Содержание

Подп. и дата | Инв. № | Взам. инв. | Подп. и дата | Инв. №

Изм.	Лист	№ докум.	Подп.	Дата	ТПЖА 09.03.01.066		
Разраб.		Рзаев А. Э.			Разработка системы Bluetooth-передатчиков	Лит.	Лист 3 / Листов 41
Пров.		Караваева О.В.				Кафедра ЭВМ Группа ИВТ-32	

Введение

На сегодняшний день разработка устройств в рамках концепции «Умный дом» очень популярна благодаря тому, что имеются очень широкий набор модулей, из которых довольно легко собрать нужное устройство: умный выключатель света, датчик перемещения, освещенности, температуры; умная розетка и т. п., имея минимальные знания в области электроники и программирования.

Для автоматизации контроля микроклимата на садовом участке возникла необходимость в разработке устройства, представляющего собой сильно упрощенную версию метеостанции.

1 Анализ предметной области

1.1 Актуальность разработки

Устройство предназначено для мониторинга температуры и влажности на территории садового участка.

На данный момент устройства, предоставляющие схожий функционал не имеют возможности объединения их в сеть и передачи данных ведущему устройству, например, ПК или смартфону, не только напрямую от устройства с датчиком, но и по цепочке.

1.2 Краткая характеристика области применения

Устройство предназначено к использованию на садовых и огородных участках.

1.3 Требования к устройству

Разрабатываемое устройство должно соответствовать следующим требованиям:

- устройство должно быть выполнено в виде отдельного блока;
- устройство должно работать от автономного источника питания;
- устройство должно считывать и передавать следующие показатели окружающей среды: температура воздуха и влажность;
- передача данных выполняется по сети Bluetooth;
- данные должны передаваться по цепочке от одного устройства к другому;
- диапазон измерений температуры: -5°C до +35°C;
- диапазон измерений влажности: от 0 до 100 %;
- диапазон рабочих температур должен быть от -5°C до +35°C.

2 Разработка структуры устройства

2.1 Выбор микроконтроллера

При выборе микроконтроллера основными критериями были цена, потребление энергии и сложность в настройке.

2.1.1 Raspberry Pi 3

Основными достоинствами данного микрокомпьютера являются самый мощный процессор (одноядерный ARM Cortex-A53, с частотой до 1.2 ГГц) из рассматриваемых вариантов, большой объем оперативной памяти (1024 МБ), а также наличие большого количества разъемов для подключения различных датчиков.

Однако главным недостатком данного устройства является высокое по сравнению с Arduino потребление электроэнергии (350-400 мА против 30-50 мА). Для этого могут потребоваться достаточно емкие автономные источники питания, цена на которые сравнима со стоимостью самого микрокомпьютера.

Таким образом, несмотря на отличные аппаратные характеристики, данное устройство не может быть использовано из-за высокого потребления энергии.

2.1.2 Arduino Nano

Данный микроконтроллер является наиболее приемлемым в виду низкой стоимости, компактности и низкому потреблению энергии.

Однако данный микроконтроллер доступен в продаже только в разобранном виде: к основной плате не припаяны штекеры выводов. Это значительно усложняет первоначальную сборку и настройку, также при ручной распайке штекеров возможно как механическое, так и термическое повреждение основной платы микроконтроллера.

2.1.3 Arduino Uno

Рассматриваемый микроконтроллер является улучшенной версией Arduino Nano. Его использование решает проблемы, возникающие при работе с Arduino Nano: на месте штекеров выводов уже припаяны блоки с разъемами для вставки соединительных проводов.

Основными недостатками в данном случае является более высокая цена и размеры используемой платы. Однако эти недостатки не являются критичными для функциональной части, а цена возрастает незначительно.

В результате проведенного анализа было принято решение использовать в качестве микроконтроллера Arduino Uno, так как он удовлетворяет всем требованиям заказчика.

2.2 Выбор модулей, необходимых для функционирования устройства

Основными критериями выбора модулей являлась цена и возможность получения модуля в кратчайшие сроки.

В качестве модуля Bluetooth LE было рассмотрено два варианта: HM-10 и MLT-BT05. Оба модуля имеют одинаковое конструктивное исполнение, интерфейс управления и рабочие характеристики. Однако модуль HM-10 стоит значительно дороже (в 3 и более раз), чем MLT-BT05. Поэтому было принято решение использовать модуль MLT-BT05.

В качестве модуля для измерения температуры и влажности было рассмотрено также два варианта: DHT-11 и DHT-22. Оба модуля имеют одинаковое конструктивное исполнение и способ подключения. Но модуль DHT-11 не отвечает требованиями к устройству, так как он может измерять влажность только в пределах 20-80 % и температуру от 0°C до +50°C. Поэтому выбор остановился на модуле DHT-22.

В качестве источника автономного питания были выбраны солнечные батареи, так как в отличие от аккумуляторов не требуют подзарядки.

Таким образом, для реализации функционала, предъявляемого к разрабатываемому устройству, были выбраны следующие модули:

- микроконтроллер Arduino Uno;
- датчик температуры и влажности DHT-22;
- модуль Bluetooth LE MLT-BT05;
- источник питания (солнечные панели).

2.3 Схема подключения оборудования

2.3.1 Arduino Uno

Arduino Uno является связующим устройством для всех остальных. Питание микроконтроллера возможно как через USB-разъем, так и через специальные разъемы: VCC и GND. В рабочем режиме питание подается именно через них с солнечной панели. На плате расположено три блока разъемов: питания, цифровой и аналоговый. К цифровым разъемам 1 и 2 возможно подключить конвертер UART-USB для просмотра отладочных сообщений.

2.3.2 Датчик температуры и влажности DHT-22

Датчик DHT-22 состоит из двух частей: емкостной датчик влажности и термистор. Также в корпусе установлен чип для преобразования аналогового сигнала в цифровой на выходе.

Основные характеристики датчика:

- питание от 3 до 5 В;

- максимально потребляемый ток – 2.5 мА (при обработке запроса данных);
- измерение уровня влажности в диапазоне от 0% до 100%, точность измерений – 2-5%;
- измерение температуры в диапазоне от -40 до 125 °C с точностью 0.5 °C;
- частота измерений – 0.5 Гц.

Датчик имеет 4 коннектора:

- питание (VCC) – от 3 до 5 В;
- вывод данных (DATA);
- не используется;
- земля (GND).

Коннектор ввода данных подключается к 8-му цифровому разъему на плате Arduino.

Для считывания данных используется библиотека DHT-sensor-library от Adarfruit, которая интегрирована в Arduino IDE. Первоначальная настройка включает в себя создание экземпляра класса DHT с указанием номера разъема ввода данных и типа датчика. Данный класс предоставляет следующие методы для чтения данных:

- float readHumidity() – считывание влажности;
- float readTemperature() – считывание температуры.

2.3.3 Модуль Bluetooth LE MLT-BT05

MLT-BT05 – небольшой Bluetooth 4.0 LE модуль, основанный на SoC Texas Instruments CC2541.

Основные характеристики:

- питание от 2.5 до 3.3 В;

– максимально потребляемый ток – 50 мА (при передаче данных);
– рабочая частота – 2.4 ГГц ISM;
– дальность действия – до 100 м;
– рабочая температура – от -5 до +65 °С.

Датчик имеет 4 коннектора:

– питание (VCC) – 3.3 В;
– UART RX;
– UART TX;
– земля (GND).

Коннекторы UART RX и UART TX подключаются к разъемам 9 и 10 платы Arduino соответственно.

Для управления модулем и обработки данных прошивкой предоставляется последовательный UART интерфейс. Ardiono IDE предоставляет стандартный интерфейс Serial, для которого достаточно указать номера разъемов RX и TX. Управление поведением модуля осуществляется с помощью специальных AT-команд. Необходимые для реализации устройства команды приведены в таблице 1.

Таблица 1 – Основные команды устройства

Команда	Описание
AT+ROLE	Установка режима работы: Master – 1 Slave – 0
AT+CONA	Подключиться к устройству по его MAC-адресу
AT+DISC	Разорвать соединение
AT	Проверить работу порта
AT+RESET	Программный перезапуск

В приложении А представлена схема подключения модулей к плате Arduino.

3 Разработка алгоритмов функционирования

3.1 Разработка формата сообщений для передачи

Передача данных с датчиков происходит по цепочке, от одного устройства к другому. Порядок передачи определен заранее и программно не изменяется.

Данные между двумя устройствами передаются в виде сообщений фиксированной длины в 6 байт. Сообщение состоит из следующих полей:

- тип сообщения (1 байт);
- идентификатор отправителя (1 байт);
- идентификатор источника данных (1 байт);
- температура (1 байт);
- влажность (1 байт);
- контрольная сумма (1 байт).

Тип сообщения определяет, является ли данное сообщение последним в цепочке или нет. Определено два типа: «обычный» и «терминальный». Если сообщение имеет тип «обычный», то оно просто пересылается дальше. Тип сообщения «терминальный» означает, что нужно после пересылки этого сообщения сформировать новое, содержащее данные со своих датчиков. Код сообщения с типом «обычный» – 0x00, с типом «терминальный» – 0x01.

Идентификатор отправителя является уникальным номером устройства, от которого было принято сообщение. Данное поле необходимо для того, чтобы избежать нарушения порядка приема-передачи сообщений по цепочке. Данное устройство может принимать сообщения от любых других, но обрабатывать и пересылать только от одного, заранее определенного.

Идентификатор источника данных является уникальным номером устройства, которое сформировало данные с датчиков влажности и температуры. Необходимо для ведения статистики.

Поле с температурой занимает один байт, хранит целочисленное значение.

Поле с влажностью, аналогично занимает один байт, хранит целочисленное значение.

Поле с контрольной суммой необходимо для проверки, правильно ли были считаны данные.

Так как один и тот же последовательный интерфейс используется для передачи данных и для посылки управляющих команд, то возможен случай, когда данные могут быть интерпретированы как команды и соединение будет прервано. Такое может случится если в сообщении окажется следующая последовательность символов: «AT\n\r». Для того, чтобы избежать данной ситуации было принято решение прибавлять константу 0x0D к полям (байтам) температуры и влажности. Данная поправка не приведет к искажению данных вследствие целочисленного переполнения, так как диапазон измеряемых величин может быть представлен значениями от 14 до 255. Поле с контрольной суммой должно содержать значение большее 128. Значение 0x0D среди идентификаторов устройств не должно использоваться.

3.2 Варианты устройства

Для организации передачи сообщений от одного устройства к другому было разработано три специализированных варианта устройства:

- Генератор. Осуществляет сбор данных с датчиков и передачу данных на следующее устройство. Является начальным звеном цепи.
- Ретранслятор. Осуществляет прием и передачу данных с предыдущего устройства, а также передачу данных со своих датчиков.

– Приемник. Осуществляет прием данных с предыдущего устройства и вывод данных через последовательный порт. Является конечным звеном цепи.

3.3 Алгоритм приема данных

В режиме приема данных устройство ожидает поступления данных. Так как сообщения начинается либо с байта 0x00, либо с байта 0x01, то все считанные байты, не равные данному значению пропускаются. Если считанный байт равен 0x00 или 0x01, то дальше выполняется чтение остальных пяти байт. В случае если очередной байт невозможно считать, то данный блок данных сбрасывается. Если байты продолжают поступать, то это также является ошибкой, все они считываются, но не сохраняются, блок данных сбрасывается.

Граф-схема алгоритма приема данных представлена на рисунке 1.

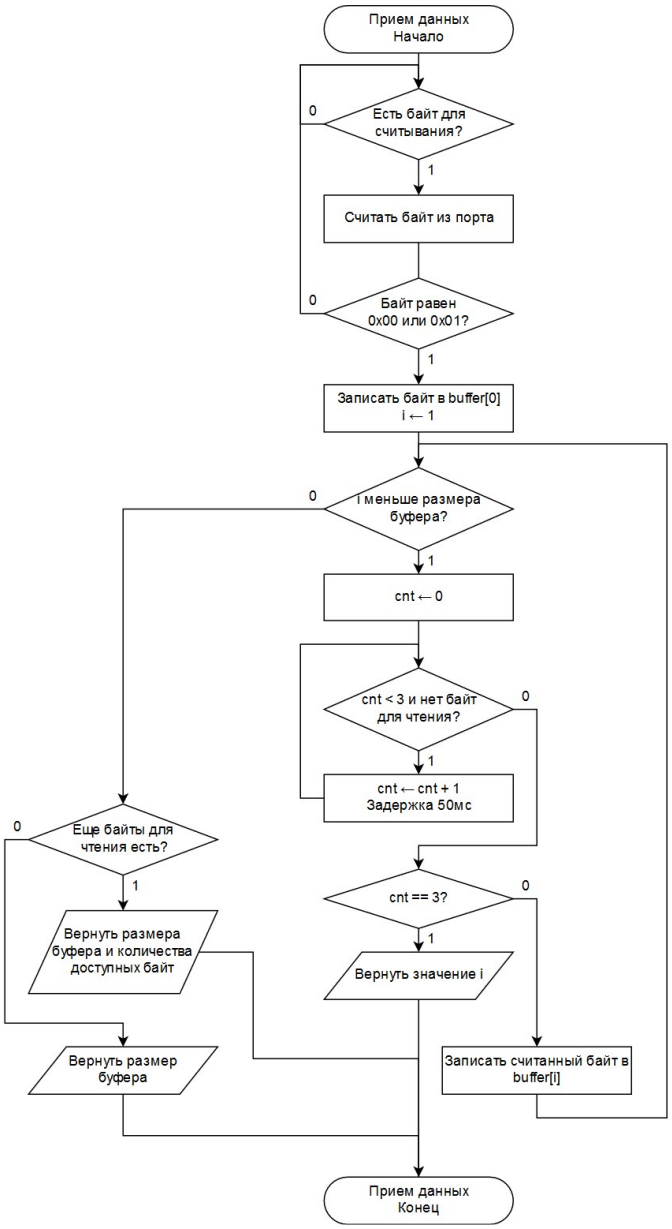


Рисунок 1 – Граф-схема алгоритма приема данных

3.4 Алгоритм проверки данных

Для проверки того, что данные от другого устройства были получены без искажений, необходимо выполнить подсчет контрольной суммы. Сумма вычисляется как XOR всех байт сообщения. Чтобы полученное значение не совпало случайно с кодом символа ‘\r’ (0x0D), в старший бит байта контрольной суммы записывается единица.

3.5 Алгоритм передачи данных

Передача данных устройством обеспечивается в режиме «Master», для перехода в который используется команда «AT+ROLE1». После переключения режима необходимо подключиться к другому Bluetooth-модулю с помощью команды «AT+CONA<MAC-адрес устройства>». Установка подключения выполняется в течение 3-5 секунд, если соединение установилось, то модуль переходит в режим передачи данных. Затем отправляется сообщение с заранее подготовленной информацией. После соединение разрывается командой «AT+DISC», устройство переводится в режим «Slave» с помощью команды «AT+ROLE0». В случае если соединение не было установлено, отправляемые данные будут проигнорированы.

Граф-схема алгоритма передачи данных представлена на рисунке 2.

Рисунок 2 – Граф-схема алгоритма передачи данных

3.6 Основной цикл работы генератора

Основной цикл работы включает в себя следующие пункты:

1. Сбор данных о температуре и влажности с датчика.
2. Преобразование данных.
3. Подготовка сообщения для передачи, расчет контрольной суммы.
4. Передача данных.
5. Приостановка работы устройства на заданное время.

Граф-схема основного цикла генератора представлена на рисунке 3.

Рисунок 3 – Граф-схема основного цикла работы генератора

3.7 Основной цикл работы приемника

Основной цикл работы включает в себя следующие пункты:

1. Выполнение процедуры приема данных
2. Проверка количества считанных байт.
3. Если количество байт не совпадает с длиной сообщения, то полученные данных игнорируются, в случае совпадения считается контрольная сумма.
4. Если контрольные суммы не совпали, то данные игнорируются, в случае совпадения проверяется ID отправителя.
5. Если ID отправителя не совпал, то данные игнорируются, в случае совпадения данные передаются в UART-порт.

Граф-схема основного цикла работы приемника представлена на рисунке 4.

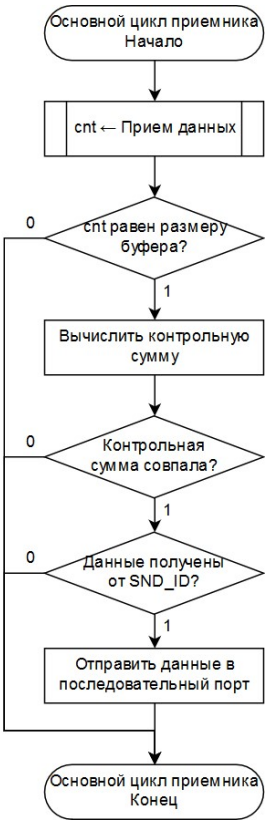


Рисунок 4 – Граф-схема основного цикла работы приемника

3.8 Основной цикл работы ретранслятора

Основной цикл работы включает в себя следующие пункты:

1. Выполнение процедуры приема данных.
2. Проверка количества считанных байт.

3. Если количество байт не совпадает с длиной сообщения, то полученные данных игнорируются, в случае совпадения считается контрольная сумма.
4. Если контрольные суммы не совпали, то данные игнорируются, в случае совпадения проверяется ID отправителя.
5. Если ID отправителя не совпал, то данные игнорируются, в случае совпадения выполняется подготовка данных для передачи.
6. Заменить ID отправителя на собственный ID.
7. Если тип сообщения – "обычное", то выполняется пересчет контрольной суммы и передача сообщения.
8. Если тип сообщения – "терминальное", то выполняются следующие действия:

8.1. Тип сообщения меняется на "обычное".

8.2. Пересчитывается контрольная сумма.

8.3. Передача сообщения.

8.4. Приостановка работы.

8.5. Сбор и преобразование данных с датчиков.

8.6. Подготовка и передача сообщения.

Граф-схема основного цикла работы ретранслятора представлена на рисунке 5.

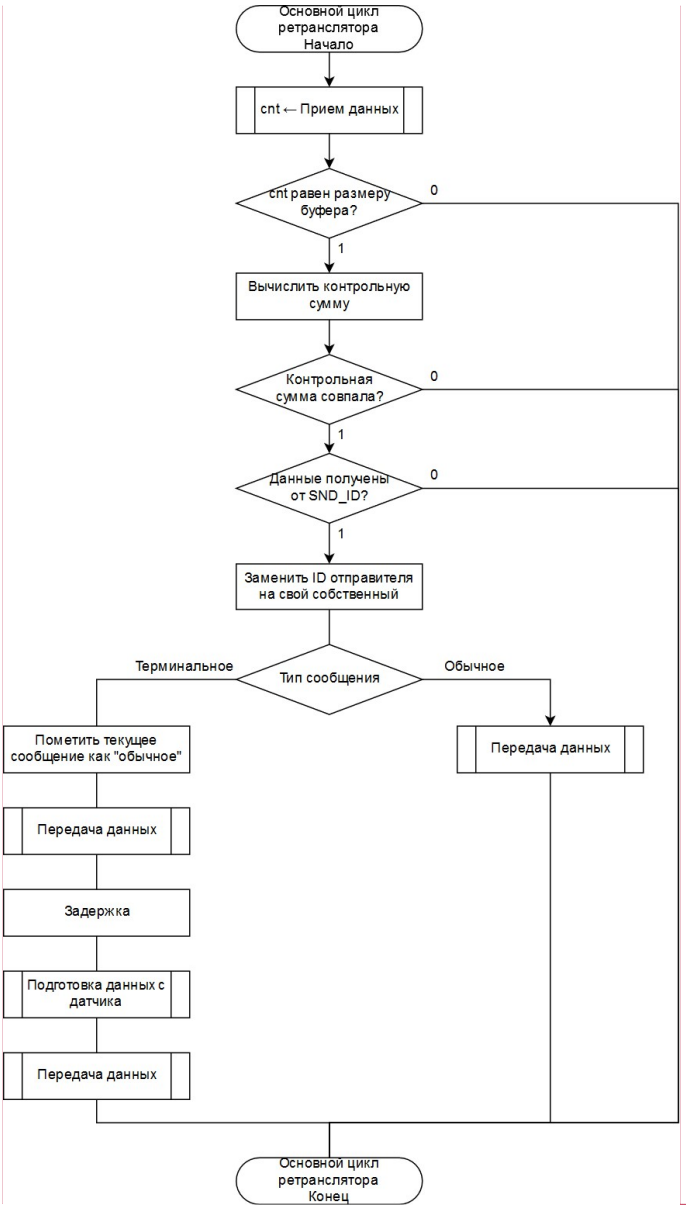


Рисунок 5 – Граф-схема основного цикла работы ретранслятора

Добавлено примечание ([РАЭо1]): Зачем помечать сообщение как «обычное»?

4 Программная реализация

Для программирования микроконтроллера используется упрощенная версия C++ (Wiring). В качестве среды разработки используется Arduino IDE. Программа состоит из двух частей: процедур setup() и loop(). В процедуре setup() производится начальная настройка устройства, подключается периферия. Код, размещенный в процедуре loop(), представляет собой основной цикл работы программы.

В процедуре setup() выполняется настройка UART-порта для вывода отладочных данных с помощью метода Serial.begin(), установка соединения с Bluetooth-модулем – метод SoftwareSerial.begin(). Затем модуль переводится в режим "Slave" командой "AT+ROLE0". Передача команд осуществляется методом SoftwareSerial.println().

В исходных текстах программ определены следующие константы:

- STD_MSG – тип сообщения «обычный», 0x00;
- TER_MSG – тип сообщения «терминальный», 0x01;
- SLEEP_INT – интервал простоя, значение для различных устройств разное;
- MAX_ATTEMPTS – количество попыток считывания для каждого байта, 3;
- BUFFER_SIZE – длина буфера (сообщения), 6;
- RX_PIN, TX_PIN – номера разъемов на плате Arduino для подключения Bluetooth-модуля, 9 и 10 соответственно.

Также для каждого устройства должно быть определено значение CUR_ID – ID текущего устройства.

4.1 Программа генератора

В исходном тексте дополнительно определены следующие значения:

- DHT_PIN – 8, номер разъема для подключения датчика температуры и влажности;
- DHT_TYPE – тип датчика, DHT22;
- RECEIVER_MAC – MAC-адрес устройства, принимающего данные;
- SLEEP_INT – 30000 (30 секунд).

В соответствии с алгоритмом работы устройства выполняется сбор данных с счетчиков с помощью процедуры prepareOwnData(). В качестве аргумента она получает указатель на начало буфера, в котором будет сформировано сообщение для передачи. В этой процедуре выполняется считывание температуры и влажности с помощью методов DHT.readTemperature() и DHT.readHumidity(). Тип сообщения устанавливается как «терминальный» – в нулевой байт буфера записывается значение TER_MSG. В следующие два байта записывается ID текущего устройства – CUR_ID, т. к. он одновременно является и источником данных, и отправителем сообщения. Температура и влажность переводятся в целые числа, к ним прибавляется константа 0x0D и записываются в третий и четвертый байты буфера соответственно. В последний байт записывается контрольная сумма, которая вычисляется в функции compCheckSum().

Сформированное сообщение пересылается на другое устройство с помощью процедуры sendData(). В этой процедуре, используя метод SoftwareSerial.println(), Bluetooth-модулю последовательно отправляются команды "AT+ROLE1" и "AT+CONA<RECEIVER_MAC>", которые устанавливают соединение между двумя модулями. После установки подключения выполняется пересылка данных. Для этого используется метод SoftwareSerial.write(), который принимает в качестве параметров указатель на начало буфера и длину буфера. После передачи сообщения соединение разрывается командами "AT" и "AT+DISC". После модуль переводится в

режим приема командой "AT+ROLE0". Работа устройства приостанавливается на интервал SLEEP_INT вызовом процедуры delay().

Исходный текст программы генератора представлен в приложении Б.

4.2 Программа приемника

В исходном тексте дополнительно определены следующие значения:

– SND_ID – ID устройства-отправителя. Только от него нужно принимать данные.

В соответствии с алгоритмом работы устройства выполняется прием данных. Для этого была реализована процедура readData(), принимающая указатель на начало буфера, в который необходимо записать данные. В качестве результата процедура возвращает количество считанных байт. На первом этапе процедура ожидает начала сообщения – STD_MSG или TER_MSG. Считывание байт выполняется в бесконечном цикле с помощью метода SoftwareSerial.read(). Если нет доступных данных, то метод возвращает -1. Если считанный байт не удовлетворяет условиям (не равен STD_MSG или TER_MSG), то он пропускается, в противном случае записывается в нулевой байт буфера, цикл прерывается. На следующем этапе в цикле считываются остальные байты. На чтение каждого байта дается MAX_ATTEMPTS попыток с задержкой после каждой в 50 мс. Если за эти попытки байт считать не удалось, то считается, что сообщение не пришло, процедура завершает свою работу. После успешного считывания данных проверяется, есть ли еще данные для чтения. При нормальном режиме данных больше не должно быть. Если же они есть, то считается, что сообщение передалось с ошибкой, байты считываются и процедура завершает свою работу. Во всех случаях выхода из процедуры возвращается количество только считанных байт.

В основном цикле программы анализируется количество прочитанных байт. Если оно не совпадает с BUFFER_SIZE, то данные игнорируются. На

следующем шаге проверяется целостность данных с помощью контрольной суммы. Если контрольная сумма не совпала или ID отправителя не равен SND_ID, то данные игнорируются. В случае совпадения данные выводятся последовательный порт – метод Serial.print().

Исходный текст программы приемника представлен в приложении В.

4.3 Программа ретранслятора

В исходном тексте дополнительно определены следующие значения:

- DHT_PIN – 8, номер разъема для подключения датчика температуры и влажности;
- DHT_TYPE – тип датчика, DHT22;
- RECEIVER_MAC – MAC-адрес устройства, принимающего данные;
- SLEEP_INT – 1500 (1.5 секунд);
- SND_ID – ID устройства-отправителя. Только от него нужно принимать данные.

Данный тип устройства совмещает в себе функционал генератора и приемника.

На первом шаге выполняется прием данных с Bluetooth-модуля, используя процедуру readData() аналогично приемнику. Считанные данные подлежат обработке и пересылке только при соблюдении следующих условий:

- длина считанного сообщения равна BUFFER_SIZE;
- контрольная сумма совпадает с вычисленной;
- ID отправителя совпадает с SND_ID.

На следующем шаге выполняется обработка сообщения. ID отправителя заменяется на CUR_ID путем записи данного значения в первый байт буфера. Если сообщение имеет тип "обычное" (в нулевом байте буфера хранится значение STD_MSG), то контрольная сумма пересчитывается и

сообщение отправляется с помощью процедуры sendData() так же как и в программе генератора. На этом цикл работы ретранслятора заканчивается.

Если у сообщения тип – «терминальный», то он меняется на тип «обычный» (значение STD_MSG записывается в нулевой байт буфера). Сообщение пересылается также с помощью процедуры sendData(). Работа устройства приостанавливается на время SLEEP_INT вызовом процедуры delay(). После выхода из простоя данные с датчиков собираются и передаются на другое устройство с помощью процедур prepareOwnData() и sendData() как в программе генератора. Цикл работы устройства на этом заканчивается.

Исходный текст программы ретранслятора представлен в приложении Г.

5 Тестирование

Для тестирования разработанных программ использовалось приложение для смартфона BLE Scanner. Данное приложение позволяет подключаться к устройствам Bluetooth LE и непосредственно передавать им данные в двоичном виде.

Были отработаны следующие сценарии работы устройства:

1. Сообщение отправлено с не тем ID устройства. В случае если в сообщение был отправлен ID не того, устройства, которое было установлено в принимающем устройстве, то в отладочном выводе должно появиться сообщение об ошибке, прием и выдача данных производиться не должна.
2. Количество байт в сообщении не равно BUFFER_SIZE. В данном случае в отладочном выводе должно появиться сообщение об ошибке с указанием количества принятых байт. Обработка и последующая передача данных производиться не должна.
3. Отправлено сообщение типа «обычный». В данном случае в отладочном выводе должны появиться обработанные данные и сообщение об успешной отправке данных.
4. Отправлено сообщение типа «терминальный». В данном случае в отладочном выводе должны появиться обработанные данные, данные, считанные с датчиков и сообщения об успешной передаче данных.
5. Несовпадение контрольных сумм. Если при приеме данных контрольная сумма в сообщении и вычисленная не совпали, то в отладочном выводе должно появиться сообщение об ошибке, включающее значения обоих контрольных сумм; передачи данных не происходит.

Заключение

В ходе выполнения курсового проекта было разработано устройство для мониторинга температуры и влажности на территории садового участка. Устройство было сделано в трех вариантах: приемник, генератор и ретранслятор. На этапе анализа было составлено техническое задание к разрабатываемому устройству. В ходе проектирования был определен состав оборудования и схема подключения модулей. При проектировании решались следующие проблемы: особенности управления Bluetooth-модулей; проверки целостности данных; особенности передачи данных по сети Bluetooth. На следующем этапе были разработаны алгоритмы функционирования устройства, протокол передачи данных. В ходе программной реализации на основе алгоритмов функционирования были разработаны прошивки для устройства и проведено тестирование на наиболее возможных сценариях.

В качестве направлений дальнейшего развития данного устройства можно определить следующие улучшения: расширение количества способов передачи данных – использования сетей WiFi и GPRS, расширение функционала – добавление команд для управления системой полива, использование комбинированных источников питания (солнечная панель + аккумулятор).

Перечень сокращений

ПК – персональный компьютер

Bluetooth LE – Bluetooth Low Energy, беспроводная технология Bluetooth с низким энергопотреблением

UART – universal asynchronous receiver-transmitter, универсальный асинхронный приёмопередатчик

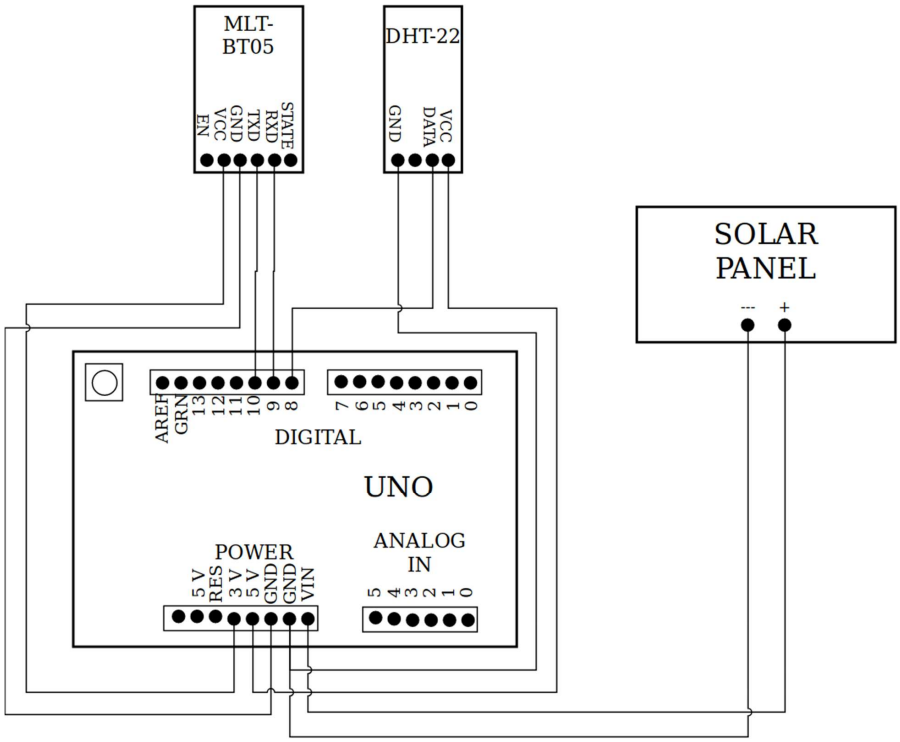
USB – universal serial bus, универсальная последовательная шина

SoC – system-on-a-chip, система на кристалле

Приложение А

(обязательное)

Схема подключения модулей



Приложение Б

(обязательное)

Исходный текст программы генератора

```
#include <SoftwareSerial.h>
#include <DHT.h>

#define CUR_ID 0x01
#define RECEIVER_MAC "1234567800AB"

#define STD_MSG 0x00
#define TER_MSG 0x01

#define SLEEP_INT 30000
#define MAX_ATTEMPTS 3

#define       BUFFER_SIZE 6
#define SND_DEV_ID_OFFSET 1
#define   MSG_TYPE_OFFSET 0

#define LED_PIN 13
#define  RX_PIN 9
#define  TX_PIN 10
#define DHT_PIN 8
#define DHT_TYPE DHT22

unsigned char data[BUFFER_SIZE] = { 0xFF };
unsigned char buff[BUFFER_SIZE] = {};

SoftwareSerial BT(RX_PIN, TX_PIN);
DHT dht(DHT_PIN, DHT_TYPE);

unsigned char compCheckSum(unsigned char *p) {
  unsigned char sum = 0;
  for (size_t i = 0; i < BUFFER_SIZE - 1; ++i) {
    sum ^= p[i];
  }
  return sum;
}

void prepareOwnData(unsigned char *p) {
  int t = (int) dht.readTemperature();
  t += 13;
  p[0] = TER_MSG;
  p[1] = CUR_ID;
  p[2] = CUR_ID;
  p[3] = (unsigned char) t;
  p[4] = compCheckSum(p);
}

void sendData(unsigned char *p) {
```

```
  BT.println("AT+IMME1");
  delay(200);
  BT.println("AT+ROLE1");
  delay(200);
  BT.println("AT+CONA" RECEIVER_MAC);
  delay(5000);
  BT.write(p, BUFFER_SIZE);
  delay(200);
  BT.println("AT");
  BT.println("AT+DISC");
  delay(200);
  BT.println("AT+ROLE0");
}

void printData(unsigned char* p) {
  for (size_t i = 0; i < BUFFER_SIZE; ++i) {
    Serial.print((int) p[i], HEX);
    Serial.print(" ");
  }
  Serial.println("");
}

void skipAscii() {
  do {
    int b = BT.peek();
    if (b >= 32 && b <= 127) {
      BT.read();
    } else {
      break;
    }
  } while (true);
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(38400);
  BT.begin(9600);
  delay(500);
  BT.print("AT+ROLE0");
  delay(500);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  prepareOwnData(buff);

  sendData(buff);
  delay(SLEEP_INT);

  digitalWrite(LED_PIN, LOW);
}
```

Приложение В

(обязательное)

Исходный текст программы приемника

```
#include <SoftwareSerial.h>
#define CUR_ID 0x01
#define SND_ID 0x00

#define STD_MSG 0x00
#define TER_MSG 0x01

#define SLEEP_INT 1500
#define MAX_ATTEMPTS 3

#define       BUFFER_SIZE 6
#define SND_DEV_ID_OFFSET 1
#define   MSG_TYPE_OFFSET 0

#define LED_PIN 13
#define  RX_PIN 9
#define  TX_PIN 10

unsigned char buff[BUFFER_SIZE] = {};

SoftwareSerial BT(RX_PIN, TX_PIN);

size_t readData(unsigned char *p) {
  while (true) {
    int c = BT.read();
    if (c == -1) {
      continue;
    } else if (c == STD_MSG || c == TER_MSG) {
      p[0] = c;
      break;
    }
  }

  for (size_t i = 1; i < BUFFER_SIZE; ++i) {
    int cnt = 0;
    unsigned int c;
    while ((c = BT.read()) == -1 && cnt < MAX_ATTEMPTS) {
      cnt++;
      delay(50);
    }

    if (cnt == MAX_ATTEMPTS) {
      return i;
    }

    p[i] = c;
  }
```

```
  size_t cnt = BT.available();
  if (cnt) {
    return BUFFER_SIZE + cnt;
  }

  return BUFFER_SIZE;
}

unsigned char getCheckSum(unsigned char *p) {
  return p[BUFFER_SIZE - 1];
}

unsigned char compCheckSum(unsigned char *p) {
  unsigned char sum = 0;
  for (size_t i = 0; i < BUFFER_SIZE - 1; ++i) {
    sum ^= p[i];
  }
  return sum;
}

unsigned char getDeviceId(unsigned char *p) {
  return p[SND_DEV_ID_OFFSET];
}

unsigned char getMsgType(unsigned char *p) {
  return p[MSG_TYPE_OFFSET];
}

void printData(unsigned char* p) {
  for (size_t i = 0; i < BUFFER_SIZE; ++i) {
    Serial.print((int) p[i], HEX);
    Serial.print(" ");
  }
  Serial.println("");
}

void skipAscii() {
  do {
    int b = BT.peek();
    if (b >= 32 && b <= 127) {
      BT.read();
    } else {
      break;
    }
  } while (true);
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(38400);
  BT.begin(9600);
```

```
  delay(500);
  BT.print("AT+ROLE0");
  delay(500);
  skipAscii();
}

void loop() {
  size_t cnt = readData(buff);
  if (cnt != BUFFER_SIZE) {
    Serial.print("Invalid size of data chunk: ");
    Serial.println(cnt);
    return;
  }

  Serial.println("DEBUG");
  printData(buff);

  if (compCheckSum(buff) != getCheckSum(buff)) {
    Serial.print("Check sum mismatch: ");
    Serial.print(compCheckSum(buff), HEX);
    Serial.print(" ");
    Serial.println(getCheckSum(buff), HEX);
  } else if (getDeviceId(buff) == SND_ID) {
    digitalWrite(LED_PIN, HIGH);

    Serial.print("Got data from ");
    Serial.println(getDeviceId(buff), HEX);
    printData(buff);

    digitalWrite(LED_PIN, LOW);
  }
}
```

Приложение Г

(обязательное)

Исходный текст программы ретранслятора

```
#include <SoftwareSerial.h>
#include <DHT.h>

#define CUR_ID 0x01
#define SND_ID 0x00
#define RECEIVER_MAC "1234567800AB"

#define STD_MSG 0x00
#define TER_MSG 0x01

#define SLEEP_INT 30000
#define MAX_ATTEMPTS 3

#define       BUFFER_SIZE 5
#define SND_DEV_ID_OFFSET 1
#define   MSG_TYPE_OFFSET 0

#define LED_PIN 13
#define  RX_PIN 9
#define  TX_PIN 10
#define DHT_PIN 8
#define DHT_TYPE DHT22

unsigned char data[BUFFER_SIZE] = { 0xFF };
unsigned char buff[BUFFER_SIZE] = {};

SoftwareSerial BT(RX_PIN, TX_PIN);
DHT dht(DHT_PIN, DHT_TYPE);

size_t readData(unsigned char *p) {
  while (true) {
    int c = BT.read();
    if (c == -1) {
      continue;
    } else if (c == STD_MSG || c == TER_MSG) {
      p[0] = c;
      break;
    }
  }

  for (size_t i = 1; i < BUFFER_SIZE; ++i) {
    int cnt = 0;
    unsigned int c;
    while ((c = BT.read()) == -1 && cnt < MAX_ATTEMPTS) {
      cnt++;
      delay(50);
    }
```

```
    if (cnt == MAX_ATTEMPTS) {
      return i;
    }

    p[i] = c;
  }

  size_t cnt = BT.available();
  if (cnt) {
    return BUFFER_SIZE + cnt;
  }

  return BUFFER_SIZE;
}

bool equal(unsigned char *p1, unsigned char *p2, size_t n) {
  for (size_t i = 0; i < n; ++i) {
    if (p1[i] != p2[i]) {
      return false;
    }
  }
  return true;
}

unsigned char getCheckSum(unsigned char *p) {
  return p[BUFFER_SIZE - 1];
}

unsigned char compCheckSum(unsigned char *p) {
  unsigned char sum = 0;
  for (size_t i = 0; i < BUFFER_SIZE - 1; ++i) {
    sum ^= p[i];
  }
  return sum;
}

unsigned char getDeviceId(unsigned char *p) {
  return p[SND_DEV_ID_OFFSET];
}

unsigned char getMsgType(unsigned char *p) {
  return p[MSG_TYPE_OFFSET];
}

void copy(unsigned char *src, unsigned char *dst, size_t n) {
  for (size_t i = 0; i < n; ++i) {
    dst[i] = src[i];
  }
}

void prepareOwnData(unsigned char *p) {
```

```
  int t = (int) dht.readTemperature();
  t += 13;
  p[0] = TER_MSG;
  p[1] = CUR_ID;
  p[2] = CUR_ID;
  p[3] = (unsigned char) t;
  p[4] = compCheckSum(p);
}

void sendData(unsigned char *p) {
  BT.println("AT+IMME1");
  delay(200);
  BT.println("AT+ROLE1");
  delay(200);
  BT.println("AT+CONA" RECEIVER_MAC);
  delay(5000);
  BT.write(p, BUFFER_SIZE);
  delay(200);
  BT.println("AT");
  BT.println("AT+DISC");
  delay(200);
  BT.println("AT+ROLE0");
}

void printData(unsigned char* p) {
  for (size_t i = 0; i < BUFFER_SIZE; ++i) {
    Serial.print((int) p[i], HEX);
    Serial.print(" ");
  }
  Serial.println("");
}

void skipAscii() {
  do {
    int b = BT.peek();
    if (b >= 32 && b <= 127) {
      BT.read();
    } else {
      break;
    }
  } while (true);
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(38400);
  BT.begin(9600);
  delay(500);
  BT.print("AT+ROLE0");
  delay(500);
  skipAscii();
}
```

```
void loop() {
  size_t cnt = readData(buff);
  if (cnt != BUFFER_SIZE) {
    Serial.print("Invalid size of data chunk: ");
    Serial.println(cnt);
    return;
  }

  Serial.println("DEBUG");
  printData(buff);

  if (compCheckSum(buff) != getCheckSum(buff)) {
    Serial.print("Check sum mismatch: ");
    Serial.print(compCheckSum(buff), HEX);
    Serial.print(" ");
    Serial.println(getCheckSum(buff), HEX);
  } else if (getDeviceId(buff) == SND_ID) {
    digitalWrite(LED_PIN, HIGH);

    Serial.print("Got data from ");
    Serial.println(getDeviceId(buff), HEX);
    printData(buff);

    buff[SND_DEV_ID_OFFSET] = CUR_ID;

    if (getMsgType(buff) == STD_MSG) {
      buff[BUFFER_SIZE - 1] = compCheckSum(buff);

      sendData(buff);
    } else {
      buff[MSG_TYPE_OFFSET] = STD_MSG;
      buff[BUFFER_SIZE - 1] = compCheckSum(buff);

      sendData(buff);
      delay(SLEEP_INT);
      prepareOwnData(buff);
      sendData(buff);

      digitalWrite(LED_PIN, LOW);
    }
  }
}
```

Приложение Д

(справочное)

Библиографический список

1 MLT-BT05 4.0 Bluetooth module [Электронный ресурс]. - Режим доступа: http://denethor.wlu.ca/arduino/MLT-BT05-AT-commands-TRANSLATED.pdf. - MLT-BT05-AT-commands-TRANSLATED.pdf. - (Дата обращения: 01.04.2018).

2 arduino-ble-ident-n-set [Электронный ресурс]. - Режим доступа: https://github.com/ayavilevich/arduino-ble-ident-n-set. - Arduino BLE module identification and setup sketch. Supports HM-10, CC41 and similar generic BLE modules. - (Дата обращения: 01.04.2018).

3 ELECTRIC INFO [Электронный ресурс]. - Режим доступа: http://electrik.info/main/praktika/1334-podklyuchenie-i-programmirovanie-arduino-dlya-nachinayuschih.html. - Подключение и программирование Ардуино для начинающих. - (Дата обращения: 01.04.2018).

4 adarfruit learning system [Электронный ресурс]. - Режим доступа: https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf. - DHT11, DHT22 and AM2302 Sensors. - (Дата обращения: 01.04.2018).

5 Bluetooth Modules | Martyn Currey [Электронный ресурс]. - Режим доступа: http://www.martyncurrey.com/bluetooth-modules/. - Bluetooth Modules. - (Дата обращения: 01.04.2018).

6 Power Consumption Benchmarks [Электронный ресурс]. - Режим доступа: https://www.pidramble.com/wiki/benchmarks/power-consumption. - Power Consumption Benchmarks | Raspberry Pi Dramble. - (Дата обращения: 01.04.2018).