RAA489220 Battery Front End Sample Code

This software manual provides a detailed description and application guidelines for using the RAA489220 sample code. It includes API functions and application examples to speed up the design of high voltage battery management systems that consist of multiple (stacked) battery manager ICs.

Contents

1. Introduction

The RAA489220 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:

- Easy access to RAA489220 resources and advanced features
- Custom configurations
- Simplified status and error monitoring
- Integrated fault diagnostics and processing
- Full compatibility with Renesas Advanced (RA) Family 32-bit MCUs
- Application Programming Interface (API) for easy integration

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.
- Before proceeding, Renesas recommends reviewing the *EK-RA4W1 Manual*, the *RAA489220 Datasheet*, and the *RAA489220 Evaluation Kit Manual* to get acquainted with the MCU and BFE features.

2. RAA489220 Battery Front End Overview

2.1 Features

RAA489220 is a 10 cell Li-ion battery management front end IC that have the following features:

- High hot plug rating: 44V
- Autonomous protection functions
- Qualified for industrial temperature range: -40°C to +85°C
- Monitors and manages 10 Li-Ion cells
- Built-in low side FET drivers
- Supports wide range of current sense resistors
- Monitors two external temperature inputs
- High-voltage GPIO for fuse blow
- Cell voltage measurement accuracy: ±10mV
- 12-bit voltage, current, and temperature measurements
- I^2C interface with optional CRC for higher reliability
- Integrated self-test features

This manual and the sample code are used for the case when the external MCU is present and RAA489220 is *not* operating autonomously. The MCU contains the sample code and it controls the Battery Front End. Figure 1 shows a typical application.

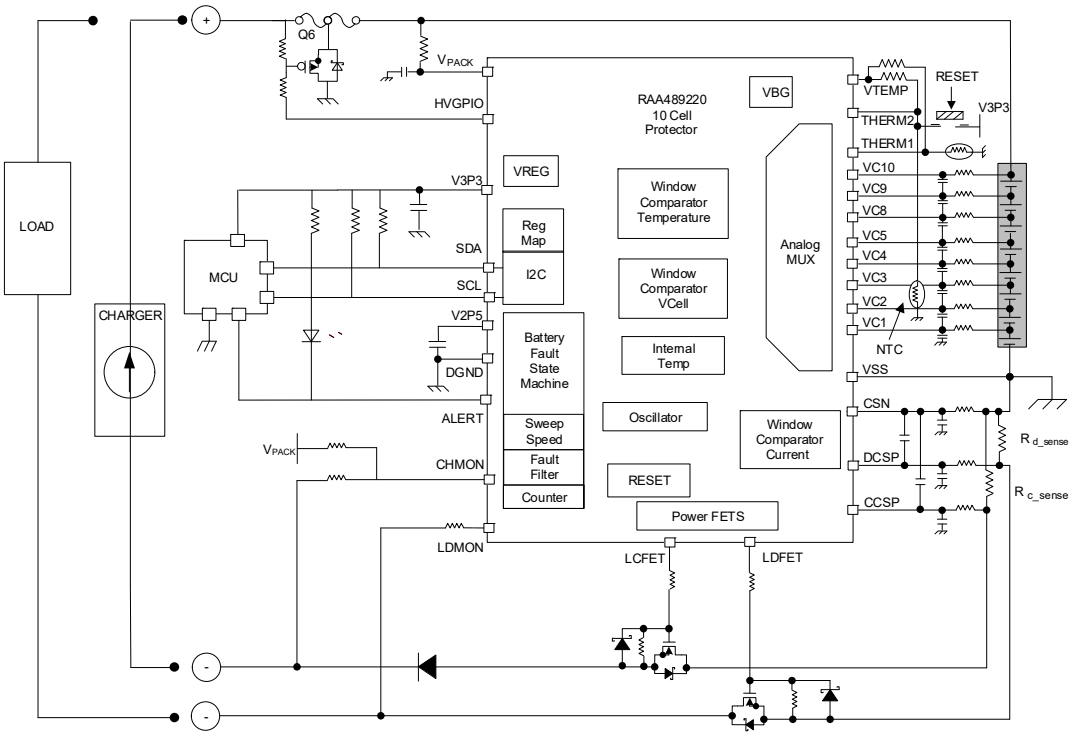

Figure 1. Typical Application of RAA489220

2.2 Applications

- Power tools
- Handheld electronics
- Battery protector

2.3 RAA489220 Sample Code Structure

The RAA48220 sample code contains the following software components:

- Battery Front End Application Programming Interface (BFE API)
- RAA4489220 implementation of the BFE API
- Demo application with finite-sate machine, cell balancing algorithm, and command line user interface
- Configuration file for Renesas Flexible Software Package that is used to generate the peripheral drivers (Hardware Abstraction Layer (HAL)) 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 containing macros and dedicated 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' implementations	
ra_gen	---			Instantiation of HAL modules and **main.c** that calls the entry point	
ra_cfg	fsp_cfg		r_*_cfg.h	Configuration options files	
src	---		hal_entry.c	Entry point that calls the application main	
			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	
			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	
			usb_pcdc.h	USB driver definitions	
	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_raa489220.c	Actual code for the interface implementation	
			r_raa489220.h	Definitions, structures, enumerations and declarations of the API functions	

The BFE instance and its structures define the contract and features common to most of the BFEs. The BFE instance for RAA489220 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.

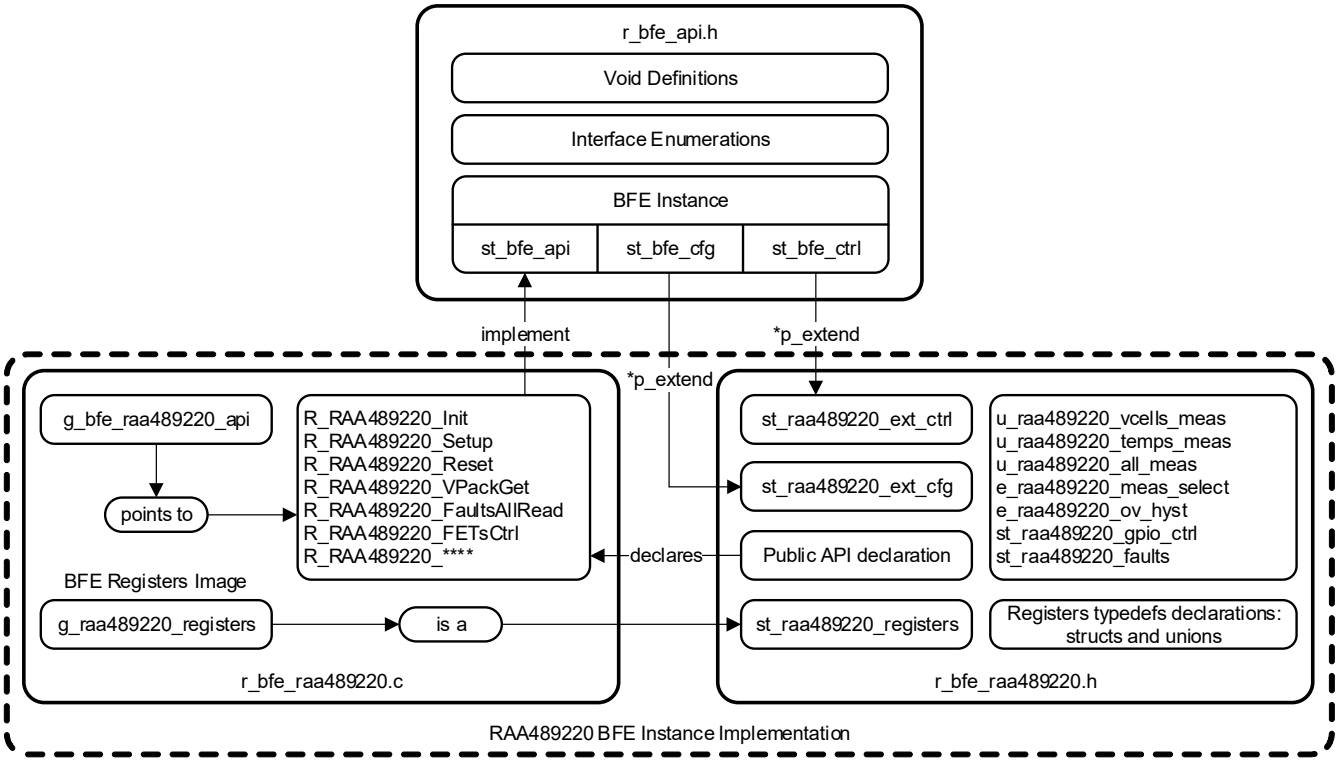

Figure 2. Main Software Components of BFE Instance Implementation

3. RAA489220 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_raa489220.h**. Table 2 shows the content of the extended control structure. It contains information about the I^2C slave address of the BFE communication interface and the behavior of the built-in voltage regulator during Low Power Mode. It also points to the timings structure (Table 3). *IMPORTANT*: Ensure that reset has completed before communicating further with the BFE or that the measurement is not taking too much time. The timings are fixed and their values are assigned in the initialization API function.

Table 2. Members of the RAA489220 Extended Control Structure

typedef struct st_raa489220_ext_ctrl		
Member	**Type**	**Description**
i2c_slave_address	uint16_t	Battery Front End I2C communication slave address
weak_regulator	bool	Weak regulator state flag
timings	st_raa489220_timings_t	Structure holding the BFE timings

Table 3. Members of the RAA489220 Timings Structure

typedef struct st_raa489220_ext_ctrl		
Member	**Type**	**Description**
reset_wait_time_ms	uint32_t	Reset processing time [us]
reset_therm2_hold_time_ms	uint32_t	Hold time for THERM2 pin to trigger a reset [ms]
busy_bit_timeout_us	uint32_t	Busy bit read timeout during measurement [us]

Table 4 shows the content of the extended configuration structure. Its members correspond to all fixed settings of the BFE. The extended configuration structure stores constant variables directly related to the hardware, that is, shunt resistance, over/undervoltage lockout threshold (Li-Ion chemistry), short-circuit delay time (load current profile), alert signal MCU input pin (routing between the master BFE and the MCU), and others. The limits are entered as real values (Voltages, Amperes). Some of the variables have types that are defined as enumerations. Therefore, you can select from a list of options facilitating the device configuration. Be aware 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. For more information about the available options and their effect over the BFE performance, refer to the *RAA489220 Datasheet*.

Table 4. Members of the RAA489220 Extended Configuration Structure

typedef struct st_raa489220_ext_cfg		
Member	**Type**	**Description**
r_shunt_charge_mohm	const float	Shunt resistance for charge current [mohm]
r_shunt_discharge_mohm	const float	Shunt resistance for discharge current [mohm]
thold_overvolt_lockout_v	const float	Cell overvoltage lockout threshold [V] (range: 3.0 - 4.5V, step: 100mV)
thold_overvolt_v	const float	Cell overvoltage threshold [V] (range: 3.23 - 4.5V, step: 10mV)
thold_undervolt_v	const float	Cell undervoltage threshold [V] (range: 1.5 - 3.0V, step: 100mV)
thold_undervolt_lockout_v	const float	Cell undervoltage lockout threshold [V] (range: 1.5 - 3.0V, step: 100mV)
thold_delta_cell_overvolt_v	const float	Delta cell overvoltage threshold [V] (range: 0.5 - 2.0V, step: 100mV)
hyst_overovltage	const e_raa489220_ov_hyst_t	Cell overvoltage hysteresis
hyst_undervltage	const e_raa489220_uv_hyst_t	Cell undervoltage hysteresis
thold_charge_over_temp	const e_raa489220_cot_th_t	Over-temperature threshold when charging
thold_charge_under_temp	const e_raa489220_cut_th_t	Under-temperature threshold when charging
thold_discharge_over_temp	const e_raa489220_dot_th_t	Over-temperature threshold when discharging
thold_discharge_under_temp	const e_raa489220_dut_th_t	Under-temperature threshold when discharging
thold_charge_overcurrent_a	const float	Charge overcurrent threshold [A]
thold_discharge_overcurrent_a	const float	Discharge overcurrent threshold [A]
delay_ch_d_oc	const e_raa489220_oc_delay_t	Discharge/Charge overcurrent delay
thold_sc	const e_raa489220_sc_th_t	Short-circuit threshold
delay_sc	const e_raa489220_sc_delay_t	Short-circuit delay
ieoc_threshold	const e_raa489220_ieoc_th_t	End of charge current threshold

Table 4. Members of the RAA489220 Extended Configuration Structure (Cont.)

typedef struct st_raa489220_ext_cfg		
Member	**Type**	**Description**
discharge_overcurrent_disable	const bool	Disable discharge overcurrent protection
comm_timeout	const e_raa489220_comm_tout_t	Communication timeout
self_test_enable	const bool	Include self-testing in single scan sequence
open_wire_test_enable	const bool	Include open wire test in the self-test loop
fuse_blow_enable	const bool	Enable fuse blow on the BFE
alert_pin_cfg	const e_raa489220_alert_cfg_t	Select control source for the ALERT pin
scan_delay	const e_raa489220_scan_delay_t	Auto scan interval under normal conditions
fault_delay	const e_raa489220_fault_delay_t	Auto scan interval when fault condition is detected
lp_mode_settings	const e_st_raa489220_lp_mode_t	Low Power Mode Settings when transition from Auto scan
pin_alert	const bsp_io_port_pin_t	ALERT pin
pin_reset	const bsp_io_port_pin_t	THERM2 pin

Table 5 shows the options for **hyst_overovltage** constant variable, which are listed in the RAA489220 Vcell Overvoltage Hysteresis Enumeration. The selected constant is used in the overvoltage comparison during cell voltage measurement.

Table 5. RAA489220 Vcell Overvoltage Hysteresis Enumeration

typedef enum e_raa489220_ov_hyst		
Constant	**Value**	**Description**
BFE_CELL_OV_HYST_25MV	0x00	Vcell OV hysteresis 25mV
BFE_CELL_OV_HYST_50MV	0x01	Vcell OV hysteresis 50mV
BFE_CELL_OV_HYST_100MV	0x02	Vcell OV hysteresis 100mV
BFE_CELL_OV_HYST_200MV	0x03	Vcell OV hysteresis 200mV
BFE_CELL_OV_HYST_250MV	0x04	Vcell OV hysteresis 250mV
BFE_CELL_OV_HYST_300MV	0x05	Vcell OV hysteresis 300mV
BFE_CELL_OV_HYST_350MV	0x06	Vcell OV hysteresis 350mV
BFE_CELL_OV_HYST_400MV	0x07	Vcell OV hysteresis 400mV

Table 6 shows the options for **hyst_undervltage** constant variable which are listed in the RAA489220 Vcell Undervoltage Hysteresis Enumeration. The selected constant is used in the undervoltage comparison during cell voltage measurement.

Table 6. RAA489220 Vcell Undervoltage Hysteresis Enumeration

typedef enum e_raa489220_ov_hyst		
Constant	**Value**	**Description**
BFE_CELL_UV_HYST_100MV	0x00	Vcell UV hysteresis 100mV
BFE_CELL_UV_HYST_200MV	0x01	Vcell UV hysteresis 200mV
BFE_CELL_UV_HYST_300MV	0x02	Vcell UV hysteresis 300mV
BFE_CELL_UV_HYST_400MV	0x03	Vcell UV hysteresis 400mV
BFE_CELL_UV_HYST_500MV	0x04	Vcell UV hysteresis 500mV
BFE_CELL_UV_HYST_600MV	0x05	Vcell UV hysteresis 600mV
BFE_CELL_UV_HYST_700MV	0x06	Vcell UV hysteresis 700mV
BFE_CELL_UV_HYST_800MV	0x07	Vcell UV hysteresis 800mV

Table 7 shows the options for **thold_charge_over_temp** constant variable, which are listed in the RAA489220 Charge Over-Temperature Threshold Enumeration. The selected constant is used in the external over-temperature comparison during temperature measurement.

Table 7. RAA489220 Charge Over-Temperature Threshold Enumeration

typedef enum e_raa489220_cot_th		
Constant	**Value**	**Description**
BFE_COT_THLD_0_678V	0x00	Charge over-temperature threshold 0.678V
BFE_COT_THLD_0_590V	0x01	Charge over-temperature threshold 0.590V
BFE_COT_THLD_0_518V	0x02	Charge over-temperature threshold 0.518V
BFE_COT_THLD_0_450V	0x03	Charge over-temperature threshold 0.450V
BFE_COT_THLD_0_390V	0x04	Charge over-temperature threshold 0.390V
BFE_COT_THLD_0_338V	0x05	Charge over-temperature threshold 0.338V
BFE_COT_THLD_0_294V	0x06	Charge over-temperature threshold 0.294V
BFE_COT_THLD_0_258V	0x07	Charge over-temperature threshold 0.258V

Table 8 shows the options for **thold_charge_under_temp** constant variable, which are listed in the RAA489220 Charge Under-Temperature Threshold Enumeration. The selected constant is used in the external under-temperature comparison during temperature measurement.

Table 8. RAA489220 Charge Under-Temperature Threshold Enumeration

typedef enum e_raa489220_cut_th		
Constant	**Value**	**Description**
BFE_CUT_THLD_1_622V	0x00	Charge under-temperature threshold 1.622V
BFE_CUT_THLD_1_526V	0x01	Charge under-temperature threshold 1.526V
BFE_CUT_THLD_1_422V	0x02	Charge under-temperature threshold 1.422V
BFE_CUT_THLD_1_314V	0x03	Charge under-temperature threshold 1.314V
BFE_CUT_THLD_1_198V	0x04	Charge under-temperature threshold 1.198V
BFE_CUT_THLD_1_086V	0x05	Charge under-temperature threshold 1.086V
BFE_CUT_THLD_0_974V	0x06	Charge under-temperature threshold 0.974V
BFE_CUT_THLD_0_866V	0x07	Charge under-temperature threshold 0.866V

Table 9 shows the options for **thold_discharge_over_temp** constant variable, which are listed in the RAA489220 Discharge Over-Temperature Threshold Enumeration. The selected constant is used in the external over-temperature comparison during temperature measurement.

Table 9. RAA489220 Discharge Over-Temperature Threshold Enumeration

typedef enum e_raa489220_dot_th		
Constant	**Value**	**Description**
BFE_DOT_THLD_0_222V	0x00	Discharge over-temperature threshold 0.222V
BFE_DOT_THLD_0_194V	0x01	Discharge over-temperature threshold 0.194V
BFE_DOT_THLD_0_170V	0x02	Discharge over-temperature threshold 0.170V
BFE_DOT_THLD_0_150V	0x03	Discharge over-temperature threshold 0.150V
BFE_DOT_THLD_0_130V	0x04	Discharge over-temperature threshold 0.130V
BFE_DOT_THLD_0_114V	0x05	Discharge over-temperature threshold 0.114V
BFE_DOT_THLD_0_098V	0x06	Discharge over-temperature threshold 0.098V
BFE_DOT_THLD_0_086V	0x07	Discharge over-temperature threshold 0.086V

Table 10 shows the options for thold_discharge_under_temp constant variable, which are listed in the RAA489220 Discharge Under-Temperature Threshold Enumeration. The selected constant is used in the external under-temperature comparison during temperature measurement.

Table 10. RAA489220 Discharge Under-Temperature Threshold Enumeration

typedef enum e_raa489220_dut_th		
Constant	**Value**	**Description**
BFE_DUT_THLD_1_622V	0x00	Discharge under-temperature threshold 1.622V
BFE_DUT_THLD_1_526V	0x01	Discharge under-temperature threshold 1.526V
BFE_DUT_THLD_1_422V	0x02	Discharge under-temperature threshold 1.422V
BFE_DUT_THLD_1_314V	0x03	Discharge under-temperature threshold 1.314V
BFE_DUT_THLD_1_198V	0x04	Discharge under-temperature threshold 1.198V
BFE_DUT_THLD_1_086V	0x05	Discharge under-temperature threshold 1.086V
BFE_DUT_THLD_0_974V	0x06	Discharge under-temperature threshold 0.974V
BFE_DUT_THLD_0_866V	0x07	Discharge under-temperature threshold 0.866V

Table 11 shows the options for **delay_ch_d_oc** constant variable, which are listed in the RAA489220 Charge/Discharge Overcurrent Delay Enumeration. The selected constant is determining the delay source for overcurrent delay interval.

Table 11. RAA489220 Charge/Discharge Overcurrent Delay Enumeration

typedef enum e_raa489220_oc_delay	
Constant	**Description**
BFE_OC_USE_SCAN_DELAY	Use Scan Delay for Charge/Discharge Overcurrent detection.
BFE_OC_USE_FAULT_DELAY	Use Fault Delay for Charge/Discharge Overcurrent detection.

Table 12 shows the options for **thold_sc** constant variable, which are listed in the RAA489220 Short-circuit Threshold Enumeration. The selected constant is determining the threshold voltage (respectively current) for short-circuit detection with analog comparator.

Table 12. RAA489220 Short-Circuit Threshold Enumeration

typedef enum e_raa489220_sc_th		
Constant	**Value**	**Description**
BFE_SC_THLD_12_5MV	0x00	Short circuit threshold 12.5mV
BFE_SC_THLD_25MV	0x01	Short circuit threshold 25mV
BFE_SC_THLD_50MV	0x02	Short circuit threshold 50mV
BFE_SC_THLD_100MV	0x03	Short circuit threshold 100mV
BFE_SC_THLD_200MV	0x04	Short circuit threshold 200mV

Table 13 shows the options for **delay_sc** constant variable, which are listed in the RAA489220 Short-Circuit Delay Enumeration. The selected constant is determining the delay during short-circuit detection with analog comparator.

Table 13. RAA489220 Short-circuit Delay Enumeration

typedef enum e_raa489220_sc_delay		
Constant	**Value**	**Description**
BFE_SC_DELAY_0FF	0x00	No short circuit delay
BFE_SC_DELAY_0_1MS	0x01	Short circuit delay 0.1ms
BFE_SC_DELAY_1MS	0x02	Short circuit delay 1ms
BFE_SC_DELAY_10MS	0x03	Short circuit delay 10ms

Table 14 shows the options for **ieoc_threshold** constant variable, which are listed in the RAA489220 End of Charge Current Threshold Enumeration. The selected constant is determining the threshold voltage (respectively current) threshold for end of charge detection.

Table 14. RAA489220 End of Charge Current Threshold Enumeration

typedef enum e_raa489220_ieoc_th		
Constant	**Value**	**Description**
BFE_IEOC_THLD_0_65MV	0x00	IEOC threshold 0.65mV
BFE_IEOC_THLD_0_70MV	0x01	IEOC threshold 0.70mV
BFE_IEOC_THLD_1MV	0x02	IEOC threshold 1mV
BFE_IEOC_THLD_3MV	0x03	IEOC threshold 3mV

Table 15 shows the options for **comm_timeout** constant variable, which are listed in the RAA489220 Communication Timeout Enumeration. The selected constant is determining the timeout after which the BFE detects communication fault and goes to Low Power or Ship Mode.

Table 15. RAA489220 Communication Timeout Enumeration

typedef enum e_raa489220_comm_tout		
Constant	**Value**	**Description**
BFE_COMM_TO_OFF	0x00	Communication timeout is disabled
BFE_COMM_TO_0_1S	0x01	Communication timeout is 100ms
BFE_COMM_TO_1S	0x02	Communication timeout is 1s
BFE_COMM_TO_5S	0x03	Communication timeout is 5s

Table 16 shows the options for **e_raa489220_alert_cfg_t** constant variable, which are listed in the RAA489220 ALERT Pin Internal Connection Configuration Enumeration. The selected constant is determining how the pin is connected to the internal logic and the event that asserts it.

Table 16. RAA489220 ALERT Pin Internal Connection Configuration Enumeration

typedef enum e_raa489220_alert_cfg		
Constant	**Value**	**Description**
BFE_ALRT_NORMAL	0x00	Use normal ALERT pin functionality
BFE_ALRT_BUSY_CONNECT	0x01	Connect Busy Mask bit to the ALERT pin
BFE_ALRT_V_I_EOC_CONNECT	0x02	Connect the status of the VEOC and IEOC bits to the ALERT pin
BFE_ALRT_CH_LD_PRES_CONNECT	0x03	Connect the status of the CH PRESI and LD PRESI bits to the ALERT pin

Table 17 shows the options for **scan_delay** constant variable, which are listed in the RAA489220 Scan Delay Enumeration. The selected constant is determining the time interval length between system scans in Auto Scan Mode under nominal operating conditions.

Table 17. RAA489220 Scan Delay Enumeration

typedef enum e_raa489220_scan_delay		
Constant	**Value**	**Description**
BFE_SCAN_DELAY_0S	0x00	Scan Delay 0s
BFE_SCAN_DELAY_0_1S	0x01	Scan Delay 100ms
BFE_SCAN_DELAY_0_5S	0x02	Scan Delay 500ms
BFE_SCAN_DELAY_1S	0x03	Scan Delay 1s

Table 18 shows the options for **fault_delay** constant variable, which are listed in the RAA489220 Fault Delay Enumeration. The selected constant is determining the time interval between the system scans in Auto Scan Mode after a fault is detected (before going to Low Power/Ship Mode).

Table 18. RAA489220 Fault Delay Enumeration

typedef enum e_raa489220_fault_delay		
Constant	**Value**	**Description**
BFE_FAULT_DELAY_OFF	0x00	No fault delay
BFE_FAULT_DELAY_0_1S	0x01	Fault Delay 100ms
BFE_FAULT_DELAY_1S	0x02	Fault Delay 1s
BFE_FAULT_DELAY_5S	0x03	Fault Delay 5s

Table 19 shows the options for **lp_mode_settings** constant variable, which are listed in the RAA489220 Low Power Mode Settings Enumeration. The selected constant is determining the behavior of the internal voltage regulator resulting in transition to Low Power or Ship Mode under certain conditions. For more information, refer to the *RAA489220 Datasheet*.

Table 19. RAA489220 Low Power Mode Settings Enumeration

typedef enum e_st_raa489220_lp_mode		
Constant	**Value**	**Description**
BFE_SHIP_MODE	0x00	Lowest power; Weak regulators; Register values are not retained.
BFE_LP_MODE_STRONG_REG	0x11	Strong regulators; Register values are retained.

3.2 Register Bank

The register bank holds all BFE registers in its fields. Each member is a nested structure with a predefined data type (Table 20). It contains the register address itself, the register type which has fixed values, the default register value loaded after reset, and the last read register value. Table 21 shows the register type options. The information is used by the middleware communication driver to assemble the data packet and manage the I^2C communication, but the information is also used to check if the correct default values are loaded on start-up or after reset.

Table 20. Members of the RAA489220 Register Container Structure

typedef struct st_raa489220_register		
Member	**Type**	**Description**
address	**const** e_raa489220_reg_addr_t	Register address
type	**const** e_raa489220_reg_type_t	Register type
def_value	**const** uint16_t	Register default value
value	uint16_t	Register value

Table 21. RAA489220 Register Type Enumeration

typedef enum e_raa489220_reg_type	
Constant	**Description**
READ_ONLY	Read only register
READ_WRITE	Read/Write register
COMMAND	Command

To set a target register to the communication drivers, you need to use the register bank member as a function parameter. To return the obtained register value into the register bank, you must use the pointer to the nested structure value member, that is:

```
bfe_i2c_register_read(p_ctrl,
                      g_raa489220_registers.system_config_1,
                     &g_raa489220_registers.system_config_1.value);
```

You can find the full register bank declaration in **bfe/r_raa489220.h** and definition in **bfe/r_raa489220.c**. The following code demonstrates a part of it:

```
/* RAA489220 registers' bank */
st_raa489220_registers_t g_raa489220_registers =
{
 .pack_voltage      = {.address = BFE_PACK_VOLTAGE, .type = READ_ONLY,
                       .value = 0x00, .def_value = BFE_PACK_VOLTAGE_DEF},

 .cell_max_voltage  = {.address = BFE_CELL_MAX_VOLTAGE,  .type = READ_ONLY,
                       .value = 0x00, .def_value = BFE_CELL_MAX_VOLTAGE_DEF},

 .cell_min_voltage  = {.address = BFE_CELL_MIN_VOLTAGE,  .type = READ_ONLY,
                       .value = 0x00, .def_value = BFE_CELL_MIN_VOLTAGE_DEF},

 .thermistor_1      = {.address = BFE_THERMISTOR_1,      .type = READ_ONLY,
                       .value = 0x00, .def_value = BFE_THERMISTOR_1_DEF},
 ....

 .system_config_1   = {.address = BFE_SYSTEM_CONFIG_1,   .type = READ_WRITE,
                       .value = 0x00, .def_value = BFE_SYSTEM_CONFIG_1_DEF},

 .measure_select    = {.address = BFE_MEASURE_SELECT,    .type = READ_WRITE,
                       .value = 0x00, .def_value = BFE_MEASURE_SELECT_DEF},

 .system_config_2   = {.address = BFE_SYSTEM_CONFIG_2,   .type = READ_WRITE,
                       .value = 0x00, .def_value = BFE_SYSTEM_CONFIG_2_DEF},

 .system_config_3   = {.address = BFE_SYSTEM_CONFIG_3,   .type = READ_WRITE,
                       .value = 0x00, .def_value = BFE_SYSTEM_CONFIG_3_DEF},

 .ov_eoc            = {.address = BFE_OV_AND_EOC,        .type = READ_WRITE,
                       .value = 0x00, .def_value = BFE_OV_AND_EOC_DEF},
 ....

 .cmnd_fault_stat   = {.address = BFE_FAULT_AND_STATUS,  .type = COMMAND,
                       .value = 0x00, .def_value = BFE_FAULT_AND_STATUS_DEF},

 .cmnd_measurements = {.address = BFE_READ_MEASUREMENTS, .type = COMMAND,
                       .value = 0x00, .def_value = BFE_READ_MEASUREMENTS_DEF},

 .cmnd_i_pack       = {.address = BFE_I_PACK,            .type = COMMAND,
                      .value = 0x00, .def_value = BFE_I_PACK_DEF},
};
```

3.3 Private (Static) Functions

Table 22 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 BFE, change of state, or certain diagnostic. For more details, refer to the examples in the next sections and the comments in the source code in **bfe/r_raa489220.c**.

Table 22. Static Functions Defined in the Source Code

Function	Description
static e_bfe_err_t bfe_reset_hard (st_bfe_ctrl_t * const p_ctrl);	Pulls up THERM2 pin for more than 500ms to reset the BFE.
static e_bfe_err_t bfe_reset_soft (st_bfe_ctrl_t * const p_ctrl);	Sets the Soft Reset Bit to reset all register values to the factory settings, including data registers.
static e_bfe_err_t bfe_i2c_register_write (st_bfe_ctrl_t * const p_ctrl, st_raa489220_register_t tar_reg, uint16_t data);	Writes data in a BFE register.
static e_bfe_err_t bfe_i2c_register_read (st_bfe_ctrl_t * const p_ctrl, st_raa489220_register_t tar_reg, uint16_t * const p_data);	Read data from a BFE register.
static e_bfe_err_t bfe_i2c_block_read (st_bfe_ctrl_t * const p_ctrl, st_raa489220_register_t consttar_reg, uint16_t * const p_data);	Reads data from multiple BFE registers.
static fsp_err_t bfe_i2c_event_validate (void);	Validates that the I2C transfer has completed or was aborted.
static e_bfe_err_t bfe_crc8_calculate (uint8_t * const p_data_buffer, uint32_t length, uint32_t * const p_crc_result);	Calculates CRC8 checksum.
static e_bfe_err_t bfe_meas_complete_wait (st_bfe_ctrl_t * const p_ctrl, const e_st_raa489220_meas_select_t selection);	Waits in a loop until the measurement has completed.
static float bfe_adc_to_vpack (uint16_t value);	Converts ADC value to pack voltage.
static float bfe_adc_to_ipack (uint16_t value, float r_shunt_mohm);	Converts ADC value to pack current.
static float bfe_adc_to_vcell (uint16_t value);	Converts ADC value to cell voltage.
static float bfe_adc_to_vtherm (uint16_t value);	Converts ADC value to thermistor voltage.
static float bfe_adc_to_vregpin (uint16_t value);	Converts ADC value to 3.3V and 2.5V regulator voltage.
static float bfe_adc_to_vbg2ref (uint16_t value);	Converts ADC value to secondary Band Gap reference voltage.

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_RAA489220_Init

e_bfe_err_t R_RAA489220_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. It modifies the **p_ctrl->is_initialized**, **p_ctrl->is_low_power** and **p_ctrl->is_cont_scanning** flags in **p_ctrl**.	
Operation	▪ Checks function parameters. ▪ Calculates BFE timings. ▪ Initializes a CRC peripheral module. ▪ Initializes a I^2C interface peripheral module to connect to the BFE device. ▪ Initializes the ISR peripheral modules ▪ Resets the BFE. ▪ Reads BFE die revision, manufacturing ID and device ID.	
Precondition	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_reset_hard()
	BMS_ERR_...	Inherit from bfe_i2c_register_read()

3.4.2 R_RAA489220_Deinit

e_bfe_err_t R_RAA489220_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 variables **p_ctrl->is_initialized** and **p_ctrl->is_low_power**.	
Operation	▪ Checks function parameters. ▪ Deinitializes the I2C peripheral module. ▪ Deinitializes the CRC peripheral module. ▪ 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_FSP	Error in the FSP layer

3.4.3 R_RAA489220_Setup

e_bfe_err_t R_RAA489220_Setup (st_bfe_ctrl_t * const p_bfe_ctrl, st_bfe_cfg_t const * const p_bfe_cfg);		
Description	This function configures the BFE by writing into all device setup registers. It extracts the necessary data from the control **p_ctrl** and configuration **p_cfg** structures.	
Operation	▪ Checks function parameters. ▪ Unlocks registers write. ▪ Sends a multiple register write command to set System Config 1, System Config 2, System Config 3, OV and EOC Thresholds, DVcellOV UV Thresholds, SCC OT/UT Thresholds and Current Threshold Registers. ▪ Verifies if values are correctly written. ▪ Locks registers write.	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
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_INVALID_CONF	An invalid configuration parameter was detected.
	BFE_ERR_WRITE_VERIFY	Register write verification error
	BMS_ERR_...	Inherit from bfe_i2c_register_write().
	BMS_ERR_...	Inherit from bfe_i2c_register_read().

3.4.4 R_RAA489220_Reset

e_bfe_err_t R_RAA489220_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. It modifies the **p_ctrl->is_low_power** and **p_ctrl->is_cont_scanning** flags in **p_ctrl**.	
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.	
Precondition	The BFE interface should have already been initialized.	
Warnings	You should reconfigure the BFE after reset by calling R_RAA489220_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()

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 23 lists the supported reset options by the BFE.

Table 23. 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_RAA489220_ModeSet

e_bfe_err_t R_RAA489220_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 **p_ctrl->is_low_power**, **p_ctrl->is_cont_scanning** and **p_ext_ctrl->weak_regulator** flags in **p_ctrl**. Be aware that when put in Reset State, the BFE makes a transition to *Idle Mode* afterwards.	
Operation	▪ Checks input parameters. ▪ Changes bits into Device Setup Registers 1, 2, and 3 so that the BFE enters the desired mode.	
Precondition	The BFE interface should have already been initialized.	
Warnings	After reset you must rewrite all configuration registers by calling R_RAA489220_Setup!	
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_UNSUPPORTED_MODE	Invalid mode was selected
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized
	BFE_ERR_INVALID_ARGUMENT	Invalid mode was selected
	BMS_ERR_...	Inherit from bfe_i2c_register_read()
	BMS_ERR_...	Inherit from bfe_i2c_register_write()
	BMS_ERR_...	Inherit from bfe_reset_soft()

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 24 shows the supported modes by the sample code.

Table 24. BFE States and Modes Enumeration

typedef enum e_bfe_mode	
Constant	**Description**
BFE_STATE_RESET	Initial state when all circuits and oscillators are off.
BFE_MODE_IDLE	The device is ready waiting for a task to be executed.
BFE_MODE_AUTOSCAN	Periodic execution of measurements.
BFE_MODE_LOW_POWER_MODE	The BFE is currently in low power mode.
BFE_MODE_SHIP	Lowest power consumption suitable for long-term storage.

3.4.6 R_RAA489220_ModeRead

e_bfe_err_t R_RAA489220_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	-	
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 24 shows the expected modes returned as a result.

3.4.7 R_RAA489220_CommTest

e_bfe_err_t R_RAA489220_CommTest (st_bfe_ctrl_t * const p_bfe_ctrl);		
Description	This function tests the communication between the MCU and the BFE.	
Operation	▪ Checks function parameters. ▪ Reads Device ID Register to test communication. ▪ Compares the acquired Device ID with the one from **p_ctrl** structure.	
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_COMM_FAULT	Communication fault was found.
	BFE_ERR_...	Inherit from bfe_i2c_register_read().

3.4.8 R_RAA489220_SelfDiag

e_bfe_err_t R_RAA489220_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. The **only** supported option is 'BFE_FULL_TEST'. If the BFE is in Auto Scan Mode and a fault is detected **is_cont_scanning** and **is_low_power** flags are updated.
Operation	▪ Checks function parameters. ▪ Triggers a Single System Scan that includes regulators test, cells and temperature inputs open-wire test, oscillator test and other (Please, refer to the RAA489220 datasheet). ▪ Waits for the scan to complete. ▪ Reads the Fault Register to check for any asserted flags.
Precondition	The BFE interface should have already been initialized.

Warnings	Always check the **p_ctrl->is_fault_detected** flag after calling this function. Be aware that open wire test behavior depends on the **p_ext_cfg->open_wire_test_enable** setting.	
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_i2c_register_write().
	BFE_ERR_...	Inherit from bfe_meas_complete_wait().

3.4.9 R_RAA489220_MemCheck

e_bfe_err_t R_RAA489220_MemCheck (st_bfe_ctrl_t * const p_bfe_ctrl, e_bfe_mem_check_option_t option);		
Description	This function checks if the default configuration registers values are loaded correctly. The **only** supported option is BFE_CHECK_DEF_VALS.	
Operation	▪ Checks function parameters. ▪ Reads the current configuration registers values. ▪ Compares the obtained values with the expected defaults.	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
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_i2c_register_read ().

3.4.10 R_RAA489220_VPackGet

e_bfe_err_t R_RAA489220_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. You must always select to trigger a measurement as the function does not support reading of prior measured values.
Operation	▪ Checks function parameters. ▪ Select a measurement type. ▪ Trigger a measurement. ▪ Wait for the Busy bit to clear. ▪ Read and convert the ADC value.
Precondition	The BFE interface should have already been initialized.
Warnings	Always set **trigger** input parameter to be **true**. Be aware that the measured values are not compared for faults in the BFE internally.

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_INVALID_ARGUMENT	Invalid option was selected.
	BFE_ERR_...	Inherit from bfe_i2c_register_write().
	BFE_ERR_...	Inherit from bfe_i2c_register_read().
	BFE_ERR_...	Inherit from bfe_meas_complete_wait().

3.4.11 R_RAA489220_IPackGet

e_bfe_err_t R_RAA489220_IPackGet (st_bfe_ctrl_t * const p_bfe_ctrl, st_bfe_i_pack_meas_t * const p_values, bool trigger);		
Description	This function acquires the battery pack current with the BFE. The returned data is converted into current. The pointer **p_values** points to a structure where the measured currents can be found. The function measures both charge and discharge currents. You must always select to trigger a measurement as the function does not support reading of prior measured values.	
Operation	▪ Checks function parameters. ▪ Select a measurement type. ▪ Trigger a measurement. ▪ Wait for the Busy bit to clear. ▪ Read and convert the ADC value.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always set **trigger** input parameter to be **true**. Be aware that the measured values are not compared for faults in the BFE internally.	
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_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_i2c_register_write().
	BFE_ERR_...	Inherit from bfe_i2c_register_read().
	BFE_ERR_...	Inherit from bfe_meas_complete_wait().

The void pointer *p_values* points to the structure where this API function returns the measured values. The structure is defined in the file **bfe/r_bfe_api.h** and has the following content:

```
/** BFE battery pack current measurement structure */
typedef struct st_bfe_i_pack_meas
{
    float                           i_charge;         ///< Charge current [A]
    float                           i_discharge;      ///< Discharge current [A]
} st_bfe_i_pack_meas_t;
```

3.4.12 R_RAA489220_ VCellsGet

e_bfe_err_t R_RAA489220_ VCellsGet (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 a union where the measured cell voltages can be found. You must always select to trigger a measurement as the function does not support reading of prior measured values.	
Operation	▪ Checks function parameters. ▪ Select a measurement type. ▪ Trigger a measurement. ▪ Wait for the Busy bit to clear. ▪ Read and convert the ADC value.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always set **trigger** input parameter to be **true**. Be aware that the measured values are not compared for faults in the BFE internally.	
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_INVALID_ARGUMENT	Invalid option was selected.
	BFE_ERR_...	Inherit from bfe_i2c_register_write().
	BFE_ERR_...	Inherit from bfe_i2c_register_read().
	BFE_ERR_...	Inherit from bfe_meas_complete_wait().

The void pointer *p_values* points to the union where this API function returns the measured values. The union type is redefined in the file **bfe/r_raa489220.h** and has the following content:

```
/** RAA489220 measured cells voltages data structure */
typedef union u_raa489220_vcells_meas
{
    float vector[11];

    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_pack;                           ///< Pack voltage [V]
    } measurements;
} u_raa489220_vcells_meas_t;
```

3.4.13 R_RAA489220_Temps

e_bfe_err_t R_RAA489220_Temps (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_temp_meas_t * const p_values, bool trigger);		
Description	This function acquires the external temperature voltage. The returned data is converted into voltage. The pointer **p_values** points to a union where the measured voltages can be found. You must always select to trigger a measurement as the function does not support reading of prior measured values.	
Operation	▪ Checks function parameters. ▪ Enable VTEMP pin voltage. ▪ Select a measurement type. ▪ Trigger a measurement. ▪ Wait for the Busy bit to clear. ▪ Read and convert the ADC value. ▪ Disable VTEMP pin voltage.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always set **trigger** input parameter to be **true**, Be aware that the measured values are not compared for faults in the BFE internally.	
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_INVALID_ARGUMENT	Invalid option was selected.
	BFE_ERR_...	Inherit from bfe_i2c_register_write().
	BFE_ERR_...	Inherit from bfe_i2c_register_read().
	BFE_ERR_...	Inherit from bfe_meas_complete_wait().

The void pointer *p_values* points to the union where this API function returns the measured values. The union type is redefined in the file **bfe/r_raa489220.h** and has the following content:

```
/** RAA489220 measured temperatures data structure */
typedef union u_raa489220_temps_meas
{
    float vector[3];

    struct
    {
        float v_therm_1;                    ///< Thermistor 1 voltage [V]
        float v_therm_2;                    ///< Thermistor 2 voltage [V]
        float v_temp;                       ///< VTEMP pin voltage [V]
    } measurements;
} u_raa489220_temps_meas_t;
```

3.4.14 R_RAA489220_AllGet

e_bfe_err_t R_RAA489220_AllGet (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_all_meas_t * const p_values, bool trigger);		
Description	This function acquires the voltages of all cells, the battery pack voltage, charge and discharge currents, external temperatures and regulator voltage. The returned data is converted into voltages and currents. The pointer **p_values** points to a union where the measured values can be found. You must always select to trigger a measurement as the function does not support reading of prior measured values.	
Operation	▪ Checks function parameters. ▪ Select a measurement type. ▪ Trigger a measurement. ▪ Wait for the Busy bit to clear. ▪ Read and convert the ADC value.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always set **trigger** input parameter to be **true**. Be aware that the measured values are not compared for faults in the BFE internally.	
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_INVALID_ARGUMENT	Invalid option was selected.
	BFE_ERR_...	Inherit from bfe_i2c_register_write().
	BFE_ERR_...	Inherit from bfe_i2c_register_read().
	BFE_ERR_...	Inherit from bfe_meas_complete_wait().

The void pointer '*p_values*' points to the union where this API function returns the measured values. The union's type is redefined in the file **bfe/r_raa489220.h** and has the following content:

```
/** RAA489220 measure all data structure */
typedef union u_raa489220_all_meas
{
    float vector[19];

    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_pack;///< Pack voltage [V]
        float v_pin_v3p3;///< Pin V3P3 voltage [V]
        float v_pin_v2p5;///< Pin V2P5 voltage [V]
        float v_ref_bg2;///< Second Band Gap reference voltage [V]
        float v_temp;///< VTEMP pin voltage [V]
        float v_therm_1;///< Thermistor 1 voltage [V]
        float v_therm_2;///< Thermistor 2 voltage [V]
        float i_discharge;///< Discharge current [A]
        float i_charge;///< Charge current [A]
    } measurements;
} u_raa489220_all_meas_t;
```

3.4.15 R_RAA489220_SysScanGet

e_bfe_err_t R_RAA489220_SysScanGet(st_bfe_ctrl_t * const p_bfe_ctrl, bfe_other_meas_t * const p_values, bool trigger);		
Description	This function runs a single system scan and reads the acquired values for pack voltage, min and max cell voltage, charge and discharge current and temperatures. The returned data is converted into voltages and currents. The pointer **p_values** points to a union where the measured values can be found. You can select whether to trigger a measurement and read the obtained values or only read the last obtained values.	
Operation	▪ Checks function parameters. ▪ Select a measurement type. ▪ Trigger a measurement. ▪ Wait for the Busy bit to clear. ▪ Read and convert the ADC value.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Be aware that the measured values are not compared for faults in the BFE internally.	
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_INVALID_ARGUMENT	Invalid option was selected.
	BFE_ERR_...	Inherit from bfe_i2c_register_write().
	BFE_ERR_...	Inherit from bfe_i2c_register_read().
	BFE_ERR_...	Inherit from bfe_meas_complete_wait().

The void pointer '*p_values*' points to the union where this API function returns the measured values. The union's type is redefined in the file '**bfe/r_raa489220.h**' and has the following content:

```
/** RAA489220 single system scan data structure */
typedef union u_raa489220_sys_scan_meas
{
    float vector[7];

    struct
    {
        float v_pack;                   ///< Pack voltage [V]
        float v_cell_max;               ///< Maximum cell voltage [V]
        float v_cell_min;               ///< Minimum cell voltage [V]
        float v_therm_1;                ///< Thermistor 1 voltage [V]
        float v_therm_2;                ///< Thermistor 2 voltage [V]
        float i_discharge;              ///< Discharge current [A]
        float i_charge;                 ///< Charge current [A]
    } measurements;
} u_raa489220_sys_scan_meas_t;
```

3.4.16 R_RAA489220_FaultsAllRead

e_bfe_err_t R_RAA489220_FaultsAllRead (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_faults_t * const p_faults);		
Description	This function reads the Fault Register. The pointer **p_faults** points to a structure where the fault data can be found. It modifies the **p_ctrl->is_low_power** flags in **p_ctrl**.	
Operation	▪ Checks function parameters. ▪ Reads Fault Register. ▪ Copies fault bits to faults data structure.	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
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_i2c_register_read().

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

```
/** RAA489220 fault data structure */
typedef struct st_raa489220_faults
{
    bool    flt_charge_over_current;///< Charge Overcurrent Fault
    bool    flt_discharge_over_current;///< Discharge Overcurrent Fault
    bool    flt_short_circuit;      ///< Short-Circuit Fault
    bool    flt_under_temeprature;      ///< Under-Temperature Fault
    bool    flt_over_temeprature;        ///< Over-Temperature Fault
    bool    flt_undervoltage;           ///< Undervoltage Fault
    bool    flt_overvoltage;            ///< Overvoltage Fault
    bool    flt_delta_cell_ov;           ///< Delta Cell Voltage Fault
    bool    flt_comm_timeout;            ///< Communication Time Out
    bool    flt_self_test;               ///< Self Test Fault
    bool    flt_lock_out;               ///< Lockout (voltage) Fault
    bool    flt_vtemp;                   ///< Open-wire on temperature pins
    bool    flt_internal_over_temp;     ///< Internal Over Temperature Fault
    bool    flt_open_wire;              ///< Open-Wire Fault
    bool    flt_regulator;               ///< Under/over voltage conditions at the
                    ///< V3P3 and V2P5 pins
} st_raa489220_faults_t;
```

3.4.17 R_RAA489220_FaultsCheck

e_bfe_err_t R_RAA489220_FaultsCheck (st_bfe_ctrl_t * const p_bfe_ctrl);		
Description	This function checks ALERT pin and reads the Fault Register. When a fault is detected the **is_fault_detected** flag is set in **p_ctrl**. If the BFE is in auto scan mode and a fault is detected **is_cont_scanning** and **is_low_power** flags are updated.	
Operation	Checks function parameters. Checks the ALERT pin for assertion if its normal functionality is used. Check Fault Register for set flags.	
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_i2c_register_read().

3.4.18 R_RAA489220_FaultsAllClear

e_bfe_err_t R_RAA489220_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. ▪ Sets Clear All Faults bit in System Config 1 Register. ▪ Reads Fault Register to verify fault clear.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the boolean variable pointed by **p_success** 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_RAA489220_CellBalanceCtrl

e_bfe_err_t R_RAA489220_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 is unsupported by the current API implementation.	
Operation	-	
Precondition	-	
Warnings	This function is not supported by the current API implementation!	
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_ERR_UNSUPPORTED_FEATURE	This feature is not supported by the current API implementation.

3.4.20 R_RAA489220_IsCellBalancing

e_bfe_err_t R_RAA489220_IsCellBalancing (st_bfe_ctrl_t * const p_bfe_ctrl);		
Description	This function is unsupported by the current API implementation.	
Operation	-	
Precondition	-	
Warnings	This function is not supported by the current API implementation!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure
Return Values	BFE_ERR_UNSUPPORTED_FEATURE	This feature is not supported by the current API implementation.

3.4.21 R_RAA489220_ContScanCtrl

e_bfe_err_t R_RAA489220_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 is unsupported by the current API implementation.	
Operation	-	
Precondition	-	
Warnings	This function is not supported by the current API implementation!	
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_ERR_UNSUPPORTED_FEATURE	This feature is not supported by the current API implementation.

3.4.22 R_RAA489220_WatchdogCtrl

e_bfe_err_t R_RAA489220_WatchdogCtrl (st_bfe_ctrl_t * const p_bfe_ctrl, bfe_watchdog_ctrl_t * const p_control_options);		
Description	This function is unsupported by the current API implementation.	
Operation	-	
Precondition	-	
Warnings	This function is not supported by the current API implementation!	
Parameters	p_bfe_ctrl	Pointer to the BFE control structure
Return Values	BFE_ERR_UNSUPPORTED_FEATURE	This feature is not supported by the current API implementation.

3.4.23 R_RAA489220_FETsCtrl

e_bfe_err_t R_RAA489220_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 controls the power FETs. The FET driver output state is set with the parameters **c_fet_state** and **d_fet_state**. You can control more than one pair of charge and discharge FETs by using the parameter **group_num**.	
Operation	▪ Checks function parameters. ▪ Reads System Config 1 Register. ▪ Changes LCFET and LDFET bits according to input parameters. ▪ Writes back to System Config 1 Register. ▪ Reads System Config 1 Register to verify status of the FETs control bits.	
Precondition	The BFE interface should have already been initialized.	
Warnings	Call the function only when the device is in *Idle Mode*!	
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.
	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 FET state was not correct.
	BFE_ERR_FET_CONTROL	The FET state was not changed according to input parameters.
	BFE_ERR_INVALID_OPERATION	The FETs cannot be controlled manually only in *Idle Mode*.
	BMS_ERR_...	Inherit from bfe_i2c_register_read().
	BMS_ERR_...	Inherit from bfe_i2c_register_write().

The states are fixed. Table 25 shows the available options for parameter **c_fet_state** controlling the charge FET, and Table 26 shows the available options for parameter **d_fet_state** controlling the discharge FET.

Table 25. BFE Discharge FET State Options Enumeration

typedef enum e_bfe_dfet_state	
Constant	**Description**
BFE_DFET_ON	The discharge FET is conducting.
BFE_DFET_OFF	The discharge FET is not conducting.

Table 26. BFE Charge FET State Options Enumeration

typedef enum e_bfe_cfet_state	
Constant	**Description**
BFE_CFET_ON	The charge FET is conducting.
BFE_CFET_OFF	The charge FET is not conducting.

3.4.24 R_RAA489220_HVGPIOCtrl

e_bfe_err_t R_RAA489220_HVGPIOCtrl(st_bfe_ctrl_t * const p_bfe_ctrl, bfe_gpio_ctrl_t * const p_control_options);		
Description	This function controls the HVGPIO used for Fuse blow and battery pack disabling. The void pointer **p_ options** points to the GPIO parameters.	
Operation	▪ Checks function parameters. ▪ Reads System Config 1 Register. ▪ Changes HVGPIO_ASSERT bit according to input parameters. ▪ Writes back to System Config 1 Register. ▪ Reads System Config 1 Register to verify status of the HVGPIO control bit.	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
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_INVALID_ARGUMENT	The selected FET state was not correct.
	BMS_ERR_...	Inherit from bfe_i2c_register_read().
	BMS_ERR_...	Inherit from bfe_i2c_register_write().

The output level for the HVGPIOs is controlled by the void function parameter **p_control_options**. It points to a control structure that is redefined in file **bfe/r_raa489220.h**. Table 27 shows the content of that structure having only one member, and Table 28 shows the fixed options listed in RAA489220 HVGPIO Output State Options Enumeration.

Table 27. Members of the RAA489220 HVGPIO Control Structure

typedef struct st_raa489220_gpio_ctrl		
Member	**Type**	**Description**
hvgpio_state	e_raa489220_hvgpio_states_t	HVGPIO output state

Table 28. RAA489220 HVGPIO Output State Options Enumeration

Constant	Description
BFE_HVGPIO_LOW	High-voltage output is asserted to Vss pin
BFE_HVGPIO_HIGH_Z	High-voltage output is in high-impedance state

3.4.25 R_RAA489220_RegisterRead

e_bfe_err_t R_RAA489220_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	-	
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_i2c_register_read().

The void type API function parameter **p_register** provides path to the target register and data. It is redefined as a RAA489220 Quick Register Access Structure in file **bfe/r_raa489220.h**. Table 29 shows the content the structure. The register address is assigned with a member of the register bank that is described in section Register Bank. The read data are available after calling the function in the third member of the structure data.

Table 29. Members of the RAA489220 Quick Register Access Structure

typedef struct st_raa489220_quick_reg		
Member	**Type**	**Description**
p_targer_register	st_raa489220_register_t	Pointer to register address data container.
data	uint16_t	Register data field.

3.4.26 R_RAA489220_RegisterWrite

e_bfe_err_t R_RAA489220_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	-	
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_i2c_register_write().

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 *RAA489220 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 desired 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_raa489220.h** you can find instructions about what values can be assigned to the members. Be aware 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_raa489220_ext_cfg_t g_bfe0_ext_cfg =
{
 .r_shunt_charge_mohm = 1000.0f,// Set charge current shunt resistance in
                     // mohms!
 .r_shunt_discharge_mohm = 100.0f,// Set discharge current shunt resistance
                     // in mohms!

 .thold_overvolt_lockout_v = 4.30f,// Set the overvoltage lockout threshold
                     // in Volts! (3.0-4.5V, 100mV)
 .thold_overvolt_v = 4.25f,               // Set the overvoltage threshold in
                     // Volts! (3.23-4.5V, 10mV)
 .thold_undervolt_v = 2.70f,              // Set the undervoltage threshold in
                     // Volts! (1.5-3.0V, 100mV)
 .thold_undervolt_lockout_v = 2.00f,     // Set the undervoltage lockout threshold
                     // in Volts! (1.5-3.0V, 100mV)
 .thold_delta_cell_overvolt_v = 0.5f,// Set the delta cell overvoltage
```

```
                    // threshold in Volts! (0.5-2.0V, 100mV)

 .hyst_overovltage = BFE_CELL_OV_HYST_100MV,        // Set the cell overvoltage
                        // hysteresis!
 .hyst_undervltage = BFE_CELL_UV_HYST_300MV,        // Set the cell undervoltage
                        // hysteresis!

 .thold_charge_over_temp = BFE_COT_THLD_0_294V,     // Set the charge over-
                        // temperature threshold!
 .thold_charge_under_temp = BFE_CUT_THLD_1_198V,    // Set the charge under-
                        // temperature threshold!
 .thold_discharge_over_temp = BFE_DOT_THLD_0_150V,  // Set the discharge over-
                        // temperature threshold!
 .thold_discharge_under_temp = BFE_DUT_THLD_1_622V, // Set the discharge under-
                        // temperature threshold!

....

 .pin_alert = BSP_IO_PORT_01_PIN_05,                // Set MCU pin connected to the
                        // ALERT pin of the BFE
 .pin_reset = BSP_IO_PORT_00_PIN_14,                // Set MCU pin connected to the
                        // THERM2 pin of the BFE
};

/* Configuration structure */
const st_bfe_cfg_t g_bfe0_cfg =
{
 .p_cells_select = &g_bfe0_cells_cfg,                  // Do not modify!!!
 .p_temps_select = &g_bfe0_ext_temps_cfg,              // Do not modify!!!
 .peripheral_type = BFE_COMMUNICATION_INTERFACE_I2C,   // Do not modify!!!
 .driver_cfg = BFE_DRIVER_LOW_SIDE,                    // Do not modify!!!
 .fet_cfg = BFE_FET_CONFIG_PARALLEL,                   // Select FETs configuration!
 .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 30). They are used for enabling/disabling parts of the code (that is, verify write into register, check input parameters). 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. Be aware that all macro values are Boolean (0 or 1).

Table 30. BFE Software Library Configuration Settings

Option (Macro name)	Default Value	Description
BFE_I2C_CRC_USE	1	CRC check in I2C communication: 0 - Do not use/ 1 - Use (Recommended)
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).

3.5.4 Demo Application

The demo application is configured in **r_bms_cfg.h**. The file contains pre-processor macros (Table 31). They are used for inserting constants inside the source code like main loop time interval, number of loops before running device tests and data visualization period during auto scan. 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 are whole numbers (unsigned).

Table 31. BFE Demo Project Configuration Settings

Option	Default	Description
BMS_FAULT_CHECK	10U (loops)	Faults check interval: After how many loops the BFE is tested for faults?
BMS_SELF_DIAG	50U (loops)	Self-diagnostic interval: After how many loops a complete BFE self-diagnostic test is accomplished?
BMS_AUTO_SCAN	50U (loops)	Display data interval: After how many loops the measured data is displayed in Auto Scan Mode?
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 implementation from file **bfe/r_raa489220.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_raa489220.c**).

- Initialization, setup and testing the BFE

```
/* Initialize the Battery Front End. */
bfe_err = R_RAA489220_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();
}

/* Check if registry defaults are correctly loaded. */
bfe_err = R_RAA489220_MemCheck(&g_bfe0_ctrl, BFE_CHECK_DEF_VALS);

/* Check for error return */
if(bfe_err != BFE_SUCCESS)
{
    bfe_faults_handler();
}

/* Configure the Battery Front End. */
bfe_err = R_RAA489220_Setup(&g_bfe0_ctrl, &g_bfe0_cfg);

/* Check for error return */
if(bfe_err != BFE_SUCCESS)
{
    bfe_faults_handler();
}

/* Run self diagnostic. */
```

```
bfe_err = R_RAA489220_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, temperatures, battery voltages and current (It is assumed that the BFE is already initialized and configured)

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

static u_raa489220_all_meas_t s_meas_data_all = {0};

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

/* Measure all voltages. */
bfe_err = R_RAA489220_AllGet(&g_bfe0_ctrl, & s_meas_data_all, true);

/* Check for error return */
if(bfe_err != BFE_SUCCESS)
{
    bfe_faults_handler();
}
```

- Enabling the CFET and DFET

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

bfe_err = R_RAA489220_FETsCtrl(&g_bfe0_ctrl, 1, BFE_CFET_ON, BFE_DFET_ON);

/* Check for error return */
if(bfe_err != BFE_SUCCESS)
{
  bfe_faults_handler();
}
```

- Turning ON Auto Scan

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

bfe_err = R_RAA489220_ModeSet(&g_bfe0_ctrl, BFE_MODE_AUTOSCAN);

/* Check for error return */
if(bfe_err != BFE_SUCCESS)
{
  bfe_faults_handler();
}
```

- Accessing Single Register

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

```
st_raa489220_quick_reg_t s_reg_container =
{
   .p_targer_register = &g_raa489220_registers.measure_select;
   .data = 0x0080;
};

bfe_err = R_RAA489220_RegisterWrite(&g_bfe0_ctrl, &s_reg_container);

/* Check for error return */
if(bfe_err != BFE_SUCCESS)
{
   bfe_faults_handler();
}
```

- Using the communication drivers to write and then read a register

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

u_raa489220_config1_reg_t system_config_1 = {0};

/* Set System Configure 1 Register */
system_config_1.value_b.LCFET_EN = 0; // Ensure Charge FET is off
system_config_1.value_b.LDFET_EN = 0; // Ensure Discharge FET is off

bfe_err = bfe_i2c_register_write(p_ctrl,
                      g_raa489220_registers.system_config_1,
                      system_config_1.value);
BFE_ERROR_RETURN(BFE_SUCCESS == bfe_err, bfe_err); // Check for errors.

bfe_err = bfe_i2c_register_read(p_ctrl,
                     g_raa489220_registers.system_config_1,
                     &g_raa489220_registers.system_config_1.value);
BFE_ERROR_RETURN(BFE_SUCCESS == bfe_err, bfe_err); // Check for errors.
```

- Using the communication driver to read multiple registers

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

static uint16_t s_data_buffer[BFE_MAX_READ_DATA_LGTH] = {0};

/* Read measured data. */
bfe_err = bfe_i2c_block_read(p_ctrl,
                        g_raa489220_registers.cmnd_measurements,
                        &s_data_buffer[0]);
BFE_ERROR_RETURN(BFE_SUCCESS == bfe_err, bfe_err); // Check for errors.
```

CAUTION: Be aware that **bfe_i2c_register_write()**, **bfe_i2c_register_read()**, and **bfe_i2c_block_read()** are *not* global functions. You can use them for custom code development. If you want to work directly with registers or commands, consider the R_RAA489220_RegisterRead() or R_RAA489220_RegisterWrite() API functions.

4. Demo Application

The sample code contains a demo application which 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, 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 retuned. The transitions are managed by a command line user's interface. You can send simple commands by inputting numbers from 1 to 5 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 (that is entering *Fault State* has higher priority overrides other states when multiple transition flags are set). For more information about the user interface and running the demo, refer to the *RAA489220 Sample Code Quick Start Guide*.

Be aware 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.

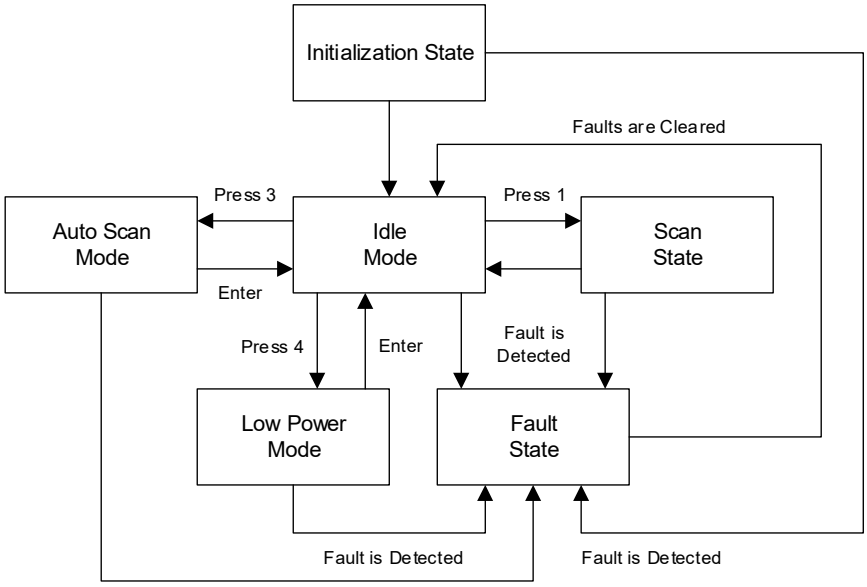

Figure 3. Sample Code State Machine Flow Diagram

After power on reset the first entered state is *Initialization State*. Figure 4 shows the flow. The MCU initializes the BFE interface (initialize I^2C, IRQs and CRC peripherals, reset the BFE, and read device ID). The reset command ensures that the current condition of the BFE is known in case only the MCU has been reset. Then, a memory check is accomplished to compare the register values with the expected values and verify that the registers are loaded correctly. After that a full BFE setup is run followed by communication test and full self-diagnostic that includes all tests in a single system scan. The containing code is executed once, followed by a transition to *Idle Mode*.

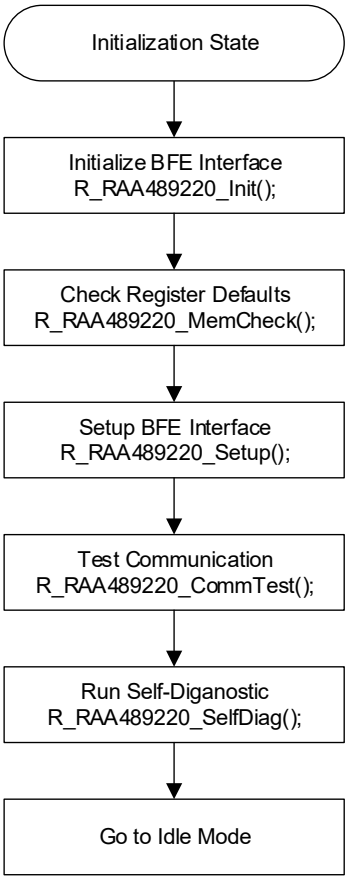

Figure 4. Initialization State Flowchart

In *Idle Mode* the demo application loops, waiting for user 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 50 loops the MCU runs a full self-test. In this mode, the charge and discharge FETs can be manually controlled. Their state is toggled after receiving a user command and remains unchanged if *Scan State* is entered.

Figure 6 shows the *Scan State* flow. All voltages, currents and temperatures are measured by calling the RAA489220_AllGet() API function, which is sending a sequence of measure and read commands followed by transition back to *Idle Mode*.

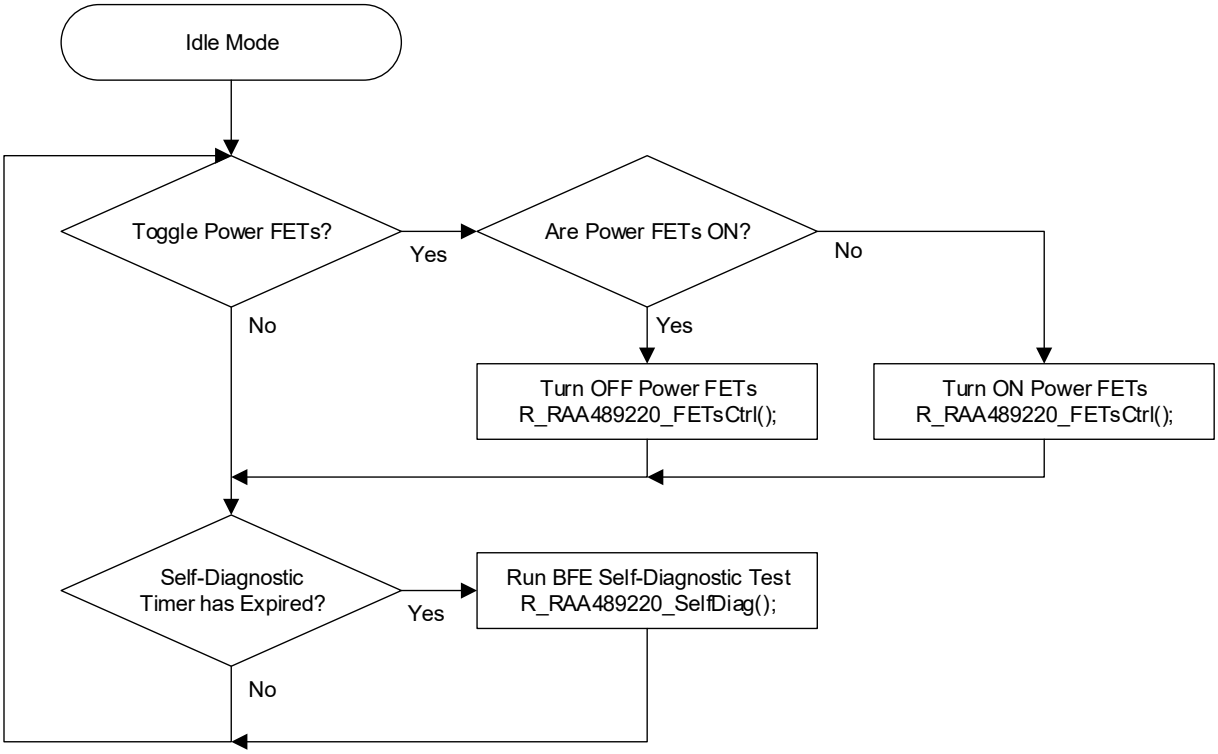

Figure 5. Idle Mode Flowchart

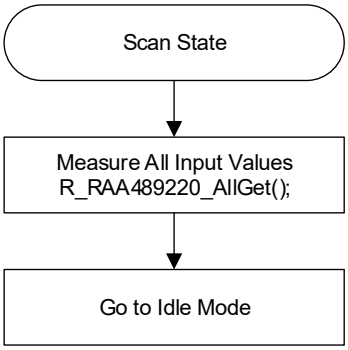

Figure 6. Scan State Flowchart

In Autoscan Mode, the RAA489220 functionality having the same name is enabled starting system scans and controlling the FETs by itself. Figure 7 shows the mode loop. There are two software counters used in this mode. Every 10 loops the MCU checks for any faults registered by the BFE; as well, every 10 loops, the measured voltages, currents, and temperatures are displayed in the user interface. On every loop, the MCU is waiting for the user command to disable auto scanning and return to *Idle Mode*.

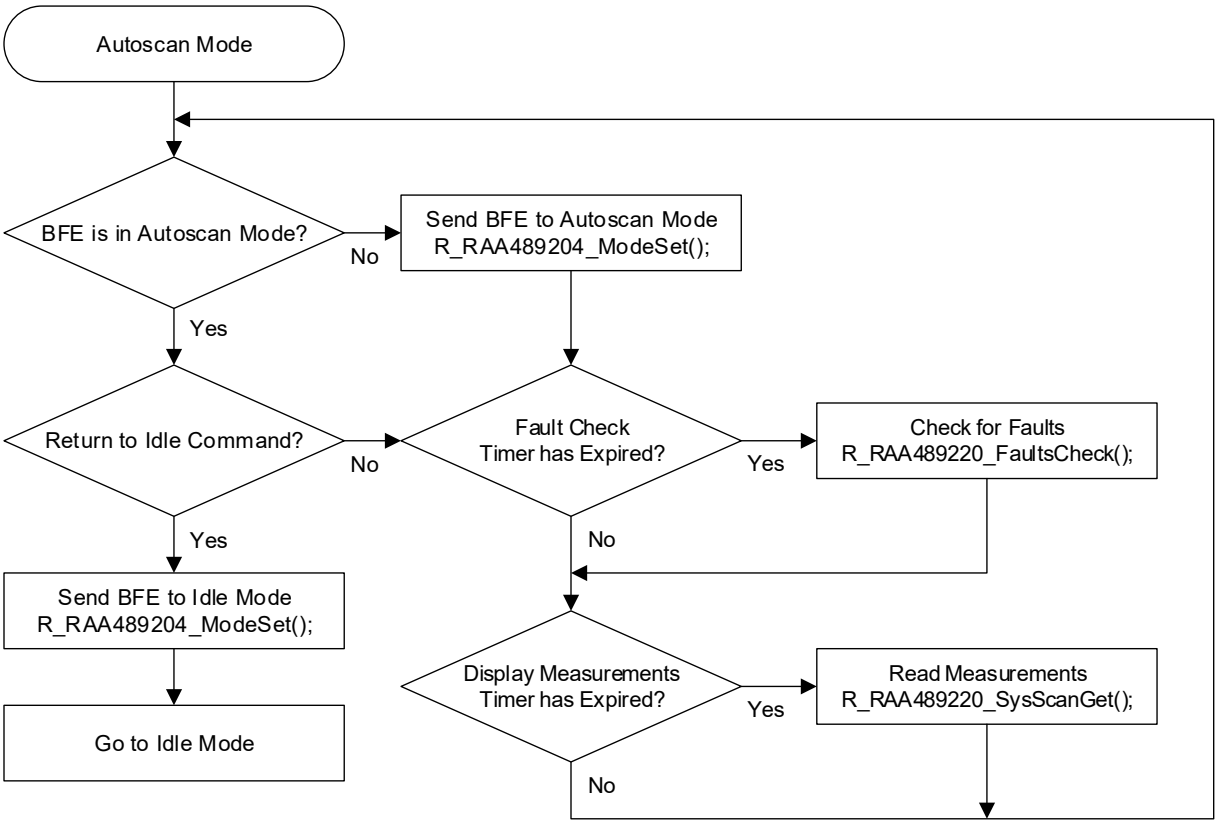

Figure 7. Autoscan Mode Flowchart

The demo application provides an option for entering sleep mode and waking up the BFE. 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). The *Autoscan* and *Low Power Modes* demonstrate the use of the API function R_RAA489220_ModeSet();.

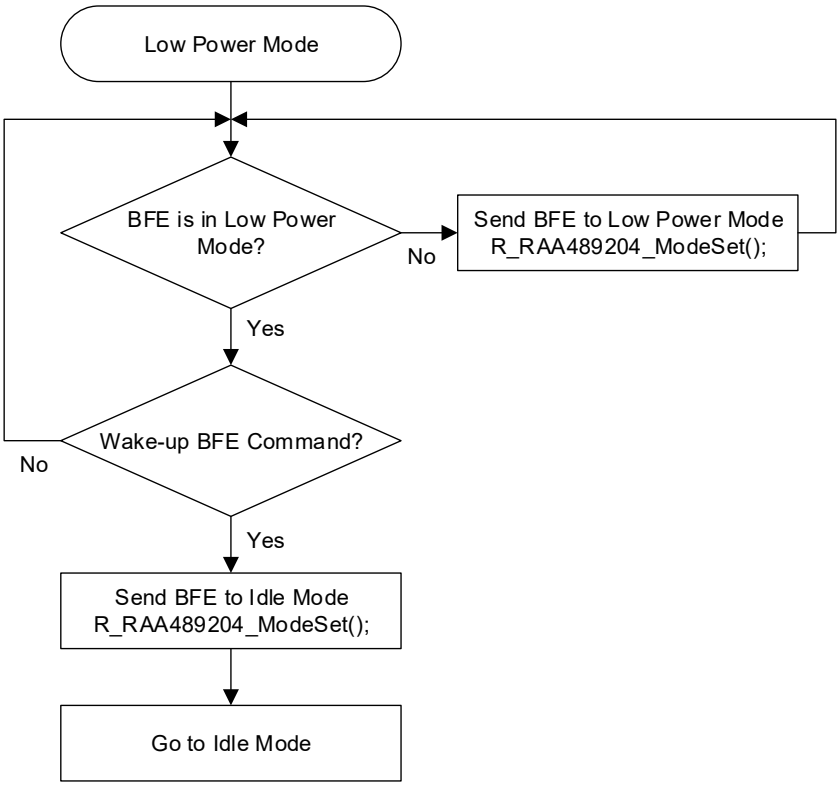

Figure 8. Low Power Mode Flowchart

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 the ALERT pin was asserted or the Fault 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 the BFEs, all Fault Status Register is read. The information is returned by the API function R_RAA489220_FaultsAllRead() into a data structure and visualized in the user interface. The BFE faults can be cleared in two ways: by setting the Clear All Faults flag in System Config 1 Register or by resetting the BFE. The fault management procedures can be found in the source code in file **r_bms.c**. If the fault is not able to be cleared (or the errors are not resolved), the state machine halts and waits for user interaction.

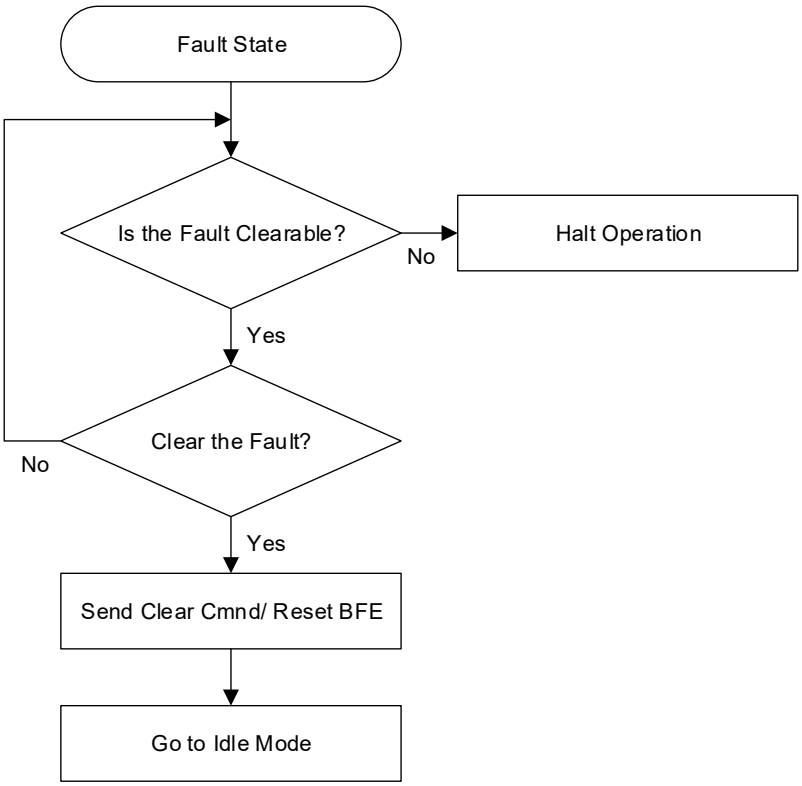

Figure 9. Fault State Flowchart

5. Revision History

Revision	Date	Description
1.00	Oct 17, 2022	Initial release.

IMPORTANT NOTICE AND DISCLAIMER

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

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

(Rev.1.0 Mar 2020)

Corporate Headquarters

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

Contact Information

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