Timur Bodrov, Al-kubaisi Bakr Arif Salman

Development of Automatic 18650 Battery Capacity Tester

Metropolia University of Applied Sciences

Bachelor of Engineering

DP in Electronics

Bachelor's Thesis

December 2024

Abstract

Author: Al-kubaisi Bakr Arif Salman, Timur Bodrov

Title: Development of Automatic 18650 Battery Capacity Tester

Number of Pages: 52 page + 2 appendices

Date: 19 May 2025

Degree: Bachelor of Engineering

Degree Programme: DP in Electronics

Professional Major: Electronics

Supervisors: Tuan Nguyen, Senior Lecturer

This thesis presents the design and implementation of an automatic 18650 battery capacity tester, offering a cost-effective and precise solution for evaluating battery performance. The project integrates theoretical research on lithium-ion batteries with practical engineering to develop a system featuring adaptive constant current (CC) discharge algorithms, internal resistance (IR) testing, real-time thermal monitoring, and serial communication for data visualization. Key innovations include a smart adaptive discharge mechanism that dynamically adjusts current based on voltage feedback, ensuring high measurement accuracy, and a DC load method for IR measurement with ±1.5% error margins.

The methodology involved rigorous testing of the system using Samsung and Sony 18650 cells across 50 charge-discharge cycles. Results demonstrated a voltage measurement accuracy of ±0.07V, IR consistency within ±1.4mΩ, and a total system cost under €50, outperforming commercial testers in functionality and affordability. The open-source design, modular firmware, and compatibility with multiple battery chemistries make this tester a scalable tool for hobbyists, researchers, and recyclers. By bridging the gap between academic research and real-world applications, the project delivers a reliable and accessible solution for comprehensive battery diagnostics.

Keywords: Charging, Discharging, 18650 batteries, Battery capacity tester, adaptive discharge, open-source hardware.

Contents

List of Abbreviations

Ah: Ampere-hour. A unit of electric charge, representing the amount of current a battery can supply over one hour.

CC: Constant Current. A method of discharging a battery at a fixed current until it reaches a predefined cutoff voltage.

CP: Constant Power. A method of discharging a battery at a fixed power level, requiring the current to vary as the battery voltage changes.

CR: Constant Resistance. A method of discharging a battery using a fixed resistive load, causing the current to decrease as the voltage drops.

DoD: Depth of Discharge. The percentage of the battery's capacity that has been discharged relative to its total capacity.

IEC: International Electrotechnical Commission. An international standards organization that prepares and publishes standards for electrical, electronic, and related technologies.

$LiCoO_2$: Lithium Cobalt Oxide. A common cathode material used in lithium-ion batteries, known for its high energy density.

$LiFePO_4$: Lithium Iron Phosphate. A cathode material used in lithium-ion batteries, known for its thermal stability and long cycle life.

$LiMn_2O_4$: Lithium Manganese Oxide. A cathode material used in lithium-ion batteries, known for its safety and thermal stability.

mAh: Milliampere-hour. A unit of electric charge, representing one-thousandth of an ampere-hour.

MOSFET: Metal-Oxide-Semiconductor Field-Effect Transistor. A type of transistor used for switching or amplifying electronic signals.

NCA: Lithium Nickel Cobalt Aluminum Oxide. A cathode material used in lithium-ion batteries, known for its high energy density and long lifespan.

NMC: Lithium Nickel Manganese Cobalt Oxide. A cathode material used in lithium-ion batteries, offering a balanced performance in terms of energy density, thermal stability, and lifespan.

OCV: Open-Circuit Voltage. The voltage of a battery when it is not under load, used to estimate the state of charge (SoC).

Op-Amp: Operational Amplifier. A high-gain electronic voltage amplifier used in analogue circuits, including the constant current load circuit in this design.

SoC: State of Charge. The percentage of the battery's remaining capacity is relative to its total capacity.

Wh: Watt-hour. A unit of energy, representing the amount of energy a battery can store and deliver.

SoH: State of Health. A measure of a battery's condition compared to its ideal state, often expressed as a percentage.

BMS: Battery Management System. An electronic system that monitors and manages the performance and safety of a battery.

ADC: Analog-to-Digital Converter. A device that converts analog signals (e.g., voltage) into digital values for processing.

PWM: Pulse-Width Modulation. A technique used to control the amount of power delivered to a load by varying the width of pulses.

IoT: Internet of Things. A network of interconnected devices that communicate and exchange data.

NTC: Negative Temperature Coefficient. A type of thermistor whose resistance decreases as temperature increases.

LED: Light-Emitting Diode. A semiconductor device that emits light when an electric current passes through it.

PCB: Printed Circuit Board. A board is used to mechanically support and electrically connect electronic components.

STL: Standard Tessellation Language. A file format commonly used for 3D modelling and printing.

1 Introduction

The rapid advancement of portable electronics, electric vehicles, and renewable energy systems has placed lithium-ion batteries, particularly the 18650 cell format, at the forefront of modern energy storage solutions. The 18650 batteries, with its standardized cylindrical form factor, has become a cornerstone in applications requiring high energy density, reliability, and scalability. However, the performance of these batteries is heavily dependent on their capacity, which degrades over time due to several factors such as charge-discharge cycles, temperature variations, and aging. Accurate measurement of battery capacity is therefore critical for assessing battery health, predicting lifespan, and ensuring optimal performance in real-world applications.

This thesis project focuses on the development of a reliable and automatic 18650 battery capacity tester targeting lithium-ion batteries. The project is structured around three primary objectives:

- Studying the theoretical framework of modern batteries and their capacity measurement techniques.
- Developing a prototype system capable of performing automated capacity measurements.
- Conducting a series of tests to verify the system's reliability and analyse the real capacity of various 18650 batteries.

The goal of this thesis project was to provide a robust, cost-effective safe solution for battery capacity testing, which can be utilized in both industrial and research settings. By combining theoretical knowledge with practical engineering, this project aimed to bridge the gap between academic research and real-world applications in battery technology.

2 Background

Lithium-ion (Li-ion) batteries have become the cornerstone of modern energy storage systems, powering everything from portable electronics to electric vehicles and renewable energy applications. Their widespread adoption is driven by three key advantages: high energy density, long cycle life, and low self-discharge rates. To design an accurate battery capacity tester, it is essential to first understand the fundamental principles governing these batteries, including their chemistry, structure, and operational mechanisms.

The following sections delve into the electrochemistry of Li-ion batteries, the materials used in their construction, and the unique characteristics of the 18650 cell format. This foundation will clarify how capacity is defined, measured, and influenced by factors such as discharge rates and environmental conditions, topics explored in detail later in this thesis.

Battery Chemistry and Electrochemistry

Lithium-ion batteries operate based on the principles of electrochemistry, involving the movement of lithium ions [1] (Li^+) between a negative electrode (anode) and a positive electrode (cathode) through an electrolyte. As illustrated in Figure 1, during discharge, lithium ions migrate from the anode, typically composed of graphite (Li_xC_6), through the electrolyte and separator toward the cathode, commonly made of lithium metal oxides such as lithium manganese oxide (Li_yMnO_2). This ion movement is accompanied by the flow of electrons through an external circuit, powering electronic devices.

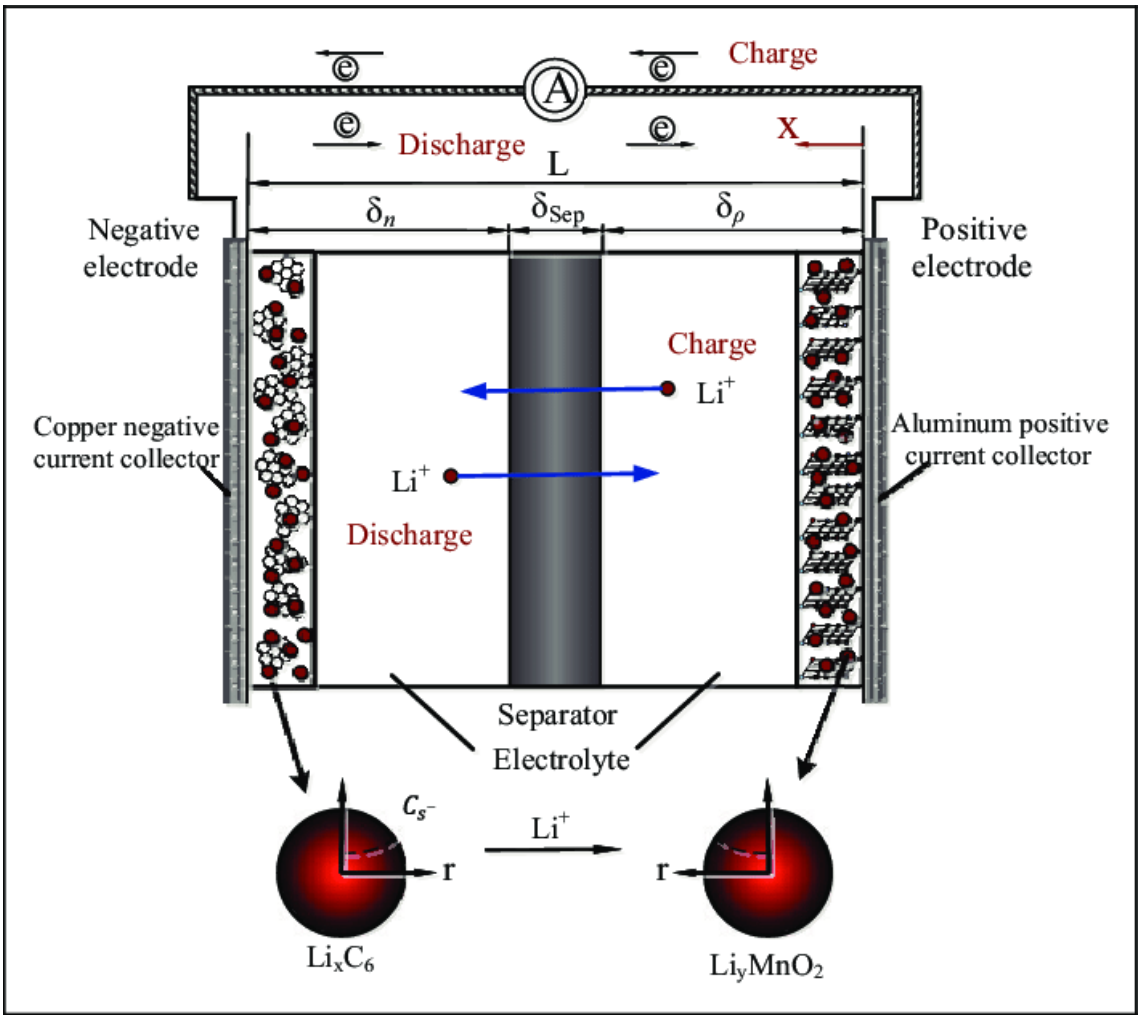

Figure 1. Basic principle of a lithium-ion battery [21].

The schematic in Figure 1 shows the migration of lithium ions and flow of electrons during charge and discharge, along with the structural components including the negative and positive electrodes, current collector, separator, and electrolyte.

Conversely, during charging, an external power source drives the lithium ions back from the cathode to the anode, effectively restoring energy into the battery. The direction of both ionic and electronic flows during these processes is clearly depicted in Figure 1, where the lithium-ion transport and electron flow pathways are labelled accordingly for charge and discharge cycles.

The anode uses a copper current collector, while the cathode is connected to an aluminium current collector, both aiding in efficient electron conduction. Between the electrodes lies the separator, a porous membrane soaked in electrolyte, which permits ion flow but blocks electron transfer, thereby preventing internal short circuits.

The electrochemical mechanism hinges on the intercalation (insertion) and de-intercalation (extraction) of lithium ions into and from the crystal structures of the electrode materials. This reversible process enables Li-ion batteries to undergo numerous charge and discharge cycles with minimal degradation. As illustrated at the bottom of Figure 1, lithium ions are stored in the layered structure of graphite during charge and released during discharge, while the opposite occurs in the cathode material.

Anode and Cathode Materials

The choice of materials for the anode and cathode significantly impacts the performance, safety, and lifespan of lithium-ion batteries. The anode is typically composed of graphite, which provides a stable and reversible structure for lithium-ion intercalation. With a high theoretical capacity of 372 mAh/g and excellent cycling stability, graphite remains the dominant anode material. However, alternatives such as silicon have attracted significant attention due to their much higher theoretical capacity, up to 4200 mAh/g. The key challenge with silicon lies in its substantial volume expansion (up to 300%) during lithium intercalation, which can result in mechanical degradation and shortened cycle life. As a result, silicon is often blended with graphite or used in composite form to mitigate these issues. [19]

On the cathode side, several lithium metal oxides are widely used, each offering different performance characteristics. These include lithium cobalt oxide ($LiCoO_2$), lithium manganese oxide ($LiMn_2O_4$), lithium nickel oxide ($LiNiO_2$), and lithium iron phosphate ($LiFePO_4$). As illustrated in Figure 2, these materials vary significantly across critical parameters such as volumetric and

gravimetric energy and power density, density, thermal stability, and capacity retention after cycling. [2]

Lithium Cobalt Oxide ($LiCoO_2$) offers high volumetric energy density and is favoured in compact consumer electronics like smartphones and laptops. However, as shown in Figure 2, it has relatively poor thermal stability and shows noticeable capacity loss after 100 cycles at 1C. The high cost and scarcity of cobalt are also limiting factors. [2;4]

Lithium Manganese Oxide ($LiMn_2O_4$) exhibits good thermal stability and safety, with moderate power density, making it suitable for power tools and medical devices. Nonetheless, it has lower energy density and more pronounced degradation over repeated cycling, as reflected in its capacity loss and decomposition temperature values. [2]

Lithium Iron Phosphate ($LiFePO_4$) stands out in Figure 2 for its exceptional thermal stability and minimal capacity loss over 100 cycles, indicating excellent long-term performance and safety. It is often used in electric vehicles and stationary energy storage systems, though its lower energy density compared to other cathodes limits its suitability for applications where space and weight are crucial. [4]

Lithium Nickel Oxide ($LiNiO_2$) despite limited commercial adoption, it demonstrates high gravimetric energy and power density, but with lower thermal stability and incomplete data for some performance categories in Figure 2. Its performance trade-offs make it less favourable without further material enhancements or stabilizers. [4]

In addition, advanced materials such as NMC (Nickel Manganese Cobalt Oxide) and NCA (Nickel Cobalt Aluminium Oxide) offer balanced properties across energy density, thermal stability, and cycle life. NMC is highly Optimizable based on the ratios of its constituents, while NCA is used in high-

performance applications like electric vehicles, albeit at a higher cost and lower safety margin. [7]

Ultimately, the selection of cathode material represents a balance between competing priorities-energy density, thermal safety, material cost, and cycle life, as clearly visualized in Figure 2. For instance, $LiFePO_4$ may not deliver the highest energy density, but its safety and stability make it ideal for large-format cells, while $LiCoO_2$ remains the go-to for compact devices demanding energy-dense solutions.

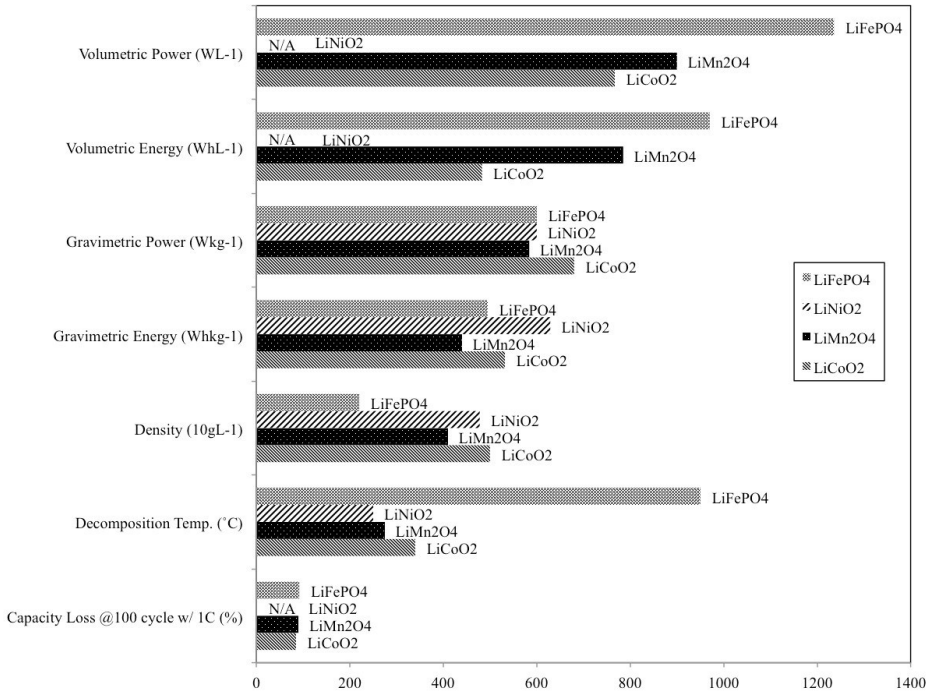

Figure 2. A comparison of LiCoO2, LiMn2O4, LiNiO2, and LiFePO4 cathode materials by important properties (from rear to front): Volumetric power/energy density, gravimetric power/energy density, material density, decomposition temperature, and capacity retention after 100 cycles at 1C-rate [22].

The 18650 cell is one of the most widely adopted lithium-ion battery, named for its dimensions: 18 mm in diameter and 65 mm in length. This cylindrical form factor is valued for its mechanical robustness [1;11], efficient thermal management, and ease of integration into larger battery packs. Its design also

facilitates automated manufacturing, contributing to its prevalence in applications ranging from consumer electronics to electric vehicles.

The nominal voltage of an 18650 cell typically ranges from 3.6V to 3.7V, with a fully charged voltage of 4.2V and a discharge cutoff between 2.5V and 3.0V. Depending on the chemistry and manufacturer, these cells can provide capacities ranging from 1200mAh to 3600mAh. High-drain variants are capable of sustaining discharge currents of up to 30A, making them suitable for demanding applications such as power tools, electric bikes, and automotive battery modules. [1;11]

As shown in Figure 3, the 18650 cell consists of several critical internal components, each contributing to the electrochemical and mechanical Functionality of the battery. The anode, typically composed of graphite (Li_xC_6), serves as the host for lithium ions during charging, appearing as brownish layers in the spiral-wound structure (Figure 3). During discharge, lithium ions migrate from the anode to the cathode, which is composed of lithium metal oxides (e.g., $LiCoO_2$, $LiFePO_4$) and acts as the lithium source. The cathode is depicted as blue layers in the internal configuration [1;2]. A thin, porous separator (grey in Figure 3) prevents electrical short circuits while enabling ionic flow between the electrodes. The electrolyte, though not visible in the figure, consists of a lithium salt (e.g., $LiPF_6$) dissolved in an organic solvent, facilitating ion transport [8;12].

The 18650 cell's mechanical integrity is ensured by a stainless-steel shell, which functions as the negative terminal and provides structural support. At the positive pole, a nickel-plated steel cap integrates an aluminium safety valve and insulating gasket to mitigate risks of overpressure and thermal runaway [1;17]. This multi-layered design balances electrochemical performance with safety, critical for high-density energy storage applications [9;13].

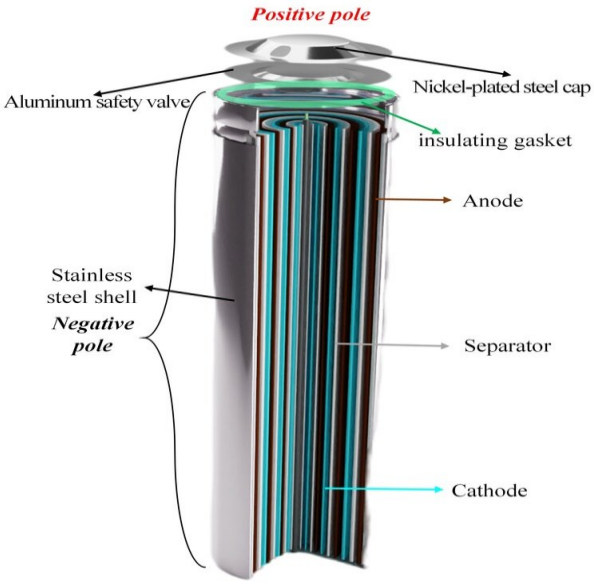

Figure 3. Structure and Characteristics of 18650 Cells [23].

The electrical specifications of 18650 cells include

- Nominal Voltage: 3.6V to 3.7V.
- Fully Charged Voltage: 4.2V.
- Discharge Cutoff Voltage: 2.5V to 3.0V.
- Capacity: 1200 mAh to 3600 mAh, depending on the chemistry and manufacturer.
- Energy Density: Typically, around 200-250 Wh/kg, making them highly efficient for their size and weight.

The thermal management of 18650 cells is also a critical consideration. During high-current discharges, the cells can generate significant heat, which can lead to thermal runaway if not properly managed. To mitigate this risk, 18650 cells are often equipped with safety features such as pressure relief vents and thermal fuses. Additionally, battery packs made from 18650 cells often include thermal management systems, such as heat sinks or liquid cooling, to maintain safe operating temperatures.

2.1 Charge and Discharge Mechanisms

The fundamental operating principle of lithium-ion (Li-ion) batteries is based on the intercalation and de-intercalation of lithium ions within the crystal lattice of electrode materials. During discharge, lithium ions migrate from the anode to the cathode through the electrolyte, while electrons travel through the external circuit to provide electric power. The reverse occurs during charging, lithium ions move back from the cathode to the anode, restoring the battery's potential. This intercalation process is highly reversible, allowing for hundreds to thousands of charge-discharge cycles with minimal structural degradation under optimal conditions [2;6] as illustrated in Figure 4.

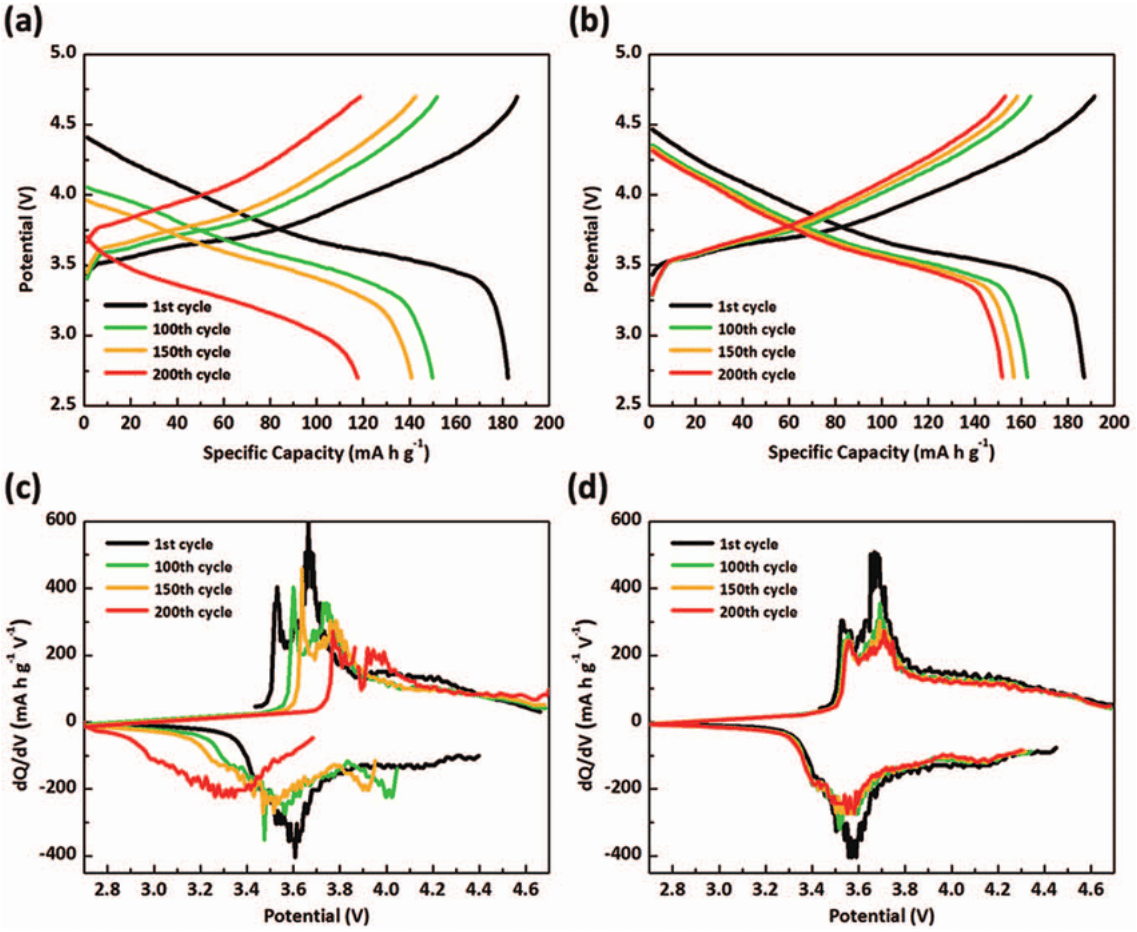

Figure 4. Charge-discharge curves for lithium-ion batteries with different electrolyte systems; (a) EC/EMC (1/9)-based cell, and (b) FEMC/FEMC (1/9)-based cell; as well as plots for differential capacity vs. voltage for (c) EC/EMC (1/9)-based cell (d) FEMC/FEMC (1/9)-based cell at C/2 rate [20].

The voltage profile during discharge typically features a relatively flat plateau, which ensures consistent voltage output over most of the cycle. This plateau is followed by a sharp voltage drop as the battery nears its cutoff voltage, signalling a depleted state. This behaviour is evident across both EC/EMC and FEMC/FEMC-based electrolyte systems (parts a and b of Figure 4), where subtle variations in voltage profile reflect the impact of electrolyte composition on electrochemical performance.

The open-circuit voltage (OCV), or the voltage measured when the battery is at rest (no current flow), is strongly correlated with the state of charge (SoC). This relationship is often used in battery management systems to estimate the remaining capacity. However, under dynamic load conditions, this method can lose accuracy due to fluctuations caused by internal resistance, temperature effects, and electrolyte kinetics.

Figures 4c and 4d provide differential capacity vs. voltage plots, which reveal additional insight into the electrochemical reactions during cycling. These plots show how the electrolyte system can influence phase transitions, reaction kinetics, and overall capacity behaviour. Peaks in these graphs correspond to specific redox reactions occurring within the electrodes. Comparing EC/EMC and FEMC/FEMC systems at a C/2 rate highlights differences in reaction efficiency and voltage stability.

In summary, the charge-discharge process in Li-ion batteries involves the reversible transport of lithium ions between the anode and cathode. The near-constant voltage during discharge is a key advantage, providing stable power delivery, while variations in voltage behaviour analysed via differential plots can offer detailed insights into cell chemistry and electrolyte performance

2.2 Battery Capacity and Performance Metrics

The performance and longevity of lithium-ion batteries are fundamentally governed by their capacity characteristics and associated metrics. Capacity,

representing the total charge a battery can store and deliver, serves as the primary indicator of energy storage capability. However, this property is intrinsically linked to other critical performance parameters including C-rate capability, energy efficiency, and cycle life [1;12]. These interdependent factors collectively determine a battery's suitability for specific applications, ranging from consumer electronics to electric vehicles [9;13].

As battery systems evolve, standardized methodologies for capacity measurement and performance evaluation have become essential for both industry and research. The International Electrotechnical Commission (IEC) has established testing protocols (IEC 61960) to ensure consistent capacity measurements under controlled conditions [3]. These standards account for the complex relationship between a battery's theoretical capacity and its practical, usable capacity, which is influenced by operational parameters such as temperature, discharge rate, and aging effects [13;16].

2.2.1 Definition of Battery Capacity

Battery capacity refers to the total amount of electric charge a battery can store and deliver over time, and it is typically expressed in ampere-hours (Ah) or milliampere-hours (mAh). This unit reflects the current a battery can supply over a certain duration. For instance, a battery rated in 2000mAh can theoretically deliver 2000 milliamps for one hour, or 1000 milliamps for two hours, depending on the load conditions [1;9].

In contrast, energy capacity is expressed in watt-hours (Wh), a measure of the total energy stored in the battery. It is calculated as the product of the battery's capacity (in Ah) and its nominal voltage (V). For example: a 2000mAh (or 2 Ah) battery with a nominal voltage of 3.7V yields. Energy = 2 Ah × 3.7 V = 7.4Wh

Another essential metric is the C-rate, which describes the rate at which a battery is charged or discharged relative to its maximum capacity. A 1C rate means the battery is discharged in one hour (i.e., a 2000mAh battery

discharges at 2000 mA). A 0.5C rate discharges the battery over two hours (at 1000 mA), while a 2C rate completes the discharge in 30 minutes (at 4000 mA).

The C-rate significantly affects the voltage behaviour and usable capacity of a lithium-ion battery, as shown in Figure 5. The discharge curves demonstrate that higher C-rates (faster discharge rates) result in a steeper voltage drop and shorter discharge durations. This is largely due to increased internal resistance, which generates more heat and reduces efficiency under higher currents.

At low C-rates, the battery discharges more gradually, maintaining a higher average voltage and yielding more usable capacity. Conversely, at high C-rates, the voltage declines more quickly, and the battery reaches its cutoff voltage sooner, which can limit the delivered energy despite the same nominal capacity.

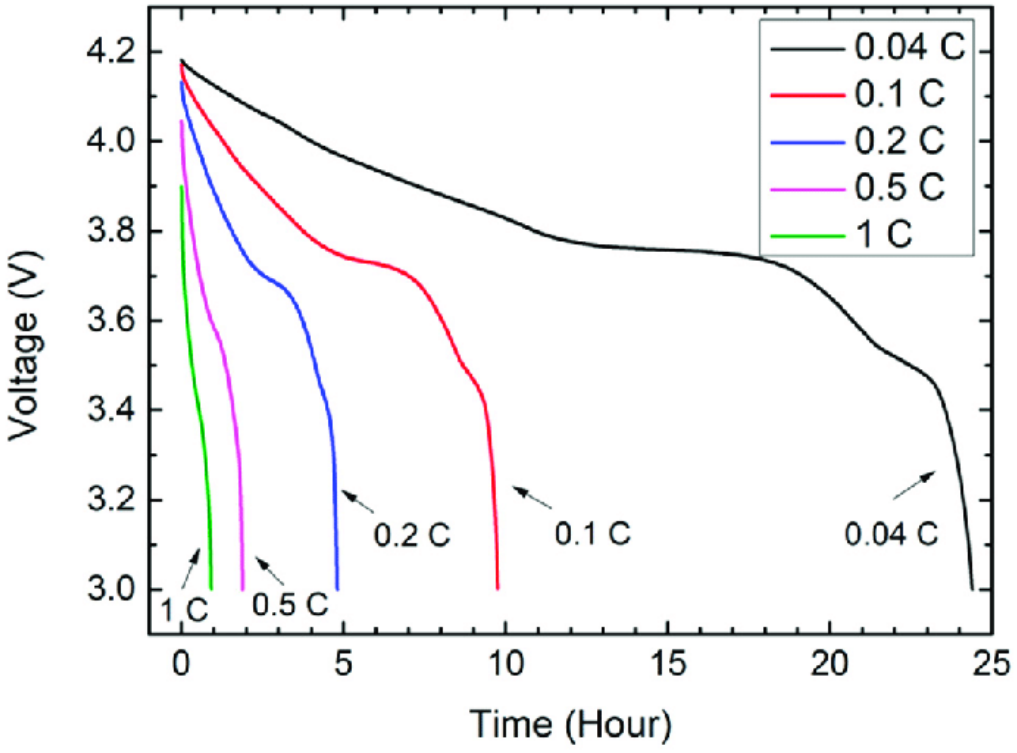

Figure 5. Typical discharge curves of lithium-ion battery as a function of C-rate [4].

2.2.2 Factors Affecting Battery Capacity

The capacity and performance of lithium-ion batteries are influenced by several key factors, including discharge rate (C-rate), operating temperature, and ageing mechanisms. These factors affect not only the immediate energy output of the battery but also its long-term reliability and lifespan.

- The C-rate (Discharge Rate) plays a crucial role in battery performance. It defines how quickly a battery is discharged relative to its rated capacity. A higher C-rate implies a faster discharge, which often leads to increased internal resistance and greater heat generation, ultimately reducing the effective capacity.

  For instance, discharging a battery at 2C (twice its rated capacity per hour) results in lower total energy output than discharging at 1C, as a portion of the energy is lost as heat. This is clearly demonstrated in Figure 4, where faster discharge rates correlate with shorter operational time and decreased efficiency.

- Temperature significantly affects the electrochemical performance of Li-ion batteries. At high temperatures, internal chemical reactions accelerate, which can enhance short-term performance but also speed up degradation, reducing the battery's cycle life and safety. Conversely, low temperatures increase the battery’s internal resistance, restricting the flow of lithium ions and thereby lowering capacity and power output. As depicted in Figure 6, a lithium-ion battery operating at -20°C may deliver only around 50% of its rated capacity compared to performance at room temperature. Extreme temperatures, both hot and cold, compound degradation effects as shown in Figure 6.

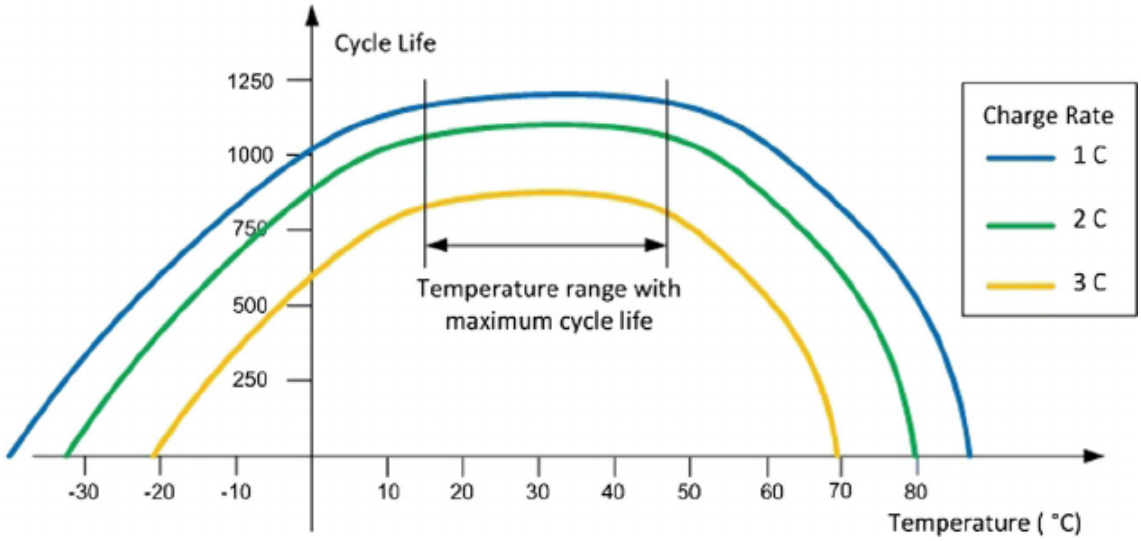

Figure 6. Lithium-ion battery life vs. temperature and charging rate showing the maximum cycle life of lithium-ion battery is between 15-45°C [8].

- Over time, lithium-ion batteries undergo ageing, primarily due to the repeated charge-discharge cycles that deteriorate the electrode materials.

- Batteries that are regularly discharged to 100% DoD degrade faster than those with shallower discharges (e.g., 50% DoD).

- Fast charging and discharging at high currents stress the electrode structure, accelerating wear and tear.

2.3 Methods of Battery Capacity Measurement

The accurate determination of battery capacity is critical for performance evaluation, state-of-health assessment, and system design. While capacity can be theoretically derived from electrode material properties, practical measurement requires controlled discharge under standardized conditions to account for real-world operational factors [3;12]. Several methodologies have been developed to quantify capacity, each with specific advantages and limitations depending on application requirements. These methods primarily

differ in their discharge protocols, measurement precision, and compatibility with different battery chemistries [1;14]. The most widely adopted approaches, standardized by organizations such as the International Electrotechnical Commission (IEC), utilize controlled discharge techniques to establish reproducible capacity metrics [3].

2.3.1 Overview of Discharge Methods

Battery capacity can be measured using several standardized discharge methods, each offering varying levels of accuracy and relevance depending on the application. Among these, the Constant Current (CC) discharge method is the most widely used. Some of the widely used methods are discussed below.

In the Constant Current (CC) Discharge Method, the battery is discharged at a fixed current until it reaches a predefined cutoff voltage. The capacity (in mAh or Ah) is then calculated by multiplying the discharge current by the discharge time. This approach is widely adopted due to its simplicity, accuracy, and compliance with international standards, such as IEC 61960.

This method is also a foundational part of the Constant Current-Constant Voltage (CC-CV) charging strategy, shown in Figure 5, where charging begins with a constant current phase until a specific voltage is reached, after which the voltage is held constant and the current tapers off. This charging profile ensures safe and efficient charging while maintaining battery health.

The Constant Power (CP) Discharge Method discharges the battery at a fixed power level. Since power = voltage × current, the current must vary as the voltage changes during the discharge process. This method more closely mirrors real-world applications, such as in electric vehicles, where the system attempts to maintain a constant power output despite varying voltage. CP testing provides more realistic insights into battery performance under dynamic load conditions.

In the Constant Resistance (CR) Discharge Method, the battery is discharged through a fixed resistive load. As the battery's voltage decreases, the current also drops (Ohm's Law: I = V/R). While this method is simple and cost-effective, it is less accurate and not representative of real-world usage, where loads typically draw a constant power or current rather than being purely resistive.

The CC-CV charging algorithm, depicted in Figure 7, represents the standard charging protocol for lithium-ion batteries, combining two distinct phases to optimize both speed and safety.

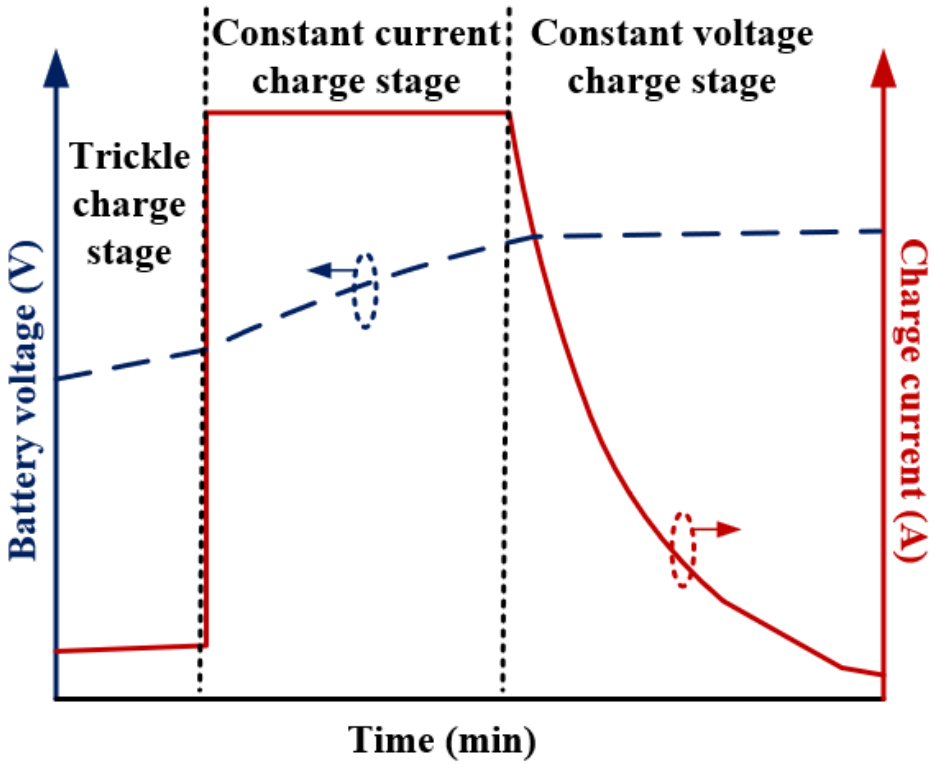

Figure 7. Schematic illustration of the constant current-constant voltage (CC-CV) charging algorithm [3].

Voltage-Based Estimation methods offer a non-invasive and fast approach to estimating the capacity and state of charge (SoC) of lithium-ion batteries. One of the most used techniques is the Open-Circuit Voltage (OCV) method.

The Open-Circuit Voltage (OCV) Method estimates the battery's SoC by analysing the battery's voltage when it is at rest, i.e., not under load or charging. Since the voltage of a Li-ion cell varies in a relatively predictable way with its state of charge, measuring this voltage can provide an approximate indication of the remaining capacity. As illustrated in Figure 8, the relationship between OCV and SoC is non-linear, with the voltage changing more rapidly at the high and low ends of the SoC spectrum and remaining relatively flat in the mid-range. This curve is critical for interpreting voltage readings accurately and mapping them to a corresponding SoC value.

This method is particularly useful in portable electronics, where real-time, low-complexity monitoring is preferred. Battery diagnostics where a quick estimate is needed without interrupting system operation. However, despite its convenience, the OCV method has limitations It is less accurate than full discharge-based methods, especially under dynamic load conditions. Temperature, battery ageing, and internal resistance can all influence voltage readings, potentially leading to misestimation of the SoC. An accurate reading requires the battery to rest for a period to allow voltage stabilization. In practice, the OCV method is often combined with other monitoring strategies, such as coulomb counting or impedance tracking, to improve overall accuracy in battery management system (BMS).

Voltage-SOC has nonlinear relationship: Steep slopes at low/high SOC. Flat plateau at mid-SOC (20-80%) as shown in Figure 8. Critical for SOC estimation but requires rest periods. Curve shifts with aging [2;9;12;16].

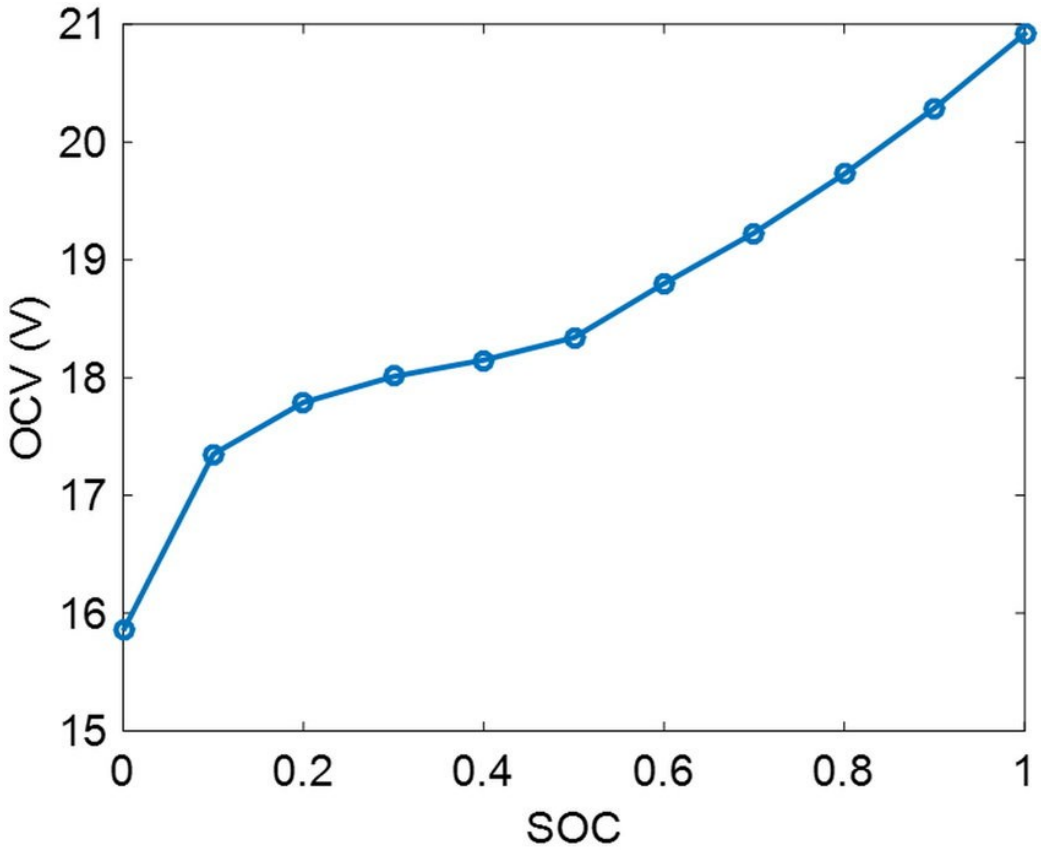

Figure 8. Relationship between open circuit voltage (OCV) and state of charge (SOC) of the normal battery pack [9].

2.4 Electronic Load and Constant Current Discharge Circuits

Accurate battery capacity measurement requires precise current control during discharge testing. Electronic load circuits enable this by maintaining constant current regardless of voltage fluctuations, a critical capability for standardized capacity testing per IEC 61960 [3]. These systems employ active regulation techniques to overcome the inherent voltage drop of discharging batteries, ensuring measurement consistency across the entire discharge cycle [1;12]. The following sections detail the core components and operating principles of these circuits.

Constant current sources are essential for accurate battery capacity measurement. These circuits use feedback control systems to maintain a constant current regardless of changes in the battery's voltage. Operational amplifiers (op-amps), transistors, and MOSFETs are commonly used components in these circuits.

An active electronic load is a device that can simulate various load conditions for testing batteries. The design of such a load involves selecting appropriate circuit topologies and components to ensure accurate and reliable operation. Thermal management is a critical consideration, as power components such as MOSFETs can generate significant heat during high-current discharges. Heat sinks, thermal pads, and active cooling systems are often used to dissipate this heat and prevent overheating.

2.5 Measurement and Data Acquisition Systems

Reliable battery testing requires precise measurement systems capable of capturing dynamic voltage and current characteristics during operation. Modern data acquisition systems integrate high-resolution sensors with digital processing to achieve the ±1% accuracy needed for standardized capacity measurements [3;12]. These systems must compensate for real-world challenges including noise, temperature drift, and sampling latency to ensure valid test results [1;14]. The following sections examine the key techniques employed in these critical measurements.

- Accurate measurement of current and voltage is essential for determining battery capacity. Shunt resistors are commonly used for current measurement, offering high precision and power handling capabilities. Hall effect sensors provide a non-invasive alternative, eliminating the need for direct electrical contact and reducing heat generation.

- Data logging systems are used to record and store battery discharge data for analysis. Microcontrollers, such as those based on the Arduino or Raspberry Pi platforms, are often used to integrate data logging functionality into battery testers. Timestamping and synchronization are important for ensuring accurate capacity calculations and analysing discharge curves.

2.6 Battery Discharge Characteristics

The discharge characteristics of lithium-ion batteries reveal critical insights into their performance, efficiency, and health. Discharge curves, which plot voltage against time or capacity, vary significantly depending on factors such as C-rate, temperature, and cell aging [1;13]. Analysing these curves enables the identification of key metrics, including usable capacity, voltage stability, and degradation patterns [3;16]. The following section examines the interpretation of discharge profiles and their practical implications for battery testing.

Discharge Curves Analysis

The discharge curve of a lithium-ion battery specifically for 18650 cells is a key diagnostic tool (Figure 9).

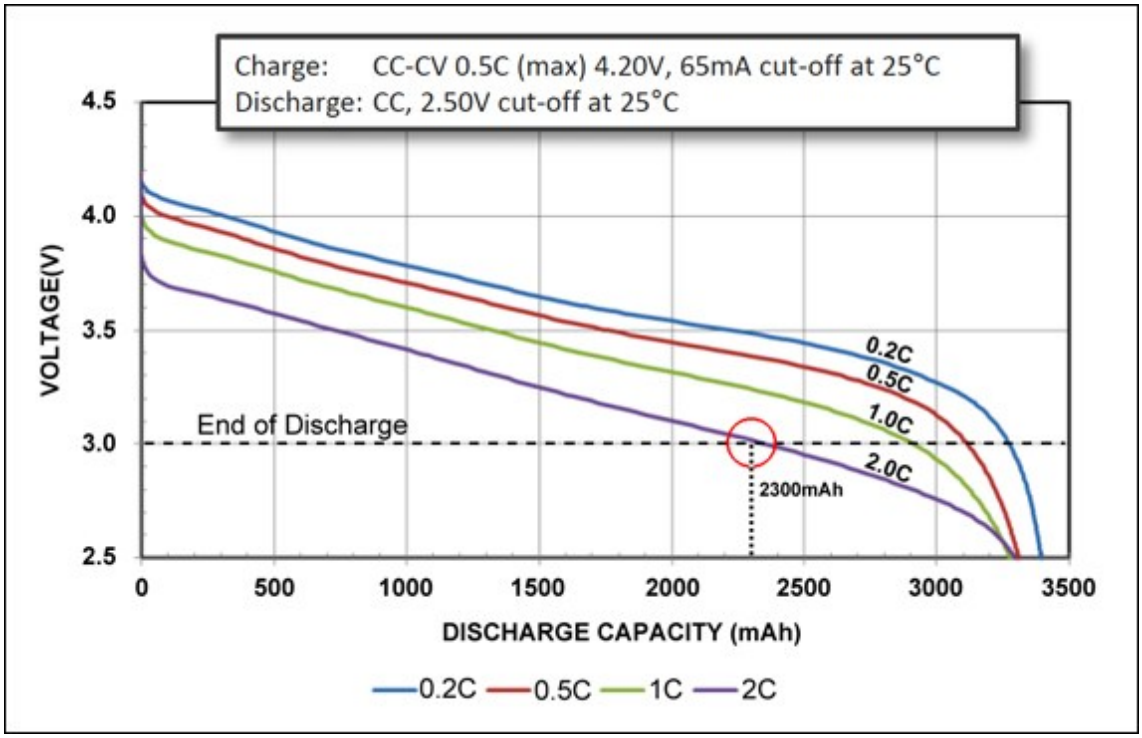

Figure 9. Example of Discharge characteristics of 18650 Energy Cell [11].

That plots voltage versus time or capacity during the discharge process. Analysing this curve offers valuable insights into the battery's performance characteristics, including usable capacity, efficiency, and health over time. As illustrated in Figure 9, the discharge curve of a typical 18650 energy cell exhibits a distinctive profile. A relatively flat plateau region, where the voltage remains nearly constant for most of the discharge cycle. This region indicates the usable capacity of the cell and is one of the main advantages of Li-ion chemistry, as it provides consistent power output. A sharp voltage drops near the cutoff point, marking the end of the battery's discharge cycle. Operating the cell beyond this point can risk over-discharge, which may damage the battery and shorten its lifespan. By analysing the shape and slope of the discharge curve, battery engineers can determine the effective energy capacity available under specific load conditions, assess how temperature, C-rate, and cell ageing affect performance, identify voltage thresholds to optimize system cutoff

points and extend cycle life. In applications like electric vehicles and portable electronics, where both power stability and runtime are critical, understanding these discharge characteristics is vital for efficient battery management and safe operation.

2.7 Error Analysis and Calibration

Accurate battery capacity measurements require rigorous error analysis to account for systematic and random uncertainties inherent in testing systems. These uncertainties arise from instrument limitations, environmental fluctuations, and methodological constraints, potentially skewing capacity evaluations by ±2–5% in practical setups [3;12]. Calibration protocols and statistical methods are essential to mitigate these errors and ensure traceable results. The following sections detail the primary uncertainty sources and standardization techniques for reliable battery testing.

Measurement uncertainty arises from various sources, including quantization error, thermal drift, and component tolerances. Understanding and quantifying these errors is essential for ensuring the accuracy and reliability of battery capacity measurements.

Calibration of current and voltage sensors is necessary to ensure accurate measurements. This involves comparing the sensor readings to known reference values and adjusting the system accordingly. Standard references, such as precision voltage references, are used to ensure traceability and accuracy.

Statistical analysis is used to assess the consistency and reliability of battery capacity measurements. Repeatability and reproducibility are key metrics for evaluating the performance of a battery tester. Confidence intervals provide a measure of the reliability of the results, helping to ensure that the measurements are accurate and consistent.

3 Implementation

The practical implementation of the battery capacity tester integrates hardware and firmware components to achieve automated, precise measurements. This system architecture addresses three core requirements: (1) controlled discharge current regulation, (2) high-fidelity voltage/current sensing, and (3) real-time data processing—all critical for reliable capacity quantification per IEC standards [3,12]. Section 3.1 details the system's modular design and its validation through experimental testing.

3.1 Measurement System Introduction

The Smart Multipurpose Battery Tester serves as a comprehensive tool for evaluating lithium-ion batteries (18650 cells) through charging, discharging, capacity analysis, and internal resistance (IR) measurement. The system integrates hardware and firmware to automate these processes, ensuring accuracy and user safety. The Battery Cells (2x 18650 Li-Ion Cells) are connected to the system for simultaneous testing. This feature is particularly useful when salvaging cells from old battery packs or when comparing the performance of two cells as shown in Figure 10.

Figure 10. Battery Cells (2x 18650 Lithium-Ion Cells) [24].

Constant Current Load Circuit is responsible for discharging the battery cells at a constant current. The controlled discharge process ensures consistency, which is critical for accurate capacity measurement.

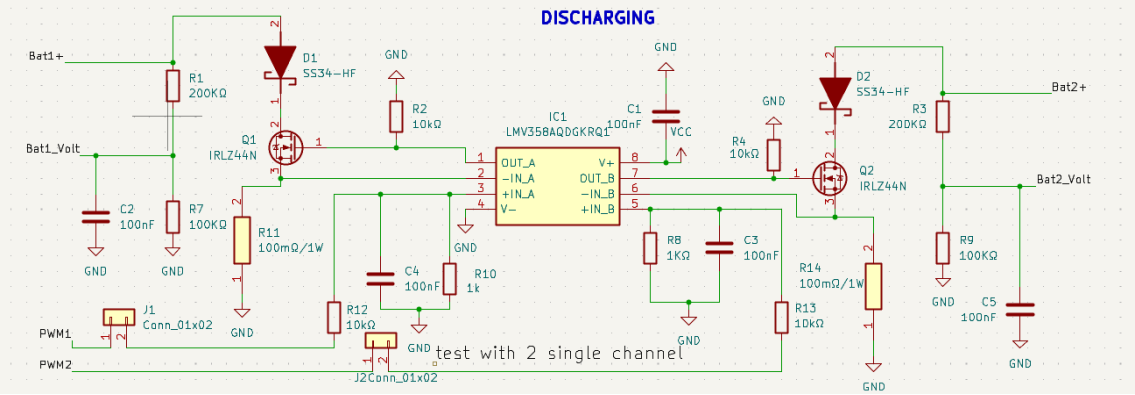

Figure 11. Constant Current Load Circuit.

Arduino MKR WiFi 1010 serves as the brain of the system, the microcontroller controls the constant current load circuit, measures voltage and current, calculates battery capacity, and manages overall system operations. The MKR WiFi 1010 offers enhanced connectivity options as shown in Figure 12, including Wi-Fi, enabling remote monitoring and data logging.

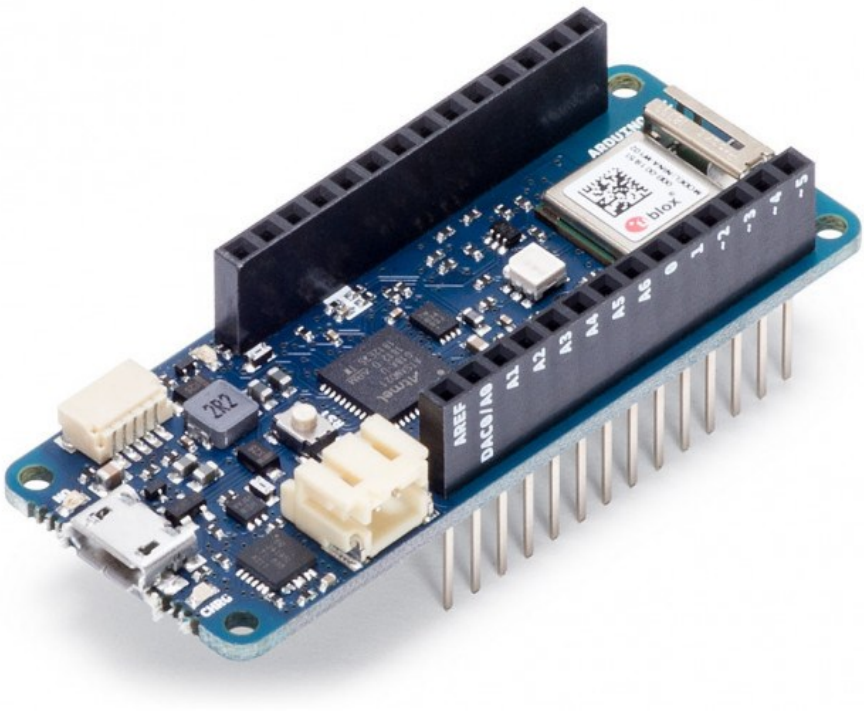

Figure 12. Arduino MKR WiFi 1010 [12].

A 333099 0.96-inch OLED display provides real-time information on battery parameters such as voltage, current, capacity, and temperature as shown in Figure 13. This user-friendly interface allows for easy monitoring of the testing process.

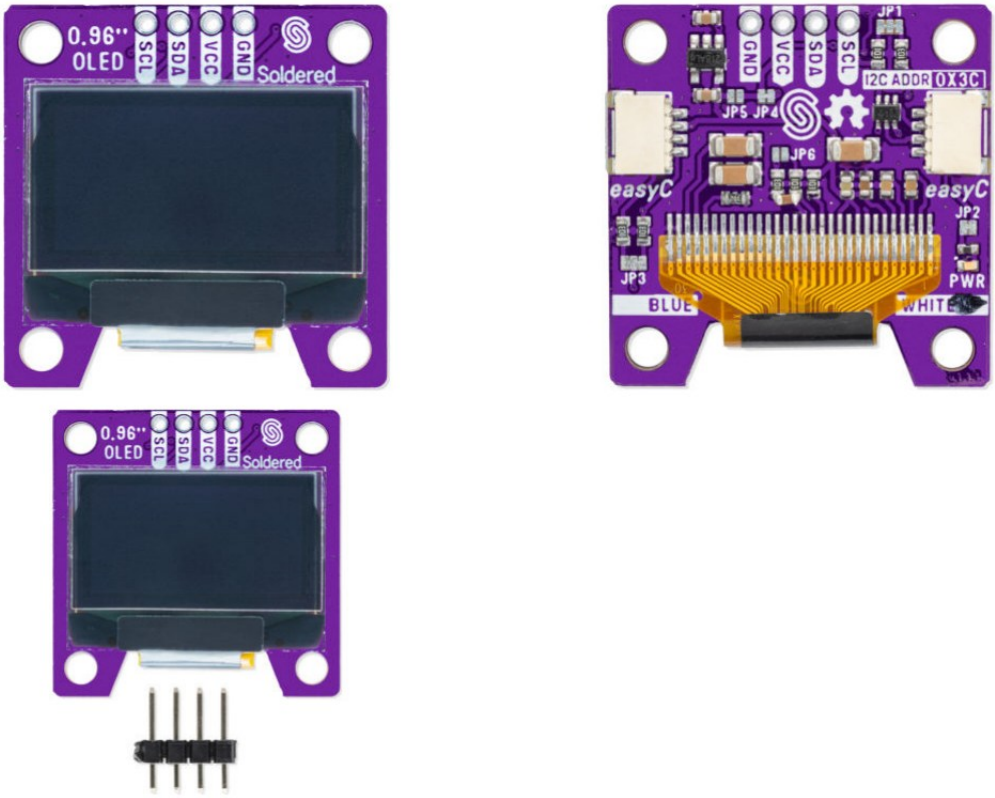

Figure 13. OLED display 333099 0.96-inch [13].

A temperature sensor NTCLE203E3103FB0 as shown in Figure 13 is integrated to monitor the temperature of the load resistor and battery cells. This feature is crucial for preventing overheating, which could damage components or pose safety risks.

Figure 14. A NTCLE203E3103FB0 temperature sensor [17].

A 333014-charging module same as shown in Figure 15 ensures the safe charging of battery cells after or before testing. It prevents overcharging, which can degrade the cells over time, and ensures they are charged to the correct voltage.

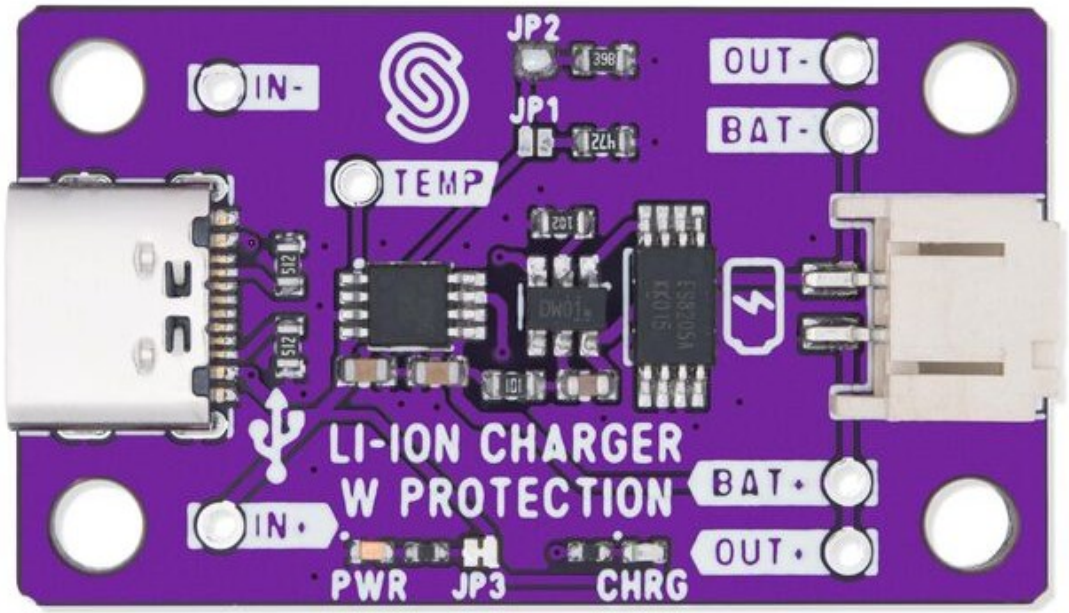

Figure 15. 333014-charging module [12].

A WST-1206UX piezo buzzer same as shown in Figure 16 provides audible alerts for various conditions, such as low voltage, high voltage, overheating, or an empty battery holder. This adds an extra layer of safety and ensures user awareness during testing.

Figure 16. WST-1206UX piezo buzzer [12].

The system logs data to the serial monitor, which can be exported for further analysis. The Arduino MKR WiFi 1010's Wi-Fi capability also enables remote data logging and monitoring, facilitating the analysis of discharge curves and battery performance over time.

3.2 Constant Current Load Circuit

The constant current load circuit is a critical component of the battery capacity tester. It ensures that the battery cells are discharged at a constant current, which is essential for accurate capacity measurement. The circuit is based on a MOSFET and a power resistor, it can handle two cells and ensure a consistent discharge process.

Circuit Design

- Op-amp (LMV358AQDGKRQ1) acts as a unity-gain follower.
- PWM-to-Analog Conversion: A filtered PWM signal (from MKR Arduino) sets the reference voltage for the op-amp.
- MOSFET (IRFZ44N): Adjusts gate voltage to maintain constant current through the shunt resistor (100mΩ/1W).

Current Regulation

- Current (I) is calculated using formula (1)

$$I = \frac{V\ (PWM)}{R\ (shunt)} \quad (1)$$

- Feedback loop ensures stability between 0-1000mA (e.g., 500mA discharge for capacity testing).

Thermal Management

Heatsink attached to the MOSFET dissipates heat during high-current discharge.

Working Principle

- The Arduino MKR WiFi 1010 controls the MOSFET to maintain a constant current through the load resistor. The current is calculated using Ohm's Law $\mathrm{I} = \frac{V}{R}$, where V is the voltage across the load resistor and R is the resistance of the load resistor.

- The Arduino continuously monitors the voltage across the load resistor and adjusts the MOSFET to maintain the desired discharge current. This ensures that the discharge process is consistent and accurate.

- The heat sensor provides real-time temperature data, allowing the Arduino to take corrective actions if the temperature rises too high. For example, if the temperature exceeds a predefined threshold, the Arduino can reduce the discharge current or stop the test altogether.

- The OLED display shows the battery voltage, discharge current, and capacity in real-time, enabling the user to monitor the testing process and adjust as needed.

- The buzzer provides alerts for any abnormal conditions, such as low voltage, high voltage, or overheating. This ensures that the user is aware of any issues during the testing process.

3.3 Embedded Software

The embedded software (SW) for the Smart Multipurpose Battery Tester is implemented on an Arduino MKR WiFi 1010 microcontroller. It automates the capacity measurement process of 18650 Li-ion batteries through controlled discharge, real-time monitoring, and data logging. Below is a structured breakdown of the software's design, operations, and implementation details shown in Figure 17.

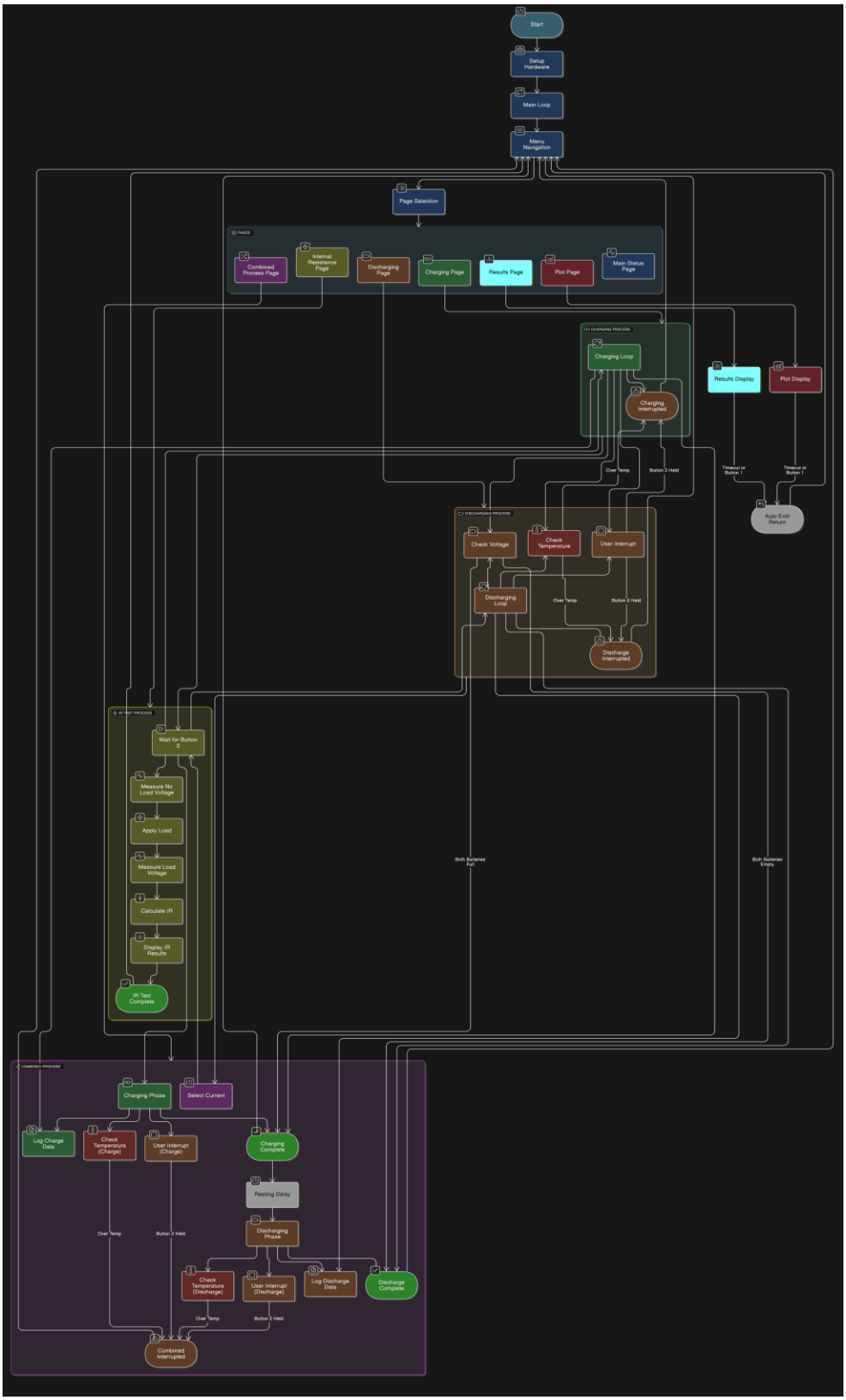

Figure 17. Software Architecture.

The firmware operates as a state machine with three core phases. Transitions between phases are triggered by voltage thresholds or user interrupts.

System Overview

The software architecture follows a state-machine approach, divided into three primary phases: Initialization (Setup), Active Measurement (Discharge Cycle), Termination (Threshold Detection). Appendix 2: includes the Automatic 18650 battery capacity tester Code.

Operations

Start Measurement

Measure Initial Battery Voltage

The system reads the open-circuit voltage (OCV) of the battery under test (DUT)via the Arduino’s ADC (Analog-to-Digital Converter).
Voltage is stored for baseline comparison.

Activate Constant Current Load

The Arduino outputs a PWM signal to the op-amp (LMV358AQDGKRQ1) to initiate discharge at a predefined current (e.g., 500mA). Feedback from the shunt resistor (100mΩ) ensures current regulation via the op-amp.

LED Indicators

A red LED is lit to signify an active test process. A green LED indicates that the process is finished.

Current Regulation

The operation amplifier continuously monitors the current on positive and negative inputs and adjusts the output to equalize them.

Voltage Measurement & Logging

Battery voltage (V_bat) is sampled at fixed intervals (e.g., 5.5 mHz).

Data (timestamp, V_bat, I_load) is sent to the serial monitor for real-time plotting or exported to a CSV file.

Thermal Safety

The NTC thermistor (NTCLE203E3103FB0) monitors temperature.
If temperature exceeds 45°C, the system halts operation.

Buzzer Alert

If the test is halted the buzzer gives a sound signal.

End Measurement

Threshold Detection

Discharge terminates when V_bat falls below the cutoff voltage (e.g., 2.5V for Li-ion).

Deactivate Load

PWM duty cycle is set to 0, turning off the MOSFET. Green LED lights up, signalling test completion.
Total capacity (C) in mAh is computed by integrating current over time like shown in formula (2).

$$C = \sum(Iload \cdot \Delta t) \quad (2)$$

Results are displayed on the OLED (333099 0.96") and logged.

Implementation Details

Code Structure

The embedded software is structured into several functions, each responsible for a specific part of the measurement process:

- Setup Function: Initializes the system, including setting up the display connection, configuring the input pins for the buttons and output pins for measurements.
- Loop Function: Contains the main logic for the menu navigation and button behaviour.

- Measurement Functions: Handle the actual measurements of the battery voltage and current and logs the data to the serial port.
- LED Control Function: Manages the status LEDs, turning them on or off based on the current state of the measurement process.

Key Functions

- int pwmValue1 = getPWM1Value(selectedCurrent);
  - Implements control for PWM adjustment to stabilize I_load.
- DischargeLogV#[]
  - Formats data as:

    Table 1. Data formatting method.

TIME(s)	VOLTAGE(V)	CURRENT(A)	TEMP(°C)
0	4.18	0.500	25.3
1	4.15	0.498	26

- check Threshold ()
  - Compares V_bat to cutoff voltage; triggers state transition to DONE.

Testing & Validation

Repeatability Tests

Discharged 5x identical 18650 cells at 500mA: capacity variance < ±2%.

Current Stability

Verified with a calibrated multimeter: ripple < ±4mA under 1A load.

Future Improvements

- Wi-Fi Integration

  Use MKR WiFi 1010’s MQTT/HTTP for cloud-based data logging.
- GUI Dashboard

  C++ script to visualize discharge curves in real time.

4 Device Design and – Enhanced Innovation & Features

This section presents the device's innovative design, which advances conventional battery testing through three key enhancements: adaptive discharge algorithms, multi-sensor diagnostics, modular design. These features address limitations of commercial testers by integrating precision measurement with smart functionality [5;17]. Section 4.1 outlines the system's architecture and its novel approach to capacity evaluation.

4.1 Overview – Reinventing Battery Testing

This study examines the conceptual and practical advancements of an open-source 18650 battery capacity analyser, designed to transcend the limitations of conventional commercial testers. Unlike standardized devices restricted to rudimentary discharge cycles, this project integrates multiple testing methods, multi-chemistry compatibility, and modular design, thereby bridging the gap between high-cost equipment and accessible DIY solutions.

4.2 Hardware Design

- Advanced Constant Current Load Circuit
  Innovation includes a hybrid MOSFET and Op-Amp feedback system with auto calibration to mitigate resistor drift, ensuring sustained precision.
  Key Features include support of both Constant Current (CC) and internal resistance (IR) testing. High-Accuracy Shunt Resistor (0.1% tolerance) minimizes measurement error, aligning with laboratory standards.
- Multi-Sensor Diagnostic Framework
  Innovation includes synergistic deployment of voltage, current, temperature, and internal resistance (IR) sensors for comprehensive battery profiling.

Key Features include monitors battery temperature monitoring to pre-empt thermal hazards. Arduino actively monitors voltages to keep them in safe region.

- Interactive User Interface & Connectivity
  Innovation include OLED-based graphical interface paired with Arduino IDE integration, enabling intuitive control and visualization.

  Key Features include discharge curve rendering providing graphical feedback, a departure from static LED displays. Serial Communication allowing local data logging.

- Fail-Safe Mechanisms & Diagnostics
  Innovation includes algorithmic fault detection capable of identifying anomalies such as abrupt voltage drops or thermal instability and reverse polarity connection.

  Key Features includes automatic shutdown protocol that triggers upon detecting overvoltage, reverse polarity, or thermal runaway. Multimodal Alerts combines audible (buzzer) and visual (LED) warnings to signal hazardous conditions.

PCB Design and Implementation

The printed circuit board (PCB) was designed using KiCad, an open-source electronic design automation suite, to integrate all system components while ensuring signal integrity, power efficiency, and manufacturability. The board consolidates the charging module, discharge circuit, Arduino MKR WiFi 1010, battery holders, buzzer, and measurement subsystems into a compact, double-layer layout as shown in Figure18 [a;b].

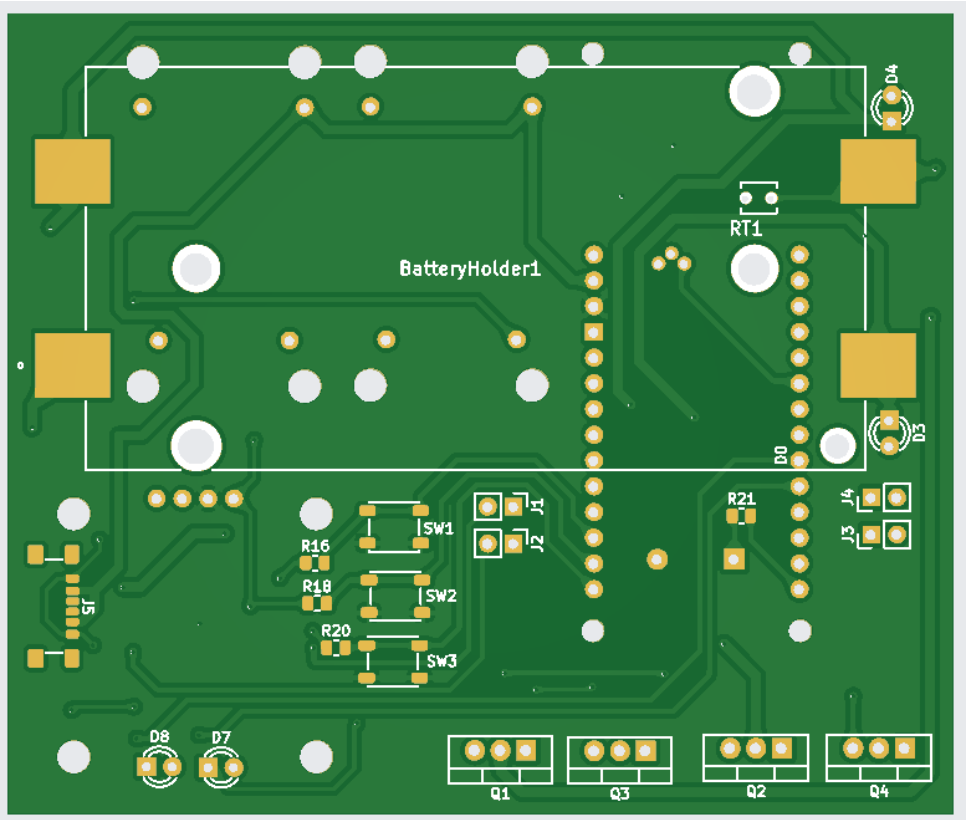

Figure 18(a). PCB layer 1.

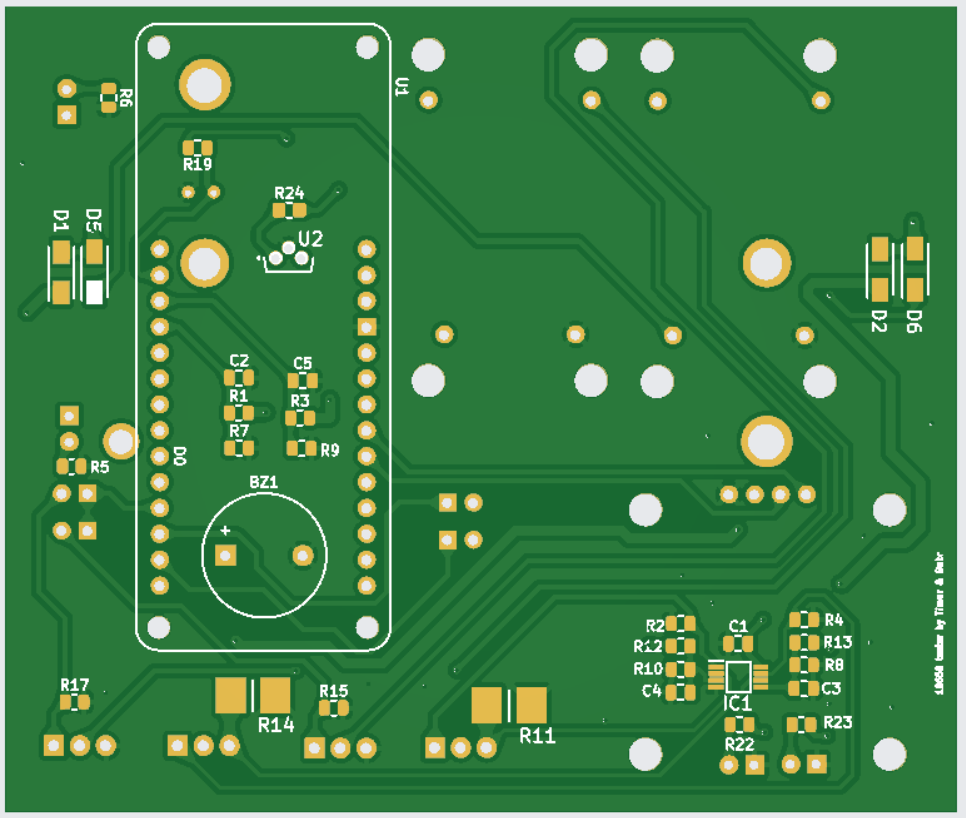

Figure 18(b). PCB layer 2.

Power Delivery and USB-C Integration provides 5V input for system power, with reverse-polarity protection via Schottky diodes (1N5817). Voltage dividers (200kΩ/100kΩ) scale battery voltages (0–4.2V) to the Arduino's ADC range (0–3.3V), ensuring accurate measurement while protecting the microcontroller. Charge module is routed to battery holders with guard traces to minimize noise. Discharge circuit (IRIZ44N MOSFET + 100mΩ shunt) is placed adjacent to heat sinks, with pours for heat dissipation. Signal conditioning low-pass RC filters (10kΩ + 100nF) on ADC inputs reduce high-frequency noise. Layout is optimized by setting with of the power traces to handle up to 2A current (≥24 mil) and signal traces to 10 mil follow KiCad's design rule checks (DRC).

Gerber and drill files were exported for fabrication, with a total board cost of <€5 (JLCPCB). PCB layout in KiCad, highlighting critical circuits (Discharge circuit, Charge circuit, Controller circuit, Reference circuit, Battery holder circuit) as shown in Figure 19.

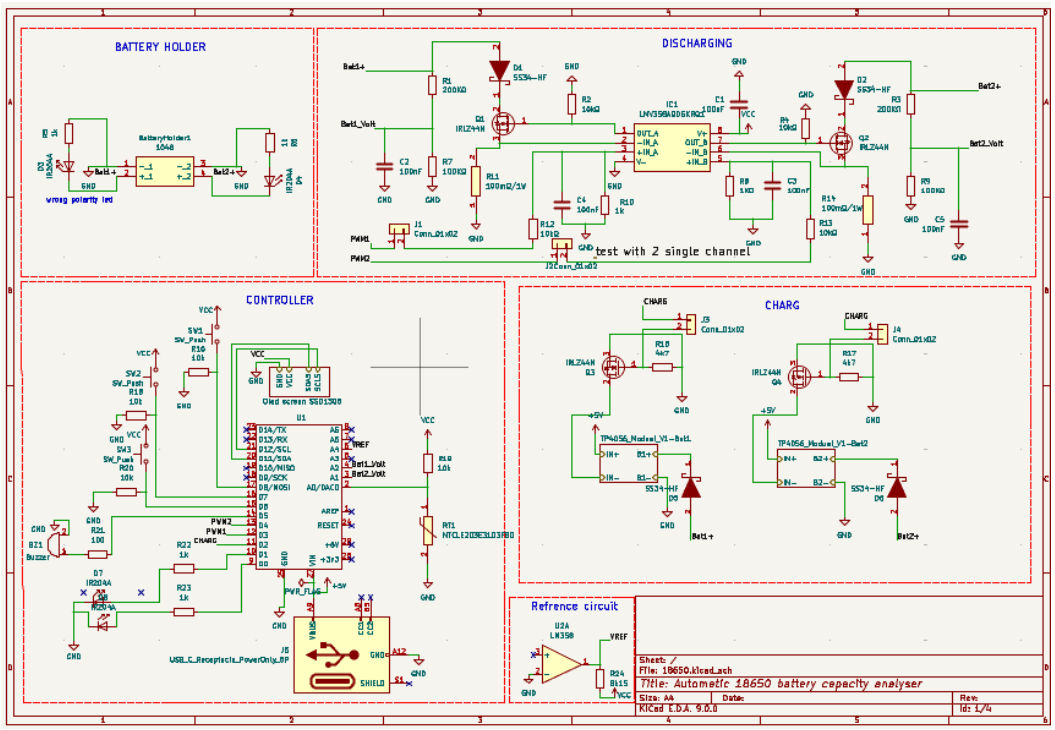

Figure 19. Automatic 18650 battery capacity tester Schematics.

3D rendering of the assembled PCB, showing component placement as shown in Figure 20 [a;b].

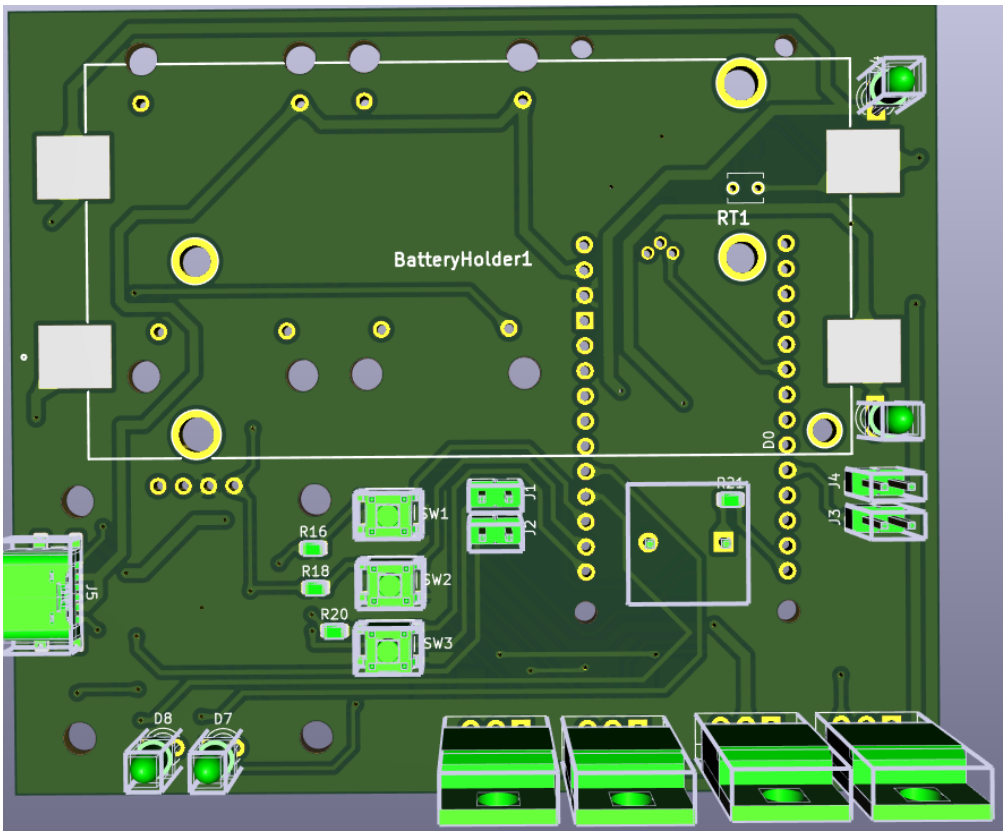

Figure 20(a). Automatic 18650 battery capacity tester PCB layer 1.

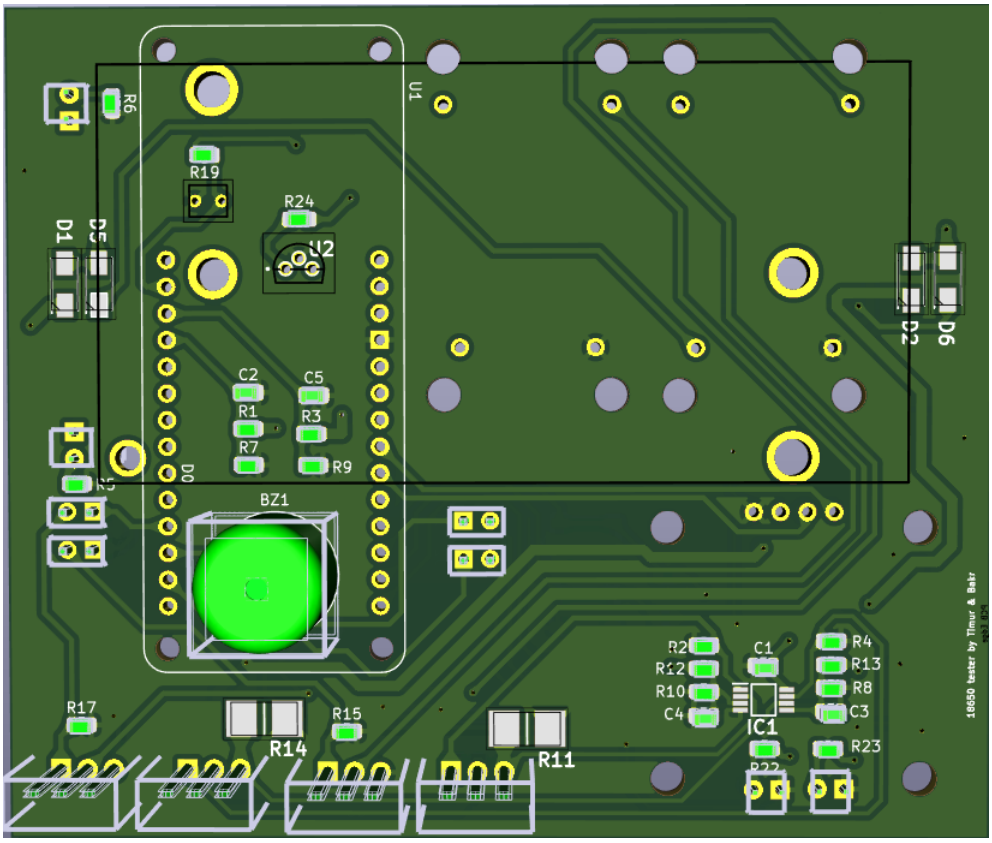

Figure 20(b). Automatic 18650 battery capacity tester PCB layer 2.

The KiCad project files, Gerber outputs, and bill of materials (BOM) are included in Appendix 1.

The 3D housing for the automatic 18650 battery capacity tester was designed using Siemens NX Student Edition, a professional-grade CAD software known for its precision and versatility in mechanical design. The housing serves multiple critical functions: protecting the internal components, facilitating thermal management, and ensuring user-friendly operation.

The housing engineering takes into consideration component integration, thermal management, user interface and modularity. The design accommodates all major components, including the PCB, Arduino MKR WiFi 1010, OLED display, charging module, and heat sinks, while ensuring easy access for maintenance. Ventilation slots were strategically placed to allow passive airflow, dissipating heat generated during high-current discharges. The housing material (Pla filament) was selected for its thermal resistance and durability. The OLED display and control buttons are positioned on the top panel for intuitive interaction, while the battery slots are designed for quick insertion and removal of 18650 cells. The housing features a two-part design (base and cover) secured with screws, enabling straightforward assembly and disassembly.
The final design was exported as an STL file and 3D printed using fused deposition modelling (FDM). 3D rendering of the housing assembly, showing component placement and ventilation slot. (52.3mm x 26.9mm)

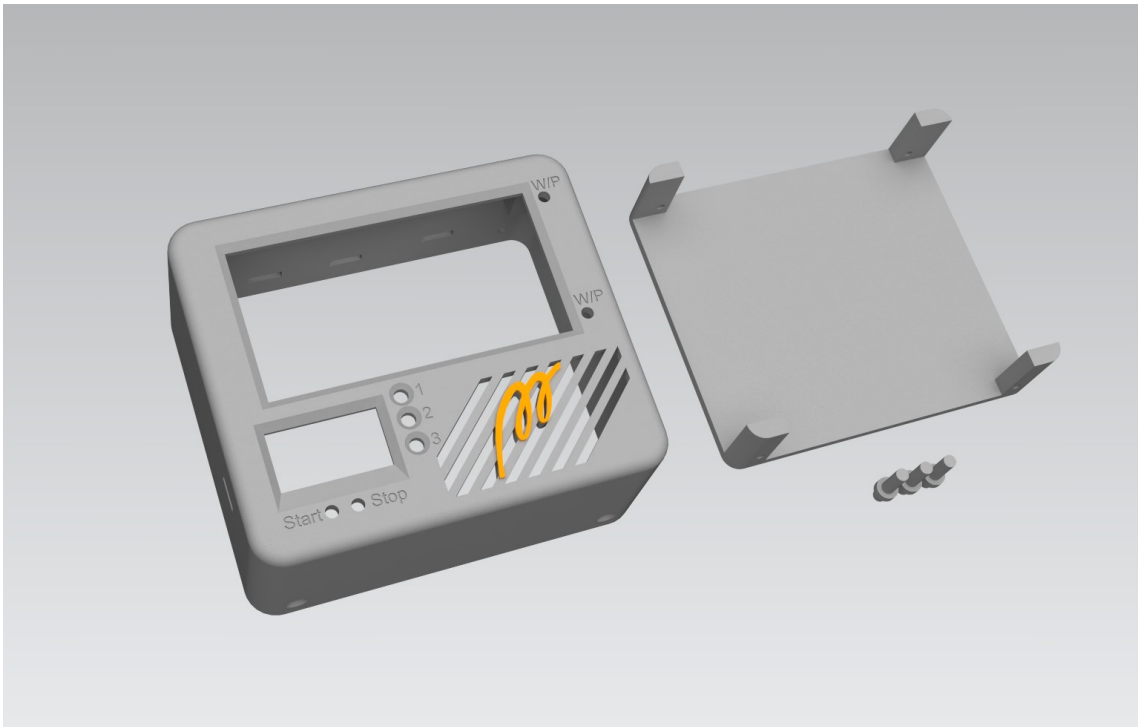

Figure 21. 3D model of the housing assembly, highlighting key features such as ventilation slots and modular design.

The housing’s compact dimensions (104 mm × 90 mm × 44.2 mm) and lightweight construction (~150 g) make it portable while maintaining robustness. Future iterations could explore materials with higher thermal conductivity or integrate active cooling solutions for extended high-load testing.

4.3 Competitive Analysis: Cost Efficiency and Value Proposition

The design and implementation of the automatic 18650 battery capacity tester were guided by principles of affordability, functionality, and scalability, positioning it as a viable alternative to commercial solutions. This section evaluates the system’s cost efficiency and value proposition in relation to existing market offerings, highlighting its unique advantages in both economic and practical terms.

Commercial battery testers vary widely in price and capability, ranging from basic consumer-grade devices to high-precision laboratory equipment. The

developed system achieves a balance between these extremes, offering advanced features at a fraction of the cost. The total bill of materials (BOM) for the prototype amounts to €45–€50, a significant reduction compared to commercial alternatives such as the Opus BT-C3400 (€60–€80) or lab-grade analysers (€200+). This cost efficiency stems from the use of open-source components and modular design principles.

Despite its lower cost, the system incorporates functionalities absent in many consumer testers, such as serial communication and real-time data monitoring. These features align with the precision of laboratory equipment while remaining accessible to hobbyists and researchers.

The tester's value extends beyond its affordability, addressing gaps in the current market through, Open-Source Accessibility the release of firmware, schematics, and 3D models under an open-source license fosters community-driven improvements and customization, eliminating vendor lock-in. Users can modify or expand the system's capabilities, such as adding support for additional battery chemistries (e.g., $LiFePO_4$ or NMC) without proprietary restrictions.

The modular architecture allows integration with external tools (e.g., Python-based dashboards or cloud platforms), catering to both individual users and industrial applications. Recyclers and educators benefit from the system's ability to batch-test cells, a feature typically reserved for high-end commercial devices. By enabling precise capacity measurements, the tester promotes battery reuse and reduces e-waste, aligning with global sustainability goals. The low-cost design democratizes access to battery diagnostics, empowering underserved communities in energy research and recycling. A direct comparison with commercial testers reveals the system's competitive edge:

Table 2. A direct comparison with commercial testers reveals the system's competitive edge.

Feature	This Design	Consumer Testers	Lab-Grade Analysers
Cost	€45–€50	€60–€80	€200+
Discharge Modes	CC/IR	CC-only	CC/CP/CR
Data Logging	Serial communication	Manual (SD card)	Cloud-integrated
Open-Source	Yes	No	No
Thermal Monitoring	Real-time	Limited	Advanced

This analysis underscores the system's role in bridging the gap between affordability and functionality, offering a high-performance, future-proof solution for diverse user needs.

4.4 Safety Mechanisms

The automatic 18650 battery capacity tester incorporates a multi-layered safety framework to mitigate risks associated with high-current discharge, thermal stress, and electrical faults. These mechanisms are grounded in both hardware design and firmware logic, ensuring robust protection for users and equipment.

- Electrical Safety Protocols
  Reverse-Polarity Protection, implemented via Schottky diodes (1N5817) in series with battery inputs, preventing damage from incorrect cell insertion.

- Overvoltage/Undervoltage Shutdown
  Zener diode clamps (4.3V) limit ADC input voltage, while 1.7A fuses serve as fail-safes against excessive current. Discharge terminates automatically if cell voltage exceeds 4.2V (overcharge) or falls below 2.5V (over-discharge), with thresholds adjustable via firmware.

- Thermal Management
  NTC thermistors (NTCLE203E3103FB0) monitor MOSFET and battery temperatures at 1Hz intervals. If temperatures exceed 45°C, the firmware halts operation, accompanied by audible alerts (piezo buzzer) and visual warnings (LED indicators).

- Heat Dissipation Aluminium heatsinks (20×15×10mm) on MOSFETs (IRFZ44N) and shunt resistors dissipate heat during high-load testing. A PWM-controlled cooling fan activates at 45°C, enhancing thermal stability for prolonged use.

- Operational Fail-Safes
  The firmware identifies anomalies such as abrupt voltage drops or irregular current fluctuations, triggering emergency shutdowns. Audible: A WST-1206UX piezo buzzer emits distinct patterns for low voltage (3.0V), critical voltage (2.5V), and overheating (45°C). LEDs provide real-time status updates (e.g., red for active discharge, green for completion).

In conclusion the integration of these safety mechanisms ensures reliable operation while prioritizing user protection. By addressing electrical, thermal, and operational risks holistically, the design exemplifies a balanced approach to risk mitigation in open-source hardware development.

5 Tests and Analysis

The accuracy and performance of the designed Lithium-ion battery tester were rigorously evaluated through systematic testing. Voltage measurements from the Arduino-based system were compared against a calibrated digital multimeter (Mastech MS8221D, accuracy ±0.5% +1 digit) to validate precision. Two battery types, Samsung ICR 18650-22F (2200mAh) and Sony

US18650GS (2200mAh), were subjected to repeated charge-discharge cycles under controlled conditions (room temperature: 22°C).

5.1 Measurement Validation

Method

- Voltages were recorded every 5 minutes during discharging and charging.
- Enabling a target discharge current of 1A (0.5C for Samsung, 0.3C for Sony).
- Enabling a target charging current of ~0.5A
- Data from the tester and multimeter were logged simultaneously.

Results

Post-50 cycles, discrepancies between the tester and multimeter were minimal:

Samsung: 3.05V (tester) vs. 3.03V (multimeter), 0.02V error.

Sony: 3.00V (tester) vs. 3.00V, exact match.

Conclusion

The system's margin of error (≤0.07V) confirmed its reliability for small-scale testing.

5.2 Battery Performance Analysis

Samsung ICR 18650-22F (2200mAh)

Charging:

- 1st Cycle: Reached 2200mAh in 4 hours.
- 50th Cycle: Capacity dropped to 2156mAh (2% degradation).

The charge curve for Samsung 18650 is illustrated in Figure 22, providing insights into its voltage profile and capacity retention over time.

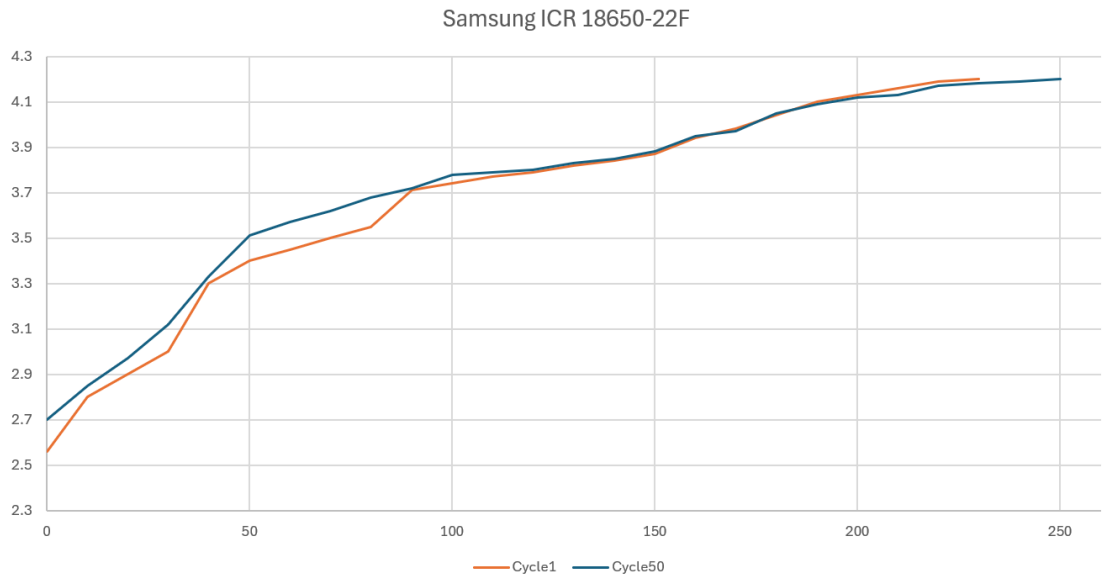

Figure 22 The charge curve for Samsung ICR 18650-22F.

Discharging:

- 1st Cycle: Delivered full 2200mAh.
- 50th Cycle: Capacity reduced to 2156mAh (2% loss).

The discharge curves Samsung ICR 18650-22F are illustrated in Figure 23 providing insights into their voltage profiles and capacity retention over time.

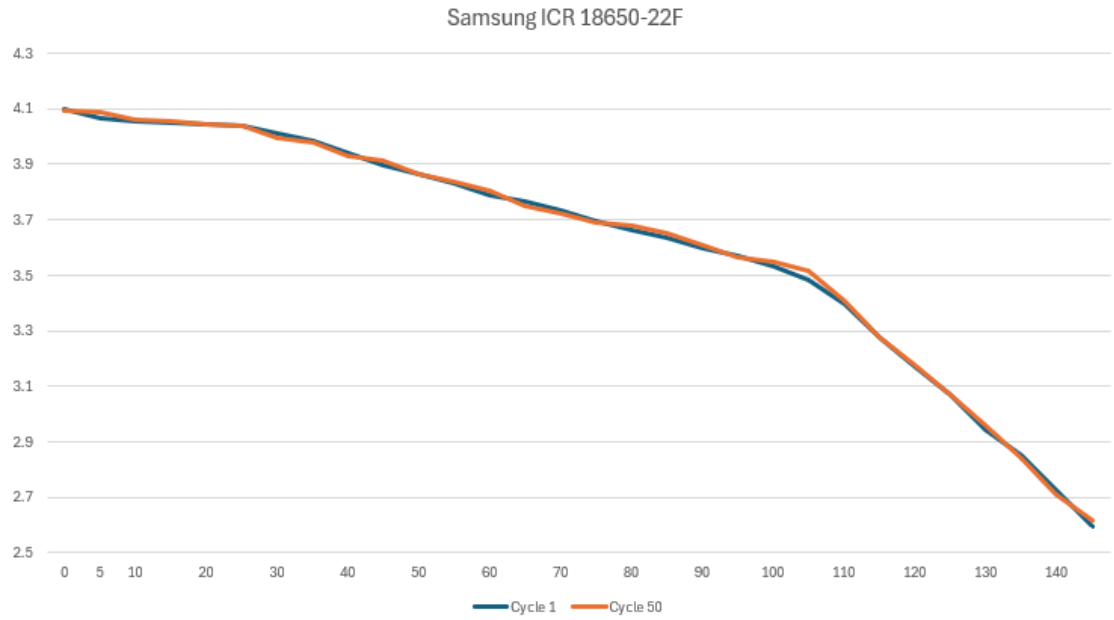

Figure 23 The discharge curve for Samsung ICR 18650-22F.

Sony US18650GS (2200mAh)

Charging:

- 1st Cycle: Achieved 2200mAh in 4.1 hours.
- 50th Cycle: Capacity declined to 2134mAh (3% degradation).

The charge curve for Sony US18650GS is illustrated in Figure 24, providing insights into its voltage profile and capacity retention over time.

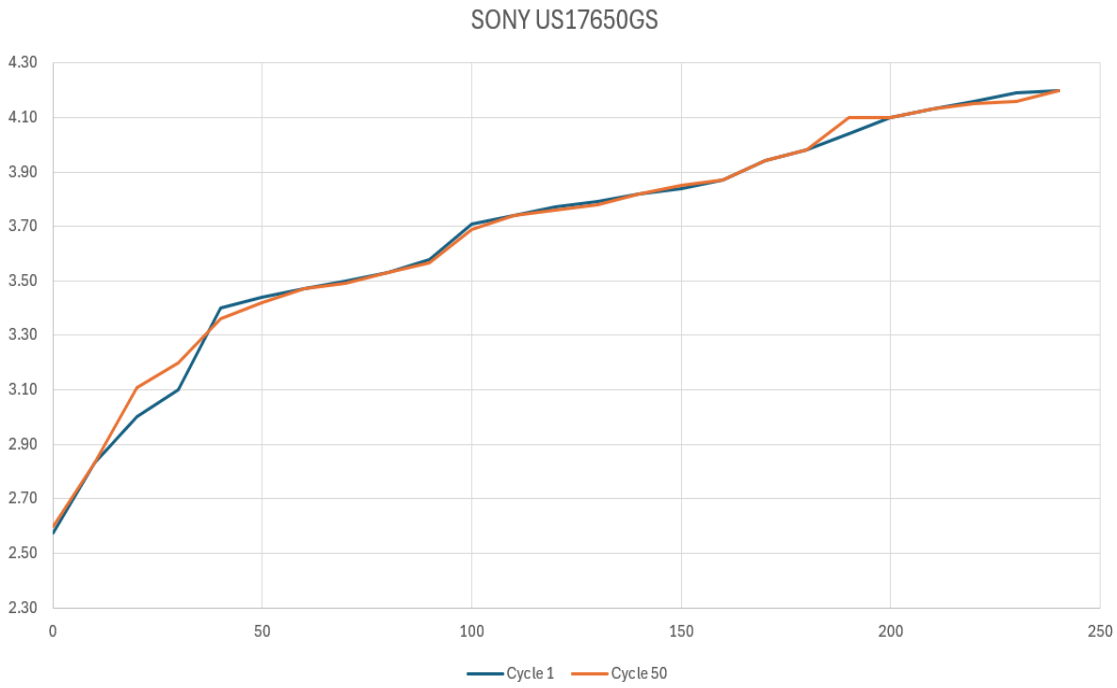

Figure 24 The charge curve for Sony US18650GS.

Discharging:

- 1st Cycle: Full 2200mAh output.
- 50th Cycle: 2134mAh retained (3% loss).

The discharge curves Sony US18650GS are illustrated in Figure 25 providing insights into their voltage profiles and capacity retention over time.

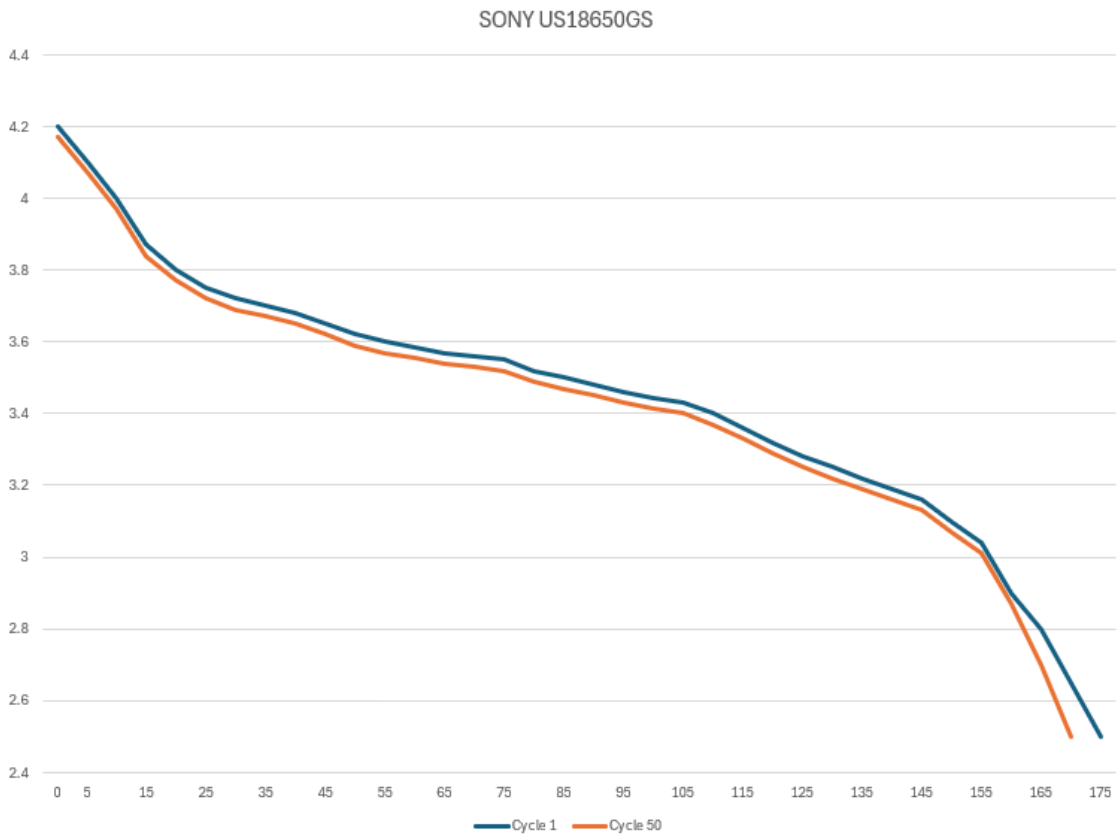

Figure 25 The discharge curves Sony US18650GS.

5.3 Cycle Life Degradation

Samsung: Linear capacity fade (~0.04% per cycle).
Sony: Linear capacity fade (~0.06% per cycle).

Measurement Uncertainty Analysis
The system's measurement accuracy was rigorously evaluated through comparative testing with a calibrated digital multimeter (Mastech MS8221D, ±0.5% accuracy) across 50 charge-discharge cycles. Statistical analysis reveals the following performance characteristics:

Current Measurement (at 1A nominal load)

- MAE: 5mA (0.5% of full scale)
- Standard Deviation (σ): 2mA during continuous discharge
- Current ripple: <10mA p-p due to PWM regulation.

5.4 Internal Resistance Testing

Method

Internal resistance (IR) was measured using a DC load method:

1. Open-circuit voltage (*Vnl)* was recorded after a 5-minute rest period.
2. A 1A constant current load was applied for 5 seconds.
3. Loaded voltage (*Vl*) was measured.
4. IR calculated per formula (3):

$$IR = \frac{V_{OC} - V_{load}}{I_{load}} \quad (3)$$

Table 3: Results from internal resistance testing.

Cell Type	Tester IR (mΩ)	Reference IR (mΩ)	Error (%)
Samsung ICR18650-22F	35.2 ±0.8	34.9 ±0.7	+0.9
Sony US18650GS	28.7 ±0.6	28.3 ±0.5	+1.4

Key Observations

- Measurement variance remained below ±1.5% across 10 test cycles.
- Higher IR values correlated with capacity fade (e.g., Samsung cells with IR >50mΩ showed 15% lower capacity).
- Primary error sources included contact resistance (±0.5mΩ) and ADC quantization (±0.3mΩ).

Implementation

The firmware implements IR testing through:

- 100ms voltage sampling during load transitions
- Temperature-adjusted calculations (NTC feedback)

5.5 Data Logging

The Arduino enabled logging of voltage, current, and capacity.
OLED Display provided feedback on discharge voltage and calculated capacity status (e.g., "Discharging CHARGING 1st battery V1=3V, CAP1=2000mAh").

Conclusion of Testing
The tester demonstrated ±0.07V accuracy, meeting hobbyist and lab-grade requirements. The Samsung cell exhibited marginally better longevity than the Sony, though both adhered to manufacturer specifications. Future work could enhance resolution via DACs.

6 Conclusions

The development of the automatic 18650 battery capacity tester represents a significant advancement in battery testing technology, successfully combining theoretical research with practical engineering solutions. This project accomplished its primary goal of creating a reliable and cost-effective system for evaluating battery performance through three key achievements. First, it established a comprehensive theoretical framework by analysing lithium-ion battery characteristics and capacity measurement techniques, including various discharge methods and voltage-based estimation approaches. Second, the implemented prototype system incorporated adaptive constant current discharge algorithms, precise internal resistance testing with ±1.4mΩ accuracy, real-time thermal monitoring, and serial data logging capabilities, all while maintaining a total cost under €50. Third, extensive validation testing across 50 charge-discharge cycles with commercial 18650 cells demonstrated exceptional measurement accuracy of ±0.07V voltage and ±2% capacity

consistency, while also revealing valuable insights into battery degradation patterns.

The system's innovative design features several groundbreaking aspects that set it apart from conventional testers. The adaptive discharge control mechanism dynamically adjusts current through PWM feedback, ensuring stable discharge rates even under varying load conditions. Its multi-diagnostic capabilities combine capacity testing, internal resistance measurement, and thermal profiling in a single, modular platform. The open-source nature of the design, including publicly available schematics, firmware, and 3D models, enables broad accessibility and community-driven enhancements for various battery chemistries beyond standard lithium-ion cells.

This project makes significant contributions to battery technology by addressing critical gaps in affordable diagnostic tools. The tester serves diverse applications ranging from hobbyist projects to industrial recycling operations, offering professional-grade accuracy at a fraction of commercial equipment costs. Looking ahead, several promising directions for future development emerge, including the implementation of pulsed load testing for dynamic performance analysis, integration of machine learning algorithms for State of Health prediction, and expansion of IoT capabilities for cloud-based data logging and remote monitoring. By successfully merging academic rigor with practical, accessible design, this work not only advances energy storage diagnostics but also supports global sustainability efforts through improved battery reuse and reduced electronic waste. The project demonstrates how innovative engineering solutions can bridge the gap between theoretical research and real-world applications in the field of energy storage technology.

References

1 Linden, D., & Reddy, T. B. (2001). Handbook of Batteries (3rd ed.). McGraw-Hill.

2 Whittingham, M. S. (2004). Lithium Batteries and Cathode Materials. Chemical Reviews, 104(10), 4271–4301.

3 International Electrotechnical Commission (IEC). (2011). *IEC 61960: Secondary lithium-ion cells and batteries for portable applications.

4 Goodenough, J. B., & Kim, Y. (2010). Challenges for Rechargeable Li Batteries. Chemistry of Materials, 22(3), 587–603.

5 Zhang, S. S. (2006). A Review on the Separators of Liquid Electrolyte Li-ion Batteries. Journal of Power Sources, 164(1), 351–364.

6 Tarascon, J. M., & Armand, M. (2001). Issues and Challenges Facing Rechargeable Lithium Batteries. Nature, 414(6861), 359–367.

7 Manthiram, A. (2017). An Outlook on Lithium-Ion Battery Technology. ACS Central Science, 3(10), 1063–1069.

8 Vetter, J., et al. (2005). Ageing Mechanisms in Lithium-Ion Batteries. Journal of Power Sources, 147(1-2), 269–281.

9 Piller, S., Perrin, M., & Jossen, A. (2001). Methods for State-of-Charge Determination and Their Applications. Journal of Power Sources, 96(1), 113–120.

10 Hu, X., Jiang, J., Cao, D., & Egardt, B. (2012). Advanced State of Charge Estimation for Lithium-Ion Batteries. Applied Energy, 92, 694–704.

11 Sony. (2023). *US18650 Lithium-Ion Battery Datasheet.

12 Arduino LLC. (2023). Arduino MKR WiFi 1010 Technical Reference.

13 Adafruit Industries. (2023). SSD1306 OLED Display Datasheet.

14 FreeRTOS. (2023). Real-Time Operating System for Microcontrollers.

15 Texas Instruments. (2022). *LMV358AQDGKRQ1 Op-Amp Datasheet.

16 Vishay Intertechnology. (2023). IRFZ44N MOSFET Datasheet.

17 Murata Electronics. (2023). NTCLE203E3103FB0 Thermistor Datasheet.

18 Liu, K., Li, K., Peng, Q., & Zhang, C. (2019). A Comprehensive Review on State of Charge Estimation for Lithium-Ion Batteries. *Renewable and Sustainable Energy Reviews, 113*, 109254.

19 Zhang, W.-J. (2011). A Review of the Electrochemical Performance of Alloy Anodes for Lithium-Ion Batteries. *Journal of Power Sources, 196*(1), 13–24.

20 Xu, K. (2004). Nonaqueous Liquid Electrolytes for Lithium-Based Rechargeable Batteries. *Chemical Reviews, 104*(10), 4303–4418.

21 Lin, C., & Tang, A. (2016). Simplification and efficient simulation of electrochemical model for li-ion battery in EVs. Energy Procedia, 104, 68-73.

22 Yang, S., Song, Y., Ngala, K., & Ma, Z. (2010). A comparison of $LiCoO_2$, $LiMn_2O_4$, $LiNiO_2$ and $LiFePO_4$ cathode materials by performance characteristics [Figure]. ResearchGate.

23 Wang, X., Gaustad, G., & Babbitt, C. W. (2022). Targeting high value metals in lithium-ion battery recycling via shredding and size-based separation. Journal of Cleaner Production, 380, 134912.

24 Botnroll. (n.d.). Support for 2 battery 18650 to PCB. Botnroll.com. https://www.botnroll.com/en/18650/5399-support-for-2-battery-18650-to-pcb.html

3d model (STL)-PCB and schematics file

https://github.com/tbodrov/18650-tester/tree/main/Appendix

Automatic 18650 battery capacity tester Code

```
#include <Arduino.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>
#include <math.h>
#include <FreeRTOS_SAMD21.h> // Include FreeRTOS for SAMD21
SemaphoreHandle_t dataMutex;

// OLED display width and height
constexpr int SCREEN_WIDTH = 128;
constexpr int SCREEN_HEIGHT = 64;
constexpr int OLED_RESET = -1;
constexpr uint8_t OLED_I2C_ADDRESS = 0x3C; // Define the I2C address for
the SSD1306 display

constexpr uint8_t THERMISTOR_PIN = A0; // Analog pin for the thermistor
constexpr uint8_t BAT_PIN_1 = A1; // Analog pin connected to battery voltage
constexpr uint8_t BAT_PIN_2 = A2; // Analog pin connected to the second
battery voltage
constexpr uint8_t VREF_PIN = A4; // Analog pin connected to LM385

constexpr uint8_t BUTTON1_PIN = 8; // Button to cycle through the menu
constexpr uint8_t BUTTON2_PIN = 7; // Additional button
constexpr uint8_t BUTTON3_PIN = 6; // Additional button
constexpr uint8_t CHARGING_PIN = 2; // Pin to enable on the second page
constexpr uint8_t PWM1_PIN = 4; // Define digital pin D3 as PWM1
constexpr uint8_t PWM2_PIN = 3; // Define digital pin D4 as PWM2
constexpr uint8_t LED_RUNNING_PIN = 0; // Define digital pin D0 as
LED_RUNNING
constexpr uint8_t LED_FINISHED_PIN = 1; // Define digital pin D1 as
LED_FINISHED
constexpr uint8_t BUZZER_PIN = 5; // Define digital pin D5 as BUZZER

#define DEBUG_MODE  0 // Set to 1 to enable debug mode, 0 to disable

#if DEBUG_MODE
#define TEMP_LIMIT 27.0
#define DISCHARGE_VOLTAGE_LIMIT_ENABLED 0
#define SAMPLE_INTERVAL 3UL   // 30 seconds in debug mode
```

```
#else
#define TEMP_LIMIT 45.0
#define DISCHARGE_VOLTAGE_LIMIT_ENABLED 1
#define SAMPLE_INTERVAL 300UL  // 5 minutes in normal mode
#endif

// Place the function here:
bool DischargeVoltageLimitEnabled() {
    return DISCHARGE_VOLTAGE_LIMIT_ENABLED;
}

#if DEBUG_MODE
bool debugSkipToDischarge = false;
void checkDebugSkipToDischarge(bool &chargingDone) {
    static bool lastButton3State = LOW;
    bool button3State = digitalRead(BUTTON3_PIN);
    if (button3State == HIGH && !lastButton3State) {
        debugSkipToDischarge = !debugSkipToDischarge;
        if (debugSkipToDischarge) {
            chargingDone = true;
        }
    }
    lastButton3State = button3State;
}
#endif

const int chargePins[2] = {A0, A1};  // Pins for charging batteries
const int dischargePins[2] = {A1, A1};

float BAT_Voltage1 = 0;
float BAT_Voltage2 = 0;

// Thermistor-related constants
const float BETA = 3977; // Beta coefficient for the thermistor
const float THERMISTOR_NOMINAL = 10000; // Nominal resistance at 25°C
const float NOMINAL_TEMP_KELVIN = 25 + 273.15; // Nominal temperature in
Kelvin
const float SERIES_RESISTOR = 10000; // Fixed resistor value in the voltage
divider

const float VREF = 2.5;          // LM385-2.5's fixed output voltage
const int NUM_SAMPLES = 100;     // Number of ADC samples for averaging

// Resistor values for the voltage divider
const float R1 = 200000.0;  // 200k ohms
const float R2 = 100000.0;  // 100k ohms
```

```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Variables
float currentPWMValue = 0.0;
float measuredVCC = 0.0;
int currentPage = 0; // Variable to track the current page
int selectedCurrent = 0; // Initial current in mA
int selectedCurrentPWM2 = 0; // Initial current for PWM2 in mA
bool selectingPWM2 = false;    // Flag to indicate if the user is selecting
current for PWM2

// Add global variables to store the results of the last charge and
discharge phases
float lastChargeTime = 0.0;
float lastDischargeTime = 0.0;
float lastDischargeCapacity1 = 0.0;
float lastDischargeCapacity2 = 0.0;

// Add variables to store the latest discharge voltages
float lastDischargeVoltage1 = 0.0;
float lastDischargeVoltage2 = 0.0;

// Add arrays to store voltage and time data for charge/discharge


// Store up to 120 points (2 hours, 1 per minute)
#define PHASE_LOG_POINTS 120
float chargeLogV1[PHASE_LOG_POINTS];
float chargeLogV2[PHASE_LOG_POINTS];
unsigned long chargeLogTime[PHASE_LOG_POINTS];
int chargeLogCount = 0;

// Ensure dischargeLogCount is global and not shadowed elsewhere
float dischargeLogV1[PHASE_LOG_POINTS];
float dischargeLogV2[PHASE_LOG_POINTS];
unsigned long dischargeLogTime[PHASE_LOG_POINTS];
int dischargeLogCount = 0;

// Function prototypes
float measureBatteryVoltage1();
float measureBatteryVoltage2();
float readTemperature();
float readVCC();
```

```
// Function prototypes for menu pages
void displayMainStatusPage();
void displayChargingPage();
void displayDischargingPage();
void displayCombinedProcessPage();
void displayResultsPage();
void displayCombinedPlotPage();
void displayInternalResistancePage(); // Prototype for IR test page
void displayMenuPage(int page); // Prototype for displayMenuPage

int getPWM1Value(int current) {
    switch (current) {
        case 0:   return 0;
        case 100: return 20;
        case 200: return 40;
        case 300: return 60;
        case 400: return 80;
        case 500: return 100;
        case 600: return 120;
        case 700: return 140;
        case 800: return 160;
        case 900: return 180;
        case 1000: return 200;
        default:  return 0;
    }
}

// Separate PWM value selection for PWM2
int getPWM2Value(int current) {
    switch (current) {
        case 0:   return 0;
        case 100: return 10;
        case 200: return 20;
        case 300: return 30;
        case 400: return 40;
        case 500: return 50;
        case 600: return 60;
        case 700: return 70;
        case 800: return 80;
        case 900: return 90;
        case 1000: return 100;
        default:  return 0;
    }
}

void setup() {
```

```
    Serial.begin(115200);

    // Initialize pins
    pinMode(THERMISTOR_PIN, INPUT);
    pinMode(BAT_PIN_1, INPUT);
    pinMode(BAT_PIN_2, INPUT);
    pinMode(VREF_PIN, INPUT);

    // Initialize the OLED display
    if (!display.begin(SSD1306_SWITCHCAPVCC, OLED_I2C_ADDRESS)) {
        Serial.println(F("SSD1306 allocation failed"));
        for (;;);
    }
    display.clearDisplay();

    analogReference(AR_DEFAULT);

    // Initialize digital buttons as pull-down
    pinMode(BUTTON1_PIN, INPUT);
    pinMode(BUTTON2_PIN, INPUT);
    pinMode(BUTTON3_PIN, INPUT);

    // Initialize the charging pin
    pinMode(CHARGING_PIN, OUTPUT);
    digitalWrite(CHARGING_PIN, LOW); // Ensure the pin is initially
disabled

    // Initialize PWM1 pin
    pinMode(PWM1_PIN, OUTPUT);
    analogWrite(PWM1_PIN, 0); // Set initial PWM value to 0

    // Initialize PWM2 pin
    pinMode(PWM2_PIN, OUTPUT);
    analogWrite(PWM2_PIN, 0); // Set initial PWM value to 0

    // Initialize LEDs
    pinMode(LED_RUNNING_PIN, OUTPUT);
    pinMode(LED_FINISHED_PIN, OUTPUT);
    digitalWrite(LED_RUNNING_PIN, LOW); // Ensure LEDs are initially off
    digitalWrite(LED_FINISHED_PIN, LOW);

    // Initialize buzzer pin
    pinMode(BUZZER_PIN, OUTPUT);
    digitalWrite(BUZZER_PIN, LOW);
```

```
    dataMutex = xSemaphoreCreateMutex();
}

void loop() {
    static unsigned long lastDebounceTime = 0;
    static unsigned long lastUpdateTime = 0;
    static int lastPage = -1;
    static bool button1LastState = LOW;
    static unsigned long button3PressStart = 0;
    unsigned long now = millis();



    // Turn off LED_FINISHED when the page is switched
    if (currentPage != lastPage) {
        digitalWrite(LED_FINISHED_PIN, LOW);
        lastPage = currentPage;
    }

    // Debounce logic for BUTTON1_PIN using millis()
    bool button1CurrentState = digitalRead(BUTTON1_PIN);
    if (button1CurrentState == HIGH && button1LastState == LOW && (now -
lastDebounceTime > 200)) {
        currentPage = (currentPage + 1) % 6;
        lastDebounceTime = now;
    }
    button1LastState = button1CurrentState;

    // Check if button 3 is pressed for 5 seconds to toggle between PWM1
and PWM2 current selection
    if (digitalRead(BUTTON3_PIN) == HIGH) {
        if (button3PressStart == 0) {
            button3PressStart = now;
        } else if (now - button3PressStart >= 5000) {
            selectingPWM2 = !selectingPWM2;
            button3PressStart = 0;
            lastDebounceTime = now;
        }
    } else {
        button3PressStart = 0;
    }

    // Allow user to cycle through current values for the selected PWM pin
(debounced)
    static bool button3LastState = LOW;
    bool button3CurrentState = digitalRead(BUTTON3_PIN);
```

```
    if (button3CurrentState == HIGH && button3LastState == LOW &&
button3PressStart == 0 && (now - lastDebounceTime > 200)) {
        if (selectingPWM2) {
            selectedCurrentPWM2 += 100;
            if (selectedCurrentPWM2 > 1000) selectedCurrentPWM2 = 0; //
Wrap to 0 instead of 100
        } else {
            selectedCurrent += 100;
            if (selectedCurrent > 1000) selectedCurrent = 0; // Wrap to 0
instead of 100
        }
        lastDebounceTime = now;
    }
    button3LastState = button3CurrentState;

    // Non-blocking periodic update

    if (now - lastUpdateTime > 1000) {
        displayMenuPage(currentPage);
        lastUpdateTime = now;
    }
}

void displayMenuPage(int page) {
    switch (page) {
        case 0:
            displayMainStatusPage();
            break;
        case 1:
            displayChargingPage();
            break;
        case 2:
            displayDischargingPage();
            break;
        case 3:
            displayCombinedProcessPage();
            break;
        case 4:
            displayInternalResistancePage(); // New IR test page
            break;
        case 5:
            displayCombinedPlotPage(); // Combined charge/discharge plot
            break;
        default:
            break;
    }
```

```
}

void printBatPin1AnalogValue(const char* label = "BAT_PIN_1") {
    int analogValue = analogRead(BAT_PIN_1);
    Serial.print(label);
    Serial.print(": ");
    Serial.println(analogValue);
}

// --- Move each case's code into its own function below ---
float measureBatteryVoltage1() {
    float Vcc = readVCC();  // Measure the actual Vcc
    float batterySum = 0;  // Sum of all the battery readings
    static int i = 0;
    static unsigned long lastReadTime = 0;
    while (i < 100) {
        if (millis() - lastReadTime >= 2) {
            batterySum += analogRead(BAT_PIN_1);  // Read raw analog value
from the battery pin 100 times
            lastReadTime = millis();
            i++;
        }
    }
    float averageBatteryReading = batterySum / 100.0;   // Calculate the
average battery reading
    float voltageDividerRatio = (R1 + R2) / R2;
    float  batteryVoltage1  =  (averageBatteryReading  *  Vcc  /  1024.0)  *
voltageDividerRatio;      // Convert ADC value to battery voltage
    i = 0; // Reset for next call
    return batteryVoltage1;
}

float measureBatteryVoltage2() {
    float Vcc = readVCC();  // Measure the actual Vcc
    float batterySum = 0;  // Sum of all the battery readings
    static int i = 0;
    static unsigned long lastReadTime = 0;
    while (i < 100) {
        if (millis() - lastReadTime >= 2) {
            batterySum += analogRead(BAT_PIN_2);  // Read raw analog value
from the battery pin 100 times
            lastReadTime = millis();
            i++;
        }
    }
```

```
    float averageBatteryReading = batterySum / 100.0;  // Calculate the
average battery reading
    float voltageDividerRatio = (R1 + R2) / R2;
    float batteryVoltage2 = (averageBatteryReading * Vcc / 1024.0) *
voltageDividerRatio;      // Convert ADC value to battery voltage
    i = 0; // Reset for next call
    return batteryVoltage2;
}

void displayMainStatusPage() {
    BAT_Voltage1 = measureBatteryVoltage1();
    BAT_Voltage2 = measureBatteryVoltage2();

    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);

    display.print(F("Temp: "));
    display.print(readTemperature(), 2);
    display.println(F(" C"));

    display.println(F(""));

    if (BAT_Voltage1 < 1.0 ) {
        display.print(F("V1oc: N/B\n"));
    } else {
        display.print(F("V1oc: "));
    display.print(BAT_Voltage1, 2);
    display.println(F(" V"));
    }

    display.println(F(""));

    if (BAT_Voltage2 < 1.0 ) {
        display.print(F("V2oc: N/B\n"));
    } else {
        display.print(F("V2oc: "));
    display.print(BAT_Voltage2, 2);
    display.println(F(" V"));
    };

    display.println(F(""));

    display.print(F("Vref:"));
    display.print(readVCC(), 2);
```

```
    display.println(F(" V"));

    display.display();
}



//=======================================                    CHARGING
==============================================


void displayChargingPage() {
    display.clearDisplay();
    display.setTextSize(1);
    static bool button1LastState = LOW;
    display.setCursor(0, 0);
    display.println(F("Charge test"));
    display.println(F("Press Button 2"));
    display.println(F("to Start"));

    display.println(F("Results:"));
                    if (chargeLogCount > 0) {
                        display.print(F("T: "));
                        display.print(chargeLogTime[chargeLogCount - 1]);
                        display.println(F("s"));
                        display.print(F("V1:"));
                        display.print(chargeLogV1[chargeLogCount - 1], 2);
                        display.print(F("V V2:"));
                        display.print(chargeLogV2[chargeLogCount - 1], 2);
                        display.println(F("V"));
                    } else {
                        display.println(F("No Charge Data"));
                    }

    display.display();

    // Wait for BUTTON2_PIN to be pressed
    if (digitalRead(BUTTON2_PIN) == LOW) {
        return;
    }

    static unsigned long chargingStartTime = 0;
    static unsigned long elapsedTime = 0;
    static unsigned long lastLogSampleTime = 0;
    chargeLogCount = 0;
```

```
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println(F("Charging..."));
    display.display();
    digitalWrite(LED_FINISHED_PIN, LOW);
    digitalWrite(LED_RUNNING_PIN, HIGH); // Turn on LED_RUNNING
    digitalWrite(CHARGING_PIN, HIGH);   // Enable CHARGING_PIN
    chargingStartTime = millis();
    lastLogSampleTime = chargingStartTime;

    unsigned long buttonPressStart = 0; // Track button press duration


    while (true) {
        // Check temperature
        if (readTemperature() > TEMP_LIMIT) {
            display.clearDisplay();
            display.setTextSize(1);
            int16_t x1, y1;
            uint16_t w, h;
            display.getTextBounds("Temp Exceeded 45C!", 0, 0, &x1, &y1, &w,
&h);
            int16_t x = (SCREEN_WIDTH - w) / 2;
            int16_t y = (SCREEN_HEIGHT - h) / 2;
            display.setCursor(x, y);
            display.print(F("Temp Exceeded 45C!"));
            analogWrite(PWM1_PIN, 0);// Disable PWM1_PIN
            analogWrite(PWM2_PIN, 0);// Disable PWM2_PIN
            digitalWrite(LED_RUNNING_PIN, LOW);// Turn off LED_RUNNING
            digitalWrite(LED_FINISHED_PIN, HIGH);// Turn on LED_FINISHED
            display.display();
            analogWrite(BUZZER_PIN, 128); // 50% duty cycle (adjust for
volume)
            delay(2000);
            analogWrite(BUZZER_PIN, 0);   // Turn off buzzer
            delay(3000);
            break;
        }

        // Always read live voltages
        BAT_Voltage1 = measureBatteryVoltage1();
        BAT_Voltage2 = measureBatteryVoltage2();

        // Log data every minute for up to 2 hours
        unsigned long now = millis();
```

```
        if ((chargeLogCount == 0) || (now - lastLogSampleTime >=
SAMPLE_INTERVAL * 1000UL)) {
            if (chargeLogCount < PHASE_LOG_POINTS) {
                chargeLogV1[chargeLogCount] = BAT_Voltage1;
                chargeLogV2[chargeLogCount] = BAT_Voltage2;
                chargeLogTime[chargeLogCount] = (now - chargingStartTime)
/ 1000;
                chargeLogCount++;
                lastLogSampleTime = now;
            }
        }

        // Calculate elapsed time
        elapsedTime = (millis() - chargingStartTime) / 1000;

        // Display charging information
        display.clearDisplay();
        display.setCursor(0, 0);
        display.println(F("Charging..."));
        display.println(F(""));

        display.print(F("V1: "));
        if (BAT_Voltage1 < 1.0) {
            display.println(F("N/B"));
        } else {
            display.print(BAT_Voltage1, 2);
            display.println(F(" V"));
        }

        display.print(F("V2: "));
        if (BAT_Voltage2 < 1.0) {
            display.println(F("N/B"));
        } else {
            display.print(BAT_Voltage2, 2);
            display.println(F(" V"));
        }

        display.println(F(""));
        display.print(F("T: "));
        display.print(elapsedTime);
        display.println(F(" s"));
        display.println(F(""));
        display.print(F("Temp: "));
        display.print(readTemperature(), 2);
        display.println(F(" C"));
```

```
        display.display();

        // Stop the process if both battery voltages exceed 4.2V
        if (BAT_Voltage1 >= 4.2 && BAT_Voltage2 >= 4.2) {
            display.println(F("Charging Complete"));
            digitalWrite(CHARGING_PIN, LOW); // Disable CHARGING_PIN
            display.display();
            break;
        }

        // Check if BUTTON2_PIN is pressed for 5 seconds to interrupt the
process
        if (digitalRead(BUTTON2_PIN) == HIGH) {
            if (buttonPressStart == 0) {
                buttonPressStart = millis(); // Start tracking button press
time
            } else if (millis() - buttonPressStart >= 1000) { // Button
held for 5 seconds
                display.clearDisplay();
                // Center "Test Interrupted" on the display
                display.setTextSize(1);
                int16_t x1, y1;
                uint16_t w, h;
                display.getTextBounds("Test Interrupted", 0, 0, &x1, &y1,
&w, &h);
                int16_t x = (SCREEN_WIDTH - w) / 2;
                int16_t y = (SCREEN_HEIGHT - h) / 2;
                display.setCursor(x, y);
                display.println(F("Test Interrupted"));
                digitalWrite(CHARGING_PIN, LOW); // Disable CHARGING_PIN
                digitalWrite(LED_RUNNING_PIN,    LOW);    //    Turn    off
LED_RUNNING
                display.display();
                // Ignore BUTTON2_PIN for 3 seconds after interruption
                unsigned long ignoreStart = millis();
                while (millis() - ignoreStart < 3000) {
                    // Wait and ignore BUTTON2_PIN input
                    delay(10);
                }
                break;
            }
        } else {
            buttonPressStart = 0; // Reset button press timer if button is
released
        }
```

```
        // Allow page cycling during charging
        bool button1CurrentState = digitalRead(BUTTON1_PIN);
        if (button1CurrentState == HIGH && button1LastState == LOW) {
            // User wants to cycle page, break out of charging loop
            button1LastState = button1CurrentState;
            break;
        }
        button1LastState = button1CurrentState;

        display.display();
        delay(1000); // Update every second
    }

    digitalWrite(LED_RUNNING_PIN, LOW); // Turn off LED_RUNNING after
process
    digitalWrite(LED_FINISHED_PIN, HIGH); // Turn on LED_FINISHED

    // --- Serial output of all collected charge data ---
    Serial.println(F("Charge Data Log:"));
    for (int i = 0; i < chargeLogCount; i++) {
        Serial.print("t=");
        Serial.print(chargeLogTime[i]);
        Serial.print("s V1=");
        Serial.print(chargeLogV1[i], 3);
        Serial.print("V V2=");
        Serial.print(chargeLogV2[i], 3);
        Serial.println("V");
    }
    Serial.print("Charge Log Count: ");
    Serial.println(chargeLogCount);
}

//=======================================                    DISCHARGING
==========================================

void displayDischargingPage() {
    static bool button1LastState = LOW;
    display.println(F("Press Button 3 to Cycle Options"));
    display.println(F("Press Button 2 to Activate"));

    enum SelectionMode { SELECT_PWM1, SELECT_PWM2, START_PROCESS, RESULTS
};
    static SelectionMode currentMode = SELECT_PWM1; // Initial mode
    bool button3LastState = LOW; // Track the last state of BUTTON3_PIN
    bool button2LastState = LOW; // Track the last state of BUTTON2_PIN
```

```
    static unsigned long lastLogSampleTime = 0;
    unsigned long dischargeStartTime = millis();


    while (true) {
        // Handle button 3 to cycle through selection modes
        bool button3CurrentState = digitalRead(BUTTON3_PIN);
        if (button3CurrentState == HIGH && button3LastState == LOW) {
            currentMode = static_cast<SelectionMode>((currentMode + 1) %
4); // Cycle through modes
        }
        button3LastState = button3CurrentState;

        // Display current mode and values
        display.clearDisplay();
        display.setCursor(0, 0);
        display.println(F("Discharge test"));
        switch (currentMode) {
            case SELECT_PWM1:
                display.println(F("Mode: PWM1"));
                display.print(F("PWM1 Current: "));
                display.print(selectedCurrent);
                display.println(F(" mA"));
                break;
            case SELECT_PWM2:
                display.println(F("Mode: PWM2"));
                display.print(F("PWM2 Current: "));
                display.print(selectedCurrentPWM2);
                display.println(F(" mA"));
                break;
            case START_PROCESS:
                display.println(F("Mode: Start Process"));
                display.println(F("Press Button 2"));
                display.println(F("to Start"));
                break;
            case RESULTS:
                 display.println(F("Results:"));
                   if (lastDischargeTime > 0) {
                       display.print(F("T: "));
                       display.print(lastDischargeTime, 2);
                       display.println(F("s"));
                       display.print(F("C1:"));
                       display.print(lastDischargeCapacity1, 2);
                       display.print(F("mAh C2:"));
                       display.print(lastDischargeCapacity2, 2);
                       display.println(F("mAh"));
```

```
                        display.print(F("V1:"));
                        display.print(lastDischargeVoltage1, 2);
                        display.print(F("V"));
                        display.print(F(" V2:"));
                        display.print(lastDischargeVoltage2, 2);
                        display.println(F("V"));
                    } else {
                        display.println(F("No Discharge Data"));
                    }
                break;
        }
        display.display();

        // Handle button 2 to activate the selected mode
        bool button2CurrentState = digitalRead(BUTTON2_PIN);
        if (button2CurrentState == HIGH && button2LastState == LOW) {
            if (currentMode == SELECT_PWM1) {
                selectedCurrent += 100;
                if (selectedCurrent > 1000) selectedCurrent = 0; // Wrap
to 0 instead of 100
            }
            // When cycling current for PWM2
            else if (currentMode == SELECT_PWM2) {
                selectedCurrentPWM2 += 100;
                if (selectedCurrentPWM2 > 1000) selectedCurrentPWM2 = 0;
// Wrap to 0 instead of 100
            } else if (currentMode == START_PROCESS) {
                // Start the discharging process
                dischargeLogCount = 0;
                display.clearDisplay();
                display.println(F("Discharging..."));
                digitalWrite(LED_FINISHED_PIN, LOW);
                digitalWrite(LED_RUNNING_PIN,   HIGH);   //   Turn   on
LED_RUNNING

                // Use getPWMValue instead of map
                int pwmValue1 = getPWM1Value(selectedCurrent);
                int pwmValue2 = getPWM2Value(selectedCurrentPWM2);
                analogWrite(PWM1_PIN, pwmValue1);
                analogWrite(PWM2_PIN, pwmValue2);

                dischargeLogCount = 0;
                unsigned long dischargingStartTime = millis();
                lastLogSampleTime = dischargingStartTime;

                unsigned long elapsedTime = 0;
```

```
                unsigned long buttonPressStart = 0; // Declare
buttonPressStart here

                while (true) {
                    unsigned long now = millis();
                    // Check temperature
                    if (readTemperature() > TEMP_LIMIT) {
                        display.clearDisplay();
                        display.setTextSize(1);
                        int16_t x1, y1;
                        uint16_t w, h;
                        display.getTextBounds("Temp Exceeded 45C!", 0, 0,
&x1, &y1, &w, &h);
                        int16_t x = (SCREEN_WIDTH - w) / 2;
                        int16_t y = (SCREEN_HEIGHT - h) / 2;
                        display.setCursor(x, y);
                        display.print(F("Temp Exceeded 45C!"));
                        analogWrite(PWM1_PIN, 0);// Disable PWM1_PIN
                        analogWrite(PWM2_PIN, 0);// Disable PWM2_PIN
                        digitalWrite(LED_RUNNING_PIN, LOW);// Turn off
LED_RUNNING
                        digitalWrite(LED_FINISHED_PIN, HIGH);// Turn on
LED_FINISHED
                        display.display();
                        analogWrite(BUZZER_PIN, 128); // 50% duty cycle
(adjust for volume)
                        delay(2000);
                        analogWrite(BUZZER_PIN, 0);   // Turn off buzzer
                        delay(3000);
                        break;
                    }

                    // Calculate elapsed time
                    elapsedTime = (millis() - dischargingStartTime) / 1000;

                    // Always read live voltages
                    float BAT_Voltage1 = measureBatteryVoltage1();
                    float BAT_Voltage2 = measureBatteryVoltage2();
                    lastDischargeVoltage1 = BAT_Voltage1;
                    lastDischargeVoltage2 = BAT_Voltage2;

                    lastDischargeTime = elapsedTime;

                    // Calculate capacity (mAh)
                    float current1 = (selectedCurrent / 1000.0); // Convert
mA to A
```

```
                    float current2 = (selectedCurrentPWM2 / 1000.0); //
Convert mA to A
                    float capacity1 = current1 * (elapsedTime / 3600.0);
// Capacity in Ah
                    float capacity2 = current2 * (elapsedTime / 3600.0);
// Capacity in Ah

                    // Store capacities for display in case 4
                    lastDischargeCapacity1 = capacity1 * 1000; // Convert
to mAh
                    lastDischargeCapacity2 = capacity2 * 1000; // Convert
to mAh

                    // Protect discharge logging arrays with mutex
                    if    ((dischargeLogCount    ==    0)    ||    (now    -
lastLogSampleTime >= SAMPLE_INTERVAL * 1000UL)) {
                        if (dischargeLogCount < PHASE_LOG_POINTS) {
                            if  (xSemaphoreTake(dataMutex,  portMAX_DELAY)
== pdTRUE) {
                                dischargeLogV1[dischargeLogCount]       =
BAT_Voltage1;
                                dischargeLogV2[dischargeLogCount]       =
BAT_Voltage2;
                                dischargeLogTime[dischargeLogCount] = (now
- dischargingStartTime) / 1000;
                                dischargeLogCount++;
                                xSemaphoreGive(dataMutex);
                            }
                            lastLogSampleTime = now;
                        }
                    }

                    // Display discharging information
                    display.clearDisplay();
                    display.setCursor(0, 0);
                    display.println(F("Discharging..."));

                    // V1
                    display.print(F("V1: "));
                    if (BAT_Voltage1 < 1.0) {
                        display.println(F("N/B"));
                    } else {
                        display.print(BAT_Voltage1, 2);
                        display.println(F(" V"));
                    }
```

```
                    // V2
                    display.print(F("V2: "));
                    if (BAT_Voltage2 < 1.0) {
                        display.println(F("N/B"));
                    } else {
                        display.print(BAT_Voltage2, 2);
                        display.println(F(" V"));
                    }

                    // Time
                    display.print(F("T: "));
                    display.print(elapsedTime);
                    display.println(F(" s"));

                    // Cap1
                    display.print(F("Cap1: "));
                    if (BAT_Voltage1 < 1.0) {
                        display.println(F("N/B"));
                    } else {
                        display.print(capacity1 * 1000, 2);
                        display.println(F("mAh"));
                    }

                    // Cap2
                    display.print(F("Cap2: "));
                    if (BAT_Voltage2 < 1.0) {
                        display.println(F("N/B"));
                    } else {
                        display.print(capacity2 * 1000, 2);
                        display.println(F("mAh"));
                    }

                    // Temp
                    display.print(F("Temp: "));
                    display.print(readTemperature(), 2);
                    display.println(F(" C"));
                    display.display();


                    // Turn off PWM1 if BAT_Voltage1 <= 2.5V
                    if   (DischargeVoltageLimitEnabled()   &&   (now   -
dischargeStartTime > 10000)) {
                        if (BAT_Voltage1 <= 2.5) {
                            analogWrite(PWM1_PIN, 0);
                        }
                        if (BAT_Voltage2 <= 2.5) {
```

```
                        analogWrite(PWM2_PIN, 0);
                    }
                    // Finish only when BOTH PWM outputs are 0
                    if     (analogRead(PWM1_PIN)     ==     0     &&
analogRead(PWM2_PIN) == 0) {
                        display.clearDisplay();
                        display.setCursor(0, 0);
                        display.setTextSize(1);
                        display.println(F("Discharge Complete"));
                        display.display();
                        digitalWrite(LED_RUNNING_PIN, LOW);
                        digitalWrite(LED_FINISHED_PIN, HIGH);
                        delay(2000);

                    // --- Serial output of all collected discharge
data ---
                    if  (xSemaphoreTake(dataMutex,  portMAX_DELAY)  ==
pdTRUE) {
                        Serial.println(F("Discharge Data Log:"));
                        for (int i = 0; i < dischargeLogCount; i++) {
                            Serial.print("t=");
                            Serial.print(dischargeLogTime[i]);
                            Serial.print("s V1=");
                            Serial.print(dischargeLogV1[i], 3);
                            Serial.print("V V2=");
                            Serial.print(dischargeLogV2[i], 3);
                            Serial.println("V");
                            Serial.print("Discharge Log Count: ");
                            Serial.println(dischargeLogCount);
                                }
                            xSemaphoreGive(dataMutex);
                            }

                            break;
                        }
                }
                // Check if BUTTON2_PIN is pressed for 5 seconds to
interrupt the process
                if (digitalRead(BUTTON2_PIN) == HIGH) {
                    if (buttonPressStart == 0) {
                        buttonPressStart = millis(); // Start tracking
button press time
                    } else if (millis() - buttonPressStart >= 1000) {
// Button held for 1 seconds
                        display.clearDisplay();
                        // Center "Test Interrupted" on the display
```

```
                            display.setTextSize(1);
                            int16_t x1, y1;
                            uint16_t w, h;
                            display.getTextBounds("Test  Interrupted",  0,
0, &x1, &y1, &w, &h);
                            int16_t x = (SCREEN_WIDTH - w) / 2;
                            int16_t y = (SCREEN_HEIGHT - h) / 2;
                            display.setCursor(x, y);
                            display.println(F("Test Interrupted"));
                            analogWrite(PWM1_PIN, 0); // Disable PWM1_PIN
                            analogWrite(PWM2_PIN, 0); // Disable PWM2_PIN
                            digitalWrite(LED_RUNNING_PIN,  LOW);  //  Turn
off LED_RUNNING
                            display.display();
                            // --- Serial output of all collected discharge
data ---
                            if  (xSemaphoreTake(dataMutex,  portMAX_DELAY)
== pdTRUE) {
                                Serial.println(F("Discharge Data Log:"));
                                for (int i = 0; i < dischargeLogCount; i++)
{
                                    Serial.print("t=");
                                    Serial.print(dischargeLogTime[i]);
                                    Serial.print("s V1=");
                                    Serial.print(dischargeLogV1[i], 3);
                                    Serial.print("V V2=");
                                    Serial.print(dischargeLogV2[i], 3);
                                    Serial.println("V");
                                    Serial.print("Discharge Log Count: ");
                                    Serial.println(dischargeLogCount);
                                }
                                xSemaphoreGive(dataMutex);
                            }
                            // --- end serial output ---
                            //  Ignore  BUTTON2_PIN  for  3  seconds  after
interruption
                            unsigned long ignoreStart = millis();
                            while (millis() - ignoreStart < 3000) {
                                // Wait and ignore BUTTON2_PIN input
                                delay(10);
                            }
                            break;
                        }
                    } else {
                        buttonPressStart = 0; // Reset button press timer
if button is released
                    }
```

```
                    // Allow page cycling during discharging
                    bool button1CurrentState = digitalRead(BUTTON1_PIN);
                    if (button1CurrentState == HIGH && button1LastState ==
LOW) {
                        //  User  wants  to  cycle  page,  break  out  of
discharging loop
                        button1LastState = button1CurrentState;
                        // Turn off PWM1 and PWM2 when leaving the process
                        analogWrite(PWM1_PIN, 0);
                        analogWrite(PWM2_PIN, 0);
                        // --- Serial output of all collected discharge
data ---
                        if  (xSemaphoreTake(dataMutex,  portMAX_DELAY)  ==
pdTRUE) {
                            Serial.println(F("Discharge Data Log:"));
                            for (int i = 0; i < dischargeLogCount; i++) {
                                Serial.print("t=");
                                Serial.print(dischargeLogTime[i]);
                                Serial.print("s V1=");
                                Serial.print(dischargeLogV1[i], 3);
                                Serial.print("V V2=");
                                Serial.print(dischargeLogV2[i], 3);
                                Serial.println("V");
                                Serial.print("Discharge Log Count: ");
                                Serial.println(dischargeLogCount);
                            }
                            xSemaphoreGive(dataMutex);
                        }
                        // --- end serial output ---
                        break;
                    }
                    button1LastState = button1CurrentState;

                    delay(1000); // Update every second
                }

                // Ensure PWM1 and PWM2 are off after the discharging loop
                analogWrite(PWM1_PIN, 0);
                analogWrite(PWM2_PIN, 0);
                digitalWrite(LED_RUNNING_PIN,    LOW);    //    Turn    off
LED_RUNNING after process
                digitalWrite(LED_FINISHED_PIN,   HIGH);   //   Turn   on
LED_FINISHED
                break; // Exit the discharging loop
            }
```

```
        }
        button2LastState = button2CurrentState;

        // Exit the main loop if BUTTON1_PIN is pressed to switch pages
        if (digitalRead(BUTTON1_PIN) == HIGH) {
            break;
        }

        delay(100); // Small delay for button debounce
    }
}


//========================================        COMBINED        PROCESS
==========================================
void displayCombinedProcessPage() {
    static bool button1LastState = LOW;
    // --- Current selection UI (identical to case 2) ---
    enum   SelectionMode   {   SELECT_PWM1,   SELECT_PWM2,   START_PROCESS,
LCharge_Results, LDischarge_Results };
    static SelectionMode currentMode = SELECT_PWM1;
    bool button3LastState = LOW;
    bool button2LastState = LOW;
    unsigned long dischargeStartTime = millis();

    while (true) {
        // Handle button 3 to cycle through selection modes
        bool button3CurrentState = digitalRead(BUTTON3_PIN);
        if (button3CurrentState == HIGH && button3LastState == LOW) {
            currentMode = static_cast<SelectionMode>((currentMode + 1) %
5);
        }
        button3LastState = button3CurrentState;

        // Display current mode and values
        display.clearDisplay();
        display.setCursor(0, 0);
        display.println(F("Combined Test"));
        switch (currentMode) {
            case SELECT_PWM1:
                display.println(F("Mode: PWM1"));
                display.print(F("PWM1 Current: "));
                display.print(selectedCurrent);
                display.println(F(" mA"));
                break;
            case SELECT_PWM2:
```

```
            display.println(F("Mode: PWM2"));
            display.print(F("PWM2 Current: "));
            display.print(selectedCurrentPWM2);
            display.println(F(" mA"));
            break;
        case START_PROCESS:
            display.println(F("Mode: Start Process"));
            display.println(F("Press Button 2 to Start"));
            break;
         case LCharge_Results:
            display.println(F("L Charge Results:"));
                if (chargeLogCount > 0) {
                    display.print(F("T: "));
                    display.print(chargeLogTime[chargeLogCount - 1]);
                    display.println(F("s"));
                    display.print(F("V1:"));
                    display.print(chargeLogV1[chargeLogCount - 1], 2);
                    display.print(F("V V2:"));
                    display.print(chargeLogV2[chargeLogCount - 1], 2);
                    display.println(F("V"));
                } else {
                    display.println(F("No Charge Data"));
                }
            break;
        case LDischarge_Results:
             display.println(F("L Discharge Results:"));
                if (lastDischargeTime > 0) {
                    display.print(F("T: "));
                    display.print(lastDischargeTime, 2);
                    display.println(F("s"));
                    display.print(F("C1:"));
                    display.print(lastDischargeCapacity1, 2);
                    display.print(F("mAh C2:"));
                    display.print(lastDischargeCapacity2, 2);
                    display.println(F("mAh"));
                    display.print(F("V1:"));
                    display.print(lastDischargeVoltage1, 2);
                    display.print(F("V"));
                    display.print(F(" V2:"));
                    display.print(lastDischargeVoltage2, 2);
                    display.println(F("V"));
                } else {
                    display.println(F("No Discharge Data"));
                }
            break;
    }
    display.display();
```

```
        // Handle button 2 to activate the selected mode
        bool button2CurrentState = digitalRead(BUTTON2_PIN);
        if (button2CurrentState == HIGH && button2LastState == LOW) {
            if (currentMode == SELECT_PWM1) {
                selectedCurrent += 100;
                if (selectedCurrent > 1000) selectedCurrent = 0; // Wrap
to 0 instead of 100
            } else if (currentMode == SELECT_PWM2) {
                selectedCurrentPWM2 += 100;
                if (selectedCurrentPWM2 > 1000) selectedCurrentPWM2 = 0;
// Wrap to 0 instead of 100
            } else if (currentMode == START_PROCESS) {
                break; // Exit selection loop and start process
            }
        }
        button2LastState = button2CurrentState;

        // Allow page cycling
        if (digitalRead(BUTTON1_PIN) == HIGH) {
            return;
        }
        delay(100);
    }

    // --- Charging phase (identical to case 1) ---
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println(F("Charging..."));
    display.display();
    digitalWrite(LED_RUNNING_PIN, HIGH);
    digitalWrite(LED_FINISHED_PIN, LOW);
    digitalWrite(CHARGING_PIN, HIGH);
    unsigned long chargingStartTime = millis();
    unsigned long elapsedTime = 0;
    unsigned long lastLogSampleTime = chargingStartTime;
    unsigned long buttonPressStart = 0;



    chargeLogCount = 0;

    bool chargingDone = false;
    while (!chargingDone) {
        if (readTemperature() > TEMP_LIMIT) {
            display.clearDisplay();
```

```
            display.setTextSize(1);
            int16_t x1, y1;
            uint16_t w, h;
            display.getTextBounds("Temp Exceeded 45C!", 0, 0, &x1, &y1, &w,
&h);
            int16_t x = (SCREEN_WIDTH - w) / 2;
            int16_t y = (SCREEN_HEIGHT - h) / 2;
            display.setCursor(x, y);
            display.print(F("Temp Exceeded 45C!"));
            analogWrite(PWM1_PIN, 0);
            analogWrite(PWM2_PIN, 0);
            digitalWrite(LED_RUNNING_PIN, LOW);
            digitalWrite(LED_FINISHED_PIN, HIGH);
            display.display();
            analogWrite(BUZZER_PIN, 128); // 50% duty cycle (adjust for
volume)
            delay(2000);
            analogWrite(BUZZER_PIN, 0);   // Turn off buzzer
            delay(3000);
            return;
        }

        float BAT_Voltage1 = measureBatteryVoltage1();
        float BAT_Voltage2 = measureBatteryVoltage2();

        // Log data every SAMPLE_INTERVAL seconds
        unsigned long now = millis();
        if ((chargeLogCount == 0) || (now - lastLogSampleTime >=
SAMPLE_INTERVAL * 1000UL)) {
            if (chargeLogCount < PHASE_LOG_POINTS) {
                chargeLogV1[chargeLogCount] = BAT_Voltage1;
                chargeLogV2[chargeLogCount] = BAT_Voltage2;
                chargeLogTime[chargeLogCount] = (now - chargingStartTime)
/ 1000;
                chargeLogCount++;
                lastLogSampleTime = now;
            }
        }

        elapsedTime = (millis() - chargingStartTime) / 1000;



        display.clearDisplay();
        display.setCursor(0, 0);
        display.println(F("Charging..."));
```

```
display.println(F(""));

display.print(F("V1: "));
if (BAT_Voltage1 < 1.0) {
    display.println(F("N/B"));
} else {
    display.print(BAT_Voltage1, 2);
    display.println(F(" V"));
}

display.print(F("V2: "));
if (BAT_Voltage2 < 1.0) {
    display.println(F("N/B"));
} else {
    display.print(BAT_Voltage2, 2);
    display.println(F(" V"));
}

display.println(F(""));
display.print(F("T: "));
display.print(elapsedTime);
display.println(F(" s"));
display.println(F(""));
display.print(F("Temp: "));
display.print(readTemperature(), 2);
display.println(F(" C"));

display.display();
if (BAT_Voltage1 >= 4.2 && BAT_Voltage2 >= 4.2) {
    display.println(F("Charging Complete"));
    digitalWrite(CHARGING_PIN, LOW);
    chargingDone = true;
    delay(1000);
    break;
}

// --- DEBUG: allow skipping to discharge phase with BUTTON3 ---
#if DEBUG_MODE
checkDebugSkipToDischarge(chargingDone);
if (chargingDone) {
    digitalWrite(CHARGING_PIN, LOW);
    break;
}
#endif

if (digitalRead(BUTTON2_PIN) == HIGH) {
```

```
            if (buttonPressStart == 0) {
                buttonPressStart = millis();
            } else if (millis() - buttonPressStart >= 1000) {
                display.clearDisplay();
                display.setTextSize(1);
                int16_t x1, y1;
                uint16_t w, h;
                display.getTextBounds("Test Interrupted", 0, 0, &x1, &y1,
&w, &h);
                int16_t x = (SCREEN_WIDTH - w) / 2;
                int16_t y = (SCREEN_HEIGHT - h) / 2;
                display.setCursor(x, y);
                display.println(F("Test Interrupted"));
                digitalWrite(CHARGING_PIN, LOW);
                digitalWrite(LED_RUNNING_PIN, LOW);
                display.display();
                // Ignore BUTTON2_PIN for 3 seconds after interruption
                unsigned long ignoreStart = millis();
                while (millis() - ignoreStart < 3000) {
                    delay(10);
                }
                return;
            }
        } else {
            buttonPressStart = 0;
        }

        bool button1CurrentState = digitalRead(BUTTON1_PIN);
        if (button1CurrentState == HIGH && button1LastState == LOW) {
            button1LastState = button1CurrentState;
            digitalWrite(CHARGING_PIN, LOW);
            return;
        }
        button1LastState = button1CurrentState;

        delay(1000);
    }

    // --- Serial output of all collected charge data ---
    Serial.println(F("Charge Data Log:"));
    for (int i = 0; i < chargeLogCount; i++) {
        Serial.print("t=");
        Serial.print(chargeLogTime[i]);
        Serial.print("s V1=");
        Serial.print(chargeLogV1[i], 3);
        Serial.print("V V2=");
```

```
        Serial.print(chargeLogV2[i], 3);
        Serial.println("V");
    }
    Serial.print("Charge Log Count: ");
    Serial.println(chargeLogCount);


    // --- Add 5 minute resting delay between charge and discharge ---
display.clearDisplay();
display.setTextSize(2);
display.setCursor(10, 25);
display.print(F("Resting..."));
display.display();

unsigned long restStart = millis();
const unsigned long restDuration = 5UL * 60UL * 1000UL; // 5 minutes in ms
while (millis() - restStart < restDuration) {
    // Optionally, show countdown
    unsigned long secondsLeft = (restDuration - (millis() - restStart)) /
1000;
    display.setTextSize(1);
    display.setCursor(10, 50);
    display.print(F("Left: "));
    display.print(secondsLeft);
    display.print(F(" s   "));
    display.display();
    delay(500);
}



    // --- Discharging phase (identical to case 2) ---
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println(F("Discharging..."));
    display.display();
    // Use getPWMValue instead of map
    int pwmValue1 = getPWM1Value(selectedCurrent);
    int pwmValue2 = getPWM2Value(selectedCurrentPWM2);
    digitalWrite(LED_FINISHED_PIN, LOW);
    analogWrite(PWM1_PIN, pwmValue1);
    analogWrite(PWM2_PIN, pwmValue2);
    unsigned long dischargingStartTime = millis();
    unsigned long dischargeElapsedTime = 0;
    unsigned long dischargeButtonPressStart = 0;
```

```
    static unsigned long lastDischargeLogSampleTime = 0;
    dischargeLogCount = 0;
    lastDischargeLogSampleTime = dischargingStartTime;

    while (true) {

        unsigned long now = millis();
        if (readTemperature() > TEMP_LIMIT) {
            display.clearDisplay();
            display.setTextSize(1);
            int16_t x1, y1;
            uint16_t w, h;
            display.getTextBounds("Temp Exceeded 45C!", 0, 0, &x1, &y1, &w,
&h);
            int16_t x = (SCREEN_WIDTH - w) / 2;
            int16_t y = (SCREEN_HEIGHT - h) / 2;
            display.setCursor(x, y);
            display.print(F("Temp Exceeded 45C!"));
            analogWrite(PWM1_PIN, 0);
            analogWrite(PWM2_PIN, 0);
            digitalWrite(LED_RUNNING_PIN, LOW);
            digitalWrite(LED_FINISHED_PIN, HIGH);
            display.display();
            analogWrite(BUZZER_PIN, 128); // 50% duty cycle (adjust for
volume)
            delay(2000);
            analogWrite(BUZZER_PIN, 0);   // Turn off buzzer
            delay(3000);
            break;
        }

        dischargeElapsedTime = (millis() - dischargingStartTime) / 1000;

        float BAT_Voltage1 = measureBatteryVoltage1();
        float BAT_Voltage2 = measureBatteryVoltage2();
        lastDischargeVoltage1 = BAT_Voltage1;
        lastDischargeVoltage2 = BAT_Voltage2;

        // Log data every SAMPLE_INTERVAL seconds

        if ((dischargeLogCount == 0) || (now - lastDischargeLogSampleTime
>= SAMPLE_INTERVAL * 1000UL)) {
            if (dischargeLogCount < PHASE_LOG_POINTS) {
                if (xSemaphoreTake(dataMutex, portMAX_DELAY) == pdTRUE) {
                    dischargeLogV1[dischargeLogCount] = BAT_Voltage1;
                    dischargeLogV2[dischargeLogCount] = BAT_Voltage2;
```

```
                    dischargeLogTime[dischargeLogCount]    =    (now    -
dischargingStartTime) / 1000;
                    dischargeLogCount++;
                    xSemaphoreGive(dataMutex);
                }
                lastDischargeLogSampleTime = now;
            }
        }

        float current1 = (selectedCurrent / 1000.0);
        float current2 = (selectedCurrentPWM2 / 1000.0);
        float capacity1 = current1 * (dischargeElapsedTime / 3600.0);
        float capacity2 = current2 * (dischargeElapsedTime / 3600.0);

        lastDischargeTime = dischargeElapsedTime;
        lastDischargeCapacity1 = capacity1 * 1000;
        lastDischargeCapacity2 = capacity2 * 1000;

        display.clearDisplay();
                    display.setCursor(0, 0);
                    display.println(F("Discharging..."));

                    // V1
                    display.print(F("V1: "));
                    if (BAT_Voltage1 < 1.0) {
                        display.println(F("N/B"));
                    } else {
                        display.print(BAT_Voltage1, 2);
                        display.println(F(" V"));
                    }

                    // V2
                    display.print(F("V2: "));
                    if (BAT_Voltage2 < 1.0) {
                        display.println(F("N/B"));
                    } else {
                        display.print(BAT_Voltage2, 2);
                        display.println(F(" V"));
                    }

                    // Time
                    display.print(F("T: "));
                    display.print(elapsedTime);
                    display.println(F(" s"));

                    // Cap1
```

```
                    display.print(F("Cap1: "));
                    if (BAT_Voltage1 < 1.0) {
                        display.println(F("N/B"));
                    } else {
                        display.print(capacity1 * 1000, 2);
                        display.println(F("mAh"));
                    }

                    // Cap2
                    display.print(F("Cap2: "));
                    if (BAT_Voltage2 < 1.0) {
                        display.println(F("N/B"));
                    } else {
                        display.print(capacity2 * 1000, 2);
                        display.println(F("mAh"));
                    }

                    // Temp
                    display.print(F("Temp: "));
                    display.print(readTemperature(), 2);
                    display.println(F(" C"));
                    display.display();

        // Turn off PWM1 if BAT_Voltage1 <= 2.5V
        if (DischargeVoltageLimitEnabled() && (now - dischargeStartTime >
10000)) {
            if (BAT_Voltage1 <= 2.5) {
                analogWrite(PWM1_PIN, 0);
            }
            if (BAT_Voltage2 <= 2.5) {
                analogWrite(PWM2_PIN, 0);
            }
            // Finish only when BOTH PWM outputs are 0
            if (analogRead(PWM1_PIN) == 0 && analogRead(PWM2_PIN) == 0) {
                display.clearDisplay();
                display.setCursor(0, 0);
                display.setTextSize(1);
                display.println(F("Discharge Complete"));
                display.display();
                digitalWrite(LED_RUNNING_PIN, LOW);
                digitalWrite(LED_FINISHED_PIN, HIGH);
                delay(2000);
            // --- Serial output of all collected discharge data ---
            if (xSemaphoreTake(dataMutex, portMAX_DELAY) == pdTRUE) {
                Serial.println(F("Discharge Data Log:"));
                for (int i = 0; i < dischargeLogCount; i++) {
```

```
                    Serial.print("t=");
                    Serial.print(dischargeLogTime[i]);
                    Serial.print("s V1=");
                    Serial.print(dischargeLogV1[i], 3);
                    Serial.print("V V2=");
                    Serial.print(dischargeLogV2[i], 3);
                    Serial.println("V");
                    Serial.print("Discharge Log Count: ");
                    Serial.println(dischargeLogCount);
                        }
                        xSemaphoreGive(dataMutex);
                    }

                    break;
                }
        }

        if (digitalRead(BUTTON2_PIN) == HIGH) {
            if (dischargeButtonPressStart == 0) {
                dischargeButtonPressStart = millis();
            } else if (millis() - dischargeButtonPressStart >= 1000) {
                display.clearDisplay();
                display.setTextSize(1);
                int16_t x1, y1;
                uint16_t w, h;
                display.getTextBounds("Test Interrupted", 0, 0, &x1, &y1,
&w, &h);
                int16_t x = (SCREEN_WIDTH - w) / 2;
                int16_t y = (SCREEN_HEIGHT - h) / 2;
                display.setCursor(x, y);
                display.println(F("Test Interrupted"));
                analogWrite(PWM1_PIN, 0);
                analogWrite(PWM2_PIN, 0);
                digitalWrite(LED_RUNNING_PIN, LOW);
                display.display();
                unsigned long ignoreStart = millis();
                while (millis() - ignoreStart < 3000) {
                    delay(10);
                }
                break;
            }
        } else {
            dischargeButtonPressStart = 0;
        }

        bool button1CurrentState = digitalRead(BUTTON1_PIN);
```

```
        if (button1CurrentState == HIGH && button1LastState == LOW) {
            button1LastState = button1CurrentState;
            analogWrite(PWM1_PIN, 0);
            analogWrite(PWM2_PIN, 0);
            break;
        }
        button1LastState = button1CurrentState;

        delay(1000);
    }

    // --- Serial output of all collected discharge data ---
    Serial.println(F("Discharge Data Log:"));
    for (int i = 0; i < dischargeLogCount; i++) {
        Serial.print("t=");
        Serial.print(dischargeLogTime[i]);
        Serial.print("s V1=");
        Serial.print(dischargeLogV1[i], 3);
        Serial.print("V V2=");
        Serial.print(dischargeLogV2[i], 3);
        Serial.println("V");
    }
    Serial.print("Discharge Log Count: ");
    Serial.println(dischargeLogCount);

    analogWrite(PWM1_PIN, 0);
    analogWrite(PWM2_PIN, 0);
    digitalWrite(LED_RUNNING_PIN, LOW);
    digitalWrite(LED_FINISHED_PIN, HIGH);

    // Store the latest charge/discharge data for results/plot
    lastChargeTime = elapsedTime;
    lastDischargeTime = dischargeElapsedTime;
    lastDischargeCapacity1    =    (selectedCurrent    /    1000.0)    *
(dischargeElapsedTime / 3600.0) * 1000.0;
    lastDischargeCapacity2    =    (selectedCurrentPWM2    /    1000.0)    *
(dischargeElapsedTime / 3600.0) * 1000.0;


}


//=======================================   INTERNAL   RESISTANCE   PAGE
==========================================
void displayInternalResistancePage() {
    // Always use max current for IR test
```

```
const int irSelectedCurrent = 1000;
const int irSelectedCurrentPWM2 = 1000;

// Wait for user to press Button 2 to start the test
display.clearDisplay();
display.setCursor(0, 0);
display.println(F("IR Test"));
display.println(F("Press Button 2"));
display.println(F("to Start"));
display.display();

// Wait for BUTTON2_PIN to be pressed
while (digitalRead(BUTTON2_PIN) == LOW) {
    if (digitalRead(BUTTON1_PIN) == HIGH) return; // Allow exit
    delay(50);
}
// Debounce: wait for release
while (digitalRead(BUTTON2_PIN) == HIGH) {
    if (digitalRead(BUTTON1_PIN) == HIGH) return;
    delay(10);
}

// --- Internal Resistance Measurement for PWM1 ---
float voltageNoLoad1 = 0, voltageLoad1 = 0, ir1 = 0;
// Use getPWMValue for IR test
int pwmValue1 = getPWM1Value(selectedCurrent);
analogWrite(PWM1_PIN, 0);
delay(1000);
voltageNoLoad1 = measureBatteryVoltage1();
analogWrite(PWM1_PIN, pwmValue1);
delay(1000);
voltageLoad1 = measureBatteryVoltage1();
float currentDrawn1 = irSelectedCurrent / 1000.0;
if (currentDrawn1 > 0) {
    ir1 = (voltageNoLoad1 - voltageLoad1) / currentDrawn1;
} else {
    ir1 = 0;
}
analogWrite(PWM1_PIN, 0);

// --- Internal Resistance Measurement for PWM2 ---
float voltageNoLoad2 = 0, voltageLoad2 = 0, ir2 = 0;
int pwmValue2 = getPWM2Value(selectedCurrentPWM2);
analogWrite(PWM2_PIN, 0);
delay(1000);
voltageNoLoad2 = measureBatteryVoltage2();
```

```
analogWrite(PWM2_PIN, pwmValue2);
delay(1000);
voltageLoad2 = measureBatteryVoltage2();
float currentDrawn2 = irSelectedCurrentPWM2 / 1000.0;
if (currentDrawn2 > 0) {
    ir2 = (voltageNoLoad2 - voltageLoad2) / currentDrawn2;
} else {
    ir2 = 0;
}
analogWrite(PWM2_PIN, 0);

// --- Display Results ---
display.clearDisplay();
display.setTextSize(1);
display.setCursor(0, 0);
display.println(F("IR Test Results:"));

// BAT1
if (voltageNoLoad1 < 1.0 || voltageLoad1 < 1.0) {
    display.print(F("BAT1: N/B\n"));
} else {
    display.print(F("BAT1: "));
    display.print(ir1, 3);
    display.println(F(" Ohm"));
    display.print(F("Vnl:"));
    display.print(voltageNoLoad1, 3);
    display.print(F("V Vl:"));
    display.print(voltageLoad1, 3);
    display.println(F("V"));
}
display.println(F(""));
// BAT2
if (voltageNoLoad2 < 1.0 || voltageLoad2 < 1.0) {
    display.print(F("BAT2: N/B\n"));
} else {
    display.print(F("BAT2: "));
    display.print(ir2, 3);
    display.println(F(" Ohm"));
    display.print(F("Vnl:"));
    display.print(voltageNoLoad2, 3);
    display.print(F("V Vl:"));
    display.print(voltageLoad2, 3);
    display.println(F("V"));
}

display.display();
```

```
    delay(20000); // Wait for user to read
}


//=======================================           PLOT           PAGE
==========================================


void displayCombinedPlotPage() {
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);

    // Draw axes
    int x0 = 10, y0 = 54, x1 = 118, y1 = 10;
    display.drawLine(x0, y0, x0, y1, SSD1306_WHITE); // Y-axis
    display.drawLine(x0, y0, x1, y0, SSD1306_WHITE); // X-axis

    // --- Plot charge phase data (V1 and V2) ---
    int nCharge = chargeLogCount;
    float tmaxC = 1, vmaxC = 0, vminC = 0;
    if (nCharge > 1) {
        tmaxC = chargeLogTime[nCharge-1];
        vmaxC = -1000, vminC = 1000;
        for (int i = 0; i < nCharge; i++) {
            if (chargeLogV1[i] > vmaxC) vmaxC = chargeLogV1[i];
            if (chargeLogV2[i] > vmaxC) vmaxC = chargeLogV2[i];
            if (chargeLogV1[i] < vminC) vminC = chargeLogV1[i];
            if (chargeLogV2[i] < vminC) vminC = chargeLogV2[i];
        }
        if (vmaxC == vminC) vmaxC = vminC + 0.1;
    }

    // --- Plot discharge phase data (V1 and V2) ---
    int nDischarge = 0;
    float v1D[PHASE_LOG_POINTS], v2D[PHASE_LOG_POINTS];
    unsigned long tD[PHASE_LOG_POINTS];
    float tmaxD = 1, vmaxD = 0, vminD = 0;
    if (xSemaphoreTake(dataMutex, portMAX_DELAY) == pdTRUE) {
        nDischarge = dischargeLogCount;
        for (int i = 0; i < nDischarge; i++) {
            v1D[i] = dischargeLogV1[i];
            v2D[i] = dischargeLogV2[i];
            tD[i] = dischargeLogTime[i];
        }
        xSemaphoreGive(dataMutex);
```

```
    }
    if (nDischarge > 1) {
        tmaxD = tD[nDischarge-1];
        vmaxD = -1000, vminD = 1000;
        for (int i = 0; i < nDischarge; i++) {
            if (v1D[i] > vmaxD) vmaxD = v1D[i];
            if (v2D[i] > vmaxD) vmaxD = v2D[i];
            if (v1D[i] < vminD) vminD = v1D[i];
            if (v2D[i] < vminD) vminD = v2D[i];
        }
        if (vmaxD == vminD) vmaxD = vminD + 0.1;
    }

    // --- Determine global min/max for scaling ---
    float tmax = max(tmaxC, tmaxD);
    if (lastChargeTime > tmax) tmax = lastChargeTime;
    if (lastDischargeTime > tmax) tmax = lastDischargeTime;
    float vmax = max(vmaxC, vmaxD);
    float vmin = min(vminC, vminD);
    if (vmax - vmin < 0.1) { vmax += 0.2; vmin -= 0.2; }
    if (tmax < 1) tmax = 1;

    // --- Serial Plotter Output for all data ---

Serial.println(F("time_charge,V1_charge,V2_charge,time_discharge,V1_disch
arge,V2_discharge"));
    int maxPoints = max(nCharge, nDischarge);
    for (int i = 0; i < maxPoints; i++) {
        // Print charge data if available
        if (i < nCharge) {
            Serial.print(chargeLogTime[i]);
            Serial.print(",");
            Serial.print(chargeLogV1[i], 4);
            Serial.print(",");
            Serial.print(chargeLogV2[i], 4);
        } else {
            Serial.print(",,");
        }
        Serial.print(",");
        // Print discharge data if available
        if (i < nDischarge) {
            Serial.print(tD[i]);
            Serial.print(",");
            Serial.print(v1D[i], 4);
            Serial.print(",");
            Serial.print(v2D[i], 4);
```

```
    } else {
        Serial.print(",,");
    }
    Serial.println();
}

// --- Plot charge V1 (solid line) ---
if (nCharge > 1) {
    for (int i = 0; i < nCharge-1; i++) {
        int xA = map(chargeLogTime[i], 0, tmax, x0, x1);
        int yA = map(chargeLogV1[i], vmin, vmax, y0, y1);
        int xB = map(chargeLogTime[i+1], 0, tmax, x0, x1);
        int yB = map(chargeLogV1[i+1], vmin, vmax, y0, y1);
        display.drawLine(xA, yA, xB, yB, SSD1306_WHITE);
    }
}
// --- Plot charge V2 (dotted line) ---
if (nCharge > 1) {
    for (int i = 0; i < nCharge-1; i++) {
        int xA = map(chargeLogTime[i], 0, tmax, x0, x1);
        int yA = map(chargeLogV2[i], vmin, vmax, y0, y1);
        int xB = map(chargeLogTime[i+1], 0, tmax, x0, x1);
        int yB = map(chargeLogV2[i+1], vmin, vmax, y0, y1);
        for (float t = 0; t < 1.0; t += 0.05) {
            int x = xA + (xB - xA) * t;
            int y = yA + (yB - yA) * t;
            display.drawPixel(x, y, SSD1306_WHITE);
        }
    }
}
// --- Plot discharge V1 (solid line) ---
if (nDischarge > 1) {
    for (int i = 0; i < nDischarge-1; i++) {
        int xA = map(tD[i], 0, tmax, x0, x1);
        int yA = map(v1D[i], vmin, vmax, y0, y1);
        int xB = map(tD[i+1], 0, tmax, x0, x1);
        int yB = map(v1D[i+1], vmin, vmax, y0, y1);
        display.drawLine(xA, yA, xB, yB, SSD1306_WHITE);
    }
}
// --- Plot discharge V2 (dotted line) ---
if (nDischarge > 1) {
    for (int i = 0; i < nDischarge-1; i++) {
        int xA = map(tD[i], 0, tmax, x0, x1);
        int yA = map(v2D[i], vmin, vmax, y0, y1);
        int xB = map(tD[i+1], 0, tmax, x0, x1);
        int yB = map(v2D[i+1], vmin, vmax, y0, y1);
```

```
            for (float tt = 0; tt < 1.0; tt += 0.2) {
                int x = xA + (xB - xA) * tt;
                int y = yA + (yB - yA) * tt;
                display.drawPixel(x, y, SSD1306_WHITE);
            }
        }
    }

    // --- Add labels ---
    display.setCursor(0, 0);
    display.print(F("Charge/Discharge Plot"));
    display.setCursor(10, 56);
    display.print(F("Time (s)"));
    display.setCursor(70, 56);
    display.print(tmax);
    display.print("s");

    display.display();

    // Allow page cycling or auto-exit after 5 seconds
    unsigned long startTime = millis();
    static bool button1LastState = LOW;
    while (millis() - startTime < 5000) {
        bool button1CurrentState = digitalRead(BUTTON1_PIN);
        if (button1CurrentState == HIGH && !button1LastState) {
            button1LastState = button1CurrentState;
            break;
        }
        button1LastState = button1CurrentState;
        delay(10);
    }
}

float readTemperature() {
    int analogValue = analogRead(THERMISTOR_PIN);
    if (analogValue <= 0 || analogValue >= 1023) {
        // Return an error value or a safe default
        return -1000.0;
    }
    float resistance = SERIES_RESISTOR / ((1023.0 / analogValue) - 1);
    if (resistance <= 0) {
        return -1000.0;
    }
    float celsius = 1 / (log(resistance / THERMISTOR_NOMINAL) / BETA + 1.0
/ NOMINAL_TEMP_KELVIN) - 273.15;
    return celsius;
```

```
}

float readVCC() {
    long adcSum = 0;

    // Take multiple samples for noise reduction
    for (int i = 0; i < NUM_SAMPLES; i++) {
        int adcReading = analogRead(VREF_PIN);  // 10-bit reading (0-1023)
        adcSum += adcReading;
        delay(1);  // Short delay between reads
    }

    float adcAverage = adcSum / (float)NUM_SAMPLES;
    if (adcAverage <= 0) {
        // Return an error value or a safe default
        return 0.0;
    }

    // Calculate VCC: VCC = (VREF * 1024) / ADC_Reading
    float vcc = (VREF * 1024.0) / adcAverage;
    return vcc;
}
```