Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
«Высшая школа электроники и компьютерных наук»
Кафедра «Инфокоммуникационные технологии»

ДОПУСТИТЬ К ЗАЩИТЕ

Рецензент

Руководитель направления

_____. _____ С.Н. Даровских

“____” _____ 2019 г. “____”_____ 2019 г.

«ПРОГРАММНО-АППАРАТНЫЙ ЛАБОРАТОРНЫЙ КОМПЛЕКС ДЛЯ ИЗУЧЕНИЯ ТЕХНОЛОГИИ ПЕРЕДАЧИ ДАННЫХ В ИНТЕРНЕТЕ ВЕЩЕЙ»

Направление **11.04.02 «Инфокоммуникационные технологии и системы связи»**

магистерская программа **«Системы мобильной связи»**

ЮУрГУ – М 11.04.02.19.158.00 ПЗ

Научный руководитель:
Николаев А.Н.
“____”_____2019 г.

Магистрант
студент группы КЭ-223
Демина Е.Ю.
“____”_____2019 г.

Нормоконтролер
Спицына В.Д._____
“____”_____2019 г.

Челябинск
2019

РЕФЕРАТ

Демина Е.Ю. Программно-аппаратный лабораторный комплекс для изучения технологии передачи данных в Интернете Вещей –
Челябинск: ЮУрГУ, КЭ, 2019, 85 с., 5 табл., 27 ил., 13 источников, 1 прил., 6 плакатов ф. А1.

В дипломной работе представлен процесс создания универсального конечного устройства, предназначенного для использования в учебном процессе при изучении технологии передачи данных в Интернете Вещей.

Универсальное конечное устройство совместимо со стандартом LoRaWAN и уникальным протоколом Unwired Devices, предусмотрена связь с пользователем через последовательный интерфейс UART. Листинг программы на языке СИ приводится в приложении А.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время, одним из ключевых направлений развития сетей связи стала концепция Интернета вещей [1, 2, 3]. Согласно подавляющему большинству прогнозов, “Интернет вещей” продолжит бурно развиваться, и к 2020 году количество устройств в этом сегменте достигнет отметки в несколько миллиардов. Мир стоит на пороге четвертой промышленной революции. Так, согласно обзору, подготовленному к форуму в Давосе, Интернет вещей входит в топ-5 технологических драйверов четвертой промышленной [4, p. 7]. По данным всемирного исследования PwC Digital IQ за 2017 год, IoT занимает первое место среди восьми прорывных технологий, способных изменить бизнес-модели компаний или целых индустрий [5].

Таким образом, интернет вещей – очень молодая и перспективная индустрия и не удивительно, что в учебных программах университетов все чаще появляется одноименная дисциплина. Широкий спектр отраслей, где востребован IoT, а также разнообразие устройств и датчиков, подключаемых к сети, увеличивают спрос в специалистах, разбирающихся в таких системах и способных с ними работать. Увеличение популярности IoT приведет к новому буму на рынке труда. Поэтому специалисты, понимающие как создавать и применять решения на основе IoT, имеют прекрасные перспективы профессионального роста.

В 2018 году на базе высшей школы электроники и компьютерных наук Южно-Уральского государственного университета открылась “IoT Академия Sumsung”,цель которой – изучение интернета вещей [6]. Наряду с ЮУрГУ подобные лаборатории появились еще в десяти ВУЗах страны. Для реализации данной программы компания Sumsung предоставила учебные методические материалы, а также расширенный комплект оборудования, который представляет собой учебный набор Unwired Kit российской компании Unwired Devices для оснащения IoT –лаборатории. Оборудование включает в себя радиомодули, взаимодействующие по протоколу LoRa, микрокомпьютеры Artik10 всевозможные датчики и другие высокотехничные устройства.

Процесс изучения технологии передачи данных в Интернете Вещей построен следующим образом:

- В течение первого семестра студенты изучают учебные кейсы - задачи, построенных на индустриальных примерах по внедрению технологий Интернета вещей.
- Завершающей частью обучения является выполнение индивидуального проекта на базе технологий Интернета вещей.

Для выполнения индивидуальных проектов, студенты должны иметь в своем распоряжении необходимые технические средства и компоненты, в том числе радиомодемы.

Основные проблемы, с которыми столкнулись при проведении курса:

- Задержка поставки и нехватка оборудования, отсюда имеющиеся радиомодемы Unwired Devices студентам отдавать нельзя, т.к. они нужны для выполнения кейсов;
- Покупка модемов не решит проблему, так как работают они в соответствии со спецификацией LoRaWAN и по протоколу обмена не совместимы с Unwired Device , поскольку у них свой уникальный протокол MAC уровня. Отсутствие исходников ПО покупных модемов исключает возможность студентам корректировать работу устройства.

Разработка универсального радиомодема, совместимого со стандартом LoRaWAN и уникальным протоколом Unwired Device позволила бы решить вышеуказанные проблемы. Универсальный радиомодем можно будет давать студентам для проектной работы. При этом студент сможет работать как в стандартной сети LoRaWAN, развернутыми на территории Челябинска (например, Интерсвязь), так и с оборудованием Unwired Device. Разрабатываемый радиомодем может быть основой набора (кита) для проектной работы студентов. Открытое программное обеспечение позволит студентам корректировать работу устройства и дооснастить его разными датчиками.

Связь с пользователем по последовательному интерфейсу UART позволит изучить работу протокола.

Целью выпускной квалификационной работы является разработка универсального радиомодема, предназначенного для использования в учебном процессе при изучении технологии передачи данных в Интернете Вещей.

Для достижения поставленной цели решались следующие задачи:

- исследование протоколов и выявления особенностей протокола MAC уровня от Unwired Device;
- определения места разрабатываемого устройства в сети LoRaWAN;
- исследование и аппаратная реализация устройства;
- разработка ПО для микроконтроллера;
- разработка индивидуального проекта в рамках курса “IOT Академия Samsung”, с целью показать применение разрабатываемого устройства в реальном проекте.

Структура пояснительной записки в целом совпадает с поставленными задачами. Первая глава посвящена исследованию спецификации LoRaWAN и протокола, использованного для устройств фирмы Unwired Range. Во второй главе определяется место разрабатываемого устройства в сети LoRaWAN, исследуется аппаратное обеспечения такого устройства. Третья глава посвящена рассмотрению структуры программного обеспечения устройства и алгоритмов работы устройства. В четвертой главе приведена инструкция по работе с радиомодулем, описана команды для работы с устройством через интерфейс UART. В пятой главе приведено описание индивидуального проекта, выполненного в рамках курса “IOT Академия Samsung”, в котором использовался разработанный программно-аппаратный комплекс.

1 Спецификация LoRaWAN и вариант протокола MAC-уровня от UNWIRED DEVICE

Low Power Wide Area Networks (LPWAN) – энергоэффективные сети большого радиуса действия, создавались непосредственно для интернета вещей и являются лучшим решением для построения сети.

LoRaWAN – стандарт протокола LPWAN, работающий в технологической среде LoRa. [7,8] Технологии LoRaWAN является открытым стандартом работающем в нелицензионном диапазоне частот. Чипы LoRaWAN для конечных устройств присутствуют в свободной продаже, документация на них открыта, делать устройства на них могут все желающие. Для образовательных целей – LoRaWAN является лучшим решением.

Согласно спецификации, LoRaWAN является протоколом канального уровня, который должен обеспечить:

- передачу блоков данных между конечным устройством и сетевым сервером;
- шифрование (на уровне сети) полезной нагрузки, передаваемой между конечным устройством и приложением;
- управление выделением окон передачи данных в линии «вниз»;
- адаптацию скорости передачи данных.

Устройства фирмы Unwired Range имеют свои уникальный протокол MAC уровня. Для исследования на степень соответствия устройства спецификации LoRaWAN необходимо рассмотреть формат MAC пакета, время запуска и длительность временного окна приема, безопасность в сетях, а именно используемые ключи и алгоритмы шифрования.

Данная глава посвящена исследованию спецификации LoRaWAN и протокола, использованного для устройств фирмы Unwired Range.

1.1 MAC уровень протокола LoRaWAN

1.1.1 Формат сообщения MAC уровня

На рисунке 1 приведен формат MAC пакета.

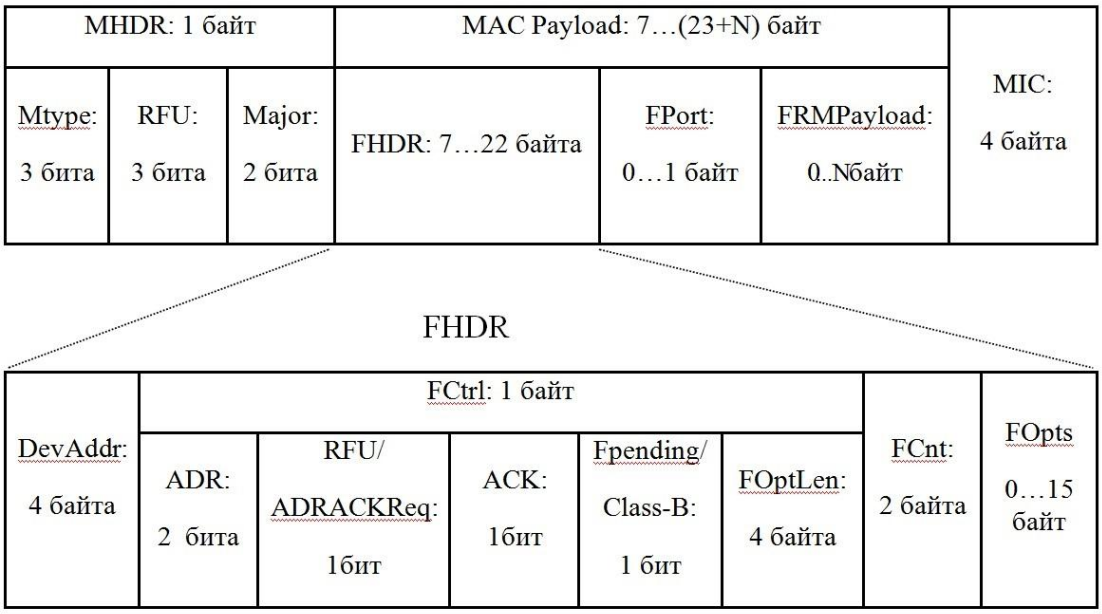

Рисунок 1 – Формат сообщений MAC уровня

Сообщение содержит в себе следующие поля [1]:

а) MHDR – заголовок пакета MAC уровня.

Заголовок MAC определяет тип сообщения (MType) и в соответствии с какой основной версией (Major) формата кадра спецификации уровня LoRaWAN кадр был закодирован.

LoRaWAN различает 8 различных типов сообщений MAC. При передаче данных наиболее важные – сообщения требующее подтверждения получения и сообщения без подтверждения.

б) MACPayload – фрейм данных.

Фрейм данных (MACPayload) содержит заголовк фрейма (FHDR), за которым следует номера порта фрейма (FPort) и поле полезной нагрузки фрейма (FRMPayload).

1) Заголовок фрейма (FHDR) содержит короткий адрес устройства конечного устройства (DevAddr), поле управления информацией фрейма (FCtrl), 2-

октетный счетчик фреймов (FCnt) и до 15 байт опций фрейма (FOpts), используемых для передачи команд MAC.

Поле управления информацией фрейма (FCtrl) состоит из:

a) ADR – флаг активации режима адаптации скорости;

b) ADRACKReq – флаг запроса конечным устройством подтверждения факта получения сетью сообщений от данного устройства;

c) ACK – флаг, индицирующий получение одной стороной (сетью или конечным устройством) сообщения от другой стороны;

d) FPending (только в DL канале) – флаг, индицирующий наличие запроса со стороны сети на необходимость передачи конечному устройству дополнительных данных сверх объема, который может быть передан в рамках окна передачи.

e) CLASS-B (только в UL канале) – флаг, индицирующий, что конечное устройство переключено в режим "класс B".

f) FOptLen – актуальный размер поля опций FOpt заголовка MAC уровня.

Конечное устройство и сетевой сервер после процедуры активации по воздуху (OTA – over-the-air) (join accept) инициализируют два счетчика – счетчик колличества переданных фреймов и счетчик кол-ва принятых фреймов (FCntUp/FCntDown). Спецификацией допускается использование 16-ти и 32-х битных счетчиков. При передаче сообщения встречной стороне конечное устройство / сетевой сервер указывают номер передаваемого фрейма (в поле FCnt заголовка MAC уровня). При получении каждого нового сообщения принимающая сторона (конечное устройство / сетевой сервер) сравнивает поле FCnt со значением внутреннего счетчика принятых фреймов (FCntUp/FCntDown). Если разница превышает величину MAX_FCNT_GAP, принимается решение о значительном колличестве потерянных пакетов.

2) FPort – номер порта фрейма:

a) значение 0 означает, что поле полезной нагрузки фрейма (FRMPayload) содержит MAC команду ;
b) значения 1..223 определяются уровнем приложений (application specific);
c) значения 224-225 зарезервированы для будущего использования.

3) FRMPayload – полезная нагрузка фрейма. Содержимое поля FRMPayload шифруется стандарту AES либо на уровне приложения (с использованием ключа AppSKey), либо на уровне сетевого сервера (с использованием ключа NwkSKey). Оба ключа имеют длину 128бит.

4) MIC – код контроля целостности. Вычисляется по всем полям сообщения на основе алгоритма AES128 и секретного ключа NwkSKey.

1.1.2 Окна приема информации

Для решения различных задач и применений в сети LoRaWAN предусмотрено три класса конечных устройств:

А) Двунаправленные конечные устройства «класса А» (Bi-directional end-devices, Class A). После инициализации связи выделяются два временных окна, в течение которых ожидается ответ от сети. Конечные устройства «класса А» применяются в приложениях, где передача данных от сети возможна только как ответная реакция на получения данных от конечного устройства и требуется максимальное время работы от автономного источника питания.

Б) Двунаправленные конечные устройства «класса Б» (Bi-directional end-devices, Class B) в дополнение к функциям устройств «класса А», открывают дополнительные окна приема по расписанию. Для того, чтобы открыть окно приема, конечное устройство синхронизируется по специальным сигналам от шлюза (по маякам – Beacon). Это позволяет сети знать время, когда конечное устройство готово принимать данные.

В) Двунаправленные конечные устройства «класса С» с максимальным приемным окном (Bi-directional end-devices, Class C). Конечные устройства «класса

С» имеют почти непрерывно открытое окно приема. Приемное окно закрывается только на время передачи данных. Этот тип конечных устройств подходит для задач, когда необходимо получать большие объемы данных и не требуется длительная работа от автономного источника питания.

В дальнейшем будут рассматриваться только устройства класса А.

Для устройств класса "А" после завершения каждого сеанса передачи данных конечным устройством (в линии «вверх») открываются два коротких временных окна, в течении которых конечное устройство может принять данные от сети (в линии «вниз»), как показано на рисунке 2. При этом на протяжении длительности окон приема передача данных конечным оборудованием запрещена.

Для передачи данных шлюзом LoRa в первом временном окне (RX1) используются те же параметры передачи (включая номер частотного канала и скорость передачи данных), которые использовались для передачи данных конечным устройством.

Для передачи данных шлюзом LoRa во втором временном окне (RX2) используются предустановленные параметры передачи (включая номер частотного канала и скорость передачи данных).

Длительность временного окна предустанавливается и должна быть достаточной для приема преамбулы.

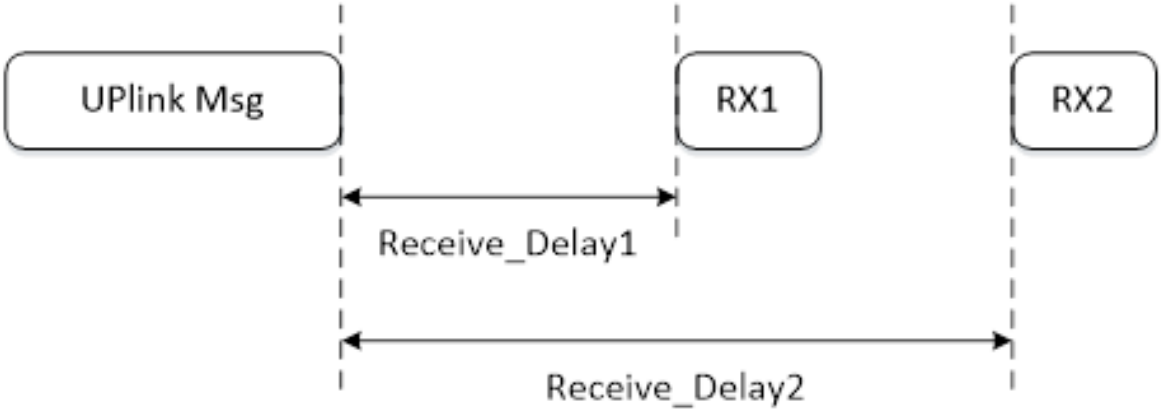

Рисунок 2 – Окна приема информации LoRaWAN

1.1.3 Адаптивная скорость передачи (Adaptive Data Rate – ADR)

В технологии LoRa предусмотрены механизмы адаптации скорости передачи данных конечных устройств с тем, чтобы оптимизировать загрузку сети и обеспечить каждому конечному устройству возможность работы на максимальных скоростях, обеспечивающих надлежащую помехоустойчивость в тех радиоусловиях, в которых данное устройство находится.

Адаптацию скорости передачи данных конечных устройств (End Node) выполняет сетевой сервер посредством соответствующих MAC команд. Решение о выборе той или иной скорости принимается на основании оценки качества принятого от конечного устройства сигнала.

1.1.4 Основные константы стека протоколов LoRaWAN

Основные константы стека протокола LoRaWAN:

- RECEIVE_DELAY1 (длительность временного окна приема RX1) – 1 сек.
- RECEIVE_DELAY2 (длительность временного окна приема RX2) – 2 сек. (RECEIVE_DELAY2 = RECEIVE_DELAY1 + 1 сек.)
- JOIN_ACCEPT_DELAY1 – 5 сек.
- JOIN_ACCEPT_DELAY2 – 6 сек.
- MAX_FCNT_GAP (максимальная разница значений внутреннего счетчика принятых пакетов и номера полученного фрейма – FCNT) – 16384.
- ADR_ACK_LIMIT (в режиме адаптации скорости передачи – предельное кол-во фреймов, направив которые, конечное устройство запрашивает подтверждение со стороны сети) – 64.
- ADR_ACK_DELAY (в режиме адаптации скорости – время ожидания подтверждения со стороны сети после запроса конечным устройством) – 32.
- ACK_TIMEOUT – случайное значение в диапазоне от 1 до 3 сек

1.1.5 Безопасность в сетях LoRa

В сети LoRaWAN обеспечивается полная конфиденциальность данных при прохождении всех задействованных в цепочке устройств, при этом содержимое пакета доступно только отправителю (конечному устройству) и получателю (приложению), для которого оно предназначено. Сетевой сервер оперирует данными в зашифрованном виде, производит аутентификацию и проверяет целостность каждого пакета, но при этом не имеет доступа к полезной нагрузке, т.е. к информации от подключенных сенсоров.

В сети используются три вида ключей, показанные на рисунке 3.

Рисунок 3 – Ключи шифрования LORAWAN

Ключ аутентификации приложения AppKey известен только конечному устройству и серверу приложений. В случае если конечное устройство подключается к сети в режиме Over-The-Air-Activation (OTAA), ключ аутентификации приложения AppKey используется для вычисления сетевого ключа NwkSKey и ключа приложения AppSKey. В случае если конечное устройство подключается к сети в режиме Activation By Personalization (ABP), ключи NwkSKey и AppSKey предустановлены на конечном устройстве.

Ключ NwkSKey известен сетевому серверу и конечному устройству и используется для проверки целостности каждого сообщения, используя Message Integrity Code (MIC). MIC вычисляется по алгоритму AES-CMAC, который аналогичен контрольной сумме, за исключением того, что он предотвращает умышленную подделку сообщений.

Ключ приложения AppSKey используется для шифрования полезной нагрузки, используя алгоритм AES-128, между конечным устройством и сервером приложений.

1.1.6 Активация конечных устройств

Для подключения к сети LoRaWAN каждое конечное устройство должно быть идентифицировано и активировано в сети.

Предусмотрено два режима активации конечных устройств, активация по воздуху – Over-The-Air Activation (OTAA) и активация персонализацией – Activation by Personalization (ABP).

1) Активация по воздуху – Over-The-Air Activation (OTAA)

При активации по воздуху конечные устройства LoRa не привязаны жестко к какой-то конкретной сети. На конечных устройствах LoRa прописываются идентификатор устройства (DevEUI), индификатор приложения (AppEUI) и ключ приложения (AppKey). Конечное устройство при активации инициирует JOIN процедуру. Ключи шифрования (AppSKey и NwkSKey), необходимые для передачи информации, вычисляются самим конечным устройством. Данный метод активации обеспечивает высокий уровень безопасности и рекомендуется для использования.

Формат сообщений JOIN_REQUEST и JOIN_ACCEPT показан на рисунке 4:

Size (bytes)	8	8	2
Join Request	AppEUI	DevEUI	DevNonce

Size (bytes)	3	3	4	1	1	(16) Optional
Join Accept	AppNonce	NetID	DevAddr	DLSettings	RxDelay	CFList

Рисунок 4 − Формат сообщений JOIN_REQUEST и JOIN_ACCEPT

Сообщения содержат следующие поля:

- AppEUI - идентификатор приложения в адресном пространстве IEEE EUI64.
- DevEUI - глобальный идентификатор устройства в адресном пространстве IEEE EUI64.
- DevNonce - случайное число, генерируемое конечным устройством (используется при расчете NwkSKey и AppSKey).
- AppNonce - случайное число, генерируемое сервером приложений (используется при расчете NwkSKey и AppSKey).
- NetID - идентификатор сети, старшие 7 бит которого (31..25) соответствуют идентификатору NwkID, младшие 17 бит могут произвольно назначаться оператором.
- DevAddr - адрес конечного устройства, старшие 7 бит которого соответствуют идентификатору NwkID, младшие 25 бит являются адресом конечного устройства в сети NwkAddr.
- DLSettings - указывает на смещение скорости передачи данных в DL канале окна приема RX1 (относительно скорости передачи данных в UL канале) и скорость передачи данных в DL канале окна приема RX2.
- RXDelay - задержка между завершением передачи данных в UL канале и открытием окна приема RX1.
- CFList - список радиочастотных каналов с Freq Ch4 по Freq Ch8 (первые три канала являются обязательными и неизменными).

Формулы вычисления сетевого ключа NwkSKey и ключа приложения AppSKey

NwkSKey = aes128_encrypt(AppKey, 0x01 | AppNonce | NetID | DevNonce | pad_{16})
AppSKey = aes128_encrypt(AppKey, 0x02 | AppNonce | NetID | DevNonce | pad_{16})

2) Активация персонализацией – Activation by Personalization (ABP)

При активации персонализацией конечные устройства жестко прописываются для работы в конкретной сети оператора. Конечные устройства прошиваются с определенными сетевым ключом (NwkSKey) и ключом приложения (AppSKey). Данный метод активации как правило не используется.

1.2 Протокол MAC уровня от Unwired Device

1.2.1 Формат сообщения MAC уровня

На рисунке 5 приведен формат MAC пакета.

Header						***Payload***	
MHDR 1 байт	MIC 3 байта	Dev_addr 4 байта	Type 1 байт	fID 1 байт	Status 1 байт	Len 1 байт	Data 0…N байт

Рисунок 5 – Формат сообщений MAC уровня

Сообщение содержит в себе следующие поля:

– Header – заголовок сообщения, определяет следующее:

1) MHDR – заголовок пакета MAC уровня. Поскольку, протокол проприетарный , MHDR = 0xFF;

2) MIC – код контроля целостности. Вычисляется по всем полям сообщения по основе алгоритма sha256 и секретного ключа key_mic.

3) Dev_addr – адрес назначения;

4) Type – тип сообщения

Как и в стандартной сети LoRaWAN существует два типа сообщений – сообщения, требующие подтверждения получения и сообщения без подтверждения.

5) fID – серийный номер сообщения.

6) Status – статус узла, в котором задается температура со встроенного датчика температуры и напряжение питания vdd.

– Payload – полезная нагрузка фрейма. Шифруется при помощи ключа key_aes. Содержит следующие поля:
  1) Len - длина данных;
  2) Data – данные.

1.2.2 Окна приема информации

Узел передает данные на базовую станцию короткими посылками по заданному графику. Инициатором обмена выступает сам конечный узел. После завершения каждого сеанса передачи данных конечным устройством (в линии «вверх») открываются два коротких временных окна, в течении которых конечное устройство может принять данные от сети (в линии «вниз»).

Для передачи данных шлюзом LoRa в первом временном окне (RX1) используются те же параметры передачи (включая номер частотного канала и скорость передачи данных), которые использовались для передачи данных конечным устройством.

Для передачи данных шлюзом LoRa во втором временном окне (RX2) используются предустановленные параметры передачи (включая номер частотного канала и скорость передачи данных).

Узел переходит в режим приема, т.е. открывает первое окно приема, сразу после отправки данных на заданный интервал времени Receive_Delay1, далее открывается второе окно приема на время Receive_Delay2. Все остальное время, находится в режиме энергосбережения или сна.

Процесс открытия окон показан на рисунке 6.

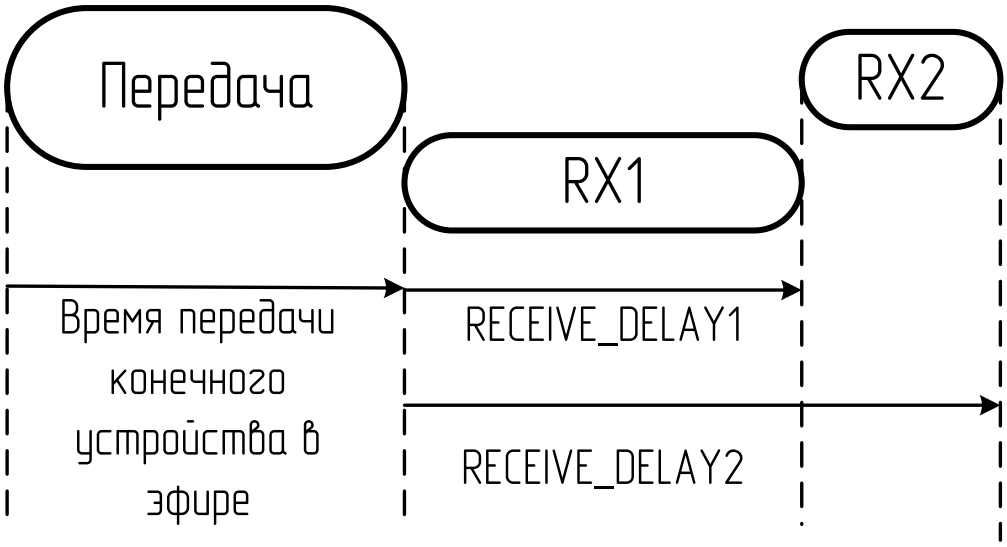

Рисунок 6 – Окна приема информации

1.2.3 Безопасность в сетях

В сети используются три вида ключей. Ключ шифрования сети join_key известен только конечному устройству и серверу приложений. Ключ join_key используется для вычисления ключей key_mic и key_aes.

Для вычисления сетевого ключа key_mic и ключа приложения key_aes используется алгоритм *SHA256*.

Ключ key_mic используется для расчета кода контроля целостности (MIC) на основе алгоритма sha256. Ключ key_aes используется для шифрования полезной нагрузки, используя алгоритм AES-128.

1.2.4 Активация конечных устройств

Для подключения к сети LoRaWAN каждое конечное устройство должно быть идентифицировано и активировано в сети. В данном варианте протокола используется активация по воздуху.

На конечном устройстве прописываются:

- идентификатор устройства (Dev_id);
- идентификатор приложения (App_id);

- Класс устройства (node_class);
- Ключ шифрования сети (join_key).

Конечное устройство при активации инициализирует join процедуру. Ключи шифрования (key_mic и key_aes) необходимые для передачи информации вычисляется самим конечным устройством.

Формат сообщения join request и join accept показан на рисунках 7 и 8.

Size	8 байт	8 байт	4 байта	1 байт
Join request	Dev_id	App_id	Dev_nonce	Node_class

Рисунок 7 – Формат сообщения join request

Size	8 байт	8 байт	4 байта
Join accept	Dev_id	Addr	app_nonce

Рисунок 8– Формат сообщения join accept

Сообщения содержат следующие поля:

- Dev_id - идентификатор устройства;
- App_id - идентификатор приложения;
- Dev_nonce – случайное число, генерируемое конечным устройством;
- Node_class – класс устройство. Устройство LoRA может работать:

Класс A - Node_class = 0;

Класс B - Node_class = 1;

Класс B - Node_class = 2;

- Addr – адрес конечного устройства в сети;
- app_nonce - случайное число, генерируемое сервером приложений.

При инициализации join процедуры необходимо предусмотреть следующее:

- В заголовке dev_add = 0xFFFFFFFF; fID = 0; mhdr = 0xFF.
- В полезной нагрузке фрейма dev_nonce = 0.

При расчете MIC и для шифрования полезной нагрузки в сообщении join request используется ключ join_key.

Формат сообщения Join Request показан на рисунке 9.

Header						***Payload***				
MHDR	MIC 24 бита	Dev_addr 32 бит	type	fID 8бит	Status 8 бит	Len 8 бит	Data 0…255 слов			
0xFF	MIC	0xFFFFFFFF	LS_UL_JOIN_REQ	0	Status	Len	Dev_id	App_id	0	Node_class

Рисунок 9– Формат сообщения Join Request

1.2.5 Основные константы

Основные константы, используемые в стеке протокола от Unwired Device:

− RECEIVE_DELAY1 (длительность временного окна приема RX1) – 3 секунды.

− RECEIVE_DELAY2 (длительность временного окна приема RX2) – RECEIVE_DELAY1+ 1 = 4 секунды.

− ACK_TIMEOUT – 15 секунд.

1.3 Выводы

В данной главе было исследована спецификация LoRaWAN и вариант протокола MAC уровня от Unwired Device. Особое внимание было уделено рассмотрению формата MAC пакета, времени запуска и длительности временного окна приема, безопасности в сетях и активации конечных устройств в сети.

Можно выделить следующие особенности протокола MAC уровня от Unwired Device:

− Формат сообщения MAC уровня отличается от спецификации LoRaWAN. Сообщение содержит в себе заголовок сообщения и полезную нагрузку фрейма. Заголовок сообщения определяет заголовок пакета MAC уровня, код

контроля целостности, адрес назначения и тип сообщения. Полезная нагрузка фрейма включает в себя длину данных и данные.

- Как и в сети LoRaWAN существует два типа сообщений – сообщения, требующие подтверждения получения и сообщения без подтверждения.

- Время запуска и длительности временного окна приема отличается от спецификации. Узел переходит в режим приема, т.е. открывает первое окно приема, сразу после отправки данных на заданный интервал времени, равный трем секундам, далее открывается второе окно приема еще на одну секунду.

- Адаптация скорости передачи данных не используется. Скорость задается пользователем и не изменяется.

- В сети используются три вида ключей, аналогично спецификации, однако изменен алгоритм шифрования. Для вычисления сетевого ключа key_mic и ключа приложения key_aes используется алгоритм *SHA256*. Ключ key_mic используется для расчета кода контроля целостности (MIC) на основе алгоритма sha256. Ключ key_aes используется для шифрования полезной нагрузки, используя алгоритм AES-128.

- Для активации конечных устройств используется активация по воздуху.

- Изменен формат сообщения join request и join accept.

2 Структура программно - аппаратного комплекса

Типичная архитектура интернета вещей состоит из следующих элементов: конечные узлы, шлюзы, сетевой сервер[9].

На рисунке 10 показана схема построения сети LoRaWAN.

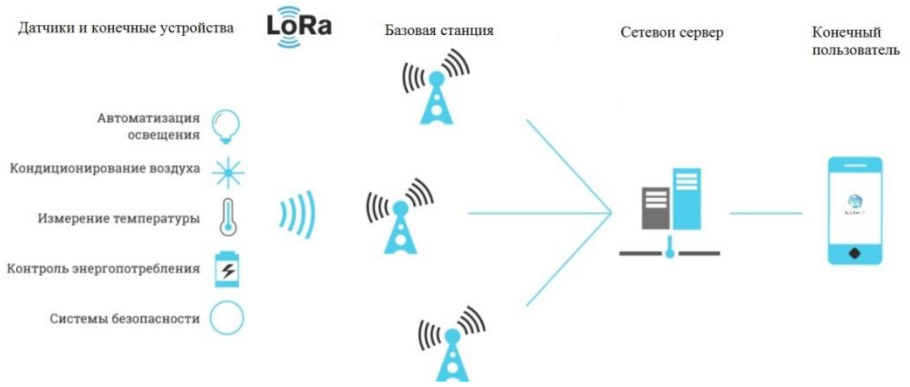

Рисунок 10 – Архитектура LoRaWAN

Узлы также известны как конечные устройства, используются для измерения или управления внешними системами. Конечные устройства связываются по беспроводной связи со шлюзами и имеют низкое энергопотребление.

Шлюзы передают данные с узлов на сетевой сервер. Их меньше, потому что один шлюз может поддерживать тысячи конечных устройств. Поскольку соединение между шлюзом и сетевым сервером осуществляется через IP-соединения, пакеты должны быть преобразованы. Шлюзы действуют как мосты, преобразуя RF-пакеты в IP-пакеты или наоборот.

Сетевой сервер предназначен для управления сетью: заданием расписания, адаптацией скорости, хранением и обработкой принимаемых данных.

Данная глава предназначена для определения места разрабатываемого устройства в сети LoRaWAN, исследованию аппаратного обеспечения такого устройства

2.1 Обзор элементной базы учебного набора Unwired Kit

На рисунке 11 представлен программно-аппаратный лабораторный комплекс предложенный компанией Unwired Devices [10].

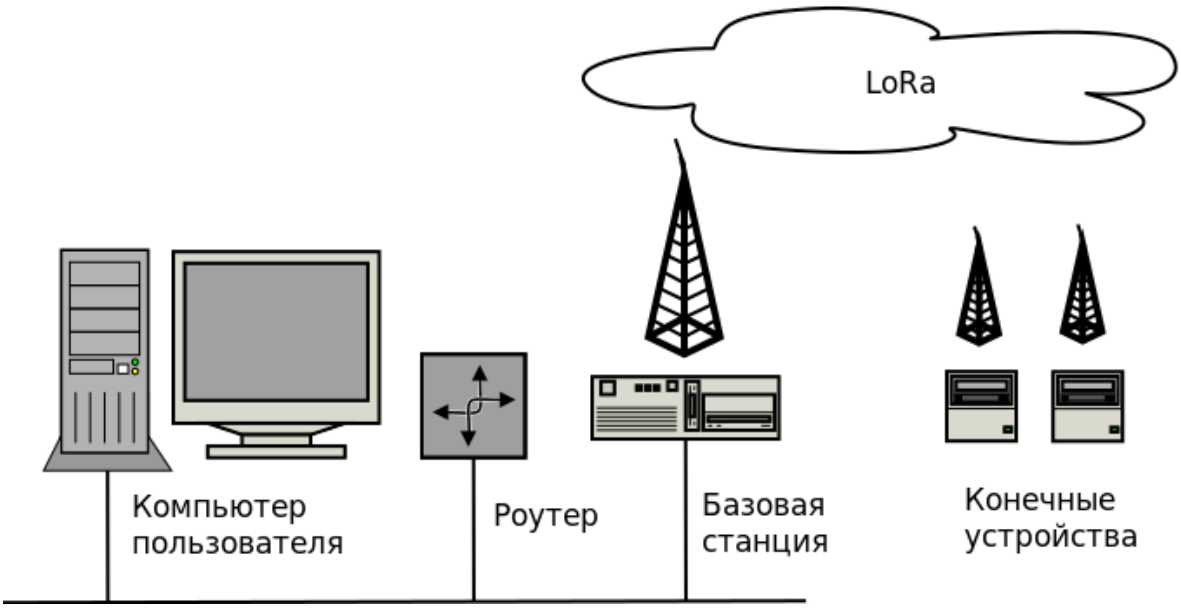

Рисунок 11 − Программно-аппаратный лабораторный комплекс

В качестве конечного устройства используется плата Unwired Range совместно с платой UMDK-RF. Конечное устройство показано на рисунке 12. Unwired Range содержит в себе микроконтроллер STM32 и приемопередатчик LoRa, как показано на рисунке 13. Плата UMDK-RF - это адаптер-переходник. К ней можно подключать Unwired Range или различные датчики и исполнительные устройства. Конечное устройство считывает измеряемые данные с датчиков, управляет исполнительными механизмами и осуществляет радиосвязь с LoRaWAN шлюзом (базовой станцией).

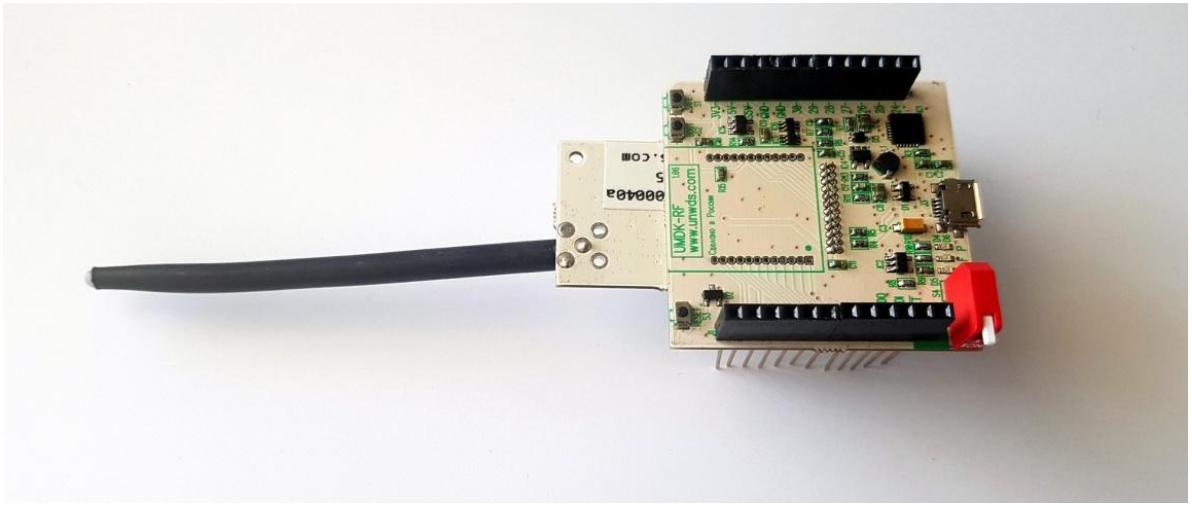

Рисунок 12– Конечное устройство

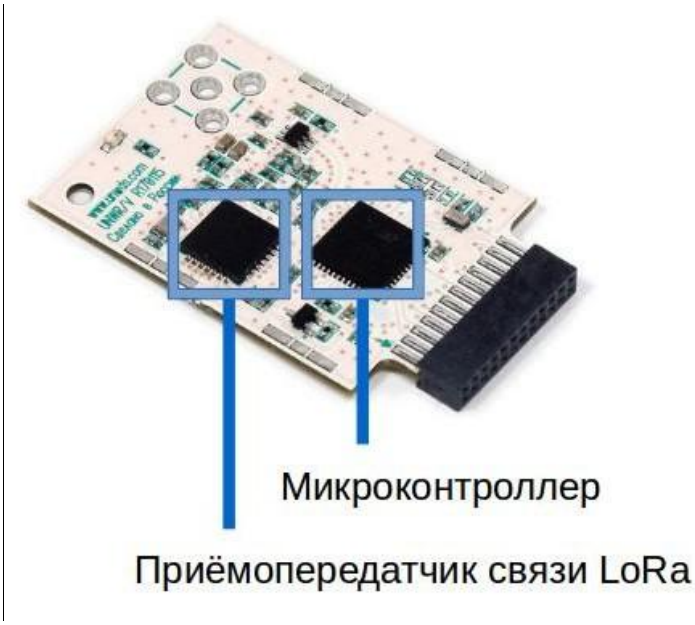

Рисунок 13 − Плата Unwired Range

Базовая станция UNWD-BASE представляет собой комбинацию из радиомодуля Unwired Range и микрокомпьютера Artic 10 с ОС OpenWRT Linux. Задача микрокомьютера — поддерживать работу радиочастотной сети, включая механизмы аутентификации, авторизации и шифрования, а также транслировать компактный бинарный протокол радиообмена в сообщения в формате JSON для установленного на микрокомпьютере MQTT-брокера. Подключиться пользователю к UNWD-BASE можно по сети Ethernet или с помощью Wi-Fi роутера.

Результатом магистерской диссертации является разработка конечного устройства LoRaWAN. Остальные узлы, такие как компьютер пользователя, роутер и базовую станцию предлагается использовать из учебного набора Unwired Kit. Предлагаемое конечное устройство должно быть универсальным. Данное устройство должно быть совместимо с устройствами фирмы Unwired Device, что позволит использовать данное устройство в учебных целях. Кроме того, необходимо учесть возможность использования данного устройства для последующих разработок и подключения к открытым сетям LoRaWAN, развернутым на территории Челябинска. Разрабатываемая модель должна иметь возможность работать с различными датчиками и исполнительными устройствами.

2.2 Конечное устройство LoRaWAN

Оконечный узел сети LoRaWAN представляет собой беспроводной модуль, объединяющий в своем составе приемопередатчик с необходимой для заданной частоты пассивной обвязкой и отдельный микроконтроллер для хранения стека протокола LoRaWAN. Структура аппаратного обеспечения типового устройства LoRaWAN соответствует универсальной архитектуре конечного устройства, приведенной на рисунке 14.

Оконечное устройство включает в себя:

- Микроконтроллер, который отвечает за считывание измеряемых данных с датчиков, управление исполнительными механизмами, а так же реализацию стека протокола LoRaWAN;
- LoRaWAN-трансивер, который состоит из приемопередатчика, реализованного на базе технологии LoRa, антенного тракта и непосредственно самой антенны. Благодаря радиотрансиверу осуществляется радиосвязь с LoRaWAN шлюзом, а также подключение конечного устройства к LoRaWAN сети;
- Периферия: различные датчики (температуры, влажности, давления, дыма и пр.), измерительные и исполняющие устройства, имеющие интерфейс ввода-вывода.

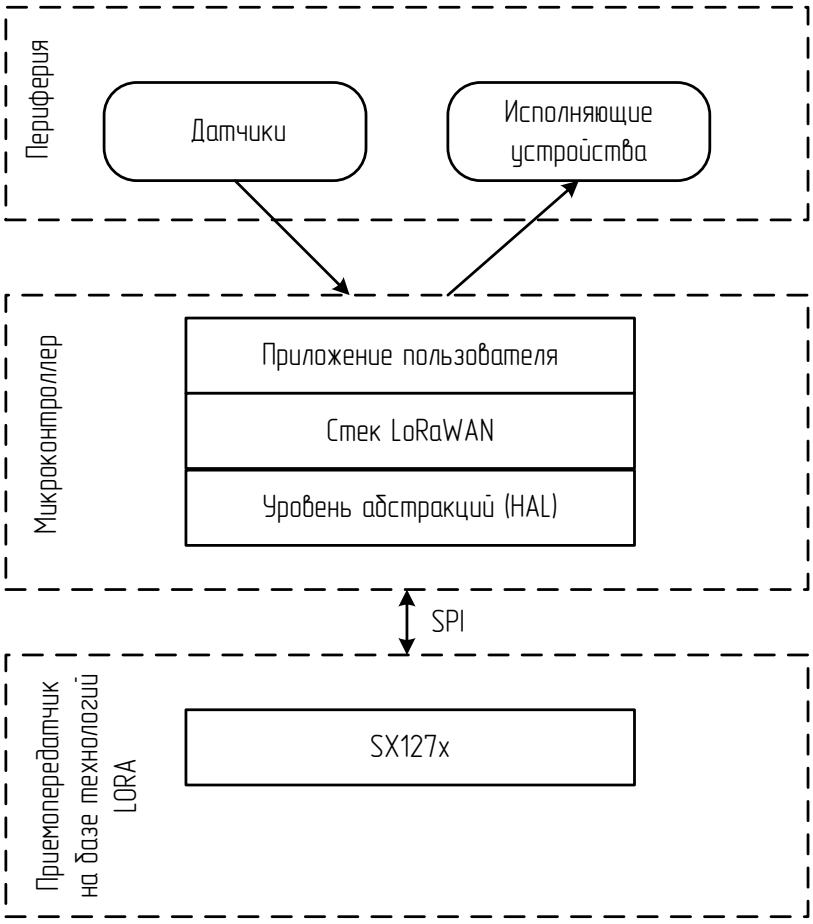

Рисунок 14 – Универсальной архитектуре конечного устройства

Связь между микроконтроллером и радиомодулем осуществляется по интерфейсу SPI.

Электропитание LoRaWAN-устройств может осуществляться от внешнего источника, посредством соединения с электрической сетью или от автономного источника (батареи).

Согласно ТЗ, в качестве радиотрансивера LORA необходимо использовать радиотрансивер RFM95W.

Главные критерии выбора микроконтроллера, при создании устройства интернета вещей:

- низкое энергопотребление;
- наличие отладочных средств;

– наличие собственных библиотек целевой платформы.

Для отладки программного обеспечения в составе макета, так же для учебных целей нет жесткого требования в экономии энергии, устройство может постоянно питаться от внешнего источника. Поэтому было решено использовать микроконтроллер stm32f103c8t6, с последующей заменой на stm32L серии для возможности работы в режиме пониженного энергопотребления.

2.3 Описание макета

Упрощенная блок схема макета показана на рисунке 15.

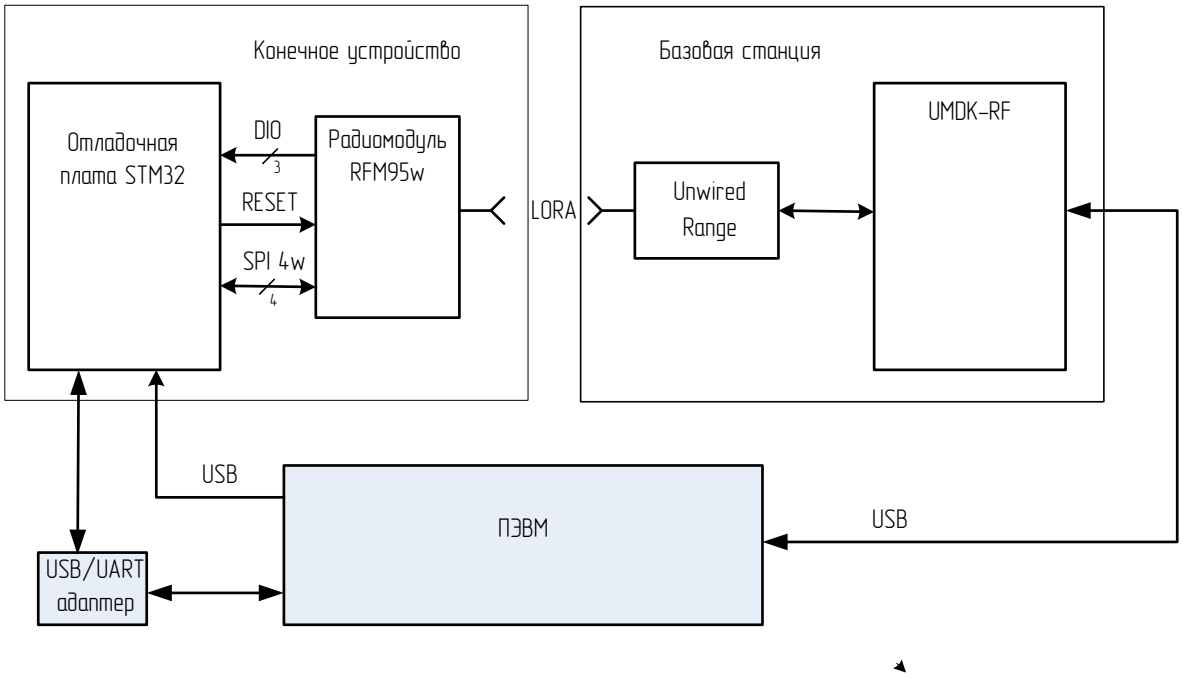

Рисунок 15 – Блок схема макета

Конечное устройство состоит из отладочной платы STM32F103C8T6 , отладчика/программатора ST-Link и радиотрансивера RFM95w. Обмен данными с радиотрансивером происходит через SPI интерфейс. В радиотрансивере для отслеживания состояния передатчика и приемника предусмотрено три вывода DIO, и вывод RESET для сброса. Для связи с пользователем используется интерфейс UART.

В качестве базовой станции используется плата Unwired Range совместно с

платой адаптера-переходника UMDK-RF. Unwired Range содержит в себе микроконтроллер STM32 и приемопередатчик LoRa.

Работа пользователя с конечным устройством осуществляется через консольное приложение на персональном компьютере, с базовой станцией - через командную консоль LINUX.

2.4 Выводы

В данной главе была изучена архитектура сетей LoRaWAN и определено места разрабатываемого устройства в такой сети. Проведен обзор элементной учебного набора Unwired Kit. Была предложена разработка оконечного устройства, остальные узлы, такие как компьютер пользователя, роутер и базовую станцию предлагается использовать из учебного набора Unwired Kit.

Изучена универсальная архитектура конечного устройства. Оконечное устройство включает в себя микроконтроллер, LoRaWAN-трансивер и периферию: различные датчики, измерительные и исполняющие устройства.

Произведен выбор микроконтроллера и радиотрансивера. Разрабатываемое устройство состоит из отладочной платы STM32F103C8T6, отладчика/программатора ST-Link и радиомодуля RFM95w. Для связи с пользователем предусмотрен интерфейс UART.

3 Алгоритмы и программное обеспечение устройства

В настоящий момент существуют несколько открытых программных реализации стека протокола LoRaWAN, такие как LoRaMAC, предложенная компанией Semtech, и LMiC (LoRa WAN in C) от IBM. Обе библиотеки используют концепцию HAL драйверов, облегчающую процедуру портирования программ при изменении аппаратной части оборудования. Для реализации оконечного устройства было решено использовать библиотеку LMIC.

Библиотека LMIC обеспечивает реализацию стандарта на MAC-уровне, абстрагируется от физических тонкостей протокола LORA и представляет собой драйвер для работы с радиотрансивером.

Для реализации предложенного универсального модема исходный код библиотеки LMIC был модифицирован. Программное обеспечение устройства реализует как MAC – уровень протокола LoRaWAN, так и уникальный протокол, по которому работают устройства Unwired Device. Кроме того предусмотрена связь с пользователем через последовательный интерфейс UART. С помощью команд высокого уровня пользователь может выбрать, согласно какому протоколу, устройство должно работать, настроить параметры работы в сети. Через последовательный интерфейс устройство информирует пользователя об определенных изменениях состояния внутри модема.

Данная глава посвящена рассмотрению программного обеспечения и основных алгоритмов работы.

3.1 Структура программного обеспечения

Архитектура программного обеспечения устройства можно представить как набор компонентов, выстроенных в определённую иерархию:

− нижний уровень (уровень абстракции HAL) — код, непосредственно работающий с используемым микроконтроллером, необходим облегчения процедуры портирования программ при изменении аппаратной части оборудования.

– промежуточный уровень — реализует библиотеки протоколов связи, драйверы различных внешних устройств, планировщик задач, различные вспомогательные службы;

– верхний уровень — собственно пользовательское приложение. Содержит все функциональные приложения, в которых реализованы функции управления устройством.

На рисунке 16 представлен общий состав программного обеспечения устройства, назначение файлов и их взаимосвязь.

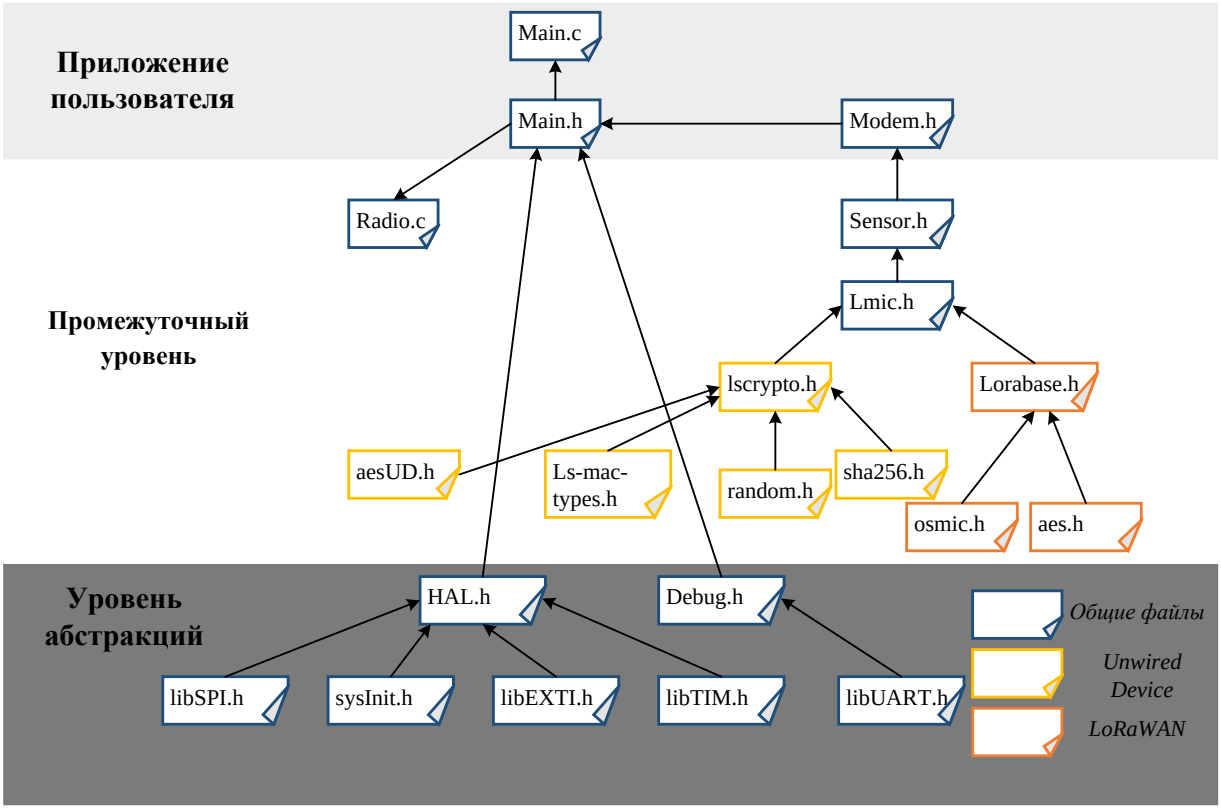

Рисунок 16 – Структура программного обеспечения

3.1.1 HAL интерфейс

На уровне абстракции HAL были настроены тактовые частоты, периферийные модули, разрешены прерывания, где это необходимо. Настройка тактовой частоты находится в файле sysInit.c. Для корректной работы устройства используются следующие модулями микроконтроллера:

- Модуль интерфейса SPI, используется для чтения и записи регистров радиосвязи. Процесс инициализации прописан в файле libSPI.c, функции для работы с модулем – в файле HAL.c;
- Таймер TIM2. Блок таймера необходим для точной записи событий и планирования новых действий протокола. Процесс инициализации прописан в файле libTIM.c;
- выводы GPIO, используются для работы с радиомодулем. В режиме вывода требуются три линии цифрового ввода-вывода для управления антенным переключателем радиостанции (RX и TX), выбором микросхемы SPI (NSS) и линией сброса (RST).В режиме ввода необходимы три линии цифрового ввода-вывода для определения состояния передатчика и приемника радиосигнала (DIO0, DIO1 и DIO2).Процесс инициализации прописан в файле libSPI.c, libEXTI.c и sysInit.c, функции для работы с выводами GPIO – в файле HAL.c;
- Модуль интерфейса UART, используется для связи с пользователем. Процесс инициализации прописан в файле libUART.c, функции для работы с модулем – в файле debug.c ;
- Модуль АЦП, используется для считывания напряжения питания микроконтроллер и температуры. Процесс инициализации и функции для работы с модулем прописаны в файле adc.c,;
- Контроллер прерывания необходим для пересылки прерываний, сгенерированных цифровыми входными линиями.

3.1.2 Промежуточный уровень

На промежуточном уровне реализованы протоколы связи, настраивается связь с радиотрансивером, реализуется планировщик задач и прописаны драйвера для работы с датчиками.

Все функции реализующие MAC уровень протоколов описаны в файле lmic.h. В файлах ls-mac-types.h и lorabase.h описан формат кадра и региональные параметры для протокола от unwired device и протокола LoRaWAN соответственно. В файлах aesUD.c, sha256.c и Random.c описаны функции реализующие шифрование сообщений для протокола unwired device. . В файле aes.c описаны функции реализующие шифрование сообщений для стандартного протокола LoRaWAN.

В файле osmih.c реализуется планировщик задач. Библиотека LMiC предлагает простую модель программирования на основе событий, где все события протокола отправляются на функцию обратного вызова onEvent ().

Весь код приложения выполняется в так называемых заданиях, которые выполняются в главном потоке с помощью функции планировщика времени выполнения os_runloop () . Эти прикладные задания кодируются как обычные функции C и могут управляться с помощью функций планирования задач. Для управления заданиями требуется дополнительная структура управления заданиями osjob_t, которая идентифицирует задание и сохраняет контекстную информацию. Задания не должны быть длительными, чтобы обеспечить бесперебойную работу!

Они должны только обновлять состояние и планировать действия, которые будут запускать новые задания или обратные вызовы событий. Функции планирования задач, находящиеся в файле osmih.c включают инициализацию, планирование и выполнение заданий.

В файле radio.прописаны функции реализующие инициализацию радиотрансивера и настройку радиотрансивера в режим приема и в режим передачи. Настройка осуществляется через запись соответствующих регистров.

3.1.3 Приложение пользователя

Точкой входа в программу является функция main() в файле main.c. Эта функция начальную инициализацию устройства и создание первой задачи.

```
int main () {
// настройка тактирования
   InitRCC ();
// создание первой задачи
   osjob_t initjob;
// инициализация планировщика задач
   os_init();
// инициализация UART
   debug_init();
// настройка первой задачи
   os_setCallback(&initjob, initfunc);
// выполнять запланированные задания и события
   os_runloop();
   return 0;
}
```

Первой вызывается функция InitRCC (), которая выполняет инициализацию микроконтроллера, системы синхронизации и т.д. После ее вызова микроконтроллер готов к функционированию на штатной тактовой частоте. Далее выполняется инициализация среды выполнения с помощью функции os_init () и запускается функция планировщика заданий os_runloop (). Чтобы загружать действия протокола и генерировать события, необходимо настроить начальное задание. Поэтому задание запуска назначается с помощью функции os_setCallback ().

```
static void initfunc (osjob_t* j) {
// сброс состояния MAC
   LMIC_reset();
// инициализация модема
   modem_init();
}
```

Код, показанный в функции initfunc (), инициализирует MAC и работу модема modem_init().

В файле modem.c реализуется связь с пользователем по интерфейсу UART и начинается соединение с сетью.

3.2 Настройка радиотрансивера

Настройка радиотрансивера в режим передачи производилась в функции txlora(), в режим приема rxlora(). Функции находятся в файле radio.c. Настройка осуществляется через запись соответствующих регистров. Для режима приема радиотрансивера RFM95 были использованы настройки, представленные в таблице 1

Таблица 1

Регистры (адрес)	Биты	Значение	Примечание
RegOpMode (0x01)	LongRangeMode[7]	‘1’	Включение режима LoRa
	Mode [2:0]	‘101’	Непрерывный прием
RegModemConfig1 (0x1D)	BW[7:4]	‘0111’	Полоса пропускания 125 кГц
	CodingRate[3:1]	‘001’	Скорость кодирования = 4/5
	ImplicitHeaderModeOn [0]	‘0’	Пакеты имеют открытый заголовок
RegModemConfig2 (0x1E)	SpreadingFactor[7:4]	От ‘0111’ до ‘1100’	Коэффициент расширения спектра SF, опредиляет скорость передачи данных Для ‘0111’ (SF7) – 6кбит/с Для ‘1100’ (SF12) – 300бит/с
	RxPayloadCrcOn	‘1’	Контрольная сумма пакета
RegModemConfig3 (0x26)	LowDataRateOptimiza[3]	‘0’ или ‘1’	‘0’− когда SF<=10; ‘1’ − когда SF>=11;
	AgcAutoOn[2]	‘1’	Усиление установлено внутренней петлей АРУ
RegFfMsb (0x06)	Frf[23:16]	FQ = (FRF * 32 Mhz) / (2 ^ 19) Старшие 8 бит	Задается несущая частота

Продолжение таблицы 1

Регистры (адрес)	Биты	Значение	Примечание
RegFfMid (0x07)	Frf[15:8]	FQ = (FRF * 32 Mhz) / (2 ^ 19) средние 8 бит	Задается несущая частота
RegFfLsb (0x08)	Frf[15:8]	FQ = (FRF * 32 Mhz) / (2 ^ 19) младшие 8 бит	Задается несущая частота
RegLna(0x0C)	LnaBoostHf[1:0]	‘11’	Включение Lna Boost
RegMaxPayloadlength (0x23)	PayloadMaxLength [7:0]	0xFF	Установка максимально-возможный размер полезной нагрузки нисходящего канала.
RegInvertIQ (0x33)	InvertIQ[6]	‘0’	I и Q сигналы не инвертированы
(0x3B)	[7:0]	0x1D	Зарезервировано
RegPreambleLsb (0x21)	Preamblelength [7:0]	0x08	Длина преамбулы
RegSymbTimeoutLsb (0x1F)	SymbTimeout[7:0]	0x05	Длина окна приема в символах
(0x36)	[7:0]	0x03	Зарезервировано
(0x30)	[7:0]	0x00	Зарезервировано
(0x2F)	[7:0]	0x40	Зарезервировано
RegDioMapping1 (0x40)	DioMapping0[7:6]	‘00’	Установка DIO0 в режим RXDONE
	DioMapping1[5:4]	‘00’	Установка DIO1 в режим RXTOUT
	DioMapping2[3:2]	‘11’	Установка DIO2 в режим NOP
RegIrqFlagsMask (0x11)	[7:0]	0x215	Замаскировали прерывания
RegIrqFlags (0x12)	[7:0]	0xFF	Сброс всех прерываний

Для режима передачи радиотрансивера RFM95 были использованы настройки, представленные в таблице 2

Таблица 2

Регистры (адрес)	Биты	Значение	Примечание
RegOpMode(0x01)	LongRangeMode[7]	‘1’	Включение режима LoRa
	Mode[2:0]	‘011’	Режим передачи
RegPaRamp (0x0A)	PaRamp[3:0]	‘1000’	50 мкс

Продолжение таблицы 2

Регистры (адрес)	Биты	Значение	Примечание
RegModemConfig1(0x1D)	BW[7:4]	‘0111’	Полоса пропускания 125 кГц
	CodingRate[3:1]	‘001’	Скорость кодирования = 4/5
	ImplicitHeaderModeOn[0]	‘0’	Пакеты имеют открытый заголовок
RegModemConfig2 (0x1E)	SpreadingFactor[7:4]	От ‘0111’ до ‘1100’	Коэффициент расширения спектра SF, опредиляет скорость передачи данных Для ‘0111’ (SF7) – 6кбит/с Для ‘1100’ (SF12) – 300бит/с
	RxPayloadCrcOn	‘1’	Контрольная сумма пакета
RegModemConfig3 (0x26)	LowDataRateOptimiza[3]	‘0’ или ‘1’	‘0’− когда SF<=10; ‘1’ − когда SF>=11;
	AgcAutoOn[2]	‘1’	Усиление установлено внутренней петлей АРУ
Test39(0x39)	LoRa sync word	0x34	Установка синхрослова для сети LoRaWAN

3.3 Алгоритмы работы модуля

На рисунке 17 представлен общий алгоритм работы модуля. После включения питания происходит инициализация микроконтроллера, здесь настраивается тактовая частота, периферийные модули, выводы микроконтроллера, разрешают-

ся прерывания. Инициализация радиомодуля включает в себя начальную настройку приемо-передатчика на базе технологий LORA и настройка планировщика заданий.

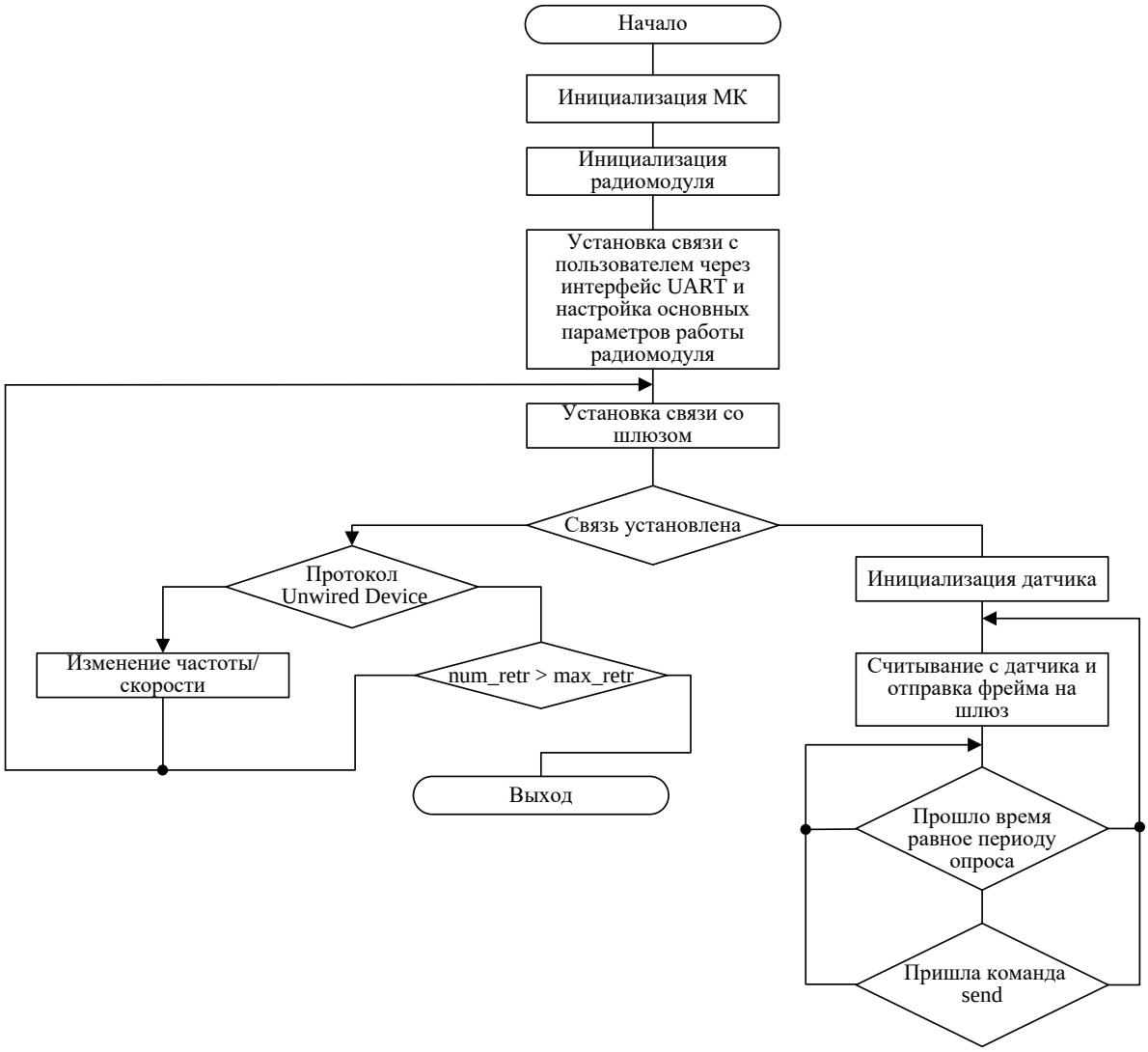

Рисунок 17 – Общий алгоритм работы модема

Предусмотрена связь с пользователем через последовательный интерфейс UART. С помощью команд высокого уровня пользователь выбирает, согласно какому протоколу, устройство должно работать, и настраивает параметры работы в сети, такие как уникальный идентификатор приложения, идентификационный номер радиомодуля, ключ шифрования, скорость передачи данных, максимальное количество попыток передать данные (только для Unwired Device).

Кроме того пользователь может настроить период опроса датчиков, с дискретой равной одной минуте.

После установки всех необходимых параметров, модуль пытается подключиться к сети LoRaWAN. Для этого он формирует и отравляет согласно выбранному варианту протокола сообщение JOIN. В протоколе Unwired Device предусмотрено ограниченное количество попыток соединения к сети. При исчерпании попыток модуль прекращает работу. В оригинальном протоколе LoRaWAN модуль продолжает пытаться присоединиться к сети, используя разные частотные каналы и скорости передачи.

После успешного присоединения к сети устройство инициализирует работу датчиков и начинает опрашивать датчики с периодом, который был настроен ранее, формировать фрейм согласно выбранному варианту протокола, и отравляет сообщение на базовую станцию. Предусмотрена возможность опроса датчиков и оправки сообщения на шлюз по команде send от пользователя.

3.4 Алгоритмы работы пользователя

Связь пользователя с модулем происходит через приложение терминал, с помощью команд высокого уровня. Связь с ПК осуществляется по последовательному интерфейсу передачи данных UART. На все команды пользователя, модем отвечает соответствующим ответным сообщением. Более подробно используемые команды описаны в разделе 4. В случае введения неверной команды в консоли отразится сообщение об ошибке.

Алгоритм работы пользователя показан на рисунке 18. После включения питания, пользователю необходимо выбрать вариант протокола на котором будет работать модем в сети. Далее в консоли отобразятся первичные настройки работы модема. В случае необходимости, пользователь в течение 20 секунд может обнулить ключ шифрования или все настроек радиомодема. После обнуления, необходимо ввести новые значения параметров, сохранить изменения. После чего снова запуститься время ожидания. После того как время ожидания прошло, считается

что пользователь согласен со всеми текущими настройками, далее настройки изменить невозможно. Модем сохраняет все текущие настройки и начинает работать согласно протоколу. После активации устройства в сети, предусмотрена возможность моментальной отправки сообщения шлюзу при помощи соответствующей команды.

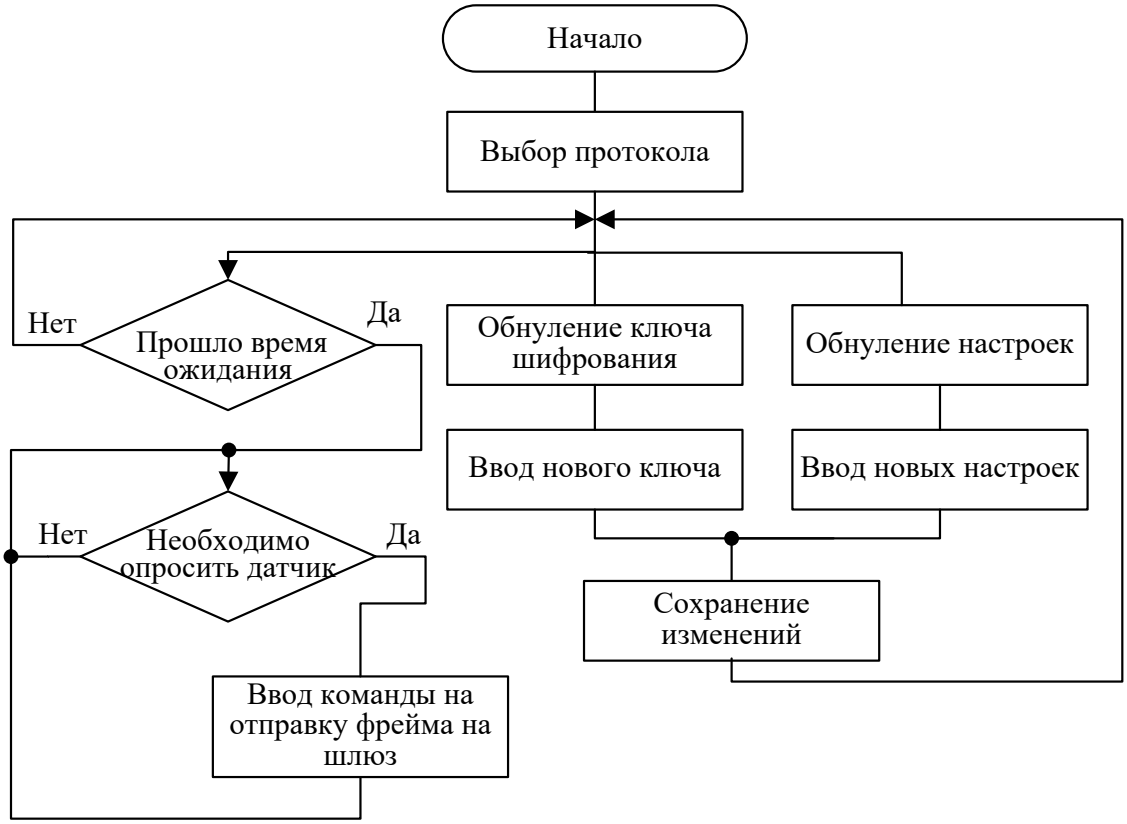

Рисунок 18 − Алгоритмы работы пользователя

3.5 Алгоритм работы радиомодуля в сети LoRaWAN

На рисунке 19 представлен алгоритм работы радиомодуля в сети, а именно отправка сообщения, открытие окон приема информации и прием сообщений.

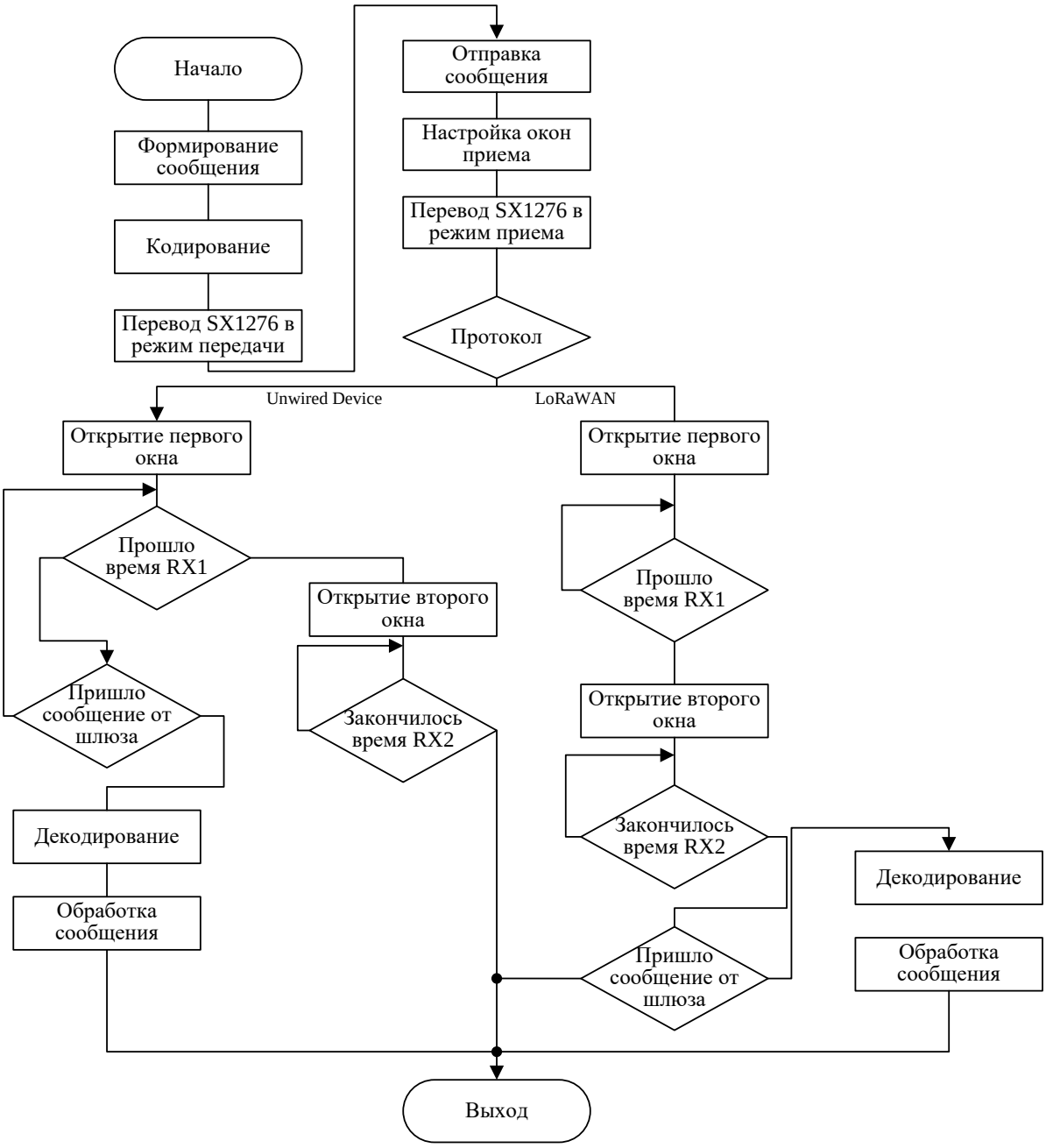

Рисунок 19 – Алгоритм работы радиомодуля в сети LoRaWAN

После прохождения процедуры присоединения к сети, модем работает по следующему алгоритму:

1) Модем с выбранным интервалом времени опрашивает датчики и формирует сообщение согласно спецификации используемого протокола. Формирование сообщения происходит в buildDataFrame(). Данные функции можно найти в файле lmic.c

2) Сформированное сообщение проходит процедуру шифрования. Про-

цедура шифрования осуществляется в функции ls_encrypt_frame() для протокола Unwired Device и в функции aes_appendMic() для протокола LoRaWAN.

3) Радиотрансивер переводится в режим передачи с помощью функции txlora(), находящейся в файле radio.c.

4) После завершения сеанса передачи данных конечным устройством (в линии «вверх»), радиотрансивер переводится в режим приема и открывается первое временное окно заданный интервал времени Receive_Delay1. Используемая функция - rxlora(), находящейся в файле radio.c, а также setRX1 и setupRx1.

5) Если используется протокол Unwired Device модем проверяет пришло ли сообщение от шлюза, если пришло – сообщение декодируется и обрабатывается, т.е выделяются полезные данные.

6) Открывается второе окно приема на время Receive_Delay2.

7) Если используется протокол LoRaWAN модем проверяет пришло ли сообщение от шлюза, если пришло – сообщение декодируется в функции decodeFrame () и обрабатывается, т.е выделяются полезные данные.

3.6 Выводы

Данная глава посвящена рассмотрению структуры программного обеспечения устройства и алгоритмов работы устройства.

Для реализации универсального модема была модифицирована открытая библиотека LMIC, реализующая стек протокола LoRaWAN. Основной задачей стояло реализация MAC – уровня протоколов LoRaWAN и уникального протокола, по которому работают устройства Unwired Device с возможностью их переключения, а так же предусмотреть связь с пользователем через последовательный интерфейс UART.

Архитектура программного обеспечения устройства можно представить как набор компонентов, выстроенных в определённую иерархию.

На нижнем уровне абстракции HAL были настроены тактовые частоты, периферийные модули, разрешены прерывания, где это необходимо.

На промежуточном уровне реализованы протоколы связи, настраивается связь с радиотрансивером, реализуется планировщик задач и прописаны драйвера для работы с датчиками.

Верхний уровень представляет собой пользовательское приложение. Здесь содержатся все функциональные приложения, в которых реализованы функции управления устройством.

В главе так же подробно описаны алгоритмы работы модуля и представлены соответствующие блок – схемы.

4 Интерфейс модема

В разработанном программно – аппаратном комплексе предусмотрена связь с пользователем через последовательный интерфейс UART. С помощью команд высокого уровня пользователь может выбрать, согласно какому протоколу, устройство должно работать, настроить параметры работы в сети. Через последовательный интерфейс устройство информирует пользователя об определенных изменениях состояния внутри модема.

В данной главе представлена инструкция по работе с радиомодулем.

4.1 Соединение

Устройство подключается к ПК через стандартный последовательный интерфейс USART с настройками связи 115200 бит/с, в полудуплексном режиме.

4.2 Сообщения

Устройство обрабатывает все радиосообщения и состояния протокола LoRaWAN и может управляться пользователем через последовательный канал. Он распознает набор команд для настройки и запроса параметров, а также для обмена данными с сетью.

На все команды модем отвечает соответствующими ответными сообщениями.

В дополнение к ответным сообщениям модем может генерировать сообщения о событиях. Эти сообщения о событиях могут информировать пользователя об определенных изменениях состояния внутри модема, инициируемых протоколом.

Модем поддерживает набор команд ASCII.

Если команда введена неверно, в терминале отображается ответное сообщение ERROR.

4.3 Настройка параметров радиомодуля

Для настройки параметров радиомодуля необходимо запустить программу терминала (например, Termite) и открыть нужный COM-порт, выбрав параметры 115200-8N1, без контроля потока. Терминал должен поддерживать полный перенос строки с возвратом каретки (CR-LF) для устройств, передающих строки только с символом переноса (LF).

При корректном подключении, после загрузки прошивки в микроконтроллер, в терминале появится сообщение о выборе протокола LoRaWAN, как показано на рисунке 20.

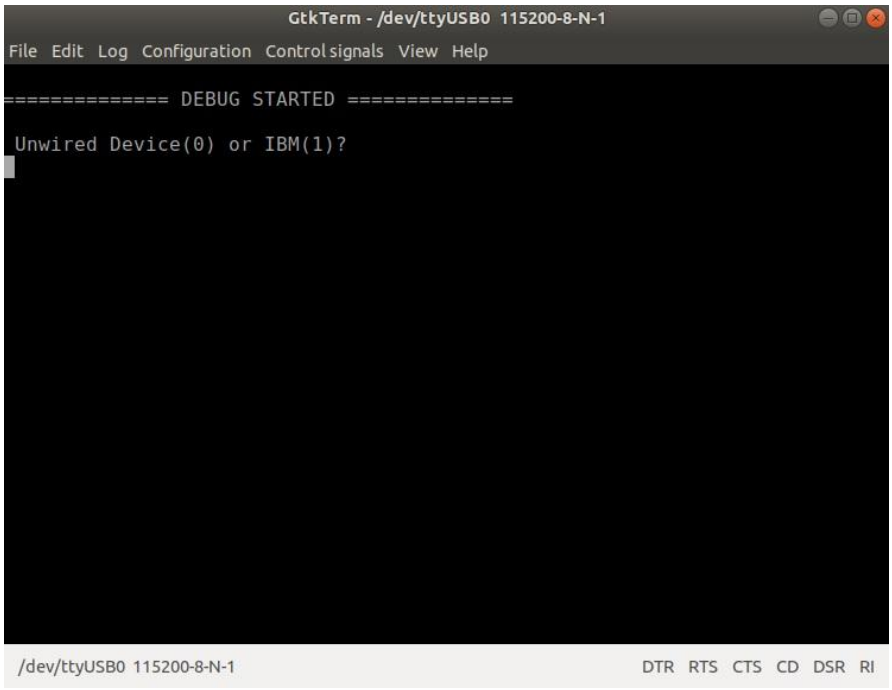

Рисунок 20 – Сообщение о выборе протокола

После выбора протокола, в терминале отразится заставка первичных настроек радиомодуля, как показано на рисунке 21.

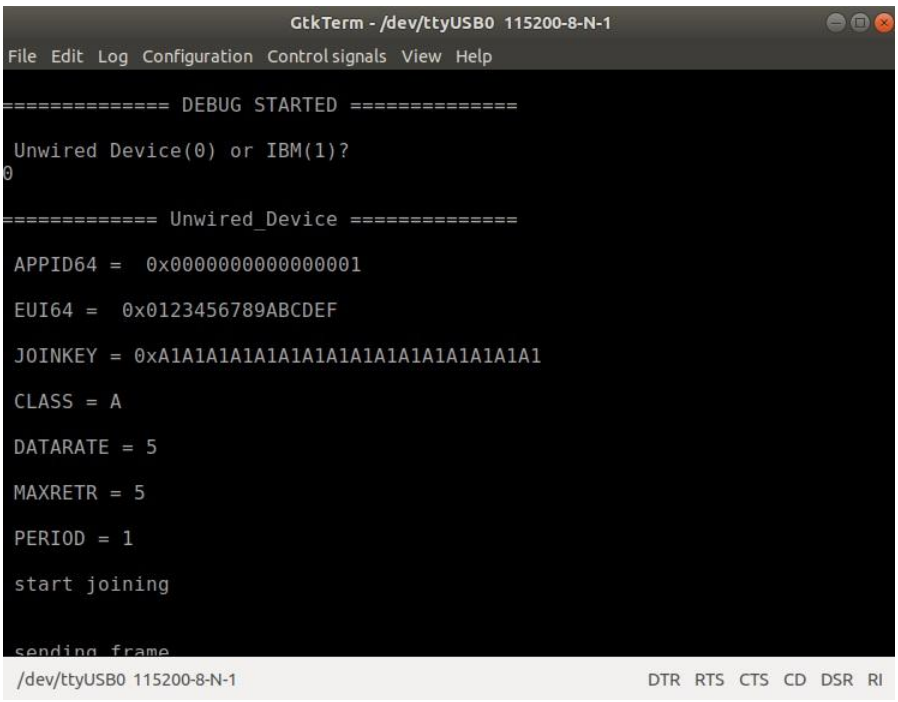

Рисунок 21 – Первичные настройки радиомодуля

Ниже представлены параметры, доступные для настройки модуля оконечного устройства:

– APPID64 - уникальный идентификатор приложения, работающего на радиомодуле. APPID64 может использоваться на шлюзе или сервере сети LoRaWAN для определения, что именно за данные поступают с этого радиомодуля; при этом у множества радиомодулей, выполняющих одну и ту же задачу, может быть один и тот же APPID64.

– EUI64 - идентификационный номер радиомодуля. В пределах одной сети каждый радиомодуль должен иметь уникальный адрес;

– JOINKEY - ключ шифрования, используемый в данной сети. JOINKEY должен быть абсолютно одинаковым для всех устройств сети, а также для базовой станции.

Ключ имеет разрядность 128 бит и задаётся в виде шестнадцатиричного числа (всего 32 символа);

– DATARATE - скорость передачи данных (dr 0 соответствует минимальной скорости и SF=12, dr 7 — максимальной скорости; дальность связи уменьшается с увеличением скорости);

Зависимость режима работы от скорости передачи показана в таблице 3.

Таблица 3

Режим работы	Скорость передачи
DRO	292 бит/с
DR1	537 бит/с
DR2	976 бит/с
DR3	1757 бит/с
DR4	3125 бит/с
DR5	5468 бит/с
DR6	10937 бит/с

– MAXRETR - максимальное количество попыток передать данные;

– PERIOD - период опроса датчиков, задается в секундах.

Через 30 секунд после подачи питания на модем новый сеанс устанавливается по беспроводной сети.

Посмотреть список параметров можно в любой момент времени работы модуля задав в терминале команду lsmod.

Конфигурация радиомодуля может быть сброшена командой `clear` с параметрами `all` или `key`.

– clear all сбрасывает всю конфигурацию модуля, включая ключи шифрования и EUI64

– clear keys сбрасывает только ключ шифрования.

Сброс конфигурации радиомодуля и соответствующее ответное сообщение показано на рисунке 22.

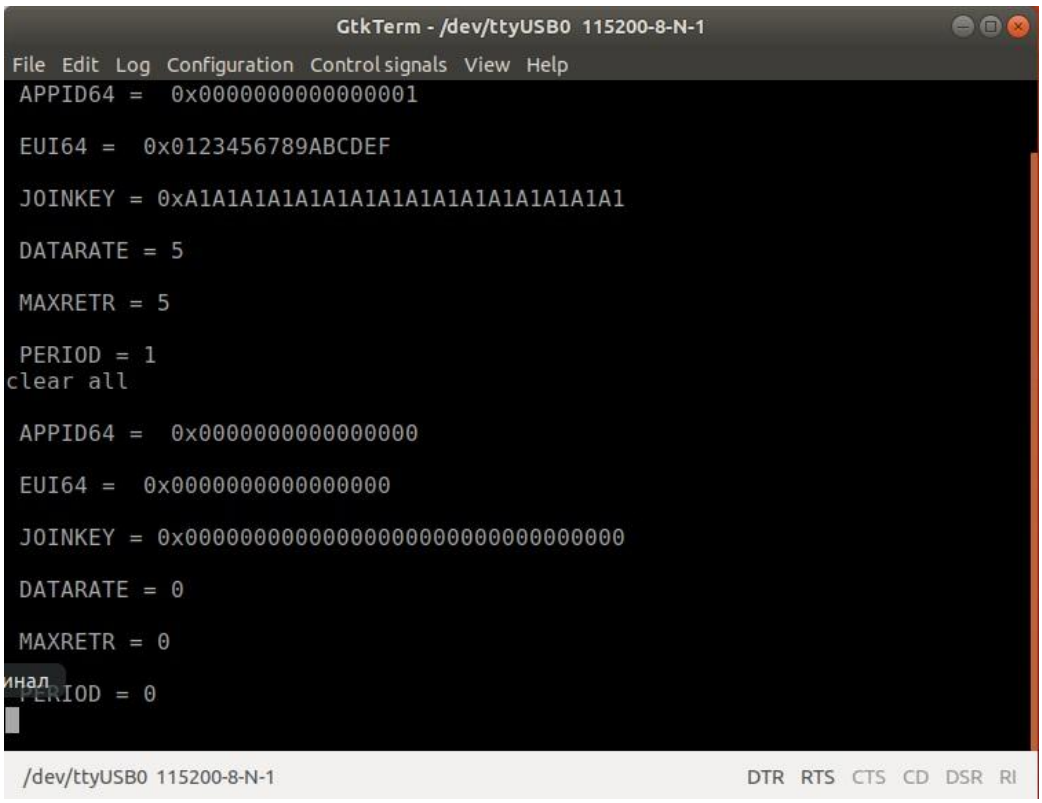

Рисунок 22 − Сброс конфигурации радиомодуля и соответствующее ответное сообщение

Установка параметров осуществляется командой set. Установка параметров подтверждается соответствующим ответным сообщением. Список команд представлен в таблице 4.

Таблица 4

Назначение	Команда	Ответное сообщение
Установка уникального идентификатора приложения	set appid <value>	APPID = <value>
Установка идентификационного номера радиомодуля	set eui <value>	EUI64 = <value>
Установка ключа шифрования	set key <value>	JOINKEY = <value>
Установка скорости передачи данных	set dr <value>	DATARATE = <value>
Установка максимального количества попыток передать данные	set maxretr <value>	MAXRETR = <value>
Установка периода опроса датчиков	set period <value>	PERIOD = <value>

Продолжение таблицы 4

Назначение	Команда	Ответное сообщение
Сохранение обновленной конфигурации	save	========SAVE=========
Моментальная отправка сообщения шлюзу	send	sending frame

После установки параметров необходимо сохранить обновленную конфигурацию командой save.Ответное сообщение показано на рисунке 23. После этого модуль начинает пытаться присоединиться к сети.

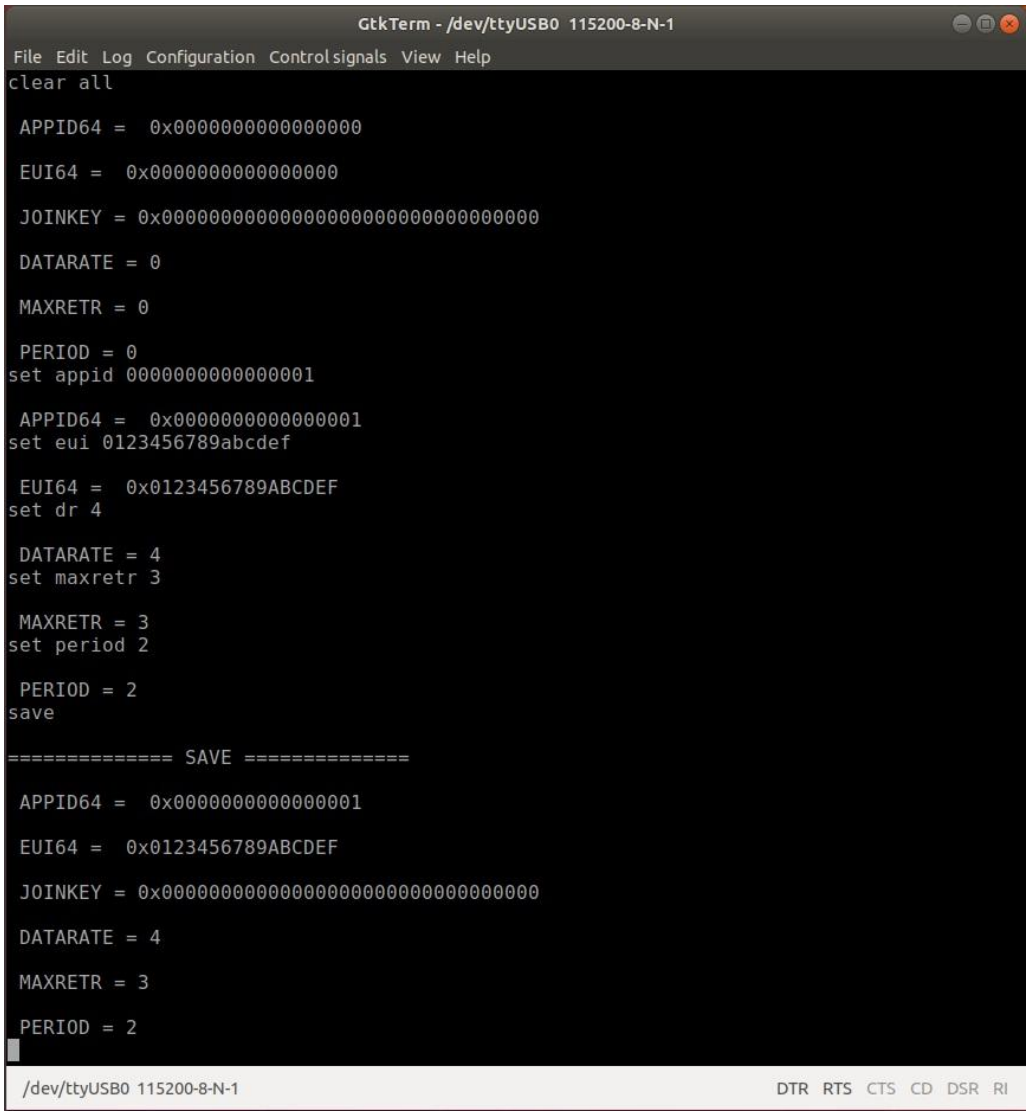

Рисунок 23 – обновление конфигурации

4.4 Отправка данных шлюзу

После активации устройства в сети конечное устройство с выбранным периодом считывает данные с датчика и отправляет шлюзу. Предусмотрена возможность моментальной отправки сообщения шлюзу при помощи команды send, как показано на рисунке 24.

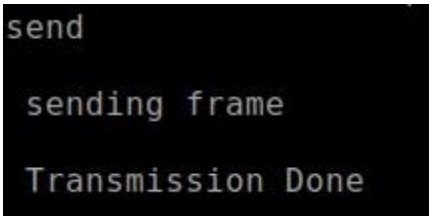

Рисунок 24 − Команда на отправку данных шлюзу

5 Применение программно-аппаратного комплекса в проекте «Умная урна»

Одними из главных точек притяжения людей в крупных городах являются городские парки, куда люди убегают от транспортного шума и суеты в поисках отдыха и новых развлечений. Однако, массовое скопление людей – это всегда проблема мусора.

Ключевой проблемой в управлении отходами заключается в том, что мусорное ведро в общественных местах заблаговременно переполняется до начала следующего процесса очистки. Кроме того, как правило, урны никак не закрепляются и имеют переворачивающуюся конструкцию, что приводит к возможности опрокидывания и вандализма.

При несвоевременной утилизации мусора появляются мини – свалки. Такие мини – свалки наносят вред окружающей среде, приводит к антисанитарии, может нанести вред здоровью человека и создает большую опасность для животных.

Стоит отметить еще одну проблему. Непотушенная сигарета или поджог урны в чертах лесополосы может иметь драматические последствия для окружающей среды (так как, это может вызвать лесной пожар).

Таким образом, в местах скопления людей наиболее остро стоит проблема контроля урн в лесопарковои зоне.

Традиционный способ ручного мониторинга отходов в мусорных баках - это громоздкий процесс, который требует больше человеческих усилий, времени и затрат, которых можно легко избежать с помощью современных технологий. Наиболее удачным решением вышеуказанных проблем является использование IOT технологий.

«УМНАЯ УРНА» является индивидуальным учебным проектом, выполненным в рамках курса “IOT Академия Sumsung”.Целью данного проекта является разработка IoT системы контроля урн в лесопарковои зоне.

Предлагается установка на урны датчиков и объединение урн с датчиками в единую сеть, позволяющую контролировать состояние урн.

Для реализации данного проекта использовался программно-аппаратный комплекс, разрабатываемый в рамках магистерской выпускной квалификационной работы.

В данной главе приведено краткое описание данного проекта, в качестве демонстрации примера применения разрабатываемого программно-аппаратного комплекса.

4.5 Архитектура системы

Данный проект представляет собой IoT систему, включающую в себя следующие компоненты:

- конечное устройство;
- базовая станция, предназначенная для разворачивания сети LoRaWAN
- сетевой сервер IOT Vega Server
- пользовательское веб приложение.

Оконечные устройства устанавливаются под дно урны. Каждое устройство имеет уникальный идентификатор, чтобы была возможность распознания расположения урны в парке. Конечное устройство считывает данные с датчиков с интервалом 15 минут и передают по беспроводной связи LoRaWAN базовой станции. Базовая станция передает принятые данные с узлов на сетевой сервер. В качестве сетевого сервера используется готовое решение от компании Vega – IoT Vega Server. Открытый API, основанный на технологии Web-Socket позволил подключить к IOT Vega Server внешнее веб приложение.

Архитектура системы IoT показана на рисунке 25.

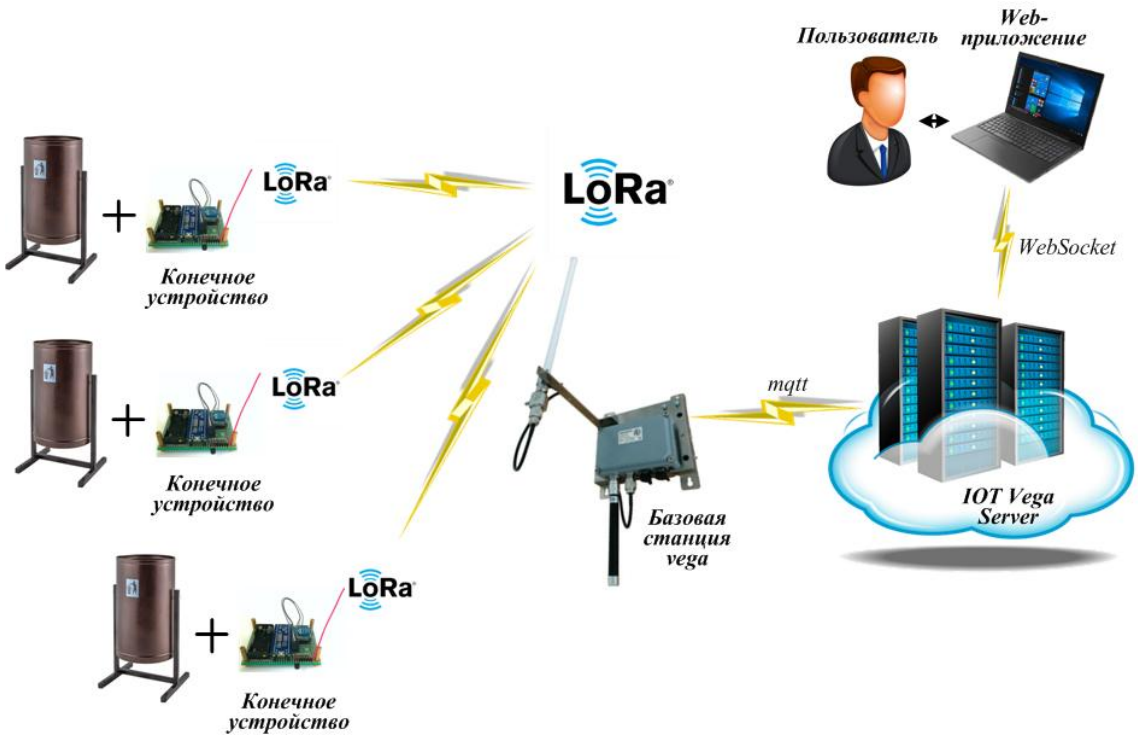

Рисунок 25 − Архитектура системы IoT

4.6 Оконечное устройство системы контроля урн

На рисунке 26 представлена структура оконечного устройства системы контроля урн. Устройство имеет автономное питание от двух батареек типа AAA.

Конечное устройство включает в себя:

− микроконтроллер, который отвечает за считывание измеряемых данных с датчиков, а так же реализацию стека протокола LoRaWAN;

− LoRaWAN-трансивер [11], который состоит из приемопередатчика, реализованного на базе технологии LoRa, антенного тракта и непосредственно самой антенны. Благодаря радиотрансиверу осуществляется радиосвязь с LoRaWAN шлюзом, а также подключение конечного устройства к LoRaWAN сети;

− периферию. Для проекта использовались следующие датчики:

1) Датчик приближения измеряет степень заполнения урны.

2) Датчики гироскопа и акселерометра могут замечать внезапные движения или тряску, что автоматически активирует оповещения о вандализме [12].

3) Датчик температуры может ощутить внезапное повышение температуры в урне, что активирует пожарную сигнализацию [13].

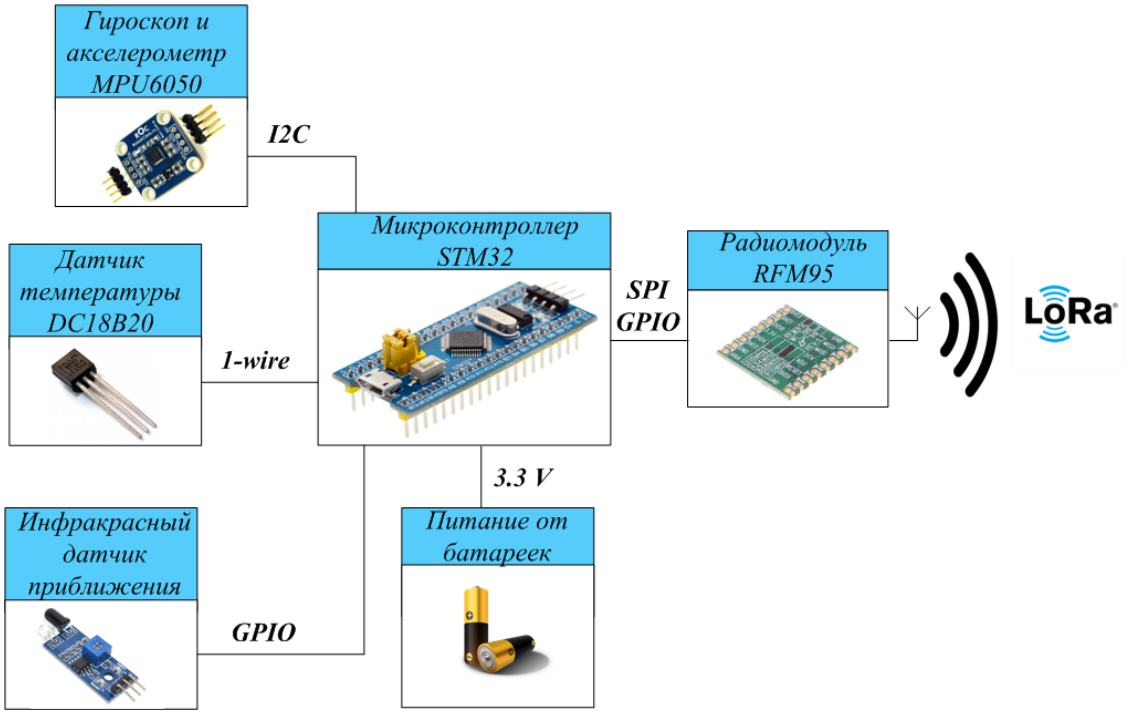

Рисунок 26 – Оконечное устройство системы контроля урн

Для взаимодействия с датчиком гироскопа и акселерометра MPU6050 используется интерфейс I2C, для взаимодействия с датчиком температуры DC18B20 используется интерфейс 1wire.Для считывания значения с датчика приближения используется вывод GPIO микроконтроллера. Взаимодействие с LoRaWAN-трансивером осуществляется через интерфейс SPI и выводы GPIO микроконтроллера.

4.7 Пользовательский интерфейс

Для работы оператора разработано пользовательское Web – приложение. Внешний вид пользовательского интерфейса показан на рисунке 27.

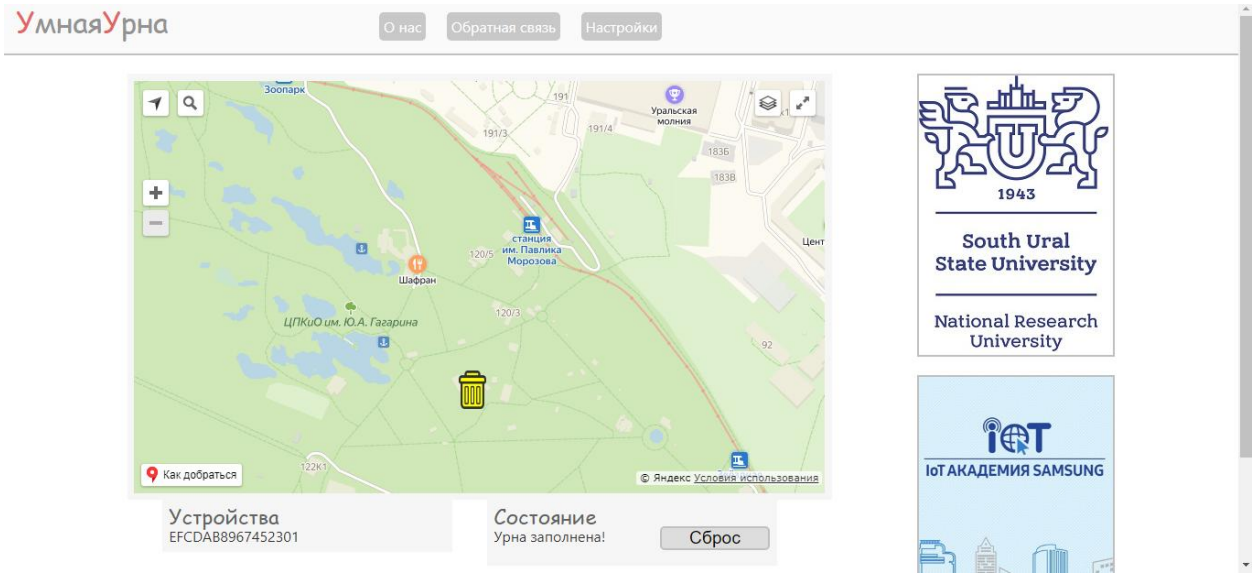

Рисунок 27 – Пользовательский интерфейс

Пользовательский интерфейс состоит из карты с отмеченными метками месторасположения урн, списка подключённых устройств и их состояния. Строка состояния и цвет урны меняются с изменением состояния урны, что позволяет оператору отслеживать такие ситуации как, переворот, заполнение или возгорание урны. Таким образом, в строке состояние отображается один из 4х возможных состояний урны:

– Все впорядке. В данном состоянии урна окрашена в зеленый цвет.

– Урна горит. В данном состоянии урна окрашена в красный цвет.

– Урна перевернута/горит. В данном состоянии урна окрашена в желтый цвет.

– Урна заполнена. В данном состоянии урна окрашена в желтый цвет.

При наведении курсора на урну, появляется окно со значением температуры. Предусмотрена кнопка, которую оператор может использовать для сброса состояния «Урна перевернута/упала», после проверки состояния урны.

Предусмотрена возможность добавлять устройства и устанавливать их месторасположение на карте.

4.8 Расчет энергопотребления

В данном устройстве в качестве автономного источника питания применяется батарейки типа AAA. Нами были использованы обычное щелочные батарейки с емкостью 1200 мАч, которые полностью обеспечивают работоспособность прототипа.

4.8.1 Расчет времени передачи пакетов через интерфейс

Каждый пакет, передаваемый по сети LoRaWAN, включает в себя преамбулу и блок данных физического уровня.

Кол-во символов в преамбуле в данном проекте $n_{preamble} = 6$ байт.

Количество символов в блоке данных физического уровня определяется следующей формулой

$$payloadSymbNB = ceil\left(\frac{8 \cdot PL - 4 \cdot SF + 28 + 16 \cdot CRC - 20 \cdot H}{4 \cdot (SF - 2 \cdot DE)}\right) \cdot (CR + 4),$$

где

$PL = 12 + FRM = 12 + 4 = 16 -$ количествово байт полезных данных в блоке физического уровня (PHYPayload);

$FRM = 4 -$ количество байт полезных данных на уровне приложения (FRMPayload);

$SF = 7 -$ коэффициент расширения спектра;

$CRC = 1 -$ скорость кода;

$H = 0 -$ передача заголовка (PHDR + PHDR_CRC) включена;

$DE = 0 -$ оптимизация для низких скоростей передачи выключена;

$ceil -$ операция округления до ближайшего целого числа.

С учетом вышесказанных значений, вычислим количество символов в блоке данных физического уровня.

$$payloadSymbNB = ceil\left(\frac{8 \cdot 16 - 4 \cdot 7 + 28 + 16 \cdot 1 - 20 \cdot 0}{4 \cdot (7 - 2 \cdot 0)}\right) \cdot (1 + 4) = 18 \text{ байт}$$

Длительность передачи преамбулы

$$T_{preamble} = (n_{preamble} + 4{,}25) \cdot T_{sym} = (6 + 4{,}25) \cdot 1{,}024 = 10{,}496 \text{ мс},$$

где $T_{sym} = 1{,}024$ мс – длительность передачи одного символа;

Длительность передачи блока данных физического уровня

$$T_{payload} = payloadSymbNB \cdot T_{sym} = 18 \cdot 1{,}024 = 18{,}432 \text{ мс}$$

Длительность передачи всего пакета по сети LoRaWAN

$$T_{packet} = T_{payload} + T_{preamble} = 18{,}432 + 10{,}496 = 28{,}928 \text{ мс}$$

Передача данных по сети LoRaWAN осуществляется каждые 15 минут (900 секунд), тогда в процентном соотношении радиотрансивер находится в режиме передачи данных $\frac{0{,}028}{900} \cdot 100$ =0,003 %, остальное время радиотрансивер находится в режиме сна.

4.8.2 Расчет времени автономной работы устройства

Ток потребления рассчитывается исходя из потребления датчиков, радиотрансивера и микроконтроллера. Устройство работает в 2х режимах, режим сна, с уменьшенным током потребления и активным режимом, в котором происходит опрос датчиков, передача сообщения радиотрансиверу.

Потребление тока питания приведены в таблице 5.

Таблица 5

Оборудование	Режим	Потребляемый ток	Время работы, %
STM32F103C8	Активный режим	2,57 мА	0,57
	Режим сна	0,7576 мА	99,43
RFM95w	Передача данных	29 мА	0,003
	Режим сна	0,2 мкА	99,997
MPU6050	Активный режим	3,9 мА	0,478
	Режим сна	5 мкА	99,522
DC18B20	Активный режим	1 мА	0,07
	Режим сна	790 нА	99,93

Средний ток потребления всего устройства высчитывается путем суммирования потребляемого тока всех компонентов с учетом процентного соотношения времени работы в режиме сна и в активном режиме .

$$I_{\text{ср}} = 0{,}0057 \cdot 2{,}57 + 0{,}9943 \cdot 0{,}7576 + 0{,}00003 \cdot 29 + 0{,}99997 \cdot 0{,}2 \cdot 10^{-3} +$$
$$0{,}00478 \cdot 3{,}9 + 0{,}99522 \cdot 5 \cdot 10^{-3} + 0{,}0007 \cdot 1 + 0{,}9993 \cdot 0{,}79 \cdot 10^{-3} = 0{,}8 \text{ мА}$$

При использовании двух щелочных батареек с емкостью $\text{С} = 1200$ мАч, время автономной работы устройства

$$t_{\text{авт}} = \frac{\text{С} \cdot 2}{I_{\text{ср}}} = \frac{1200 \cdot 2}{0{,}8} = 3000 \text{ часов} = 125 \text{ дней}$$

Таким образом, время автономной работы устройства составляет 4 месяца.

4.9 Выводы

В данной главе представлен пример применения программно-аппаратный комплекса, разработанного в рамках магистерской ВКР.

«УМНАЯ УРНА» является индивидуальным учебным проектом, выполненным в рамках курса “IOT Академия Sumsung”. Целью данного проекта является разработка IoT системы контроля урн в лесопарковои зоне.

В данном проекте проработаны все уровни системы интернета вещей от конечного устройства до пользовательского интерфейса. Время автономной работы устройства составляет 4 месяца.

Для реализации конечного устройства использовался разрабатываемый программно-аппаратный комплекс.

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе был разработан программно-аппаратный лабораторный комплекс для изучения технологии передачи данных в Интернете Вещей. Данная разработка представляет собой универсальное конечное устройства интернета вещей, совместимое со стандартом LoRaWAN и уникальным протоколом unwired devices.

В ходе работы, был исследован вариант протокола MAC уровня от Unwired Device. Особое внимание было уделено исследованию формата MAC пакета, времени запуска и длительности временного окна приема, безопасности в сетях и активации конечных устройств в сети. Выполнено сравнение со спецификацией LoRaWAN и выявлены основные особенности данного протокола.

Было разработано программное обеспечение на основе библиотеки LMIC с открытым исходным кодом. Предусмотрена связь с пользователем через последовательный интерфейс UART. С помощью команд высокого уровня пользователь может выбрать, согласно какому протоколу, устройство должно работать, настроить параметры работы в сети. Через последовательный интерфейс устройство информирует пользователя об определенных изменениях состояния внутри модема.

Разработанное устройство было успешно протестировано в проекте, выполненном в рамках курса “IOT Академия Sumsung”.

СПИСОК ИСПОЛЬЗОВАННЫХ ИНСТОЧНИКОВ

1. Кучерявый А. Е. Интернет Вещей // Электросвязь. 2013. № 1. С. 21–24.
2. Кучерявый А. Е., Кучерявый Е. А., Прокопьев А. В. Самоорганизующиеся сети. СПб.: Любавич. 2011. 312 с.
3. Киричек Р. В., Парамонов А. И., Прокопьев А. В., Кучерявый А. Е. Эволюция исследований в области беспроводных сенсорных сетей // Информационные технологии и телекоммуникации.
4. The Future of Jobs. Employment, Skills and Workforce strategy for The Fourth Industrial Revolution : report [Электронный ресурс]. 2016. January. URL: http://www3.weforum.org/docs/WEF_Future_of_Jobs.pdf (дата обращения: 27.01.2016).
5. Всемирное исследование Digital IQ за 2017 год: десятое, юбилейное издание. [Электронный ресурс].2017. URL: https://www.pwc.ru/ru/publications/global-digital-iq-survey-rus.pdf
6. IoTАкадемия Samsung [Электронный ресурс].https://www.samsung.com/ru/iotacademy/project/
7. LoRaWAN™ Specification, N.Sornin (Semtech), M.Luis (Semtech), T.Eirich (IBM), T.Kramp (IBM), O.Hersent (Actility), V1.0, 2015 January.
8. LoRaWAN Regional Parameters, RU 864-869MHz ISM Band.
9. https://itechinfo.ru/content/Обзор-технологии-lora
10. https://unwds.ru/docs/#unwired-range
11. RFM95/96/97/98(W) - Low Power Long Range Transceiver Module V1.0. Datasheet
12. MPU-6000 and MPU-6050 Product Specification Revision 3.4
13. Programmable Resolution 1-Wire Digital Thermometer

ПРИЛОЖЕНИЕ А

Из-за того, что сам код достаточно объёмный и занимает много страниц, в приложении изложено описание основных файлов.

main.c

```
#include "main.h"

/********** Первая задача Инициализация и запуск ************/
static void initfunc (osjob_t* j) {
        // Сброс MAC
        LMIC_reset();
        modem_init();
}
/************************** main ***************************/
int main () {
        PERSIST.library = Unwired_Device;          //Unwired_Device/ IBM;
        os_getArtlibrary(&LMIC.library);
        InitRCC(); // Настройка SysTick
  osjob_t initjob;
        //инициализация планировщика задач
  os_init();
        //инициализация библиотеки debug
  debug_init();
  //задаем первую задачу
  os_setCallback(&initjob, initfunc);
  //Запуск выполнения запланированных задач
  os_runloop();
  return 0;
}
```

main.h

```
/************************ library ************************/
#include "stm32f10x.h"
#include "HAL.h"
#include "debug.h"
#include "modem.h"

/************************ defines ************************/
#define maxRetr 5 // максимальное колличество ретрансляций
```

sysInit.c

```
/************************ library ************************/
```

```
#include "sysInit.h"
/********************* Use function *********************/
void InitRCC (void){

	RCC->CR |= ((uint32_t)RCC_CR_HSEON);		// Вкл HSE
	while (!(RCC->CR & RCC_CR_HSERDY));		// Ожидание старта HSE
	FLASH->ACR = FLASH_ACR_PRFTBE | FLASH_ACR_LATENCY;  //		Тактирование
Flash памяти
	RCC->CFGR |= RCC_CFGR_HPRE_DIV2;		// AHB = SYSCLK/2
	RCC->CFGR |= RCC_CFGR_PPRE1_DIV1;		// APB1 = HCLK/1
	RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;		// APB2 = HCLK/1
	RCC->CFGR &= ~RCC_CFGR_PLLMULL;		// стереть PLLMULL биты
	RCC->CFGR &= ~RCC_CFGR_PLLSRC;		// стереть PLLSRC биты
	RCC->CFGR &= ~RCC_CFGR_PLLXTPRE;		// стереть PLLXTPRE биты
	RCC->CFGR |= RCC_CFGR_PLLSRC_HSE;		// источник HSE
	RCC->CFGR |= RCC_CFGR_PLLXTPRE_HSE;	// источник HSE = 8 MHz
	RCC->CFGR |= RCC_CFGR_PLLMULL6;		// PLL x6: clock = 8 MHz * 6 = 48
	RCC->CR |= RCC_CR_PLLON;			// Вкл PLL
	while((RCC->CR & RCC_CR_PLLRDY) == 0) {}	// ждем PLL
	RCC->CFGR &= ~RCC_CFGR_SW;			// стереть SW биты
	RCC->CFGR |= RCC_CFGR_SW_PLL;		// выбрать источник SYSCLK = PLL
	while((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_1) {}
}

void InitGPIO (void){
	/*			Output GPIO
		RXTX		-- PA 4 : (выход двухтактный, общего назначения,2MHz)
				-- PC 13 : (выход двухтактный, общего назначения,2MHz)
		RST		-- PA 2  : (в третьем состоянии) */

	RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;	// тактирование порта A
	RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;	// тактирование порта C
	// RXTX
	GPIOA->CRL   &= ~GPIO_CRL_MODE4;
	GPIOA->CRL   &= ~GPIO_CRL_CNF4;
	GPIOA->CRL   |=  GPIO_CRL_MODE4_1;
	// RST
	GPIOA->CRL   &= ~GPIO_CRL_MODE2;
	GPIOA->CRL   &= ~GPIO_CRL_CNF2;
	GPIOA->CRL   |=  GPIO_CRL_CNF2_0;
}
```

sysInit.h

```
/************************ library ************************/
#include "stm32f10x.h"
/********************* Use function *********************/

void InitRCC (void);
```

```
void GenMCO (void);
void InitGPIO (void);
void InitLED (void);
```

libUART.c

```
/************************ library ************************/
#include "libUART.h"
#include "main.h"

/************************ defines ************************/
#define USART           USART1
#define USART_TX_PORT    GPIOA
#define USART_TX_PIN     9
#define USART_RX_PORT    GPIOA
#define USART_RX_PIN     10
#define USART_AF         0x07
#define USART_RCCREG     APB2ENR
#define USART_RCCVAL     RCC_APB2ENR_USART1EN
#define USART_IRQN       USART1_IRQn
#define USART_IRQHANDLER  USART1_IRQHandler

/********************* Use function **********************/

void InitUART (void){

        //USART_InitTypeDef USART_InitStructure;
        GPIO_InitTypeDef GPIO_InitStructure;

        RCC->APB2ENR    |=    RCC_APB2Periph_GPIOA    |    RCC_APB2Periph_GPIOB    |
RCC_APB2Periph_AFIO ;
        RCC->APB2ENR |=  RCC_APB2Periph_USART1;

        //Tx on PA9 as alternate function push-pull
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_Init(GPIOA, &GPIO_InitStructure);

  //Rx on PA10 as input floating
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOA, &GPIO_InitStructure);

        USART1->BRR = 0x00D0;

        //Включаем прием и передачу TE и RE в регистре CR1.
        //Тут же включаем работу UART-а в бите UE
```

```
	USART1->CR1 |= USART_CR1_TE;
	USART1->CR1 |= USART_CR1_UE;

	//Разрешаем прерывание по приему данных битом RXNEIE

	USART1->CR1 |= USART_CR1_RXNEIE;
	NVIC_EnableIRQ(USART1_IRQn);
}
 state_tt state;

void usart_startrx () {
   USART->CR1 |= (USART_CR1_RE | USART_CR1_RXNEIE);
}

/********************** Interupt ************************/
static char  uart1_rx_buf[80];
uint8_t rxnum;

void USART_IRQHANDLER (void) {
hal_disableIRQs();
   // прерывание по приему
if (USART1->SR & USART_SR_RXNE){ // Убедились , что прерывание по приему
	USART1->SR &= ~USART_SR_RXNE;		// сбрасываем флаг прерывания
	uart1_rx_buf[rxnum] = USART1->DR;
	rxnum++;
	if (uart1_rx_buf[rxnum-1]== '\r'){
		if(strcmp(uart1_rx_buf,"lscfg\r")==0){
		lscfg();
		}
	else	switch(state){
		case s0:{
				if(strncmp(uart1_rx_buf,"0",1)==0){
					PERSIST.library = Unwired_Device;
					modem_init2();
					break;
				}
				else if (strncmp(uart1_rx_buf,"1",1)==0){
					PERSIST.library = IBM;
					modem_init2();
					break;
				}
				else {
					debug_str("\r\n error \r\n");
					break;
				}
		}
		case s1 : {
				if(strncmp(uart1_rx_buf,"clear key",9)==0){
					memset(&PERSIST.joinkey, 0, 16);
					os_clearCallback(&LMIC.osjob);
					debug_str("\r\n JOINKEY = 0x");
```

```
                        debug_buf(PERSIST.joinkey,16);
                        state = s2;
                    break;
                    }
                    else if(strncmp(uart1_rx_buf,"clear all",9)==0){
                        memset(&PERSIST.joinkey, 0, 16);
                        memset(&PERSIST.appeui, 0, 8);
                        memset(&PERSIST.deveui, 0, 8);
                        PERSIST.Datarate = 0;
                        PERSIST.max_retr = 0;
                        PERSIST.period = 0;
                        os_clearCallback(&LMIC.osjob);
                        lscfg();
                        state = s3;
                    break;
                    }
                    else {
                        debug_str("\r\n error \r\n");
                    break;
                    }
            }

            case s2 : {
                    if(strncmp(uart1_rx_buf,"set key",7)==0){
                        uint8_t joinkey[32];
                        memcpy(joinkey, (&uart1_rx_buf[8]), 32);
                        gethex(PERSIST.joinkey,joinkey , 32);
                        debug_str("\r\n JOINKEY = 0x");
                        debug_buf(PERSIST.joinkey,16);
                        state = s2;
                    break;
                    }
                    if(strncmp(uart1_rx_buf,"save",4)==0){
                        debug_str("\r\n==============   SAVE
==============\r\n");
                        modem_init();
                        state = s1;
                    break;
                    }

                    else {
                        debug_str("error\r\n");
                    break;
                    }
            }

            case s3 : {

                    if(strncmp(uart1_rx_buf,"set key",7)==0){
                        uint8_t joinkey[32];
                        memcpy(joinkey, (&uart1_rx_buf[8]), 32);
```

```
        gethex(PERSIST.joinkey,joinkey , 32);
        debug_str("\r\n JOINKEY = 0x");
        debug_buf(PERSIST.joinkey,16);
        state = s3;
break;
}


else if(strncmp(uart1_rx_buf,"set appid",9)==0){
        uint8_t APPID[16];
        memcpy(APPID, (&uart1_rx_buf[10]), 16);
        gethex(PERSIST.appeui,APPID , 16);
        debug_str("\r\n APPID64 =  0x");
        debug_buf(PERSIST.appeui,8);
        state = s3;
break;
}
else if(strncmp(uart1_rx_buf,"set eui",7)==0){
        uint8_t EUI[16];
        memcpy(EUI, (&uart1_rx_buf[8]), 16);
        gethex(PERSIST.deveui,EUI , 16);
        debug_str("\r\n EUI64 =  0x");
        debug_buf(PERSIST.deveui,8);
        state = s3;
break;
}
else if(strncmp(uart1_rx_buf,"set class",9)==0){
        char buf = uart1_rx_buf[10];
        if(strcmp(&buf,"a")==0){
                PERSIST.nodeClass = LS_ED_CLASS_A;
        }
        else if(strcmp(&buf,"b")==0){
                PERSIST.nodeClass = LS_ED_CLASS_B;
        }
        else if(strcmp(&buf,"c")==0){
                PERSIST.nodeClass = LS_ED_CLASS_C;
        }
        else{
                debug_str("error\r\n");
        }
 debug_str("\r\n CLASS = ");
 switch(PERSIST.nodeClass){
        case LS_ED_CLASS_A : {
                debug_str("A");
        break;
        }
        case LS_ED_CLASS_B : {
         debug_str("C");
        break;
        }
        case LS_ED_CLASS_C : {
```

```
                        debug_str("C");
                    break;
                    }
                }
                debug_char('\r');
                debug_char('\n');
                state = s3;
            break;
            }
            else if(strncmp(uart1_rx_buf,"set dr",6)==0){
                    uint8_t buf;
                    uint8_t Datarate;
                    Datarate = 0;
                    buf = uart1_rx_buf[7];
                    gethex(&Datarate,&buf, 2);
                    if (Datarate > 7){
                                debug_str("error\r\n");
                    }
                    else{
                    PERSIST.Datarate = Datarate;
                    debug_valdec("\r\n DATARATE = " ,PERSIST.Datarate);
                    debug_char('\r');
                    debug_char('\n');
                    state = s3;
                    }
            break;
            }

            else if(strncmp(uart1_rx_buf,"set maxretr",11)==0){
                    uint8_t buf;
                    uint8_t maxretr;
                    maxretr = 0;
                    buf = uart1_rx_buf[12];
                    gethex( &maxretr,&buf, 2);
                    PERSIST.max_retr = maxretr;
                    debug_valdec("\r\n MAXRETR = " ,PERSIST.max_retr);
                    debug_char('\r');
                    debug_char('\n');
                    state = s3;
            break;
            }

        else if(strncmp(uart1_rx_buf,"set period",10)==0){
                    uint8_t buf;
                    uint8_t period;
                    period = 0;
                    buf = uart1_rx_buf[11];
                    gethex( &period,&buf, 2);
                    PERSIST.period = period;
                    debug_valdec("\r\n PERIOD = " ,PERSIST.period);
                    debug_char('\r');
```

```
                                        debug_char('\n');
                                        state = s3;
                                break;
                                }

                                if(strncmp(uart1_rx_buf,"save",4)==0){
                                        debug_str("\r\n==============   SAVE
==============\r\n");
                                        lscfg ();
                                        JOIN();
                                        state = s1;
                                break;
                                }

                                else {
                                        debug_str("error\r\n");
                                break;
                                }
                        }
                        case s4 : {
                                if(strncmp(uart1_rx_buf,"send",4)==0){
                                        READSensor();
                                        memset(&uart1_rx_buf, 0, rxnum); //Очищаем буфер
                                        rxnum=0; //Сбрасываем счетчик
                                }
                                else {
                                        debug_str("error\r\n");
                                break;
                                }
                        }
                }
                                memset(&uart1_rx_buf, 0, rxnum); //Очищаем буфер
                                rxnum=0; //Сбрасываем счетчик
        }
}
  hal_enableIRQs();
}
```

libUART.h

```
/************************ library ************************/
#include "stm32f10x.h"
/********************* Use function **********************/
#ifndef _libUART_h_
#define _libUART_h_
void InitUART (void);
void InitTerminal(void);
void usart_starttx (void);
void usart_startrx (void);
```

```
typedef enum {
		s0,
		s1,
		s2,
		s3,
		s4
} state_tt;
extern  state_tt state;
#endif // _libUART_h_
```

libTIM.c

```
/************************ library ************************/
#include "libTIM.h"
/********************* Use function **********************/
// TIMER 2 используется для обеспечения тактовых импульсов 32 кГц и для создания прерываний
void InitTIM2(void){
	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; //подаем тактирование на TIM6
	TIM2->PSC = 750 - 1;				// Настраиваем делитель
	TIM2->ARR = 65535 ;				// 32 kHz
	TIM2->DIER |= TIM_DIER_UIE;			//разрешаем прерывание от таймера
	TIM2->CR1 |= TIM_CR1_CEN;
	NVIC_EnableIRQ(TIM2_IRQn);			//Разрешение прерывания
	__enable_irq();
}
```

libTIM.h

```
/************************ library ************************/
#include "stm32f10x.h"
/********************* Use function **********************/
void InitTIM2(void);
```

libEXTI.c

```
/************************ library ************************/
#include "libEXTI.h"
/********************* Use function **********************/
/*
Input GPIO
				DIO 0		-- PB 1
				DIO 1		-- PB 10
				DIO 2		-- PB 11
DIO1 требуется для LoRa, DIO2 для FSK
DIO0, DIO1 и DIO2 должны быть сконфигурированы так, чтобы инициировать прерывание по нарастающему фронту
*/
```

```
/*****************************Use function*************************************/

void InitEXTI_DIO0 (void){
	//включаем тактирование порта и альтернативных функций
	RCC->APB2ENR |= RCC_APB2ENR_AFIOEN | RCC_APB2ENR_IOPBEN;
	//настраиваем вывод DIO 0 на вход с подтяжкой
	GPIOB->CRL &= ~GPIO_CRL_CNF1;
	GPIOB->CRL |= GPIO_CRL_CNF1_1;      //  PB0 CRL [1:0] ={1,0}
	GPIOB->CRL &= ~GPIO_CRL_MODE1;  //  PB0 MODE [1:0] ={0,0}
	//подтягиваем к питанию, сбросили бит  в ODR включив PULL DOWN.
	GPIOB->BSRR |= GPIO_BSRR_BR0;
	//выбор порта и пина для внешнего прерывания
	 AFIO->EXTICR [0] |= AFIO_EXTICR1_EXTI1_PB;
	//по возврастающему фронту
	 EXTI->RTSR |= EXTI_RTSR_TR1;
	//устанавливаем маску
	 EXTI->IMR |= EXTI_IMR_MR1;
	// Функции CMSIS разрешающие прерывания в NVIC
	NVIC_EnableIRQ (EXTI1_IRQn);
}

void InitEXTI_DIO1 (void){
	//настраиваем вывод DIO 1 на вход с подтяжкой
	GPIOB->CRH &= ~GPIO_CRH_CNF10;
	GPIOB->CRH |= GPIO_CRH_CNF10_1;    //  PB 10 CRL [1:0] ={1,0}
	GPIOB->CRH &= ~GPIO_CRH_MODE10;//  PB 10 MODE [1:0] ={0,0}
	//подтягиваем к питанию, сбросили бит  в ODR включив PULL DOWN.
	GPIOB->BSRR |= GPIO_BSRR_BR10;
	//выбор порта и пина для внешнего прерывания
	 AFIO->EXTICR [2] |= AFIO_EXTICR3_EXTI10_PB ;
	  //по возрастающему фронту
	 EXTI->RTSR |= EXTI_RTSR_TR10;
	// разрешаем прерывания в периферии
	EXTI->IMR |= EXTI_IMR_MR10;
	// Функции CMSIS разрешающие прерывания в NVIC
	NVIC_EnableIRQ (EXTI15_10_IRQn); //
}

void InitEXTI_DIO2 (void){
	//настраиваем вывод DIO 2 на вход с подтяжкой
	GPIOB->CRH &= ~GPIO_CRH_CNF11;
	GPIOB->CRH |= GPIO_CRH_CNF11_1;    //  PB 10 CRL [1:0] ={1,0}
	GPIOB->CRH &= ~GPIO_CRH_MODE11;          //  PB 10 MODE [1:0] ={0,0}
	//подтягиваем к питанию, сбросили бит  в ODR включив PULL DOWN.
	GPIOB->BSRR |= GPIO_BSRR_BR11;
	//выбор порта и пина для внешнего прерывания
	 AFIO->EXTICR [2] |= AFIO_EXTICR3_EXTI11_PB ;
	 //по возрастающему фронту
	 EXTI->RTSR |= EXTI_RTSR_TR11;
	// разрешаем прерывания в периферии
	EXTI->IMR |= EXTI_IMR_MR11;
```

```
}
```

libEXTI.h

```
/************************ library ************************/
#include "stm32f10x.h"
/********************* Use function **********************/
void InitEXTI_DIO0(void);
void InitEXTI_DIO1 (void);
void InitEXTI_DIO2 (void);
```

libSPI.c

```
/************************ library ************************/
#include "libSPI.h"
/********************* Use function **********************/
/*-------------------------------------------------------------------------
Инициализация SPI1 -> установка SPI2 в режим мастера 16bit
        SCK    -- PA5 : выход двухтактный, альтернативная функция, 50MHz
        MISO  -- PA6 : вход цифровой с подтягивающим резистором, подтяжка к плюсу
        MOSI  -- PA7 : выход двухтактный, альтернативная функция, 50MHz
        NCC    -- PB 0 (выход двухтактный, общего назначения,50MHz)
*-------------------------------------------------------------------------*/
void InitSPI (void){
        RCC->APB2ENR |=  RCC_APB2ENR_AFIOEN; //включить тактирование альтернативных
функций       /
        RCC->APB2ENR |=  RCC_APB2ENR_IOPBEN; //включить тактирование порта B
        RCC->APB2ENR |=  RCC_APB2ENR_IOPAEN; //включить тактирование порта A
        // управление NSS
        GPIOB->CRL   |=  GPIO_CRL_MODE0;
        GPIOB->CRL   &= ~GPIO_CRL_CNF0;
        GPIOB->BSRR   =  GPIO_BSRR_BS0;
        // SCK
        GPIOA->CRL   |=  GPIO_CRL_MODE5;
        GPIOA->CRL   &= ~GPIO_CRL_CNF5;
        GPIOA->CRL   |=  GPIO_CRL_CNF5_1;
        // MOSI
        GPIOA->CRL   &= ~GPIO_CRL_MODE6;
        GPIOA->CRL   &= ~GPIO_CRL_CNF6;
        GPIOA->CRL   |=  GPIO_CRL_CNF6_1;
        GPIOA->BSRR   =  GPIO_BSRR_BS6;
         // MISO
        GPIOA->CRL   |=  GPIO_CRL_MODE7;
        GPIOA->CRL   &= ~GPIO_CRL_CNF7;
        GPIOA->CRL   |=  GPIO_CRL_CNF7_1;

        RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; //подать тактирование
        SPI1->CR1     = 0x0000;          //очистить первый управляющий регистр
        SPI1->CR2     = 0x0000;          //очистить второй управляющий регистр
```

```
	SPI1->CR1 &= ~SPI_CR1_DFF; // Бит11 Формат данных 0-8бит 1-16бит
	SPI1->CR1 |= SPI_CR1_SSM; // Бит9 SSM – выбирает источник сигнала NSS (0 — с внешнего вывода, 1 — программно);
	SPI1->CR1 |= SPI_CR1_SSI; // Бит8 SSI – если SSM = 1 определяет значение NSS;
	SPI1->CR1 &= ~SPI_CR1_LSBFIRST; // Бит7 LSBFIRST – задаёт способ передачи (0 - старшим, 1 — младшим разрядом вперёд);
	//Бит3-5 BR[2:0] — делитель скорости обмена fPCLK/x (000:2, 001:4, 010:8, 011:16, 100:32, 101:64, 110:128, 111:256)
	SPI1->CR1 |= SPI_CR1_BR_0; // Задать скорость fPCLK/4 = 24/4
	SPI1->CR1 |= SPI_CR1_MSTR; // MSTR - делает модуль ведущим(1)/ведомым(0);
	SPI1->CR1 &= ~SPI_CR1_CPOL; // CPOL - задаёт полярность тактового сигнала;
	SPI1->CR1 &= ~SPI_CR1_CPHA; // CPHA - задаёт фазу тактового сигнала 0-\ 1-/;
	SPI1->CR1 |= SPI_CR1_SPE; // Бит6 SPE - работа SPI (1 – вкл. 0 – откл.)
}
```

libSPI.h

```
/********************** library ************************/
#include "stm32f10x.h"
/******************** Use function **********************/
void InitSPI (void);
void InitDMASPI (void);
```

HAL.c

```
/********************** library ************************/
#include "HAL.h"
/******************** Use function **********************/

uint32_t tick;  // 32-битное системное время в тиках
uint8_t irqlevel; // для вкл/выкл прерываний
/* инициализация IO, SPI, TIMER, IRQ */
 void hal_init (void) {
	 hal_disableIRQs();
	 InitSPI ();
	 InitGPIO ();
	 InitEXTI_DIO0 ();
	 InitEXTI_DIO1 ();
	 InitEXTI_DIO2 ();
	 InitTIM2();
	 hal_enableIRQs();
}
// сбой
void hal_failed (void) {
	hal_disableIRQs();
	hal_sleep();
	while(1);
}
// запись в RXTX(PA4) val == 1  => tx 1
```

```
void hal_pin_rxtx (uint8_t val){
	GPIOA->ODR = (GPIOA->ODR    & ~(1 << 4))  | ((val & 1) << 4 );
}
//управление пином NSS (PB 0) (0 = низкий / выбранный, 1 = высокий / снят)
void hal_pin_nss (uint8_t val){
	GPIOB->ODR = (GPIOB->ODR    & ~(1 << 0))  | ((val & 1) << 0 );
}
/* установить RST(PA 2) на заданное значение
(0 = низкий, 1 = высокий, 2 = плавающие)*/
void hal_pin_rst (uint8_t val){
	if(val == 0 || val == 1) {
		GPIOA->CRL   &= ~GPIO_CRL_CNF2;
		GPIOA->CRL   |=  GPIO_CRL_MODE2_1;
		GPIOA->ODR = (GPIOA->ODR    & ~(1 << 2))  | ((val & 1) << 2 );
	}
	else {
		GPIOA->CRL   &= ~GPIO_CRL_MODE2;
		GPIOA->CRL   &= ~GPIO_CRL_CNF2;
		GPIOA->CRL   |=  GPIO_CRL_CNF2_0;
	}
}
//Выполнение 8-битной транзакции SPI.
uint8_t hal_spi (uint8_t outval){
	SPI1->DR = outval;
	while((SPI1->SR  & SPI_SR_RXNE)==0){}
	return SPI1->DR;
}
//Возвращает 32-битное системное время в тиках.
uint32_t hal_ticks (void){
	hal_disableIRQs();
  uint32_t t = tick;
  uint16_t cnt = TIM2->CNT;
  if( (TIM2->SR & TIM_SR_UIF)){
   cnt = TIM2->CNT;
   t++;
  }
  hal_enableIRQs();
	return((t<<16)|cnt);
}
static uint32_t delta_time(uint32_t time) {
 uint32_t t = hal_ticks();
 int32_t d = time - t;
 if( d<=0 ) return 0;    // in the past
 if( (d>>16)!=0 ) return 0xFFFF; // far ahead
 return (uint16_t)d;
}
// ожидание, пока tick не достигнет указанной отметки времени
void hal_waitUntil (uint32_t time){;
	while((delta_time(time))!= 0){}
}
```

```
uint8_t hal_checkTimer (uint32_t targettime){
        uint16_t dt;
  TIM2->SR &= ~TIM_SR_CC2IF; // clear any pending interrupts
        if((dt = delta_time(targettime)) < 5) { // event is now (a few ticks ahead)
  TIM2->DIER &= ~TIM_DIER_CC2IE; // disable IE
  return 1;
 }
        else { // rewind timer (fully or to exact time))
  TIM2->CCR2 = TIM2->CNT + dt;   // set comparator
  TIM2->DIER |= TIM_DIER_CC2IE;  // enable IE
  TIM2->CCER |= TIM_CCER_CC2E;   // enable capture/compare uint 2
  return 0;
 }
}

//Запрет всех прерывания процессора
void hal_disableIRQs (void){
        __disable_irq();
         irqlevel++;
}

//Разрешение на прерывания процессора
void hal_enableIRQs (void){
        if(--irqlevel == 0) {
        __enable_irq();
        }
}
// Сон до тех пор, пока не произойдет прерывание.
void hal_sleep (){
        //__WFI();
}

extern void radio_irq_handler(uint8_t dio);
/********************* Interrupt ************************/
/*DIO0, DIO1 и DIO2 инициируют прерывания по нарастающему фронту,
и соответствующие обработчики прерываний вызывают функцию radio_irq_handler ()
и передают  аргумент i = 0, 1 или 2 , ктр соответствует DIO0, DIO1 и DIO2*/

void EXTI1_IRQHandler(void) {
        if (EXTI->PR & EXTI_PR_PR1){
                EXTI->PR |= EXTI_PR_PR1;//сбрасываем флаг прерывания
                radio_irq_handler(0);
        }
}
void EXTI15_10_IRQHandler(void) {
        if (EXTI->PR & EXTI_PR_PR10){
                EXTI->PR |= EXTI_PR_PR10;//сбрасываем флаг прерывания
                radio_irq_handler(1);
        }
        else    if (EXTI->PR & EXTI_PR_PR11){
                EXTI->PR |= EXTI_PR_PR11;
```

```
                                        //сбрасываем флаг прерывания
        }
}

/*прерывание по таймеру 2 считает 32-битное системное время в тиках*/

void TIM2_IRQHandler(void) {
        if (TIM2->SR & TIM_SR_UIF){
                tick++;
        }
}
```

HAL.h

```
/********************** library ************************/
#include "stm32f10x.h"
#include "libSPI.h"
#include "sysInit.h"
#include "libEXTI.h"
#include "libTIM.h"
/******************** Use function **********************/
void hal_init (void);
void hal_failed (void);
void hal_pin_nss (uint8_t val);
void hal_pin_rxtx (uint8_t val);
void hal_pin_rst (uint8_t val);
uint8_t hal_spi (uint8_t outval);
void hal_disableIRQs (void);
void hal_enableIRQs (void);
void hal_sleep (void);
uint32_t hal_ticks (void);
void hal_waitUntil (uint32_t time);
uint8_t hal_checkTimer (uint32_t targettime);
void hal_failed (void);
```

debug.c

```
/********************** library ************************/
#include "debug.h"
/*-------------------------------------------------------------------------
Небольшая библиотека отладки  полезна для разработки и отладки. Функции
библиотеки обеспечивают простое ведение журнала последовательной консоли
и доступ к светодиоду.
                                USART1        -- PA9 : as alternate function push-pull
*-------------------------------------------------------------------------*/
/******************** Use function **********************/
// Иницация периферийных устройств
void debug_init () {
```

```
        InitUART ();
  debug_str("\r\n============== DEBUG STARTED ==============\r\n");
}
// управление светодиодом  (0 = выкл. 1 = вкл.)
void debug_led (int val) {
        if(val){
        GPIOC-> BSRR |= GPIO_BSRR_BS8;}
        else{
        GPIOC-> BSRR |= GPIO_BSRR_BR8;}
}
//Запись одиночного символа в UART.
void debug_char (char c) {
        while( !(USART1->SR & USART_SR_TXE) );
        USART1->DR = c;
}
//Ввод значение байта в виде двух шестнадцатеричных символов в UART.
void debug_hex (u1_t b) {
        debug_char("0123456789ABCDEF"[b>>4]);
        debug_char("0123456789ABCDEF"[b&0xF]);
}
// 32-битное целое число в виде восьми шестнадцатеричных цифр в USART
void debug_buf (const u1_t* buf, int len) {
 while(len--) {
          debug_hex(*buf++);
          }
          debug_char('\r');
          debug_char('\n');
}

//Ввод 32-битного значения unsigned int в виде восьми шестнадцатеричных цифр в последовательную консоль.
void debug_uint (u4_t v) {
  for(s1_t n=24; n>=0; n-=8) {
                debug_hex(v>>n);
  }
}

 //32-битное целое число в виде десятичных цифр, подписанных в USART
void debug_int (s4_t v) {
        char buf[10], *p = buf;
        int n = debug_fmt(buf, sizeof(buf), v, 10, 0, 0);
        while(n--)
                debug_char(*p++);
}
//записать строку с нулевым символом в USART
void debug_str (const char* str) {
  while(*str) {
    debug_char(*str++);
  }
}
//записать метку и 32-битное значение как hex в USART
```

```
void debug_val (const char* label, u4_t val) {
	debug_str(label);
	debug_uint(val);
	debug_char('\r');
	debug_char('\n');
}

//записать метку и 32-битное значение в качестве десятичного знака в USART
void debug_valdec (const char* label, s4_t val) {
	debug_str(label);
	debug_int(val);
}
// конвертируем целое число 'val' в строку ASCII (bin / oct / dec / hex / base36)
// сохраняем строку в 'buf', возвращаем количество написанных символов
int debug_fmt (char* buf, int max, s4_t val, int base, int width, char pad) {
	char num[33], *p = num, *b = buf;
	u4_t m, v;
	v = (base == 10 && val < 0) ? -val : val;
	do {
		*p++ = ((m=v%base) <= 9) ? m+'0' : m+'A'-10;
	} while( v /= base );
	if(base == 10 && val < 0) {
			*p++ = '-';
	}
	while( b-buf < max-1 && b-buf < width-(p-num) ) {
			*b++ = pad;
	}
	do *b++ = *--p;
	while( b-buf < max && p > num );
	return b - buf;
}
```

debug.h

```
/************************ library ************************/
#include "lmic.h"
#include "libUART.h"
/********************* Use function **********************/
void debug_init (void);
void debug_led (int val);
void debug_char (char c);
void debug_hex (u1_t b);
void debug_buf (const u1_t* buf, int len);
void debug_uint (u4_t v);
void debug_int (s4_t v);
void debug_str (const char* str);
void debug_event (int ev);
void debug_val (const char* label, u4_t val);
void debug_valdec (const char* label, s4_t val);
```

int debug_fmt (char* buf, int max, s4_t val, int base, int width, char pad);

sensor.c

```
/********************** library ************************/
#include "sensor.h"
/********************** defines ************************/
#undef _UMDK_MID_
#define _UMDK_BME280_ 17
#define _UMDK_tem_ 6
/********************** Use function ********************/
static bool is_polled = false;
module_data_t data;
uint8_t val = 0;

void READSensor (void){
        os_radio(RADIO_RST);
        ADC1-> CR2 |= ADC_CR2_SWSTART;// пуск преобразования
        data.as_ack = is_polled;
  is_polled = false;
        if (LMIC.library == IBM){
                data.length = 1;
                data.data[0] = val;
                val++;
        }
        else{
                data.length = 7;
                data.data[0] = 6;
                data.data[1] = 0x03;
                data.data[2] = 0xD1;
                data.data[3] = 0x97;
                data.data[4] = 0x96;
        }
        memcpy( LMIC.frame , &data.data,data.length);
        if (LMIC.library == Unwired_Device){
                LMIC_setTxDataUD(1, LMIC.frame, data.length, 0, 0); // (port 1, datalen 1, no ack re-
quested,confirmed =0)
        }
        if (LMIC.library == IBM){
                LMIC_setTxData2   (1, LMIC.frame,data.length, 0);
        }
}
```

sensor.h

```
/********************** library ************************/
#include "stm32f10x.h"
#include "lmic.h"
/********************** Use function ********************/
```

```
void InitSensor (void);
void READSensor (void);
/********************** defines **********************/
#define UNWDS_MAX_DATA_LEN 126
typedef struct {
	uint8_t length;
	uint8_t data[UNWDS_MAX_DATA_LEN];
	int16_t rssi;
	bool as_ack;	/**< This data could be sent as ACK for downlink command */
} module_data_t;
```

adc.c

```
/********************** library **********************/
#include "adc.h"
/******************** Use function **********************/
void InitADC (void){
	RCC->APB2ENR	|= RCC_APB2ENR_ADC1EN;
	ADC1->CR1	|= ADC_CR1_DISCEN;
	ADC1->CR2	|=ADC_CR2_TSVREFE;
	ADC1->SQR1 = 0x00000000;
	ADC1->SQR2 = 0x00000000;
	ADC1->SMPR1 |= ADC_SMPR1_SMP16;// количество тактов между выборками
	ADC1->CR2	|= ADC_CR2_EXTSEL;
	ADC1->CR2			&= ~ADC_CR2_CONT;// 0 - единичное преобразование, 1 - непрерывное пробразование
	ADC1->CR2			|= ADC_CR2_EXTTRIG;
	ADC1->CR2	|=ADC_CR2_ADON;
}

uint16_t RD_ADC(uint8_t nk){
	ADC1->SQR3 = nk; // Задать номер канала
	ADC1->CR2 |= ADC_CR2_SWSTART; // Пуск преобразования
	while(!(ADC1->SR & ADC_SR_EOC)){}; // Ждать окончания преобразования
	return ADC1->DR; // Считать результат преобразования
}
```

adc.h

```
/********************** library **********************/
#include "stm32f10x.h"
#include "lmic.h"
/******************** Use function **********************/
void InitADC (void);
uint16_t RD_ADC(uint8_t nk);
```

modem.c

```
/********************** library *************************/
#include "modem.h"
#include "debug.h"
//ключи сети
u1_t APPEUI[8]  = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }; //{ 0x02, 0x00, 0x00, 0x00,
0x00, 0xEE, 0xFF, 0xC0 };
u1_t DEVEUI[8]  = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF };
u1_t DEVKEY[16] = { 0xA1, 0xA1, 0xA1, 0xA1, 0xA1, 0xA1, 0xA1, 0xA1, 0xA1, 0xA1, 0xA1,
0xA1, 0xA1, 0xA1, 0xA1, 0xA1 };
struct persist_t PERSIST;
static void startJOIN (xref2osjob_t osjob);

void os_getDevEui (u1_t* buf) {
  memcpy(buf, PERSIST.deveui, 8);
}


void os_getDevKey (u1_t* buf) {
  memcpy(buf, PERSIST.joinkey, 16);
}

void os_getArtEui (u1_t* buf) {
 memcpy(buf, PERSIST.appeui, 8);
}

void os_getArtClass (ls_node_class_t *buf) {
 *buf = PERSIST.nodeClass;
}

void os_getArtDatarate (uint8_t *buf) {
 *buf = PERSIST.Datarate;
}

void os_getArtmax_retr (uint8_t *buf) {
 *buf = PERSIST.max_retr;
}

void os_getArtperiod (uint8_t *buf) {
 *buf = PERSIST.period;
}

void os_getArtlibrary (library_t *buf) {
 *buf = PERSIST.library;
}
void lscfg () {
 if (PERSIST.library == Unwired_Device){
          debug_str("\r\n============= Unwired_Device ==============\r\n");
          debug_str("\r\n APPID64 =  0x");
          debug_buf(PERSIST.appeui,8);

          debug_str("\r\n EUI64 =  0x");
```

```
        debug_buf(PERSIST.deveui,8);

        debug_str("\r\n JOINKEY = 0x");
        debug_buf(PERSIST.joinkey,16);

        debug_str("\r\n CLASS = ");
        switch(PERSIST.nodeClass){
                case LS_ED_CLASS_A : {
                        debug_str("A");
                        break;
                }
                case LS_ED_CLASS_B : {
                        debug_str("C");
                        break;
                }
                case LS_ED_CLASS_C : {
                        debug_str("C");
                        break;
                }
        }
        debug_char('\r');
        debug_char('\n');

        debug_valdec("\r\n DATARATE = " ,PERSIST.Datarate);
        debug_char('\r');
        debug_char('\n');
        debug_valdec("\r\n MAXRETR = " ,PERSIST.max_retr);
        debug_char('\r');
        debug_char('\n');

        debug_valdec("\r\n PERIOD = " ,PERSIST.period);
        debug_char('\r');
        debug_char('\n');
}

else{

        debug_str("\r\n================== IBM ==================\r\n");
        debug_str("\r\n APPID64 =  0x");
        debug_buf(PERSIST.appeui,8);

        debug_str("\r\n EUI64 =  0x");
        debug_buf(PERSIST.deveui,8);

        debug_str("\r\n DEVKEY = 0x");
        debug_buf(PERSIST.joinkey,16);

        debug_valdec("\r\n DATARATE = " ,PERSIST.Datarate);
  debug_char('\r');
  debug_char('\n');
```

```
            debug_valdec("\r\n PERIOD = " ,PERSIST.period);
            debug_char('\r');
            debug_char('\n');
 }
}
void library_sel () {
          debug_str("\r\n Unwired Device(0) or IBM(1)? \r\n");
}

void modem_init () {
            state = s0;
            library_sel ();
   usart_startrx();
}

void modem_init2 () {
 state = s1;
 memcpy(PERSIST.appeui, APPEUI, 8);
 memcpy(PERSIST.deveui, DEVEUI, 8);
 memcpy(PERSIST.joinkey, DEVKEY, 16);
 PERSIST.Datarate = 5;
 PERSIST.max_retr = 5;
 PERSIST.period = 1;
 PERSIST.nodeClass = LS_ED_CLASS_A;
 lscfg ();
 JOIN();
}

/*****************************HEX*****************************/
static const s1_t hexval[] = {
   /*00-1F*/ -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
   /*20-3F*/ -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,-1,-1,-1,-1,-1,-1,-1,
   /*40-5F*/ -1,10,11,12,13,14,15,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
   /*60-7F*/ -1,10,11,12,13,14,15,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
   /*80-9F*/ -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
   /*A0-BF*/ -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
   /*C0-DF*/ -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
   /*E0-FF*/ -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
};

u1_t gethex (u1_t* dst, const u1_t* src, u2_t len) {
 u1_t n = 0;
 if(len & 1) { // odd number of digits
          return 0;
 }
 while(len--) {
          s1_t v = hexval[*src++];
          if(v < 0) { // bad hex digit
            return 0;
          }
          *dst = (*dst << 4) | v; // shift nibble
```

```
        if((len & 1) == 0) { // advance at every second digit
          dst++;
          n++;
        }
 }
 return n;
}

/*****************************************************************/
void JOIN (void) {
        os_setTimedCallback(&LMIC.osjob, os_getTime()+sec2osticks(10), FUNC_ADDR(startJOIN
));
}

static void startJOIN (xref2osjob_t osjob) {
        LMIC_reset();
        LMIC_startJoining();
}

static osjob_t reportjob;

// Опрос датчиков
static void reportfunc (osjob_t* j) {
        READSensor();
  os_setTimedCallback(j, os_getTime()+sec2osticks(LMIC.period*60), reportfunc);
}

void onEvent (ev_t ev) {
 switch(ev) {
        case EV_JOINED:
                debug_led(1);
                reportfunc(&reportjob);
                break;
 }
}
```

modem.h

```
/********************** library ************************/
#include "stm32f10x.h"
#include "sensor.h"

#ifndef _modem_h_
#define _modem_h_

#ifndef os_getArtlibrary
void os_getArtlibrary   (library_t *buf);
#endif

#ifndef os_getArtlibrary
```

```
void os_getArtlibrary   (library_t *buf);
#endif

#ifndef os_getArtClass
void os_getArtClass (ls_node_class_t *buf);
#endif

#ifndef os_getArtDatarate
void os_getArtDatarate (uint8_t *buf);
#endif

#ifndef os_getArtmax_retr
void os_getArtmax_retr  (uint8_t *buf);
#endif

#ifndef os_getArtperiod
void os_getArtperiod  (uint8_t *buf);
#endif

struct persist_t{
                u1_t deveui[8];
   u1_t appeui[8];
                u1_t joinkey[16];
                uint8_t Datarate;
                uint8_t max_retr;
                uint8_t period;
                library_t library;
                ls_node_class_t nodeClass;
};
extern struct persist_t PERSIST;
void JOIN (void);
u1_t gethex (u1_t* dst, const u1_t* src, u2_t len);
void modem_init (void);
void modem_init2 (void);
void lscfg (void);
void library_sel (void);
#endif // _modem_h_
```