RAA489204 Battery Front End Sample Code

This software manual provides a detailed description and application guidelines for using the RAA489204 sample code. It includes Application Programming Interface (API) functions and application examples to speed up the design of high voltage battery management systems, consisting of multiple (stacked) battery manager ICs.

Contents

1. Introduction 3
1.1 Assumptions and Advisory Notes 3
2. RAA489204 Battery Front End Overview 3
2.1 Features 3
2.2 Applications 4
2.3 RAA489204 Sample Code Structure 4
3. RAA489204 Application Programming Interface Implementation 6
3.1 Control and Configuration Structures 6
3.2 Registers Bank 10
3.3 Private (Static) Functions 12
3.4 API Implementation 13
3.4.1 R_RAA489204_Init 13
3.4.2 R_RAA489204_Deinit 14
3.4.3 R_RAA489204_Setup 14
3.4.4 R_RAA489204_Reset 15
3.4.5 R_RAA489204_ModeSet 15
3.4.6 R_RAA489204_ModeRead 16
3.4.7 R_RAA489204_CommTest 16
3.4.8 R_RAA489204_SelfDiag 17
3.4.9 R_RAA489204_MemCheck 18
3.4.10 R_RAA489204_VPackGet 18
3.4.11 R_RAA489204_IPackGet 19
3.4.12 R_RAA489204_VoltagesGet 19
3.4.13 R_RAA489204_Temps 20
3.4.14 R_RAA489204_AllGet 21
3.4.15 R_RAA489204_VMixGet 22
3.4.16 R_RAA489204_FaultsAllRead 24
3.4.17 R_RAA489204_FaultsCheck 25
3.4.18 R_RAA489204_FaultsAllClear 25
3.4.19 R_RAA489204_CellBalanceCtrl 25
3.4.20 R_RAA489204_IsCellBalancing 28
3.4.21 R_RAA489204_ContScanCtrl 28
3.4.22 R_RAA489204_WatchdogCtrl 29
3.4.23 R_RAA489204_FETsCtrl 30
3.4.24 R_RAA489204_GPIOsCtrl 31
3.4.25 R_RAA489204_RegisterRead 32
3.4.26 R_RAA489204_RegisterWrite 33
3.5 Configuration 33
3.5.1 MCU Hardware Abstraction Layer 33

3.5.2 Battery Front End 33
3.5.3 Battery Abstraction Layer 34
3.5.4 Demo Application 35
3.6 Examples 36

4. Demo Application 40

5. Revision History 45

1. Introduction

The RAA489204 sample code provides robust and easy access to the resources and functionality of the Battery Front End (BFE) device. The code includes a specialized-to-the-device control code (Battery Abstraction Layer – BAL), a demo battery management application, and a user interface. These components are portable and suitable for integration into multitasking software projects.

The sample software package has the following features:

- Stand-alone or daisy chain operation support
- Full system scalability
- Custom configurations
- Easy access to RAA489204 resources and advanced features
- Simplified status and error monitoring
- Integrated fault diagnostics and processing
- Functional safety mechanisms
- Application Programming Interface (API) for easy integration
- Full compatibility with Renesas Advanced (RA) Family 32-bit MCUs

1.1 Assumptions and Advisory Notes

- It is assumed that you possess basic understanding of microcontrollers, embedded systems hardware, battery management systems and Li-based battery cells.
- It is assumed that you have prior experience working with Integrated Development Environments (IDEs) such as e2studio, Flexible Software Package (FSP), and terminal emulation programs such as Tera Term.
- Renesas recommends reviewing the *Industrial Battery Front End API Software Manual* to get familiar with the Battery Abstraction Layer and the interface concepts.
- Renesas recommends reviewing the *EK-RA2A1 Quick Start Guide* and *EK-RA2A1 Manual*, in addition to the *RAA489204 Datasheet* and *Evaluation Kit Manual*, to get acquainted with MCU and BFE features before proceeding further.

2. RAA489204 Battery Front End Overview

2.1 Features

RAA489204 is a 14 cell Li-ion battery manager IC that have the following features:

- High hot plug rating: 65V
- Qualified for industrial temperature range: -40°C to +85°C
- Monitors and manages up to 14 Li-Ion cells
- Monitors up to six external temperature inputs
- Cell voltage measurement accuracy: ±10mV
- 14-bit voltage and temperature measurements with user-selectable averaging function
- Two GPIO pins
- Daisy chain hardware providing robust and redundant board-to-board communications, using differential, AC-coupled signaling or transformer coupling at speed up to 1Mbps.
- High security communication protocol
- Can operate in standalone mode or with up to 30 devices in a stack, monitoring up to 420 Li-ion cells in total
- Fully tolerant to EMC and transients
- Integrated system diagnostics for all key functions

- Watchdog timer to put the device into sleep mode if communication is lost

The external MCU communicates with the stack master device using a high-speed SPI communication interface. Figure 1 shows a typical application of the RAA489204. The MCU contains and runs the sample software package.

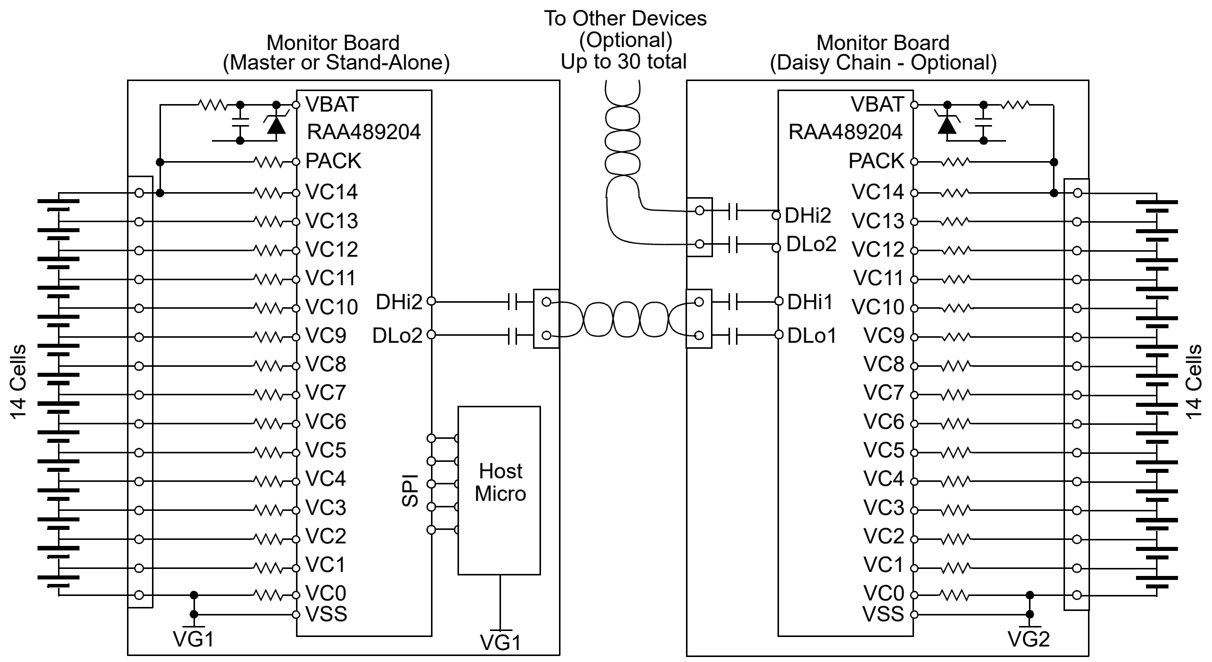

Figure 1. Typical Application of RAA489204

2.2 Applications

- Electric mobility battery packs
- Backup batteries
- Energy storage equipment
- Portable and semi-portable equipment

2.3 RAA489204 Sample Code Structure

The RAA48204 sample code contains the following software components:

- Battery Front End Application Programming Interface (BFE API)
- RAA4489204 implementation of the BFE API
- Demo application with finite-sate machine, cell balancing algorithm and command line user’s interface
- Configuration file for Renesas Flexible Software Package, which is used to generate the peripheral (Hardware Abstraction Layer - HAL) drivers for the MCU used for running the sample code

Table 1 shows the sample code directory structure. Besides the main interface, implementation and application files there are additional ones containing macros and specialized functions.

Table 1. Directory Structure of the Sample Code

Directory			Filename	Description	Module
ra	fsp	inc	api	Modules APIs	HAL (Generated by FSP)
			instances	Definition of module instances	
		src	r_*.c	APIs' implementation	
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	
			bal_data.h	Exported global variables of the interface	Applications Layer
			bal_data.c	BFE instance and definitions of the major structures	
			common_utils.h	Common macros for the Battery Abstraction Layer (BAL)	
			r_bms_cfg.h	Battery Management System (BMS) configuration macros	
			r_bms.h	Definitions, structures, enumerations and declarations of functions for the BMS	
			r_bms.c	BMS application code	
			r_usb_pcdc_descriptor.c	USB driver descriptors	
	bfe		r_bfe_api.h	Battery Abstraction Layer (BAL) API	Battery Abstraction Layer
			r_bfe_cfg.h	BAL configuration macros	
			r_bfe_common.h	Common macros for the BAL	
			r_raa489204.c	Actual code for the interface implementation	
			r_raa489204.h	Definitions, structures, enumerations and declarations of the API functions	
			r_raa489204_crc.c	CRC related functions	
			r_raa489204_crc.h	Declarations of the CRC exported global functions	

The BFE instance and its structures define the contract and features common to most of the BFEs. The BFE instance for RAA489204 contains the actual code implementation of BFE functionalities. Figure 2 shows the main software components, structures, and files of the BFE interface and instance implementation. Both configuration and control structures are extended to fit the device specifics.

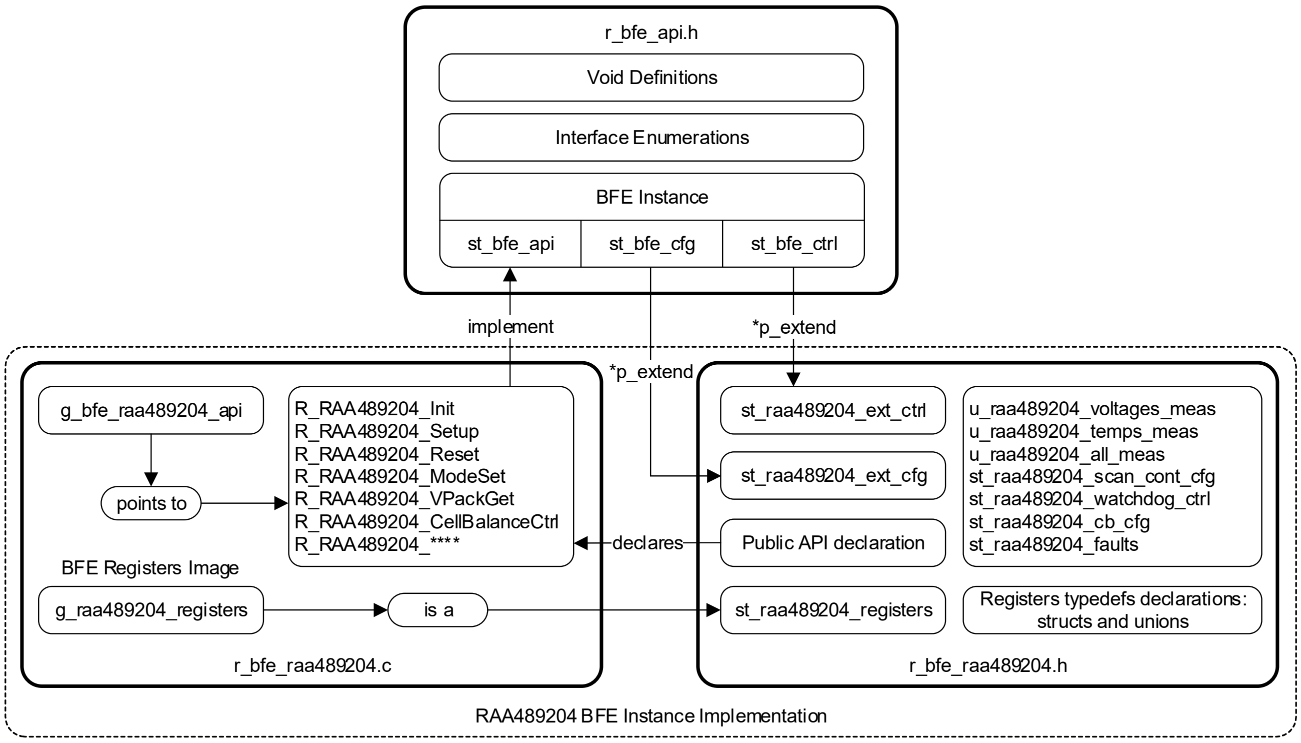

Figure 2. Main Software Components of BFE Instance Implementation

3. RAA489204 Application Programming Interface Implementation

3.1 Control and Configuration Structures

The control and configuration functions used as parameters for the API functions and holding the BFE settings, state flags, registers and more are extended to cover the device specifics. The extended structures and the relevant enumerations can be found in file **bfe/r_raa489204.h**. Table 2 shows the content of the extended control structure. It contains information about the stack identification attempts tracking during BFE initialization. It also points to the timings structure (Table 3). Timings are crucial for providing reliability and fault-safe operation in a stack with up to 30 BFEs. Operations are synchronized together with the high-speed SPI communication with the MCU and the vertical daisy-chain communication between the devices within the stack. For example, after sending a Scan command the MCU must wait for the input sampling and ADC conversion to complete before reading the measured data. Or after sending a data packet the MCU is waiting for a response for a certain period of time that is determined by many factors such as packet size, stack size, and communication speed. Most of the timings are based on the particular BFE and MCU settings and are automatically calculated in the initialization API function.

Table 2. Members of the RAA489204 Extended Control Structure

typedef struct st_raa489204_ext_ctrl		
Member	**Type**	**Description**
identify_attempts	uint16_t	Number of identify attempts
timings	st_raa489204_timings_t	Structure holding the BFE timings

Table 3. Members of the RAA489204 Timings Structure

typedef struct st_raa489204_ext_ctrl		
Member	**Type**	**Description**
daisy_chain_period_us	uint32_t	Daisy chain communication period [us]
max_bytes_delay_us	uint32_t	Maximum delay between two adjacent response Bytes [us]
min_wait_time_us	uint32_t	Maximum time for daisy chain ports to clear [us]
sleep_time_us	uint32_t	Maximum wait time for devices entering sleep mode [us]
wakeup_time_us	uint32_t	Maximum wait time for devices to exit sleep mode [us]
scan_volts_time_us	uint32_t	Scan Voltages Command processing time [us]
scan_mixed_time_us	uint32_t	Scan Mixed Command processing time with voltage averaging [us]
scan_mixed_ext_time_us	uint32_t	Scan Mixed Command processing time extension when temperature averaging is used [us]
scan_temps_time_us	uint32_t	Scan Temperatures Command processing time with temp averaging [us]
scan_wires_time_us	uint32_t	Scan Wires Command processing time [us]
scan_all_time_us	uint32_t	Scan All Command processing time [us]
scan_all_ext_time_us	uint32_t	Scan All Command processing time extension when temperature averaging is used [us]
scan_mux_time_us	uint32_t	Scan Voltage Cell MUX Command processing time [us]
reset_wait_time_us	uint32_t	Reset processing time [us]
start_up_wait_time_ms	uint32_t	Start-up delay (power on or enable) [ms]
timer_freq_hz	uint32_t	MCU Clock frequency [Hz]

Table 4 shows the content of the extended configuration structure. Its members correspond to all fixed settings of the BFE. An exception are those related to functionalities like scan continuous or cell balancing which can be reconfigured later in the code. The extended configuration structure stores constant variables directly related to the hardware such as stack size (total cell count), vertical communication speed (COMMRATE pins connection), overvoltage limit (Li-Ion chemistry), open-wire scan time (VC inputs RC filter time constant), and data ready signal MCU input pin (routing between the master BFE and the MCU). The limits are entered as real values (Voltages, Temperatures). Some of the variables have types which are defined as enumerations. Therefore, you can select from a list of options facilitating the device configuration. Keep in mind that the initialization function is checking some of the members of the extended configuration structure for correct values and if a mismatch is detected, an error code is returned. Some members of the described structures are defined as arrays. Each element of those arrays refers to a level in the stack (BFE device number). For example, the elements of cell_num array denote how many battery cells are monitored by each BFE from the stack. For more information about the available options and their effect over the BFE performance, refer to the *RAA489204 Datasheet*.

Table 4. Members of the RAA489204 Extended Configuration Structure

typedef struct st_raa489204_ext_cfg		
Member	**Type**	**Description**
stack_size	const uint8_t	Number of BFE devices in the stack
d_ch_data_speed	const e_raa489204_dch_rate_t	Daisy chain data rate
cell_num[BFE_STACK_SIZE]	const uint8_t	Number of cells per device
gpio_conf[BFE_STACK_SIZE]	const e_raa489204_gpio_conf_t	GPIO inputs configuration
temp_flt_mon[BFE_STACK_SIZE]	const uint16_t	Temperature fault monitoring selection

Table 4. Members of the RAA489204 Extended Configuration Structure (Cont.)

typedef struct st_raa489204_ext_cfg		
Member	**Type**	**Description**
gpio_flt_mon[BFE_STACK_SIZE]	const uint16_t	GPIO inputs fault monitoring selection
limit_overvolt_v	const float	Overvoltage limit
limit_undervolt_v	const float	Undervoltage limit
limit_int_temp_warning_degc	const float	Internal temperature warning
limit_ext_temp_v	const float	External temperature limit
wire_scan	const e_raa489204_i_scan_t	Wire scan current on time
mux_scan	const e_raa489204_mux_scan_t	Scan Cell MUX Command timing
flt_tot_samples	const e_raa489204_tot_smpl_t	Number of consecutive samples before a fault is registered
avging_volt	const e_raa489204_avg_t	Cell voltage averaging
avging_ext_temp	const e_raa489204_avg_t	External temperature inputs averaging
cb_test_drop_th_v	const float	The minimal voltage drop on the VCn pin when a balancing FET is on. [V]
cb_drop_time_ms	const uint32_t	The timeout [ms] needed for the voltage on the VCn pin to drop after cell balancing is enabled.
pin_data_rdy	const bsp_io_port_pin_t	Data ready pin
pin_data_rdy_mode	const bsp_io_port_pin_t	Data ready mode pin
pin_enable	const bsp_io_port_pin_t	BFE IC enable pin
pin_chip_select	const bsp_io_port_pin_t	Chips select pin

The daisy-chain data rate options for the *d_ch_data_speed* constant variable are listed in the RAA489204 Daisy-Chain Data Rate Options Enumeration (Table 5). The selected constant is used for communication timings calculation. It must match the selection by the COMMRATE pins voltage levels as they are available in the BFE registers and are compared in the code.

Table 5. RAA489204 Daisy-Chain Data Rate Options Enumeration

typedef enum e_raa489204_dch_rate		
Constant	**Value**	**Description**
BFE_D_RT_333_KHZ	0	Daisy-chain data rate is configured physically on COMMRATE0/1 pins as 333kHz.
BFE_D_RT_500_KHZ	1	Daisy-chain data rate is configured physically on COMMRATE0/1 pins as 500kHz.
BFE_D_RT_1000_KHZ	3	Daisy-chain data rate is configured physically on COMMRATE0/1 pins as 1000kHz.

Table 6 shows the options for the GPIO configuration array *gpio_conf* listed in RAA489204 GPIO Configuration Options Enumeration. The GPIO pins of RAA489204 can be configured as input, output, or special function output. The selection for both pins is assigned to the same element of the array using the logical operator OR such as:

```
.gpio_conf = {(BFE_GPIO1_INPUT | BFE_GPIO2_OUTPUT_SH_PLS),  // BFE1
              (BFE_GPIO1_ OUTPUT | BFE_GPIO2_OUTPUT)},// BFE2
```

In the example, GPIO1 of BFE #1 is used as additional external temperature input, and GPIO2 as an output indication voltage measurement. GPIO1 and GPIO2 of BFE#2 are used as general-purpose outputs.

Table 6. RAA489204 GPIO Configuration Options Enumeration

typedef enum e_raa489204_gpio_conf		
Constant	**Value**	**Description**
BFE_GPIO1_INPUT	0x0000	Configure GPIO1 pin as a general-purpose input.
BFE_GPIO1_OUTPUT_NORMAL	0x0001	Configure GPIO1 pin as a general-purpose output, controlled by G1VAL bit in Device Setup 2 Register.
BFE_GPIO1_OUTPUT_SPECIAL	0x0005	Configure GPIO1 pin as a secondary fault output and select custom fault indication using GPIO1 Fault Mask Register.
BFE_GPIO2_INPUT	0x0000	Configure GPIO2 pin as a general-purpose input.
BFE_GPIO2_OUTPUT_NORMAL	0x0002	Configure GPIO2 pin as a general-purpose output, controlled by G2VAL bit in Device Setup 2 Register.
BFE_GPIO2_OUTPUT_SH_PLS	0x001A	Configure GPIO2 pin to have a short pulse with the start of the voltage measurement.
BFE_GPIO2_OUTPUT_LNG_PLS	0xF02A	Configure GPIO2 pin to have a long synchronized pulse during the voltage measurement.

Table 7 shows the options for open-wire scan timing constant variable *wire_scan* listed in the RAA489204 Open-Wire Scan Current ON-Time Options Enumeration. The value is selected so that the RC filter time constant is not affecting the open-wire scan result.

Table 7. RAA489204 Open-Wire Scan Current ON-Time Options Enumeration

typedef enum e_raa489204_i_scan		
Constant	**Value**	**Description**
BFE_WIRE_I_SCAN_1_5MS	0x00	Configure open-wire scan current ON-time to 1.5ms.
BFE_WIRE_I_SCAN_5MS	0x01	Configure open-wire scan current ON-time to 5ms.

Table 8 shows the options for multiplexer scan timing constant variable *mux_scan* listed in the RAA489204 MUX Scan Bias Current ON-Time Options Enumeration. This setting is related to one of the self-diagnostic procedures.

Table 8. RAA489204 MUX Scan Bias Current ON-Time Options Enumeration

typedef enum e_raa489204_mux_scan		
Constant	**Value**	**Description**
BFE_CELL_MUX_SCAN_0_5MS	0	Configure MUX scan bias current ON-time to 0.5ms.
BFE_CELL_MUX_SCAN_1_25MS	1	Configure MUX scan bias current ON-time to 1.25ms.

Table 9 shows the options for *flt_tot_samples* constant variable for how many consecutive fault samples are required to set in a fault condition. They are listed in the RAA489204 Number of Samples for Registering a Fault Enumeration. This setting is valid for both voltage and temperature measurements.

Table 9. RAA489204 Number of Samples for Registering a Fault Enumeration

typedef enum e_raa489204_tot_smpl		
Constant	**Value**	**Description**
BFE_TOT_1_SMPL	0x00	Configure for 1 totalizer sample.
BFE_TOT_2_SMPL	0x01	Configure for 2 totalizer samples.
BFE_TOT_4_SMPL	0x02	Configure for 4 totalizer samples.

Table 9. RAA489204 Number of Samples for Registering a Fault Enumeration (Cont.)

typedef enum e_raa489204_tot_smpl		
Constant	**Value**	**Description**
BFE_TOT_8_SMPL	0x03	Configure for 8 totalizer samples.
BFE_TOT_16_SMPL	0x04	Configure for 16 totalizer samples.
BFE_TOT_32_SMPL	0x05	Configure for 32 totalizer samples.
BFE_TOT_64_SMPL	0x06	Configure for 64 totalizer samples.
BFE_TOT_128_SMPL	0x07	Configure for 128 totalizer samples.

Table 10 shows the options for the voltage and external temperature averaging constant variables *avging_volt* and *avging_ext_temp* listed in the RAA489204 Number of Samples Averaged Enumeration. The averaging function results in triggering additional consecutive samples and prolonging the total scan time.

Table 10. RAA489204 Number of Samples Averaged Enumeration

typedef enum e_raa489204_avg		
Constant	**Value**	**Description**
BFE_AVG_1_SMPL	0x00	Configure for no averaging.
BFE_AVG_2_SMPL	0x01	Configure for 2 samples averaging.
BFE_AVG_4_SMPL	0x02	Configure for 4 samples averaging.
BFE_AVG_8_SMPL	0x03	Configure for 8 samples averaging.
BFE_AVG_16_SMPL	0x04	Configure for 16 samples averaging.
BFE_AVG_32_SMPL	0x05	Configure for 32 samples averaging.

3.2 Registers Bank

The registers' bank holds all BFE registers in its fields. Each member is a nested structure with a predefined data type (Table 11). It contains the register address itself, the register type which has fixed values and information if the data packet can be sent to all BFEs simultaneously (broadcasted) and if an acknowledge or no response is waited by the MCU. Table 12 shows the register type options. All this information is used by the Middleware communication driver to determine how to assemble the data packet and manage the SPI communication.

Table 11. Members of the RAA489204 Register Container Structure

typedef struct st_raa489204_register		
Member	**Type**	**Description**
address	const e_raa489204_reg_addr_t	Register address
type	const e_raa489204_reg_type_t	Register type
broadcast	const bool	Register can be broadcasted
ack	const bool	BFE returns ACK after receiving the command
no_resp	const bool	BFE returns no response after receiving the command

Table 12. RAA489204 Register Type Enumeration

typedef enum e_raa489204_reg_type	
Constant	**Description**
READ_ONLY	Read only register
READ_WRITE	Read/Write register
COMMAND	Command
RESPONSE	Response
CMND_RESP	Command or Response

To set a target register to the communication driver, you need only to assign the register bank member such as:

```
st_raa489204_register_t target_register = g_raa489204_registers.cmnd_scan_voltage;
```

You can find the full register bank declaration in **bfe/r_raa489204.h** and definition in **bfe/r_raa489204.c**. The following code demonstrates a part of it:

```
/* RAA489204 registers' bank */
st_raa489204_registers_t g_raa489204_registers =
{
 .cell_setup             = {.address = BFE_REG_CELL_SETUP, .type = READ_WRITE,
                        .broadcast = false, .ack = false, .no_resp = false},

 .cell_1_voltage         = {.address = BFE_REG_CELL_1_VOLT, .type = READ_ONLY,
                        .broadcast = false, .ack = false, .no_resp = false},

 .cell_2_voltage         = {.address = BFE_REG_CELL_2_VOLT, .type = READ_ONLY,
                        .broadcast = false, .ack = false, .no_resp = false},

 .cell_3_voltage         = {.address = BFE_REG_CELL_3_VOLT, .type = READ_ONLY,
                        .broadcast = false, .ack = false, .no_resp = false},
....

 .cmnd_scan_voltage      = {.address = BFE_CMND_SCAN_VOLTS, .type = COMMAND,
                        .broadcast = true,  .ack = false, .no_resp = true},

 .cmnd_scan_temps        = {.address = BFE_CMND_SCAN_TEMPS, .type = COMMAND,
                        .broadcast = true,  .ack = false, .no_resp = true},
....

 .cmnd_nak               = {.address = BFE_CMND_NAK, .type = RESPONSE,
                        .broadcast = false, .ack = false, .no_resp = false},

 .cmnd_ack               = {.address = BFE_CMND_ACK, .type = CMND_RESP,
                        .broadcast = false, .ack = true,  .no_resp = false},

 .resp_comms_failure     = {.address = BFE_CMND_COMMS_FAILURE, .type = RESPONSE,
                        .broadcast = false, .ack = false, .no_resp = false},
....

 .cmnd_hreset            = {.address = BFE_CMND_HRESET, .type = COMMAND,
                            .broadcast = true,  .ack = false, .no_resp = true},
};
```

3.3 Private (Static) Functions

Table 13 shows the declaration and description of the private functions used in API functions in the source code. They operate on different levels in the BAL from data conversion to handling the communication with the stack and running procedures for stack identification, change of state, or certain diagnostic. The most used private function is *bfe_spi_msg_send_resp_get()*. It implements the communication protocol and actually manages the data transfer between the MCU and all BFEs in the stack. For more details, see the examples in the next sections and the comments in the source code in **bfe/r_raa489204.c**.

Table 13. Static Functions Defined in the Source Code

Function	Description
static e_bfe_err_t bfe_identify (st_bfe_ctrl_t * const p_ctrl);	Runs identification procedure of the stack and reads the serial numbers of the BFEs.
static e_bfe_err_t bfe_reset_hard (st_bfe_ctrl_t * const p_ctrl);	Sends a hard reset command to the Battery Front End to toggle the EN pin internally or externally, depending on the selection in **bfe/r_bfe_cfg.h**.
static e_bfe_err_t bfe_reset_soft (st_bfe_ctrl_t * const p_ctrl);	Sends a soft reset command to all BFEs in the stack to reset only the digital part.
static e_bfe_err_t bfe_watchdog (st_bfe_ctrl_t * const p_ctrl, e_raa489204_wd_timeout_t time);	Enables or disables the watchdog timer and selects time.
static e_bfe_err_t bfe_sleep (st_bfe_ctrl_t * const p_ctrl);	Sends a sleep command to all BFEs in the stack.
static e_bfe_err_t bfe_wake_up (st_bfe_ctrl_t * const p_ctrl);	Sends a wake-up command to all BFEs in the stack.
static e_bfe_err_t bfe_random_wake_up (st_bfe_ctrl_t * const p_ctrl);	Wakes up a sleeping BFE on random position in the stack.
static e_bfe_err_t bfe_adc_check (st_bfe_ctrl_t * const p_ctrl);	Tests the ADC of all BFEs in the stack.
static e_bfe_err_t bfe_mux_check (st_bfe_ctrl_t * const p_ctrl);	Tests cell and temperature multiplexers of all BFEs in the stack.
static e_bfe_err_t bfe_eeprom_check (st_bfe_ctrl_t * const p_ctrl);	Compares shadow registers and factory written checksum to detect EEPROM data corruption of all BFEs in the stack.
static e_bfe_err_t bfe_conf_reg_check (st_bfe_ctrl_t * const p_ctrl);	Checks configuration registers of all BFEs in the stack for data corruption.
static e_bfe_err_t bfe_cell_balancing_check (st_bfe_ctrl_t * const p_ctrl);	Tests the cell balancing circuits.
static e_bfe_err_t bfe_wires_check (st_bfe_ctrl_t * const p_ctrl);	Checks all measurement inputs for open wire condition.
static e_bfe_err_t bfe_spi_msg_send_resp_get (st_bfe_ctrl_t * const p_ctrl, st_raa489204_spi_msg_t * const p_spi_msg);	Manages SPI data transmission between the MCU and all BFEs in the stack.
static e_bfe_err_t bfe_spi_resp_get (st_bfe_ctrl_t * const p_ctrl, st_raa489204_spi_msg_t * const p_spi_msg);	Checks for available response data packet and receives it.
static e_bfe_err_t bfe_spi_pending_resp_get (st_bfe_ctrl_t * const p_ctrl, st_raa489204_spi_msg_t * const p_spi_msg);	Receives all pending response data bytes.
static e_bfe_err_t bfe_command_encode (st_bfe_ctrl_t * const p_ctrl, st_raa489204_spi_msg_t * const p_spi_msg);	Encodes the data packet from the message structure address and data fields to the encoded command buffer to send using SPI.

Table 13. Static Functions Defined in the Source Code (Cont.)

Function	Description
static e_bfe_err_t bfe_response_decode (st_bfe_ctrl_t * const p_ctrl, st_raa489204_spi_msg_t * const p_spi_msg);	Decodes the data received via SPI from the message structure encoded response buffer and copies it the response data fields.
static e_bfe_err_t bfe_response_examine (st_bfe_ctrl_t * const p_ctrl, st_raa489204_spi_msg_t * const p_spi_msg);	Examines the response and check for expected registers and message length to verify a valid response.
static float bfe_adc_to_vbat (uint16_t value);	Converts ADC value to battery voltage.
static float bfe_adc_to_vcell (uint16_t value);	Converts ADC value to cell voltage.
static float bfe_adc_to_vext (uint16_t value);	Converts ADC value to external temperature input voltage.
static float bfe_adc_to_tempc (uint16_t value);	Converts ADC value to temperature.
static float bfe_adc_to_sref (uint16_t value);	Converts ADC value to secondary reference voltage.
static uint16_t bfe_vcell_to_adc (float value);	Converts cell voltage to ADC value.
static uint16_t bfe_vext_to_adc (float value);	Converts external temperature voltage to ADC value.
static uint16_t bfe_tempc_to_adc (float value);	Converts temperature to ADC value.

3.4 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.

3.4.1 R_RAA489204_Init

e_bfe_err_t R_RAA489204_Init (st_bfe_ctrl_t * const p_bfe_ctrl, st_bfe_cfg_t const * const p_bfe_cfg)		
Description	This function initializes the BFE by enabling and configuring the necessary peripheral modules of the MCU, identifying the BFE and other device-specific actions. It modifies the Boolean variable **p_ctrl->is_initialized**.	
Operation	▪ Checks function parameters. ▪ Calculates BFE timings. ▪ Initializes a communication timeout timer peripheral module. ▪ Initializes a CRC peripheral module. ▪ Initializes a SPI interface peripheral module to connect to the master BFE device. ▪ Initializes the ISR peripheral modules ▪ Sets SPI transfer mode. ▪ Calls stack identification procedure. ▪ Resets the stack. ▪ Calls stack identification procedure. ▪ Checks the EEPROM data for corruption.	
Precondition	Please, perform a MCU peripheral communication module setup according to the requirements stated in the BFE datasheet.	
Warnings	This function does not check the communication module settings.	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure
	p_bfe_cfg	Pointer to the BFE configuration structure

Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_COMM_UNSUP_INTERFACE	The selected communication interface is unsupported.
	BFE_ERR_UNSUPPORTED_MODE	The selected configuration setting is unsupported.
	BFE_ERR_FSP	Error in the FSP layer.
	BMS_ERR_...	Inherit from bfe_identify().
	BMS_ERR_...	Inherit from bfe_reset_hard().

3.4.2 R_RAA489204_Deinit

e_bfe_err_t R_RAA489204_Deinit (st_bfe_ctrl_t * const p_bfe_ctrl);		
Description	This function deinitializes the BFE. It disables the used peripheral modules of the MCU. It modifies the Boolean variable **p_ctrl->is_initialized**.	
Operation	▪ Checks function parameters. ▪ Deinitializes the SPI peripheral module. ▪ Deinitializes the CRC peripheral module. ▪ Deinitializes the Timer peripheral modules. ▪ Deinitializes the IRQ peripheral module.	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_FSP	Error in the FSP layer.

3.4.3 R_RAA489204_Setup

e_bfe_err_t R_RAA489204_Setup (st_bfe_ctrl_t * const p_bfe_ctrl, st_bfe_cfg_t const * const p_bfe_cfg);		
Description	This function configures the BFE device (stack) by writing into all configuration registers. It extracts the necessary data from the control **p_ctrl** and configuration **p_cfg** structures.	
Operation	▪ Checks function parameters. ▪ Sends a multiple register write command to set Fault Setup Register, Overvoltage, Undervoltage and External ▪ Temperature Limit Registers, FAULT Pin Mask Register, GPIO1 Pin Mask Register, Internal Temperature Warning and Limit Registers, Cell Balance Setup Register, Watchdog/Balance Time Register, Device Setup 1 and 2 Registers, Cell Setup Register. ▪ Verifies if values are correctly written. ▪ Check Communication Setup ▪ Register to verify daisy chain data speed. ▪ Send a command to recalculate Page 2 ▪ Registers checksum.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_bfe_cfg	Pointer to the BFE configuration structure.

Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_WRITE_VERIFY	Register write verification error.
	BMS_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

3.4.4 R_RAA489204_Reset

e_bfe_err_t R_RAA489204_Reset (st_bfe_ctrl_t * const p_bfe_ctrl, e_bfe_reset_type_t type);		
Description	This function resets the BFE. Several predefined reset options can be set with the **type** input parameter.	
Operation	▪ Checks function parameters. ▪ Resets the digital part of the device (soft reset) or both digital and analog parts (hard) according to the selected reset type. ▪ Runs stack identification procedure. ▪ Reads serial numbers of all BFE devices.	
Precondition	The BFE interface should have already been initialized.	
Warnings	▪ Always check the **p_ctrl->is_fault_detected** flag after calling this function! ▪ You should reconfigure the BFE after reset by calling R_RAA489204_Setup()!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	type	Hard or soft reset type selector.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	Invalid reset type was selected.
	BMS_ERR_...	Inherit from bfe_reset_soft().
	BMS_ERR_...	Inherit from bfe_reset_hard().
	BMS_ERR_...	Inherit from bfe_identify().

The values for the function parameter *type* are defined as constants in the BFE Reset Types Enumeration in file **bfe/r_bfe_api.h**. Table 14 lists the supported reset options by the BFE.

Table 14. BFE Reset Types Enumeration

typedef enum e_bfe_reset_type	
Constant	**Description**
BFE_RESET_TYPE_SOFT	Reset only the digital part.
BFE_RESET_TYPE_HARD	Reset both the digital and analog parts.

3.4.5 R_RAA489204_ModeSet

e_bfe_err_t R_RAA489204_ModeSet (st_bfe_ctrl_t * const p_bfe_ctrl, e_bfe_mode_t mode);	
Description	This function forces the BFE to enter sleep mode or wakes it up. Mode or state is selected with the **mode** input parameter from a predefined list of modes. It modifies the Boolean variable **p_ctrl->is_low_power**. When more than two unsuccessful wake-up attempts are made, a procedure is followed to wake up a random sleeping device from the stack.

Operation	▪ Checks function parameters. ▪ Calls a local function for entering sleep mode or waking up the BFE depending on the selected mode.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	mode	Mode selection.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	Invalid mode was selected.
	BMS_ERR_...	Inherit from bfe_wake_up().
	BMS_ERR_...	Inherit from bfe_random_wake_up().
	BMS_ERR_...	Inherit from bfe_sleep().

The values for the function parameter *mode* are defined as constants in the BFE States and Modes Enumeration in file **bfe/r_bfe_api.h**. Table 15 shows the supported modes by the sample code.

Table 15. BFE States and Modes Enumeration

typedef enum e_bfe_mode	
Constant	**Description**
BFE_MODE_IDLE	The device is ready waiting for a task to be executed.
BFE_MODE_LOW_POWER_MODE	The BFE is currently in low power mode.

3.4.6 R_RAA489204_ModeRead

e_bfe_err_t R_RAA489204_ModeRead (st_bfe_ctrl_t * const p_bfe_ctrl, e_bfe_mode_t * const p_mode);		
Description	This function reads the current BFE mode. The pointer **p_mode** points to a variable where the result can be found.	
Operation	▪ Checks function parameters. ▪ Compare control structure parameters to obtain the current mode.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_mode	Pointer to the obtained mode.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.

Table 15 shows the expected modes returned as a result.

3.4.7 R_RAA489204_CommTest

e_bfe_err_t R_RAA489204_CommTest (st_bfe_ctrl_t * const p_bfe_ctrl);	
Description	This function tests communication between the MCU and single or multiple BFE devices. If communication cannot be established, an error is returned accordingly.

Operation	▪ Checks function parameters. ▪ Sends ACK command and waits for response form the top stack device.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

3.4.8 R_RAA489204_SelfDiag

e_bfe_err_t R_RAA489204_SelfDiag (st_bfe_ctrl_t * const p_bfe_ctrl, e_bfe_diag_option_t option);		
Description	This function runs a self-diagnostic test for the BFE. Several predefined diagnostic options can be set with the **option** input parameter.	
Operation	▪ Checks function parameters. ▪ Calls a local function to run the selected diagnostic.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	option	Selected option for the self-diagnostic.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	Invalid option was selected.
	BFE_ERR_...	Inherit from bfe_adc_check().
	BFE_ERR_...	Inherit from bfe_mux_check().
	BFE_ERR_...	Inherit from bfe_eeprom_check().
	BFE_ERR_...	Inherit from bfe_conf_reg_check().
	BFE_ERR_...	Inherit from bfe_cell_balancing_check().
	BFE_ERR_...	Inherit from bfe_wires_check().

The values for the function parameter *option* are defined as constants in the BFE Diagnostic Options Enumeration in file **bfe/r_bfe_api.h**. Table 16 shows the supported diagnostic options by the sample code.

Table 16. BFE Diagnostic Options Enumeration

typedef enum e_bfe_diag_option	
Constant	**Description**
BFE_FULL_TEST	Run a complete self-test.
BFE_TEST_ADC	Test the ADC.
BFE_TEST_MUX	Test the multiplexer.

Table 16. BFE Diagnostic Options Enumeration

typedef enum e_bfe_diag_option	
Constant	**Description**
BFE_TEST_CB	Test cell balancing circuit.
BFE_TEST_OW	Check for open wires.

3.4.9 R_RAA489204_MemCheck

e_bfe_err_t R_RAA489204_MemCheck (st_bfe_ctrl_t * const p_bfe_ctrl, e_bfe_mem_check_option_t option);		
Description	This function runs memory tests inside a BFE for corrupted registers and data. Several predefined memory test options can be set with the **option** input parameter.	
Operation	▪ Checks function parameters. ▪ Calls a local function to run the selected memory test.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	option	Selected option for the memory test.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	Invalid option was selected.
	BFE_ERR_...	Inherit from bfe_conf_reg_check().
	BFE_ERR_...	Inherit from bfe_eeprom_check().

The values for the function parameter *option* are defined as constants in the BFE Memory Check Options Enumeration in file **bfe/r_bfe_api.h**. Table 17 shows the supported memory check options by the sample code.

Table 17. BFE Memory Check Options Enumeration

typedef enum e_bfe_mem_check_option	
Constant	**Description**
BFE_CHECK_EEPROM	Verify content of EEPROM memory.
BFE_CHECK_DEF_VALS	Check registers for default values.

3.4.10 R_RAA489204_VPackGet

e_bfe_err_t R_RAA489204_VPackGet (st_bfe_ctrl_t * const p_bfe_ctrl, float * const p_value, bool trigger);	
Description	This function acquires the battery pack voltage with the BFE. The returned data is converted into voltage. The pointer **p_value** points to a variable where the measured voltage can be found. The Boolean input parameter **trigger** indicates whether a measurement is executed before reading the value.
Operation	▪ Checks function parameters. ▪ Sends a measure pack voltage command to the stand-alone device or all devices in the stack. ▪ Confirms the reception of the measure command (optional). ▪ Sends commands to read battery pack voltage.
Precondition	The BFE interface should have already been initialized.

Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_value	Pointer to the acquired data.
	trigger	Triggered a measurement or only read data for the last one.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_SCAN_CNTR	Scan command was not received.
	BFE_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

3.4.11 R_RAA489204_IPackGet

e_bfe_err_t R_RAA489204_IPackGet (st_bfe_ctrl_t * const p_bfe_ctrl, st_bfe_i_pack_meas_t * const p_values, bool trigger);		
Description	This function is unsupported!	
Operation	-	
Precondition	-	
Warnings	This function is unsupported!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure
	p_values	Pointer to the acquired data structure.
	trigger	Triggered a measurement or only read data for the last one.
Return values	BFE_ERR_UNSUPPORTED_FEATURE	This function is not supported by the current API implementation.

3.4.12 R_RAA489204_VoltagesGet

e_bfe_err_t R_RAA489204_VoltagesGet (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_vcell_meas_t * const p_values, bool trigger);		
Description	This function acquires the voltages of all cells and the battery pack voltage. The returned data is converted into voltage. The pointer **p_values** points to an array of unions where the measured voltages for the whole stack can be found. The Boolean input parameter **trigger** indicates whether a measurement is executed before reading the values.	
Operation	▪ Checks function parameters. ▪ Sends a scan voltage commands to the stand-alone device or all devices in the stack. ▪ Confirms the reception of the scan command (optional). ▪ Sends commands to block-read the cell and battery pack voltages.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_values	Pointer to the acquired data.
	trigger	Triggered a measurement or only read data for the last one.

Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_SCAN_CNTR	Scan command was not received.
	BFE_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

The void pointer *p_values* points to the array of unions where this API function returns the measured values. The unions' type is redefined in the file **bfe/r_ raa489204.h** and has the following content:

```
/** RAA489204 Measured Voltages Data Union */
typedef union u_raa489204_voltages_meas
{
    float vector[15];

    struct
    {
        float v_cell_1;///< Cell 1 voltage [V]
        float v_cell_2;///< Cell 2 voltage [V]
        float v_cell_3;///< Cell 3 voltage [V]
        float v_cell_4;///< Cell 4 voltage [V]
        float v_cell_5;///< Cell 5 voltage [V]
        float v_cell_6;///< Cell 6 voltage [V]
        float v_cell_7;///< Cell 7 voltage [V]
        float v_cell_8;///< Cell 8 voltage [V]
        float v_cell_9;///< Cell 9 voltage [V]
        float v_cell_10;///< Cell 10 voltage [V]
        float v_cell_11;///< Cell 11 voltage [V]
        float v_cell_12;///< Cell 12 voltage [V]
        float v_cell_13;///< Cell 13 voltage [V]
        float v_cell_14;///< Cell 14 voltage [V]
        float v_pack;///< Pack voltage [V]
    } measurements;
} u_raa489204_voltages_meas_t;
```

3.4.13 R_RAA489204_Temps

e_bfe_err_t R_RAA489204_Temps (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_temp_meas_t * const p_values, bool trigger);	
Description	This function acquires the internal temperature, ExTn and GPIO input voltages. The returned data is converted into temperature and voltage. The pointer **p_values** points to an array of unions where the measured values for the whole stack can be found. The Boolean input parameter **trigger** indicates whether a measurement is executed before reading the values.
Operation	▪ Checks function parameters. ▪ Sends a scan temperatures commands to the stand-alone device or all devices in the stack. ▪ Confirms the reception of the scan command (optional). ▪ Sends commands to block-read the internal temperature and all ExT input voltages.
Precondition	The BFE interface should have already been initialized.
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!

Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_values	Pointer to the acquired data.
	trigger	Triggered a measurement or only read data for the last one.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_SCAN_CNTR	Scan command was not received.
	BFE_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

The void pointer *p_values* points to the array of unions where this API function returns the measured values. The unions' type is redefined in the file **bfe/r_ raa489204.h** and has the following content:

```
/** RAA489204 Measured Temperatures Data Union */
typedef union u_raa489204_temps_meas
{
    float vector[7];

    struct
    {
        float temp_int_c;///< Internal temperature [deg. C]
        float v_ext_1;       ///< External temperature input 1 voltage [V]
        float v_ext_2;       ///< External temperature input 2 voltage [V]
        float v_ext_3;            ///< External temperature input 3 voltage [V]
        float v_ext_4;             ///< External temperature input 4 voltage [V]
        float v_gpio_1;            ///< General purpose input 1 voltage [V]
        float v_gpio_2;           ///< General purpose input 2 voltage [V]
    } measurements;
} u_raa489204_temps_meas_t;
```

3.4.14 R_RAA489204_AllGet

e_bfe_err_t R_RAA489204_AllGet (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_all_meas_t * const p_values, bool trigger);		
Description	This function acquires the cell voltages, the battery pack voltage, the secondary reference voltage, internal temperature, ExTn and GPIO input voltages. The returned data is converted into relevant units. The pointer **p_values** points to an array of unions where the measured values for the whole stack can be found. The Boolean input parameter **trigger** indicates whether a measurement is executed before reading the values.	
Operation	▪ Checks function parameters. ▪ Sends a scan all commands to the stand-alone device or all devices in the stack. ▪ Confirms the reception of the scan command (optional). ▪ Sends commands to block-read all page 1 measurement data registers.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_values	Pointer to the acquired data.
	trigger	Triggered a measurement or only read data for the last one.

Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_SCAN_CNTR	Scan command was not received.
	BFE_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

The void pointer *p_values* points to the array of unions where this API function returns the measured values. The unions' type is redefined in the file **bfe/r_ raa489204.h** and has the following content:

```
/** RAA489204 Measure All Data Union */
typedef union u_raa489204_all_meas
{
    float vector[23];

    struct
    {
        float v_cell_1;///< Cell 1 voltage [V]
        float v_cell_2;      ///< Cell 2 voltage [V]
        float v_cell_3;      ///< Cell 3 voltage [V]
        float v_cell_4;      ///< Cell 4 voltage [V]
        float v_cell_5;     ///< Cell 5 voltage [V]
        float v_cell_6;      ///< Cell 6 voltage [V]
        float v_cell_7;     ///< Cell 7 voltage [V]
        float v_cell_8;     ///< Cell 8 voltage [V]
        float v_cell_9;      ///< Cell 9 voltage [V]
        float v_cell_10;    ///< Cell 10 voltage [V]
        float v_cell_11;   ///< Cell 11 voltage [V]
        float v_cell_12;    ///< Cell 12 voltage [V]
        float v_cell_13;    ///< Cell 13 voltage [V]
        float v_cell_14;    ///< Cell 14 voltage [V]
        float v_pack;       ///< Pack voltage [V]
        float temp_int_c;    ///< Internal temperature [deg. C]
        float v_ext_1;      ///< External temperature input 1 voltage [V]
        float v_ext_2;      ///< External temperature input 2 voltage [V]
        float v_ext_3;      ///< External temperature input 3 voltage [V]
        float v_ext_4;      ///< External temperature input 4 voltage [V]
        float v_gpio_1;     ///< General purpose input 1 voltage [V]
        float v_gpio_2;     ///< General purpose input 2 voltage [V]
        float v_ref_sec;     ///< Secondary reference voltage [V]
    } measurements;
} u_raa489204_all_meas_t;
```

3.4.15 R_RAA489204_VMixGet

e_bfe_err_t R_RAA489204_VMixGet (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_other_meas_t * const p_values, bool trigger);	
Description	This function acquires all voltages, internal temperature and ExT1 input voltage. The returned data is converted into voltages and temperature. The pointer **p_values** points to an array of unions where the measured values for the whole stack can be found. The Boolean input parameter **trigger** indicates whether a measurement is executed before reading the values.

Operation	▪ Checks function parameters. ▪ Sends a scan mixed commands to the stand-alone device or all devices in the stack. ▪ Confirms the reception of the scan command (optional). ▪ Sends commands to block-read all cell, pack, ExT1 input voltages and the internal temperature.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_values	Pointer to the acquired data.
	trigger	Triggered a measurement or only read data for the last one.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_SCAN_CNTR	Scan command was not received.
	BFE_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

The void pointer *p_values* points to the array of unions where this API function returns the measured values. The unions' type is redefined in the file **bfe/r_ raa489204.h** and has the following content:

```
/** RAA489204 Measure All Data Union */
typedef union u_raa489204_mixed_voltages_meas
{
    float vector[16];

    struct
    {
        float v_cell_1;///< Cell 1 voltage [V]
        float v_cell_2;     ///< Cell 2 voltage [V]
        float v_cell_3;     ///< Cell 3 voltage [V]
        float v_cell_4;     ///< Cell 4 voltage [V]
        float v_cell_5;     ///< Cell 5 voltage [V]
        float v_cell_6;     ///< Cell 6 voltage [V]
        float v_cell_7;    ///< Cell 7 voltage [V]
        float v_cell_8;      ///< Cell 8 voltage [V]
        float v_cell_9;      ///< Cell 9 voltage [V]
        float v_cell_10;     ///< Cell 10 voltage [V]
        float v_cell_11;    ///< Cell 11 voltage [V]
        float v_cell_12;    ///< Cell 12 voltage [V]
        float v_cell_13;    ///< Cell 13 voltage [V]
        float v_cell_14;    ///< Cell 14 voltage [V]
        float v_pack;       ///< Pack voltage [V]
        float v_ext_1;      ///< External temperature input 1 voltage [V]
    } measurements;
} u_raa489204_mixed_voltages_meas_t;
```

3.4.16 R_RAA489204_FaultsAllRead

e_bfe_err_t R_RAA489204_FaultsAllRead (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_faults_t * const p_faults);		
Description	This function reads all fault registers of the BFE. The pointer **p_faults** points to an array of structures where the fault data for the whole stack can be found.	
Operation	▪ Checks function parameters. ▪ Read all fault related (non-setup) registers.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_faults	Pointer to faults data structure.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

The void pointer *p_ faults* points to the array of structures where this API function returns the fault data. The structures' type is redefined in the file **bfe/r_raa489220.h**. A non-zero member of any fault structure (true for Boolean types) indicates that an error is detected. It has the following content:

```
/** RAA489204 fault data structure */
typedef struct st_raa489204_faults
{
    bool     flt_under_run_buffer;///< Memory buffer under run
    bool     flt_over_run_buffer;    ///< Memory buffer over run
    bool     flt_oscillator;            ///< Oscillator fault
    bool     flt_wdt_timout;            ///< Watchdog timeout fault
    bool     flt_ow_vbat;               ///< Open wire fault on VBAT connection
    bool     flt_ow_vss;               ///< Open wire fault on VSS connection
    bool     flt_reg_parity;             ///< Register checksum parity error
    bool     flt_ee_parity;             ///< EEPROM parity error
    bool     flt_v_ref;                  ///< Voltage reference fault
    bool     flt_v_reg;                 ///< Voltage regulator fault
    bool     flt_temp_mux;              ///< Temperature multiplexer error
    bool     flt_cell_mux;              ///< Cells multiplexer error
    bool     flt_ic_temp_warning;        ///< IC temperature warning
    bool     flt_ic_temp_fault;         ///< IC temperature fault
    uint16_t flt_over_temp;              ///< Over-temperature fault
    uint16_t flt_overvolt;               ///< Overvoltage fault
    uint16_t flt_undervolt;             ///< Undervoltage fault
    uint16_t flt_open_wire;             ///< Open wire fault
    uint16_t flt_open_temp_input;        ///< Open temperature input
} st_raa489204_faults_t;
```

3.4.17 R_RAA489204_FaultsCheck

e_bfe_err_t R_RAA489204_FaultsCheck (st_bfe_ctrl_t * const p_bfe_ctrl);		
Description	This function checks the BFE for faults. It monitors the relevant fault pin or checks a fault status register. The result is returned into the Boolean variable **p_ctrl->is_fault_detected**.	
Operation	▪ Checks function parameters. ▪ Checks the fault pin for assertion. ▪ Check fault status register of all devices in stack and returns fault if non zero.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

3.4.18 R_RAA489204_FaultsAllClear

e_bfe_err_t R_RAA489204_FaultsAllClear (st_bfe_ctrl_t * const p_bfe_ctrl, bool * const p_success);		
Description	This function attempts to clear all faults in the BFE. The pointer p_success points to a variable where the result of clearing faults can be found.	
Operation	▪ Checks function parameters. ▪ Sends a command to clear fault status, over-temperature, overvoltage, undervoltage, open wire registers and general fault status register. ▪ Resets the fault filter. ▪ Cleans the faults data structure. ▪ Reset totalizer counters.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_success	Pointer to boolean faults clear status variable.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_WRITE_VERIFY	Register write verification error.
	BMS_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

3.4.19 R_RAA489204_CellBalanceCtrl

e_bfe_err_t R_RAA489204_CellBalanceCtrl (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_cb_cfg_t * const p_bal_cfg, e_bfe_process_ctrl_t ctrl_option);	
Description	This function configures and controls cell balancing process in the BFE. The void pointer **p_bal_cfg** points to the cell balancing configuration parameters. The process is controlled by the input parameter **ctrl_option**.

Operation	▪ Checks function parameters. ▪ Inhibits cell balancing. ▪ Reconfigure configure Cell Balance Status, Value, Time and Setup registers according to the selected balance mode. ▪ Recalculates Page 2 registers checksum.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_bal_cfg	Pointer to the balancing configuration structure.
	ctrl_option	Specify action to enable or inhibit cell balancing.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	Invalid balancing option was selected.
	BFE_ERR_WRITE_VERIFY	Register write verification error.
	BMS_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

The cell balancing is controlled with the function parameter *ctrl_option*. Its values are fixed and defined as constants in the BFE Process Control Enumeration in file **bfe/r_bfe_api.h**. Table 18 shows the supported control options.

Table 18. BFE Process Control Enumeration

typedef enum e_bfe_process_ctrl	
Constant	**Description**
BFE_PROCESS_ENABLE	Start the process.
BFE_PROCESS_INHIBIT	Stop the process.

The pointer *p_bal_cfg* points to the structure with cell balancing configuration redefined in file **bfe/r_raa489204.h**. Table 19 shows the content of that structure. Keep in mind that the members that indicate the cell balancing mode and use of external FETs are constants and must be set during definition of the structure. The other members can be modified every time before calling the function. Depending on the selected mode, not all configuration parameters are used. Table 20 shows the supported balancing modes. Table 21 and Table 22 show the timing enumerations used in Timed and Auto Mode. For more information refer to the *RAA489204 Datasheet*.

Table 19. Members of the RAA489204 Cell Balancing Configuration Structure

typedef struct st_raa489204_cb_cfg_t		
Member	**Type**	**Description**
mode	const e_raa489204_cb_mode_t	BFE cell balancing mode
external_cb	const bool	Enable external cell balancing.
timeout	e_raa489204_cb_timeout_t	Cell balancing timeout to keep balancing FETs on in Timed or Auto Balance Mode.
wait_time	e_raa489204_cb_wait_time_t	Cell balancing wait time before measurement in Auto Balance Mode.
cell_sel[BFE_STACK_SIZE]	uint16_t	Select cells to be balanced.

Table 19. Members of the RAA489204 Cell Balancing Configuration Structure (Cont.)

typedef struct st_raa489204_cb_cfg_t		
Member	**Type**	**Description**
pattern[BFE_CB_STAT_REG]	uint16_t	Cell balancing pattern (Bitmask)
charge[BFE_STACK_SIZE][BFE_CB_VAL_REG]	uint16_t	The charge to be removed from each cell in Auto Balance Mode.

Table 20. RAA489204 Cell Balancing Modes Enumeration

typedef enum e_raa489204_cb_mode		
Constant	**Value**	**Description**
BFE_BALANCE_MODE_OFF	0x00	Cell balancing is OFF
BFE_BALANCE_MODE_MANUAL	0x01	Manual cell balancing mode
BFE_BALANCE_MODE_TIMED	0x02	Timed cell balancing mode
BFE_BALANCE_MODE_AUTO	0x03	Auto cell balancing mode

Table 21. RAA489204 Cell Balancing Timeout Options Enumeration

typedef enum e_raa489204_cb_timeout		
Constant	**Value**	**Description**
BFE_BAL_A_T_DSBL	0x00	Timed or Auto cell balancing is disabled.
BFE_BAL_00M_20S	0x01	Keep cell balancing FETs ON for 20 s.
BFE_BAL_00M_40S	0x02	Keep cell balancing FETs ON for 40 s.
...	...	...
BFE_BAL_42M_00S	0xFE	Keep cell balancing FETs ON for 42 min 00 s.
BFE_BAL_42M_20S	0xFF	Keep cell balancing FETs ON for 42 min 20 s.

Table 22. RAA489204 Cell Balancing Wait Time Options Enumeration

typedef enum e_raa489204_cb_wait_time		
Constant	**Value**	**Description**
BFE_BAL_WAIT_0S	0x00	Cell balancing wait time is 0s.
BFE_BAL_WAIT_1S	0x01	Cell balancing wait time is 1s.
BFE_BAL_WAIT_2S	0x02	Cell balancing wait time is 2s.
BFE_BAL_WAIT_4S	0x03	Cell balancing wait time is 4s.
BFE_BAL_WAIT_8S	0x04	Cell balancing wait time is 8s.
BFE_BAL_WAIT_16S	0x05	Cell balancing wait time is 16s.
BFE_BAL_WAIT_32S	0x06	Cell balancing wait time is 32s.
BFE_BAL_WAIT_64S	0x07	Cell balancing wait time is 64s.

3.4.20 R_RAA489204_IsCellBalancing

e_bfe_err_t R_RAA489204_IsCellBalancing (st_bfe_ctrl_t * const p_bfe_ctrl);		
Description	This function checks if cell balancing is in progress in the BFE. The data is returned into the Boolean variable **p_ctrl->is_balancing**.	
Operation	▪ Checks function parameters. ▪ Reads Balance Setup Register of each device and checks EOB bit.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BMS_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

3.4.21 R_RAA489204_ContScanCtrl

e_bfe_err_t R_RAA489204_ContScanCtrl (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_scan_cont_cfg_t * const p_scan_cfg, e_bfe_process_ctrl_t ctrl_option);		
Description	This function controls scan continuous function of the BFE. The void pointer **p_scan_cfg** points to the scan continuous configuration parameters. The process is controlled by the input parameter **ctrl_option**. The function modifies the Boolean variable **p_ctrl->is_scan_continuous**.	
Operation	▪ Checks function parameters. ▪ Updates Scan Continuous interval. ▪ Sends scan continuous enable or inhibit command. ▪ Recalculates register checksum.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_scan_cfg	Pointer to the continuous scan configuration structure.
	ctrl_option	Specify action to enable or inhibit continuous scan.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	The selected control option is invalid.
	BFE_ERR_WRITE_VERIFY	Register write verification error.
	BMS_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

The continuous scanning is controlled with the function parameter *ctrl_option*. Its values are fixed and defined as constants in the BFE Process Control Enumeration in file **bfe/r_bfe_api.h**. Table 18 shows the supported control options. The pointer *p_scan_cfg* points to the structure with continuous scan configuration redefined in file **bfe/r_raa489204.h**. Table 23 shows its content. It has a single member used for setting the interval between scans. Table 24 shows the options for the interval listed in RAA489204 Scan Continuous Interval Options Enumeration.

Table 23. Members of the RAA489204 Scan Continuous Configuration Structure

typedef struct st_raa489204_scan_cont_cfg		
Member	**Type**	**Description**
interval	e_raa489204_scan_cont_int_t	Scan continuous interval time

Table 24. RAA489204 Scan Continuous Interval Options Enumeration

typedef enum e_raa489204_cb_wait_time		
Constant	**Value**	**Description**
BFE_SCN_CNT_16MS	0x00	The scan continuous interval is 16ms.
BFE_SCN_CNT_32MS	0x01	The scan continuous interval is 32ms.
BFE_SCN_CNT_64MS	0x02	The scan continuous interval is 64ms.
BFE_SCN_CNT_128MS	0x03	The scan continuous interval is 128ms.
BFE_SCN_CNT_256MS	0x04	The scan continuous interval is 256ms.
BFE_SCN_CNT_512MS	0x05	The scan continuous interval is 512ms.
BFE_SCN_CNT_1024MS	0x06	The scan continuous interval is 1024ms.
BFE_SCN_CNT_2048MS	0x07	The scan continuous interval is 2048ms.
BFE_SCN_CNT_4096MS	0x08	The scan continuous interval is 4096ms.
BFE_SCN_CNT_8192MS	0x09	The scan continuous interval is 8192ms.
BFE_SCN_CNT_16384MS	0x0A	The scan continuous interval is 16384ms.
BFE_SCN_CNT_32768MS	0x0B	The scan continuous interval is 32768ms.
BFE_SCN_CNT_65536MS	0x0C	The scan continuous interval is 65536ms.

3.4.22 R_RAA489204_WatchdogCtrl

e_bfe_err_t R_RAA489204_WatchdogCtrl (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_watchdog_ctrl_t * const p_control_options);		
Description	This function controls watchdog timer function in the BFE. The void pointer **p_ options** points to the watchdog timer parameters.	
Operation	▪ Checks function parameters. ▪ Calls a local function to change watchdog timeout.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	p_control_options	Pointer to control options structure.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	The selected control option is invalid.
	BFE_ERR_...	Inherit from bfe_watchdog().

The pointer *p_control_options* points to the structure with watchdog timer control parameters redefined in file **bfe/r_raa489204.h**. Table 25 shows its content. The structure has two members. The first one determines the time after which the BFEs goes to sleep mode. Table 26 shows the timeout options listed in RAA489204 Watchdog Timer Interval Options Enumeration. The second member is used for turning on or off the timer. Its has fixed options shown in Table 18.

Table 25. Members of the RAA489204 Watchdog Timer Control Structure

typedef struct st_raa489204_watchdog_ctrl		
Member	**Type**	**Description**
time	e_raa489204_wd_timeout_t	Watchdog time interval.
ctrl_option	e_bfe_process_ctrl_t	Start/stop watchdog timer.

Table 26. RAA489204 Watchdog Timer Interval Options Enumeration

typedef enum e_raa489204_wd_timeout		
Constant	**Value**	**Description**
BFE_WDT_OFF	0x00	Watchdog Timer is disabled.
BFE_WDT_1S	0x01	Watchdog timeout is 1s.
BFE_WDT_2S	0x02	Watchdog timeout is 2s.
...	...	...
BFE_WDT_63S	0x3F	Watchdog timeout is 63s.
BFE_WDT_2MIN	0xFF	Watchdog timeout is 2min.
BFE_WDT_4MIN		Watchdog timeout is 4min.
...	...	...
BFE_WDT_1520MIN	0xFF	Watchdog timeout is 1520min.

3.4.23 R_RAA489204_FETsCtrl

e_bfe_err_t R_RAA489204_FETsCtrl (st_bfe_ctrl_t * const p_bfe_ctrl, uint8_t group_num, e_bfe_fet_state_t c_fet_state, e_bfe_fet_state_t d_fet_state);		
Description	This function is unsupported!	
Operation	-	
Precondition	-	
Warnings	This function is unsupported!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	group_num	Select FET group to control.
	c_fet_state	Specify state of the charge FET control pin.
	d_fet_state	Specify state of the discharge FET control pin.
Return values	BFE_ERR_UNSUPPORTED_FEATURE	This function is not supported by the current API implementation.

3.4.24 R_RAA489204_GPIOsCtrl

e_bfe_err_t R_RAA489204_GPIOsCtrl (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_gpio_ctrl_t * const p_control_options);		
Description	This function controls the GPIO pins of the BFE. The variable **p_options** points to structure that contains the pin parameters.	
Operation	▪ Checks function parameters. ▪ Modifies G1VAL and G2VAL bits in all Device Setup 2 Registers. ▪ Writes to all Device Setup 2 Registers in the stack. ▪ Verifies the register write.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure.
	ctrl_option	Pointer to GPIO pins control structure.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_WRITE_VERIFY	Register write verification error.
	BMS_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

When the GPIOs of the BFEs are configured as regular digital outputs, their output levels are controlled by the generic type function parameter *p_control_options*. It points to a control structure which is redefined in file **bfe/r_raa489204.h**. Table 27 shows the content of that structure and Table 28 shows the fixed options listed in BFE GPIOs Output State Options Enumeration. *Note:* The members of the GPIO control structure are arrays which elements correspond to a position of a BFE in the stack.

Table 27. Members of the RAA489204 GPIO Control Structure

typedef struct st_raa489204_gpio_ctrl		
Member	**Type**	**Description**
gpio1_level[BFE_STACK_SIZE]	e_raa489204_gpio_states_t	Set as output level of GPIO1 when it is configured as general-purpose output.
gpio2_level[BFE_STACK_SIZE]	e_raa489204_gpio_states_t	Set as output level of GPIO2 when it is configured as general-purpose output.

Table 28. BFE GPIOs Output State Options Enumeration

typedef enum e_raa489204_gpio_states		
Constant	**Value**	**Description**
BFE_GPIO_LOW	0x00	Set general-purpose output to LOW.
BFE_GPIO_HIGH	0x01	Set general-purpose output to HIGH.

3.4.25 R_RAA489204_RegisterRead

e_bfe_err_t R_RAA489204_RegisterRead (st_bfe_ctrl_t * const p_ctrl, bfe_register_t * const p_register);		
Description	This function reads a register in the BFE. The pointer **p_register** points to a structure that contains the register address, value and other device specific parameters.	
Operation	▪ Checks function parameters. ▪ Sends a read command.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure
	p_register	Pointer to register address and data container.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BMS_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

The void type API function parameter *p_register* provides path to the target register and data. It is redefined as a RAA489204 Quick Register Access Structure in file **bfe/r_raa489204.h**. Table 29 shows the content the structure. Table 30 shows the options about the *device_address* member listed in RAA489204 Device Addresses Enumeration. The register address is assigned with a member of the register bank as described in the Registers Bank section. The read data are available after calling the function in the third member of the structure data.

Table 29. Members of the RAA489204 Quick Register Access Structure

typedef struct st_raa489204_quick_reg		
Member	**Type**	**Description**
device_address	e_raa489204_dev_addr_t	Device number in the stack.
p_reg	st_raa489204_register_t	Pointer to register address data container.
data	uint16_t	Command or response data field.

Table 30. RAA489204 Device Addresses Enumeration

typedef enum e_raa489204_dev_addr		
Constant	**Value**	**Description**
BFE_DAISY_CHAIN_IDENTIFY	0x00	Identify devices in stack address.
BFE_DAISY_CHAIN_DEVICE1	0x01	The master device address.
BFE_DAISY_CHAIN_DEVICE2	0x02	Device # 2 address.
...	...	...
BFE_DAISY_CHAIN_DEVICE30	0x1E	Device # 30 address.
BFE_DAISY_CHAIN_ADDRESS_ALL	0x1F	Address all devices (broadcast).

3.4.26 R_RAA489204_RegisterWrite

e_bfe_err_t R_RAA489204_RegisterWrite (st_bfe_ctrl_t * const p_ctrl, bfe_register_t * const p_register);		
Description	This function writes in a register of the BFE. The pointer **p_register** points to a structure that contains the register address, value and other device specific parameters.	
Operation	▪ Checks function parameters. ▪ Sends a write command.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure
	p_register	Pointer to register address and data container.
Return values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BMS_ERR_...	Inherit from bfe_spi_msg_send_resp_get().

3.5 Configuration

3.5.1 MCU Hardware Abstraction Layer

The Hardware Abstraction Layer drivers used for the peripherals of the selected MCU from Renesas RA Family are generated in e2studio using Flexible Software Package (FSP). They can be modified in file **configuration.xml**. It is not necessary to change anything in the FSP configuration as long as you are using the same MCU and evaluation board as described in the *RAA489204 Sample Code Quick Start Guide*.

3.5.2 Battery Front End

The BFE settings are entered in the configuration structures, defined in **bal_data.c**. The members of *g_bfe0_cfg* and its extension *g_bfe0_ext_cfg* are constant variables that are initialized with the required settings during definition and cannot be further modified in the code. In the comment sections of the type definitions of those structures in **src/bfe/r_bfe_api.h** and **src/bfe/r_bfe_raa489204.h**, you can find instructions about what values can be assigned to the members. Keep in mind that some of the variable types are enumerations with fixed constants. The following code demonstrates only part of the definition and initialization of the structures:

```
/* Extended configuration structure */
const st_raa489204_ext_cfg_t g_bfe0_ext_cfg =
{
 .stack_size = BFE_STACK_SIZE,          // Do not modify!!!
 .d_ch_data_speed = BFE_D_RT_1000_KHZ, // Set daisy chain data speed according to
                       // hardware settings!
 ....

 .limit_overvolt_v = 4.50F,             // Set cell overvoltage limit in Volts!
 .limit_undervolt_v = 2.00F,            // Set cell undervoltage limit in Volts!
 .limit_ext_temp_v = 2.49F,             // Set external temperature input
                       // overvoltage limit in Volts!

 .wire_scan = BFE_WIRE_I_SCAN_1_5MS,    // Set open-wire scan current on time!
 .mux_scan = BFE_CELL_MUX_SCAN_0_5MS,  // Set cell multiplexer test scan timing!
 .flt_tot_samples = BFE_TOT_4_SMPL,    // Set number of consecutive fault
                       // conditions!
 .avging_volt = BFE_AVG_1_SMPL,         // Set number of voltage averaging samples!
 .avging_ext_temp = BFE_AVG_1_SMPL,     // Set number of temperatures averaging
                      // samples!
 ....

};

/* Set which cells exist in the battery stack! */
const uint16_t g_bfe0_cells_cfg[BFE_STACK_SIZE] =
{
     /* Device 1 */
     ( BFE_REG_MASK_CELL_1
     | BFE_REG_MASK_CELL_2
     | BFE_REG_MASK_CELL_3
     | BFE_REG_MASK_CELL_4
     | BFE_REG_MASK_CELL_11
     | BFE_REG_MASK_CELL_12
     | BFE_REG_MASK_CELL_13
     | BFE_REG_MASK_CELL_14),
     ....
};

/* Configuration structure */
const st_bfe_cfg_t g_bfe0_cfg =
{
 .p_cells_select = &g_bfe0_cells_cfg[0],                // Do not modify!!!
 .p_temps_select = &g_bfe0_ext_temps_cfg[0],            // Do not modify!!!
 .peripheral_type = BFE_COMMUNICATION_INTERFACE_SPI,    // Do not modify!!!
 .driver_cfg = BFE_DRIVER_NONE,                         // Do not modify!!!
 .fet_cfg = BFE_FET_NONE,                               // Do not modify!!!
 .p_extend = &g_bfe0_ext_cfg,                           // Do not modify!!!
};
```

3.5.3 Battery Abstraction Layer

The battery abstraction layer is configured in **bfe/r_bfe_cfg.h**. The file contains pre-processor macros (Table 31). They are used for:

- Inserting constants inside the source code (such as stack size and number of attempts)
- Enabling/disabling parts of the code (such as verify write into register, check input parameters of functions or work with battery emulator)
- Controlling certain features (such as byte or block SPI transfer mode or use watchdog timeout for entering sleep mode).

You can find the available options for the values in the description section inside the table but also in the comment sections of the source code. Keep in mind that some of the macro values can be only Boolean (0 or 1) and the others are whole numbers (unsigned).

Table 31. BFE Software Library Configuration Settings

Option (Macro name)	Default Value	Description
BFE_STACK_SIZE	3U	Number of devices in a stack (30 max).
BFE_CFG_SPI_MODE	1	BFE SPI transfer mode: 0 - Byte 1 - Block.
BFE_CFG_WDT_SLEEP_EN	0	BFE goes to sleep using watchdog timeout 0 - Disabled (Recommended) 1 - Enabled.
BFE_CFG_EN_PIN_RESET	0	For hard reset use: 0 - Daisy Chain communication (Recommended) 1 - ENABLE pin.
BFE_CFG_PARAM_CHECKING_EN	1	Functions check input parameters: 0 - Disable 1 - Enable (Recommended).
BFE_CFG_REG_WRITE_VERIFY_EN	1	Register verification after write command: 0 - Disable 1 - Enable (Recommended).
BFE_CFG_SCAN_DIAG_CMND_VERIFY_EN	1	Scan/ Diagnostic command verification: 0 - Disable 1 - Enable (Recommended).
BFE_CFG_USE_RESISTOR_LADDER	1	Work with MCB_PS4_Z resistor ladder board: 0 - Disable 1 - Enable.
BFE_CFG_STACK_IDENT_MAX	3U	Maximum attempts for stack identification.

3.5.4 Demo Application

The demo application is configured in **r_bms_cfg.h**. The file contains pre-processor macros (Table 32). They are used for inserting constants inside the source code like main loop time interval, number of loops before running device tests, and cell balancing algorithm settings. You can find the available options for the values in the description section inside the table but also in the comment sections of the source code. The macro values can be whole numbers (unsigned), fractional numbers (signed) or Boolean (0 or 1).

Table 32. BFE Demo Project Configuration Settings

Option	Default	Description
BMS_MEMORY_CHECK	10U (loops)	Memory check interval: After how many loops the BFE memory is tested?
BMS_SELF_DIAG	1000U (loops)	Self-diagnostic interval: After how many loops a complete BFE self-diagnostic test is accomplished?
BFE_BAL_MODE	0 (Timed)	BFE balancing mode: Is the balancing algorithm using Manual 1 or Timed 0 Balance Mode?
BMS_DELTA_V_MAX	0.02f (20mV)	Max cell delta voltage threshold. Cells having higher voltage difference compared to the one with lowest voltage are balanced.
BMS_DELTA_V_MAX_F_TH	0.5f (500mV)	Max cell delta voltage fault threshold. If any cell has higher voltage difference compared to the one with lowest voltage, cell balancing is inhibited.
BMS_CB_VCELL_MIN	3.4f (3.4V)	Minimum cell voltage to allow balancing.
BMS_CB_VCELL_MAX	4.2f (4.2V)	Maximum cell voltage to allow balancing.

Table 32. BFE Demo Project Configuration Settings

Option	Default	Description
BMS_CB_LOOPS_MAX	86U (60min)	Maximum number of cell balancing cycles.
BMS_CB_ON_TIMER	400U (40s)	The balancing time interval for odd and even patterns in manual balancing mode.
BMS_CB_OFF_TIMER	10U (1s)	Cell balancing off time interval per cycle. Needed for voltage relaxation before measurement.
TIME_PERIOD_MS_PERIODIC	100U (100 ms)	The main loop time interval.

3.6 Examples

This section demonstrates the API functions but also the communication drivers. You can use either the API functions from file **bfe/r_bfe_api.h** or their implementations from file **bfe/r_raa489204.h**. In the second case when keeping the application but changing the BFE, you have to replace all the functions rather than just reconnect the interface. For more examples refer to the sample code (files **r_bms.c** and **bfe/r_raa489204.c**).

- Initialization, setup and testing the BFE

```
/* Initialize the Battery Front End. */
bfe_err = R_RAA489204_Init(&g_bfe0_ctrl, &g_bfe0_cfg);

/* Check for error return */
if((bfe_err != BFE_SUCCESS) || (g_bfe0_ctrl.is_fault_detected == true)
{
    bfe_faults_handler();
}

/* Configure the Battery Front End. */
bfe_err = R_RAA489204_Setup(&g_bfe0_ctrl, &g_bfe0_cfg);

/* Check for error return */
if((bfe_err != BFE_SUCCESS) || (g_bfe0_ctrl.is_fault_detected == true)
{
    bfe_faults_handler();
}

/* Run self diagnostic. */
bfe_err = R_RAA489204_SelfDiag(&g_bfe0_ctrl, BFE_FULL_TEST);

/* Check for error return */
if((bfe_err != BFE_SUCCESS) || (g_bfe0_ctrl.is_fault_detected == true)
{
    bfe_faults_handler();
}
```

- Measuring cell voltages, battery voltages and temperatures (It is assumed that the BFE is already initialised and configured)

```
e_bfe_err_t bfe_err = BFE_SUCCESS; // Error code

static u_raa489204_all_meas_t s_meas_data_all[BFE_STACK_SIZE] = {0};

/* Clean the data structure. */
memset(&s_meas_data_all[0], 0, sizeof(s_meas_data_all));
```

```
/* Measure all voltages. */
bfe_err = R_RAA489204_AllGet(&g_bfe0_ctrl, & s_meas_data_all[0], true);

/* Check for error return */
if((bfe_err != BFE_SUCCESS) || (g_bfe0_ctrl.is_fault_detected == true)
{
    bfe_faults_handler();
}
```

- Running cell balancing

```
e_bfe_err_t bfe_err = BFE_SUCCESS; // Error code

static st_raa489204_cb_cfg_t s_cell_balance_cfg =
{
   .mode         = BFE_BALANCE_MODE_TIMED,
   .timeout      = BFE_BAL_01M_00S,
   .wait_time    = BFE_BAL_WAIT_0S,
   .external_cb = true,
   .pattern      = BFE_REG_BAL_MASK_NO_CELL,
   .cell_sel     = {0, 0, 0}
};

/* Select cells to be balanced. */
s_cell_balance_cfg.cell_sel[0] = 0x3FFF; // Select all cells
s_cell_balance_cfg.cell_sel[1] = 0x2227; // Select cells 1,2,3,6,10,14
s_cell_balance_cfg.cell_sel[2] = 0x0010; // Select cell 5

/* Enable cell balancing for odd cells. */
s_cell_balance_cfg.pattern[0] = BFE_REG_BAL_MASK_ODD_CELL;

bfe_err = R_RAA489204_CellBalanceCtrl(&g_bfe0_ctrl,
                      &s_cell_balance_cfg,
                      BFE_PROCESS_ENABLE);

/* Check for error return */
if((bfe_err != BFE_SUCCESS) || (g_bfe0_ctrl.is_fault_detected == true)
{
   bfe_faults_handler();
}

while(g_bfe0_ctrl.is_balancing == true)
{
   /* Check if cell balancing has completed. */
   bfe_err = R_RAA489204_IsCellBalancing(&g_bfe0_ctrl);

   /* Check for error return */
   if((bfe_err != BFE_SUCCESS) || (g_bfe0_ctrl.is_fault_detected == true)
   {
      bfe_faults_handler();
   }
}
```

```
/* Enable cell balancing for even cells. */
s_cell_balance_cfg.pattern[0] = BFE_REG_BAL_MASK_EVEN_CELL;

bfe_err = R_RAA489204_CellBalanceCtrl(&g_bfe0_ctrl,
                      &s_cell_balance_cfg,
                      BFE_PROCESS_ENABLE);

/* Check for error return */
if((bfe_err != BFE_SUCCESS) || (g_bfe0_ctrl.is_fault_detected == true)
{
  bfe_faults_handler();
}

while(g_bfe0_ctrl.is_balancing == true)
{
  /* Check if cell balancing has completed. */
  bfe_err = R_RAA489204_IsCellBalancing(&g_bfe0_ctrl);

  /* Check for error return */
  if((bfe_err != BFE_SUCCESS) || (g_bfe0_ctrl.is_fault_detected == true)
  {
    bfe_faults_handler();
  }
}
```

- Accessing Single Register

```
e_bfe_err_t bfe_err = BFE_SUCCESS; // Error code

st_raa489204_quick_reg_t s_reg_container =
{
  .device_address
  .p_reg = &g_raa489204_registers.user_register_1;
  .data = 0x0001;
};

bfe_err = R_RAA489204_RegisterWrite(&g_bfe0_ctrl, &s_reg_container);

/* Check for error return */
if((bfe_err != BFE_SUCCESS) || (g_bfe0_ctrl.is_fault_detected == true)
{
  bfe_faults_handler();
}
```

- Using the communication driver to broadcast a command

```
e_bfe_err_t bfe_err = BFE_SUCCESS; // Error status

static st_raa489204_spi_msg_t s_spi_msg = {0}; // SPI message container

/* Clean message structure. */
memset(&s_spi_msg, 0, sizeof(st_raa489204_spi_msg_t));

/* Send scan continuous command. */
```

```
s_spi_msg.command.device_address = BFE_DAISY_CHAIN_ADDRESS_ALL;
s_spi_msg.r_w_data = BFE_READ_REG;
s_spi_msg.command.frame = 0;
s_spi_msg.command.reg_number = 0;
s_spi_msg.command.tar_reg = & g_raa489204_registers.cmnd_scan_continuous;
s_spi_msg.command.data[0] = 0;

/* Send the command to all BFEs. */
bfe_err = bfe_spi_msg_send_resp_get(p_ctrl, &s_spi_msg);
BFE_ERROR_RETURN(BFE_SUCCESS == bfe_err, bfe_err); // Check for errors.
```

- Using the communication driver to write into multiple registers

```
e_bfe_err_t bfe_err = BFE_SUCCESS; // Error status

static st_raa489204_spi_msg_t s_spi_msg = {0}; // SPI message container

/* Clean message structure. */
memset(&s_spi_msg, 0, sizeof(st_raa489204_spi_msg_t));

/* Write setup registers of each device from the stack. */
for(uint16_t i = 0; i < p_ext_cfg->stack_size; i++ )
{
  s_spi_msg.command.device_address = (e_raa489204_dev_addr_t) (i + 1U);

  s_spi_msg.r_w_data = BFE_WRITE_REG;
  s_spi_msg.command.frame = 0;
  s_spi_msg.command.reg_number = 3;
  s_spi_msg.command.tar_reg = & g_raa489204_registers.undervoltage_fault;
  /* Over Voltage Limit Value. */
  s_spi_msg.command.data[0] = bfe_vcell_to_adc(4.25F);

  /* Under Voltage Limit Value. */
  s_spi_msg.command.data[1] = bfe_vcell_to_adc(2.50F);

  /* External Temperature Limit Value. */
  s_spi_msg.command.data[2] = bfe_vext_to_adc(3.00F);

  /* Write threshold registers of all BFEs. */
  bfe_err = bfe_spi_msg_send_resp_get(p_ctrl, &s_spi_msg);
  BFE_ERROR_RETURN(BFE_SUCCESS == bfe_err, bfe_err); // Check for errors.
}
```

Keep in mind that *bfe_spi_msg_send_resp_get()* is **not** a global function! You can use it for custom code development. If you are using the interface and want to work directly with registers or commands, consider the R_RAA489204_RegisterRead() or R_RAA489204_RegisterWrite() API functions.

4. Demo Application

The sample code contains a demo application that demonstrates the use and operation of the battery abstraction layer with API. Its source code can be found in file **r_bms.c**. There is a finite state machine and a cell balancing algorithm, controlled by a simple user interface. Figure 3 shows the state machine flow diagram. It generalizes the relations between states and modes as well as the conditions for transition between them. A state executes its function and moves to the next state or mode, while mode can remain static or loop inside until a transition flag is set. Fault State can be entered from any other if a BFE fault is detected or any error code different than BFE_SUCCESS is returned. The transitions are managed by a command line user interface. You can send simple commands by inputting numbers from 1 to 4 to select options from a list. Data are returned and visualized back. When a transition command is received, a respective transition flag is set. It can be set in any place of the code. However, the transition flags are processed on a single place in the code where the transition logic actually changes the state or mode of the state machine. The idea behind is to provide prioritization of transitions (for example, *Fault State* has highest priority and overrides other states when multiple transition flags are set). For more information about the user interface and running the demo, refer to the *RAA489204 Sample Code Quick Start Guide*.

Keep in mind that the demo application has limited capabilities and the sample code is **not** a system solution that can directly manage a battery rather than demonstrate the interface and provide easy access to BFE resources.

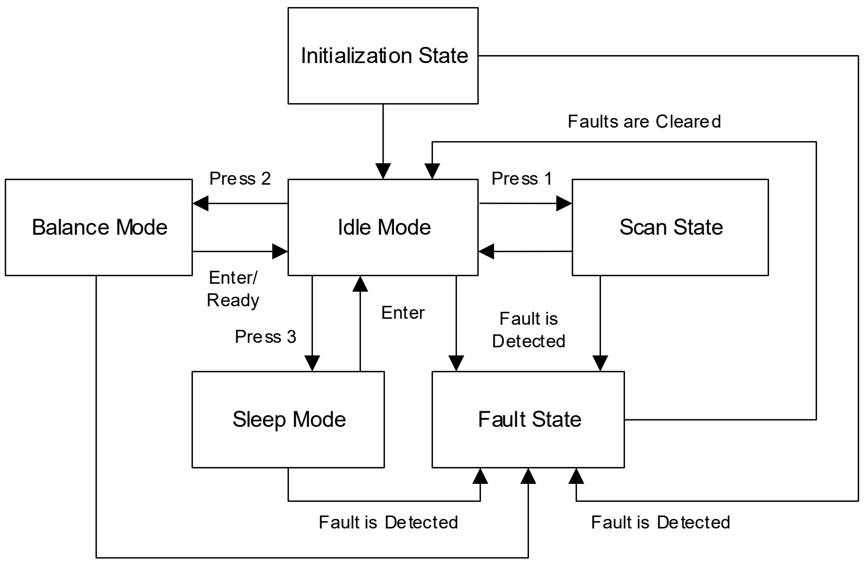

Figure 3. Sample Code State Machine Flow Diagram

After a power-on reset, the first entered state is Initialization State. Figure 4 shows the flow. The MCU initializes the BFE interface (initialize SPI, IRQs, timers and CRC calculator, reset, identify stack and read serial numbers of all devices in the stack). The reset command ensures that the current condition of every RAA489204 IC is known in case only the MCU has been reset. Then, full BFE setup is run followed by communication test and full self-diagnostic that includes ADC, multiplexer, EEPROM, open wires and balancing circuits test. The containing code is executed once, followed by a transition to Idle Mode.

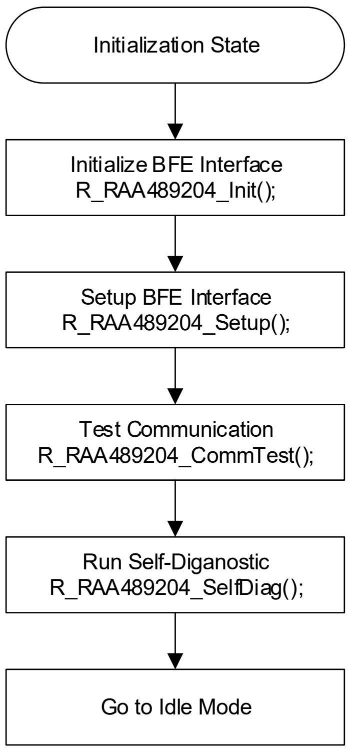

Figure 4. Initialization State Flowchart

In Idle Mode, the demo application loops waiting for user's input from the command line interface (Figure 5). There are software counters that track the number of loops and approximately the duration of this mode. Every 10 loops the MCU runs a memory test to check all configuration registers of the BFE for any corruption or unintentional change of any bit. On other hand, every 1000 loops a full self-test is made to check internal and external circuitry besides the memory.

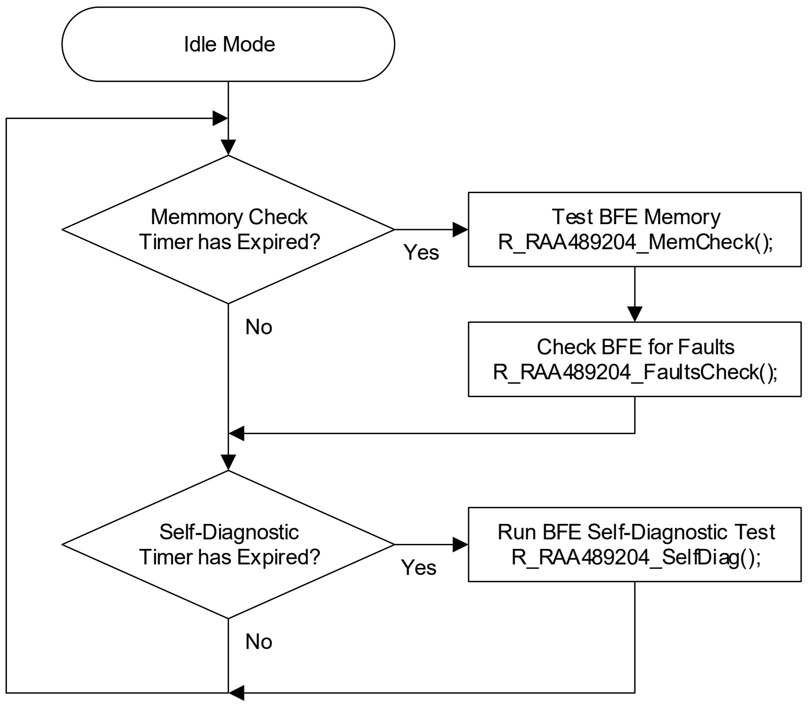

Figure 5. Idle Mode Flowchart

Figure 6 shows the Scan State flow. All voltages and temperatures are measured by calling the RAA489204_AllGet() API function, which sends a sequence of scan and read commands followed by a transition back to Idle Mode.

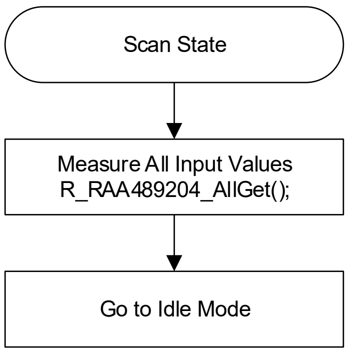

Figure 6. Scan State Flowchart

The RAA489204 sample code includes an algorithm for cell balancing. Figure 7 shows the algorithm flowchart. Cell balancing is a continuous process therefore the algorithm uses own state machine and loops the states. MCU runs periodically through the code and is not halting or sticking there (for example, during the time the charge is removed from cells) so that less resources are engaged. In the beginning of every loop, all cell voltages are measured and the cell with lowest voltage is found. All cell voltages are checked to be within the range where cell balancing is allowed or the process is stopped. Renesas recommends cell balancing only during charging. Nevertheless, it is not taken into account in this algorithm as the given system does not determine the current direction. In the next step, the voltage difference between each cell and the one with lowest voltage is calculated and compared with predeclared thresholds. By default, if the voltage difference is more than 20mV (Table 32), the cell is identified as balanced. On other hand, if the voltage difference exceeds a predefined threshold of 500mV, indicating a severely unbalanced battery pack, a problem with the battery is considered and an error is returned. If any cell is found to need balancing, the algorithm continues. First, the odd cells are unmasked and the respective balancing FETs are enabled for a predefined period of time. The time intervals are measured as number of cycles of the loop. Next, the same action is accomplished for the even cells. Afterwards, all CB FETs are turned OFF and a timer is activated to provide a recovery time for the cell voltages to obtain more accurate measurements before the next balancing cycle (avoiding the RC time constant of the filters and Li- chemistries). If the differences in cells voltages are within limits, the balancing activity is inhibited after the voltage measurement in the next cycle and a transition to Normal state is made. *IMPORTANT:* Avoid setting the CB delta voltage thresholds too low and the balancing intervals too long. This can result in nonconvergence and excess battery drain. To help avoid this situation, there is a balancing cycles counter that limits the number of loops.

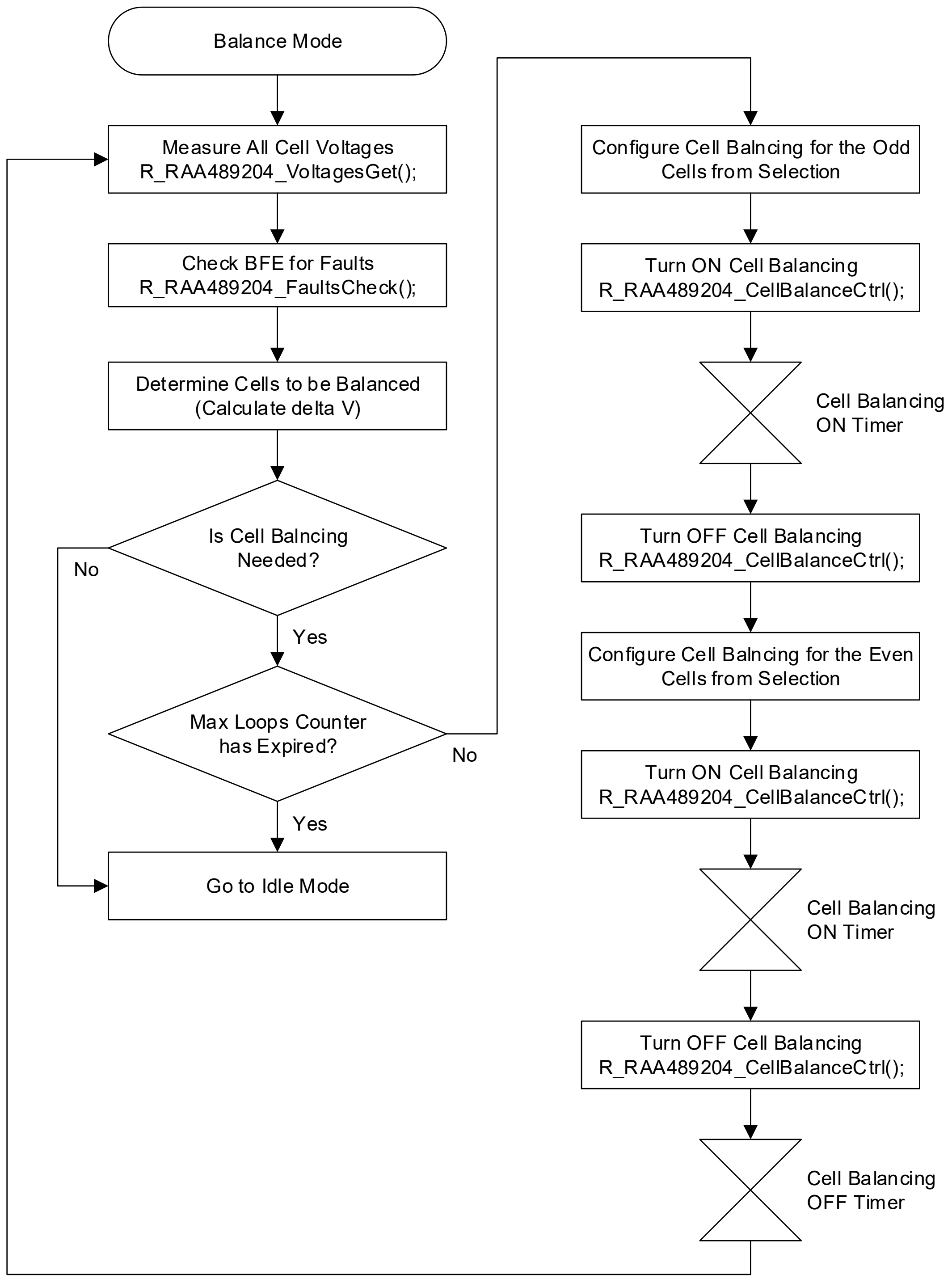

Figure 7. Cell Balancing Mode Flowchart

The demo application provides an option for entering Sleep Mode and waking up all devices in the stack. When Sleep Mode is active, the BFE enters Sleep Mode immediately and waits for user input to wake up and make a transition to Idle Mode (Figure 8). This part of the code demonstrates the use of the API function R_RAA489204_ModeSet();

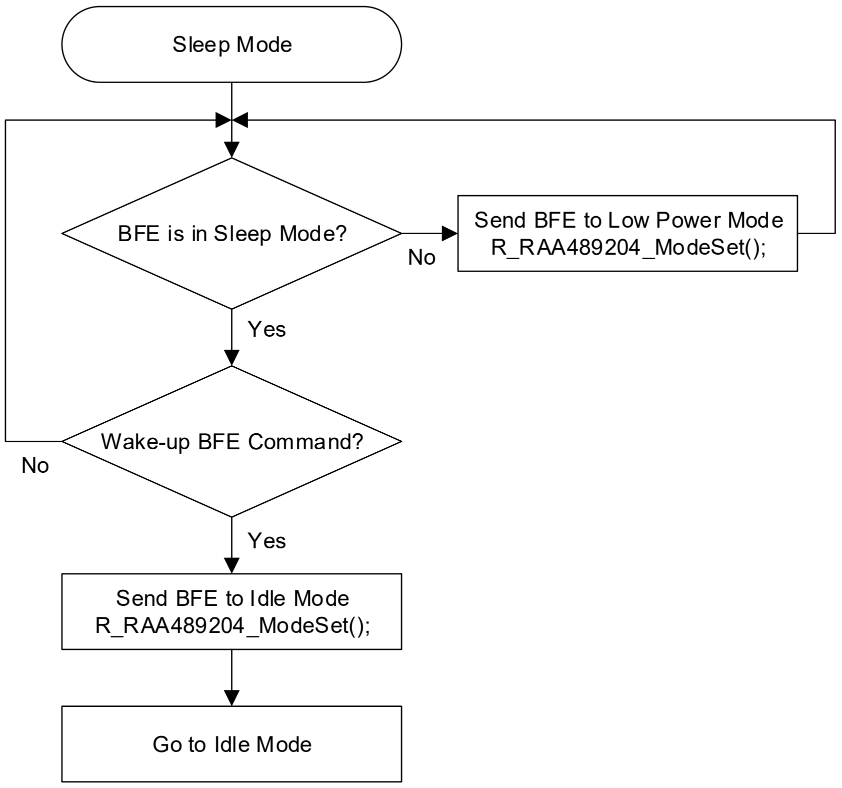

Figure 8. Sleep Mode Flowchart

The following conditions can force the state machine to enter Fault State in the next loop:

- An error code different than *BFE_SUCCESS* was returned by any API function, indicating incorrect behavior of the Battery Abstraction Layer, the Battery Front End, or a communication problem.
- The *g_bfe0_ctrl.is_fault_detected* flag was set after calling an API function that affects it, indicating that a fault condition is detected by the BFEs in the stack and at least one Fault Status Register is non-zero.
- An error code different than *FSP_SUCCESS* was returned by any API function from the Hardware Abstraction Layer of the MCU, indicating an error in the flexible software package.
- The BMS algorithm has encountered an error.

Figure 9 shows the state flow. If there is a fault in any BFE all Fault Status Registers are read. The information is returned by the API function R_RAA489204_FaultsAllRead() into a data structure and visualized in the user's interface. Some faults can be cleared and some errors can be resolved. Several fault and error management procedures are integrated into the demo application. They can be found in the source code in file **r_bms.c** and are described in the *Advanced Safety of RAA489204 Battery Front End Application Note*. If the fault is not clearable or the errors – not resolved the state machine halts and waits for user's interaction.

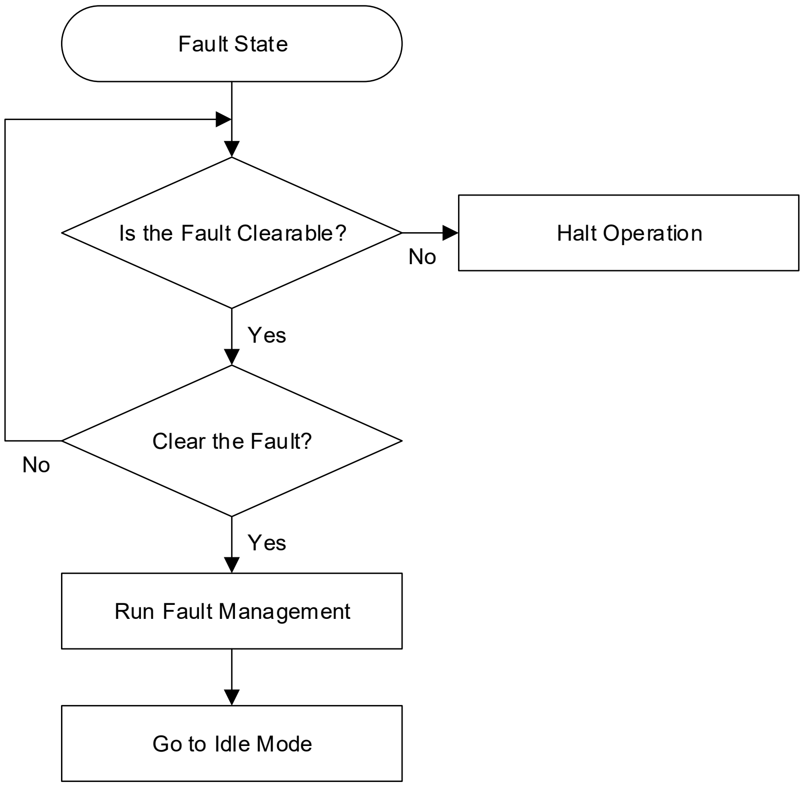

Figure 9. Fault State Flowchart

5. Revision History

Revision	Date	Description
2.00	Sept 28, 2022	Total rewrite to fix the structure and add more details.
1.03	Feb 17, 2022	Updated Target Device and Cell Balancing sections. Updated Table 2.
1.02	Dec 21, 2021	Updated the following sections: ▪ Configuration of the Battery Front-End Library ▪ Configuration and Control Structures ▪ Description of Functions ▪ Using the Software Library ▪ Updated Table 4.
1.01	Nov 1, 2021	Updated Battery Front-End Library section. Updated Tables 1 and 3. Updated Configuration and Control Structures section. Updated Battery Management System Demo section. Updated Initialization section. Updated Obtaining Measurement section. Updated Hardware Assembly section. Updated Figures 3, 4, 6, 7, 8, 13, 14, 17, 18, 20, 21, 22, 23, and 24.
1.00	Sep 28, 2021	Initial release.