

LoRa Module Low Level Development

Overview

Introduction

The product portfolio of the RAK LoRa node RAK4200 module. These modules come with a standard version of firmware that allows the customers to integrate quickly these modules in their solutions for LoRaWAN or LoRa P2P communication through the AT commands interface.

Further customization of the firmware can be done through the RUI Online compiler. At this layer the customized firmware interface with the hardware through the RUI Core abstraction layer. In RAK we called this a secondary firmware development/customization.

Additionally, RAK offers a third alternative for advanced customers who need to have deeper integration of their solutions with these modules. In this alternative, the customer could develop their own version of firmware that runs inside of the RAK modules.

How to Implement Your App on the RAK Module

Schematic

One of the essential aspects that allow customers to develop their own version of firmware is the module's hardware schematic. This allows the customers to understand the module's pinout, connections between the inner MCU and the LoRa transceiver.

 **NOTE**


There are two versions of the RAK4200 module: One for the high-frequency bands (i.e. 915 MHz, 866 MHz) and one for the low-frequency bands (i.e. 433 MHz). RAK4200 shares the same hardware connections between high frequency and low frequency models.

Porting LoRa Protocol Stack

When implementing the LoRa protocol stack, special attention must be paid to the SPI connections, since the LoRa transceivers are controlled by the MCU through an SPI interface. The important pins are the following: SPI1_MISO, SPI1_MOSI, SPI_NSS, and SPI_CLK. Additionally, the DIO and RFI paths are important as well to have a functioning LoRa communication. After that, RTC must be properly configured in the MCU to ensure accurate timing of the protocol stack during the runtime. Finally, the protocol stack code can be added after other pins are configured.

Application

Once the porting protocol stack is ready, customers can focus on the development of their applications. There are two options:

- a. Do not use the original bootloader that comes in RAK modules from the factory. In this case, the customer must provide his own version of the bootloader.
- b. Use RAK's bootloader and upgrade the custom firmware by using RAK's Device Firmware Upgrade Tool. You can download it from here:
 - [RAK Device Firmware Upgrade \(DFU\) Tool](#) 

Bootloader

Bootloader Introduction

In any MCU, after the power is connected, the System bootloader is in charge to bootstrap all the necessary to set up the Interrupt Vector table, initialize variables, and jump to the address of the main() symbol.

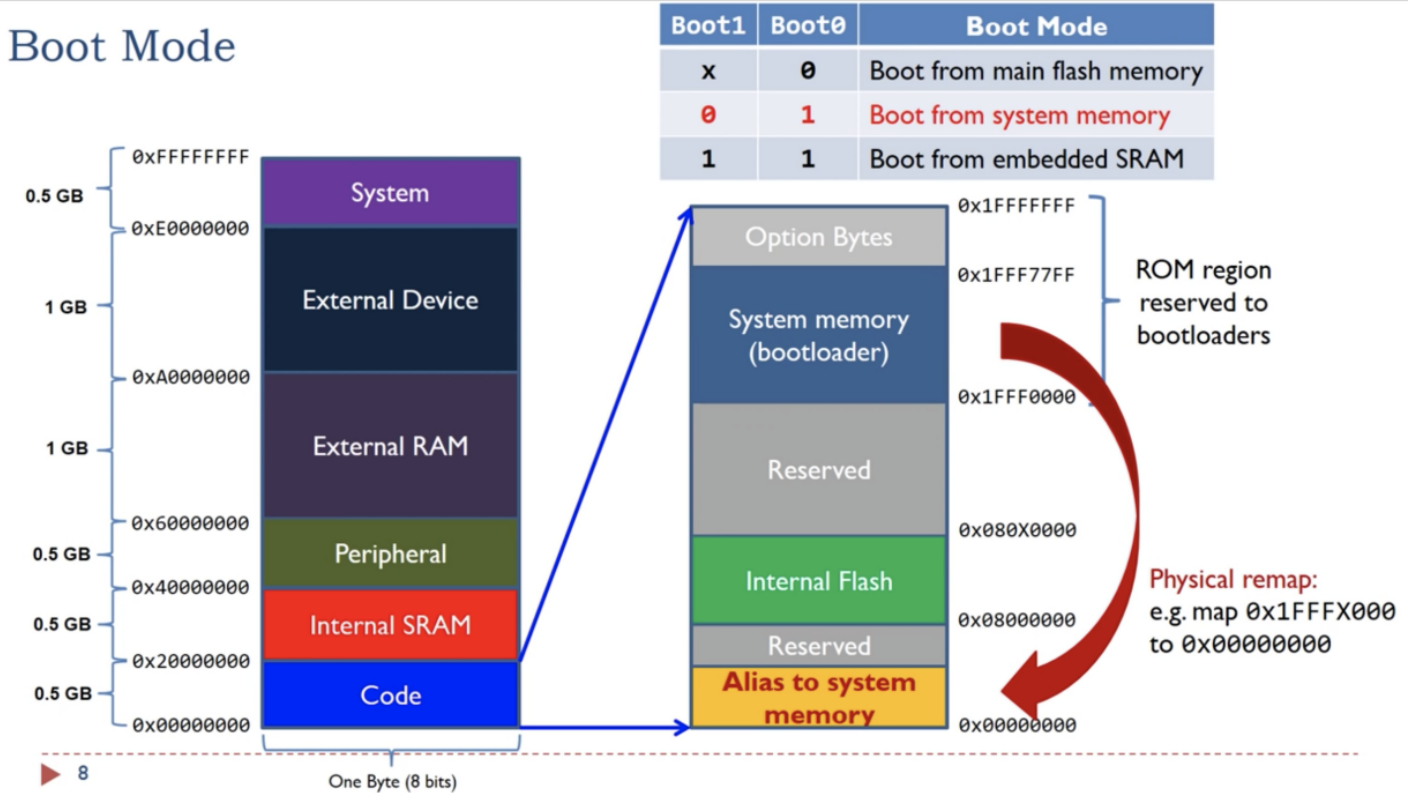


Figure 1: The flash section is between the 0x0800 0000 and 0x080X 0000. The X depends on the different models of MCU

The RAK's bootloader is stored in the internal flash section and has a size of 12K, located between 0x0800 0000 to 0x0800 2FFF. Its primary function is to write a new version of firmware received from the serial port into the flash memory section. The bootloader uses the Ymodem protocol and supervises internally the exceptions in the upgrade process. When the upgrade process is interrupted and restarted, the bootloader will detect abnormal events and enable the upgrade again.

RAK4200's bootloader uses the segment between 0x0808 1700 to 0x0808 17FF to store its parameters.

In the bootloader parameter storage area, 256 bytes are planned, but only two words are used to store the jump flag and upgrade status flag.

Finally, the serial port to communicate with the RAK's bootloader in these modules are the UART2 pins: PA9 (UART2_TX) and PA10 (UART2_RX). The parameters of the UART communication are 115200 / 8-N-1, which need to be properly configured in the RAK firmware upgrade tool.

Application Requirements

Since the RAK's bootloader is stored between the 0x0800 0000 and 0x0800 2FFF segments of the flash memory, the customer's application should be shifted accordingly. In the application code, you need to modify the interrupt vector table address as the following:

```
SCB->VTOR = FLASH_BASE | 0x3000;
```

The linker script must be updated accordingly. For example, in case you use GCC, please modify your linker script like the following:

```
FLASH (rx) : ORIGIN = 0x8003000, LENGTH = 116K
```

The customer's application firmware should implement a minimum of one AT command: `at +boot\r\n`. The function of this command is to jump from the application state into the bootloader state in preparation for the further application firmware upgrade. The logic of this command is the following:

- a) For RAK4200 , write the value 0x00000000 in the address 0x0808 1700.
- b) Reset MCU. You can call the `NVIC_SystemReset()` interface in the ST library to reset the MCU.

When initializing the clock in the mode, the LSE driver needs to be set to a high level. If you use ST's HAL library for development, the following code is placed at the beginning of the `SystemClock_Config` function.

```
HAL_PWR_EnableBkUpAccess();  
__HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_HIGH);
```

C

 **NOTE**

The bootloader disables the global interrupt when jumping from the application state. Therefore, when the application code is initialized, the global interrupt should be enabled again.

Last Updated: 11/17/2021, 9:05:23 AM
