Infineon Mobile Robot (IMR) battery management control

Using DEMO_IMR_BMSCTRL_V1

About this document

Scope and purpose

This document describes the DEMO_IMR_BMSCTRL_V1 Board along with its hardware and software components. Furthermore, the “Quick start guide” section provides guidance on how to use the board as a battery management module for the Infineon Mobile Robot.

Intended audience

This document is intended for design engineers, technicians, and developers of electronic systems.

Infineon components featured

- OptiMOS™ BSD235N 2 small-signal transistors
- IRLML6401 single P-channel power MOSFET
- PSOC™ 6 CY8C6245AZI-S3D72, 32-bit dual-CPU microcontroller with Arm® Cortex®-M4F and M0+
- CY15B256Q-SXA 256 Kb automotive serial (SPI) FRAM

Important notice

Safety & Operating Instructions:

Customer shall check the Evaluation Board for any physical damage which may have occurred during transport. If customer detects any damages or defects in the Evaluation Board, customer shall not connect the Evaluation Board to a power source. Customer shall contact Infineon for further support. If customer observes unusual operating behavior during the evaluation process, customer shall immediately shut off the power supply to the Evaluation Board and consult Infineon for support.

Customer shall not touch the Evaluation Board during operation and keep a safe distance.

Customer shall not touch the Evaluation Board after disconnecting the power supply, several components may still store electrical voltage and can discharge through physical contact. Several parts, like heat sinks and transformers, may still be very hot. Allow the components to cool before touching or servicing.

The electrical installation must be completed in accordance with the appropriate safety requirements.

Safety precautions

Note: *Please note the following warnings regarding the hazards associated with development systems.*

Table 1 Safety precautions

⚠	**Warning:** *The DC link potential of this board is up to 1000 VDC. When measuring voltage waveforms by oscilloscope, high-voltage differential probes must be used. Failure to do so may result in personal injury or death.*
⚠	**Warning**: *The evaluation or reference board contains DC bus capacitors, which take time to discharge after removal of the main supply. Before working on the drive system, wait 5 minutes for capacitors to discharge to safe voltage levels. Failure to do so may result in personal injury or death. Darkened display LEDs are not an indication that capacitors have discharged to safe voltage levels.*
⚠	**Warning:** *The evaluation or reference board is connected to the grid input during testing. Hence, high-voltage differential probes must be used when measuring voltage waveforms by an oscilloscope. Failure to do so may result in personal injury or death. Darkened display LEDs are not an indication that capacitors have discharged to safe voltage levels.*
⚠	**Warning:** *Remove or disconnect power from the drive before you disconnect or reconnect wires, or perform maintenance work. Wait five minutes after removing power to discharge the bus capacitors. Do not attempt to service the drive until the bus capacitors have discharged to zero. Failure to do so may result in personal injury or death.*
⚠	**Caution:** *The heat sink and device surfaces of the evaluation or reference board may become hot during testing. Hence, necessary precautions are required while handling the board. Failure to comply may cause injury.*
⚠	**Caution:** *Only personnel familiar with the drive, power electronics and associated machinery should plan, install, commission and subsequently service the system. Failure to comply may result in personal injury and/or equipment damage.*
⚠	**Caution:** *The evaluation or reference board contains parts and assemblies sensitive to electrostatic discharge (ESD). Electrostatic control precautions are required when installing, testing, servicing or repairing the assembly. Component damage may result if ESD control procedures are not followed. If you are not familiar with electrostatic control procedures, refer to the applicable ESD protection handbooks and guidelines.*
⚠	**Caution:** *A drive that is incorrectly applied or installed can lead to component damage or reduction in product lifetime. Wiring or application errors such as undersizing the motor, supplying an incorrect or inadequate AC supply, or excessive ambient temperatures may result in system malfunction.*
⚠	**Caution:** *The evaluation or reference board is shipped with packing materials that need to be removed prior to installation. Failure to remove all packing materials that are unnecessary for system installation may result in overheating or abnormal operating conditions.*

Table of contents

About this document 1
Important notice 2
Safety precautions 3
Table of contents 4
1 Introduction 6
1.1 Mobile robot general description 6
1.2 Infineon Mobile Robot (IMR) 6
1.3 DEMO_IMR_BMSCTRL_V1 Demo Board 8
1.4 Connectors 9
2 Hardware overview 13
2.1 PSOC™ 6 MCU 14
2.2 Piezoelectric buzzer circuit 15
2.3 Serial FRAM memory circuit 16
2.4 ePaper display circuitry 17
2.5 Unique identification circuit 17
3 Software 19
3.1 Software architecture 19
3.1.1 Software structure and settings 19
3.1.2 Main design 19
3.1.1 Initialization 22
3.1.2 Measurement 23
3.1.2.1 Analog frontend measurement 23
3.1.2.2 ADC measurement 24
3.1.3 Diagnostic system 24
3.1.3.1 Analog frontend diagnostic 26
3.1.3.2 Safety switch diagnostic 27
3.1.3.3 Internal diagnostic 27
3.1.3.4 Error codes 27
3.1.4 Safety switch control and state decision 28
3.1.4.1 Dynamic pre-charge and switch-on 29
3.1.4.2 Hot-swap 30
3.1.5 Balancing 32
3.1.6 Cell monitoring and state 34
3.1.6.1 State of charge (SoC) 34
3.1.6.2 State of health (SoH) 34
3.1.6.3 Usable remaining time 34
3.1.7 Memory 35
3.1.8 Shutdown and start 37
3.1.9 Watchdog 37
3.1.10 Slot detection 37
3.1.11 Analog frontend library 38
3.1.11.1 Interface library design 38
3.1.11.2 Commands and data access 40
3.1.11.3 Queue system 40
3.1.11.4 Interpreter design 41
3.1.12 User interface on secondary core 44
3.1.13 CAN communication 46

4 Quick start guide........49
4.1 Software deployment and debugging........49
4.2 Debugging using onboard UART communication........51
4.3 Setting the identification........52
4.4 BMS control buttons and LEDs........53
4.5 Charging and discharging using the BMS interfacing board........54
5 Bill of materials (BOM)........56
6 PCB layout........58
7 Appendix........60
7.1 Schematics........60
References........68
Glossary........69
Revision history........71
Disclaimer........72

1 Introduction

1.1 Mobile robot general description

Mobile robots are now common in applications from logistics and warehouse centers and production sites to hospitals, restaurants, and schools, and as last-mile package delivery vehicles. Essentially, mobile robots are of two main types:

- **Automated guided vehicles (AGV):** “Fixed” vehicles that follow predefined paths using lasers, barcodes, radio waves, vision sensors, or magnetic tape for navigation
- **Autonomous mobile robots (AMR):** Not “fixed”, and do not need external paths as these robots use autonomous mapping, localization, navigation, and obstacle avoidance by using sensors

Usually, robots are battery-powered where the voltage level depends on the size and weight characteristics.

1.2 Infineon Mobile Robot (IMR)

The board described in this document is primarily targeted to be used in combination with the Infineon Mobile Robot (IMR). The IMR is a comprehensive robotic platform for use with a wide variety of boards such as sensors, motor controls, wireless communication, and battery management.

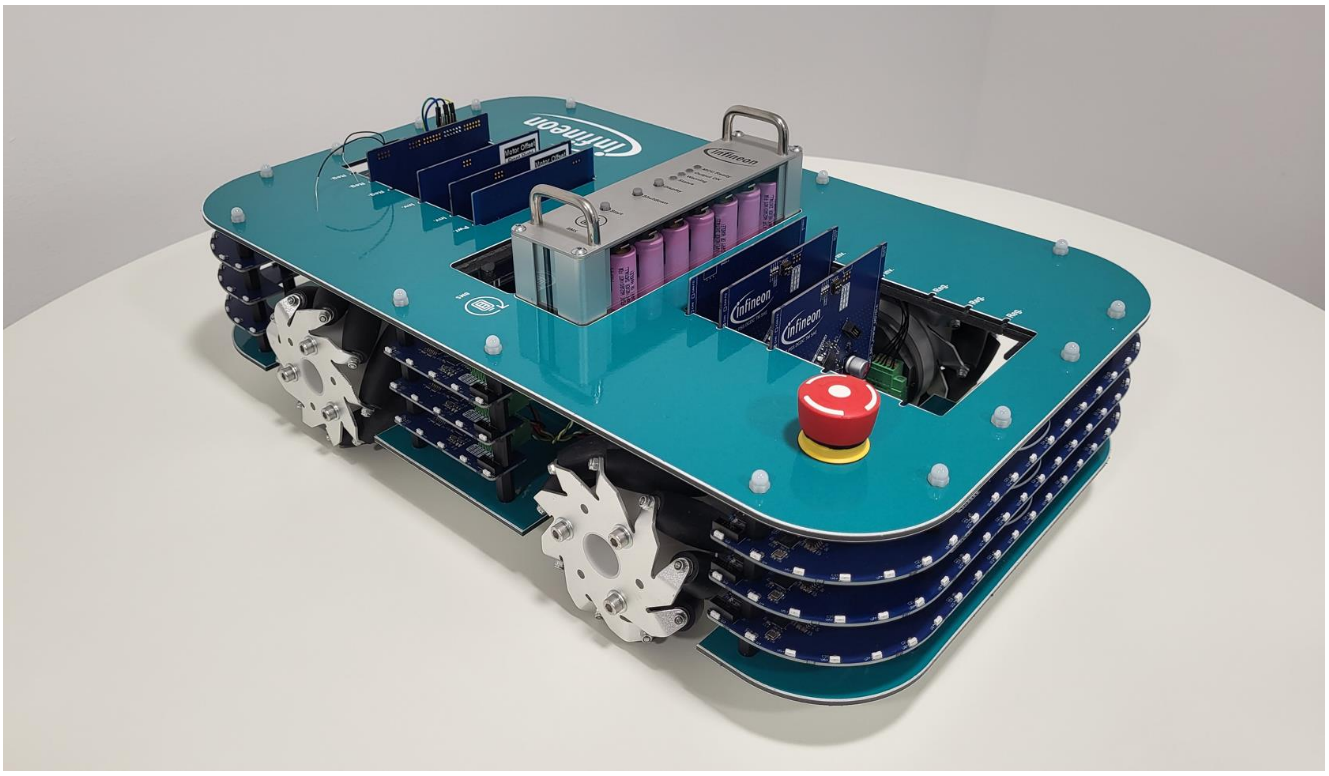


Figure 1 Isometric view of IMR

The IMR is intended to provide a demonstration platform for autonomous service robot functionalities by using Infineon components. This document describes how to use the DEMO_IMR_BMSCTRL_V1 board as the processing unit for all battery management system (BMS) tasks and interface the BMS power board including the following:

- Battery monitoring
- Battery balancing
- Estimation of state of charge (SoC)
- Estimation of state of health (SoH)

Figure 2 shows the system diagram of IMR.

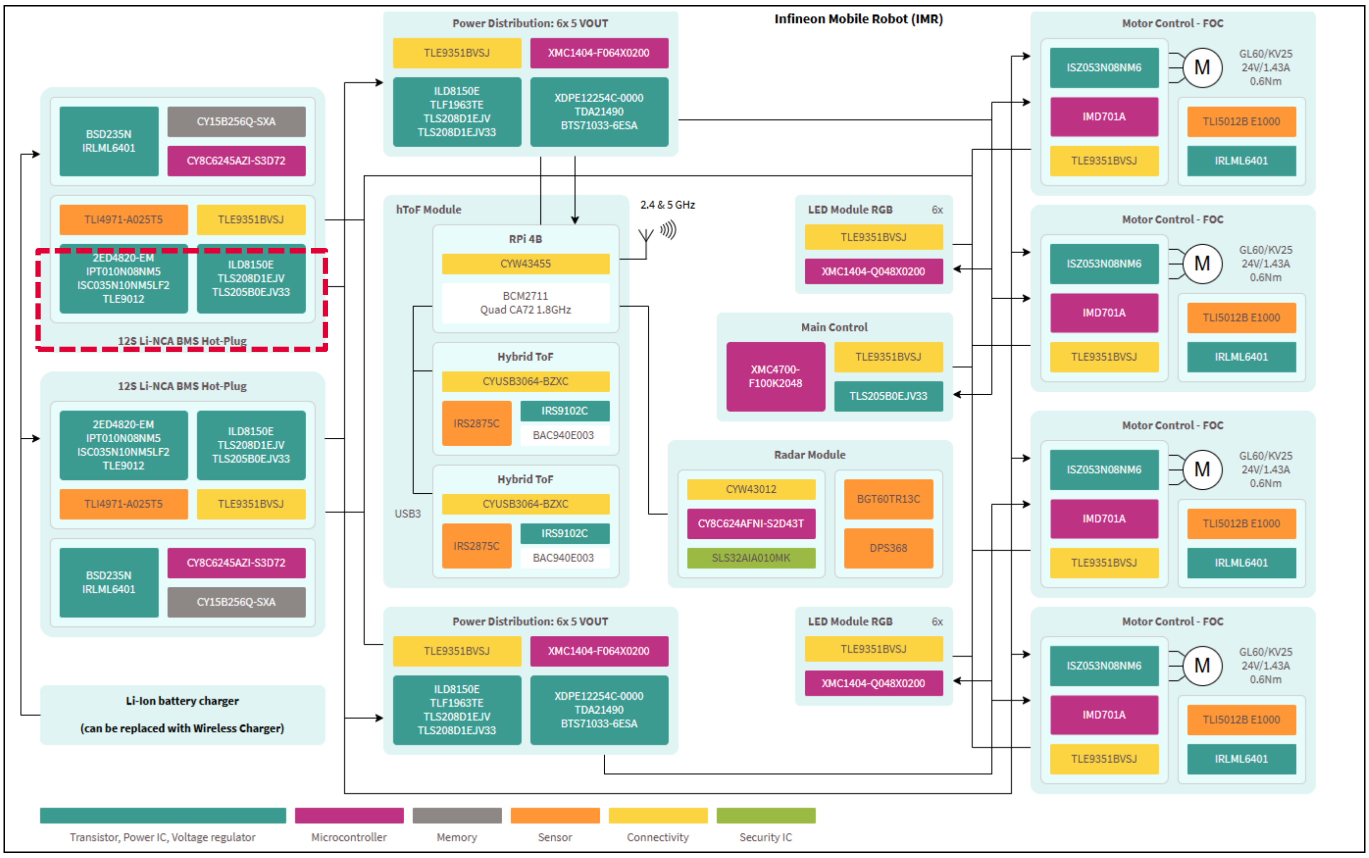


Figure 2 IMR overview

Note: DEMO_IMR_BMSCTRL_V1 board highlighted in red. Two BMS modules are installed in the IMR to facilitate hot swap.

Figure 3 comprises the Power section (DEMO_IMR_BMSPWR_V1) and the Control section (DEMO_IMR_BMSCTRL_V1) of the BMS.

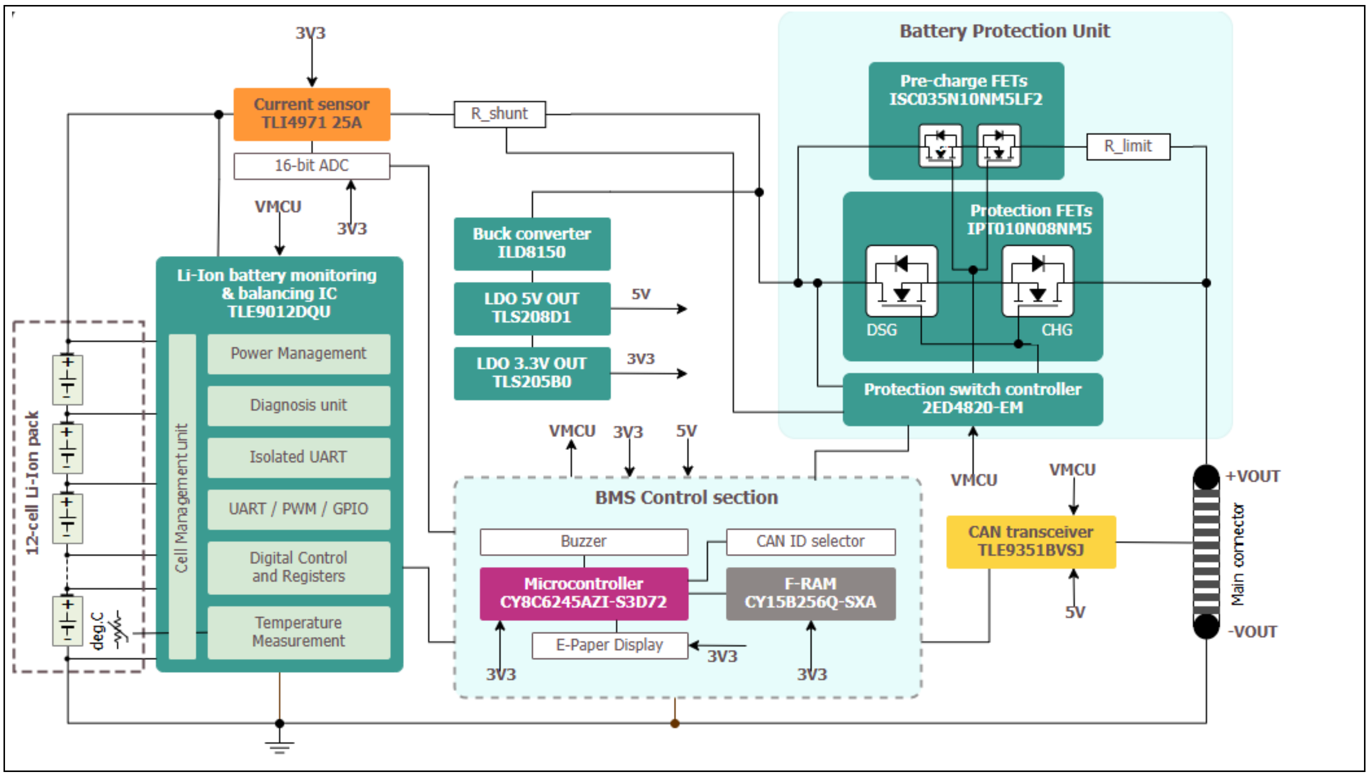


Figure 3 **BMS overview**

1.3 DEMO_IMR_BMSCTRL_V1 Demo Board

The DEMO_IMR_BMSCTRL_V1 Demo Board is a modular battery management solution, specifically designed to work with the battery-driven power board for the IMR. The board features a PSOC™ 6 series Arm® Cortex®-M4/M0+ MCU performing all necessary tasks including measurements and calculations. It estimates the battery's state of charge (SoC) and state of health (SoH) using measured voltages and currents and displays the calculated values on the provided ePaper display.

After combining the board with an appropriate IMR battery power board (that is, DEMO_IMR_BMSPWR_V1), the complete system is installed vertically into one of the IMR's battery management slots. After installation, all necessary communication is handled via the CAN/CAN FD interface.

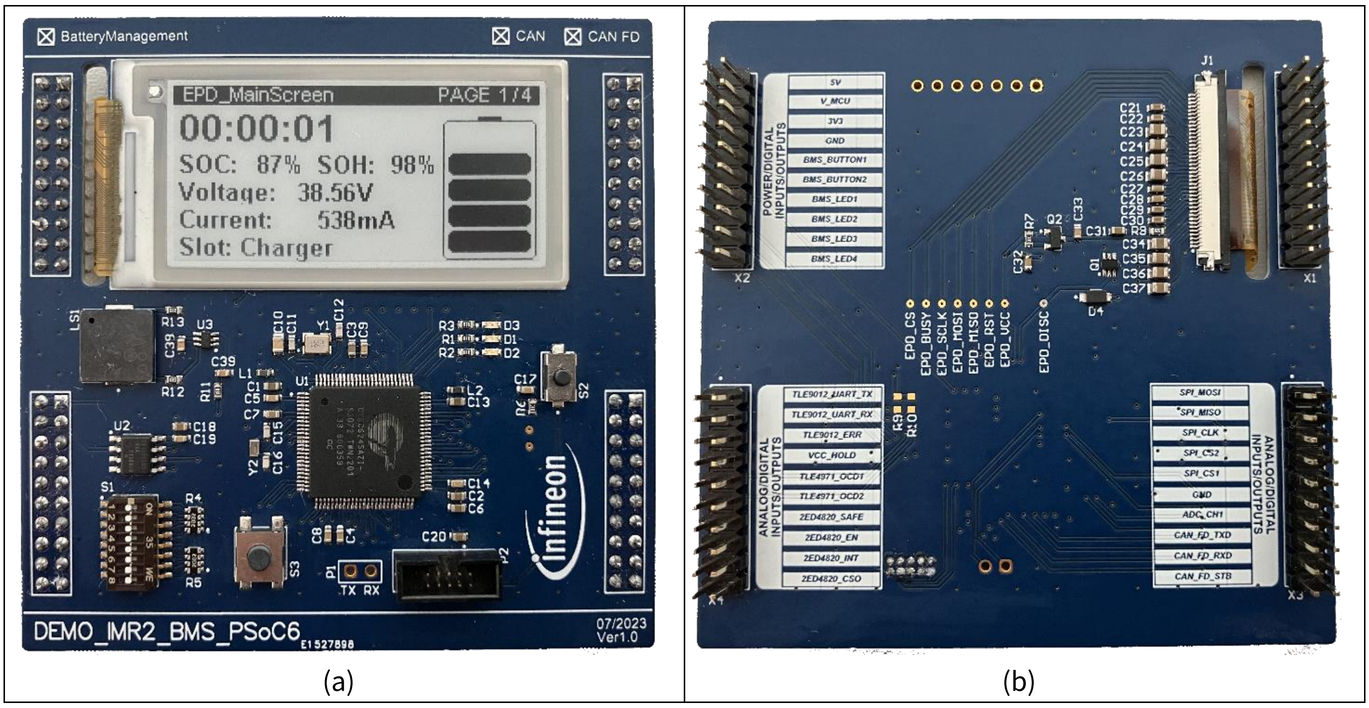


Figure 4 **(a) Top view; (b) Bottom view (80 mm x 80 mm)**

1.4 Connectors

The DEMO_IMR_BMSCTRL_V1 board provides connectors to support several peripherals, communication protocols, and connections to control the IMR battery power board (DEMO_IMR_BMSPWR_V1). This includes analog, digital, and communication lines for the following:

- Two external BMS buttons
- Four external status LEDs
- SPI serial interface and chip-select lines for two peripherals
- CAN communication lines for an external CAN transceiver
- UART debugging interface
- UART interface for the TLE9012 cell balancing IC
- Additional digital/analog lines for further functionality

In addition to the onboard ePaper display, an additional connector featuring both SPI and I2C interface is available for other displays (for example, SSD1306). Figure 5 shows the four main communication connectors and the debug communication header.

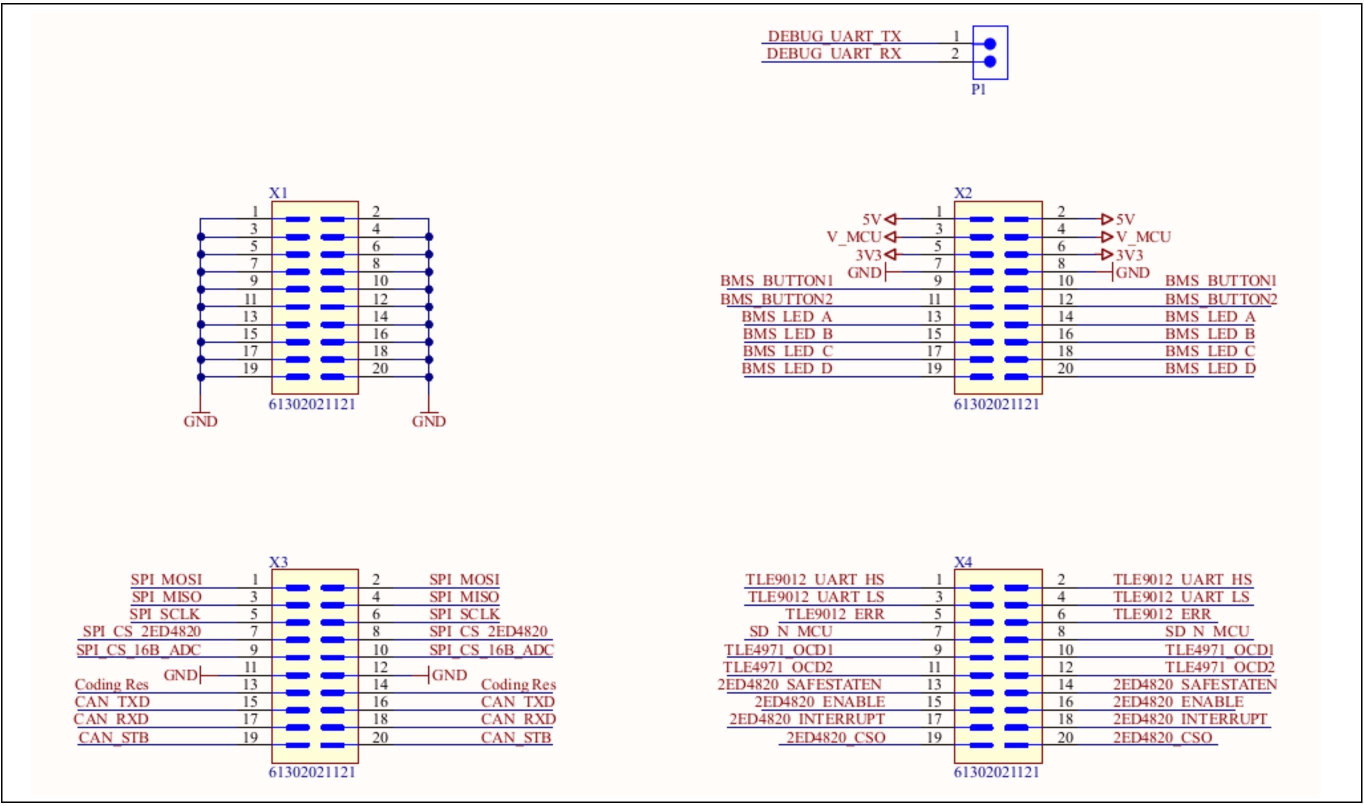


Figure 5 COM_CommunicationPort.SchDoc - Schematic overview of peripheral connectors

Table 2 shows the connection between connector P1 and the corresponding PSOC™ 6 MCU pins (UART transmit/receive lines) for the UART debugging interface. See the Quick start guide section for details on how to use this connector.

Table 2 Connection between connector P1 and PSOC™ 6 MCU for UART debugging interface

Name	PSOC™ 6 MCU pin	Function	Description
DEBUG_UART_TX	P0.3	Serial debug output	UART transmit line
DEBUG_UART_RX	P0.2	Serial debug input	UART receive line

Table 3 shows the common GND connector (X1) closest to the 40-pin FPC e-display connector; therefore, it is chosen without any additional communication lines.

Table 3 Connector X1 – Common GND connection

Name	PSOC™ 6 MCU pin	Function	Description
GND	–	Signal ground	–

Table 4 shows the connection between connector X2 and the corresponding PSOC™ 6 MCU pins for LEDs and button control. Additionally, both 5 V and 3.3 V are placed on the connector to supply any microcontrollers. The V_MCU line is used to set the digital supply voltage for peripherals on externally connected boards (the BMS power board in this case). The selected voltage for the PSOC™ 6 MCU is 3.3 V; V_MCU is also routed to that voltage level and the 5 V supply is unused.

Table 4 Connection between connector X2 and PSOC™ 6 MCU for BMS control/indicator and power connection

Name	PSOC™ 6 MCU pin	Function	Description
5 V	–	Signal power input	5 V DC power input
V_MCU	–	Signal power output	Peripheral power selector (selected: 3V3)
3V3	–	Signal power input	3.3 V DC power input
GND	–	Signal ground	–
BMS_BUTTON1	P7.1	Digital input/ERU	Digital ERU button input
BMS_BUTTON2	P7.0	Digital input/ERU	Digital ERU button input
BMS_LED_A	P10.5	Digital output	Digital status LED output
BMS_LED_B	P7.7	Digital output	Digital status LED output
BMS_LED_C	P7.6	Digital output	Digital status LED output
BMS_LED_D	P7.5	Digital output	Digital status LED output

Table 5 shows the connection between connector X3 and the corresponding PSOC™ 6 MCU pins for SPI and CAN serial interface lines. Additionally, this connector has an analog input that lets the system detect the hardware that the BMS is currently inserted in. The SPI interface has two Chip Select (CS) lines, which are currently used communication with the following:

- 2ED4820 high-side gate driver
- 16-bit ADC for current and load voltage measurements

Table 5 Connection between connector X3 and PSOC™ 6 MCU for the serial communication

Name	PSOC™ 6 MCU pin	Function	Description
SPI_ MOSI	P11.0	SPI MOSI	SPI master out; slave in
SPI_ MISO	P11.1	SPI MISO	SPI master in; slave out
SPI_ SCLK	P11.2	SPI SCLK	SPI clock
SPI_ CS_2ED4820	P11.4	SPI Chip Select	SPI 2ED4820 Chip Select
SPI_ CS_16B_ADC	P11.3	SPI Chip Select	SPI MCP3465 Chip Select
GND	–	Signal ground	–
Coding_Res	P10.7	Analog input	Analog slot detection
CAN_TXD	P5.1	CAN output	CAN transmit line
CAN_RXD	P5.0	CAN input	CAN receive line
CAN_STB	P5.6	Digital output	Optional CAN standby

Table 6 shows the connection between connector X4 and the corresponding PSOC™ 6 MCU pins for serial communication lines to the following external components on the IMR battery power board:

- Main UART communication interface for the TLE9012 battery balancing IC
- Auxiliary digital and analog inputs/outputs for the TLE4971 current sensor and 2ED4820 high-side gate driver

Table 6 Connection between connector X4 and PSOC™ 6 MCU for the serial communication and auxiliary inputs/outputs

Name	PSOC™ 6 MCU pin	Function	Description
TLE9012_UART_HS	P6.0	UART HS	UART TLE9012 HS line
TLE9012_UART_LS	P6.1	UART LS	UART TLE9012 LS line
TLE9012_ERR	P6.5	Digital input	TLE9012 error output
SD_N_MCU	P8.3	Digital output	Power Stay-on output
TLE4971_OCD1	P8.0	Digital input	TLE4971 overcurrent detection output 1
TLE4971_OCD2	P8.1	Digital input	TLE4971 overcurrent detection output 2
2ED4820_SAFESTATEN	P7.2	Digital output	2ED4820 safe state enable output
2ED4820_ENABLE	P7.4	Digital output	2ED4820 enable output
2ED4820_INTERRUPT	P7.3	Digital input	2ED4820 error flag output
2ED4820_CSO	P10.6	Analog input	2ED4820 current analog output

In addition to the ePaper display on the board, an additional optional connector (J2) supports an external I2C or SPI TFT display (for example, SSD1306). The connector is supplied via the 3V3 voltage supplied input as shown in Figure 6 and Table 7.

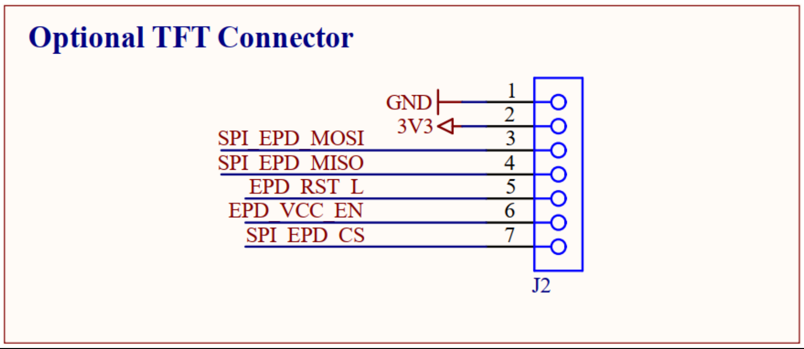


Figure 6 COM_CommunicationPort.SchDoc – Optional TFT connector

Table 7 Connection between connector J2 and PSOC™ 6 MCU – Optional TFT connector for I2C display (for example, SSD1306)

Function	PSOC™ 6 MCU pin	Function	Description
GND	–	Signal ground	–
3V3	–	Signal power	Regulated 3.3 V DC supply
SPI_EPD_MOSI	P9.0	SPI MOSI	SPI MOSI/I2C SCL
SPI_EPD_MISO	P9.1	SPI MISO	SPI MISO/I2C SDA
EPD_RST_L	P10.4	Digital output	OLED Reset (RST)
EPD_VCC_EN	P12.0	Digital output	OLED Data Comm. (DC)
SPI_EPD_CS	P9.3	SPI CS	OLED Chip Select (CS)

2 Hardware overview

This section describes the functional parts of the board hardware. An overview of the hardware components and their connection is shown in Figure 7 and Figure 8. See the Appendix for the schematics.

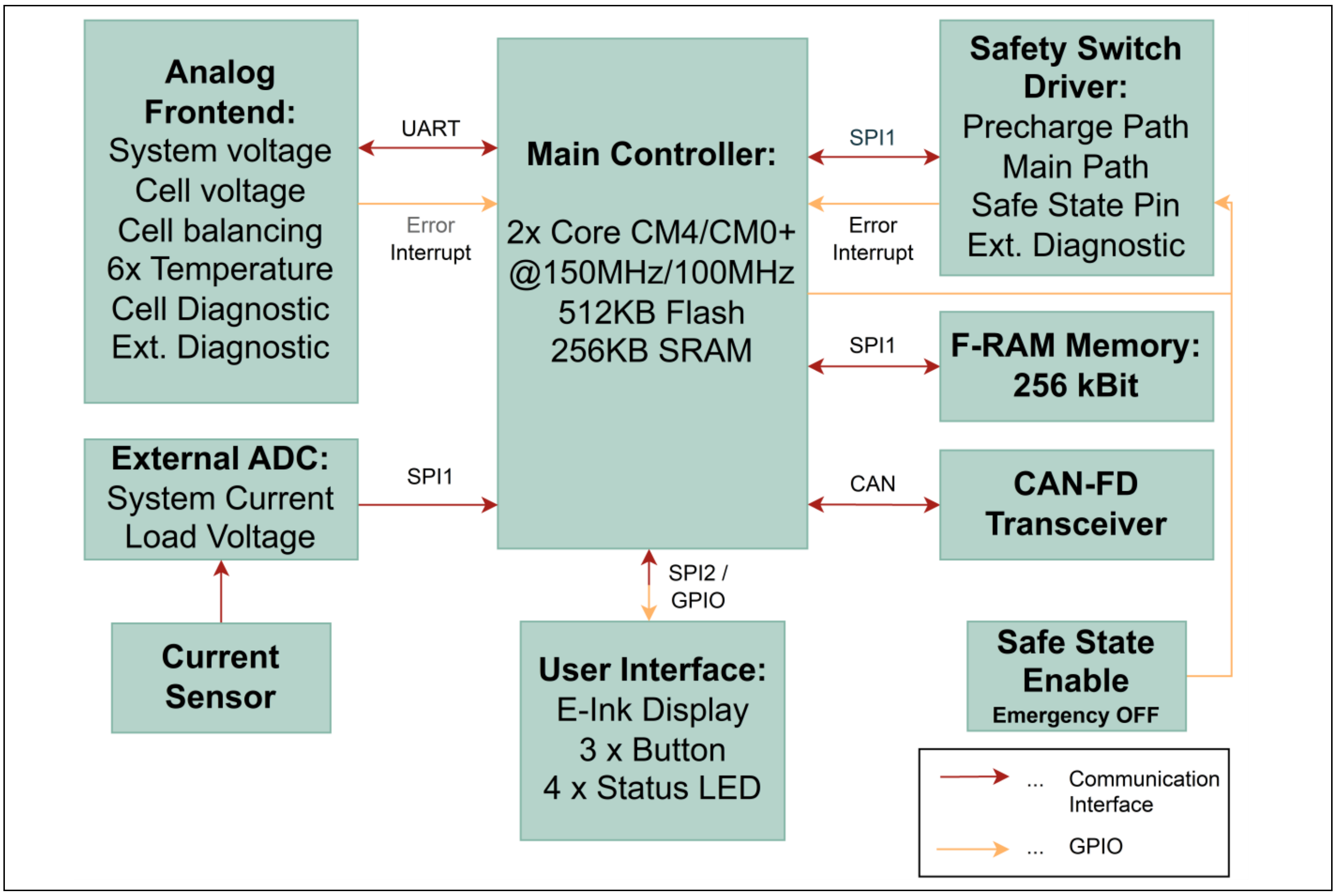


Figure 7 **Block diagram of the system and connections**

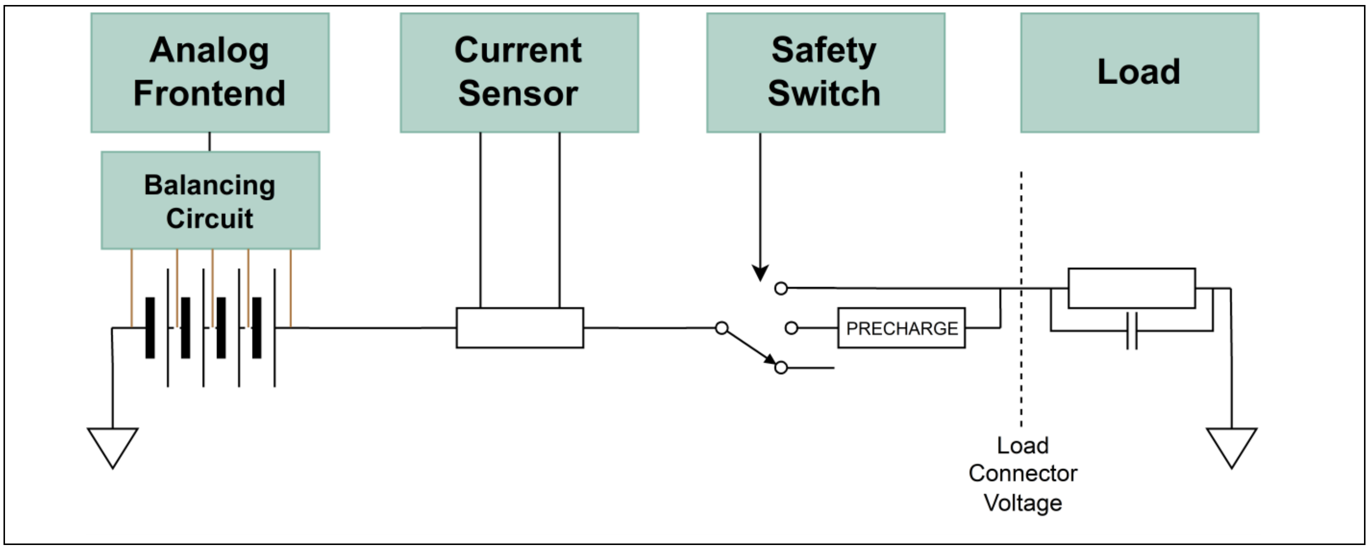


Figure 8 **Overview of the connections between the components**

The battery, composed of multiple battery cells connected in series, is connected to the analog frontend, current sensor, safety switch and to the load. The block diagram illustrates the devices used, the key specifications and the logical connections between them.

The main processor (PSOC™ 6 CY8C6245AZI) is a 32-bit dual-core system housed in a 100-pin package. For this application, two SPI interfaces, a UART, a CAN interface as well as multiple GPIOs for interrupt signals, LEDs and buttons are utilized.

Note that the external ADC, safety switch driver and memory share the same SPI bus, which is managed in the software. Additionally, for the UART of the analog frontend (TLE9012) the TX and RX lines are connected to each other to enable half-duplex communication.

2.1 PSOC™ 6 MCU

The PSOC™ 6 family is built on an ultra-low-power architecture and the MCUs feature low-power design techniques that are ideal for battery-powered applications. The dual-core Arm® Cortex®-M4 and Cortex®-M0+ architecture lets designers optimize power and performance simultaneously. Using its dual cores combined with configurable memory and peripheral protection units, the PSOC™ 6 MCU delivers the highest level of protection defined by the Platform Security Architecture (PSA) from Arm®.

The CY8C6245AZI-S3D72 device in a 100-pin TQFP100 package supports both a 150 MHz Cortex®-M4F and a 100 MHz Cortex®-M0+ and has 512 KB application flash and 256 KB of SRAM. This device provides programmable power control inside the largest available package with the most I/O pins available for this device family.

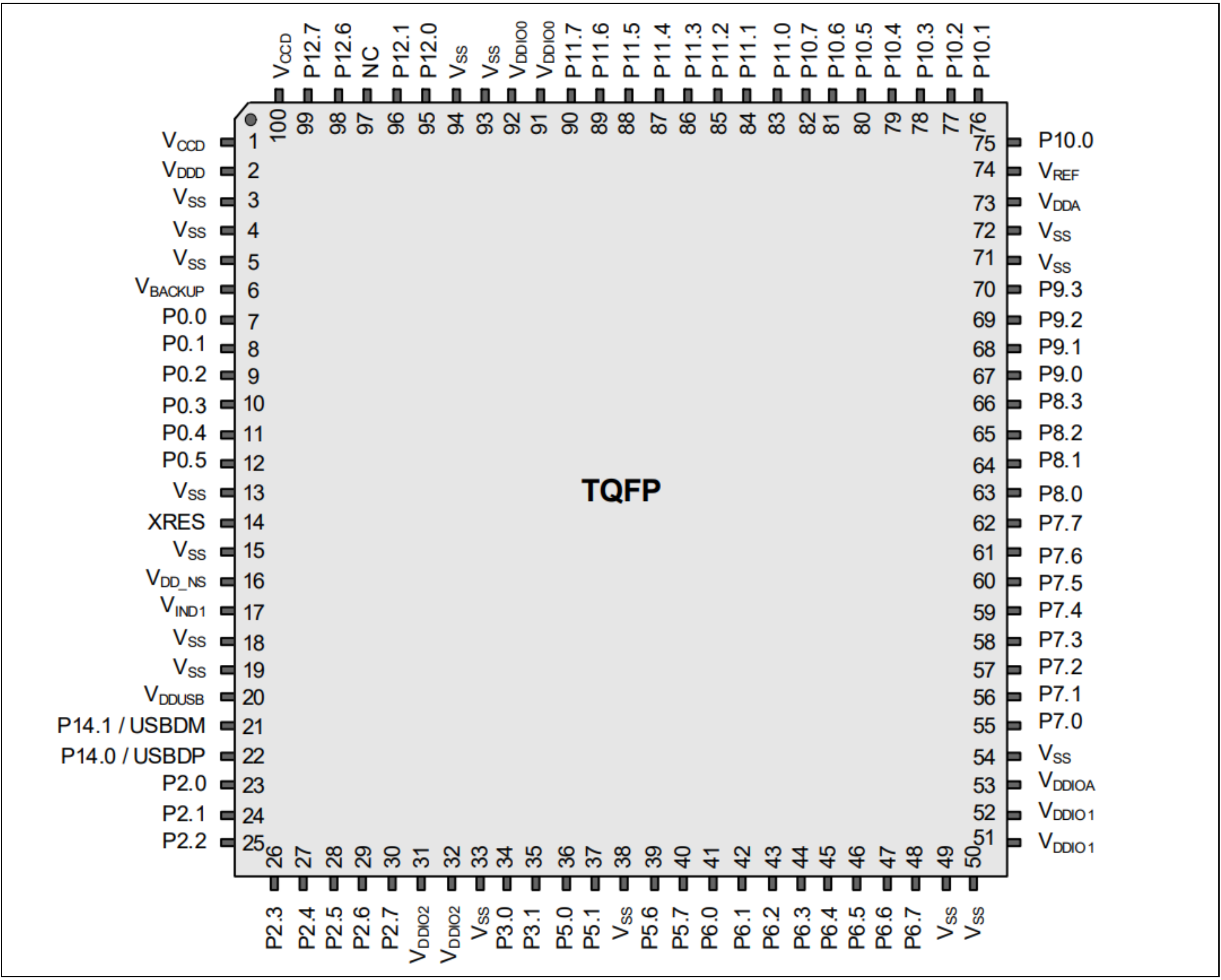


Figure 9 CY8C6245AZI-S3D72 PSOC™ 6 TQFP 100-pin configuration (top view) [2]

2.2 Piezoelectric buzzer circuit

To provide audio feedback, a piezoelectric sounder is provided, along with the necessary auxiliary components. Audio feedback is provided for the following events:

- End of charging
- Low-voltage warning
- Overcurrent warning
- Overtemperature warning

The speaker can be controlled using one of the PSOC™ 6 MCU's digital outputs.

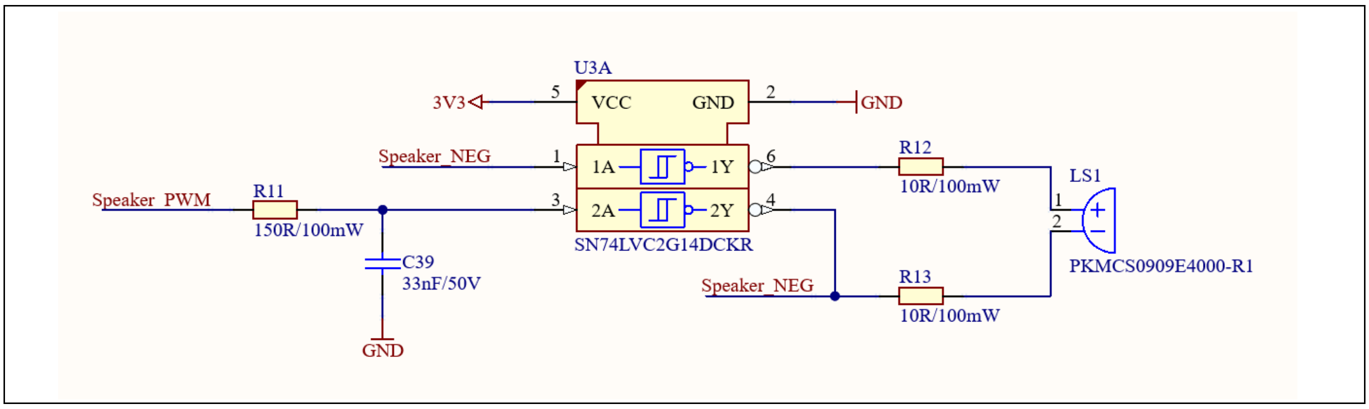


Figure 10 Piezoelectric buzzer circuit including auxiliary components

2.3 Serial FRAM memory circuit

The CY15B256Q 256 Kb nonvolatile memory is based on an advanced ferroelectric process and performs reads and writes similar to a RAM. It provides reliable data retention for 121 years while eliminating the complexities, overhead, and system-level reliability problems of serial flash, EEPROM, and other nonvolatile memories.

Unlike serial flash and EEPROM, CY15B256Q performs write operations at bus speed without write delays. Data is written to the memory array immediately after each byte is successfully transferred to the device [1].

In this application, the CY15B256Q device is used to store the measured individual cell voltages and overall current values for further calculation of SoC and estimation of SoH. The chip provides an SPI interface, which is directly connected to one of the PSOC™ 6 MCU's peripheral SPI blocks. In addition to the common lines for SPI communication, additional pins are provided:

- **Write Protect (WP) pin:** An input that prevents write operations; can be used to prevent unwanted changes to the Status Register
- **HOLD pin:** An Input that allows the host CPU temporarily to determine if it must interrupt a memory operation to perform another task

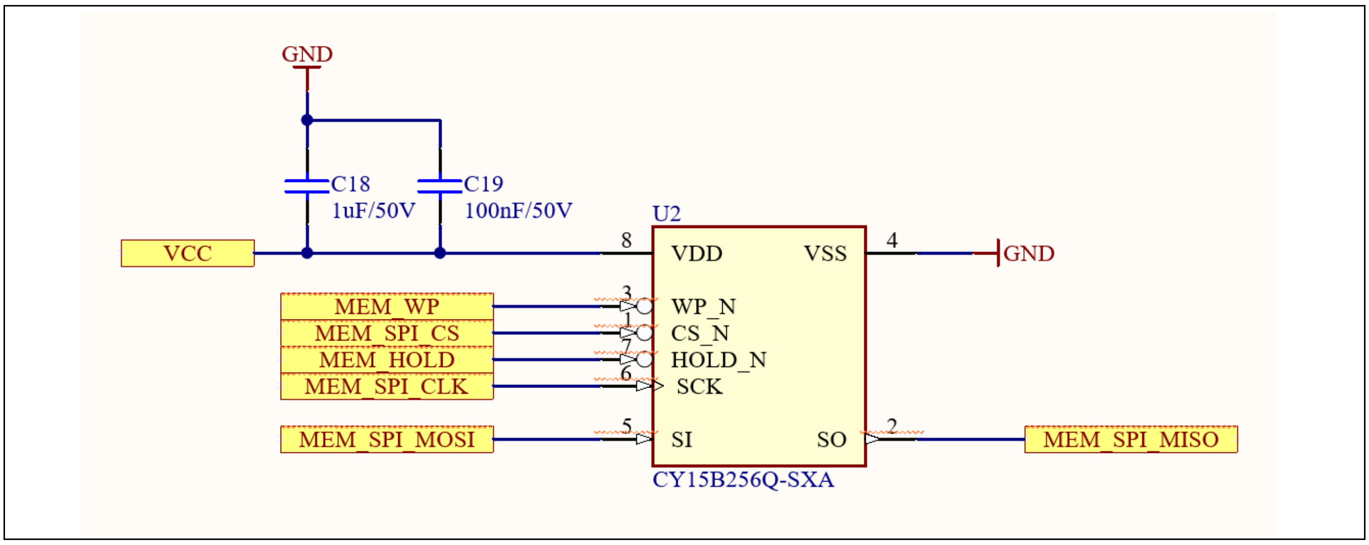


Figure 11 CY15B256Q automotive serial F-RAM circuit (256 Kb) interfaced via SPI

2.4 ePaper display circuitry

The board has an ePaper display to display all necessary information and BMS status, including the last BMS state even when the system is powered off. The display is powered up directly from the microcontroller via IRLML6401 P-channel MOSFET. Necessary discharge lines are driven via the BSD235 dual-package switch. The connector is a 40-pin FPC connected to various auxiliary components. As a reference, the E-ink Display Shield Board (CY8CKIT-028-EPD) is used for auxiliary components and communication lines.

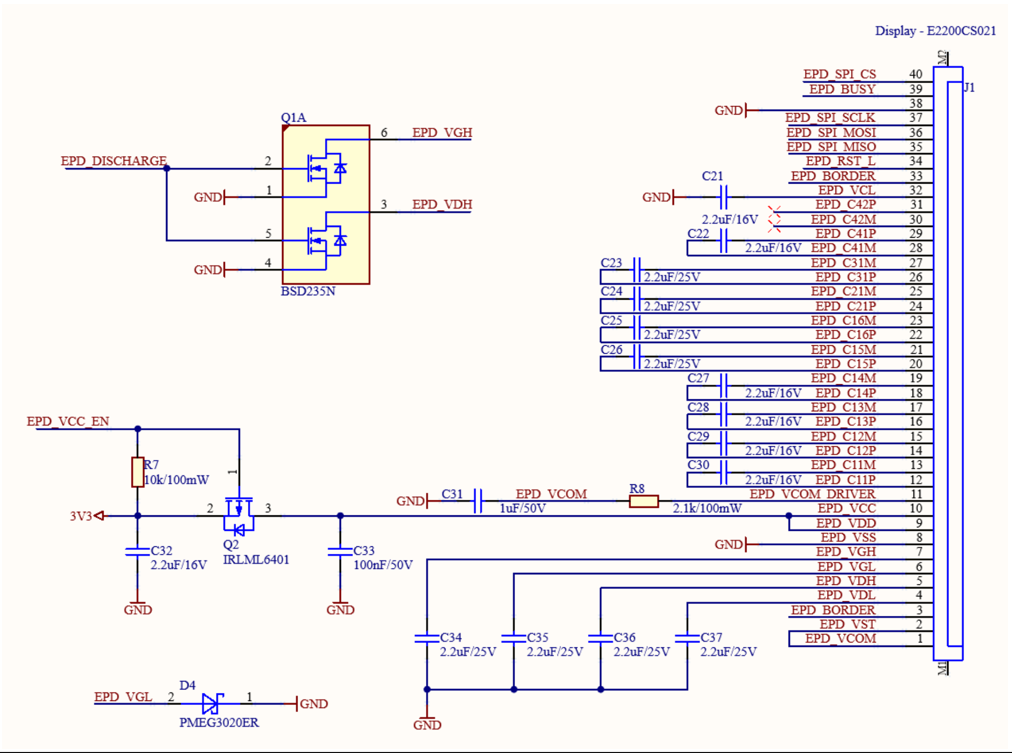


Figure 12 **ePaper display connector circuit with auxiliary components (for Pervasive Displays E2200CS021)**

2.5 Unique identification circuit

To identify each board interfaced uniquely, a DIP switch is implemented. Each pin is pulled up via a 20 kΩ resistor. This provides an inverted logic that must be considered when reading the pin via the PSOC™ 6 MCU. This can be used to identify multiple boards. For the CAN bus, the identification is achieved by a coding resistor on the BMS motherboard (R1), see Section 3.1.10 Slot Detection.

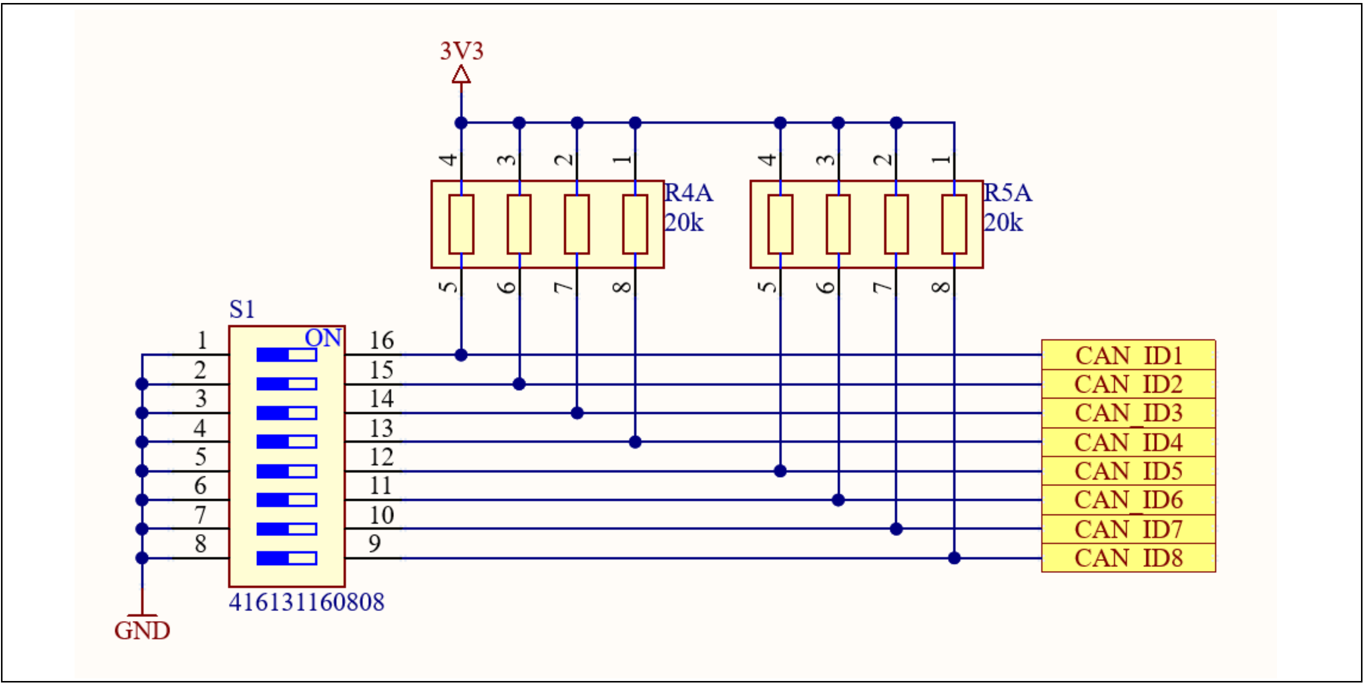


Figure 13 CAN node identification circuit including 8-pin DIP switch

Table 8 shows each DIP switch pin and the corresponding PSOC™ 6 MCU pin for reference, as well as the decimal value assigned to the pin. When pulled up to 3.3 V, the pins follow an inverted logic: the OFF state on the DIP switch represents a logic HIGH on the PSOC™ 6 MCU pin and conversely.

Table 8 DIP switch S1 – CAN node identifier logic and pin assignment

ID bit	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
DIP switch	Pin 8 (MSB)	7	6	5	4	3	2	Pin 1 (LSB)
PSOC™ 6 MCU pin	P3.1	P3.0	P2.7	P2.6	P2.1	P2.0	P0.5	P0.4

In the IMR implementation, which allows up to two BMS modules, the 8-pin DIP switch is currently not utilized, and the slot-specific CAN ID ranges of the BMS control in IMR have been assigned for each slot.

3 Software

This section covers an introduction to the environment of PSOC™ microcontrollers, and the software interfaces used for communicating with the BMS system. For more information on software source code, see the Infineon IMR_PSoC6_BMS GitHub repository.

3.1 Software architecture

3.1.1 Software structure and settings

The project is divided in two sub-projects for the main (proj_cm4) and secondary (proj_cm0) cores.

In the main project (CM4), the files named *global_management.c* and *global_management.h* incorporate all central objects and functions that may be used by all modules. Here, the structure object 'statusBMS' is used to access all status-related values such as voltage, current, bitmaps, and states. The structure object 'setupBMS' is used for initial settings and comparison values, which could be changed during runtime. The structure object 'batteryState' is used to store all intermediate vales for the battery state algorithm, as well as the State of Charge (SoC), (SoH), and the remaining runtime value.

In *global_management.h*, all settings and hard-coded values used anywhere in the main project are condensed (around 100 values). For every value, its unit and an extensive explanation, including possible limitations and implications, are given inside the software code.

The actual logic of the BMS is implemented in the main file. The main loop itself is abstracted into many modules to keep it short and clear (see flowchart in Figure 15).

Alongside these files, software modules such as 'balancing', 'battery_state', 'diagnostic', 'memory' and 'safety_switch' are implemented to be exchanged by advancements of the project if needed. Usually there are only two functions, one for initialization and one for a repeated call during the main loop. These functions are allowed to access the earlier-mentioned objects and structures directly. Details are described in the files.

In the secondary project (CM0P), most of the display logic and data synchronization are implemented in the main file (see flowchart in Figure 29). The display menu design itself is defined in the *PSoC6_BMS_EPD_ScreenConfig.c and PSoC6_BMS_EPD_ScreenConfig.h* files. This core is not used for any higher-priority functions of the BMS.

3.1.2 Main design

The core functions and main loop implementation of this project (on CM4) adheres to three design principles:

- **Non-blocking software design**: The main loop of the battery management system (BMS) is designed to operate without blocking or waiting for specific actions to be completed. All actions utilize state machines that run code at full speed until they reach a point where they need to wait. At this juncture the state machine switches states and ends the action. In subsequent loops the new state continuously checks if the waiting condition is met (for example, peripherals are ready). After the condition is satisfied, the code executes and transitions to the next state. This design ensures that critical actions run with high cycle times
- **Event-driven execution**: Non-critical actions are triggered by specific intervals or other actions, allowing processes that do not need constant execution (such as user interface updates, communication, and balancing algorithms) to run only when necessary. Many actions depend on data from other processes (for example, measurement data) and it is inefficient to continue calculations with outdated data. In these scenarios, a special flag is set by the triggering function and is cleared by the receiving function. This method enables rapid system responses to changing conditions and minimizes unnecessary code execution

- **Timeout protection**: To guarantee the reliability and safety of the BMS, critical steps in the main loop are protected by timeouts. If a critical task exceeds the expected completion time the system will time out and take corrective measures. For instance, if a read command to the analog frontend takes longer than anticipated the request will time out, revert to the previous state and reattempt the data request

Figure 14 illustrates these principles through state diagrams of two actions executed sequentially in the main loop. In 'Action 1' the system remains in the 'Wait Finish' state until the measurement is complete. After completion, the data read over the peripheral is triggered, transitioning from the state of 'Action 1' to 'Wait Read.' The system then waits until the data is read from the registers of the analog front-end IC. After reading, the raw data is converted into readable decimal values and the state changes to "Wait Trigger." The action remains in this state until a predefined interval elapses allowing for consistent sampling time. At this point a flag is set to trigger 'Action 2'. Unlike 'Action 1', 'Action 2' is not interval-based but depends on the interval of 'Action 1'. This approach ensures a high cycle-time while allowing individual actions to operate at their own speed. The interval execution is structured so that the actual execution time does not extend the interval, with the only inaccuracy being the cycle time itself. Figure 14 illustrates the design principles through the depiction of three states for 'Action 1' and 'Action 2'.

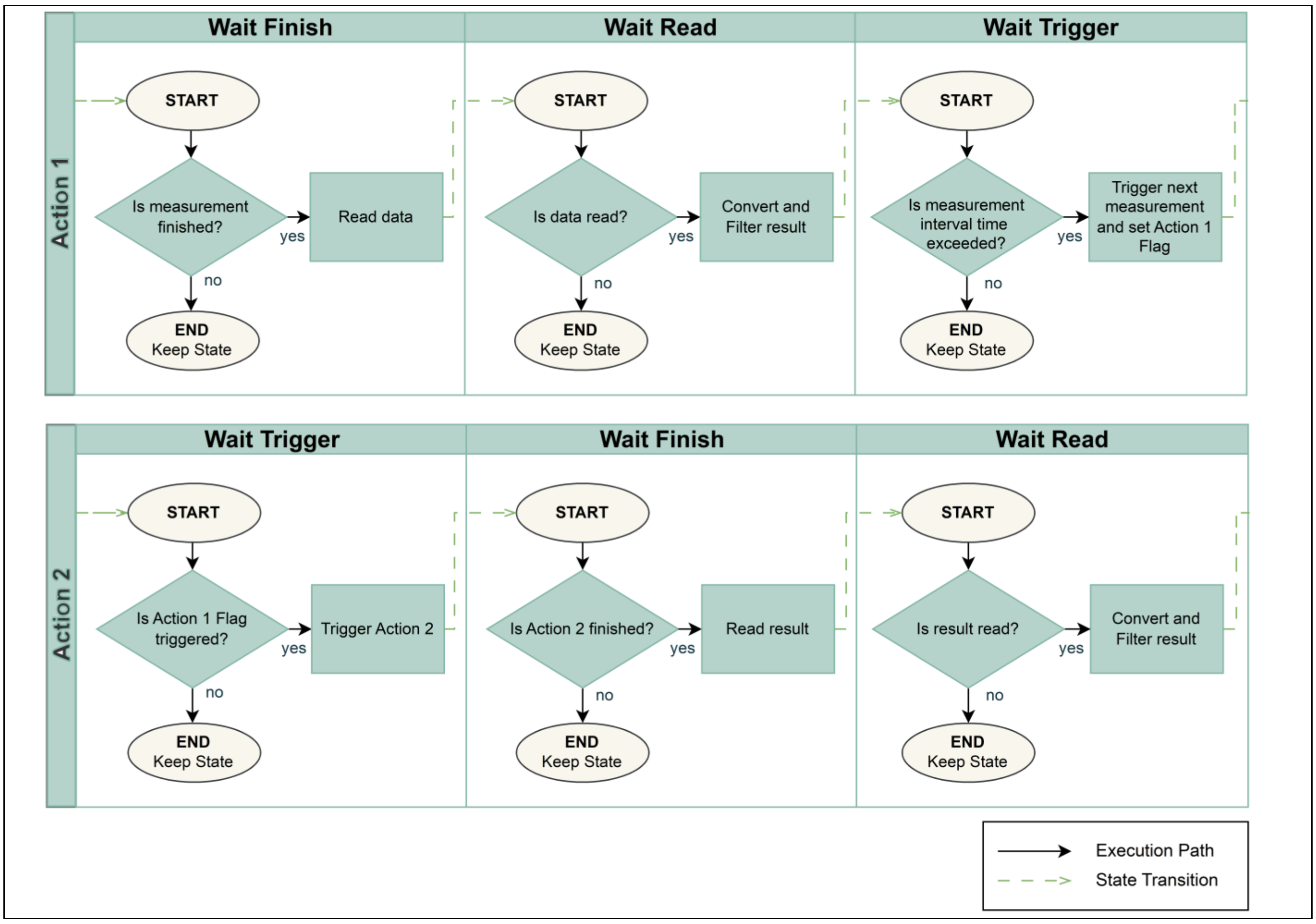


Figure 14 **Depiction of three states for 'Action 1' and 'Action 2'.**

Note: *The green dashed arrows indicate state transitions, while the black solid arrows represent the execution flow within a state.*

The actual main loop sequence is depicted in Figure 15. In essence, the system performs measurements, conducts diagnostics based on these measurements, determines the system state, controls the switch, and executes multiple housekeeping tasks. Detailed descriptions of the most critical components are provided on the following pages.

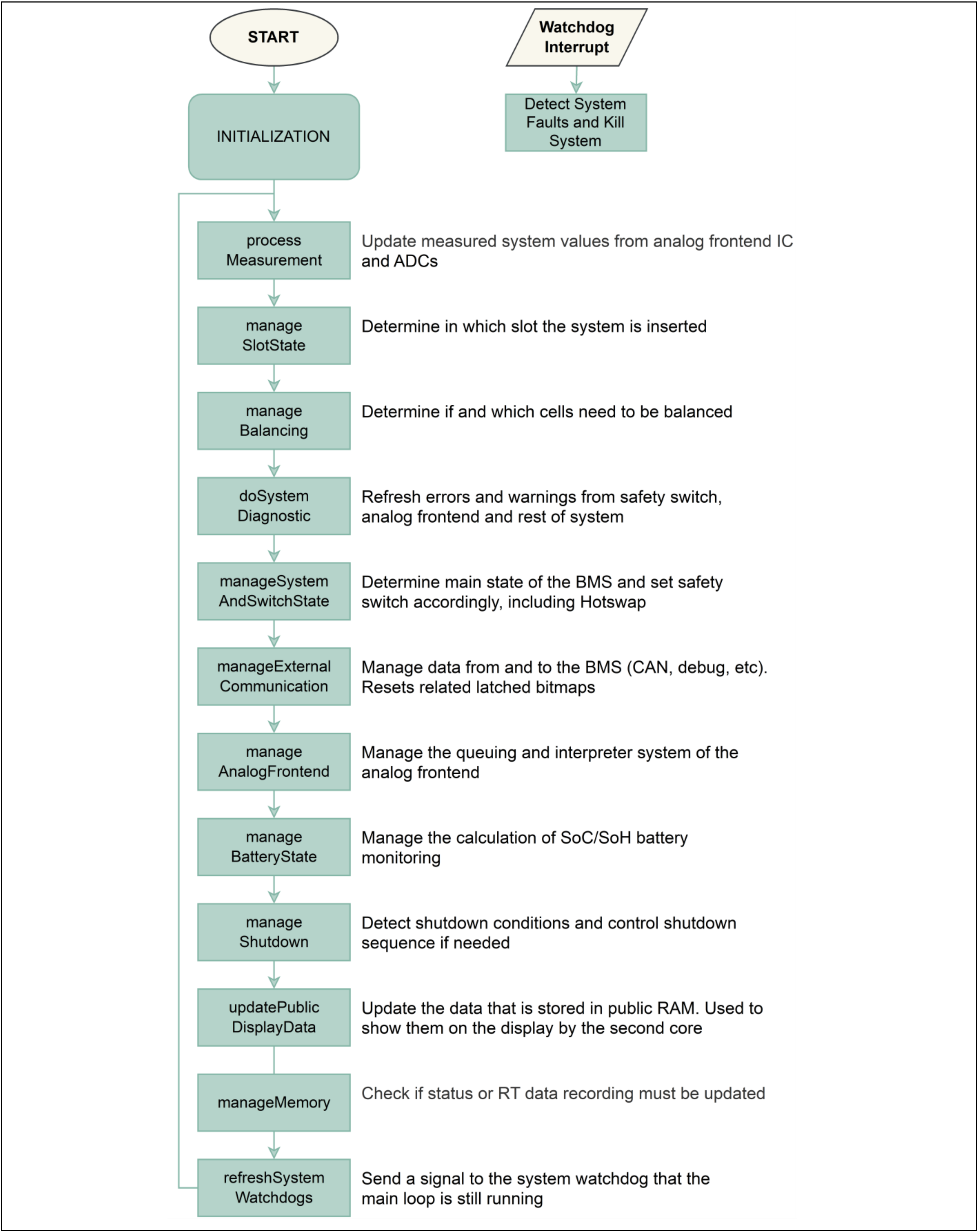

START
Watchdog Interrupt
INITIALIZATION
Detect System Faults and Kill System
process Measurement
Update measured system values from analog frontend IC and ADCs
manage SlotState
Determine in which slot the system is inserted
manage Balancing
Determine if and which cells need to be balanced
doSystem Diagnostic
Refresh errors and warnings from safety switch, analog frontend and rest of system
manageSystem AndSwitchState
Determine main state of the BMS and set safety switch accordingly, including Hotswap
manageExternal Communication
Manage data from and to the BMS (CAN, debug, etc). Resets related latched bitmaps
manage AnalogFrontend
Manage the queuing and interpreter system of the analog frontend
manage BatteryState
Manage the calculation of SoC/SoH battery monitoring
manage Shutdown
Detect shutdown conditions and control shutdown sequence if needed
updatePublic DisplayData
Update the data that is stored in public RAM. Used to show them on the display by the second core
manageMemory
Check if status or RT data recording must be updated
refreshSystem Watchdogs
Send a signal to the system watchdog that the main loop is still running

Figure 15 **The main loop sequence in linear and abstracted design**

Note: The non-blocking architecture ensures that all safety-critical measurements and their associated actions are processed periodically and in rapid succession.

3.1.1 Initialization

The goal of the initialization phase is to start up all peripherals and subsystems. To ensure that the main loop runs correctly, all system variables must be set during this phase; otherwise, the main loop may use NULL data for initial checks. The following are the actions taken during this phase, with details explained in the dedicated following sections:

1. The dual-core PSOC™ 6 processor starts with the secondary (CM0+) core. This core enables the user interface peripherals (for example, Display SPI, Button), initializes the public RAM used for communication with the main core (CM4), and subsequently enables the main core. The processor then waits for CM4 to start before entering the main loop (see Figure 29)
2. The main core begins by initializing the software and hardware watchdogs. This step is crucial to ensure that the system does not lock in to an on-state and remains capable of being shut down. As an additional precaution, the secondary CM0+ core monitors the shutdown button while the main core is initializing. Subsequently, the remaining peripherals, including CAN, SPI2, internal ADC, and interrupts are initialized
3. With the internal ADC running, the system performs the first detection of the slot in which it is inserted. Four states can be differentiated: “None”, “System 1”, “System 2” and “Charger”. The internal ADC measures the output of a voltage divider consisting of a fixed resistor on the BMS hardware and a variable resistor based on the slot that the BMS is inserted in. The measured voltage determines the slot. This slot detection is repeatedly run during the main loop
4. It can be ensured that the system can keep running even after the start button is released. Therefore, the green LED is turned on and the VCC_HOLD pin is enabled, keeping the logic power supply and system active until shuts down
5. The safety switch driver is initialized. The hardware starts up with the enable pin, that all registers are read and initial errors are reset. Default settings, such as internal over-current thresholds and configurations, are written to the registers
6. To determine the last state of the system and facilitate later real-time data recording, the memory is initialized. The last SoC and SoH are read from memory and will be used during battery state initialization. If the system starts in one of the system slots, the real-time data recording is reset. Otherwise, it is not reset, as data reading is expected
7. For accurate current and load voltage measurement, the external ADC is configured by setting the appropriate configuration registers. The current is then read and averaged multiple times to determine the system's idle current, which helps establish an accurate current offset
8. The analog frontend (TLE9012) is initialized by setting all configuration registers to the values defined in the settings. This includes configurations for cell setup, system/error/balancing behavior, and thresholds for temperature, balancing current, open-loop voltage, and cell voltage
9. The SoC and SoH are retrieved from memory or determined to provide a starting point for battery state estimation. SoH is always read from memory as it cannot be directly estimated at startup using the Coulomb Counting method. The SoC is determined from the voltage in the linear regions of the stored open circuit voltage (OCV) curve, which describes the SoC of the battery at a given voltage if no load is applied. In non-linear regions, the last stored SoC is used
10. At this point, the role of the BMS for hot-swap operations is determined (for details, see 3.1.4.2 Hot-swap). The system sends out CAN messages on the bus to identify another potential BMS. If none is found, the system starts as the first BMS (“MAIN” Role) and attempts a soft switch-on with pre-charge. If another BMS is found, the one inserted in system slot 1 becomes “MAIN”, and the other system starts as “TAKER”, waiting for the main BMS to initiate a handover

11. All counters and data for CAN messaging are reset to achieve a defined starting state

3.1.2 Measurement

A main loop cycle always begins with the measurement phase (Figure 15). During this phase, the system reads voltages using the analog frontend (TLE9012), current/load voltage through the external ADC, and the coding resistor for slot detection via the internal ADC. This process is depicted in the "Trigger Next" state in Figure 16. The analog frontend measurement occurs at a set interval, and the current measurement trigger is synchronized with the analog frontend trigger.

3.1.2.1 Analog frontend measurement

After the analog frontend measurement is triggered, the state changes to "Wait Result" (see Figure 16). In this state, the system repeatedly reads the measurement control registers [MEAS_CTRL] until a marker, indicating the measurement is ongoing, disappears. Once it disappears, two actions are initiated for the analog frontend.

1. The balancing bitmap, determined by the 'manageBalancing' function, is written to the frontend to enforce balancing. This is done immediately because any voltage measurement halts the balancing for accurate reasons. By performing this action promptly, balancing can resume as quickly as possible after it has been stopped
2. The multi-read process is triggered, causing the analog frontend to send the result registers of all set values. Note that the field addressed by the multi-read feature is configured during initialization of the analog frontend

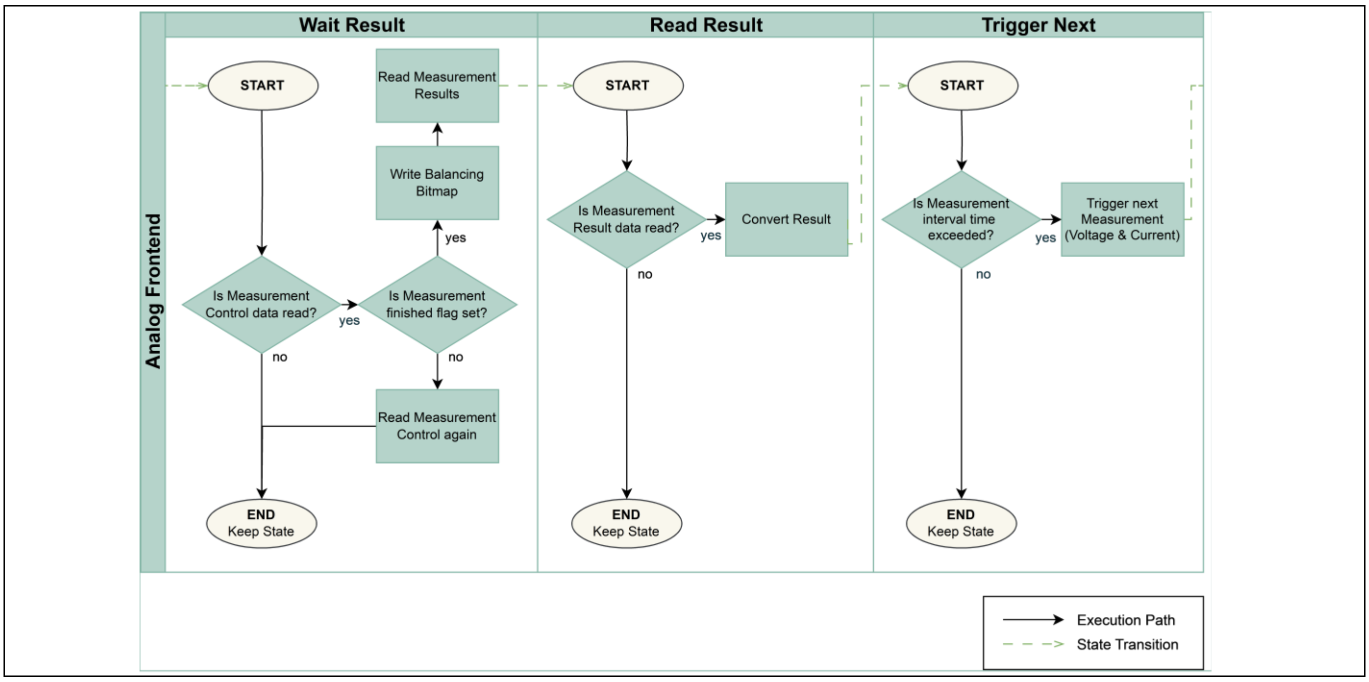


Figure 16 **Analog frontend measurement flow chart**

3. The state changes to "Read Result". Here, the system waits until the multi-read is complete
4. The results are converted to usable values and stored in the system objects
5. The state changes back to "Trigger Next," where the system waits for the remaining interval time before starting the cycle anew. If the external ADC measurement is still ongoing when the next analog frontend measurement is triggered, a debug message is written to UART

3.1.2.2 ADC measurement

The external ADC measurement state machine, shown in Figure 17, that is always executed after the analog frontend. In the "Wait Trigger" state, the system waits for the analog frontend measurement to be activated to achieve synchronization. After this synchronization, the current measurement is triggered, changing the state to "Wait Current Ready". In this state, the system continuously checks the data-ready register of the ADC until the data is ready for use.

The measured value is then converted and filtered before the state transitions to "Wait Load Voltage Ready". The same procedure is used to read the load voltage. During system initialization, the current offset is calculated (see Section 3.1.1 Initialization).

The processor's internal ADC is used to detect the system slot (see Section 3.1.1 Initialization). The detection measurement is performed every cycle.

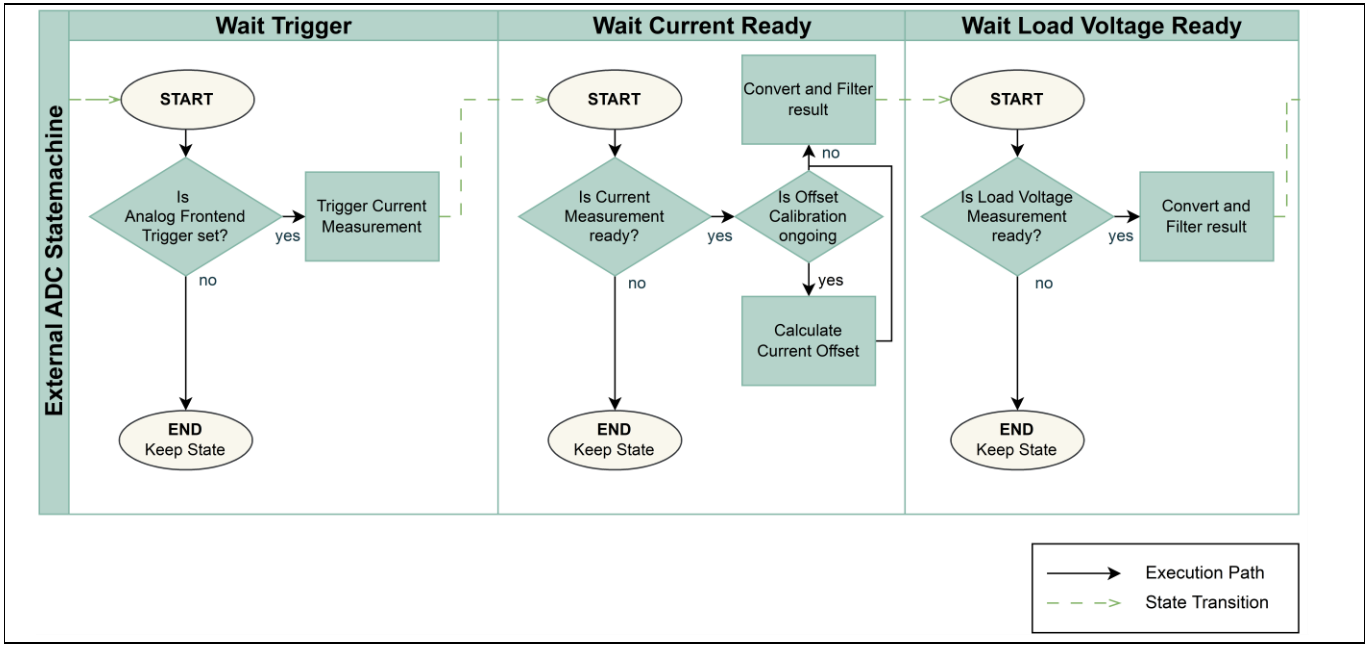

External ADC Statemachine
Wait Trigger
Wait Current Ready
Wait Load Voltage Ready
START
Is Analog Frontend Trigger set?
yes
no
Trigger Current Measurement
END Keep State
START
Is Current Measurement ready?
yes
no
Is Offset Calibration ongoing
no
yes
Convert and Filter result
Calculate Current Offset
END Keep State
START
Is Load Voltage Measurement ready?
yes
no
Convert and Filter result
END Keep State
Execution Path
State Transition

Figure 17 External ADC measurement flow chart

3.1.3 Diagnostic system

The diagnostic system is designed to detect and store all errors and warnings, indicate them, and subsequently determine the appropriate state for the system and switch. This is achieved through a continuous loop of requesting, processing, and resetting the debug registers for all peripheral ICs, implemented using state machines. All internal and peripheral errors/warnings are aggregated into an error/warning bitmap for efficient communication and utilization.

Figure 18 shows an overview of the diagnostic actions. The following are three distinct error/warning sources:

- Registers of the safety switch driver
- Registers of the analog frontend
- Internal checks

During each cycle, these resources are refreshed to obtain the current error/warning status. For each source, all related errors are cleared from the error bitmap, current errors are detected, and the error bitmap is updated accordingly. Whenever possible, actual errors on the ICs are automatically reset. A detailed concept for all three sources is illustrated in Figure 19.

After refreshing the error/warning bitmap, the errors are latched to ensure they accumulate and are not lost (for example, for subsequent external communication via CAN, display, and so on). If any errors are present, the BMS status is set to "ERROR", which will subsequently disable the safety switch in the next step. Note that any fault interrupts from peripheral IC error pins will immediately set the system to a safe state and deactivate the output (safe state pin of the intelligent gate driver). If no errors are made, the safe state is clear.

Finally, the behavior of the red LED is controlled. When errors exist, the LED remains on until the errors are resolved. If no errors are detected, the LED turns off. If the state of charge (SoC) falls below a certain threshold limit, the LED blinks. The LED directly indicates current errors, while the display shows all errors that have occurred, including those that have been resolved, until they are manually reset (latched bitmap).

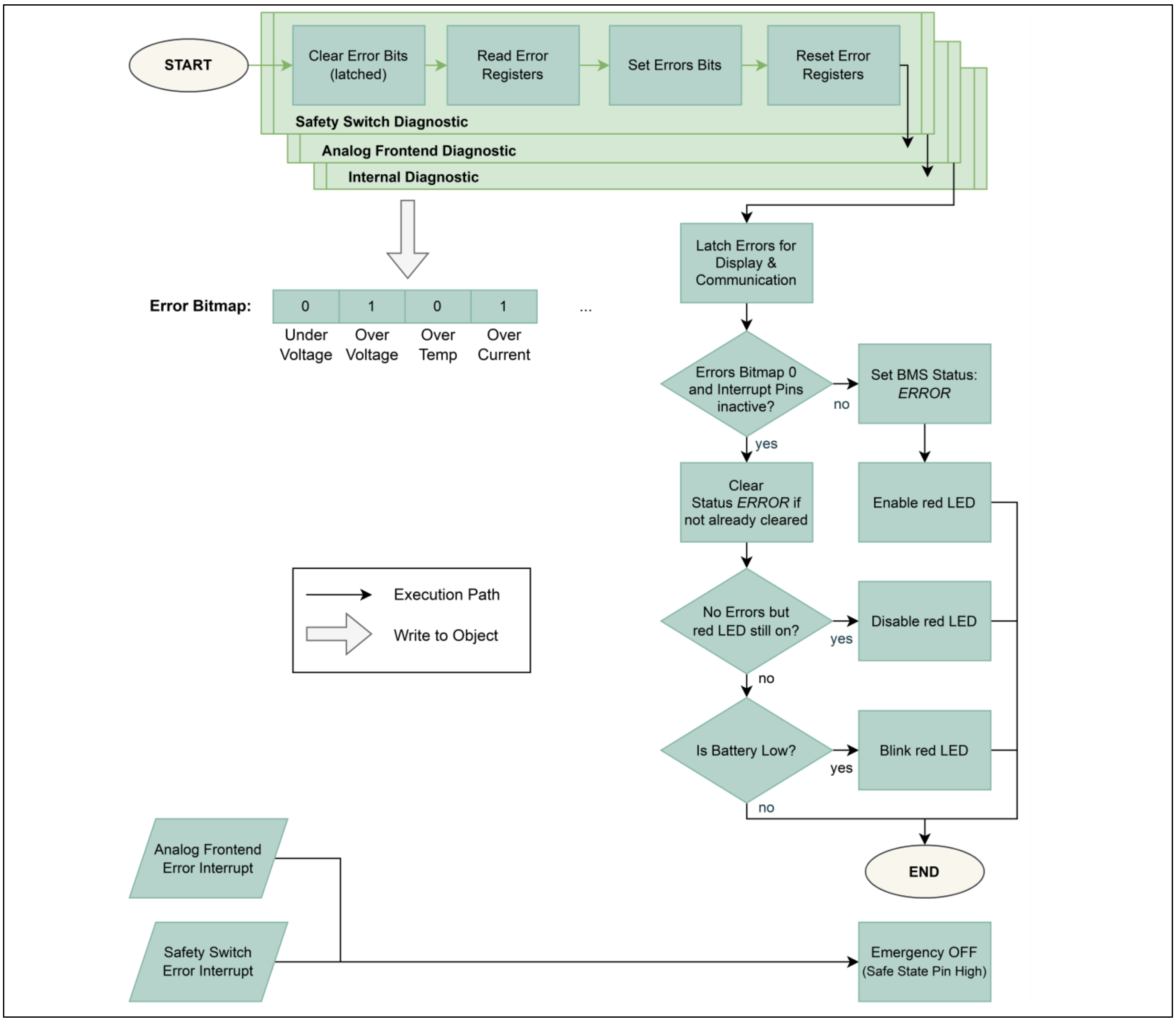

START
Clear Error Bits (latched)
Read Error Registers
Set Errors Bits
Reset Error Registers
Safety Switch Diagnostic
Analog Frontend Diagnostic
Internal Diagnostic
Error Bitmap:
0
1
0
1
...
Under Voltage
Over Voltage
Over Temp
Over Current
Latch Errors for Display & Communication
Errors Bitmap 0 and Interrupt Pins inactive?
no
Set BMS Status: ERROR
yes
Clear Status ERROR if not already cleared
Enable red LED
No Errors but red LED still on?
yes
Disable red LED
no
Is Battery Low?
yes
Blink red LED
no
END
Execution Path
Write to Object
Analog Frontend Error Interrupt
Safety Switch Error Interrupt
Emergency OFF (Safe State Pin High)

Figure 18 Diagnostic system overview

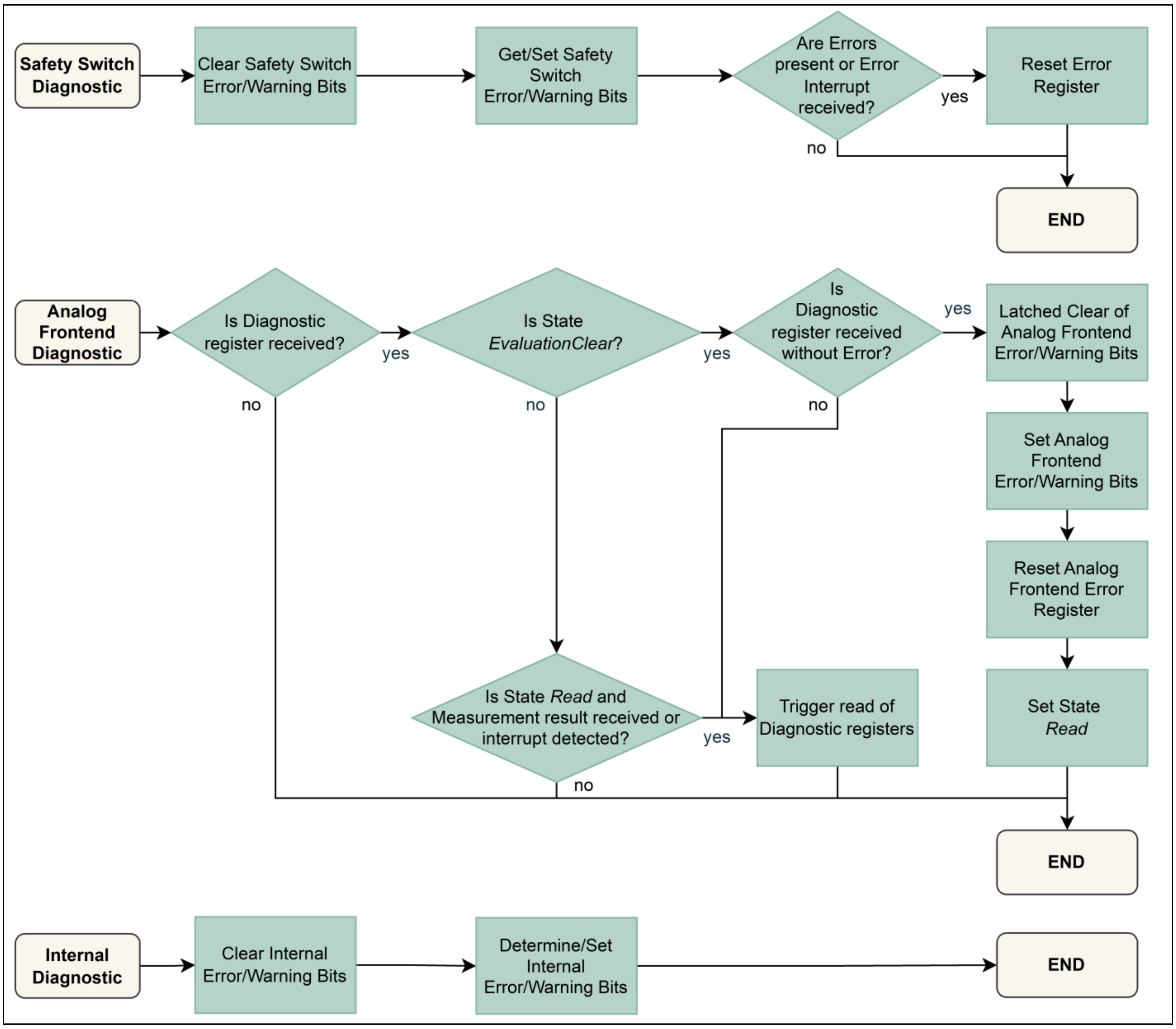

Safety Switch Diagnostic
Clear Safety Switch Error/Warning Bits
Get/Set Safety Switch Error/Warning Bits
Are Errors present or Error Interrupt received?
yes
no
Reset Error Register
END
Analog Frontend Diagnostic
Is Diagnostic register received?
yes
no
Is State EvaluationClear?
yes
no
Is Diagnostic register received without Error?
yes
no
Latched Clear of Analog Frontend Error/Warning Bits
Set Analog Frontend Error/Warning Bits
Reset Analog Frontend Error Register
Is State Read and Measurement result received or interrupt detected?
yes
no
Trigger read of Diagnostic registers
Set State Read
END
Internal Diagnostic
Clear Internal Error/Warning Bits
Determine/Set Internal Error/Warning Bits
END

Figure 19 **Detailed flowchart of three diagnostic sources**

3.1.3.1 Analog frontend diagnostic

The diagnostic process for the analog frontend adheres to the queue-interpreter concept (see Section 3.1.11), dividing it into two states (Figure 19, middle branch). This state machine continuously reads the diagnostic register as quickly as possible. As long as a read is triggered but not yet received, no code is executed. When data is ready and the state is "EvaluationClear," the system clears the analog frontend-specific bits from the error map, sets them again as read, resets the error on the IC, and transitions to the "Read" state. The diagnostic register of the analog frontend is only updated with each Round Robin (RR) cycle (see manual of the TLE9012), which can result in receiving an error-less diagnostic register even if errors are present. This situation occurs specifically when the error register is reset and read shortly before the RR cycle completes. Therefore, a latched error clear concept is implemented, requiring an error to be absent for a specified period before the error bit is reset. All cell, balancing, temperature, and communication errors are defined and listed in Table 9.

3.1.3.2 Safety switch diagnostic

The safety switch diagnostic follows the established principles of clearing, setting, and resetting errors (refer to Figure 19, top branch). This diagnostic is crucial for communication and display purposes, as the IC will deactivate the switch upon detecting any errors. The safety switch driver recognizes errors when its voltage and current limits are exceeded. These limits are configured to be wider than the system's specified maximum and minimum ratings, triggering only in extreme conditions, such as short circuits or transients. All detectable errors are listed in the corresponding Table 9.

3.1.3.3 Internal diagnostic

System specification limits and logical errors are detected internally. These errors are checked directly within the main loop and do not necessitate any peripheral IC reset. Different over-current thresholds for charging and discharging, as well as the maximum and minimum battery voltage ratings, are applied. Additionally, general communication errors with the analog frontend and external ADC are indicated. All errors are detailed in the corresponding table.

3.1.3.4 Error codes

If any errors occur, the error diagnostic subsystem sets error bits inside the dedicated bitmap. For each error, a code is shown on the display, up to a maximum of '3'. At every third display refresh, the actual value of the bitmap is shown on the display to allow the user to extract all set errors by manual check of the bit position. Table 9 explains the meaning of each error and its bit position:

Table 9 List of errors detected with their code and bit position in the bitmap

Bit	Code	Name	Description
1	SI	SSW_INTERRUPT	Error interrupt from safety switch
2	SUV	SSW_VBAT_UNDERVOLTAGE	Safety switch Battery undervoltage
3	SOV	SSW_VBAT_OVERVOLTAGE	Safety switch battery overvoltage
4	SVD	SSW_VDD_UNDERVOLTAGE	Safety switch supply undervoltage
5	SOT	SSW_CHIP_OVERTEMPERATURE	Safety switch overtemperature
6	SAO	SSW_VDS_OVERVOLTAGE_A	Safety switch MOSFET A overvoltage
7	SAU	SSW_VGS_UNDERVOLTAGE_A	Safety switch MOSFET A gate undervoltage
8	SBO	SSW_VDS_OVERVOLTAGE_B	Safety switch MOSFET B overvoltage
9	SBU	SSW_VGS_UNDERVOLTAGE_B	Safety switch MOSFET B gate undervoltage
10	SOC	SSW_OVERCURRENT	Safety switch overcurrent
11	SCU	SSW_CHARGEPUMP_UNDERVOLTAGE	Safety switch charge pump undervoltage
12	SSS	SSW_SAVESTATE_ENABLED	Safety switch emergency off enabled
13	FI	CAF_TLE9012_INTERRUPT	Error interrupt from analog frontend
14	COL	CAF_OL_ERR	Analog frontend cell open-loop (missing)
15	FIC	CAF_INT_IC_ERR	Analog frontend internal error
16	FCE	CAF_REG_CRC_ERR	Analog frontend internal error
17	COT	CAF_EXT_T_ERR	Analog frontend external overtemperature
18	FOT	CAF_INT_OT	Analog frontend internal overtemperature
19	CUV	CAF_CELL_UV	Analog frontend cell under-voltage

Bit	Code	Name	Description
20	COV	CAF_CELL_OV	Analog frontend cell over-voltage
21	BUC	CAF_BAL_UC	Analog frontend balancing undercurrent
22	BOC	CAF_BAL_OC	Analog frontend balancing overcurrent
23	FPS	CAF_PS_SLEEP	Analog frontend power supply fault
24	FAD	CAF_ADC_ERR	Analog frontend ADC error
25	IQF	INT_TLE9012_QUEUE_FAILED	Internal queue error for analog frontend
26	IOC	INT_OVERCURRENT	Internal overcurrent threshold exceeded
27	ISV	INT_SWITCH_ON_VETO_RECEIVED	Another BMS sent switch-on prevention
28	IAT	INT_ADC_COM_TIMEOUT	External ADC communication error
29	IOV	INT_VBAT_OV	Battery overvoltage
30	IUV	INT_VBAT_UV	Battery undervoltage
31	ICE	INT_TLE9012_COM_ERROR	Analog frontend communication error

3.1.4 Safety switch control and state decision

Based on the diagnostic results, the system determines its state and controls the safety switch accordingly. A simplified scheme of this process is shown in Figure 20. The primary decision point is whether the diagnostic set the BMS state to “ERROR”. If so, the switch is deactivated, and all related objects are reset. If no errors are present, the BMS state is set to “DISCHARGING” or “CHARGING” based on the slot configuration. Subsequently, the system evaluates whether any conditions necessitate a hot-swap to another system. This is the stage where a shutdown is executed if the hot-swap has failed or completed. The BMS Role state machine is then executed, defining the tasks for the current BMS, whether it is the primary supplying BMS (“MAIN BMS” role) or involved in a hot-swap (“GIVER BMS” and “TAKER BMS” role).

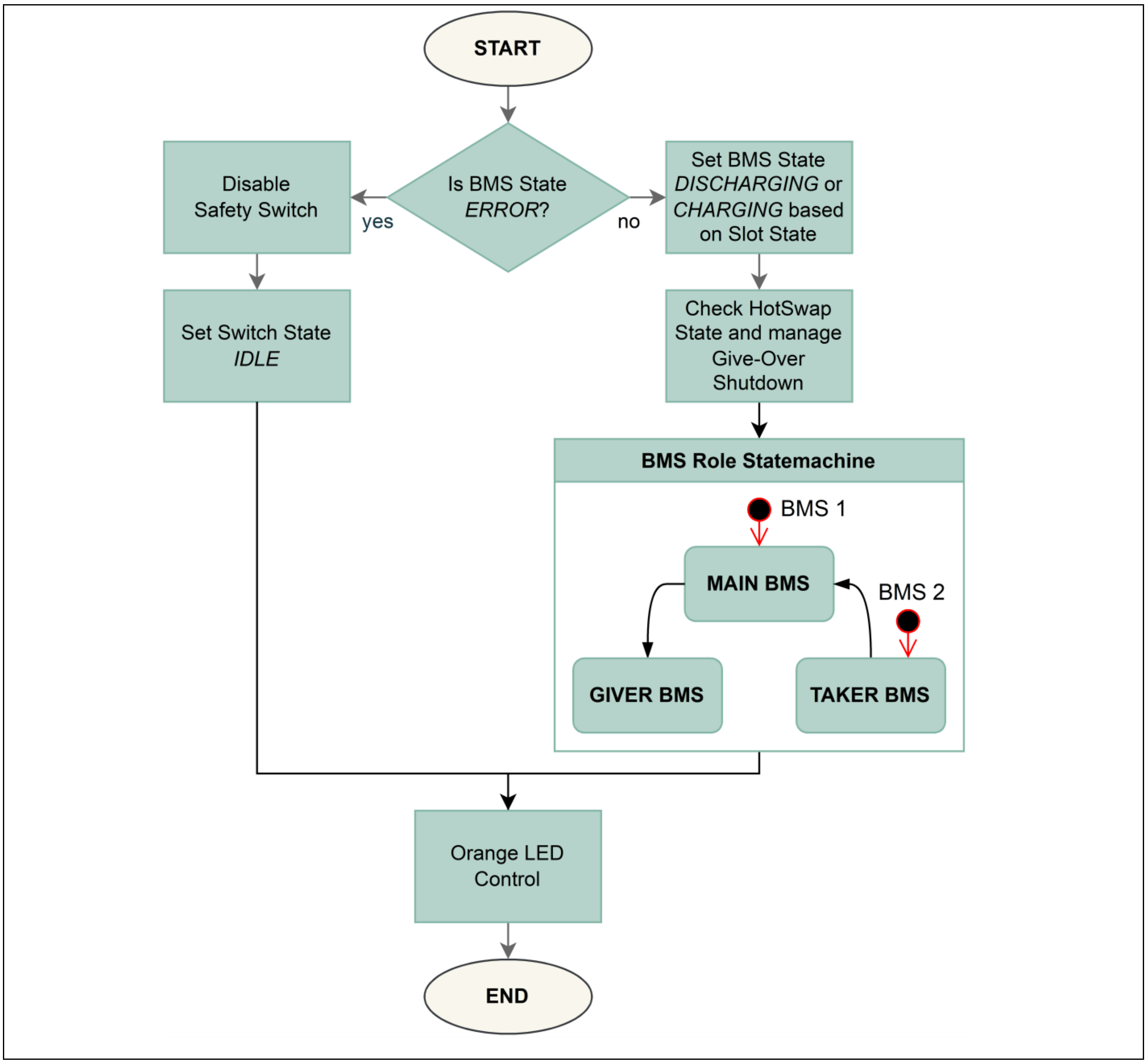


Figure 20 **Overview of the state and switch control**

3.1.4.1 Dynamic pre-charge and switch-on

The first BMS to start is assigned the “MAIN” role, managing the initial switch-on and pre-charge via a state machine (see Figure 21). The process begins in the “IDLE” state, where the state machine attempts to initialize pre-charge if all conditions are met, including switches being off, low load voltage, and retry attempts within limits. If these conditions are satisfied, the pre-charge switch path is activated, transitioning the state to “PRECHARGE”.

In the “PRECHARGE” state, the system monitors the load voltage change rate to detect if the pre-charge is complete. After a minimum time has elapsed and either the maximum time is exceeded or the rate stabilizes below a threshold, indicating sufficient charge, the main switch path is activated, and the state returns to “IDLE”.

The “IDLE” state remains constant while the output is on, with state changes occurring only if the switch is turned off. This behavior persists even after a hot-swap.

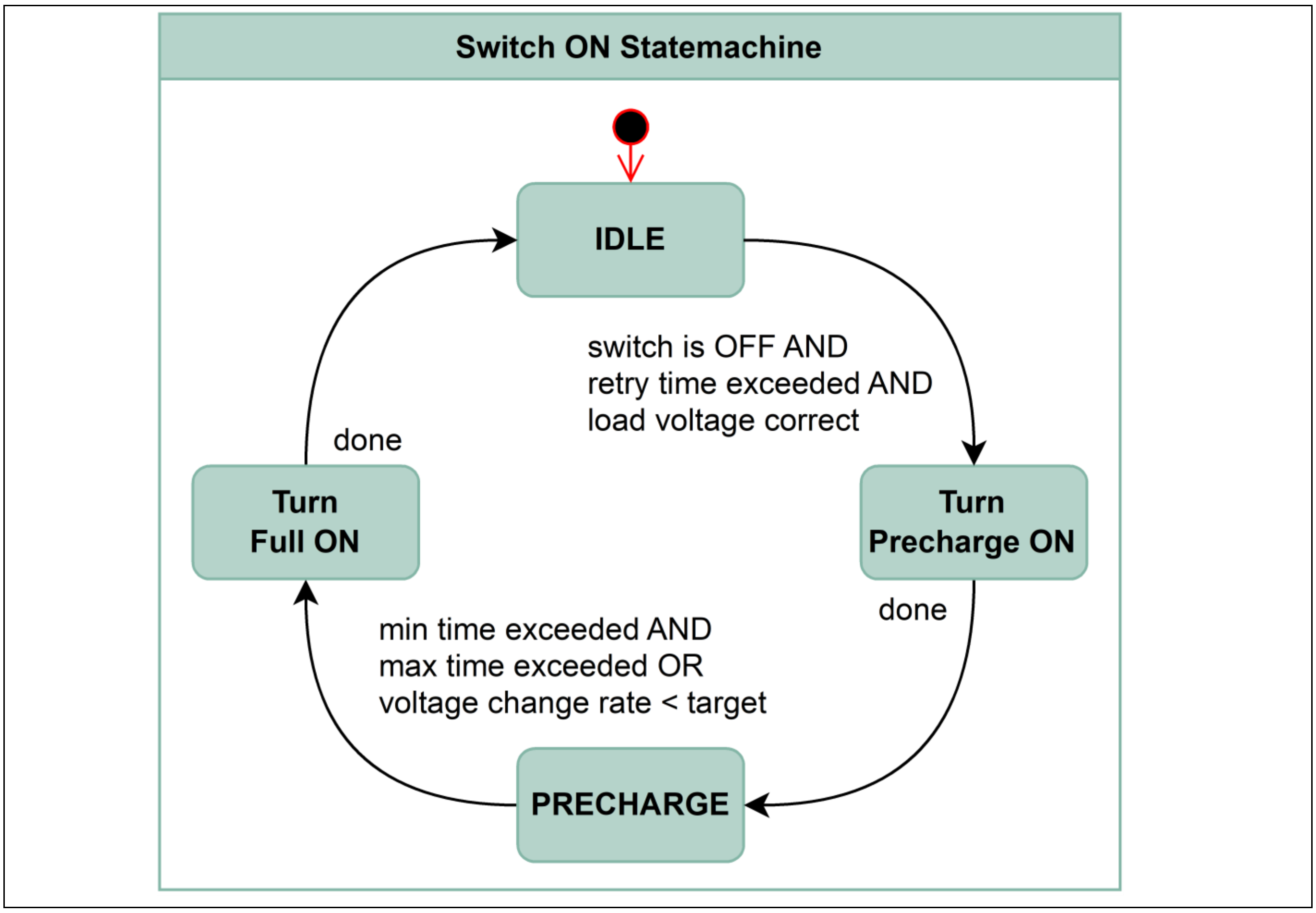


Figure 21 **Pre-charge state machine**

3.1.4.2 Hot-swap

As depicted in Figure 20 , the BMS role state machine includes three states: “MAIN”, “TAKER”, and “GIVER”. The first BMS to start up always assumes the “MAIN” role. The second BMS enters the “TAKER” role, where it remains inactive until a give-over handshake is initiated.

The detailed interaction of these roles is shown in the Figure 22. The MAIN BMS manages the soft switch-on with pre-charge. Upon detecting a hot-swap give-over condition, the role changes to “GIVER”. This initiates a CAN-based communication scheme to transfer the supply from “MAIN” to “TAKER”.

The “GIVER” starts the process by sending a handshake to the “TAKER” to confirm its presence and responsiveness. Upon receiving a response, the “GIVER” instructs the load to reduce power demand to a safe level to prevent hardware damage.

Next, the “GIVER” sends a swap command to the “TAKER”, prompting it to activate its pre-charge path and acknowledge the command. Using the pre-charge path helps limit the compensation current between the empty and full batteries, preventing safety features from triggering or system damage due to the high current capabilities of the batteries.

Upon receiving the swap command return, the “GIVER” turns off all its paths, sends a finish message to the “TAKER”, and shuts down. The “TAKER”, upon receiving this message, activates its main path fully and instructs the load to lift the power reduction, then transitions to the “MAIN” role.

A timeout check is performed at every step of the scheme. If a timeout occurs, both BMS units will shut down.

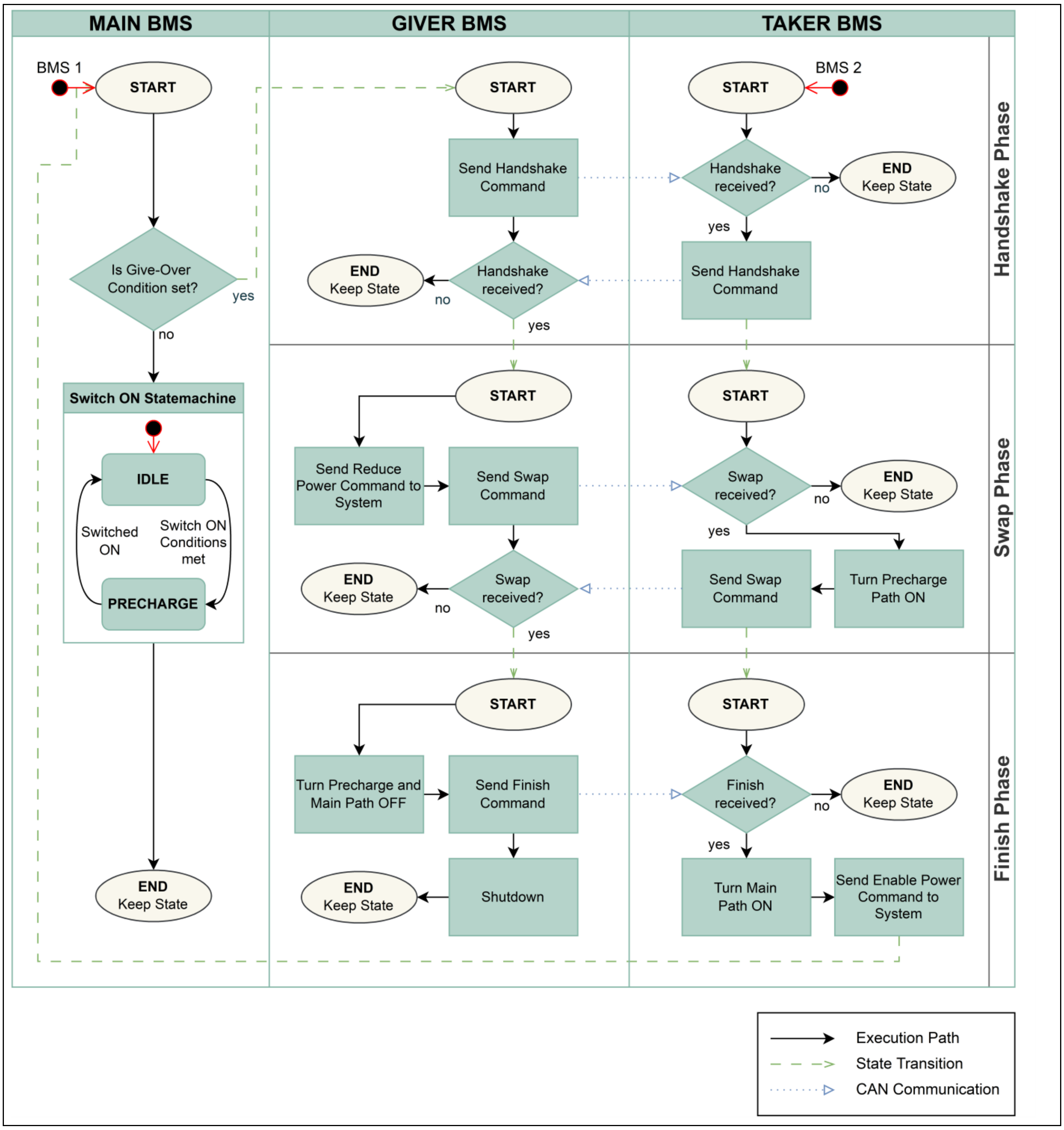


Figure 22 **Detailed flow chart of the roles, switch on, and hot swap**

3.1.5 Balancing

A simple threshold algorithm is used for cell balancing. If the voltage imbalance among cells is within an acceptable range, no balancing is necessary. However, if the imbalance exceeds this range, cells with voltages higher than the highest voltage minus an offset are balanced. This process involves sorting the cells by voltage and determining the imbalance by subtracting the lowest voltage from the highest voltage.

For system implementation, synchronization with the analog frontend communication is required, as all necessary values are read from it and the balancing is controlled by it. As shown in Figure 23, the balancing calculation is executed only when new results are received, and a specific interval has elapsed. In such cases, a selection sort algorithm ranks the cells by voltage. After ranking, the imbalance value is calculated and filtered.

If the system is in charging mode and the imbalance exceeds the threshold, the highest cell voltage is stored as a reference. In all other cases, balancing is halted. Any cell voltages within the range of the reference voltage minus the balancing goal is marked in a bitmap. If the bitmap is empty for any reason, balancing is deactivated. Conversely, if the bitmap is not empty, the value is stored internally and sent to the analog frontend after each measurement cycle (see Figure 16). This step is essential because any voltage measurement disables the balancing. Reactivating balancing as soon as possible after measurement maximizes active balancing time and enhances balancing performance.

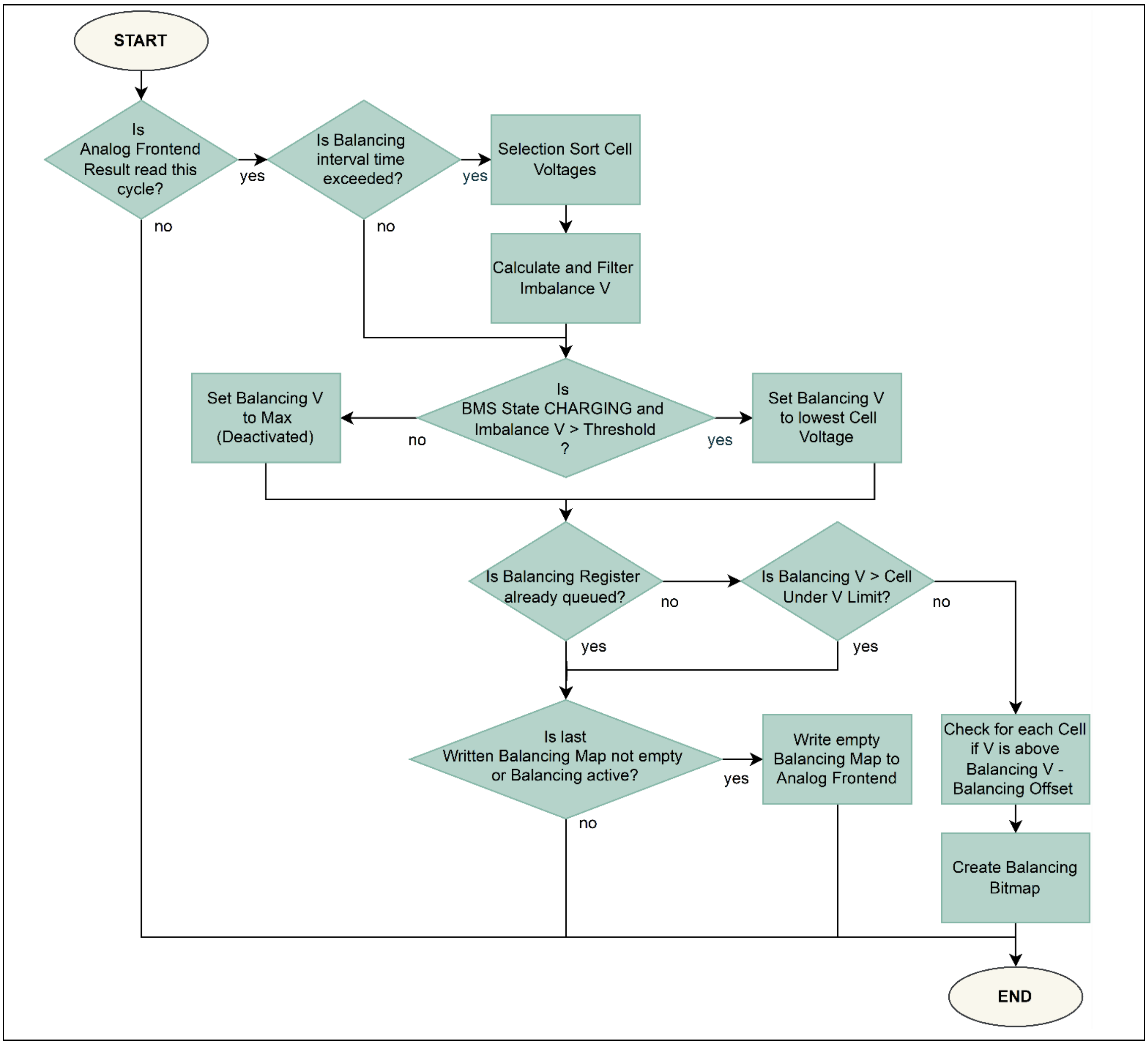

START
Is Analog Frontend Result read this cycle?
yes
no
Is Balancing interval time exceeded?
yes
no
Selection Sort Cell Voltages
Calculate and Filter Imbalance V
Is BMS State CHARGING and Imbalance V > Threshold ?
no
yes
Set Balancing V to Max (Deactivated)
Set Balancing V to lowest Cell Voltage
Is Balancing Register already queued?
no
yes
Is Balancing V > Cell Under V Limit?
no
yes
Is last Written Balancing Map not empty or Balancing active?
yes
no
Write empty Balancing Map to Analog Frontend
Check for each Cell if V is above Balancing V - Balancing Offset
Create Balancing Bitmap
END

Figure 23 **Detailed flowchart of the balancing**

3.1.6 Cell monitoring and state

The battery state estimation utilizes a simple implementation of the coulomb counting method. At initialization, the SoC and SoH are retrieved from memory or determined to establish a baseline for battery state estimation. Note that this feature is abstracted to *battery_state.c and battery_state.h*. To implement another algorithm, check the header of the file for instructions on how to do this.

3.1.6.1 State of charge (SoC)

To determine the SoC, initially the remaining charge in the battery is estimated. If the voltage is in the linear regions of the stored open circuit voltage (OCV) curve, the initial state is based on it. The OCV curve describes the SoC of the battery at a given voltage when no load is applied. In the non-linear regions, the last stored SoC is used.

Subsequently, during the main loop, the current is integrated over time at each measurement to track changes in the battery's charge. This method directly monitors the charge level. The SoC is then obtained by comparing the current charge state to the maximum charge state.

3.1.6.2 State of health (SoH)

At startup the SoH is always read from memory because it cannot immediately be estimated using the implemented coulomb counting method. For the estimation, a full charge cycle is typically monitored. If the amount of charge passing into the battery differs from that of a new battery, the SoH is adjusted accordingly.

To improve applicability, SoH is estimated only during charging because integration errors due to load fluctuations are minimal. Additionally, only a specific region of the charge cycle is monitored, as the battery is rarely charged from 0%. This provides a rough estimation of SoH, though it may not be perfectly accurate.

3.1.6.3 Usable remaining time

The calculation of usable remaining runtime differs for charging and discharging scenarios. During discharging, this value indicates how long the battery will last under the current load until a certain SoC threshold is reached. Equation 1 shows this calculation, where the difference between the currently available capacity (C_{actual}, estimated by the SoC algorithm) and the rated capacity (C_{rated}) compensated by the missing SoH and the minimal usable SoC is calculated. Dividing this result by the system current at a given time yields the momentary remaining time.

$$T_{R\_Discharge} = \frac{C_{actual} - C_{rated} \cdot SoH_{\%} \cdot SoC_{\%\_useful\_remain}}{I}$$

Equation 1 Discharge time calculation

During charging, the remaining time until the battery is fully charged, assuming the current remains constant, is estimated, as shown in Equation 2. The remaining charge required to fully charge the battery is calculated and divided by the current. Note that the current must be negated since the current sensor detects charging currents as negative.

$$T_{R\_Charge} = \frac{C_{rated} \cdot SoH_{\%} - C_{actual}}{-I}$$

Equation 2 Charge time calculation

3.1.7 Memory

The FRAM memory is divided into two parts (see Figure 24). The first part stores system status data, such as the last State of Charge (SoC) and State of Health (SoH) of the BMS. This data is only updated if the current value differs from the stored value.

The second part stores real-time (RT) data while the system output is enabled. It is designed to capture full operational cycles (discharge or charge) and captures all values presented in Table 10. Note that the value of the current can later be calculated through division of the charge and time difference.

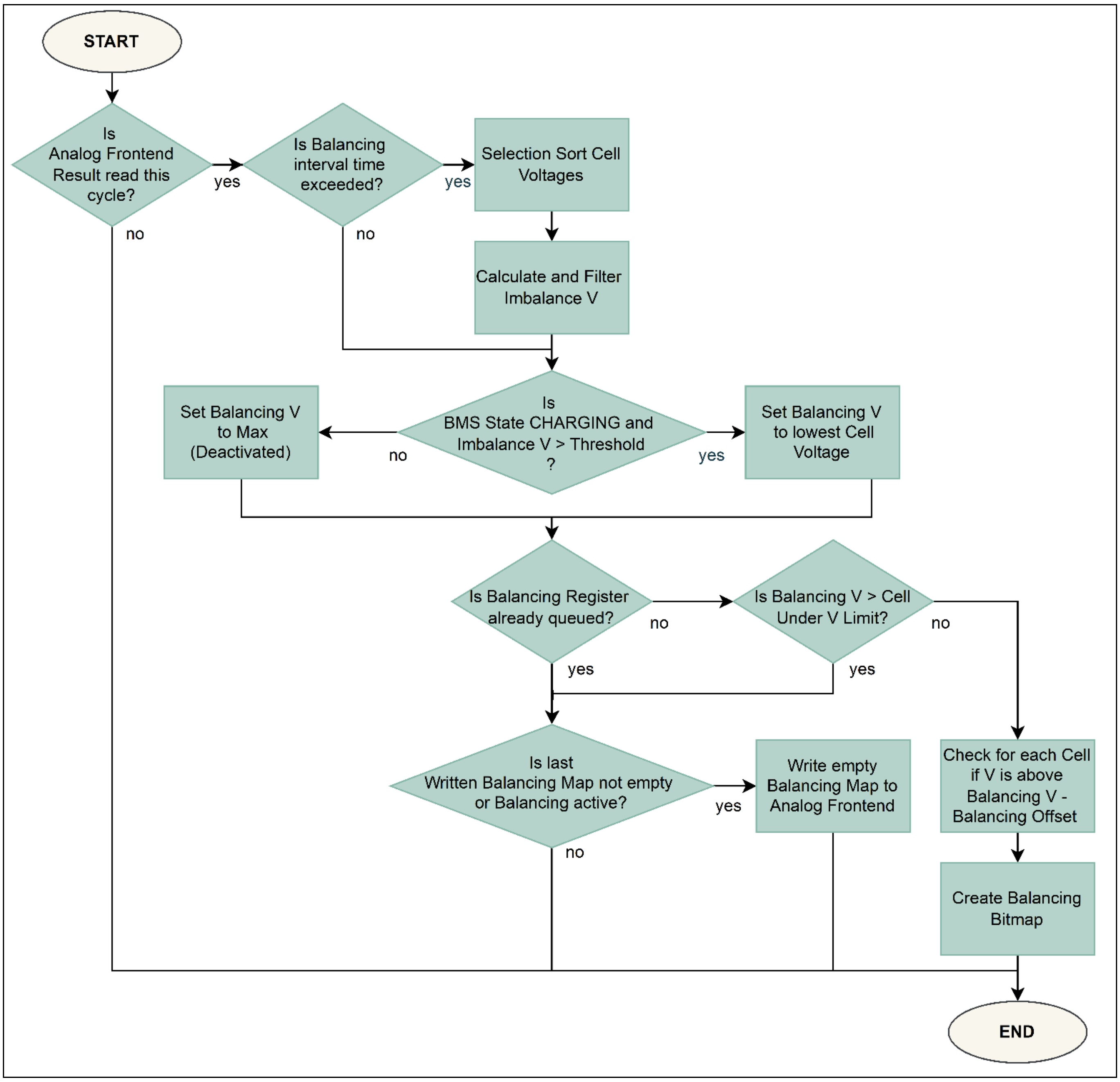

START
Is Analog Frontend Result read this cycle?
yes
no
Is Balancing interval time exceeded?
yes
no
Selection Sort Cell Voltages
Calculate and Filter Imbalance V
Is BMS State CHARGING and Imbalance V > Threshold ?
no
yes
Set Balancing V to Max (Deactivated)
Set Balancing V to lowest Cell Voltage
Is Balancing Register already queued?
no
yes
Is Balancing V > Cell Under V Limit?
no
yes
Is last Written Balancing Map not empty or Balancing active?
yes
no
Write empty Balancing Map to Analog Frontend
Check for each Cell if V is above Balancing V - Balancing Offset
Create Balancing Bitmap
END

Figure 24 **Detailed flow chart of the memory functions for status retention and real-time data recording**

For optimal compactness and usability, values are stored in 2 byte little-endian format using fixed-point representation. This involves multiplying values by a power of 10 to fit all desired decimal places within a 16-bit integer.

To achieve precise time resolution with minimal storage demand, the time difference between consecutive points is stored rather than the absolute time from the start. During data evaluation, these time differences can be summed up to obtain the absolute time. This field also marks system resets by showing a value of “0”.

The value of the current is not stored directly; instead, it is calculated by dividing the charge difference by the time difference. The state of charge (Coulomb Counting) is refreshed at the same rate as the current measurement. However, values for time and charge differences are updated at a higher rate, providing a more accurate average current between lines while reducing outliers and noise. To prevent charge loss when converting from floating-point to fixed point, a compensation mechanism is implemented. After each calculation, the integer value is summed up and used next time to subtract from the current floating-point state to get the charge difference. This ensures the system tracks which charge has been accounted for. When lost decimals accumulate to ‘1’, they are considered in the next line. While this approach may not provide the highest precision per line, it maintains overall accuracy when integrating lines over time. The value is stored in ampere-seconds with four decimal places moved to a fixed point (see Table 10).

For the temperature field, the system uses the average temperature of the two sensors located near the cells. The block voltage and imbalance values are stored directly. This results in a total of 10 bytes per line.

During RT data recording, it is advantageous to record a full battery cycle before the battery is fully charged. However, the optimal sampling rate depends heavily on load conditions. With a high current draw, the battery depletes faster, allowing for shorter sampling times. Conversely, low load currents may necessitate longer times to drain the battery. To account for this, a dynamic sampling time estimation is implemented. The system divides the usable remaining runtime ($T_{remaining}$) by the number of remaining lines ($n_{remaining}$) to determine the sampling time. This adaptive sampling time ensures the recording of a full charge or discharge cycle.

$$T_{sampling} = \frac{T_{remaining}}{n_{remaining}}$$

Equation 3 Estimating the sampling time

If a higher data resolution is preferred over completeness, a static record rate can be set in the settings, and the recording can be reset via CAN messages. At the default set maximum specified current of 6 A, the highest possible capacity difference value of 32.767 ampere-seconds is reached after 5.4 seconds (dividing charge by current). This must be considered when choosing the record rate. If the value still does not fit, the maximum value will be used, and the mechanism to compensate for lost decimals will distribute the actual current to later values where it fits. This is easily detected during evaluation, as some points will reach the int16 maximum value of 32767.

Table 10 List of fields that are stored per record line

Name	Type	Unit	Min	Max
Time difference	uint16	ms	0	65535
Capacity difference	int16	As	-32.767	32.767
Battery voltage	uint16	mV	0.0	65.535
Cell imbalance	uint16	mV	0.0	65.535
Temperature	int16	°C	-327.67	327.67

3.1.8 Shutdown and start

The 5 V and 3.3 V supply required for all components is managed by an intermediate buck-converter connected directly to the battery voltage. The enable pin of this converter is pulled low by default and can be set high by pressing the start button or via the “VCC_HOLD” pin on the processor. This design allows the system to be started manually using the button and kept operational by activating the “VCC_HOLD” pin until shutdown.

The controller directly manages the logical power supply by maintaining the “VCC_HOLD” pin in a high state. Therefore, there must be always a mechanism to shut down the system, even if code execution fails. However, a controlled (soft) shutdown procedure is preferable to ensure the system ends in a defined state and to prevent interruption of the last display refresh. This is especially critical with the E-Ink display used, as refreshing can take more than a second, and an incomplete refresh will result in undesirable behavior. For a proper shutdown, the switch is disabled, the display is refreshed to the main screen, and the “VCC_HOLD” pin is set low, stopping the logical power supply.

The shutdown process can be triggered by the shutdown button, deep discharge protection, or certain logic states. If a BMS in the “MAIN” role is shut down via the button, it attempts to hand over to a “TAKER” (see Section 3.1.4.2). If any part of the hot-swap fails, the system performs a soft shutdown. Similarly, the “TAKER BMS” will execute a soft shutdown if the load voltage disappears, indicating it is unable to take over.

In any scenario, the watchdog will enforce the shutdown after a specified time following the shutdown button press. Additionally, continuously pressing the shutdown button for a set duration will result in a hard shutdown. This is monitored by both processor cores.

3.1.9 Watchdog

Two watchdogs are implemented - a software watchdog on a short interval and a hardware watchdog on a long interval. Both watchdogs must be kicked faster than their trigger intervals to keep the system running.

The software watchdog attempts a controlled shutdown if the hot-swap takes too long or if the shutdown button is pressed continuously. It can also perform a hard shutdown if the main loop cycle count does not increase.

The hardware watchdog, integrated into the microcontroller hardware, triggers a system reset if a specific function is not executed within a defined timeframe. This hardware watchdog will act even if the processor cores are in an error state and no code is being executed.

3.1.10 Slot detection

Slot detection within the BMS is achieved through a voltage divider circuit, which incorporates a fixed resistor located on the BMS hardware and a variable 'coding' resistor found on the slots. The internal ADC measures the output voltage from this voltage divider to determine the specific slot in which the BMS is inserted.

The detection process is executed on every main loop. The measured voltage is compared against predefined thresholds corresponding to each slot state: None, System 1, System 2, and Charger. This arrangement allows the system to accurately identify the active slot and respond accordingly to maintain operational integrity and prevent conflicts during hot-swapping scenarios.

Table 11 lists the detectable slots, their detection ranges, and the typical coding resistor used on the BMS mainboard.

Table 11 List of detectable slots

Slot	Voltage range for detection	Resistance value
“None”	> 2.578 V	Infinite (open)
“System 2”	1.800 – 2.578 V	13.0 kΩ
“System 1”	0.705 – 1.800 V	7.5 kΩ
“Charger”	< 0.705 V	0.0 kΩ

3.1.11 Analog frontend library

The analog frontend connects to the battery main terminals and all tabs between the series cells. A balancing circuit between the analog frontend and the battery incorporates feedback circuitry and balancing load resistors. The internal switches of the analog frontend perform the balancing over these resistors, setting the current to approximately 100 mA as recommended in the datasheet. Feedback is also used for cell and system voltage measurements with a 16-bit resolution. Additionally, five external NTC sensors and internal temperatures can be read with a 10-bit resolution.

This device incorporates numerous features designed to enhance performance and ensure system safety and expandability. For instance, interfacing is performed over an ‘isoUART’, which allows multiple devices (nodes) to be daisy chained. Communication is half-duplex, meaning sending and receiving cannot occur simultaneously. General diagnostics are conducted using a Round Robin (RR) scheme, performing diagnostic measurements, plausibility checks, and safety functions cyclically at a typical rate of 50 ms. Externally triggered measurements have higher priority and can postpone RR cycles multiple times before they are enforced. Error detection and error interrupt pin settings can be specifically adapted. Communication with the device uses different frame layouts depending on the action and is always CRC-protected to detect transmission errors. Optimization functions, such as multi-read, are implemented for the most needed registers. The device includes over 180 fields for measurement, diagnostic, balancing, and electrical settings.

The complex, half-duplex communication with the analog frontend IC requires a special interface to allow non-blocking operation according to the main design principles. For details check the files and their header text in the directory ‘*BMS_TLE9012*’ inside the CM4 subproject.

3.1.11.1 Interface library design

The interface library design specifies three command processing components (see Figure 25):

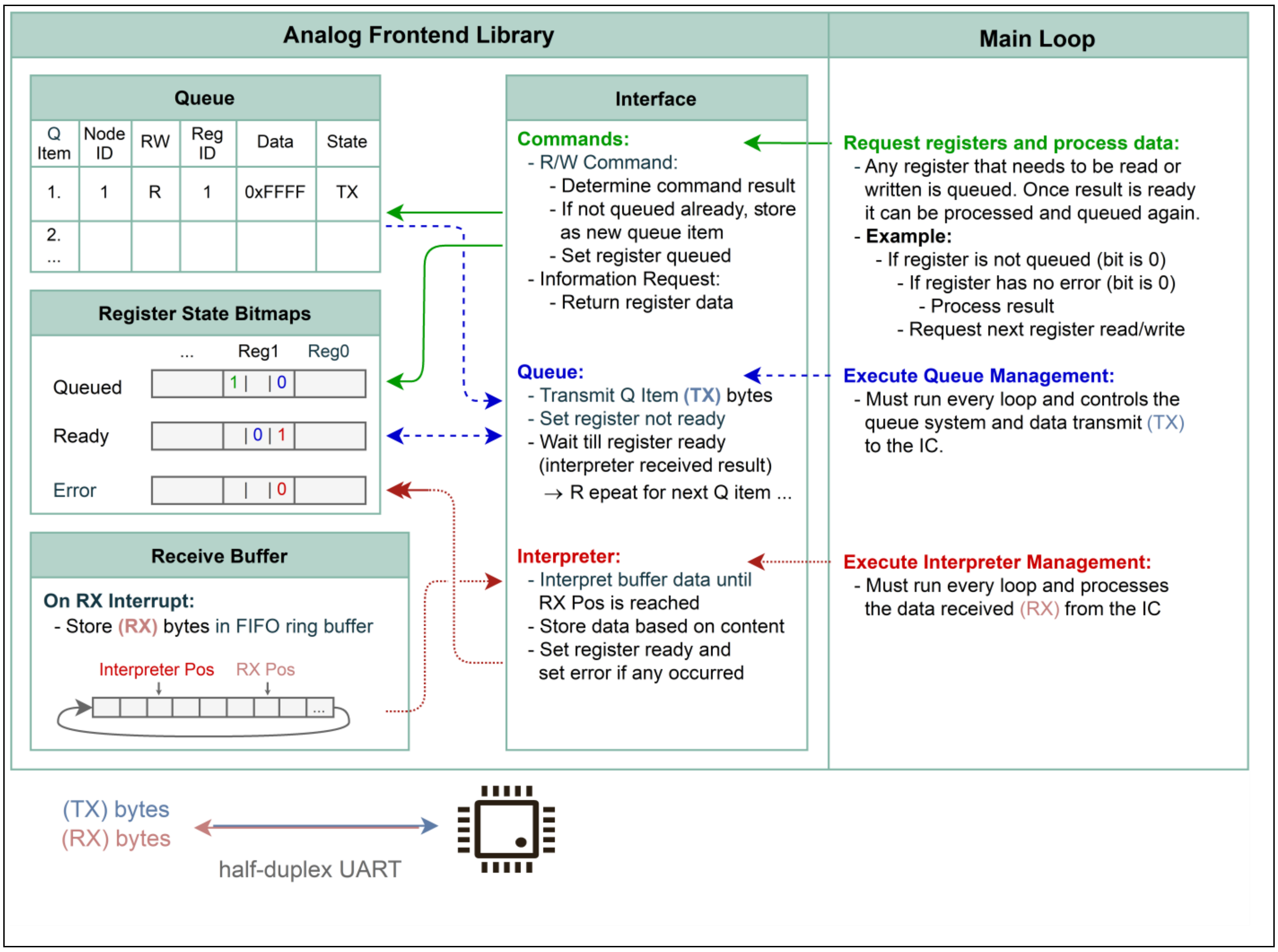

Analog Frontend Library
Main Loop
Queue
Q Item | Node ID | RW | Reg ID | Data | State
1. | 1 | R | 1 | 0xFFFF | TX
2. ...
Register State Bitmaps
... Reg1 Reg0
Queued 1 | 0
Ready 0 | 1
Error 0
Receive Buffer
On RX Interrupt:
- Store (RX) bytes in FIFO ring buffer
Interpreter Pos
RX Pos
Interface
Commands:
- R/W Command:
- Determine command result
- If not queued already, store as new queue item
- Set register queued
- Information Request:
- Return register data
Queue:
- Transmit Q Item (TX) bytes
- Set register not ready
- Wait till register ready (interpreter received result)
→ R epeat for next Q item ...
Interpreter:
- Interpret buffer data until RX Pos is reached
- Store data based on content
- Set register ready and set error if any occurred
Request registers and process data:
- Any register that needs to be read or written is queued. Once result is ready it can be processed and queued again.
- Example:
- If register is not queued (bit is 0)
- If register has no error (bit is 0)
- Process result
- Request next register read/write
Execute Queue Management:
- Must run every loop and controls the queue system and data transmit (TX) to the IC.
Execute Interpreter Management:
- Must run every loop and processes the data received (RX) from the IC
(TX) bytes
(RX) bytes
half-duplex UART

Figure 25 Interface library design

Note: *Any request from the main program generates queue items and marks the corresponding registers as queued. A dedicated queue handler detects new queue items, transmits their content to the IC, marks the registers as not ready, and waits for the result. The interpreter handler processes new data in the receive buffer, performs fault checks, and marks the registers as not queued and ready. The set ready bit informs the queue handler that the channel is free to send the next queue item.*

Commands can be called multiple times and anywhere during execution. These commands do not directly communicate with the analog frontend system but compose the action details and store them as queue items. These items are processed by a queue system that sends the current item's data over the bus and waits for the result before sending the next item. This ensures data is never sent and received over the half-duplex bus simultaneously. A special interpreter monitors the bus, records any data, and processes all packages once they are complete.

These three components work independently and communicate through special register state bitmaps. These bitmaps indicate if a register is currently queued, if there was an error in communication, and if the register is ready and not processed currently.

Any command will determine the register data that needs to be sent, whether a register is read or written, and which register ID must be addressed. After adding the queue item, the queued bit for the specific register is set to true. The state field of the queue item tells the queue handler if the item must be sent or if it must wait for the result. Once the queue handler finishes processing the last item, it sends the new one and marks the register as not ready. Meanwhile, an interrupt handler triggered by incoming communication adds data bytes automatically to a ring buffer. Every time the interpreter handler executes, it checks if a complete data package is in the buffer and tries to interpret it. If the package's CRC is valid, its data is stored in the shadow register (buffer for all register values). Finally, the interpreter marks the register as not queued and ready. If there were any errors, the error bit is set to true; otherwise, it is reset to false.

The interpreter does not interact with the queue items directly. It only interprets whatever arrives on the bus. If a package does not arrive or is invalid, the interpreter ignores it. Every part of this scheme is timeout-protected and will eventually set the error bit and discard the queue item.

3.1.11.2 Commands and data access

There are many commands for all important functions of the analog frontend. Most end up using a function to read or write a register. This function checks whether the register is ready and not already queued, returning an error if this is not the case. This ensures the system can react directly to the register state and prevents multiple requests for the same register. Commands can perform blocking operations with timeout protection, which is useful during initialization when actions depend on previous steps.

Once commands are finished, the data is written to a shadow register representing those in the analog frontend. Special conversion functions can then extract and convert the data into usable values. All this data is stored in objects representing each daisy-chain node used (see '*tle9012_node*' structure in '*BMS_TLE9012/tle9012_registers.c and BMS_TLE9012/tle9012_registers.h*').

3.1.11.3 Queue system

The queuing system focuses on preventing simultaneous transmission and reception over the bus. The basic principle is shown in Figure 26. It assumes that no incoming traffic occurs without a prior request. A queue item request starts in the "ExecuteTx" state, where it is sent over the bus and the register is marked as not ready. The state then changes to "WaitForRx", where the queue waits for the result or a timeout. Based on the outcome, the register bitmaps are updated. If another item is in the queue, the index changes to it. If not, the state changes to "WaitForNextItem", where the system stays until a new item is queued. Transmission for the next item (if available) is performed in the same cycle.

For safety, the analog frontend IC enters the sleep mode if a keep-alive command is not sent within a certain period, which in the current implementation has been set to 1300 ms. Commands can only be sent safely in the "ExecuteTx" state, which may delay transmissions by one cycle. If sleep mode is entered, re-initialization occurs at the queue handler start.

When adding a new queue item, a queue overflow check is performed. If the queue is full, the request is rejected, and an error code is returned to the caller.

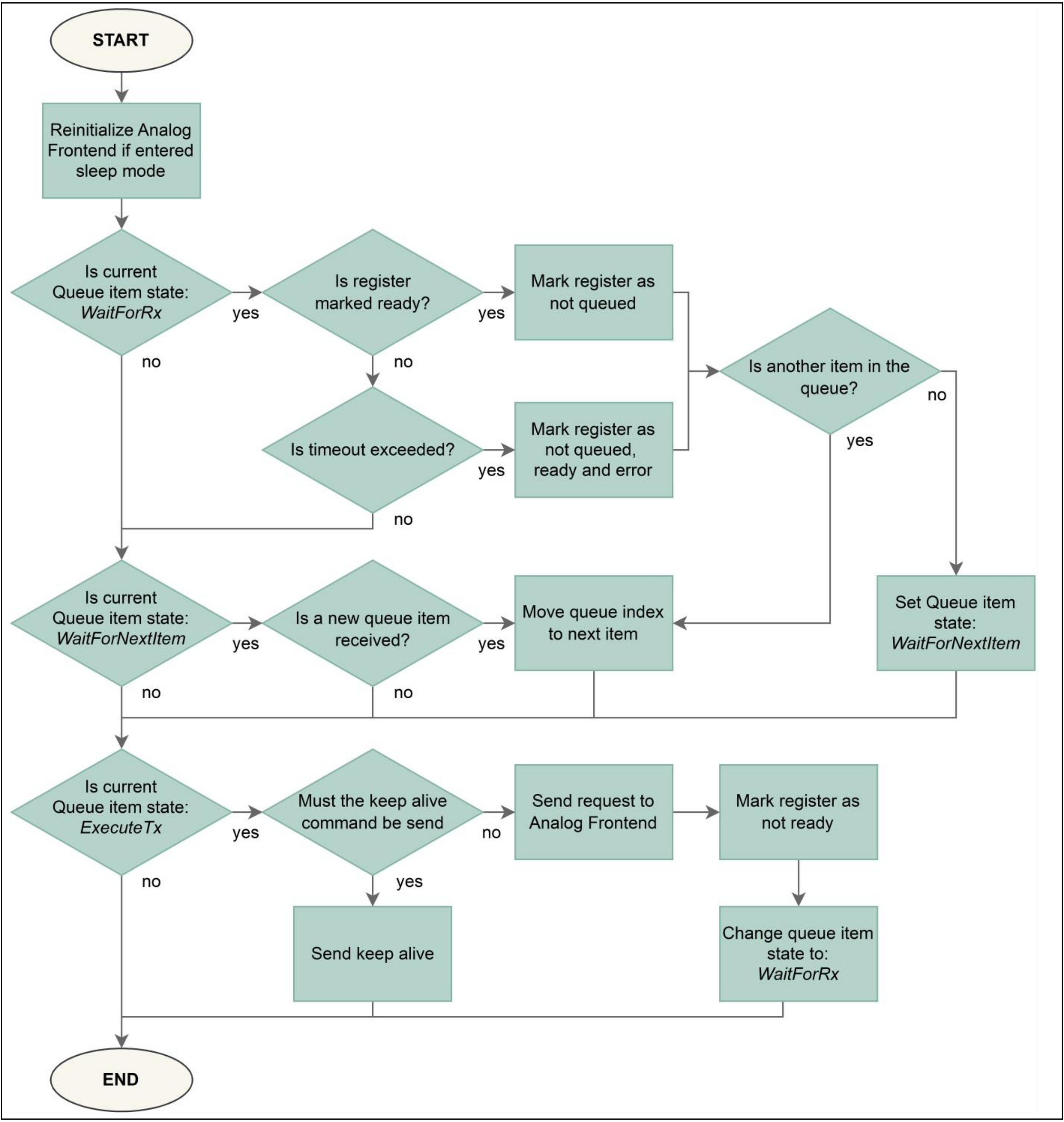

START
Reinitialize Analog Frontend if entered sleep mode
Is current Queue item state: *WaitForRx*
yes
no
Is register marked ready?
yes
no
Mark register as not queued
Is timeout exceeded?
yes
no
Mark register as not queued, ready and error
Is another item in the queue?
no
yes
Is current Queue item state: *WaitForNextItem*
yes
no
Is a new queue item received?
yes
no
Move queue index to next item
Set Queue item state: *WaitForNextItem*
Is current Queue item state: *ExecuteTx*
yes
no
Must the keep alive command be send
no
yes
Send request to Analog Frontend
Mark register as not ready
Send keep alive
Change queue item state to: *WaitForRx*
END

Figure 26 **Overview of the command queueing system implemented by the library**

3.1.11.4 Interpreter design

The interpreter design addresses multiple challenges. Each communication package consists of multiple frames and does not have a fixed size. The size varies between read and write packages and can be dynamic for special functions. The expected package size must be determined from the information of the frame bytes. To ensure validity, the last byte always contains a cyclic redundancy check (CRC). All packages start with a SYNC byte and an ID byte that specifies the node and whether it is for read or write. An ADDRESS byte defines the register. After this, packages differ based on type.

Write packages include two additional bytes for the DATA and the CRC byte. Here the analog frontend also sends a REPLY byte with the result and a 3-bit CRC. The result provides detailed information on the success of the write operation and any general errors.

Read packages are followed directly by the CRC byte. The IC returns an ID byte, 2 bytes for the DATA, and another CRC byte for its read return package. The multi-read feature allows the read command to receive multiple read return packages, one for each requested register. This feature must be configured at startup to specify which registers to expect; otherwise, the interpreter cannot determine the package length and will fail.

An overview of the interpreter concept handling these conditions is provided in Figure 27. The logic is surrounded by a while loop that tries to process all complete packages in the receive buffer until insufficient bytes are left to form a package. This allows the interpreter to process all available replies in one cycle.

Inside the loop, there are three stages to identify and process the data. The first stage checks if the first byte matches the SYNC frame. If so, the execution path splits depending on whether the package is marked for read or write. Both cases verify if the buffer has sufficient data for the package type and if the CRC is valid. On success, the interpreter state is set to “ParseReadReturn”. For valid write packages, the data and reply byte are directly interpreted. If the CRC fails, the interpreter state is set to “ParseFailed”.

The second stage checks read return packages if the interpreter state is set accordingly. If multiple return packages exist, the loop interprets them until the next SYNC frame is found.

The last stage discards faulty bytes and packages. If the interpreter state is set to “ParseFailed”, it ignores the RX buffer bytes until the next SYNC frame is found. This typically occurs during system wake-up or if CRC errors occur.

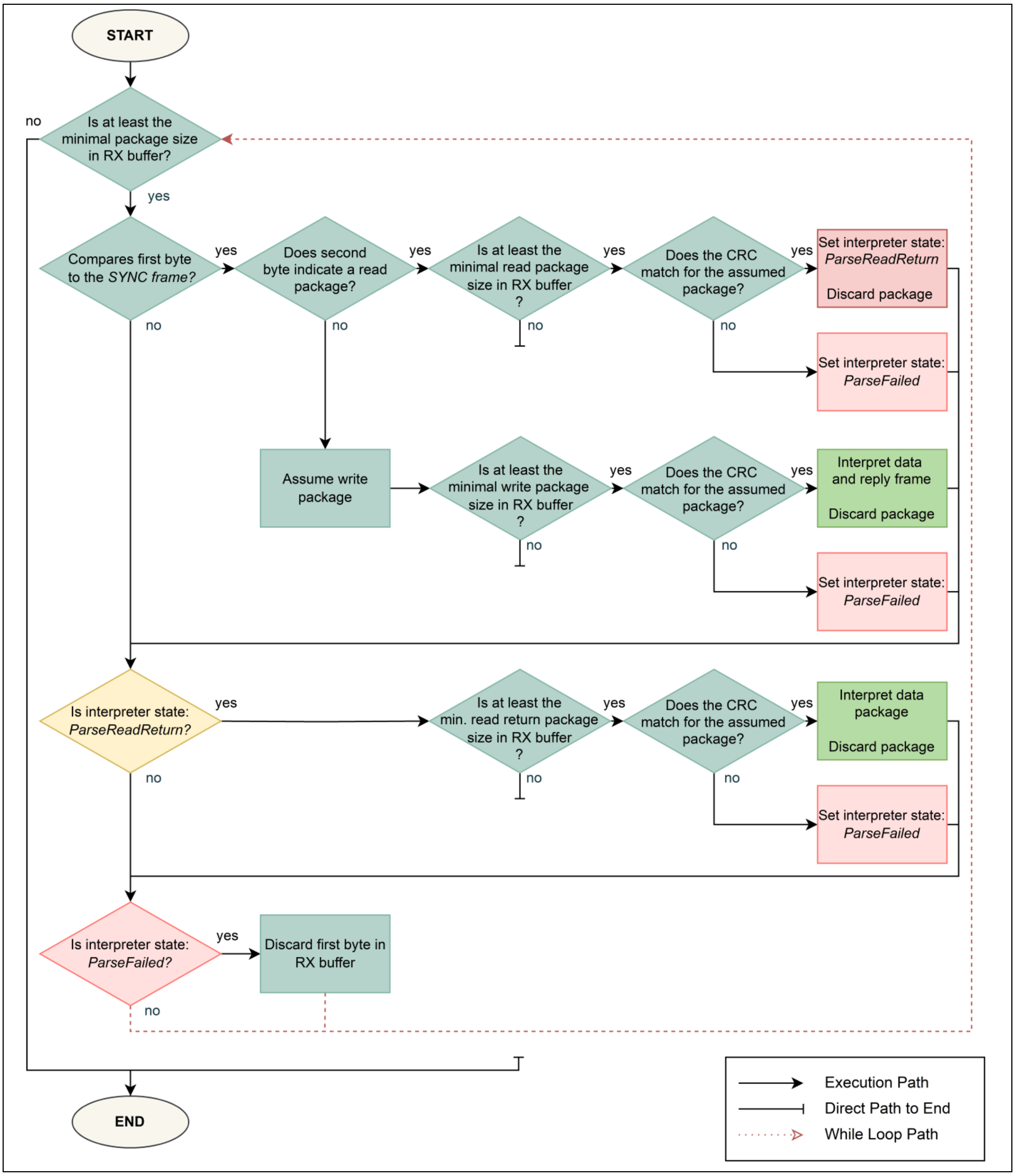

START
Is at least the minimal package size in RX buffer?
no
yes
Compares first byte to the *SYNC frame?*
Does second byte indicate a read package?
Is at least the minimal read package size in RX buffer ?
Does the CRC match for the assumed package?
Set interpreter state: *ParseReadReturn*
Discard package
Set interpreter state: *ParseFailed*
Assume write package
Is at least the minimal write package size in RX buffer ?
Interpret data and reply frame
Discard package
Is interpreter state: *ParseReadReturn?*
Is at least the min. read return package size in RX buffer ?
Interpret data package
Is interpreter state: *ParseFailed?*
Discard first byte in RX buffer
END
Execution Path
Direct Path to End
While Loop Path

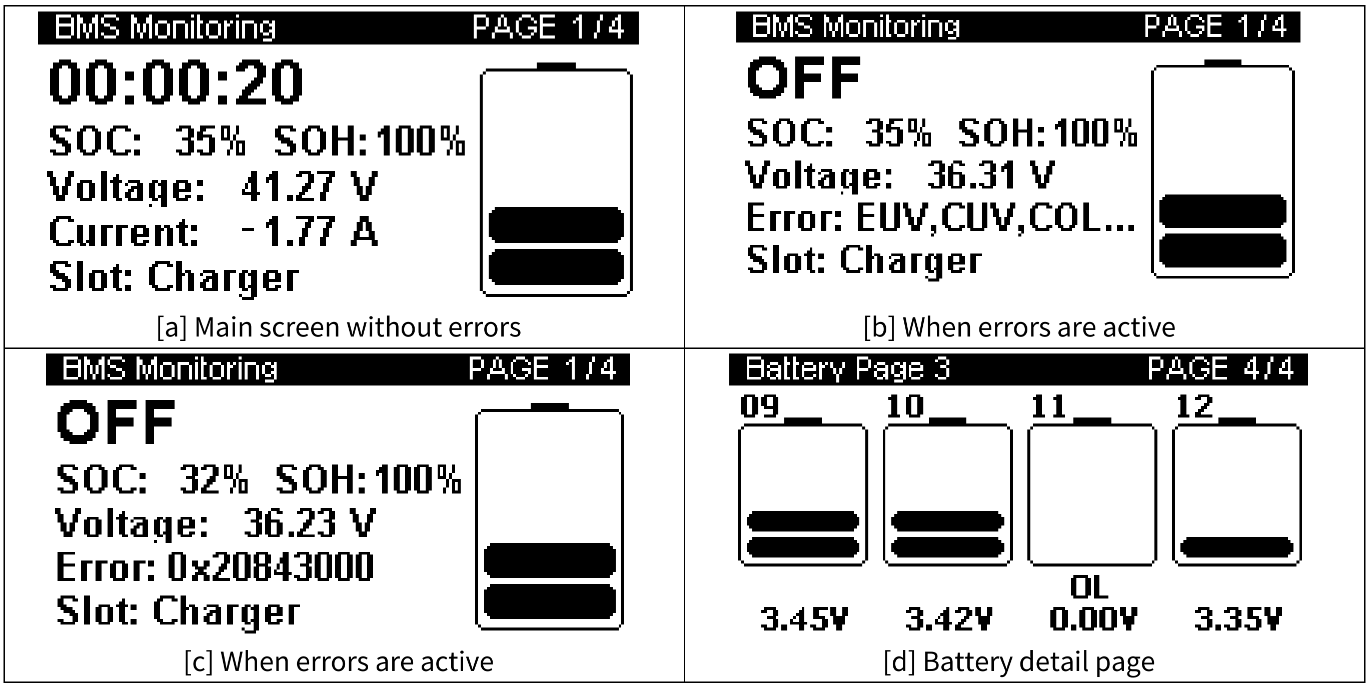
Figure 27 Overview of the communication interpreter system implemented by the library

3.1.12 User interface on secondary core

The logic and flowchart of the secondary CM0+ core is depicted in Figure 29. During the main loop, the CM0+ core continuously monitors conditions that may necessitate a display refresh, while adhering to the minimal refresh interval. Additionally, the on-time and public RAM are refreshed at set intervals.

A signal from the primary CM4 core can trigger a final display update during the shutdown process, after which the CM0+ core will stop. If the display needs to be refreshed due to a significant change in the state, the main menu page is shown (see Figure 28).

Pressing the display button will navigate to the next menu page, such as the battery detail page (see Figure 28 [d]).



```
BMS Monitoring          PAGE 1/4
00:00:20
SOC:  35%  SOH:100%
Voltage:   41.27 V
Current:   -1.77 A
Slot: Charger
```

[a] Main screen without errors

```
BMS Monitoring          PAGE 1/4
OFF
SOC:  35%  SOH:100%
Voltage:   36.31 V
Error: EUV,CUV,COL...
Slot: Charger
```

[b] When errors are active

```
BMS Monitoring          PAGE 1/4
OFF
SOC:  32%  SOH:100%
Voltage:   36.23 V
Error: 0x20843000
Slot: Charger
```

[c] When errors are active

```
Battery Page 3          PAGE 4/4
09      10      11      12

                OL
3.45V   3.42V   0.00V   3.35V
```

[d] Battery detail page

Figure 28 **BMS monitoring**

Note: *Figure 28 [b] shows the detailed error code where all errors can be identified externally (see Table 9 for the list of the errors), while Figure 28 [c] shows the combined error hex code for external undervoltage, cell undervoltage, and cell open loop.*

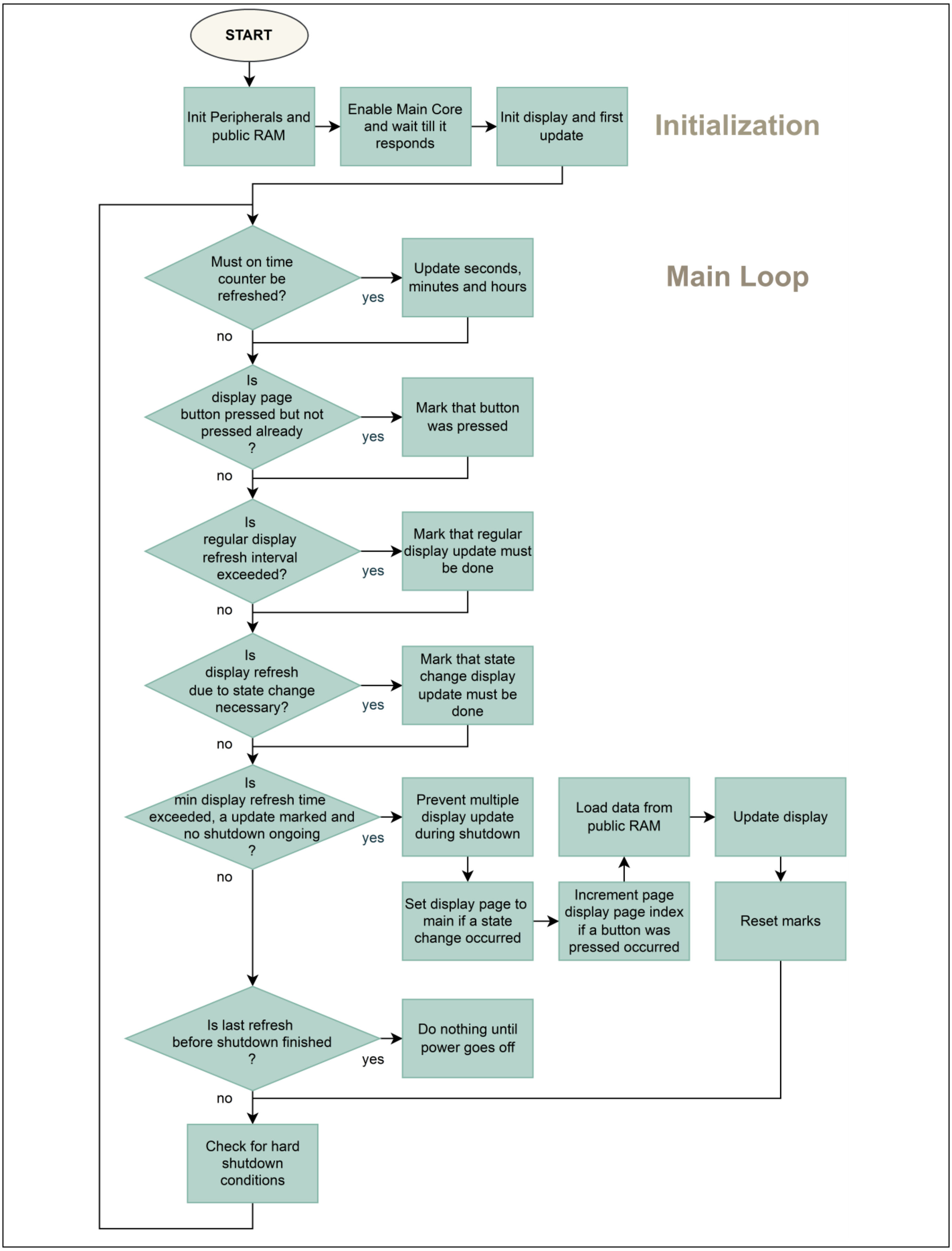

START
Init Peripherals and public RAM
Enable Main Core and wait till it responds
Init display and first update
Initialization
Main Loop
Must on time counter be refreshed?
yes
Update seconds, minutes and hours
no
Is display page button pressed but not pressed already ?
yes
Mark that button was pressed
no
Is regular display refresh interval exceeded?
yes
Mark that regular display update must be done
no
Is display refresh due to state change necessary?
yes
Mark that state change display update must be done
no
Is min display refresh time exceeded, a update marked and no shutdown ongoing ?
yes
Prevent multiple display update during shutdown
Set display page to main if a state change occurred
Increment page display page index if a button was pressed occurred
Load data from public RAM
Update display
Reset marks
no
Is last refresh before shutdown finished ?
yes
Do nothing until power goes off
no
Check for hard shutdown conditions

Figure 29 **Flowchart of the secondary core logic**

3.1.13 CAN communication

The system can be interfaced using CAN messages to exchange status information, requests, and commands. The central function for all outgoing CAN communications is shown in Figure 30.

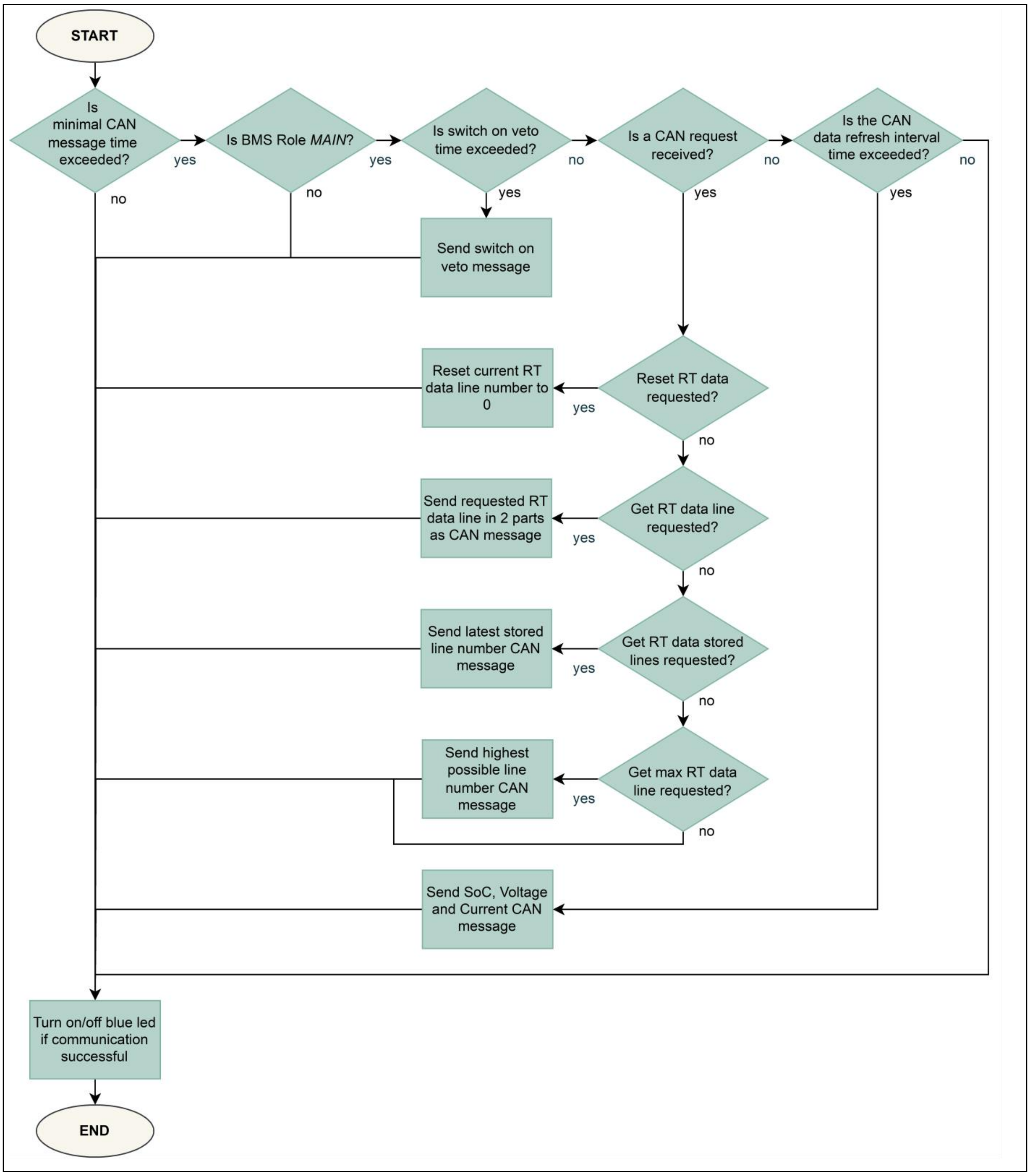

START
Is minimal CAN message time exceeded?
yes
no
Is BMS Role *MAIN*?
yes
no
Is switch on veto time exceeded?
no
yes
Send switch on veto message
Is a CAN request received?
no
yes
Is the CAN data refresh interval time exceeded?
no
yes
Reset RT data requested?
yes
no
Reset current RT data line number to 0
Get RT data line requested?
yes
no
Send requested RT data line in 2 parts as CAN message
Get RT data stored lines requested?
yes
no
Send latest stored line number CAN message
Get max RT data line requested?
yes
no
Send highest possible line number CAN message
Send SoC, Voltage and Current CAN message
Turn on/off blue led if communication successful
END

Figure 30 **Detailed flow chart of the communication**

To prevent the peripheral from blocking messages due to bus errors, re-initialization is triggered by specific errors. All communication is formatted in big-endian, where bytes are sent from the highest value byte to the lowest.

As a prerequisite for any outgoing traffic, a minimum interval between messages must be maintained to ensure that the bus usage remains within safe limits. Status information and requests are processed only by the supplying BMS assigned the “MAIN” BMS role (see Figure 30). Additionally, CAN messages are sent during hot-swap operations (in the roles of “GIVER” or “TAKER”).

If the necessary conditions are met and messages can be sent, three types of messages are processed in order of priority. The highest priority is given to the *SwitchOnVeto* message. The “MAIN” BMS sends this message at short intervals when the output is active to signal other BMS units that they are not allowed to switch on, which is part of the hot-swap initialization. If this message does not need to be sent in a cycle, the system checks the next type. Here, responses to previously registered requests are processed. Finally, if no other message type needs to be processed, system status data is sent.

Requests that result in returning data typically send the information using the same message ID and do not require any parameters. An exception to this rule is the *RT_Data_GetLines* command, which requires specifying the parameters [StartIndex] and [Offset] as 2 bytes each. After performing plausibility checks, the system sends the number of lines specified in the [Offset] starting from the [StartIndex]. Since the data size of default CAN messages is limited, each line is sent in two parts.

For a full list of available CAN command, see Table 12. For every physical signal contained in the data portion, the decoding description is given in italics:

- U / S: Unsigned / Signed
- Factor: Factor with which the value in the CAN data must be multiplied to arrive at the physical value
- Unit: The unit of the signal

For example,

Voltage signal of the BMS_STATE message (Bytes 1 and 2), “*U, 0.001, V*”

This means that the 16-bit value is an unsigned integer, and needs to be multiplied by 0.001 to convert the signal to V. All signals use Big Endianness (“Motorola” convention) for their byte order.

Table 12 BMS CAN messages

		Byte #									
Message	**ID**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Description**	**Rate**
BMS_state_ (slo_ 1|2)	0x100 |0x101	SoC *U, 1, %*	Voltage *U, 0.001, V*		Current *S, 0.001, A*		SoH *U, 1, %*	Uptime *U, 1, s*		BMS state information	3.33 Hz
Reduce_PowEr_ For_HS	0x180									Command from BMS to main controller to reduce power for how-swap (1 s delay before hot-swap initiated)	On demand during hotswap
Enable_Power_ For_HS	0x181									Information from BMS to main controller that hot-swap is finished, and default power can be drawn again	On demand during hotswap
(slot 1|2) HS_ Handshake	0x182 |0x192									Initial handshake command during hot-swap	100 Hz during hotswap
(slot 1|2) HS_ Swap	0x183 |0x193									Supply swap command during hot-swap	100 Hz during hotswap
(slot 1|2) HS_ Finish	0x184 |0x194									Finish command during hot-swap	100 Hz during hotswap

		Byte #									
Message	**ID**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Description**	**Rate**
(slot 1|2) Switch_on_Veto	0x185 |0x195	VETO_state 0xC801 for active; 0x[SoC], 00 for inactive								Message between the two BMS to prevent the other one from switching on.	10 Hz normal operation, 100 Hz during hotswap
RT_Data_ Restart	0x199									Command to the BMS to restart real-time data recording which sets current stored line number to 0	On demand
RT_Data_ Get_Lines	0x19A	Start_index		Offset						Request to the BMS to send out real-time data recording lines.	On demand
RT_Data_ Get_Line_Part	0x19B	Δt *U, 1, ms* or $T_{cell\,avg}$ *S, 100, C*		ΔC *S, 1, mAs*		V_{bat} *S, 1, mV*		$V_{imbalance}$ *S, 1, mV*		Answer to RT_Data_GetLines request. The message is first sent with byte 0, 1 containing Δt, then $T_{cell\,avg}$ is sent in a new message	On demand
RT_Data_ Get_Stored_ Line_Number	0x19C	Number of Lines (Empty for RX)								Request to the BMS to send out how many real-time data lines are currently recorded	On demand
RT_Data_ Get_Max_ Line_Number	0x19D	Max Number of Lines (Empty for RX)								Request to the BMS to send out how many real-time data lines are possibly recorded	On demand
RT_Data_Set_ Sampling_Time	0x19E	Sampling Time *U, 1, ms*								Command to the BMS to set real-time data sampling rate to a specific value in milliseconds. By default, if '0' is sent, the sampling time is changed dynamically. Therefore, the full charge or discharge cycle is automatically recorded	On demand
RT_Data_Get_ Sampling_Time	0x19F	Sampling Time *U, 1, ms*								Reply from BMS to send out the current sampling time used for real-time data recording	On demand

4 Quick start guide

This section helps to get started with using the board as a battery management module for the Infineon Mobile Robot. Later chapters describe how to program and debug the hardware, and set up the hardware for CAN communication. These sections also describe the relevant buttons, LEDs, and interfaces.

4.1 Software deployment and debugging

The software controlling this demo board is developed using ModusToolbox™ Software, a collection of easy-to-use libraries and tools enabling rapid development with Infineon MCUs. Programming and debugging are carried out via the MiniProg4 Program and Debug Kit (CY8CKIT-005). The DEMO_IMR_BMSCTRL_V1 board is connected to the debug probe using the smaller 1.27 mm pitch ribbon cable and the 10-pin box header P2. Larger connector pins are used only if additional UART is necessary. The debug probe can be used as both isolated (default) and non-isolated when using an appropriate software such as CYPRESS™ Programmer.

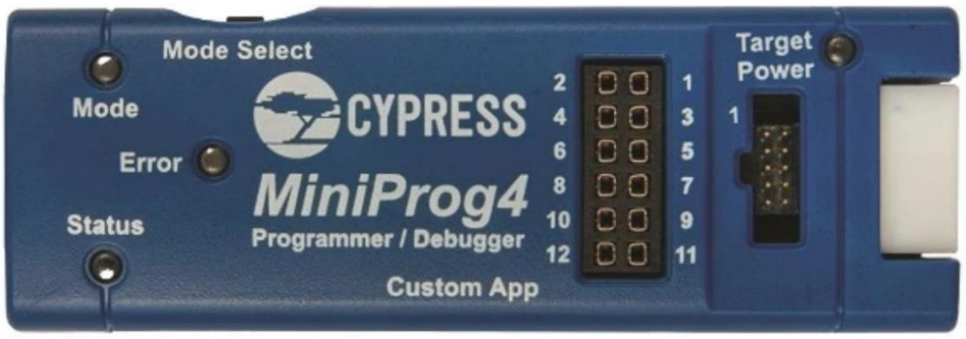


Figure 31 **MiniProg4 Programmer and Debugger Kit (CY8CKIT-005)**

This project is created within ModusToolbox™ IDE containing the device definition, settings, and source files required to compile and build the executable code, which can be downloaded into the Flash program memory of the PSOC™ 6 microcontroller. To debug and apply the software to the board, the newest version of the ModusToolbox™ IDE is necessary and can be downloaded directly from the Infineon website.

1. Go to **File** > **Import** > **General** > **Existing Projects into Workspace** into ModusToolbox™ project, and select the project folder, which is cloned from the Infineon GitHub repository

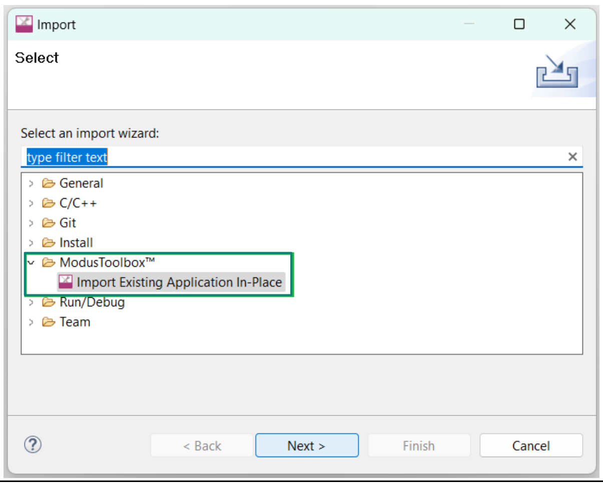


Figure 32 **Eclipse IDE for ModusToolbox™ Import window**

Note: *After importing the project into the ModusToolbox™ IDE, its contents can be found in the "Project Explorer" tab as shown below. Each of the microcontrollers CPU cores owns its own subproject inside the IDE with separate main.c files containing the main body of the source code. Only if these sub-projects (i.e., proj_cm0p and proj_cm4) happen to be not recognized correctly, (see Figure 33 and Figure 34), right click on the projects and select "**Import as Project**".*

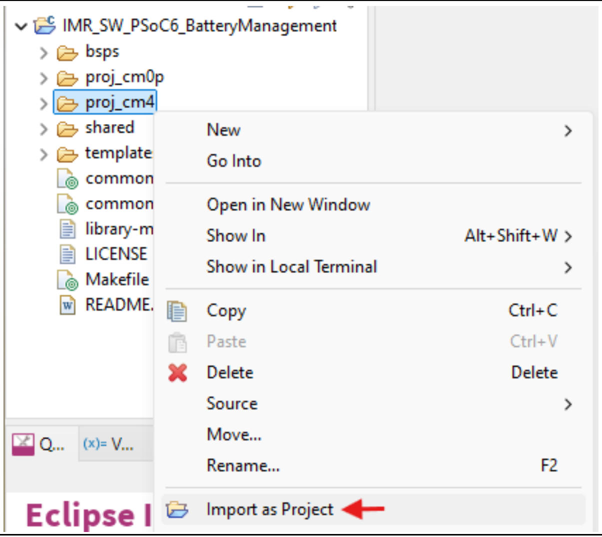
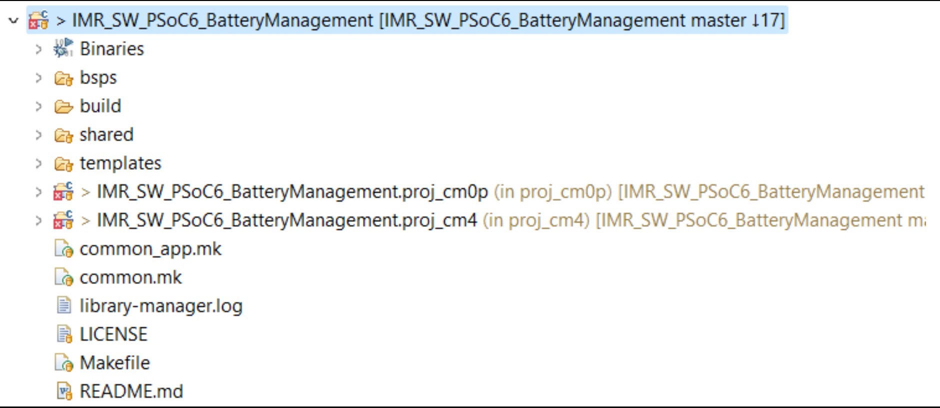
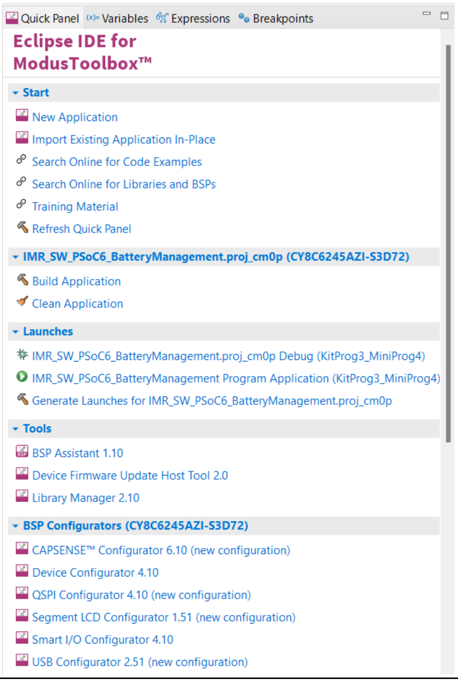


Figure 33 Importing subprojects when the sub-projects are not yet correctly imported (recognized as folders only)



```
> IMR_SW_PSoC6_BatteryManagement [IMR_SW_PSoC6_BatteryManagement master ↓17]
    Binaries
    bsps
    build
    shared
    templates
    > IMR_SW_PSoC6_BatteryManagement.proj_cm0p (in proj_cm0p) [IMR_SW_PSoC6_BatteryManagement
    > IMR_SW_PSoC6_BatteryManagement.proj_cm4 (in proj_cm4) [IMR_SW_PSoC6_BatteryManagement ma
    common_app.mk
    common.mk
    library-manager.log
    LICENSE
    Makefile
    README.md
```

Figure 34 IMR_SW_PSOC™6_BatteryManagement – ModusToolbox™ project structure



Quick Panel | Variables | Expressions | Breakpoints

Eclipse IDE for ModusToolbox™

Start
- New Application
- Import Existing Application In-Place
- Search Online for Code Examples
- Search Online for Libraries and BSPs
- Training Material
- Refresh Quick Panel

IMR_SW_PSoC6_BatteryManagement.proj_cm0p (CY8C6245AZI-S3D72)
- Build Application
- Clean Application

Launches
- IMR_SW_PSoC6_BatteryManagement.proj_cm0p Debug (KitProg3_MiniProg4)
- IMR_SW_PSoC6_BatteryManagement Program Application (KitProg3_MiniProg4)
- Generate Launches for IMR_SW_PSoC6_BatteryManagement.proj_cm0p

Tools
- BSP Assistant 1.10
- Device Firmware Update Host Tool 2.0
- Library Manager 2.10

BSP Configurators (CY8C6245AZI-S3D72)
- CAPSENSE™ Configurator 6.10 (new configuration)
- Device Configurator 4.10
- QSPI Configurator 4.10 (new configuration)
- Segment LCD Configurator 1.51 (new configuration)
- Smart I/O Configurator 4.10
- USB Configurator 2.51 (new configuration)

Figure 35 ModusToolbox™ IDE main commands

2. Ensure that the root of the project is selected. Using the “Quick Panel” in the ModusToolbox™ IDE, you can adjust the projects-used libraries and pin configurations using either “Library Manager” or “Device Configurator”. These menus will guide you through the selection of peripherals and assigning the corresponding pins to these peripherals. After importing, start the “Library Manager” and press the **Update** button. This loads all needed libraries to the shared library folder. If any error occurs during this step, try to execute “Clean Application” in the Quick Panel and try to update the “Library Manager” again
3. The software can be compiled by selecting **Build Application**. To flash the imported software to the board, the “Launches” for “Debug” and “Program” are used. The ModusToolbox™ IDE will then start the flashing

4.2 Debugging using onboard UART communication

Additional debugging can be performed using the designated debugging header (P1) on the board. The header provides both RX and TX lines for standalone UART communication. The header can easily be interfaced using the MiniProg4 already used for flashing the board. The debugger provides all necessary pins (pins 6 and 8) to connect with the debugging header.

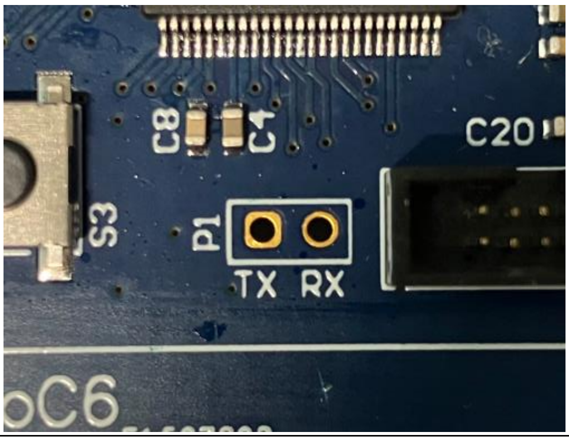


Figure 36 UART debugging connector P1 interfaced using MiniProg4

Use a suitable serial terminal to read the received debugging messages. Several programs are openly available (for example, Putty, hterm) and can be used. Additionally, the ModusToolbox™ IDE also has a built-in serial terminal in **Window > Show View > Other... > Terminal > Terminal**. Here, the COM port of the debugger.

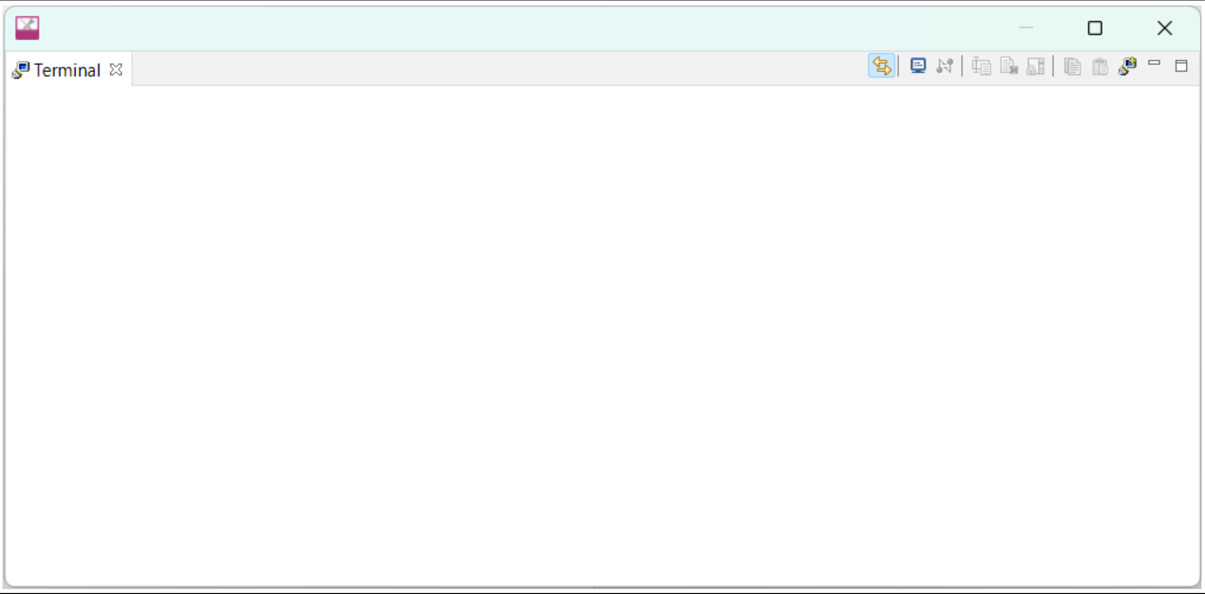


Figure 37 UART debugging connector P1 interfaced using MiniProg4

4.3 Setting the identification

After flashing the board with the software, the board's unique identification can be set using the 8-pin DIP switch S1. See the Hardware section. Note that in the current implementation, the behavior of the board regarding CAN communication is not affected.

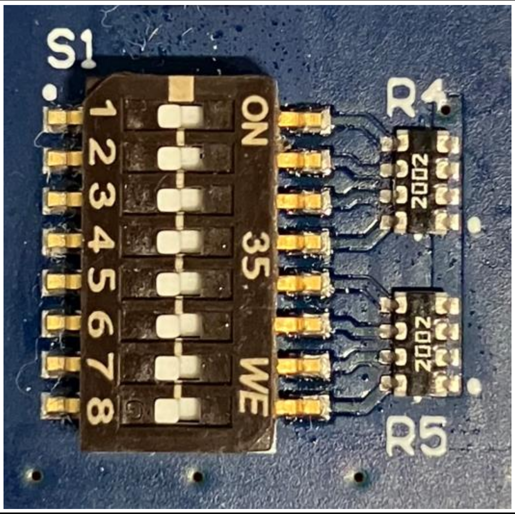


Figure 38 DIP switch to set a unique ID for the board

4.4 BMS control buttons and LEDs

The board provides the following onboard buttons and LEDs (on the top of the system to be easily visible and reachable) to interface BMS functions without using onboard CAN communication.

- **Start:** Long-press the button until the MCU Ready LED glows. This powers the system and enables all functions

Note: *The BMS expects to be the only active participant on the CAN bus during its startup. If this is the case, its first CAN messages are NOT acknowledged. If messages are acknowledged before power is supplied to the rest of the system, the BMS shuts down as a safety measure. Please ensure that any CAN probes or adapters connected to the system for debugging are not acknowledging these messages.*

- **Shutdown:** Press this button to make the system disengage the output power supply and turn off the system completely when ready
- **Display:** Displays the next available page on the ePaper display

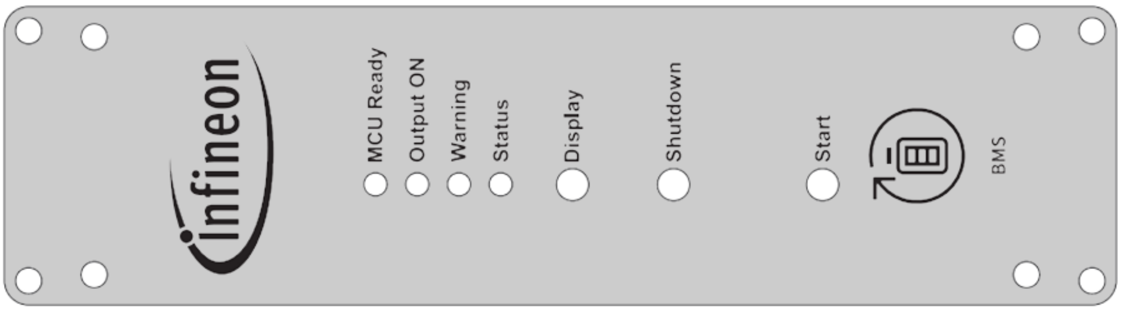


Figure 39 Top view of IMR BMS controls and indicators

The following LEDs are provided to indicate the current system status:

- **MCU Ready:** Indicates whether the system is currently powered. Press the **Start** button to display the current status:
  - *LED ON:* System powered on
  - *LED blinking:* Shutdown sequence active
  - *LED OFF:* System powered off
- **Output ON:** Shows the status of the controlled system output. Indicates whether the system was able to turn on the output successfully:
  - *LED ON:* Output active
  - *LED blinking:* Pre-charge sequence active
  - *LED OFF:* Output not active
- **Warning:** Indicates whether a system error is detected, such as pre-charge failure:
  - *LED ON:* System error detected
  - *LED blinking:* Not applicable
  - *LED OFF:* No system error
- **Status:** Indicates communication with the target system:
  - *LED ON:* Communication active

- *LED blinking:* Not applicable
- *LED OFF:* Communication not active

4.5 Charging and discharging using the BMS interfacing board

This section describes how to charge and discharge the BMS system (BMS power board and BMS controller board). To interface the system, the BMS motherboard (DEMO_IMR_SET_BMS_MB_V1) with the corresponding onboard resistor settings must be set.

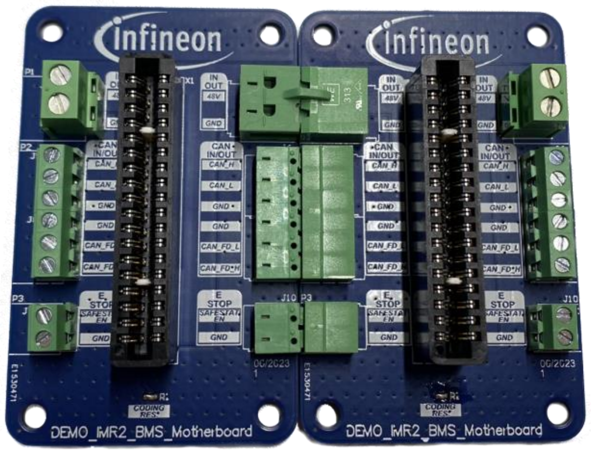


Figure 40 DEMO_IMR_SET_BMS_MB_ V1 boards in IMR consisting of 2 PCBs for Slot 1 and Slot 2

To use the system as a standalone power supply solution for an application, the BMS motherboard must be configured with the resistor R1 = 7.5 kΩ (Slot 1) or 13 kΩ (Slot 2). This allows the system to detect if the inserted slot is designed to charge or discharge the BMS. In the image below, both interface cards can be seen in the IMR in the discharging configuration. After inserting the system into the slot, it can be powered on using the **Start** button; after a successful initialization, the IMR is powered up.

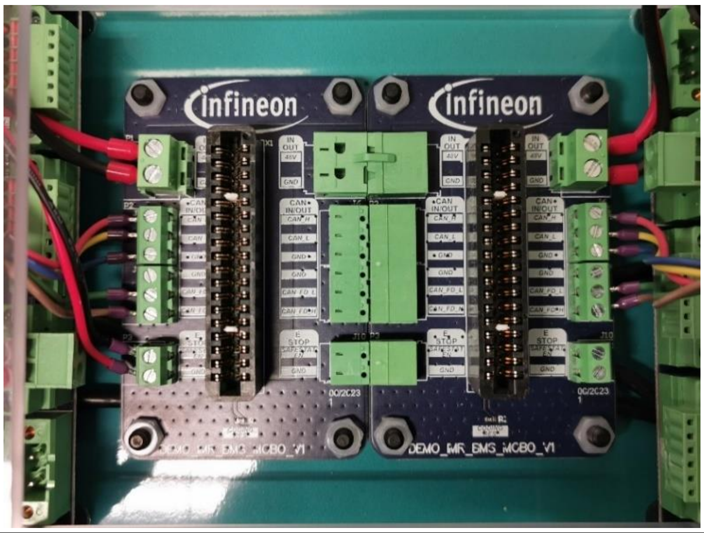


Figure 41 Using the BMS as the IMR power supply (discharging)

To charge the BMS, an external charging unit is to be used in combination with the BMS motherboard (in this case with the resistor R1 = 0 Ω). It must be ensured that the charging voltage and current are set according to the limits specified for Li-Ion batteries used.

Do the following to charge the batteries:

1. Insert the system into the slot
2. Press the **Start** button to power on

Wait for successful initialization. Charging starts as shown in the BMS E-Paper display with the current value being negative, which means the current is flowing into the BMS from the charger.

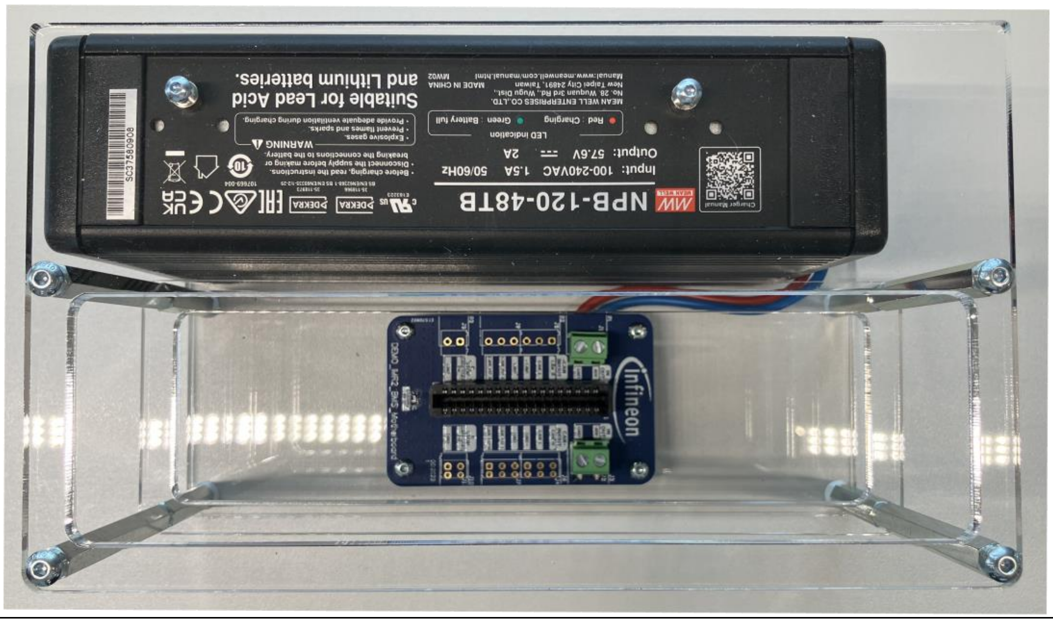


Figure 42 Recharging onboard batteries in the external charging unit (charging)

5 Bill of materials (BOM)

Table 13 Bill of materials

Designator	Manufacturer	Part Number	Quantity	Value/Rating
C1, C13	Würth Elektronik	885012106031	2	10 µF/25 V/0603/20%
C2, C3, C4, C18, C20, C31	Würth Elektronik	885012206076	6	1 µF/25 V/0603/10%
C5, C6, C7, C8, C9, C14, C17, C19, C33, C38	Würth Elektronik	885012206071	10	100 nF/25 V/0603/10%
C10	Würth Elektronik	885012107018	1	4.7 µF/25 V/0805/10%
C11, C12	Würth Elektronik	885012006093	2	8.2 pF/100 V/0603/10%
C15, C16	Würth Elektronik	885012006050	2	6.8 pF/50 V/0603/10%
C21, C22, C27, C28, C29, C30, C32	Würth Elektronik	885012106029	7	2.2 µF/16 V/0603/10%
C23, C24, C25, C26, C34, C35, C36, C37	Würth Elektronik	885012207079	8	2.2 µF/25 V/0805/10%
C39	Würth Elektronik	885012206092	1	33 nF/50 V/0603/10%
D1	Würth Elektronik	150060BS75000	1	Blue/470 nm
D2	Würth Elektronik	150060GS75000	1	Green/525 nm
D3	Würth Elektronik	150060RS75000	1	Red/625 nm
D4	Nexperia	PMEG3020ER	1	Low VF Schottky 30 V/2 A
J1	Würth Elektronik	687140183722	1	WR-FPC SMT ZIF 40p Horizontal low profile
J2	Würth Elektronik	61300711821	DNP	WR-PHD 7p 2.54 mm
L1, L2	Würth Elektronik	742792663	2	1k@100 MHz 100 mA
LS1	Murata	416131160808	1	Piezo Buzzer 4 kHz
P1	Würth Elektronik	61300211121	1	WR-PHD 2p 2.54 mm
P2	Würth Elektronik	62701021621	1	WR-BHD 10p 1.27 mm
Q1	Infineon Technologies	BSD235N	1	OptiMOS™ Dual N-Channel small-signal MOSFET
Q2	Infineon Technologies	IRLML6401	1	HEXFET™ P-Channel power MOSFET
R1, R2, R3	Vishay	CRCW06031K00FK	3	1k/100 mW/0603/1%
R4, R5	Bourns	CAT16A-2002F4LF	2	20k/63 mW/1206/1%
R6, R7, R14, R15	Vishay	CRCW060310K0FKEA	4	10k/100 mW/0603/1%
R8	Vishay	CRCW06032K10FK	1	2k1/100 mW/0603/1%
R9, R10	Yageo	RC0603FR-074K7L	2	4k7/100 mW/0603/1%
R11	Vishay	CRCW0603150RFK	1	150 R/100 mW/0603/1%
R12, R13	Vishay	CRCW060310R0FKEA	2	10 R/100 mW/0603/1%
S1	Würth Elektronik	416131160808	1	WS-DISV 8p DIP Switch

Designator	Manufacturer	Part Number	Quantity	Value/Rating
S2	Würth Elektronik	434121025816	1	WS-TASV Tact Switch
S3	Würth Elektronik	430481031816	1	WS-TASV SMT Switch
U1	Infineon Technologies	CY8C6245AZI-S3D72	1	32-bit PSOC™ 6 Arm® Cortex®-M4/M0+
U2	Infineon Technologies	CY15B256Q-SXA	1	256-Kbit (32K × 8) Automotive Serial F-RAM
U3	Texas Instruments	SN74LVC2G14DCKR	1	Dual Schmitt-Trigger Inverter
X1, X2, X3, X4	Würth Elektronik	61302021121	4	WR-PHD Dual Row 20p 2.54 mm
Y1	Würth Elektronik	830108207309	1	24 MHz/8 pF/10 ppm
Y2	Würth Elektronik	830105946101	1	32.768 kHz/7 pF/20 ppm

6 PCB layout

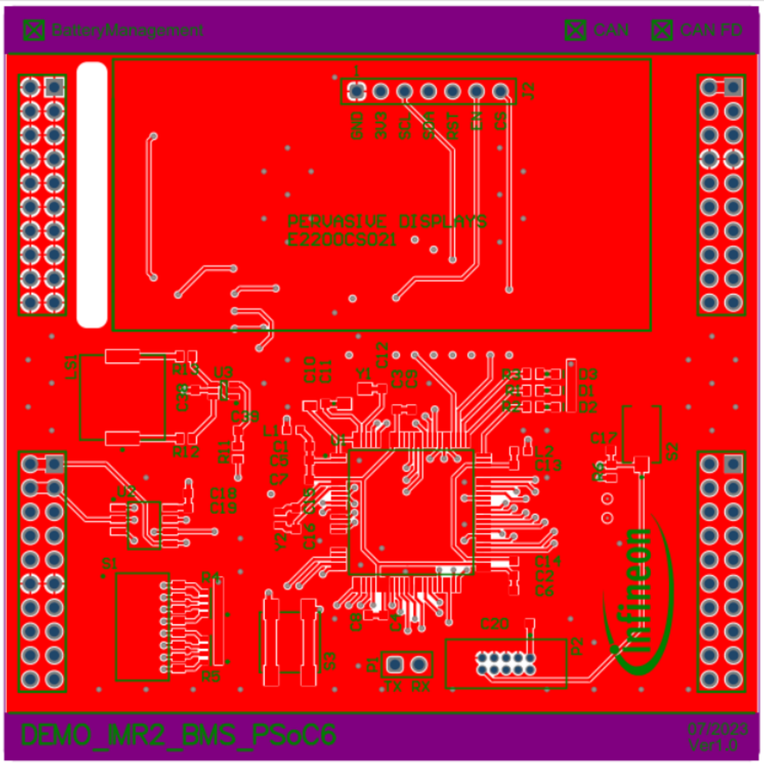


Figure 43 DEMO_IMR_BMSCTRL_V1 board top view

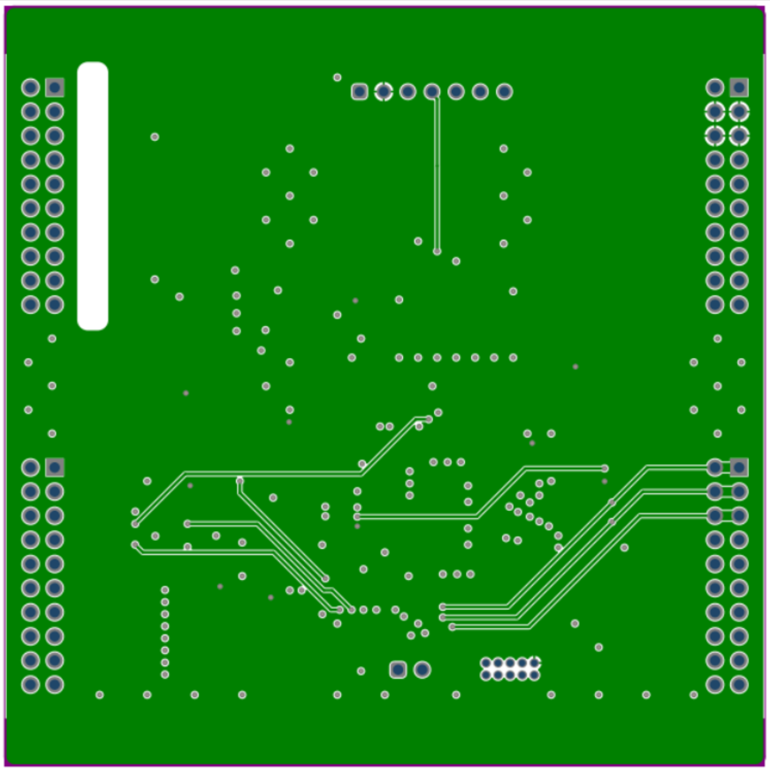


Figure 44 DEMO_IMR_BMSCTRL_V1 board first inner layer

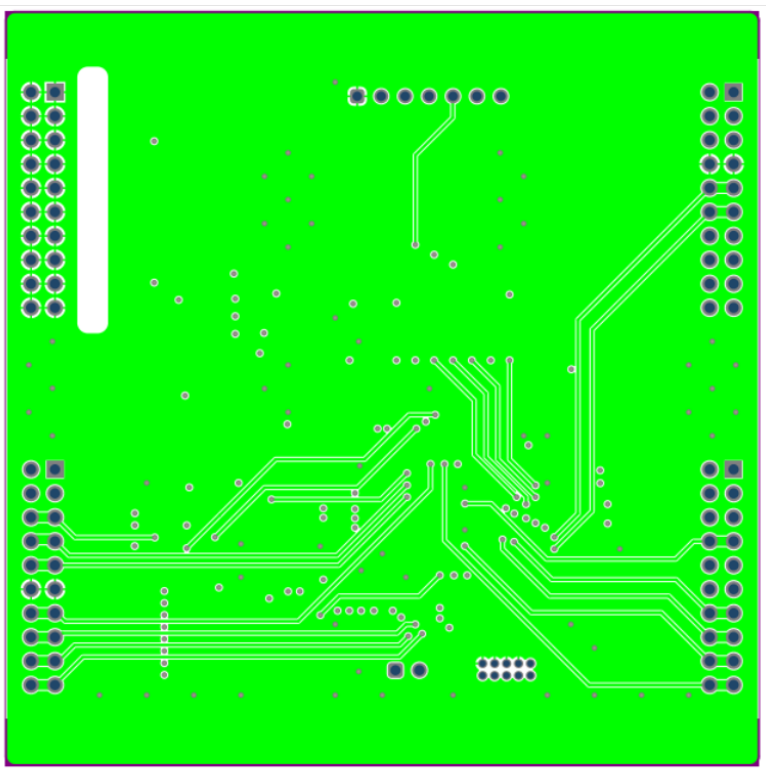


Figure 45 DEMO_IMR_BMSCTRL_V1 board second inner layer

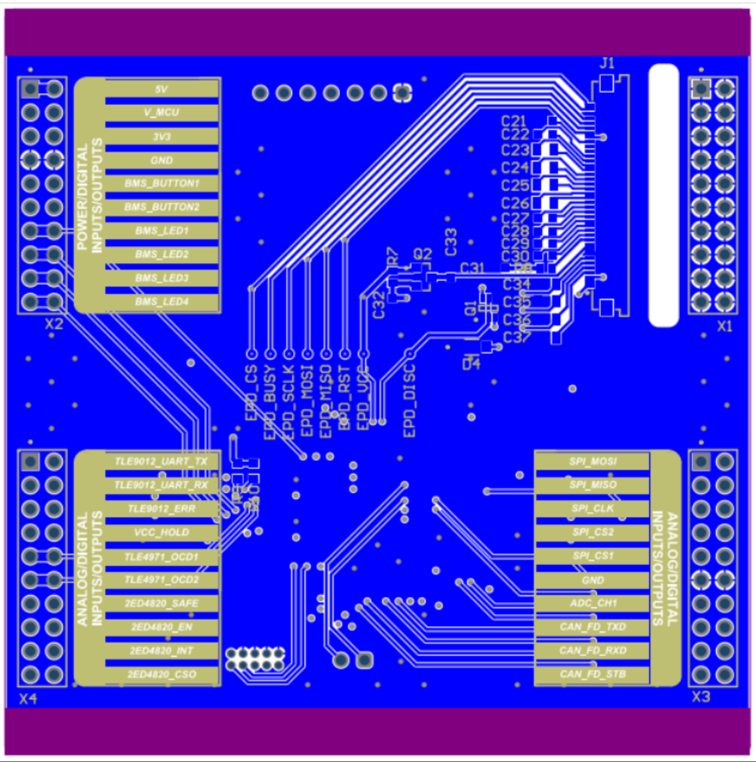


Figure 46 DEMO_IMR_BMSCTRL_V1 board bottom side

7 Appendix

7.1 Schematics

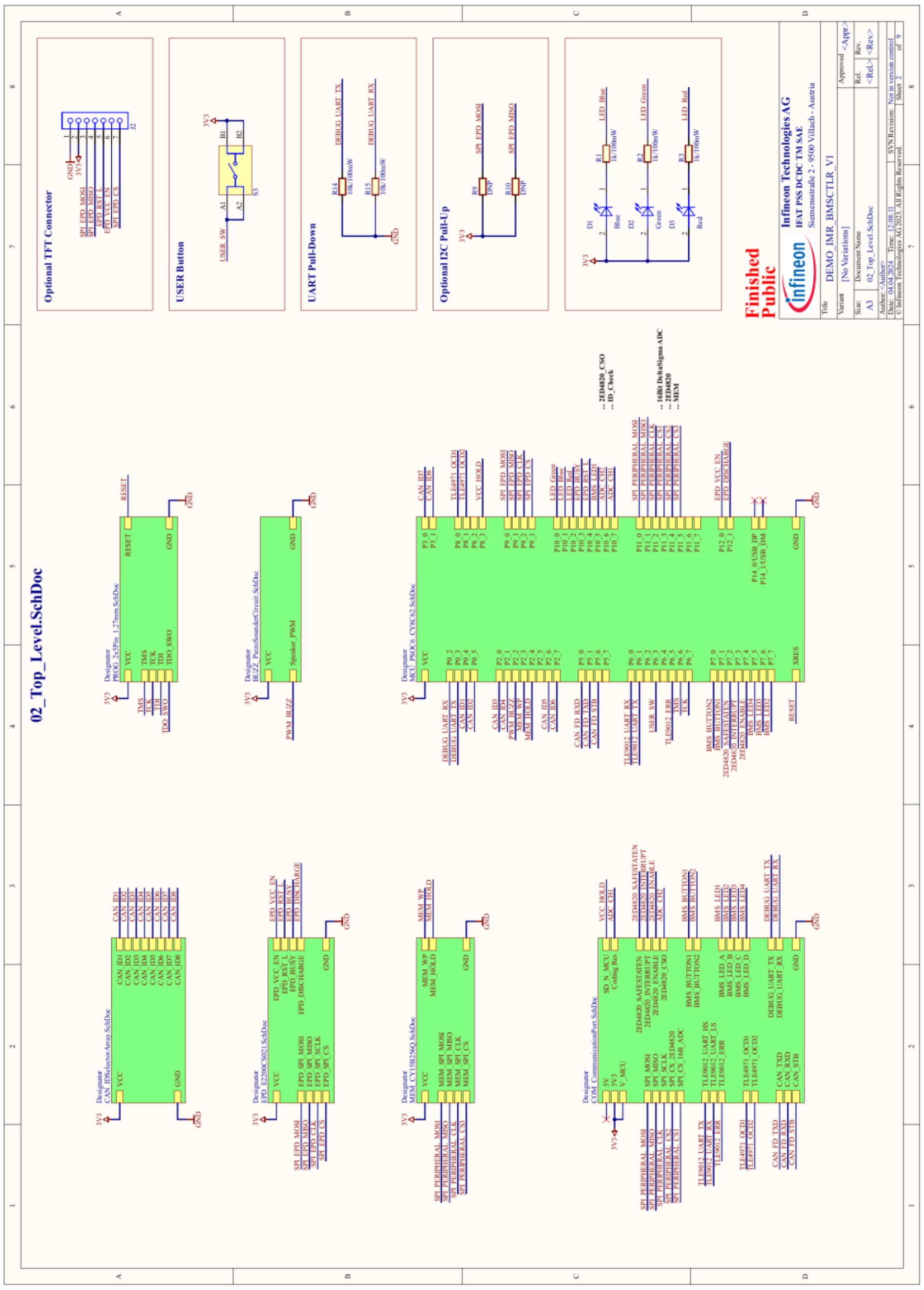


Figure 47 DEMO_IMR_BMSCTRL_V1 – 02_Top_Level.SchDoc

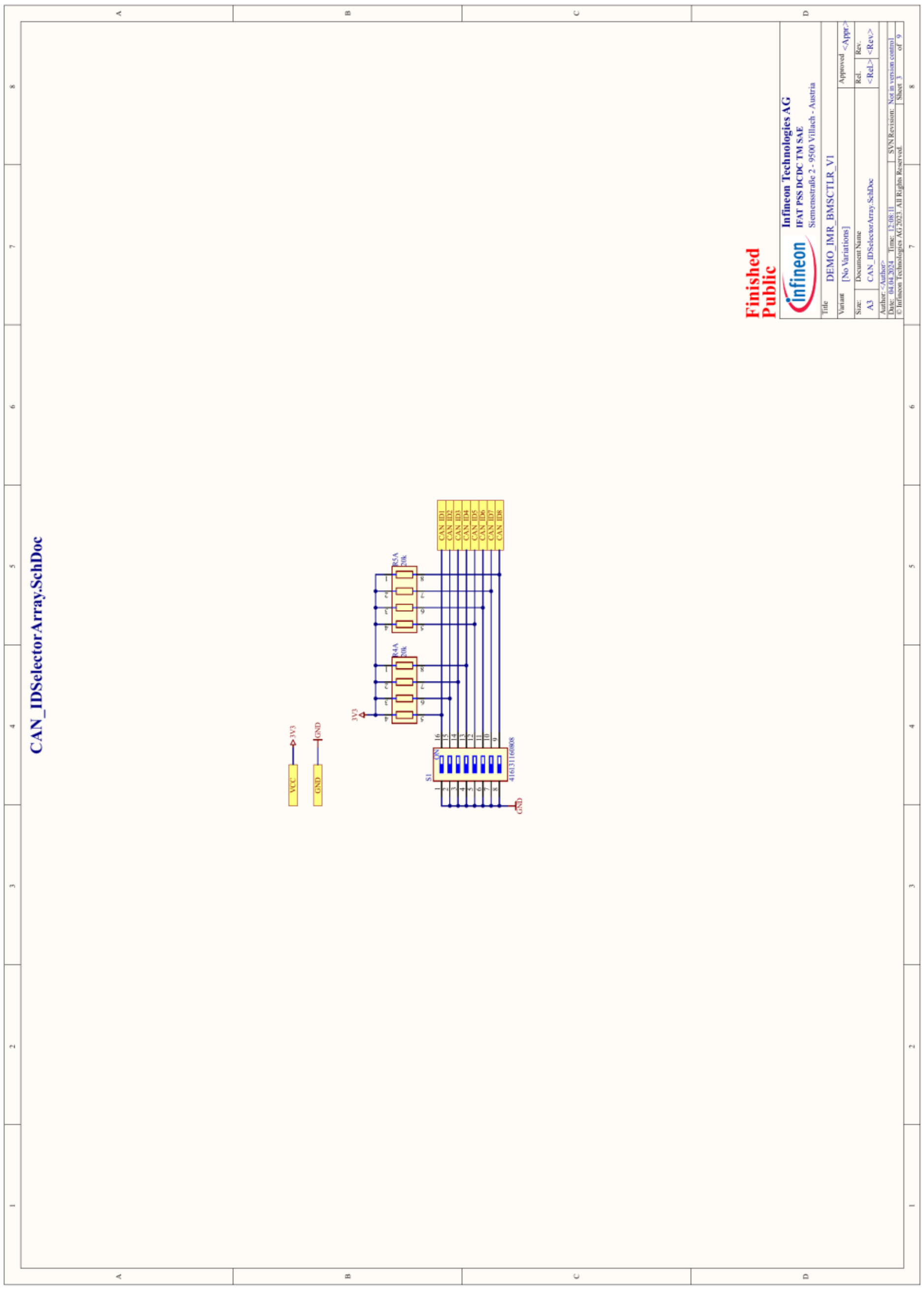


Figure 48 DEMO_IMR_BMSCTRL_V1 – CAN_IDSelectorArray.SchDoc

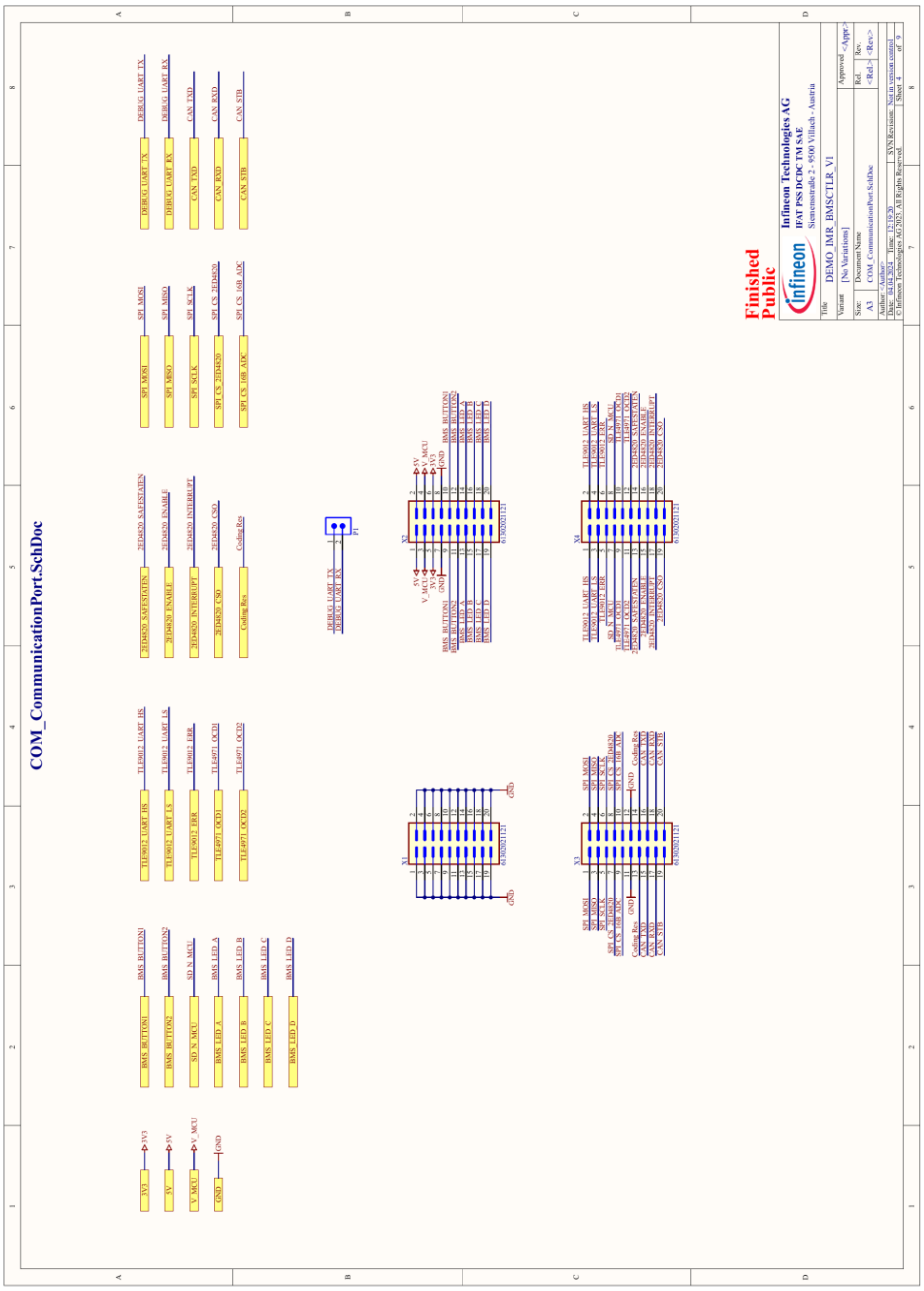


Figure 49 DEMO_IMR_BMSCTRL_V1 – COM_CommunicationPort.SchDoc

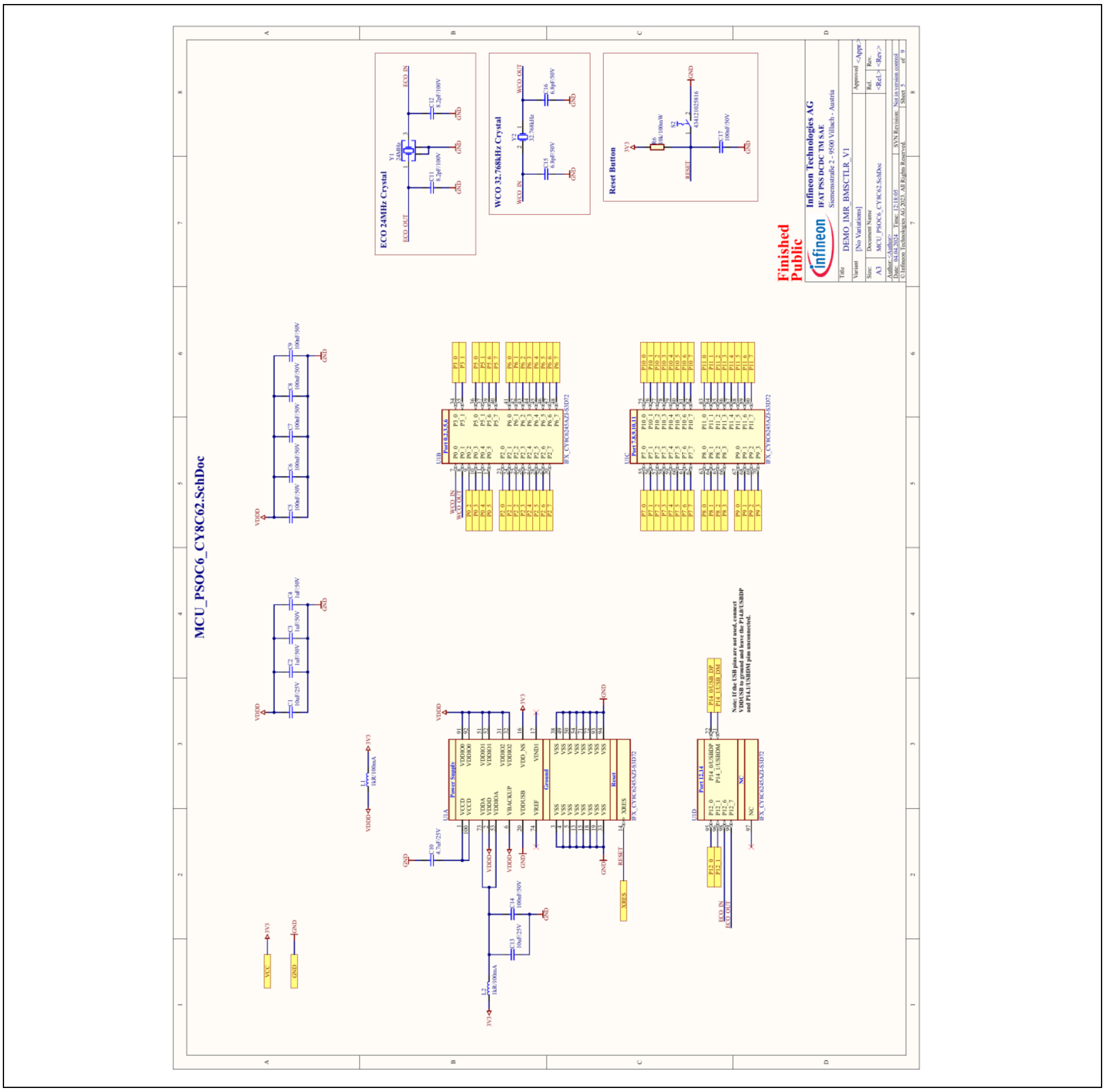


Figure 50 DEMO_IMR_BMSCTRL_V1 – MCU_PSOC6_CY8C62.SchDoc

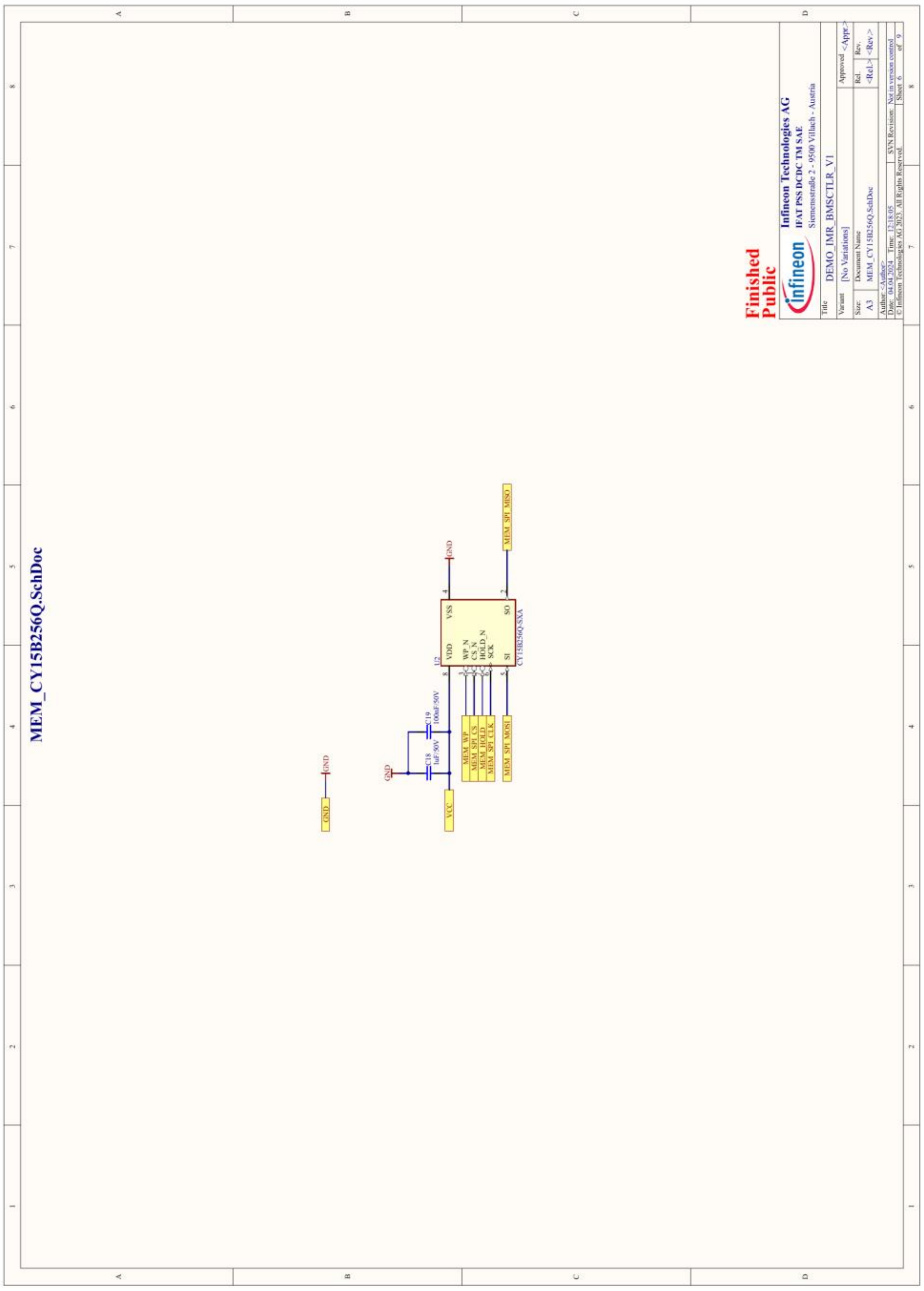


Figure 51 DEMO_IMR_BMSCTRL_V1 – MEM_CY15B256Q.SchDoc

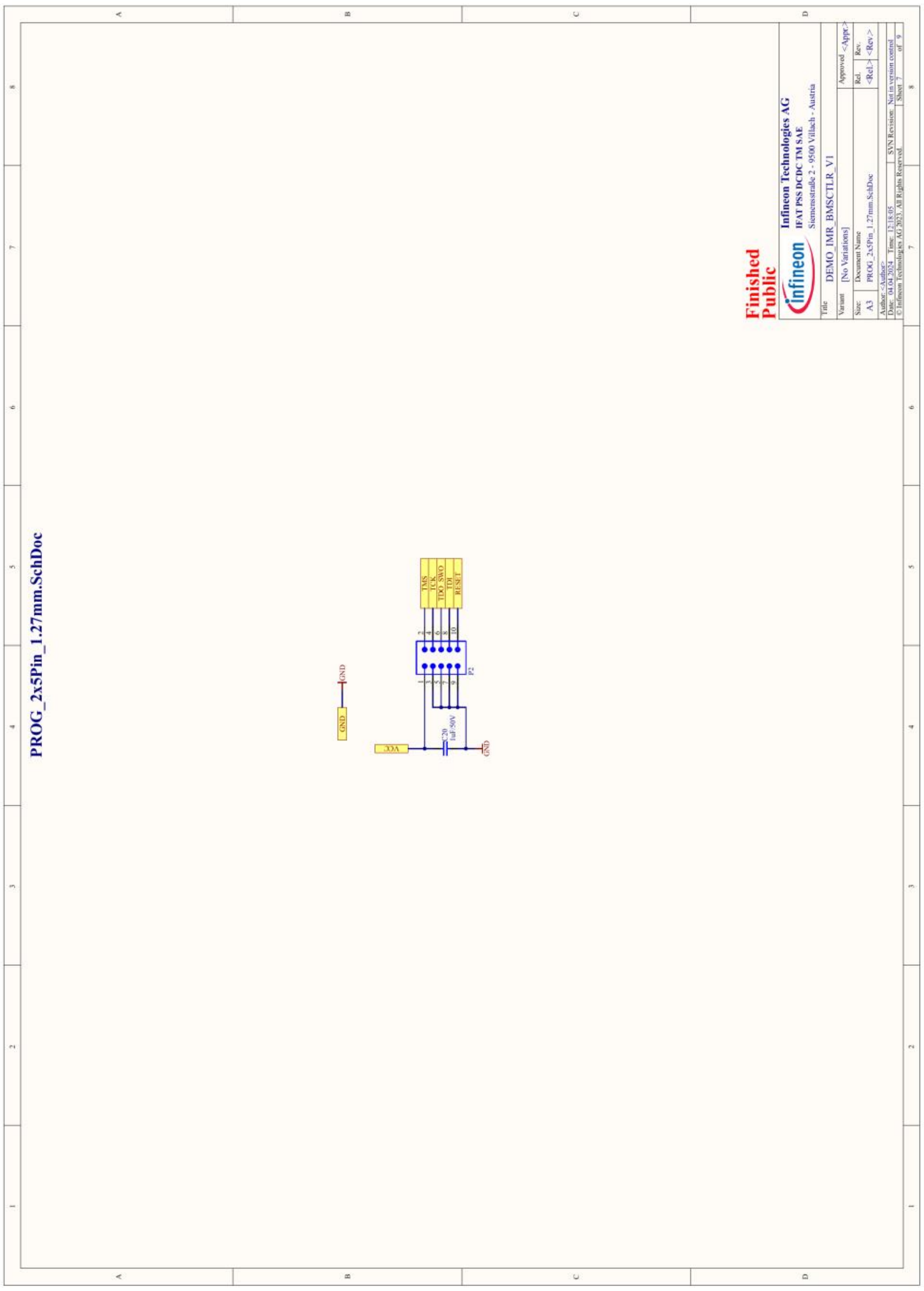


Figure 52 DEMO_IMR_BMSCTRL_V1 – PROG_2x5Pin_1.27mm.SchDoc

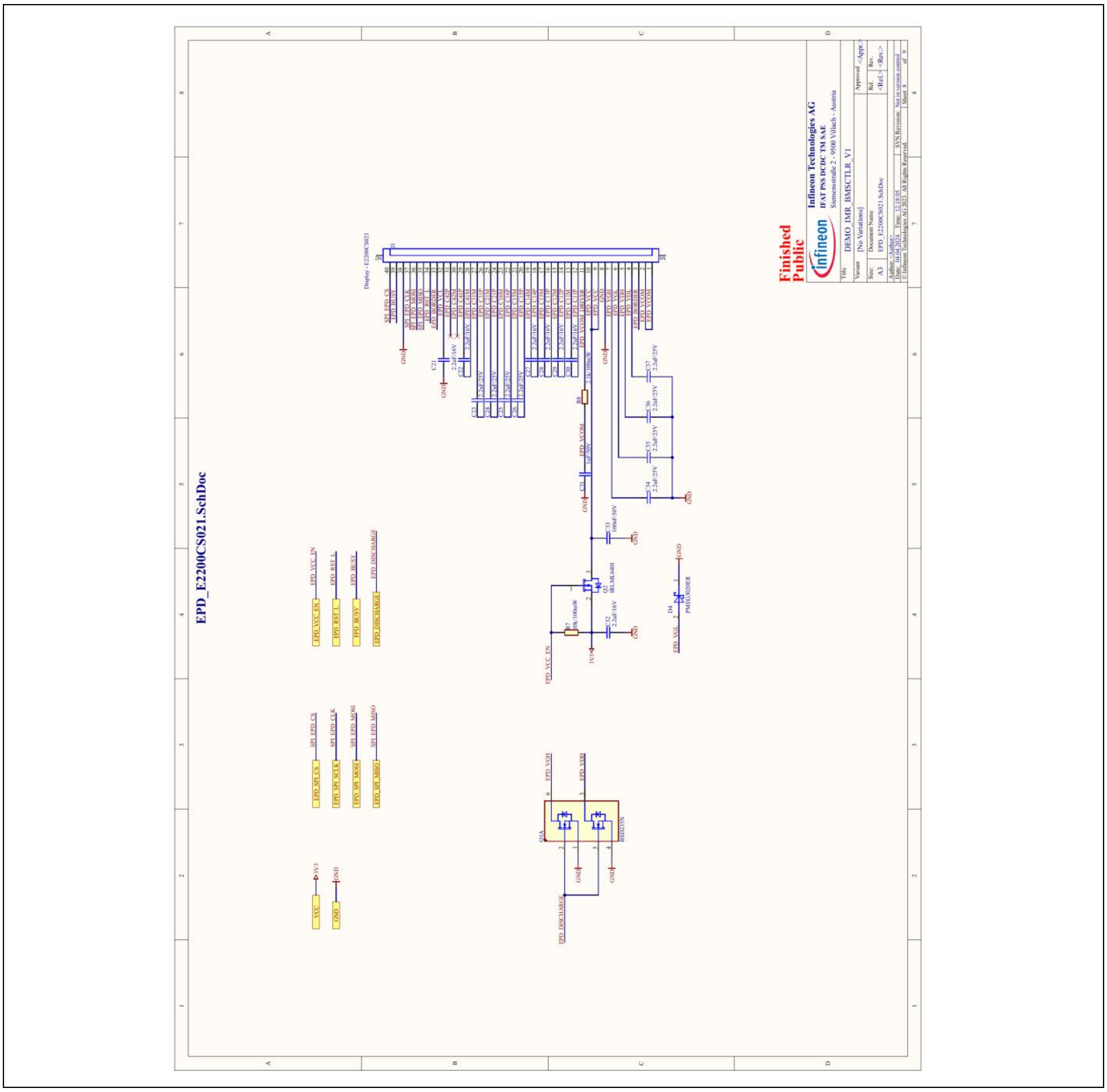


Figure 53 DEMO_IMR_BMSCTRL_V1 – EPD_E2200CS021.SchDoc

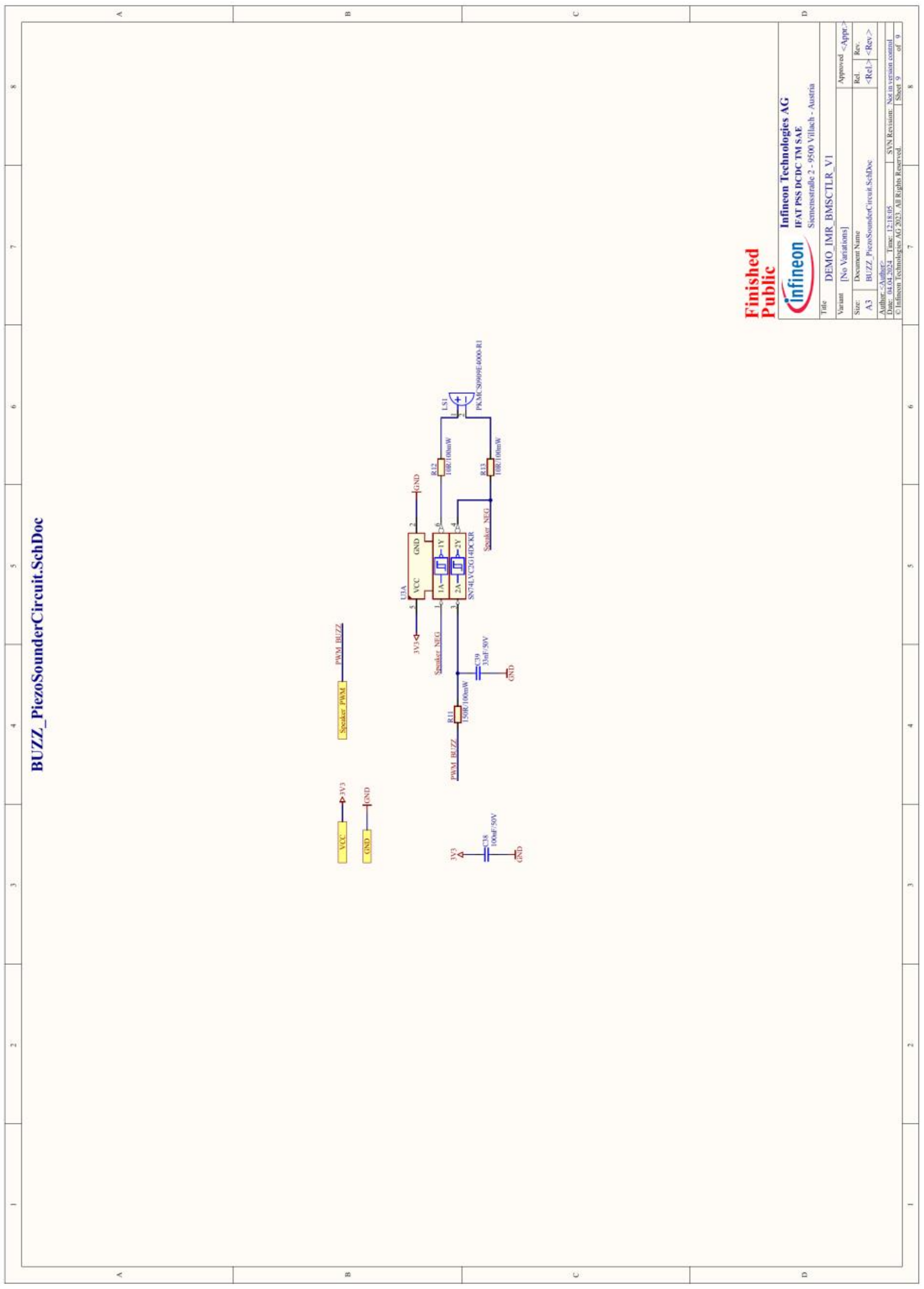


Figure 54 DEMO_IMR_BMSCTRL_V1 – BUZZ_PiezoSounderCircuit.SchDocz

References

[1] Infineon Technologies AG: *PSOC™ 6 MCU: CY8C62x5 datasheet Rev. *M (2022-10);* Available online

[2] Infineon Technologies AG: *32-bit PSOC™ 6 Arm® Cortex®-M4 / M0+ Overview*; Available online

[3] Canis Automotive Labs: *The canframe.py tool, 25 October 2020*; Available online

[4] Infineon Technologies AG: *Infineon Mobile Robot (IMR) webpage*; Available online

[5] Infineon Technologies AG: *DEMO_IMR_BMSCTRL_V1 webpage*; Available online

[6] Infineon Technologies AG: *TLE9012DQU webpage*; Available online

Glossary

ADC
analog-to-digital converter

AGV
automated guided vehicles

AMR
autonomous mobile robots

BMS
Battery Management System

BOM
bill of materials

CAN

Controller Area Network Flexible

CAN FD

Controller Area Network Flexible Data Rate

CE
Conformité Européenne

COM
communication

CRC
cyclic redundancy check

CRC
cyclic redundancy check

CS
Chip Select

DLC
Data Length Code

EMC
electromagnetic compatibility

ESD
electrostatic discharge

GND
ground

IC
integrated controller

IDE
integrated development environment

IMR
Infineon Mobile Robot

OCV
Open Circuit Voltage

PCB
printed circuit board

PSA
Platform Security Architecture

PMA
returned material analysis

RR
Round Robin

RST
reset

RT
real time

SoC
State of Charge

SoH
State of Health

Revision history

Document revision	Date	Description of changes
1.0	2024-04-08	Initial release
1.1	2024-12-04	Update document with software details
1.2	2025-01-23	Updated CAN communication
2.0	2025-09-12	Updated SW repository, SW import method, and resistance values of BMS interfacing boards

Edition 2025-09-12

Published by
Infineon Technologies AG
85579 Neubiberg, Germany

Document reference
UG091835