NXP Semiconductors
User's guide

Document Number: PEXBC3770PUG
Rev. 1.0, 2/2016

MC32BC3770 Programming Guide

Contents

1 General Info

FRDM_BC3770 and BC_MC32BC3770 Processor Expert components are software drivers which encapsulate functionality of MC32BC3770CS Battery charger and FRDM-BC3770-EVB Freedom board. These components create a layer between hardware and user application and enable rapid application development by providing an interface which covers options for charging parameters, settings of registers, measurement and testing.

MC32BC3770CS is a fully programmable switching charger with dual-path output for single-cell Li-Ion and Li-Polymer battery. This dual-path output allows mobile applications with fully discharged or dead battery to boot up the system. The MC32BC3770CS features single 20 V withstanding input and charges the battery with the current up to 2.0 A.

2 Embedded Component Description

2.1 Component API

BC_MC32BC3770 component provides API, which can be used for dynamic real-time configuration of device in user code. Available methods and events are listed under component selection Some of those methods/events are marked with ticks and other ones with crosses, it distinguishes which methods/events are supposed to be generated or not. You can change this setting in Processor Expert Inspector. Note that methods with grey text are always generated because they are needed for proper functionality. This forced behavior depends on various combinations of settings of component properties. For summarization of available API methods and events and their descriptions, see Table 1 BC_MC32BC3770 Component API

Table 1

Method	Description
Init	Initializes the device according to the component properties. This method writes the data according to the component properties into registers via I2C. When auto initialization is enabled, this method will be called automatically within PE initialization function PE_low_level_init().
ReadRegister	Reads data from a single register defined by RegAddr argument. If the method returns ERR_OK, it doesn't mean that reception was successful. The state of reception is detectable by means of events (OnMasterSendComplete or OnError).
WriteRegister	Writes data to a single register defined by RegAddr argument.
ReadBurstData	This method reads data from multiple registers via I2C. The first parameter is an address of the first read register. Addresses of following registers are incremented automatically so incoming bytes of data are content of consecutive registers.
WriteBurstData	This method writes data to multiple registers via I2C. The first parameter is an address of the first register to be written. Addresses of following registers are incremented automatically so outcoming data is written to consecutive registers automatically.
SetInterrupt	Enables or disables an interrupt in INTMSK1..3 register. Interrupts can be set either individually or all at a time. It is not possible to set for example two interrupts at a time.
ClearInterrupt	Clears flag of interrupt in INT1..3 register. Interrupt flags can be cleared either individually or all at a time. It is not possible to clear for example two interrupts.
EnDisComparators	This method enables/disables comparators which are enabled by default. The comparators detect weak battery, supply voltage status, battery OVP, discharge limit.
Reset	This method resets the device registers except INTMASK and STATUS.

SetChargerMode	This method sets charger mode. The charger can be on or off. When the charger is on, it charges the battery or maintains constant voltage on the battery. In Suspend mode PMID output is bypassed to VBUS which means that the charger does not influence output voltage and current. In boost mode the device provides a regulated output voltage to VBUS from the battery. In shutdown mode if there is not valid input source the charger is functional except I2C interface which is turned off in order to minimize power consumption. The device gets in charge mode when valid input source is present.
EnDisShutdown	Enables/disables shutdown pin which means that the device is put in shutdown mode. In shutdown mode I2C interface is turned off in order to minimize power consumption. However, this applies only in case of invalid input power source. This pin is not effective as long as a valid input power source is present.
SetAICLCurrentLimit	This method sets input current limit by writing to VBUSCTRL register. This value limits the fastcharge current when the device is in fastcharge mode. It also sets the limit for adaptiveinput current limit (AICL) when the device in Startup mode automatically starts incrementing the input current limit to either the default or preprogrammed value until either the input current limit is detected or the VBUS voltage detects the AICL threshold, to keep input supply voltage as a valid power source to provide the load for the application.
SetAICLVoltageThreshold	This method sets AICL threshold voltage on VBUS. To keep the device functional with a current and voltage limited VBUS source, the device in Startup mode automatically starts incrementing the input current limit to either the default or preprogrammed value until either the input current limit is detected or the VBUS voltage detects the AICL threshold, to keep input supply voltage as a valid power source to provide the load for the application. The device allows the maximum current the input supply can possibly provide without severely collapsing.
EnDisAICL	Enables/disables adaptiveinput current limit (AICL). AICL is mostly used at the beginning of charging process in cases when current dissipation is higher than the current that the power source can provide. This feature prevents the power source from collapse.
SetPreChargeCurrent	Precharge current is current which charges the battery in precharge mode. The battery charger enters precharge mode when battery voltage is higher than 2.5 V. If the battery voltage does not exceed the VVSYS_MIN threshold before the precharge timer expires, charging is suspended and a fault signal is asserted via the INTB pin.
SetTopOffCurrent	Topoffcharge current is current which charges the battery in topoff mode. After topoff timer expires the topoff event is reported to the processor via the INTB pin which means that the battery is fully charged. As soon as the processor reads the interrupt registers, the processor is able to turn off the charger.
SetFastChargeCurrent	Fastcharge current is current which charges the battery in fastcharge mode. The Fastcharge mode is entered when the battery voltage exceeds the VSYS_MIN threshold of a typical 3.6 V. If the battery voltage does not reach the VBAT_REG threshold before the timer expires, charging is suspended and a fault signal is asserted via the INTB pin.
SetDischargeCurrent	This method sets discharge current limit in discharge mode.

EnDisAutostop	This method enables or disables autostop feature. If autostop is enabled after topoff timer is expired, the charger turns off and goes into DONE state. If it is disabled charger is on continuously and stays in CV mode after topoff timer is expired.
SetWeakBatteryThreshold	This method sets the voltage of weak battery threshold. The threshold ranges from 3.0 V to 3.75 V in 50 mV steps. A weak battery detection function allows the processor to acknowledge the lowbattery condition by asserting INTB event.
SetBatteryRegulationThreshold	Based on this threshold the charger transits from fastcharge mode (constant current mode) to fullcharge mode (constant voltage mode). In fullcharge mode the fast charge current is reduced to a programmable topoff current. Up to this threshold the VSYS output tracks the battery voltage in Trickle and Precharge mode.
SetBoostOTGVoltage	This method sets OTG voltage in Boost (OTG) mode. In boost mode the device provides a regulated output voltage to VBUS from the battery.
SetFastChargeTimer	Fast charge timer watches the device during fast charging mode. If the battery voltage does not reach its required value within this ”fastcharge” time frame an interrupt is asserted. Possible values are 3.5, 4.5, 5.5 hours or the timer can be disabled.
SetTopOffTimer	Topoff timer watches the device during topoff mode. If the battery voltage does not reach its required value within this ”topoff” time frame an interrupt is asserted. Possible values are: 10, 20, 30 or 45 minutes.
GetStatus	This method returns the content of status register.

2.2 Events

OnInterrupt -This event is invoked every time there is a falling edge on the INT interrupt pin. Contents of registers in device structure are updated prior this event so you can read the interrupt registers directly from this structure (without sending I2C command).

ANSIC prototype: void OnInterrupt(void)

2.3 Methods

Init -Initializes the device according to the component properties. This method writes the data according to the component properties into registers via I2C. When auto initialization is enabled, this method will be called automatically within PE initialization function - PE_low_level_init().

ANSIC prototype: TDeviceDataPtr Init(TUserDataPtr UserDataPtr)

TUserDataPtr :UserDataPtr- user data pointer

Return value:TDeviceDataPtr - device data pointer

ReadRegister -Reads data from a single register defined by RegAddr argument. If the method returns ERR_OK, it doesn’t mean that reception was successful. The state of reception is detectable by means of events (OnMasterSendComplete or OnError).

ANSIC prototype: LDD_TError ReadRegister(TRegisterAddress RegAddr,uint8_t *RetValPtr)

TRegisterAddress :RegAddr- an address of register to be read

uint8_t : Pointer to RetValPtr- a pointer to memory where incoming data is stored

Return value:LDD_TError - error code, ERR_OK if successful

WriteRegister -Writes data to a single register defined by RegAddr argument.

ANSIC prototype: LDD_TError WriteRegister(TRegisterAddress RegAddr,uint8_t RegVal)

TRegisterAddress :RegAddr- an address of register to be written

uint8_t :RegVal- a variable where outcoming data is stored

Return value:LDD_TError - error code, ERR_OK if successful

ReadBurstData -This method reads data from multiple registers via I2C. The first parameter is an address of the first read register. Addresses of following registers are incremented automatically so incoming bytes of data are content of consecutive registers.

ANSIC prototype: LDD_TError ReadBurstData(TRegisterAddress RegAddr,uint8_t *DataPtr,uint8_t NumBytes)

TRegisterAddress :RegAddr- an address of first register to be read

uint8_t : Pointer to DataPtr- a pointer to memory where incoming data will be stored

uint8_t :NumBytes- a number of bytes to be read from the device

Return value:LDD_TError - error code, ERR_OK if successful

WriteBurstData -This method writes data to multiple registers via I2C. The first parameter is an address of the first register to be written. Addresses of following registers are incremented automatically so outcoming data is written to consecutive registers automatically.

ANSIC prototype: LDD_TError WriteBurstData(TRegisterAddress RegAddr,uint8_t *DataPtr,uint8_t NumBytes)

TRegisterAddress :RegAddr- an address of first register to be written

uint8_t : Pointer to DataPtr- a pointer to memory where outcoming data is stored

uint8_t :NumBytes- a number of bytes to be written to the device

Return value:LDD_TError - error code, ERR_OK if successful

SetInterrupt -Enables or disables an interrupt in INTMSK1..3 register. Interrupts can be set either individually or all at a time. It is not possible to set for example two interrupts at a time.

ANSIC prototype: LDD_TError SetInterrupt(TInterrupt Interrupt,TEnDisState State)

TInterrupt :Interrupt- an interrupt to be enabled or disabled

TEnDisState :State- possible values: edsENABLED/edsDISABLED, based on this the interrupt will be either enabled or disabled

Return value:LDD_TError - error code, ERR_OK if successful

ClearInterrupt -Clears flag of interrupt in INT1..3 register. Interrupt flags can be cleared either individually or all at a time. It is not possible to clear for example two interrupts.

ANSIC prototype: LDD_TError ClearInterrupt(TInterrupt Interrupt)

TInterrupt :Interrupt- an interrupt to be cleared (or all of them)

Return value:LDD_TError - error code, ERR_OK if successful

EnDisComparators -This method enables/disables comparators which are enabled by default. The comparators detect weak battery, supply voltage status, battery OVP, discharge limit.

ANSIC prototype: LDD_TError EnDisComparators(TEnDisState State)

TEnDisState :State- possible values: edsENABLED/edsDISABLED

Return value:LDD_TError - error code, ERR_OK if successful

Reset -This method resets the device registers except INTMASK and STATUS.

ANSIC prototype: LDD_TError Reset(void)

Return value:LDD_TError - error code, ERR_OK if successful

SetChargerMode -This method sets charger mode. The charger can be on or off. When the charger is on, it charges the battery or maintains constant voltage on the battery. In Suspend mode PMID output is bypassed to VBUS which means that the charger does not influence output voltage and current. In boost mode the device provides a regulated output voltage to VBUS from the battery. In shutdown mode if there is not valid input source the charger is functional except I2C interface which is turned off in order to minimize power consumption. The device gets in charge mode when valid input source is present.

ANSIC prototype: LDD_TError SetChargerMode(TChargerMode Mode)

TChargerMode :Mode- mode in which the charger will operate, possible values are: cmSHUTDOWN, cmCHARGE, cmSUSPEND, cmBOOST_OTG

Return value:LDD_TError - error code, ERR_OK if successful

EnDisShutdown -Enables/disables shutdown pin which means that the device is put in shutdown mode. In shutdown mode I2C interface is turned off in order to minimize power consumption. However, this applies only in case of invalid input power source. This pin is not effective as long as a valid input power source is present.

ANSIC prototype: void EnDisShutdown(TEnDisState State)

TEnDisState :State- possible values: edsENABLED/edsDISABLED

SetAICLCurrentLimit -This method sets input current limit by writing to VBUSCTRL register. This value limits the fast-charge current when the device is in fast-charge mode. It also sets the limit for adaptive-input current limit (AICL) when the device in Start-up mode automatically starts incrementing the input current limit to either the default or pre-programmed value until either the input current limit is detected or the VBUS voltage detects the AICL threshold, to keep input supply voltage as a valid power source to provide the load for the application.

ANSIC prototype: LDD_TError SetAICLCurrentLimit(uint16_t CurrentVal)

uint16_t :CurrentVal- value of maximum input current, admissible values are from 100 mA to 2050 mA in 50 mA steps (default value is 500 mA), please insert this value in mA (for example 2050 so it is integer type) instead of amperes.

Return value:LDD_TError - error code, ERR_OK if successful

SetAICLVoltageThreshold -This method sets AICL threshold voltage on VBUS. To keep the device functional with a current and voltage limited VBUS source, the device in Start-up mode automatically starts incrementing the input current limit to either the default or pre-programmed value until either the input current limit is detected or the VBUS voltage detects the AICL threshold, to keep input supply voltage as a valid power source to provide the load for the application. The device allows the maximum current the input supply can possibly provide without severely collapsing.

ANSIC prototype: LDD_TError SetAICLVoltageThreshold(TAICLThreshold AICLThreshold)

TAICLThreshold :AICLThreshold- this is an enumerated type, possible values are: aicl4_3V, aicl4_4V, aicl4_5V, aicl4_6V, aicl4_7V, aicl4_8V, aicl4_9V which correspond to voltage from 4.3 V to 4.9 V in 0.1 step.

Return value:LDD_TError - error code, ERR_OK if successful

EnDisAICL -Enables/disables adaptive-input current limit (AICL). AICL is mostly used at the beginning of charging process in cases when current dissipation is higher than the current that the power source can provide. This feature prevents the power source from collapse.

ANSIC prototype: LDD_TError EnDisAICL(TEnDisState State)

TEnDisState :State- possible values: edsENABLED/edsDISABLED

Return value:LDD_TError - error code, ERR_OK if successful

SetPreChargeCurrent -Pre-charge current is current which charges the battery in pre-charge mode. The battery charger enters pre-charge mode when battery voltage is higher than 2.5 V. If the battery voltage does not exceed the VVSYS_MIN threshold before the pre-charge timer expires, charging is suspended and a fault signal is asserted via the INTB pin.

ANSIC prototype: LDD_TError SetPreChargeCurrent(TPrechargeCurrent CurrentVal)

TPrechargeCurrent :CurrentVal- this is an enumerated type, admissible values are: pc150MA, pc250MA, pc350MA, pc450MA which correspond to current from 150 mA to 450 mA in 100 mA steps.

Return value:LDD_TError - error code, ERR_OK if successful

SetTopOffCurrent -Top-off-charge current is current which charges the battery in top-off mode. After top-off timer expires the top-off event is reported to the processor via the INTB pin which means that the battery is fully charged. As soon as the processor reads the interrupt registers, the processor is able to turn off the charger.

ANSIC prototype: LDD_TError SetTopOffCurrent(TTopoffCurrent CurrentVal)

TTopoffCurrent :CurrentVal- this is an enumerated type, admissible values are: tc100MA, tc150MA, tc200MA, tc250MA, tc300MA, tc350MA, tc400MA, tc450MA, tc500MA, tc550MA, tc600MA, tc650MA which corresponds to top-off current from interval 100 mA - 650 mA in 50 mA steps. 100 mA is the default.

Return value:LDD_TError - error code, ERR_OK if successful

SetFastChargeCurrent -Fast-charge current is current which charges the battery in fast-charge mode. The Fast-charge mode is entered when the battery voltage exceeds the VSYS_MIN threshold of a typical 3.6 V. If the battery voltage does not reach the VBAT_REG threshold before the timer expires, charging is suspended and a fault signal is asserted via the INTB pin.

ANSIC prototype: LDD_TError SetFastChargeCurrent(uint16_t CurrentVal)

uint16_t :CurrentVal- admissible range is from 100 mA to 2000 mA in 50 mA steps and with 500 mA default, please insert this in mA so it is integer type instead of amperes.

Return value:LDD_TError - error code, ERR_OK if successful

SetDischargeCurrent -This method sets discharge current limit in discharge mode.

ANSIC prototype: LDD_TError SetDischargeCurrent(TDischargeCurrent CurrentVal)

TDischargeCurrent :CurrentVal- this is an enumerated type with values: dc0_0A, dc2_0A, dc2_5A, dc3_0A, dc3_5A, dc4_0A, dc4_5A, dc5_0A. These values correspond to current limit from interval 2.0 A - 5.0 A in 0.5 A steps.

Return value:LDD_TError - error code, ERR_OK if successful

EnDisAutostop -This method enables or disables autostop feature. If autostop is enabled after top-off timer is expired, the charger turns off and goes into DONE state. If it is disabled charger is on continuously and stays in CV mode after top-off timer is expired.

ANSIC prototype: LDD_TError EnDisAutostop(TEnDisState State)

TEnDisState :State- possible values: edsENABLED/edsDISABLED

Return value:LDD_TError - error code, ERR_OK if successful

SetWeakBatteryThreshold -This method sets the voltage of weak battery threshold. The threshold ranges from 3.0 V to 3.75 V in 50 mV steps. A weak battery detection function allows the processor to acknowledge the low-battery condition by asserting INTB event.

ANSIC prototype: LDD_TError SetWeakBatteryThreshold(TWeakBattery WeakBatTh)

TWeakBattery :WeakBatTh- value of weak battery threshold. It is an enumerated type, admissible values are: wb3_00V, wb3_05V, wb3_10V, wb3_15V, wb3_20V, wb3_25V, wb3_30V, wb3_35V, wb3_40V, wb3_45V, wb3_50V, wb3_55V, wb3_60V, wb3_65V, wb3_70V, wb3_75V which correspond to voltage 3.0 V - 3.75 V in 50 mV steps.

Return value:LDD_TError - error code, ERR_OK if successful

SetBatteryRegulationThreshold -Based on this threshold the charger transits from fast-charge mode (constant current mode) to full-charge mode (constant voltage mode). In full-charge mode the fast charge current is reduced to a programmable top-off current. Up to this threshold the VSYS output tracks the battery voltage in Trickle and Pre-charge mode.

ANSIC prototype: LDD_TError SetBatteryRegulationThreshold(TBatteryRegulation VoltageTh)

TBatteryRegulation :VoltageTh- this is an enumerated type, admissible values are: br4_100V, br4_125V, br4_150V, br4_175V, br4_200V, br4_225V, br4_250V, br4_275V, br4_300V, br4_325V, br4_350V, br4_375V, br4_400V, br4_425V, br4_450V, br4_475V which correspond to 4.1 V - 4.475 V in 25 mV steps.

Return value:LDD_TError - error code, ERR_OK if successful

SetBoostOTGVoltage -This method sets OTG voltage in Boost (OTG) mode. In boost mode the device provides a regulated output voltage to VBUS from the battery.

ANSIC prototype: LDD_TError SetBoostOTGVoltage(TOTGBoost VoltageVal)

TOTGBoost :VoltageVal- this is an enumerated type whose items are: otg5_0V, otg5_1V, otg5_2V and correspond to 5.0V, 5.1V and 5.2V of Boost (OTG) voltage.

Return value:LDD_TError - error code, ERR_OK if successful

SetFastChargeTimer -Fast charge timer watches the device during fast charging mode. If the battery voltage does not reach its required value within this "fast-charge" time frame an interrupt is asserted. Possible values are 3.5, 4.5, 5.5 hours or the timer can be disabled.

ANSIC prototype: LDD_TError SetFastChargeTimer(TFastchargeTimeout TimerVal)

TFastchargeTimeout :TimerVal- this is an enumerated type whose items are: ft0_0H, ft3_5H, ft4_5H, ft5_5H and correspond to 3.5, 4.5, 5.5 hrs or the timer can be disabled.

Return value:LDD_TError - error code, ERR_OK if successful

SetTopOffTimer -Top-off timer watches the device during top-off mode. If the battery voltage does not reach its required value within this "top-off" time frame an interrupt is asserted. Possible values are: 10, 20, 30 or 45 minutes.

ANSIC prototype: LDD_TError SetTopOffTimer(TTopoffTimeout TimerVal)

TTopoffTimeout :TimerVal- this argument is an enumerated type, its items are: tt10MIN, tt20MIN, tt30MIN, tt45MIN and correspond to 10, 20, 30 and 40 minutes.

Return value:LDD_TError - error code, ERR_OK if successful

GetStatus -This method returns the content of status register.

ANSIC prototype: LDD_TError GetStatus(uint8_t *DataPtr)

uint8_t : Pointer to DataPtr- pointer to variable which stores the value of status register

Return value:LDD_TError - error code, ERR_OK if successful

2.4 Properties

Component Name - Name of the component.

General Settings - General settings of the charger (modes, pins).

Charger Enabled - Charging enabled after initialization.

There are 2 options:

yes

no

Suspend Mode - Charging suspended after initialization.

There are 2 options:

yes

no

Comparator Enabled - Internal comparators detecting various interrupt events could be disabled to save the idle current. The comparators are enabled by default. If disabled, Weak Battery, VSYSOK, VSYS Low, Battery OVP and Discharge Current Limit events are not detected.

There are 2 options:

yes

no

Shutdown Enabled - If shutdown mode is enabled the charger is fully functional except I2C interface which is turned off in order to minimize power consumption. However Shutdown pin is not effective as long as a valid input power source is present.

There are 2 options:

yes

no

OTG Enabled - OTG Boost mode after initialization. In OTG Boost mode the device provides a regulated output voltage to VBUS from the battery with load current up to 900 mA to support USB OTG devices.

There are 2 options:

<u>**yes**</u>
<u>**no**</u>

Control Pins - Control pins used to enable or suspend charging and put the charger in shutdown mode.

CHGEN Link - Linked BitIO_LDD component.

CHGEN Enable Pin - CHGEN charger enable pin. This pin is used to enable or suspend charging.

SHDN Link - Linked BitIO_LDD component.

SHDN Shutdown Pin - SHDN shutdown pin. This pin is used to put the charger in shutdown mode to reduce the idle current as low as possible. I2C is disabled in shutdown mode.

I2C Communication - I2C communication settings (link to shared I2C component).

I2C Link - Linked I2C_LDD component.

VBUS Control - VBUS related settings.

AICL on VBUS - The Adaptive Input Current Limit (AICL) function prevents the current limited input supply voltage from sagging below a certain preset AICL threshold voltage. The input current increases as the battery voltage increases. Eventually, the input current may exceed the VBUS input current limit. If this happens, the AICL function takes over and lowers the charge current below the programmed value to keep VBUS around AICL threshold voltage.

AICL Threshold [V] - Adaptive Input Current Limit (AICL) voltage threshold. If VBUS voltage drops below the threshold, input current is reduced. Possible values are 4.3 - 4.9 V, default value is 4.5 V.

There are 7 options:

<u>**4.3**</u>
<u>**4.4**</u>
<u>**4.5**</u>
<u>**4.6**</u>
<u>**4.7**</u>
<u>**4.8**</u>
<u>**4.9**</u>

Input Current Limit [mA] - Input current can be limited to ensure USB compliance and minimize charging time. The value is programmable in range 100 - 2050 mA with 50 mA step.

Charger Control - Programmable charger voltage, current or time limits.

Auto Stop - If enabled, charger is disabled automatically after top-off timer expires and stays in Done mode. If disabled, the charger stays enabled in Constant Voltage mode after the top-off timer expires.

There are 2 options:

<u>**Enabled**</u>: Feature enabled.
<u>**Disabled**</u>: Feature disabled.

OTG Boost Voltage [V] - Programmable OTG Boost voltage in range 5.0 - 5.2 V. Default value is 5.0 V.

There are 3 options:

<u>**5.0**</u>
<u>**5.1**</u>
<u>**5.2**</u>

Battery Regulation [V] - Battery regulation voltage threshold in range 4.1 - 4.475 V in 25 mV steps. Default value is 4.2 V.

There are 16 options:

<u>**4.100**</u>

- **4.125**
- **4.150**
- **4.175**
- **4.200**
- **4.225**
- **4.250**
- **4.275**
- **4.300**
- **4.325**
- **4.350**
- **4.375**
- **4.400**
- **4.425**
- **4.450**
- **4.475**

Weak Battery Threshold [V] - A weak battery detection function allows the processor to acknowledge the low-battery condition. The range is 3.0 - 3.75 V in 50 mV step. Default value is 3.6 V.

There are 16 options:

- **3.00**
- **3.05**
- **3.10**
- **3.15**
- **3.20**
- **3.25**
- **3.30**
- **3.35**
- **3.40**
- **3.45**
- **3.50**
- **3.55**
- **3.60**
- **3.65**
- **3.70**
- **3.75**

Precharge Current [mA] - Pre-charge mode allows a deeply discharged battery to charge safely. The pre-charge current is programmable from 150 mA to 450 mA in 100 mA steps. Default value is 450 mA.

There are 4 options:

- **150**
- **250**
- **350**
- **450**

Fast Charge Current [mA] - The Fast-charge mode is entered when the battery voltage exceeds the VSYS_MIN threshold of a typical 3.6 V. During this mode, the battery is charged with a programmable fast-charge current. The fast-charge current is programmable from 100 mA to 2000 mA in 50 mA steps. Default value is 500 mA. Fast-charge current is always limited by the input current limit setting.

Topoff Current [mA] - As soon as the battery voltage reaches the battery regulation voltage threshold, the fast-charge current is reduced to a programmable top-off current. The range is 100 - 650 mA in 50 mA steps. Default value is 100 mA.

There are 12 options:

<u>**100**</u>
<u>**150**</u>
<u>**200**</u>
<u>**250**</u>
<u>**300**</u>
<u>**350**</u>
<u>**400**</u>
<u>**450**</u>
<u>**500**</u>
<u>**550**</u>
<u>**600**</u>
<u>**650**</u>

Discharge Current Limit [**A**] - Discharge current limit in discharge mode. The range is 2.0 - 5.0 A in 0.5 A steps. Default value is 4.0 A.

There are 8 options:

<u>**0.0**</u>
<u>**2.0**</u>
<u>**2.5**</u>
<u>**3.0**</u>
<u>**3.5**</u>
<u>**4.0**</u>
<u>**4.5**</u>
<u>**5.0**</u>

Ifast Timeout [**h**] - During fast-charge mode, the safety timer called fast charge timer counts. If the battery voltage does not reach the battery regulation voltage threshold before the timer expires, charging is suspended. The range is 3.5 - 5.5 hours in 1 hours steps or the timer can be disabled (by default).

There are 4 options:

<u>**0.0**</u>
<u>**3.5**</u>
<u>**4.5**</u>
<u>**5.5**</u>

Topoff Timeout [**min**] - If the charge current down to a pre-programmed top-off current threshold is sensed, the safety timer called top-off timer automatically counts. When the timer expires, charger is put in Constant Voltage mode or the charging is suspended if Autostop is enabled. Possible values are 10, 20, 30 and 45 min, default value is 45 min.

There are 4 options:

<u>**10**</u>
<u>**20**</u>
<u>**30**</u>
<u>**45**</u>

Interrupts - Interrupts reported on the INTB pin after any status change (can be enabled/disabled individually).

INT Link - Linked ExtInt_LDD component.

INT Interrupt Pin - Active-low when status change on interrupt registers occurs.

Thermal Regulation - Thermal regulation threshold detected. (THEMREG)

There are 2 options:

<u>**Enabled**</u>: Interrupt enabled.
<u>**Disabled**</u>: Interrupt disabled.

Thermal Shutdown - Thermal shutdown detected. (THEMSHDN)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

Battery OVP - Battery OVP detected. (BATOVP)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

VBUS Limit - VBUS input current limit detected. (VBUSLIMIT)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

AICL Threshold - AICL threshold detected. (AICL)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

VBUS OK - Valid VBUS detected. (VBUSINOK)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

VBUS UVLO - VBUS falling UVLO detected. (VBUSUVLO)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

VBUS OVP - VBUS OVP event detected. (VBUSOVP)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

Charger Topoff - Top-off threshold is detected. (TOPOFF)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

Charger Done - Top-off charge timer expired. (DONE)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

Charger Restart - Charger restart detected. (CHGRSTF)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

Precharge Timeout - Pre-charge timer expired. (PRETMROFF)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

OTG Boost Fail - Boost fail detected due to overload. (OTGFAIL)

There are 2 options:

Enabled: Interrupt enabled.
Disabled: Interrupt disabled.

Weak Battery - Weak battery threshold detected. (WEAKBAT)

There are 2 options:

- **Enabled**: Interrupt enabled.
- **Disabled**: Interrupt disabled.

Battery Detection - No battery detected. (NOBAT)

There are 2 options:

- **Enabled**: Interrupt enabled.
- **Disabled**: Interrupt disabled.

Fast Charge Timeout - Fast charger timer expired. (FASTTMROFF)

There are 2 options:

- **Enabled**: Interrupt enabled.
- **Disabled**: Interrupt disabled.

Discharge Current Limit - Current limit threshold detected in discharge mode. (DISLIMIT)

There are 2 options:

- **Enabled**: Interrupt enabled.
- **Disabled**: Interrupt disabled.

VSYS Overload - VSYS overload condition debounce is detected in a valid VBUS attached. (VSYSOLP)

There are 2 options:

- **Enabled**: Interrupt enabled.
- **Disabled**: Interrupt disabled.

VSYS Low Threshold - VSYS falling 3.4 V detected in a valid VBUS attached. (VSYSNG)

There are 2 options:

- **Enabled**: Interrupt enabled.
- **Disabled**: Interrupt disabled.

VSYS OK - VSYS rising 3.6 V detected in a valid VBUS attached. (VSYSOK)

There are 2 options:

- **Enabled**: Interrupt enabled.
- **Disabled**: Interrupt disabled.

Auto Initialization - When auto initialization is enabled, Init method will be called automatically within PE initialization function - PE_low_level_init().

There are 2 options:

- **yes**
- **no**

3 Typical Usage

Examples of typical settings and usage of BC_MC32BC3770 component

- Device initialization.
- Reading/writing registers.
- Device interrupt processing.

Device initialization.

This example shows how to handle device initialization when auto-initialization feature is disabled.

Required component setup and dependencies:

- *Auto initialization*: no
- Methods: Init

Content of main.c:

Listing 1: Source code

```
void main(void)
{
        BC1_TDeviceDataPtr BC1_DeviceDataPtr;

        ...

        BC1_DeviceDataPtr = BC1_Init(&UserData); /* It is possible to pass
    pointer to your own data, which is then stored in device data structure
    as TUserDataPtr. */
        if (BC1_DeviceDataPtr != NULL) {
                /* Initialization was successful. */
        } else {
                /* Initialization was not successful. */
        }

        ...

        TUserData *MyData = (TUserData *)(BC1_DeviceDataPtr->UserDataPtr);
    /* You can access your data later. Explicit retype is needed because
    UserDataPtr is just typedef of (void *). */

}
```

Reading/writing registers.

This example shows how to read/write single or multiple registers of the device.

It is important to note, that all written values are stored in structure representing device global state, which is accessible through **BC1_DeviceDataPtr** pointer.

Required component setup and dependencies:

Shutdown Enabled: no

Methods: ReadRegister WriteRegister ReadBurstData WriteBurstData

Content of main.c:

Listing 2: Source code

```
void main(void)
{
        LDD_TError Error;
        uint8_t Single[1] = {0};     /* Variable for single register read/
    write. */
        uint8_t Burst[3] = {0,0,0}; /* Variable for burst register read/
    write. */

        ...

        /* Reading one register - for example INT1 register. */
        Error = BC1_ReadRegister(BC_INT1, Single); /* Single as a pointer.
    */
        if (Error != ERR_OK ) {
                /* something went wrong */
        }

        ...

        /* Writing one register - for example INTMSK1 register. */
        Error = BC1_WriteRegister(BC_INTMSK1, Single[0]); /* Single as a
    value. */
```

```
        if (Error != ERR_OK ) {
                /* something went wrong */
        }

        ...

        /* Reading multiple registers - for example INT1..3 registers. */
        /* Write only address of first register and total amount of
registers to read. */
        Error = BC1_ReadBurstData(BC_INT1, Burst, 3); /* Burst as a pointer.
 */
        if (Error != ERR_OK ) {
                /* something went wrong */
        }

        ...

        /* Writing multiple registers - for example INTMSK1..3 registers. */
        /* Write only address of first register and total amount of
registers to write. */
        Error = BC1_WriteBurstData(BC_INTMSK1, Burst, 3); /* Burst as a
pointer. */
        if (Error != ERR_OK ) {
                /* something went wrong */
        }

        ...

}
```

Device interrupt processing.

This example shows how to setup and handle device interrupts.

There are more ways how to setup device interrupts (which internally detected event will be reflected via interrupt pin):

By configuring these events in component properties under Interrupts.

By using SetInterrupt method.

By using WriteBurstData method to write to INTMSK1..3 registers.

Only the second possibility will be shown in this example (using WriteBurstData method is described under Reading/writing registers.)

There are also more ways how to handle device interrupts:

By utilization of OnInterrupt event together with ReadBurstData method.

By periodical reading of INT1..3 registers using ReadBurstData method.

In all cases user is responsible for taking corrective actions and clearing interrupt flags (ClearInterrupt).

The interrupt processing is started by reading (ReadBurstData) actual status data of internal device events (in registers INT1..3).

Note that if there are multiple pending interrupts at the same time, the OnInterrupt event is invoked only once and it is up to user to correctly handle all of them. Also the Interrupt pin value will toggle only **after all of the faults are cleared**.

You can use **BC1_SetInterrupt(intSET_ALL, edsENABLED);** to enable all interrupts and **BC1_ClearInterrupt(intCLEAR_ALL);** to clear all flags at once.

Required component setup and dependencies:

Interrupts: Enabled

INT Interrupt Pin properly set

Methods: ReadBurstData SetInterrupt ClearInterrupt

Events: OnInterrupt

Content of main.c:

Listing 3: Source code

```
bool MyInterruptHandler() {
        uint8_t InterruptReg[3]; /* Variable for storing INT1..3 registers
    values. */

        /* Read actual device internal event flags. */
        if (BC1_ReadBurstData(BC_INT1, InterruptReg, 3) != ERR_OK) {
                return FALSE;
        }

        /* Take corrective actions (if needed) and clear the faults - only 3
     faults set and handled in this example. */

        /* VBUSLIMIT - in register INT1. */
        /* VBUS input current limit detected. */
        if (InterruptReg[0] & BC_INT1_VBUSLIMIT_MASK) {

                /* Write your own code. */

                /* Clear interrupt flag. */
                Error = BC1_ClearInterrupt(intVBUSLIMIT);
                if (Error != ERR_OK ) {
                        return FALSE;
                }
        }

        /* WEAKBAT - in register INT2. */
        /* Weak battery threshold detected. */
        if (InterruptReg[1] & BC_INT2_WEAKBAT_MASK) {

                /* Write your own code. */

                /* Clear interrupt flag. */
                Error = BC1_ClearInterrupt(intWEAKBAT);
                if (Error != ERR_OK ) {
                        return FALSE;
                }
        }

        /* DISLIMIT - in register INT3. */
        /* Current limit threshold detected in discharge mode. */
        if (InterruptReg[2] & BC_INT3_DISLIMIT_MASK) {

                /* Write your own code. */

                /* Clear interrupt flag. */
                Error = BC1_ClearInterrupt(intDISLIMIT);
                if (Error != ERR_OK ) {
                        return FALSE;
                }
        }

        return TRUE;
}

bool InterruptFlag = FALSE;
```

```
void main(void)
{
        LDD_TError Error;
        bool Result; /* Result of MyInterruptHandler - TRUE if successfull.
    */

        ...

        /* Set VBUSLIMIT, WEAKBAT and DISLIMIT events to be reflected via
    interrupt pin. */
        Error = BC1_SetInterrupt(intVBUSLIMIT, edsENABLED);
        Error |= BC1_SetInterrupt(intWEAKBAT, edsENABLED);
        Error |= BC1_SetInterrupt(intDISLIMIT, edsENABLED);
        if (Error != ERR_OK ) {
                /* something went wrong */
        }

        ...

        for(;;) { /* MAIN LOOP */

                ...

                /* Checking flag set in OnInterrupt event. */
                if (InterruptFlag) {
                        Result = MyInterruptHandler();
                        if (!Result)
                                /* something went wrong */
                        else
                                InterruptFlag = FALSE;
                }

        ...

        }
}
```

<u>**Content of Events.c:**</u>

Listing 4: Source code

```
extern uint8_t InterruptFlag;

/* It is recommended that interrupt handlers should be as simple as possible
     because of computational overhead. */
void BC1_OnInterrupt() {
        InterruptFlag = TRUE;
}
```

4 User Types

***ComponentName*_TDeviceDataPtr** = *device data pointer*

TUserDataPtr = *user data pointer*

***ComponentName*_TRegisterAddress** = enum { regINT1, regINT2, regINT3, regINTMSK1, regINTMSK2, regINTMSK3, regSTATUS, regCTRL, regVBUSCTRL, regCHGCTRL1, regCHGCTRL2, regCHGCTRL3, regCHGCTRL4, regCHGCTRL5} *address of register*

***ComponentName*_TEnDisState** = enum { edsDISABLED, edsENABLED} *enabled/disabled*

***ComponentName*_TReadWrite** = enum { rwREAD, rwWRITE} *read/write*

***ComponentName*_TChargerMode** = enum { cmSHUTDOWN, cmCHARGE, cmSUSPEND, cmBOOST_OTG} *enumeration of charger operational modes*

***ComponentName*_TInterrupt** = enum { intTHEMREG, intTHEMSHDN, intBATOVP, intVBUSLIMIT, intAICL, intVBUSINOK, intVBUSUVLO, intVBUSOVP, intTOPOFF, intDONE, intCHGRSTF, intPRETMROFF, intOTGFAIL, intWEAKBAT, intNOBAT, intFASTTMROFF, intDISLIMIT, intVSYSOLP, intVSYSNG, intVSYSOK, intSET_ALL, intCLEAR_ALL} *enumeration of all charger interrupts*

***ComponentName*_TAICLThreshold** = enum { aicl4_3V, aicl4_4V, aicl4_5V, aicl4_6V, aicl4_7V, aicl4_8V, aicl4_9V} *enumeration of AICL thresholds*

***ComponentName*_TOTGBoost** = enum { otg5_0V, otg5_1V, otg5_2V} *enumeration of OTG Boost voltages*

***ComponentName*_TBatteryRegulation** = enum { br4_100V, br4_125V, br4_150V, br4_175V, br4_200V, br4_225V, br4_250V, br4_275V, br4_300V, br4_325V, br4_350V, br4_375V, br4_400V, br4_425V, br4_450V, br4_475V} *enumeration of battery regulation voltages*

***ComponentName*_TWeakBattery** = enum { wb3_00V, wb3_05V, wb3_10V, wb3_15V, wb3_20V, wb3_25V, wb3_30V, wb3_35V, wb3_40V, wb3_45V, wb3_50V, wb3_55V, wb3_60V, wb3_65V, wb3_70V, wb3_75V} *enumeration of weak battery thresholds*

***ComponentName*_TPrechargeCurrent** = enum { pc150MA, pc250MA, pc350MA, pc450MA} *enumeration of pre-charge currents*

***ComponentName*_TTopoffCurrent** = enum { tc100MA, tc150MA, tc200MA, tc250MA, tc300MA, tc350MA, tc400MA, tc450MA, tc500MA, tc550MA, tc600MA, tc650MA} *enumeration of top-off currents*

***ComponentName*_TDischargeCurrent** = enum { dc0_0A, dc2_0A, dc2_5A, dc3_0A, dc3_5A, dc4_0A, dc4_5A, dc5_0A} *enumeration of discharge currents*

***ComponentName*_TFastchargeTimeout** = enum { ft0_0H, ft3_5H, ft4_5H, ft5_5H} *enumeration of fast-charge timeouts*

***ComponentName*_TTopoffTimeout** = enum { tt10MIN, tt20MIN, tt30MIN, tt45MIN} *enumeration of top-off timeouts*

How to Reach Us:

Home Page:
NXP.com

Web Support:
http://www.nxp.com/support

Document Number: PEXBC3770PUG
Rev. 1.0
2/2016