YRKESHÖGSKOLAN
NOVIA

Streamlining Electrical Battery Testing Through Automation

A method to automate the testing of automotive batteries employing LabVIEW, TestStand, and Ardiuno.

Shahab Khalghani

Master's Thesis for Master of Science

Degree Programmed in Automation Engineering

Vaasa 2025

DEGREE THESIS

Author: Shahab Khalghani

Degree Program and place of study: Master of Science, Novia University of Applied Science, Vasa.

Specialisation: Automation engineering

Supervisor(s): Joachim Böling

Title: Streamlining Electrical Battery Testing Through Automation

---

Date: 9.6.2025 Number of pages: 134 Appendices: 11

Abstract

The thesis presents an affordable automated test bench tailored specifically for a unique prototyping automotive battery pack. This automated test bench removes the need for manual involvement during the testing process. Software tools applied in this automated test bench are NI TestStand, NI LabVIEW and Arduino. The automated test sequence begins after the testing operator verifies the wiring connections and scans the battery pack's QR code to log its serial number. The already logged serial number is then saved into NI TestStand, which initiates the necessary test sequences. NI TestStand coordinates the test process by integrating NI LabVIEW and Arduino programming files, running them as part of the automated workflow. Once the test sequence is complete, NI TestStand generates a detailed final report on the battery's performance.

The thesis describes a range of tests and verifications performed on a prototype battery, such as measuring environmental temperature and humidity, monitoring cell voltage and temperature, writing and reading data to EEPROM, and conducting equipotential, isolation, and dielectric tests. The hardware tools utilized in this automated test setup comprise a prototyping battery pack, a GW Instek GPT 9804 safety tester for quality test measurements, two Arduino controllers for managing relays and communicating with the battery, and a sensor for monitoring environmental humidity and temperature.

Since this automation system functions autonomously, the test report data is automatically saved in cloud storage. Microsoft Azure is selected as the cloud storage platform, offering both Cold and Hot storage options. The test setup can be accessed remotely via the RDP protocol, or data can be transferred across multiple platforms using the FTP protocol. Cybersecurity protections and IT readiness are set up to keep cloud storage, remote monitoring, and file transfers safe and reliable.

Language: English
Key Words: Automation test Setup, NI TestStand, NI LabVIEW, Arduino, EV & Hybrid Battery

Abbreviations

ACK Acknowledge (bit)
API Application Programming Interface
ASCII American Standard Code for Information Interchange
BMS Battery Management System
CMC Cell Module Controller (board)
CIPO Controller Input Peripheral Output (SPI serial communication protocol)
COPI Controller Output Peripheral Input (SPI serial communication protocol)
CR Carriage Return (UART)
CRC Cyclic Redundancy Check (mathematical technique for error correction)
CS Chip Select (SPI serial communication protocol)
DAC Digital-to-Analog Converter
DCIR Direct Current Internal Resistance (quality test for battery)
DEC Digital Equipment Corporation
DLC Data Length Code (bits)
ECU Electronic Control Unit
EEPROM Electronically Erasable Programmable Read-Only Memory
EIA Electronic Industries Association
EOL End of Line
EMC ElectroMagnetic Compatibility
EMF Electro-Magnetic Field
EOF End of Frame (CAN frame)
EV Electrical Vehicle
FPCB Flexible Printed Circuit Board
FTP File Transfer Protocol
GBA Global Battery Alliance
GND Ground (electrical wiring)
GRS Geo-Redundant Storage (Microsoft Azure)
GSM Global System for Mobile Communications
HDD Hard Disk Drive
HDL Hardware Description Language
HID Human Interface Device
HTML Hyper Text Markup Language
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
I2C Inter-Integrated Circuit
IC Integrated Circuits
IDE Integrated Development Environment (Arduino)
IEC International Electrotechnical Commission (standard)
IIS Internet Information Services (Windows operating system)
IOPS Input/Output Per Second
I/O Input/Output
ISO International Organization for Standardization (standard)
IT Information Technology
LabVIEW Laboratory Virtual Instrument Engineering Workbench
LED Light-Emitting Diode
LF Line Feed (UART)
LFP Lithium Iron Phosphate
LRS Locally Redundant Storage (Microsoft Azure)
LTO Lithium Titanate Oxide
NACK Not Acknowledge (bit)

NAT Network Address Translation
NI National Instruments
NL New Line (UART)
NMC Nickel Manganese Cobalt
NO Normally Open (relay)
NoSQL Not Only Structured Query Language
NTC Negative Temperature Coefficient
NXP Next eXPerience
OEM Original Equipment Manufacturer
OLED Organic Light-Emitting Diode (display screen)
OSI Open Systems Interconnection
PCB Printed Circuit Board
PDF Portable Document Format
PDU Protocol Data Unit
QR Quick Response
R&D Research and Development
RA-GRS Read-Access Geo-Redundant Storage (Microsoft Azure)
RAM Random Access Memory
RDP Remote Desktop Protocol
RG Resource Groups (Microsoft Azure)
RTL Register Transfer Level
RTOS Real-Time Operating System
RTU Remote Terminal Unit
SA Storage Account (Microsoft Azure)
SCL Serial Clock (I2C serial communication protocol)
SCLK Serial Clock signal (SPI serial communication protocol)
SDA Serial Data (I2C serial communication protocol)
SAE Society of Automotive Engineers (Standard)
SFTP SSH File Transfer Protocol
SLA Sealed Lead Acid (battery)
SMB Server Message Block (Microsoft Azure)
SOC State of Charge (battery)
SOF Start of Frame (CAN frame)
SPI Serial Peripheral Interface (communication protocol)
SQL Structured Query Language
SRAM Static Random-Access Memory
SSH Secure Socket Shell
SSL Secure Sockets Layer (encryption)
SSD Solid State Drive
TCP Transmission Control Protocol
TLS Transport Layer Security (encryption)
TX Texas Instruments
UART Universal Asynchronous Receiver/Transmitter
UL Underwriters Laboratories (standard)
UN United Nations (Standard)
USB Universal Serial Bus
VI Virtual Instrument (LabVIEW)
VPN Virtual Private Network
WIFI Wireless Fidelity
XML Extensible Markup Language
XOR Exclusive OR logic
ZRS Zone-Redundant Storage (Microsoft Azure)

Table of Contents

1 Introduction

The global automotive battery market is experiencing substantial growth. According to a 2019 report by McKinsey and the Global Battery Alliance (GBA), global battery demand was forecasted to reach 2.6 TWh annually with a 25 percent yearly growth rate by 2030. However, a 2022 analysis by McKinsey Battery Insights indicates that the entire lithium-ion (Li-ion) battery chain, spanning from mining to recycling, could expand by over 30 percent annually from 2022 to 2030. By 2030, the Li-ion battery market is anticipated to exceed $400 billion, with a total market size of 4.7 TWh [1].

Batteries are rapidly produced in high-speed, automated factories to meet the significant demand mentioned earlier. Battery technology advances every year in many different ways. As a result, numerous automotive companies are creating **prototypes** for their latest electric vehicle (EV) and hybrid batteries to stay ahead of these fast-paced innovations. During the development stage, these new and evolving batteries require adjustments in various aspects like design, functionality, software integration, assembly processes, performance metrics, safety standards, and testing protocols—from examining individual battery cells to end-of-line testing. During the prototyping phase, the production rate is relatively low, but the emphasis on the development process is significant. This means that high-speed serial factories are not required at this stage, as the assembly is primarily carried out manually by human workers, and testing during production is only partially automated.

The thesis aims to create an affordable semi-automatic testing setup (shown in Figure 18) specifically designed for a unique prototyping automotive battery pack (shown in Figure 19). This proposed semi-automatic test bench is tailored exclusively for one unique battery pack and is not compatible with other types of batteries. Information regarding the unique battery pack is provided in Section 3. This automated test bench verifies environmental conditions and battery communication signals, performing quality tests automatically without the need for human intervention. Once all testing steps concluded, the semi-automated test bench generates a test report and stores the test result in Azure cloud storage. This automated test bench can be remotely monitored via developers to check the test results if needed. Also, developers can remotely transfer test report files to their personnel PCs or company Azure server if intended. The development of this

automatic testing setup took place at the Prototyping factory within IONCOR Oy situated in Uusikaupunki.

In 2019, IONCOR Oy (previously known as Valmet Automotive until rebranding in August 2024) opened a battery factory in Salo, Finland, and later expanded with another facility at their vehicle manufacturing site in Uusikaupunki. These developments highlight IONCOR's strong market position and commitment to electric battery mobility. Since 2017, the company has also independently produced battery systems and packs. IONCOR's expertise includes testing high-voltage batteries, centered around a state-of-the-art testing facility in Bad Friedrichshall, Germany, which serves as the hub for their testing operations [2].

Section 1 will first cover the technical background of automotive batteries, followed by an introduction to the primary focus: the proposed automated test bench. Section 2 covers the background knowledge and motivations for the thesis. Section 3 details the unique prototype battery pack used with the proposed test bench. Section 4 presents the overall schematic of the test bench, including the integration of components and introduction to its controllers and software tools. Section 5 thoroughly examines the testing and validation sequence for the prototype battery pack. Sections 6, 7, 8, and 9 discuss IT features like Virtual Private Network (VPN), Azure cloud storage, remote monitoring, and remote data transfer, respectively. Section 10 delves into the methodology employed in this thesis and explores other methodologies applied in automotive battery testing. It provides a concise overview of the process of designing a battery from its electrical foundations and its progression toward mass production in a factory setting. Section 11 explores further studies and tools used during the thesis. Finally, Section 12 summarizes all the findings.

1.1 Evolution of automotive batteries

The following chapters will provide a summary of the evolution of quality tests for EV-hybrid automotive batteries compared to SLA (Sealed Lead-Acid) batteries. The historical development of the electrical aspects of automotive batteries will be examined, with a focus on their evolution independently of chemical differences and cell architecture. Additional academic references will be included to discuss the electrical behavior of

batteries with varying cell chemistry and architecture. Furthermore, communication protocols for digital data exchange in printed circuit board (PCB) circuits will be reviewed, along with an overview of how messages appear in these protocols. Each chapter will conclude with a summary to enhance understanding of automotive battery evolution.

1.2 Battery testing in automotive industry

When this thesis discusses automotive batteries in the context of automation test setups, it specifically means hybrid/EV batteries, not SLA batteries. The proposed test setup outlined in this thesis includes specialized tests designed for hybrid/EV battery packs, which are not applicable to SLA batteries. For instance, tests like Isolation (also known as electrical insulation), dielectric (also known as high voltage withstand), and equipotential (also known as ground bonding) outlined in this automation test setup, are tailored for hybrid/EV batteries and are not conducted for SLA batteries.

SLA batteries were traditionally utilized in combustion engine vehicles for tasks like initiating the engine and powering auxiliary systems such as lighting. Typically, SLA batteries in standard passenger vehicles operate at 12 volts, with their positive terminals linked to various electrical components. Meanwhile, the negative terminal of the SLA battery is commonly connected to the vehicle's metal chassis. Similarly, the negative terminals of other electrical components are also often connected to the metal chassis. To prevent any potential variance in voltage across the metal chassis, it is advisable for the negative wires of these components to be of equal length, as illustrated in Figure 1 (right).

SLA batteries are still used in hybrid and electric vehicles. For instance, during the startup process of an EV or hybrid battery, contactors need to be closed. To do this, an external power source like an SLA battery or another DC power supplier is required. Additionally, some SLA batteries provide power for auxiliary equipment such as the Battery Management System (BMS) and lighting. The electrical schematic for SLA batteries is the same one witnessed in combustion engine cars. However, unlike combustion engine vehicles, in hybrid and electric cars, the negative terminal of the EV battery is not connected to the chassis but rather to the DC/AC converter. Both the positive and negative terminals of EV batteries must be effectively isolated from the vehicle's chassis

[and its battery metal's chassis] due to the higher power/energy density compared to SLA batteries. This crucial isolation for both terminals of EV/Hybrid batteries from exterior metal frame of battery is guaranteed via isolation and dielectric tests. The detailed logic behind isolation and dielectric tests will be explained in Subsections 5.8 and 5.9. Additionally, these test conditions will be outlined in Subsection 1.7.

In Figure 1 (left) ground wires are included to connect electrical parts like the EV/hybrid battery, converter, and electric motor, etc, ensuring they all have a common grounding point. To guarantee safety, equipotential tests are carried out to make sure that all exposed metal parts can be safely touched like the car's chassis, are at the same voltage level. This helps prevent electrical shocks by ensuring that these parts are properly grounded. A crucial aspect of UN regulation ECE R100 specifies that the equipotential bonding between two high-voltage components must be less than 100 milliohms in their conductivity resistance. This mentioned conductivity resistance restriction applies to the vehicle's chassis [3]. Therefore, the conductivity resistance across the battery's chassis alone should be less than 100 milliohms before assembling with an EV vehicle. The detailed logic behind the equipotential test will be discussed in Subsection 5.6 and test conditions will be outlined in Subsection 1.7.

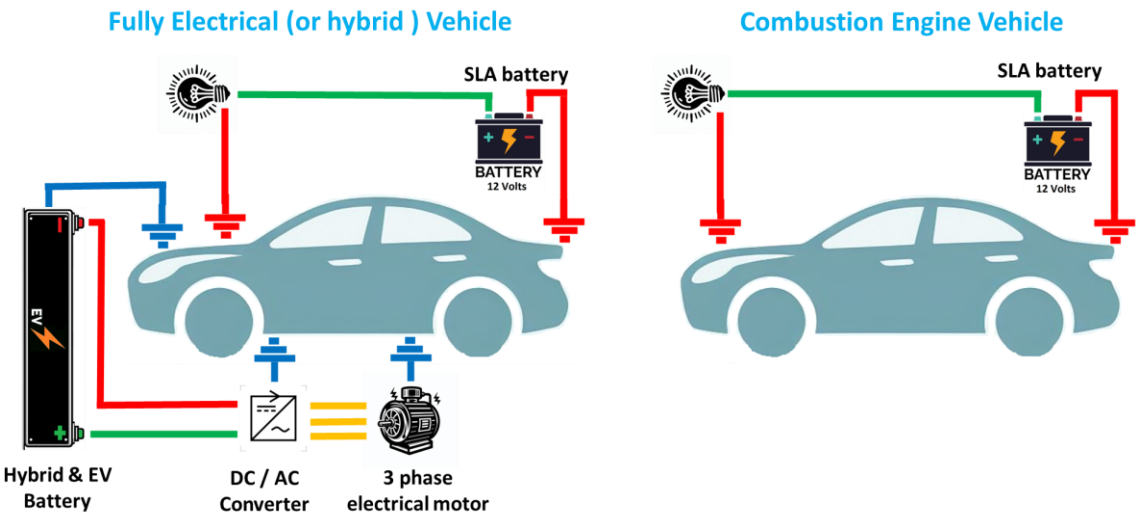


Figure 1. Battery wiring in Combustion Engine and EV cars

1.3 Battery cell chemistry history used in automotive industry

Contrary to popular belief, battery technology has ancient roots dating back to approximately 1000 BCE during the Parthian Empire in ancient Iran, as evidenced by the discovery of early batteries near Baghdad. Although the precise purpose of these ancient batteries is unclear, their components bear resemblance to modern counterparts. The first recognized battery in Western history was developed by Allesandro Volta in 1782. Additionally, the Daniell cell, based on Faraday's principles, emerged as the first modern storage cell in 1836 [4].

This thesis exclusively examines batteries used in the automotive sector. The practical application of batteries in vehicles commenced in the late 19th century with the advent of rechargeable SLA batteries, initially utilized for electrical accessories like lights and ignition systems. As the early 20th century dawned, Lead-Acid batteries emerged as the predominant energy solution for electric cars, primarily due to their cost-effectiveness, notwithstanding their disadvantages of heaviness, low energy density, and restricted longevity. These limitations prompted the investigation of alternative solutions [5].

In the present era, three prominent contenders for lithium-ion battery cell chemistry have emerged: NMC (Nickel Manganese Cobalt), LFP (Lithium Iron Phosphate), and LTO (Lithium Titanate). Determining superiority among these options is challenging as each one exhibits both advantages and disadvantages. Figure 2 illustrates a radar chart comparing cost, energy density, power density, safety features, susceptibility to overcharging and over-discharging, tolerance to high and low temperatures, mechanical stress, and lifespan for all three options [6] [7] [8] [9].

Power density denotes the amount of power delivered per unit of weight or volume. In automotive batteries, higher power density is favored for applications emphasizing rapid car acceleration, superior performance, and swift charging/discharging. The power density characteristic is similar to how capacitors behave electrically. On the other hand, energy density signifies the quantity of energy stored per unit of weight or volume. Batteries with higher energy density can store more energy, thereby extending the driving range of electric vehicles between charging stations. An excellent example of energy density is gasoline, commonly used as fuel in internal combustion engines.

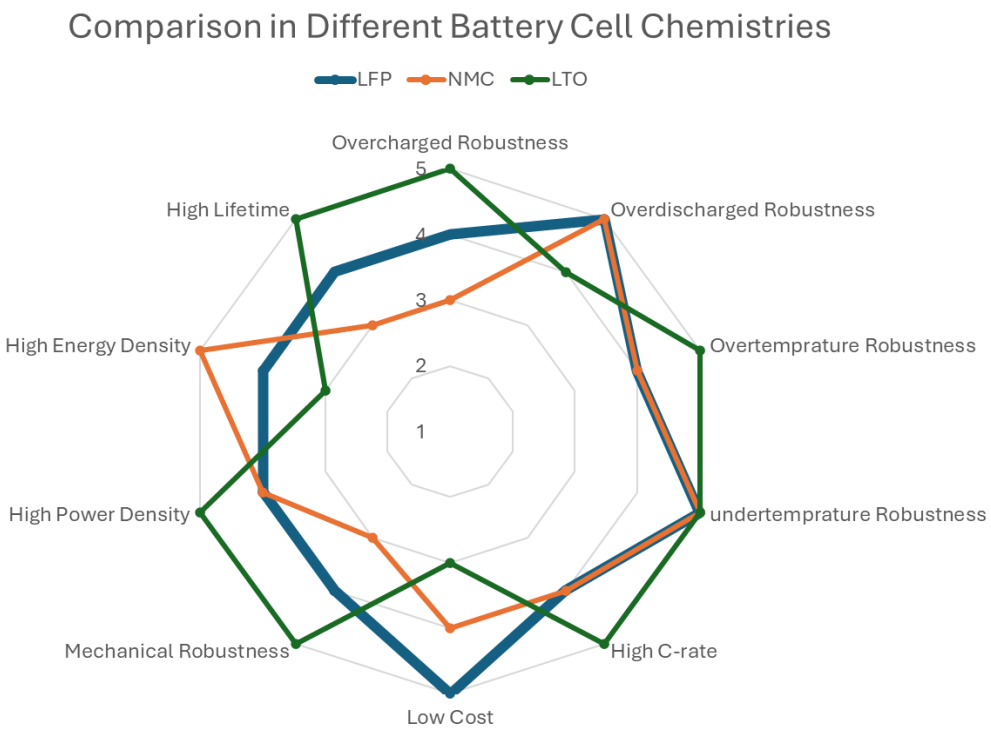


Figure 2. Battery Cell Chemistry Comparison for LFP, NMC and LTO

In Figure 2, a high C-rate indicates that batteries in electric or hybrid vehicles can be charged or discharged quickly relative to their capacity, beneficial in situations requiring rapid energy exchange. Figure 2 shows that, in terms of C-rate, LTO chemistry can be charged faster than NMC and LFP. The figure also shows that for overcharging (beyond 100% SOC), LTO, LFP, and NMC are prone to degradation and capacity loss, with LTO being the least and NMC the most susceptible. Conversely, for over discharging (below 0% SOC), both NMC and LFP show similar susceptibility to degradation and capacity loss, while LTO is more affected.

Additionally, the references evaluated temperature tolerance under load and without load. At lower operational temperatures, all chemistries perform similarly; however, at higher temperatures, LTO is more stable than NMC and LFP. Mechanical stress, especially from direct damage, is crucial for passenger safety, with LTO chemistry showing the least susceptibility in this regard.

Different battery cell chemistries can impact electrical test standards due to changes in internal resistance. The choice of battery cells in the industry is often a subject of debate

depending on their intended application. In the next subsection, battery cell architectures that can directly affect electrical behaviors, will be introduced and studied.

In this thesis, the prototyping battery used in the proposed test bench features NMC chemistry. NMC cells operate over a wider voltage range than the other chemistries discussed in Figure 2. Specifically, NMC operates between 3.0V and 4.2V per cell, whereas LFP ranges from 2.5V to 3.65V, and LTO ranges from 1.8V to 2.85V [10]. The battery pack tested in this thesis uses NMC chemistry. Subsection 5.4 explains that the voltage tolerance for NMC cells, as shown in the TestStand report, can be verified by reading the battery cell signals' voltage in the prototyping battery pack using the Arduino controller.

1.4 Automotive battery cell architectures

Automotive battery cell architectures are catergorized into three subgroups namely, cylidrical cell, prismatic cell, and pouch cell. In a simple battery cell architecture, electrode defines battery polarity, electrolyte enables the flow of ions between electrodes, facilitating the conversion of chemical energy into electrical energy, and Separators prevent direct contact between electrodes while allowing ion flow.

Subsection 1.4 provides visual explanations for the definitions of electrodes, separators, and electrolytes in different cell architectures. The battery pack assembly process can be directly influenced by the shape of the battery cell architecture. At the conclusion of Subsection 1.4, the architecture utilized in the battery pack tested on the proposed test bench will be specified.

1.4.1 Cylindrical Cell Architecture

This familiar and widely acknowledged configuration (Figure 3) consists of a metallic enclosure housing internal elements like positive and negative electrodes, electrolyte, and separators. The cathode is the positive electrode, and the anode is the negative electrode. The most common cylindrical battery cell size widely utilized in the automotive industry, particularly in vehicles like Tesla Model S and Model X, is 18650. This designation number signifies its dimensions of 18 millimeters in outer diameter and 65 millimeters in length [11].

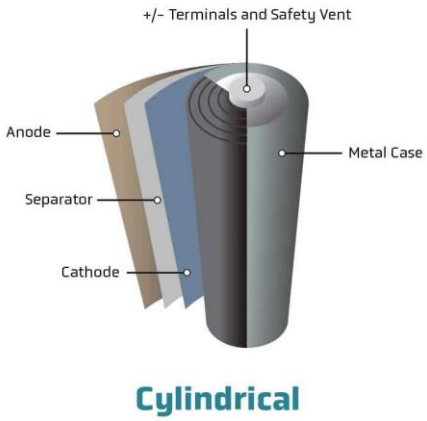


Figure 3. Cylindrical battery cell architecture [12]

1.4.2 Prismatic Cell Architecture

Lithium prismatic cells have a flat, rectangular, or square shape that can be stacked easily, unlike cylindrical cells. They are called "prismatic" because they look like prisms, with their shape resembling that of a box or rectangle. Typically, this type of cell is housed within a robust metal casing for protection [11]. Well-known automotive companies like BMW and Volkswagen utilize this cell architecture in their vehicles.

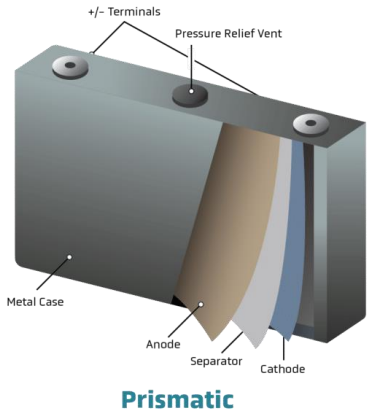


Figure 4. Prismatic battery cell architecture [12]

1.4.3 Pouch Cell Architecture

A pouch lithium-ion battery cell is recognized for its flexible and flat-cell design, resembling a pouch. It is usually created by layering flat electrodes and separators, which are then enclosed in a flexible pouch or bag made of materials like aluminum or various polymers, sealed with heat [11]. Automotive manufacturers such as Hyundai and Ford incorporate pouch cells into their vehicles' battery systems.

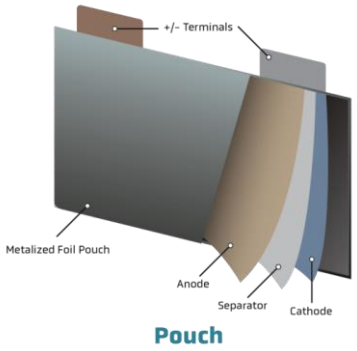


Figure 5. Pouch battery cell architecture [12]

1.4.4 Battery Cell Architectures comparison

The battery cell architectures, as depicted in Figure 6, were compared based on space efficiency for stacking within battery modules, manufacturability in terms of time and complexity, weight relative to electrical capacity, resistance to physical damage, cost, durability, and technology maturity development, with citations from [11] [12] [13] [14]. In Figure 6, comparisons of power density, energy density, and heat dissipation were excluded due to significant disagreements in the sources previously mentioned. It is noted that unlike cylindrical cell, pouch cell technology is still in its early stages of development, while prismatic cell technology is not yet fully matured, indicating potential changes might be depicted in Figure 6 in the future.

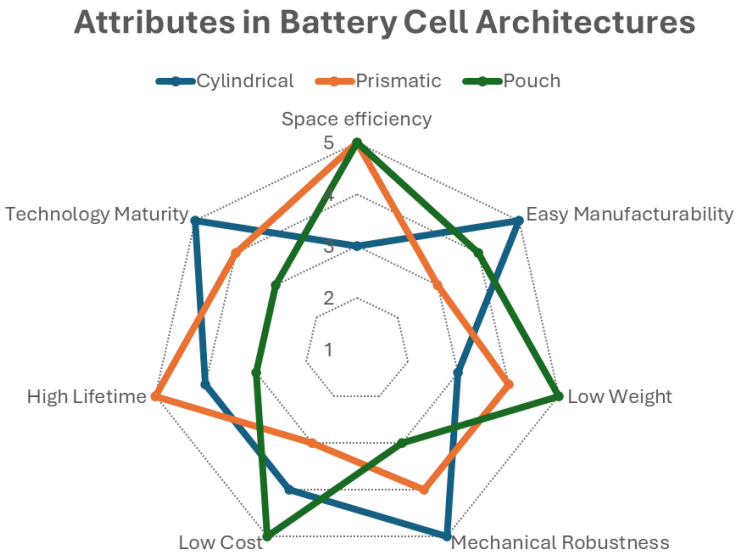
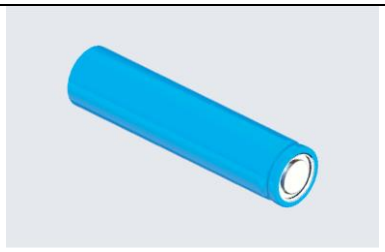
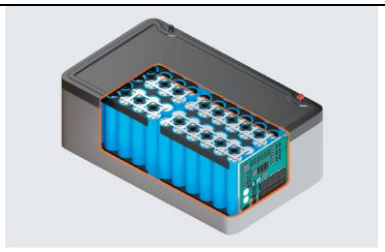
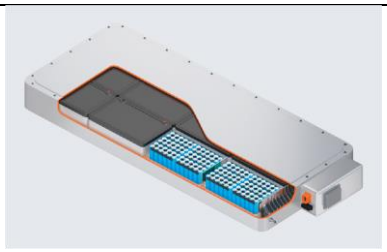
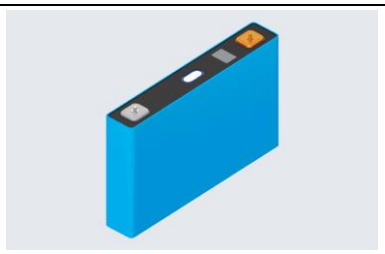
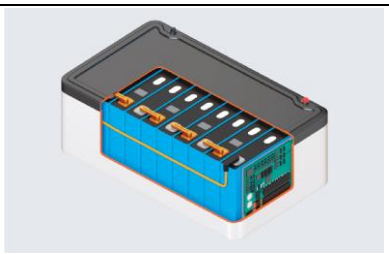
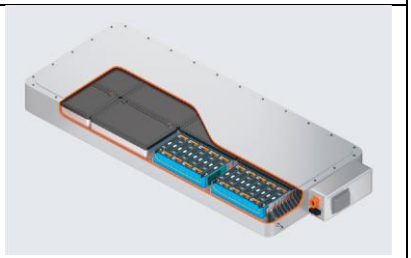
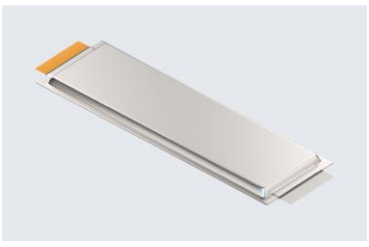
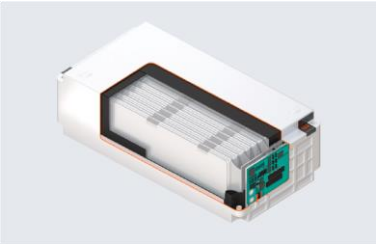
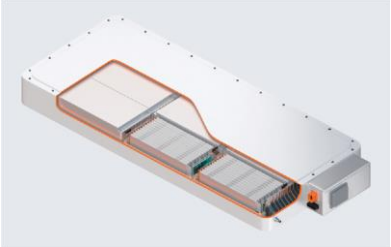


Figure 6. Battery cell Architecture comparison for Cylindrical, Prismatic and Pouch cell

1.4.5 Battery Cell Architecture in assembly process

EV-hybrid batteries are made up of individual battery cells. These cells can be stacked to form larger units, known as “Stacked Cell units” or “Battery modules,” to meet the required nominal voltage and current. When assembling these cells, it is important to consider their polarities. For instance, to achieve a higher voltage, cells are connected in series—two 3.7V cells connected in series will result in a total voltage of 7.4V. Conversely, to achieve a higher current, cells are connected in parallel—two cells with 2000mAh capacities each, when connected in parallel, will provide a combined capacity of 4000mAh at the same voltage. A CMC (Cell Module Controller) board is embedded within battery modules to monitor individual cell conditions, such as voltage, regardless of whether the cells are connected in series or parallel.

Table 1. Visual comparisons of various cell architectures and demonstrations of battery modules and packs [15]

Battery made by	**Battery Cell**	**Battery Module**	**Battery Pack**
Cylindrical cell			
Prismatic cell			
Pouch cell			

Before stacking cells into a battery module, it is crucial to ensure that all voltage cells are evenly balanced (refer to Subsection 5.4 for more details). As shown in Table 1 [15], all

three cell architectures can be stacked to form larger units. These stacked cell modules can be further assembled into a "Battery Pack" unit. A single battery pack is usually not the biggest unit because several battery packs can be combined together. These combined packs are connected to a BMS (Battery Management System), which is the smart part of the battery setup. The proposing automation test setup introduced in this thesis, focuses on testing a unique prototyping battery pack composed of prismatic cells.

1.5 Automotive battery communication protocols

The serial digital communication protocols applicable for intelligent automotive batteries can be outlined as follows: UART (Universal Asynchronous Receiver-Transmitter), I2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface), CAN (Controller Area Network) and etc. These communication protocols have their own limitations with cable functioning range, wiring line quantity, and data exchange speed rate; however, these are all considered as **serial communication protocols** [16]. Non-serial communication protocols, often referred to as parallel communication protocols, transfer multiple bits of data simultaneously, using multiple wires. The parallel method is typically faster but requires more wiring and is more complex, which makes it less suitable for long distances especially in automotive battery hardware. In this thesis on serial communication protocols, the term "controller" is used instead of "master," and "peripheral" is used instead of "slave" to utilize more appropriate language in the technical field.

The following sections provide an overview of each serial communication method employed in the proposed test bench, along with an analysis of the digital binary signal logs using a logic analyzer. These digital log figures do not include error detection bits (parity bits) or bytes (CRC bytes). Detailed information on these error detection methods can be found in Subsection 3.6.1.

1.5.1 UART communication Protocol

UART (Universal Asynchronous Receiver/Transmitter) is an old school protocol, was firstly developed by Gordon Bell of Digital Equipment Corporation (DEC) in the 1960s. UART is now utilized for communication between the microcontroller and GSM (Global System for Mobile Communications), Bluetooth, or WIFI modules. Additionally, UART is widely used for **debugging** during firmware development of a battery's BMS, allowing specific sections

or lines of code to be checked. The maximum speed in this serial protocol is up to 115,200 bps, being very low when comparing with new other serial protocols [16], [17].

This serial communication protocol includes only two communication lines of Rx (Receive) and Tx (Transmit). UART supports half-duplex communication, but UART can be used in full duplex as well when using new separate lines of Tx and Rx. In a half-duplex system, data transmission occurs in both directions but not simultaneously. On the other hand, in a full-duplex system, data transmission occurs in both directions simultaneously [16], [17].

UART is a low-cost protocol and needless of clock frequency (asynchronous) unlike I2C and SPI protocol. UART is suitable for long-distance communication unlike other serial communication protocols [16], [17]. In Figure 7, the Vcc (power) and GND (ground) lines are shared between the controller and the peripheral. The Rx and Tx lines are connected in reverse when linking the controller to the peripheral.

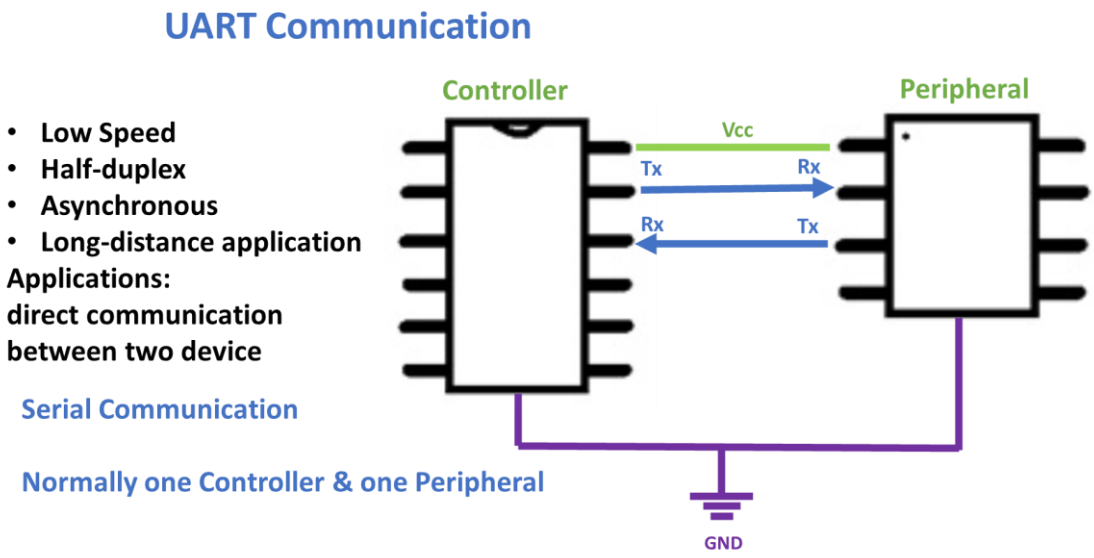


Figure 7. UART communication Protocol Configuration

The bit per second (bps) communication speed and transmission distance in UART are directly related; shorter distances enable faster data transfer rates [17].

- Less than 1 Foot at 115,200 bps
- 50 Feet at 19,200 bps
- 500 Feet at 9,600 bps

- 1000 Feet at 4,800 bps
- 3000 Feet at 2,400 bps

On Arduino boards, UART serves as a gateway from USB to PC. In the Arduino IDE application, "serial monitoring" primarily adheres to the UART protocol, labelled as Rx0 and Tx0. The inactive binary mode in UART is "High" status. UART is unidirectional, as demonstrated in Figure 8, with Rx active (message detected) and Tx totally inactive (no message detected). The communication always starts with a status change from "High to Low" and ends with a status change from "Low to High". Figure 8 illustrates the physical measurement of the value "0" (or “0x30” in hexadecimal) via a logic analyser hardware connected to Rx line and Tx line. At the end of UART message in Figure 8, in ASCII (American Standard Code for Information Interchange), CR (Carriage Return) or "0x0D" and LF (Line Feed) or "0x0A" are witnessed. Together, CR and LF are frequently used to initiate a new line. For instance, in many systems, sending both CR and LF (CR+LF or 0x0D 0x0A) moves the cursor to the beginning of the next line. In Arduino IDE’s serial monitoring, both LF (or NL "New Line") and CR can be viewed if the option "both NL & CR" is selected.

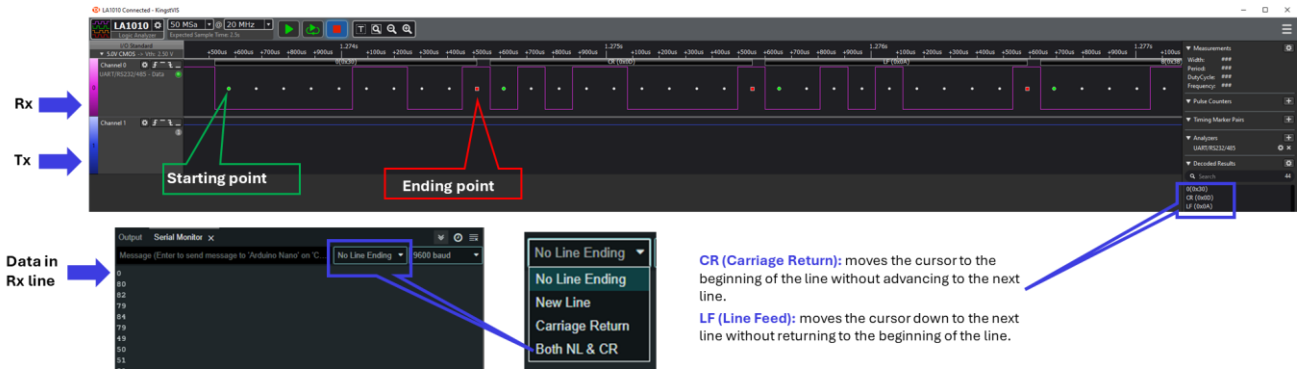


Figure 8. UART message inspected by logic analyzer hardware

1.5.2 I2C communication Protocol

I2C (Inter-Integrated Circuits) is a communication protocol designed for communication between integrated circuits (ICs). It is primarily used for short-distance communication, typically between two ICs situated on the same printed circuit board (PCB). The protocol was developed by Philips Semiconductor (now NXP Semiconductors) in 1982. It gained

popularity in applications prioritizing low costs and simple implementation over high-speed performance [18].

I2C is supported by all major IC manufacturers, and it is commonly implemented in development boards and single-board computers like Arduino and Raspberry Pi for communication with peripherals. Many types of computers, including those running on Windows, MacOC, Linux, and other operating systems, can utilize I2C protocol for communication purposes such Human Interface Device (HID) devices (Keyboards, Mice and etc.) [18]. Older HID devices can be connected using I2C instead of USB, but since its introduction in 1996, the USB protocol has become more popular because of its plug-and-play capability and faster speed compared to I2C. Nowadays, I2C is more commonly used in embedded systems.

The I2C protocol involves a two-wire connection [half duplex] and supports communication in only one direction at a time, meaning devices can either receive or send data but not both simultaneously. The I2C protocol offers various speed modes. Below the speed modes of the I2C protocol are defined by their maximum bit per second (bps) rates [19].

- Standard-mode 100 Kbps
- Fast-mode 400 Kbps
- Fast-mode Plus 1 Mbps
- High-speed mode 3.4 Mbps
- Ultra-Fast mode 5 Mbps

The I2C protocol is commonly known as on-board communication since it is mainly utilized within a single PCB board containing one or more controllers and multiple peripherals. Half duplex I2C protocol consists of two wires namely SDA (Serial Data) and SCL (Serial Clock), which SDA receives and sends the serial data and SCL carries clock signal making the communication time synchronous [20].

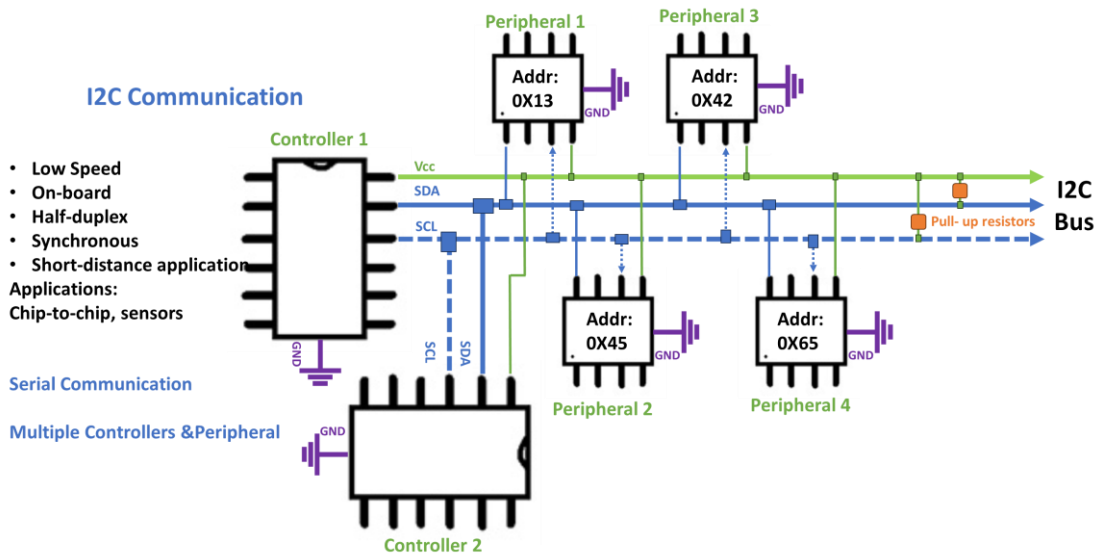


Figure 9. I2C communication protocol configuration

As depicted in Figure 9, the controller sends message to peripherals based on hexadecimal address. This hexadecimal address depending on the design, can have 7-bit addressing or 10-bit addressing, which obviously 10 bits of address makes more peripherals connect to I2C controllers. One controller cannot directly communicate with another available controller, which is why no address is assigned to any I2C controllers in Figure 9. Controllers can only communicate with each other through peripherals.

The whole peripherals receive the SCL (Serial Clock Line) signal exclusively from a single predefined primary I2C controller. As a result, other secondary I2C controllers are unable to manage this SCL line at any given time during a communication session. When SCL line is not in use by any I2C controllers, another controller can take control to start its communication using SCL line. This coordination ensures that data collisions are avoided on I2C bus [18-20].

As shown in Figure 9, both the SDA and SCL lines must be open-drain. This means that I2C controllers can only pull these two lines to a low state. To return the lines to a high state, each line needs a pull-up resistor connected to power line Vcc. A single pull-up resistor for each SDA and SCL line is sufficient in an I2C bus. In Figure 9, the grounding points for all I2C controllers and peripherals are connected to a single GND point, which means they share a common grounding location.

In this thesis, the I2C protocol was **bitmapped** for an Arduino OLED (Organic Light-Emitting Diode) display monitor to show ambient temperature and humidity on a test bench (see Figure 10). Figure 10 (left) shows ambient temperature and humidity are 27 ◦C and 31% in order. Regarding this ambient verification with Arduino OLED will be discussed later on Subsection 5.3. Reference [21] provides a detailed explanation with an example of bitmapping for an Arduino OLED display monitor using the I2C protocol. To better understand the communication between the I2C controller and the I2C peripheral, the OLED was deliberately programmed to display nothing. Consequently, all data should be "Null" as nothing is displayed on the screen. In hexadecimal ASCII, Null is represented as "0x00", which should not be confused with the number "0", represented as "0x30" in hexadecimal ASCII. With Null data displaying nothing, it becomes easier to inspect the communication between the controller and the peripheral using a logic analyzer.

Figure 10. Setting the OLED display to show nothing while in active mode

Figure 11 illustrates the SCL and SDA lines of the I2C protocol, inspected through a logic analyzer, showing communication between the OLED screen (I2C peripheral) and the Arduino kit board (I2C controller). This segment of serial bitmapping consists of 10 bytes. The first byte indicates the I2C peripheral's address along with the write bit, the second byte indicates the I2C peripheral's registry byte, and the remaining bytes are related to the data necessary for bitmapping to display characters on the screen. Each byte is accompanied by an ACK (Acknowledge) bit, represented as 0. When the peripheral successfully receives a byte of data, it pulls the SDA (Serial Data) line low to send an ACK bit, which corresponds to 0, while a NACK is represented by a 1 in the I2C protocol.

The detailed description regarding Figure 11 is provided below.

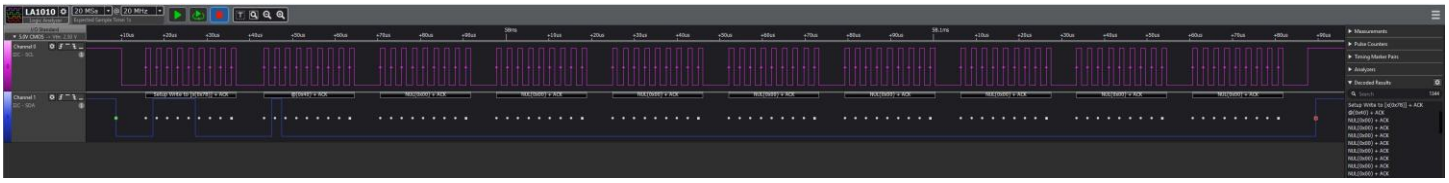


Figure 11. Analyzing I2C communication message

I2C starting Bit:

SDA line goes from high (1) to low (0) while SCL line starts clocking.

Setup write to [x(0x78)]+ACK:

Which indicates that the I2C controller is initiating a write operation to the I2C peripheral device with a hexadecimal address of 0x78. The peripheral acknowledges (ACK) this action, indicating its readiness to receive data. Actually, 0x78 not only represents the hexadecimal address but also signifies the write mode operation. In binary, 0x78 is “01111000”, which consists of “0111100” on the left and “0” as the last digit on the right. “0111100” is the actual peripheral address, which in hexadecimal is 0x3c, while the “0” on the right indicates the Write operation imposed by the I2C controller on the I2C peripheral. If the operation were set to read, the byte would be “01111001” or 0x79, with the last digit on the right being “1”.

Peripheral’s registry byte @(0x40)+ACK:

“@(0x40)” represents the I2C peripheral's registry byte. If the I2C controller does not set this byte with payload data, the I2C peripheral will be unable to receive the payload needed for screen display. This “@(0x40)” byte acts as a predefined signal for the peripheral to listen to its controller.

Null bytes (0x00)+ACK:

Null bytes are payload data deliberately provided to the I2C peripheral to ensure the display screen shows nothing, as programmed in Figure 10. There will be eight bytes with null values. If any characters are programmed to be printed on the screen, their bitmapping process will be reflected within these bytes.

I2C ending Bit:

SDA goes from low (0) to high (1) while SCL ends functioning and finalizes with high (1).

1.5.3 SPI communication Protocol

Motorola developed the SPI (Serial Peripheral Interface) architecture in the 1980s. The SPI protocol facilitates very high speeds communication supporting full-duplex (four wires), serial and asynchronous communication. Full duplex means that data transmission and reception can happen simultaneously, which is a significant advantage over I2C communication where only one-way data transfer can occur at a time. However, SPI requires four wires for communication, compared to the two wires needed for I2C. The four wires are named COPI (Controller Output Peripheral Input), CIPO (Controller input Peripheral Output), SCLK (Serial Clock Signal) and CS (Chip Select) [16],[20], [22].

The maximum speed observed for SPI is 40 MHz, with a typical speed of 25 MHz. SPI can transmit either 8 or 16 bits of data at a time. SPI speed is Normally mentioned with Hz unit. In SPI protocol, there is 1:1 relationship between bits per second (bps) and hertz (Hz). Therefore, to calculate the maximum bits per second (bps) for SPI protocol, the following equation is used [23].

$$\text{Data rate (bps)} = \text{Clock frequency (Hz)} \times 1 \text{ bit}$$

Hence, according to [20], the SPI speed rates are:

- In maximum speed is 40 Mbps
- In routine speed is 20 Mbps

In certain references like [24], there is a claim that the maximum SPI speed can exceed 100 Mbps, although there is no clear explanation provided on how this is achieved. In contrast to I2C, the SPI protocol does not require pull-up resistors. The three output pins are SCLK, COPI, and CS(s) from the controller and CIPO is the input pin for the controller shown in Figure 12. In SPI, V_{cc} and ground wires are shared between peripherals and the controller.

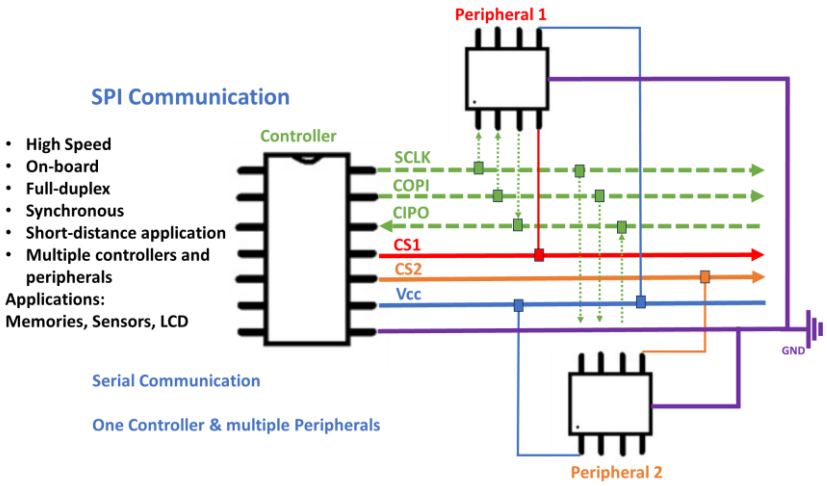


Figure 12. SPI communication Protocol Configuration

In the automotive industry, the SPI configuration shown in Figure 12 is less used. Instead, isoSPI (isolated SPI) is preferred due to its advantages in galvanic isolation, noise immunity, quick switching, robust signal integrity, low power consumption, longer communication distances, and safety benefits [23]. IsoSPI setup isolates all peripherals and the controller, meaning the Vcc and ground lines are not shared with any units. Each of the three lines—SCLK, COPI, and CIPO—has a pair. For instance, SCLK has SCLK+ and SCLK- lines. If SCLK in Figure 12 is 5 volts, then in Figure 13, SCLK+ is +2.5 volts and SCLK- is -2.5 volts, making the differential (SCLK+) – (SCLK-) equal to 5 volts. Thus, isoSPI can be seen as differential signalling for communication. The CS line remains a single line in the isoSPI protocol, similar to the SPI protocol, as shown in Figure 12 [23]. It is important not to confuse isoSPI's voltage differential signalling with CAN High and CAN Low (will be introduced next subsection), as CAN High is around 3.5 volts and CAN Low is around 1.5 volts, whereas in isoSPI, each line in a pair has the **same** voltage but with **inverse polarity**.

For instance, if a noise of 0.2 volts affects the voltage lines of COPI+ and COPI-, differential signaling can eliminate this noise through a straightforward mathematical operation.

IsoSPI protocol without noise:

$$\text{COPI Voltage} = (\text{COPI+}) - (\text{COPI-}) = (2.5\ \text{V}) - (-2.5\ \text{V}) = 2.5\ \text{V} + 2.5\ \text{V} = 5\ \text{V}$$

IsoSPI protocol with noise of 0.2 V:

$$\text{COPI Voltage} = ((\text{COPI+}) + 0.2\text{ V}) - ((\text{COPI-}) + 0.2\text{ V}) = ((+2.5\text{ V}) + 0.2\text{ V}) - ((-2.5\text{ V}) + 0.2\text{ V})$$

$$(2.7\text{ V}) - (-2.3\text{ V}) = 2.7\text{ V} + 2.3\text{ V} = 5\text{ V}$$

A similar noise-canceling effect, as described above, also occurs in the CAN protocol.

IsoSPI is used to keep the ground and power supply (Vcc) connections of a battery **decouple** from the testing station (or even battery BMS unit). This separation prevents problems like faults or power surges in the testing station from harming the battery. Section 3 goes into more detail about how this separation is maintained by peripheral devices. Essentially, IsoSPI keeps the circuits independent, acting like an isolating transformer to protect the battery from potential damage during testing.

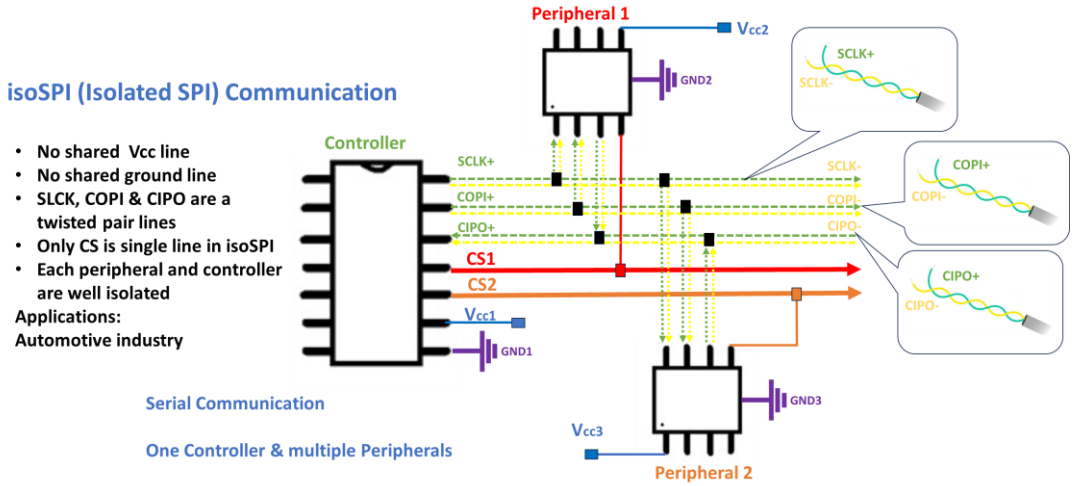


Figure 13. isoSPI (Isolated SPI) communication protocol

In Figure 13, battery components function as SPI peripherals, and the proposed test bench acts as the SPI controller. It is essential to ensure safe electrical isolation between the test bench and battery components. The battery SPI peripherals are powered by their respective battery cells, while the SPI controller is powered by the test bench's computer. This setup ensures that any electrical hazard originating from the test bench cannot affect the battery components being tested.

To better understand SPI messages, a logic analyser device is connected between an SPI controller and an SPI peripheral. Figure 14 shows two pictures depicting messages on the MOSI (also known as COPI) and MISO (also known as CIPO) lines. In the top picture, the message "P (0x50)" was sent from the controller to the peripheral via the COPI line. The CIPO line shows no message before "P (0x50)", indicating "255 (0xFF)" as the default state. The SCLK line, which is the frequency clock line, operates during message transmission.

SCS (also known as CS) should be low when the controller communicates with its peripheral and will be high once communication is finished. In the bottom picture, the next message "R (0x52)" appears on the COPI line, while the previous message "P (0x50)" is now visible on the CIPO line, due to the full-duplex feature. This full-duplex capability makes SPI a fast serial communication protocol, unlike I2C. The complete message displayed in Figure 14 is "PROTO1122," which is the serial number of the battery pack saved into the memory of the battery pack board (Further details in Subsection 5.5).

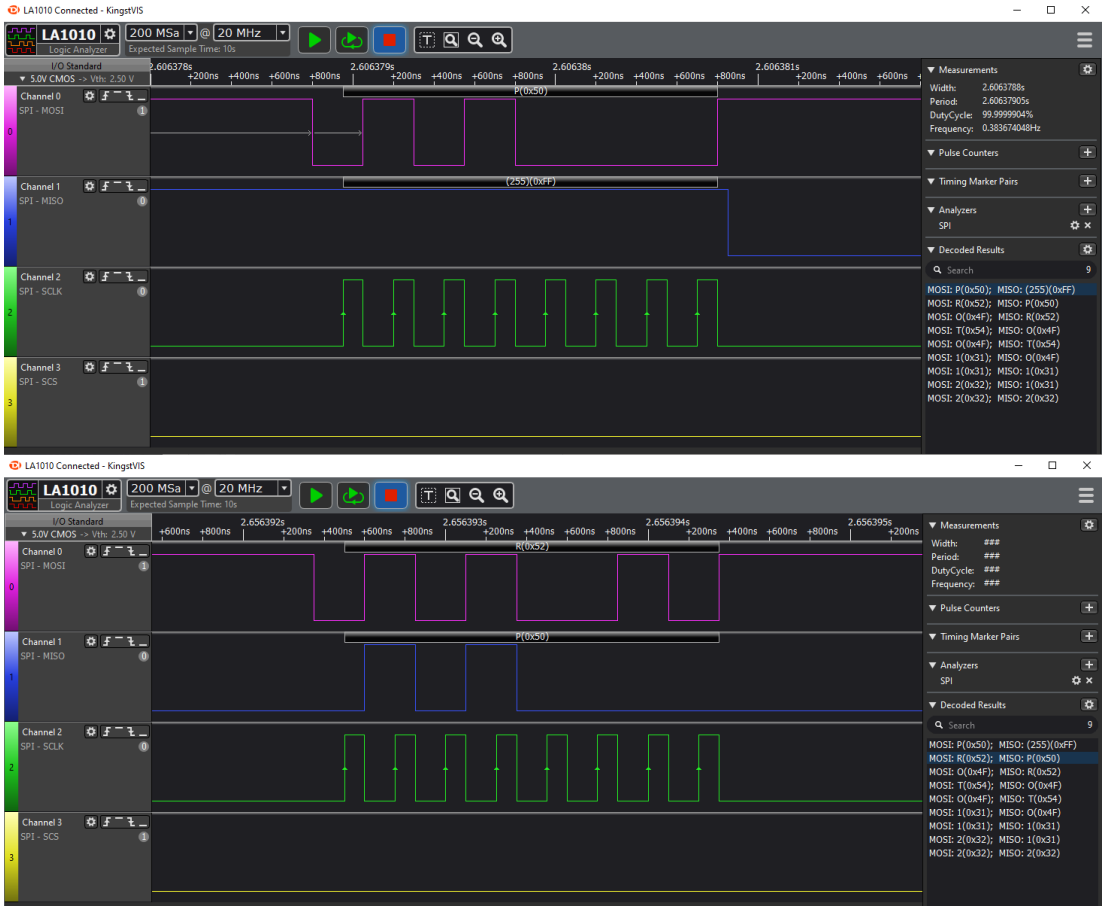


Figure 14. Analyzing SPI communication message

1.5.4 CAN communication Protocol

The CAN serial bus system was created by Robert Bosch in 1986 and is now widely used in cars to simplify wiring with reducing wiring harnesses inside vehicle chassis. Mercedes-Benz and Intel engineers were among the first to work on this system. CAN protocol allows different ECUs embedded in a vehicle to communicate with each other, without a central host computer [25].

ECUs (Electronic Control Units) in passenger cars can include various electrical instruments such as a battery BMS unit, a converter between an electric motor and the battery, the electric motor, regenerative brakes, sensors (such as rain sensors, obstacle detection sensors, etc.), and other electrical instruments that can send and receive signals. These ECUs, also referred to as system "nodes" in some discussions, function like the nervous system of a car, enabling smooth communication and exchange of information. When a CAN message is transmitted, it is received by all nodes. However, each node can determine if the message is intended for them (each message is specifically addressed to a particular node) or not, allowing irrelevant nodes to disregard to interact with the message if necessary.

Figure 15 illustrates how using the CAN protocol can streamline wiring, reducing the complexity of connections among ECUs; otherwise, modern cars would suffer from an excessive amount of wiring.

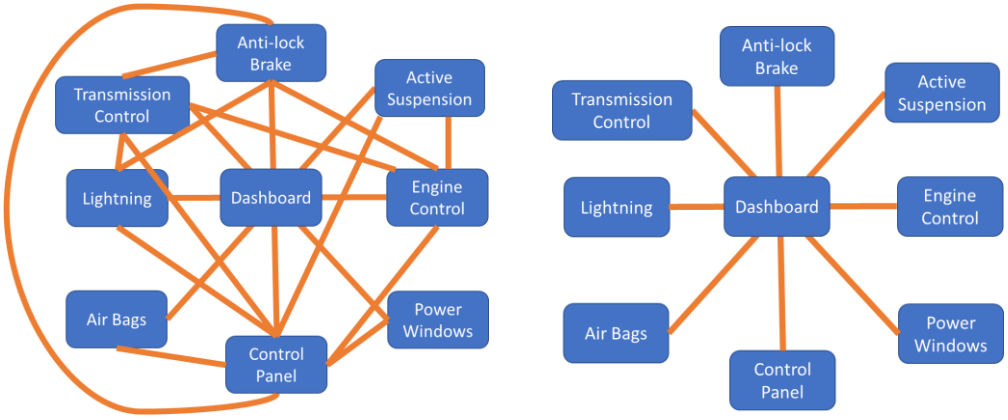


Figure 15. Vehicle cable harness without CAN Protocol (Left) and with CAN Protocol (Right)

Another well-known advantage for CAN protocol is priority hierarchy in message. Each message is assigned as a priority. When two nodes (or ECUs) attempt to send messages at

the same time, the one with the higher priority will be sent first, while the lower priority one will be delayed. This process ensures that the most important message gets through without interruption, helping networks stick to specific timing requirements [26].

The CAN protocol evolved over time with the development of ISO 11898 standards. CAN protocol primarily utilizes two wires: CAN_Low and CAN_High, which are physically twisted pairs, as shown in Figure 16, functioning as binary data channels. In a typical passenger car, CAN cable lengths is around 40 meters by rule of thumb and data exchange speed with this length is 1 Mbit/s. Other maximum cable lengths vary, but these values are approximate [27]:

- 100 meters at 500 kbps
- 200 meters at 250 kbps
- 500 meters at 125 kbps
- 6 kilometers at 10 kbps

In a Controller Area Network (CAN) bus system, the voltage levels for CAN High and CAN Low determine whether a bit is considered a 1 (recessive) or a 0 (dominant):

- Recessive (1): Both CAN High and CAN Low are at approximately 2.5 volts (differential voltage between CAN High and CAN Low is close to zero volts).
- Dominant (0): CAN High is around 3.5 volts, and CAN Low is around 1.5 volts (differential voltage is approximately 2 volts).

These differential voltage levels ensure reliable communication between Electronic Control Units (ECUs) in automotive systems [27]. In Figure 16, ECUs are normally powered (Vcc) from an SLA battery.

As shown CAN twisted pairs in Figure 16, this wiring appeared to work as Capacitive Coupling enhancing Capacitance behavior. This can cause signal integrity to be weakened and rising noise. To damp this unwanted noise, CAN resistors [normally 120 ohm] are embedded on ending parts of CAN bus [27]. In D9 shaped resistor, CAN High and CAN Low are located in pin number of 7 and 2 in order. In this thesis, CAN communication protocol (unlike I2C, SPI and UART) was not applied in automation test setup. However, CAN communication protocol would be for further study in this thesis as discussed in Subsection 11.4.3.

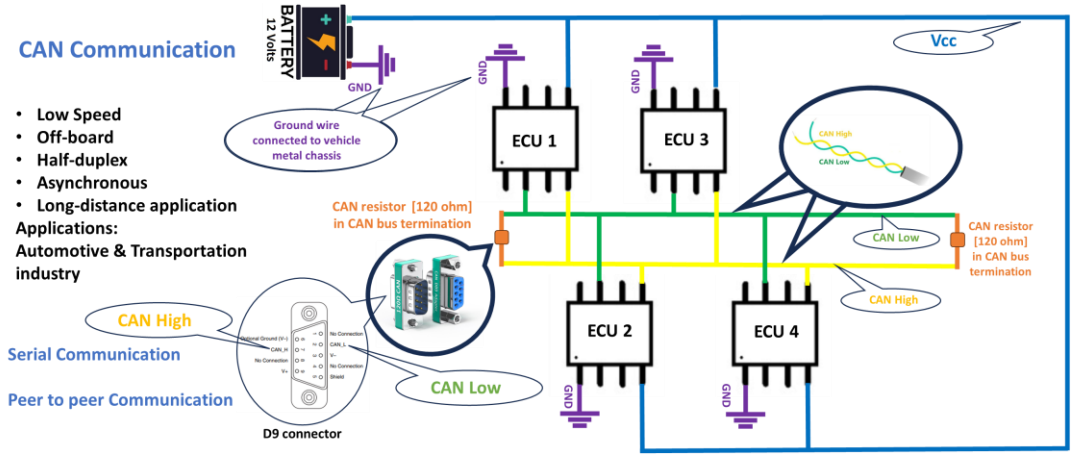


Figure 8. CAN communication Protocol Configuration in automotive

1.5.5 Communication Protocol discussion

In automotive battery industry, I2C is commonly used within PCB chipsets, while SPI is ideal for high-speed communication between multiple chipsets. CAN communication is considered the safest protocol due to its message encoding, although it is the most complex. UART is the simplest and oldest protocol among already-introduced protocols. Typically, I2C, UART and SPI protocols are used for monitoring cell battery conditions on PCB chipsets inside battery packs. In contrast, the CAN protocol not only monitors but also controls battery functions in Battery Management System (BMS) unit. In the upcoming subsection, the tasks performed by the Battery Management System (BMS) will be outlined using the CAN protocol.

Table 2 provides a comparative analysis of existing serial communication protocols used in automotive batteries, focusing on key parameters such as Pin Drive, Maximum Speed, Maximum Peripherals, Communication Method, and Flow Control [16-27]. All the attributes listed in Table 2 were discussed previously, except for “flow control”. UART can have “flow control” features to ensure data is sent and received without loss. Flow control helps manage the data flow between devices, preventing one device from overwhelming the other with too much data. UART includes both Hardware Flow Control and Software Flow Control, but the specifics of these features are beyond the scope of this thesis. I2C requires flow control because of its multi-controller/peripheral architecture and shared bus, utilizing clock stretching and arbitration to manage communication. In contrast, SPI does not require flow control since it operates under the control of a single controller, with simplicity and speed as its primary design goals. CAN,

on the other hand, incorporates flow control to manage a multi-node environment, using arbitration and error handling to ensure reliable and prioritized communication.

Section 3 provides additional details on how SPI and UART were internally integrated into the battery pack's circuit board. The I2C protocol was used for the OLED display, while the CAN protocol was not applied in this thesis. The CAN protocol will be recommended for future study at the end of the thesis.

Table 2. Comparative analysis of communication protocols

	I2C	**UART**	**SPI**	**CAN**
Pin drive	Open drain	Normally one controller & one peripheral	Push-pull	differential voltage levels
Signal lines	2	2	4 (plus 1 for each additional peripheral)	2
Maximum speed	400 kbps in fast mode (3.4 Mbps is possible with high-speed mode)	115,200 bps	No limit (10-100 Mbps is common)	1 Mbps in 40 meters
No. of peripherals	112 with 7-bit addressing (Common) 1024 with 10-bit addressing (Rare)	Normally for 1 peripheral, but more peripherals can be added in some cases, the messages bypass through peripherals	Only limited by number of pins available for CS lines on controller	Fewer than 100 nodes in automotive due to bus length and data rate constraints
Communication method	Multipoint (controllers) to Multipoint (peripherals)	Point (controller) to point (peripheral)	Point (controller) to multipoint (peripherals)	Multipoint to multipoint [without any controller unit but there is in-built controller inside each CAN node]
Flow control	Yes	Yes	No	Yes
Distance	Short	Moderate	Short	Long

1.6 BMS unit tasks in automotive batteries:

The battery BMS unit is mentioned multiple times in this thesis. Here shortly, how BMS functions as the battery’s central processing unit will be explained. BMS unit is providing

intelligent control over the battery's condition. BMS unit helps prevent potential hazards, overheating, fires, or a high surge current by opening of battery pack contactors to disconnect electrical circuits when necessary. The BMS unit efficiently balances SOC levels among individual cells to avoid overcharging or undercharging any single cell. BMS unit manages thermal conduction across the battery with controlling available cooling system. BMS unit also communicates information to other vehicle ECUs using the CAN protocol [28].

The BMS is able to broadcast "Read" or "Write" messages to all battery nodes without addressing them individually, to expedite the emergency message transmission. Network broadcasting is not solely a BMS function; another unit with broadcasting capabilities is introduced in Subsection 3.6. However, only the battery BMS can perform "CAN" network broadcasting across battery nodes. In the battery industry, a battery without a BMS unit is casually called a "brainless battery pack". The battery pack used in this thesis is a brainless battery. Additional information about the EV battery pack tested in this thesis can be found in Section 3.

1.7 Electrical Standards used in Automotive battery testing

In prior subsection, certain battery attributes such as cell chemistry and cell architecture have been discussed as influencing battery testing criteria to some extent. However, the primary factor impacting automotive battery testing criteria is the testing standards agreed upon by the manufacturer and the customer. This section briefly introduces some of the key electrical standards utilized in automotive battery testing.

The selection of electrical standards for quality testing in any battery testing setup depends on the specific application of the battery and the acceptance criteria set by the customer. Commonly used electrical standards in automotive battery testing include those established by organizations such as the United Nations (UN), the International Electrotechnical Commission (IEC), the International Organization for Standardization (ISO), the Society of Automotive Engineers (SAE), and Underwriters Laboratories (UL).

Since the battery in this automated test setup is a prototype, the testing standards evolve over time for isolation, equipotential, and dielectric tests. This thesis does not delve into

the rationale behind specific battery testing standards, as that is considered a matter between manufacturers and customers. Instead, it primarily concentrates on the implementation of a proposed testing procedure, without addressing the reasons behind the chosen testing standard limits. Table 3 in the thesis shows the conditions for performing the three tests on the battery pack, along with the acceptable criteria for each test.

Section 4 details the software tools used in the thesis's automated test bench. Subsection 4.2 discusses NI LabVIEW, while Subsection 4.4 covers NI TestStand. In Table 3, each battery pack test (highlighted in green) is linked to its specific LabVIEW program. The test conditions (highlighted in orange) in Table 3 are inputs for TestStand, which integrates with the corresponding LabVIEW program. The acceptable requirements (highlighted in blue) in Table 3 are the outputs from the TestStand environment and are also integrated with their respective LabVIEW programs.

Subsections 5.6, 5.8, and 5.9 will offer more detailed information about the equipotential test, isolation test, and dielectric test, respectively. The appendices will include specifics on the LabVIEW programs, TestStand reports and graphical figures over time for each test. As previously mentioned in Subsection 1.2, both polarities of the battery pack must be examined for the isolation and dielectric tests. Table 3 equally addresses both battery polarities in the (isolation and dielectric) test conditions and acceptable requirements.

Table 3. Test conditions performed in automation test setup against the battery pack under test

Battery pack test	Test conditions			Acceptable Requirement
	Test time duration	Imposed Voltage or Current	DC or AC current	
Equipotential [ground bonding] test	5 seconds	8 Ampere	AC	Conductivity resistance of metal parts between 0 and 5 milli ohm
Isolation [electrical insulation] test	10 seconds	500 Voltage	DC	Electrical insulation between 100 and 1000 Mega ohm
Dielectric [High voltage withstands] test	2 seconds	1000 Voltage	DC	Leakage current between 0 and 2 milli Ampere

1.8 Purpose of the thesis

The thesis aims to develop an automated test bench (shown in Figure 18, right side) for a unique prototyping battery pack (shown in Figure 19) that operates without human intervention. This test bench includes automated test sequence measuring environmental conditions (humidity and temperature), battery cell verification (reading voltage cell and internal temperature sensors), writing to EEPROM memory embedded in the battery's circuit, equipotential test, isolation resistance tests, and dielectric tests. These tests are performed using NI LabVIEW and Arduino software tools. Each NI LabVIEW VI (Virtual Instrument) is integrated into NI TestStand. Upon completing the measurements, the automated test bench uses NI TestStand to generate the test results. The automated test bench employs the GW Instek GPT-9804 safety tester hardware (shown in Figure 17) for quality measurements such as isolation resistance, dielectric tests, and equipotential tests.

Subsection 1.7 had mentioned that LabVIEW programs for each quality test are available. These LabVIEW programs were actually downloaded from the GW Instek (safety tester hardware) company's website (referenced in [29]). Sections 4 and 5 will offer more information on these LabVIEW programs and other software tools such NI TestStand and Arduino. Additionally, Section 5 provides a visual demonstration of how the GW Instek GPT-9804 safety tester device is connected to the battery pack for each specified test.

Figure 9. GW-Instek GPT-9804 Safety tester instrument applied in test automation setup

The automation test setup also involves several IT components to support: 1. Azure cloud storage for storing test results via the SMB protocol, 2. Remote monitoring using RDP with a third-party server host, and 3. Remote data transferring using FTP protocol. Azure Cloud was selected because IONCOR company utilizes this cloud service for their battery data projects. NI TestStand saves the generated test reports on a virtual drive designated as the Azure drive for cloud storage (detailed in Section 7). To facilitate remote operations like data transfer, monitoring, and editing test sequences, the PC test setup is designed for remote access via the RDP protocol (detailed in Section 8), with a user activity tracking feature. The FTP protocol is used for transferring data between different users (detailed in Section 9). All IT components, including Azure storage, remote monitoring, and remote data transfer, are secured through a VPN (detailed in Section 6).

The next chapter presents crucial background information relevant to the thesis's purpose. It features a performance comparison between the proposed automated setup and the existing automation setup used at the IONCOR prototyping factory. Additionally, it explores the motivations behind the thesis's purpose and what inspired the author to create an automated test bench for the prototyping automotive battery pack.

2 Background

In Subsection 1.4.5, it was mentioned that this thesis presents an automation setup for testing a unique battery pack. There is an existing automation testing station located in the IONCOR prototyping factory that automatically tests the unique battery pack, which inspired the proposed test bench in this thesis. The existing battery pack tester station, shown in Figure 18 (left), was built by an external Finnish third-party company. When the IONCOR prototyping factory was established a few years ago, only a few specialists were available, and the R&D department was too small to build such a test bench. Therefore, it was decided that a third-party company would construct the existing battery pack tester station. Nowadays, IONCOR Oy has hired more specialists, so it is expected that more test benches will be built by IONCOR personnel without the need for external contributions. Figure 18 (right) illustrates one of the test benches that is anticipated to be constructed as an alternative.

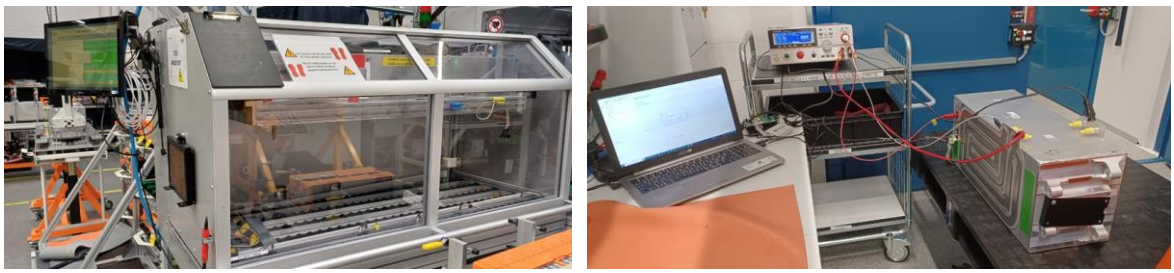


Figure 10. Existing battery pack tester (left side) versus proposed battery pack tester (right side)

The existing bench employs costly controllers and needs ongoing maintenance and development by external Finnish third-party experts as the battery development process progresses. As prototyping batteries evolve over time, their testing stations are upgraded with appropriate software and hardware tools to meet the new requirements.

Collaborating with the existing battery pack testing station (made by a third-party) at IONCOR is challenging due to the lack of local expertise. Engaging external third-party experts for future development is more expensive compared to having local specialists at IONCOR handle the development. Therefore, there is a need for local expertise to establish a local testing bench. As a result, more existing test benches will be replaced with internal ones created by local IONCOR experts, and future test benches for upcoming projects will also be built by local specialists without relying on external contributions.

The next subsection compares the features of the existing battery pack tester with the proposed one. It also discusses the inspirations behind this thesis topic.

2.1 Comparing structure between existing setup and thesis setup

Table 4 provides a detailed comparison between the existing testing setup and the proposed thesis testing setup. The details of feature rows 1 to 7, which are common in both test benches in Table 4, will be further elaborated in Subsections 5.1, 5.6, 5.8, 5.9, 5.3, 5.4, and 5.5. Regarding feature 8 in Table 4, the DCIR (Direct Current Internal Resistance) test requires an expensive device with a high-power bidirectional converter (sometimes up to 400 DC Amperes) for short transient periods of discharging or charging the battery pack. By measuring the voltage drop during discharge or the voltage increase

during charging over a short time, the internal resistance of the battery pack can be calculated. This DCIR test was excluded from the thesis setup. Features 9 and 10 are further explained in Section 4.

In Table 4, feature number 11 refers to the safety hardware device, which for the thesis setup introduced in Subsection 1.8 is GW Instek, while the existing setup uses Chroma. This difference in hardware directly affects the LabVIEW programs for each quality test. Typically, each safety tester hardware brand provides its own LabVIEW programs supported by the hardware on their websites. For the thesis setup, it was decided to use a different safety tester hardware to tackle new challenges with new LabVIEW programs, which will later be integrated with TestStand. In feature row number 12, the thesis setup includes a graph over time for measurements in the TestStand test results for each quality test. These timeline graphs enable easy monitoring of measurements within the test duration. Additionally, as mentioned in feature number 12, there are alarms with LED and buzzer to inform testing operators upon the completion of the test cycle.

Table 4. Comparison of the existing test bench with the proposed test bench for the battery pack

N.	Features	Thesis setup	Existing setup
1	Saving battery Serial Number in test report	Yes	Yes
2	Equipotential test	Yes, but only in some cases needed	No
3	Isolation test	Yes	Yes
4	Di-electric test	Yes	Yes
5	Reporting environment conditions [Temperature & Humidity]	Yes	No
6	Reading voltage and temperature sensors of battery pack under test	Yes	Yes
7	EEPROPM writing	Yes	Yes
8	DCIR test	No, due to expensive testing device	Yes
9	Softwares in use	LabView, TestStand, Arduino	LabView, TestStand
10	Controllers in use	Arduino [cheap]	Texas Instruments & National Instruments [expensive]
11	Safety testing device	GW-instek	Chroma
12	Other features	1. Providing graphs for each test report 2. Activating sound and LED alarm when the test finished	No
13	Battery Connection fastening	Manual by testing operator [safety reminder available in test sequence (see Subsection 5.2)]	Fastened by automated pneumatic actuators [not recommended in prototyping level]
14	Manufacturing Party	IONCOR Oy	Third party Finnish company

In Table 4, feature row number 13, the thesis test bench utilizes banana and alligator clip connection cables to connect the safety test hardware to the battery pack (battery polarity terminals and chassis). In contrast, the existing test bench uses automated pneumatic actuators to establish these connections. These pneumatic robotic actuators are not recommended during the prototyping phase due to safety reasons. As both the test bench and the battery pack are in development, programming errors can lead to unintended accidents with the pneumatic actuators. Additionally, lack of frequent maintenance sessions can sometimes result in missed calibrations or even damage to the battery. Since the prototyping factory prioritizes development quality over quantity production (as mentioned in Section 1), the testing process is not rushed, and the testing sequence is typically configured as a semi-automated model. Given that quantity and time are not prioritized during the prototyping phase, securing connections with the assistance of a testing operator may be a safer choice than using pneumatic actuators.

2.2 Motivations behind thesis topic

There are several motivations behind this thesis topic that benefit both the author's career and the company (IONCOR Oy), where the author is currently employed. All motivations are listed here:

1. The thesis topic and its focus are directly aligned with the author's current role as a battery test engineer.

2. Valuable educational resource for the author's future career as a test setup developer

3. Learning how to build automation setup and have a local testing station (local knowledge in IONCOR Oy)

4. A third-party testing station (like the existing setup) requires third-party experts for future development or maintenance, which would be costly for IONCOR Oy.

5. Familiarity with integration of various software and hardware into a unique system

6. Learning test procedure logics for automotive prototyping batteries

7. Familiarity with different testing software tools such LabView and TestStand

8. Familiarity with famous communication protocols such SPI, I2C, and UART in automotive and battery industry

9. Understanding how to simulate battery behavior (as discussed in upcoming Section 3)

10. Learning how to program a battery controller based on battery behavior and features

3 Knowledge About the Battery Pack Under Test

Subsection 1.4.5 provided basic information about what a battery pack is and its components. This section delves into the specifics of a unique battery pack appropriate with the proposed test bench in this thesis. In Subsection 1.4.5, it was mentioned that a Battery Management System (BMS) unit is integrated when multiple battery packs are assembled, so a single battery pack does not have a BMS unit. Consequently, a single automotive battery pack lacks internal CAN communication. CAN protocol communication for automotive EV-hybrid batteries is normally defined within the battery BMS unit. Therefore, a battery pack typically uses communication protocols other than the CAN protocol for data exchange. In this section, information which protocols are used in this battery pack will be introduced.

Following this, the internal components of a unique battery pack are described. Details about the CMC board within the battery module are included, along with practical information about the CMC board's EEPROM. Finally, the section discusses how the hardware simulation of the battery pack is conducted and the necessity for this simulation.

3.1 Battery pack connections with safety tester instrument

Figure 19 shows the battery pack compatible with the thesis test bench. Before connecting this battery pack to the safety tester device shown in Figure 17, the busbar bridge on the right side of the battery pack should be shortened. This busbar bridge links two internal battery modules together for isolation and dielectric testing. When connecting the respective banana plugs of the safety tester device to the battery terminals, the battery pack's polarities should be noted. Additionally, alligator clip connectors are attached to the exterior metal shield of the battery pack for equipotential test. Additional visual details regarding these connections for each quality test will be provided again in Subsections 5.6, 5.8, and 5.9. To check the voltage of individual cells and the internal temperature sensors within the battery pack, communication ports should also be connected.

Figure 11. Battery pack under test used for proposing thesis test bench

3.2 Battery pack components

The unique battery pack shown in Figure 20 is made up of two battery modules, also known as cell-stacked devices. The prior subsection had mentioned these two modules are shortened via a bus bridge when isolation and dielectric tests begin. As previously mentioned in Subsection 1.4.5, each module contains a CMC board. Therefore, there are two CMC boards in Figure 20 made by Texas Instruments. Each CMC board includes a microprocessor that monitors 19 battery cells and 6 temperature sensors within the module. Additionally, an EEPROM (Electrically Erasable Programmable Read-Only Memory) is integrated into each CMC board. In summary, the battery pack, consisting of two modules, has a total of 38 battery cells, 12 temperature sensors, two microprocessors, and two EEPROM memories. Next subsection discusses more information regarding CMC board embedded on battery module.

Battery module device:
- Communication board created by Texas Instrument
- Includes 19 battery cells
- Owns 6 temperature sensors
- Operating one microprocessor
- Having one EEPROM memory on board

Battery pack under test:
- Includes two cell stacked devices
- Includes totally 38 battery cells
- Owns totally 12 temperature sensors
- Operating two microprocessors
- Having two EEPROM memories on board

Figure 20. Components of the battery pack under test

3.3 Battery pack's CMC board

The specifics of the prototyping CMC boards produced by Texas Instruments are shrouded by a confidentiality barrier set by IONCOR. For reading voltage cells and internal temperature sensors, differential signaling through the Modbus protocol is utilized. Due to the limited details on the CMC board used in the battery module, this thesis will not delve deeply into the Modbus protocol. Before addressing Modbus, a brief overview of the Open Systems Interconnection (OSI) layer mapping is necessary.

3.3.1 Introduction to OSI layer mapping

The OSI layer mapping outlines the bit (frame) layers and data payload for each serial communication protocol discussed, as shown in Figure 21. Reference [30] offers a detailed explanation of these layers, so their exhaustive definition is omitted from this thesis. This thesis emphasizes the correlation of I2C, SPI, CAN, and UART with these layers. The mentioned serial communication protocols cover both the Physical Layer (Layer 1) and the Data Link Layer (Layer 2).

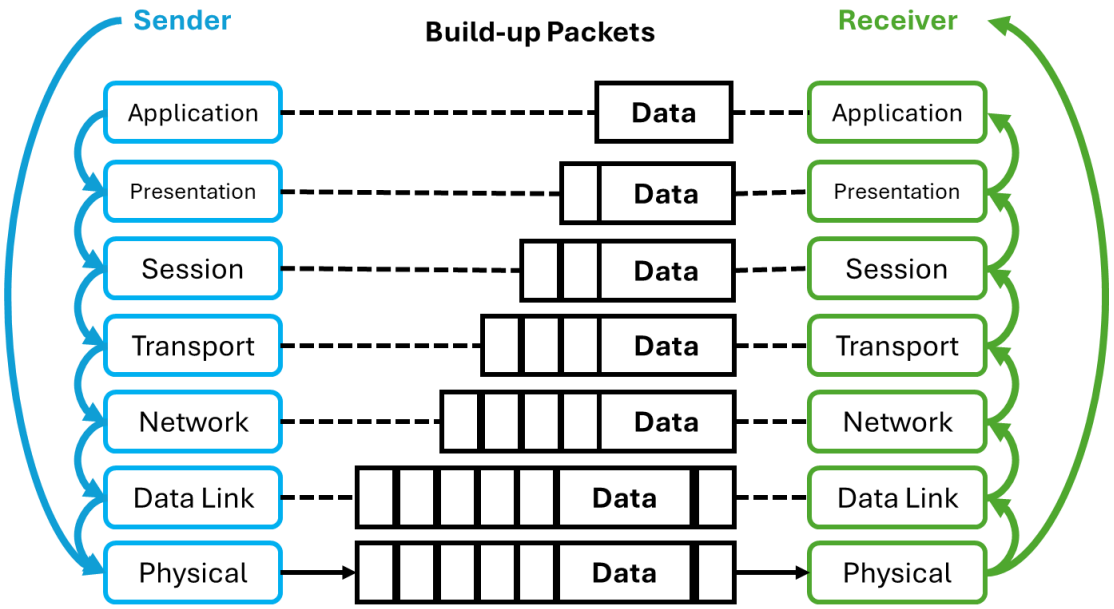


Figure 21. OSI layer mapping in general data payload

Modbus is available in two formats: 1) serial and 2) Ethernet (TCP/IP). Subsection 7.4 will provide further details on the Transmission Control Protocol (TCP) and Internet Protocol (IP). This thesis focuses solely on the serial format of Modbus, which includes two communication formats within the Data Link Layer (Layer 2): RTU (Remote Terminal Unit) and ASCII (American Standard Code for Information Interchange). The difference between RTU and ASCII will be explained in Subsection 3.3.4.

In contrast to the serial communication protocols covered in Section 1, which consist of only two layers, the Modbus serial format includes three layers: the physical layer, the data link layer, and the application layer. **The application layer introduces a complexity that required a "Poll Request" feature for the battery CMC board.** Thus, an explanation of the OSI layer mapping was necessary before addressing the CMC boards embedded in battery modules.

The Physical Layer defines the physical connection between devices, including hardware elements like wires. The Data Link Layer is responsible for data transfer between devices and error detection/correction. Bits are typically appended to the payload data for error handling, such as parity bits and Cyclic Redundancy Check (CRC) bits. However, these bits were omitted when introducing serial protocols in Section 1 due to simplification. More information on parity bits and CRC bits can be found in Subsection 3.6.1.

Additionally, CR/LF bits in UART introduction (Subsection 1.5.1) and the ACK [Acknowledge] or NACK [Not Acknowledge] bits, starting and ending bits introduced in I2C introduction (Subsection 1.5.2) are parts of the Data Link Layer. The placement of these Data Link Layer bits once more will be reviewed in Figures 22, 23, and 24.

As depicted in Figure 21, once the data transitions from the Data Link Layer (Layer 2) to the Network Layer (Layer 3), no further bits are appended to the data payload, though prefixing bits are present. Figure 21 also shows that the Application Layer involves a pure data payload between the sender and receiver. The Application Layer (Layer 7) is intimately connected to the **firmware** of the microcontroller or microprocessor.

3.3.2 Firmware introduction

Firmware, often described as "software for hardware", which is a program code embedded in hardware devices to ensure their proper functionality. It is updated regularly to fix common issues, add new features, or improve compatibility with emerging technologies. Unlike regular software, which can be easily modified or updated, firmware is usually stored in non-volatile memory (such as ROM, EPROM, or flash memory) and is not meant to be frequently changed. In the battery industry, flashing a CMC board indicates that the firmware has been successfully uploaded to the board's flash memory. Firmware serves as an intermediary, translating data from Layer 7 to the hardware (in this case, the CMC board) [31].

When Layer 7 is absent in certain communication protocols, the firmware translates data instead of Applicable Layer. For instance, in the UART protocol, Layer 2 is the final layer, and data from this layer is passed to the firmware for translation. In Modbus, during this firmware translation process, the firmware requests data from the hardware using a "Polling Request" in Layer 7. More details about "Polling Request" in Modbus can be found in Subsection 3.6.

3.3.3 OSI mapping in UART, I2C, SPI and CAN

UART, I2C, SPI, and CAN are protocols that mainly operate at the Physical and Data Link Layers (with CAN also partially operating at the Transport Layer). Unlike Modbus, these

protocols do not inherently define an Application Layer. The duties of the Application Layer in protocols like UART, I2C, SPI, and CAN is left up to the firmware.

UART protocol serial communication in OSI layer mapping [17]:

UART Physical layer (Layer 1):

- Defines electrical signalling (baud rate) and timing for serial communication.
- Includes Tx (transmit) line, Rx (receive) line

UART Data Link layer (Layer 2):

- Frames data with start bits, stop bits, and optional parity for error detection.
- Ensures byte-level synchronization and basic error detection.

UART Firmware:

In UART, the firmware operates without layer 7. The UART controller polls the peripheral by transmitting data and awaiting an acknowledgment or response. The controller constantly monitors the status of the UART lines to determine if data is available for reading or if the transmit buffer is empty. Figure 22 illustrates 7-character bits as a data payload for the firmware, along with start and stop bits and a parity bit for use in the Data Link Layer (Layer 2) [17].

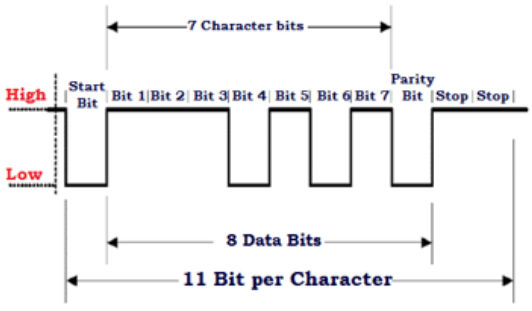


Figure 22. UART data frame structure [17]

I2C protocol serial communication in OSI layer mapping [18-20]:

I2C Physical Layer (Layer 1):

- Defines the electrical characteristics of the two wires (SCL for clock and SDA for data).
- Specifies voltage levels, pull-up resistors, and the signal protocol.

I2C Data Link Layer (Layer 2):

- Manages addressing bits (7-bit or 10-bit addressing) and ensures data integrity with ACK/NACK signalling.
- Protocol defines start/stop conditions and data frame format.

I2C Firmware:

In I2C, firmware works in absence of Layer 7. I2C controller initiates communication by sending a start condition followed by the hexadecimal address of the peripheral device. The controller then polls the peripheral by reading or writing data. The polling is managed by the controller device itself, which continuously checks the status of the I2C bus to see if the peripheral is ready to communicate. Figure 23 shows the primary data payload which needs to be accessed to firmware is 8-bit data (from D0 to D7). Other bits such Start, Stop, ACK, R/W (in another words, read or write), 7-bit addressing (from A0 to A6) are concerned with layer 2.

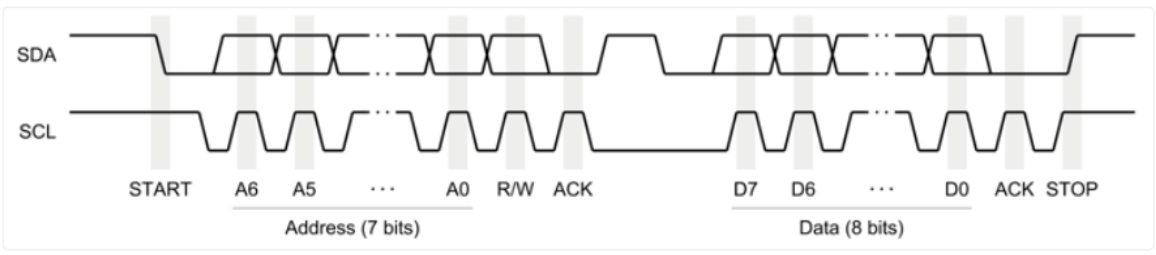


Figure 23. I2C data frame structure [24]

SPI protocol serial communication in OSI layer mapping [20-24]:

SPI Physical Layer (Layer 1):

- Defines the electrical signals for COPI, CIPO, SCK, and optionally CS lines.
- Specifies voltage levels and timing relationships.

SPI Data Link Layer (Layer 2):

- Implements basic data framing and synchronization between the controller and peripheral(s).
- No inherent addressing or error-checking mechanism—relies on the application.

SPI Firmware:

In SPI, the controller device manages communication by generating the clock signal. The controller polls the peripheral by sending data and checking for acknowledgments. The controller continuously monitors the status of the SPI bus to ensure data is being transmitted and received correctly. Rarely error detection bits are in use for SPI communication for Data Link Layer (Layer 2). Therefore, Figure 14 can still be a good example for the data frame configuration. **Normally** in SPI, there are no starting, ending, CRC, or parity bits.

CAN protocol serial communication in OSI layer mapping [25-27]:

CAN Physical Layer (Layer 1):

- Defines the electrical characteristics of the CAN bus (differential signals using CAN-H and CAN-L).

CAN Data Link Layer (Layer 2):

- Handles message framing, arbitration, and error detection/correction.
- Uses CRC for data integrity and defines identifiers for addressing.

CAN Transport Layer (Layer 4) [Not Common]:

- Related to extra 18-bit Arbitration ID in Figure 24 for Extended CAN protocol

CAN Firmware:

In CAN, the built-in controller within any CAN node polls other nodes by sending a request message on the CAN bus. Nodes on the network respond to the request if they are addressed or possess relevant data to transmit. The polling process is managed by the CAN controller, which is responsible for message arbitration and error checking. This controller is integrated within the CAN node itself, with no need for an external unit as a controller (unlike other serial protocols), as mentioned earlier in Table 2. For instance, there are built-in CAN controllers within a battery BMS unit, a vehicle's dashboard unit and any CAN nodes embedded in a vehicle.

Figure 24 highlights various components, including the Start of Frame (SOF) bit, End of Frame (EOF) bit, CRC bits, ACK bit, and arbitration ID (11-bit for the standard CAN protocol). It also shows Data Length Code (DLC) bits, which indicate the number of bytes used in the CAN data message. In Figure 24, the DLC indicator should be "8", if 8 bytes of CAN byte data was intentionally reserved. All these bits, except for the 8-byte data payload and additional 18-bit arbitration ID, pertain to the Data Link Layer (Layer 2). The additional 18-bit section for the Extended CAN protocol is related to Layer 4. The firmware processes the 8-byte data payload situated before the CRC bits [26].

Figure 24. Standard CAN frame structure [26]

3.3.4 Modbus in OSI layer mapping

Table 5 summarizes the information from the previous subsections. Table 5 explains the Physical Layer of Modbus (serial), can use either RS232 or RS485 [32]. Modbus was initially designed for RS232, a point-to-point connection interface. In Figure 17, the safety

tester device can be connected to a PC for remote control using either a USB or RS232 connection. The USB option is chosen for this purpose, which will be explained further in Section 4. However, using RS232 to connect multiple peripherals to one controller requires multiple links, which can be inconvenient and costly. To enhance connectivity among multiple peripherals with a single controller, a point-to-multipoint connection is more efficient.

RS485 supports point-to-multipoint communication [33]. Reference [34] provides detailed explanations of point-to-point and point-to-multipoint connections. RS485, a standard utilizing differential signaling, ensures reliable communication in noisy environments over longer distances [33]. "RS" in RS485 and RS232 stands for "Recommended Standard," a recommendation by the Electronic Industries Association (EIA) to ensure compatibility and interoperability between different devices and systems. RS485 transmits data using two complementary signals, one being the inverse of the other, reducing noise and enhancing signal integrity over long distances.

Table 5. OSI layer mapping in communication protocols introduced in this thesis.

ISO Layer N.	7 Layer OSI Model	UART (Serial)	I2C (Serial)	SPI (Serial)	CAN (Serial)	Modbus (Serial)	Modbus (TCP/IP)
7	**Application**	---	---	---	---	Modbus application	Modbus application
6	**Presentation**	---	---	---	---	---	---
5	**Session**	---	---	---	---	---	---
4	**Transport**	---	---	---	---	---	TCP
3	**Network**	---	---	---	---	---	IP
2	**Data Link**	Start, stop & optional parity bits	ACK/NACK signaling, Start, stop bits	SCK (clock) signal and framing data packets	Framing, arbitration, ACK bit, CRC bits & etc	Modbus (RTU or ASCII)	Ethernet
1	**Physical**	(Rx, Tx)	(SCL, SDA)	(COPI, CIPO, SCK, CS)	(CAN-H, CAN-L)	(RS232) or (RS485)	

An RTU Modbus (serial communication) message is composed of two primary components, illustrated in Figure 25 with distinct colors. The CRC bits and address field bits (in blue) pertain to the Data Link Layer (Layer 2). Meanwhile, the Function Code and Data (in green) are part of the Modbus Protocol Data Unit (PDU), associated with the

Application Layer (Layer 7). The firmware needs to interpret certain Function Codes, which are numerical codes that specify whether it should read or write data, and how to perform these operations. Reference [32] offers detailed some examples for the key Function Codes. Modbus ASCII format has two extra parts namely starting and ending characters. In some certain applications, RTU format can include Starting or Ending characters, for example in Subsection 3.6.

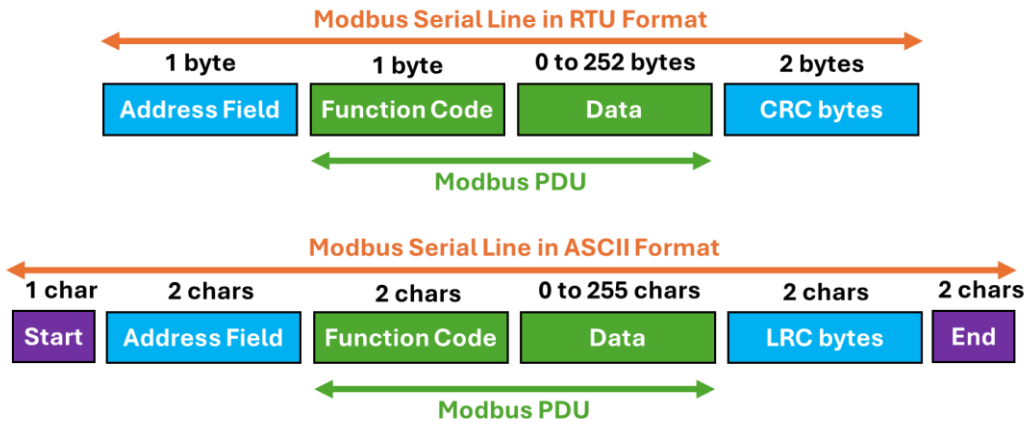


Figure 25. Serial Line Modbus frame structure in both RTU and ASCII formats

In upcoming subsections, it will be highlighted that the unique battery pack comprises two battery modules connected via Modbus in an RS485 Physical Layer using **Linear Daisy Chain Topology**. Additionally, detailed information on how Address field bits (earlier shown in Figure 25) work in RTU Modbus will be presented.

3.4 Linear Daisy Chain Topology

Reference [34] describes various network topology architectures. Serial communication topologies can adopt the same architectural designs used in internet or bus networks. For instance, the Bus topology mentioned in [34] can be applied to the CAN serial communication protocol. In a daisy chain linear topology, devices are connected through a two-way link. Each client requires two network ports: one for receiving (input) and one for transmitting (output). In this setup, data flows in one direction. When a peripheral shown in Figure 26 expects a message, the data packet must travel through the entire chain to reach its destination. This means that other irrelevant peripherals must pass the message from their terminals to ensure it reaches the intended peripheral.

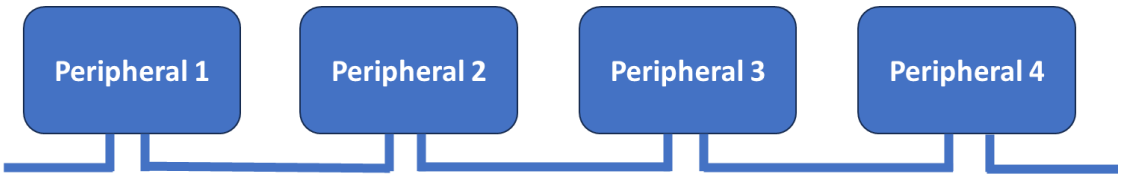


Figure 26. Linear Daisy Chain Topology

Figure 26 illustrates the positioning of peripherals in a serial communication setup. The serial communication controller is connected from one end of this linear configuration. It is important to note that in this topology, bidirectional communication cannot occur simultaneously. After the entire data packet passes through the topology line, the controller can decide whether to reverse the data flow direction or continue in the same direction. Before communication with these peripherals, the serial communication controller should know peripheral addresses within the Linear Daisy Chain topology. To establish the peripheral addressing, a routine "Auto Addressing" needs to be performed by the serial controller. Subsection 3.6 provides detailed information on "Auto Addressing".

3.5 Battery pack's EEPROM

EEPROM (Electrically Erasable Programmable Read-Only Memory) is a form of non-volatile memory that retains data even after power is lost. It supports both reading and writing, making it versatile for applications requiring periodic data updates, like device settings or sensor adjustments. Unlike Flash memory, EEPROM allows writing data one byte at a time, making it ideal for small-scale storage in embedded systems. Widely used in automotive, industrial, and consumer electronics, EEPROM is appreciated for its low power consumption and durability, making it perfect for long-term data storage [35].

3.5.1 EEPROM application

EEPROMs have memory volumes ranging from 128 bits to 1 Mbit and are compatible with industry-standard communication protocols like I2C and SPI serial communications. Figure 27 illustrates an SPI-based EEPROM connected via GPIOs to a battery CMC board's microprocessor. (General Purpose Input/Output) GPIO port is being introduced in the next subsection. EEPROMs operate at voltages between 1.7V and 5.5V and function

effectively across a wide temperature range from −40°C to 150°C [35]. Sensitive data being stored in CMC board's EEPROM are: battery cycle life, battery barcodes, battery contactor's life counters, calibration information, operating temperature ranges, and battery safety parameters like minimum and maximum customer SOC (State of Charge) and nominal cell voltage. Subsection 5.5 discusses how the battery barcode will be stored in the EEPROM memory of battery modules. Figure 27 shows SDI and SDO, also known as COPI and CIPO, as explained earlier in Subsection 1.5.3.

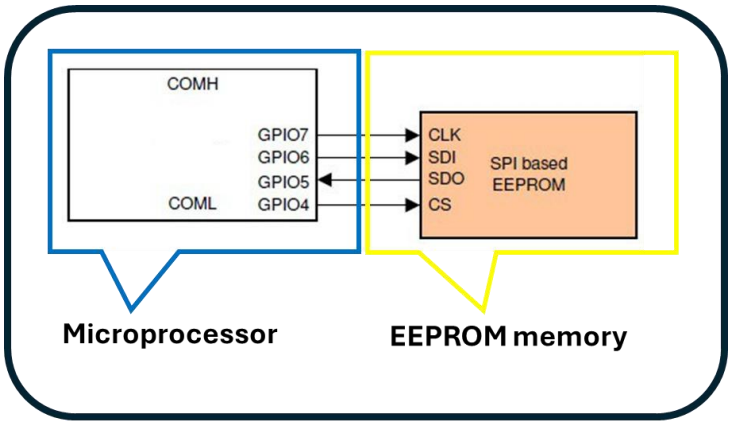


Figure 27. Each CMC board of a battery module is equipped with two embedded peripherals

3.5.2 GPIO introduction

General-purpose input/output (GPIO) is a versatile digital signal pin found on any integrated circuit or electronic circuit in all industry fields, such as a battery module's CMC board. These pins can be used as inputs, outputs, or both, and are controllable by software. As their name suggests, GPIO pins serve a "general" purpose and do not perform any specific function. They are embedded in microprocessors and can be configured as inputs or outputs. GPIO pins can be set up via firmware by embedding developers to interact with various serial communication protocols. In some microprocessors, GPIO pins can also function as analog pins, thanks to the Digital-to-Analog Converter (DAC) tool [36]. In Figure 27, the embedding developers have defined GPIO4, 5, 6, and 7 as SPI terminals, with GPIO4, 6, and 7 serving as output pins and GPIO5 as an input pin for the microprocessor.

In the CMC board's battery module, numerous GPIO pins connect to the microprocessor, allowing for various data transmissions. Some GPIO pins transmit temperature

measurements from sensors to the microprocessor, while others continuously send voltage cell values from the battery cell voltage bus bar. According to Figure 20, there are 19 output GPIO pins connected to the voltage cell bus bar to transmit voltage values to the microprocessor, and 6 output GPIO pins connected to internal temperature sensors to send temperature values. Additionally, 4 GPIO pins are dedicated to EEPROM communication.

3.6 Comprehensive information on Battery Pack under test

With all basic information for peripherals, OSI layers, Modbus and GPIO, now battery module CMC board specification and battery pack details can be wrapped up easier. In Subsection 3.2 was mentioned, CMC boards for battery modules manufactured by Texas Instruments (TX). Since CMC boards are considered peripherals, there should be a controller for communication with these peripherals. The controller can be battery BMS unit or an TX controller matching with these TX peripherals. Figure 28 indicates the TX controller schematic and schematic of battery module CMC board connected with each one and controller.

The communication protocol between EEPROM and microprocessor inside a CMC board is SPI and communication protocol between microprocessor of each battery module is Modbus in differential signaling RS485 physical layers. In Figure 28, the first microprocessor for the first battery module and the second microprocessor for the second microprocessor were named “S1” and “S2” in order. The TX controller has the same microprocessor (named “Base, B0”) and same EEPROM memory used in battery module’s CMC board. Also, TX controller uses internal UART protocol between its own Microcontroller Unit (MCU) and its own “B0” microprocessor. As illustrated in Figure 28, the TX controller includes three peripherals: the MCU (or Host unit in another name), the B0 microprocessor, and the EEPROM of the B0.

The communication topology in use between TX microprocessors of CMC boards and TX controller is linear daisy chain. “B0” microprocessor can communication with S1 and S2 with two twisted pair wires “COMH” (High side or contractually in another name, North) or “COML” (Low side or contractually in another name, South). In other words, when

mentioning North or South direction, it indicates which "COMH" port or "COML" port of the "B0" microprocessor was chosen for communication by MCU unit.

Figure 28 indicated only North side direction is currently in communication. MCU or (host) unit can decide whether to reverse the data flow direction or continue in the same direction; however, when changing direction, no package data should be transmitted across the daisy chain topology. After the entire data packet passes through the linear topology line, MCU can decide for changing direction. In appendix A, there is a figure showing how with Reverse Broadcast Write Command, MCU unit can change communication direction. Specifying communication direction by MCU is one step before Auto Addressing process to be done by MCU unit. Next subsection is about this Auto Addressing.

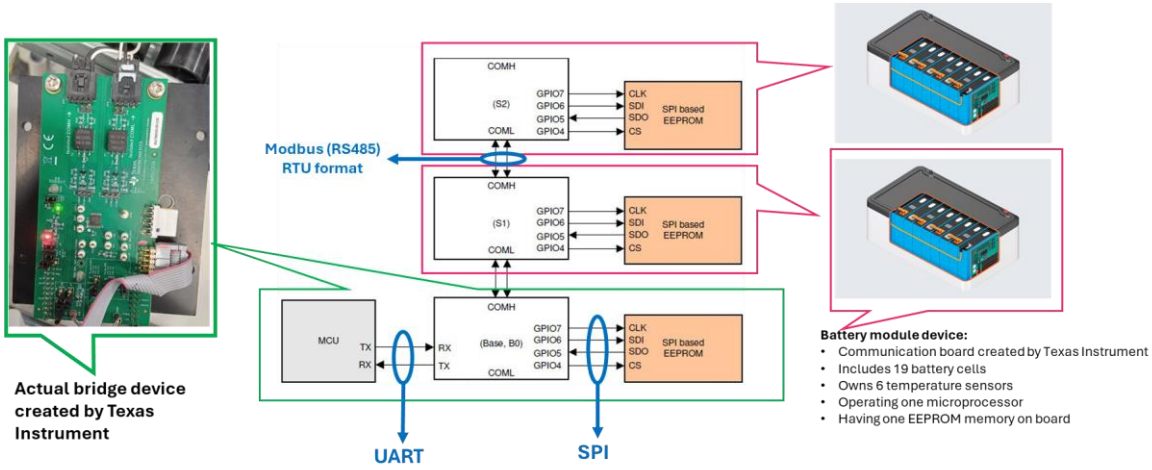


Figure 28. Embedded peripherals on CMC board for each battery module and respective controller

Before explaining regarding Auto Addressing process and how MCU unit can change communication direction, a glimpse to Modbus messages between MCU to multiple peripherals or MCU to individual peripheral or a peripheral to another peripheral need to be disclosed. Table 6 represents Modbus construction performing between CMC modules and CMC controllers (or even battery BMS). The Modbus format is RTU introduced in Figure 25 with extra initialization byte (which only ASCII format includes it) in the beginning. In some RTU Modbus application, initialization and ending bytes might be needed like Table 6. In Table 6, the controller does not need to reference peripheral addresses for **broadcast messages** (read or write) because of their significance and urgency.

Table 6. Modbus message formats between battery modules' peripherals and the controller

Command Frame	Frame segmentation	Data	Comments	Communication flow
Broadcast Read	Initialization Byte	0xC0	Always 0xC0	From host controller to all peripheral units in daisy chain without individually addressing one by one
	Device ID Address	--	No address byte is sent in broadcast mode	
	Register Address	0x0215	Start with address 0x215	
	Data	0x0B	Send 12 bytes worth of data back (register contents from 0x215 to 0x220).	
	CRC	0xD2B3	16-bit CRC calculation	
Broadcast Write	Initialization Byte	0xD3	Writing four bytes to all the devices	From host controller to all peripheral units in daisy chain without individually addressing one by one
	Device ID Address	--	No address byte is sent in broadcast mode	
	Register Address	0x0100	Start with address 0x100	
	Data	0x02B778BC	Write four bytes to registers 0x100-0x103 to all devices	
	CRC	0x6A33	16-bit CRC calculation	
Stack Read	Initialization Byte	0xA0	Always 0xA0	A peripheral transmits data (provided by host controller or itself) to another next peripheral in daisy chain
	Device ID Address	--	No address byte is sent in stack read	
	Register Address	0x0215	Start with address 0x215	
	Data	0x0B	Send 12 bytes worth of data back (register contents from 0x215 to 0x220) from each device in the stack	
	CRC	0xCCB3	16-bit CRC calculation	
Stack Write	Initialization Byte	0xB3	Writing 4 bytes to the stack devices	A peripheral transmits data (provided by host controller or itself) to another next peripheral in daisy chain
	Device ID Address	--	No address byte is sent in stack write	
	Register Address	0x0100	Start with address 0x100	
	Data	0x02B778BC	Write 4 bytes to registers 0x100-0x103 to all devices in stack	
	CRC	0x0A35	16-bit CRC calculation	
Single Device Read	Initialization Byte	0x80	Always 0x80	The host controller communicates with its desire peripheral via its unique device address (for example 0x01) and other peripherals between these two are only bypassing the data
	Device ID Address	0x01	Device address 0x01 is addressed in this	
	Register Address	0x0215	Start with address 0x215	
	Data	0x0B	Send 12 bytes worth of data back (register contents from 0x215 to 0x220)	
	CRC	0xCAB5	16-bit CRC calculation	
Single Device Write	Initialization Byte	0x93	Writing four data bytes to a single device (0x90 for 1 byte of data)	The host controller communicates with its desire peripheral via its unique device address (for example 0x01) and other peripherals between these two are only bypassing the data
	Device ID Address	0x01	Device address 0x01 is addressed in this case	
	Register Address	0x0100	Start with address 0x100	
	Data	0x02B778BC	Write 4 bytes to registers 0x100-0x103	
	CRC	0x8A4C	16-bit CRC calculation	

Subsection 3.6.2 will detail the Device ID addressing bytes. Device ID address bytes, initialization bytes, and CRC bytes are associated with the Modbus Data Link Layer (OSI Layer 2), while Register address bytes and Data bytes pertain directly to the Application Layer (OSI Layer 7).

3.6.1 Error detection methods

Error detection methods are crucial when noise and disturbances are likely, such as in long communication lines or the presence of electro-magnetic fields (EMF). While there are many different error detection methods, this thesis focuses on those that are frequently mentioned [37].

CRC calculation:

In this thesis, CRC (Cyclic Redundancy Check) bytes mentioned multiple times. Since Table 6 referred with some samples, it would be informative to explain how CRC byte is calculated [37]. This thesis is not supposed to provide mathematical operation how CRC calculation is done. Reference [38] can easily measure the CRC values in many famous formats. This subsection seeks the format CRC used in Table 6 within reference [38] with few information.

Here's a step-by-step explanation:
1) Data Preparation: The respective hexadecimal (or binary) needs to be prepared before CRC calculation. For example, for Single Device Write in Table 6, the whole message along with CRC result were given.

- Initialization Byte 0x93
- Device ID Address 0x01
- Register Address 0x0100
- Data 0x02B778BC
- CRC 0x8A4C

So, the original data is: 93 01 01 00 02 B7 78 BC

And final expected CRC is: 0x8A4C.

2) Polynomial: a predefined "divisor" which is a binary (or Hexadecimal) number represented by a polynomial. This is agreed upon by both the sender and receiver. Data in Modbus RTU framing, it is needed to follow the CRC-16 Modbus standard, which uses the polynomial 0x8005.

3) Append Zeros: Adding a series of zeros to the end of data. The number of zeros should be one less than the number of bits in the divisor.

4) Division: Perform binary division of your augmented data (data + zeros) by the divisor using XOR (exclusive OR) operations. Keep performing the division until you've processed all the bits in the data.

5) Remainder: The remainder left after the division is the CRC value. This remainder is attached to the end of the original data and sent to the receiver. In the recent example, the 16-bit-CRC value is 0x8A4C.

6) Verification: The receiver also performs the same division on the received data (which now includes the CRC). If the resulting remainder is zero, it means the data was received correctly. If it is not zero, it means there was an error in transmission.

Figure 29 shows that CRC-16/MODBS is 0x4C8A, which corresponds to 0x8A4C. The only slight variation is the placement of the CRC bytes, with the Most Significant Byte (MSB) and the Least Significant Byte (LSB) being reversed.

CRC-16	Result	Check	Poly	Init	RefIn	RefOut	XorOut
CRC-16/ARC	0x47CA	0xBB3D	0x8005	0x0000	true	true	0x0000
CRC-16/CDMA2000	0xDF56	0x4C06	0xC867	0xFFFF	false	false	0x0000
CRC-16/CMS	0xFABD	0xAEE7	0x8005	0xFFFF	false	false	0x0000
CRC-16/DDS-110	0xF68D	0x9ECF	0x8005	0x800D	false	false	0x0000
CRC-16/DECT-R	0x9515	0x007E	0x0589	0x0000	false	false	0x0001
CRC-16/DECT-X	0x9514	0x007F	0x0589	0x0000	false	false	0x0000
CRC-16/DNP	0x38C7	0xEA82	0x3D65	0x0000	true	true	0xFFFF
CRC-16/EN-13757	0x940D	0xC2B7	0x3D65	0x0000	false	false	0xFFFF
CRC-16/GENIBUS	0x4AE0	0xD64E	0x1021	0xFFFF	false	false	0xFFFF
CRC-16/GSM	0x7BDE	0xCE3C	0x1021	0x0000	false	false	0xFFFF
CRC-16/IBM-3740	0xB51F	0x29B1	0x1021	0xFFFF	false	false	0x0000
CRC-16/IBM-SDLC	0x7C15	0x906E	0x1021	0xFFFF	true	true	0xFFFF
CRC-16/ISO-IEC-14443-3-A	0xAA5C	0xBF05	0x1021	0xC6C6	true	true	0x0000
CRC-16/KERMIT	0xFF66	0x2189	0x1021	0x0000	true	true	0x0000
CRC-16/LJ1200	0x1B00	0xBDF4	0x6F63	0x0000	false	false	0x0000
CRC-16/M17	0xDA32	0x772B	0x5935	0xFFFF	false	false	0x0000
CRC-16/MAXIM-DOW	0xB835	0x44C2	0x8005	0x0000	true	true	0xFFFF
CRC-16/MCRF4XX	0x83EA	0x6F91	0x1021	0xFFFF	true	true	0x0000
CRC-16/MODBUS	0x4C8A	0x4B37	0x8005	0xFFFF	true	true	0x0000
CRC-16/NRSC-5	0x4E84	0xA066	0x080B	0xFFFF	true	true	0x0000

Figure 29. CRC calculation with various CRC format outputs for the same hexadecimal input

Parity calculation:

Parity check calculation is a straightforward error detection technique that involves appending an additional bit to a data transmission. There are two types of parity checks:

1. Even Parity: The total number of 1s in the data (including the parity bit) is even. Example:
    - Data: 1010110 (4 ones)
    - Add parity bit: 0 (to keep it even)
    - Transmitted data: 10101100
2. Odd Parity: The total number of 1s in the data (including the parity bit) is odd. Example:
    - Data: 1010110 (4 ones)
    - Add parity bit: 1 (to make it odd)
    - Transmitted data: 10101101

3.6.2 Auto addressing on battery module

In Figure 28, the Auto Addressing mechanism eliminates the need for manual addressing of each peripheral when the MCU controller polls data from available peripherals. This mechanism ensures that data is polled in the correct order. With each peripheral having its own address, voltage cell data and temperature sensors can be accurately located on the hardware, ensuring data integrity and making debugging and troubleshooting easier on battery hardware and software.

In Figure 29, for in the first action for auto addressing, the MCU broadcasts a **WAKE ping** via UART protocol connected to “B0” base peripheral and “B0” was programed to know this WAKE ping and upon this WAKE ping message, the initial peripheral will identify itself as “base” peripheral as expected. “B0” base peripheral sends **WAKE tone** via RS485 differential signaling for all remaining peripherals in the linear daisy chain. These remaining peripherals are familiar with WAKE tone message, and they will identify themselves as “cell stacked” or “stack” peripherals. This identification regarding being “base” or “stack” will be stored in memory of peripherals. In their memories, in “COMM_CTRL” registery shown in Appendix A, [STACK_DEV] bit should be “0” for “base” and should be “1” for “stack”. Better visual desicritpion can be referred to Appendix A, part 1.

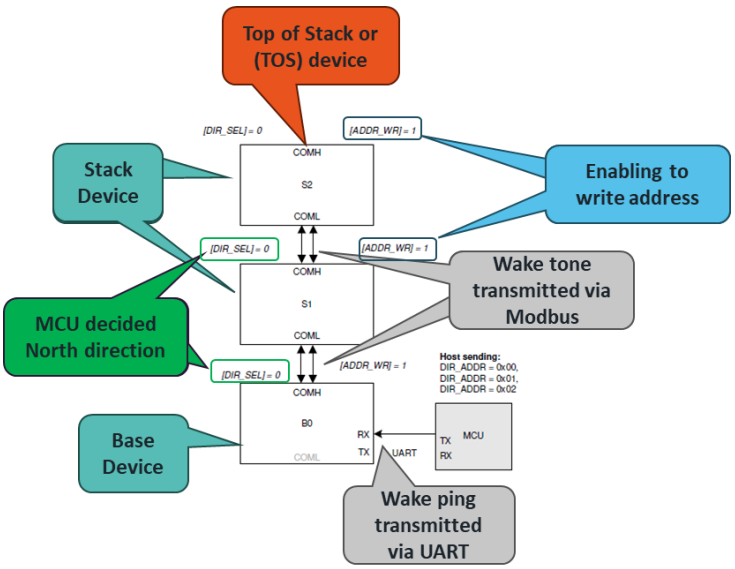


Figure 29. Auto Address process controlled by MCU

Once all peripherals are awake, thanks to Wake Ping and Wake Tone, and have identified themselves, they are ready for communication. The MCU must decide the direction of the daisy chain communication. It needs to send a new message with a sufficient delay to ensure the Wake toning from the previous step is complete. This new message includes a "Direction Selection" bit for all peripherals. If the bit [DIR_SEL] = 0 in the "CONTROL1" Modbus registry (as shown in Appendix A, part 1), the controller directs the communication flow with all available "stack" peripherals through the High side (COMH or North) of B0. "DIR_SEL" stands for "Direction Selection". If [DIR_SEL] = 1, the MCU directs the communication flow through the Low side (COML or South) of B0 [or Reverse Broadcast Write command in Appendix A, part 2].

To proceed auto addressing, the MCU must send another message with the "address writing" activation bit. By default, this bit is set to [ADDR_WR]=0. When the MCU sets [ADDR_WR]=1 in the "CONTROL1" registry (see Appendix A), all peripherals become active to receive address values. The MCU then broadcasts writing to assign consecutive addresses to each DIR0_ADDR[ADDRESS5:0] in the "DIR0_ADDR_OPT" registry. To insert the given address, the data from the "DIR0_ADDR_OPT" registry is transferred from the first stack to the nearest stack into the "DIR0_ADDR" registry. Essentially, the "DIR0_ADDR_OPT" communication registry is used for MCU to stack communication,

while the "DIR0_ADDR" communication registry is used for communication from one stack to the nearest stack (as detailed in Table 6, referred to as Stack Writing).

When the initial default peripheral address begins with 0x00, the B0 peripheral is assigned 0x00, and the other peripherals are assigned consecutive hexadecimal values like 0x01, 0x02, and so on. As Stack Writing progresses sequentially from the first to the last stack in the linear daisy chain, the last stack eventually receives no response from the next stack after waiting for a while. In this scenario, the last stack updates its "COMM_CTRL" registry to TOP_Stack=1, indicating that it recognizes itself as the top and final peripheral in the linear daisy chain. Subsequently, the last stack will notify the MCU using the Stack Reading format communication shown in Table 6.

If the MCU decides to change the direction in the Daisy Chain, [ADDR_WR] should be set to "0" again to reset the addresses for the peripherals. Then, in the "CONTROL1" registry, instead of DIR_SEL=[0], set DIR_SEL=[1] for the opposite direction. Appendix A, part 4 provides a visual representation of how these steps are performed.

3.6.3 Different Write and Read functions on battery module

In the previous subsection, Table 6 illustrates how communication between the MCU and stacks through broadcasting messages, as well as communication between stacks without requiring the peripheral's address, is possible. This type of communication is necessary to assign each peripheral a unique address for further essential communication. This further essential communication refers to interactions between the MCU and individual stacks. These communications are also known as "Single Read Command" and "Single Write Command". In the "Single Command" communication between the MCU and **individual** stacks, data such as voltage cells, temperature sensors, and EEPROM writing and reading are performed. The next subsection delves into the communication between the controller and individual stacks, as well as the communication between stacks, explaining how the proposed test bench can communicate with the battery pack. Appendix A, part 6 visuallly explains the ”Single Read” and ”Single Write” commands.

3.7 Battery pack's simulation

The author of this thesis decided to set up an automated test bench to conduct quality tests (isolation, dielectric, and equipotential tests) using a real battery pack. However, for battery pack communication, instead of using the CMC boards inside the battery pack, the automated test bench employed Arduino boards to simulate the CMC hardware.

There are several reasons for this choice:

- A simulated battery is always available for study, even when working remotely.
- The original cell stack battery boards and the bridge controller were manufactured by Texas Instruments. Due to Texas Instruments's monopoly, the messages exchanged between the battery peripherals and the controller are not certainly determined.
- In the actual testing setup, a Texas Instruments bridge device was used, and there were no spare parts available for this proposing test bench.
- This battery is from a third party (not IONCOR's brand), and limited information about its communication protocols.
- Since this battery pack is a third-party product, a permission was required from the third party for sharing communication data.

Figure 30 illustrates the use of alternative Arduino peripherals in place of the actual battery module peripherals. The Arduino Mega replaces the MCU unit and the B0 stack described in Subsection 3.6.2. Arduino Micros serve as substitutes for the battery module peripheral microprocessors, while Arduino Nanos are used instead of the EEPROM memories found in the external EEPROM memories within the battery modules. Given that the battery packs contain only two battery modules (or cell stacks), there are two simulated microprocessors and EEPROM memories per pair. This simulation does not incorporate the Modbus protocol. Due to budget constraints, Modbus hardware tools are not available, and the UART protocol was used as a substitute. However, the SPI protocol remains in use for communication between EEPROMs and battery microprocessors for reading and writing EEPROM memories.

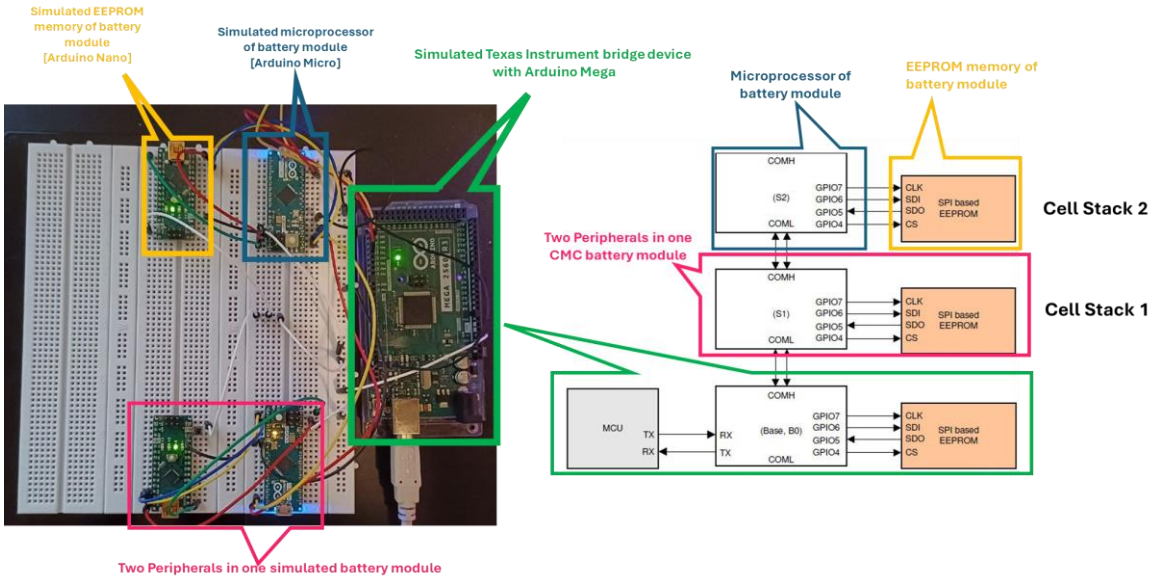


Figure 30. Simulated battery pack's peripherals and its controller with Arduino boards

Table 7 presents the proposed register addresses for communication between the Arduino Mega (acting as the Test Bench Controller) and other peripherals such as the Arduino Micro (serving as the battery module microprocessor) and the Arduino Nano (acting as the battery module EEPROM). Registry addresses "A1," "A2," "B1," and "B2" refer to voltage cell values, with "A" corresponding to the first module connected to the controller and "B" referring to the second module in the linear daisy chain via UART. The numbers "1" (in "A1" and "B1") encompass data from cells 1 to 10, while the numbers "2" (in "A2" and "B2") cover data from cells 11 to 19.

The reason for voltage cells, two registry such "1" (for cells from 1 to 10) and "2" (for cells from 11 to 19) were considered, is due to fact that UART message on Arduino Micro cannot be very long. 64 bytes of data at a time is the hardware restriction on UART for Arduino Micro. For example, for the message below for all 19 cells in single UART message in ASCII, the total byte needed can be calculated such:

"3.556,3.553,3.554,3.552,3.551,3.550,3.554,3.552,3.556,3.553,3.554,3.552,3.551,3.550,3.554,3.552,3.552,3.551,3.550"

Each voltage cell data, such as "3.522," consists of 5 characters (4 number character and one "." floating sign character). With 19 cells, this totals 95 characters (5 characters * 19 cells). Additionally, there are 18 commas between the voltage cell data in the message.

Therefore, in ASCII, the total message length is 113 characters or bytes (95 characters + 18 commas), which exceeds the 80-byte limitation.

Table 7. Simulated battery pack and simulated controller registry addresses

Module	Controlling message	Communication type	Description
"A" or stack 1	A1	Controller-peripheral	Reading voltage cells [from cell 1 to cell 10] in module "A" via UART protocol. Character "A" addresses to Stack 1 and the number "1" was registered to cell voltage from cell 1 to cell 10.
"A" or stack 1	A2	Controller-peripheral	Reading voltage cells [from cell 11 to cell 19] in module "A" via UART protocol. "A" address to Stack 1 and the number "2" was registered to cell voltage from cell 11 to cell 19.
"A" or stack 1	A3	Controller-peripheral	Reading six temperature sensors embedded across module "A" via UART protocol. "A" address to Stack 1 and the number "3" was registered to temperature sensors.
"A" or stack 1	A4*	Peripheral-peripheral	Writing battery barcode to external EEPROM memory of module "A" via SPI protocol. "A" address to Stack 1 and the number "4" was registered to **activating SPI** for **sending data**. The sign "*" received by the external EEPROM, keeps the EEPROM memory on **writing mode**.
"A" or stack 1	A5+	Peripheral-peripheral	Reading battery barcode from the external EEPROM memory of module "A" via SPI protocol. "A" address to Stack 1 and the number "5" was registered to **activating SPI** for **receiving data**. The sign "+" received by the external EEPROM, keeps the EEPROM memory on **reading mode**.
"B" or Stack 2	B1	Controller-peripheral	Reading voltage cells [from cell 1 to cell 10] in module "B" via UART protocol. Character "B" addresses to Stack 2 and the number "1" was registered to cell voltage from cell 1 to cell 10.
"B" or Stack 2	B2	Controller-peripheral	Reading voltage cells [from cell 11 to cell 19] in module "B" via UART protocol. "B" address to Stack 2 and the number "2" was registered to cell voltage from cell 11 to cell 19.
"B" or Stack 2	B3	Controller-peripheral	Reading six temperature sensors embedded across module "B" via UART protocol. "B" address to Stack 2 and the number "3" was registered to temperature sensors.
"B" or Stack 2	B4*	Peripheral-peripheral	Writing battery barcode to external EEPROM memory of module "B" via SPI protocol. "B" address to Stack 2 and the number "4" was registered to **activating SPI** for **sending data**. The sign "*" received by the external EEPROM, keeps the EEPROM memory on **writing mode**.
"B" or Stack 2	B5+	Peripheral-peripheral	Reading battery barcode from the external EEPROM memory of module "B" via SPI protocol. "B" address to Stack 2 and the number "5" was registered to **activating SPI** for **receiving data**. The sign "+" received by the external EEPROM, keeps the EEPROM memory on **reading mode**.

The numbers "3" following "A" and "B" indicate temperature sensors using the Negative Temperature Coefficient (NTC) format. The resistance of NTC thermistors is usually measured in ohms (Ω) and changes with temperature; measurements can also be shown in degrees Celsius (°C) or Fahrenheit (°F) within the thermistor's operational range. Reference [39] explains the NTC calculation method. In Figure 31, the response for number "3" is:

"1.557,1.852,1.555,1.848,1.555,1.852"

In the message above, two distinct numerical values are observed. This indicates that there are two different reference NTC resistors used for each CMC board in this module. Consequently, the values approximately equal to ≈1.5 have different resistance references compared to the values approximately equal to ≈1.8 mentioned in the message.

The numbers "4" (in "A4*" and "B4*") pertain to writing EEPROM data, accompanied by a character like "*" which prompts the EEPROM (Arduino Nano) to switch to writing mode. The Arduino Micro sends the "*" character via the SPI protocol to the Arduino Nano. This "*" character acts as a trigger, indicating to the Arduino Nano to store the subsequent characters into its EEPROM until the SPI message concludes. In Figure 31, the Arduino Nano IDE window's serial monitor displays the data stored byte by byte in the EEPROM as "PROTO8695," which is the battery pack's serial number. Subsection 3.5.1 mentioned that it is recommended to digitally store the battery serial number in the EEPROM. The numbers "5" (in "A5+" and "B5+") indicate reading EEPROM data and sending the stored data back to Arduino Micro. This process is activated by a character like "+" which puts the EEPROM (Arduino Nano) into reading mode when the specified EEPROM addresses are encountered in SPI communication.

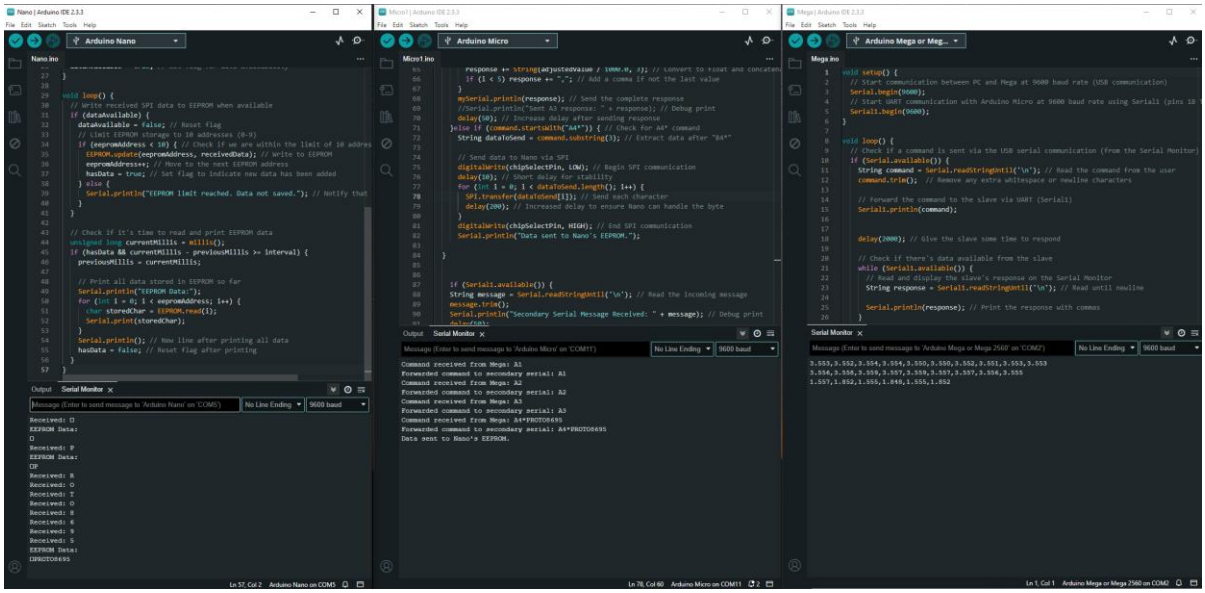


Figure 31. Simulation battery pack's peripherals and the controller in Arduino IDE environment. The right-side IDE window refers to Arduino Mega, the middle IDE window refers to Arduino Micro, and the left-side IDE window refers to Arduino Nano.

This chapter has offered thorough information about the battery pack being tested in comparison to the proposed test bench. Now, it is time to focus on the specifics of the proposed test bench. In the upcoming chapter, there will be an in-depth examination of the software and hardware elements that make up the proposed automation test setup system.

4 Automation System Setup Structure

Up to this point, the thesis has covered details about automotive batteries and the battery pack under test, laying the groundwork for developing an automated test setup. This chapter will introduce the potential structure of the automation testing system. Figure 32 shows the electrical and digital (software & hardware) components used in the automation testing system unit. The following chapters will provide a step-by-step breakdown of Figure 32. Three main software programs—Arduino IDE, NI LabVIEW, and NI TestStand—are used for integrating the hardware components. The components include Normally Open Relays, a temperature/humidity sensor for measuring environmental conditions, the testing device, one Arduino shield along with two Arduino controllers, an OLED display, a buzzer and LED alarm, and the battery pack. Also, digital communication protocols such as SMB, RDP, and FTP were employed for Azure cloud storage, remote monitoring, and data transferring, respectively. The following subsection explains the reasons for selecting each component and introduces the key components.

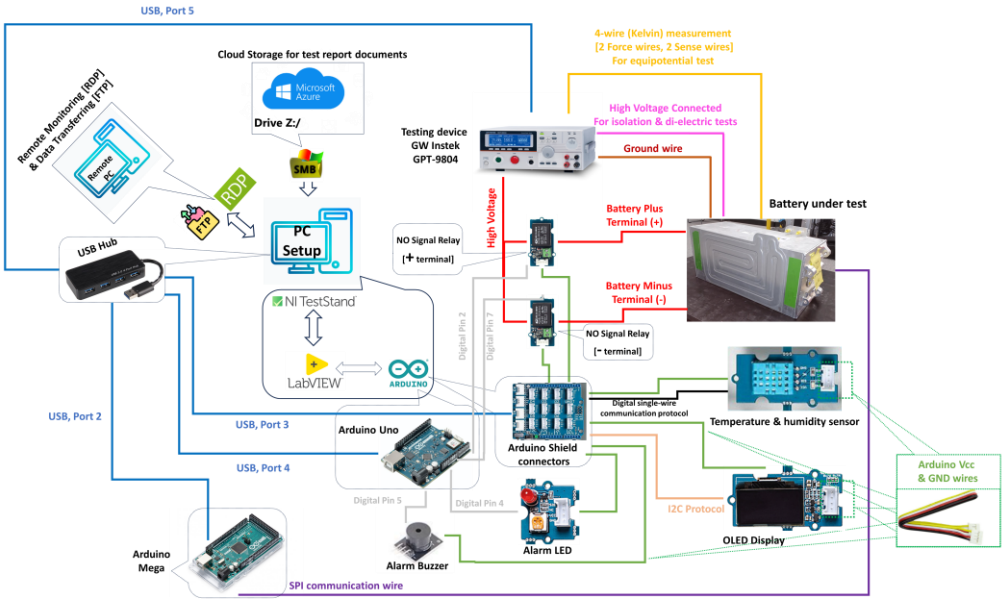


Figure 32. Semi-automation Test Setup Components

4.1 Analyzing testing setup components selection

Figure 32 shows the test setup elements, which were chosen for specific reasons. IONCOR already uses LabVIEW and TestStand software in production lines, so these, along with open-source software like Arduino IDE, are included in the setup. The company also uses Microsoft Azure for cloud storage. The remote connection and data transfer features mentioned in this thesis align somewhat with the company's policies but need authorization from HelpDesk personnel. In this thesis, software practices such as cloud programming, remote controlling, and remote data transferring were never used within the company's environment or facilities; they were only applied outside the company.

At IONCOR, more expensive signal controllers like those from National Instruments or Texas Instruments are used. This thesis proposes using much cheaper Arduino boards for signal control. The safety testing device used in this thesis is available in multiple units. The NO relays shown in Figure 32 are for practice; more powerful relays are needed for the actual setup. Arduino temperature and humidity sensors, used in this thesis, are reliable for this setup due to their low cost, but they can be calibrated with other sensors available in the proto-factory over time.

4.2 Introducing NI LabVIEW

NI LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical programming tool popularly used within the engineering and scientific domains for the purpose of data acquisition, control of instruments, and automation. It allows for system designing and creating test scenarios in a visual block diagram approach that makes such complex programming quite intuitive. The flexibility of LabVIEW, when combined with an enormous library of prebuilt functions, contributes to its wide popularity in fields such as automotive, aerospace, and biomedical research. LabVIEW can interface directly with popular languages such as C++, C, Python, and .NET [40].

4.2.1 NI LabVIEW application in electrical testing

Electrical tests are possible with powerful tools such as NI LabVIEW, which allows the design of automated test set-ups. NI LabVIEW makes the development of a custom test procedure much easier because of its graphical programming interface, which integrates

with a wide variety of instruments and hardware. LabVIEW is best suited for applications such as circuit validation, power analysis, and fault detection of electrical systems, all because of real-time data acquisition and its capability for analysis. Most of the famous testing devices in the market are programmed with LabVIEW. All the tests of the introduced safety tester device in Figure 17 are programmed with LabVIEW VIs: isolation, dielectric and equipotential. These open-source LabVIEW VIs can be modified based on the application and can be used for further development and automation. In this thesis, GWInstek VIs were modified in order to be used in the environment of NI TestStand.

4.2.2 NI LabVIEW configuration as a function

NI LabVIEW should be set up like a function to be used with other applications. Similar to a function, which includes inputs and outputs, an NI LabVIEW VI can be configured as shown in Figure 33. In this figure, the selected element, such as “VISA resource name” (labelled as number 1), acts as a controller block in the VI’s Front Panel. Because the Front Panel controller requires an input value (in this case, the COM USB number), LabVIEW automatically treats it as an input when assigned in the Front Panel terminal. Conversely, “VISA resource name out” (labelled as number 4) is a printing block and is considered an output. As depicted in Figure 33, different variable types in the LabVIEW environment are color-coded: strings in purple, floating numbers in orange, enumerations in blue, errors in light green, and binary variables in green.

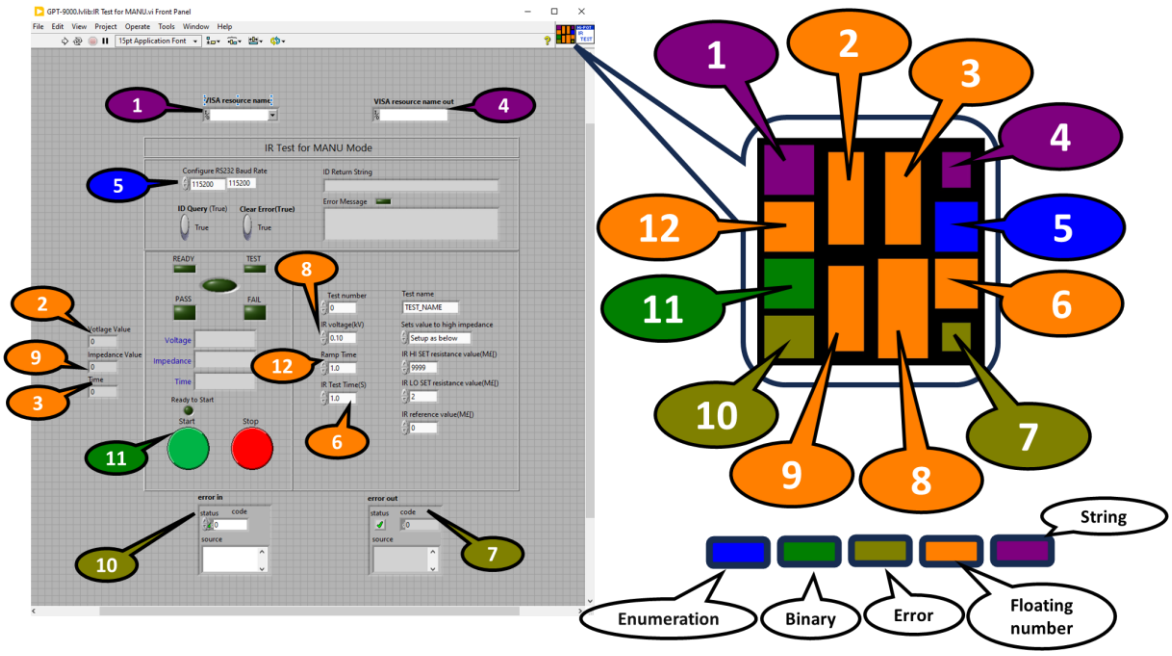


Figure 33. Configuring input and output terminals in LabVIEW for reuse in other applications as a function

4.3 Introducing Arduino

Arduino is an open-source electronic platform with an easy-to-use hardware-software interface, intended for interactive projects. It consists of a microcontroller board and a development environment for writing and uploading the code. Thanks to ease of use, versatility, and great community support, Arduino has become very popular not only among hobbyists but also in the educator and professional worlds for prototyping, robotics, and IoT applications. [41].

4.3.1 Arduino Controllers

in Subsection 3.7, Arduino “Micro” and “Nano” were used for battery pack simulation as peripherals. In this subchapter, the concentration is on Arduino boards used in proposing test bench as controllers, not peripherals. In Figure 32, two Arduino controllers, "Mega" and "Uno", have been applied. Arduino Uno controls NO relays via digital pins with simple serial communication and read ambient temperature and humidity. Arduino Mega handles some tasks like reading SPI messages from a battery and EEPROM writing. In this thesis, both controllers of Uno and Mega communicate with LabView through USB COM serial communication. Using two separate controllers speeds up the troubleshooting process.

4.3.2 Arduino Programming

The Arduino IDE (Integrated Development Environment) is an open-source platform designed for programming user-friendly hardware like Arduino controllers (e.g., Uno, Mega, Nano, Micro) and sensors (e.g., temperature, noise, smoke detectors). It can also be used with non-Arduino electrical products. Scripts in the Arduino IDE are written in C or C++. The IDE first preprocesses the script to identify potential errors, then compiles it into a Hex file that the microcontroller can understand. Finally, the Hex file is uploaded to the microcontroller's memory.

4.4 Introducing NI TestStand

NI TestStand is a software solution for managing and automating test sequences in manufacturing and validation settings. It enables users to create, manage, and execute automated tests for various products, integrating different test systems and tools. Typically, the developer user, identified as the administrator, has a unique password to prevent regular users from editing test sequences. Regular users can have their own usernames and passwords, and their names appear in the generated test reports. With features like sequence control, report generation, and database logging, TestStand enhances the efficiency of testing processes in industries such as electronics, aerospace, and automotive [42].

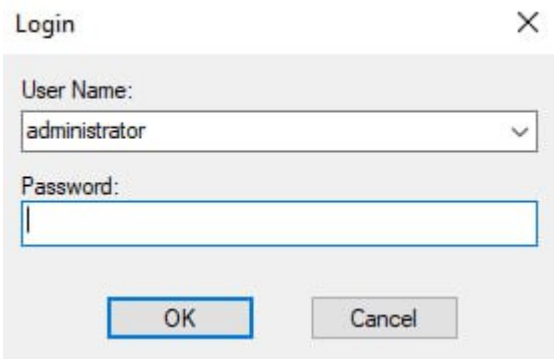


Figure 34. Username & Password for each user can be defined

In the automation testing setup within TestStand, most of the steps in the testing sequence are illustrated in Figure 35. Similar to many text-based or graphical

programming languages, TestStand supports action steps such as if conditions, for and while loops. Additionally, TestStand is compatible with LabView, C/C++, .Net, and Python.

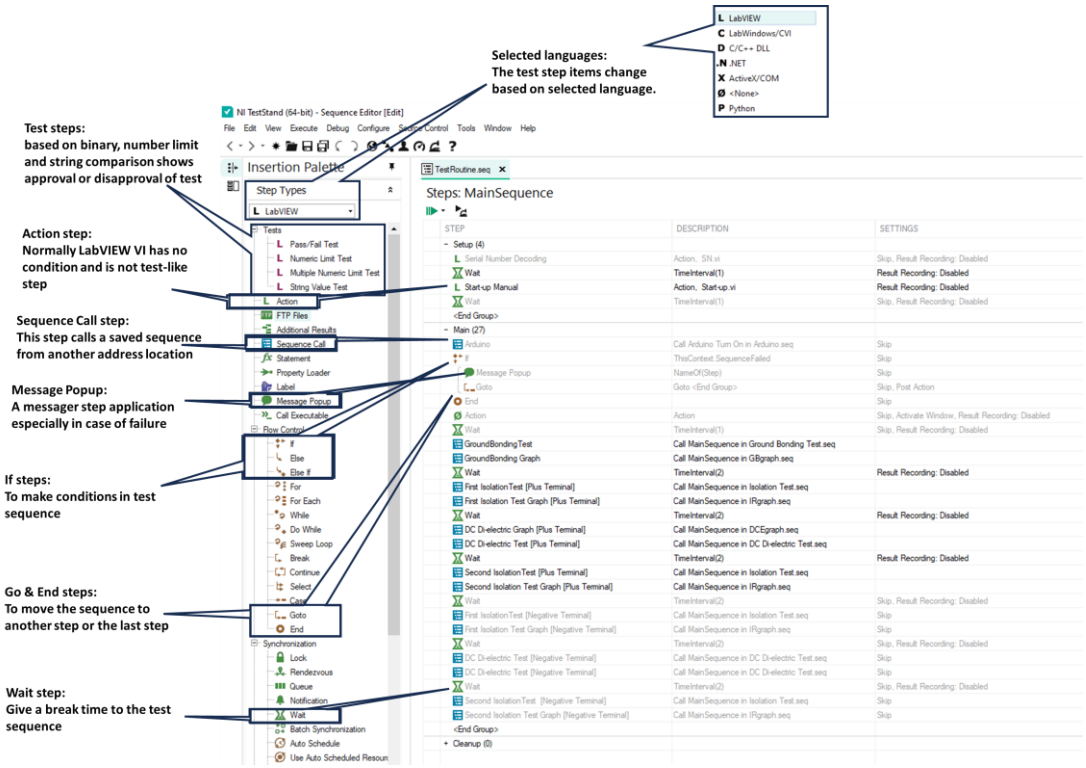


Figure 35. Introducing famous steps used in TestStand environment

4.4.1 NI TestStand integration with other applications

Figure 36 displays the VI loaded in the TestStand environment, as previously shown in Figure 33. This VI, named IR Test.vi, is used for isolation resistance testing and indicates which variables are input and output from the LabVIEW VI. The developer can choose whether to use default inputs, such as setting the Baud Rate to 115200, or modify them within the TestStand environment. TestStand adheres to LabVIEW's conventions for handling input and output variables and supports various parameter types as detailed in Reference [43].

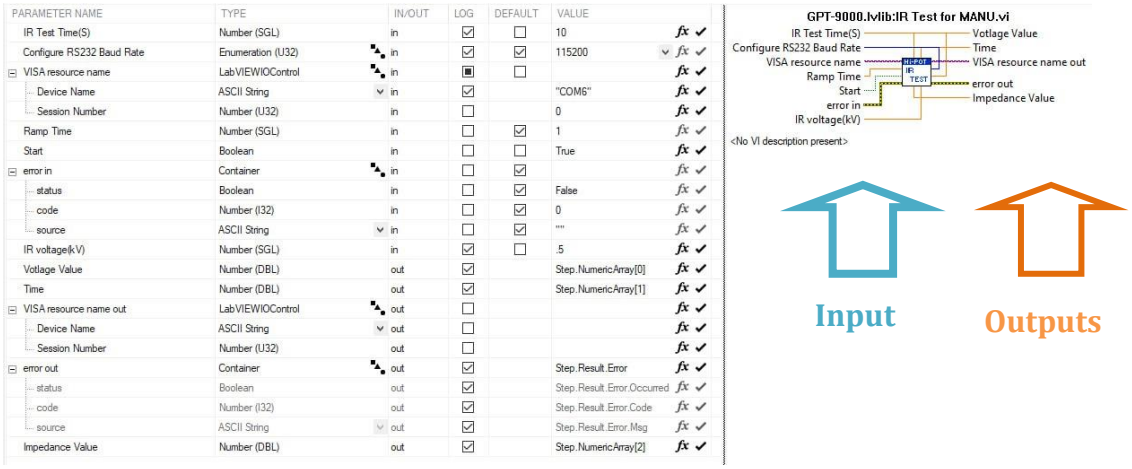


Figure 36. LabVIEW VI configured into TestStand environment

4.4.2 Proposed NI TestStand test sequence strategy

Figure 32 had displayed the components used in the automation test setup. To clarify how these components in Figure 32 cooperate with each one, a testing strategy for these components needs to be defined. This strategy evolves throughout development, influencing the selection of the optimal scenario. Given that a TestStand sequence plays a crucial role in integrating all setup components, the strategy should be tailored specifically for TestStand. The final strategy, suitable for TestStand, is shown in Figure 37. In Section 5, the automation testing sequence follows the steps outlined in Figure 37 sequentially.

Figure 37. Semi-automation test sequence strategy

5 Automation Testing Sequence Strategy

This section outlines the step-by-step automation test sequence programmed using NI LabVIEW and Arduino. It describes the programming of each NI LabVIEW VI and Arduino script, demonstrating how these scripts conduct tests. To automate the execution of these LabVIEW and Arduino files, they are integrated into the TestStand environment as function modules with specified inputs and outputs. Section 5 is mapping the final strategy steps as depicted in Figure 37.

Before starting with the automation test setup, it is crucial to note (as highlighted in the introduction) that the setup is not fully automated. For example, testing operators must manually attach and later detach the cables for testing. Additionally, operators are responsible for battery pack's QR code scanning and conducting safety checks. This semi-automated setup is intended for a prototyping factory, rather than for high-speed serial production. Figure 38 illustrates the factors influencing the strategy for the semi-automation testing setup.

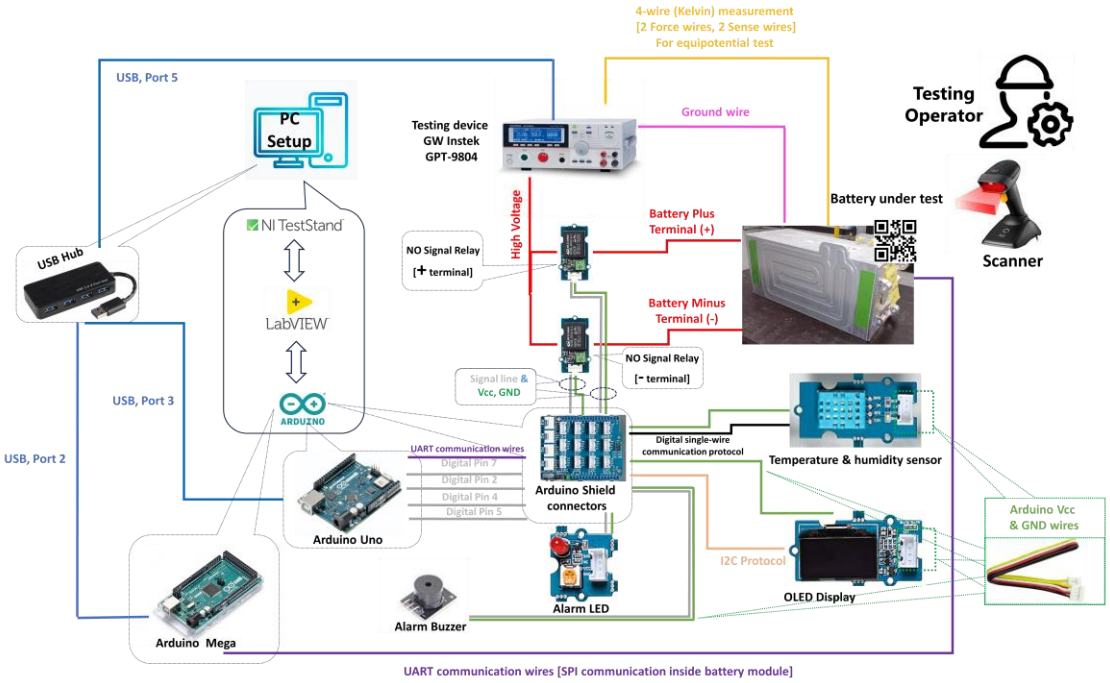


Figure 12. Semi-automation Test Setup Components (Scoping only testing strategy)

5.1 Battery serial number verification

The automation testing system must ensure that the battery serial number is valid and matches the test sequence. Occasionally, the testing operator might scan the QR code incorrectly, or the QR code on the battery or document might be faded or damaged. Even a minor mistake, such as a single digit error, can occur. If the QR code is invalid when the automation sequence starts, the system should alert the operator of the failure.

The first step in the semi-automated testing process is to read the serial number of the Unit Under Test (UUT), which refers to the serial number of the battery pack. The expected serial number of the battery pack begins with the capital letters "PROTO," indicating Prototyping factory (the assembly location), followed by a four-digit number. In Figure 14, the example provided is "PROTO1122".

A designated LabVIEW VI file is programmed to verify the serial number, and the resulting VI file is subsequently loaded into the TestStand environment. To execute this step, the procedure outlined in Figure 39 is followed:

1. A testing operator scans the QR code of the battery pack under test
2. The scanned QR barcode appears in the NI TestStand UUT serial number window before the test sequence begins. The operator must press “OK” in Figure 39 to proceed.
3. As shown in Figure 39, NI TestStand then loads a pre-programmed VI created with NI LabVIEW and inputs the scanned digits into the loaded VI.
4. The pre-programmed VI validates the scanned code and returns a result of True or False. If the code is valid, it returns True; otherwise, it returns False.
5. NI TestStand displays True or False during the test sequence. If False, the TestStand test sequence is halted; if True, it continues to the next sequence.
6. The testing operator can see on the PC monitor whether the scanned code has failed or not.

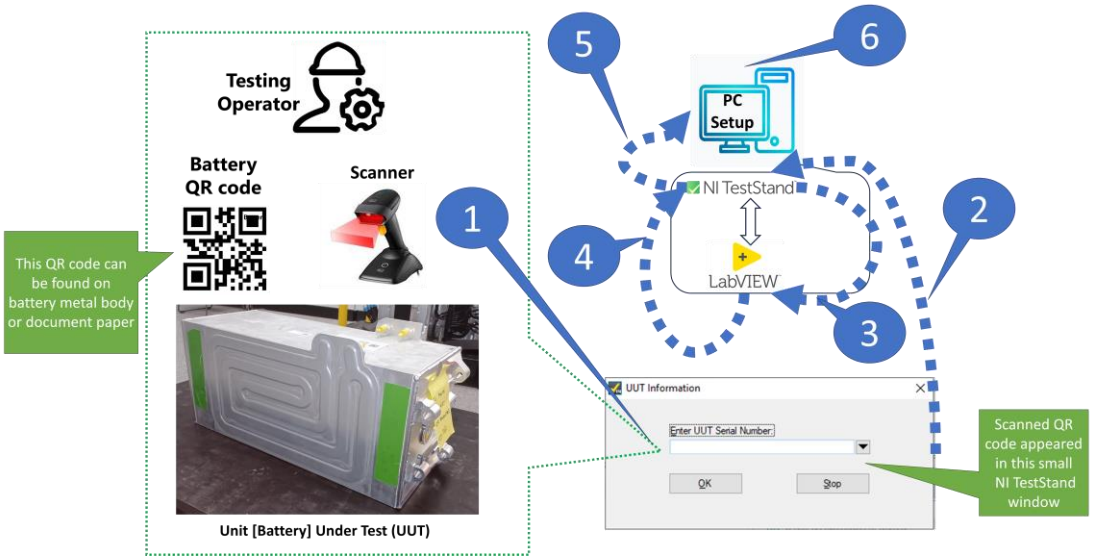


Figure 39. QR code verification via test setup

The programmed VI created using NI LabVIEW (Front Panel & Block Diagram) is shown in Appendix C. This VI is the initial step in the TestStand test sequence. To verify the battery barcode, the VI checks five conditions.

1. Is the barcode length correct?
2. Is the string match correct?
3. Is the number length correct?
4. Is the number greater than 0?
5. Is there no offset in number detection?

The test step in TestStand for verifying the serial number employs a binary test output. If any preceding condition fails, the test sequence stops, and TestStand notifies the testing operator to scan the correct barcode, indicating a false error.

5.2 Safety manual for testing operator

This is another manual step for testing operators. Ensuring the safety of both the operators and the product is crucial. A start-up manual is required before beginning the test sequence. Appendix D displays the VI created for this step. Figure 40 illustrates the wiring components that the operator needs to verify for this safety manual. For instance, the operator should ensure that the equipotential cables, both battery terminals cables, ground wire, and SPI communication wire are securely fastened to the battery pack under test.

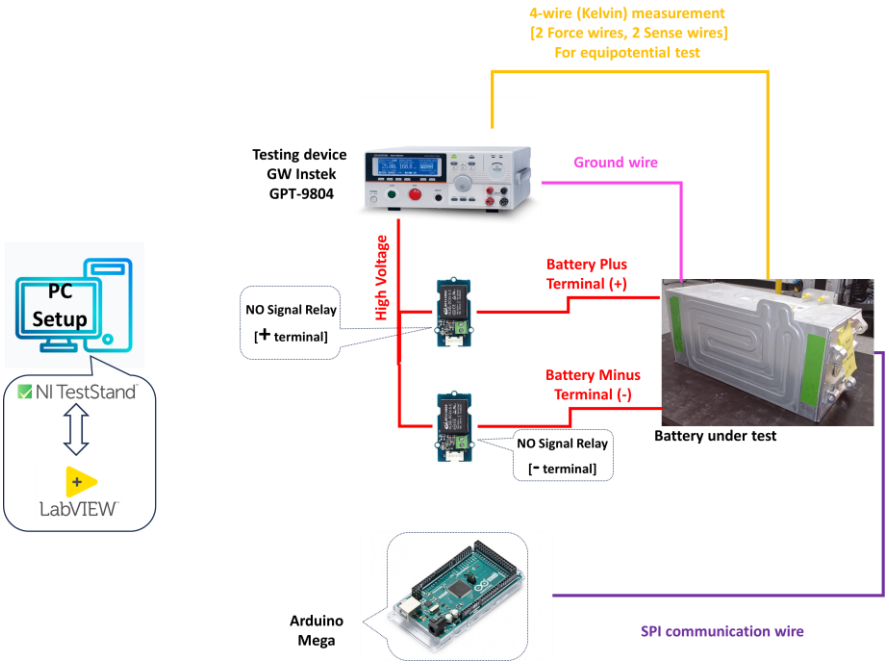


Figure 40. Operator examining wiring connections

5.3 Environment test condition

The optimal environmental conditions for testing batteries typically involve maintaining a controlled temperature and humidity range to ensure accurate and reliable results. The ideal temperature range should be maintained between 22°C and 27°C, with humidity levels kept between 20% and 70%. Figure 41 illustrates the components used for measuring the ambient temperature and humidity. Appendix E includes Arduino programs for Arduino temperature and humidity sensors, as well as the OLED display. Also, Appendix E details the VI developed to receive temperature and humidity values from port 3 into LabVIEW and subsequently transfer them from LabVIEW to the TestStand

environment. Finally, TestStand report regarding this step generated at the end of Appendix E.

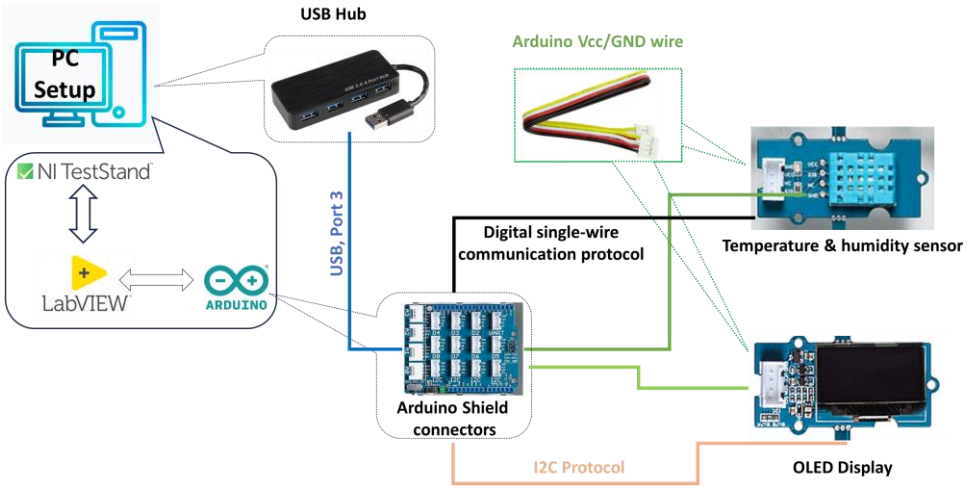


Figure 41. Components involved in environment test conditions

5.4 Battey communication conditions

Section 3 thoroughly detailed the information about the battery pack under test. As depicted in Figure 42, the Arduino Mega serves as the battery controller, responsible for reading voltage cell and temperature sensor data. Table 7 demonstrated the process by which the Arduino Mega reads this data from each battery module.

The battery pack consists of two battery modules, each module is containing 19 prismatic cells, making the battery pack totally 38 cells. It is crucial that all cells maintain a proper voltage tolerance, with the low difference between the highest and lowest voltage cells not exceeding more than 30 millivolts. Voltage imbalances can lead to reduced capacity by causing some cells to overcharge or undercharge, ultimately shortening the battery pack's lifespan over numerous charge and discharge cycles.

In Subsection 1.3, the battery pack under test was noted to use NMC chemistry. This subsection also specifies that the typical operating voltage range for NMC chemistry is between 3000 millivolts (at 0% SOC) and 4200 millivolts (at 100% SOC). The Arduino Mega, as discussed in Subsection 3.7, is the controller responsible for interpreting the communication signals from the battery pack, which can include voltage cell signals, temperature sensor signals, and EEPROM data. For a battery pack with an SOC of 60%, the expected cell voltage is 3600 millivolts. The predicted minimum and maximum cell

voltages at this 60% SOC are 3405 millivolts and 3705 millivolts, respectively. Appendix F provides LabVIEW VI and TestStand reports for reading voltage cell and battery temperature sensors.

Due to the factory's ventilation system maintaining ambient temperature within a typical room temperature range, the internal battery temperature sensors should display nearly identical values. Each battery module contains 6 internal temperature sensors, resulting in a total of 12 sensors in the battery pack. According to Appendix F, the acceptable temperature range is between 20°C and 24°C, with a temperature variation of 3°C

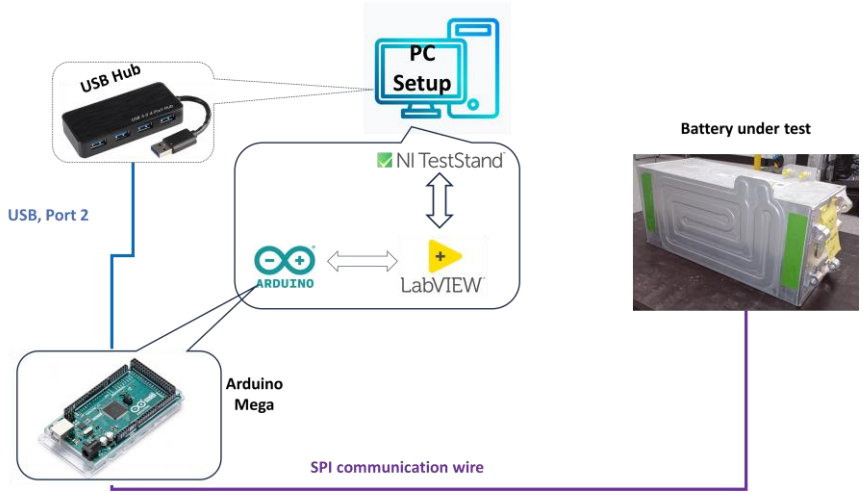


Figure 42. Arduino Mega monitors cell voltage and temperature sensors of the battery pack

5.5 Writing / Reading EEPROM information

In Subsection 3.5.1, it was noted that the EEPROM can store important information about the battery pack, such as its serial number. Figure 14, which illustrates SPI communication, demonstrates how each byte is transferred. Specifically, in Figure 14, the serial number "PROTO1122", scanned from a QR code, is transferred byte by byte via SPI protocol into the EEPROM memory.

Appendix G includes the LabVIEW VI and the TestStand report for EEPROM writing and reading. Table 7 illustrated the registry codes of the reading and writing commands for the Arduino Mega. As shown in Figure 43, the scanned QR code is transferred from the TestStand environment to the corresponding LabVIEW VI, then via USB serial communication to the Arduino script defined for the Arduino Mega (acting as the

controller unit). The data is then transmitted via the UART protocol to the Arduino Micro (which simulates the battery module's processor) and finally reaches the external EEPROM, which is an Arduino Nano, embedded on the CMC board of the battery module.

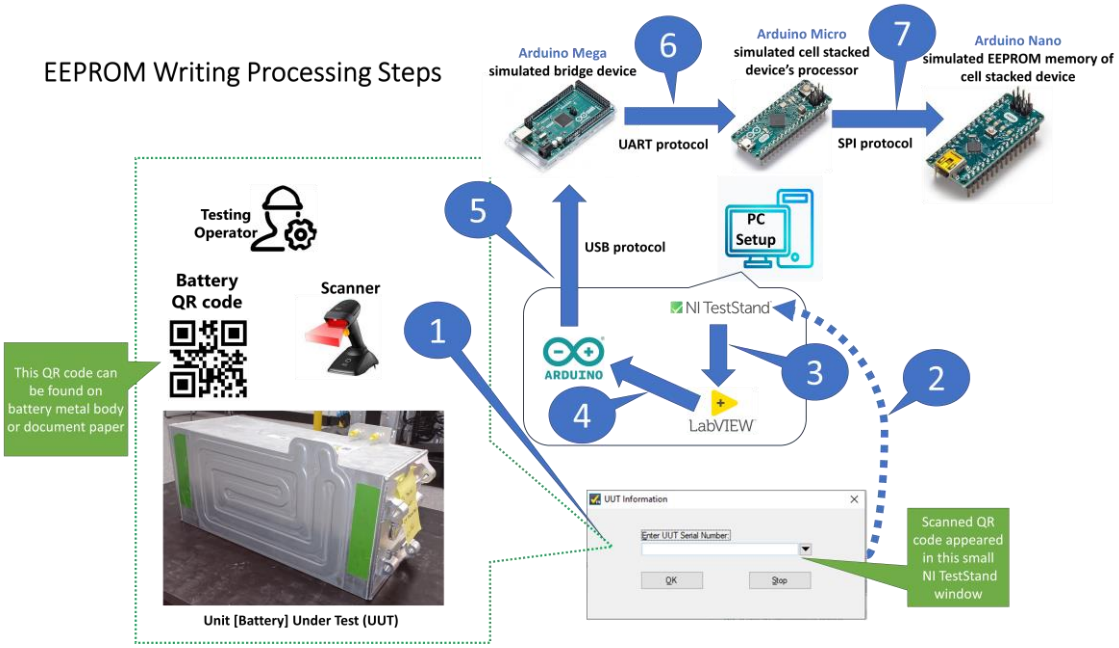


Figure 43. EEPROM writing processing steps from TestStand environment to EEPROM memory

5.6 Equipotential test measurement

The equipotential test, also known as the ground bonding test, ensures that the entire EV battery pack maintains the same electrical potential. This reduces the risk of voltage differences and electrical shock hazards. This test is crucial for safety, as significant voltage differences between the battery pack's metal parts or the vehicle chassis parts (electrically conductive parts) can cause electric leakage, arcing, and potential failures or safety issues. The test uses the Kelvin method to measure very small resistances, typically within the range of 0.01 Ohms [44].

Table 4 in the thesis setup indicates that the equipotential test can only be performed in certain cases. The exterior side of the battery pack, as shown in Figure 19, is metal and homogeneous, so no voltage difference is expected from it. However, when with a precisely calibrated torque device this homogeneous battery pack is mounted with other battery packs using a metal structure plate, a voltage difference may occur across the whole. This is when the equipotential test is necessary. Figure 44 illustrates the need for secure fastening between the battery pack and the structure plate. The equipotential test

can guarantee that the junction between the battery pack and the structure plate is securely fastened, and no voltage variance can happen between these two metal parts.

Figure 44. Equipotential test is required to ensure that the torque tool has securely fastened the battery pack and structure plate

To ensure accuracy due to the stringent resistance criteria of battery metal chassis, it is advised to use a four-wire (Kelvin) measurement. A two-wire resistance measurement is unsuitable for this test. This is because the resistance measurement focuses on the conductive parts of the battery chassis, which have very low resistance (in the milli-ohm range). The criteria can be smaller than the resistance of the wiring itself. Even long conductive wires can have greater resistance than the junction between the battery metal plate shield and battery structure plate shown in Figure 44. The Kelvin (four-wire) method separates the four wires into pairs: current-carrying (Force) wires and voltage-carrying (Sense) wires. In the current-carrying wires, the current is high, and the voltage is low, while in the voltage-carrying wires, the voltage is high, and the current is low.

In a series circuit in Figure 45, 2-wire measurement (in left side) includes both resistance values of (R_{wires}) and ($R_{subject}$); however, in 4-wire measurement (in right side) is only focused on ($R_{subject}$). In 4-wire measurement, the current is consistent throughout in crurrent-carrying (force) wires. When measuring voltage across the specific resistance being tested, while excluding the resistance of the connecting wires (R_{wires}), the resulting value represents solely the resistance of the target component ($R_{subject}$). This ensures that only the intended resistance is reflected in the measurement. The goal was to measure this resistance from a distance, thus requiring the voltmeter to be placed near the ammeter, connected across the target resistance with another pair of wires that have

their own resistance. Initially, it appears 4-wire measurement method reintroduces stray resistance into the circuit because the voltmeter measures voltage through long, resistive wires. However, closer analysis reveals that this is not a problem since the voltmeter's wires carry minimal current. By dividing the voltmeter reading by the ammeter reading, the target resistance ($R_{subject}$) can be determined [45].

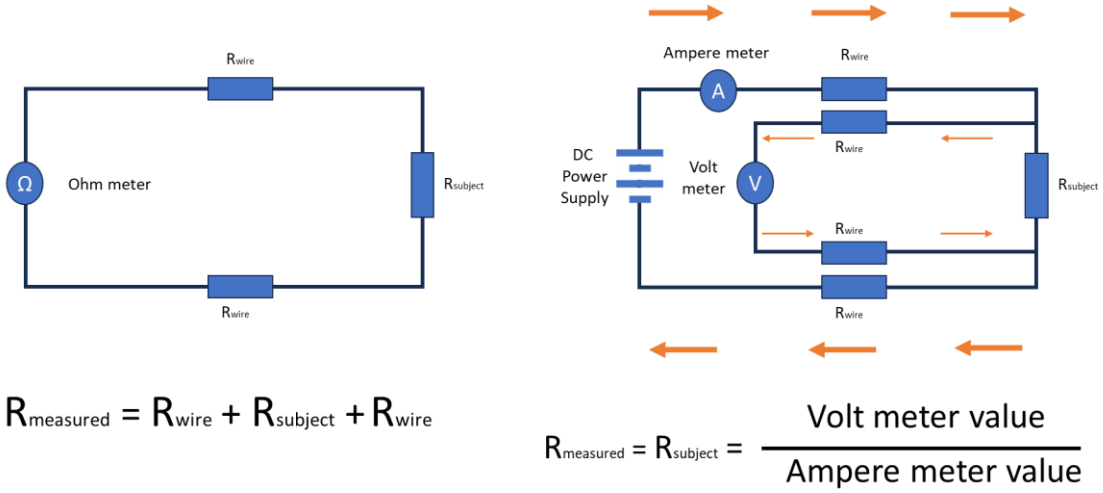


Figure 45. Comparison between 2 wires and 4 wires method for small resistance measurement

Figure 46 shows the elements used for the equipotential test. In Appendix H, the VI programmed for equipotential test is represented. All testing VI files were downloaded from testing device's website, GW Instek (introduced in Figure 17) [29]. All these LabVIEW VIs were modified to be used later in TestStand environment (including equipotential VI). USB port 5 is the communication between testing device and PC setup. VI commends the test and live measurement is stored in PC setup. The test results and measurement values are stored in TestStand report. The test conditions had been outlined in Table 3. Due to electromagnetic field presence, DC measurement in Equipotential test can be more accurate than AC measurement, if all 4 wires are short. However, the safety tester device, GW Instek was supporting only AC measurement for Equipotential test.

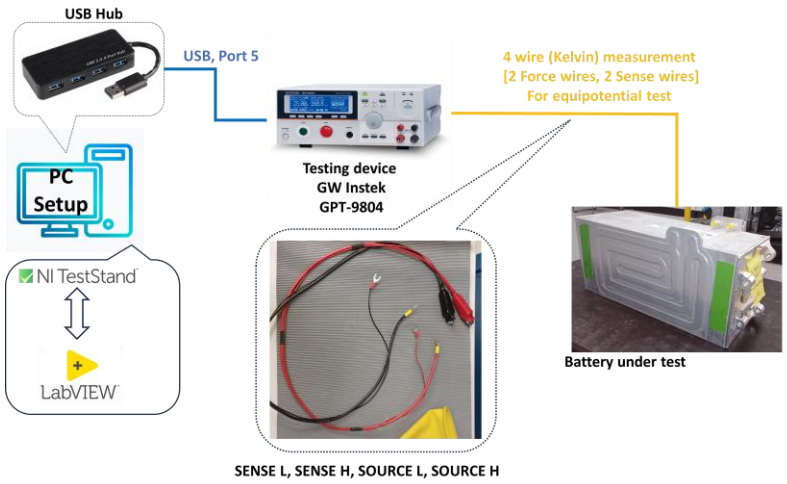


Figure 46. Components applied for equipotential measurement

Figure 47 illustrates the connection of four wires from the testing device to the battery, clarifying the setup.

- Connect SOURCE H to one end of the ground bond.
- Connect SOURCE L to the other end of the ground bond.
- Connect SENSE H close to where SOURCE H is connected.
- Connect SENSE L close to where SOURCE L is connected.

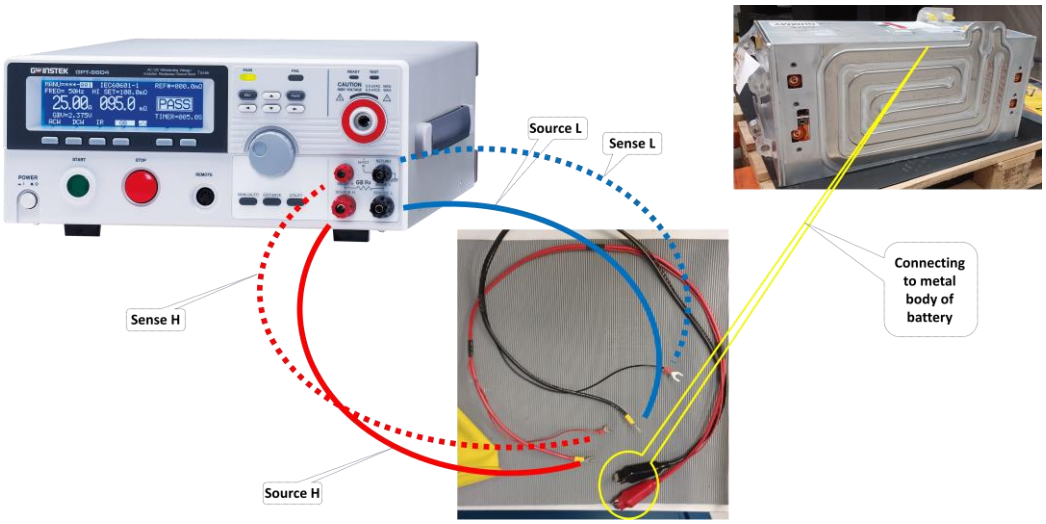


Figure 47. Four wires [Kelvin] method for equipotential test

Appendix H, TestStand report, involved an 8 Ampere AC current for a duration of 5 seconds, with an impedance measurement of 3.4 milliohms, which is below the maximum limit of 5 milliohms. The AC current frequency was set at 60 Hertz, and the communication speed was 115200 baud [this baud rate sustains for all tests]. The graph measurement can help identify any noise in the test results. Typically, the results should

be steady in this test because the measuring components are conductive, and the impedance measurement should remain stable. The graph indicates that the current smoothly ramped up to the optimal level within the first Deci second, after which both the current and impedance measurements stabilized at 8 Amperes and 3.5 milliohms, respectively.

5.7 Closing Normally open [Battery Plus side]

To start isolation test, the electrical flow circuit should be conducted via relays. Here firstly, the normally open relay connected to positive terminal of battery is closed to conduction the flow circuit for isolation test and dielectric test from plus terminal of battery to the ground point. Arduino shield provides ground and V_{cc} connections for relays but controlling relays' signals are from Arduino Uno. Arduino script regarding activation and deactivation of relays can be found in Appendix I. In Appendix I, the respective VI for communication between the Arduino script and TestStand is represented.

In this step, LabView, running in TestStand, sends the character variable "1" via serial communication (Port 3) to the Arduino Uno. Upon receiving this signal, the Arduino Uno sets Pin 2 to "High," activating a Normally Open (NO) relay (see Appendix I). This causes the NO relay to close. The relay is connected to the positive terminal of a battery, completing the electrical circuit needed for isolation and dielectric testing. Figure 48 illustrates the components involved in this step of the test sequence.

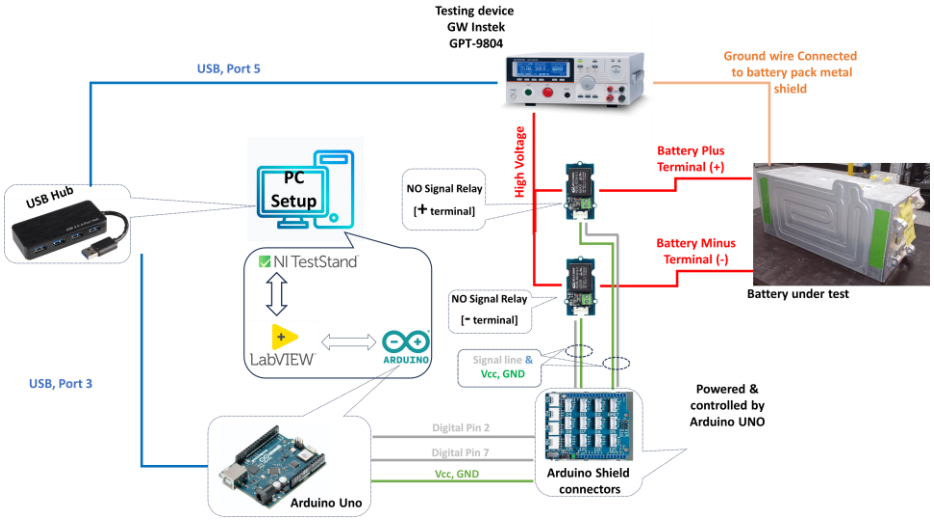


Figure 48. Terminal relays functioning and components for isolation and dielectric tests

5.8 First isolation test measurement [Battery Plus side]

Isolation (or insulation) test measurement is a safety test assuring the battery is safely galvanized from the outer side. This test is done for both battery terminals (plus and minus). The test wiring for plus side is depicted in Figure 49. The ground connection could be anywhere on battery pack's metal chassis.

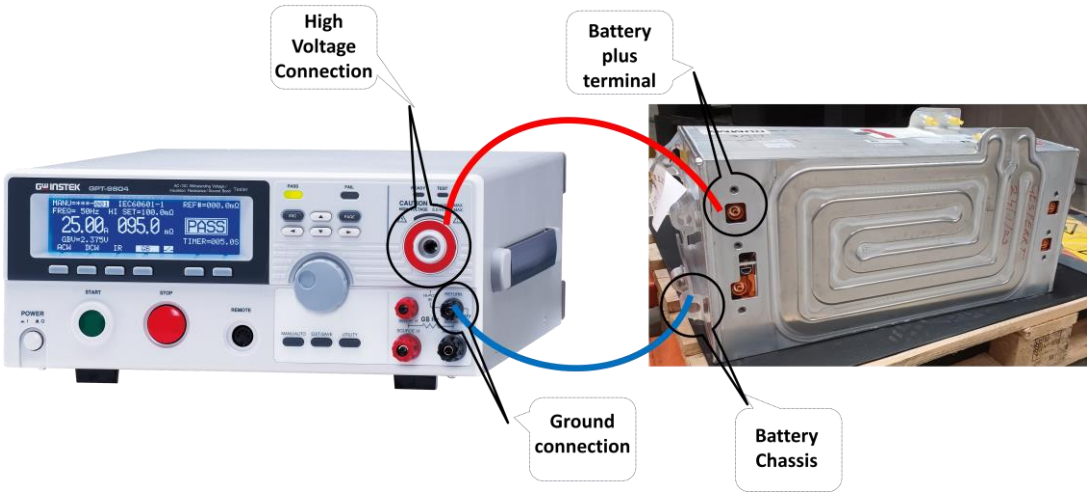


Figure 49. Isolation & dielectric test wiring between safety testing device and battery under test [battery plus side]

Appendix J shows VI provided by GW Instek testing device with modifications to be used later in TestStand environment. USB port 5 is the communication bridge between the PC and the testing device. Figure 50 explains the reasons for the gradual increase in isolation resistance test results over time witnessed in Appendix J. This gradual increase during the isolation test is due to three main factors:

1) Polarization of insulation materials (short-term transient):

It is important to note that insulation materials can become polarized when subjected to high voltages. This process draws a temporary current known as the absorption current, which decreases over time as polarization stabilizes. The absorption current shown in Figure 50 refers to this polarization behavior [46].

2) Capacitive behavior of the battery (medium-term transient):

Some electrical devices exhibit capacitance or inductive characteristics, influenced by factors such as wiring configuration, energy storage, insulation, and conductivity. In this case, the EV battery behaves like a capacitor. The capacitance charging current flows initially to charge this "capacitor" and it decays relatively quickly [46].

3) leakage current (permanent):

Conduction or leakage current, a weak but permanent phenomenon, occurs in strong insulation materials when exposed to high voltage. If the insulation is not robust enough, the leakage current can compromise the insulation, leading to conductivity [46].

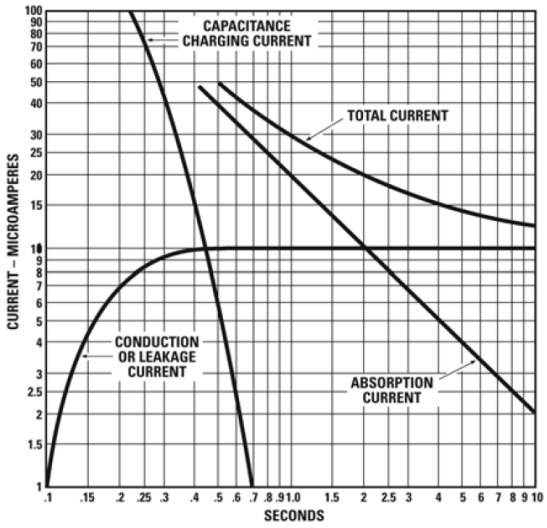


Figure 50. Three reasons behind increasing insulation resistance over time [46]

Appendix J details the test conducted at 500 DC volts for a duration of 10 seconds (as agreed on Table 3), with a ramp-up time of 1 second. The acceptable resistance range was between 100 megaohms and 1000 megaohms, and the final result was 599 megaohms. This confirms that the insulation is reliably safe from the inside out.

5.9 Di-electric test measurement [Battery Plus side]

The dielectric (or high voltage withstand) test follows the isolation test logic to measure leakage current, as discussed in the previous step. This test typically requires a higher voltage, and a shorter duration compared to the isolation test and targets only leakage

current measurement. The components and wiring setup used are the same as those in Figures 48 and 49.

Appendix K shows the modified LabVIEW VI for the dielectric test, which is used in TestStand. The TestStand report indicates that the test was conducted at 1000 DC volts, with a 1-second ramp-up time and a 2-second test duration, and the measured leakage current was nearly zero (as agreed in Table 3). This battery pack required 2000 DC volts to detect a minor amount of leakage current.

5.10 Second isolation test measurement [Battery Plus side]

The second isolation test ensures that the battery insulation remains intact (no damage to insulation) following the dielectric test. While the procedure in the second isolation test is identical to the first isolation test, the key difference is that the insulation measurement in the second test appears more robust compared to the first. This is because dielectric polarization and charge trapping improve the insulation's capacity to resist current flow. As a result, there is less absorption current and capacitive charging current, leading to a higher perceived resistance [46].

5.11 Opening Normally open [Battery Plus side]

In this step, LabView, operating through TestStand, transmits the character variable "2" to the Arduino Uno via serial communication on USB Port 3. When the Arduino Uno receives this signal, it sets Pin 2 to "Low", which deactivates the Normally Open (NO) relay, causing it to open (refer to Appendix I for further details).

5.12 Closing Normally open [Battery Minus side]

It is now time to begin the isolation test for the opposite polarity of the battery. In Figure 49, the NO relay on the positive side opened in the previous step, and now the NO relay on the negative side of the battery needs to be closed. The wiring configuration for the upcoming isolation and dielectric tests is shown in Figure 51. To close the corresponding relay, the VI LabView file, running through TestStand, sends the character variable "3" to the Arduino Uno via serial communication on Port 3. Upon receiving this signal, the Arduino Uno sets Pin 7 to "High", closing the Normally Open (NO) relay.

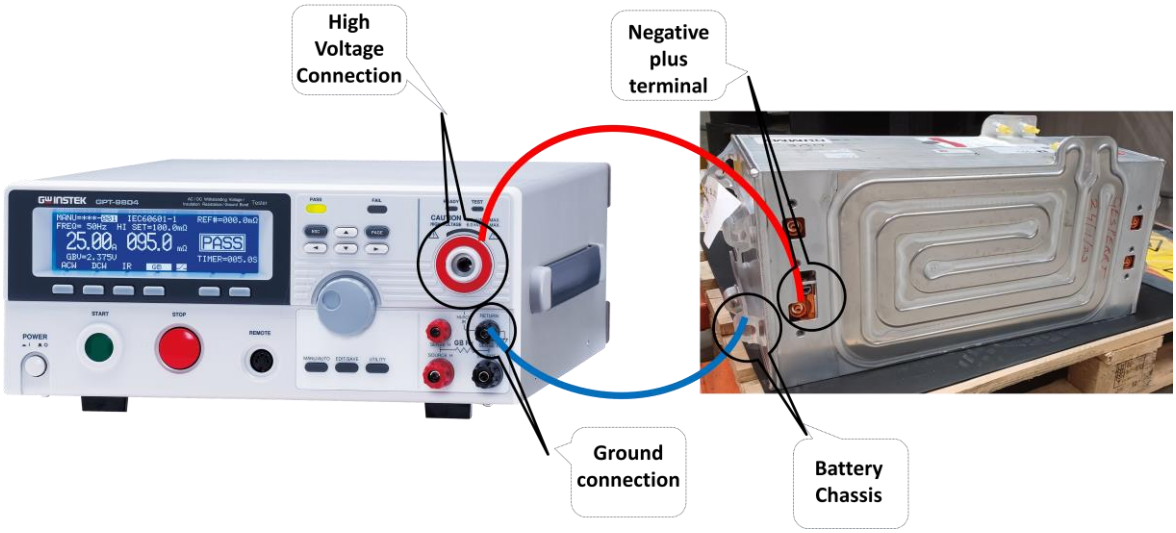


Figure 51. Isolation & dielectric test wiring between safety testing device and battery under test [battery minus side]

5.13 First isolation test measurement [Battery Minus side]

The same VI and TestStand report found in Appendix J can be utilized with a minor adjustment in the TestStand report text, specifying the “Minus side” isolation results. Typically, in EV batteries, the isolation test on the negative terminal shows lower resistance compared to the positive terminal. This occurs because the battery and its circuitry often contain capacitive elements that impact insulation resistance readings. The positive terminal, being less exposed, may experience fewer capacitive effects affecting the measurement compared to the negative terminal. To summarize the Appendix, the results of the second isolation test and the battery's negative side were excluded

5.14 Di-electric test measurement [Battery Minus side]

Accordingly to section 5.9, the same dielectric test happens this time for minus side of battery, measuring the leakage test. In TestStand report will be mentioned this test performed for minus side of battery.

5.15 Second isolation test measurement [Battery Minus side]

The second isolation test is conducted to ensure the battery can withstand the stress of the dielectric test. The Teststand report provides both the measurement results and a graph for this test.

5.16 Opening Normally open [Battery Minus side]

To deactivate the corresponding relay (battery minus side), the VI LabView file, running through TestStand, transmits the character variable "4" to the Arduino Uno via serial communication on Port 3. When the Arduino Uno receives this signal, it sets Pin 7 to "Low", which opens the normally open (NO) relay (see Appendix I).

5.17 Buzzer & LED Alarm activated

After the test sequence is completed, the test setup alerts the operator by activating an LED alarm and a sound buzzer for about 5 seconds. Appendix I in the Arduino script shows that the VI LabView file, running through TestStand, sends the character variable "5" to the Arduino Uno via serial communication on Port 4. The Arduino Uno then sends a "High" signal through digital pins 4 and 5 to activate both elements. After 5 seconds, another VI is loaded in TestStand to deactivate these elements, sending the character variable "6" to the Arduino Uno, which in turn sends a "Low" signal through the same pins. As shown in Figure 52, the buzzer and alarm LED are connected to V_{cc} and ground via Arduino shield connectors, with the signal control connected to the Arduino Uno through the specified digital pins.

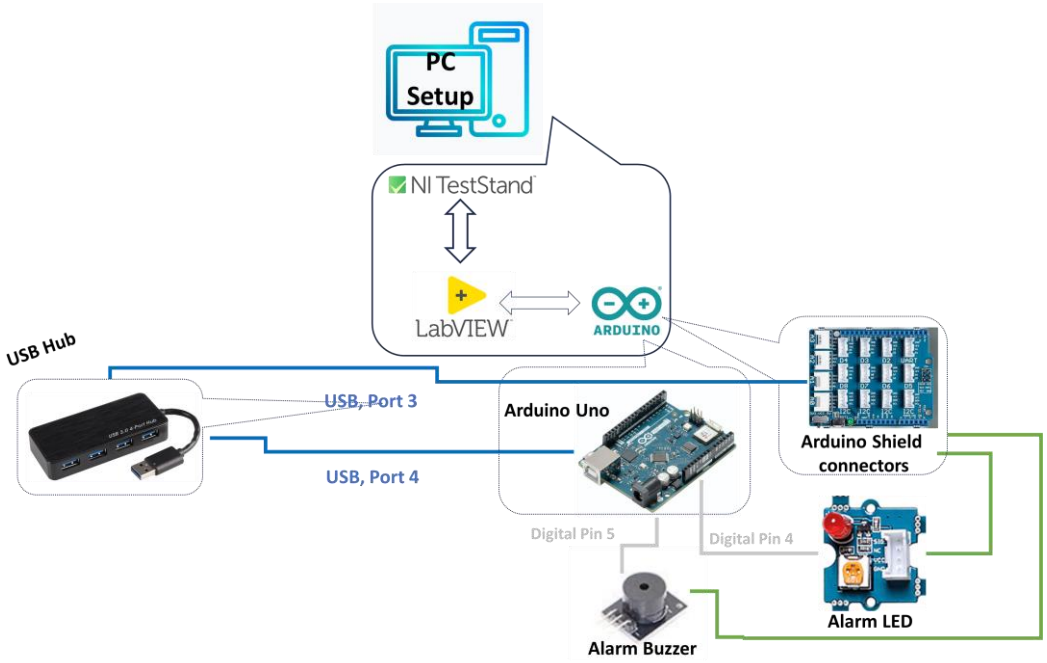


Figure 52. Components under use for activating and deactivating buzzer and alarm LED

5.18 Generating Test Report

Figure 53 illustrates the test sequence in the TestStand environment, following the steps outlined in Figure 37. If the barcode verification step fails, an “if” condition (colored in orange in Figure 53) is triggered, causing the sequence to skip the remaining steps and jump to buzzer step (to notify the operator with message popup window). Similarly, the environment condition step has an “if” condition that halts the sequence if the environment conditions are not met. Additionally, if the battery condition step fails due to out-of-range voltage cells or temperature sensors, or significant differences in their readings between voltage cells (30 milli volts) and temperature sensors difference tolerance (3°C), the test sequence stops before proceeding to write EEPROM information and perform the equipotential test. These conditional steps in Figure 53 were colored in orange.

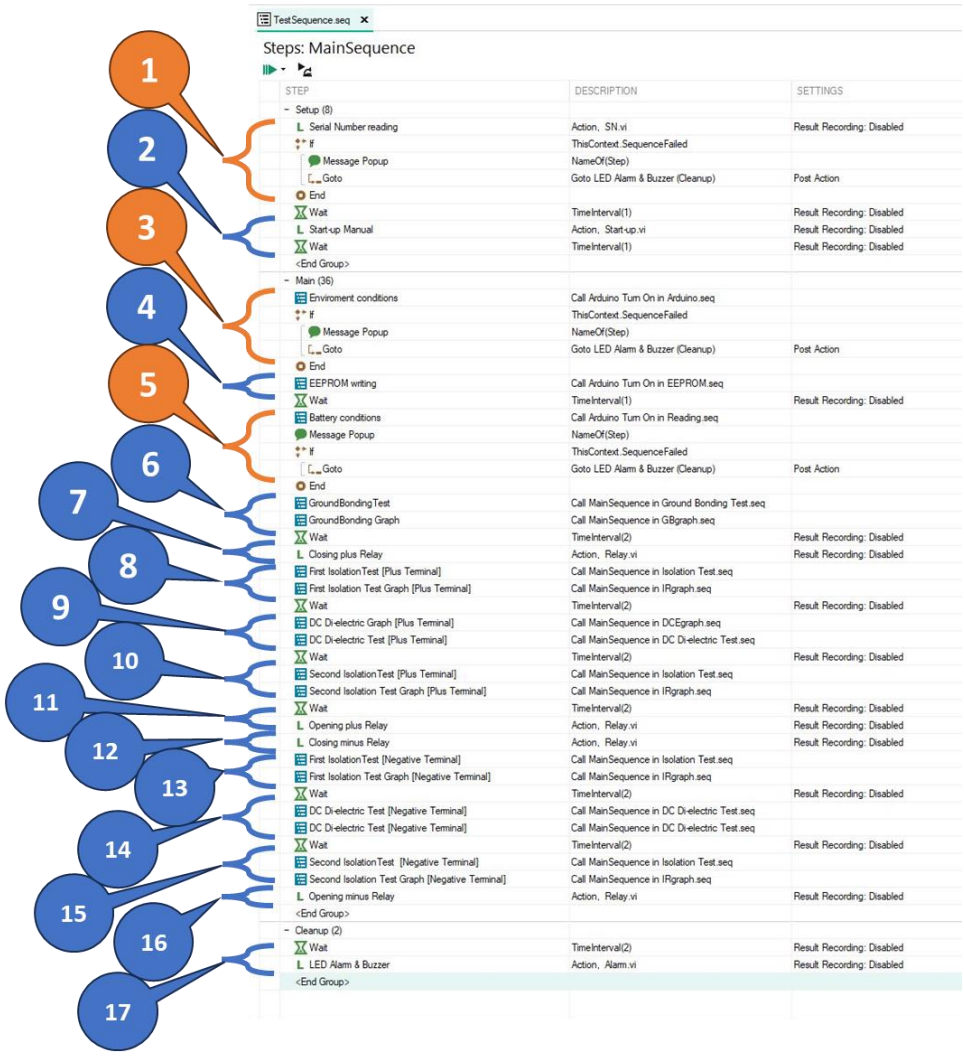


Figure 53. Semi-automation test sequence in NI TestStand environment

6 VPN

A Virtual Private Network (VPN) offers substantial security advantages for corporate networks. To counteract internal and external cybersecurity threats, companies globally now secure their networks with unique VPN credentials for each employee. It is crucial to select a trustworthy local domain service, such as “Pulse Secure” or “Nord VPN”. If a user tries to connect to the automation test setup using public Wi-Fi (which is not recommended), a Virtual Private Network (VPN) can protect the data by creating an encrypted tunnel. This extra layer of security prevents others from intercepting the data transfer. Both remote access and cloud storage features in this automation test setup must function exclusively through the encrypted tunnel created by an authorized VPN.

For Remote Access and data transferring:

When this proposing automation test setup PC uses a VPN, the data is encrypted, ensuring that even if communication is intercepted, it cannot be deciphered. The VPN establishes a secure tunnel between automation test PC and the remote PC, preventing unauthorized access and safeguarding the data from potential threats on public networks. Moreover, a VPN can assign a virtual IP address, making it more difficult for attackers to determine the actual location.

For Cloud Storage Security:

When someone uses a VPN to access cloud storage like Google Drive or Microsoft Azure, the data is encrypted before it leaves from the computer. This means that even if someone intercepts the data, they cannot read it. A VPN also hides the real IP address, making it hard for anyone, including cloud services, to track what data content is being sent. Using a VPN on public Wi-Fi, like in a coffee shop, keeps the connection to cloud storage secure.

In the following chapters will explore how VPN IPs restrict access to remote PCs and cloud storage.

7 Cloud Storage [Microsoft Azure]

Cloud storage, such as Microsoft Azure, is crucial for securely storing and accessing testing report files from anywhere. IONCOR utilizes Microsoft Azure due to its extraordinary features, including scalability, accessibility, security, reliability, and transparency in cost. Microsoft Azure allows storage capacity to grow alongside test data volume, ensures users can access results remotely, and backs up data across various locations to prevent data loss. Moreover, it reduces maintenance expenses, automates tasks like provisioning and disaster recovery, and promotes team collaboration by offering a centralized data repository. These advantages underscore the importance of cloud storage in streamlining and enhancing test data management [47].

The Azure subscription used in this thesis was an Academic Student subscription, associated with a Novia student email. While the process of creating a Student Azure subscription is not detailed and can be overlooked, Section 7 exclusively concentrates on developing a cloud storage **solution**.

7.1 Creating Azure Resource Groups

To commence the setup of cloud storage for storing test reports on the testing setup PC, the first step involves creating an Azure storage resource group. This is done through the Azure portal accessed via a web browser, as illustrated in Figure 54.

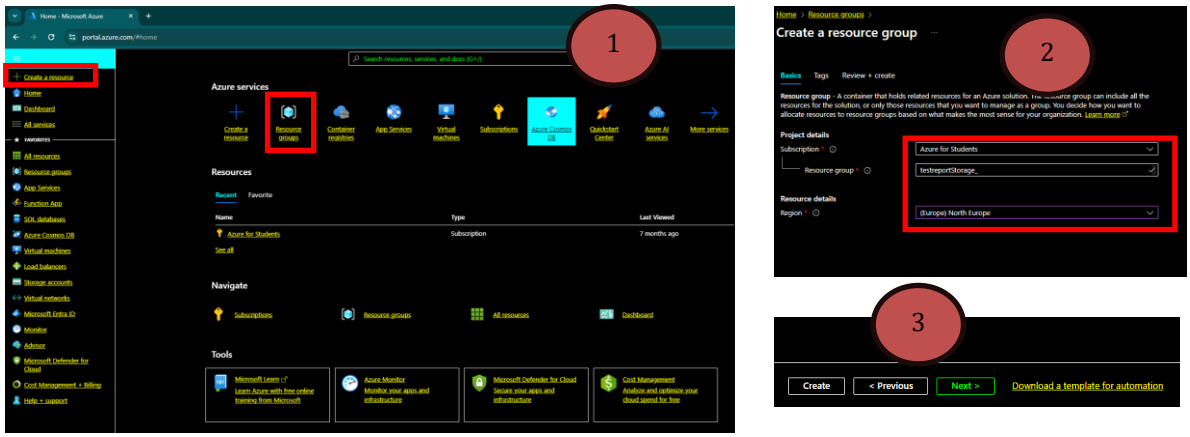


Figure 54. Creating Azure resource group

It should be noted that a Resource Group name must be unique (for more information, refer to [48]), and it was named "testReportStorage". Region was selected North Europe

to have the least latency since the on-premise server (meaning local server, test setup PC HDD) is located in Finland.

7.2 Creating Azure Storage Account

Next step is Azure “Storage Account” creation. Storage Account can contain all Azure Storage data objects such: “Blobs (or containers)”, “Queues”, “Tables”, and “Files”. “Blobs” are used for storing massive amounts of unstructured data, such as text, binary data, and media files. “Blobs” are often used for serving content directly to the web, storing files and streaming video and audio. “Queues” are for storing large number of messages accessible via HTTP or HTTPS. “Tables” are non-relational databases to store structed NoSQL data providing a key/attribute store. In this thesis, Azure Storage Account is used for “Files” only. “Files” are accessible via the industry-standard SMB (Server Message Block) protocol for easy storage and management of data. Azure Files can be mounted concurrently by cloud or on-premises deployments of Windows, Linux, and macOS [49].

Once inside the already created Resource Group in the Azure portal, a Storage Account, can be created as shown in Figure 55.

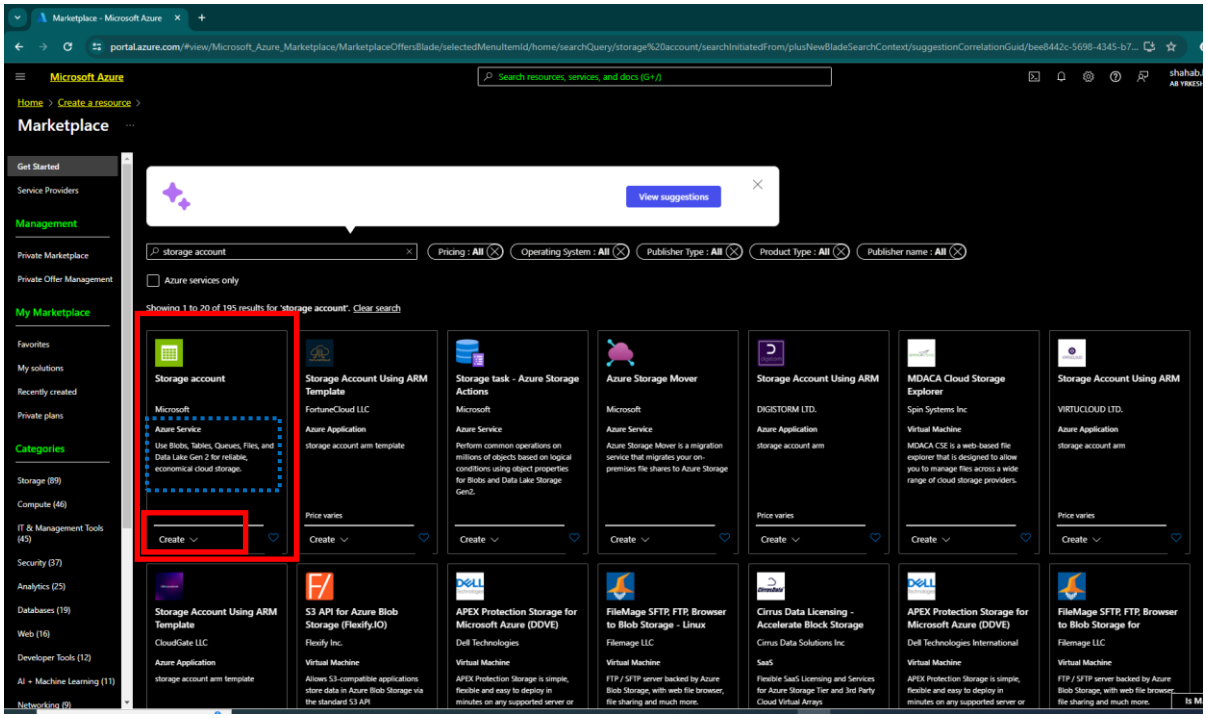


Figure 55. Creating Azure storage account

The new Storage Account name was labelled as “databaseteststorage” and the instance region was selected “North Europe” (see Figure 56). Performance method between “Premium” and “Standard”, “Standard” was selected, due to cheaper service, low rate of Input/Output Per Second (IOPS) and non-urgent latency. Premium is suited for Solid State Drive (SSD) for fast data transfer and low latency; on the other hand, the standard is suited for Hard Disk Drive (HDD) when latency is not critical.

Among Redundancy options, RA-GRS was selected (to be explained more in section 7.2.1). Also, Standard Performance offers several access tiers such as: transaction optimized, hot, and cool, that allows to balance the cost with the performance needs of the application (to be explained more in section 7.2.2). For this application, hot access tier appeared more reasonable. For security reasons, Secure Transfer was enabled. This option ensures a secure connection to data over HTTPs, while insecure connection such HTTP is ignored.

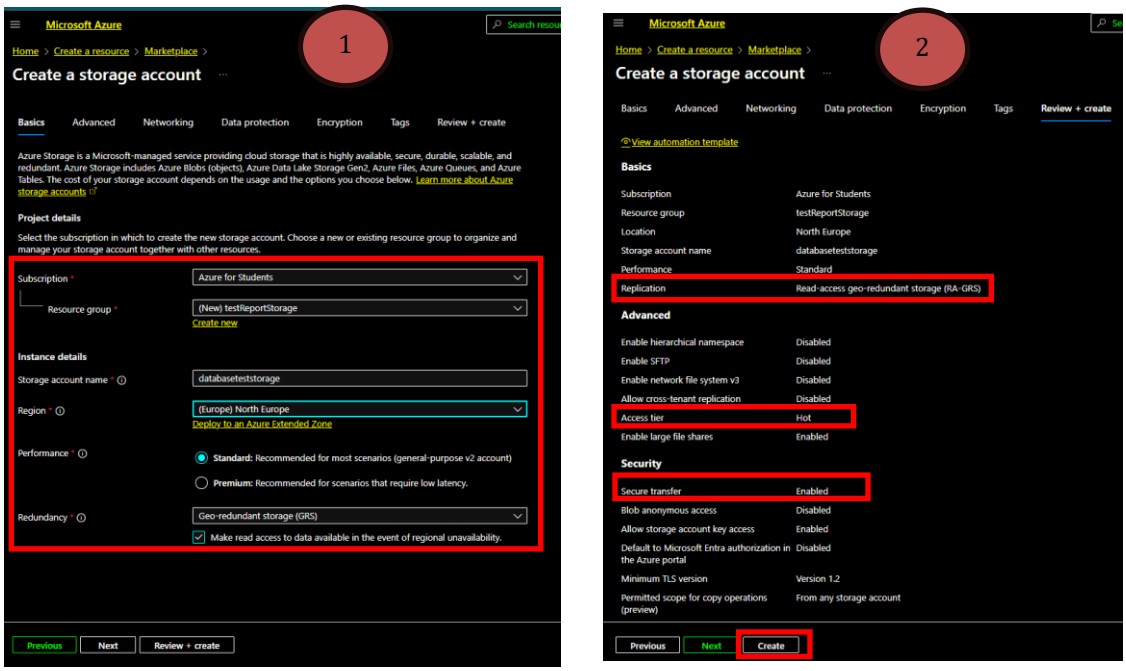


Figure 56. Creating Azure storage account

7.2.1 Microsoft Azure Redundancy

In Microsoft Azure, redundancy involves continuously replicating your data to safeguard against hardware failures, network or power outages, and large-scale natural disasters. Below is a brief overview of the redundancy options available in Azure Storage [50]:

Locally Redundant Storage (LRS): This option replicates the data three times within a single physical location in the primary region. It is the least expensive option and is suitable for scenarios where high availability is not a primary concern.

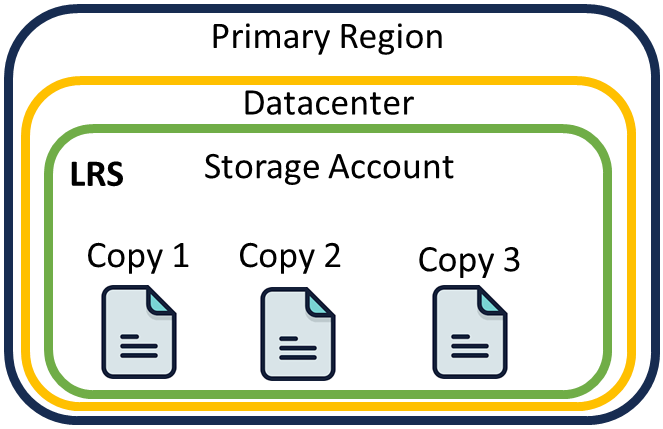


Figure 57. Locally Redundant Storage configuration

Zone-Redundant Storage (ZRS): ZRS ensures the data is duplicated in real-time across three Azure availability zones within the main region. It is the ideal choice for applications that demand maximum uptime.

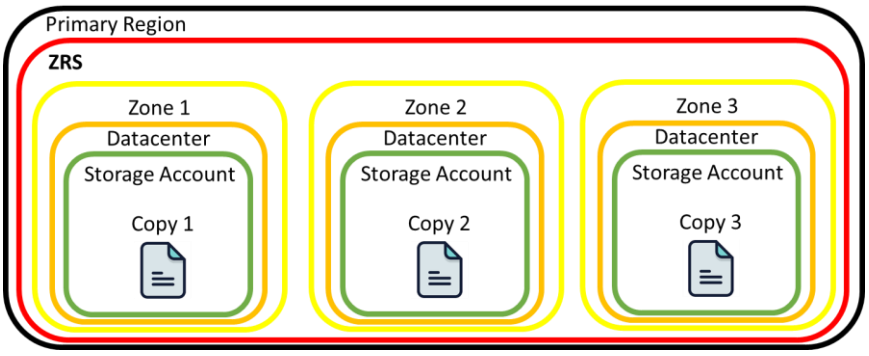


Figure 58. Zone-Redundant Storage configuration

Geo-Redundant Storage (GRS): GRS involves replicating the data to a secondary region located far from the primary one, offering safeguarding against regional calamities.

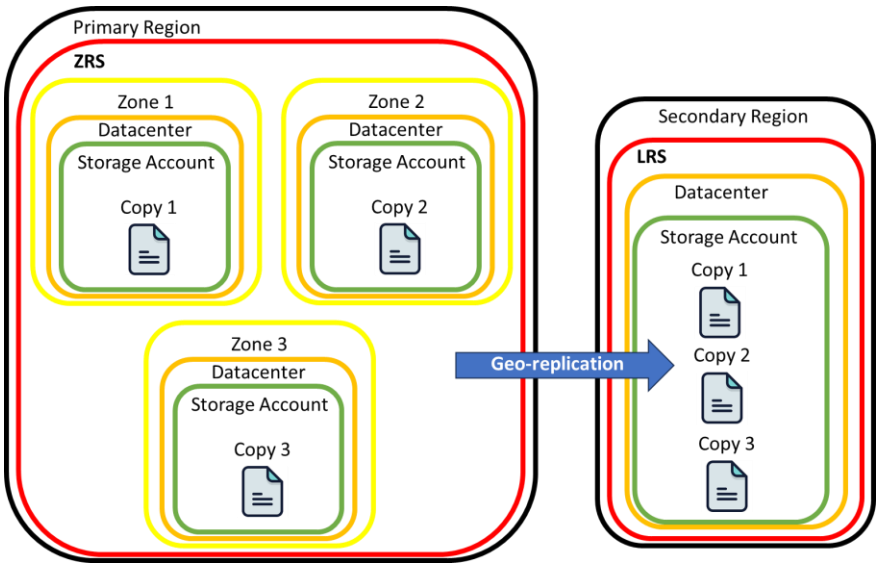


Figure 59. Geo-Redundant Storage configuration

Read-Access Geo-Redundant Storage (RA-GRS): RA-GRS functions similarly to GRS but additionally grants read access to the replicated data in the secondary region in case the primary region experiences unavailability.

Of the options mentioned, RA-GRS is the priciest but deemed more practical for the application's needs, particularly due to the criticality of test report importance. Hence, RA-GRS was chosen for this cloud storage server.

7.2.2 Azure Access tier

Among Azure's access tiers—"Hot," "Cool," and "Transaction optimized"—the Transaction optimized tier is less commonly used. As shown in Figure 56, the "Hot" tier was intentionally selected for this thesis application due to the frequent access to test data by test engineers, test operators, and quality engineers working in two shift routines (16 hours frequent data access per working day). If data stored on the Azure server is accessed frequently, the Hot tier is more cost-efficient than the Cool tier. However, the Cool tier is more cost-effective for cloud storage when data access rates are low. In other words, in the Hot tier, data transfer costs are lower, but storage costs are higher. Conversely, in the Cool tier, data transfer costs are higher, but storage costs are lower [51].

The primary zone has been designated as North Europe, with GRS as the redundancy type. For this automation test setup, Azure Files was chosen as the storage data object. Table 8 illustrates the data storage costs across various access tiers, given the same region and redundancy for Azure File Share. Similarly, Table 9 shows the transaction and data transfer costs under identical conditions. The reference currency for Microsoft Azure is the US dollar, with the current exchange rate between the Euro (€) and the US dollar ($) provided below.

1 USD = 0.9247 EUR

In Table 8, a GiB (gibibyte) is a unit of digital storage measurement based on the binary system, where one GiB equals 2^{30} bytes. This differs from a GB (gigabyte), which is based on the decimal system and equals 10^9 bytes [37].

Table 8. Data storage price comparison in the same redundancy and region for Azure File Share data

Region	**North Europe**			
Redundancy	**GRS**			
Storage Data Object	**Azure File Share**			
Performance	**Premium**	**Standard**		
Access tier	**Premium**	**Transaction Optimized**	**Hot**	**Cool**
Data at-rest (GiB/month) per used GiB	N/A	€0.0925	€0.0470	€0.0278

Table 9. Transactions and data transfer price comparison in the same redundancy and region for Azure File Share data

Region	**North Europe**	
Redundancy	**GRS**	
Storage Data Object	**Azure File Share**	
Performance	**Premium**	**Standard**

Transactions and data transfer	Premium	Transaction Optimized	Hot	Cool
Write transactions (per 10,000)	Included	€0.0278	€0.1203	€0.2405

In Table 8, the provisioned size of a premium file share is specified by the quota field via the API (Application Programming Interface). If the provisioned share size decreases below the used GiB, billing will be based on the used GiB for premium shares.

7.3 Creating Azure File Share

The automation test setup is designed to save various types of test report files, such as XML, PDF, and HTML, created by NI TestStand or other file formats. As previously explained, due to reasons such as 1) varied file formats, 2) lack of strict organization regulation in dataset, and 3) the need for quick accessibility, Azure File Share is preferred over other data objects like Blob Containers, Queues, and Tables. Within a Storage Account, a new Azure File Share can be created. As mentioned earlier, the primary server location is North Europe, with Western Europe considered as the secondary server location for redundancy (see Figure 60).

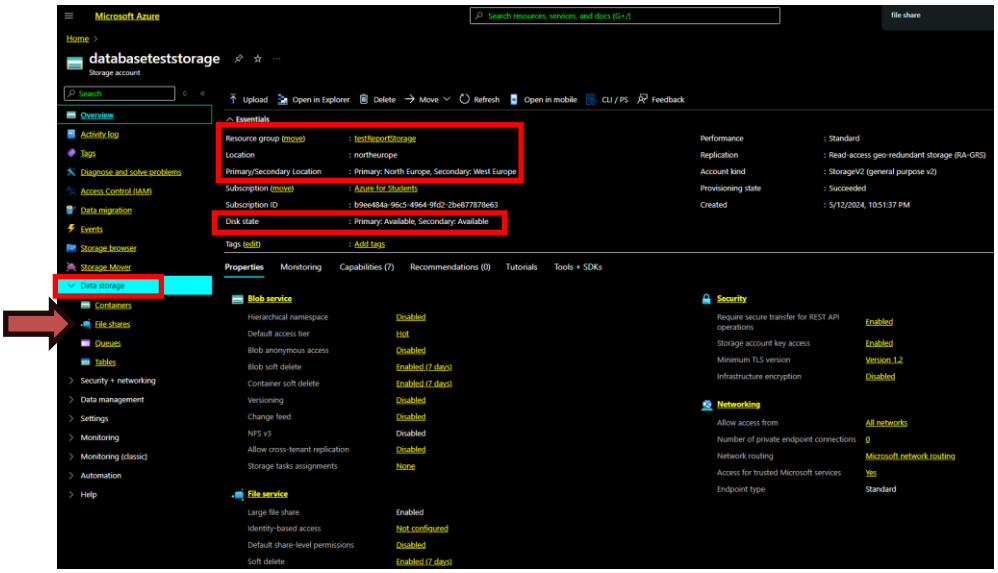


Figure 60. Data storage account settings

A File Share can be created by selecting the + sign shown in Figure 61. The File Share was labeled "testreport." The access tier was intentionally selected as Transaction optimized, as it is the cheapest method for daily data transferring, which occurs actively for 16 out of 24 hours each day. The maximum IO/s (Input/Output per second) is 20,000. Additionally, a backup policy was set for 19:30 every day (normally the last hours of the second afternoon shift), with a retention period of 30 days.

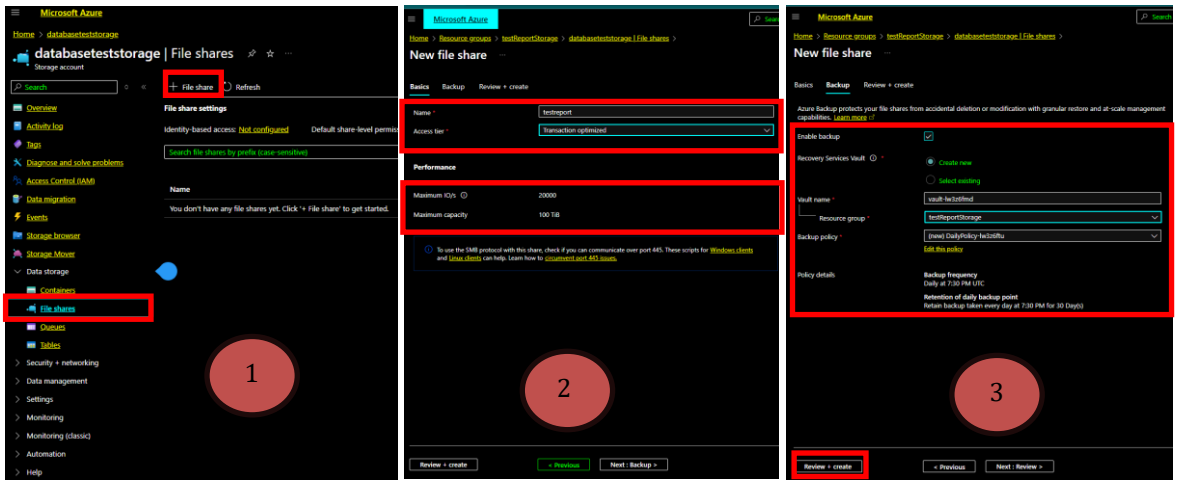


Figure 61. File Shares settings

7.4 Creating Azure Virtual Storage Network Disk on Windows 10

Now it is time to connect the File Share to the operating system. The operating system used in the automation test setup is Windows 10. Within the File Share, using the SMB protocol, select the “Connect” text, then choose the correct operating system and assign a drive letter, in this case, Drive Z. Microsoft Azure states that the Transmission Control Protocol (TCP) for this connection will be assigned to Port 445. Subsequently, copy the generated script for the next step.

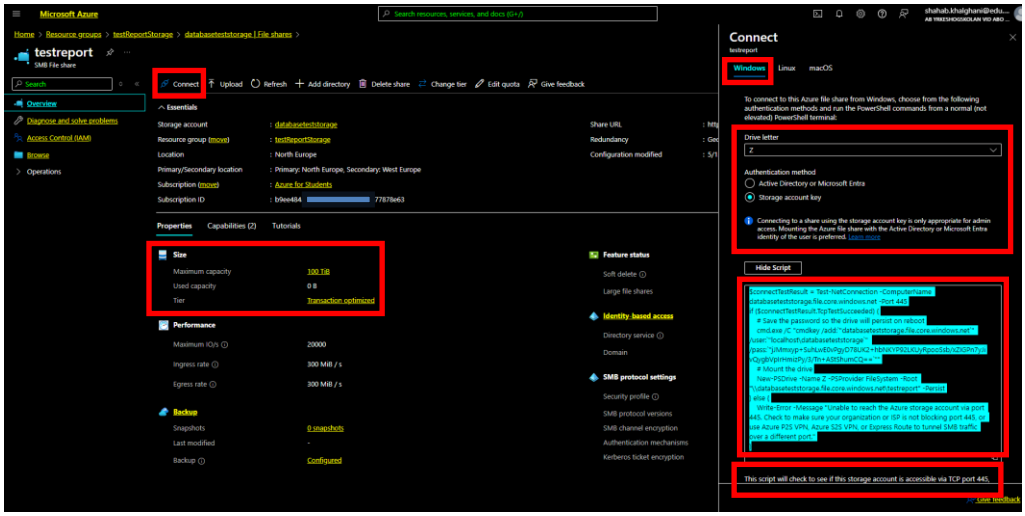


Figure 62. Creating a direct connection of Azure virtual storage to Windows 10

Transmission Control Protocol (TCP):

Transmission Control Protocol (TCP) is a connection-oriented protocol that facilitates communication between different devices over a network. In this thesis, TCP was discussed in the context of the Modbus protocol in Subsection 3.3.4; however, the current focus is on the internet network rather than the Modbus network. As a core component of the TCP/IP suite, this protocol operates on the Network Layer (Layer 3) and Transport Layer (Layer 4) of the OSI model (see Table 5) [52].

TCP works alongside the Internet Protocol (IP), which defines the method for sending data packets between computers. It ensures error-free, ordered delivery of data packets using acknowledgments (ACKs) to confirm receipt. To prevent data overflow, TCP adjusts the transmission rate according to the receiver's buffer size. The TCP header includes a checksum to detect corrupted data and requests retransmission if necessary [52].

TCP is commonly used in applications that require reliable and ordered data transfer, such as web browsing, email, and remote login. Internet Protocol (IP) is a set of rules that governs how data is sent and received over the internet. It is responsible for addressing and routing data packets so they can travel from the sender to the correct destination across multiple networks. Each device has a unique IP address, enabling it to communicate and exchange data with other devices on the internet [52].

Backing to Azure Virtual Storage Network Disk creation:

In Windows PowerShell, the copied script (shown in Figure 62) is pasted and “Enter” is pressed. Windows PowerShell will display "Credential added successfully" for Drive Z, with the Storage Account name "databaseteststorage".

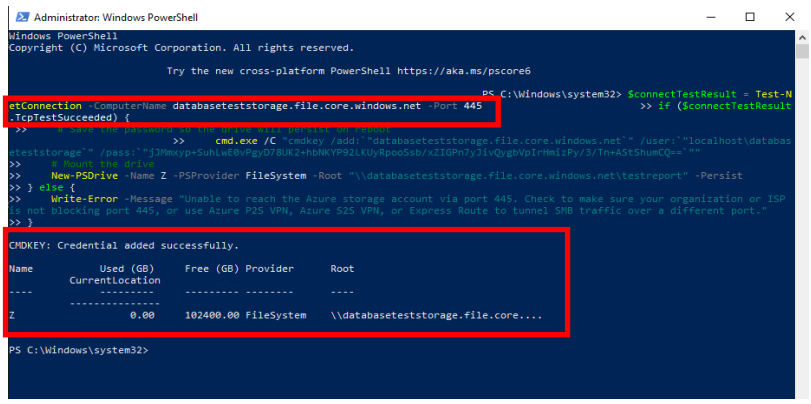


Figure 63. Connecting storage account as virtual storage drive in Windows 10

Then, return to the script and copy the highlighted text. Next, right-click on “This PC” and select “Map Network Drive...”.

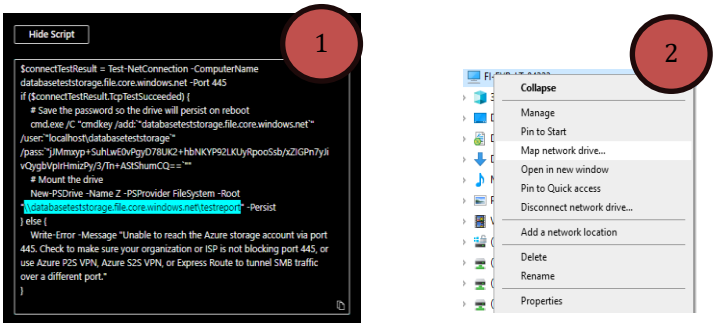


Figure 64. Mapping network drive in Windows 10

The prior action leads to another window where the drive letter "Z" should be selected, and the folder should be addressed from the copied text from the script. Next, in the "Browse" option, the "testreport" folder is selected, followed by pressing "OK". In the "This PC" window, the new network drive appears as expected. Drive Z is the physical location where TestStand reports are locally saved and stored.

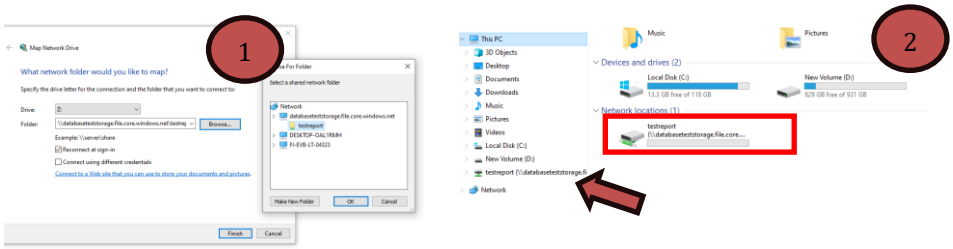


Figure 65. Mapping network drive in Windows 10

To verify the Azure File Share that has been configured, three random files were copied to Drive Z. To confirm their presence on the File Share Server, go to the Azure Portal, access the Azure File Share, and check that the copied files are visible (as shown in Figure 66).

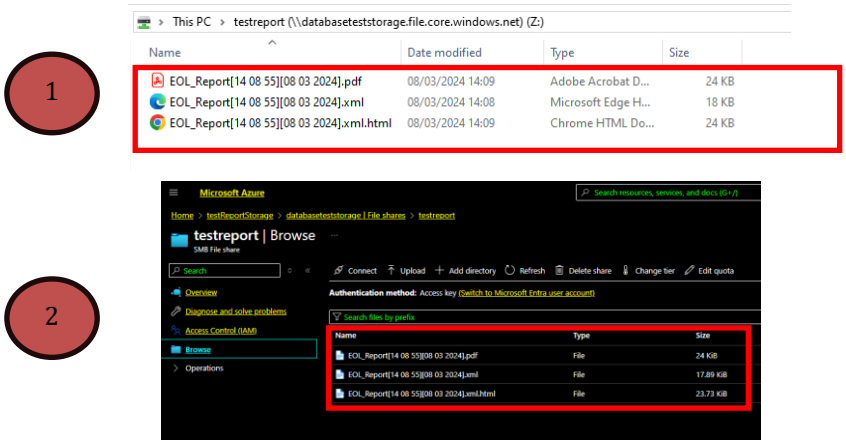


Figure 66. Verifying Azure cloud storage

In Storage Account settings, Access Control (IAM) can restrict user access by assigning rules to users, as shown in Figure 67, or by adding email addresses from a specific domain. Since this Storage Account is part of the Novia University Student Subscription, student emails can easily be added to this Azure Storage Account. Although external users such as clients and third parties can also be granted access, it is advisable not to do this within the organization's Azure subscription. Reference [53] explains how to establish a new Azure Data Share and begin sharing the organization's data with external customers and partners outside of the Azure environment.

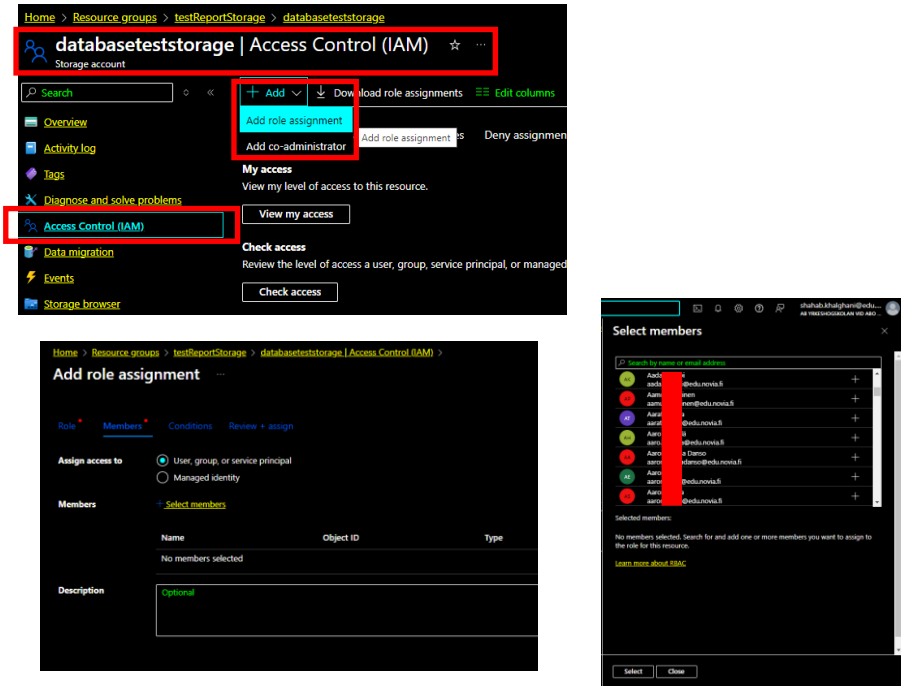


Figure 67. Access account settings in Azure cloud storage

7.5 Access to Azure server only through VPN

Section 6 covered the significance of using a VPN. Due to cybersecurity risks, it is advised that the Azure Storage Account be accessed only through VPN IPs. As depicted in Figure 68, the Azure Storage Account is, by default, open to all networks, but it can be configured to allow access only from specific networks in Azure Firewall setting.

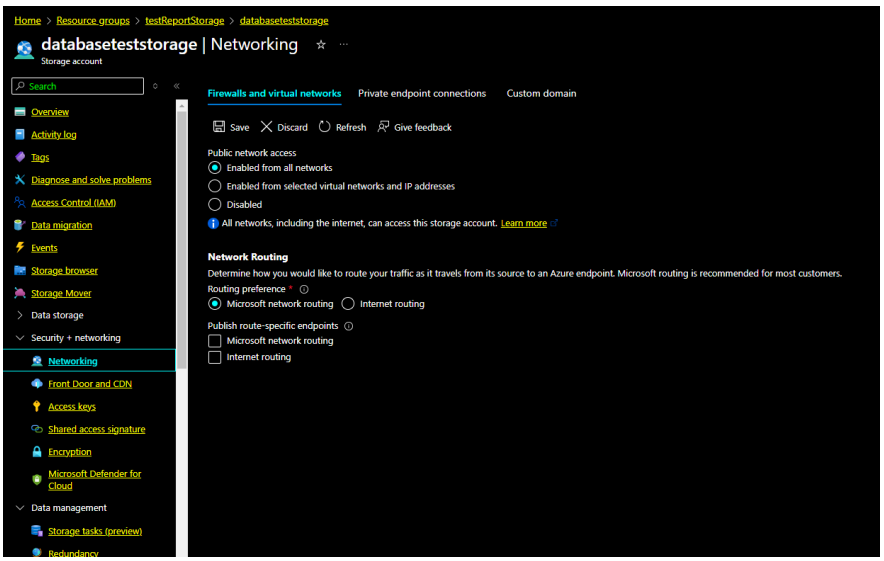


Figure 68. Firewall and virtual network access settings

In Figure 69, the IP range used with the VPN was specified in the Networking settings.

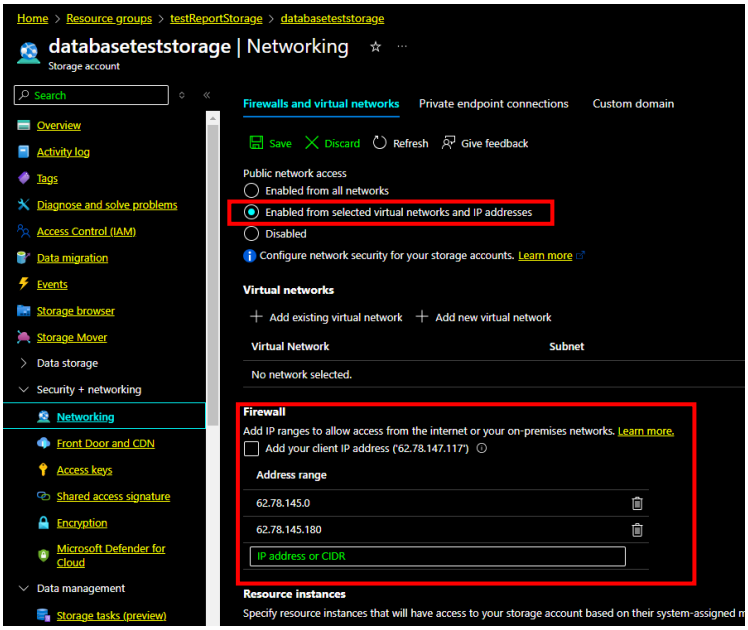


Figure 69. IP address range limitation in Azure cloud storage firewall

To test the new IP settings, the VPN was intentionally turned off, as shown in Figure 70, to verify if access was feasible. As expected, Figure 70 demonstrates that access was blocked, resulting in the anticipated error.

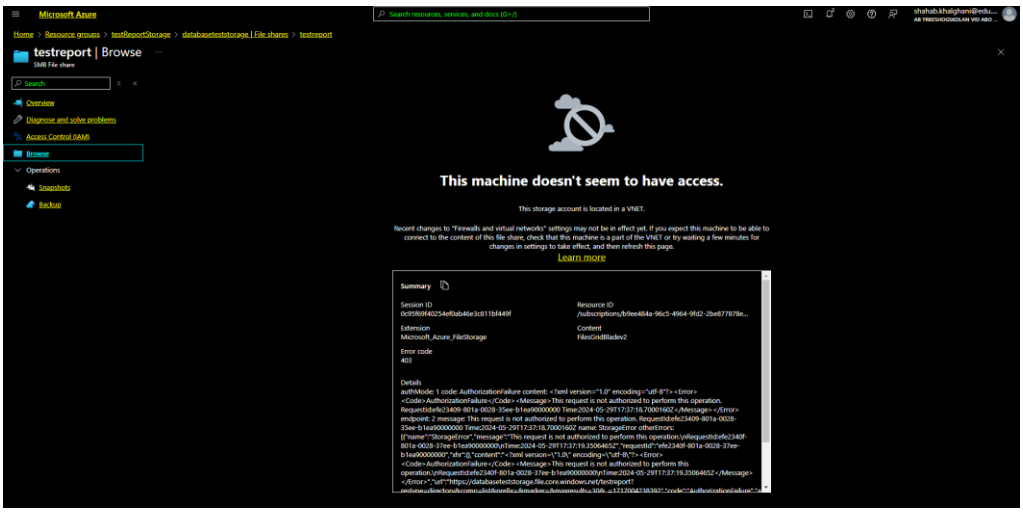


Figure 70. Testing VPN IP address in Azure cloud access

8 Remote Access to Testing Setup

The battery pack is currently in the prototyping stage, which involves numerous modifications until it reaches full maturity. These changes affect not only the battery pack itself but also the software tools used on the test bench. Remote monitoring is required for making edits to these software tools and performing any necessary maintenance.

At times, the PC used for automation testing requires remote observation from an external network for tasks such as debugging, modifications, monitoring, and data transferring. This section outlines the procedures for monitoring via Remote Desktop Protocol (RDP) and File Transfer Protocol (FTP) for various users.

The operating system for the automation testing setup is Windows 10 64-bit. Before using RDP and FTP protocols, it is important to ensure that firewall permissions are configured for both private and public networks. Additionally, the FTP Server feature must be activated in Windows Features window (Figure 71).

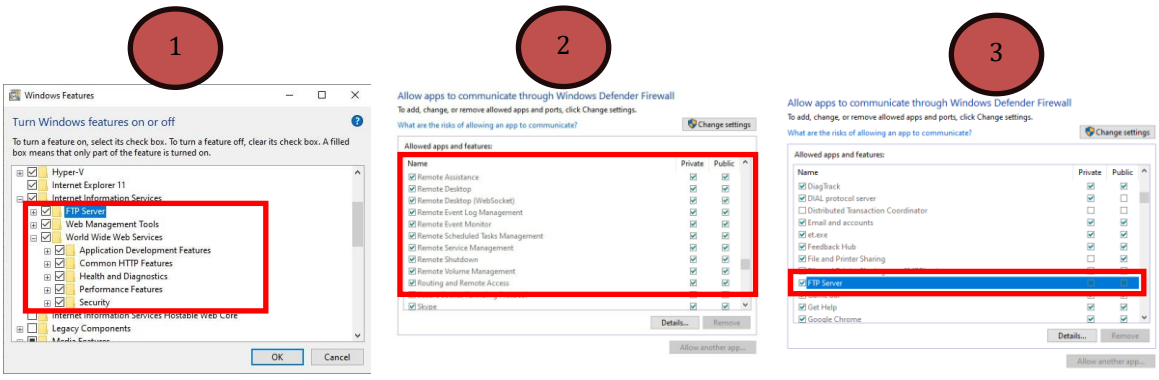


Figure 71. Awarding permissions to FTP server and remote desktop in public and private server

8.1 RDP Protocol introduction

The RDP protocol, developed by Microsoft, is one of the most widely used methods for remote monitoring. It is compatible with a variety of platforms, including Windows, Linux, macOS, iOS, and Android, and allows remote connections to multiple network PCs. While port forwarding is unnecessary when PCs are on the same network, it becomes essential when PCs are on different networks, unless alternative methods to bypass this requirement are employed.

Port forwarding, a networking concept, involves redirecting communication requests from one address and port number combination to another as packets pass through a network gateway, such as a router or firewall. Typically configured in router (modem) settings, port forwarding is an application of Network Address Translation (NAT). However, in large-scale industrial companies, managing port forwarding settings can be time-consuming due to the multitude of routers involved.

8.2 Effective remote monitoring for automation test setup

For effective remote monitoring, certain criteria should be met:

- Minimal latency between the host and the remote user
- Affordable remote service subscription
- Tracking of remote user activity
- Robust cybersecurity protection
- IP access restriction within an authorized VPN domain
- Compatibility with various operating systems
- Virtual port forwarding capabilities
- The ability to monitor an on-premises PC while allowing remote user access
- Support for multiple remote users simultaneously
- An intuitive interface for industrial company Help Desk administrators
- Use of an additional IP domain for enhanced cybersecurity (another extra IP layer protection separate from VPN IP)
- Support for future open-source applications for remote control

Among the third-party platforms available, TailScale [54] met the previously mentioned requirements as a mediator between PCs on different networks for RDP communication. TailScale converts external IP networks into its own IP domain, effectively acting as virtual port forwarding for the RDP protocol. It utilizes open-source software to enable connections across different networks, allowing users to access host PCs, set rules and restrictions for each user, and monitor user activity. In this thesis, the free standard service package of TailScale was selected, though businesses can opt for enterprise services that offer exclusive domains and additional features.

8.3 Third party company for a better service

Figure 72 illustrates that TailScale is compatible with various operating systems including MacOS, iOS, Windows, Linux, and Android. As the testing setup PC operates on Windows 10, the Windows version was downloaded accordingly and installed. Both user and host parties need to install this user-friendly application on their system.

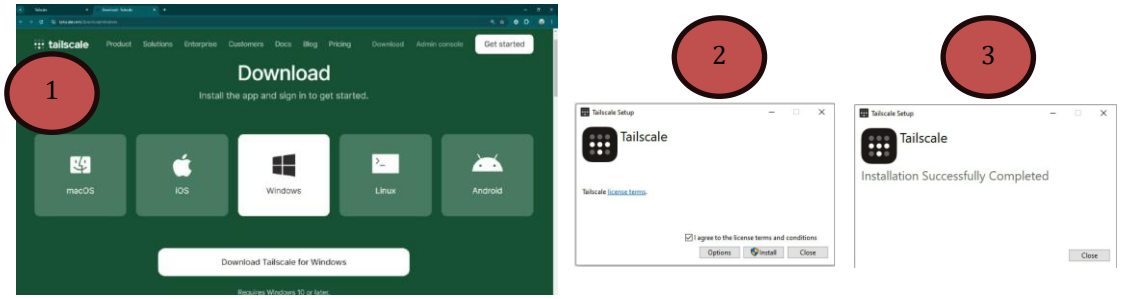


Figure 72. Operating systems working in TailScale server

Figure 73 below demonstrates how TailScale application can be logged in Windows 10 (Taskbard) from either a host or remote PC.

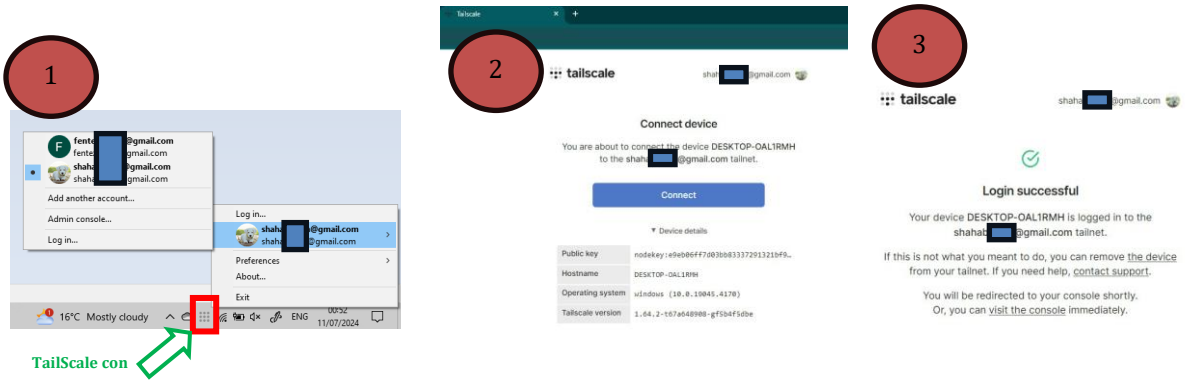


Figure 73. logging in on Taskbar for TailScale server

In Figure 73, after completing the login process, the public key and hostname of the host PC are shown. Figure 74 illustrates the initial home page of the TailScale environment, which includes a "Machine" section where additional machines (both remote and host PCs) can be added to the TailScale web server domain. Additionally, there is an option to switch the roles of remote and host machines on the TailScale server if necessary.

TailScale is also compatible with Docker containers and private GitHub repositories. In Figure 74, the "Apps" section allows open-source applications (including those from GitHub repositories), VPNs, or company interfaces to be integrated into the TailScale environment, with traffic monitoring capabilities. The "Service" section enables real-time monitoring from local hosts through a browser, without requiring a remote user's computer, which is beneficial for HelpDesk members. In the "Users" section, permissions and access can be assigned to remote users based on their roles.

In the "Access Control" section, monitoring of local computers can be shared externally, such as with the internet, using TailScale Funnel and SSH security protocols, which can be configured in this section. The "Logs" section offers user tracking details, including log-on and log-off times for each user. In the "DNS" section, custom DNS domains can be purchased, and HTTPS certificates can be imported. Furthermore, the server domains for the open-source applications added in the "Apps" section must be defined within the "DNS" section.

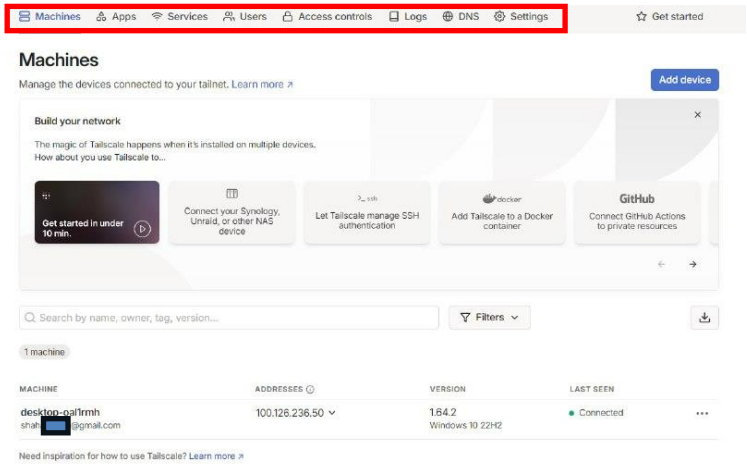


Figure 74. Home page settings in TailScale

In Figure 75, the user permission accessibility feature of TailScale is displayed. Remote users can be invited to join a network via their email addresses. As mentioned earlier,

these users need to install the TailScale interface beforehand. Figure 75 also shows the invited member can access remote connection with host server.

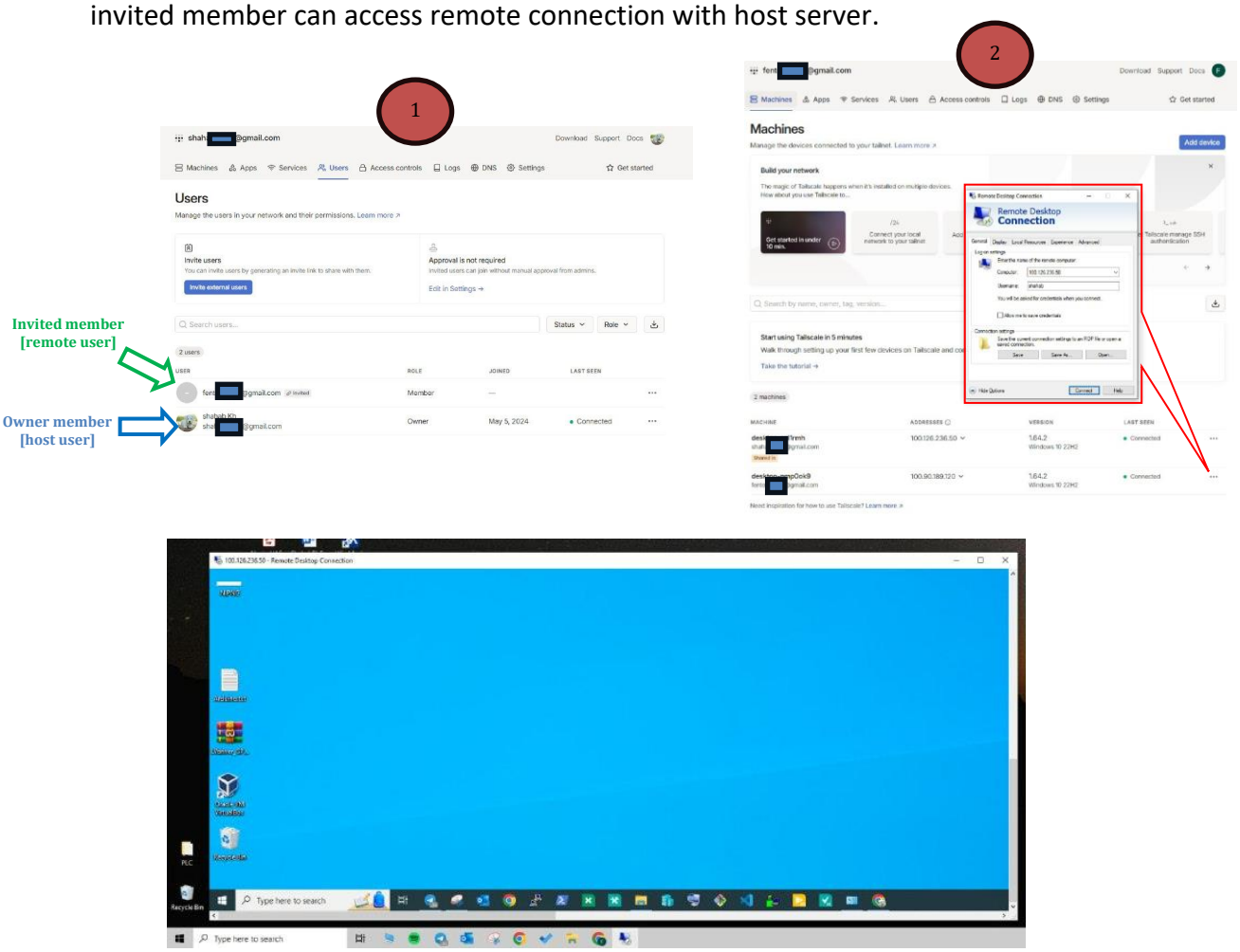


Figure 75. User settings in TailScale and connecting remotely

9 Remote Data Transferring to the Testing Setup

In section 8, the File Transfer Protocol (FTP) was introduced for remote data transfer. This protocol is well-known for this purpose and has undergone numerous updates over the years to enhance its cybersecurity features, unlike the TCP protocol. Section 9 explains the FTP protocol and demonstrates how to configure it.

9.1 FTP Protocol

Security always takes priority over simplicity. For transferring files between host and remote computers, the FTP (File Transfer Protocol) is commonly used, whereas RDP (Remote Desktop Protocol) is not recommended for this task. It is not recommended that

company data is transferred via the third-party service which is in use for remote monitoring service. FTP is specifically designed for file transfers, and there are more secure versions like SFTP (SSH File Transfer Protocol), which uses SSH encryption, and FTPS (FTP Secure), which combines SSL/TLS encryption with the original FTP protocol.

9.2 FTP Protocol configuration in ISS environment

In the IIS (Internet Information Services) Manager window, FTP communication can be configured and managed.

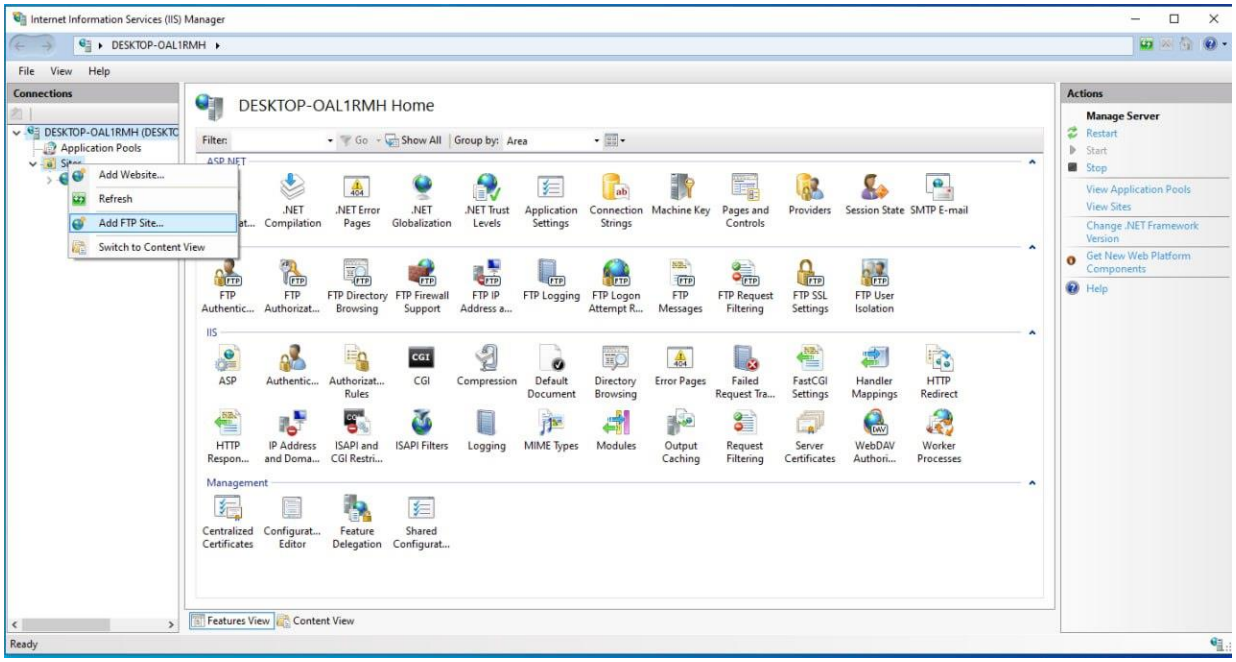


Figure 76. IIS window for creating FTP site

Next, a desired physical directory can then be added to the FTP site. The desired physical directory was preferred, a folder name in the desktop screen, namely “FTP Data Transfer”, where TestStand reports are saved and stored in this folder. The name of FTP site was labeled. FTP protocols occasionally use port 21. The target IP address is provided. In this case, only a basic FTP site setup was implemented without an SSL certificate (Figure 77).

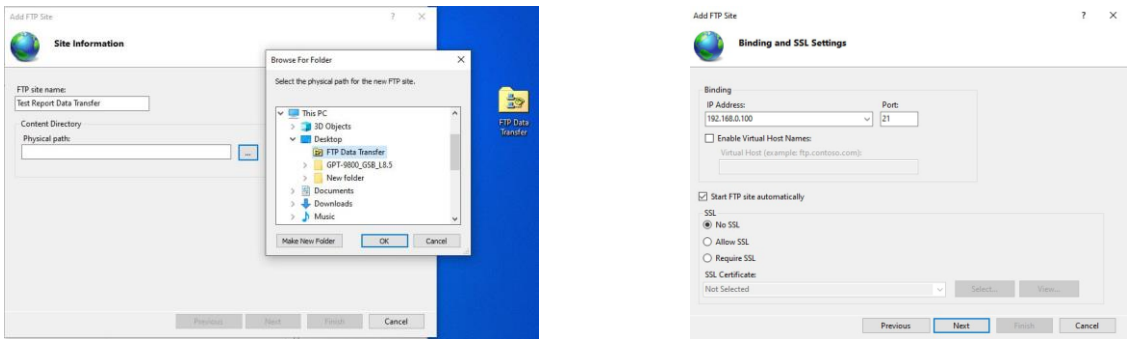


Figure 77. Adding physical path, IP & port in FTP site

Deliberately, username-based access was implemented for FTP file transferring, providing read and write permissions. The FTP site has now been created and is ready for implementation (Figure 78).

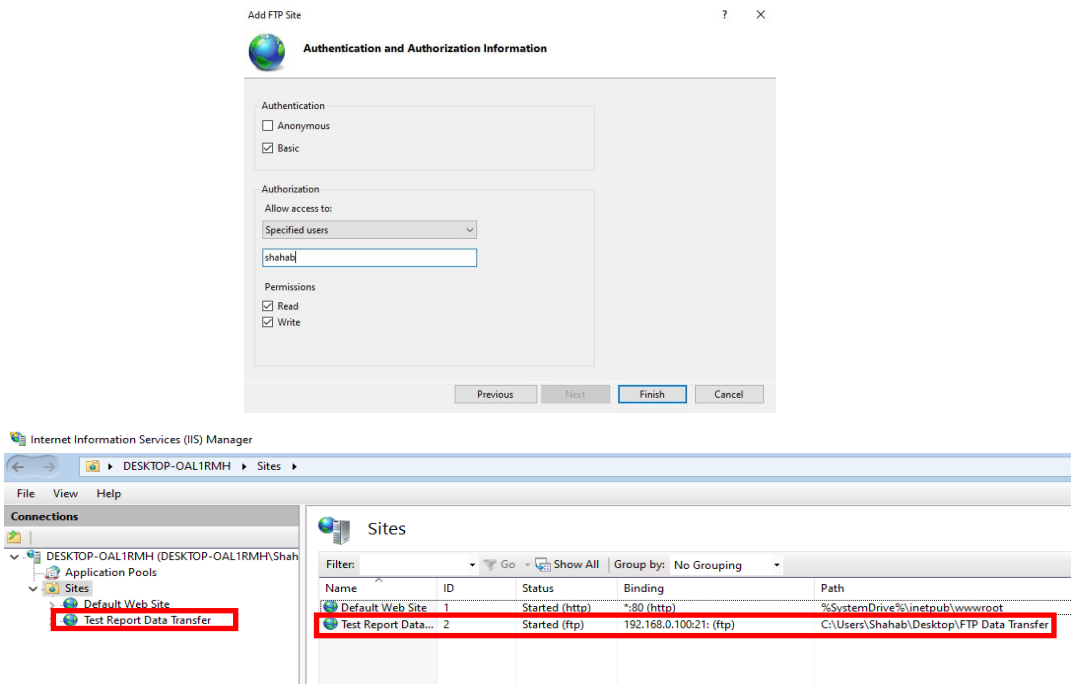


Figure 78. Adding username access and permissions in the FTP site

9.3 Adding users to FTP Protocol

To add more users to the FTP site server, the users need to be created in Windows settings, in Family & other users.

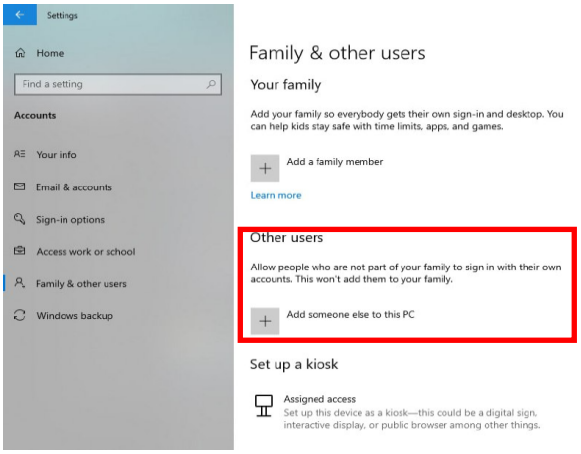


Figure 79. Adding users in FTP site

Username “VA1234” is created as a local account.

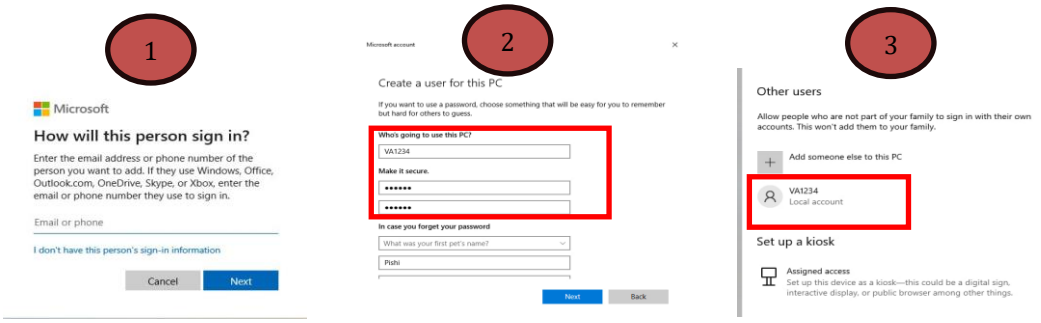


Figure 80. Creating an user to be used in FTP site

Returning to the IIS Manager, in the FTP Authorization Rules, permissions for the existing site “Test Report Data Transfer” can be granted.

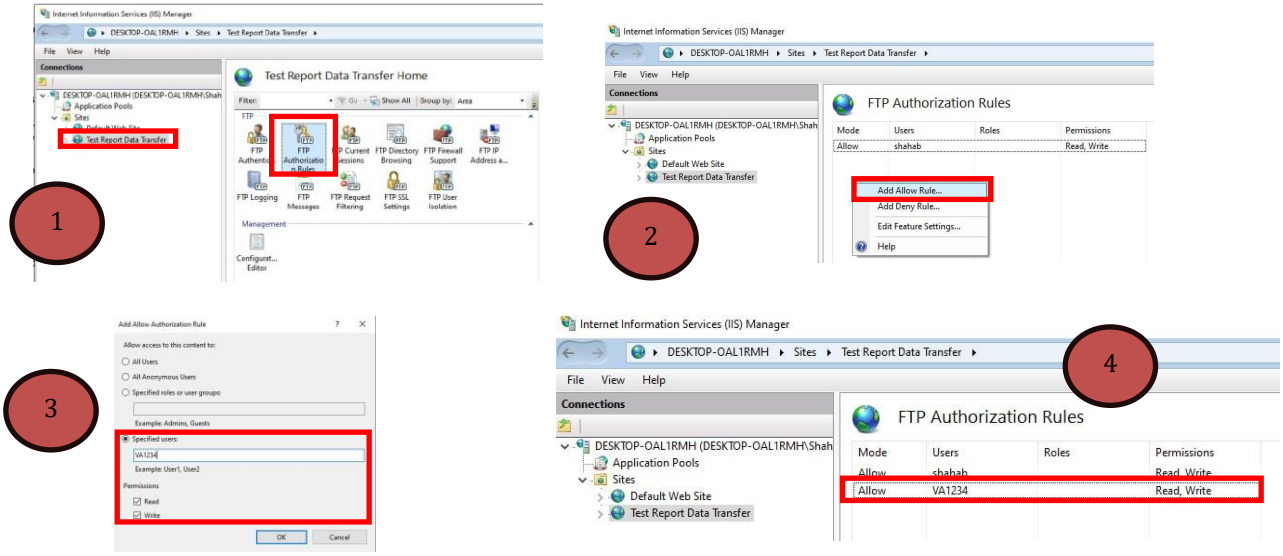


Figure 81. Granting rule and user in IIS manager window

On the selected physical FTP folder address, right-click and choose “Properties”, then under the Security tab, add the new user “VA1234” and grant them “Full Control” permissions for FTP transfer.

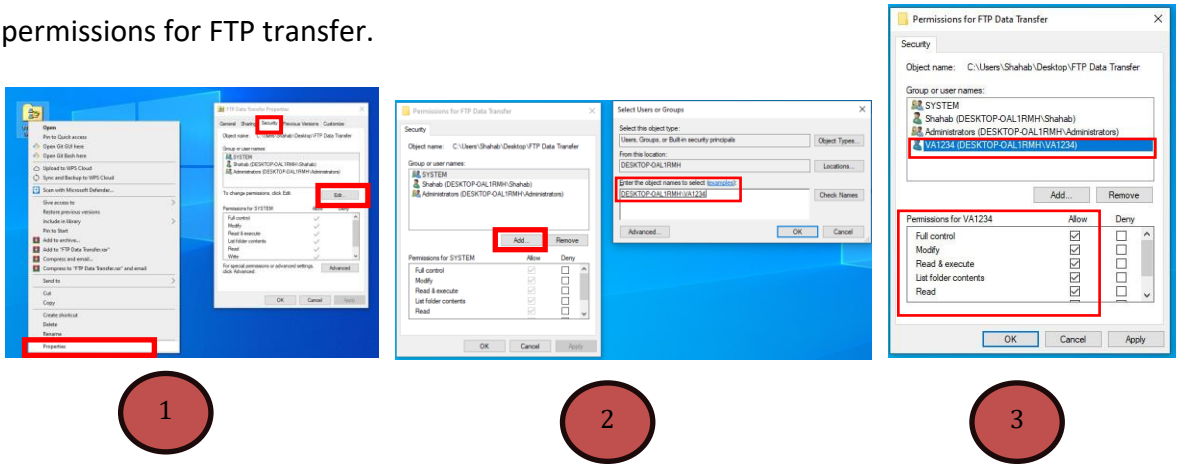


Figure 82. Adding user and allowing permission for FTP site

9.4 Verifying FTP configuration

To test the FTP transfer, three files and one folder were added to the FTP physical address on the local host [name: "Shahab"]. On the remote computer [name: "Shaha"], an FTP connection was established using the Command Prompt by entering the IP address, username, and password. As a result, the available folder and files in the local host directory are visible in the remote Command Prompt.

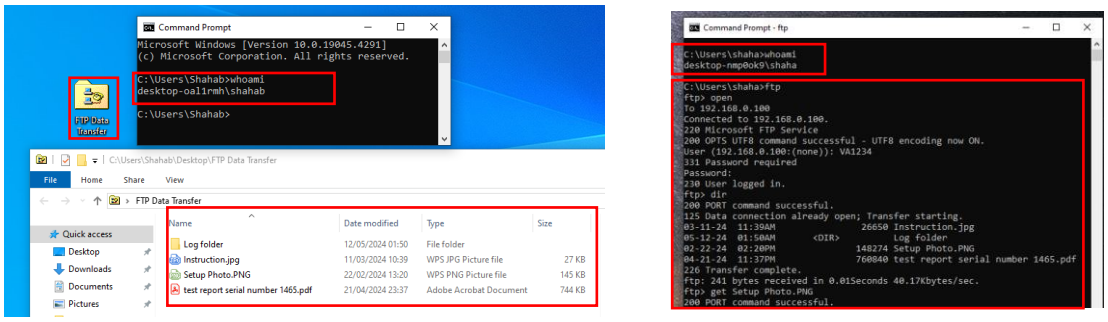


Figure 83. Verifying FTP site with sharing files and a folder

9.5 VPN restriction in FTP application

VPN should be implemented at all industrial levels to protect against cybersecurity attacks. To restrict FTP site access to a limited set of IPs, select “FTP IP Address and Domain Restrictions” in the IIS environment and impose the VPN IP address range.

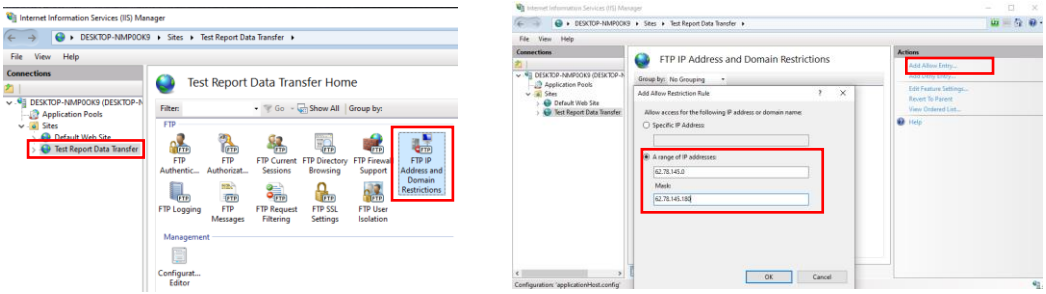


Figure 84. Adding VPN IP domain into FTP site rules

9.6 Introducing appropriate platforms for FTP protocol

There are various platforms available for transferring data under the FTP protocol license. Reference [55] introduces "WinSCP", a free platform for working within an FTP environment. As shown in Figure 85, the WinSCP platform installed on a remote computer illustrates how easily file transfers can be accomplished. The left side of WinSCP displays the storage directory of the remote computer, while the right side shows the connected host's storage directory. Files can be conveniently transferred between these computers through the FTP protocol using a simple drag-and-drop.

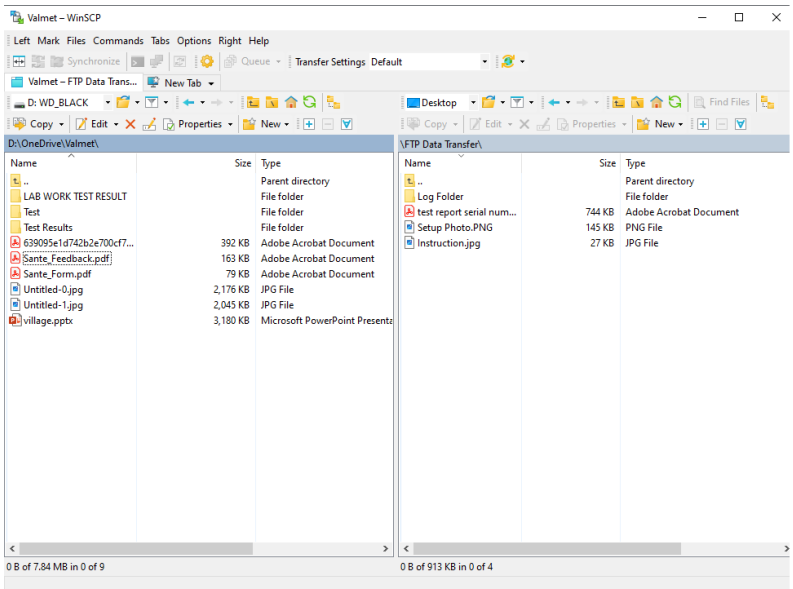


Figure 85. Intuitive application for FTP Protocol in Windows operating system

10 Proposed Method Implementation

In this part, the research approach is examined, including whether it is a conventional method or not, as well as its distinctive or innovative aspects. The methodology for automating test setup in this thesis is built on four key characteristics, shown in Figure 86. As the thesis title suggests, optimizing automation efficiency was the primary focus. This streamlined approach reduces maintenance and troubleshooting time while also being cost-effective—a crucial feature. Since this automated test setup is designed for short-term use (specifically for prototyping battery packs), affordability is essential. To enhance cost efficiency, it is recommended to utilize existing hardware and software resources (i.e., leveraging available capabilities) within the company (IONCOR Oy). Additionally, adopting an open-source framework for this automation setup is advised, enabling seamless expansion of features without restrictions.

Figure 86. four key characteristics for establishing an automated test setup for a prototype battery pack

To demonstrate how the four key traits are incorporated into the proposed automation test setup, a broader discussion of battery testing is required. While the battery testing and manufacturing aspects addressed in this section may initially appear unrelated, they are found to provide essential context for understanding the four key traits illustrated in Figure 86. The rationale behind the adopted methodology is clarified through an examination of battery testing approaches, particularly with regard to the communication

control selection, Azure storage configuration, and remote access monitoring. At the conclusion of this section, after all components of the system overview presented in Figure 87 have been explained, the applied methodology is justified based on the four key features displayed in Figure 86.

To properly select controls and develop programming scripts for communication with CMC boards in battery packs or a BMS unit functioning, a more thorough understanding of the battery design process, both in software and electrical aspects, is required. Figure 87 outlines four key phases of battery design: PCB design, embedded system development, and SIL/HIL testing using both simulated, dummy, and real battery cells. If any hardware modifications are made during prototyping, testing must first be conducted with dummy cells. It is important to note that lithium-ion battery cells can be hazardous, posing a risk of explosion if subjected to physical damage or excessive heat.

As shown in Figure 87, the electrical aspects of battery design involve components such as the CMC boards (covered in Subsection 3.3), the BMS unit (discussed in Subsection 1.6), and additional supporting elements like contactors, flexible PCB boards, pyro fuses, FET switches, DC links (bus bar), and noise-filtering boards. In the next chapter, more information is given further for these auxiliary parts.

The document later clarifies the distinction between prototyping facilities and high-volume production plants for battery manufacturing. This comparative analysis highlights key differences in testing methodologies between these two production approaches. As introduced earlier, during battery development phases, prototypes are exclusively produced in specialized prototyping facilities. Only after finalizing the product design does manufacturing transition to rapid serial production facilities for mass manufacturing of the mature battery design.

As noted in Subsection 1.4.5, cell architecture significantly influences the physical configuration of battery modules and packs. From an electrical design perspective, only the signals transmitted through cells to CMC boards require consideration. While Subsection 1.3 explains how varying cell chemistries alter operational voltage ranges, the software framework incorporates standardized logical range values that accommodate all chemistry types. Consequently, CMC boards typically maintain compatibility across different cell architectures and chemistries. Exceptions to this universal compatibility will

be detailed in Subsection 10.1.1 of the Texas Instruments (TX) CMC board documentation, which specifies the supported series cell configurations and permissible chemistry types for various TX microcontroller models.

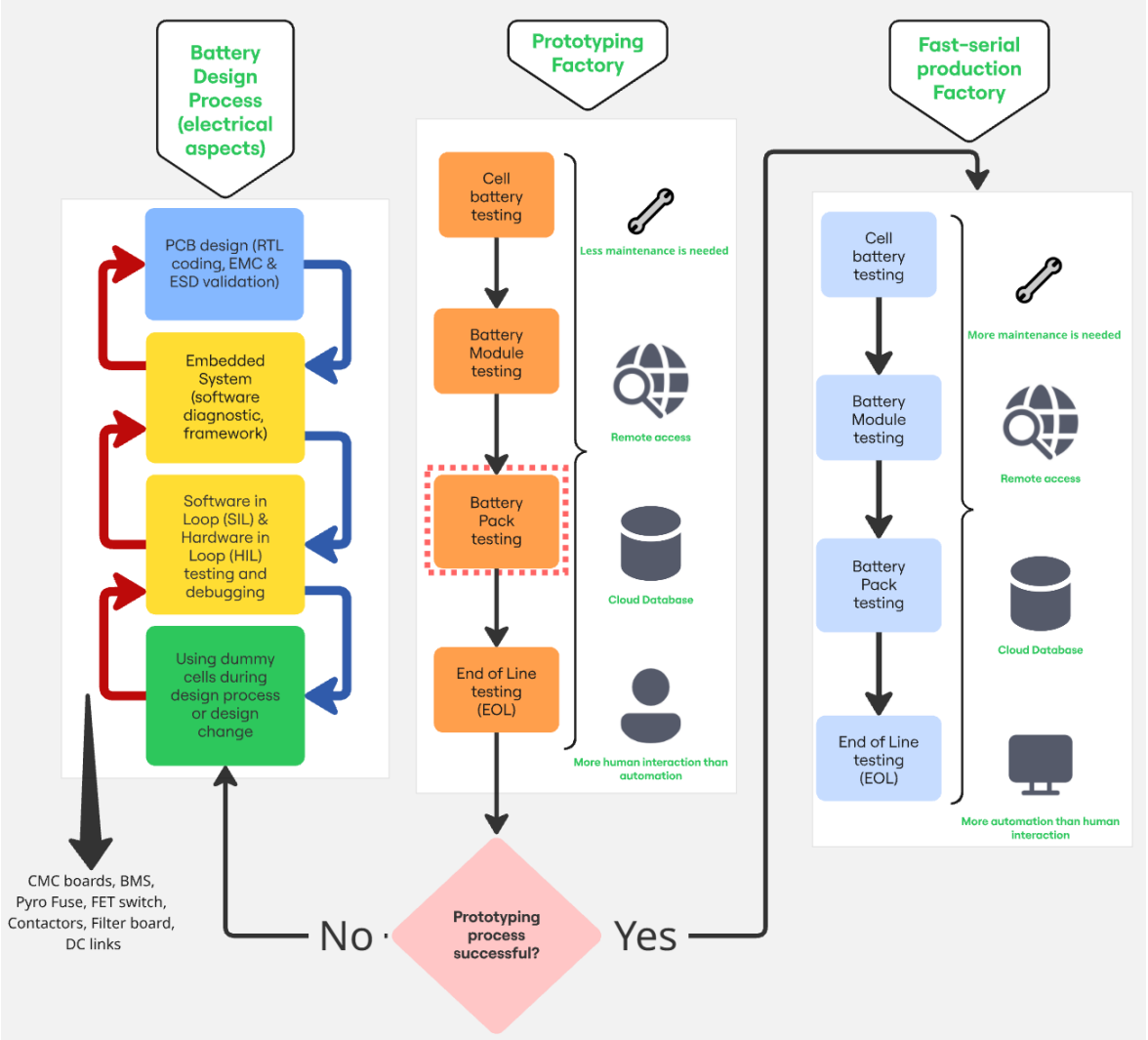


Figure 87. Overview of the Battery Design, Development, and Mass Production Process

10.1 Design process

During design process, an automotive battery technology undergoes multiple tests which the famous ones are such RTL coding, EMC and ESD testing while PCB design, software diagnostics, software-in-loop (SIL) testing and hardware-in-loop (HIL) against software or dummy or real of battery cells or battery pack.

The automotive battery development process involves rigid testing protocols, including several critical validation stages. Key tests include RTL (Register Transfer Level) coding verification, electromagnetic compatibility (EMC) and electrostatic discharge (ESD)

assessments during PCB development, software framework, software diagnostic evaluations, and comprehensive SIL (Software-in-the-Loop) testing using software simulated battery cells and battery packs. Additionally, HIL (Hardware-in-the-Loop) testing is conducted using various configurations - either with prototype (dummy) battery cells, or actual battery packs to validate system performance under real battery cells and battery packs.

10.1.1 PCB design

Battery manufacturers face a critical decision between utilizing commercially available OEM (Original Equipment Manufacturer) CMC boards or developing custom solutions from scratch. While OEM automotive brands are typically equipped with pre-built CMC boards, higher costs are often incurred due to their premium pricing. Based on this choice, the PCB design process is significantly affected and split into two distinct scanrios.

Scanario 1:

In Scenario 1, where non-OEM CMC boards are to be custom-designed from scratch, an extensive development process must be undertaken. This requires programming Integrated Circuits (ICs) at the hardware level using Register Transfer Level (RTL) coding through hardware description languages like Verilog or VHDL. As detailed in Reference [56], these languages are essential for such implementations. The ICs in question specifically refer to high-speed programmable logic devices such as Field-Programmable Gate Arrays (FPGAs)

RTL coding:

Field-Programmable Gate Arrays (FPGAs) are reconfigurable integrated circuits composed of numerous logic gates, flip-flops, multiplexers, and memory blocks [57]. Due to their low latency and real-time adaptability, FPGAs are widely adopted in telecommunications, automotive, and aerospace applications [58]. Unlike microcontrollers (MCUs, Section 3.3), which are optimized for specific functions, FPGAs are designed for broader, flexible tasks. In battery applications, CMC boards integrate FPGAs to manage input/output operations, on-chip memory, and communication protocol translation (e.g., UART to SPI). A key advantage of FPGAs over conventional processors is their ability to execute parallel

operations, eliminating resource contention between simultaneous processes [58]–[59]. FPGAs are configured using hardware description languages (HDLs) like Verilog or VHDL. When these languages describe digital logic at the register transfer level (RTL) for synthesis, the process is termed RTL coding.

PCB design and ESD verification:

Following the hardware description programming phase, attention shifts to PCB design for the CMC boards. The automotive industry mandates strict static-free design (SFD) compliance for all battery-embedded circuit boards. Personnel handling battery components at any level - cells, modules, or complete packs - must wear full ESD-protective clothing, including gloves, pants, and shirts, to eliminate static discharge risks. The predominant ESD standard for automotive applications, ISO 10605, governs all road vehicle electronics. This standard specifically addresses human-body discharge scenarios both inside and outside vehicles while specifying unique testing equipment requirements [60]. Complementary to this, electromagnetic compatibility (EMC) testing validates that the electronic systems operate reliably without generating or suffering from electromagnetic interference [61].

PCB design and EMC effect:

Electromagnetic Compatibility (EMC) plays a critical role in PCB design for automotive battery systems. As discussed in this thesis, certain serial communication protocols demonstrate varying levels of noise immunity. Section 1.5.3 illustrates how protocols like ISO-SPI and CAN can effectively reject both internal and external noise interference. Figure 88 demonstrates that CAN [62], RS-485 [63], and ISO-SPI [64] achieve this robustness through differential signaling. In contrast, SPI [65], I2C [66], and UART [67] show greater susceptibility to EMC effects due to their use of single-ended signaling (SPI, UART) and open-drain signaling (I2C). This inherent noise vulnerability explains why I2C is primarily employed as an on-board protocol for communication between chips on the same PCB rather than between separate boards.

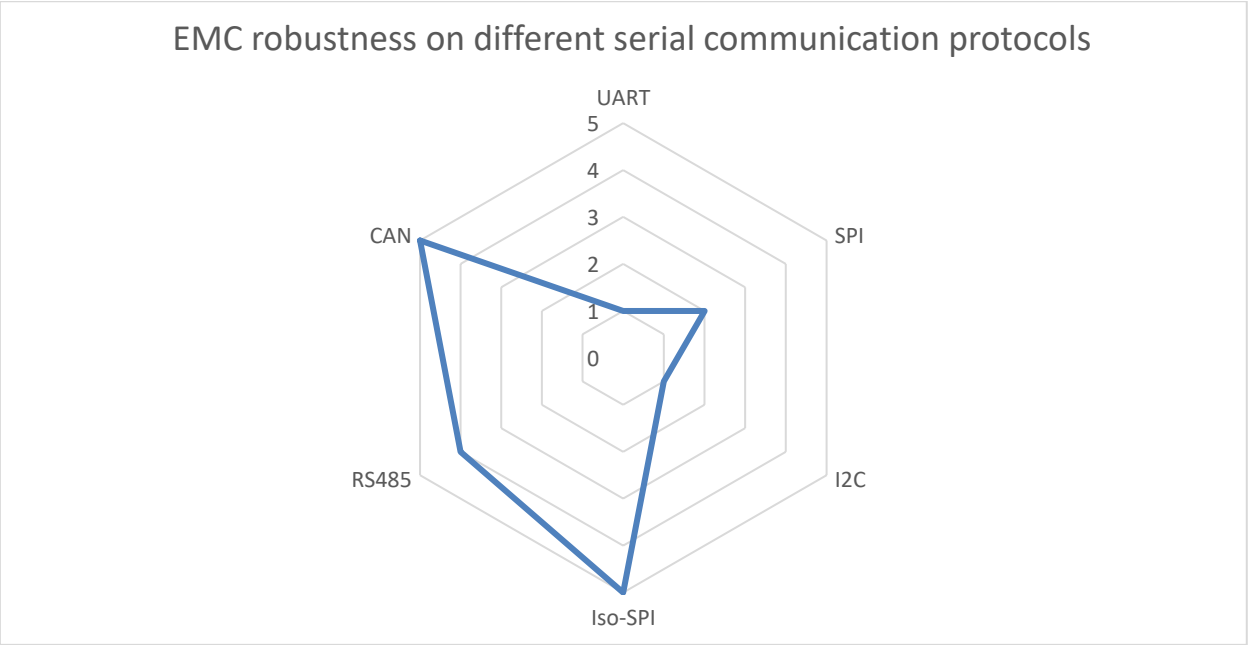


Figure 88. EMC robustness on different serial communication protocols

In automotive battery applications, flexible PCBs or FPCB (illustrated in Figure 89) are commonly employed. These flexible circuits enable reliable connections between battery cells, temperature sensors, and CMC boards. Given their extended length and pliable nature, noise-resistant communication protocols like ISO-SPI, RS-485, or CAN are typically implemented on these flexible PCBs. For the prototype battery pack developed in this thesis, RS-485 was selected (as detailed in Section 3.6) due to its differential signaling capability, which provides superior noise immunity. Also, this protocol via the flexible PCB facilitated communication between battery modules within the pack and with the central MCU controller.

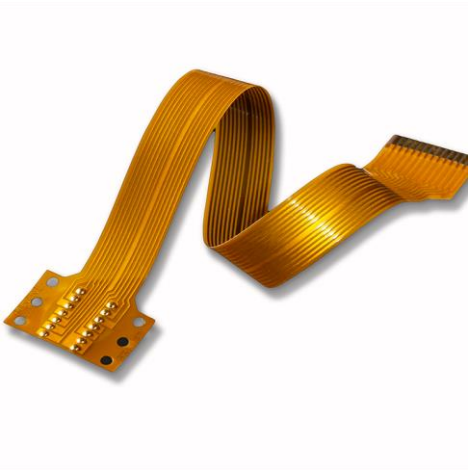


Figure 89. Appearance of flexible PCB or FPCB

Scenario 2:

In Scenario 2, where OEM CMC boards are implemented (as presented in Section 3.3 of this thesis), Texas Instruments solutions were selected for the battery system. Among available OEM CMC board manufacturers, Texas Instruments provides comprehensive battery management components, including battery monitors (CMC boards), chargers, and related products compatible with lithium-based and other battery chemistries [68]. Specifically, the stackable battery monitor models referenced in [69] - identical to those utilized in this thesis's battery pack (Section 3.3) demonstrate Texas Instruments' capability for scalable battery monitoring solutions.

The battery module configuration described in Section 3.3 consists of 19 cells arranged in two separate series strings - one with 10 cells and another with 9 cells. This specific arrangement indicates that the Texas Instruments battery monitor models suitable for this application (as referenced in [69]) would be the BQ79616-Q1, BQ79614-Q1, or BQ79612-Q1 series.

OEM CMC boards from Texas Instruments incorporate built-in protection against ESD and EMC interference. As a result, battery manufacturers utilizing these pre-certified solutions can eliminate the need for additional ESD/EMC compliance testing and avoid the requirement for RTL-level hardware programming.

10.1.2 Embedded Systems

Returning to the two scenarios outlined earlier, the firmware flashing process (detailed in Subsection 3.3.2) for CMC boards depends fundamentally on the chosen embedded software architecture. Following PCB design completion, the development focus shifts to embedded system implementation. Embedded developers typically employ C/C++ programming languages to create firmware that executes on the board's microcontroller, specifically tailored to meet the application's functional requirements.

The electrical industry employs diverse embedded system software architectures, but automotive battery applications predominantly utilize three key approaches: (1) Bare-Metal (Super Loop) Architecture, (2) Real-Time Operating System (RTOS) Based Architecture, and (3) Event-Driven Architecture. While CMC boards may implement any of

these architectures, Battery Management System (BMS) units in automotive applications typically adopt Event-Driven Architecture. This preference stems from the BMS's reliance on CAN protocol communication, which naturally aligns with event-driven paradigms for handling real-time battery data and system commands.

1) Bare-Metal (Super Loop) Architecture

Bare-Metal (Super Loop) Architecture is a method for programming embedded systems that operates directly on the hardware, bypassing the need for an OS or abstraction layers such as device drivers. It relies on a continuous loop (known as the "super loop" or "main loop") that cycles through all system tasks in a fixed sequence. Due to its straightforward and lightweight design, this approach is commonly employed in systems with limited resources, as it eliminates the complexity and overhead of an operating system [70].

Super-loop key characteristics [70]:

1) Infinite Loop: The super loop's central component is a never-ending cycle (similar to ”Void loop ()” function in Arduino IDE shown in Figure 90) that continuously performs operations such as gathering inputs, handling computations, and refreshing outputs. The ”void setup()” function in the Arduino IDE was a one-time initialization routine. It handled tasks like configuring pin modes (input/output), setting up timers, or defining initial states, all executed once before the ”void loop()” began its continuous cycle.

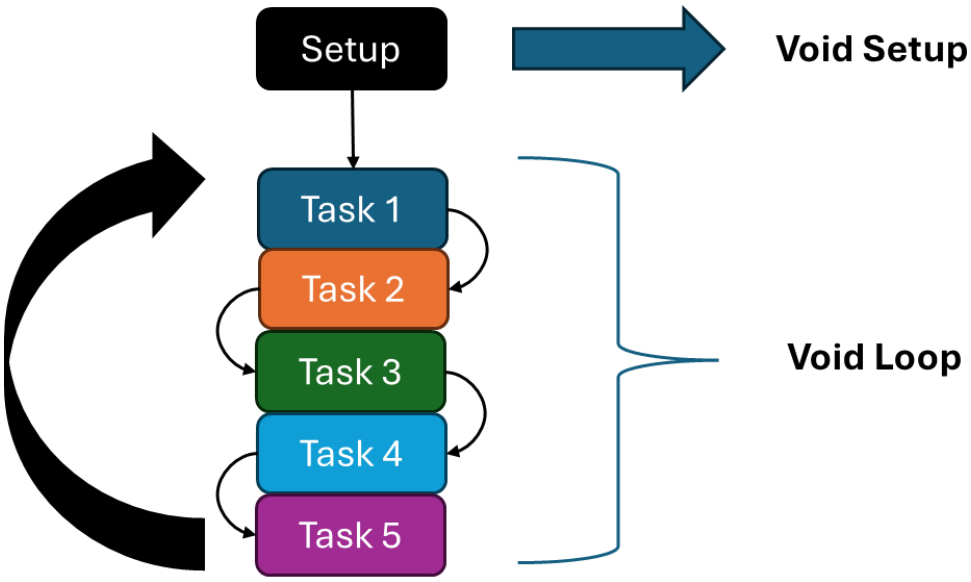


Figure 90. Super-Loop (bare-Metal) Architecture in embedded system

2) Sequential Execution: Tasks are executed one after another, with interrupts providing limited preemption for time-critical events (refer Figure 90)

3) No Operating System: The software interacts with hardware registers at a low level, handling system resources without relying on an operating system or kernel.

All programs developed for Arduino boards in this thesis follow the Super Loop Architecture. For instance, in “Appendix I”, tasks are executed sequentially as shown in Figure 90. For example:

- Task 1: Waits for input
- Task 2: Prints the input value into the serial monitoring
- Task 3: Checks if the input value is valid, unless it gets ignored
- Task 4: Processes the valid input (either “1”, “2”, “3”, or “4”) to the respective pin location defined in “Void Setup()”
- Task 5: Activates or deactivates the relay based on the received input value and then returns to Task 1

For this thesis, the controller used to communicate with the battery modules is the Arduino Mega. The Arduino Mega features 8KB of SRAM (Static Random-Access Memory), which is used for temporary data storage during program execution. Additionally, it includes 256KB of flash memory for storing the program code.

2) Real-Time Operating system (RTOS) Architecture

A Real-Time Operating System (RTOS) is a purpose-built OS tailored for embedded systems and applications that require strict timing control and deterministic performance. Unlike general-purpose OSs (such as Windows or Linux), an RTOS prioritizes minimal latency and reliable task scheduling [71]. It employs various operating systems to efficiently handle task management, scheduling, and resource allocation. It utilizes multiple operating systems to effectively manage task execution, scheduling, and resource distribution. Some of the most widely used RTOS platforms include [72]:

- FreeRTOS & Zephyr (open-source) for general purposes
- VxWorks (commercial), commonly found in industrial PLCs from brands like Allen-Bradley, Siemens, and Schneider Electric
- QNX (commercial) is mainly used for automotive industry
- Along with various other specialized real-time operating systems.

RTOS architecture is employed across various critical applications, including medical devices, robotics, telecommunications, automotive ECUs (Electronic Control Units), drones, and military systems [73].

In FreeRTOS, tasks run in a continuous loop, executing concurrently with other tasks in parallel, as illustrated in Figure 91. This parallel task execution demands significant SRAM capacity from the microcontroller to handle multiple tasks simultaneously. For instance, the STM32F405xx and STM32F407xx microcontroller families (suited for RTOS) feature:

- High-speed embedded memories (up to 1 MB of Flash memory)
- Up to 192 KB of SRAM for real-time processing
- An additional 4 KB of backup SRAM for critical operations [74].

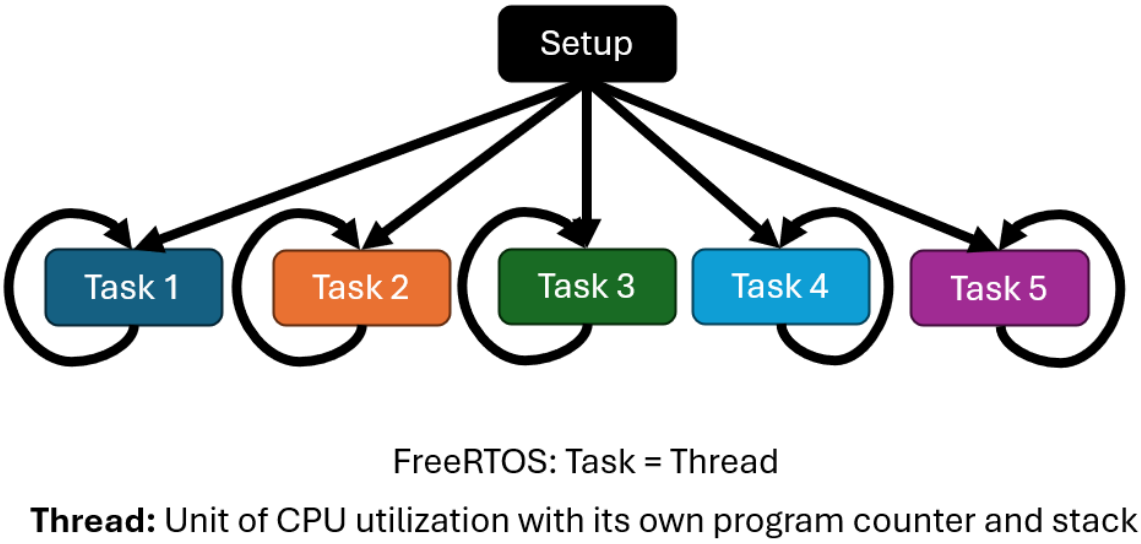


Figure 91. RTOS Architecture in embedded system

3) Event-driven Architecture

The automotive sector heavily relies on Event-Driven Architecture (EDA), a foundational framework for modern vehicle systems. As highlighted earlier, a prime example is the Battery Management System (BMS), which communicates via the CAN protocol—an inherently event-driven technology.

EDA's dominance extends far beyond automotive applications. From real-time multiplayer gaming and secure online banking to high-volume streaming platforms and cutting-edge generative AI, this architecture has become the backbone of digital innovation. In fact, industry reports reveal that over 72% of enterprises worldwide leverage EDA to drive their critical applications and operational workflows [75], solidifying its status as the preferred architectural paradigm for embedded and distributed systems.

The experimental battery pack developed for this research utilized Texas Instruments' CMC modules (including BQ79616-Q1, BQ79614-Q1, and BQ79612-Q1 variants), all programmed using Event-Driven Architecture principles. As illustrated in Figure 10, the test setup incorporated a National Instruments DAQ 6500 unit, which served as the communication interface with these TI CMC boards - all operating on the same EDA programming framework.

As outlined in Subsection 1.5.4, automotive ECUs primarily function as CAN nodes (with exceptions for protocols like FlexRay and LIN). These nodes prioritize CAN messages - higher-priority ECUs transmit first, while lower-priority messages experience controlled delays. This prioritization guarantees uninterrupted delivery of critical communications while maintaining strict network timing constraints.

ECUs typically trigger messages based on various events, including:

- Timer expiration
- Bus error detection
- Message reception/transmission requests

Event-driven processing forms the foundation of CAN software programming, with industry-standard tools like CANoe, CANalyzer, CANLab, SavvyCAN, CanKing, and ETAS all implementing this core mechanism.

Comparative Analysis of introduced Embedded System Architectures

Having presented three distinct embedded system architectures, their evaluation of respective strengths and limitations is shortly concluded in this section. As References [70] and [75] comprehensively document in Table 10, each architecture demonstrates unique characteristics. Table 10 particularly highlights that event-driven architectures may present debugging challenges depending on the application context. The subsequent subsection details specific methodologies to address these diagnostic difficulties

Table 10. Trade-off between introduced embedded system architectures

	Super-loop	RTOS	Event-driven
Advantages	1) Straightforward coding and troubleshooting process 2) No operating system requirement 3) Ideal for basic operations (particularly dedicated-function systems)	1) Enhanced concurrent processing with deterministic performance 2) Improved system organization and expandable architecture	1) Ensure real-time connectivity across critical applications and systems. 2) impossible to scale, accommodate modern business needs without EDA
Disadvantages	1) Challenges in scaling for intricate, multi-process environments 2) No native task hierarchy, reducing suitability for real-time scheduling demands	1) Increased resource consumption (memory and processing power) 2) Greater implementation complexity	1) Can increase the overall system complexity 2) Requires careful design and implementation in event ordering 3) Debugging and Troubleshooting **can be very challenging** to trace the flow and identify the root cause of problems

10.1.3 SIL & HIL

As previously noted, troubleshooting in Event-driven architectures or Real-Time Operating Systems (RTOS) can be challenging due to the complexity of the workflow. To address this, Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL) are critical methods employed in embedded systems, control systems, and automotive

development. These techniques help validate system safety, efficiency, and reliability before actual hardware deployment. SiL allows the simulation of embedded software on a host computer, eliminating the need for physical hardware. Meanwhile, HiL involves connecting the embedded controller to a simulated environment using real input/output (I/O) interfaces for more accurate testing.

Software-in-the-Loop (SiL):

In SIL testing, message timing and priority can be adjusted to simulate specific bus data traffic conditions. SIL offers several benefits that make it highly useful for automotive testing [76]:

- Cost Efficiency: Since no physical hardware is required, testing costs are minimal. The only major expense is licensing commercial software.
- Fast Execution: Simulations run under ideal conditions, enabling faster processing due to noiseless serial communication and immediate responses.
- Real-Time Limitation: Real-time constraints cannot be evaluated since the system operates in simulation mode.
- Easier Debugging: Faulty nodes can be isolated and removed during debugging, simplifying the process due to the modular design of the simulated software.
- Simulated Test Environment: All inputs and outputs are software-generated, such as scripted random data for voltage cells or temperature sensors, as demonstrated in this thesis with the battery pack simulation.

Hardware-in-the-Loop (HiL):

HIL testing enables the evaluation of high-risk or impractical real-world scenarios without road testing. For instance, simulating a faulty contactor in a battery BMS unit during HIL testing can hinder potential vehicle hazards. HIL offers several key benefits for automotive validation [77]:

- Cost Considerations: Testing expenses are higher due to the need for physical hardware. However, a hybrid approach combining HIL with SIL (partial simulation) can reduce costs in some cases.

- Execution Realism: Unlike simulations, HIL operates in real-time, exposing actual system delays and programming flaws that may affect performance.

- Real-Time Capabilities: HIL validates true real-time constraints, including maximum bus data capacity under actual operating conditions.

- Debugging Advantages: The inclusion of real-world factors (e.g., electrical noise) makes debugging more accurate, though more complex, compared to SIL.

- Test Environment Flexibility: HIL supports fully hardware-based testing or mixed hardware/software setups. For example, sensor data (voltage, temperature) comes from physical sensors rather than scripted simulations.

10.1.4 Dummy cells/dummy module

When creating new products, designers often use dummy or simulated batteries for testing instead of real ones. What kind of dummy battery they use depends on the type of test.

- Computer Models: Early tests that are purely software-based (SIL tests) use computer simulations of batteries.

- Physical Dummies: Later tests that involve actual hardware (HIL tests) use physical, non-working dummy batteries.

If a test with one of these dummy units fails, the engineers know they need to go back and improve the product's design.

Dummy implementations can simulate either entire battery cells or specific cell characteristics. In this research, for instance, battery insulation properties were effectively emulated using a 50 MΩ high-power resistor (Figure 92). Regarding the equipotential testing discussed in Section 5.6, typically this test suited when multiple battery packs are mechanically joined using torque deveice fasteners. Since this study

involved a single battery pack, a metal alloy with higher conductor resistance was employed to introduce a resistance of 3.5 mΩ for the TestStand report.

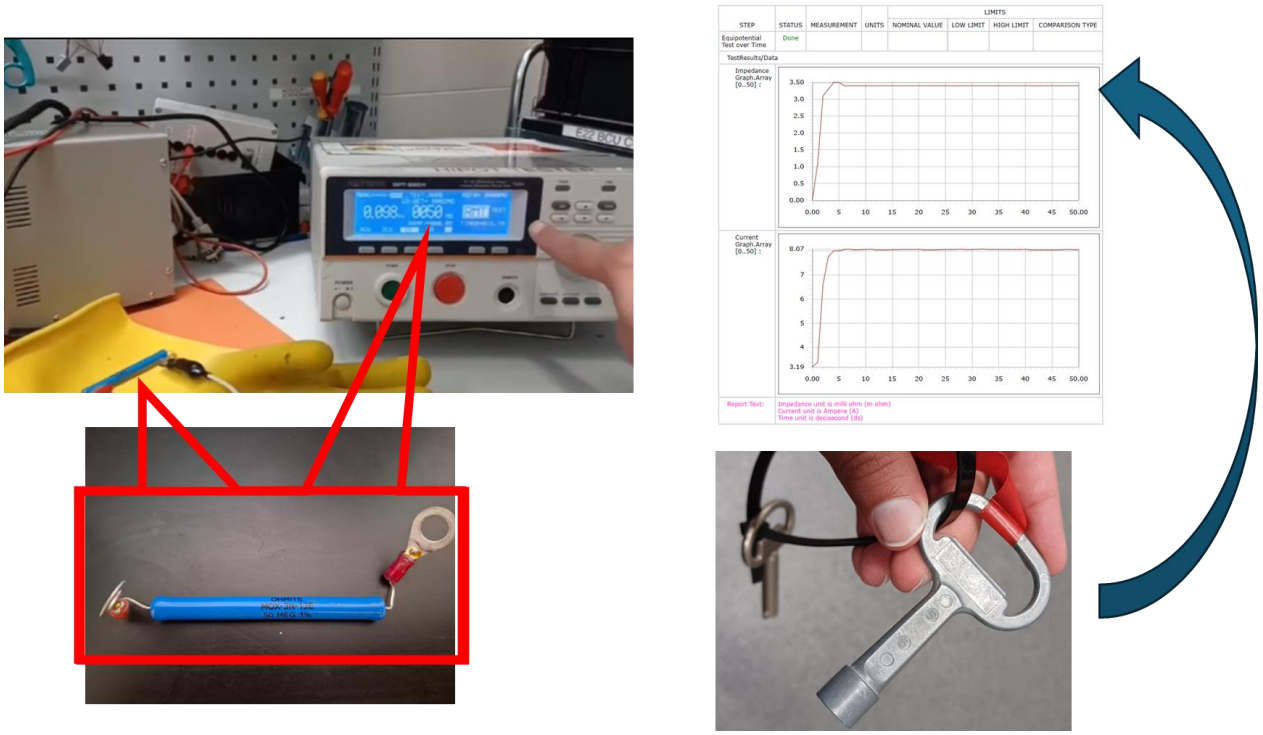


Figure 92. Used dummies in this thesis

Dummy batteries are utilized not only in the design and prototyping stages but also during the initial fast-serial production of the first product. It is safer to use these non-functional replicas to simulate the appearance and characteristics of real lithium batteries throughout the logistics, assembly, and testing processes. This is because real lithium batteries are explosive. Using dummy batteries prevents accidents that could occur if robots in the early stages of mass production were to accidentally damage an actual battery cell. For instance, Figure 93 illustrates a dummy battery with dummy cells being used to test the welding quality (by measuring conductor resistance) of the cell bus in a testing station. In this scenario, excessive pressure from the testing needles, as shown in the figure, could have severe consequences if applied to a real battery.

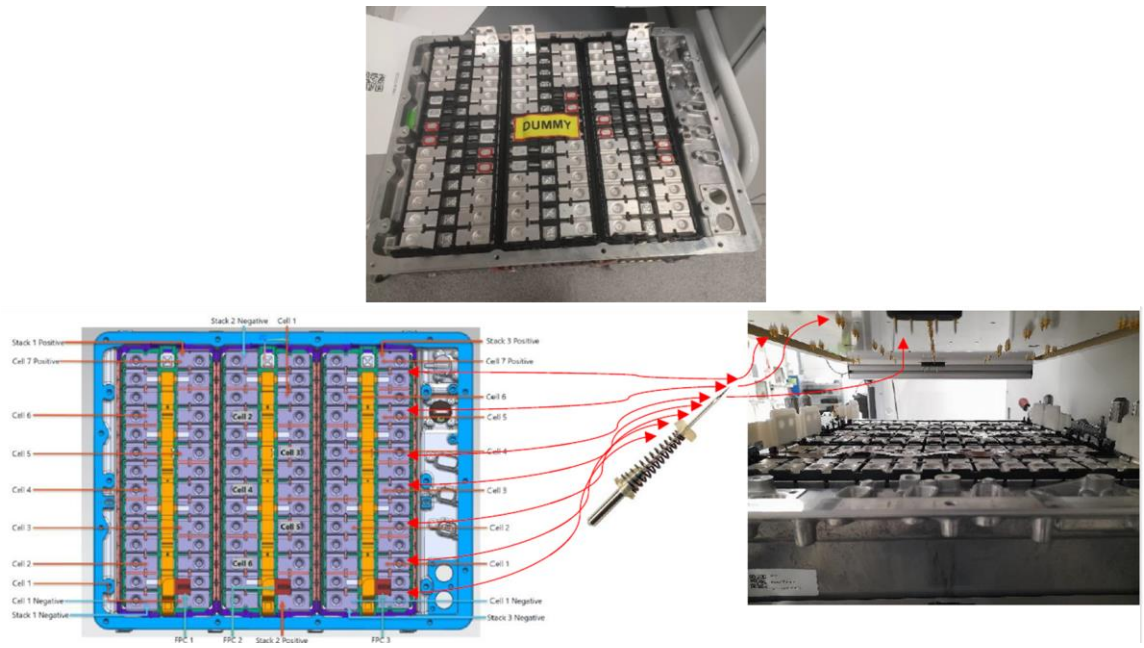


Figure 93. Dummy battery used in prototyping phase

10.2 Development Vs. mass production

To better understand battery manufacturing, it's helpful to compare prototyping workshops with mass production factories. This comparison will clarify the reasoning behind the embedded system architecture and controller choices made in this thesis.

The similarities between these two types of facilities can be outlined:

1) Dummy Batteries: As previously noted, both prototyping workshops and mass production factories use dummy batteries during their initial production phases for safety purposes.

2) ESD Clothing: In both types of factories, all staff members who handle battery components are required to wear clothing that prevents electrostatic discharge (ESD). These garments are typically marked with the sticker shown in Figure 94.

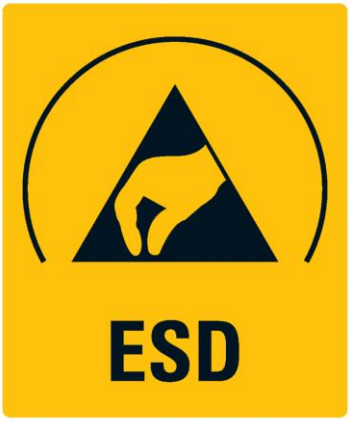


Figure 94. ESD sticker marked on clothing and electric device

3) ESD Flooring: To avoid transferring an electric shock to the batteries, the floors in both factories are designed to be ESD-free [78].

4) Testing process: According to Figure 87, the fundamental battery testing process is consistent in both factories, although the specific methodologies employed may vary.

The distinctions between a prototyping factory and a fast-serial production factory are outlined as follows:

1) Level of Automation and Workforce Size: In a prototyping factory, human involvement outweighs automation. The focus is on manual labor for assembling and testing batteries, with operators using jacks and lifters to move batteries, and no automated conveyor belts are utilized. In contrast, a fast-serial production factory prioritizes automation over human interaction, employing PLCs, sensors, HMIs, and autonomous conveyor belts to transport battery modules, reducing the need for human effort in assembly lines. Figure 95, sourced from IONCOR's website, depicts the settings of both a prototyping workshop and a fast-serial production facility. The HMIs displayed in Figure 95 for the fast-serial production line highlight the critical role of automation in the process.

Figure 95. Mass production (fast-serial line) factory [left] and prototyping workshop [right]

2) Maintenance: Prototyping factories require minimal maintenance because of their smaller-scale production and limited machinery. In contrast, fast-serial production factories need regular, scheduled maintenance due to their high-speed, machine-intensive production processes.

3) AI, IoT, and Sensors: Prototyping factories lack AI and IoT features to reduce costs. Conversely, fast-serial production factories utilize advanced sensors, such as gyroscopes, accelerometers, and magnetometers, integrated with AI and IoT capabilities. Notably, fast-serial factories commonly employ RTOS (for IoT and PLCs) and event-driven architectures (for battery-controller interactions), while prototyping workshops can use simpler architectures, such super-loop for their embedded system needs, which are rarely seen in fast-serial lines.

4) Azure Cloud: As discussed in Subsection 7.2.2, Hot and Cool tiers were described. In prototyping workshops, Hot tiers are more practical and cost-effective due to frequent data access needs. In fast-serial factories, however, cloud data is less frequently reviewed because of routine operations, making Cool tiers more suitable and cost-effective for these facilities. In a fast-serial factory, disruptions to the routine caused by issues lead to increased data review, though such occurrences are infrequent given the high production rate.

As mentioned earlier, the testing processes in both battery factories are comparable, with variations mainly due to automation. Below, the standard tests for each stage are briefly outlined, along with potential differences in methodologies. Note that not all stages can occur within a single factory. For instance, cell testing might take place in factory “A,” while factory “B” focuses on cell stacking and assembly up to the battery pack, and factory “C” serves as a dedicated hub for EOL (End of Line) testing. In another scenario,

factory “A” is responsible for all manufacturing process. Depending on the battery model, some tests described below may not be required, or additional tests may be necessary.

10.2.1 Cell battery testing

Irrespective of the cell architecture discussed in Section 1.4, certain tests are strongly recommended for battery cell testing. These include:

- Cell Voltage Measurement: Measuring the voltage at the cell terminals.
- Cell Voltage Comparison: Comparing the measured voltage with the original voltage specified by the manufacturer, typically indicated on a QR code along with the manufacturing date. At this stage, the testing station must account for differences between the measured and original voltage, as lithium battery cells exhibit significant self-discharge in the initial days [79]. If the self-discharge rate is too high, the cell will not be used and should be recycled.
- Cell Internal Resistance: Measuring the cell’s internal resistance using AC or DC voltage to assess its health.

Since this phase focuses solely on testing individual cell characteristics, without involving CMC communication, the testing methodology is generally consistent across both fast-serial and prototyping factories.

10.2.2 Battery module testing

At this stage, the CMC board is typically assembled with multiple stacked battery cells. The controller interfacing with the CMC board may use super-loop, EDA, or RTOS architecture, depending on the battery design.

The tests performed at this level are:

• Isolation Resistance Measurement: Follows the same principles outlined in Subsection 5.8.

• Dielectric Strength Measurement: Follows the same principles outlined in Subsection 5.9.

• Cell Voltage/Temperature Validation: Follows the same principles outlined in Subsection 5.4.

• Cell Voltage Balancing: Performed if necessary.

In prototyping workshops, the simplicity of the super-loop architecture makes it suitable for the battery controller at this stage. However, the choice of output serial protocols is critical. If the CMC board uses common protocols like UART, SPI, ISO-SPI, or RS485, which are “control-based driven,” the super-loop architecture is feasible. However, when the CAN protocol is used, super-loop becomes less practical due to CAN’s inherent “controller-independent” nature. Typically, CMC boards do not use the CAN protocol for output serial communication, but it is commonly employed in the BMS unit of the battery.

10.2.3 Battery Pack testing

The controller interfacing with the CMC board may utilize super-loop, EDA, or RTOS architecture, depending on the battery design, as previously mentioned.

The tests performed at this level are:

• Equipotential Testing: Follows the same principles outlined in Subsection 5.6 and may be conducted during battery EOL testing, depending on the battery design.

• Isolation Resistance Measurement: Follows the same principles outlined in Subsection 5.8.

• Dielectric Strength Measurement: Follows the same principles outlined in Subsection 5.9.

• Cell Voltage/Temperature Measurement: Follows the same principles outlined in Subsection 5.4.

• EEPROM Writing: Follows the same principles outlined in Subsection 5.5 and may occur during battery module or EOL testing, depending on the battery design.

• DCIR Test: Follows the same principles outlined in Subsection 2.1.

This thesis proposes an automated test setup using super-loop architecture for this stage. Since the Texas Instruments CMC boards used RS485 Modbus for communication, super-loop was feasible provided all registry addresses for communication were available. To maintain confidentiality of the prototyping battery, the battery pack was simulated in the prototyping factory, and instead of using RS485 Modbus registry addresses, the author defined custom UART registry addresses. In contrast, fast-serial factories prioritize efficiency over simplicity, making EDA and RTOS architectures more suitable for this stage.

10.2.4 End of Line testing (EOL)

At this stage, with the BMS unit integrated into the complete battery system, the controller communicates with the battery solely using an EDA architecture. This stage is termed End of Line (EOL) because it represents the final step in the assembly process, where the BMS unit is typically the last component installed on the battery system.

At this stage, the test sequence is more detailed, in-depth, and time-consuming. The testing methodology for EOL testing is consistent between prototyping workshops and fast-serial factories. Simplicity in EOL testing is not advisable, as data communication is closely monitored and recorded during the tests. The BMS unit must remain in active mode and requires an EDA message format, typically using the CAN protocol, which operates on the CAN bus without needing controller polling. Without the CAN bus's inherent "controller-independent" nature, the BMS unit would remain in standby mode but not active mode. This characteristic allows the BMS unit to transmit time-based messages independently of any controller on the CAN bus.

The tests performed at this level are:

• Equipotential Testing: Follows the same principles as described in Subsection 5.6 and may be performed during battery EOL testing, depending on the battery design.

• Isolation Resistance Measurement: Follows the same principles as outlined in Subsection 5.8.

• Dielectric Strength Measurement: Follows the same principles as outlined in Subsection 5.9.

• EEPROM Writing: Conducted if not performed in the previous stage.

• DCIR Test: Follows the same principles as described in Subsection 2.1.

• Tightness Testing: Ensures the battery enclosure is properly sealed to prevent leaks in cooling pipes.

• Software Diagnostic: Requires an EDA-based controller to interface with the BMS unit, recording data such as bootloader version, general firmware details, interlock status, active or passive BMS errors, contactor status for isolation and dielectric tests, pilot line status, pyro fuse status, cooling system status, temperature sensor data across the battery, cell voltages, and cell balancing if needed, among other parameters.

• Capacity Test: Measures the battery's energy storage and delivery capacity, expressed in Ampere-hours (Ah) or milliampere-hours (mAh), indicating how many amps the battery can deliver over a specific duration. The EDA controller actively monitors battery conditions during this test.

The test setup proposed in this thesis is not suitable for this battery assembly stage, as the super-loop architecture discussed in Subsection 10.2.2 is not easily compatible with the EDA format. A solution was proposed in Subsection 11.4.3, recommending an EDA-based controller for interaction with the battery's BMS unit.

10.3 Defending applied methodology

This section outlines the rationale behind the chosen methodology. As shown in Figure 86, four main criteria were considered: 1) Technically streamlined, 2) Utilization of existing resources, 3) Cost-effectiveness, 4) Broad adaptability and open-source compatibility. In addition to these factors, safety also played a crucial role. For instance, any controller interfacing with a battery pack should be protected against electrostatic discharge (ESD). Popular microcontrollers and microprocessors like the RP2040, nRF52840, and Raspberry Pi 4 are known to be sensitive to electronic shocks. However, newer versions—such as the Raspberry Pi 5—have addressed this vulnerability.

Subsections 10.1 and 10.2 provided insight into how these four key traits were implemented in the automation setup described in this thesis.

1) Technically streamlined:

The use of a controller like Arduino significantly simplified system development for the developer. As shown in Figure 10, the original testing station employed a hybrid controller setup incorporating both Texas Instruments and National Instruments hardware. This combination complicated programming and debugging processes, as LabVIEW was the only available programming language. Debugging became especially challenging due to the integration of two different hardware platforms. The debugging sometimes can demand more than one software specialist .

Given that the prototyping phase is supposed to be a short-term and involved only a few battery samples, simplicity is crucial. The automation test setup developed in this thesis was based on a super-loop architecture, which proved ideal for testing battery modules and packs. Compared to the event-driven architecture used in the original test station (Figure 10), the super-loop structure greatly simplified debugging. In the LabVIEW-based system, extra complexity arose from the need to filter peripheral data caused by the periodic polling of the Texas Instruments MCU. These filtering functions added unnecessary difficulty to the LabVIEW programming process.

This thesis focused solely on battery packs without an integrated Battery Management System (BMS), making the super-loop architecture suitable for testing battery packs or battery modules, and for the setup shown in Figure 87. However, for End-of-Line (EOL) testing, where hardware relies on event-driven communication, the super-loop architecture is not practical. Therefore, as described in Subsection 11.4.3, a new hardware upgrade is proposed to support an event-driven architecture for EOL applications.

2) Utilization existing resources

LabVIEW and TestStand were used in this thesis to control the safety tester device. Since IONCOR already had licenses for both software tools, they were a practical and convenient choice for this thesis.

Additionally, because Microsoft Azure is the cloud platform currently used at IONCOR, it was selected for this thesis as well—rather than alternatives like Amazon Web Services or Google Cloud.

3) Economically affordable

Arduino controllers are more affordable and widely accessible compared to the Texas Instruments and National Instruments controllers used in the original testing station.

In Microsoft Azure, SQL Server is more costly than File Share. However, during the prototyping phase, where the volume of data stored in the cloud is relatively low compared to high-speed production environments, File Share offers a more cost-effective solution. This is the reason File Share was selected for this thesis. In contrast, for fast-paced industrial settings where data has higher value and needs to be well-organized, a structured database like SQL Server is the more appropriate choice.

4) Widely adaptive and open source

The upcoming Subsections 11.4.1, 11.4.2, and 11.4.3 present additional features that enhance the automation testing setup developed in this thesis. These hardware upgrades are feasible only if the controllers used are truly open source.

11 Discussion

The semi-automated test bench is tailored for a low-budget prototyping factory. This setup allows for gradual battery upgrades, prioritizing quality over rapid production. Consequently, the semi-automated test bench is suitable for this application. However, it requires human intervention to move the battery pack to the test bench, connect it to the safety measuring device, and have a testing operator scan the battery pack's QR barcode.

In prototyping factories, unlike high-speed serial factories, most projects are developed within a short, temporary timeframe. This eliminates the need for expensive automated robots for production. As a result, a low-budget electrical setup is required due to the brief development period, and no PLC programming was implemented to fully automate the testing process.

This section outlines the challenges encountered, details the tools utilized in this thesis, offers recommendations, and proposes future improvements.

11.1 Thesis Obstacles

The main challenges in this thesis were the company's confidentiality policies and financial constraints. As a result, detailed information about the actual prototyping battery pack could not be shared when discussing communication signals with the corresponding controller. This made the author simulate the battery pack for serial communication with its controller.

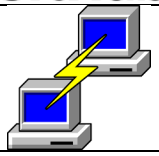
The need for an affordable solution led the author to create a simulated battery circuit with a corresponding controller. To keep costs low, all simulated boards were Arduino boards. Financial constraints also influenced the choice of communication protocol in the simulation. While the real battery uses the Modbus (RS485) protocol, the simulated battery pack utilized UART protocol for communication. Additionally, the inexpensive NO relays used were not suitable for high voltage testing, which is why lower voltage was applied in the isolation and dielectric tests in this thesis.

As noted in Subsection 3.7, Arduino hardware has a limitation of sending data in the UART protocol with an 80-byte maximum. To deal with this limitation for the simulated battery pack's voltage cell data (which exceeds 80 bytes), two registry addresses, "1" for data information from cell 1 to cell 10 and "2" for data information from cell 11 to cell 19 were embedded.

11.2 Applied Tools

This thesis utilizes hardware, software, and repository sources to develop the proposed test bench and battery simulation. The signal figures for serial communication protocols shown in Section 1 were analyzed using the Kingst logic analyzer (LA1010), which, as indicated in Table 11, supports various serial protocols. GitHub repositories provided valuable information for battery simulation and programming SPI, UART, and I2C protocols. Additionally, other software tools like Putty and Realterm were employed to manipulate data in USB ports while working with LabVIEW VIs. Com0Com was used to create virtual USB ports on a PC, facilitating easier USB port communication within the LabVIEW environment.

Table 11. List of applied tools in this thesis

N.	Name	Application	icon
1	Logic analyzer	This hardware is able to read data from different protocols such: SPI, I2C, UART, CAN, CAN FD, Ethernet	
2	GitHub	Learning from other's experience [GitHub's repositories]	
3	Putty	is a free and open-source terminal emulator, serial console. Useful for serial communication programming	
4	Realterm	open-source emulator for serial Ports, USB Serial and TCP/IP & Telnet, I2C Bus, SPI& 1-Wire chip control via BL233B / I2C2PC	
5	Com0Com	Free and open-source application for creating virtual COM port in PC. Very useful for serial communication programming in LabVIEW	

11.3 Recommendations

This section outlines some **potential minor improvements** for the proposed test bench that can be implemented **without** requiring additional financial support:

11.3.1 Adding Error Detection Methods

In Subsection 3.6.1, CRC and parity error detection methods were discussed. However, this thesis did not implement any error detection. It is important to note that if either of these methods is applied in the simulated battery pack, the corresponding LabVIEW VI must be configured to interpret the chosen error detection method.

11.3.2 Battery Pack Capacitance Behavior Calculation

Figure 50 highlights three factors impacting the results during the isolation test. While leakage current can be measured in the dielectric test, the capacitance charging current

can be calculated if the isolation test is extended for a longer period, allowing the isolation resistance to reach its steady-state value.

In this case, the capacitance C can be calculated using the time constant τ (tau) in an RC circuit. R is the isolation resistance measurement. The time constant τ is defined as the time it takes for the voltage across the capacitor to reach approximately 63.2% of its final value during charging. In Appendix J, the isolation test result waveform is available and can be utilized to determine τ (tau) if the test duration is extended further. For example, 20 seconds more, the isolation resistance would be in a steady state. The relationship between the capacitance, steady state resistance, and the time constant is given by:

$$\tau = R * C$$

Where:

- τ is the time constant (in seconds)
- R is the steady state resistance (in ohms)
- C is the battery capacitance behavior (in farads)

Rearranging the equation to solve for capacitance C, results in:

$$C = \frac{\tau}{R}$$

Understanding battery capacitance helps designers of prototype battery packs see how significant capacitance behavior impacts the charging and discharging processes.

11.4 Future amendments

This section outlines some **potential major improvements** for the proposed test bench that can be implemented **with** requiring additional financial support:

11.4.1 IOT feature

The temperature and humidity in the prototyping factory's environment can directly influence the insulation paste injection during the assembly process. If these ambient conditions are too high, poor insulation may be detected in isolation tests. This highlights the importance of continuously recording ambient temperature and humidity in the assembly line (prototyping factory) to ensure the factory's ventilation system is sufficiently stable.

Figure 32 shows an Arduino Uno and a DHT11 sensor used to measure humidity and temperature. However, new hardware such as the Arduino Ethernet Shield 2 [A000024] needs to be integrated with the Arduino Uno using the Ethernet protocol to store these ambient measurements in a cloud storage dataset. This Arduino Ethernet Shield can also support an SD memory card for physically storing the ambient measurement data. Figure 96 shows Arduino Ethernet Shield 2 [A000024] and its needed components for this further amendment.

Figure 96. Components needed for IOT affairs

11.4.2 Raspberry Pi Local Server

A Raspberry Pi 5 local server can continuously record ambient conditions such as humidity and temperature. The device illustrated in Figure 97 can be directly connected to the Arduino Ethernet Shield via an Ethernet cable. If opting to avoid third-party servers for remote monitoring (as discussed in Subsection 8.3), the Raspberry Pi server can serve as a local and secure option for hosting remote monitoring or cloud storage data.

Figure 97. Components needed for local server

11.4.3 Simulated BMS for battery pack:

In Subsection 1.6, the battery pack under test was called a “Brainless battery” because of the lack of a BMS unit. However, since the battery pack under test in serial communication was simulated, the BMS unit can also be simulated as well with Seeed Studio CAN Bus Shield V2 shown in Figure 98. With simulating BMS unit, a fully assembled unit of electric car battery can be simulated. With Conrverter Savvy device, UART protocol will be converted into CAN protocol and Converter Savvy CAN-FD is able to communicate with Seeed Studio CAN Bus Shield. The famous software tools that can be used for Converter Savvy CAN-FD are SavvyCAN and Busmaster which are able to communicate with LabVIEW. Since these softwares are supporting LabVIEW through USB ports, TestStand can be applied as well for testing simulated BMS communication.

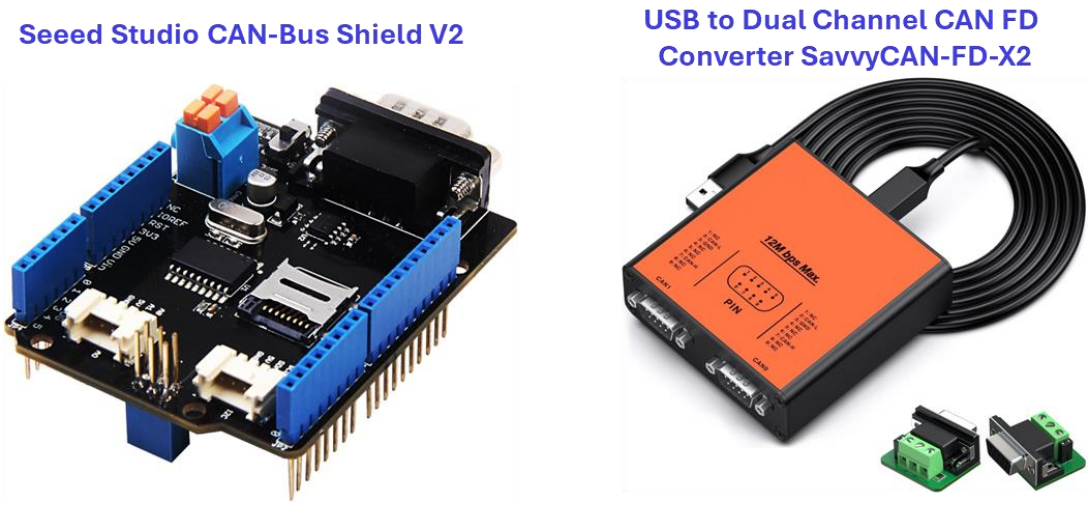


Figure 98. Components needed for IOT affairs

12 Conclusion

This proposed automation testing setup was designed on a low budget, using inexpensive Arduino boards as controllers, unlike the actual testing model at IONCOR's prototyping factory, which uses costly boards. In this setup, Arduino Uno and Arduino Mega were used instead of the expensive and complex combination of National Instruments and Texas Instruments devices for reading battery signals. The safety testing device in the actual model is a "Chroma" device, while this setup uses a "GW Instek" device, which required the author to learn new LabVIEW modifications for the proposed test bench.

The same software tools, NI LabVIEW and NI TestStand, were used in this setup due to license availability at IONCOR's prototyping factory. Essential tests such as equipotential, dielectric, and isolation tests, along with new features like graphs, were successfully developed in this setup, though the graphs were not available in the introduced actual automation test setup. Super-loop architecture was selected for easier debugging and interaction with the battery, instead of EDA used by the actual automation test setup. Instead of using Microsoft SQL Server in the actual setup, File Share in Microsoft Azure was used, which seems more intuitive and cheaper. For remote control, a different third-party company with more features was suggested, replacing the one used in the actual model. Additionally, test report files are now more securely transferred using the FTP protocol instead of direct access to the SQL Server database.

Reference

[1] "Automotive Battery Market - Global Industry Analysis and forecast (2024-2030)," MAXIMIZE MARKET RESEARCH, https://www.maximizemarketresearch.com/market-report/global-automotive-battery-market/55569/ (accessed May 06, 2024).

[2] IONCOR Oy, "Electrifying the global energy transition - Ioncor," Ioncor, Aug. 27, 2024. https://ioncor-batteries.com/ (accessed August 30, 2024).

[3] J. Captieux and J. Captieux, "EV Equipotential Bonding – Part 1," Professional Motor Mechanic, May 07, 2024. https://pmmonline.co.uk/article/ev-equipotential-bonding-part-1/ (accessed May 26, 2024).

[4] S. Lower,"16.7: Timeline of Battery Development," *Chemistry LibreTexts*, Jul. 05, 2017.https://chem.libretexts.org/Bookshelves/General_Chemistry/Chem1_%28Lower%29/16%3A_Electrochemistry/16.07%3A_Timeline_of_Battery_Development (accessed May 06, 2024).

[5] V. Scott, "From Lead-Acid To Lithium: A History of the Automotive Battery," *Road Track*, Aug. 22, 2023. https://www.roadandtrack.com/car-culture/a45597186/history-of-automotive-batteries/ (accessed May 06, 2024).

[6] D. Galatro, M. Al-Zareer, C. Da Silva, D. A. Romero, and C. H. Amon, "THERMAL BEHAVIOR OF LITHIUM-ION BATTERIES: AGING, HEAT GENERATION, THERMAL MANAGEMENT AND FAILURE," *Frontiers in Heat and Mass Transfer*, vol. 14, May 2020, doi: https://doi.org/10.5098/hmt.14.17.

[7] M. Dubarry, N. Qin, and P. Brooker, "Calendar aging of commercial Li-ion cells of different chemistries – A review," *Current Opinion in Electrochemistry*, vol. 9, pp. 106–113, Jun. 2018, doi: https://doi.org/10.1016/j.coelec.2018.05.023.

[8] Sustainable-bus, "NMC, LFP, LTO. What's the Difference? [The Battery Cycle #2]," *Sustainable Bus*, Apr. 09, 2021. https://www.sustainable-bus.com/news/nmc-lfp-lto-battery-explained/ (accessed May 06, 2024).

[9] Celltech, "Battery Technologies | Celltech Solutions," *celltechsolutions.fi*, Mar. 27, 2021. https://celltechsolutions.fi/products/our-technologies/ (accessed May 06, 2024).

[10] CircuitDigest, "A Detailed Comparison of Popular Li-ion Battery Chemistries used in Electric Vehicles," circuitdigest.com, Apr. 11, 2022. https://circuitdigest.com/article/a-detailed-comparision-of-popular-li-ion-battery-chemistries-used-in-evs (accessed May 06, 2024).

[11] Grepow, "Prismatic vs Pouch vs Cylindrical Lithium Ion Battery Cell | Grepow," *www.grepow.com*, Jan. 31, 2024. https://www.grepow.com/blog/prismatic-vs-pouch-vs-cylindrical-lithium-ion-battery-cell.html#:~:text=Prismatic%20cells%20are%20less%20flexible (accessed May 06, 2024).

[12] QuantumScape, “EV Battery Cell Formats for Lithium Metal,” *QuantumScape*, Jan. 16, 2024. https://www.quantumscape.com/blog/ev-battery-cell-formats-for-lithium-metal/ (accessed May 06, 2024).

[13] BlueHeronBattery, “Lithium Batteries Deep Cycle,” *Blue Heron Lithium Battery*, Oct. 23, 2023. https://www.blueheronbattery.com/prismatic-cylindrical-pouch-pros-and-cons (accessed May 06, 2024).

[14] CellSaviors, “Pouch Vs Cylindrical Lithium Cells Which is Best,” *cellsaviors.com*, Feb. 29, 2024. https://cellsaviors.com/blog/pouch-or-cylindrical-cells (accessed May 06, 2024).

[15] Hioki, “Inspection in Assembly of Li-ion Battery Packs for EVs | Manufacturing & Inspection | Hioki,” *Hioki.com*, 2021. https://www.hioki.com/global/industries-solutions/manufacturing/liion-pack-production-testing.html (accessed May 06, 2024).

[16] “Communication Protocols for a Battery Management System (BMS),” *www.learningaboutelectronics.com*, 2018. https://www.learningaboutelectronics.com/Articles/Communication-protocols-for-a-battery-management-system.php (accessed May 06, 2024).

[17] WatElectronics, “UART Communication : Features, Modes & Applicatons,” WatElectronics.com, Dec. 14, 2019. https://www.watelectronics.com/basics-of-uart-communication/ (accessed May 06, 2024).

[18] Arrow Electronics, “What is I2C? How Inter-Integrated Circuits Work,” *Arrow.com*, Feb. 04, 2019. https://www.arrow.com/en/research-and-events/articles/what-is-i2c-how-inter-integrated-circuits-work (accessed May 06, 2024).

[19] J. Wu, “A Basic Guide to I2C,” Nov. 2022. Accessed: May 06, 2024. [Online]. Available: https://www.ti.com/lit/an/sbaa565/sbaa565.pdf (accessed May 06, 2024).

[20] Parlez-vous Tech, “Understanding Basic Communication Protocols: I2C, SPI, and UART,” *Parlez-vous Tech*, Dec. 11, 2023. https://www.parlezvoustech.com/en/ (accessed May 06, 2024).

[21] Autodesk Instructables, “How to Display Images on OLED Using Arduino,” Instructables, 2021. https://www.instructables.com/How-to-Display-Images-on-OLED-Using-Arduino/ (accessed May 05, 2024).

[22] R. Sheldon, “What is serial peripheral interface (SPI)? - Definition from WhatIs.com,” *TechTarget*, Aug. 2023. https://www.techtarget.com/whatis/definition/serial-peripheral-interface-SPI (accessed May 06, 2024).

[23] M. Cantrell, “Isolating SPI for High Bandwidth Sensors,” *Analog.com*, 2014. https://www.analog.com/en/resources/technical-articles/isolating-spi-for-high-bandwidth-sensors.html (accessed May 06, 2024).

[24] Articles Saleae, “SPI vs I2C Protocol Differences and Things to Consider - Saleae Articles,” *Saleae.com*, 2019. https://articles.saleae.com/logic-analyzers/spi-vs-i2c-protocol-differences-and-things-to-consider (accessed May 06, 2024).

[25] Can in automation (CIA), “CAN in Automation (CiA): History of the CAN technology,”, 2023. https://www.can-cia.org/can-knowledge/can/can-history (accessed May 06, 2024).

[26] National Instruments, “Controller Area Network (CAN) Overview & Specifications,” *www.ni.com*, May 23, 2024. https://www.ni.com/en/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/controller-area-network--can--overview.html (accessed May 31, 2024).

[27] Kvaser, “CAN Bus Protocol Tutorial,” *Kvaser*, Sep. 29, 2022. https://kvaser.com/can-protocol-tutorial/ (accessed May 31, 2024).

[28] Raghumanth A, “What Is a Battery Management System (BMS)? Definition, Objectives, Components, Types, and Best Practices,” *TenXer Labs*, Oct. 07, 2024. https://tenxerlabs.com/resources/blogs/the-basics-of-battery-management-systems/ (accessed May 06, 2024).

[29] GW Instek, “Electrical Safety Tester, GPT-9000 / GPT-9000A Series,” GW Instek, 2021. Accessed: May 06, 2024. [Online]. Available: https://www.gwinstek.com/en-global/products/detail/GPT-9800

[30] GeeksforGeeks, “What is OSI Model | 7 Layers Explained,” GeeksforGeeks, Aug. 30, 2017. https://www.geeksforgeeks.org/open-systems-interconnection-model-osi/(accessed May 06, 2024).

[31] M. Flinders and I. Smalley, “Firmware,” Ibm.com, Sep. 16, 2024. https://www.ibm.com/think/topics/firmware (accessed September 31, 2024).

[32] Embien, “Embien Technologies:: Modbus: In-Depth guide Industrial Protocol,” Embien, Oct. 23, 2023. https://www.embien.com/industrial-insights/modbus-an-in-depth-guide-to-the-widely-used-industrial-protocol (accessed May 05, 2024).

[33] Ccontrols, “Introduction to Modbus Serial and Modbus TCP,” 2008. https://www.ccontrols.com/pdf/Extv9n5.pdf (accessed May 05, 2024).

[34] D. Asturias, “Network Topology Guide for 2024: Mapping, Types, & Design,” *Websentra*, Dec. 07, 2021. https://www.websentra.com/network-topology-guide/ (accessed May 07, 2024).

[35] Microchip Innovation, “Home | Microchip Technology,” Microchip.com, 2019. https://www.microchip.com/en-us/solutions/automotive-and-transportation/automotive-products/memory-products/serial-eeproms (accessed May 31, 2024).

[36] Artekit, “All about GPIOs,” Artekit.eu, Jan. 13, 2020. https://www.artekit.eu/doc/guides/all-about-gpios/ (accessed May 05, 2024).

[37] GeeksforGeeks, “Error Detection in Computer Networks,” GeeksforGeeks, Jul. 12, 2024. https://www.geeksforgeeks.org/error-detection-in-computer-networks/ (accessed May 06, 2024).

[38] CRCcalc, “Online CRC-8 CRC-16 CRC-32 Calculator,” Crccalc.com, 2024. https://crccalc.com/?crc=93%2001%2001%2000%2002%20B7%2078%20BC&method=CRC-16&datatype=hex&outtype=hex (accessed May 05, 2024).

[39] NTC thermistor calculator, “NTC thermistor calculator,” www.lasercalculator.com, 2024. https://www.lasercalculator.com/ntc-thermistor-calculator/ (accessed May 05, 2024).

[40] National Instruments, “What is LabVIEW?,” www.ni.com, 2024. https://www.ni.com/en/shop/LabVIEW.html (accessed May 31, 2024).

[41] Arduino, “What Is Arduino?,” *Arduino.cc*, Feb. 05, 2018. https://www.arduino.cc/en/guide/introduction (accessed May 31, 2024).

[42] National Instruments, “What Is TestStand?,” *www.ni.com*, 2024. https://www.ni.com/en/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-teststand.html (accessed May 31, 2024).

[43] National Instruments, “Product Documentation - NI,” Ni.com, 2024. https://www.ni.com/docs/en-US/bundle/labview/page/labview-manager-data-types.html?srsltid=AfmBOopoWQzihhjlVNG1bhKW2Vhy2bi0gpsA0VkjWYRkiwdd1okg0aB3 (accessed May 31, 2024).

[44] Rally Tec, “ELECTRIC VEHICLE HIGH VOLTAGE SYSTEM TESTING INFORMATION SHEET,” 2023. Accessed: May 31, 2024. [Online]. Available: https://www.rallytec.co.uk/electric_vehicle_testing%20_the%20_high_voltage_system_information_sheet.pdf (accessed May 31, 2024).

[45] All About Circuits, “Kelvin (4-wire) Resistance Measurement | DC Metering Circuits | Electronics Textbook,” *Allaboutcircuits.com*, 2015. https://www.allaboutcircuits.com/textbook/direct-current/chpt-8/kelvin-resistance-measurement/ (accessed May 31, 2024).

[46] Megger, *A Stitch in TIME “The Complete Guide to Electrical Insulation Testing.”* 2006. Accessed: May 31, 2024. [Online]. Available: www.megger.com

[47] TestSigma, “A Detailed Overview on Cloud Testing: Benefits, Types and Tools,” *Testsigma Blog*, Jan. 04, 2023. https://testsigma.com/blog/cloud-testing/ (accessed May 06, 2024).

[48] Microsoft Learning, “Resource naming restrictions - Azure Resource Manager,” *learn.microsoft.com*, May 20, 2024. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/resource-name-rules (accessed May 06, 2024).

[49] Microsoft Learning, “Understand Azure Files performance,” *learn.microsoft.com*, May 13, 2024. https://learn.microsoft.com/en-us/azure/storage/files/understand-performance (accessed May 06, 2024)

[50] Microsoft Learning, “Data redundancy in Azure Files,” *learn.microsoft.com*, Mar. 27, 2024. https://learn.microsoft.com/en-us/azure/storage/files/files-redundancy (accessed May 06, 2024)

[51] Microsoft Learning, “Azure Files Pricing | Microsoft Azure,” *azure.microsoft.com*, 2024. https://azure.microsoft.com/en-us/pricing/details/storage/files/ (accessed May 06, 2024)

[52] GeeksforGeeks, “What is Transmission Control Protocol (TCP)?,” GeeksforGeeks, Nov. 25, 2021. https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/ (accessed May 05, 2024).

[53] Microsoft Learning, “Tutorial: Share outside your org - Azure Data Share,” Microsoft.com, Dec. 19, 2023. https://learn.microsoft.com/en-us/azure/data-share/share-your-data?tabs=azure-portal (accessed May 06, 2024).

[54] TailScale, “Secure, remote access ,” *Tailscale*, 2024. https://tailscale.com/ (accessed May 06, 2024).

[55] WinSCP, “WinSCP,” WinSCP - Free SFTP and FTP client, Dec. 05, 2019. https://winscp.net/eng/index.php (accessed May 06, 2024).

[56] GeeksforGeeks, “VHDL Very High Speed Integrated Circuit Hardware Description Language,” *GeeksforGeeks*, Jun. 25, 2024. https://www.geeksforgeeks.org/vhdl-very-high-speed-integrated-circuit-hardware-description-language/ (accessed May 24, 2025).

[57] Octopart, “FPGA vs. MCU: Which Processor Should You Use?,” *Octopart.com*, 2020. https://octopart.com/pulse/p/fpga-vs-mcu-which-processor-should-you-use (accessed May 24, 2025).

[58] IBM, “What is a field programmable gate array (FPGA)? | IBM,” *www.ibm.com*, May 10, 2024. https://www.ibm.com/think/topics/field-programmable-gate-arrays (accessed May 24, 2025).

[59] A. K. Oudjida, M. L. Berrandjia, R. Tiar, A. Liacha, and K. Tahraoui, “FPGA implementation of I2C & SPI protocols: A comparative study,” *2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009)*, Dec. 2009, doi: https://doi.org/10.1109/icecs.2009.5410881

[60] Transient Specialists, “ESD Testing - Overview, Methods, Testing Explained,” *Transient Specialists*, Jan. 12, 2021. https://transientspecialists.com/blogs/blog/esd-testing-equipment-methods-standards-air-contact-discharge (accessed May 10, 2025

[61] Compliance Engineering, “Compliance Engineering,” *Compliance Engineering*, Jul. 18, 2023. https://www.compeng.com.au/emc-testing-automotive-electronics/ (accessed May 10, 2025).

[62] Texas Instruments, “SSZTAM0 Technical article | TI.com,” *Ti.com*, 2016. https://www.ti.com/document-viewer/lit/html/SSZTAM0 (accessed May 24, 2025).

[63] Texas Instruments, “High EMC Immunity RS485 Interface Reference Design for Tamagawa Encoders High EMC Immunity RS485 Interface Reference Design for Tamagawa Encoders,” 2018. Accessed: May 24, 2025. [Online]. Available: https://www.ti.com/lit/ug/tidue61/tidue61.pdf (accessed May 24, 2025).

[64] Analog Devices, “isoSPI Isolated Communications Interface,” May 2019. https://www.analog.com/media/en/technical-documentation/data-sheets/ltc6820.pdf (accessed Feb. 10, 2025).

[65] T. Kugelstadt, “Extending the SPI bus for long-distance communication,” 2011. Available: https://www.ti.com/lit/an/slyt441/slyt441.pdf (accessed May 24, 2025).

[66] A. Nadler, “APPLICATION NOTE ANP121 | Filter and surge protection for I 2 C Bus,” Mar. 2023. Accessed: May 24, 2025. [Online]. Available: https://www.we-online.com/components/media/o734709v410%20ANP121a%20%20Filter%20and%20surge%20protection%20for%20I2C%20Bus%20EN.pdf (accessed May 24, 2025).

[67] R. Li, “Development and Evaluation of a Robust UART Communication System with Enhanced Interference Resistance,” *Science and Technology of Engineering, Chemistry and Environmental Protection*, vol. 1, no. 9, Oct. 2024, doi: https://doi.org/10.61173/h86z4q46

[68] Texas Instruments, “Battery management ICs | TI.com,” *www.ti.com*, 2025. https://www.ti.com/battery-management/overview.html (accessed May 10, 2025)

[69] Texas Instruments, “Battery monitors & balancers | TI.com,” *Ti.com*, 2024. https://www.ti.com/battery-management/monitors-balancers/overview.html (accessed Mar. 10, 2025).

[70] Microcontrollers Lab, “Bare-metal and RTOS Based Embedded Systems,” Microcontrollers Lab, Sep. 10, 2020. https://microcontrollerslab.com/difference-bare-metal-and-rtos-based-embedded-systems/ (accessed May 02, 2025).

[71] A. Serino and L. Cheng, “A Survey of Real-Time Operating Systems,” 2019. [Online]. Available: https://engineering.lehigh.edu/sites/engineering.lehigh.edu/files/_DEPARTMENTS/cse/research/tech-reports/2019/LU-CSE-19-003.pdf (accessed May 02, 2025).

[72] I. Ferguson, “What Are the Most Popular Real-Time Operating Systems in 2024?,” Lynx.com. https://www.lynx.com/blog/most-popular-real-time-operating-systems-rtos (accessed May 02, 2025).

[73] IBM, “Real-time operating system (RTOS),” Ibm.com, Mar. 26, 2025. https://www.ibm.com/think/topics/real-time-operating-system (accessed May 01, 2025).

[74] STMicroelectronics, “STM32F407VG - High-performance foundation line, Arm Cortex-M4 core with DSP and FPU, 1 Mbyte of Flash memory, 168 MHz CPU, ART Accelerator, Ethernet, FSMC - STMicroelectronics,” www.st.com, 2025. https://www.st.com/en/microcontrollers-microprocessors/stm32f407vg.html (accessed May 01, 2025).

[75] I. Ferguson, “Event-Driven Architecture (EDA): A Complete Introduction,” Confluent, 2025. https://www.confluent.io/learn/event-driven-architecture/#event-driven-architecture-and-microservices (accessed May 01, 2025).

[76] M. Insider, "What is software-in-the-loop testing?," Aptiv, Mar. 17, 2022. [Online]. Available: https://www.aptiv.com/en/insights/article/what-is-software-in-the-loop-testing (accessed May 01, 2025).

[77] M. Insider, "What is hardware-in-the-loop testing?," Aptiv, Mar. 24, 2022. [Online]. Available: https://www.aptiv.com/en/insights/article/what-is-hardware-in-the-loop-testing (accessed May 01, 2025).

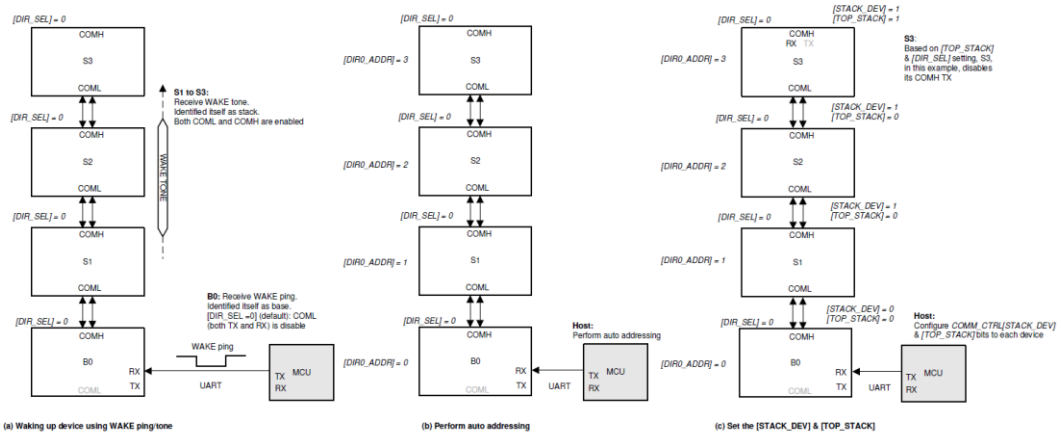
[78] Protective Industrial Polymers, “Static Control Flooring & How it Works With ESD,” Protective Industrial Polymers, Feb. 06, 2017.

https://protectiveindustrialpolymers.com/esd-floor-take-static-charge-ground/ (accessed May 01, 2025).

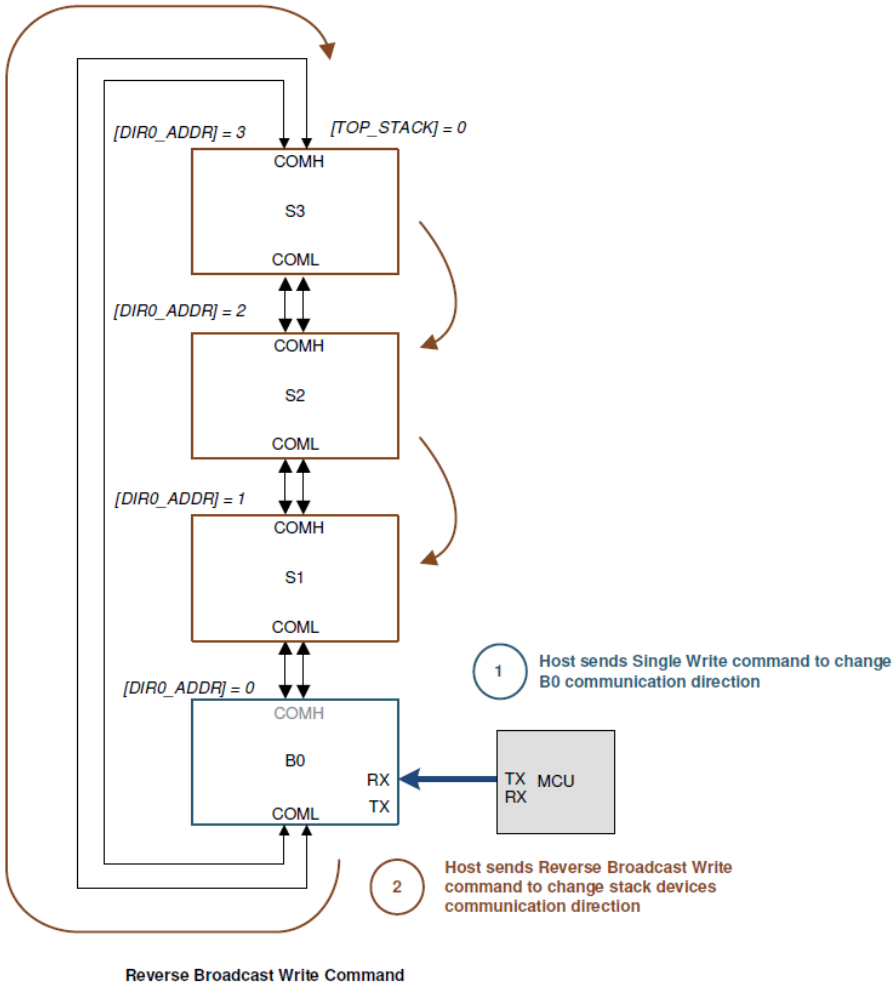
[79] Battery University, “BU-802b: What does Elevated Self-discharge Do?,” Battery University, Sep. 26, 2010. https://batteryuniversity.com/article/bu-802b-what-does-elevated-self-discharge-do (accessed May 01, 2025).

Appendix A. Auto-addressing Details

Part 1. Configuring the peripherals (Stack or base) for initialzing the communication:

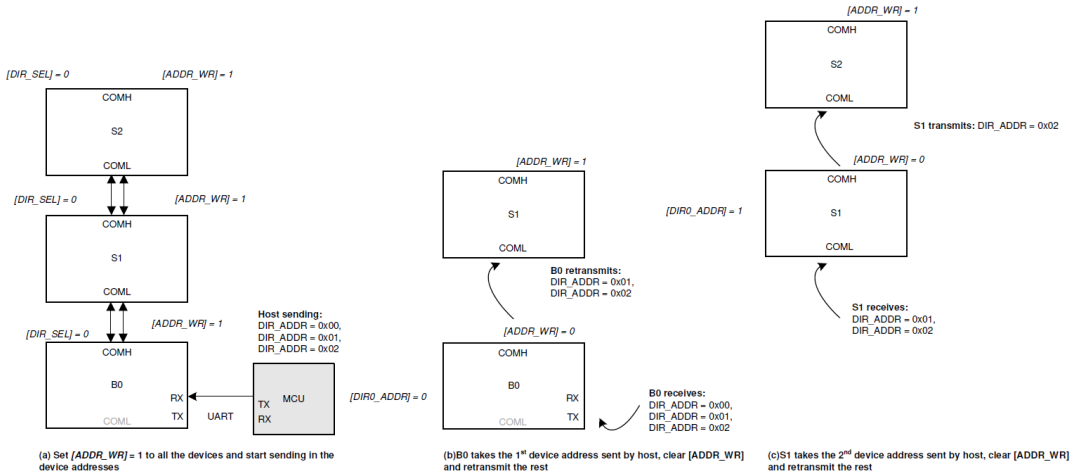


Part 2. Broadcast Write Reverse Direction



Reverse Broadcast Write Command

Part 3. Auto-Addressing Steps:

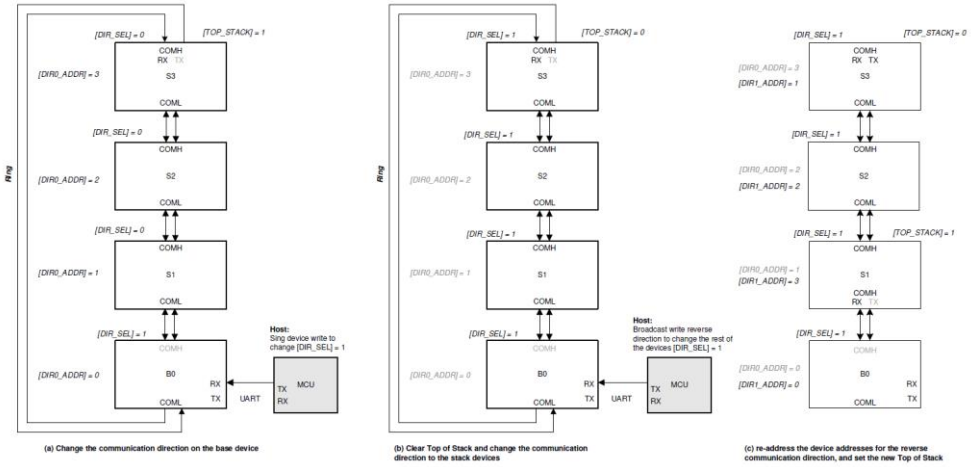


(a) Set *[ADDR_WR]* = 1 to all the devices and start sending in the device addresses

(b)B0 takes the 1st device address sent by host, clear [ADDR_WR] and retransmit the rest

(c)S1 takes the 2nd device address sent by host, clear [ADDR_WR] and retransmit the rest

Part 4. Example to change communication direction in Daisy Chain:



(a) Change the communication direction on the base device

(b) Clear Top of Stack and change the communication direction to the stack devices

(c) re-address the device addresses for the reverse communication direction, and set the new Top of Stack

Part 5. Various registery addresses used in Modbus communication

COMM_CTRL Registery address of 0x0308

COMM_CTRL

Address	**0x0308**							
RW	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Name	RSVD						STACK_DEV	TOP_STACK
Reset	0	0	0	0	0	0	0	0

RSVD = Reserved

STACK_DEV = Defines device as a base or stack device in daisy chain configuration.
0 = Base device
1 = Stack device

TOP_STACK = Defines device as highest addressed device in the stack.
0 = Not the ToS device
1 = Is the ToS device

CONTROL1 Registery address of 0x0309

CONTROL1

Address	0x0309							
RW	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Name	DIR_SEL	SEND_ SHUTDOWN	SEND_WAKE	SEND_ SLPTOACT	GOTO_ SHUTDOWN	GOTO_ SLEEP	SOFT_RESET	ADDR_WR
Reset	0	0	0	0	0	0	0	0

DIR_SEL = Selects daisy chain communication direction.
0 = With two devices connected in daisy chain, command frame travels from COMH of the lower device to COML of the next device.
1 = With two devices connected in daisy chain, command frame travels from COML of the lower device to COMH of the next device.

SEND_SHUTDOWN = Sends SHUTDOWN tone to next device up the stack. The device receiving this bit set is unaffected. Bit is cleared on read.
0 = Ready
1 = Send SHUTDOWN tone up the stack

SEND_WAKE = Sends WAKE tone to next device up the stack. Bit is cleared on read.
0 = Ready
1 = Send WAKE tone to next device up the stack.

SEND_SLPTOACT = Sends SLEEPtoACTIVE tone up the stack. Bit is cleared on read.
0 = Ready
1 = Send SLEEPtoACTIVE tone up the stack

GOTO_SHUTDOWN = Transitions device to SHUTDOWN mode. Bit is cleared on read.
0 = Ready
1 = Enter SHUTDOWN mode

GOTO_SLEEP = Transitions device to SLEEP mode. Bit is cleared on read.
0 = Ready
1 = Enter SLEEP mode

SOFT_RESET = Resets the digital to OTP default. Bit is cleared on read. Setting this bit will cause the device to generate WAKE tone to the upper stack devices.
0 = Ready
1 = Reset device

ADDR_WR = Enables device to start auto-addressing. When this bit is set, device will not forward the first transition it receives, allowing the device address to be written to a single device.
0 = Not performing auto-address. Device forwards communication transaction as normal.
1 = Device is being auto-addressed; the first communication transaction it receives will not be forwarded.

DIR0_ADDR_OTP Registery address of 0x0000

DIR0_ADDR_OTP

Address	0x0000							
NVM	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Name	SPARE[1:0]		ADDRESS[5:0]					
Reset	0	0	0	0	0	0	0	0

SPARE[1:0] = Spare

ADDRESS[5:0] = This register shows the default device address used when *[DIR_SEL]* = 0 and programmed in the OTP. Writing to this register won't change the device address actively in use.
This register is used for the system to program the device address to OTP, which will be loaded to the *DIR0_ADDR* register at POR. For programming, follow the OTP programming procedure.

DIR0_ADDR Registery address of 0x0306

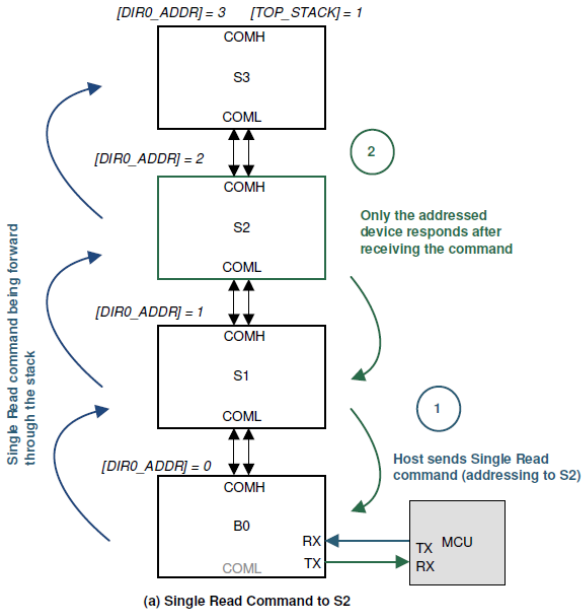
DIR0_ADDR

Address	0x0306							
RW	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Name	RSVD		ADDRESS[5:0]					
Reset	0	0	0	0	0	0	0	0

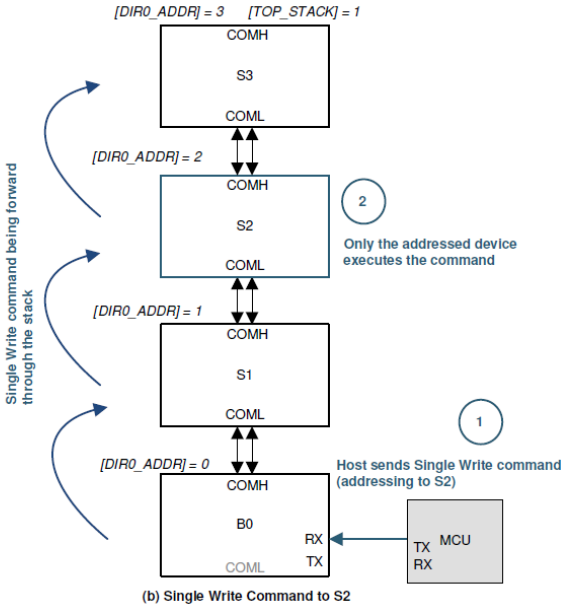
RSVD = Reserved

ADDRESS[5:0] = Always shows the current device address used by the device when *[DIR_SEL]* = 0. At POR, this register is loaded from the device address value in the OTP (same OTP device address loaded to *DIR0_ADDR_OTP* register). Host can re-address the device by writing a different device address to this register, and the device will take on the new address immediately.
Note: *CONTROL1[ADDR_WR]* = 1 is required to write to this register.

Part 6. Single read or single write command:



(a) Single Read Command to S2



(b) Single Write Command to S2

Appendix B. Battery Pack and Controller Simulation

Part 1. Arduino Mega (Test bench's Controller)

```
void setup() {
  // Start communication between PC and Mega at 9600 baud rate (USB
communication)
  Serial.begin(9600);
  // Start UART communication with Arduino Micro at 9600 baud rate using
Serial1 (pins 18 TX1, 19 RX1)
  Serial1.begin(9600);
}

void loop() {
  // Check if a command is sent via the USB serial communication (from the
Serial Monitor)
  if (Serial.available()) {
    String command = Serial.readStringUntil('\n'); // Read the command from
the user
    command.trim();  // Remove any extra whitespace or newline characters

    // Forward the command to the slave via UART (Serial1)
    Serial1.println(command);


    delay(2000); // Give the slave some time to respond

    // Check if there is data available from the slave
    while (Serial1.available()) {
      // Read and display the slave's response on the Serial Monitor
      String response = Serial1.readStringUntil('\n'); // Read until newline

      Serial.println(response); // Print the response with commas
    }
  }
}
```

Part 2. Arduino Micro 1 (Battery module's peripheral)

```
#include <SoftwareSerial.h>
#include <SPI.h>
// Define RX and TX pins for SoftwareSerial
SoftwareSerial mySerial(9, 10); // RX = 9, TX = 10
const int chipSelectPin = 17; // New CS pin for the Micro

void setup() {
  // Start communication between PC and Micro at 9600 baud rate
  Serial.begin(9600);
  Serial1.begin(9600);
  // Start UART communication with Arduino Mega at 9600 baud rate
```

```
  mySerial.begin(9600);
  // Start UART communication with secondary device at 9600 baud rate
  SPI.begin();
  pinMode(chipSelectPin, OUTPUT);
  // Seed the random number generator
  randomSeed(analogRead(0)); // Seed with noise from an analog pin
}

void loop() {
  if (mySerial.available()) {
    String command = mySerial.readStringUntil('\n'); // Read the incoming
command

    // Debugging: Print the received command to Serial Monitor
    Serial.println("Command received from Mega: " + command);

    // Send the command to secondary UART pins 5 and 4
    Serial1.println(command);
    Serial.println("Forwarded command to secondary serial: " + command); //
Debug print
    delay(50); // Increase delay to ensure stable forwarding

    // Trim any whitespace, newline, or carriage return characters
    command.trim();

    // Now check the command and respond accordingly
    if (command == "A1") {
      String response = ""; // Initialize response string
      for (int i = 0; i < 10; i++) {
        float randomValue = random(3550, 3555) / 1000.0; // Generate random
float value between 3.550 and 3.554
        response += String(randomValue, 3); // Concatenate individual value
as string with 3 decimal places
        if (i < 9) response += ","; // Add a comma if not the last value
      }
      mySerial.println(response); // Send the complete response
      //Serial.println("Sent A1 response: " + response); // Debug print
      delay(50); // Increase delay after sending response
    } else if (command == "A2") {
      String response = ""; // Initialize response string
      for (int i = 0; i < 9; i++) {
        float randomValue = random(3555, 3560) / 1000.0; // Generate random
float value between 3.555 and 3.560
        response += String(randomValue, 3); // Concatenate individual value
as string with 3 decimal places
        if (i < 8) response += ","; // Add a comma if not the last value
      }
      mySerial.println(response); // Send the complete response
      //Serial.println("Sent A2 response: " + response); // Debug print
      delay(50); // Increase delay after sending response
    } else if (command == "A3") {
```

```
      String response = ""; // Initialize response string
      for (int i = 0; i < 6; i++) {
        // Generate a random float value within ±2 units of 1.555 and 1.850
        float baseValue = (i % 2 == 0) ? 1555.0 : 1850.0; // Alternate
between 1.555 and 1.850
        int randomOffset = random(-2, 3); // Random offset between -2 and +2
        float adjustedValue = baseValue + randomOffset; // Adjust base value
        response += String(adjustedValue / 1000.0, 3); // Convert to float
and concatenate
        if (i < 5) response += ","; // Add a comma if not the last value
      }
      mySerial.println(response); // Send the complete response
      //Serial.println("Sent A3 response: " + response); // Debug print
      delay(50); // Increase delay after sending response
    }else if (command.startsWith("A4*")) { // Check for A4* command
      String dataToSend = command.substring(3); // Extract data after "B4*"

      // Send data to Nano via SPI
      digitalWrite(chipSelectPin, LOW); // Begin SPI communication
      delay(10); // Short delay for stability
      for (int i = 0; i < dataToSend.length(); i++) {
        SPI.transfer(dataToSend[i]); // Send each character
        delay(200); // Increased delay to ensure Nano can handle the byte
      }
      digitalWrite(chipSelectPin, HIGH); // End SPI communication
      Serial.println("Data sent to Nano's EEPROM.");
  }


    if (Serial1.available()) {
    String message = Serial.readStringUntil('\n'); // Read the incoming
message
    message.trim();
    Serial.println("Secondary Serial Message Received: " + message); //
Debug print
    delay(50);
       while (Serial1.available()) {
        String response2 = Serial1.readStringUntil('\n'); // Read until
newline

      mySerial.println(response2); // Print the response with commas
    }
   } else if (message == "A5+") {
      // Check for A5 command
      // Read data from Nano's EEPROM via SPI
      String receivedData = "";
      String dataToActiveRead = message;
      Serial.println(message);
      Serial.println(dataToActiveRead);

      digitalWrite(chipSelectPin, LOW); // Begin SPI communication
```

```
      delay(10); // Short delay for stability
      SPI.transfer(dataToActiveRead[0]);

      for (int i = 0; i < 12; i++) { // Read 12 bytes
        char data = SPI.transfer(0x03); // Send dummy byte to receive data
        Serial.println(data);
        receivedData += data; // Append received byte to the string
      }
      digitalWrite(chipSelectPin, HIGH); // End SPI communication
      Serial.println("Data read: " + receivedData);
    }
  }
}
```

Part 3. Arduino Micro 2 (Battery module's peripheral)

```
#include <SoftwareSerial.h>
#include <SPI.h>

// Define RX and TX pins for SoftwareSerial
SoftwareSerial mySerial(11, 12); // RX = 11, TX = 12

const int chipSelectPin = 17; // New CS pin for the Micro

void setup() {
  // Start communication between PC and Arduino at 9600 baud rate
  Serial.begin(9600);
  // Start UART communication at 9600 baud rate on pins 11 and 12
  mySerial.begin(9600);

  // SPI setup
  SPI.begin();
  pinMode(chipSelectPin, OUTPUT);
  digitalWrite(chipSelectPin, HIGH);
}

void loop() {
  // Check for incoming messages on Software Serial
  if (mySerial.available()) {
    // Read the incoming message
    String message = mySerial.readStringUntil('\n');
    // Trim any whitespace, newline, or carriage return characters
    message.trim();

    // Now check the command and respond accordingly
    if (message == "B1") {
      String response = ""; // Initialize response string
      for (int i = 0; i < 10; i++) {
        float randomValue = random(3650, 3655) / 1000.0; // Generate random
float value between 3.550 and 3.554
```

```
        response += String(randomValue, 3); // Concatenate individual value
as string with 3 decimal places
        if (i < 9) response += ","; // Add a comma if not the last value
      }
      mySerial.println(response); // Send the complete response
      delay(50); // Increase delay after sending response
    } else if (message == "B2") {
      String response = ""; // Initialize response string
      for (int i = 0; i < 9; i++) {
        float randomValue = random(3655, 3660) / 1000.0; // Generate random
float value between 3.555 and 3.560
        response += String(randomValue, 3); // Concatenate individual value
as string with 3 decimal places
        if (i < 8) response += ","; // Add a comma if not the last value
      }
      mySerial.println(response); // Send the complete response
      delay(50); // Increase delay after sending response
    } else if (message == "B3") {
      String response = ""; // Initialize response string
      for (int i = 0; i < 6; i++) {
        // Generate a random float value within ±2 units of 1.555 and 1.850
        float baseValue = (i % 2 == 0) ? 1555.0 : 1850.0; // Alternate
between 1.555 and 1.850
        int randomOffset = random(-2, 3); // Random offset between -2 and +2
        float adjustedValue = baseValue + randomOffset; // Adjust base value
        response += String(adjustedValue / 1000.0, 3); // Convert to float
and concatenate
        if (i < 5) response += ","; // Add a comma if not the last value
      }
      mySerial.println(response); // Send the complete response
      delay(50); // Small delay after sending to ensure stable message
transfer
    } else if (message.startsWith("B4*")) { // Check for B4* command
      String dataToSend = message.substring(3); // Extract data after "B4*"

      // Send data to Nano via SPI
      digitalWrite(chipSelectPin, LOW); // Begin SPI communication
      delay(10); // Short delay for stability
      for (int i = 0; i < dataToSend.length(); i++) {
        SPI.transfer(dataToSend[i]); // Send each character
        delay(200); // Increased delay to ensure Nano can handle the byte
      }
      digitalWrite(chipSelectPin, HIGH); // End SPI communication
      Serial.println("Data sent to Nano's EEPROM.");
   } else if (message == "B5+") { // Check for B5 command
      // Read data from Nano's EEPROM via SPI
      String receivedData = "";
      String dataToActiveRead = message;
      Serial.println(dataToActiveRead);

      digitalWrite(chipSelectPin, LOW); // Begin SPI communication
```

```
      delay(10); // Short delay for stability
      SPI.transfer(dataToActiveRead[0]);

      for (int i = 0; i < 12; i++) { // Read 12 bytes
        char data = SPI.transfer(0x03); // Send dummy byte to receive data
        Serial.println(data);
        receivedData += data; // Append received byte to the string
      }
      digitalWrite(chipSelectPin, HIGH); // End SPI communication
      Serial.println("Data read: " + receivedData);
    }
  }
}
```

Part 4. Arduino Nano (Battery module's EEPROM)

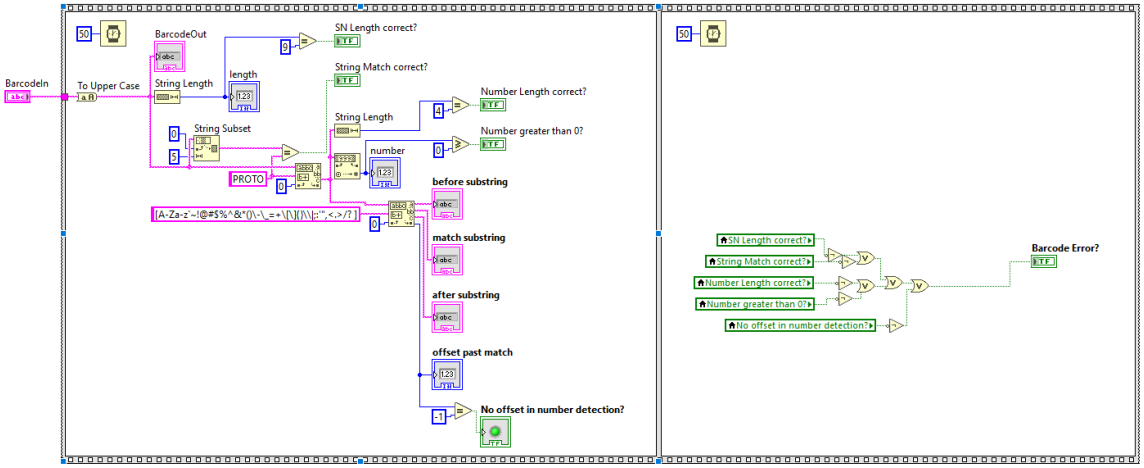
```
#include <SPI.h>
#include <EEPROM.h>

volatile bool dataAvailable = false;
volatile byte receivedData;
volatile int eepromAddress = 0; // Start at address 0
unsigned long previousMillis = 0;
const long interval = 2000; // Interval to print EEPROM data (2 seconds)
bool hasData = false; // Flag to check if data is present in EEPROM

void setup() {
  Serial.begin(9600);
  SPI.begin();
  SPI.setDataMode(SPI_MODE0);
  SPI.setClockDivider(SPI_CLOCK_DIV8); // Set SPI speed
  pinMode(10, INPUT); // SS pin for Nano (must stay as D10)
  pinMode(MISO, OUTPUT); // MISO as output in slave mode
  SPCR |= _BV(SPE);      // Enable SPI as Slave
  SPCR |= _BV(SPIE);     // Enable SPI interrupt
}

ISR(SPI_STC_vect) { // SPI Interrupt routine
  receivedData = SPDR;  // Read received data
  Serial.print("Received: "); // Debugging output
  Serial.println((char)receivedData); // Print received character for
debugging
  dataAvailable = true; // Set flag for data availability
}

void loop() {
  if (dataAvailable) {
    dataAvailable = false; // Reset flag
    if (receivedData == '*') {
      if (readingMode) {
```

```
        Serial.println("[Reading mode deactivated!]");
        readingMode = false; // Break reading mode explicitly
      }
      writingMode = true;  // Enable writing mode
      Serial.println("[Writing mode enabled!]");
    }
    else if (receivedData == '+') {
      if (writingMode) {
        Serial.println("[Writing mode deactivated!]");
        writingMode = false; // Break writing mode explicitly
      }
      inputBuffer = "+"; // Start building the input buffer for the reading
command
    }
    else if (receivedData == '\n') { // End of input command
      if (inputBuffer.startsWith("+")) {
        processReadingCommand(); // Parse the reading command
      }
    }
    else if (writingMode) {
      // Write to EEPROM only in writing mode
      if (eepromAddress < 10) { // Check EEPROM address limit
        EEPROM.update(eepromAddress, receivedData); // Save to EEPROM
        Serial.print("Writing to EEPROM Address ");
        Serial.print(eepromAddress);
        Serial.print(": ");
        Serial.println(receivedData);  // Debug output for EEPROM write
        eepromAddress++;
        hasData = true;
      } else {
        Serial.println("EEPROM limit reached. Data not saved.");
        resetEEPROMAddress(); // Reset the EEPROM address after writing
completes
      }
    } else {
      inputBuffer += (char)receivedData; // Append received data to buffer
    }
  Serial.print(dataAvailable);
  }

  if (readingMode) {
    Serial.println("Reading EEPROM data:");
    for (int i = startAddress; i <= endAddress; i++) {
      char storedChar = EEPROM.read(i); // Read from EEPROM
      Serial.print("Reading from EEPROM Address ");
      Serial.print(i);
      Serial.print(": ");
      Serial.println(storedChar);      // Debug output for EEPROM read
      while (!(SPSR & _BV(SPIF))) ;    // Wait for SPI transfer
      SPDR = storedChar;               // Send via SPI
      delay(50);                       // Ensure stable transmission
```

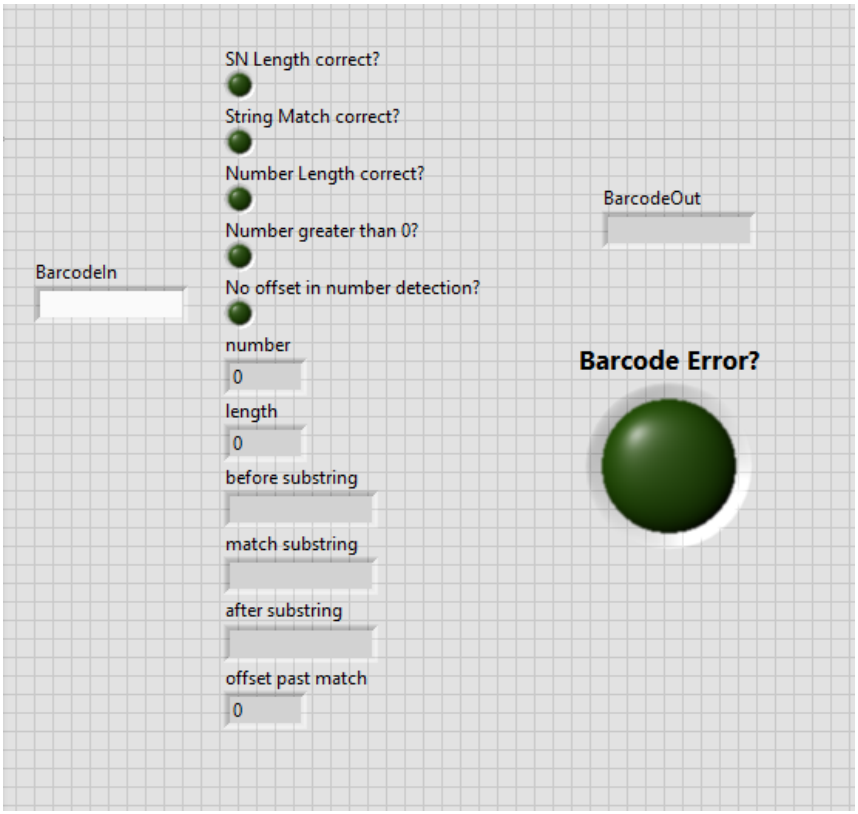
```
    }
    readingMode = false; // Exit reading mode after reading EEPROM
  }
}
```

Appendix C. Battery Pack Barcode Verification

Part1. QR barcode verification VI Block Diagram in NI LabVIEW

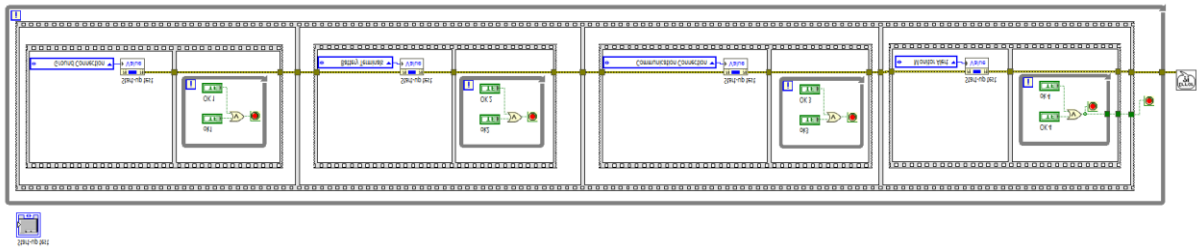


Part 2. QR barcode verification VI Front Panel in NI LabVIEW

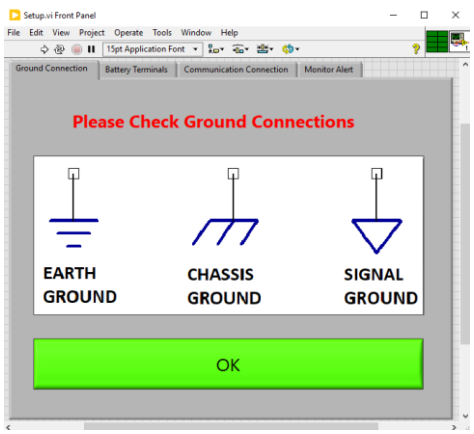


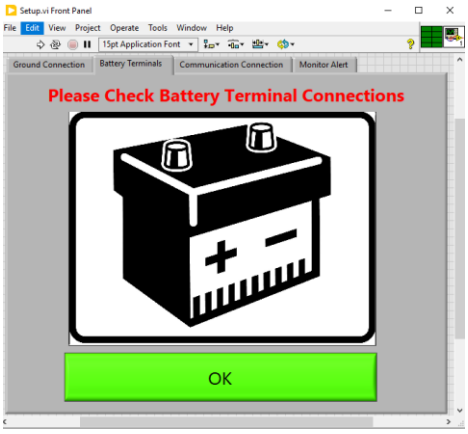
Appendix D. Safety Manual

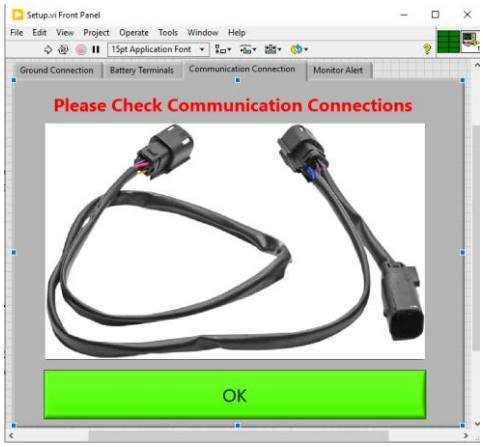
Part 1. Safety manual VI Block Diagram in NI LabVIEW

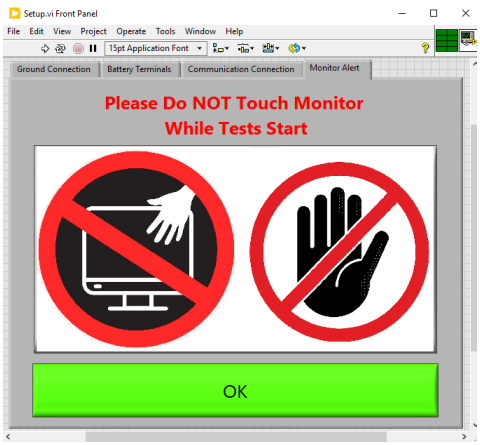


Part 2. Safety manual VI Front Panel in NI LabVIEW







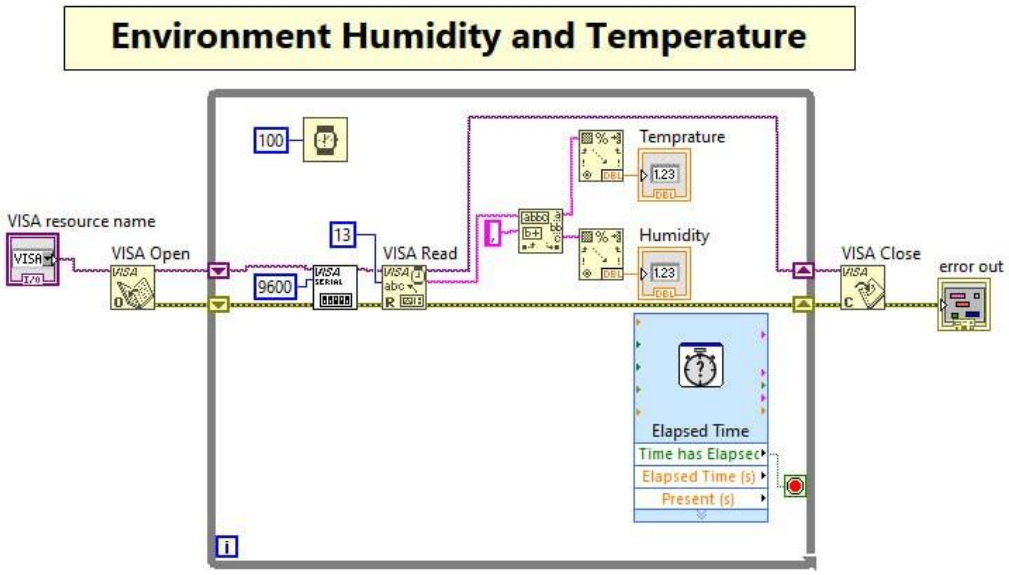


Appendix E. Ambient Evaluation (Temperature and Humidity)

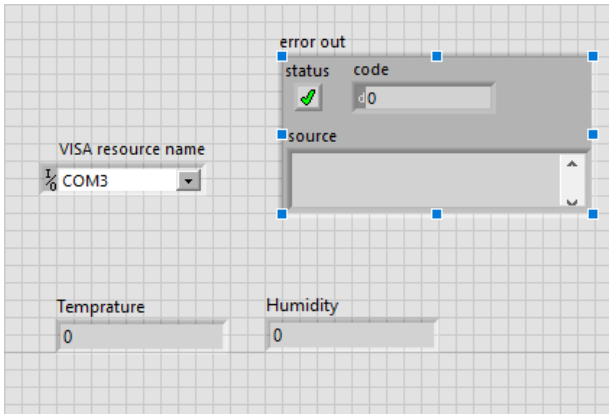
Part 1. Arduino Script for environment monitoring such temperature and humidity

```
// Temperature and Humidity Sensor monitoring with OLED Display
#include "DHT.h"
#include <Arduino.h>
#include <U8x8lib.h>
#define DHTPIN 3 // The pin is connected to
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE);
U8X8_SSD1306_128X64_ALT0_HW_I2C u8x8(/* reset=*/ U8X8_PIN_NONE);

void setup(void) {
  Serial.begin(9600);
  Serial.println("DHTxx test!");
  dht.begin();
  u8x8.begin();
  u8x8.setPowerSave(0);
  u8x8.setFlipMode(1);
}

void loop(void) {
  float temp, humi;
  temp = dht.readTemperature();
  humi = dht.readHumidity();
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.setCursor(0, 33);
  u8x8.print("Temp:");
  u8x8.print(temp);
  u8x8.print("C");
  u8x8.setCursor(0,50);
  u8x8.print("Humidity:");
  u8x8.print(humi);
  u8x8.print("%");
  u8x8.refreshDisplay();
  delay(1000);
}
```

Part 2. Environment Humidity & Temperature VI Block Diagram in NI LabVIEW



Part 3. Environment Humidity & Temperature VI Front Panel in NI LabVIEW

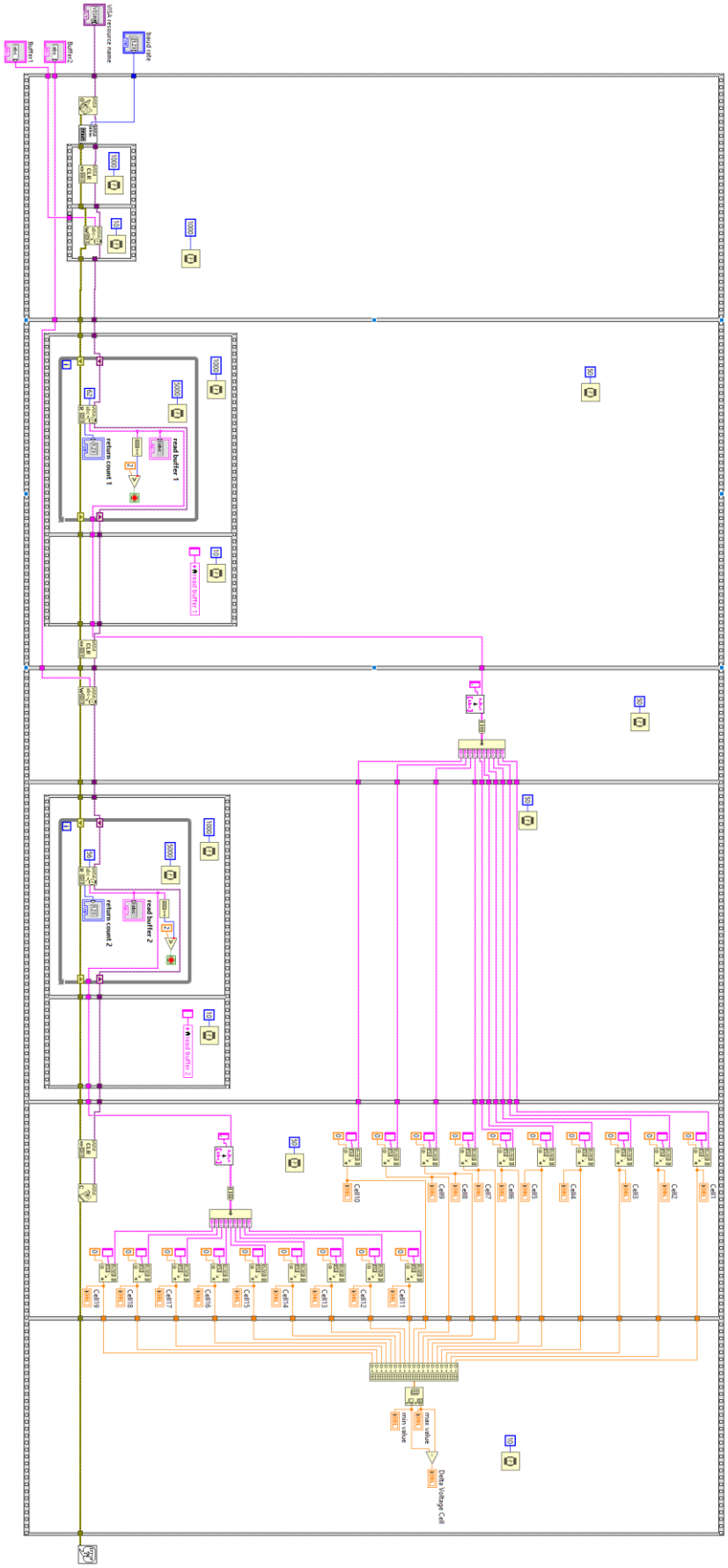


Part 4. Environment Humidity & Temperature TestStand Report

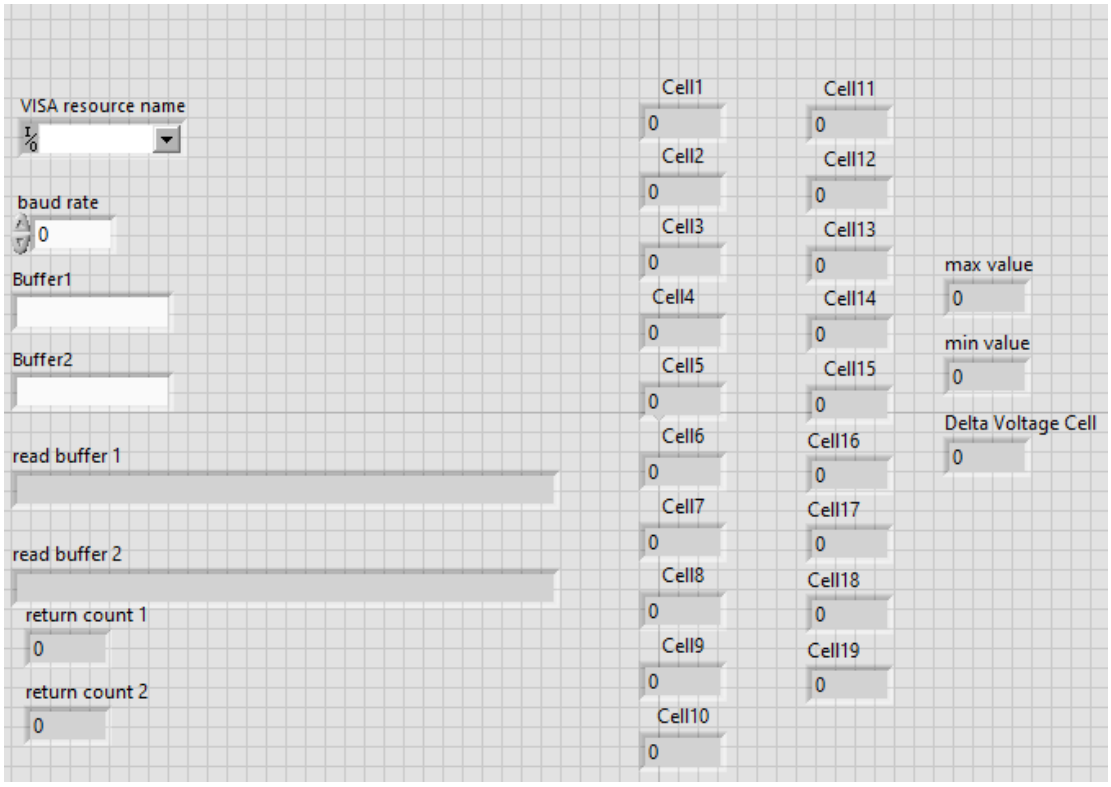
STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Environment conditions	Passed						
Measurement:							
Temprature	Passed	25.07	degrees Celsius		22	27	GELE(>= <=)
Humidity	Passed	30.49	%		20	70	GELE(>= <=)
Parameters							
VISA resource name.DeviceName:	COM3						
TestResults/Data							
error out:							
Code:	0						
Msg:	''						
Occurred:	False						

Appendix F. Battery Communication

Part 1. Monitoring voltage levels of all the battery pack cells within the LabVIEW environment (Block Diagram)



Part 2. Monitoring voltage levels of all the battery pack cells within the LabVIEW environment (Front Panel)



Part 3. Recording voltage levels of all the battery pack cells within the TestStand report (Cell Stack 1 or battery Module 1)

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Voltage Cell Stack1	Passed						

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Stack1	Passed						
Measurement:							
Cell1	Passed	3.550000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell2	Passed	3.554000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell3	Passed	3.554000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell4	Passed	3.551000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell5	Passed	3.554000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell6	Passed	3.550000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell7	Passed	3.550000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell8	Passed	3.552000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell9	Passed	3.554000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell10	Passed	3.553000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell11	Passed	3.556000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell12	Passed	3.556000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell13	Passed	3.555000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell14	Passed	3.559000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell15	Passed	3.559000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell16	Passed	3.557000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell17	Passed	3.558000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell18	Passed	3.559000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell19	Passed	3.556000	millivolt		3.405000	3.705000	GELE(>= <=)
Maximum Voltage Cell	Passed	3.559000	millivolt		3.405000	3.705000	GELE(>= <=)
Minmum Votlage Cell	Passed	3.550000	millivolt		3.405000	3.705000	GELE(>= <=)
Delta Voltage cell	Passed	0.009000	millivolt		0.000000	30.000000	GELE(>= <=)
Parameters							
VISA resource name.DeviceName:	COM2						
baud rate :	9600						

End Sequence: MainSequence

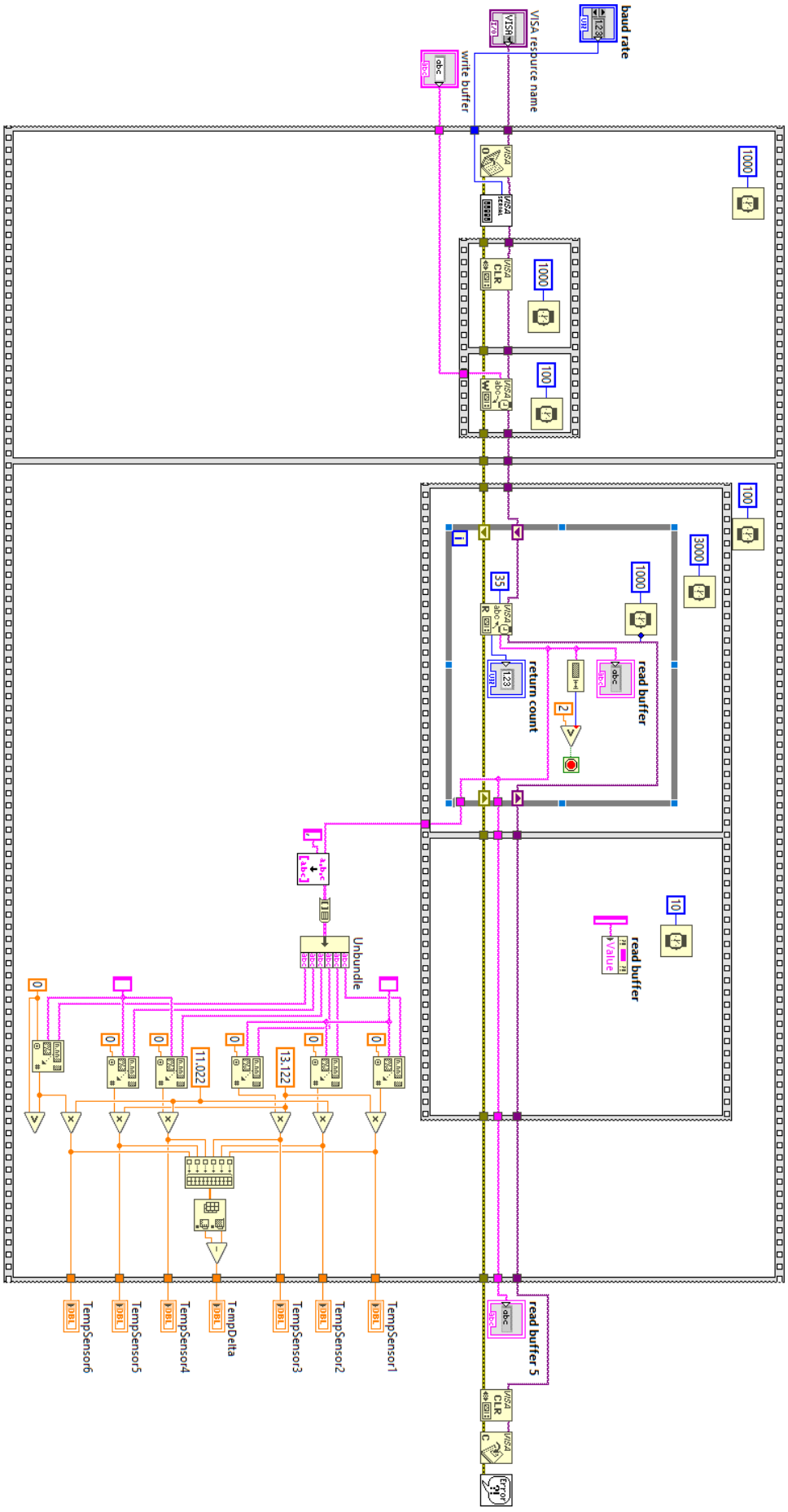
Part 4. Recording voltage levels of all the battery pack cells within the TestStand report (Cell Stack 2 or battery Module 2)

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Voltage Cell Stack2	Passed						

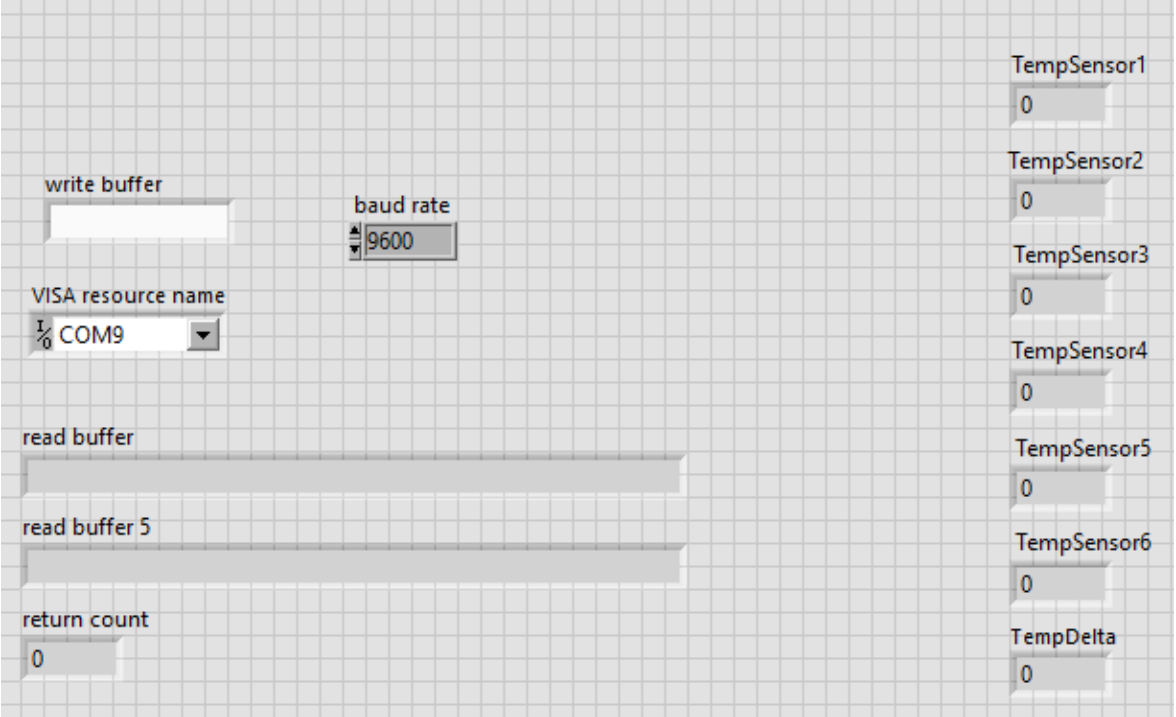
STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Stack2	Passed						
Measurement:							
Cell1	Passed	3.651000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell2	Passed	3.652000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell3	Passed	3.651000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell4	Passed	3.653000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell5	Passed	3.652000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell6	Passed	3.653000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell7	Passed	3.652000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell8	Passed	3.652000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell9	Passed	3.654000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell10	Passed	3.654000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell11	Passed	3.659000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell12	Passed	3.656000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell13	Passed	3.655000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell14	Passed	3.655000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell15	Passed	3.655000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell16	Passed	3.658000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell17	Passed	3.657000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell18	Passed	3.659000	millivolt		3.405000	3.705000	GELE(>= <=)
Cell19	Passed	3.658000	millivolt		3.405000	3.705000	GELE(>= <=)
Maximum Voltage Cell	Passed	3.659000	millivolt		3.405000	3.705000	GELE(>= <=)
Minmum Votlage Cell	Passed	3.651000	millivolt		3.405000	3.705000	GELE(>= <=)
Delta Voltage cell	Passed	0.008000	millivolt		0.000000	30.000000	GELE(>= <=)
Parameters							
VISA resource name.DeviceName:	COM2						
baud rate :	9600						

End Sequence: MainSequence

Part 5. Monitoring temperature sensors of all the battery pack within the LabVIEW environment (Block Diagram)



Part 6. Monitoring temperature sensors of all the battery pack within the LabVIEW environment (Front Panel)



Part 7. Recording temperature sensors of all the battery pack within the TestStand report (Cell Stack 1 or battery Module 1)

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Temprature Sensors Stack1	Passed						

Begin Sequence: MainSequence
D:\OneDrive\Novia Thesis\TestStand\BatteryTempSensors1.seq

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Stack1	Passed						
Measurement:							
Temprature Sensor 1	Passed	20.430954	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 2	Passed	20.368656	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 3	Passed	20.417832	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 4	Passed	20.401722	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 5	Passed	20.378466	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 6	Passed	20.401722	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Delta Temprature Sensors	Passed	0.062298	degrees Celsius		0.000000	3.000000	GELE(>= <=)
Parameters							
VISA resource name.DeviceName:	COM2						

End Sequence: MainSequence

Part 8. Recording temprature sensors of all the battery pack within the TestStand report (Cell Stack 2 or battery Module 2)

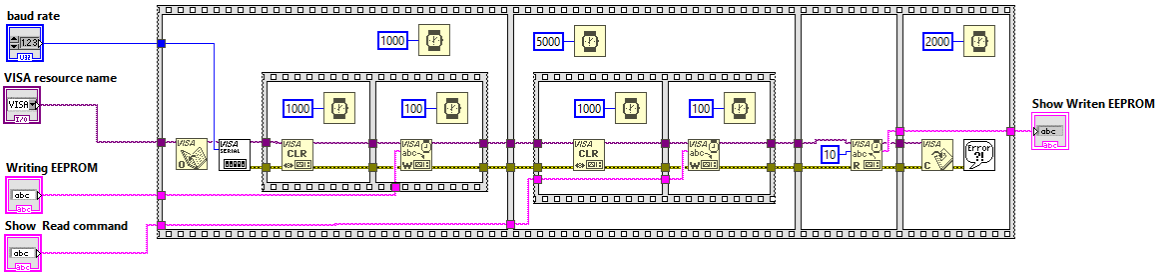
STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Temprature Sensors Stack2	Passed						

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Stack2	Passed						
Measurement:							
Temprature Sensor 1	Passed	20.404710	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 2	Passed	20.368656	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 3	Passed	20.404710	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 4	Passed	20.412744	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 5	Passed	20.417832	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Temprature Sensor 6	Passed	20.368656	degrees Celsius		20.000000	24.000000	GELE(>= <=)
Delta Temprature Sensors	Passed	0.049176	degrees Celsius		0.000000	3.000000	GELE(>= <=)
Parameters							
VISA resource name.DeviceName:	COM2						

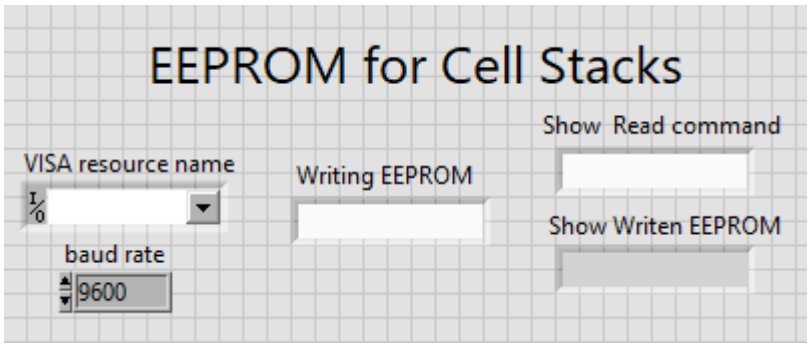
End Sequence: MainSequence

Appendix G. EEPROM Writing/Reading

Part 1. Writing and reading EEPROM data in LabVIEW environment (Block Diagram)



Part 2. Writing and reading EEPROM data in LabVIEW environment (Front Panel)

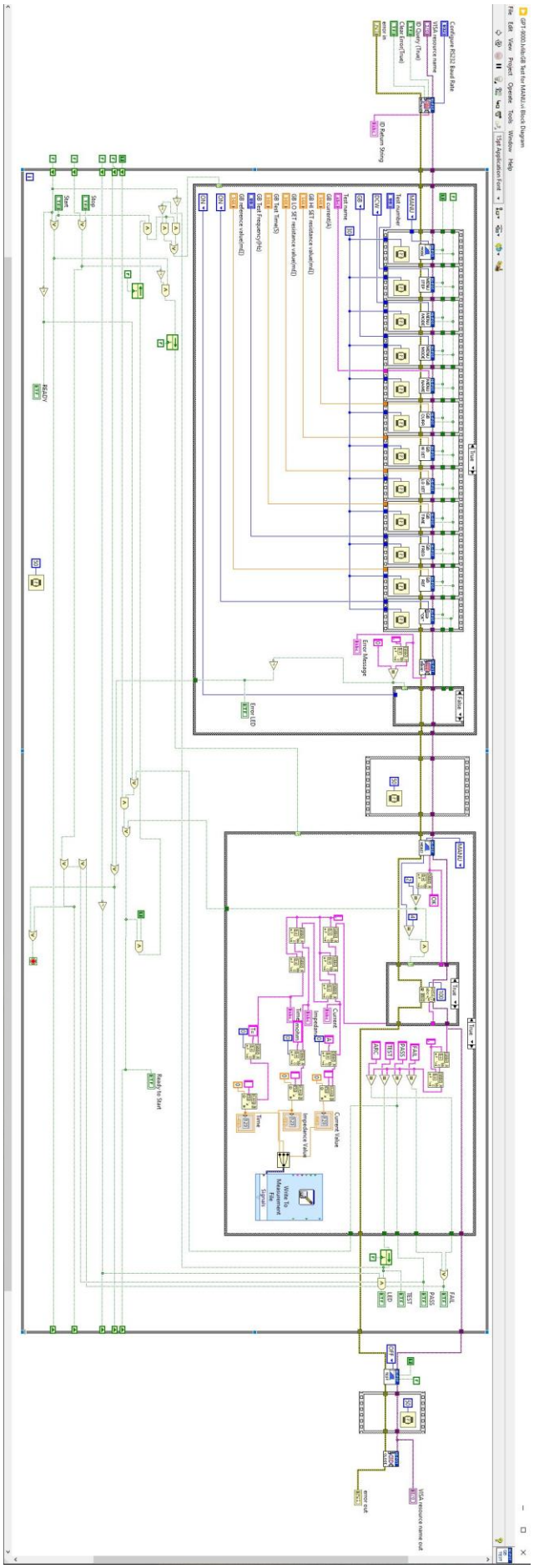


Part 3. Writing and reading EEPROM data in TestStand report

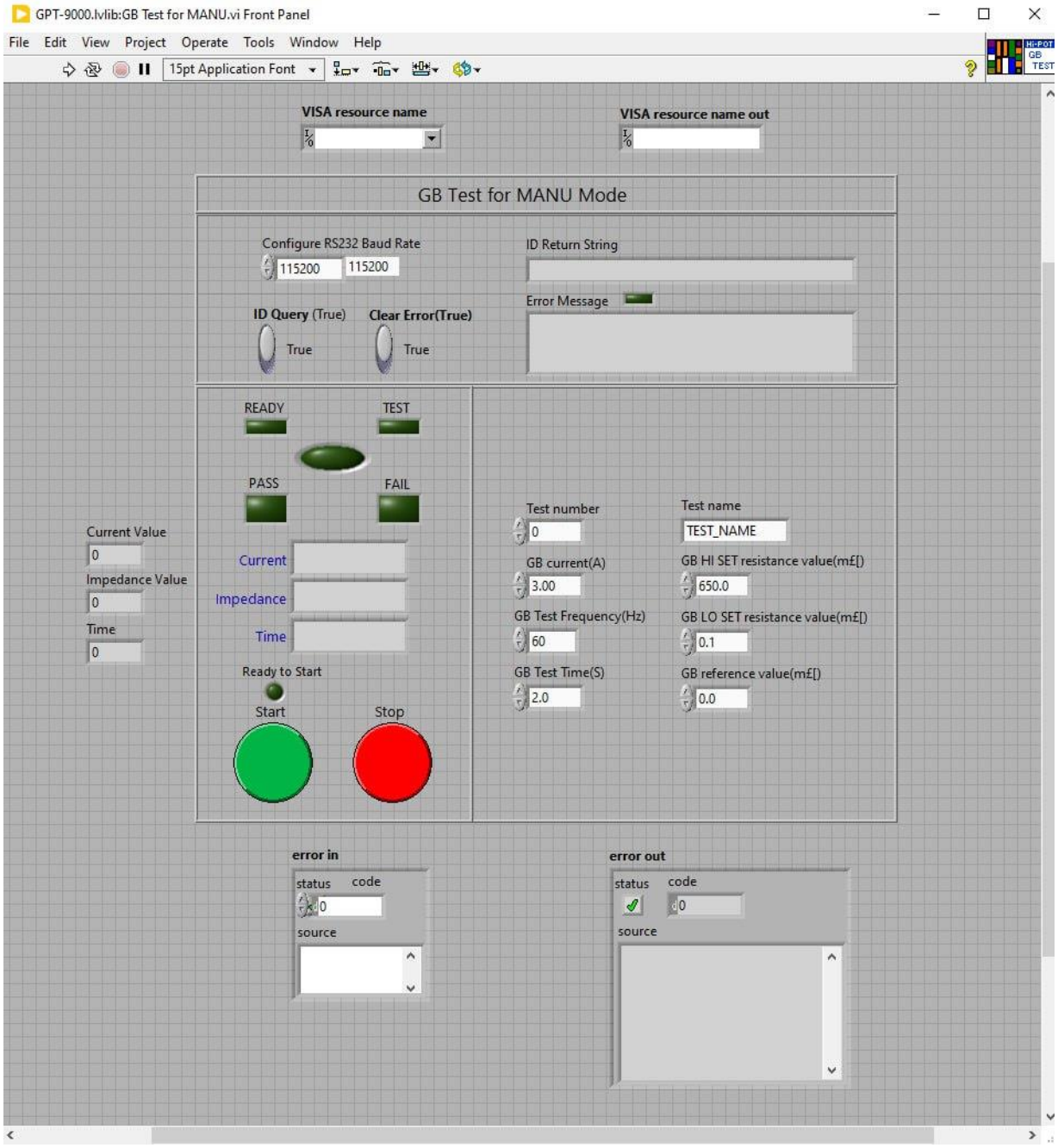
STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Writting EEPROM Stack1	Done						
Parameters							
Writing EEPROM:	A4*PROTO5531						
TestResults/Data							
Show Writen EEPROM:	PROTO5531ÿ						
Report Text:	EEPROM written for Stack1						
Writting EEPROM Stack2	Done						
Parameters							
Writing EEPROM:	B4*PROTO5531						
TestResults/Data							
Show Writen EEPROM:	PROTO5531ÿ						
Report Text:	EEPROM written for Stack2						
Relay1C	Passed						

Appendix H. Equipotential Test

Part 1. Equipotential test VI Block Panel in NI LabVIEW



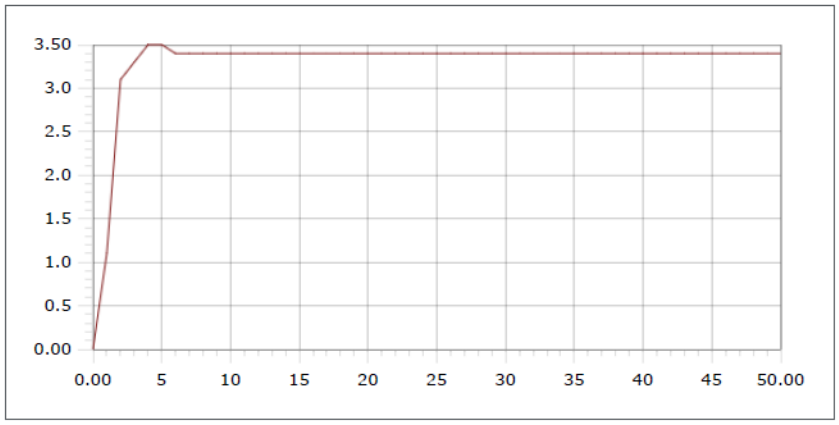
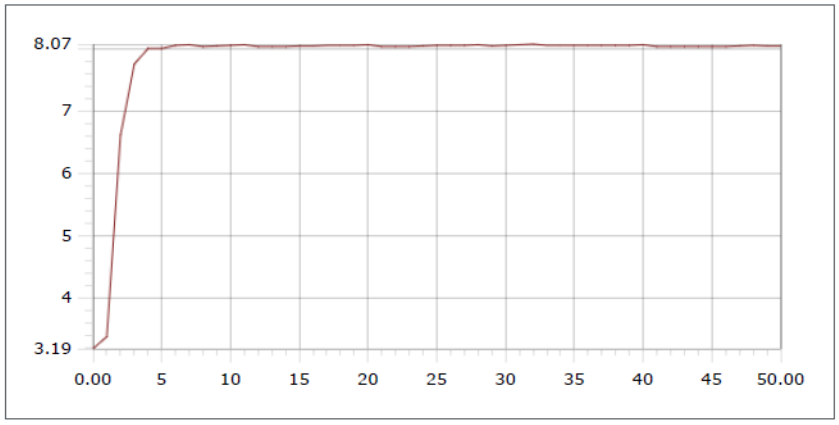
Part 2. Equipotential test VI Front Panel in NI LabVIEW



Part 3. Equipotential test TestStand Report

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Ground Bonding Test	Passed						
Measurement:							
Current Value Applied	Passed	8.040000	Current (A)		0.000000	10.000000	GELE(>= <=)
Impedance Measured	Passed	3.400000	Impedance (m ohm)		0.000000	5.000000	GELE(>= <=)
Time Under Test	Passed	5.000000	Time (Seconds)		0.000000	6.500000	GELE(>= <=)
Parameters							
GB Test Time(S):	5						
GB current(A):	8						
Configure RS232 Baud Rate:	115200						
GB Test Frequency(Hz):	60						

Part 4. Equipotential test TestStand Report [Graph]

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Equipotential Test over Time	Done						
TestResults/Data							
Impedance Graph.Array [0..50] :							
Current Graph.Array [0..50] :							
Report Text:	Impedance unit is milli ohm (m ohm) Current unit is Ampere (A) Time unit is decisecond (ds)						

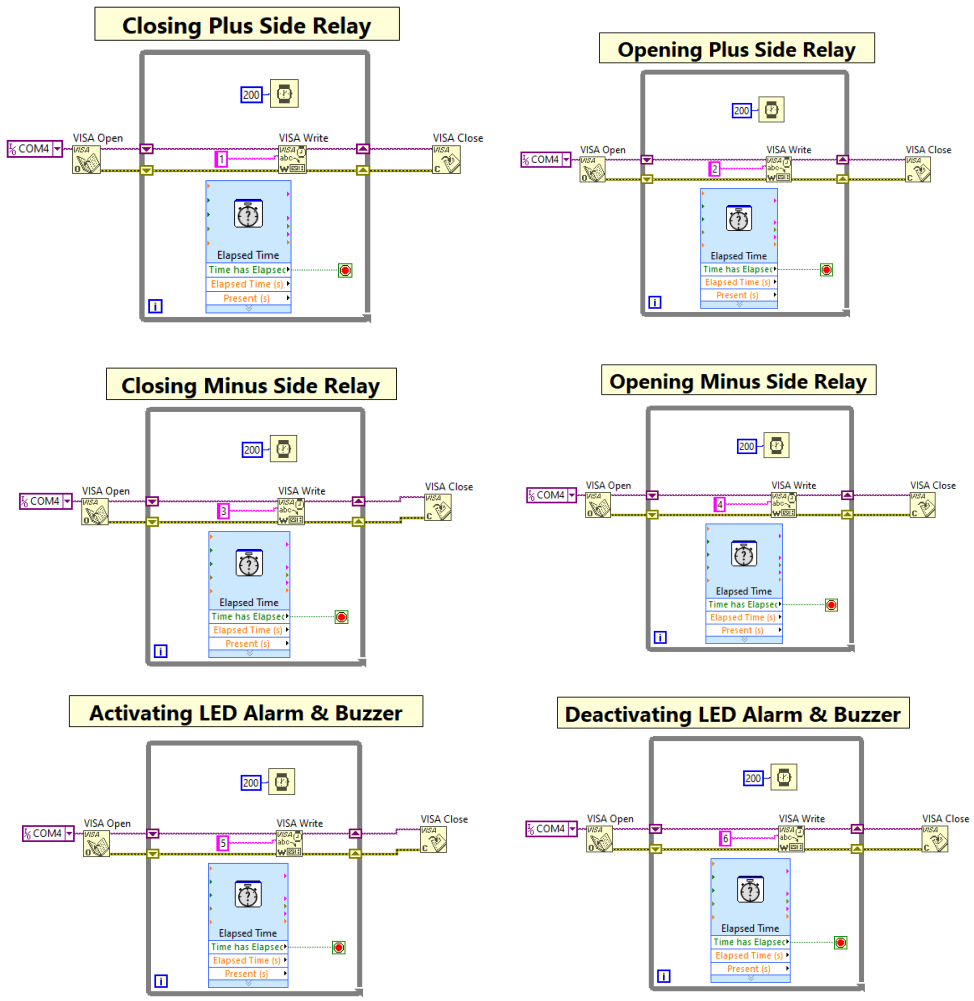
Appendix I. Relay/Alarm/Buzzer Controller

Part 1. Arduino Script for controlling NO relays, Buzzer and Alarm LED

```
//Purpule Signal Wire of Relay signal port connected to Arduino UNO
int RelayPlus = 2; // pin number for NO Relay for plus side of battery
//Dark Blue Signal Wire of Relay signal port connected to Arduino UNO
int RelayMinus = 7; //pin number for NO for Minus side of battery
//Yellow signal wire of Buzzer connected to Arduino UNO
int Buzzer = 5; //pin number for Buzzer alarm
int LED = 4;

char LabVIEW = 0; // Character variable via LabVIEW in Serial Communication

void setup()
{
  Serial.begin(9600);// Baud rate used
  pinMode(RelayPlus, OUTPUT); // output usage for relay as expected
  pinMode(RelayMinus, OUTPUT); // output usage for relay as expected
  pinMode(Buzzer, OUTPUT); // output usage for buzzer as expected
  pinMode(LED, OUTPUT); // output usage for LED Alarm as expected
}

void loop()
 {
  int  Value; // value for buzzer noise level volume
  Value =  100;// value for buzzer noise level volume

  if (Serial.available ()>0) //checking always Serial communicaiton line
  {
    LabVIEW = Serial.read(); //starts reading Serial communication

    if (LabVIEW == '1') // character value of '1' read from serial
communication
    {
      digitalWrite(RelayPlus, HIGH); //closing plus side NO relay
    }
    if (LabVIEW == '2') // character value of '2' read from serial
communication
    {
       digitalWrite(RelayPlus, LOW); //opening plus side NO relay
    }
      if (LabVIEW == '3') // character value of '3' read from serial
communication
    {
      digitalWrite(RelayMinus, HIGH); //closing minus side NO relay
    }
    if (LabVIEW == '4') // character value of '4' read from serial
communication
```

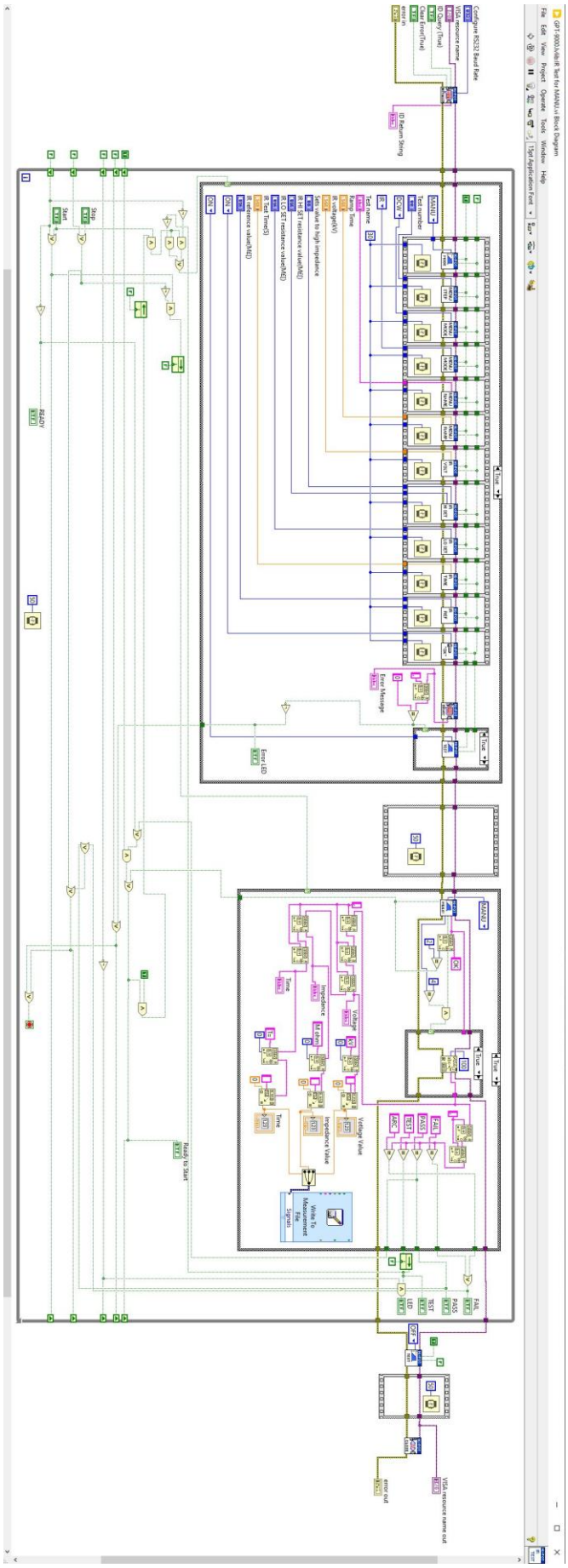
```
    {
       digitalWrite(RelayMinus, LOW); //opening minus side NO relay
    }
     if (LabVIEW == '5')  // character value of '5' read from serial
communication
    {
      analogWrite(Buzzer, Value); // turning on the buzzer
      digitalWrite(LED, HIGH); //turning on the LED Alarm
    }
    if (LabVIEW == '6') // character value of '6' read from serial
communication
    {
      analogWrite(Buzzer, LOW); // turning off the buzzer
      digitalWrite(LED, LOW); //turning off the LED Alarm
    }
  }
}
```

Part 2. Controlling NO relays, Buzzer and Alarm LED VI Block Panel in NI LabVIEW

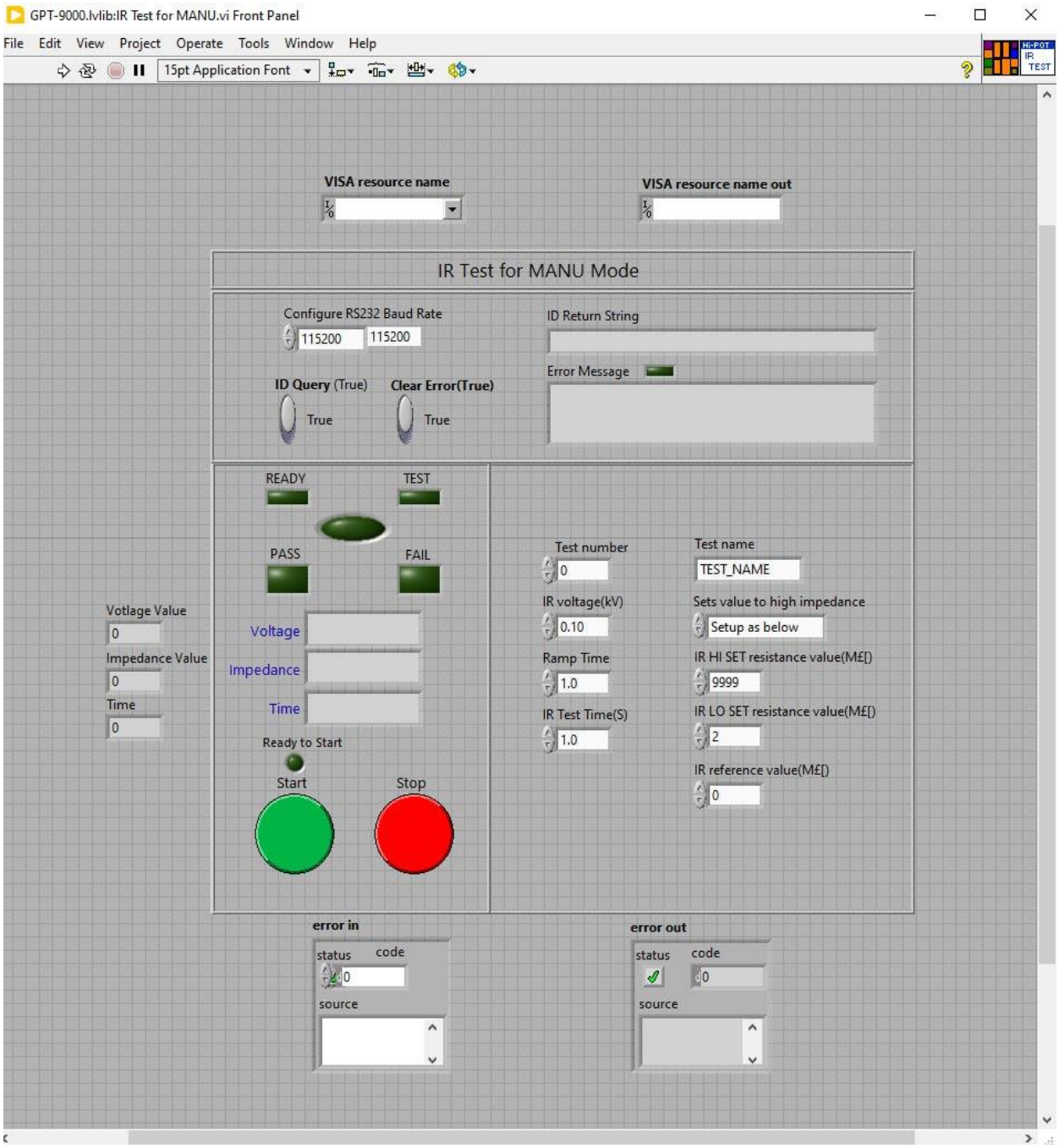


Appendix J. Isolation Test

Part 1. Isolation test VI Block Panel in NI LabVIEW



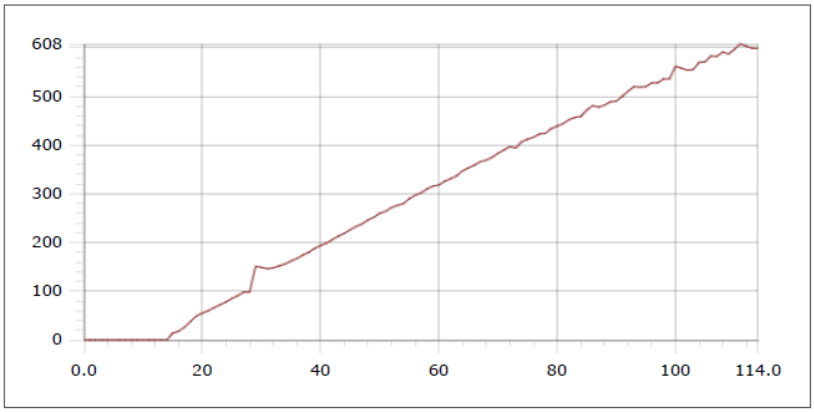
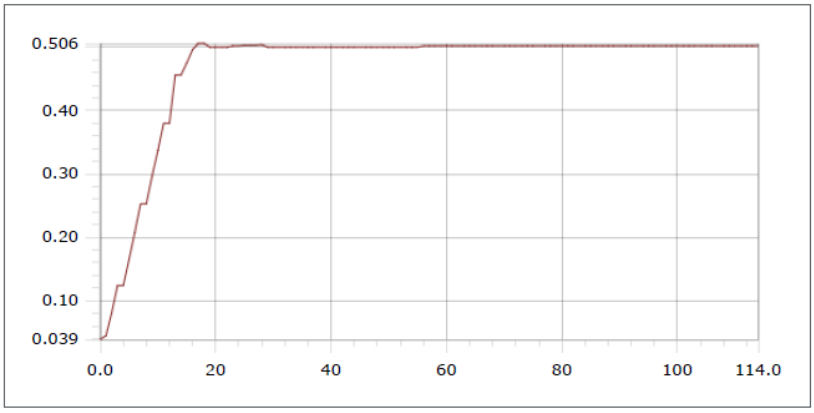
Part 2. Isolation test VI Front Panel in NI LabVIEW



Part 3. Isolation test TestStand Report

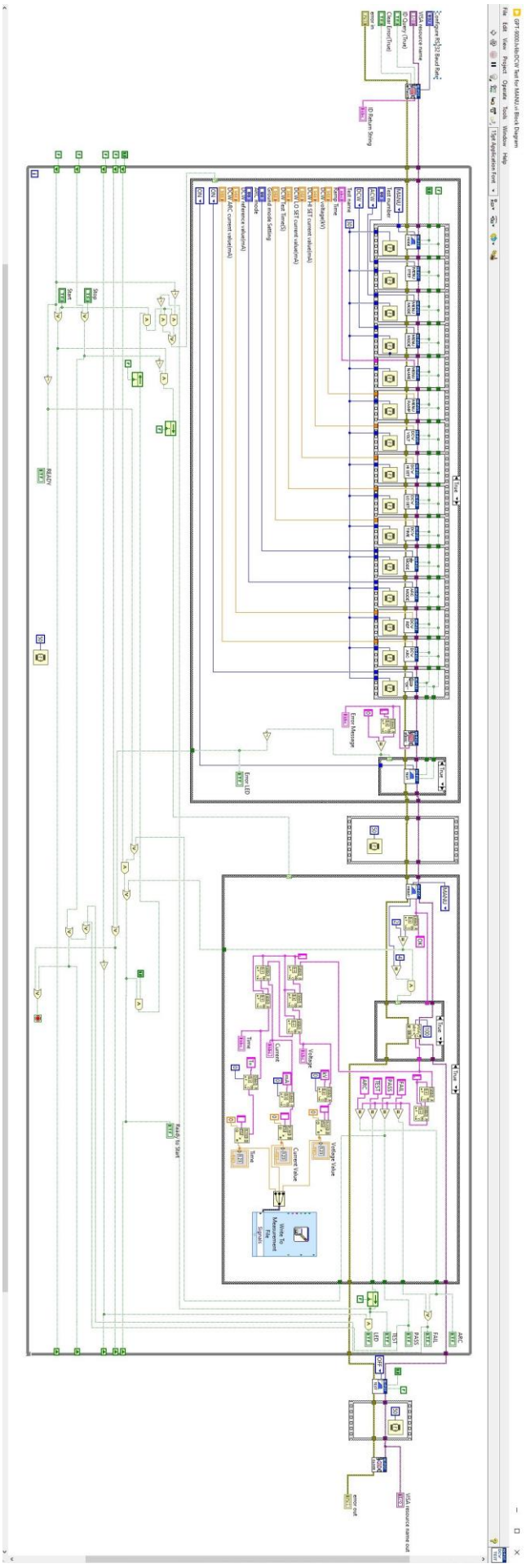
STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Multiple Numeric Limit Test	Passed						
Measurement:							
Voltage Applied	Passed	0.502000	Kilo Volt		0.000000	1.000000	GELE(>= <=)
Time Under Test	Passed	10.000000	second		0.000000	12.000000	GELE(>= <=)
Isolation Resistance	Passed	599.000000	Mega ohm		10.000000	1000.000000	GELE(>= <=)
Parameters							
Configure RS232 Baud Rate:	115200						
VISA resource name.DeviceName:	COM6						

Part 4. Isolation test TestStand Report [Graph]

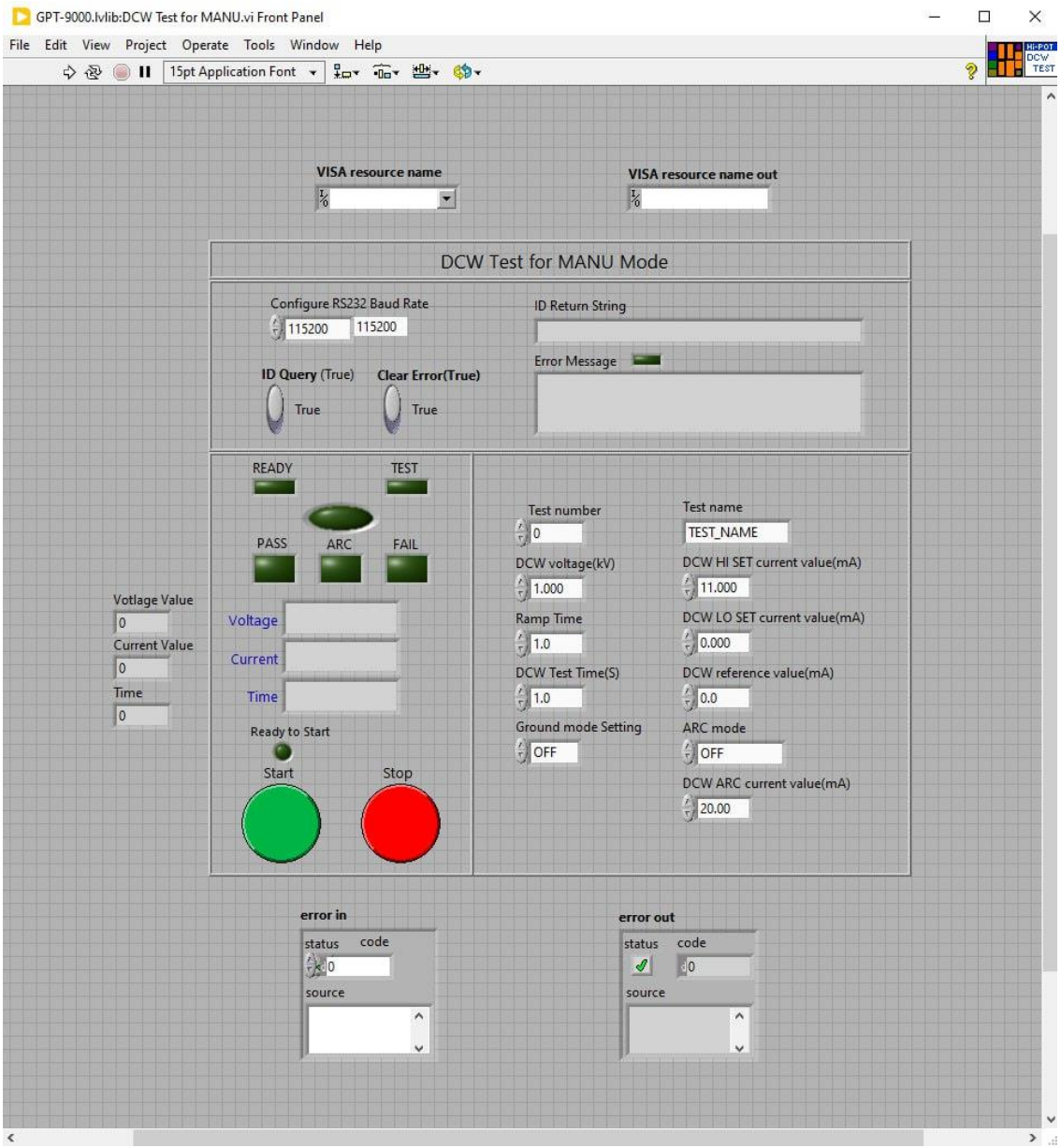
STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Isolation Resistance over Time	Done						
TestResults/Data							
Impedance Graph.Array [0..114] :							
Voltage Graph.Array [0..114] :							
Report Text:	Impedance unit is Mega ohm (M ohm) Voltage unit is Kilo Voltage (kV) Time unit is decisecond (ds)						

Appendix K. Dielectric Test

Part 1. Dielectric test VI Block Panel in NI LabVIEW



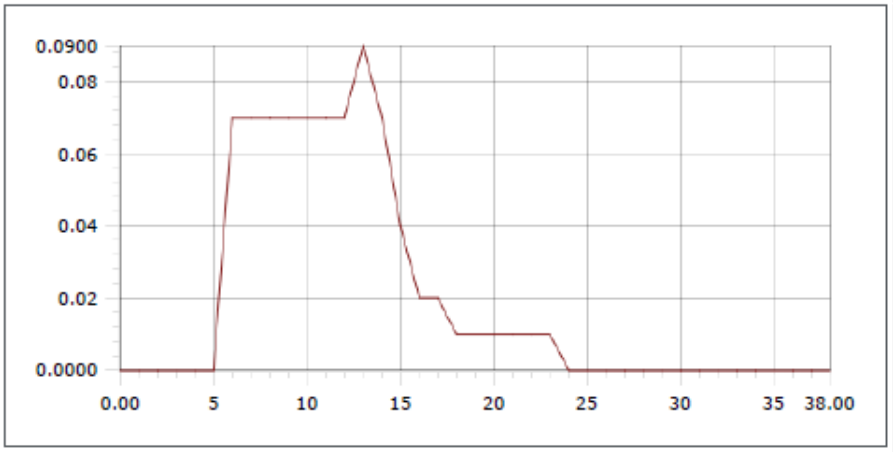
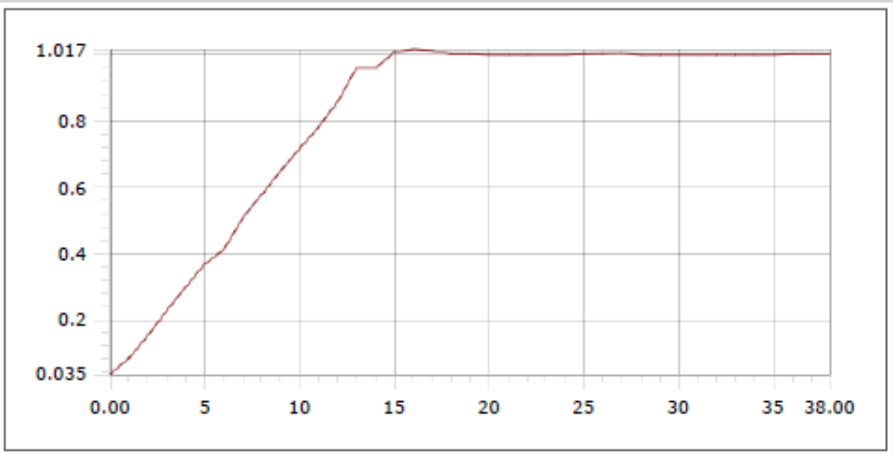
Part 2. Dielectric test VI Front Panel in NI LabVIEW



Part 3. Dielectric test TestStand Report

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
DC Di-electric Test	Passed						
Measurement:							
Measured Voltage Value	Passed	1.000000	Voltage (kV)		0.000000	2.000000	GELE(>= <=)
Measured Time	Passed	2.000000	Time (Seconds)		0.000000	3.000000	GELE(>= <=)
Measured Current Value	Passed	0.000000	mA		0.000000	2.000000	GELE(>= <=)
Parameters							
VISA resource name.DeviceName:	COM6						
Configure RS232 Baud Rate:	115200						

Part 4. Dielectric test TestStand Report [Graph]

STEP	STATUS	MEASUREMENT	UNITS	LIMITS			
				NOMINAL VALUE	LOW LIMIT	HIGH LIMIT	COMPARISON TYPE
Di-Electric Test over Time	Done						
TestResults/Data							
Current Graph.Array [0..38] :							
Voltage Graph.Array [0..38] :							
Report Text:	Current unit is Milli Ampere (mA) Voltage unit is Kilo Voltage (kV) Time unit is decisecond (ds)						