**МИНОБРНАУКИ РОССИИ**

федеральное государственное бюджетное образовательное

учреждение высшего образования

**«Национальный исследовательский университет «МЭИ»**

Кафедра ОРТ

Пояснительная записка к проекту

**«IT Академия Samsung» по треку «Интернет вещей» на тему**

Система сбора данных по технологии LoRa для задач Интернета Вещей

**Автор: Зиновьев А. А.**

**Преподаватель: Стрелков Н.О.**

Москва 2022

# Аннотация

Данная пояснительная записка содержит информацию о ходе разработки системы сбора данных по технологии LoRa для задач Интернета Вещей, характеристики системы и также результаты тестирования.

Разработка вдохновлена работой "Application of Wi-Fi and LoRa Technologies for Wireless Measurement of Physical Quantities" за авторством Стрелкова Н.О. и Михайлова М.С.

Оглавление

[Аннотация 2](#_Toc107430283)

[1. Принцип работы 4](#_Toc107430284)

[2. Архитектура системы 4](#_Toc107430285)

[3. Разработка сенсорного узла с интерфейсом LoRa 4](#_Toc107430286)

[3.1. Комплектующие 4](#_Toc107430287)

[3.1.1. Плата с микроконтроллером 4](#_Toc107430288)

[3.1.2. Периферийные устройства и датчики 5](#_Toc107430289)

[3.1.3. Выбор частотного диапазона и модуля LoRa 8](#_Toc107430290)

[3.2. Логика и алгоритм 8](#_Toc107430291)

[3.3. Программные модули 9](#_Toc107430292)

[3.4. Результат разработки сенсорного узла 10](#_Toc107430293)

[3.5. Расчет себестоимости сенсорного узла 11](#_Toc107430294)

[4. Программирование приемного шлюза с интерфейсами LoRa и Wi-Fi/4G 12](#_Toc107430295)

[4.1. Выбор приемного шлюза 12](#_Toc107430296)

[4.2. Логика и алгоритм 12](#_Toc107430297)

[4.3. Программные модули 13](#_Toc107430298)

[5. Тестирование собранной системы 13](#_Toc107430299)

[*Приложение 1* 16](#_Toc107430300)

[*Приложение 2* 17](#_Toc107430301)

[*Приложение 3* 18](#_Toc107430302)

[*Приложение 4* 22](#_Toc107430303)

[*Приложение 5* 23](#_Toc107430304)

[*Приложение 6* 24](#_Toc107430305)

1. **Принцип работы**

Система предназначена для сбора данных с различных устройств, передающих данные при помощи электрических импульсов микроконтроллеру с последующей передачей данных по сети LoRaWAN приемному шлюзу, который в свою очередь перенаправляет данные в сеть Интернет посредством 4G на сервера «облачных» сервисов для их последующей обработки и визуализации.

1. **Архитектура системы**

Система состоит из сенсорного узла, который собирает данных с периферийных устройств, приемного шлюза, выполняющего перенаправление данных в сеть Интернет, сервера центра обработки данных (ЦОД), представляющим собой «облако» и пользовательский интерфейс (см. рис. 1).

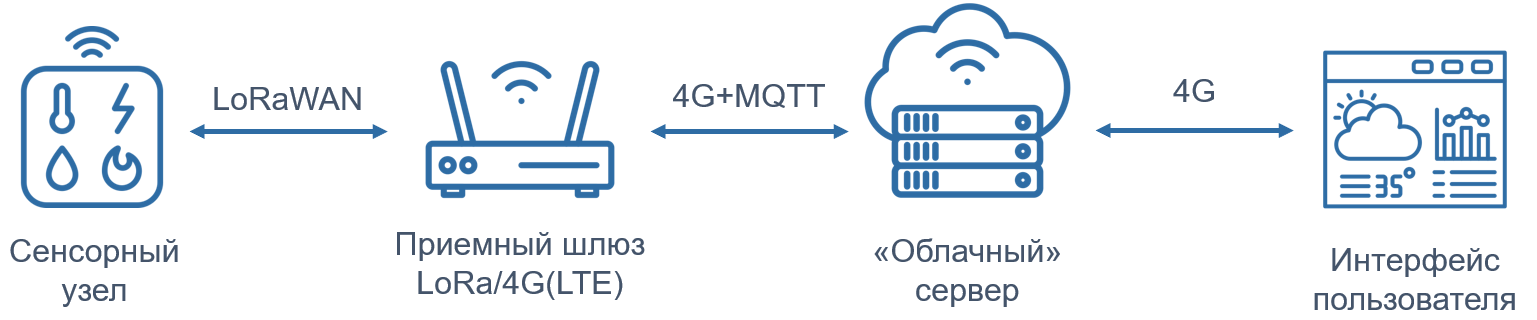


Рисунок 1 – Архитектура системы

1. **Разработка сенсорного узла с интерфейсом LoRa**
   1. **Комплектующие**
      1. **Плата с микроконтроллером**

В качестве аппаратной базы используется отладочная плата NUCLEO-64 F446RE (см. рис. 2) c микроконтроллером STM32 c архитектурой ARM. Выбор данной платы обусловлен ее наличием у автора на разработки.

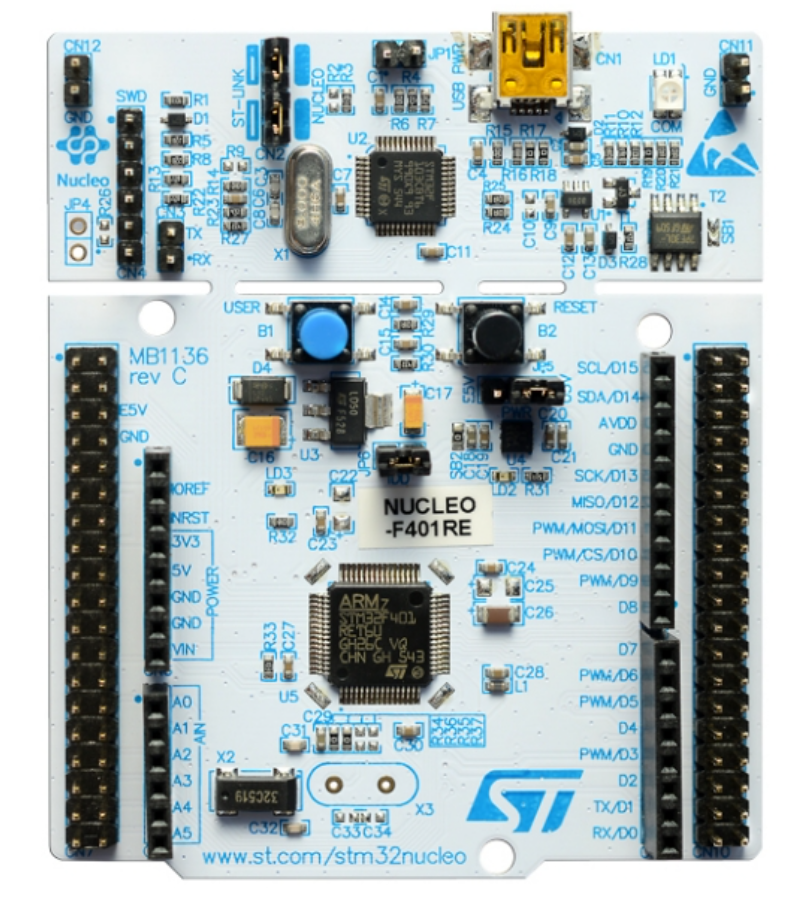


Рисунок 2 – Отладочная плата Nucleo-64 F446RE

**Достоинства**: плата имеет большинство используемых и поддерживаемых интерфейсов подключения периферийных устройств: I2C, SPI, 1-Wire, USART и т.д. Производительность платы является избыточной для задач, которые будут на ней выполняться. Arduino-совместимый форм-фактор, что позволяет подключать совместимые внешние устройства, расширители и т.д.

**Недостатки**: средняя энергоэффективность для рассматриваемых задач, которые плата будет выполнять. Геометрические размеры так же велики сравнительно других плат. Относительно высокая цена. Вышеперечисленными недостатками можно пренебречь в условиях создания макета.

* + 1. **Периферийные устройства и датчики**

**Минимальный функционал**, которым должна обладать устройство – **снятие параметров окружающей среды** (температура, влажность и/или давление). В соответствии с этим требованием к устройству долен быть подключен метеодатчик. На выбор можно установить любой метеодатчик. В данной конфигурации установлен один из двух вариантов (см. табл. 1).

**Таблица 1**

**Сравнение метеодатчиков**

|  |  |  |
| --- | --- | --- |
|  | SHT31 | BME280 |
| Снимаемые параметры | Температура, влажность | Температура, влажность, давление |
| Поддерживаемые интерфейсы | I2C | I2C, SPI |
| Питание | 2.4 – 5.0 В | 1.7 – 3.6 В |
| Диапазон измерений | Т: -40…+90oC  RH: 0…100% | Т: -40…+85oC  RH: 0…100%  PA: 300…1100 hPa |
| Точность измерений | Т: ±0.3oC  RH: ±3% | Т: ±0.5oC  RH: ±0.3%  PA: ±1 hPa |

На момент разработки датчик **SHT31** (см. рис. 3) уже имелся в наличии, поэтому решение было принято в пользу этого варианта, однако при дальнейшей разработке ПО будет учитываться возможность подключения BME280.

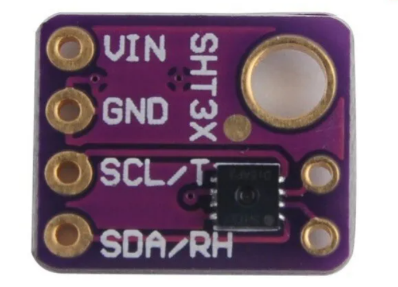
****

Рисунок 3 – Метеодатчик SHT31

Вторым, на что придется обратить внимание, является возможность установки на мобильном объекте, из чего вытекают **два условия**: **автономность питания и определение местоположения**.

**Питание** будет обеспечиваться расширительным Arduino-совместимым модулем **Battery Shield** со встроенным литий-ионным аккумулятором на 3200 мАч (см. рис. 4). Данный модуль имеет коннекторы, совпадающие с расположением таковых у отладочной платы. Подключение происходит путем вставки ножек Battery Shield в соответствующие коннекторы на отладочной плате. Вся остальная периферия подключается уже поверх аккумуляторного модуля (см. приложение 1). Модуль полностью дублирует входы/выходы отладочной платы.



Рисунок 4 – аккумулятор Battery Shield

**Определение местоположения** будет определяться модулем **GPS NEO-6M** (см. рис. 5). Данный модуль имеет интерфейс UART. Имеет EEPROM (память) для сохранения настроек модуля, а также встроенную батарею для сохранения координат спутников и индикацию наличия связи.

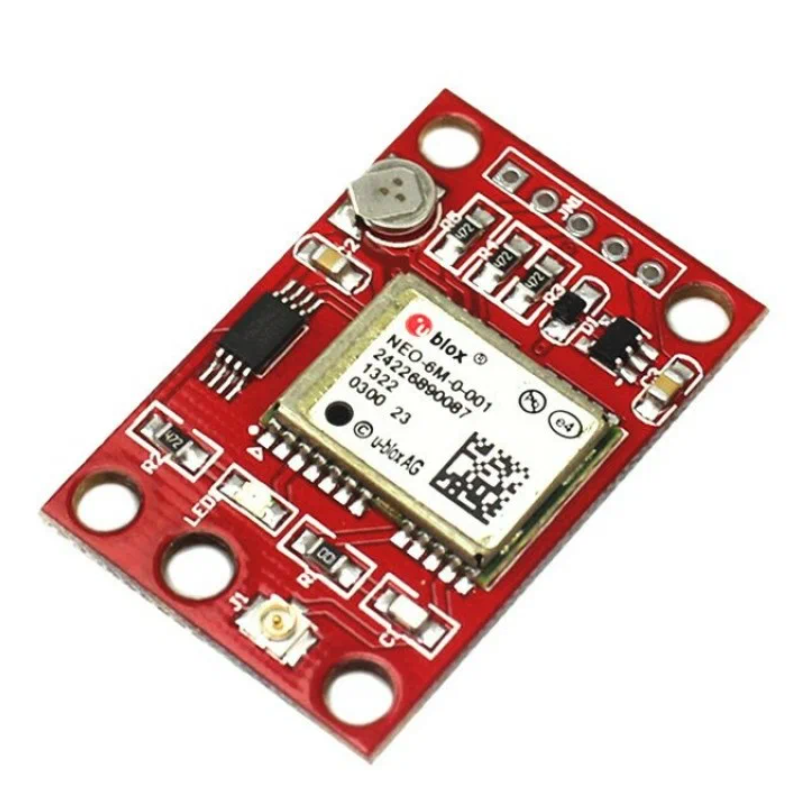


Рисунок 5 – GPS-модуль U-blox NEO-6M

* + 1. **Выбор частотного диапазона и модуля LoRa**

В Российской Федерации разрешенный диапазон для LoRaWAN является диапазон с центральной частотой 868,6 МГц. Для удешевления макета системы был выбран модуль на 433 МГц, что ввиду малой мощности передачи не нарушает законодательство.

В качестве модуля используется Ai-Thinker RA-02 (см. рис. 6), построенный на базе SX1278. Интерфейс подключения SPI. Данный модуль сравнительно дешевый, обладает удобным форм-фактором.

Для создания конечного устройства придется заменить модуль LoRa на любой, построенный на SX1276, который поддерживает диапазон 868.6 МГц.



Рисунок 5 – Модуль LoRa Ai-Thinker RA-02 433 МГц

* 1. **Логика и алгоритм**

Показания с датчика передаются на микроконтроллер по интерфейсу I2C. Координаты с GPS-модуля передаются на микроконтроллер по интерфейсу SPI. Микроконтроллер создает пакет из полученных данных, и передает их через радио-модуль LoRa приемному устройству.

Согласно описанной выше логики составлена структурная схема сенсорного узла (см. рис. 6).

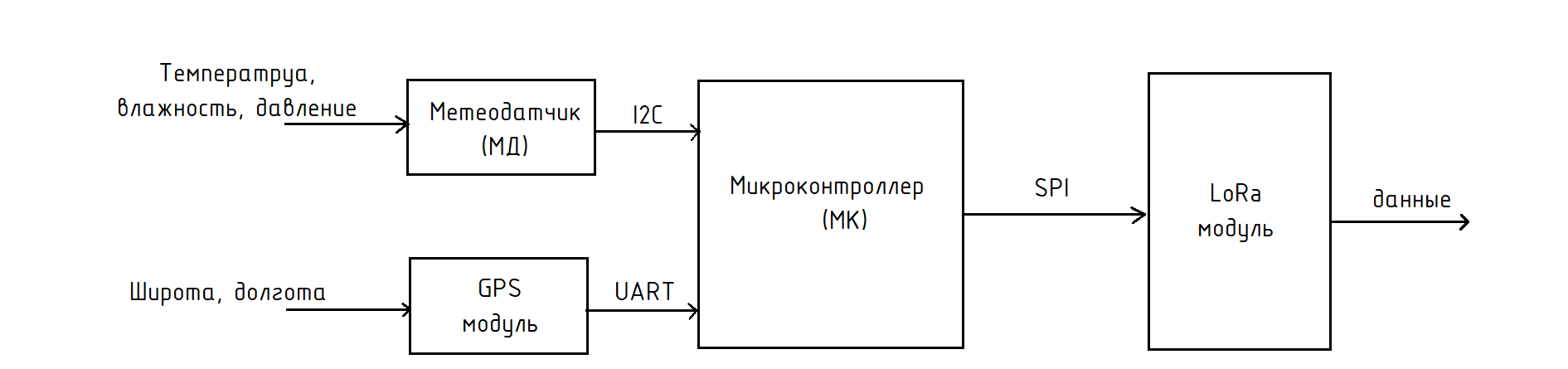


Рисунок 6 –Структурная схема сенсорного узла системы

**Алгоритм программы**: программа проверяет состояние подключенных модулей при помощи опроса их статуса. Если модули дают ответ на запрос, то программа выводит сообщение в терминал об успешной инициализации. Если модули не отзываются – выводит сообщение в терминал об ошибке для каждого из модулей. Затем программа проверяет установленную частоту передачи и в случае совпадения с необходимой переходит к следующему шагу, если нет – выдает ошибку. На основе данного алгоритма составлена блок-схема работы программы (см. приложение 2), по которой разработан код программы (см. приложение 3). Подключение модулей к отладочной плате проводилось через расширительную плату с предварительным впаиванием входов/выходов в соответствии со схемой подключения (см. приложение 4).

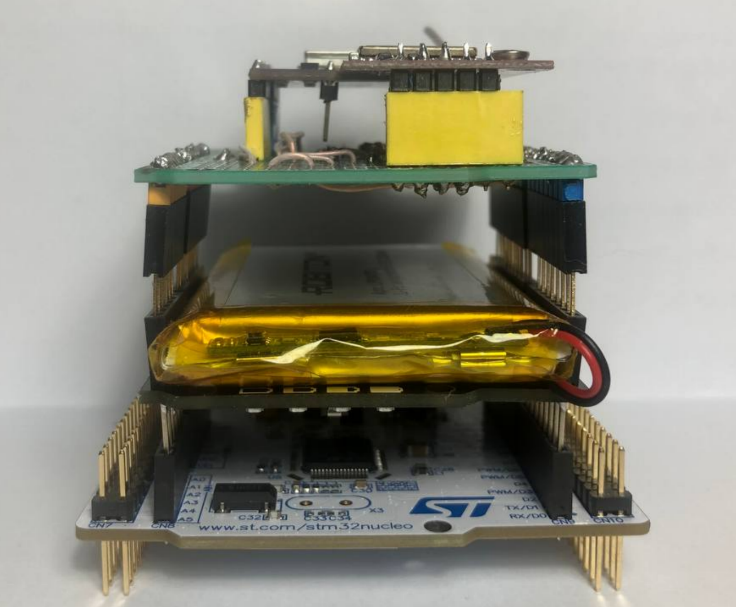
* 1. **Программные модули**

Для программы были использованы сторонние открытые библиотеки.

* **RH\_RF95.h** – библиотека, необходимая для комфортного взаимодействия с LoRa модулями SX1278 и SX1276;
* **Adafruit\_SHT31.h** – библиотека от компании Adafruit, упрощающая взаимодействие с метеодатчиком SHT31;
* **SPI.h** – библиотека, упрощающая взаимодействие по интерфейсу SPI;
* **Wire.h** – библиотека, упрощающая взаимодействие по интерфейсу I2C.
* **TinyGPS++** - библиотека, упрощающая работу с GPS-модулем.
* **SoftwareSerial –** библиотека для работы с интерфейсом UART.

Все вышеперечисленные библиотеки представляют из себя отдельные программные модули, размещенные в открытом доступе с открытым исходным кодом. Библиотеки без указания ссылок на репозитории являются встроенными в среду разработки ArduinoIDE.

* 1. **Результат разработки сенсорного узла**

После программирования отладочной платы и размещения модулей на расширительной плате путем распайки контактов все компоненты собираются вместе (см. рис. 7).

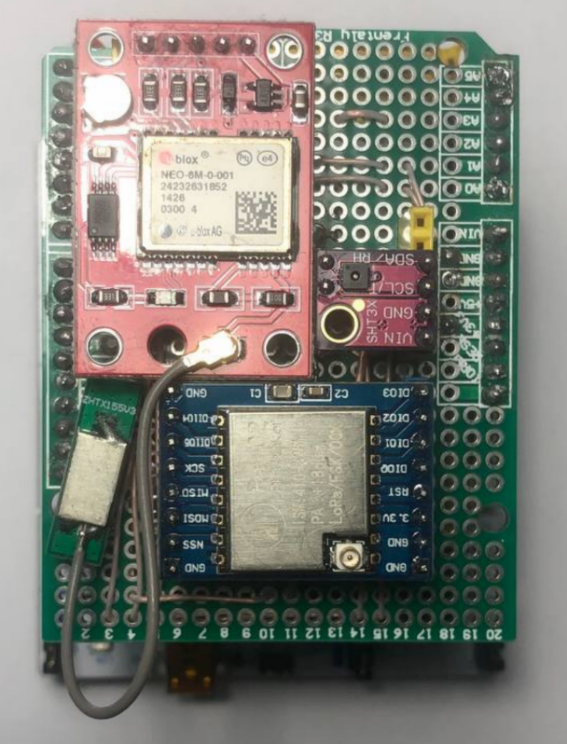


Рисунок 7 – Собранный сенсорный узел вид сверху и вид сзади

* 1. **Расчет себестоимости сенсорного узла**

Сведем цены на использованные при разработке модули в таблицу для наглядности.

**Таблица 1**

**Себестоимость компонентов сенсорного узла**

|  |  |
| --- | --- |
| **Компонент** | **Цена, руб.** |
| Nucleo F446RE | 5 900 |
| Battery Shield | 1 431 |
| Макетная плата | 168 |
| Ai-Thinker RA-02 | 750 |
| U-blox NEO-6M | 650 |
| SHT31 | 169 |
| Провода, коннекторы и прочее | 150 |
| **Итог** | **9 224** |

Стоимость разработки можно сильно удешевить, если выбрать в качестве отладочной платы Arduino UNO, стоимость которой до 500 руб., тогда получится сэкономить около 5 400 руб.

1. **Программирование приемного шлюза с интерфейсами LoRa и Wi-Fi/4G**
   1. **Выбор приемного шлюза**

В качестве приемного шлюза выбран готовый вариант Dragino LG-01 (см. рис. 8). Данный шлюз уже имелся в наличии и удовлетворял тем требованиям, что перед ним стояли: прием сигнала с сенсорного модуля и передача в облако. Сам шлюз выполнен в форм-факторе Wi-Fi маршрутизатора c питанием от сети 220 В. Интерфейс взаимодействия с пользователем для настройки представляет собой подключение к создаваемой шлюзом точки доступа Wi-Fi и взаимодействие через страницу в любом доступном браузере.



Рисунок 8 – Приемный шлюз LoRa / 4G Dragino LG-01

* 1. **Логика и алгоритм**

Приемный шлюз принимает пакет данных от сенсорного узла по протоколу LoRa. Затем устанавливает соединение с сервером облачного сервиса через MQTT протокол через Интернет. Распаковывает пакет и передает снятые параметры по-отдельности в «топики», предназначенные для каждого из параметров. На основе описанной выше логики работы шлюза разработан код программы (см. приложение 6) и блок-схема алгоритма работы программы (см. приложение 7).

Общая структурная схема всей системы состоит из объединенных структурных схем сенсорного узла и шлюза, а также включает «облако» и пользователя (см. рис. 28).

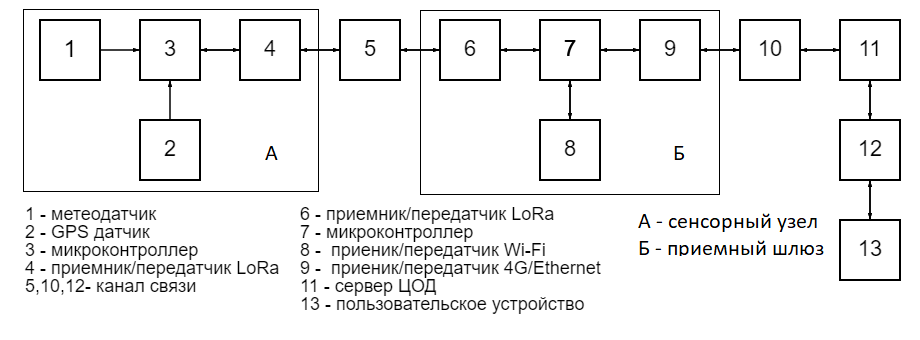


Рисунок 9 – Обобщенная структурная схема системы

* 1. **Программные модули**

Для создания программы шлюза LG-01 использовались следующие библиотеки:

* **RH\_RF95.h** – библиотека, необходимая для комфортного взаимодействия с LoRa модулями;
* **SPI.h** – библиотека, упрощающая взаимодействие по интерфейсу SPI;
* **Console.h** – библиотека для взаимодействия через консоль.

1. **Тестирование собранной системы**

Тестирование полученной системы проводилось в 3 этапа. **Первый этап** представлял собой тестирование передачи информации с сенсорного узла. Для этого, как говорилось выше, был использован отладочный модуль с приемником LoRa, подключенный по USB к ПК. Данные, принимаемые отладочным модулем, отображались в последовательном порте через терминал (см. рис. 10).

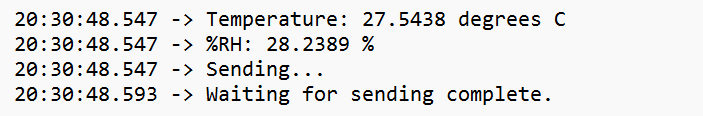


Рисунок 10 – Тестирование сенсорного модуля

**Второй этап** заключался в подключении и тестировании приема данных шлюзом. Для этого шлюх был подключен к ПК через сетевой порт и выводил информацию в терминал (см. рис. 11).

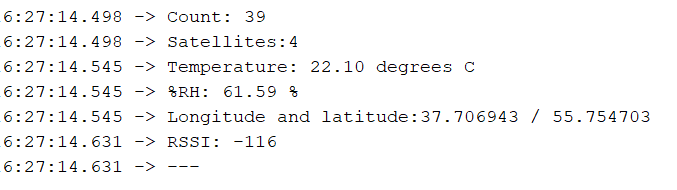


Рисунок 11 – Результаты тестирования GPS-модуля

Измеренное положение показало нахождение сенсорного узла возле Красноказарменной 14Ак1 (см. рис. 31). Погрешность составила около 114 метров. Для применения полученной системы внутри объектов необходимо выбирать более дорогой и точный GPS-модуль. Но на уровне макета полученные данные удовлетворительны.

**Третий этап** заключался в передачи полученных данный в «облако» уже с использованием шлюза LG-01 (см. рис. 32)**.**

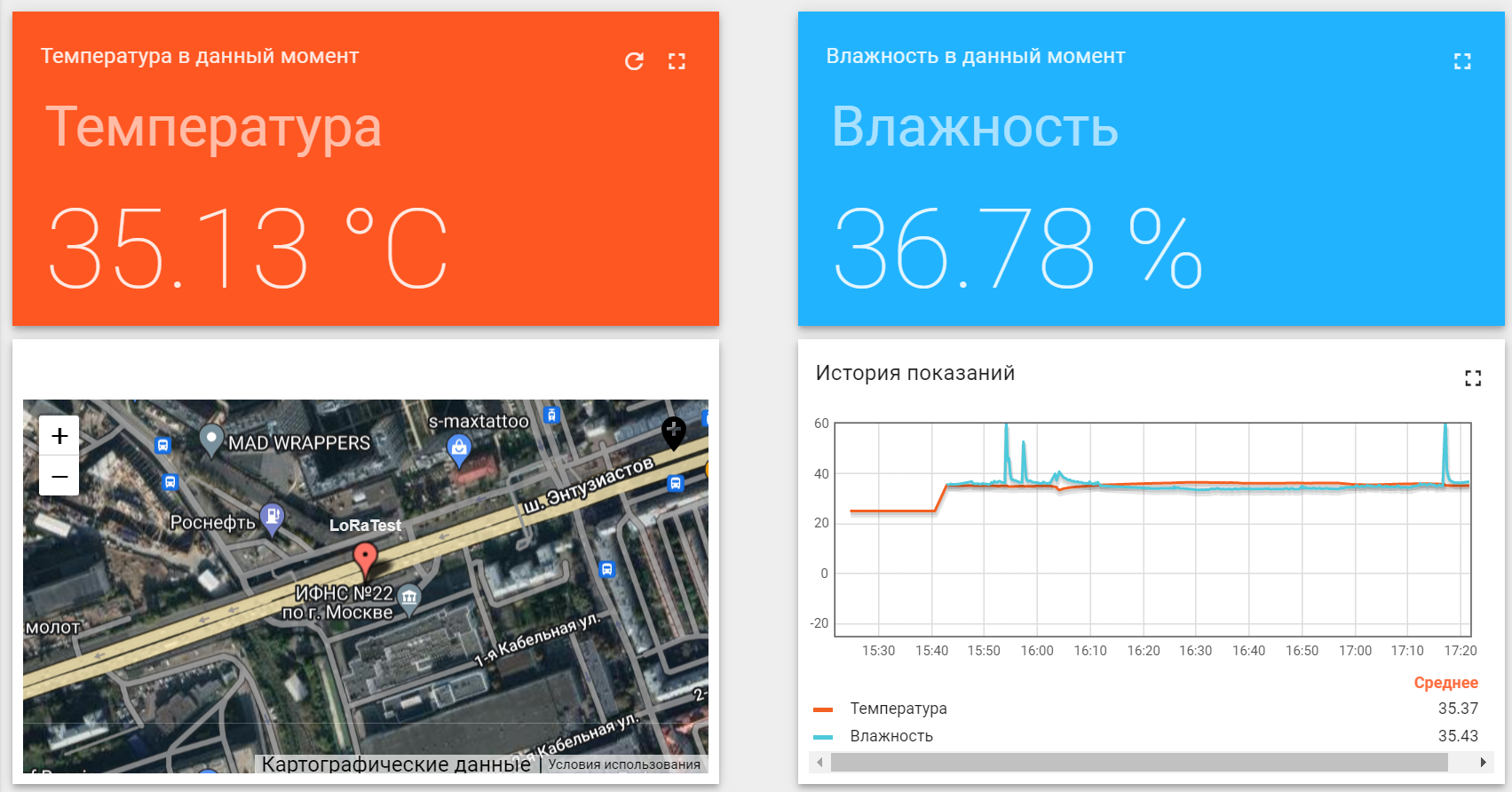
****

Рисунок 12 – Результаты тестирования «облачной» платформы ThingsBoard

Для удобства тестирования была выбрана частота обновления 200 мс. В реальных условиях достаточно 2-3 раз в час, что не изменит общей картины о состоянии окружающей среды в месте расположения системы, но позволит экономить заряд аккумулятора.

# *Приложение 1*

**Подключение внешнего аккумулятора Battery Shield**

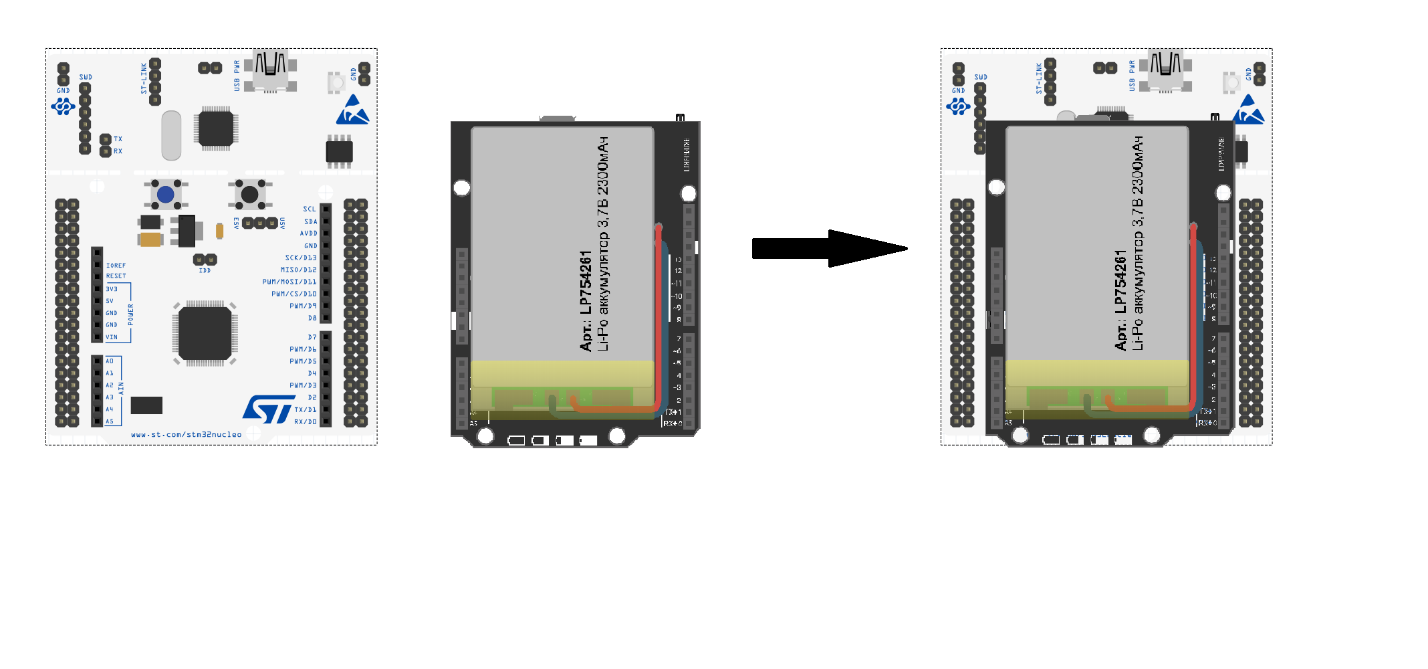
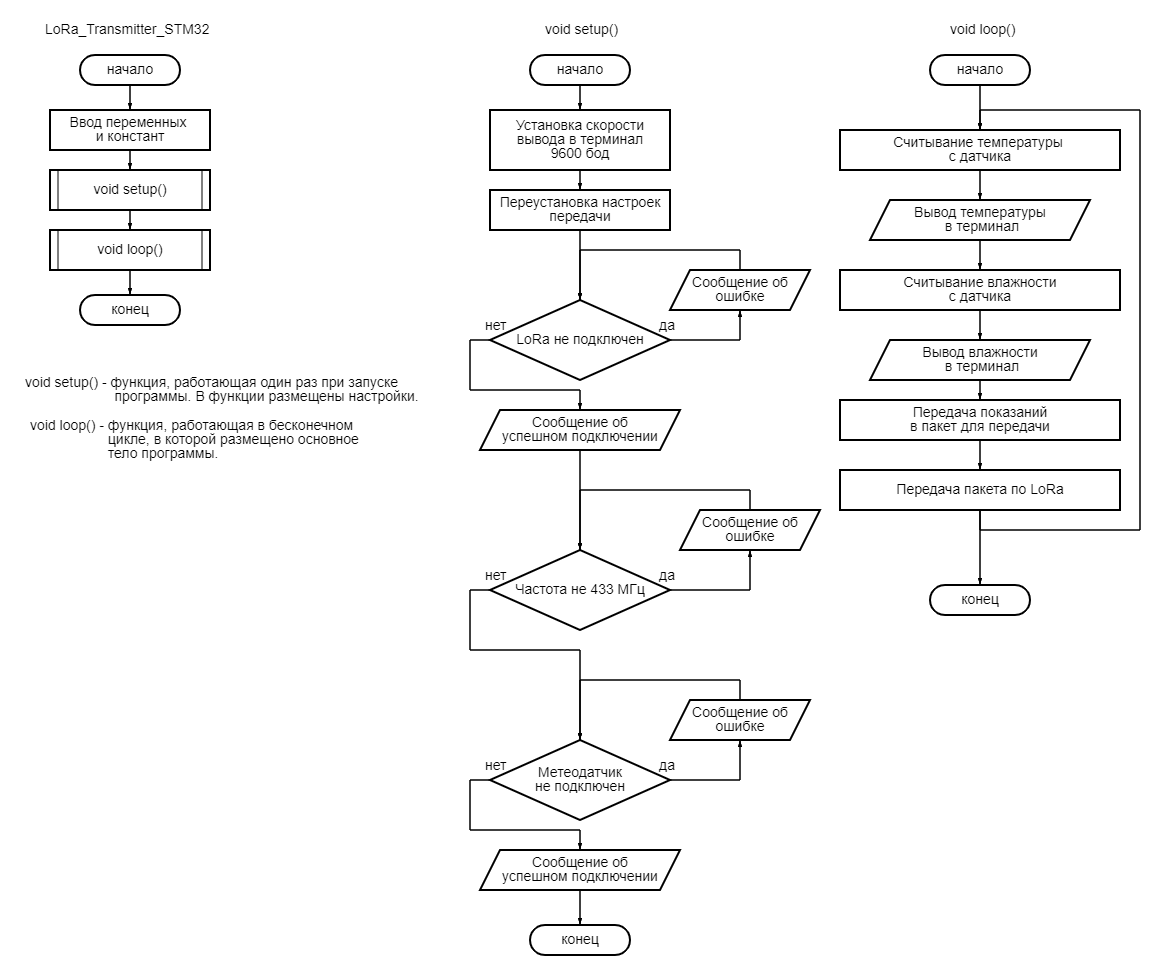


Рисунок П1 – Подключение Battery Shield

# *Приложение 2*

**Блок-схема алгоритма работы программы сенсорного узла**

****

# *Приложение 3*

**Код программы сенсорного узла**

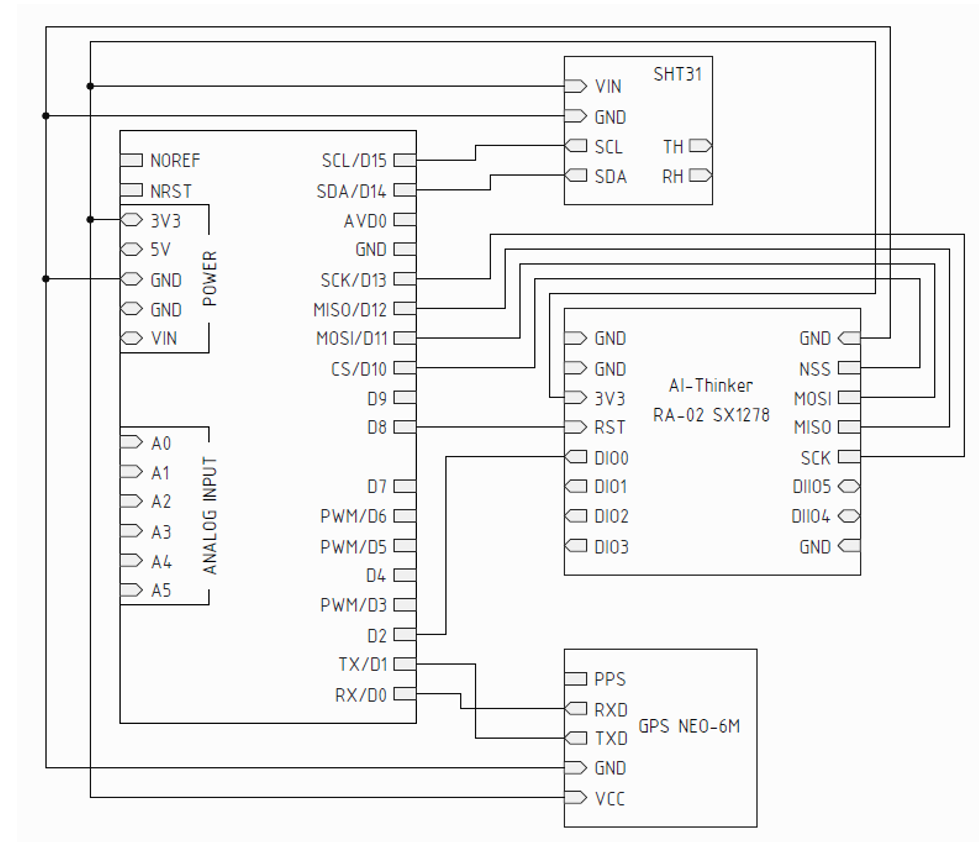
2. #include <TinyGPS++.h> // include GPS library for NEO-6M
3. #include <Adafruit\_SHT31.h> //include library for SHT31
4. #include <SPI.h> // include library for SPI interface
5. **#include <RH\_RF95.h> // include library for SX1276 LoRa**
6. #include <Wire.h> // include library for I2C interface
7. #include <stdint.h>
8. #include <SoftwareSerial.h>
10. **#define RFM95\_RST 8 // define number of reset pin**
11. #define RF95\_FREQ 433.0 // define channel freq const = 433 MHz
13. static const int RXPin = 3, TXPin = 4;
14. static const uint32\_t GPSBaud = 9600;
16. *//Battery\_Shield battery;*
17. TinyGPSPlus gps;
18. RH\_RF95 LoRa;
19. Adafruit\_SHT31 sht31;
20. **bool status;**
22. SoftwareSerial ss(RXPin,TXPin);
24. *//--------Define stucture of datapacket ------------*
25. **typedef struct**
26. {
27. float temperature; *// SHT31 get temp and hum*
28. float humidity;
29. float longitude;
30. **float latitude;**
31. *// float charge;*
32. int satellit;
33. } send\_sht31\_data;
35. **send\_sht31\_data LoRaPack;**
36. *//--------------------------------------------*
38. void setup()
39. {
40. **pinMode(RFM95\_RST, OUTPUT);**
41. digitalWrite(RFM95\_RST, HIGH);
42. *//-------- Serial Out Rate Config--------------*
43. while (!Serial);
44. Serial.begin(9600);
45. **ss.begin(9600);**
47. *//--------------------------------------------*
48. Serial.println("Service info:");
49. Serial.println("");
50. **Serial.println("Transmit is started!");**
52. *//----------Reset transmission config------------------*
53. digitalWrite(RFM95\_RST, LOW);
54. smartDelay(0);
55. **digitalWrite(RFM95\_RST, HIGH);**
56. smartDelay(0);
57. *//-----------------------------------------------------*
59. *//----------SX1278 initialization check---------------*
60. **while (!LoRa.init()) {**
61. Serial.println("LoRa status: Not detected");
62. while (1);
63. }
64. Serial.println("LoRa satus: OK");
65. ***//----------------------------------------------------***
66. *//----------------Set freq check-----------------------*
67. if (!LoRa.setFrequency(RF95\_FREQ)) {
68. Serial.println("Set Frequency failed.");
69. while (1);
70. **}**
71. Serial.print("Set Freq to: "); Serial.println(RF95\_FREQ);
72. LoRa.setTxPower(5, **false**);
73. *//---------------------------------------------------------*
75. ***//----------------------SHT31 sensor init check----------------------------------***
76. status = sht31.begin();
77. if (!status) {
78. Serial.println("SHT31: Not detected");
79. } else {
80. **Serial.println("SHT31: OK");**
81. }
82. }
83. *//------------------------------------------------------------------------------*
84. void loop()
85. **{**
86. Serial.println("----------------------------------");
88. *//--------------Get SHT31 data--------------------*
89. float temperature = sht31.readTemperature();
90. **Serial.print("Temperature: ");**
91. Serial.print(temperature, 2);
92. Serial.println(" degrees C");
94. float humidity = sht31.readHumidity();
95. **Serial.print("%RH: ");**
96. Serial.print(humidity, 2);
97. Serial.println(" %");
98. int satellit = gps.satellites.value();
99. float longitude = gps.location.lng();
100. **float latitude = gps.location.lat();**
101. Serial.print("Latitude / longitude:");
102. Serial.print(latitude, 6);
103. Serial.print(" / ");
104. Serial.println(longitude, 6);
105. **Serial.println("");**

108. *//---------------------------------------------------------*
109. */\*Serial.println("GPS debugg information:");*
110. ***Serial.print("CharsProcessed:"); Serial.println(gps.charsProcessed());\*/***
111. if (millis() > 5000 & gps.charsProcessed() < 10){
112. Serial.println(F("GPS was not detected: check connection"));
113. Serial.println("GPS stream dump:");
114. while (**true**) *// infinite loop*
115. **if (ss.available() > 0) *// any data coming in?***
116. Serial.write(ss.read());}
118. */\* Serial.print("Satellites:"); Serial.println(gps.satellites.value());*
119. *Serial.print("Sentences that failed checksum=");*
120. ***Serial.println(gps.failedChecksum());***
122. *// Testing overflow in SoftwareSerial is sometimes useful too.*
123. *Serial.print("Soft Serial device overflowed? ");*
124. *Serial.println(ss.overflow() ? "YES!" : "No");\*/*


128. *//----------------DataPack-------------------*
129. LoRaPack.temperature = temperature;
130. **LoRaPack.humidity = humidity;**
131. LoRaPack.longitude = longitude;
132. LoRaPack.latitude = latitude;
133. *// LoRaPack.satellit = satellit;*
135. **Serial.println("Sending..."); smartDelay(100);**
136. LoRa.send((uint8\_t \*)&LoRaPack, sizeof(LoRaPack));
138. Serial.println("Waiting for sending complete..."); smartDelay(100);
139. LoRa.waitPacketSent();
140. **Serial.println("----------------------------------");**
141. Serial.println("");
143. smartDelay(10000);
144. }
145. **static void smartDelay(unsigned long ms) *//Custom delay that guarantees the availability of GPS data***
146. {
147. unsigned long start = millis();
148. do
149. {
150. **while (ss.available())**
151. gps.encode(ss.read());
152. } while (millis() - start < ms);
153. }

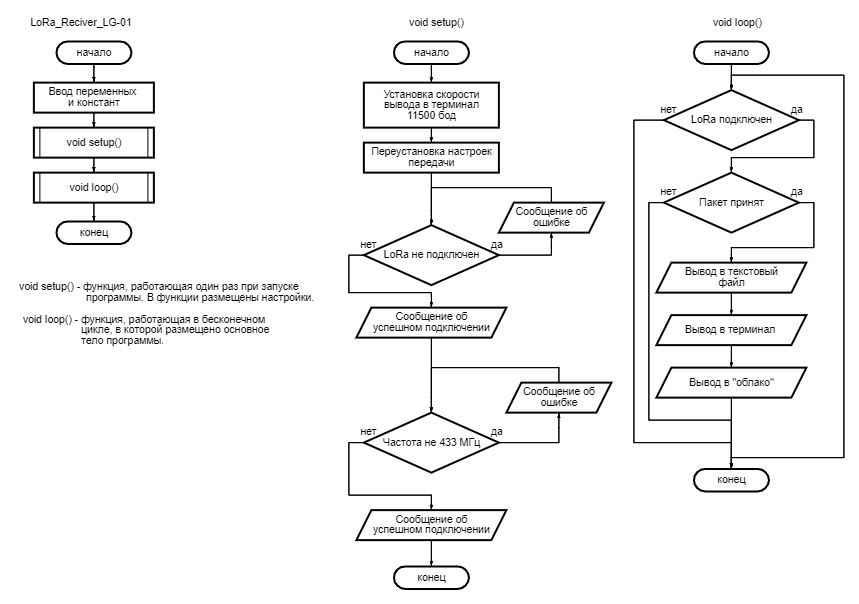
# *Приложение 4*

**Схема электрическая принципиальная сенсорного узла**



# *Приложение 5*

**Блок-схема алгоритма работы программа приемного шлюза**

****

# *Приложение 6*

**Код программы приемного шлюза**

1. *// Arduino9x\_RX*
2. *// -\*- mode: C++ -\*-*
3. *// Example sketch showing how to create a simple messaging client (receiver)*
4. *// with the RH\_RF95 class. RH\_RF95 class does not provide for addressing or*
5. ***// reliability, so you should only use RH\_RF95 if you do not need the higher***
6. *// level messaging abilities.*
7. *// It is designed to work with the other example Arduino9x\_TX*
9. #include <Console.h>
10. **#include <SPI.h>**
11. #include <RH\_RF95.h>
13. *//#define RFM95\_CS 10*
14. *//#define RFM95\_RST 9*
15. ***//#define RFM95\_INT 2***
17. *// Change to 434.0 or other frequency, must match RX's freq!*
18. #define RF95\_FREQ 433.0
20. ***// Singleton instance of the radio driver***
21. RH\_RF95 rf95;*//(RFM95\_CS, RFM95\_INT);*
23. *// Blinky on receipt*
24. #define LED A2


28. *// data struct*
29. typedef struct
30. **{**
31. float temperature;
32. float humidity;
33. float lng;
34. float lat;
35. **} s\_bme\_data;**
37. s\_bme\_data radiopacket;
38. uint8\_t datalen = sizeof(radiopacket);
39. int16\_t last\_rssi;
40. **bool recv\_state = false;**
41. uint16\_t counter = 0;
43. *//#define USE\_LCD*
44. #define USE\_FILE
45. **#define USE\_THINGSBOARD**
47. #ifdef USE\_LCD
48. #include "U8glib.h"
49. U8GLIB\_SSD1306\_128X64 u8g(U8G\_I2C\_OPT\_NONE); *// I2C / TWI*
51. char \*status\_text, temp\_text[25], s\_tmp[15];
52. #endif
54. #ifdef USE\_FILE
55. **#include <FileIO.h>**
56. const char\* data\_filename = "/mnt/sda1/data/datalog.csv";
58. *// This function return a string with the time stamp*
59. *// LG01 will call the Linux "date" command and get the time stamp*
60. ***/\* from Logger\_USB\_Flash \*/***
61. String getTimeStamp() {
62. String result;
63. Process time;
64. time.begin("date");
65. **time.addParameter("+%s");**
66. time.run(); *// run the command*
68. *// read the output of the command*
69. while(time.available()>0) {
70. **char c = time.read();**
71. if(c != '**\n**')
72. result += c;
73. }
75. **return result;**
76. }
78. */\*Function for data writing\*/*
79. bool write\_data(String dataString, const char\* file\_name)
80. **{**
81. File dataFile = FileSystem.open(file\_name, FILE\_APPEND);
83. *// if the file is available, write to it:*
84. if (dataFile) {
85. **dataFile.println(dataString);**
86. dataFile.close();
87. return **true**;
88. } else {
89. return **false**;
90. **}**
91. }
93. #endif
95. **#ifdef USE\_THINGSBOARD**
96. #include <Process.h>
98. void uploadData(String dataString) {*//Upload Data to ThingsBoard*
99. Console.print("**\t\t**Call Linux Command to send ");
100. **Console.print("JSON data: ");**
101. Console.println(dataString);
103. Process p;
104. p.begin("send\_data");
105. **p.addParameter(dataString);**
106. p.run(); *// Run the process and wait for its termination*
107. Console.print("ended with " + String(p.exitValue()));
109. Console.print("**\t\t**Feedback from Linux: ");
110. ***// If there's output from Linux,***
111. *// send it out the Console:*
112. while (p.available()>0)
113. {
114. char c = p.read();
115. **Console.write(c);**
116. }
117. Console.println("");
118. Console.println("**\t\t**Call Finished");
119. }
121. #endif
123. void setup()
124. {
125. **pinMode(LED, OUTPUT);**
126. *// pinMode(RFM95\_RST, OUTPUT);*
127. *// digitalWrite(RFM95\_RST, HIGH);*
129. Bridge.begin(115200);
130. **Console.begin();**
131. *// while (!Console) ; // Wait for console port to be available*
132. delay(100);
134. #ifdef USE\_FILE
135. **FileSystem.begin();**
136. #endif
137. Console.println("Arduino LoRa RX Test!");
139. *// manual reset*
140. ***/\* digitalWrite(RFM95\_RST, LOW);***
141. *delay(10);*
142. *digitalWrite(RFM95\_RST, HIGH);*
143. *delay(10);*
144. *\*/*
145. **while (!rf95.init()) {**
146. Console.println("LoRa radio init failed");
147. while (1);
148. }
149. Console.println("LoRa radio init OK!");
151. *// Defaults after init are 434.0MHz, modulation GFSK\_Rb250Fd250, +13dbM*
152. if (!rf95.setFrequency(RF95\_FREQ)) {
153. Console.println("setFrequency failed");
154. while (1);
155. **}**
156. Console.print("Set Freq to: "); Console.println(RF95\_FREQ);
158. *// Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on*
160. ***// The default transmitter power is 13dBm, using PA\_BOOST.***
161. *// If you are using RFM95/96/97/98 modules which uses the PA\_BOOST transmitter pin, then*
162. *// you can set transmitter powers from 5 to 23 dBm:*
163. rf95.setTxPower(23, **false**);
165. **#ifdef USE\_LCD**
166. u8g.setFont(u8g\_font\_5x7);
167. u8g.setFontPosTop();
168. status\_text = ":) Receiver started! :)";
169. #endif
170. **}**
172. void loop()
173. {
174. #ifdef USE\_LCD
175. **status\_text = "... Receiving ...";**
176. #endif
177. *// recv\_state = false;*
179. if (rf95.available())
180. **{**
181. *// Should be a message for us now*
182. *//uint8\_t buf[RH\_RF95\_MAX\_MESSAGE\_LEN];*
183. *//uint8\_t len = sizeof(buf);*
185. **if (rf95.recv((uint8\_t\*)&radiopacket, &datalen))**
186. {
187. #ifdef USE\_LCD
188. status\_text = ":) Receiver got data :) ";
189. #endif
190. **recv\_state = true;**
192. digitalWrite(LED, HIGH);
193. *//RH\_RF95::printBuffer("Received: ", radiopacket, datalen);*
194. Console.print("Count: ");
195. **Console.println(counter);**
197. Console.print("Temperature: ");
198. Console.print(radiopacket.temperature, 2);
199. Console.println(" degrees C");
201. Console.print("%RH: ");
202. Console.print(radiopacket.humidity, 2);
203. Console.println(" %");
205. **Console.print("GPS lng: ");**
206. Console.println(radiopacket.lng, 2);
208. Console.print("GPS lat: ");
209. Console.println(radiopacket.lat, 2);
211. *//Console.println((char\*)radiopacket);*
212. Console.print("RSSI: ");
213. last\_rssi = rf95.lastRssi();
214. Console.println(last\_rssi, DEC);
216. *// Send a reply*
217. */\*uint8\_t data[] = "And hello back to you";*
218. *rf95.send(data, sizeof(data));*
219. *rf95.waitPacketSent();*
220. ***Console.println("Sent a reply");\*/***
221. digitalWrite(LED, LOW);
223. #ifdef USE\_FILE
224. String timestamp = getTimeStamp();
226. Console.println("- writing file -");
227. String dataString;
228. dataString += "{";
229. dataString += "**\"**temperature**\"**:";
230. **dataString += String(radiopacket.temperature);**
231. dataString += ", ";
232. dataString += "**\"**humidity**\"**:";
233. dataString += String(radiopacket.humidity);
234. dataString += ", ";
235. **dataString += "\"lng\":";**
236. dataString += String(radiopacket.lng);
237. dataString += ", ";
238. dataString += "**\"**lat**\"**:";
239. dataString += String(radiopacket.lng);
240. **dataString += "}";**
242. write\_data(dataString, data\_filename);
243. #endif
245. **#ifdef USE\_THINGSBOARD**
246. String space = "%20";
247. Console.println("- uploading to ThingsBoard -");
248. uploadData(dataString);
249. #endif
251. Console.println("---");
253. counter++;
254. }
255. **else**
256. {
257. Console.println("Receive failed");
258. recv\_state = **false**;
259. }
260. **}**
262. #ifdef USE\_LCD
263. *// U8GLIB picture loop*
264. u8g.firstPage();
265. **do {**
266. u8g.drawStr(0, 0, status\_text);
267. if (recv\_state) {
268. sprintf(temp\_text, "N: %8d RSSI: %6s", counter, dtostrf(last\_rssi, 5, 0, s\_tmp));
269. u8g.drawStr(0, 15, temp\_text);
271. sprintf(temp\_text, "Temperature: %10s C", dtostrf(radiopacket.temperature, 5, 2, s\_tmp));
272. u8g.drawStr(0, 25, temp\_text);
274. sprintf(temp\_text, "Humidity: %13s %%", dtostrf(radiopacket.humidity, 8, 2, s\_tmp));
275. **u8g.drawStr(0, 55, temp\_text);**
276. }
277. } while( u8g.nextPage() );
278. #endif
280. **}**