SEMTECH

LoRa Basics™ Modem

WIRELESS & SENSING PRODUCTS

Porting Guide

Table of Contents

1 Introduction 4
2 LoRa Basics™ Modem Library 5
3 SMTC BSP 5
3.1 GPIO 6
3.2 GPIO Pin Names 8
3.3 ADC 9
3.4 NVM 10
3.5 RNG 11
3.6 RTC 12
3.7 SPI 14
3.8 Timer 15
3.9 UART 16
3.10 Watchdog 17
3.11 BSP Options 18
3.12 BSP RAL 18
3.13 BSP MCU 19
4 Head and Stack Size 23
5 References 23
6 Revision History 24
7 Glossary 25

List of Figures

Figure 1: LoRa Basics™ Modem Layers 4

1 Introduction

This document provides guidelines for porting the LoRa Basics™ Modem library onto targets that are not covered in the examples provided by Semtech.

The LoRa Basics™ Modem is composed of several layers:

- Application: implemented by the final user
- LoRa Basics™ Modem API: accessible by the user
- LR1MAC
- Radio Planner: manages radio scheduling
- Radio Abstraction Layer (RAL): addresses the command to the right radio driver
- Radio Driver
- SMTC_BSP: (board support package) modified for a new target
- PHY: the target

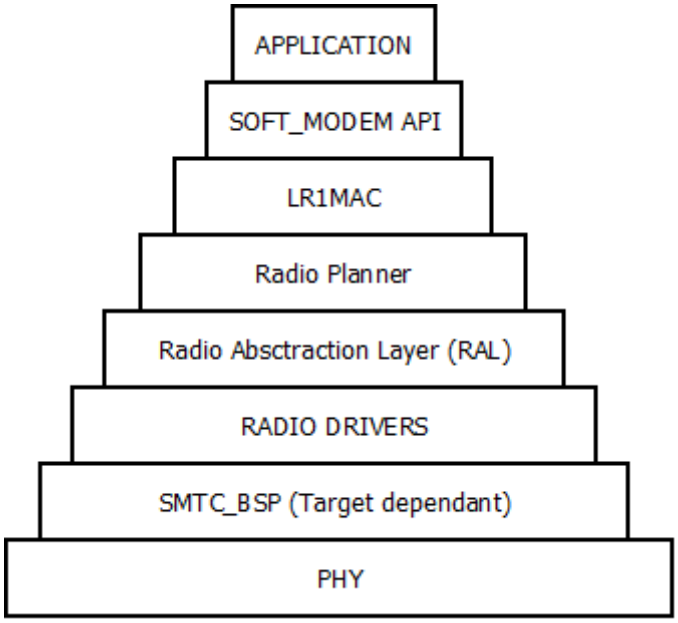

Figure 1: LoRa Basics™ Modem Layers

2 LoRa Basics™ Modem Library

The LoRa Basics™ Modem library contains the following folders:

- lr1mac: the LoRaWAN MAC layer
- radio_planner
- smtc_bsp
- smtc_crypto
- smtc_modem_core
- smtc_ral
- sx1280_driver
- user_app containing:
    - main
    - bsp specific to the Radio Planner / lr1mac / ral / radio driver / mcu

3 SMTC BSP

In order to port onto a target, the following files must be implemented by the user:

- In smtc_bsp folder :
    - smtc_bsp_adc.c
    - smtc_bsp_gpio.c
    - smtc_bsp_gpio_pin_names.h
    - smtc_bsp_nvm.c
    - smtc_bsp_rng.c
    - smtc_bsp_rtc.
    - smtc_bsp_spi.c
    - smtc_bsp_tmr.c
    - smtc_bsp_uart.c
    - smtc_bsp_watchdog.c
- In user_app/bsp_specific folder :
    - smtc_bsp_options.h
    - smtc_bsp_mcu.c
    - ral_hal.c

3.1 GPIO

Any usage of the GPIO functions must respect this API:

```
/*!
 * Initializes given pin as output with given initial value
 *
 * \param [in] pin   MCU pin to be initialized
 * \param [in] value MCU initial pin state
 *
 */
void bsp_gpio_init_out( const bsp_gpio_pin_names_t pin, const uint32_t value );

/*!
 * Initializes given pin as input
 *
 * \param [in]     pin       MCU pin to be initialized
 * \param [in]     pull_mode MCU pin pull mode [BSP_GPIO_PULL_MODE_NONE,
 *                                              BSP_GPIO_PULL_MODE_UP,
 *                                              BSP_GPIO_PULL_MODE_DOWN]
 * \param [in]     irq_mode  MCU IRQ mode [BSP_GPIO_IRQ_MODE_OFF,
 *                                         BSP_GPIO_IRQ_MODE_RISING,
 *                                         BSP_GPIO_IRQ_MODE_FALLING,
 *                                         BSP_GPIO_IRQ_MODE_RISING_FALLING]
 * \param [in/out] irq       Pointer to IRQ data context.
 *                              NULL when BSP_GPIO_IRQ_MODE_OFF
 *                              pin parameter is initialized
 */
void bsp_gpio_init_in( const bsp_gpio_pin_names_t pin, const gpio_pull_mode_t pull
_mode, const gpio_irq_mode_t irq_mode,
                       bsp_gpio_irq_t* irq );

/*!
 * Attaches given callback to the MCU IRQ handler
 *
 * \param [in] irq     Pointer to IRQ data context
 */
void bsp_gpio_irq_attach( const bsp_gpio_irq_t* irq );
```

```c
/*!
 * Detaches callback from the MCU IRQ handler
 *
 * \param [in] irq     Pointer to IRQ data context
 */
void bsp_gpio_irq_deatach( const bsp_gpio_irq_t* irq );

/*!
 * Enables all GPIO MCU interrupts
 */
void bsp_gpio_irq_enable( void );

/*!
 * Disables all GPIO MCU interrupts
 */
void bsp_gpio_irq_disable( void );

/*!
 * Sets MCU pin to given value
 *
 * \param [in] pin   MCU pin to be set
 * \param [in] value MCU pin state to be set
 */
void bsp_gpio_set_value( const bsp_gpio_pin_names_t pin, const uint32_t value );

/*!
 * Toggles MCU pin state value
 *
 * \param [in] pin   MCU pin to be toggled
 */
void bsp_gpio_toggle( const bsp_gpio_pin_names_t pin );

/*!
 * Gets MCU pin state value
 *
 * \param [in] pin   MCU pin to be read
 *
 * \retval value Current MCU pin state
 */
uint32_t bsp_gpio_get_value( const bsp_gpio_pin_names_t pin );

/*
 * Indicates if there are gpio IRQs pending.
 *
 * \retval pendig [true: IRQ pending
 *                 false: No IRQ pending]
 */
bool bsp_gpio_is_pending_irq( void );
```

3.2 GPIO Pin Names

The GPIO pin names can be adapted to the new target if necessary.

```
typedef enum bsp_gpio_pin_names_e
{
    // GPIOA
    PA_0  = 0x00,
    PA_1  = 0x01,
    PA_2  = 0x02,
    PA_3  = 0x03,
    PA_4  = 0x04,
    PA_5  = 0x05,
    PA_6  = 0x06,
    PA_7  = 0x07,
    PA_8  = 0x08,
    PA_9  = 0x09,
    PA_10 = 0x0A,
    PA_11 = 0x0B,
    PA_12 = 0x0C,
    PA_13 = 0x0D,
    PA_14 = 0x0E,
    PA_15 = 0x0F,
    // GPIOB
    PB_0  = 0x10,
    PB_1  = 0x11,
    PB_2  = 0x12,
    PB_3  = 0x13,
    PB_4  = 0x14,
    PB_5  = 0x15,
    PB_6  = 0x16,
    PB_7  = 0x17,
    PB_8  = 0x18,
    PB_9  = 0x19,
    PB_10 = 0x1A,
    PB_11 = 0x1B,
    PB_12 = 0x1C,
    PB_13 = 0x1D,
    PB_14 = 0x1E,
    PB_15 = 0x1F,
    // GPIOC
    PC_0  = 0x20,
    PC_1  = 0x21,
    PC_2  = 0x22,
    PC_3  = 0x23,
    PC_4  = 0x24,
    PC_5  = 0x25,
```

```
    PC_6  = 0x26,
    PC_7  = 0x27,
    PC_8  = 0x28,
    PC_9  = 0x29,
    PC_10 = 0x2A,
    PC_11 = 0x2B,
    PC_12 = 0x2C,
    PC_13 = 0x2D,
    PC_14 = 0x2E,
    PC_15 = 0x2F,
    // GPIOD
    PD_2 = 0x32,
    // GPIOH
    PH_0 = 0x70,
    PH_1 = 0x71,
    // Not connected
    NC = -1
} bsp_gpio_pin_names_t;
```

3.3 ADC

This optional function is used in the main example to read the MCU's internal temperature.

```
/*!
 *  Initializes the MCU ADC peripheral
 *
 * \param [IN] id   ADC interface id [1:N]
 */
void bsp_adc_init( const uint32_t id );

/*!
 *  Deinitialize the MCU ADC peripheral
 *
 * \param [IN] id   ADC interface id [1:N]
 */
void bsp_adc_deinit( const uint32_t id );

/*!
 * Sends out_data and receives in_data
 *
 * \param [IN] id       ADC interface id [1:N]
 * \param [IN] out_data adc conversion timeout in ms

 * \retval  RAW adc value, 0 if fail
 */
uint32_t bsp_adc_get_value( const uint32_t id, const uint32_t timeout_ms );
```

3.4 NVM

These functions store and restore the LoRa Basics™ Modem context.

The NVM block needs access to a persistent memory zone (Flash or EEPROM for example) with two different memory zones of 45 bytes.

Any usage of the Non-Volatile Memory functions must respect this API:

```
/*!
 * Restores the data context from an NVM device.
 *
 * \remark This method invokes memcpy - reads number of bytes from the address
 *
 *  \param addr   Flash address to begin reading from
 *  \param buffer Buffer pointer to write to
 *  \param size   Buffer size to read in bytes
 *  \retval       0 on success, negative error code on failure
 */
int32_t bsp_nvm_context_restore( const uint32_t addr, uint8_t* buf-
fer, const uint32_t size );

/*!
 * Stores the data context to a nvm device
 *
 *  \remark To be safer this function has to implement a read/check data
 *          sequence after programming
 *
 *  \param addr   Flash address to begin writing to
 *  \param buffer Buffer pointer to write from
 *  \param size   Buffer size to be written in bytes
 *  \retval       0 on success, negative error code on failure
 */
int32_t bsp_nvm_context_store( const uint32_t addr, const uint8_t* buf-
fer, const uint32_t size );
```

3.5 RNG

Any usage of the Random Number Generator functions must respect this API:

```
/*!
 * Returns a hardware generated random number.
 *
 * \retval random Generated random number
 */
uint32_t bsp_rng_get_random( void );

/*!
 * Returns a hardware generated unsigned random number between min and max
 *
 * \param [IN] val_1 first range unsigned value
 * \param [IN] val_2 second range unsigned value
 *
 * \retval random Generated random unsigned number between small-
est value and biggest
 * value between val_1 and val_2
 */
uint32_t bsp_rng_get_ran-
dom_in_range( const uint32_t val_1, const uint32_t val_2 );

/*!
 * Returns a hardware generated signed random number between min and max
 *
 * \param [IN] val_1 first range signed value
 * \param [IN] val_2 second range signed value
 *
 * \retval random Generated random signed number between small-
est value and biggest
 * value between val_1 and val_2
 */
int32_t bsp_rng_get_signed_ran-
dom_in_range( const int32_t val_1, const int32_t val_2 );
```

3.6 RTC

Any usage of the RTC functions must respect this API:

```
/*!
 *  Initializes the MCU RTC peripheral
 */
void bsp_rtc_init( void );

/*!
 * Returns the current RTC time in seconds
 *
 * \remark Used for scheduling autonomous retransmissions (i.e: NbTrans),
 *         transmitting MAC answers, basically any delay without accurate time
 *         constraints. It is also used to measure the time spent inside the
 *         LoRaWAN process for the integrated failsafe.
 *
 * retval rtc_time_s Current RTC time in seconds
 */
uint32_t bsp_rtc_get_time_s( void );

/*!
 * Returns the current RTC time in milliseconds
 *
 * \remark Used to timestamp radio events (i.e: end of TX), will also be used
 * for ClassB
 *
 * retval rtc_time_ms Current RTC time in milliseconds wraps every 49 days
 */
uint32_t bsp_rtc_get_time_ms( void );

/*!
 * Waits delay milliseconds by polling RTC
 *
 * \param[IN] milliseconds Delay in ms
 */
void bsp_rtc_delay_in_ms( const uint32_t milliseconds );
```

```c
/*!
 * Sets the rtc wakeup timer for seconds parameter. The RTC will gener-
ate an IRQ
 * to wakeup the MCU.
 *
 * \param[IN] seconds Number of seconds before wakeup
 */
void bsp_rtc_wakeup_timer_set_s( const int32_t seconds );

/*!
 * Sets the rtc wakeup timer for milliseconds parameter. The RTC will generate
 * an IRQ to wakeup the MCU.
 *
 * \param[IN] milliseconds Number of seconds before wakeup
 */
void bsp_rtc_wakeup_timer_set_ms( const int32_t milliseconds );
```

3.7 SPI

Any usage of the SPI functions must respect this API:

```
/*!
 *  Initializes the MCU SPI peripheral
 *
 * \param [IN] id   SPI interface id [1:N]
 * \param [IN] mosi SPI MOSI pin name to be used
 * \param [IN] miso SPI MISO pin name to be used
 * \param [IN] sclk SPI SCLK pin name to be used
 */
void bsp_spi_init( const uint32_t id, const bsp_gpio_pin_names_t mosi, const b
sp_gpio_pin_names_t miso,
                   const bsp_gpio_pin_names_t sclk );

/*!
 *  Deinitialize the MCU SPI peripheral
 *
 * \param [IN] id   SPI interface id [1:N]
 */
void bsp_spi_deinit( const uint32_t id );

/*!
 * Sends out_data and receives in_data
 *
 * \param [IN] id       SPI interface id [1:N]
 * \param [IN] out_data Byte to be sent

 * \retval in_data      Received byte.
 */
uint16_t bsp_spi_in_out( const uint32_t id, const uint16_t out_data );
```

3.8 Timer

Usage of the Timer functions must respect this API:

```
/*!
 *  Initializes the MCU TMR peripheral
 */
void bsp_tmr_init( void );

/*!
 * Starts the provided timer objet for the given time
 *
 * \param [in] milliseconds Number of milliseconds
 * \param [in] tmr_irq      Timer IRQ handling data ontext
 */
void bsp_tmr_start( const uint32_t millisec-
onds, const bsp_tmr_irq_t* tmr_irq );

/*!
 * Starts the provided timer objet for the given time
 */
void bsp_tmr_stop( void );

/*!
 * Returns the current TMR time in milliseconds
 *
 * \remark is used to timestamp ra-
dio events (end of TX), will also be used for
 *
 * retval tmr_time_ms Current TMR time in milliseconds wraps every 49 days
 */
uint32_t bsp_tmr_get_time_ms( void );

/*!
 * Enables timer interrupts (HW timer only)
 */
void bsp_tmr_irq_enable( void );

/*!
 * Disables timer interrupts (HW timer only)
 */
void bsp_tmr_irq_disable( void );
```

3.9 UART

Any usage of the UART functions must respect this API:

The functions using DMA are used only in the case of hardware modem utilization, otherwise a classic TX only for debug trace is enough.

```
/*!
 *  Initializes the uart1 peripheral
 */
void bsp_uart1_init( void );

/*!
 *  Initializes the uart2 peripheral
 */
void bsp_uart2_init( void );

/*!
 *  Deinitializes the uart1 peripheral
 */
void bsp_uart1_deinit( void );

/*!
 *  Deinitializes the uart1 peripheral
 */
void bsp_uart2_deinit( void );

/*!
 * Sends buffer on uart1 and using dma
 *
 * \param [IN] buff       buffer to send
 * \param [IN] size       buffer size
 */
void bsp_uart1_dma_start_rx( uint8_t* buff, uint16_t size );

/*!
 * Stop reception on uart1
 */
void bsp_uart1_dma_stop_rx( void );
```

```c
/*!
 * Sends buffer on uart1
 *
 * \param [IN] buff       buffer to send
 * \param [IN] size       buffer size
 */
void bsp_uart1_tx( uint8_t* buff, uint8_t len );

/*!
 * Sends buffer on uart2
 *
 * \param [IN] buff       buffer to send
 * \param [IN] size       buffer size
 */
void bsp_uart2_tx( uint8_t* buff, uint8_t len );
```

3.10 Watchdog

Any usage of the Watchdog functions must respect this API:

```c
/*!
 * Initializes the MCU watchdog peripheral
 *
 * \remark The watchdog period is equal to WATCHDOG_RELOAD_PERIOD seconds
 */
void bsp_watchdog_init( void );

/*!
 * Reloads watchdog counter
 *
 * \remark Application has to call this function periodically.
 *         The call period must be less than WATCHDOG_RELOAD_PERIOD
 *
 */
void bsp_watchdog_reload( void );
```

3.11 BSP Options

Only two or three constants need to be adapted for the new target.

- In case of hardware modem usage:

```
#define BSP_USER_UART_ID                            1
```

- For traces:

```
#define BSP_PRINTF_UART_ID                          2
```

- SPI ID:

```
#define BSP_RADIO_SPI_ID                            1
```

3.12 BSP RAL

If the board uses a TCXO, both these functions should be implemented:

```
ral_hal_status_t ral_hal_set_tcxo_on( const void* con-
text, uint16_t tcxo_startup_time_ms )

{

    // Please populate if user driven tcxo is used in project (tcxo_cfg.tcxo_c
trl_mode == RAL_TCXO_CTRL_HOST_EXT)

    return RAL_HAL_STATUS_OK;

}



ral_hal_status_t ral_hal_set_tcxo_off( const void* context )

{

    // Please populate if user driven tcxo is used in project (tcxo_cfg.tcxo_c
trl_mode == RAL_TCXO_CTRL_HOST_EXT)

    return RAL_HAL_STATUS_OK;

}
```

3.13 BSP MCU

This BSP is above the others and essentially uses the previously implemented BSP functions.

```c
/*!
 * Disables interrupts, begins critical section
 *
 * \param [IN] mask Pointer to a variable where to store the CPU IRQ mask
 */
void bsp_mcu_critical_section_begin( uint32_t* mask );


/*!
 * Ends critical section
 *
 * \param [IN] mask Pointer to a variable where the CPU IRQ mask was stored
 */
void bsp_mcu_critical_section_end( uint32_t* mask );


/*!
 * Disables MCU peripherals specific IRQs
 */
void bsp_mcu_disable_periph_irq( void );


/*!
 * Enables MCU peripherals specific IRQs
 */
void bsp_mcu_enable_periph_irq( void );


/*!
 * Initializes BSP used MCU
 */
void bsp_mcu_init( void );
```

```c
/*!
 * Disables irq at core side
 */
void bsp_disable_irq( void );


/*!
 * Enables irq at core side
 */
void bsp_enable_irq( void );


/*!
 * Resets the MCU
 */
void bsp_mcu_reset( void );


/*!
 * To be called in case of panic @mcu side
 */
void bsp_mcu_panic( void );


/*!
 * To be called in case of lr1mac stack issue
 */
void bsp_mcu_handle_lr1mac_issue( void );


/*!
 * This function is called when a basic modem reset is asked
 */
void bsp_mcu_modem_need_reset( void );
```

```c
/*!
 * Initializes BSP used MCU radio pins
 *
 * \param [IN] context Pointer to a variable holding the communication interfa
ce
 *                     id as well as the radio pins assignment.
 */
void bsp_mcu_init_radio( const void* context );


/*!
 * Sets the MCU in sleep mode for the given number of seconds.
 *
 * \param[IN] seconds Number of seconds to stay in sleep mode
 */
void bsp_mcu_set_sleep_for_s( const int32_t seconds );


/*!
 * Sets the MCU in sleep mode for the given number of milliseconds.
 *
 * \param[IN] milliseconds Number of milliseconds to stay in sleep mode
 */
void bsp_mcu_set_sleep_for_ms( const int32_t milliseconds );


/*!
 * Waits for delay microseconds
 *
 * \param [in] delay Delay to wait in microseconds
 */
void bsp_mcu_wait_us( const int32_t microseconds );
```

```c
/*!
 * Return the battery level
 *
 * \return battery level for lorawan stack
 */
uint8_t bsp_mcu_get_battery_level( void );
/*!
 * Prints debug trace
 *
 * \param variadics arguments
 */
void bsp_trace_print( const char* fmt, ... );


/*!
 * Suspend low power process and avoid looping on it
 */
void bsp_mcu_disable_low_power_wait( void );


/*!
 * Enable low power process
 */
void bsp_mcu_enable_low_power_wait( void );


/*!
 * Suspend once low power process and avoid looping on it once
 */
void bsp_mcu_disable_once_low_power_wait( void );
/*!
 * Return MCU temperature in celsius
 */
int32_t bsp_mcu_get_mcu_temperature( void );
```

4 Head and Stack Size

In order to run the LoRa Basics™ Modem correctly the stack and head sizes must be:

- Stack: 0x800
- Heap: 0x400

5 References

[1] SX1280 information: https://www.semtech.com/products/wireless-rf/lora-transceivers/SX1280ED1ZHP

6 Revision History

Version	Date	Modifications
1.0	April 2020	First Release

7 Glossary

BB	BaseBand
BoM	Bill Of Materials
BW	BandWidth
CLK	Clock
CW	Continuous Wave
ETSI	European Telecommunications Standard Institute
DFU	Device Firmware Update
EU	Europe
EUI	Extended Unique Identifier
GB	GigaByte
GPS	Global Positioning System
GW	GateWay
HAL	Hardware Abstraction Layer
HDMI	High-Definition Multimedia Interface
HW	HardWare
IP	Intellectual Property
ISM	Industrial, Scientific and Medical applications
LAN	Local Area Network
LBT	Listen Before Talk
LO	Local Oscillator
LoRa®	LOng RAnge modulation technique
LoRaWAN	LoRa® low power Wide Area Network protocol
LPF	Low Pass Filter
LSB	Least Significant Bit
LUT	Look Up Table
MAC	Media Access Control address
MCU	Micro-Controller Unit
MPU	Micro-Processing Unit
PA	Power Amplifier
RSSI	Received Signal Strength Indication
RF	Radio-Frequency
RX	Receiver
SAW	Surface Acoustic Wave filter
SD Card	Secure Digital Card
SF	Spreading Factor
SPI	Serial Peripheral Interface
SPDT	Single-Pole, Double-Throw switch
SSH	Secure SHell
SW	SoftWare
TX	Transmitter
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
USB	Universal Serial Bus

Important Notice

Information relating to this product and the application or design described herein is believed to be reliable, however such information is provided as a guide only and Semtech assumes no liability for any errors in this document, or for the application or design described herein. Semtech reserves the right to make changes to the product or this document at any time without notice. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. Semtech warrants performance of its products to the specifications applicable at the time of sale, and all sales are made in accordance with Semtech's standard terms and conditions of sale.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS, OR IN NUCLEAR APPLICATIONS IN WHICH THE FAILURE COULD BE REASONABLY EXPECTED TO RESULT IN PERSONAL INJURY, LOSS OF LIFE OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

Contact Information

Semtech Corporation
Wireless & Sensing Products
200 Flynn Road, Camarillo, CA 93012
E-mail: sales@semtech.com
Phone: (805) 498-2111, Fax: (805) 498-3804
www.semtech.com