MCU Sample Code for Driving the RAA489206 16-Cell Battery Front End

This document describes the accompanying sample code that demonstrates the features of the RAA4889206 Battery Front End (BFE) and its interactions with an MCU. It provides examples of routines, sequences, and good practices to configure, initialize, and interact with the BFE. However, this sample code is not intended to be a complete system solution for product deployment.

Contents

1. Overview **4**
1.1 Assumptions and Advisory Notes 4
2. RAA489206 BFE Overview **5**
2.1 Features 5
2.2 Applications 5
2.3 Typical Application 6
3. General Software Structure **6**
4. How to Use the Demo Project **9**
4.1 Operating Environment 9
4.2 Importing the Demo Project 10
4.3 Building and Debugging 12
4.4 Demo Project Functional Description 12
4.4.1 BFE and EK-RA4W1 Boards 12
4.5 Terminal Emulator 12
4.6 Use of Command-Line Interface (CLI) 13
5. Demo Project Implementation **14**
5.1 FSP Architecture 14
5.2 BAL Implementation 16
5.2.1 BAL Interface 16
5.2.2 BFE API Structure 17
5.2.3 BFE Interface Instance Structure 22
5.3 RAA489206 BFE Instance Implementation 23
5.3.1 Header File r_bfe_raa489206.h 23
5.3.2 Source File r_bfe_raa489206.c 31
5.3.3 Reset and Device Registers 32
5.3.4 Registers Bank 33
5.3.5 Private (Static) Variables and Functions 35
5.3.6 Interface API Implementation 37
6. Sample Battery Management System **64**
6.1 Overview 64
6.2 Header File r_bms.h 65
6.3 Source Code r_bms.c 65
6.4 Declarations 65
6.5 The bms_main Function 67
7. State-of-Charge Application **71**
8. CLI Commands List **72**

8.1 BFE command group 72
8.1.1 Initialize Device 72
8.1.2 Discharge Overcurrent (DOC) Threshold 72
8.1.3 Charge Overcurrent (COC) Threshold 73
8.1.4 Discharge Short-Circuit Current (DSC) Threshold 73
8.1.5 Internal Over-Temperature Fault (IOTF) Threshold 73
8.1.6 Internal Over-Temperature Warning (IOTW) Threshold 73
8.1.7 Maximum Cell Voltage Delta (MAXDELTA) Threshold 74
8.1.8 Cell Overvoltage (VCELLOV) Threshold 74
8.1.9 Cell Undervoltage (VCELLUV) Threshold 74
8.1.10 Pack Overvoltage (VPACKOV) Threshold 74
8.1.11 Pack Undervoltage (VPACKUV) Threshold 75
8.1.12 Thresholds 75
8.1.13 BFE status 75
8.1.14 Scan 76
8.1.15 FETs Commands 77
8.1.16 Mode 78
8.1.17 Cells Count - Cells Select 78
8.1.18 Shunt Resistor Value 78
8.2 Register (REG) Command 79
8.2.1 Read Register 79
8.2.2 Write Register 79
8.3 Measurement (MEAS) Command Group 79
8.3.1 Vpack 79
8.3.2 Ipack 80
8.3.3 Vcells 80
8.3.4 Vcell N 80
8.3.5 Total Cell Voltage 80
8.3.6 Internal Temperature 81
8.3.7 Regulator Voltage 81
8.3.8 Regulator Current 81
8.4 Cell Balancing Command Group 81
8.4.1 Cell Balancing Enable/Disable 81
8.4.2 End-of-Charge Voltage 82
8.4.3 End-of-Charge Current 82
8.4.4 Automatic Cell Balancing Enable/Disable 82
8.4.5 Cell Balancing FETs Configuration 82
8.4.6 Cell Balancing Trigger 83
8.4.7 Cell Balancing Mask 83
8.4.8 Cell Balancing End-of-Charge Enable/Disable 83
8.4.9 Current End-Of-Charge Enable/Disable 83
8.4.10 Cell Balancing Charge Enable/Disable 84
8.4.11 Cell Balancing Cell State 84
8.4.12 Cell Balancing Minimum Delta Threshold 84
8.4.13 Cell Balancing Maximum Threshold 84
8.4.14 Cell Balancing Minimum Threshold 85
8.4.15 Cell Balancing On Timer 85
8.4.16 Cell Balancing Off Timer 85
8.5 Sample Battery Management System 85

8.6 State-of-Charge Application 86
9. Revision History 86

1. Overview

Figure 1 shows the operating environment of the demo project bfe_raa849206_ek_ra4w described in this document, which runs on the EK-RA4W1 board. The BFE board and the attached battery cells can be the RTKA489206DE0000BU evaluation and resistor ladder boards, or any custom board that includes connectivity between the BFE device and the target MCU. The project implements a command-line interface (CLI), which is accessed by a terminal emulator, such as Tera Term on a PC connecting with the EF-RA4W1 board using an USB cable.

The CLI provides commands, which execute the interaction sequences that systems and devices interfacing with BFEs follow to use BFE features. This sample code also contains a sample BMS application that monitors the status of the BFE and reports critical fault events over the terminal interface.

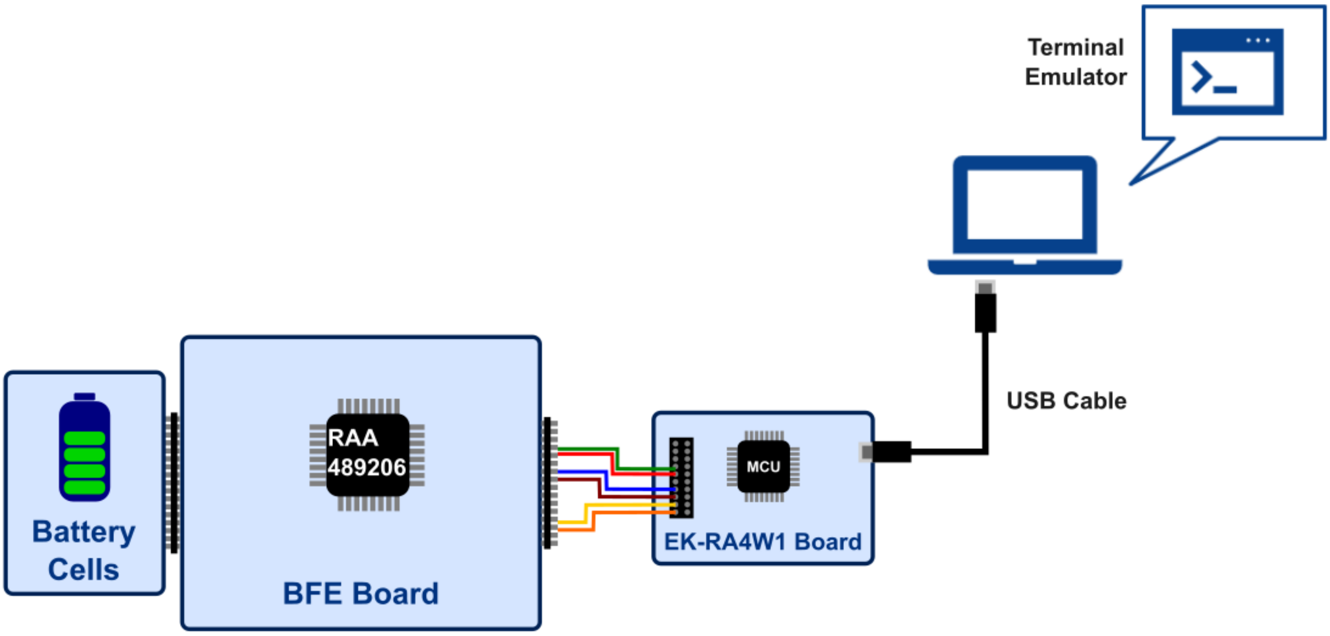

Figure 1. Demo Project Operating Environment

1.1 Assumptions and Advisory Notes

1. It is assumed you possess basic understanding of microcontrollers and embedded systems hardware.
2. Renesas recommends reviewing the *EK-RA4W1 Quick Start Guide* and *EK-RA4W1 Manual*, in addition to the RAA489206 Datasheet and Evaluation Kit Manual, to get acquainted with MCU and BFE features before proceeding further.
3. Flexible Software Package (FSP) and Integrated Development Environment (IDE) such as e2 studio are required to modify, extend, or develop embedded applications on the target EK-RA4W1 kit.
4. Instructions to download and install software, import example projects, build them and program the EK-RA4W1 board are provided in *Renesas e2studio 2021-07 or Higher User's Manual: Quick Start Guide* (R20UT5034EJ).

Note: Do not install the sample code into your product. The operation of sample code is not guaranteed. Confirming the operation is your own responsibility.

2. RAA489206 BFE Overview

The RAA489206 is a 16-cell Battery Front End (BFE) IC, an essential component of BMS that periodically scans battery status and the operating environment to optimize battery life and prevent catastrophic failures.

To manage the overall state of the battery pack, a differential multiplexer and 16-bit ADC allow for the accurate monitoring of cell voltage, temperature, and load current.

Low current consumption with an average IDLE mode current of 200µA and a SHIP mode current of less than 18µA maximizes the storage and discharge life of a battery pack.

2.1 Features

- High hot plug rating: 62V
- V_{CELL} accuracy: ±10mV
- I_{PACK} accuracy: ±0.2%
- 16-bit V_{CELL} and I_{PACK} measurements
- Charge/Load wakeup detection circuitry
- 4-pin GPIO port
- Integrated 3.3V regulator
- Supports I^2C, SPI, and SPI w/CRC communications

2.2 Applications

- Light electric vehicles such as e-bikes, e-scooters, and e-motorcycles
- Cordless power and gardening tools
- Home appliances
- 24V, 36V, 42V, and 48V portable battery packs
- Telecom and server farms
- Solar farms
- Energy storage systems

2.3 Typical Application

Figure 2 shows a typical MCU-BFE application. See the RAA489206 Datasheet for further information on functionality and communication interfaces.

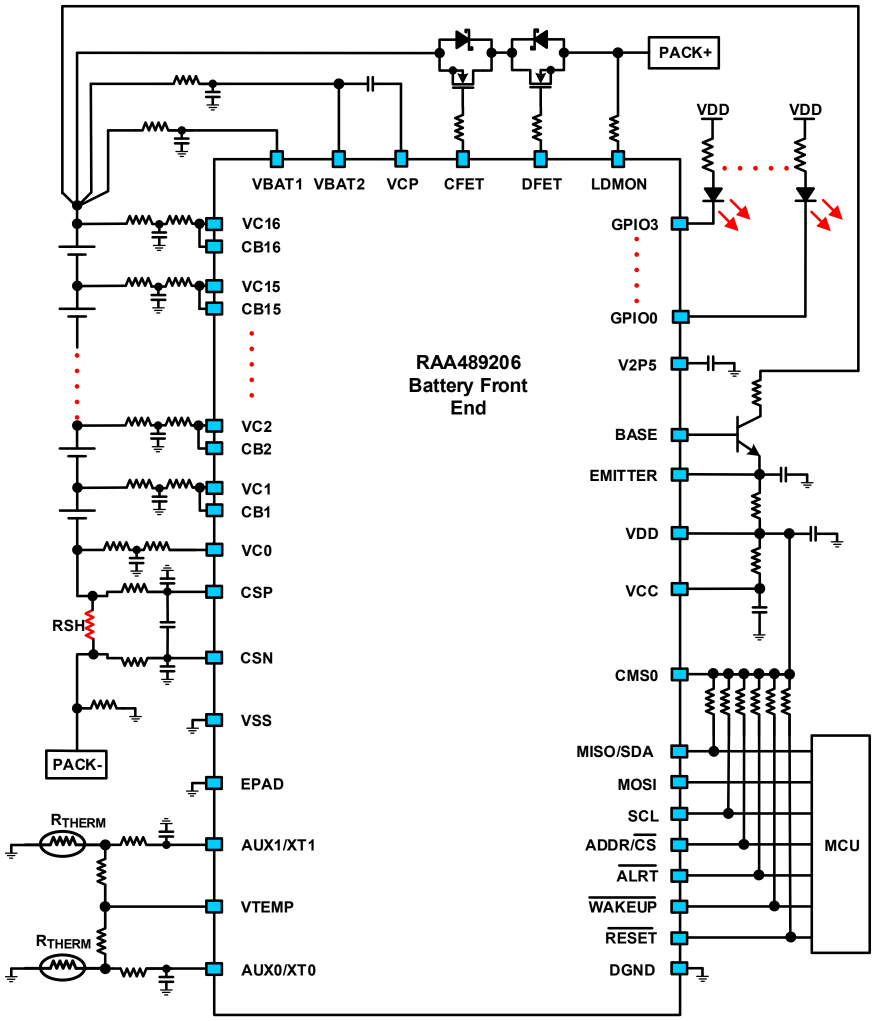

Figure 2. MCU-BFE Typical application

3. General Software Structure

Figure 3 shows the software structure of the sample code described in this document. The user application code block consists of two modules: the CLI and the sample BMS. The CLI provides commands to interact with the BFE and execute tasks such as:

- Set and read BFE registers using hexadecimal notation.
- Read fault and status indicators, and measurements, such as current, voltage and temperature.
- Set protection thresholds, such as overvoltages and undervoltages, maximum voltage difference between cells, internal over-temperature, and discharge, charge and short-circuit currents.
- Clear faults reported by the BFE.
- Read and set BFE mode.
- Perform continuous scan operation to monitor the battery pack, in addition to single system scans.
- Turn ON and OFF power FET drivers for charge and discharge.

The sample BMS can be started by the CLI. It is a sample application that uses the continuous scan operation feature BFE to monitor and protect the battery pack, typical functions of BMSs.

Both CLI and sample BMS applications interact with the BFE through the BFE Abstraction Layer (BAL). The BAL defines a BFE Interface as a structure consisting of an Application Program Interface (API), a Control Structure, and a Configuration Structure. The BAL works as a middleware between the user application code and the hardware. It decouples user applications code from the software that drives the direct interaction with the BFE and allows usage of BFE features through the API of the BFE interface module. Whereas BFE interface structures (API, Control, and Configuration) are mainly declarations of BFE features, the RAA489206 Instantiation defines and implements the interactions that provide those features. The instance uses the Hardware Abstraction Layer (HAL) of Renesas Flexible Software Package (FSP) to access and use MCU peripherals and modules.

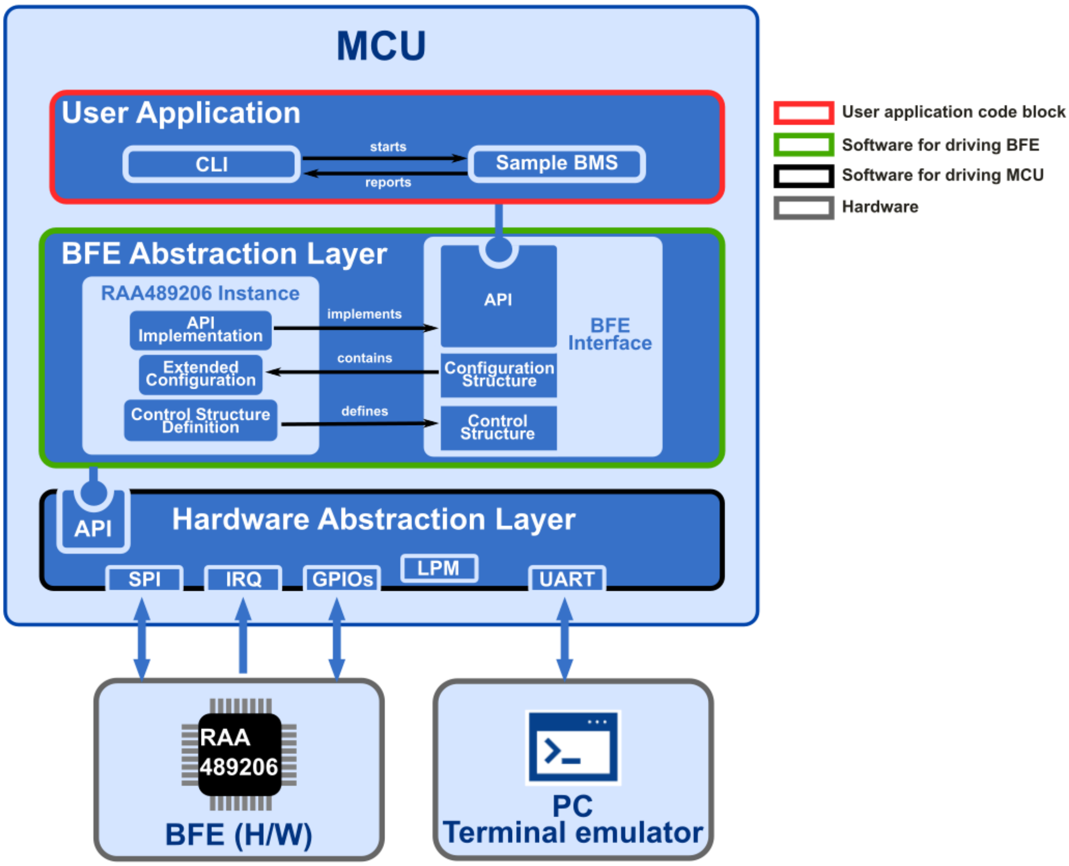

Figure 3. Software Structure of the Sample Code

The BFE instance uses the following APIs of HAL to interact with the BFE device:

- External Interruption Request (IRQ) Interface to detect the ALERT pin events generated by the BFE.
- Serial Peripheral Interface (SPI) Interface to communicate with the BFE.
- General Purpose Input/Output (GPIO) to access and configure I/O ports that configure the communication interface and reset the BFE.
- Universal Asynchronous Receiver-Transmitter (UART) to communicate with the terminal emulator.

Low Power Mode (LPM) to control the power consumption of the MCU during the execution of applications.

Table 1 shows the structure of the sample code. The modules shown in bold within the gray cells contain the code related to the use of BFE functionalities; their code can be modified to extend BFE features or adapt to the requirements of the intended case.

Table 1. Directory Structure of the Sample Code

Directory				Description	Module
ra	fsp	inc	api	Modules APIs	HAL (Generated by FSP)
			instances	Definition of modules instances	
		src	r_*.c	APIs Implementations	
ra_gen	---			Instantiation of HAL modules and main.c that calls the entry point	
ra_cfg	fsp_cfg	r_*_cfg.h		Configuration options files	
src	hal_entry.c	---		Entry point that calls the application main	
	bfe	---	r_bfe_api.h	BAL API	BAL
			r_bfe_raa489206_cfg.h	Configuration macros	
			r_bfe_raa489206.*	BFE instance and API implementation	
	app_lib	cli	*.c *.h	Command-Line Interface implementation	User Application
		cmd	*.c *.h	CLI commands	
		user_app	r_bms.*	BMS sample application	
			r_cli_main.*	CLI application main called by entry function in hat_entry.c	
			r_coulomb_counting.*	Coulomb Counting functions	
			r_icr1865026j_02a.*	Lookup table values of released capacity vs. open circuit voltage of the ICR18650_26J cell	
			r_lookup_table.*	Function that looks and interpolates values in a lookup table	
			r_soc.*	State-of-Charge (SOC) application	

4. How to Use the Demo Project

This section describes the procedure to import the demo project that contains the sample code.

4.1 Operating Environment

Table 2 and Table 3 show the hardware and software requirements to build and debug the provided sample software.

Table 2. Hardware Requirements

Hardware	Description
Host PC	Windows® 10 PC with USB interface
MCU Board	EK-RA4W1 [RTK7EKA4W1S00000BJ]
On-chip debugging emulator	The EK-RA4W1 has a J-Link on-board debugger, so no external debugger is necessary
USB cables	Two USB A/USB micro B cables to connect the EK-RA4W1 (Debugger and serial) to the PC

Table 3. Software Requirements

<table>
<tr><th colspan="2">Software</th><th>Version</th><th>Description</th></tr>
<tr><td rowspan="4">GCC environment</td><td>e2 studio</td><td>2022-04</td><td>Windows® 10 PC with USB interface</td></tr>
<tr><td>GCC ARM Embedded</td><td>10.2.1.20201103</td><td>C/C++ compiler (download available from e^2 studio installer)</td></tr>
<tr><td>Renesas Flexible Software Package (FSP)</td><td>3.3.0 or higher</td><td>Software package for development of projects with the Renesas RA series of MCU devices</td></tr>
<tr><td>Segger J-Flash</td><td>V6.94</td><td>Tool to program on-chip flash memories of MCU devices</td></tr>
<tr><td colspan="2">Header files</td><td></td><td>All API calls and their supporting interface definitions located in the header files (*.h) contained in the src directory</td></tr>
<tr><td colspan="2">Integer types</td><td></td><td>ANSI C99 Exact width integer types declared in stdint.h</td></tr>
</table>

4.2 Importing the Demo Project

The Demo project provided with this document can be imported into an e2 studio workspace by completing the following steps:

1. Select **File** > **Import**

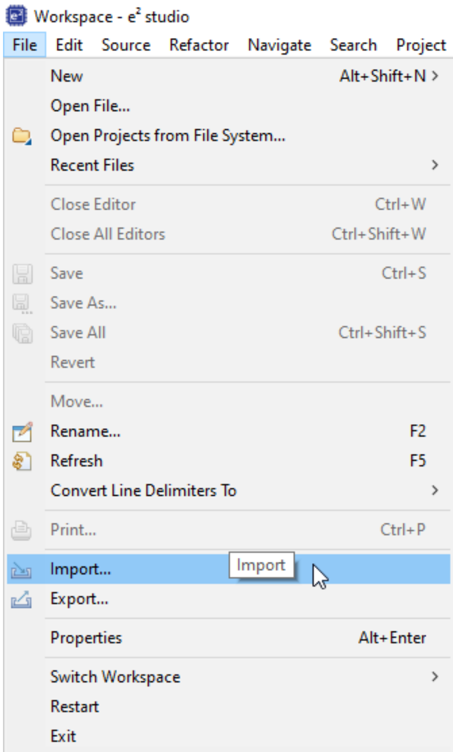

Figure 4. File Menu to Import the Demo Project

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

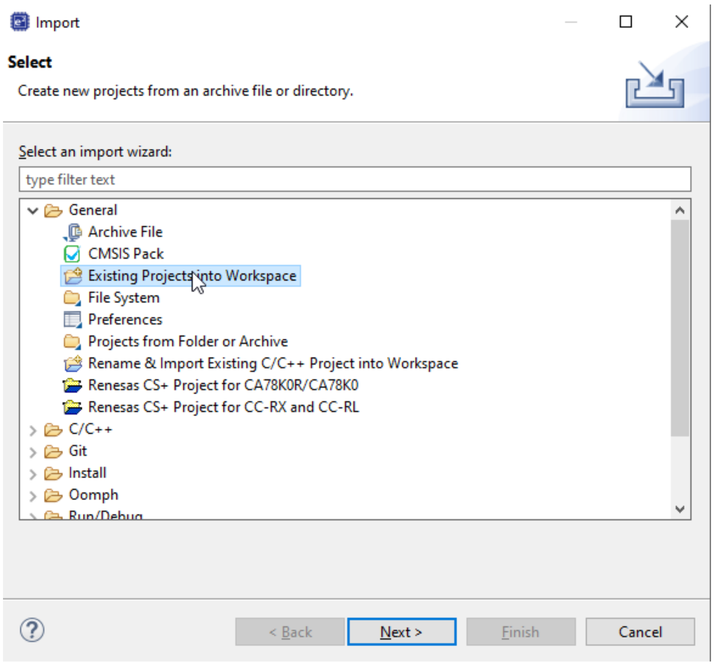

Figure 5. Selection of the Import Option

3. Select the **Select archive file** option, click the **Browse**… button and then select the demo project file (.zip). Click the **Finish** button.

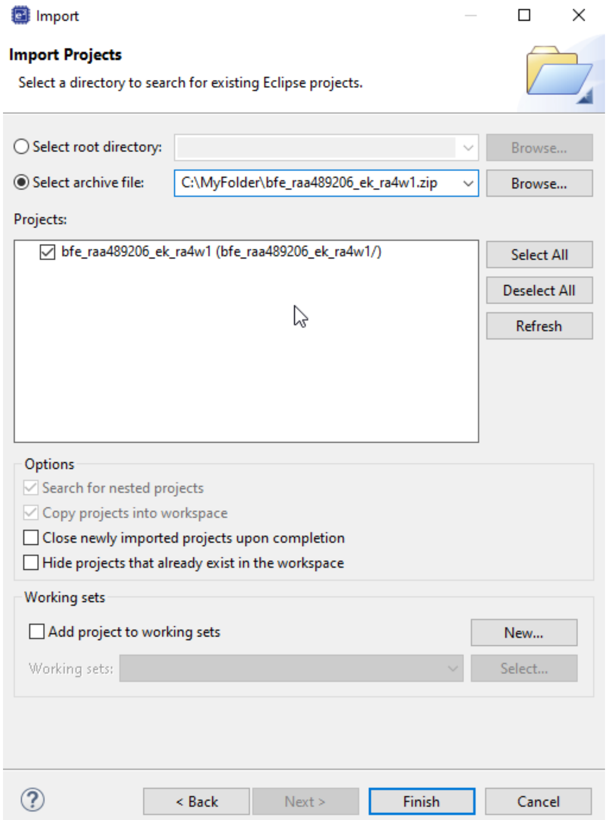

Figure 6. Import the Sample Project

4. The project is now imported into the e2 studio workspace. Figure 7 shows the imported project structure

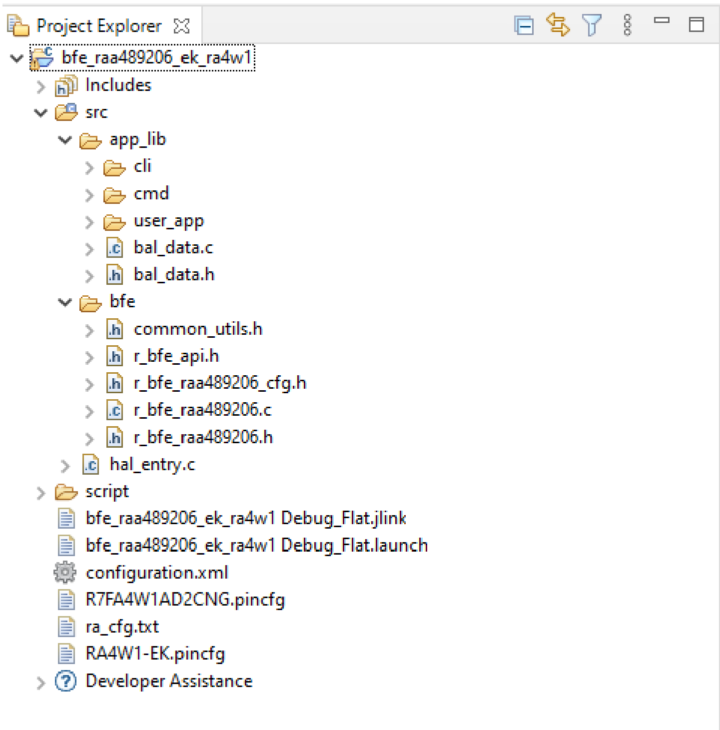

Figure 7. Structure of the Sample Project

4.3 Building and Debugging

Reference the *Renesas e2studio 2021-07 or Higher - User's Manual: Quick Start Guide (R20UT5034EJ0100).*

4.4 Demo Project Functional Description

4.4.1 BFE and EK-RA4W1 Boards

The sample project requires the RA489206 BFE device to be properly mounted on a board with the required circuitry as specified by its datasheet. It is also necessary that the BFE board allows direct connectivity between the EK-RA4W1 board and the BFE chip. Table 4 shows the pin assignments of the connections required between the EK-RA4W1 board and the BFE device.

Table 4. Pin Assignments

Signal Name	BFE Pin	MCU Port	Evaluation Kit Pin
DGND	35, 36	--	J6 (-)
MISO/SDA	44	P100	27
MOSI	43	P101	26
SCL	42	P102	25
ADD/ /CS	41	P103	24
/ALERT	40	P111	17
/WAKEUP	39	P110	16
/RESET	38	P104	23
CMS0	34	P106	21

4.5 Terminal Emulator

The CLI of the Demo project enables the interaction of the user with the MCU to command the actions performed by the BFE. To access the CLI, the user requires serial communication between the PC and EK-RA4W1. Because the EK-RA4W is equipped with a USB-Serial converter IC, this communication can be handled as a COM port by a terminal emulator such as Tera Term. Table 5 shows the terminal setup for the project CLI.

Table 5. Settings of the Terminal

Parameter	Value
New Line (Receive)	LF
New Line (Transmit)	CR
Terminal Mode	VT100
Baud Rate	115200
Data Bits	8 bits
Parity	None
Stop Bits	1 bit
Flow Control	None

4.6 Use of Command-Line Interface (CLI)

The CLI is a text-based interactive access to command the execution of MCU routines that interact directly with the BFE. This section provides general guidelines on the use and features of the CLI. Sample Battery Management System details the set of available commands, parameters, and examples.

When the EK-RAW4W1 is powered on, the terminal emulator program shows the CLI prompt raa489206 indicating readiness to accept commands. Figure 8 shows the initial CLI prompt.

Figure 8. Initial CLI Prompt when MCU is Powered On

CLI commands have the following syntax:

```
[command-group] [sub-command] <value> <option> [LF or CR]
```

Command-group and sub-command are mandatory fields, whereas value and option (single character preceded by the hyphen minus) are optional parameters. Figure 9 shows some examples of command executions using the CLI. *Note:* Successful executions of commands produce the string [OK], whereas wrong or unsuccessful executions produce the string [ERROR] and its corresponding description.

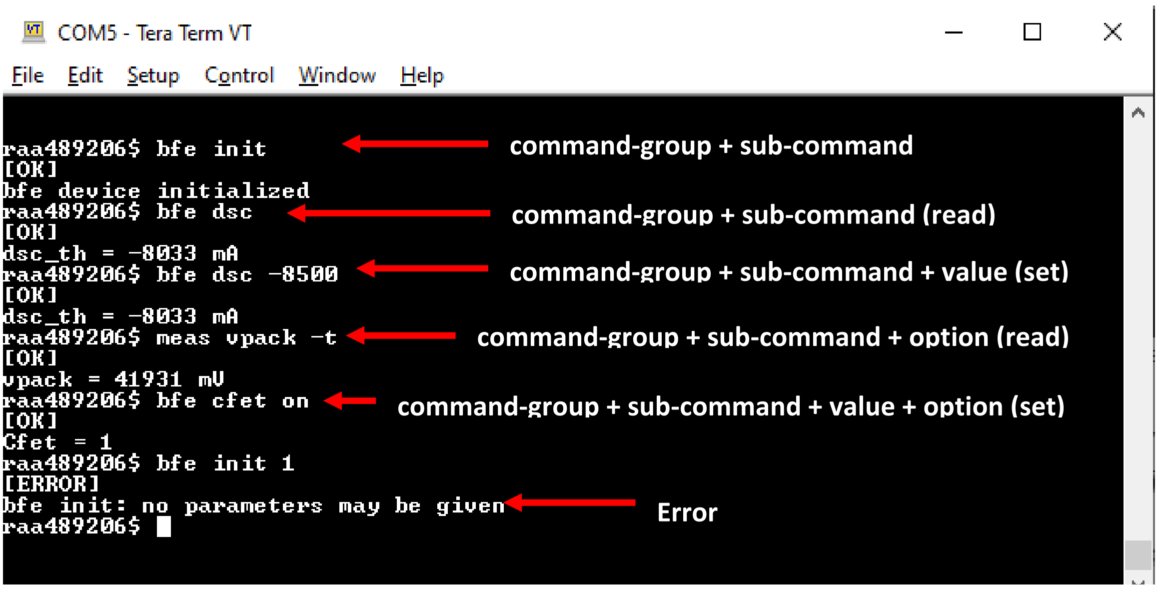

Figure 9. Examples of CLI Commands and their Syntax

The CLI includes Command-line completion. This feature enables the CLI to automatically fill partially typed commands. To use this feature, type the first few characters of a command, then press the **Tab** key. The CLI either completes the command or shows the commands that match the beginning of the typed characters. When the Tab key is pressed before typing any character, the CLI list all available commands or subcommands as a help feature. All commands include the implicit subcommand help, which displays a short description of the command use and the action it executes. Figure 10 depicts examples for the use of command completion and help features.

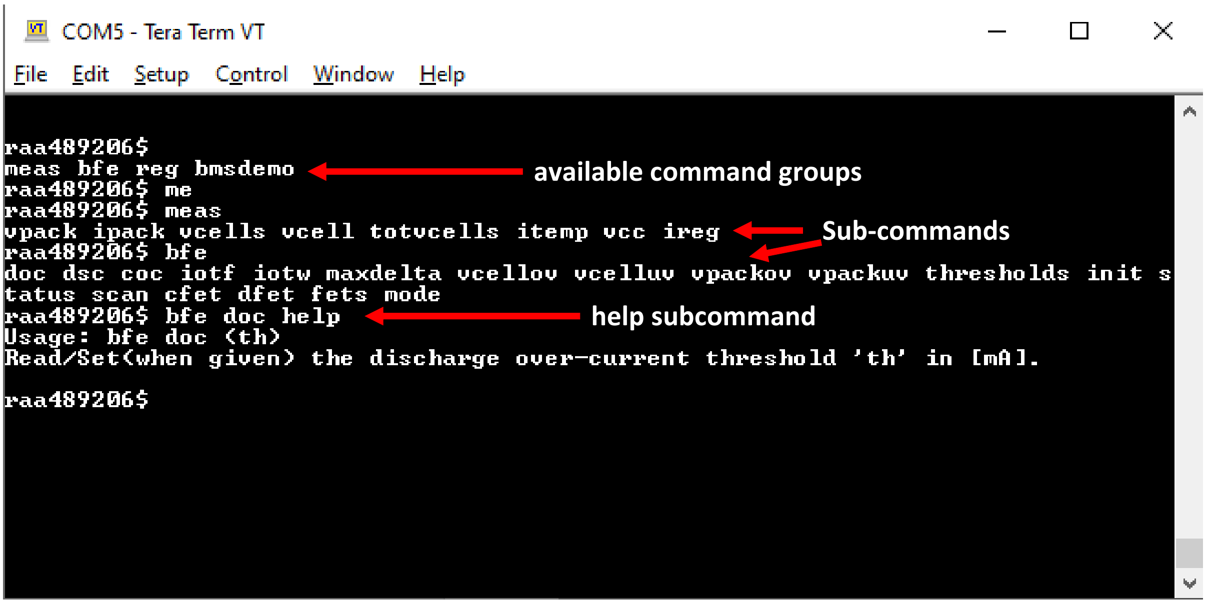

Figure 10. Command-Line Completion and Help Sub-command Features

5. Demo Project Implementation

This sample code addresses the implementation of routines and functions that enable an MCU to interact with the RA489206 BFE. This section focuses on the description of the code that implements the BFE Abstraction Layer (BAL) and the Sample BMS Application. The CLI is a user interface whose implementation details are out of the scope of this document.

5.1 FSP Architecture

BAL is a software abstraction layer designed to enable applications to use BFE features without dealing with low-level implementations. To achieve this, BAL implements the modular architecture of FSP. According to this architecture, applications are composed of modules that provide and require functionalities. Figure 11 shows the concept of the module of FSP.

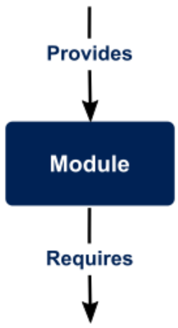

Figure 11. Module Concept of FSP

The modules interact and collaborate matching the required and provided functionalities. This functionality-based approach allows matching modules to form a layered structure with top-level modules and their dependencies. This structure is called Stack. Figure 12 shows an example of three modules (including an application) whose required and provided functionalities match, so they form a 3-level stack.

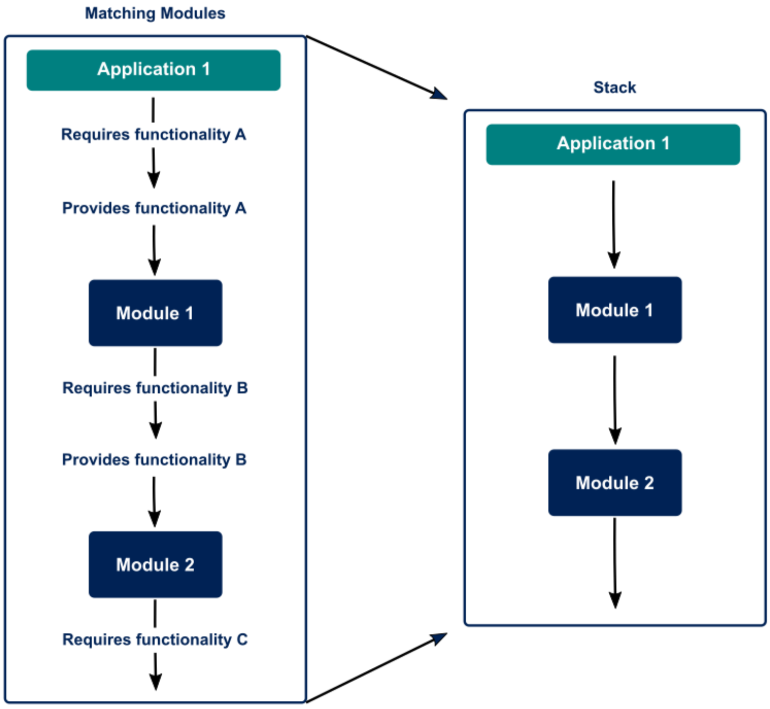

Figure 12. Example of Matching Modules that Form a Stack

The functionality provided by a module can match the functionality required by multiple modules. So having a layered structure allows sharing the code among several modules simultaneously. Figure 13 illustrates an application that can send and receive data over three different driver-driven communication interfaces: Serial Peripheral Interface (SPI), Universal Asynchronous Receiver-Transmitter (UART), and Inter-Integrated Circuit (I2C). All three drivers require the transfer functionality provided by the Data Transfer Controller (DTC), so they can share the same module.

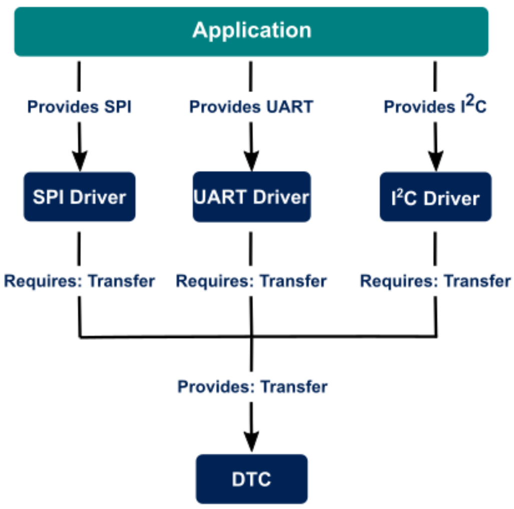

Figure 13. Example of a Providing Module (DTC) Matching the Requirement of Multiple Modules

On the other hand, multiple modules can provide a common functionality. In the scenario previously described, drivers differ in the implementation of several items such as their specific protocol, data rate, and physical layer; however, they all provide the common functionality data communication. Therefore, a middleware module (layer) providing the functionality data communication functionality can be added between the application and the drivers. Figure 14 shows the stack of the data communication scenario when a middleware layer is added to provide the common functionality.

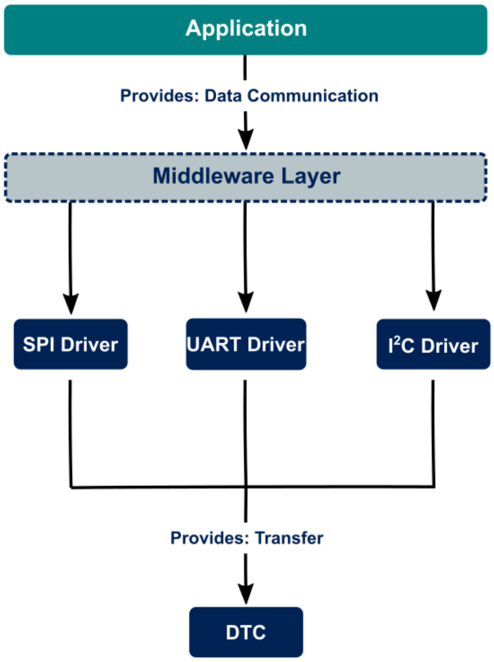

Figure 14. Redefinition of the Communication Data Scenario Adding a Middleware Layer

The advantage of adding the middleware layer to the stack is that the application does not have to deal directly with configuration, interruption routines, and other low-level details of each driver. Conversely, the application has access to a common interface that provides the required functionality and the option to select one of the three interfaces to transfer/receive data. Since the application is not driver-dependent anymore, it can transfer/receive data regardless of the communication interface, the peripheral availability, or moreover, the MCU. The BAL in this sample code follows the modular architecture of FSP to provide a middleware layer for BFEs features and implements the functionalities of the RAA489206 BFE.

5.2 BAL Implementation

At the architecture level, modules provide their functionalities using interfaces, which can be thought of as contracts between the module providing a functionality and the module requiring it. FSP specifies that interfaces are declared in header files with the naming convention *_api.h. The declaration of the BAL interface can be found in the header file: src/bfe/r_bfe_api.h (see the project directory structure in Table 1). On the other hand, interface implementations referred to as instances, are c files containing the function definitions and code bodies that provide the functionalities declared by the interface. This sample code implements the interface for the RAA489206 BFE, and its code can be found in the file src/bfe/r_bfe_ra489206.c. The following sections describe the general structure and components of the BFE interface and the details of its implementation for the RAA489206 BFE.

5.2.1 BAL Interface

FSP interfaces must include at least three data structures: a configuration structure, an API structure, and an instance structure. In BAL, these structures correspond to BFE configuration, BFE API, and BFE instance.

5.2.1.1 BFE Configuration Structure

The configuration structure is an input into the module used for setting up the interface. It contains configuration settings and parameters the module can reference at runtime to determine the functional behavior of the implementation. BAL defines the BFE configuration structure as the data type typedef struct st_bfe_cfg. The definition of the configuration struct intends to collect settings common to most of the BFEs. Table 6 describes its members.

Table 6. Members of the BFE Configuration Structure

Typedef struct st_bfe_cfg		
Member	**Type**	**Description**
shunt_resistor	float	Value [Ω] of the shunt resistor
max_cells	uint8_t	Maximum number of supported cells
min_cells	uint8	Minimum number of supported cells
cell_select	uint32_t	Selected or active cells
peripheral_type	e_bfe_comm_peripheral_t	Type of communication interface
read_after_write	bool	Enabler of the feature read-after-write (RAW)
num_read_after_write	uint8_t	Number of RAW tries before write operation fails
driver_conf	e_bfe_driver_configuration_t	Enumeration indicating the position of high-power drivers: high-side or low-side configuration
power_conf	e_bfe_fet_configuration_t	Enumeration indicating the power FET configuration: Series or parallel
*p_extend	void	Pointer to BFE-specific configuration settings

5.2.2 BFE API Structure

The BFE API declares the signature of the functions that modules implementing the interface, namely instances, MUST implement to provide the intended features of the module. The API allows modules to be swapped in and out by instances that implement the same interface. Therefore, applications such as BMS can use potentially any BFE instance that implements the BLA interface because the API ensures the provided functionalities are the same. In this sample code, the instance of the BFE API structure corresponds to the RAA489206 BFE. Applications using its instance could swap it with any other BFE instance that implements the structures defined by the BLA interface.

Table 7 describes the members of the BFE API structure, which declares the functionalities provided by BLA. All members are pointers to functions, return values defined by the enumeration **e_bfe_err_t** to indicate error or success of the execution, and require pointers to variables to return values. Some of the variable types containing returned values are of type void. The BFE instance must then define its content structure, (for example, structure fields) according to the BFE features. These generic types are referred to as instance-defined parameters. The parameter of data type **st_bfe_ctrl_t** is a control structure. It is common to all API functions and is an instance-defined parameter that works as a unique identifier of the BFE instance. Details about the control structure definition and members are given in RAA489206 BFE Instance Implementation.

Table 7. BFE API Structure and its Fields

Typedef struct st_bfe_cfg		
Member	**Parameters**	**Description**
*p_init	st_bfe_ctrl_t * p_ctrl st_bfe_cfg_t const * const p_cfg	Sets the control structure with the settings given by the configuration structure and initialize the BFE device.
*p_reset	st_bfe_ctrl_t * ctrl e_bfe_reset_type_t reset	Resets the BFE performing the reset type specified by the enumeration reset
*p_startSystemScan	st_bfe_ctrl_t * ctrl	Initiates a complete system scan
*p_startContinuousScan	st_bfe_ctrl_t * p_ctrl bfe_cs_configuration_t * const p_sc	Sets the scan options specified by the instance-defined parameter **p_sc** and start continuous scan operation
*p_stopContinuousScan	st_bfe_ctrl_t * p_ctrl	Stops continuous scan operation
*p_isBusy	st_bfe_ctrl_t * p_ctrl bool * p_busy	Indicates the current availability of the device by returning ***p_busy = true** if the device is currently executing any task, or ***p_busy = false**, otherwise.
*p_readStatus	st_bfe_ctrl_t * p_ctrl bfe_status_t * const p_status	Returns the BFE status by storing status indicators/registers in the instance-defined variable pointed by **p_status**
*p_readMode	st_bfe_ctrl_t * p_ctrl bfe_mode_t * const p_mode	Reads the current BFE mode and return it in the variable pointed by **p_mode**.
*p_clearAllFaults	st_bfe_ctrl_t * p_ctrl	Clears all current BFE faults
*p_clearFault	st_bfe_ctrl_t * p_ctrl const bfe_status_t * const p_status	Clears the fault indicators specified by the instance-defined variable pointed by **p_status**
*p_readVpack	st_bfe_ctrl_t * p_ctrl float * const p_value bool trigger	Reads the pack voltage and return its value in mV in the variable pointed by **p_value**. The boolean trigger indicates whether a measurement must be executed (trigger = true) before reading the value.
*p_readVcells	st_bfe_ctrl_t * p_ctrl bfe_vcells_measurements_t * const p_value bool trigger	Reads cell voltages and returns their values in mV in the instance-defined variable pointed by **p_value**. The boolean trigger indicates whether measurements must be executed (trigger = true) before reading the value.
*p_readIpack	st_bfe_ctrl_t * p_ctrl float * const p_value bool trigger	Reads the pack current and returns its value in mA (calculated from the shunt resistor) in the variable pointed by **p_value**. The boolean trigger indicates whether a measurement must be executed (trigger = true) before reading the value.

Table 7. BFE API Structure and its Fields (Cont.)

Typedef struct st_bfe_cfg		
Member	**Parameters**	**Description**
*p_readOther	st_bfe_ctrl_t * p_ctrl bfe_other_measurements * const p_value bool trigger	Reads other measurements (for example, V_{CC}) and returns their values in the instance-defined variable pointed by **p_value**. The boolean trigger indicates whether measurements must be executed (trigger = true) before reading the value.
*p_readAuxExt	st_bfe_ctrl_t * p_ctrl bfe_auxext_measurements_t * const p_value bool trigger	Reads the auxiliary/extern measurements and returns their values in mV in the instance-defined variable pointed by **p_value**. The trigger parameter indicates whether measurements must be executed (trigger = true) before reading the value.
*p_readTemperature	st_bfe_ctrl_t * p_ctrl bfe_temperature_measurements_t * const p_value bool trigger	Reads the temperatures measured by the BFE and returns their values in °C in the instance-defined variable pointed by **p_value**. The trigger parameter indicates whether measurements must be executed (trigger = true) before reading the value.
*p_turnChargePumpOn	st_bfe_ctrl_t * p_ctrl	Turns BFE charge pump on
*p_ turnChargePumpOff	st_bfe_ctrl_t * p_ctrl	Turns BFE charge pump off
*p_turnDfetOn	st_bfe_ctrl_t * p_ctrl	Turns discharge FET (DFET) on.
*p_turnDfetOff	st_bfe_ctrl_t * p_ctrl	Turns discharge FET (DFET) off.
*p_turnCfetOn	st_bfe_ctrl_t * p_ctrl	Turns the charge FET (CFET) on.
*p_turnCfetOff	st_bfe_ctrl_t * p_ctrl	Turns the charge FET (CFET) off.
*p_turnDfetOnCfetOn	st_bfe_ctrl_t * p_ctrl	Turns discharge and charges FETs on.
*p_turnDfetOffCfetOn	st_bfe_ctrl_t * p_ctrl	Turns DFET off and CFET on.
*p_turnDfetOnCfetOff	st_bfe_ctrl_t * p_ctrl	Turns DFET on and CFET off.
*p_turnDfetOffCfetOff	st_bfe_ctrl_t * p_ctrl	Turns discharge and charges FETs off.
*p_setMode	st_bfe_ctrl_t * p_ctrl e_bfe_mode_t mode	Sets BFE mode to the modes specifies by the enumeration **e_bfe_mode_t** mode
*p_setAlerts	st_bfe_ctrl_t * p_ctrl bfe_alerts_masks_t masks	Sets the BFE events that notify the MCU by any means, such as asserting an external pin. This function unmasks the events specified by the instance-defined parameter masks
*p_setDoc	st_bfe_ctrl_t * p_ctrl float current_th_ma	Sets the discharge overcurrent threshold to the value **current_th_ma** in mA

Table 7. BFE API Structure and its Fields (Cont.)

Typedef struct st_bfe_cfg		
Member	**Parameters**	**Description**
*p_setCoc	st_bfe_ctrl_t * p_ctrl float current_th_ma	Sets the charge overcurrent threshold to the value **current_th_ma** in mA
*p_setDsc	st_bfe_ctrl_t * p_ctrl float current_th_ma	Sets the discharge short-circuit current threshold to the value **current_th_ma** in mA
*p_setMaxVCellDelta	st_bfe_ctrl_t * p_ctrl float voltage_th_mv	Sets the maximum delta cell voltage threshold to the value **voltage_th_mv** in mV
*p_setCellUnderVoltage	st_bfe_ctrl_t * p_ctrl float voltage_th_mv	Sets the cell undervoltage threshold to the value **voltage_th_mv** in mV
*p_setCellOverVoltage	st_bfe_ctrl_t * p_ctrl float voltage_th_mv	Sets the cell overvoltage threshold to the value **voltage_th_mv** in mV
*p_setVpackUnderVoltage	st_bfe_ctrl_t * p_ctrl float voltage_th_mv	Sets the pack undervoltage threshold to the value **voltage_th_mv** in mV
*p_setVpackOverVoltage	st_bfe_ctrl_t * p_ctrl float voltage_th_mv	Sets the pack overvoltage threshold to the value **voltage_th_mv** in mV
*p_setInternalOvertempWarn	st_bfe_ctrl_t * p_ctrl float temp_th	Sets the internal over-temperature warning threshold to the value **temp_th** in °C
*p_setInternalOvertempFault	st_bfe_ctrl_t * p_ctrl float temp_th	Sets the internal over-temperature fault threshold to the value **temp_th** in °C
* p_setVoltageEndOfCharge	st_bfe_ctrl_t * const p_ctrl float veoc_mv	Sets the end-of-charge voltage to the value **veoc_mv** in mV.
* p_setCurrentEndOfCharge	st_bfe_ctrl_t * const p_ctrl float ieoc_ma	Sets the end-of-charge current to the value **ieoc_ma** in mA.
*p_readAlerts	st_bfe_ctrl_t * p_ctrl bfe_alerts_masks_t *const p_alerts	Reads and returns indicators of the violation of the thresholds monitored by the BFE, storing their values in the instance-defined variable pointed by **p_alerts**
*p_readDoc	st_bfe_ctrl_t * p_ctrl float *p_current_th_ma	Reads the discharge overcurrent threshold and returns its value in mA in the variable pointed by **p_current_th_ma**
*p_readCoc	st_bfe_ctrl_t * p_ctrl float *p_current_th_ma	Reads the charge overcurrent threshold and returns its value in mA in the variable pointed by **p_current_th_ma**
*p_readDsc	st_bfe_ctrl_t * p_ctrl float *p_current_th_ma	Reads the discharge short-circuit current threshold and returns its value in mA in the variable pointed by **p_current_th_ma**
*p_readMaxVCellDelta	st_bfe_ctrl_t * p_ctrl float *p_voltage_th_mv	Reads the maximum cell voltage delta threshold and returns its value in mV in the variable pointed by **p_voltage_th_mv**

Table 7. BFE API Structure and its Fields (Cont.)

Typedef struct st_bfe_cfg		
Member	**Parameters**	**Description**
*p_readCellUnderVoltage	st_bfe_ctrl_t * p_ctrl float *p_voltage_th_mv	Reads the cell undervoltage threshold and returns its in mV value in the variable pointed by **p_voltage_th_mv**
*p_readCellOverVoltage	st_bfe_ctrl_t * p_ctrl float *p_voltage_th_mv	Reads the cell overvoltage threshold and returns its value in mV in the variable pointed by **p_voltage_th_mv**
*p_readVpackUnderVoltage	st_bfe_ctrl_t * p_ctrl float *p_voltage_th_mv	Reads the pack undervoltage threshold and returns its value in mV in the variable pointed by **p_voltage_th_mv**
*p_readVpackOverVoltage	st_bfe_ctrl_t * p_ctrl float *p_voltage_th_mv	Reads the pack overvoltage threshold and returns its value in mV the variable pointed by **p_voltage_th_mv**
*p_readInternalOvertempWarn	st_bfe_ctrl_t * p_ctrl float *p_temp_th	Reads the internal over-temperature warning threshold and returns its value in °C in the variable pointed by **p_temp_th**
*p_readInternalOvertempFault	st_bfe_ctrl_t * p_ctrl float *p_temp_th	Reads the internal over-temperature fault threshold and returns its value in °C in the variable pointed by **p_temp_th**
* p_readVoltageEndOfCharge	st_bfe_ctrl_t * const p_ctrl float * p_veoc_th	Reads the end-of-charge voltage and returns its value in mV in the variable pointed by **p_veoc_th**.
* p_readCurrentEndOfCharge	st_bfe_ctrl_t * const p_ctrl float * p_ieoc_th	Reads the end-of-charge current and returns its value in mA in the variable pointed by **p_ieoc_th**.
*p_configLowPowerMode	st_bfe_ctrl_t * p_ctrl bfe_lpm_cfg_t *p_lpm_cfg	Configures BFE Low Power Mode settings according to the instance-defined variable **p_lpm_cfg**
*p_startLowPowerMode	st_bfe_ctrl_t * p_ctrl	Sets the device to low power mode
*p_getDieInformation	st_bfe_ctrl_t * p_ctrl st_bfe_information_t *p_info	Reads the die information and returns its value in the structure pointed by **p_info**, which contains version, manufacturing id, device id and nickname of the BFE
*p_readRegister	st_bfe_ctrl_t * p_ctrl st_bfe_register_t *p_register	Reads the BFE register specified by the structure pointed by **p_register**. The structure contains: address, type (R/W), current value and reset/default value.

Table 7. BFE API Structure and its Fields (Cont.)

Typedef struct st_bfe_cfg		
Member	**Parameters**	**Description**
*p_writeRegister	st_bfe_ctrl_t * p_ctrl st_bfe_register_t *p_register	Writes the BFE register specified by the structure pointed by **p_register**. The structure contains: address, type (R/W), current value and reset/default value.
*p_readAllRegisters	st_bfe_ctrl_t * p_ctrl	Reads all BFE registers. Renesas recommends storing read values in global variables to keep an image of BFE registers accessible any code.

5.2.3 BFE Interface Instance Structure

The instance structure encapsulates all the structures necessary to use a module implementation:

- Pointer to the instance API structure
- Pointer to the configuration structure
- Pointer to the control structure

In BLA, the interface instance structure is defined in src/bfe/r_bfe_api.h as:

```
typedef struct st_bfe_instance
{
    st_bfe_ctrl_t       * p_ctrl; ///< Pointer to the control structure
    const st_bfe_cfg_t  * p_cfg; ///< Pointer to the configuration structure
    const st_bfe_api_t  * p_api; ///< Pointer to the API structure
} st_bfe_instance_t;
```

5.3 RAA489206 BFE Instance Implementation

The BFE instance and its structures define the contract and features common to most of the BFEs, the RAA489206 BFE instance contains the actual code implementation of BFE functionalities. Figure 15 shows the main software components, structures, and files of the BFE interface and instance implementations. This section details these components with focusing on the source file **r_bfe_raa489206**.c, because it contains the routines, sequences, and logic that interact with the RAA489206 BFE.

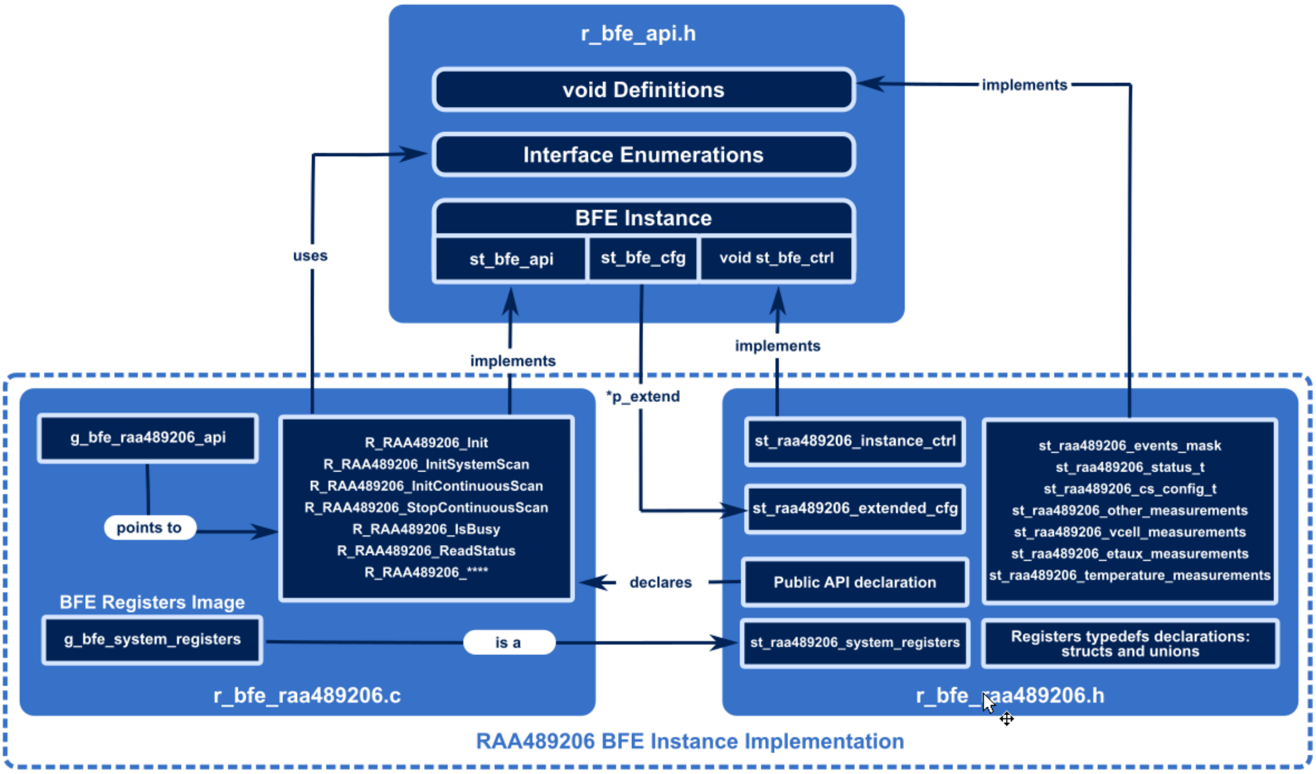

Figure 15. Main Software Components of the RAA489206 BFE Instance Implementation

5.3.1 Header File r_bfe_raa489206.h

This BFE header file contains the following relevant declarations:

- **Interface control:** Structure that stores references to system register structure, configuration of pins driven by API functions and pointers to functions that read and write registers.

```
typedef struct st_raa489206_instance_ctrl
{
    u_raa489206_cells_select_t                    cells_select;
    const spi_instance_t                        * p_spi;          ///< spi instance
    const i2c_master_instance_t                 * p_i2c;          ///< i2c instance
    const ioport_instance_t                   * p_ioport;        ///< ioport instance
    bsp_io_port_pin_t                           reset_pin;        ///< reset pin port
    bsp_io_port_pin_t                            cms0_pin;         ///< CMS0 pin port
    bsp_io_port_pin_t                          ss_pin;          ///< Slave Selection
pin if needed (SPI on SCI)
    bsp_io_port_pin_t                             i2c_add_sel_pin;  ///< I2C address
selector pin
    bsp_io_port_pin_t                             alert_pin;        ///< alert pin
    bsp_io_port_pin_t                           wakeup_pin;       ///< Wakeup pin port
```

```
    bool                                        init;             ///< Indicates
whether the init() API has been successfully called.
    bool                                     use_crc;          ///< use crc feature
and commands
    const st_bfe_cfg_t                        * p_cfg;            ///< Pointer to
configuration entity
    st_raa489206_system_registers_t           * p_regs;           ///< raa489206
system registers
    e_bfe_err_t (* p_writeRegisterValues)(uint8_t address, uint8_t const * p_values,
uint16_t num_values, bool use_crc); ///< Pointer to  write function
    e_bfe_err_t (* p_readRegisterValues)(uint8_t address, uint8_t * p_values,
uint16_t num_values, bool use_crc); ///< pointer to read function
    bool read_after_write;
} st_raa489206_instance_ctrl_t;
```

- **Extended configuration:** Structure containing the configuration settings that are particular of the RAA489206 BFE.

```
typedef struct st_raa489206_extended_cfg
{
    float                          reg_resistor;                 ///< Sense resistor
between emitter and Vdd pins used to measure Ireg
    bool                           use_crc;                      ///< Use cyclic
redundancy check
    bool                          enable_fuse_blow;           ///< enable fuse blow
option
    bool                         enable_chr_pump_init;       ///< indicates whether
the charge pump shall be enable during initialization
    const spi_instance_t           * p_spi;                      ///< spi instance
    const i2c_master_instance_t    * p_i2c;                      ///< i2c instance
    const ioport_instance_t        * p_ioport;                  ///< ioport instance
    bsp_io_port_pin_t              reset_pin;                    ///< reset pin port
    bsp_io_port_pin_t              cms0_pin;                     ///< CMS0 pin port
    bsp_io_port_pin_t             ss_pin;                      ///< Slave Selection
pin
    bsp_io_port_pin_t              i2c_add_sel_pin;              ///< I2C address
selector pin
    bsp_io_port_pin_t              alert_pin;                    ///< alert pin
    bsp_io_port_pin_t             wakeup_pin;                   ///< Wakeup pin port
    uint8_t                      device_spi_add_no_crc;      ///< SPI slave address
to read data
    uint8_t                      device_spi_add_with_crc;    ///< SPI slave address
to read data using crc
    uint8_t                        device_i2c_add;             ///< I2C slave address

} st_raa489206_extended_cfg_t;
```

- **System registers:** Structure that stores all BFE registers referenced by API functions and the control structure. It represents the bank of registers of the BFE.

```
typedef struct st_raa489206_system_registers
{
    //Device details
    st_bfe_register_t die_information;
    //Global IC Controls
    st_bfe_register_t global_operation;
    //Vcell and Ipack Controls
    st_bfe_register_t vcell_operation;
    st_bfe_register_t ipack_operation;
    st_bfe_register_t cell_select;
    . . . . .

    st_bfe_register_t etaux_faults_mask;
    st_bfe_register_t other_faults_mask;
    st_bfe_register_t cb_status_masks;
    st_bfe_register_t status_masks;
    st_bfe_register_t open_wire_mask;
    //System Operation
    st_bfe_register_t scan_operation;
} st_raa489206_system_registers_t;
```

- **API functions declarations:** group of declarations of the functions implementing the BFE API defined by the member described in Table 6.

```
e_bfe_err_t R_RAA489206_Init (st_bfe_ctrl_t * const p_ctrl, const st_bfe_cfg_t  *
const p_cfg );

e_bfe_err_t R_RAA489206_Reset (st_bfe_ctrl_t * const p_ctrl, e_bfe_reset_type_t
reset_type);

e_bfe_err_t R_RAA489206_InitSystemScan (st_bfe_ctrl_t * const p_ctrl);

e_bfe_err_t R_RAA489206_InitContinuousScan (st_bfe_ctrl_t * const p_ctrl, const
bfe_cs_configuration_t * const st_cs_config);

e_bfe_err_t R_RAA489206_StopContinuousScan (st_bfe_ctrl_t * const p_ctrl);

e_bfe_err_t R_RAA489206_IsBusy (st_bfe_ctrl_t * const p_ctrl, bool * p_is_busy);

e_bfe_err_t R_RAA489206_ReadStatus (st_bfe_ctrl_t * const p_ctrl, bfe_status_t *
const p_bfe_status);

e_bfe_err_t R_RAA489206_ClearAllFaults (st_bfe_ctrl_t * const p_ctrl);


. . . . .

e_bfe_err_t R_RAA489206_ConfigLowPowerMode (st_bfe_ctrl_t * const p_ctrl,
bfe_lpm_cfg_t * const p_lpm_options);

e_bfe_err_t R_RAA489206_StartLowPowerMode (st_bfe_ctrl_t * const p_ctrl);

e_bfe_err_t R_RAA489206_GetDieInformation (st_bfe_ctrl_t * const p_ctrl,
st_bfe_information_t * p_information);

e_bfe_err_t R_RAA489206_ReadRegister (st_bfe_ctrl_t * const p_ctrl,
st_bfe_register_t * const p_bfe_register);

e_bfe_err_t R_RAA489206_ReadAllRegisters (st_bfe_ctrl_t * const p_ctrl);

e_bfe_err_t R_RAA489206_WriteRegister (st_bfe_ctrl_t * const p_ctrl, const
st_bfe_register_t * const bfe_register);
```

- **Declarations of void structures of the BFE instance:** Group of instance-defined structures that are used by API functions to configure features and return values.

```
/* BFE events masks*/
typedef struct
{
    u_raa489206_prifault_masks_t priority_masks;
    u_raa489206_etauxfault_masks_t etaux_masks;
    u_raa489206_otherfault_masks_t other_masks;
    u_raa489206_cbstatus_masks_t cb_masks;
    u_raa489206_status_masks_t status_masks;
    u_raa489206_ow_masks_t ow_masks;
    u_raa489206_vcell_fault_delay_t fault_delays;
    raa489206_dsc_delay_register_t dsc_delay;
    u_raa489206_oc_delay_t oc_delay;
    e_raa489206_ld_delay_t ld_delay;
    bool idir_delay;
} st_raa489206_events_masks_t;

/* BFE status: reports the status of the BFE*/
typedef struct
{
    e_raa489206_fault_register_t fault_register_type;
    u_raa489206_prifault_register_t priority_status;
    u_raa489206_etauxfault_reg_t etaux_status;
    u_raa489206_otherfault_reg_t other_status;
    u_raa489206_cbstatus_reg_t cb_status;
    u_raa489206_status_reg_t general_status;
    u_raa489206_owstatus_reg_t ow_status;
}st_raa489206_status_t;

/*RAA489206 continuous scan configuration*/
typedef struct
{
    uint8_t VCELL_EN                          : 1; /*Vcell measurement*/
    e_raa489206_vcell_avg_t VCELL_AVG         : 3; /*Number of samples to average*/
} st_raa489206_cs_cfg_vcell_t;

typedef struct
{
    uint8_t IPACK_EN                          : 1; /*I pack measurement*/
    e_raa489206_ipack_avg_t IPACK_AVG         : 3; /*Number of samples to average*/
} st_raa489206_cs_cfg_ipack_t;

typedef struct
{
    uint8_t OW_EN                             : 1; /*I pack measurement*/
    e_raa489206_ow_period_t  OW_UPDATE        : 2; /*How often the open-wire test is
executed*/
} st_raa489206_cs_cfg_open_wire_t;

typedef struct
{
uint8_t VBAT_EN                                : 1; /*enable vpack measurement*/
```

```
    uint8_t ITEMP_EN                             : 1; /*enable internal temperature
measurement*/
    e_raa489206_other_avg_t OTHER_AVG             : 3; /*number of samples to
average*/
    e_raa489206_otherupdate_period_t OTHER_UPDATE : 2; /* number of scans required
before ETAUX,
                                                        Vbat,Vcc, Ireg and int. temp
are made*/
} st_raa489206_cs_cfg_other_t;

typedef struct
{
    e_raa489206_etaux_enable_t ETAUX_EN                    : 2; /*External/auxiliar
measurements*/
    e_raa489206_etaux_avg_t ETAUX_AVG                      : 3; /*Number of samples
to average*/
} st_raa489206_cs_cfg_etaux_t;

typedef struct
{
    st_raa489206_cs_cfg_vcell_t vcell_cfg;
    st_raa489206_cs_cfg_ipack_t ipack_cfg;
    st_raa489206_cs_cfg_open_wire_t ow_cfg;
    st_raa489206_cs_cfg_other_t other_cfg;
    st_raa489206_cs_cfg_etaux_t etaux_cfg;
    e_raa489206_scan_delay_t scan_delay;
} st_raa489206_cs_config_t;

typedef struct
{
    float veoc_th;
    float ieoc_th;
    u_raa489206_cb_op_register_t cb_operation;
    float min_delta_th;
    float cbmax_th;
    float cbmin_th;
    e_raa489206_cb_timer_unit_t cb_timer_unit;
    uint8_t cbon_time;
    uint8_t cboff_time;
} st_raa489206_cb_config;


typedef union
{
    float vector[3];

    struct
    {
        float vcc;
        float ireg;
        float vtemp;
    } measurements;
} u_raa489206_other_measurements_t;
```

```
typedef union
{
    float vector[17];

    struct{
        float cell1;
        float cell2;
        float cell3;
        float cell4;
        float cell5;
        float cell6;
        float cell7;
        float cell8;
        float cell9;
        float cell10;
        float cell11;
        float cell12;
        float cell13;
        float cell14;
        float cell15;
        float cell16;
        float vcell_max_delta;
} measurements;
} u_raa489206_vcell_measurements_t;

typedef union
{
    float vector[2];

    struct
    {
        float xt0_aux0;
        float xt1_aux1;
    } measurements;
} u_raa489206_etaux_measurements_t;

typedef float st_raa489206_temperature_measurements_t;

typedef struct
{
    e_raa489206_lpm_timer_t LPM_TIMER         : 3; /*Low power mode timer*/
    e_raa489206_lpm_regulator_t REG_TYPE      : 1; /*Regulator type*/
    uint8_t COMTO_EN                          : 1; /*Enable communication time out*/
    e_raa489206_comm_timeout_t COM_TO         : 2; /*Communication timeout*/
    uint8_t LDLP                              : 1; /*Load detect while in low power
mode*/
} st_raa4892206_lpm_cfg_t;
```

- **Register typedef declarations:** These declarations form a complete library of all RAA489206 registers and their corresponding bits. The library is composed of unions and structures type definitions named according to the BFE datasheet. Using the type definitions contained in this library allows manipulating whole register values, specific bits-fields within the register, or individual bits using the names defined in the datasheet. This library also contains declarations of enumerations to set bits-fields, which aim at ensuring correctness when setting/reading values and removing uncertainty when you must decide what values to set. The following are some examples of this library declarations.

```
/* 0x09 Fault Delay register*/
typedef union
{
    uint8_t value;

    struct
    {
        uint8_t VCELL_FAULT_DELAY        : 4; /*0x09.1...0x09.3 enables scans delay 
for
                                              Vcell OV and UV faults */
       uint8_t ETAUX_FAULT            : 1; /*0x09.4 enables 3-scan delay for ETAUX
                                              voltage threshold*/
        uint8_t OTHER_FAULT_DELAY        : 1; /*0x09.5 enables 3-scans delay for
                                            Vcc, Vtmpf, Ireg1, Ireg2, Vbovf, Vbuvf,
IOTF,
                                               IOTW faults*/
        uint8_t DELTA_VCELL_FAULT_DELAY  : 1; /*0x09.6 enables 3-scans delay for
delta
                                               vcell fault*/
        uint8_t AUX_XTN_PULLUP           : 1; /*0x09.7 when set to 1, internal
resistors are connected bwtween AUX0/1 and Vcc*/
    } value_b;
} u_raa489206_vcell_fault_delay_t;
/* 0x03 Ipack operation*/
typedef union
{
    uint8_t value;

    struct
    {
        uint8_t IPACK_TRIGGER           : 1;    /*0x03.0 initiates an Ipack 
measurement*/
        uint8_t IDIR_DELAY              : 1;    /*0x03.1 number of measurements to 
determine charge/discharge: 0=1, 1=3 readings*/
        uint8_t IPACK_AVG               : 3;    /*0x03.2...0x03.4 number of samples 
averaged before writing the result*/
        uint8_t OW_UPDATE               : 2;    /*0x03.5...0x03.6 Control how often 
the
                                                open-wire test is executed*/
        uint8_t IPACK_EN                : 1;    /*0x03.7 Set to 1 to enable Ipack
                                                 measurements*/
    } value_b;

} u_raa489206_ipackop_reg_t;
typedef enum e_raa489206_ipack_avg
{
```

```
    RAA489206_IPACK_AVG_1_SAMPLE           = 0x00,
    RAA489206_IPACK_AVG_2_SAMPLES          = 0x01,
    RAA489206_IPACK_AVG_4_SAMPLES          = 0x02,
    RAA489206_IPACK_AVG_8_SAMPLES          = 0x03,
    RAA489206_IPACK_AVG_16_SAMPLES         = 0x04,
    RAA489206_IPACK_AVG_32_SAMPLES         = 0x05,
    RAA489206_IPACK_AVG_64_SAMPLES         = 0x06,
    RAA489206_IPACK_AVG_128_SAMPLES        = 0x07,
} e_raa489206_ipack_avg_t;
typedef enum e_raa489206_ow_period
{
    RAA489206_OW_PERIOD_256_SCANS          = 0x00,
    RAA489206_OW_PERIOD_512_SCANS          = 0x01,
    RAA489206_OW_PERIOD_1024_SCANS         = 0x02,
    RAA489206_OW_PERIOD_2048_SCANS         = 0x03,
} e_raa489206_ow_period_t;
```

5.3.2 Source File r_bfe_raa489206.c

The routines, sequences, and logic contained in this source file demonstrate the features that the RAA489206 BFE provides by implementing the BFE interface. This section highlights the most relevant code, and the recommended practices for interfacing successfully with the BFE. You can follow this documentation while reviewing the source code as the following sections explain the code in the same top-down order they are in the source code.

5.3.2.1 Global API Instantiation

After including the necessary headers, the code defines the global constant **g_bfe_raa489206_api**. This constant of type **st_bfe_api_t** is the instantiation of the API structure defined in the BFE interface. It contains pointers to the functions that implement the bodies and behavior of the functions declared by API structure fields. The functions naming adopts the convention R_<BFE>_<API_function>, where <BFE> is the BFE device for which the function is implemented (RAA489206), and <API_function> is the name of the API structure field the function corresponds to. The following are some examples:

```
const st_bfe_api_t g_bfe_raa489206_api =
{                                                BFE
 .p_init = R_RAA489206_Init,
 .p_reset = R_RAA489206_Reset,
 .p_initSystemScan = R_RAA489206_StartSystemScan,
 .p_initContinuousScan = R_RAA489206_StartContinuousScan,
 .p_stopContinuousScan = R_RAA489206_StopContinuousScan,
 .p_isBusy = R_RAA489206_IsBusy,
 .p_readStatus = R_RAA489206_ReadStatus,
 .p_clearAllFaults = R_RAA489206_ClearAllFaults,
 .p_clearFault = R_RAA489206_ClearFault,
 .p_readMode = R_RAA489206_ReadMode,
 .p_readVpack = R_RAA489206_ReadVpack,
 .p_readIpack = R_RAA489206_ReadIpack,
 .p_readVcells = R_RAA489206_ReadVcells,
 .p_readOther = R_RAA489206_ReadOther,
 .p_readAuxExt = R_RAA489206_ReadAuxExt,
 .p_readTemperature = R_RAA489206_ReadTemperature,
 .p_turnDfetOn = R_RAA489206_TurnDFetOn,
 .p_turnDfetOff = R_RAA489206_TurnDFetOff,
```

```
 .p_turnCfetOn = R_RAA489206_TurnCFetOn,
```

API pointer – Function name correspondence

```
. . .
 .p_readRegister = R_RAA489206_ReadRegister,
 .p_readAllRegisters = R_RAA489206_ReadAllRegisters,
 .p_writeRegister = R_RAA489206_WriteRegister,
};
```

5.3.3 Reset and Device Registers

As a mechanism to verify the correct initial state of the BFE, or after performing a reset, it is good practice to verify the default content of all or a set of the BFE registers. To perform this task, the source code defines static constants that contain the default values defined in the datasheet:

```
/*Reset registers values*/
static const u_raa489206_productionid_reg_t g_reset_dieinformation_register =
{.value = 0xF2};
static const raa489206_iotw_th_t g_reset_iotw_th_register = 0x51;
static const raa489206_iotf_th_t g_reset_iotf_th_register = 0x45;
static const u_raa489206_vregop_register_t g_reset_vregop_register = {.value =
0xF0};
static const u_raa489206_otherfault_reg_t g_reset_other_faults = {.value = 0x00};
static const u_raa489206_globalop_reg_t g_reset_globalop_register = {.value = 0x00};
static const u_raa489206_vcellop_reg_t g_reset_vcellop_register = {.value = 0x80};
static const u_raa489206_ipackop_reg_t g_reset_ipackop_register = {.value = 0x80};
static const u_raa489206_cells_select_t g_reset_cells_select_register = {.value =
0xFFFF};
static const u_raa489206_vcell_voltage_t g_reset_vcell_voltage = {.value = 0x0000};
static const u_raa489206_vcell_max_delta_t g_reset_vcell_max_delta = {.value =
0x0000};
static const u_raa489206_ipack_voltage_t g_reset_ipack_voltage = {.value = 0x0000};
static const u_raa489206_ipack_timer_t g_reset_ipack_timer_register = {.value =
0x00000000};
static const raa489206_vcell_ov_th_t g_reset_vcell_ov_th_register = 0xFF;
static const raa489206_vcell_uv_th_t g_reset_vcell_uv_th_register = 0x00;
static const raa489206_vcell_max_delta_th_t g_reset_vcell_max_delta_th = 0xFF;
static const u_raa489206_vcell_fault_delay_t g_reset_fault_delay_register = {.value
= 0x00};
static const raa489206_dsc_threshold_t g_reset_dsc_threshold_register = 0x0F;
static const raa489206_doc_th_register_t g_reset_doc_threshold_register = 0xFF;
static const raa489206_dsc_delay_register_t g_reset_dsc_delay_register = 0x00;
static const u_raa489206_scanop_register_t g_reset_scanop_register = {.value =
0x1B};
static const u_raa489206_pwr_fet_op_t g_reset_pwr_fet_op_register = {.value = 0x5C};
```

Register Data Type

```
.....
static const u_raa489206_cboff_timer_t g_reset_cboff_timer = {.value = 0x00};
static const raa489206_cb_min_delta_th_t g_reset_cb_min_delta_th = 0x00;
static const raa489206_cb_max_th_t g_reset_cb_max_th = 0xFF;
static const raa489206_cb_min_delta_th_t g_reset_cb_min_th = 0x00;
static const raa489206_veoc_th_t g_reset_veoc_th = 0xFF;
static const raa489206_ieoc_th_t g_reset_ieoc_th = 0x00;
```

Each reset constant is defined using the register type definition (declared in the **r_raa489206.h** header file) to which the reset value corresponds. This facilitates the comparison of their values or the bits-fields of interest.

Local images of the BFE registers are stored in the MCU as global variables to track and cross-check the state, configuration, and behavior of the BFE. In addition, functions in the source file and applications implemented in other source files can access and manipulate their content directly at any time.

```
/*Device registers which are linked to the registers bank contained in the control
entity*/
u_raa489206_productionid_reg_t g_dieinformation_register;
u_raa489206_globalop_reg_t g_globalop_register;
u_raa489206_vcellop_reg_t g_vcellop_register;
u_raa489206_ipackop_reg_t g_ipackop_register;
u_raa489206_cells_select_t g_cells_select_register;
u_raa489206_vcell_voltage_t g_vcell1_voltage;
u_raa489206_vcell_voltage_t g_vcell2_voltage;
u_raa489206_vcell_voltage_t g_vcell3_voltage;
u_raa489206_vcell_voltage_t g_vcell4_voltage;
u_raa489206_vcell_voltage_t g_vcell5_voltage;
u_raa489206_vcell_voltage_t g_vcell6_voltage;
u_raa489206_vcell_voltage_t g_vcell7_voltage;
u_raa489206_vcell_voltage_t g_vcell8_voltage;
u_raa489206_vcell_voltage_t g_vcell9_voltage;
. . . .
u_raa489206_cboff_timer_t g_cboff_timer;
raa489206_cb_min_delta_th_t g_cb_min_delta_th;
raa489206_cb_max_th_t g_cb_max_th;
raa489206_cb_min_delta_th_t g_cb_min_th;
raa489206_veoc_th_t g_veoc_th;
raa489206_ieoc_th_t g_ieoc_th;
```

5.3.4 Registers Bank

The register bank collects all BFE registers within the fields of a structure. The reasoning behind its implementation is to group all BFE registers data types into a generalized type definition. By doing so, functions that must handle BFE registers values regardless of their bit-fields composition, for instance SPI data transmission and reception, can deal with a generic data type containing registers information, such as address and size. The generic structure data type is declared as:

```
/** Generic BFE register: address= register address, type=R/R_W, p_value pointer to
the value(s), size=number of bytes, p_reset_value= pointer to the default value
(used for self-diagnosis after reset)*/
typedef struct st_bfe_register{

    uint8_t address;
    e_register_type_t type;
    uint8_t * const p_value;
    const uint8_t * const p_reset_value;
    uint8_t size;
} st_bfe_register_t;
```

To avoid data duplication, this structure declares two pointers: **p_value** and **p_reset_value**, which point to the global device register, and to the reset constant value, respectively. The field address contains the register address in the BFE; the field size specifies the size of the register in bytes; and the type enumeration indicates whether only read operation or both read and write operations are allowed on the register. The bank register is defined as:

```
/*Registers bank*/
st_raa489206_system_registers_t g_raa489206_registers =
{
 .die_information =
            {.address = RAA489206_REGISTER_SYS_INFO, .p_value =
&(g_dieinformation_register.value),
            .p_reset_value = &(g_reset_dieinformation_register.value), .type = READ,
             .size = (sizeof(g_dieinformation_register.value))/(sizeof(uint8_t))},

 .global_operation =
            {.address = RAA489206_REGISTER_GLOBAL_OP,.p_value =
&(g_globalop_register.value),
           .p_reset_value = (&g_reset_globalop_register.value), .type = READ_WRITE,
             .size = (sizeof(g_globalop_register.value))/(sizeof(uint8_t))},



 .vcell_operation =
            {.address= RAA489206_REGISTER_VCELL_OP, .p_value =
&(g_vcellop_register.value),
            .p_reset_value = (&g_reset_vcellop_register.value), .type = READ_WRITE,
             .size = (sizeof(g_vcellop_register.value))/(sizeof(uint8_t))},

 .ipack_operation =
            {.address = RAA489206_REGISTER_IPACK_OP, .p_value =
&(g_ipackop_register.value),
            .p_reset_value = (&g_reset_ipackop_register.value), .type = READ_WRITE,
             .size = (sizeof(g_ipackop_register.value))/(sizeof(uint8_t))},

 .cell_select =
          {.address = RAA489206_REGISTER_CELL_SEL, .p_value =
&(g_cells_select_register.lsb_value),
           .p_reset_value = (&g_reset_cells_select_register.lsb_value), .type =
READ_WRITE,
           .size = (sizeof(g_cells_select_register.value))/(sizeof(uint8_t))},

 .vcell_1 =
           {.address = RAA489206_REGISTER_VCELL_1, .p_value =
&(g_vcell1_voltage.lsb_value),
           .p_reset_value = (&g_reset_vcell_voltage.lsb_value), .type = READ,
           .size = (sizeof(g_vcell1_voltage.value))/(sizeof(uint8_t))},

 .vcell_2 =
          {.address = RAA489206_REGISTER_VCELL_2, .p_value =
&(g_vcell2_voltage.lsb_value),
           .p_reset_value = (&g_reset_vcell_voltage.lsb_value), .type = READ,
           .size = (sizeof(g_vcell2_voltage.value))/(sizeof(uint8_t))},
. . . .
```

```
 .cb_max_th =
          {.address = RAA489206_REGISTER_CB_MAX_TH, .p_value = &(g_cb_max_th),
           .p_reset_value = (&g_reset_cb_max_th), .type = READ_WRITE,
           .size = (sizeof(g_cb_max_th))/(sizeof(uint8_t))},

 .cb_min_th =
          {.address = RAA489206_REGISTER_CB_MIN_TH, .p_value = &(g_cb_min_th),
           .p_reset_value = (&g_reset_cb_min_th), .type = READ_WRITE,
           .size = (sizeof(g_cb_min_th))/(sizeof(uint8_t))},

 .eoc_voltage_th =
          {.address = RAA489206_REGISTER_VEOC_TH, .p_value = &(g_veoc_th),
           .p_reset_value = (&g_reset_veoc_th), .type = READ_WRITE,
           .size = (sizeof(g_veoc_th))/(sizeof(uint8_t))},

 .eoc_current_th =
          {.address = RAA489206_REGISTER_IEOC_TH, .p_value = &(g_ieoc_th),
           .p_reset_value = (&g_reset_ieoc_th), .type = READ_WRITE,
           .size = (sizeof(g_ieoc_th))/(sizeof(uint8_t))},
};
```

5.3.5 Private (Static) Variables and Functions

Most of the static definitions correspond to constants used during the conversion of register values into voltage or temperature, in addition to voltages and temperatures into register values. Other static variables, such as **s_device_busy** and **s_mode**, are defined as static to ensure valid addresses and share data between functions in the source code.

Table 8 shows the declaration and description of the static functions used by API functions in the source code. Some of them are detailed in the next sections as part of the description of API functions implementation.

Table 8. Static Functions Defined in the Source Code

Function	Description
static **e_bfe_err_t** write_spi **(uint8_t address, uint8_t** const * **p_values, uint16_t num_values, bool use_crc)**	Use the SPI interface to write **num_values** bytes of the data contained in the variable pointed by **p_values** in the register with address **address**. The boolean **use_crc** indicates whether CRC is used during the data transaction.
static **e_bfe_err_t** read_spi **(uint8_t address, uint8_t * p_values, uint16_t num_values, bool use_crc)**	Use the SPI interface to read **num_values** bytes of the register with address **address** and store the read value in the variable pointed by **p_values.**The boolean **use_crc** indicates whether CRC is used during the data transaction.
static e_bfe_err_t **read_spi_all_registers_no_crc**(uint8_t * p_values)	Reads all registers using the SPI interface without CRC and stores the values in the array starting at the position **p_values**
static e_bfe_err_t **read_spi_crc_command(uint8_t crc_command, uint8_t * p_values, uint16_t num_values)**	Use the SPI interface to read the group of registers determined by the special CRC read code **crc_command.** See the Read Operation section of RAA489206 datasheet.
static e_bfe_err_t **write_i2c**(uint8_t **reg_address**, const uint8_t * const **p_values**, uint16_t **num_values**, bool **use_crc**)	Use the I^{2}C interface to write **num_values** bytes of the data contained in the variable pointed by **p_values** in the register with address **reg_address**. The boolean **use_crc** is unused.
static e_bfe_err_t **read_i2c**(uint8_t **reg_address**, uint8_t * **p_values**, uint16_t **num_values**, bool is_cont)	Use the I^{2}C interface to read **num_values** bytes of the register with address **address** and store the read value in the variable pointed by **p_values.**The boolean **use_crc** is unused.

Table 8. Static Functions Defined in the Source Code (Cont.)

Function	Description
static **e_bfe_err_t** execute_startup_sequence **(st_raa489206_instance_ctrl_t *p_raa489206_ctrl)**	Executes the startup sequence recommended for the RAA489206 BFE
static **e_bfe_err_t** execute_basic_init (st_raa489206_instance_ctrl_t *p_raa489206_ctrl)	Initializes the BFE device using a group of configuration settings to evaluate its basic features
static **e_bfe_err_t** compare_reset_values **(st_raa489206_system_registers_t *p_regs)**	Reads the current values of all BFE registers and compare them with the reset values. The error **BFE_ERR_REGISTER_RESET_UNMATCHED** is returned when any register does not match its reset value
static **uint8_t** voltage_to_register **(**float **init,** float **set,** float **offset)**	Converts a voltage float value given in mV into its register value equivalent
static float register_to_voltage **(uint16_t reg_val,** float **set,** float **offset)**	Converts a register value into its voltage value equivalent in mV
static **uint8_t** temperature_to_register **(float deg_val)**	Converts a temperature float value given in °C into its register value equivalent
static float register_to_temperature **(uint8_t reg_val)**	Converts a register value into its temperature value equivalent in °C
static inline void wait_until_free **(st_raa489206_instance_ctrl_t *p_raa489206_ctrl, bool *p_is_busy, uint8_t loop_times)**	Query the device availability (busy bit state) for at most **loop_times*20ms.** This routine returns when the device is available clearing the boolean pointed by **p_is_busy,** or setting it if the device remains busy after **loop_times*20ms.**
static **uint16_t** calculate_crc **(uint16_t numbytes,** const **uint8_t *** const **input_buf)**	Calculates and returns the CRC value for **numbytes** bytes of the data starting at the address **input_buf,** according to the CRC-CITT16 X25 specification. Refer to Section 8.2.4 of the *RAA489206 Datasheet f*or details on its implementation.
void spi_callback (spi_callback_args_t *p_args)	SPI Interruption Service Routine
void i2c_callback (i2c_master_callback_args_t * p_args)	I^2C Interruption Service Routine

5.3.6 Interface API Implementation

The group of functions named in accordance with the convention R_<BFE>_<API_function> implement the functionalities that can be accessed by applications over the API structure. This section describes their implementations and interactions with the BFE device.

5.3.6.1 R_RAA489206_Init

<table>
<tr><th colspan="3">e_bfe_err_t R_RAA489206_Init(st_bfe_ctrl_t * p_api_ctrl, const st_bfe_cfg_t * const p_cfg)</th></tr>
<tr><td colspan="3">Initialize the control structure according to the configuration settings specified by the configuration structure:
1. Cast pointer p_api_ctrl to point to a structure of type st_raa489206_ctrl.
2. Set the use of CRC for data transactions between the MCU and the BFE.
3. Set the selected cells.
4. Set the pointer to the register bank.
5. Set the option Read-After-Write to read back written registers to verify their values.
6. Initialize the MCU I/O pins and assign them to the control structure.
7. Initialize the communication interface (SPI or I^2C) selected by the field p_cfg->peripheral_type.

Execute the startup sequence to set up the BFE to a functional state by calling execute_startup_sequence:
1. Wait 10ms until voltages stabilize.
2. Execute a hard reset clearing device reset pin for 50ms.
3. Wait 20ms for the whole power-up sequence to finish.
4. Read all registers and compare their values with reset values by calling compare_reset_values. Registers whose pointer-to-reset value is NULL are not compared.

Execute a basic initialization setting general operational settings by calling execute_basic_init:
1. Set Idle mode by setting and writing SYS-MODE bits to the enumeration RAA489206_SYSTEM_MODE_IDLE.
2. Disable communication timer by clearing the COM_TIMEOUT_EN bit of the V_{BAT1} operation register.
3. Select the strong regulator for low power mode by setting the LP_REG bit in the V_{REG} operation register.
4. Unmask the busy bit to assert the ALERT pin by clearing the BUSYM in the other faults register.
5. Set the internal temperature warning threshold to 85°C.
6. Set the internal temperature fault threshold to 95°C.
7. Set the selected cells that have specified by the control structure.
8. Enable the charge pump if it the corresponding control structure field has been set.
9. Set Scan select to 1 (single scan mode).</td></tr>
<tr><td rowspan="4">Returned values</td><td>BFE_SUCCESS</td><td>BFE successfully initialized</td></tr>
<tr><td>BFE_ERR_FSP_ERROR</td><td>Error initializing any FSP module (I/O, SPI)</td></tr>
<tr><td>BFE_ERR_COMM_NONSUPPORTED_
INTERFACE</td><td>Communication interface not supported</td></tr>
<tr><td>BFE_ERR_REGISTER_RESET_UNMATCHED</td><td>Register values after reset do not match default values. This might indicate the device is malfunctioning or is not connected to the MCU.</td></tr>
<tr><td>Observations</td><td colspan="2">This function initializes the control structure, which contains all data and references to the instances needed to interact with the BFE. Therefore, modules using the RA489206 API implementation must call R_RAA489206_Init before calling any other API function.
The caller function must ensure the pointer p_api_ctrl points to a structure of type st_raa489206_ctrl to avoid undetermined behavior.</td></tr>
</table>

5.3.6.2 R_RAA489206_Reset

e_bfe_err_t R_RAA489206_Reset(st_bfe_ctrl_t * const p_ctrl, e_bfe_reset_type_t reset_type)		
Perform the reset type determined by the enumeration **e_bfe_reset_type_t reset_type**: ▪ BFE_RESET_TYPE_SOFT Soft reset – Set SFT_RST bit to 1 in the global operation register. Reset all register values back to default values, including data registers. All counters are set to 0, all timers and faults are cleared, and the system mode is set to IDLE. A low voltage offset calibration is performed, the power and cell balancing FETs are turned off, and the state machines are reset. ▪ BFE_RESET_TYPE_TOIDLE Reset to Idle – Set the RST2IDLE bit of the global operation register to 1. Stop all state machines and moves the chip state to IDLE mode. Set all counters to 0 and clears timers and faults. Power and cell balancing FETs are turned off. The state machines are reset. All other register settings are NOT affected by this command and remain unchanged. ▪ BFE_RESET_TYPE_HARD Hard Reset – Clear the reset pin for 50ms.		
Returned values	BFE_SUCCESS	Reset has been successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_INVALID_ARGUMENT	Invalid reset type
	BFE_ERR_DEVICE_NOT_INITIALIZED	R_RAA489206_Init has not been called
Observations	---	

5.3.6.3 R_RAA489206_StartSystemScan

e_bfe_err_t R_RAA489206_StartSystemScan(st_bfe_ctrl_t * const p_ctrl)		
Start a single system scan sequence: 1. Cast pointer p_api_ctrl to point to a structure of type st_raa489206_ctrl and verify the device has been initialized. 2. Read the global operation register. Reading-Before-Writing (RBW) is a good practice to update the MCU registers bank and avoid overwriting bit-field in the BFE. 3. Clear the SYS_SCAN_TRIGGER bit of the global operation register to ensure it is 0 before setting it to 1. The transition from 0 to 1 initiates a system scan sequence. Setting this bit to 1 while its value is already 1 does not triggers a system scan sequence, so the bit is cleared to ensure the scan sequence can be triggered. 4. Store the RAW configuration setting in a temporary variable. 5. Deactivate the RAW feature before writing the global operation register. On completion the SYS_SCAN_TRIGGER bit is set to 0, so deactivating the RAW feature avoids generating a BFE_ERR_COM_READ_AFTER_WRITE_FAILED error. 6. Write the global operation register. 7. Restore the RAW configuration setting to its original value stored in the temporary variable.		
Returned values	BFE_SUCCESS	System scan successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	`R_RAA489206_Init` has not been called
Observations	This command does set the device to SCAN mode. Because a scan sequence can only be initiated while the device is in SCAN mode, the BFE ignores the execution of this function if it is in SHIP, LOW POWER, or IDLE mode.	

5.3.6.4 R_RAA489206_StartContinuousScan

<table>
<tr><th colspan="3">e_bfe_err_t R_RAA489206_StartContinuousScan(st_bfe_ctrl_t * const p_ctrl,
const bfe_cs_configuration_t * const st_cs_config)</th></tr>
<tr><td colspan="3">Start continuous scan operation:
1. Cast pointer p_api_ctrl to point to a structure of type st_raa489206_ctrl and verify the device has been initialized.
2. Cast the pointer st_cs_config to point to an instance-defined structure st_raa489206_cs_config_t which contains the continuous scan operation settings to be set.
3. Call the function R_RAA489206_StopContinuousScan to ensure the device is not already in continuous scan operation before configuring scan-related parameters.
4. Read the scan operation register.
5. Clear all current faults and verify the pointer to the continuous scan configuration struct is not NULL.
6. Read the scan operation register (RBW).
7. Set the SCAN_DELAY bit fields of the scan operation register to the enumeration value specified by the scan configuration field p_cont_scan_cfg->scan_delay.
8. Write the scan operation register in the BFE.
9. Read and set the measurements to be make during system scans according to the settings specified by the fields of the scan configuration structure:
• VCELL_EN and VCELL_AVG in the V_{CELL} operation register
• IPACK_EN and IPACK_AVG in the I_{PACK} operation register
• OW_EN in the power FET operation register
• OW_UPDATE in the I_{PACK} operation register
• OTHER_AVG, VBAT_ENABLE and ITEMP_ENABLE in the V_{BAT1} operation register
• OTHER_UPDATE in the V_{REG} operation register
• ETAUX_AVG and ETAUX_ENABLE in the etaux operation register
10. Read the global operation register (RBW).
11. Set the global operation register bits SCAN_SEL and SYS_SCAN_TRIGGER to 0.
12. Write the global operation register to set the device to continuous scan operation.
13. Read the scan operation register (RBW).
14. Set SYS_MODE bits-field of the scan operation register to SCAN mode.
15. Init a complete system scan.</td></tr>
<tr><td rowspan="3">Returned values</td><td>BFE_SUCCESS</td><td>Continuous scan successfully started</td></tr>
<tr><td>BFE_ERR_FSP_ERROR</td><td>FSP module error</td></tr>
<tr><td>BFE_ERR_DEVICE_NOT_INITIALIZED</td><td>R_RAA489206_Init has not been called</td></tr>
<tr><td>Observations</td><td colspan="2">The caller function must ensure st_cs_config points to a structure of type st_raa489206_cs_config_t to avoid undetermined behavior.</td></tr>
</table>

5.3.6.5 R_RAA489206_StopContinuousScan

<table>
<tr><th colspan="3">e_bfe_err_t R_RAA489206_StopContinuousScan(st_bfe_ctrl_t * const p_ctrl)</th></tr>
<tr><td colspan="3">Stop continuous scan operation:
1. Cast pointer p_api_ctrl to point to a structure of type st_raa489206_ctrl and verify the device has been initialized.
2. Cast the void pointer bfe_cs_configuration_t * const st_cs_config to the instance-defined structure that contains continuous scan settings: st_raa489206_cs_config_t *p_cont_scan_cfg.
3. Read the global operation register and return BFE_SUCCESS if the device is not in continuous scan.
4. Set the global operation bits SCAN_SEL to 1 and SYS_SCAN_TRIGGER to 0.
5. Read the scan operation register to obtain the current scan delay SCAN_DELAY.
6. Wait for at least the SCAN_DELAY time to ensure continuous scan operation has stopped.
7. Clear all faults</td></tr>
<tr><td rowspan="3">Returned values</td><td>BFE_SUCCESS</td><td>Function successfully executed</td></tr>
<tr><td>BFE_ERR_FSP_ERROR</td><td>FSP module error</td></tr>
<tr><td>BFE_ERR_DEVICE_NOT_INITIALIZED</td><td>R_RAA489206_Init has not been called</td></tr>
<tr><td>Observations</td><td colspan="2">When the continuous scan operation is stopped by setting the SCAN_SEL bit of the global operation register to 0, the device performs a last system scan after the current SCAN_DELAY times out. Writing scan settings before this last scan operation is performed may result in undetermined device behavior, so this function reads the current scan delay and waits for the last scan to be completed.</td></tr>
</table>

5.3.6.6 R_RAA489206_IsBusy

<table>
<tr><th colspan="3">e_bfe_err_t R_RAA489206_IsBusy(st_bfe_ctrl_t * const p_ctrl, bool * p_is_busy)</th></tr>
<tr><td colspan="3">Read the busy bit value in the global operation register and return it in the boolean variable pointed by p_is_busy:
1. Cast pointer p_api_ctrl to point to a structure of type st_raa489206_ctrl and verify the device has been initialized.
2. Read the global operation register.
3. Set the value pointed by p_is_busy to the value of the BUSY bit.</td></tr>
<tr><td rowspan="3">Returned values</td><td>BFE_SUCCESS</td><td>Function successfully executed</td></tr>
<tr><td>BFE_ERR_FSP_ERROR</td><td>FSP module error</td></tr>
<tr><td>BFE_ERR_DEVICE_NOT_INITIALIZED</td><td>R_RAA489206_Init has not been called</td></tr>
<tr><td>Observations</td><td colspan="2">---</td></tr>
</table>

5.3.6.7 R_RAA489206_InitSystemScan

e_bfe_err_t R_RAA489206_ReadStatus(st_bfe_ctrl_t * const p_ctrl, bfe_status_t * const p_bfe_status)		
Read status and faults registers values and return them in the fields of the structure p_bfe_status points to: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Cast the input pointer **p_bfe_status** to point to an instance-defined structure of type **st_raa489206_status_t** 3. Read the priority faults register value and store it in the **priority_status** structure field. 4. Read the ETAUX faults register value and store it in the **etaux_status** structure field. 5. Read the other faults register value and store it in the **other_status** structure field. 6. Read the cell balancing status register value and store it in the **cb_status** structure field. 7. Read the general status register value and store it in **general_status** structure field. 8. Read the open-wire status register value and store it in the **ow_status** structure field.		
Returned values	BFE_SUCCESS	Read status successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	The caller function must ensure **p_bfe_status** points to a structure of type **st_raa489206_status_t** to avoid undetermined behavior.	

5.3.6.8 R_RAA489206_ClearAllFaults

e_bfe_err_t R_RAA489206_ClearAllFaults(st_bfe_ctrl_t * const p_ctrl)		
Clear all faults and status bits in the register range 0x63 – 0x69, except for 0x67.5 – CH_PRESI, along with all counters: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the V_{CELL} operation register (RBW). 3. Set the CLR_FAULTS_STATUS bit to 1 4. Store the RAW setting in a temporary variable. 5. Deactivate the RAW feature before writing the global operation register. On completion the bit CLR_FAULTS_STATUS is set to 0, so deactivating the RAW feature avoids generating a **BFE_ERR_COM_READ_AFTER_WRITE_FAILED** error. 6. Write the V_{CELL} operation register. 7. Restore the RAW configuration setting to its original value stored in the temporary variable. 8. Wait until the CLR_FAULTS_STATUS and fault registers are cleared to 0, which indicates the clear operation is finished.		
Returned values	BFE_SUCCESS	Clear all faults successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This function cannot clear the faults and status registers while the condition that sets them is present.	

5.3.6.9 R_RAA489206_ClearFault

e_bfe_err_t R_RAA489206_ClearFault(st_bfe_ctrl_t * const p_ctrl, const bfe_fault_type_t * const p_bfe_fault_type)		
Clear the fault(s) and status bit(s) specified by the instance-defined enumeration **p_bfe_status** points to: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Cast the input pointer **p_bfe_fault_type** to point to an **e_raa489206_fault_type_t** enumeration. 3. Store the RAW setting in a temporary variable. 4. Deactivate the RAW feature before writing the fault/status register. The bits of faults and status registers are only cleared while the condition that sets them is present, so deactivating RAW feature avoids generating **BFE_ERR_COM_READ_AFTER_WRITE_FAILED** error if the condition persists. 5. Clear the bits of the fault or status register indicated by the enumeration. 6. Write the fault or status register. 7. Read back the written register to update its value in the register bank. 8. Restore the RAW configuration setting to its original value stored in the temporary variable.		
Returned values	BFE_SUCCESS	Clear fault successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	The caller function must ensure **p_bfe_fault_type** points to an enumeration of type **e_raa489206_fault_type_t** to avoid undetermined behavior. This function cannot clear the faults and status registers while the condition that sets them is present.	

5.3.6.10 R_RAA489206_ReadMode

e_bfe_err_t R_RAA489206_ReadMode (st_bfe_ctrl_t * const p_ctrl, e_bfe_mode_t * const p_value)		
Read and return the current BFE mode as the value of the variable pointed by p_value: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the scan operation register 3. Map the value of the SYS_MODE bits-field to a value of the **e_bfe_mode_t** enumeration setting the variable p_value points to.		
Returned values	BFE_SUCCESS	Read mode successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_NONSUPPORTED_MODE	The mode specified by the bits-field is not included in the enumeration.
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.11 R_RAA489206_ReadVpack

<table>
<tr><th colspan="3">e_bfe_err_t R_RAA489206_ReadVpack(st_bfe_ctrl_t * const p_ctrl, float * const p_value, bool trigger)</th></tr>
<tr><td colspan="3">Read and return the pack voltage in mV storing its value in the variable pointed by p_value. The boolean parameter trigger specifies whether a V_{PACK} measurement precedes (trigger = true) the reading operation.
1. Cast pointer p_api_ctrl to point to a structure of type st_raa489206_ctrl and verify the device has been initialized.
2. If trigger is true, then:
a. Read the global operation register.
b. Check whether the device is in continuous scan operation by reading the values of SCAN_SEL and device mode. If the device is in continuous operation, stop it before triggering any measurement.
c. Read V_{BAT1} operation register (RBW) and make sure VBAT_TRIGGER bit is set to 0 to ensure 0-1 transition. Write the register in the device accordingly if the bit value is 1.
d. Set VBAT_TRIGGER bit to 1.
e. Wait until the device is available by calling wait_for_free.
f. Store the RAW setting in a temporary variable and deactivate the RAW feature to avoid generating BFE_ERR_COM_READ_AFTER_WRITE_FAILED error when the trigger bit is set back to 0 after completion of the measurement.
g. Write the V_{BAT1} operation register to trigger a V_{PACK} measurement.
h. Restore the RAW configuration setting to its original value stored in the temporary variable.
3. Wait until the device is available by calling wait_for_free. This avoids reading measurements while the device is processing any task, which may result in outdated readings.
4. Read the V_{BAT1} voltage register and convert its value to mV.
5. Write the value in mV in the value pointed by p_value.
6. Restart continuous scan operation if it has been stopped in previous steps.
7. Evaluate whether the measurement has been read while the device was in busy.</td></tr>
<tr><td rowspan="4">Returned values</td><td>BFE_SUCCESS</td><td>Pack voltage reading successfully executed</td></tr>
<tr><td>BFE_WARN_BUSY</td><td>The returned value has been read while the device was busy.</td></tr>
<tr><td>BFE_ERR_FSP_ERROR</td><td>FSP module error</td></tr>
<tr><td>BFE_ERR_DEVICE_NOT_INITIALIZED</td><td>R_RAA489206_Init has not been called</td></tr>
<tr><td>Observations</td><td colspan="2">This routine attempts to read the V_{PACK} value while the device is available (this is, the busy bit is 0). However, the wait_for_free routine does not guarantee the device is free after returning to avoid blocking the program flow. Therefore, if the device is busy after wait_for_free returns, this routine generates the code BFE_WARN_BUSY to alert the caller function to a possible outdated reading.</td></tr>
</table>

5.3.6.12 R_RAA489206_ReadIpack

e_bfe_err_t R_RAA489206_ReadIpack(st_bfe_ctrl_t * const p_ctrl, float * const p_value, bool trigger)		
Read and return the pack current in mA storing its value in the variable pointed by p_value. The boolean parameter trigger specifies whether an I_{PACK} measurement precedes (trigger = true) the reading operation. 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. If trigger is true, then: a. Read the global operation register. b. Check whether the device is in continuous scan operation by reading the values of SCAN_SEL and device mode. If the device is in continuous operation, stop it before triggering any measurement. c. Read I_{PACK} operation register (RBW) and make sure IPACK_TRIGGER bit is set to 0 to ensure 0-1 transition. Write the register in the device accordingly if the bit value is 1. d. Set IPACK_TRIGGER bit to 1. e. Wait until the device is available by calling **wait_for_free**. f. Store the RAW setting in a temporary variable and deactivate the RAW feature to avoid generating **BFE_ERR_COM_READ_AFTER_WRITE_FAILED** error when the trigger bit is set back to 0 after completion of the measurement. g. Write the I_{PACK} operation register to trigger an I_{PACK} measurement. h. Restore the RAW configuration setting to its original value stored in the temporary variable. 3. Wait until the device is available by calling **wait_for_free**. This avoids reading measurements while the device is processing any task, which may result in outdated readings. 4. Read the I_{PACK} voltage register and convert its value into mV. 5. Convert the voltage value into mA using the shunt resistor value. 6. Write the value in mA in the value pointed by **p_value**. 7. Restart continuous scan operation if it has been stopped in previous steps. 8. Evaluate whether the measurement has been read while the device was in busy.		
Returned values	BFE_SUCCESS	Pack current reading successfully executed
	BFE_WARN_BUSY	The returned value has been read while the device was busy.
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This routine attempts to read the I_{PACK} value while the device is available (this is, the busy bit is 0). However, the **wait_for_free** routine does not guarantee the device is free after returning to avoid blocking the program flow. Therefore, if the device is busy after **wait_for_free** returns, this routine generates the code **BFE_WARN_BUSY** to alert the caller function to a possible outdated reading.	

5.3.6.13 R_RAA489206_ReadVcells

<table>
<tr><th colspan="3">e_bfe_err_t R_RAA489206_ReadVcells(st_bfe_ctrl_t * const p_ctrl, bfe_vcell_measurements_t * const p_values, bool trigger)</th></tr>
<tr><td colspan="3">Read and return the cells voltage in mV storing their values in the instance-defined structure pointed by p_value. The boolean parameter trigger specifies whether cells voltage measurements precede (trigger = true) the reading operation.
1. Cast pointer p_api_ctrl to point to a structure of type st_raa489206_ctrl and verify the device has been initialized.
2. Cast pointer p_values to point to a union of type u_raa489206_vcell_measurements_t.
3. If trigger is true, then:
a. Read the global operation register.
b. Check whether the device is in continuous scan operation by reading the values of SCAN_SEL and device mode. If the device is in continuous operation, stop it before triggering any measurement.
c. Read V_{CELL} operation register (RBW) and make sure VCELL_TRIGGER bit is set to 0 to ensure 0-1 transition. Write the register in the device accordingly if the bit value is 1.
d. Set VCELL_TRIGGER bit to 1.
e. Wait until the device is available by calling <code>wait_for_free</code>.
f. Store the RAW setting in a temporary variable and deactivate the RAW feature to avoid generating BFE_ERR_COM_READ_AFTER_WRITE_FAILED error when the trigger bit is set back to 0 after completion of the measurement.
g. Write the V_{CELL} operation register to trigger measurements of cells voltage.
h. Restore the RAW configuration setting to its original value stored in the temporary variable.
4. Wait until the device is available by calling wait_for_free. This avoids reading measurements while the device is processing any task, which may result in outdated readings.
5. Read cell voltage registers and convert their values into mV.
6. Write the values in mV in the fields of the structure of type u_raa489206_vcell_measurements_t.
7. Restart continuous scan operation if it has been stopped in previous steps.
8. Evaluate whether measurements have been read while the device was in busy.</td></tr>
<tr><td rowspan="4">Returned values</td><td>BFE_SUCCESS</td><td>Cell voltage readings successfully executed</td></tr>
<tr><td>BFE_WARN_BUSY</td><td>The returned values have been read while the device was busy.</td></tr>
<tr><td>BFE_ERR_FSP_ERROR</td><td>FSP module error</td></tr>
<tr><td>BFE_ERR_DEVICE_NOT_INITIALIZED</td><td>R_RAA489206_Init has not been called</td></tr>
<tr><td>Observations</td><td colspan="2">This routine attempts to read the cell voltages while the device is available (this is, the busy bit is 0). However, the wait_for_free routine does not guarantee the device is free after returning to avoid blocking the program flow. Therefore, if the device is busy after wait_for_free returns, this routine generates the code BFE_WARN_BUSY to alert the caller function to a possible outdated reading.
The caller function must ensure p_values points to a union of type u_raa489206_vcell_measurements_t to avoid undetermined behavior.
If the CRC option has been enabled, this function executes the command that retrieves all registers values in one data transaction (CRC command 0x9C). Otherwise, voltage registers are read one by one in different data transactions.</td></tr>
</table>

5.3.6.14 R_RAA489206_ReadOther

<table>
<tr><th colspan="3">e_bfe_err_t R_RAA489206_ReadOther(st_bfe_ctrl_t * const p_ctrl, bfe_other_measurements_t * const p_values, bool trigger)</th></tr>
<tr><td colspan="3">Read and return the voltage in mV of other measurements storing their value in the instance-defined structure pointed by p_value. The boolean parameter trigger specifies whether other voltage measurements precede (trigger=true) the reading operation.
1. Cast pointer p_api_ctrl to point to a structure of type st_raa489206_ctrl and verify the device has been initialized.
2. Cast pointer p_values to point to a union of type <code>u_raa489206_other_measurements_t.</code>
3. If trigger is true, then:
a. Read the global operation register.
b. Check whether the device is in continuous scan operation by reading the values of SCAN_SEL and device mode. If the device is in continuous operation, stop it before triggering any measurement.
c. Read V_{REG} operation register (RBW) and make sure VREG_TRIGGER bit is set to 0 to ensure 0-1 transition. Write the register in the device accordingly if the bit value is 1.
d. Set VREG_TRIGGER bit to 1.
e. Wait until the device is available by calling wait_for_free.
f. Store the RAW setting in a temporary variable and deactivate the RAW feature to avoid generating BFE_ERR_COM_READ_AFTER_WRITE_FAILED error when the trigger bit is set back to 0 after completion of the measurement.
g. Write the V_{REG} operation register to trigger measurements of other voltages.
h. Restore the RAW configuration setting to its original value stored in the temporary variable.
4. Wait until the device is available by calling wait_for_free. This avoids reading measurements while the device is processing any task, which may result in outdated readings.
5. Read Vcc, Ireg, and Vtemp voltage registers and convert their values into mV.
6. Write the values in mV in the fields of the structure of type u_raa489206_other_measurements_t.
7. Restart continuous scan operation if it has been stopped in previous steps.
8. Evaluate whether measurements have been read while the device was in busy.</td></tr>
<tr><td rowspan="4">Returned values</td><td>BFE_SUCCESS</td><td>Other voltage readings successfully executed</td></tr>
<tr><td>BFE_WARN_BUSY</td><td>The returned values have been read while the device was busy.</td></tr>
<tr><td>BFE_ERR_FSP_ERROR</td><td>FSP module error</td></tr>
<tr><td>BFE_ERR_DEVICE_NOT_INITIALIZED</td><td>R_RAA489206_Init has not been called</td></tr>
<tr><td>Observations</td><td colspan="2">This routine attempts to read the other voltage values while the device is available (this is, the busy bit is 0). However, the wait_for_free routine does not guarantee the device is free after returning to avoid blocking the program flow. Therefore, if the device is busy after wait_for_free returns, this routine generates the code BFE_WARN_BUSY to alert the caller function to a possible outdated reading.
The caller function must ensure p_values points to a union of type u_raa489206_other_measurements_t to avoid undetermined behavior.</td></tr>
</table>

5.3.6.15 R_RAA489206_ReadAuxExt

<table>
<tr><th colspan="3">e_bfe_err_t R_RAA489206_ReadAuxExt(st_bfe_ctrl_t * const p_ctrl, bfe_auxext_measurements_t * const p_values, bool trigger)</th></tr>
<tr><td colspan="3">Read and return the voltage in mV of auxiliary/external measurements storing their value in the instance-defined structure pointed by p_value. The boolean parameter trigger specifies whether auxiliary/external voltage measurements precede (trigger = true) the reading operation.
1. Cast pointer p_api_ctrl to point to a structure of type st_raa489206_ctrl and verify the device has been initialized.
2. Cast pointer p_values to point to a union of type u_raa489206_etaux_measurements_t.
3. If trigger is true, then:
a. Read the global operation register.
b. Check whether the device is in continuous scan operation by reading the values of SCAN_SEL and device mode. If the device is in continuous operation, stop it before triggering any measurement.
c. Read Etaux operation register (RBW) and make sure ETAUX_TRIGGER bit is set to 0 to ensure 0-1 transition. Write the register in the device accordingly if the bit value is 1.
d. Set ETAUX_TRIGGER bit to 1.
e. Wait until the device is available by calling wait_for_free.
f. Store the RAW setting in a temporary variable and deactivate the RAW feature to avoid generating BFE_ERR_COM_READ_AFTER_WRITE_FAILED error when the trigger bit is set back to 0 after completion of the measurement.
g. Write the Etaux operation register to trigger measurements of auxiliary/external voltages.
h. Restore the RAW configuration setting to its original value stored in the temporary variable.
4. Wait until the device is available by calling wait_for_free. This avoids reading measurements while the device is processing any task, which may result in outdated readings.
5. Read etaux0 and etaux1 voltage registers and convert their values into mV.
6. Write the values in mV in the fields of the structure of type u_raa489206_etaux_measurements_t.
7. Restart continuous scan operation if it has been stopped in previous steps.
8. Evaluate whether measurements have been read while the device was in busy.</td></tr>
<tr><td rowspan="4">Returned values</td><td>BFE_SUCCESS</td><td>Etaux voltage readings successfully executed</td></tr>
<tr><td>BFE_WARN_BUSY</td><td>The returned values have been read while the device was busy.</td></tr>
<tr><td>BFE_ERR_FSP_ERROR</td><td>FSP module error</td></tr>
<tr><td>BFE_ERR_DEVICE_NOT_INITIALIZED</td><td>R_RAA489206_Init has not been called</td></tr>
<tr><td>Observations</td><td colspan="2">This routine attempts to read the auxiliary/external voltage values while the device is available (this is, the busy bit is 0). However, the wait_for_free routine does not guarantee the device is free after returning to avoid blocking the program flow. Therefore, if the device is busy after wait_for_free returns, this routine generates the code BFE_WARN_BUSY to alert the caller function to a possible outdated reading.
The caller function must ensure p_values points to a union of type u_raa489206_etaux_measurements_t to avoid undetermined behavior.</td></tr>
</table>

5.3.6.16 R_RAA489206_ReadTemperature

e_bfe_err_t R_RAA489206_ReadTemperature(st_bfe_ctrl_t * const p_ctrl, bfe_temperature_measurements_t * const p_value, bool trigger)		
Read and return the device internal temperature in °C storing its value in the instance-defined structure pointed by p_value. The boolean parameter trigger specifies whether an internal temperature measurement precede (trigger = true) the reading operation. 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Cast pointer p_value to point to a structure of type **st_raa489206_temperature_measurements_t**. 3. If trigger is true, then: a. Read the global operation register. b. Check whether the device is in continuous scan operation by reading the values of SCAN_SEL and device mode. If the device is in continuous operation, stop it before triggering any measurement. c. Read V_{BAT1} operation register (RBW) and make sure ITEMP_TRIGGER bit is set to 0 to ensure 0-1 transition. Write the register in the device accordingly if the bit value is 1. d. Set ITEMP _TRIGGER bit to 1. e. Wait until the device is available by calling **wait_for_free**. f. Store the RAW setting in a temporary variable and deactivate the RAW feature to avoid generating **BFE_ERR_COM_READ_AFTER_WRITE_FAILED** error when the trigger bit is set back to 0 after completion of the measurement. g. Write the Vba1 operation register to trigger the internal temperature measurement. h. Restore the RAW configuration setting to its original value stored in the temporary variable. 4. Wait until the device is available by calling **wait_for_free**. This avoids reading measurements while the device is processing any task, which may result in outdated readings. 5. Read the internal temperature voltage register and convert its value into °C. 6. Write the value in °C in the structure of type **st_raa489206_temperature_measurements_t**. 7. Restart continuous scan operation if it has been stopped in previous steps. 8. Evaluate whether measurements have been read while the device was in busy.		
Returned values	BFE_SUCCESS	Temperature reading successfully executed
	BFE_WARN_BUSY	The returned value has been read while the device was busy.
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	`R_RAA489206_Init` has not been called
Observations	This routine attempts to read the internal temperature while the device is available (this is, the busy bit is 0). However, the **wait_for_free** routine does not guarantee the device is free after returning to avoid blocking the program flow. Therefore, if the device is busy after **wait_for_free** returns, this routine generates the code **BFE_WARN_BUSY** to alert the caller function to a possible outdated reading. The caller function must ensure **p_value** points to a structure of type **st_raa489206_temperature_measurements_t** to avoid undetermined behavior.	

5.3.6.17 R_RAA489206_ReadDOC

e_bfe_err_t R_RAA489206_ReadDOC(st_bfe_ctrl_t * const p_ctrl, float * p_current_ma)		
Read and return the Discharge Overcurrent (DOC) threshold in mA storing its value the variable pointed by p_current_ma: 1. Cast pointer `p_api_ctrl` to point to a structure of type `st_raa489206_ctrl` and verify the device has been initialized. 2. Read the DOC threshold register and convert its value into mV. 3. Convert the voltage value into mA using the shunt resistor value. 4. Write the value in mA in the variable pointed by `p_current_ma`.		
Returned values	BFE_SUCCESS BFE_ERR_FSP_ERROR BFE_ERR_DEVICE_NOT_INITIALIZED	DOC reading successfully executed FSP module error `R_RAA489206_Init` has not been called
Observations	---	

5.3.6.18 R_RAA489206_ReadCOC

e_bfe_err_t R_RAA489206_ReadCOC(st_bfe_ctrl_t * const p_ctrl, float * p_current_ma)		
Read and return the Charge Overcurrent (COC) threshold in mA storing its value the variable pointed by p_current_ma: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the COC threshold register and convert its value into mV. 3. Convert the voltage value into mA using the shunt resistor value. 4. Write the value in mA in the variable pointed by **p_current_ma.**		
Returned values	BFE_SUCCESS	COC reading successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.19 R_RAA489206_ReadDSC

e_bfe_err_t R_RAA489206_ReadDSC(st_bfe_ctrl_t * const p_ctrl, float * p_current_ma)		
Read and return the Discharge Short-circuit Current (DSC) threshold in mA storing its value in the variable pointed by p_current_ma: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the DSC threshold register and convert its value into mV. 3. Convert the voltage value into mA using the shunt resistor value. 4. Write the value in mA in the variable pointed by **p_current_ma**.		
Returned values	BFE_SUCCESS	DSC reading successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.20 R_RAA489206_ReadMaxVcellDeltaVoltage

e_bfe_err_t R_RAA489206_ReadMaxVcellDeltaVoltage(st_bfe_ctrl_t * const p_ctrl, float * p_cells_maxd_th_mv)		
Read and return the cells voltage maximum delta in mV storing its value in the variable pointed by **p_cells_maxd_th_mv**: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the maximum cells voltage threshold register and convert its value into mV. 3. Write the value in mV in the variable pointed by **p_cells_maxd_th_mv.**		
Returned values	BFE_SUCCESS	Maximum delta reading successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.21 R_RAA489206_ReadCellUndervoltage

e_bfe_err_t R_RAA489206_ReadCellUndervoltage(st_bfe_ctrl_t * const p_ctrl, float * p_cells_uv_th_mv)		
Read and return the cell undervoltage (V_{CELL} UV) threshold in mV storing its value in the variable pointed by **p_cells_uv_th_mv**: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the cells undervoltage threshold register and convert its value into mV. 3. Write the value in mV in the variable pointed by **p_cells_uv_th_ma**.		
Returned values	BFE_SUCCESS	V_{CELL} UV threshold reading successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.22 R_RAA489206_ReadCellOvervoltage

e_bfe_err_t R_RAA489206_ReadCellOvervoltage(st_bfe_ctrl_t * const p_ctrl, float * p_cells_ov_th_mv)		
Read and return the cell undervoltage (V_{CELL} OV) threshold in mV storing its value in the variable pointed by **p_cells_ov_th_mv**: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the cells overvoltage threshold register and convert its value into mV. 3. Write the value in mV in the variable pointed by **p_cells_ov_th_ma**.		
Returned values	BFE_SUCCESS	V_{CELL} OV threshold reading successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.23 R_RAA489206_ReadVpackUndervoltage

e_bfe_err_t R_RAA489206_ReadVpackUndervoltage(st_bfe_ctrl_t * const p_ctrl, float * p_vpack_uv_th_mv)		
Read and return the pack undervoltage (V_{PACK} UV) threshold in mV storing its value in the variable pointed by **p_vpack_uv_th_mv**: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the pack undervoltage threshold register and convert its value into mV. 3. Write the value in mV in the variable pointed by **p_vpack_uv_th_ma**.		
Returned values	BFE_SUCCESS	V_{CELL} UV threshold reading successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.24 R_RAA489206_ReadVpackOvervoltage

e_bfe_err_t R_RAA489206_ReadVpackOvervoltage(st_bfe_ctrl_t * const p_ctrl, float * p_vpack_ov_th_mv)		
Read and return the pack overvoltage (V_{PACK} OV) threshold in mV storing its value in the variable pointed by **p_vpack_ov_th_mv**: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the pack overvoltage threshold register and convert its value into mV. 3. Write the value in mV in the variable pointed by **p_vpack_ov_th_ma**.		
Returned values	BFE_SUCCESS	V_{PACK} OV threshold reading successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.25 R_RAA489206_ReadInternalOTWarning

e_bfe_err_t R_RAA489206_ReadInternalOTWarning(st_bfe_ctrl_t * const p_ctrl, float * p_war_temperature)		
Read and return the internal over-temperature warning (IOTW) threshold in °C storing its value in the variable pointed by **p_war_temperature**: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the internal over-temperature warning threshold register and convert its value into °C. 3. Write the value in °C in the variable pointed by **p_war_temperature.**		
Returned values	BFE_SUCCESS	IOTW threshold reading successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.26 R_RAA489206_ReadInternalOTFault

e_bfe_err_t R_RAA489206_ReadInternalOTFaultst_bfe_ctrl_t * const p_ctrl, float * p_fault_temperature)		
Read and return the internal over-temperature fault (IOTF) threshold in °C storing its value in the variable pointed by **p_fault_temperature**: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the internal over-temperature warning threshold register and convert its value into °C. 3. Write the value in °C in the variable pointed by **p_fault_temperature.**		
Returned values	BFE_SUCCESS	IOTF threshold reading successfully executed
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.27 R_RAA489206_TurnChargePumpOn

e_bfe_err_t R_RAA489206_TurnChargePumpOn(st_bfe_ctrl_t * const p_ctrl)		
Turn the BFE charge pump on: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set the CPMP_EN bit to 1. 4. Write the power FET operation register to turn the pump on. 5. Wait for 10ms to ensure its output rises.		
Returned values	BFE_SUCCESS	Charge pump is on
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.28 R_RAA489206_TurnChargePumpOff

e_bfe_err_t R_RAA489206_TurnChargePumpOff(st_bfe_ctrl_t * const p_ctrl)		
Turn the BFE charge pump off: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set the CPMP_EN bit to 0. 4. Write the power FET operation register to turn the pump off.		
Returned values	BFE_SUCCESS	Charge pump is off
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.29 R_RAA489206_TurnDFetOn

e_bfe_err_t R_RAA489206_TurnDFetOn(st_bfe_ctrl_t * const p_ctrl)		
Turn the BFE DFET on: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set the DFET_EN bit to 1. 4. Write the power FET operation register to turn the DFET on.		
Returned values	BFE_SUCCESS	DFET is on
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.30 R_RAA489206_TurnDFetOff

e_bfe_err_t R_RAA489206_TurnDFetOff(st_bfe_ctrl_t * const p_ctrl)		
Turn the BFE DFET off: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set the DFET_EN bit to 0. 4. Write the power FET operation register to turn the DFET off.		
Returned values	BFE_SUCCESS	DFET is off
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.31 R_RAA489206_TurnCFetOn

e_bfe_err_t R_RAA489206_TurnCFetOn(st_bfe_ctrl_t * const p_ctrl)		
Turn the BFE CFET on: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set the CFET_EN bit to 1. 4. Write the power FET operation register to turn the CFET on.		
Returned values	BFE_SUCCESS	CFET is on
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.32 R_RAA489206_TurnCFetOff

e_bfe_err_t R_RAA489206_TurnCFetOff(st_bfe_ctrl_t * const p_ctrl)		
Turn the BFE CFET off: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set the CFET_EN bit to 0. 4. Write the power FET operation register to turn the CFET off.		
Returned values	BFE_SUCCESS	CFET is on
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.33 R_RAA489206_TurnDFetOnCFetOn

e_bfe_err_t R_RAA489206_TurnDFetOnCFetOn(st_bfe_ctrl_t * const p_ctrl)		
Turn both BFE DFET and CFET on: 1. Cast pointer **`p_api_ctrl`** to point to a structure of type **`st_raa489206_ctrl`** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set both DFET_EN and CFET_EN bits to 1. 4. Write the power FET operation register to turn both FETs on.		
Returned values	BFE_SUCCESS	CFET is on
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.34 R_RAA489206_TurnDFetOffCFetOn

e_bfe_err_t R_RAA489206_TurnDFetOffCFetOn(st_bfe_ctrl_t * const p_ctrl)		
Turn BFE DFET off and CFET on: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set DFET_EN bit to 0 and CFET_EN bit to 1. 4. Write the power FET operation register to turn DFET off and CFET on.		
Returned values	BFE_SUCCESS	DFET is off and CFET is on
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.35 R_RAA489206_TurnDFetOnCFetOff

e_bfe_err_t R_RAA489206_TurnDFetOnCFetOff(st_bfe_ctrl_t * const p_ctrl)		
Turn BFE DFET on and CFET off: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set DFET_EN bit to 1 and CFET_EN bit to 0. 4. Write the power FET operation register to turn DFET on and CFET off.		
Returned values	BFE_SUCCESS	DFET is on and CFET is off
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.36 R_RAA489206_TurnDFetOffCFetOff

e_bfe_err_t R_RAA489206_TurnDFetOffCFetOff(st_bfe_ctrl_t * const p_ctrl)		
Turn both BFE DFET and CFET off: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the power FET operation register (RBW). 3. Set both DFET_EN and CFET_EN bits to 0. 4. Write the power FET operation register to turn both FETs off.		
Returned values	BFE_SUCCESS	DFET and CFET are off
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.37 R_RAA489206_SetAlerts

e_bfe_err_t R_RAA489206_SetAlerts (st_bfe_ctrl_t * const p_ctrl, const bfe_alerts_masks_t * const p_alert_events)		
Set the fault delays and events that assert the ALERT pin. **p_alert_events** points to a structure containing fault delay settings and faults to be unmasked: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Cast the input pointer **p_alert_events** to point to an instance-defined structure of type **st_raa489206_events_masks_t.** 3. Set the fault delays values and write their corresponding registers: • Fault delay register. • DSC delay register. • Over-current delay register. • LD_DELAY bits-group in the V_{REG} operations register (execute RBW). • IDIR_DELAY bits-group in the I_{PACK} operation register (execute RBW). 4. Unmask the faults allowed to assert the ALERT pin: • Priority faults mask register. • Etaux faults mask register. • Other faults mask register. • Cell balancing faults register. • Status masks register. • Open-wire mask register.		
Returned values	BFE_SUCCESS	Alert events are successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	Alert events have not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	The caller function must ensure **p_alert_events** points to a structure of type **st_raa489206_events_masks_t** to avoid undetermined behavior. The RBW practice only precedes writing operations that update bits-fields: LD_DELAY in the V_{REG} operation register, and IDIR_DELAY in the I_{PACK} operation register. The other writing operations overwrite the whole registers values, so RBW is not executed. The RAW feature verifies the values are successfully set.	

5.3.6.38 R_RAA489206_SetDOC

e_bfe_err_t R_RAA489206_SetDOC(st_bfe_ctrl_t * const p_ctrl, float current_ma)		
Set the Discharge Overcurrent (DOC) threshold in mA: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the current threshold **current_ma** in mA into mV using the shunt resistor value. 3. Convert the voltage in mV into its register value. 4. Set and write the discharge overcurrent threshold register.		
Returned values	BFE_SUCCESS	DOC threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	DOC threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This function expects a negative value, which indicates that charge is drained from the battery pack. The DOC threshold is an 8-bit register with valid range **0x00 ≤ register value ≤ = 0xFF**. Input thresholds that result in conversions out of this range are set accordingly to either the maximum (0xFF) or minimum (0x00) register value. See the datasheet for detailed information about the threshold range and granularity the register can represent. The RAW feature verifies the value is successfully set.	

5.3.6.39 R_RAA489206_SetCOC

e_bfe_err_t R_RAA489206_SetCOC(st_bfe_ctrl_t * const p_ctrl, float current_ma)		
Set the Charge Overcurrent (COC) threshold in mA: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the current threshold **current_ma** in mA into mV using the shunt resistor value. 3. Convert the voltage in mV into its register value. 4. Set and write the charge overcurrent threshold register.		
Returned values	BFE_SUCCESS	COC threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	COC threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This function expects a positive value, which indicates charge is being given to the battery pack. The COC threshold is an 8-bit register with valid range **0x00 ≤ register value ≤ = 0xFF**. Input thresholds that result in conversions out of this range are set accordingly to either the maximum (0xFF) or minimum (0x00) register value. See the datasheet for detailed information about the threshold range and granularity the register can represent. The RAW feature verifies the value is successfully set.	

5.3.6.40 R_RAA489206_SetDSC

e_bfe_err_t R_RAA489206_SetDSC(st_bfe_ctrl_t * const p_ctrl, float current_ma)		
Set the Discharge Short-circuit Current (DSC) threshold in mA: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the current threshold **current_ma** in mA into mV using the shunt resistor value. 3. Convert the voltage in mV into its register value. 4. Set and write the short-circuit overcurrent threshold register.		
Returned values	BFE_SUCCESS	DSC threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	DSC threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This function expects a negative value, which indicates charge is drained from the battery pack. The DSC threshold is an 8-bit register with valid range **0x00 ≤ register value ≤ = 0xFF.** Input thresholds that result in conversions out of this range are set accordingly to either the maximum (0xFF) or minimum (0x00) register value. See the datasheet for detailed information about the threshold range and granularity the register can represent. The RAW feature verifies the value is successfully set.	

5.3.6.41 R_RAA489206_SetMaxVcellDeltaVoltage

e_bfe_err_t R_RAA489206_SetMaxVcellDeltaVoltage(st_bfe_ctrl_t * const p_ctrl, float cells_maxd_th_mv)		
Set the maximum cell voltages delta (Max V_{CELL} Delta) threshold in mV: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the voltage threshold **cells_maxd_th_mv** in mV into its register value. 3. Set and write the V_{CELL} Max Delta threshold register.		
Returned values	BFE_SUCCESS	V_{CELL} Max Delta threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	V_{CELL} Max Delta threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This function expects a positive value. The V_{CELL} Max Delta threshold is an 8-bit register with valid range **0x00 ≤ register value ≤ = 0xFF**. Input thresholds that result in conversions out of this range are set accordingly to either the maximum (0xFF) or minimum (0x00) register value. See the datasheet for detailed information about the threshold range and granularity the register can represent. The RAW feature verifies the value is successfully set.	

5.3.6.42 R_RAA489206_SetCellUndervoltage

e_bfe_err_t R_RAA489206_SetCellUndevoltage(st_bfe_ctrl_t * const p_ctrl, float cells_uv_th_mv)		
Set the Cell Undervoltage (V_{CELL} UV) threshold in mV: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the voltage threshold **cells_uv_th_mv** in mV into its register value. 3. Set and write the V_{CELL} UV threshold register.		
Returned values	BFE_SUCCESS	V_{CELL} UV threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	V_{CELL} UV threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This function expects a positive value. The V_{CELL} UV threshold is an 8-bit register with valid range **0x00 ≤ register value ≤ = 0xFF.** Input thresholds that result in conversions out of this range are set accordingly to either the maximum (0xFF) or minimum (0x00) register value. See the datasheet for detailed information about the threshold range and granularity the register can represent. The RAW feature verifies the value is successfully set.	

5.3.6.43 R_RAA489206_SetCellOvervoltage

e_bfe_err_t R_RAA489206_SetCellOvervoltage(st_bfe_ctrl_t * const p_ctrl, float cells_ov_th_mv)		
Set the Cell Overvoltage (V_{CELL} OV) threshold in mV: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the voltage threshold **cells_ov_th_mv** in mV into its register value. 3. Set and write the V_{CELL} OV threshold register.		
Returned values	BFE_SUCCESS	V_{CELL} OV threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	V_{CELL} OV threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This function expects a positive value. The V_{CELL} OV threshold is an 8-bit register with valid range **0x00 ≤ register value ≤ = 0xFF**. Input thresholds that result in conversions out of this range are set accordingly to either the maximum (0xFF) or minimum (0x00) register value. See the datasheet for detailed information about the threshold range and granularity the register can represent. The RAW feature verifies the value is successfully set.	

5.3.6.44 R_RAA489206_SetVpackUndervoltage

e_bfe_err_t R_RAA489206_SetVpackUndevoltage(st_bfe_ctrl_t * const p_ctrl, float vpack_uv_th_mv)		
Set the V_{BAT1} (V_{PACK}) Undervoltage (V_{PACK} UV) threshold in mV: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the voltage threshold **vpack_uv_th_mv** in mV into its register value. 3. Set and write the V_{BAT1} UV threshold register.		
Returned values	BFE_SUCCESS	V_{PACK} UV threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	V_{PACK} UV threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This function expects a positive value. The V_{BAT1} (V_{PACK}) UV threshold is an 8-bit register with valid range **0x00 ≤ register value ≤ = 0xFF**. Input thresholds that result in conversions out of this range are set accordingly to either the maximum (0xFF) or minimum (0x00) register value. See the datasheet for detailed information about the threshold range and granularity the register can represent. The RAW feature verifies the value is successfully set.	

5.3.6.45 R_RAA489206_SetVpackOvervoltage

e_bfe_err_t R_RAA489206_SetVpackOvervoltage(st_bfe_ctrl_t * const p_ctrl, float vpack_ov_th_mv)		
Set the V_{BAT1} (V_{PACK}) Overvoltage (V_{PACK} OV) threshold in mV: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the voltage threshold **vpack_ov_th_mv** in mV into its register value. 3. Set and write the V_{BAT1} OV threshold register.		
Returned values	BFE_SUCCESS	V_{PACK} OV threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	V_{PACK} OV threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This function expects a positive value. The V_{BAT1} (V_{PACK}) OV threshold is an 8-bit register with valid range **0x00 ≤ register value ≤ = 0xFF**. Input thresholds that result in conversions out of this range are set accordingly to either the maximum (0xFF) or minimum (0x00) register value. See the datasheet for detailed information about the threshold range and granularity the register can represent. The RAW feature verifies the value is successfully set.	

5.3.6.46 R_RAA489206_SetInternalOTFault

e_bfe_err_t R_RAA489206_SetInternalOTFault(st_bfe_ctrl_t * const p_ctrl, float fault_temperature)		
Set the Internal Over-Temperature Fault (IOTF) threshold in °C: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the voltage threshold **fault_temperature** in °C into its register value. 3. Set and write the IOTF threshold register.		
Returned values	BFE_SUCCESS	IOTF threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	IOTF threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	The IOTF register can represent temperatures between -63.7°C and +151.1 °C. Out-of-range thresholds are set to the closest range limit. The RAW feature verifies the value is successfully set.	

5.3.6.47 R_RAA489206_SetInternalOTWarning

e_bfe_err_t R_RAA489206_SetInternalOTWarning(st_bfe_ctrl_t * const p_ctrl, float war_temperature)		
Set the Internal Over-Temperature Warning (IOTW) threshold in °C: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Convert the voltage threshold **war_temperature** in °C into its register value. 3. Set and write the IOTW threshold register.		
Returned values	BFE_SUCCESS	IOTW threshold successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	IOTW threshold has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	The IOTW register can represent temperatures between -63.7°C and +151.1 °C. Out-of-range thresholds are set to the closest range limit. The RAW feature verifies the value is successfully set.	

5.3.6.48 R_RAA489206_SetMode

e_bfe_err_t R_RAA489206_SetMode (st_bfe_ctrl_t * const p_ctrl, e_bfe_mode_t e_bfe_mode)		
Set the BFE to the mode specified by the enumeration **e_bfe_mode**: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the scan operation register (RBW). 3. Map the enumeration to the BFE mode value 4. Set the bits-field SYS_MODE of the scan operation register to the mapped value. 5. Write the scan operation register.		
Returned values	BFE_SUCCESS	BFE mode successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	BFE mode has not been set
	BFE_ERR_NONSUPPORTED_MODE	Specified mode is not supported by the BFE
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.49 R_RAA489206_ConfigLowPowerMode

e_bfe_err_t R_RAA489206_ConfigLowPowerMode(st_bfe_ctrl_t * const p_ctrl, bfe_lpm_cfg_t * const p_lpm_options)		
Setup BFE Low Power Mode (LPM) settings according to the instance-defined options **p_lpm_options** points to: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Cast the pointer **p_lpm_options** to point to a structure of the instance-defined type **st_raa4892206_lpm_cfg_t**. 3. Set the Low Power Timer value: a. Read the scan operation register (RBW). b. Set the Low Power Timer (LPT) bits-field of the scan operation register. c. Write the scan operation register. 4. Set the Low Power Regulator option: a. Read the V_{REG} operation register (RBW). b. Set the Low Power Regulator (LP_REG) bit of the V_{REG} operation register. c. Write the V_{REG} operation register. 5. Enable/disable the communication time-out: a. Read the V_{BAT1} operation register (RBW) b. Set the communication time-out enable bit (COMTO_EN) of the V_{BAT1} operation register. c. Write the V_{BAT1} operation register. 6. Set load detection in LPM: a. Read the load charge operation register. b. Set the Low detection in Low Power (LDLP) bit of the load charge operation register. c. Write the load charge operation register.		
Returned values	BFE_SUCCESS	LPM settings successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	LPM settings have not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.50 R_RAA489206_StartLowPowerMode

e_bfe_err_t R_RAA489206_StartLowPowerMode(st_bfe_ctrl_t *const p_ctrl)		
Set the BFE to Low Power Mode (LPM) operation: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the scan operation register. 3. Set the SYS_MODE bits-field of the scan operation register to the value of the enumeration **RAA489206_SYSTEM_MODE_LPM (0x02)**. 4. Write the scan operation register.		
Returned values	BFE_SUCCESS	LPM successfully set
	BFE_ERR_COM_READ_AFTER_WRITE_FAILED	LPM has not been set
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.51 R_RAA489206_GetDieInfomation

e_bfe_err_t R_RAA489206_GetDieInformation(st_bfe_ctrl_t * const p_ctrl, st_bfe_information_t * p_information)		
Read and return the Die information storing its ID, Revision, manufacturing ID and nickname in the structure **p_information** points to: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Read the die information register. 3. Set the fields **device_id** and **die_revision** of the structure pointed by **p_information** *to the* ID, and REVISION bits-fields of the read die information register. 4. Set **manufacturing_id** and **nickname** fields with the constants values defined for RA489206 BFE: **0x00** and the enumeration **BFE_NICKNAME_RAA206,** respectively.		
Returned values	BFE_SUCCESS	Die information successfully retrieved
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	---	

5.3.6.52 R_RAA489206_ReadRegister

e_bfe_err_t R_RAA489206_ReadRegister(st_bfe_ctrl_t * const p_ctrl, st_bfe_register_t * const p_bfe_register)		
Read and return the BFE register the structure **p_bfe_register** points to: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Call the control structure function **p_readRegisterValues** using the following parameters: 3. **p_bfe_register->address:** register address. 4. **p_bfe_register->p_value:** pointer to the value that stores the read value 5. **p_bfe_register->size:** register size in Bytes.		
Returned values	BFE_SUCCESS	Register value successfully retrieved
	BFE_ERR_INVALID_POINTER	**p_bfe_register->p_value** is a null pointer
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	The caller function must ensure the pointer **p_bfe_register->p_value** points to a valid address. To meet this requirement, all calls to **R_RAA489206_ReadRegister** in this sample code, use registers of the register bank, which point to global definitions of the BFE registers.	

5.3.6.53 R_RAA489206_ReadAllRegisters

e_bfe_err_t R_RAA489206_ReadAllRegisters(st_bfe_ctrl_t * const p_ctrl)		
Read all BFE registers and store their values in the global definitions to which the registers bank fields point: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Wait until the device is available by calling **wait_for_free**. This attempts to avoid reading registers while the device is processing any task. 3. Read all BFE registers by using either the CRC Read command **RAA489206_CRC_COMMAND_ALL_REGISTERS (0x9A)** or iterative calls to the function **R_RAA489206_ReadRegister**		
Returned values	BFE_SUCCESS	BFE registers successfully retrieved
	BFE_WARN_BUSY	BFE registers have been retrieved while the device was busy
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	This routine attempts to read BFE registers while the device is available (this is, the busy bit is 0), so that all registers contain updated values. However, the **wait_for_free** routine does not guarantee the device is free after returning to avoid blocking the program flow. Therefore, if the device is busy after **wait_for_free** returns, this routine generates the code **BFE_WARN_BUSY** to alert the caller function to possible outdated registers values.	

5.3.6.54 R_RAA489206_WriteRegister

e_bfe_err_t R_RAA489206_WriteRegister(st_bfe_ctrl_t * const p_ctrl, const st_bfe_register_t * const p_bfe_register)		
Write the BFE register the structure **p_bfe_register** points to: 1. Cast pointer **p_api_ctrl** to point to a structure of type **st_raa489206_ctrl** and verify the device has been initialized. 2. Call the control structure function **p_writeRegisterValues** using the following parameters: 3. **p_bfe_register->address:** register address. 4. **p_bfe_register->p_value:** pointer to the value to be written in the BFE. 5. **p_bfe_register->size:** register size in Bytes.		
Returned values	BFE_SUCCESS	Register value successfully retrieved
	BFE_ERR_INVALID_POINTER	**p_bfe_register->p_value** is a null pointer
	BFE_ERR_FSP_ERROR	FSP module error
	BFE_ERR_DEVICE_NOT_INITIALIZED	**R_RAA489206_Init** has not been called
Observations	The caller function must ensure the pointer **p_bfe_register->p_value** points to a valid address. To meet this requirement, all calls to **R_RAA489206_WriteRegister** in this sample code, use registers of the register bank, which point to global definitions of the BFE registers.	

6. Sample Battery Management System

6.1 Overview

The command **bmsdemo** starts the execution of the sample BMS application, which demonstrates the use of the RAA489206 BFE features to

- Monitor the status of a battery pack
- Measure and report periodically voltage, current and temperature
- Protect the battery pack against faults
- Reduce power consumption when no load is present
- Detect automatically load presence and restart normal operation

The sample BMS application sets up the BFE to monitor a battery pack using continuous scan operation and assert the ALERT pin to inform the MCU about the occurrence of events of interest.

The BMS application executes the following sequence:

1. Unmask all priority, etaux, other and status faults bits enabling them to assert the ALERT pin.
2. Set fault delays to 0.
3. Enable the MCU pin connected to the BFE ALERT pin to generate IRQ interruptions.
4. Set the BFE to continuous scan operation.
5. Command the BFE to start continuous scan operation.
6. Set the MCU to SLEEP MODE.
7. The MCU (application) exists SLEEP MODE and enters NORMAL MODE (Program execution State) when the BFE triggers an IRQ interruption in the MCU by asserting the RESET pin.
8. Iterate over call-back routines that verify the status of the battery pack, report measurements via CLI, and execute actions when necessary.
9. Goes to Step 5.
10. Terminate application execution when CTRL + C received over CLI.

The header file **r_bms.h** and source code **r_bms.c** implement the sample BMS application. The following sections detail these files.

6.2 Header File r_bms.h

This header file includes the external header files imported by the implementation, the signatures of functions defined in **r_bms.c,** and the declaration of the type definitions **st_monitoring_callbacks_t** and **u_monitoring_callbacks_t.** The structure type **st_monitoring_callbacks_t** contains members with pointers to call-back routines, which monitor the battery in response to events and status indicated by BFE status and fault registers. The union **u_monitoring_callbacks_t** allows the application to iterate over the structure call-backs as an array.

6.3 Source Code r_bms.c

6.4 Declarations

The first block of the source code defines variables and functions used in the application:

Static variables to share events and data between functions:

```
static bool s_alert_pin_asserted = false;
static bool s_monitor_timer_to = false;
static st_bms_measurements_t s_bms_measurements;
static const uint8_t s_read_allregisters_retries = 5;
static e_bfe_mode_t s_mode;
static st_raa489206_status_t s_device_status;
```

Declaration of call-back functions, which are the kernel routines that implement the functionalities demonstrated by this sample BMS application. Call-backs monitor BFE measurements, status and faults registers to determine the occurrence of an event (or events) of interest, and report battery pack state and properties such as cells/pack voltages, and pack current.

```
/*--------------------------------------------------------------------------------------
------------
    Callbacks to monitor BFE status and faults
--------------------------------------------------------------------------------------
----------*/
/*Discharge short-current*/
static bool dsc_callback (const st_bfe_instance_t * p_args);
/*Read all registers*/
static bool read_registers_callback (const st_bfe_instance_t * p_args);
/*Discharge overcurrent*/
static bool doc_callback (const st_bfe_instance_t * p_args);
/*Charge overcurrent*/
static bool coc_callback (const st_bfe_instance_t * p_args);
/*Cells undervoltage*/
static bool vcell_uv_callback (const st_bfe_instance_t * p_args);
/*Pack undervoltage*/
static bool vpack_uv_callback (const st_bfe_instance_t * p_args);
/*Cells overvoltage*/
static bool vcell_ov_callback (const st_bfe_instance_t * p_args);
/*Pack overvoltage*/
static bool vpack_ov_callback (const st_bfe_instance_t * p_args);
/*Maximum cells voltage delta*/
static bool delta_vcell_ov_callback (const st_bfe_instance_t * p_args);
/*Internal over-temperature warning*/
static bool iotw_callback (const st_bfe_instance_t * p_args);
/*Internal over-temperature fault*/
```

```
static bool iotf_callback (const st_bfe_instance_t * p_args);
/*Read all measurements*/
static bool read_measurements (const st_bfe_instance_t * p_args);
/*Read device status*/
static bool read_status_callback (const st_bfe_instance_t * p_args);
```

Call-backs structure and array where pointers to call-backs functions are stored. The structure **st_monitoring_callbacks_t callbacks_prio** stores call-backs pointers in the fields **.p_prioN**, where N = 0, 1, 2, ...15 is the callback priority. The union **u_monitoring_callbacks_vector_t callbacks_vector** allows iterating over call-back pointers to implement a simple Circular First-Come,First-Served (CFCFS) scheduling scheme for call-backs prioritization. The details and implications of this scheduling scheme are described in The bms_main Function. The Boolean return value indicates whether the call-back has detected a critical fault (return bool = true).

```
static const st_monitoring_callbacks_t callbacks_prio =
{
 .p_prio0 = dsc_callback,
 .p_prio1 = read_registers_callback,
 .p_prio2 = doc_callback,
 .p_prio3 = coc_callback,
 .p_prio4 = vcell_ov_callback,
 .p_prio5 = vpack_ov_callback,
 .p_prio6 = vcell_uv_callback,
 .p_prio7 = vpack_uv_callback,
 .p_prio8 = read_other_callback,
 .p_prio9 = iotf_callback,
 .p_prio10 = delta_vcell_ov_callback,
 .p_prio11 = iotw_callback,
 .p_prio12 = read_measurements,
 .p_prio13 = read_status_callback,
 .p_prio14 = NULL,
 .p_prio15 = NULL,
};

static const u_monitoring_callbacks_vector_t callbacks_vector =
{
 .callbacks = callbacks_prio,
};
```

6.5 The bms_main Function

The CLI calls the execution of this function when the **bmsdemo start** command is entered by the user. As it has been mentioned previously, call-backs routines are the kernel of the sample BMS application. The following code extract highlights the lines that implement the iterative calls to them (this is, the CFCFS scheduling scheme).

```
e_bfe_err_t bms_start(void)
{
    /*Variables Initialization*/
    ...

    /*Read all registers to ensure that code that uses register bank mirror gets
updated values*/
    raa489206_error = g_bfe_raa489206.p_api-
>p_readAllRegisters(g_bfe_raa489206.p_ctrl);

    BFE_ERROR_RETURN(raa489206_error == BFE_SUCCESS, raa489206_error);

    /*MCU Settings: LPM and IRQ modules*/
    ...

    /*Start continuous scan operation*/
    raa489206_error = g_bfe_raa489206.p_api-
>p_startContinuousScan(g_bfe_raa489206.p_ctrl, &g_bfe_cs_config);

    BFE_ERROR_RETURN(raa489206_error == BFE_SUCCESS, raa489206_error);

    /*Turn Charge pump on*/
    g_pwr_fet_op_register.value_b.CPMP_EN = 1;
    raa489206_error = R_RAA489206_WriteRegister(g_bfe_raa489206.p_ctrl,
&(g_bfe_raa489206_ctrl.p_regs->power_fet_operation));

    uint8_t num_prio =
sizeof(callbacks_vector.prioCallbacks)/sizeof(callbacks_vector.prioCallbacks[0]);

    while (1)
    {
        /*MCU Sleep Mode*/
if ((g_alert_pin_asserted == false) && (get_uart_transmission() != true) &&
(get_uart_reception() != true))
        {
            fsp_error = g_lpm.p_api->lowPowerModeEnter(g_lpm.p_ctrl);

            BFE_ERROR_RETURN(fsp_error == FSP_SUCCESS, raa489206_error);
        }

   /*Wake up: has ALERT pin been asserted?*/
        if (g_alert_pin_asserted)
        {
            g_alert_pin_asserted = false;

            s_fault_detected = false;

                for (uint8_t i = 0; i < num_prio; i++ )
```

```
                {
                    if (callbacks_vector.prioCallbacks[i] != NULL)
                    {
                        if ((*callbacks_vector.prioCallbacks[i])(&g_bfe_raa489206)
== true)
                        {
                            critical_fault_detected(&g_bfe_raa489206);
                        }
                    }
                }

    /*Stop execution in a safe state if faults has been detected?*/

                if ((s_fault_detected == false) && (p_gtp_ctrl->open == GPT_OPEN))
                {
                    g_timer.p_api->stop(g_timer.p_ctrl);
                    g_timer.p_api->close(g_timer.p_ctrl);
                    g_ioport.p_api->pinWrite(g_ioport.p_ctrl, LED_PIN,
BSP_IO_LEVEL_HIGH);
                }

            if (s_fault_detected == true)
            {
                R_BFE_CLI_Printf("\nFault Detected: Press CTRL + C to exit");
            }
        }

        R_BFE_CLI_Process();

        if (s_exit == true)
        {
            break;
        }
    }

    return BFE_SUCCESS;
}
```

Because function pointers stored in the **callbacks_vector** union are called sequentially from the 0th to 15th array element, the CFCFS scheduling scheme assigns higher priority to lower array positions (this is, lower priority structure fields **.p_prioN**). The priority vector adopted by this sample BMS application, its call-backs tasks and actions, and observations on call-backs implementations are summarized in Table 9.

Table 9. Summary of BMS Vector: Task, Actions and Observations

Priority	Callback	Task	Actions	Observations
0 (highest)	dsc_callback	Monitor Discharge Short-Circuit overcurrent event	Turns DFET and CFET off and returns true if event is detected.	DSC is a user-safety-critical event triggered by an analog comparator that does not depend on scan measurements
1	read_registers_callback	Update MCU registers bank	Waits for BFE to be available and reads all registers	It aims to provide call-backs with updated registers values
2	doc_callback	Monitor Discharge Overcurrent event	Turns DFET off and returns true if event is detected. Clears fault if event is not detected.	DOC causes overheating of battery pack leading to cells deterioration or dangerous situations.
3	coc_callback	Monitor Charge Overcurrent event	Turns CFET off and returns true if event is detected. Clears fault if event is not detected.	This prevents from charging the battery pack with current above manufacturer specifications, which may lead to functional and safety issues.
4	vcell_ov_callback	Monitor cell overvoltage event	If event is detected: Turns CFET off during charge; Turns DFET on during discharge. Clears fault if event is not detected.	Cell OV may result in safety issues. Discharge may be allowed to enable cells to reduce their voltage
5	vpack_ov_callback	Monitor pack overvoltage	If event is detected: Turns CFET off during charge; Turns DFET off during discharge. Clears fault if event is not detected.	Pack OV may result in safety issues. Discharge may be allowed to enable the battery pack to reduce its voltage
6	vcell_uv_callback	Monitor cells undervoltage	If event is detected: Turns CFET on during charge; Turns DFET off during discharge. Clears fault if event is not detected.	Cells undervoltage reduces their capacity and life. Charge may be allowed to enable cells to increase their voltage
7	vpack_uv_callback	Monitor pack undervoltage	If an event is detected: Turns CFET on during charge; Turns DFET off during discharge. Clears fault if event is not detected.	Pack undervoltage reduces pack capacity and life. Charge may be allowed to enable pack to increase its voltage
8	read_other_callback	Monitor other faults: VCC Fault, Open Wire, ETAUX, Charge Pump not ready, Other Faults bit, Regulator Current 1 and 2 and VTMP Fault.	If any fault bit indicates an other fault has occurred, DFET and CFET are turned off.	Indicate whether internal IC faults occur or wires have been disconnected.

Table 9. Summary of BMS Vector: Task, Actions and Observations (Cont.)

Priority	Callback	Task	Actions	Observations
9	Iotf_callback	Monitor internal over-temperature	If event is detected: Turns DFET and CFET off Clears fault if event is not detected.	Over-temperature may indicate battery pack failures, BFE failures, or extreme environmental conditions under which the battery pack should not operate.
10	delta_vcell_ov_callback	Monitor maximum difference between cells voltages	This event is reported as critical fault	Excessive cell voltage unbalance limits the pack capacity to weak cells capacity, and deteriorates life and capacity of strong cells
11	iotw_callback	Monitor the over-temperature warning	Reports warning	It warns the BMS of possible excess of BFE internal temperature
12	read_measurements	Report pack voltage, cells voltages, pack current and other measurements	Reads and reports measurements registers	---
13	read_status_callbacks	Update registers, and report BFE status/faults and current mode	Reads all registers, status/fault registers and BFE mode	Previous call-backs may have modified registers and/or BFE operation mode, so this function updates the registers bank and verified whether a fault has set the device to IDLE mode. If it is, this callback set it back to SCAN mode and resume continuous scan operation.
14		User-defined		
15 (Lowest)		User-defined		

When the conditions described in the Scan Will Not Start section the RAA489206 datasheet are met, the ALERT pin can remain permanently asserted and the continuous scan operation may not start. Because the sample BMS application uses the IRQ interruption to exit from the MCU Low Power Mode operation, the permanent assertion of the ALERT pin avoids setting back the MCU to NORMAL MODE. The BMS application reports this event as a critical error, so it is necessary to exit the application entering CTRL+C, which clears the fault condition and sets the BFE into a safe state.

Entering the key sequence CTRL + C calls the function **bms_stop**:

```
void bms_stop(void)
{
    g_alert_irq.p_api->disable(g_alert_irq.p_ctrl);
    g_alert_irq.p_api->close(g_alert_irq.p_ctrl);
    gpt_instance_ctrl_t *p_gtp_ctrl = (gpt_instance_ctrl_t *) g_timer.p_ctrl;
    g_ioport.p_api->pinWrite(g_ioport.p_ctrl, LED_PIN, BSP_IO_LEVEL_HIGH);


    if (p_gtp_ctrl->open == GPT_OPEN)
    {
        g_timer.p_api->close(g_timer.p_ctrl);
    }
```

```
    g_bfe_raa489206.p_api->p_startSystemScan(g_bfe_raa489206.p_ctrl);

    g_bfe_raa489206.p_api->p_turnDfetOffCfetOff(g_bfe_raa489206.p_ctrl);

    g_bfe_raa489206.p_api->p_stopContinuousScan(g_bfe_raa489206.p_ctrl);

    static st_raa489206_status_t device_status;

    g_bfe_raa489206.p_api->p_readStatus(g_bfe_raa489206.p_ctrl, &device_status);

    R_BFE_CLI_Printf("\nAfter stopping scan mode-->prio_status = ");

    for (int8_t i = 7; i >= 0; i-- )
    {
        uint8_t bit = (device_status.priority_status.value >> i) & 1U;

        R_BFE_CLI_Printf("%d", bit);
    }

    R_BFE_CLI_Printf(" ; ");

    R_BFE_CLI_Printf(" COCF = %d ; ", device_status.priority_status.value_b.COCF);
    R_BFE_CLI_Printf(" DOCF = %d ; ", device_status.priority_status.value_b.DOCF);
    R_BFE_CLI_Printf(" DSCF = %d ; ", device_status.priority_status.value_b.DSCF);
    R_BFE_CLI_Printf(" IOTF = %d ; ", device_status.priority_status.value_b.IOTF);
    R_BFE_CLI_Printf(" OVF = %d ; ", device_status.priority_status.value_b.OVF);
    R_BFE_CLI_Printf(" OWF = %d ; ", device_status.priority_status.value_b.OWF);
    R_BFE_CLI_Printf(" UVF = %d ; ", device_status.priority_status.value_b.UVF);
    R_BFE_CLI_Printf(" VCCF = %d\n ", device_status.priority_status.value_b.VCCF);
    //g_bfe_raa489206.p_api->p_clearAllFaults(g_bfe_raa489206.p_ctrl);
    s_exit = true;

    g_bfe_raa489206.p_api->p_readAllRegisters(g_bfe_raa489206.p_ctrl);
}
```

This function finishes the BMS application as follows:

1. Disable and close the IRQ and timer modules of the MCU.
2. Execute a system scan to be able to read updated register value upon termination of the application.
3. Turn off CFET and DFET to set the BFE and the battery to a safe state.
4. Stop continuous scan.
5. Read the status of the BFE.
6. Print out the priority fault bits.

7. State-of-Charge Application

The CLI calls the execution of this function when the **soc <soci>** command is entered by the user. Refer to *R16AN0029: Coulomb Counting and State-of-Charge Estimation featuring the RAA489206/ISL94216A Battery Front End* for detailed information about the application implementation and how to use it.

8. CLI Commands List

This section lists commands available in the CLI, and describes their format, parameters and action on the BFE device. As convention, parameters within square brackets ([ ]) are mandatory, whereas parameters within angle brackets (< >) are optional.

8.1 BFE command group

The BFE command group allows you to:

- Read and set the voltage, current and temperature threshold registers.
- Read BFE faults and status flags.
- Start a complete system scan.
- Start and stop continuous scan operation, as well as set the scan delay.
- Manipulate the state of charge and discharge FETs by turning each individually on/off, or setting the state of both FETs with one command.

8.1.1 Initialize Device

init command		
Format	bfe init	
Parameters	none	---
Example	bfe init – Initializes the control structure of the BFE interface and setup the basic operating configuration settings of the device. This command must be executed before executing any other command.	

8.1.2 Discharge Overcurrent (DOC) Threshold

doc command		
Format	bfe doc <threshold>	
Parameters	<threshold>	<**none**> – Reads BFE DOC threshold <**Float < 0**> – Sets BFE DOC threshold in mA.The negative sign denotes charge is drained from the device.
Examples	bfe doc – Reads BFE DOC threshold bfe doc -300 – Sets DOC threshold to -300mA	

8.1.3 Charge Overcurrent (COC) Threshold

coc command		
Format	bfe coc <threshold>	
Parameters	<threshold>	<**none**> – Reads BFE COC threshold in mA <**Float > 0**> – Sets COC threshold in mA. Positive sign denotes charge is given to the device.
Examples	bfe coc – Reads BFE COC threshold bfe coc 800 – Sets COC threshold to 800mA	

8.1.4 Discharge Short-Circuit Current (DSC) Threshold

dsc command		
Format	bfe dsc <threshold>	
Parameters	<threshold>	<**none**> – Reads BFE DSC threshold in mA. <**Float < 0**> – Sets DSC threshold in mA. Negative sign denotes charge is drained from the device.
Examples	bfe dsc – Reads BFE DSC threshold bfe dsc -8000 – Sets DSC threshold to -8000mA	

8.1.5 Internal Over-Temperature Fault (IOTF) Threshold

iotf command		
Format	bfe iotf <threshold>	
Parameters	<threshold>	<**none**> – Reads IOTF threshold in °C. <**Float**> – Set IOTFs threshold in °C.
Examples	bfe iotf – Reads BFE IOTF threshold bfe iotf 95 – Sets IOTF threshold to 95°C	

8.1.6 Internal Over-Temperature Warning (IOTW) Threshold

iotw command		
Format	bfe iotw <threshold>	
Parameters	<threshold>	<**none**> – Reads BFE IOTW threshold in °C. <**Float**> – Sets IOTF threshold in °C.
Examples	bfe iotw – Reads BFE IOTW threshold bfe iotw 85 – Sets IOTW threshold to 85°C	

8.1.7 Maximum Cell Voltage Delta (MAXDELTA) Threshold

maxdelta command		
Format	bfe maxdelta <threshold>	
Parameters	<threshold>	<**none**> – Reads BFE MAXDELTA threshold in mV. <**Integer > 0**> – Sets MAXDELTA threshold in mV.
Examples	bfe maxdelta – Reads BFE MAXDELTA threshold bfe maxdelta 480 – Sets MAXDELTA threshold to 480mV	

8.1.8 Cell Overvoltage (VCELLOV) Threshold

vcellov command		
Format	bfe vcellov <threshold>	
Parameters	<threshold>	<**none**> – Reads BFE VCELLOV threshold in mV. <**Integer > 0**> – Sets VCELLOV threshold in mV.
Examples	bfe vcellov – Reads BFE VCELLOV threshold bfe vcellov 3000 – Sets VCELLOV threshold to 3000mV	

8.1.9 Cell Undervoltage (VCELLUV) Threshold

vcelluv command		
Format	bfe vcelluv <threshold>	
Parameters	<threshold>	<**none**> – Reads BFE VCELLUV threshold in mV. <**Integer > 0**> – Sets VCELLUV threshold in mV.
Examples	bfe vcelluv – Reads BFE VCELLUV threshold bfe vcelluv 2000 – Sets VCELLUV threshold to 2000mV	

8.1.10 Pack Overvoltage (VPACKOV) Threshold

vpackov command		
Format	bfe vpackov <threshold>	
Parameters	<threshold>	<**none**> – Reads BFE VPACKOV threshold in mV. <**Integer > 0**> – Sets VPACKOV threshold in mV.
Examples	bfe vpackov – Reads BFE VPACKOV threshold bfe vpackov 40000 – Sets VPACKOV threshold to 40000mV	

8.1.11 Pack Undervoltage (VPACKUV) Threshold

vpackuv command		
Format	bfe vpackuv <threshold>	
Parameters	<threshold>	<**none**> – Reads BFE VPACKUV threshold in mV. <**Integer > 0**> – Sets VPACKUV threshold in mV.
Example	bfe vpackuv – Reads BFE VPACKUV threshold bfe vpackuv 30000 – Sets VPACKUV threshold to 30000mV	

8.1.12 Thresholds

thresholds command		
Format	bfe thresholds	
Parameters	none	---
Example	bfe thresholds – Reads all BFE thresholds	

8.1.13 BFE status

status command		
Format	bfe status <-t> bfe status clrfaults	
Parameters	<-t> clrfaults	Triggers a system scan before reading BFE status. Clears all status and faults bits
Example	bfe status – Reads priority and general status flags bfe status -t – Triggers a complete system scan and read, after completion of scan measurements, priority and general status flags. bfe status clrfaults – Clears all bits in faults and status registers. All bits in registers 0x63-0x69, except for 0x67.5 - CH PRESI, along with all counters are cleared (set to 0).	

8.1.14 Scan

<table>
<tr><th colspan="3">scan command</th></tr>
<tr><td>Format</td><td colspan="2">bfe scan <option> <value></td></tr>
<tr><td rowspan="2">Parameters</td><td><option></td><td>Select one of the following sub-command options:
<none> – Starts a complete system scan.
start – Starts continuous operation
stop – Stops continuous operation
delay – Sets the scan delay given as parameter <value> in ms.
Note: <none> and delay are allowed to be executed only in single scan operation. Trying to execute this action in continuous scan operation generates a command error execution.</td></tr>
<tr><td><value></td><td>scan delay in ms for the continuous scan operation when <option> = delay. Available delays in ms: 64, 128, 256, 512, 1024, 2048 and 4096. Specifying an unavailable delay or no value results in execution error.
Note: This option stops continuous scan operation before setting the delay value! Use the start option of the scan command to restart continuous scan operation.</td></tr>
<tr><td>Example</td><td colspan="2">bfe scan – Starts a complete system scan
bfe scan start – Starts continuous scan
bfe scan stop – Stops continuous scan operation
bfe scan delay – Reads the current scan delay
bfe scan delay 256 – Sets the scan delay to 256</td></tr>
</table>

8.1.15 FETs Commands

fets commands		
Format	bfe cfet <value> bfe dfet <value> bfe fets <value> bfe cfet [value1] dfet [value2] bfe dfet [value1] cfet [value2]	
Parameters	[value(1/2)]	Select one of the following values: <1> or <on>: Turns FET on <0> or <off>: Turns FET off
Example	bfe cfet bfe dfet Reads CFET/DFET status bfe cfet 1 bfe cfet on Turns CFET on bfe cfet 0 bfe cfet off Turns CFET off bfe dfet Reads DFET status bfe dfet 1 bfe dfet on Turns DFET on bfe dfet 0 bfe dfet off Turns DFET off bfe fets 1 bfe fets on Turns both DFET and CFET on bfe fets 0 bfe fets off Turns DFET and CFET off bfe cfet 1 dfet 1 bfe dfet 1 cfet 1 Turns both DFET and CFET on bfe cfet 0 dfet bfe dfet 0 cfet 0 Turns both DFET and CFET off bfe cfet 1 dfet 0 bfe dfet 0 cfet 1 Turns DFET off and CFET on bfe cfet 0 dfet 1 bfe dfet 1 cfet 0 Turns DFET on and CFET off	

8.1.16 Mode

mode command		
Format	bfe mode <mode>	
Parameters	<mode>	Select one of the following modes: **<none>** – Reads BFE mode **scan** – Sets the BFE to scan mode **idle** – Sets the BFE to idle mode **lpm** – Sets the BFE to lpm mode **ship** – Sets the BFE to ship mode
Example	bfe mode – Reads and returns the current BFE mode bfe mode scan – Sets the BFE device to SCAN mode	

8.1.17 Cells Count - Cells Select

Cells Count Command		
Format	bfe cellcount <num>	
Parameters	<num>	**<none>** – Reads the cells existing in the BMS **<0 < Integer < 17>** – Sets the number of cells existing in the BMS
Example	bfe cellcount 10 – Sets the bits of the Cell Select register (0x04-0x05) that correspond to 10 cells according to the Figure 131. Cell Count Matrix of the datasheet.	

8.1.18 Shunt Resistor Value

Shunt Resistor Command		
Format	bfe rshunt <res>	
Parameters	<res>	**<none>** – Reads the resistance value of the shunt resistor in mΩ. **<Float > 0>** – Sets the resistance value in mΩ of the shunt resistor.
Examples	bfe rshunt – Reads the shunt resistance value in mΩ bfe rshunt 30 – Sets the shunt resistance value to 30mΩ, which is used to calculate current readings and thresholds in mA.	

8.2 Register (REG) Command

The register (REG) command allows the user to read or write BFE registers specifying their hexadecimal address.

8.2.1 Read Register

read register command		
Format	reg [register-address] <-b, all> <-s>	
Parameters	[register-address]	Hexadecimal value 0xXX of the register address
	-b	Prints register value in hexadecimal and binary 0xXXXXXXXX formats.
	all	Prints all registers as a list grouped according to Table 2. System Registers of the datasheet
	-s	Prints all resisters sorted from lowest to highest register address.
Example	reg 0x01 – Reads the BFE register with address 0x01 (Global Operation Register) reg 0x01 -b – Reads the BFE register with address 0x01 and prints its value in hexadecimal and binary formats. reg all – Reads all registers sorted as in Table 2. System Registers of the datasheet. reg all -s – Reads all registers sorted from lowest to highest register address.	

8.2.2 Write Register

write register command		
Format	reg [register-address] [hex-value]	
Parameters	[register-address] [hex-value]	Hexadecimal value **0xXX** of the register address Hexadecimal value **0xXX** to be written
Example	reg 0x01 0x8F – Writes the value 0x8F in the BFE register with address 0x01 (Global Operation Register)	

8.3 Measurement (MEAS) Command Group

The Measurement (MEAS) command group allows the user to read and trigger voltage, currents, and internal temperature measurements.

8.3.1 Vpack

vpack command		
Format	reg Vpack <-t>	
Parameters	<-t>	Triggers a V_{BAT1} (Vpack) measurement before reading the BFE Vpack register
Example	reg Vpack – Reads pack voltage register in mV reg Vpack -t – Triggers a pack voltage measurement and read the measured value in mV after its completion	

8.3.2 Ipack

ipack command		
Format	reg Ipack <-t>	
Parameters	<-t>	Triggers a current measurement before reading the BFE Ipack register.
Example	meas Ipack – Reads BFE Ipack register value in mA meas Ipack -t – Triggers an Ipack measurement and read the measured value in mA after its completion	

8.3.3 Vcells

vcells command		
Format	meas vcells <-t>	
Parameters	<-t>	Triggers cells voltage measurements before reading V_{CELL} registers.
Example	meas vcells – Reads cells voltage registers in mV meas vcells -t – Triggers cell voltage measurements and read values in mV after their completion	

8.3.4 Vcell N

Vcell N command		
Format	meas vcell [cell_number] <-t>	
Parameters	[cell_number]	Number of the cell (1-16)
	<-t>	Triggers measurement of cells voltages before reading the voltage register of the specified cell.
Example	meas vcell 5 – Reads voltage in mV of cell 5 meas vcell 5 -t – Triggers measurement of cells voltages and read the voltage of cell 5 in mV after measurements completion.	

8.3.5 Total Cell Voltage

totvcells command		
Format	meas totvcells <-t>	
Parameters	<-t>	Triggers measurement of cells voltages before reading and summing their voltages
Example	meas totvcells – Reads the cells voltage register and returns the total sum of their values in mV meas totvcells -t – Triggers measurement of cells voltages and return the total sum of cells voltages in mV after measurements completion.	

8.3.6 Internal Temperature

itemp command		
Format	meas itemp <-t>	
Parameters	<-t>	Triggers a measurement of the internal temperature before reading BFE internal temperature register.
Example	meas itemp -t – Triggers measurement of internal temperature and return the measured value in °C after its completion.	

8.3.7 Regulator Voltage

vreg command		
Format	meas vreg <-t>	
Parameters	<-t>	Triggers a measurement of the internal temperature before reading BFE internal temperature register.
Example	meas vreg -t – Starts measurements of V_{CC}, V_{TEMP} and I_{REG}, and return the measured value of V_{REG} in mV after its completion.	

8.3.8 Regulator Current

ireg command		
Format	meas ireg <-t>	
Parameters	<-t>	Triggers a measurement of other measurements before reading BFE regulator register.
Example	meas ireg -t – Starts measurements of Vcc, Vtemp and Ireg, and return the measured value of Ireg in mA after its completion.	

8.4 Cell Balancing Command Group

The cell balancing command group allows you to set the configuration settings of cell balancing functionalities and trigger cell balancing.

8.4.1 Cell Balancing Enable/Disable

cb enable/disable command		
Format	cb <enable/disable>	
Parameters	<enable/disable>	**<none>** – Reads the value of the Cell Balancing Enable bit (CB EN 0x25.7) **<enable>** – Enables cell balancing by setting CB EN. **<disable>** – Disables cell balancing by clearing CB EN
Examples	cb – Reads the value of CB EN cb enable – Sets CB EN cb disable – Clears CB EN	

8.4.2 End-of-Charge Voltage

veoc command		
Format	cb veoc <value>	
Parameters	<value>	**<none>** – Reads the end-of-charge voltage in mV **<Float > 0>** – Sets the end-of-charge voltage to value in mV.
Examples	cb veoc – Reads the end-of-charge voltage in mV cb veoc 4900 – Sets the end-of-charge voltage to 4900 mV	

8.4.3 End-of-Charge Current

ieoc command		
Format	cb ieoc <value>	
Parameters	<value>	**<none>** – Reads the end-of-charge current in mA **<Float > 0>** – Sets the end-of-charge current to value in mA
Examples	cb ieoc – Reads the end-of-charge current in mA cb ieov 20 – Sets the end-of-charge current to 20 mA	

8.4.4 Automatic Cell Balancing Enable/Disable

cb auto enable/disable command		
Format	cb auto <enable/disable>	
Parameters	<enable/disable>	**<none>** – Reads the value of the Automatic Cell Balancing Enable bit (Auto CB EN 0x25.6) **<enable>** – Enables automatic cell balancing by setting Auto CB EN. **<disable>** – Disables automatic cell balancing by clearing Auto CB EN **<int>** – Sets the BFE to use internal FETs to perform cell balancing by clearing the CB Configuration bit
Examples	cb auto – Reads the value of auto CB EN cb auto enable – Sets auto CB EN cb auto disable – Clears auto CB EN	

8.4.5 Cell Balancing FETs Configuration

cb fets configuration command		
Format	cb fets <ext/int>	
Parameters	<ext/int>	**<none>** – Reads the value of the cell balancing enable configuration bit (CB Configuration 0x25.5) **<ext>** – Sets the BFE to use external FETs to perform cell balancing by setting the CB Configuration bit..
Examples	cb fets – Reads the FETs configuration selected to perform cell balancing cb fets ext – Sets external FETs cb fets int – Sets internal FETs	

8.4.6 Cell Balancing Trigger

cb trigger command		
Format:	cb trigger	
Parameters	none	---
Examples	cb trigger – Triggers one cell balancing cycle	

8.4.7 Cell Balancing Mask

cb mask command		
Format	cb mask <enable/disable>	
Parameters	<enable/disable>	**<none>** – Reads value of the Cell Balancing Mask bit (CB Mask 0x25.2) **<enable>** – Enables mask feature, which prevents adjacent cells from balancing at the same time **<disable>** – Disables masking feature
Examples	cb mask – Reads the value of the CB Mask bit cb mask enable – Does not allow adjacent cells to balance at the same time cb mask enable – Allows adjacent cells to balance at the same time	

8.4.8 Cell Balancing End-of-Charge Enable/Disable

cb ieoc enable/disable command		
Format	cb eoc <enable/disable>	
Parameters	<enable/disable>	**<none>** – Reads the value of the cell balance end-of-charge bit (CB EOC 0x25.1) **<enable>** – Enables cell balancing after one or more cells reach the VEOC threshold. **<disable>** – Disables cell balancing when any cell voltage reaches the VEOC threshold
Examples	cb eoc – Reads the value of auto CB EOC cb eoc enable – Enables cell balancing after cell voltages reach VEOC threshold cb eoc disable – Disables cell balancing after any cell voltage reaches VEOC threshold	

8.4.9 Current End-Of-Charge Enable/Disable

cb ieoc_en enable/disable command		
Format	cb ieoc_en <enable/disable>	
Parameters	<enable/disable>	**<none>** – Read the value of the Current End-Of-Charge Enable bit (IEOC EN 0x25.3) **<enable>** – Charging stops after the VEOC bit is set and the charge current drops below the IEOC Threshold. **<disable>** – Charging ends when any cell voltage is above VEOC
Examples	cb ieoc_en – Reads the value of auto IEOC EN cb ieoc_en enable – Sets IEOC EN cb ieoc_en disable – Clears IEOC EN	

8.4.10 Cell Balancing Charge Enable/Disable

cb chrg enable/disable command		
Format	cb chrg <enable/disable>	
Parameters	<enable/disable>	**<none>** – Reads the value of the Cell Balancing Charge bit (CB CHRG 0x25.0) **<enable>** – Charging stops after the VEOC bit is set and the charge current drops below the IEOC Threshold. **<disable>** – Charging ends when any cell voltage is above VEOC
Examples	cb ieoc_en – Reads the value of CB CHRG cb ieoc_en enable – Sets CB CHRG cb ieoc_en disable – Clears CB CHRG	

8.4.11 Cell Balancing Cell State

cb cell_state command		
Format	cb cell_state <hex>	
Parameters	<hex>	**<none>** – Reads Cell Balancing Cell State register (CB Cell State 0x26-27) **<hex>** – 16-bit hexadecimal value.
Examples	cb cell_state – Reads CB Cell State cb cell_state 0xF01F – Sets CB Cell state to 0xF01F	

8.4.12 Cell Balancing Minimum Delta Threshold

cb mindelta command		
Format	cb mindelta <value>	
Parameters	<value>	**<none>** – Reads the Cell Balancing Minimum Delta (CB Min Delta) Threshold in mV **<Float > 0>** – Sets CB Min Delta Threshold in mV to value in mV
Examples	cb mindelta – Reads CB Min Delta in mV cb mindelta 100 – Sets the CB Min Delta to 100mV	

8.4.13 Cell Balancing Maximum Threshold

cb max_th command		
Format	cb max_th <value>	
Parameters	<value>	**<none>** – Reads the Cell Balancing Maximum (CBMAX) Threshold in mV **<Float > 0>** – Sets CBMAX Threshold to value in mV
Examples	cb max_th – Reads CBMAX in mV cb max_th 4900 – Sets CBMAX to 4900 mV	

8.4.14 Cell Balancing Minimum Threshold

cb min_th command		
Format	cb min_th <value>	
Parameters	<value>	**<none>** – Reads the Cell Balancing Minimum (CBMIN) Threshold in mV. **<Float > 0>** – Sets CBMIN Threshold to value in mV
Examples	cb min_th – Reads CBMIN in mV cb min_th 3000 – Sets CBMIN to 3000 mV	

8.4.15 Cell Balancing On Timer

cb on_timer command		
Format	cb on_timer <value> <ms/s>	
Parameters	<value> <ms / s>	**<none>** – Reads Cell Balancing On Timer (CBON) **<0 < Integer < 1017>** – Sets CBON in ms or s <ms> milliseconds <s> seconds
Examples	cb on_timer – Reads CBON in ms or s cb on_timer 8 s – Sets CBON to 8 s cb on_timer 8 ms – Sets CBON to 8 ms	

8.4.16 Cell Balancing Off Timer

cb off_timer command		
Format	cb off_timer <value> <ms/s>	
Parameters	<value> <ms / s>	<**none**> – Reads Cell Balancing Off Timer (CBOFF) <**0 < Integer < 1017**> – Sets CBOFF in ms or s <ms> milliseconds <s> seconds
Examples	cb off_timer – Reads CBOFF in ms or s cb of_timer 8 s – Sets CBOFF to 8 s cb off_timer 8 ms – Sets CBOFF to 8 ms	

8.5 Sample Battery Management System

bmsdemo command		
Format	bmsdemo	
Parameters	---	Starts the sample Battery Management System application
Example	bmsdemo – Starts the sample BMS using the current BFE protection settings (such as vpackov, vpackuv, dsc-threshold).	

8.6 State-of-Charge Application

This command starts the State-of-Charge (SOC) Estimation application. Refer to *R16AN0029: Coulomb Counting and State-of-Charge Estimation featuring the RAA489206/ISL94216A Battery Front End* for detailed information about the application implementation and how to use it.

soc command		
Format:	soc <soci>	
Parameters	<soci>	**<none>** – Starts the SOC application by estimating the initial SOC of the battery pack **<0 ≤ Integer ≤ 100>** – Sets the initial SOC of the battery pack to soci in percent (%)
Examples	soc – Starts the SOC application performing first initial SOC estimation soc 45.5 – Starts the SOC application using 45.5 % as initial SOC	

9. Revision History

Revision	Date	Description
1.01	Jan 31, 2022	Updated Tables 1, 3, 7, 8, and 9. Updated codes in the Header File r_bfe_raa489206.h, Declarations, The bms_main Function sections. Added State-of-Charge Application, Cells Count - Cells Select, and Shunt resistor value sections. Added Cell Balancing Command Group section and subsections. Updated the following sections: ▪ R_RAA489206_Init ▪ R_RAA489206_StartContinuousScan ▪ R_RAA489206_StopContinuousScan ▪ R_RAA489206_ClearAllFaults ▪ R_RAA489206_ReadVcells
1.00	Aug 18, 2021	Initial release

IMPORTANT NOTICE AND DISCLAIMER

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