Dedication

A special feeling of gratitude to my loving mother, **Halima**, whose words of encouragement and push for tenacity ring in my ears.

To my sisters **Imen**, **Shayma** and my brother **Wadid** who first taught me the value of education and critical thought. They are the ultimate role models.

I dedicate this work and give special thanks to my Academic Supervisor **Dr. Bouraoui MAHMOUD** and Industrial Supervisor **Mr. Hamdi BEN AMOR** for being there for me throughout the entire process and helped develop and shape my entire skill set.

I consider this humble work as a tribute to all my friends and specially those who have been my greatest support in my academic years as well as my personal life.

Acknowledgement

This work would have never been possible without the help and guidance of many people with whom I have had the pleasure to work during this project.

First and foremost, I would like to express my greatest gratitude **to Mr. Hamdi BEN AMOR**, my industrial supervisor at Micro Device Tunisie, for providing guidance and feedback throughout this project and for teaching me a great deal along my path and that inspired me to reach my full potential.

I would like to sincerely thank **Dr. Bouraoui MAHMOUD**, my academic supervisor for his guidance and support throughout this entire internship and especially for his confidence in me.

Finally, I would like to thank the jury members for putting the time and effort in evaluating this humble work.

Abstract

We are entering a brand-new era of computing technology known as the Internet of Things (IoT) and its potential is extremely important towards technological advancement and will eventually encompass every aspect of our lives. Its main foundation, in a way, is the simple yet becoming very essential, intelligence that embedded devices and embedded processing provides.

The Internet of Things represents a vast market for many worldwide known pioneers, among those we shall not pass without mentioning Espressif Systems, the multinational semiconductor company that have created the popular ESP series of chips and most importantly the ESP32. Powerful Wi-Fi and Bluetooth modules that target a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks. While this series of chips are widely used, unfortunately there still no proper solution to how to program it and/or debug it considering professional and industrial tools.

The project presented in this document and proposed by the company Micro Device Tunisie is developed for Programming and debugging these chips in real time. Our focus is to deliver both a hardware and software solution not only capable of what is mentioned before but also well designed and suited for industrial purposes.

Keywords: ESP32, Hardware Design, PyQt, Software Development, 3D Design.

Résumé

Nous entrons dans une toute nouvelle ère de technologie informatique appelée Internet des objets (IoT) et son potentiel est extrêmement important pour le progrès technologique et englobera tous les aspects de notre vie. D'une certaine manière, son fondement repose sur l'intelligence simple, mais essentielle, que fournissent les appareils et les systèmes embarqués.

L'Internet des objets représente un vaste marché pour de nombreuses entreprises mondialement connues, parmi ceux que nous ne passerons pas sans mentionner Espressif Systems, la multinationale des semi-conducteurs qui a créé la série de puces populaires ESP, et plus particulièrement le ESP32. De puissants modules Wi-Fi et Bluetooth destinés à une grande variété d'applications, allant des réseaux de capteurs à faible consommation aux tâches les plus exigeantes. Bien que cette série de puces soit largement utilisée, malheureusement, il n'existe toujours pas de solution adéquate pour la programmer et / ou la configurer en tenant compte des outils professionnels et industriels.

Le projet présenté dans ce document et proposé par la société Micro Device Tunisie est développé pour la programmation et le débogage de ces puces en temps réel. Notre objectif est de fournir à la fois une solution matérielle et logicielle capable non seulement de ce qui a été mentionné précédemment, mais également bien conçue et adaptée aux besoins industriels.

Mots clés : ESP32, conception électronique, PyQt, Développement de logiciels, Conception 3D.

Contents

Chapter I : Bibliographic study

Chapter II : Hardware solution development &3D design

Chapter III : Software solution development

List of tables

List of figures

List of Abbreviations

- **2D:** 2-Dimensional
- **3D:** 3-Dimensional
- **API**: Application Program Interface
- **Bps**: Bytes per Second
- **CAD**: Computer-Aided Design
- **CS**: Chip Select
- **CLK**: Clock
- **CSS**: Cascading Style Sheets
- **DI**: Data In
- **DO**: Data Out
- **DTR**: Data Terminal Ready
- **EEPROM**: Electrically Erasable Programmable Read Only Memory
- **ESD**: Electrostatic Discharge
- **EN**: Enable
- **EMI**: Electromagnetic interference
- **FIFO**: first-in, first-out
- **FTDI**: Future Technology Devices International
- **GUI**: Graphical User Interface
- **GPIO**: General Purpose Input/output
- **GND**: Ground
- **IDE**: integrated development environment
- **IC**: Integrated Circuit
- **IO**: Input Output
- **IoT**: Internet of Things
- **JTAG**: Joint Test Action Group
- **LED**: light emitting diode
- **MPSSE**: Multi-Protocol Synchronous Serial Engine
- **MCU**: Micro Controller Unit
- **MDT**: Micro Device Tunisia

- **Mb**/s: megabyte per second
- **MHz**: megahertz
- **Mac**: Media Access Control
- **OSCI**: Oscillator Input
- **OSCO**: Oscillator Output
- **PCB**: Printed Circuit Board
- **pF**: picofarads
- **QSS**: Qt Style Sheet
- **RST**: Reset
- **RS232**: Recommended Standard 232
- **RTS**: Request to Send
- **ROM**: read-only memory
- **RX**: Receiver
- **RAM**: random access memory
- **SMT**: Surface-mount technology
- **SoC**: System on Chip
- **ST**: STMicroelectronics
- **TVS**: transient-voltage-suppression
- **TX**: Transmitter
- **USB**: Universal Serial Bus
- **UART**: Universal Asynchronous Receiver-Transmitter
- **VCC**: voltage at the common collect

General introduction

Discerning the remarkable synergy between the field of embedded systems and the elegant internet of things was the essential stimuli behind the idea of this project.

IoT solutions have largely had to be built from scratch with a high degree of customization to specific requirements, which has driven up the cost and complexity of development and deterred many prospective entrants to the market.

What have been missing are development tools that alleviate the costs associated with building the foundational infrastructure of their solutions, so they can focus on optimizing the core functionality and bring solutions to market more quickly with less cost.

This end of studies project, entitled “Programmer and Debugger for ESP32”, represents the backbone and a cornerstone of a much larger merger between both fields of embedded systems and the Internet of Things to design and create even more innovative products.

Chapter I

Bibliographic study

Introduction

The subject covered in this chapter is required to understand the following chapters. It covers aspects related to the embedded systems, information about the chip (ESP32) which the project targets, as well as existing solutions on the market, followed by the host company's work within the intersection of these two fields. Finally, we'll go through the project description and specifications as the last step in this chapter.

1. Preliminary Study

1. ESP32 chip

ESP32 is a series of low-cost and low-power SoC microcontrollers . It integrates Wi-Fi and dual-mode Bluetooth technologies . Equipped with a Tensilica Xtensa LX6 microprocessor and includes in-built antenna switches ,power amplifier, low-noise receive amplifier, filters, and power-management modules. ESP32 is created and developed by Espressif Systems, a Shanghai-based Chinese company.[1]

Figure I.1: ESP-WROOM-32 Module

ESP32 is capable of functioning reliably in industrial environments. Engineered for mobile devices, wearable electronics and IoT applications, ESP32 achieves ultra-low power consumption with a combination of several types of proprietary software. ESP32 can perform as a complete standalone system or as a slave device to a host MCU, reducing communication stack overhead on the main application processor.

[1] Source: https://www.espressif.com/en/products/hardware/esp-wroom-32/overview/

Bootloader [2] is a piece of code that runs before any operating system is running. The ESP Bootloader, is located at the address 0x1000 in the flash, performs the following functions:

- Minimal initial configuration of internal modules.
- Select the application partition to boot based on the partition table.
- Load the image to RAM and transfer management to it.

2. Existing Solutions

Up until now, as shown in the figure I.2, there has been all sorts of unprofessional and none practical techniques and hacks involved in programming an ESP32 chip. But never an actual hardware solution mainly designed for industrial purposes.

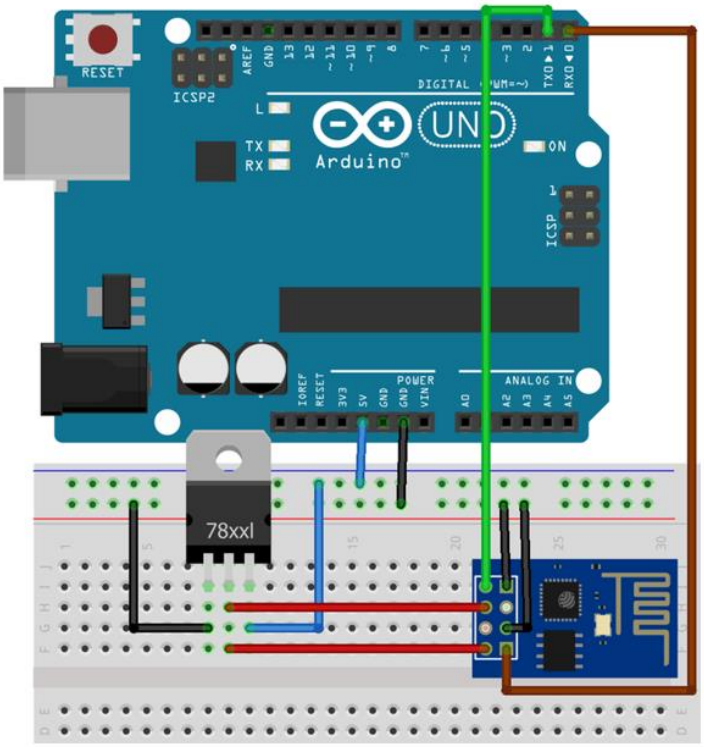


Figure I.2: Programming ESP8266 with Arduino UNO [3]

On the other hand, only the software part of the solution exists. “Flash Download Tool”, a desktop application made by Espressif Systems based on esptool libraries and python GUI API. However, this software is poorly designed which resulted in an unsatisfying user experience.

[2] Source: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/bootloader.html/
[3] Source: https://www.espressif.com/en/products/hardware/esp-wroom-32/overview/

The features offered by this application are very minimalistic since it does not implement all major features of the esptool library. Hence, one cannot pass without concluding that this piece of software was clearly designed as a second thought for windows users. Thus, making it unpractical for most production environments and severely penalize developer's productivity.

Micro Device Tunisie is addressing these challenges with new solution that have the potential to expand the market for IoT by organizing, simplifying and reducing the complexity of manufacturing and configuring ESP32 chips.

3. Tools and Software

The major two applications used in designing this project were Altium Designer and Solidworks:

- **Altium Designer:** Altium Designer is a PCB and electronic design automation software package for printed circuit boards. Using Altium Designer as the tool to design the electronic system was the ultimate choice Since it's a very powerful and practical tool to work with and considered the one which most professionals use for PCB design and such. [4]
- **SolidWorks:** SolidWorks is a solid modeling CAD program published by Dassault Systems. SolidWorks helps in creating 2D or 3D solid models without any complexity, faster and in the cost-effective way. It is very easy to use, introduces simple graphics user interface as compared with other CAD solid modeling software [5]

2. Host company: Micro Device Tunisie

Facing a great deal of challenges and seizing massive opportunities in the intersections of IoT and Embedded systems, MDT has always been thinking beyond the ordinary ways and providing precise and carefully tailored solutions.

Figure I.3 : Micro Device Tunisie's Logo

[4] Source: https://www.altium.com/altium-designer/
[5] Source: https://www.solidworks.com/

Micro Device Tunisie (MDT) is a young startup established in 2014, in Sousse, Tunisia. [6] MDT provides service to the industrial sector such as studying, designing and prototyping product in the launch phase or improvement and optimization. The company's work aims at both Hardware and Software Solutions and the interaction between them.

3. Project description

This project was proposed by the host company, Micro Device Tunisie, and it mainly consists of designing and creating a prototype of an electronic device fully capable of connecting the ESP32 based modules to a computer in order to program its memory, flash a new firmware / Bootloader into it and/or perform some debugging procedures.

Meanwhile, the project also comprises a major part in which the development of a piece of software is needed. the software is supposed to simplify the firmware flashing procedure inside industrial environment by presenting two operating modes. First mode is the simplest and it is designed to be used by operators in production departments whereas an advanced mode is introduced for more complex configurations.

The figure I.4 shows the project diagram where the left block indicates the Hardware part and the right one shows the software and its main functionalities.

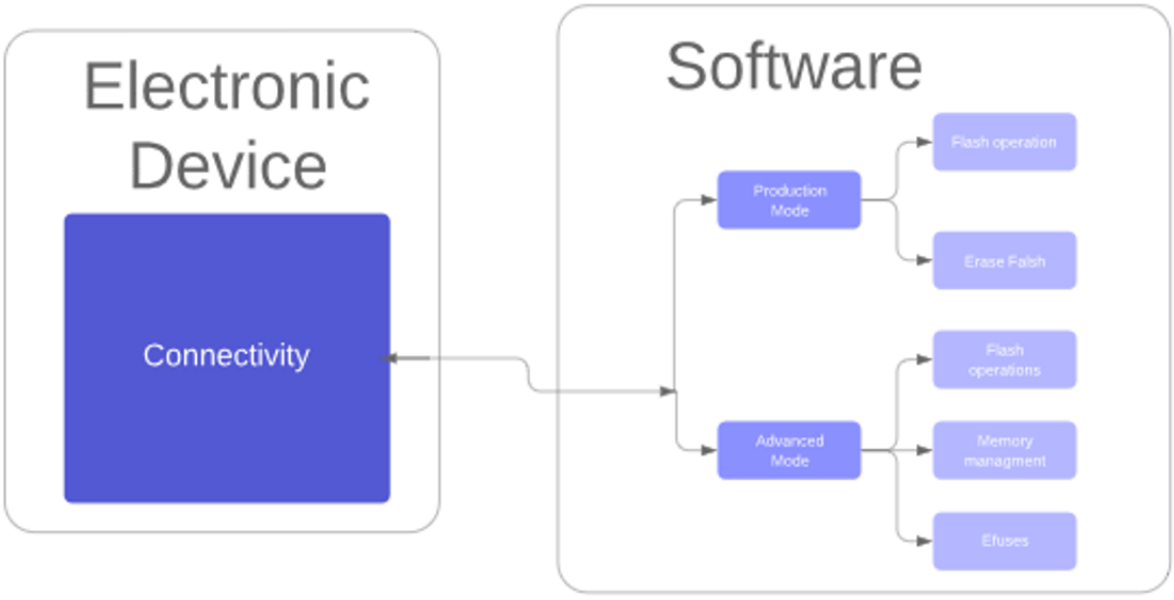


Figure I.4: Project functionalities diagram [7]

[6] Source: http://mdtunisie.com/
[7] Source: https://www.draw.io/

4. Project specifications

For us to describe such large project efficiently, we need to break it down into multiple parts. Therefore, we were requested to execute the following:

- Study the possible solution for the problem
- Draft system architecture (Blocks diagram)
- Choose Components, create and store them in database for further use.
- Create and optimize schematic
- Move to PCB layout and 3D Model hand in hand.
- Design user interface and implement all necessary functionalities.

Conclusion

This first chapter presented a brief introduction to the target chip of our work followed by shedding the light on the existing solutions available on the market. We have had the opportunity also to highlight the host company, Micro Device Tunisie, and its field of work and important technological contributions in those respected fields.

Last but not least, we went through a brief description of the project in hand followed by the specifications.

Chapter II

Hardware solution development & 3D design

Introduction

Unlike the previous chapter, in which we had gone through the introductory phase and the main idea addressed by the project. During this second chapter will define the hardware architecture of the soon to be presented product and will eventually cover the 3-dimensional design for the product and the major alterations made throughout the process.

1. Hardware solution development

1.1. Electronic system architecture

The Figure II.1 describes the general architecture of the desired electronic system. It consists of six major building blocks:

- **Power Supply and USB communication port:** for powering the device the most flexible and suitable way to do so is by using the USB Bus powered configuration which gets its power from the USB bus (+5V) and since no heavy load will be connected to the input/output of the system an external power supply won't be necessary, especially when it's more convenient and practical to use with a computer.

- **ESD Protection circuit:** Although the system is to be used inside industrial sites in most of which ESD protection tools and protocols are already taken into consideration. We went an extra mile here to include an ESD protection circuit both at the input and the output of the system to limit any kind of electrostatic discharge from damaging the vital parts of the circuit from both ends.

- **USB to UART/JTAG Communication:** this by far is the most intuitive block in the system diagram. In order to go from USB communication protocol to UART and/or JTAG we need a chip to do so. We will discuss which chip we are going to choose for this task in the following section.

- **Complementary Blocks:** Meanwhile some blocks in the system would need complementary circuits in order to fully function with an extremely small chance of failing. Among those, we mention external oscillators, bypass capacitors, filters, pull-ups etc.…

- **Hardware Interface:** For us to switch between the ESP chip modes we need a two-switch circuit which will allow for entering the boot mode and reset functionalities by following a sequence of clicks. We will introduce a more sophisticated circuit for that purpose in the coming sections.

- **I/O Connectivity Ports:** as intuitive as it might seem the system needs a bidirectional port to be able to connect and communicate with the target ESP chip and in some cases to power it too.

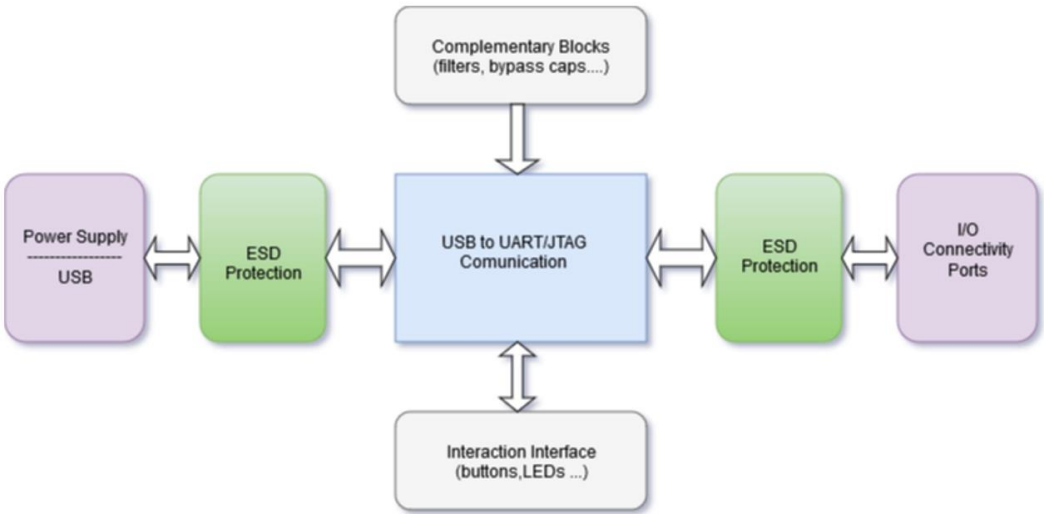


Figure II.1: Electronic System Architecture

1.2. Component selection

Aside from being the longest task during this project, it is also a critical step towards a well-designed electronic system. Throughout this phase we had gone through a variety of components and manufacturers in order to get to the most convenient and efficient set of components.

In this section, we shall not go through all the choices but only the ones involving the vital parts of the system:

- **FT2232:** The FT2232H is a USB 2.0 Hi-Speed (480Mb/s) to UART/FIFO IC. It may be used to convert one USB port to either 2 UARTs, 2 FIFOs, 2 MPSSE or a combination of these interfaces. We could have gone with just a microcontroller for the central block in the diagram in the Figure II.1. However, we were aiming at satisfying the duality of having a UART and a JTAG output/input.

After going through many IC's, we picked FTDI's chip designated FT2232 which happens to fill the criteria mentioned above. The Block Diagram in the Figure II.2 bellow shows in fact two separate channels respectively designated A and B each of them is configurable in a variety of industry standard serial or parallel interfaces. In addition, it comprises a USB Protocol Engine hence the Entire USB protocol is handled on the chip, so No USB specific firmware programming is required. [8]

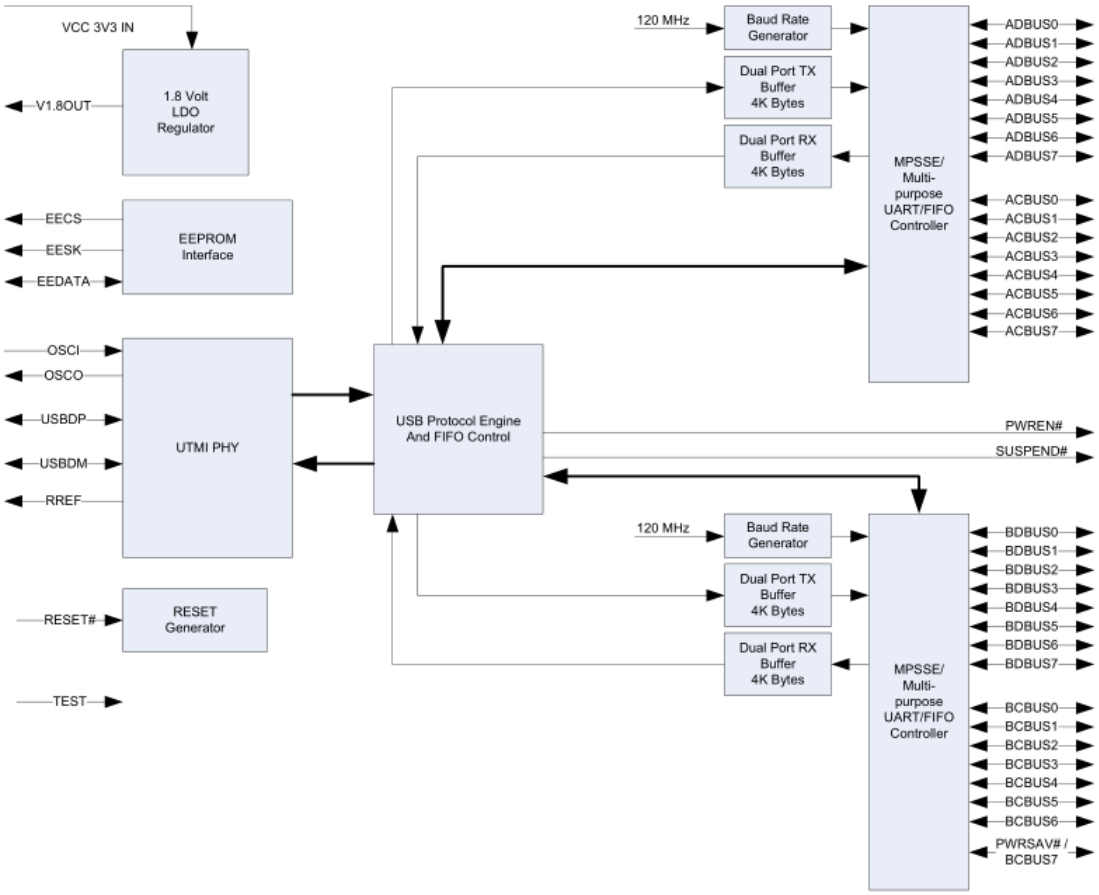


Figure II.2 – FT2232H Block Diagram[9]

- **EEPROM 93C46:** a type of non-volatile memory used in computers, integrated in microcontrollers and other electronic devices to store small amounts of data while gives you the ability to erase or reprogram it. The FTDI FT2232 chips are usually fitted with an EEPROM which is then used for configuring descriptors and operational parameters.

[8] Source: https://www.ftdichip.com/Products/ICs/FT2232H.htm/
[9] Source: https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232D.pdf/

When used without an external EEPROM interface, the FT2232H automatically defaults to a USB to dual asynchronous serial port device. Thus, adding an external EEPROM allows us to configure each of the chip's channels independently either as a serial UART (RS232 mode), parallel FIFO mode or fast serial. However, we can also exploit other aspects such as configuring the USB serial number, product description strings, this may also be used to program vendor specific USB ID. [10]

- **ESD Suppressors / TVS Diodes**: For ESD protection we went with a couple of ON Semiconductor's TVS diodes for every output / input signal instead of just using a single IC for maintenance purposes later on. As for the USB port we chose the STM's USBLC6-2SC6 ESD suppressor IC for the same task. [11]

- **USB Port:** an SMT USB Mini (instead of micro) with through hole leads from Molex to be less exposed to any radial movement related fracture and to be intact with the PCB later on to contribute even more to the robustness of the product as a whole.

1.3. Component Creation and Database

At the previous phase in the project's life cycle, the electronic system design was rather blocks containing the essential components for the well-functioning of the system. In order to move to the next phase in the design, we needed to provide an adequate model and footprint for each component and store them along with their parameters.

Thus, a components database is needed to arrange every component in its adequate category. The components database included different parameters such as library and footprint references, manufacturer, description and so on.

This database is then used to generate instances of the project components whenever needed. This way the project documents can be moved without any change in the schematics and without any loss.

[10] Source: https://www.microchip.com/wwwproducts/en/93C46C/

The Figure II.5 shows one of the components' both symbol and footprint, which will then be used to create both the circuit schematic and PCB.

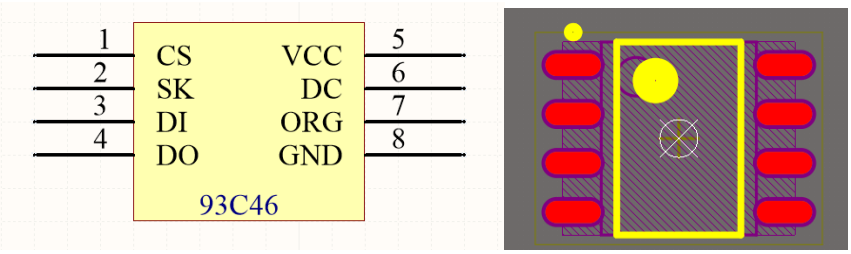


Figure II.3: From left to right, component's symbol and footprint

For the components' 3D bodies, there was no need for us to reinvent the wheel, so we either used Altium's 3D body generating tool or we acquired them from online sources[12] since it's not a critical part of the project. The Figure II.4 shows the 3D body of the EEPROM 93C46.

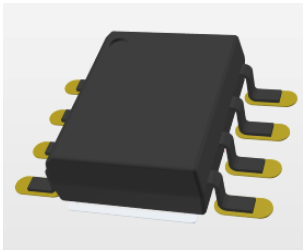


Figure II.4: EEPROM 93C46 3D body

1.4. Electronic circuit schematic

Building the schematic follows a specific modular approach consisting in building separate sheets and then connecting them using harnesses and ports. The schematic is then broken down into four separate sheets each of them serves a unique function.

- **USB and Power Connector circuit:** the USB connector and power supply are the least complicated circuit of the schematic. Yet it's a vital part of the design not to mention very practical also since it relies on USB power bus.

[11] Source: https://www.st.com/en/protection-devices/usblc6-2.html/

The Figure II.5 shows the chosen SMT USB port from Molex where both its data and power connections are protected by an ESD suppressor IC from STMicroelectronics to minimise if not eleminate the risk of damaging both the voltage regulator and the FTDI's IC for that matter.

The voltage regulator, fitted on both its ends with two identical 0.1 uF decoupling capacitors, delivers on its end a stable +3.3 volts to the rest of the circuit while taking as an input the USB +5 volts bus. Meanwhile, to notify the user that the device is properly powered we added a red LED, as a state indicator, with a protection resistor pulled down to the ground.

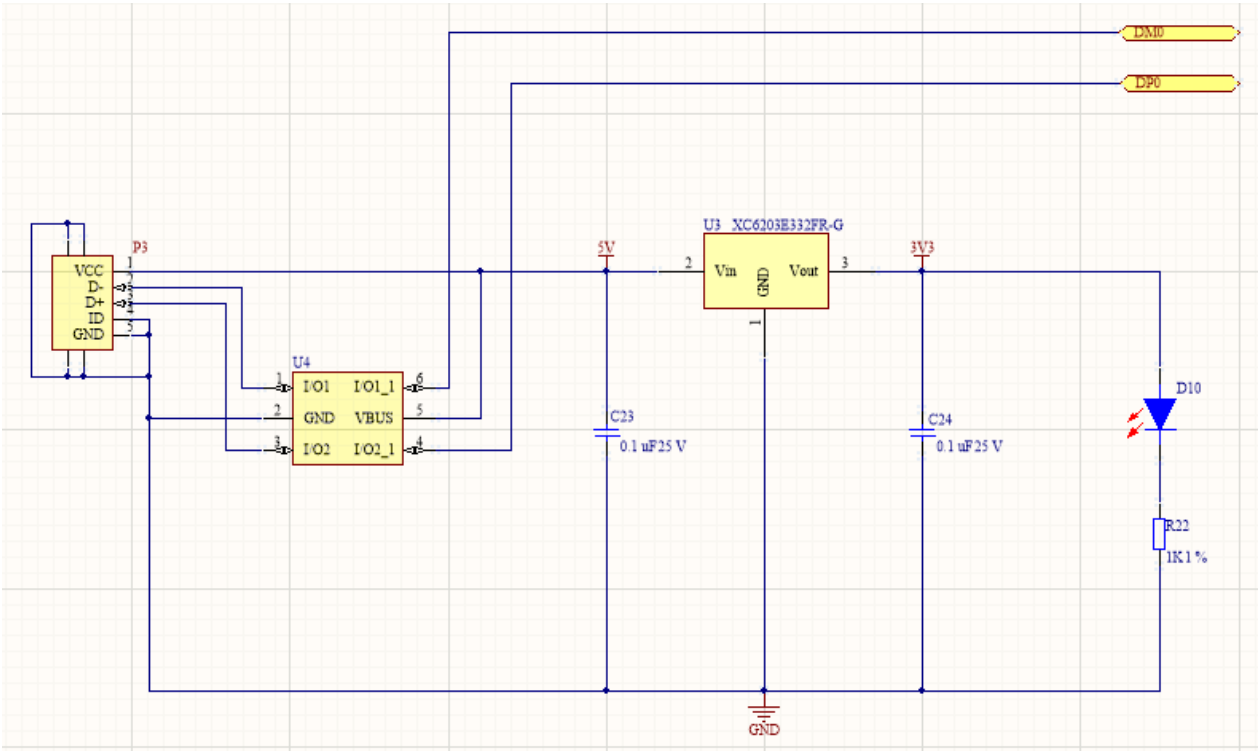


Figure II.5: USB and Power Connector circuit schematic

[12] Source: https://www.3dcontentcentral.com//

- **FT2232 USB to UART/JTAG Circuit:** as shown in the Figure II.6, the FT2232 placed at the center, makes up an important part in the schematic along with the EEPROM IC at the bottom left of the sheet equiped with three 10K pull-up resistors connecting the chip select (CS) , clock (CLK) and the data (DI/DO) pins to the +3.3 volts.

  It is possible to connect the Data In and Data Out pins. The data in and data out pins are connected together as instructed in the Microchip's 93C46 datasheet however with this configuration it's obvious that a bus conflict bewtween both pins could occur hence a resistor should be connected between DI and DO since both of them are connected to the same FT2232 pin on the other end.

  The number of bypass capacitors in this circuit is relatively noticeble for there are more power pins present in the FTDI's IC not to mention the sensitivity of the chip twowards tiny voltage drops caused by fluctuation in the delivered power .

  According to FTDI the recommended FT2232 external crystal oscillator configuration is a 12MHz ± 0.003% crystal along with two identical 27pF loading capacitors fitted between OSCI , OSCO and GND. The figure II.6 illustrates the different connections between the FT2232 IC and all other major components in the second sheet of the schematic.

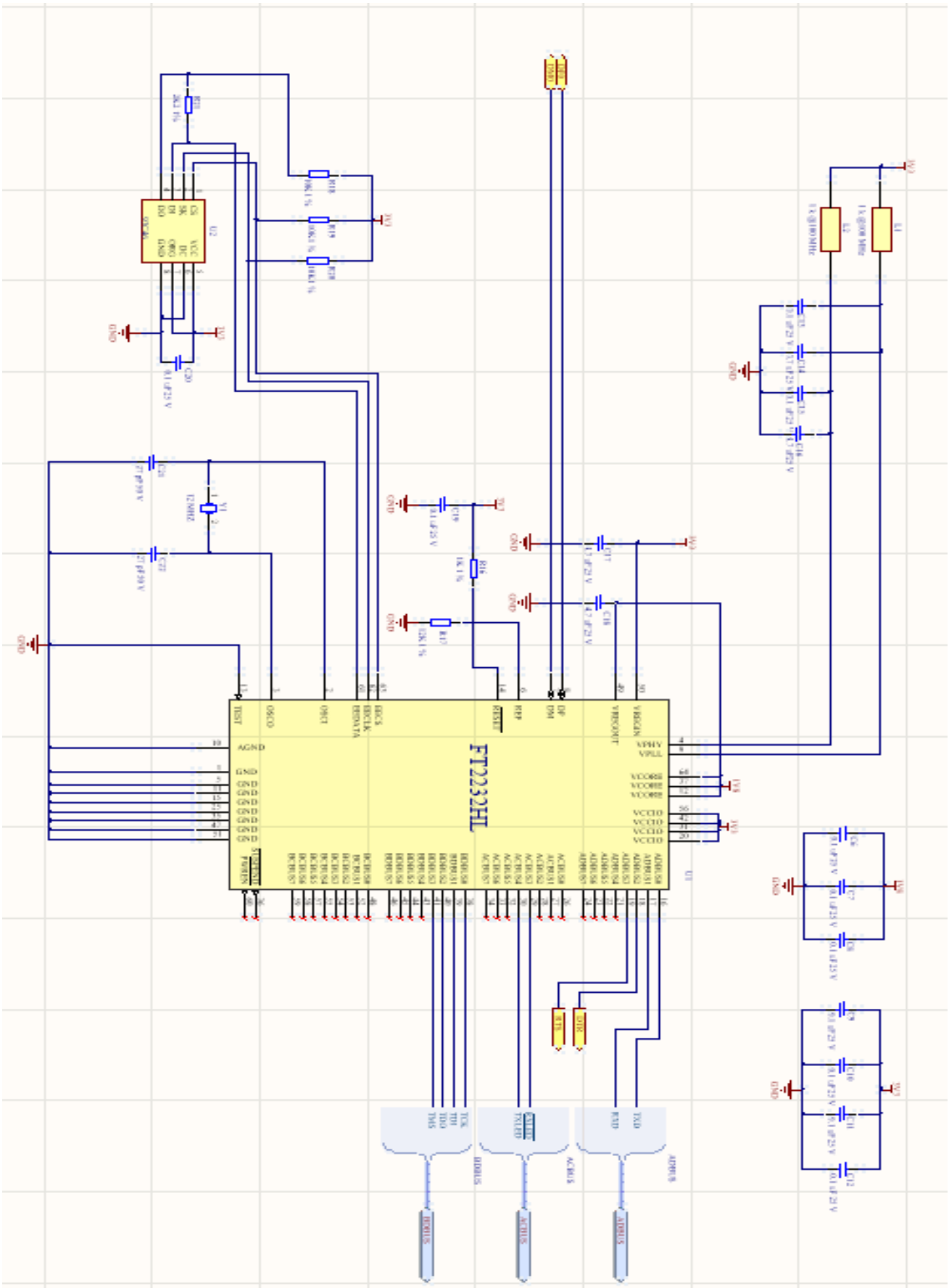
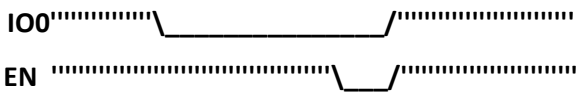


Figure II.6: FT2232 USB to UART/JTAG Circuit schematic

- **Auto program circuit:** RST and GPIO0 pins are used to reset and enter the bootload mode on target ESP device before programming starts. It's relatively hard to enter bootload mode manually. Press FLASH button and hold, press and release RESET button, release flash as shown below.



The ESP32 flashing mode which should be automatically entered using a USB serial interface such as the one we're working on. For this purpose we added a resistor / transistor bridge circuit from DTR and RTS to properly toggle the ESP32 EN and IO0 pins. In order to enter serial flash upload mode, a specific sequence must be executed . IO0 needs to be held low, EN is pulsed low, then IO0 is brought high. The ESP32 will enter the serial bootloader when GPIO0 is held low on reset.Otherwise it will run the program already in flash.

DTR	RTS	EN	IO0
1	1	1	1
0	0	1	1
1	0	0	1
0	1	1	0

Table II.1: logic table for mode selection

When GPIO0 pin is pulled low the ROM serial bootloader mode is activated whereas connecting it to VCC activates normal execution mode. If both DTR and RTS are pulled low, then neither GPIO0 nor EN are actively pulled down. Which means both DTR and RTS must be oppositely engaged in order for either output to be actively pulled down (one or the other, but not both).

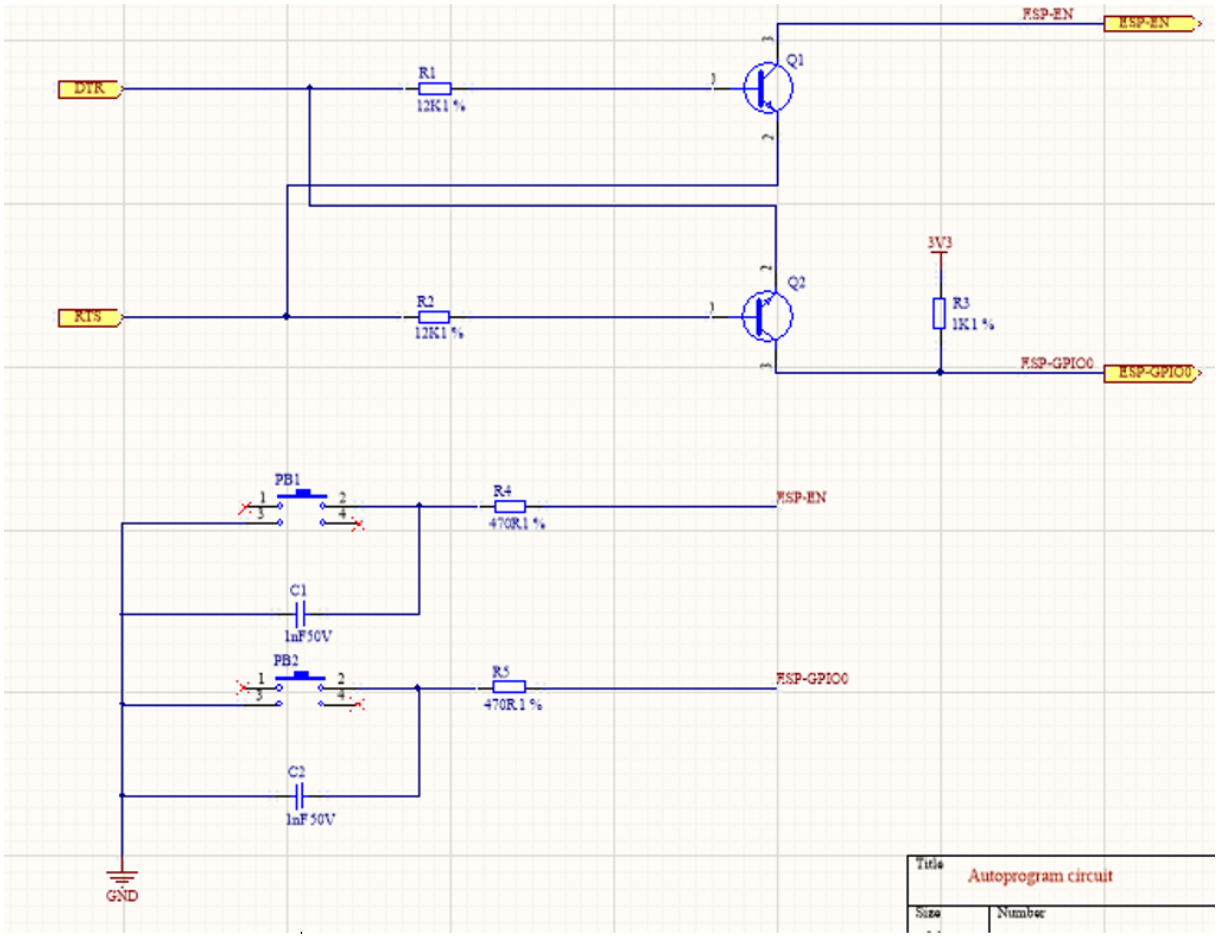


Figure II.7 Auto program circuit schematic

- **USB and JTAG Connectors circuit:** Figure II.8 illustrates the last part in the schematic is the two 2mm pitch male headers as the bidirectional data transfer ports fitted with a couple of ESD protection TVS diodes and an SMT bicolor LED module to indicate the receive and transfer operations (RX/TX).

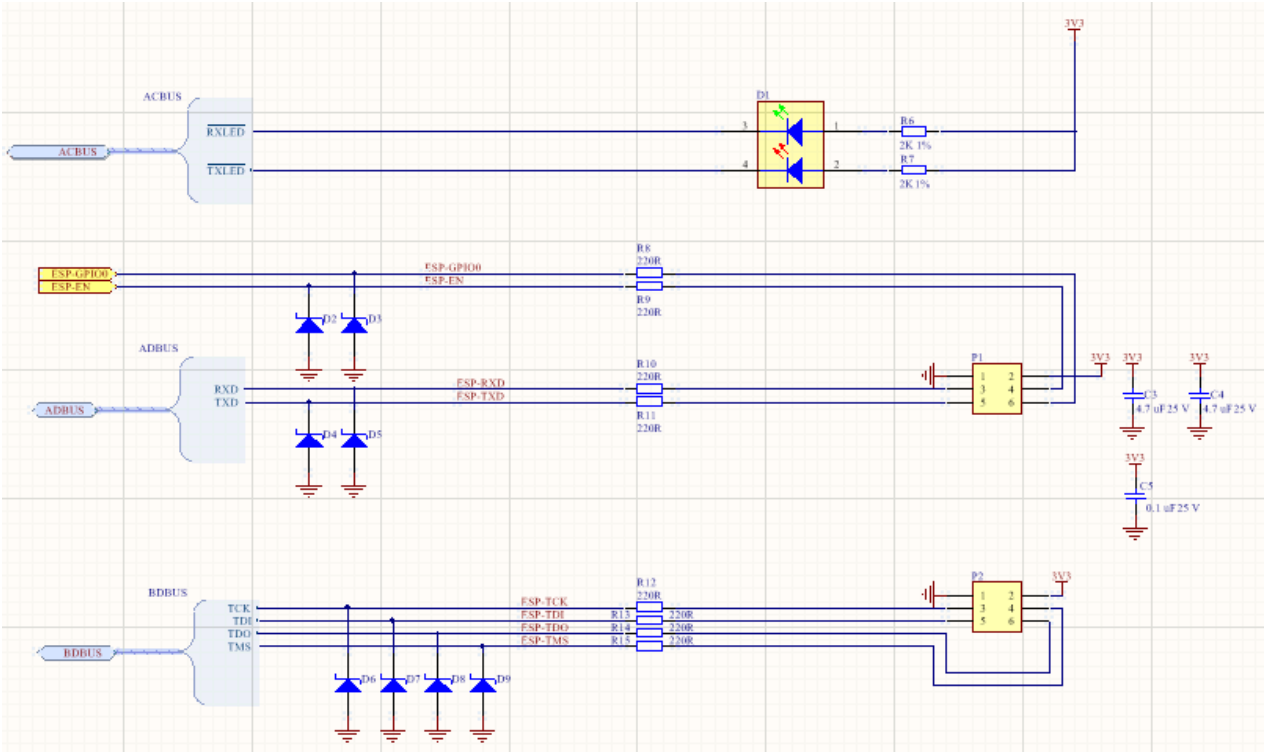


Figure II.8 Input/output Connectors Circuit schematic

1.4.1. PCB layout and routing

At a preliminary stage of the PCB design phase, the board layout and dimensions were carefully tailored accordingly to match and fit inside the enclosure which we'll cover in the next section. The corners, top and bottom parts, as shown in the Figure II.9, were shaped that specific way aiming to get the connectors in their respective places regarding the enclosure. The Figure II.9 also illustrates the different drills to be made in the PCB for the mounting holes and for placing the LED light pipes later on.

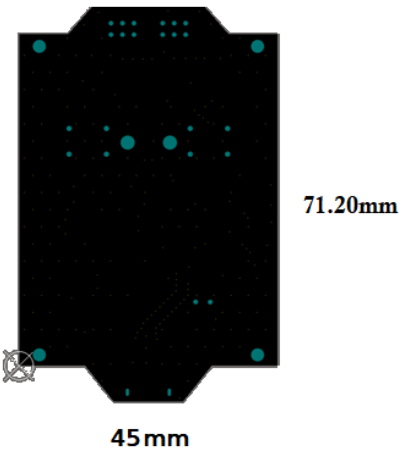


Figure II.9: PCB Layout and drills

Another important point that revealed necessary through the PCB design, is the components placement. During this stage of the PCB layout, the components should be moved into their functional blocks so that associated components are close to each other and the circuit can be routed easily later.

Therefore, we defined a placement methodology to work with. Like the schematic we placed the components as groups defined by their global function.

The FT2232 IC is placed relatively at the center of the board since it's the one with the most connections and signals to almost every component in the design and for being the heart of the system at hand.

The ESD suppressors are placed immediately after both connectors since they represent the higher risk of an electrostatic discharge. The Figure II.10 represents the final disposition for the component placement.

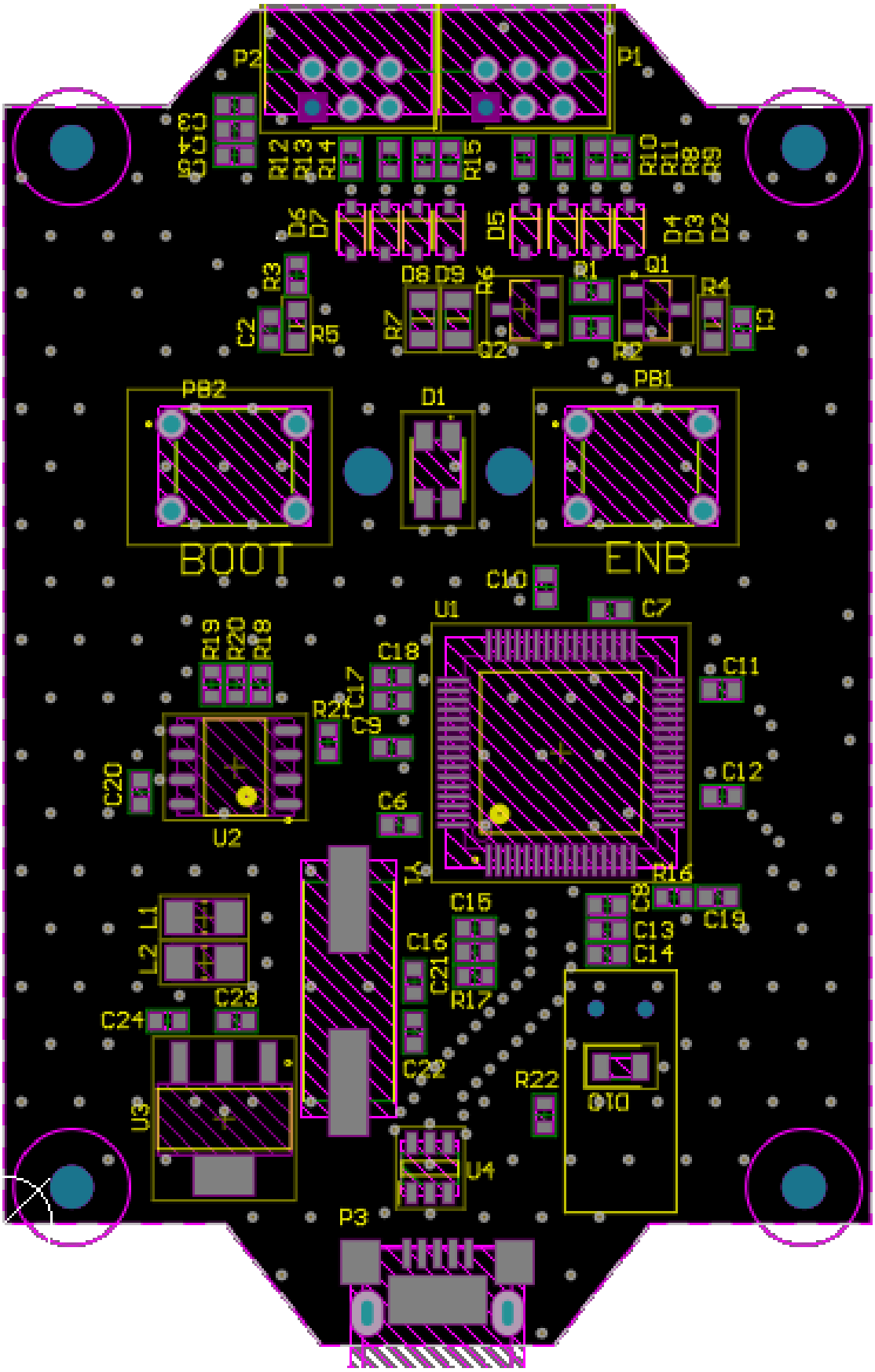


Figure II.10: Final Components placement

Most of the bypass capacitors were placed as to be the closest possible to their respective pins without interfering with any other component at the correspondent area. Especially the two capacitors fitted with the crystal oscillator in order to not add to any unwanted parasitic capacity to the circuit. As for the switches and the LEDs were conveniently placed close to the center of the board facing upward to provide the user with an easier experience. Last but not least, the disposition of the ports illustrates how intuitive and clear would be to use the product for it has a clear input and output disposed at each end of the board.

After we had gone with the disposition as shown in the Figure II.10, setting up the footprint library and then optimizing the placement of the components on the board and to properly connect the placed components while obeying all design rules we need specify the size and topology rules for power, differential pairs like USB high speed data pairs. Before we start laying down copper traces on the board, first we need to fixate all requirement on minimum trace width 0.3mm, spacing (clearance 0.2mm) and more importantly the number of layers (two layers in this case) because that’s what’s going to determine the price at the manufacturing stage.

Meanwhile, a closer look at the figure II.11 would reveal that we deliberately avoided using 90-degree trace angles and the reason why we did so is Because the outside corner of that 90-degree angle has the likelihood of being etched narrower than the standard trace width. Worst case scenario we could end up with some 90-degree traces that aren’t fully etched, resulting in short circuits. As a solution to this problem we used 45-degree angle traces which also produce an esthetically appealing PCB layout.

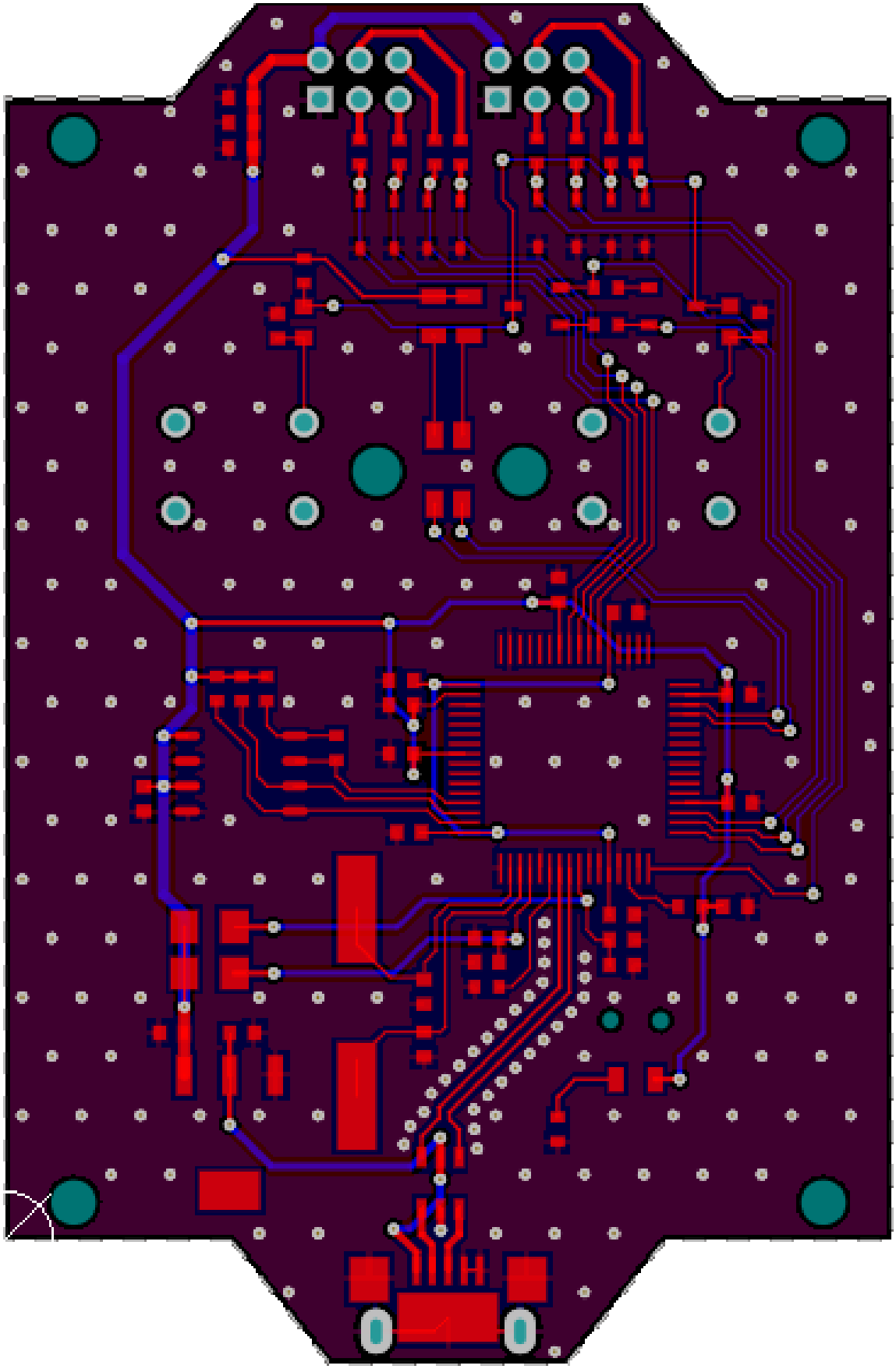


Figure II.11 Final routing for the PCB

Another important aspect we come across during this phase is having a common ground on our PCB, which is imperative as it gives all of the traces the same point of reference. Thus, different trace resistance and voltage drops can be somewhat frustrating and could lead to serious damages or malfunctioning. Therefore, we dedicated a perfectly continuous ground plane on both layers of the PCB.

For power traces, we increased the width of the traces to 0.7mm because power traces will have more current flowing through them and not making them wider will generate a lot of heat, which can end up burning wires and damaging the whole board.

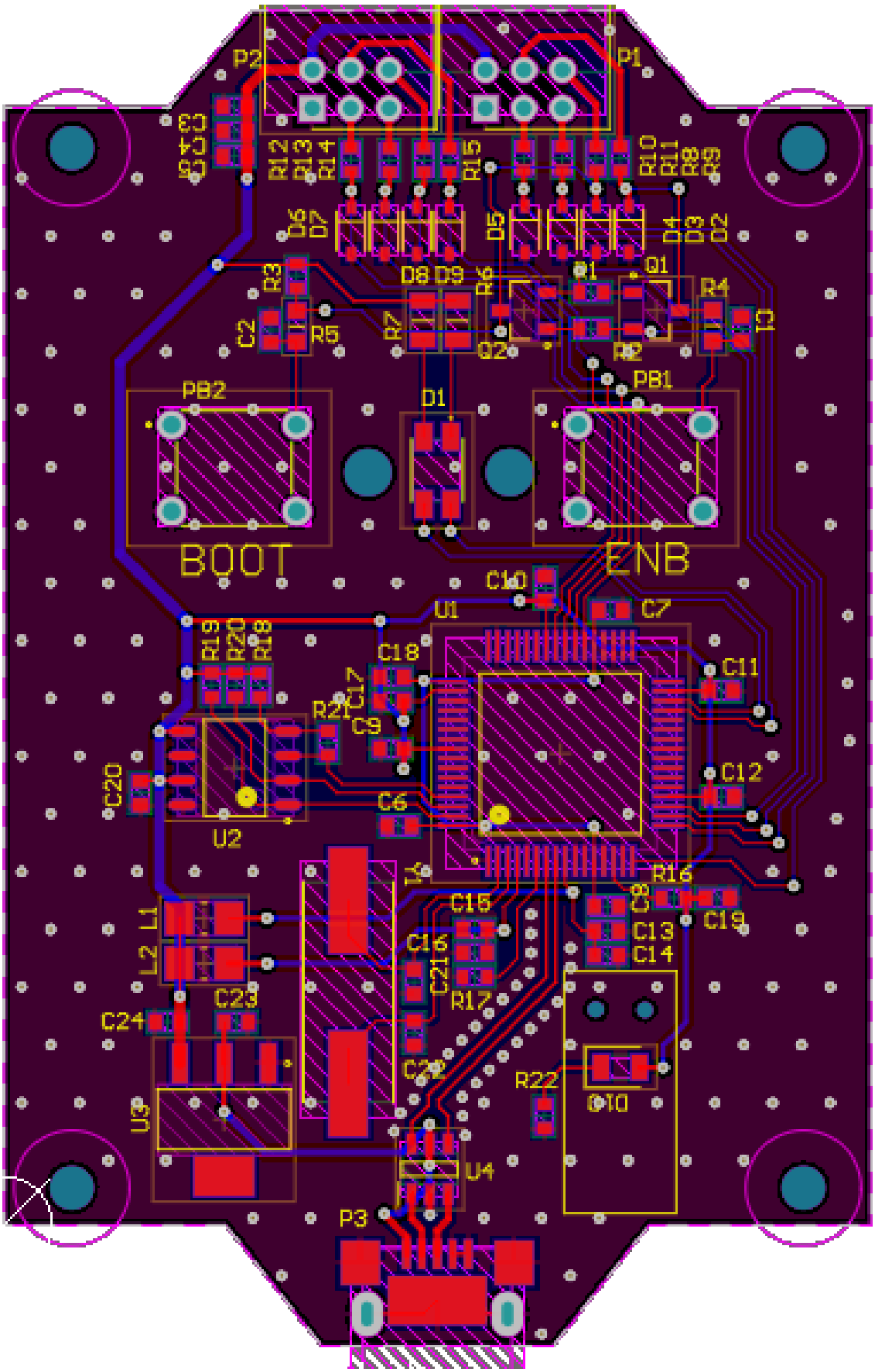


Figure II.12: Final Routed PCB

Last but not least, we used vias for providing connectivity between both layers but also to sink heat whenever present from under ICs such as transistors. One more trick that could make our design more reliable in the long run, is using vias as a Faraday cage. After finishing up with the whole design we managed to dispatch a considerable number of vias around the surface of the board and around the USB data wires to protect the whole board from EMI.

Figure II.13 provides a closer look at the 3D model of the finally ready PCB design.

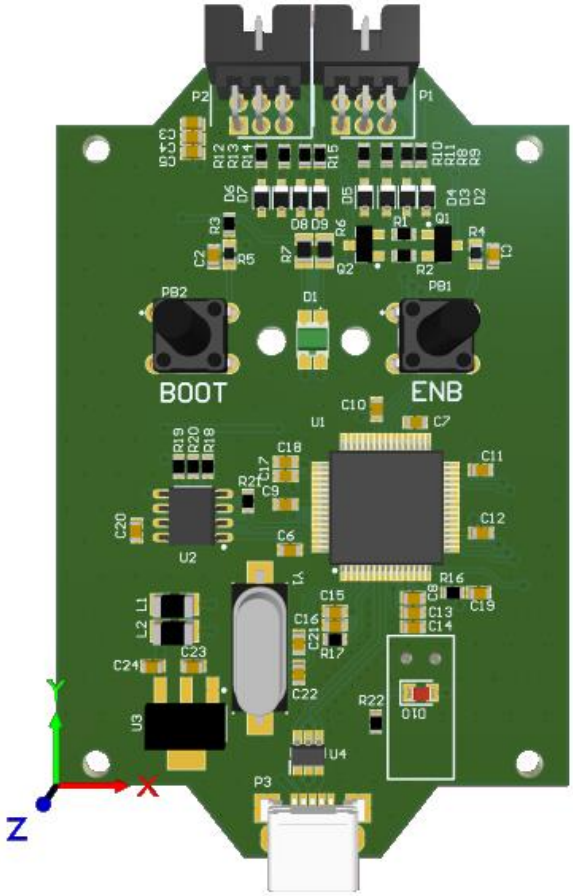


Figure II.14 final PCB 3D model

1.5. Product Enclosure 3D Design

At an early stage when working on the enclosure for the Board designed in the previous section, we started working on Autodesk Fusion 360 due to its simplicity and it could get the job done. However, later on we switched to the famous Solidworks because it is the most commonly known tool for 3D design and because of its vast community. Not to mention also the ability to realize the needed rendering for product visualization before diving into manufacturing without the need for any dedicated graphic design software.

Since we're not aiming for mass market production considering it's only a prototype. We selected a ready 3D model from OKW vast catalogue for electronic devices enclosures. Then we altered the model and customized it to meet our specific needs. OKW[13] is a German company operating since 1959. It is one of the world's leading manufacturers of enclosures and tuning knobs technology.

Figure II.15 shows the enclosure 3D final design before rendering it with the rendering tool made available by Solidworks.

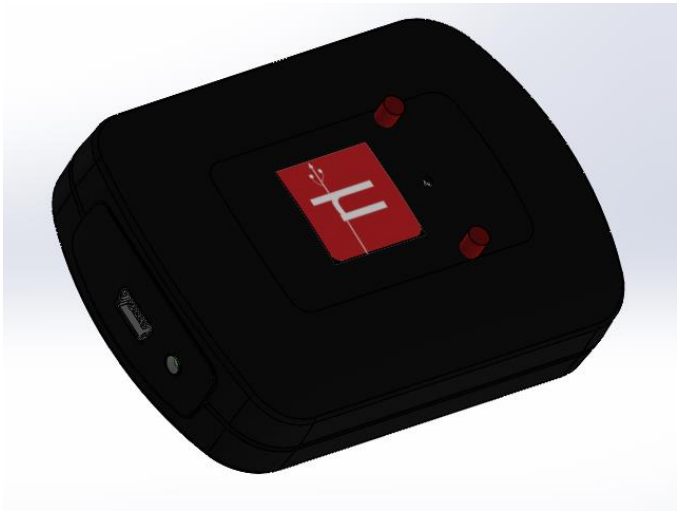


Figure II.15: 3D design for the enclosure before rendering

The modification made were a couple of drills for the tactile buttons' heads and both light pipes. In Addition, we modified both ends of the model to fit our connectors. The final result can be observed clearly in the Figure II.16. After conducting the rendering procedure.

Figure II.16 enclosure design after rendering

[13] Source: https://www.okw.com/en/

The figure II.17 shows the final product as a whole after exporting the 3D model of the PCB from Altium Designer and importing it in Solidworks to fit perfectly inside the enclosure.

Figure II.17 Final product design after adding PCB

Conclusion

During this second chapter, we covered a deeper description of the different building blocks of the hardware part of this project. We then specified most of the components we chose and moved to the realization phase where we talked about the schematic and PCB design. Last but not least, we stepped by the 3D design of the enclosure as the last part of this chapter.

Chapter III

Software solution development

Introduction

Finally, during the third and the last chapter of this dissertation report we'll go through the software development phase in which we'll discuss the programming language, APIs and the different tools employed in this process.

1. Programming Language and APIs

With development of embedded systems and IoT, multiple programming languages like C, C++ and Python can be used to pass the instructions to embedded system enabled electronic devices. For this part of the project, we chose Python as our programming language for various reasons among which its extensive support for open source libraries even for commercial purposes, first choice for embedded programming these days so it's a trend. Besides, it's well known for its fast speed on microcontrollers for embedded application. Also, we shall not pass without mentioning the easy readability and organization of the python code.

Since we are aiming to create a clear, modern and easy to use user interface for our program, we needed an API to help us do that. At first, we went through several options, like tkinter and wxpython, to get to the chosen one, PyQt, which is a Python binding of the famous cross-platform GUI toolkit Qt.

Figure III.1: from left to right Qt API and Python

The whole software was completely developed in PyCharm, an integrated development environment used in computer programming, specifically for the Python language, developed by JetBrains.

Last but not least, esptool, a Python-based, open source, platform independent, utility to communicate with the ROM bootloader in Espressif ESP8266 & ESP32 chips.

With the esptool library in hand we did not have to reinvent the wheel, and since this library is open source we integrated it into our software with some modification to suit the purpose of our dedicated program.

The software is divided into two major modes, production mode and advanced mode. The production mode is a simple user interface containing the minimum number of features allowing the operator to conduct the necessary flashing and or flash erasing procedures with minimum complexity and errors. The advanced mode on the other hand is more complex and sums up all the features and commands made possible with the esptool library but with an easier and more advanced user experience. In the next sections we shall provide a more detailed explanation of both modes.

Unlike the software introduced by Espressif Systems, Microtool, the software we developed as part of this project not only implements all-important features made possible with the integration of the esptool library but also focuses on the UI too making it more user friendly, modern and aiming for the most practical and favorable user experience.

2. Production Mode

The production mode as we mentioned in the last section was created in PyCharm IDE environment using Python as the programming language and PyQt library for the user interface.

First, we need to fetch all available serial ports on the computer and try to establish a serial connection with one of them. For this reason, we added a button designated Open COM, as illustrated in the figure III.2 below. To try to end the previously established serial connection we proceeded with the same idea but with an adjacent button designated Close COM.

In the left top corner of the figure III.2, we introduced few combo boxes. These are necessary to choose the com port, select the desired Baud Rate, the type of the chip and its flash size.

If we were to forget to select the flash size, it defaults to “detect” which will eventually detect the connected chip’s flash size and append it to its respective variable.

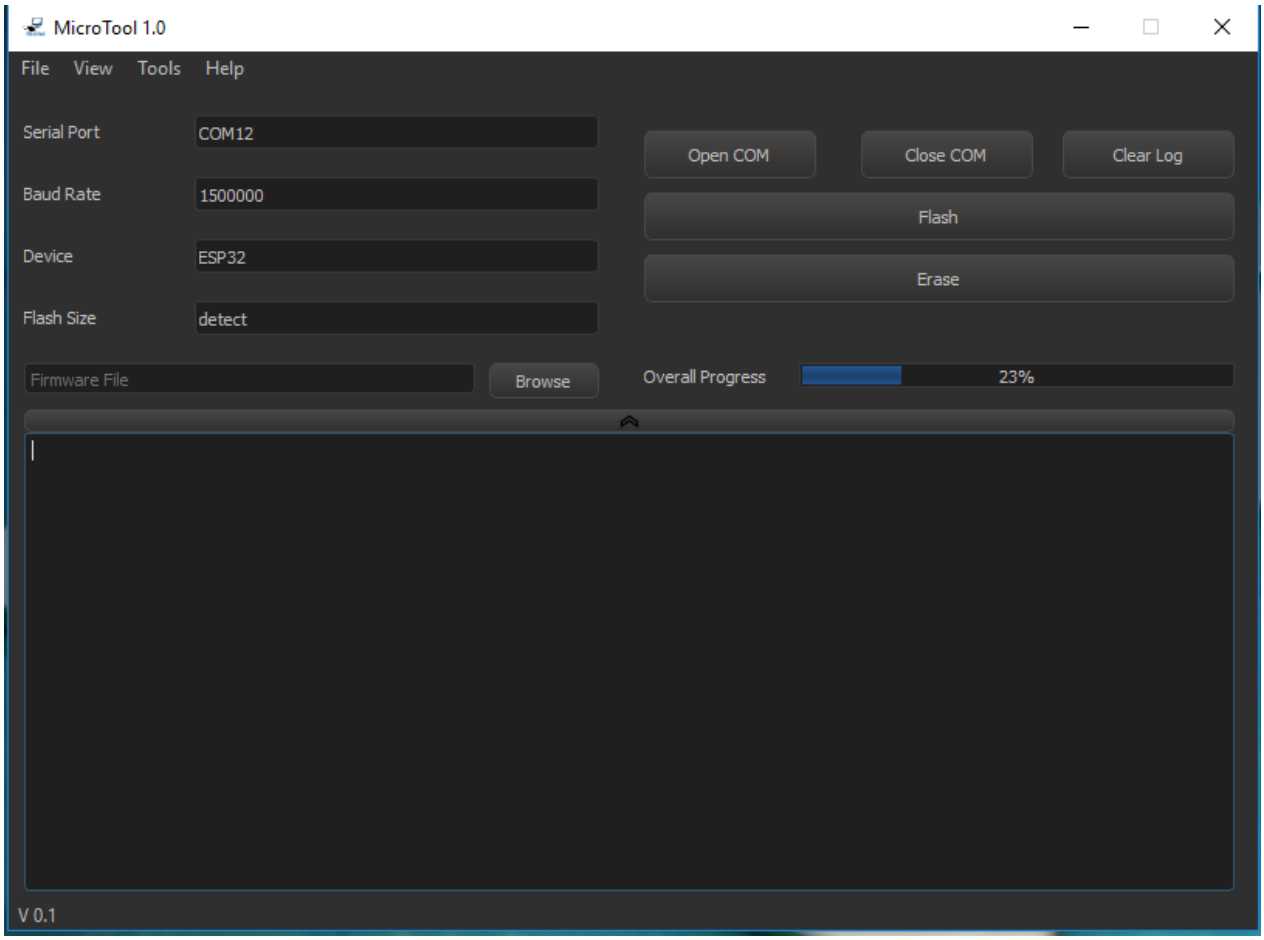


Figure III.2: Preview of the production mode UI

A line edit widget from Qt library were fitted with a button to browse and select the right firmware file or package to be flashed into the target chip. Meanwhile, two buttons were added to the interface to deal with the flashing and the erasing of the flash memory of the chip. To do so, we implemented the commands from esptool library into a function for each operation.

The piece of code in the Figure III.3 bellow illustrates the calls made of the commands into the respective flash function.

```
def flash(self):
    self.port = self.comboBoxCOMPort.currentText()
    if self.labelfirmware.text() == "":
        self.plainTextEditStatus.appendPlainText("Error: Firmware is missing.")
        return
    self.ser.close()
    if os.name == 'nt':
        if self.chip == 'ESP32':
            self.process.start(self.bundle_dir + '/esptool.exe --chip esp32 --port {0} --baud {1}\
                                --before default_reset --after hard_reset write_flash\
                                -z --flash_freq 80m --flash_mod dio --flash_size {2} \
                                0x10000 {3}'.format(self.port, self.baudRate, self.flasSize,
                                                    self.labelfirmware.text()))
        else:
            self.process.start(self.bundle_dir + '/esptool.exe --chip esp8266 --port {0} --baud {1}\
                                --before default_reset --after hard_reset write_flash --flash_size={2}\
                                 0 {3}'.format(self.port, self.baudRate, self.flasSize,
                                               self.labelfirmware.text()))
    else:
        if self.chip == 'ESP32':
            self.process.start('sudo python ' + self.bundle_dir + '/esptool.py --chip esp32 --port {0} --baud {1}\
                               --before default_reset --after hard_reset write_flash\
                               -z --flash_freq 80m --flash_mod dio --flash_size {2} \
                               0x10000 {3}'.format(self.port, self.baudRate, self.flasSize,
                                                   self.labelfirmware.text()))
        else:
```

Figure III.3: Flash firmware function

Finally, as the last part of the production mode interface we added a plaintext edit also from Qt widget library to show some log information to the user and a button with the name “Clear Log” to clear the widget whenever needed.

The menu bar at the top is not fully developed yet. It only consists of four submenus. The submenu named view contains the different style preferences for the GUI. Each style was created using QSS which is an implementation of usual CSS for Qt library widgets. To make it easier and comfortable for the user we decided to make both a light and dark theme for the GUI.

Under tools we created a submenu named “Advanced mode”, clicking it would allow us to switch to an even simpler window by the title “Log in “as shown in the figure III.4. The reason we included a log in is to limit the use of the advanced mode to an expert so that no ordinary operator would tamper with the firmware files or any kind of configuration made available by this software for that matter.

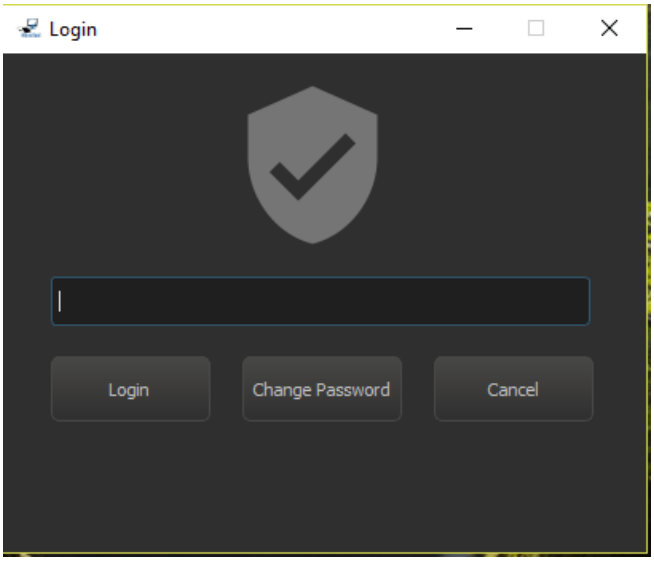


Figure III.4: Login window

By clicking the “Change Password “button we end up switching to another window as illustrated in figure III.5 bellow.

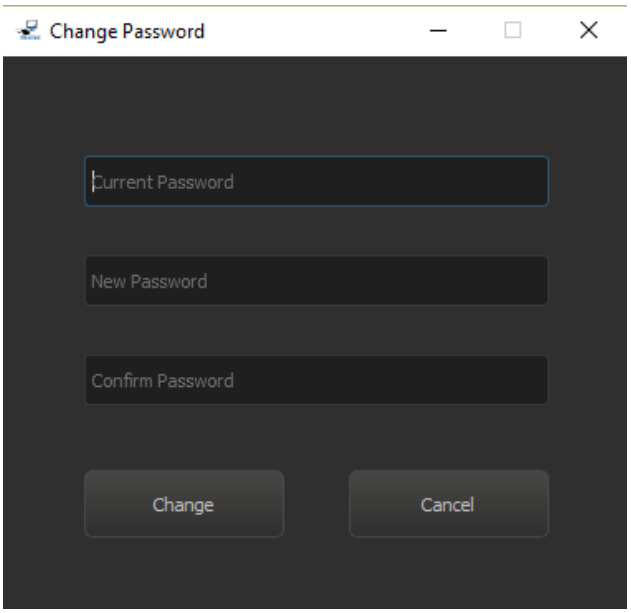


Figure III.5: Password change window

The window demonstrates a very typical password changing procedure where the inputs get loaded into an SQLite database to override the previous password. Canceling the process would bring us back to the login window.

3. Advanced Mode

The Advanced mode is divided into seven tabs each of them consists of multiple widgets serving a specific purpose yet part of a function. The first tab as illustrated in figure III.6 gives the user the ability to connect, disconnect from target device or even to log back out to the production mode. The default baud rate is 115200bps. However, different rates may be set using the combo box associated for the task. Having chosen the serial port and the baud rate the user can by then move to the next tab.

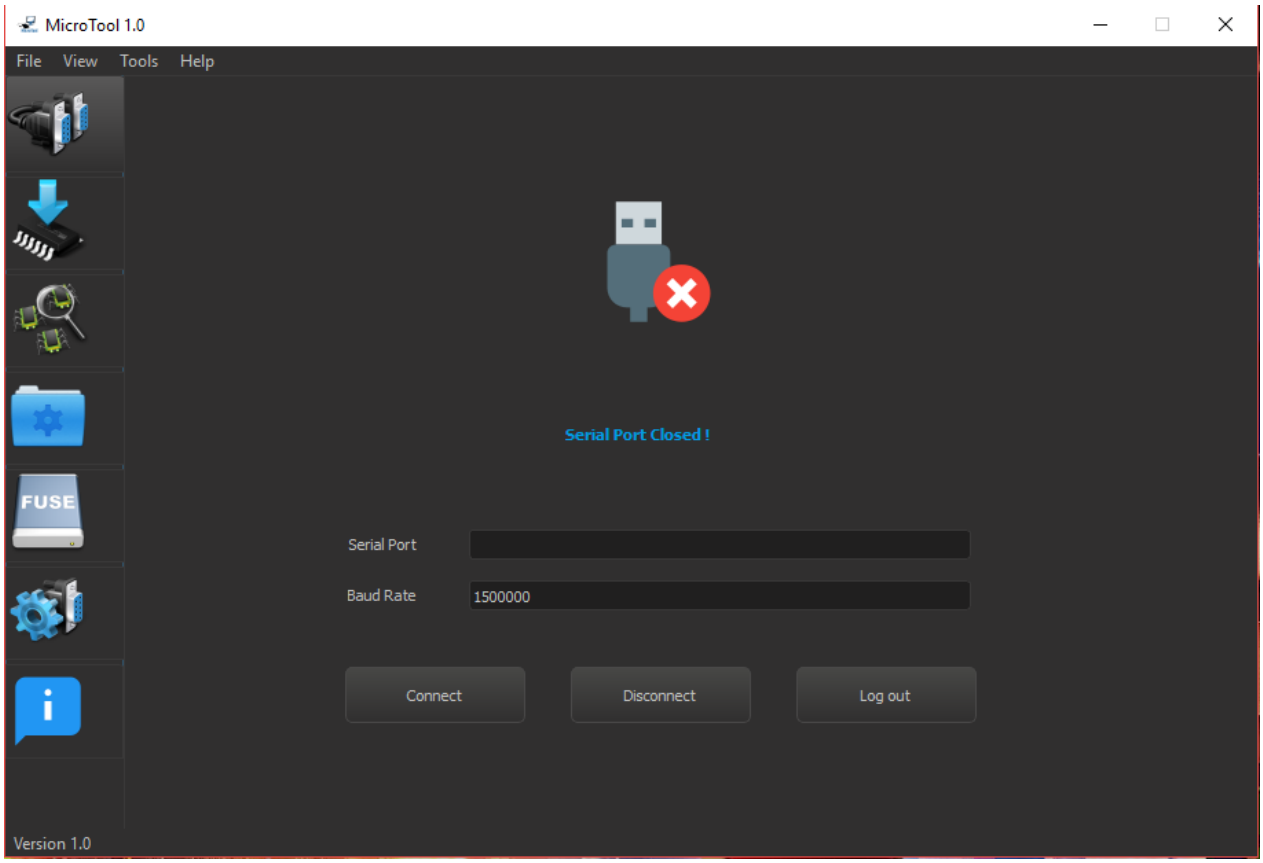


Figure III.6: first tab of the advanced mode

The second tab shown in figure III.7 holds all the actions needed to flash either firmware, bootloader and partition table or all at the same time. Once again, the functionalities mentioned here were nothing but mere commands that we managed to embed inside a process within our functions in order to call them whenever needed.

Using the flash set of buttons that we created in the second tab we can write binary data files to the ESP's flash chip via the command write_flash within our custom function.

The following line of code is an illustration of what we explained above:

```
self.process.start('python ' + self.bundle_dir + '/esptool.py --chip esp32 --port {0} --baud {1}\
--before default_reset --after hard_reset write_flash\ -z --flash_freq 80m --flash_mod dio --
flash_size{2}\{3}{4}'.format(self.port,self.baudRate,self.flasSize,offsetflash,self.lineEdit_pat
h1.text()))
```

In this line of code we simply call the command, write_flash, from the esptool library and using a simple string concatenation we associate the variables like serial port, baud rate, flash mode, flash size, offset and path to the firmware with their respective fields in the command.

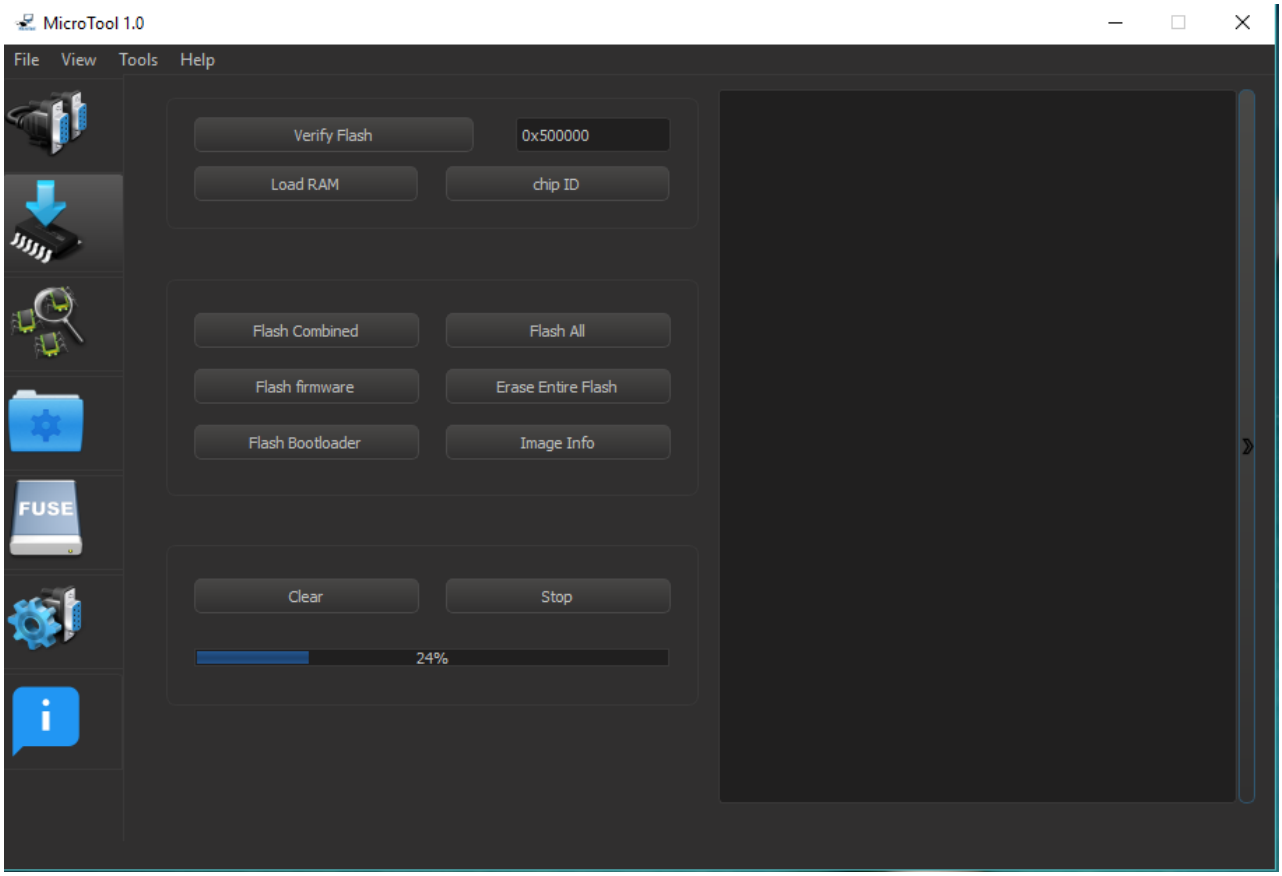


Figure III.7: second tab of the advanced mode

We then added three more buttons at the top allowing us to verify that data in flash memory matches a local file, load an executable binary image directly into RAM and then execute the program contained within it, display the version and Mac of the chip. All three functionalities were implemented the same way as the previous ones.

Third tab gathered all the functionalities related to memory management:

- **Read & Write Memory:** allows us to read and write single words (4 bytes) of RAM. this can be used for debugging as peek and poke at registers.
- **Erase Region**: to erase a region of the flash memory. takes as an argument the starting address and the length.
- **Dump Memory:** dumps a region from the chip's memory (ROM for example) to a file
- **Read status Register:** can be used for debugging hardware flash related problems. This can be used to check write protection status.
- **Write status Register:** same as read status register except it can be used to clear write protection bits.
- **Read Flash:** allows reading back the contents of the flash memory. It takes in and address, the size and a file to dump the output to.
- **Efuse Dump:** To display raw Efuse register values.
- **Encrypt Flash**: for encrypting the contents of the ESP32's attached SPI flash. When flash encryption is enabled, physical readout of the SPI flash is not sufficient to recover most flash contents.

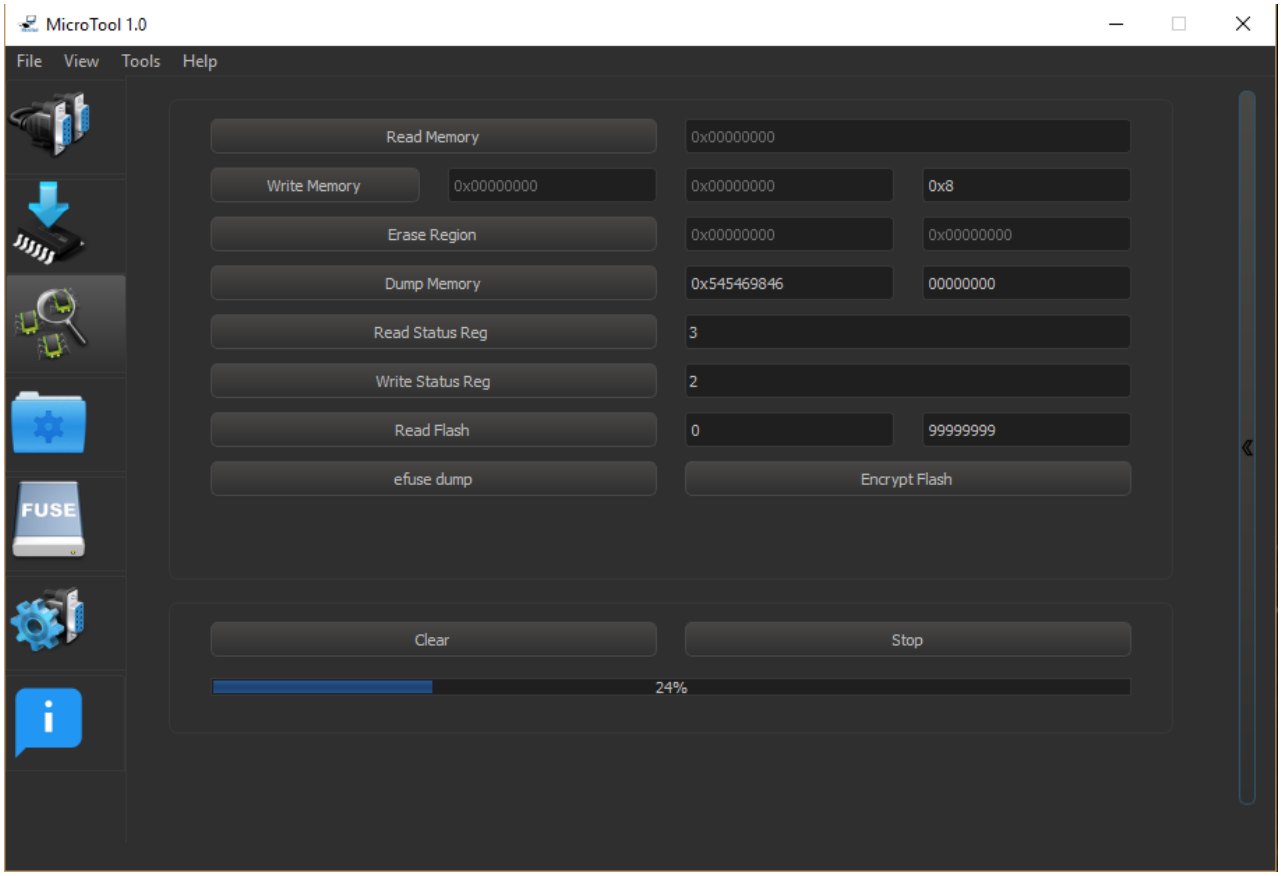


Figure III.8: third tab of the advanced mode

In the fourth tab of the advanced mode UI we introduced all the file management functionalities:

- Combine: combines all selected binary images to produce a single binary firmware image.
- Package: creates a new package with the extension ".mdt" holding the chosen binary files of the firmware and a configuration file (.ini) containing the offset addresses and other settings.
- Elf to Bin: converts an elf file to a bin image.
- Generate key: to generate the flash encryption key.

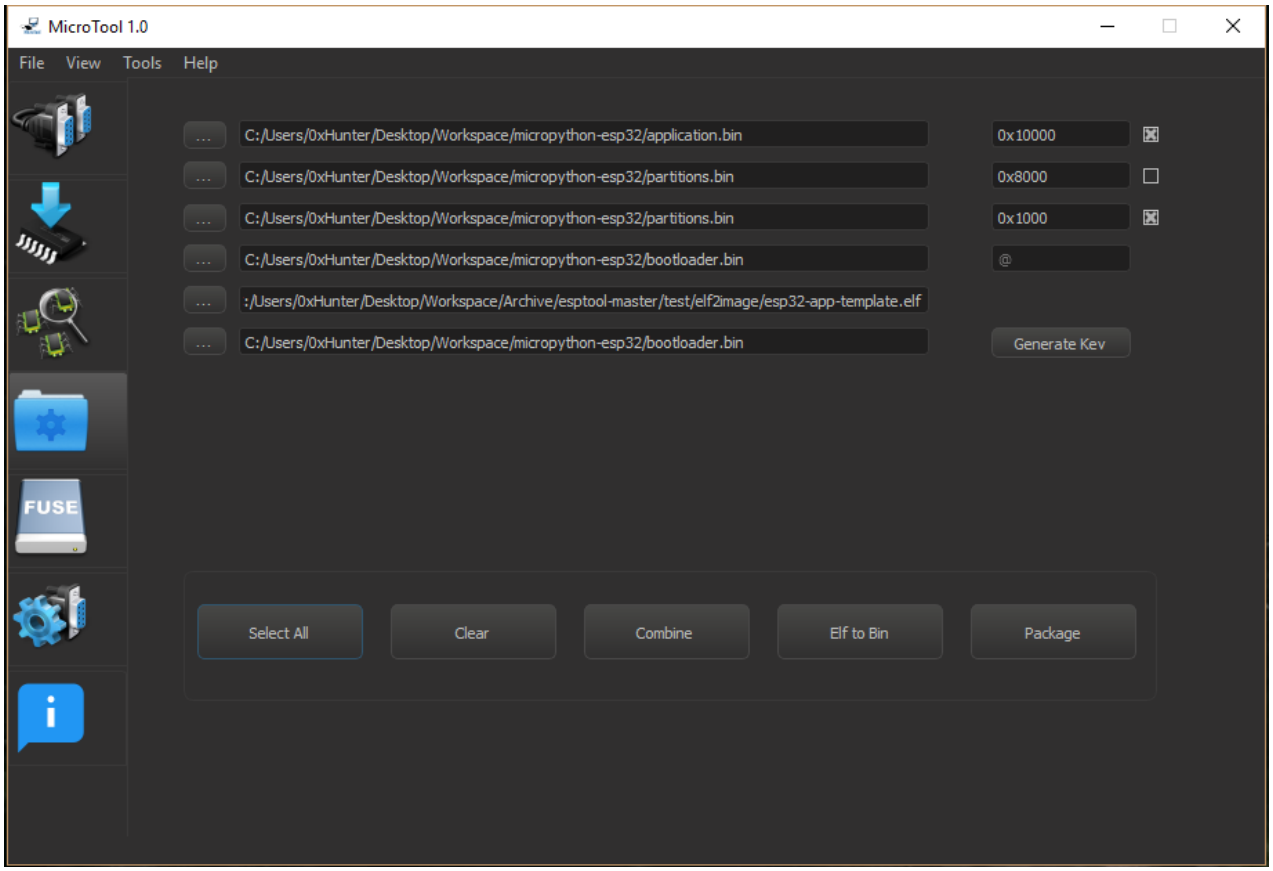


Figure III.9: fourth tab of the advanced mode

In the fifth tab the application loads the chip's Efuse registers in a clean and organized disposition inside a table widget along with an action column holding a set of buttons to allow the user to select and burn any Efuse of the loaded list.

All the buttons are protected with a warning message box to prevent any misfortune leading to a damaged ESP module. Since most of the Efuse bits are programmable only once (from zero to one).

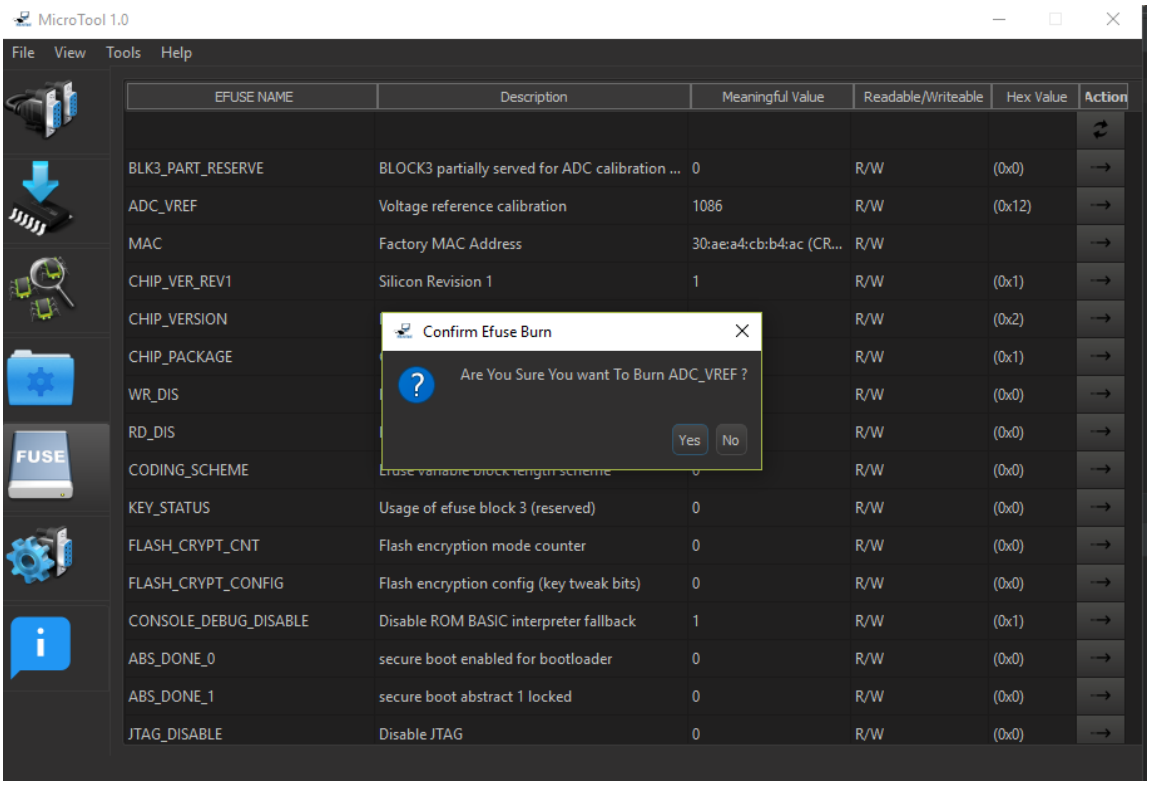


Figure III.10: fifth tab of the advanced mode

Finally, the last ready tab is the settings tab in which we are adding the necessary configurations before moving to the flashing phase.

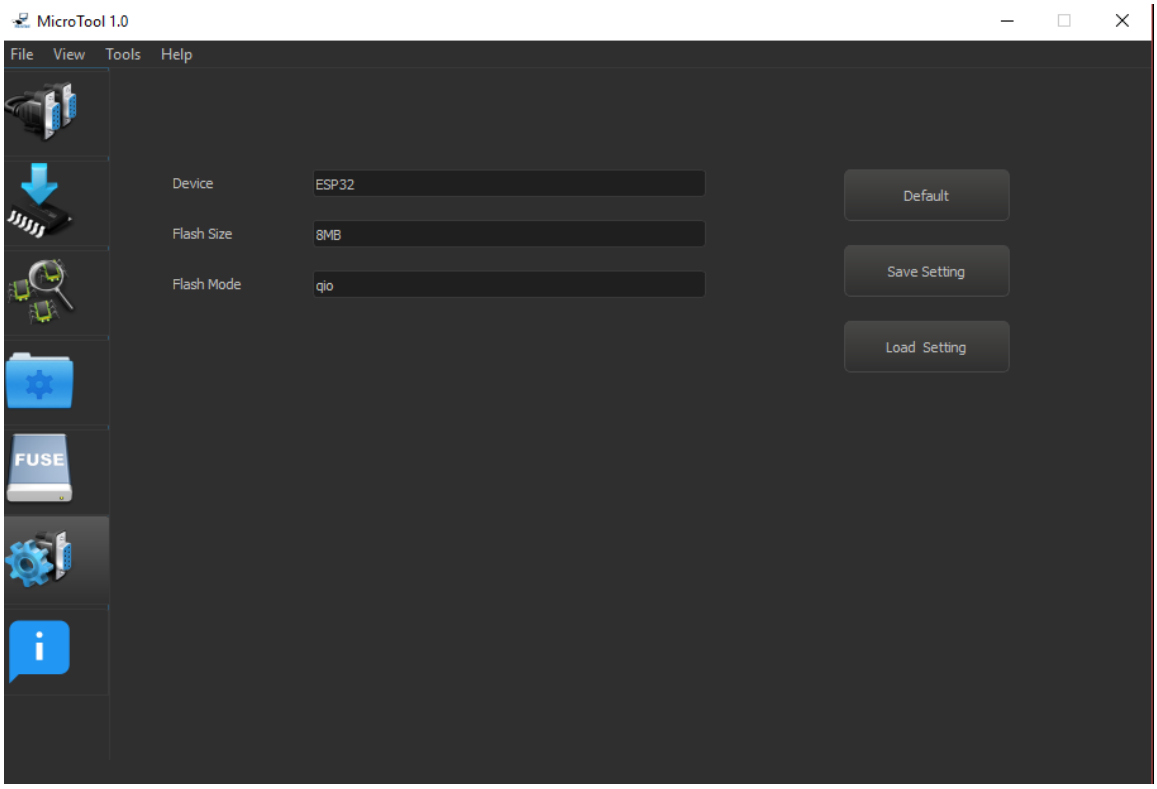


Figure III.11: sixth tab of the advanced mode

Conclusion

During this last chapter, we had gone through the description of our software solution and we covered a more in-depth description of the different tools and API's used in developing this software. The explanation was supported by several screenshots of the UI, Which made the concept easier to grasp.

Conclusion and Outlook

During the course of this project, we started with an actual need and we built our solution proceeding from that point. The work conducted in this project consisted in building a working prototype of a Programmer and debugger for the ESP32 modules.

Meanwhile, we succeeded in creating the electronic device to fill the need we just mentioned yet we also included a desktop application, equipped with the necessary libraries, fully capable of programming, erasing and debugging the memory of the ESP32 and ESP8266 modules. Introducing multiple functionalities that we studied and implemented within the software for the benefit of the user and for a better user experience.

This work will be continued to encompass major features such as strictly limiting the file type used by the software to only those generated by the packaging functionality. Besides, the exclusivity would go even further to customizing the software to only connect to our Hardware.

Finally, in the future we could consider adding a custom IDE integrated within our software and dedicated to programing ESP chips.

Bibliography

[1] https://www.espressif.com/en/products/hardware/esp-wroom-32/overview/ 14

[2] https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/bootloader.html/ 15

[3] https://www.espressif.com/en/products/hardware/esp-wroom-32/overview/ 15

[4] https://www.altium.com/altium-designer/ 16

[5] https://www.solidworks.com/ 16

[6] http://mdtunisie.com/ 17

[7] https://www.draw.io/ 17

[8] https://www.ftdichip.com/Products/ICs/FT2232H.htm/ 18

[9] https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232D.pdf/ 18

[10] https://www.microchip.com/wwwproducts/en/93C46C/ 23

[11] https://www.st.com/en/protection-devices/usblc6-2.html/ 24

[12] https://www.3dcontentcentral.com// 25

[13] https://www.okw.com/en/ 38