Battery Cell Controller Software Driver v1.1
(Lite version)

Fri Jun 14 2019

Table of Contents

Module Index

Modules

Here is a list of all modules:

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

File Index

File List

Here is a list of all documented files with brief descriptions:

Module Documentation

Enum_group

Enumerations

- enum **bcc_status_t** { **BCC_STATUS_SUCCESS** = 0U, **BCC_STATUS_SPI_INIT** = 1U, **BCC_STATUS_SPI_BUSY** = 2U, **BCC_STATUS_PARAM_RANGE** = 4U, **BCC_STATUS_CRC** = 5U, **BCC_STATUS_COM_TAG_ID** = 6U, **BCC_STATUS_COM_RC** = 7U, **BCC_STATUS_COM_TIMEOUT** = 8U, **BCC_STATUS_DIAG_FAIL** = 9U, **BCC_STATUS_EEPROM_ERROR** = 10U, **BCC_STATUS_EEPROM_PRESENT** = 11U, **BCC_STATUS_NULL_RESP** = 12U }
- *Error codes.* enum **bcc_cid_t** { **BCC_CID_UNASSIG** = 0U, **BCC_CID_DEV1** = 1U, **BCC_CID_DEV2** = 2U, **BCC_CID_DEV3** = 3U, **BCC_CID_DEV4** = 4U, **BCC_CID_DEV5** = 5U, **BCC_CID_DEV6** = 6U, **BCC_CID_DEV7** = 7U, **BCC_CID_DEV8** = 8U, **BCC_CID_DEV9** = 9U, **BCC_CID_DEV10** = 10U, **BCC_CID_DEV11** = 11U, **BCC_CID_DEV12** = 12U, **BCC_CID_DEV13** = 13U, **BCC_CID_DEV14** = 14U, **BCC_CID_DEV15** = 15U }
- *Cluster Identification Address.* enum **bcc_mode_t** { **BCC_MODE_SPI** = 0U, **BCC_MODE_TPL** = 1U }
- *BCC communication mode.* enum **bcc_device_t** { **BCC_DEVICE_MC33771** = 0U, **BCC_DEVICE_MC33772** = 1U }
- *BCC device.* enum **bcc_measurements_t** { **BCC_MSR_CC_NB_SAMPLES** = 0U, **BCC_MSR_COULOMB_CNT1** = 1U, **BCC_MSR_COULOMB_CNT2** = 2U, **BCC_MSR_ISENSE1** = 3U, **BCC_MSR_ISENSE2** = 4U, **BCC_MSR_STACK_VOLT** = 5U, **BCC_MSR_CELL_VOLT14** = 6U, **BCC_MSR_CELL_VOLT13** = 7U, **BCC_MSR_CELL_VOLT12** = 8U, **BCC_MSR_CELL_VOLT11** = 9U, **BCC_MSR_CELL_VOLT10** = 10U, **BCC_MSR_CELL_VOLT9** = 11U, **BCC_MSR_CELL_VOLT8** = 12U, **BCC_MSR_CELL_VOLT7** = 13U, **BCC_MSR_CELL_VOLT6** = 14U, **BCC_MSR_CELL_VOLT5** = 15U, **BCC_MSR_CELL_VOLT4** = 16U, **BCC_MSR_CELL_VOLT3** = 17U, **BCC_MSR_CELL_VOLT2** = 18U, **BCC_MSR_CELL_VOLT1** = 19U, **BCC_MSR_AN6** = 20U, **BCC_MSR_AN5** = 21U, **BCC_MSR_AN4** = 22U, **BCC_MSR_AN3** = 23U, **BCC_MSR_AN2** = 24U, **BCC_MSR_AN1** = 25U, **BCC_MSR_AN0** = 26U, **BCC_MSR_ICTEMP** = 27U, **BCC_MSR_VBGADC1A** = 28U, **BCC_MSR_VBGADC1B** = 29U }
- *Measurements provided by Battery Cell Controller.* enum **bcc_fault_status_t** { **BCC_FS_CELL_OV** = 0U, **BCC_FS_CELL_UV** = 1U, **BCC_FS_CB_OPEN** = 2U, **BCC_FS_CB_SHORT** = 3U, **BCC_FS_GPIO_STATUS** = 4U, **BCC_FS_AN_OT_UT** = 5U, **BCC_FS_GPIO_SHORT** = 6U, **BCC_FS_COMM** = 7U, **BCC_FS_FAULT1** = 8U, **BCC_FS_FAULT2** = 9U, **BCC_FS_FAULT3** = 10U }

Status provided by Battery Cell Controller.

Detailed Description

Enumeration Type Documentation

enum bcc_cid_t

Cluster Identification Address.

Note that SPI communication mode uses one cluster/device only. The maximum number of clusters/devices for TPL mode is 15.

Enumerator:

BCC_CID_UNASSIG	ID of uninitialized BCC device.
BCC_CID_DEV1	Cluster ID of device 1. In TPL mode, this is the first device in daisy chain (connected directly to MC33664).
BCC_CID_DEV2	Cluster ID of device 2.
BCC_CID_DEV3	Cluster ID of device 3.
BCC_CID_DEV4	Cluster ID of device 4.
BCC_CID_DEV5	Cluster ID of device 5.
BCC_CID_DEV6	Cluster ID of device 6.
BCC_CID_DEV7	Cluster ID of device 7.
BCC_CID_DEV8	Cluster ID of device 8.
BCC_CID_DEV9	Cluster ID of device 9.
BCC_CID_DEV10	Cluster ID of device 10.
BCC_CID_DEV11	Cluster ID of device 11.
BCC_CID_DEV12	Cluster ID of device 12.
BCC_CID_DEV13	Cluster ID of device 13.
BCC_CID_DEV14	Cluster ID of device 14.
BCC_CID_DEV15	Cluster ID of device 15.

enum bcc_device_t

BCC device.

Enumerator:

BCC_DEVICE_MC33771	MC33771B.
BCC_DEVICE_MC33772	MC33772B.

enum bcc_fault_status_t

Status provided by Battery Cell Controller.

Enumerator:

BCC_FS_CELL_OV	CT overvoltage fault (register CELL_OV_FLT).
BCC_FS_CELL_UV	CT undervoltage fault (register CELL_UV_FLT).
BCC_FS_CB_OPEN	Open CB fault (register CB_OPEN_FLT).
BCC_FS_CB_SHORT	Short CB fault (register CB_SHORT_FLT).
BCC_FS_GPIO_STATUS	GPIO status (register GPIO_STS).
BCC_FS_AN_OT_UT	AN undertemperature and overtemperature (register AN_OT_UT_FLT).
BCC_FS_GPIO_SHORT	GPIO short and analog inputs open load detection (register GPIO_SHORT_Anx_OPEN_STS).
BCC_FS_COMM	Number of communication errors detected (register COM_STATUS).
BCC_FS_FAULT1	Fault status (register FAULT1_STATUS).
BCC_FS_FAULT2	Fault status (register FAULT2_STATUS).
BCC_FS_FAULT3	Fault status (register FAULT3_STATUS).

enum bcc_measurements_t

Measurements provided by Battery Cell Controller.

Note that MC33772 doesn't have MEAS_CELL7, ..., MEAS_CELL14 registers. Function BCC_Meas_GetRawValues returns 0x0000 at these positions.

Enumerator:

BCC_MSR_CC_NB_SAMPLES	Number of samples in Coulomb counter (register CC_NB_SAMPLES).
BCC_MSR_COULOMB_CNT1	Coulomb counting accumulator (register COULOMB__CNT1).
BCC_MSR_COULOMB_CNT2	Coulomb counting accumulator (register COULOMB__CNT2).
BCC_MSR_ISENSE1	ISENSE measurement (register MEAS_ISENSE1).
BCC_MSR_ISENSE2	ISENSE measurement (register MEAS_ISENSE2).

BCC_MSR_STACK_VOLT	Stack voltage measurement (register MEAS_STACK).
BCC_MSR_CELL_VOLT14	Cell 14 voltage measurement (register MEAS_CELL14).
BCC_MSR_CELL_VOLT13	Cell 13 voltage measurement (register MEAS_CELL13).
BCC_MSR_CELL_VOLT12	Cell 12 voltage measurement (register MEAS_CELL12).
BCC_MSR_CELL_VOLT11	Cell 11 voltage measurement (register MEAS_CELL11).
BCC_MSR_CELL_VOLT10	Cell 10 voltage measurement (register MEAS_CELL10).
BCC_MSR_CELL_VOLT9	Cell 9 voltage measurement (register MEAS_CELL9).
BCC_MSR_CELL_VOLT8	Cell 8 voltage measurement (register MEAS_CELL8).
BCC_MSR_CELL_VOLT7	Cell 7 voltage measurement (register MEAS_CELL7).
BCC_MSR_CELL_VOLT6	Cell 6 voltage measurement (register MEAS_CELL6).
BCC_MSR_CELL_VOLT5	Cell 5 voltage measurement (register MEAS_CELL5).
BCC_MSR_CELL_VOLT4	Cell 4 voltage measurement (register MEAS_CELL4).
BCC_MSR_CELL_VOLT3	Cell 3 voltage measurement (register MEAS_CELL3).
BCC_MSR_CELL_VOLT2	Cell 2 voltage measurement (register MEAS_CELL2).
BCC_MSR_CELL_VOLT1	Cell 1 voltage measurement (register MEAS_CELL1).
BCC_MSR_AN6	Analog input 6 voltage measurement (register MEAS_AN6).
BCC_MSR_AN5	Analog input 5 voltage measurement (register MEAS_AN5).
BCC_MSR_AN4	Analog input 4 voltage measurement (register MEAS_AN4).
BCC_MSR_AN3	Analog input 3 voltage measurement (register MEAS_AN3).
BCC_MSR_AN2	Analog input 2 voltage measurement (register MEAS_AN2).
BCC_MSR_AN1	Analog input 1 voltage measurement (register MEAS_AN1).
BCC_MSR_AN0	Analog input 0 voltage measurement (register MEAS_AN0).
BCC_MSR_ICTEMP	IC temperature measurement (register MEAS_IC_TEMP).
BCC_MSR_VBGADC1A	ADCIA Band Gap Reference measurement (register

	MEAS_VBG_DIAG_ADC1A).
BCC_MSR_VBG ADC1B	ADCIB Band Gap Reference measurement (register MEAS_VBG_DIAG_ADC1B).

enum bcc_mode_t

BCC communication mode.

Enumerator:

BCC_MODE_SPI	SPI communication mode.
BCC_MODE_TPL	TPL communication mode.

enum bcc_status_t

Error codes.

Enumerator:

BCC_STATUS_SUCCESS	No error.
BCC_STATUS_SPI_INIT	SPI initialization failure.
BCC_STATUS_SPI_BUSY	SPI instance is busy.
BCC_STATUS_PARAM_RANGE	Parameter out of range.
BCC_STATUS_CRC	Wrong CRC.
BCC_STATUS_COM_TAG_ID	Response Tag ID does not match with provided ID.
BCC_STATUS_COM_RC	Response Rolling Counter (RC) value does not match with expected RC.
BCC_STATUS_COM_TIMEOUT	Communication timeout.
BCC_STATUS_DIAG_FAIL	It is not allowed to enter diagnostic mode.
BCC_STATUS_EEPROM_ERROR	An error occurred during the communication to EEPROM.
BCC_STATUS_EEPROM_PRESENT	No EEPROM detected.
BCC_STATUS_NULL_RESP	Response frame of BCC device is equal to zero (except CRC). This occurs only in SPI communication mode during the very first message.

Struct_group

Data Structures

- struct **bcc_drv_data_t**
  Driver internal data.
- struct **bcc_drv_config_t**
  Driver configuration.

Detailed Description

Function_group

Functions

- **bcc_status_t BCC_Init** (**bcc_drv_config_t** *const drvConfig, const uint16_t devConf[][**BCC_INIT_CONF_REG_CNT**])
  This function initializes the Battery Cell Controller device(s), configures its registers, assigns CID and initializes internal driver data.
- **bcc_status_t BCC_VerifyCom** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function uses No Operation command of BCC to verify communication with device specified by CID without performing any operation.
- **bcc_status_t BCC_Sleep** (**bcc_drv_config_t** *const drvConfig)
  This function sets sleep mode to all Battery Cell Controller devices.
- void **BCC_WakeUp** (const **bcc_drv_config_t** *const drvConfig)
  This function sets normal mode to all Battery Cell Controller devices.
- **bcc_status_t BCC_SoftwareReset** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function resets BCC device using software reset. It enters reset via SPI or TPL interface.
- void **BCC_HardwareReset** (const **bcc_drv_config_t** *const drvConfig)
  This function resets BCC device using GPIO pin.
- **bcc_status_t BCC_TPL_Enable** (const **bcc_drv_config_t** *const drvConfig)
  This function enables MC33664 device (sets a normal mode). Intended for TPL mode only!
- void **BCC_TPL_Disable** (const **bcc_drv_config_t** *const drvConfig)
  This function sets MC33664 device into sleep mode. Intended for TPL mode only!
- **bcc_status_t BCC_Reg_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint8_t regCnt, uint16_t *regVal)
  This function reads a value from addressed register (or desired number of registers) of selected Battery Cell Controller device.
- **bcc_status_t BCC_Reg_Write** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regVal, uint16_t *retReg)
  This function writes a value to addressed register of selected Battery Cell Controller device.
- **bcc_status_t BCC_Reg_WriteGlobal** (**bcc_drv_config_t** *const drvConfig, uint8_t regAddr, uint16_t regVal)
  This function writes a value to addressed register of all configured BCC devices. Intended for TPL mode only!
- **bcc_status_t BCC_Reg_Update** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regMask, uint16_t regVal)

This function updates content of a selected register. It affects bits specified by a bit mask only.

- **bcc_status_t BCC_Meas_StartConversion** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function starts ADC conversion. It sets Start of Conversion bit and new value of TAG ID in ADC_CFG register.
- **bcc_status_t BCC_Meas_StartConversionGlobal** (**bcc_drv_config_t** *const drvConfig, uint16_t adcCfgValue)
  This function starts ADC conversion for all devices in TPL chain. It uses a Global Write command to set ADC_CFG register. Intended for TPL mode only!
- **bcc_status_t BCC_Meas_IsConverting** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, bool *completed)
  This function checks status of conversion defined by End of Conversion bit in ADC_CFG register.
- **bcc_status_t BCC_Meas_GetRawValues** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint16_t measurements[])
  This function reads the measurement registers and returns raw values. Macros defined in BCC header file can be used to perform correct unit conversion.
- **bcc_status_t BCC_Fault_GetStatus** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint16_t status[])
  *This function reads the status registers and returns raw values. You can use constants defined in **bcc_mc3377x.h** file.*
- **bcc_status_t BCC_Fault_ClearStatus** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, **bcc_fault_status_t** statSel)
  This function clears selected fault status register.
- **bcc_status_t BCC_GPIO_SetOutput** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t gpioSel, bool val)
  This function sets output value of one BCC GPIO pin. This function should be used only when at least one GPIO is in output mode. Resets BCC device using GPIO pin.
- **bcc_status_t BCC_CB_Enable** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, bool enable)
  This function enables or disables the cell balancing via SYS_CFG1[CB_DRVEN] bit.
- **bcc_status_t BCC_CB_SetIndividual** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t cellIndex, bool enable, uint16_t timer)
  This function enables or disables cell balancing for a specified cell and sets its timer.
- **bcc_status_t BCC_CB_Pause** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, bool pause)
  This function can be used to manual pause cell balancing before on demand conversion. As a result more precise measurement can be done. Note that it is user obligation to re-enable cell balancing after measurement ends.
- **bcc_status_t BCC_FuseMirror_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t fuseAddr, uint16_t *const value)
  This function reads a fuse mirror register of a BCC device specified by CID.
- **bcc_status_t BCC_FuseMirror_Write** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t fuseAddr, uint16_t value)
  This function writes a fuse mirror register of a BCC device specified by CID.
- **bcc_status_t BCC_GUID_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint64_t *const guid)
  This function reads an unique serial number of the BCC device from the content of mirror registers.
- **bcc_status_t BCC_EEPROM_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t addr, uint8_t *const data)
  This function reads a byte from specified address of EEPROM memory connected to BCC device via I2C bus.
- **bcc_status_t BCC_EEPROM_Write** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t addr, uint8_t data)

This function writes a byte to specified address of EEPROM memory connected to BCC device via I2C bus.

- void **BCC_MCU_WaitMs** (uint16_t delay)
  Waits for specified amount of milliseconds. This function needs to be implemented for specified MCU by the user.
- void **BCC_MCU_WaitUs** (uint32_t delay)
  Waits for specified amount of microseconds. This function needs to be implemented for specified MCU by the user.
- void **BCC_MCU_Assert** (bool x)
  User implementation of assert.
- **bcc_status_t BCC_MCU_TransferSpi** (uint8_t drvInstance, uint8_t transBuf[], uint8_t recvBuf[])
  This function performs one 40b transfer via SPI bus. Intended for SPI mode only. This function needs to be implemented for specified MCU by the user.
- **bcc_status_t BCC_MCU_TransferTpl** (uint8_t drvInstance, uint8_t transBuf[], uint8_t recvBuf[], uint16_t recvTrCnt)
  This function sends and receives data via TX and RX SPI buses. Intended for TPL mode only. This function needs to be implemented for specified MCU by the user.
- void **BCC_MCU_WriteCsbPin** (uint8_t drvInstance, uint8_t value)
  Writes logic 0 or 1 to the CSB pin (or CSB_TX in case of TPL mode). This function needs to be implemented by the user.
- void **BCC_MCU_WriteRstPin** (uint8_t drvInstance, uint8_t value)
  Writes logic 0 or 1 to the RST pin. This function needs to be implemented by the user. If no RST pin is used, keep the function body empty.
- void **BCC_MCU_WriteEnPin** (uint8_t drvInstance, uint8_t value)
  Writes logic 0 or 1 to the EN pin of MC33664. This function is called only in the TPL and it needs to be implemented by the user.
- uint32_t **BCC_MCU_ReadIntbPin** (uint8_t drvInstance)
  Reads logic value of INTB pin of MC33664. This function is called in the TPL mode only and it needs to be implemented by the user.
- void **BCC_PackFrame** (uint16_t data, uint8_t addr, **bcc_cid_t** cid, uint8_t cmd, uint8_t frame[])
  This function packs all the parameters into a frame according to the BCC frame format (see BCC datasheet).
- **bcc_status_t BCC_CheckCRC** (const uint8_t *resp)
  This function calculates CRC of a received frame and compares it with CRC field of the frame.
- **bcc_status_t BCC_CheckRcTagId** (**bcc_device_t** devType, const uint8_t *resp, uint8_t rc, uint8_t tagId)
  This function checks value of the Command field of a frame.
- **bcc_status_t BCC_Reg_ReadSpi** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint8_t regCnt, uint16_t *regVal)
  This function reads a value from addressed register of selected Battery Cell Controller device. Intended for SPI mode only.
- **bcc_status_t BCC_Reg_WriteSpi** (const **bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regVal, uint16_t *retReg)
  This function writes a value to addressed register of selected Battery Cell Controller device. Intended for SPI mode only.
- **bcc_status_t BCC_VerifyComSpi** (const **bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function uses No Operation command of BCC to verify communication without performing any operation. Intended for SPI mode only.
- **bcc_status_t BCC_Reg_ReadTpl** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint8_t regCnt, uint16_t *regVal)
  This function reads a value from addressed register of selected Battery Cell Controller device. Intended for TPL mode only.

- **bcc_status_t BCC_Reg_WriteTpl** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regVal, uint16_t *retReg)
  This function writes a value to addressed register of selected Battery Cell Controller device. Intended for TPL mode only.
- **bcc_status_t BCC_Reg_WriteGlobalTpl** (**bcc_drv_config_t** *const drvConfig, uint8_t regAddr, uint16_t regVal)
  This function writes a value to addressed register of all configured BCC devices. Intended for TPL mode only.
- **bcc_status_t BCC_VerifyComTpl** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function uses No Operation command of BCC to verify communication without performing any operation. Intended for TPL mode only.

Detailed Description

Function Documentation

bcc_status_t BCC_CB_Enable (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, bool *enable*)

This function enables or disables the cell balancing via SYS_CFG1[CB_DRVEN] bit.

Note that each cell balancing driver needs to be setup separately, e.g. by BCC_CB_SetIndividual function.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
enable	Drivers state. False (all drivers are disabled) or true (drivers are enabled).

Returns:

bcc_status_t Error code.

bcc_status_t BCC_CB_Pause (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, bool *pause*)

This function can be used to manual pause cell balancing before on demand conversion. As a result more precise measurement can be done. Note that it is user obligation to re-enable cell balancing after measurement ends.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
pause	True (pause) / false (unpause).

Returns:

bcc_status_t Error code.

bcc_status_t BCC_CB_SetIndividual (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *cellIndex*, bool *enable*, uint16_t *timer*)

This function enables or disables cell balancing for a specified cell and sets its timer.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
cellIndex	Index of the cell. Note the cells are indexed from 0.
enable	True for enabling of CB, False otherwise.
timer	Timer for enabled CB driver in minutes. Note that a zero value represents 30 seconds.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_CheckCRC (const uint8_t * *resp*)

This function calculates CRC of a received frame and compares it with CRC field of the frame.

Parameters:

resp	Pointer to memory that contains a response (frame) to be checked.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_CheckRcTagId (bcc_device_t *devType*, const uint8_t * *resp*, uint8_t *rc*, uint8_t *tagId*)

This function checks value of the Command field of a frame.

Parameters:

devType	Device type.
resp	Pointer to memory that contains a response (frame) to be checked.
rc	Expected value of Rolling Counter.
tagId	Expected value of TAG ID.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_EEPROM_Read (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *addr*, uint8_t *const *data*)

This function reads a byte from specified address of EEPROM memory connected to BCC device via I2C bus.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address of BCC device the EEPROM memory is connected to.
addr	Address of EEPROM data will be read from. The admission range is from 0 to 127.
data	Data read from specified address of EEPROM memory.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_EEPROM_Write (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *addr*, uint8_t *data*)

This function writes a byte to specified address of EEPROM memory connected to BCC device via I2C bus.

Note that the EEPROM write time (depends on device selection) is usually around 5 ms. Therefore, another EEPROM (write & read) operations to the same EEPROM memory cannot be done 5 ms after end of BCC_EEPROM_Write function.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address of BCC device the EEPROM memory is connected to.
addr	Address of EEPROM data will be written to. The admission range is from 0 to 127.
data	Data written to specified address of EEPROM memory.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Fault_ClearStatus (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, bcc_fault_status_t *statSel*)

This function clears selected fault status register.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
statSel	Selection of a fault status register to be cleared. See definition of this enumeration in BCC header file. COM_STATUS register is read only and cannot be cleared.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Fault_GetStatus (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint16_t *status*[])

This function reads the status registers and returns raw values. You can use constants defined in **bcc_mc3377x.h** file.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
status	Array containing all fault status information provided by BCC. Indexes into the array are defined in bcc_fault_status_t enumeration placed in BCC header file. Required size of the array is 11. You can use macro BCC_STAT_CNT defined in BCC header file, which contains appropriate value.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_FuseMirror_Read (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *fuseAddr*, uint16_t *const *value*)

This function reads a fuse mirror register of a BCC device specified by CID.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
fuseAddr	Address of a fuse mirror register to be read.
value	Pointer to memory where the read value will be stored.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_FuseMirror_Write (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *fuseAddr*, uint16_t *value*)

This function writes a fuse mirror register of a BCC device specified by CID.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
fuseAddr	Address of a fuse mirror register to be written.
value	Value to be written.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_GPIO_SetOutput (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *gpioSel*, bool *val*)

This function sets output value of one BCC GPIO pin. This function should be used only when at least one GPIO is in output mode. Resets BCC device using GPIO pin.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
gpioSel	Index of GPIO output to be set. Index starts at 0 (GPIO 0).
val	Output value. Possible values are FALSE (logical 0, low level) and TRUE (logical 1, high level).

Returns:

bcc_status_t Error code.

bcc_status_t BCC_GUID_Read (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint64_t *const *guid*)

This function reads an unique serial number of the BCC device from the content of mirror registers.

GUID is created according to the following table:

Device	GUID [36:21]	GUID [20:5]	GUID [4:0]
MC33771B	0x18 [15:0]	0x19 [15:0]	0xA1 [4:0]
fuse address	(16 bit)	(16 bit)	(5 bit)
:----------—:	:----------—:	:----------—:	:---------—:
MC33772B	0x10 [15:0]	0x11 [15:0]	0x12 [4:0]
fuse address	(16 bit)	(16 bit)	(5 bit)

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
guid	Pointer to memory where 37b unique ID will be stored.

Returns:

bcc_status_t Error code.

void BCC_HardwareReset (const bcc_drv_config_t *const *drvConfig*)

This function resets BCC device using GPIO pin.

Parameters:

drvConfig	Pointer to driver instance configuration.

bcc_status_t BCC_Init (bcc_drv_config_t *const *drvConfig*, const uint16_t *devConf*[][BCC_INIT_CONF_REG_CNT])

This function initializes the Battery Cell Controller device(s), configures its registers, assigns CID and initializes internal driver data.

Parameters:

drvConfig	Pointer to driver instance configuration.
devConf	Initialization values of BCC device registers specified by BCC_INIT_CONF_REG_ADDR. If NULL, registers are not initialized. devConf[0][x] belongs to device with CID 1, devConf[1][x] belongs to device with CID 2, etc.

Returns:

bcc_status_t Error code.

void BCC_MCU_Assert (bool *x*)

User implementation of assert.

Parameters:

x	- True if everything is OK.

uint32_t BCC_MCU_ReadIntbPin (uint8_t *drvInstance*)

Reads logic value of INTB pin of MC33664. This function is called in the TPL mode only and it needs to be implemented by the user.

Parameters:

drvInstance	Instance of BCC driver.

Returns:

Zero value for logic zero, non-zero value otherwise.

bcc_status_t BCC_MCU_TransferSpi (uint8_t *drvInstance*, uint8_t *transBuf*[], uint8_t *recvBuf*[])

This function performs one 40b transfer via SPI bus. Intended for SPI mode only. This function needs to be implemented for specified MCU by the user.

The byte order of buffers is given by BCC_MSG_BIGEND macro (in **bcc.h**).

Parameters:

drvInstance	Instance of BCC driver.
transBuf	Pointer to 40b data buffer to be sent.
recvBuf	Pointer to 40b data buffer for received data.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_MCU_TransferTpl (uint8_t *drvInstance*, uint8_t *transBuf*[], uint8_t *recvBuf*[], uint16_t *recvTrCnt*)

This function sends and receives data via TX and RX SPI buses. Intended for TPL mode only. This function needs to be implemented for specified MCU by the user.

TX SPI bus always performs only one 40b SPI transfer. Expected number of RX transfers is passed as a parameter. The byte order of buffers is given by BCC_MSG_BIGEND macro (in **bcc.h**).

Parameters:

drvInstance	Instance of BCC driver.
transBuf	Pointer to 40b data buffer to be sent.
recvBuf	Pointer to buffer for received data. Its size must be at least (5 * recvTrCnt) bytes.
recvTrCnt	Number of 40b transfers to be received.

Returns:

bcc_status_t Error code.

void BCC_MCU_WaitMs (uint16_t *delay*)

Waits for specified amount of milliseconds. This function needs to be implemented for specified MCU by the user.

Parameters:

delay	- Number of milliseconds to wait.

void BCC_MCU_WaitUs (uint32_t *delay*)

Waits for specified amount of microseconds. This function needs to be implemented for specified MCU by the user.

Parameters:

delay	- Number of microseconds to wait.

void BCC_MCU_WriteCsbPin (uint8_t *drvInstance*, uint8_t *value*)

Writes logic 0 or 1 to the CSB pin (or CSB_TX in case of TPL mode). This function needs to be implemented by the user.

Parameters:

drvInstance	Instance of BCC driver.
value	- Zero or one to be set to CSB (CSB_TX) pin.

void BCC_MCU_WriteEnPin (uint8_t *drvInstance*, uint8_t *value*)

Writes logic 0 or 1 to the EN pin of MC33664. This function is called only in the TPL and it needs to be implemented by the user.

Parameters:

drvInstance	Instance of BCC driver.
value	- Zero or one to be set to EN pin.

void BCC_MCU_WriteRstPin (uint8_t *drvInstance*, uint8_t *value*)

Writes logic 0 or 1 to the RST pin. This function needs to be implemented by the user. If no RST pin is used, keep the function body empty.

Parameters:

drvInstance	Instance of BCC driver.
value	- Zero or one to be set to RST pin.

bcc_status_t BCC_Meas_GetRawValues (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint16_t *measurements*[])

This function reads the measurement registers and returns raw values. Macros defined in BCC header file can be used to perform correct unit conversion.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
measurements	Array containing all values measured by BCC. Indexes into the array are defined in enumeration bcc_measurements_t placed in BCC header file. For required size of the array see BCC_MEAS_CNT constant defined in BCC header file).

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Meas_IsConverting (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, bool * *completed*)

This function checks status of conversion defined by End of Conversion bit in ADC_CFG register.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.

completed	Pointer to check result. True if a conversion is complete.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Meas_StartConversion (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*)

This function starts ADC conversion. It sets Start of Conversion bit and new value of TAG ID in ADC_CFG register.

TAG ID is incremented for each conversion. You can use function BCC_Meas_IsConverting to check conversion status.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Meas_StartConversionGlobal (bcc_drv_config_t *const *drvConfig*, uint16_t *adcCfgValue*)

This function starts ADC conversion for all devices in TPL chain. It uses a Global Write command to set ADC_CFG register. Intended for TPL mode only!

As a TAG ID, incremented TAG ID of the first device is used. You can use function BCC_Meas_IsConverting to check conversion status.

Parameters:

drvConfig	Pointer to driver instance configuration.
adcCfgValue	Value of ADC_CFG register to be written to all devices in the chain. Note that TAG_ID and SOC bits are automatically added by this function.

Returns:

bcc_status_t Error code.

void BCC_PackFrame (uint16_t *data*, uint8_t *addr*, bcc_cid_t *cid*, uint8_t *cmd*, uint8_t *frame*[])

This function packs all the parameters into a frame according to the BCC frame format (see BCC datasheet).

Note the frame is packed in Little-endian because of the LPSPI periphery.

Parameters:

data	16 bit Memory Data field of the BCC frame.
addr	7 bit Memory Address field of the BCC frame.
cid	4 bit Physical Address field of the BCC frame.
cmd	4 bit Command field of the BCC frame.
frame	Pointer to a 5-byte array where all the fields and computed CRC will be stored. Note that fields are stored into an array in reverse order (due to processing order in the LPSPI master and slave drivers).

bcc_status_t BCC_Reg_Read (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *regAddr*, uint8_t *regCnt*, uint16_t * *regVal*)

This function reads a value from addressed register (or desired number of registers) of selected Battery Cell Controller device.

In case of simultaneous read of more registers, address is incremented in ascending manner.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
regAddr	Register address. See BCC header file with register map for possible values.
regCnt	Number of registers to read.
regVal	Pointer to memory where content of selected 16 bit registers is stored.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Reg_ReadSpi (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *regAddr*, uint8_t *regCnt*, uint16_t * *regVal*)

This function reads a value from addressed register of selected Battery Cell Controller device. Intended for SPI mode only.

In case of simultaneous read of more registers, address is incremented in ascending manner.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
regAddr	Register address. See BCC header file with register map for possible values.
regCnt	Number of registers to read.
regVal	Pointer to memory where content of selected 16 bit registers is stored.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Reg_ReadTpl (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *regAddr*, uint8_t *regCnt*, uint16_t * *regVal*)

This function reads a value from addressed register of selected Battery Cell Controller device. Intended for TPL mode only.

In case of simultaneous read of more registers, address is incremented in ascending manner.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
regAddr	Register address. See BCC header file with register map for possible values.
regCnt	Number of registers to read.
regVal	Pointer to memory where content of selected 16 bit registers is stored.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Reg_Update (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *regAddr*, uint16_t *regMask*, uint16_t *regVal*)

This function updates content of a selected register. It affects bits specified by a bit mask only.

Parameters:

drvConfig	Pointer to driver instance configuration.

cid	Cluster Identification Address.
regAddr	Register address. See BCC header file with register map for possible values.
regMask	Bit mask. Bits set to 1 will be updated.
regVal	New value of register bits defined by bit mask.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Reg_Write (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *regAddr*, uint16_t *regVal*, uint16_t * *retReg*)

This function writes a value to addressed register of selected Battery Cell Controller device.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
regAddr	Register address. See BCC header file with register map for possible values.
regVal	New value of selected register.
retReg	Automatic response of BCC, which contains updated register (TPL mode) or a register addressed in previous access (SPI mode). You can pass NULL when you do not care about the response.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Reg_WriteGlobal (bcc_drv_config_t *const *drvConfig*, uint8_t *regAddr*, uint16_t *regVal*)

This function writes a value to addressed register of all configured BCC devices. Intended for TPL mode only!

Parameters:

drvConfig	Pointer to driver instance configuration.
regAddr	Register address. See BCC header file with register map for possible values.
regVal	New value of selected register.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Reg_WriteGlobalTpl (bcc_drv_config_t *const *drvConfig*, uint8_t *regAddr*, uint16_t *regVal*)

This function writes a value to addressed register of all configured BCC devices. Intended for TPL mode only.

Parameters:

drvConfig	Pointer to driver instance configuration.
regAddr	Register address. See BCC header file with register map for possible values.
regVal	New value of selected register.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Reg_WriteSpi (const bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *regAddr*, uint16_t *regVal*, uint16_t * *retReg*)

This function writes a value to addressed register of selected Battery Cell Controller device. Intended for SPI mode only.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
regAddr	Register address. See BCC header file with register map for possible values.
regVal	New value of selected register.
retReg	Automatic response of BCC, which contains register addressed in previous access. You can pass NULL when you do not care about the response.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Reg_WriteTpl (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*, uint8_t *regAddr*, uint16_t *regVal*, uint16_t * *retReg*)

This function writes a value to addressed register of selected Battery Cell Controller device. Intended for TPL mode only.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
regAddr	Register address. See BCC header file with register map for possible values.
regVal	New value of selected register.
retReg	Automatic response of BCC, which contains updated register. You can pass NULL when you do not care about the response.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_Sleep (bcc_drv_config_t *const *drvConfig*)

This function sets sleep mode to all Battery Cell Controller devices.

In case of TPL mode MC33664TL goes to sleep mode automatically.

Parameters:

drvConfig	Pointer to driver instance configuration.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_SoftwareReset (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*)

This function resets BCC device using software reset. It enters reset via SPI or TPL interface.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.

Returns:

bcc_status_t Error code.

void BCC_TPL_Disable (const bcc_drv_config_t *const *drvConfig*)

This function sets MC33664 device into sleep mode. Intended for TPL mode only!

This function can be (optionally) used after BCC_Sleep function. Function BCC_TPL_Enable must be then called before BCC_WakeUp function!

Parameters:

drvConfig	Pointer to driver instance configuration.

bcc_status_t BCC_TPL_Enable (const bcc_drv_config_t *const *drvConfig*)

This function enables MC33664 device (sets a normal mode). Intended for TPL mode only!

During the driver initialization (BCC_Init function), normal mode of MC33664 is set automatically. This function can be used e.g. when sleep mode of both BCC device(s) and MC33664 is required. The typical function flow can be then: BCC_Sleep -> BCC_TPL_Disable -> ... -> BCC_TPL_Enable -> BCC_WakeUp.

Parameters:

drvConfig	Pointer to driver instance configuration.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_VerifyCom (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*)

This function uses No Operation command of BCC to verify communication with device specified by CID without performing any operation.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_VerifyComSpi (const bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*)

This function uses No Operation command of BCC to verify communication without performing any operation. Intended for SPI mode only.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.

Returns:

bcc_status_t Error code.

bcc_status_t BCC_VerifyComTpl (bcc_drv_config_t *const *drvConfig*, bcc_cid_t *cid*)

This function uses No Operation command of BCC to verify communication without performing any operation. Intended for TPL mode only.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.

Returns:

bcc_status_t Error code.

void BCC_WakeUp (const bcc_drv_config_t *const *drvConfig*)

This function sets normal mode to all Battery Cell Controller devices.

In case of TPL mode, MC33664 goes to normal mode automatically.

Parameters:

drvConfig	Pointer to driver instance configuration.

Returns:

bcc_status_t Error code.

Data Structure Documentation

bcc_drv_config_t Struct Reference

Driver configuration.
`#include <bcc.h>`

Data Fields

- uint8_t **drvInstance**
- **bcc_mode_t commMode**
- uint8_t **devicesCnt**
- **bcc_device_t device** [**BCC_DEVICE_CNT_MAX**]
- uint16_t **cellCnt** [**BCC_DEVICE_CNT_MAX**]
- **bcc_drv_data_t drvData**

Detailed Description

Driver configuration.

This structure contains all information needed for proper functionality of the driver, such as used communication mode, BCC device(s) configuration or internal driver data.

Field Documentation

uint16_t bcc_drv_config_t::cellCnt[BCC_DEVICE_CNT_MAX]

Number of connected cells to each BCC. [0] BCC with CID 1, [1] BCC with CID 2, etc.

bcc_mode_t bcc_drv_config_t::commMode

BCC communication mode.

bcc_device_t bcc_drv_config_t::device[BCC_DEVICE_CNT_MAX]

BCC device type of [0] BCC with CID 1, [1] BCC with CID 2, etc.

uint8_t bcc_drv_config_t::devicesCnt

Number of BCC devices. SPI mode allows one device only, TPL mode allows up to 15 devices.

bcc_drv_data_t bcc_drv_config_t::drvData

Internal driver data.

uint8_t bcc_drv_config_t::drvInstance

BCC driver instance. Passed to the external functions defined by the user.

The documentation for this struct was generated from the following file:

- bcc/**bcc.h**

bcc_drv_data_t Struct Reference

Driver internal data.

`#include <bcc.h>`

Data Fields

- uint16_t **cellMap** [**BCC_DEVICE_CNT_MAX**]
- uint8_t **rcTbl** [**BCC_DEVICE_CNT_MAX**]
- uint8_t **tagId** [**BCC_DEVICE_CNT_MAX**]
- uint8_t **rxBuf** [**BCC_RX_BUF_SIZE_TPL**]

Detailed Description

Driver internal data.

Note that it is initialized in BCC_Init function by the driver and the user mustn't change it at any time.

Field Documentation

uint16_t bcc_drv_data_t::cellMap[BCC_DEVICE_CNT_MAX]

Bit map of used cells of each BCC device.

uint8_t bcc_drv_data_t::rcTbl[BCC_DEVICE_CNT_MAX]

Rolling counter index (0-4).

uint8_t bcc_drv_data_t::rxBuf[BCC_RX_BUF_SIZE_TPL]

Buffer for receiving data in TPL mode.

uint8_t bcc_drv_data_t::tagId[BCC_DEVICE_CNT_MAX]

TAG IDs of BCC devices.

The documentation for this struct was generated from the following file:

- bcc/**bcc.h**

File Documentation

bcc/bcc.c File Reference

```
#include "bcc_spi.h"
#include "bcc_tpl.h"
```

Macros

- #define **BCC_CM_MC33771_7CELLS** 0x380FU
  Cell map for 7 cells connected to MC33771.
- #define **BCC_CM_MC33772_3CELLS** 0x0023U
  Cell map for 3 cells connected to MC33772.
- #define **BCC_T_VPWR_READY_MS** 5U
  Time after VPWR connection for the IC to be ready for initialization (t_VPWR(READY)) in [ms].
- #define **BCC_T_WAKE_T1_US** 21U
  CSB_TX LOW period in CSB_TX wake-up pulse sequence (t_1, typ.) in [us].
- #define **BCC_T_WAKE_T2_US** 600U
  CSB_TX HIGH period in CSB_TX wake-up pulse sequence (t_2, typ.) in [us].
- #define **BCC_T_INTB_PULSE_DELAY_US** 100U
  EN LOW to HIGH transition to INTB verification pulse (t_INTB_PULSE_DELAY, maximum) in [us].
- #define **BCC_T_INTB_PULSE_US** 100U
  INTB verification pulse duration (t_INTB_PULSE, typ.) in [us].
- #define **BCC_T_RESETFLT_US** 100U
  RESET de-glitch filter (t_RESETFLT, typ.) in [us].

Functions

- **bcc_status_t BCC_Init** (**bcc_drv_config_t** *const drvConfig, const uint16_t devConf[][**BCC_INIT_CONF_REG_CNT**])
  This function initializes the Battery Cell Controller device(s), configures its registers, assigns CID and initializes internal driver data.
- **bcc_status_t BCC_VerifyCom** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function uses No Operation command of BCC to verify communication with device specified by CID without performing any operation.
- **bcc_status_t BCC_Sleep** (**bcc_drv_config_t** *const drvConfig)
  This function sets sleep mode to all Battery Cell Controller devices.
- void **BCC_WakeUp** (const **bcc_drv_config_t** *const drvConfig)
  This function sets normal mode to all Battery Cell Controller devices.
- **bcc_status_t BCC_SoftwareReset** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function resets BCC device using software reset. It enters reset via SPI or TPL interface.
- void **BCC_HardwareReset** (const **bcc_drv_config_t** *const drvConfig)
  This function resets BCC device using GPIO pin.
- **bcc_status_t BCC_TPL_Enable** (const **bcc_drv_config_t** *const drvConfig)
  This function enables MC33664 device (sets a normal mode). Intended for TPL mode only!
- void **BCC_TPL_Disable** (const **bcc_drv_config_t** *const drvConfig)
  This function sets MC33664 device into sleep mode. Intended for TPL mode only!
- **bcc_status_t BCC_Reg_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint8_t regCnt, uint16_t *regVal)
  This function reads a value from addressed register (or desired number of registers) of selected Battery Cell Controller device.

- **bcc_status_t BCC_Reg_Write** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regVal, uint16_t *retReg)
  This function writes a value to addressed register of selected Battery Cell Controller device.
- **bcc_status_t BCC_Reg_WriteGlobal** (**bcc_drv_config_t** *const drvConfig, uint8_t regAddr, uint16_t regVal)
  This function writes a value to addressed register of all configured BCC devices. Intended for TPL mode only!
- **bcc_status_t BCC_Reg_Update** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regMask, uint16_t regVal)
  This function updates content of a selected register. It affects bits specified by a bit mask only.
- **bcc_status_t BCC_Meas_StartConversion** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function starts ADC conversion. It sets Start of Conversion bit and new value of TAG ID in ADC_CFG register.
- **bcc_status_t BCC_Meas_StartConversionGlobal** (**bcc_drv_config_t** *const drvConfig, uint16_t adcCfgValue)
  This function starts ADC conversion for all devices in TPL chain. It uses a Global Write command to set ADC_CFG register. Intended for TPL mode only!
- **bcc_status_t BCC_Meas_IsConverting** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, bool *completed)
  This function checks status of conversion defined by End of Conversion bit in ADC_CFG register.
- **bcc_status_t BCC_Meas_GetRawValues** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint16_t measurements[])
  This function reads the measurement registers and returns raw values. Macros defined in BCC header file can be used to perform correct unit conversion.
- **bcc_status_t BCC_Fault_GetStatus** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint16_t status[])
  *This function reads the status registers and returns raw values. You can use constants defined in **bcc_mc3377x.h** file.*
- **bcc_status_t BCC_Fault_ClearStatus** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, **bcc_fault_status_t** statSel)
  This function clears selected fault status register.
- **bcc_status_t BCC_GPIO_SetOutput** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t gpioSel, bool val)
  This function sets output value of one BCC GPIO pin. This function should be used only when at least one GPIO is in output mode. Resets BCC device using GPIO pin.
- **bcc_status_t BCC_CB_Enable** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, bool enable)
  This function enables or disables the cell balancing via SYS_CFG1[CB_DRVEN] bit.
- **bcc_status_t BCC_CB_SetIndividual** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t cellIndex, bool enable, uint16_t timer)
  This function enables or disables cell balancing for a specified cell and sets its timer.
- **bcc_status_t BCC_CB_Pause** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, bool pause)
  This function can be used to manual pause cell balancing before on demand conversion. As a result more precise measurement can be done. Note that it is user obligation to re-enable cell balancing after measurement ends.
- **bcc_status_t BCC_FuseMirror_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t fuseAddr, uint16_t *const value)
  This function reads a fuse mirror register of a BCC device specified by CID.
- **bcc_status_t BCC_FuseMirror_Write** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t fuseAddr, uint16_t value)
  This function writes a fuse mirror register of a BCC device specified by CID.
- **bcc_status_t BCC_GUID_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint64_t *const guid)

This function reads an unique serial number of the BCC device from the content of mirror registers.

- **bcc_status_t BCC_EEPROM_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t addr, uint8_t *const data)
  This function reads a byte from specified address of EEPROM memory connected to BCC device via I2C bus.
- **bcc_status_t BCC_EEPROM_Write** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t addr, uint8_t data)
  This function writes a byte to specified address of EEPROM memory connected to BCC device via I2C bus.

Detailed Description

Battery cell controller SW driver V1.1. Supports boards based on MC33771B and MC33772B.

This module is common for all supported models.

bcc/bcc.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>
#include "bcc_mc3377x.h"
```

Data Structures

- struct **bcc_drv_data_t**
  Driver internal data.
- struct **bcc_drv_config_t**
  Driver configuration.

Macros

- #define **BCC_MSG_BIGEND**
  Use #define BCC_MSG_BIGEND for big-endian format of the TX/RX SPI buffer ([0] DATA_H, [1] DATA_L, ..., [4] CRC). If BCC_MSG_BIGEND is not defined, little-endian is used ([0] CRC, ..., [3] DATA_L, [4] DATA_H)
- #define **BCC_DEVICE_CNT_MAX_SPI** 1U
  Maximal number of Battery Cell Controller devices in SPI mode.
- #define **BCC_DEVICE_CNT_MAX_TPL** 15U
  Maximal number of Battery Cell Controller devices in TPL mode.
- #define **BCC_DEVICE_CNT_MAX** **BCC_DEVICE_CNT_MAX_TPL**
  Maximal number of Battery Cell Controller devices.
- #define **BCC_MIN_CELLS_MC33771** 7U
  Minimal battery cell count connected to MC33771.
- #define **BCC_MAX_CELLS_MC33771** 14U
  Maximal battery cell count connected to MC33771.
- #define **BCC_MIN_CELLS_MC33772** 3U
  Minimal battery cell count connected to MC33772.
- #define **BCC_MAX_CELLS_MC33772** 6U
  Maximal battery cell count connected to MC33772.
- #define **BCC_MAX_CELLS** 14U
  Maximal battery cell count connected to any BCC device.
- #define **BCC_MAX_CELLS_DEV**(dev) ((dev == **BCC_DEVICE_MC33771**) ? **BCC_MAX_CELLS_MC33771** : **BCC_MAX_CELLS_MC33772**)
  Maximal battery cell count connected to MC33772.
- #define **BCC_IS_CELL_CONN**(drvConfig, cid, cellNo) ((drvConfig)->drvData.cellMap[(cid) - 1U] & (1U << ((cellNo) - 1U)))
  Returns a non-zero value when desired cell (cellNo) is connected to the BCC specified by CID. Otherwise returns zero.
- #define **BCC_SPI_FREQ_MC3377x_MAX** 4200000U
  Maximal frequency of SPI clock in SPI mode.
- #define **BCC_SPI_FREQ_MC33664_MIN** 1900000U
  Minimal frequency of SPI_TX clock in TPL mode.
- #define **BCC_SPI_FREQ_MC33664_MAX** 2100000U
  Maximal frequency of SPI_TX clock in TPL mode.
- #define **BCC_INIT_CONF_REG_CNT** 68U
  Number of MC33771 registers configured in the initialization with user values.
- #define **BCC_MEAS_CNT** 30U

Number of MC33771 measurement registers.

- #define **BCC_STAT_CNT** 11U
  Number of BCC status registers.
- #define **BCC_MSG_SIZE** 5U
  Message size in bytes.
- #define **BCC_RX_LIMIT_TPL** 0x7FU
  Max. number of frames that can be read at once in TPL mode.
- #define **BCC_RX_BUF_SIZE_TPL** (**BCC_MSG_SIZE** * (**BCC_RX_LIMIT_TPL** + 1U))
  Size of buffer that is used for receiving via SPI in TPL mode.
- #define **BCC_GPIO_INPUT_CNT** 7U
  Number of GPIO/temperature sensor inputs.
- #define **BCC_GET_ISENSE_VOLT**(iSense1, iSense2) ((**BCC_GET_ISENSE_RAW_SIGN**(**BCC_GET_ISENSE_RAW**(iSense1, iSense2)) * 6) / 10)
  Calculates ISENSE value in [uV]. Resolution is 0.6 uV/LSB. Result is int32_t type. Note: Vind (Differential Input Voltage Range) is min. -150 mV and max. 150 mV (see datasheet).
- #define **BCC_GET_ISENSE_AMP**(rShunt, iSense1, iSense2)
  *This macro calculates ISENSE value in [mA]. Resolution is (600/R_SHUNT) mA/LSB (V2Res / Rshunt = 1000 * 0.6 uV / rShunt uOhm).*
- #define **BCC_GET_STACK_VOLT**(reg) ((uint32_t)**BCC_GET_MEAS_RAW**(reg) * 24414U / 10U)
  This macro converts value of the MEAS_STACK register to [uV]. Resolution is 2.4414 mV/LSB. Result is in range 0 - 80 000 000 uV.
- #define **BCC_GET_VOLT**(reg) (((uint32_t)**BCC_GET_MEAS_RAW**(reg) * 15259U) / 100U)
  Converts value of a register to [uV]. Resolution is 152.59 uV/LSB. Result is in range 0 - 5 000 000 uV. This macro is intended for the following registers: MEAS_CELLx, MEAS_ANx, MEAS_IC_TEMP, MEAS_VBG_DIAG_ADC1A and MEAS_VBG_DIAG_ADC1B.
- #define **BCC_GET_IC_TEMP**(reg) ((((int32_t)(**BCC_GET_MEAS_RAW**(reg))) * 32 - 273150) / 100)
  Converts value of the MEAS_IC_TEMP register to degrees Celsius. Resolution is 0.032 Kelvin/LSB.
- #define **BCC_IS_IN_RANGE**(val, min, max) (((val) >= (min)) && ((val) <= (max)))

Enumerations

- enum **bcc_status_t** { **BCC_STATUS_SUCCESS** = 0U, **BCC_STATUS_SPI_INIT** = 1U, **BCC_STATUS_SPI_BUSY** = 2U, **BCC_STATUS_PARAM_RANGE** = 4U, **BCC_STATUS_CRC** = 5U, **BCC_STATUS_COM_TAG_ID** = 6U, **BCC_STATUS_COM_RC** = 7U, **BCC_STATUS_COM_TIMEOUT** = 8U, **BCC_STATUS_DIAG_FAIL** = 9U, **BCC_STATUS_EEPROM_ERROR** = 10U, **BCC_STATUS_EEPROM_PRESENT** = 11U, **BCC_STATUS_NULL_RESP** = 12U }
- *Error codes.* enum **bcc_cid_t** { **BCC_CID_UNASSIG** = 0U, **BCC_CID_DEV1** = 1U, **BCC_CID_DEV2** = 2U, **BCC_CID_DEV3** = 3U, **BCC_CID_DEV4** = 4U, **BCC_CID_DEV5** = 5U, **BCC_CID_DEV6** = 6U, **BCC_CID_DEV7** = 7U, **BCC_CID_DEV8** = 8U, **BCC_CID_DEV9** = 9U, **BCC_CID_DEV10** = 10U, **BCC_CID_DEV11** = 11U, **BCC_CID_DEV12** = 12U, **BCC_CID_DEV13** = 13U, **BCC_CID_DEV14** = 14U, **BCC_CID_DEV15** = 15U }
- *Cluster Identification Address.* enum **bcc_mode_t** { **BCC_MODE_SPI** = 0U, **BCC_MODE_TPL** = 1U }
- *BCC communication mode.* enum **bcc_device_t** { **BCC_DEVICE_MC33771** = 0U, **BCC_DEVICE_MC33772** = 1U }
- *BCC device.* enum **bcc_measurements_t** { **BCC_MSR_CC_NB_SAMPLES** = 0U, **BCC_MSR_COULOMB_CNT1** = 1U, **BCC_MSR_COULOMB_CNT2** = 2U, **BCC_MSR_ISENSE1** = 3U, **BCC_MSR_ISENSE2** = 4U, **BCC_MSR_STACK_VOLT** = 5U, **BCC_MSR_CELL_VOLT14** = 6U, **BCC_MSR_CELL_VOLT13** = 7U, **BCC_MSR_CELL_VOLT12** = 8U, **BCC_MSR_CELL_VOLT11** = 9U,

BCC_MSR_CELL_VOLT10 = 10U, **BCC_MSR_CELL_VOLT9** = 11U, **BCC_MSR_CELL_VOLT8** = 12U, **BCC_MSR_CELL_VOLT7** = 13U, **BCC_MSR_CELL_VOLT6** = 14U, **BCC_MSR_CELL_VOLT5** = 15U, **BCC_MSR_CELL_VOLT4** = 16U, **BCC_MSR_CELL_VOLT3** = 17U, **BCC_MSR_CELL_VOLT2** = 18U, **BCC_MSR_CELL_VOLT1** = 19U, **BCC_MSR_AN6** = 20U, **BCC_MSR_AN5** = 21U, **BCC_MSR_AN4** = 22U, **BCC_MSR_AN3** = 23U, **BCC_MSR_AN2** = 24U, **BCC_MSR_AN1** = 25U, **BCC_MSR_AN0** = 26U, **BCC_MSR_ICTEMP** = 27U, **BCC_MSR_VBGADC1A** = 28U, **BCC_MSR_VBGADC1B** = 29U }

- *Measurements provided by Battery Cell Controller.* enum **bcc_fault_status_t** { **BCC_FS_CELL_OV** = 0U, **BCC_FS_CELL_UV** = 1U, **BCC_FS_CB_OPEN** = 2U, **BCC_FS_CB_SHORT** = 3U, **BCC_FS_GPIO_STATUS** = 4U, **BCC_FS_AN_OT_UT** = 5U, **BCC_FS_GPIO_SHORT** = 6U, **BCC_FS_COMM** = 7U, **BCC_FS_FAULT1** = 8U, **BCC_FS_FAULT2** = 9U, **BCC_FS_FAULT3** = 10U }

Status provided by Battery Cell Controller. Functions

- **bcc_status_t BCC_Init** (**bcc_drv_config_t** *const drvConfig, const uint16_t devConf[][**BCC_INIT_CONF_REG_CNT**])
  This function initializes the Battery Cell Controller device(s), configures its registers, assigns CID and initializes internal driver data.
- **bcc_status_t BCC_VerifyCom** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function uses No Operation command of BCC to verify communication with device specified by CID without performing any operation.
- **bcc_status_t BCC_Sleep** (**bcc_drv_config_t** *const drvConfig)
  This function sets sleep mode to all Battery Cell Controller devices.
- void **BCC_WakeUp** (const **bcc_drv_config_t** *const drvConfig)
  This function sets normal mode to all Battery Cell Controller devices.
- **bcc_status_t BCC_SoftwareReset** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function resets BCC device using software reset. It enters reset via SPI or TPL interface.
- void **BCC_HardwareReset** (const **bcc_drv_config_t** *const drvConfig)
  This function resets BCC device using GPIO pin.
- **bcc_status_t BCC_TPL_Enable** (const **bcc_drv_config_t** *const drvConfig)
  This function enables MC33664 device (sets a normal mode). Intended for TPL mode only!
- void **BCC_TPL_Disable** (const **bcc_drv_config_t** *const drvConfig)
  This function sets MC33664 device into sleep mode. Intended for TPL mode only!
- **bcc_status_t BCC_Reg_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint8_t regCnt, uint16_t *regVal)
  This function reads a value from addressed register (or desired number of registers) of selected Battery Cell Controller device.
- **bcc_status_t BCC_Reg_Write** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regVal, uint16_t *retReg)
  This function writes a value to addressed register of selected Battery Cell Controller device.
- **bcc_status_t BCC_Reg_WriteGlobal** (**bcc_drv_config_t** *const drvConfig, uint8_t regAddr, uint16_t regVal)
  This function writes a value to addressed register of all configured BCC devices. Intended for TPL mode only!
- **bcc_status_t BCC_Reg_Update** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regMask, uint16_t regVal)
  This function updates content of a selected register. It affects bits specified by a bit mask only.
- **bcc_status_t BCC_Meas_StartConversion** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function starts ADC conversion. It sets Start of Conversion bit and new value of TAG ID in ADC_CFG register.
- **bcc_status_t BCC_Meas_StartConversionGlobal** (**bcc_drv_config_t** *const drvConfig, uint16_t adcCfgValue)

This function starts ADC conversion for all devices in TPL chain. It uses a Global Write command to set ADC_CFG register. Intended for TPL mode only!

- **bcc_status_t BCC_Meas_IsConverting** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, bool *completed)
  This function checks status of conversion defined by End of Conversion bit in ADC_CFG register.
- **bcc_status_t BCC_Meas_GetRawValues** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint16_t measurements[])
  This function reads the measurement registers and returns raw values. Macros defined in BCC header file can be used to perform correct unit conversion.
- **bcc_status_t BCC_Fault_GetStatus** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint16_t status[])
  *This function reads the status registers and returns raw values. You can use constants defined in **bcc_mc3377x.h** file.*
- **bcc_status_t BCC_Fault_ClearStatus** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, **bcc_fault_status_t** statSel)
  This function clears selected fault status register.
- **bcc_status_t BCC_GPIO_SetOutput** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t gpioSel, bool val)
  This function sets output value of one BCC GPIO pin. This function should be used only when at least one GPIO is in output mode. Resets BCC device using GPIO pin.
- **bcc_status_t BCC_CB_Enable** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, bool enable)
  This function enables or disables the cell balancing via SYS_CFG1[CB_DRVEN] bit.
- **bcc_status_t BCC_CB_SetIndividual** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t cellIndex, bool enable, uint16_t timer)
  This function enables or disables cell balancing for a specified cell and sets its timer.
- **bcc_status_t BCC_CB_Pause** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, bool pause)
  This function can be used to manual pause cell balancing before on demand conversion. As a result more precise measurement can be done. Note that it is user obligation to re-enable cell balancing after measurement ends.
- **bcc_status_t BCC_FuseMirror_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t fuseAddr, uint16_t *const value)
  This function reads a fuse mirror register of a BCC device specified by CID.
- **bcc_status_t BCC_FuseMirror_Write** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t fuseAddr, uint16_t value)
  This function writes a fuse mirror register of a BCC device specified by CID.
- **bcc_status_t BCC_GUID_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint64_t *const guid)
  This function reads an unique serial number of the BCC device from the content of mirror registers.
- **bcc_status_t BCC_EEPROM_Read** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t addr, uint8_t *const data)
  This function reads a byte from specified address of EEPROM memory connected to BCC device via I2C bus.
- **bcc_status_t BCC_EEPROM_Write** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t addr, uint8_t data)
  This function writes a byte to specified address of EEPROM memory connected to BCC device via I2C bus.
- void **BCC_MCU_WaitMs** (uint16_t delay)
  Waits for specified amount of milliseconds. This function needs to be implemented for specified MCU by the user.
- void **BCC_MCU_WaitUs** (uint32_t delay)

Waits for specified amount of microseconds. This function needs to be implemented for specified MCU by the user.

- void **BCC_MCU_Assert** (bool x)
  User implementation of assert.
- **bcc_status_t BCC_MCU_TransferSpi** (uint8_t drvInstance, uint8_t transBuf[], uint8_t recvBuf[])
  This function performs one 40b transfer via SPI bus. Intended for SPI mode only. This function needs to be implemented for specified MCU by the user.
- **bcc_status_t BCC_MCU_TransferTpl** (uint8_t drvInstance, uint8_t transBuf[], uint8_t recvBuf[], uint16_t recvTrCnt)
  This function sends and receives data via TX and RX SPI buses. Intended for TPL mode only. This function needs to be implemented for specified MCU by the user.
- void **BCC_MCU_WriteCsbPin** (uint8_t drvInstance, uint8_t value)
  Writes logic 0 or 1 to the CSB pin (or CSB_TX in case of TPL mode). This function needs to be implemented by the user.
- void **BCC_MCU_WriteRstPin** (uint8_t drvInstance, uint8_t value)
  Writes logic 0 or 1 to the RST pin. This function needs to be implemented by the user. If no RST pin is used, keep the function body empty.
- void **BCC_MCU_WriteEnPin** (uint8_t drvInstance, uint8_t value)
  Writes logic 0 or 1 to the EN pin of MC33664. This function is called only in the TPL and it needs to be implemented by the user.
- uint32_t **BCC_MCU_ReadIntbPin** (uint8_t drvInstance)
  Reads logic value of INTB pin of MC33664. This function is called in the TPL mode only and it needs to be implemented by the user.

Detailed Description

Battery cell controller SW driver V1.1. Supports boards based on MC33771B and MC33772B.

This module is common for all supported models.

Macro Definition Documentation

#define BCC_GET_IC_TEMP(reg) ((((int32_t)(BCC_GET_MEAS_RAW(reg))) * 32 - 273150) / 100)

Converts value of the MEAS_IC_TEMP register to degrees Celsius. Resolution is 0.032 Kelvin/LSB.

Parameters:

reg	Value of the MEAS_IC_TEMP register.

Returns:

Converted value in [deg C] multiplied by 10 (i.e. resolution of 0.1 deg C).

#define BCC_GET_ISENSE_AMP(rShunt, iSense1, iSense2)

```
Value:( \
    (BCC_GET_ISENSE_RAW_SIGN(BCC_GET_ISENSE_RAW(iSense1, iSense2)) * 600) /
(int32 t)(rShunt)                                                  \
)
```

This macro calculates ISENSE value in [mA]. Resolution is (600/R_SHUNT) mA/LSB (V2Res / Rshunt = 1000 * 0.6 uV / rShunt uOhm).

Parameters:

rShunt	Resistance of Shunt resistor in [uOhm].
iSense1	Content of register MEAS_ISENSE1.
iSense2	Content of register MEAS_ISENSE2.

Returns:

ISENSE current in [mA].

#define BCC_GET_ISENSE_VOLT(iSense1, iSense2) ((BCC_GET_ISENSE_RAW_SIGN(BCC_GET_ISENSE_RAW(iSense1, iSense2)) * 6) / 10)

Calculates ISENSE value in [uV]. Resolution is 0.6 uV/LSB. Result is int32_t type. Note: Vind (Differential Input Voltage Range) is min. -150 mV and max. 150 mV (see datasheet).

Parameters:

iSense1	Content of register MEAS_ISENSE1.
iSense2	Content of register MEAS_ISENSE2.

Returns:

ISENSE voltage in [uV].

#define BCC_GET_STACK_VOLT(reg) ((uint32_t)BCC_GET_MEAS_RAW(reg) * 24414U / 10U)

This macro converts value of the MEAS_STACK register to [uV]. Resolution is 2.4414 mV/LSB. Result is in range 0 - 80 000 000 uV.

Parameters:

reg	Value of the MEAS_STACK register.

Returns:

Converted value in [uV].

#define BCC_GET_VOLT(reg) (((uint32_t)BCC_GET_MEAS_RAW(reg) * 15259U) / 100U)

Converts value of a register to [uV]. Resolution is 152.59 uV/LSB. Result is in range 0 - 5 000 000 uV. This macro is intended for the following registers: MEAS_CELLx, MEAS_ANx, MEAS_IC_TEMP, MEAS_VBG_DIAG_ADC1A and MEAS_VBG_DIAG_ADC1B.

Parameters:

reg	Value of a measurement register.

Returns:

Converted value in [uV].

#define BCC_INIT_CONF_REG_CNT 68U

Number of MC33771 registers configured in the initialization with user values.

Note that the number of configured registers of MC33772 is 52 only. See BCC_INIT_CONF_REG_ADDR in **bcc.c** for more details.

#define BCC_IS_CELL_CONN(drvConfig, cid, cellNo) ((drvConfig)->drvData.cellMap[(cid) - 1U] & (1U << ((cellNo) - 1U)))

Returns a non-zero value when desired cell (cellNo) is connected to the BCC specified by CID. Otherwise returns zero.

Parameters:

drvConfig	Pointer to driver instance configuration.
cid	Cluster Identification Address.
cellNo	Number of a cell (range is {1, ..., 14} for MC33771 and {1, ..., 6} for MC33772).

Returns:

Non-zero value if cell is connected, zero otherwise.

#define BCC_IS_IN_RANGE(val, min, max) (((val) >= (min)) && ((val) <= (max)))

Returns true if value VAL is in the range defined by MIN and MAX values (range includes the border values).

Parameters:

val	Comparison value.
min	Minimal value of the range.
max	Maximal value of the range.

Returns:

True if value is the range. False otherwise.

#define BCC_MAX_CELLS_DEV(dev) ((dev == BCC_DEVICE_MC33771) ? BCC_MAX_CELLS_MC33771 : BCC_MAX_CELLS_MC33772)

Maximal battery cell count connected to MC33772.

Parameters:

dev	BCC device type.

#define BCC_MEAS_CNT 30U

Number of MC33771 measurement registers.

Note MC33772 contains 22 measurement registers. For compliance with bcc_measurements_t indexes, BCC_Meas_GetRawValues function requires 30 x uint16_t array for both BCC devices.

bcc/bcc_communication.c File Reference

`#include "bcc_communication.h"`

Macros

- #define **BCC_CRC_TBL_SIZE** 256U
  Size of CRC table.
- #define **BCC_HAS_TAG_ID_MC33771**(regAddr)
  This macro determines format of a MC33771 frame with use of register address. Following registers only use TAG ID (reading): SYS_DIAG (5), FAULT1_STATUS (24), FAULT2_STATUS (25), FAULT3_STATUS (26) and from CC_NB_SAMPLES (2D) to MEAS_ VBG_DIAG_ADC1B (4A).
- #define **BCC_HAS_TAG_ID_MC33772**(regAddr)
  This macro determines format of a MC33772 frame with use of register address. Following registers only use TAG ID (reading): SYS_DIAG (5), FAULT1_STATUS (24), FAULT2_STATUS (25), FAULT3_STATUS (26), from CC_NB_SAMPLES (2D) to MEAS_STACK (32) and from MEAS_CELL7 (3A) to MEAS_VBG_DIAG_ADC1B (4A).

Functions

- void **BCC_PackFrame** (uint16_t data, uint8_t addr, **bcc_cid_t** cid, uint8_t cmd, uint8_t frame[])
  This function packs all the parameters into a frame according to the BCC frame format (see BCC datasheet).
- **bcc_status_t BCC_CheckCRC** (const uint8_t *resp)
  This function calculates CRC of a received frame and compares it with CRC field of the frame.
- **bcc_status_t BCC_CheckRcTagId** (**bcc_device_t** devType, const uint8_t *resp, uint8_t rc, uint8_t tagId)
  This function checks value of the Command field of a frame.

Detailed Description

This file implements functions for SPI communication of BCC driver used in both SPI and TPL mode.

Macro Definition Documentation

#define BCC_HAS_TAG_ID_MC33771(regAddr)

```
Value:(((regAddr) == BCC_REG_SYS_DIAG_ADDR) || \
   ((regAddr) == BCC_REG_FAULT1_STATUS_ADDR) || \
   ((regAddr) == BCC_REG_FAULT2_STATUS_ADDR) || \
   ((regAddr) == BCC_REG_FAULT3_STATUS_ADDR) || \
   (((regAddr) >= BCC_REG_CC_NB_SAMPLES_ADDR) && \
   ((regAddr) <= BCC_REG_MEAS_VBG_DIAG_ADC1B_ADDR)) \
  )
```

This macro determines format of a MC33771 frame with use of register address. Following registers only use TAG ID (reading): SYS_DIAG (5), FAULT1_STATUS (24), FAULT2_STATUS (25), FAULT3_STATUS (26) and from CC_NB_SAMPLES (2D) to MEAS_ VBG_DIAG_ADC1B (4A).

Parameters:

regAddr	Address of a register (typically extracted from received frame).

Returns:

True if format of frame with selected register address use TAG ID, false otherwise.

#define BCC_HAS_TAG_ID_MC33772(regAddr)

```
Value:(((regAddr) == BCC_REG_SYS_DIAG_ADDR) || \
   ((regAddr) == BCC_REG_FAULT1_STATUS_ADDR) || \
   ((regAddr) == BCC_REG_FAULT2_STATUS_ADDR) || \
   ((regAddr) == BCC_REG_FAULT3_STATUS_ADDR) || \
   (((regAddr) >= BCC_REG_CC_NB_SAMPLES_ADDR) && \
   ((regAddr) <= BCC_REG_MEAS_STACK_ADDR)) || \
   (((regAddr) >= BCC_REG_MEAS_CELLX_ADDR_MC33772_START) && \
   ((regAddr) <= BCC_REG_MEAS_VBG_DIAG_ADC1B_ADDR)) \
  )
```

This macro determines format of a MC33772 frame with use of register address. Following registers only use TAG ID (reading): SYS_DIAG (5), FAULT1_STATUS (24), FAULT2_STATUS (25), FAULT3_STATUS (26), from CC_NB_SAMPLES (2D) to MEAS_STACK (32) and from MEAS_CELL7 (3A) to MEAS_VBG_DIAG_ADC1B (4A).

Parameters:

regAddr	Address of a register (typically extracted from received frame).

Returns:

True if format of frame with selected register address use TAG ID, false otherwise.

bcc/bcc_communication.h File Reference

`#include "bcc.h"`

Macros

- #define **BCC_MSG_IDX_DATA_H** 0U
  Index to memory data field of SPI frame (higher byte).
- #define **BCC_MSG_IDX_DATA_L** 1U
  Index to memory data field of SPI frame (lower byte).
- #define **BCC_MSG_IDX_ADDR** 2U
  Index to memory address field of SPI frame.
- #define **BCC_MSG_IDX_CID_CMD** 3U
  Index to physical address (CID) and command fields of SPI frame.
- #define **BCC_MSG_IDX_CRC** 4U
  Index to CRC field of SPI frame.
- #define **BCC_CMD_NOOP** 0x00U
  No operation command.
- #define **BCC_CMD_READ** 0x01U
  Read command.
- #define **BCC_CMD_WRITE** 0x02U
  Write command.
- #define **BCC_CMD_GLOB_WRITE** 0x03U
  Global write command.
- #define **BCC_GET_MSG_DATA**(msg)
  Returns Memory Data field from a frame.
- #define **BCC_MSG_ADDR_MASK** 0x7FU
  Mask for memory address field of frame.
- #define **BCC_MSG_RC_MASK** 0x0CU
  Mask for RC field of frame.
- #define **BCC_MSG_TAGID_MASK** 0x0FU
  Mask for TAG ID field of frame.
- #define **BCC_GET_RC**(binVal) ((((binVal) >> 1) ^ (binVal)) << 2)
  Converts binary value to Gray code used for Rolling Counter. It is a 2-bit Gray code. Final value is shifted left by 2 bits.
- #define **BCC_INC_RC_IDX**(rcIdx) (((rcIdx) + 1U) & 0x03U)
  Increments rcIdx value and executes modulo 4.

Functions

- void **BCC_PackFrame** (uint16_t data, uint8_t addr, **bcc_cid_t** cid, uint8_t cmd, uint8_t frame[])
  This function packs all the parameters into a frame according to the BCC frame format (see BCC datasheet).
- **bcc_status_t BCC_CheckCRC** (const uint8_t *resp)
  This function calculates CRC of a received frame and compares it with CRC field of the frame.
- **bcc_status_t BCC_CheckRcTagId** (**bcc_device_t** devType, const uint8_t *resp, uint8_t rc, uint8_t tagId)
  This function checks value of the Command field of a frame.

Detailed Description

This file provides access to the functions used for SPI communication of BCC driver in both SPI and TPL mode.

Macro Definition Documentation

#define BCC_CMD_NOOP 0x00U

No operation command.

BCC Commands.

#define BCC_GET_MSG_DATA(msg)

```
Value:(((uint16_t)*((msg) + BCC_MSG_IDX_DATA_H) << 8U) | \
     (uint16_t)*((msg) + BCC_MSG_IDX_DATA_L))
```

Returns Memory Data field from a frame.

Parameters:

msg	Pointer to the frame.

Returns:

Memory data field.

#define BCC_GET_RC(binVal) ((((binVal) >> 1) ^ (binVal)) << 2)

Converts binary value to Gray code used for Rolling Counter. It is a 2-bit Gray code. Final value is shifted left by 2 bits.

Parameters:

binVal	Binary value to be converted.

Returns:

Converted value shifted left by 2 bits.

#define BCC_INC_RC_IDX(rcIdx) (((rcIdx) + 1U) & 0x03U)

Increments rcIdx value and executes modulo 4.

Parameters:

rcIdx	Index to be incremented.

Returns:

Incremented rcIdx value.

#define BCC_MSG_IDX_DATA_H 0U

Index to memory data field of SPI frame (higher byte).

SPI Frame.

bcc/bcc_mc3377x.h File Reference

Macros

- #define **BCC_REG_SET_BIT_VALUE**(reg, mask) ((reg) | (mask))
- #define **BCC_REG_UNSET_BIT_VALUE**(reg, mask) ((reg) & ~(mask))
- #define **BCC_REG_GET_BIT_VALUE**(reg, mask, shift) (((reg) & (mask)) >> (shift))
- #define **BCC_REG_SET_BITS_VALUE**(reg, mask, shift, val, range) (((reg) & ~(mask)) | (((val) & (range)) << (shift)))
- #define **BCC_REG_GET_BITS_VALUE**(reg, mask, shift) (((reg) & (mask)) >> (shift))
- #define **BCC_MAX_REG_ADDR** 0x7FU
- #define **BCC_MAX_FUSE_ADDR** 0x1FU
- #define **BCC_MAX_EEPROM_ADDR** 0x7FU
- #define **BCC_REG_INIT_ADDR** 0x01U
- #define **BCC_REG_INIT_DEFAULT** 0x0000U
- #define **BCC_RW_CID_MASK** 0x000FU
- #define **BCC_RW_BUS_SW_MASK** 0x0010U
- #define **BCC_RW_RTERM_MASK** 0x0020U
- #define **BCC_RW_CID_SHIFT** 0x00U
- #define **BCC_RW_BUS_SW_SHIFT** 0x04U
- #define **BCC_RW_RTERM_SHIFT** 0x05U
- #define **BCC_BUS_SWITCH_DISABLED** (0x00U << BCC_RW_BUS_SW_SHIFT)
- #define **BCC_BUS_SWITCH_ENABLED** (0x01U << BCC_RW_BUS_SW_SHIFT)
- #define **BCC_RTERM_COMM_SW** (0x00U << BCC_RW_RTERM_SHIFT) /* Depends on the status of the communication switch. */
- #define **BCC_RTERM_CONNECTED** (0x01U << BCC_RW_RTERM_SHIFT) /* Connected regardless of the bus switch status. */
- #define **BCC_SET_CID**(reg, cid) **BCC_REG_SET_BITS_VALUE**(reg, BCC_RW_CID_MASK, BCC_RW_CID_SHIFT, cid, 0x0FU)
- #define **BCC_REG_SYS_CFG_GLOBAL_ADDR** 0x02U
- #define **BCC_REG_SYS_CFG_GLOBAL_DEFAULT** 0x0000U
- #define **BCC_W_GO2SLEEP_MASK** 0x0001U
- #define **BCC_W_GO2SLEEP_SHIFT** 0x00U
- #define **BCC_GO2SLEEP_DISABLED** (0x00U << BCC_W_GO2SLEEP_SHIFT)
- #define **BCC_GO2SLEEP_ENABLED** (0x01U << BCC_W_GO2SLEEP_SHIFT)
- #define **BCC_REG_SYS_CFG1_ADDR** 0x03U
- #define **BCC_REG_SYS_CFG1_DEFAULT** 0x9001U
- #define **BCC_RW_WAVE_DC_MASK** 0x0006U
- #define **BCC_RW_FAULT_WAVE_MASK** 0x0008U
- #define **BCC_W_SOFT_RST_MASK** 0x0010U
- #define **BCC_RW_CB_MANUAL_PAUSE_MASK** 0x0020U
- #define **BCC_W_GO2DIAG_MASK** 0x0040U
- #define **BCC_R_DIAG_ST_MASK** 0x0040U
- #define **BCC_RW_CB_DRVEN_MASK** 0x0080U
- #define **BCC_RW_CB_AUTO_PAUSE_MASK** 0x0100U
- #define **BCC_RW_I_MEAS_EN_MASK** 0x0200U
- #define **BCC_RW_DIAG_TIMEOUT_MASK** 0x1C00U
- #define **BCC_RW_CYCLIC_TIMER_MASK** 0xE000U
- #define **BCC_RW_WAVE_DC_SHIFT** 0x01U
- #define **BCC_RW_FAULT_WAVE_SHIFT** 0x03U
- #define **BCC_W_SOFT_RST_SHIFT** 0x04U
- #define **BCC_RW_CB_MANUAL_PAUSE_SHIFT** 0x05U
- #define **BCC_W_GO2DIAG_SHIFT** 0x06U
- #define **BCC_R_DIAG_ST_SHIFT** 0x06U
- #define **BCC_RW_CB_DRVEN_SHIFT** 0x07U
- #define **BCC_RW_CB_AUTO_PAUSE_SHIFT** 0x08U

- #define **BCC_RW_I_MEAS_EN_SHIFT** 0x09U
- #define **BCC_RW_DIAG_TIMEOUT_SHIFT** 0x0AU
- #define **BCC_RW_CYCLIC_TIMER_SHIFT** 0x0DU
- #define **BCC_WAVE_DC_500US** (0x00U << BCC_RW_WAVE_DC_SHIFT)
- #define **BCC_WAVE_DC_1MS** (0x01U << BCC_RW_WAVE_DC_SHIFT)
- #define **BCC_WAVE_DC_10MS** (0x02U << BCC_RW_WAVE_DC_SHIFT)
- #define **BCC_WAVE_DC_100MS** (0x03U << BCC_RW_WAVE_DC_SHIFT)
- #define **BCC_FAULT_WAVE_DISABLED** (0x00U << BCC_RW_FAULT_WAVE_SHIFT)
- #define **BCC_FAULT_WAVE_ENABLED** (0x01U << BCC_RW_FAULT_WAVE_SHIFT)
- #define **BCC_SW_RESET_DISABLED** (0x00U << BCC_W_SOFT_RST_SHIFT)
- #define **BCC_SW_RESET_ENABLED** (0x01U << BCC_W_SOFT_RST_SHIFT)
- #define **BCC_CB_MAN_PAUSE_DISABLED** (0x00U << BCC_RW_CB_MANUAL_PAUSE_SHIFT)
- #define **BCC_CB_MAN_PAUSE_ENABLED** (0x01U << BCC_RW_CB_MANUAL_PAUSE_SHIFT)
- #define **BCC_DIAG_MODE_DISABLED** (0x00U << BCC_W_GO2DIAG_SHIFT)
- #define **BCC_DIAG_MODE_ENABLED** (0x01U << BCC_W_GO2DIAG_SHIFT)
- #define **BCC_CB_DRV_DISABLED** (0x00U << BCC_RW_CB_DRVEN_SHIFT)
- #define **BCC_CB_DRV_ENABLED** (0x01U << BCC_RW_CB_DRVEN_SHIFT)
- #define **BCC_CB_AUTO_PAUSE_DISABLED** (0x00U << BCC_RW_CB_AUTO_PAUSE_SHIFT)
- #define **BCC_CB_AUTO_PAUSE_ENABLED** (0x01U << BCC_RW_CB_AUTO_PAUSE_SHIFT)
- #define **BCC_I_MEAS_DISABLED** (0x00U << BCC_RW_I_MEAS_EN_SHIFT)
- #define **BCC_I_MEAS_ENABLED** (0x01U << BCC_RW_I_MEAS_EN_SHIFT)
- #define **BCC_DIAG_TIMEOUT_NO_TIMER** (0x00U << BCC_RW_DIAG_TIMEOUT_SHIFT)
- #define **BCC_DIAG_TIMEOUT_0_05S** (0x01U << BCC_RW_DIAG_TIMEOUT_SHIFT)
- #define **BCC_DIAG_TIMEOUT_0_1S** (0x02U << BCC_RW_DIAG_TIMEOUT_SHIFT)
- #define **BCC_DIAG_TIMEOUT_0_2S** (0x03U << BCC_RW_DIAG_TIMEOUT_SHIFT)
- #define **BCC_DIAG_TIMEOUT_1S** (0x04U << BCC_RW_DIAG_TIMEOUT_SHIFT)
- #define **BCC_DIAG_TIMEOUT_2S** (0x05U << BCC_RW_DIAG_TIMEOUT_SHIFT)
- #define **BCC_DIAG_TIMEOUT_4S** (0x06U << BCC_RW_DIAG_TIMEOUT_SHIFT)
- #define **BCC_DIAG_TIMEOUT_8S** (0x07U << BCC_RW_DIAG_TIMEOUT_SHIFT)
- #define **BCC_CYCLIC_TIMER_DISABLED** (0x00U << BCC_RW_CYCLIC_TIMER_SHIFT)
- #define **BCC_CYCLIC_TIMER_CONTINOUS** (0x01U << BCC_RW_CYCLIC_TIMER_SHIFT)
- #define **BCC_CYCLIC_TIMER_0_1S** (0x02U << BCC_RW_CYCLIC_TIMER_SHIFT)
- #define **BCC_CYCLIC_TIMER_0_2S** (0x03U << BCC_RW_CYCLIC_TIMER_SHIFT)
- #define **BCC_CYCLIC_TIMER_1S** (0x04U << BCC_RW_CYCLIC_TIMER_SHIFT)
- #define **BCC_CYCLIC_TIMER_2S** (0x05U << BCC_RW_CYCLIC_TIMER_SHIFT)
- #define **BCC_CYCLIC_TIMER_4S** (0x06U << BCC_RW_CYCLIC_TIMER_SHIFT)
- #define **BCC_CYCLIC_TIMER_8S** (0x07U << BCC_RW_CYCLIC_TIMER_SHIFT)
- #define **BCC_REG_SYS_CFG2_ADDR** 0x04U
- #define **BCC_REG_SYS_CFG2_DEFAULT** 0x0334U
- #define **BCC_RW_HAMM_ENCOD_MASK** 0x0001U
- #define **BCC_RW_NUMB_ODD_MASK** 0x0002U
- #define **BCC_R_VPRE_UV_MASK** 0x0004U
- #define **BCC_RW_TIMEOUT_COMM_MASK** 0x0030U
- #define **BCC_RW_FLT_RST_CFG_MASK** 0x03C0U
- #define **BCC_R_PREVIOUS_STATE_MASK** 0x1C00U
- #define **BCC_RW_HAMM_ENCOD_SHIFT** 0x00U
- #define **BCC_RW_NUMB_ODD_SHIFT** 0x01U
- #define **BCC_R_VPRE_UV_SHIFT** 0x02U
- #define **BCC_RW_TIMEOUT_COMM_SHIFT** 0x04U
- #define **BCC_RW_FLT_RST_CFG_SHIFT** 0x06U

- #define **BCC_R_PREVIOUS_STATE_SHIFT** 0x0AU
- #define **BCC_HAMM_ENCOD_DECODE** (0x00U << BCC_RW_HAMM_ENCOD_SHIFT)
- #define **BCC_HAMM_ENCOD_ENCODE** (0x01U << BCC_RW_HAMM_ENCOD_SHIFT)
- #define **BCC_EVEN_CELLS** (0x00U << BCC_RW_NUMB_ODD_SHIFT)
- #define **BCC_ODD_CELLS** (0x01U << BCC_RW_NUMB_ODD_SHIFT)
- #define **BCC_VPRE_UV_NO_UNDERVOLTAGE** (0x00U << BCC_R_VPRE_UV_SHIFT)
- #define **BCC_VPRE_UV_DETECTED** (0x01U << BCC_R_VPRE_UV_SHIFT)
- #define **BCC_TIMEOUT_COMM_32MS** (0x00U << BCC_RW_TIMEOUT_COMM_SHIFT)
- #define **BCC_TIMEOUT_COMM_64MS** (0x01U << BCC_RW_TIMEOUT_COMM_SHIFT)
- #define **BCC_TIMEOUT_COMM_128MS** (0x02U << BCC_RW_TIMEOUT_COMM_SHIFT)
- #define **BCC_TIMEOUT_COMM_256MS** (0x03U << BCC_RW_TIMEOUT_COMM_SHIFT)
- #define **BCC_FLT_RST_CFG_RESET_DIS** (0x03U << BCC_RW_FLT_RST_CFG_SHIFT)
- #define **BCC_FLT_RST_CFG_OSC_MON** (0x05U << BCC_RW_FLT_RST_CFG_SHIFT)
- #define **BCC_FLT_RST_CFG_OSC** (0x06U << BCC_RW_FLT_RST_CFG_SHIFT)
- #define **BCC_FLT_RST_CFG_COM** (0x09U << BCC_RW_FLT_RST_CFG_SHIFT)
- #define **BCC_FLT_RST_CFG_COM_OSC_MON** (0x0AU << BCC_RW_FLT_RST_CFG_SHIFT)
- #define **BCC_FLT_RST_CFG_COM_OSC** (0x0CU << BCC_RW_FLT_RST_CFG_SHIFT)
- #define **BCC_REG_SYS_DIAG_ADDR** 0x05U
- #define **BCC_REG_SYS_DIAG_DEFAULT** 0x0000U
- #define **BCC_RW_CB_OL_EVEN_MASK** 0x0001U
- #define **BCC_RW_CB_OL_ODD_MASK** 0x0002U
- #define **BCC_RW_CT_OL_EVEN_MASK** 0x0004U
- #define **BCC_RW_CT_OL_ODD_MASK** 0x0008U
- #define **BCC_RW_CT_OV_UV_MASK** 0x0010U
- #define **BCC_RW_CT_LEAK_DIAG_MASK** 0x0020U
- #define **BCC_RW_POLARITY_MASK** 0x0040U
- #define **BCC_RW_DA_DIAG_MASK** 0x0080U
- #define **BCC_RW_ANX_TEMP_DIAG_MASK** 0x0100U
- #define **BCC_RW_ANX_OLDIAG_MASK** 0x0200U
- #define **BCC_RW_ISENSE_OL_DIAG_MASK** 0x0400U
- #define **BCC_RW_I_MUX_MASK** 0x1800U
- #define **BCC_RW_FAULT_DIAG_MASK** 0x8000U
- #define **BCC_RW_CB_OL_EVEN_SHIFT** 0x00U
- #define **BCC_RW_CB_OL_ODD_SHIFT** 0x01U
- #define **BCC_RW_CT_OL_EVEN_SHIFT** 0x02U
- #define **BCC_RW_CT_OL_ODD_SHIFT** 0x03U
- #define **BCC_RW_CT_OV_UV_SHIFT** 0x04U
- #define **BCC_RW_CT_LEAK_DIAG_SHIFT** 0x05U
- #define **BCC_RW_POLARITY_SHIFT** 0x06U
- #define **BCC_RW_DA_DIAG_SHIFT** 0x07U
- #define **BCC_RW_ANX_TEMP_DIAG_SHIFT** 0x08U
- #define **BCC_RW_ANX_OLDIAG_SHIFT** 0x09U
- #define **BCC_RW_ISENSE_OL_DIAG_SHIFT** 0x0AU
- #define **BCC_RW_I_MUX_SHIFT** 0x0BU
- #define **BCC_RW_FAULT_DIAG_SHIFT** 0x0FU
- #define **BCC_CB_OL_EVEN_OPEN** (0x00U << BCC_RW_CB_OL_EVEN_SHIFT)
- #define **BCC_CB_OL_EVEN_CLOSED** (0x01U << BCC_RW_CB_OL_EVEN_SHIFT)
- #define **BCC_CB_OL_ODD_OPEN** (0x00U << BCC_RW_CB_OL_ODD_SHIFT)
- #define **BCC_CB_OL_ODD_CLOSED** (0x01U << BCC_RW_CB_OL_ODD_SHIFT)
- #define **BCC_CT_OL_EVEN_OPEN** (0x00U << BCC_RW_CT_OL_EVEN_SHIFT)
- #define **BCC_CT_OL_EVEN_CLOSED** (0x01U << BCC_RW_CT_OL_EVEN_SHIFT)
- #define **BCC_CT_OL_ODD_OPEN** (0x00U << BCC_RW_CT_OL_ODD_SHIFT)
- #define **BCC_CT_OL_ODD_CLOSED** (0x01U << BCC_RW_CT_OL_ODD_SHIFT)
- #define **BCC_CT_OV_UV_DISABLED** (0x00U << BCC_RW_CT_OV_UV_SHIFT)

- #define **BCC_CT_OV_UV_ENABLED** (0x01U << BCC_RW_CT_OV_UV_SHIFT)
- #define **BCC_CT_LEAK_DIAG_NORMAL** (0x00U << BCC_RW_CT_LEAK_DIAG_SHIFT)
- #define **BCC_CT_LEAK_DIAG_DIFF** (0x01U << BCC_RW_CT_LEAK_DIAG_SHIFT)
- #define **BCC_POL_NON_INVERTED** (0x00U << BCC_RW_POLARITY_SHIFT)
- #define **BCC_POL_INVERTED** (0x01U << BCC_RW_POLARITY_SHIFT)
- #define **BCC_DA_DIAG_NO_CHECK** (0x00U << BCC_RW_DA_DIAG_SHIFT)
- #define **BCC_DA_DIAG_CHECK** (0x01U << BCC_RW_DA_DIAG_SHIFT)
- #define **BCC_ANX_DIAG_SW_OPEN** (0x00U << BCC_RW_ANX_TEMP_DIAG_SHIFT)
- #define **BCC_ANX_DIAG_SW_CLOSED** (0x01U << BCC_RW_ANX_TEMP_DIAG_SHIFT)
- #define **BCC_ANX_OL_DIAG_DISABLED** (0x00U << BCC_RW_ANX_OLDIAG_SHIFT)
- #define **BCC_ANX_OL_DIAG_ENABLED** (0x01U << BCC_RW_ANX_OLDIAG_SHIFT)
- #define **BCC_ISENSE_OL_DIAG_DISABLED** (0x00U << BCC_RW_ISENSE_OL_DIAG_SHIFT)
- #define **BCC_ISENSE_OL_DIAG_ENABLED** (0x01U << BCC_RW_ISENSE_OL_DIAG_SHIFT)
- #define **BCC_IMUX_ISENSE** (0x00U << BCC_RW_I_MUX_SHIFT)
- #define **BCC_IMUX_GPIO5_6** (0x01U << BCC_RW_I_MUX_SHIFT)
- #define **BCC_IMUX_DIFF_REF** (0x02U << BCC_RW_I_MUX_SHIFT)
- #define **BCC_IMUX_PGA_ZERO** (0x03U << BCC_RW_I_MUX_SHIFT)
- #define **BCC_FAULT_PIN_PACK_CTRL** (0x00U << BCC_RW_FAULT_DIAG_SHIFT)
- #define **BCC_FAULT_PIN_FORCED_HIGH** (0x01U << BCC_RW_FAULT_DIAG_SHIFT)
- #define **BCC_REG_ADC_CFG_ADDR** 0x06U
- #define **BCC_REG_ADC_CFG_DEFAULT** 0x0417U
- #define **BCC_RW_ADC2_DEF_MASK** 0x0003U
- #define **BCC_RW_ADC1_B_DEF_MASK** 0x000CU
- #define **BCC_RW_ADC1_A_DEF_MASK** 0x0030U
- #define **BCC_RW_DIS_CH_COMP_MASK** 0x0040U
- #define **BCC_W_CC_RST_MASK** 0x0080U
- #define **BCC_W_PGA_GAIN_MASK** 0x0700U
- #define **BCC_R_PGA_GAIN_S_MASK** 0x0700U
- #define **BCC_W_SOC_MASK** 0x0800U
- #define **BCC_R_EOC_N_MASK** 0x0800U
- #define **BCC_RW_TAG_ID_MASK** 0xF000U
- #define **BCC_RW_ADC2_DEF_SHIFT** 0x00U
- #define **BCC_RW_ADC1_B_DEF_SHIFT** 0x02U
- #define **BCC_RW_ADC1_A_DEF_SHIFT** 0x04U
- #define **BCC_RW_DIS_CH_COMP_SHIFT** 0x06U
- #define **BCC_W_CC_RST_SHIFT** 0x07U
- #define **BCC_W_PGA_GAIN_SHIFT** 0x08U
- #define **BCC_R_PGA_GAIN_S_SHIFT** 0x08U
- #define **BCC_W_SOC_SHIFT** 0x0BU
- #define **BCC_R_EOC_N_SHIFT** 0x0BU
- #define **BCC_RW_TAG_ID_SHIFT** 0x0CU
- #define **BCC_ADC2_RES_13BIT** (0x00U << BCC_RW_ADC2_DEF_SHIFT)
- #define **BCC_ADC2_RES_14BIT** (0x01U << BCC_RW_ADC2_DEF_SHIFT)
- #define **BCC_ADC2_RES_15BIT** (0x02U << BCC_RW_ADC2_DEF_SHIFT)
- #define **BCC_ADC2_RES_16BIT** (0x03U << BCC_RW_ADC2_DEF_SHIFT)
- #define **BCC_ADC1_B_RES_13BIT** (0x00U << BCC_RW_ADC1_B_DEF_SHIFT)
- #define **BCC_ADC1_B_RES_14BIT** (0x01U << BCC_RW_ADC1_B_DEF_SHIFT)
- #define **BCC_ADC1_B_RES_15BIT** (0x02U << BCC_RW_ADC1_B_DEF_SHIFT)
- #define **BCC_ADC1_B_RES_16BIT** (0x03U << BCC_RW_ADC1_B_DEF_SHIFT)
- #define **BCC_ADC1_A_RES_13BIT** (0x00U << BCC_RW_ADC1_A_DEF_SHIFT)
- #define **BCC_ADC1_A_RES_14BIT** (0x01U << BCC_RW_ADC1_A_DEF_SHIFT)
- #define **BCC_ADC1_A_RES_15BIT** (0x02U << BCC_RW_ADC1_A_DEF_SHIFT)
- #define **BCC_ADC1_A_RES_16BIT** (0x03U << BCC_RW_ADC1_A_DEF_SHIFT)

- #define **BCC_CHAR_COMP_ENABLED** (0x00U << BCC_RW_DIS_CH_COMP_SHIFT)
- #define **BCC_CHAR_COMP_DISABLED** (0x01U << BCC_RW_DIS_CH_COMP_SHIFT)
- #define **BCC_CC_RESET** (0x01U << BCC_W_CC_RST_SHIFT)
- #define **BCC_ADC2_PGA_4** (0x00U << BCC_W_PGA_GAIN_SHIFT)
- #define **BCC_ADC2_PGA_16** (0x01U << BCC_W_PGA_GAIN_SHIFT)
- #define **BCC_ADC2_PGA_64** (0x02U << BCC_W_PGA_GAIN_SHIFT)
- #define **BCC_ADC2_PGA_256** (0x03U << BCC_W_PGA_GAIN_SHIFT)
- #define **BCC_ADC2_PGA_AUTO** (0x04U << BCC_W_PGA_GAIN_SHIFT)
- #define **BCC_INIT_CONV_SEQ** (0x01U << BCC_W_SOC_SHIFT)
- #define **BCC_SET_TAG_ID**(reg, tagID) **BCC_REG_SET_BITS_VALUE**(reg, BCC_RW_TAG_ID_MASK, BCC_RW_TAG_ID_SHIFT, tagID, 0x0FU)
- #define **BCC_REG_ADC2_OFFSET_COMP_ADDR** 0x07U
- #define **BCC_REG_ADC2_OFFSET_COMP_DEFAULT** 0x4000U
- #define **BCC_RW_ADC2_OFFSET_COMP_MASK** 0x00FFU
- #define **BCC_R_CC_OVT_MASK** 0x0400U
- #define **BCC_R_SAMP_OVF_MASK** 0x0800U
- #define **BCC_R_CC_N_OVF_MASK** 0x1000U
- #define **BCC_R_CC_P_OVF_MASK** 0x2000U
- #define **BCC_RW_FREE_CNT_MASK** 0x4000U
- #define **BCC_RW_CC_RST_CFG_MASK** 0x8000U
- #define **BCC_RW_ADC2_OFFSET_COMP_SHIFT** 0x00U
- #define **BCC_R_CC_OVT_SHIFT** 0x0AU
- #define **BCC_R_SAMP_OVF_SHIFT** 0x0BU
- #define **BCC_R_CC_N_OVF_SHIFT** 0x0CU
- #define **BCC_R_CC_P_OVF_SHIFT** 0x0DU
- #define **BCC_RW_FREE_CNT_SHIFT** 0x0EU
- #define **BCC_RW_CC_RST_CFG_SHIFT** 0x0FU
- #define **BCC_SET_ADC2_OFFSET**(reg, offset)
- #define **BCC_GET_ADC2_OFFSET**(offset) (BCC_SET_ADC2_OFFSET(0x00U, (uint16_t)(((offset) * 10U) / 6U)))
- #define **BCC_FREE_CC_CLAMP** (0x00U << BCC_RW_FREE_CNT_SHIFT)
- #define **BCC_FREE_CC_ROLL_OVER** (0x01U << BCC_RW_FREE_CNT_SHIFT)
- #define **BCC_READ_CC_NO_ACTION** (0x00U << BCC_RW_CC_RST_CFG_SHIFT)
- #define **BCC_READ_CC_RESET** (0x01U << BCC_RW_CC_RST_CFG_SHIFT)
- #define **BCC_REG_OV_UV_EN_ADDR** 0x08U
- #define **BCC_REG_OV_UV_EN_DEFAULT** 0x3FFFU
- #define **BCC_RW_CTX_OVUV_EN_MASK**(ctNumber) (0x0001U << ((ctNumber) - 1U))
- #define **BCC_RW_COMMON_UV_TH_MASK** 0x4000U
- #define **BCC_RW_COMMON_OV_TH_MASK** 0x8000U
- #define **BCC_RW_CTX_OVUV_EN_SHIFT**(ctNumber) ((ctNumber) - 1U)
- #define **BCC_RW_COMMON_UV_TH_SHIFT** 0x0EU
- #define **BCC_RW_COMMON_OV_TH_SHIFT** 0x0FU
- #define **BCC_CTX_OVUV_DISABLED**(ctNumber) (0x00U << BCC_RW_CTX_OVUV_EN_SHIFT(ctNumber))
- #define **BCC_CTX_OVUV_ENABLED**(ctNumber) (0x01U << BCC_RW_CTX_OVUV_EN_SHIFT(ctNumber))
- #define **BCC_CTX_UV_TH_INDIVIDUAL** (0x00U << BCC_RW_COMMON_UV_TH_SHIFT)
- #define **BCC_CTX_UV_TH_COMMON** (0x01U << BCC_RW_COMMON_UV_TH_SHIFT)
- #define **BCC_CTX_OV_TH_INDIVIDUAL** (0x00U << BCC_RW_COMMON_OV_TH_SHIFT)
- #define **BCC_CTX_OV_TH_COMMON** (0x01U << BCC_RW_COMMON_OV_TH_SHIFT)
- #define **BCC_REG_CELL_OV_FLT_ADDR** 0x09U
- #define **BCC_RW_CTX_OV_FLT_MASK**(ctNumber) (0x0001U << ((ctNumber) - 1U))
- #define **BCC_REG_CELL_UV_FLT_ADDR** 0x0AU
- #define **BCC_RW_CTX_UV_FLT_MASK**(ctNumber) (0x0001U << ((ctNumber) - 1U))
- #define **BCC_REG_CB1_CFG_ADDR** 0x0CU

- #define **BCC_REG_CB2_CFG_ADDR** 0x0DU
- #define **BCC_REG_CB3_CFG_ADDR** 0x0EU
- #define **BCC_REG_CB4_CFG_ADDR** 0x0FU
- #define **BCC_REG_CB5_CFG_ADDR** 0x10U
- #define **BCC_REG_CB6_CFG_ADDR** 0x11U
- #define **BCC_REG_CB7_CFG_ADDR** 0x12U
- #define **BCC_REG_CB8_CFG_ADDR** 0x13U
- #define **BCC_REG_CB9_CFG_ADDR** 0x14U
- #define **BCC_REG_CB10_CFG_ADDR** 0x15U
- #define **BCC_REG_CB11_CFG_ADDR** 0x16U
- #define **BCC_REG_CB12_CFG_ADDR** 0x17U
- #define **BCC_REG_CB13_CFG_ADDR** 0x18U
- #define **BCC_REG_CB14_CFG_ADDR** 0x19U
- #define **BCC_REG_CBX_CFG_DEFAULT** 0x0000U
- #define **BCC_RW_CB_TIMER_MASK** 0x01FFU
- #define **BCC_W_CB_EN_MASK** 0x0200U
- #define **BCC_R_CB_STS_MASK** 0x0200U
- #define **BCC_RW_CB_TIMER_SHIFT** 0x00U
- #define **BCC_W_CB_EN_SHIFT** 0x09U
- #define **BCC_R_CB_STS_SHIFT** 0x09U
- #define **BCC_SET_CB_TIMER**(reg, minutes) **BCC_REG_SET_BITS_VALUE**(reg, BCC_RW_CB_TIMER_MASK, BCC_RW_CB_TIMER_SHIFT, minutes, 0x01FFU)
- #define **BCC_CB_DISABLED** (0x00U << BCC_W_CB_EN_SHIFT)
- #define **BCC_CB_ENABLED** (0x01U << BCC_W_CB_EN_SHIFT)
- #define **BCC_REG_CB_OPEN_FLT_ADDR** 0x1AU
- #define **BCC_RW_CBX_OPEN_FLT_MASK**(cbNumber) (0x0001U << ((cbNumber) - 1U))
- #define **BCC_REG_CB_SHORT_FLT_ADDR** 0x1BU
- #define **BCC_RW_CBX_SHORT_FLT_MASK**(cbNumber) (0x0001U << ((cbNumber) - 1U))
- #define **BCC_REG_CB_DRV_STS_ADDR** 0x1CU
- #define **BCC_R_CBX_STS_MASK**(cbNumber) (0x0001U << ((cbNumber) - 1U))
- #define **BCC_REG_GPIO_CFG1_ADDR** 0x1DU
- #define **BCC_REG_GPIO_CFG1_DEFAULT** 0x0000U
- #define **BCC_RW_GPIOX_CFG_MASK**(GPIONumber) (0x0003U << ((GPIONumber) * 2))
- #define **BCC_GPIOX_CFG_SHIFT**(GPIONumber) ((GPIONumber) * 2)
- #define **BCC_GPIOX_AN_IN_RM_MEAS**(gpioNumber) (0x00U << BCC_GPIOX_CFG_SHIFT(gpioNumber))
- #define **BCC_GPIOX_AN_IN_ABS_MEAS**(gpioNumber) (0x01U << BCC_GPIOX_CFG_SHIFT(gpioNumber))
- #define **BCC_GPIOX_DIG_IN**(gpioNumber) (0x02U << BCC_GPIOX_CFG_SHIFT(gpioNumber))
- #define **BCC_GPIOX_DIG_OUT**(gpioNumber) (0x03U << BCC_GPIOX_CFG_SHIFT(gpioNumber))
- #define **BCC_REG_GPIO_CFG2_ADDR** 0x1EU
- #define **BCC_REG_GPIO_CFG2_DEFAULT** 0x0000U
- #define **BCC_RW_GPIOX_DR_MASK**(gpioNumber) (0x0001U << (gpioNumber))
- #define **BCC_RW_GPIO0_FLT_ACT_MASK** 0x0080U
- #define **BCC_RW_GPIO0_WU_MASK** 0x0100U
- #define **BCC_RW_GPIO2_SOC_MASK** 0x0200U
- #define **BCC_RW_GPIO0_FLT_ACT_SHIFT** 0x07U
- #define **BCC_RW_GPIO0_WU_SHIFT** 0x08U
- #define **BCC_RW_GPIO2_SOC_SHIFT** 0x09U
- #define **BCC_GPIOx_LOW**(GPIONumber) (0x00U << (GPIONumber))
- #define **BCC_GPIOx_HIGH**(GPIONumber) (0x01U << (GPIONumber))
- #define **BCC_GPIO0_INP_HIGH_FP_NACT** (0x00U << BCC_RW_GPIO0_FLT_ACT_SHIFT)
- #define **BCC_GPIO0_INP_HIGH_FP_ACT** (0x01U << BCC_RW_GPIO0_FLT_ACT_SHIFT)

- #define **BCC_GPIO0_NO_WAKE_UP** (0x00U << BCC_RW_GPIO0_WU_SHIFT)
- #define **BCC_GPIO0_WAKE_UP** (0x01U << BCC_RW_GPIO0_WU_SHIFT)
- #define **BCC_GPIO2_ADC_TRG_DISABLED** (0x00U << BCC_RW_GPIO2_SOC_SHIFT)
- #define **BCC_GPIO2_ADC_TRG_ENABLED** (0x01U << BCC_RW_GPIO2_SOC_SHIFT)
- #define **BCC_REG_GPIO_STS_ADDR** 0x1FU
- #define **BCC_R_GPIOX_ST_MASK**(gpioNumber) (0x0001U << (gpioNumber))
- #define **BCC_RW_GPIOX_H_MASK**(gpioNumber) (0x0001U << ((gpioNumber) + 8U))
- #define **BCC_REG_AN_OT_UT_FLT_ADDR** 0x20U
- #define **BCC_RW_ANX_UT_MASK**(anNumber) (0x0001U << (anNumber))
- #define **BCC_RW_AN_UT_MASK** 0x007F
- #define **BCC_RW_ANX_OT_MASK**(anNumber) (0x0001U << ((anNumber) + 8U))
- #define **BCC_RW_AN_OT_MASK** 0x7F00
- #define **BCC_REG_GPIO_SHORT_ADDR** 0x21U
- #define **BCC_RW_ANX_OPEN_MASK**(anNumber) (0x0001U << (anNumber))
- #define **BCC_RW_AN_OPEN_MASK** 0x007F
- #define **BCC_RW_GPIOX_SH_MASK**(gpioNumber) (0x0001U << ((gpioNumber) + 8U))
- #define **BCC_REG_I_STATUS_ADDR** 0x22U
- #define **BCC_R_PGA_DAC_MASK** 0xFF00U
- #define **BCC_R_PGA_DAC_SHIFT** 0x08U
- #define **BCC_REG_COM_STATUS_ADDR** 0x23U
- #define **BCC_R_COM_ERR_COUNT_MASK** 0xFF00U
- #define **BCC_R_COM_ERR_COUNT_SHIFT** 0x08U
- #define **BCC_GET_COM_ERR_COUNT**(reg)
- #define **BCC_REG_FAULT1_STATUS_ADDR** 0x24U
- #define **BCC_R_CT_UV_FLT_MASK** 0x0001U
- #define **BCC_R_CT_OV_FLT_MASK** 0x0002U
- #define **BCC_R_AN_UT_FLT_MASK** 0x0004U
- #define **BCC_R_AN_OT_FLT_MASK** 0x0008U
- #define **BCC_RW_IS_OC_FLT_MASK** 0x0010U
- #define **BCC_RW_IS_OL_FLT_MASK** 0x0020U
- #define **BCC_RW_I2C_ERR_FLT_MASK** 0x0040U
- #define **BCC_RW_GPIO0_WUP_FLT_MASK** 0x0080U
- #define **BCC_RW_CSB_WUP_FLT_MASK** 0x0100U
- #define **BCC_RW_COM_ERR_FLT_MASK** 0x0200U
- #define **BCC_RW_COM_LOSS_FLT_MASK** 0x0400U
- #define **BCC_RW_VPWR_LV_FLT_MASK** 0x0800U
- #define **BCC_RW_VPWR_OV_FLT_MASK** 0x1000U
- #define **BCC_RW_COM_ERR_OVR_FLT_MASK** 0x2000U
- #define **BCC_RW_RESET_FLT_MASK** 0x4000U
- #define **BCC_RW_POR_MASK** 0x8000U
- #define **BCC_REG_FAULT2_STATUS_ADDR** 0x25U
- #define **BCC_RW_FUSE_ERR_FLT_MASK** 0x0001U
- #define **BCC_RW_DED_ERR_FLT_MASK** 0x0002U
- #define **BCC_RW_OSC_ERR_FLT_MASK** 0x0004U
- #define **BCC_R_CB_OPEN_FLT_MASK** 0x0008U
- #define **BCC_R_CB_SHORT_FLT_MASK** 0x0010U
- #define **BCC_R_GPIO_SHORT_FLT_MASK** 0x0020U
- #define **BCC_R_AN_OPEN_FLT_MASK** 0x0040U
- #define **BCC_RW_IDLE_MODE_FLT_MASK** 0x0080U
- #define **BCC_RW_IC_TSD_FLT_MASK** 0x0100U
- #define **BCC_RW_GND_LOSS_FLT_MASK** 0x0200U
- #define **BCC_RW_ADC1_A_FLT_MASK** 0x0400U
- #define **BCC_RW_ADC1_B_FLT_MASK** 0x0800U
- #define **BCC_RW_VANA_UV_FLT_MASK** 0x1000U
- #define **BCC_RW_VANA_OV_FLT_MASK** 0x2000U
- #define **BCC_RW_VCOM_UV_FLT_MASK** 0x4000U
- #define **BCC_RW_VCOM_OV_FLT_MASK** 0x8000U

- #define **BCC_REG_FAULT3_STATUS_ADDR** 0x26U
- #define **BCC_RW_EOT_CBX_MASK**(cbNumber) (0x0001U << ((cbNumber) - 1U))
- #define **BCC_R_VCP_UV_MASK** 0x2000U /* MC33772 only. */
- #define **BCC_R_DIAG_TO_FLT_MASK** 0x4000U
- #define **BCC_R_CC_OVR_FLT_MASK** 0x8000U
- #define **BCC_REG_FAULT_MASK1_ADDR** 0x27U
- #define **BCC_REG_FAULT_MASK1_DEFAULT** 0x0000U
- #define **BCC_CT_UV_FLT_EN** 0x0000U
- #define **BCC_CT_UV_FLT_DIS** 0x0001U
- #define **BCC_CT_OV_FLT_EN** 0x0000U
- #define **BCC_CT_OV_FLT_DIS** 0x0002U
- #define **BCC_AN_UT_FLT_EN** 0x0000U
- #define **BCC_AN_UT_FLT_DIS** 0x0004U
- #define **BCC_AN_OT_FLT_EN** 0x0000U
- #define **BCC_AN_OT_FLT_DIS** 0x0008U
- #define **BCC_IS_OC_FLT_EN** 0x0000U
- #define **BCC_IS_OC_FLT_DIS** 0x0010U
- #define **BCC_IS_OL_FLT_EN** 0x0000U
- #define **BCC_IS_OL_FLT_DIS** 0x0020U
- #define **BCC_I2C_ERR_FLT_EN** 0x0000U
- #define **BCC_I2C_ERR_FLT_DIS** 0x0040U
- #define **BCC_GPIO0_WUP_FLT_EN** 0x0000U
- #define **BCC_GPIO0_WUP_FLT_DIS** 0x0080U
- #define **BCC_CSB_WUP_FLT_EN** 0x0000U
- #define **BCC_CSB_WUP_FLT_DIS** 0x0100U
- #define **BCC_COM_ERR_FLT_EN** 0x0000U
- #define **BCC_COM_ERR_FLT_DIS** 0x0200U
- #define **BCC_COM_LOSS_FLT_EN** 0x0000U
- #define **BCC_COM_LOSS_FLT_DIS** 0x0400U
- #define **BCC_VPWR_LV_FLT_EN** 0x0000U
- #define **BCC_VPWR_LV_FLT_DIS** 0x0800U
- #define **BCC_VPWR_OV_FLT_EN** 0x0000U
- #define **BCC_VPWR_OV_FLT_DIS** 0x1000U
- #define **BCC_REG_FAULT_MASK2_ADDR** 0x28U
- #define **BCC_REG_FAULT_MASK2_DEFAULT** 0x0000U
- #define **BCC_FUSE_ERR_FLT_EN** 0x0000U
- #define **BCC_FUSE_ERR_FLT_DIS** 0x0001U
- #define **BCC_DED_ERR_FLT_EN** 0x0000U
- #define **BCC_DED_ERR_FLT_DIS** 0x0002U
- #define **BCC_OSC_ERR_FLT_EN** 0x0000U
- #define **BCC_OSC_ERR_FLT_DIS** 0x0004U
- #define **BCC_CB_OPEN_FLT_EN** 0x0000U
- #define **BCC_CB_OPEN_FLT_DIS** 0x0008U
- #define **BCC_CB_SHORT_FLT_EN** 0x0000U
- #define **BCC_CB_SHORT_FLT_DIS** 0x0010U
- #define **BCC_GPIO_SHORT_FLT_EN** 0x0000U
- #define **BCC_GPIO_SHORT_FLT_DIS** 0x0020U
- #define **BCC_AN_OPEN_FLT_EN** 0x0000U
- #define **BCC_AN_OPEN_FLT_DIS** 0x0040U
- #define **BCC_GND_LOSS_FLT_EN** 0x0000U
- #define **BCC_GND_LOSS_FLT_DIS** 0x0200U
- #define **BCC_ADC1_A_FLT_EN** 0x0000U
- #define **BCC_ADC1_A_FLT_DIS** 0x0400U
- #define **BCC_ADC1_B_FLT_EN** 0x0000U
- #define **BCC_ADC1_B_FLT_DIS** 0x0800U
- #define **BCC_VANA_UV_FLT_EN** 0x0000U
- #define **BCC_VANA_UV_FLT_DIS** 0x1000U

- #define **BCC_VANA_OV_FLT_EN** 0x0000U
- #define **BCC_VANA_OV_FLT_DIS** 0x2000U
- #define **BCC_VCOM_UV_FLT_EN** 0x0000U
- #define **BCC_VCOM_UV_FLT_DIS** 0x4000U
- #define **BCC_VCOM_OV_FLT_EN** 0x0000U
- #define **BCC_VCOM_OV_FLT_DIS** 0x8000U
- #define **BCC_REG_FAULT_MASK3_ADDR** 0x29U
- #define **BCC_REG_FAULT_MASK3_DEFAULT** 0x0000U
- #define **BCC_EOT_CBX_FLT_DIS**(cbNumber) (0x0001U << ((cbNumber) - 1U))
- #define **BCC_EOT_CBX_FLT_EN** 0x0000U
- #define **BCC_VCP_UV_EN** 0x0000U /* MC33772 only. */
- #define **BCC_VCP_UV_DIS** 0x2000U /* MC33772 only. */
- #define **BCC_DIAG_TO_FLT_EN** 0x0000U
- #define **BCC_DIAG_TO_FLT_DIS** 0x4000U
- #define **BCC_CC_OVR_FLT_EN** 0x0000U
- #define **BCC_CC_OVR_FLT_DIS** 0x8000U
- #define **BCC_REG_WAKEUP_MASK1_ADDR** 0x2AU
- #define **BCC_REG_WAKEUP_MASK1_DEFAULT** 0x0000U
- #define **BCC_CT_UV_WAKEUP_EN** 0x0000U
- #define **BCC_CT_UV_WAKEUP_DIS** 0x0001U
- #define **BCC_CT_OV_WAKEUP_EN** 0x0000U
- #define **BCC_CT_OV_WAKEUP_DIS** 0x0002U
- #define **BCC_AN_UT_WAKEUP_EN** 0x0000U
- #define **BCC_AN_UT_WAKEUP_DIS** 0x0004U
- #define **BCC_AN_OT_WAKEUP_EN** 0x0000U
- #define **BCC_AN_OT_WAKEUP_DIS** 0x0008U
- #define **BCC_IS_OC_WAKEUP_EN** 0x0000U
- #define **BCC_IS_OC_WAKEUP_DIS** 0x0010U
- #define **BCC_GPIO0_WUP_WAKEUP_EN** 0x0000U
- #define **BCC_GPIO0_WUP_WAKEUP_DIS** 0x0080U
- #define **BCC_CSB_WUP_WAKEUP_EN** 0x0000U
- #define **BCC_CSB_WUP_WAKEUP_DIS** 0x0100U
- #define **BCC_VPWR_LV_WAKEUP_EN** 0x0000U
- #define **BCC_VPWR_LV_WAKEUP_DIS** 0x0800U
- #define **BCC_VPWR_OV_WAKEUP_EN** 0x0000U
- #define **BCC_VPWR_OV_WAKEUP_DIS** 0x1000U
- #define **BCC_REG_WAKEUP_MASK2_ADDR** 0x2BU
- #define **BCC_REG_WAKEUP_MASK2_DEFAULT** 0x0000U
- #define **BCC_DED_ERR_WAKEUP_EN** 0x0000U
- #define **BCC_DED_ERR_WAKEUP_DIS** 0x0002U
- #define **BCC_OSC_ERR_WAKEUP_EN** 0x0000U
- #define **BCC_OSC_ERR_WAKEUP_DIS** 0x0004U
- #define **BCC_CB_SHORT_WAKEUP_EN** 0x0000U
- #define **BCC_CB_SHORT_WAKEUP_DIS** 0x0010U
- #define **BCC_GPIO_SHORT_WAKEUP_EN** 0x0000U
- #define **BCC_GPIO_SHORT_WAKEUP_DIS** 0x0020U
- #define **BCC_IC_TSD_WAKEUP_EN** 0x0000U
- #define **BCC_IC_TSD_WAKEUP_DIS** 0x0100U
- #define **BCC_GND_LOSS_WAKEUP_EN** 0x0000U
- #define **BCC_GND_LOSS_WAKEUP_DIS** 0x0200U
- #define **BCC_ADC1_A_WAKEUP_EN** 0x0000U
- #define **BCC_ADC1_A_WAKEUP_DIS** 0x0400U
- #define **BCC_ADC1_B_WAKEUP_EN** 0x0000U
- #define **BCC_ADC1_B_WAKEUP_DIS** 0x0800U
- #define **BCC_VANA_UV_WAKEUP_EN** 0x0000U
- #define **BCC_VANA_UV_WAKEUP_DIS** 0x1000U
- #define **BCC_VANA_OV_WAKEUP_EN** 0x0000U

- #define **BCC_VANA_OV_WAKEUP_DIS** 0x2000U
- #define **BCC_VCOM_UV_WAKEUP_EN** 0x0000U
- #define **BCC_VCOM_UV_WAKEUP_DIS** 0x4000U
- #define **BCC_VCOM_OV_WAKEUP_EN** 0x0000U
- #define **BCC_VCOM_OV_WAKEUP_DIS** 0x8000U
- #define **BCC_REG_WAKEUP_MASK3_ADDR** 0x2CU
- #define **BCC_REG_WAKEUP_MASK3_DEFAULT** 0x0000U
- #define **BCC_EOT_CBX_WAKEUP_DIS**(cbNumber) (0x0001U << ((cbNumber) - 1U))
- #define **BCC_EOT_CBX_WAKEUP_EN** 0x0000U
- #define **BCC_VCP_UV_WAKEUP_EN** 0x0000U /* MC33772 only. */
- #define **BCC_VCP_UV_WAKEUP_DIS** 0x2000U /* MC33772 only. */
- #define **BCC_DIAG_TO_WAKEUP_EN** 0x0000U
- #define **BCC_DIAG_TO_WAKEUP_DIS** 0x4000U
- #define **BCC_CC_OVR_WAKEUP_EN** 0x0000U
- #define **BCC_CC_OVR_WAKEUP_DIS** 0x8000U
- #define **BCC_REG_CC_NB_SAMPLES_ADDR** 0x2DU
- #define **BCC_R_CC_NB_SAMPLES_MASK** 0xFFFFU
- #define **BCC_REG_COULOMB_CNT1_ADDR** 0x2EU
- #define **BCC_R_COULOMB_CNT_MSB_MASK** 0xFFFFU
- #define **BCC_REG_COULOMB_CNT2_ADDR** 0x2FU
- #define **BCC_R_COULOMB_CNT_LSB_MASK** 0xFFFFU
- #define **BCC_GET_COULOMB_CNT**(coulombCnt1, coulombCnt2)
- #define **BCC_REG_MEAS_ISENSE1_ADDR** 0x30U
- #define **BCC_R_MEAS1_I_MASK** 0x7FFFU
- #define **BCC_REG_MEAS_ISENSE2_ADDR** 0x31U
- #define **BCC_R_MEAS2_I_MASK** 0x000FU
- #define **BCC_RW_PGA_GCHANGE_MASK** 0x0040U
- #define **BCC_RW_ADC2_SAT_MASK** 0x0080U
- #define **BCC_R_PGA_GAIN_MASK** 0x0300U
- #define **BCC_GET_ISENSE_RAW**(measISense1, measISense2)
- #define **BCC_GET_ISENSE_RAW_SIGN**(iSenseRaw) ((int32_t)(((iSenseRaw) & 0x040000U) ? ((iSenseRaw) | 0xFFF80000U) : (iSenseRaw)))
- #define **BCC_REG_MEAS_STACK_ADDR** 0x32U
- #define **BCC_R_MEAS_MASK** 0x7FFFU
- #define **BCC_R_DATA_RDY_MASK** 0x8000U
- #define **BCC_GET_MEAS_RAW**(reg) ((reg) & BCC_R_MEAS_MASK)
- #define **BCC_REG_MEAS_CELLX_ADDR_MC33771_START** 0x33U
- #define **BCC_REG_MEAS_CELLX_ADDR_MC33772_START** 0x3BU
- #define **BCC_REG_MEAS_CELLX_ADDR_END** 0x40U
- #define **BCC_REG_MEAS_ANX_ADDR_START** 0x41U
- #define **BCC_REG_MEAS_ANX_ADDR_END** 0x47U
- #define **BCC_REG_MEAS_IC_TEMP_ADDR** 0x48U
- #define **BCC_REG_MEAS_VBG_DIAG_ADC1A_ADDR** 0x49U
- #define **BCC_REG_MEAS_VBG_DIAG_ADC1B_ADDR** 0x4AU
- #define **BCC_REG_TH_ALL_CT_ADDR** 0x4BU
- #define **BCC_REG_TH_ALL_CT_DEFAULT** 0xD780U
- #define **BCC_RW_ALL_CT_UV_TH_MASK** 0x00FFU
- #define **BCC_RW_ALL_CT_OV_TH_MASK** 0xFF00U
- #define **BCC_RW_ALL_CT_UV_TH_SHIFT** 0x00U
- #define **BCC_RW_ALL_CT_OV_TH_SHIFT** 0x08U
- #define **BCC_SET_ALL_CT_UV_TH**(threshold) ((uint8_t)(((threshold) * 10U) / 195U) << BCC_RW_ALL_CT_UV_TH_SHIFT)
- #define **BCC_ALL_CT_UV_TH_DEFAULT** 0x80U
- #define **BCC_SET_ALL_CT_OV_TH**(threshold) ((uint16_t)(((threshold) * 10U) / 195U) << BCC_RW_ALL_CT_OV_TH_SHIFT)
- #define **BCC_ALL_CT_OV_TH_DEFAULT** 0xD7U
- #define **BCC_REG_TH_CT14_ADDR** 0x4CU

- #define **BCC_REG_TH_CT13_ADDR** 0x4DU
- #define **BCC_REG_TH_CT12_ADDR** 0x4EU
- #define **BCC_REG_TH_CT11_ADDR** 0x4FU
- #define **BCC_REG_TH_CT10_ADDR** 0x50U
- #define **BCC_REG_TH_CT9_ADDR** 0x51U
- #define **BCC_REG_TH_CT8_ADDR** 0x52U
- #define **BCC_REG_TH_CT7_ADDR** 0x53U
- #define **BCC_REG_TH_CT6_ADDR** 0x54U
- #define **BCC_REG_TH_CT5_ADDR** 0x55U
- #define **BCC_REG_TH_CT4_ADDR** 0x56U
- #define **BCC_REG_TH_CT3_ADDR** 0x57U
- #define **BCC_REG_TH_CT2_ADDR** 0x58U
- #define **BCC_REG_TH_CT1_ADDR** 0x59U
- #define **BCC_REG_TH_CTX_DEFAULT** 0xD780U
- #define **BCC_RW_CTX_UV_TH_MASK** 0x00FFU
- #define **BCC_RW_CTX_OV_TH_MASK** 0xFF00U
- #define **BCC_RW_CTX_UV_TH_SHIFT** 0x00U
- #define **BCC_RW_CTX_OV_TH_SHIFT** 0x08U
- #define **BCC_SET_CTX_UV_TH**(threshold)
- #define **BCC_CTX_UV_TH_DEFAULT** 0x80U
- #define **BCC_SET_CTX_OV_TH**(threshold)
- #define **BCC_CTX_OV_TH_DEFAULT** 0xD7U
- #define **BCC_REG_TH_AN6_OT_ADDR** 0x5AU
- #define **BCC_REG_TH_AN5_OT_ADDR** 0x5BU
- #define **BCC_REG_TH_AN4_OT_ADDR** 0x5CU
- #define **BCC_REG_TH_AN3_OT_ADDR** 0x5DU
- #define **BCC_REG_TH_AN2_OT_ADDR** 0x5EU
- #define **BCC_REG_TH_AN1_OT_ADDR** 0x5FU
- #define **BCC_REG_TH_AN0_OT_ADDR** 0x60U
- #define **BCC_REG_TH_ANX_OT_DEFAULT** 0x00EDU
- #define **BCC_RW_ANX_OT_TH_MASK** 0x03FFU
- #define **BCC_RW_ANX_OT_TH_SHIFT** 0x00U
- #define **BCC_SET_ANX_OT_TH**(threshold)
- #define **BCC_ANX_OT_TH_DEFAULT** 0x0EDU
- #define **BCC_REG_TH_AN6_UT_ADDR** 0x61U
- #define **BCC_REG_TH_AN5_UT_ADDR** 0x62U
- #define **BCC_REG_TH_AN4_UT_ADDR** 0x63U
- #define **BCC_REG_TH_AN3_UT_ADDR** 0x64U
- #define **BCC_REG_TH_AN2_UT_ADDR** 0x65U
- #define **BCC_REG_TH_AN1_UT_ADDR** 0x66U
- #define **BCC_REG_TH_AN0_UT_ADDR** 0x67U
- #define **BCC_REG_TH_ANX_UT_DEFAULT** 0x030EU
- #define **BCC_RW_ANX_UT_TH_MASK** 0x03FFU
- #define **BCC_RW_ANX_UT_TH_SHIFT** 0x00U
- #define **BCC_SET_ANX_UT_TH**(threshold)
- #define **BCC_ANX_UT_TH_DEFAULT** 0x30EU
- #define **BCC_REG_TH_ISENSE_OC_ADDR** 0x68U
- #define **BCC_REG_TH_ISENSE_OC_DEFAULT** 0x00U
- #define **BCC_RW_TH_ISENSE_OC_MASK** 0x0FFFU
- #define **BCC_RW_TH_ISENSE_OC_SHIFT** 0x00U
- #define **BCC_SET_TH_ISENSE_OC**(threshold)
- #define **BCC_REG_TH_COULOMB_CNT_MSB_ADDR** 0x69U
- #define **BCC_RW_TH_COULOMB_CNT_MSB_MASK** 0xFFFFU
- #define **BCC_RW_TH_COULOMB_CNT_MSB_SHIFT** 0x00U
- #define **BCC_SET_TH_COULOMB_CNT_MSB**(threshold) ((uint16_t)((((threshold) >> 16U) * 10U) / 6U) << BCC_RW_TH_COULOMB_CNT_MSB_SHIFT)
- #define **BCC_REG_TH_COULOMB_CNT_LSB_ADDR** 0x6AU

- #define **BCC_RW_TH_COULOMB_CNT_LSB_MASK** 0xFFFFU
- #define **BCC_RW_TH_COULOMB_CNT_LSB_SHIFT** 0x00U
- #define **BCC_SET_TH_COULOMB_CNT_LSB**(threshold) ((uint16_t)((((threshold) && 0xFFFF) * 10) / 6) << BCC_RW_TH_COULOMB_CNT_LSB_SHIFT)
- #define **BCC_REG_SILICON_REV_ADDR** 0x6BU
- #define **BCC_R_MREV_MASK** 0x0007U
- #define **BCC_R_FREV_MASK** 0x0038U
- #define **BCC_REG_EEPROM_CTRL_ADDR** 0x6CU
- #define **BCC_R_READ_DATA_MASK** 0x00FFU
- #define **BCC_R_EE_PRESENT_MASK** 0x2000U
- #define **BCC_R_ERROR_MASK** 0x4000U
- #define **BCC_R_BUSY_MASK** 0x8000U
- #define **BCC_W_DATA_TO_WRITE_MASK** 0x00FFU
- #define **BCC_W_EEPROM_ADD_MASK** 0x7F00U
- #define **BCC_W_RW_MASK** 0x8000U
- #define **BCC_R_READ_DATA_SHIFT** 0x00U
- #define **BCC_R_EE_PRESENT_SHIFT** 0x0DU
- #define **BCC_R_ERROR_SHIFT** 0x0EU
- #define **BCC_R_BUSY_SHIFT** 0x0FU
- #define **BCC_W_DATA_TO_WRITE_SHIFT** 0x00U
- #define **BCC_W_EEPROM_ADD_SHIFT** 0x08U
- #define **BCC_W_RW_SHIFT** 0x0FU
- #define **BCC_EEPROM_RW_W** (0x00U << BCC_W_RW_SHIFT)
- #define **BCC_EEPROM_RW_R** (0x01U << BCC_W_RW_SHIFT)
- #define **BCC_REG_DED_ENCODE1_ADDR** 0x6DU
- #define **BCC_R_DED_HAMMING1_MASK** 0xFFFFU
- #define **BCC_REG_DED_ENCODE2_ADDR** 0x6EU
- #define **BCC_R_DED_HAMMING2_MASK** 0xFF00U
- #define **BCC_REG_FUSE_MIRROR_DATA_ADDR** 0x6FU
- #define **BCC_RW_FMR_DATA_MASK** 0xFFFFU
- #define **BCC_REG_FUSE_MIRROR_CTRL_ADDR** 0x70U
- #define **BCC_R_FST_ST_MASK** 0x0007U
- #define **BCC_RW_FMR_ADDR_MASK** 0x1F00U
- #define **BCC_R_SEC_ERR_FLT_MASK** 0x8000U
- #define **BCC_W_FST_MASK** 0x0007U
- #define **BCC_W_FSTM_MASK** 0x0008U
- #define **BCC_R_FST_ST_SHIFT** 0x00U
- #define **BCC_RW_FMR_ADDR_SHIFT** 0x08U
- #define **BCC_R_SEC_ERR_FLT_SHIFT** 0x0FU
- #define **BCC_W_FST_SHIFT** 0x00U
- #define **BCC_W_FSTM_SHIFT** 0x03U
- #define **BCC_FSTM_WRITE_EN** (0x01U << BCC_W_FSTM_SHIFT)
- #define **BCC_FSTM_WRITE_DIS** (0x00U << BCC_W_FSTM_SHIFT)
- #define **BCC_FST_EN_SPI_WRITE** (0x00U << BCC_W_FST_SHIFT)
- #define **BCC_FST_LP** (0x04U << BCC_W_FST_SHIFT)
- #define **BCC_FUSE_TR_0_ADDR_MC33771** 0x18U
- #define **BCC_FUSE_TR_1_ADDR_MC33771** 0x19U
- #define **BCC_FUSE_TR_2_ADDR_MC33771** 0x1AU
- #define **BCC_FUSE_TR_0_ADDR_MC33772** 0x10U
- #define **BCC_FUSE_TR_1_ADDR_MC33772** 0x11U
- #define **BCC_FUSE_TR_2_ADDR_MC33772** 0x12U
- #define **BCC_FUSE_TR_0_MASK** 0xFFFFU
- #define **BCC_FUSE_TR_1_MASK** 0xFFFFU
- #define **BCC_FUSE_TR_2_MASK** 0x001FU

Detailed Description

This header file contains register map for the MC33771B and MC33772B Battery Cell Controllers.

Macro Definition Documentation

#define BCC_GET_COM_ERR_COUNT(reg)

```
Value:(BCC_REG_GET_BITS_VALUE(reg, BCC_R_COM_ERR_COUNT_MASK, \
          BCC_R_COM_ERR_COUNT_SHIFT))
```

This macro returns COM_ERR_COUNT bits from raw value of COM_STATUS register.

Parameters:

reg	Unsigned 16 bit raw value of COM_STATUS register.

Returns:

Unsigned 8 bit value containing number of communication errors detected.

#define BCC_GET_COULOMB_CNT(coulombCnt1, coulombCnt2)

```
Value:((int32_t)(((uint32_t)((coulombCnt1) & BCC_R_COULOMB_CNT_MSB_MASK) << 0xFU) |
\
  (((uint32_t)(coulombCnt2) & BCC_R_COULOMB_CNT_LSB_MASK))))
```

This macro returns 32 bit signed value of COULOMB_CNT. Note: COULOMB_CNT1 register represents higher part of final value (MSB). COULOMB_CNT2 is lower part (LSB).

Parameters:

coulombCnt1	Content of register COULOMB_CNT1.
coulombCnt2	Content of register COULOMB_CNT2.

#define BCC_GET_ISENSE_RAW(measISense1, measISense2)

```
Value:((((uint32_t)(measISense1) & BCC_R_MEAS1_I_MASK) << 4U) | \
     ((uint32_t)(measISense2) & BCC_R_MEAS2_I_MASK))
```

This macro returns 32 bit unsigned value of ISENSE. Note: MEAS_I from MEAS_ISENSE1 register represents higher part of final value (MSB). MEAS_I from MEAS_ISENSE2 is lower part (LSB).

Parameters:

measISense1	Content of register MEAS_ISENSE1.
measISense2	Content of register MEAS_ISENSE2.

#define BCC_GET_ISENSE_RAW_SIGN(iSenseRaw) ((int32_t)(((iSenseRaw) & 0x040000U) ? ((iSenseRaw) | 0xFFF80000U) : (iSenseRaw)))

This macro converts two 19 bit value (result of BCC_GET_ISENSE_RAW macro) to signed 32 bit value. Sign extension is performed.

Parameters:

iSenseRaw	Raw value of measured current (result of BCC_GET_ISENSE_RAW macro).

#define BCC_GET_MEAS_RAW(reg) ((reg) & BCC_R_MEAS_MASK)

This macro masks register value and returns raw measured value.

Parameters:

reg	Value from a measurement register (MEAS_STACK, MEAS_CELL14~1, MEAS_AN6~0, MEAS_IC_TEMP, MEAS_VBG_DIAG_ADC1A and MEAS_VBG_DIAG_ADC1A registers).

#define BCC_REG_GET_BIT_VALUE(reg, mask, shift) (((reg) & (mask)) >> (shift))

Macro for getting value of the bit given by the mask.

Parameters:

reg	Register to be read.
mask	Bit selection in register.
shift	Bit shift in register.

#define BCC_REG_GET_BITS_VALUE(reg, mask, shift) (((reg) & (mask)) >> (shift))

Macro for getting value of bits given by the mask.

Parameters:

reg	Register to be read.
mask	Bits selection in register.
shift	Bits shift in register.

#define BCC_REG_SET_BIT_VALUE(reg, mask) ((reg) | (mask))

Macro for setting value to 1 of the bit given by the mask.

Parameters:

reg	Register to be modified.
mask	Bit selection in register.

#define BCC_REG_SET_BITS_VALUE(reg, mask, shift, val, range) (((reg) & ~(mask)) | (((val) & (range)) << (shift)))

Macro for setting value of bits given by the mask.

Parameters:

reg	Register value to be modified.
mask	Bits selection in register.
shift	Bits shift in register.
val	Value to be applied.
range	Admissible range of value.

#define BCC_REG_UNSET_BIT_VALUE(reg, mask) ((reg) & ~(mask))

Macro for setting value to 0 of the bit given by the mask.

Parameters:

reg	Register to be modified.
mask	Bit selection in register.

#define BCC_SET_ADC2_OFFSET(reg, offset)

```
Value:BCC_REG_SET_BITS_VALUE(reg, BCC_RW_ADC2_OFFSET_COMP_MASK, \
          BCC_RW_ADC2_OFFSET_COMP_SHIFT, offset, 0xFFU)
```

#define BCC_SET_ANX_OT_TH(threshold)

```
Value:((uint16_t)((((((uint32_t)(threshold)) * 100U) / 488U) > BCC_RW_ANX_OT_TH_MASK) ?
\
   BCC_RW_ANX_OT_TH_MASK : ((((uint32_t)(threshold)) * 100U) / 488U)) <<
BCC_RW_ANX_OT_TH_SHIFT)
```

#define BCC_SET_ANX_UT_TH(threshold)

```
Value:((uint16_t)((((((uint32_t)(threshold)) * 100U) / 488U) > BCC_RW_ANX_UT_TH_MASK) ?
\
   BCC_RW_ANX_UT_TH_MASK : ((((uint32_t)(threshold)) * 100U) / 488U)) <<
BCC_RW_ANX_UT_TH_SHIFT)
```

#define BCC_SET_CID(reg, cid) BCC_REG_SET_BITS_VALUE(reg, BCC_RW_CID_MASK, BCC_RW_CID_SHIFT, cid, 0x0FU)

Sets cluster identifier.

Parameters:

reg	Register value to be modified.
cid	Cluster Identifier [0x00 - 0x0F].

#define BCC_SET_CTX_OV_TH(threshold)

```
Value:((uint16_t)(((((threshold) * 10U) / 196U) > 0x00FF) ?           \
   0x0FF : (((threshold) * 10U) / 196U)) << BCC_RW_CTX_OV_TH_SHIFT)
```

#define BCC_SET_CTX_UV_TH(threshold)

```
Value:((uint16_t)(((((threshold) * 10U) / 196U) > 0x00FF) ?           \
   0x0FF : (((threshold) * 10U) / 196U)) << BCC_RW_CTX_UV_TH_SHIFT)
```

#define BCC_SET_TAG_ID(reg, tagID) BCC_REG_SET_BITS_VALUE(reg, BCC_RW_TAG_ID_MASK, BCC_RW_TAG_ID_SHIFT, tagID, 0x0FU)

The TAG_ID is provided by the system controller during each conversion request. Tag ID should be incremented for each conversion request sent by the system controller. When reading the data for the requested conversion the tag field contains the TAG_ID.

Parameters:

reg	Register to be modified.
tagID	Tag ID provided in conversion [0x00 - 0x0F].

#define BCC_SET_TH_ISENSE_OC(threshold)

```
Value:(((uint16_t)(((threshold) * 5) / 6) << BCC_RW_TH_ISENSE_OC_SHIFT) &  \
          BCC_RW_TH_ISENSE_OC_MASK)
```

bcc/bcc_spi.c File Reference

```
#include "bcc_spi.h"
```

Macros

- #define **BCC_IS_NULL_RESP**(resp)
  Returns true when a response message is equal to zero except CRC field.

Functions

- **bcc_status_t BCC_Reg_ReadSpi** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint8_t regCnt, uint16_t *regVal)
  This function reads a value from addressed register of selected Battery Cell Controller device. Intended for SPI mode only.
- **bcc_status_t BCC_Reg_WriteSpi** (const **bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regVal, uint16_t *retReg)
  This function writes a value to addressed register of selected Battery Cell Controller device. Intended for SPI mode only.
- **bcc_status_t BCC_VerifyComSpi** (const **bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function uses No Operation command of BCC to verify communication without performing any operation. Intended for SPI mode only.

Detailed Description

This file implements low level access functions for SPI communication of BCC driver.

Macro Definition Documentation

#define BCC_IS_NULL_RESP(resp)

```
Value:(((resp)[BCC_MSG_IDX_DATA_H] == 0U) && \
    ((resp)[BCC_MSG_IDX_DATA_L] == 0U) && \
    ((resp)[BCC_MSG_IDX_ADDR] == 0U) && \
    ((resp)[BCC_MSG_IDX_CID_CMD] == 0U))
```

Returns true when a response message is equal to zero except CRC field.

Parameters:

resp	Response message to be checked.

Returns:

True when the response is zero (except CRC), false otherwise.

bcc/bcc_spi.h File Reference

```
#include "bcc_communication.h"
```

Functions

- **bcc_status_t BCC_Reg_ReadSpi** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint8_t regCnt, uint16_t *regVal)
  This function reads a value from addressed register of selected Battery Cell Controller device. Intended for SPI mode only.
- **bcc_status_t BCC_Reg_WriteSpi** (const **bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regVal, uint16_t *retReg)
  This function writes a value to addressed register of selected Battery Cell Controller device. Intended for SPI mode only.
- **bcc_status_t BCC_VerifyComSpi** (const **bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function uses No Operation command of BCC to verify communication without performing any operation. Intended for SPI mode only.

Detailed Description

This file provides access to the functions for SPI communication of BCC driver in SPI mode.

bcc/bcc_tpl.c File Reference

```
#include "bcc_tpl.h"
```

Functions

- **bcc_status_t BCC_Reg_ReadTpl** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint8_t regCnt, uint16_t *regVal)
  This function reads a value from addressed register of selected Battery Cell Controller device. Intended for TPL mode only.
- **bcc_status_t BCC_Reg_WriteTpl** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regVal, uint16_t *retReg)
  This function writes a value to addressed register of selected Battery Cell Controller device. Intended for TPL mode only.
- **bcc_status_t BCC_Reg_WriteGlobalTpl** (**bcc_drv_config_t** *const drvConfig, uint8_t regAddr, uint16_t regVal)
  This function writes a value to addressed register of all configured BCC devices. Intended for TPL mode only.
- **bcc_status_t BCC_VerifyComTpl** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function uses No Operation command of BCC to verify communication without performing any operation. Intended for TPL mode only.

Detailed Description

This file implements low level access functions for SPI communication of BCC driver in TPL mode.

bcc/bcc_tpl.h File Reference

```
#include "bcc_communication.h"
```

Functions

- **bcc_status_t BCC_Reg_ReadTpl** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint8_t regCnt, uint16_t *regVal)
  This function reads a value from addressed register of selected Battery Cell Controller device. Intended for TPL mode only.
- **bcc_status_t BCC_Reg_WriteTpl** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid, uint8_t regAddr, uint16_t regVal, uint16_t *retReg)
  This function writes a value to addressed register of selected Battery Cell Controller device. Intended for TPL mode only.
- **bcc_status_t BCC_Reg_WriteGlobalTpl** (**bcc_drv_config_t** *const drvConfig, uint8_t regAddr, uint16_t regVal)
  This function writes a value to addressed register of all configured BCC devices. Intended for TPL mode only.
- **bcc_status_t BCC_VerifyComTpl** (**bcc_drv_config_t** *const drvConfig, **bcc_cid_t** cid)
  This function uses No Operation command of BCC to verify communication without performing any operation. Intended for TPL mode only.

Detailed Description

This file provides access to the functions for SPI communication of BCC driver in TPL mode.

Index