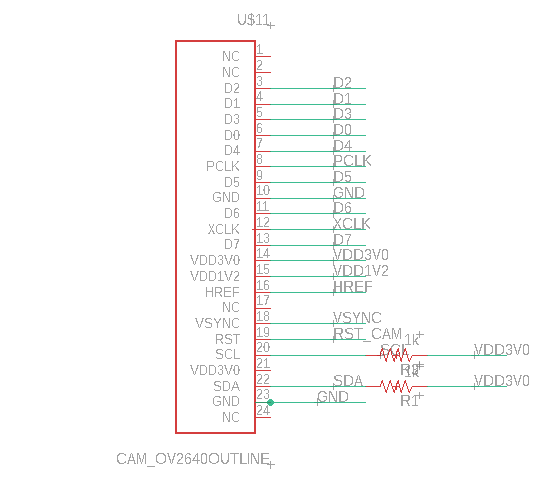


Figure 2: The camera MCU interface on Eagle Schematic and the PCB

Another point of interest that we made sure to address while transferring the designs from breakout to the PCB is the connection to power the core, the embedded memory and peripherals. This particular system requires a very clean 1.2V input. We are using an external 3V buck voltage regulator to convert 5V to 3V, and we found that the MCU has an internal voltage divider to convert to 1.2V. In the breakout board, this was connected by a wire, however, we decided to connect this directly on the PCB. This is an example of one of the few, if any, changes we made from the breakout board design.

Next, we needed to add all of the connections that we had added with wires between the breakout boards. Since it can be troublesome to look at the mess of wires and figure out which pins are connected where, we made sure to write down all of our connections on a document after we completed that step of the design. This made the step of adding these connections to the schematic very easy. Most of these connections involved the communication lines between the MCU and ESP32, such as UART and SPI lines. We also connected four general purpose in/out pins (GPIO) between the two main chips to facilitate high/low communication.

The next step in creating the PCB is to design the board layout. This process is potentially much more difficult than creating the schematic diagram. Now, all the symbolic connections that were made previously need to become real wires on the board. Since we were constrained to a board size of 50mm x 50mm, we needed to be wise about the placement of chips and the wiring. Additionally, it is important to consider the user interface aspect of PCB design, since we will be interacting and using the PCB, we should think about how we will want the pins and boards to be configured to make later use as easy as possible. This consideration affected most of the buttons, the pins for flashing and UART communication, the barrel jack, and the LEDs. We decided to keep most of these elements on the back of the PCB (the side without the camera) so that we could easily access them since we anticipated our camera side being flush with the enclosure.

From here, there were no clear guidelines on how to design the circuit layout. For this reason, this process was accomplished with a lot of trial and error. For example, many of the pins on the MCU that connect to the camera are on the same side of the chip. Therefore, it makes sense to place these two sides close to each other. However, this may make many of the pins that are connected to the ESP32 face in a direction that is not close to where we placed the ESP32 chip. Competing interests, like the one I just described, are what need balancing when designing a PCB. I, Michael Jenz, spent a lot of time trying to figure out the best way to connect pins by manually drawing traces. This took a lot of time to find a configuration that worked, especially in the area around the MCU because there are so many pin connections there. Michael Kim on the other hand used the autorouter, which was able to complete the task much more efficiently. We would find out later that the autorouter performed well and did not cause any problems on its own. The autorouter method is so much faster, however, if the PCB is not laid out properly or if there are incorrect connections you will not have the chance of catching them if you use the autorouter.

Once all the connections are made, the board is almost ready for manufacturing. The last important aspect to PCB design is the labeling that will go on the silkscreen layer of the PCB. We made sure to include capacitor and resistor values since they would help while soldering the board. That brings me to the final step of PCB design: soldering. Once the board has been ordered and the shipment received, the last step is to solder the components onto the board. This step is straightforward, however, debugging the PCB is a difficult task and will be discussed further in the challenges section.

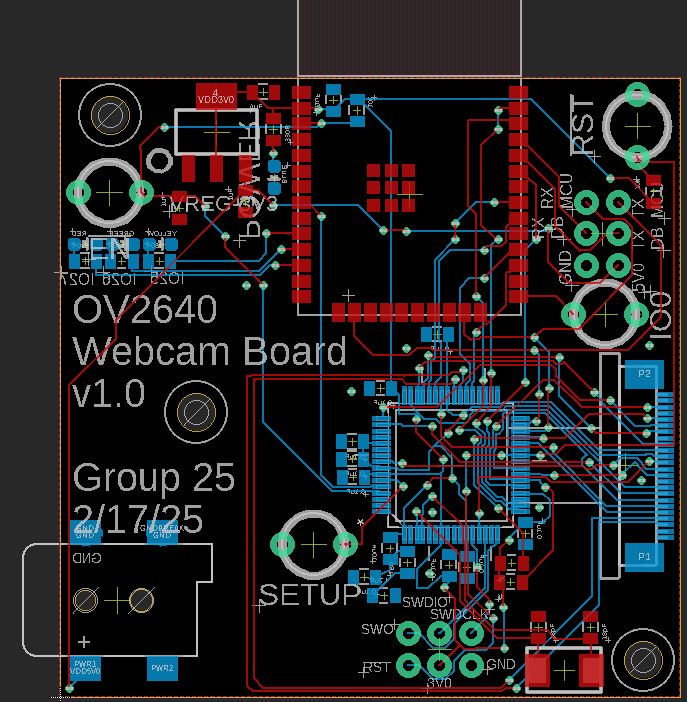


Figure 3: Completed printed circuit board

*C Code:*

The code or firmware for this project might have been the most difficult portion of the project. To write the code we used the Microchip Studio application since it has the Advanced Software Framework for our SAM type MCU. The firmware we wrote was specifically for the MCU, not for the ESP32 (this code was provided). The goal of the firmware is to take pictures using the camera peripheral, check to make sure that they are valid pictures, and then send them over to the ESP32 for uploading to the website. Of course, the actual process is much more complicated and involves a lot of initialization that can make getting the code ready to run difficult. The process for writing the firmware mirrors that of the PCB design where we relied heavily on example code either from the existing SAM examples, or from lectures. With this, we were able to breeze through much of the code.

For example, much of the firmware base was initializing GPIO. The real challenge to this task is not learning how to initialize a pin but knowing which pins you needed to initialize. On the MCU there are 64 total pins, and since we are using most of them that is nearly 64 pin definitions that need to be made. This is one of the cases where the example code came in handy. Especially for the camera initialization where there are 15 different pins to match up between the camera and MCU, we made sure to validate our code with that provided in examples.

Since most of the code to initialize peripherals already existed, much of the code that we wrote on our own had to do with the logic of how our system worked together. One function in particular needed to calculate the length of an image. Not only did this help the image send to the ESP32 via SPI communication, it also validated that the image captured from the camera was stored in memory as a valid image. We did this verification by checking for the start and end signature of a JPEG (the compression form we used for the images). Each JPEG starts with a 0xFFD8 and ends with a 0xFFD9. Therefore, if our code is able to identify these hexadecimal encodings in our MCU memory, we know both how long the image is, and that it is a valid image. The way this function works is that it searches through all the memory allocated for images for the start of an image (0xFFD8) until it finds one. It then records the start index, and continues to find the end index. If this expected plan does not go as follows, it returns that there has been a failure, but if not it saves the image length to a global variable to be used elsewhere in the codebase.

Like the PCB design, the most difficult element of writing the firmware was debugging it. We ran into many errors that were difficult to detect, however, we learned how to overcome these challenges and will dive into this more later.

*Website:*  
We designed a website to host the video streamed from our webcam, as well as some other information. The first step in designing this website was to configure the ESP32 wi-fi. After flashing the provided firmware onto the ESP32, we connected to the access point it gave us to connect the chip to eduroam. Then we accessed the IP address and were able to add our custom website files.

Before we designed the website, we drew sketches of each page. The pages include a home page with some information about our webcam, a live webcam page which hosts the video stream, and an ‘about us’ page with some info about the creators. For ease of access we included a navigation bar at the top of each page that allows a user to easily switch between each page, as well as access the edit page for the website.

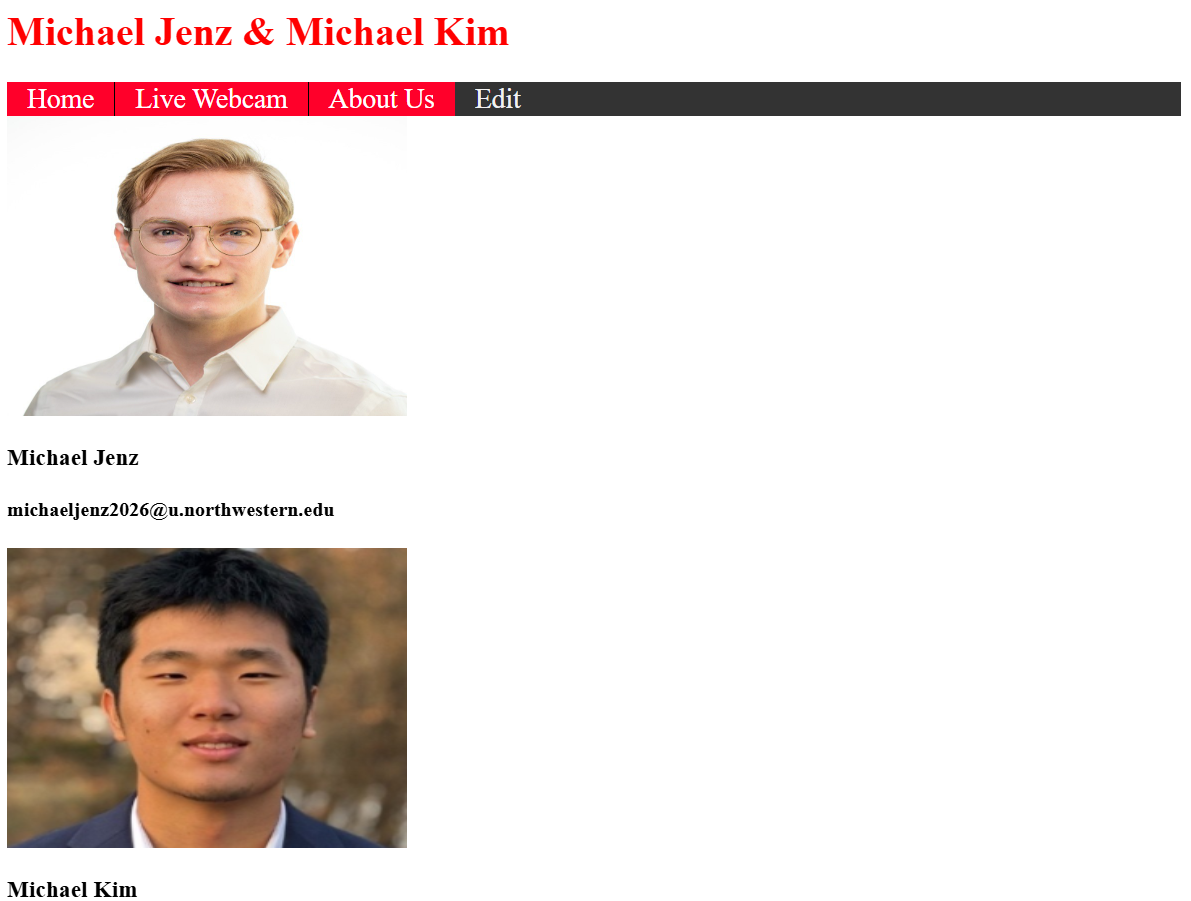
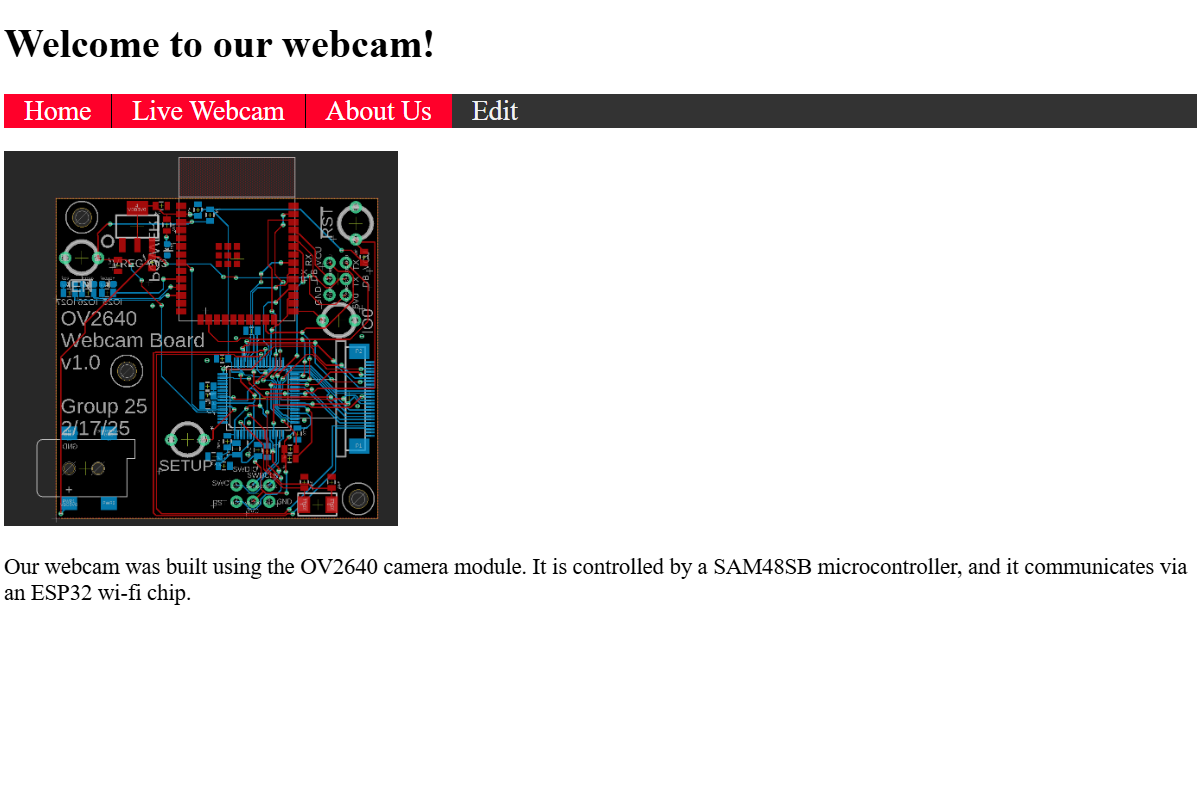


Figure 4: Website pages

We designed each page in html files. We used a tool called Phoenix to design these files. The advantage of using this program is that it provides a live preview of the website as the code is changed, making testing and debugging much easier than having to re-upload a file and connect to the website. The home and about us pages were quite simple, with just text and some images. For these we simply used the paragraph and header tags in html. The webcam page was a little more involved. It included a button to start and stop the webcam, an image where the video stream would appear, and a log to record recent connections and disconnections to the webcam. After the html files were designed, we stylized the pages using css files. We used these files to add new colors and fonts to our website. Finally, we added functions with javascript files. This was definitely the most involved part of the website design. For the live video page, we used functions to configure the websocket, as well as to make the button interactive. When the camera was on and the button was pressed, it would disconnect the video feed. When the camera was off, the button would reconnect to the video feed. We also used javascript to add some simple animations, like changing the color of the background when the button was pressed.

*3D Design:*

One of our final tasks for this project was to design and 3D print an enclosure for our PCB. The goal of this part was to turn our bare PCB into something more polished that can be held and used easily by a user.

The first step for this task was collecting measurements of our PCB. Some of this was done using calipers to measure parts that protruded from the PCB, such as the barrel jack header, programming pins height, button height, etc. The rest of the measurements were taken straight from EAGLE for components that are directly on the PCB, such as LEDS, button location, etc. These measurements were taken with an accuracy of one tenth of a millimeter, which is sufficient accuracy for the size of components we are working with.

Next we designed our enclosure using OnShape, an online computer aided design software. First we added a rectangle to fit the basic shape of our PCB. Knowing that our PCB is 50x50 millimeters, we initially tried making the rectangle 53x53 millimeters, with one inch wall and base thickness, which would leave one millimeter of wiggle room in each direction to squeeze the PCB in. Next we added cutouts for the barrel jack and the antenna of the ESP32. Because the barrel jack and antenna both hang off the PCB, we knew that trying to squeeze both of them into their cutouts would make it very difficult to get the PCB into the printed enclosure. Therefore, we added five more millimeters of coverage on the antenna side so that only the barrel jack would have to be squeezed in. This can be seen below in Figure 3. Next, we added support pillars inside the box to hold the PCB. Our PCB has three mounting holes, so we added three support columns below each. These pillars were each 6x6 millimeters with a 3 millimeter diameter hole to hold the screw. Finally, we added cutouts on the bottom of the board to expose the camera as well as the LEDs.

We also designed a lid to close the box so that the PCB is not exposed in the final product. This lid was also one millimeter in thickness. It included 1.5 millimeter holes for each of the four pushbuttons, so that they can be accessed with pins but are not readily accessible from the outside. It also included two key-shaped mounting holes that could be used to hold the device on a wall with the camera oriented outward. The lid has two 3 millimeter holes in opposite corners that are meant to close the enclosure when screwed together. The lid and the box were put together using the Assembly tool in OnShape. We used a revolute mate so that the lid could spin around an axis if only one of the screws was attached. This allows it to be temporarily opened and closed without having to unscrew and rescrew every time. We added two support pillars in our box to hold the screws that go through the lid. One support pillar went from top to bottom, while the other was in the way of the PCB so had to be cut short.

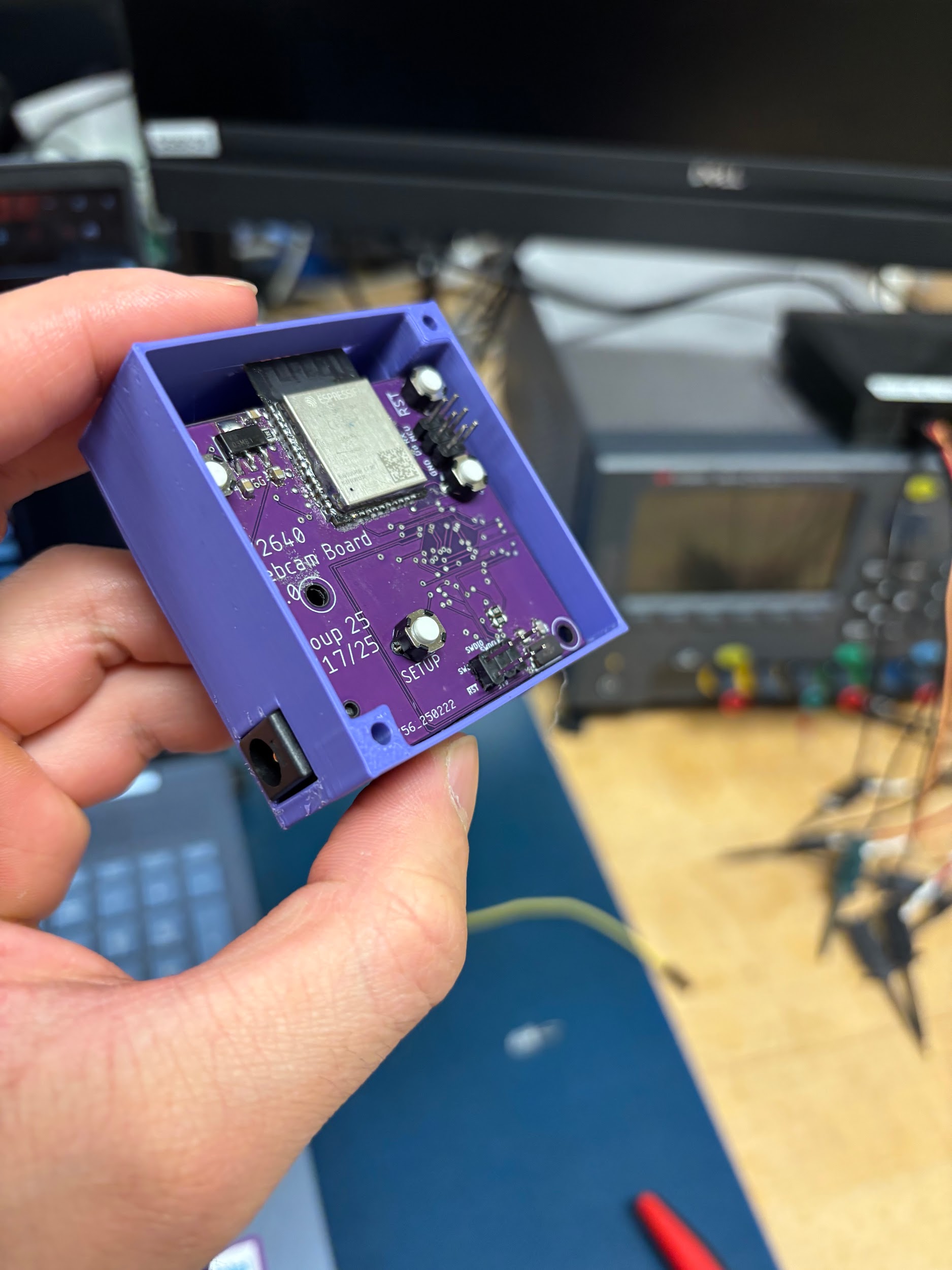


Figure 5: PCB resting inside of 3D printed enclosure

We printed a first draft of our enclosure and tested it with an extra PCB that we had soldered that we were not using. Supports were used to hold up the one unsupported pillar in the corner, as well as the cutouts for the barrel jack and the antenna. The first thing we noticed was that the base was warped in one of the corners. We fixed this by increasing the base thickness from 1 to 2 millimeters for the final design. As expected, getting the PCB into the enclosure was difficult. Because of the one screw holder that was cut short, the PCB was wedged between two different supports from the top and the bottom. Another problem was that the button holes were measured improperly, without taking into account the one millimeter thickness of the walls. This was easily fixed for the final design.

After making these adjustments, we reprinted the final version of the enclosure. Increasing the base thickness effectively removed any warping problems that we had. The one thing that we might change if we were to redesign the enclosure would be to make it slightly bigger. Because of the tight fit, it was very difficult to get the PCB back out of the enclosure without damaging the enclosure itself. Therefore, we were unable to test our board inside the enclosure until we had verified that it was fully functional first.

*What were some challenges you faced? What did you learn from them?*

We faced many challenges during the design process, as we have hinted at in the discussion above. In this section, we will discuss some examples of these challenges and what we have learned from them.

Board construction issues were a problem for us from the very beginning. When we were designing and verifying our breakout boards, we were having trouble getting a valid image over USART. After fixing some potential soldering issues, the problem persisted. While we were trying to debug with the logic analyzers, we realized that the tests would work when the connections were broken out to the breadboard, but not if the logic analyzer was also plugged into the breadboard. From this, we discovered that using two wires to connect one of the MCU pins to the ESP32 pins allowed us to get a valid image, while using only one wire did not. We never found out what was really causing this issue, though we assumed that it was just a signal integrity problem. This was a good example of how touchy electrical connections can be and the importance of testing connections that could be causing issues.

For our PCB design, the most difficult challenges we faced were after the design had been created and we worked to get our firmware working on the board. This process of debugging the hardware was difficult because with this new board it was not clear what parts worked and what did not. Although the soldering process went well initially, we ran into an issue where the camera initialization would not complete. This was despite all of the clock pins working, the I2C pins running as expected, and several attempts to identify the issue with the board with the professor. In the end, we needed to restart soldering from scratch on a new board. Despite the fact that we were not able to identify the exact issue with our board, through the process of logically walking through the possible issues that could be stopping board initialization, we got a lot of practice in thinking critically about the issues we were having, rather than looking for help from the TA’s.

Similar to the process of debugging our PCB, writing the firmware involved a lot of debugging. The biggest mistake we made during this process was incorrectly initializing the camera to capture every other bit. This was an artifact of the camera example code that we used, since they were using a different image communication protocol which allowed them to capture grayscale images by not collecting every other byte. This does not work when transferring JPEG images, and was resulting in the image missing the 0xFFD8 and 0xFFD9 bytes. We were only able to figure this error out after the professor pointed it out to us. I think this error emphasizes the importance of double checking any code that you are copying from examples, and to pay careful attention during lecture when this common mistake is pointed out. Many of the other issues we were having were a result of this bug. Our find length function was not operating correctly, because often the start or end byte was missing from memory. Of course, our function was written correctly; it was just the bug mentioned earlier that caused this. This is further evidence to suggest that code should be copied with caution, since it will make code that is written properly seem like it is not working.

*What would you change if you started over?*

The first thing I would change if we started over is to start working on the firmware earlier. We did not give ourselves enough time to finish this task before the 100% deadline, and even so we finished the night of the 90% deadline. If we had started earlier we would have been able to front load more work that was delayed until finals week. This put us in the precarious position of resoldering our entire board the night that the project was due. I believe that this could have been avoided if we had started working on the firmware earlier and allowed ourselves more time later in the quarter to work on debugging the PCB and getting the code up and running on it.

Our approach in general worked fairly well. I believe our documentation that we kept helped with the later design steps of creating the PCB since it was easier to reference pins. However, looking back I would surely keep more documentation for our design decisions. Since so much happens in the class, it can be difficult to remember why, for example, you chose to put a GPIO pin at a certain location. If we recorded more information for our design rationale it would act as a helpful reminder for what we have already learned in the design process and prevent us from making mistakes. Another approach change that we would make to the firmware design process is to think with the bigger picture in mind at all times. It was easy at the start to get caught up in copying code from the example files without thinking about what the code was doing or what it meant for the larger firmware design. If we had instead ‘white boarded’ how each of the different aspects of the code work in tandem, as well as all the different pins that we would need to initialize, it would have provided much more context to the example code that we used for reference. Furthermore, this big picture thinking is an approach that can help while working on any aspect of this project from the PCB to the 3D design.

For our actual design, we would probably tweak our PCB design in a few ways. First, we would place the bypass capacitors farther from their respective pins. In the PCB design that we used, these capacitors were placed as close as possible to the pins which made soldering difficult and made the process of debugging the board more difficult as well. We would also have moved the barrel jack to the back side of the board, and we would have been more intentional about placing vias away from pins. Additionally, it would have been better to have the correct TX and RX connections between the MCU and ESP32. We forgot to flip this connection, so we had to go into the ESP32 firmware and change the pins. This mistake was not catastrophic by any means, but correcting it would have made the design process that much easier.

More generally, understanding the finished product and the role that each part plays in it before starting the design would have been helpful. For example, if we had been more familiar with the dataflow from the camera to the MCU to the ESP32 before starting our firmware, the design of our code could have been more intuitive and quicker. Additionally, when we were first putting together our breakout boards, we didn’t really understand what many of the pin names even meant or what their functions were. This made debugging a lot more difficult, because if there was a problem with camera initialization for example, we didn’t know to check the I2C connections with the camera. Being familiar with the overall operation and dataflow of the final webcam would have helped us debug and problem solve more efficiently.

.

TEAMWORK (individual)

*Did you contribute fairly to your team effort?*  *Do you think that you did an unfair amount of*

*work, either too much or too little, as compared to your partner?*

I think that we both contributed fairly to our team effort, or in other words we did an equitable amount of work. Our team divided up the work more on a task by task basis rather than working together on each task. Of course, when we got stuck on something or needed help, it was really helpful to have the other person looking over our shoulder. But for tasks such as soldering the PCB, or even writing sections of the firmware, it made sense to have the same group member take those tasks from start to finish. Given this approach, we made sure to split the tasks up by the amount of work that each of them entailed. For example, I did much of the firmware writing during the last few weeks of the project. While I was writing the firmware, my partner worked on the other tasks that were due at the same time. I want to be clear, I did not complete it just on my own – I could not have done it without Michael’s help. But, this structure allowed us to keep up with the deadlines that were set for this class.

LEARNING (individual)

*Why did you take this class and what did you hope to learn? Did you learn as much as you hoped to in this class? Were there any topics that you hoped to learn but did not?*

I took Electronic System Design because I enjoyed the mechatronics sequence with Prof. Nick Marchuck and was interested in getting more experience with embedded systems. This class is clear in what it offers, a step by step walkthrough on an embedded project with the goal of giving you the skills necessary to take on a project like this yourself. This sounded exciting to me, and now having taken it I can see that the vision for this class is well thought out and well executed.  
  
Now at the end of the course, I see that each of the tasks we spent a week on in this class could be an entire class. For example, writing industry standard firmware for the SAM4S8B was extremely difficult and I could not have done it without the guidance/directions provided by Ilya. Furthermore, there were elements of this project that were kept a bit of a mystery to us, such as the websocket running on the ESP32. I am not suggesting that the course structure should be changed, but rather there are still deeper elements of these systems I have yet to get experience with. In essence, it is one thing to follow directions, and it is another to ideate and execute a plan on your own from start to finish. I feel so much more knowledgeable in this field having taken the class, but I feel left with so many more questions on where to go from here. Given that, I am excited to continue my studies in the next class in this sequence.

Like I mentioned before, I am curious about the methods involved in running a websocket on the ESP32, and also how the firmware was written for that part of the project. I cannot speak very well about what I want to learn about this aspect of the project, since we were not given too much insight into the inner workings of the ESP32 firmware. Even so, wireless connection to other devices is such a commonplace feature on devices today that more exploration into these types of communication protocols would be an interesting addition to the class.