Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

forward together
sonke siya phambili
saam vorentoe

Test Station Upgrade for Radiation Tests on Microcontroller Boards

Richard Alexander Knipe
23540427

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at the University of Stellenbosch.

Supervisor: Dr A. Barnard

June 2024

Acknowledgements

I would like to thank the following persons for their help with my project:

- Dr Arno Barnard, for his willingness to supervise me even though he already had a hectic schedule, for being patient with all my questions, for all his helpful feedback, and for teaching me not only technical skills but also life skills such as prioritisation.
- Mrs Kannemeyer, for doing a flawless job of soldering the intricate components.
- My parents and brother, for encouraging and supporting me on a daily basis.
- All the other people who supported me in various ways, including my undeserved and selfless friends Franco and Herman. A special thanks to Franco for lending me his oscilloscope so that I could work from home.
- The Lord Jesus, for giving me the strength to face each day.

Stellenbosch UNIVERSITY IYUNIVESITHI UNIVERSITEIT

forward together
sonke siya phambili
saam vorentoe

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

23540427 Studentenommer / *Student number*	 Handtekening / *Signature*
R. A. Knipe Voorletters en van / *Initials and surname*	June 11, 2024 Datum / *Date*

Abstract

English

Radiation effects such as Single Event Effects (SEE) and Total Ionising Dose pose significant threats to electronics in space, specifically microcontrollers not designed for this environment, but which are employed in small satellites. Research is being conducted to better understand their behaviour when exposed to radiation. This is done by placing microcontroller boards in a vacuum chamber, where they are connected to a test station that monitors some of their parameters. To facilitate a smoother and more user-friendly testing experience, this project successfully addressed the shortcomings of a previous test station through the design and implementation of a new version that was more reliable and modular, and that offered more features. The second goal with the project was to design and develop a wired, long-distance communication scheme to control the test station from a remote computer by means of a Graphical User Interface (GUI), which was also done successfully.

Afrikaans

Radiasie-effekte soos enkelgebeurteniseffekte (SEE) en Totale Ioniserende Dosis (TID) is beduidende bedreigings vir ruimte-elektronika, spesifiek mikrobeheerders wat nie vir hierdie omgewing bedoel is nie, maar wat wel in klein satelliete gebruik word. Navorsing word onderneem om hul gedrag as gevolg van radiasie beter te verstaan. Dit word gedoen deur mikrobeheerderborde in 'n vakuumkamer te plaas, waar hulle aan 'n toetsstasie gekoppel word wat sommige van hul parameters monitor. Om 'n gladder en meer gebruikersvriendelike toetservaring te fasiliteer, het hierdie projek suksesvol die tekortkominge van 'n vorige toetsstasie aangespreek deur die ontwerp en implementering van 'n nuwe weergawe wat meer betroubaar en modulêr was, en wat meer funksies gebied het. Die tweede doel met die projek was om 'n bedraadde, langafstandskommunikasiestelsel te ontwerp en te ontwikkel om die toetsstasie vanaf 'n rekenaar oor 'n afstand en deur middel van 'n Grafiese Gebruikerskoppelvlak (GGK) te beheer, wat ook suksesvol gedoen is.

Contents

List of Figures

List of Tables

Nomenclature

Variables and functions

β	DC current gain of a BJT
θ_{JA}	Thermal resistance, junction to ambient
i_B	Emitter current of a BJT
i_C	Collector current of a BJT
P_D	Device power
T_A	Ambient temperature
T_J	Junction temperature
$R_{DS(on)}$	Drain-to-source on-resistance
V_{DS}	Drain-to-source voltage
V_{GS}	Gate-to-source voltage

Acronyms and abbreviations

BJT	Bipolar Junction Transistor
CSV	Comma-Separated Values
DMA	Direct Memory Access
ECAD	Electronic Computer-Aided Design
ESD	Electrostatic Discharge
GPIO	General-Purpose Input-Output
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
HSE	High-Speed External oscillator
HSI	High-Speed Internal oscillator
IC	Integrated Circuit
LED	Light-Emitting Diode
LSb	Least Significant bit
MCO	Microcontroller Clock Output
MCU	Microcontroller Unit
MOSFET	Metal-Oxide Semiconductor Field-Effect Transistor
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
SBC	Single-Board Computer
SPDT	Single Pole Double Throw
SPI	Serial Peripheral Interface
SWO	Serial Wire Output
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
UTP	Unshielded Twisted Pair
VNC	Virtual Network Computing

Chapter 1

Introduction

1.1 Background

Microcontrollers are cost-effective candidates for on-board computers in small spacecraft like CubeSats [4]. Among other electronic devices, they show unexpected behaviour when exposed to space radiation, which can lead to a range of adverse affects. Radiation effects can be categorised into cumulative or *long-term* effects and Single Event Effects (SEE) [5].

Cumulative effects happen over extended periods of time after repeated radiation exposure, usually causing irreversible damage to the exposed device. An example is Total Ionising Dose (TID), where a device is exposed up to a certain dosage level, which can be used to gauge the lifetime of the device to be placed in space [6]. An SEE is caused by a single, energetic particle. These effects are generally non-permanent and last for a short times. They occur more frequently with a higher dose rate, up to a saturation point [7, 8]. Types of SEEs include the Single Event Upset (SEU), a soft error that can cause a bit-flip in a register, leading to data corruption. The device normally recovers quickly without requiring a reset. There are also hard errors like the Single Event Latchup (SEL), which causes an abnormally high current flow due to factors like a parasitic BJTs induced by ion strikes. A power cycle can restore the device to its nominal state, provided that no permanent damage occurred [8].

Since radiation clearly poses a threat to electronics in space, particularly consumer microcontrollers not designed for this purpose, it is crucial to ensure that these devices are safe on earth before sending them on a mission. A test station is needed that can monitor microcontroller test subjects when they are exposed to space-like radiation. A first version of the test station was already developed, but because of the short time that was available for its development, it lacks in a few areas like modularity, robustness, neatness, and user-friendliness. Because tests are not performed often and the trips are expensive, a reliable and hassle-free support system is crucial. The original test station therefore required an upgrade that could address some (if not all) of its shortcomings in the form of modular PCB design to ease hardware replacement due to the harsh testing environment, support for more test subjects, and more parameters to be monitored. Radiation tests are performed in a room that contains specialised equipment such a vacuum chamber and/or cyclotron [1], while the test computer and operator are in another room [10]. Furthermore,

[1] A particle accelerator that can generate radiation beams such as proton-, neutron-, and alpha beams [9].

since radiation effects have to be detected quickly, readings have to be transmitted using a wired communication line.

1.2 Problem Statement

The project aimed to solve two main problems:

- To design and build a reliable, elegant, and modular test station to be placed in a vacuum chamber together with microcontroller boards to monitor and control them during radiation tests.
- To design and implement a user-friendly solution for monitoring and controlling this test station from a computer located a distance away.

1.3 Objectives

The project had three main objectives to achieve:

1. To design and build a test station base board consisting of minimal hardware that can be used together with multiple identical sets of test subject hardware to monitor and control the test subjects simultaneously, and to write software for the base board to accomplish this task.
2. To design and build multiple identical sets of complementary test subject hardware to be placed on the test station's base board, to design and build a user-friendly connection mechanism between each set of test subject hardware and each test subject, and to write software for the test subjects to allow the test station to monitor them.
3. To design and implement a long-distance wired communication scheme between a PC and the test station and/or subjects, and to design and implement software for the PC to monitor and control the test station and/or subjects in a user-friendly way.

1.3.1 Requirements

The project was originally defined as a support system that had to support microcontroller boards while doing radiation tests on them, and the requirements are explained in this section. The test station had to support at least four boards. Since they behave differently for different power supplies, they had to be powered by a configurable supply that had to be turned on and off remotely via UART, USB, or Ethernet, and the voltage and current of the supplies had to be monitored. The configurable power supply's 5V source had to

be switchable between a regulator and an external source outside the vacuum. Other parameters to be monitored included UART, SPI, and I2C verification, GPIO input- and output testing, and clock frequency measurement. Software was required to both gather and display this real-time data on a GUI running on a PC located 100m away. Live GUI graphs were required, and history functionality for these would be beneficial. Programming the test station and subjects over this distance is required, while debugging would be convenient. PCB design was required for reliability and easy replacement of subsystems, and had to be done in a modular way such that potentially broken sections could be swapped for operational equivalents. The system had to be robust and safe, and had to be immune to radiation-induced short-circuits caused by the boards at unpredictable times. It had to be vacuum-proof, and no sealed components or electrolytic capacitors were allowed. The test station had to be STM32-based. The test subjects had to report some of their readings directly through the PC and not through the test station, which would ease the communication bottleneck.

1.4 Project Scope

All designs including the PCB design had to be done from scratch, while optionally using the limited information given on the current test station as inspiration (Section 2.1.1). All software had to be written from scratch, and all hardware had to be chosen individually. Circuits had to be built and tested according to specifications formulated from the above requirements. The testing scope would include testing all designed hardware and software, but it would be limited to outside the vacuum chamber, and no radiation exposure would be done. Selection of low outgassing materials were not necessary. Total component cost did not have a hard upper limit, although it had to be minimised where possible. Other factors outside the project scope included automatic detection of SEEs, forward-error correction such as parity checks, run-time testing, over-voltage and over-current protection, and finally short-circuit detection and the automatic removal of power in such a scenario.

1.5 Report Layout

Throughout the report there are references to jump to different sections and appendices, and to return to the place where that section or appendix was referred to. The report comprises six chapters, with Chapter 2 being a literature review that investigates similar solutions and explores foreign topics, Chapter 3 explaining the high-level system design and formulated project specifications, Chapter 4 giving the detailed design of the previous chapter, Chapter 5 investigating the results of the designed system, and Chapter 6 concluding the report with a summary and future recommendations.

Chapter 2

Literature Review

To solve the problem at hand, it was first necessary to investigate similar solutions. This section starts with an examination of the first test station, which is followed by possible solutions to relevant sub-problems of the project, such as remotely programming a microcontroller. Finally, considerations for the vacuum environment are discussed.

2.1 Related Work

2.1.1 First Version of the Test Station

The existing test station could monitor three microcontroller development boards simultaneously, and consisted of an STM32F413 on a NUCLEO-144 development board, a relay module for power control, voltage regulators for power supplies, and a three-channel current- and voltage sensing module - the INA3221. A Raspberry Pi was used to exchange data between a PC and the test station and subjects over a 100m Ethernet cable by means of its four USB ports and Ethernet port. The free software called VirtualHere was used to make the USB ports of the Pi available over the Ethernet cable. This way, the test subjects could be programmed as if they were plugged directly into the PC. A similar approach showed that VirtualHere could be used together with a Raspberry Pi and a Wi-Fi connection shared between the PC and the Pi to stream data from a web cam to another room [11]. Not much is known about the GUI except that it was capable of displaying readings and live graphs, and that it could store the readings in a Comma Separated Values (CSV) file. To measure clock frequency, it used a timer with an external clock source, and another timer to reset the former and to read its counter value such that a frequency could be determined, but no further information on this is known.

2.1.1.1 Remaining challenges

Although the first test station solved many problems in clever and practical ways, there were still some features that needed attention. Cable management was done with a wire harness, which was a nuisance because the vacuum-proof low outgassing cables that were used were only available in one colour. There was no quick way of connecting the test subjects to the test station, and each wire had to be connected individually. The test station's MCU was also overkill, since many pins were not being used. Using only one

sensor module and having all the hardware on one board was risky, because if the one sensor failed due to the radiation or for some other reason, the entire test station would be rendered useless. Furthermore, the test station could only support up to three subjects. Since it was not built on a PCB, it was also not very reliable. Finally, it had a GUI and graphs, but these had no history function.

2.1.2 Options for Programming an STM32

There are various ways to upload firmware to an STM32 microcontroller, such as with an ST-LINK debugger/programmer, through the USB peripheral, or even over Ethernet.

2.1.2.1 ST-LINK

The easiest way to program an STM32 MCU is to use an ST-LINK, the official debugger from ST Microelectronics that can be categorised into the ST-LINK V2 and the newer ST-LINK V3, and there are many versions under both categories. These devices support both debugging and the flashing of firmware directly from the STM32CubeIDE, which makes them convenient [12], [13]. Another benefit of these devices is that they can translate the USB signal from the PC into a UART signal, which allows the MCU to use the straight-forward UART peripheral to communicate with the PC for reasons other than programming [14]. A terminal program can then be used to send data to and receive data from the STM32.

2.1.2.2 USB DFU

It is also possible to program an STM32 MCU through its USB peripheral by using Direct Firmware Upgrade (DFU) [15], [16]. This approach eliminates the need for an external ST-LINK, but it sacrifices the convenience of debugging and requires one to upload a generated file from the IDE to the STM32CubeProgrammer software, pulling the BOOT0 pin high, and doing a power cycle every time new firmware needs to be uploaded. If this needs to be done frequently, this approach would be cumbersome. It also does not address the issue of programming over a long distance, and simply reduces the amount of hardware on the MCU's PCB. Although the method does not utilise a convenient USB-to-UART converter like that of an ST-LINK, the USB peripheral of the STM32 can still be used to communicate with the PC. A terminal program can then be used in the exact same way as with the ST-LINK.

2.1.2.3 IAP over Ethernet

It has been shown that In-Application Programming (IAP) can be used to upload firmware over Ethernet to an STM32 MCU by using Trivial File Transfer Protocol (TFTP) and the

STM32CubeProgrammer software [17]. Furthermore, an application note gives instructions for IAP over Ethernet using either TFTP or Hypertext Transfer Protocol (HTTP) [18]. However, there are not many examples of projects using this approach. Moreover, only some of the larger NUCLEO-144 development boards embed an Ethernet port, such as the NUCLEO-F429ZI [19]. This means that one will not be able to test the programming functionality using an ordinary NUCLEO-64 before obtaining the PCB with the custom Ethernet hardware. Devices exist that can give Ethernet capability to an ordinary STM32, like the popular ENC28J60, which comes in a breakout board version (Figure 2.1) or as a standalone Integrated Circuit (IC). However, it is not certain whether a firmware upgrade will be possible using this approach. The Ethernet peripheral of the STM32 (or an external Ethernet module) can be used to communicate with the PC for other reasons than programming, but the convenience of communicating with the PC through a terminal program is absent with this approach.

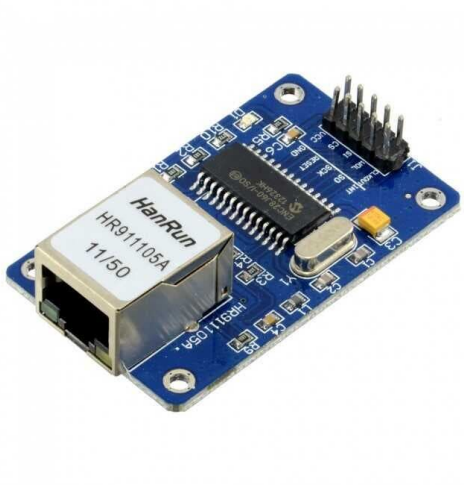


Figure 2.1: ENC28J60 network module [1].

2.1.3 Extending USB devices over long distances

It can be desirable to extend multiple USB devices over a long distance, such as when wanting to exchange data over a serial port or remotely program via an ST-LINK. It has been shown that an Ethernet cable can extend a USB port over 10 metres simply by cutting the USB cable, putting two male Ethernet connectors at each cut end, and connecting the ends together using an Ethernet cable and two female-to-female adapters [20]. This works because the twisted pair of the Ethernet cable blocks external noise. However, to reach distances such as 100 metres, the only option is to convert the USB protocol to an Ethernet protocol using a physical layer chip.

2.1.3.1 Ethernet cable categories

Figure 2.2 shows a comparison of different Ethernet cable categories that will be referred to throughout the report.

	Length	10Mb/s	100Mb/s	1GbE	10GbE	PoE	Mhz
CAT5	100	✔	✔			✔	100
CAT5e	100	✔	✔	✔		✔	100
CAT6	100 (55 for 10GbE)	✔	✔	✔	✔	✔	250
CAT6a	100	✔	✔	✔	✔	✔	500

Figure 2.2: Comparison of Ethernet cable categories [2].

2.1.3.2 Commercial solutions

There are reputable commercial solutions which can extend multiple USB ports over 100m. An example is the 4KEX100-KVM, which can among other things extend four USB 2.0 portd over a distance of up to 100m and at a speed of up to 480 Mbps using a Cat6 or above cable [21], but it costs over R4000 ($240). Another example is the JTECH-BE100, which can extend two USB 2.0 ports at the same specifications as the 4KEX100-KVM, and which costs just over R2000 ($120) [22]. There are also options under R1000, but no reputable product claiming 100m could be found. An example is the Cable Matters 202159 for just over R900 ($45), which can extend two USB 2.0 ports over a distance of 45m using a Cat5e/6 cable [23].

2.1.4 Python GUI for Live Serial Readings

It has been demonstrated how Python could be used to read data from USB ports, how it can then be displayed in the form of graphs on a GUI, and how it can be saved in a CSV file. The tutorial used the popular TKinter library for the GUI and although it looked very basic, the software was simple to understand [24]. Other candidates for writing a GUI with Python include the PyQt library, which has much more widgets, style sheets, and even a QT Designer feature, where widgets can be dragged and dropped. It is also more object-oriented than TKinter, but takes longer to learn. Kivy is another alternative which, similar to PyQt, has a separate file for styling with a .kv extension. Kivy is a popular choice for cross-platform applications, supporting Windows, macOS, Linux, Android, and iOS. PysimpleGui, built on top of TKinter, simplifies TKinter's boilerplate code but is not suitable for larger projects [25].

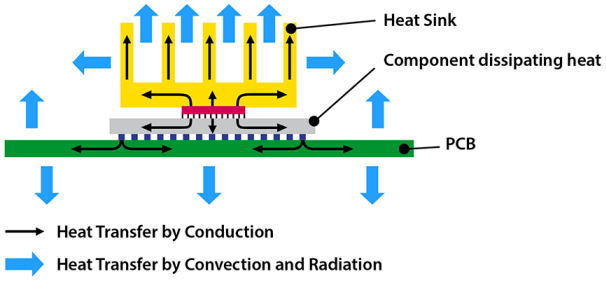
2.2 Considerations for Electronics in a Vacuum

Besides the electrolytic capacitors and other sealed components that cannot be used in a vacuum (as Section 1.3.1 stated), there are other factors to consider.

Figure 2.3a shows the three ways heat are transferred from an electronic component mounted to a PCB and a heat sink. Radiation (when heat is transferred from a warmer to a colder body through electromagnetic waves) transfers much less heat than convection and conduction in the context of electronics [26]. Heat transfer by convection, which happens through the movement of fluids, is almost negligible in a vacuum [27]. This leaves conduction as the only effective way of transferring heat, which is through direct contact between materials. Heat sinks are therefore much less effective in a vacuum, and material conductivity and size are of greater concern than factors like shape and colour.

Outgassing is when non-metals like polymers, adhesives, rubber, or epoxy release a gas when exposed to heat and/or a vacuum [28]. In the context of electronics, this includes thermal paste, plastics, glues, and PCB materials [29]. The released gas can deposit on components, which can change their behaviour or even cause short-circuits between closely spaced conductors [29]. To prevent this, components such as low outgassing wires can be used, which are made from materials that meet standards like a certain Total Mass Index (TML) and Collected Volatile Condensable Material (CVCM). For example, NASA's low outgassing test involves placing a material in a vacuum chamber heated to 125 °C for 24 hours, and the material's TML and CVCM must be less than 1% and 0.1%, respectively, for the material to be suitable for outer space [30].

To get wires such as data or power cables into a vacuum chamber, a wire feedthrough with a limited number of pins is used (Figure 2.3b).



(a) Heat flow of an electronic component [26].

(b) Vacuum wire feedthrough [31].

Figure 2.3: Factors to consider for electronics in a vacuum.

Chapter 3

System Design

3.1 System Overview

A diagram showing the different system components can be seen in Figure 3.1.

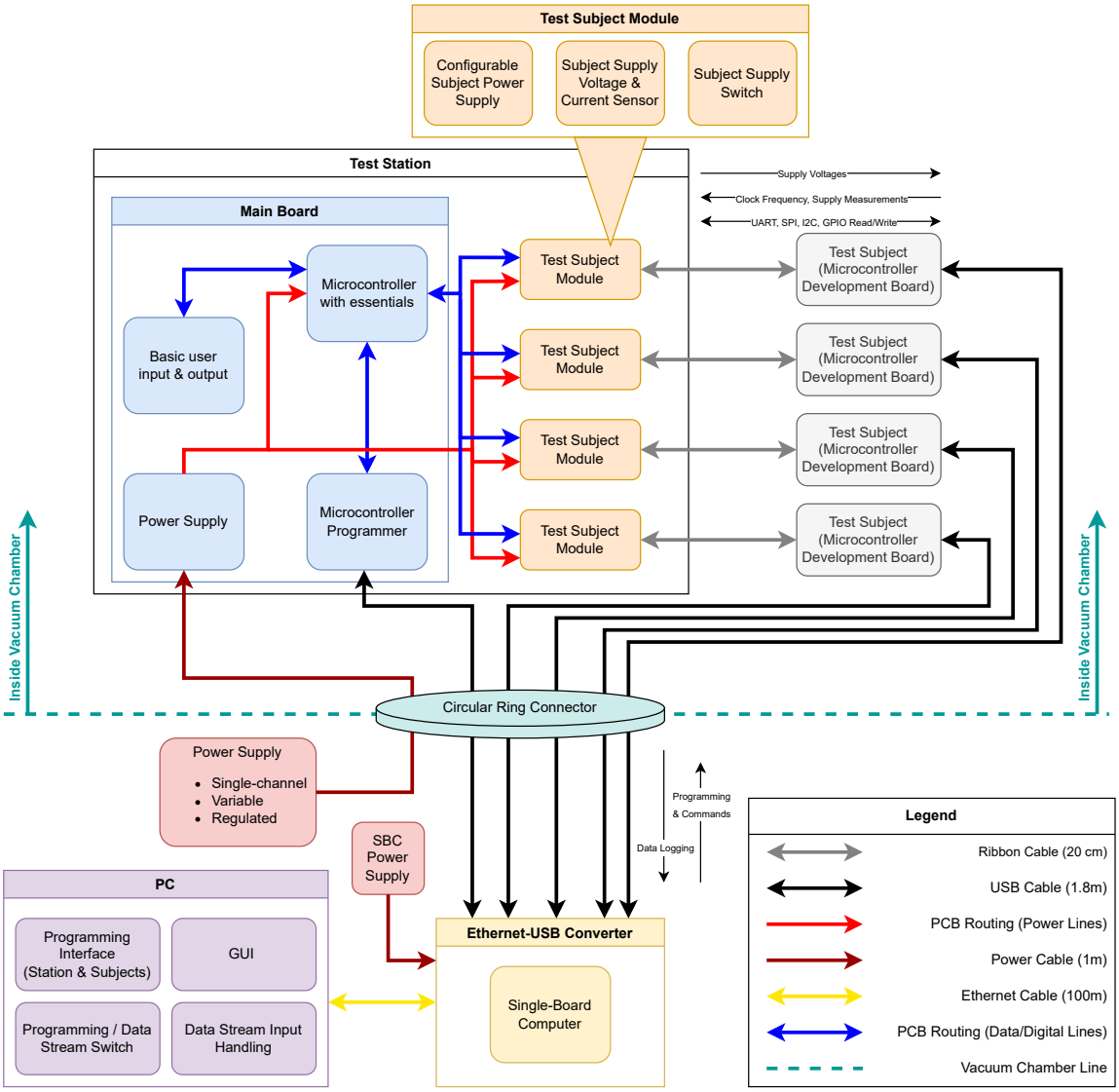


Figure 3.1: Functional block diagram of the system.

The heart of the system is the test station, which consists of a *main board* and four *modules* that plug into it (see Section 3.3). These are each connected to a separate test subject (see Section 3.2). Although only the subjects need radiation exposure, both they and the test station are placed inside the vacuum to limit the cables through the wire feedthroughs (see Section 2.2). The USB cables of the test subjects and the test station pass through one of these feedthroughs instead. A Single-Board Computer (SBC) is used to convert the USB data from the test subjects to Ethernet packages, which can be sent to a PC located about 100m away. Finally, a GUI on the PC displays data to a user and gives them control over the test station.

3.2 Test Subjects

Since the test station had to be STM32-based (Section 1.3.1), the subjects were chosen from the same MCU family to standardise the embedded software. Although a Nucleo-32's peripherals and power supply options would suffice, Nucleo-64s were chosen since they cost the same on Digi-Key, offer more pins, and have a larger surface area, which would allow for better radiation exposure. NUCLEO-F303REs were selected due to their familiarity.

3.3 Test Station

This section explains the high-level design of the test station's main board and four modules.

3.3.1 Test Station Modules

3.3.1.1 Configurable power supply

The configurable power supply required for each subject (Section 1.3.1) was designed to cover all possible voltage selections for the subject, namely VIN (7-12V), E5V (4.75V-5.25V) and 3V3 (3-3.6V) [32]. A custom jumper-header combination was designed to allow for easy selection between these three. Regulators were required for the latter two, while VIN would be supplied from outside the vacuum. Additionally, the 5V supply was designed to be selectable between a regulator and an external source, as required in Section 1.3.1. Section 4.1.1 explains the details of the design.

3.3.1.2 Power supply switch

A relay was used as a robust option to switch a subject's power supply, which would sometimes experience shorts (Section 1.3.1). Although sealed components were not allowed

in the vacuum, a hole could be drilled into them. Section 4.1.2 explains how a relay driver was designed such that the MCU could control it.

3.3.1.3 Current- and voltage sensor

A current- and voltage sensor was placed in series with the power supply and its switch, and before the switch, such that the voltage could potentially be double-checked by the user before supplying power to the subject. Section 4.1.3 details the design.

3.3.1.4 Connection with main board and test subjects

To easily connect a module to the main board and to a test subject, standard 2.54mm header-socket combinations were used, as Section 4.1.4 will explain. Four connection PCBs were created to connect each subject to its module with a single ribbon cable, which was much simpler than the wire harness of the first test station (Section 2.1.1). Section 4.1.5 gives more details on the connection PCB.

3.3.2 Test Station Main Board

3.3.2.1 Microcontroller choice

To select a microcontroller, a requirement was that it had to come in a Nucleo version, since its open-source schematics would simplify the design process, and if the main board PCB failed to work, a Nucleo could serve as a backup. Another requirement was an LQFP-64 package, which is easy to solder and has enough pins. Three candidates were narrowed down, and their details can be seen in Appendix D.1. For flexibility, the PCB was designed to support both a familiar STM32F303RE and a high-performance STM32F446RE, as explained in the same appendix.

3.3.2.2 Power Supply

A 3.3V regulator was needed to power the STM32 MCU [33]. To minimise power dissipation in the form of heat (which is challenging to get rid of in a vacuum), this regulator was supplied with 5V instead of a higher voltage like 12V. This 5V could then also be used to supply the relay and current- and voltage sensor on each module. To meet the requirement in Section 1.3.1 of having the option to switch between 5V coming from a regulator or from an external supply, it was decided to include jumper-header combinations to select between either 5V or a higher voltage like 12V. Having multiple voltages from a single supply line required a variable supply, which is why a variable bench power supply, that could provide any regulated voltage between 5V and 12V, was used. A bench power supply also gave the option of a variable current limit (up to a maximum of 3A), which would protect the system against the short-circuits caused by the radiation effects of Chapter 1.

To adhere to the requirement of safe system handling (see Section 1.3.1), reverse-polarity protection was included. Section 4.2.1 explains the power supply design in detail.

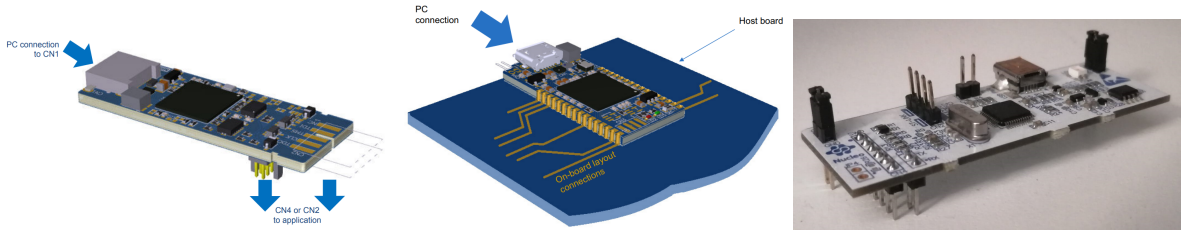
3.3.2.3 Basic user input and output

User output consisted of a debug LED and status LEDs to indicate the presence of various voltages and voltage selections. User input comprised a reset button, boot switch, and the mentioned jumper-header combinations.

3.3.2.4 Programmer

As the literature of Section 2.1.2 showed, using an ST-LINK to program an STM32 is the only approach that allows debugging, which was a desirable feature (see Section 1.3.1). It also had the benefit of communicating with the PC (for other reasons than programming) by using a straight-forward UART peripheral. This would require the ST-LINK to be permanently connected to the test station. Section 2.1.1 also showed that a USB device (such as an ST-LINK) can be extended over Ethernet using VirtualHere.

There were many options for an ST-LINK, including a custom one (outside the scope of this project), an external one like the ST-LINK V2 ISOL (not affordable) or a knockoff version of the ST-LINK V2, which could not be updated by the ST-LINK Utility since it only supports authentic ST-LINKs. Other options included the mentioned ST-LINK V3, like the one in Figure 3.2a, or even a snapped-off ST-LINK V2 from a Nucleo development board. Since a spare Nucleo board was available, it was decided to snap off its ST-LINK and use it. Although it would not be the most aesthetic, it would work fine for a first version of the new test station. It would also be one of the more mechanically secure options for connecting to the main board if some of the the headers were soldered to the bottom side (see Figure 3.2c), unlike the akward connector of the V3 MINIE. Although the ST-LINK V3MODS is more than half the size of the snapped off ST-LINK from the Nucleo and can be soldered on to the main board (see Figure 3.2b), this would limit the user's flexibility in case it stopped working.



(a) ST-LINK V3MINIE [13] (b) ST-LINK V3MODS [34] (c) Modified Nucleo ST-LINK

Figure 3.2: Various options for an ST-LINK programmer.

3.4 Single-Board Computer

As the literature of Sections 2.1.1 and 2.1.3 illustrated, no affordable and practical solution for extending USB ports over a distance of 100m for programming could be found besides using a single-board computer, as was done with the first test station. Since the Raspberry Pi was much more popular and hence had a much larger online community to consult than some of its competitors like the Banana Pi, Orange Pi, or Rock Pi [35], it was chosen. Specifically, the model 3B+ was selected because it was the cheapest one that had both an Ethernet port and four USB ports. Since an additional USB port was required, a small USB hub was used with it. Section 4.4 explains the setup and programming of the Raspberry Pi, and how it was used for two purposes - to program each test subject and the test station, and to send data logs to and receive commands from the PC.

3.5 Graphical User Interface

A Graphical User Interface (GUI) was developed to display live readings from the test station to the user and to store these readings for analysis at a later stage. It was also used to enable or disable test subjects should less than four be used during a test, and power could be applied to and removed from each subject independently. It also had other features such as writing to a GPIO pin of a subject and displaying graphs. Section 4.5 expands on the GUI functionality.

3.6 Layout and Size of Main Board and Modules

As will be mentioned in Section 4.1.5.1, the size of the main board was influenced by the manufacturing service, leading it to be 102X102mm. To determine a size for the module PCBs, four of them were arranged together with the snapped-off ST-LINK that was modified (see Section 3.3.2.4). The modules ended up being rather small (24x77.62mm; about the size of the ST-LINK), but they were still big enough for all the components. Placing the modules over the microcontroller hardware would serve the additional benefit of shielding the microcontroller from potential radiation damage. It was decided to overhang the modules a bit to make removal of them easier. Appendix D.2 includes a figure showing a rough layout of the modules and ST-LINK on the main board.

3.7 Project Specifications

Since some of the initial requirements in Section 1.3.1 were vague, it was necessary to define clear metrics against which the system would be measured in Chapter 5. This

section summarises these specifications, and Appendix C contains a more detailed list.

3.7.1 General Hardware

Hardware must be on three PCBs - the main board, four modules, and four connection PCBs. Users must be able to connect any number of subjects and modules for radiation testing, with no duplicate hardware permitted on the main board. Only one-way connectors are allowed, and reverse-polarity protection for the main power supply is required. Regulators must maintain junction temperatures below 80% of their rated values. No electrolytic capacitors, sealed components, or thermal paste may be used.

3.7.2 Module Hardware

The voltage sensor must maintain a 5% accuracy over 0-12V, and the current sensor a 5% accuracy from 0mA to the maximum subject current. Voltage selections include 3.3V (3-3.6V), 5V (4.75-5.25V), and 7-12V. Each module should offer a 5V selection from its regulator or an external source. All voltages must sufficiently power the subjects.

3.7.3 Embedded Software

UART, I2C, and SPI transmission and reception between the station and all subjects must be verified. The station needs GPIO read/write capabilities for all subjects, with clock accuracies within 1% of a subject's system clock (72 MHz). It must operate independently of long-distance communication, allowing direct logging to and control by a PC. The data logging frequency should be adjustable with a minimum of 1 Hz. Finally, all software should function even with only one subject and module plugged in.

3.7.4 Long-Distance Communication Scheme

Programming of the MCUs and data exchange between them and the PC must happen over Ethernet, with the subjects reporting some of their data directly to the PC.

3.7.5 Graphical User Interface

Subjects must be enabled and powered individually, and not all at the same time. A test must be startable and stoppable from the GUI and the elapsed test time must be displayed. The logging frequency must be changeable. Writing to a subject's GPIO pin is required and must be verified. Simultaneously displaying voltages, currents, clock frequencies, UART-, I2C-, and SPI statuses, and GPIO readings for all subjects must be optional. Optional CSV file storage and graphs displaying at least two live subject parameters are required.

Chapter 4

Detailed Design

This chapter explains the detailed design of all the elements mentioned in Chapter 3, which includes the hardware of the modules and test subjects, hardware of the main board, embedded software of the test station and subjects, Raspberry Pi hardware and software, and GUI software. The sizes of components like resistors, capacitors, and LEDs mentioned in the hardware design sections of this chapter are 1206 unless otherwise stated. Although large, this size was chosen to simplify soldering.

4.1 Hardware of Modules and Test Subjects

4.1.1 Configurable Power Supply

As mentioned in Section 3.3.1.1, two linear regulators (3.3V and 5V) were needed to power the test subjects, which had a maximum current draw of 300 mA [32]. Switched-mode regulators were avoided for their high cost and fragility.

4.1.1.1 5V regulator

The popular L7805CV was chosen as the 5V regulator since it has short-circuit protection, can handle input voltages of up to 35V (larger than 12V), and can deliver currents of up to 1.5A [36]. Input- and output capacitors of 330 nF and 100 nF, respectively, were included according to the datasheet. Ceramic capacitors were used without regard for ESR and possible instability, because its datasheet states that no output capacitor is needed for stability.

4.1.1.2 3.3V regulator

By supplying the 3.3V regulator with the 5V one, the heat dissipation would be spread between the two. The LD1117 was chosen because of its wide availability, dropout voltage of 1V (less than 1.3 V), internal current limiting, and output current of up to 800 mA [37]. A 100 nF ceramic capacitor was used at the regulator's input, as recommended by the datasheet. The regulator required a large 10 µF output capacitor to ensure stability, which is typically only available in electrolytic or tantalum packages. Since the former cannot be used (see Section 3.7) and the latter can leader to thermal runaway, fires, or small explosions when strained [38], ceramic was left as the only option. The only concern was

ESR, which is sufficient in electrolytic or tantalum capacitors, but virtually non-existent in ceramics [39]. To combat this, a low-value resistor can be used in series with the capacitor to dampen the output voltage ringing. Hence, a 0 Ω resistor was inserted to give the user the option of changing the ESR if necessary. The value can be calculated using Eq. 4.1 [39], where $f_{oscillation}$ can be measured using an oscilloscope and a test circuit consisting of a function generator driving a MOSFET that switches a load resistor [39].

$$ESR_{min} = \frac{1}{2\pi f_{oscillation} C_{out}} \tag{4.1}$$

4.1.1.3 Thermal considerations

Thermal design was done while considering the specification of keeping T_J under 80% of its rated junction temperature (Section 3.7). The T0-220 package was selected for the regulators due to its good thermal resistance. For both regulators, $\theta_{JA} = 50$ °C/W and $T_{J(max)} = 125$ °C [37], [36]. According to Eq. 4.2, the junction temperature of the LD1117 would rise to 50.5 °C for an ambient temperature of 25 °C and a current of 300 mA, which meets the specification. For the L7805, a worst-case scenario would be a supply voltage of 12V and a supply current of 370 mA, where the relay is also powered by the regulator. With $T_A = 25$ °C, T_J would rise to 154.5 °C, which does not meet the specification. Assuming that the main board's regulator supplies the relay, a maximum subject current draw of 214.3 mA is possible, while 144.3 mA is allowed when the module's regulator supplies the relay. Although below 300 mA, the current draw of a NUCLEO-F303RE was measured to luckily never exceed 100 mA (see Section 5.1.3). Therefore, a heat sink was not used. Besides making the modules bulky, it would not be very effective, as mentioned in Section 2.2. Instead, the PCB was designed such that its copper pours (with good conductivity) could be used as a basic heat sink (see Section 4.1.6 for details).

$$\begin{aligned} T_J &= P_D \times \theta_{JA} + T_A \\ &= [(V_{IN} - V_{OUT}) \times I_{OUT}] \times \theta_{JA} + T_A \end{aligned} \tag{4.2}$$

4.1.1.4 Final features and schematic

Solder jumpers were placed at the input of each regulator to isolate them and to simplify testing and debugging. Three LEDs were added to inform the user of the presence of a subject's supply voltage. The desired LED brightness occurred at 10 mA, which translates to a forward voltage of 2 V [40]. The resistances were calculated and the closest standard values in the available kit were selected as 1 kΩ, 510 Ω, and 220 Ω, for 12V, 5V, and 3.3V, respectively. The complete module power supply circuit excluding the LEDs can be seen

in Appendix F.1.1. The terminology is explained in Section 4.1.1.5.

4.1.1.5 Clarification of module power supply terminology

It is worth clarifying the meanings of the module's power supply terms used in the schematics and referred to from this point onwards to avoid confusion:

- *MB_+7-12V*: The 7-12V that the main board obtains from outside the vacuum (should the main board's jumper be placed on "E7-12V").
- *MB_E5V*: The 5V that the main board obtains from outside the vacuum (should the main board's jumper be placed on "E5V").
- *MB_+5V*: The 5V from the main board, which is obtained from either its 5V regulator or from outside the vacuum (should the main board's jumper be placed on "E5V").
- *+5V*: The 5V supply to the subjects, which is obtained either from the module's 5V regulator or from *MB_E5V*.

The reason for these terms is that the relay and current/voltage sensor of the module should be powered from a separate source than the supply to the subjects, because the former two should have a reliable supply, and the supply to the subjects will be unstable because of the regular shorts caused by the subjects during radiation testing.

4.1.1.6 Voltage selection jumper

As mentioned in Section 3.3.1.3, the current- and voltage sensor was placed in front of the relay to allow the user to double-check the voltages before applying power. A diagram in Appendix F.1.1.1 shows how different voltages can be selected by placing dual two-pin jumpers on one of three four-pin male headers.

4.1.2 Relay Driver

The design of the relay driver mentioned in Section 3.3.1.2 is discussed in this section. The chosen relay's coil resistance of 70 Ω [41] would draw 71.4 mA when powered. The STM32 cannot supply this current directly from its GPIO pins, since the maximum output current that guarantees a stable voltage for any of its GPIO pins is 8 mA [33]. Therefore, a BJT (2N2222A) was used as a low-side switch to drive the 5V relay, and not a MOSFET, because the latter is generally more expensive. A common 1N4148 was used as a flyback diode to prevent voltage spikes when the relay would be turned off. A status LED was introduced to provide feedback to the user. Since the LED current in parallel with a base current of the BJT would be too much for a single GPIO pin, an optocoupler was used. Finally, a three-pad solder bridge was used for the relay's power supply to give the user

more flexibility. Motivations for component choices, the calculation of resistor values, and the final relay driver circuit can be seen in Appendix E.1.1.

4.1.3 Current- and Voltage Sensor

Besides the 5% current- and voltage measurement specification (Section 3.7) the sensor introduced in Section 3.3.1.3 had to be immune against short-circuits for at least a few seconds, and had to utilise serial communication. The INA219(A) [3], a single-channel current- and voltage sensor, was chosen because it can read bus voltages up to 26V, has maximum current- and voltage measurement errors of ±1%, and communicates via I2C. It has a programmable gain, which allows for resolution-tuning. It also comes in a breakout board version, which would make prototyping easier. Appendix E.1.2 explains why the sensor would be immune to short-circuits, and it gives more details on the circuit design. It motivates why a sense resistor of 100 mΩ was used, the power supply choice, and the choices of I2C pull-up resistors and address pins. Appendix F.1.2 includes a schematic of the final INA219 circuit. Section 4.3.2 explains how the INA219 is programmed to send current- and voltage measurements to the test station.

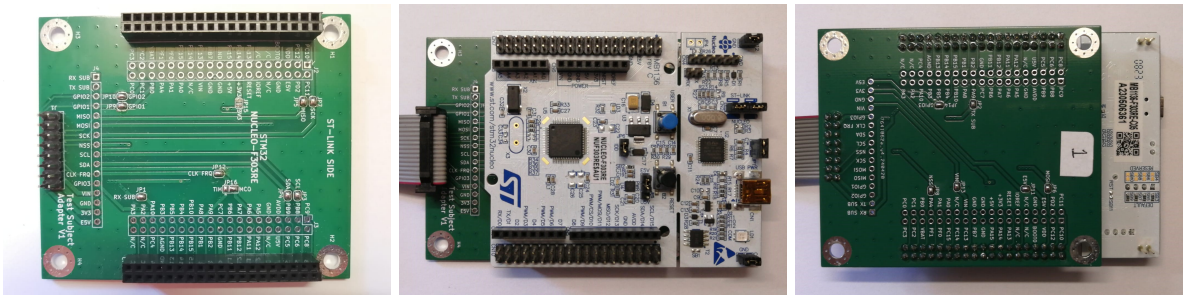
4.1.4 Module Connectors with Main Board

The module connectors introduced in Section 3.3.1.4 were designed in light of the PCB size and shape constraints on the modules (see Section 3.6). The verdict was that the four modules would have to be next to each other, with most of the main board's hardware beneath them. To ensure a rigid connection while leaving space beneath the modules, two sets of male headers were placed on the underside of the modules (the female headers were placed on the main board, since power would come from there). These sets were made different in size and shape to prevent the user from plugging the module in backwards, according to the specification in Section 3.7. The one set was for the various voltages and I2C lines for the sensor coming from the main board, and the other for the pins going to the subjects, most of these coming directly from the main board. As seen in the system diagram of Section 3.1, 20 cm ribbon cables were used to connect the test subjects to the main board via a connection PCB (see Section 4.1.5). A schematic of the module connectors showing all the signals on them can be seen in Appendix F.1.3.

4.1.5 Connection PCB

A connection PCB was designed for each module, as mentioned in Section 3.3.1.4. This consisted of two 2x19 female headers for a test subject, and a 2x8 male header for the ribbon cable. Solder bridges were added for each connection with the test station, and an unpopulated, 1x16 header was added to serve as testing points for an oscilloscope probe. A

three-pin solder bridge was added to select between two different sources of a clock output from the test subject (see Section 4.3.4). Figure 4.1 shows a test subject and ribbon cable plugged into one of the connection PCBs, and Appendix G.1 shows the full schematic.



(a) Top view (unconnected) (b) Top view (connected) (c) Bottom view (connected)

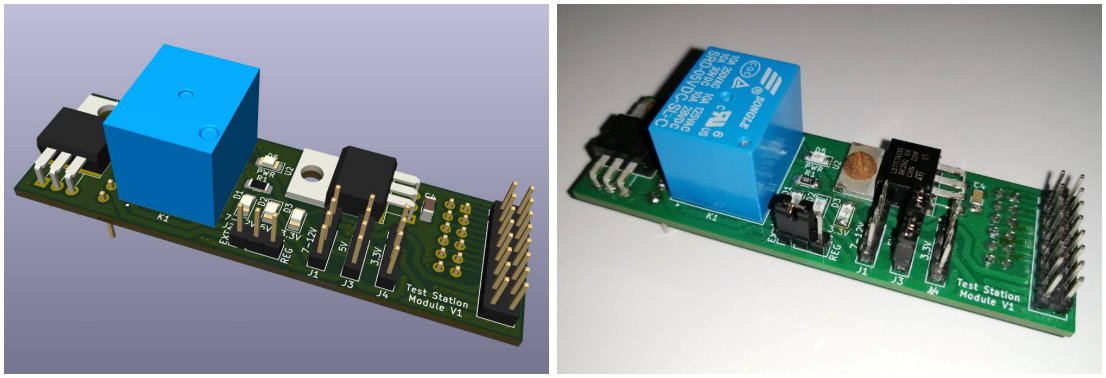
Figure 4.1: A connection PCB with a test subject and ribbon cable.

4.1.5.1 PCB Software and Manufacturing Service

KiCad was chosen for the PCB design, since it is free and has a reasonable learning curve. The popular JLCPCB was used for PCB manufacturing. Since their minimum offer of 2$ was applicable to any PCB smaller than 102x102mm, it was decided to make the main board no larger than this (see Section 3.6).

4.1.6 Module PCB

A ground pour was used as an artificial heat sink for the voltage regulators on the module to minimise their junction temperatures in the event of a short caused by an SEE or TID. The copper was exposed to increase the thermal conductivity to the pads and the regulators were spaced as far apart as possible. Thermal paste was not used, as per Section 2.2. Three vias were used instead of one to ensure reliable connections. Figure 4.2 shows the modules in KiCad and in real life, and Appendix H.2 shows the top- and bottom views. A complete module schematic can be seen in Appendix G.2.



(a) KiCad (b) Assembled

Figure 4.2: Perspective view of the module (see Appendix H.2 for detailed views).

4.2 Hardware of Main Board

4.2.1 Power Supply

This section discusses the design of the main board's power supply, which was introduced in Section 3.3.2.2.

4.2.1.1 3.3V regulator

Since the 3.3V regulator would mainly be needed to supply the MCU and its essential components, thermal considerations were not as important as with the modules. The regulator would need to handle at least 160 mA, which is the maximum current consumption of an STM32F303RE [33]. An extra 40 mA was added to compensate for LEDs and other small components. The popular AMS1117 was chosen since it can deliver 900 mA, comes in a small SOT-223 package (no shorts would occur), and has an acceptable dropout voltage of 1.3V [42]. Although the datasheet recommends a 22 µF tantalum output capacitor for stability, this is only needed if the adjust pin has a capacitor as well, which was not the case, so a smaller, 10 µF capacitor could be used. This capacitor had a 0 Ω resistor in series to increase the ESR if later required (see Section 4.1.1.2). An additional 68 nF ceramic capacitor was used in parallel with the series branch, which would provide good mid- and high-frequency decoupling characteristics to minimise output ripple due to the digital electronics [43]. A standard 100 nF capacitor was used at the input.

4.2.1.2 5V regulator

The 5V regulator and its design are identical to those in Section 4.1.1.1. It was decided to again use the TO-220 package for its good thermal resistance. Since the 3.3V regulator's supply current requirement is 200 mA, it dissipates roughly 340 mW when powered by 5V. Therefore, the 5V regulator would need to supply 68 mA to it. The relays each draw 70 mA (see Section 4.1.2), and the current consumption of the current- and voltage sensors used in the modules are negligible at 1mA each [3]. This totals to a total demand of roughly 350 mA. If the regulator is supplied by the maximum supply voltage of 12V, its junction temperature would reach about 147.5 °C (using Eq. 4.2), which is unfortunately above its limit of 125 °C. The only way to ensure it stays under 100 °C when all the relays are powered by this regulator is to use a supply voltage of 9.3V or less. An alternative was to let two modules power their relays with their own 5V regulators, which would make even a 12V supply safe. Considering that the regulator would be mounted to the massive ground plane of the main board, it was decided to not use an additional heat sink for the same reasons mentioned in Section 4.1.1.3.

4.2.1.3 Reverse-polarity protection

A power connector with a key was used, as specified in Section 3.7. Reverse-polarity protection was also a specification since the wires to the bench power supply could potentially be swapped. Although a diode is a simple solution, it introduces a voltage drop which would be a nuisance when wanting to power the subjects with a precise, external 5V. Power consumption due to the drop is also a concern. A Schottky diode has a smaller voltage drop, but a larger reverse-leakage current, which would lower the protection [44]. Although some Schottky diodes have low reverse-leakage currents, the voltage drop is still significant. A P-Channel MOSFET is a better solution. Although it also introduces a voltage drop due to its drain-to-source resistance $R_{DS(on)}$, this is very small for some MOSFETs. The maximum V_{DS} and maximum V_{GS} also have to be higher than the input voltage. The IRF4905 was selected for its superior $R_{DS(on)}$ of 20 mΩ and sufficient current- and voltage ratings [45]. To contextualise, the voltage drop of an 1N4007 rectifier diode is 1.1V at 1A [46], while that of the popular 1N5817 Schottky diode is 450 mV at this current [47]. The MOSFET, however, would drop a mere 20 mV at the same current [45]. Appendix E.2.1 explains how this works.

4.2.1.4 Final features and schematic

Status LEDs were added to indicate the presence of various power supply voltages. Being the same as those in Section 4.1.1.4, their resistors had the same values. As mentioned in Appendix M.2.1.1, these simple LEDs were responsible for saving the test subjects from possible damage. Lastly, a jumper was used to select between E5V and E7-12V, and a three-pin header was included to optionally bypass the MOSFET in case it did not function properly. Appendix F.2 contains the power supply schematic.

4.2.2 Microcontroller and Supporting Components

As mentioned in Section 3.3.2.1, the STM32F303RE was chosen for the test station's microcontroller, while the STM32F446RE served as a backup. To design the surrounding hardware, the application note for each MCU [48], [49] and the schematic for the Nucleo-64 board [50] were consulted.

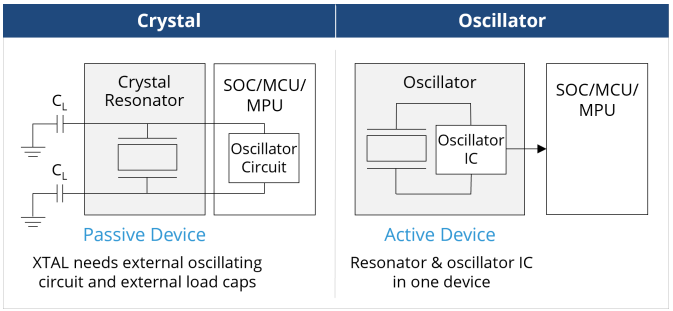
4.2.2.1 Power supply pins and decoupling capacitors

Since the "analogue" version of V_{DD} (V_{DDA}) always had to be greater than or equal to V_{DD}, their pins were tied together. For the same reason, V_{SSA} was connected to V_{SS}. Although a filter could be used to obtain a cleaner V_{DDA}, there were no clear guidelines as to how to design this, and V_{DDA} is mainly used for ADCs, DACs, and comparators [48], which were not used. Two decoupling capacitors for V_{DDA} with values 1 µF and 10 nF

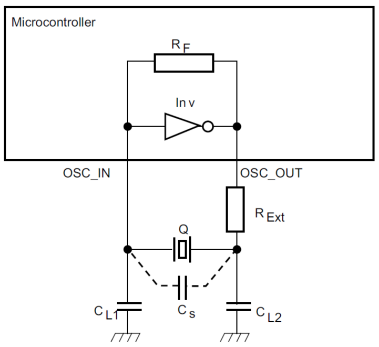
are recommended [48], but the latter was replaced with a 100 nF one since the Nucleo schematic did so [50]. V_{BAT} was tied to V_{DD} since no external battery was used, and each V_{DD} pin (including V_{BAT}) was decoupled to ground with a 100 nF decoupling capacitor, while one 4.7 µF capacitor was placed in parallel with them all [48]. Since the latter is recommended to be an electrolytic, a small (1 Ω) resistor was instead added in series with a ceramic to increase the ESR, as in Section 4.1.1.2. The decoupling capacitors were placed close to the relevant pins, as recommended, and a 4.7 µF capacitor was added on the $VCAP\_1$ pin of the STM32F446RE, which had to be soldered if this MCU would be used instead.

4.2.2.2 HSE oscillator

Due to potential frequency drift in the MCU's internal RC oscillator, which affects accurate clock frequency measurement, a High-Speed External oscillator (HSE) was designed. It is worth clarifying the difference between a crystal *oscillator* and a crystal *resonator*, or simply a *crystal*. The former has the oscillator circuit built into the package, while the latter requires external circuitry. Figure 4.3a illustrates this difference. Since pins PF0 and PF1 of the MCU are connected to an internal feedback resistor and inverter, an external crystal was used instead. This configuration is known as a Pierce oscillator (Figure 4.3b).



(a) Difference between a crystal and an oscillator [51]



(b) Pierce oscillator [52]

Figure 4.3: Types of crystal oscillator circuits.

Appendix E.2.2 explains the crystal choice and details the calculations for the capacitors and resistor, carefully following the relevant application note. Section M.2.2.2 shows the results for the HSE oscillator.

4.2.2.3 LSE oscillator

Although not as crucial as the HSE oscillator, a Low-Speed External (LSE) oscillator was also designed to give the user the option of using a more accurate clock. Luckily, the 32.768 kHz crystal used in the Nucleo schematic (ABS25-32.768KHZ-6-1-T [53]) was available online, so the design was simply copied along with its two external 4.3 pF capacitors [50].

4.2.2.4 Other components

Other components related to the microcontroller itself included a reset button with a 100 nF capacitor in parallel for hardware debouncing, a SPDT switch to pull the BOOT0 pin high or low by means of a pull-down or pull-up resistor, and a debug LED, for which a 300 Ω resistor was used to obtain a current of approximately 5 mA at a voltage drop of 1.92 V [40], which was below the 8 mA limit of the GPIO pins [33].

4.2.2.5 MCU pin assignments and final schematic

As mentioned in Section 3.3.2.1, the pin assignments for the MCU were done such that both the STM32F303RE and the STM32F446RE would be compatible with the test station. For each peripheral it was ensured that the relevant pins were available on both the 303 and the 446, otherwise it was not used. TIM4 is an example, which is available on the 303, but not on the 446, which is why TIM8 was used for the fourth test subject's clock frequency pin instead. To meet the specification of GPIO reading and writing (Section 3.7), almost all the pins were used such that each subject could have three GPIOs (see the related software in Section 4.3.3). Appendix O contains a table of the final pin assignments, including the specific peripherals that were used. Appendix F.2.1 shows a schematic of the final MCU circuit with all its surrounding components.

4.2.3 ST-LINK connections

To be safe, it was decided to copy the exact connections between the Nucleo's ST-LINK and its MCU, as shown in its schematic [50]. This is the reason for the 22 Ω resistors shown in the schematic of Appendix F.2.2.

4.2.4 Serial Communication Hardware

To interface with the test subjects via I2C, pull-down resistors were required. As with the INA219, standard 2.2 kΩ resistors were used on both the Serial Data (SDA) and Serial Clock (SCL) lines, and these would be proved to yield acceptable signals in Section 5.3.5. Section 4.3.5 explains the I2C software. Since UART is a point-to-point communication protocol, the test station could only support one subject per peripheral, requiring four peripherals. Although the chosen MCU has five UART peripherals on-board, conflicts would occur between these, allowing only some to be used. Instead, a single UART channel was used together with three nested AND gates [54] (see schematic in Appendix F.2.3). Since a UART signal is in a logical high state by default, the AND gate would output a logical zero if any one of the test subjects pulled it low. This way, the subjects could transmit data to the station one at a time. Luckily, the test station could transmit to all subjects at the same time without requiring additional hardware. Section 4.3.6

explains the UART software. Since not all the modules would always be plugged into the test station (see Section 3.7), pull-up resistors had to be used at the AND gate inputs. This was unfortunately overlooked in the design, but Section 5.3.4 explains how this was circumvented. The SPI pins were also connected between the MCU and the subjects. A Not Slave Select pin (NSS) was required for each subject [55].

4.2.5 Connectors and Test Points

The main board used female connectors complementing the male connectors on the modules (Section 4.1.4). Test points were added for most GPIO pins to aid debugging. Though not very professional-looking, they would usually be hidden under the modules.

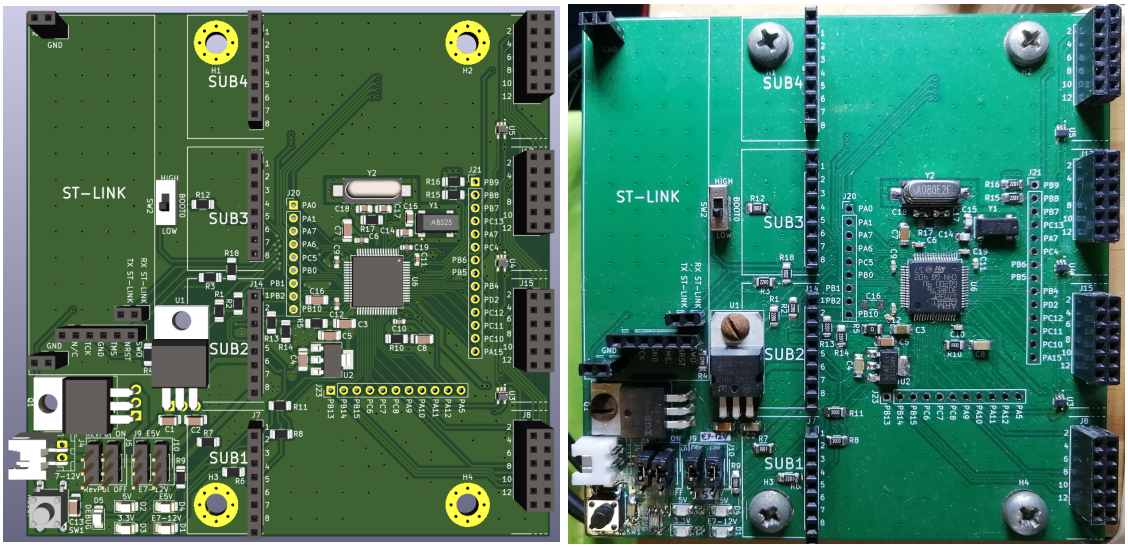
4.2.6 Main Board PCB

4.2.6.1 Design considerations and images

PCB design choices were influenced by datasheets and tutorials [56]. An application note was followed as a guideline to design the oscillators, which included close placement to the STM32, and local ground planes around the oscillators to shield them from the noise introduced by the digital electronics [52]. Signal integrity was optimised by using stitching vias for the ground planes and by utilising three vias where one would typically be used.

4.2.6.2 Images and schematic

Figure 4.4 shows a comparison between the PCB in KiCad and in real life. Appendix H.1 shows the underside. Appendix G.3 shows a complete schematic of the main board PCB, and Appendix H.3 includes an image of the modules and ST-LINK on the main board.



(a) KiCad rendering (b) Assembled PCB

Figure 4.4: Top views of the main board in KiCad and in real life.

4.3 Software of Test Station and Test Subjects

This section explains the embedded software of the test station and test subjects. To program the various STM32 MCUs, the standard STM32CubeIDE was used. Figure 4.5 shows a system diagram of the embedded software that will be discussed. All the software can be accessed through the link in Appendix I.

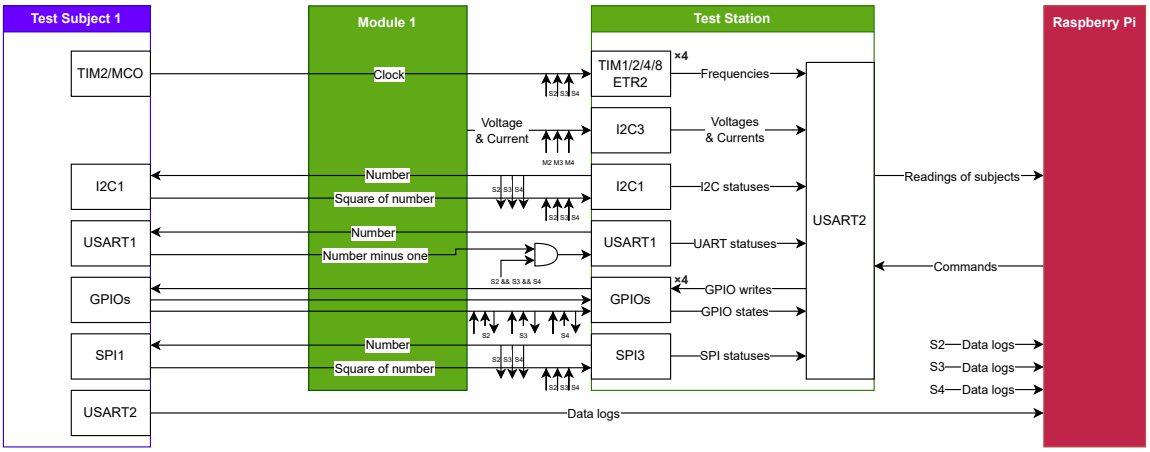


Figure 4.5: System diagram of the embedded software.

4.3.1 Test Station Software Fundamentals

Since the test station would contain a lot of code, header- and source files were used for clarity. To make the software more modular, the INA219's firmware was also kept separate. The main source file included the main program logic while the other files included the functions and Interrupt Service Routines (ISRs). In the main file, the test station would prioritise UART commands from the PC (Section 4.3.8.1), where a flag would be set for an incoming command which the main loop would detect, and the command would be processed. If the test was running, the readings of enabled subjects would be updated and printed, and the loop would delay for a time set by the PC. A struct was used for each subject to group its variables together, which would enable one to simply pass struct pointers as function arguments instead of individual variables. A diagram of the main file's logic and a screenshot of the test subject struct can be seen in Appendix E.3.1.

4.3.2 Current- and voltage sensor

After controlling the power supplies with relays using straight-forward `HAL_GPIO_WritePin` functions, it was first necessary to measure the currents and voltages of these supplies before other readings could be reported. This section explains the firmware of the sensor for which the hardware design was done in Section 4.1.3.

The INA219A has six 16-bit registers. These are each treated as two 8-bit registers in I2C communication, and most-significant bytes are written and read first. The configuration register is used during initialisation to set the full-scale bus voltage range (16V or 32V), full-scale shunt voltage range (40, 80, 160, or 320 mV), and ADC resolution/averaging. Other registers include the calibration-, shunt voltage-, bus voltage- and current registers [3].

4.3.2.1 INA219 firmware

The INA219 driver code is based on a GitHub library [57] inspired by Adafruit's INA219 library [58]. To modify any register of an INA219 on the I2C bus, one must initially write the address of the specific register to the sensor, after which the register's contents can be written. Although this can be done by calling two `HAL_I2C_Master_Transmit` functions, a more concise way is to use `HAL_I2C_Mem_Read`, which takes the memory address and its size as additional arguments. Since the registers are 16 bits in size and the function takes an 8-bit data buffer, two consecutive bytes need to be written. The function assumes a 16-bit register address. A similar approach is required for reading from a register, where `HAL_I2C_Mem_Read` can be used. The complete function definitions can be found in Appendix Q.1.3. The first step is to call a function that both resets the sensor and calibrates it. A reset is performed by writing 0x8000 to the configuration register, which sets the RST bit to 1 and the other bits to zero. This operation sets all registers to their default values and clears the RST bit again. Following this, a function writes a desired calibration value to the calibration register, and then the configuration register is updated with the desired value. The value for the calibration register is calculated using Eq. 4.3. The term $Current\_LSB$ can be calculated using the maximum expected current, as Eq. 4.4 shows. The result is then rounded up to the nearest milliamp to simplify calculations. The two variables `supplyVoltage_V` and `supplyCurrent_mA` are updated for each test subject struct (see Section 4.3.1) from within the main program loop. To read the current, the current register's contents are read and then multiplied with $Current\_LSb[mA]$. The bus voltage has a constant LSb of 4 mV, which is multiplied with the contents of the bus voltage register after bit-shifting the contents to the right by three, which is necessary because the rightmost three bits of the bus voltage register stores data other than the bus voltage [3].

$$Cal = trunc(\frac{0.04096}{Current\_LSb[A] \times R_{Shunt}[\Omega]}) \tag{4.3}$$

$$Current\_LSb[A] = \frac{I_{max(expected)}[A]}{2^{15}} \tag{4.4}$$

4.3.3 GPIO Input and Output Measurement

As mentioned in Section 4.2.2.5, three GPIO pins were assigned to each test subject for GPIO reading and writing. It was decided to use two of these to read the *state* of a test subject (using `HAL_GPIO_ReadPin`), and the third to write to a test subject's GPIO pin (using `HAL_GPIO_WritePin`). Having two GPIOs meant four possible combinations, or *states*. State zero (with both GPIOs off) was reserved for an *error* state, which would be the case when the subject was not powered. The subjects would iterate through states one and two every second, which would allow the user to verify on the GUI that state changes did indeed happen. State three was used to report whether the subject detected the test station writing to its third GPIO pin. The results can be seen in Section 5.3.2.

4.3.4 Clock Frequency Measurement

To meet the requirement of clock frequency measurement in Section 1.3.1, it was decided to let each test subject toggle one of its GPIO pins at a scaled version of the system clock frequency, which was set to the default value of 72 MHz. This could be accomplished using either a timer or the dedicated Microcontroller Clock Output (MCO) pin, which can be enabled in `System Core>>RCC` in the device configuration tool. As mentioned in Section 4.1.5, a three-pin solder bridge was included to select between these two. To measure this frequency with the test station, two solutions were considered.

4.3.4.1 Input capture mode

The first and rather common solution involved a timer in input capture mode, which could be used to measure the time between two rising edges of the input signal. Input capture mode works by storing the counter value (CNT) in the Capture and Compare Register (CCR) when the external signal changes, such as with a rising edge of a clock signal. When the global interrupt of the timer is enabled, an ISR would be called on every rising edge for example. Inside ISR, the contents of the CCR can be stored in a first variable and a flag can be set to indicate that it is waiting for another rising edge. When the ISR is called again, the variable can be subtracted from the current CCR value. Multiplied by the timer's clock period, this gives the external signal's period, which can be inverted to find the frequency. Furthermore, an overflow interrupt can be enabled to account for counter overflow at low frequencies. Keeping track of the number of overflow interrupts, which would happen at low frequencies, these can be measured too [59]. A benefit of this approach is that it uses only a single timer with four inputs, thus saving on timers. However, an interrupt on every rising edge would not leave more room for other tasks, especially considering the high external input frequency of 72 MHz and that there are four of them. Although a clock divider can be used to trigger an interrupt only every

eighth rising edge and although the external signal may be scaled down from 72 MHz to at most 1 MHz (see Section 3.7), these limitations would still make input capture mode problematic.

4.3.4.2 External clock source

The second solution, partially inspired from the original test station (see Section 2.1.1), involved a timer with an external clock source and another, simpler timer clocked by the internal clock. The options for an advanced timer's clock source in the device configuration tool are the internal clock and *ETR2* (and *ETR2 through ADC remap*). When selecting ETR2, a pin is assigned to the MCU where the external clock signal for the timer can be applied. A basic timer (like TIM6) can then be set up in interrupt mode to call an ISR every few milliseconds. This handler can then read the current count value of the more advanced timer, store the value, and reset the count such that it starts counting from zero again. This way, the advanced timer never reaches the auto-reload value. The benefit of this approach is that the high frequency timer never triggers an interrupt, because its interrupts are disabled in the NVIC settings. This method can also monitor clocks with much higher frequencies than the first approach. However, the method restricts one to a single channel per timer, and these timers all have to be advanced timers that support the ETR2 functionality. Luckily, both MCUs have five timers with this functionality, and when doing the MCU pin assignments (see Section 4.2.2.5), there were no conflicts with other peripherals when using four of these timers simultaneously.

4.3.4.3 Calculations and software setups

The calculations for the auto-reload value and prescaler can be seen in Appendix E.3.2. This appendix also includes the software setups for the test station and test subjects.

4.3.5 I2C Verification

To verify the I2C functionality for which the hardware design was done in Section 4.2.4, the test station was set up to transmit an I2C message to the particular subject, and to wait up to a timeout value to receive an I2C message back. To keep track of the specific verification instance, a different message was sent each time - one of 14 numbers in an array. As long as the test station received the square of the number, the I2C was operational. Although the utilised polling mode receive of the test station was not as efficient as interrupts or DMA would be, this was affordable since the MCU was not doing intense processing. Appendix E.3.3 includes the software setups for the various MCUs, and the results can be seen in Section 5.3.5.

4.3.6 UART Verification

As mentioned in Section 4.2.4, one UART peripheral was designed to communicate with four test subjects by means of three cascaded AND gates. As with the I2C, the test station would transmit a message to each subject and wait for a response up to a timeout value. If it received one minus the number sent, it would report that the UART was operational. For an undetermined reason a subject could not receive multiple bytes from the test station at a time, since the start and end of a message could not be detected reliably, as opposed to the I2C. The test station's receive had the same problem. To resolve this, a polling mode UART transmit function was called for every byte to be transmitted, and reception happened via interrupts. A custom communication format was used for both the station's and subjects' receive methods to identify message starts and ends using the characters `@` and `!\n`, respectively. This method of reception allowed for any message length up to a predefined maximum. The callback function named `HAL_UART_RxCpltCallback` (contents in Appendix Q.1.1) would be called after one byte had been received, and from there the UART receive would be re-initialised. Valid messages would then be processed with another function. Since all subjects shared the same peripheral, each subject's ID was included in the message transmitted to it and by it. The software setups for the test station and test subjects can be seen in Appendix E.3.4.

4.3.7 SPI Verification

The SPI software was not implemented due to a lack of time. However, its verification would have worked in the exact same way as the UART and I2C. The intent was to follow a promising tutorial which explained how to write an inter-MCU SPI driver [60].

4.3.8 Data Exchange via a Terminal Program

4.3.8.1 UART communication with PC

To exchange data between the PC and the test station via UART, USART2 was used as a virtual COM port, since the ST-LINK could act as a UART-USB bridge using this peripheral. Commands were sent using Termite for its convenient command box, but PuTTY was used for reception since it displayed readings more legibly. A function called `consolePrint` was used that utilised `HAL_UART_Transmit` to replace the convenient `printf` function, which is not available in STM32 unless using Serial Wire Output (SWO) [1]. To receive data from a terminal program, the same UART peripheral was set up in interrupt mode to receive one byte at a time, for the same reason as Section 4.3.6. The full function definitions of `consolePrint` and of the callback function can be seen in Appendix Q.1.2.

[1] Although the test station was designed to include support for SWO (see Section 4.2.3), it was decided not to rely on this and to rather make use of the more familiar UART.

4.3.8.2 Logging readings

A function called `printLogs` was written to send all the readings of a particular test subject over the serial port using the `consolePrint` function from Section 4.3.8.1. Readings were condensed into efficient, single lines per subject, using letters to denote different readings (e.g., `N` for subject number) as Figure 4.6 shows. Start- and end characters were added to each set of readings to guide the GUI in grouping them (see Section 4.5.1). Section 5.3.1 shows these readings in a terminal program. These commands came from the test station and not directly from the subjects, since the latter functionality was found to be problematic (see Section 5.4).

```
N@X!E@X!P@X!G@X!V@X.XXX!C@XX.XX!F@XXXXX.X!U@X!I@X!S@X!\n
```

Figure 4.6: Test subject readings format over Ethernet cable.

4.3.8.3 Receiving commands

To receive commands from a terminal program, the function of Section 4.3.8.1 was used, and commands had to be in a specific format (Table 4.1 [2]). They also had to be in a logical order, e.g., the user was not allowed to power a subject if it was not first enabled. Table lists the valid commands. It was found that data rates of higher than about 50 Hz crashed the program, which is why 20 Hz was chosen as the maximum. Section 5.3.1 shows how some of these commands are sent via a terminal program.

Command	Description
`@START!\n`	Start the test
`@STOP!\n`	Stop the test
`@SUBXENA1!\n`	Enable a subject
`@SUBXENA0!\n`	Disable a subject
`@SUBXREL1!\n`	Apply power to a subject
`@SUBXREL0!\n`	Remove power from a subject
`@SUBXGPIO1!\n`	Write a subject's GPIO high
`@SUBXGPIO0!\n`	Write a subject's GPIO low
`@CLK001HZ!\n`	Change the data rate to 1 Hz
`@CLK002HZ!\n`	Change the data rate to 2 Hz
`@CLK005HZ!\n`	Change the data rate to 5 Hz
`@CLK010HZ!\n`	Change the data rate to 10 Hz
`@CLK020HZ!\n`	Change the data rate to 20 Hz

Table 4.1: List of commands that could be sent to the terminal.

[2]The `X` characters are placeholders for subject numbers (1, 2, 3, 4).

4.4 Raspberry Pi

This section explains the setup and programming of the Raspberry Pi introduced in Section 3.4. The full script can be accessed through the link in Appendix I, and the results can be seen in Section 5.4.

4.4.1 Hardware and Setup

The Raspberry Pi required additional hardware, which included a USB power brick of 5V and 2.5A [61], a 32 GB SD-card for the OS [62], a USB hub to increase the number of ports to five, and a straight, Cat5 cable for the PC [3]. As the literature of Section 2.1.3.1 showed, this would allow a data rate of up to 100 Mbps. The cable is sufficient, because the length of all four subject lines is about 220 bytes and the maximum data rate is 20 Hz (Section 4.3.8), so this would equate to a mere 4.4 kbps. Since most Ethernet cables support a maximum length of 100 metres [63], the only available cable, which was one meter long, sufficed for a demonstration. The Raspberry Pi Imager was used to install an OS named "Raspberry Pi OS Full (32-bit)", formerly called Raspbian [64]. The pre-installed Thonny editor was used for the Python script. The Raspberry Pi was controlled through a Virtual Network Computing (VNC) server called RealVNC, which is also pre-installed [64]. This allowed the Pi to be managed from a PC using one cable instead of a keyboard, mouse, and monitor. The Ethernet cable could be used both for the VNC server and for data exchange between the PC and STM32s, although disconnecting the server would allow for higher data rates. Appendix P explains the RealVNC setup.

4.4.2 Programming the STM32s

Since the literature of Section 2.1.1 demonstrated that the VirtualHere software could extend USB devices over either an Ethernet cable or Wi-Fi, the Ethernet approach was taken again for its higher speed. Although the free version of the software allows only one USB port at a time, this is a minor issue since programming is not done simultaneously. A VirtualHere server was downloaded from a browser on the Raspberry Pi. A complementary VirtualHere client was installed on the server, and the ports were renamed from within the software to easily identify them. After programming, the data stream of Section 4.4.3.1 can be started again. Appendix E.4.1 shows how the server is started from the terminal [11] and how the client software successfully shows all five MCUs being recognised.

[3] As opposed to a crossover cable.

4.4.3 Raspberry Pi Script

Appendix E.4.2 shows a flow diagram of the server script which ran on the Raspberry Pi.

4.4.3.1 Data exchange over Ethernet

A UDP socket was set up for data exchange between the STM32s and the Raspberry Pi to encode and decode messages. UDP was chosen over TCP because it is connectionless [4], faster, and more tolerant of packet loss [65], which is crucial given the high data speed. Appendix E.4.3 details how to create the socket for Ethernet communication.

4.4.3.2 Identifying changing USB ports

When the STM32s are plugged into the Pi, they are each assigned a device name (e.g., `ttyACM0`, `ttyACM1`), similar to a COM port. The command `/dev/tty*` can be used to see the names of the plugged in devices. To read data from one of these ports, the port is first opened using a device name, baud rate, and timeout as parameters, and the buffer is flushed to wait for possible outgoing serial data to complete [66]. The port can then be read line-by-line, after which each line is decoded and finally any white space is stripped [67].

```
1 ser = serial.Serial("/dev/ttyACM0", '230400', timeout=1).flush()
2 line = ser.readline().decode('utf-8').rstrip()
```

The issue with multiple STM32s was their unpredictable device names, which change when reconnected. Only the unique serial numbers of the ST-LINKs distinguished them. A function retrieved the device name and the last six digits of the serial number for each ST-LINK, identified by a unique product ID of '374b', as Appendix Q.2.1.1 shows.

4.4.3.3 Test station and subjects reporting to the PC

Knowing which serial port corresponded to which STM32 device, and having decoded and stripped an incoming line from the test station that contained a subject's readings, this line could simply be encoded using `encode` and sent to the PC using `sendto`. The script was designed such that the subjects could report their serial port data directly to the PC in the same way as above, where a `for` loop would be used to read a line for every device, and the data would also be sent to the PC.

4.4.3.4 Reading and processing PC commands

As alluded to in the system diagram, the script would only check for certain commands from the PC in certain states, such as checking for a test start command only when the client was connected. Commands that were more understandable would be sent from the GUI and be translated to those in Table 4.1 (see Section 4.5.2).

[4] The server can send data without the client having to be connected.

4.5 Graphical User Interface

TKinter was chosen for the GUI, since the literature of Section 2.1.4 illustrated that although the library would result in basic-looking user interface, a GUI with similar requirements than Section 3.7 could be written without too much effort. Before writing the software, a GUI layout was done to gauge a feel for the program, which can be seen in Appendix L. This helped considerably with the development process. Appendix E.5.1 includes a flow diagram that explains the gist of the GUI software. The full GUI code can be accessed through the link in Appendix I, and Section 5.5 explains where screenshots of the final GUI can be seen.

4.5.0.1 Modular code organisation

To display a widget like a button on a TKinter GUI, it has to be created and then added to a frame. Multiple frames can be used to arrange the widgets on the GUI as desired. To keep the individual files clear and to enhance modularity, separate files were used to house classes for the largest frames. These included a frame for each test subject, a settings frame, and a run frame from which the test could be started. A main file was used to create objects for these classes, such as four separate test subject frame objects.

4.5.0.2 Connecting to the Raspberry Pi server

A button was added to connect to the Raspberry Pi, and if not clicked, all other widgets would be disabled. The user also first had to disconnect before closing the window. These practices made the GUI more robust. To connect to the Pi, a socket had to be created as in Section 4.4.3.1. A timeout of three seconds was included in case the server was down. A message would then be sent to the server to connect to it. If successful, the client would start to receive data using the `recvfrom` method. This would then be decoded, and the readings would be extracted.

4.5.1 Extracting Readings from the Raspberry Pi

As mentioned in Section 4.3.8.2, the test station sent each subject's data line-by-line, with a start character before- and an end character after the subjects' lines. This allowed the GUI to place the data of four subjects in a buffer, from which a function could extract the individual measurements based on their unique keywords such as `V@`, which preceded the voltage. The variables of each test subject frame object were then updated accordingly, and the label widgets would be updated to show the latest readings.

4.5.2 Commands to the Test Station

To send the required commands of Section 4.3.8.3 to the test station, a function named `send_message_to_server` was called (full definition in Appendix Q.2.2.1). The widgets that sent commands and ultimately executed this function (besides the connect button) included a drop-down menu for the clock frequency, a button to start or stop the test, a button to enable or disable each subject, a button to apply or remove power to or from each subject, and a button to write to a subject's GPIO pin. As mentioned in Section 4.4.3.4, commands like `s1_rel_on` were translated to ones like `@S1REL1!\n`.

4.5.2.1 CSV storage

To save the data of a test, it was stored in a CSV file that can be opened in Excel. A heading row was first printed, and after that four rows for the four subjects at every time instance, with the time instance included at the end of the line as a formatted string, using the `datetime` package. The time each file was saved was appended to the end of the file name to keep them unique and identifiable. The CSV file would not be updated only at the end of a test, but at every time instance, to keep the data safe in case an unexpected error occurred. The CSV storage results can be seen in Appendix M.5.0.6.

4.5.2.2 Graphs

To meet the specification of displaying the live graphs of at least two parameters for all subjects (Section 3.7), `matplotlib` was used. An array of measurements was updated at every update instance, which happened at the set data rate. The graphs would then be redrawn to show all the data of the new array. It was decided to plot voltage and current rather than clock frequency. A button to plot each was included on the GUI instead of displaying voltage and current on the same graph, since this would give a better resolution for both measurements, and sometimes the user would only want to see one of the parameters. These buttons would only be enabled if a test was running, and if a graph window was not open already, which would make the system less error-prone. The graphs were made to appear in floating windows separate from the GUI, since this would allow the user to resize them according to their preferences. Appendix M.5.0.7 includes screenshots of these graphs in the GUI.

4.5.2.3 Direct subject reporting

The intent was to include for each subject a miniature terminal window consisting of a scrollable box where its serial data could be printed, similar to a terminal program. Appendix M.5.0.7 shows the widget included in the GUI, but the functionality was unfortunately not implemented due to reasons mentioned in Section 5.4.

Chapter 5

Results

To ensure a smooth testing process, the test station hardware and complementary software were tested in sections, rather than all at once. Therefore, this chapter starts with an extensive test of the modules before moving on to a test of the main board, the Raspberry Pi, and finally the GUI. Detailed results are in appendices.

5.1 Modules Functionality

5.1.1 Relay Driver Functionality

Before testing the configurable power supply, the relay driver designed in Section 4.1.2 first had to be operational. To test this, two continuity tests and one output voltage test were performed. For each test, a different relay supply voltage was used to cover all scenarios. Appendix M.1.1 illustrates that the relay driver worked as designed.

5.1.2 Current- and voltage sensor

After getting a breakout board version of the INA219 to work using a breadboard prototype of the test station (see Appendix K), the module's I2C lines could simply be connected to the same I2C lines on this prototype to test it. It proved to give the same readings as the breakout boards, and the readings were printed to a terminal program (PuTTY). The voltage readings were calibrated using linear regression in Excel, but not the current readings, since no accurate current measurement tool was available. However, since the voltages were already very close to being perfectly calibrated, it was assumed that the currents would also be. The 5% specification for voltage and current (see Section 3.7) were thus met. The I2C signals were also investigated on an oscilloscope, and the signals illustrate that the pull-up resistors of Section E.1.2.2 were appropriate enough for sensible signals. Appendix M.1.2 gives the detailed results for the sensor.

5.1.3 Configurable Power Supply

Having confirmed that the relay driver and sensor worked, the configurable power supply of Section 4.1.1 could be tested for no-load and full-load conditions. For the 3.3V regulator, the no-load and full-load voltage specification of 3-3.6V was met, and the current specification

of "being able to power a subject" (100 mA) was comfortably met with at least 390 mA. For the 5V regulator, the voltage specification of 4.75-5.25V at both no-load and full-load was met across the range of 7.5V to 12V, since 7V was too low for the dropout voltage. The full-load current specification was also comfortably met for both 7.5V and 12V, at 500 mA and 200 mA, respectively. All LEDs also came on at the right times. Lastly, when the USB cable of a subject was plugged in, it measured about 45 mA for any supply voltage, while this was doubled when the USB cable was plugged out for all voltages except 3.3V, where it stayed at about 45 mA. The currents never exceeded 100 mA. Appendix M.1.3 explains the details.

5.2 Main Board Functionality

5.2.1 Power Supply

To test the main board's power supply, the voltage selection was started with. The appropriate LEDs came on as designed for both an external 5V and 7-12V. The regulator circuit was not tested exhaustively again, since it was identical to that of the modules. The reverse-polarity protection was tested next, which worked since a -9V supply did not damage the board. Finally, all four test subjects and their modules were powered simultaneously, with each subject having a different supply voltage. The total current draw was an acceptable 700 mA, and the MOSFET introduced a negligible 20 mV voltage drop, which allowed the 5V specification for the subjects to be met (should they be powered exactly 5V externally). Appendix M.2.1 illustrates the above results in detail.

5.2.2 ST-LINK Programming and Crystal Oscillators

Other features of the main board that required testing was the ST-LINK programming capability and the oscillators. The main board could be programmed successfully, and the HSE oscillator of Section 4.2.2.2 had a much higher accuracy than the internal oscillator (1.000 MHz compared to 0.9976 MHz). Appendix M.2.2 goes into more details.

5.3 Embedded Software Functionality

5.3.1 Readings and Commands via a Terminal Program

To test the embedded software, it was first necessary to verify that the communication between the test station and terminal program worked as designed in Section 4.3.8. Appendix M.3.1 verifies this, as readings are being logged in the condensed format and the test station's logging frequency is changed by a command.

5.3.2 GPIOs

The GPIO writing and reading were tested together. The test subject alternately reported its GPIO state as 1 or 2, and when the test station wrote to its GPIO pin, it reported state 3, as designed in Section 4.3.3. Appendix M.3.2 gives more information.

5.3.3 Clock frequency

The clock frequency waveform quality was found to be acceptable up until about 9 MHz, which is why the subjects outputted their clocks at this frequency. The frequency measurement specification of 1% was comfortably met for all the subjects. Appendix M.3.3 gives more details.

5.3.4 UART

To test the UART functionality, the AND gates first had to work, which was verified after modifying the board with external pull-up resistors. The signal quality before and after the AND gates were then found to be excellent. Next, it was verified that a terminal program could be used to send a message to the subject, which replied with the expected message. Finally, the test station replaced the terminal program, and the GUI verified that the UART verification worked, but only for one subject at a time, and the cause for this could not be found. Appendix M.3.4 explains the above results in detail.

5.3.5 I2C

The I2C functionality designed in Section 4.3.5 was verified by firstly investigating the signal quality with an oscilloscope, which proved to be acceptable. After that, a number was sent from the test station to subject 1, which responded with the square of the number. Since the test station expected this, it would report the I2C status as operational. Appendix M.3.5 goes into more detail.

5.4 Raspberry Pi

The Raspberry Pi was able to recognise all the serial ports of the MCUs according to their serial numbers, as described in Section 4.4.3.2. It was able to receive commands from the GUI such as `s3_rel_off`, which it translated to the correct message for the test station (`@SUB3REL0!\n`). It was also able to read the lines from the serial port of the test station containing the readings of all four subjects, and also the serial port of a subject itself. However, directly reporting the subjects' data significantly slowed down the GUI

and made it almost unusable, which is why it was avoided. It was demonstrated that an MCU could be programmed via VirtualHere. Appendix M.4 includes more information.

5.5 Graphical User Interface

As alluded to in Section 4.5, Appendix M.5 includes screenshots showing how the GUI is used to connected and disconnect to the server, to enable subjects, to start a test, to power subjects, and to change the data rate. Screenshots of warning messages and CSV file storage functionality are also shown. All the GUI specifications of Section 3.7 were met besides the data sent directly from the subjects to the GUI.

5.6 Full System Setup

Figure 5.1 shows the full system setup. The test subjects were placed on a wooden base with a slider to move them slightly further from or closer to the test station if desired. The subjects were arranged such that the MCUs were close to each other and the ST-LINKs far from the radiation beam's centre. The USB cables were kept together with small pieces of wire. The figure shows the Ethernet cable going to the PC, which was used for programming, data exchange, and a remote desktop connection. The Raspberry Pi's cable coming from the power brick and the bench power supply's cables are also shown.

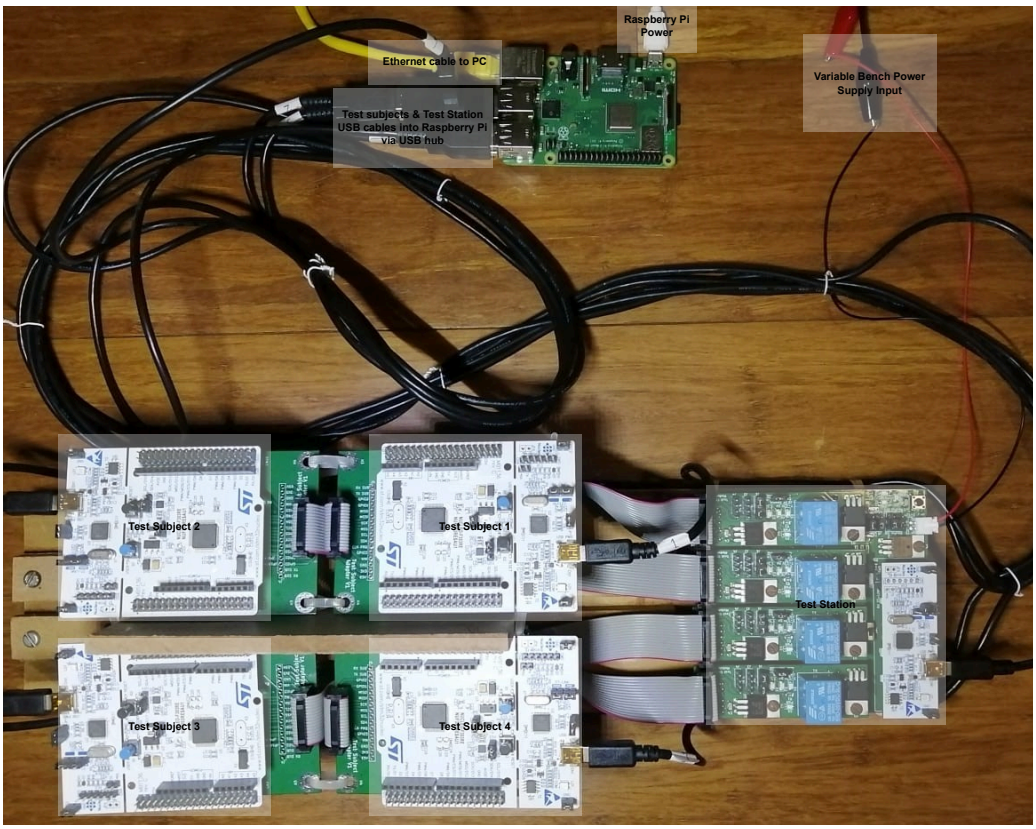


Figure 5.1: A setup of the full system (excluding PC and power supplies).

Chapter 6

Conclusion

6.1 Overview

Recall from Section 1.2 that the two goals of the project were to design and build a test station to monitor the behaviour of multiple microcontroller boards during radiation tests, and to design and implement a system such that a user located a distance away could control and monitor the test station via a graphical user interface. The system was successfully designed and built, and would enable a test operator to perform radiation tests more quickly and safely while providing more holistic monitoring of the exposed microcontrollers' behavior. The only outstanding requirements were the SPI software and direct subject reporting. Nevertheless, the test station was still usable and much more convenient, modular, and robust than the first version.

6.2 Objectives and Metrics

All three objectives defined in Section 1.3 were met. These were the design and implementation of multiple identical sets of test subject hardware and the embedded test subject software, a complementary test station base board with the test station's embedded software, and a long-distance wired communication scheme between the PC and test station and/or subjects, including the GUI.

6.3 Specification Compliance

Almost all specifications of Section 3.7 were met. The required PCBs were designed and built, and they all worked as expected. A user could power less than four subjects/modules during a test. The module met the 5% accuracy specification of voltage and current, and the subjects could be supplied by the three specified voltages, all within the required ranges except the minimum voltage being 7.5V instead of 7V. The 5V's source worked with both the regulator and the external supply. UART and I2C transmission and reception could be verified for all subjects, with the UART only working for one subject at a time. The SPI software was not implemented due to time constraints. The specification of GPIO reading and writing was met, and also the clock frequency specification of 1%, which was ensured by the accurate HSE oscillator that worked as designed. The logging frequency

was adjustable from the GUI. Programming of all MCUs could be done over Ethernet. The test station could report all the subject readings to the PC without any bottlenecks, although the subjects could not do so directly without compromising the GUI response time. CSV storage was implemented as specified, and although the soft requirement of graphs with a history function was not met, graphs for each subject could successfully display its voltage and current. The elapsed time could also be displayed during a test, as per the specification. Although neither a requirement nor in the project scope, it has to be mentioned that tests could only be ran for a maximum of 1m45s using four subjects, and this time decreased at higher data rates, such as 15 seconds for four subjects at 10 Hz. This is most likely due to an array being indexed outside of its size, leading to memory corruption. The problem is definitely in the test station's embedded software, since it still occurs when directly interacting with the test station via a terminal program as opposed to through the GUI. The issue is considered a minor one for the purpose of this project and can most likely be solved with a single line of code. The independent terminal program control is another specification that was met.

It is worth mentioning that an unanticipated network problem arose during the project, which took at least two full days to work around. This problem prevented any code to be uploaded to GitHub and also prevented the installation of Python libraries using `pip`, which were both crucial for the project. Had the error not occurred, there would probably have been enough time to implement the SPI.

6.4 Future Work

This section gives recommendations should a similar project be done. The 10 kΩ pull-up resistors for the UART AND gates on the main board should be included in the PCB design, as well as minor silkscreen mistakes such as E7-12V and E5V being swapped on the main board. Changing the supply voltage on the modules with jumpers would be easier if these were further apart. Making all the LEDs the same brightness as opposed to having the debug- and relay LEDs at half the brightness of the others would look better. Using thicker cables for the power connector would make the system more robust. Putting all the solder bridges of the connection PCB on the bottom side would not require removing the subjects for a modification. The broken-off ST-LINK could be replaced by a more aesthetic solution such as a soldered V3. Furthermore, a mechanism for remotely resetting the test station would be beneficial in case the Raspberry Pi would go out of sync with it. Having the Pi's script starting automatically as opposed to having to log into it via RealVNC would speed up the testing process. If in-application programming (Section 2.1.2.3) was used to program the test station and the subjects were replaced by Nucleo-144 boards with Ethernet ports, the Raspberry Pi could be replaced altogether by a simple network switch, which would eliminate various nuisances.

Bibliography

[1] Robotistan, "*ENC28J60 Ethernet LAN Modul* [Online]." Available: https://www.robotistan.com/enc28j60-ethernet-lan-modul.

[2] M. Zeng, "*CAT5 vs. CAT5e vs. CAT6. For More Bandwidth?* [Online]." Available: https://www.router-switch.com/faq/comparison-of-cat5-cat5e-and-cat6-cables.html, Dec. 2017.

[3] Texas Instruments, "*INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I2C Interface* [Online]." Available: https://www.ti.com/lit/ds/symlink/ina219.pdf, Aug. 2008 [Revised Dec. 2015].

[4] S. M. Guertin, "*Candidate Cubesat Processors* [Online]." Available: https://nepp.nasa.gov/workshops/eeesmallmissions/talks/11%20-%20THUR/1350%20-%20CubesatMicroprocessor_V1.pdf, Sept. 2014.

[5] K. A. LaBel, "*Radiation Effects on Electronics 101: Simple Concepts and New Challenges* [Online]." Available: https://nepp.nasa.gov/docuploads/392333B0-7A48-4A04-A3A72B0B1DD73343/Rad_Effects_101_WebEx.pdf, Apr. 2004.

[6] J. Harris, "*A Quick Overview of Radiation Effects* [Online]." Available: https://www.planetanalog.com/a-quick-overview-of-radiation-effects/, Dec. 2017.

[7] M. Campola, "*Single Event Effects* [Online]." Available: https://radhome.gsfc.nasa.gov/radhome/see.htm, [Revised Oct. 2021].

[8] J. Harris, "*A Quick Overview of Radiation Effects – Single Event Effects* [Online]." Available: https://www.planetanalog.com/a-quick-overview-of-radiation-effects-single-event-effects/, Jan. 2018.

[9] Crocker Nuclear Laboratory, "*RADIATION'S EFFECTS ON AEROSPACE ELECTRONICS* [Online]." Available: https://cyclotron.crocker.ucdavis.edu/radiations-effects/, [Revised May 2023].

[10] European Space Agency, "*Radiation testing for space* [Online]." Available: https://www.youtube.com/watch?v=xySwrn32GTY, Feb. 2015.

[11] DroneBot Workshop, "*Extend your USB with VirtualHere* [Online]." Available: https://dronebotworkshop.com/pi-10-virtualhere/, Feb. 2022.

[12] ST Microelectronics, "*ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32* [Online]." Available: https://www.st.com/resource/en/user_manual/um1075-stlinkv2-incircuit-debuggerprogrammer-for-stm8-and-stm32-stmicroelectronic pdf, Apr. 2011 [Revised Apr. 2024].

[13] ST Microelectronics, "*STLINK-V3MINIE debugger/programmer tiny probe for STM32 microcontrollers* [Online]." Available: https://www.st.com/resource/en/data_brief/stlink-v3minie.pdf, Sept. 2021 [Revised Dec. 2021].

[14] ST Microelectronics, "*ST-LINK* [Online]." Available: https://wiki.st.com/stm32mpu/wiki/ST-LINK, Oct. 2022.

[15] ST Microelectronics, "*USB DFU protocol used in the STM32 bootloader* [Online]." Available: https://www.st.com/resource/en/application_note/an3156-usb-dfu-protocol-used-in-the-stm32-bootloader-stmicroelectronics.pdf, Mar. 2020 [Revised Feb. 2024].

[16] P. Salmony, "*STM32 Programming via USB (DFU) - Phil's Lab 72* [Online]." Available: https://www.youtube.com/watch?v=VlCYI2U-qyM, Aug. 2022.

[17] J. Lee, "*STM32 TFTP IAP* [Online]." Available: https://www.youtube.com/watch?v=2qlK4djgGdM, May 2022.

[18] ST Microelectronics, "*STM32F407/STM32F417 in-application programming (IAP) over Ethernet* [Online]." Available: https://www.st.com/resource/en/application_note/an3968-stm32f407stm32f417-inapplication-programming-iap-over-ethernet-stmicroelec pdf, Oct. 2022.

[19] ST Microelectronics, "*STM32 Nucleo-144 boards* [Online]." Available: https://www.st.com/content/ccc/resource/technical/document/data_brief/group1/8c/c9/8a/9b/3e/f0/45/bb/DM00357737/files/DM00357737.pdf/jcr:content/translations/en.DM00357737.pdf, Feb. 2017 [Revised Jan. 2024].

[20] My Mate VINCE, "*How to install/extend a USB cable over CAT5e/CAT6 network wiring* [Online]." Available: https://www.youtube.com/watch?v=CtyCiZ2LEWE, June 2015.

[21] AV Access, "*4KEX100-KVM* [Online]." Available: https://www.avaccess.com/products/4kex100-kvm/.

[22] J-Tech Digital, "*USB 2.0 Extender (100M)* [Online]." Available: https://jtechdigital.com/wp-content/uploads/woocommerce_uploads/product/jtd-406/JTD-617.pdf.

[23] Cable Matters, "*USB Over Ethernet Extender with 2X USB Ports up to 165ft* [Online]." Available: https://www.amazon.com/Cable-Matters-Extender-Over-Ethernet/dp/B0CFGBVST6?ref_=ast_sto_dp.

[24] WeeW - Stack, "*000-Python Live stream Graphic User Interface Arduino-STM32: Tutorial Introduction* [Online]." Available: https://www.youtube.com/watch?v=jdCMOOBDots, July 2021.

[25] TurbineThree, "*Best Python GUI Libraries Compared! (PyQt, Kivy, Tkinter, PySimpleGUI, WxPython PySide)* [Online]." Available: https://www.youtube.com/watch?v=Q72b6tDQMKQ, Apr. 2023.

[26] Y. Mohammed, "*What I Learned About Heatsinks Using Thermal Simulation* [Online]." Available: https://semiengineering.com/what-i-learned-about-heatsinks-using-thermal-simulation/, May 2016.

[27] NASA, "*Heat Sinks for Parts Operated in Vacuum* [Online]." Available: https://llis.nasa.gov/lesson/809, Jan. 1999.

[28] B. Niehoff, "*What is Outgassing and How is it Tested?* [Online]." Available: https://blog.samtec.com/post/what-is-outgassing-and-how-is-it-tested/, Apr. 2020.

[29] Cadence PCB solutions, "*Challenges for Electronic Circuits in Space Applications* [Online]." Available: https://resources.pcb.cadence.com/blog/2023-challenges-for-electronic-circuits-in-space-applications, 2023.

[30] epoxyset, "*EB-135 Epoxy: Meeting NASA's Rigorous Outgassing Standards* [Online]." Available: https://epoxysetinc.com/uncategorized/nasa-outgassing-requirements/, Feb. 2017.

[31] Douglas Electrical, "*VACUUM ELECTRICAL FEEDTHROUGHS* [Online]." Available: https://douglaselectrical.com/douglas-applications/vacuum/.

[32] ST Microelectronics, "*UM1724 User manual* [Online]." Available: https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf, Feb. 2014 [Revised Aug. 2020].

[33] ST Microelectronics, "*STM32F303xD STM32F303xE* [Online]." Available: https://www.st.com/resource/en/datasheet/stm32f303re.pdf, Jan. 2015 [Revised Oct. 2016].

[34] ST Microelectronics, "*STLINK-V3MODS mini debugger and programmer for STM32 microcontrollers* [Online]." Available: https://www.st.com/resource/en/data_brief/stlink-v3mods.pdf, Dec. 2018 [Revised Dec. 2023].

[35] Linus Tech Tips, "*I Can Save You Money! – Raspberry Pi Alternatives* [Online]." Available: https://www.youtube.com/watch?v=uJvCVw1yONQ, Feb. 2023.

[36] ST Microelectronics, "*Positive voltage regulator ICs* [Online]." Available: https://www.st.com/resource/en/datasheet/l78.pdf, [Revised Sep. 2018].

[37] ST Microelectronics, "*Adjustable and fixed low drop positive voltage regulator* [Online]." Available: https://www.st.com/resource/en/datasheet/ld1117.pdf, [Revised Feb. 2020].

[38] EEPower, "*Tantalum Capacitor* [Online]." Available: https://eepower.com/capacitor-guide/types/tantalum-capacitor/.

[39] T. Reyes, "*Selecting ESR for Linear Regulator Stability* [Online]." Available: https://www.ti.com/lit/ta/ssztbj1/ssztbj1.pdf, Mar. 2016.

[40] Lighthouse LEDs, "*1206 High Bright Orange LED* [Online]." Available: https://lighthouseleds.com/media/wysiwyg/documentation/1206%20Orange%20Amber%20SMD%20LED%20Datasheet.pdf.

[41] Songle Relay, "*SUBMINATURE HIGH POWER RELAY* [Online]." Available: http://www.songlerelay.com/Public/Uploads/20161104/581c81ac16e36.pdf, Nov. 2016.

[42] Advanced Monolithic Systems, "*1A LOW DROPOUT VOLTAGE REGULATOR* [Online]." Available: http://www.advanced-monolithic.com/pdf/ds1117.pdf.

[43] T. Hegarty, "*Paralleling Electrolytic and Ceramic Capacitors Perks Up POL Transient Response* [Online]." Available: https://www.how2power.com/newsletters/1103/articles/H2PToday1103_design_National.pdf?NOREDIR=1, Mar. 2011.

[44] Afrotechmods, "*How to protect circuits from reversed voltage polarity!* [Online]." Available: https://www.youtube.com/watch?v=IrB-FPcv1Dc, Dec. 2011.

[45] International Rectifier, "*IRF4905PbF* [Online]." Available: https://www.infineon.com/dgdl/irf4905pbf.pdf?fileId=5546d462533600a4015355e329b1197e, Mar. 2011.

[46] Vishay General Semiconductor, "*1N4001, 1N4002, 1N4003, 1N4004, 1N4005, 1N4006, 1N4007* [Online]." Available: https://www.vishay.com/docs/88503/1n4001.pdf, [Revised Apr. 2020].

[47] Vishay General Semiconductor, "*1N5817, 1N5818, 1N5819* [Online]." Available: https://www.vishay.com/docs/88525/1n5817.pdf, [Revised Jul. 2020].

[48] ST Microelectronics, "*Getting started with STM32F3 series hardware development* [Online]." Available: https://www.st.com/resource/en/application_note/an4206-getting-started-with-stm32f3-series-hardware-development-stmicroelectronic pdf, Dec. 2012 [Revised Feb. 2015].

[49] ST Microelectronics, "*Getting started with STM32F4xxxx MCU hardware development* [Online]." Available: https://www.st.com/resource/en/application_note/an4488-getting-started-with-stm32f4xxxx-mcu-hardware-development-stmicroelectroni pdf, June 2014 [Revised Oct. 2018].

[50] ST Microelectronics, "*MB1136* [Online]." Available: https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group2/5a/85/d6/9a/34/e2/47/1d/MB1136-DEFAULT-C05_Schematic/files/MB1136-DEFAULT-C05_Schematic.pdf/jcr:content/translations/en.MB1136-DEFAULT-C05_Schematic.pdf, [Revised Aug. 2022].

[51] B. Potvin, "*Do you know when to use a crystal or an oscillator? The wrong answer can cost you.* [Online]." Available: https://www.sitime.com/company/news/blog/do-you-know-when-use-crystal-or-oscillator-wrong-answer-can-cost-you, Feb. 2023.

[52] ST Microelectronics, "*Guidelines for oscillator design on STM8AF/AL/S and STM32 MCUs/MPUs* [Online]." Available: https://www.st.com/resource/en/application_note/an2867-guidelines-for-oscillator-design-on-stm8afals-and-stm32-mcusmpus-stmicroel pdf, Jan. 2009 [Revised May 2014].

[53] Abracon, "*32.768 kHz LOW FREQUENCY SMD CRYSTAL* [Online]." Available: https://abracon.com/Resonators/abs25.pdf, [Revised Sep. 2019].

[54] ST Microelectronics, "*RM0316 Reference manual* [Online]." Available: https://www.st.com/resource/en/reference_manual/rm0316-stm32f303xbcde-stm32f303x68-stm32f328x8-stm32f358xc-stm32f398xe-advanced-a pdf, Dec. 2013 [Revised Jan. 2024].

[55] P. Dhaker, "*Introduction to SPI Interface* [Online]." Available: https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html, Sept. 2018.

[56] P. Salmony, "*KiCad 6 STM32 PCB Design Full Tutorial - Phil's Lab 65* [Online]." Available: https://www.youtube.com/watch?v=aVUqaB0IMh4, July 2022.

[57] P. Smolen, "*PSM_INA219_STM32* [Online]." Available: https://github.com/komuch/PSM_INA219_STM32/tree/main, Dec. 2020 [Revised Apr. 2021].

[58] Adafruit Industries, "*Adafruit_INA219* [Online]." Available: https://github.com/adafruit/Adafruit_INA219, Apr. 2020 [Revised Nov. 2023].

[59] K. Magdy, "*STM32 Input Capture Frequency Measurement Example — Timer Input Capture Mode* [Online]." Available: https://deepbluembedded.com/stm32-input-capture-frequency-measurement-example-timer-input-capture-mode/, 2020.

[60] N. Ruttley, "*The definitive guide on writing a SPI communications protocol for STM32* [Online]." Available: https://medium.com/embedism/the-definitive-guide-on-writing-a-spi-communications-protocol-for-stm32-73594add4 Jan. 2021.

[61] Raspberry Pi, "*Raspberry Pi 3 Model B+* [Online]." Available: https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf, [Revised Nov. 2023].

[62] Raspberry Pi, "*Getting started with your Raspberry Pi* [Online]." Available: https://www.raspberrypi.com/documentation/computers/getting-started.html, Dec. 2022.

[63] G. Shukla, "*How Long Can an Ethernet Cable Be?* [Online]." Available: https://www.howtogeek.com/813419/how-long-can-an-ethernet-cable-be/, July 2022.

[64] Tegan, "*RealVNC Connect and Raspberry Pi* [Online]." Available: https://help.realvnc.com/hc/en-us/articles/360002249917-RealVNC-Connect-and-Raspberry-Pi, [Revised Feb. 2024].

[65] B. Gorman, "*TCP vs UDP: What's the Difference and Which Protocol Is Better?* [Online]." Available: https://www.avast.com/c-tcp-vs-udp-difference, Feb. 2023.

[66] J. Lewis, "*When do you use the Arduino's Serial.flush()?* [Online]." Available: https://www.baldengineer.com/when-do-you-use-the-arduinos-to-use-serial-flush.html, Jan. 2014.

[67] Eli the Computer Guy, "*Arduino - Read Serial Communication with Raspberry Pi* [Online]." Available: https://www.youtube.com/watch?v=3QSsnnbJYFc, Dec. 2020.

[68] ST Microelectronics, "*STM32F411xC STM32F411xE* [Online]." Available: https://www.st.com/resource/en/datasheet/stm32f411ce.pdf, June 2014 [Revised Jan. 2024].

[69] ST Microelectronics, "*STM32F446xC/E* [Online]." Available: https://www.st.com/resource/en/datasheet/stm32f446re.pdf, Feb. 2015 [Revised Jan. 2021].

[70] Onsemi, "*General Purpose Transistors* [Online]." Available: https://www.onsemi.com/pdf/datasheet/mmbt2222lt1-d.pdf, [Revised Aug. 2021].

[71] Rectron Semiconductor, "*SMALL SIGNAL DIODE* [Online]." Available: https://www.mouser.co.za/Search/DownloadDatasheet?url=https%3A%2F%2Fwww.mouser.co.za%2Fdatasheet%2F2%2F345%2F1n4148w-30781.pdf, Mar. 2006.

[72] R. Arora, "*I2C Bus Pullup Resistor Calculation* [Online]." Available: https://www.ti.com/lit/an/slva689/slva689.pdf, Feb. 2015.

[73] Abracon, "*HC/49US (AT49) SMD LOW PROFILE CRYSTAL* [Online]." Available: https://abracon.com/Resonators/ABLS.pdf, [Revised Feb. 2021].

[74] ControllersTech, "*STM32 I2C SLAVE* [Online]." Available: https://controllerstech.com/stm32-i2c-slave/, 2017.

[75] P. McWhorter, "*Raspberry Pi LESSON 41: How to Send Data to the PC over WiFi or Ethernet Using UDP* [Online]." Available: https://www.youtube.com/watch?v=S7Yle8clJ30, Dec. 2022.

[76] Python, "*socket — Low-level networking interface* [Online]." Available: https://docs.python.org/3/library/socket.html.

[77] vcubingx, "*How to connect to your Raspberry Pi using Ethernet! (Secure Shell[SSH] and Remote Desktop)* [Online]." Available: https://www.youtube.com/watch?v=oM2zVD9rL8I, July 2017.

Appendix A

Project Planning Schedule

Figure A.1 shows a Gantt chart that was made in Excel at the end of the second week of the project, at which point there was enough context for such a task. Although it was not followed perfectly, it helped a lot to keep different timestamps in perspective.

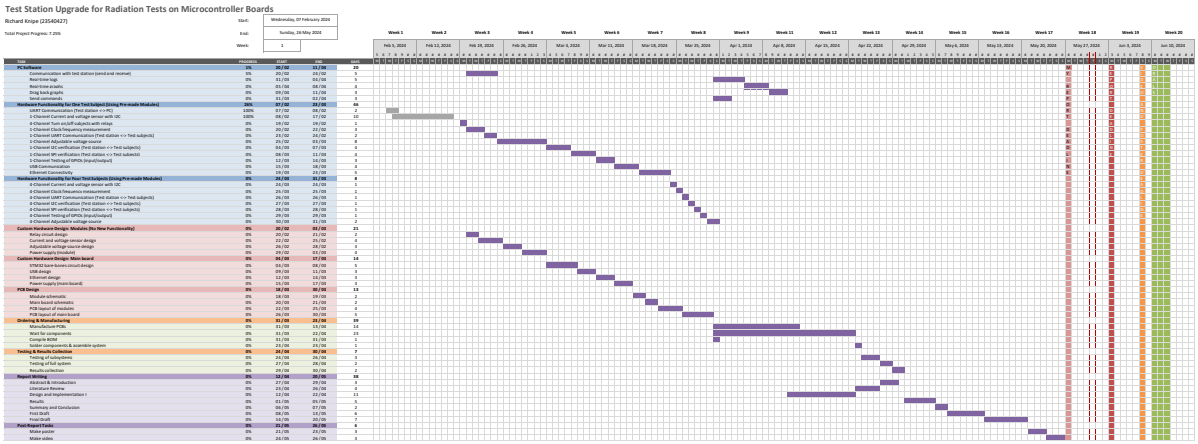


Figure A.1: A Gantt chart for the project, compiled at the end of the second week.

Some major events during the project had the following dates:

- 3 April: Ordered components
- 3-18 April: PCB design
- 19 April: Ordered PCBs
- 29 April - 3 May: Soldering and piece-wise testing
- GUI design: 13 May
- Handed in first draft: 27 May

Appendix B

Outcomes Compliance

This appendix explains how each ECSA Exit Level Outcome (ELO) was achieved through the project's execution, and section references are included.

B.1 ELO 1 - Problem Solving

Identify, formulate, analyse and solve complex engineering problems creatively and innovatively.

Section 1.3.1 lists the vague requirements that were given and for which specifications were formulated in Section 3.7. An example of an infrequently encountered issue is the vacuum environment, for which special considerations had to be made, as Section 2.2 illustrated. Similar solutions to the specialised problem addressed in the project is not widely available, requiring creative thinking. This was demonstrated in the design of the voltage selection jumper of Section 4.1.1.6, the PCB placement and design of Sections 3.6, 4.1.6, and 4.2.6.2, the GUI design of Section 4.5, and the UART receive algorithm mentioned in Section 4.3.6, just to name a few. The problem was broken down into a few sub-problems that could be solved separately, as was done in Section 1.3. Appendices K (test station prototype) and L (GUI planning) show how simpler versions of a problem was first tacked before attempting a full solution. Examples of solutions not commonly implemented are the one-to-many UART circuit of Section 4.2.4 for which the MCU's reference manual was consulted, and the snapped-off ST-LINK approach, which required consultation of the Nucleo schematics (Section 4.2.3).

B.2 ELO 2 - Application of Scientific and Engineering Knowledge

Apply knowledge of mathematics, natural sciences, engineering fundamentals and an engineering speciality to solve complex engineering problems.

Mathematics and engineering sciences were used throughout the report. Some examples are the design of circuits like the relay driver in Section 4.1.2, the design of embedded software like the firmware of the sensor in Section 4.3.2, the calculations used in the clock frequency measurement of Section E.3.2.1, and the PCB design of Sections 4.1.6 and 4.2.6.

The natural science aspect can be seen in the explanation of radiation effects on electronics in Section 1.1, the considerations made around the vacuum environment (Section 2.2), and the basic electromagnetism considerations for kickback voltage (Section 4.1.2 and Appendix E.1.2.1).

B.3 ELO 3 - Engineering Design

Perform creative, procedural and non-procedural design and synthesis of components, systems, engineering works, products or processes.

As mentioned under ELO 1, the problem is a complex engineering problem because it is under-specified, consists of sub-problems, and involves uncommon challenges. The problem is a major electronic engineering design problem, since it involves the design and implementation of a research tool in the field of radiation effects on electronics in the context of space, as introduced in Section 1.1. Examples of creativity in design was demonstrated under ELO 1. Elements of procedural design can be seen throughout the project. In a broad sense, this includes starting with specifications and system design in Chapter 3, then moving on to detail design in Chapter 4, or starting with an embedded software system diagram (Section 4.3) before explaining the details. Procedural design was also manifested in specific areas such as PCB design, which started with component choices and the design of a schematic (as can be seen in Appendix G), followed by the layout and placement (Section 3.6), the routing, and finally the soldering to produce a final product, as in Section 4.1.6. Another example is the GUI design, where the front-end was first designed before the functionality was implemented (Appendix L). An example of non-procedural design is the Raspberry Pi's server script, which could not really be implemented in clear steps but rather had to be almost fully functional the moment it could be used for the first time, and it did not involve much planning beforehand (Section 4.4).

B.4 ELO 4 - Investigations, Experiments and Data Analysis

Demonstrate competence to design and conduct investigations and experiments.

The research done in Chapter 2 illustrates that a thorough investigation was made into existing solutions that address at least parts of the same problem. This literature directly influenced design choices in later chapters, such as the ST-LINK research of Section 2.1.2.1 affecting the choice in Section 3.3.2.1, or the literature on the GUI (Section 2.1.4) affecting the choice of TKinter in Section 4.5. Additionally, research was not merely done on a surface level, since uncommon solutions like IAP over Ethernet (Section 2.1.2.3) were found. The experiments of Chapter 5 were also done thoroughly, such as in Section 5.1, where experiments were performed according to specific setups (Appendix N) which were measured according to the metrics defined in Section 3.7, as opposed to merely stating

that a particular part of the system "worked". Overall, it is clear that the project had a good balance of investigation and experiments.

B.5 ELO 5 - Engineering Methods, Skills and Tools, Including Information Technology

Demonstrate competence to use appropriate engineering methods, skills and tools, including those based on information technology.

The software applicable to engineering that was used include KiCad for PCB design (Section 4.1.5.1), STM32CubeIDE to write the embedded software (Section 4.3), and Python to write the GUI (Sections 4.5) and the server script for the Raspberry Pi (Section 4.4). Furthermore, Git was used for version control and GitHub was used to make backups of all the software written (Appendix I). Microsoft Excel was used to make and periodically adjust a Gantt chart used for time management (Appendix A). Finally, Overleaf was used to write the report in LaTeX.

B.6 ELO 6 - Professional and Technical Communication

Demonstrate competence to communicate effectively, both orally and in writing, with engineering audiences and the community at large.

The report appears professional as it was written in LaTeX, and it has an overall formal tone. Effective communication is demonstrated not only with clear sentences, but through diagrams that guide the engineering audience, such as in Sections 3.1, 4.3, 4.3.1, 4.4.3, and 4.5. A poster was made that could explain to the general public why the problem was necessary, how it was solved, and what was learned from the solution, without going into unnecessary details.

B.7 ELO 8 - Individual Work

Demonstrate competence to work effectively as an individual.

The report in its entirety was written individually, which was also the case for the video and poster. Although some parts of the solution were inspired from the first test station, no copying of any kind was done, and no information besides what is given in Section 2.1.1 was used. The only other inspired work includes a handful of functions written or partially written by AI (such as in Appendix Q.1.2) or taken from online sources (such as in Appendix Q.1.3), but these were not used without understanding the gist of them, and they were all acknowledged. Even soldering was mostly done individually,

besides the smaller components mentioned in the acknowledgements. All other aspects of the project were done individually.

B.8 ELO 9 - Independent Learning Ability

Demonstrate competence to engage in independent learning through well-developed learning skills.

Independent learning was done throughout the project, and specific examples in the report where new skills were obtained individually are the GUI written in TKinter (Section 4.5), the design of the reverse-polarity protection using a unique but efficient solution (Section 4.2.1.3), the design of a crystal oscillator circuit (Section 4.2.2.2), the use of a Raspberry Pi (Section 4.4), the implementation of an Ethernet communication protocol (Section 4.4.3.1), and the inter-MCU serial communication software of Section 4.3. All content in the literature review of Chapter 2 was learned during the duration of the project. All deadlines of the project besides the hand-in date were self-formulated, with the supervisor only giving suggestions on the crucial deadlines, such as when to order components. All the tasks of the project (see the Gantt chart in Appendix A) and their estimated durations were individually formulated.

Appendix C

Detailed Project Specifications

The specifications of the project summarised in Section 3.7 are given below.

- General hardware:
  - All hardware besides the SBC must be on PCBs. This includes the main board PCB, a module PCB for each subject, and a connection PCB for each subject.
  - No repeated test subject-related hardware can be on the main board, and the user must be able to use any number of subjects and plug in any number of modules to perform a radiation test.
  - All connectors should be designed to plug in only one way, and reverse-polarity protection is required for the bench power supply.
  - Regulators must maintain junction temperatures below 80% of their rated values.
  - No electrolytic capacitors, sealed components, or thermal paste may be used.
- Module hardware:
  - The voltage sensor must be accurate to 5% over a range of 0-12V, and the current sensor accurate to 5% over a range of 0-300 mA (the maximum current of a subject [32]).
  - The 3.3V selection must be between 3V and 3.6V, the 5V selection between 4.75V and 5.25V, and the 7-12V selection must range from 7V to 12V.
  - Each module must have selection for its 5V coming from either its regulator or from an external 5V.
  - Each voltage selection must output enough current to power a test subject.
- Embedded software:
  - UART transmission and reception between station and subject must be verified for all subjects.
  - I2C transmission and reception between station and subject must be verified for all subjects.
  - SPI transmission and reception between station and subject must be verified for all subjects.

  - The station must have the option of writing to a GPIO and reading from a GPIO of all subjects.
  - The clock frequency must be accurate to within 1% of subject's system clock (72 MHz [33]).
  - The test station must be independent from the long-distance communication interface such that it can log data to the PC directly and be controlled by it directly.
  - The logging frequency must be changeable, and must be at least 1 Hz.
  - The software should still work even if only one subject is used and one module is plugged in.

- Long-distance communication scheme:
  - Each subject and the test station must be programmed over Ethernet.
  - The data exchange between the PC and the test station and/or subjects must happen over Ethernet.
  - In addition to the test station's reporting, the subjects must report some data to the PC directly.

- Graphical User Interface:
  - Subjects must be enabled and powered individually, and not all at the same time.
  - A test must be started and stopped, and a stopwatch should give an indication of the elapsed time.
  - There has to be an option to change the logging frequency.
  - There has to be an option of writing to a subject's GPIO pin, and this must be verified by the test station.
  - The readings of all four subjects must be displayed simultaneously, which are the GPIO readings, voltage and current readings, clock frequency readings, UART statuses, I2C statuses, and SPI statuses.
  - Optional storage must be implemented, where all readings of all subjects are saved to a CSV file.
  - Graphs must display at least two live parameters of each subject such as current and voltage.

Appendix D

System Design Additional Information

D.1 Microcontroller Choice

Table D.1 shows the three MCUs that were considered for the main board, as mentioned in Section 3.3.2.1. The reason for these three was that a Nucleo version of the 303RE was already available for prototyping, the 411RC was available for a good price at a nearby store, and the 446RE had the best processing power out of those which had a Nucleo version. The 411RC was not chosen because it did not have support for four external timer clock inputs (discussed in Section 4.3.4). It was decided to use the 303RE because of familiarity with it, but the pin assignment for the test station's MCU was nevertheless done such that a 446RE could be used, should the user find that the slower microcontroller was not fast enough. Section 4.2.2 explains the detailed design of the microcontroller's surrounding components.

Name	STM32F303RE [33]	STM32F411CE [68]	STM32F446RE [69]
Clock frequency	72 MHz	100 MHz	180 MHz
RAM	80 kB	128 kB	128 kB
Flash	512 kB	256 kB	512 kB

Table D.1: Specifications of the three candidates for the test station MCU.

D.2 PCB Placement

Figure D.1 shows the test station PCB layout, as mentioned in Section 3.6.

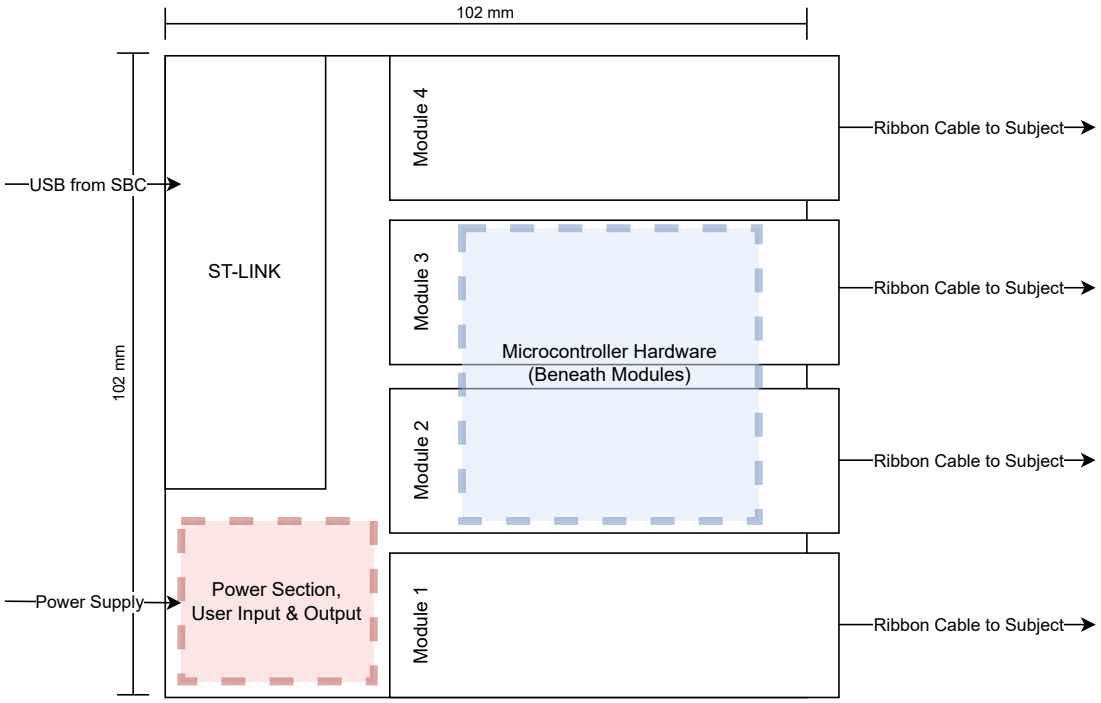


Figure D.1: Diagram showing a rough layout of the modules on the main board.

Appendix E

Detailed Design Additional Information

E.1 Test Station Modules

E.1.1 Relay Driver

This section includes the detailed calculations of the relay driver mentioned in Section 4.1.2, and it motivates component choices. The schematic is repeated in Figure E.1 for convenience.

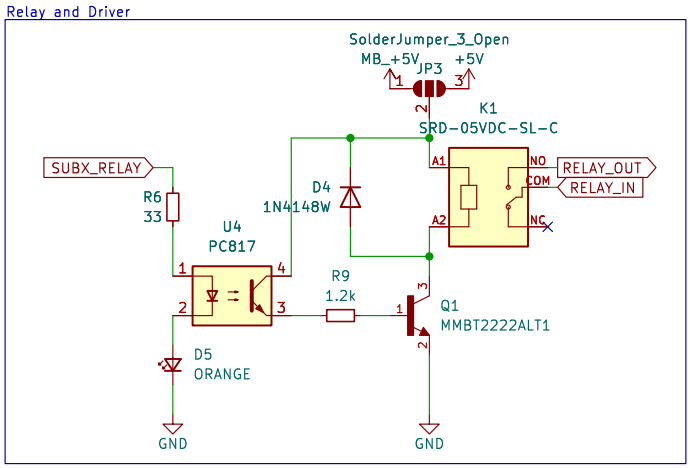


Figure E.1: Circuit schematic of the custom relay driver.

Although the common 2N2222A's maximum collector current of 600 mA and maximum collector-emitter voltage of 40 V [70] would make it an overkill for this scenario, it was chosen because it is widely available. The circuit was designed to drive the relay at a higher current than expected, to ensure that the BJT is "fully-on". A maximum collector current of 120 mA was chosen. Since the minimum gain for the chosen BJT is 35 [70], the base current could be calculated as 3.43 mA according to the formula $i_C = \beta i_B$.

The 1N4148 was chosen because it could handle a reverse voltage of 75 V and a non-repetitive forward surge current of 1 A for one second, which made it suitable for the task [71].

The status LED was identical to the power LEDs of Section 4.1.1.4 for which the desired brightness occurred at 10 mA, but it was decided to sacrifice a bit of brightness to stay within the limit of the GPIO pin. Therefore, 5 mA was chosen. Since the LED's datasheet states a *typical* forward voltage of 2.2V and a minimum of 1.8V at the testing

condition of 20 mA, this indicated that the forward voltage is not constant for a constant current. Therefore, the LED was not connected to the transistor's base such that it would not influence the base current. It was also not connected to the 5 V supply, since, although there is a flyback diode, the inductive kickback could still damage the LED. Instead, an optocoupler was used to alleviate the load of the GPIO pin, which would now only have to drive the optocoupler and LED, while the external 5 V would drive the relay and the transistor's base. This also provided galvanic isolation between the GPIO pin and the relay. The LED's datasheet states a forward voltage of 1.92 V at 5 mA [40]. At the same current, the forward voltage of the optocoupler's LED is 1.2 V. For a 3.3 V supply, this required a series resistance of 36 Ω. A 33 Ω resistor was chosen. Assuming $V_{BE(on)}$ of the BJT to be 0.7 V, taking the supply voltage to the base as 5 V, and $V_{CE(sat)}$ from the optocoupler's datasheet as 0.4 V, the required base resistance for a base current of 3.43 mA could be found as 1.137 kΩ. A 1.2 kΩ resistor was chosen.

Finally, the three-pad solder bridge was inserted to select between two different 5V sources, namely *MB_+5V* and *+5V* (see Section 4.1.1.5 for an explanation of these terms). Normally the user would want to use *MB_+5V* to separate the subject and relay supplies, but the option is just given to the user for more flexibility. A solder bridge was used instead of a header/jumper combination since PCB space was limited (see Section 3.6).

E.1.2 Current- and Voltage Sensor

This section gives more information on the design of the sensor in Section 4.1.3.

E.1.2.1 Short-circuit considerations

Since the sensor is in the high-side configuration, a short to ground on the load side would result in the full power supply voltage across the shunt, given that the power supply can keep its voltage stable during such a condition. When removing this short, an inductive kickback voltage exceeding the rated voltage across the shunt (26V) might occur [3]. Even though the load is a microcontroller and not mainly inductive (although this is hard to justify), dangerous kickback voltages are still a possibility because of the high current of a short. The higher the current, the larger the kickback voltage according to the formula ($v_{kickback} = L\frac{di}{dt}$), given the same load inductance and switch-off time. Luckily, the external power supply has current limiting capacity, which would limit the kickback voltage. Furthermore, if the subjects would be powered by the regulators on the modules, these regulators would limit their power dissipation by pulling their voltages lower in the event of a short-circuit, which would reduce the current.

E.1.2.2 Sense resistor calculation

The value for the sense resistor can be calculated using the maximum expected current [3]. However, since the breakout boards already met the 5% accuracy specification[1] when using a 100 mΩ resistor, the sense resistor was chosen to have the same value.

E.1.2.3 Power Supply

The INA219 can be supplied with 3-5.5V, so 5V is chosen instead of 3.3V, because 5V is already used on the modules to power the relays. This would remove the need for a 3.3V pin coming from the main board. The sensors were powered with *MB_+5V* since this would normally come from the main board's 5V regulator, which is a separate from the subject's supply (see Section 4.1.1.5 for clarification).

E.1.2.4 I2C pull-up resistors and address pins

Since I2C data and clock lines are open-drain and not push-pull (a logical zero is driven but a logical high is represented by a floating signal), pull-up resistors had to be used to pull the SDA and SCL lines to the supply voltage of the IC. The ideal choices for these resistors are not straight-forward, and depends on many factors like bus capacitance, low-level output voltage of the microcontroller, and rise time. Selecting a value that is too high will result in SDA or SCL signals with slow rise times, which can corrupt the data being transferred. The lower the value, the more power it will dissipate, and if too low, it may even prevent the signal from being driven low [72]. The INA219 breakout boards come with on-board 10 kΩ pull-up resistors, so if four of them are connected to the same I2C bus, this would reduce to 2.5 kΩ. A better approach is to place the pull-up resistors on the microcontroller's side. This will ensure that the resistor value stays constant no matter the number of connected test subjects, and it will reduce the component count on each module. The internal pull-up resistors of the STM32 are typically 40 kΩ [33], which are a bit too large. Instead, external resistors were used. Since the bus capacitance for the test station's I2C lines are not yet known, a standard value close to the collective total of 2.5 kΩ for the four INA219 breakout boards is chosen, i.e. 2.2 kΩ. An oscilloscope was used to verify acceptable signals with this choice (Section 5.1.2). The INA219 IC has two address pins, and depending on which of GND, VCC, SCL, or SDA are connected to these in a particular fashion, up to 16 different I2C addresses are possible. The breakout board only supports four different addresses, since it assumes only GND and VCC lines. Since the new test station also has four sensors, the same approach was followed as that of the breakout board.

[1] It yielded a 4% current measurement error for a 45 mA load when calibrated to 16V and 400 mA. The 45 mA was a rough indication of the Nucleo's total current draw, since JP6 on the prototype board (see Appendix K) was used to measure only the microcontroller current according to Nucleo's user manual [32].

E.2 Test Station Main Board

E.2.1 MOSFET Reverse-Polarity Protection Operation

The MOSFET reverse-polarity protection mentioned in Section 4.2.1.3 works in the following way: Firstly, a P-Channel MOSFET turns on when its gate-to-source voltage is more negative than the threshold, $V_{GS(th)}$, which is at most -4V for the IRF4905. The parasitic body diode of the MOSFET will introduce a voltage drop from drain to source, which is at most 0.75V at 3A (the power supply current limit) for the chosen MOSFET. For the lowest supply of 5V, V_{GS} will be -4.25V, which is more negative than the worst-case $V_{GS(th)}$. When the MOSFET turns on as a result of this, the voltage across the diode is replaced with the voltage across $R_{DS(on)}$. The MOSFET prevents damage from a reverse-polarity connection as follows: Assuming that the MOSFET is on and the power supply is -5V, the source voltage is at almost -5V as well. This means V_{GS} is at 5V, but since V_{GS} has to be less than -4V for the MOSFET to turn on, it is impossible for the MOSFET to be on. Hence, it is off [44].

E.2.2 Pierce Oscillator Circuit for HSE

This section details the design of the pierce oscillator mentioned in Section 4.2.2.2.

The user manual for the Nucleo-64 boards states the specifications of the 8 MHz crystal that is to be used for component X3 [32]. However, this circuit could not be used because the crystal is not commonly available, so a custom circuit had to be designed. The datasheet of the MCU states a 5 pF to 25 pF crystal. The 8 MHz ABLS-8.000MHZ-16-D-4-H-T was chosen for its recommended brand. It has a maximum shunt capacitance of $C_0 = 7\,\text{pF}$, a load capacitance of $C_L = 16\,\text{pF}$, a maximum ESR of $80\,\Omega$, and a frequency stability of 30 ppm (0.003%) [73], which is much better than the internal RC oscillator's $\pm 1\%$ [33].

To calculate the value of the external capacitors, Eq. E.1 is used, where C_s is the stray capacitance [52].

$$C_L = \frac{C_{L1} \times C_{L2}}{C_{L1} + C_{L2}} + C_s \tag{E.1}$$

Letting $C_{L1} = C_{L2} = C_{ext}$, taking C_L to be 16 pF, and assuming C_s to be 10 pF, as recommended by the datasheet of the MCU [33], the external capacitors can be calculated as follows:

$$16 = \frac{C_{ext}^2}{2 \times C_{ext}} + 10$$
$$\implies C_{ext} = 12\,\text{pF}$$

Furthermore, the gain margin requirement of $GM = \frac{g_m}{g_{mcrit}} > 5$ has to be met to maintain a stable oscillation [52]. The oscillator transconductance can be taken from the MCU's datasheet as $g_m = 10mA/V$ [33]. The minimal transconductance, g_{mcrit}, can be calculated as follows:

$$\begin{aligned} g_{mcrit} &= 4 \times ESR \times (2\pi f)^2 \times (C_0 + C_L)^2 \\ &= 4 \times 80 \times (2\pi(8 \times 10^6))^2 \times ((7 + 16) \times 10^{-12})^2 \\ &= 0.427706mA/V \\ \implies GM &= \frac{g_m}{g_{mcrit}} \\ &= \frac{10}{0.427706} \\ &= 23.381 \end{aligned}$$

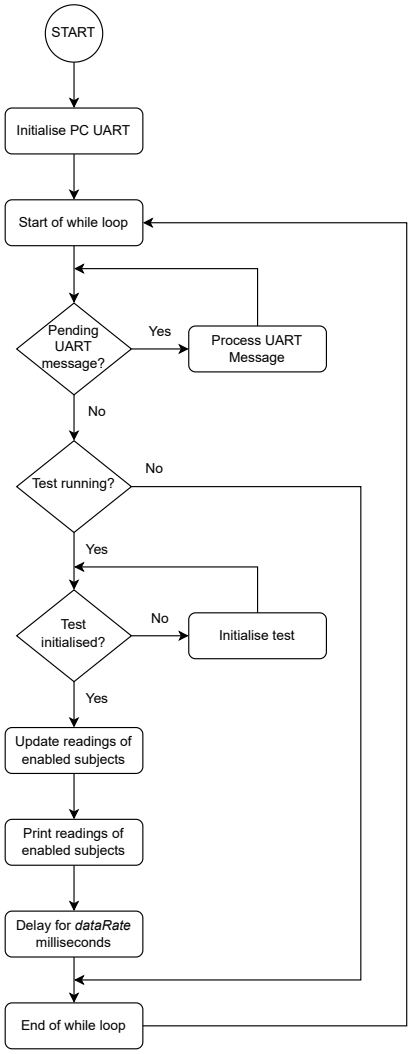
Since the gain margin is greater than 5, the oscillations will start. Finally, the drive level needs to be calculated. This is the total power dissipated in the crystal, and can be controlled by adding an external resistor, as was done in Figure 4.3b. If the resistor is too small, the crystal can be damaged or have its lifetime shortened, while a resistor that is too large can prevent oscillations from starting [52]. Since the external resistor can be calculated by measuring the RMS current through the crystal on an oscilloscope [61], both of which was not available at the time, this approach could not be used. The external resistor affects g_{mcrit} according to Eq. E.2 [52]. Therefore, a gain margin of just over the minimum was chosen to calculate R_{EXT}. For $GM = 6$, R_{EXT} was found to be $231.74\,\Omega$. Therefore, a $220\,\Omega$ resistor was chosen, which led to a safe gain margin of $GM = 6.23$.

$$g_{mcrit} = 4 \times (ESR + R_{EXT}) \times (2\pi f)^2 \times (C_0 + C_L)^2 \tag{E.2}$$

E.3 Embedded Software of Test Station and Test Subjects

E.3.1 Test Station Software Architecture

As mentioned in Section 4.3.1, Figure E.2a shows a diagram illustrating the main file's logic, and Figure E.2b shows the test subject struct.



(a) Test station main loop

```c
typedef struct TestSubject {
    char id;
    uint8_t enabled;
    uint8_t powerState;
    GPIO_TypeDef* GPIO_Port;
    uint16_t GPIO_Pin;
    uint16_t i2cAddress;
    const uint8_t ivSensorI2cAddress;
    float supplyVoltage_V;
    float supplyCurrent_mA;
    TIM_HandleTypeDef* htim;
    uint32_t clk_freq_Hz;
    uint8_t uart_operational;
    uint8_t i2c_operational;
    GPIO_TypeDef* SPI_NSS_Port;
    uint16_t SPI_NSS_Pin;
    GPIO_TypeDef* gpio1_port;
    uint16_t gpio1_pin;
    GPIO_TypeDef* gpio2_port;
    uint16_t gpio2_pin;
    GPIO_TypeDef* gpio3_port;
    uint16_t gpio3_pin;
    uint8_t stateReadByGPIOs;
} TestSubject;
```

(b) Test subject struct

Figure E.2: Test station main loop flow diagram and test subject struct.

E.3.2 Clock Frequency Measurement

As mentioned in Section 4.3.4, the calculation of the auto-reload value and prescaler, and also the setups for the test station and subjects are discussed in this section.

E.3.2.1 Calculation of ARR and PSC

To determine the prescaler and auto-reload value of the timer that uses an external clock signal, a few assumptions were made. Firstly, that the maximum allowable clock frequency would be 10 MHz, which is higher than the minimum requirement of 1 MHz (see Section 3.7). Secondly, that the advanced timer would not overflow more than once every 100 ms, otherwise the basic timer would have to reset it more frequently than that, causing interrupts too often, and hence slowing down the processor. And thirdly, that the auto-reload value is the largest round number closest to its maximum 16-bit value to simplify calculations. It is chosen to be 50000, making ARR 49999. According to Eq. E.3, when assuming $f_{clk} = 10\,000\,000\,\text{Hz}$ and $T_{TIM} = 0.1\,\text{s}$, the minimum prescaler required to stay above the timer period is 20 (PSC = 19). Choosing the calculated prescaler and assuming one measures the timer's counter value every 50 ms, this meant that for a 10 MHz signal (which overflows every 100 ms), the count would be ($\frac{50}{100} \times 50000 =$) 25000. For a 1 MHz signal (overflowing every 50 ms), the expected count is 2500, etc.

$$T_{TIM} = \frac{(ARR + 1)(PSC + 1)}{f_{clk}} \tag{E.3}$$

E.3.2.2 Test station software

The setup for the advanced timers was according to the default settings, besides selecting ETR2 as a clock source and changing ARR and PSC to the calculated values. Appendix O shows the timer and pin used for each subject. Since Timer 6 was one of the basic timers, it was used to periodically read and reset the counter for each of the test subject timers. This was done by reading `subjectX->htim->Instance->CNT` and then setting it to zero. Since the subjects outputted a 9 MHz clock (Section E.3.2.3) which would cause a timer overflow every 111.11 ms, the count would be 22500 when measuring every 50 ms. Therefore, to get to the system clock of 72 MHz, the count had to be multiplied by 3200. The advanced timers were started using `HAL_TIM_Base_Start`, and the basic one using `HAL_TIM_Base_Start_IT`. Appendix Q.1.4 includes the common timer callback function.

E.3.2.3 Test subjects software

To toggle the GPIO pin such that a square wave representative of the system clock could be outputted, a clock divider could simply be used in the clock configuration tool if MCO was used. However, when dividing down to a low enough frequency, this yielded odd values like 1.125 MHz. Instead, it was decided to toggle a timer at 9 MHz, which was the highest frequency that could be reached without significant distortion (see Section 5.3.3). To achieve this, Timer 2's channel 1 was set to *Output Compare CH1*, PSC was set to 0 and ARR to 3 to reach 18 MHz, and *Update Event* and *Toggle on match* were selected

under the parameter settings. Besides this, the default settings were used. Finally, the timer was started using `HAL_TIM_OC_Start(&htim2, TIM_CHANNEL_1)`.

E.3.3 I2C Verification

This Section explains the software setups for the I2C verification, as mentioned in Section 4.3.5.

E.3.3.1 Test subjects setup and software

The I2C setup in the device configuration tool was mostly done according to the default settings, except for enabling interupts and setting the 7-bit slave address in the device configuration tool. The I2C slave code was put in a separate file for modularity, and was inspired by an online tutorial [74]. Listen mode was enabled using `HAL_I2C_EnableListen_IT`, which would trigger an ISR named `HAL_I2C_ListenCpltCallback`, and from which listen mode would be enabled again. Whenever the I2C address sent by the test station would match that of the subject, the ISR named `HAL_I2C_AddrCallback` would be called automatically. Inside this function, the subject could detect whether the test station wanted to write or read to it. For the former, the function `HAL_I2C_Slave_Sequential_Receive_IT` would be used to receive the incoming data. Should the test station then request data, the subject would square the number that it received, and send it back using `HAL_I2C_Slave_Seq_Transmit_IT`. The sequential transmit- and receive functions of the slave only support interrupt- and DMA mode, which is why polling mode was not used. The function definition of `HAL_I2C_AddrCallback` can be seen in Appendix Q.1.5.

E.3.3.2 Test station setup and software

For the test station, the default *parameter settings* were used. As mentioned, `HAL_I2C_Master_Transmit` and `HAL_I2C_Master_Receive` was used for transmission and reception. The subjects' I2C addresses had to be bit-shifted to the left by one since the I2C functions assume eight-bit addresses. The full test station I2C code can be seen in Appendix Q.1.8.

E.3.4 UART Verification

As mentioned in Section 4.3.6, this section includes the UART verification software setups for the test station and test subjects.

E.3.4.1 Test station setup and software

The default settings were used for the UART setup of the test station, except for changing the baud rate from the slow 9.6 kbps to a more standard 115.2 kbps and enabling

interrupts for data reception using `HAL_UART_Receive_IT`. To transmit data, the polling mode `HAL_UART_Transmit` was used. The test station's UART transmit code can be seen in Appendix Q.1.6, and its receive code is very similar to the callback function of Section 4.3.8.1. It only also checked if the number it received matched one minus the number sent.

E.3.4.2 Test subjects setup and software

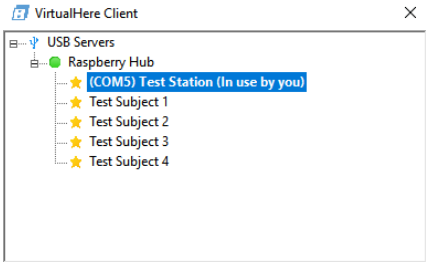
For the subjects, the same settings were used as for the station. They also had the same callback function as the station, which called a function to process the message, subtract one, and to transmit the result back, which can be seen in Appendix Q.1.7.

E.4 Raspberry Pi

E.4.1 VirtualHere Server and Client

As mentioned in Section 4.4.2, Figure E.3a shows how the server is started from the terminal, and Figure E.3b shows the client software displaying the test station's COM port and all four subjects' COM ports.

(a) Server



(b) Client

Figure E.3: VirtualHere server (on Raspberry Pi) and client (on PC).

E.4.2 Script Flow Diagram

Figure E.4 shows a diagram of the Raspberry Pi's server script, as referred to in Section 4.4.3.

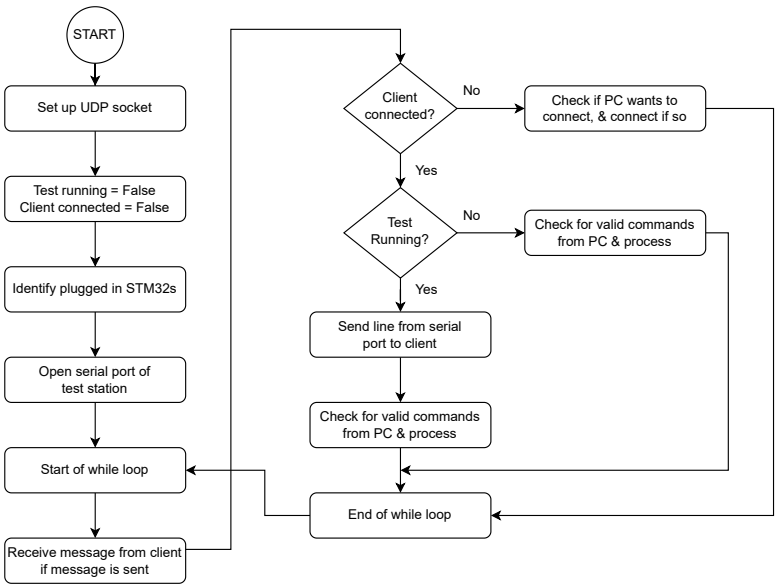


Figure E.4: Flow diagram of the Raspberry Pi Script.

E.4.3 UDP Communication

This section explains how to send and receive UDP messages in Python, as Section 4.4.3.1 mentioned. To send a message using UDP, a UDP socket first has to be created. This can be done as follows, where the IP address has to be the same one as for the VNC server, and the port any available port on the PC [75]:

```
import socket

buffer_size = 1024
msg_to_client = "Hello client"
server_port = 2222
server_ip = '192.168.1.1'
encoded_msg_to_client = msg_to_client.encode('utf-8')
udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
rpi_socket.bind((server_ip, server_port))
rpi_socket.setblocking(0)  # set the socket to be non-blocking
```

To receive and decode a message from the client (PC), the `recvfrom` and `decode` methods can be used as follows, where `buffer_size` is the maximum number of bytes that can be received, so setting it to a large value will not affect performance [76]:

```
client_msg, client_addr = rpi_socet.recvfrom(buffer_size)
decoded_client_msg = client_msg.decode('utf-8')
```

In this way, the Pi can identify the address of the client, to whom it can then send data like the current and voltage. The address is a tuple with the first element the IP address and the second element the port. The server can then send messages to the client using `sendto`:

```
rpi_socket.sendto(encoded_msg_to_client, client_addr)
```

E.5 Graphical User Interface

E.5.1 High-Level Flow Diagram

As mentioned in Section 4.5, Figure E.5 shows a high-level flow diagram of the GUI software, excluding the graphs and GPIO writes.

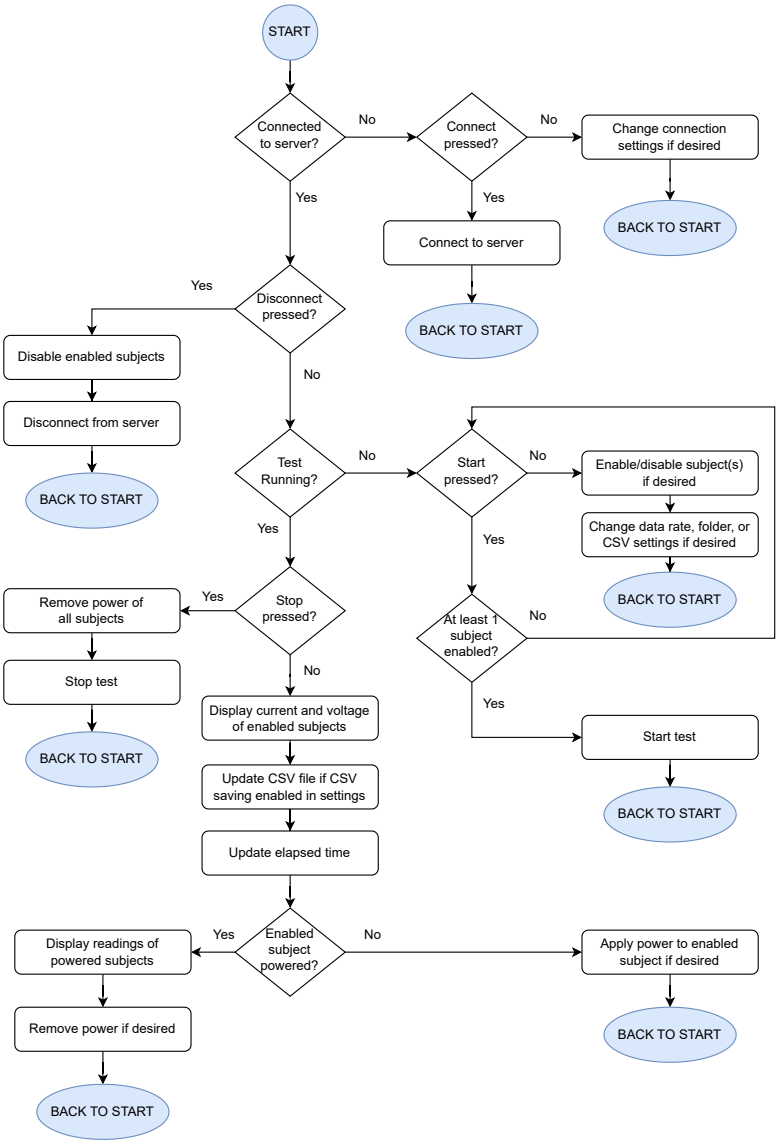


Figure E.5: High-level flow diagram of the GUI software.

Appendix F

Individual Circuit Schematics

F.1 Test Station Modules

F.1.1 Configurable Power Supply

As mentioned in Section 4.1.1.4, Figure F.1 shows the shematic of the configurable power supply without the LEDs.

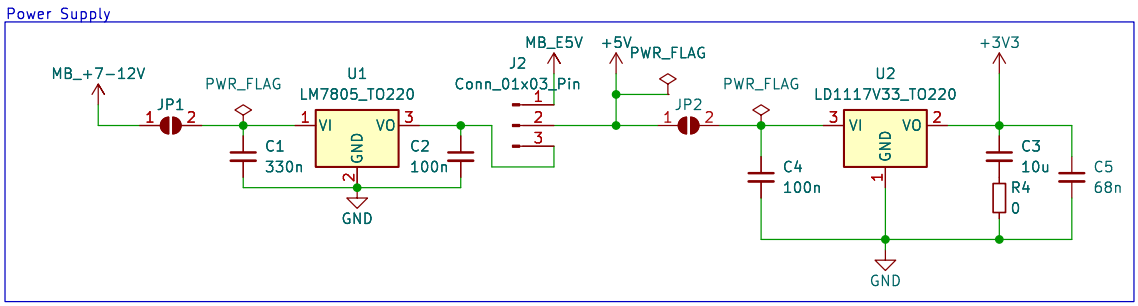
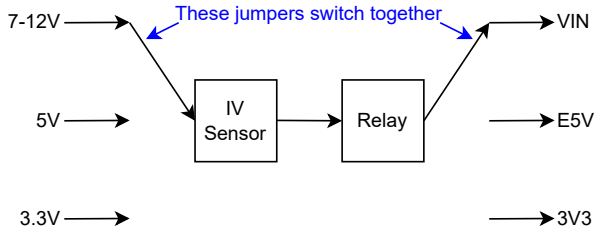


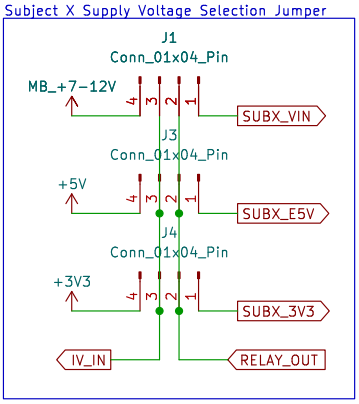
Figure F.1: Configurable power supply supply for the modules.

F.1.1.1 Voltage selection jumper

Figure F.2 shows a diagram of the voltage selection jumper, which Section 4.1.1.6 referred to.



(a) Diagram



(b) Schematic

Figure F.2: Voltage selection jumper.

F.1.2 Current- and Voltage Sensor

As mentioned in Section 4.1.3, Figure F.3 shows the final circuit for the INA219.

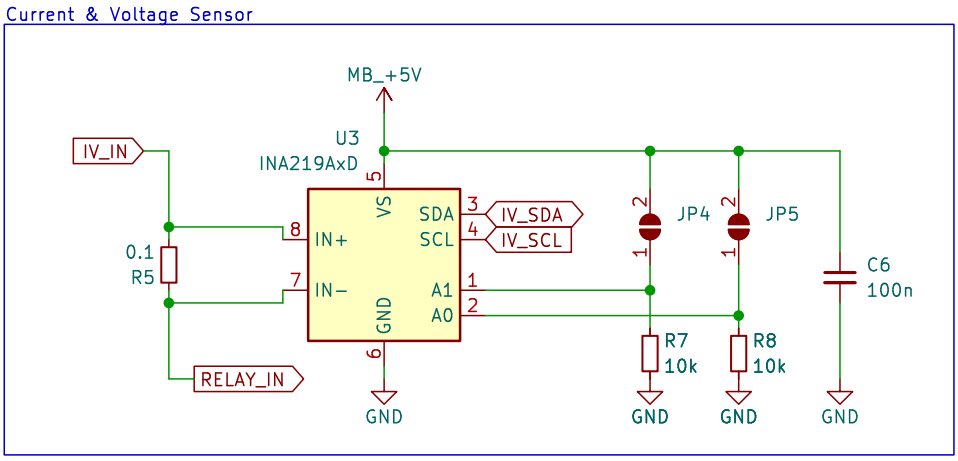


Figure F.3: Circuit schematic of the current and voltage sensor [3].

F.1.3 Module Connectors

Figure F.4 shows a schematic of the module connectors, as referred to in Section 4.1.4.

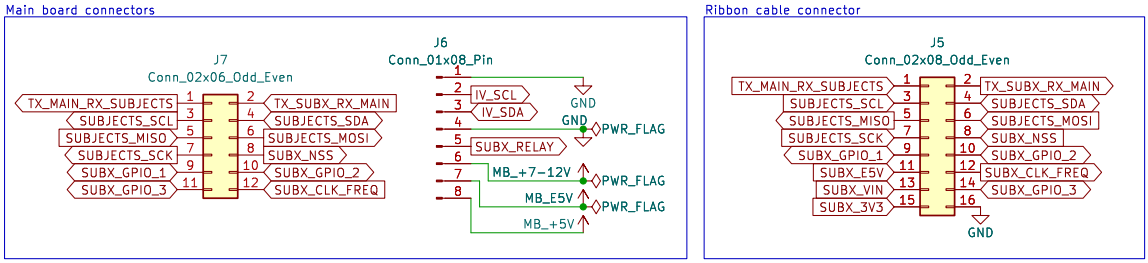


Figure F.4: Schematic of the various module connectors.

F.2 Test Station Main Board

As mentioned in Section 4.2.1.4, Figure F.5 shows a complete schematic of the main board's power supply, excluding the indicator LEDs.

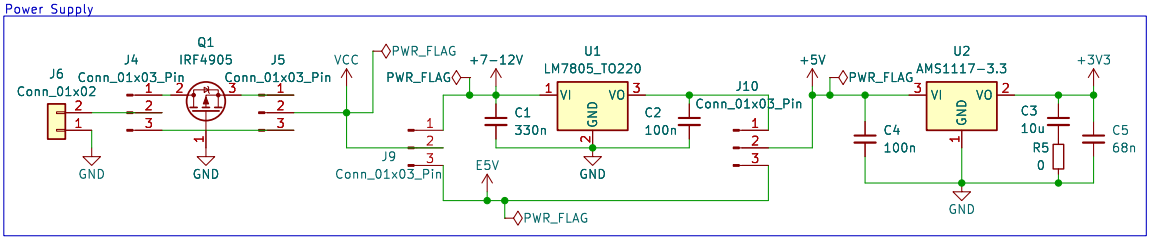


Figure F.5: Power supply of the main board.

F.2.1 MCU and Surrounding Components

Figure F.6 shows the final schematic of the MCU and its surrounding components, as referred to in Section 4.2.2.5. The STM32F446RE is shown, although the 303 is used in the final circuit. The only difference in their pinouts is the VCAP_1 pin, which is called PB11 for the 303.

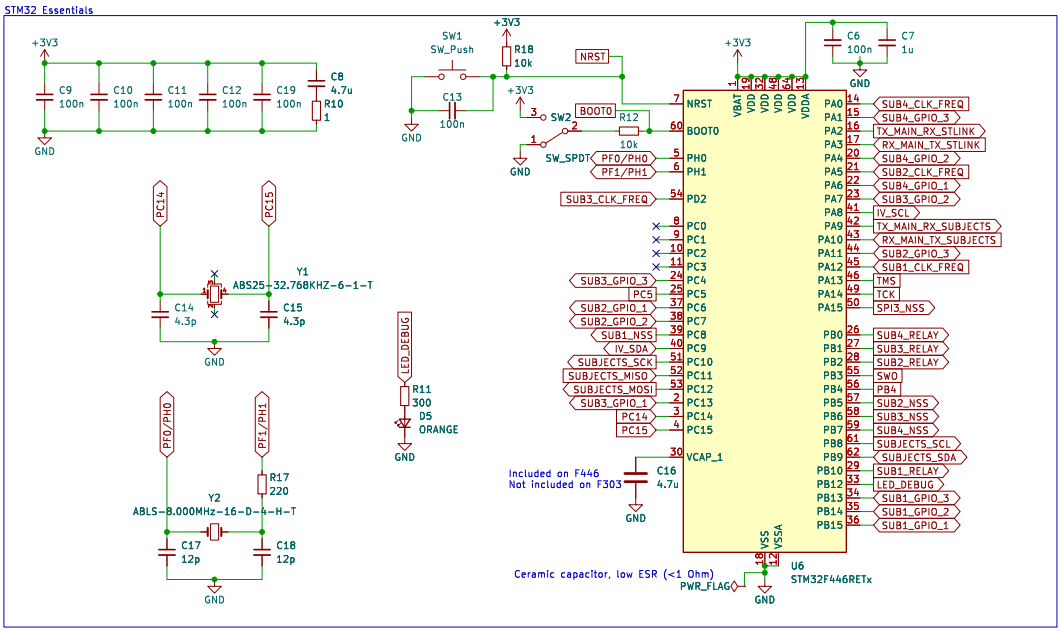


Figure F.6: Schematic of the MCU and its essential components.

F.2.2 ST-LINK Connections

As mentioned in Section 4.2.3, Figure F.7 shows the connections between the MCU and the snapped-off ST-LINK. Pin 1 is not connected, because the series resistor in the Nucleo schematic is not fitted.

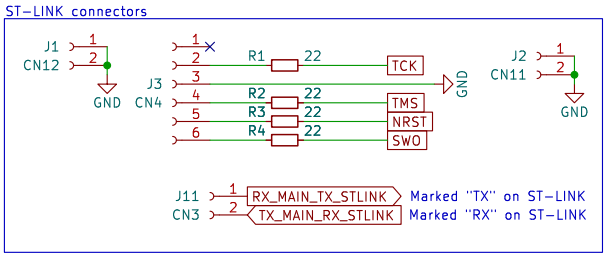


Figure F.7: Connections between the ST-LINK and MCU.

F.2.3 UART One-to-Many Circuit

Figure F.8 shows the nested AND gates used in the one-to-many UART circuit introduced in Section 4.2.4. As can be seen, they were powered with the main board's 3.3V regulator.

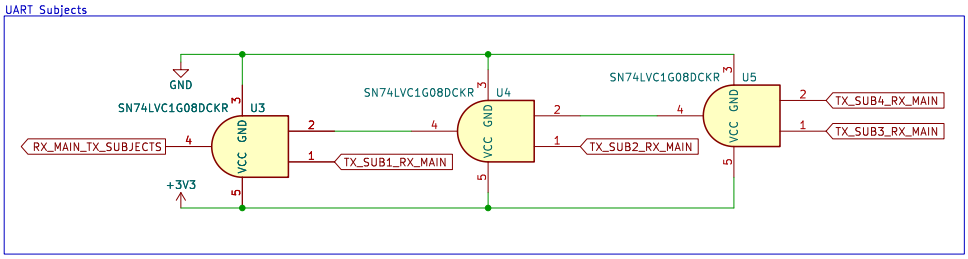


Figure F.8: UART one-to-many circuit.

Appendix G

PCB Schematics

G.1 Test Station Connection PCB Schematic

Figure G.1 shows a full schematic of the connection PCB, as referred to in Section 4.1.5.

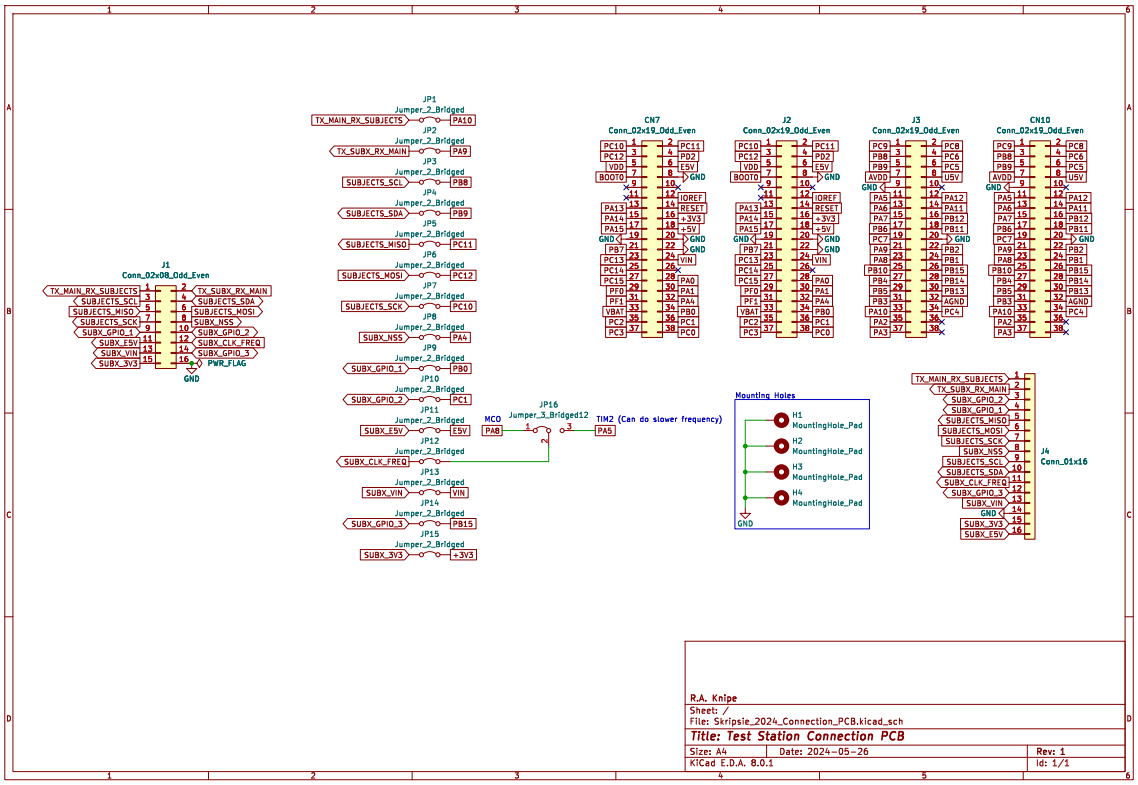


Figure G.1: KiCad schematic of connection PCB.

G.2 Test Station Modules Schematic

Figure G.2 shows a full schematic of the test station module PCB, as referred to in Section 4.1.6.

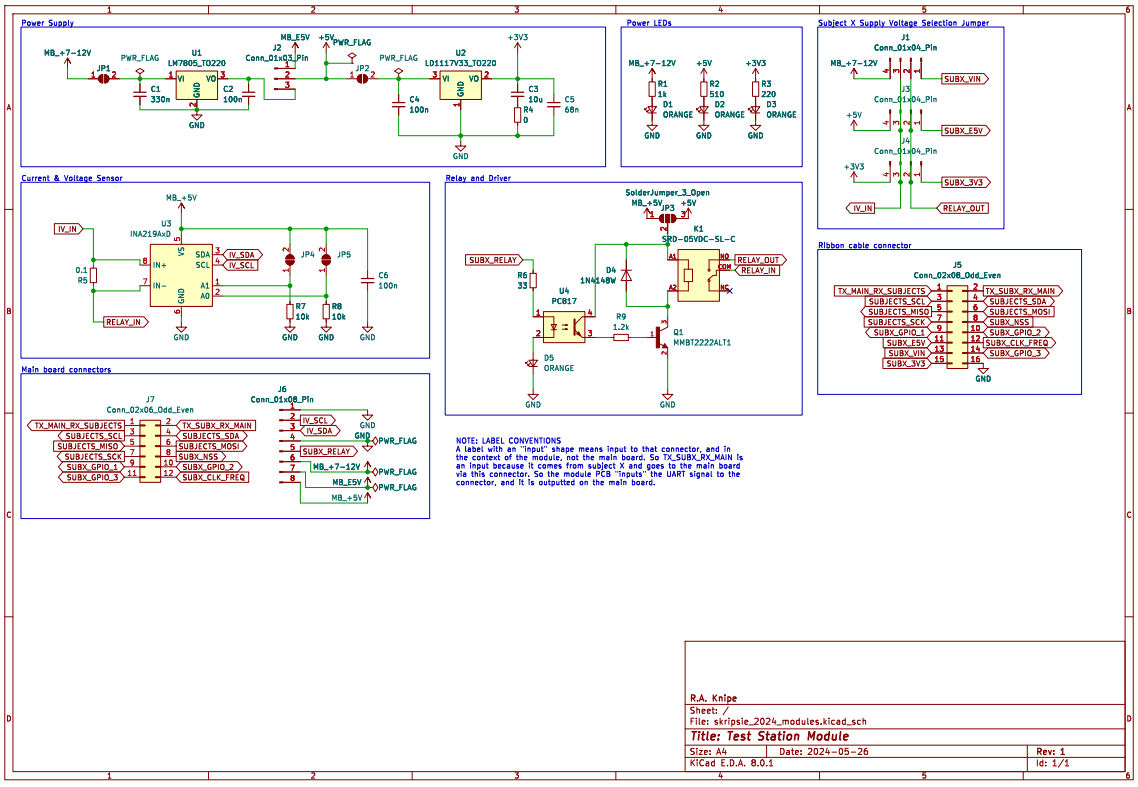


Figure G.2: KiCad schematic of a test station module PCB.

G.3 Test Station Main Board Schematic

Figure G.3 shows a full schematic of the test station main board PCB, as referred to in Section 4.2.6.2.

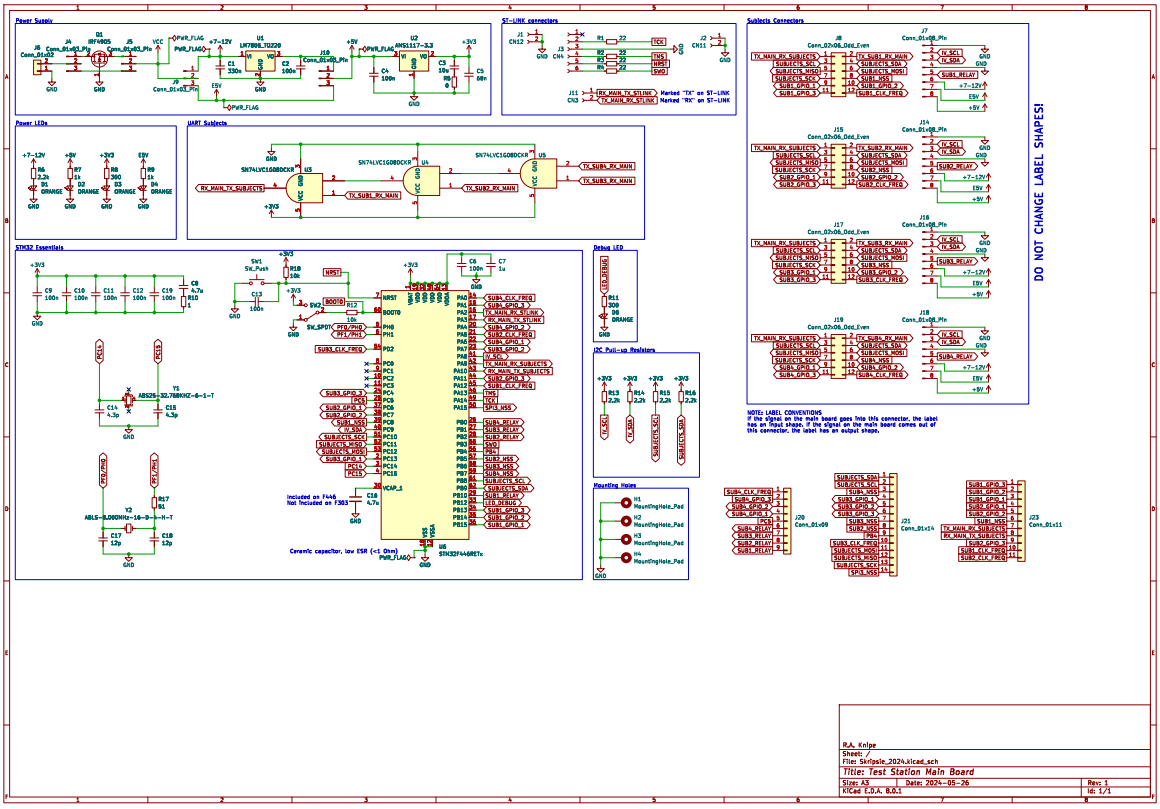


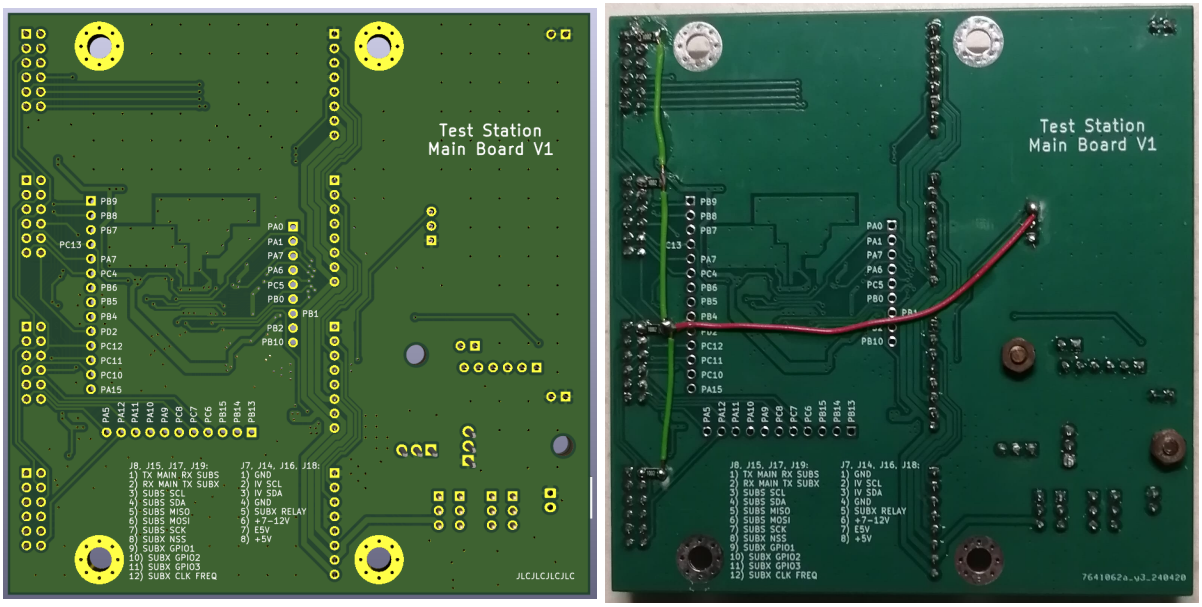
Figure G.3: KiCad schematic of test station main board PCB.

Appendix H

PCB Views

H.1 Main Board PCB: Further Views

Figure H.1 shows the underside of the main board PCB introduced in Section 4.2.6.2. As mentioned in Section M.3.4.3, a UART-related modification was made.



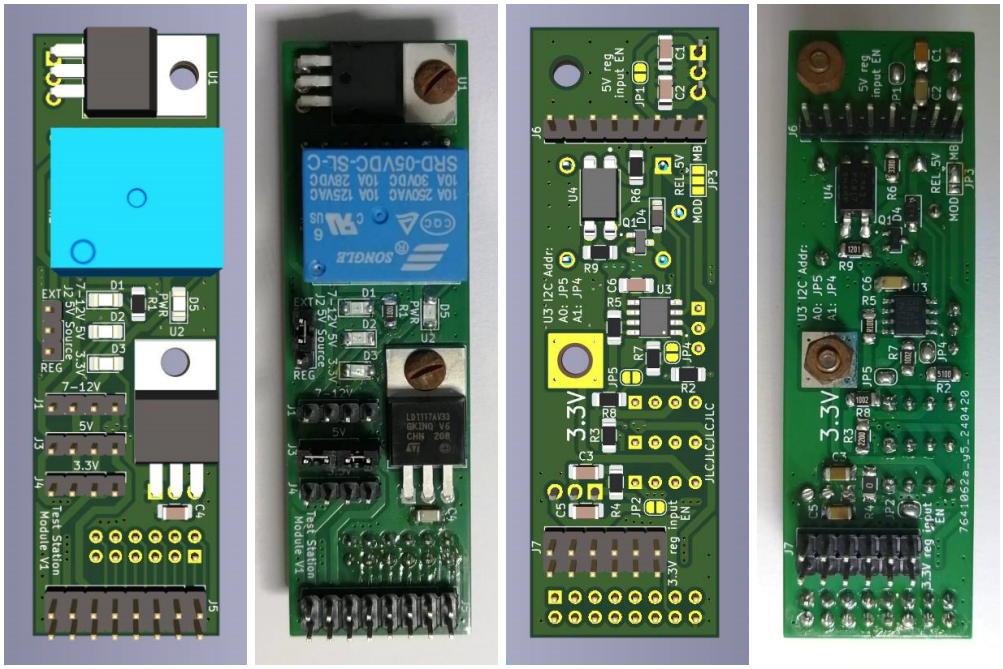
(a) KiCad rendering.

(b) Assembled PCB with modifications.

Figure H.1: Bottom view of the main board in KiCad and in real life.

H.2 Modules PCB: Further Views

As mentioned in Section 4.1.6, Figure H.2 shows the top- and bottom views of the modules PCB in KiCad and in real life.



(a) KiCad (b) Assembled (c) KiCad (d) Assembled

Figure H.2: Top (left) and bottom (right) views of the module.

H.3 Assembled Test Station

Figure H.3 shows the complete test station main board populated with the modules and ST-LINK programmer, as referred to in Sections 3.6 and 4.2.6.2.

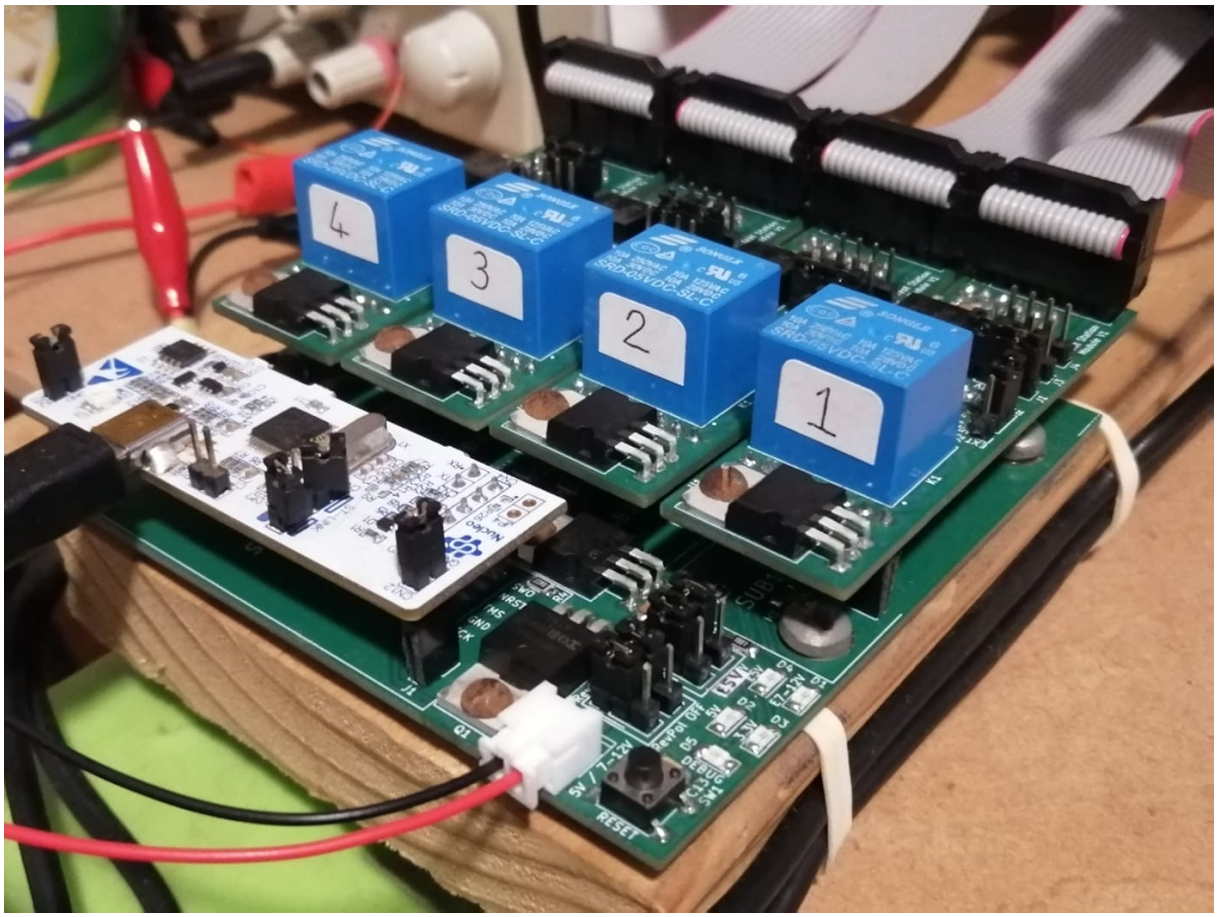


Figure H.3: Assembled test station.

Appendix I

GitHub Repositories

Figure I.1 shows the repositories used to backup the software for the project. This included the embedded software of the test station and each test subject (Section 4.3), the software for the GUI (Section 4.5), and that of the Raspberry Pi's server script (Section 4.4). These repositories are not public but copies of them can be accessed through the following OneDrive link by members of Stellenbosch University: https://stellenbosch-my.sharepoint.com/:f:/g/personal/23540427_sun_ac_za/EikVt5U2T_FClg4ixW8w1sQB3vhC2rRbgw00ve5qKWPDiw?e=xlFTYK

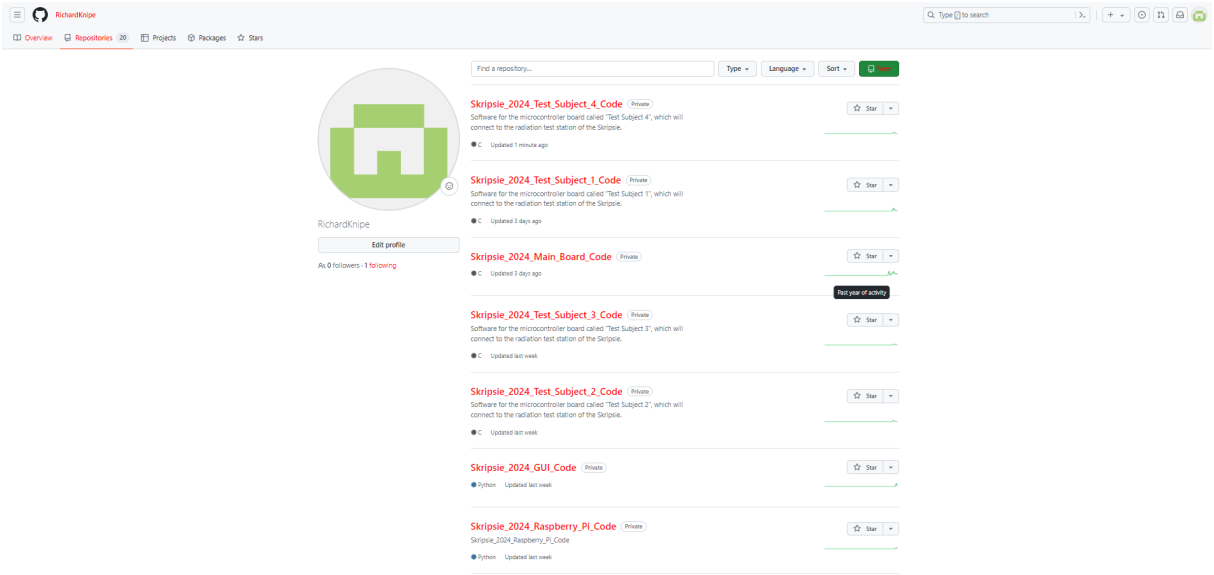


Figure I.1: GitHub repositories created for the project.

Appendix J

Project Expenses

Figure J.1 shows the expenses of the project, which totalled R5356.46.

Description	Supplier	Item	Qty	Cost per Item/Kit	Total Cost
Current sense resistors	DigiKey	MFHA1206R1000FC	4	R5.25	R21.02
IV sensor	DigiKey	INA219AIDR	4	R47.25	R189.00
VCAP capacitor	DigiKey	C3216X7R1E475K160AC	1	R6.05	R6.05
LSE oscillator capacitors	DigiKey	C0805C439BCGAC7800	2	R7.56	R15.12
HSE oscillator capacitors	DigiKey	VJ0805A120FXJCW1BC	2	R6.05	R12.10
PMOS reverse polarity	DigiKey	IRF4905PBF	1	R42.53	R42.53
AND gate	DigiKey	SN74LVC1G08DCKR	3	R6.24	R18.72
Main MCU	DigiKey	STM32F446RET6	1	R189.95	R189.95
Backup MCU	DigiKey	STM32F303RET6	1	R196.75	R196.75
LSE crystal	DigiKey	ABS25-32.768KHZ-6-1-T	1	R13.99	R13.99
HSE crystal	DigiKey	ABLS-8.000MHZ-16-D-4-H-T	1	R10.02	R10.02
Mini USB Cable	DigiKey	AK672M/2-2	5	R60.86	R304.30
D-Sub Connector Male	DigiKey	171-025-103L001	1	R31.94	R31.94
D-Sub Connector Female	DigiKey	171-025-203L001	1	R37.61	R37.61
Test Subject	DigiKey	NUCLEO-F303RE	4	R207.71	R830.84
USB to Ethernet Raspberry Pi	Takealot	Raspberry Pi 3 Model B+, Single Board Computer, RPI3-MODBP	1	R1,099.00	R1,099.00
USB to Ethernet USB hub	Takealot	3-Port USB Hub For 3 USBs EB-111	1	R107.00	R107.00
USB to Ethernet Charger	Takealot	LDNIO 18W A303Q Adaptive Fast Charger + Micro USB Cable	1	R229.00	R229.00
Relays	Micro Robotics	Relay 5V Coil 10A (4 Pack)	1	R32.20	R32.20
Subject cables	Micro Robotics	Ribbon Cable, 16 Way with Female Connectors, 20cm, 2 Pack	2	R20.70	R41.40
Boot switch	Micro Robotics	Slide Switch - Horizontal - Pitch 2.54 3mm (4 Pack)	1	R11.50	R11.50
Reset button	Micro Robotics	Tactile Switch 4 Pin H=6mm SMD (10 pack)	1	R11.50	R11.50
Capacitor kit	Micro Robotics	Capacitor Kit SMD 1206 (720 Pieces)	1	R131.10	R131.10
Resistor kit	Micro Robotics	Resistor Kit SMD 1206 (800 pcs)	1	R89.70	R89.70
BJT	Micro Robotics	Tran SN2222A-SMD SOT23 (25 Pack)	1	R40.25	R40.25
Optocouplers	Micro Robotics	4N25 Optoisolator - SMD	1	R20.70	R20.70
LEDs	Micro Robotics	LED SMD 1205 - Orange (100 Pack)	1	R17.25	R17.25
Diode	Micro Robotics	1N4148W Diode SMD T4, SOD-123 (100 Pack)	1	R17.25	R17.25
Module headers	Micro Robotics	IDC 2X8 Male Header - 2.54mm (4 Pack)	2	R11.50	R23.00
3.3V regulators	Micro Robotics	LD1117 Voltage Regulator, 3.3V 0.8A, 2 Pack	3	R26.45	R79.35
5V regulators	Micro Robotics	Linear Voltage Regulator, 7805 5V 1A, 2 Pack	3	R10.35	R31.05
Headers	Micro Robotics	Header Male Straight, 40P 2.54 (10)	1	R11.50	R11.50
Jumpers	Micro Robotics	Jumper - Pitch 2.54mm (20 Pack)	1	R11.50	R11.50
Main board power connector	Micro Robotics	JST-XH 2.5mm 2 Pin Cable and Connector Kit 20cm (4 Pack)	1	R16.10	R16.10
PCBs	JLCPCB	5x main board pcbs, 10x module pcbs, 5x connection pcbs	n/a	n/a	R1,416.17
				Grand total	**R5,356.46**

Figure J.1: Project expenses.

Appendix K

Test Station Prototype

Figure K.1 shows a prototype version of the test station and a test subject, all on a Lego support board, which is referred to in Sections E.1.2.2 and 5.1.2. The test station's development board was connected with jumper cables and breadboards to the test subject, the INA219 breakout boards, and a four-channel relay module. A UART FTDI chip was used to debug the UART communication between two STM32 boards. Without this prototype the final test station would likely not have been successful.

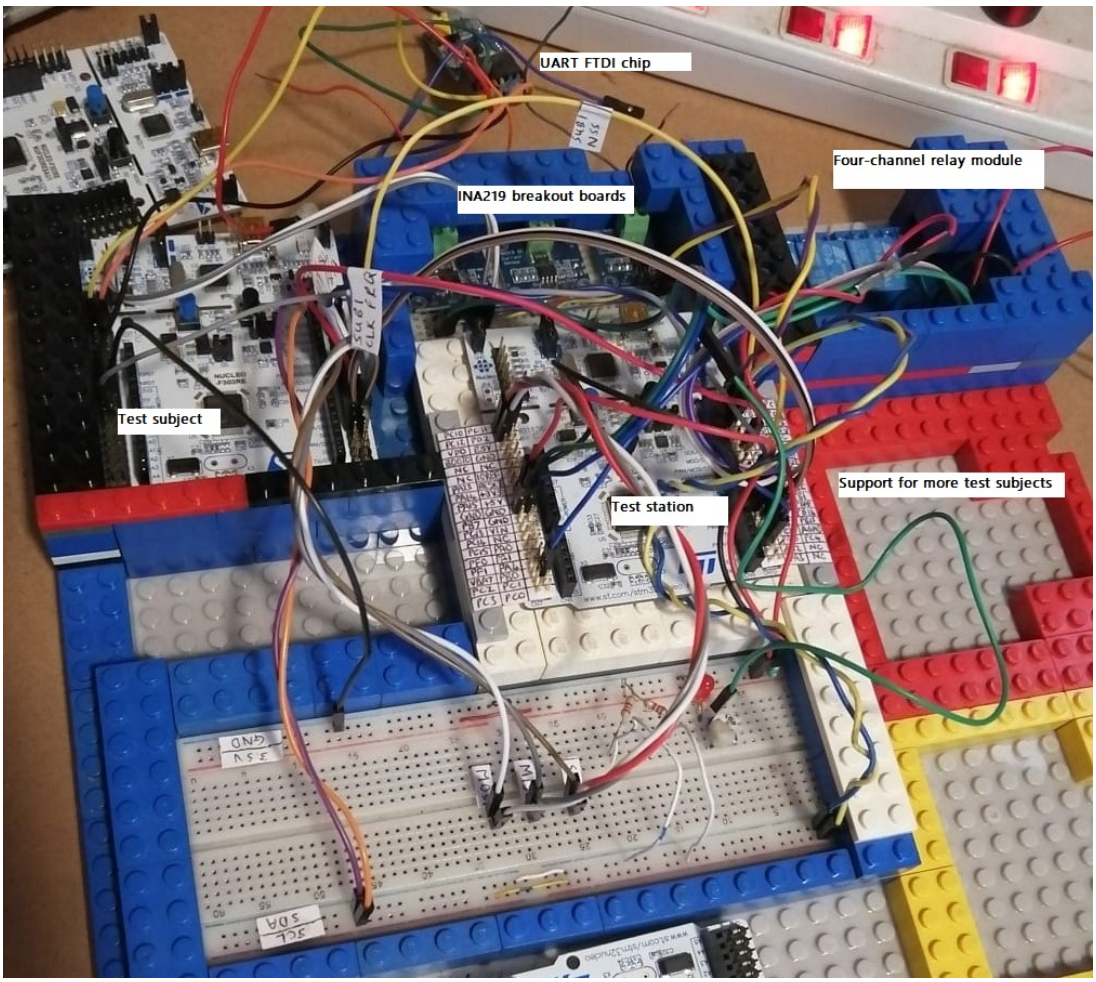


Figure K.1: Test station prototype connected to a single test subject.

Appendix L

GUI Planning

Figure L.1 shows the GUI design before starting with the implementation, as explained in Section 4.5. This ensured that the final product would be satisfactory, and having a picture of this beforehand sped up the development process.

Figure L.1: Test station prototype connected to a single test subject.

Appendix M

Results Details

M.1 Test Station Modules

M.1.1 Relay Driver Functionality

The details for the relay driver test mentioned in Section 5.1.1 are shown below.

M.1.1.1 Relay supplied with the main board's 5V regulator pin

Firstly, a continuity test was done according to the setup in Appendix N.1.1.1, where the relay was supplied at the main board's 5V regulator pin. After applying 3.3V, the multimeter confirmed the anticipated short circuit and the D5 LED (PWR) clearly indicated that the relay driver was switched on. After removing power, the LED switched off and there was no more short circuit.

M.1.1.2 Relay supplied with the main board's external 5V pin

After doing the setup of Appendix N.1.1.2, a continuity test was done where the relay supply voltage was the main board's external 5V. Continuity was again verified with the multimeter when applying 3.3V. The PWR LED came on as expected, and also the 5V LED, indicating that 5V was present on the board and could be supplied to the subject.

M.1.1.3 Relay supplied with the module's 5V regulator

A third test was done to see if the relay could be supplied with the module's 5V regulator, and the setup of Appendix N.1.1.3 was done. When applying 3.3V, the output voltage was measured as 12V, which was expected. The PWR, 5V, and 7-12V LEDs also came on as expected.

M.1.2 Current- and Voltage Sensor

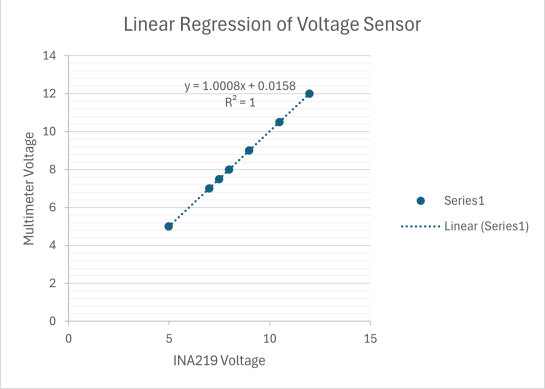
This section expands on the current- and voltage sensor test introduced in Section 5.1.2.

M.1.2.1 Readings and Calibration

Figure M.1a shows the INA219 readings printed to a terminal program every second, where a 9V supply was used to power a test subject [1], and Figure M.1b shows the linear regression used to calibrate the sensor in Excel.

```
Subject 1 Supply voltage (V): 8.824
Subject 1 Supply current (mA): 48.30
Subject 1 Supply voltage (V): 8.820
Subject 1 Supply current (mA): 47.89
Subject 1 Supply voltage (V): 8.824
Subject 1 Supply current (mA): 48.10
```

(a) INA219 readings printed to a terminal program.

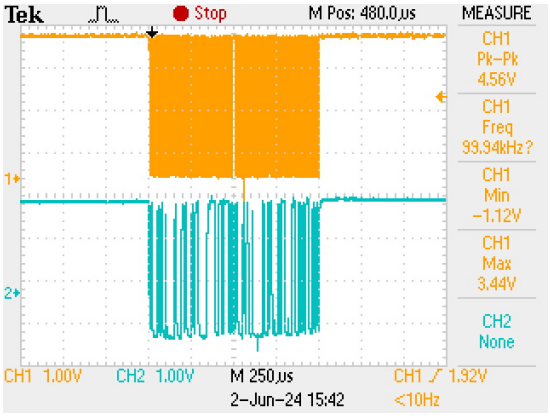


(b) INA219 voltage calibration in Excel.

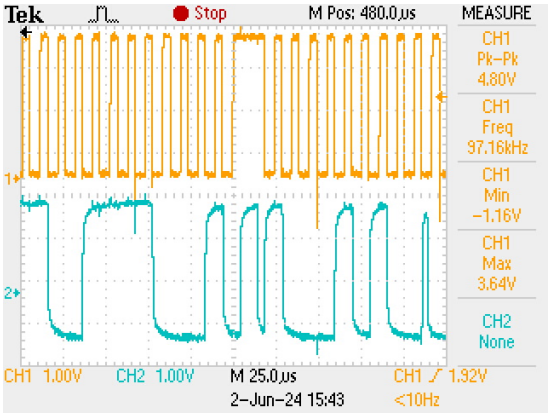
Figure M.1: INA219 readings and calibration.

M.1.2.2 I2C Signals

Figure M.2 shows oscilloscope screenshots of the INA219's I2C signals, which were acceptable enough to be recognised by the embedded software.



(a) Zoomed out.



(b) Zoomed in.

Figure M.2: INA219 I2C signals on an oscilloscope.

Table M.1 shows that the voltage calibration comfortably met the 5% accuracy specification of Section 3.7. The setup of Appendix N.1.3 was used.

[1] This screenshot of the PuTTY terminal program was edited to show only the readings of the current and voltage, and not other readings such as clock frequency, which were irrelevant for this test.

INA219 reading	4.952V	7.015V	9.410V	11.969V
Multimeter reading	4.95V	7.02V	9.41V	11.97V

Table M.1: Voltage readings after calibration.

M.1.3 Configurable Power Supply

This section contains the detailed tests for the module's configurable power supply, as mentioned in Section 5.1.3. For all of these tests, the main board's 5V was used to power the relay such that the additional load of the relay would not interfere with the load of the test subject, which would typically be the default setting. The external 7-12V and 5V were only tested with the test subjects and not for full-load and no-load conditions, since the external supply can deliver up to 3A (see Section 3.3.2.2).

M.1.3.1 Module 3.3V regulator supply

A no-load test was done according to the setup in Appendix N.1.2.1. It is worth noting that the test subjects cannot be programmed when powered by 3.3V, since the output of the 5V regulator on the Nucleo is disabled by desoldering SB2, and the reset line of the ST-LINK must also be disabled by desoldering SB12, according to the user manual of the test subjects [32]. The result for the no-load test was an output voltage of 3.31V for its never-changing input voltage of 5V. For the full-load test, the same load of the 5V supply test was used, which conveniently meant a current of 390 mA that was close to the specification of 300 mA. The result was an output voltage of 3.26V, and the 3.3V regulator was barely warm.

M.1.3.2 Module 5V regulator supply

The no-load test was performed first, using firstly the minimum and then the maximum regulator supply voltage. For the 7V supply, the output voltage measured 4.96V when the relay was turned on with an external 3.3V. For the 12V supply, it measured 4.97V.

According to Section 4.1.1.3, the module's 5V regulator was designed to supply at least 200 mA for the maximum supply voltage of 12V (when the relay was supplied with the main board's 5V regulator). A 200 mA resistive load (25 Ω) was therefore tested according to the setup in Appendix N.1.2.2. The result was an output voltage of 4.97V. One could still touch the regulator, although not for long. This confirmed that the artificial PCB heat sink was working, since the temperature was definitely nowhere near the calculated junction temperature of 95 °C for this current at no heat sink. Although the vacuum environment would cause a higher junction temperature than what was observed, the regulator would definitely still be fine even for long tests.

For the same setup but with the minimum supply voltage of 7V, a current of 750 mA was theoretically possible (see Section 4.1.1.3). A 500 mA load was tested because that was the only load for which a power resistor was available. The voltage dropped to 4.4V. Since the minimum supply voltage of the test subjects are 4.75V (see Section 3.3.1.1), the power supply would not be able to supply the test subjects from its 5V regulator at the minimum supply for such a high current. This was then confirmed to be a dropout voltage issue, since at a supply of 7.5V and a slightly larger current, the voltage was at an acceptable 4.81V.

M.1.3.3 Test subject supply measurements with sensor

Finally, the current- and voltage sensor was used to measure the supply voltage and current of a test subject for five different supply voltages. A supply voltage of 9V is used unless otherwise stated, and the relay is supplied by the main board's 5V. Table M.2 summarises the results, and shows that the current draw is practically doubled when the USB cable is not plugged in, except for 3.3V. This is expected, because the on-board regulator of the subject is bypassed by the removal of SB2.

	12V	7.5V	5V REG	5V EXT	3.3V
A	49mA, 11.81V	48.5mA, 7.28V	45.5mA, 4.93V	43.5mA, 4.79V	41mA, 3.29V
B	91mA, 11.77V	92mA, 4.92V	88mA, 4.92V	87mA, 4.74V	41mA, 3.29V

Table M.2: Current draws of a test subject at various supply voltages measured on the connection PCB, with the USB cable plugged in (A) and out (B).

M.1.3.4 Module Powering a Test Subject

Figure M.3 shows a module's 5V regulator powering a test subject. Each LED coming on indicates that 7-12V, 5V, and 3.3V were available on the module. The PWR LED also showed that the relay was powering the subject via the ribbon cable.

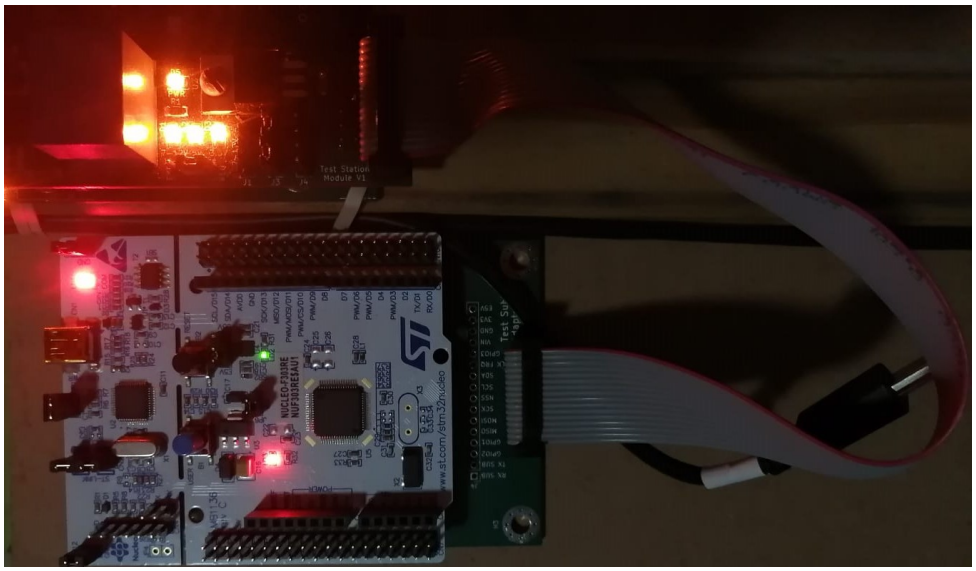


Figure M.3: Test subject being powered by a module.

M.2 Test Station Main Board

M.2.1 Power Supply

This section expands on the main board's power supply results introduced in Section 5.2.1.

M.2.1.1 7-12V and E5V selection

Figure M.4a shows how the main board is supplied by 9V, how the selection is on 7-12V, and how the correct LEDs light up. Figure M.4b shows the main board LEDs lighting up as expected when an external 5V is supplied, and the jumper is on the appropriate setting. Two stickers were added to the PCB to correct the silkscreen text that was accidentally swapped for E5V and E7-12V. Luckily, the LEDs made the issue apparent before the test subjects would possibly be damaged by supplying them with 12V on their E5V pins, as Section 4.2.1.4 mentioned.

(a) 7-12V supply

(b) E5V supply

Figure M.4: Main board LEDs and jumpers for the two voltage selections.

M.2.1.2 Reverse-polarity protection

Figure M.5 shows the reverse-polarity protection working as designed in Section 4.2.1.3, since a -9V supply is applied to the main board without damaging it. To measure the voltage drop across the MOSFET in the event where the polarity was correct, the worst-case scenario of the maximum load was considered. This was when all subjects were powered by 12V, which led to a total bench power supply current draw of about 700 mA. The result was a 20 mV voltage drop from drain to the source, which is practically negligible.

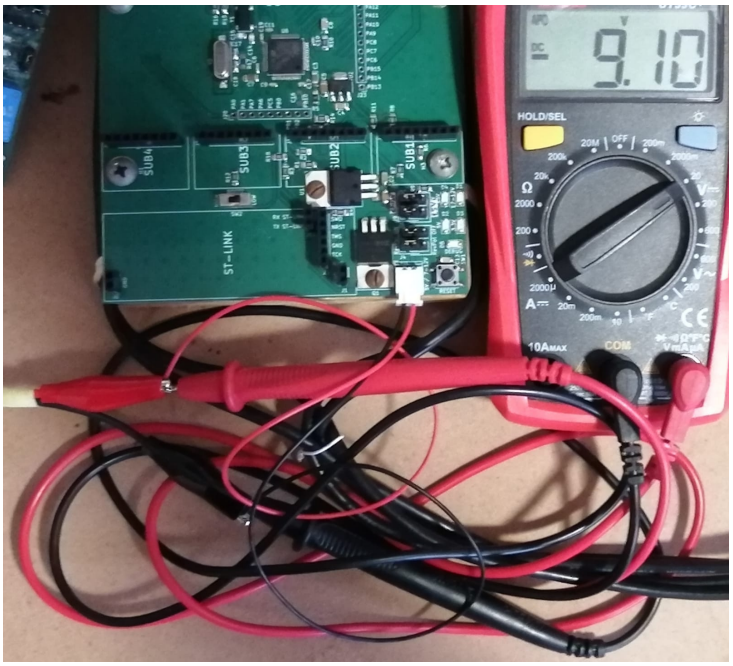


Figure M.5: Main board reverse-polarity protection demonstrated.

M.2.1.3 Powering the Modules and Test Subjects

The four modules were plugged into the main board, which powered them successfully, since their respective LEDs came on. The current of the variable bench power supply when no subjects were powered measured 150 mA for 7V and 170 mA for 12V. Figure M.6 shows how all the test subjects are powered by the main board. Subjects 1 and 2 are powered by 12V, subject 3 by the module's 5V regulator, and subject 4 by the module's 3.3V regulator, as can be deduced from the jumper settings. The voltages and currents measured were practically identical to those in Table M.2, and the current draw of the variable bench power supply was close to the 700 mA of Section M.2.1.2. Since JP5 of each Nucleo is on $E5V$ and not on $U5V$ (except for subject 4, where it is on neither), the subjects are clearly receiving their power from the test station and not from the USB cables.

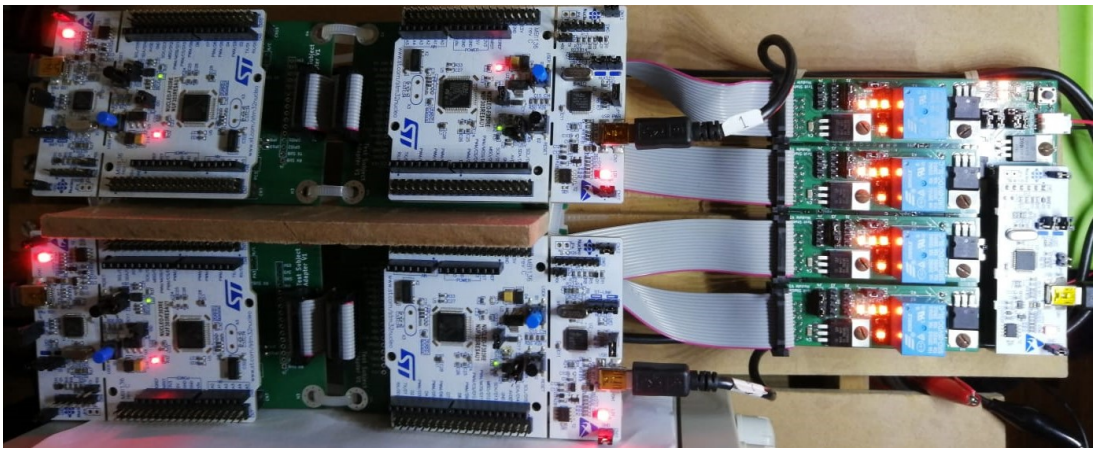


Figure M.6: Subjects powered with different power supplies from the test station.

M.2.2 ST-LINK Programming and Crystal Oscillators

This section includes the details of the results mentioned in Section 5.2.2.

M.2.2.1 ST-LINK Programming

Before controlling the modules, the programming functionality had to work first. This was tested by programming the main board via the ST-LINK with a simple program that turned on the debug LED, as Figure M.7 shows.

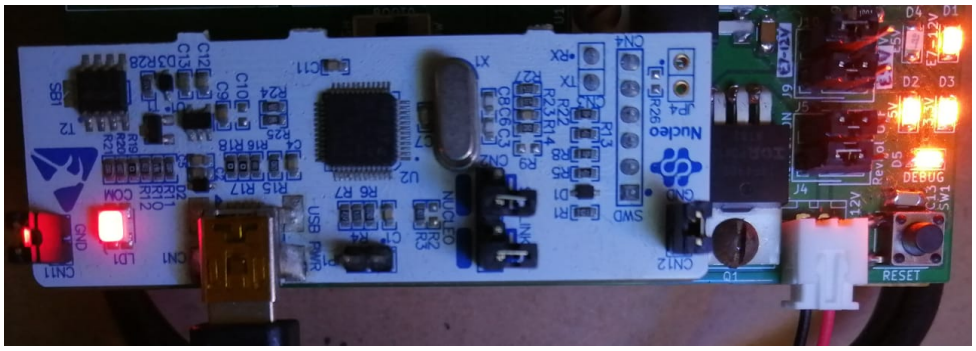
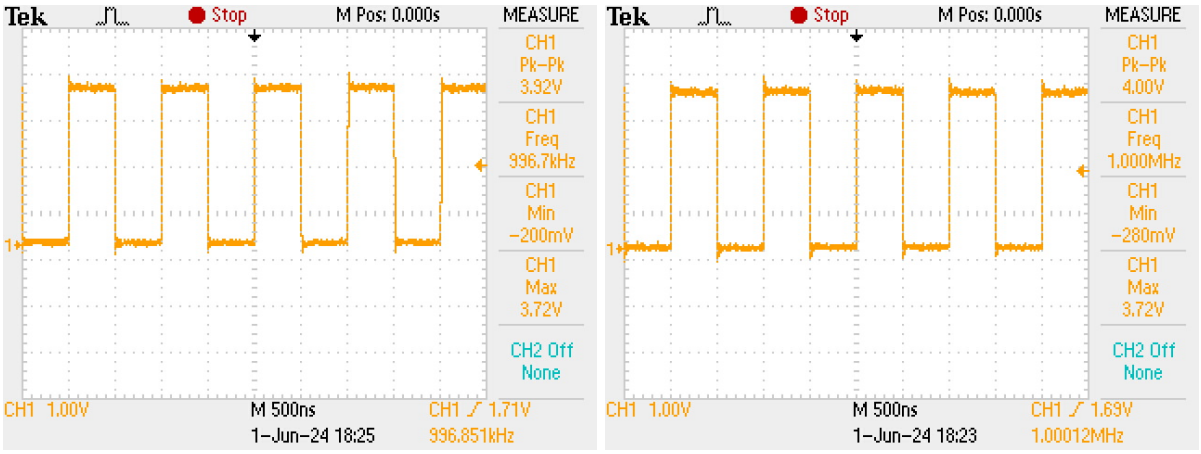


Figure M.7: Main board debug LED illustrating programming functionality.

M.2.2.2 Crystal Oscillators

To test the functionality and effectiveness of the high-speed crystal designed in Section 4.2.2.2, an oscilloscope was used to accurately measure the frequency. This was done by toggling a vacant GPIO pin (using a timer in output compare mode) at a scaled-down version of the system clock, which was fed by either HSE or HSI (see Appendix O for these pins). The oscilloscope captures of Figure M.8 clearly demonstrate that the HSE oscillator (1.000 MHz) led to a much more accurate clock than the HSI one (0.9967 MHz). LSE was not tested since there was no reason to use it in the project; it was simply included to give the user flexibility.



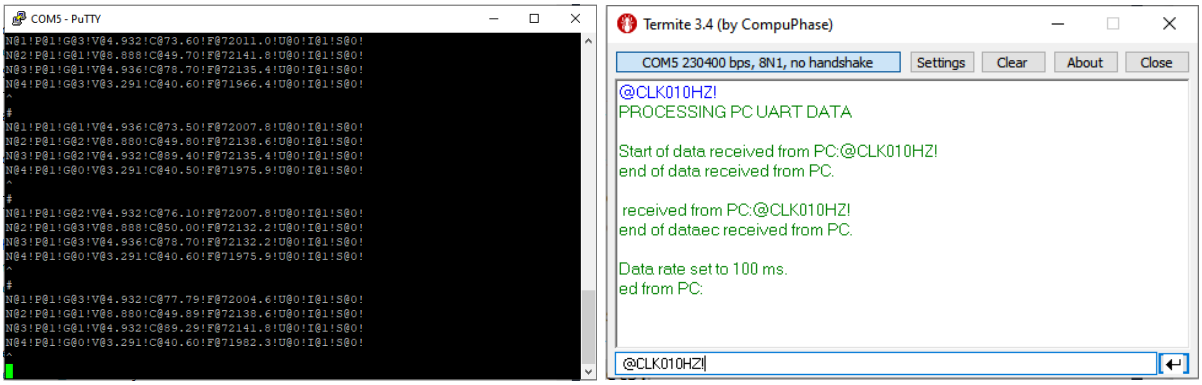
(a) HSI (0.9967 MHz). (b) HSE (1.000 MHz).

Figure M.8: GPIO frequency of a timer when using HSI and HSE.

M.3 Embedded Software

M.3.1 Readings and Commands via a Terminal Program

As alluded to in Section 5.3.1, Figure M.9a shows a screenshot of the PuTTY terminal program, where the test station's serial port outputs the data of the four subjects line-by-line. Figure M.9b shows the test station successfully reading a command and changing its logging frequency.



(a) Test subject readings. (b) Response to a command.

Figure M.9: Terminal programs showing readings and commands.

M.3.2 GPIO Verification

The detailed version of the results in Section 5.3.2 are discussed below.

M.3.2.1 GPIO reading and writing

Figure M.10 shows a screenshot of the Termite terminal program, which shows how subject 1 reports its state as 1 and 2 (represented by `G@1` and `G@2`) alternately every second. Since it does not report state zero, this indicated that it is functional. When a command is written to the test station (COM5) to write to the GPIO of subject 1, the subject (COM9) detects this and changes its GPIO state to 3, which can be seen in the test station's terminal window.

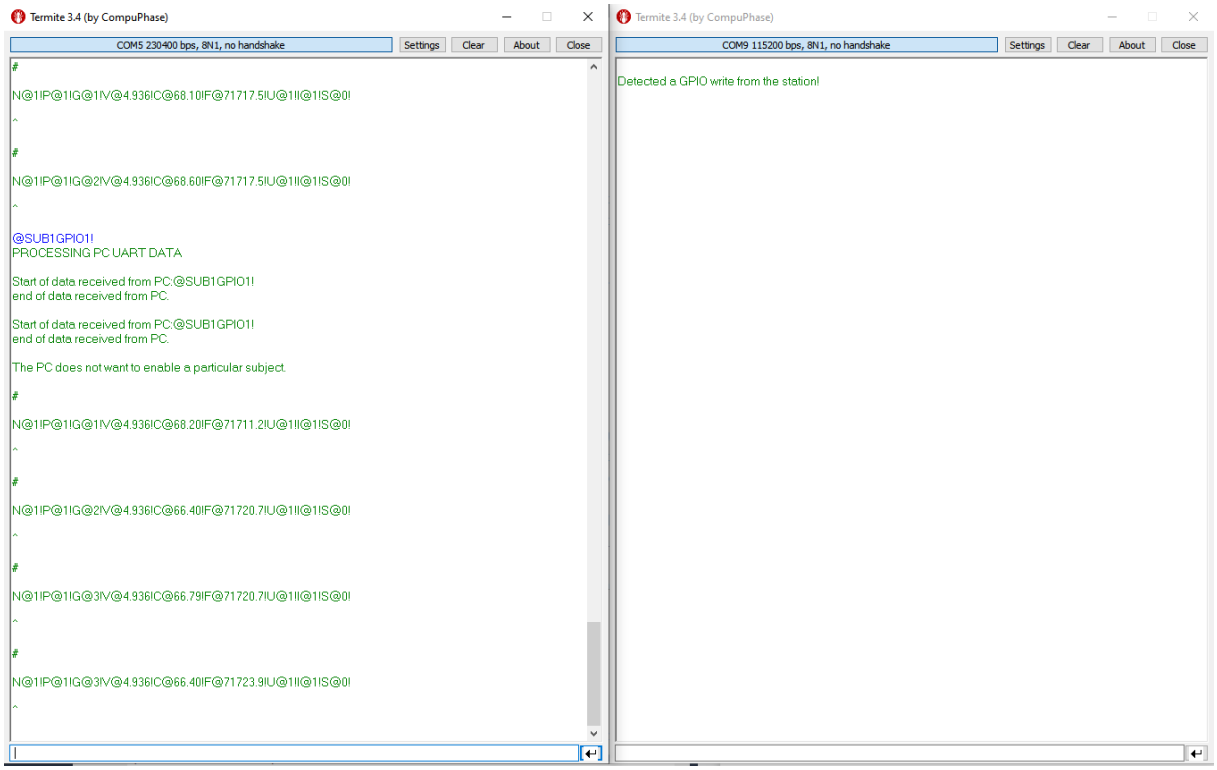


Figure M.10: GPIO reading and writing functionality demonstrated in Termite, with the serial ports of the test station (left) and of subject 1 (right).

M.3.3 Clock Frequency Measurement

The detailed version of the results in Section 5.3.3 are discussed below.

M.3.3.1 Output waveform

Before measuring the scaled-down version of a subject's clock frequency (see Section 4.3.4 for details), the signal quality had to be investigated. Figure M.11 shows an acceptably-shaped 9 MHz waveform of a test subject's clock output when connected to the test station, as set up in Section E.3.2.3. At a higher frequency, the signal would start to deteriorate. Note that this scope capture was done with the internal RC oscillator, not HSE, which is why it is not exactly 9 MHz.

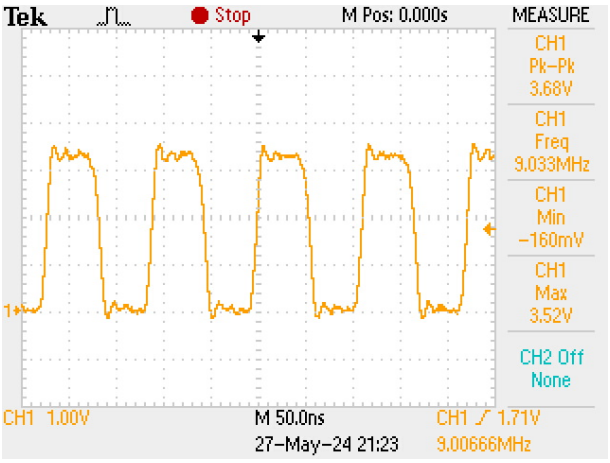


Figure M.11: Clock frequency waveform of a test subject.

M.3.3.2 Measurement

The specification for clock frequency verification was an accuracy of 1% (see Section 3.7). Since the subjects' system clocks were at 72 MHz, this means an acceptable range of 71.28 to 72.72 MHz. This is verified in the terminal program readings of Figure M.9a, since the frequency of any one of the four subjects hovers between 71.95 and 72.15 MHz.

M.3.4 UART Verification

This section expands on the UART verification results of Section 5.3.4.

M.3.4.1 AND gates functionality

As Section 4.2.4 mentioned, the design oversaw the need for pull-up resistors at the inputs to the AND gates, which are necessary if not all four test subjects are connected and powered. To test the AND gates, the modules were removed and the gate inputs were either connected to ground or pulled to 3.3V with external 10 kΩ pull-up resistors, and the output of the AND gate tree was measured on the test point of PA10 (see Appendix O) with a multimeter. The result was that when at least one of the four AND gate inputs was connected to ground, the output was 0.03V, and when all of them were pulled high, the output was 3.36V. Knowing that the AND gates worked at DC, the oscilloscope was used to capture a UART message sent by the first test subject to investigate the signal quality. Figure M.12 shows the UART signals before and after the AND gate of the number 2 being sent from a subject to the test station, at a baud rate of 115200. Clearly, the signal had virtually no loss in quality or strength.

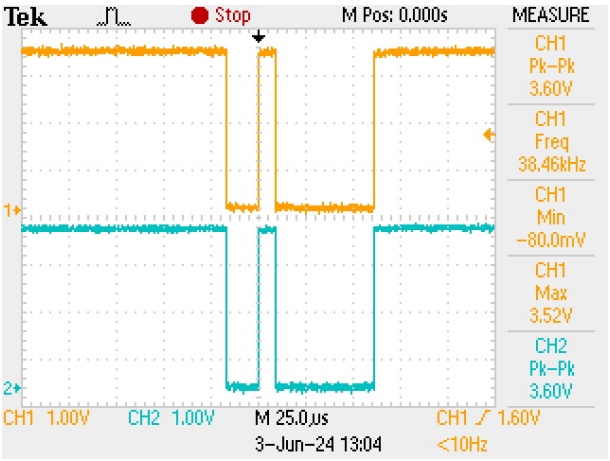


Figure M.12: Test subject UART TX line (bottom) vs AND gate output (top).

M.3.4.2 Subject's receive from and response to FTDI

As explained in Section 4.3.6, the subject would receive a number between 0 and 9 via UART, subtract one, and send back the result. Messages were encoded in a format that started with the character '@' and ended with the character '!'. The character after the start character was the subject's number (1, 2, 3, or 4), and the character after that the number to be subtracted. It was decided to first test the subject's UART receive functionality together with its response of the result, and Figure M.13 illustrates that this worked as expected. The UART FTDI chip of Appendix K was used to test this.

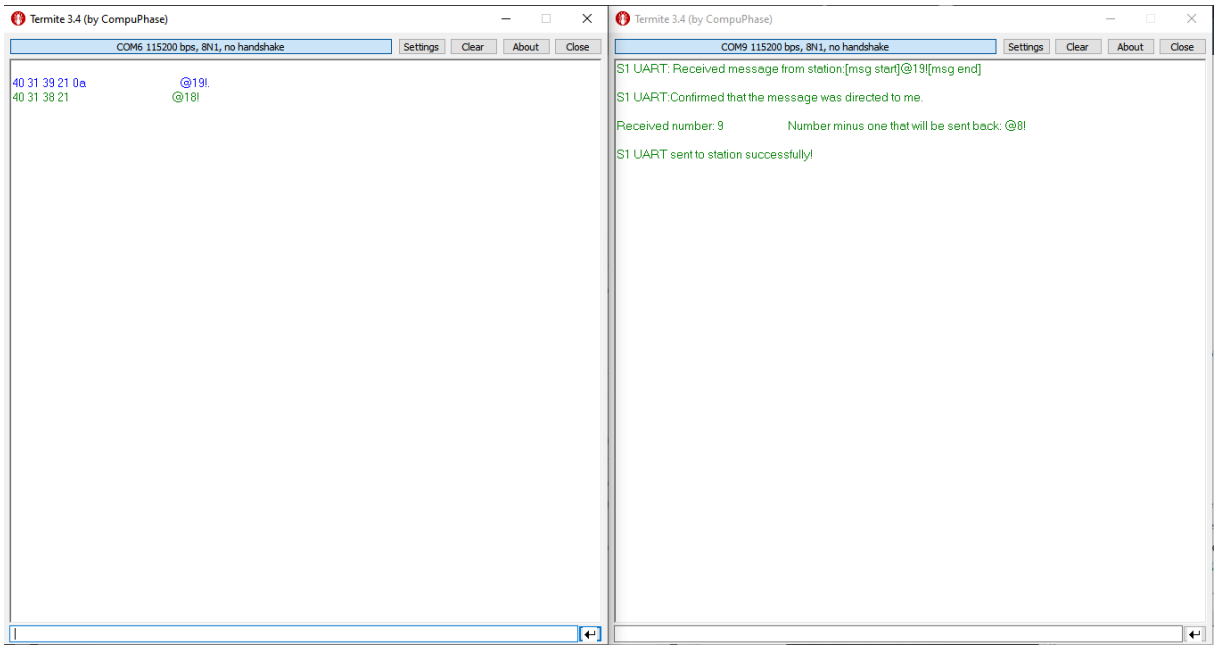


Figure M.13: FTDI serial port (left) sending the number 9 to the subject, and the subject responding with the number 8, as its serial port on the right also indicates.

M.3.4.3 Test station sending and subject responding

Knowing that the subject's UART worked as designed and that the AND gates worked with external pull-up resistors, SMD 1206 versions of these pull-up resistors were soldered to the bottom of the board. The 3.3 V source came from one of the pins on the BOOT0 switch. The modification can be seen in Appendix H.1.

The test station was able to transmit UART data to a subject and receive the subtracted number. In this way, it could verify that the peripheral was operational and it could be reported to the PC. This can be seen in Figure M.14, where COM5 is the test station and COM9 subject 1. However, when more than one subject was powered, the UART verification did not work for any of the subjects. The cause for this could unfortunately not be determined.

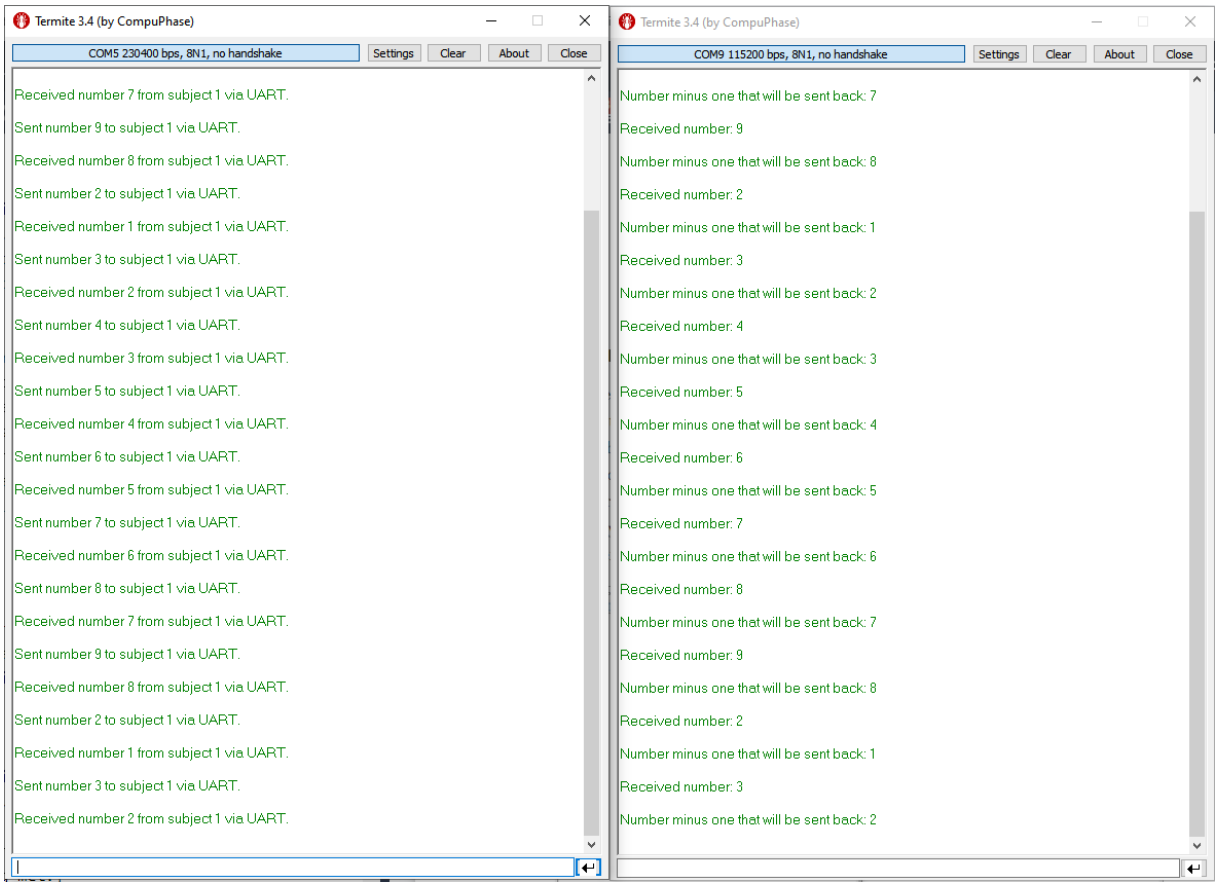
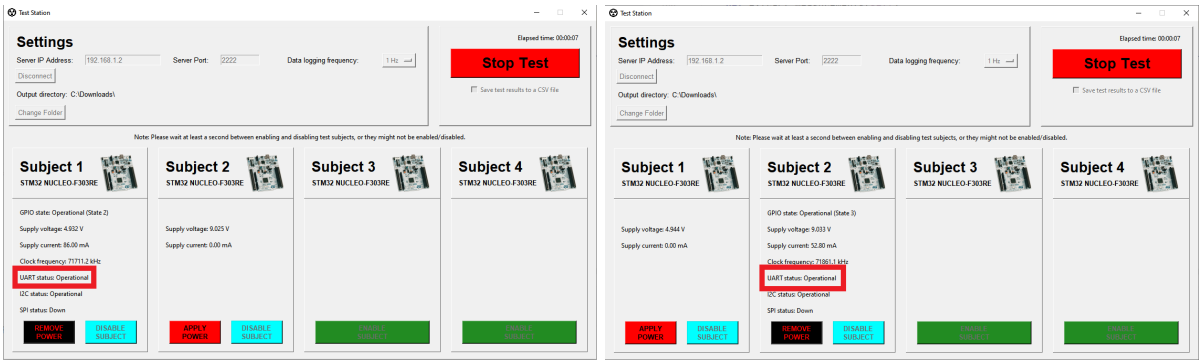


Figure M.14: Terminal program showing working UART transmission and reception for one subject.

Figure M.15 shows the verification working for different subjects on the GUI. Note that this is an earlier version of the GUI that did not yet include the graphs and miniature terminal boxes. The full GUI results can be seen in Section 5.5.



(a) Subject 1 UART operational.

(b) Subject 2 UART operational.

Figure M.15: GUI screenshot showing that only one subject's UART could be verified at a time.

M.3.5 I2C Verificaton

This section is the detailed vesion of the results in Section 5.3.5.

M.3.5.1 Test station sending and subject responding

Figure M.16 shows a terminal program (Termite), where the test station (COM5) sends a number to the first subject (COM9), which squares the number and sends it back. The test station then prints the received number to the terminal. The *1* in front of the result is to indicate which subject it is talking to. Clearly, the I2C did not need the more complicated start- and end characters as the UART did.

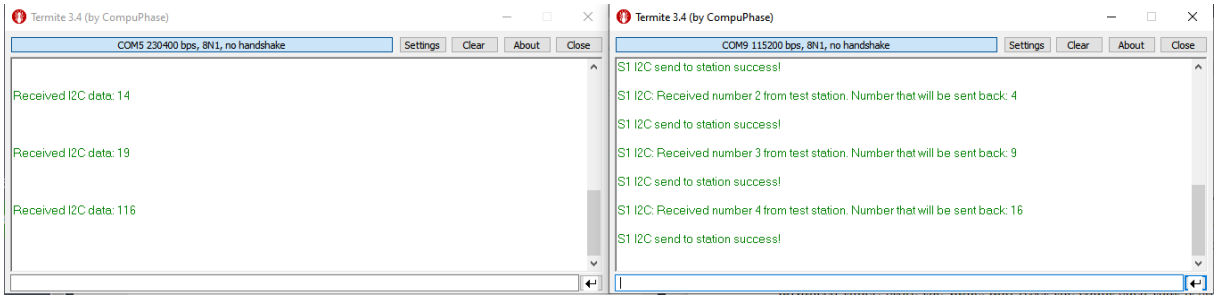


Figure M.16: Terminal program showing the test station (left) receiving the square of the number it sent to subject 1 from that subject (right).

Figure M.17 shows the I2C signals (SDA, SCL) measured on the connection PCB of the subject. Clearly the pull-up resistors chosen in Section 4.2.4 yield adequate signals. The reason the signals are not in a logical high state before changing is that this specific message is part of a sequential transmission, as opposed to the UART, where each byte was transmitted with a separate function call. Finally, the frequency of 95 kHz is close enough to the default 100 kHz set up in the device configuration tool.

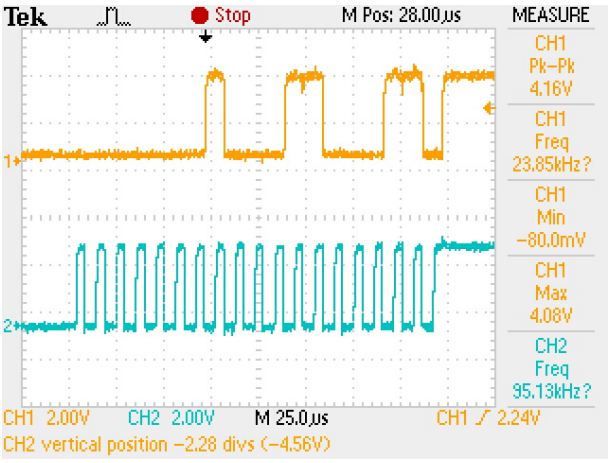
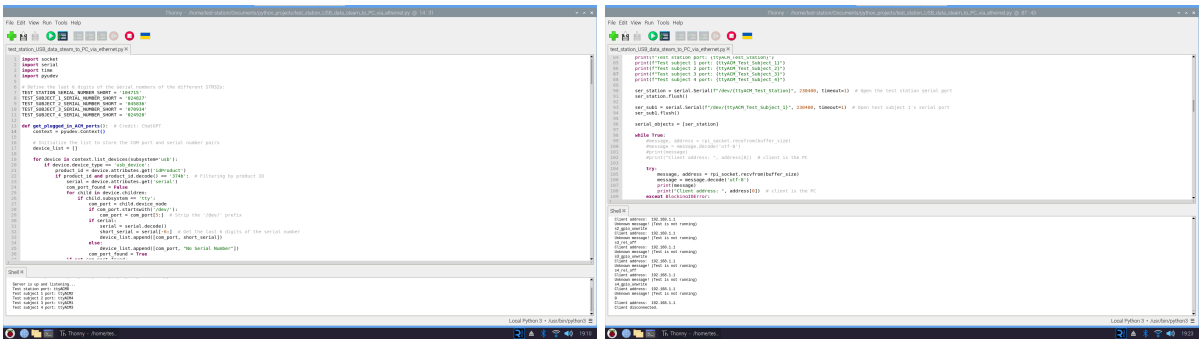


Figure M.17: Test subject 1's I2C SDA (top) and SCL (bottom).

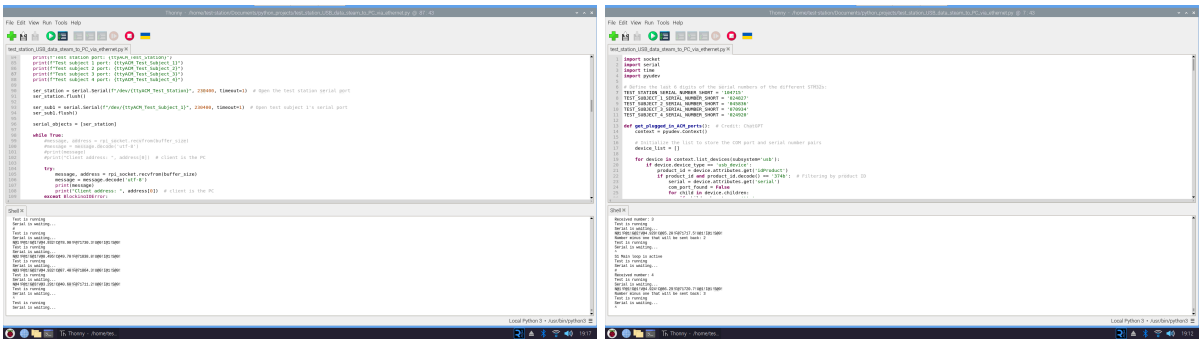
M.4 Raspberry Pi

This section expands on the Raspberry Pi results of Section 5.4. Figure M.18a shows the Raspberry Pi's editor and console, which shows that all the plugged in test subjects are recognised based on their serial numbers, and Figure M.18b shows that the script can receive commands from the GUI. Figure M.19a shows all four subjects' readings in the console, and Figure M.19b shows how one subject's readings are interwoven with the data coming from its serial port directly, without passing through the test station first. Although it looks like it is working, having a subject reporting its readings directly significantly slowed down the GUI, which barely updated its readings for an undetermined reason. It therefore had the exact opposite effect of what it was intended to do in Section 1.3.1, so the readings were instead directly reported from the station. The efficient communication format of Section 4.3.8.2 meant that direct subject readings would not even theoretically improve the bottleneck significantly. Figure M.20 shows how test subject 1 is being programmed via VirtualHere.



(a) All devices recognised. (b) Receiving commands.

Figure M.18: Thonny editor showing devices being recognised and GUI commands processed.



(a) Readings of four subjects. (b) Subject 1's readings and serial port data.

Figure M.19: Thonny editor showing readings of four subjects and serial port of a subject.

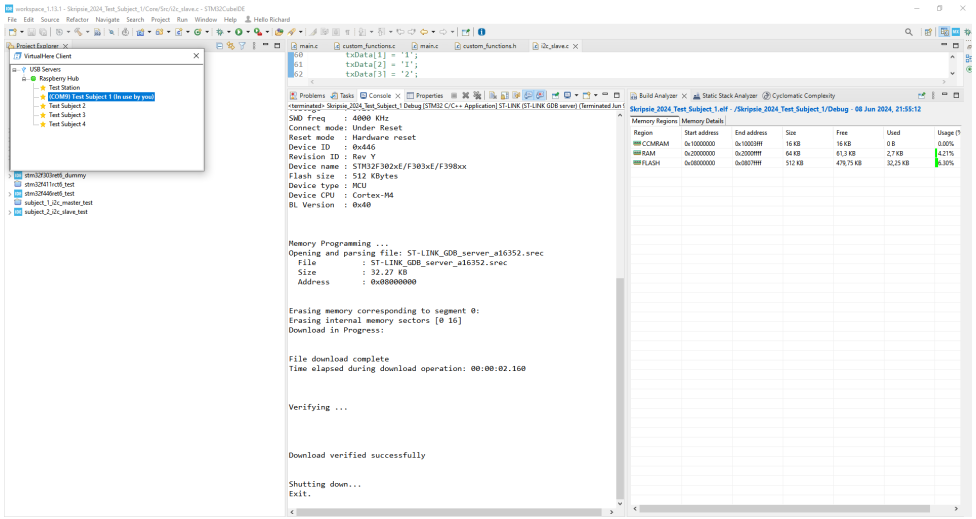
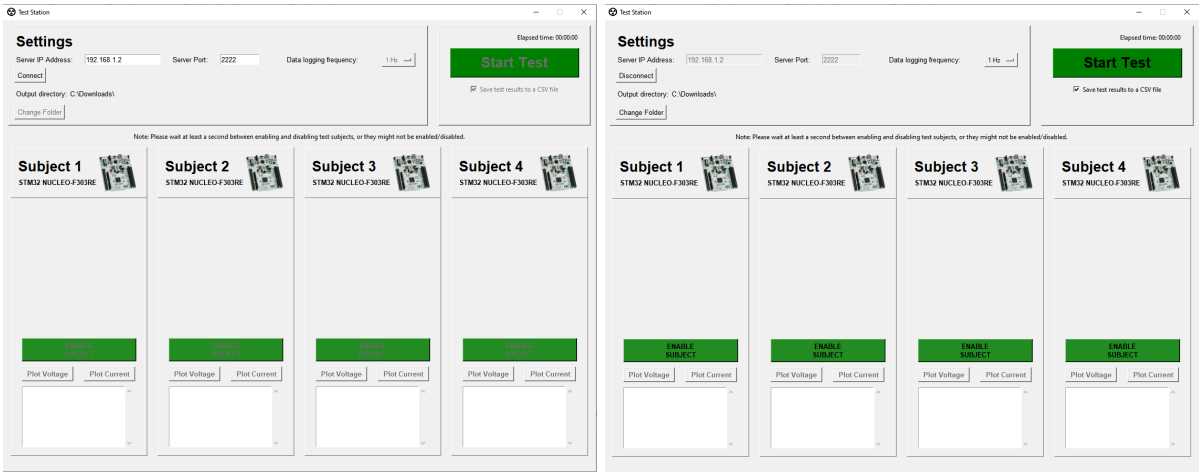


Figure M.20: Test subject 1 being programmed via VirtualHere.

M.5 Graphical User Interface

This section includes screenshots of the GUI results, as referred to in Section 5.5.

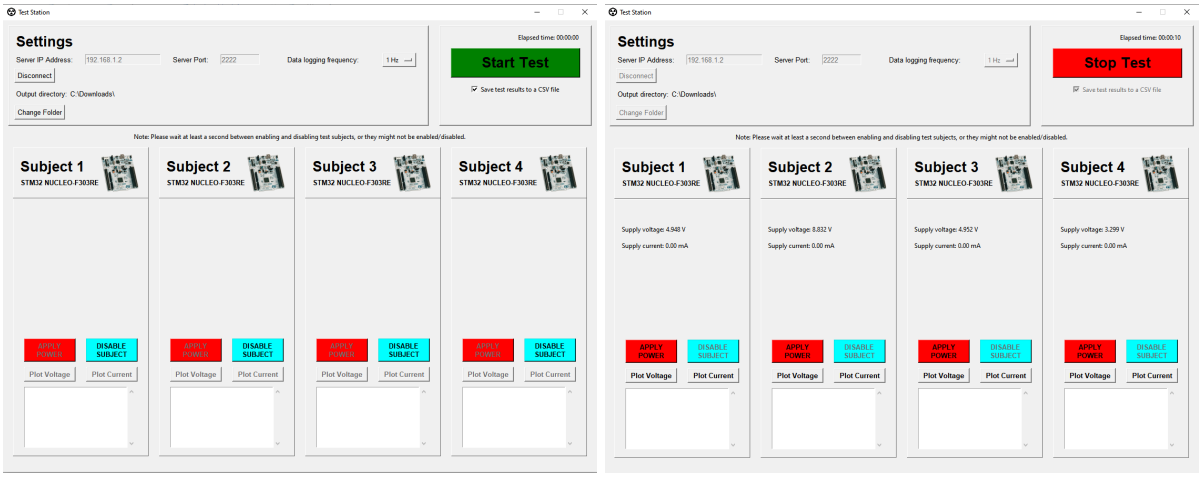
M.5.0.1 Connecting to and disconnecting from the server



(a) Disconnected (b) Connected

Figure M.21: GUI when client is connected and disconnected.

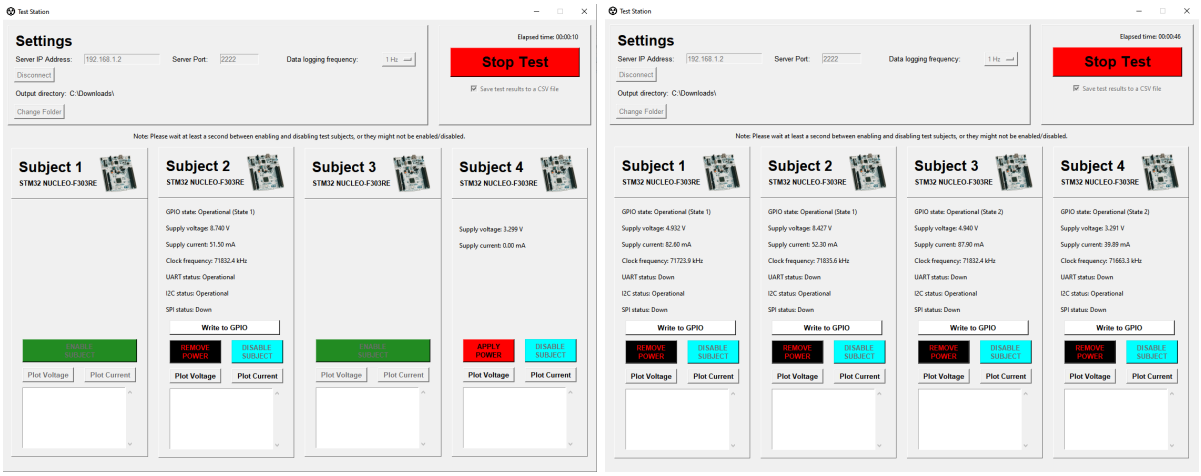
M.5.0.2 Enabling subjects and starting the test



(a) Subjects enabled, test not running. (b) Subjects enabled, test running.

Figure M.22: GUI when all subjects are enabled, and the test is running vs not running.

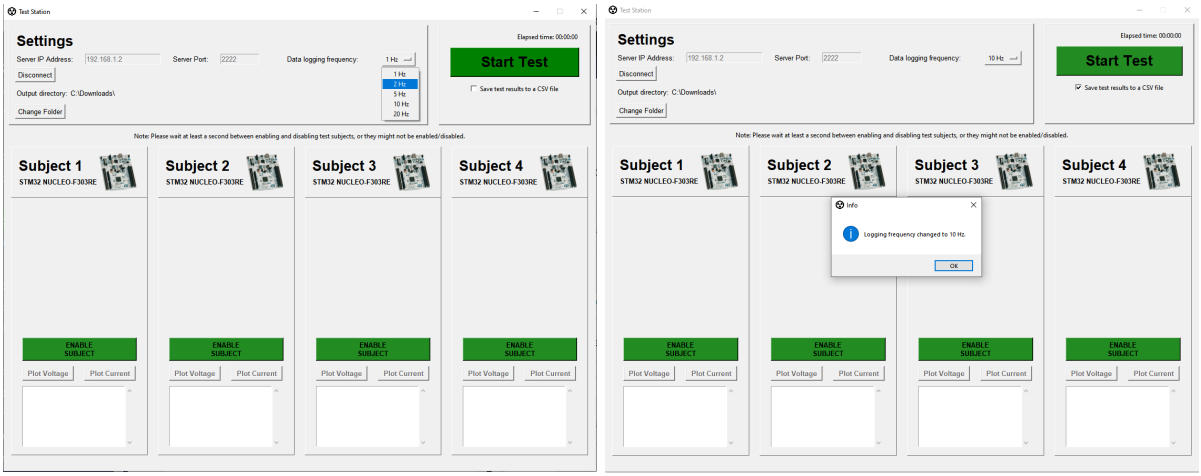
M.5.0.3 Powering subjects



(a) Some subjects enabled, some powered. (b) All subjects enabled and powered.

Figure M.23: GUI when some vs all subjects are powered.

M.5.0.4 Changing the data rate

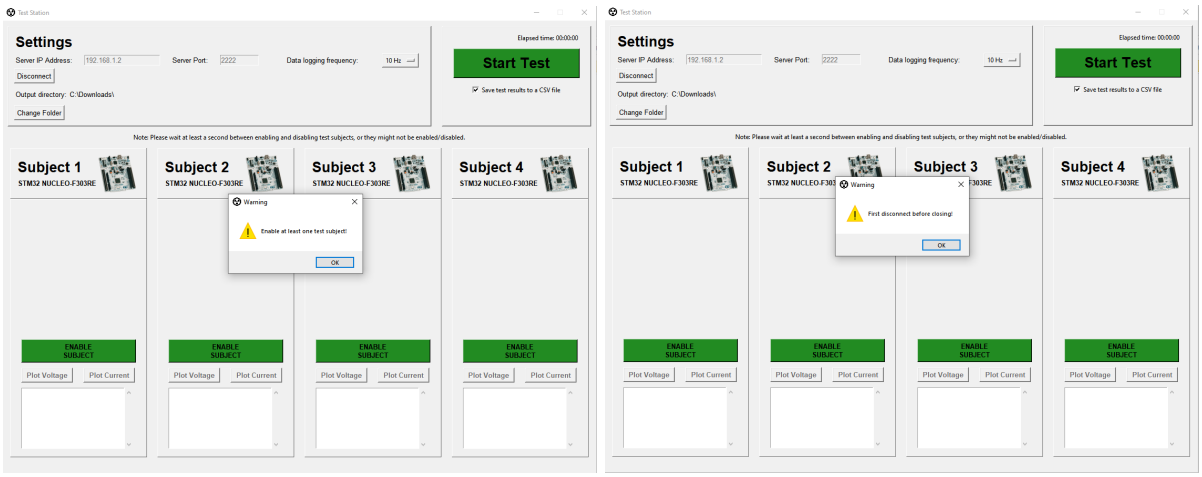


(a) Drop-down menu to select rates. (b) Message to inform the user.

Figure M.24: GUI showing that different data logging rates can be selected.

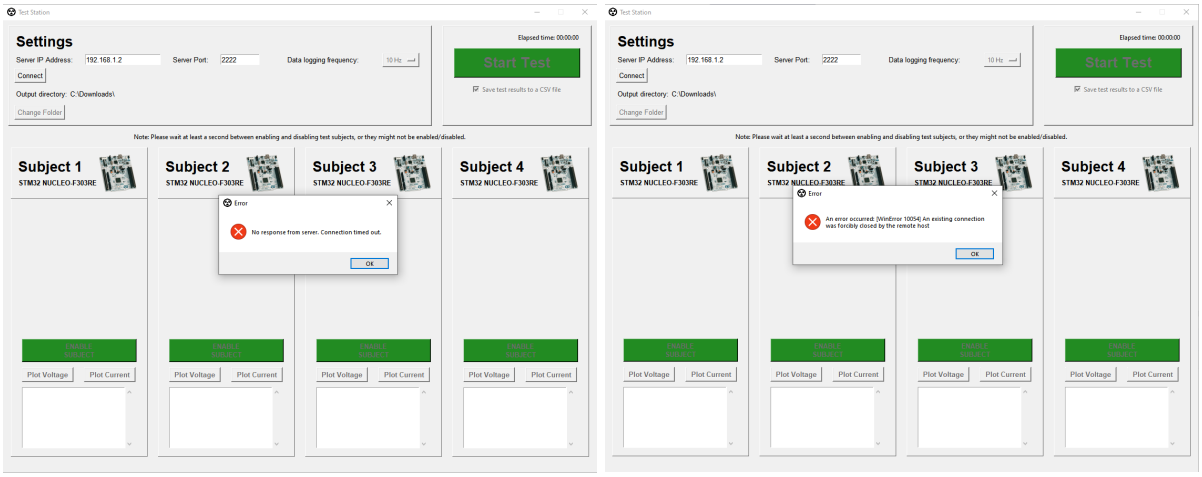
M.5.0.5 Error and warning messages

To make the GUI robust, error messages were implemented to prevent the program from crashing, and to provide a more user-friendly experience.



(a) Starting test before enabling subjects. (b) Closing before disconnecting.

Figure M.25: GUI showing warning messages to inform the user.

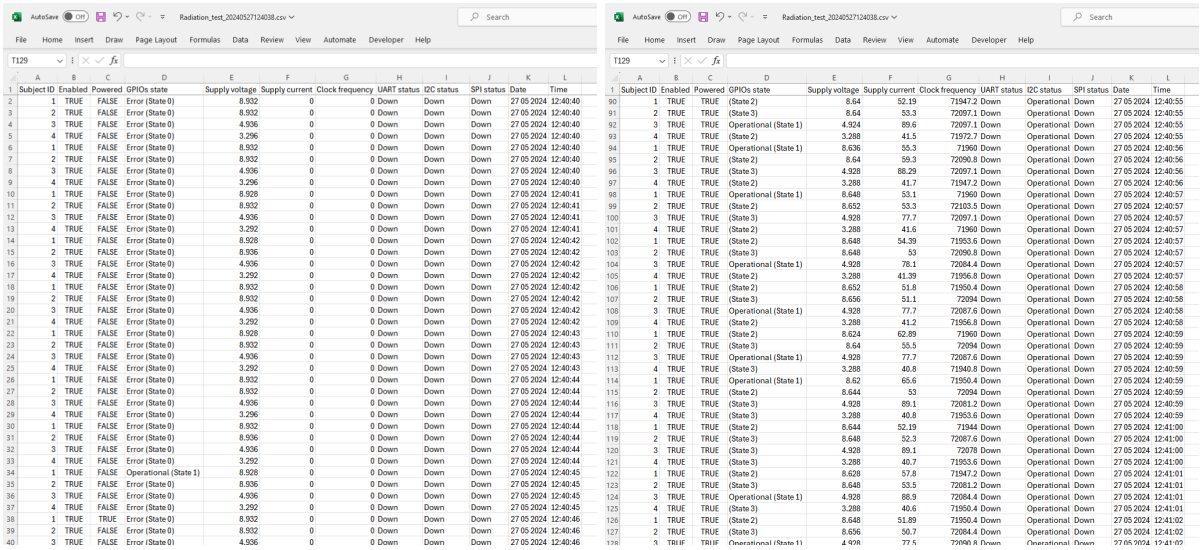


(a) When Pi is not powered or connected. (b) When server script is not running.

Figure M.26: GUI showing error messages to the user.

M.5.0.6 Storing to a CSV file

Figure M.27 shows screenshots of CSV files created by the GUI, as designed in Section 4.5.2.1.



Subject ID	Enabled	Powered	GPIOs state	Supply voltage	Supply current	Clock frequency	UART status	I2C status	SPI status	Date	Time
1	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:40
2	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:40
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:40
4	TRUE	FALSE	Error (State 0)	3.296	0	0	Down	Down	Down	27 05 2024	12:40:40
1	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:40
2	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:40
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:40
4	TRUE	FALSE	Error (State 0)	3.296	0	0	Down	Down	Down	27 05 2024	12:40:40
1	TRUE	FALSE	Error (State 0)	8.928	0	0	Down	Down	Down	27 05 2024	12:40:41
2	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:41
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:41
4	TRUE	FALSE	Error (State 0)	3.292	0	0	Down	Down	Down	27 05 2024	12:40:41
1	TRUE	FALSE	Error (State 0)	8.928	0	0	Down	Down	Down	27 05 2024	12:40:42
2	TRUE	FALSE	Error (State 0)	8.936	0	0	Down	Down	Down	27 05 2024	12:40:42
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:42
4	TRUE	FALSE	Error (State 0)	3.292	0	0	Down	Down	Down	27 05 2024	12:40:42
1	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:42
2	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:42
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:42
4	TRUE	FALSE	Error (State 0)	3.292	0	0	Down	Down	Down	27 05 2024	12:40:42
1	TRUE	FALSE	Error (State 0)	8.928	0	0	Down	Down	Down	27 05 2024	12:40:43
2	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:43
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:43
4	TRUE	FALSE	Error (State 0)	3.292	0	0	Down	Down	Down	27 05 2024	12:40:43
1	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:44
2	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:44
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:44
4	TRUE	FALSE	Error (State 0)	3.296	0	0	Down	Down	Down	27 05 2024	12:40:44
1	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:44
2	TRUE	FALSE	Error (State 0)	8.936	0	0	Down	Down	Down	27 05 2024	12:40:44
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:44
4	TRUE	FALSE	Error (State 0)	3.292	0	0	Down	Down	Down	27 05 2024	12:40:44
1	TRUE	FALSE	Operational (State 1)	8.928	0	0	Down	Down	Down	27 05 2024	12:40:45
2	TRUE	FALSE	Error (State 0)	8.936	0	0	Down	Down	Down	27 05 2024	12:40:45
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:45
4	TRUE	FALSE	Error (State 0)	3.292	0	0	Down	Down	Down	27 05 2024	12:40:45
1	TRUE	TRUE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:46
2	TRUE	FALSE	Error (State 0)	8.932	0	0	Down	Down	Down	27 05 2024	12:40:46
3	TRUE	FALSE	Error (State 0)	4.936	0	0	Down	Down	Down	27 05 2024	12:40:46

(a) Subjects not yet powered.

Subject ID	Enabled	Powered	GPIOs state	Supply voltage	Supply current	Clock frequency	UART status	I2C status	SPI status	Date	Time
1	TRUE	TRUE	(State 2)	8.64	52.19	71947.2	Down	Operational	Down	27 05 2024	12:40:55
2	TRUE	TRUE	(State 3)	8.64	53.3	72097.1	Down	Operational	Down	27 05 2024	12:40:55
3	TRUE	TRUE	Operational (State 1)	4.924	89.6	72097.1	Down	Operational	Down	27 05 2024	12:40:55
4	TRUE	TRUE	(State 2)	3.288	41.5	71972.7	Down	Operational	Down	27 05 2024	12:40:55
1	TRUE	TRUE	Operational (State 1)	8.636	55.3	71960	Down	Operational	Down	27 05 2024	12:40:56
2	TRUE	TRUE	(State 2)	8.64	59.3	72090.8	Down	Operational	Down	27 05 2024	12:40:56
3	TRUE	TRUE	(State 3)	4.928	88.29	72097.1	Down	Operational	Down	27 05 2024	12:40:56
4	TRUE	TRUE	(State 2)	3.288	41.7	71947.2	Down	Operational	Down	27 05 2024	12:40:56
1	TRUE	TRUE	Operational (State 1)	8.648	53.1	71960	Down	Operational	Down	27 05 2024	12:40:57
2	TRUE	TRUE	(State 2)	8.652	53.3	72103.5	Down	Operational	Down	27 05 2024	12:40:57
3	TRUE	TRUE	(State 3)	4.928	77.7	72097.1	Down	Operational	Down	27 05 2024	12:40:57
4	TRUE	TRUE	(State 2)	3.288	41.6	71960	Down	Operational	Down	27 05 2024	12:40:57
1	TRUE	TRUE	(State 2)	8.648	54.39	71953.6	Down	Operational	Down	27 05 2024	12:40:57
2	TRUE	TRUE	(State 3)	8.648	53	72090.8	Down	Operational	Down	27 05 2024	12:40:57
3	TRUE	TRUE	Operational (State 1)	4.928	78.1	72084.4	Down	Operational	Down	27 05 2024	12:40:57
4	TRUE	TRUE	(State 2)	3.288	41.39	71956.8	Down	Operational	Down	27 05 2024	12:40:57
1	TRUE	TRUE	(State 2)	8.652	51.8	71950.4	Down	Operational	Down	27 05 2024	12:40:58
2	TRUE	TRUE	(State 3)	8.656	51.1	72094	Down	Operational	Down	27 05 2024	12:40:58
3	TRUE	TRUE	Operational (State 1)	4.928	77.7	72087.6	Down	Operational	Down	27 05 2024	12:40:58
4	TRUE	TRUE	(State 2)	3.288	41.2	71956.8	Down	Operational	Down	27 05 2024	12:40:58
1	TRUE	TRUE	(State 2)	8.624	62.89	71960	Down	Operational	Down	27 05 2024	12:40:59
2	TRUE	TRUE	(State 3)	8.64	55.5	72094	Down	Operational	Down	27 05 2024	12:40:59
3	TRUE	TRUE	Operational (State 1)	4.928	77.7	72087.6	Down	Operational	Down	27 05 2024	12:40:59
4	TRUE	TRUE	(State 3)	3.288	40.8	71940.8	Down	Operational	Down	27 05 2024	12:40:59
1	TRUE	TRUE	Operational (State 1)	8.62	65.6	71950.4	Down	Operational	Down	27 05 2024	12:40:59
2	TRUE	TRUE	(State 2)	8.644	53	72094	Down	Operational	Down	27 05 2024	12:40:59
3	TRUE	TRUE	(State 3)	4.928	89.1	72081.2	Down	Operational	Down	27 05 2024	12:40:59
4	TRUE	TRUE	(State 3)	3.288	40.8	71953.6	Down	Operational	Down	27 05 2024	12:40:59
1	TRUE	TRUE	(State 2)	8.644	52.19	71944	Down	Operational	Down	27 05 2024	12:41:00
2	TRUE	TRUE	(State 3)	8.648	52.3	72087.6	Down	Operational	Down	27 05 2024	12:41:00
3	TRUE	TRUE	(State 3)	4.928	89.1	72078	Down	Operational	Down	27 05 2024	12:41:00
4	TRUE	TRUE	(State 3)	3.288	40.7	71953.6	Down	Operational	Down	27 05 2024	12:41:00
1	TRUE	TRUE	(State 2)	8.628	57.8	71947.2	Down	Operational	Down	27 05 2024	12:41:01
2	TRUE	TRUE	(State 3)	8.648	53.5	72081.2	Down	Operational	Down	27 05 2024	12:41:01
3	TRUE	TRUE	Operational (State 1)	4.928	88.9	72084.4	Down	Operational	Down	27 05 2024	12:41:01
4	TRUE	TRUE	(State 3)	3.288	40.6	71950.4	Down	Operational	Down	27 05 2024	12:41:01
1	TRUE	TRUE	(State 2)	8.648	51.89	71950.4	Down	Operational	Down	27 05 2024	12:41:02
2	TRUE	TRUE	(State 3)	8.656	50.7	72084.4	Down	Operational	Down	27 05 2024	12:41:02

(b) All subjects powered.

Figure M.27: CSV file generated by the GUI if the user did not deselect the check box.

M.5.0.7 Live graphs of voltage and current

Figures M.28 and M.29 show screenshots of the GUI together with the graphs for four subjects, as referred to in the results of Section 5.5, and the detailed design of Section 4.5. The design for these graphs are explained in Section 4.5.2.2. As expected, the voltage drops as power is applied to the subjects.

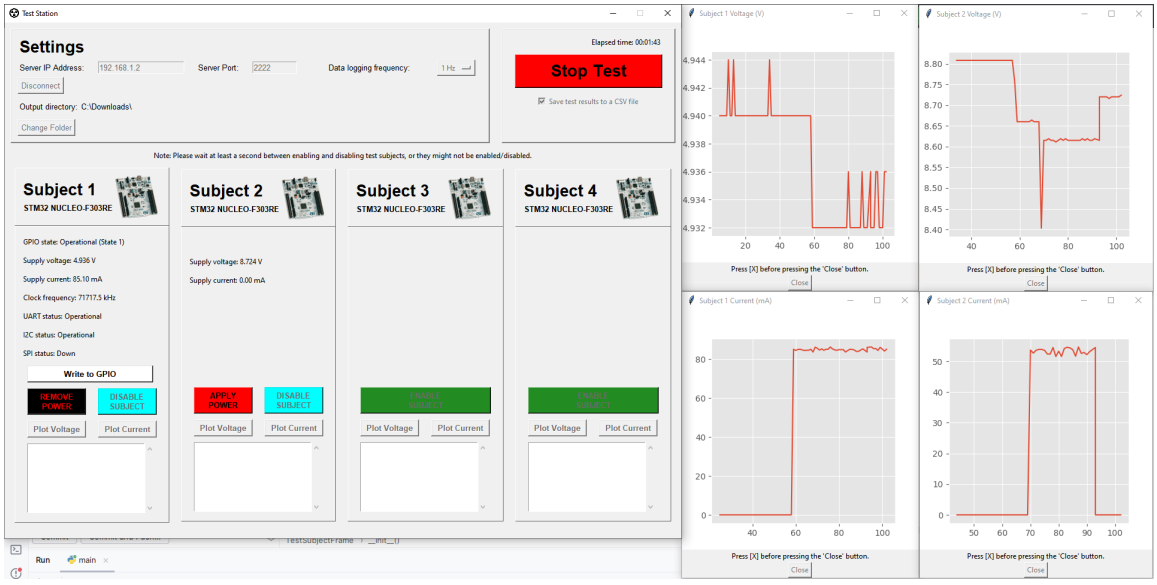


Figure M.28: Voltage and current graphs of subjects 1 and 2.

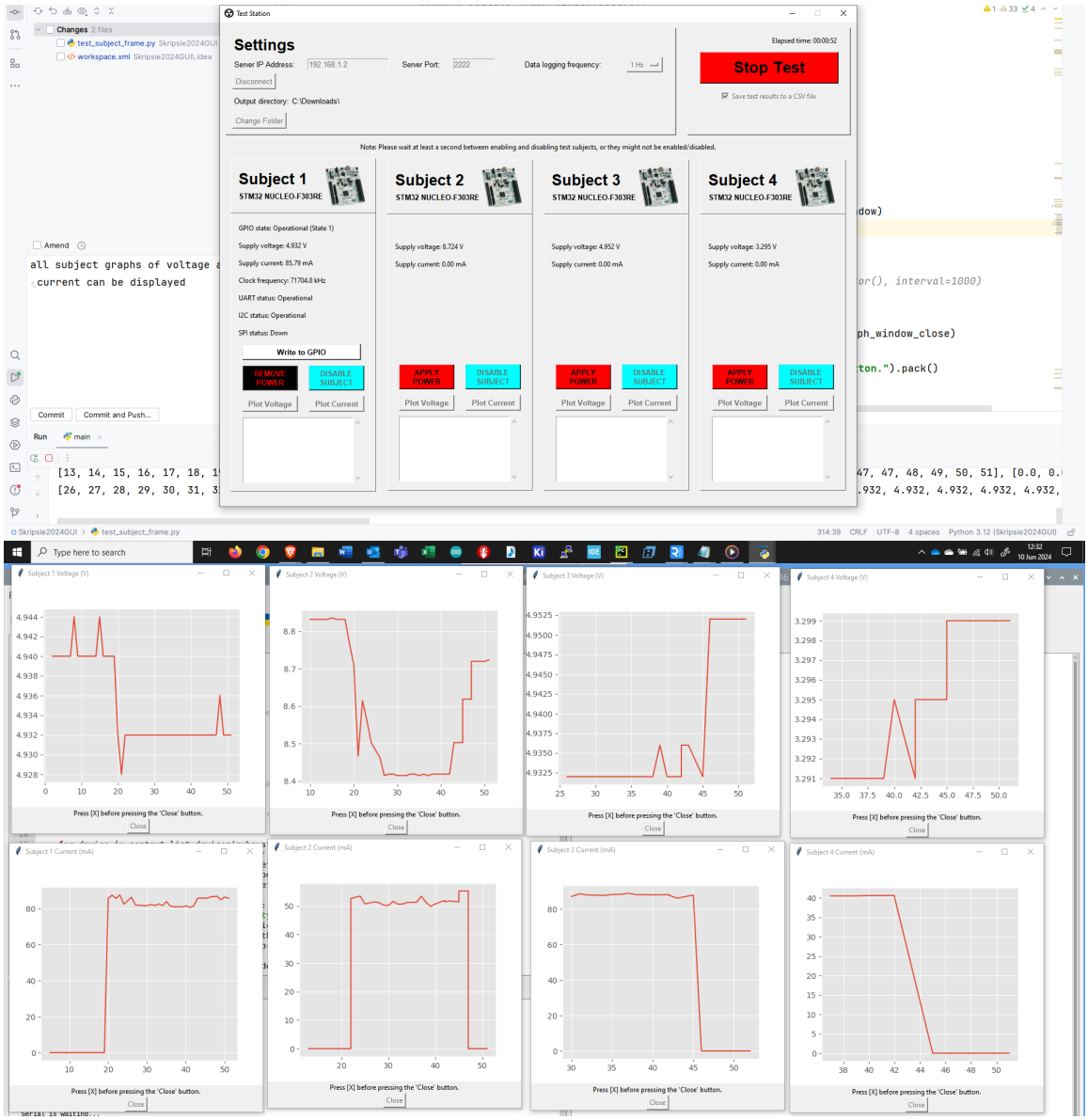


Figure M.29: Voltage and current graphs of all four subjects.

Appendix N

Setups for Test Results

This appendix contains all the test setups that was used to obtain the results in Chapter 5. If a particular setting is not mentioned, it does not matter to what it is set.

N.1 Test Station Module Hardware Tests

Figure N.1 shows a pinout of a module, which will be referred to in this section. Section 4.1.1.5 includes a detailed explanation of the power supply terminology.

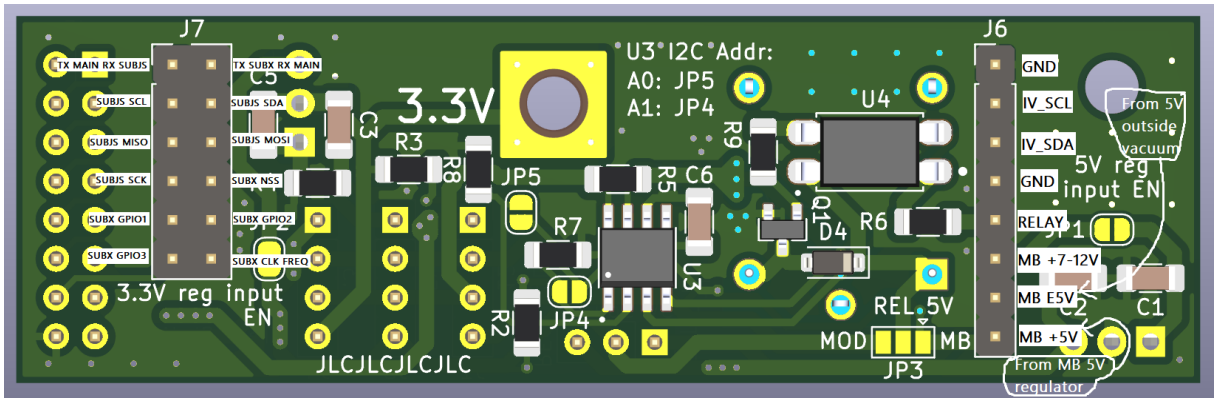


Figure N.1: A pinout of the underside of a module.

N.1.1 Relay Driver Continuity Tests

N.1.1.1 Relay supplied with the main board's 5V regulator pin

The following setup was used for the relay driver test mentioned in Section M.1.1.1:

- Solder bridge JP3 (relay 5V) on MB (main board) of the module
- External 5V supply to the *MB_+5V* pin on the module
- Module jumper J1 (7-12V) selected
- Connection PCB connected to module with a ribbon cable
- Multimeter's one probe connected to the *MB_+7-12V* pin of the module
- Multimeter's other probe connected to the *VIN* pad of the connection PCB
- Multimeter set to continuity mode
- External 3.3V supply to the *RELAY* pin of the module

N.1.1.2 Relay supplied with the main board's 5V regulator pin

The following setup was used for the relay driver test mentioned in Section M.1.1.2:

- Solder bridge JP3 (relay 5V) on MB (main board) of the module
- External 5V supply to the *MB_E5V* pin on the module
- Module jumper J2 (5V source) on *EXT*
- Module jumper J1 (7-12V) selected
- Connection PCB connected to module with a ribbon cable
- Multimeter's one probe connected to the *MB_+7-12V* pin of the module
- Multimeter's other probe connected to the *VIN* pad of the connection PCB
- Multimeter set to continuity mode
- External 3.3V supply to the *RELAY* pin of the module

N.1.1.3 Relay supplied with the module's 5V regulator

The following setup was used for the relay driver test mentioned in Section M.1.1.3:

- Solder bridge JP3 (relay 5V) on MOD (module) of the module
- External 12V supply to the *MB_+7-12V* pin on the module
- Module jumper J2 (5V source) on *REG*
- Module jumper J1 (7-12V) selected
- Connection PCB connected to module with a ribbon cable
- Multimeter's negative probe connected to any one of the two *GND* pins of the module
- Multimeter's positive probe connected to the *VIN* pad of the connection PCB
- Multimeter set to DC voltage mode
- External 3.3V supply to the *RELAY* pin of the module

N.1.2 Configurable Power Supply Test

N.1.2.1 Module regulator 3.3V supply

The following setup was used for the configurable power supply test mentioned in Appendix M.1.3.1:

- Solder bridge JP3 (relay 5V) of the module on MB (main board)
- External 5V supply to the *MB_+5V* pin on the module

- A second external 5V supply to the *MB_E5V* pin on the module
- External 3.3V supply to the *RELAY* pin of the module
- Module jumper J2 (5V source) on *EXT*
- Module jumper J4 (3.3V) selected
- Connection PCB connected to module with a ribbon cable
- 8 Ω resistor connected to GND and 3V3 of connection PCB
- Multimeter's negative probe connected to any one of the two *GND* pins of the module
- Multimeter's positive probe connected to the *3V3* pad of the connection PCB
- Multimeter set to DC voltage mode
- Test subject JP5 removed
- Test subject SB2 and SB12 removed
- Test subject plugged into connection PCB
- Module 5V reg input EN solder bridge soldered
- Module 3.3V reg input EN solder bridge soldered

N.1.2.2 Module regulator 5V supply

The following setup was used for the configurable power supply test mentioned in Appendix M.1.3.2:

- Solder bridge JP3 (relay 5V) of the module on MB (main board)
- External 7V/12V supply to the *MB_+7-12V* pin on the module
- External 5V supply to the *MB_+5V* pin of the module
- External 3.3V supply to the *RELAY* pin of the module
- Module jumper J2 (5V source) on *REG*
- Module jumper J3 (5V) selected
- Connection PCB connected to module with a ribbon cable
- 8 Ω/25 Ω resistor connected to GND and E5V of connection PCB
- Multimeter's negative probe connected to any one of the two *GND* pins of the module
- Multimeter's positive probe connected to the *E5V* pad of the connection PCB
- Multimeter set to DC voltage mode
- Test subject plugged into connection PCB
- Test subject JP5 set to E5V
- Module 5V reg input EN solder bridge soldered

N.1.3 Current- and Voltage Sensor

The following setup was used for the current- and voltage sensor test mentioned in Appendix 5.1.2:

- Solder bridge JP3 (relay 5V) of the module on MB (main board)
- External 7/12V supply to the *MB_+7-12V* pin on the module
- External 5V supply to the *MB_+5V* pin of the module
- External 3.3V supply to the *RELAY* pin of the module
- Module jumper J1 (7-12V) selected
- Connection PCB connected to module with a ribbon cable
- Multimeter's negative probe connected to any one of the two *GND* pins of the module
- Multimeter's positive probe connected to the *VIN* pad of the connection PCB
- Multimeter set to DC voltage mode
- Test subject plugged into connection PCB
- Test subject JP5 set to E5V
- Module JP5, JP4 (A0, A1) not soldered
- Test station SDA to module IV_SDA pin
- Test station SCL to module IV_SCL pin
- Calibration set to 16V and 300 mA (refer to Section 4.3.2 for details).
- Test station: Module's I2C address set to 0x40

Appendix O

Test Station Pin Assignments

The pin assignments of the main board MCU mentioned in Section 4.2.2.5 can be seen in Table O. The pins are valid for both the main MCU (STM32F303RET6) and the backup MCU (STM32F446RET6). Pins marked *N/A* can be used as desired. Some pins should not be used since they are not routed to a pad on the PCB due to space constraints caused by the ground rings of the oscillators.

Pin name	Peripheral	Name on schematic
PA0	TIM8_ETR	SUBJ4_CLK_INPUT
PA1	GPIO_Output	SUB4_GPIO_3
PA2	USART2_TX	TX_MAIN_RX_STLINK
PA3	USART2_RX	RX_MAIN_TX_STLINK
PA4	Gpio_Input	SUB4_GPIO_2
PA5	TIM2_ETR	SUB2_CLK_FREQ
PA6	GPIO_Input	SUB4_GPIO_1
PA7	GPIO_Input	SUB3_GPIO_2
PA8	I2C3_SCL	IV_SCL
PA9	USART1_TX	TX_MAIN_RX_SUBJECTS
PA10	USART1_RX	RX_MAIN_TX_SUBJECTS
PA11	GPIO_Output	SUB2_GPIO_3
PA12	TIM1_ETR	SUB1_CLK_FREQ
PA13	SYS_JTMS-SWDIO	TMS
PA14	SYS_JTCK-SWCLK	TCK
PA15	SPI3_NSS	SPI3_NSS
PB0	GPIO_Output	SUB4_RELAY
PB1	GPIO_Output	SUB3_RELAY
PB2	GPIO_Output	SUB2_RELAY
PB3	SYS_JTDO-TRACESWO	SWO
PB4	N/A	N/A
PB5	GPIO_Output	SUB2_NSS
PB6	GPIO_Output	SUB3_NSS
PB7	GPIO_Output	SUB4_NSS
PB8	I2C1_SCL	SUBJECTS_SCL
PB9	I2C1_SDA	SUBJECTS_SDA
PB10	GPIO_Output	SUB1_RELAY
PB11/VCAP_1	4.7u to GND for 446	N/A
PB12	GPIO_Output	LED_DEBUG
PB13	GPIO_Output	SUB1_GPIO3
PB14	GPIO_Input	SUB1_GPIO2
PB15	GPIO_Input	SUB1_GPIO1
PC0	Do not use	Do not use
PC1	Do not use	Do not use
PC2	Do not use	Do not use
PC3	Do not use	Do not use
PC4	GPIO_Output	SUB3_GPIO3
PC5	N/A	N/A
PC6	GPIO_Input	SUB2_GPIO1
PC7	GPIO_Input	SUB2_GPIO2
PC8	GPIO_Output	SUB1_NSS
PC9	I2C3_SDA	IV_SENSORS_SDA
PC10	SPI3_SCK	SUBJECTS_SCK
PC11	SPI3_MISO	SUBJECTS_MISO
PC12	SPI3_MOSI	SUBJECTS_MOSI
PC13	GPIO_Input	SUB3_GPIO1
PC14	RCC_OSC32_IN	LSE_IN
PC15	RCC_OSC32_OUT	LSE_OUT
PD2	TIM3_ETR	SUB3_CLK_FREQ
PH0/PF0	RCC_OSC_IN	HSE_IN
PH1/PF1	RCC_OSC_OUT	HSE_OUT

Table O.1: Detailed pin assignments of the main board's MCU.

Appendix P

PC Setup for Raspberry Pi Ethernet

To connect to the Pi using RealVNC (as mentioned in Section 4.4), the IP address of the server (Pi) had to be entered into the client software. This IP address of the Pi is automatically assigned when the connection is a WiFi network, but it has to be assigned manually when using Ethernet. An online tutorial [77] was used to do this, and the steps are summarised below. To see the IP address on the Pi, one can type `ifconfig` into the terminal, and under `eth0`, the assigned IP address should be next to `inet`. If wanting to use a Wi-Fi network instead, one can use the IP address under `wlan0`.

1. Plug the Pi into the PC with an Ethernet cable.
2. On the PC, open *Network and Sharing Center » Change adapter settings » Ethernet » Properties » Internet Protocol Version 4 (TCP/IPv4) » Properties*
3. Select *Use the following IP address* and enter any IP address like 192.168.1.1, and press *OK » Close » Close*.
4. Download the latest DHCP server from *http://www.dhcpserver.de/cms/download/*.
5. Extract all and run *dhcpwiz.exe*
6. Go to *Ethernet » Next » Next » Next*, select *Overwrite existing file*, and press *Write INI file*.
7. Select *Next* and select *Run DHCP server immediately*, click *Admin*, click *Install*, then *Configure*, then *Start*
8. Click *Exit » Finish » Continue as tray app. » Yes*
9. A popup message should appear in a few moments showing the new IP address of the Pi. This can be used as explained in Section 4.4.

Appendix Q

Function Definitions

Q.1 STM32 Function definitions

Q.1.1 Test Station UART Receive from Subjects

As mentioned in Section 4.3.6, the contents of the callback function with name `HAL_UART_RxCpltCallback`, and that is used to receive data from the subjects, can be seen below.

```
if (huart == huartSubjects) {
    rxCharUartSubjects = (char)(rxByteUartSubjects[0]);
    if (rxBusyUartSubjects) {
        rxLenUartSubjects += 1;
        rxDataUartSubjects[rxLenUartSubjects - 1] = rxCharUartSubjects;
        if (rxByteUartSubjects[0] == '!') {
            rxBusyUartSubjects = 0;
            processUartSubjectsRxMsg();
            rxLenUartSubjects = 0;
        }
    } else {
        if (rxCharUartSubjects == '@') {
            rxBusyUartSubjects = 1;
            rxLenUartSubjects = 1;
            rxDataUartSubjects[rxLenUartSubjects - 1] =
   rxCharUartSubjects;
        }
    }
    HAL_UART_Receive_IT(huartSubjects, rxByteUartSubjects, 1); // re-
   prime the receiver for the next byte
```

Q.1.2 Test Station UART Transmit and Receive with PC

Functions to transmit and receive data via UART over a serial port, which was introduced in Section 4.3.8.1:

```
void consolePrint(const char* format, ...) { // Credit: ChatGPT
    va_list args;
    va_start(args, format);
    txDataLenPC = vsprintf(txDataPC, format, args);
```

```
    va_end(args);
    HAL_UART_Transmit(huartPC, (uint8_t*)txDataPC, txDataLenPC,
    UART_TIMEOUT_MS);
}
```

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart) {
    if (huart == huartPC) {
        rxCharUartPC = (char)(rxByteUartPC[0]);
    if (rxBusyUartPC) {
      rxLenUartPC += 1;
      rxDataUartPC[rxLenUartPC - 1] = rxCharUartPC;
      if (rxByteUartPC[0] == '\n') {
        rxBusyUartPC = 0;
        for (int i = 0; i < MAX_RX_DATA_LEN_UART_PC; ++i) {
            msg[i] = rxDataUartPC[i];
        }
        len = rxLenUartPC;
        pendingUartPCMessageToProcess = 1;
        rxLenUartPC = 0;
      }
    }
    else {
      if (rxCharUartPC == '@') {
        rxBusyUartPC = 1;
        rxLenUartPC = 1;
        rxDataUartPC[rxLenUartPC - 1] = rxCharUartPC;
      }
    }
    HAL_UART_Receive_IT(huartPC, rxByteUartPC, 1);
    }
}
```

Q.1.3 INA219 Memory Read and Write

Functions to read from and write to the 16-bit registers of the INA219, which were introduced in Section 4.3.2.1:

```
char Read16(volatile TestSubject *subject, uint8_t Register, uint16_t*
    result) {
    char functionReturnVal = 'e'; // error by default
    uint8_t Value[2];
    HAL_StatusTypeDef ret = HAL_I2C_Mem_Read(hi2cIVSensors, subject->
    ivSensorI2cAddress, Register, 1, Value, 2, 1000);
    if (ret == HAL_OK) {
        functionReturnVal = 's'; // success
        *result = ((Value[0] << 8) | Value[1]);
    }
```

```
    return functionReturnVal;
}
```

```
char Write16(volatile TestSubject *subject, uint8_t Register, uint16_t
    Value) {
    char functionReturnVal = 'e'; // error by default
    uint8_t addr[2];
    addr[0] = (Value >> 8) & 0xff;  // upper byte
    addr[1] = (Value >> 0) & 0xff; // lower byte
    HAL_StatusTypeDef ret = HAL_I2C_Mem_Write(hi2cIVSensors, subject->
    ivSensorI2cAddress, Register, 1, (uint8_t*)addr, 2, 1000);
    if (ret == HAL_OK) {
        functionReturnVal = 's'; // success
    }
    return functionReturnVal;
}
```

Q.1.4 Clock Frequency Measurement

This is the callback function used by the basic timer to read and reset the counter value of each of the other test subject timers, as explained in Section E.3.2.2:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  // Prevent unused argument(s) compilation warning /
  UNUSED(htim);

  if (htim == htimClkFreq) { // timer used to measure clock frequencies
    for (uint8_t i = 0; i < 4; ++i) {
      if (testSubjects[i]->enabled == 1) {
        uint32_t timCaptureVal = 0;
        timCaptureVal = testSubjects[i]->htim->Instance->CNT;
        testSubjects[i]->htim->Instance->CNT = 0;
        testSubjects[i]->clk_freq_Hz = 0;
        testSubjects[i]->clk_freq_Hz = timCaptureVal * 3190;
      }
    }
    }
}
```

Q.1.5 Test subject I2C code

This is the test subject code used to receive a number from the test station via I2C, to square it, and to send it back to the test station, as explained in Section 4.3.5:

```
#define RX_SIZE 1
#define TX_SIZE 2
uint8_t rxData[RX_SIZE],
```

```
    txData[TX_SIZE];

void HAL_I2C_AddrCallback(I2C_HandleTypeDef *hi2c, uint8_t
    TransferDirection, uint16_t AddrMatchCode)
{
  if (TransferDirection == I2C_DIRECTION_TRANSMIT) { // if the master
    wants to transmit the data
    clearArray(rxData, RX_SIZE);
    HAL_StatusTypeDef retRx = HAL_I2C_Slave_Sequential_Receive_IT(hi2c,
    rxData, RX_SIZE, I2C_FIRST_AND_LAST_FRAME);
    if (retRx != HAL_OK) consolePrint("S1: Failed to receive I2C data
    from the test station!\r\n");
  } else { // if the master requests the data from the slave
    txData[1] = rxData[0]*rxData[0];// Square the received number and
    send it together with the Subject's ID:
    txData[0] = 1; // For Subject 1
    HAL_StatusTypeDef retTx = HAL_I2C_Slave_Seq_Transmit_IT(hi2c, (
    uint8_t*)txData, TX_SIZE, I2C_FIRST_AND_LAST_FRAME);
    if (retTx != HAL_OK) consolePrint("S1: Failed to send I2C data to
    the test station!\n\r");
    else consolePrint("I2C send success!\n\r");
  }
}
```

Q.1.6 Test station UART code

The UART transmit code for the test station, introduced in Section E.3.4.1:

```
for (int i = 0; i < 4; ++i) {
    if ((testSubjects[i]->enabled == 1) && (testSubjects[i]->powerState
    == 1)) {
        uartTxSubjects[0] = '@';
        char charUartNumber;
        switch (commsNums[uartCommsNumsCounter]) {
        case 1:
            charUartNumber = '1';
            break;
        case 2:
            charUartNumber = '2';
            break;
        case 3:
            charUartNumber = '3';
            break;
        case 4:
            charUartNumber = '4';
            break;
        case 5:
```

```
            charUartNumber = '5';
            break;
        case 6:
            charUartNumber = '6';
            break;
        case 7:
            charUartNumber = '7';
            break;
        case 8:
            charUartNumber = '8';
            break;
        case 9:
            charUartNumber = '9';
            break;
        }
        uartTxSubjects[1] = testSubjects[i]->id;
        uartTxSubjects[2] = charUartNumber;
        uartTxSubjects[3] = '!';

        HAL_StatusTypeDef retUartTx = HAL_UART_Transmit(huartSubjects, (uint8_t*)uartTxSubjects, 4, 400);

        switch (retUartTx) {
        case HAL_OK:
            startUartSubjects(testSubjects[i]);
            // Move on to the next "random number" in the list:
            ++uartCommsNumsCounter;
            if (uartCommsNumsCounter > (SUBJECTS_COMMS_NUMS_SIZE - 1)) uartCommsNumsCounter = 0;
            break;
        default:
            break;
        }
    }
}
```

Q.1.7 Test subjects UART code

The UART code for test subject 1, introduced in Section E.3.4.2:

```
void processUartStationMsg(char* msg, uint8_t msgLen) {
  // Check for valid subject, square the result, and send back to the test station:
  if (msg[1] == '1') { // if the message is directed to subject 1
    // Print the number that was received and the message that will be sent back:
    uint8_t recvdNum = (uint8_t)(msg[2] - '0');
```

```
    consolePrint("Received number: %u\n\r", recvdNum);
    uint8_t result = recvdNum-1;
    consolePrint("Number minus one that will be sent back: %u\n\r",
result);

    char startChar = '@', endChar = '!';
    uartTxDataStationByteVersion[0] = (uint8_t)(startChar - '0');
    uartTxDataStationByteVersion[1] = result;
    uartTxDataStationByteVersion[2] = (uint8_t)(endChar - '0');

    char charResult;
    switch (result) {
    case 0:
      charResult = '0';
      break;
    case 1:
      charResult = '1';
      break;
    case 2:
      charResult = '2';
      break;
    case 3:
      charResult = '3';
      break;
    case 4:
      charResult = '4';
      break;
    case 5:
      charResult = '5';
      break;
    case 6:
      charResult = '6';
      break;
    case 7:
      charResult = '7';
      break;
    case 8:
      charResult = '8';
      break;
    }
    uartTxDataStation[0] = '@';
    uartTxDataStation[1] = '1'; // from subject 1
    uartTxDataStation[2] = charResult;
    uartTxDataStation[3] = '!';

    HAL_StatusTypeDef retUartTx = HAL_UART_Transmit(huartStation, (
uint8_t*)uartTxDataStation, 4, 400);
```

```
51
52      if (retUartTx != HAL_OK) consolePrint("S1 UART failed to send data
        to the test station!\n\r");
53      else consolePrint("S1 UART sent to station successfully!\n\r");
54    } else {
55      consolePrint("S1 UART:Received a message directed not to me but to
        subject %u.\n\r", (char)msg[1]);
56    }
57  }
```

Q.1.8 Test station I2C code

The I2C code for the test station, introduced in Section 4.3.5:

```
1  for (int i = 0; i < 4; ++i) {
2    if (testSubjects[i]->enabled == 1) {
3      testSubjects[i]->i2c_operational = 0; // Before testing I2C, set it
       to not being operational by default
4      // Send the number in the list of numbers that is currently indexed:
5      i2cTxSubjects[0] = commsNums[i2cCommsNumsCounter];
6      HAL_StatusTypeDef retTx = HAL_I2C_Master_Transmit(hi2cSubjects,
       testSubjects[i]->i2cAddress, (uint8_t*)i2cTxSubjects,
       SUBJECTS_I2C_TX_SIZE, 1000);
7      // If successful transmission, attempt to receive. If successful
       receive, set I2C as operaitonal.
8      if (retTx == HAL_OK) {
9        // Clear the receive buffer before initiating a receive:
10       for (int j = 0; j < SUBJECTS_I2C_RX_SIZE; ++j) i2cRxSubX[j] = 0;
11       HAL_StatusTypeDef retRx = HAL_I2C_Master_Receive(hi2cSubjects,
       testSubjects[i]->i2cAddress, (uint8_t*)i2cRxSubX,
       SUBJECTS_I2C_RX_SIZE, 1000);
12       if ((retRx == HAL_OK) && (i2cRxSubX[0] == i+1) && (i2cRxSubX[1] ==
        i2cTxSubjects[0]*i2cTxSubjects[0])) testSubjects[i]->i2c_operational
        = 1;
13       // Move on to the next "random number" in the list
14       ++i2cCommsNumsCounter;
15       if (i2cCommsNumsCounter > (SUBJECTS_COMMS_NUMS_SIZE - 1))
       i2cCommsNumsCounter = 0;
16     }
17   }
18 }
```

Q.2 Python Function definitions

Q.2.1 Raspberry Pi

Q.2.1.1 Identifying serial ports

A function to retrieve the device name and last six digits of the serial number of the STM32s plugged into the USB ports of the Raspberry Pi, which was discussed in Section 4.4.3.2:

```
def get_plugged_in_ACM_ports():  # Credit: ChatGPT
    context = pyudev.Context()

    # Initialize the list to store the COM port and serial number pairs
    device_list = []

    for device in context.list_devices(subsystem='usb'):
        if device.device_type == 'usb_device':
            product_id = device.attributes.get('idProduct')
            if product_id and product_id.decode() == '374b':  # Filtering by product ID
                serial = device.attributes.get('serial')
                com_port_found = False
                for child in device.children:
                    if child.subsystem == 'tty':
                        com_port = child.device_node
                        if com_port.startswith('/dev/'):
                            com_port = com_port[5:]  # Strip the '/dev/' prefix
                        if serial:
                            serial = serial.decode()
                            short_serial = serial[-6:]  # Get the last 6 digits of the serial number
                            device_list.append([com_port, short_serial])
                        else:
                            device_list.append([com_port, "No Serial Number"])
                        com_port_found = True
                if not com_port_found:
                    if serial:
                        serial = serial.decode()
                        short_serial = serial[-6:]  # Get the last 6 digits of the serial number
                        device_list.append(["No COM Port", short_serial])
                    else:
                        device_list.append(["No COM Port", "No Serial Number"])
```

```
32
33      return device_list
```

Q.2.2 GUI

Q.2.2.1 Message to server

As mentioned in Section 4.5.2, the following function was used to send commands to the Raspberry Pi, which would translate the commands into the correct commands to the test station.

```
1 def send_message_to_server(self, msg):
2     try:
3         app.frm_settings.msg_from_client = msg
4         app.frm_settings.msg_from_client_bytes = app.frm_settings.
    msg_from_client.encode("utf-8")
5         app.frm_settings.msg_from_client = 'e'  # Error by default
6         app.frm_settings.udp_client.sendto(app.frm_settings.
    msg_from_client_bytes, app.frm_settings.full_server_address)
7         return 'success'
8     except Exception as e:
9         return e
```