Елисеев Д.

Цифровая электроника для начинающих

Arduino
Raspberry Pi
ESP32
BBC Microbit

Версия текста 1.4, (c) 2018

Введение

Современный мир наполнен интеллектуальной электроникой. Еще 50 лет назад единственными устройствами в доме были утюг и телевизор, сейчас, чтобы перечислить всё, не хватит пальцев на обеих руках. Маршрутизатор WiFi “раздает” интернет, “умные” розетки подключены к беспроводной сети и передают текущее потребление электроэнергии, датчик за окном передает температуру на висящую на стене метеостанцию, которая не только показывает погоду, но и синхронизирует и показывает точное время по радиосигналу из Германии. Напольные весы не только позволяют узнать вес, но и отправляют его на смартфон, наручные часы не только показывают время, но и считают число пройденных шагов.

Что объединяет все эти устройства? То, что внутри каждого из них есть микроконтроллер - небольшой компьютер, способный выполнять программу, без которой устройство не будет работать. И чтобы понимать современную технику, надо понимать не только электронику, но и программирование. Тем более, что это занимательно и вполне интересно.

В данной книге описаны основы электроники, от простого к сложному - от светодиода до микропроцессора, от мультиметра до осциллографа. К сожалению, не везде есть радиокружки и не у каждого есть “знающие” знакомые. Эта книга позволит хоть сколько-то восполнить данный пробел. Как подключить светодиод, как воспользоваться мультиметром, книга даст ответы и на такие вопросы. Основной акцент книги - на цифровой электронике, поэтому “аналоговые” понятия рассказаны лишь поверхностно. Желающие узнать подробнее, например, о режимах работы транзистора, могут найти это в специальной литературе.

Книга рассчитана на начинающих свой путь в освоении

электроники, и на школьников от 10-12 лет. Поэтому теория дается в минимальном объеме, а основной акцент делается на практических занятиях.

Если кто-либо благодаря этой книге заинтересуется электроникой или программированием, значит она написана не зря. Книга распространяется бесплатно в электронном виде, последнюю версию можно скачать на странице https://cloud.mail.ru/public/F6Vf/nY6iSxXcd. Дополнения, пожелания или примеры программ можно присылать по адресу dmitryelj@gmail.com, наиболее интересные из них могут быть включены в книгу.

Приятного чтения.

Часть 1. Основы электроники

Биты и байты в компьютере - для пользователя зачастую лишь данные, которые можно посмотреть на экране. Однако на физическом уровне, это все равно остается в виде изменения токов и напряжений, тех самых с которыми экспериментировали еще Гальвани и Тесла более 100 лет назад. Поэтому при создании цифровых схем, нужно хотя бы на базовом уровне понимать, как это работает. Это не требуется для пользователя обычного персонального компьютера, но если мы делаем какое-то устройство, способное “общаться” с внешним миром, без этого никак не обойтись. К примеру, можно подключить мотор напрямую к выводу микроконтроллера - и испортить контроллер, т.к. он в состоянии выдавать лишь небольшой ток. Можно подключить датчик, рассчитанный на 3.3В, к 5-вольтовому выводу Arduino, результат будет аналогичный. Понимание таких моментов позволит делать эффективные и надежные схемы.

Поэтому мы начнем с основ электричества - с тока и напряжения, с измерений в электрической цепи.

1.1 Детали и инструменты

Электроника - это в первую очередь практическая наука. Чтобы проведенное с книгой время было полезным и интересным, желательно заранее приобрести соответствующие компоненты. Если в наличии нет ничего, то стоит приобрести сразу готовый набор, это выйдет немного дешевле.

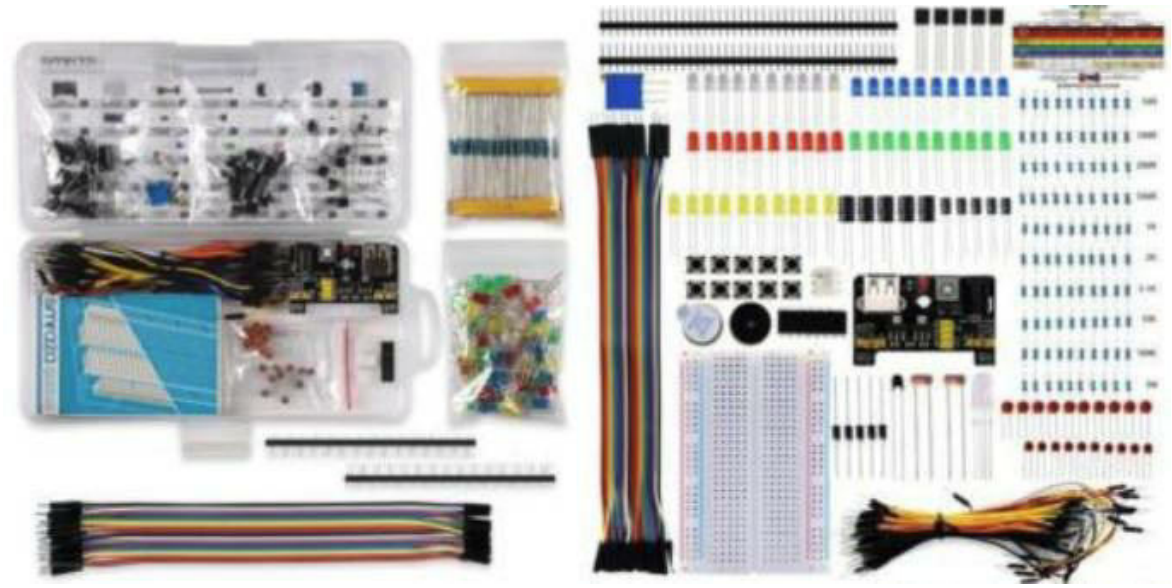
Следующий вопрос - что и где купить. Весьма рекомендуется научиться приобретать товары на www.ebay.com - все комплектующие сейчас производятся в Китае, и заказывать их напрямую у китайских продавцов в 3-5 раз выгоднее, чем взять тот же товар в российском магазине. Для покупки на eBay достаточно зарегистрировать аккаунт Paypal, он позволяет оплачивать товары банковской картой, не передавая данные платежной карты непосредственно продавцу. Имея аккаунт Paypal, можно делать покупки и на ebay. Адрес и ФИО получателя стоит указать латинскими буквами, т.к. разные почтовые программы могут иметь разную кодировку, и не факт что в китайском магазине корректно отобразится русский шрифт. Ebay также имеет программу защиты покупателей - если товар имеет дефект, то продавец обязан вернуть деньги или выслать замену.

Для сравнения порядка цен, плата Arduino Nano 3.0 стоит в популярном магазине “Чип и дип” 440 рублей, точно такая же плата на eBay стоит 3.5$ с бесплатной почтовой доставкой, т.е. порядка 200р. 3-4 таких покупки в сумме дадут вполне приличную экономию, но заплатить за это придется 3-4 недельным ожиданием посылки.

Названия наборов для сборки приводятся на английском, чтобы их было легче найти в поиске.

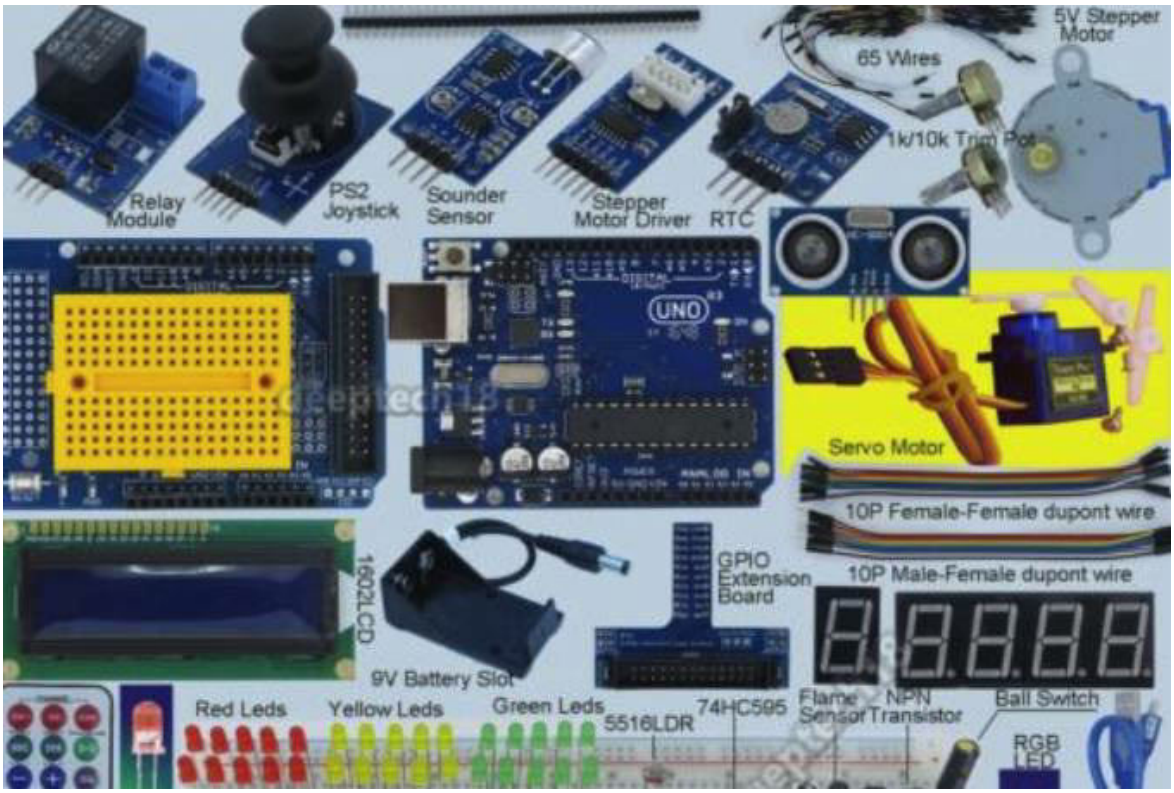
Electronics Starter Kit with Breadboard

Неплохой набор из макетной платы, проводов, светодиодов, “пищалки”, конденсаторов и пр. Цена вопроса порядка 20$.



UNO R3 Starter Kit for Arduino

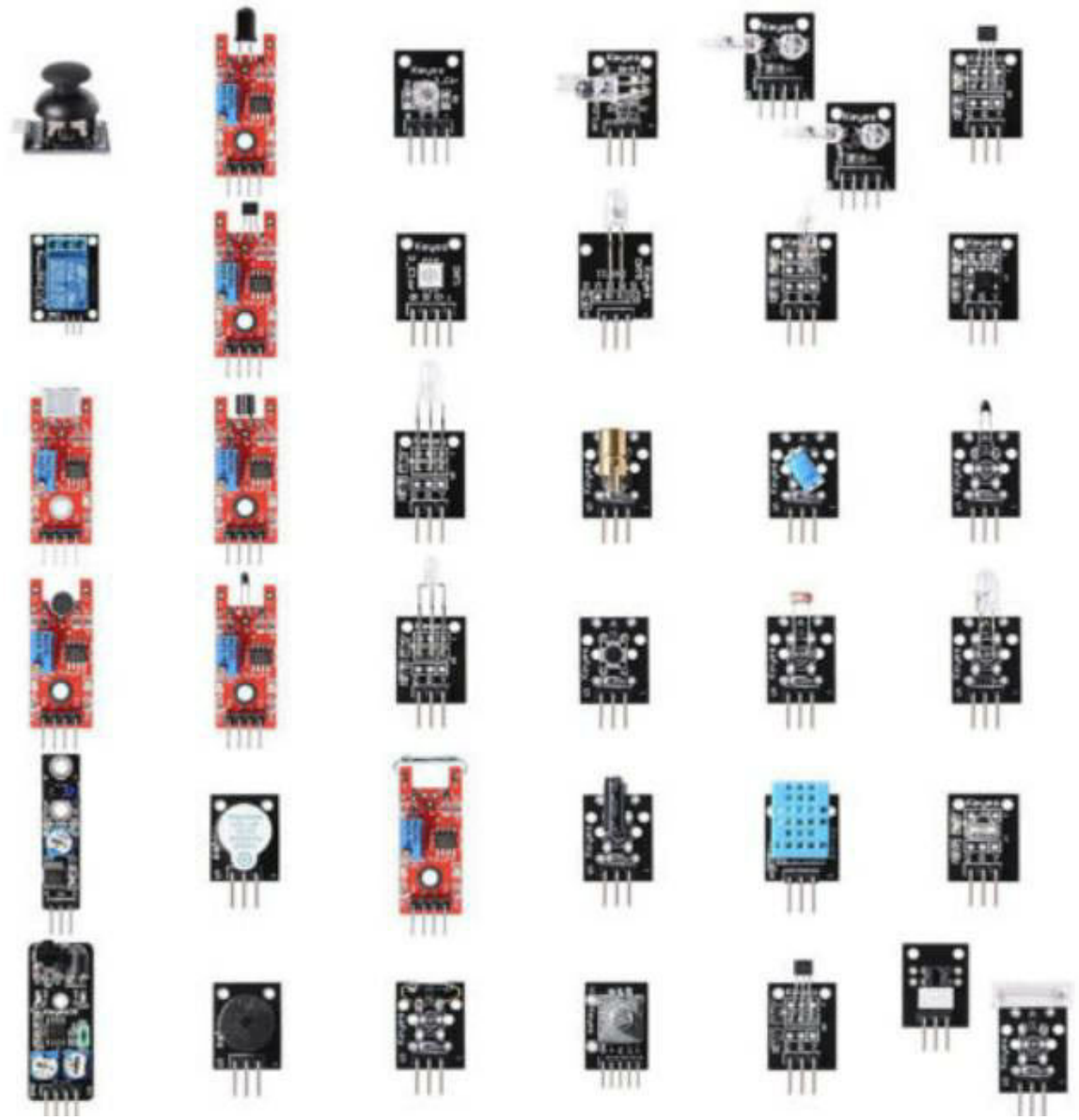
Набор из платы Arduino и различных сенсоров. Пригодится ко второй части книги.



Примеры в книге даны как раз для платы Arduino Uno, так что ее имеет смысл взять для повторения описанных опытов. Цена вопроса порядка 35$.

37 in 1 Sensor Module Kit

Если у кого-то уже есть Arduino, отдельно можно докупить блок сенсоров на все случаи жизни - от цветного светодиода до датчика Холла или лазера.

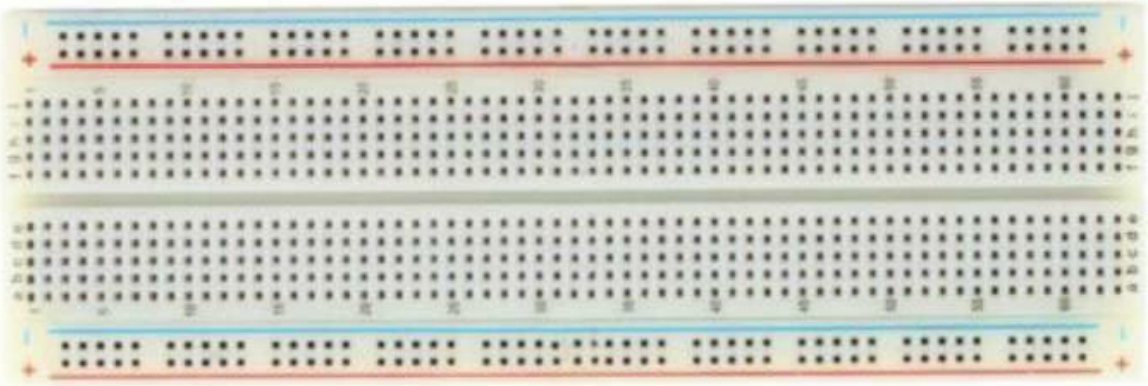


Возможности всех этих сенсоров выходят за рамки данной книги, но некоторые из них точно пригодятся, остальные можно будет использовать в дальнейшем.

Разумеется, все эти компоненты можно приобрести и отдельно, по цене 2-3$ за штуку, но учитывая сроки ожидания посылок на почте,

может быть целесообразнее взять сразу побольше.

Кстати, целесообразно купить и макетную плату (breadboard). Она позволяет соединять элементы без пайки, просто вставляя их плату. К такой плате сразу можно приобрести и набор соединительных проводов.



ESP32 Development Board

Эта плата имеет встроенный WiFi и может “общаться” с внешним миром, используя интернет. Она будет рассматриваться в третьей части книги.

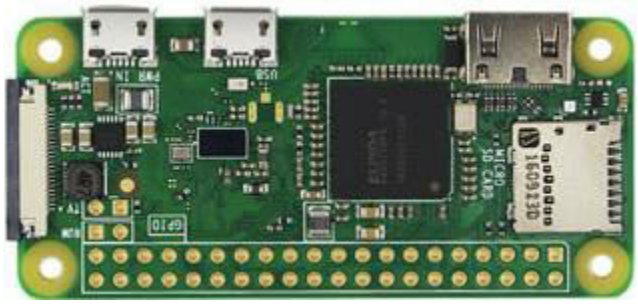
Raspberry Pi 3

Это полноценный компьютер с Linux, на котором можно запустить практически все что угодно, даже собственный веб-сервер. Он будет рассматриваться в 4й части книги.

Имеет смысл купить Raspberry Pi и блок питания, корпус при желании также можно докупить отдельно. А вот карту памяти лучше взять в местном компьютерном магазине, по личному опыту, все китайские карты памяти оказались некачественными.

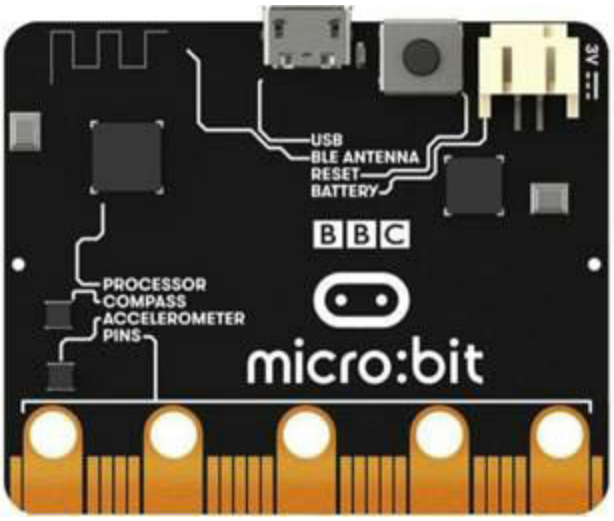
Raspberry Pi Zero W

Это более компактная версия Raspberry Pi, имеющая примерно те же возможности.



BBC Micro:bit

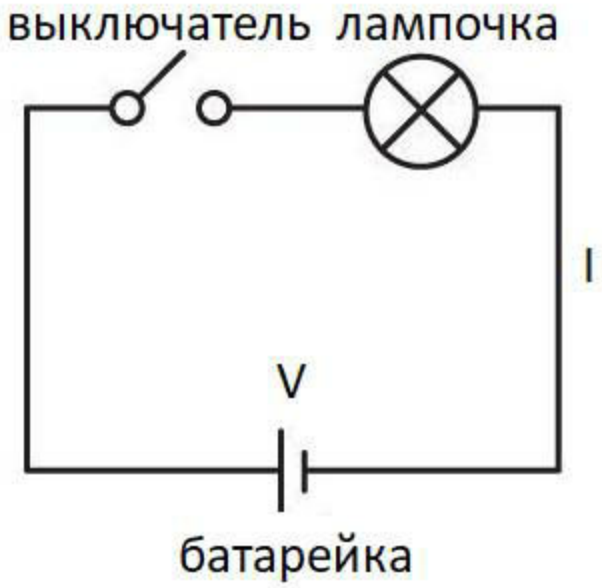
Это микрокомпьютер для самых маленьких, и рассмотрен он будет в 5й части. Плата имеет светодиодную панель, микроконтроллер, гироскоп и акселерометр, а программируется она через простой визуальный интерфейс.



На этом мы закончим с покупками, и перейдем собственно, к электронике. Начнем с азов, и с самого начала. Кто это уже знает, может сразу перейти ко второй части книги.

1.2 Электрическая цепь, ток и напряжение

На столе есть 3 детали - лампочка, батарейка и выключатель. Что нужно сделать, чтобы схема заработала? Нужно соединить устройства проводами в так называемую **электрическую цепь**. Это будет выглядеть примерно так:



Что важно в этой схеме? Во-первых, все элементы должны быть соединены между собой. Электрический ток - это поток электронов,

“бегущий” по проводам. Можно упрощенно представить как такой поток “вытекает” из одного полюса батарейки, доходит до лампочки, проходит через вольфрамовую нить, нагревая ее, затем “втекает” обратно в другой полюс батарейки. Если хоть где-то цепь будет разомкнута, ток не пойдет, и лампочка гореть не будет.

Важно: показанная схема работает только с лампочкой накаливания. Светодиод нельзя подключать напрямую к батарейке, о его подключении будет рассказано в отдельной главе.

Что нам еще нужно знать о схеме?

Батарейка (источник питания). Каждый источник питания имеет свое **напряжение** (оно обозначается буквой **V** и измеряется в **вольтах**). Самым популярным является напряжение в 1.5В, такие батарейки используются в большом числе устройств.

Чем больше батарея, тем больше энергии она может хранить, и ее хватит на большее время работы. Поэтому без специальной нужды не стоит запитывать схему от самых маленьких батареек, они израсходуются слишком быстро.

Разные схемы имеют разное напряжение питания, и схема, рассчитанная на 9 или 12В, не заработает от напряжения 5В. Еще хуже превышение напряжения - если детали в схеме рассчитаны на 5В, а на нее подать 12В, компоненты могут просто испортиться.

Второй важный момент - **полярность питания**: нельзя путать “+” и “-” батареи, т.к. от этого зависит направление потока электронов в

схеме. Для лампочки это безразлично, а многие другие детали могут испортиться. На каждой схеме, и на каждом электронном устройстве должно быть указано где подсоединять “+” и где “-”, это нужно строго соблюдать.

Что касается цифровых схем, то в большинстве случаев описанных в книге, они питаются от USB, напряжением 5В. Уже готовую схему можно подключить к отдельному блоку питания, например от мобильного телефона.

Лампа накаливания в данной схеме - это потребитель энергии. Лампа тоже имеет свое номинальное напряжение: если подать слишком малое напряжение, лампа будет гореть слабо, если подать слишком большое напряжение, лампочка будет гореть чересчур ярко, но очень быстро перегорит. Лампа горит потому, что через нее протекает электрический ток, силу этого тока обозначают буквой **I** и она измеряется в **амперах**. Каждая батарейка может отдать только некий максимальный ток, именно поэтому к одной батарейке нельзя подключить 10 лампочек - тока на всех не хватит, и они будут гореть слишком слабо.

Кстати. *Ток и напряжение - это основные параметры электрической цепи. Произведение тока на напряжение называется* ***мощностью****, этот параметр измеряется в* ***ваттах****. Если на домашнем электрочайнике указана мощность 2200Вт, это значит что чайник потребляет ток 10А при напряжении 220В.*

Слишком большой ток может даже испортить электрическую цепь, именно поэтому в каждой квартире стоят предохранители - они отключают цепь, если ток слишком большой. То же самое и с батарейкой - если случайно замкнуть ее выводы накоротко, ток будет слишком большим, и батарейка может испортиться. Поэтому при создании любой схемы важно следить, чтобы провода от батареи случайно не замкнулись между собой. Это называется **короткое замыкание**, и является аварийным режимом для схемы - она может испортиться, или даже загореться.

Ни ток, ни напряжение нельзя увидеть глазами. Для того чтобы

увидеть и ток и напряжение, нам понадобится **мультиметр**. Его мы рассмотрим в следующей главе.

1.3 Измерения

Одним из самых важных для начинающего радиолюбителя приборов, является мультиметр. Выглядит он примерно так:

Мультиметр имеет несколько режимов работы, которые нам пригодятся. Режим изменяется поворотом переключателя настроек в одно из положений. Рассмотрим разные режимы подробнее.

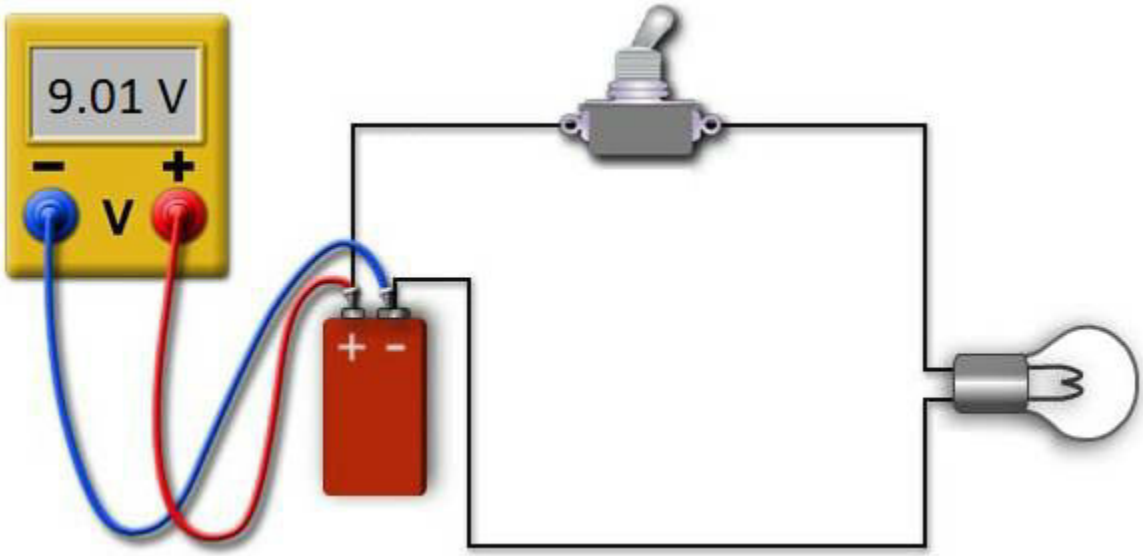
Измерение постоянного напряжения

Значок на корпусе мультиметра ”V” (Вольты):

Именно постоянное напряжение мы и получаем от батарейки, этот режим нам пригодится чаще всего. Как можно видеть на мультиметре,

диапазон измеряемых напряжений довольно-таки велик: 200мВ (0.2В), 2000мВ (2000мВ=2В, приставка “милли” обозначает одну тысячную), 20В, 200В и 500В.

Сначала нужно выбрать тот диапазон, в котором находится измеряемое напряжение. Например, для батарейки в 1.5В можно выбрать 2000мВ или 20В. Для измерения, щупы мультиметра надо подключить **прямо к батарее**:



(фото с сайта https://www.dlsweb.rmit.edu.au)

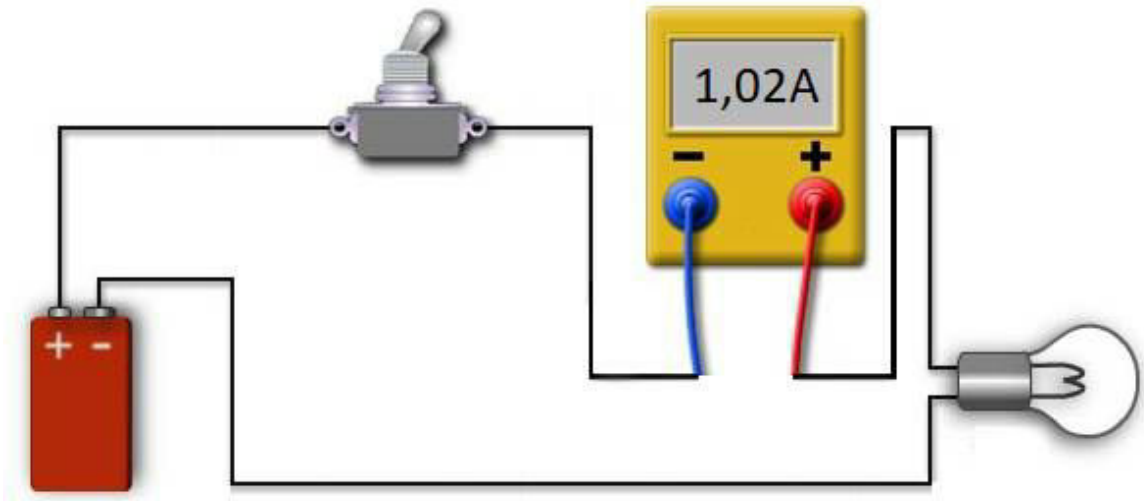
Измерение переменного напряжения

Значок на корпусе мультиметра:

Такое напряжение присутствует в электросети, в наших опытах оно не пригодится. На корпусе мультиметра мы видим максимальные значения 200 и 500В.

Измерение тока

На мультиметре ему соответствует значок “А” (Амперы). В наших опытах ток измерять скорее всего, не придется. Но важно знать, что в отличие от напряжения, ток измеряется при включении мультиметра **в разрыв цепи**:



Это важно потому, что в режиме “А” сопротивление мультиметра очень мало. И если перепутать, и включить мультиметр в режиме “А” для измерения напряжения, получится короткое замыкание - и батарея и мультиметр могут выйти из строя.

Измерение сопротивления

Значок на корпусе мультиметра:

Для измерения достаточно подключить резистор к выводам мультиметра:

О сопротивлении и резисторах мы поговорим позже.

“Прозвонка” цепи

Значок на корпусе мультиметра:

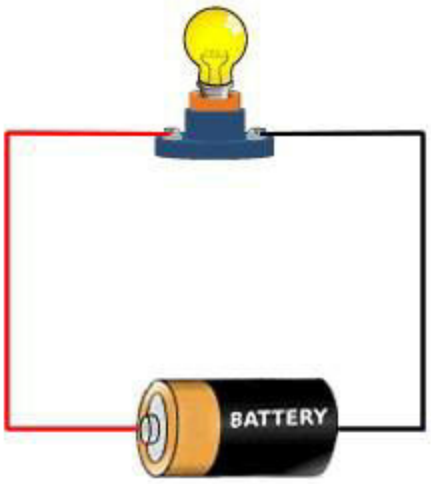
Это важный для радиолюбителя режим - им можно проверить, есть ли контакт между двумя точками схемы. Если коснуться щупами двух концов провода, то мультиметр запищит: это значит, что провод исправен. Если внутри контакта нет, например провод оборван, то звука не будет. Как говорилось раньше, если электрическая цепь где-то разорвана, ток через нее не пойдет, и схема работать не будет. Этот режим позволяет проверить правильность соединений.

В конце измерений **мультиметр надо выключить** - он тоже работает от батарейки, и если оставить прибор включенным, она разрядится. Старые механические вольтметры и амперметры работали без батареек, современные приборы, увы, так не могут.

Важно: батарейки и аккумуляторы содержат едкие химические вещества. Поэтому их лучше не выкидывать вместе с обычным мусором - эти вещества попадают в воду и почву, отравляют растения и могут вредить людям и животным. Использованные батарейки стоит сдать в специальные пункты приема, обычно они расположены в крупных магазинах электроники.

1.3 Обозначения на схемах

Электрическую схему можно изобразить вот так:



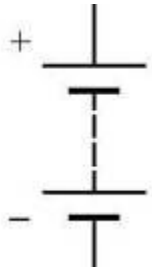
Это конечно, красиво, но рисовать такую схему долго и неудобно, да и при большом количестве элементов читать схему будет сложно.

Поэтому инженеры придумали стандартные обозначения элементов, чтобы при взгляде на схему сразу было понятно, какие элементы в ней используются.

Рассмотрим обозначения, которые пригодятся нам в дальнейшем.

Батарея

Это обозначение мы уже видели в предыдущей главе:



Рядом с батареей обычно указывается напряжение, например 9В.

Лампа накаливания

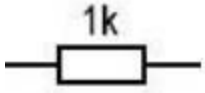
Лампа содержит внутри вольфрамовую нить, которая светится при прохождении тока.

Светодиод

В отличие от лампы накаливания, светодиод содержит внутри кристалл, светящийся определенным цветом. Второе важное отличие от лампы - при подключении светодиода важно соблюдать полярность. Диод может пропускать ток только в одном направлении, если включить его наоборот, он гореть не будет.

Подробнее про светодиод будет рассказано в следующей главе.

Резистор

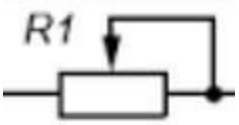


Резистор - это элемент, обладающий определенным *электрическим сопротивлением*, оно измеряется в Омах или Килоомах. Рядом с резистором на схеме должно быть написано его значение.

На самом резисторе значение кодируется цветом полосок:

Их цвета можно посмотреть в интернете, но часто бывает проще измерить сопротивление мультиметром.

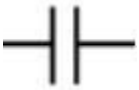
Резистор также может быть переменным: он может менять свое сопротивление, обычно вращением небольшой рукоятки.



Такие резисторы используются, например, в радиоприемнике для регулировки громкости.

В англоязычных статьях или компьютерных программах может использоваться другое обозначение резистора:

Конденсатор

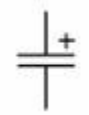


Конденсатор - это элемент, способный накапливать электрическую энергию. Его емкость измеряется в нано, микро или пикофарадах, и обозначается нФ, пФ или мкФ (на схемах может применяться также обозначение u, например 10u = 10мкФ).

Конденсаторы небольшой емкости обозначаются специальными кодами, например “150” или “152”. Первые две или три цифры обозначают значение емкости в пикофарадах (пФ), а последняя цифра — количество нулей. Легко подсчитать, что “151” обозначает емкость “15 0” = 150 пикофарад, а “152” - “15 00”, т.е. 1500 пикофарад. Эти коды можно найти в Интернет, набрав в поиске “маркировка конденсаторов”. Бывают также конденсаторы переменной емкости, но в наших опытах они не пригодятся. Обычно они используются в радиоприемниках для

перестройки частоты.

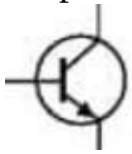
Наибольшую емкость имеют **электролитические конденсаторы**. В отличие от обычных, для их подключения важна полярность, и на схеме такие конденсаторы отмечают значком “+”.



На самом конденсаторе белой полоской отмечен “-”, как показано на рисунке.

Второй важный параметр для электролитического конденсатора - **максимальное напряжение**, оно указано на корпусе конденсатора. Если его превысить, конденсатор может выйти из строя или даже взорваться, но при нормальном использовании он совершенно безопасен. Поэтому, если на схеме написано 100мкФх15В, то можно использовать конденсатор с большим напряжением (например 100мкФх50В), а вот с меньшим (например 100мкФх6В) уже нельзя.

Транзистор

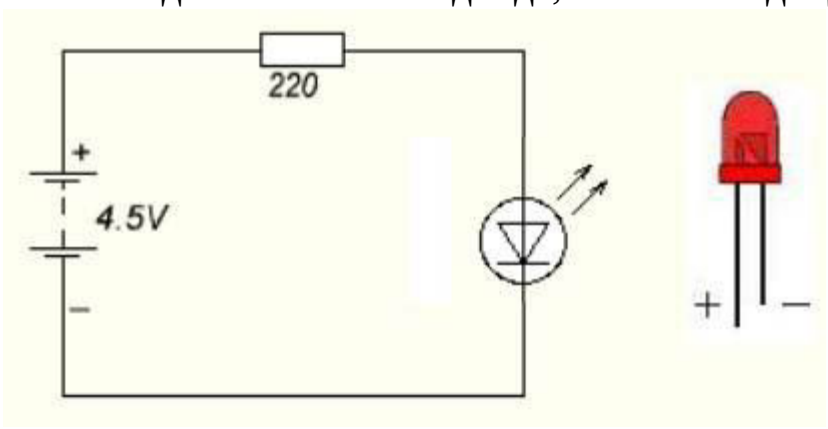
Транзистор - ключевой элемент современной электроники, причем как в переносном, так и в прямом смысле. С помощью транзистора можно усилить слабый сигнал, или управлять включением и выключением лампы большой мощности. К примеру, нельзя подключить напрямую мощный светодиод к выводу микроконтроллера, он не загорится. На помощь придет транзистор, способный управлять мощным током с помощью слабого сигнала. 

Транзисторы бывают биполярные и полевые, о разных видах мы поговорим отдельно. А пока закончим с теорией, и приступим к практике.

1.4 Подключаем светодиод

Одним из наиболее популярных элементов для начинающих радиолюбителей является светодиод. Светодиодным освещением можно украсить колесо велосипеда, новогоднюю елку, корпус компьютера, есть даже женские украшения на основе светодиодов:

Схема подключения светодиода, и сам светодиод, выглядят так:



При подключении важно запомнить, что:

- Для светодиода важна полярность. Если подключить его

наоборот, светодиод гореть не будет. Обозначение выводов показано справа от схемы.

- Обязательно нужен резистор для ограничения тока. Без него светодиод сразу же перегорит.

- Для светодиода важнее сила тока, чем напряжение. Можно использовать практически любой источник напряжения (6В, 9В, 12В), главное правильно подобрать номинал резистора.

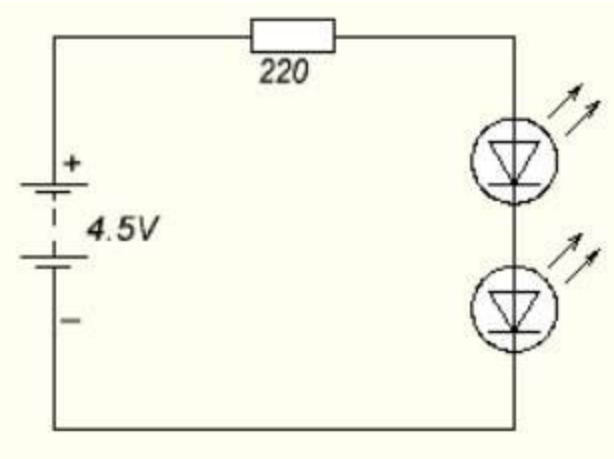
Некоторые значения резистора для разных напряжений показаны в таблице:

Напряжение	Сопротивление
3.0В	56 Ом
4.5В	150 Ом
9В	390 Ом
12В	560 Ом
24В	1.2 КОм

Эти значения являются примерными. Например вместо сопротивления в 150 Ом подойдет и 220 Ом, светодиод лишь будет гореть чуть слабее, что возможно будет даже незаметно на глаз. Если установить резистор слишком малого сопротивления, светодиод будет гореть ярче, но быстрее перегорит. При слишком большом номинале резистора, он будет гореть очень слабо, но и потребление тока при этом будет меньше. Это можно использовать, например, для ночника.

Для более точного расчета можно воспользоваться онлайн калькулятором, например по адресу http://cxem.net/calc/ledcalc.php.

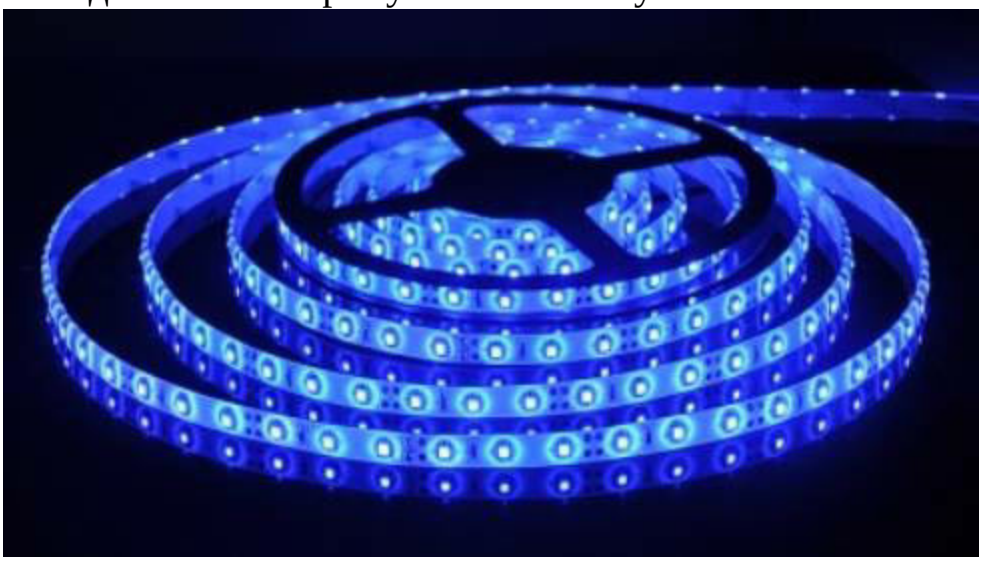
Можно подключить несколько светодиодов подряд, соединяя последовательно “плюс” с “минусом”, такое подключение называется **последовательным**:



Светодиоды будут гореть слабее, чтобы повысить яркость, можно взять резистор меньшего сопротивления.

Последовательно можно соединять не только светодиоды, но и батареи - их напряжение при этом **суммируется**. Это очень полезно, если нужно из нескольких батарей собрать источник питания с более высоким напряжением.

Кстати, в продаже есть и готовые светодиодные ленты разных цветов, в них уже встроены и светодиоды и резисторы. Такие ленты можно подключать напрямую к источнику в 12В.



Если посмотреть на такую ленту вблизи, то мы увидим знакомые

элементы - светодиоды и резисторы.

Такой лентой можно не только украсить что-либо, но и даже использовать их для освещения комнаты.

Задание: опыты со светодиодом и батарейками

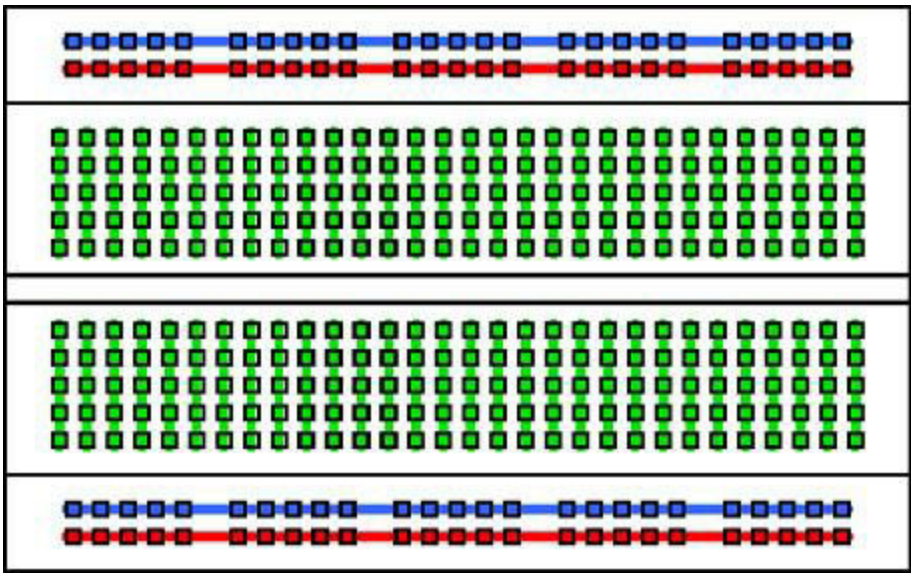
Опыт 1. Собрать источник питания.

Взять 3 батарейки АА, соединить их последовательно в одну большую батарею.

Следует убедиться, что общее напряжение соответствует сумме напряжений отдельных батарей.

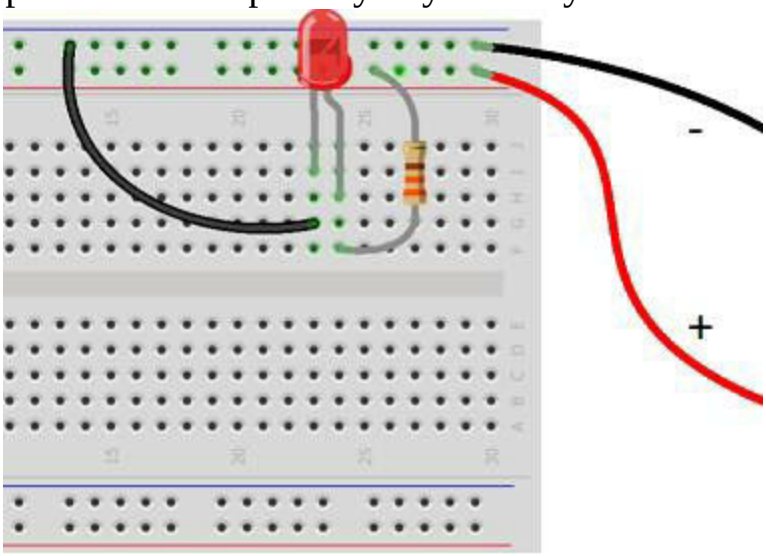
Опыт 2. Собрать схему из светодиода и резистора на 220 Ом.

Для сборки нам понадобится **макетная плата**. Это плата с отверстиями, которые внутри соединены следующим образом:



Верхние и нижние линии используются для подключения питания, внутренние нужны для подключения элементов.

С помощью макетной платы и соединительных проводов очень быстро и легко собрать нужную схему:



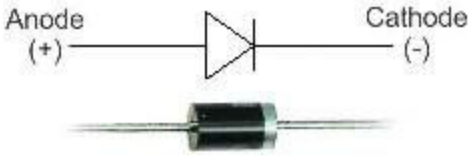
Нетрудно убедиться, что все соединения образуют электрическую цепь, и при подключении батареи светодиод будет гореть. Плюс такой платы в том, что переключение или замена элемента занимает всего лишь несколько секунд.

Самостоятельная работа: испытать последовательное соединение нескольких светодиодов. Также интересно проверить разные резисторы, номиналом от 100 до 1000 Ом, и посмотреть, насколько сильно будет

отличаться яркость светодиодов во всех случаях.

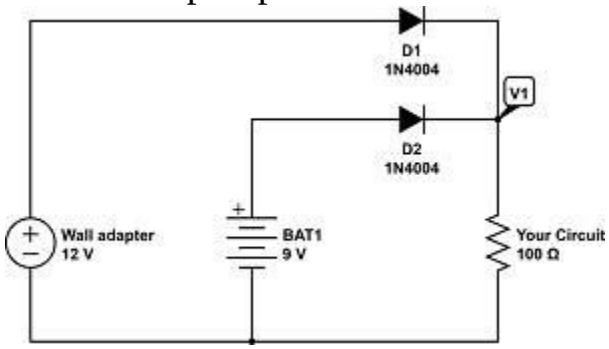
1.5 Опыты с диодом

Кроме светодиодов, существуют и обычные диоды. Нам впрочем, они не так уж часто будут пригождаться, но знать об их свойствах стоит. Диод - это полупроводниковый элемент, способный проводить ток только в одном направлении.



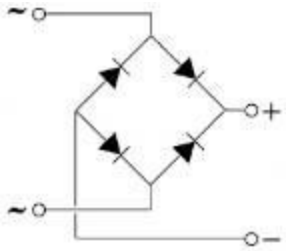
Направление тока легко запомнить, представив изображение диода как воронку для воды - очевидно что лить воду нужно в “широкую” часть. Можно взять самую первую схему с лампочкой и батарейкой, и включить в цепь диод. Легко убедиться, что если перевернуть диод, лампочка гореть не будет. Это может пригодиться, например если нужно защитить схему от неправильной полярности подключения.

Вторая полезная схема, которая может пригодиться - использование резервного питания:



В такой схеме схема работает от внешнего блока питания на 12В, также имеется резервная 9-вольтовая батарея. Ток может течь через диод только “от большего к меньшему”. Поэтому когда напряжение блока питания присутствует (а 12В > 9В), диод D2 “закрывает” батарею, если напряжения нет, схема будет работать от батареи.

Диоды также используются в любом блоке питания - они преобразуют переменный ток в постоянный. Такая схема подключения называется “диодный мост”.



По схеме несложно увидеть, что при любой входной полярности на верхнем выходе всегда будет “+”, а на нижнем “-”.

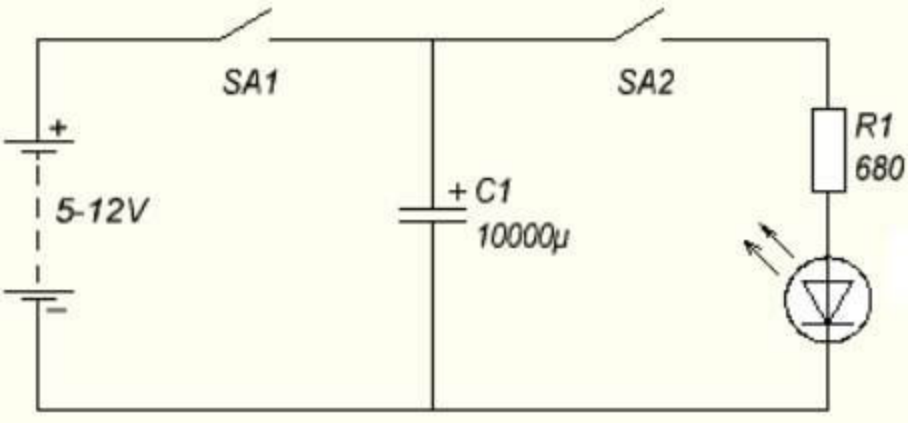
Самостоятельная работа: собрать схему из диодного моста, светодиода и резистора. Убедиться, что подключенный через мост, светодиод горит при любой полярности подключения батареи.

1.6 Опыты с конденсатором

Конденсатор - это элемент, способный накапливать электрический ток. Первый конденсатор был изобретен еще в 1745 году в голландском городе Лейдене, тогда он часто назывался “лейденской банкой” (leiden jar). Фактически, это и была банка, оклеенная изнутри и снаружи фольгой. Обкладки конденсатора способны накапливать электрический заряд, т.к. через изолятор электроны пройти не могут.

Современные конденсаторы, в принципе, работают по такому же принципу, только слои, разделенные изолятором, скручены и помещены в цилиндр. Емкость современного конденсатора гораздо больше, чем у старинных “банок”.

Соберем простую схему:



Испытать ее просто. Нажимаем кнопку “1” и держим ее некоторое время - конденсатор заряжается до напряжения, равного напряжению батареи. Затем отпускаем кнопку, конденсатор больше не соединен с батареей. Нажимаем кнопку “2” - и видим, что светодиод горит, хотя по сути, схема от батареи уже отключена. Светодиод горит за счет заряда, накопленного в конденсаторе. Разумеется, довольно-таки быстро он погаснет. Чем больше емкость конденсатора, тем дольше будет гореть светодиод. От электролитического конденсатора емкостью 10000мкф светодиод может гореть несколько секунд. Если же взять, конденсатор еще большей емкости, например так называемый **ионистор**, то светодиод может гореть до получаса. Бывают батареи ионисторов столь большой емкости, что от них может несколько километров ехать троллейбус.

Кстати, конденсаторы можно соединять **параллельно**, при этом их емкость суммируется. Это может пригодиться, если в наличии нет нужного конденсатора: его можно собрать из нескольких меньшей емкости.

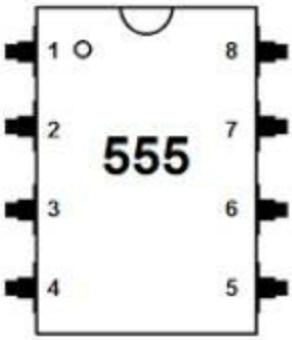
Самостоятельная работа: испытать в схеме конденсаторы разной емкости, проверить как влияет емкость на время свечения светодиода. Также можно испытать параллельное соединение конденсаторов.

1.7 Микросхема NE555

Следующей, и весьма полезной для радиолюбителя микросхемой, является таймер NE555. Она была создана еще в 1971, но до сих пор актуальна - с помощью NE555 можно создавать различные генераторы

сигналов. Это может пригодиться в разных схемах, от мигания светодиодом, до трансформатора Тесла.

Сама микросхема и нумерация ее выводов выглядят так:



Рассмотрим простую схему: мигающий светодиод.

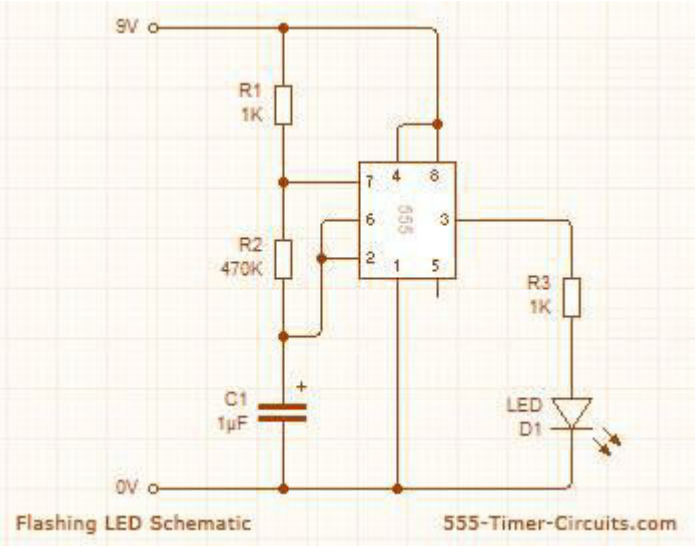


Схема весьма проста. Частота задается деталями R1, R2 и C1, и определяется по формуле:

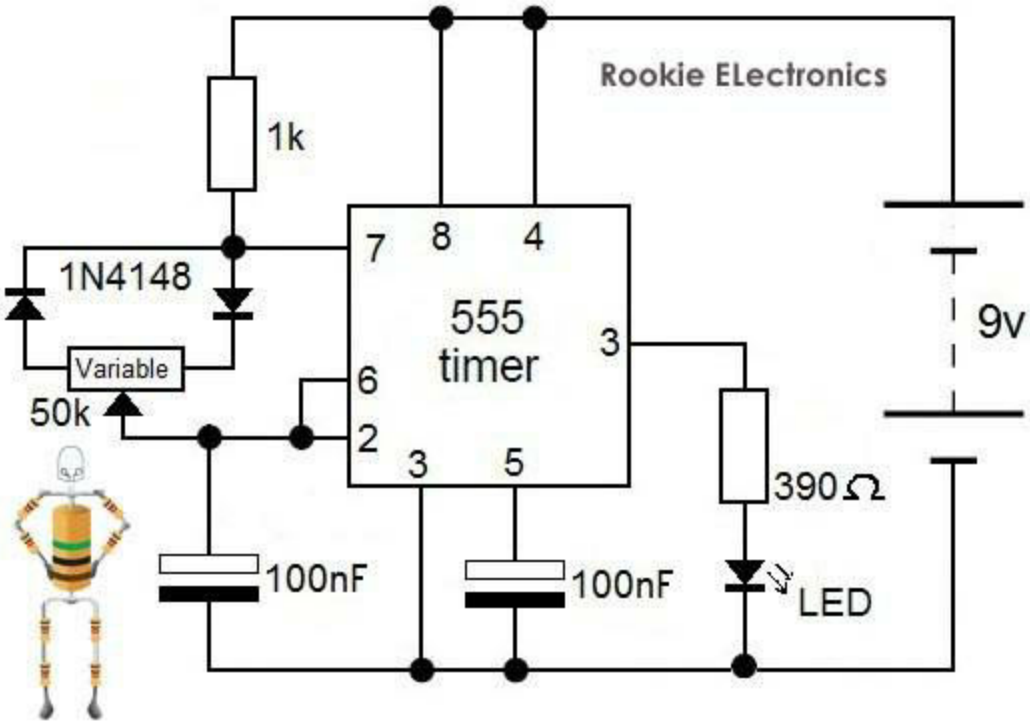
$$f_{osc} = \frac{1.4}{(R1 + 2R2)C1}$$

Частота, как мы знаем, измеряется в Герцах, 1Гц это одно колебание в секунду. NE555, R1, R2 и C1 создают генератор нужной частоты, справа мы видим уже знакомый нам светодиод с ограничивающим ток резистором.

Если схема собрана правильно, то мы увидим мигание светодиода.

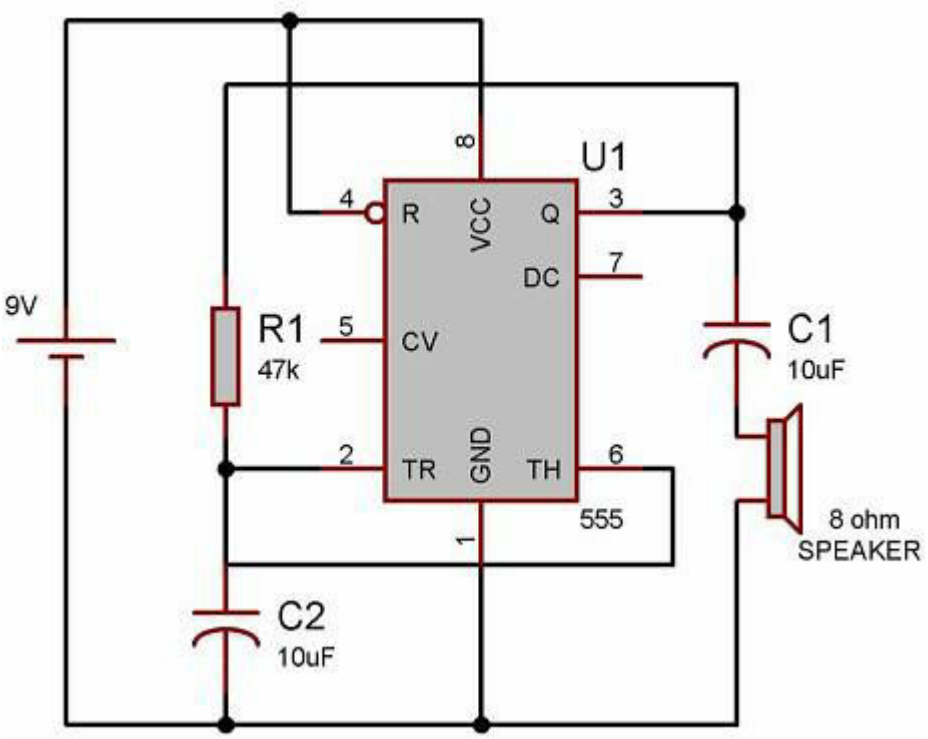
Если заменить резистор R1 на переменный, то частоту мигания можно будет изменять.

Немного усложнив схему, можно получить *диммер* - прибор, способный изменять яркость светодиода от нуля до максимума.



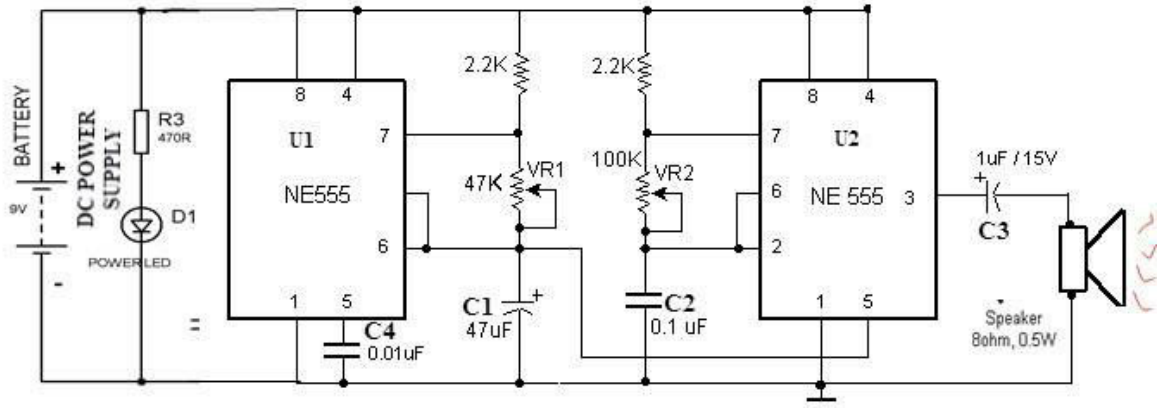
Такую схему можно использовать в качестве регулируемого ночника.

Как несложно догадаться, с помощью NE555 несложно воспроизвести и звук - нужно лишь изменить номиналы элементов, чтобы получить более высокую частоту, и поставить динамик вместо светодиода.

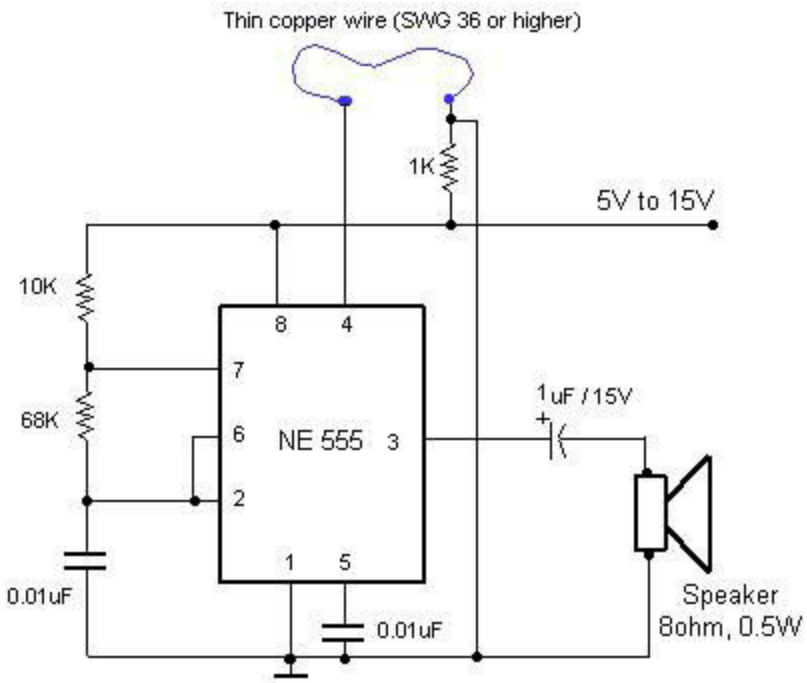


Здесь вместо светодиода и резистора подключен динамик с конденсатором.

Существует большое разнообразие схем с применением NE555. Например, подключив 2 микросхемы, можно получить “полицейскую сирену”:



Еще одна несложная схема - сигнализация, которая подаст звуковой сигнал при обрыве провода:



Аналогично, с применением NE555 есть схемы датчика протечки воды, ультразвукового отпугивателя собак, автоматического включения освещения с фоторезистором, и многое другое. Есть даже книга “Радиолюбительские схемы на ИС типа 555”, скачать ее можно в Интернете.

Самостоятельная работа: используя динамик, переменный резистор и NE555, собрать звуковой генератор, подобрав параметры так чтобы диапазон частот попадал в интервал 0-15КГц. С этим генератором легко проверить, насколько высокие звуки может слышать человек. Этот опыт можно провести с друзьями или одноклассниками - у каждого человека порог слышимости различный, более того, он меняется с возрастом.

1.8 Полевые и биполярные транзисторы

В предыдущей главе описывалась схема ночника, в которой яркость светодиода регулировалась от нуля до максимума. Но что делать, если мы захотим подключить целую светодиодную ленту чтобы осветить всю комнату? Включить ее напрямую к выводу микросхемы мы не можем, потребляемый ток слишком велик. На помощь придет *полевой транзистор*.

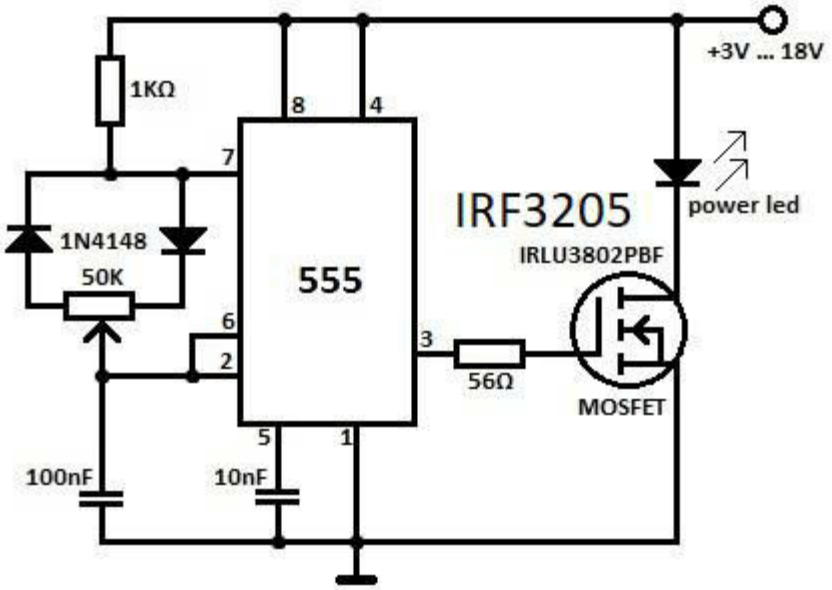
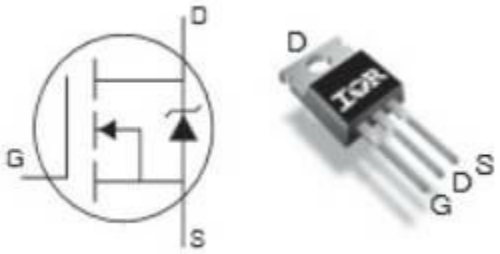


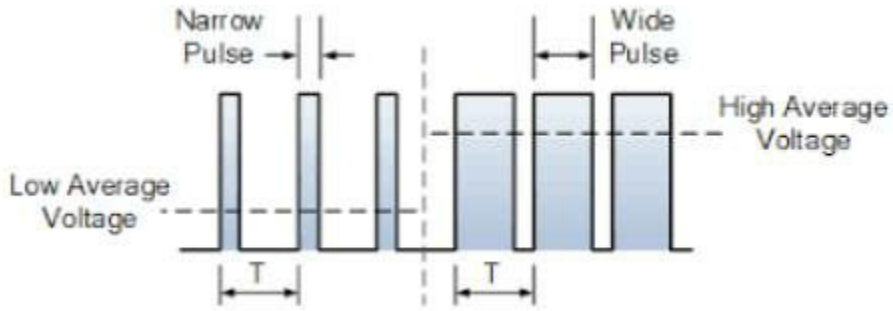
Схема подключения выводов транзистора показана на рисунке.



Упрощенно говоря, полевой транзистор - это электронный ключ, способный с помощью небольшого входного напряжения, управлять гораздо более мощной нагрузкой. Это как раз то, что нужно в нашей схеме.

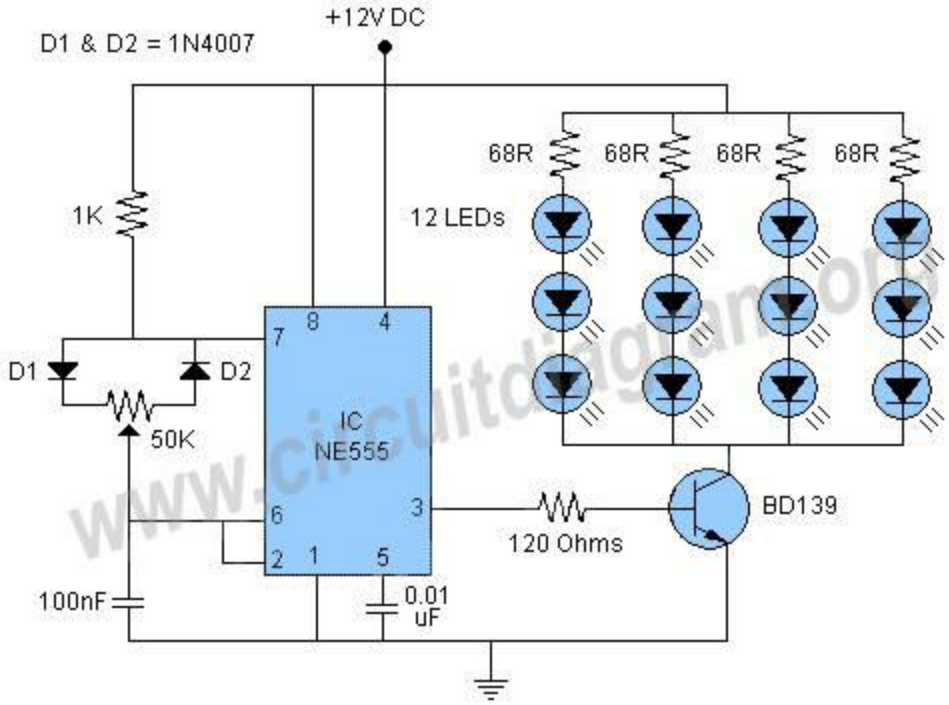
Собрав схему, как показано на рисунке, мы можем подключить светодиодную ленту и изменять ее яркость вращением переменного резистора.

Кстати, как же в действительности изменяется яркость свечения? Здесь применяется так называемая *широтно-импульсная модуляция* (ШИМ). В ней меняется не яркость светодиода, а продолжительность периодов его свечения:

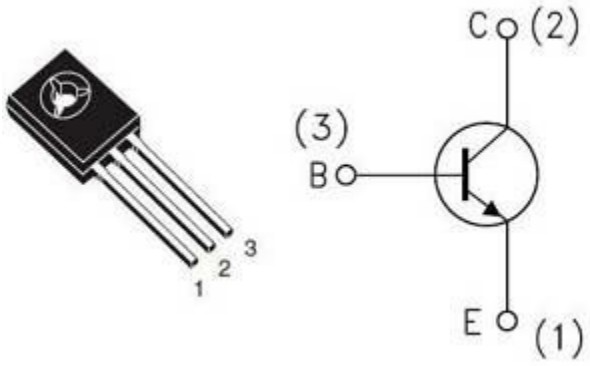


Человеческий глаз не может видеть пульсацию с частотой тысячи раз в секунду, и воспринимает это как более или менее яркий свет, в зависимости от вида пульсаций.

Аналогичную схему можно собрать на другом виде транзистора - *биполярном*.



Такой транзистор подключается следующим образом:



С точки зрения физических процессов, принцип работы биполярного и полевого транзисторов, различен, но конечный результат для нас тот же - небольшое изменение входного тока базы (обозначена буквой B) вызывает значительное изменение тока коллектор-эмиттер (C-E).

Транзисторы также активно используются в усилителях звуковой и радиочастоты, как управляющие элементы в блоках питания, в компьютерной технике, и так далее. Фактически это один из основных элементов современной схемотехники. Но “в чистом виде” нам его использовать практически не придется - в основном, мы будем использовать готовые микросхемы (все они внутри себя, разумеется, содержат транзисторы).

На этом мы закончим поверхностное знакомство с основными электронными компонентами, и перейдем к цифровой технике. Желающие углубленно изучить аналоговую схемотехнику, могут найти в интернете книгу Хоровица и Хилла “Искусство схемотехники” (The Art of Electronics), которая была выпущена еще в 80е, но до сих пор актуальна.

Часть 2. Знакомство с Arduino

В настоящее время существует большое количество различных микроконтроллеров - STM, Atmega, PIC и пр. Микроконтроллер - это по сути, небольшой но полноценный компьютер, имеющий оперативную и флеш-память, тактовый генератор, порты ввода-вывода для связи с “внешним миром”.

Типичный микроконтроллер выглядит примерно так:

Чтобы его использовать, необходимо:

- припаять его к печатной плате,
- добавить элементы, минимально необходимые для работы контроллера (питание, reset, тактовый генератор, и пр),
- добавить специальную схему для загрузки программы (“прошивки”),
- установить на ПК так называемую “среду разработки” (IDE) для написания кода.

Все это весьма трудоемко, поэтому в 2003м году итальянские инженеры (тогда еще студенты) придумали разместить все это на одной готовой плате. Так появился проект Arduino. Система стала настолько популярной, что к 2013 году на руках у пользователей было уже 700000 плат.

Arduino выглядит примерно так:

Она содержит:

- уже готовый к работе микроконтроллер,
- модуль для подключения к компьютеру по USB - через него осуществляется загрузка и отладка программы,
- базовый набор функций (светодиод, кнопка reset),
- большое количество выводов для подключения различных устройств (кнопки, экраны, датчики).

Существуют разные варианты плат - Arduino Uno, Arduino Mega,

Arduino Nano и пр. Также можно приобрести различные платы расширения, например плату управления мотором, или плату с ЖК-экраном.

Помимо плат, существует и бесплатная среда разработки Arduino IDE, позволяющая писать код и загружать программу в плату. Загруженная программа сохраняется в Arduino и после отключения питания, готовую плату потом можно использовать отдельно от компьютера.

2.1 Основы языка Си

Для начала … отложим плату Arduino в сторону, и научимся писать несложные программы на языке Си. Ведь центральный процессор Arduino - это почти полноценный компьютер, а значит его нужно будет программировать. Для этого используется язык Си, весьма популярный для написания программ различных микроконтроллеров.

Чтобы писать программу на каком-либо языке программирования, нужны специальный редактор (так называемая “среда разработки” или IDE) и компилятор, преобразующий текст в готовую программу. Для упрощения мы воспользуемся онлайн компилятором, для чего можно зайти на сайт https://repl.it/languages/c или https://www.onlinegdb.com/online_c_compiler.

Простейшая программа на Си выглядит так:

```
#include <stdio.h>

int main(void) {
printf("Hello World\n");
return 0;
}
```

Директива *#include* подключает служебный файл, в котором описаны необходимые нам функции. Функция *printf* выводит текст на экран. Запустим программу нажатием кнопки “>”, и справа мы увидим результат ее выполнения - появится текст Hello world.

Мы также можем создать целочисленную переменную, написав:

int i = 42;

Или вещественную:

float a = 1.0;

С переменными можно осуществлять математические действия:

float b = 3*val + 5;

Можно увеличить или уменьшить значение переменной:

```
i = i+1;       // Более короткая запись: i += 1; или еще короче i++;
i = i-1;            // Более короткая запись: i -= 1; или еще короче i--;
```

Можно вывести на экран значения переменных:

```
int i = 42;
float a = 1.0;
printf("I = %d, A = %f\n", i, a);
```

Нужный фрагмент программы можно повторить нужное число раз с помощью оператора **for**. Выведем значение переменной 10 раз:

```
for(int v=0; v<10; v++) {
printf("I = %d, A = %f\n", i, a);
}
```

Блок, который будет повторен, выделяется фигурными скобками {

и }. Оператор for можно использовать не только для вывода, например вот так можно подсчитать и вывести сумму квадратов чисел от 1 до 100:

```
#include <stdio.h>

int main(void) {
int sum = 0;
for(int v=0; v<100; v++) {
sum += v*v;
}
printf("Sum = %d\n", sum);
return 0;
}
```

В программе могут также быть условия, которые записываются в виде оператора if: фрагмент кода внутри фигурных скобок выполнится только если условие истинно.

```
if (sum > 100) {
printf("Sum > 100\n");
}
```

Это небольшое введение позволит нам ориентироваться в коде программ для Arduino. Желающие могут найти более подробное руководство по C++ самостоятельно.

Самостоятельная работа: найти любые примеры из школьного задачника по математике, и решить их с помощью программы на C.

2.2 Типы данных в Arduino

Мы уже рассмотрели некоторые виды переменных, например int и float. Рассмотрим более подробно типы данных, доступные для Arduino.

Arduino Data Types	Value Assigned	Value Ranges
boolean	8 Bit	True or False
byte	8 Bit	0 to 255
char	8 Bit	-127 to 128
unsigned char	8 Bit	0 to 255
word	16 Bit	0 to 65535
unsigned int	16 Bit	0 to 65535
int	16 Bit	-32768 to 32767
long	32 Bit	-2,147,483,648 to 2,147,483,647
float	32 Bit	-3.4028235E38 to 3.4028235E38

Переменные можно разбить на 3 основные группы.

Целочисленные. Они содержат целые числа, например в диапазоне -32768..32767, или 0..65535. Каждой переменной выделяется определенный размер памяти, который и определяет, насколько большое число можно хранить.

Вещественные. Это упрощенно говоря, все нецелые числа - например 2.5, или 3.14. Они занимают в памяти больше места и работа с ними выполняется гораздо медленнее, так что использовать их на маломощных процессорах вроде Arduino рекомендуется лишь при реальной необходимости.

Символьные. Это та же целочисленная переменная, только используемая для хранения кодов букв. Одна переменная хранит код одного символа в формате ASCII. Строка представляется в виде *массива символов*.

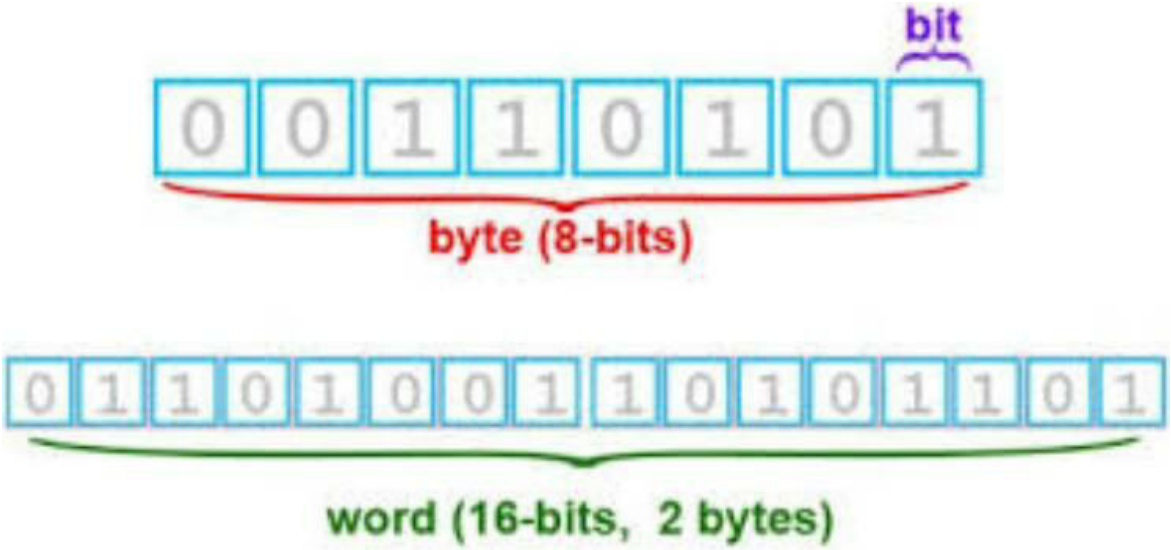
Таким образом, мы можем создать разные типы переменных:

```
int a = 5;
float pi = 3.1415;
```

```
char s = "B";
char str[10] = "Hello";
```

Важно помнить, что в отличие от обычных персональных компьютеров, платы Arduino имеют ограниченный объем памяти, так что ее надо экономить - использовать тип, минимально достаточный для хранения данных. К примеру, если нужно хранить 255 кодов вводимых клавиш “0”..”9”, то для этого достаточно написать *byte symbols[255]*. Если же мы напишем *int symbols[255]* - результат будет тот же, но объем занимаемой памяти будет вдвое больше. Точные значения размера переменных можно посмотреть в таблице, приведенной выше.

Кстати, откуда взялись подобные ограничения? Это связано с тем, что физически данные хранятся в виде бит, в так называемой **двоичной системе счисления**.



Один *бит* - это минимально возможная величина, способная принимать значения “0” или “1”. 8 бит объединяются в *байт*. Любое число можно представить в виде двоичной записи, как сумму чисел степени 2 (1,2,4,8,16,32,...), например:

5 = **0***128 + **0***64 + **0***32 + **0***16 + **0***8 + **1***4 + **0***2 + **1***1 = 00000101.

Нетрудно подсчитать, что максимально возможное число при таком виде записи будет составлять 128+64+32+16+8+4+2+1 = 255. Это так называемое *беззнаковое целое число* (unsigned integer). Если один из

разрядов зарезервировать под знак (+ или -), то останется 7 разрядов для чисел, что и соответствует -127..128.

Для 16-битных чисел диапазон значений соответственно больше, при желании их нетрудно подсчитать самостоятельно.

Вещественные числа хранятся в немного более сложном формате, так называемом формате “чисел с плавающей точкой”, состоящего из двух хранимых величин - мантиссы и экспоненты. К примеру, число 3.14 в двоичном виде будет храниться так: 01000000010010001111010111000011. Первый 0 - это знак (+), 1000000 = 128 - это экспонента, а 10010001111010111000011 - это мантисса. Вещественное число получается по формуле:

$$value = (1 + b_0/2 + b_1/4 + b_2/8 + b_3/16 + \dots) * 2^{e-127}$$

Действительно, нетрудно подсчитать, что:

$$(1 + 1/2 + 1/16 + ...) * 2^1 = 3.14$$

Важно отметить, что в отличие от целых чисел, вещественные числа представляются с некоторой небольшой погрешностью, она невелика, но она есть.

Разумеется, для пользователя все это прозрачно - можно написать float pi = 3.14, и не задумываться как оно внутри хранится. Но чтобы писать эффективные программы, про хранение данных хотя бы в общих чертах необходимо знать.

Теперь, вооруженные всеми этими знаниями, мы можем вновь взять плату Arduino обратно, и продолжить эксперименты с “железом”.

2.3 Мигаем светодиодом

В предыдущей части мы уже подключали светодиод к источнику питания. Arduino Nano содержит уже подключенный светодиод, для его использования достаточно лишь написать программу.

Каждый вывод микроконтроллера может работать в двух режимах:

как **вход** (input), или как **выход** (output). Режим “выход” - это то, что нам нужно, при подаче на вывод “1” на выводе появляется напряжение 5В, при подаче на вывод “0” напряжение становится равным 0.

Для того, чтобы запрограммировать Arduino, нам понадобится:

- Скачать и установить Arduino IDE со страницы https://www.arduino.cc/en/Main/Software

- Подключить плату к компьютеру, в системе при этом должен появиться новый порт, например COM7

- Запустить Arduino IDE, набрать текст программы, как показано на рисунке

- Нажать кнопку левую кнопку “Проверить” и соседнюю “Загрузить” - программа будет загружена в Arduino.

Если все было сделано правильно, через несколько секунд мы

увидим мигающий на плате светодиод (при самом первом запуске, Arduino IDE вначале предложит сохранить текст исходного кода в файле). Если вместо сообщения “Done compiling” мы видим ошибку, нужно внимательно прочитать, что она значит. К примеру, может быть выбран неправильный порт, или в тексте программы есть опечатка.

Теперь, можно отложить мигающую плату, и разобраться, что же мы такое сделали.

Разберем программу по шагам.

int led = 13; Здесь создается глобальная переменная, хранящая номер вывода “13”, к которому подключен светодиод (номера выводов можно посмотреть в документации к плате).

void **setup()** - здесь объявляется функция setup, которая будет вызвана только один раз при запуске программы. В ней мы настраиваем наш вывод 13 как “выход”, вызовом функции **pinMode(led, OUTPUT);**.

Функция **loop()**, в отличие от setup, выполняется постоянно, бесконечное число раз. В ней мы и размещаем всю логику работы программы. В данном случае, логика проста - мы посылаем в порт логическую “1” командой **digitalWrite(led, HIGH)**, затем ждем одну секунду с помощью вызова **delay(1000)**, затем посылаем логический “0”, опять ждем. Данный цикл будет автоматически повторяться, пока плата включена и работает.

Огромный плюс использования микроконтроллеров - в их огромной гибкости, изменяя код, мы можем полностью менять логику работы программы. Например, несложно сделать чтобы светодиод мигал в режиме “2 коротких, 1 длинный”, для этого достаточно лишь изменить текст кода:

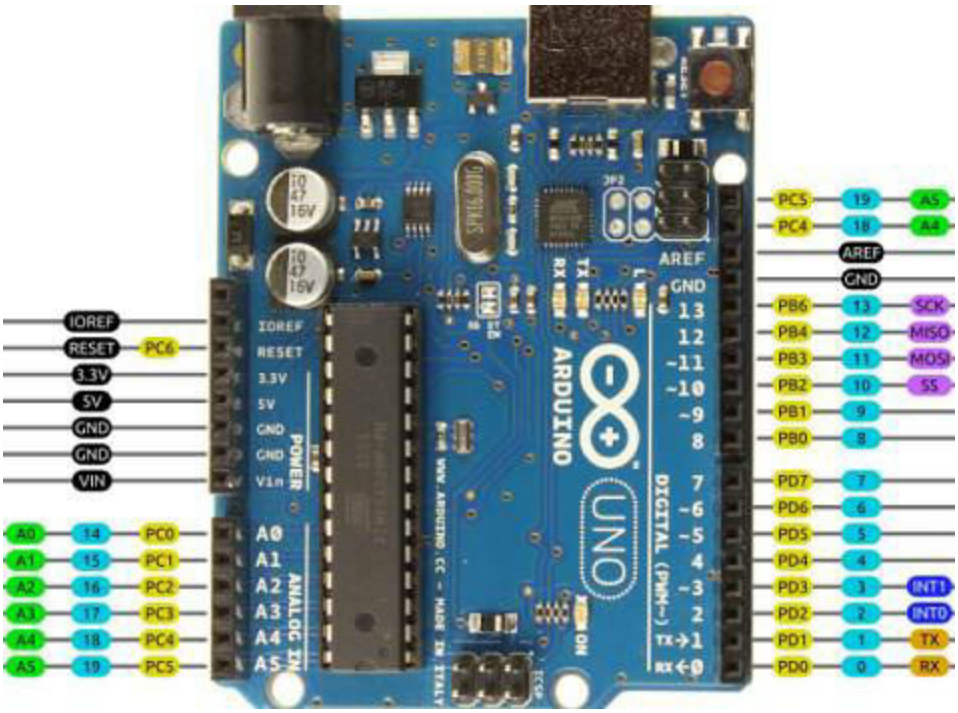
```
void loop() {
digitalWrite(led, HIGH);
delay(500);
digitalWrite(led, LOW);
delay(500);
digitalWrite(led, HIGH);
```

```
delay(500);
digitalWrite(led, LOW);
delay(500);
digitalWrite(led, HIGH);
delay(2000);
digitalWrite(led, LOW);
delay(2000);
}
```

Не нужно ни пайки, ни какой-либо перенастройки, все делается чисто программно.

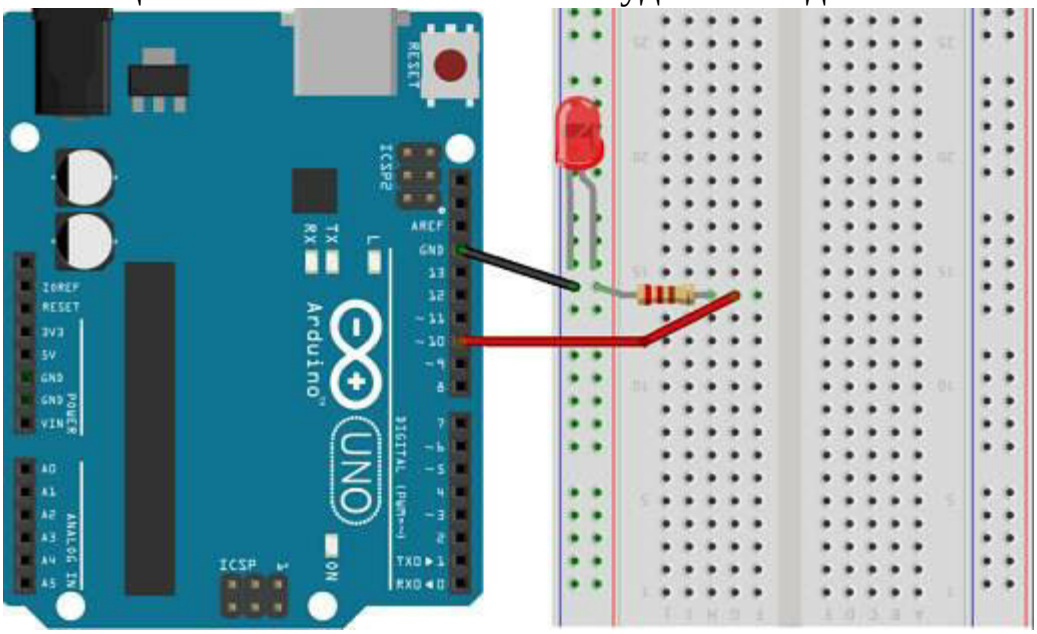
Кстати, зачем нужен вызов функции delay? Все просто, без нее программа тоже будет работать - но светодиод будет переключаться со скоростью тысячи раз в секунду, что будет неразличимо глазом. Тактовая частота процессора составляет несколько мегагерц, и без пауз программа будет работать слишком быстро.

Можно ли подключить светодиод к другому выводу, или подключить несколько светодиодов? Разумеется, можно. Для этого нужно найти инструкцию к плате, где будут указаны номера выводов (номера подписаны и на самой плате). Для Arduino Uno такая схема выглядит примерно так:



Далее, достаточно подключить к нужному выводу (например это может быть пин “10”) светодиод, не забыв и ограничительный резистор. Вторым выводом будет общий вывод, или GND (это аналог вывода “-” в схеме с батарейкой из первой части книги). На плате несколько выводов GND, можно использовать любой из них, они соединены вместе.

Схема целиком на макетной плате будет выглядеть так:



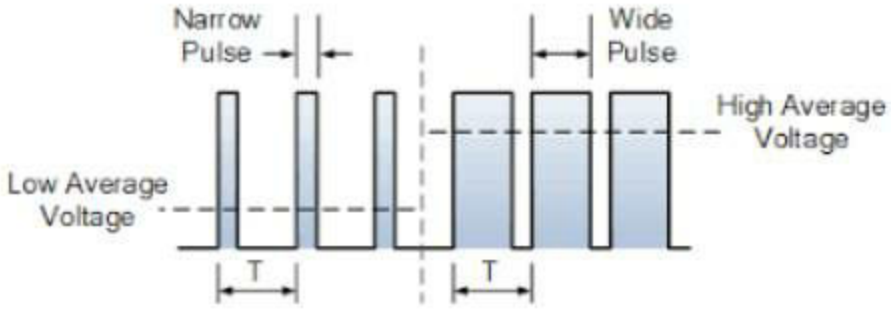
Разумеется, текст кода тоже придется изменить, поменяв номер вывода с 13 на 10.

Самостоятельная работа #1: Замедлить скорость мигания светодиодов до 5-10с. Тестером померять напряжение на выходе Arduino, и убедиться что оно изменяется от 0 до 5В с соответствующей частотой.

Самостоятельная работа #2: подключить 2-3 дополнительных светодиода, каждый через свой токоограничительный резистор. Добавить код для их переключения, можно также поэкспериментировать с различными световыми эффектами (поочередное или параллельное мигание и пр).

2.4 Мигаем светодиодом: широтно-импульсная модуляция

В первой части мы уже рассматривали изменение яркости светодиода с помощью ШИМ - широтно-импульсной модуляции. Там мы использовали таймер NE555, чтобы создать напряжение такого вида:



То же самое легко запрограммировать с помощью контроллера. Напишем программу, которая будет плавно повышать яркость светодиода от нуля до максимума.

```
int led = 13;

int pwm = 0;

void setup() {
pinMode(led, OUTPUT);
}
```

```
void loop() {
for(int i=0; i<1000; i++) {
digitalWrite(led, HIGH);
delayMicroseconds(pwm);
digitalWrite(led, LOW);
delayMicroseconds(100 - pwm);
}
pwm += 1;
if (pwm > 100) pwm = 0;
}
```

Мы создали глобальную переменную **pwm**, в которой сохраняется текущее значение уровня заполнения в процентах. Дальше мы включаем “высокое” и “низкое” состояние вывода, в соответствии с этим значением - когда одно значение велико, второе, наоборот, мало. Цикл “for(int i=0; i<1000; i++)” повторяет участок кода 1000 раз - без него светодиод менял бы яркость слишком быстро.

Если загрузить этот код, мы увидим плавно увеличивающий яркость светодиод. Но у вышеприведенного кода есть недостатки. Во-первых, он довольно-таки громоздкий - слишком много строк для переключения только одного вывода. Во-вторых, процессор занят только переключением светодиода, любая другая задача нарушит согласованность временных интервалов. К счастью для нас, разработчики процессора пошли навстречу пользователям, и формирование ШИМ может выполняться автоматически, на аппаратном уровне. Для этого достаточно использовать функцию analogWrite, в качестве параметра указав степень заполнения в виде параметра 0..255.

Например, для установки яркости 50% достаточно написать:

```
analogWrite(led, 128);
```

Процессор сам сделает все остальное, и сформирует на выходе нужный сигнал. Наш код в это время может делать что-то другое, например выводить информацию на ЖК-экран. Единственное ограничение - режим ШИМ может работать не на всех выводах, это определяется моделью процессора. Например, для Arduino Uno для

ШИМ доступны только номера выводов 3, 5, 6, 9, 10, и 11.

Разумеется, с помощью ШИМ управлять можно не только яркостью одного светодиода, но и более мощной нагрузкой постоянного тока (лампа, светодиодная лента и пр), подключив ее через транзистор.

Самостоятельная работа: переписать вышеприведенную программу с использованием analogWrite. Проверить работоспособность, подключив светодиод с резистором к соответствующему выводу.

2.5 Вывод данных через Serial port

В простых случаях можно понять, что делает программа, просто посмотрев на ее текст. Но увы, так бывает далеко не всегда. Более сложные платы, например STM32, имеют специальный разъем для программирования, позволяющий не только загружать программы, но и задавать точки останова, просматривать значения переменных, выполнять программу по шагам. На Arduino такой возможности нет, зато есть возможность вывода данных через “последовательный порт”.

На старых компьютерах были такие порты, называемые COM и LPT. Разумеется, физически отдельного COM-порта на Arduino нет. Его роль играет микросхема FTDI, создающая виртуальный порт при подключении платы по USB.

Еще раз посмотрим в правый нижний угол Arduino IDE.

Отображаемый в углу “COM7” - это и есть тот самый порт, через который Windows “общается” с Arduino. Его несложно использовать в различных программах, например передавать с платы на компьютер значения датчика температуры. Но не менее важная функция - это вывод значений переменных, что позволяет проверить правильность работы программы. Этот процесс называется “отладка” или “debugging”, что переводится как “поиск жучков”. Самые первые компьютеры тоже работали в двоичной системе счисления, но вместо транзисторов, имели механические реле. По легенде, бабочка попала в такое реле, из-за чего контакты перестали замыкаться, и разумеется, программа стала работать неправильно. Много лет прошло, и компьютеры уже давно не механические, а название так и осталось (подробнее можно прочитать в Википедии).

Рассмотрим вывод данных в порт из программы:

```
void setup() {
// Открыть порт на нужной скорости

Serial.begin(9600);

}

void loop() {
for(int x=0; x< 64; x++) {

// Вывод числа в разных форматах:

Serial.print(x);

Serial.print("\t");


Serial.print(x, DEC);

Serial.print("\t");


Serial.print(x, HEX);

Serial.print("\t");


Serial.println(x, BIN);

delay(200);

}

Serial.println();
}
```

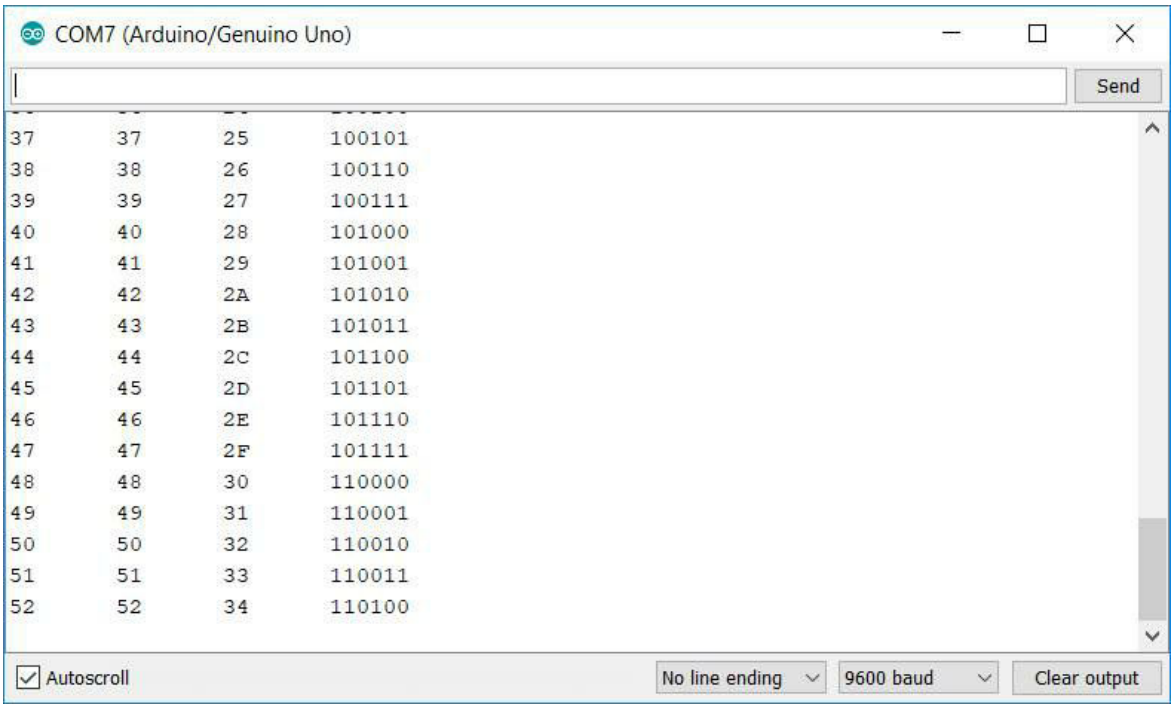
Рассмотрим код программы подробнее.

Скорость порта. Она задается в функции **setup()**. Для того, чтобы два устройства обменивались данными между собой, данные должны передаваться на одинаковой скорости. Скорость измеряется в *бодах*, и может быть различной, например 9600 или 115200 бод. Бод - это количество изменений сигнала в секунду. В нашем случае сигналы могут быть только двоичными, так что скорость в бодах соответствует скорости в битах в секунду. Можно использовать любую скорость, главное чтобы на приёмной и передающей сторонах они были одинаковыми.

Что будет если установить неправильную скорость? Ничего хорошего - система не знает, какой скорость должна быть, так что вместо данных мы получим просто мусор. Последовательный протокол - это одна из самых простых систем передачи данных, и какой-либо защиты или автоматического определения параметров тут нет, пользователь должен настроить все параметры самостоятельно.

Собственно, вывод данных, реализуется с помощью функции **Serial.print**. Тут все просто, функция посылает данные “как есть”, причем можно послать как текстовую строку, так и значение числовой переменной. Также для удобства чтения можно использовать вариант функции **println** - она делает так называемый “возврат каретки” (CR, carriage return), устаревший термин, обозначающий перевод курсора на новую строку. Как можно видеть в коде, число можно вывести в разных системах счисления - десятичной, 16-ричной, двоичной. Знак “табуляции” - "\t" вставляет отступ между числами, что также делает чтение более удобным.

Наконец, программа готова, загружаем ее в Arduino. Мы видим как светодиод на плате начинает мигать - Arduino передает данные в порт. Чтобы увидеть их, выбираем в Arduino IDE пункт меню Serial Monitor и видим наши данные.

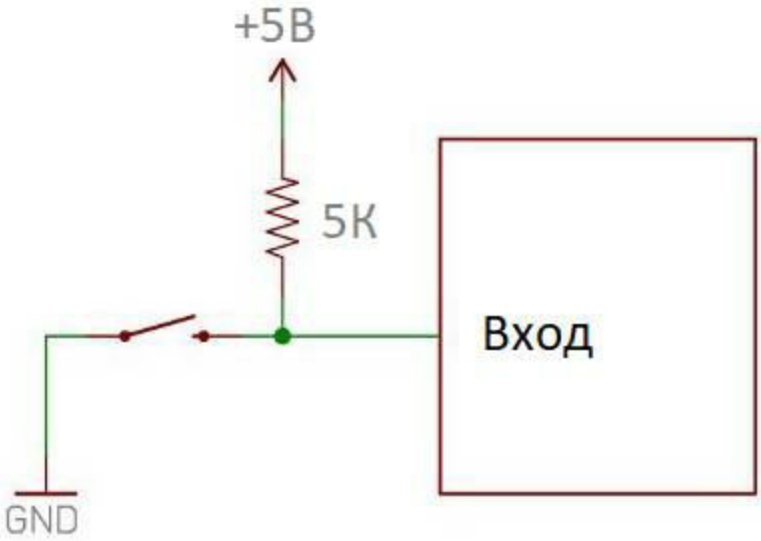


Для опыта можно попробовать выбрать другую скорость приема - мы увидим, что вместо данных на экране появляются нечитаемые символы.

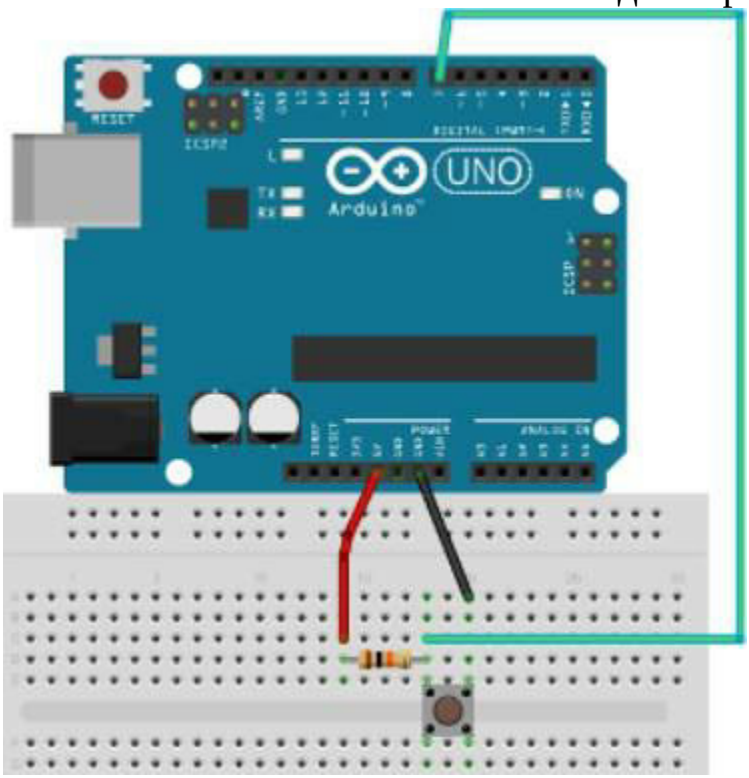
2.5 Ввод данных: определяем нажатие кнопки

Сложно найти хоть какое-либо электронное устройство, совсем не имеющее кнопок. Кнопки несложно подключить и к Arduino - любой вход микроконтроллера может работать не только как “выход”, но и как “вход”.

Сначала, кнопку надо подключить, как показано на схеме:



На макетной плате это может выглядеть примерно так:



Принцип работы схемы прост. Когда кнопка не нажата, вход Arduino подключен через резистор к линии “питания”, +5В, что соответствует логической единице. Если пользователь нажимает кнопку, напряжение на входе становится равным нулю.

Резистор - важный компонент схемы. Если бы его не было, при

отпущенной кнопке значение входа было бы неопределенным - вход Arduino ни к чему был бы не подключен. Точное значение резистора кстати, не столь важно, оно может быть и 1КОм, и 5КОм, и 10КОм. Такой резистор называется “подтягивающим” (pull up), т.к. он соединяет (подтягивает) вход к напряжению питания. Можно кстати, сделать и наоборот - резистор соединить с “землей”, а кнопку замыкать на питание. Такая схема называется pull down. Очевидно, что во втором случае при отпущенной кнопке напряжение будет равно нулю.

Теперь, когда мы собрали схему и разобрались с ней, напишем программу, читающую значение кнопки. Для наглядности, будем зажигать светодиод, когда кнопка нажата:

```
int buttonPin = 2;

int ledPin = 13;


void setup() {

// Вывод настроен как “выход”

pinMode(ledPin, OUTPUT);

// Вывод настроен как “вход”

pinMode(buttonPin, INPUT);

}


void loop() {

// Читаем состояние вывода:

int buttonState = digitalRead(buttonPin);


// Устанавливаем состояние светодиода:
```

```
if (buttonState == HIGH) {

// LED off

digitalWrite(ledPin, LOW);

} else {

// LED on

digitalWrite(ledPin, HIGH);

}

}
```

Как можно видеть, текст программы довольно прост. Сначала пин устанавливается как “вход” командой pinMode(buttonPin, INPUT), затем в функции loop состояние входа читается с помощью функции digitalRead. Напомним, что функция loop вызывается постоянно, пока схема включена и контроллер работает.

Сделаем программу чуть более полезной. Например, создадим таймер для чистки зубов - при нажатии кнопки светодиод будет гореть ровно 2 минуты. Для создания задержки будем использовать функцию delay, которая в качестве параметра принимает длительность в миллисекундах.

Нам достаточно изменить функцию loop.

```
void loop() {
// Ждем пока кнопка будет нажата

while (digitalRead(buttonPin) == HIGH) delay(100);

// Зажигаем светодиод
digitalWrite(ledPin, HIGH);
delay(120*1000);
```

```
// Гасим светодиод
digitalWrite(ledPin, LOW);

}
```

Логика работы программы достаточно проста, и не требует отдельных пояснений. Программа ждет нажатия кнопки, для чего используется функция **while** - она выполняет нужный фрагмент кода, пока условие истинно. После нажатия кнопки, программа включает светодиод, затем запускается 2х-минутная пауза с помощью функции **delay**, затем цикл повторяется.

Самостоятельная работа #1: Изменить таймер так, чтобы во время 2х-минутного интервала светодиод мигал с частотой 1 раз в секунду. Для этого, функцию delay следует заменить на цикл, внутри которого светодиод будет включаться и выключаться.

Самостоятельная работа #2: Усложнить программу, сделав так, чтобы к концу интервала (например, последние 10 секунд) светодиод мигал чаще, показывая что время подходит к концу.

2.6 Ввод аналоговых величин

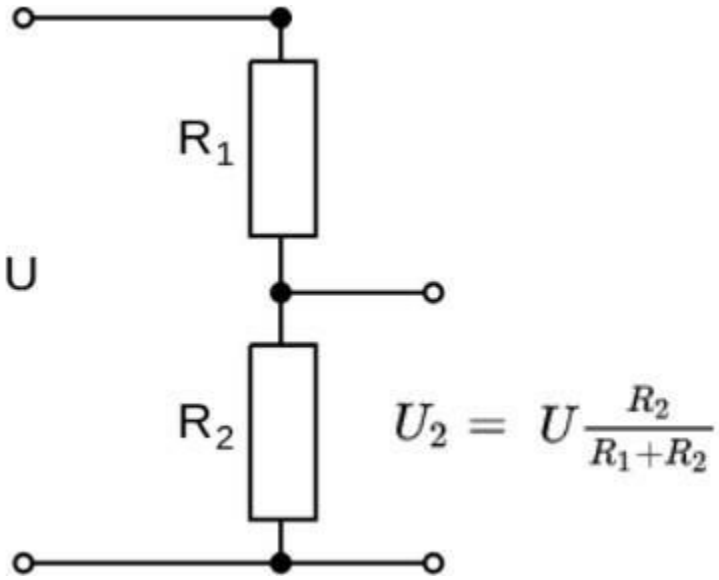
Как мы видели в предыдущей главе, получить состояние кнопки весьма просто - она или нажата или нет, что соответствует либо “0”, либо “1”. Но Arduino также имеет возможность измерять напряжение напрямую, для этого есть специальные аналоговые входы (analog Inputs). Это часто бывает полезно, например, сопротивление фоторезистора зависит от освещенности, и мы можем использовать это в программе для включения света, сопротивление датчика влажности может показывать, что пора полить цветок, и т.д.

Другой пример использования аналоговой величины: диджейский пульт.

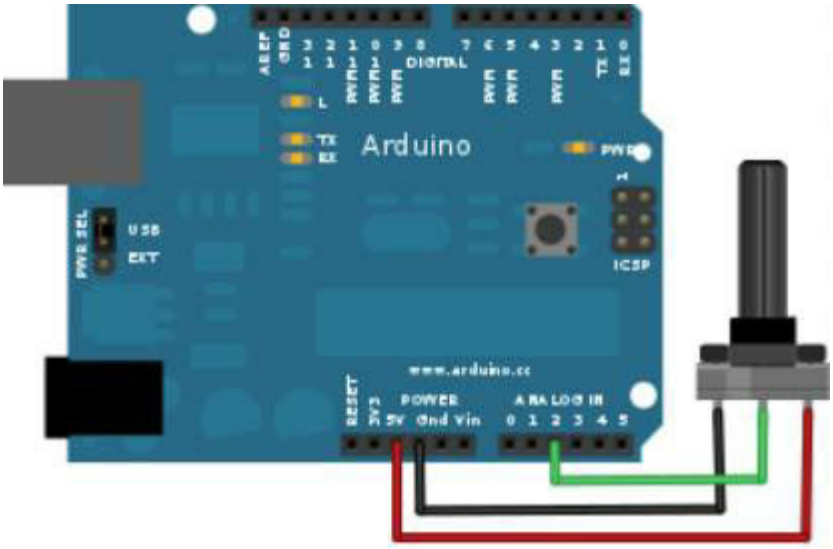
Сам принцип основан на работе делителя напряжения: если переменный резистор подключить к источнику напряжения, то значение его выхода будет варьироваться от нуля до максимума.

Сам переменный резистор может быть круглым, либо в виде слайдера:

Точное значение резистора не так уж важно, он может быть 5КОм, 10КОм, 50КОм - в делителе напряжения важно соотношение сопротивлений, а не их абсолютное значение.



Дополним таймер из предыдущей главы переменным резистором, которым можно будет регулировать интервал в диапазоне от 0 до 10 минут. Для подключения резистора нужно использовать входы, помеченные как Analog input, в нашем случае A3.



Теперь мы можем считывать положение ручки резистора, используя функцию **analogRead**. Есть лишь одна небольшая тонкость - analogRead возвращает значения от 0 до 1024. Чтобы преобразовать их в интервал от 0 до 600 секунд, мы домножаем полученные значения на 600/1024. Тип **unsigned long** используется потому, что максимальное значение 1024*600 = 614400, что уже превосходит диапазон значений

int, который составляет в Arduino -32768...32767.

Код программы целиком:

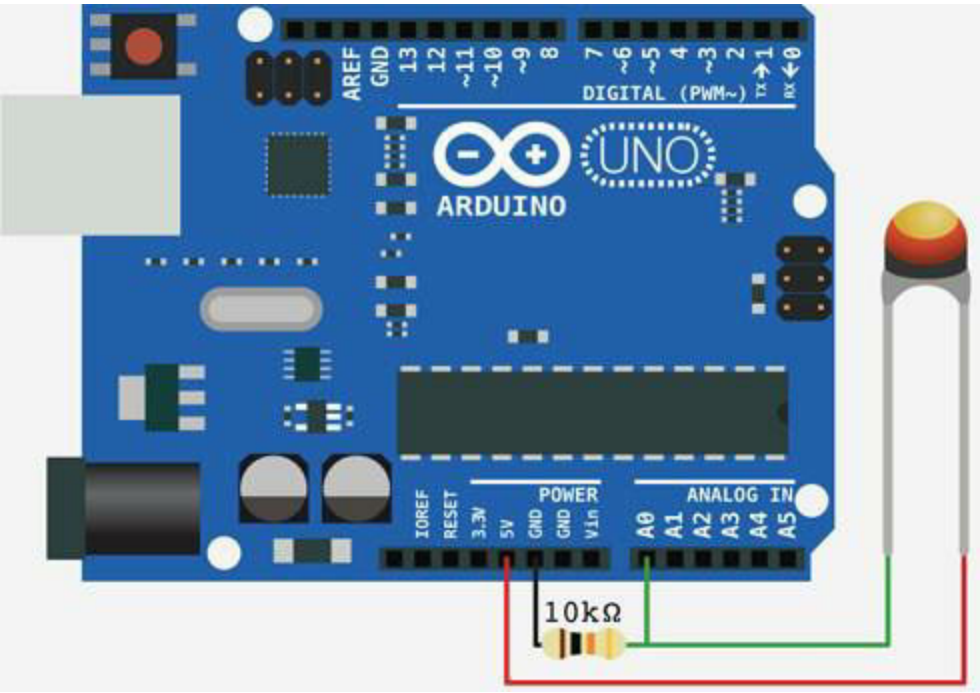
```
int buttonPin = 2;

int analogPin = 3;
int ledPin = 13;


void setup() {

// Вывод настроен как “выход”

pinMode(ledPin, OUTPUT);

// Вывод настроен как “вход”

pinMode(buttonPin, INPUT);

}


void loop() {
// Читаем положение потенциометра (0..1023)
int pos = analogRead(analogPin);
unsigned long time_sec = pos*600/1024;

// Ждем пока кнопка будет нажата

while (digitalRead(buttonPin) == HIGH) delay(100);

// Зажигаем светодиод
digitalWrite(ledPin, HIGH);
delay(time_sec*1000);
// Гасим светодиод
digitalWrite(ledPin, LOW);

}
```

В следующей главе мы подключим к Arduino динамик, что позволит создать уже полноценное и законченное устройство: кухонный таймер.

Самостоятельная работа #1: сделать шкалу для переменного резистора, как показано на рисунке. Оценить точность работы таймера, например с помощью программы для смартфона.

Разумеется, есть множество других разнообразных устройств, напряжение от которых можно считывать с помощью analogRead.

Термистор - это резистор, сопротивление которого зависит от температуры. Его подключение аналогично переменному резистору.



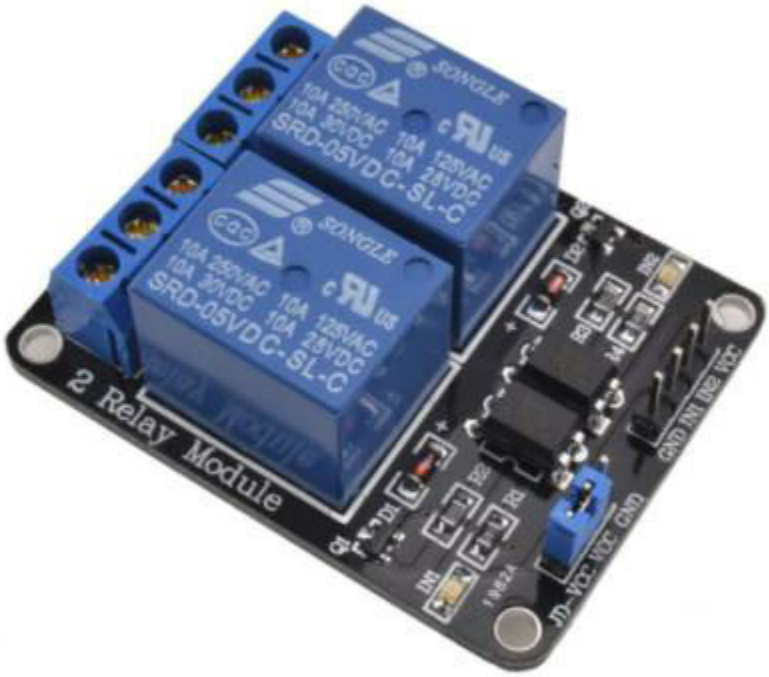
Подключив его в схему как показано на рисунке, можно измерять, например, температуру воздуха на улице.

Фоторезистор - как понятно из названия, изменяет свое сопротивление в зависимости от освещенности.

Датчик атмосферного давления: напряжение на его выходе пропорционально давлению.

В продаже также есть датчики влажности воздуха или почвы. Все это позволяет создать несложные системы автоматизации, например для освещения или обогрева теплицы. Разумеется, Arduino не может напрямую управлять мощной нагрузкой, для этого в продаже есть

специальные реле, вроде таких:



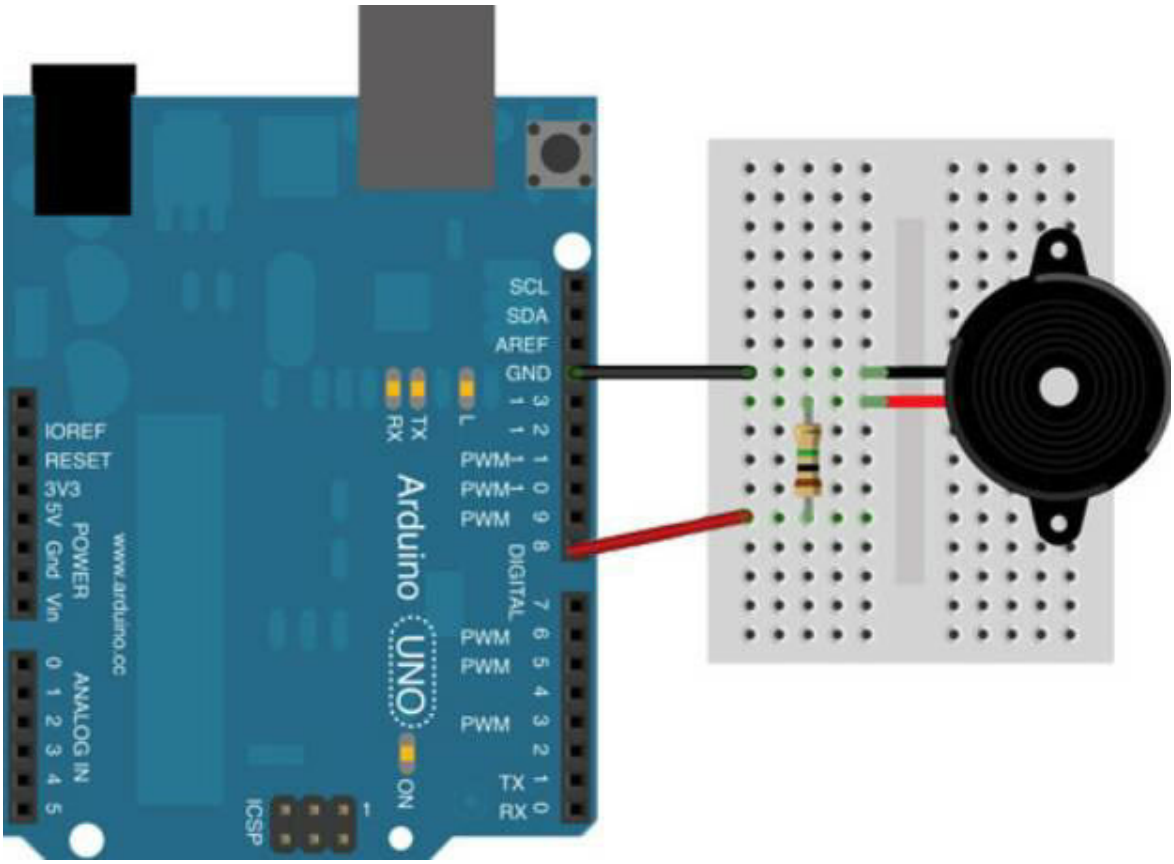
Множество современных датчиков имеют цифровые интерфейсы, но за счет своей простоты и дешевизны, аналоговые датчики тоже весьма широко используются.

Самостоятельная работа #2: подключить к Arduino термистор, вывести значения принимаемые от analogRead в последовательный порт, что позволит просматривать их на компьютере. Погружая термистор в лед или кипяток, построить график зависимости значений от температуры. В следующих главах мы рассмотрим подключение ЖК-экрана, что позволит нам на этом принципе сделать термометр.

2.6 Вывод звука

Мы уже подключали светодиод к Arduino. Практически тем же способом можно подключить к Arduino пьезодинамик, что позволит нам создавать несложные звуки.

Пьезодинамик подключается к выводу Arduino примерно также, как и светодиод, через ограничительный резистор сопротивлением 100-200 Ом.



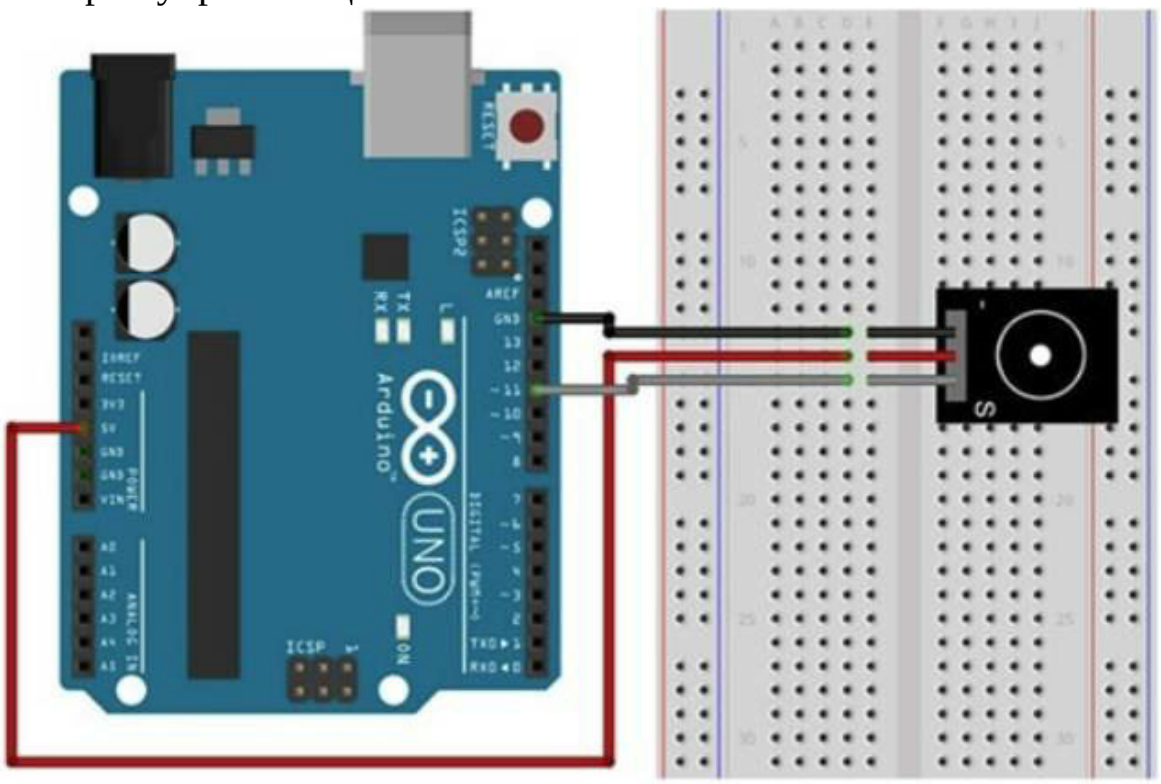
Для вывода звука используется функция **tone**, которая имеет два параметра - номер вывода и частоту звука. Для отключения звука есть функция со схожим названием **noTone**.

Простейший код, в котором Arduino будет издавать (весьма противные) звуки каждую секунду, выглядит так:

```
const int buzzer = 8; // arduino pin 8

void setup(){
pinMode(buzzer, OUTPUT);
}

void loop() {
tone(buzzer, 1000);       // 1000Гц = 1КГц
delay(1000);       // 1с
noTone(buzzer);       // стоп
delay(1000);       // пауза 1с
```

}

Кстати, в продаже бывают и так называемые “активные пьезодинамики” (active buzzer). Они имеют 3 входа, один из которых +5В, а второй управляющий.



Использование такого динамика проще - частоту можно не задавать, достаточно просто установить в “1” соответствующий вывод. Для этого достаточно заменить функцию tone(buzzer, 1000); на digitalWrite(buzzer, HIGH); и функцию noTone(buzzer); на digitalWrite(buzzer, LOW). Такой динамик обычно громче, но отсутствие возможности смены частоты звука может быть недостатком.

Самостоятельная работа #1: С помощью пьезодинамика воспроизвести гамму или несложную мелодию, воспользовавшись таблицей частот нот в герцах.

До
262
Соль
392

Ре
294
Ля
440

Ми
330
Си
494

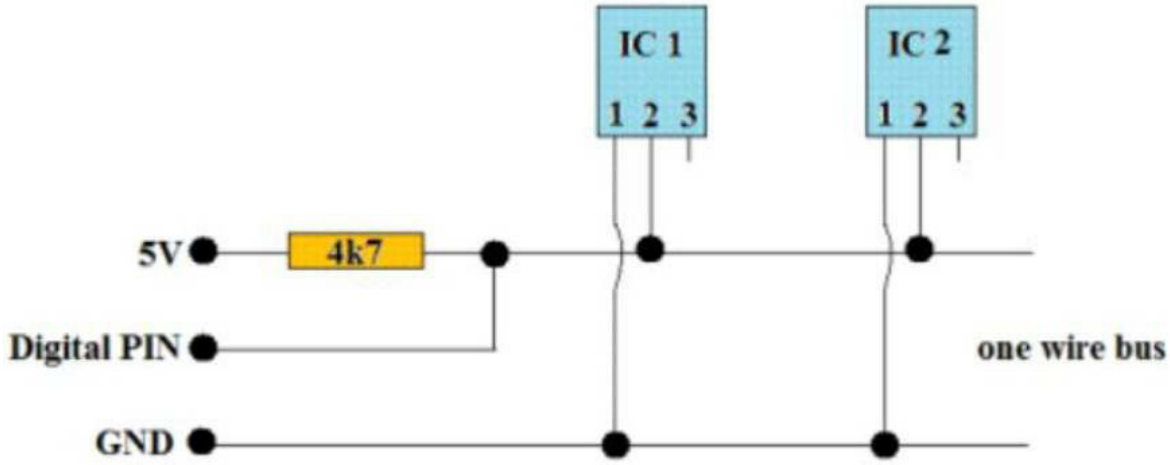
Фа
349
До
523

Самостоятельная работа #2: Дополнить таймер из предыдущей главы возможностью вывода звука при окончании интервала. Для этого заменить функцию delay функцией воспроизведения соответствующего тона.

2.7 Подключаем датчик температуры DS1820

Мы уже упоминали термистор - резистор, сопротивление которого зависит от температуры, однако его точность весьма невелика. Большую точность можно получить с помощью цифрового датчика DS1820 - данные от него передаются в цифровой форме. Это не только точнее, но и удобнее - не нужно пересчитывать данные с помощью коэффициентов или таблиц, мы сразу имеем готовую величину, которую можно использовать в коде программы.

В отличие от рассмотренных выше аналоговых устройств, DS1820 “общается” с контроллером в цифровой форме, посылая данные в уже готовом, двоичном формате. Для этого используется специальный формат передачи, названный 1Wire, таким способом можно даже подключить несколько устройств к одному проводу.



Сам протокол связи достаточно сложный, но к счастью для нас, его поддержка уже добавлена в библиотеки для Arduino, почти ничего для этого делать не нужно.

Для установки библиотеки достаточно скачать библиотеку с сайта https://github.com/milesburton/Arduino-Temperature-Control-Library и установить ее в папку “Мои документы\Arduino\libraries” (для этого достаточно создать новую папку по этому адресу и скопировать файлы туда).

Сам датчик DS1820 имеет весьма много функций, например возможность установки верхнего и нижнего порога срабатываний. Простейший код подключения датчика для Arduino с выводом информации в serial port, выглядит так:

```
#include <OneWire.h>

#include <DallasTemperature.h>


// Номер порта для подключения датчика

int portPin = 2;


OneWire oneWire(portPin);
```

```
DallasTemperature sensors(&oneWire);

void setup(void)

{

// Открытие порта

Serial.begin(9600);

Serial.println("DS1820");

// Запуск датчика

sensors.begin();

}

void loop(void)

{
// Запрос температуры

sensors.requestTemperatures();

float tempInC = sensors.getTempCByIndex(0);

// Вывод в порт (опционально)

Serial.print("T = ");
Serial.println(tempInC, 2);
Serial.println();

delay(5000);

}
```

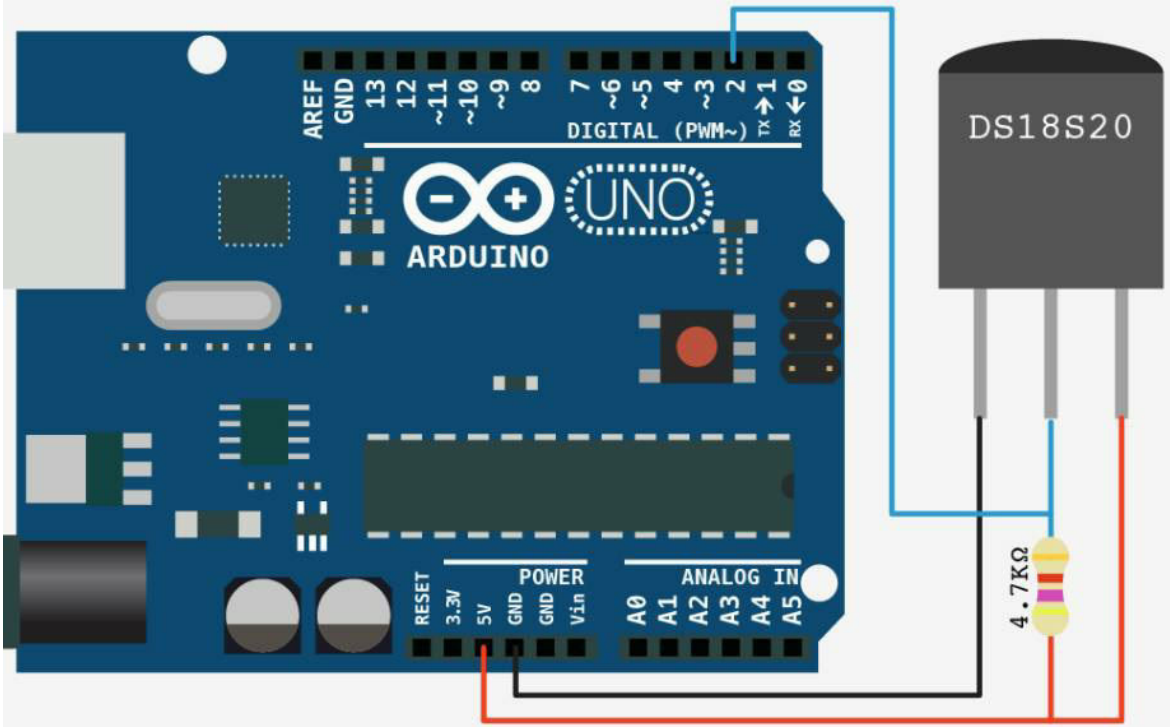
Как можно видеть, мы сначала запрашиваем данные с помощью функции requestTemperatures, затем читаем полученные данные с помощью getTempCByIndex(0). Цифра 0 здесь, это номер датчика, как было сказано выше, их может быть несколько. Вывод в порт используется лишь для удобства просмотра результатов.

Сам DS1820 имеет небольшой размер, и по форме напоминает транзистор.

Впрочем, они также продаются и в виде выносных датчиков в водонепроницаемом корпусе, что позволяет использовать DS1820 для измерения температуры в удаленных местах.

Такой датчик можно использовать, например, для измерения температуры за окном. Диапазон измеряемых температур составляет от -55 до 125°C, что будет достаточно даже в случае глобального потепления или похолодания.

Подключение датчика к Arduino весьма просто:



После подключения, достаточно загрузить вышеприведенную программу в Arduino, открыть serial monitor в Arduino IDE, в появившемся окне можно будет наблюдать значения температуры.

Самостоятельная работа #1: оставить компьютер с подключенным датчиком на сутки. Построить график температуры, например с помощью Excel или https://plot.ly/create/.

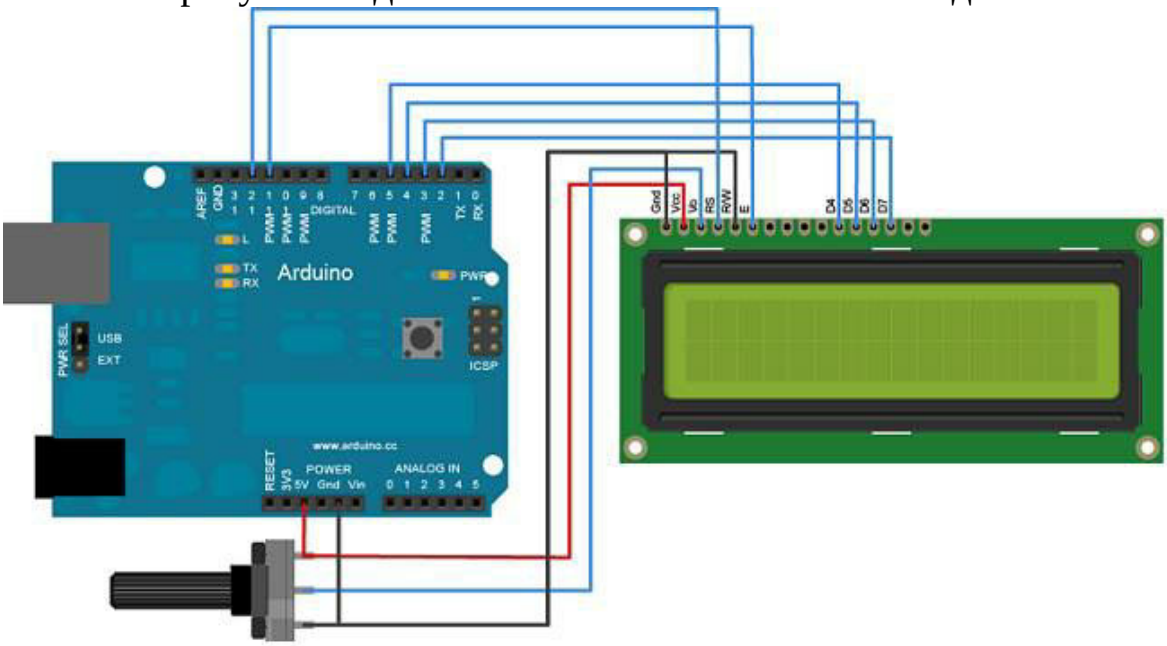
Самостоятельная работа #2: добавить в код зажигание светодиода, если температура становится выше определенного предела. Вместо светодиода можно также подключить пьезодинамик, как описано в предыдущей главе. Данная система может быть основой для автоматического контроля температуры, например в теплице.

2.8 Подключаем OLED-экран

Мы уже умеем обрабатывать нажатия кнопок, подключать разнообразные датчики и выводить информацию в компьютер через serial port. Осталось подключить отдельный экран, чтобы получить полностью автономно работающее устройство.

Обычный ЖК-экран может работать с Arduino, но для его

подключения требуется задействовать слишком много выводов:

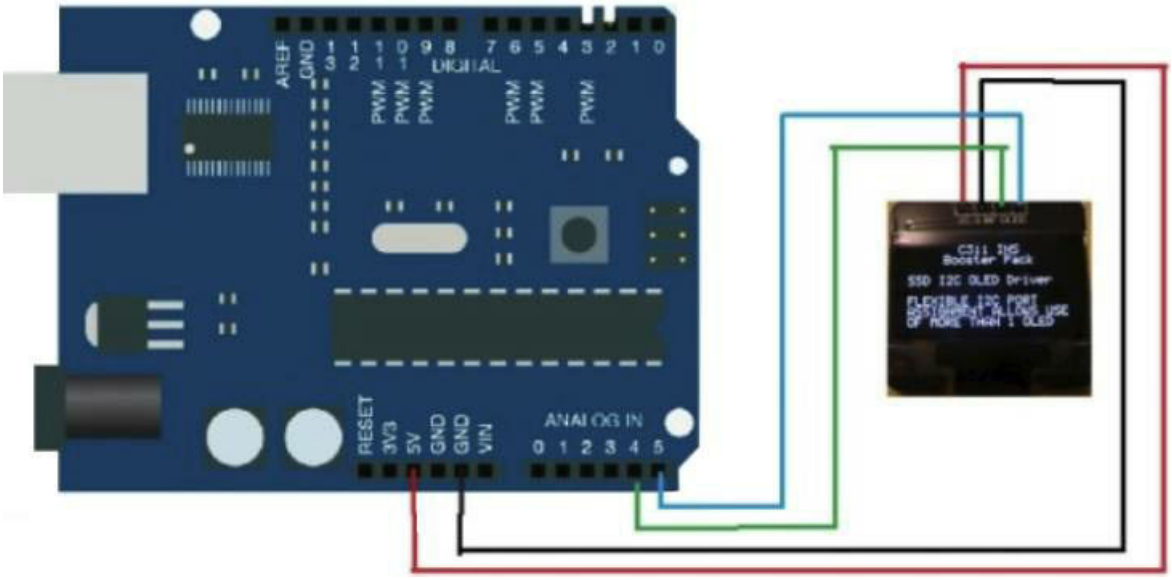


К счастью для нас, сейчас есть более удобный и современный путь - экраны, подключаемые по шине **I2C**. Шина I2C позволяет подключать различные устройства всего лишь с помощью двух проводов.

Мы будем использовать OLED-экран, имеющий 4 вывода для подключения - он показан на картинке **слева**. В продаже бывают и другие дисплеи, как на картинке справа - такой вариант не подойдет, при покупке важно не перепутать.

Само подключение при этом достаточно просто, для работы шины I2C требуется всего 4 провода - 2 линии данных, “земля” и “питание”. Также как и с 1Wire, на одной шине может быть подключено несколько разных устройств.

Само подключение показано на картинке:

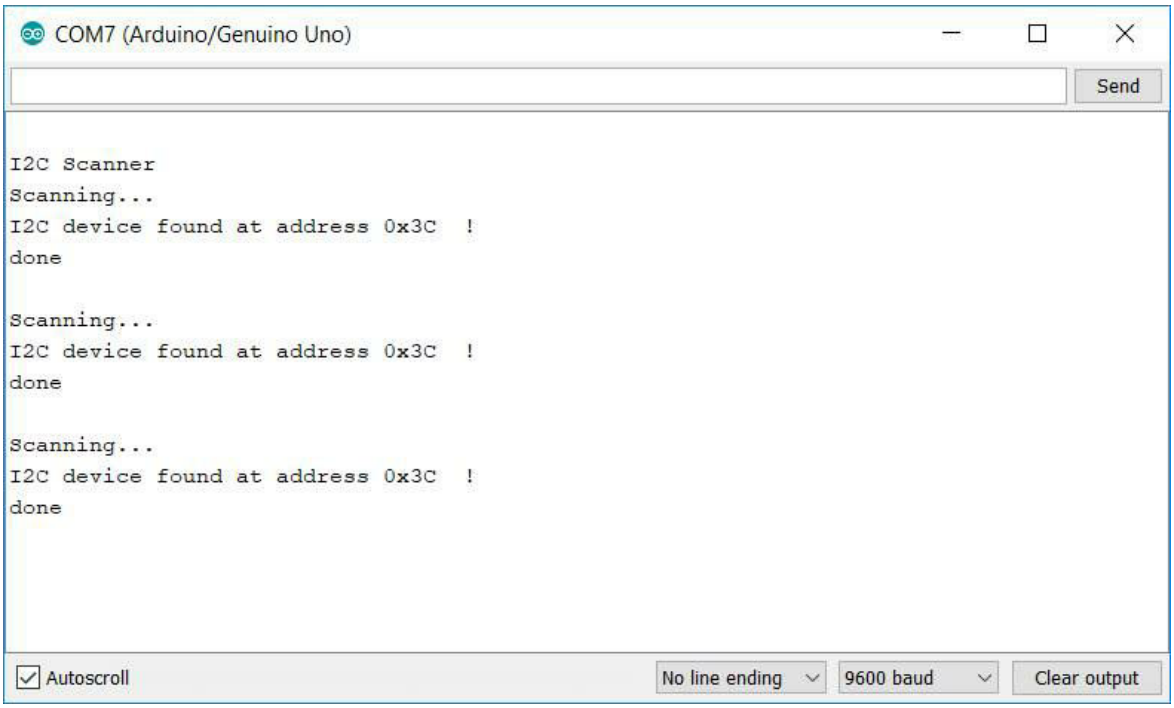


На шине I2C может быть несколько устройств, поэтому чтобы передавать данные, мы должны знать **адрес устройства**. Удобнее всего для этого использовать программу i2c_scan, код которой показан ниже.

```
#include <Wire.h>

void setup()
{
Wire.begin();

Serial.begin(9600);
while (!Serial);
Serial.println("\nI2C Scanner");
}

void loop()
{
byte error, address;
Serial.println("Scanning...");

int nDevices = 0;
for(address = 1; address < 127; address++) {
Wire.beginTransmission(address);
```

```
error = Wire.endTransmission();

if (error == 0) {
Serial.print("I2C device found at address 0x");
if (address<16)
Serial.print("0");
Serial.print(address,HEX);
Serial.println(" !");

nDevices++;
}
else if (error==4) {
Serial.print("Unknow error at address 0x");
if (address<16)
Serial.print("0");
Serial.println(address,HEX);
}
}
if (nDevices == 0)
Serial.println("No I2C devices found\n");
else
Serial.println("done\n");
delay(5000); // wait 5 seconds for next scan
}
```

Если дисплей подключен правильно, то запустив Serial Monitor, мы увидим примерно такой текст:



0x3C - это и есть адрес нашего дисплея. Адрес также можно посмотреть на обратной стороне дисплея - но в моем случае там было написано 0x78. С неправильным адресом, разумеется, ничего не работало. Так что лучше лишний раз проверить.

Непосредственная работа с дисплеем состоит в посылке различных команд по шине I2C, но к счастью, нам этого делать не нужно - уже написаны готовые библиотеки. Их можно скачать по ссылкам https://github.com/adafruit/Adafruit-GFX-Library и https://github.com/adafruit/Adafruit_SSD1306 соответственно. Файлы необходимо скачать и распаковать в папку Документы\Arduino\libraries. В моем случае, я распаковал файлы в папки Adafruit_SSD1306 и Adafruit-GFX-Library.

Следующим шагом необходимо указать тип используемого в проекте дисплея. Для этого достаточно в файле Adafruit_SSD1306-master\Adafruit_SSD1306.h раскомментировать соответствующую строку. Всего доступны 3 варианта, например для дисплея 128x64 код будет выглядеть так:

```
#define SSD1306_128_64
```

```
// #define SSD1306_128_32
// #define SSD1306_96_16
```

Теперь все готово, и можно выводить информацию на дисплей. Перезапускаем Arduino IDE, чтобы загрузились новые библиотеки. В качестве примера рассмотрим несложный код.

```
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(0);

static const unsigned char PROGMEM logo16_glcd_bmp[] = {
B00000000, B11000000,
B00000001, B11000000,
B00000001, B11000000,
B00000011, B11100000,
B11110011, B11100000,
B11111110, B11111000,
B01111110, B11111111,
B00110011, B10011111,
B00011111, B11111100,
B00001101, B01110000,
B00011011, B10100000,
B00111111, B11100000,
B00111111, B11110000,
B01111100, B11110000,
B01110000, B01110000,
B00000000, B00110000
};

void setup() {
display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.println("Hello, world!");
display.drawBitmap(2, 14, logo16_glcd_bmp, 16, 16, 1);
display.drawLine(0, 40, 128, 40, WHITE);
```

```
display.drawLine(10, 50, 118, 50, WHITE);
display.display();
}

void loop() {

}
```

Разберем код подробнее.

Переменная display хранит объект, содержащий все методы для работы с дисплеем. Далее объявляется битовый массив PROGMEM logo16_glcd_bmp, который, как нетрудно догадаться, хранит непосредственно изображение - один бит соответствует одному пикселу. В функции setup происходит инициализация дисплея, там же указывается адрес 0xC3, который мы нашли ранее. Затем вызываются функции clearDisplay, setTextSize, println, назначение которых понятно из названия. При вызове всех этих функций данные заносятся в промежуточный блок памяти. И лишь при вызове метода display() эти данные реально переносятся на экран. Такая технология называется “двойной буфер”, она позволяет избежать мерцания при обновлении экрана.

Результат - загружаем программу в Arduino и видим запрограммированную нами картинку.

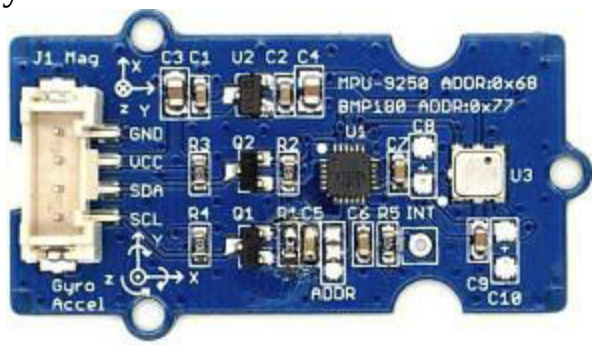
Самостоятельная работа: Вывести на экран переменную в различных форматах. Для этого воспользоваться функцией println, аналог которой для последовательного порта выглядит так:

```
int value = 24;
Serial.println(value);

Serial.println(value, DEC);

Serial.println(value, HEX);

Serial.println(value, OCT);

Serial.println(value, BIN);
```

Дополнительно можно вывести значения с датчика температуры, который мы рассматривали в предыдущей главе.

2.9 Подключаем гироскоп, компас и акселерометр

С помощью шины I2C можно подключать различные устройства, например многочисленные датчики. Для примера можно рассмотреть плату “Grove - IMU 10DOF”.



Плата работает по той же шине I2C и подключается точно так же, как и дисплей из предыдущей главы, 4 проводами. На плате находятся датчик **MPU-9250**, содержащий гироскоп, акселерометр и компас, и цифровой барометр **BMP280**.

Подключение платы точно такое же, как на картинке с дисплеем из предыдущей главы. Сам обмен данных с датчиками достаточно сложен, но готовые библиотеки для Arduino уже существуют, и их весьма просто использовать.

Чтобы получить данные с датчика **MPU-9250**, нужно скачать библиотеку с сайта https://github.com/Snowda/MPU9250 (выбрать Download - zip) и распаковать ее в папку Документы\Arduino\libraries.

Сам код чтения данных с датчика и их вывода в последовательный порт весьма прост.

```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU9250.h"
```

```
MPU9250 accelgyro;
int index = 0;

void setup() {
// Запуск шины I2C
Wire.begin();

// Инициализация порта
Serial.begin(115200);
// Инициализация датчика
accelgyro.initialize();

// Проверка подключения
Serial.println("Testing device connections...");
Serial.println(accelgyro.testConnection() ? "MPU9250 connected" : "MPU9250 failed");
}

void loop() {
// Чтение данных
int16_t ax, ay, az, gx, gy, gz, mx, my, mz;
accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);

// Получение ускорения или вращения (опционально)
//accelgyro.getAcceleration(&ax, &ay, &az);
//accelgyro.getRotation(&gx, &gy, &gz);

// Вывод в порт
Serial.print(index); Serial.print("\t");
Serial.print(ax); Serial.print("\t");
Serial.print(ay); Serial.print("\t");
Serial.print(az); Serial.print("\t");
Serial.print(gx); Serial.print("\t");
Serial.print(gy); Serial.print("\t");
Serial.print(gz); Serial.print("\t");
Serial.print(mx); Serial.print("\t");
Serial.print(my); Serial.print("\t");
```

```
Serial.println(mz);

index++;
}
```

Как можно видеть, все просто, и для получения данных достаточно одной строчки кода getMotion9. Остальной код имеет вспомогательное значение, и служит для передачи данных в serial port. Разумеется, вместо него можно использовать что-то другое, например включать или выключать светодиод, если данные превосходят некую заданную величину.

Переменная index используется для вывода результатов с увеличением счетчика. Она создана в виде глобальной переменной, т.к. функция loop каждый раз вызывается заново. Стоит заметить, что при работе программы в течении долгого времени, переменная index может переполниться, для избежания таких случаев стоит использовать тип данных с большей разрядностью.

Запустив программу, мы получим в Serial Monitor данные типа таких:

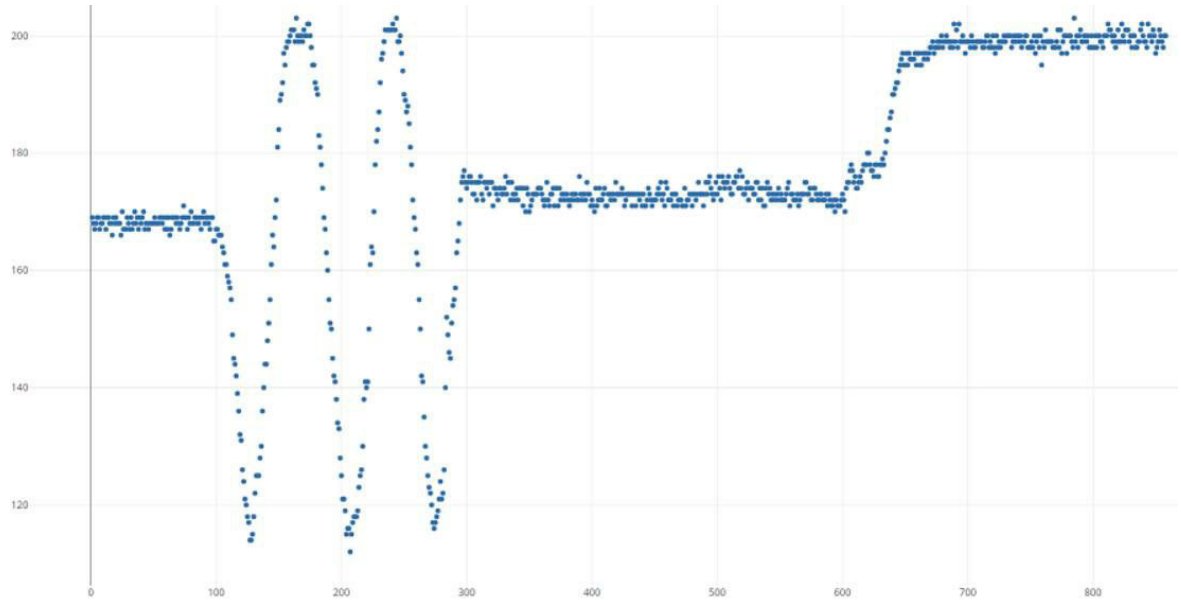
```
0,    225,153,15401,    22,44,15,    540,302,192
1,    223,175,15434,    7,41,15,    540,302,192
2,    226,161,15417,    3,46,16,    540,302,192
3,    233,166,15411,    7,51,3,    540,302,192
4,    233,166,15411,    13,44,2,    540,302,192
5,    223,161,15435,    13,44,2,    540,302,192
```

Каждый датчик - 3х-осевой, соответственно мы имеем 3 колонки цифр с каждого из сенсоров (акселерометр, гироскоп и компас). Примерно такие же датчики стоят и в смартфонах, что используется например в играх, для управления наклонами устройства.

Данные также можно открыть в любой программе построения графиков, например онлайн на https://plot.ly/create/, и наглядно посмотреть как изменяются значения при вращении или повороте датчика. К примеру, на картинке показан график с магнитометра при

поднесении к датчику металлического предмета.

Сам график построен с помощью бесплатного сервиса https://plot.ly/create/.



Чтение данных с барометра **BMP280** аналогично. Нужно скачать библиотеки с сайта https://github.com/adafruit/Adafruit_BMP280_Library и поместить их в папку Документы\Arduino\libraries.

Код аналогичен приведенному выше.

```
#include <Wire.h>

#include <SPI.h>

#include <Adafruit_Sensor.h>

#include <Adafruit_BMP280.h>


Adafruit_BMP280 bmp;


void setup() {
```

```
Serial.begin(115200);

Serial.println(F("BMP280 test"));

if (!bmp.begin()) {

Serial.println(F("Could not find a valid BMP280 sensor"));

while (1);

}

}

void loop() {
// Чтение температуры

Serial.print(F("T = "));

Serial.print(bmp.readTemperature());

Serial.println(" *C");

// Атмосферное давление

Serial.print(F("Pressure = "));

Serial.print(bmp.readPressure());

Serial.println(" Pa");

// Барометрическая высота относительно “нулевой” отметки

Serial.print(F("Approx altitude = "));

Serial.print(bmp.readAltitude(1013.25)); // Давление на “нулевой”
```

отметке

```
Serial.println(" m");


Serial.println();

delay(2000);

}
```

Разумеется, атмосферное давление не меняется столь же быстро, как показания акселерометра. Оставив программу работать некоторое время, можно получить график изменения атмосферного давления. Барометрическая высота кстати, активно используется в авиации, т.к. позволяет получать довольно-таки точную высоту над уровнем моря относительно нулевой отметки. Точность цифровых датчиков весьма высока, и позволяет улавливать даже разницу в высоте менее 1м. Это используется например, в квадрокоптерах для удержания заданной высоты полета.

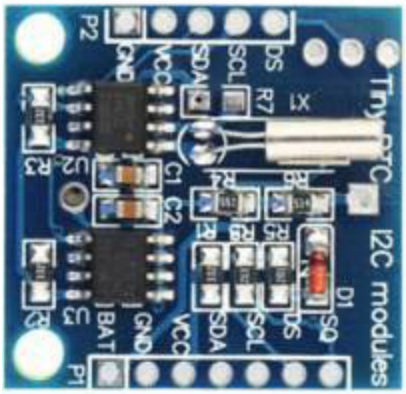
Самостоятельная работа #1: Вывести на графике изменение атмосферного давления за ночь. Т.к. давление изменяется медленно, паузу в коде можно увеличить, это уменьшит требуемое количество сохраняемых данных.

Самостоятельная работа #2: Создать устройство “тревожной сигнализации”, которое будет мигать светодиодом и включать звук, если кто-то сдвинул прибор с места. Для этого при включении устройства стоит запоминать текущее значение акселерометра или компаса, и если значения вышли за заданные пределы, значит устройство было сдвинуто. Также можно предусмотреть отдельную кнопку “сброс” для запоминания новых значений. Разместив такой прибор в компактном корпусе, его можно использовать например, для охраны чемодана.

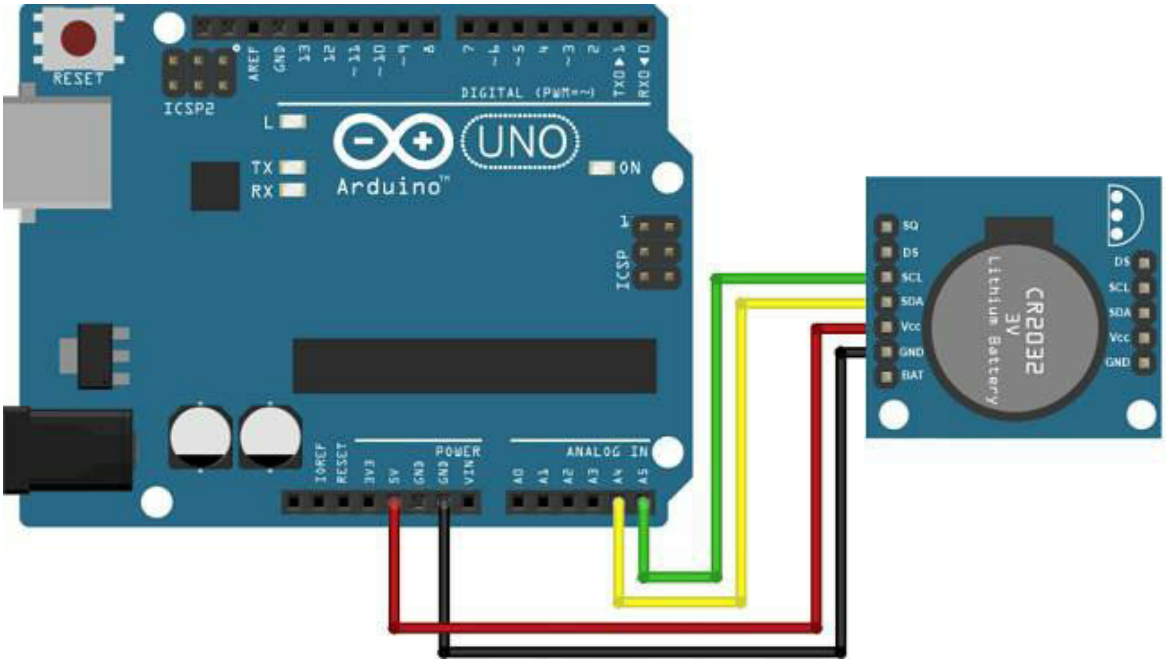
2.10 Подключаем часы реального времени (RTC)

Мы уже знаем, как подключить к Arduino внешнюю нагрузку через

полевой транзистор или реле, как запрограммировать паузы и считывать температуру с внешнего сенсора. Допустим, мы решили сделать полив цветов по расписанию - утром и вечером. Но как Arduino узнает, какое сейчас время суток? Для этого может использоваться специальная микросхема - модуль часов реального времени (Real Time Clock). Данный модуль передает данные по I2C, время хранится в специальной микросхеме DS1307, на плате также есть батарейка, обеспечивающая работу модуля когда питание отключено.



Подключение модуля к Arduino весьма просто.



Для использования модуля, необходимо скачать и установить библиотеку https://github.com/Makuna/Rtc.

Пример использования DS1707 показан ниже.

```
#include <Wire.h>
#include <RtcDS1307.h>

RtcDS1307<TwoWire> Rtc(Wire);

void setup ()
{
Serial.begin(57600);

Rtc.Begin();

// Если время не было установлено, установить его
if (!Rtc.IsDateTimeValid()) {
RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
Rtc.SetDateTime(compiled);
}
// Запустить отсчет времени, если не запущен
if (!Rtc.GetIsRunning()) {
Rtc.SetIsRunning(true);
}

// Выход square wave не используется
Rtc.SetSquareWavePin(DS1307SquareWaveOut_Low);
}

void loop ()
{
if (!Rtc.IsDateTimeValid()) {
// Ошибка, возможно, пропадание питания при отсутствии батареи
Serial.println("RTC Error!");
return;
}

RtcDateTime now = Rtc.GetDateTime();
Serial.println(now.Year());
```

```
Serial.println(now.Month());
Serial.println(now.Day());
Serial.println(now.Hour());
Serial.println(now.Minute());
Serial.println(now.Second());
Serial.println();

// 10с пауза
delay(10000);
}
```

Код довольно-таки прост. Если часы DS1307 не установлены (функция IsDateTimeValid возвращает FALSE), то они устанавливаются с помощью констант __DATE__ и __TIME__ - они содержат время компиляции программы. Таким образом, при первом запуске в таймер будет автоматически занесено текущее время. Затем с помощью функции GetDateTime мы получаем время и дату, из которой можно узнать год, месяц, день, часы, минуты и секунды.

Теперь, загрузив программу в Arduino, мы можем отключить устройство. При последующем включении программа всегда будет “знать” текущее время.

Самостоятельная работа #1: сделать “вечернее освещение” с помощью Arduino и таймера, например, настроить зажигание светодиода с 23 вечера до 7 часов утра.

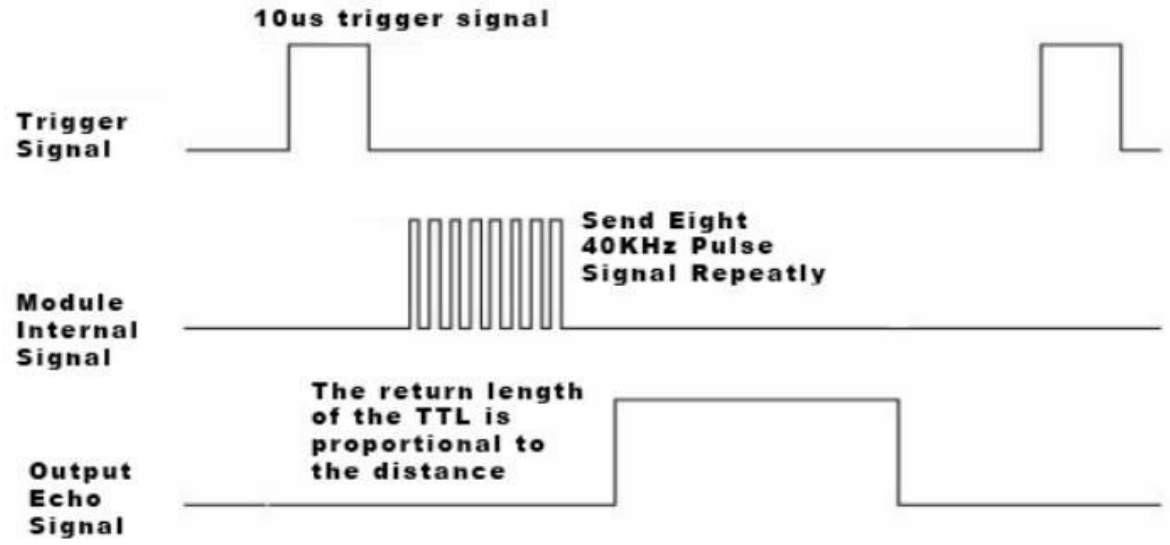
Самостоятельная работа #2: сделать “настольный будильник”, подключив к Arduino модуль RTC, “пищалку” и кнопку. Задать в коде срабатывание будильника в определенный час и минуту, кнопку использовать для остановки звучания.

2.11 Подключаем ультразвуковой дальномер HC-SR04

Еще одним популярным в любительской электронике прибором, является дальномер HC-SR04. Это недорогой прибор, стоимостью 2-4$ на eBay, позволяющий измерять расстояние до препятствий с помощью ультразвука.

Дальность обнаружения от нескольких сантиметров до 4м, позволяет использовать его в различных устройствах, например в самодельных роботах.

Принцип использования HC-SR04 показан на рисунке.



Когда активируется запускающий сигнал (trigger signal), устройство посылает серию ультразвуковых импульсов, и в завершении формирует ответный сигнал, длительность которого пропорциональна расстоянию. Таким образом, для подключения достаточно всего двух проводов, для выходного и для входного импульсов.

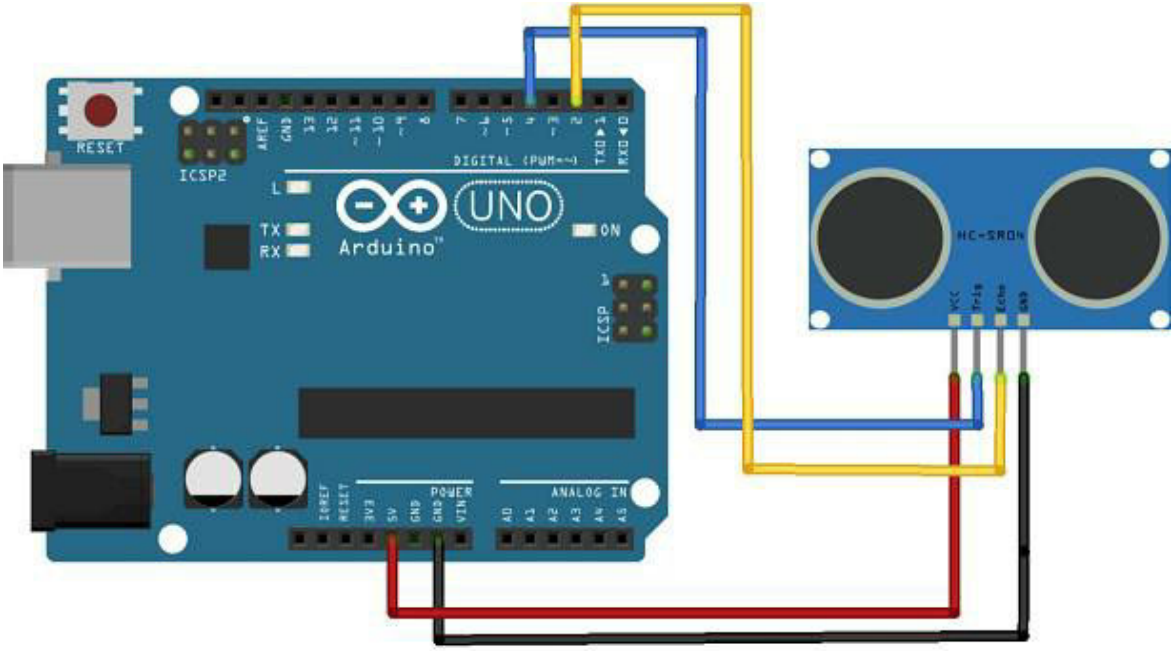
Использование датчика весьма просто - для Arduino уже написаны готовые библиотеки, позволяющие получить расстояние. Вначале нужно скачать библиотеку по адресу https://github.com/JRodrigoTech/Ultrasonic-HC-SR04, ее необходимо распаковать в уже знакомую нам папку Документы\Arduino\libraries.

Сам код, считывающий показания сенсора и отправляющий их в serial port, весьма прост.

```
#include <Ultrasonic.h>

Ultrasonic ultrasonic(4,2); // (Trig,Echo)

void setup() {
Serial.begin(9600);
}

void loop()
{
Serial.print(ultrasonic.Ranging(CM));
Serial.println(" cm" );
delay(100);
}
```

Как можно видеть, создается объект Ultrasonic, в качестве параметров которого указываются номера выводов. Дальше функция ultrasonic.Ranging(CM) делает всю работу.

Само подключение сенсора тоже не вызывает сложностей.



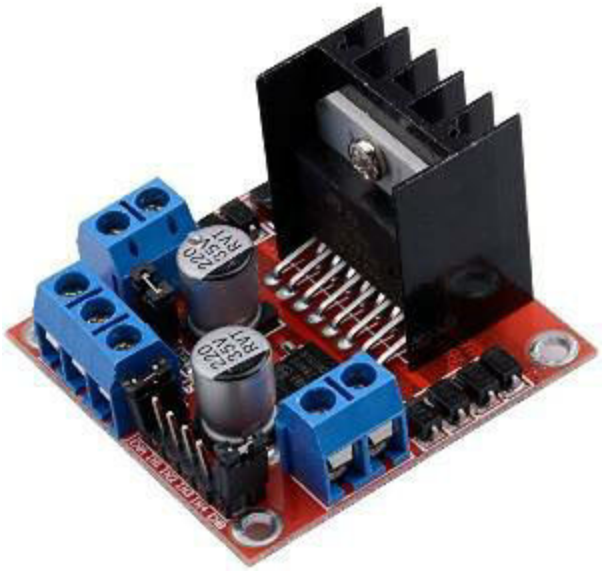
Самостоятельная работа: подключить к данной схеме LCD-дисплей и сделать вывод на экран расстояния до объектов. Опционально, можно подключить светодиод и зажигать его, если расстояние до объекта меньше критического.

2.12 Подключаем плату управления моторами

Мы уже можем подключить к Arduino практически все необходимое - датчики, светодиоды, индикаторы. Остался последний шаг, после которого можно собрать вполне полноценного робота - это научиться управлять моторами.

Как уже говорилось в предыдущей главе, любой микроконтроллер не может управлять мотором напрямую, у вывода не хватит мощности. Существуют специальные микросхемы, называемые “драйверами”, которые и выполняют эту работу. Разумеется, мотор можно подключить и через транзистор, но полноценный драйвер имеет больше возможностей, например возможность смены направления вращения мотора.

Для примера рассмотрим драйвер на микросхеме L298N, его можно купить в виде готовой платы ценой 2-5$.



Плата имеет вполне неплохие для своей цены возможности. Левые и правые разъемы используются для подключения моторов. Плата также имеет стабилизатор напряжения, что позволяет использовать для питания моторов 12В, а выход 5В использовать для питания Arduino.

Описание комбинаций управляющих импульсов приведено в документации на микросхему (C и D - входы каждого канала).

Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Соответственно, линейка из 6 выводов имеет 2 переключателя ENA ENB (Enable A, B) для активации левого и правого моторов, 4 вывода IN1, IN2, IN3, IN4 используются для подачи управляющих импульсов.

Пример кода управления моторами показан ниже. Здесь входы EN1, EN2 используются для управления скоростью моторов уже рассмотренным ранее методом широтно-импульсной модуляции.

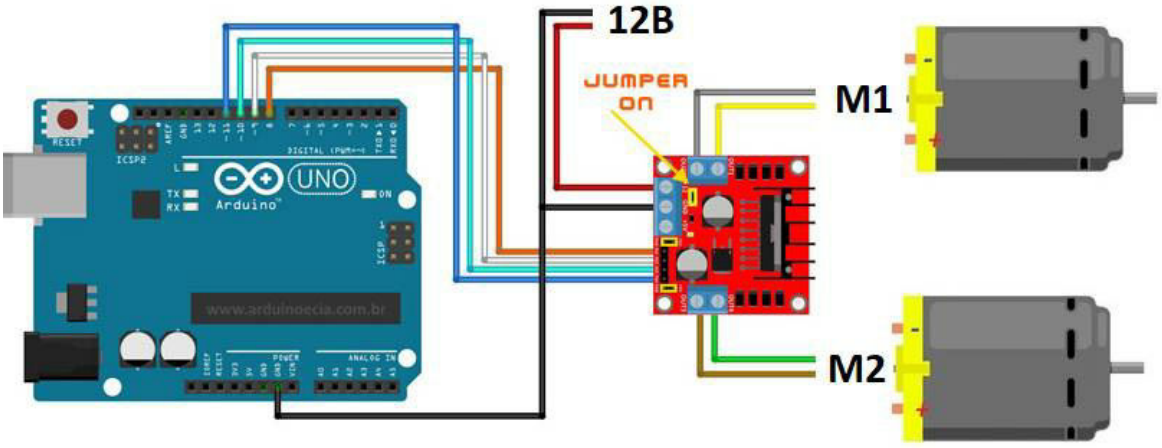
```
// Моторы M1, M2
```

```
int enA = 10, in1 = 9, in2 = 8;
int enB = 5, in3 = 7, in4 = 6;
void setup() {
pinMode(enA, OUTPUT);
pinMode(enB, OUTPUT);
pinMode(in1, OUTPUT);
pinMode(in2, OUTPUT);
pinMode(in3, OUTPUT);
pinMode(in4, OUTPUT);
}
void runMotors() {
// Запустить мотор A
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
// Установить скорость 200 (диапазон 0~255)
analogWrite(enA, 200);
// Запустить мотор B
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
// Установить скорость 200 (диапазон 0~255)
```

```
analogWrite(enB, 200);
delay(2000);
// Изменить направление
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
delay(2000);
// Остановить моторы
digitalWrite(in1, LOW);
digitalWrite(in2, LOW);
digitalWrite(in3, LOW);
digitalWrite(in4, LOW);
}
void loop() {
runMotors();
delay(5000);
}
```

Другой вариант схемы подключения показан на картинке ниже, здесь входы EN1/EN2 программно не управляются, они просто замкнуты переключателями, идущими в комплекте с платой. Это не позволяет управлять скоростью моторов, зато делает подключение

более простым.



Таким образом, с помощью одного или двух драйверов можно управлять двумя или четырьмя моторами.

Желающие заняться робототехникой более серьезно, могут также приобрести специальную платформу с колесами и моторами. Оснастить ее электроникой и датчиками можно по своему вкусу.

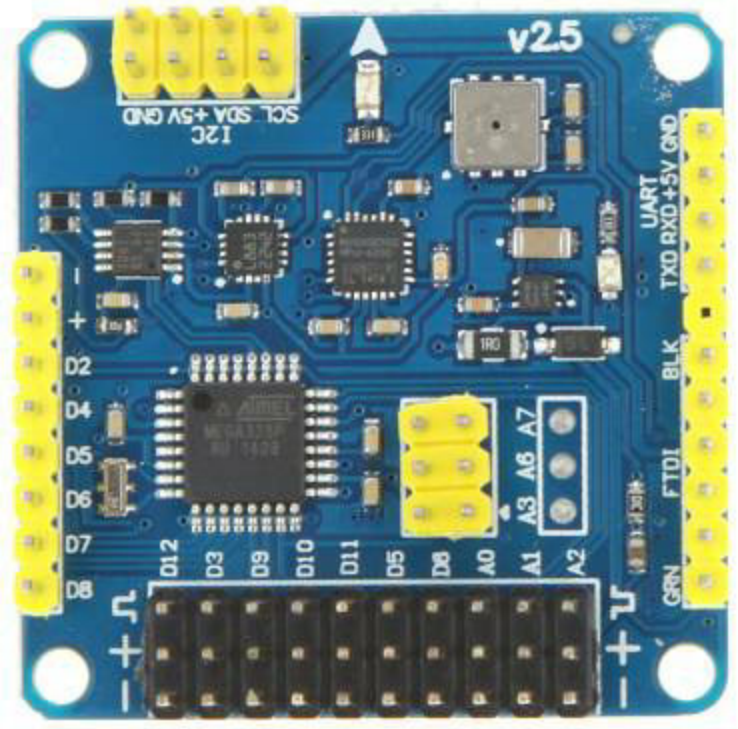
Стоимость такой платформы составляет от 20$ до 100$ в зависимости от размера, мощности моторов и качества изготовления.

2.13 Multiwii - делаем квадрокоптер

Платы Arduino имеют весьма неплохие возможности и вычислительную мощность, что позволяет не только мигать светодиодом, но даже сделать вполне функциональный дрон, способный летать в разных режимах, делать фотосъемку местности, удерживать высоту полета и пр. Для этого используется популярный проект с открытым исходным кодом Multiwii. Его название произошло от давно популярного контроллера Wii Nunchuck, имеющего внутри гироскоп и акселерометр. Соединив его с Arduino и платой управления моторами, можно было получить вполне летающий квадрокоптер. Позже появились уже готовые платы со всеми датчиками на борту.

Чтобы собрать квадрокоптер на базе Multiwii, потребуется:

- Специальная плата Multiwii, имеющая “на борту” процессор и набор датчиков.



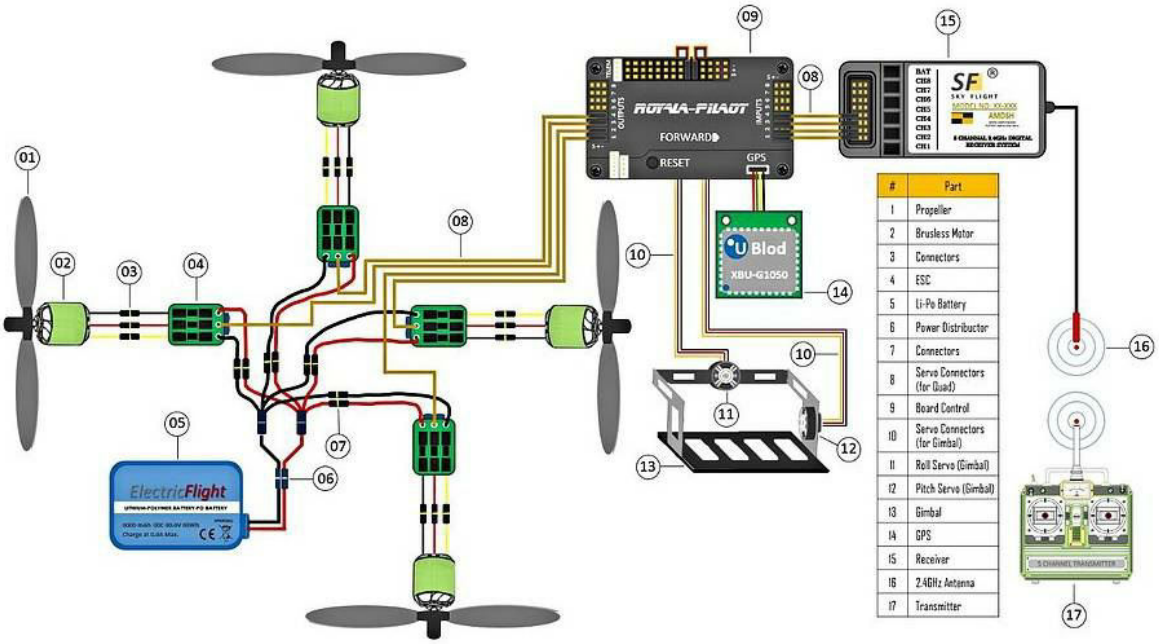
- Пульт управления для радиоуправляемых моделей, приемник которого подключается к каналам D2 - D8.

- 4 бесколлекторных мотора и 4 регулятора скорости, выводы которых подключаются к площадкам D12, D3, D9, D10.

- 4 пропеллера разного направления вращения.

- Литий-полимерный аккумулятор для авиамоделей (они способны отдавать большие токи, в отличие например, от телефонных)

Примерная схема всех возможных устройств, подключенных к квадрокоптеру, показана на рисунке.



#	Part
1	Propeller
2	Brusless Motor
3	Connectors
4	ESC
5	Li-Po Battery
6	Power Distributor
7	Connectors
8	Servo Connectors (for Quad)
9	Board Control
10	Servo Connectors (for Gimbal)
11	Roll Servo (Gimbal)
12	Pitch Servo (Gimbal)
13	Gimbal
14	GPS
15	Receiver
16	2.4GHz Antenna
17	Transmitter

Здесь используются не только плата управления, но и GPS и подвес для фотокамеры, в простейшем случае без них можно обойтись.

Готовый самодельный квадрокоптер может выглядеть примерно так:

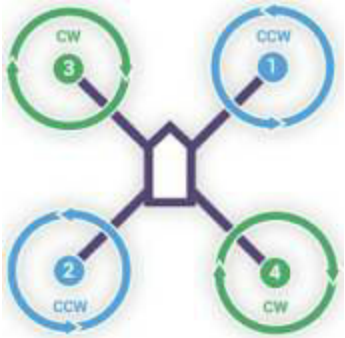
Все детали (раму, пропеллеры, моторы, контроллер) можно приобрести на специализированных сайтах RC-моделей, например на www.hobbyking.com.

Когда квадрокоптер собран, плату необходимо подключить к компьютеру. С помощью Arduino IDE в нее необходимо загрузить программу управления, скачать которую можно по адресу https://github.com/multiwii/multiwii-firmware.

Важно: все изменения и загрузку кода следует делать со снятыми винтами, во избежание травм.

Когда программа загружена в контроллер, необходимо запустить на компьютере программу управления, которая выглядит примерно так:

В ней можно проверить и настроить реакцию квадрокоптера на наклоны и ручки управления. Лишь когда все работает, можно сделать пробный запуск, сначала без пропеллеров, и лишь убедившись что моторы реагируют нормально, можно делать пробный полет. Кстати, в правом нижнем углу программы показана схема подключения моторов квадрокоптера. Их важно не перепутать, в противном случае коптер не сможет лететь в нужном направлении. Также важно не перепутать направление вращения винтов:



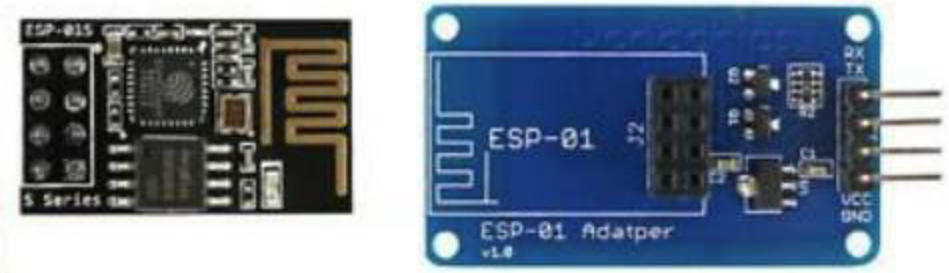
Настройка и постройка квадрокоптера выходит за рамки этой книги, желающим рекомендуется почитать статьи в Интернете или обратиться на специальные форумы, например на http://forum.rcdesign.ru.

2.14 Выходим в интернет: Serial to WiFi

Как можно видеть, возможности плат Arduino весьма неплохи. Но основной их недостаток, который в настоящее время становится все

более критичным - невозможность выхода в Интернет. “Интернет вещей” заполняет нашу жизнь, и хочется к примеру, не только сделать робота, но и управлять им со смартфона, не только сделать метеостанцию, но и разместить график температуры онлайн, и т.к.

Разумеется, пожеланий пользователей не могли остаться без внимания, и в продаже появились платы Serial to wifi - несложные WiFi-модули, с помощью которых можно принимать или отправлять данные через последовательный порт.



Подключение платы к Arduino весьма несложно, достаточно 4х проводов. Для использования платы есть готовая библиотека ESP8266WiFi, найти которую можно на сайте https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi.

Пример кода для подключения к домашней сети WiFi показан ниже.

```
#include <ESP8266WiFi.h>


const char* ssid = "MYFI_HOME";

const char* password = "12345678;


void setup() {

Serial.begin(115200);

delay(10);
```

```
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void loop() {
...
}
```

Как можно видеть, код вполне прост. Более сложные методы взаимодействия с сетью, чтение и отправка запросов на сервер, выходят за рамки данной книги, желающие могут изучить это самостоятельно.

Интересно заметить, что появление модулей ESP, цена которых составляет всего лишь 1-5$ произвели настоящую “революцию” в сфере любительской электроники. Теперь практически любой модуль получил

возможность не только работать автономно, но и принимать или отправлять данные в сеть. Датчики температуры, реле, термостаты, часы, сигнализации - список устройств, для которых это актуально, весьма велик.

На этом мы закончим главу про Arduino и в третьей части рассмотрим модули ESP более подробно. Четвертая часть будет посвящена Linux и Raspberry Pi.

Часть 3. Выходим в Web: платы ESP32

3.1 Общее знакомство

Платы Arduino в свое время стали настоящей находкой для радиолюбителей - просто и без особых сложностей можно использовать различные компоненты, датчики, писать логику программы. Но увы, во-первых, 8-битный процессор, работающий на частоте 16МГц, не позволяет запускать более-менее сложные алгоритмы. Во-вторых, современный мир - это мир Интернета. Практически для любого устройства возможность обмена данными становится критически важной.

Итак, настала пора предоставить героя нашей следующей главы - плату ESP32.

Всего за 6$ мы получаем 32-разрядный процессор, работающий на частоте 240МГц, 520Кб памяти, 4Мб флеш-памяти и поддержку Bluetooth и WiFi.

Писать программы для ESP32 можно также с помощью Arduino IDE, необходимо лишь доустановить компилятор для этого типа процессоров.

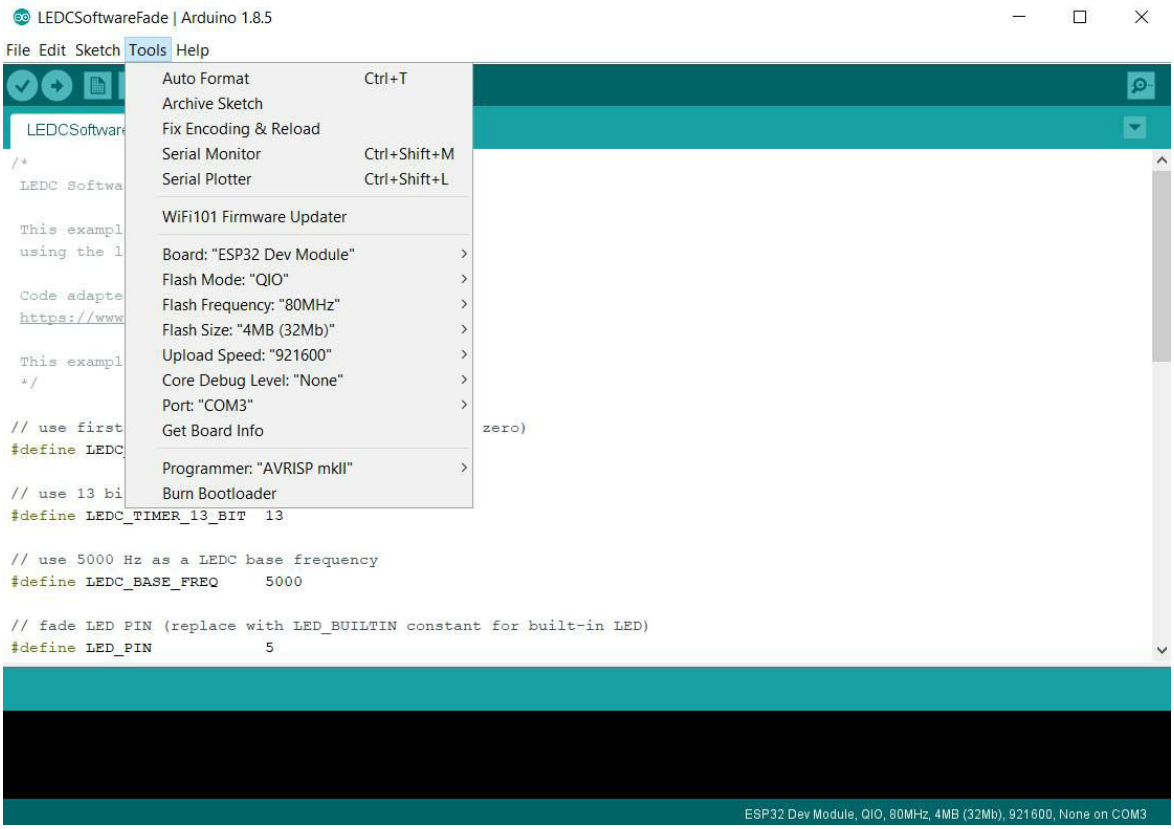
Для этого нужно:

- Скачать по адресу https://git-scm.com/download/win и установить программу контроля версий Git.
- В папке Документы\Arduino создать папку hardware\espressif\.
- Скачать библиотеки и компилятор для ESP32. Для этого нужно открыть консоль (Win+R - cmd), перейти в папку (cd C:\Users\Имя пользователя\Documents\Arduino\hardware\espressif\) и выполнить команду

git clone --recurse-submodules -j8 https://github.com/espressif/arduino-esp32.git (вместо консоли можно также воспользоваться Git GUI). Библиотеки будут устанавливаться довольно долго, около 10 минут.

- По окончании установки, нужно зайти в папку hardware\espressif\arduino-esp32\tools и запустить файл get.exe, который скачает недостающие файлы. Процесс также может занять несколько минут.

После этого заново запустим Arduino IDE - мы увидим, что количество доступных типов плат увеличилось, выберем ESP32 Dev Module.



Теперь можно подключить плату, и испытать какие-нибудь примеры. Поскольку, возможность работы с WiFi и Bluetooth является одним из основных преимуществ платы, это мы и проверим. Выберем в меню File-Examples-WiFi scan. Запускаем компиляцию… и получаем сообщение об ошибке. Оказывается, библиотека WiFi.h, идущая в комплекте с ESP32 SDK, конфликтует с уже имеющейся в папке "C:\Program Files (x86)\Arduino\libraries" библиотекой с тем же названием. Удаляем папку WiFi оттуда, перезапускаем Arduino IDE, после этого компиляция происходит нормально.

Код сканирования WiFi-сетей приведен ниже, как можно видеть, он весьма прост, и по стилю практически ничем не отличается от рассматриваемых ранее примеров для Arduino.

```
#include "WiFi.h"

void setup() {
Serial.begin(115200);
WiFi.mode(WIFI_STA);
```

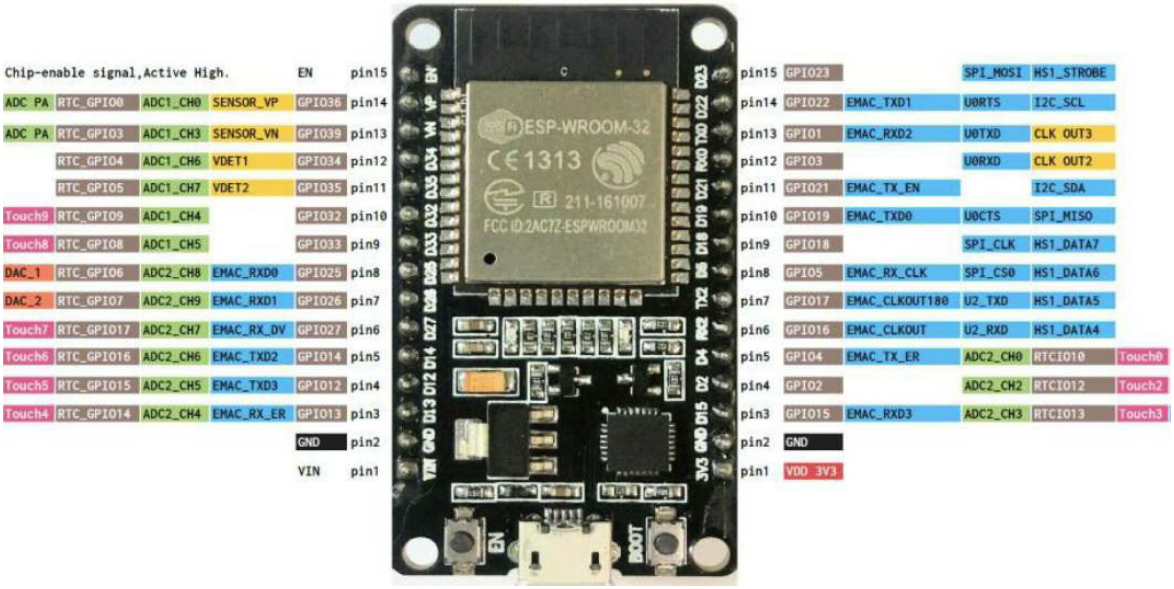
```
WiFi.disconnect();
delay(100);
}

void loop() {
Serial.println("scan start");
int n = WiFi.scanNetworks();
Serial.println("scan done");
if (n == 0) {
Serial.println("no networks found");
} else {
Serial.print(n);
Serial.println(" networks found");
for (int i = 0; i < n; ++i) {
Serial.print(i + 1);
Serial.print(": ");
Serial.print(WiFi.SSID(i));
Serial.print(" (");
Serial.print(WiFi.RSSI(i));
Serial.print(")");
Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?"
":"*");
delay(10);
}
}
Serial.println("");
delay(5000);
}
```

Запускаем программу, и получаем список доступных WiFi-сетей:

Ура - мы проверили возможность новой платы работать с сетевым окружением, то, что не было доступно для Arduino.

3.2 Порты ввода-вывода

Перед использованием любой платы нужно разобраться, где и какие выводы расположены. Самый простой способ в нашем случае - набрать в поиске гугла фразу “ESP32 dev board pinout”. Для используемой в опытах платы ESP32 расположение пинов показано на картинке, впрочем для разных модификаций оно может отличаться.



Протестируем возможности ввода вывода на этой плате. Код программы практически ничем не отличается от Arduino, хотя и есть некоторые отличия. Для вывода результатов будем использовать serial port.

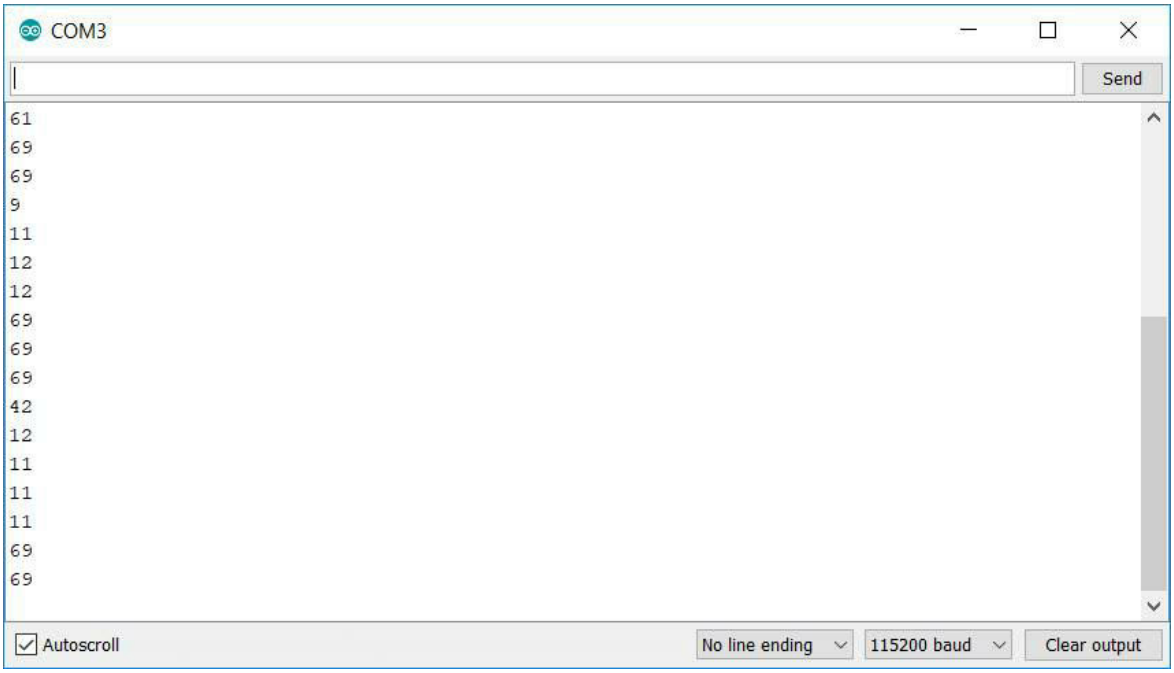
```
// Встроенный светодиод на плате
int ledPin = 2;
// Кнопка
int inputPin = 23;
// Тач-панель
int touchPin = 4;

void setup() {
Serial.begin(115200);
pinMode(ledPin, OUTPUT);
pinMode(inputPin, INPUT_PULLUP);
}

void loop() {
// Читаем значение touchPin
int touch_value = touchRead(touchPin);
Serial.println(touch_value);
// Читаем значение inputPin
int buttonState = digitalRead(inputPin);
Serial.println(buttonState);

// Мигаем светодиодом
digitalWrite(ledPin, HIGH);
delay(1000);
digitalWrite(ledPin, LOW);
delay(1000);
}
```

Итак, несколько отличий. Как говорится в старом анекдоте, есть 2 новости - плохая и хорошая. Точнее, хороших новостей даже несколько.

1. Ввод данных с кнопки. Точно также, нужно настроить порт как “вход” (input). Но обратим внимание на новый параметр

INPUT_PULLUP. Да, теперь “подтягивающий резистор” на плату можно не ставить, он уже есть в процессоре и его можно активировать программно.

2. Аппаратная поддержка касания (touch). Обратим внимание на выводы, отмеченные на картинке как Touch0...Touch9. Их можно использовать как тач-панель, вместо кнопок. Достаточно просто подключить к такому выводу кусок провода, и его касание рукой будет изменять значение переменной. Для примера выведем в порт значения touch_value:



Как можно видеть, в момент нажатия возвращаемые значения уменьшаются. Это можно использовать в коде с помощью обычной проверки вроде *if (touch_value < 32) {...}*.

3. Что касается вывода, то по сравнению с Arduino, он ничем не изменился, те же функции digitalWrite(ledPin, HIGH) и digitalWrite(ledPin, LOW).

Теперь обещанная плохая новость. Если мы возьмем плату с мигающим светодиодом, и тестером померяем напряжение на выходе, то увидим не 5В, а только 3.3В. Действительно, новые процессоры используют более низкое напряжение, по сравнению со старыми

платами Arduino. Это не является какой-то глобальной проблемой, но при покупке датчиков или аксессуаров (например ЖК-экрана), стоит обратить внимание на то, чтобы датчик был совместим с 3.3В.

К примеру, посмотрим описание экрана на eBay:

Product Introduction:

1.Size:1.3 inch
2.Resolution:128*64
3.Controlling Chip:SSH1106
4.Display Area:29.42*14.7mm
5.Driving Voltage:3.3-5V

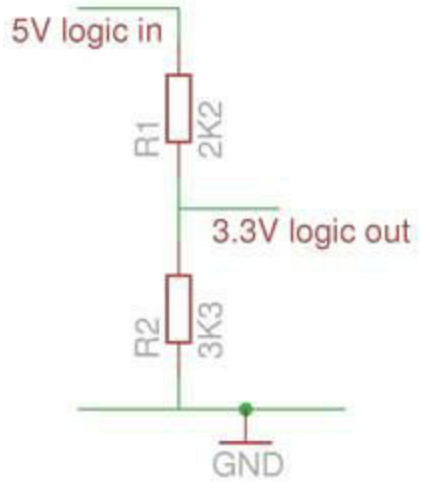
Последняя строка Driving voltage - это то что нам нужно знать, чтобы убедиться что экран или датчик будет корректно работать с напряжением 3.3В.

Однако, что делать если есть какой-то нужный нам сенсор, работающий только от 5В? Выход 5В для питания сенсора на плате есть, он помечен как VIn, это не проблема. Проблема в вводе-выводе данных.

Здесь есть три способа:

1. Подключить “как есть”. При этом на входные пины платы будет подаваться напряжение 5В вместо 3.3В, скорее всего такое подключение работать будет, но это не гарантируется, да и не рекомендуется. Повышенное напряжение может привести к сокращению срока службы процессора.

2. Если данные поступают только на вход, то можно использовать делитель напряжения, который понизит напряжение с 5 до 3.3В:



3. Если нужен двухсторонний обмен данными, то можно использовать специальные платы, которые называются “logic level shifter”.

Такая плата ценой 1-2$ включается между устройствами, и обеспечивает необходимое преобразование уровней с помощью полевого транзистора. В наших опытах она не понадобится, но о том что такие платы есть, может быть полезно знать.

3.3 Подключаемся к WiFi

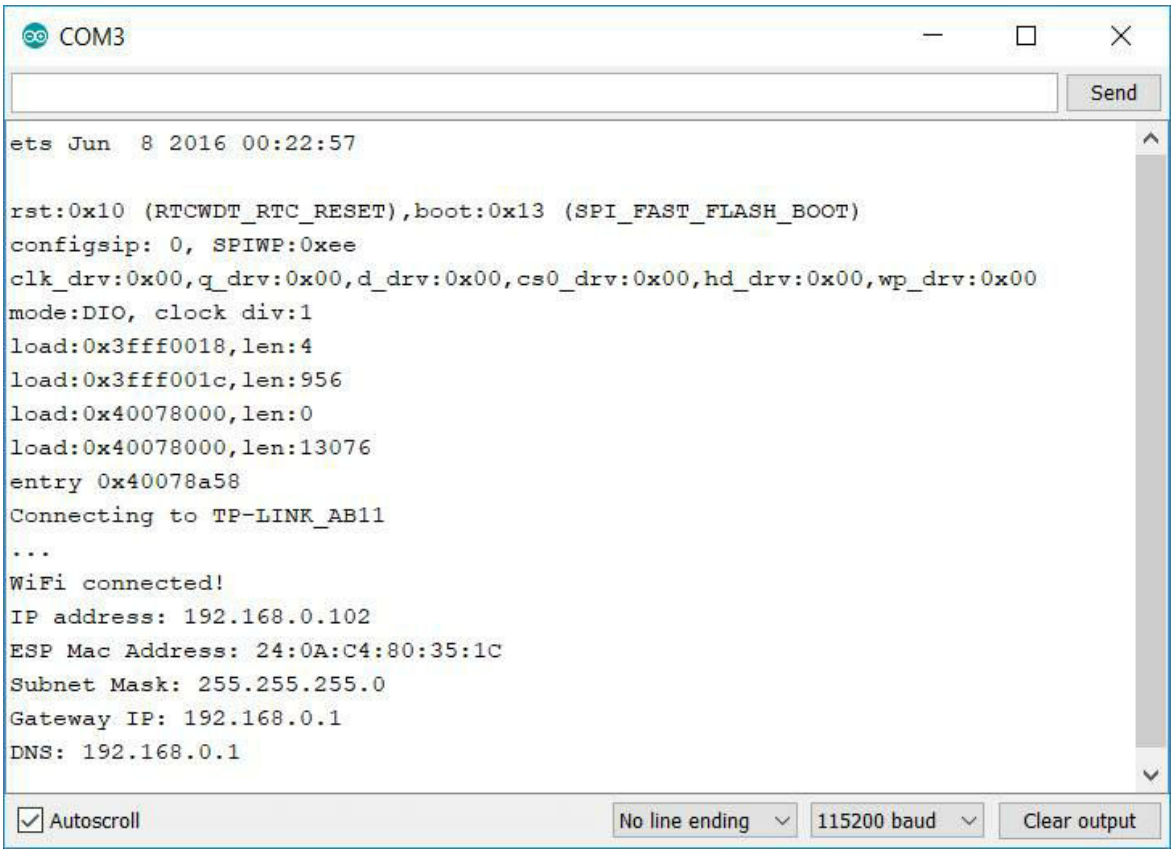
С этой главы мы начнем более глубокое погружение в мир “интернета вещей”. Это несложно, но потребует от нас некоторого знания сетевых технологий. Начнем с самого основного, без чего мы не сможем двигаться дальше - мы подключим нашу плату к WiFi-сети.

Код, делающий это, весьма прост, нужно знать лишь имя и пароль.

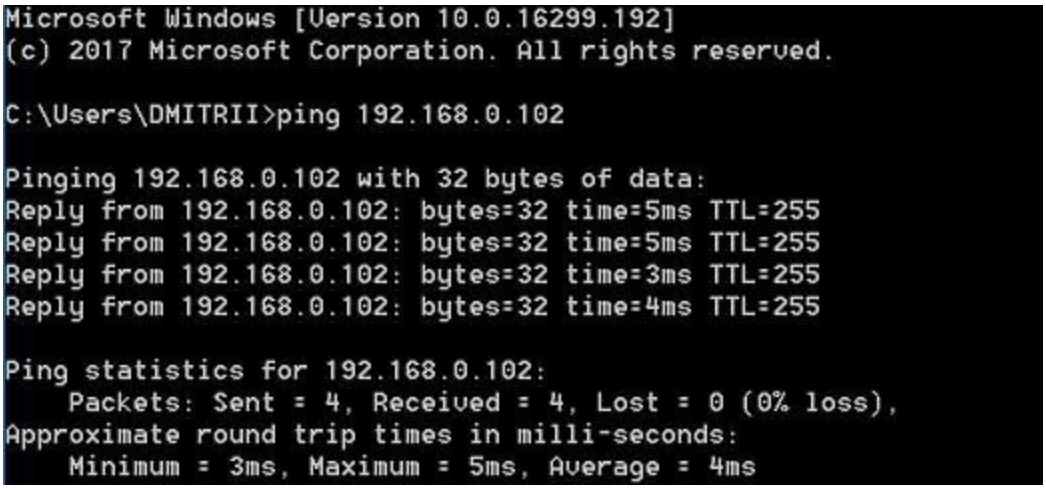
```
#include <WiFi.h>
```

```
const char* ssid = "TP-LINK_AB11";
const char* password = "12345678";

void setup() {
Serial.begin(115200);

Serial.print("Connecting to ");
Serial.println(ssid);

while (WiFi.status() != WL_CONNECTED) {
delay(1000);
Serial.print(".");
WiFi.begin(ssid, password);
}

Serial.println("");
Serial.println("WiFi connected!");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
Serial.print("ESP Mac Address: ");
Serial.println(WiFi.macAddress());
Serial.print("Subnet Mask: ");
Serial.println(WiFi.subnetMask());
Serial.print("Gateway IP: ");
Serial.println(WiFi.gatewayIP());
Serial.print("DNS: ");
Serial.println(WiFi.dnsIP());
}

void loop() {
}
```

Как можно видеть, мы вызываем функцию WiFi.begin для установки соединения, и проверяем WiFi.status() для проверки, активно ли соединение. Параметры ssid и password хранят данные, необходимые для подключения. Когда подключение установлено, мы можем получить параметры соединения: IP-адрес, маску подсети и пр. Включив Serial monitor, мы можем увидеть следующие данные:



В логе можно видеть, что соединение установлено, и плата получила IP-адрес 192.168.0.102. Каждое устройство в сети имеет свой собственный IP-адрес, по которому к нему можно обратиться. То что адрес доступен, можно проверить командой ping: открываем консоль (Win+R - cmd), набираем ping 192.168.0.102. В ответ мы должны получить примерно такой результат, если плата отвечает на запрос:



Итак, мы успешно подключились к Сети, и можем двигаться дальше.

Важно: Как можно видеть в вышеприведенном коде, соединение с сетью WiFi происходит лишь в функции setup, которая активируется только один раз при включении платы. Соответственно, при перезагрузке маршрутизатора или потере WiFi-сигнала соединение не будет восстановлено. Чтобы этого избежать, нужно добавить в функцию loop проверку на необходимость восстановления коннекта.

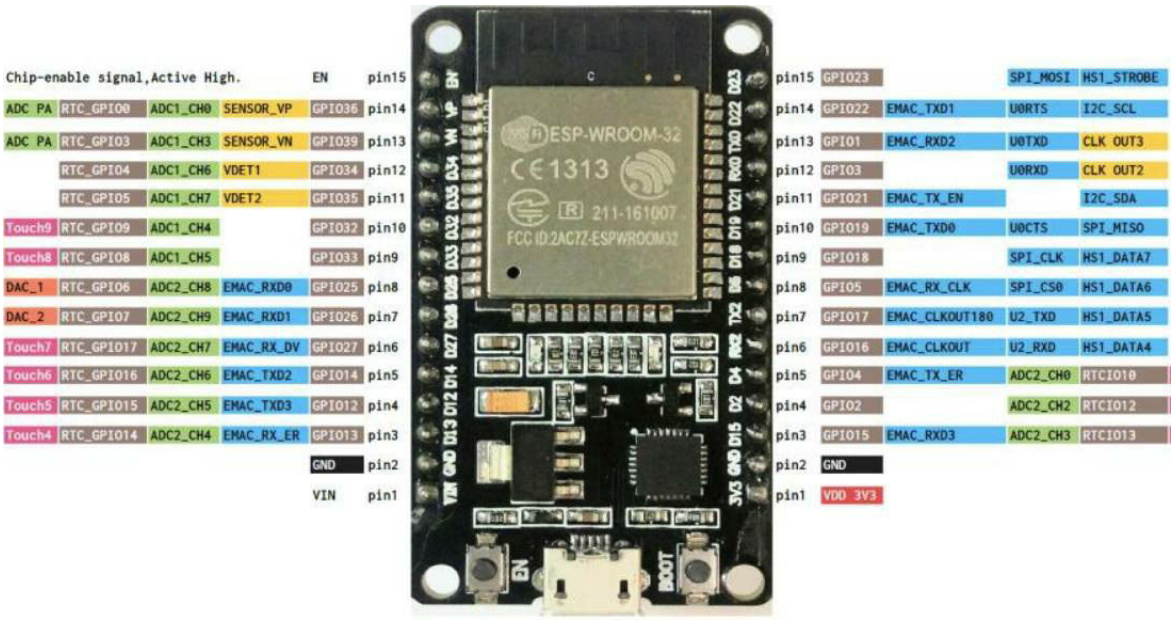
```
while (WiFi.status() != WL_CONNECTED) {
delay(1000);
WiFi.begin(ssid, password);
}
```

Для наших тестовых программ это не требуется, но в реально действующем коде такая проверка необходима.

3.4 Подключаем дисплей

Мы уже подключали OLED-дисплей к Arduino, сделаем то же самое и для ESP32. Дисплей пригодится нам в следующих проектах, когда мы будем получать данные по WiFi из Интернет.

Посмотрим еще раз на картинку выводов платы (напомню, для разных плат они могут отличаться).



Нам нужны выводы GPIO21 и GPIO22, которые подписаны как I2C_SCL и I2C_SDA. Для тех кто забыл предыдущую часть, напомним что дисплей подключается по I2C и выглядит вот так:

Подключаем выводы дисплея к контактам 3.3V, GND, SCL и SDA. Этот дисплей может работать и от 5В и от 3.3В, так что здесь проблем нет. Затем скачиваем и устанавливаем библиотеки для OLED-дисплея по адресу https://github.com/ThingPulse/esp8266-oled-ssd1306.

Код для работы дисплея приведен ниже. Как можно видеть, он практически не отличается по сути от предыдущей рассмотренной версии для Arduino. Значения 21 и 22 - это номера выводов, соответствующие GPIO21 и GPIO22.

```
#include "SSD1306.h"
```

```
SSD1306 display(0x3c, 21, 22);

void setup() {
display.init();
display.flipScreenVertically();
}

void loop() {
display.clear();
display.setFont(ArialMT_Plain_10);
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.drawString(0, 0, "Hello world");
display.setFont(ArialMT_Plain_16);
display.drawString(0, 10, "Hello world");
display.setFont(ArialMT_Plain_24);
display.drawString(0, 26, String(42));
display.display();

delay(5000);
}
```

Из полезных моментов можно отметить функцию setFont, с помощью которой можно задать разный размер шрифта. Кстати, можно выводить не только строки, но и числа, преобразовав их с помощью String, как показано в коде.

Теперь мы можем выводить нужные нам данные на экран. Следующим шагом узнаем, как получать необходимые данные из сети Интернет.

Самостоятельная работа: изучить исходный текст библиотеки рисования по адресу https://github.com/ThingPulse/esp8266-oled-ssd1306/blob/master/OLEDDisplay.h. Помимо функции drawString, там описано множество других полезных функций, например drawCircle или drawProgressBar. Испытать их в программе.

3.5 Получаем время от атомных часов

Раз уж мы подключились к Интернет, можно сделать много чего

полезного. Например, получить точное время с атомных часов от NTP-сервера, тем более что библиотеки для этого уже есть.

Добавим в программу соединения с WiFi код вывода точного времени.

```
#include <WiFi.h>
#include <time.h>

const char* ssid = "TP-LINK_AB11";
const char* password = "12345678";

const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 3600;
const int daylightOffset_sec = 3600;

void setup() {
Serial.begin(115200);
while (WiFi.status() != WL_CONNECTED) {
delay(1000);
Serial.print(".");
WiFi.begin(ssid, password);
}
Serial.println("WiFi connected!");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

// Настройки службы времени
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
}

void loop() {
// Получение времени
struct tm timeinfo;
if (!getLocalTime(&timeinfo)){
Serial.println("getLocalTime: error");
delay(10000);
return;
```

```
}
Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
delay(30000);
}
```

Как можно видеть, мы добавили вызов двух функций: configTime и getLocalTime. Первая функция настраивает нужные параметры (например часовой пояс), вторая получает время. Кстати, само время хранится в структуре tm, которая хранится в файле time.h и имеет следующий вид:

```
struct tm {

int tm_sec; // Seconds after the minute [0, 59]

int tm_min; // Minutes after the hour [0, 59]

int tm_hour; // Hours since midnight [0, 23]

int tm_mday; // Day of the month [1, 31]

int tm_mon; // Months since January [0, 11]

int tm_year; // Years since 1900

int tm_wday; // Days since Sunday [0, 6]

int tm_yday; // Days since January 1 [0, 365]

int tm_isdst; // Daylight Saving Time flag

int tm_gmtoff; // Seconds east of UTC

char *tm_zone; // Timezone abbreviation

};
```

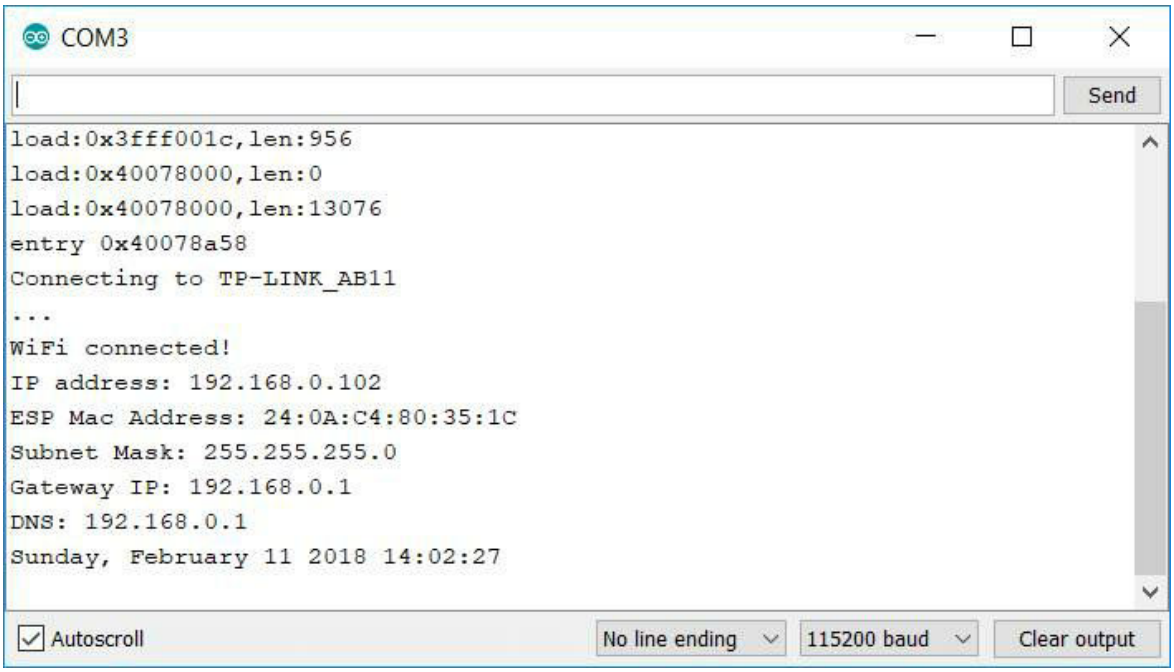
Все эти поля можно использовать. Например, можно зажечь светодиод (или запустить зуммер) в 8 часов утра:

```
if (timeinfo.tm_hour == 8 && timeinfo.tm_min == 0) {
```

...
}

Мы также используем строку “%A, %B %d %Y %H:%M:%S”, которая указывает в каком формате выводить дату и время. Например, чтобы вывести только время, достаточно написать “%H:%M:%S”. Список разных форматов можно найти здесь.

Запустим программу, и в Serial Monitor увидим на экране точное время.



Самостоятельная работа: вывести точное время на экран дисплея.

3.6 Выводим количество друзей в “Контакте”

Продолжим углубляться в тонкости Web. На этот раз мы выведем количество друзей “ВКонтакте” для заданного пользователя. Подобная задача довольно часто встречается в профессиональном программировании, когда нужно получить и отобразить данные с какого-то сайта. Поэтому рассмотрим подробнее, как это делается.

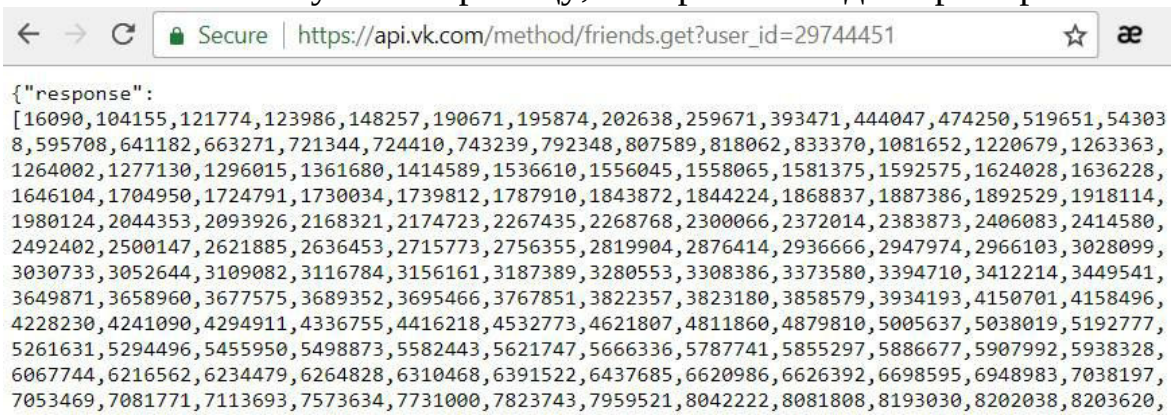
Шаг-1. Находим, как получить список друзей

Каждый крупный сайт имеет так называемый “интерфейс разработчика”, или API - Application Program Interface, набор функций,

доступный программистам для доступа к данным. Набираем в гугле “vk.com get friends API” (международным языком в программировании является английский), и первая же ссылка возвращает нам нужную страницу с описанием параметров: https://vk.com/dev/friends.get.

Еще немного ищем в интернете примеры использования, и в итоге получаем, что нам нужно выполнить такой запрос: https://api.vk.com/method/friends.get?user_id=ID, где ID - это идентификатор пользователя, который можно посмотреть в подсказке к ссылке на каждой странице. Берем первый попавшийся идентификатор, например 29744451, и открываем ссылку https://api.vk.com/method/friends.get?user_id=29744451 в браузере.

Работает! Мы получили страницу, которая выглядит примерно так:



Данный ответ сервера представляет собой список идентификаторов друзей пользователей в формате Json. Это текстовый формат, для чтения которого есть специальные библиотеки. Но сначала нужно загрузить сами данные.

Шаг-2. Получение данных с сервера.

Для получения данных в Arduino есть специальный класс HTTPClient.

Код для загрузки данных выглядит так:

```
const char *url = "https://api.vk.com/method/friends.get?user_id=29744451";
HTTPClient http;
http.begin(url);
```

```
int http_code = http.GET();
if (http_code > 0) {
String json = http.getString();
...
}
```

Можно вывести данные в serial port и убедиться, что возвращается правильный список друзей. Он должен выглядеть также, как и в браузере по ссылке выше.

Шаг-3. Обработка данных

Нам не нужен весь список целиком, достаточно лишь узнать количество друзей. Для этого используем библиотеку обработки (парсинга) Json, скачать которую можно по ссылке https://github.com/bblanchon/ArduinoJson. Как обычно, файлы необходимо скачать и распаковать в папку Документы\Arduino\libraries.

Чтение данных из массива и получение его количества с помощью этой библиотеки выглядит так.

```
DynamicJsonBuffer jsonBuffer(16*1024);
JsonObject& parsed = jsonBuffer.parseObject(json);
if (parsed.success()) {
JsonArray& response = parsed["response"];
int count = response.size();
Serial.print("Number of friends: ");
Serial.println(count);
} else {
Serial.println("Parsing error");
}
```

Как можно видеть, мы получаем объект типа JsonArray, у которого узнаем размер вызовом метода size(). Также мы создаем объект DynamicJsonBuffer для хранения распакованных данных, при этом выделяется 16Кб памяти для временного буфера в памяти.

Результат готов! Мы получили данные в виде переменной count, теперь мы можем вывести данные в serial port.

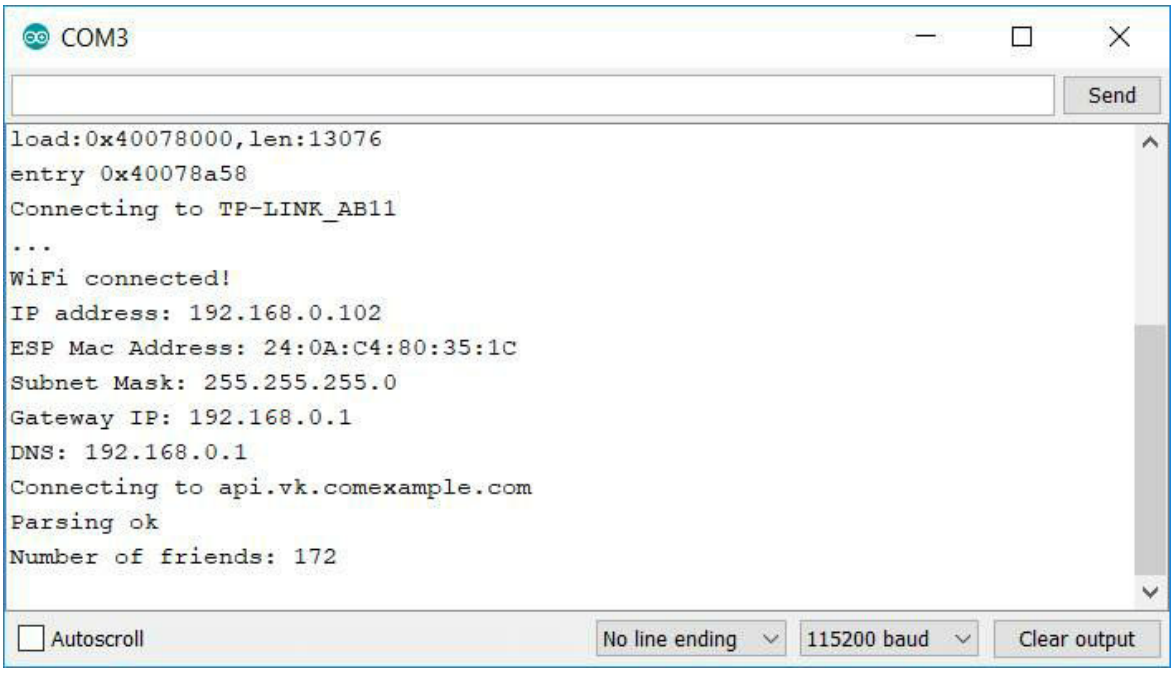
Код программы полностью выглядит так.

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

const char* ssid = "TP-LINK_AB11";
const char* password = "12345678";

void setup() {
Serial.begin(115200);

Serial.print("Connecting to ");
Serial.println(ssid);

while (WiFi.status() != WL_CONNECTED) {
delay(1000);
Serial.print(".");
WiFi.begin(ssid, password);
}

Serial.println("");
Serial.println("WiFi connected!");
}

void loop() {
const char *url = "https://api.vk.com/method/friends.get?
user_id=134212064";
Serial.println("Connecting to api.vk.com");

HTTPClient http;
http.begin(url);
int httpCode = http.GET();
if (httpCode > 0) {
String json = http.getString();
// Обработка Json
DynamicJsonBuffer jsonBuffer(16*1024);
```

```
JsonObject& parsed = jsonBuffer.parseObject(json);
if (parsed.success()) {
JsonArray& response = parsed["response"];
int count = response.size();
Serial.print("Number of friends: ");
Serial.println(count);
} else {
Serial.println("Json parsing error");
}
} else {
Serial.println("HTTP request error");
}
http.end();

// Пауза 2 мин
delay(120000);
}
```

Важно: при создании любых запросов к серверу рекомендуется делать их как можно реже, чтобы не перегружать сервер. Именно поэтому в конце функции стоит пауза на 2 минуты. Слишком частые запросы могут стать даже причиной блокировки (бана) по IP-адресу, особенно это касается плат типа ESP32, которые могут работать круглые сутки. Поэтому стоит делать запросы так редко как возможно, исходя из логики работы программы. Например, если мы делаем устройство, показывающее количество друзей в “Контакте”, то нет смысла запрашивать число друзей каждую минуту, скорее всего и интервала проверки раз в 10 минут или даже 1 час будет вполне достаточно.

Наконец, все готово. Запустим программу на ESP32, и в окне Serial monitor мы можем увидеть количество друзей у заданного нами пользователя.



Может возникнуть вопрос, как вывести не только число пользователей, но и более личные данные, например число входящих сообщений. Увы, такие запросы требуют аутентификации (логина) по имени и паролю, и так просто, одним запросом их не сделать. Желающие могут изучить документацию к API самостоятельно на сайте https://vk.com/dev/manuals.

Самостоятельная работа: вывести число друзей на OLED-экран. Дополнительно можно добавить звуковой сигнал, который будет срабатывать, когда число друзей увеличивается.

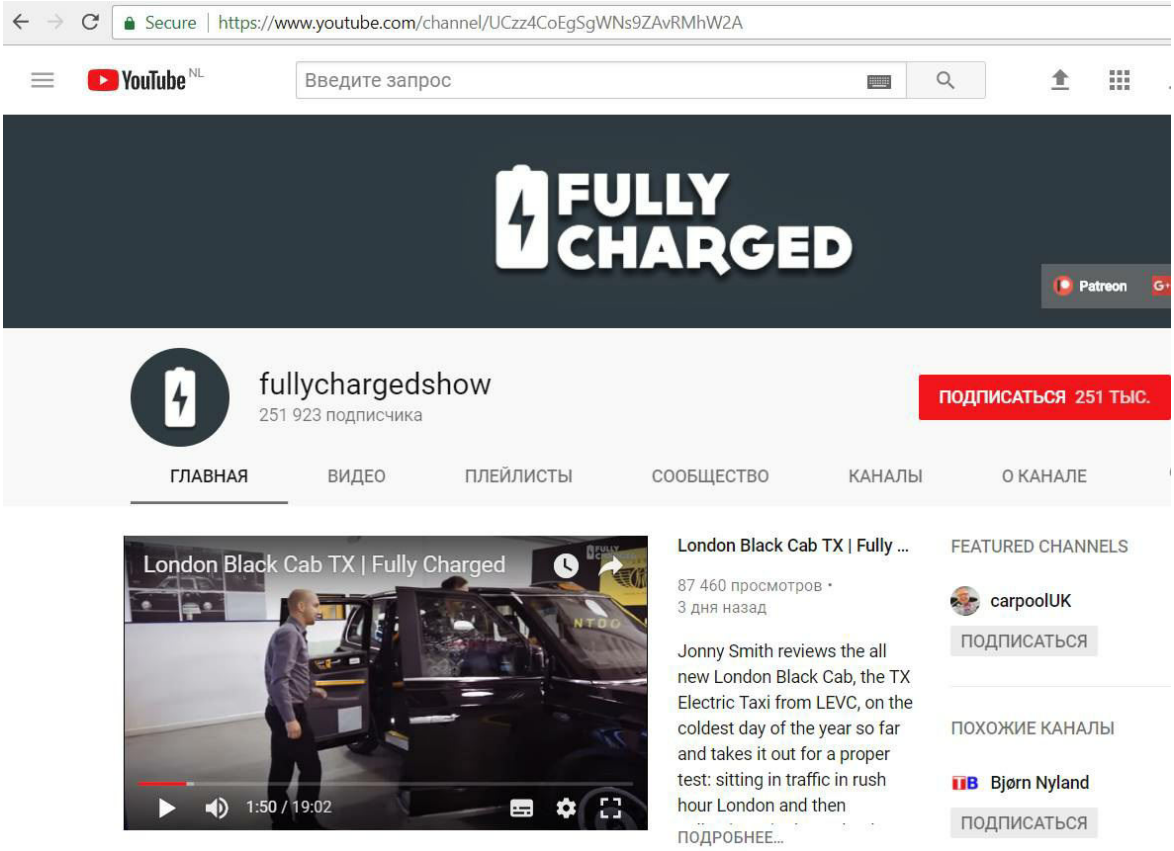
3.7 Выводим число подписчиков, просмотров и лайков в Youtube

У популярных ведущих youtube-каналов иногда можно увидеть на столе устройство, показывающее число подписчиков канала. Сделаем такое же, с платой ESP32 это не сложно. Те, кто читал предыдущую главу, уже примерно представляют что нужно делать. Действительно, принцип точно такой же. Приступим.

Шаг-1. Узнаем идентификатор канала.

Сначала нужно узнать идентификатор канала. Это самое простое,

кликаем на любой популярный канал в youtube и смотрим на адресную строку в браузере.



Идентификатор UCzz4CoEgSgWNs9ZAvRMhW2A в заголовке - это и есть идентификатор канала, запомним его.

Шаг-2. Находим API для получения информации

Функцию для получения информации о канале можно найти на странице для разработчиков https://developers.google.com/youtube/v3/docs/channels/list (да, там все на английском, английский - это язык современной науки и технологии, и его надо знать). Там же есть примеры использования.

Нужный нам запрос имеет следующий вид:
https://www.googleapis.com/youtube/v3/channels?id=**ID**&part=statistics&key=**KEY**

Как получить ID канала, мы уже знаем. С ключом чуть сложнее.

Ключ для доступа к API бесплатный, получить его можно на странице https://console.developers.google.com в разделе “учетные данные”. Сам ключ - это обычная текстовая строка, которая выглядит примерно так: AIzaSyC26UJw-ubU6N6ukrbZ_C_nBaxxxxxxxxx.

Инструкция по получению ключа также есть на странице подсказки google.

Go to the API Console.

From the projects list, select a project or create a new one.

If the APIs & services page isn't already open, open the left side menu and select **APIs & services**.

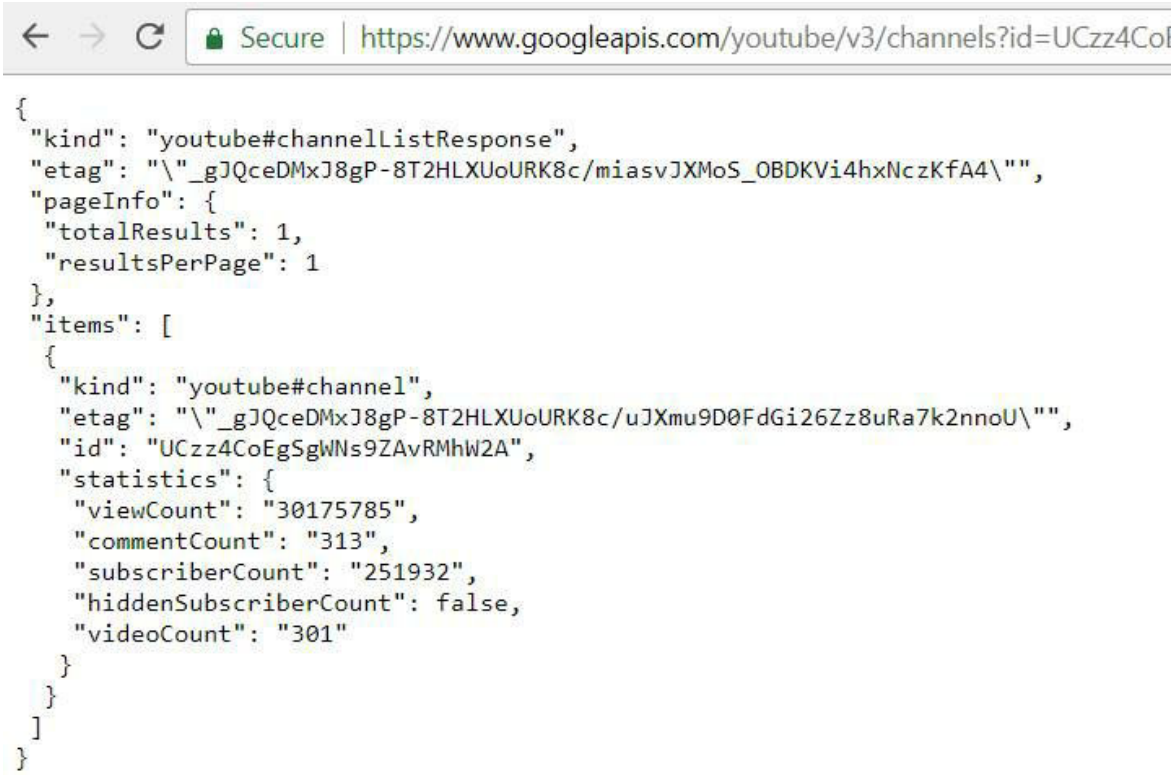
On the left, choose **Credentials**.

Click **Create credentials** and then select **API key**.

Я специально оставлю ее как есть без перевода, чтобы еще раз подчеркнуть важность знания английского в современной разработке. URL целиком выглядит так:

https://www.googleapis.com/youtube/v3/channels?id=UCzz4CoEgSgWNs9ZAvRMhW2A&part=statistics&key=AIzaSyC26UJw-ubU6N6ukrbZ_C_nBaxxxxxxxxx

Здесь UCzz4CoEgSgWNs9ZAvRMhW2A - это идентификатор канала, а AIzaSyC26UJw-ubU6N6ukrbZ_C_nBaxxxxxxxxx - это ключ доступа. Мы можем вставить эту строку в браузер, и получить информацию о канале.



Как можно видеть, нужные нам данные хранятся в разделе items/statistics. Запомним это, когда будем делать обработку json.

Шаг-3. Чтение данных

Здесь мы повторяем фактически то же, что мы делали для чтения с сайта “ВКонтакте”.

Код для запроса к сайту:

```
const char *url = "https://www.googleapis.com/youtube/v3/channels?id=UCzz4CoEgSgWNs9ZAvRMhW2A&part=statistics&key=XXXXX";

HTTPClient http;
http.begin(url);
int httpCode = http.GET();
if (httpCode > 0) {
Serial.print("httpCode: "); Serial.println(httpCode);
String json = http.getString();

} else {
Serial.println("HTTP request error");
```

```
Serial.println(httpCode);
}
```

Кстати, коды ошибок сервера - очень важны для анализа, если что-то не так. Например, код 404 говорит о том, что страница не найдена, 403 - “доступ запрещен” и пр. Полный список можно почитать в Википедии.

Шаг-4. Обработка Json

Еще раз внимательно посмотрим на json, который мы получили:

```
{

"kind": "youtube#channelListResponse",

"items": [

{

"kind": "youtube#channel",

"statistics": {

"viewCount": "30177125",

"commentCount": "313",

"subscriberCount": "251966",

"videoCount": "301"

}

}

]

}
```

Как можно видеть, items - это массив элементов, т.к. он содержит скобки []. Внутри есть объект statistics, а внутри него нужное нам поле subscriberCount.

Код чтения Json будет выглядеть вот так:

```
DynamicJsonBuffer jsonBuffer(16*1024);
JsonObject& parsed = jsonBuffer.parseObject(json);
if (parsed.success()) {
JsonArray& items = parsed["items"];
if (items.size()> 0) {
JsonObject& statistics = items[0]["statistics"];
long subscriberCount = statistics["subscriberCount"];
Serial.print("Number of subscribers: ");
Serial.println(subscriberCount);
}
} else {
Serial.println("Json parsing error");
}
```

Мы создаем объект типа JsonArray, и если он не пуст, обращаемся к элементу с первым номером (в Си массивы нумеруются с нуля, items[0] это первый элемент).

Задача решена! Код полностью выглядит так:

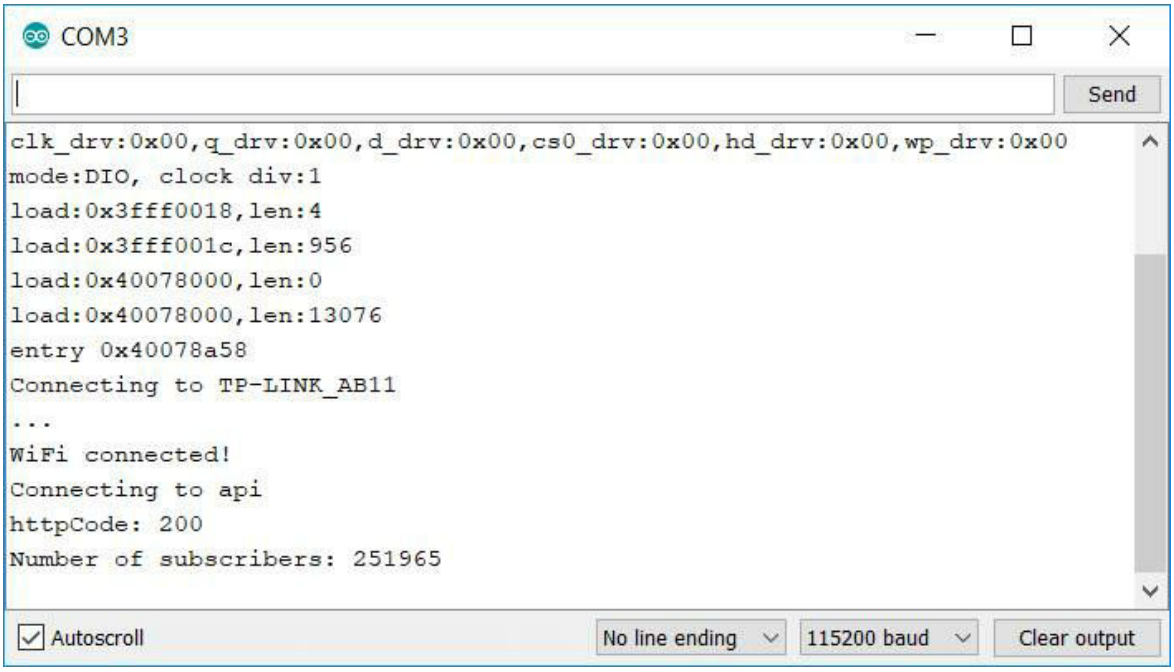
```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

const char* ssid = "TP-LINK_AB11";
const char* password = "12345678";

void setup() {
Serial.begin(115200);
Serial.print("Connecting to ");
Serial.println(ssid);
while (WiFi.status() != WL_CONNECTED) {
delay(1000);
```

```
Serial.print(".");
WiFi.begin(ssid, password);
}
Serial.println("");
Serial.println("WiFi connected!");
}

void loop() {
const char *url = "https://www.googleapis.com/youtube/v3/channels?
id=UCzz4CoEgSgWNs9ZAvRMhW2A&part=statistics&key=XXXXXXXX
Serial.println("Connecting to www.googleapis.com");

HTTPClient http;
http.begin(url);
int httpCode = http.GET();
if (httpCode > 0) {
Serial.print("httpCode: "); Serial.println(httpCode);
String json = http.getString();
// Парсинг Json
DynamicJsonBuffer jsonBuffer(16*1024);
JsonObject& parsed = jsonBuffer.parseObject(json);
if (parsed.success()) {
JsonArray& items = parsed["items"];
if (items.size()> 0) {
JsonObject& statistics = items[0]["statistics"];
long subscriberCount = statistics["subscriberCount"];
Serial.print("Number of subscribers: ");
Serial.println(subscriberCount);
}
} else {
Serial.println("Json parsing error");
}
} else {
Serial.println("HTTP request error");
Serial.println(httpCode);
}
http.end();
```

```
// Пауза
delay(120000);
}
```

Для использования этого кода будет необходимо заменить имя и пароль WiFi-сети, а также вставить идентификатор канала и правильный ключ для доступа к Google API.

При запуске программы данные будут выведены в Serial Monitor:



Самостоятельная работа: По аналогии с вышенаписанным, получить число просмотров для видеоролика на youtube. Для этого воспользоваться следующей функцией из Google API:

https://www.googleapis.com/youtube/v3/videos?part=statistics&id=ID&key=XXXX

(здесь ID - это идентификатор видеоролика, XXXX - Google API Key)

Функция вернет json, который выглядит примерно так:

```
{
```

"kind": "youtube#videoListResponse",

"etag": "\"_gJQceDMxJ8gP-8T2HLXUoURK8c/mISbZAYZi79hOI4WFXbLXPUphmg\"",

"items": [

{

"kind": "youtube#video",

"etag": "\"_gJQceDMxJ8gP-8T2HLXUoURK8c/PoZyfbkQEtIkGNfjDsAafjr_R08\"",

"id": "CizKnuwvXXg",

"statistics": {

"viewCount": "184277",

"likeCount": "8585",

"dislikeCount": "212",

"favoriteCount": "0",

"commentCount": "1349"

}

}

]

```
}
```

Необходимо исправить в коде обработку json, чтобы получить интересующие нас поля.

При желании можно вывести данные на OLED-дисплей, сделав автономно работающее устройство.

3.8 Запускаем собственный Web-сервер

Мы уже умеем получать и обрабатывать данные с различных серверов. Настала пора двигаться дальше - мы запустим на ESP32 собственный сервер. Его можно будет открыть в браузере, введя IP-адрес платы, подключенной к WiFi-сети. А если настроить статический IP-адрес и перенаправление портов на маршрутизаторе, то можно будет получить доступ к нашему серверу через Интернет, с любой точки земного шара!

Итак, приступим. Для использования ESP32 в качестве сервера мы будем использовать уже готовый класс WiFiServer. Он “слушает” входящие соединения, а его единственным параметром является номер порта (мы будем использовать порт 8000).

Код сервера приведен ниже.

```
#include <WiFi.h>

const char* ssid = "TP-LINK_AB11";
const char* password = "12345678";

WiFiServer server(8000);

void setup()
{
Serial.begin(115200);
delay(10);

// We start by connecting to a WiFi network
Serial.print("Connecting to "); Serial.println(ssid);
```

```
while (WiFi.status() != WL_CONNECTED) {
delay(500);
WiFi.begin(ssid, password);
Serial.print(".");
}
Serial.println("");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();
}

void loop() {
WiFiClient client = server.available(); // Получение входящего “клиента”
if (client) {
Serial.println("New Client.");
String currentLine = "";
while (client.connected()) {
if (client.available()) { // Есть непрочитанные данные
char c = client.read(); // читаем 1 байт
Serial.write(c);
if (c == '\n') { // Символ перевода строки
// 2 пустые строки - признак конца запроса, посылаем ответ
if (currentLine.length() == 0) {
client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println();
client.print("<html><head> <title>ESP32 Server</title> </head>”);
client.print("<body>Hello world</body> </html>");
client.println();
break;
} else {
// Очистить строку
currentLine = "";
}
} else if (c != '\r') { // if you got anything else but a carriage return
```

```
character,
    currentLine += c; // add it to the end of the currentLine
    }
    }
    }
    // Закрыть соединение:
    client.stop();
    }
    }
```

Рассмотрим код подробнее.

Мы создаем объект WiFiServer, который “слушает” входящие соединения на определенном порту (в нашем случае 8000). Тот, кто подсоединяется к нашему серверу, называется “клиентом”. За взаимодействие с ним отвечает класс WiFiClient.

Далее, мы побайтно читаем данные с помощью вызова функции client.read(), и складываем данные в строку currentLine. К примеру, строка запроса может быть такой - GET / HTTP/1.1, здесь “GET” это тип запроса, “/” - это адрес главной страницы, 1.1 - тип стандарта. Но сам запрос мы не анализируем, а просто посылаем всегда один и тот же ответ - заглавную страницу с HTML-текстом “Hello world”. Строки типа “HTTP/1.1 200 OK” - это служебная информация, которая нужна браузеру (клиенту) чтобы понять, что мы ответили (200 - это код возврата OK, говорящий о том что все нормально). Строка “Content-type:text/html” говорит клиенту о том, что возвращаться будет HTML-страница.

Наша страница имеет следующий вид:

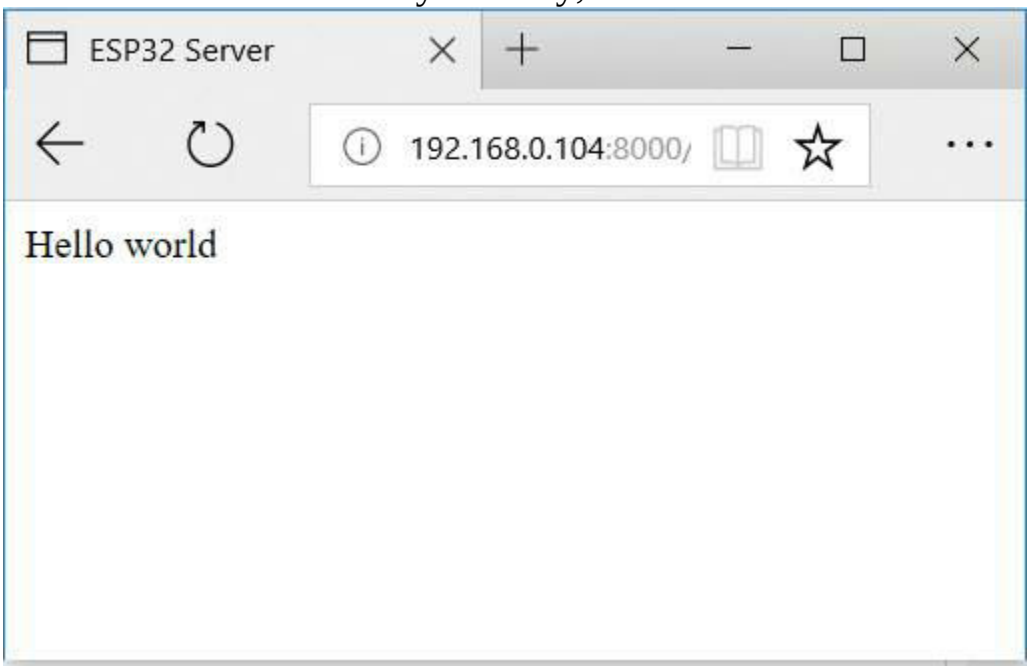
```
<html>
<head>
<title>ESP32 Server</title>
</head>

<body>Hello world</body>
</html>
```

Как можно видеть, она содержит “заголовок” (header) и “тело”

(body). Заголовок содержит тег title - в нем хранится строка, которая будет выведена в заголовке браузера.

Итак, компилируем данный код, загружаем его в плату, и в Serial Monitor смотрим какой IP-адрес получила ESP32 при доступе к WiFi-сети. Вводим этот адрес в строку браузера, не забыв указать номер порта, например http://192.168.0.104:8000.

Все готово! Через небольшое время ожидания мы видим страницу браузера с нашим текстом Hello world. Обращаем внимание на то, что и заголовок и текст соответствуют тому, что мы ввели.



Самостоятельная работа: изучить команды HTML, поэкспериментировать с разными вариантами разметки и форматирования текста.

3.9 Управляем светодиодом через Web

Мы уже умеем запустить свой Web-сервер, но пока он не делает ничего полезного, кроме отображения Hello world. Исправим это упущение, и сделаем кнопки для управления светодиодом через веб браузер.

Принцип простой - мы добавим в HTML-страницу 2 ссылки,

нажатие которых и будет отслеживаться сервером.

Новая HTML-страница будет выглядеть так:

```
<html>
<head><title>ESP32 Server</title></head>
<body>
Turn <a href=\"/H\">LED ON</a><br>
Turn <a href=\"/L\">LED OFF</a><br>
</body>
</html>
```

В HTML-странице можно видеть 2 ссылки с значением адреса /H и /L, когда пользователь нажмет на них, на сервер будет отправлен GET-запрос с соответствующим адресом. Это нам и нужно.

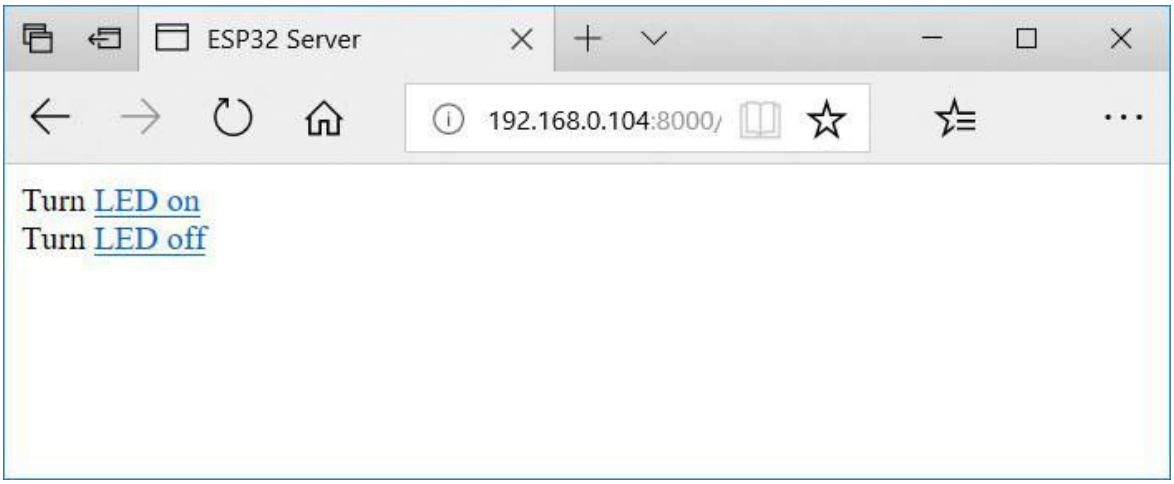
Код программы целиком приведен ниже.

```
#include <WiFi.h>

const char* ssid = "TP-LINK_AB11";
const char* password = "12345678";
int ledPin = 2;

WiFiServer server(8000);

void setup() {
Serial.begin(115200);
pinMode(ledPin, OUTPUT);

delay(10);

Serial.print("Connecting to ");
Serial.println(ssid);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
WiFi.begin(ssid, password);
Serial.print(".");
}
Serial.println("WiFi connected.");
```

```
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

server.begin();
}

void loop(){
WiFiClient client = server.available();
if (client) {
Serial.println("New Client.");
String currentLine = "";
while (client.connected()) {
if (client.available()) {
char c = client.read();
if (c == '\n') {
if (currentLine.length() == 0) {
client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println();
client.print("<html>");
client.print("<head><title>ESP32 Server</title></head>");
client.print("<body>");
client.print("Turn <a href=\"/H\">LED on</a><br>");
client.print("Turn <a href=\"/L\">LED off</a><br>");
client.print("</body></html>");
client.println();
break;
} else {
currentLine = "";
}
} else if (c != '\r') {
currentLine += c;
}

// Проверка ссылок от клиента, может быть "GET /H" или "GET /L":
if (currentLine.endsWith("GET /H")) {
digitalWrite(ledPin, HIGH); // GET /H - включить светодиод
```

```
}
if (currentLine.endsWith("GET /L")) {
digitalWrite(ledPin, LOW); // GET /L выключить светодиод
}
}
}

// Закрыть соединение
client.stop();
}
}
```

Запускаем программу, обновляем страницу, и видим нашу “панель управления”:



Проверяем, что нажатие на одну из ссылок действительно включает или выключает светодиод.

Кстати, если у интернет-провайдера доступна услуга “статический IP”, то можно настроить перенаправление портов в роутере, и управлять светодиодом через Интернет, даже из другого города или другой страны.

Самостоятельная работа #1: изучить разные варианты форматирования текста в HTML, например, таблицы, шрифты, выравнивание. Это позволит делать более сложный и более красивый интерфейс.

Самостоятельная работа #1: подключить к плате “пищалку”, и настроить управление ею через web. Это позволит посылать сигналы родственникам или друзьям.

3.10 Выводим изображения на web-сервере

Вышеприведенный сервер конечно, работает, однако далеко не является шедевром web-дизайна. Немного улучшить его можно, добавив изображения. Однако, это не так просто, т.к. наш мини-сервер не имеет файловой системы, и просто положить нужные файлы мы не можем. Вместо этого используем так называемое base64-кодирование, позволяющее вставить изображение прямо в код страницы.

Чтобы вставить картинку в HTML, нужно.

1. Выбрать любую картинку. Возьмем картинку попроще, чтобы она не занимала много места, например изобразим светодиод:

2. Сконвертируем изображение в base64-формат, для этого можно воспользоваться любым онлайн-конвертором, например этим: https://www.base64-image.de.

Мы получим строку такого вида:

3. Создадим текстовую переменную, которая будет хранить наше изображение и скопируем в нее всю строку:

const PROGMEM char *image01 = "….5n/2Q==”;

Тег *PROGMEM* указывает компилятору, что это константа, которую можно хранить во флеш-памяти, сэкономив тем самым место для других переменных. Это особенно актуально, если картинок много.

4. Теперь мы можем использовать наше изображение с помощью

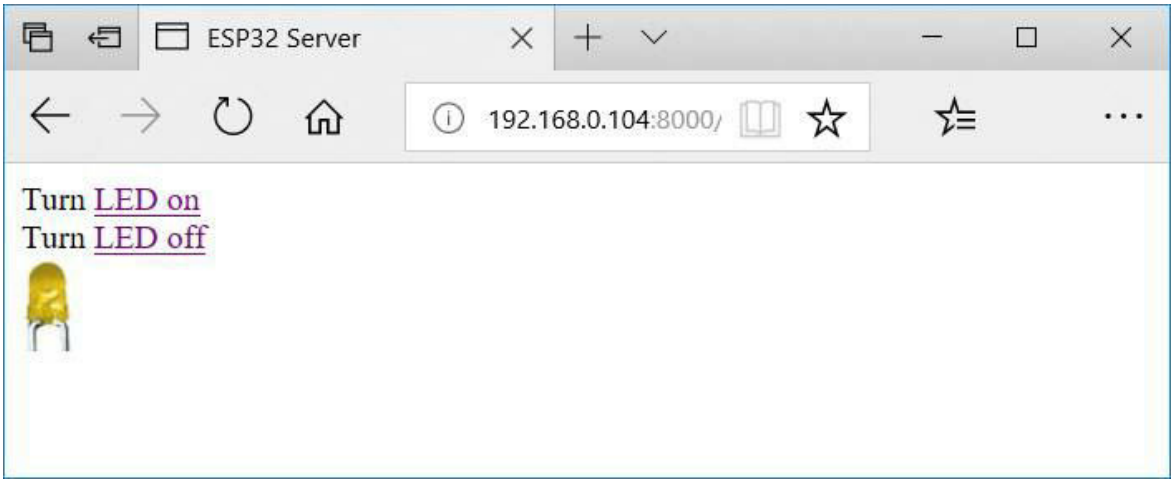
тега img следующим образом:

<img src="data:image/jpeg;base64...5n/2Q==">

Для этого добавим следующий код в следующую строку после “Turn LED off”:

client.print("<img src=\""); client.print(image01); client.print("\">");

Наш веб-сервер вряд ли получит премию года за лучший дизайн, но по крайней мере, пользователь теперь видит, что он управляет именно светодиодами:



Кстати, в реальном проекте все изображения целесообразно вынести в отдельный файл, который можно назвать например, “images.h”. Тогда в основной программе можно будет написать #include "images.h". Это позволит отделить ресурсы от кода, и сделать текст программы более читаемым.

Самостоятельная работа: вывести разные типы изображений, например кнопки, светодиоды.

3.11 Удаленный мониторинг

С помощью web-сервера можно не только включать и выключать светодиод, но и получать информацию с удаленного объекта. Например, можно узнать температуру в квартире, находясь на каникулах или в отпуске. Как нетрудно догадаться, для этого достаточно вывести

нужные нам параметры в HTML, который и отобразится на веб-странице.

К примеру, достаточно добавить такую строку в код формирования HTML:

```
if (digitalRead(buttonPin) == HIGH) {
client.print("Button state: ON<br>");
} else {
client.print("Button state: OFF<br>");
}
```

При обновлении страницы в браузере мы увидим соответствующий текст. Разумеется, это может быть не только кнопка, но и например, датчик закрывания двери или датчик освещенности (желающие могут перечитать главу 2.6 “Ввод аналоговых величин”).

Рассмотрим пример подробнее. Допустим, у нас есть данные с разных сенсоров (подробнее можно прочитать в главе 2.7-2.10). Чтобы не загромождать тестовый код, представим данные в виде уже готовых переменных, реальный код чтения желающие могут добавить самостоятельно.

```
float temperature = 22.5;
int humidity = 60;
int pressure = 1010;
bool doorClosed = true;
bool windowClosed = false;
```

Наша задача - отобразить эти данные на веб-сервере, для чего проще всего воспользоваться HTML-тегом “Table” (таблица).

Пример таблицы в HTML:

```
<table border = "1">

<tr>

<td>Row 1, Column 1</td>
```

```
<td>Row 1, Column 2</td>

</tr>


<tr>

<td>Row 2, Column 1</td>

<td>Row 2, Column 2</td>

</tr>

</table>
```

В браузере это будет выглядеть вот так:

Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

Здесь **tr** - это table row, строка таблицы, а **td** - это элемент ячейки. Таким образом, мы можем группировать данные так, как нам удобно. Добавим таблицу в наш сервер. Для экономии места будет приведена только функция loop, остальное не изменилось.

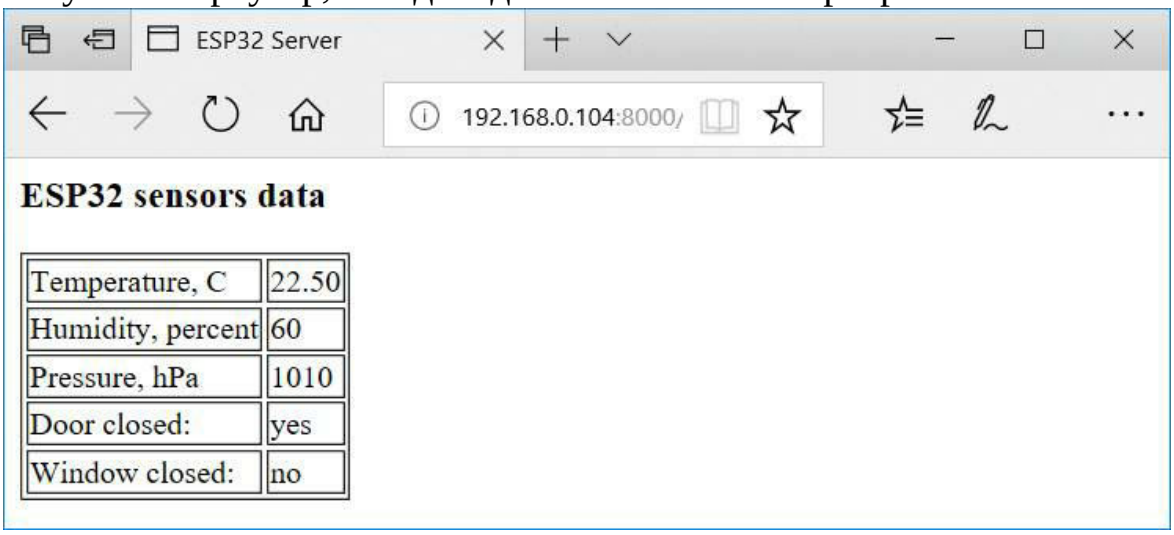
```
void loop() {
// Возобновление соединения при необходимости
while (WiFi.status() != WL_CONNECTED) {
delay(500);
WiFi.begin(ssid, password);
Serial.print(".");
}

// Данные с датчиков
float temperature = 22.5;
int humidity = 60;
int pressure = 1010;
bool doorClosed = true;
bool windowClosed = false;
```

```
// Здесь можно добавить код чтения
// ...

WiFiClient client = server.available();
if (client) {
Serial.println("New Client.");
String currentLine = "";
while (client.connected()) {
if (client.available()) {
char c = client.read();
if (c == '\n') {
if (currentLine.length() == 0) {
client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println();
client.print("<html>");
client.print("<head><title>ESP32 Server</title></head>");
client.print("<body>");
client.print("<h3>ESP32 sensors data</h3>");
client.print("<table border = \"1\">");
client.print(" <tr><td>Temperature, C</td><td>");
client.print(String(temperature)); client.print("</td></tr>");
client.print(" <tr><td>Humidity, percent</td><td>");
client.print(String(humidity)); client.print("</td></tr>");
client.print(" <tr><td>Pressure, hPa</td><td>");
client.print(String(pressure)); client.print("</td></tr>");
client.print(" <tr><td>Door closed:</td><td>");
client.print(doorClosed ? "yes" : "no"); client.print("</td></tr>");
client.print(" <tr><td>Window closed:</td><td>");
client.print(windowClosed ? "yes" : "no"); client.print("</td></tr>");
client.print("</table>");
client.print("");
client.print("</body></html>");
client.println();
break;
} else {
currentLine = "";
```

```
}
} else if (c != '\r') {
currentLine += c;
}
}
}
client.stop();
Serial.println("Client Disconnected.");
}
}
```

Как можно видеть, код довольно прост. Мы сформировали таблицу с нужными нам полями. Для вывода строк yes/no мы использовали конструкцию языка C *doorClosed ? "yes" : "no"*, она позволяет выбрать одно из двух значений в зависимости от условия.

Запускаем браузер, и видим данные с нашего сервера:



Теперь мы можем оставить устройство подключенным, и наблюдать состояние дома или квартиры удаленно, с любой точки земного шара.

Самостоятельная работа #1: подключить реальные датчики, например DS1820.

Самостоятельная работа #2: Поэкспериментировать с раскраской и стилями таблицы. Например, так можно добавить красный цвет к

ячейке:

<td bgcolor="#FF0000">Row 1, Column 1</td>

3.12 Делаем односторонний мессенджер

Плата ESP32 может быть включена круглосуточно, что легко позволяет использовать ее для приема входящих сообщений: достаточно лишь добавить в веб-сервер форму для отправки текстовых данных. Это позволит любому кто знает адрес сервера, зайти на страницу и оставить сообщение его владельцу.

На языке HTML код для отправки выглядит просто, достаточно добавить следующие строки:

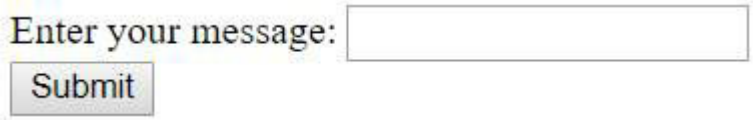
<form action="/send">

Enter your message: <input type="text" name="msg">

<input type="submit" value="Submit">

</form>

При этом страница в браузере будет иметь следующий вид:



При этом, если пользователь введет текст, например “123 456”, и нажмет кнопку “Submit”, браузером будет отправлен запрос “GET /send?msg=123+456 HTTP/1.1”, который мы можем обработать на сервере. Дальше, как говорится, дело техники - надо извлечить из строки подстроку, убрав начальные и конечные части.

Добавим код там, где проверяется начало строки:

```
if (currentLine.startsWith("GET /send?msg=") && currentLine.endsWith(" HTTP/1.1")) {
  // String looks like GET /send?msg=123+456 HTTP/1.1
  String msg = currentLine.substring(14, currentLine.length()-8);
  msg.replace("+", " ");
  Serial.println(msg);
```

}

Как можно видеть, с помощью функции substring мы взяли подстроку, пропустив 14 символов сначала (длина “GET /send?msg=”) и 8 символов с конца (длина “ HTTP/1.1”). Функция replace заменяет символы “+” на пробелы, чтобы из строки “123+456” получить “123 456”.

Добавим вместо Serial.println вывод на дисплей, который мы рассмотрели в главе 3.4.

Готовый код “мессенджера” целиком приведен ниже.

```
#include <WiFi.h>
#include "SSD1306.h"

const char* ssid = "TP-LINK_AB11";
const char* password = "12345678";
int ledPin = 2;

WiFiServer server(8000);
SSD1306 display(0x3c, 21, 22);

void setup()
{
Serial.begin(115200);
pinMode(ledPin, OUTPUT);

// Инициализация дисплея
display.init();
display.flipScreenVertically();
display.clear();
display.setFont(ArialMT_Plain_10);
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.drawString(0, 0, "App started");
display.display();

delay(10);
```

```
// Запуск WiFi
Serial.print("Connecting to ");
Serial.println(ssid);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
WiFi.begin(ssid, password);
Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();
}

void loop(){
while (WiFi.status() != WL_CONNECTED) {
delay(500);
WiFi.begin(ssid, password);
Serial.print(".");
}

WiFiClient client = server.available();
if (client) {
Serial.println("New Client.");
String currentLine = "";
while (client.connected()) {
if (client.available()) {
char c = client.read();
if (c == '\n') { // if the byte is a newline character
if (currentLine.length() == 0) {
client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println();
client.print("<html>");
client.print("<head><title>ESP32 Server</title></head>");
```

```
client.print("<body>");
client.print("<h3>ESP32 sensors data</h3>");
client.print("<form action=\"/send\">");
client.print("Enter the message: <input type=\"text\" name=\"msg\"> <br>");
client.print("<input type=\"submit\" value=\"Submit\">");
client.print("</form>");
client.print("");
client.print("</body></html>");
client.println();
break;
} else {
if (currentLine.startsWith("GET /send?msg=") &&
currentLine.endsWith(" HTTP/1.1")) {
// Строка выглядит как “GET /send?msg=123+456 HTTP/1.1”
String msg = currentLine.substring(14, currentLine.length()-8);
msg.replace("+", " ");

// Вывод строки на дисплей
display.clear();
display.setFont(ArialMT_Plain_10);
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.drawString(0, 0, msg);
display.display();
}
currentLine = "";
}
} else if (c != '\r') {
currentLine += c;
}
}
}
client.stop();
Serial.println("Client Disconnected.");
}
}
```

Теперь можно запустить программу, открывать сервер в браузере и ввести текст - он появится на экране. Это можно использовать например, для отправки сообщений своим близким, находясь в другом городе, в каникулах, отпуске или командировке.

Разумеется, данный код может быть улучшен. Например, данная библиотека не поддерживает переносы строк, так что длинная веденная строка не поместится на экран. Читатели могут сделать это самостоятельно, для этого можно воспользоваться функциями length и substring для разбивки длинной строки на строки.

Самостоятельная работа #1: добавить к плате “пищалку” или светодиод, которая будет сигнализировать о наличии нового сообщения. При желании также можно добавить на плату кнопку “Сообщение прочитано”, которой получатель подтвердит получение. Для этого достаточно добавить переменную messageIsRead, и при получении нового сообщения устанавливать ее в False, а при нажатии кнопки менять значение на True. В общем, простор для творчества тут весьма велик.

Самостоятельная работа #2: Добавить вывод нескольких сообщений. Для этого можно добавить несколько переменных, например “String msg1, msg2, msg3”, а при получении нового сообщения добавлять его в “конец очереди”:

```
msg1 = msg2;
msg2 = msg3;
msg3 = msg;
```

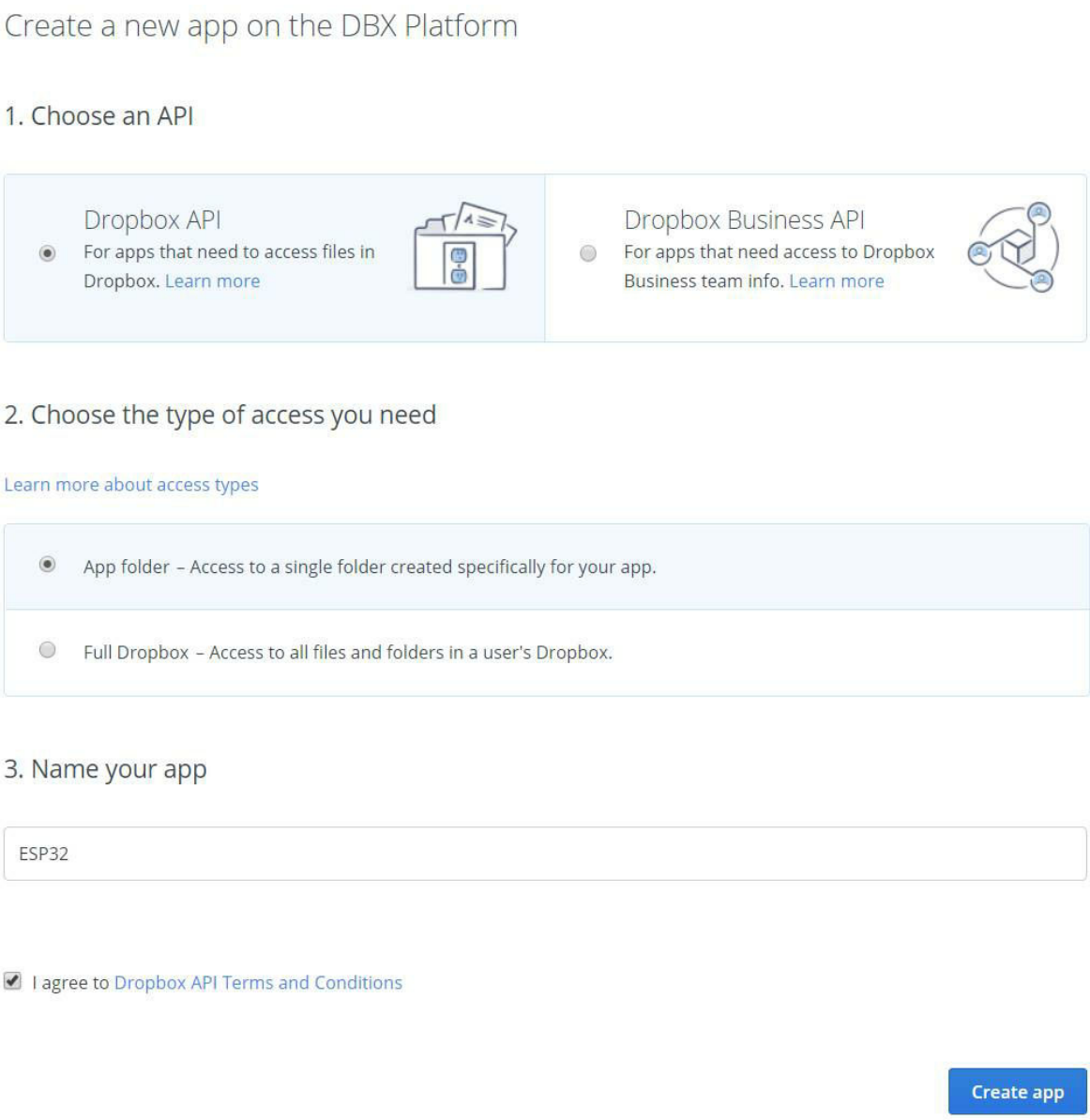
Функцию вывода на экран также придется изменить.

3.13 Отправляем данные через Dropbox

Мы уже умеем выводить с ESP32 данные через встроенный веб-сервер. К сожалению, главный его недостаток - необходимость наличия статического IP-адреса, если мы хотим получать доступ извне. Это удобно, но статический IP-адрес - это обычно платная услуга, этого хотелось бы избежать.

Есть альтернативный способ - сохранять данные в бесплатном сервисе Dropbox. Это позволит ESP32 периодически сохранять логи или какую-либо другую информацию в файлах Dropbox, и они будет автоматически синхронизироваться и появляться на домашнем компьютере или смартфоне.

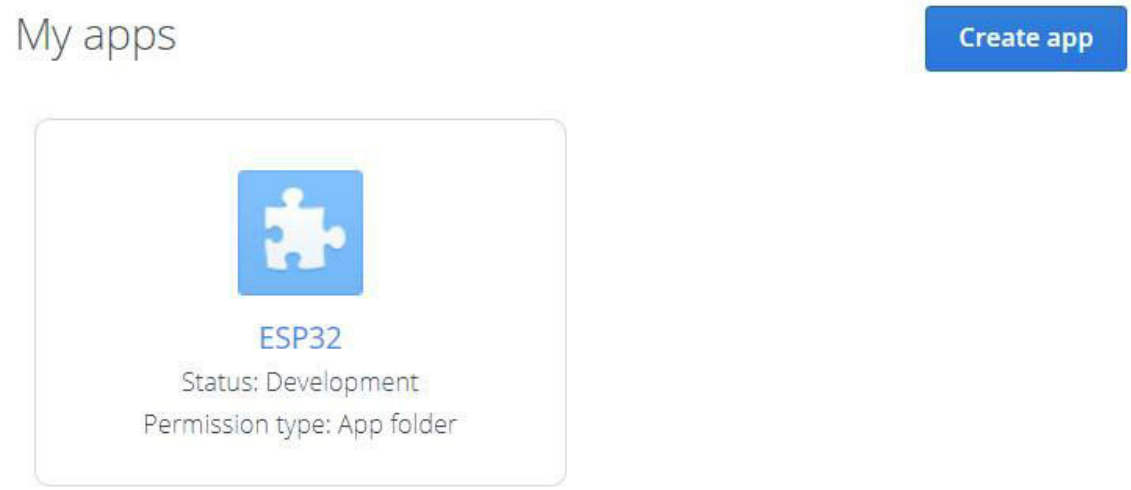
Сначала нужно создать так называемое “приложение dropbox”, сделать это можно по ссылке https://www.dropbox.com/developers/apps/create:



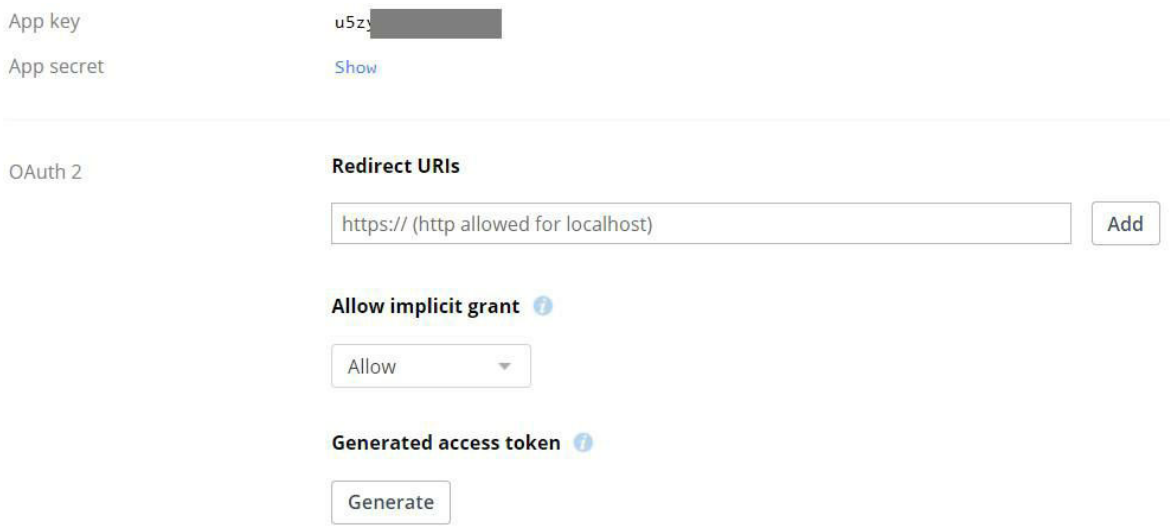
Когда приложение создано, мы можем выбрать его в разделе My apps на странице https://www.dropbox.com/developers/apps.

Приложение Dropbox имеет доступ только к определенной папке, которая будет синхронизироваться с домашним или рабочим компьютером, и будет иметь путь “Имя_пользователя \Dropbox\Приложения\Имя_приложения”.

Внутри настроек приложения есть параметр “access token”, он будет нужен нам для получения доступа.



Там же выбираем пункт Generate access token:



Нажимаем кнопку “Generate” рядом с “access token”, и получаем ключ вида V3z9NpYlRxEAAAAAAACG5cjBXXXXXXXXXXXXXX, его надо сохранить, мы будем использовать его в дальнейшем.

На этом подготовительная часть закончена. Сам сервис Dropbox имеет разнообразное API для работы с данными, посмотреть список функций можно по адресу https://www.dropbox.com/developers/documentation/http/documentation. Нам для отправки файлов нужна будет всего лишь одна функция upload.

Для того, чтобы отличать файлы друг от друга, мы будем создавать файлы по шаблону “год-месяц-день-чч-мм-сс.txt”. Dropbox требует

защищенного соединения по https, поэтому мы используем класс WiFiClientSecure.

Код программы целиком приведен ниже.

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <time.h>

const char* ssid = "TP-LINK_AB11";
const char* password = "12345678";
WiFiClientSecure client;

void uploadData(String content) {
Serial.println("Dropbox connecting...");
if (client.connect("content.dropboxapi.com", 443)) {
Serial.println("Dropbox connected");

// Сформировать имя файла по шаблону времени
time_t now = time(nullptr);
struct tm timeinfo;
gmtime_r(&now, &timeinfo);
char file_name[64] = {0};
strftime(file_name, 64, "log-%Y-%m-%d-%H-%M-%S.txt", &timeinfo);
Serial.print("Upload "); Serial.println(file_name);

// Отправка запроса
client.println("POST /2/files/upload HTTP/1.1");
client.println("Host: content.dropboxapi.com");
client.println("Authorization: Bearer XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
char dropbox_args[255] = {0};
sprintf(dropbox_args,
"{\"path\": \"/%s\", \"mode\": \"overwrite\", \"autorename\": true, \"mute\": false}", file_name);
client.print("Dropbox-API-Arg: "); client.println(dropbox_args);
client.println("Content-Type: application/octet-stream");
```

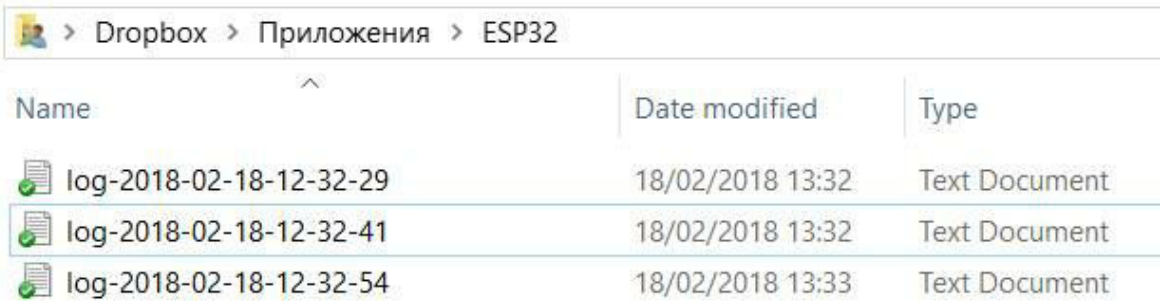
```
client.print("Content-Length: "); client.println(content.length());
client.println();
client.println(content);
delay(5000);
client.stop();
Serial.println("Disconnect");
Serial.println();
} else {
Serial.println("Error: cannot connect");
Serial.println();
}
}

void setup() {
Serial.begin(115200);
delay(10);
Serial.print("Connecting to "); Serial.println(ssid);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
WiFi.begin(ssid, password);
Serial.print(".");
}
Serial.println("");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

// Установка времени через SMTP
Serial.print("Setting time using SNTP");
configTime(8*3600, 0, "pool.ntp.org", "time.nist.gov");
time_t now = time(nullptr);
while (now < 8 * 3600 * 2) {
delay(500);
Serial.print(".");
now = time(nullptr);
}
Serial.println("done");
```

```
// Загрузка данных #1
uploadData(String("Data from ESP32 - this is a test 1"));
delay(5000);

// Загрузка данных #2
uploadData(String("Data from ESP32 - this is a test 2"));
delay(5000);

// Загрузка данных #3
uploadData(String("Data from ESP32 - this is a test 3"));
}

void loop(){
}
```

Как можно видеть, мы создали функцию uploadData, которую можно вызывать нужное число раз (имеет смысл делать это не очень часто, например раз в 5 или 30 минут). В данном примере функция вызывается только из setup(), чтобы файлы не создавались постоянно.

Вместо XXXXXXXXXXXXXXXXXXXXXXXXXXXX нужно будет вставить свой токен, полученный в панели управления https://www.dropbox.com/developers/apps.

Результат - запускаем программу, и через некоторое время видим в папке Dropbox 3 файла.



При использовании в реальном приложении, целесообразно отправлять лог лишь по каким-то редким событиям (например срабатывание датчика открывания двери). Если же нужно отправлять лог постоянно, имеет смысл накапливать данные в буфере памяти объемом несколько килобайт, и отправлять данные лишь по мере

заполнения буфера.

Важно. Это наверное должно быть и так очевидно, но лучше повторить. Функции любого сервиса, такого как Dropbox, Google, Amazon, всегда поставляются на условиях “как есть” - создатели не гарантируют их абсолютно бесперебойной работы (тем более, если речь идет о бесплатных услугах). Также эти функции могут периодически совершенствоваться и меняться, так что гарантий многолетней и бесперебойной работы устройства, увы, нет. Поэтому подходы, аналогичные описанным выше, не стоит использовать там, где от этого зависит безопасность людей или имущества.

На этом мы закончим описание работы с платой ESP32. Как можно видеть, свою цену в 8$ она отрабатывает на 110%. В следующей части мы рассмотрим работу с Linux на базе одноплатного компьютера Raspberry Pi.

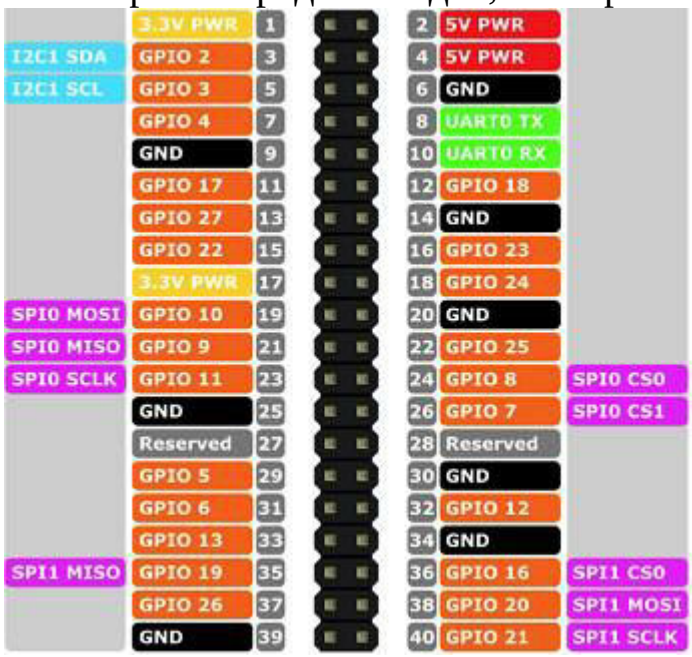
Часть 4. Осваиваем Linux: Raspberry Pi

4.1 Общее знакомство

Одноплатные компьютеры Raspberry Pi появились на рынке несколько лет назад, и сразу получили признание у пользователей. Действительно, всего за 20-30$ покупатель получал практически полноценный компьютер с операционной системой Linux. Мы уже рассматривали платы Arduino, и как нетрудно было видеть, они весьма ограничены в своих возможностях. Linux позволяет использовать практически любые современные средства разработки, от Си или Python, до многопоточных или даже распределенных вычислений.

Компьютер Raspberry Pi - это небольшая плата длиной около 10см, которая выглядит примерно так:

На плате есть Ethernet и WiFi, 4 порта USB, видеовыход HDMI, порт для подключения камеры и ряд выводов, которыми можно управлять программно.

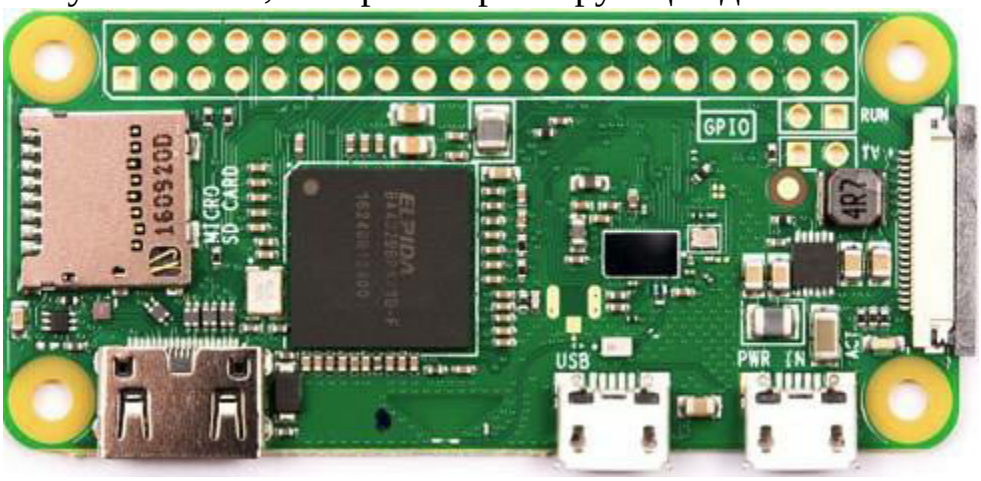


Доступен весьма большой набор различных интерфейсов (показан на картинке справа), от простого переключения пинов до уже рассмотренных нами I2C и SPI.

В этом пожалуй, одно из главных отличий - в отличие от обычных компьютеров, Raspberry Pi легко может подключаться к различным устройствам, ничуть не отличаясь в этом от Arduino. Работу с дисплеем, датчиками, сенсорами и различным оборудованием можно легко

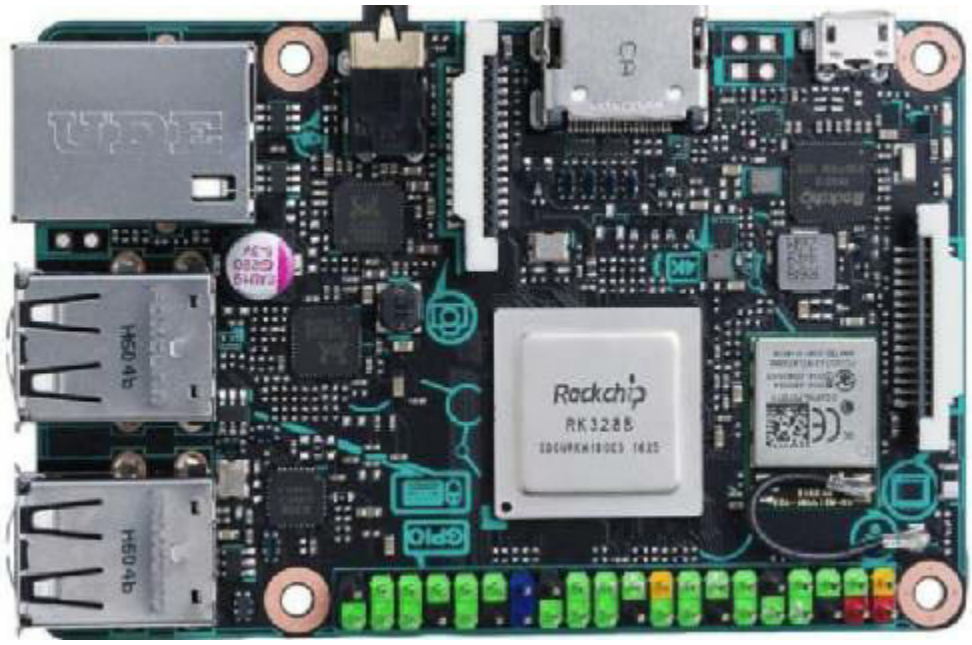
совмещать с гибкостью и настраиваемостью Linux. Например, Raspberry Pi легко подключить к домашней WiFi-сети. Или наоборот, Raspberry Pi может работать как точка доступа и не только “раздавать” интернет на другие устройства, но и например, мониторить температуру в помещении и посылать уведомления хозяевам. На Raspberry Pi вполне может работать полноценный Web или FTP-сервер, также можно подключить и USB-модем или иное оборудование. Вместо диска на Raspberry Pi используется карта MicroSD, с нее осуществляется загрузка системы.

Желающим сделать более компактное устройство, можно выбрать Raspberry Pi Zero W, которая по размеру еще вдвое меньше:



На момент написания этой главы (2018й год) последней версией является Raspberry Pi 3, более старые модели брать не рекомендуется, т.к. они имеют менее производительный процессор и не имеют WiFi.

Помимо оригинальной Raspberry Pi, на рынке появилось много клонов, плат такого же размера. Некоторые отличаются ценой, некоторые производительностью. Среди вполне удачных, можно отметить Asus Tinkerboard от известного производителя материнских

плат Asus. 

Asus Tinkerboard вдвое дороже Raspberry Pi, но имеет более мощный процессор, больше памяти, и может пригодиться для более ресурсоемких задач.

Еще одним плюсом популярности Raspberry Pi является большое количество аксессуаров - корпусов, экранов, реле и прочих дополнительных устройств. Это позволяет вполне гибко подобрать ресурсы под интересующие задачи. К примеру, можно докупить вот такой экран, позволяющий сделать из Raspberry Pi домашний мини-сервер, метеостанцию или пульт управления “умным домом”:

Важно заметить и то, что в отличие от обычного компьютера, Raspberry Pi потребляет мало энергии и не имеет вентиляторов, так что работает совершенно бесшумно.

Кстати о питании, для работы Raspberry Pi нужен блок питания с MicroUSB и максимальным током 2-2.5A, так что подойдет обычное зарядное устройство от любого современного смартфона.

Для первого включения нам понадобятся: Raspberry Pi, клавиатура, HDMI-кабель для подключения к монитору и кард-ридер для записи операционной системы Linux на карту памяти.

4.2 Настройка системы

Вначале нужно подготовить Raspberry Pi к первому использованию. В принципе, ничего не мешает использовать Raspberry Pi как обычный компьютер - подключить клавиатуру и мышь, и пользоваться. Но во-

первых, это все же будет довольно-таки медленный компьютер, а во-вторых, это не так интересно, мы пойдем настоящим путем Linux, используя Raspberry Pi в режиме удаленного терминала через консоль.

Шаг-1. Подготовка карты памяти

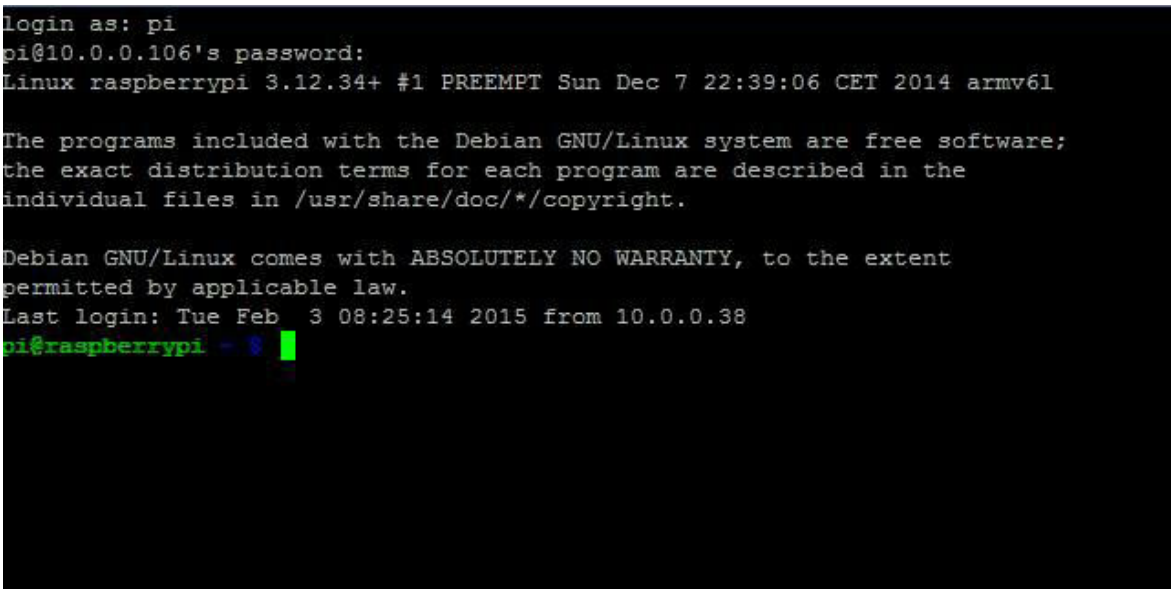
Скачиваем дистрибутив Raspbian Stretch Lite со страницы на официальном сайте - https://www.raspberrypi.org/downloads/raspbian/. Мы выберем Lite-версию, т.к. она занимает меньше места, и для наших задач будет более эффективна.

Пока дистрибутив скачивается, устанавливаем программу Win32DiskImager. Вставляем карту памяти в кард-ридер, и с помощью win32diskimager записываем сохраненный дистрибутив на карту. По завершении записи карту памяти можно вставить обратно в Raspberry Pi.

Шаг-2. Настройка Raspbian

Подключаем Raspberry Pi к монитору и клавиатуре (мы будем использовать удаленный доступ, но один раз сделать это все-таки придется). Запускаем Raspberry Pi, дожидаемся приглашения ввода имени и пароля, вводим **pi** и **raspberry**.

Ура, мы в Linux! Никакого графического интерфейса, и ничего лишнего. Как настоящие хакеры (шутка) мы будем только вводить команды через Linux-терминал. Мышь можно не подключать, нам она не пригодится.



Вводим нашу первую Linux-команду: **sudo raspi-config**. Raspi-config это программа для настройки системы, а sudo говорит о том, что команда будет запущена с правами администратора, значит она может менять системные файлы и настройки.

Запустится текстовое окно конфигурации системы, в котором нужно будет выбрать следующие параметры:

- Активировать SSH (Advanced options - SSH). Это позволит нам иметь удаленный доступ к Raspberry Pi.
- Активировать I2C (кто забыл что это такое, может перечитать главу 2.8)
- Активировать SPI (через этот интерфейс могут подключаться некоторые дисплеи).
- При желании, можно настроить страну и часовой пояс (Internationalization options).

По завершении ввода настроек, перезагружаемся, если система не предложит сделать это сама, можно набрать команду **sudo reboot**.

Шаг-3. Подключаемся к Интернет

Следующим шагом мы должны подключить Raspberry Pi к Интернет, без этого на нее не удастся установить никаких программ. Здесь есть 2 варианта.

Самый простой способ - просто подключить Raspberry Pi к сетевому кабелю. При этом ничего настраивать не нужно, все заработает само. Однако, с точки зрения удобства использования платы, это не очень удобный вариант, лишний провод на столе будет только мешаться.

Способ сложнее, зато удобнее - подключиться к домашней сети WiFi. Если мы вдруг забыли имя своей WiFi-сети, можно набрать команду **sudo iwlist wlan0 scan**. Затем мы открываем текстовый редактор командой **sudo nano /etc/wpa_supplicant/wpa_supplicant.conf**. Nano - это текстовый редактор, а /etc/wpa_supplicant/wpa_supplicant.conf это путь к файлу, который мы будем редактировать.

Запустив редактор, добавляем в этот файл строки с именем сети и паролем:

network={

ssid="TP-LINK_AB11"

psk="12345678"

}

Для завершения редактирования нажимаем Ctrl+X, и на вопрос, сохранить ли файл, нажимаем Y (yes). Перезагружаемся.

В завершении, можно проверить наличие интернета с помощью команды **ping www.google.com**. Если все нормально, мы должны увидеть время ответа от сервера. Раз уж у нас появился интернет, запустим обновление системы, введем 2 команды **sudo apt-get update** и затем **sudo apt-get upgrade**. Процесс может занять минут 5-10, после чего Raspberry Pi можно выключить (ввести команду **sudo halt**). Теперь монитор и клавиатуру можно отключить.

Шаг-4. Настраиваем удаленный доступ

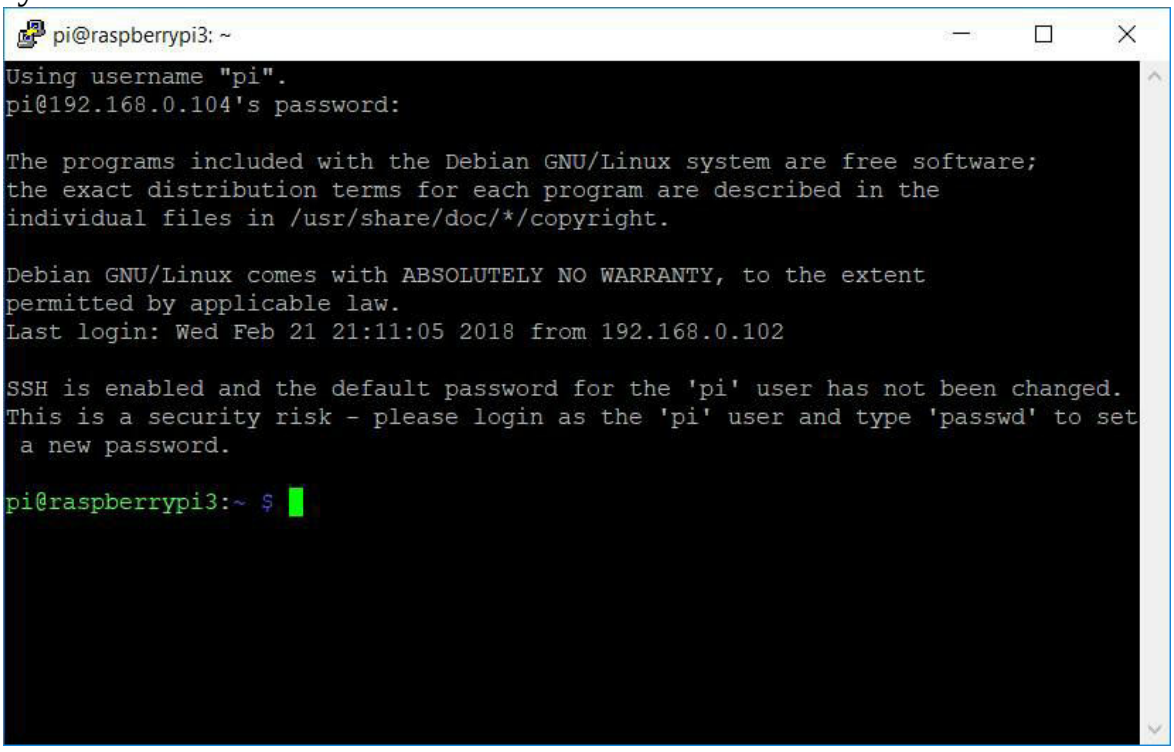
Теперь мы можем разместить Raspberry Pi там где нам удобно, и включить ее. Первым делом нужно узнать IP-адрес платы, его можно посмотреть в настройках домашнего маршрутизатора, либо в командной

строке Windows (Win+R - cmd) ввести команду **arp -a**. В результате мы должны узнать IP-адрес, в моем случае это 192.168.0.104.

Если в качестве настольного компьютера мы используем OSX или Linux, то для коннекта к Raspberry Pi нам достаточно ввести команду **ssh pi@192.168.0.104**. Если же мы используем Windows, то сначала нужно скачать SSH-клиент, в качестве которого удобно использовать putty. Устанавливаем putty в какую-либо папку, затем вводим команду:

putty.exe pi@192.168.0.104

Если все было сделано правильно, появится окно ввода, примерно такое же, какое мы видели, когда подключали Raspberry Pi к монитору. Только теперь все те же действия мы можем делать удаленно. После ввода пароля raspberry, мы попадаем в уже знакомую нам командную строку:



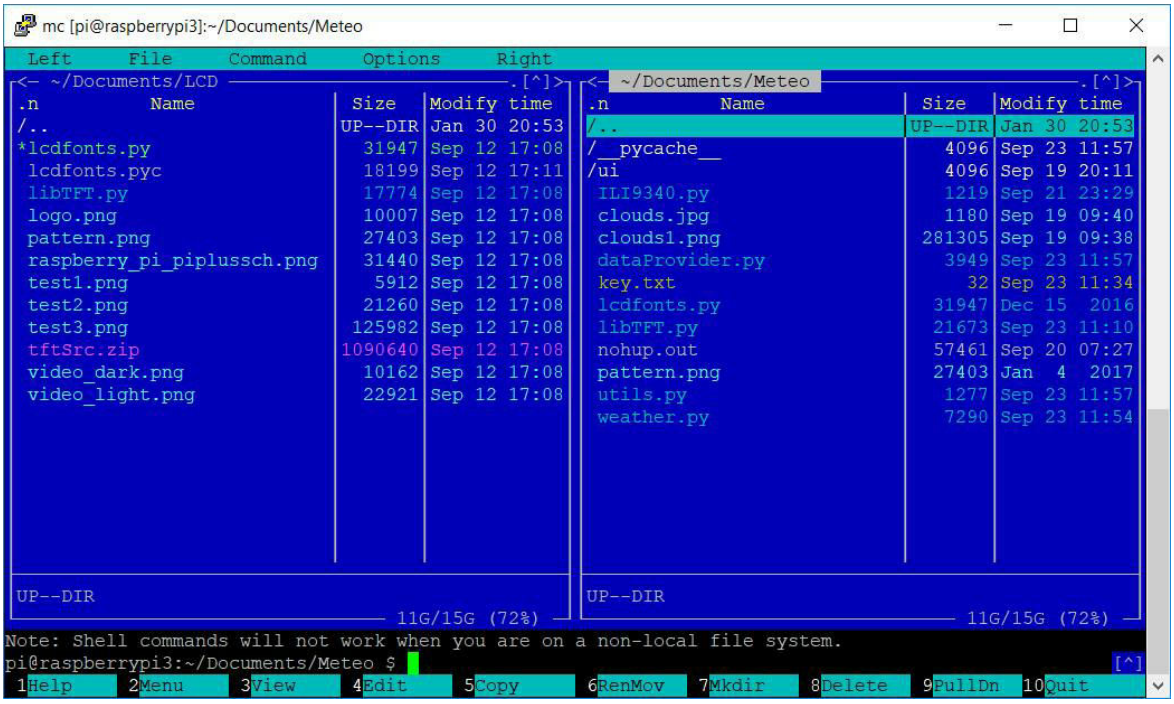
На этом настройка завершена, мы можем использовать Raspberry Pi.

Кстати, вполне удобным является создание *статического IP-адреса* для Raspberry Pi в домашней сети (например 192.168.0.111), при

этом он не будет меняться, что удобнее. Желающие могут найти соответствующие онлайн-руководства по настройке.

4.2 Основные команды Linux

Мы настроили удаленный доступ, и сначала нужно немного освоиться в среде Linux. Для начала целесообразно поставить файловый менеджер mc, для чего ввести команду **sudo apt-get install mc**. Apt-get это “менеджер программ”, а mc это название программы, которую мы устанавливаем. После завершения установки командой **mc** можно открыть файловый менеджер, напоминающий FAR или (если кто помнит) Norton Commander.



Здесь удобно то, что доступны как операции с файлами, так и доступ к консоли, просмотреть вывод которой можно с помощью клавиш Ctrl+O.

Также можно поэкспериментировать с командами **ls** (список файлов), **mkdir** (создание папки) и **cd** (смена текущей папки).

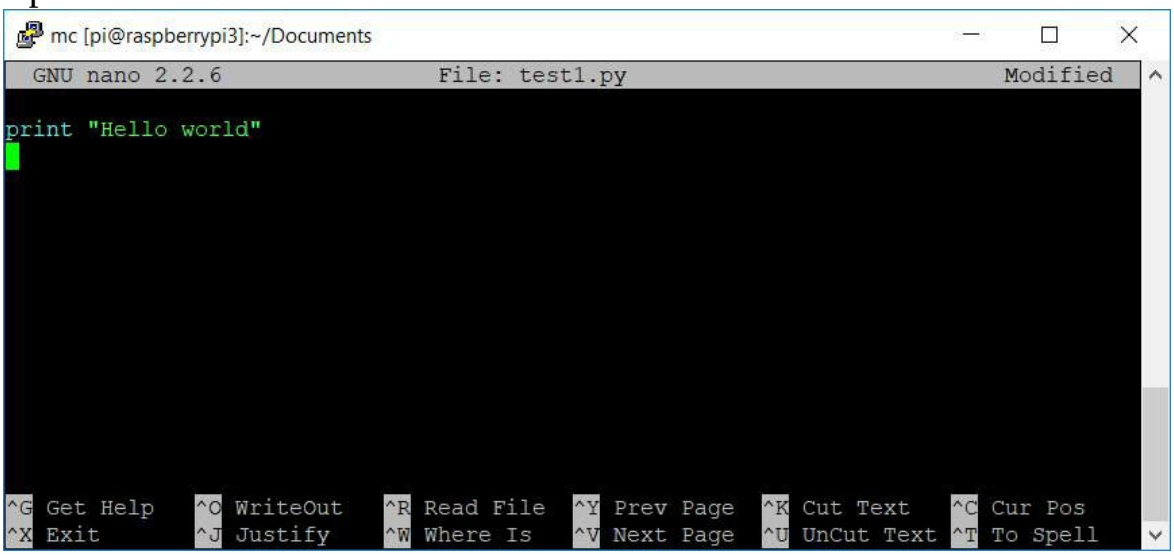
Чтобы создать текстовый файл, например текст программы, можно воспользоваться уже известным нам редактором nano. Например, чтобы создать программу на языке Python, наберем **nano file1.py**. В

открывшемся окне введем текст программы (его можно просто скопировать через буфер обмена):

print “Hello world”

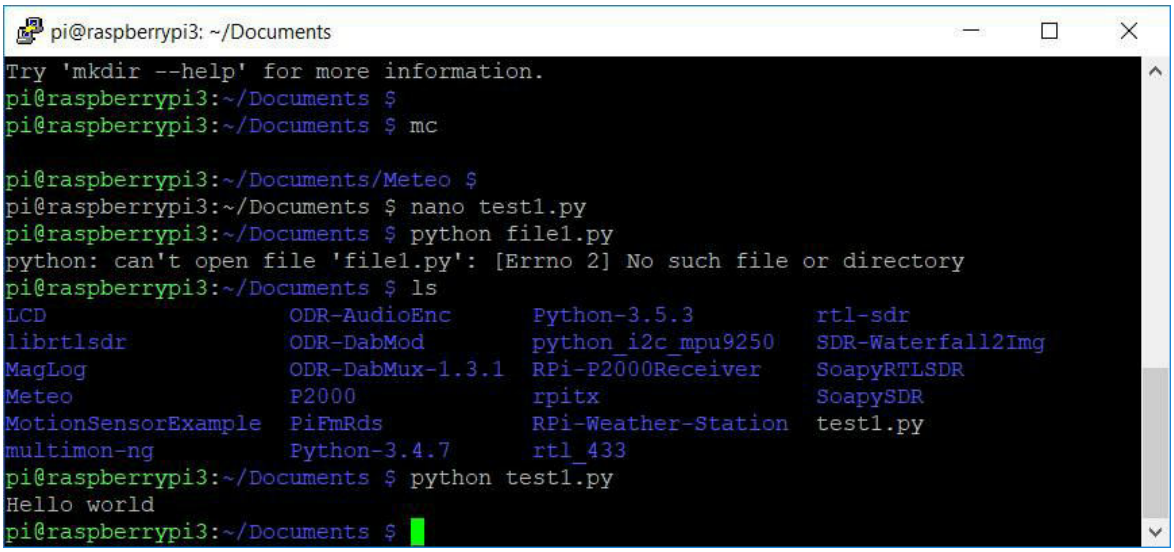
Завершить редактирование можно нажатием Ctrl+X, затем нужно нажать Y для подтверждения сохранения файла.

Сам редактор во время редактирования должен выглядеть примерно так:



Теперь можно выполнить программу, введя команду python test1.py.

Если все было сделано правильно, мы увидим результат выполнения:



Ура! Мы запустили нашу первую программу в среде Linux.

Установим также модуль для дополнительных библиотек языка Python, он нам пригодится в дальнейшем: **sudo apt-get install python-pip**. Также установим программу git, с помощью которой удобно скачивать исходные тексты программ: **sudo apt-get install git**.

Для копирования программ с “большого” компьютера можно использовать программу WinSCP, которая позволяет иметь прямой доступ к файловой системе Raspberry Pi. Это позволит редактировать программы на компьютере, затем копировать их в папку на Raspberry Pi, что достаточно удобно.

4.3. Основы языка Python

Код для Arduino мы писали на Си, здесь же мы будем использовать язык Python. Он удобен своей компактностью, отсутствием необходимости компиляции программ, и наличием большого количества дополнительных библиотек.

Для запуска Python-программы необходимо:

- Создать файл с текстом программы, например с помощью редактора nano:

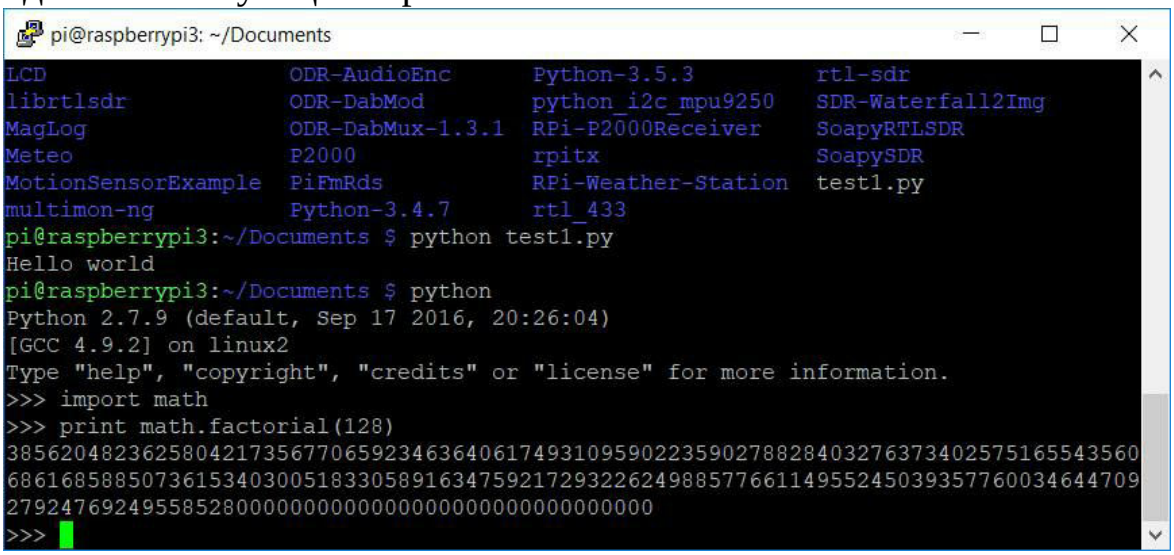
nano test1.py

- Запустить программу на исполнение:

sudo python test1.py

Мы используем sudo т.к. наши программы будут работать с портами ввода-вывода, без этого они доступны не будут.

Кстати, чтобы запустить короткую программу на Python, ее необязательно сохранять в файл, код можно ввести непосредственно в интерпретатор. Для этого достаточно запустить python и просто ввести подряд соответствующие строки:



Закрыть интерпретатор Python можно нажатием Ctrl+D.

Рассмотрим простые примеры использования языка Python.

Объявление и вывод переменных

В Python достаточно ввести имя и значение.

```
x = 3
y = 10
print("x=", x)
print(x+y)
```

В отличие от языка C++, тип переменной будет определен автоматически, указывать его не нужно. Его можно при необходимости узнать, введя print (type(x)).

Циклы

В отличие от того же C++ или Java, циклы в Python задаются

отступами, что после других языков программирования может быть непривычным. Часть кода, находящаяся внутри цикла, будет выполнена заданное количество раз.

Вывод чисел от 1 до 9:
```
for p in range(1,10):
print (p)
```

Вывод чисел от 1 до 9 с шагом 2:
```
for p in range(1,10,2):
print (p)
```

Для сравнения, вот такой код без отступов работать не будет, и это важно помнить, например при копировании кода из этой книги:
```
for p in range(1,10):
print (p)
```

Кстати, в **Python 2.7** функция range возвращает объект “список”, и соответственно выделяет память заданного размера. Для большинства примеров в книге это не критично, но если нужно написать код типа *for p in range(1,10000000)*, то **range** следует заменить на **xrange**, в противном случае программа вполне может занять гигабайт памяти даже на простом с виду цикле. Для Python 3.xx это не требуется.

Массивы

Массив это линейный набор чисел, с которыми удобно выполнять однотипные операции, например вычисление суммы или среднего арифметического.

Объявляем массив чисел:
```
values = [1,2,3,5,10,15,20]
```

Добавляем элемент в массив:
```
values.append(7)
```

Выводим массив на экран:
```
print (values)
```

Выводим элементы массива построчно:

```
for p in values:
print (p)
```

Это же можно сделать с помощью индексов (нумерация элементов массива начинается с 0):

```
for i in range(0,len(values)):
print (values[i])
```

Можно создать массив определенного размера, заполненный определенными числами. Создадим массив из 100 элементов, заполненный нулями.

```
values = [0.0] * 100
print (values)
```

Есть немного более сложный, но и более гибкий вариант создания массива. Создадим массив из 100 элементов, заполненный нулями:

```
values = [0.0 for i in range(100)]
```

Создадим массив, заполненный числами 0,1,..,99:

```
values = [i for i in range(100)]
```

Создадим массив, заполненный квадратами чисел:

```
values = [i*i for i in range(100)]
```

Создать двухмерный массив в Python также несложно:

```
matrix4x4 = [ [0,0,0,0], [0,0,0,0], [0,0,0,0], [0,0,0,0]]
matrix4x4[0][0] = 1
print (matrix4x4)
```

Аналогично вышеописанному способу, можно создать 2х-мерный массив 100x100:

```
values100x100 = [ [0.0 for j in range(100)] for i in range(100)]
```

Арифметические операции

Сложение, умножение, деление:

```
x1 = 3
x2 = (2*x1*x1 + 10*x1 + 7)/x1
```

Возведение в степень:

```
x3 = x1**10
print (x1,x2,x3)
```

Переменную также можно увеличить или уменьшить:

```
x1 += 1
x1 -= 10
print (x1)
```

Остаток от деления:

```
x2 = x1 % 6
print (x2)
```

Условия в Python кстати, задаются отступами, аналогично циклам:

```
x1 = 123
print (x1)
if x1 % 2 == 0:
print("x1 even number")
else:
print("x1 odd number")
```

Подсчитаем сумму элементов массива:

```
values = [1,2,3,5,10,15,20]
s = 0
for p in values:
s += p
print(s)
```

Также для этого можно воспользоваться встроенной функцией sum:

```
values = [1,2,3,5,10,15,20]
print(sum(values))
```

Пожалуй, этого не хватит чтобы устроиться на работу программистом, но вполне достаточно для понимания примеров в книге. Теперь вернемся к Raspberry Pi.

Примечание

У языка Python есть одна неприятная особенность - по умолчанию он “не знает” русской кодировки, ее нужно принудительно указывать в начале файла:

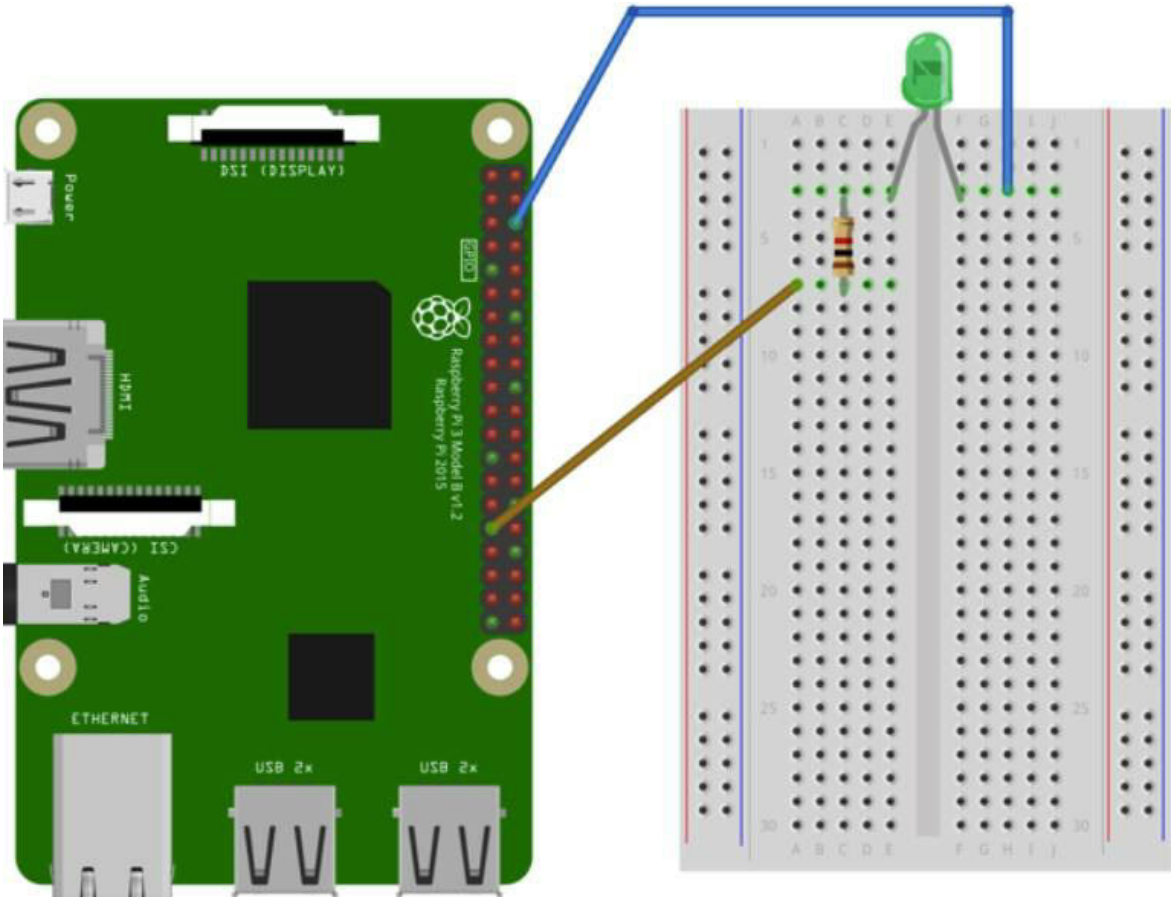
```
# -*- coding: utf-8 -*-
print "Русский текст"
```

Без этого интерпретатор выдаст ошибку. Поэтому чтобы не загромождать код и не писать # -*- coding: utf-8 -*- каждый раз в каждом примере, в дальнейших программах все комментарии будут писаться на английском.

4.4. GPIO: порты ввода-вывода

Знакомство с любой новой платой, мы начнем с уже традиционного - подключим светодиод. Тем более, что как можно увидеть, принципы по сути ничем не отличаются от Arduino. В этом, как и в любых следующих проектах, будем считать что Raspberry Pi включена и готова к работе, а доступ осуществляется через putty.

Наша первая схема будет совсем простой:



Принцип, как можно видеть, точно такой же - мы подключаем светодиод через токоограничительный резистор (кто забыл как это работает, может перечитать главу 1.4). Когда резистор подключен, включим Raspberry Pi и напишем код.

Если мы делаем такой проект первый раз, то сначала нужно поставить необходимые библиотеки. Введем команду **sudo apt-get install python-rpi.gpio**. GPIO - это general-purpose input/output, или “общие порты ввода вывода”, как раз то что нам нужно.

Запустим редактор для создания или редактирования файла, введем команду:

nano led_blink.py

Скопируем и вставим туда следующий код:

import RPi.GPIO as GPIO

```
import time

LedPin = 31 # GPIO6

# Setup

GPIO.setmode(GPIO.BOARD)

GPIO.setup(LedPin, GPIO.OUT)

# Loop
try:

while True:

    GPIO.output(LedPin, GPIO.HIGH) # led on

    time.sleep(1.0)

    GPIO.output(LedPin, GPIO.LOW) # led off

    time.sleep(1.0)

except KeyboardInterrupt: # Ctrl+C stop

pass

# Close
GPIO.cleanup()
```

Нажмем Ctrl+X для выхода из редактора, на вопрос сохранения файла нажмем Y (yes).

Теперь можно запустить программу, и мы увидим мигающий светодиод:

sudo python led_blink.py

Для завершения работы нажмем Ctrl+C, и мы вернемся обратно в

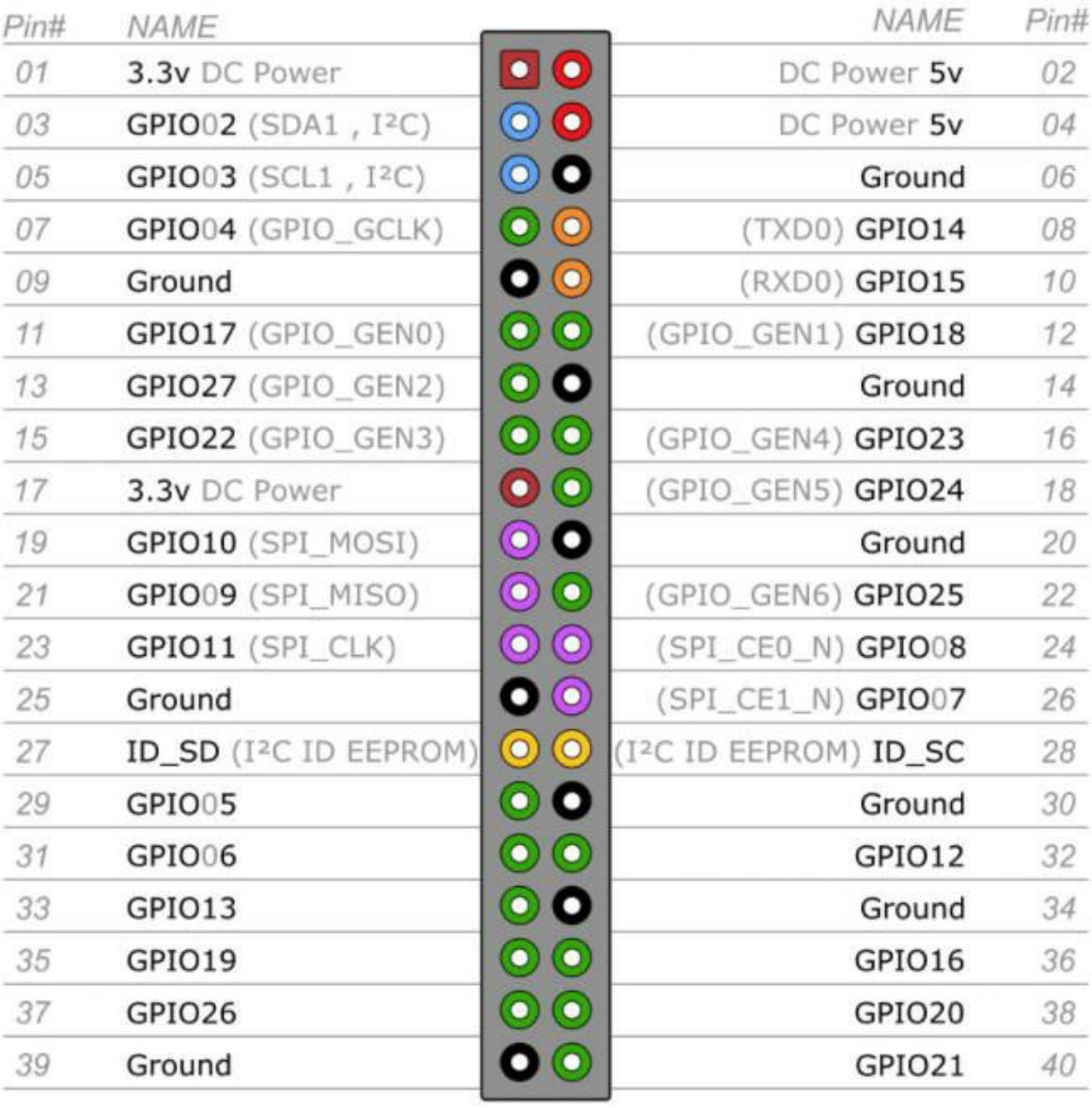
командную строку.

Ура, программа работает! Разберем код программы подробнее.

- Строка *import RPi.GPIO as GPIO* указывает интерпретатору, что надо загрузить модуль RPi.GPIO и использовать его под названием GPIO (писать каждый раз RPi.GPIO было бы слишком длинно). Команда import time таким же способом загружает модуль time. Строка LedPin = 31 создает переменную с нужным номером вывода, тут все просто.

- Строка GPIO.setmode(GPIO.BOARD) указывает, какую нумерацию выводов мы будем использовать. Есть два варианта: **GPIO.BOARD** “говорит” о том, что будет использоваться сквозная нумерация выводов на плате. В этом случае пин имеет номер 31 (см. картинку на следующей странице). Второй вариант, **GPIO.BCM** задает нумерацию в номерах GPIO, в этом случае наш вывод 31 имел бы номер 6, т.к. он называется GPIO06. В принципе, практически все равно, какой способ использовать - вся эта путаница пошла от старых плат Raspberry Pi, которые имели меньшее число выводов. Главное, придерживаться какого-то одного стандарта и не путать одни номера с другими. Разумеется, если номер будет неправильный, светодиод не загорится.

На картинке показаны оба варианта нумерации выводов Raspberry Pi:



Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

- Строки GPIO.output(LedPin, GPIO.HIGH) и GPIO.output(LedPin, GPIO.LOW) устанавливают нужный логический уровень “1” или “0”, что соответствует 0 или 3.3В, также как на плате ESP32. Функция time.sleep(1.0) делает паузу в 1 секунду.

- try..except - это незнакомая нам до этого конструкция, называемая обработчиком исключений. Цикл while True выполняется бесконечное число раз, но если пользователь нажмет Ctrl+C, то система пошлет программе исключение KeyboardInterrupt, в котором мы можем выполнить какой-то код. В данном случае кода нет, строка pass просто указывает на то, что нужно перейти на следующую строку.

- В завершении работы программы, функция GPIO.cleanup() освобождает ресурсы GPIO, и программа закрывается.

Важно

В отличие от Arduino, наша программа не будет стартовать сама при старте Raspberry Pi. Если нужно, можно добавить это самостоятельно, отредактировав файл автозапуска rc.local. Введем команду: **sudo nano /etc/rc.local**

Внутрь файла rc.local нужно будет добавить команду запуска:
sudo python /home/pi/led_blink.py &

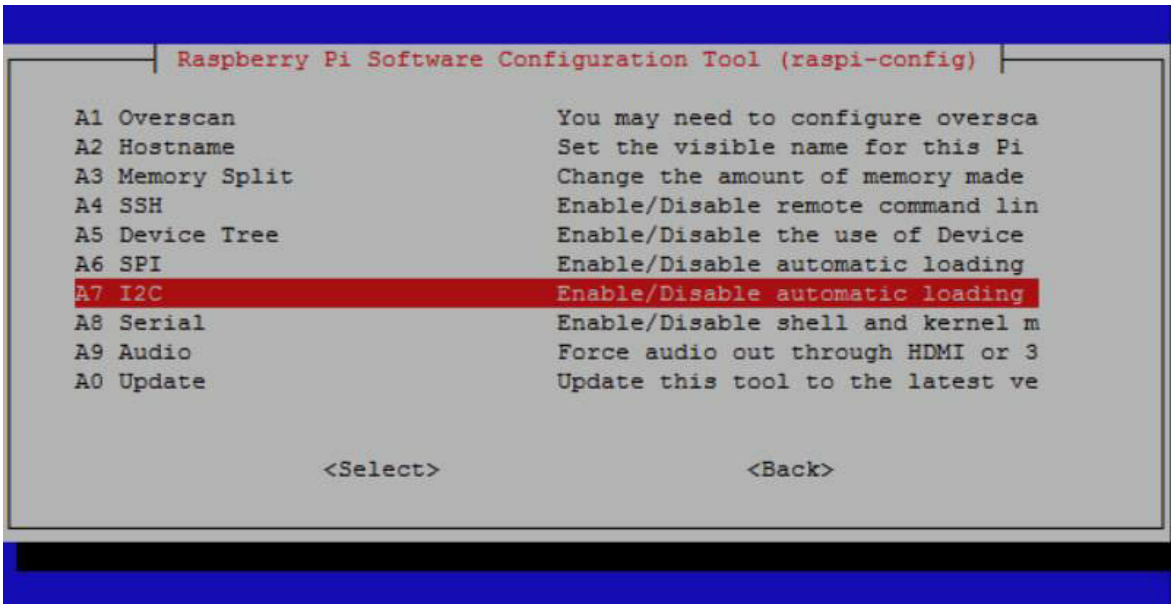
Команду sudo мы уже знаем, знак & говорит о том, что программа будет запущена в фоновом режиме. /home/pi/led_blink.py - это путь к программе, он может быть и другим. Кстати, узнать путь к текущему каталогу можно, введя команду **pwd**.

Сохранив файл, перезагрузим Paspberry Pi - мы увидим мигающий светодиод.

4.5. Используем I2C

Достаточно большое количество разнообразных устройств (экраны, датчики, АЦП) можно подключить к шине I2C. Но перед ее первым использованием, необходимо настроить Raspberry Pi.

Шаг-1. Запустить **sudo raspi-config** и активировать I2C, как показано на рисунке (это достаточно сделать только один раз):



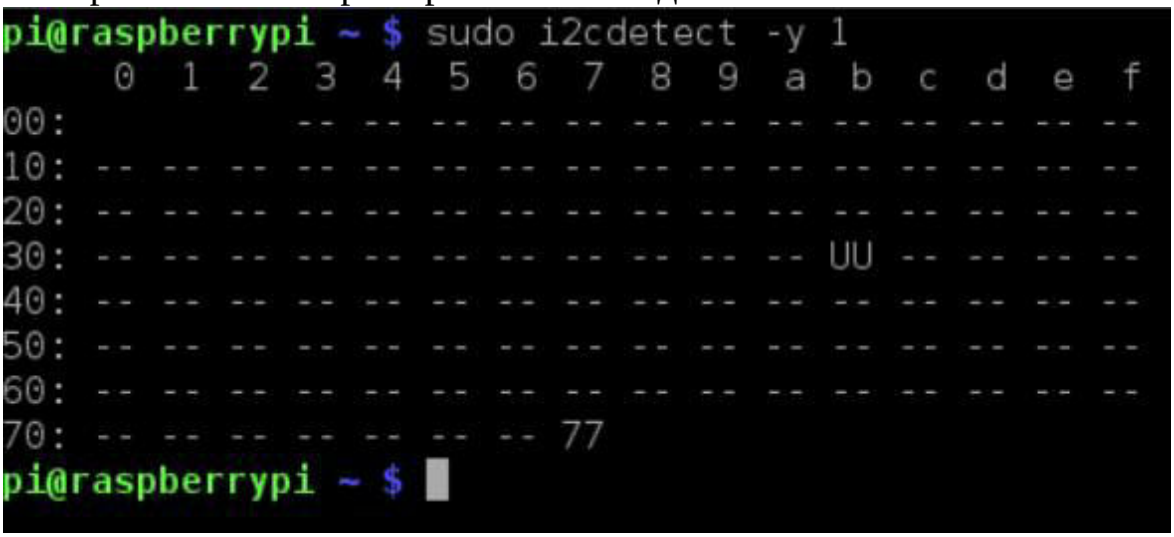
После изменения настроек следует перезагрузить Raspberry Pi.

Шаг-2. Поставить программу i2c-tools, введя команду: **sudo apt-get install i2c-tools**

Теперь можно ввести команду и посмотреть доступные i2c-устройства:

sudo i2cdetect -y 1

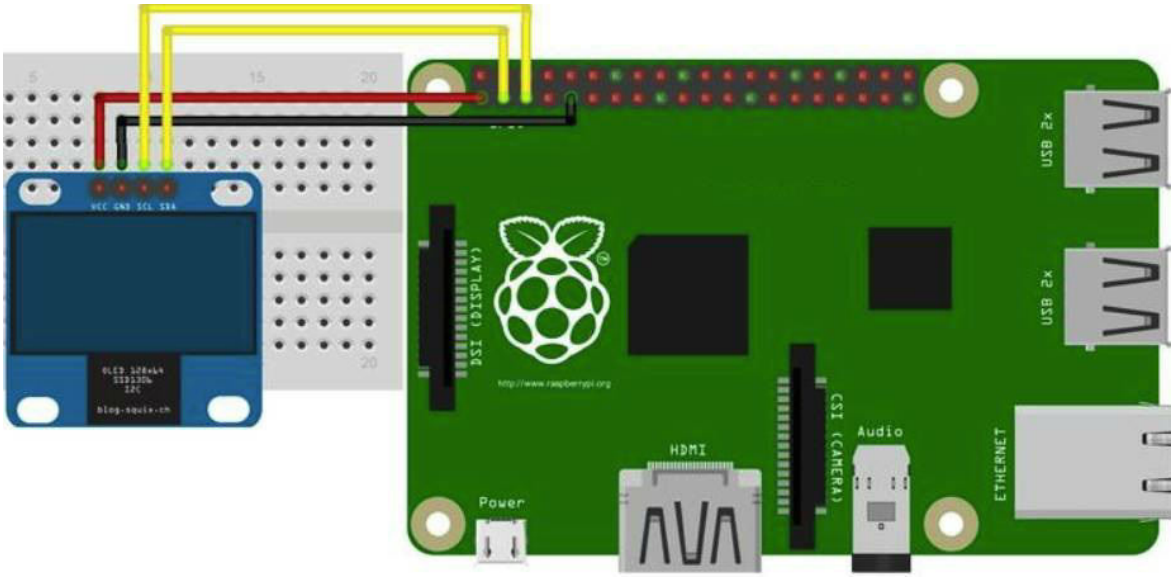
Отобразится окно примерно такого вида:



Если ничего не было подключено, список будет пуст, иначе мы увидим номера подключенных устройств.

Шаг-3. Подключение устройства

Шина I2C использует 2 провода для передачи данных (пины “3” и “5” на Raspberry Pi), еще 2 будут нужны для “0” и питания. Пример для подключения I2C-дисплея показан на рисунке. Обратим внимание, что для питания устройства используется вывод “3.3В”.



После того как устройство подключено, следует еще раз набрать команду **i2cdetect -y 1** и убедиться, что его адрес виден в списке. Без этого пытаться запустить какой-либо код бессмысленно.

4.6. Подключаем OLED-дисплей

Наличие дисплея безусловно актуально для многих устройств, так что и Raspberry Pi мы также не оставим без внимания. Тем более, что дисплей подойдет тот же самый, который мы уже рассматривали.

Подключим дисплей, как показано на предыдущей странице. Убедимся, что дисплей “виден” в системе, должен отображаться код 3C:

```
pi@raspberrypi3:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi3:~ $
```

Теперь установим библиотеки, необходимые для работы дисплея. Введем следующие команды:

sudo apt-get install python-imaging python-smbus pillow
sudo apt-get install git

git clone https://github.com/adafruit/Adafruit_Python_SSD1306.git

На этом шаге будет создан каталог Adafruit_Python_SSD1306. Зайдем в него и запустим установку библиотек для Python:

cd Adafruit_Python_SSD1306

sudo python setup.py install

Все готово. Проверить работу дисплея можно, запустив программу stats.py, находящуюся в папке examples:

sudo python examples/stats.py

Если все было сделано правильно, дисплей покажет информацию о системе:

Рассмотрим пример использования такого дисплея:

```
from PIL import Image, ImageDraw, ImageFont

import Adafruit_SSD1306
import time


disp = Adafruit_SSD1306.SSD1306_128_64(rst=None, i2c_address=0x3C)

# disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST, i2c_address=0x3C)

disp.begin()


# Load default font.

font = ImageFont.load_default()

# Create blank image for drawing. '1' - monochrome 1-bit color

image = Image.new('1', (disp.width, disp.height))

draw = ImageDraw.Draw(image)

value = 0

while True:
# Clear screen
```

```
draw.rectangle((0,0,width,height), outline=0, fill=0)

draw.text((x, top), "Hello world", font=font, fill=255)

draw.text((x, top+8), "Value "+ str(value), font=font, fill=255)

disp.image(image)

disp.display()

time.sleep(0.5)
```

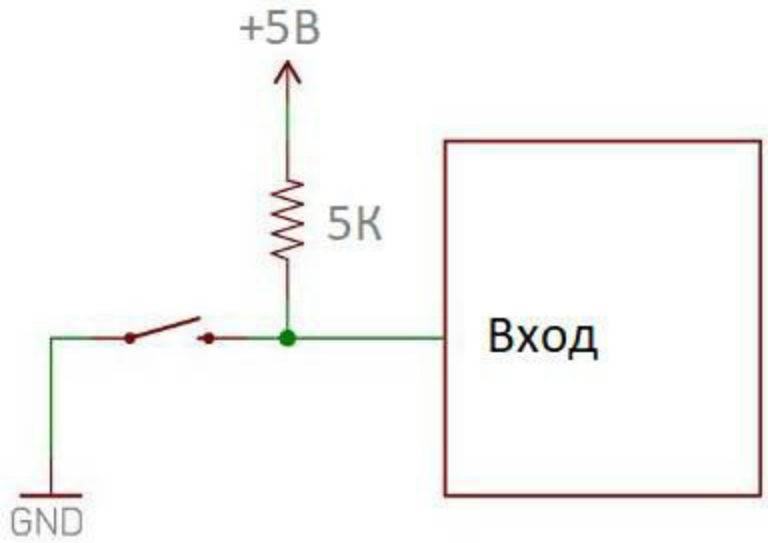
Разберем код подробнее. Директивы *import* подключают необходимые библиотеки. Затем мы создаем в памяти объект disp класса Adafruit_SSD1306.SSD1306_128_64 (либо 128_32, если мы имеем такой дисплей). i2c_address - это адрес дисплея. Функция *ImageFont.load_default*, как нетрудно догадаться из названия, загружает в память шрифт. Затем, с помощью функции *Image.new* мы создаем в памяти монохромное изображение-буфер, в которое будем рисовать данные. Подготовленное изображение выводится с помощью метода *disp.image()*, метод *disp.display()* обновляет картинку на экране. Такая технология называется “двойной буфер” (double buffer), изображение сначала формируется в памяти, и только затем обновляется, это позволяет избежать мерцания.

Самостоятельная работа: изучить код примера examples/stats.py, в нем можно найти полезные функции, например для получения IP-адреса устройства. Исходный текст stats.py можно найти на github.

4.7. Подключаем кнопки

Раз уж мы подключили дисплей, остался последний шаг для создания автономного устройства на базе Raspberry Pi - добавить поддержку кнопок. Тем более, что для тех кто читал главу 3.2 про порты ввода-вывода на ESP32, все будет просто - принцип остается тот же.

Напомним общую идею, которая обсуждалась еще в первой части про Arduino - специальный резистор “подтягивает” напряжение до шины питания, а кнопка при необходимости замыкает его на землю:



На Raspberry Pi все то же самое, только резистор уже встроенный, и ставить его отдельно не надо. Ну и напряжение не 5В, а 3.3В, но для пользователя это ни на что не влияет. В итоге, схема получается проще некуда: просто подключаем кнопку, которая соединяет вывод с “землей”:

А вот код, в отличие от Arduino, будет немного отличаться.
Точнее говоря, есть два способа.

Способ-1. Код в стиле Arduino.

```
import RPi.GPIO as GPIO

import time

btn_pin = 24 # Button to GPIO24


# Setup
```

```
GPIO.setmode(GPIO.BCM)

GPIO.setup(btn_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

try:
# Loop

while True:

button_state = GPIO.input(btn_pin)

if button_state == False:

print 'Button Pressed...'

time.sleep(0.2)

except:

GPIO.cleanup()
```

Как можно видеть, здесь состояние кнопок читается в бесконечном цикле while True, и если значение кнопки равно логическому нулю, мы делаем какое-то действие.

Чем плох этот код? Он нормально работает на Arduino, т.к. там используется простой процессор, умеющий выполнять только одну задачу. Но на Raspberry Pi используется многоядерный процессор и полноценная многозадачная операционная система Linux, и постоянно работающий бесконечный цикл будет лишь зря забирать ресурсы процессора.

Гораздо правильнее “подписаться” на системные уведомления об изменении состояния кнопки, тогда код автоматически вызовется когда нужно.

Способ-2. Код с использованием событий (events).

```
import RPi.GPIO as GPIO
```

```
import time

btn_pin = 24 # Button to GPIO24

# Setup

GPIO.setmode(GPIO.BCM)

GPIO.setup(btn_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def btn_callback(channel):

if GPIO.input(btn_pin) != False:

print "Rising edge detected"

else:

print "Falling edge detected"


GPIO.add_event_detect(btn_pin, GPIO.RISING, callback=btn_callback,
bouncetime=300)

# Loop

try:
while True:

time.sleep(1.0)

except:

GPIO.cleanup()
```

Разница значительна - здесь в основном цикле программы не делается ничего, кроме вызова time.sleep, такой код будет гораздо меньше нагружать процессор. Когда кнопка будет нажата, сработает

функция btn_callback. Функция здесь только выводит сообщение, в реальном коде может обновляться содержимое дисплея или выполняться какое-то другое действие. Как можно видеть из кода, можно подписаться как на срабатывание кнопки (GPIO.RISING), так и на ее отпускание.

Разумеется, в реальном проекте может быть более одной кнопки, достаточно будет лишь добавить соответствующие вызовы GPIO.setup и GPIO.add_event_detect.

Самостоятельная работа: изучить код вывода текста на дисплей из предыдущей части, и объединив его с чтением состояния 4х кнопок, сделать систему меню. Простор для творчества тут большой, можно например использовать кнопки “вверх”, “вниз”, “OK” и “Cancel”.

4.8. Выходим в Web: запросы к серверу

Мы уже рассматривали получение количества друзей в “Контакте” в главе 3.6. Сделаем тот же самый запрос на языке Python. Здесь мы видим всю красоту и мощность данного языка, позволяющего делать вполне сложные вещи очень коротко и лаконично. Весь код умещается в 6 строк:

```
import json, urllib2

url = "https://api.vk.com/method/friends.get?user_id=29744451"
try:
data = json.load(urllib2.urlopen(url))
cnt = len(data['response'])
print cnt
except:
pass
```

Кстати, одна из полезных особенностей Python - его кроссплатформенность. Этот код можно выполнить и на Raspberry Pi, и на Windows, и на OSX, он будет работать одинаково.

Практически, подобные фрагменты удобно вынести в отдельную функцию, это делает код читабельнее и понятнее.

```
def getFriends(friendID):
url = "https://api.vk.com/method/friends.get?user_id=" + str(friendID)
try:
data = json.load(urllib2.urlopen(url))
cnt = len(data['response'])
return cnt
except:
return -1
```

Тогда вызвать его можно так:

```
n = getFriends(29744451)
print “Number of friends:”, n
```

Аналогично с числом подписчиков канала Youtube, все это можно записать в виде функции:

```
apiKey = "AIzaSyC26UJw-ubU6NXXXXXXXXXXXXXXXXXX"

def getSubscribersCount(channelID):
url = "https://www.googleapis.com/youtube/v3/channels?id=" + channelID + "&part=statistics&key=" + apiKey
try:
data = json.load(urllib2.urlopen(url))
return data["items"][0]["statistics"]["subscriberCount"]
except:
return -1
```

Может возникнуть вопрос, как мы получили строчку data["items"][0]["statistics"]["subscriberCount"]? Это просто, если посмотреть на json в браузере. Выглядит он напомним, так:

```
{

"kind": "youtube#channelListResponse",

"items": [

{
```

"kind": "youtube#channel",

"etag": "\"_gJQceDMxJ8gP-8T2HLXUoURK8c/Ea_ipJwsnrECB064UURA_RcRR0Y\"",

"id": "UCzz4CoEgSgWNs9ZAvRMhW2A",

"statistics": {

"viewCount": "30872448",

"commentCount": "313",

"subscriberCount": "258797",

"hiddenSubscriberCount": false,

"videoCount": "303"

}

}

]

}

Фигурными скобками обозначается элемент dictionary, доступ к элементам которого осуществляется через квадратные скобки - data["items"] вернет раздел items. Дальше смотрим на сам items: квадратные скобки в json обозначают массив, таким образом "items": [{... }] это массив из одного элемента типа dictionary, обратиться к нему можно как data["items"][0]. Дальше все аналогично, внутри есть еще один вложенный dictionary statistics, внутри него есть поле subscriberCount. Кстати, что такое dictionary, хорошо видно по полю statistics. Это так называемый “словарь” из парных элементов вида “имя”-”значение”, например { "viewCount": "30872448", "commentCount": "313", "subscriberCount": "258797" }.

Формат json сейчас является де-факто стандартом в web, за счет возможности описывать сложные иерархические структуры данных. Примеры разных файлов json желающие могут посмотреть на http://json.org/example.html. Поэтому при желании получать данные с различных веб-сервисов, нужно будет понимать как этот формат устроен.

Кстати, для отладки удобно выводить json прямо из Python, для этого достаточно стандартной функции print:

```
data = json.load(urllib2.urlopen(url))
print data["items"]
print data["items"][0]
print data["items"]["statistics"]
```

Так легко видеть, не ошиблись ли мы где-нибудь с полями данных.

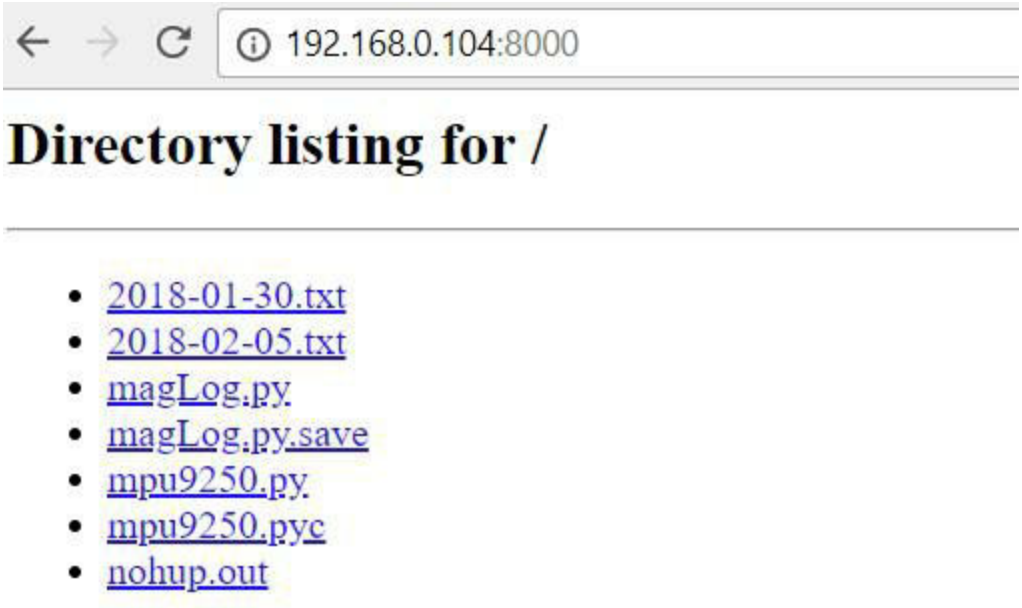
4.9. Выходим в Web: запускаем свой сервер

Теперь рассмотрим обратную задачу - запустим свой сервер, который пользователь сможет открыть в браузере. Разумеется, Raspberry Pi имеет полноценную ОС Linux, и здесь можно запустить хоть Apache, хоть FTP. Но это не так интересно - мы запустим собственный сервер, который напишем на языке Python.

Для начала - самое простое, чего в 95% случаев будет вполне достаточно. Если мы хотим просто дать пользователю доступ к текущей папке, то в консоли достаточно ввести команду:

```
python -m SimpleHTTPServer 8000
```

Пользователь сможет зайти из браузера и увидеть (и скачать) любой файл:



Если нужно что-то посложнее, например управлять каким-либо устройством, придется написать свой код.

Рассмотрим минимальный пример работающего web-сервера на Python. Для примера, будем управлять светодиодом, как и в случае с ESP32. Но в отличие от ESP32, Raspberry Pi полноценно может работать с файлами, так что хранить HTML прямо в коде не требуется, просто сохраним файл как index.html.

```
<html>
<head><title>Raspberry Pi server</title></head>
<body>
Turn <a href="/H">LED ON</a><br>
Turn <a href="/L">LED OFF</a><br>
</body>
</html>
```

Сам код сервера будет весьма прост. Естественно, это минимальный пример, реальный веб-сервер может поддерживать практически все современные технологии - POST и GET-запросы, Javascript и пр. Полет фантазии тут неограничен. Нам же будет достаточно следующего кода:

```
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
```

```
class Server(BaseHTTPRequestHandler):

def do_GET(self):
self.send_response(200)
self.send_header('Content-type', 'text/html')
self.end_headers()
with open('index.html', 'r') as myfile:
data = myfile.read()
self.wfile.write(data)
if "/H" in self.path:
print "LED ON"
if "/L" in self.path:
print "LED OFF"

if __name__ == "__main__":
server_address = ('', 8000)
httpd = HTTPServer(server_address, Server)
print 'Starting httpd...'
try:
httpd.serve_forever()
except:
pass
print 'Done'
```

Самостоятельная работа: дописать код включения и выключения светодиода вместо функции print "LED ON" и print "LED OFF". Перед вызовом сервера serve_forever также нужно будет дописать код инициализации GPIO.

4.10. Дистанционное управление со смартфона

С помощью портов ввода-вывода мы уже можем управлять внешними устройствами, например платой управления двигателем. Мы также можем запустить собственный сервер, как было показано выше. Добавим в наш сервер возможность реакции на нажатие и отпускание кнопок, это позволит использовать смартфон как полноценный пульт дистанционного управления. Такой интерфейс сейчас весьма популярен

и используется, например, для управления мини-дронами.

Наш интерфейс, впрочем, будет попроще - сделаем страницу с всего двумя кнопками LEFT и RIGHT, при нажатии на которые на Raspberry Pi будут генерироваться соответствующие события.

Основная работа здесь будет в модификации файла index.html - в него мы добавим код, срабатывающий при нажатии и отпускании кнопок. Это позволит например, управлять радиоуправляемым танком, если поставить на него Raspberry Pi.

Для создания кнопки в HTML есть тег button, а для подписки на события нажатия и отпускания мы будем использовать Javascript, который поддерживается всеми браузерами.

Обновленный код index.html приведен ниже.

```
<html>
<head><title>Raspberry Pi server</title></head>
<body>
<button id="btn_left" type="button" style="height:60px;width:60px">Left</button>
<button id="btn_right" type="button" style="height:60px;width:60px">Right</button>

<script>
document.getElementById("btn_left").onmousedown = function() {
```

```
mouseDownL() };
    document.getElementById("btn_left").onmouseup = function() {
mouseUpL() };
    document.getElementById("btn_left").onmouseleave = function() {
mouseUpL() };
    document.getElementById("btn_right").onmousedown = function() {
mouseDownR() };
    document.getElementById("btn_right").onmouseup = function() {
mouseUpR() };
    document.getElementById("btn_right").onmouseleave = function() {
mouseUpR() };

    function httpGetAsync(theUrl, callback) {
    console.log('httpGetAsync: ' + theUrl);
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.onreadystatechange = function() {
    if (xmlHttp.readyState == 4 && xmlHttp.status == 200) {
    // callback(xmlHttp.responseText);
    console.log(xmlHttp.responseText);
    }
    }
    xmlHttp.open("GET", theUrl, true); // true for asynchronous
    xmlHttp.send(null);
    }

    function mouseDownL() {
    httpGetAsync(window.location.href + "?BtnLeftDown", null);
    }

    function mouseUpL() {
    httpGetAsync(window.location.href + "?BtnLeftUp", null);
    }

    function mouseDownR() {
    httpGetAsync(window.location.href + "?BtnRightDown", null);
    }

    function mouseUpR() {
```

```
httpGetAsync(window.location.href + "?BtnRightUp", null);
}
</script>

</body>
</html>
```

Разберем код подробнее. Как можно видеть, мы создали 2 кнопки с идентификаторами *btn_left* и *btn_right*. Далее мы добавили к каждой кнопке обработчики событий *onmousedown*, *onmouseup* и *onmouseleave* (на смартфоне нет мыши, но события нажатия и отпускания пальца называются также). Они будут вызываться когда пользователь нажимает или отпускает кнопку. Внутри каждого события вызывается соответствующая функция для левой (Left) или правой (Right) кнопки, например mouseDownL(). Наша задача - уведомить сервер, посылкой ему соответствующего GET-запроса. Для этого написана функция httpGetAsync. В обработчике каждой из кнопок мы вызываем разные функции, например BtnRightDown, BtnLeftDown. Переменная window.location.href хранит адрес сервера, таким образом при нажатии кнопки LEFT на сервер отправляется запрос http://192.168.0.124:8000/?BtnLeftDown.

На этом клиентская часть закончена. Сохраним этот файл как index.html. Сам сервер тоже необходимо дописать, чтобы он корректно обрабатывал новые запросы. Код сервера показан ниже, как можно видеть, изменения в нем минимальны.

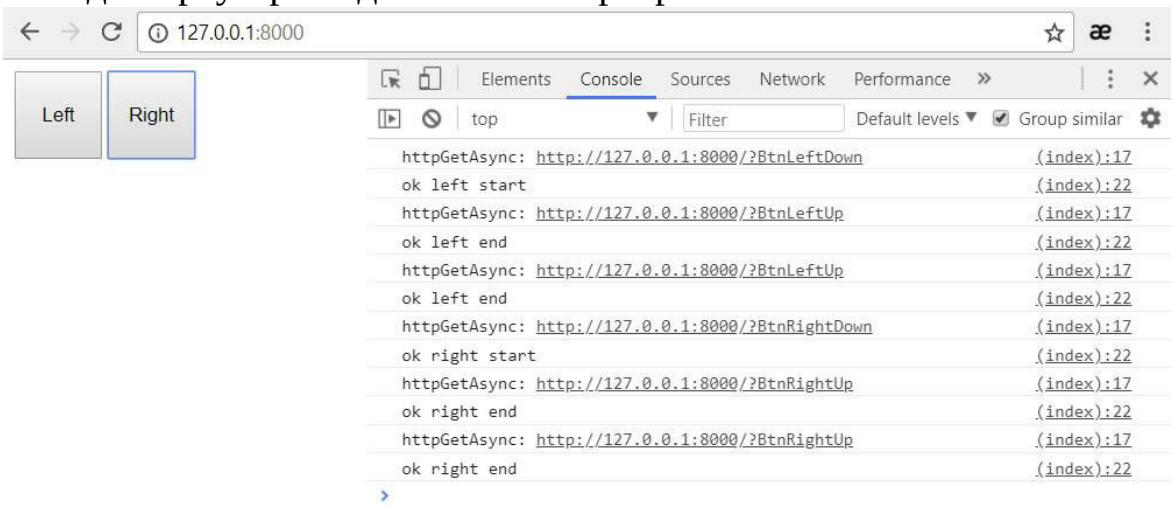
```
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer

class Server(BaseHTTPRequestHandler):
def do_GET(self):
print self.path
if "?BtnLeftDown" in self.path:
self.send_response(200)
self.send_header('Content-type', 'text/plain')
self.end_headers()
self.wfile.write("ok left start")
return
```

```
if "?BtnLeftUp" in self.path:
self.send_response(200)
self.send_header('Content-type', 'text/plain')
self.end_headers()
self.wfile.write("ok left end")
return
if "?BtnRightDown" in self.path:
self.send_response(200)
self.send_header('Content-type', 'text/plain')
self.end_headers()
self.wfile.write("ok right start")
return
if "?BtnRightUp" in self.path:
self.send_response(200)
self.send_header('Content-type', 'text/plain')
self.end_headers()
self.wfile.write("ok right end")
return
with open('index.html', 'r') as htmlfile:
self.send_response(200)
self.send_header('Content-type', 'text/html')
self.end_headers()
data = htmlfile.read()
self.wfile.write(data)

if __name__ == "__main__":
# Start server
server_address = ('', 8000)
httpd = HTTPServer(server_address, Server)
print 'Starting httpd...'
try:
httpd.serve_forever()
except:
pass
```

Мы проверяем, есть ли в запросе нужная строка (например BtnLeftDown), и если есть, то выполняем соответствующее действие. В

данном случае мы отправляем клиенту подтверждение вида “ok left start”, это удобно для отладки. В реальности нужно будет добавить дополнительный код, например для активации соответствующих портов ввода-вывода.

Теперь все готово, сохраняем наш файл и запускаем сервер. Кстати, для тестирования можно использовать тот же компьютер, используя ip-адрес http://127.0.0.1:8000. Запускаем браузер, нажимаем кнопки, и в окне отладки браузера видим ответы сервера.



Теперь мы можем управлять чем угодно с помощью длинных или коротких нажатий кнопок. Можно кстати, приобрести на ebay специальную платформу с гусеницами и моторами, и управлять ею удаленно.

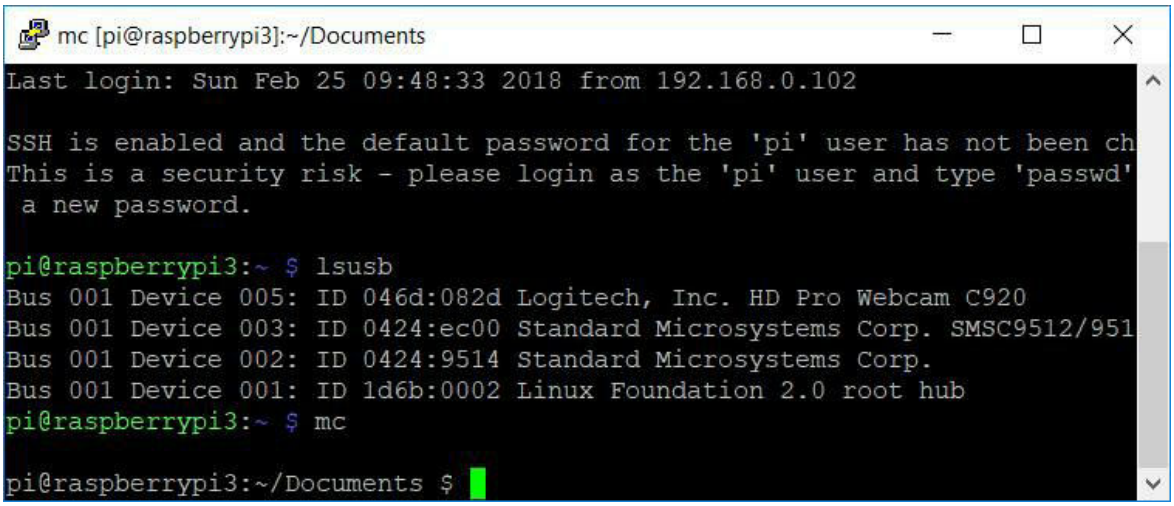
Самостоятельная работа: добавить код еще для двух кнопок “вперед”, “назад”, это позволит сделать вполне полноценное управление.

4.11. Подключаем веб-камеру

Мы не будем рассматривать подключение к Raspberry Pi всех датчиков, которые рассматривались в главе про Arduino - по сути принцип тот же, и сделать “по аналогии” совсем не сложно. На крайний случай, подключение например, того же DS18S20 к Raspberry Pi можно найти в Гугле за 5 минут. Мы пойдем дальше, и рассмотрим то, что на Arduino или ESP32 сделать нельзя. Например, подключим к Raspberry Pi веб-камеру и посмотрим, что с ней можно сделать. Вычислительная мощность Raspberry Pi весьма неплоха, и позволяет решать интересные задачи, в том числе и по обработке изображений. Тем более, что камера вполне пригодится рассмотренном ранее на радиоуправляемом танке, если мы захотим сделать его на базе Raspberry Pi.

Рекомендуется использовать веб-камеры, которые уже были проверены на совместимость с Raspberry Pi, список таких камер можно найти здесь: https://elinux.org/RPi_USB_Webcams. Я использовал Logitech C920, которая в этом списке как раз есть.

Первым делом необходимо подключить камеру и убедиться, что она видна в системе. Наберем команду **lsusb**, мы должны увидеть список типа такого.



Если камера видна в системе, можно двигаться дальше. Самый простой способ получить изображение с камеры - установить программу fscam. Наберем команду **sudo apt-get install fswebcam**.

Получим картинку с камеры: введем команду **fswebcam image.jpg**. В текущем каталоге будет создано изображение с одноименным названием. В моем случае изображение оказалось пересвеченным, камера не успела настроиться на освещение. Для исправления этого можно добавить параметр задержки: **fswebcam --delay=1 image.jpg**.

Чтобы узнать список доступных разрешений, можно ввести команду **v4l2-ctl --list-formats-ext**. Для Logitech C920 максимальное разрешение составляет 1920x1080, что вполне неплохо. С fswebcam почему-то оно не заработало, максимальное разрешение составило 1280x720, что тоже неплохо. Для того чтобы указать разрешение, надо ввести команду: **fswebcam -r 1280x720 image.jpg**. Несложно сделать и так, чтобы изображения делались постоянно: **fswebcam -r 1280x720 --loop 10 image.jpg**. С помощью параметра loop задается время в секундах, в нашем примере оно равно 10 - каждые 10 секунд изображение будет перезаписываться на новое.

Несложно записать и видео, для этого нужно поставить программу

avconv: **sudo apt-get install libav-tools**. Теперь можно записать видео, введя команду **avconv -f video4linux2 -s 640x480 -i /dev/video0 video0.avi** (для примера выбрано разрешение 640x480). Если в системе присутствует несколько камер, вместо **/dev/video0** возможно придется указать другое устройство (список можно посмотреть, введя команду **ls /dev/video***).

Самостоятельная работа: поэкспериментировать с различными настройками фото и видео, оценить требуемый объем данных для хранения или передачи таких данных.

4.12. Запускаем видеонаблюдение

Используя материал предыдущей страницы, несложно сделать на базе Raspberry Pi свой сервер видеонаблюдения. Для этого достаточно добавить на наш веб-сервер код отдачи изображения, и параллельно запустить fswebcam, который будет делать снимки. Правда, это будет не совсем “видео”, а скорее “фото” наблюдение, но тем не менее, это все равно интересно.

Шаг 1. Мы уже умеем запустить веб-сервер, который будет показывать страницу index.html из текущей папки. Поменяем файл index.html, чтобы он имел следующий вид:

```
<html>
<head><title>Raspberry Pi server</title></head>
<body>
Camera image:<BR/>
<img src="image.jpg"/>
</body>
</html>
```

Как нетрудно видеть, мы добавили туда элемент img, чтобы показывалась наша картинка.

Шаг 2. Сделаем чтобы программа fswebcam запускалась при старте, и закрывалась при выходе. Это не сложно сделать средствами Python:

```
cmd = 'fswebcam -r 640x480 --loop 10 image.jpg'
```

```
photoThread = subprocess.Popen("exec " + cmd, stdout=subprocess.PIPE, shell=True)
…
os.kill(photoThread.pid, signal.SIGINT)
```

Мы создаем поток photoThread командой subprocess.Popen, а в конце работы приложения посылаем потоку команду закрытия.

Шаг 3. Добавим в наш веб-сервер возможность “отдавать” файлы jpeg. Для этого нужно добавить новый тип Content-type = 'image/jpg', чтобы браузер “знал” что передаваемые данные это картинка jpg.

Готовый код показан ниже.

```
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
import os, subprocess, signal

class Server(BaseHTTPRequestHandler):
def do_GET(self):
if "image.jpg" in self.path:
try:
self.send_response(200)
self.send_header('Content-type', 'image/jpg')
self.end_headers()
f = open('image.jpg', 'rb')
self.wfile.write(f.read())
f.close()
except:
pass
return
with open('index.html', 'r') as imgfile:
self.send_response(200)
self.send_header('Content-type', 'text/html')
self.end_headers()
data = imgfile.read()
self.wfile.write(data)

if __name__ == "__main__":
# Start photos shooting
```

```
cmd = 'fswebcam -r 640x480 --loop 10 image.jpg'
photoThread = subprocess.Popen("exec " + cmd, stdout=subprocess.PIPE, shell=True)

# Start server
server_address = ('', 8000)
httpd = HTTPServer(server_address, Server)
print 'Starting httpd...'
try:
httpd.serve_forever()
except:
pass

# Close photo shooting
os.kill(photoThread.pid, signal.SIGINT)
```

Как можно видеть, все вполне просто. Если в строке запроса присутствует “image.jpg”, мы загружаем картинку, иначе “index.html”. Изменить частоту обновления изображений можно, поменяв параметр loop, там же можно поменять и разрешение.

Все готово - запускаем python web.py, и с любого браузера можем зайти на страницу http://192.168.0.104:8000 (адрес может быть другим) и увидеть картинку с камеры. Это можно будет сделать и удаленно, если настроить статический IP адрес и перенаправление порта 8000 на роутере.

Чтобы сервер автоматически стартовал при запуске Raspberry Pi, нужно будет добавить строку вызова в /etc/rc.local, например так:

```
cd /home/pi/Documents/Cam
python /home/pi/Documents/Cam/web.py &
```

Важно: поскольку сервер постоянно сохраняет изображения локально на карте памяти, это негативно скажется на ее ресурсе - количество операций записи для карт памяти ограничено. Если планируется постоянное использование такого сервера, рекомендуется создать RAM-диск в памяти и указать путь к нему в настройках fswebcam и index.html. Настройку RAM-диска желающие могут найти в онлайн-руководствах по Raspberry Pi.

4.13. Интервальная съемка (time-lapse photo)

С помощью fswebcam несложно делать серию фото, например каждые 5 секунд. Это позволяет делать интересные кадры, например для записи движения Луны или облаков.

Команда для запуска выглядит так:

fswebcam -l 10 -r 1280x720 test-%Y-%m-%d--%H-%M-%S.jpg

Здесь “-l 10” задает временной интервал, “-r 1280x720” задает разрешение, имя файла “test-%Y-%m-%d--%H-%M-%S.jpg” задает шаблон даты и времени.

Можно указывать разные форматы шаблонов даты-времени, воспользовавшись таблицей:

%d
День месяца [01..31]

%H
Час [00..23]

%I
Час в 12-часовом формате [00..12]

%j
День [001..366]

%m
Месяц [01..12]

%M
Минута [00..59]

%S
Секунда [00..59]

%W
Номер недели в году [00..53]

%w
День [0(Вс)..6]

%y
Год, 2х значное число [00..99]

%Y
Год, 4х значное число

Соответственно, при запуске вышеприведенной команды в текущей папке будут создаваться файлы вида test-2018-02-25--12-39-40.jpg, test-2018-02-25--12-40-50.jpg, test-2018-02-25--12-40-00.jpg и т.д. Потом их можно склеить в видео с помощью специальных утилит.

4.14. Подключаемся к камере с помощью OpenCV

Выше мы использовали fswebcam для записи изображений с камеры. Это хорошо работает, но если мы хотим большей гибкости настроек, то лучше написать собственный код. К тому же, как было сказано выше, постоянная запись файлов уменьшает ресурс карты памяти. Поэтому далее мы будем работать с камерой напрямую.

Для работы с изображениями есть полезная библиотека OpenCV. Для ее установки введем команду: **sudo apt-get install python-opencv**.

Теперь всего лишь в несколько строк кода, мы можем написать свой аналог fswebcam:

```
import cv2
import time

cam = cv2.VideoCapture(0) #set the port of the camera as before
try:
count = 0
while True:
retval, image = cam.read() #return a True bolean and and the image if all go right
```

```
print count
cv2.imwrite("frame-%d.jpg" % count, image)
count += 1
time.sleep(0.5)
except:
pass

cam.release()
```

Как можно видеть, мы создаем объект cam = VideoCapture и с помощью функции cam.read читаем изображение. Оно уже хранится в памяти, нам не нужен промежуточный файл, и это большой плюс. В частности, его можно сохранить в папке, вызовом функции cv2.imwrite. Мы также можем сохранять не все изображения, а лишь в случае срабатывания внешнего триггера, например кнопки или реле.

Помимо простого сохранения файлов, OpenCV имеет огромные возможности по обработке изображений - поворот, кадрирование, ресайз и прочие операции могут быть легко реализованы.

Самостоятельная работа: сделать селфи-камеру на Raspberry Pi. Для этого добавить в вышеприведенный код чтения состояния кнопки, и писать изображение в файл только в том случае, если кнопка нажата.

4.15. Запускаем сервер видеотрансляции

Мы уже транслировали фото с вебкамеры через веб-сервер. Осталось сделать следующий шаг, и транслировать полноценное видео. Тем более, что для этого никаких программ писать не потребуется, достаточно лишь установить необходимые компоненты.

Введем команду: **sudo apt-get install motion**.

Откроем файл настроек, введем команду **sudo nano /etc/motion/motion.conf**.

Зададим следующие параметры:

```
daemon off
width 640
height 480
framerate 15
```

output_pictures off
ffmpeg_output_movies off
stream_port 8081
stream_quality 100
stream_localhost off
webcontrol_localhost off

Кстати, параметры output_pictures и ffmpeg_output_movies отвечают за сохранение изображений и видеороликов - программу motion также можно использовать как детектор движения. Файлы при этом будут складываться в папку /var/lib/motion. Нам эта функция не нужна, так что устанавливаем параметр output_pictures в off.

Наконец, введем команду **sudo motion**. Сервер работает - можно зайти в браузере на адрес http://192.168.0.104:8081 и увидеть изображение с камеры (из соображений приватности картинка не показана).

4.16 Отправляем данные через Dropbox

Мы уже рассматривали отправку данных из ESP32 в Dropbox в главе 3.13. Разумеется, то же самое можно сделать и на Raspberry Pi. Рассмотрим отправку обычного файла и текстовых данных в виде строки. Предварительно поставим библиотеки для Dropbox командой **sudo apt-get install dropbox**. Затем необходимо подключить приложение к Dropbox и получить ключ, как описано в главе 3.13.

Сам код весьма прост. В отличие от ESP32, здесь нам не нужно писать самостоятельно запросы к серверу, все уже реализовано в библиотеке.

```
import dropbox
import os

token = "V3z9NpYlRxEAAAAAAACHVdBdhVRCnXXXXXXXX"
dbx = dropbox.Dropbox(token)

# 1. Upload file "data1.txt" from the current folder
fullname = "data1.txt"
with open(fullname, 'rb') as f:
data = f.read()
try:
dbx.files_upload(data, "/" + fullname, dropbox.files.WriteMode.overwrite, mute=True)
except dropbox.exceptions.ApiError as err:
print 'Dropbox API error', err

# 2. Upload string as a file
try:
dbx.files_upload("1234567".encode(), "/data2.txt", dropbox.files.WriteMode.overwrite)
except dropbox.exceptions.ApiError as err:
print 'Dropbox API error', err
```

Отметим здесь 2 полезных параметра. *WriteMode.overwrite* указывает, что файл будет перезаписан, в противном случае мы получим ошибку, если файл был уже создан. Опциональный параметр

mute=True указывает, что файл надо добавить “молча”, без уведомления пользователя (в противном случае на синхронизированном с Dropbox компьютере появляется всплывающее окно). Это бывает полезно, если файлы обновляются постоянно, например, картинка с камеры, которая сохраняется каждые 5 минут.

4.17 Распознавание лиц с помощью OpenCV

Вычислительная мощность Raspberry Pi весьма высока, и позволяет использовать довольно сложные алгоритмы, недоступные на Arduino или ESP32. Одна из таких задач - это обработка изображений. Мы уже умеем получать изображения с Web-камеры, добавим теперь возможность распознавания лиц. Это позволит к примеру, отправлять изображение с охранной камеры только тогда, когда на нем есть человеческое лицо, или сделать робота, который будет поворачиваться к стоящему перед ним человеку.

Разумеется, сама по себе задача распознавания лиц на изображении, весьма сложна, ее программирование требует весьма сложной математики. К счастью для нас, такие алгоритмы уже написаны, потребуется лишь использовать готовые библиотеки. Для обработки изображений мы будем использовать библиотеку OpenCV, чтобы поставить ее, предварительно введем команду **sudo apt-get install libopencv-dev python-opencv**.

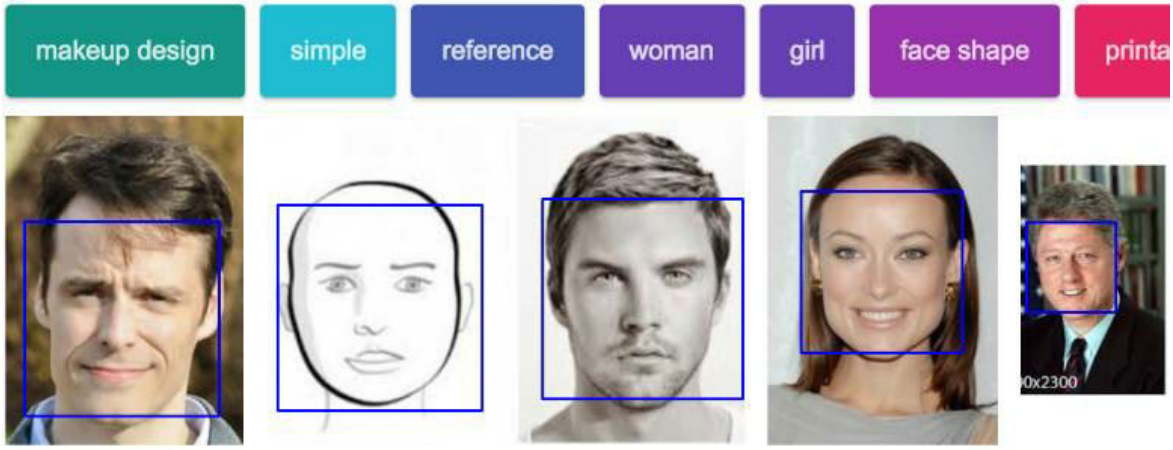
Для начала, рассмотрим задачу распознавание лиц на уже готовом изображении. Для распознавания лиц используется так называемый классификатор Хаара, реализация которого уже есть в OpenCV.

```
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

img = cv2.imread('faces.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

```
for (x,y,w,h) in faces:
cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
print " ", x,y, x+w,y+h

cv2.imwrite('faces_out.png', img)
```

Как можно видеть, код весьма прост. Мы создаем объект “CascadeClassifier”, в качестве параметра которого указываем файл 'haarcascade_frontalface_default.xml'. Этот файл хранит заранее обученный на большом числе изображений, набор данных, его необходимо взять из дистрибутива OpenCV и положить в папку с программой. Далее, исходное изображение “faces.png” переводится в черно-белое, а вызов метода detectMultiScale собственно и делает всю работу по распознаванию - на выходе мы получаем массив прямоугольников faces, с которым можем делать что угодно, например вывести на экран. Для удобства просмотра результатов мы также рисуем прямоугольник поверх исходной картинки с помощью метода cv2.rectangle. Итоговая картинка сохраняется под именем faces_out.png.

В качестве теста был взят обычный скриншот из браузера, результаты работы программы показаны на картинке:



Как можно видеть, алгоритм работает вполне неплохо. В окне вывода мы также получаем координаты прямоугольников:

```
1092 256 1187 351
853 223 1024 394
30 255 236 461
579 231 791 443
```

298 238 515 455

Самостоятельная работа: добавить распознавание лиц в код чтения изображения с веб-камеры из главы 4.14. Опционально, можно сделать анализ координат полученного прямоугольника, что позволит определить, находится человек слева или справа от камеры.

На этом мы закончим изучение Raspberry Pi, хотя по сути, это только начало. На этой платформе можно сделать много чего интересного, по мере появления нового материала главы будут дополняться.

А сейчас мы перейдем к изучению платы для самых маленьких - микрокомпьютера BBC:Micro.

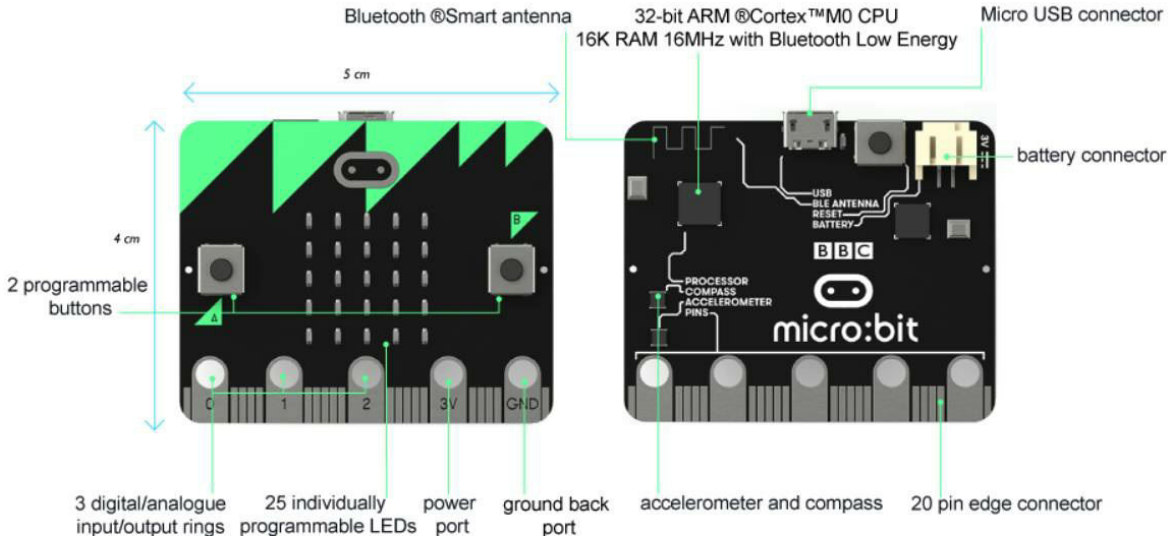
Часть 5. BBC Micro:bit - электроника для самых маленьких

5.1 Введение

Компьютеры BBC Micro имеют давнюю историю. Еще в 80х британская компания BBC запустила обучающий проект, целью которого было повышение уровня компьютерной грамотности. Компьютер по тем временам был действительно “микро”, и выглядел примерно так:

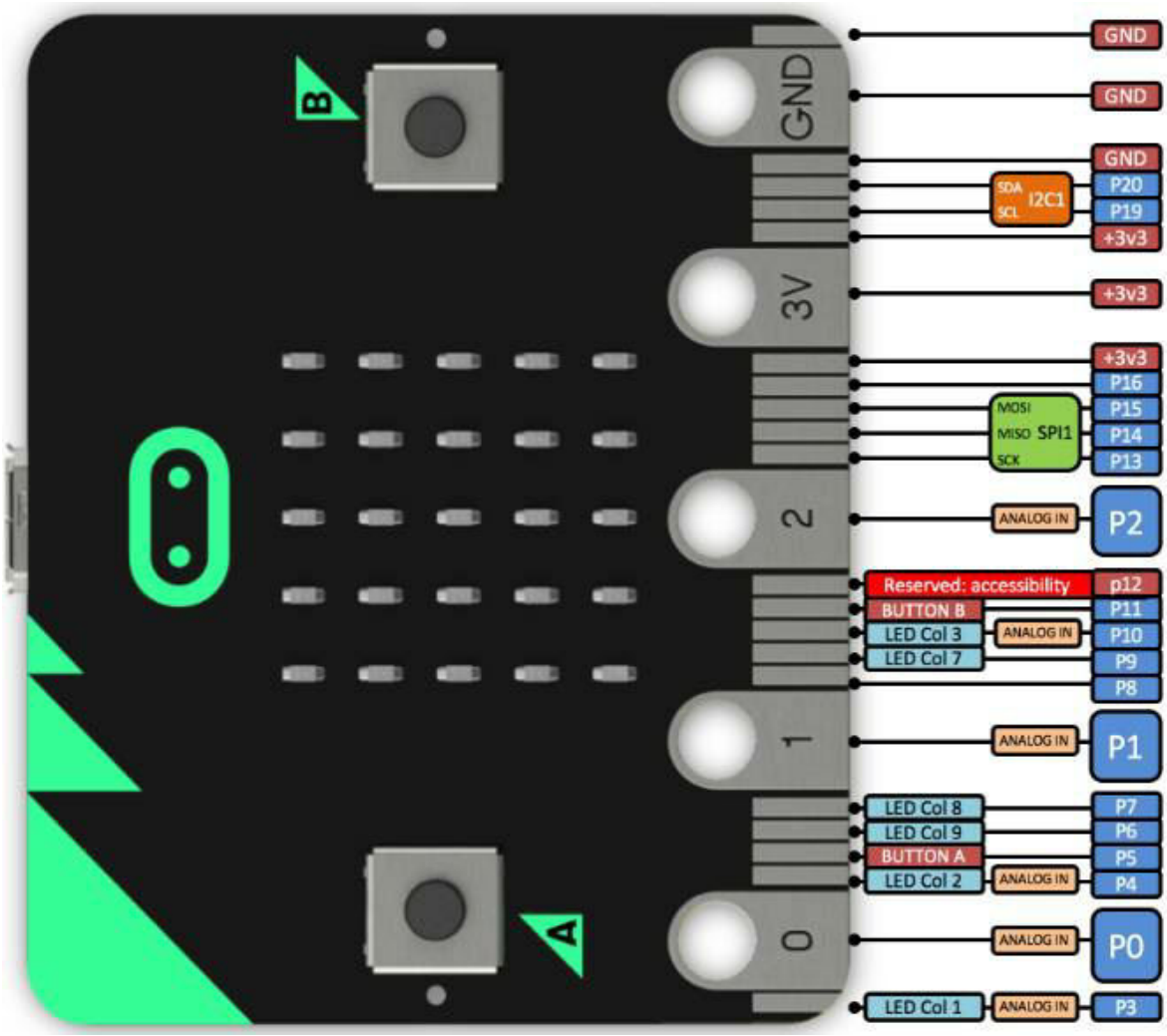
С тех пор прошло около 30 лет, и на свет появилась новая

“модификация” на современной элементной базе - BBC Micro:bit.



Плата предназначена для обучения младших школьников программированию, причем всем школьникам Великобритании она поставляется … бесплатно. Цель устройства - в простой и понятной форме показать как создавать программы, с уклоном именно в современную электронику и “интернет вещей”.

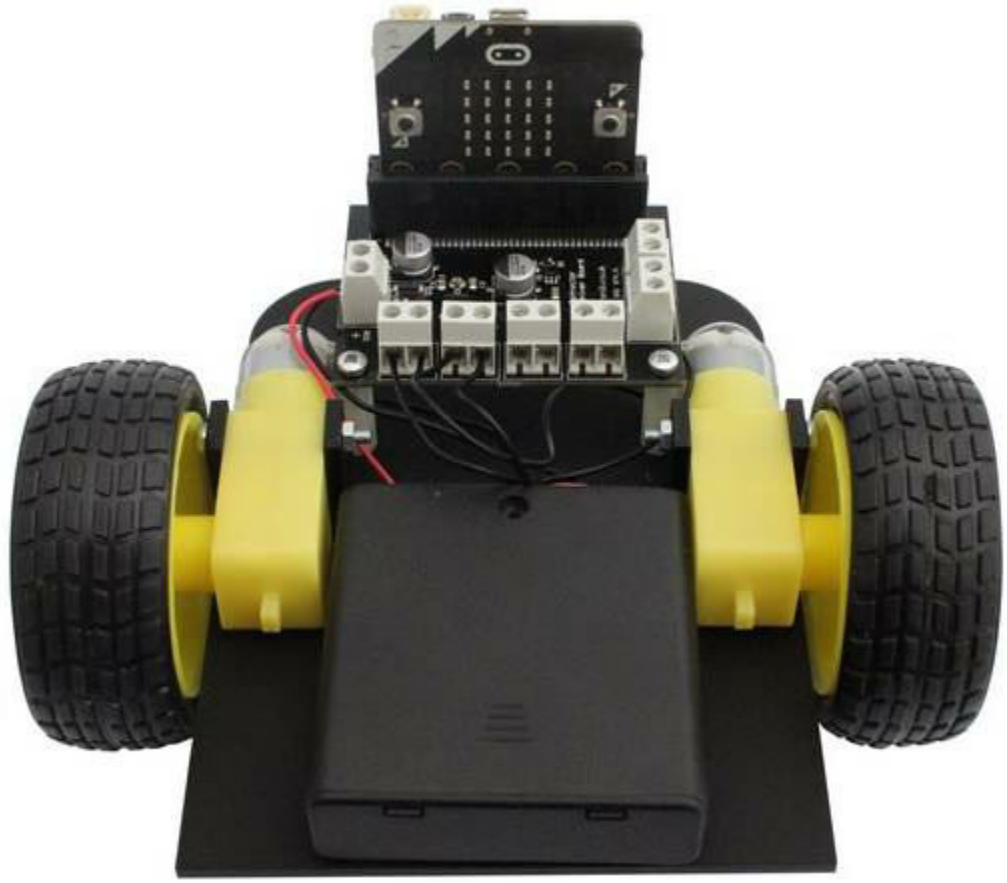
«На борту» BBC Micro:bit есть процессор ARM Cortex-M0, 256Кб Flash ROM, 16Кб RAM, 16МГц тактовая частота. Также есть поддержка BTLE, 2.4ГГц-трансмиттер для одноранговой связи (101 канал), акселерометр, компас, термометр, и линейка пинов GPIO, среди которых довольно много всего:



Также есть 2 кнопки для ввода (еще для ввода доступен жест «встряхивание»), светодиодная матрица 5x5, и 4 «крупных» пина, рассчитанных на то, чтобы ребенок прикрутил проводами или «крокодилами» что-нибудь несложное, например датчик влажности для цветка или переменный резистор.

Гребенка пинов сделана плоской, так что ее можно вставить в плату расширения (цена вопроса 10-15 Евро).

Есть разные платы расширения, например плата управления моторами, что позволяет сделать такого робота:



Для тех, кто учится не в Англии, плата бесплатной, разумеется, не будет. Но заказать ее по вполне невысокой цене можно на eBay или на Amazon. Кстати, пока плата идет почтой, можно уже начинать писать программы - на сайте доступен полностью функциональный эмулятор, к которому мы и приступим.

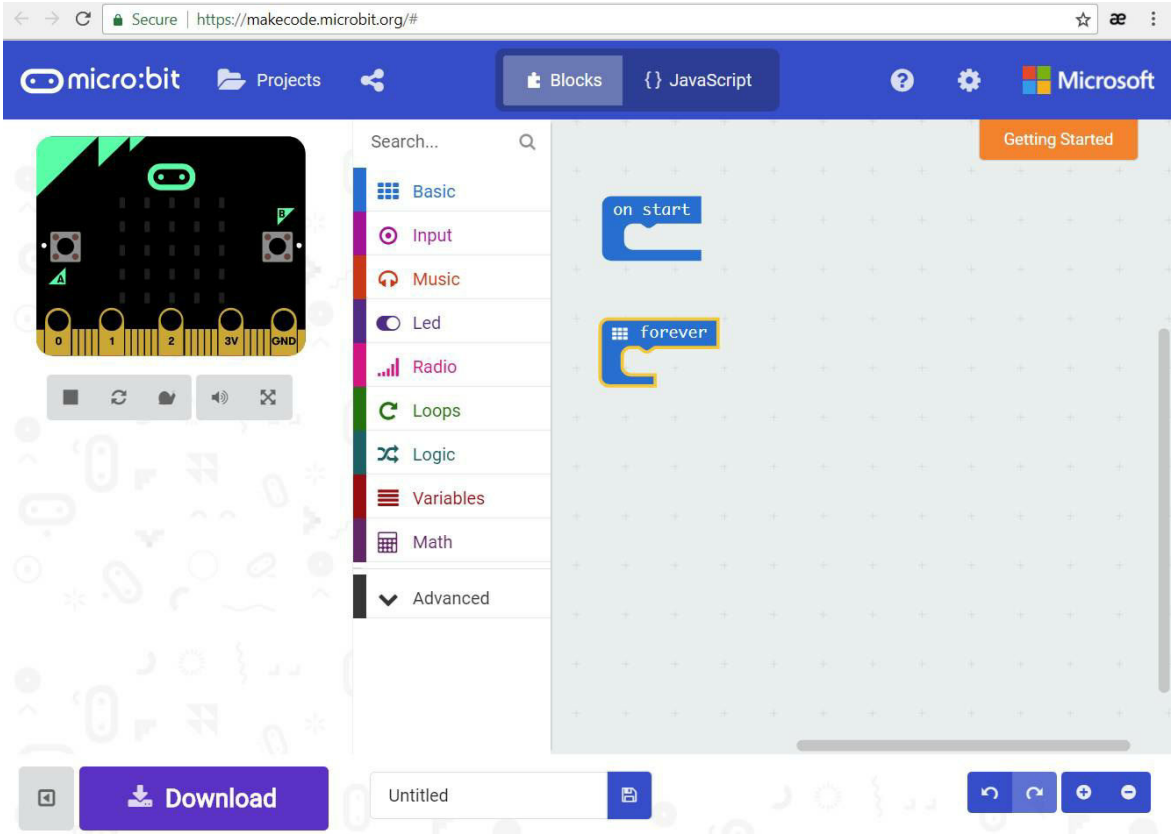
Для тех же, кому возможностей платы станет мало, можно заказать “Набор изобретателя” (inventor’s kit), состоящий из платы расширения, макетной платы и различных деталей.

А мы пока начнем с самого начала.

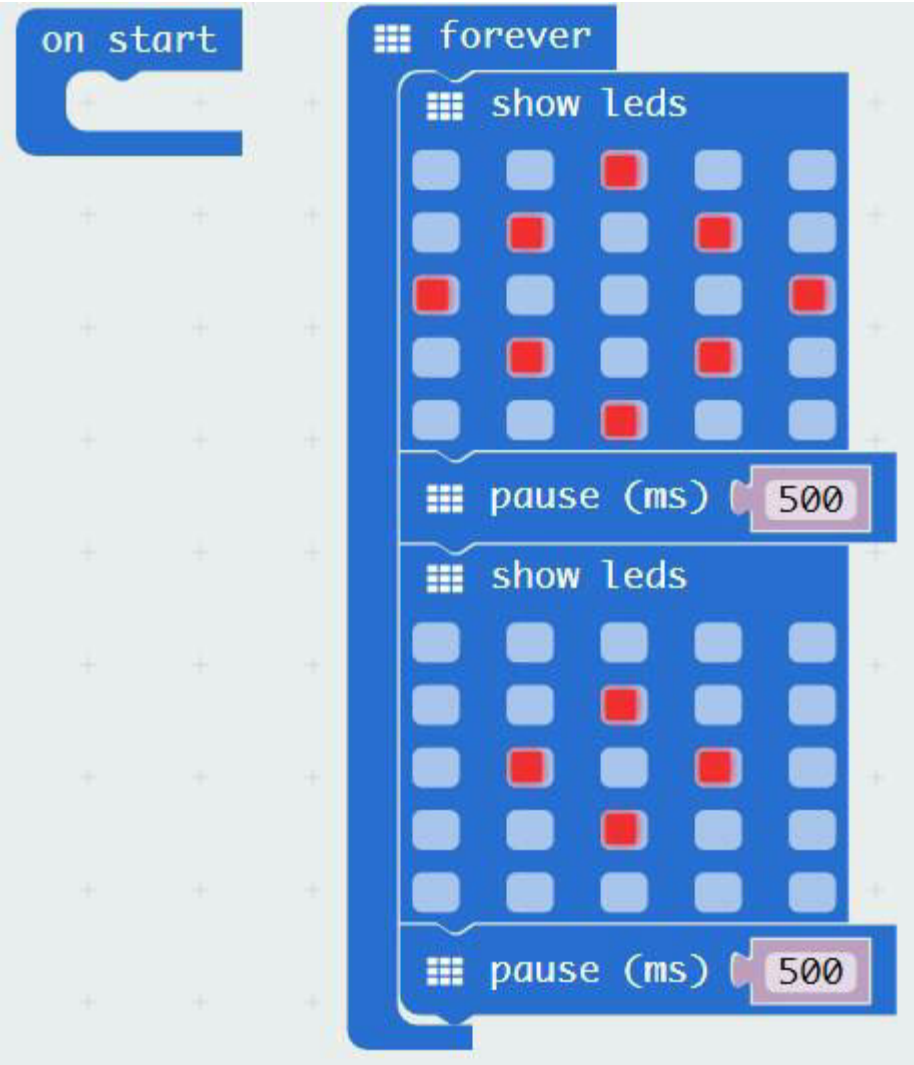
5.2 Первая программа для BBC Micro:bit

Первое что удивляет, никакого софта на компьютер ставить не нужно вообще (кстати, это удобно и для школ - им не придется покупать лицензионное программное обеспечение). При подключении платы к компьютеру, она просто видна как съемный диск. Дальше достаточно зайти на сайт http://microbit.org/code/ и выбрать на каком языке мы хотим писать — Javascript или Python. Мы выберем блочный редактор Javascript, как самый простой для начинающих. Сам код писать не придется, все конструкции перетаскиваются мышью прямо на экране.

Редактор Javascript в «блочном» режиме открывается прямо в браузере и выглядит вот так:



Традиционно, начнем с самого простого - мигания светодиодом. Здесь у нас даже не один светодиод, а целая матрица 5x5. Выберем раздел Basic, из него “перетащим” на экран компоненты show leds и pause, как показано на рисунке справа.



Как можно видеть, принципиального отличия от того, что мы делали в Arduino, в общем и нет. Лишь названия немного другие, например, вместо setup используется функция on_start, а вместо функции loop, используется функция forever. Инициализация режима ввода-вывода делается автоматически, что проще.

Сам код очень прост. Мы выводим определенную картинку на матрицу светодиодов, затем делаем паузу на 500мс (0.5с), затем меняем картинку на другую, и процесс повторяется. Режим перетаскивания блоков мышью удобен тем, что не дает сделать синтаксических ошибок, это позволяет создавать программы даже самым маленьким детям. Кстати, желающие посмотреть, как код устроен внутри, могут переключиться из “блочного” редактора в “обычный”, нажав кнопку

Javascript. При этом откроется окно с текстом программы:

```
basic.forever(() => {
basic.showLeds(`
. . # . .
. # . # .
# . . . #
. # . # .
. . # . .
`)
basic.pause(500)
basic.showLeds(`
. . . . .
. . # . .
. # . # .
. . # . .
. . . . .
`)
basic.pause(500)
})
```

Здесь, как можно видеть, задается лишь функция *basic.forever*, что делается внутри нее, вполне читабельно и понятно.

Еще один удобный момент - при добавлении компонентов, симулятор платы слева начинает работать автоматически, показывая что получится в результате. Таким образом, для написания и отладки программы даже не нужна сама плата BBC Micro:bit, все можно сделать прямо в симуляторе на странице.

Наконец, посмотрев на программу в симуляторе, можно загрузить ее в реальную плату. Подключаем Micro:bit к компьютеру USB-кабелем (она определится в системе как диск), на странице нажимаем Download, и сохраняем получившийся Hex-файл на новый диск. После сохранения, программа запустится автоматически, и мы увидим горящие светодиоды. Как и в случае с Arduino, программа сохраняется во флеш-памяти, и при следующем включении платы будет запущена автоматически. Это позволяет использовать BBC Micro:bit автономно,

отдельно от компьютера.

5.3 Возможности ввода-вывода

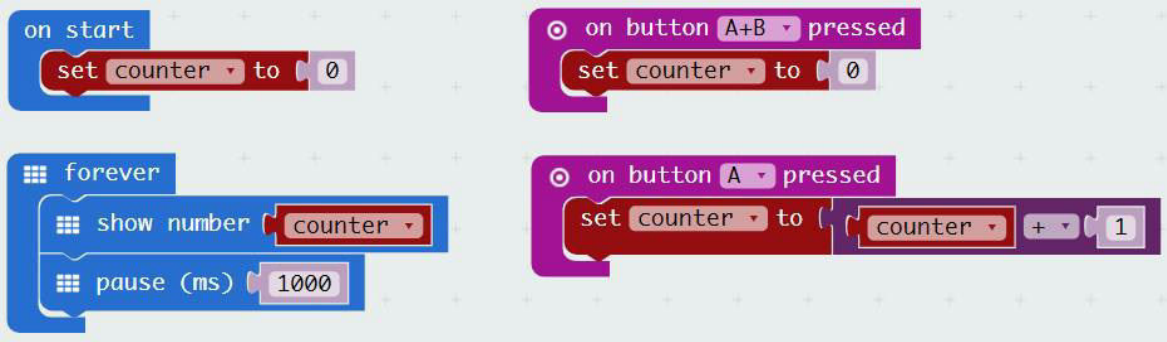
Несмотря на небольшой размер, BBC Micro:bit имеет достаточно разнообразные способы ввода данных.

Пользователю доступны:

- нажатие кнопок A или B,
- одновременное нажатие кнопок A+B,
- прикосновение одной рукой к контакту 0, 1 или 2, другой к GND,
- встряхивание,
- использование акселерометра.

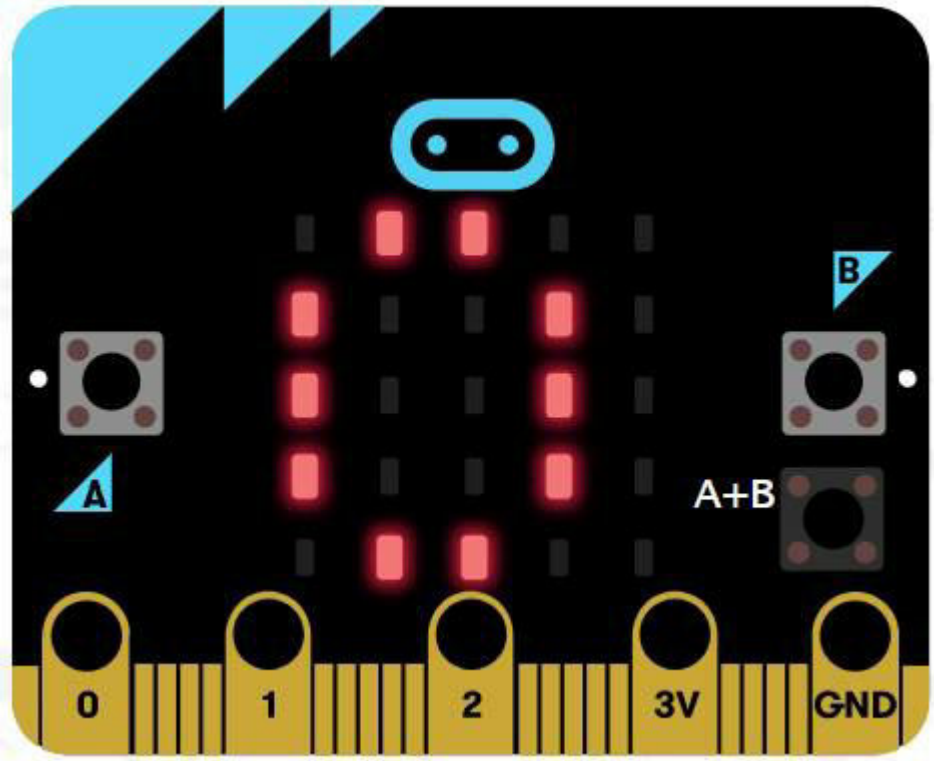
Для примера, сделаем простой счетчик, увеличивающий значение при нажатии кнопки “A” или “B”, одновременное нажатие будет сбрасывать счетчик в 0. Он может пригодиться например, при подсчете числа посетителей на спортивном соревновании, птиц, прилетающих к кормушке, и пр. Такие устройства, кстати, до сих пор производятся и продаются в “механическом” виде:

Ну а мы сделаем электронную версию такого счетчика. “Код” несложно создать мышью, блок-схема показана на рисунке.



Те, кто использовал кнопки в Arduino, наверняка увидят знакомые конструкции. В целом, код не нуждается в комментариях, все просто и очевидно. Ординарное нажатие кнопки используется для инкремента счетчика, двойное нажатие используется для его сброса. К сожалению, матрица светодиодов не позволяет отображать длинные цифры, поэтому если значение счетчика больше 10, то автоматически включается прокрутка.

Как говорилось выше, код можно протестировать прямо в браузере, с помощью симулятора:



Когда программа проверена и отлажена, ее можно загрузить в

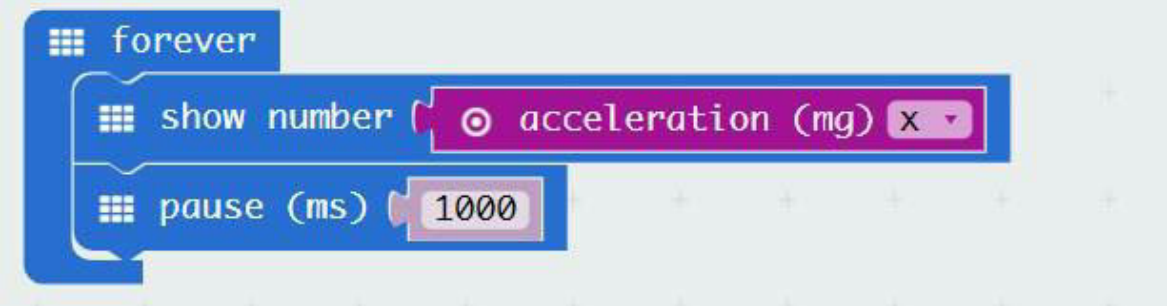
реальную плату. Кстати, в комплекте с платой идет блок питания от батареек, так что BBC Micro:bit можно брать с собой и использовать не только дома, но и с друзьями или на улице.

Самостоятельная работа: попробовать остальные способы ввода, доступные на BBC Micro:bit. Кстати, среди них есть довольно-таки экзотические, например в качестве события можно использовать “свободное падение”.

5.4 Управление наклонами платы - использование акселерометра

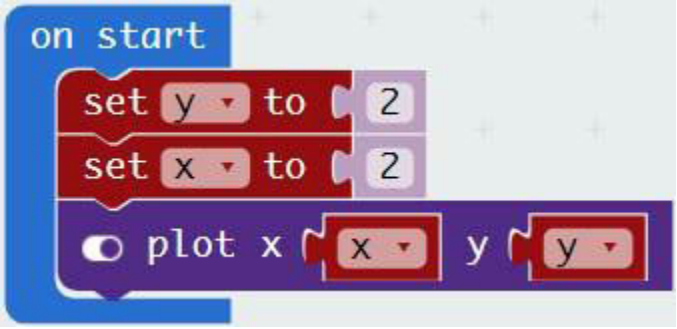
Рассмотрим режим, наиболее подходящий для несложных игр - выведем на экране пиксел, которым можно будет управлять, наклоняя плату в разную сторону. За наклоны платы “отвечает” акселерометр - сенсор, позволяющий получить значения относительно осей x, y и z.

Шаг-1: Узнаем, как меняются показания акселерометра при наклоне платы. Это можно найти в документации, но гораздо проще и быстрее просто проверить. Для этого создаем простую программу, выводящую значения на экран.

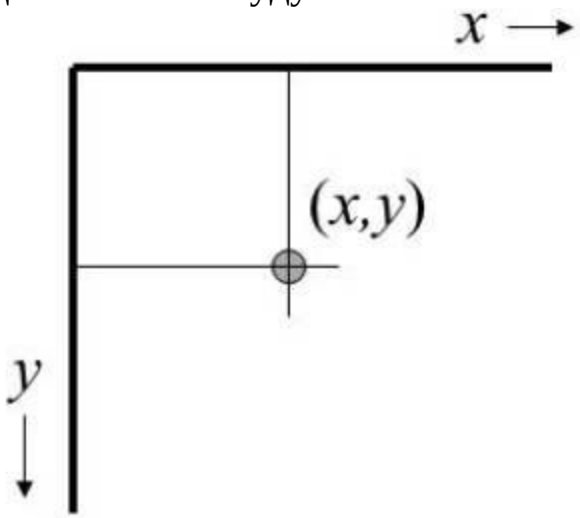


Запускаем программу, наклоняем плату и убеждаемся что значения варьируются в диапазоне -1023...1023.

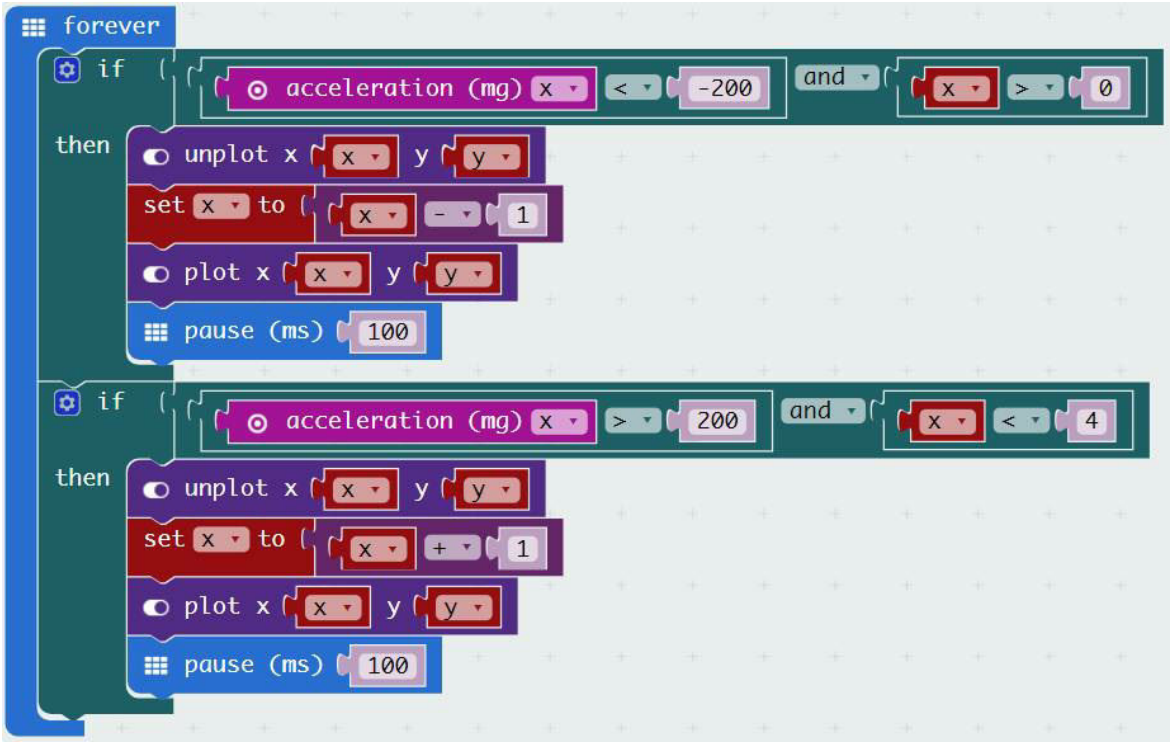
Шаг-2. Выведем на экране точку, положение которой будет определяться двумя координатами x и y.



Матрица светодиодов имеет размерность 5x5, соответственно, координаты точек будут изменяться от (0,0) до (4,4).



Шаг-3. Изменим значение x, если наклон платы превышает определенную величину. Максимальный наклон равен 1023, возьмем нечто меньшее, например 200. Также необходимо будет добавить проверку на пересечение краев экрана: x не должно быть меньше 0 и больше 4. Также добавим паузу, чтобы точка не “убегала” слишком быстро. Получается немного громоздкая, но вполне понятная конструкция.

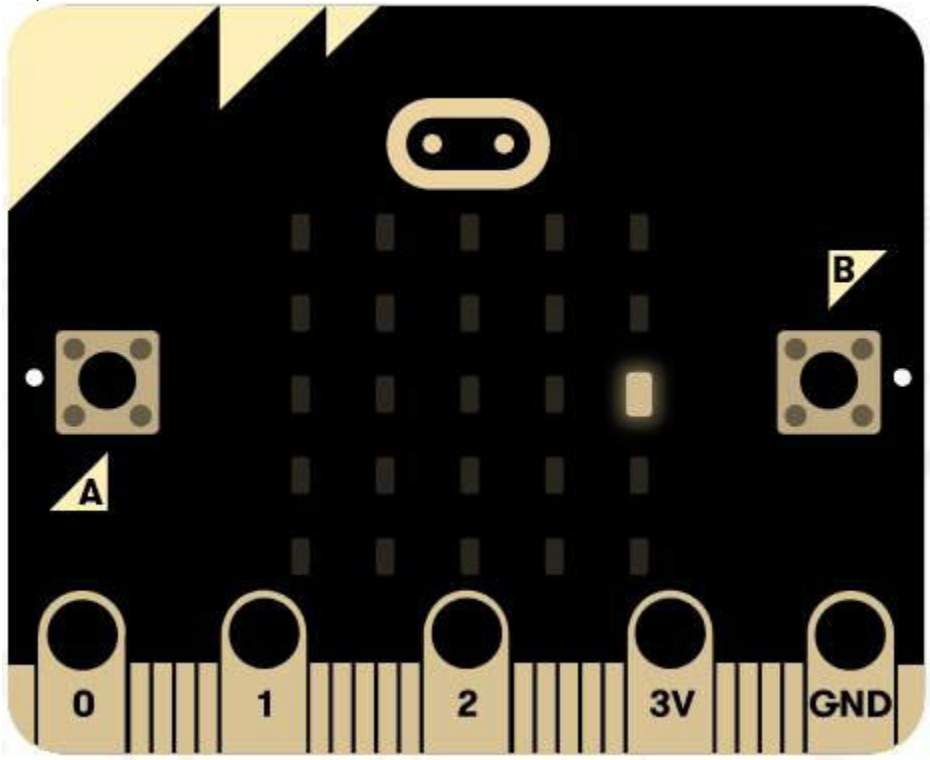


Сам код вполне понятен - если наклон платы превышает пороговую величину, мы гасим старый пиксел командой *unplot*, увеличиваем (или уменьшаем) значение координаты, и зажигаем пиксел снова командой *plot*. Кстати, в данном случае бывает полезно переключиться в режим прямого редактирования кода - однотипные блоки и логические выражения вводить так гораздо быстрее. Полностью код выглядит так:

```
let y = 2
let x = 2
led.plot(x, y)
basic.forever(() => {
if (input.acceleration(Dimension.X) < -200 && x > 0) {
led.unplot(x, y)
x = x - 1
led.plot(x, y)
basic.pause(100)
}
if (input.acceleration(Dimension.X) > 200 && x < 4) {
led.unplot(x, y)
x = x + 1
led.plot(x, y)
```

```
basic.pause(100)
}
})
```

Запускаем программу, убеждаемся что можем управлять точкой с помощью наклонов платы:

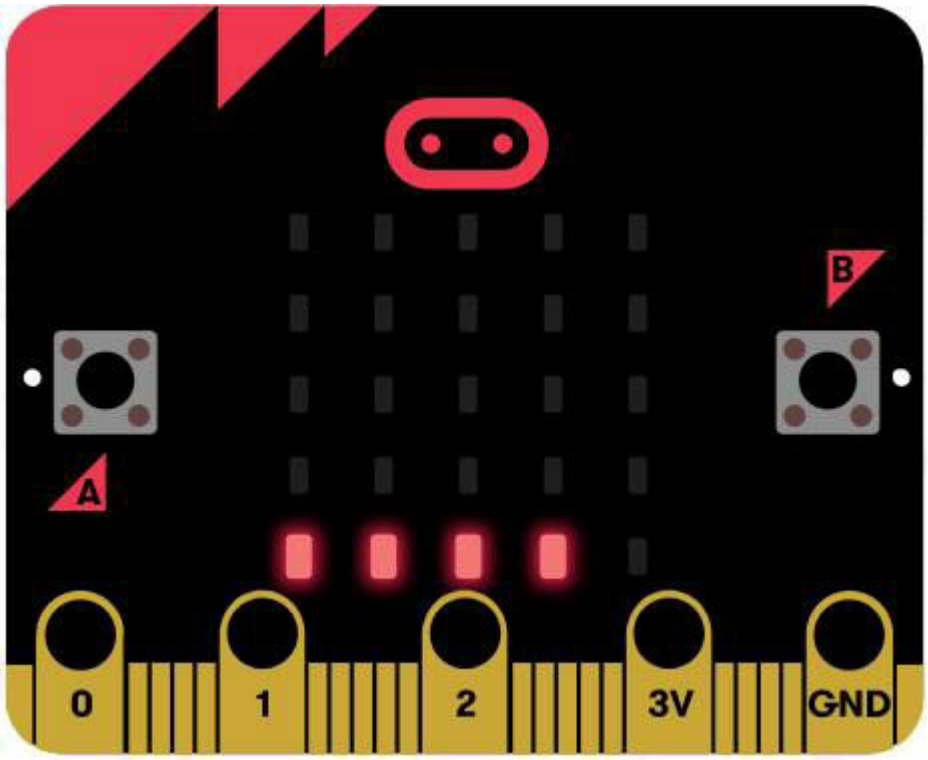


Самостоятельная работа #1: Добавить обработку второй координаты y, чтобы точка могла двигаться не только по горизонтали, но и по вертикали.

Самостоятельная работа #2: Продумать и реализовать сценарий мини-игры, где сверху будут падать объекты, а с помощью “каретки” их можно будет ловить. “Каретку” можно будет сделать их двух точек, находящихся на нижней плоскости (y-координата всегда будет = 4), наклонами платы ее можно будет двигать вправо-влево. Для хранения информации о падающих объектах нужно будет добавить отдельные переменные.

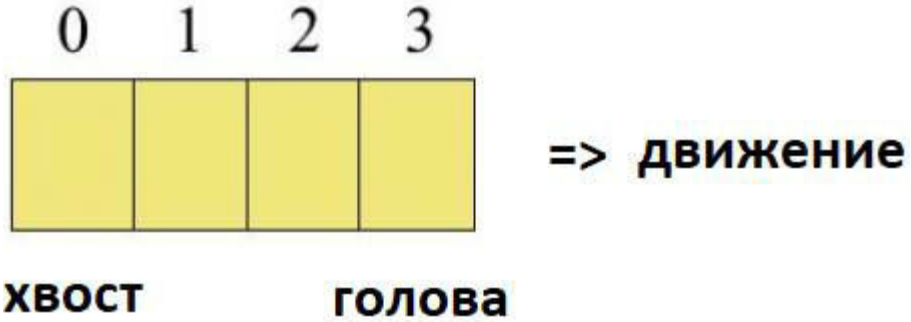
5.5 Игра “Змейка”

Мы уже умеем двигать точку по экрану. Рассмотрим более сложную задачу - будем двигать “змейку”, состоящую из нескольких точек.



Основная сложность здесь - в хранении координат, для чего мы используем структуру данных, называемую массив. Массив - это список объектов одного типа, как раз то что нам нужно.

Шаг 1. Продумаем структуру данных. В нашем случае, мы будем использовать два массива для хранения координат x и y. Наша “змея”, таким образом, будет представляться в памяти следующим образом:



Важно иметь в виду, что массивы нумеруются с нуля. Соответственно, элемент x[0], y[0] указывает на “хвост” змеи, элемент x[3], y[3] указывает на “голову”. Обозначим количество элементов как

N=4, это позволит легко менять длину змеи, изменив всего один параметр. Тогда “головой” будет элемент x[N-1], y[N-1].

К сожалению, создавать массивы в визуальном редакторе не очень удобно. Проще переключиться в режим кода, и добавить следующий текст:

```
let N = 4
let x: number[] = [0, 1, 2, 3]
let y: number[] = [4, 4, 4, 4]
```

Мы создали массивы x и y и проинициализировали их начальными значениями. Теперь выведем точки на экран:

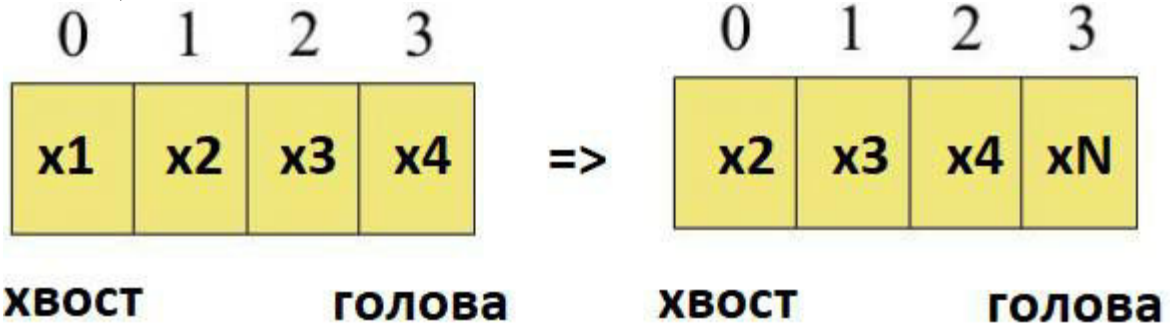
```
for (let p = 0; p < N; p++) {
led.plot(x[p], y[p])
}
```

Шаг-2. Движение “змеи”. Это пожалуй, самая сложная часть данной задачи.

Чтобы передвинуть змею, мы должны:

- выключить крайний элемент в “хвосте”,
- сдвинуть все остальные элементы массива влево (первый станет нулевым, второй первым и пр),
- изменить значение элемента в “голове” и включить новый светодиод.

Графически для массива X это выглядит так (для Y все аналогично):



Все это несложно по сути, главное не ошибиться с индексами и не перепутать, что куда сдвигается. Новое значение xN будет вычисляться из старого, в зависимости от того, куда движется змея. Напишем универсальную функцию moveSnake, которая сдвигает змею в любую

сторону.

```
function moveSnake(dx: number, dy: number) {
if (x[N - 1] + dx < 0 || x[N - 1] + dx > 4 ||
y[N - 1] + dy < 0 || y[N - 1] + dy > 4) {
return
}

led.unplot(x[0], y[0])
for (let q = 0; q < N - 1; q++) {
x[q] = x[q + 1]
y[q] = y[q + 1]
}
x[N - 1] = x[N - 2] + dx
y[N - 1] = y[N - 2] + dy
led.plot(x[N - 1], y[N - 1])
}
```

Рассмотрим код подробнее. Параметры dx и dy - приращения координат: если змея движется вправо, то dx=1, если влево, то dx=-1, если вверх то dy=-1, если вниз то dy=1. Т.к. “змея” движется головой вперед, то эти приращения добавляются к координатам “головы”. Но вначале мы проверяем, что “змея” не уйдет за края экрана, и что новые значения не будут меньше 0 или больше 4. Команда led.unplot(x[0], y[0]) выключает последний светодиод в “хвосте” змеи. Затем в цикле for мы сдвигаем все элементы “змеи”, как было показано выше на рисунке. Последним шагом мы формируем новое значение координат “головы”, которое мы определяем как старое значение, плюс новое приращение координат dx, dy. Наконец, нам достаточно включить новый светодиод функцией led.plot(x[N - 1], y[N - 1]).

Шаг-3. Подключение акселерометра. В зависимости от наклона платы, будем двигать “змею” в нужную сторону.

```
if (input.acceleration(Dimension.X) > 200) {
moveSnake(1, 0)
basic.pause(200)
}
if (input.acceleration(Dimension.X) < -200) {
moveSnake(-1, 0)
```

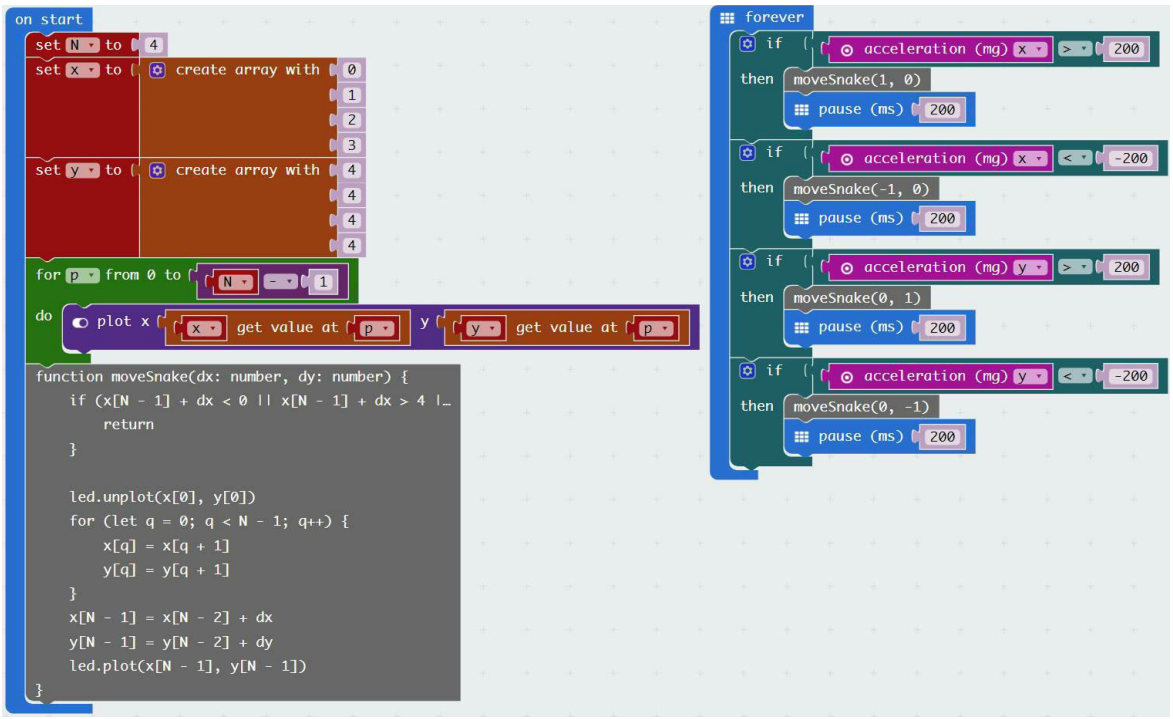
```
basic.pause(200)
}
...
```

На этом программа готова. Код полностью приведен ниже, его можно просто скопировать в редактор для запуска на плате или в симуляторе.

```
let N = 4
let x: number[] = [0, 1, 2, 3]
let y: number[] = [4, 4, 4, 4]
for (let p = 0; p < N; p++) {
led.plot(x[p], y[p])
}
function moveSnake(dx: number, dy: number) {
if (x[N-1] + dx < 0 || x[N-1] + dx > 4 || y[N-1] + dy < 0 || y[N-1] + dy > 4)
{
return
}

led.unplot(x[0], y[0])
for (let q = 0; q < N - 1; q++) {
x[q] = x[q + 1]
y[q] = y[q + 1]
}
x[N - 1] = x[N - 2] + dx
y[N - 1] = y[N - 2] + dy
led.plot(x[N - 1], y[N - 1])
}

basic.forever(() => {
if (input.acceleration(Dimension.X) > 200) {
moveSnake(1, 0)
basic.pause(200)
}
if (input.acceleration(Dimension.X) < -200) {
moveSnake(-1, 0)
```

```
basic.pause(200)
}
if (input.acceleration(Dimension.Y) > 200) {
moveSnake(0, 1)
basic.pause(200)
}
if (input.acceleration(Dimension.Y) < -200) {
moveSnake(0, -1)
basic.pause(200)
}
})
```

К сожалению, такой код уже достаточно сложен для редактирования в блочном редакторе, хотя в принципе это и возможно. Если все-таки переключиться в редактор, мы увидим такую картину:



Также редактор формирует не совсем удобный для чтения код, например строки

let N = 4
let x: number[] = [0, 1, 2, 3]
let y: number[] = [4, 4, 4, 4]

при переключении “туда-обратно” автоматически заменяются на

let y: number[] = []
let x: number[] = []
let N = 0
N = 4
x = [0, 1, 2, 3]
y = [4, 4, 4, 4]

Это не влияет на работоспособность программы, но читать такой код становится менее удобно.

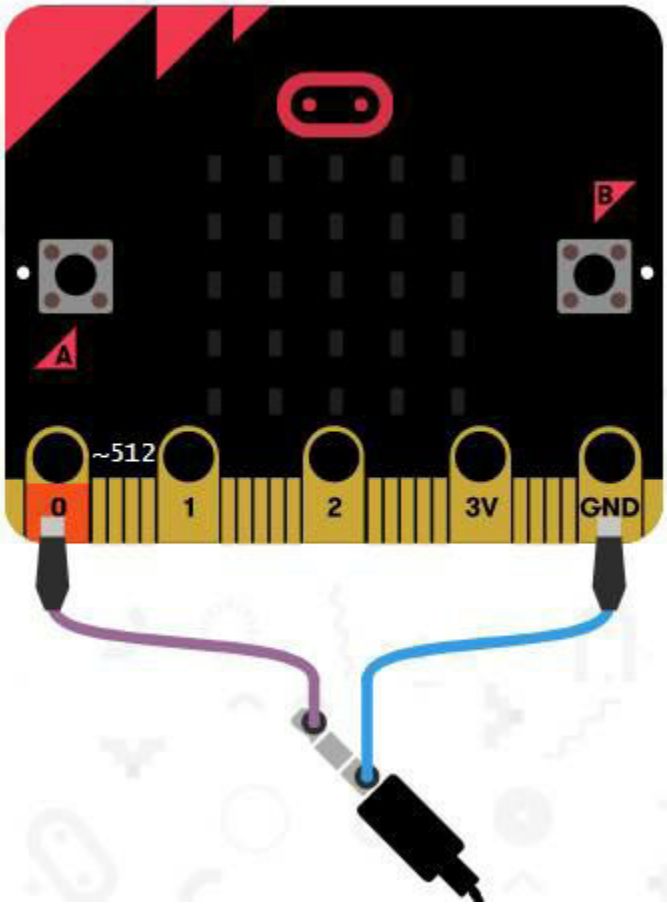
Самостоятельная работа #1: Изменить длину “змеи”, для этого достаточно заменить параметр N и изменить начальные значения массивов координат.

Самостоятельная работа #2: Разместить на экране 3-4 дополнительные точки, которые змея сможет “съедать”. Для проверки

“съедения” достаточно убедиться что новые координаты “головы” совпадают с координатами точки. Когда точка “съедена”, ее координаты можно заменить на недействительные, например на -1,-1. Опционально, можно увеличивать длину змеи, если она “съела” одну точку, для этого придется добавить новые элементы в массивы x и y и увеличить значение N. Добавить элемент к массиву можно, используя функцию push, таким образом, код увеличения “змеи” будет выглядеть так:

```
x.push(x[N - 1])
y.push(y[N - 1])
N = N + 1
```

5.6 Воспроизведение звука

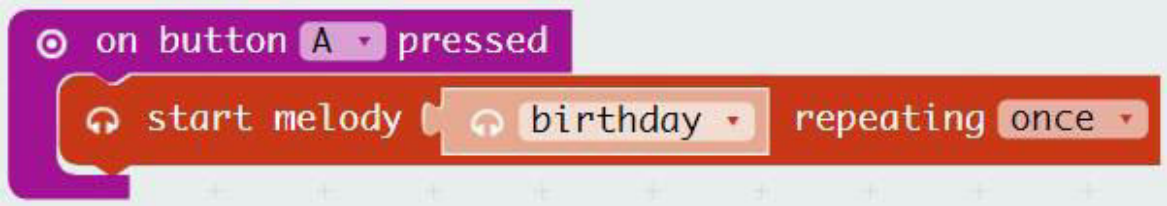


Весьма интересными и несложными являются опыты со звуком. Сначала нужно подключить к BBC:Micro наушники или внешнюю колонку, схема подключения показана на рисунке справа.

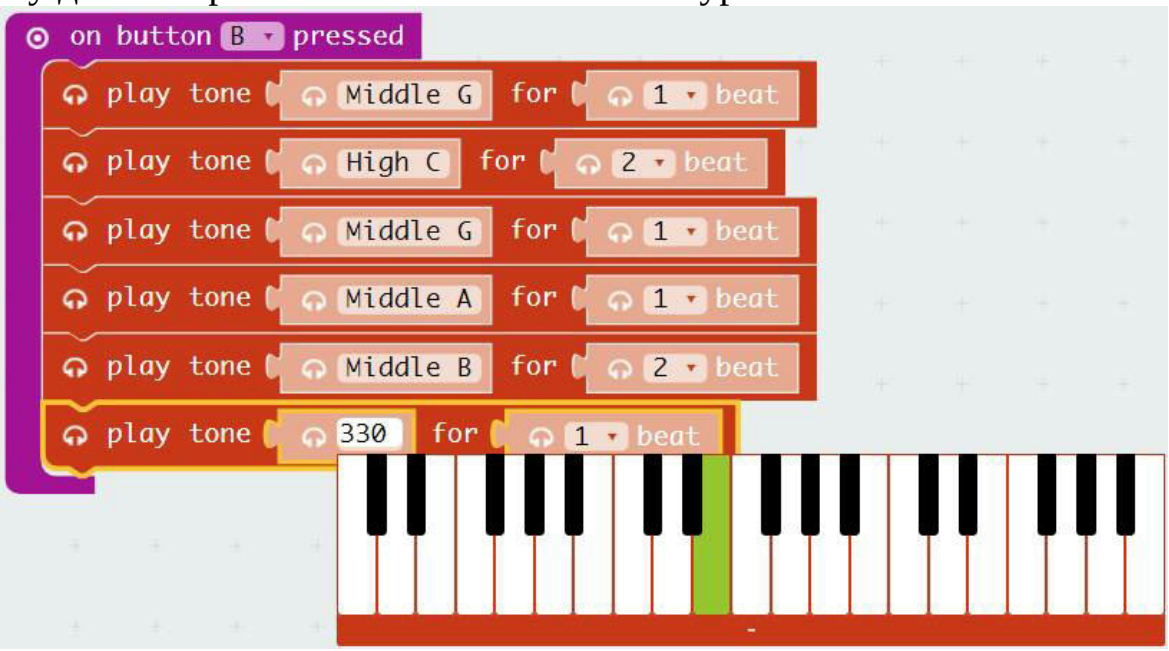
Кабель с зажимами можно сделать самостоятельно, или купить

готовый. Само воспроизведения звука не представляет какой-либо сложности. Есть два варианта.

1) Можно воспроизвести несложную мелодию из списка уже предустановленных, это может пригодиться, например в игре. Вот такой несложный блок воспроизведет Happy Birthday при нажатии на кнопку “A”:



2) Можно воспроизвести собственную мелодию, для чего потребуется задать высоту и длительность каждой ноты. При нажатии на ноту даже открывается окно мини-клавиатуры:



Разумеется, можно воспроизводить ноту или мелодию не только при нажатии на кнопку, но и при каком-либо событии, например при начале или окончании игры.

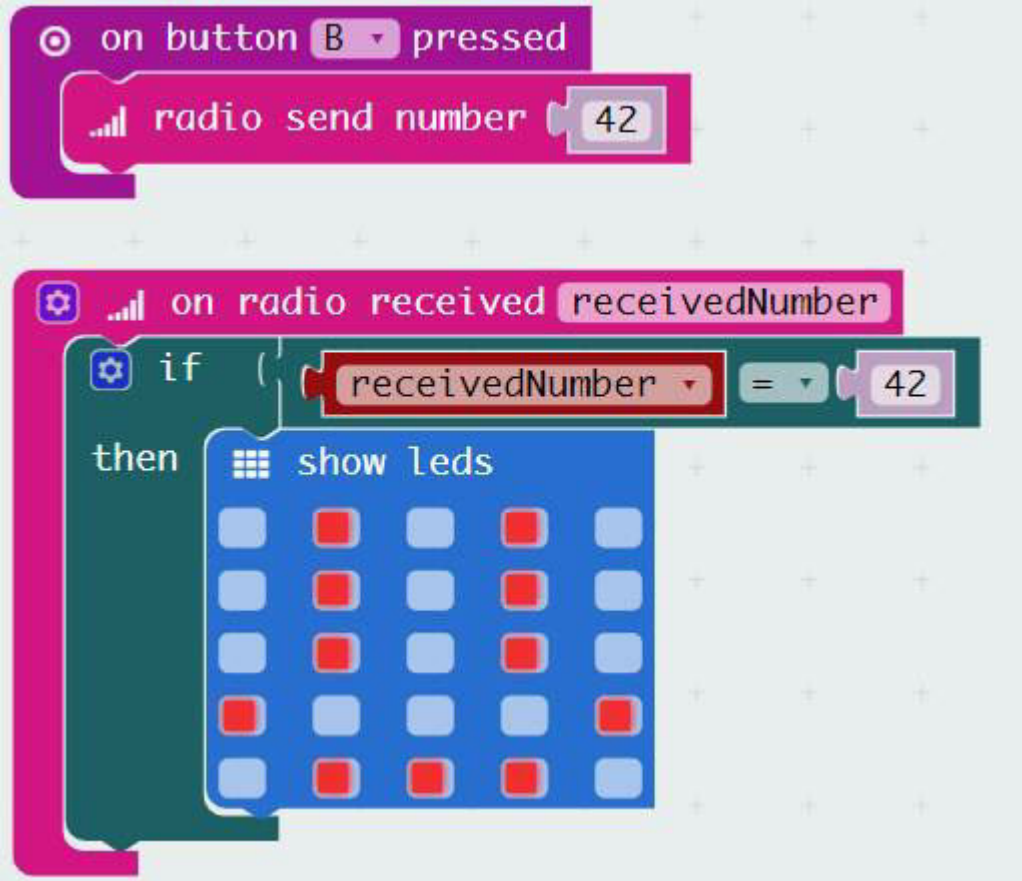
5.7 Использование радиомодуля

Помимо обычных способов ввода-вывода, BBC Micro:bit имеет встроенный радиомодуль. Это позволяет например, делать игры, в которые можно играть вдвоем - можно обмениваться данными между

устройствами (разумеется, для этого нужно иметь как минимум 2 платы Micro:bit).

Функции для использования радиомодуля весьма просты - можно посылать число, строку или пару “имя-значение”. Для приема данных обработчики сообщений с аналогичными названиями.

Простая программа, показанная ниже, отправляет число 42 по нажатию кнопки B, на второй плате в это же время загорится “смайлик”.

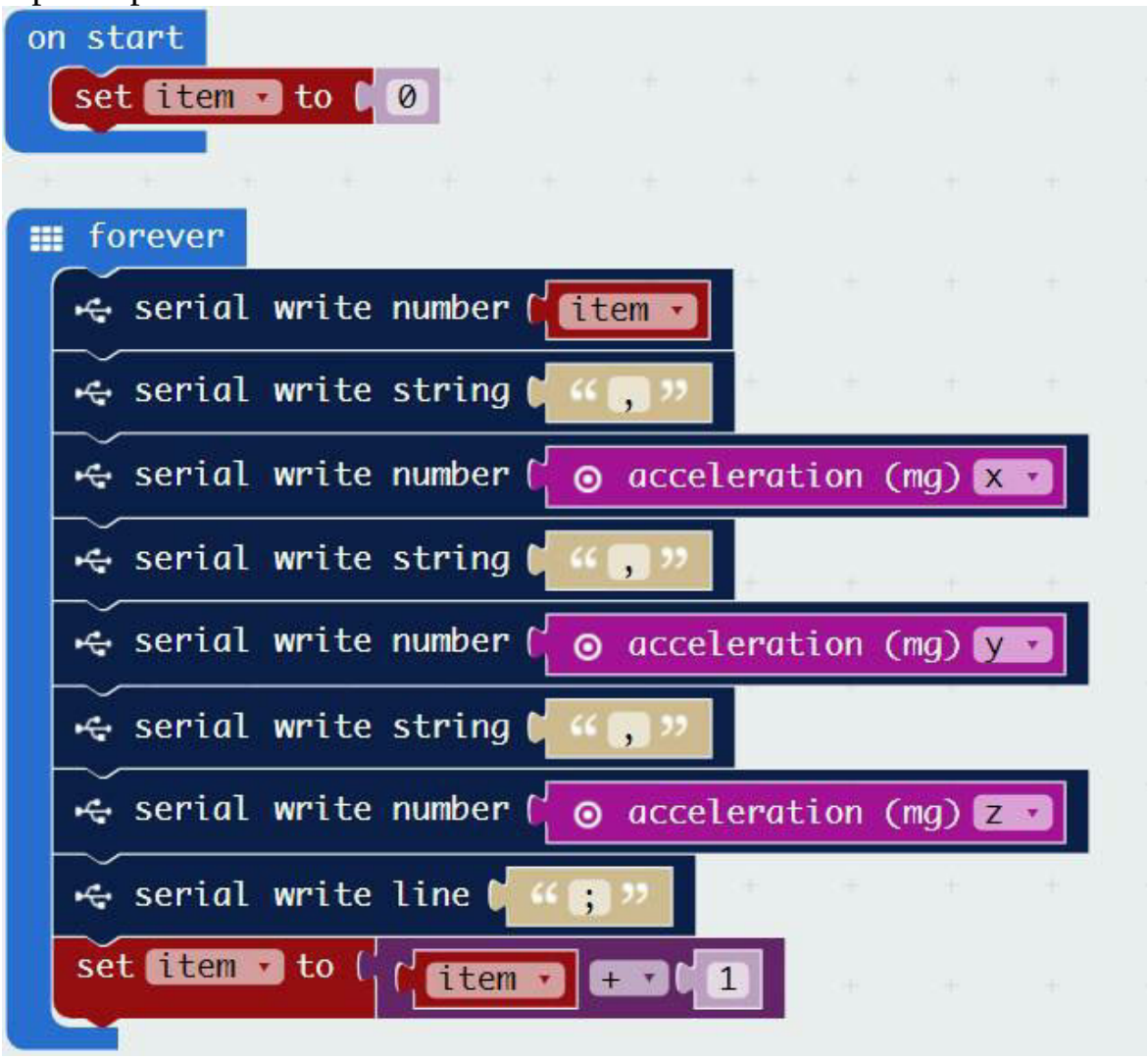


5.8 Используем serial-порт

В главе про Arduino мы уже рассматривали использование последовательного порта для отладки программ. То же самое можно сделать и на BBC Micro:bit.

Шаг 1. Поставить драйвер последовательного порта, ссылка на который есть на странице mbed.com. При подключении платы к компьютеру в системе должен появиться новый COM-порт.

Шаг 2. Добавить компонент Serial port в визуальном редакторе. К примеру, вот такая несложная программа отправляет в порт значения акселерометра:



Отправлять данные можно различными способами, весьма удобен формат CSV (comma separated values), который выглядит примерно так:

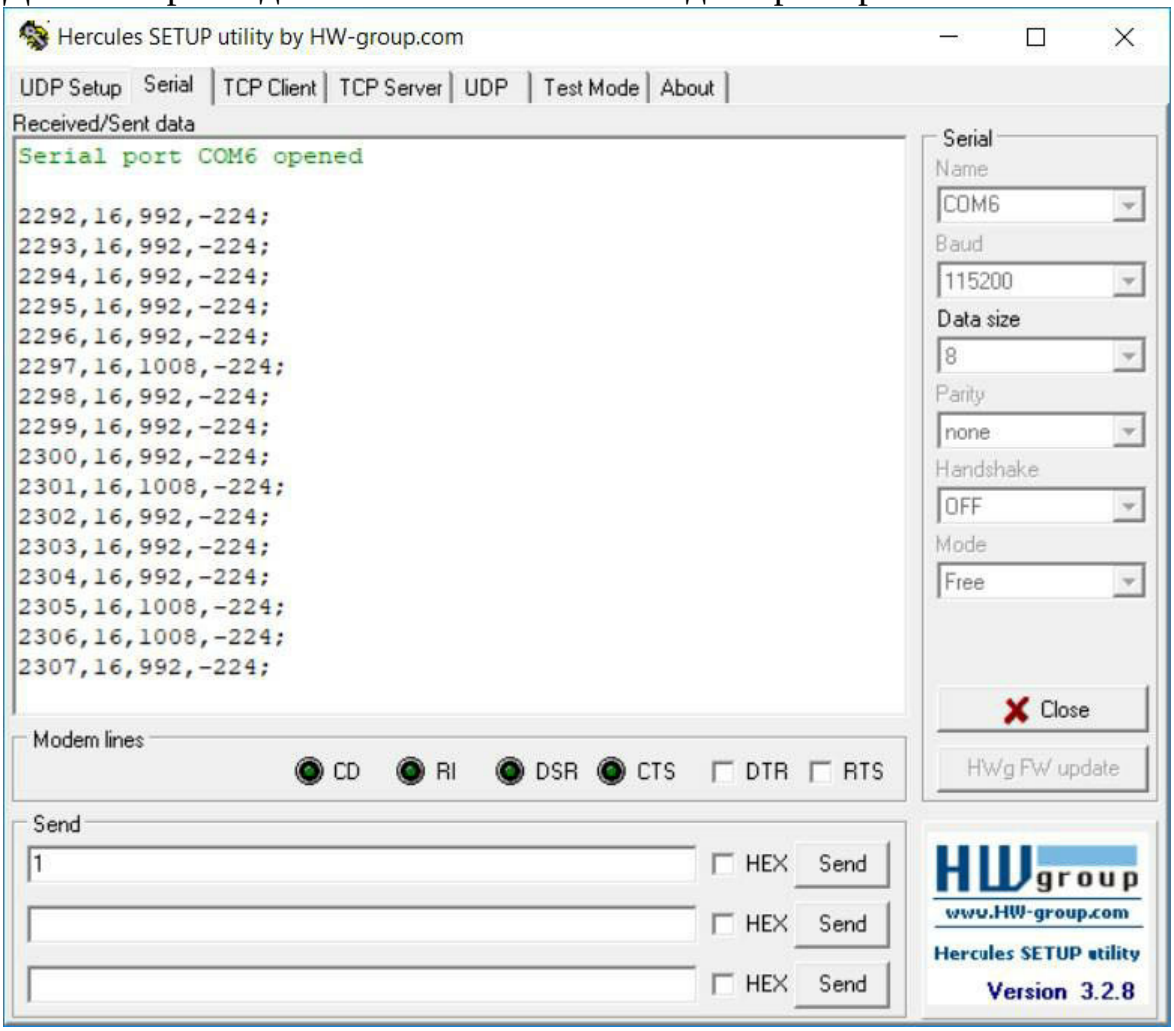
0,-240,608,-768;
1,-288,848,-384;
2,-288,912,-672;
3,-368,672,-640;
4,-224,320,-960;
5,-192,-240,-976;
6,-160,-544,-752;

Формат CSV удобен тем, что для его обработки есть много

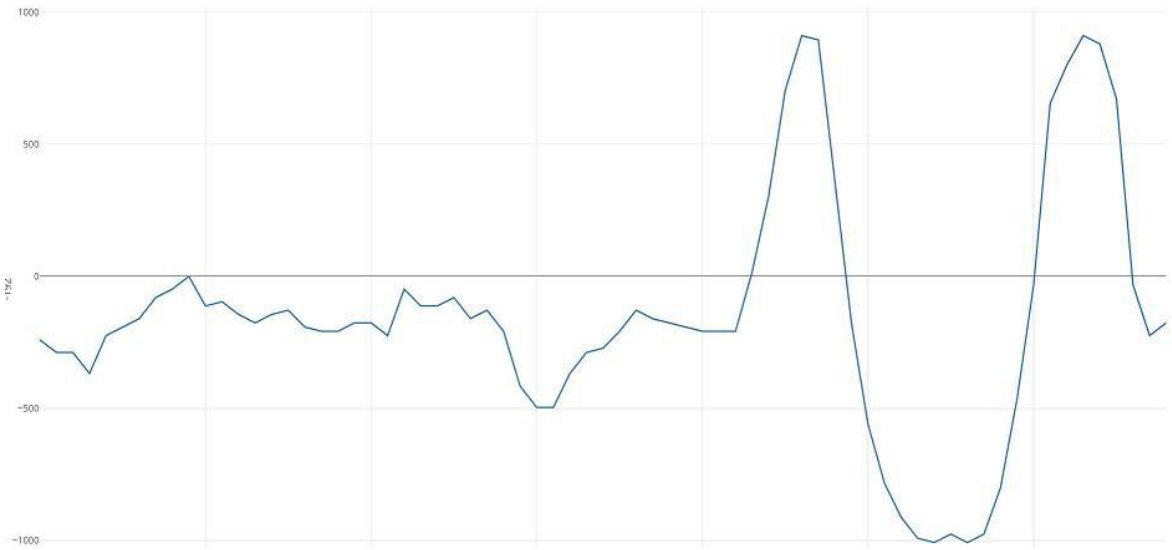
различных программ, например его можно открыть в Microsoft Excel.

Шаг 3. Поставить программу чтения данных. Для этого можно использовать любую программу, умеющую работать с COM-портом, например, Hercules. Предварительно в “Диспетчере устройств” нужно найти новый порт, появляющийся при подключении платы (если порт не появляется, нужно переустановить драйвер).

Данные при подключенной плате выглядят примерно так:

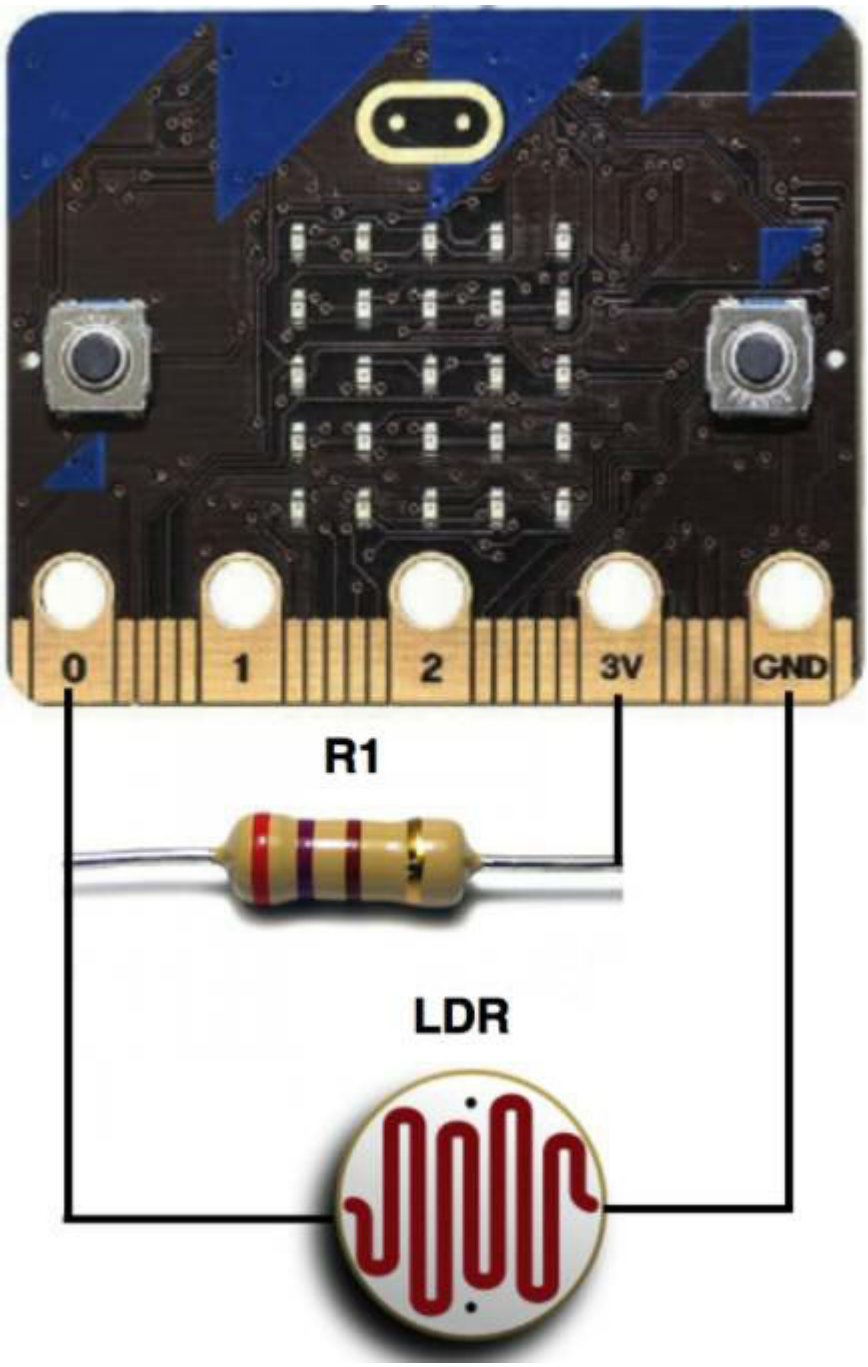


Получив принятые данные, их можно открыть на компьютере. К примеру, вот такой график значений акселерометра получается при покачивании платы в руке (график был построен онлайн с помощью https://plot.ly/create/):



5.9 Подключаем внешние устройства

Как было показано выше, BBC Micro:bit имеет практически все необходимое для автономной работы. Но разумеется, при желании, можно подключить и внешние устройства.

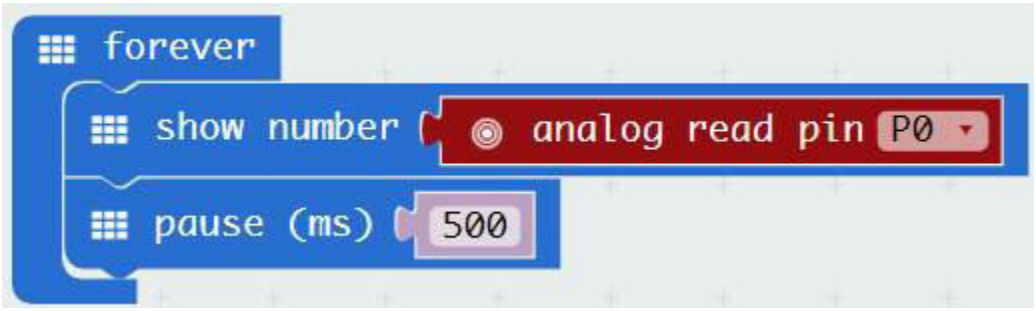


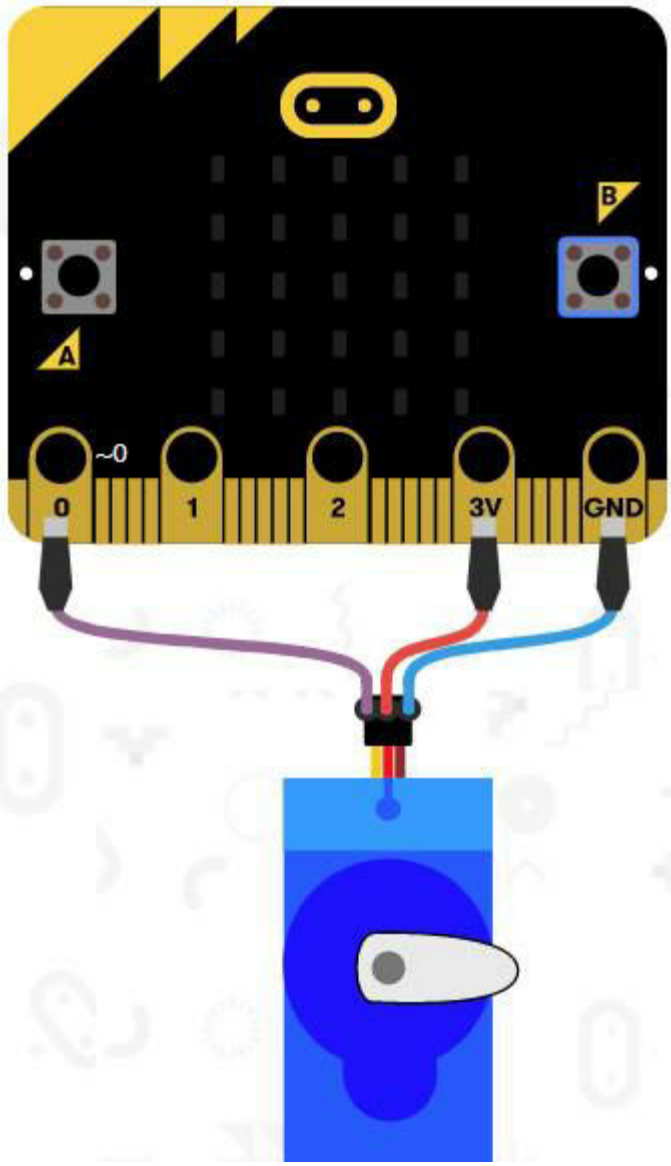
Аналоговые величины можно считывать с помощью функции analog read. Это позволит подключать большое количество аналоговых устройств - датчики температуры, освещенности, влажности и пр.

В простейшем случае, можно подключить переменный резистор между выводами GND и 3V, и при вращении ручки значение будет меняться от 0 до 1023. Справа приведен пример подключения

фоторезистора, что позволит измерять освещенность.

Блок-схема для вывода аналоговой величины на экране показана ниже:





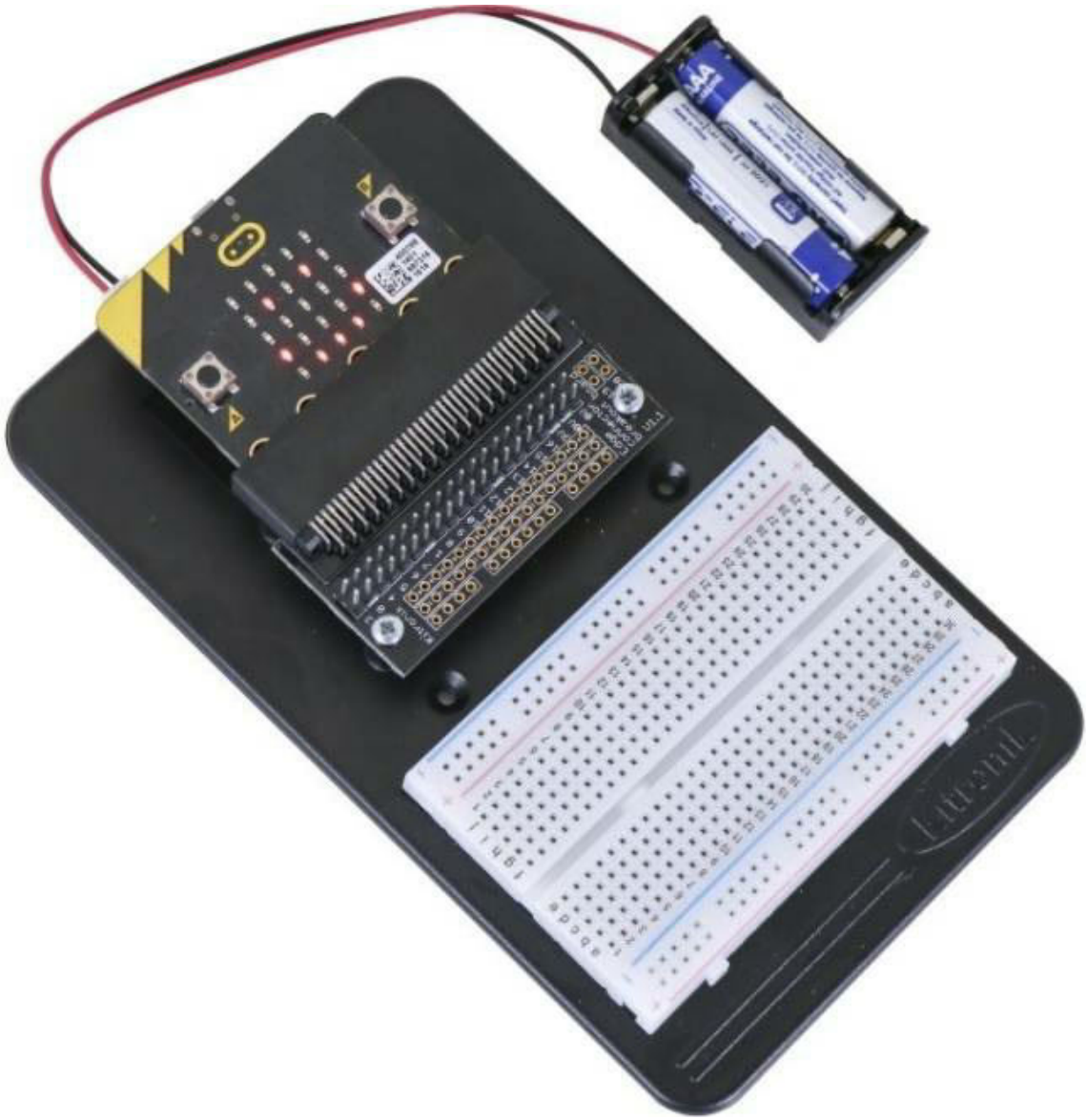
Кроме чтения, есть и аналогичная функция **analog write**,

позволяющая выводить значения, используя ШИМ. Желающие могут перечитать главу 2.4, где это описывалось подробнее.

Функция **servo write pin** позволяет управлять серво-машинкой, которую можно повернуть за заданный угол. Схема подключения показана на рисунке справа.

При желании использовать различные устройства и датчики, проще купить специальную плату расширения, т.к. контакты Micro:bit не очень удобны для самостоятельного подключения.

Можно также приобрести и модуль расширения с макетной платой, как показано на рисунке ниже. Такая плата позволит легко подключать различные устройства - кнопки, переключатели, светодиоды, потенциометры.



Стоит лишь иметь в виду, что BBC Micro:bit изначально ориентирована на начинающих, и делать на ней более-менее сложные проекты нецелесообразно. В том случае, когда возможностей Micro:bit станет мало, проще перейти на Arduino или на ESP32.

На этом данная часть книги закончена. Разумеется, описать все невозможно, но можно надеяться, что приведенный материал станет неплохой точкой для старта. Что-то недостающее или непонятное, уже можно будет найти с помощью Гугла или Яндекса.

По мере появления нового материала, он будет добавляться, стоит следить за обновлением версии книги в начале текста. Последнюю

версию можно скачать на странице https://cloud.mail.ru/public/F6Vf/nY6iSxXcd.