Document Number: PEXFRDMBC3770EVBPUG
Rev. 1.0, 2/2016

FRDM-BC3770-EVB 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.

FRDM-BC3770-EVB is a Freedom development platform for quick application prototyping. This module features MC32BC3770CS Battery charger, Current Sense Amplifiers (CSAs) which permit the real-time measurement of current and voltage on the VBUS input supply, the VSYS output supply and the battery. The Freedom board also features programmable Electronic load, programmable in range from 0 to 1 A, which is used to demonstrate system performance with an active load applied to either the VSYS output or the battery. When attached to the battery, the Electronic load can be used to discharge the battery in a controlled manner.

2 Embedded Component Description

2.1 Component API

FRDM_BC3770 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 FRDM_BC3770 Component API

Table 1

Method	Description
FreedomBoard_Init	FreedomBoard_Init Initializes devices on the board: current sense amplifiers and electronic load and assigns user defined data to structure describing the board.
SelectDevice	This method selects one of devices on the board: electronic load or some of current sense amplifiers to measure current or voltage.
GetSelectedDevice	This method returns last selected device.
ELO_Init	ELO_Init This method initializes electronic load (ELOAD). Device MPC4728 is initialized with default values. Outputs B, C and D are in power down mode. Only output A, which is used for ELOAD control, is in normal mode.
ELO_SetCurrent	ELO_SetCurrent This method sets SetPoint of ELOAD (the amount of current ELOAD will sink). This method is same as internal ELO_SetPointEload except function parameter which is a real number [mA]. This method sends command 'new value' to output register of channel A. Resolution of output voltage is 12 bit. Command 'send new value' sets default settings for other registers.
CSA_Init	CSA_Init Initializes the device and sets default configuration (see documentation of INA230) of VBUS, VSYS, BATTERY amplifier and settings of calibration register.
CSA_ReadRegister	CSA_ReadRegister This method reads a register of selected device. Read value is a 16 bit number from selected register.
CSA_WriteRegister	CSA_WriteRegister This method writes a register of selected device. Outgoing value is a 16 bit number.

CSA_GetCurrent	CSA_GetCurrent This method reads content of Current Register of selected device and converts it to current in milliamperes [mA]. To change selected device use BCF_CSA_SelectDevice method.
CSA_GetVoltage	CSA_GetVoltage This method reads Bus Voltage or Shunt Voltage register of selected device according to the first parameter. Value of register is converted to voltage in millivolts [mV].
NTC_GetTemperature	NTC_GetTemperature This method measures voltage on NTC thermistor and calculates temperature according to SteinhartHart equation. The temperature is thermodynamic T[K]. Precision of resulting temperature depends on NTC constants of equation. Add correct constants A, B, C, D from your NTC thermistor datasheet. For mor information see NTC datasheet.
SetLED	This method turns on/off green or red LED.

2.2 Events

There are no Events in this component

2.3 Methods

FreedomBoard_Init - Initializes devices on the board: current sense amplifiers and electronic load and assigns user defined data to structure describing the board.

ANSIC prototype: TDeviceDataPtr FreedomBoard_Init(TUserDataPtr *UserDataPtr)

TUserDataPtr : Pointer to UserDataPtr- User data pointer

Return value:TDeviceDataPtr - If initialization was successful method returns a pointer to a structure describing the freedom board. Otherwise NULL pointer is returned. The structure contains an item TUserDataPtr which enables to add any custom data to the structure.

SelectDevice - This method selects one of devices on the board: electronic load or some of current sense amplifiers to measure current or voltage.

ANSIC prototype: LDD_TError SelectDevice(TDeviceName Device)

TDeviceName :Device- Device type. Possible values are: dnVBUS_CSA - I2C ADDR: 0x80 dnVSYS_CSA - I2C ADDR: 0x81 dnVBAT_CSA - I2C ADDR: 0x82 dnELOAD - I2C ADDR: 0xC0

Return value:LDD_TError - ERR_OK : No error, ERR_PARAM_RANGE : Parameter is out of defined range

GetSelectedDevice - This method returns last selected device.

ANSIC prototype: TDeviceName GetSelectedDevice(void)

Return value:TDeviceName - Device type. Possible values are: dnVBUS_CSA, dnVSYS_CSA, dnVBAT_CSA, dnELOAD

ELO_Init - This method initializes electronic load (ELOAD). Device MPC4728 is initialized with default values. Outputs B, C and D are in power down mode. Only output A, which is used for ELOAD control, is in normal mode.

ANSIC prototype: LDD_TError ELO_Init(void)

Return value:LDD_TError - ERR_OK : No error, ERR_PARAM_RANGE : Parameter is out of defined range, ERR_PARAM_ADRESS : NULL pointer, ERR_BUSY : I2C bus is not ready

ELO_SetCurrent - This method sets SetPoint of ELOAD (the amount of current ELOAD will sink). This method is same as internal ELO_SetPointEload except function parameter which is a real number [mA]. This method sends command 'new value' to output register of channel A. Resolution of output voltage is 12 bit. Command 'send new value' sets default settings for other registers.

ANSIC prototype: LDD_TError ELO_SetCurrent(uint16_t Current_mA)

uint16_t :Current_mA- ELOAD current value in milliamperes [mA].

Return value:LDD_TError - ERR_OK : No error, ERR_VALUE : Wrong value, ERR_BUSY : I2C bus is not ready

CSA_Init - Initializes the device and sets default configuration (see documentation of INA230) of VBUS, VSYS, BATTERY amplifier and settings of calibration register.

ANSIC prototype: LDD_TError CSA_Init(void)

Return value:LDD_TError - ERR_OK : No error, ERR_PARAM_RANGE : Parameter is out of defined range, ERR_BUSY : I2C bus is not ready

CSA_ReadRegister - This method reads a register of selected device. Read value is a 16 bit number from selected register.

ANSIC prototype: LDD_TError CSA_ReadRegister(uint8_t RegAddress, uint16_t *RegValuePtr)

uint8_t : RegAddress- Address of register which you want to read. Addresses of registers are in BCF_CSA_INA230.h.

uint16_t : Pointer to RegValuePtr- Pointer to memory where read value will be stored.

Return value:LDD_TError - ERR_OK : No error, ERR_PARAM_ADDRESS_TYPE : Invalid address type, ERR_PARAM_ADRESS : NULL pointer, ERR_BUSY : I2C bus is not ready, ERR_BUSOFF : Bus not available

CSA_WriteRegister - This method writes a register of selected device. Outgoing value is a 16 bit number.

ANSIC prototype: LDD_TError CSA_WriteRegister(uint8_t RegAddress, uint16_t RegValue)

uint8_t : RegAddress- Address of register to be written. Addresses of registers are in BCF_CSA_INA230.h.

uint16_t : RegValue- Value to be sent to the register.

Return value:LDD_TError - ERR_OK : No error, ERR_PARAM_ADDRESS_TYPE : Invalid address type, ERR_PARAM_ADRESS : NULL pointer, ERR_BUSY : I2C bus is not ready, ERR_BUSOFF : Bus not available

CSA_GetCurrent - This method reads content of Current Register of selected device and converts it to current in milliamperes [mA]. To change selected device use BCF_CSA_SelectDevice method.

ANSIC prototype: LDD_TError CSA_GetCurrent(float *CurrentPtr_mA)

float : Pointer to CurrentPtr_mA- Result of current measurement in milliamperes [mA].

Return value:LDD_TError - ERR_OK : No error, ERR_PARAM_RANGE : Parameter is out of defined range, ERR_PARAM_ADRESS : NULL pointer, ERR_BUSY : I2C bus is not ready, ERR_BUSOFF : Bus not available

CSA_GetVoltage - This method reads Bus Voltage or Shunt Voltage register of selected device according to the first parameter. Value of register is converted to voltage in millivolts [mV].

ANSIC prototype: LDD_TError CSA_GetVoltage(CSA_TVoltage VoltageType,float *VoltagePtr_mV)

CSA_TVoltage :VoltageType- BCF_CSA_TVoltage enum. There are two options: volCSA_SHUNT - Shunt Voltage volCSA_BUS - BUS voltage

float : Pointer to VoltagePtr_mV- Result of voltage measurement in millivolts [mV] is written.

Return value:LDD_TError - ERR_OK : No error, ERR_PARAM_RANGE : Parameter is out of defined range, ERR_PARAM_ADRESS : NULL pointer, ERR_BUSY : I2C bus is not ready, ERR_BUSOFF : Bus not available

NTC_GetTemperature - This method measures voltage on NTC thermistor and calculates temperature according to Steinhart-Hart equation. The temperature is thermodynamic T[K]. Precision of resulting temperature depends on NTC constants of equation. Add correct constants A, B, C, D from your NTC thermistor datasheet. For mor information see NTC datasheet.

ANSIC prototype: LDD_TError NTC_GetTemperature(float* TemperaturePtr_K)

Pointer to float :TemperaturePtr_K- Pointer to variable for saving current temperature T[K].

Return value:LDD_TError - ERR_OK : No error, ERR_PARAM_VALUE : Wrong value, ERR_PARAM_ADRESS : NULL pointer of parameter

SetLED - This method turns on/off green or red LED.

ANSIC prototype: LDD_TError SetLED(TLEDName LED,TLEDState State)

TLEDName :LED- Name of LED to turn on/off. Possible values are: lnGreenLED, lnRedLED.

TLEDState : State- New state of LED. Possible values are: lsLEDOn, lsLEDOff.

Return value:LDD_TError - ERR_OK : No error, ERR_PARAM_RANGE : Parameter is out of defined range

2.4 Properties

Component Name - Name of the component.

BC_MC32BC3770 - Link to BC_MC32BC3770 Battery Charger component.

ChannelAllocator - The component is used to manage allocation of ADC_LDD component channels.

I2C Communication - Settings for I2C communication.

I2C Link - Link to I2C_LDD component.

ELOAD - Settings for electronic load.

LDAC Link - Link to BitIO_LDD component.

LDAC Synchronization Pin - Link to LDAC pin which is used as a synchronization input. When set to Low the contents of the DAC input registers is transfered to the output registers.

RDY/BSY Link - Link to Ready/Busy pin.

Ready/Busy Status Pin - This pin is a status indicator of EEPROM programming activity. If it is "High" the EEPROM has no programming activity (is ready) and if it is "Low" the EEPROM is in programming mode (busy).

ELoad Sink Current - This is the current that the electronic load sinks from either battery or VSYS. Admissible range is from 0 to 1000 mA.

Voltage and Current Measurement - Voltage and current measurement through three current sense amplifiers on board.

NTC Thermistor - External thermistor for temperature measurement settings.

ADC Link - Linked ADC_LDD component.

ADC Device - name of component driving ADC peripheral

ADC Conversion Time - duration of ADC conversion

NTC_TEMP Pin - ADC pin which is connected to NTC thermistor

NTC Reference [Ohm] - resistance of thermistor at 25 C

NTC A [10^{-3} K^{0}] - first coefficient of Steinhart-Hart equation. Note: The format of coefficients varies from datasheet to datasheet, so it is important to pay attention to position of decimal point.
For example, if datasheet says $0.5x10^{-4}$ (or 0.5E-04) convert this to $0.05x10^{-3}$ and enter only the significand part of the number (i.e. 0.05)

NTC B [10^{-4} K^{-1}] - second coefficient of Steinhart-Hart equation. Note: The format of coefficients varies from datasheet to datasheet, so it is important to pay attention to position of decimal point.
For example, if datasheet says $0.5x10^{-5}$ (or 0.5E-05) convert this to $0.05x10^{-4}$ and enter only the significand part of the number (i.e. 0.05)

NTC C [10^{-6} K^{-2}] - third coefficient of Steinhart-Hart equation. Note: The format of coefficients varies from datasheet to datasheet, so it is important to pay attention to position of decimal point.
For example, if datasheet says $0.5x10^{-7}$ (or 0.5E-07) convert this to $0.05x10^{-6}$ and enter only the significand part of the number (i.e. 0.05)

NTC D [10^{-7} K^{-3}] - fourth coefficient of Steinhart-Hart eqution. Note: The format of coefficients varies from datasheet to datasheet, so it is important to pay attention to position of decimal point. For example, if datasheet says $0.5x10^{-8}$ (or 0.5E-07) convert this to $0.05x10^{-7}$ and enter only the significand part of the number (i.e. 0.05)

Additional Pins - Pins for green and red LEDs for various indication.

Green LED Pin - green LED indicates charging

Red LED Pin - red LED indicates discharging

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

3 Typical Usage

Examples of typical settings and usage of FRDM_BC3770 component

Device initialization.

Setting ELOAD current and measurement of voltage, current and temperature.

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: FreedomBoard_Init

Content of main.c:

Listing 1: Source code

```
void main(void)
{
        BCF1_TDeviceDataPtr BCF1_DeviceDataPtr;

        ...

        BCF1_DeviceDataPtr = BCF1_FreedomBoard_Init(&UserData); /* It is
    possible to pass pointer to your own data, which is then stored in device
     data structure as TUserDataPtr. */
        if (BCF1_DeviceDataPtr != NULL) {
                /* Initialization was successful. */
        } else {
                /* Initialization was not successful. */
        }

        ...

        TUserData *MyData = (TUserData *)(BCF1_DeviceDataPtr->UserDataPtr);
    /* You can access your data later. Explicit retype is needed because
    UserDataPtr is just typedef of (void *). */

}
```

Setting ELOAD current and measurement of voltage, current and temperature.

This example shows how to set electronic load (ELOAD) current and measure VBUS, VSYS and VBAT voltages and currents and NTC thermistor temperature.

It is important to note, that ELOAD and all current sense amplifiers (CSAs) share the same I2C bus and in order to operate them, you need to call SelectDevice method first. ELO_SetCurrent method already contains this call. All currents (both to set and measured) are in milliamperes, voltages in millivolts and temperature in Kelvins.

Required component setup and dependencies:

ELOAD: Enabled

Voltage and Current Measurement: Enabled

NTC Thermistor: Enabled

All corresponding pins properly set

Methods: SelectDevice ELO_SetCurrent CSA_GetCurrent CSA_GetVoltage NTC_GetTemperature

Content of main.c:

Listing 2: Source code

```
void main(void)
{
        float Current[3], Voltage[3], Temp[1]; /* Measured currents,
    voltages and temperature. */

        ...

        /* Set ELOAD current to 200 mA. */
        /* You don't need to use SelectDevice for ELOAD - it is directly in
    ELO_SetCurrent. */
        BCF1_ELO_SetCurrent(200);

        for (;;) {
                /* Measure VBUS current and voltage. */
                BCF1_SelectDevice(dnVBUS_CSA);
                BCF1_CSA_GetCurrent(&Current[0]);
                BCF1_CSA_GetVoltage(volCSA_BUS, &Voltage[0]);

                /* Measure VSYS current and voltage. */
                BCF1_SelectDevice(dnVSYS_CSA);
                BCF1_CSA_GetCurrent(&Current[1]);
                BCF1_CSA_GetVoltage(volCSA_BUS, &Voltage[1]);

                /* Measure VBAT current and voltage. */
                BCF1_SelectDevice(dnVBAT_CSA);
                BCF1_CSA_GetCurrent(&Current[2]);
                BCF1_CSA_GetVoltage(volCSA_BUS, &Voltage[2]);

                /* Measure NTC temperature, convert from Kelvin to Celsius.
    */
                BCF1_NTC_GetTemperature(Temp);
                Temp[0] = Temp[0] - 273.15;

                ...

        }
}
```

4 User Types

***ComponentName*_CSA_TVoltage** = enum { volCSA_BUS, volCSA_SHUNT } *Which voltage we want to measure. Either BUS voltage or SHUNT voltage.*

***ComponentName*_TDeviceName** = enum { dnELOAD, dnVSYS_CSA, dnVBAT_CSA, dnVBUS_CSA } *Name of a device that we want to talk to.*

***ComponentName*_CSA_TMode** = enum { omCSA_POWERDOWN, omCSA_SHUNT_TRIGGERED, omCSA_BUS_TRIGGERED, omCSA_SHUNT_BUS_TRIGGERED, omCSA_SHUNT_CONTINUOUS, omCSA_BUS_CONTINUOUS, omCSA_SHUNT_BUS_CONTINUOUS } *Operating Mode*

***ComponentName*_CSA_TVscht** = enum { ctCSA_VSHCT_140_US, ctCSA_VSHCT_204_US, ctCSA_VSHCT_332_US, ctCSA_VSHCT_588_US, ctCSA_VSHCT_1_1_MS, ctCSA_VSHCT_2_1_MS, ctCSA_VSHCT_4_1_MS, ctCSA_VSHCT_8_2_MS } *Shunt Voltage Conversion Time*

***ComponentName*_CSA_TVbusct** = enum { ctCSA_VBUSCT_140_US, ctCSA_VBUSCT_204_US, ctCSA_VBUSCT_332_US, ctCSA_VBUSCT_588_US, ctCSA_VBUSCT_1_1_MS, ctCSA_VBUSCT_2_1_MS, ctCSA_VBUSCT_4_1_MS, ctCSA_VBUSCT_8_2_MS } *Bus Voltage Conversion Time*

***ComponentName*_CSA_TAvg** = enum { amCSA_AVG_1 = 0, amCSA_AVG_4, amCSA_AVG_16, amCSA_AVG_64, amCSA_AVG_128, amCSA_AVG_256, amCSA_AVG_512, amCSA_AVG_1024 } *Averaging Mode*

***ComponentName*_TLEDName** = enum { lnGreenLED, lnRedLED } *LED Name*

***ComponentName*_TLEDState** = enum { lsLEDOn, lsLEDOff } *State of LED*

***ComponentName*_TUserDataPtr** = *User data pointer*

How to Reach Us:

Home Page:
NXP.com

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

Document Number: PEXFRDMBC3770EVBPUG
Rev. 1.0
2/2016