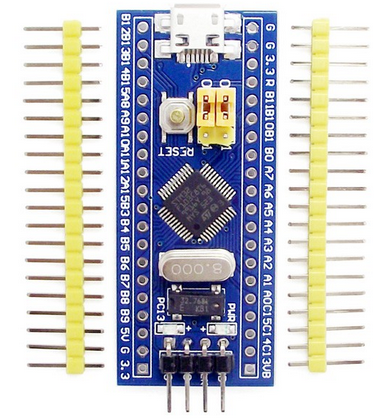
Lorawan node with STM32F103C8 and RFM95.

# Introduction.

This document describes the hardware and the software for a LoRa Wan node (EU version) with an STM32F103C8 microcontroller and an RFM95 LoRa radio module. The microcontroller has enough flash memory (64 kB) to hold the application and the lmic library. The software can easily be adapted for using any available sensor.  
Software is based on ttn\_otaa by Thomas Telkamp and Matthijs Kooijman.

# STM32F103C8.

I used a cheap Chinese board fort his project:



Th SPI-1 bus was used for communication with the RFM95. The Serial1 port (pins A10 and A9) was used for debug output.

# RFM95.

This tiny board looks like figure 1, I used a small perfboard to make the unit breadboard friendly, see figure 2. The result is in figure 3. The antenna is directly connected to the RFM95 board (right top).

|  |  |  |
| --- | --- | --- |
| Afbeeldingsresultaat voor rmf95  Figure 1 | Figure 2 | Figure 3 |

# Wiring STM32F103C -> RFM95

The following connection were made to connect the RMF95 to the STM32:

|  |  |  |
| --- | --- | --- |
| RFM95 | STM32F103C | Description |
| RESET | RST | Reset switch |
| NSS | PA4 | CS/SS |
| SCK | PA5 | SPI - SCK |
| MOSI | PA7exit | SPI - MOSI |
| MISO | PA6 | SPI - MISO |
| GND | GND | Ground |
| 3.3V | 3.3V | 3.3 Volt |
| DIO0 | PA0 | IRQ |
| DIO1 | PA1 | DIO1 |

In the Arduino sketch, the lmic must be configured as follows:

#define RESET 0xFF // Reset not used

#define DIO0 PA0

#define DIO1 PA1

#define SS PA4 // SPI-1 default CS

#define DIO2 0xFF // DI02 not used

const lmic\_pinmap lmic\_pins = {

.nss = SS,

.rxtx = 0xFF,

.rst = RESET,

.dio = { DIO0, DIO1, DIO2 },

} ;

# Software.

The software is developed as a sketch for the Arduino IDE. The sketch is called “STM32\_LoRa” and is available on [github](https://github.com/Edzelf/LoRa/tree/master/STM32_LoRa).

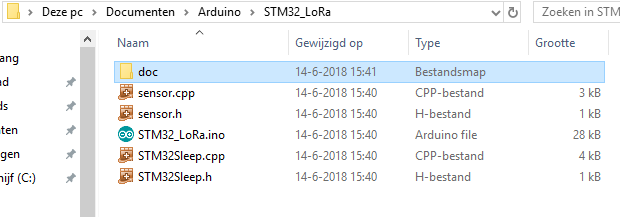
The sketch uses OTAA to connect to TTN. After successful joining the network, the payload is transmitted and network keys are saved in the ESP32’s EEPROM. That way, successive connections are made through ABP. The sequence number of the payload is kept in RTC memory, and after every 100 transmissions also in EEPROM. As long as the power is not removed from the STM32, the RTC memory will survive. If power is removed, the sequence number will be retrieved from EEPROM. This number in EEPROM may be lower than the actual sequence number, therefore it is incremented by 100 in this situation.

# Sleep mode.

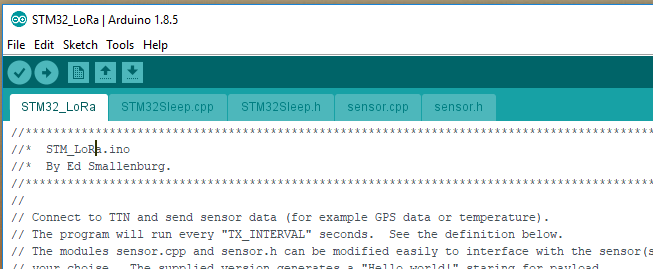
After transmission of the payload, the device goes into deep sleep to save power (batteries). In the current configuration it will It will be restarted every “TX\_INTERVAL” (see configuration section).

# Quick start.

Download the project as a zip-file and put the contents in a map in your Arduino sketch directory, for example C:\Users\<User>\Documents\Arduino\STM32\_Lora. It shoul look like:



No You can compile and upload the application to your STM32F103C. You will see several tabs in the IDE:



The first tab is the main program. Here you have to configure your TTN settings in the area marked:  
 “C O N F I G U R A T I O N S E C T I O N”.

The tab with “sensor.cpp” hold the part that is responsible for the conversion of the sensor data to the payload and can be modified to your needs. The current version is just an example and fills the payload with the string “Hello world!”.

During compilation there may be some warnings about “ASSERT”. You can ignore that.