RAK4270 Low Level Development Reference

Overview

RAK4270 module comes with a standard version of firmware that allows you to configure its functionality via AT commands. This simplicity helps you develop LoRa(P2P) and LoRaWAN projects quickly.

Further customization of the firmware can be done through the RUI (RAKwireless Unified Interface) Online compiler. RAK's LoRa modules support not only out-of-the-box integration via AT commands but also allow you to customize RAK4270 firmware and access other functionalities of the internal MCU using RUI API. More so, you can adapt and extend the logic in the firmware to meet your requirements.

Additionally, RAK offers a third alternative for advanced customers who need to have deeper integration of their solutions with these modules. In this alternative, you could develop your own version of STM32 firmware that runs inside of the RAK4270 module.

How to Implement Your Own Firmware

Schematic

One of the essential aspects that allows you to develop your own version of firmware is the RAK4270 Hardware Schematic . This allows you to understand the module's pinout and the connections between the internal STM32 MCU and the LoRa transceiver. Other important details can be found on RAK4270 Datasheet.

📝 **NOTE:**

There are two versions of the RAK4270 module: the high-frequency band RAK4270(H) used on EU868, US915, AU915, KR920, AS923, and IN865, and the low-frequency band RAK4270(L) used on EU433 and CN470. These two modules share the same schematic diagram which will be helpful to you when you develop your own firmware.

Porting Lora Protocol Stack

When implementing the LoRa protocol stack, special attention must be given to the SPI connections since the LoRa transceivers are controlled by the MCU through an SPI interface. Hence, the following are the important pins: **SPI1_MISO, SPI1_MOSI, SPI_NSS, SPI_CLK**.

Additionally, the DIO pins and RF signal paths are significant as well to have functional LoRa communication. Another important thing to consider is the RF switch logic table. The complete details of pin connections can be found on the RAK4270 Datasheet.

After that, the **Real-Time Clock (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 configuring the other pins.

Application

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

- 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.

- 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
    - Device Firmware Upgrade Tool for MacOS
    - Device Firmware Upgrade Tool for Ubuntu

If you want to fully develop your own, you can refer to the schematic diagram and the datasheet of the MCU to implement the code. If you want to use RAK’s bootloader, continue reading the next section.

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.

In Figure 1, it shows a usual memory map for an ARM Cortex M0+ MCU, which is the architecture of the MCU of the RAK4270.

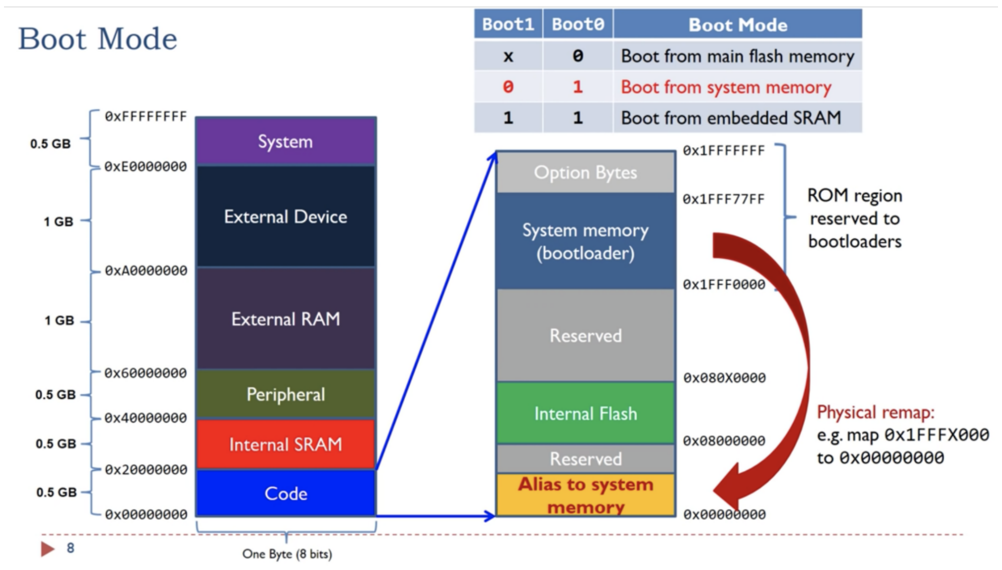

Figure 1: Memory map for an ARM Cortex M0+ MCU

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 all possible exceptions internally during the upgrade process. When the upgrade process is interrupted, the bootloader will detect abnormal events, and the FW upgrade will fail. You can perform the FW upgrade again using the bootloader after recycling the power.

RAK4270’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 is the UART1 (pin PA9, pin PA10). The parameters of the UART1 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, your 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
```

In the linker, the script must be updated accordingly. For example, in case you use GCC, modify your linker script as follows:

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

Your application firmware should implement as minimum 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:

- For RAK4270, write the value 0x00000000 in the address 0x0808 1700.
- Reset MCU. You can call the **NVIC_SystemReset()** interface in the ST library to reset the MCU.

> 📝 **NOTE**
>
> The bootloader turned off the global interrupt when jumping from the application state. Therefore, when the application code is initialized, the global interrupt should be turned on again.

Last Updated: 7/29/2022, 10:17:19 PM