

ISL94212 Sample Code

This manual provides a detailed description and application guidelines for using the ISL94212 sample code. It includes sample library functions and examples to speed up the design of high voltage battery management systems, consisting of multiple (stacked) battery front end ICs. The ISL94212 software package provides robust and easy access to the resources and functionality of the device. The sample code includes a specialized-to-the-device library, a demo battery management application and a user interface; these components are designed to be portable and suitable for integration into multitasking software projects.

Features

The software package has the following features:

- Stand-alone or daisy chain operation
- Full system scalability
- Custom configurations
- Easy status and error monitoring
- Integrated fault diagnostics and processing
- Application Programming Interface (API) for easy integration
- Fully compatible with Renesas Advanced (RA) Family 32bit MCUs

Target Device

The library is intended to be used with ISL94212 in stand-alone or stacked configuration. The target device is the ISL94212 Li-ion battery manager IC that supervises up to 12 series connected cells and can operate in standalone mode or with up to 14 devices in a stack, monitoring up to 168 Li-ion cells in total. The daisy chain hardware provides robust, redundant board-to-board communications using differential, AC-coupled signaling. The MCU communicates with the stack master device using a high-speed SPI communication interface.

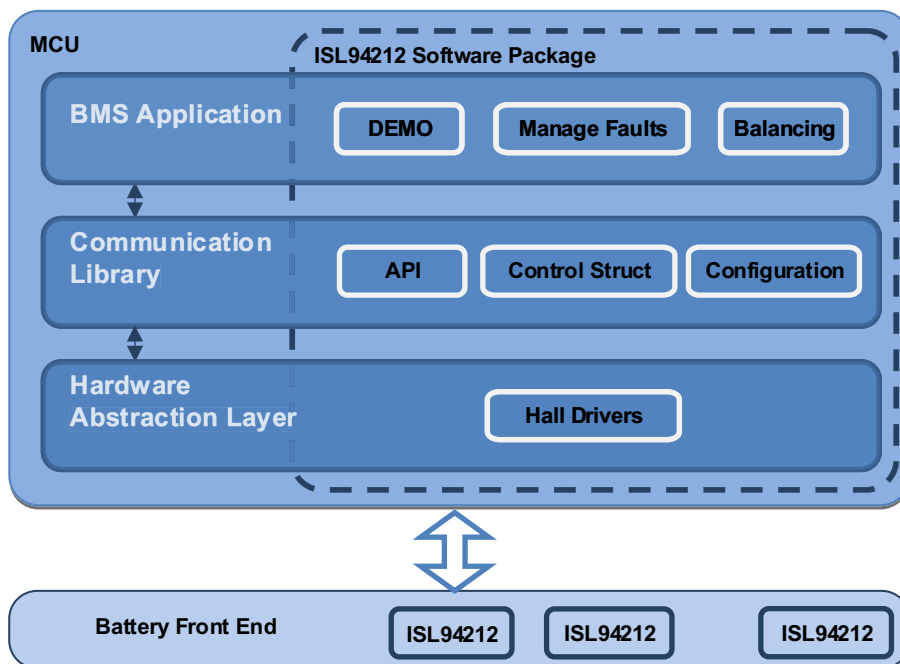


Figure 1. Software Package Structure

Contents

1. Battery Front End Library	3
1.1 Configuration of the BFE Library	3
1.2 Configuration and Control Structures	4
1.3 Description of Functions	5
1.4 Using of the Software Library	5
2. Battery Management System Demo	6
2.1 Initialization	7
2.2 Obtaining Measurement	8
2.3 Cell Balancing	9
2.4 Sleep Mode	10
2.5 Fault Management	10
3. Requirements and Setup	11
3.1 Software Setup	11
3.2 Importing the Sample Project	12
3.3 Hardware Setup	14
3.4 Battery Front End Evaluation Board	14
3.5 MCU Evaluation Board	15
3.6 Power Supply	15
3.7 Precision multimeter	15
3.8 Oscilloscope	15
3.9 Hardware Assembly	16
4. User Interface	17
4.1 Terminal setup	17
4.2 Turning On the Setup	18
4.3 Using the Demo	19
4.4 Debugging	21
5. Glossary	22
6. Ordering Information	22
7. Revision History	22

1. Battery Front End Library

The Battery Front End (BFE) library is a major part of the ISL94212 software package, providing an application programming interface (API) to the user and a level of abstraction for the battery management system (BMS) from the BFE specifics (such as communication sequence, timing, register access, verification, and state monitoring). It comprises the middleware routines, standing in the layer between the HAL drivers and higher-level application algorithms (Figure 1), control structures and definitions. All files, routines, global variables and macros have the prefix `bfe_` or `BFE_` so that they are easy distinguishable from other parts of a multitasking project.

The library file tree is shown in Figure 2. The files have the following content:

- **File `r_bfe_api.h`:** Instances of functions, providing a level of independence for the battery management system and the used battery front end; Configuration and control structures; Battery management system error codes, used for diagnostics of state of functions; General enumerations, used for control of the library functions.
- **File `r_bfe_isl94212.h`:** Declarations of functions; ISL94212 specific structures, enumerations and macros; Table with all registers and their specifics.
- **File `r_bfe_isl94212.c`:** Implementations of API functions; Initialization of time intervals, holding structures.
- **File `r_bfe_crc4.h`:** Declarations of functions used for the non-standard CRC4.
- **File `r_bfe_crc4.c`:** Functions used for the non-standard CRC4 calculation, check and appending to data arrays.
- **File `r_bfe_cfg.h`:** Library configuration, used by the pre-processor, related to hardware specifics.
- **File: `r_bfe_common.h`:** Common macros, used in the other library source code and header files.

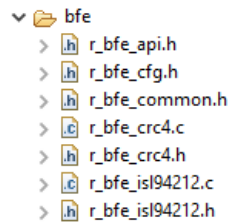


Figure 2. BFE Library File Structure

1.1 Configuration of the BFE Library

It is important that the battery front end library is correctly configured to function properly. This process takes place in the file `r_bfe_cfg.h` that holds the pre-processor macros and is dictated mostly by the hardware setup and wanted features, providing the following options:

- Stack size: How many devices are in the stack?
- Battery front end devices enable pin usage: Is the master device enable pin connected to the MCU?
- The battery front end goes to sleep mode using a Sleep command or watchdog timeout.
- Use parameter checking for input arguments of functions: Every implementation of API function has an optional build-in input parameter checking mechanism.
- Verify content after register write command: Every function that writes into any register of ISL94212 has an optional build-in register content verification mechanism.
- Check if scan command has been received by all devices: Every function that sends a scan command has an optional build-in scan command acceptance checking mechanism.
- Number of identify attempts: How many times to try to identify devices inside the stack? This option is valid only in a daisy chain!
- The physical pin of the MCU to which the fault pin is connected.
- The physical pin of the MCU to which the data ready pin is connected.
- The physical pin of the MCU to which the enable pin is connected.

- The overvoltage lockout that triggers fault condition. (Default value is 0x1CCC equal to 4.5V)
- The undervoltage lockout that triggers fault condition. (Default value is 0x0CCC equal to 2.0V)
- The external temperature lockout voltage that triggers fault condition. (Default value is 0x3FFF equal to 2.5V)
Depends on the thermistor type and connection.

1.2 Configuration and Control Structures

The configuration structure is used when calling the function for opening the BFE interface. It is initialized in an external source file (see **bal_data.c**) and should not be updated anymore. It holds the following constants:

- Operation mode: The battery front end is in stack (daisy chain) or stand-alone mode.
- Daisy chain data speed: This constant is set according to the input levels of pins COMMS RATE 0 and COMMS RATE 1.

The control structure is used as an input argument in every function. It is initialized in an external source file and most of its content can be updated any time by application or by the relevant function itself. The relevant variables must be modified before calling a specific function. For more details, check the description of functions. It holds the following constants and nested structures:

- Cell balancing pattern: Cell balancing pattern used for applying masks while running the algorithm. This is an input argument.
- Select cells for balancing: The array size is as the number of devices in the stack. Every bit is relevant to a cell balancing FET control output. This array is an input argument.
- Watchdog timeout: Set timeout or disable watchdog. This is an input argument.
- Status structure: This structure is modified by certain functions.
 - Serial numbers: The array holds the serial numbers of all devices in the stack. This is an output argument.
 - Sleep mode: The variable denotes if the battery front end is in sleep mode. This is an output argument.
 - Cell balancing status: The variable denotes if battery balancing has been initiated. This is an output argument.
 - Internal temperature limits: The array holds the internal temperature limits of all devices in the stack. This is an output argument.
 - Trim voltage: The array holds the nominal cell voltages of all devices in the stack. This is an output argument.
 - Stack identification counter: Stack identification cycles that has passed. This is an output argument.
 - Stacked devices: Number of detected devices in the stack. This is an output argument.
- Setup structure: This structure must be initialized and further modification of internal variables is not recommended.
 - Overvoltage limit: The variable holds the overvoltage limit to which each cell voltage is compared after scan. This is an input argument. (0 = 0V; 8191 = 5V)
 - Undervoltage limit: The variable holds the undervoltage limit to which each cell voltage is compared after scan. This is an input argument. (0 = 0V; 8191 = 5V)
 - External temperature limit: The variable holds the current voltage limit to which each external temperature input voltage is compared after scan. This is an input argument. (0 = 0V; 16383 = 2.5V)
 - Wire scan current: The variable holds the selected option for an open-wire scan current (from enumeration). This is an input argument.
 - Fault samples totalizator: The variable holds the number of consecutive compare events that trigger fault detection and protection. This is an input argument.
 - Balance mode: The variable holds the cell balancing mode enumeration. This is an input argument.
 - Configuration of cells: The array holds the configuration of cells of all devices in the stack. Bit [0] – Bit [11] are relevant to a cell voltage input that is measured and fault tested. This is an input argument. When using less than 12 cells per device, distribute cells according to description in datasheet and set the respective bits.

- Configuration of temperatures: The array holds the configuration of external temperature inputs of all devices in the stack. Bit [0] to Bit [3] is relevant to an external temperature voltage input that is measured. This is an input argument.
- Configuration of temperatures faults: The array holds the fault configuration of external temperature inputs of all devices in the stack. Bit [0] to Bit [3] is relevant to an external temperature voltage input that is fault monitored. This is an input argument.

1.3 Description of Functions

The API functions documentation can be found in an interactive catalogue, which is a part of the ISL94212 software package (see [isl94212_sw_package_doc/index.html](#)). This HTML-based document contains a detailed description of all API functions, input/output parameters, error returns, data structures, enumerations, and definitions (Figure 3). A common input parameter is the BFE control structure. Functions use specific variables from this structure but could also write into them as an output. More information about what you should set and what output to expect can be found into the description of each function.

Renesas does not recommend calling functions in any interrupt. If so, special precautions should be taken to avoid interrupt masking and getting stuck in a loop. The functions use two interrupts. The first one is triggered by any SPI event (receive, transmit buffer empty, transfer complete or error interrupt) and must have higher priority than the second one or the others in nested interrupt. A dedicated hardware timer is used for detection of communication timeouts. Its overflow also triggers an interrupt. There are parallel software counters that return an error to prevent getting stuck in a loop.

Two pins of the master stack device are connected to inputs of the MCU: FAULT and DATA READY. They are monitored at specific moments during the execution of code and do not trigger interrupts that could compromise the commands, therefore, compromising the SPI and stack (daisy chain) communication.

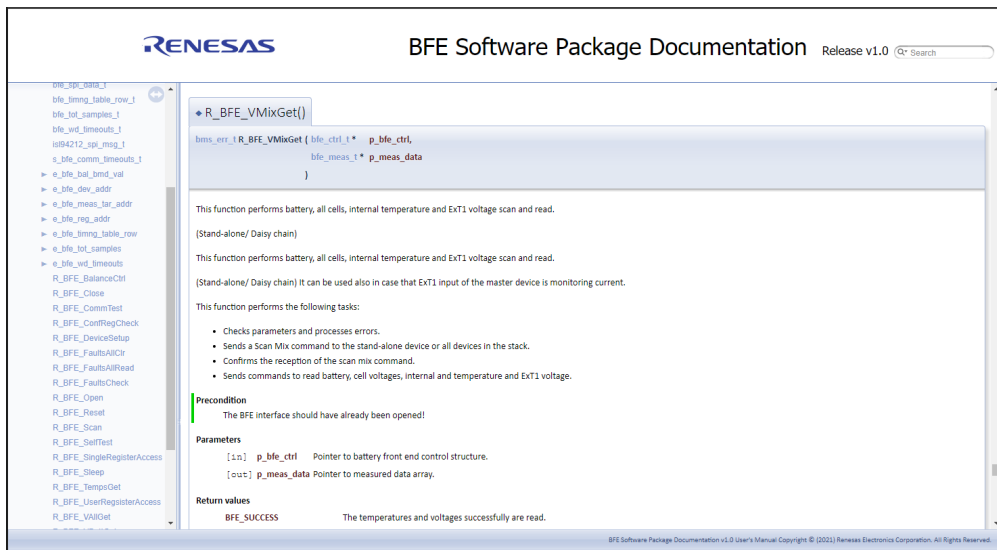


Figure 3. Software Package Reference Documentation

1.4 Using of the Software Library

The software package and the BFE library is designed to be directly used with RA2 series MCUs and is fully compatible with the Renesas Advanced (RA) family. However, with the current MCU, the maximum supported daisy chain data speed is 250kHz as the timing when receiving bytes is critical. Higher speeds can cause a master stack device buffer overflow and communication mishmash. If higher daisy chain data speed is required, a higher performance MCU (such as RA6 series) should be used.

In some cases, when receiving data from the stack, the software can wait for a response in a loop for as long as 46.4ms, depending on the data speed and stack size (see the *Maximum Time to Communications Failure*

Response table from ISL94212 Datasheet). If a watchdog timer is used in the MCU, you should take measures to avoid system reset in this case.

If another MCU is used, it must have the required peripheral modules preconfigured according to the requirements in the BFE initialization function and the connected API (see **ra_gen/hal_data.h**). Because the library is placed in the middleware layer, it has a higher level of abstraction than specific hardware drivers and is tolerant of changing hardware if it provides the required performance.

2. Battery Management System Demo

The battery management system demo is a part of the ISL94212 software package. Its main purpose is to demonstrate the application and functionality of the BFE library but also to provide the customer with examples for basic fault management solutions, described in the product datasheet. The supported toolchain for the sample project is: e² studio Integrated Development Environment (IDE), default toolchain is GCC Arm Embedded. The library file tree is given in [Figure 4](#). The files have the following content:

- **File: bal_data.c:** Declaration and initialization of global configuration and control structures. Connection of the API.
- **File: bal_data.h:** Definitions of the external global variables.
- **File: r_bms.c:** Battery management demo state machine, fault management, balancing algorithm and USB communication related functions.
- **File: r_bms.h:** Definitions of structures, enumerations, configuration macros and other. The over-temperature, under-temperature, and external temperature inputs voltage lockout limits, memory check interval, and the balancing algorithm related parameters must be set into this file.

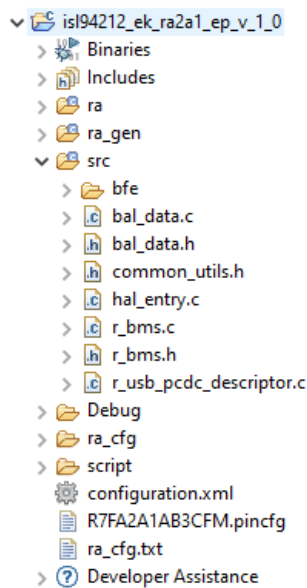


Figure 4. Demo Project File Structure

The states of the sample application and possible transitions between them are given in Figure 5. The execution of the code begins with Initialization state. If initialization is successful, it transitions to Normal state. From Normal state a transition to Balance or Sleep state can be made. A transition to Fault state is accomplished from any other state if an error is returned by any function. The Fault state exits only to Normal state. When a transition from state to state must be made, the relevant flag into a global structure with Boolean type variables is raised and the state is changed in a single place inside the main loop so that no unexpected transition or Fault state skipping transitions are possible.

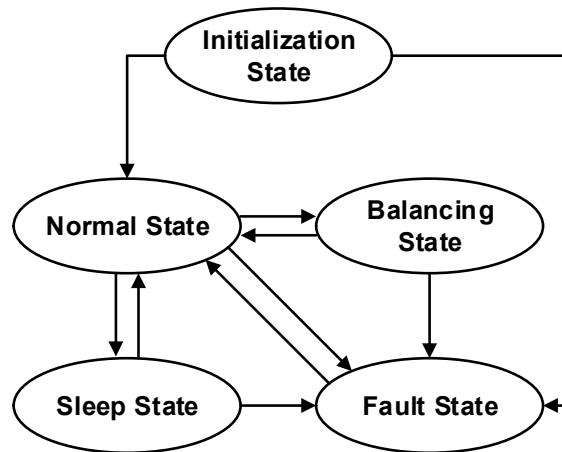


Figure 5. State Machine

2.1 Initialization

The Initialization state is the default one on MCU start-up. The code in it is executed once, followed by a transition to Normal state. After executing every function, if an error return is detected then Fault state is entered. The sequence is given in Figure 6. The open BFE interface function accomplishes SPI interface initialization, stack reset and identification and reads serial numbers of all devices. The reset command ensures that the current condition of every ISL94212 IC is known if the MCU is reset but the BFE was not. The communications test is for demonstration purposes. It could be implemented also in the other states.

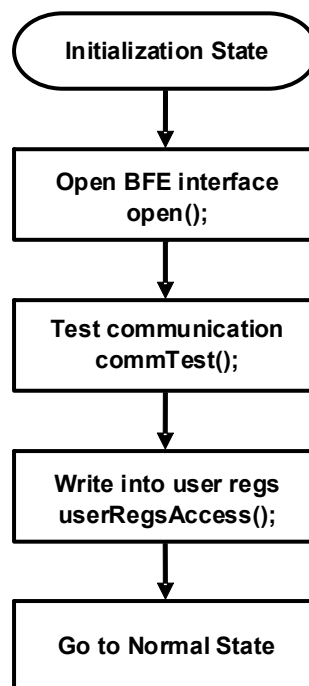


Figure 6. Initialization State Flowchart

2.2 Obtaining Measurement

The Normal state flowchart is given in [Figure 7](#). In the demo project, the voltage and temperature measurements are initiated by you instead of having a continuous action. Therefore, the MCU stays in a loop and waits for your input. After a selected period (10 cycles equal to 1 second by default), determined by a software timer, a memory check function is called that checks the Page 2 registers checksum for corruption of the configuration registers. Periodically, calling the memory check function that sends commands also resets the watchdog timeout of the BFE. At the same interval, the BFE is monitored for indications of any fault conditions by checking the master FAULT pin for assertion and reading the Fault Status Registers of all devices.

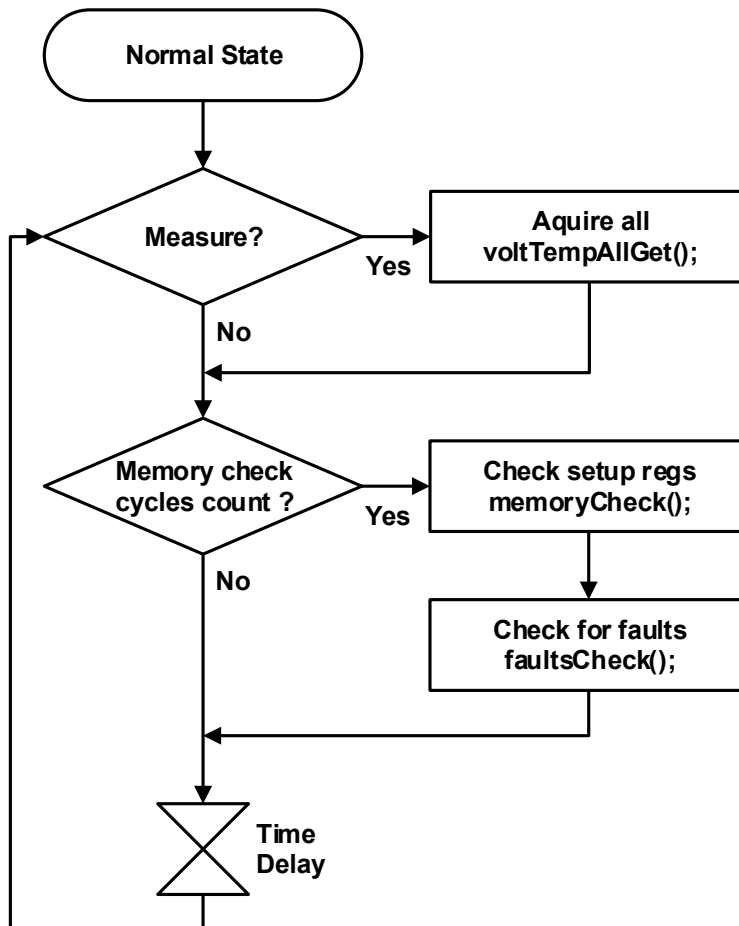


Figure 7. Normal State Flowchart

2.3 Cell Balancing

The BFE library sample project includes a sample algorithm for cell balancing (CB). The algorithm flowchart is given in Figure 8. It contains a loop, at the beginning all cell voltages are acquired and the cell with lowest voltage is found. Next, the voltage difference between each cell and the one with lowest voltage is calculated and compared with predefined thresholds. By default, if the voltage difference is more than 20mV the cell is identified as needing balancing. However, if the voltage difference is more than 500mV a problem with the battery is considered and an error is returned.

If any cell is detected to need balancing, the algorithm continues. First, the odd cells are unmasked and the respective balancing FETs are enabled for a predefined period of time (20s by default). Next, the same action is accomplished for the even cells. Afterwards, all CB FETs are off and an off timer is activated to provide a recovery time for the cell voltages to obtain more accurate measurements before the next balancing cycle (avoiding the RC time constant of the filters and Li- chemistries). If the differences in cells voltages are within limits, the balancing activity is inhibited after the voltage measurement in the next cycle and a transition to Normal state is made. The application, Li-chemistry, and battery pack specifics must be considered when the balancing parameters are set. **Important:** Avoid setting the CB delta voltage thresholds too low and balancing intervals that are too long as this can lead to non-convergence and excess battery drain.

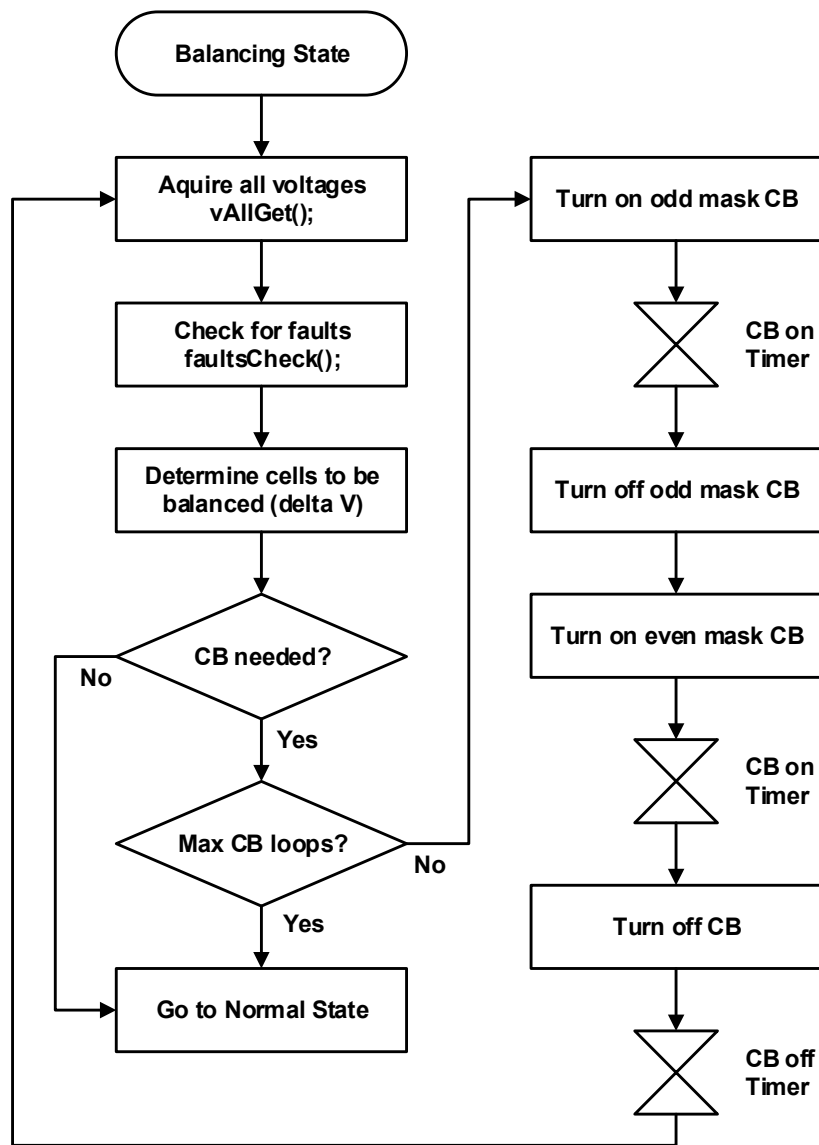


Figure 8. Cell Balancing Algorithm Flowchart

To help avoid this situation, there is a balancing cycles counter that limits the number of loops. To avoid long software delays and taking unnecessary MCU resources, the balancing algorithm is implemented with a dedicated state machine allowing the MCU to run other code in the main loop.

2.4 Sleep Mode

The sample project provides an option for entering sleep mode and waking up all devices in the stack. When Sleep state is active, the BFE enters sleep mode immediately and waits for user input to wake up and make a transition to Normal state (Figure 9). If any of the operations is unsuccessful, the system goes to Fault state. However, when in sleep state the master stack device fault pin is not monitored.

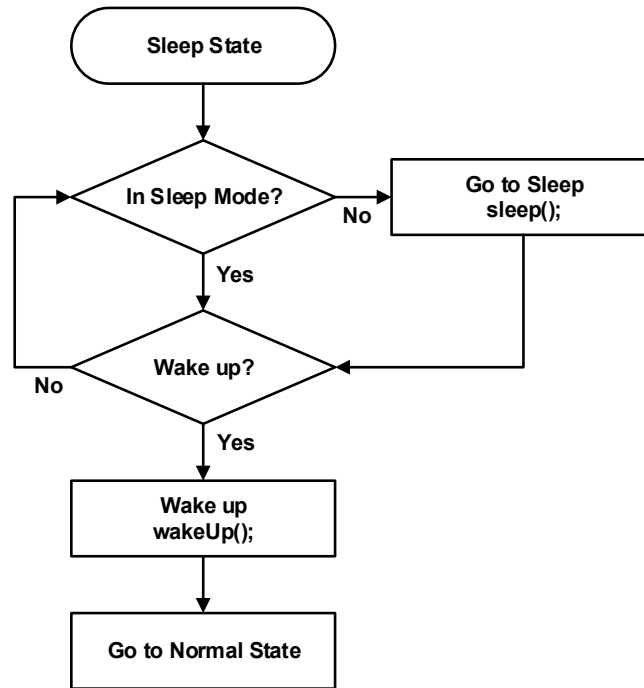


Figure 9. Sleep State Flowchart

2.5 Fault Management

The BMS sample project contains fault management mechanism that incorporates and demonstrates the fault management examples described in the ISL94212 datasheet. The fault management flowchart is given in Figure 10.

When any function returns an error (including the one checking the fault pin) the BMS state machine makes a transition to Fault state and processes the error.

A fault pin assertion or a fault status register automatic response results in a BMS fault error return. In that case, all fault registers are read and information is extracted into a data structure. It has built in the following mechanisms:

- **Oscillator fault:** Wakes up any sleeping device in the stack
- **Parity error (register checksum):** Re-enters setup, checks memory

If NAK or communication fault response, incorrect CRC or DATA READY assertion timeout error is returned the following sequence is called:

- Send a packet with ones to reset potential flipped daisy chain port
- Send ACK and wake up any sleeping device inside the stack

If EEPROM error is returned (incorrect shadow registers), the whole stack is reset and BFE is reconfigured.

If the BFE has successfully completed fault management procedures and faults are successfully cleared, the state machine makes a transition to Normal state, otherwise a hardware reset is needed.

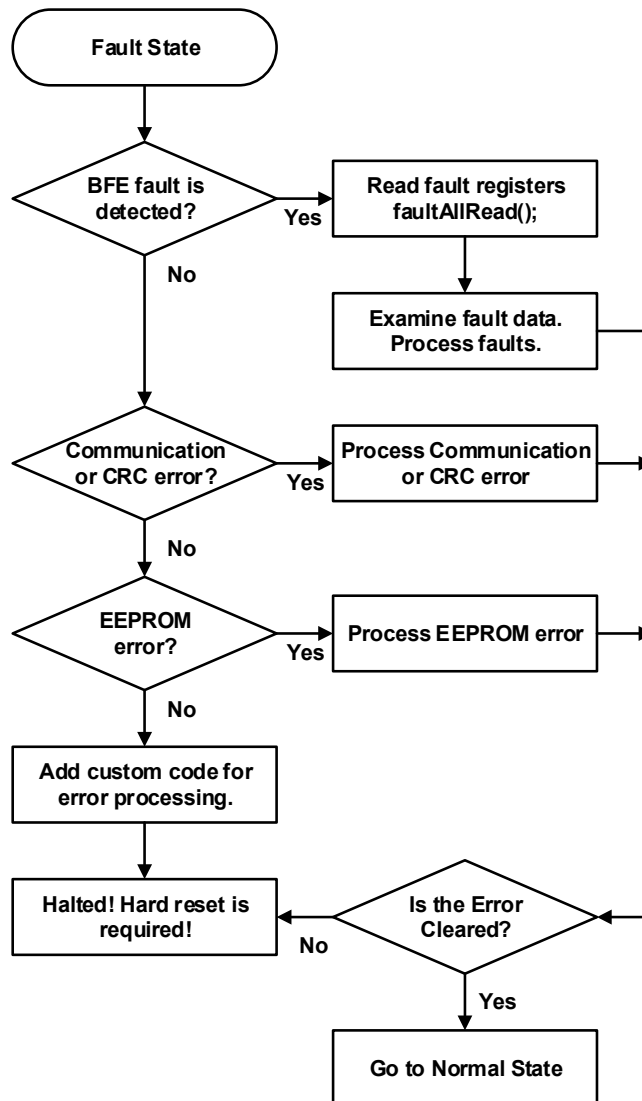


Figure 10. Fault Management Flowchart

3. Requirements and Setup

3.1 Software Setup

The required tools for directly compiling, debugging and running the project, provided with the BFE software package under scope, include e² studio Integrated Development Environment (IDE) with the GCC Arm® Embedded toolchain, the Flexible Software Package (FSP) and a terminal emulator program. Visit [e² studio](#) for instructions and more information. If an alternative microcontroller and/or integrated development environment and toolchain are used, you should install them according to related instructions, then port the software library, connect it to hardware abstraction layer API, and then port the sample code to be able to go through the example.

A terminal emulator program is needed for running the user interface (UI). A useful option is Tera Term, which can be downloaded from the [Tera Term Home Page](#).

3.2 Importing the Sample Project

The sample project provided with this document can be imported into e² studio workspace by following these steps:

1. Select **File > Import**

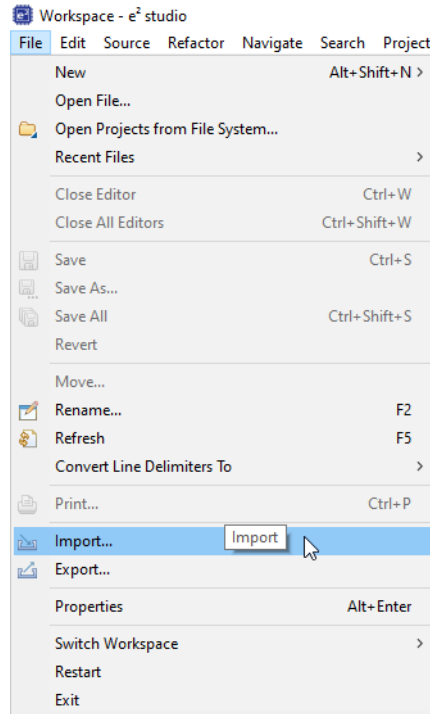


Figure 11. File Menu to Import the Sample Project

2. Select **Existing Project into Workspace** and click **Next** button.

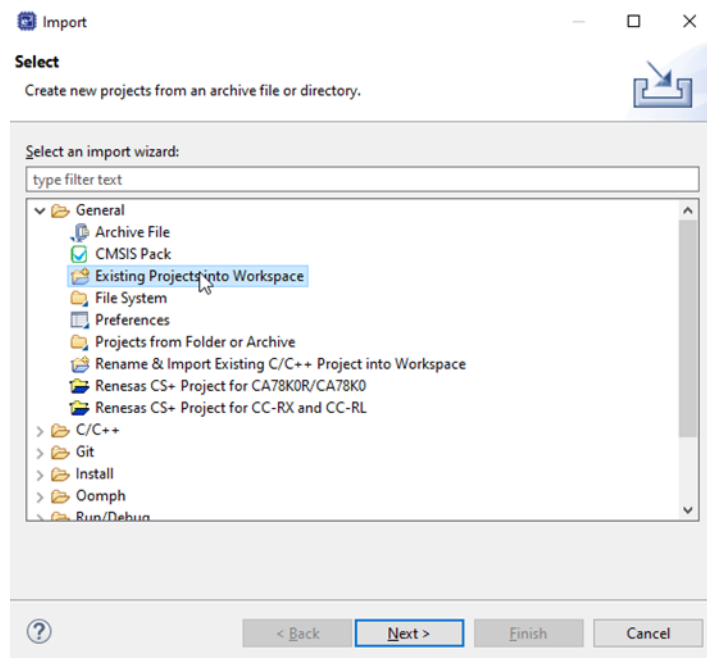


Figure 12. Selection of the Import Option

- Choose **Select archive file** option, click the **Browse...** button and then select the sample project file (.zip). Click the **Finish** button.

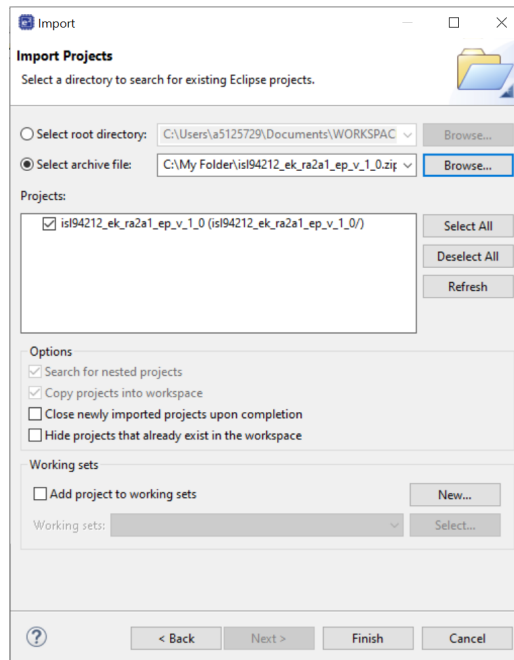


Figure 13. Import the Sample Project

- The project is now imported into the e2 studio workspace. Figure 14 shows the imported project structure. Double-click on **configuration.xml** and press **Generate Project Content** to generate the additional hardware specific project files.

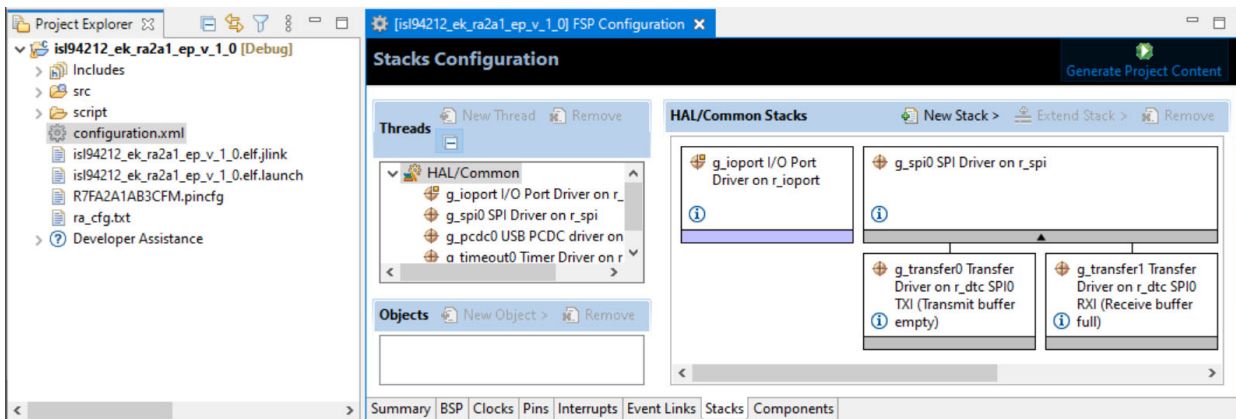


Figure 14. Generate Additional Project Files

- Now the project is ready to be built. Open Project from top menu and press **Build All**. Then open run **Run** from top menu and select **Debug As->3 Renesas GDB Hardware Debugging**. The IDE changes automatically to a debug perspective and a debug session is started. To run the de code you must press the **Resume** button twice.

3.3 Hardware Setup

The hardware setup, needed for running the demo and familiarization with the software package, is given in Table 1. It includes all evaluation boards, power supply, measurement and control tools. Some of the items are considered as optional. The used MCU evaluation board does not provide any galvanic insulation. Therefore, additional isolation may be required when working with high voltage battery packs (not needed in the proposed test setup).

Table 1. Hardware Setup

#	Item	Qty.	Description
1	ISL94212EV1Z	3 ^[1]	ISL94212 Evaluation Boards
2	MCB_PS2_Z	3 ^[1]	12 Cell resistor ladder board (or target battery pack)
3	EK-RA2A1	1	RA2A1 Evaluation Board
4	USB micro cable	2	Connection between EK-RA2A1 and PC
5	USB isolator (optional)	1	Dual isolator. Additional 5V power supply might be required
6	DC Power Supply	1	6V to 60V Regulated
7	Precision multimeter	1	Multipurpose DC voltmeter compatible with total battery voltage
8	Oscilloscope (optional)	1	Equipped with Logic Analyzer
9	Cables and wires	-	Connection between MCU and BFE boards, Resistor ladders and power supply
10	Personal computer or notebook	1	Running Windows® or Linux with USB support

1. The quantity depends on the stack size.

3.4 Battery Front End Evaluation Board

The BFE evaluation board ISL94212EVKIT1Z demonstrates the application of the ISL94212 Li-ion battery manager IC. It consists of ISL94212EVZ board and MCB_PS2_Z resistor ladder, simulating the battery pack (Figure 15). The ISL94212 communicates to a host microcontroller via an SPI interface and to other ISL94212 devices using a robust, two-wire daisy chain system. The primary evaluation board provides configuration options that can be set using the switches. More information can be found on the ISL94212EVKIT1Z product page. For a detailed setup, see the ISL94212EVKIT1Z manual.

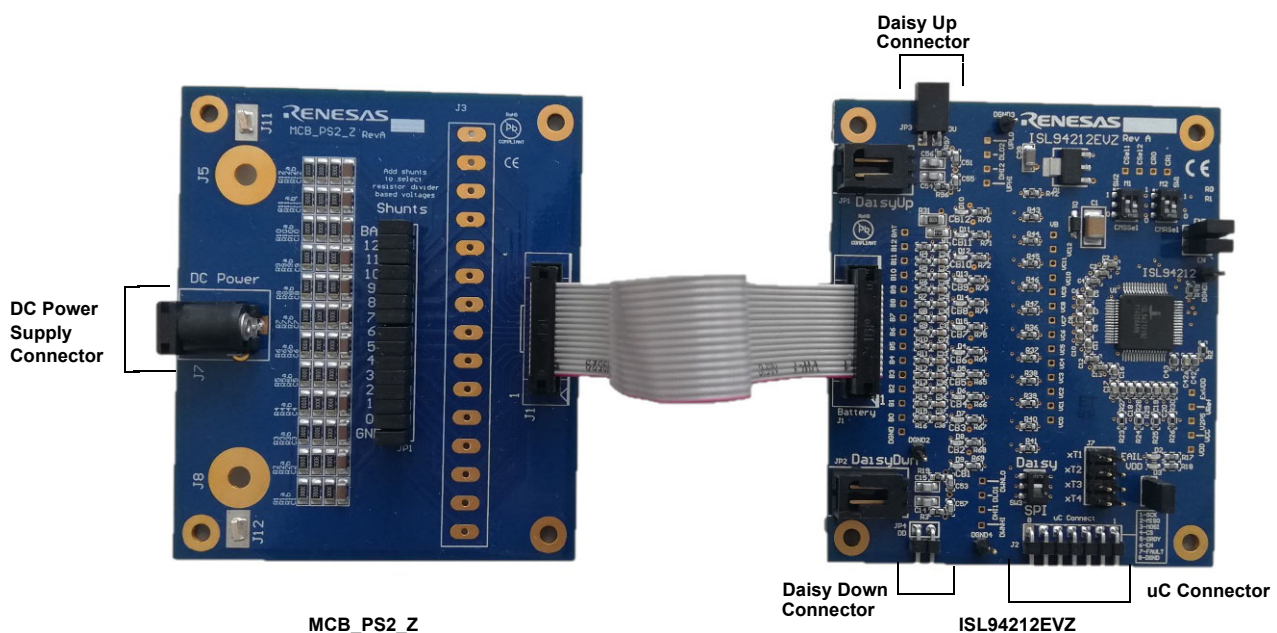


Figure 15. Resistor Ladder and ISL94212 Evaluation Board

3.5 MCU Evaluation Board

The MCU evaluation board is EK-RA2A1 that facilitates firmware development of R7FA2A1AB3CFM 32-Bit MCU from RA2A1 microcontroller group of Renesas Advanced (RA) family (Figure 16). The selected microcontroller is suitable for BMS applications as it has low power modes combined with 24-bit Sigma-Delta ADC, appropriate for current monitoring (Coulomb counting algorithm). Another alternative from the same family is RA4W1 that has a built-in Bluetooth module.

The evaluation board provides access to its resources and USB micro device connector for establishing virtual serial communication with the terminal. There is also a LED that is used to indicate fault state.

More information can be found on the [EK-RA2A1](#) product page. For detailed setup, see the EK-RA2A1 User's manual.

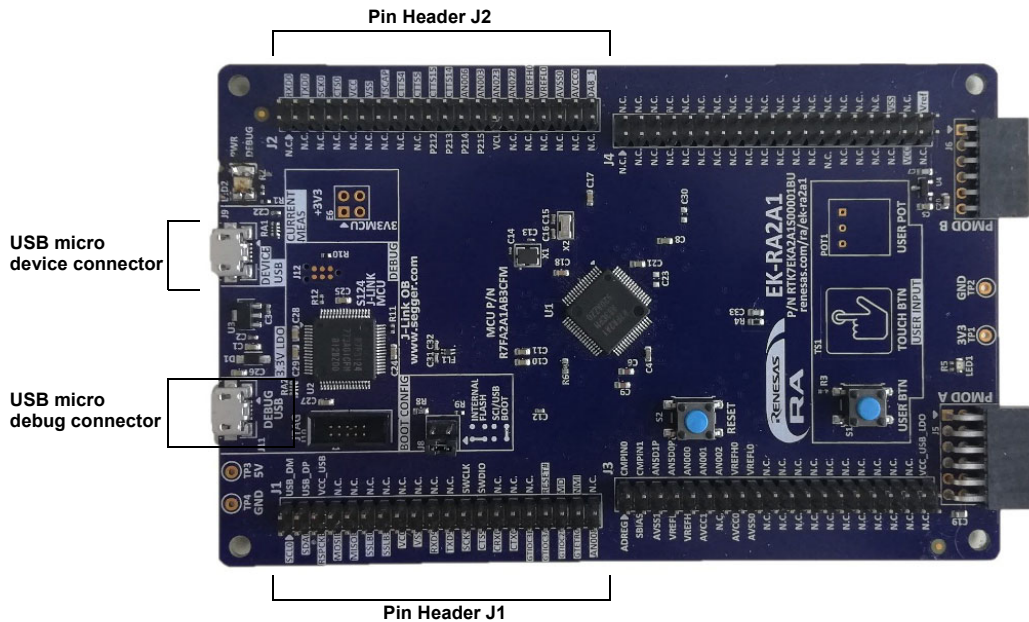


Figure 16. RA2A1 Evaluation Board

3.6 Power Supply

When a battery pack is not available, a combination of a DC power supply and a resistor ladder is used. It must provide the necessary voltage to simulate full cells voltage range without entering current limit to power the resistor ladder and the evaluation board with all LEDs on. At a typical operating voltage of 3.9.6V, when connected between J11 and J12, simulates around 3.3V cell voltage. This board consumes about ~120mA in normal operation mode and ~102mA in sleep mode. Do not go under 6V (the ISL94212 internal voltage regulator inhibits operation) or exceed 60V!

3.7 Precision multimeter

A precision multimeter is used as a voltmeter to monitor battery pack voltage, individual cell voltages or another voltage of interest. The recommended start-up connection is between Pack+ (J11) and Pack- (J12).

3.8 Oscilloscope

An oscilloscope can be used optionally to capture signals in the SPI communication channel between the MCU and stack master device together with the FAULT and DATA READY signals. The vertical daisy chain communication ports can be captured when using a differential probe or two passive probes, referenced to ground and divided by scope math. For easier capture and better results, Renesas recommends equipping the scope with a logic analyzer.

3.9 Hardware Assembly

The hardware assembly for standalone mode (single ISL94212) is shown in Figure 17 and for daisy chain mode (stack of 3 ISL94212) – in Figure 18. When stacked, the ISL94212 boards are interconnected directly using headers JP3 and JP4. However, twisted pair cables can also be used. The MCU and ISL94212 boards are connected with external cables (see Table 2). Make sure that all jumpers are in place and all microswitches (SW1, SW2, SW3) are in the right position as shown in the figures to set the right daisy chain speed and device position inside the stack. For more details, see the ISL94212EVKIT1Z Manual.

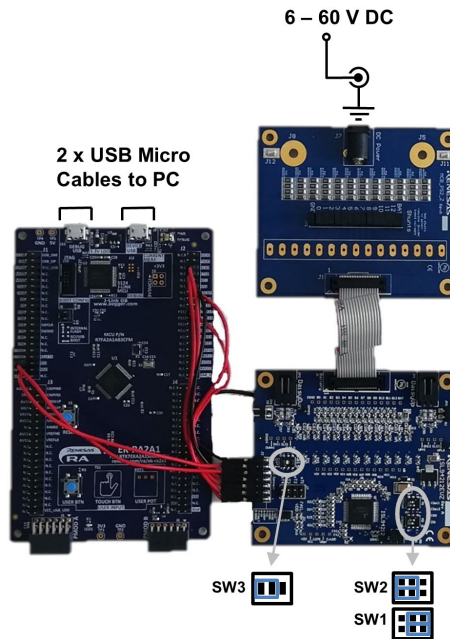


Figure 17. BFE and MCU Boards Hardware Assembly in Standalone Mode

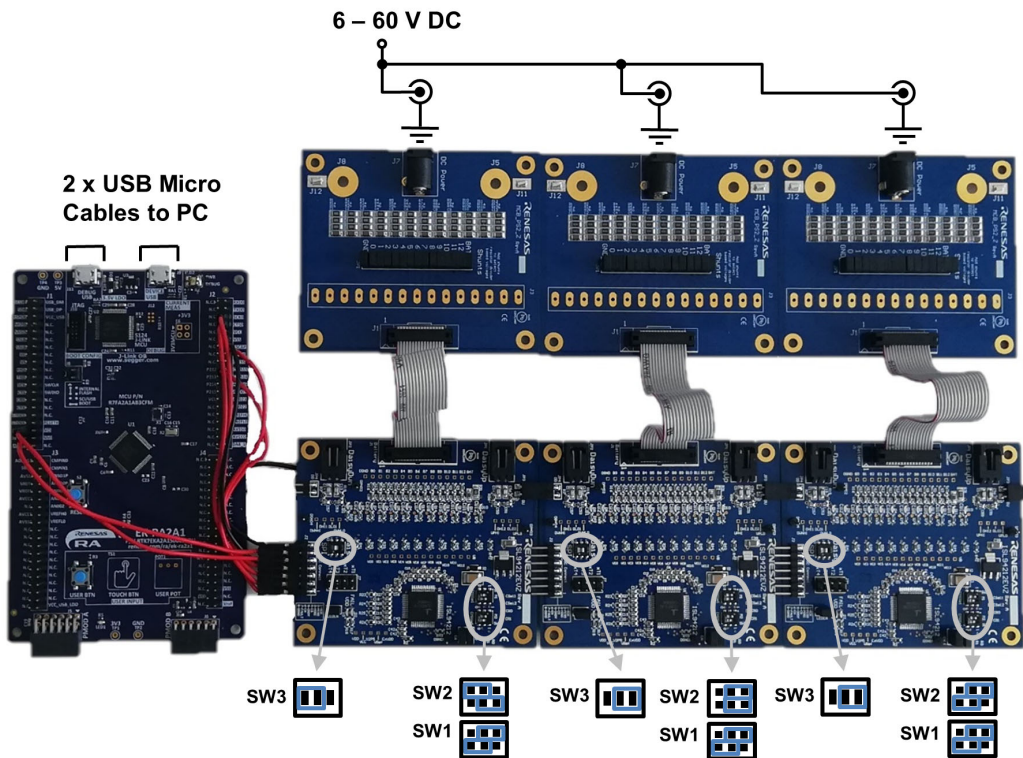


Figure 18. BFE and MCU Boards Hardware Assembly in Daisy Chain Mode (3 Devices Stacked)

The MCU board is powered by the host computer through the debug USB connector J11. However, device USB J9 must be also connected to enable the virtual serial communication with the terminal software. The resistor ladders are connected and powered up in parallel so that no matter how many boards are stacked the supply voltage is always up to 60V for safety reasons. The positive terminal of the MCB_PS2_Z board is J5/J11 and the negative terminal is J8/J12. Connector J7 can also be used to provide power to the board. The J7 tip is positive and the ring is negative.

Table 2. BFE and MCU Board Connection

Signal name	R7FA2A1AB3CFM - Pin	EK-RA2A1 - Header	ISL94212EVZ (master) - Header
SCK	P103	J1-RSPCKB	J2 - 1
MISO	P104	J1-MISOB	J2 - 2
MOSI	P105	J1-MOSIB	J2 - 3
CS	P102	J1-SSLB0	J2 - 4
DRDY	P101	J1-RXD9	J2 - 5
FAULT	P204	J1-SCK9	J2 - 6
DGND	--	J2 - VSS	J2 - 7
EN (optional)	P402	J1-CTS9	EN

4. User Interface

The user interface (UI) demonstrates the functionality of the sample code by displaying the current state, acquired data from BFE, fault data, and providing control over the measurement rate, cell balancing, sleep/wake up, and fault clearing. It directly controls the BMS state machine. However, the UI indirectly sends commands to the BFE.

4.1 Terminal setup

To initiate a terminal session, a new connection must be opened (Figure 19). When using terminal software, the serial connection settings are given in Table 3. The PC user must have read/write access permission for the USB port.

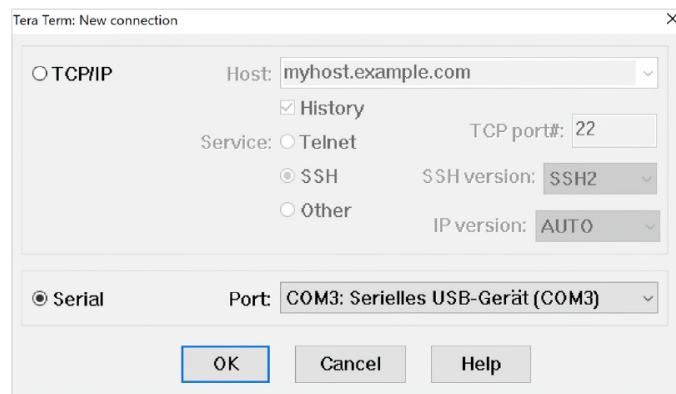


Figure 19. Tera Term New Connection Window

If Windows® recognized the board correctly, it is listed in Tera Term as a serial connection. If the board is not listed at all or the Device Manager indicates an error, there might be a problem with the driver. See the latest support entry for this topic in the [Renesas Knowledge Base](#) to resolve this.

Table 3. Serial Connection (Terminal) Settings

Parameter	Value
New Line (Receive)	CR
New Line (Transmit)	CR
Terminal Mode	VT100
Baud Rate	9600
Data Bits	8 bits
Parity	none
Stop Bits	1 bit
Flow Control	none

4.2 Turning On the Setup

Before turning on the setup, ensure the following:

- The BMS sample project is imported into e² studio IDE (See [Importing the Sample Project](#)).
- The library has the right configuration, corresponding to the hardware setup (See to [Configuration of the BFE Library](#)).
- If real battery is used, the battery limits are set, corresponding to the used chemistry and balancing requirements (See to [Configuration of the BFE Library](#)).

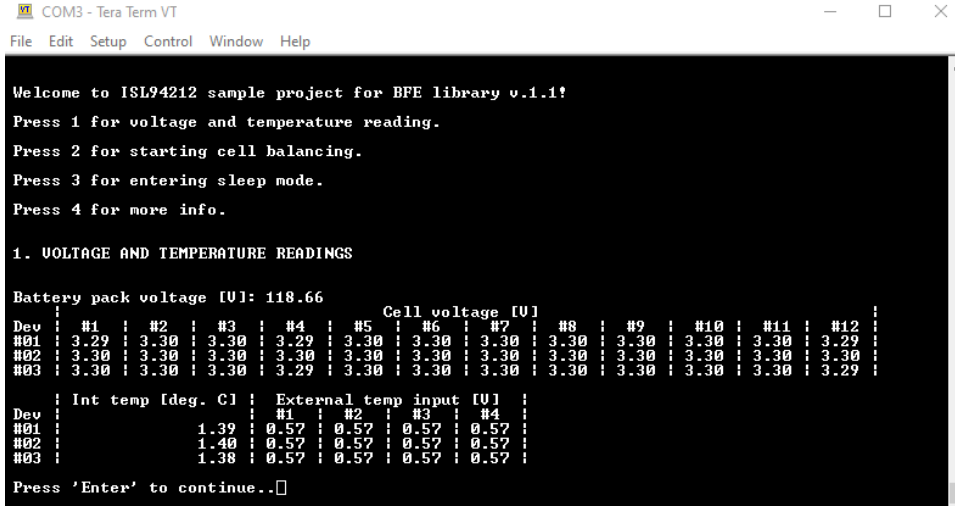
The hardware setup must be turned on in the following sequence:

1. Ensure that everything is properly connected and both USB cables are connecting the MCU evaluation board and the PC. Check jumpers and switches of all boards.
2. Set the power supply to 39.6V and power up the resistor ladder (MCB_PS2_Z board). Check that the voltages on the J3 connector of the board are +3.3V from PIN(n) to Pin(n-1).
3. In e² studio, build the sample project and initiate debugging to download the software to the MCU. Run the software.
4. Open the Tera Term terminal and press **Enter** to display the menu.

The MCU should start together with or after the BFE, otherwise it returns an error when trying to initialize an unpowered BFE.

4.3 Using the Demo

When the hardware setup is fully connected and powered-up, and the terminal session is initiated, the user interface becomes active. After pressing **Enter**, the MCU should respond with the menu shown in [Figure 20](#). Now the BMS is in Normal state and the MCU expects a user input. By pressing **1**, all voltage and temperature scans are triggered, results are read and displayed onto the terminal inside a table. Return to the main menu by pressing **Enter**.



```

COM3 - Tera Term VT
File Edit Setup Control Window Help

Welcome to ISL94212 sample project for BFE library v.1.1!
Press 1 for voltage and temperature reading.
Press 2 for starting cell balancing.
Press 3 for entering sleep mode.
Press 4 for more info.

1. VOLTAGE AND TEMPERATURE READINGS

Battery pack voltage [U]: 118.66
Cell voltage [U]
Dev  #1  #2  #3  #4  #5  #6  #7  #8  #9  #10 #11 #12
#01  3.29 3.30 3.30 3.29 3.30 3.30 3.30 3.30 3.30 3.30 3.30 3.29
#02  3.30 3.30 3.30 3.30 3.30 3.30 3.30 3.30 3.30 3.30 3.30 3.30
#03  3.30 3.30 3.30 3.29 3.30 3.30 3.30 3.30 3.30 3.30 3.30 3.29

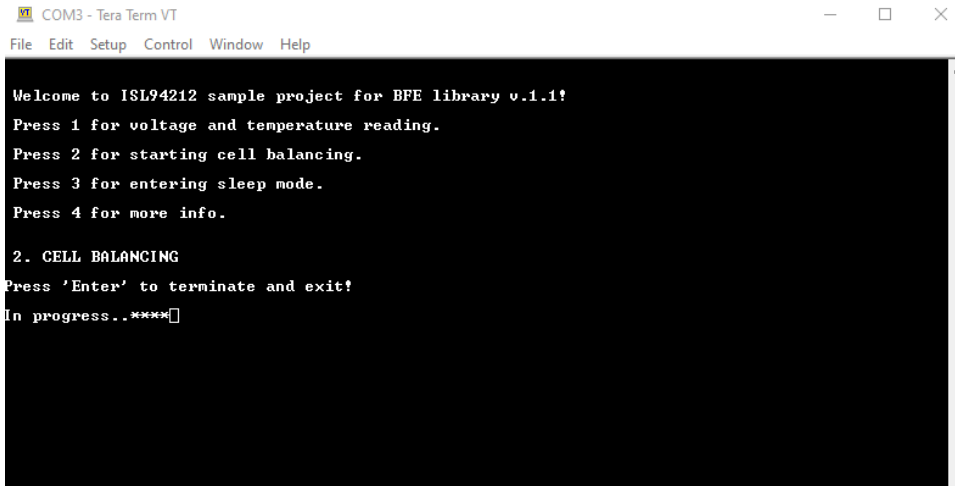
Int temp [deg. C] External temp input [U]
Dev  #1  #2  #3  #4
#01  1.39 0.57 0.57 0.57 0.57
#02  1.40 0.57 0.57 0.57 0.57
#03  1.38 0.57 0.57 0.57 0.57

Press 'Enter' to continue..

```

Figure 20. User Interface Voltage and Temperature Readings

When you press **2** inside the menu, the BMS goes to cell balancing state ([Figure 21](#)) and the balancing algorithm immediately runs. After each loop, an asterisk is displayed. When all cells are balanced, the MCU sends a message and BMS goes back to normal state. The balancing activity can be interrupted by pressing **Enter**. The parameters, related to cell balancing, cannot be modified using the UI and must be hardcoded into the sample project.



```

COM3 - Tera Term VT
File Edit Setup Control Window Help

Welcome to ISL94212 sample project for BFE library v.1.1!
Press 1 for voltage and temperature reading.
Press 2 for starting cell balancing.
Press 3 for entering sleep mode.
Press 4 for more info.

2. CELL BALANCING

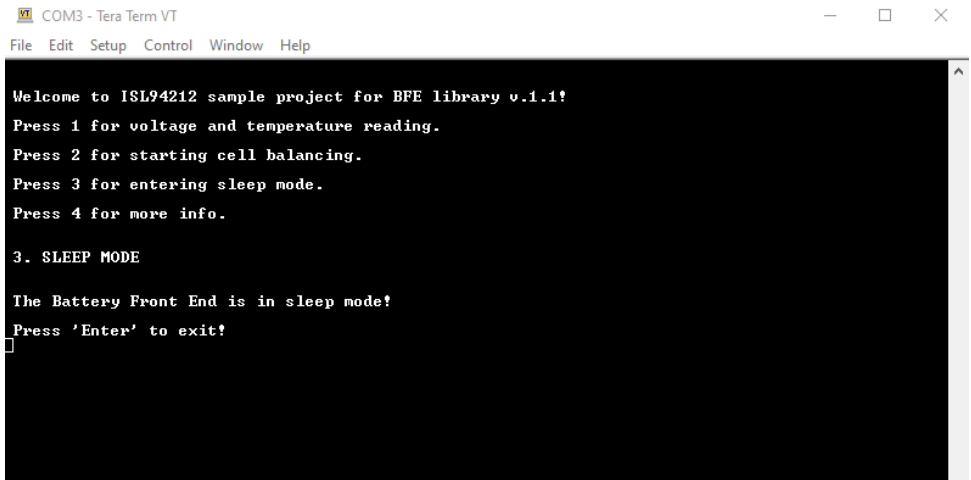
Press 'Enter' to terminate and exit!
In progress..****

```

Figure 21. User Interface Cell Balancing

When you press **3** inside the menu, the BMS goes to Sleep state ([Figure 22](#)). This can be also noticed as the VDD LEDs of the ISL94212 boards fade out slightly because in sleep mode the voltage regulator drops down to 2.5V. If watchdog timeout is used for sleep mode or it expires when the BFE is already in sleep mode, the FAULT pin is asserted and the FAIL LED is on. When the system is in sleep mode, it can wake up and return to normal state by pressing **Enter**. When a fault condition is detected, the system goes automatically into fault state and the UI displays more details ([Figure 23](#)). The FAIL LED of the MCU board is on. You can run the built-in fault

management algorithms by pressing **Enter**. If the faults are successfully cleared, the BMS goes to normal state otherwise a message is displayed that a hard reset is needed (Figure 24).



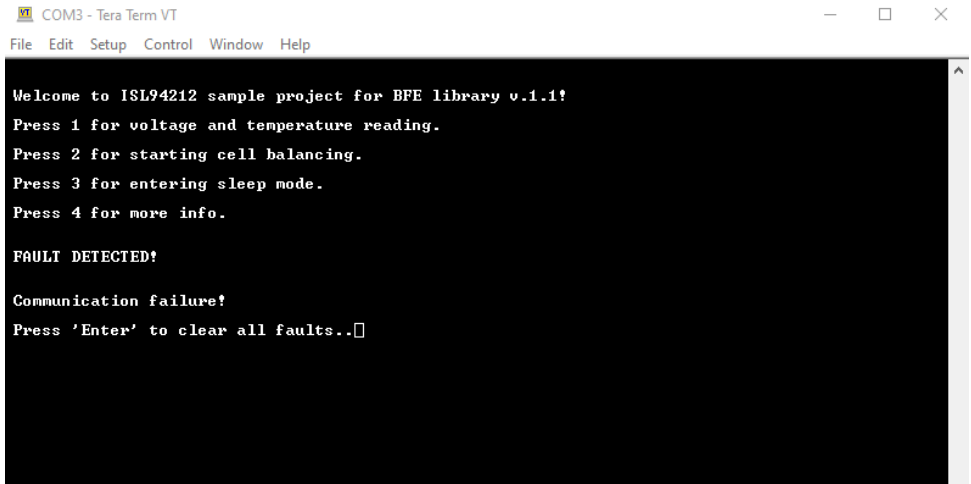
```
COM3 - Tera Term VT
File Edit Setup Control Window Help

Welcome to ISL94212 sample project for BFE library v.1.1!
Press 1 for voltage and temperature reading.
Press 2 for starting cell balancing.
Press 3 for entering sleep mode.
Press 4 for more info.

3. SLEEP MODE

The Battery Front End is in sleep mode!
Press 'Enter' to exit!
█
```

Figure 22. User Interface Go to Sleep and Wake Up Mode



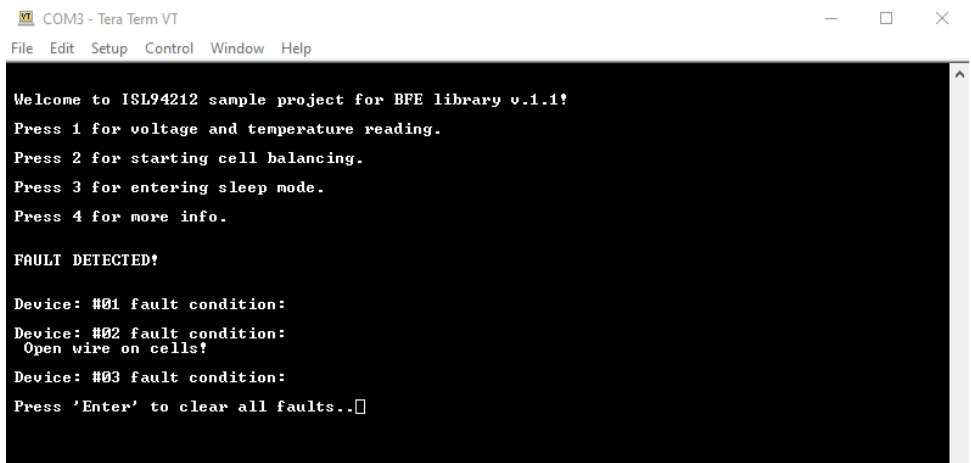
```
COM3 - Tera Term VT
File Edit Setup Control Window Help

Welcome to ISL94212 sample project for BFE library v.1.1!
Press 1 for voltage and temperature reading.
Press 2 for starting cell balancing.
Press 3 for entering sleep mode.
Press 4 for more info.

FAULT DETECTED!

Communication failure!
Press 'Enter' to clear all faults..█
```

Figure 23. User Interface Fault State



```
COM3 - Tera Term VT
File Edit Setup Control Window Help

Welcome to ISL94212 sample project for BFE library v.1.1!
Press 1 for voltage and temperature reading.
Press 2 for starting cell balancing.
Press 3 for entering sleep mode.
Press 4 for more info.

FAULT DETECTED!

Device: #01 fault condition:
Device: #02 fault condition:
Open wire on cells!
Device: #03 fault condition:
Press 'Enter' to clear all faults..█
```

Figure 24. Message when Faults and Errors Cannot be Cleared and a Hard Reset is Required

4.4 Debugging

The e² studio integrated development environment and the GCC Arm® Embedded toolchain provide various tools for debugging, including variables and memory viewer (Figure 25). You can add breakpoints and examine, in what situation the CPU runs that code. If a fault cannot be cleared and system hard reset is needed, the execution of code is halted so that you can identify and analyze the error. The error return can be found in the `bms_err` variable and decoded using the error enumerator in `r_bfe_api.h`.

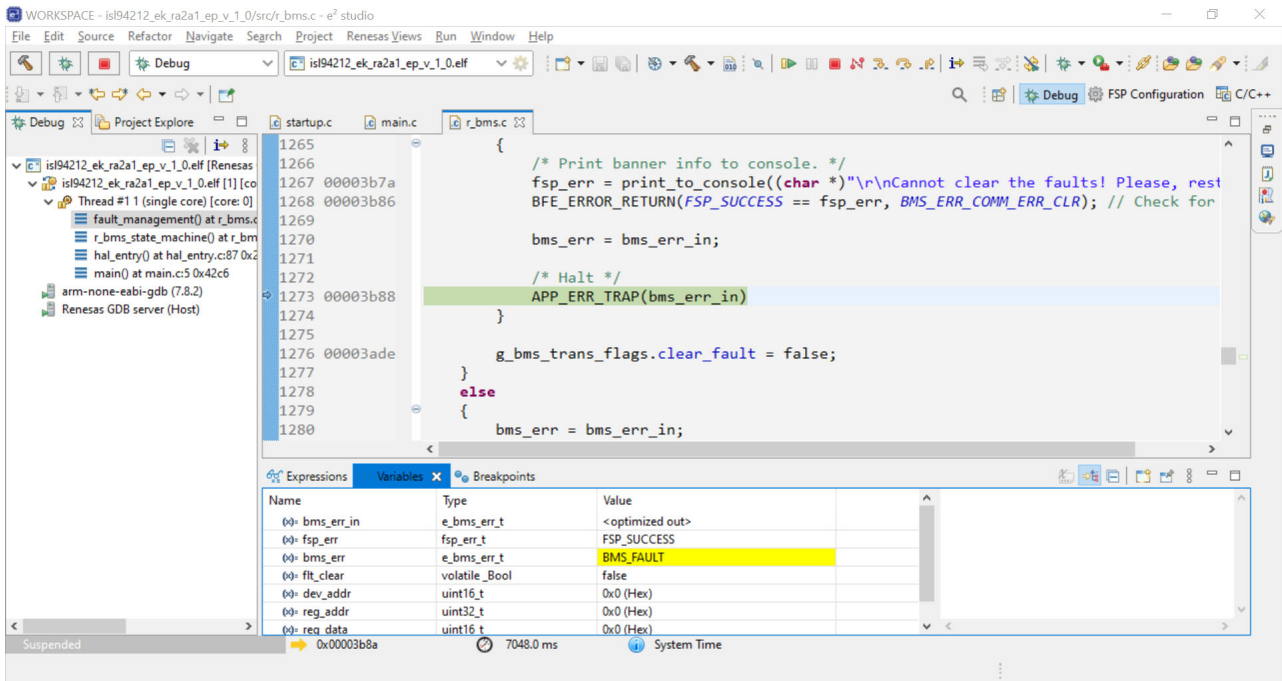


Figure 25. Debugging the Sample Project

5. Glossary

Term	Definition
ADC	Analog-to-Digital Converter is a device that converts an analog signal to a digital code.
AC-Coupling	A method of connecting transmitter to a receiver using series capacitors. This only transmits signal transitions, not DC voltage levels, therefore, can be useful for constantly toggling signals like clocks, especially when trying to isolate the DC voltage levels.
BFE	Battery Front End is a device that is part of a battery management system and provides system protection and cell monitoring and balancing.
BMS	Battery Management System manages a rechargeable battery. Typically Li- based chemistries.
CRC	Cyclic Redundancy Check – A method of determining if a block of data was stored or transmitted correctly. Involves the addition of one or more bytes of extra information to determine data integrity. One of several algorithms can be used.
EEPROM	Electrically-Erasable Programmable Read-Only Memory – Commonly used non-volatile memory device.
EVK	Evaluation Kit is a demonstration board built for a specific device to easy its evaluation.
FET	Field-Effect Transistor is a type of transistor that uses electric field to control the current though its channel.
FSP	Flexible Software Package supplies engineers with lightweight and highly efficient functions and drivers to ease implementation of common use cases in embedded systems.
GCC	GNU Compiler Collection is an optimizing compiler produced by GNU Project.
GPI	General-Purpose Input - An input signal that can be programmed for many different purposes.

6. Ordering Information

All necessary parts for the hardware setup to be ordered from Renesas are given in [Table 4](#).

Table 4. Software Package Demo Hardware Order List

Part Number	Description	Product page
ISL94212EV1Z ^[1]	ISL94212 Evaluation Board	ISL94212EVKIT1Z
MCB_PS2_Z ^[1]	12-Cell resistor ladder board	
EK-RA2A1	RA2A1 Evaluation Board	EK-RA2A1 - Evaluation Kit for RA2A1 MCU Group Renesas

1. Multiple items may have to be ordered, depending on the desired stack size.

7. Revision History

Revision	Date	Description
1.01	Nov 17, 2021	Updated the following sections: <ul style="list-style-type: none"> ▪ Configuration of the BFE Library ▪ Configuration and Control Structures ▪ Battery Management System Demo ▪ Obtaining Measurement ▪ Power Supply ▪ Turning On the Setup Updated Figures 5, 7, 8, 9, 10, 20, 21, 22, 23, 24 Updated Table 2.
1.00	Aug 16, 2021	Initial release

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.