ECE 395
Fall 2023
Final Project

Digital Target System

Andrew Sherwin, Jalen Chen, Minjun Kim
December 12, 2023
Professor Casey Smith

Table of Contents

Introduction

Our initial idea was that we were going to create a project that was based on a target system that would use some type of sensor to electronically read an impact. This target would be able to be hit many times and detect the hit of the pellets. The sensors would be ultrasonic which would detect the pellet and where it hits. We would have multiple of these in a grid-like array which would allow us to figure out where exactly the object hit. This precision would then be indicated using an LED either physically or set as a display on the computer screen. We later determined that it would be easier to use a ball to hit the object, instead of an airsoft gun because these objects are prohibited in the room. Our block diagram is listed below where we have the initial first 2 revisions. The sensors will be in a grid-like formation and should be able to directly detect objects hitting it. Our initial design and block diagram will be listed below.

Summary of Operation

The final design works like this: First, the circuit will work by having the PCB inside a 3D-printed board. This material will be several centimeters thick to withstand the impact of the object hitting it. Then we need to plug in the barrel jack connector to connect the project to power, which will allow us to read the inputs from the ball. After that, we need the UART pin to be able to read the inputs to our microcontroller and to be able to use our software to sync with the hardware. First, the board will be hit with the ball, which will change the resistance of the sensor and allow a signal to travel through to an op-amp. The sensor is connected to 6.6 V on one input side, and the output we set will be connected to the op-amp explained above. This op-amp is bound by resistors to have a certain amplification to carry our signal across. This will then be tied into an inverter, because the resulting signal was negative. After the signal is inverted, we will tie it back to the UART, reading and writing to the microcontroller's GPIO port. There will be separate ports for each of the sensors. Our software will then read it, displaying a hit specific to the force sensor on the Putty system on the monitor, which was officially programmed using C in the STM32 Cube IDE.

Block Diagram

Our block diagram is listed in *Figure 1, 2, and 3,* where we have the 3 revisions. The first block diagram is the test circuit, with one force sensor, with measurements included, and an LED. Then the sensor will be connected to our computer software via the STM32 microcontroller which will interact, reading and writing forward and backward with a USB programmer connected to our board, and finally reading and writing to the PC itself that is running the program. Additionally, there is a power source that is present that will be used to power the entire circuit. The second revision, which is the block diagram, was our full project, with 16 sensors in a 4x4 array and 16 LEDs that match each one. The total length is measured and the rate to the microcontroller is measured in 16 bits, because we need to measure each one. Other than that, the schematic is pretty much the same. The third and final revision was a reduction to 9 sensors in a 3x3 array due to us feeling like the functionality from 3x3 to 4x4 wouldn't be

impacted too much in relation to the cost. The LEDS would also go from 16 to 9, because of the 1:1 matching. We also included the PCB diagram to indicate that the sensors would relate to the PCB. Additionally, we added extra length to the sensors to account for space difference when we put the sensors on the actual lid/board. Finally, we added the different types of power sources we are using, instead of just indicating a singular power source. We used DC power of 3.3 V and 6.6 V from AC power, which was our barrel jack connector unit attached to a AC/DC adapter which draws power from the wall sockets. This would be converted using the adapter, and then reduced to fit our needs on the board using voltage regulators, which have the two different DC powers mentioned above.

Explanation of Hardware

Following the Project Proposal meeting, we were recommended to quickly figure out the target detection method. We were originally planning to use an ultrasound sensor to detect the Nerf dart approaching the target. However, we realized that a resistive force sensor would be the better choice for detection. After doing some research, decided to order one 30-73258 force sensor. Referring to *figure 6*, we chose this sensor due to its dimensions in reference to a nerf ball. The force sensor works like a potentiometer. The more force applied to the force resistor pad; the more voltage is sent out on the output. One of the first things we did after received our single force sensor was read a signal in the microcontroller.

We did all our testing on the STM32L031K6T7 microcontroller. This was the default microcontroller given to us on our breadboards. The first test we did using the force sensor, and the microcontroller was to receive a signal on an IO pin and output another signal signifying a signal was received. This was successful, though a high amount of force had to be applied to the force sensor to consistently send a receivable signal to the microcontroller. As a result, we decided to add the addition of an amplifier to our circuit.

Using the available components from our ECE210 parts, we added a LM741N operational amplifier to our component. We were able to amplify the force sensor's output signal and consistently output a 3.3V signal to the microcontroller. Furthermore, with the addition of the operational amplifier, the force sensor behaved a lot more sensitive, which is a good thing for hit detection of the Nerf ball.

With the confirmation of a working circuit, we ordered 10 additional force sensors. Unlike *figure 2*'s block diagram with a 4x4 sensor setup, we modified our design to take a 3x3 design. This is because, looking at *figure 7*, our microcontroller has enough ports for only 9 sensors, with 16 sensors being too much. Because we were able to validate a working circuit in our current microcontroller, we decided to keep using the same component, hence using a 3x3 force sensor layout.

After the verification of a working force sensor circuit, we considered how the final product will be powered. We wanted to have the device be portable. This resulted in the design choice of a barrel jack with a wall adapter for the power source to be considered. However, before deciding that, there was an issue. Our LM741N (*figure 8)* operational amplifier requires a two-polarity input power source. We can learn methods to create an inverted voltage polarity, but it would be a lot simpler to find an amplifier does not require a negative voltage, as Professor Smith mentioned. Ultimately, we decided to try the TLC271CP (*figure 9*) operational amplifier.

We chose the CP package because it was suitable for plugging into our breadboard. Looking at *figure 9,* we can see that this operational amplifier uses two voltages as an input, with one voltage being one-half less than the other. We chose 3.3V and 6.6V as the inputs, as 3.3V can be used also to power the microcontroller. When testing the TLC271CP, we noticed that the output was always inverted. Unable to change the outcome, we decided to try routing the output of the operational amplifier to an inverter. Originally a temporary solution, we used an inverter (SN74LS04N) from the ECE120 kit to invert the signal to the microcontroller. We powered the inverter using the 6.6V voltage source. The downside of using the 6.6V voltage source was that the output of the inverter was around 6.0V. To rectify this, we use a voltage divider circuit with two resistors to have a 3.3V output to the microcontroller. Furthermore, we also added a green LED in the output circuit to visibly show the signal. After the addition of the LED, we adjusted the voltage divider circuit, accounting for the LED's voltage drop, to continue outputting 3.3V.

Following the confirmation of a working circuit with the new operation amplifier and the inverter, we laid out the schematic on KiCad for 9 force sensor circuits. During the design of the footprints, we realized that a new package was needed for the TLC271CP and SN74LS04N IC chips, as we did not want them to be through-hole components on our final design. After considering the package variations in *figure 10* and *figure 12*, we decided to use TLC271CDR and SN74LS04DR IC chips.

While designing the schematic, we also decided on the barrel jack and the voltage regulators to use. When deciding on how the board was to be powered, we previously decided on using the WSU 120-2000 AC/DC Wall-Mount in *figure 13*. This wall adapter transforms 110V AC voltage to a 12V DC output. Matching the socket dimensions of the power plug, for the barrel jack, we decided to use the PJ-202AH. This was a generic barrel jack on Digikey and was one of the most stocked ones. For the voltage regulators, we had to find one that outputs 3.3V and 6.6V. Finding a voltage regulator for 3.3V was relatively simple, as that voltage is quite common. However, there were no voltage regulators that regulates a 6.6V output voltage. Upon doing some research, we searched for an adjustable regulator. Ultimately, we settled on the LD1117SC-R (*figure 15, figure 16*). This regulator can accept our 12V input from the barrel jack and is able to output a customizable regulated voltage output based on R2 in *figure 16*.

On the 10th revision of the PCB and approaching the deadline for PCB orders, we decided to add an alternative voltage input option in case the untested barrel jack and voltage regulators fail to

function. We added 4 100SP1T1B4M2QE flip-switches to change between power from the barrel jacks or a voltage and ground input from the lab DC power supply. One of the switches will be for the 12V input, one for the 6.6V input, one for the 3.3V input, and finally one for ground.

In our design, we used four 1x1 horizontally mounted female connectors for the alternate voltage inputs. The reason for the horizontally mounted connectors was because the top of the enclosure for the PCB was where the force sensors were to be mounted (*figure 34*). Side mounted inputs make more sense, as the lid is not supposed to be taken off unless if a problem with the PCB arises. Aside from the 1x1 horizontally mounted female connectors, we also had 9 1x2 horizontally mounted female connectors and a 1x6 horizontally mounted female connector for the FTDI232 UART breakout board. We have a vertically mounted 2x5 vertically mounted female connector for the ST-Link V2 programmer. As the board theoretically only needs to be programmed once, it makes sense for this connector to be vertically mounted.

Ultimately, our PCB came out looking like *figures 18-21*.

Written Description of Software

Upon receiving the 30-73258 force sensor, we decided to create a simple circuit to test the functionality of the part. The first circuit we created was simply powering one end of the force sensor to 3.3V, and the other to a green LED with a 330 Ω resistor. Utilizing this circuit, we were able to prove that the force sensor is like a potentiometer. However, unlike a potentiometer with a wiper, the wiper is the force applied on the resistive pad. The more force applied to the force sensor, the brighter the LED would glow.

Following the verification of the force sensor's functionality and behavior, we removed the LED and wired the output end of the force sensor to an open pin on the STM32L031K6T7 microcontroller. Using the "Hello World" main.c code (*appendix 39*) and modifying the microcontroller's input pins via the IOC file, we derived an algorithm that searches for an input signal. When a signal if found, it will output, on another pin, a voltage that turns on an LED. Following this test, we were able to conclude that a signal can be received by the microcontroller when a certain amount of force is exerted on the force sensor.

Further improving on the detection idea, we included a UART connection to our circuit with the "red UART breakout board". Instead of activating an LED when a signal is detected by the microcontroller, the microcontroller will now output a "Hit!" message to PuTTY via UART. This was done using an infinite "while (1)" loop with an if statement inside the loop that checks for the condition of HAL_GPIO_ReadPin (GPIO_A, GPIO_PIN_0) != 0). Whenever Pin0 of I/O input port section A is not 0, we output the "Hit!" message. The "Hit!" message is a byte array that gets transmitted through UART using HAL_UART_Transmit (&h1puart1, Test, sizeof(Test), 10). This is followed by a 500ms delay, which prevents continued "Hit!" messages. We configured the UART to 19200 BAUD with 8-bits of data being sent each time. There are no parity bits in

our transmission. The connection type, serial, and the serial line must be specified in the "Session" category in the PuTTY Configuration.

Following the progress of getting a hit message, we reconfigured the IOC file with the pin assignments on the microcontroller. We wanted to organize the pins so that they are easily accessible during the wire routing in the PCB design. However, we realized that only pin pairs 8 and 9, as well as pairs 24 and 25 could be used for UART. UART was not able to be configured to work on any pin. We suspect this was a design choice by the designers of the microcontroller to prevent interreference, as pins 8, 9, 24, and 25 are in corners of the microcontroller, far from VDD and GND. As a result, we could not fulfill the organized ideal of the pin assignments we wanted. This is ok but means that our track routing in KiCad will not be as neat, and more VIAs will need to be used.

After creating a total of 9 inputs, as well as an UART output on the IOC file *figure 22*, we proceeded to update the program to take an input from 9 force sensor circuits. Like the algorithm used to read the hit detection of 1 force sensor from Pin0, we totaled 9 pins, from Pin0-Pin8, in GPIO port section A. Instead of the message: "Hit!", we now have a message that specifies the sensor hit. For example, if sensor 2 was hit, the UART will output: "Hit 2!!!".

3D Printed Enclosure

The 3D printed enclosure is one that consists of a box to hold the PCB and a lid to hold the sensors, thick enough to withstand the impact of the hits from the ball. The 3D printed model was created to have holes on the sides, which explains why the corners are larger, for the wires connected to the force sensors, so we can fold the sensors on top of the lid. We also made the box to consider the switches, which were a significant height mounted on the PCB. The lid itself fits on top of the box by pushing the PCB down with the legs as shown in *Figure 31*. As you can see, there are three contact points or legs that push our circuit down to fit snugly into the box. We had extra volume on the middle and corners of the box's base which would act as a friction fit for our PCB. This is to ensure that the PCB doesn't start rattling around when the ball impacts the board.

Our initial enclosure is demonstrated in *Figure 23.* The second revision, *Figure 24*, had increased height because we made a miscalculation in the height of the switches, so we need a larger box height to compensate for that. Our third revision in *Figure 25* was a hole in the side, to compensate for the barrel jack extension that we failed to acknowledge. The component soldered on the PCB fit in the box, but not the actual extension connected to it. Then *Figure 26* demonstrates the last box revision and our final design. This would have another hole on the side to include the UART connection. Like the barrel jack connector, we forgot to account for the wire coming out, and so we needed a hole for that as well. The *Figure 27-30* will show the different views of the final box.

As for the lid, *figure 31* has the three points to push the PCB. This will fit perfectly as we later found out, based on thickness. However, we made another revision in *figure 32,* where we made the board thicker, because we found that the lid was too thin at the first revision. Our final revision of *figure 33 and figure 34*, which are just different angles of the same box, had the legs longer because of the switch height error mentioned above when revising our base box. Additionally, there is a hole in the middle of the lid to reduce the number of wires coming out the side, especially as the center force sensor in a 3x3 array would be very hard to lay on top without intersection points. These served to act as our total enclosure for our project.

Final Assembly

The final assembly was simple given all the components were ready. We first laid down the box and fit the PCB cozily into it. Afterwards, we made sure that it fit properly, then put the wires from the force sensors into their proper connectors created by us on the PCB board. Then we laid the force sensors on top of the lid and taped the bottom down to the top of the lid. We then put the lid on top, closing the box and forming our total enclosure. Afterwards, we connected the wall adapter giving power to our design, then plugged in the UART to measure the values needed to demonstrate what we wanted on the monitor, based on the software that we programmed. This was the total layout of the steps we took in assembly, and luckily for us, it worked on the first try after all this, hooray!

Weekly Progression

Below is a summary of our weekly project timeline from the 4th week. During the first three weeks, we were getting orientated with the course, with one of our members joining during the second week and another member changing from 2 credit hours to 3 credit hours. We were also focused on getting the course “exercises” working.

Week 4 (09/12):

Meet with Professor Smith and our TA, Yilong Niu, to confirm our idea of the Digital Targeting System. In this week, we will work towards making an acceptable block diagram that highlights, at a high level, what we plan to achieve. Furthermore, we will research and determine the sensor type that we will be using.

Week 5 (09/19):

We will demonstrate and confirm our block diagram with Professor Smith. We will determine the sensor type that we will be using, find a suitable sensor, depending on the projectile we will be firing, and order one on Digikey. If the sensor arrives before the end of the week, we will make a "quick and dirty" circuit to test out the sensor.

Week 6 (09/26):

A lot of documentation reading is required for this week. The sensor was confirmed to be working, however, in our demonstration to Professor Smith, the signal might be too weak. This is because the amount of output depends on the force exerted on the sensor. Reviewing the sensor documentation, the force exerted by the Nerf ball will probably be too weak. Therefore, we will be using a circuit divider, or an OP-Amp of some kind to amplify the signal. We will have to do some research and testing to figure out whether to use a resistor

divider network, or an OP-Amp. If we determine to use an OP-Amp, which model do we choose? A lot of documentation reading will be needed.

Week 7 (10/03): This week, we hope to finalize on an OP-Amp model to use. Currently, we are using the LM-741N model. It works, but we worry it might not amplify the force sensor's signal enough. We will also order a cheap Nerf gun. This is because we have no idea the force exerted by the projectile on the sensor. Hopefully we can get an idea of the force. If the Nerf gun does not arrive on time, we will try to read some signals on the computer. Maybe we will modify an old exercise's program for this.

Week 8 (10/10): In week 8, we finalized our OP-Amp, choosing the LM-741N. However, discovered a problem with the LM-741N circuit. On idle, the output of the OP-Amp is negative at around -0.84V. When the force sensor is activated, it jumps to around 2.0V. We need to troubleshoot this, but we are wondering if it is because we left the "offset" pins floating. Moreover, we are testing if the OP-Amp can run at 3.3V and -3.3V. That way, we can have the same power supply for the microcontroller and the force sensor circuit. We are also looking to buy or create a 110V AC adapter to 12V DC converter. We will need to speak to Professor Smith about which path to take, as 110V AC might be unsafe to work on directly on a breadboard. We will also need to look at how to create an inverted DC voltage of, let's say -12V. On the software side, we will continue to work on receiving signal from the force sensor. Hopefully we can demonstrate the microcontroller reading the signal from the force sensor successfully on next week's mid-semester demo.

Week 9 (10/17): In this week, we demonstrated to the whole class our project and listened to advice from them. One major thing that we need to resolve is the negative voltage supply requirement of our LN-741N OP-Amp. We need to do more documentation reading on the OP-Amp to see if it is possible to run it with just V+. If that option is not possible, what next steps we need to take to address the V- requirement, either a different OP-Amp or figure out a way to supply V- using some polarity inversion methods. We are also trying to determine what display to use for our project. Talking to Yilong Niu, using an HDMI port might be impossible for the microcontroller. We are thinking of using a HEX LED display this semester. Furthermore, we have ordered 10 more force sensors, as it is cheaper than ordering our required 8, AC to DC wall plug, and single HEX digit LED display. Currently, we are looking at the TLC271CP OP-Amp because of its minimum V_dd voltage of 3V.

Week 10 (10/24): In the 10th week, we have ordered the remaining parts for the board. We decided to change a new OP-Amp, since, after reading documentation, we realized that the LN-741N is dual power supplied, and there is no physical way to make it run from one power source setting. We have settled with the TI TLC271CP. The reason for the "CP" ending is because the operating point of this OP-Amp is 3-16V and works from 0C to 70C. We will start drafting up the schematic this week with this OP-Amp in mind. We tried to draft it up over the weekend, but there was a version error, where Ki-CAD needs to be updated. Before finalizing the schematic, the parts will hopefully arrive, so that we can test it on a breadboard before finalizing the design.

Week 11 (10/31): This week, we will be drawing our breadboard and moving it to KiCad. We spent time going over documentation and examining which footprints and which packages to use. We discovered that we need to use a new package for the inverter and the op-amp. We decided on board dimensions of 120 x 120 mm, and this would help with debugging even though it is a little big. In our footprint layouts, we decided to put the microcontroller in the very center with the inverters near it. The components are on the top layer only. The top copper layer is ground, and the bottom layer is 6.6 V. We will route the 3.3 V using tracks and VIAs. The UART will be on the top of the board, and on the other 3 sides, each will have 3 1x2 female connectors for the force sensors. On the bottom right of the board will be where the power components are. In Rev10, we added 4 switches to either use the DC power supply or the barrel jack.

Week 12 (11/7): In this week, we ordered our PCB on 11/7. We decided on a matte-black silkscreen with white text. The pads will be ENIG. The overall aesthetics was to match the Apple PCBs. We will proceed to go over and double check if there were any issues with the schematic and PCB design this week. If there is additional time, we will come up with a parts list with all the components on the board and make a large order for the components on the board. We have 139 components; we will order some extra parts in case soldering issues arise. We are ordering our own solder paste because the solder paste in the class should be refrigerated.

Week 13 (11/14): We practiced and touched up on our soldering skills. We took some spare PCBs to practice SMD and through-hole components with paste and wire solder.

Week 14 (11/21): Fall Break.

Week 15 (11/28): This week, we are hoping to solder all the components on the PCB. However, we realized that on 11/27 that the order form submitted on 11/12 was not approved. After having Yilong talk to the professor, the instructor decided to do 2-day shipping. Luckily, the parts arrived 12/1 afternoon. We were able to solder and get the board working on 12/2. We have also tested that the force sensors work with the program. We are working on 3D printing the enclosure for the PCB and the force sensors. We hope that the printing will be completed on 12/5, the presentation date.

Week 16 (12/05): Demo Day!

Conclusion

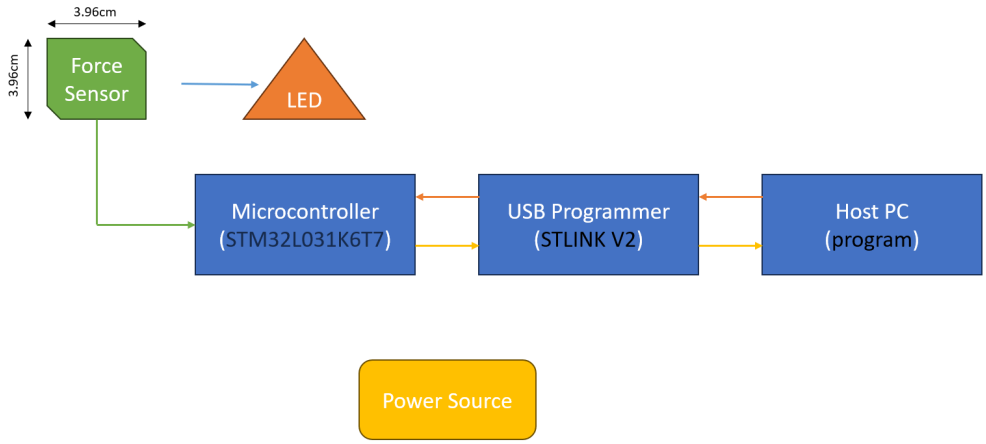
The original proposal of the Digital Target System was to build a device that can detect, and display airsoft pellet hit locations. However, in the initial project proposal meeting, the idea was modified, as the testing and demonstration with an airsoft gun cannot be performed on school grounds. As a result, a Nerf gun with shoots a foam ball projectile was used in replacement to the airsoft gun shooting 6.00mm round pellet projectiles. At a high level, with the use of a resistive force sensor, operational amplifier, and an inverter, the location of where the projectile hit is sent to the microcontroller, as a signal, which is then relayed to a PC, and is ultimately displayed on a monitor display. The functioning circuit was constructed in a 120mm-by-120mm

PCB board, which is enclosed in a custom 3D printed box. The lid of the box consists of 9 sensors, in a 3 by 3 formation. In the result, the product can detect and tell the user where their Nerf ball projectile hit on the target. We are very impressed with the result, as no modifications to the PCB had to be made. Ultimately, the product functions as intended, and we are pleased to leave this project behind in the ECE395 room for future Illinois engineers to interact with.

The project could not be finished without thanking a few people inside and outside the course that helped make this functioning product a success. Professor Casey Smith helped a tremendous amount throughout the course by giving advice on trouble shooting our circuit and PCB, as well as confirming parts we chose for our design. Yilong Niu, the TA in the course was always helpful during and outside of office hours to answer any questions, as well as keeping us updated on the logistics of parts. Jason Guo, a fellow ECE student from outside the course was very helpful in aiding to create the enclosure, with four revisions, under our specifications. James Dylan Menezes was very helpful in helping us slice Jason Guo's models for the class's MakerGear M2, as our slicing on Cura was not fit for the 3D printer in the class. Finally, Stephen Messing, an ECE PhD student from the Laboratory of Optical Physics and Engineering was helpful with any other questions and advice that arose in our project. This project could not be finished without these people, and we are forever grateful for their help.

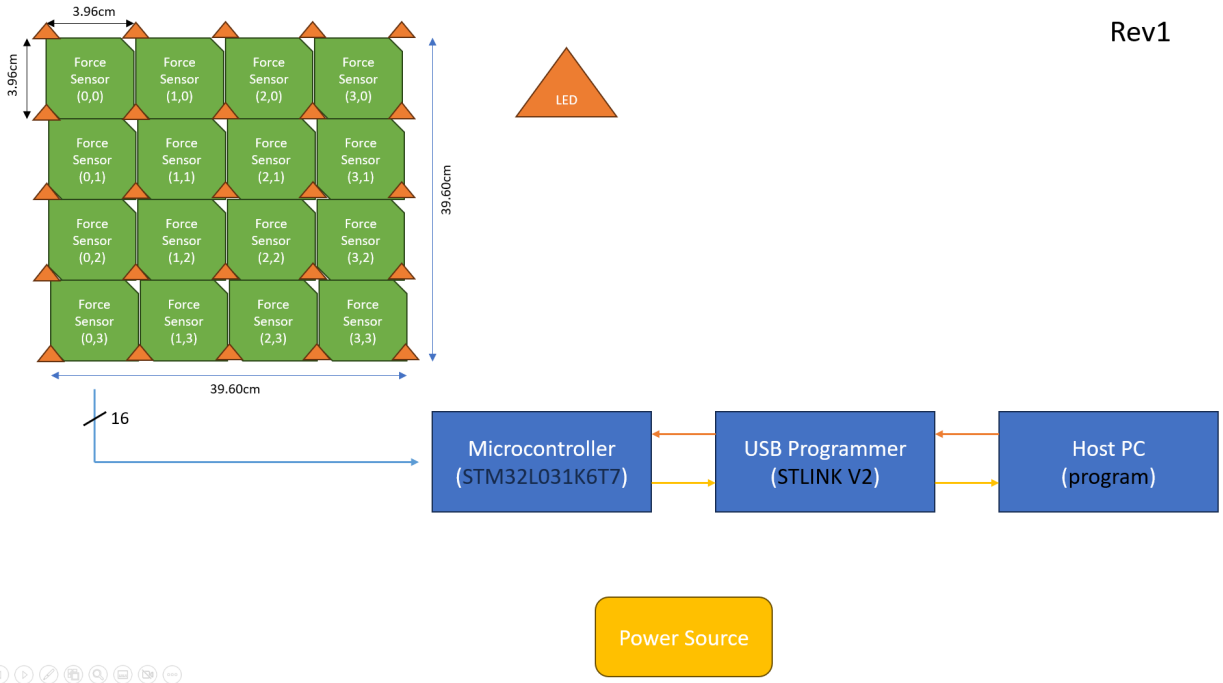
References

Rev0

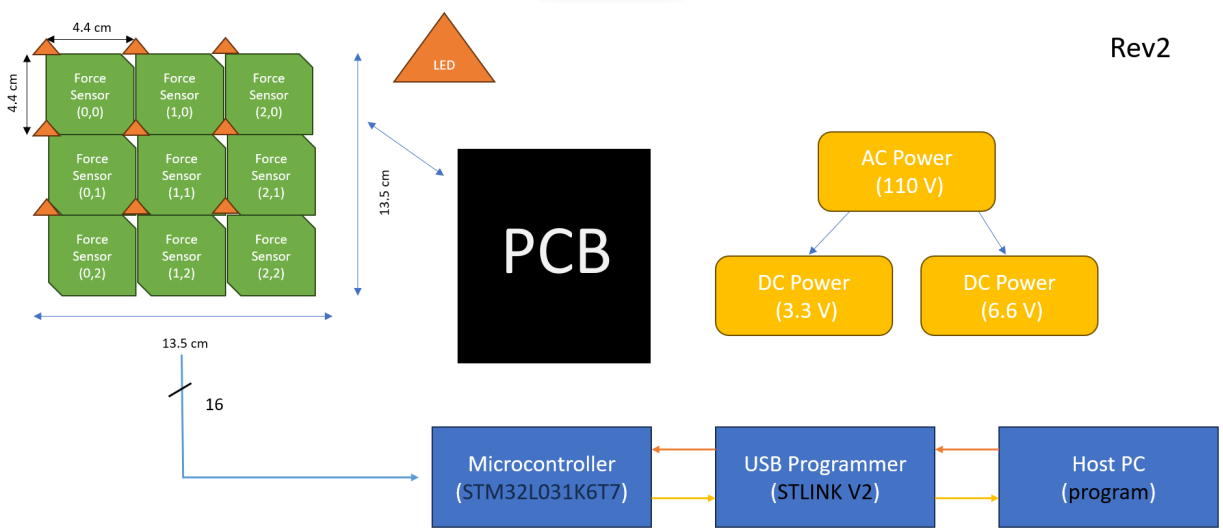


(Figure 1 – Block Diagram Rev0)

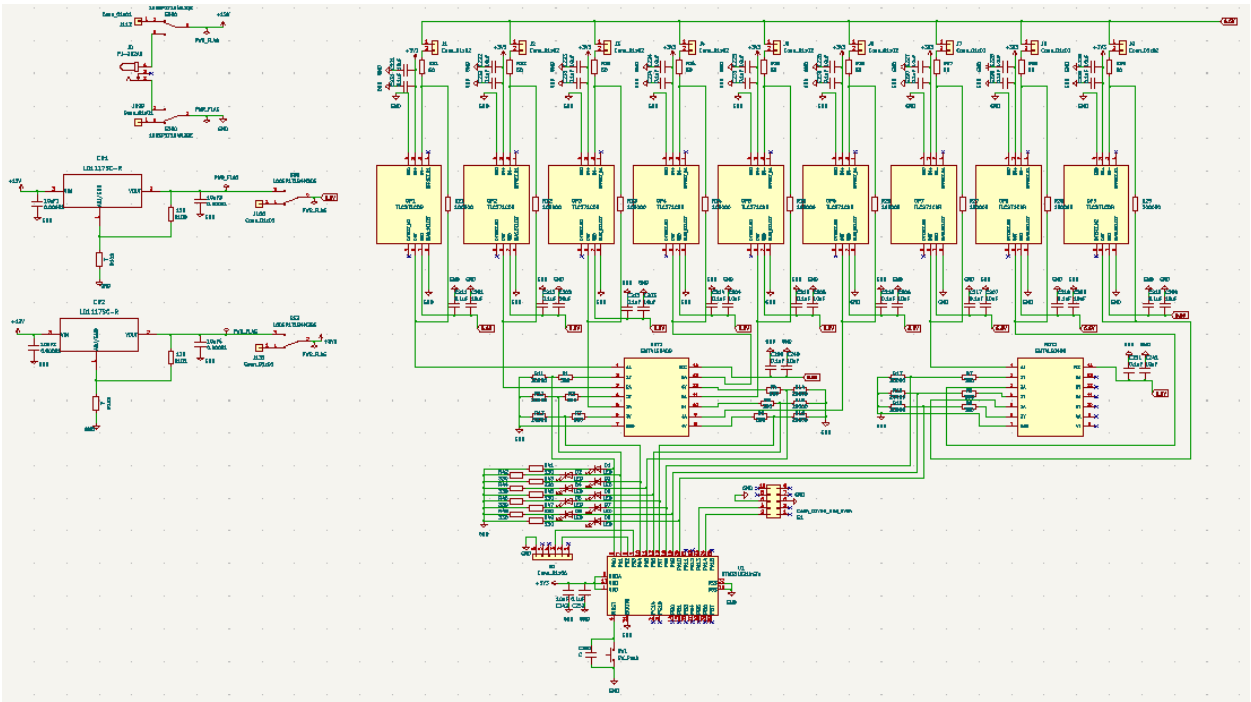
Rev1



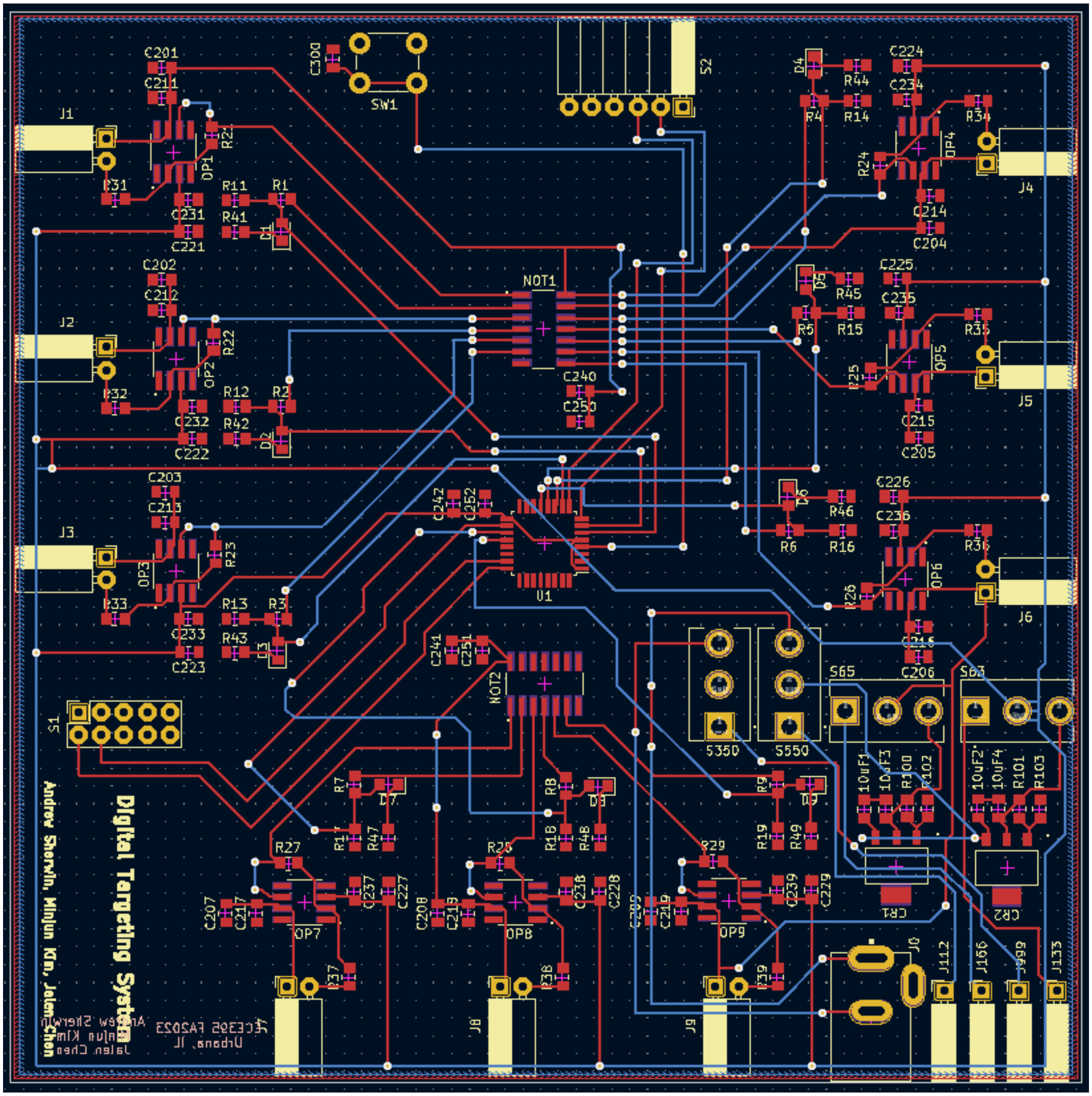
(Figure 2 – Block Diagram Rev1)



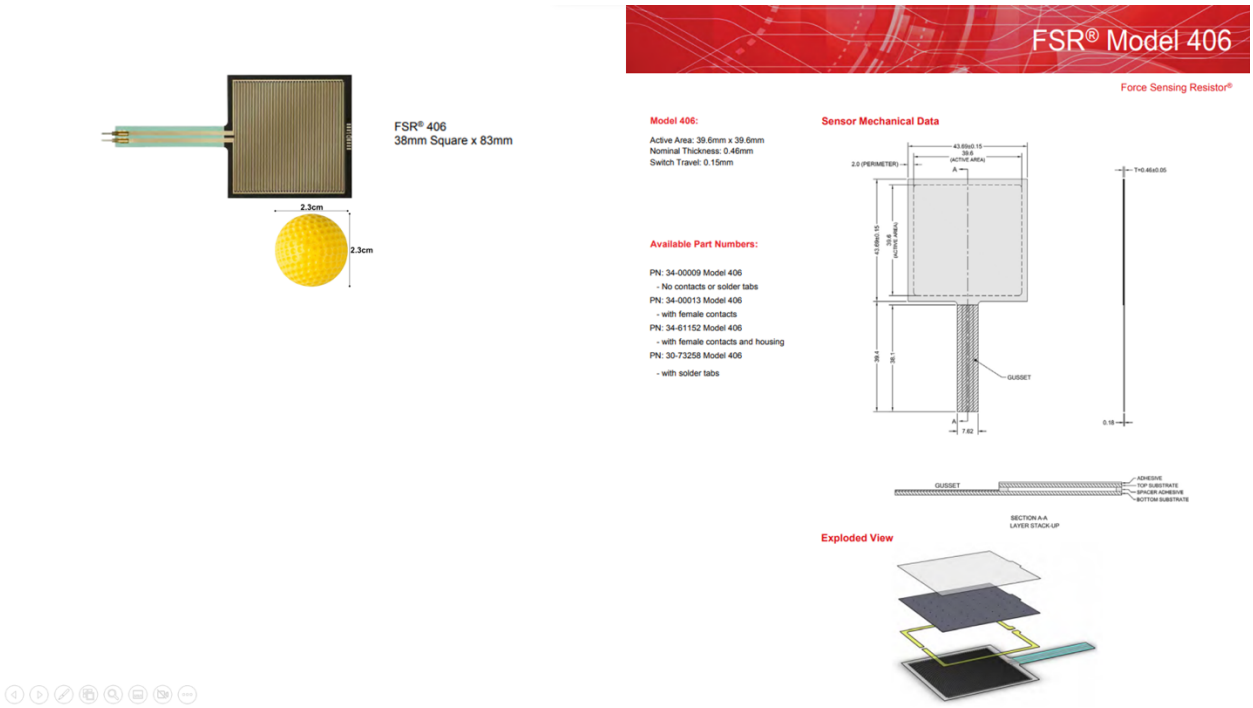
(Figure 3 – Block Diagram Rev2)



(Figure 4 – KiCad Schematic)

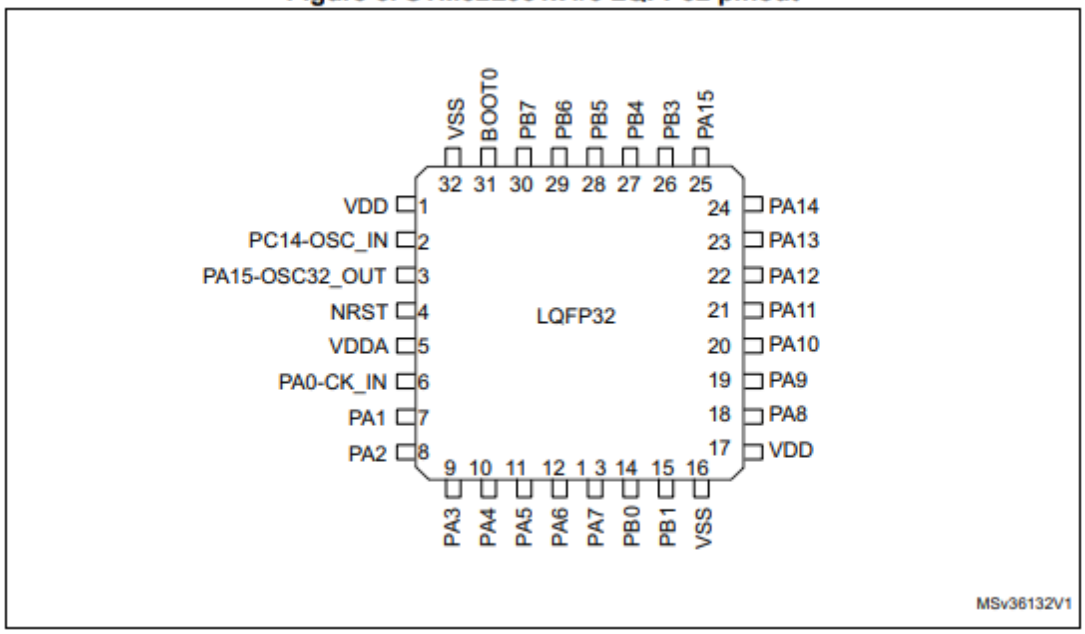


(Figure 5 – KiCad PCB Footprint)



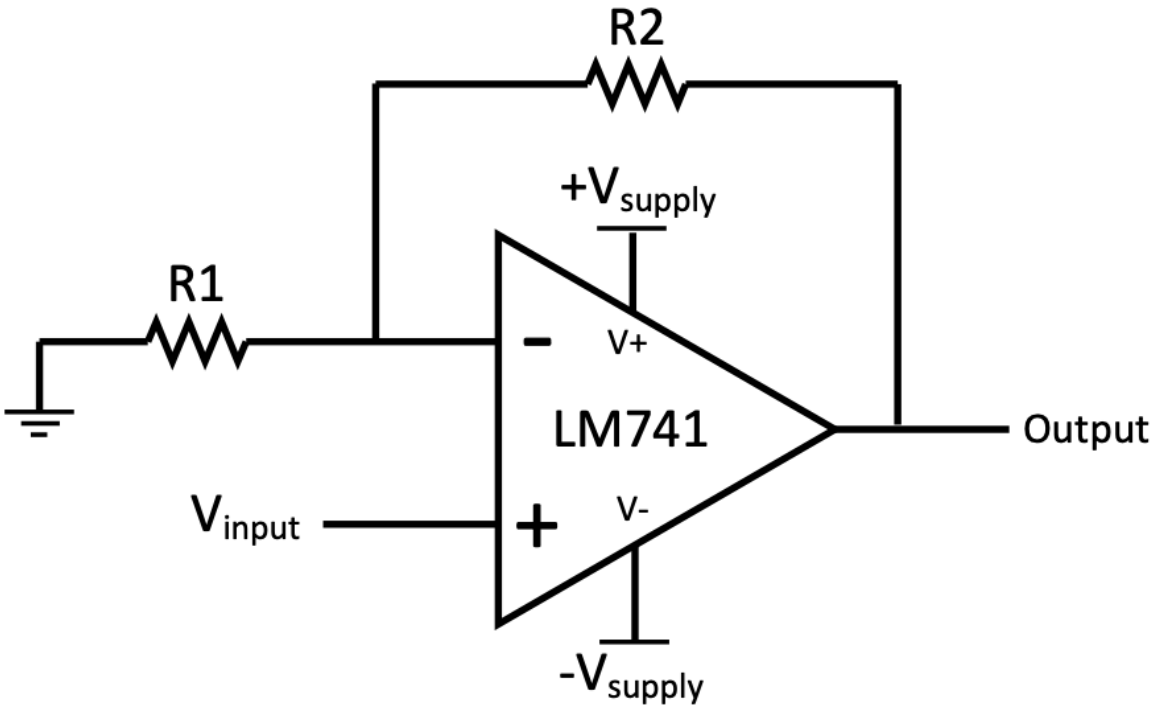
(Figure 6 – 30-73258 (Force sensor) with Nerf Ball)

Figure 5. STM32L031x4/6 LQFP32 pinout

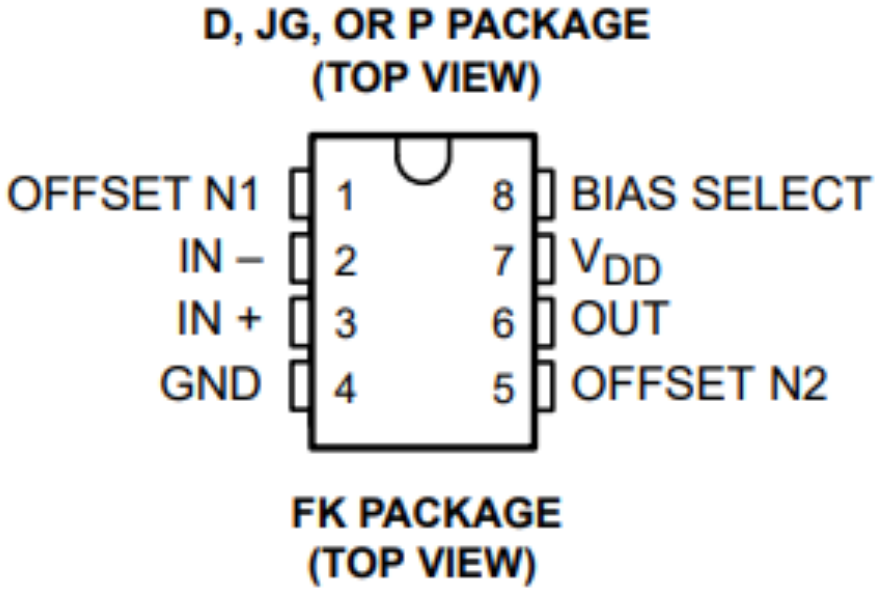


1. The above figure shows the package top view.

(Figure 7 STM32L031K6T7 Microcontroller)



(Figure 8 – LM741N Operational Amplifier)



(Figure 9 – TLC271CDR Package Footprint)

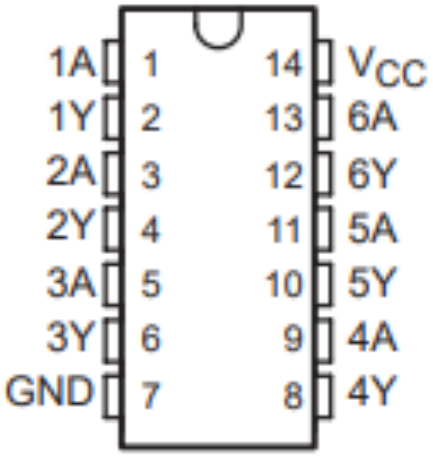
AVAILABLE OPTIONS

T_A	V_{IO}max AT 25°C	PACKAGE			
		SMALL OUTLINE (D)	CHIP CARRIER (FK)	CERAMIC DIP (JG)	PLASTIC DIP (P)
0°C to 70°C	2 mV 5 mV 10 mV	TLC271BCD TLC271ACD TLC271CD	—	—	TLC271BCP TLC271ACP TLC271CP
−40°C to 85°C	2 mV 5 mV 10 mV	TLC271BID TLC271AID TLC271ID	—	—	TLC271BIP TLC271AIP TLC271IP
−55°C to 125°C	10 mV	TLC271MD	TLC271MFK	TLC271MJG	TLC271MP

The D package is available taped and reeled. Add R suffix to the device type (e.g., TLC271BCDR).

(Figure 10 – TLC271CDR Available Packages)

SN5404 . . . J PACKAGE
SN54LS04, SN54S04 . . . J OR W PACKAGE
SN7404, SN74S04 . . . D, N, OR NS PACKAGE
SN74LS04 . . . D, DB, N, OR NS PACKAGE
(TOP VIEW)



(Figure 11 – SN74LS04 Package Footprint)

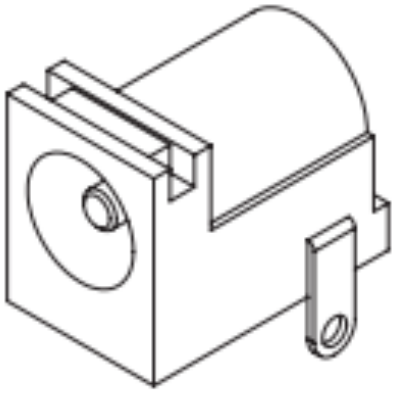
ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	PDIP – N	Tube	SN7404N	SN7404N
		Tube	SN74LS04N	SN74LS04N
		Tube	SN74S04N	SN74S04N
	SOIC – D	Tube	SN7404D	7404
		Tape and reel	SN7404DR	
		Tube	SN74LS04D	LS04
		Tape and reel	SN74LS04DR	
		Tube	SN74S04D	S04
		Tape and reel	SN74S04DR	
	SOP – NS	Tape and reel	SN7404NSR	SN7404
		Tape and reel	SN74LS04NSR	74LS04
		Tape and reel	SN74S04NSR	74S04
	SSOP – DB	Tape and reel	SN74LS04DBR	LS04
−55°C to 125°C	CDIP – J	Tube	SN5404J	SN5404J
		Tube	SNJ5404J	SNJ5404J
		Tube	SN54LS04J	SN54LS04J
		Tube	SN54S04J	SN54S04J
		Tube	SNJ54LS04J	SNJ54LS04J
		Tube	SNJ54S04J	SNJ54S04J
	CFP – W	Tube	SNJ5404W	SNJ5404W
		Tube	SNJ54LS04W	SNJ54LS04W
		Tube	SNJ54S04W	SNJ54S04W
	LCCC – FK	Tube	SNJ54LS04FK	SNJ54LS04FK
		Tube	SNJ54S04FK	SNJ54S04FK

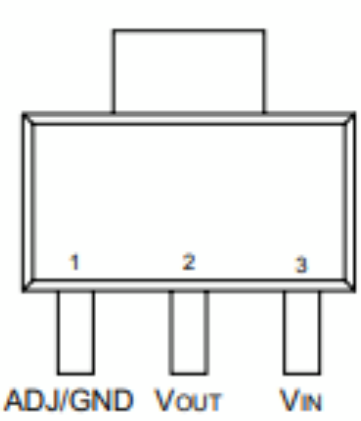
† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

(Figure 12 – SN74LS04 Available Packages)

(Figure 13 - WSU 120-2000 AC/DC Wall Mount)

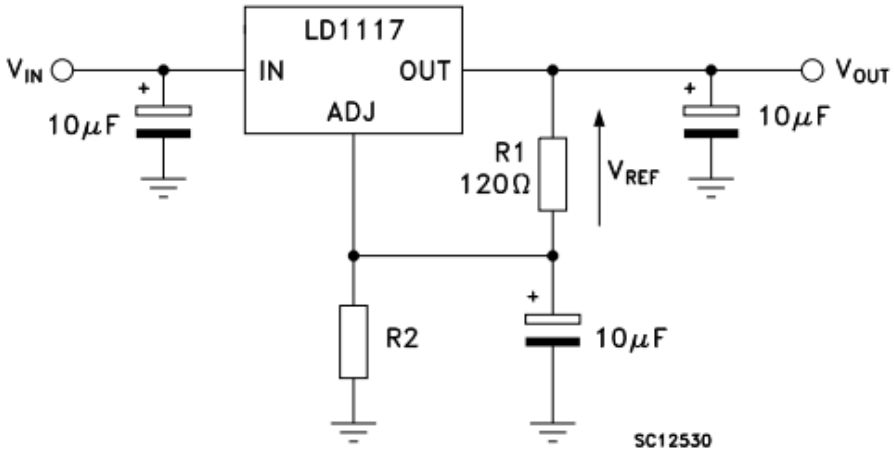


(Figure 14 – PJ-202AH Barrel Jack)

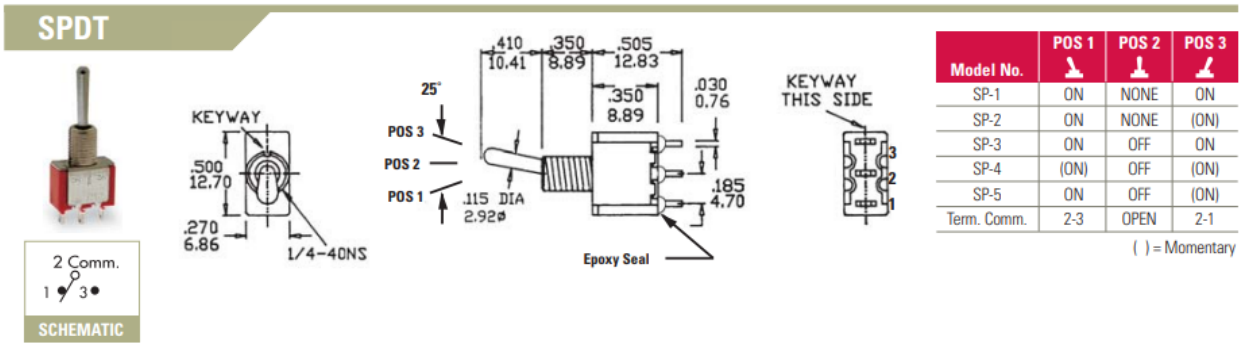


(Figure 15 – LD1117SC-R Package)

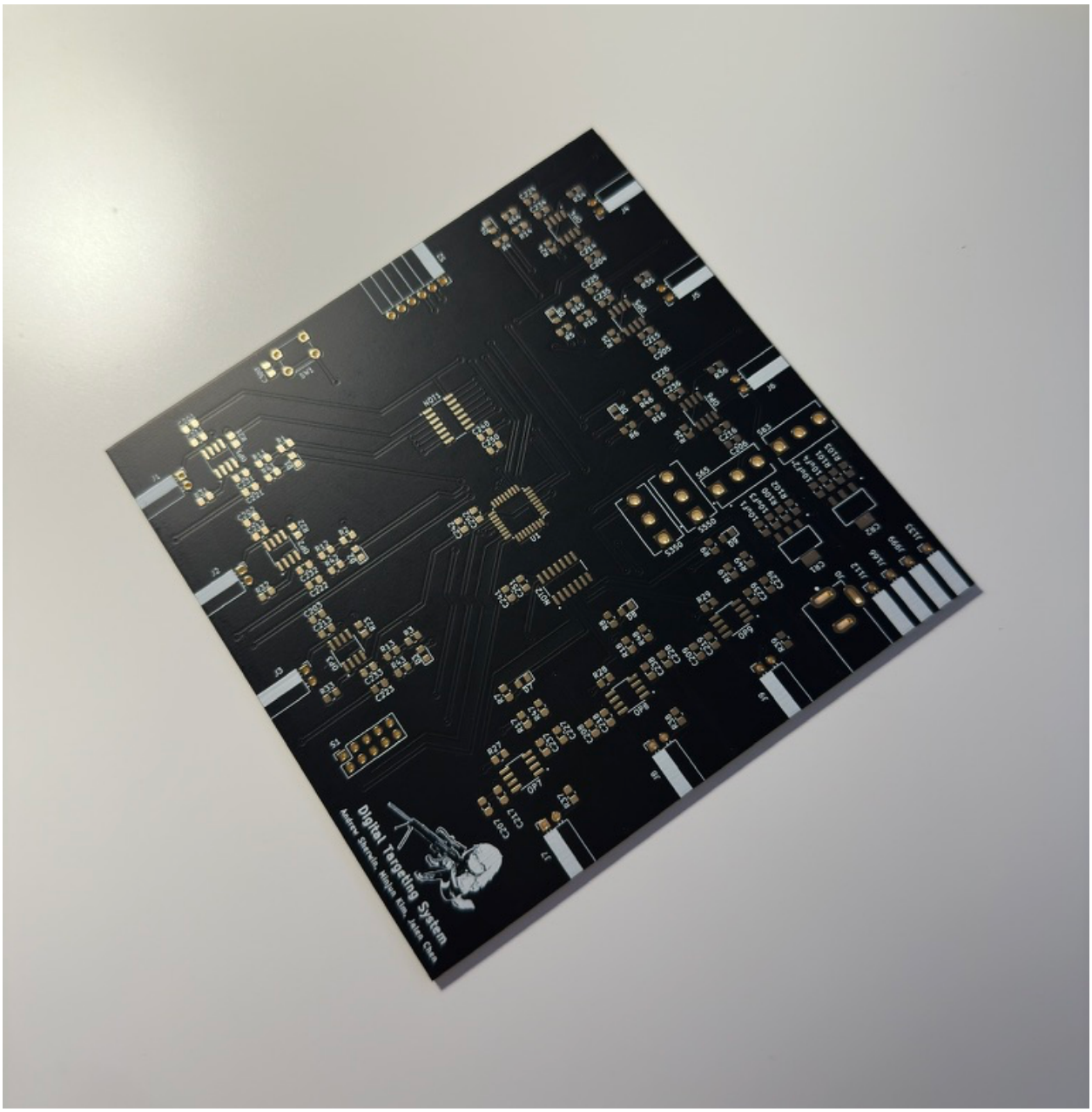
Figure 11. Adjustable output voltage application with improved ripple rejection



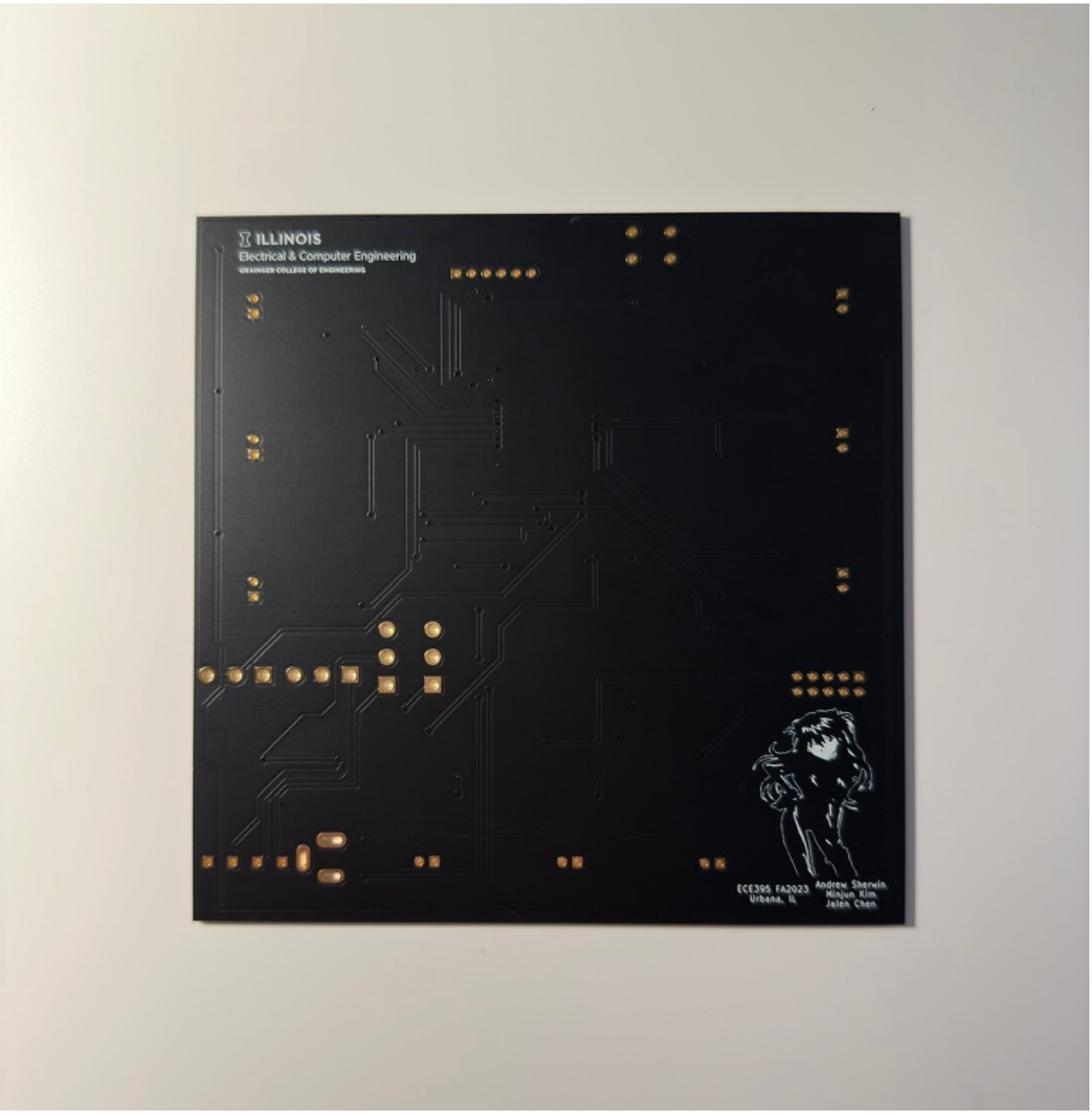
(Figure 16 – LD1117SC-R Schematic)



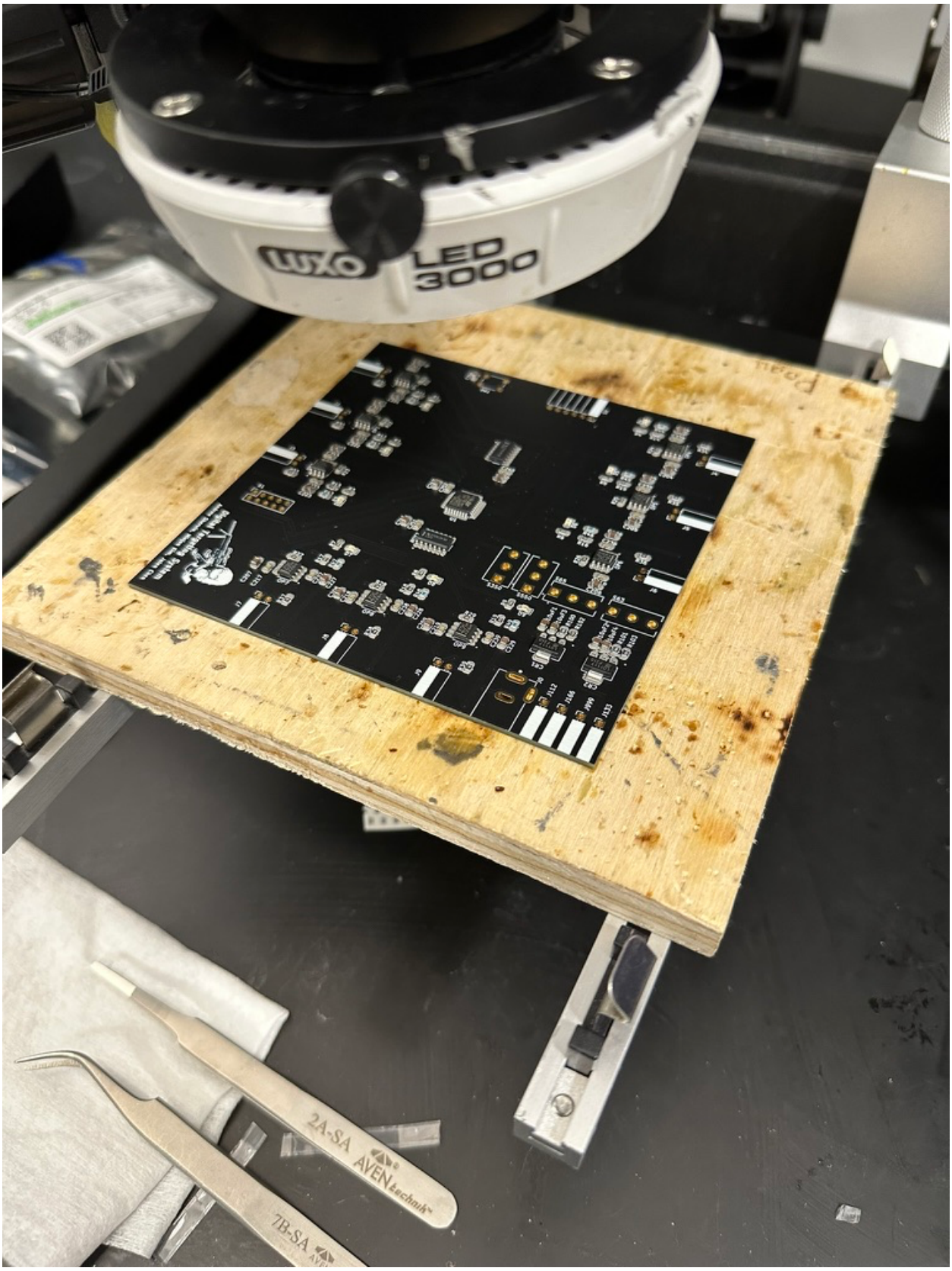
(Figure 17 - 100SP1T1B4M2QE Switch)



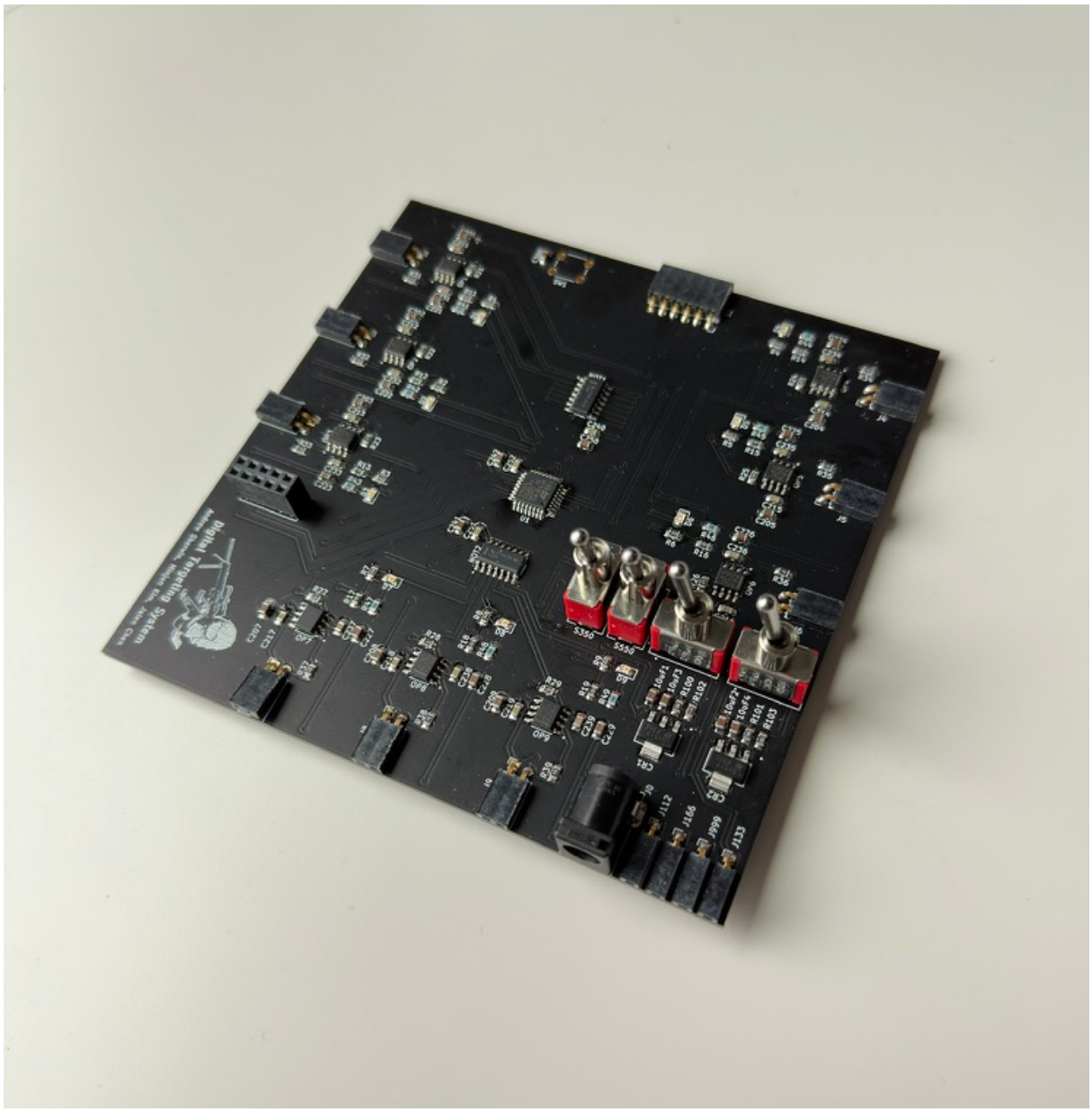
(Figure 18 – PCB Front)



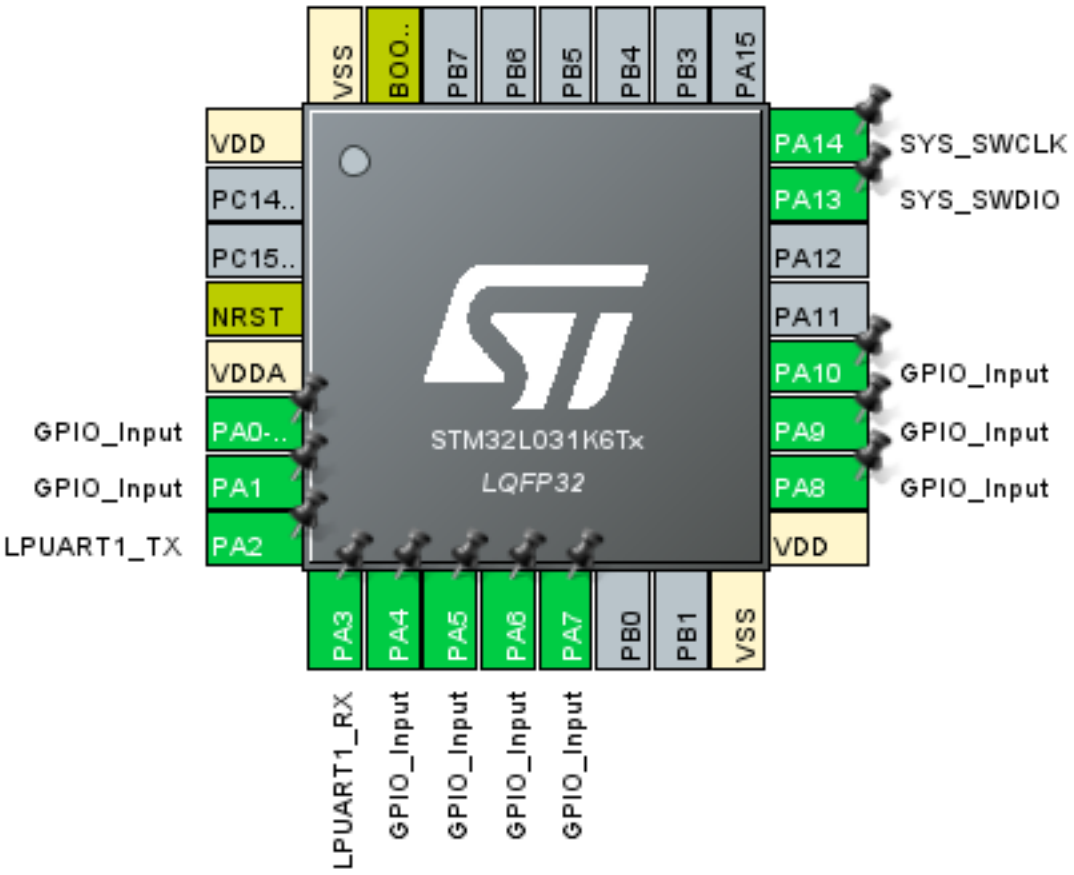
(Figure 19 – PCB Rear)



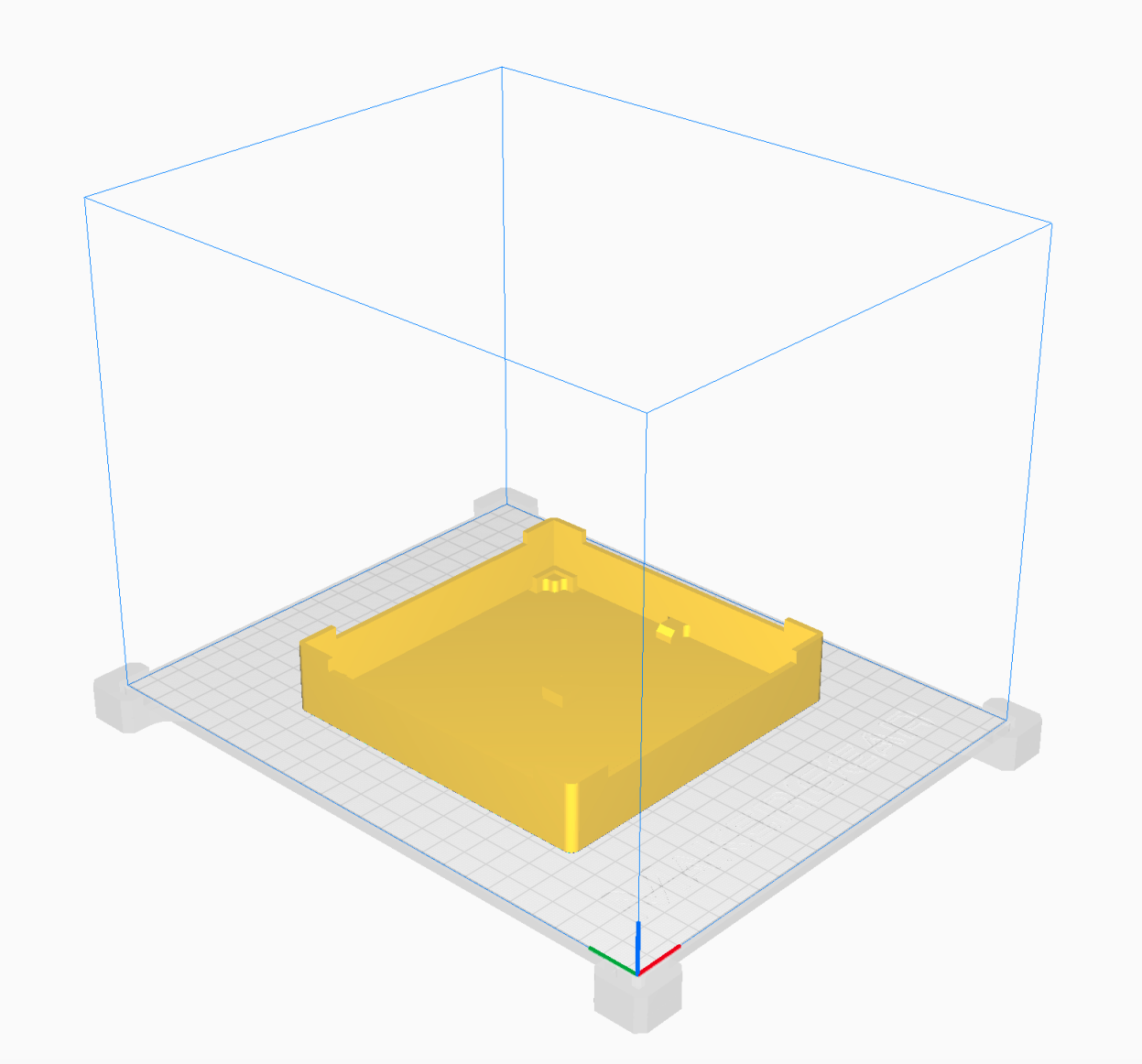
(Figure 20 – PCB Paste with Components)



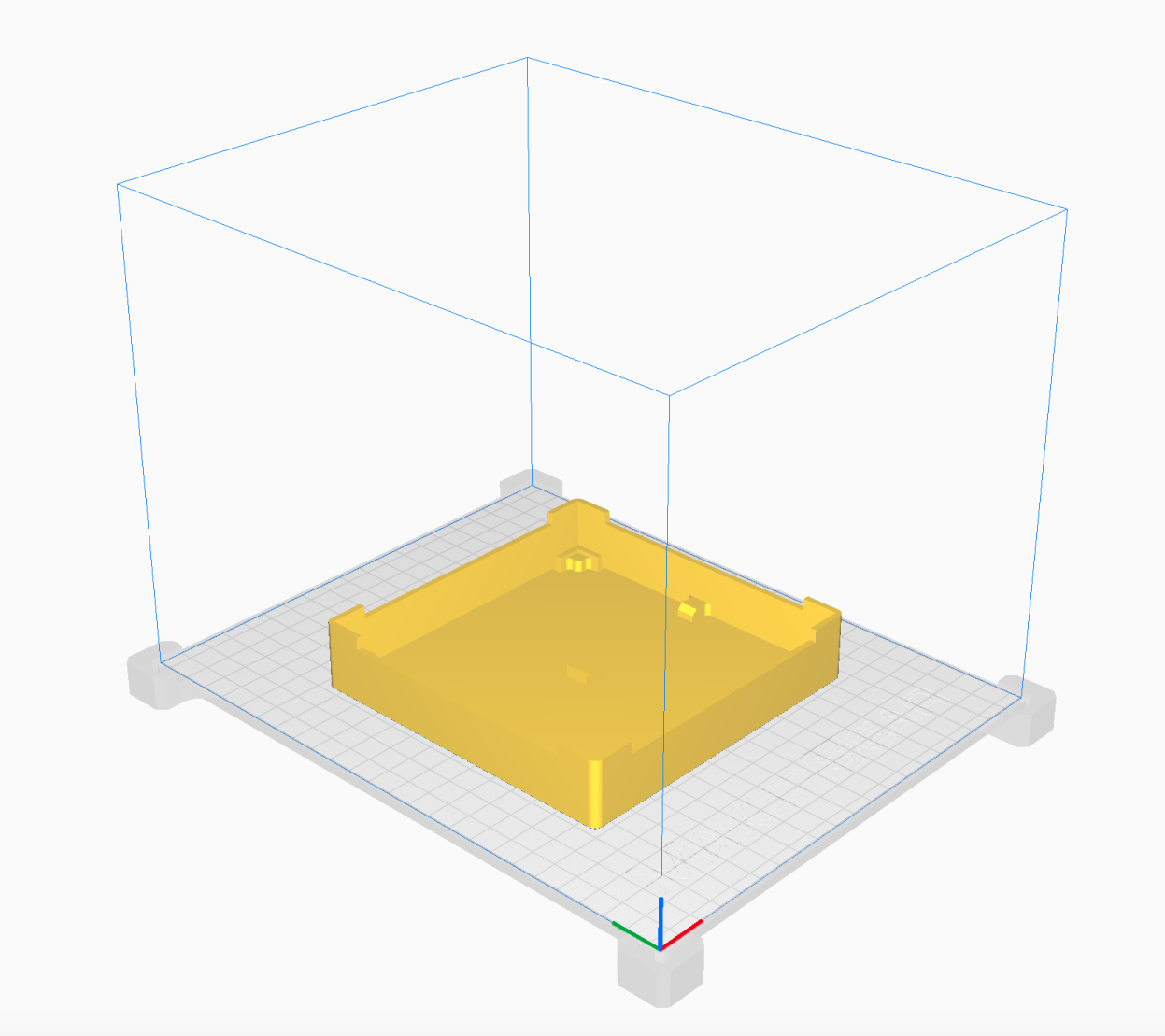
(Figure 21 – PCB Soldered)



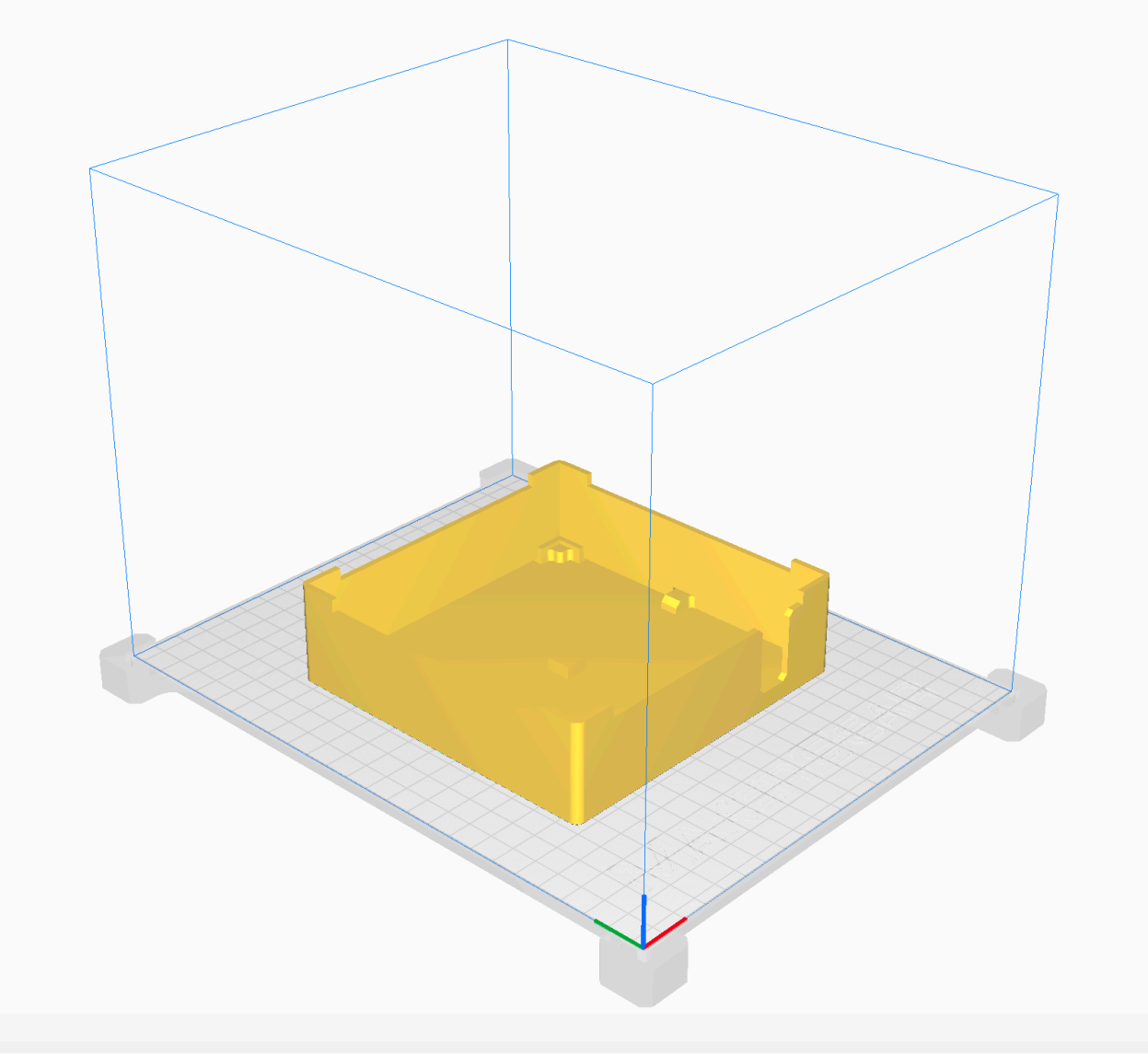
(Figure 22 – STM32L031K6T7 IOC Pin Assignments)



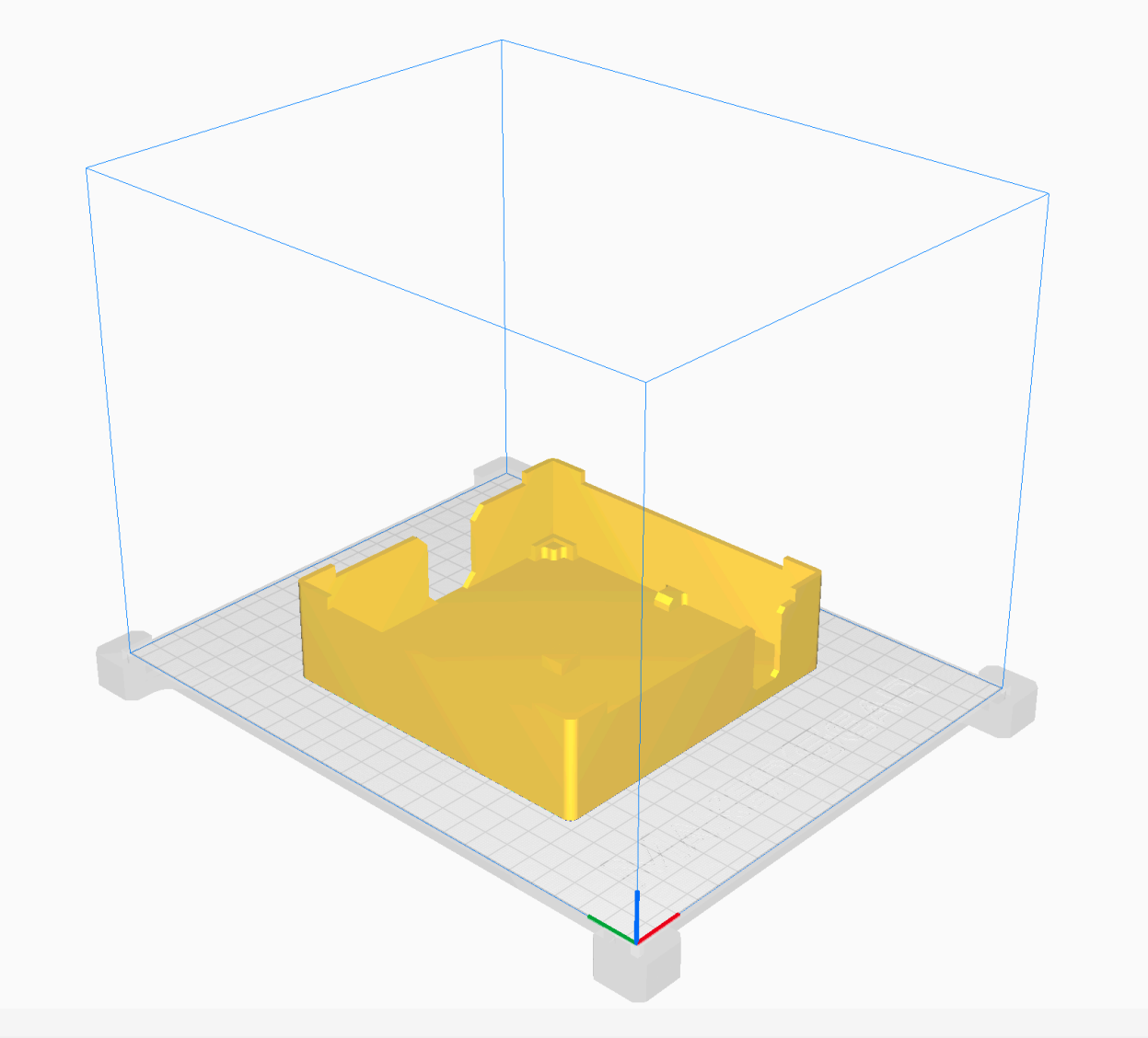
(Figure 23 – Box Base Rev1)



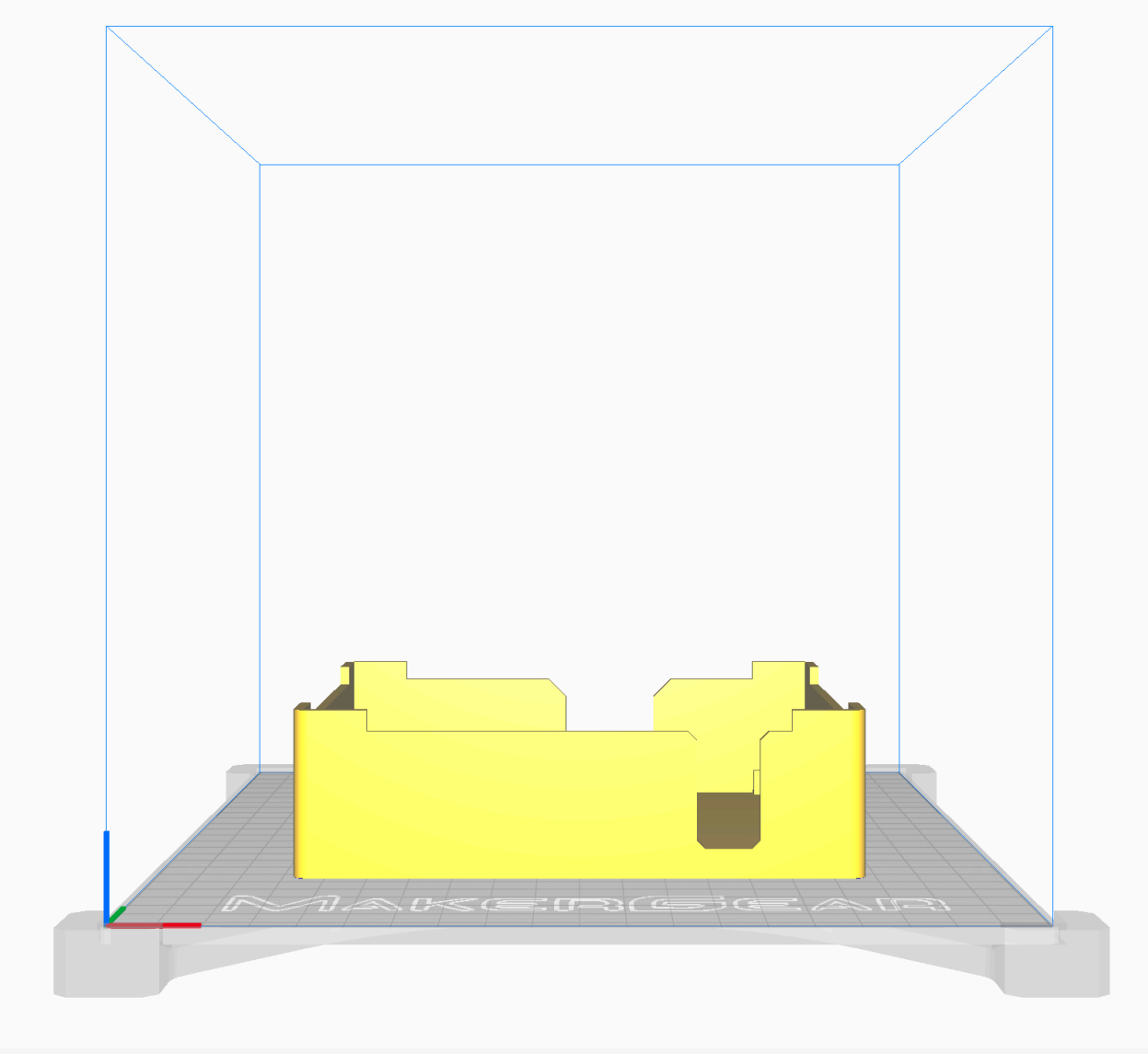
(Figure 24 – Box Base Rev2)



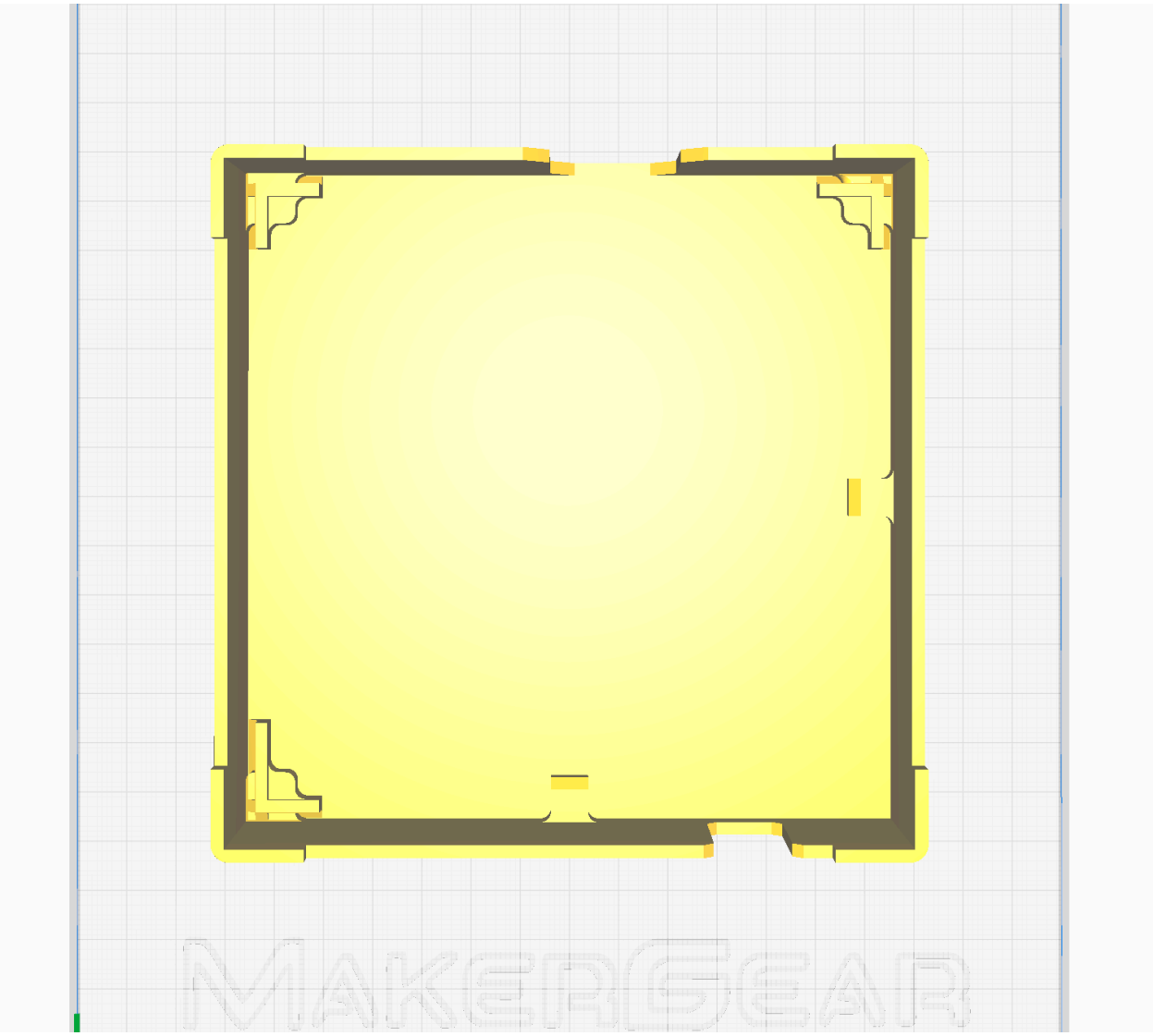
(Figure 25 – Box Base Rev3)



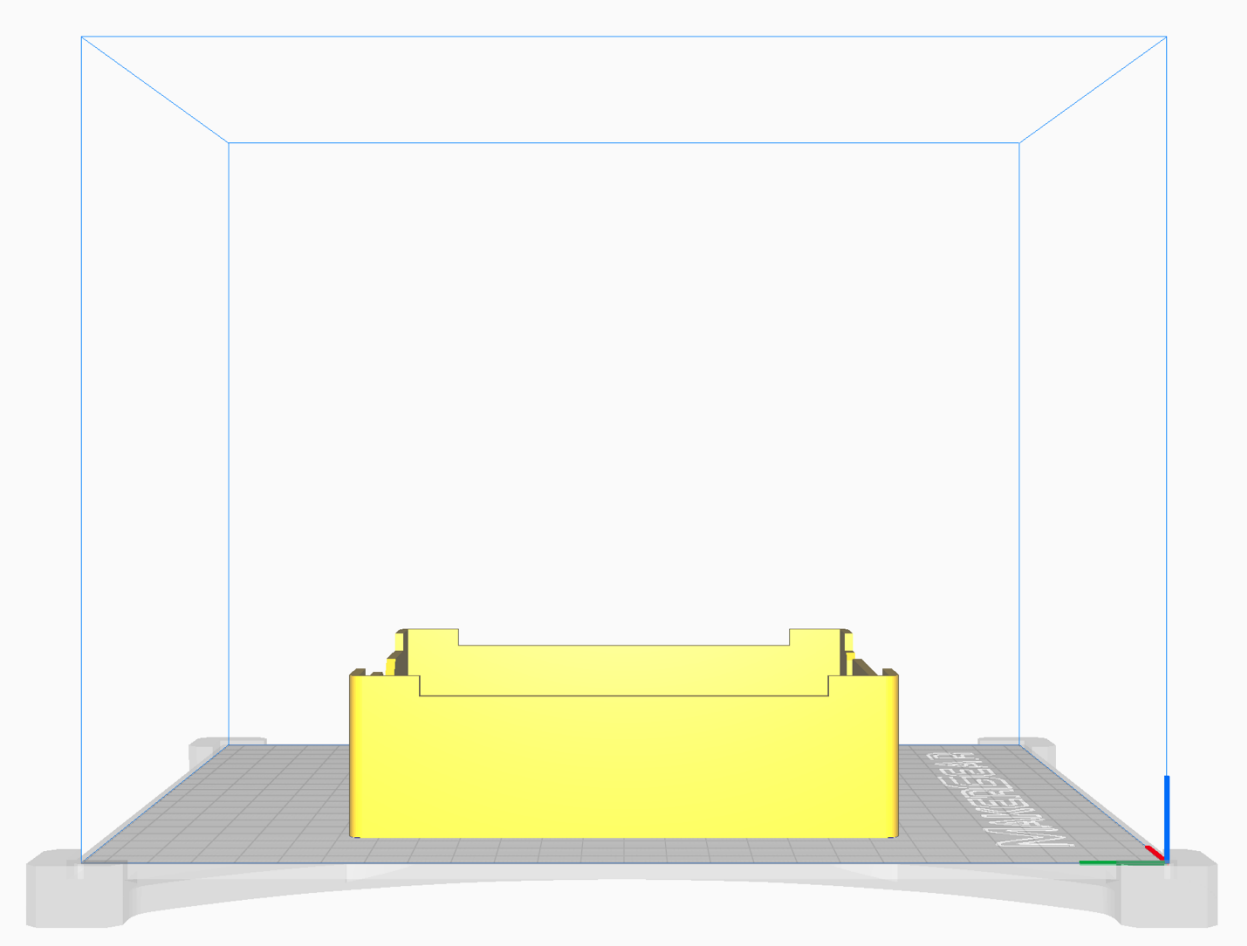
(Figure 26 – Box Base Rev4)



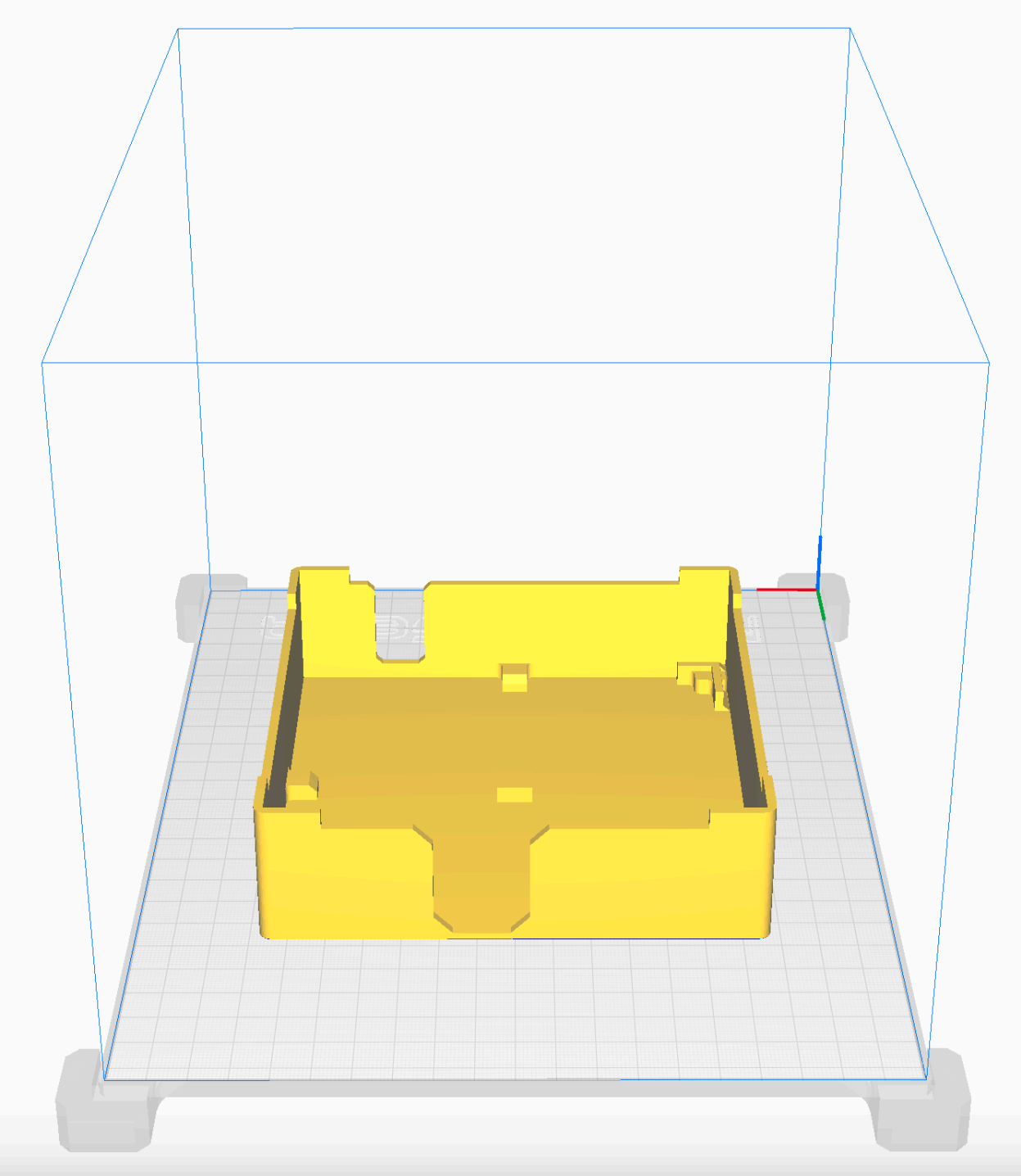
(Figure 27 – Box Base Rev4 Front View)



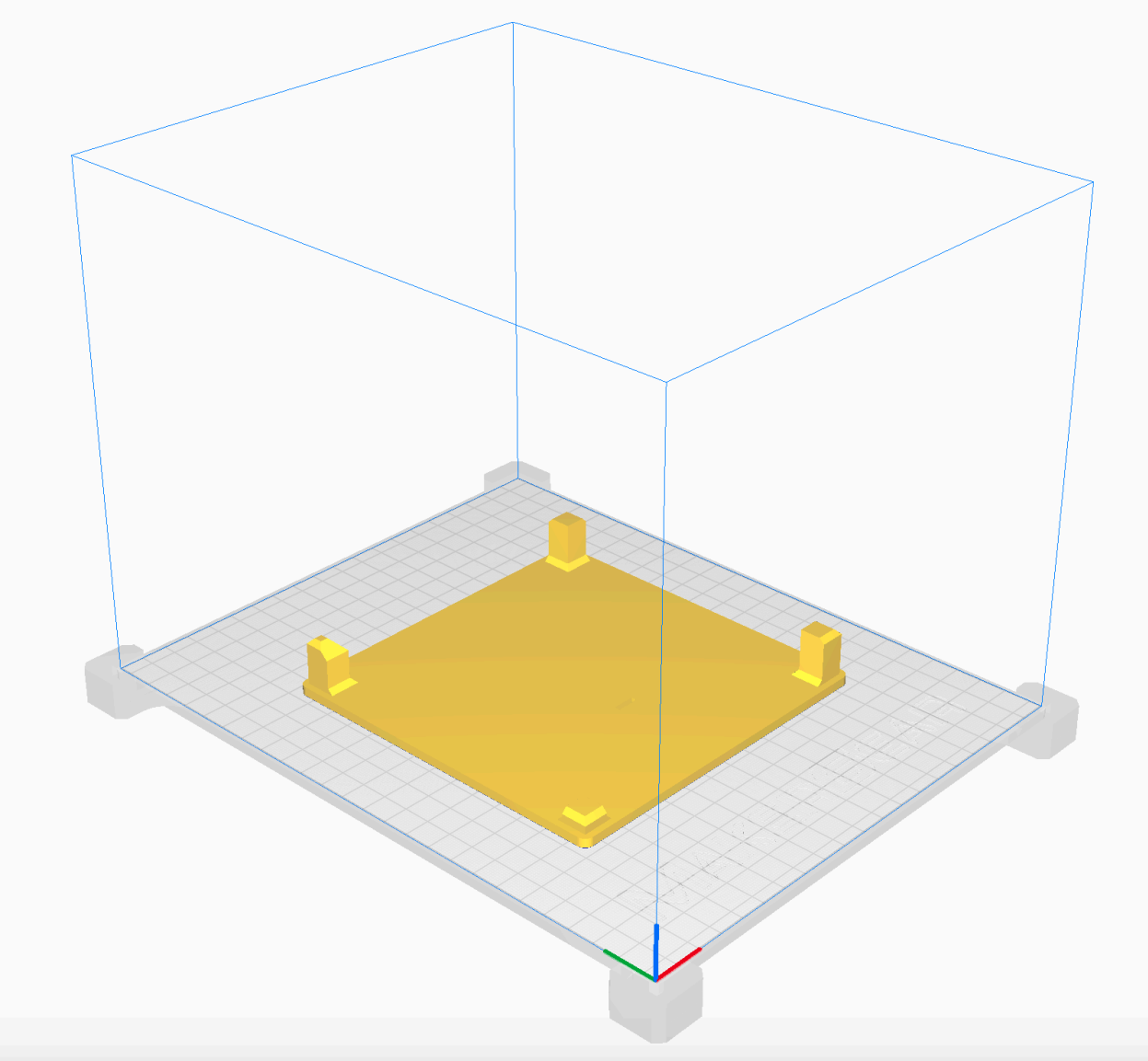
(Figure 28 – Box Base Rev4 Top View)



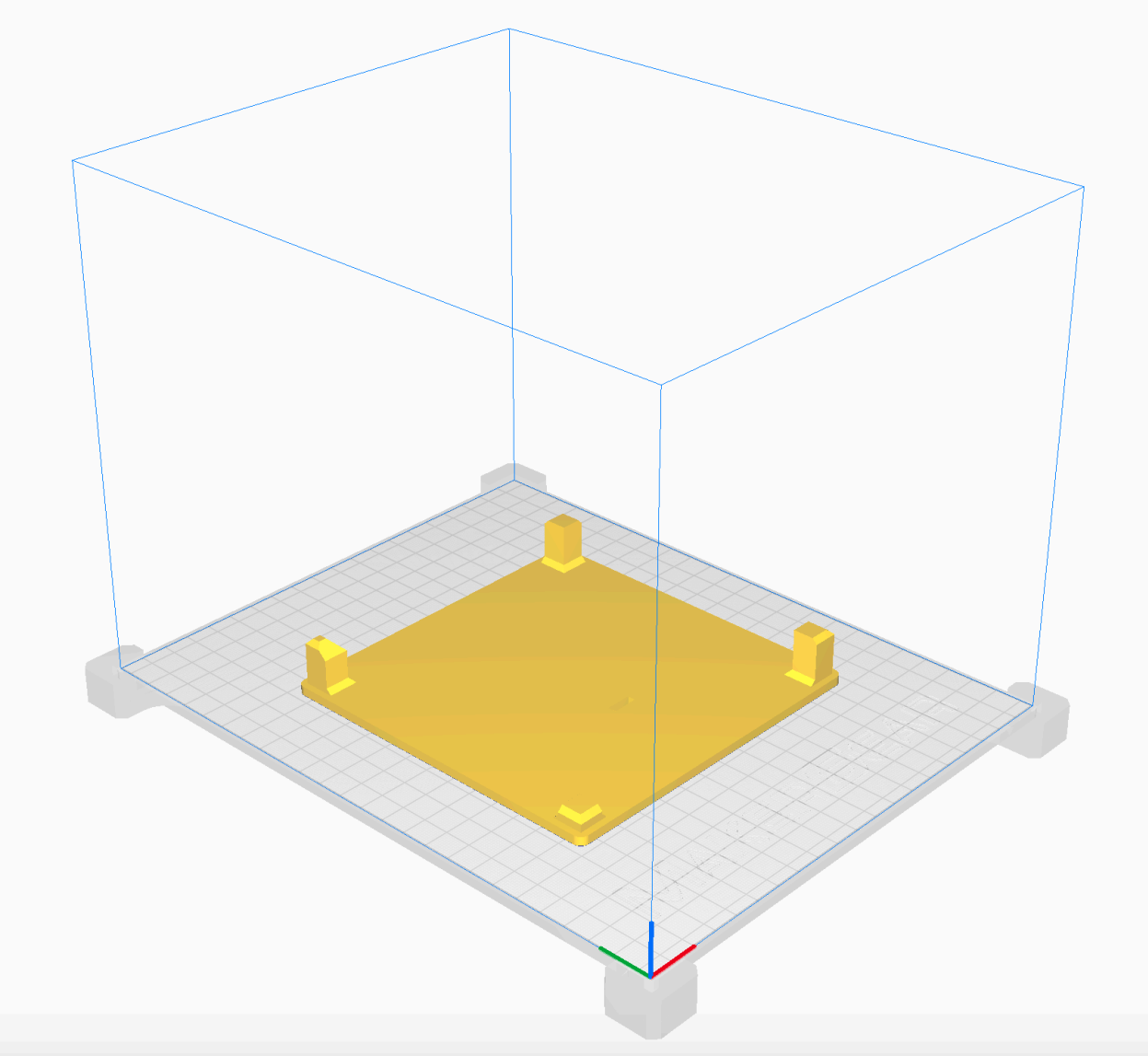
(Figure 29 – Box Base Rev4 Side View)



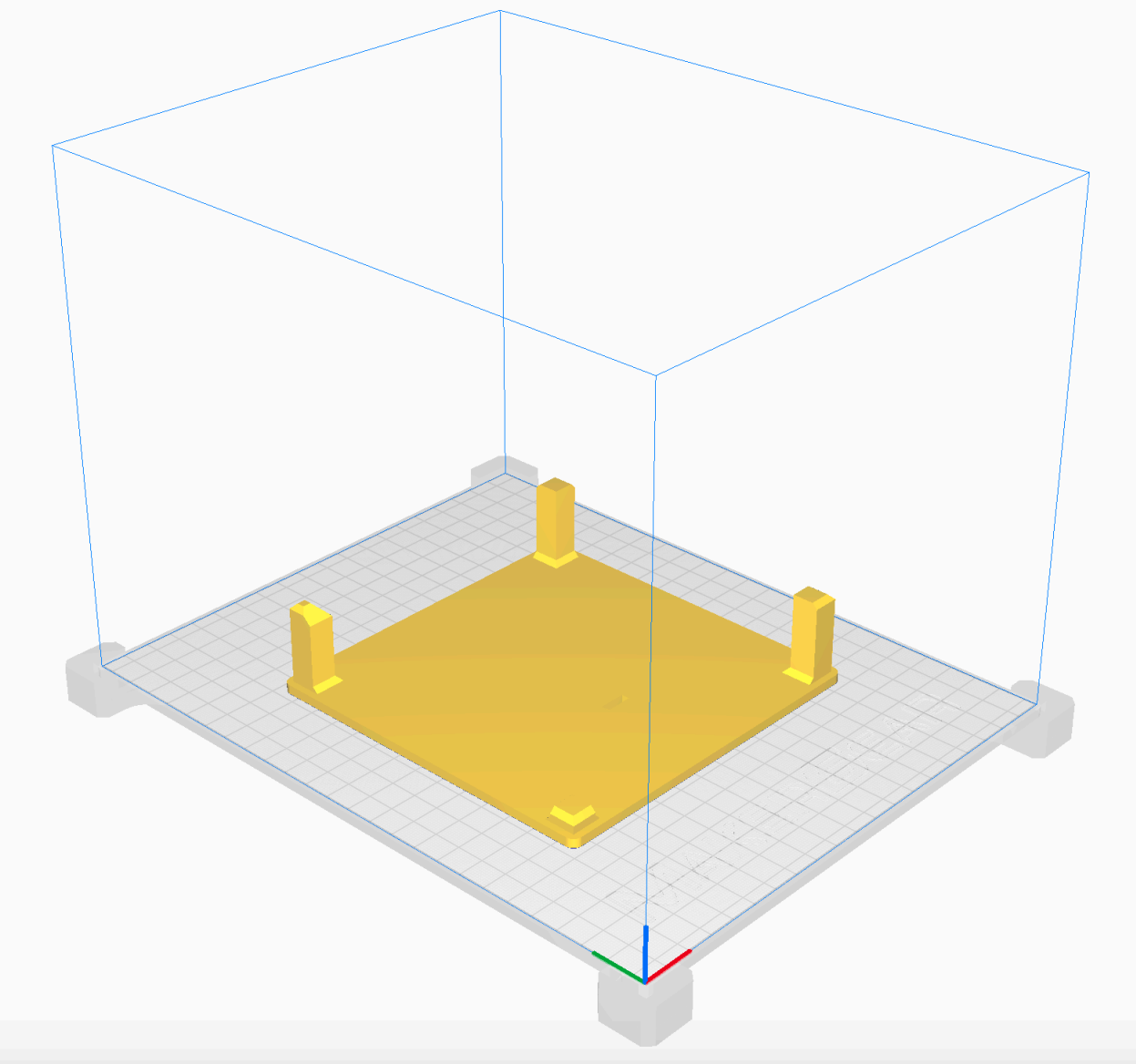
(Figure 30 – Box Base Rev4 Rear View)



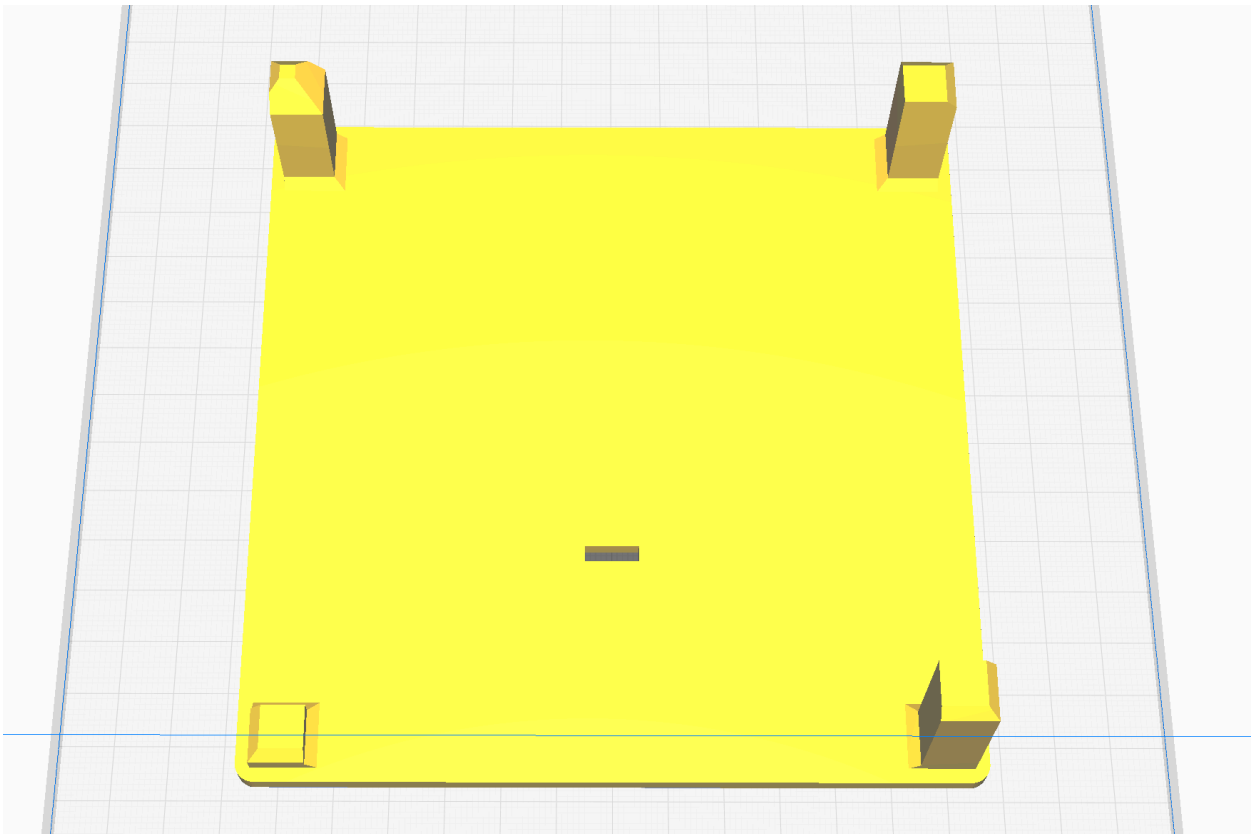
(Figure 31 – Box Lid Rev1)



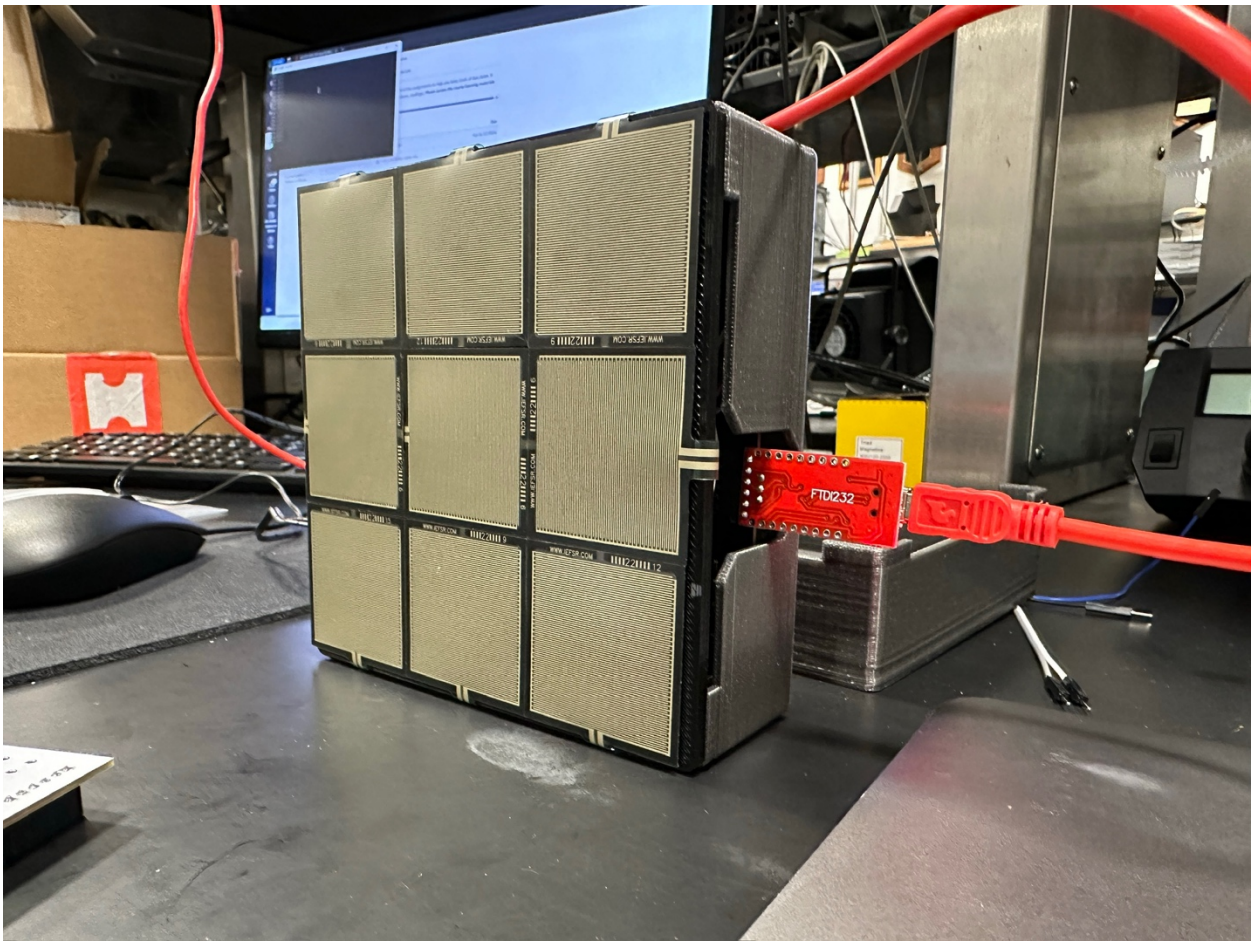
(Figure 32 – Box Lid Rev2)



(Figure 33 – Box Lid Rev3)



(Figure 34 – Box Lid Rev3 Top View)



(Figure 35 – Final Product)

Appendix

main.c:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

#include <math.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
```

```
UART_HandleTypeDef hlpuart1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_LPUART1_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_LPUART1_UART_Init();
```

```
  /* USER CODE BEGIN 2 */
//  HAL_UART_Transmit(&huart1,msg1,sizeof(msg1),1000);
//  HAL_UART_Transmit(&huart1,msg2,sizeof(msg2),1000);
//  BSP_ENV_SENSOR_Init(0, ENV_FORCE);
//  BSP_ENV_SENSOR_Enable(0, ENV_FORCE);
//  HAL_UART_Transmit(&huart1,msg3,sizeof(msg3),1000);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

//    BSP_ENV_SENSOR_GetValue(0, ENV_FORCE, &force_value);
//    int forceInt1 = force_value;
//    float forceFrac = force_value - tmpInt1;
//    int forceInt2 = trunc(tmpFrac * 100);
//    snprintf(str_tmp,100," FORCE = %d.%02d\n", forceInt1, forceInt2);
//    HAL_UART_Transmit(&huart1,( uint8_t *)str_tmp,sizeof(str_tmp),500);

    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) != 0){

    /* USER CODE END WHILE */
    uint8_t Test[] = "Hit 3!!!\r\n"; //Data to send
    HAL_UART_Transmit(&hlpuart1,Test,sizeof(Test),10);// Sending in normal mode
    HAL_Delay(500);

    /* USER CODE BEGIN 3 */
    }

    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1) != 0){

    /* USER CODE END WHILE */
    uint8_t Test[] = "Hit 2!!!\r\n"; //Data to send
    HAL_UART_Transmit(&hlpuart1,Test,sizeof(Test),10);// Sending in normal mode
    HAL_Delay(500);

    /* USER CODE BEGIN 3 */
    }

    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_4) != 0){

    /* USER CODE END WHILE */
    uint8_t Test[] = "Hit 1!!!\r\n"; //Data to send
    HAL_UART_Transmit(&hlpuart1,Test,sizeof(Test),10);// Sending in normal mode
```

```
HAL_Delay(500);

/* USER CODE BEGIN 3 */
}

if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_5) != 0){

/* USER CODE END WHILE */
uint8_t Test[] = "Hit 6!!!\r\n"; //Data to send
HAL_UART_Transmit(&hlpuart1,Test,sizeof(Test),10);// Sending in normal mode
HAL_Delay(500);

/* USER CODE BEGIN 3 */
}

if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_6) != 0){

/* USER CODE END WHILE */
uint8_t Test[] = "Hit 9!!!\r\n"; //Data to send
HAL_UART_Transmit(&hlpuart1,Test,sizeof(Test),10);// Sending in normal mode
HAL_Delay(500);

/* USER CODE BEGIN 3 */
}

if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_7) != 0){

/* USER CODE END WHILE */
uint8_t Test[] = "Hit 8!!!\r\n"; //Data to send
HAL_UART_Transmit(&hlpuart1,Test,sizeof(Test),10);// Sending in normal mode
HAL_Delay(500);

/* USER CODE BEGIN 3 */
}

if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8) != 0){

/* USER CODE END WHILE */
uint8_t Test[] = "Hit 4!!!\r\n"; //Data to send
HAL_UART_Transmit(&hlpuart1,Test,sizeof(Test),10);// Sending in normal mode
HAL_Delay(500);

/* USER CODE BEGIN 3 */
}

if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_9) != 0){

/* USER CODE END WHILE */
```

```
    uint8_t Test[] = "Hit 5!!!\r\n"; //Data to send
    HAL_UART_Transmit(&hlpuart1,Test,sizeof(Test),10);// Sending in normal mode
    HAL_Delay(500);

    /* USER CODE BEGIN 3 */
    }

    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_10) != 0){

    /* USER CODE END WHILE */
    uint8_t Test[] = "Hit 7!!!\r\n"; //Data to send
    HAL_UART_Transmit(&hlpuart1,Test,sizeof(Test),10);// Sending in normal mode
    HAL_Delay(500);

    /* USER CODE BEGIN 3 */
    }


  }




        //HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_1);
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
```

```
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_5;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_LPUART1;
  PeriphClkInit.Lpuart1ClockSelection = RCC_LPUART1CLKSOURCE_PCLK1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief LPUART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_LPUART1_UART_Init(void)
{

  /* USER CODE BEGIN LPUART1_Init 0 */

  /* USER CODE END LPUART1_Init 0 */

  /* USER CODE BEGIN LPUART1_Init 1 */

  /* USER CODE END LPUART1_Init 1 */
  hlpuart1.Instance = LPUART1;
  hlpuart1.Init.BaudRate = 19200;
  hlpuart1.Init.WordLength = UART_WORDLENGTH_8B;
  hlpuart1.Init.StopBits = UART_STOPBITS_1;
  hlpuart1.Init.Parity = UART_PARITY_NONE;
```

```
  hlpuart1.Init.Mode = UART_MODE_TX_RX;
  hlpuart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  hlpuart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  hlpuart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&hlpuart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN LPUART1_Init 2 */

  /* USER CODE END LPUART1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pins : PA0 PA1 PA4 PA5
                          PA6 PA7 PA8 PA9
                          PA10 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_5
                        |GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9
                        |GPIO_PIN_10;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
```

```
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

stm32l0xx_hal_msp.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file         stm32l0xx_hal_msp.c
  * @brief        This file provides code for the MSP Initialization
  *               and de-Initialization codes.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN TD */

/* USER CODE END TD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN Define */

/* USER CODE END Define */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN Macro */

/* USER CODE END Macro */

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
```

```c
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* External functions --------------------------------------------------------*/
/* USER CODE BEGIN ExternalFunctions */

/* USER CODE END ExternalFunctions */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */
/**
  * Initializes the Global MSP.
  */
void HAL_MspInit(void)
{
  /* USER CODE BEGIN MspInit 0 */

  /* USER CODE END MspInit 0 */

  __HAL_RCC_SYSCFG_CLK_ENABLE();
  __HAL_RCC_PWR_CLK_ENABLE();

  /* System interrupt init*/

  /* USER CODE BEGIN MspInit 1 */

  /* USER CODE END MspInit 1 */
}

/**
* @brief UART MSP Initialization
* This function configures the hardware resources used in this example
* @param huart: UART handle pointer
* @retval None
*/
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==LPUART1)
  {
  /* USER CODE BEGIN LPUART1_MspInit 0 */

  /* USER CODE END LPUART1_MspInit 0 */
```

```
    /* Peripheral clock enable */
    __HAL_RCC_LPUART1_CLK_ENABLE();

    __HAL_RCC_GPIOA_CLK_ENABLE();
    /**LPUART1 GPIO Configuration
    PA2     ------> LPUART1_TX
    PA3     ------> LPUART1_RX
    */
    GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF6_LPUART1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN LPUART1_MspInit 1 */

  /* USER CODE END LPUART1_MspInit 1 */
  }

}

/**
* @brief UART MSP De-Initialization
* This function freeze the hardware resources used in this example
* @param huart: UART handle pointer
* @retval None
*/
void HAL_UART_MspDeInit(UART_HandleTypeDef* huart)
{
  if(huart->Instance==LPUART1)
  {
  /* USER CODE BEGIN LPUART1_MspDeInit 0 */

  /* USER CODE END LPUART1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_LPUART1_CLK_DISABLE();

    /**LPUART1 GPIO Configuration
    PA2     ------> LPUART1_TX
    PA3     ------> LPUART1_RX
    */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_2|GPIO_PIN_3);

  /* USER CODE BEGIN LPUART1_MspDeInit 1 */

  /* USER CODE END LPUART1_MspDeInit 1 */
  }
```

```
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

system_stm32l0xx.c

```
/**
  ******************************************************************************
  * @file    system_stm32l0xx.c
  * @author  MCD Application Team
  * @brief   CMSIS Cortex-M0+ Device Peripheral Access Layer System Source File.
  *
  *   This file provides two functions and one global variable to be called from
  *   user application:
  *      - SystemInit(): This function is called at startup just after reset and
  *                      before branch to main program. This call is made inside
  *                      the "startup_stm32l0xx.s" file.
  *
  *      - SystemCoreClock variable: Contains the core clock (HCLK), it can be used
  *                                  by the user application to setup the SysTick
  *                                  timer or configure other parameters.
  *
  *      - SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must
  *                                 be called whenever the core clock is changed
  *                                 during program execution.
  *
  *
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2016 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */

/** @addtogroup CMSIS
  * @{
  */

/** @addtogroup stm32l0xx_system
  * @{
  */

/** @addtogroup STM32L0xx_System_Private_Includes
  * @{
  */
```

```
#include "stm32l0xx.h"

#if !defined  (HSE_VALUE)
  #define HSE_VALUE    ((uint32_t)8000000U) /*!< Value of the External oscillator in
Hz */
#endif /* HSE_VALUE */

#if !defined  (MSI_VALUE)
  #define MSI_VALUE    ((uint32_t)2097152U) /*!< Value of the Internal oscillator in
Hz*/
#endif /* MSI_VALUE */

#if !defined  (HSI_VALUE)
  #define HSI_VALUE    ((uint32_t)16000000U) /*!< Value of the Internal oscillator in
Hz*/
#endif /* HSI_VALUE */


/**
  * @}
  */

/** @addtogroup STM32L0xx_System_Private_TypesDefinitions
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32L0xx_System_Private_Defines
  * @{
  */
/************************* Miscellaneous Configuration ************************/

/* Note: Following vector table addresses must be defined in line with linker
         configuration. */
/*!< Uncomment the following line if you need to relocate the vector table
     anywhere in Flash or Sram, else the vector table is kept at the automatic
     remap of boot address selected */
/* #define USER_VECT_TAB_ADDRESS */

#if defined(USER_VECT_TAB_ADDRESS)
/*!< Uncomment the following line if you need to relocate your vector Table
     in Sram else user remap will be done in Flash. */
/* #define VECT_TAB_SRAM */
#if defined(VECT_TAB_SRAM)
```

```
#define VECT_TAB_BASE_ADDRESS   SRAM_BASE       /*!< Vector Table base address field.
                                                     This value must be a multiple of
0x200. */
#define VECT_TAB_OFFSET         0x00000000U     /*!< Vector Table base offset field.
                                                     This value must be a multiple of
0x200. */
#else
#define VECT_TAB_BASE_ADDRESS   FLASH_BASE      /*!< Vector Table base address field.
                                                     This value must be a multiple of
0x200. */
#define VECT_TAB_OFFSET         0x00000000U     /*!< Vector Table base offset field.
                                                     This value must be a multiple of
0x200. */
#endif /* VECT_TAB_SRAM */
#endif /* USER_VECT_TAB_ADDRESS */

/******************************************************************************/
/**
  * @}
  */

/** @addtogroup STM32L0xx_System_Private_Macros
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32L0xx_System_Private_Variables
  * @{
  */
  /* This variable is updated in three ways:
      1) by calling CMSIS function SystemCoreClockUpdate()
      2) by calling HAL API function HAL_RCC_GetHCLKFreq()
      3) each time HAL_RCC_ClockConfig() is called to configure the system clock
frequency
         Note: If you use this function to configure the system clock; then there
               is no need to call the 2 first functions listed above, since
SystemCoreClock
               variable is updated automatically.
  */
  uint32_t SystemCoreClock = 2097152U; /* 32.768 kHz * 2^6 */
  const uint8_t AHBPrescTable[16] = {0U, 0U, 0U, 0U, 0U, 0U, 0U, 0U, 1U, 2U, 3U, 4U,
6U, 7U, 8U, 9U};
  const uint8_t APBPrescTable[8] = {0U, 0U, 0U, 0U, 1U, 2U, 3U, 4U};
  const uint8_t PLLMulTable[9] = {3U, 4U, 6U, 8U, 12U, 16U, 24U, 32U, 48U};
```

```
/**
  * @}
  */

/** @addtogroup STM32L0xx_System_Private_FunctionPrototypes
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32L0xx_System_Private_Functions
  * @{
  */

/**
  * @brief  Setup the microcontroller system.
  * @param  None
  * @retval None
  */
void SystemInit (void)
{
  /* Configure the Vector Table location add offset address ------------------*/
#if defined (USER_VECT_TAB_ADDRESS)
  SCB->VTOR = VECT_TAB_BASE_ADDRESS | VECT_TAB_OFFSET; /* Vector Table Relocation in
Internal SRAM */
#endif /* USER_VECT_TAB_ADDRESS */
}

/**
  * @brief  Update SystemCoreClock variable according to Clock Register Values.
  *         The SystemCoreClock variable contains the core clock (HCLK), it can
  *         be used by the user application to setup the SysTick timer or configure
  *         other parameters.
  *
  * @note   Each time the core clock (HCLK) changes, this function must be called
  *         to update SystemCoreClock variable value. Otherwise, any configuration
  *         based on this variable will be incorrect.
  *
  * @note   - The system frequency computed by this function is not the real
  *           frequency in the chip. It is calculated based on the predefined
  *           constant and the selected clock source:
  *
  *           - If SYSCLK source is MSI, SystemCoreClock will contain the MSI
  *             value as defined by the MSI range.
  *
  *           - If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(*)
```

```
  *
  *           - If SYSCLK source is HSE, SystemCoreClock will contain the
HSE_VALUE(**)
  *
  *           - If SYSCLK source is PLL, SystemCoreClock will contain the
HSE_VALUE(**)
  *             or HSI_VALUE(*) multiplied/divided by the PLL factors.
  *
  *         (*) HSI_VALUE is a constant defined in stm32l0xx_hal.h file (default value
  *             16 MHz) but the real value may vary depending on the variations
  *             in voltage and temperature.
  *
  *         (**) HSE_VALUE is a constant defined in stm32l0xx_hal.h file (default
value
  *              8 MHz), user has to ensure that HSE_VALUE is same as the real
  *              frequency of the crystal used. Otherwise, this function may
  *              have wrong result.
  *
  *         - The result of this function could be not correct when using fractional
  *           value for HSE crystal.
  * @param  None
  * @retval None
  */
void SystemCoreClockUpdate (void)
{
  uint32_t tmp = 0U, pllmul = 0U, plldiv = 0U, pllsource = 0U, msirange = 0U;

  /* Get SYSCLK source -------------------------------------------------------*/
  tmp = RCC->CFGR & RCC_CFGR_SWS;

  switch (tmp)
  {
    case 0x00U:  /* MSI used as system clock */
      msirange = (RCC->ICSCR & RCC_ICSCR_MSIRANGE) >> RCC_ICSCR_MSIRANGE_Pos;
      SystemCoreClock = (32768U * (1U << (msirange + 1U)));
      break;
    case 0x04U:  /* HSI used as system clock */
      if ((RCC->CR & RCC_CR_HSIDIVF) != 0U)
      {
        SystemCoreClock = HSI_VALUE / 4U;
      }
      else
      {
        SystemCoreClock = HSI_VALUE;
      }
      break;
    case 0x08U:  /* HSE used as system clock */
      SystemCoreClock = HSE_VALUE;
```

```
      break;
    default:  /* PLL used as system clock */
      /* Get PLL clock source and multiplication factor ----------------------*/
      pllmul = RCC->CFGR & RCC_CFGR_PLLMUL;
      plldiv = RCC->CFGR & RCC_CFGR_PLLDIV;
      pllmul = PLLMulTable[(pllmul >> RCC_CFGR_PLLMUL_Pos)];
      plldiv = (plldiv >> RCC_CFGR_PLLDIV_Pos) + 1U;

      pllsource = RCC->CFGR & RCC_CFGR_PLLSRC;

      if (pllsource == 0x00U)
      {
        /* HSI oscillator clock selected as PLL clock entry */
        if ((RCC->CR & RCC_CR_HSIDIVF) != 0U)
        {
          SystemCoreClock = (((HSI_VALUE / 4U) * pllmul) / plldiv);
        }
        else
        {
          SystemCoreClock = (((HSI_VALUE) * pllmul) / plldiv);
        }
      }
      else
      {
        /* HSE selected as PLL clock entry */
        SystemCoreClock = (((HSE_VALUE) * pllmul) / plldiv);
      }
      break;
  }
  /* Compute HCLK clock frequency ----------------------------------------------*/
  /* Get HCLK prescaler */
  tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) >> RCC_CFGR_HPRE_Pos)];
  /* HCLK clock frequency */
  SystemCoreClock >>= tmp;
}



/**
  * @}
  */

/**
  * @}
  */
```

sysmem.c

```
/**
 ******************************************************************************
 * @file      sysmem.c
 * @author    Generated by STM32CubeIDE
 * @brief     STM32CubeIDE System Memory calls file
 *
 *            For more information about which C functions
 *            need which of these lowlevel functions
 *            please consult the newlib libc manual
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2023 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */

/* Includes */
#include <errno.h>
#include <stdint.h>

/**
 * Pointer to the current high watermark of the heap usage
 */
static uint8_t *__sbrk_heap_end = NULL;

/**
 * @brief _sbrk() allocates memory to the newlib heap and is used by malloc
 *        and others from the C library
 *
 * @verbatim
 * ############################################################################
 * #  .data  #  .bss  #       newlib heap       #          MSP stack          #
 * #         #        #                         # Reserved by _Min_Stack_Size #
 * ############################################################################
 * ^-- RAM start      ^-- _end                             _estack, RAM end --^
 * @endverbatim
 *
 * This implementation starts allocating at the '_end' linker symbol
 * The '_Min_Stack_Size' linker symbol reserves a memory for the MSP stack
```

```
 * The implementation considers '_estack' linker symbol to be RAM end
 * NOTE: If the MSP stack, at any point during execution, grows larger than the
 * reserved size, please increase the '_Min_Stack_Size'.
 *
 * @param incr Memory size
 * @return Pointer to allocated memory
 */
void *_sbrk(ptrdiff_t incr)
{
  extern uint8_t _end; /* Symbol defined in the linker script */
  extern uint8_t _estack; /* Symbol defined in the linker script */
  extern uint32_t _Min_Stack_Size; /* Symbol defined in the linker script */
  const uint32_t stack_limit = (uint32_t)&_estack - (uint32_t)&_Min_Stack_Size;
  const uint8_t *max_heap = (uint8_t *)stack_limit;
  uint8_t *prev_heap_end;

  /* Initialize heap end at first call */
  if (NULL == __sbrk_heap_end)
  {
    __sbrk_heap_end = &_end;
  }

  /* Protect heap from growing into the reserved MSP stack */
  if (__sbrk_heap_end + incr > max_heap)
  {
    errno = ENOMEM;
    return (void *)-1;
  }

  prev_heap_end = __sbrk_heap_end;
  __sbrk_heap_end += incr;

  return (void *)prev_heap_end;
}
```

syscalls.c

```c
/**
 ******************************************************************************
 * @file      syscalls.c
 * @author    Auto-generated by STM32CubeIDE
 * @brief     STM32CubeIDE Minimal System calls file
 *
 *            For more information about which c-functions
 *            need which of these lowlevel functions
 *            please consult the Newlib libc-manual
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2020-2023 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */

/* Includes */
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>
#include <sys/times.h>


/* Variables */
extern int __io_putchar(int ch) __attribute__((weak));
extern int __io_getchar(void) __attribute__((weak));


char *__env[1] = { 0 };
char **environ = __env;


/* Functions */
void initialise_monitor_handles()
{
```

```
}

int _getpid(void)
{
  return 1;
}

int _kill(int pid, int sig)
{
  (void)pid;
  (void)sig;
  errno = EINVAL;
  return -1;
}

void _exit (int status)
{
  _kill(status, -1);
  while (1) {}    /* Make sure we hang here */
}

__attribute__((weak)) int _read(int file, char *ptr, int len)
{
  (void)file;
  int DataIdx;

  for (DataIdx = 0; DataIdx < len; DataIdx++)
  {
    *ptr++ = __io_getchar();
  }

  return len;
}

__attribute__((weak)) int _write(int file, char *ptr, int len)
{
  (void)file;
  int DataIdx;

  for (DataIdx = 0; DataIdx < len; DataIdx++)
  {
    __io_putchar(*ptr++);
  }
  return len;
}

int _close(int file)
{
```

```
  (void)file;
  return -1;
}


int _fstat(int file, struct stat *st)
{
  (void)file;
  st->st_mode = S_IFCHR;
  return 0;
}

int _isatty(int file)
{
  (void)file;
  return 1;
}

int _lseek(int file, int ptr, int dir)
{
  (void)file;
  (void)ptr;
  (void)dir;
  return 0;
}

int _open(char *path, int flags, ...)
{
  (void)path;
  (void)flags;
  /* Pretend like we always fail */
  return -1;
}

int _wait(int *status)
{
  (void)status;
  errno = ECHILD;
  return -1;
}

int _unlink(char *name)
{
  (void)name;
  errno = ENOENT;
  return -1;
}
```

```c
int _times(struct tms *buf)
{
  (void)buf;
  return -1;
}

int _stat(char *file, struct stat *st)
{
  (void)file;
  st->st_mode = S_IFCHR;
  return 0;
}

int _link(char *old, char *new)
{
  (void)old;
  (void)new;
  errno = EMLINK;
  return -1;
}

int _fork(void)
{
  errno = EAGAIN;
  return -1;
}

int _execve(char *name, char **argv, char **env)
{
  (void)name;
  (void)argv;
  (void)env;
  errno = ENOMEM;
  return -1;
}
```

stm32l0xx_it.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file    stm32l0xx_it.c
  * @brief   Interrupt Service Routines.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32l0xx_it.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN TD */

/* USER CODE END TD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */

/* USER CODE END PV */
```

```
/* Private function prototypes -----------------------------------------------*/
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/* External variables --------------------------------------------------------*/

/* USER CODE BEGIN EV */

/* USER CODE END EV */

/******************************************************************************/
/*           Cortex-M0+ Processor Interruption and Exception Handlers          */
/******************************************************************************/
/**
  * @brief This function handles Non maskable Interrupt.
  */
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
  while (1)
  {
  }
  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
  * @brief This function handles Hard fault interrupt.
  */
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_HardFault_IRQn 0 */
    /* USER CODE END W1_HardFault_IRQn 0 */
  }
```

```
}

/**
  * @brief This function handles System service call via SWI instruction.
  */
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVC_IRQn 0 */

  /* USER CODE END SVC_IRQn 0 */
  /* USER CODE BEGIN SVC_IRQn 1 */

  /* USER CODE END SVC_IRQn 1 */
}

/**
  * @brief This function handles Pendable request for system service.
  */
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */

  /* USER CODE END PendSV_IRQn 0 */
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}

/**
  * @brief This function handles System tick timer.
  */
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

/******************************************************************************/
/* STM32L0xx Peripheral Interrupt Handlers                                    */
/* Add here the Interrupt Handlers for the used peripherals.                  */
/* For the available peripheral interrupt handler names,                      */
/* please refer to the startup file (startup_stm32l0xx.s).                    */
/******************************************************************************/
```

```
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```