

RAA489206 Battery Front End Sample Code

Introduction

This software manual provides a detailed description and application guidelines for using the RAA489206 sample code. It includes API functions and application examples to speed up the design of battery management systems up to 16S.

Contents

1.	Introduction	3
1.1	Assumptions and Advisory Notes	3
2.	RAA489206 Battery Front End Overview	3
2.1	Features	3
2.2	Applications	5
2.3	RAA489206 Sample Code Structure	5
3.	RAA489206 Application Programming Interface Implementation	7
3.1	Control and Configuration Structures	7
3.2	Register Bank	19
3.3	Command Bank	22
3.4	Private (Static) Functions	23
3.5	API Implementation	24
3.5.1	R_RAA489206_Init	24
3.5.2	R_RAA489206_Deinit	25
3.5.3	R_RAA489206_Setup	25
3.5.4	R_RAA489206_Reset	26
3.5.5	R_RAA489206_ModeSet	27
3.5.6	R_RAA489206_ModeRead	28
3.5.7	R_RAA489206_CommTest	28
3.5.8	R_RAA489206_SelfDiag	28
3.5.9	R_RAA489206_MemCheck	29
3.5.10	R_RAA489206_VPackGet	30
3.5.11	R_RAA489206_IPackGet	31
3.5.12	R_RAA489206_VCellsGet	32
3.5.13	R_RAA489206_Temps	33
3.5.14	R_RAA489206_AllGet	34
3.5.15	R_RAA489206_MixGet	36
3.5.16	R_RAA489206_FaultsAllRead	37
3.5.17	R_RAA489206_FaultsCheck	39
3.5.18	R_RAA489206_FaultsAllClear	39
3.5.19	R_RAA489206_CellBalanceCtrl	40
3.5.20	R_RAA489206_IsCellBalancing	40
3.5.21	R_RAA489206_ContScanCtrl	41
3.5.22	R_RAA489206_WatchdogCtrl	41
3.5.23	R_RAA489206_FETsCtrl	42
3.5.24	R_RAA489206_FETsRead	43
3.5.25	R_RAA489206_GPIOCtrl	44
3.5.26	R_RAA489206_RegisterRead	45
3.5.27	R_RAA489206_RegisterWrite	45

- 3.6 Configuration 46
 - 3.6.1 MCU Hardware Abstraction Layer 46
 - 3.6.2 Battery Front End 46
 - 3.6.3 Battery Abstraction Layer 48
- 3.7 Examples 49
- 4. Revision History 56

1. Introduction

The RAA489206 software library provides robust and easy access to the resources and functionality of the Battery Front End (BFE) device. It is a specialized-to-the-device control sample code, called a Battery Abstraction Layer (BAL). It is designed to be portable and suitable for integration into multitasking software projects that include higher level system control application. The software package includes the RAA489206 software library and examples.

The sample software package has the following features:

- Easy access to RAA489206 resources and advanced features
- Custom configurations
- Simplified status and error monitoring
- Full compatibility with Renesas Advanced (RA) Family 32-bit MCUs
- Application Programming Interface (API) for easy integration
- Examples of a use case for each API function supported by the library

1.1 Assumptions and Advisory Notes

- It is assumed that user possess basic understanding of microcontrollers, embedded systems hardware, battery management systems and Li-based battery cells.
- It is assumed that user has prior experience working with Integrated Development Environments (IDEs) such as e2studio and is familiar with the Flexible Software Package (FSP).
- 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 RAA489206 Datasheet, RAA489206 Evaluation Kit Manual, the RA4E1 Datasheet, and the RTKAPMODDONGLEV1Z Communications Dongle Hardware Manual to get acquainted with MCU and BFE features before proceeding further.

2. RAA489206 Battery Front End Overview

2.1 Features

RAA489206 is a 16-cell battery front end IC that have the following features:

- High hot plug rating: 62V
- Automatic protection functions
- Qualified for industrial temperature range: -40°C to +85°C
- Internal and external cell balancing
- Cell voltage measurement accuracy: $\pm 10\text{mV}$
- Pack current measurement accuracy: $\pm 0.2\%$
- 16-bit voltage and current measurements
- Monitors two external temperature inputs
- 4-pin GPIO port with LED driver
- Charge pump and high/low-side FET drivers
- Integrated 3.3V regulator
- Supports I²C, SPI, and SPI with CRC communications

The sample code from the software library runs in an external MCU that communicates with RAA489206 and controls the Battery Front End.

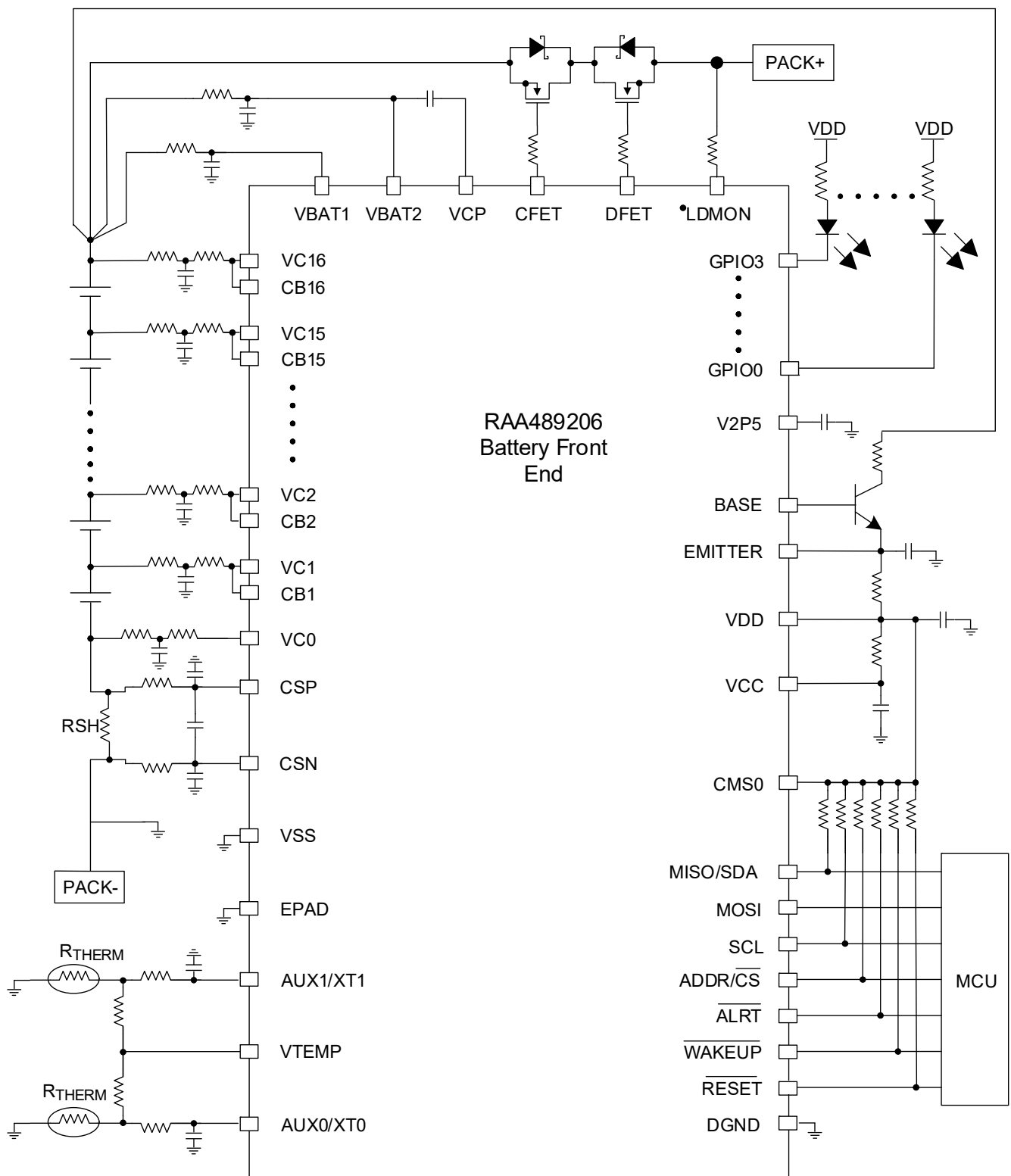


Figure 1. Typical Application of RAA489206

2.2 Applications

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

2.3 RAA489206 Sample Code Structure

The RAA489206 sample code contains the following software components:

- Battery Front End Application Programming Interface (BFE API)
- RAA489206 implementation of the BFE API
- Demo application with use cases of all supported API functions
- Configuration file for Renesas Flexible Software Package, which generates the peripheral (Hardware Abstraction Layer - HAL) drivers for the MCU used for running the sample code

The BFE instance and its structures define the contract and features common to most of the BFEs. The BFE instance for RAA489206 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.

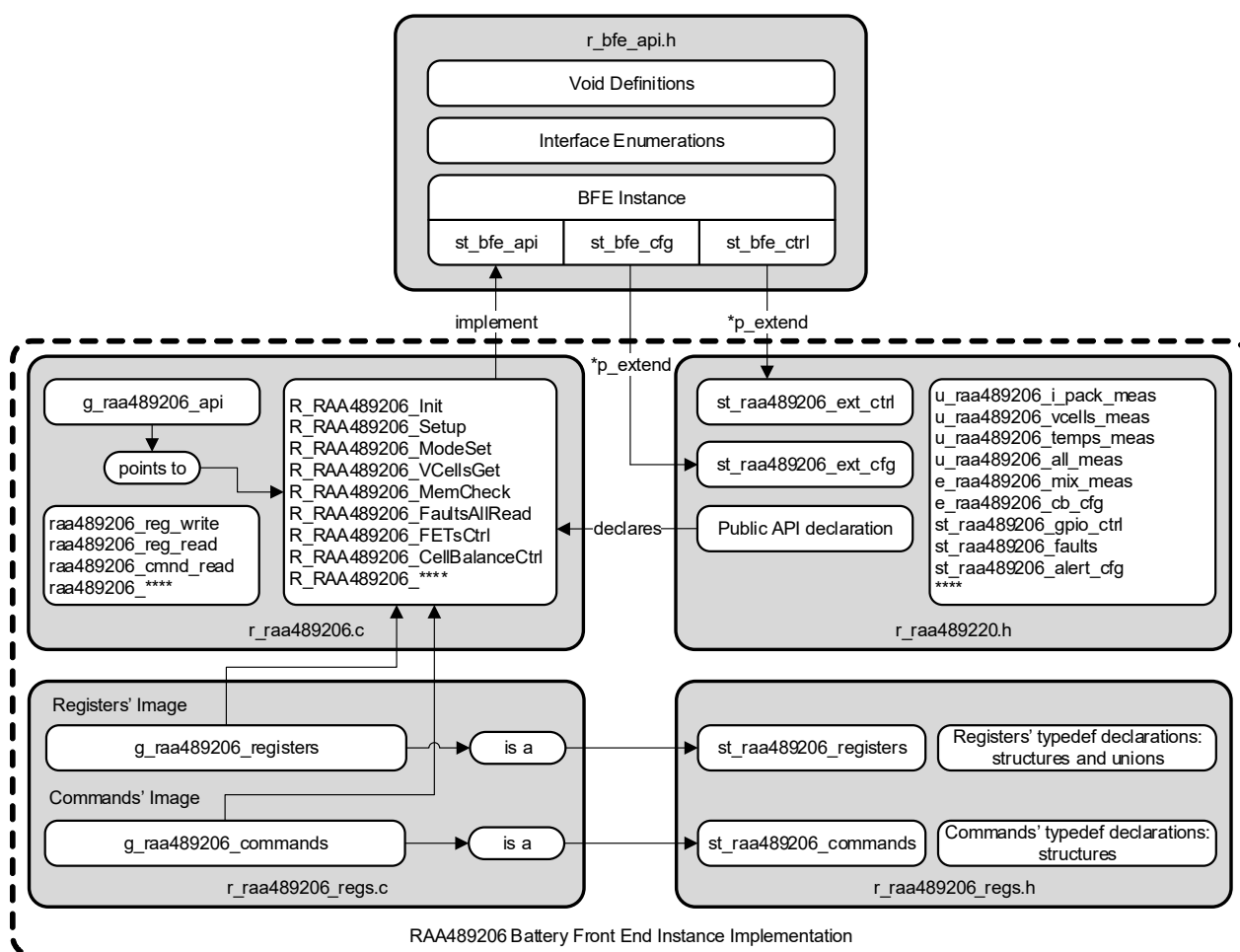


Figure 2. Main Software Components of BFE Instance Implementation

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

Table 1. Directory Structure of the RAA489206 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	Applications Layer
	---		board_cfg.h	Common macros related to the MCU board	
	apps		bal_data.c	BFE instance and definitions of the major structures	
			bal_data.c	BFE instance and definitions of the major structures	
			bal_data.h	Exported global variables of the interface	
			bms_common.h	Common macros related to the BMS application	
			bms.c	API use case examples	
			bms.h	Definitions and declarations related to the BMS application	
	r_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_raa489206_regs.c	Declarations of the BFE registers	
			r_raa489206_regs.h	Definitions, structures and enumerations of the BFE registers	
			r_raa489206.c	Actual code for the interface implementation	
			r_raa489206.h	Definitions, structures, enumerations and declarations of the API functions	
			r_typedefs.c	The uniform type names to solve portability problems with variable width of the data types	

3. RAA489206 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 `r_bfe/r_raa489206.h`.

Table 2 shows the content of the extended control structure. It contains information about the communication slave address, cell balancing cycle duration and flags indicating different internal states of the BFE. It also points to the bank with the commands.

Table 2. Members of the RAA489206 Extended Control Structure

typedef struct st_raa489206_ext_ctrl		
Member	Type	Description
slave_address	uint8_t	I ² C/SPI communication slave address of the Battery Front End
cb_cycle_duration_ms	uint32_t	The duration of one cell balancing cycle
load_present	uint8_t	A load has been detected.
charger_present	uint8_t	A charger has been detected.
battery_full	uint8_t	The battery is completely charged.
end_of_charge_voltage	uint8_t	At least one cell has exceeded the end of charge voltage threshold.
end_of_charge_current	uint8_t	The charge current has dropped below the end of charge current threshold.
pack_charge	uint8_t	The battery pack is being charged.
pack_discharge	uint8_t	The battery pack is being discharged.
need_cell_balancing	uint8_t	The voltage of one or more cells is higher than the CB Min Delta Threshold.
too_low_to_balance	uint8_t	All cell voltages are lower than CB Min Threshold.
too_high_to_balance	uint8_t	All cell voltages are higher than CB Max Threshold.
cb_cell_state	uint16_t	The Cell Balancing Cell State (Which cells are marked for balancing)
* p_bfe_cmnds	st_raa489206_cmnds_t	Pointer to the Battery Front End commands

Table 3 shows the content of the extended configuration structure. Its members correspond to all fixed settings of RAA489206. The extended configuration structure stores constants directly related to the hardware such as, shunt resistance, cell over/undervoltage threshold, short-circuit delay time, regulator settings, GPIO configuration. It also contains a cell balancing and an alert pin configuration structure. The thresholds are entered as real values (volts, amperes, seconds). Some of the variables have types which are defined as enumerations. Therefore, the user can select from a list of options facilitating the device configuration. The setup function is checking if the values of the thresholds in the extended configuration structure are within the correct ranges. If a mismatch is detected, the setup function returns an error code. For more information about the available options and their effect over the BFE performance, refer to the *RAA489206 Datasheet*.

Table 3. Members of the RAA489206 Extended Configuration Structure

typedef struct st_raa489206_ext_cfg		
Member	Type	Description
Shunt Resistors		
r_ipack_shunt_uohm	const uint32_t	Shunt resistance for charge and discharge currents [uohm]
r_ireg_shunt_mohm	const uint32_t	Shunt resistance for the external voltage regulator [mohm]
Fault Thresholds and Delays		
th_short_circuit_ma	const int32_t	Short-circuit threshold [mA] (range: ((-40.176mV to -321.33mV) / r_ipack_shunt_uohm) × 1000000)
delay_short_circuit_us	const uint16_t	Short-circuit delay [μs] (range: 250 to 4219μs)

Table 3. Members of the RAA489206 Extended Configuration Structure (Cont.)

typedef struct st_raa489206_ext_cfg		
Member	Type	Description
th_discharge_overcurrent_ma	const int32_t	Discharge overcurrent threshold [mA] (range: ((-0.005mV to -342.95mV) / r_ipack_shunt_uohm) × 1000000)
th_charge_overcurrent_ma	const int32_t	Charge overcurrent threshold [mA] (range: ((0.005mV to 342.95mV) / r_ipack_shunt_uohm) × 1000000)
th_cell_overnvlt_mv	const uint16_t	Cell overvoltage threshold [mV] (range: 1mV to 4.801mV)
th_cell_undervolt_mv	const uint16_t	Cell undervoltage threshold [mV] (range: 19mV to 4820mV)
th_cell_delta_overnvlt_mv	const uint16_t	Delta cell overvoltage threshold [mV] (range: 19mV to 4820mV)
th_pack_overnvlt_mv	const uint32_t	Pack overvoltage threshold [mV] (range: 300mV to 77120mV)
th_pack_undervolt_mv	const uint32_t	Pack undervoltage threshold [mV] (range: 300mV to 77120mV)
th_vcc_undervoltage_mv	const uint16_t	VCC undervoltage threshold [mV] (range: 1mV to 6414mV)
th_ireg_overcurrent_scan_idle_ma	const uint16_t	Regulator overcurrent threshold in SCAN or IDLE mode [mA] (range: ((1.34mV to 344.28mV) / r_ireg_shunt_mohm) × 1000)
th_ireg_overcurrent_lpm_ma	const uint16_t	Regulator overcurrent threshold in LOW POWER mode [mA] (range: ((1.34mV to 344.28mV) / r_ireg_shunt_mohm) × 1000)
i_dir_detection_delay	const uint8_t	Current direction detection delay (false: single system scan delay; true: three system scans delay)
th_charge_over_temp_0_mv	const uint16_t	Over-temperature threshold when charging for external input 0 [mV] (range: 6.263mV to 1606.67mV, step: 6.263mV)
th_charge_under_temp_0_mv	const uint16_t	Under-temperature threshold when charging for external input 0 [mV] (range: 6mV to 1607mV)
th_discharge_over_temp_0_mv	const uint16_t	Over-temperature threshold when discharging for external input 0 [mV] (range: 6mV to 1607mV)
th_discharge_under_temp_0_mv	const uint16_t	Under-temperature threshold when discharging for external input 0 [mV] (range: 6mV to 1607mV)
th_charge_over_temp_1_mv	const uint16_t	Over-temperature threshold when charging for external input 1 [mV] (range: 6mV to 1607mV)
th_charge_under_temp_1_mv	const uint16_t	Under-temperature threshold when charging for external input 1 [mV] (range: 6mV to 1607mV)
th_discharge_over_temp_1_mv	const uint16_t	Over-temperature threshold when discharging for external input 1 [mV] (range: 6mV to 1607mV)
th_discharge_under_temp_1_mv	const uint16_t	Under-temperature threshold when discharging for external input 1 [mV] (range: 6mV to 1607mV)
th_int_temp_warning_degC	const uint16_t	Internal temperature warning (range: 0...950 × 0.1°C)
FET Control and Fault Masks		
end_of_charge	const e_raa489206_eoc_cfg_t	The criteria for end of charge detection
th_voltage_end_of_charge_mv	const uint16_t	End of Charge Voltage Threshold [mV] (range: 19mV to 4820mV)
th_current_end_of_charge_ma	const uint16_t	End of Charge Current Threshold [mA] (range: ((0.005mV to 342.95mV) / r_ipack_shunt_uohm) × 1000000)
turn_off_fets_doc_fault	const uint8_t	(true: Turn OFF; false: Don't turn OFF) the power FETs after detection of Discharge overcurrent event
turn_off_fets_coc_fault	const uint8_t	(true: Turn OFF; false: Don't turn OFF) the power FETs after detection of Charge overcurrent event
turn_off_fets_cells_fault	const uint8_t	(true: Turn OFF; false: Don't turn OFF) the power FETs after detection of cell OV, UV or delta max fault event

Table 3. Members of the RAA489206 Extended Configuration Structure (Cont.)

typedef struct st_raa489206_ext_cfg		
Member	Type	Description
turn_off_fets_pack_fault	const uint8_t	(true: Turn OFF; false: Don't turn OFF) the power FETs after detection of pack OV or UV
turn_off_fets_etaux_fault	const uint8_t	(true: Turn OFF; false: Don't turn OFF) the power FETs after detection of an under-temperature or over-temperature fault event on xT0 or xT1
Low Voltage Offset Recalibration		
recalibrate_lv_offset_scan_idle_mode	const uint8_t	Initiate a low voltage offset recalibration of Vbat1, Vcc, Vtemp, IT, xT0, and xT1 on each scan in IDLE or SCAN mode
recalibrate_lv_offset_lp_mode	const uint8_t	Initiate a low voltage offset recalibration of Vbat1, Vcc, Vtemp, IT, xT0, and xT1 for each measurement in LOW POWER mode
Averaging of Measurements		
averaging_v_pack	const e_raa489206_other_avg_t	Pack voltage, Vcc, Ireg, Vtemp and Internal temp averaging
averaging_v_cell	const e_raa489206_vcell_avg_t	Cell voltage averaging
averaging_i_pack	const e_raa489206_ipack_avg_t	Pack current averaging
averaging_ext_temp	const e_raa489206_etaux_avg_t	External temperature inputs voltage averaging
Cell Balancing Configuration		
cb_cfg	const st_raa489206_cb_cfg_t	Cell balancing configuration structure
Cell Balancing Configuration		
scan_cont_cfg	const st_raa489206_cont_scan_cfg_t	Continuous scan configuration structure
Other		
weak_regulator	const uint8_t	(true: Weak; false: Strong) regulator in LOW POWER and SHIP mode
gpio_configuration	const e_raa489206_gpio_cfg_t	Select GPIO port direction and function
load_detection	const e_raa489206_load_detection_t	Enable or disable the load detection circuit
load_detection_lp	const uint8_t	(true: Enable; false: disable) load detection in Low Power mode
load_detect_delay	const e_raa489206_load_detect_delay_t	Time delay before testing for a load
load_recovery	const uint8_t	(False: Don't recover; true: Recover) from a short-circuit event
comm_spi_crc_enable	const uint8_t	(true: Use; false: Don't use) CRC16 control with the SPI communication
comm_i2c_address_select	const e_raa489206_i2c_addr_t	Select the I ² C communication address
Alert Pin		
alert_pin_cfg	const st_raa489206_alert_cfg_t	Configuration when to assert the ALERT pin
alert_pin_pulse_enable	const uint8_t	(False: drive alert pin constantly; true: pulse alert pin 2ms on time, 10ms period)
Pin Configuration		
pin_alert	const bsp_io_port_pin_t	ALERT pin
pin_reset	const bsp_io_port_pin_t	RESET pin
pin_wake_up	const bsp_io_port_pin_t	Wake Up pin

Table 4 shows the options for the *end_of_charge* constant from the extended configuration, which are listed in the RAA489206 End of Charge Configuration Enumeration. The selected constant is used for setting the *p_ext_ctrl->battery_full* flag, CFET control, and the automatic cell balancing algorithm.

Table 4. RAA489206 End of Charge Configuration Enumeration

typedef enum e_raa489206_eoc_cfg		
Constant	Value	Description
BFE_EOC_VOLTAGE	0x00	Detect end of charging based on voltage threshold
BFE_EOC_CURRENT	0x01	Detect end of charging based on current threshold

Table 5 shows the options for the *averaging_v_pack* constant from the extended configuration, which are listed in the RAA489206 Other Average Samples Enumeration. The selected constant sets the number of pack voltage, Vcc, Ireg, Vtemp, and internal temperature readings averaged per measurement before storing the results to the corresponding register.

Table 5. RAA489206 Other (Vbat) Average Samples Enumeration

typedef enum e_raa489206_other_avg		
Constant	Value	Description
OTHER_AVG_1_SMPL	0x00	Average 1 sample
OTHER_AVG_2_SMPL	0x01	Average 2 samples
OTHER_AVG_4_SMPL	0x02	Average 4 samples
OTHER_AVG_8_SMPL	0x03	Average 8 samples
OTHER_AVG_16_SMPL	0x04	Average 16 samples
OTHER_AVG_32_SMPL	0x05	Average 32 samples
OTHER_AVG_64_SMPL	0x06	Average 64 samples
OTHER_AVG_128_SMPL	0x07	Average 128 samples

Table 6 shows the options for the *averaging_v_cell* constant from the extended configuration, which are listed in the RAA489206 Vcell Average Samples Enumeration. The selected constant sets the number of cell voltage readings averaged per measurement before storing the results to the corresponding register.

Table 6. RAA489206 Vcell Average Samples Enumeration

typedef enum e_raa489206_vcell_avg		
Constant	Value	Description
VCELL_AVG_1_SMPL	0x00	Average 1 sample
VCELL_AVG_2_SMPL	0x01	Average 2 samples
VCELL_AVG_4_SMPL	0x02	Average 4 samples
VCELL_AVG_8_SMPL	0x03	Average 8 samples
VCELL_AVG_16_SMPL	0x04	Average 16 samples
VCELL_AVG_32_SMPL	0x05	Average 32 samples
VCELL_AVG_64_SMPL	0x06	Average 64 samples
VCELL_AVG_128_SMPL	0x07	Average 128 samples

Table 7 shows the options for the *averaging_i_pack* constant from the extended configuration, which are listed in the RAA489206 Ipack Average Samples Enumeration. The selected constant sets the number of pack current readings averaged per measurement before storing the results to the corresponding register.

Table 7. RAA489206 Ipack Average Samples Enumeration

typedef enum e_raa489206_ipack_avg		
Constant	Value	Description
IPACK_AVG_1_SMPL	0x00	Average 1 sample
IPACK_AVG_2_SMPL	0x01	Average 2 samples
IPACK_AVG_4_SMPL	0x02	Average 4 samples
IPACK_AVG_8_SMPL	0x03	Average 8 samples
IPACK_AVG_16_SMPL	0x04	Average 16 samples
IPACK_AVG_32_SMPL	0x05	Average 32 samples
IPACK_AVG_64_SMPL	0x06	Average 64 samples
IPACK_AVG_128_SMPL	0x07	Average 128 samples

Table 8 shows the options for the *averaging_ext_temp* constant from the extended configuration, which are listed in the RAA489206 ETAUX Average Samples Enumeration. The selected constant sets the number of external temperature readings averaged per measurement before storing the results to the corresponding register.

Table 8. RAA489206 ETAUX Average Samples Enumeration

typedef enum e_raa489206_etaux_avg		
Constant	Value	Description
ETAUX_AVG_1_SMPL	0x00	Average 1 sample
ETAUX_AVG_2_SMPL	0x01	Average 2 samples
ETAUX_AVG_4_SMPL	0x02	Average 4 samples
ETAUX_AVG_8_SMPL	0x03	Average 8 samples
ETAUX_AVG_16_SMPL	0x04	Average 16 samples
ETAUX_AVG_32_SMPL	0x05	Average 32 samples
ETAUX_AVG_64_SMPL	0x06	Average 64 samples
ETAUX_AVG_128_SMPL	0x07	Average 128 samples

Table 9 shows the options for the *gpio_configuration* constant from the extended configuration, which are listed in the RAA489206 GPIO Configuration Enumeration. The selected constant sets the GPIO port direction and function.

Table 9. RAA489206 GPIO Configuration Enumeration

typedef enum e_raa489206_gpio_cfg		
Constant	Value	Description
GPIO_INPUTS	0x00	Digital Inputs
GPIO_OUTPUTS	0x01	Digital Outputs
GPIO_LED_DRIVE	0x02	Drive LED
GPIO_POWER_FET_GATE_DRV	0x03	Power FETs Gate Drive Out

Table 10 shows the options for the *load_detection* constant from the extended configuration, which are listed in the RAA489206 Load Detection Enumeration. The selected constant enables the load detection circuit and selects the pull up resistance.

Table 10. RAA489206 Load Detection Enumeration

typedef enum e_raa489206_load_detection		
Constant	Value	Description
LOAD_DETECTION_DISABLED	0x00	Don't Enable Load Detection
LOAD_DETECTION_ENABLED_1MOHM	0x01	Load Detection Enabled with 1MΩ Pull Up to VBAT2
LOAD_DETECTION_ENABLED_10KOHM	0x02	Load Detection Enabled with 10kΩ Pull Up to VBAT2

Table 11 shows the options for the *load_detect_delay* constant from the extended configuration, which are listed in the RAA489206 Load Detect Time Delay Enumeration. The selected constant configures the load detection circuit time delay.

Table 11. RAA489206 Load Detect Time Delay Enumeration

typedef enum e_raa489206_load_detect_delay		
Constant	Value	Description
LOAD_DET_DELAY_250_MS	0x00	Load Detect Time Delay 250ms
LOAD_DET_DELAY_1000_MS	0x01	Load Detect Time Delay 1000ms
LOAD_DET_DELAY_2000_MS	0x02	Load Detect Time Delay 2000ms
LOAD_DET_DELAY_4000_MS	0x03	Load Detect Time Delay 4000ms

Table 12 shows the options for the *comm_i2c_addr_select* constant from the extended configuration, which are listed in the RAA489206 I²C Communication Address Enumeration. The selected constant sets the slave address when the MCU uses an I²C interface to communicate with the BFE.

Table 12. RAA489206 I²C Communication Address Enumeration

typedef enum e_raa489206_i2c_addr		
Constant	Value	Description
RAA489206_I2C_ADDRESS_0x14	0x14	I ² C address option 1
RAA489206_I2C_ADDRESS_0x34	0x34	I ² C address option 2

Table 13 shows the type definition for the *cb_cfg* member of the external configuration structure. The RAA489206 Cell Balancing Configuration Structure contains all constants related to the cell balancing function of RAA489206.

Table 13. Members of the RAA489206 Cell Balancing Configuration Structure

typedef struct st_raa489206_cb_cfg		
Member	Type	Description
mode	const e_raa489206_cb_mode_t	BFE cell balancing mode
cb_fets	const e_raa489206_cb_fets_t	Select internal or external FETs for cell balancing
cb_mask_enable	const uint8_t	(true: Don't allow; false: Allow) cell balancing of adjacent cells
cb_single_sys_scan_enable	const uint8_t	(true: Allow; false: Don't allow) auto cell balancing during single system scans
cb_timer_unit	const e_raa489206_cb_tmr_unit_t	Cell balancing ON and OFF timer unit
cb_on_timeout	const uint16_t	Cell balancing ON time (0s to 1016s, step: 8s) or (0ms to 1016ms, step: 8ms)
cb_off_timeout	const uint16_t	Cell balancing OFF time (0s to 1016s, step: 8s) or (0ms to 1016ms, step: 8ms)

Table 13. Members of the RAA489206 Cell Balancing Configuration Structure (Cont.)

typedef struct st_raa489206_cb_cfg		
Member	Type	Description
cb_auto_th_cell_delta_min_mv	const uint16_t	The minimum cell voltage difference that determines a cell needs balancing in Auto mode [mV] (range: 4.67mV to 1204.97mV, step: 4.707mV)
cb_auto_th_enable_volt_min_mv	const uint16_t	The minimum cell voltage when cell balancing is allowed in Auto mode [mV] (range: 0.037mV to 4801.25mV, step: 18.828mV)
cb_auto_th_enable_volt_max_mv	const uint16_t	The maximum cell voltage when cell balancing is allowed in Auto mode [mV] (range: 18.80mV to 4820mV, step: 18.828mV)
cb_charging_enable	const uint8_t	(true: Allow; false: Don't allow) cell balancing during charging
cb_end_of_charge	const uint8_t	(true: Allow; false: Don't allow) cell balancing after end of charging

Table 14 shows the options for the *mode* constant from the cell balancing configuration, which are listed in the RAA489206 Cell Balancing Mode Enumeration. The selected constant sets the cell balancing mode during system operation.

Table 14. RAA489206 Cell Balancing Mode Enumeration

typedef enum e_raa489206_cb_mode		
Constant	Value	Description
BFE_BALANCE_MODE_OFF	0x00	Cell balancing is disabled
BFE_BALANCE_MODE_MANUAL	0x01	Manual cell balancing mode
BFE_BALANCE_MODE_AUTO	0x02	Auto cell balancing mode

Table 15 shows the options for the *cb_fets* constant from the cell balancing configuration, which are listed in the RAA489206 Cell Balancing FETs Enumeration. The selected constant sets which cell balancing FETs is used.

Table 15. RAA489206 Cell Balancing FETs Enumeration

typedef enum e_raa489206_cb_fets		
Constant	Value	Description
BFE_BALANCE_INTERNAL	0x00	Use internal FETs for cell balancing
BFE_BALANCE_EXTERNAL	0x01	Use external FETs for cell balancing

Table 16 shows the options for the *cb_timer_unit* constant from the cell balancing configuration, which are listed in the RAA489206 Cell Balancing Timer Unit Enumeration. The selected constant sets the unit of the timer.

Table 16. RAA489206 Cell Balancing Timer Unit Enumeration

typedef enum e_raa489206_cb_tmr_unit		
Constant	Value	Description
BFE_BALANCE_TIMER_MS	0x00	Cell balancing ON and OFF Timer unit is milliseconds
BFE_BALANCE_TIMER_S	0x01	Cell balancing ON and OFF Timer unit is seconds

Table 17 shows the content of the continuous system scan configuration structure. It contains members that set all relevant scan, update and fault delays, and select what to be included in the scan.

Table 17. Members of the RAA489206 Continuous System Scan Configuration Structure

typedef struct st_raa489206_cont_scan_cfg		
Member	Type	Description
cont_scan_delay	const e_raa489206_scan_delay_t	Continuous System Scan interval
zero_current_timeout	const e_raa489206_lpt_delay_t	Zero current timeout before going to LOW POWER mode
cont_scan_ow_update	const e_raa489206_ow_update_t	How often the open-wire test is executed
cont_scan_other_update	const e_raa489206_update_other_t	How often ETAUX, Vbat1, Vcc, Ireg, Vtemp and the Internal Temperature are measured
cont_scanflt_delay_vcell	const e_raa489206flt_delay_vcell_t	Vcell fault delay
cont_scanflt_delay_delta_cell	const e_raa489206flt_delay_dvcell_t	Delta cell overvoltage fault delay
cont_scanflt_delay_other	const e_raa489206flt_delay_other_t	Vpack, Vcc, Ireg and Int temp fault delay
cont_scanflt_delay_ext_temp	const e_raa489206flt_delay_etaux_t	External temperature fault delay
cont_scanflt_delay_doc	const e_raa489206oc_delay_t	Discharge overcurrent fault delay
cont_scanflt_delay_coc	const e_raa489206oc_delay_t	Charge overcurrent fault delay
cont_scan_charge_undervolt_enable	const uin8_t	Allow charging when one or more cells are below the UV th
cont_scan_discharge_overnvoltage_enable	const uin8_t	Allow discharging when one or more cells are above the OV th
cont_scan_vcell_include	const uin8_t	Enable Vcell measurement within the cont system scan.
cont_scan_ipack_include	const uin8_t	Enable Ipack measurement within the cont system scan.
cont_scan_etaux_include	const uin8_t	Enable ETAUX measurement within the cont system scan.
cont_scan_itep_include	const uin8_t	Enable int temp measurement within the cont system scan.
cont_scan_vbat1_include	const uin8_t	Enable Vbat1 measurement within the cont system scan.
cont_scan_ow_include	const uin8_t	Enable open-wire test within the cont system scan.

Table 18 shows the options for the *cont_scan_delay* constant from the continuous system scan configuration which are listed in the RAA489206 Scan Delay Enumeration. The selected constant sets the scan interval.

Table 18. RAA489206 Scan Delay Enumeration

typedef enum e_raa489206_scan_delay		
Constant	Value	Description
SCAN_DELAY_0_MS	0x00	Scan Delay 0ms
SCAN_DELAY_64_MS	0x01	Scan Delay 64ms
SCAN_DELAY_128_MS	0x02	Scan Delay 128ms
SCAN_DELAY_256_MS	0x03	Scan Delay 256ms
SCAN_DELAY_512_MS	0x04	Scan Delay 512ms
SCAN_DELAY_1024_MS	0x05	Scan Delay 1024ms
SCAN_DELAY_2048_MS	0x06	Scan Delay 2048ms
SCAN_DELAY_4096_MS	0x07	Scan Delay 4096ms

Table 19 shows the options for the *zero_current_timeout* constant from the continuous system scan configuration, which are listed in the RAA489206 Low Power Timer Enumeration. The selected constant sets the scan interval in low power mode.

Table 19. RAA489206 Low Power Timer Enumeration

typedef enum e_raa489206_lpt_delay		
Constant	Value	Description
LPT_OFF	0x00	Low Power Timer is OFF
LPT_SCANS_512	0x01	Low Power Timer is 512 System Scans
LPT_SCANS_1024	0x02	Low Power Timer is 1024 System Scans
LPT_SCANS_2048	0x03	Low Power Timer is 2048 System Scans
LPT_SCANS_4096	0x04	Low Power Timer is 4096 System Scans
LPT_SCANS_8192	0x05	Low Power Timer is 8192 System Scans
LPT_SCANS_16384	0x06	Low Power Timer is 16384 System Scans
LPT_SCANS_32768	0x07	Low Power Timer is 32768 System Scans

Table 20 shows the options for the *cont_scan_ow_update* constant from the continuous system scan configuration, which are listed in the RAA489206 Open Wire Update Enumeration. The selected constant sets the open wire test interval.

Table 20. RAA489206 Open Wire Update Enumeration

typedef enum e_raa489206_ow_update		
Constant	Value	Description
OW_UPDATE_EACH_256	0x00	OW test on each 256 scan
OW_UPDATE_EACH_512	0x01	OW test on each 512 scan
OW_UPDATE_EACH_1024	0x02	OW test on each 1024 scan
OW_UPDATE_EACH_2048	0x03	OW test on each 2048 scan

Table 21 shows the options for the *cont_scan_other_update* constant from the continuous system scan configuration, which are listed in the RAA489206 Update Other Enumeration. The selected constant sets the pack voltage, Vcc, Ireg, Vtemp and internal temperature measurement interval.

Table 21. RAA489206 Update Other Enumeration

typedef enum e_raa489206_update_other		
Constant	Value	Description
UPDATE_OTHER_EVERY_SCAN	0x00	Update on every system scan
UPDATE_OTHER_8_SCAN	0x01	Update on every 8th system scan
UPDATE_OTHER_16_SCAN	0x02	Update on every 16th system scan
UPDATE_OTHER_64_SCAN	0x03	Update on every 64th system scan

Table 22 shows the options for the *cont_scan_fit_delay_vcell* constant from the continuous system scan configuration, which are listed in the RAA489206 Vcell Fault Delay Enumeration. The selected constant sets the fault delay when a fault during Vcell measurement is detected.

Table 22. RAA489206 Vcell Fault Delay Enumeration

typedef enum e_raa489206_fit_delay_vcell		
Constant	Value	Description
VCELL_FLT_DELAY_1_SCAN	0x00	1 scan delay
VCELL_FLT_DELAY_2_SCANS	0x01	2 scans delay
VCELL_FLT_DELAY_3_SCANS	0x02	3 scans delay
VCELL_FLT_DELAY_4_SCANS	0x03	4 scans delay
VCELL_FLT_DELAY_5_SCANS	0x04	5 scans delay
VCELL_FLT_DELAY_6_SCANS	0x05	6 scans delay
VCELL_FLT_DELAY_7_SCANS	0x06	7 scans delay
VCELL_FLT_DELAY_8_SCANS	0x07	8 scans delay
VCELL_FLT_DELAY_9_SCANS	0x08	9 scans delay
VCELL_FLT_DELAY_10_SCANS	0x09	10 scans delay
VCELL_FLT_DELAY_11_SCANS	0x0A	11 scans delay
VCELL_FLT_DELAY_12_SCANS	0x0B	12 scans delay
VCELL_FLT_DELAY_13_SCANS	0x0C	13 scans delay
VCELL_FLT_DELAY_14_SCANS	0x0D	14 scans delay
VCELL_FLT_DELAY_15_SCANS	0x0E	15 scans delay
VCELL_FLT_DELAY_16_SCANS	0x0F	16 scans delay

Table 23 shows the options for the *cont_scan_fit_delay_delta_cell* constant from the continuous system scan configuration which are listed in the RAA489206 Delta Vcell Fault Delay Enumeration. The selected constant sets the fault delay when a fault during Delta Vcell measurement is detected.

Table 23. RAA489206 Delta Vcell Fault Delay Enumeration

typedef enum e_raa489206_fit_delay_dvcell		
Constant	Value	Description
DVCELL_FLT_DELAY_1_SCAN	0x00	1 scan delay
DVCELL_FLT_DELAY_3_SCANS	0x01	3 scans delay

Table 24 shows the options for the *cont_scan_fit_delay_other* constant from the continuous system scan configuration which are listed in the RAA489206 Other Fault Delay Enumeration. The selected constant sets the fault delay when a fault during voltage, Vcc, Ireg, Vtemp or internal temperature measurement is detected.

Table 24. RAA489206 Other Fault Delay Enumeration

typedef enum e_raa489206_fit_delay_other		
Constant	Value	Description
OTHER_FLT_DELAY_1_SCAN	0x00	1 scan delay
OTHER_FLT_DELAY_3_SCANS	0x01	3 scans delay

Table 25 shows the options for the *cont_scanflt_delay_ext_temp* constant from the continuous system scan configuration which are listed in the RAA489206 ETAUX Fault Delay Enumeration. The selected constant sets the fault delay when a fault during ETAUX measurement is detected.

Table 25. RAA489206 ETAUX Fault Delay Enumeration

typedef enum e_raa489206flt_delay_etaux		
Constant	Value	Description
ETAUX_FLT_DELAY_1_SCAN	0x00	1 scan delay
ETAUX_FLT_DELAY_3_SCANS	0x01	3 scans delay

Table 26 shows the options for the *cont_scanflt_delay_doc* and *cont_scanflt_delay_coc* constants from the continuous system scan configuration which are listed in the RAA489206 Charge and Discharge Overcurrent Delay Enumeration. The selected constant sets the fault delay when a fault during current measurement is detected.

Table 26. RAA489206 Charge and Discharge Overcurrent Delay Enumeration

typedef enum e_raa489206oc_delay		
Constant	Value	Description
OC_DELAY_0_SCANS	0x00	no scan delay
OC_DELAY_1_SCAN	0x01	1 scan delay
OC_DELAY_2_SCANS	0x02	2 scans delay
OC_DELAY_3_SCANS	0x03	3 scans delay
OC_DELAY_4_SCANS	0x04	4 scans delay
OC_DELAY_5_SCANS	0x05	5 scans delay
OC_DELAY_6_SCANS	0x06	6 scans delay
OC_DELAY_7_SCANS	0x07	7 scans delay
OC_DELAY_8_SCANS	0x08	8 scans delay
OC_DELAY_9_SCANS	0x09	9 scans delay
OC_DELAY_10_SCANS	0x0A	10 scans delay
OC_DELAY_11_SCANS	0x0B	11 scans delay
OC_DELAY_12_SCANS	0x0C	12 scans delay
OC_DELAY_13_SCANS	0x0D	13 scans delay
OC_DELAY_14_SCANS	0x0E	14 scans delay
OC_DELAY_15_SCANS	0x0F	15 scans delay

Table 27 shows the type definition for the *alert_pin_cfg* member of the external configuration structure. The RAA489206 Alert Pin Configuration Structure contains all constants related to the behavior of the ALERT pin corresponding to fault events and others.

Table 27. Members of the RAA489206 Alert Pin Configuration Structure

typedef struct st_raa489206_alert_cfg		
Member	Type	Description
alert_vcc_undervoltage	const uint8_t	(true: Assert; False: Don't assert) the alert pin when an undervoltage event at the VCC pin is detected.
alert_open_wire	const uint8_t	(true: Assert; False: Don't assert) the alert pin when an open-wire is detected.
alert_intenal_over_temp	const uint8_t	(true: Assert; False: Don't assert) the alert pin when internal over-temperature is detected.
alert_charge_overcurent	const uint8_t	(true: Assert; False: Don't assert) the alert pin when charge overcurrent is detected.

Table 27. Members of the RAA489206 Alert Pin Configuration Structure (Cont.)

typedef struct st_raa489206_alert_cfg		
Member	Type	Description
alert_discharge_overcurrent	const uint8_t	(true: Assert; false: Don't assert) the alert pin when discharge overcurrent is detected.
alert_discharge_short_circuit	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a discharge short-circuit is detected.
alert_cell_undervoltage	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a cell undervoltage is detected.
alert_cell_overvoltage	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a cell overvoltage is detected.
alert_xt0_discharge_under_temp	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a discharge under-temperature on xT0 pin is detected.
alert_xt0_discharge_over_temp	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a discharge over-temperature on xT0 pin is detected.
alert_xt0_charge_under_temp	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a charge under-temperature on xT0 pin is detected.
alert_xt0_charge_over_temp	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a charge over-temperature on xT0 pin is detected.
alert_xt1_discharge_under_temp	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a discharge under-temperature on xT1 pin is detected.
alert_xt1_discharge_over_temp	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a discharge over-temperature on xT1 pin is detected.
alert_xt1_charge_under_temp	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a charge under-temperature on xT1 pin is detected.
alert_xt1_charge_over_temp	const uint8_t	(true: Assert; false: Don't assert) the alert pin when a charge over-temperature on xT1 pin is detected.
alert_pack_undervoltage	const uint8_t	(true: Assert; false: Don't assert) the alert pin when pack undervoltage is detected.
alert_pack_overvoltage	const uint8_t	(true: Assert; false: Don't assert) the alert pin when pack overvoltage is detected.
alert_charge_pump_not_ready	const uint8_t	(true: Assert; false: Don't assert) the alert pin when charge pump is not ready for use.
alert_busy	const uint8_t	(true: Assert; false: Don't assert) the alert pin when the Busy bit is set.
alert_batt_full	const uint8_t	(true: Assert; false: Don't assert) the alert pin when full battery is detected.
alert_internal_over_temp_warning	const uint8_t	(true: Assert; false: Don't assert) the alert pin when Internal Over-Temperature Warning is detected.
alert_end_of_charge_current	const uint8_t	(true: Assert; false: Don't assert) the alert pin when the pack current drops below IEOC.
alert_end_of_charge_voltage	const uint8_t	(true: Assert; false: Don't assert) the alert pin when the pack voltage drops below VEOC.
alert_delta_cell_voltage_fault	const uint8_t	(true: Assert; false: Don't assert) the alert pin when delta cell voltage fault is detected.
alert_cb_cell_voltage_high	const uint8_t	(true: Assert; false: Don't assert) the alert pin when cell voltage is too high to cell balance.
alert_cb_cell_voltage_low	const uint8_t	(true: Assert; false: Don't assert) the alert pin when cell voltage is too low to cell balance.
alert_need_cb	const uint8_t	(true: Assert; false: Don't assert) the alert pin when cell balancing is needed.
alert_discharge	const uint8_t	(true: Assert; false: Don't assert) the alert pin when discharge current is detected.
alert_charge	const uint8_t	(true: Assert; false: Don't assert) the alert pin when charge current is detected.
alert_charger_detect	const uint8_t	(true: Assert; false: Don't assert) the alert pin when charger is detected.
alert_load_detect	const uint8_t	(true: Assert; false: Don't assert) the alert pin when load is detected.
alert_other_faults	const uint8_t	(true: Assert; false: Don't assert) the alert pin when other faults, communication timeout or oscillator frequency error is detected.

Table 27. Members of the RAA489206 Alert Pin Configuration Structure (Cont.)

typedef struct st_raa489206_alert_cfg		
Member	Type	Description
alert_reg_oc_scan_idle	const uint8_t	(true: Assert; false: Don't assert) the alert pin when regulator overcurrent event is detected in SCAN or IDLE mode.
alert_reg_oc_low_power	const uint8_t	(true: Assert; false: Don't assert) the alert pin when regulator overcurrent event is detected in LOW POWER mode.
alert_vtemp_fault	const uint8_t	(true: Assert; false: Don't assert) the alert pin when Vtemp fault is detected.

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 28). Each nested structure contains the register address and type, a pointer to the default value loaded after reset, a pointer to the bit mask used during validation, a pointer to the last read register value, and the size of the value data fields in bytes. Table 29 shows the type options. All this information is used by the Middleware communication drivers to assemble the data packet that is sent over the communication interface, to validate the content of the BFE registers, and to check if the correct default values are loaded on start-up or after reset. The pointers help save memory as the size can vary between 8, 16, and 32 bits.

Table 28. Members of the RAA489206 Register Container Structure

typedef struct st_raa489206_register		
Member	Type	Description
address	const e_raa489206_reg_addr_t	Register Address
type	const e_raa489206_reg_type_t	Register type
p_def_value	const uint8_t	Pointer to register default value
p_bitmask	const uint8_t	Pointer to validation bitmask
p_value	uint8_t	Pointer to register value
size	const uint8_t	Register size (bytes)

Table 29. RAA489206 Register Type Enumeration

typedef enum e_raa489206_reg_type	
Constant	Description
READ_ONLY	Read only register
READ_WRITE	Read/Write register

The communication drivers use the pointers to the nested structures from the register bank as arguments:

```
/** Read device information */
bfe_err = raa489206_reg_read(p_ctrl, &p_bfe_regs->sys_info, RAA489206_SINGLE_REG);
```

The read value is returned in the global variable pointed by *p_value* and it can be easily extracted having the pointer and size. The write functions work in exactly the opposite way taking the value to be written from the pointed variable. During validation the bit mask designates the bits to be compared.

The full declaration of the register bank is in **r_bfe/r_raa489206_regs.c** and the definition in **r_bfe/r_raa489206_regs.h**. The following code demonstrates a part of it:

```
/* RAA489206 registers' bank */
st_raa489206_regs_t g_raa489206_registers =
{
    .sys_info          = {.address = RAA489206_SYSTEM_INFO,
                        .type = READ_ONLY,
```

```

        .p_bitmask = &g_mask_sys_info.value,
        .p_def_value = &g_def_sys_info.value,
        .p_value = &g_sys_info.value,
        .size = sizeof(g_sys_info.value) / sizeof(uint8_t)},
.global_op = {.address = RAA489206_GLOBAL_OPERATION,
        .type = READ_WRITE,
        .p_bitmask = &g_mask_global_op.value,
        .p_def_value = &g_def_global_op.value,
        .p_value = &g_global_op.value,
        .size = sizeof(g_global_op.value) / sizeof(uint8_t)},
.vcell_op = {.address = RAA489206_VCELL_OPERATION,
        .type = READ_WRITE,
        .p_bitmask = &g_mask_vcell_op.value,
        .p_def_value = &g_def_vcell_op.value,
        .p_value = &g_vcell_op.value,
        .size = sizeof(g_vcell_op.value) / sizeof(uint8_t)},
.ipack_op = {.address = RAA489206_IPACK_OPERATION,
        .type = READ_WRITE,
        .p_bitmask = &g_mask_ipack_op.value,
        .p_def_value = &g_def_ipack_op.value,
        .p_value = &g_ipack_op.value,
        .size = sizeof(g_ipack_op.value) / sizeof(uint8_t)},
.cell_select = {.address = RAA489206_CELL_SELECT,
        .type = READ_WRITE,
        .p_bitmask = &g_mask_cell_select.lsb_value,
        .p_def_value = &g_def_cell_select.lsb_value,
        .p_value = &g_cell_select.lsb_value,
        .size = sizeof(g_cell_select.value) / sizeof(uint8_t)},
.cell_overnvolt_th = {.address = RAA489206_VCELL_OV_TH,
        .type = READ_WRITE,
        .p_bitmask = &g_mask_cell_overnvolt_th,
        .p_def_value = &g_def_cell_overnvolt_th,
        .p_value = &g_cell_overnvolt_th,
        .size = sizeof(g_cell_overnvolt_th) / sizeof(uint8_t)},
.cell_undervolt_th = {.address = RAA489206_VCELL_UV_TH,
        .type = READ_WRITE,
        .p_bitmask = &g_mask_cell_undervolt_th,
        .p_def_value = &g_def_cell_undervolt_th,
        .p_value = &g_cell_undervolt_th,
        .size = sizeof(g_cell_undervolt_th) / sizeof(uint8_t)},

....

.cell_1_volt = {.address = RAA489206_VCELL_1,
        .type = READ_ONLY,
        .p_bitmask = &g_mask_cell_1_volt.lsb_value,
        .p_def_value = &g_def_cell_1_volt.lsb_value,
        .p_value = &g_cell_1_volt.lsb_value,
        .size = sizeof(g_cell_1_volt.value) / sizeof(uint8_t)},
.cell_2_volt = {.address = RAA489206_VCELL_2,

```

```

        .type = READ_ONLY,
        .p_bitmask = &g_mask_cell_2_volt.lsb_value,
        .p_def_value = &g_def_cell_2_volt.lsb_value,
        .p_value = &g_cell_2_volt.lsb_value,
        .size = sizeof(g_cell_2_volt.value) / sizeof(uint8_t)},
    ....

    .ipack_volt = {.address = RAA489206_IPACK_VOLTAGE,
        .type = READ_ONLY,
        .p_bitmask = &g_mask_ipack_volt.lsb_value,
        .p_def_value = &g_def_ipack_volt.lsb_value,
        .p_value = &g_ipack_volt.lsb_value,
        .size = sizeof(g_ipack_volt.value) / sizeof(uint8_t)},
    .ipack_timer = {.address = RAA489206_IPACK_TIMER,
        .type = READ_ONLY,
        .p_bitmask = &g_mask_ipack_timer.lsb_value,
        .p_def_value = &g_def_ipack_timer.lsb_value,
        .p_value = &g_ipack_timer.lsb_value,
        .size = sizeof(g_ipack_timer.value) / sizeof(uint8_t)},
    .etaux_0_volt = {.address = RAA489206_ETAUX_0_VOLTAGE,
        .type = READ_ONLY,
        .p_bitmask = &g_mask_etaux_0_volt.lsb_value,
        .p_def_value = &g_def_etaux_0_volt.lsb_value,
        .p_value = &g_etaux_0_volt.lsb_value,
        .size = sizeof(g_etaux_0_volt.value) / sizeof(uint8_t)},
    ....

    .pri_faults = {.address = RAA489206_PRIORITY_FAULTS,
        .type = READ_WRITE,
        .p_bitmask = &g_mask_pri_faults.value,
        .p_def_value = &g_def_pri_faults.value,
        .p_value = &g_pri_faults.value,
        .size = sizeof(g_pri_faults.value) / sizeof(uint8_t)},
    .etaux_faults = {.address = RAA489206_ETAUX_FAULTS,
        .type = READ_WRITE,
        .p_bitmask = &g_mask_etaux_faults.value,
        .p_def_value = &g_def_etaux_faults.value,
        .p_value = &g_etaux_faults.value,
        .size = sizeof(g_etaux_faults.value) / sizeof(uint8_t)},
    .other_faults = {.address = RAA489206_OTHER_FAULTS,
        .type = READ_WRITE,
        .p_bitmask = &g_mask_other_faults.value,
        .p_def_value = &g_def_other_faults.value,
        .p_value = &g_other_faults.value,
        .size = sizeof(g_other_faults.value) / sizeof(uint8_t)},
};

```

3.3 Command Bank

RAA489206 supports commands that allow reading multiple registers in a row using the SPI interface and a single CRC field for the whole transmission. The command bank holds all BFE commands in its fields. Each member is a nested structure with a predefined data type (Table 30). Each nested structure contains the command code, a pointer to the starting register, the total number of registers that a read using the command, and the total number of returned bytes.

Table 30. Members of the RAA489206 Register Container Structure

typedef struct st_raa489206_register		
Member	Type	Description
code	const e_raa489206_cmnd_code_t	Command Code
p_reg_1st	const st_raa489206_register_t	Starting register
reg_num	const uint8_t	Number of registers
length	const uint8_t	Data length (bytes)

The communication drivers use the pointers to the nested structures from the command bank as arguments:

```
/** Read all registers with a crc read command. */
bfe_err = raa489206_cmnd_read(p_ctrl, &p_ext_ctrl->p_bfe_cmnds->all_registers);
```

The read values are returned directly in the register bank. Similarly to the direct register access functions, the full declaration of the command bank can be found in **r_bfe/r_raa489206_regs.c** and the definition is in **r_bfe/r_raa489206_regs.h**. The following code demonstrates a part of it:

```
/** RAA489206 commands bank */
st_raa489206_cmnds_t g_raa489206_commands =
{
    .all_registers          = {.code = RAA489206_ALL_REGISTERS_CMND,
                              .p_reg_1st = &g_raa489206_registers.sys_info,
                              .reg_num = 83U,
                              .length = 138U},
    .measurements          = {.code = RAA489206_MEASUREMENT_CMND,
                              .p_reg_1st = &g_raa489206_registers.cell_1_volt,
                              .reg_num = 26U,
                              .length = 51U},
    .v_cell                = {.code = RAA489206_VCELL_CMND,
                              .p_reg_1st = &g_raa489206_registers.cell_1_volt,
                              .reg_num = 17U,
                              .length = 34U},
    ....
    .faults                = {.code = RAA489206_FAULTS_CMND,
                              .reg_1st = &g_raa489206_registers.pri_faults,
                              .reg_num = 6U,
                              .length = 7U},
};
```

3.4 Private (Static) Functions

Table 31 shows the declaration and description of the private functions used by the API functions in the source code. They operate on different levels in the Battery Abstraction Layer from data conversion to providing the communication with the BFE. For more details, refer to the examples in the following sections and the comments in the source code in **r_bfe/r_raa489206.c**.

Table 31. Static Functions Defined in the Source Code

Function	Description
static e_bfe_err_t raa489206_reset_hard (st_bfe_ctrl_t * const p_ctrl)	Toggles the RESET pin of RAA489206 to initiate a hard reset.
static e_bfe_err_t raa489206_reset_soft (st_bfe_ctrl_t * const p_ctrl)	Triggers a soft reset in RAA489206 to reset all register values to the factory settings, including data registers.
static e_bfe_err_t raa489206_reset_to_idle (st_bfe_ctrl_t * const p_ctrl)	Triggers a reset to idle in RAA489206 to reset all internal counters and state machines but keeps the configuration registers values intact.
static e_bfe_err_t raa489206_reg_write (st_bfe_ctrl_t * const p_ctrl, st_raa489206_register_t * const p_tar_reg)	Writes data in a register in RAA489206 using either SPI or I ² C communication interface.
static e_bfe_err_t raa489206_reg_read (st_bfe_ctrl_t * const p_ctrl, st_raa489206_register_t * const p_tar_reg, uint8_t reg_num)	Reads data from a single or multiple registers in RAA489206 using either SPI or I ² C communication interface.
static e_bfe_err_t raa489206_cmnd_read (st_bfe_ctrl_t * const p_ctrl, st_raa489206_command_t * const p_command)	Reads data from multiple RAA489206 registers within a single SPI transmission.
static e_bfe_err_t raa489206_reg_validate (st_bfe_ctrl_t * const p_ctrl, st_raa489206_register_t * const p_tar_reg, uint8_t reg_num)	Reads data from a single or multiple RAA489206 registers and compare the values with the register bank.
static e_bfe_err_t raa489206_cmnd_validate (st_bfe_ctrl_t * const p_ctrl, st_raa489206_command_t * const p_command, st_raa489206_register_t * const p_tar_reg_1, uint8_t reg_num_1, st_raa489206_register_t * const p_tar_reg_2, uint8_t reg_num_2)	Reads data from multiple RAA489206 registers using a command and compare the values with the register bank.
static e_bfe_err_t raa489206_spi_read_write (uint8_t reg_address, uint8_t slave_address, uint8_t * const p_data, uint32_t length, e_raa489206_comm_dir_t dir, uint8_t crc_use)	Reads single or multiple registers from the RAA489206 memory or writes into a single register from the RAA489206 memory, using the SPI communication interface.
static e_bfe_err_t raa489206_i2c_read_write (uint8_t reg_address, uint8_t slave_address, uint8_t * const p_data, uint32_t length, e_raa489206_comm_dir_t dir)	Reads single or multiple registers from the RAA489206 memory or writes into a single register from the RAA489206 memory, using the I ² C communication interface.
static e_bfe_err_t raa489206_crc16_calculate (uint8_t * const p_data_buffer, uint32_t length, uint32_t * const p_crc_result)	Calculates the CRC8 checksum of the data in the communication data buffer.
static e_bfe_err_t raa489206_busy_bit_clear_wait (st_bfe_ctrl_t * const p_ctrl, uint32_t time_offset, uint32_t busy_wait_max)	After triggering a measurement or a cell balancing sequence wait until the busy bit in Global Operation Register is cleared.
static fsp_err_t sci_spi_event_check (void)	Validates that the SPI transmission has completed successfully.
static fsp_err_t i2c_event_check (void)	Validates that the I ² C transmission has completed successfully.
static uint8_t raa489206_volt_to_reg (uint32_t volt, uint32_t set, uint32_t offset)	Calculates a register value from voltage.
static uint8_t raa489206_time_to_reg (uint32_t time, uint32_t set, uint32_t offset)	Calculates a register value from time.
static uint32_t raa489206_reg_to_volt (uint16_t reg_val, uint32_t set, uint32_t offset)	Calculates voltage from a value in a register.
static int32_t raa489206_reg_to_current (uint16_t reg_val, uint32_t step, uint32_t offset, uint32_t r_shunt_mohm)	Calculates bidirectional current from a value in a register.
static uint16_t raa489206_reg_to_current2 (uint16_t reg_val, uint32_t step, uint32_t offset, uint32_t r_shunt_mohm)	Calculates unidirectional current from a value in a register.

Table 31. Static Functions Defined in the Source Code

Function	Description
static uint8_t raa489206_int_temp_to_reg (uint16_t temp_degc)	Calculates register value from degree Celsius.
static uint16_t raa489206_reg_to_int_temp (uint8_t reg_val)	Calculates degree Celsius of internal NTC resistor from a register value.

3.5 API Implementation

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

3.5.1 R_RAA489206_Init

e_bfe_err_t R_RAA489206_Init (st_bfe_ctrl_t * const p_ctrl, st_bfe_cfg_t const * const p_cfg)		
Description	This function initializes RAA489206 by enabling and configuring the necessary peripheral modules of the MCU. It modifies the <i>p_ctrl->is_initialized</i> , <i>p_ctrl->is_balancing</i> , and <i>p_ctrl->is_cont_scanning</i> flags.	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Initializes a SPI or I²C interface to connect to RAA489206. Initializes a CRC interface, if required. Initializes interrupt modules. Resets RAA489206. Reads the die revision and part id of RAA489206. 	
Precondition	Perform a MCU peripheral communication module setup according to the requirements stated in the RAA489206 datasheet.	
Warnings	This function does not check the settings of the communication module!	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_cfg	A pointer to the BFE configuration structure
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_COMM_UNSUP_INTERFACE	The selected communication interface is not supported.
	BFE_ERR_INVALID_CELL	The selected total number of cells is incorrect.
	BFE_ERR_FSP	Error in the FSP layer.
	BMS_ERR_...	Inherit from raa489206_reset_hard()
	BMS_ERR_...	Inherit from raa489206_reg_read()

3.5.2 R_RAA489206_Deinit

e_bfe_err_t R_RAA489206_Deinit (st_bfe_ctrl_t * const p_ctrl);		
Description	This function deinitializes RAA489206. It disables the used peripheral modules of the MCU. It modifies the <i>p_ctrl->is_initialized</i> flag.	
Operation	<ul style="list-style-type: none"> Checks function input parameters. Deinitializes the SPI or I²C interface. Deinitializes the CRC interface, if used. Deinitializes the IRQ interfaces. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
Parameters	p_ctrl	A pointer to the control structure
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_COMM_UNSUP_INTERFACE	The selected communication interface is not supported.
	BFE_ERR_FSP	Error in the FSP layer

3.5.3 R_RAA489206_Setup

e_bfe_err_t R_RAA489206_Setup (st_bfe_ctrl_t * const p_ctrl, st_bfe_cfg_t const * const p_cfg);		
Description	This function configures RAA489206 by writing into all device setup registers. It extracts the necessary data from the configuration <i>p_cfg</i> and <i>p_cfg_ext</i> structures.	
Operation	<ul style="list-style-type: none"> Checks function parameters (optional). Checks the limits of the thresholds. Converts the configuration settings and writes into all configuration registers (0x01 - 0x2E, 0x83-0x88). Verifies if values are correctly written. Enables the charge pump when BFE_DRIVER_HIGH_SIDE is selected and checks its voltage. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_cfg	A pointer to the BFE configuration structure
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_CONF	A configuration threshold is out of limits.
	BFE_ERR_FET_CONTROL	The charge pump voltage is incorrect.
	BMS_ERR_...	Inherit from raa489206_reg_read()
	BMS_ERR_...	Inherit from raa489206_reg_write()
	BMS_ERR_...	Inherit from raa489206_cmnd_validate()
	BMS_ERR_...	Inherit from raa489206_reg_validate()

3.5.4 R_RAA489206_Reset

e_bfe_err_t R_RAA489206_Reset (st_bfe_ctrl_t * const p_ctrl, e_bfe_reset_type_t type);		
Description	This function resets RAA489206. Several predefined reset options can be set using the <i>type</i> input parameter. The function modifies the <i>p_ctrl->is_initialized</i> , <i>p_ctrl->is_cont_scanning</i> , and <i>p_ctrl->is_balancing</i> flags.	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Clears faults, timers and resets the state machine of the device (reset to idle); resets the digital part of the device (soft reset) or both digital and analog parts (hard reset) according to the selected reset type. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	After reset, user must rewrite all configuration registers by calling R_RAA489206_Setup!	
Parameters	p_ctrl	A pointer to the BFE control structure
	type	Hard, soft reset type or reset to idle selector.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	An invalid reset type was selected.
	BMS_ERR_...	Inherit from raa489206_reset_soft()
	BMS_ERR_...	Inherit from raa489206_reset_hard()
	BMS_ERR_...	Inherit from raa489206_reset_to_idle()

The values for the function parameter *type* are defined as constants in the BFE Reset Types Enumeration in file **r_bfe/r_bfe_api.h**. [Table 32](#) lists the supported reset options by the BFE.

Table 32. 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.
BFE_RESET_TYPE_TOIDLE	Reset the device to idle state

3.5.5 R_RAA489206_ModeSet

e_bfe_err_t R_RAA489206_ModeSet(st_bfe_ctrl_t * const p_ctrl, e_bfe_mode_t mode);		
Description	This function changes the mode of RAA489206. A mode or state is selected with the <i>mode</i> input parameter from a predefined list. The function modifies the <i>p_ctrl->is_low_power</i> flag.	
Operation	<ul style="list-style-type: none"> Checks input parameters (optional) Waits for the mode transition Writes into the Scan Operation Register to change the mode 	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
Parameters	p_ctrl	A pointer to the BFE control structure
	mode	Mode selection
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer
	BFE_ERR_UNSUPPORTED_MODE	The selected mode is invalid
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized
	BFE_ERR_FET_CONTROL	The charge pump is not active
	BMS_ERR_...	Inherit from raa489206_busy_bit_clear_wait()
	BMS_ERR_...	Inherit from raa489206_reg_write()
	BMS_ERR_...	Inherit from raa489206_raa489206_reg_validate()

The values for the function input parameter *mode* are defined as constants in the BFE States and Modes Enumeration in file **r_bfe/r_bfe_api.h**. [Table 33](#) shows the supported modes by the sample code.

Table 33. 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_STATE_SCAN	Single execution of measurements
BFE_MODE_LOW_POWER_MODE	BFE low power mode.
BFE_MODE_SHIP	BFE lowest power consumption

3.5.6 R_RAA489206_ModeRead

e_bfe_err_t R_RAA489206_ModeRead (st_bfe_ctrl_t * const p_ctrl, e_bfe_mode_t * const p_mode);		
Description	This function returns the current mode of RAA489206 into a parameter. It modifies the <i>p_ctrl->is_low_power</i> flag.	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Reads the Scan Operation Register. Reads the Global Operation Register when necessary. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_mode	A pointer to the returned mode.
	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	Input argument has invalid pointer.
Return Values	BFE_ERR_INVALID_STATE	The returned state by the BFE is invalid.
	BMS_ERR_...	Inherit from raa489206_reg_read().

Table 33 shows the expected modes returned as a result.

3.5.7 R_RAA489206_CommTest

e_bfe_err_t R_RAA489206_CommTest (st_bfe_ctrl_t * const p_ctrl);		
Description	This function tests the communication between the MCU and RAA489206. It is active also in LOW POWER and SHIP mode!	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Read Scan Operation Register to test communication 	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
Parameters	p_ctrl	A pointer to the BFE control structure
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an 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 raa489206_reg_read()

3.5.8 R_RAA489206_SelfDiag

e_bfe_err_t R_RAA489206_SelfDiag (st_bfe_ctrl_t * const p_ctrl, e_bfe_diag_option_t option);		
Description	This function runs a self-diagnostic test for RAA489206. The custom test measures and compares the regulator current, VCC voltage and internal temperature. The full test includes also an open-wire test.	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Puts the BFE in Idle mode. Makes sure that any previous measurement has been completed. Triggers a Vreg measurement. Checks the related fault bits in Priority Faults Register and Status Register. Triggers and open-wire test for the Vcn inputs Checks if any external temperature inputs are used and enables the internal pull-up resistor. Triggers an ETAUX measurement Checks the open-wire fault bit in the Priority Faults Register. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the <i>p_ctrl->is_fault_detected</i> flag after calling this function! Do not call this function when cell balancing or continuous scan are running!	

Parameters	p_ctrl	A pointer to the BFE control structure
	option	Selected an option for the self-diagnostic
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_OPERATION	The trigger is set to true during continuous scan or cell balancing.
	BFE_ERR_INVALID_ARGUMENT	An invalid diagnostic option was selected.
	BFE_ERR_...	Inherit from raa489206_reg_write().
	BFE_ERR_...	Inherit from raa489206_reg_read().
	BFE_ERR_...	Inherit from raa489206_busy_bit_clear_wait().

The values for the function input parameter *option* are defined as constants in the BFE Diagnostic Options Enumeration in file **r_bfe/r_bfe_api.h**. Table 34 shows the supported diagnostic options by the sample code.

Table 34. BFE Diagnostic Options Enumeration

typedef enum e_bfe_diag_option	
Constant	Description
BFE_FULL_TEST	Run a complete self-test
BFE_TEST_OW	Check for open wires
BFE_TEST_CUSTOM	Device specific test option (only regulator and internal temperature)

3.5.9 R_RAA489206_MemCheck

e_bfe_err_t R_RAA489206_MemCheck(st_bfe_ctrl_t * const p_ctrl, e_bfe_mem_check_option_t option);		
Description	This function compares the values of the registers in RAA489206 with predefined ones in the MCU memory.	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Validates the values of all configuration registers. (compares the read registers with the RAA489206 registers mirror in the MCU memory). Validates the default values of all registers. (compares the read registers with the RAA489206 default register data in the MCU memory). 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Run the default registers' values validation right after initialization and before modifying any registers!	
Parameters	p_ctrl	A 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	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_REGISTER_RESET_UNMATCHED	The default register values do not match
	BFE_ERR_...	Inherit from raa489206_cmnd_validate().
	BFE_ERR_...	Inherit from raa489206_reg_validate().
	BFE_ERR_...	Inherit from raa489206_cmnd_read().

The values for the input parameter *option* are defined as constants in the BFE Memory Check Options Enumeration in file `r_bfe/r_bfe_api.h`. Table 35 shows the supported memory check option by the sample code.

Table 35. BFE Memory Check Options Enumeration

typedef enum e_bfe_mem_check_option	
Constant	Description
BFE_CHECK_CONF_REGS	Verify the content of the BFE configuration registers.
BFE_CHECK_DEF_VALS	Check if the BFE registers have default values.

3.5.10 R_RAA489206_VPackGet

e_bfe_err_t R_RAA489206_VPackGet(st_bfe_ctrl_t * const p_ctrl, uint32_t * const p_value, uint8_t trigger, uint8_t read);		
Description	<p>This function acquires the battery pack voltage [mV]. The user can select to only trigger a measurement or read the result from last one, or both by setting the <i>trigger</i> and <i>read</i> input parameters.</p> <p>The returned data is converted into voltage!</p> <p>The function can modify the <i>p_ctrl->is_low_power</i> flag.</p> <p>The measured values are compared for faults in RAA489206 internally!</p>	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Makes sure that RAA489206 is in IDLE mode. Waits for any previous measurements to finish. Makes sure that the trigger bit is cleared. Sets the trigger bit to trigger a measurement. Waits for the Busy bit to clear. Reads the measured pack voltage and update the internal register bank. Converts the ADC value. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Ensure that any of the <i>trigger</i> or <i>read</i> input parameters is set to <i>true</i> ! Do not trigger a scan when cell balancing or continuous scan are running!	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_value	A pointer to the acquired data.
	trigger	Set <i>true</i> to trigger a measurement.
	read	Set <i>true</i> to read the results from the (last) measurement.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	None of the <i>trigger</i> or <i>read</i> is set to <i>true</i> .
	BFE_ERR_INVALID_OPERATION	The <i>trigger</i> is set to <i>true</i> during continuous scan or cell balancing.
	BFE_ERR_...	Inherit from raa489206_reg_read().
	BFE_ERR_...	Inherit from raa489206_reg_write().
	BFE_ERR_...	Inherit from raa489206_busy_bit_clear_wait().

3.5.11 R_RAA489206_IPackGet

e_bfe_err_t R_RAA489206_IPackGet(st_bfe_ctrl_t * const p_ctrl, bfe_i_pack_meas_t * const p_values, uint8_t trigger, uint8_t read)		
Description	<p>This function acquires the battery pack current and time. The user can select to only trigger a measurement or read the result from last one, or both by setting the <i>trigger</i> and <i>read</i> input parameters.</p> <p>The returned data is converted into current and milliseconds!</p> <p>The function can modify the <i>p_ctrl->is_low_power</i> flag.</p> <p>The measured values are compared for faults in RAA489206 internally!</p> <p>The <i>p_values</i> parameter must be declared as <i>u_raa489206_i_pack_meas_t</i>!</p>	
Operation	<ul style="list-style-type: none"> • Checks function input parameters (optional). • Makes sure that RAA489206 is in IDLE mode. • Waits for any previous measurements to finish. • Makes sure that the trigger bit is cleared. • Sets the trigger bit to trigger a measurement. • Waits for the Busy bit to clear. • Reads the measured pack current and time and updates the internal register bank. • Converts the ADC values. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Make sure that any of the <i>trigger</i> or <i>read</i> input parameters is set to true! Do not trigger a scan when cell balancing or continuous scan are running!	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_values	A pointer to the acquired data.
	trigger	Set <i>true</i> to trigger a measurement.
	read	Set <i>true</i> to read the results from the (last) measurement.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	None of the <i>trigger</i> or <i>read</i> is set to <i>true</i>
	BFE_ERR_INVALID_OPERATION	The <i>trigger</i> is set to <i>true</i> during continuous scan or cell balancing.
	BFE_ERR_...	Inherit from raa489206_reg_read()
	BFE_ERR_...	Inherit from raa489206_reg_write()
	BFE_ERR_...	Inherit from raa489206_cmnd_read()
	BFE_ERR_...	Inherit from raa489206_busy_bit_clear_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 **r_bfe/r_raa489206.h** and has the following content:

```
/** RAA489206 measure current data structure */
typedef union u_raa489206_i_pack_meas
{
    uint32_t vector[3];

    struct
    {
        int32_t i_pack;                ///< Pack current [mA]
        uint64_t time_ipack;           ///< Ipack time [ms]
    } measurements;
} u_raa489206_i_pack_meas_t;
```

3.5.12 R_RAA489206_VCellsGet

e_bfe_err_t R_RAA489206_VCellsGet(st_bfe_ctrl_t * const p_ctrl, bfe_vcell_meas_t * const p_values, uint8_t trigger, uint8_t read)		
Description	<p>This function acquires the voltages of all cells in the stack and the maximum delta cell voltage. You can select to only trigger a measurement or read the result from last one, or both by setting the <i>trigger</i> and <i>read</i> input parameters. The returned data is converted into voltage!</p> <p>The function can modify the <i>p_ctrl->is_low_power</i> flag.</p> <p>The measured values are compared for faults in RAA489206 internally!</p> <p>The <i>p_values</i> parameter must be declared as <i>u_raa489206_vcells_meas_t!</i></p>	
Operation	<ul style="list-style-type: none"> • Checks function input parameters (optional). • Make sure that RAA489206 is in IDLE mode. • Waits for any previous measurements to finish. • Makes sure that the trigger bit is cleared. • Sets the trigger bit to trigger a measurement. • Waits for the Busy bit to clear. • Reads the measured cell voltages and updates the internal register bank. • Converts the ADC values. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Make sure that any of the <i>trigger</i> or <i>read</i> input parameters is set to <i>true</i> ! Do not trigger a scan when cell balancing or continuous scan are running!	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_values	A pointer to the acquired data.
	trigger	Set <i>true</i> to trigger a measurement.
	read	Set <i>true</i> to read the results from the (last) measurement.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	None of the <i>trigger</i> or <i>read</i> is set to <i>true</i>
	BFE_ERR_INVALID_OPERATION	The <i>trigger</i> is set to <i>true</i> during continuous scan or cell balancing.
	BFE_ERR_...	Inherit from raa489206_reg_read()
	BFE_ERR_...	Inherit from raa489206_reg_write()
	BFE_ERR_...	Inherit from raa489206_cmnd_read()
	BFE_ERR_...	Inherit from raa489206_busy_bit_clear_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 **r_bfe/r_raa489206.h** and has the following content:

```
/** RAA489206 measured cells voltages data structure */
typedef union u_raa489206_vcells_meas
{
    uint16_t vector[17];

    struct
    {
        uint16_t v_cell_1;           ///< Cell 1 voltage [mV]
        uint16_t v_cell_2;           ///< Cell 2 voltage [mV]
        uint16_t v_cell_3;           ///< Cell 3 voltage [mV]
        uint16_t v_cell_4;           ///< Cell 4 voltage [mV]
        uint16_t v_cell_5;           ///< Cell 5 voltage [mV]
        uint16_t v_cell_6;           ///< Cell 6 voltage [mV]
        uint16_t v_cell_7;           ///< Cell 7 voltage [mV]
        uint16_t v_cell_8;           ///< Cell 8 voltage [mV]
    };
};
```

```

uint16_t v_cell_9;          ///< Cell 9 voltage [mV]
uint16_t v_cell_10;         ///< Cell 10 voltage [mV]
uint16_t v_cell_11;         ///< Cell 11 voltage [mV]
uint16_t v_cell_12;         ///< Cell 12 voltage [mV]
uint16_t v_cell_13;         ///< Cell 13 voltage [mV]
uint16_t v_cell_14;         ///< Cell 14 voltage [mV]
uint16_t v_cell_15;         ///< Cell 15 voltage [mV]
uint16_t v_cell_16;         ///< Cell 16 voltage [mV]
uint16_t v_delta_max;       ///< Max delta cell voltage [mV]
} measurements;

} u_raa489206_vcells_meas_t;

```

3.5.13 R_RAA489206_Temps

e_bfe_err_t R_RAA489206_Temps (st_bfe_ctrl_t * const p_ctrl, bfe_temp_meas_t * const p_values, uint8_t trigger, uint8_t read)		
Description	This function acquires the voltages of the external temperature inputs, the reference voltage and the internal temperature. You can select to only trigger a measurement or read the result from last one, or both by setting the <i>trigger</i> and <i>read</i> input parameters.	
	The returned data is converted into voltage and temperature! The function can modify the <i>p_ctrl->is_low_power</i> flag. The measured values are compared for faults in RAA489206 internally! The <i>p_values</i> parameter must be declared as <i>u_raa489206_temps_meas_t</i> !	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Makes sure that RAA489206 is in IDLE mode. Waits for any previous measurements to finish. Makes sure that the trigger bit is cleared. Sets the trigger bit to trigger a measurement of the external temperature inputs. Waits for the Busy bit to clear. Sets the trigger bit to trigger a measurement of the reference voltage and the internal temperature. Waits for the Busy bit to clear. Reads the measured voltages and temperature and updates the internal register bank. Converts the ADC values. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Make sure that any of the <i>trigger</i> or <i>read</i> input parameters is set to <i>true</i> ! Do not trigger a scan when cell balancing or continuous scan are running!	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_values	A pointer to the acquired data.
	trigger	Set <i>true</i> to trigger a measurement.
	read	Set <i>true</i> to read the results from the (last) measurement.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	None of the <i>trigger</i> or <i>read</i> is set to <i>true</i>
	BFE_ERR_INVALID_OPERATION	The <i>trigger</i> is set to <i>true</i> during continuous scan or cell balancing.
	BFE_ERR_...	Inherit from raa489206_reg_read()
	BFE_ERR_...	Inherit from raa489206_reg_write()
	BFE_ERR_...	Inherit from raa489206_cmnd_read()
	BFE_ERR_...	Inherit from raa489206_busy_bit_clear_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 *r_bfe/r_raa489206.h* and has the following content:

```
/** RAA489206 measured temperatures data structure */
typedef union u_raa489206_temps_meas
{
    uint16_t vector[4];

    struct
    {
        uint16_t v_therm_1;           ///< Thermistor 1 voltage [mV]
        uint16_t v_therm_2;           ///< Thermistor 2 voltage [mV]
        uint16_t int_temp;             ///< Internal temperature [0.1deg. C]
        uint16_t v_temp;               ///< VTEMP pin voltage [mV]
    } measurements;
} u_raa489206_temps_meas_t;
```

3.5.14 R_RAA489206_AllGet

e_bfe_err_t R_RAA489206_AllGet(st_bfe_ctrl_t * const p_ctrl, bfe_all_meas_t * const p_values, uint8_t trigger, uint8_t read)		
Description	<p>This function acquires the voltages of all cells, the maximum delta cell voltage, the battery pack voltage and current, the current timer, the internal temperature, the voltages of the external temperature inputs, the reference voltage, and the regulator voltage and currents. You can select to only trigger a measurement or read the result from last one, or both by setting the <i>trigger</i> and <i>read</i> input parameters.</p> <p>The returned data is converted into voltage, current, time and temperature!</p> <p>The function can modify the <i>p_ctrl->is_low_power</i> and <i>p_ctrl->is_fault_detected</i> flags.</p> <p>The function can modify the <i>p_ext_ctrl->pack_charge</i>, <i>p_ext_ctrl->pack_discharge</i>, <i>p_ext_ctrl->charger_present</i>, <i>p_ext_ctrl->load_present</i>, <i>p_ext_ctrl->end_of_charge_voltage</i>, <i>p_ext_ctrl->end_of_charge_current</i> and <i>p_ext_ctrl->battery_full</i> flags.</p> <p>The measured values are compared for faults in RAA489206 internally!</p> <p>To include cell balancing in the single system scan, you must configure for <i>BFE_BALANCE_MODE_AUTO</i> mode, set true <i>cb_single_sys_scan_enable</i> and call the <i>R_RAA489206_CellBalanceCtrl</i> function in advance to enable the process. The <i>p_values</i> parameter must be declared as <i>u_raa489206_all_meas_t</i>!</p>	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Disables cell balancing, if not allowed during single system scans. Makes sure that RAA489206 is in SCAN mode. Enables the ETAUX and Internal temperature during System Scan. Waits for any previous measurements to finish. Makes sure that the trigger bit is cleared. Sets the trigger bit to trigger a single system scan. Waits for the Busy bit to clear. Reads the measured voltages, currents and temperature and time, and update the internal register bank. Converts the ADC values. Reads the fault and status registers. Updates the <i>p_ctrl->is_fault_detected</i>, <i>p_ctrl->pack_charge</i>, <i>p_ctrl->pack_discharge</i>, <i>p_ctrl->charger_present</i>, <i>p_ctrl->load_present</i>, <i>p_ctrl->end_of_charge_voltage</i>, <i>p_ctrl->end_of_charge_current</i>, <i>p_ctrl->battery_full</i>, <i>p_ctrl->need_cell_balancing</i>, <i>p_ctrl->too_low_to_balance</i> and <i>p_ctrl->too_high_to_balance</i> flags. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Make sure that any of the <i>trigger</i> or <i>read</i> input parameters is set to 'true'! Always check the <i>p_ctrl->is_fault_detected</i> flag after calling this function! Do not trigger a scan when continuous scan is running! The status flags must be cleared manually using the <i>R_RAA489206_FaultsAllClear</i> function!	
Parameters	<i>p_ctrl</i>	A pointer to the BFE control structure
	<i>p_values</i>	A pointer to the acquired data.
	<i>trigger</i>	Set <i>true</i> to trigger a measurement.
	<i>read</i>	Set <i>true</i> to read the results from the (last) measurement.

Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	None of the <i>trigger</i> or <i>read</i> is set to <i>true</i> .
	BFE_ERR_INVALID_OPERATION	The <i>trigger</i> is set to <i>true</i> during continuous scan or cell balancing.
	BFE_ERR_...	Inherit from raa489206_reg_read()
	BFE_ERR_...	Inherit from raa489206_reg_write()
	BFE_ERR_...	Inherit from raa489206_cmnd_read()
	BFE_ERR_...	Inherit from raa489206_busy_bit_clear_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 **r_bfe/r_raa489206.h** and has the following content:

```

/** RAA489206 measure all data structure */
typedef union u_raa489206_all_meas
{
    uint16_t vector[31];

    struct
    {
        uint16_t v_cell_1;    ///< Cell 1 voltage [mV]
        uint16_t v_cell_2;    ///< Cell 2 voltage [mV]
        uint16_t v_cell_3;    ///< Cell 3 voltage [mV]
        uint16_t v_cell_4;    ///< Cell 4 voltage [mV]
        uint16_t v_cell_5;    ///< Cell 5 voltage [mV]
        uint16_t v_cell_6;    ///< Cell 6 voltage [mV]
        uint16_t v_cell_7;    ///< Cell 7 voltage [mV]
        uint16_t v_cell_8;    ///< Cell 8 voltage [mV]
        uint16_t v_cell_9;    ///< Cell 9 voltage [mV]
        uint16_t v_cell_10;   ///< Cell 10 voltage [mV]
        uint16_t v_cell_11;   ///< Cell 11 voltage [mV]
        uint16_t v_cell_12;   ///< Cell 12 voltage [mV]
        uint16_t v_cell_13;   ///< Cell 13 voltage [mV]
        uint16_t v_cell_14;   ///< Cell 14 voltage [mV]
        uint16_t v_cell_15;   ///< Cell 15 voltage [mV]
        uint16_t v_cell_16;   ///< Cell 16 voltage [mV]
        uint16_t v_delta_max; ///< Max delta cell voltage [mV]

        int32_t i_pack;        ///< Pack current [mA]
        uint64_t time_ipack;   ///< Ipack time [ms]

        uint16_t v_therm_1;    ///< Thermistor 1 voltage [mV]
        uint16_t v_therm_2;    ///< Thermistor 2 voltage [mV]

        uint32_t v_pack;       ///< Pack voltage [mV]

        uint16_t int_temp;     ///< Internal temperature [0.1deg. C]

        uint16_t v_temp;       ///< VTEMP pin voltage [mV]

        uint16_t v_vcc;        ///< Pin Vcc voltage [mV]
        uint16_t i_reg;        ///< Current of the external voltage regulator [mA]
    }
};

```

```

    } measurements;

} u_raa489206_all_meas_t;

```

3.5.15 R_RAA489206_MixGet

e_bfe_err_t R_RAA489206_AllGet (st_bfe_ctrl_t * const p_ctrl, bfe_other_meas_t * const p_values, uint8_t trigger, uint8_t read);		
Description	<p>This function acquires the voltages of all cells, the maximum delta cell voltage, the battery pack voltage and current and the current timer. You can select to only trigger a measurement or read the result from last one, or both by setting the <i>trigger</i> and <i>read</i> input parameters.</p> <p>The returned data is converted into voltage, current and time!</p> <p>The function can modify the <i>p_ctrl->is_low_power</i> and <i>p_ctrl->is_fault_detected</i> flags.</p> <p>The function can modify the <i>p_ext_ctrl->pack_charge</i>, <i>p_ext_ctrl->pack_discharge</i>, <i>p_ext_ctrl->charger_present</i>, <i>p_ext_ctrl->load_present</i>, <i>p_ext_ctrl->end_of_charge_voltage</i>, <i>p_ext_ctrl->end_of_charge_current</i> and <i>p_ext_ctrl->battery_full</i> flags.</p> <p>The measured values are compared for faults in RAA489206 internally!</p> <p>To include cell balancing in the single system scan, you must configure for <i>BFE_BALANCE_MODE_AUTO</i> mode, set <i>true cb_single_sys_scan_enable</i> and call the <i>R_RAA489206_CellBalanceCtrl</i> function in advance to enable the process.</p> <p>The <i>p_values</i> parameter must be declared as <i>u_raa489206_mix_meas_t!</i></p>	
	<ul style="list-style-type: none"> Checks function input parameters (optional). Disables cell balancing, if not allowed during single system scans. Makes sure that RAA489206 is in SCAN mode. Enables the ETAUX and Internal temperature during System Scan. Waits for any previous measurements to finish. Makes sure that the trigger bit is cleared. Sets the trigger bit to trigger a single system scan. Waits for the Busy bit to clear. Read the measured voltages, currents and time, and update the internal register bank. Converts the ADC values. Reads the fault and status registers. Updates the <i>p_ctrl->is_fault_detected</i>, <i>p_ctrl->pack_charge</i>, <i>p_ctrl->pack_discharge</i>, <i>p_ctrl->charger_present</i>, <i>p_ctrl->load_present</i>, <i>p_ctrl->end_of_charge_voltage</i>, <i>p_ctrl->end_of_charge_current</i>, <i>p_ctrl->battery_full</i>, <i>p_ctrl->need_cell_balancing</i>, <i>p_ctrl->too_low_to_balance</i>, and <i>p_ctrl->too_high_to_balance</i> flags. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Make sure that any of the <i>trigger</i> or <i>read</i> input parameters is set to true! Always check the <i>p_ctrl->is_fault_detected</i> flag after calling this function! Do not trigger a scan when continuous scan is running!	
Parameters	<i>p_ctrl</i>	A pointer to the BFE control structure
	<i>p_values</i>	A pointer to the acquired data.
	<i>trigger</i>	Set <i>true</i> to trigger a measurement.
	<i>read</i>	Set <i>true</i> to read the results from the (last) measurement.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_ARGUMENT	None of the <i>trigger</i> or <i>read</i> is set to <i>true</i> .
	BFE_ERR_INVALID_OPERATION	The <i>trigger</i> is set to <i>true</i> during continuous scan or cell balancing.
	BFE_ERR_...	Inherit from <i>raa489206_reg_read()</i>
	BFE_ERR_...	Inherit from <i>raa489206_reg_write()</i>
	BFE_ERR_...	Inherit from <i>raa489206_cmnd_read()</i>
	BFE_ERR_...	Inherit from <i>raa489206_busy_bit_clear_wait()</i>

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 **r_bfe/r_raa489206.h** and has the following content:

```

/** RAA489206 measure mix data structure */
typedef union u_raa489206_mix_meas
{
    uint16_t vector[25];

    struct
    {
        uint16_t v_cell_1;          ///< Cell 1 voltage [mV]
        uint16_t v_cell_2;          ///< Cell 2 voltage [mV]
        uint16_t v_cell_3;          ///< Cell 3 voltage [mV]
        uint16_t v_cell_4;          ///< Cell 4 voltage [mV]
        uint16_t v_cell_5;          ///< Cell 5 voltage [mV]
        uint16_t v_cell_6;          ///< Cell 6 voltage [mV]
        uint16_t v_cell_7;          ///< Cell 7 voltage [mV]
        uint16_t v_cell_8;          ///< Cell 8 voltage [mV]
        uint16_t v_cell_9;          ///< Cell 9 voltage [mV]
        uint16_t v_cell_10;         ///< Cell 10 voltage [mV]
        uint16_t v_cell_11;         ///< Cell 11 voltage [mV]
        uint16_t v_cell_12;         ///< Cell 12 voltage [mV]
        uint16_t v_cell_13;         ///< Cell 13 voltage [mV]
        uint16_t v_cell_14;         ///< Cell 14 voltage [mV]
        uint16_t v_cell_15;         ///< Cell 15 voltage [mV]
        uint16_t v_cell_16;         ///< Cell 16 voltage [mV]
        uint16_t v_delta_max;       ///< Max delta cell voltage [mV]

        int32_t i_pack;              ///< Pack current [mA]
        uint64_t time_ipack;         ///< Ipack time [ms]

        uint32_t v_pack;             ///< Pack voltage [mV]
    } measurements;
} u_raa489206_mix_meas_t;

```

3.5.16 R_RAA489206_FaultsAllRead

e_bfe_err_t R_RAA489206_FaultsAllRead (st_bfe_ctrl_t * const p_ctrl, bfe_faults_t * const p_faults);		
Description	This function checks for fault and reads the fault registers. It can be used for fault checking instead of R_RAA489206_FaultsCheck. The <i>p_faults</i> parameter must be declared as <i>st_raa489206_faults_t!</i>	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Reads all fault and status registers. Copies the status bits. Copies the fault bits into the faults data structure. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the <i>p_ctrl->is_fault_detected</i> flag after calling this function!	
Parameters	<i>p_ctrl</i>	A pointer to the BFE control structure
	<i>p_faults</i>	A pointer to the 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 raa489206_cmnd_read()
	BFE_ERR_...	Inherit from raa489206_reg_read()

The void pointer *p_faults* points to the structure where this API function returns the fault data. The structure type is redefined in the file **r_bfe/r_raa489206.h**. A non-zero member of the structure (true for Boolean types) indicates that an error is detected. It has the following content:

```
/** RAA489206 fault data structure */
typedef struct st_raa489206_faults
{
    uint8_t    flt_charge_over_current;    ///< Charge Overcurrent Fault
    uint8_t    flt_discharge_over_current; ///< Discharge Overcurrent Fault
    uint8_t    flt_short_circuit;          ///< Short-Circuit Fault

    uint8_t    flt_cell_undervoltage;      ///< Cell Undervoltage Fault
    uint8_t    flt_cell_overvoltage;       ///< Cell Overvoltage Fault
    uint8_t    flt_delta_cell_overvoltage; ///< Max delta cell voltage
                                                overvoltage Fault

    uint8_t    flt_pack_undervoltage;      ///< Pack Undervoltage Fault
    uint8_t    flt_pack_overvoltage;       ///< Pack Overvoltage Fault

    uint8_t    flt_internal_over_temp;     ///< Internal Over Temperature Fault
    uint8_t    flt_temp_ref_undervoltage;  ///< Vtemp Undervoltage Fault
    uint8_t    flt_under_temperature_charge; ///< Charge Under-Temperature Fault
    uint8_t    flt_over_temperature_charge;  ///< Charge Over-Temperature Fault
    uint8_t    flt_under_temperature_discharge; ///< Discharge Under-Temperature
                                                Fault
    uint8_t    flt_over_temperature_discharge; ///< Discharge Over-Temperature
                                                Fault
    uint8_t    flt_regulator_undervoltage;  ///< Undervoltage event on the Vcc pin
    uint8_t    flt_regulator_overcurrent;   ///< Overcurrent event on the
                                                external regulator
    uint8_t    flt_charge_pump;             ///< Charge pump Fault

    uint16_t   flt_ow_cells;                ///< Open wire fault
    uint8_t    flt_ow_ext_temp;             ///< Open-wire on xT0 and xT1 pins
    uint8_t    flt_ow_vbat;                ///< Open-wire on Vbat1 pin
    uint8_t    flt_ow_vss;                 ///< Open-wire on Vss pin

    uint8_t    flt_crc;                    ///< CRC Fault

    uint8_t    flt_other;                  ///< Communication timeout or
                                                Oscillator Fault
} st_raa489206_faults_t;
```

3.5.17 R_RAA489206_FaultsCheck

e_bfe_err_t R_RAA489206_FaultsCheck (st_bfe_ctrl_t * const p_ctrl);		
Description	This function checks ALERT pin and reads the Fault Register. When a fault is detected the <i>is_fault_detected</i> flag is set in <i>p_ctrl</i> .	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Checks the ALERT pin for assertion or reads all status registers and tests the fault bits. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the <i>p_ctrl->is_fault_detected</i> flag after calling this function!	
Parameters	p_ctrl	A 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 raa489206_reg_read()
	BFE_ERR_...	Inherit from raa489206_cmnd_read()

3.5.18 R_RAA489206_FaultsAllClear

e_bfe_err_t R_RAA489206_FaultsAllClear (st_bfe_ctrl_t * const p_ctrl, uint8_t * const p_success);		
Description	This function attempts to clear all faults in RAA489206. When all faults are successfully cleared the <i>p_success</i> flag is set.	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Sets the faults clear bit in Vbat1 Operation Register. Waits for the faults to clear. Reads back all fault and status registers to verify the clear. Updates the <i>p_ctrl->is_fault_detected</i>, <i>p_ctrl->pack_charge</i>, <i>p_ctrl->pack_discharge</i>, <i>p_ctrl->charger_present</i>, <i>p_ctrl->load_present</i>, <i>p_ctrl->end_of_charge_voltage</i>, <i>p_ctrl->end_of_charge_current</i>, <i>p_ctrl->battery_full</i>, <i>p_ctrl->need_cell_balancing</i>, <i>p_ctrl->too_low_to_balance</i>, and <i>p_ctrl->too_high_to_balance</i> flags. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Always check the <i>p_ctrl->p_success</i> flag after calling this function!	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_success	A pointer to a boolean variable use for confirmation.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_WRITE_VERIFY	Write command verification error.
	BMS_ERR_...	Inherit from raa489206_reg_read()
	BMS_ERR_...	Inherit from raa489206_reg_write()
	BMS_ERR_...	Inherit from raa489206_cmnd_read()

3.5.19 R_RAA489206_CellBalanceCtrl

e_bfe_err_t R_RAA489206_CellBalanceCtrl (st_bfe_ctrl_t * const p_ctrl, bfe_cb_cfg_t * const p_bal_cfg, e_bfe_process_ctrl_t ctrl_option);		
Description	This function controls the cell balancing function of RAA489206. The function modifies the <i>p_ext_ctrl->is_balancing</i> , <i>p_ext_ctrl->is_low_power</i> flags. The <i>p_bal_cfg</i> parameter is the CB FETs selection during manual mode and must be declared as <i>uint16_t</i> !	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Makes sure that the BFE is in IDLE mode when in BFE_BALANCE_MODE_MANUAL mode. Selects cells to be balanced when in BFE_BALANCE_MODE_MANUAL mode. Triggers cell balancing when not in BFE_BALANCE_MODE_OFF mode and BFE_PROCESS_ENABLE control option. Disables cell balancing when BFE_PROCESS_INHIBIT control option. Validates the Cell Balancing Operation Register content. 	
Precondition	The BFE interface should have already been initialized!	
Warnings	The <i>BFE_PROCESS_INHIBIT</i> control option cannot interrupt the cell balancing cycle but prevents a new one from starting!	
Parameters	<i>p_ctrl</i>	Pointer to the BFE control structure
	<i>p_bal_cfg</i>	Pointer to the balancing configuration structure.
	<i>ctrl_option</i>	Specify action to enable or inhibit cell balancing.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BFE_ERR_INVALID_CONF	Invalid control option was selected.
	BMS_ERR_...	Inherit from <i>raa489206_reg_write()</i>
	BMS_ERR_...	Inherit from <i>raa489206_reg_validate()</i>

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 *r_bfe/r_bfe_api.h*. [Table 36](#) shows the supported control options. [Table 14](#) shows the supported cell balancing modes.

Table 36. BFE Process Control Enumeration

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

3.5.20 R_RAA489206_IsCellBalancing

e_bfe_err_t R_RAA489206_IsCellBalancing (st_bfe_ctrl_t * const p_ctrl);		
Description	This function checks, if cell balancing has started in RAA489206. The function modifies the <i>p_ctrl->is_balancing</i> , <i>p_ext_ctrl->battery_full</i> , <i>p_ext_ctrl->need_cell_balancing</i> , <i>p_ext_ctrl->end_of_charge_voltage</i> , <i>p_ext_ctrl->end_of_charge_current</i> , <i>p_ext_ctrl->too_low_to_balance</i> , <i>p_ext_ctrl->too_high_to_balance</i> , and <i>p_ext_ctrl->cb_cell_state</i> flags.	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Reads the busy bit in the Global Operation Register. Reads the Cell Balancing Cell State Register. Reads the Cell Balancing Status Register and copies status bits states. 	
Precondition	The BFE interface should have already been initialized!	
Warnings	Always check the <i>p_ctrl->is_balancing</i> flag after calling this function!	
Parameters	<i>p_ctrl</i>	Pointer to the BFE control structure

Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BMS_ERR_...	Inherit from raa489206_reg_read()

3.5.21 R_RAA489206_ContScanCtrl

e_bfe_err_t R_RAA489206_ContScanCtrl (st_bfe_ctrl_t * const p_ctrl, bfe_scan_cont_cfg_t * const p_scan_cfg, e_bfe_process_ctrl_t ctrl_option)		
Description	This function controls the continuous scan function of RAA489206. The function modifies the <i>p_ctrl->is_cont_scanning</i> flag.	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Switch to SCAN mode. Check which measurements are included in the system scan Triggers or stop continuous system scan 	
Precondition	The BFE interface should have already been initialized!	
Warnings	The <i>p_scan_cfg</i> pointer must be empty!	
Parameters	p_ctrl	A pointer to the control structure
	p_scan_cfg	A 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	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BMS_ERR_...	Inherit from raa489206_reg_read()
	BMS_ERR_...	Inherit from raa489206_reg_write()
	BMS_ERR_...	Inherit from raa489206_reg_validate()

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 *r_bfe/r_bfe_api.h*. [Table 36](#) shows the supported control options.

3.5.22 R_RAA489206_WatchdogCtrl

e_bfe_err_t R_RAA489206_WatchdogCtrl (st_ctrl_t * const p_ctrl, bfe_watchdog_ctrl_t * const p_control_options);		
Description	This function configures the communication timeout of the RAA489206. The <i>p_options</i> parameter must be declared as <i>st_raa489204_comm_to_ctrl_t!</i>	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Reads Vbat1 Operation Register and enables or disables the communication timeout. Reads Vreg Operation Register and configures the communication timeout. Verifies register write (optional). 	
Precondition	The BFE interface should have already been initialized!	
Warnings	This function is not supported by the current API implementation!	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_options	A pointer to control options structure.

Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BMS_ERR_...	Inherit from raa489206_reg_read()
	BMS_ERR_...	Inherit from raa489206_reg_write()
	BMS_ERR_...	Inherit from raa489206_reg_validate()

The watchdog timeout is controlled with the control structure *p_control_options* defined in the file **r_bfe/r_raa489204.h**. [Table 37](#) shows its content. [Table 38](#) shows the options for the communication timeout listed in RAA489206 Communication Timeout Enumeration.

Table 37. Members of the RAA489206 Communication Timeout Control Structure

typedef struct st_raa489206_comm_to_ctrl		
Member	Type	Description
comm_timeout	e_raa489206_comm_to_t	Set the communication timeout before transition to LOW POWER mode
enable_timeout	uint8_t	Enable/disable the communication timeout

Table 38. RAA489206 Communication Timeout Enumeration

typedef enum e_raa489206_comm_to		
Constant	Value	Description
COMM_TO_128_MS	0x00	Communications Timeout 128ms
COMM_TO_512_MS	0x01	Communications Timeout 512ms
COMM_TO_2048_MS	0x02	Communications Timeout 2048ms
COMM_TO_4096_MS	0x03	Communications Timeout 4096ms

3.5.23 R_RAA489206_FETsCtrl

e_bfe_err_t R_RAA489206_FETsCtrl (st_ctrl_t * const p_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 controlled with the parameters <i>c_fet_state</i> and <i>d_fet_state</i> . The input parameter <i>group_num</i> is not used in the current implementation and its value is irrelevant.	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Checks the charge pump voltage if BFE_DRIVER_HIGH_SIDE has been selected. Modifies the FET control bits in Power FET Operation Register. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	Call the function only when the device is in IDLE or SCAN Mode!	
Parameters	p_ctrl	A 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_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an 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 voltage of the charge pump is incorrect.
	BFE_ERR_INVALID_OPERATION	The FETs cannot be controlled manually in the current BFE mode.
	BMS_ERR_...	Inherit from raa489206_reg_read()
	BMS_ERR_...	Inherit from raa489206_reg_write()

The states are fixed. Table 39 shows the available options for parameter *c_fet_state* controlling the charge FET and *d_fet_state* controlling the discharge FET.

Table 39. BFE Charge and Discharge FET State Options Enumeration

typedef enum e_bfe_fet_state	
Constant	Description
BFE_FET_ON	The FET is conducting.
BFE_FET_OFF	The FET is not conducting.

3.5.24 R_RAA489206_FETsRead

e_bfe_err_t R_RAA489206_FETsRead (st_bfe_ctrl_t * const p_ctrl, uint8_t group_num, e_bfe_fet_state_t p_c_fet_state, e_bfe_fet_state_t p_d_fet_state);		
Description	This function returns the state of the power FETs in the output parameters. The input parameters <i>group_num</i> is not used in the current implementation and its value is irrelevant. The function can be used in any supported BFE mode except SHIP mode!	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Reads the FET control bits in Power FET Operation Register. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
Parameters	p_ctrl	A pointer to the BFE control structure
	group_num	Select FET group to control.
	p_c_fet_state	A pointer to a status variable for the charge FET.
	p_d_fet_state	A pointer to a status variable for the discharge FET.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BMS_ERR_...	Inherit from raa489206_reg_read()

3.5.25 R_RAA489206_GPIOCtrl

e_bfe_err_t R_RAA489206_GPIOCtrl(st_bfe_ctrl_t * const p_ctrl, bfe_gpio_ctrl_t * const p_control_options);		
Description	<p>This function controls the GPIO pins of RAA489206.</p> <p>Use <code>BFE_GPIO_LOW</code> and <code>BFE_GPIO_HIGH_Z</code> as an argument for GPIO state when <code>p_ext_ctrl->gpio_configuration</code> is <code>GPIO_OUTPUTS</code>.</p> <p>Use <code>BFE_GPIO_LED_ON</code> and <code>BFE_GPIO_LED_OFF</code> as an argument for GPIO state when <code>p_ext_ctrl->gpio_configuration</code> is <code>GPIO_LED_DRIVE</code>.</p> <p>Returns <code>BFE_GPIO_LOW</code> and <code>BFE_GPIO_HIGH</code> as an argument for GPIO state when <code>p_ext_ctrl->gpio_configuration</code> is <code>GPIO_INPUTS</code>.</p> <p>The <code>p_control_options</code> parameter must be declared as <code>st_raa489206_gpio_ctrl_t!</code></p>	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Reads the content of the GPIO ALERT Operation Register Writes to the GPIO Status bits, if configured as outputs Copies the GPIO Status bits, if configured as inputs 	
Precondition	The BFE interface should have already been initialized.	
Warnings	This function is not supported when the GPIOs are used to drive LOW side FETs!	
Parameters	<code>p_ctrl</code>	A pointer to the BFE control structure
	<code>ctrl_option</code>	Pointer to GPIO pins control structure.
Return Values	<code>BFE_SUCCESS</code>	No error was returned.
	<code>BFE_ERR_INVALID_POINTER</code>	An input argument has an invalid pointer.
	<code>BFE_ERR_DEVICE_NOT_INITIALIZED</code>	The BFE interface is not initialized.
	<code>BFE_ERR_INVALID_CONF</code>	The function is not supported when low side FETs.
	<code>BFE_ERR_INVALID_ARGUMENT</code>	The selected FET state was not correct.
	<code>BMS_ERR_...</code>	Inherit from <code>raa489206_reg_read()</code>
	<code>BMS_ERR_...</code>	Inherit from <code>raa489206_reg_write()</code>
	<code>BMS_ERR_...</code>	Inherit from <code>raa489206_reg_validate()</code>

The GPIO levels are controlled by the void function parameter `p_control_options`. It points to a control structure, which is redefined in file `r_bfe/r_raa489206.h`. [Table 40](#) shows the content of that structure having members that correspond to each GPIO pin and [Table 41](#) shows the fixed options listed in RAA489206 GPIO States Enumeration.

Table 40. Members of the RAA489206 GPIO Control Structure

typedef struct st_raa489206_gpio_ctrl		
Member	Type	Description
<code>gpio_0_state</code>	<code>e_raa489206_gpio_states_t</code>	GPIO0 state
<code>gpio_1_state</code>	<code>e_raa489206_gpio_states_t</code>	GPIO1 state
<code>gpio_2_state</code>	<code>e_raa489206_gpio_states_t</code>	GPIO2 state
<code>gpio_3_state</code>	<code>e_raa489206_gpio_states_t</code>	GPIO3 state

Table 41. RAA489206 GPIO States Enumeration

typedef enum e_raa489206_gpio_states	
Constant	Description
<code>BFE_GPIO_LOW</code>	GPIO pin is asserted to the Vss pin
<code>BFE_GPIO_HIGH</code>	GPIO pin level is high
<code>BFE_GPIO_HIGH_Z</code>	GPIO pin is in high-impedance state
<code>BFE_GPIO_LED_ON</code>	GPIO pin is asserted to the Vss pin
<code>BFE_GPIO_LED_OFF</code>	GPIO pin is in high-impedance state

3.5.26 R_RAA489206_RegisterRead

e_bfe_err_t R_RAA489206_RegisterRead (st_bfe_ctrl_t * const p_ctrl, bfe_register_t * const p_register);		
Description	This function reads a register in RAA489206. The address of the target register structure must be taken from the register container in <i>p_ctrl</i> . The <i>p_register</i> parameter must be declared as <i>st_raa489206_quick_reg_t!</i>	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Sends a read command. Copies the read data. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
Parameters	p_ctrl	A pointer to the BFE control structure
	p_register	A pointer to the register address and data container.
Return Values	BFE_SUCCESS	No error was returned.
	BFE_ERR_INVALID_POINTER	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BMS_ERR_...	Inherit from raa489206_reg_read()

The void type API function parameter *p_register* provides path to the target register and data. It is redefined as a RAA489206 Quick Register Access Structure in file *r_bfe/r_raa489206.h*. Table 42 shows the content the structure. The register address is assigned with a member of the register bank, described in Table 28. The returned data is available after calling the function in the second member of the structure *data*.

Table 42. Members of the RAA489206 Quick Register Access Structure

typedef struct st_raa489206_quick_reg		
Member	Type	Description
p_target_register	st_raa489206_register_t	Pointer to register container.
data	uint32_t	Data field.

3.5.27 R_RAA489206_RegisterWrite

e_bfe_err_t R_RAA489206_RegisterWrite (st_ctrl_t * const p_ctrl, bfe_register_t * const p_register);		
Description	This function writes to a register in RAA489206. The address of the target register structure must be taken from the register container in <i>p_ctrl</i> . The <i>p_register</i> parameter must be declared as <i>st_raa489206_quick_reg_t!</i>	
Operation	<ul style="list-style-type: none"> Checks function input parameters (optional). Copies the data to be written. Sends a write command. 	
Precondition	The BFE interface should have already been initialized.	
Warnings	-	
Parameters	p_ctrl	A 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	An input argument has an invalid pointer.
	BFE_ERR_DEVICE_NOT_INITIALIZED	The BFE interface is not initialized.
	BMS_ERR_...	Inherit from raa489206_reg_write()

3.6 Configuration

3.6.1 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 the file **configuration.xml**. It is not necessary to change anything in the FSP configuration as long as the same RA4E1 MCU and ISO_DONGLE_EV1Z REV D communication dongle are used. Changing the pin-arrangement or migration to another MCU from the Renesas Advanced family requires generation of new HAL drivers from using the FSP Configurator in e2studio. The RAA489206 software library can be used also with other ARM-based MCUs. However, this requires careful replacement of all HAL drivers with alternatives that provide the same level of performance.

3.6.2 Battery Front End

Enter the BFE settings in the configuration structures, defined in **bal_data.c**. The members of *g_bfe0_cfg* and its extension *g_bfe0_ext_cfg* are constants that are initialized with the required settings during declaration and cannot be further modified in the code. Instructions about the initialization values in the comment sections of the type definitions of those structures can be found in **r_bfe/r_bfe_api.h** and **r_bfe/r_bfe_raa489206.h**. Some of the types are enumerations with fixed constants. The following code demonstrates only part of the declaration and initialization of the structures:

```
/** Extended configuration structure */
const st_raa489206_ext_cfg_t g_bfe0_ext_cfg =
{
    /** Shunt Resistors */
    /** Shunt resistance for charge and discharge currents [uohm] */
    .r_ipack_shunt_mohm = 5000,
    /** Shunt resistance for the external voltage regulator [mohm] */
    .r_ireg_shunt_mohm = 3300,

    /** Fault Thresholds and Delays */
    /** Short-circuit threshold [mA] (range: ((-40.176...-321.33mV) /
    r_ipack_shunt_uohm) * 1000000) */
    .th_short_circuit_a = -40000,
    /** Short-circuit delay [us] (range: 250...4219us) */
    .delay_short_circuit_us = 1000U,

    ....

    /** Cell overvoltage threshold [mV] (range: 1...4.820mV) */
    .th_cell_overnvolt_mv = 4250U,
    /** Cell undervoltage threshold [mV] (range: 1...4.801mV) */
    .th_cell_undervolt_mv = 2500U,

    ....

    /** Over-temp threshold when charging for external input 0 [mV] (range: 6...1607mV)
    */
    .th_charge_over_temp_0_mv = 275U,
    /** Under-temp threshold when charging for external input 0 [mV] (range: 6...1607mV)
    */
    .th_charge_under_temp_0_mv = 886U,
```

```

.....

/* Cell Balancing Configuration */
.cb_cfg =
{
/** The cell balancing mode */
.mode = BFE_BALANCE_MODE_AUTO,
/** Select internal or external FETs for cell balancing */
.cb_fets = BFE_BALANCE_EXTERNAL,

.....

/** (true: Allow; false: Don't allow) cell balancing during charging */
.cb_charging_enable = true,
/** (true: Allow; false: Don't allow) cell balancing after end of charging */
.cb_end_of_charge = true
},

/* Other */
/** (true: Weak; false: Strong) regulator in LOW POWER and SHIP mode */
.weak_regulator = false,
/** Select GPIO port direction and function */
.gpio_configuration = GPIO_LED_DRIVE,

.....

/* Alert Pin */
/** Configuration when to assert the ALERT pin */
.alert_pin_cfg =
{
/** (true: Assert; false: Don't assert) the alert pin when an undervoltage event at
the VCC pin is detected. */
.alert_vcc_undervoltage = true,

.....

/** (true: Assert; false: Don't assert) the alert pin when Vtemp fault is detected.
*/
.alert_vtemp_fault = true,
},
/** (false: drive alert pin constantly; true: pulse alert pin 2ms on time, 10ms
period) */
.alert_pin_pulse_enable = false,

/* Pins' configuration */
.pin_alert = ALERT_PIN,      ///< ALERT pin
.pin_reset = RESET_PIN,     ///< RESET pin
.pin_wake_up = WAKE_UP_PIN,  ///< Wake Up pin
};

```

```

/** Set which cells exist in the battery pack! */
const uint32_t g_bfe0_cells_cfg = CELL_COUNT_10;

/** Set which external temperature inputs are used! */
const uint16_t g_bfe0_ext_temps_cfg = EXT_TEMP_XT0_AND_XT1;

/** Configuration structure */
const st_bfe_cfg_t g_bfe0_cfg =
{
    /** Pointer to a constant for existing cells in the battery pack */
    .p_cells_select = &g_bfe0_cells_cfg,          ///< Do not modify!!!
    /** Pointer to a constant for used temperature inputs in the BFE */
    .p_temps_select = &g_bfe0_ext_temps_cfg,      ///< Do not modify!!!
    /** The total number of cells in series in the battery pack */
    .cells_in_series = 10U,
    /** The number of cells in parallel in the battery pack */
    .cells_in_parallel = 1U,
    /** The number of BFEs in a stack */
    .stack_size = RAA489206_MAX_STACK_SIZE,      ///< Do not modify!!!
    /** Communication interface used between the BFE and the MCU */
    .peripheral_type = BFE_COMMUNICATION_INTERFACE_SPI,
    /** Configuration of the FET driver */
    .driver_cfg = BFE_DRIVER_HIGH_SIDE,
    /** Configuration of power FETs to manage charge and discharge */
    .fet_cfg = BFE_FET_CONFIG_SERIES,
    /** Pointer to extended BFE configuration structure */
    .p_extend = &g_bfe0_ext_cfg,                ///< Do not modify!!!
};

```

3.6.3 Battery Abstraction Layer

The Battery Abstraction Layer is configured in **r_bfe/r_bfe_cfg.h**. The file contains pre-processor macros (Table 43). The macros are used for enabling/disabling parts of the code (such as verify write into register and check input parameters). The available options for the values can be found in the description section inside the table and in the comment sections of the source code. All macro values are Boolean (0 or 1).

Table 43. BFE Software Library Configuration Settings

Option (Macro Name)	Default Value	Description
BFE_CFG_PARAM_CHECKING_ENABLE	1	Functions check input parameters: 0 - Disable 1 - Enable (Recommended).
BFE_REG_WRITE_VERIFY_ENABLE	1	Register verification after write command: 0 - Disable 1 - Enable (Recommended).
BFE_CFG_REG_WRITE_VERIFY_ENABLE	1	Configuration register verification after write command: 0 - Disable 1 - Enable (Recommended).
BFE_ALRT_FAULT_CHECKING_ENABLE	1	Use ALRT pin for fault detection: 0 - Disable 1 - Enable (Recommended).

3.7 Examples

This section demonstrates a use case of all API functions and the communication drivers. Either the API functions from file **r_bfe/r_bfe_api.h** or directly their implementation from file **r_bfe/r_raq489206.h** can be used. In the second case when keeping the application but changing the BFE, replace all the functions rather than just reconnect the interface. For more examples, refer to the examples in **apps/bms.c** and the implementations themselves in **r_bfe/r_raq489206.c**.

- Initialization, setup, memory test, self-diagnostic, FET control, GPIO control, measure all BFE parameters

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

static e_bfe_fet_state_t      s_cfet = BFE_FET_OFF;
static e_bfe_fet_state_t      s_dfet = BFE_FET_OFF;
static u_raq489206_all_meas_t s_meas_data_all = {0};
static st_raq489206_faults_t  s_faults_data = {0};
static st_raq489206_gpio_ctrl_t s_gpio_ctrl = {0};

/** Open the Battery Front End interface. */
bfe_err = R_RAA489206_Init(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Check default values of the device registers. */
bfe_err = R_RAA489206_MemCheck(&g_bfe0_ctrl, BFE_CHECK_DEF_VALS);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Configure the Battery Front End. */
bfe_err = R_RAA489206_Setup(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Test for Open Wire. */
bfe_err = R_RAA489206_SelfDiag(&g_bfe0_ctrl, BFE_TEST_OW);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

if(false != g_bfe0_ctrl.is_fault_detected) // Check if the test passed successfully
{
    /** Read the faults. */
    bfe_err = R_RAA489206_FaultsAllRead(&g_bfe0_ctrl, &s_faults_data);
    BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

    /** Add a fault handler */
    __BKPT(0);
}

/** Turn ON the CFET and DFET. */
bfe_err = R_RAA489206_FETsCtrl(&g_bfe0_ctrl, 0, BFE_FET_ON, BFE_FET_ON);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Measure all. */
bfe_err = R_RAA489206_AllGet(&g_bfe0_ctrl, &s_meas_data_all, true, true);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return
```

```

/** Check if a fault was detected during measurement */
if(false != g_bfe0_ctrl.is_fault_detected)
{
    /** Read the faults. */
    bfe_err = R_RAA489206_FaultsAllRead(&g_bfe0_ctrl, &s_faults_data);
    BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

    /** Add fault handler */
    BKPT(0);
}

/** Read the state of the FETs. */
bfe_err = R_RAA489206_FETsRead(&g_bfe0_ctrl, 0, &s_cfet, &s_dfet);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Check the configuration registers. */
bfe_err = R_RAA489206_MemCheck(&g_bfe0_ctrl, BFE_CHECK_CONF_REGS);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/* Write to GPIOs */
s_gpio_ctrl.gpio_0_state = BFE_GPIO_LED_ON;
s_gpio_ctrl.gpio_1_state = BFE_GPIO_LED_OFF;
s_gpio_ctrl.gpio_2_state = BFE_GPIO_LED_ON;
s_gpio_ctrl.gpio_3_state = BFE_GPIO_LED_OFF;

bfe_err = R_RAA489206_GPIOCtrl(&g_bfe0_ctrl, &s_gpio_ctrl);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Turn OFF the CFET and DFET. */
bfe_err = R_RAA489206_FETsCtrl(&g_bfe0_ctrl, 0, BFE_CFET_OFF, BFE_DFET_OFF);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Close the Battery Front End interface. */
bfe_err = R_RAA489206_Deinit(&g_bfe0_ctrl);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

```

- Initialization, setup, enable communication timeout, FET control, measure pack voltage, pack current, temperatures, trigger and read separately the cell voltages, read and clear any faults

```

e_bfe_err_t bfe_err = BFE_SUCCESS; // Error code

static uint32_t          s_pack_voltage = 0;
static u_raa489206_i_pack_meas_t s_pack_current = {0};
static u_raa489206_temps_meas_t s_meas_data_temps = {0};
static u_raa489206_vcells_meas_t s_meas_data_cells = {0};

static st_raa489206_faults_t s_faults_data = {0};
static uint8_t              s_success = false;

static st_raa489206_comm_to_ctrl_t s_comm_timeout =
{.comm_timeout = COMM_TO_2048_MS, .enable_timeout = true};

```

```
/** Open the Battery Front End interface. */
bfe_err = R_RAA489206_Init(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Configure the Battery Front End. */
bfe_err = R_RAA489206_Setup(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Set a communication timeout */
bfe_err = R_RAA489206_WatchdogCtrl(&g_bfe0_ctrl, &s_comm_timeout);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Turn ON the CFET and DFET. */
bfe_err = R_RAA489206_FETsCtrl(&g_bfe0_ctrl, 0, BFE_FET_ON, BFE_FET_ON);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Measure the pack voltage. */
bfe_err = R_RAA489206_VPackGet(&g_bfe0_ctrl, &s_pack_voltage, true, true);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/* Measure pack current. */
bfe_err = R_RAA489206_IPackGet(&g_bfe0_ctrl, &s_pack_current, true, true);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/* Measure temperatures. */
bfe_err = R_RAA489206_TempsGet(&g_bfe0_ctrl, &s_meas_data_temps, true, true);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/* Trigger a cell voltage measurement. */
bfe_err = R_RAA489206_VCellsGet(&g_bfe0_ctrl, NULL, true, false);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

R_BSP_SoftwareDelay(100U, BSP_DELAY_UNITS_MILLISECONDS);

/* Read the cell voltages. */
bfe_err = R_RAA489206_VCellsGet(&g_bfe0_ctrl, &s_meas_data_cells, false, true);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Check for faults. */
bfe_err = R_RAA489206_FaultsCheck(&g_bfe0_ctrl);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Check if a fault was detected during measurement */
if(false != g_bfe0_ctrl.is_fault_detected)
{
    /** Read the faults. */
    bfe_err = R_RAA489206_FaultsAllRead(&g_bfe0_ctrl, &s_faults_data);
    BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

    /** Add fault handler */
    __BKPT(0);
}
```

```
/** Clear the faults. */
bfe_err = R_RAA489206_FaultsAllClear(&g_bfe0_ctrl, &s_success);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

if(true != s_success) // Check if the faults were cleared
{
    /** Add fault handler */
    __BKPT(0);
}

/** Turn OFF the CFET and DFET. */
bfe_err = R_RAA489206_FETsCtrl(&g_bfe0_ctrl, 0, BFE_CFET_OFF, BFE_DFET_OFF);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Close the Battery Front End interface. */
bfe_err = R_RAA489206_Deinit(&g_bfe0_ctrl);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return
```

- Initialization, setup, FET control, change BFE modes, measure mixed parameters, test communication

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

static e_bfe_mode_t s_bfe_mode = 0;
static u_kaa489206_mix_meas_t s_meas_data_mix = {0};

/** Open the Battery Front End interface. */
bfe_err = R_RAA489206_Init(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Configure the Battery Front End. */
bfe_err = R_RAA489206_Setup(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Turn ON the CFET and DFET. */
bfe_err = R_RAA489206_FETsCtrl(&g_bfe0_ctrl, 0, BFE_FET_ON, BFE_FET_ON);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Put the BFE in Low Power Mode. */
bfe_err = R_RAA489206_ModeSet(&g_bfe0_ctrl, BFE_MODE_LOW_POWER);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Read the BFE's Mode. */
bfe_err = R_RAA489206_ModeRead(&g_bfe0_ctrl, &s_bfe_mode);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Put the BFE in Idle Mode. */
bfe_err = R_RAA489206_ModeSet(&g_bfe0_ctrl, BFE_MODE_IDLE);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Measure the mixed parameters. */
bfe_err = R_RAA489206_MixGet(&g_bfe0_ctrl, &s_meas_data_mix, true, true);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return
```

```

/** Put the BFE in Ship Mode. */
bfe_err = R_RAA489206_ModeSet(&g_bfe0_ctrl, BFE_MODE_SHIP);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Wait for the mode transition */
R_BSP_SoftwareDelay(2100U, BSP_DELAY_UNITS_MILLISECONDS);

/** Test communication. */
bfe_err = R_RAA489206_CommTest(&g_bfe0_ctrl);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Close the Battery Front End interface. */
bfe_err = R_RAA489206_Deinit(&g_bfe0_ctrl);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

▪ Initialization, setup, FET control, turn on automatic cell balancing and continuous system scan
e_bfe_err_t bfe_err = BFE_SUCCESS; // Error code

static u_raa489206_all_meas_t s_meas_data_all = {0};

/** Open the Battery Front End interface. */
bfe_err = R_RAA489206_Init(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Configure the Battery Front End. */
bfe_err = R_RAA489206_Setup(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Turn ON the CFET and DFET. */
bfe_err = R_RAA489206_FETsCtrl(&g_bfe0_ctrl, 0, BFE_FET_ON, BFE_FET_ON);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Automatic Cell Balancing Control.
 * Make sure that in bal_data.c '.mode = BFE_BALANCE_MODE_AUTO'! */
bfe_err = R_RAA489206_CellBalanceCtrl(&g_bfe0_ctrl, NULL, BFE_PROCESS_ENABLE);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Start the continuous system scan. */
bfe_err = R_RAA489206_ContScanCtrl(&g_bfe0_ctrl, NULL, BFE_PROCESS_ENABLE);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Wait for a couple of loops */
R_BSP_SoftwareDelay(300U, BSP_DELAY_UNITS_MILLISECONDS);

/** Read the last measurement data */
bfe_err = R_RAA489206_AllGet(&g_bfe0_ctrl, &s_meas_data_all, false, true);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Stop the continuous system scan. */
bfe_err = R_RAA489206_ContScanCtrl(&g_bfe0_ctrl, NULL, BFE_PROCESS_INHIBIT);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

```

```

/** Reset BFE to Idle mode. */
bfe_err = R_RAA489206_Reset(&g_bfe0_ctrl, BFE_RESET_TYPE_TOIDLE);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Close the Battery Front End interface. */
bfe_err = R_RAA489206_Deinit(&g_bfe0_ctrl);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

▪ Initialization, setup, manual cell balancing, direct register read and write
e_bfe_err_t bfe_err = BFE_SUCCESS; // Error code

static u_raa489206_vcells_meas_t s_meas_data_cells = {0};
static uint16_t s_cb_man_cell_sel = 0;
static st_raa489206_quick_reg_t s_custom_reg = {0};

st_raa489206_regs_t * p_bfe_regs = (st_raa489206_regs_t *) g_bfe0_ctrl.p_bfe_regs;

/** To enable manual cell balancing change the cell balancing configuration in
bal_data.c so that '.mode = BFE_BALANCE_MODE_MANUAL'! */

/** Open the Battery Front End interface. */
bfe_err = R_RAA489206_Init(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Configure the Battery Front End. */
bfe_err = R_RAA489206_Setup(&g_bfe0_ctrl, &g_bfe0_cfg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Write directly in the Cell Balancing On Time Register */
s_custom_reg.p_targer_register = &p_bfe_regs->cb_on_time;
s_custom_reg.data = 0xFE; // Change the CB On Time to 1016ms.

bfe_err = R_RAA489206_RegisterWrite(&g_bfe0_ctrl, &s_custom_reg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Read directly the Cell Balancing On Time Register */
s_custom_reg.p_targer_register = &p_bfe_regs->cb_on_time;
s_custom_reg.data = 0;

bfe_err = R_RAA489206_RegisterRead(&g_bfe0_ctrl, &s_custom_reg);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Measure the cell voltages. */
bfe_err = R_RAA489206_VCellsGet(&g_bfe0_ctrl, &s_meas_data_cells, true, true);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

/** Manual Cell Balancing Control.
 * Make sure that in bal_data.c '.mode = BFE_BALANCE_MODE_MANUAL'! */
s_cb_man_cell_sel = 0xA0B;

bfe_err = R_RAA489206_CellBalanceCtrl(&g_bfe0_ctrl, &s_cb_man_cell_sel,
BFE_PROCESS_ENABLE);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return

```

```
/** Check the cell balancing status. */
bfe_err = R_RAA489206_IsCellBalancing(&g_bfe0_ctrl);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return
```

```
/** Close the Battery Front End interface. */
bfe_err = R_RAA489206_Deinit(&g_bfe0_ctrl);
BFE_ERR_HANDLER(bfe_err != BFE_SUCCESS); // Check for error return
```

- **Read register** – This function supports reading multiple registers within a single transmission. However, when using SPI with CRC, Renesas highly recommends using read commands for more efficient communication.

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

```
/** Data buffers */
static uint8_t s_tx_buffer[RAA489206_TX_DATA_LENGTH] = {0};
static uint8_t s_rx_buffer[RAA489206_RX_DATA_LENGTH] = {0};
static uint8_t s_data_buffer[RAA489206_RX_DATA_LENGTH] = {0};
```

```
/** Read device information */
bfe_err = raa489206_reg_read(p_ctrl, &p_bfe_regs->sys_info, RAA489206_SINGLE_REG);
BFE_ERR_RETURN_FALSE(BFE_SUCCESS == bfe_err, bfe_err); ///< Check for errors.
```

- **Write register** – The user cannot write into multiple registers within a single transmission.

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

```
/** Data buffers */
static uint8_t s_tx_buffer[RAA489206_TX_DATA_LENGTH] = {0};
static uint8_t s_rx_buffer[RAA489206_RX_DATA_LENGTH] = {0};
static uint8_t s_data_buffer[RAA489206_RX_DATA_LENGTH] = {0};
```

```
/** Write Global Operation Register */
bfe_err = raa489206_reg_write(p_ctrl, &p_bfe_regs->global_op);
BFE_ERR_RETURN_FALSE(BFE_SUCCESS == bfe_err, bfe_err); ///< Check for errors.
```

- **Validate register content** – Multiple registers within a single transmission can be validated.

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

```
/** Data buffers */
static uint8_t s_tx_buffer[RAA489206_TX_DATA_LENGTH] = {0};
static uint8_t s_rx_buffer[RAA489206_RX_DATA_LENGTH] = {0};
static uint8_t s_data_buffer[RAA489206_RX_DATA_LENGTH] = {0};
```

```
/** Read and compare the Scan Operation register. */
bfe_err = raa489206_reg_validate(p_ctrl, &p_bfe_regs->scan_op,
RAA489206_SINGLE_REG);
BFE_ERR_RETURN_FALSE(BFE_SUCCESS == bfe_err, bfe_err); ///< Check for errors.
```

- **Read multiple registers using a command** – This function works only when SPI with CRC is selected.

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

```
/** Data buffers */
static uint8_t s_tx_buffer[RAA489206_TX_DATA_LENGTH] = {0};
```

```
static uint8_t s_rx_buffer[RAA489206_RX_DATA_LENGTH] = {0};
static uint8_t s_data_buffer[RAA489206_RX_DATA_LENGTH] = {0};

/** Read all registers with a crc read command. */
bfe_err = raa489206_cmnd_read(p_ctrl, &p_ext_ctrl->p_bfe_cmnds->all_registers);
BFE_ERR_RETURN_FALSE(BFE_SUCCESS == bfe_err, bfe_err); ///< Check for errors.
```

- **Validate multiple registers using a command** – This function works only when SPI with CRC is selected.

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

/** Data buffers */
static uint8_t s_tx_buffer[RAA489206_TX_DATA_LENGTH] = {0};
static uint8_t s_rx_buffer[RAA489206_RX_DATA_LENGTH] = {0};
static uint8_t s_data_buffer[RAA489206_RX_DATA_LENGTH] = {0};

/** Read and compare the first batch of configuration registers. */
bfe_err = raa489206_cmnd_validate(p_ctrl, &p_ext_ctrl->p_bfe_cmnds->all_registers,
                                   &p_bfe_regs->global_op, RAA489206_CFG_REG_NUM_1,
                                   &p_bfe_regs->pri_faults_mask, RAA489206_CFG_REG_NUM_2);
BFE_ERR_RETURN_FALSE(BFE_SUCCESS == bfe_err, bfe_err); ///< Check for errors.
```

Important: *raa489206_reg_read()*, *raa489206_reg_write()*, *raa489206_reg_validate()*, *raa489206_cmnd_read()*, and *raa489206_cmnd_validate()* are not global functions. They can be used for custom code development. If working directly with registers or commands, consider using the *R_RAA489206_RegisterRead()* or *R_RAA489206_RegisterWrite()* API functions.

4. Revision History

Revision	Date	Description
1.00	Mar 20, 2025	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 who are 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 to develop 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 from 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.

(Disclaimer Rev.1.01)

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-us/.

Trademarks

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