Explore nRF54L15 with SX1261MB2BAS for LoRa Communication

A

Project Report

Submitted by

Aanjaneya Pandey
(Research Intern from NIT DELHI, New Delhi, India)

Under the able guidance of
Dr. Hari Prabhat Gupta

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI
Varanasi 221005, India
JULY 2025

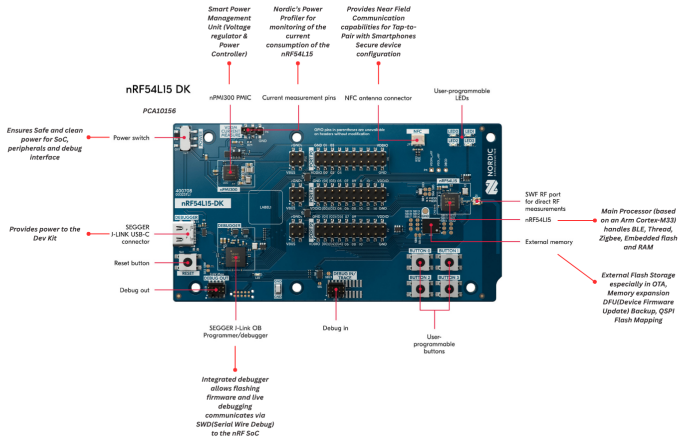
Table of Contents

1. Introduction

1.1. Project Objectives

The project was broken down into two specific, measurable objectives:

1. **Establish LoRa-to-LoRa Communication:** To implement and verify a direct, point-to-point communication link between two identical devices, each comprising an nRF54L15 SoC and an SX1261 LoRa module.
2. **Establish LoRaWAN Communication:** To develop a LoRaWAN-compliant end-node using the same hardware and successfully transmit data to an outdoor LoRaWAN gateway, confirming its reception on a network server.



2. Development Environment Setup on Windows

A correct and consistent development environment is the foundation for any successful embedded project. The following steps were taken to configure a Windows 11 machine for nRF54 development.

2.1. Procedure to download the required software

1. Download the “nRFConnect for Desktop” through the official website and then install it.

https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-Desktop/Download#infotabs

2. Download and install VS Code and also Segger JLink (latest version).

   https://code.visualstudio.com/download & https://www.segger.com/downloads/jlink/

3. Inside VS code extensions pack, search for **“nRF Connect for VS Code**” extension and download it, after that open that extension and from there install the toolchain (version example : v2.9.1).

4. Now open nRFConnect for Desktop application, install the Toolchain manager from here, after that open it and install the SDK of the same version, example v2.9.1 installed from the VSCode from here.
5. Once installed click on Open VS Code from there as shown in the SDK Environment (above image).
6. Now you have the proper west working environment to work and build applications.
7. Now go to the nRFConnect Extension on the VS Code left sidebar and click on 'Create application', it will pop up the choices.
8. Now select the options **"Copy a sample"** to run the sample program eg: "blinky" and select it. Now a directory appears and you have to select the default shown there through popup appear *(generally C drive and inside the ncs folder example : c:\ncs\my_workspace\program_name).*
9. Now click on "*Add Build configuration*" make sure to select the sdk and toolchain previously installed and also make sure the board target should be *nRF54l15dk/nrf54l15/cpuapp* then click *Generate Build*, If all the things are properly installed then the build won't give an error else you have to see the above steps again.
10. After building, go to the terminal option and open the New Terminal and in the terminal, type **"nrfjprog --eraseall"**. This command erases the previous program(if any) in nRF54L15dk.
11. Next in the Terminal type **"nrfjprog --program build\merged.hex –verify --reset"**. This flashes the program to your board, and your program should now be running.
12. Once you add the build configuration from now onwards after changing anything will not be required to add the configuration again just press the build it will do the pristine build.

2.2. Critical Build and Flash Procedure

- **NOTE :** This is the crucial step for all types of Flashing.
  "Don't press the Flash icon on the Action. tab"
- **NOTE** : **Don't change your CMakeLists.txt ever** since it is generated on its own and are particular for that project only.(Changing it might cause CMake error during build.

3. Point-to-Point (P2P) LoRa Communication

3.1. Objective

To create and test a direct communication link between two LoRa devices(nRF54L15 and SX1261MB2BAS Shield). One device will act as a transmitter, periodically sending a data packet, while the other will act as a receiver, listening for and displaying the received data.

3.2. Hardware Integration and Pinout

LoRa Module Signal	nRF54L15 Pin	GPIO/Peripheral Mapping	Purpose / Notes
SPI SCK	P1.08	`NRF_PSEL(SPIM_SCK, 1, 8)`	SPI Clock
SPI MISO	P1.05	`NRF_PSEL(SPIM_MISO, 1, 5)`	SPI Master In
SPI MOSI	P1.06	`NRF_PSEL(SPIM_MOSI, 1, 6)`	SPI Master Out
SPI CS (NSS)	P1.07	`<&gpio1 7> GPIO_ACTIVE_LOW`	SPI Chip Select
RESET	P1.04	`<&gpio1 4> GPIO_ACTIVE_LOW`	LoRa Module Reset
BUSY	P1.09	`<&gpio1 9> GPIO_ACTIVE_HIGH`	Indicates module is busy
DIO1 (IRQ)	P1.11	`<&gpio1 11> GPIO_ACTIVE_HIGH`	LoRa IRQ/Interrupt
ANT_SW (Antenna Enable)	P1.10	`<&gpio1 10> GPIO_ACTIVE_HIGH`	Controls RF switch/antenna

3.3. Software Configuration (DeviceTree & Kconfig)

- Now after connecting the two we have to go create an application and select the **“Create a blank Application”** in the nrf connect for the VS Code and select the directory

(example : c:\ncs\my_workspace\program_name).

- Now a application is created go to it and add the ***“board”*** folder in it and inside it add a file name : ***“nRF54l15dk_nrf54l15_cpuapp.overlay”***, it should look like this :

3.4. Application Logic: Transmitter & Receiver

Add the code in the following file (see there are two application that needed to be created for transmitter and receiver) as placed :

Transmitter & Receiver ::

- We have to write the logic for the main.c only, rest for the overlay and prj.conf file you can refer the appendix 6.1.
- Make sure to add the same details for the communication in the receiver and transmitter

code :

```
struct lora_modem_config config = {
    .frequency    = 868000000,
    .bandwidth    = BW_125_KHZ,
    .datarate     = SF_7,
    .preamble_len = 8,
    .coding_rate  = CR_4_5,
    .tx_power     = 14,
    .tx           = true
};
```

3.5. Building, Flashing, and Execution

- Now go to the nRf Connect in the left side bar, and in the APPLICATIONS section click on the “**Add Build Configuration**” now the same process of building and flashing takes place as shown above.

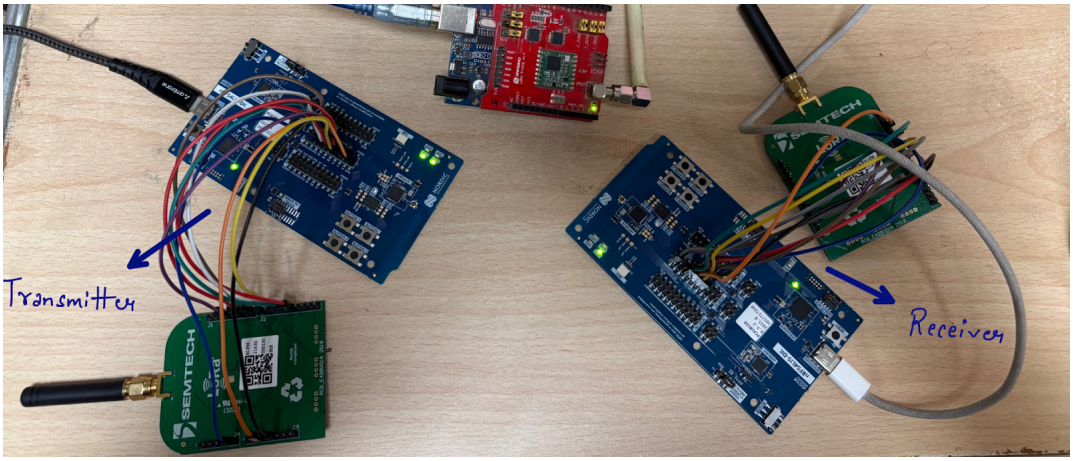
- Select the particular board name nRF54l15dk/nRF54l15/cpuapp, make sure to select the same toolchain and SDK version.
- On flashing use the command :“nrfjprog --eraseall ” **then** “nrfjprog --program build\merged.hex –verify --reset” **and if there is** some memory issue during flashing use the command “nrfjprog --recover” .

3.6. Results and Verification

Once the program is flashed you can see the following result with the blinking of led which confirms that packets are being sent and received on the nodes respectively.

```
16:02:26.005 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -54
16:02:31.056 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -54
16:02:36.140 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -54
16:02:41.223 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -53
16:02:46.294 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -60
16:02:51.372 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -56
16:02:56.431 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -59
16:03:01.516 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -62
16:03:06.597 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -53
16:03:11.646 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -53
16:03:16.725 -> Received packet: 'Hello from nRF54L15 + SX1261!' with RSSI -54
```

Actual Circuitry :



4. LoRaWAN End-Node to Gateway Communication

4.1. Objective

To configure the nRF54+SX1261 device as a LoRaWAN end-node and transmit data to a LoRaWAN gateway. The successful reception of this data will be verified on a cloud-based LoRaWAN Network Server.

4.2. LoRaWAN Network Fundamentals

- **Network Server:** A cloud service that manages the network, de-duplicates packets, and routes data to the correct application. For this project, For this project, **ChirpStack** was used as the open-source LoRaWAN Network Server.
- **OTAA (Over-the-Air Activation):** The preferred and more secure method for a device to join the network. The device uses a set of unique credentials to perform a "join procedure" with the network.
- **Credentials:** Three keys are required for OTAA:
    - **Device EUI (DevEUI):** A globally unique 64-bit identifier for the device itself.
    - **Join EUI (AppEUI):** A 64-bit identifier for the application on the network server.
    - **AppKey:** A 128-bit secret key shared between the device and the network server, used to authenticate the device.

4.3. Network Server Configuration (ChirpStack Configuration)

- Open the docker app for the desktop to start the ChirpStack.
- Log in to the gateway (use the in865 server) add the id’s required and create the device profile and in the application add the device with the keys.
- Make sure your device is showing the status on the Dashboard.

4.4. End-Node Software Configuration

- Pin-diagram is same as in the case of the P2P LoRa communication. Also the same overlay is used.

LoRa Module Signal	nRF54L15 Pin	GPIO/Peripheral Mapping	Purpose / Notes
SPI SCK	P1.08	`NRF_PSEL(SPIM_SCK, 1, 8)`	SPI Clock
SPI MISO	P1.05	`NRF_PSEL(SPIM_MISO, 1, 5)`	SPI Master In
SPI MOSI	P1.06	`NRF_PSEL(SPIM_MOSI, 1, 6)`	SPI Master Out
SPI CS (NSS)	P1.07	`<&gpio1 7> GPIO_ACTIVE_LOW`	SPI Chip Select
RESET	P1.04	`<&gpio1 4> GPIO_ACTIVE_LOW`	LoRa Module Reset
BUSY	P1.09	`<&gpio1 9> GPIO_ACTIVE_HIGH`	Indicates module is busy
DIO1 (IRQ)	P1.11	`<&gpio1 11> GPIO_ACTIVE_HIGH`	LoRa IRQ/Interrupt
ANT_SW (Antenna Enable)	P1.10	`<&gpio1 10> GPIO_ACTIVE_HIGH`	Controls RF switch/antenna

- Now the previous steps are same you have to create new Application *(in the same workspace)* add the boards folder with .overlay file remember your program structure is like this :

- Now add the specific code to the particular file :

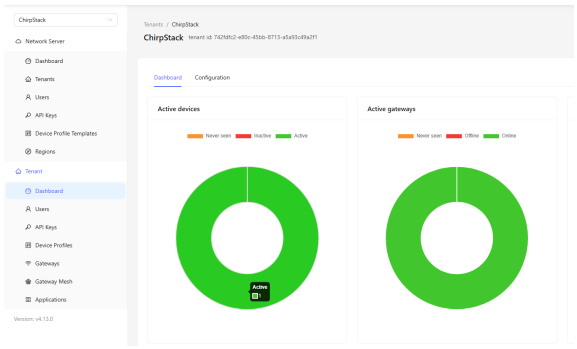
You have to write the logic for the main.c and for the overlay and prj.conf file you can refer Appendix 6.2, 6.3. Also Don't change the **CMakelists.txt** file as it may generate build error.

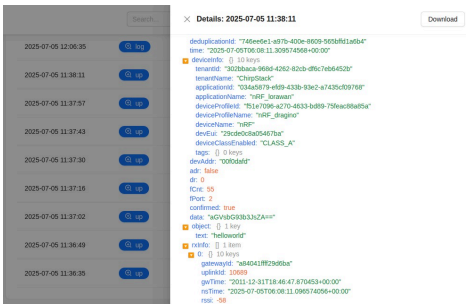
- In the code add your device keys *(`DEV_EUI ,JOIN_EUI ,APP_KEY`)* make sure the dev_eui is in MSB *(not in reverse order)*.

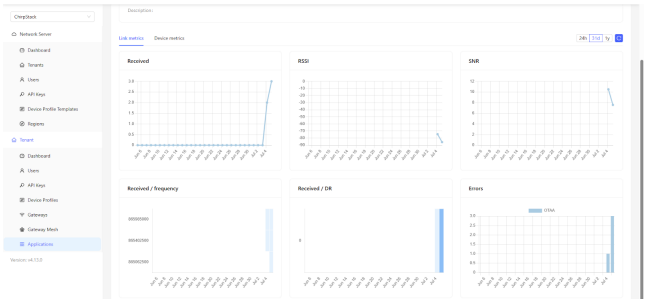
4.5. Building, Flashing, and Execution

- Now go to the nRf Connect in the left side bar, and in the APPLICATIONS section click on the "**Add Build Configuration**" now the same process of building and flashing takes place as shown above.
- Select the particular board name nRF54l15dk/nRF54l15/cpuapp, make sure to select the same toolchain and SDK version.
- On flashing use the command :"nrfjprog --eraseall " **then** "nrfjprog --program build\merged.hex –verify --reset" **and if there is** some memory issue during flashing use the command "nrfjprog --recover" .
- Now wait for a few seconds to see the results.
- Add the Javascript code for decrypting the sent data *(in the codec section)*.

4.6. Results and Verification







5. Conclusion

5.1. Summary of Achievements

This project successfully met all its objectives. A complete Windows-based development environment for the nRF54L15 was established. Both Point-to-Point LoRa and full LoRaWAN network communication were implemented and empirically verified. The successful P2P test validated the hardware integration and driver configuration, while the successful LoRaWAN test confirmed the device's ability to interoperate with a standard, public network infrastructure. This work provides a solid and reliable foundation for developing more complex IoT applications on this hardware platform.

6. Appendices

6.1. Appendix A: P2P Project Configuration (`prj.conf`)

```
# prj.conf

# --- Zephyr Kernel ---
CONFIG_MAIN_STACK_SIZE=2048
# --- Required Drivers ---
CONFIG_GPIO=y
CONFIG_SPI=y
CONFIG_PINCTRL=y
# --- LoRa and SX126x Driver ---
CONFIG_LORA=y
CONFIG_LORA_SX126X=y
# --- Logging and Console Configuration ---
# Keep logging enabled
CONFIG_LOG=y
CONFIG_LOG_MODE_IMMEDIATE=y
# CRITICAL: Disable the UART console to free up UARTE00 for SPIM00
CONFIG_SERIAL=n
CONFIG_UART_CONSOLE=n
CONFIG_LOG_BACKEND_UART=n
# CRITICAL: Enable the RTT backend for logging
CONFIG_LOG_BACKEND_RTT=y
CONFIG_USE_SEGGER_RTT=y
```

6.2. Appendix B: LoRaWAN Project Configuration (`prj.conf`)

```
CONFIG_LOG=y
CONFIG_SPI=y
CONFIG_GPIO=y
CONFIG_LORA=y
CONFIG_LORAWAN=y
CONFIG_LORAMAC_REGION_IN865=y
```

```
CONFIG_MAIN_STACK_SIZE=2048
CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE=2048
```

6.3. Appendix C: Hardware Overlay File (`<board>.overlay`)

```
/ {
    aliases {
        lora0 = &lora_semtech_sx1261mb2bas; // or the actual connected pin
    };
};

&pinctrl {
    spi20_default: spi20_default {
     group1 {
      psels = <NRF_PSEL(SPIM_SCK, 1, 8)>,
              <NRF_PSEL(SPIM_MISO, 1, 5)>,
              <NRF_PSEL(SPIM_MOSI, 1, 6)>;
     }; };
    spi20_sleep: spi20_sleep {
        group1 {
         psels = <NRF_PSEL(SPIM_SCK, 1, 8)>,
                 <NRF_PSEL(SPIM_MISO, 1, 5)>,
                 <NRF_PSEL(SPIM_MOSI, 1, 6)>;
        };
       };
};
&spi20 {
    status = "okay";
    cs-gpios = <&gpio1 7 GPIO_ACTIVE_LOW>;
    pinctrl-0 = <&spi20_default>;
    pinctrl-1 = <&spi20_sleep>;
    pinctrl-names = "default", "sleep";
    max-frequency = < 8000000 >;
    lora_semtech_sx1261mb2bas: sx1261@0 {
        compatible = "semtech,sx1261";
        reg = <0>;
        spi-max-frequency = <8000000>;
        label = "SX1261";
        reset-gpios = <&gpio1 4 GPIO_ACTIVE_LOW>;
        busy-gpios = <&gpio1 9 GPIO_ACTIVE_HIGH>; // done
        antenna-enable-gpios = <&gpio1 10 GPIO_ACTIVE_HIGH>;
        dio1-gpios = <&gpio1 11 GPIO_ACTIVE_HIGH>; // done
        dio2-tx-enable;
        tcxo-power-startup-delay-ms = <5>;
    };
};
&uart20 {
    status = "disabled";
};
```