Министерство науки и высшего образования
Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

# РАЗРАБОТКА КОНЕЧНЫХ УСТРОЙСТВ IoT

Методические указания для выполнения
лабораторных работ по дисциплине
«Технологии интернета вещей» для студентов
направлений 10.00.00 «Информационная безопасность»

Томск
Издательство ТУСУРа
2020

УДК 004.42(076)
ББК 32.973-04я73-5
Р177

**Рецензенты**

**Андронников И. В.**, ген. директор ООО «СКБ "Синти"»
**Шарамок А. В.**, канд. техн. наук, доц. каф. ТКС НИУ МИЭТ

**Авторы**

О. В. Пехов, Р. З. Хафизов, Е. М. Давыдова, А. К. Новохрестов

**Разработка** конечных устройств IoT: метод. указания для
Р177 выполнения лабораторных работ по дисциплине «Информационная безопасность интернета вещей» для студентов направлений 10.00.00 «Информационная безопасность» / О. В. Пехов [и др.]. – Томск: Изд-во Томск. гос. ун-та систем упр. и радиоэлектроники, 2020. – 71 с.

Настоящее методическое пособие используется в изучении дисциплины «Информационная безопасность интернета вещей» студентами направлений 10.00.00 «Информационная безопасность». Рассматриваются вопросы разработки конечных устройств интернета вещей на примере платформы Arduino, представлены указания для выполнения лабораторных работ, позволяющих углубленно изучить принцип работы встраиваемых систем. Представлены как основы работы с микрокон-троллером, так и работа с различными аналоговыми и цифровыми датчиками.

УДК 004.42(076)
ББК 32.973-04я73-5

© Пехов О.В., Хафизов Р.З.,
Давыдова Е.М.,
Новохрестов А.К., 2020
© Томск. гос. ун-т систем упр.
и радиоэлектроники, 2020

# Оглавление

# Лабораторная работа № 1
# Особенности организации систем ввода/вывода микроконтроллеров

## Цель работы

Изучение особенностей организации системы ввода/вывода микроконтроллера на примере использования платформы Arduino.

## Описание лабораторной работы

Arduino – это аппаратно-программная платформа для создания простых систем автоматики и робототехники, ориентированная на непрофессиональных пользователей. Система имеет полностью открытую архитектуру, это позволяет свободно копировать или дополнять линейку Arduino новыми решениями, что является одной из основных причин ее популярности. Под торговой маркой Arduino выпускается множество различных плат с микроконтроллером. Большинство плат снабжены минимально необходимым набором периферии для нормального функционирования микроконтроллера. Помимо этого, для платформы выпускается множество плат расширения (*shields*), повышающих функциональность Arduino за счет увеличения возможностей подключения дополнительного оборудования. Эти платы расширений подключаются к Arduino посредством установленных на них штыревых разъёмов.

Существует большое количество модификаций и клонов оригинальной платформы Arduino, но стоит отметить, что все они полностью совместимы с оригиналом. Ниже представлены наиболее распространенные версии плат [1]:

1) UNO – самая популярная версия платформы Arduino USB. UNO построена на микроконтроллере Atmel ATmega328p, имеет разъем USB-B для подключения к компьютеру;

2) Leonardo – версия платформы Arduino на микроконтроллере ATmega32u4. Визуально отличается разъемом microUSB, по размерам конструктивно совместима с UNO;

3) Nano – это компактная платформа, имеет небольшие размеры, отсутствует силовой разъем постоянного тока, в работе использует кабель Mini-B USB.

Использование платформы Arduino имеет как плюсы, так и минусы, например, несомненным преимуществом платформы является низкий порог вхождения, это позволяет сделать простой проект за несколько минут, применяя стандартные библиотеки, не вникая в особенности их работы. Но это же самое преимущество обращается в недостаток при попытке создать относительно сложный проект, ограничивает эффективность использования ресурсов микроконтроллера (МК).

Авторы данной работы считают, что платформа Arduino может быть использована на начальном этапе обучения работе с МК, а также для создания первичных моделей устройств с целью демонстрации принципа действия. В дальнейшем при создании образцов устройств стоит тщательно проанализировать требования к разрабатываемому устройству и на основании этих требований выбрать подходящий МК или платформу.

В данной лабораторной работе предлагается использовать плату Arduino UNO, но, учитывая аппаратную и программную совместимость разных плат платформы, большой роли это не играет. Распиновка платы Arduino UNO представлена на рисунке 1 [2].

Для связи с внешними элементами на плате Arduino UNO предусмотрены 14 цифровых выводов. Каждый вывод может быть определен программно как вход (INPUT) или выход (OUTPUT).

Для удобства работы некоторые пины совмещают в себе несколько функций:

- пины 0 и 1 – контакты UART (RX и TX соответственно);
- пины с 10 по 13 – контакты SPI (SS, MOSI, MISO и SCK соответственно);
- пины A4 и A5 – контакты I2C (SDA и SCL соответственно).

У цифрового вывода есть только два состояния: логический ноль и логическая единица. Логическая единица соответствует напряжению на выводе порядка 5 В, логический ноль – напряже-

нию близкому к 0 В. Эти состояния могут быть заданы в программе с использованием предопределенных констант HIGH и LOW соответственно. Выводы Arduino допускают подключение нагрузки с током потребления до 40 мА.

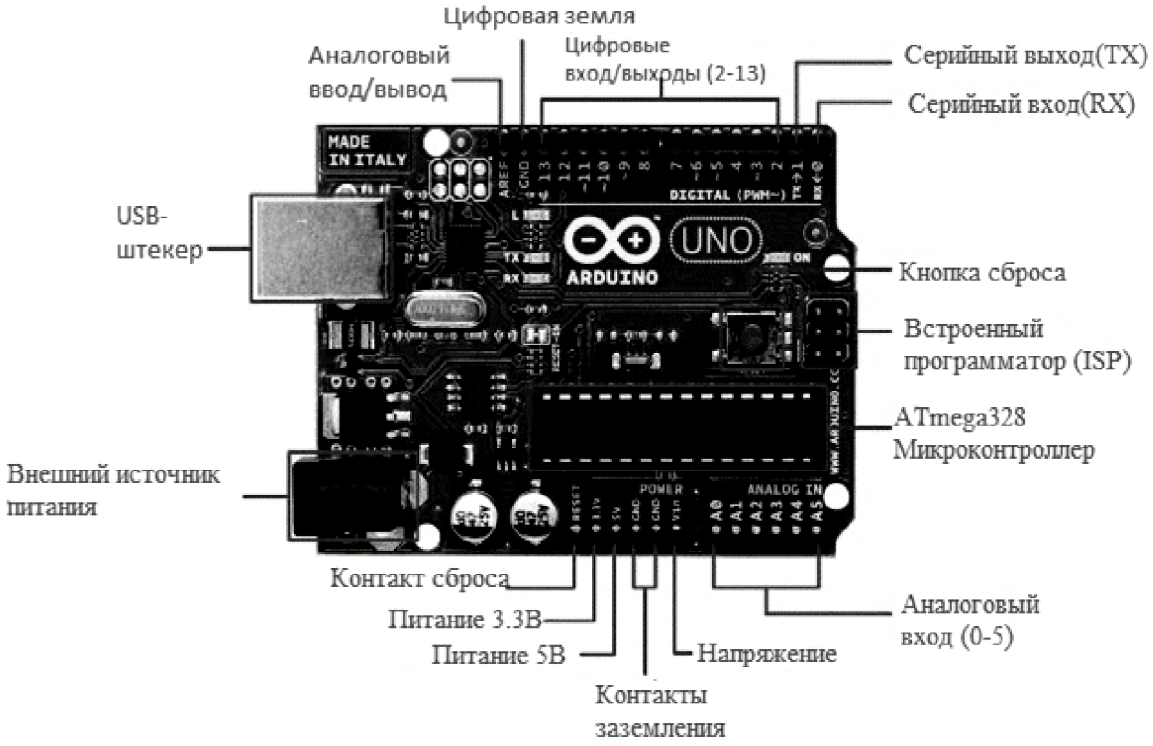



Рисунок 1 – Плата Arduino UNO
с указанием назначения пинов

Когда вывод определен как вход, считав его состояние, можно определить уровень напряжения на входе. При напряжении близком к 5 В, будет считана логическая единица. При напряжении близком к 0 В, будет считан логический ноль.

Программная часть Arduino представляет собой собственную кроссплатформенную интегрированную программную оболочку (IDE), доступную бесплатно на сайте Arduino (https://www.arduino.cc). В этой оболочке имеется текстовый редактор, менеджер проектов, препроцессор, компилятор и инструменты для загрузки программы в микроконтроллер. Для программирования плат Arduino используется язык C++ (компилятор AVR-GCC) с некоторыми особенностями, облегчающими написание первой работающей программы. В помощь новичкам предлагается применять обширный комплект библиотек, реали-

зующих возможности подключения дополнительного оборудования. Библиотеки распространяются свободно и их можно загрузить средствами IDE.

Программа, написанная в среде Arduino, называется скетч. Во время сохранения и экспорта проекта в области сообщений появляются пояснения, а также могут отображаться возникшие ошибки. Окно вывода текста (консоль) показывает сообщения Arduino, включающие полные отчеты об ошибках и другую информацию. Кнопки панели инструментов позволяют проверить и записать программу, создать, открыть и сохранить скетч, открыть мониторинг последовательной шины.

Структура программы Arduino достаточно проста и в минимальном варианте состоит из двух функций setup() и loop() (рисунок 2).

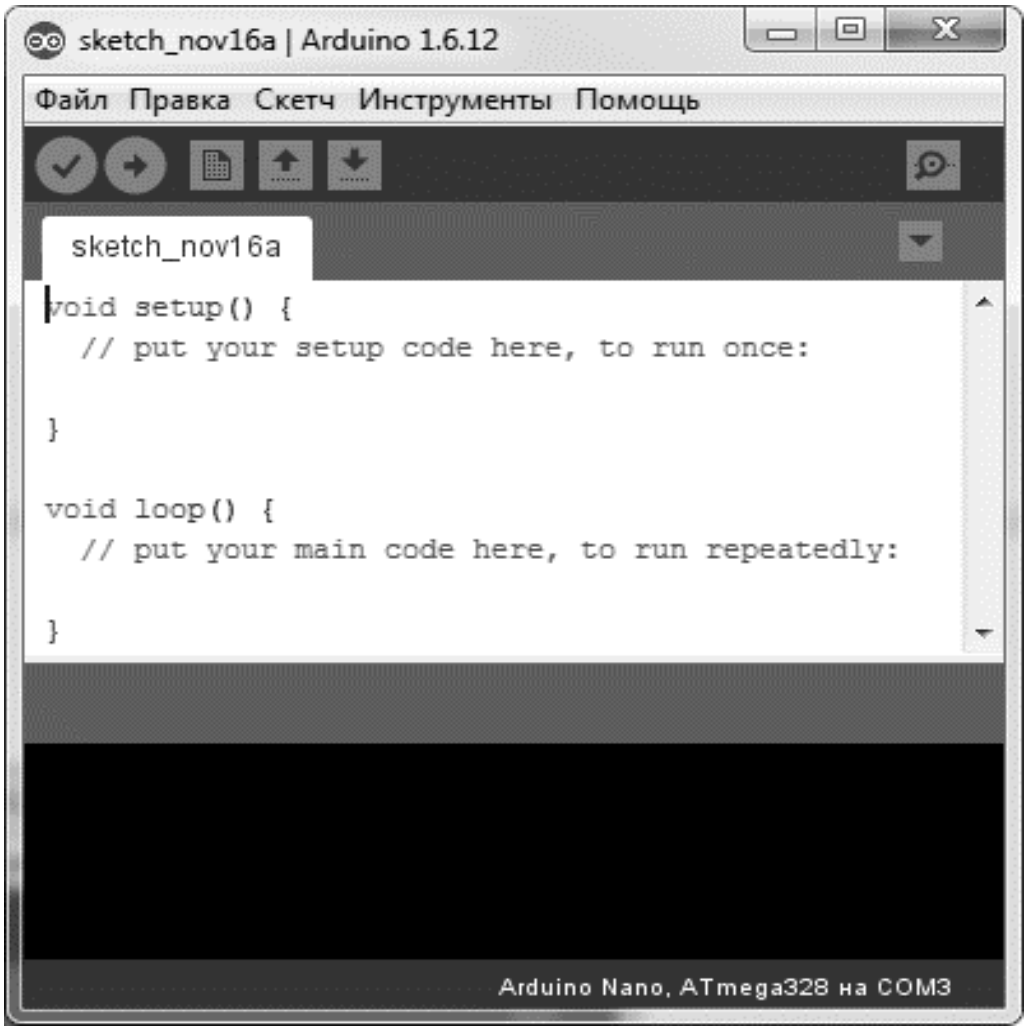



Рисунок 2 – Вид окна программы Arduino IDE

Обратите внимание на отсутствие функции main(), которая является обязательной в C++, препроцессор Arduino создает ее сам, вставляя туда необходимые «черновые» действия.

Функция setup() выполняется один раз, при включении питания или сбросе контроллера. Обычно она используется для инициализации ресурсов контроллера, объявления переменных и задания их начальных значений. Данная функция должна присутствовать в программе, даже если в ней ничего не исполняется.

Перед функцией setup() можно объявить директиву #define – это удобная директива, которая позволяет дать имя константе, перед тем как программа будет скомпилирована. Определенные этой директивой константы не занимают программной памяти, поскольку компилятор заменяет все обращения к ним их значениями на этапе компиляции, соответственно, они служат исключительно для удобства программиста и улучшения читаемости текста программы.

Для подключения сторонних библиотек в ваш скетч используется директива #include. Это дает доступ к большому числу стандартных библиотек C и библиотек, написанных специально для Arduino.

После завершения setup() управление переходит к функции loop(). Она в бесконечном цикле выполняет команды, записанные в ее теле. Собственно работа этих команд и является основным результатом функционирования прошивки контроллера.

**Функции управления вводом/выводом**

Для подключения простых внешних устройств к МК обычно используются выводы GPIO (интерфейс ввода/вывода общего назначения). Контакты GPIO могут выступать как в роли входа, так и в роли выхода – это, как правило, конфигурируется. В системе Arduino есть встроенные функции для работы с GPIO. Они позволяют установить режим вывода: считать или установить вывод в определенное состояние. Для определения состояния выводов в этих функциях используются константы HIGH и LOW, которые соответствуют высокому и низкому уровню сигнала. Ниже рассмотрены некоторые из них [3].

Функция pinMode(pin, mode) – устанавливает режим работы вывода (вход или выход), ничего не возвращает.

Аргументы:

– pin – номер вывода;

– mode – режим вывода.

Аргумент mode может принимать одно из возможных значений, представленных в таблице 1.

Таблица 1 – Значения аргумента mode

| | |
|---|---|
| mode = INPUT | Вывод определен как вход, подтягивающий резистор отключен |
| mode = INPUT_PULLUP | Вывод определен как вход, подтягивающий резистор подключен |
| mode = OUTPUT | Вывод определен как выход |

Pull-up – подтягивающий резистор, включённый между проводником, по которому распространяется электрический сигнал, и питанием либо между проводником и землёй (pull-down — подтягивающий вниз резистор).

Резисторы нужны, чтобы не оставить вход в «подвешенном» состоянии. Пусть есть схема как на рисунке 3.

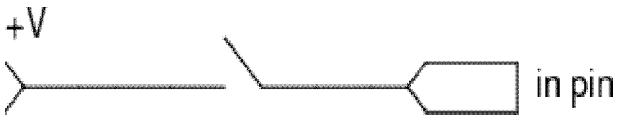



Рисунок 3 – Подключение кнопки к МК

Необходимо, чтобы, когда кнопка не нажата, вход фиксировал отсутствие напряжения. Но в данном случае вход находится в «никаком» состоянии. Он может срабатывать и не срабатывать хаотично, непредсказуемым образом. Причина тому – шумы, образующиеся вокруг: провода действуют как маленькие антенны и производят электричество из электромагнитных волн среды. Чтобы гарантировать отсутствие напряжения при разомкнутой цепи, рядом с входом ставится стягивающий резистор (рисунок 4). В этом случае при разомкнутом ключе в контроллере будет

фиксироваться уровень логического 0, при замкнутом ключе – уровень логической 1.

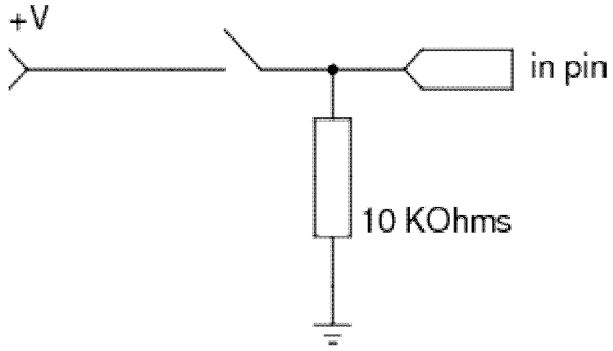



Рисунок 4 – Подключение кнопки к МК
через стягивающий резистор

Теперь нежелательный ток будет уходить через резистор в землю. Для стягивания используются резисторы больших сопротивлений (10 кОм и более). В моменты, когда цепь замкнута, большое сопротивление резистора не даёт большей части тока идти в землю: сигнал пойдёт к входному контакту. Если бы сопротивление резистора было мало (единицы Ом), при замкнутой цепи произошло бы короткое замыкание. Но сопротивление резистора не должно быть больше сопротивления входа контроллера, иначе нежелательный ток пойдет на вход контроллера.

Аналогично подтягивающий резистор удерживает вход в состоянии логической единицы, пока внешняя цепь разомкнута (рисунок 5). В этом случае при разомкнутом ключе в контроллере будет фиксироваться уровень логической 1, при замкнутом ключе – уровень логического 0.

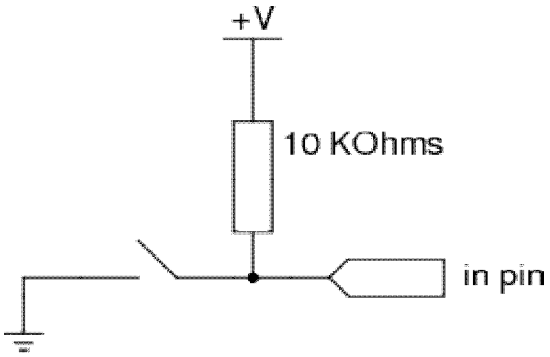



Рисунок 5 – Подключение кнопки к МК
через подтягивающий резистор

То же самое: используются резисторы больших номиналов (10 кОм и более), чтобы минимизировать потери энергии при замкнутой цепи и предотвратить короткое замыкание при разомкнутой.

Микроконтроллер ATmega в Arduino имеет внутренние подтягивающие резисторы (резисторы, подключенные к питанию внутри микросхемы), которыми можно управлять. Если нужно их подключить вместо внешних резисторов, подключенных к земле, необходимо использовать параметр INPUT_PULLUP в функции pinMode(). Это позволит инвертировать поведение подключенного к выводу внешнего датчика: HIGH будет означать его отключение, а LOW – включение.

Функция digitalWrite(pin, value) – устанавливает состояние вывода, сконфигурированного как выход, ничего не возвращает.

Аргументы:

– pin – номер вывода;

– value – состояние выхода.

Аргумент value может принимать одно из возможных значений, представленных в таблице 2.

Таблица 2 – Значения аргумента value

| | |
|---|---|
| value = LOW | Устанавливает выход в низкое состояние |
| value = HIGH | Устанавливает выход в высокое состояние |

Функция digitalRead(pin) – считывает состояние вывода, сконфигурированного как вход.

Аргументы:

– pin – номер вывода.

Возвращает состояние вывода (таблица 3).

Таблица 3 – Значения, возвращаемые функцией digitalRead(pin)

| | |
|---|---|
| digitalRead(pin) = LOW | Низкий уровень на входе |
| digitalRead(pin) = HIGH | Высокий уровень на входе |

Одним из первых проектов, реализуемых с применением любого МК, является «мигание» светодиодом, поэтому на примере подключения светодиода к плате Arduino изучите особенности работы со средой разработки Arduino IDE.

Светодиод – разновидность диода, который светится, когда через него проходит ток от анода (+) к катоду (−) (рисунок 6).

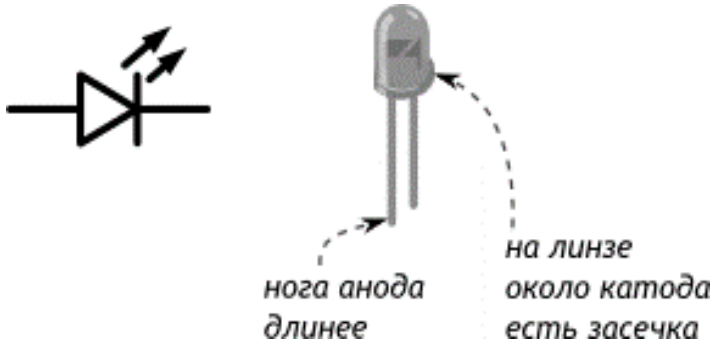



Рисунок 6 – Светодиод и его
условно графическое обозначения

Собственное сопротивление светодиода после насыщения очень мало, и без резистора, ограничивающего ток через светодиод, он перегорит. Порядок: «резистор до» или «резистор после» – не важен.

Схема подключения представлена на рисунке 7.

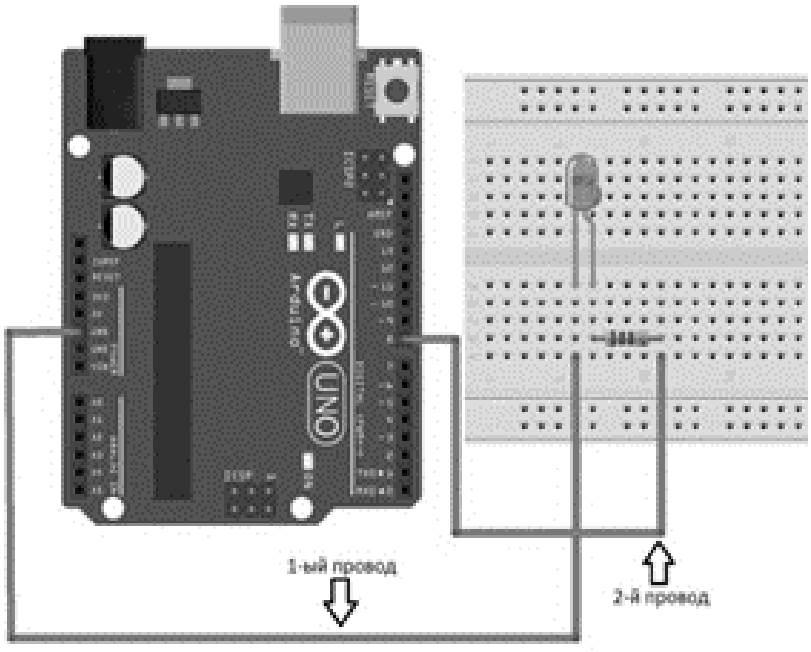



Рисунок 7 – Подключение светодиода

Принципиальная схема данного подключения показана на рисунке 8. В схеме светодиод подключен к выводу 8.

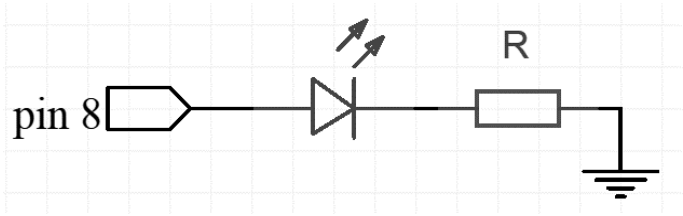



Рисунок 8 – Принципиальная схема подключения

Чтобы яркость свечения светодиода была оптимальной, необходимо выбрать резистор подходящего сопротивления. Для этого необходимо знать напряжение питания в цепи светодиода и изучить параметры выбранного светодиода в его Datasheet (справочный лист с информацией, представляет собой официальный документ производителя электронных компонентов, содержит техническое описание компонента, его параметры, режимы эксплуатации, схемы включения и другую информацию).

Пусть имеется источник питания $U_{cc} = 5$ В и светодиод с рабочим напряжением $U_d = 2{,}3$ В и током $I = 20$ мА. Сначала надо найти, какое напряжение будет падать на резисторе:

$$U_R = U_{cc} - U_d = 5 - 2{,}3 = 2{,}7 \text{ В}.$$

По закону Ома можно найти значение сопротивления, которое обеспечит такое падение:

$$R = \frac{U_R}{I} = \frac{2{,}7}{0{,}02} = 135 \text{ Ом}.$$

Таким образом:

- если взять резистор сопротивлением значительно больше 135 Ом, яркость светодиода будет ниже заявленной;
- если взять резистор сопротивлением менее 135 Ом, срок жизни светодиода будет меньше.

Чтобы не заниматься расчётами резистора, каждый раз во время проведения экспериментов можно просто запомнить правило для самого типичного сценария.

Для питания одного светодиода на 20 мА от 5 В можно использовать резистор от 150 до 360 Ом.

В рассматриваемой схеме управление свечением светодиода осуществляется с помощью вывода 8, поэтому необходимо в начале работы инициализировать его как выход. Это делается в функции setup(). Ниже представлен код программы.

```
void setup() {
   //инициализируем цифровой вывод в режиме выхода
    pinMode(8,OUTPUT);
}
void loop() {
   digitalWrite(8,HIGH); //включение светодиода
   delay(1000); //задержка в 1 с, в течение которой
светодиод горит
   digitalWrite(8,LOW); //выключение светодиода
   delay(500);
}
```

В функции setup() вызывается функция pinMode(), в значении аргумента которой устанавливается 8 пин.

Далее управление переходит к функции loop(), являющейся бесконечным циклом, она вызывается сразу после setup(). В рассматриваемом примере первой командой вызывается функция digitalWrite(), она формирует на выводе 8 уровень логической единицы. При этом на светодиод подается напряжение и зажигает его.

Затем следует команда delay(), она предназначена для остановки выполнения программы на некоторое время. В круглых скобках устанавливается время задержки в миллисекундах.

Можно описать бесконечное выполнение программы так:

1) светодиод зажигается;

2) программа останавливается на 1000 мс;

3) светодиод выключается;

4) программа останавливается на 500 мс.

**По аналогии с примером выполните следующие задания самостоятельно:**

1) подключите два светодиода к плате и сделайте так, чтобы они светились в противофазе (когда первый горит, второй не горит и наоборот);

2) подключите два светодиода к плате и сделайте так, чтобы первый светодиод мигал с интервалом 2 с, а второй – с интервалом 4 с.

Теперь усложним задачу и сделаем так, чтобы светодиод загорался только после нажатия кнопки. Кнопку подключим к другому выходу, например 2. Аппаратная часть схемы подключения кнопки должна обеспечивать уровни напряжений 5 В при отпущенной кнопке и 0 В при нажатой. Для корректной работы кнопки ее надо подключать к МК через подтягивающий или стягивающий резистор, как показано на рисунках 4 и 5. Но для начала попробуем воспользоваться встроенными в МК резисторами, которые активируются заданием функции pinMode() аргумента INPUT-PULLUP. Схему подключения светодиода оставим без изменений. В результате собранная схема должна выглядеть как на рисунке 9.

Тактовая кнопка – простой механизм, замыкающий цепь пока есть давление на толкатель. Кнопки с четырьмя контактами стоит рассматривать как две пары контактов, которые соединяются при нажатии.

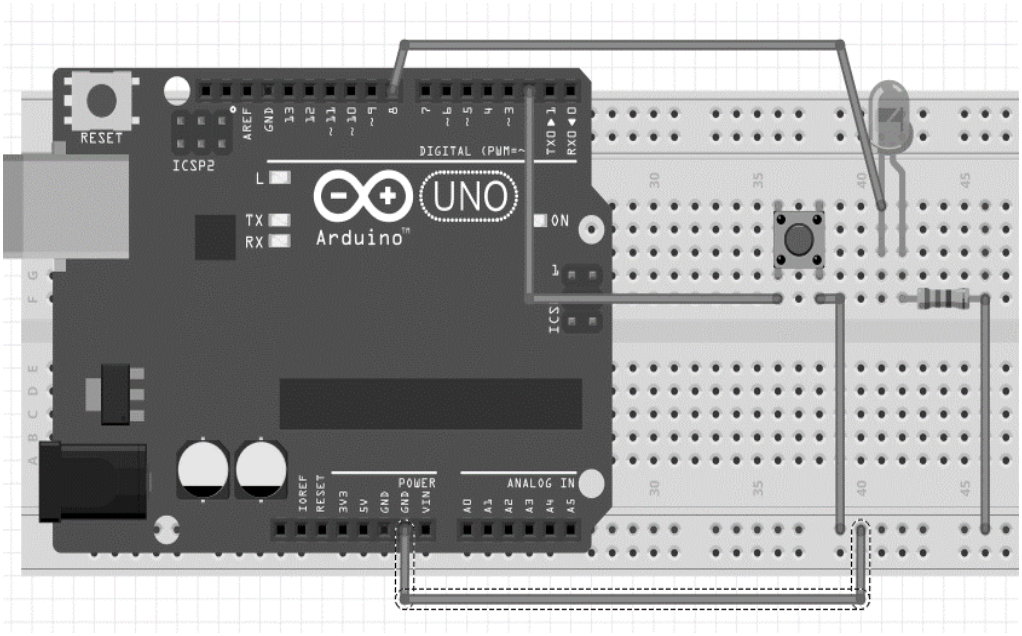



Рисунок 9 – Схема подключения кнопки и светодиода

Ниже приведен код программы с кнопкой, так чтобы при нажатой кнопке светодиод горел, а при отпущенной нет.

```
void setup() {
   pinMode(8,OUTPUT);//инициализируем цифровой выход в
режиме выхода
   pinMode(2,INPUT_PULLUP); //инициализируем цифровой
выход в режиме входа
 }
void loop() {
   if(digitalRead(2)==LOW) digitalWrite(8,HIGH);
   else digitalWrite(8,LOW);
 }
```

Как и в предыдущем задании, сначала инициализируются выводы, 8-й пин останется в режиме выхода, а 2-й пин, к которому подключена кнопка, инициализируется как вход с включенным подтягивающим резистором.

В функции loop() задано условие для кнопки: если кнопка нажата, светодиод включается, иначе он выключен.

Значение с кнопки считывается с помощью функции digitalRead().

Если же теперь отключить встроенный подтягивающий резистор и изменить схему, чтобы она выглядела как на рисунке 10, то логика работы изменится: пока кнопка отпущена светодиод горит, если нажата, то не горит. При этом необходимо изменить для пина 8 аргумент INPUT-PULLUP на INPUT.

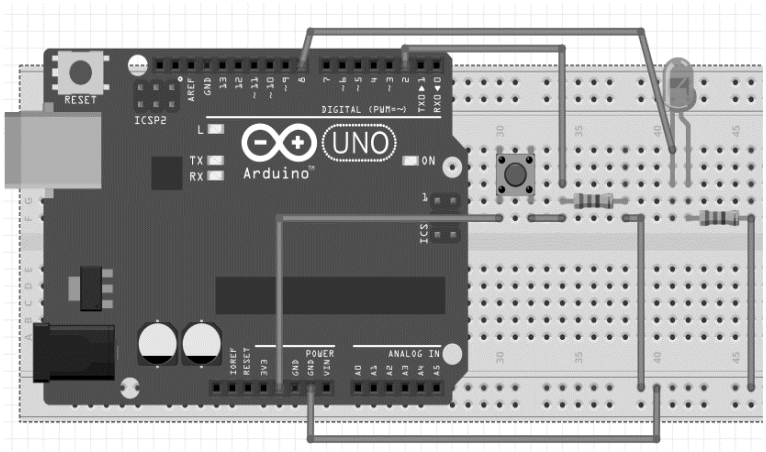



Рисунок 10 – Подключение кнопки
со стягивающим резистором

Оставив схему неизменной, напишем программу, которая нажатием кнопки переводит светодиод в противоположное со-

стояние. Определить момент нажатия кнопки несколько сложнее, чем факт того, что кнопка сейчас просто нажата. Для определения клика сначала необходимо понять, отпущена ли кнопка в данный момент времени:

```
#define BUTTON_PIN 2
#define LED_PIN 8
boolean ledEnabled = false; // переменная хранит состояние
светодиода, по умолчанию он выключен
void setup()
{
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
}
void loop()
{
   boolean buttonIsUp = digitalRead(BUTTON_PIN); //считываем
состояние кнопки
   if (!buttonIsUp) {
   ledEnabled = !ledEnabled; //если состояние кнопки
изменилось, то меняем состояние светодиода
   digitalWrite(LED_PIN, ledEnabled);
 }
}
```

Проверяя работу программы, можно заметить, что переключение светодиода происходит не всегда. Это связано с явлением дребезга или ложного сигнала.

Дребезг контактов – явление, происходящее в электромеханических коммутационных устройствах и аппаратах (кнопках, реле, герконах, переключателях, контакторах, магнитных пускателях и др.). После замыкания электрических контактов происходят многократные неконтролируемые замыкания и размыкания контактов за счет упругости материалов и деталей контактной системы – некоторое время контакты отскакивают друг от друга при соударениях, размыкая и замыкая электрическую цепь [4].

При нажатии на тактовую кнопку, перед тем как контакты плотно соприкоснутся, они будут колебаться (т. е. «дребезжать»), порождая множество срабатываний вместо одного. Соответственно, микроконтроллер зарегистрирует все эти нажатия, потому что дребезг не отличим от настоящего нажатия на кнопку.

Проблему дребезга можно решить, используя как программные способы, так и аппаратные.

Напишем программу, устраняющую проблему дребезга за счет введения задержки и анализа предыдущего положения кнопки:

```
#define BUTTON_PIN 2
#define LED_PIN 8
boolean buttonWasUp = true; // была ли кнопка отпущена?
boolean ledEnabled = false; // включен ли свет?
void setup()
 {
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
 }
 void loop()
 {
    boolean buttonIsUp = digitalRead(BUTTON_PIN);
    if (buttonWasUp && !buttonIsUp) {// проверяем,
действительно ли кнопка нажимается после предыдущего
нажатия или это ложный сигнал, поэтому вводим
задержку для кнопки, чтобы она полностью переключилась
    delay(10);
    buttonIsUp = digitalRead(BUTTON_PIN); //считываем
сигнал снова
    if (!buttonIsUp) { // если кнопка всё ещё нажата, то
переключаем сигнал светодиода
    ledEnabled = !ledEnabled;
    digitalWrite(LED_PIN, ledEnabled);
 }
 }
 buttonWasUp = buttonIsUp; //запоминаем последнее
состояние кнопки
}
```

В данной программе проверяется условие на дребезг в течение 10 мс. Если это все-таки ложный сигнал, то условие включения/выключения светодиода не выполняется.

**По аналогии с примером выполните следующие задания самостоятельно:**

1) с помощью одной кнопки реализуйте возможность управления тремя состояниями светодиода: горит, мигает, выклю-

чен, по аналогии с новогодней гирляндой. Выполните задание с учетом дребезга;
2) используя свободные цифровые выходы Arduino, подключите не менее трех светодиодов (с токоограничительными резисторами) и реализуйте на них эффект «бегущего огня»;
3) с помощью нажатия кнопки реализуйте смену направления «бегущего огня».

# Лабораторная работа №2
# Работа с интерфейсами микроконтроллеров

## Цель работы

Целью данной лабораторной работы является изучение интерфейсов МК для подключения с их помощью периферийного оборудования.

## Описание лабораторной работы

Микроконтроллер, помимо вывода общего назначения GPIO, часто имеет набор некоторых стандартизированных интерфейсов для подключения дополнительного оборудования. Состав и количество этих интерфейсов определяется производителем МК, но в некоторых случаях существует возможность подключать устройства с нестандартными интерфейсами за счет применения плат расширения. Платформа Arduino обычно оснащается следующими интерфейсами [5]:

- UART – универсальный асинхронный приёмопередатчик, который преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по одной физической цифровой линии другому аналогичному устройству. В Arduino для его реализации используются контакты 0 (RX) и 1 (TX). Для использования в своих проектах этого интерфейса может быть применена библиотека «Serial»;
- SPI – последовательный синхронный протокол передачи данных, используемый микроконтроллерами для обмена данными с одним или несколькими периферийными устройствами на небольших расстояниях. Для организации соединения SPI необходимо одно ведущее устройство, обычно это микроконтроллер, которое управляет соединением с ведомыми устройствами. Подключение осуществляется тремя общими линиями и линией выбора периферийного устройства: Master In Slave Out (MISO), переводится как «вход ведущего выход ведомого», используется для передачи данных от ведомого к ведущему; Master Out Slave In (MOSI) – выход ведущего вход ведомого, для передачи данных от ведущего к периферийным устройствам. Serial Clock (SCK) – синхронизирующая линия, синхросигнал генерируется ведущим

устройством. Slave Select pin – вход на ведомых устройствах, с помощью которого ведущий может инициировать обмен данными с периферийным устройством. Если на этом входе LOW, то ведомый взаимодействует с ведущим, если HIGH, то ведомый игнорирует сигналы от ведущего. В Arduino для SPI используются выводы: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). При использовании в своих проектах этого интерфейса может быть применена библиотека «SPI»;

- I2C – последовательная асимметричная шина для связи между интегральными схемами внутри электронных приборов. Использует две двунаправленные линии связи, применяется для соединения низкоскоростных периферийных компонентов с процессорами и микроконтроллерами. Один провод – шина данных (SDA – Serial DAta), второй – тактирование (SCS – Serial CLock), причем на одну шину можно подключать несколько различных устройств. В Arduino для I2C используются выводы A4 (SDA) и A5 (SCL). При использовании в своих проектах этого интерфейса может быть применена библиотека «Wire».

В лабораторной работе предлагается поработать с интерфейсом I2C, подключив к Arduino датчик для измерения атмосферного давления и температуры BMP180 и с датчиком температуры и влажности DHT22, подключаемым к GPIO.

BMP180 [6] представляет собой датчик для измерения атмосферного давления и температуры окружающего воздуха. Поскольку между давлением и температурой существует взаимосвязь, то последняя учитывается при расчете давления. Пределы измеряемого давления составляют 300–1100 мбар, или 225–825 мм рт. ст. Датчик продается в виде модуля GY-68 под Arduino, представляет собой миниатюрную плату, на которой установлен стабилизатор напряжения 3,3 В, обвязка и непосредственно сам датчик. Датчик представлен на рисунке 1, а на рисунке 2 представлена его распиновка.

Напряжение питания датчика может находиться в пределах 1,8–3,6 В. Обмен данными осуществляется с помощью интерфейса I2C.

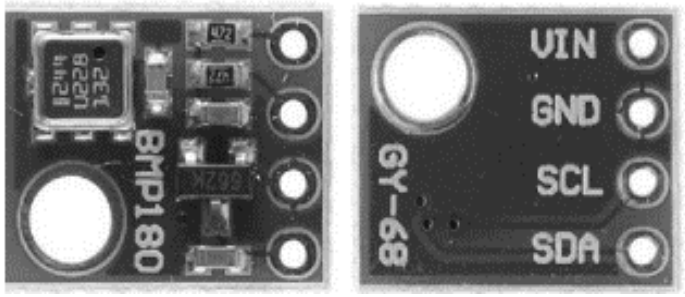



Рисунок 1 – Датчик BMP180

| № | Имя | Выполняемая функция |
|---|---|---|
| 1 | VIN | Плюс питания 2,7V…5,5V |
| 2 | GND | Общий (минус питания) |
| 3 | SCL | i2c Clock – побитное тактирование |
| 4 | SDA | i2c Data – вход / выход данных |

Рисунок 2 – Распиновка датчика

Перед тем как начать работу с датчиком, требуется установить необходимые библиотеки. Для этого нужно нажать на вкладку Инструменты и выбрать Управление библиотеками (рисунок 3).

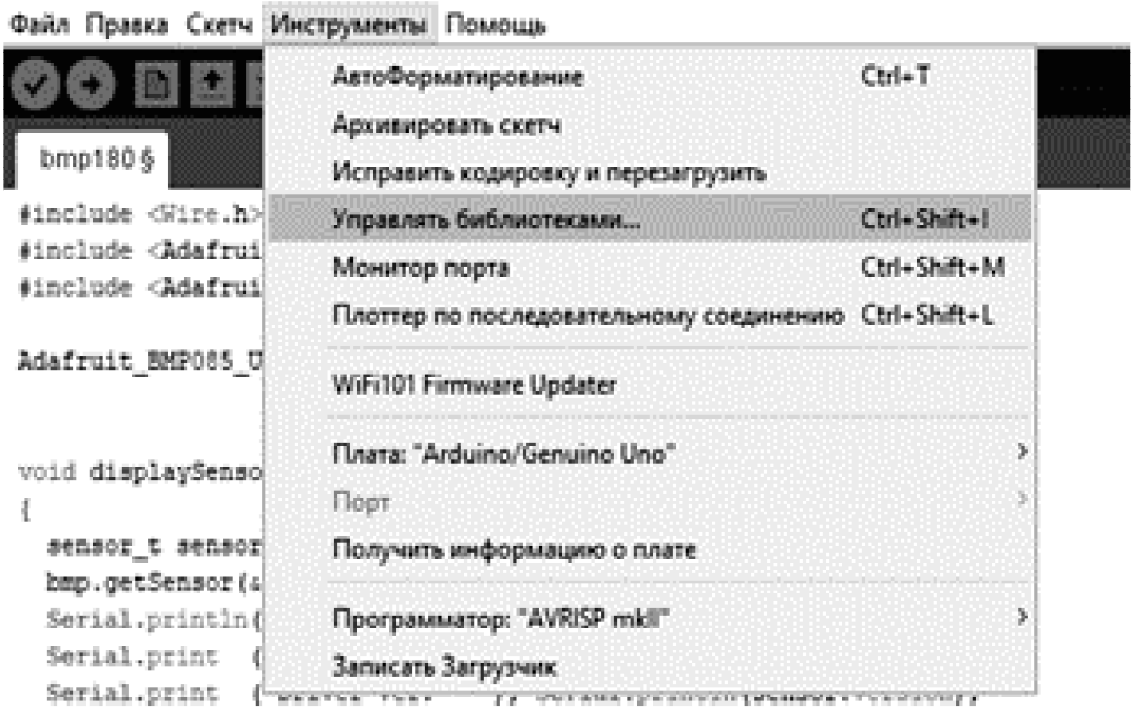



Рисунок 3 – Подключение библиотеки

Далее в строке поиска вводим имя библиотеки и устанавливаем нужную версию (рисунок 4).

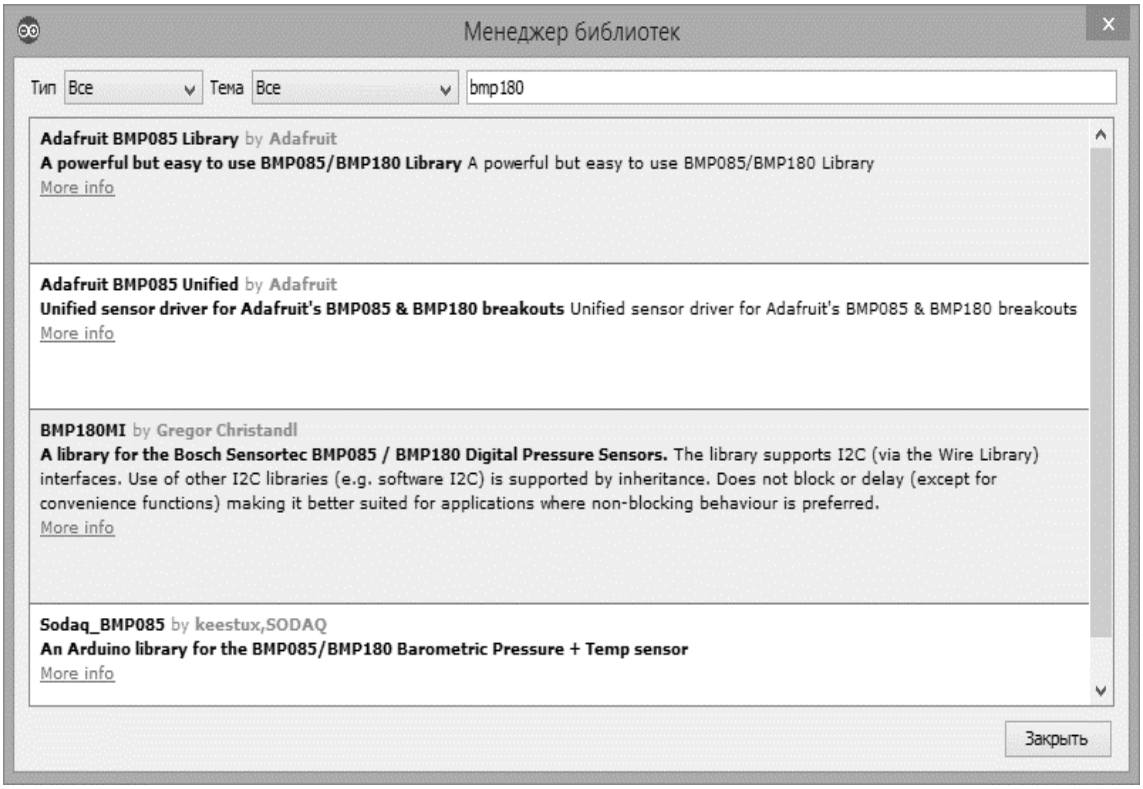


Рисунок 4 – Подключение библиотеки

Если вы работаете с версией Arduino IDE 2:1.0.5+dfsg2-4, последовательность действий несколько изменяется. Необходимо скачать библиотеку по ссылке, распаковать архив. В имени папки с библиотекой оставить только буквы. После этого в менеджере библиотек указать путь до папки с библиотекой. Если все сделано правильно, то библиотека будет добавлена в меню Скетч->Подключить библиотеку.

Для выполнения работы необходимо установить библиотеки:

- Adafruit BMP085 Unified [7] – библиотека для работы с сенсором BMP085, к проекту библиотека подключается строкой #include <Adafruit_BMP085_U.h>;
- Adafruit Unified Sensor [8] – библиотека унифицированных датчиков обеспечивает общий интерфейс и тип данных для любого поддерживаемого датчика. Она определяет некоторую базовую информацию о датчике (пределы датчика и т. д.) И возвращает стандартные единицы СИ определенного типа и шкалы для

каждого типа поддерживаемого датчика. К проекту библиотека подключается строкой #include <Adafruit_Sensor.h>;

- для работы с протоколом I2C у Arduino есть штатная библиотека «Wire» [9], которая позволяет взаимодействовать с I2C/TWI-устройствами. Она подключается к проекту строкой #include <Wire.h>. Помимо этого, в работе будет использоваться библиотека Serial для обмена данными между МК и персональным компьютером (ПК) по UART, дополнительно подключать ее не нужно, она встраивается автоматически;
- первым делом необходимо собрать тестовую схему. Принципиальная схема подключения датчика BMP180 представлена на рисунке 5. Внешний вид макета показан на рисунке 6.

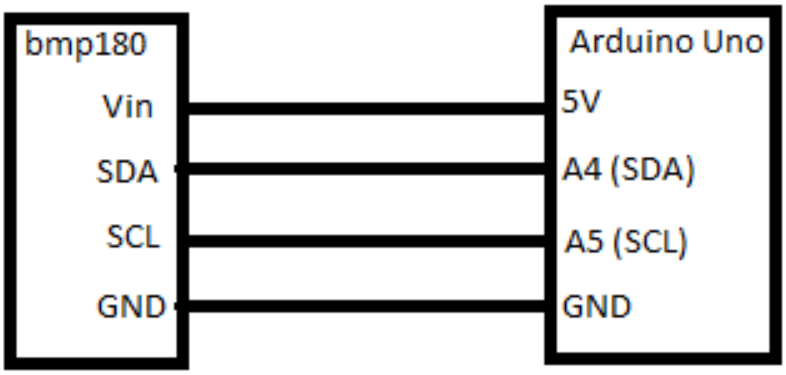



Рисунок 5 – Принципиальная схема подключения датчика к Arduino UNO

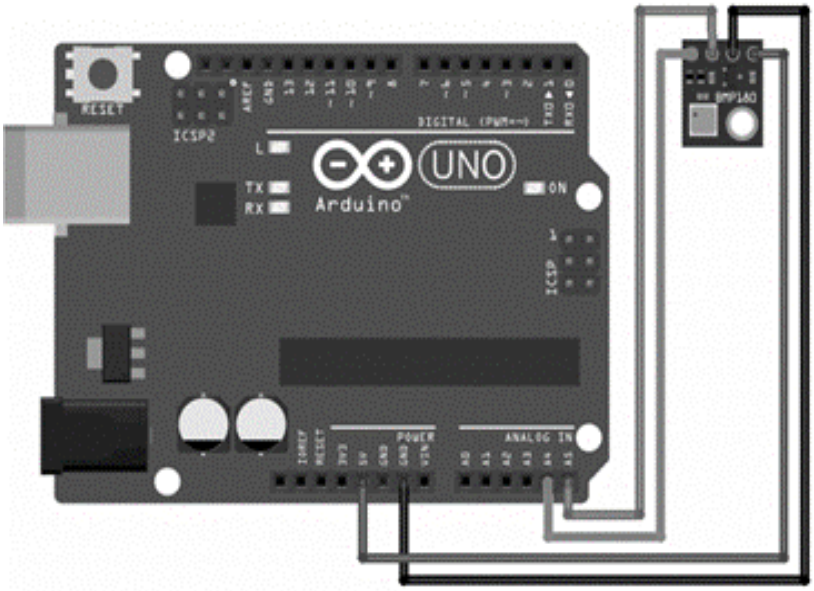



Рисунок 6 – Подключение датчика BMP180

Исходный код программы можно посмотреть в файле BMP180.ino.

Рассмотрим подробнее часть функций, используемых в этом коде. Пользовательская функция displaySensorDetails(void) осуществляет вывод в последовательный порт информации о подключенном сенсоре. Для получения информации о сенсоре используется библиотечный метод getSensor(&sensor). С помощью этого метода можно получить следующую информацию о сенсоре:

- name – имя – определяется используемой библиотекой для конкретного датчика, в нашем случае BMP180;
- version – версия драйвера – служебный параметр используются для идентификации версии библиотеки;
- sensor_id – идентификатор используемого датчика, назначается пользователем для подключения нескольких одинаковых датчиков, так как адрес I2C у них будет совпадать;
- type – тип сенсора, в нашем случае принимает значение SENSOR_TYPE_PRESSURE;
- min_delay – минимальная задержка между считываниями значений с датчика;
- max_value – максимальное значение величины, которое может измерить датчик, для BMP180 составляет 1100 гПа, это значение берется из Datasheet на датчик и записано в библиотеке, обратите внимание, что в используемой версии библиотеки для BMP180 значения констант max_value и min_value перепутаны;
- min_value – минимальное значение величины, которое может измерить датчик, для BMP180 составляет 300 гПа;
- resolution – разрешающая способность датчика, то есть наименьшее значение величины, которое датчик может зарегистрировать, для BMP180 составляет 0,01 гПа.

Функции библиотеки Serial используются для связи устройства Arduino с компьютером или другими устройствами, поддерживающими последовательный интерфейс обмена данными.

Для считывания показаний с датчиков используется метод bmp.getEvent(&event), который заполняет данными ранее

созданную структуру sensors_event_t event. Для датчика BMP180 структура содержит поля:

– version;

– sensor_id;

– type;

– timestamp – отметка времени в миллисекундах, в BMP180 не используется и имеет значение 0;

– pressure – измеренное атмосферное давление в гектопаскалях, для его считывания используется скрытая реализация метода getPressure(&pressure_kPa) библиотеки.

При считывании температуры с датчика используется отдельный метод getTemperature(&temperature).

Чтобы посмотреть результаты работы программы, загрузите на плату Arduino UNO, выберите во вкладке Инструменты Монитор порта для просмотра результатов работы датчика (рисунок 7).

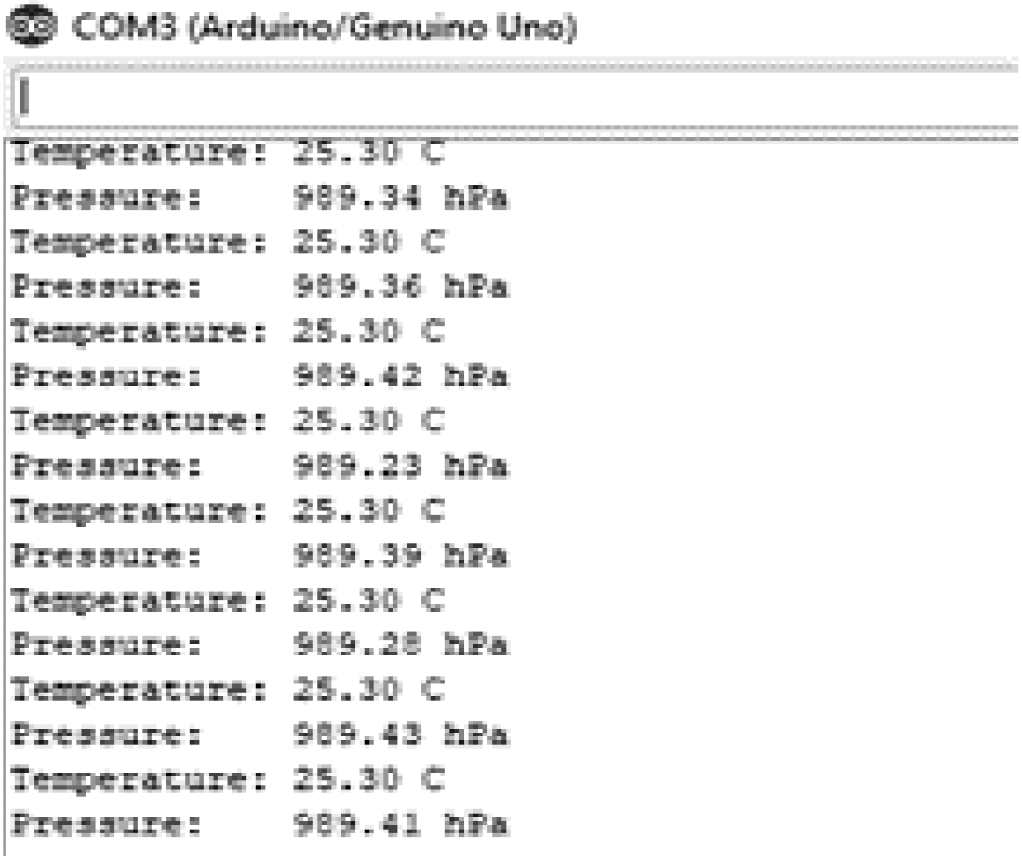



Рисунок 7 – Вывод показаний датчика BMP180

**По аналогии с примером выполните следующие задания самостоятельно:**

1) измените программу таким образом, чтобы вывод показаний давления и температуры осуществлялся в миллиметрах ртутного столба и градусах Фаренгейта соответственно;

2) оформите пересчет единиц измерения как отдельную функцию.

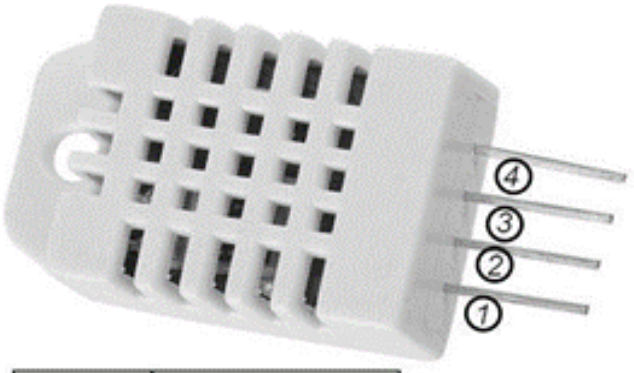
Другим датчиком, используемым в работе, является датчик температуры и влажности DHT22 [10].

Датчик состоит из двух частей – емкостного датчика температуры и гигрометра. Первый используется для измерения температуры, второй – для влажности воздуха. Находящийся внутри чип может выполнять аналого-цифровые преобразования и выдавать цифровой сигнал, который считывается посредством микроконтроллера.

Датчик может питаться от 3,3–6 В, то есть может применяться со всеми платами Arduino. Диапазоны измерения: температуры от –40 до 80 °С, относительной влажности от 0 до 100 %. Типичная погрешность: ±2 % влажности и менее 0,5 °С температуры.

Каждые 2 с датчик способен выдавать обновлённую информацию. Время передачи одного пакета данных – 5 мс.

На рисунке 8 представлены внешний вид и распиновка датчика.




| Контакт | Значение |
|---|---|
| 1 | VCC |
| 2 | DATA |
| 3 | Не используется |
| 4 | GND |

Рисунок 8 – Датчик DHT22

Для работы с датчиком необходимо, используя менеджер библиотек, скачать библиотеку «DHT sensor library by Adafruit» [11] и собрать схему как на рисунке 9. Библиотека DHT.h предназначена для работы с датчиками температуры и влажности: DHT 11, DHT 22, DHT 21, AM2301.

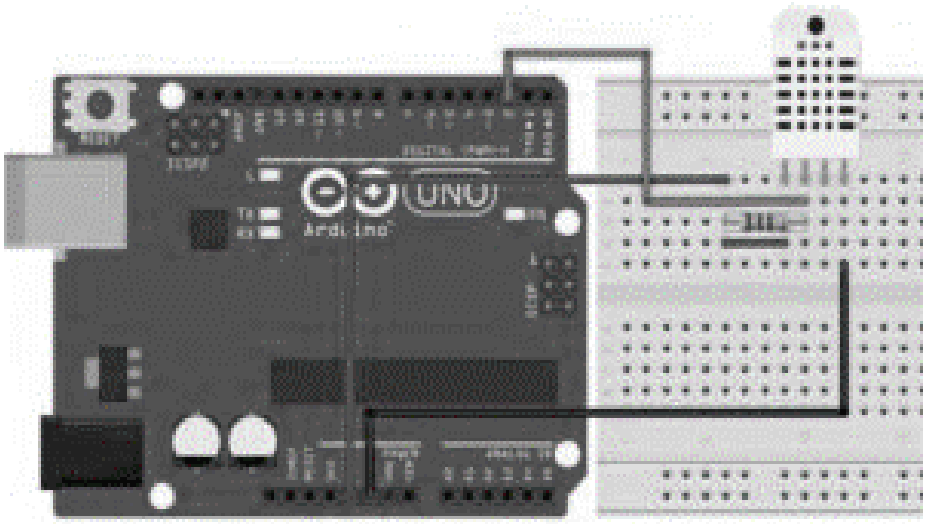


Рисунок 9 – Схема подключения датчика DHT22

О подключении датчика (рисунок 10) стоит сказать следующее: внешний подтягивающий резистор между выводом питания и D2 можно не устанавливать, так как в реализации библиотеки пин подключается в режиме INPUT_PULLUP.

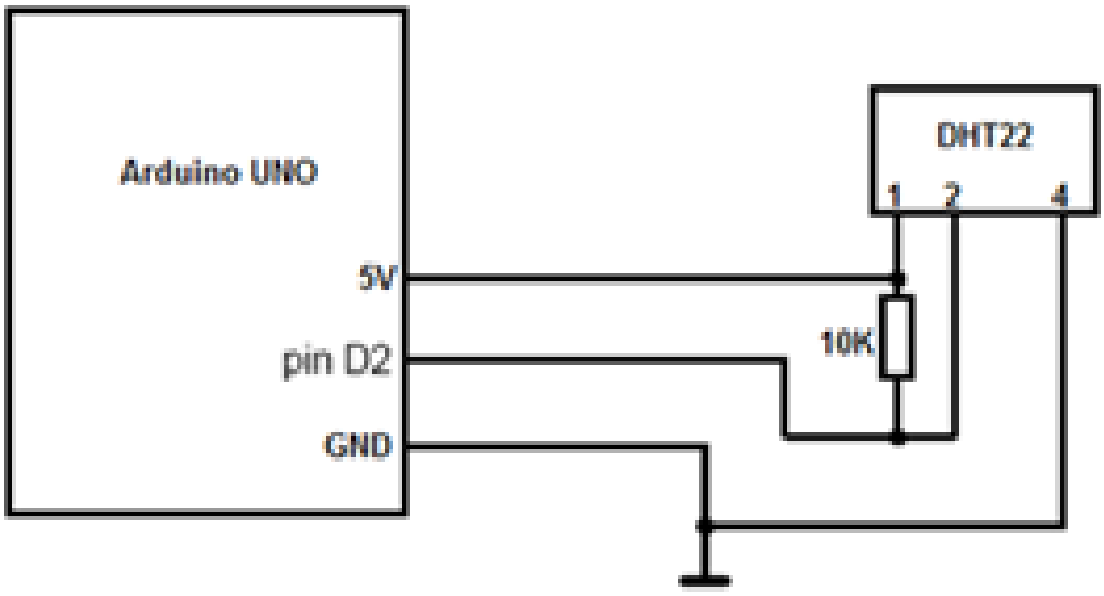



Рисунок 10 – Принципиальная схема подключения

Пример программы для работы с DHT22 представлен в файле DHT_sensor.ino.

Понимание работы функций в примере не должно вызывать трудностей, отдельно стоит отметить условие:

```
if (isnan(h) || isnan(t))
{
    Serial.println("Failed to read from DHT sensor!");
    return;
}
```

Оно использует функцию isnan, которая проверяет, является ли аргумент не числом (NAN). Возвращает значение:

– 0, если проверяемый аргумент – конечное число или бесконечность;

– отличное от нуля значение, если проверяемый аргумент не число (NAN).

Таким образом осуществляется проверка подключения датчика, если датчик не подключен, считывание значения с него возвращает NAN.

Наблюдать результаты работы программы можно, как и в прошлом примере, загрузив программу на плату Arduino UNO, использовать монитор порта для просмотра результатов работы датчика DHT22 (рисунок 11).

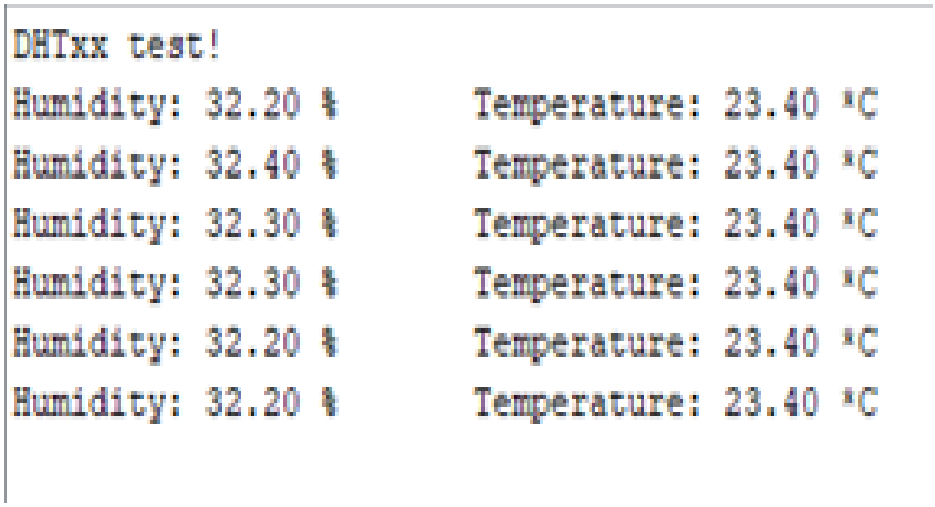



Рисунок 11 – Вывод показаний датчика DHT22

**По аналогии с примером выполните следующие задания самостоятельно:**

1) библиотека содержит методы для преобразования температуры из °F в °C и наоборот, разберитесь с работой этих методов и сделайте вывод температуры в °F;

2) в библиотеке есть интересный метод для вычисления эффективной температуры, изобретенный австралийским ученым Робертом Стедманом, разберитесь с его работой и добавьте вывод эффективной температуры в консоль.

Датчик DHT22 имеет широкие пределы измерений и потенциально может быть использован как уличный датчик температуры и влажности, датчик BMP180, в свою очередь, может быть использован как датчик комнатной температуры и атмосферного давления. Таким образом, с их применением можно создать простейшую метеостанцию, которая будет выводить необходимые сведения в консоль компьютера или пользовательское приложение. Для одновременного подключения датчиков DHT22 и BMP180 к плате Arduino, нужно собрать макет, показанный на рисунке 12.

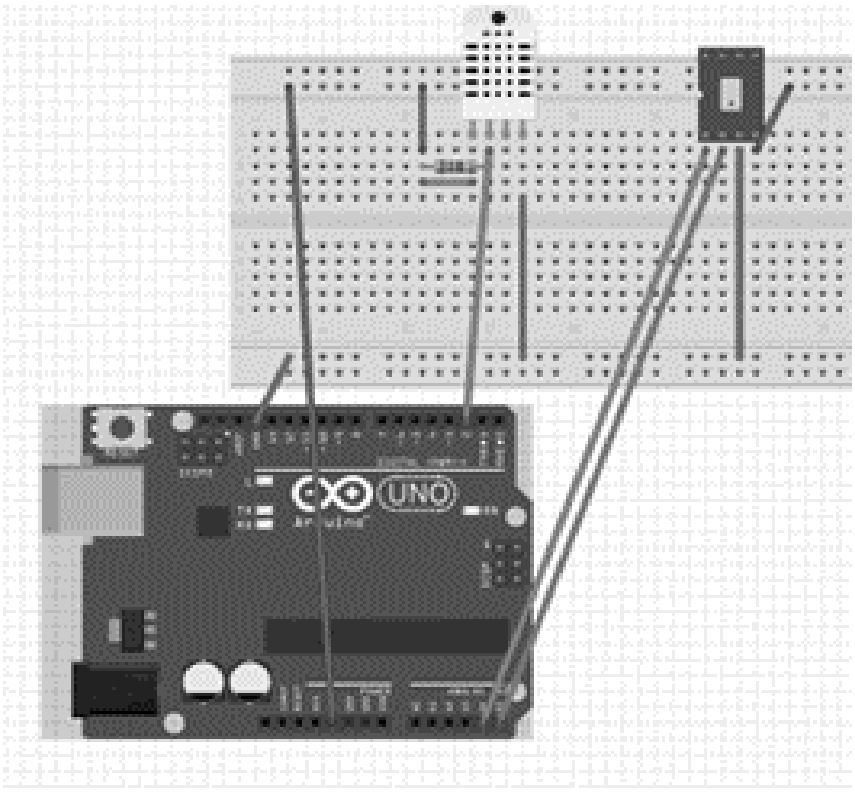



Рисунок 12 – Схема подключения двух датчиков

**Выполните следующие задания самостоятельно:**

1) подумайте, какие параметры вы хотели бы получать со своей метеостанции;
2) напишите программу для вывода в консоль компьютера информации с метеостанции в удобном для восприятия виде.

# Лабораторная работа № 3
# Использование объектно-ориентированного подхода при программировании микроконтроллеров

## Цель работы

Изучение возможностей применения объектно-ориентированного подхода при программировании МК.

## Описание лабораторной работы

Циклы, ветвления и функции – это элементы структурного программирования. Его возможностей достаточно для создания небольших проектов и программ с использованием МК. Однако при разработке сложных проектов удобнее использовать парадигму объектно-ориентированного программирования (ООП) [12]. Что оно из себя представляет и какие преимущества дает?

ООП – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Основными понятиями, используемыми в ООП, являются класс, объект, наследование, инкапсуляция и полиморфизм. Что такое класс? Проведем аналогию с реальным миром. Если взять конкретный стул, то это **объект**, но не класс. А вот общее представление о стульях и их назначении – это класс. Ему принадлежат все реальные объекты стульев, какими бы они ни были. Класс стульев дает общую характеристику всем стульев мире, он их обобщает. **Класс** – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью. Классы позволяют программисту создавать новые типы объектов. Они состоят из свойств и методов. Свойства – это данные, которыми можно характеризовать объект класса. Методы – это функции, которые могут выполнять действия над свойствами класса.

**Объект (экземпляр)** – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом, ему можно посылать сообщения и он может на них реагировать, используя свои данные.

Приступим к изучению такого раздела программирования как ООП. Начнем с создания самых простых классов, постепенно усложняя наши примеры. Вначале будет необходимо сосредоточить свое внимание на частностях, касающихся создания классов и объектов, однако в конце будет рассказано и об общих аспектах объектно-ориентированного подхода к программированию.

Первый пример содержит класс и объект этого класса. Несмотря на свою простоту, он демонстрирует синтаксис и основные черты классов C++. Листинг программы приведен ниже.

```
class LED{ //определение класса
   private:
   int pin_1;
   public:
   int pin; // поле класса
   void led_high() //метод класса
      { digitalWrite(pin,HIGH);}
   void led_low() //метод класса
      { digitalWrite(pin,LOW);}
};
LED work; //создание объекта work, принадлежащего классу
LED
void setup() {
   work.pin = 8; //инициализация выводов для работы с
объектом
   pinMode(work.pin, OUTPUT);
}
void loop() {
   work.led_high(); //объект work вызывает метод,
зажигающий светодиод
   delay(1000);
   work.led_low(); //объект work вызывает метод, гасящий
светодиод
   delay(1000);
}
```

Класс LED, определенный в этой программе, содержит одно поле данных и два метода (функции).

Объединение данных и функций является стержневой идеей объектно-ориентированного программирования. Объект является экземпляром класса, так же как автомобиль является

экземпляром колесного средства передвижения. Что это означает? Рассмотрим следующую аналогию. Практически все компьютерные языки имеют стандартные типы данных; например в C++ есть целый тип int. Можно определять переменные таких типов в своих программах:

int day;

int count.

Подобным же образом можно определять объекты класса, как показано на рисунке 1. Класс является своего рода формой, определяющей, какие данные и функции будут включены в объект класса. При объявлении класса не создаются никакие объекты этого класса по аналогии с тем, что существование типа int еще не означает существование переменных этого типа.

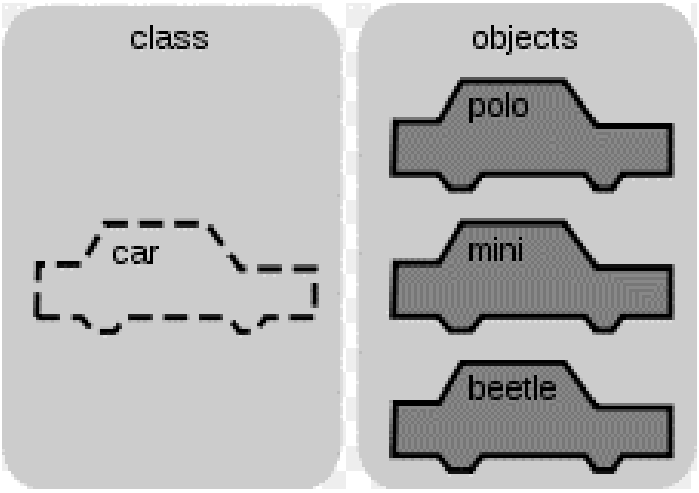



Рисунок 1 – Класс и объект

Таким образом, класс является описанием совокупности сходных между собой объектов. Это соответствует не строгому, в техническом смысле, пониманию термина «класс»: например Prince, Sting и Madonna относятся к классу рок-музыкантов. Не существует конкретного человека с именем «рок-музыкант», однако люди со своими уникальными именами являются объектами этого класса, если они обладают определенным набором характеристик .

Рассмотрим подробнее первую часть программы, где происходит определение класса LED. Затем обратимся к функции loop(), в которой задействованы объекты класса LED.

Определение класса LED в приведенной выше программе выглядит следующим образом:

```
class LED{ //определение класса
 private:
  int pin_1;
  public:
  int pin; // поле класса
  void led_high() //метод класса
    { digitalWrite(pin,HIGH);}
 void led_low() //метод класса
    { digitalWrite(pin,LOW);}
 };
```

Определение начинается с ключевого слова class, за которым следует имя класса; в данном случае этим именем является LED. Подобно структуре, тело класса заключено в фигурные скобки, после которых следует точка с запятой (;) (не забывайте ставить этот знак). Конструкции, связанные с типами данных, такие как структуры и классы, требуют после своего тела наличия точки с запятой, в отличие от конструкций, связанных с передачей управления, например функций и циклов.

Для взаимодействия между объектами и классами используются методы, которые представляют собой интерфейс класса. **Интерфейс** – это набор методов класса, доступных для использования другими классами.

Принято названия классов писать в смешанном регистре, начиная с большой буквы, а названия методов – в смешанном регистре, начиная с маленькой буквы, первая часть – глагол.

Тело класса содержит два не встречавшихся раньше ключевых слова: **private** и **public**. Сейчас раскроем их назначение.

Ключевой особенностью объектно-ориентированного программирования является возможность сокрытия данных. Этот термин понимается в том смысле, что данные заключены внутри класса и защищены от несанкционированного доступа функций, расположенных вне класса. Если необходимо защитить какие-либо данные, то их помещают внутрь класса с ключевым словом private. Такие данные доступны только внутри класса. Данные, описанные с ключевым словом public, напротив, доступны за

пределами класса. Вышесказанное проиллюстрировано на рисунке 2.

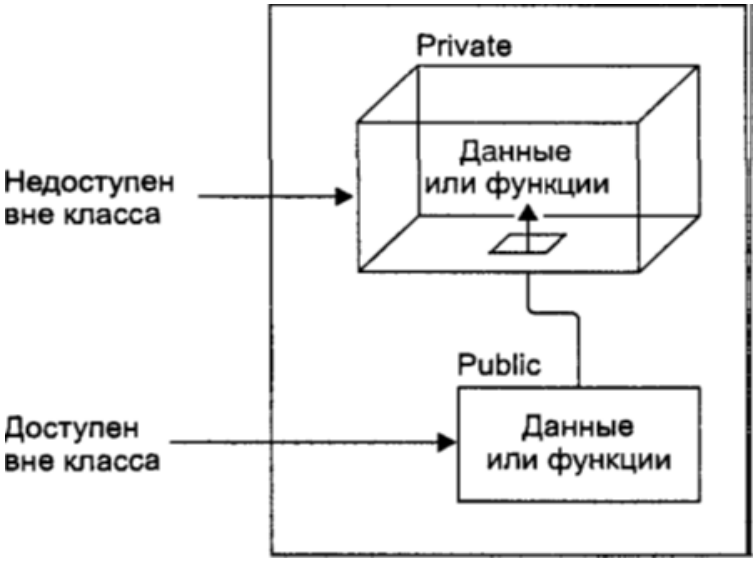



Рисунок 2 – Демонстрация сокрытия данных

Зачем скрывать данные? Не путайте сокрытие данных с техническими средствами, предназначенными для защиты данных. В последнем случае для обеспечения сохранности данных можно, например, попросить пользователя ввести пароль перед тем, как разрешить ему доступ к базе данных. Пароль обеспечивает защиту базы данных от несанкционированного или злоумышленного изменения, а также копирования и чтения ее содержимого.

Сокрытие данных в толковании ООП означает ограждение данных от тех частей программы, которые не имеют необходимости использовать эти данные. В более узком смысле это означает сокрытие данных одного класса от другого класса. Сокрытие данных позволяет уберечь опытных программистов от своих собственных ошибок. Программисты могут сами создать средства доступа к закрытым данным, что значительно снижает вероятность случайного или некорректного доступа к ним.

Необходимо выбрать, какие переменные (свойства) сделать открытыми, а какие – закрытыми. Хороший стиль предполагает, что все переменные должны быть закрытыми. А обращение к ним происходит через методы (функции) класса. Но надо сделать поправку на то, что у микроконтроллера производительность огра-

ничена. А вызов любой функции занимает определенное время. Поэтому свойства, к которым программа обращается часто, следует сделать открытыми и обращаться к ним явно. Это уменьшит время выполнения программы.

Важной частью ООП является принцип наследования – это процесс создания новых классов, называемых наследниками, или производными классами, из уже существующих или базовых классов. Производный класс получает все возможности базового класса, но может также быть усовершенствован за счет добавления собственных. Базовый класс при этом остается неизменным. Выигрыш от него состоит в том, что наследование позволяет использовать существующий код несколько раз. Имея написанный и отлаженный базовый класс, мы можем его больше не модифицировать, при этом механизм наследования позволяет нам приспособить его для работы в различных ситуациях. Используя уже написанный код, мы экономим время и деньги, а также увеличиваем надежность программы. Наследование может помочь и при начальной постановке задачи программирования, разработке общей структуры программы. Ниже представлен пример создания класса LED1, являющегося потомком класса LED

```
class LED{ //определение класса
    private:
     int pin_1;
    public:
     int pin; // поле класса
    void LED_high()
        { digitalWrite(pin,HIGH);}
    void led_low()
        { digitalWrite(pin,LOW);}
  };
 class LED1 : public LED
  {
   public:
  void leding_1()
      {LED :: led_low();}
  void leding_2()
      {LED::LED_high();}
   };
 LED work;
```

```
LED1 work1; // определение объекта LED1
        void setup() {
            work1.pin = 8;
            pinMode(work.pin, OUTPUT);
        }
        void loop() {
        work1.leding_1();
        delay(1000);
       work1.leding_2();
        delay(1000);
      }
```

Когда создаются элементы класса, у программиста нет возможности присвоить им значения в самом определении класса, компилятор выдаст ошибку. Для этих целей можно создавать отдельный метод класса и вызывать его для каждого экземпляра, воспользоваться конструктором класса. Конструктор – это специальный метод класса, который предназначен для инициализации элементов класса некоторыми начальными значениями. Важно запомнить:

1) конструктор объявляется в разделе public;

2) при объявлении конструктора тип данных возвращаемого значения не указывается!

3) имя класса и конструктора должно совпадать;

4) в классе допустимо создавать несколько конструкторов, если это необходимо. Их имена будут одинаковыми. Компилятор будет их различать по передаваемым параметрам (как при перегрузке функций). Если не передавать в конструктор параметры, он считается конструктором по умолчанию;

Конструктор класса для созданного в работе класса LED может выглядеть, например, так

```
class LED{ //определение класса
   private:
     int pin_1; //создаем скрытую переменную для указания
номера пина
   public:
    LED(){ //создаем конструктор класса для задания
начальных параметров экземпляра класса
     pin_1=8;
```

```
        pinMode(pin_1, OUTPUT);
        digitalWrite(pin_1, HIGH);
      }
        void led_high() //метод класса
          { digitalWrite(pin_1,HIGH);}
        void led_low() //метод класса
          { digitalWrite(pin_1,LOW);}
    };
    LED work; //создание объекта work, принадлежащего классу
LED
        void setup() {
          delay(3000); //задержка для демонстрации, что в момент
включения конструктор сработал и начальные значения
установлены
        }
       void loop() {
          work.led_low(); //объект work вызывает метод, гасящий
светодиод
        delay(1000);
        work.led_high(); //объект work вызывает метод,
зажигающий светодиод
          delay(1000);
       }
```

В результате работы программы конструктор создает экземпляр светодиода на пине 8, конфигурирует его на вывод и устанавливает в состояние логической 1. Задержка в 3 с в setup() позволяет увидеть, что сразу после включения светодиод действительно горит.

**По аналогии с примером выполните следующие задания самостоятельно:**

1) попробуйте создать класс для кнопки на примере кода из лабораторной работы № 1. Какие методы могут быть у этого класса?

2) создайте конструктор для класса кнопки и реализуйте устранение дребезга как метод класса.

Использование подхода ООП при создании проектов IoT позволяет усовершенствовать процесс разработки, например использовать ранее написанные классы и методы в новых проектах. Только как практически пользоваться созданным классом?

Можно искать нужный участок кода в исходных кодах старых проектов, но при этом возникнет ряд проблем:

– в новую программу надо скопировать описание класса и код методов. При этом необходимо выделить эти блоки из старой программы и ничего не перепутать;

– эти части программы давно проверены, отлажены и забыты. Тем не менее они будут постоянно попадаться на глаза, увеличивать код программы, ухудшать ее структурность и читаемость;

– в эти модули можно случайно занести ошибку, и потом скопировать ошибочный вариант в следующие программы.

Можно пойти по более разумному пути. Красивое и практичное решение – на основе необходимого участка кода создать библиотеку для объекта нужного типа, которую потом можно легко и быстро подключить к новому проекту [13]. Библиотеки позволяют использовать разработанный ранее программный код в различных программах. Таким образом, программист может не разрабатывать часть кода для своей программы, а воспользоваться тем, который входит в состав библиотек.

В языке программирования С код библиотек представляет собой функции, размещенные в файлах, которые скомпилированы в объектные файлы, а те, в свою очередь, объединены в библиотеки. В одной библиотеке объединяются функции, решающие определенный тип задач. Например, существует библиотека математических функций.

Каждая библиотека должна иметь как минимум 2 файла. У библиотеки должен быть свой заголовочный файл (расширение .h), в котором описаны прототипы (объявления) всех функций, содержащихся в этой библиотеке. Кода программы здесь нет. С помощью заголовочных файлов пользователь «сообщает» программному коду, какие библиотечные функции есть и как их использовать. Второй – это файл с исходным кодом (расширение .cpp), он содержит программный код методов.

При компиляции программы библиотеки подключаются линковщиком. Если программе требуются только стандартные библиотеки, то дополнительных параметров линковщику передавать

не надо (есть исключения). Он «знает», где стандартные библиотеки находятся, и подключит их автоматически. Во всех остальных случаях при компиляции программы требуется указать имя библиотеки и ее местоположение.

Библиотека в Arduino – это не что иное, как дополнительный класс. Поэтому прежде всего необходимо определить функции для библиотеки как класс. Выше подробно описано, что нужно сделать.

Рассмотрим, как устроена библиотека для датчика BMP085.

### Заголовочный файл Adafruit_BMP085_U.h

Сначала в файл должна быть записана текстовая информация о библиотеке. Можно сообщить там все, что считаете нужным: как называется, для чего предназначена, как пользоваться, кто ее создал и т. п. Конечно, текст необходимо оформить как комментарий.

Директива #include предписывает компилятору включить в код программы текст из файла, имя которого следует после директивы.

**Замечание!!!** Библиотеки компилируются без дополнительных преобразований, поэтому их нужно писать на C++. Если внутри библиотеки вы захотите использовать какие-нибудь функции или объекты из стандартной библиотеки Arduino, то нужно подключать соответствующий заголовочный файл («Arduino.h» (в старых версиях IDE: «WConstants.h», «WProgram.h»).

Рассмотрим класс «Adafruit_BMP085_Unified».

```
class Adafruit_BMP085_Unified : public Adafruit_Sensor
{
 public:
   Adafruit_BMP085_Unified(int32_t sensorID = -1);

   bool  begin(bmp085_mode_t mode =
BMP085_MODE_ULTRAHIGHRES);
   void  getTemperature(float *temp);
   void  getPressure(float *pressure);
   float pressureToAltitude(float seaLvel, float atmospheric);
   float seaLevelForAltitude(float altitude, float atmospheric);
```

```
    bool  getEvent(sensors_event_t*);
    void  getSensor(sensor_t*);

  private:
    int32_t computeB5(int32_t ut);
    int32_t _sensorID;
};
```

Как видно, здесь появилось новое обозначение unit32_t – это тип целого числа без знака длиной 32 бит, т. е. все равно, что 4 байта – unsigned long, а также есть uint8_t – тип целого числа без знака длиной 8 бит, т. е. все равно, что байт – byte и др. Они показывают только размерность данных, которые они принимают.

### Исходный файл Adafruit_BMP085_U.cpp

В начале файла должна быть размещена та же самая текстовая информацию, как и в заголовочном файле, так как неизвестно, какой из файлов первым будет изучать пользователь.

Далее должны быть записаны директивы #include для включения стандартных функций работы библиотеки и заголовочного файла.

```
#if ARDUINO >= 100
 #include "Arduino.h"
#else
 #include "WProgram.h"
#endif

#ifdef __AVR_ATtiny85__
 #include "TinyWireM.h"
 #define Wire TinyWireM
#else
 #include <Wire.h>
#endif

#include <math.h>
#include <limits.h>

#include "Adafruit_BMP085_U.h"
```

Для корректного подключения библиотеки к проекту необходимо:

1) запустить Arduino IDE;

2) указать папку своих проектов Arduino (например: Файл -> Настройки -> Размещение папки скетчей задать, например, D:\Users\User \Arduino);

3) в этой папке необходимо создать папку libraries (D:\Users\User\Arduino\libraries);

4) в папку libraries перемещать папки с новыми библиотеками, которые создаете;

5) для проверки надо закрыть и заново запустить Arduino IDE. Открыть Скетч -> Подключить библиотеку и посмотреть, что в списке библиотек присутствует новая библиотека Adafruit_BMP085_U;

6) в начале программы включить заголовочный файл директивой: #include <Adafruit_BMP085.h>

После этого библиотекой можно пользоваться.

**По аналогии с примером выполните следующие задания самостоятельно:**

1) изучите, какие методы имеет библиотека для датчика BMP180 и как их использовать;

2) создайте библиотеку на основе образованного ранее класса кнопки.

# Лабораторная работа № 4
# Использование прерываний в работе микроконтроллеров

## Цель работы

Изучить особенности работы аппаратными прерываниями в МК на примере платформы Arduino.

## Описание лабораторной работы

Для корректного выполнения своих функций микроконтроллер должен параллельно опрашивать датчики и анализировать информацию, поступающую от них. Под параллельной обработкой данных понимается не полноценная одновременная обработка данных, а поочередный опрос датчиков с определенным периодом. Однако сама обработка происходит за малый промежуток времени, поэтому для пользователя это выглядит как одновременная работа с несколькими датчиками. Период, с которым происходит опрос датчиков, является величиной постоянной. Причем период опроса датчиков должен быть неизменным, а его изменение может привести как к некорректной обработке датчиков, так и к полной неработоспособности системы.

В лабораторной работе № 1 была использована функция delay() для смены состояния диода через определенные промежутки времени. Однако нужно учитывать, следующие факторы:

– на опрос датчика затрачивается определенное количество времени, незаметное для человека, но существенное для микроконтроллера;

– на выполнение программы в основном цикле тоже затрачивается время;

– функция delay() ставит на паузу программу, то есть пока не пройдет заданный промежуток времени, программа не сможет выполнять никаких действий.

Все это в совокупности будет приводить к тому, что при использовании функции delay() период, с которым должны опрашиваться датчики, будет изменяться, а это может приводить к нежелательным последствиям. Из этой ситуации есть выход – обработка сигналов по прерыванию от аппаратного таймера.

Аппаратное прерывание – это сигнал, сообщающий о каком-то событии. По его приходу выполнение программы приостанавливается и сохраняется, и управление переходит на специальную подпрограмму «обработчик прерываний». После обработки управление возвращается в прерванный код программы или передается нужной функции [9].

С точки зрения программы прерывание – это вызов функции по внешнему, не связанному напрямую с программным кодом событию. Таким событием может быть срабатывание таймера, встроенного в микроконтроллер, это позволит генерировать события с заданной периодичностью, не затрачивая дополнительные ресурсы микроконтроллера. Сигнал прерывания от таймера вырабатывается циклически, с заданным временем периода. Формирует его аппаратный таймер – счетчик с логикой, сбрасывающий его код при достижении определенного значения. Программно установив код для логики сброса, мы можем задать время периода прерывания от таймера.

Аппаратное прерывание может вырабатываться не только по аппаратному таймеру, но и по наступлению какого-либо события или сигнала, поступившего от периферийного устройства. Примером таких сигналов могут быть сигналы от UART о получении символа, который должен быть обслужен немедленно, иначе полученный символ может потеряться. При поступлении нового символа UART вырабатывает сигнал о требуемой обработке.

Для работы с аппаратным таймером можно использовать библиотеку MsTimer2 [14]. Чтобы установить библиотеку, необходимо папку MsTimer2 скопировать в директорию …\Arduino\libraries и перезапустить Arduino IDE. В запущенной среде нажмите Скетч -> Подключить библиотеку и в разделе «Подключенные библиотеки» убедитесь, что появилась библиотека MsTimer2. Теперь для использования библиотеки необходимо только подключить ее как обычную библиотеку: дописать в начале кода #include <MsTimer2.h>.

Для работы с прерываниями по событию дополнительных библиотек не нужно, так как в Arduino уже есть механизм для их обработки.

Простейшим примером программы с использованием аппаратных прерываний является программа, которая будет менять состояния светодиода, отсчитывая время с помощью аппаратного таймера, код которой представлен ниже.

```
#include <MsTimer2.h>
boolean lightState = true; //переменная для хранения состояния светодиода (горит\не горит)
void timerInterrupt(){ //функция обработчика прерываний по таймеру
  lightState = !lightState;
}

void setup(){
  pinMode(LED_BUILTIN, OUTPUT); //инициализируем пин 13, подключенный к светодиоду как выход
  MsTimer2::set(1000, timerInterrupt); //задаем период прерывания по таймеру 1с и функцию для обработки прерывания
  MsTimer2::start(); //разрешаем прерывание по таймеру
}

void loop(){
  if(lightState == true){
    digitalWrite(LED_BUILTIN, HIGH);
  }
  else{
    digitalWrite(LED_BUILTIN, LOW);
  }
}
```

Эта программа работает следующим образом: через равные промежутки времени, отсчитываемые аппаратным таймером, происходит изменение состояния светодиода (высокий или низкий уровень), а в главном цикле – проверка этого признака и изменение состояния самого диода. Глобальная переменная lightState типа boolean и есть тот самый признак состояния, который будет изменяться с помощью таймера аппаратных прерываний. Она проинициализирована значением true, то есть при старте программы светодиод будет гореть. Функция timerInterrupt() – это обработчик прерывания, то есть когда от таймера поступит сигнал о прерывании, действие основного цикла прервется и про-

изойдет вызов этой функции, которая в нашем случае будет менять признак диода. В функции setup происходит установка начальных параметров:

– инициализируем диод как выход;

– задаем параметры для обработки прерываний: период прерываний (1000 мс) и имя функции – обработчика прерываний;

– разрешаем отсчет аппаратного таймера.

В главном цикле осуществляется только проверка состояния светодиода и перевод его состояния, которое соответствует переменной lightState.

В главном цикле выполнится несколько проверок признака без изменения состояния светодиода, потому что время проверки его состояния много меньше заданного периода. Затем, когда произойдет вызов прерывания по таймеру, проверка признака в главном цикле остановится, произойдет вызов обработчика прерывания, который изменит признак диода на противоположный. После этого программа вернется обратно в главный цикл, и при следующей проверке произойдет смена состояния диода.

Как уже отмечалось ранее, одной из задач микроконтроллера может быть обработка нескольких датчиков, причем не всегда с одинаковым периодом. Этот случай разберем на примере программы, которая будет переключать состояния двух диодов с периодами 2 и 3 с соответственно.

Для выполнения этой задачи используем тот же подход, что и в программе, которая изменяет состояния одного диода, но немного дополним его. Для того чтобы управлять двумя диодами, нам понадобятся две переменные признака для этих диодов. Еще потребуется создать два счетчика, задача которых следить за периодами.

Несмотря на то, что в нашем случае работа идет с двумя диодами, обработчик прерываний будет всего один. Теперь нужно определить период, с которым будет происходить вызов этого обработчика. В нашем случае периоды изменения состояний диодов равны 2 и 3 с. Наибольшее число, на которое 2 и 3 делятся нацело, это 0,5. С этим периодом и будет происходить вызов обработчика прерываний. Обработчик прерываний будет вести

счетчики для двух диодов и в случае их переполнения сбрасывать их и менять признаки диодов. Теперь нужно определиться с максимальными значениями счетчиков для двух диодов. Для первого диода период изменения состояния равен 2 с, период вызова обработчика прерывания равен 0,5 с, следовательно, значение счетчика, при котором будет происходить его обнуление равно 2 / 0,5 = 4. Соответственно, для диода, период изменения состояния которого равен 3 с, значение счетчика равно 3 / 0,5 = 6.

```
#include <MsTimer2.h>
byte counter1 = 0; //счетчик таймера, связанный с 1-м
светодиодом
byte counter2 = 0; //счетчик таймера, связанный со 2-м
светодиодом
boolean lightState1 = true;
boolean lightState2 = true;
void timerInterrupt(){
  counter1++; //Увеличение счетчика 1 на 1 при каждом
вызове обработчика
  if(counter1 == 4){ //проверка на переполнение
    lightState1 = !lightState1;
    counter1 = 0;
  }
  counter2++; //Увеличение счетчика 2 на 1 при каждом
вызове обработчика
  if(counter2 == 6){//проверка на переполнение
    lightState2 = !lightState2;
    counter2 = 0;
  }
}
void setup(){
  pinMode(8, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  MsTimer2::set(500, timerInterrupt);
  MsTimer2::start();
}
void loop(){
  if(lightState1 == true){
    digitalWrite(LED_BUILTIN, HIGH);
  }
  else{
    digitalWrite(LED_BUILTIN, LOW);
```

```
    }
    if(lightState2 == true){
    digitalWrite(8, HIGH);
  }
  else{
    digitalWrite(8, LOW);
  }
 }
```

Скомпилировав и запустив программу, вы увидите, что два светодиода будут менять свои состояния с разными периодами, как и было запланировано.

**По аналогии с примером выполните следующие задания самостоятельно:**

1) используя прерывания по таймеру и свободные цифровые выходы Arduino, подключите три светодиода (с токоограничительными резисторами) и заставьте их мигать с периодичностью 1, 3 и 5 с;

2) используя прерывания по таймеру и свободные цифровые выходы Arduino, подключите три-четыре светодиода (с токоограничительными резисторами) и заставьте их работать на манер новогодней гирлянды (2–3 режима, переключаемые по времени).

Для обработки прерываний по событию в Arduino предусмотрены собственные механизмы их обработки. Одним из таких механизмов является функция attachInterrupt(interrupt, function, mode) [15].

Большинство контроллеров Arduino умеют обрабатывать до двух внешних прерываний, пронумерованных так: 0 (на цифровом порту 2) и 1 (на цифровом порту 3). Arduino Mega обрабатывает дополнительно еще четыре прерывания: 2 (порт 21), 3 (порт 20), 4 (порт 19) и 5 (порт 18). На рисунке 1 показан список плат, количество прерываний, поддерживаемых соответствующей платой, и номера входов, на которых они работают.

Функция attachInterrupt(interrupt, function, mode) принимает три аргумента:

– номер прерывания;

– имя функции, выполняемой при получении сигнала о прерывании;

– событие, по которому будет генерироваться сигнал о прерывании.

| Плата | int.0 | int.1 | int.2 | int.3 | int.4 | int.5 |
|---|---|---|---|---|---|---|
| UNO, Ethernet | 2 | 3 | | | | |
| Mega2560 | 2 | 3 | 21 | 20 | 19 | 18 |
| Leonardo | 3 | 2 | 0 | 1 | 7 | |

Рисунок 1 – Прерывания, поддерживаемые различными платами

У функции обработчика прерываний есть некоторые особенности:

– функция не должна принимать и возвращать никаких значений;

– внутри функции не работает определение времени, а это означает, что нельзя использовать delay(), и значения, возвращаемые функцией millis(), не изменяются;

– переменные, с которыми работает функция должны быть объявлены как volatile – ключевое слово языков C/C++, которое информирует компилятор о том, что значение переменной может меняться извне и что компилятор не будет оптимизировать эту переменную. Иначе, компилятор, выполняя оптимизацию кода, может не увидеть, что переменная объявлена в функции, а изменяется в функции, связанной с обработчиком прерываний, посчитает ее ненужной и удалит. Переменные, объявленные как volatile, не будут оптимизироваться компилятором и останутся на своих местах;

– функция обработчик прерывания должна быть как можно короче. Это связано с тем, что Arduino не умеет обрабатывать несколько прерываний одновременно, и если текущее прерывание выполняется слишком долго, то в это время может поступить новый сигнал о прерывании, который будет пропущен.

Для внешних прерываний существует 4 типа событий, по которым генерируются сигналы прерываний:

▪ LOW – вызывается, когда на входе присутствует логический 0;

▪ CHANGE – прерывание вызывается при смене значения на входе с LOW на HIGH и наоборот;

▪ RISING – прерывание вызывается только при смене значения на входе с LOW на HIGH;

▪ FALLING – прерывание вызывается только при смене значения на входе с HIGH на LOW.

Воспользуемся схемой включения светодиода и кнопки из лабораторной работы № 1. Напишем для этой схемы программу, которая будет переключать состояние светодиода по нажатию кнопки, но теперь на сигнал от нажатия кнопки программа будет реагировать как на прерывание. Код программы представлен ниже.

```
volatile boolean lightState = LOW;//переменная для хранения состояния светодиода (горит\не горит), с указанием типа volatile
unsigned long currentMillis; //переменная для записи момента срабатывания прерывания
unsigned long previousMillis = 0; //переменная для хранения момента времени последнего зафиксированного нажатия
void setup(){
  pinMode(13, OUTPUT);
  attachInterrupt(0, blink, RISING); //задаем для 0-го прерывания функцию обработчик blink(), прерывание будет срабатывать по фронту
}
void loop(){
  digitalWrite(13, lightState);
}
void blink(){
  currentMillis = millis(); //получаем текущее значение системного времени
  if(currentMillis - previousMillis > 10){ //если с момента предыдущего входа в прерывание прошло достаточно времени, то нажатие действительно произошло
    previousMillis = currentMillis; //текущее значение системного времени записываем в предыдущее, чтобы сохранить время последнего нажатия
    lightState = !lightState;
  }
}
```

В данном примере аргументы функции attachInterrupt() следующие:

– номер прерывания 0 (так как кнопка подключена к пину 2, а за ним закреплено прерывание именно с этим номером);

– функция обработчик прерывания blink();

– событие вызова прерывания RISING - по фронту (был 0 стала 1).

Функция blink() работает следующим образом: при прерывании берется текущее значение системного времени, для этого используется функция millis() (возвращает количество миллисекунд с момента начала выполнения текущей программы на плате Arduino). Это количество сбрасывается на ноль вследствие переполнения значения (приблизительно через 50 дней) и сравнивается со значением времени, когда прерывание было вызвано в предыдущий раз. Если прошло больше 10 мс (этого времени достаточно, чтобы исключить дребезг контактов), значит это действительно нажатие кнопки и нужно изменить состояние светодиода и записать текущее значение системного времени в предыдущее для работы в следующих вызовах. А если прошло меньше 10 мс, то это ложное срабатывание из-за дребезга, и в этом случае менять состояние светодиода не нужно.

**По аналогии с примером выполните следующие задания самостоятельно:**

1) выполните задание из лабораторной работы № 1: «с помощью кнопки реализуйте возможность управления тремя состояниями светодиода: горит, мигает, выключен; по аналогии с новогодней гирляндой», используя аппаратные прерывания;
2) используя аппаратные прерывания, с помощью нажатия кнопки реализуйте смену направления «бегущего огня» из лабораторной работы № 1;
3) используя аппаратные прерывания, создайте пульт для игры «кто первым нажмет кнопку». Условия: есть две кнопки и два светодиода, связанных с этими кнопками. Нажатие на кнопку зажигает соответствующий светодиод, если после этого нажимать на вторую кнопку, то ничего не произойдет,

и наоборот, запуск второго раунда игры происходит через 5 с после окончания первого, для индикации начала и окончания раунда можно использовать еще один светодиод;
4) используя аппаратные и программные прерывания, реализуйте вывод показаний датчиков метеостанции в консоль по нажатию кнопки, а также через интервал 10 с.

# Лабораторная работа № 5
# Реализация протокола обмена данными с конечным устройством IoT

## Цель работы

Приобретение навыков разработки протокола обмена данными между конечным устройством и компьютером через интерфейс UART на примере метеостанции на Arduino.

## Описание лабораторной работы

Для IoT важно наличие связи между устройствами, будь это связь между двумя конечными устройствами или же связь между конечным устройством и контроллером верхнего уровня. Если представить, что разработанное на предыдущих занятиях устройство является метеостанцией, входящей в состав «умного дома», то саму систему «умного дома» можно рассматривать как малую распределенную систему управления и контроля. Основным компонентом такой простой распределенной управляющей системы можно считать локальный контроллер, который управляет работой метеостанции.

Локальные контроллеры в иерархии системы расположены на самом нижнем уровне. Это контроллеры, к которым непосредственно подключены датчики, исполнительные механизмы и средства локальной индикации. Предполагается, что локальные контроллеры способны работать в автономном режиме. Это значительно повышает надежность системы. При обрыве связи с верхним уровнем локальные контроллеры переходят на автономный режим управления или по крайней мере безопасно останавливают процесс.

Для управления работой локальных контроллеров существует верхний уровень системы. Он объединяет работу контроллеров, управляет всей системой. С него передаются на локальные контроллеры различные команды, параметры, установки, режимы. В качестве устройства верхнего уровня может выступать другой контроллер или персональный компьютер.

Для реализации обмена между устройствами в системе должен быть предусмотрен протокол обмена данными.

Протокол обмена данными – это набор соглашений, правил, которые определяют обмен данными между программами и/или устройствами, в нашем случае между локальным контроллером и центральным контроллером или компьютером. Существуют стандартизированные протоколы передачи данных [16], которые позволяют разрабатывать интерфейсы, не привязанные к конкретной аппаратной платформе и производителю.

Так как малая распределенная система управления и контроля представляет собой соединенные вместе устройства, ее можно рассматривать как небольшую сеть, а, значит, протоколы, используемые в системе, должны соответствовать модели OSI. Модель OSI – это 7-уровневая логическая модель работы сети. Модель OSI реализуется группой протоколов и правил связи, организованных в несколько уровней:

- физический уровень – определяются физические (механические, электрические, оптические) характеристики линий связи;
- канальный уровень – определяются правила использования физического уровня узлами сети;
- сетевой уровень – отвечает за адресацию и доставку сообщений;
- транспортный уровень – контролирует очередность прохождения компонентов сообщения;
- сеансовый уровень – координация связи между двумя прикладными программами, работающими на разных рабочих станциях;
- уровень представления – служит для преобразования данных из внутреннего формата компьютера в формат передачи;
- прикладной уровень – пограничный между прикладной программой и другими уровнями – обеспечивает удобный интерфейс связи сетевых программ пользователя.

При создании сложной системы необходимо тщательно подойти к вопросу разработки протокола передачи данных, чтобы обеспечить максимальную скорость передачи, достоверность данных и иметь возможность для дальнейшей модернизации системы.

В лабораторной работе предлагается рассматривать систему управления, состоящую из метеостанции на Arduino и центрального контроллера на базе ПК. Так как пока мы располагаем только одним устройством, то для реализации протокола обмена можно использовать подключение конечного устройства по USB с использованием протокола UART. В этом случае не нужно заниматься разработкой физического и канального уровня протокола.

Протокол Universal Asynchronous Receiver-Transmitter (UART) – универсальный асинхронный приемопередатчик, старейший и самый распространенный на сегодняшний день физический протокол передачи данных [13]. Представляет собой логическую схему, с одной стороны подключенную к шине вычислительного устройства, а с другой – имеющую два или более выводов для внешнего соединения.

Наиболее известен из семейства UART протокол RS-232 (COM-порт). Основные рабочие линии UART – RXD и TXD, или просто RX и TX. Передающая линия – TXD (Transmitted Data), а порт RXD (Received Data) – принимающая. Активный уровень сигналов низкий. В режиме ожидания сигналы находятся в высоком уровне.

Интерфейс UART радиальный, т. е. по линиям связи одного интерфейса подключаются только два устройства. Выход TX одного устройства подключается к входу RX второго. Сигнал TX второго UART-устройства соединяется с входом RX первого (рисунок 1).

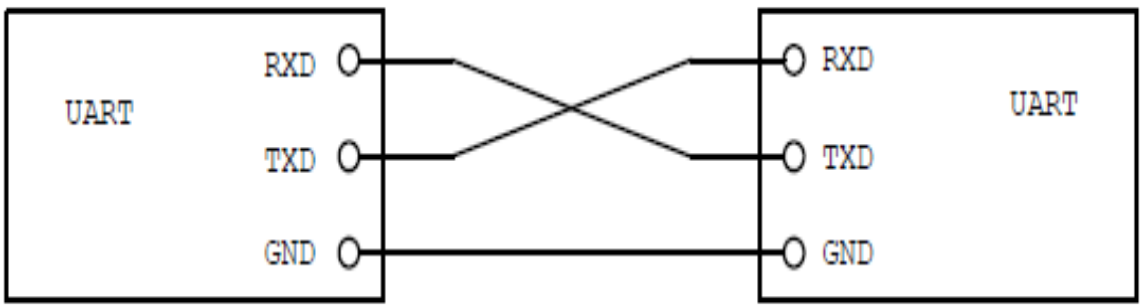



Рисунок 1 – Схема подключения двух UART-устройств

Передача данных в UART осуществляется последовательно, по одному биту в равные промежутки времени. Этот временной промежуток определяется заданной скоростью UART и для конкретного соединения указывается в бодах (что в данном случае соответствует битам в секунду). Существует общепринятый ряд стандартных скоростей: 300; 600; 1200; 2400; 4800; 9600; 19200; 38400; 57600; 115200; 230400; 460800; 921600 бод. Скорость S бод и длительность бита T секунд связаны соотношением T=1/S. Скорость в бодах иногда называют сленговым словом бодрейт или битрейт.

Стандартным размером данных в посылке является 8 бит, но помимо данных каждый пакет несет и служебную информацию, а именно:

- старт-бит (обязателен) – при приёме удаляется из информационного потока, младший информационный бит передаётся сразу после стартового. Стартовый бит соответствует логическому нулю;
- стоп-бит (обязателен) – может иметь длительность 1, 1,5 и 2 бита. Как и старт-бит, стоп-бит при приёме удаляется из потока и соответствует логической единице;
- бит четности (не обязателен) – используется для контроля целостности данных. Когда эта функция включена, последний бит данных в минимальной посылке («бит чётности») контролируется логикой UART и содержит информацию о чётности количества единичных бит в этой минимальной посылке.

Кратко параметры передаваемого сигнала записываются так:

[кол-во бит данных] [тип четности] [кол-во стоп-битов]

Так как интерфейс асинхронный, то большую значимость имеет скорость передачи данных – и у приемника, и у передатчика она должна быть одинаковой.

Порядок передачи данных через интерфейс UART:

1) передающий UART принимает данные от внутренней шины данных и добавляет начальный бит, бит четности и стоп-бит в пакета данных;

2) весь пакет отправляется последовательно от передающего к принимающему UART, который производит выборку линии

данных с заранее сконфигурированной скоростью передачи данных;

3) принимающий UART отбрасывает начальный бит, бит четности и стоповый бит из пакета данных, преобразует последовательные данные и передает их на шину данных на принимающей стороне.

Основные сведения о методе передачи и приема по UART:

– в неактивном (IDLE) режиме обе линии данных (RX и TX) подтянуты к уровню логической единицы;

– передачу начинает стартовый бит (логический ноль);

– передачу заканчивает стоп-бит (логическая единица);

– данные передаются в режиме LSB (младшим битом вперед);

– для передачи байта требуется минимум 10 бит [17].

Все платы Arduino имеют, как минимум, один аппаратный интерфейс UART. Интерфейсы не загружают контроллер, так как они отделены от ядра.

Контроллеры Arduino используют для передачи по UART логические уровни CMOS (уровень логического 0 около 0 В; уровень логической 1 около 5 В).

Для связи платы Arduino с ПК или другими устройствами, поддерживающими последовательный интерфейс обмена данными (UART), служит библиотека Serial [18]. Для обмена данными Serial используют цифровые порты ввод/вывод 0 (RX) и 1 (TX), а также USB-порт. Важно учитывать, что если в контроллере используются функции Serial, то нельзя одновременно с этим использовать порты 0 и 1 для других целей. Ниже описаны некоторые из методов, которые может использовать эта библиотека:

- Serial.begin(speed) – инициирует последовательное соединение и задает скорость передачи данных в бит/с (бод);
- Serial.available() – функция получает количество байт (символов), доступных для чтения, из последовательного интерфейса связи. Это те байты, которые уже поступили и записаны в буфер последовательного порта. Буфер может хранить до 64 байт;
- Serial.read() – считывает очередной доступный байт из буфера последовательного соединения;

▪ Serial.print(val) – передает данные по UART как ASCII-текст. Эта функция может принимать различные типы данных. Так, целые числа выводятся соответствующими им символами ASCII. Вещественные – выводятся с помощью двух ASCII-символов, для целой и дробной части. Байты передаются как символ с соответствующим номером. Символы и строки отсылаются как есть;

▪ Serial.println(val) – как и предыдущая функция передает ASCII-текст, но со следующим за ним символом переноса строки (ASCII символ 13 или '\r') и символом новой строки (ASCII 10 или '\n');

▪ Serial.write(val) – функция передает данные как бинарный код через последовательное соединение. Данные посылаются как один или серия байтов.

В данной работе одним UART-устройством является плата Arduino, вторым – преобразователь интерфейсов USB-UART, подключенный к компьютеру.

Так как в работе осуществляется обращение к одному устройству, то в разрабатываемом протоколе можно приступить к разработке протокола прикладного уровня, опустив остальные уровни.

Протоколы бывают числовыми и текстовыми. Числа, данные в текстовом протоколе, передаются как коды символов. Например, число «132» в текстовом протоколе передается как 3 байта 0x31, 0x33 и 0x32. В числовом протоколе это же число передается одним байтом со значением 132.

Преимущества числовых протоколов:

– для передачи одинакового количества информации им требуется значительно меньше передаваемых данных;

– обработка передаваемых данных требует значительно меньше ресурсов микроконтроллера. Это связано с тем, что нет необходимости преобразовывать текстовые строки в двоичные числа и наоборот.

Недостатком является то, что числовые протоколы можно отлаживать только специальными программами.

Преимущества текстовых протоколов:

– возможность использования для контроля и отладки стандартных средств и программ – текстовых терминалов.

Недостатком текстовых протоколов является необходимость применения большего количества ресурсов контроллера на преобразование текстовых строк, что критично для контроллера невысокой производительности.

При разработке реальных систем чаще всего будут использоваться стандартные протоколы, найти информацию по организации работы которых сегодня совсем не сложно. Но в этой работе будет применяться нестандартный протокол с целью продемонстрировать, что:

– разрабатывать свои протоколы несложно;

– специализированные протоколы, как правило, эффективнее стандартных;

– протокол можно оптимизировать под конкретную задачу.

В качестве локального контроллера будет использована плата Arduino с подключенными датчиками температуры и влажности и температуры и давления, имитирующая работу метеостанции, собранная в предыдущей лабораторной работе.

## Разработка протокола обмена данными локального контроллера

В общем виде любой протокол обмена выглядит так:

– команда: Адрес устройства -> Код операции -> Данные (при наличии)-> Контрольный код (при необходимости);

– ответ: <-Данные<-Контрольный код (при необходимости).

Ведущее устройство, которое инициирует обмен, посылает команду. В общем случае команда должна содержать информацию:

- адрес – адрес устройства, с которым происходит обмен;
- код операции – информация о том, что надо сделать;
- данные, над которыми надо произвести какие-то операции;
- контрольный код – код, позволяющий обнаружить ошибки данных при передаче.

В ответ устройство посылает:

– результат выполнения операции и контрольный код.

Все эти составляющие определяют свойства и качества протокола. Выбирая их, можно оптимизировать протокол под свои требования по следующим критериям:

– минимум данных, передаваемых по каналу связи (высокая скорость передачи данных);

– минимальные требования к ресурсам контроллеров (простота обработки данных);

– высокая надежность передачи данных (требуется повторная передача ошибочных данных);

– высокая достоверность данных, т. е. высокая вероятность обнаружения ошибок передачи. Это требование к контрольному коду.

В рассматриваемом в работе случае поле адреса устройства можно не использовать, так как по UART возможно соединение только двух устройств (хоста и контроллера).

Первым делом надо определиться с тем, какие команды можно отправлять контроллеру. В предыдущих работах было создано устройство, имитирующее работу метеостанции, оно было оснащено датчиками:

– DHT22 – датчик температуры и влажности, который позволяет измерять температуру от –40 до 80 °C и относительную влажность от 0 до 100 %, с погрешностью ±2 % влажности и менее 0,5 °C температуры. Это позволяет использовать его как уличный датчик температуры;

– BMP180 – датчик измерения атмосферного давления и температуры, который позволяет измерять абсолютное давление в диапазоне от 300 до 1100 гПа и температуру от 0 до +65 °C, с погрешностью –4..+2 гПа для давления и ±1 °C для температуры. С учетом этих параметров датчик может быть использован для определения параметров внутри помещения.

Возьмем самый простой случай, необходимо иметь возможность получать информацию: с комнатного датчика; с уличного датчика; с обоих датчиков одновременно. Тогда достаточно иметь три команды, так как минимальный обмен данными между устройствами происходит байтами, то и под хранение команды можно отвести 1 байт (хоть это и не очень рационально). В силу того что работа осуществляется с монитором порта Arduino, а он

позволяет производить обмен информацией только в символьной форме, команды можно представить значениями:

«1» – получить значения с комнатного датчика;

«2» – получить значения с уличного датчика;

«3» – получить значения с обоих датчиков.

В этом случае команды при передаче будут иметь вид: 0x31, 0x32, 0x33.

Алгоритм программы для МК можно описать с помощью структурной схемы на рисунке 2.

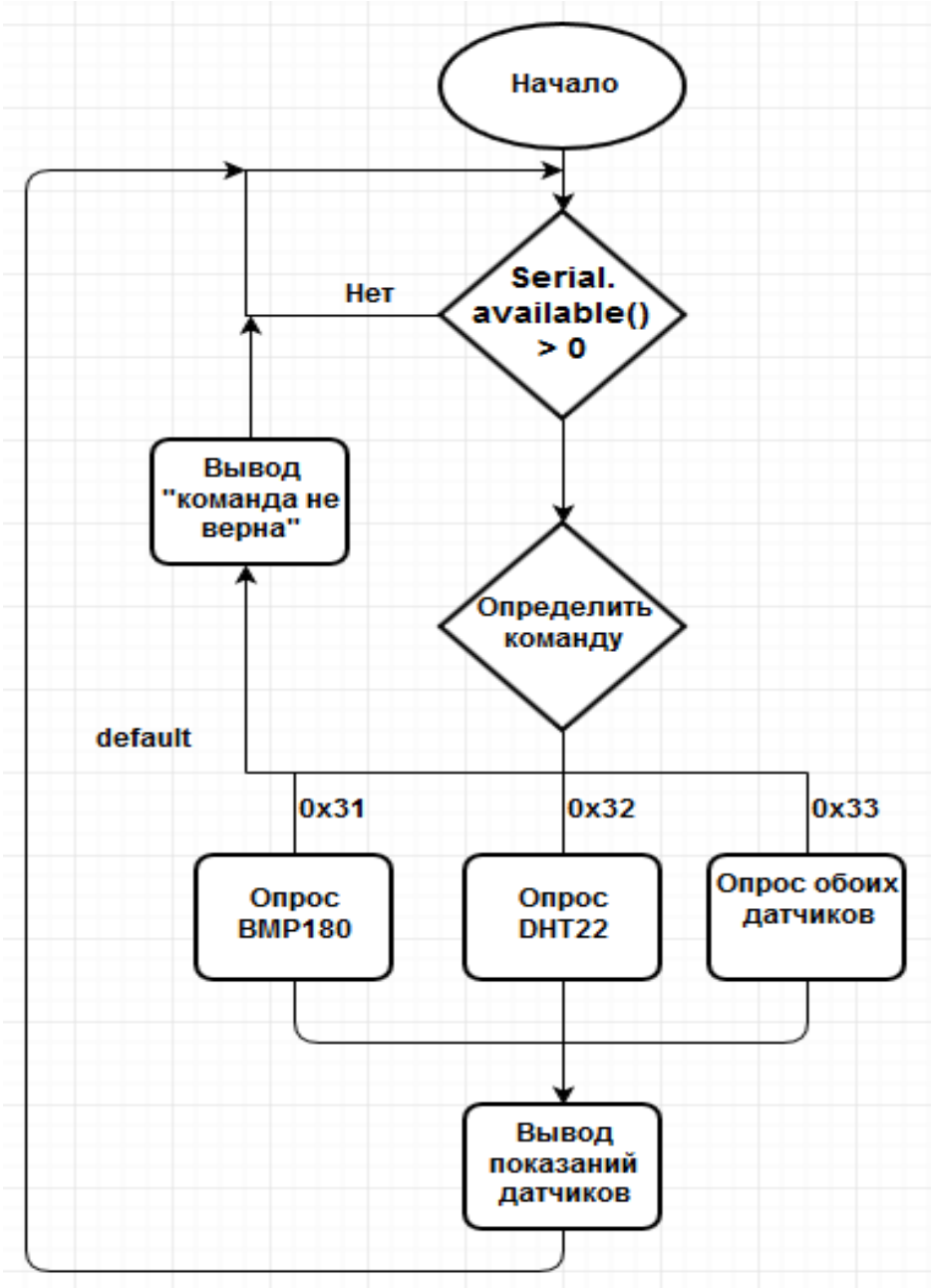



Рисунок 2 – Структурная схема алгоритма программы опроса

Здесь в loop происходит только проверка содержимого буфера com-порта, если в нем есть какие-то данные, то проверяется, какой команде они соответствуют. Если такая команда есть, то вызывается соответствующая функция для считывания значений с нужного датчика и осуществляется вывод его значения, иначе возвращается сообщение, что команда не верна.

**Выполните следующие задания самостоятельно:**

1) напишите программу по предложенной структурной схеме;

2) предложите для вашего устройства еще две команды и модернизируйте программу для их использования.

Ранее показания датчиков передавались в монитор порта Arduino в простой текстовой форме, это позволяло получить наглядную информацию о состоянии устройства. Но если устройство встраивается в сложную систему и выводит информацию о состоянии датчиков не пользователю, а другому контроллеру, то такая форма представления является избыточной.

В этом случае информацию лучше передавать как последовательность байт. Если взять систему команд, используемую в прошлом пункте, и проанализировать, что отправляет в ответ контроллер, то можно сделать следующие выводы:

– все считываемые показания датчиков имеют тип данных float, который занимает в памяти МК 4 байта;

– в случае команд 1 и 2 передавать потребуется 8 байт данных, а для команды 3 потребуется передача 16 байт.

Помимо этого, если к центральному узлу подключено несколько локальных контроллеров, то устройство при ответе должно быть идентифицировано, так как в работе используется одно устройство в качестве идентификатора, то можно добавить к пакету передаваемых данных полученную ранее команду.

Для того чтобы обеспечить корректность передаваемых данных, вместе с полезной нагрузкой обычно отправляют некоторую последовательность байт, интерпретируемую как контрольная сумма. Контрольная сумма – некоторое значение, рассчитанное по набору данных путём применения определённого алгоритма и

используемое для проверки целостности данных при их передаче или хранении. Без контрольной суммы передавать данные опасно, так как помехи присутствуют везде и всегда, весь вопрос только в их вероятности возникновения и вызываемых ими побочных эффектах. В зависимости от условий и выбирается алгоритм выявления ошибок и количество данных в контрольной сумме. Сложнее алгоритм и больше контрольная сумма – меньше нераспознанных ошибок.

Самый простой вариант контрольной суммы – бит четности, который проверяет общую четность двоичного числа, максимально простой алгоритм, использующий всего 1 дополнительный бит при передаче данных. Основной недостаток этого алгоритма – не может обнаружить четное число ошибок и не позволяет восстановить исходную информацию, может быть применен только к 1 байту данных. Протокол UART имеет поддержку контроля четности, которая может быть легко включена как на стороне контроллера, так и на стороне принимающего устройства [19].

Обычно возможностей контроля четности не достаточно, и тогда может быть использована дополнительная контрольная сумма. Рассчитать контрольную сумму можно как сумму ранее переданных байт, просто и логично:

$$CRC = byte(1) + byte(2) + byte(3) + ... + byte(N).$$

Биты переполнения не учитываются, результат укладывается в выделенные под контрольную сумму 8 бит, этот байт добавляется к передаваемому пакету данных. По окончании передачи на принимающей стороне все байты пакета суммируются по тем же правилам и сравниваются со значением контрольной суммы.

При использовании контрольной суммы можно пропустить ошибку, если при случайном сбое один байт увеличится на некоторое значение, а другой байт уменьшится на то же значение, контрольная сумма при этом не изменится.

Для данной работы можно составить следующую структуру передаваемых данных.

Если была принята команда 1 или 2:

| Номер байта | Назначение |
|---|---|
| 0 | Команда |
| 1...4 | Температура |
| 5...8 | Влажность / давление |
| 9 | Контрольная сумма |

Если была принята команда 3:

| Номер байта | Назначение |
|---|---|
| 1...4 | Температура (DHT22) |
| 5...8 | Влажность ((DHT22)) |
| 9...12 | Температура (BMP180) |
| 13...16 | Давление (BMP180) |
| 17 | Контрольная сумма |

Таким образом, в зависимости от команды контроллер будет передавать 10 или 18 байт. Это означает, что необходимо создать байтовый массив для передачи данных подходящей длины. Можно создать динамический массив, но такой подход в МК стараются не применять в силу ограниченности доступных объемов памяти и риска некорректной работы. Поэтому в данном случае лучше создать статический массив длиной, равной максимальному количеству передаваемых байт.

Для повышения надежности системы нужно предусмотреть ответ на неправильную команду, пришедшую по ошибке, договоримся в этом случае отправлять байт, заполненный единицами – 0xFF.

Помимо этого, надо предусмотреть ситуацию, когда на локальном устройстве датчики оказались некорректно подключенными, договоримся использовать в этом случае значения 0xFE для датчика DHT22 и 0xFD для BMP180.

Другая проблема, с которой предстоит столкнуться, это то, что данные с датчиков хранятся в переменных типа float, а передавать их надо как последовательность байт, стоит задача выполнить преобразование float to byte. Для этого можно использовать указатели.

Указатель – переменная, содержащая адрес объекта. Память компьютера или МК можно представить в виде последовательности пронумерованных однобайтовых ячеек, с которыми можно работать по отдельности или блоками. Указатель не несет информации о содержимом объекта, а содержит сведения о том, где размещен объект [20].

Каждая переменная в памяти имеет свой адрес – номер первой ячейки, где она расположена, а также свое значение. Указатель – это тоже переменная, которая размещается в памяти. Она имеет адрес, а ее значение является адресом некоторой другой переменной. Указатели широко используются в программировании на языке Си и часто используются при работе с массивами.

Указатель, как и любая переменная, должен быть объявлен. Общая форма объявления указателя:

тип *ИмяОбъекта;

Для того чтобы представить число в формате float, можно использовать такой подход.

```
    float a,b;
    byte buf[4]; //объявляем байтовый массив на 4 элемента для
размещения в нем переменной типа float, так как она имеет
размер 4 байта
    a=1.1526;
    Serial.println(a);
    *(float*)(buf) = a; //используя указатель, сохраняем
содержимое переменной типа float в область памяти, выделенной
для хранения массива
    Serial.write(buf,4);
    b= *(float*)(buf); //содержимое ячеек памяти массива
считывается как значение типа float и сохраняется в переменную
соответствующего типа
```

**Выполните следующие задания самостоятельно:**

1) напишите программу, реализующую работу данного протокола;

2) * чтобы проверить корректность работы реализованного протокола, возьмите еще одну плату Arduino. Соедините обе платы по UART (для первой платы можно использовать аппаратный UART – пины 0 и 1, для второй – надо использовать другие пины и библиотеку softwareSerial). Напишите программу, отправляющую команды и принимающую сообщения от первого контроллера, и выводящую результаты в монитор порта в понятной человеку форме.

# Литература

1. Аппаратная часть платформы Arduino // Аппаратная платформа Arduino | Arduino.ru. – Режим доступа: http://arduino.ru/Hardware (дата обращения: 02.07.2019).

2. Arduino – распиновка и схема подключения // ArduinoPlus.ru – сайт о Arduino, Raspberry, Wemos и других микроконтроллерах, программировании, разработке устройств. – Режим доступа: https://arduinoplus.ru/arduino-raspinovka/ (дата обращения: 02.07.2019).

3. Петин, В. А. Проекты с использованием контроллера Arduino / В. А. Петин. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2015. – 448 с.

4. Дребезг контактов // Дребезг контактов. Что такое дребезг контактов? – Режим доступа: https://dinistor.info/glossarij/drebezg-kontaktov (дата обращения: 02.07.2019).

5. Arduino UNO // Аппаратная платформа Arduino Arduino.ru. – Access mode: http://arduino.ru/Hardware/ArduinoBoardUno (дата обращения: 02.07.2019).

6. BMP180 // BMP180. – Access mode: https://www.bosch-sensortec.com/bst/products/all_products/bmp180 (дата обращения: 02.07.2019).

7. Adafruit_BMP085_Unified // Github – adafruit/ Adafruit_BMP085_Unified: Unified sensor driver for Adafruit's BMP085 & BMP180 breakouts. – Access mode: https://github.com/ adafruit/Adafruit_Sensor (дата обращения: 02.07.2019).

8. Adafruit_Sensor // Github – adafruit/ Adafruit_Sensor: Common sensor library. – Access mode: https://github.com/adafruit/ Adafruit_BMP085_Unified (дата обращения: 02.07.2019).

9. Библиотека Wire для Arduino для работы с шиной I2C. Копаем глубже... // Библиотека Wire для Arduino для работы с шиной I2C. Копаем глубже... – Режим доступа: https://radioprog.ru/ post/233 (дата обращения: 02.07.2019).

10. Temperature and humidity moduleAM2302Product Manual // AOSONG Temp, Humidity & Dew point measurement experts. – Ac-

cess mode: https://static.chipdip.ru/lib/426/DOC001426886.pdf (дата обращения: 02.07.2019).

11. DHT-sensor-library // Github – adafruit/ DHT-sensor-library: Arduino library for DHT11, DHT22, etc Temperature & Humidity Sensors. – Access mode: https://github.com/adafruit/DHT-sensor-library (дата обращения: 02.07.2019).

12. Лафоре, Роберт. Объектно-ориентированное программирование в C++ / Роберт Лафоре. – 4-е изд. – Питер, 2004. – 928 с.

13. Урок 9. Создание библиотеки для Ардуино // Уроки Ардуино. Создание библиотеки для Ардуино. | Оборудование, технологии, разработки. – Режим доступа: http://mypractic.ru/urok-9-sozdanie-biblioteki-dlya-arduino.html (дата обращения: 02.07.2019).

14. Урок 10. Прерывание по таймеру в Ардуино. Библиотека MsTimer2. Параллельные процессы // Уроки программирования Ардуино. Прерывание по таймеру в Ардуино. Библиотека MsTimer2. Параллельные процессы. | Оборудование, технологии, разработки. – Режим доступа: http://mypractic.ru/urok-10-preryvanie-po-tajmeru-v-arduino-biblioteka-mstimer2-parallelnye-processy.html (дата обращения: 02.07.2019).

15. attachInterrupt(interrupt, function, mode) // attachInterrupt(interrupt, function, mode) | Аппаратная платформа Arduino. – Access mode: http://arduino.ru/Reference/AttachInterrupt (дата обращения: 02.07.2019).

16. Урок 48. Обмен данными между платой Ардуино и компьютером через интерфейс UART. // Уроки Ардуино. Обмен данными между платой Ардуино и компьютером через интерфейс UART. | Оборудование, технологии, разработки. – Режим доступа: http://mypractic.ru/urok-48-obmen-dannymi-mezhdu-platoj-arduino-i-kompyuterom-cherez-interfejs-uart.html (дата обращения: 02.07.2019).

17. Универсальный асинхронный приёмопередатчик // Универсальный асинхронный приёмопередатчик – Википедия. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%A3%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B0%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE

%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%B8%D1%91%D0%BC%D0%BE%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%82%D1%87%D0%B8%D0%BA (дата обращения: 02.07.2019).

18. Serial // Arduino – Serial. – Access mode: https://www.arduino.cc/en/pmwiki.php?n=Reference/Serial (дата обращения: 02.07.2019).

19. Контрольная сумма // Контрольная сумма – Википедия. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D1%83%D0%BC%D0%BC%D0%B0 (дата обращения: 02.07.2019).

20. Указатели в C++ // Указатели: программирование на C++. – Режим доступа: http://cppstudio.com/post/423/ (дата обращения: 02.07.2019).