Оглавление

Общая информация о датчиках **4**
Цифровые и аналоговые датчики 4
Работа с датчиками, использующими протокол I2C 4
Работа с прерываниями 5

Инфракрасные датчики отражения **7**

Ультразвуковые дальномеры **8**

Датчик освещённости, цвета, расстояния и жестов APDS-9960 **11**

Датчик цвета TCS34725 **16**

Цифровой "компас" GY-273 на основе микросхемы HMC5883L **18**

Гироскоп и акселерометр GY-521 на микросхеме MPU6050 **20**

OLED дисплей 0,96" 128x64 с интерфейсом I2C на микросхеме SSD1306 **24**
Пример программы с использованием этой библиотеки: 24
Пример программы с использованием библиотек Adafruit: 26
Пример показа картинки с библиотекой OLED_I2C: 28
Пример показа картинки с библиотекой Adafruit: 29
Дополнение: Новые библиотеки для дисплеев I2C. 31

Варианты OLED дисплеев 0,96" 128x64 на микросхеме SSD1306 с интерфейсом SPI **32**

2,4 дюймовый сенсорный экран на микросхеме ILI9341 **36**
Пример программы-рисовалки: 36

Датчик цвета TCS230 **40**
Пример программы для определения цвета. 41

Работа с сервомоторами (сервоприводами) **43**

Использование контроллера Bluetooth **46**

Общая информация о шаговых моторах **48**

Подключение шаговых моторов к драйверам **53**
Подключение 28BYJ-48. 53
Подключение биполярных моторов к драйверам с выводами Step и Dir. 54
Пояснения к схемам. 55
Подключение мотора. 56
Настройка тока. 58

Использование других драйверов. 60
Полезные ссылки. 60

Управление шаговыми моторами 62
Прямое управление (без библиотек). 62
Драйверы Step/Dir. 62
Драйверы типа H-мост (четырёхконтактные). 62
Какой шаг выбрать? 64
Управление через библиотеку AccelStepper. 64
Подключение библиотеки: 65
Создание объекта для мотора: 65
Настройка параметров скорости: 65
Управление через позицию (по количеству шагов): 65
Управление через скорость (шагов в секунду): 66
Управление по скорости и позиции: 66
Другие команды: 66
Примеры. 67
Управление через библиотеку StepperDriver. 70
Примеры. 71
Другие команды библиотеки. 72

Использование сдвигового регистра 74HC595 для расширения портов 75
Схема подключения на примере подключения 8 светодиодов: 75
Выводы и их назначение (сравни со схемой подключения вверху): 76
Как работает эта микросхема: 76

7-сегментный индикатор. 78

Светодиодная матрица 8x8. 81
Подключение. 82
Пишем программу. 84

Сборки светодиодных матриц на микросхеме MAX7219 86
Драйвер MAX7219 и сборки на его основе. 86
Подключение к Ардуино. 88
Библиотеки. 90

Сборки 7-сегментных индикаторов на MAX7219 96
Коды знаков. 96
Библиотеки. 97

Сборки 7-сегментных индикаторов на микросхеме TM1637 101

Расширители портов (мультиплексоры) 104

Расширитель портов с интерфейсом I2C на MCP23017 106

Контакты: https://lotsman.ru, +79058020511, info@lotsman.org

Общая информация о взаимодействии с устройствами

Устройства ввода для Arduino это обычно разного рода датчики (сенсоры). Они очень разнообразны, но принципы работы с ними примерно одинаковы. В простом случае они дают обычный цифровой или аналоговый сигнал, и вы читаете его соответствующим портом.

В более сложных случаях датчик подключается сразу к нескольким портам и используются специальные протоколы I2C или OneWire или что-то ещё (но это редко). В любом случае для "сложных" датчиков обычно есть готовые библиотеки, которые сильно упрощают работу с ними.

Цифровые и аналоговые датчики

Простые датчики бывают цифровые и аналоговые. Цифровые выдают только HIGH или LOW и подключаются, соответственно, к цифровому порту. Информация с них читается обычной функцией digitalRead. Понятно, что такие датчики могут определить только факт некоего события или его отсутствия. Например, датчик движения: даёт HIGH если случилось движение поблизости. Или цифровой датчик освещённости: даёт HIGH если стало слишком темно и LOW если светло (или наоборот). Обычно чувствительность таких датчиков можно настраивать, чтобы они срабатывали именно в нужной ситуации.

Аналоговые датчики могут точно измерить какую-то величину. Подключаются, они, соответственно, к аналоговому порту и читаются функцией analogRead. Например, аналоговый датчик освещённости не просто покажет когда светло или темно, а измерит точное количество света. Аналоговый датчик температуры покажет температуру. Измерение происходит в условных единицах, так как аналоговый порт даёт значения от 0 до 1023. Но потом при желании можно пересчитать это число в градусы, сантиметры или ещё что-то.

Работа с устройствами, использующими протокол I2C

Протокол I2C позволяет подключать к Ардуино довольно сложные устройства всего двумя проводами. На этом протоколе работают многие датчики и устройства вывода.

Контакты I2C называются:

SDA — шина для передачи данных, на Arduino Uno она совмещена с портом A4, а на Leonardo с цифровым портом 2;

SCL — шина тактирования для передачи управляющих сигналов, на Arduino Uno она совмещена с портом A5, а на Leonardo с цифровым портом 3.

Когда контроллер работает в режиме I2C, то указанные порты меняют своё назначение и не могут в это время использоваться как обычные порты. На других разновидностях Arduino для I2C могут использоваться другие порты, так что

нужно всегда смотреть куда подключать ваше устройство и учитывать в программе, что занятые на I2C порты нельзя использовать обычным образом.

Для работы с I2C к программе нужно подключить библиотеку Wire:

#include <Wire.h>.

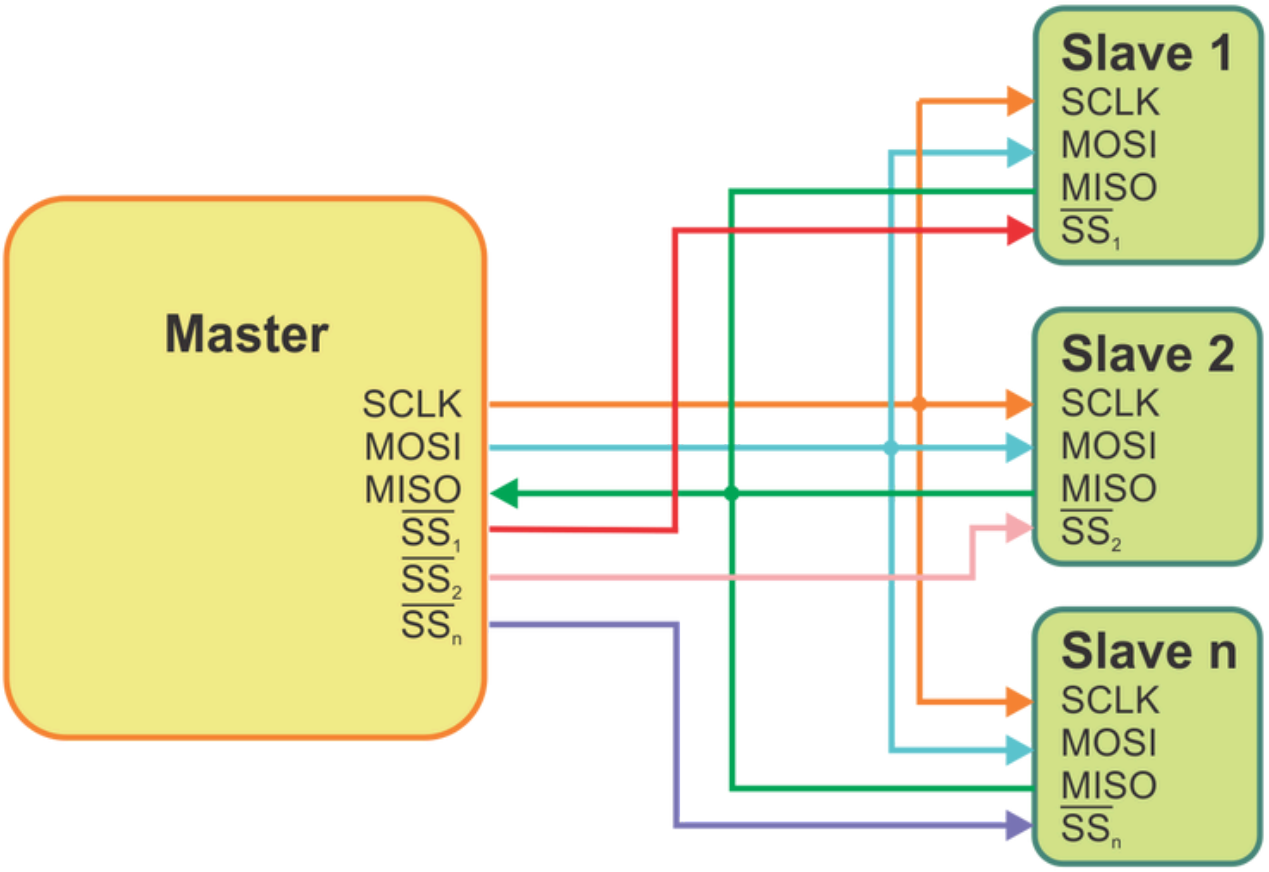
Эта библиотека есть в комплекте Arduino IDE (программа для написания программ), так что отдельно устанавливать её не нужно.

Для работы с конкретными датчиками и другими устройствами часто нужны дополнительные библиотеки, их надо не забыть тоже подключить.

Устройства на I2C имеют адреса, и благодаря этому, можно подключить одновременно несколько таких устройств. Иногда адрес устройства неизвестен, и его нужно как-то узнать. В этом может помочь программа, опубликованная на сайте Arduino: https://playground.arduino.cc/Main/I2cScanner/.

Работа с устройствами, использующими протокол SPI

SPI это ещё один протокол для подключения сложных устройств, правда, в этом случае приходится использовать 4 цифровых порта на первое устройство плюс ещё по одному на каждое последующее. Принципиальная схема подключения (Master — управляющее устройство — это у нас Ардуино, Slave — управляемое устройство):



Ниже поясняется назначение контактов.

SCLK или SCK это Serial Clock — тактовый сигнал.

MOSI (Master Out Slave In) — выход для данных.

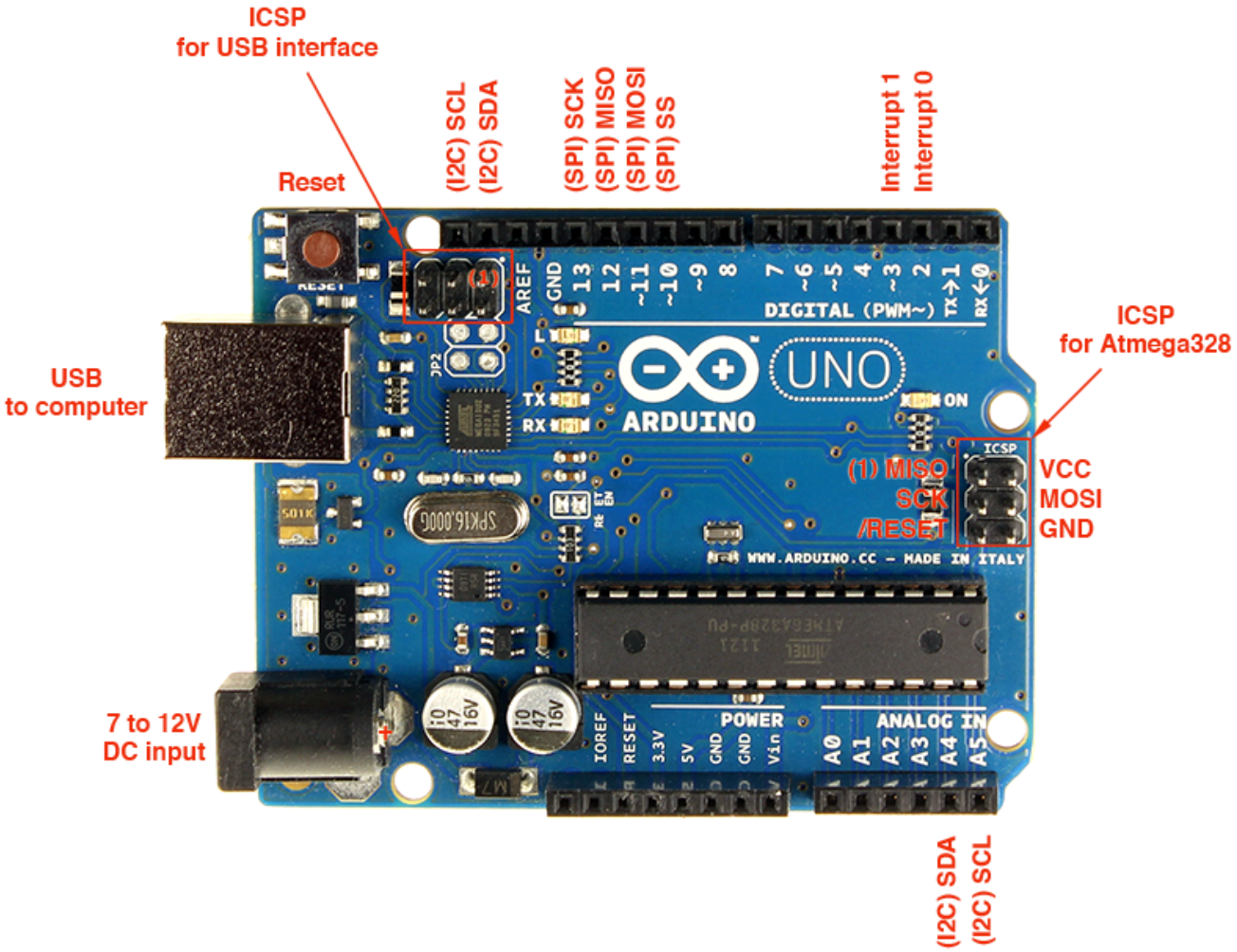
MISO (Master In Slave Out) — вход для данных.

SS (Slave Select) — выбор устройства (слэйва).

Первые три контакта — общие для всех устройств, а SS нужен отдельный для каждого устройства. В нормальном состоянии этот контакт должен быть в состоянии HIGH, а когда "мастеру" нужно пообщаться с этим "слейвом", на его SS подаётся LOW.

Впрочем, редко когда приходится подключать больше одного SPI устройства.

На Ардуино для SCLK, MOSI и MISO закреплены определённые пины, для любой модели их можно посмотреть на подробных схемах. Например, в документе "Порты и питание" на схеме Arduino UNO они помечены оранжевым цветом. Вот ещё схема, где подписаны все эти пины:



Как видно на схеме, контакты SPI совмещены с цифровыми портами:

SCK — пин 13,

MISO — пин 12,

MOSI — пин 11.

Эти же контакты выведены отдельно на группу из 6 контактов (справа) вместе с землёй и питанием (5В).

На Arduino Nano расклад SPI такой же. А вот на Leonardo используется только отдельная группа, цифровые порты не занимаются.

А что же с SS? Куда его подключать? Для SS можно назначить любой цифровой порт. Если у вас одно устройство, рекомендуется использовать пин 10. Во-первых, тогда все SPI порты будут рядом(10, 11, 12, 13). Во-вторых, этот пин рассчитан на работу Arduino в режиме Slave. Именно поэтому на картинке сверху он подписан как SS. В реальной жизни он так никогда не используется, но если ты вдруг настроишь его для чего-нибудь в режиме INPUT, процессор подумает, что он теперь SPI-slave и SPI перестанет работать. Для правильной работы в режиме Master (то есть, обычной работы), этот пин должен быть в режиме вывода (OUTPUT). И если мы его будем использовать для SS, он как раз и будет в режиме вывода.

Для работы с SPI к программе нужно подключить библиотеку SPI:

#include <SPI.h>.

Эта библиотека есть в комплекте Arduino IDE, так что отдельно устанавливать её не нужно.

Работа с прерываниями

Для работы с некоторыми датчиками требуется использовать прерывание. Прерывание это изменение хода выполнения программы в случае прихода нужного сигнала (например, с датчика). Процессор, получив сигнал прерывания, приостанавливает обычную работу и немедленно исполняет какие-то специальные действия, предусмотренные в программе для этого прерывания. Потом исполнение возвращается в то место, на котором оно остановилось.

В программе одновременно могут использоваться несколько прерываний. Но это зависит от типа платы. На Arduino Uno можно использовать два прерывания, на Leonardo их 5, на Mega — 6. У них есть номера, начинающиеся с нуля.

Для сигналов прерываний используются определённые цифровые порты в режиме INPUT.

Соответствие номеров прерываний и портов для разных плат:

Плата Arduino	**int.0**	**int.1**	**int.2**	**int.3**	**int.4**	**int.5**
Uno, Ethernet, Nano, Mini	2	3				
Mega	2	3	21	20	19	18
Leonardo, Micro	3	2	0	1	7	
Due, Zero, MKR1000, 101	**номер прерывания = номеру порта**					

Для создания прерывания используется функция:

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

pin — номер цифрового порта для прерывания.

ISR — (без скобочек!) функция, вызываемая в случае прерывания — она не должна иметь параметров и не должна ничего возвращать. Переменные, которые в ней используются, должны иметь свойство "volatile".

mode — условие прерывания, то есть, событие на порте, которое приводит к прерыванию.

События могут быть такие:

LOW — на порт поступает сигнал LOW.

HIGH — на порт поступает сигнал HIGH.

CHANGE — сигнал, поступающий на порт, меняет значение.

RISING — сигнал меняется с LOW на HIGH.

FALLING — сигнал меняется с HIGH на LOW.

Другой вариант использования:

```
attachInterrupt(interrupt, ISR, mode);
```

interrupt — номер прерывания (по таблице выше).

Функция для отмены прерывания:

```
detachInterrupt(digitalPinToInterrupt(pin));
```

или

```
detachInterrupt(interrupt);
```

Можно также временно приостановить прерывания:

```
noInterrupts();
```

А потом их возобновить:

```
interrupts();
```

Приостанавливать прерывания рекомендуется перед действиями с переменной, которая может изменяться в функции-обработчике прерывания (чтобы прерывание не сработало в процессе записи или чтения переменной) или перед любыми действиями, которые важно исполнить быстро и полностью (чтобы прерывание не прерывало их исполнения). Потом прерывание можно снова включить.

Пример использования прерывания:

Задаём:

```
attachInterrupt(digitalPinToInterrupt(2),what,RISING);
```

Объявляем функцию обработки:

```
void what(){
Serial.println("Чего надо?");
}
```

Подключаем кнопку к порту 2 так, чтобы при нажатии туда подавался сигнал HIGH. Теперь при каждом нажатии будет выводиться сообщение.

Прерывания по таймеру

Бывает, что нужно сделать какое-то неотложное действие не по сигналу снаружи, а в определённый момент времени. Например, измерять расстояние до препятствия каждые полсекунды (чтобы не врезаться) или мигать светодиодом с определённой частотой. В простых случаях достаточно отмерять время функцией millis(), как описано в задании. Но этот способ не годится, если у тебя большой loop, который исполняется дольше, чем нужный тебе отрезок времени. Можно, конечно, понатыкать несколько проверок времени в разных местах loop, но это будет неудобно и ненадёжно.

Есть хорошая новость: в Ардуино предусмотрены прерывания по таймеру. Задаёшь промежуток времени и функцию, которая должна каждый раз вызываться — и всё происходит автоматически.

Но есть и плохая новость: стандартного простого способа работать с этими прерываниями не предусмотрено. Ими нужно управлять через регистры (специальные служебные переменные) и помнить много разных буквочек — имён регистров и их битов. Для тех, кто хочет стать настоящим программистом — вот пара ссылок об этой технологии: видео, переводная статья на хабре и её англоязычный оригинал.

Для тех, кто не хочет погружаться в эти дебри, есть несколько библиотек, которые упрощают работу с прерываниями. Про них и поговорим.

Сначала важные пояснения о принципе работы таймеров.

Таймер — это отдельная часть процессора. Он просто выдаёт сигналы с определённой частотой. В Ардуино, как и в любом компьютере, есть встроенный генератор частоты. В UNO это 16 МГц (мегагерц), то есть 16 миллионов раз в секунду. Таймер может выдавать непосредственно эту частоту или уменьшать её во сколько-то раз (это настраивается).

Таймеров в Ардуино несколько. В Uno или Nano их три. Их обычно называют Timer0, Timer1, Timer2. В Leonardo 4 таймера, в Mega — 6. Есть библиотеки, работающие с одним конкретным таймером, а есть те, где можно выбирать.

Какой таймер надо выбирать (если есть выбор)? Обычно выбирают Timer1, так как он (на Uno) самый лучший, он 16-разрядный, и может отмерить за раз до нескольких секунд, а остальные два 8-разрядные и могут отмерить только сколько-то миллисекунд (точное количество зависит от конкретной библиотеки).

Ещё нужно учитывать, что таймеры в Ардуино вообще-то не для нас лично приделаны, и используя их для своих задач, мы можем помешать работе некоторых стандартных функций Ардуино. Например, для Uno:

Timer0 — используется функциями времени: millis(), micros(), delay(), а также делает сигнал ШИМ на пинах 5 и 6.

Timer1 — используется библиотекой Servo, а также делает ШИМ на пинах 9 и 10.

Timer2 — используется функцией tone(), а также делает ШИМ на пинах 3 и 11.

Именно из-за этого, например, перестаёт работать ШИМ на пинах 3 и 11, если ты используешь tone().

Таким образом, если ты используешь Timer1, то очень вероятно, что сервомоторы будут работать непредсказуемо. Проверяй!

Библиотека TimerOne и её клоны.

Эта библиотека есть в менеджере библиотек Arduino IDE. При желании скачать можно отсюда, почитать пояснения здесь, в том числе про поддержку разных типов Arduino. Там же есть аналогичная библиотека для Timer3, которую рекомендуется использовать с ArduinoMega. В менеджере библиотек есть такие же библиотеки для таймеров 4 и 5 (Mega, Leonardo). Вот специальная библиотека для Леонардо.

Библиотеки эти очень просты в использовании и позволяют делать две вещи: создавать одно (всего!) прерывание и выдавать ШИМ сигнал на пины, связанные с данным таймером (для Timer1 на Uno это будет 9 и 10, для других таймеров и моделей Arduino будут другие пины).

Вот пример на создание прерывания:

```
#include "TimerOne.h"
void setup(){
  Serial.begin(9600);
  Timer1.initialize();//включаем таймер
  Timer1.attachInterrupt(hello,1000000);//включаем прерывание с
  промежутком 1 секунда (1000000 микросекунд) и вызываем по
  прерыванию функцию "hello".
}
void loop(){
//(пока здесь пусто)
}
```

```
void hello(){ //А вот и функция hello, она просто выводит "hello".
  Serial.println("Hello!");
}
```

В примере период работы прерывания указывается в момент назначения прерывания. Но можно указать его уже при инициализации:

```
Timer1.initialize(500000);
Timer1.attachInterrupt(hello);
```

А можно задать отдельно, потом:

```
Timer1.setPeriod(1500000);
```

Можно также останавливать таймер, перезапускать, продолжать:

```
Timer1.start(); //запускает таймер с начала
Timer1.stop(); //останавливает таймер
Timer1.restart();//перезапускает с начала нового периода
Timer1.resume();//возобновляет после остановки с того же момента
```

Можно отключить прерывание:

```
Timer1.detachInterrupt();
```

Можно пользоваться стандартной приостановкой прерывания:

```
noInterrupts();
interrupts();
```

При использовании библиотеки TimerOne, стандартная функция ШИМ (analogWrite() на 9 и 10 пинах работать не будет. Но можно включить ШИМ сигнал на 9 или 10 пине через библиотеку, причём, диапазон будет не от 0 до 255, как для обычного analogWrite, а аж от 0 до 1023. Эта величина называется скважность (duty).

```
Timer1.pwm(pin, duty);//включить ШИМ (пин 9 или 10, duty 0-1023)
Timer1.setPwmDuty(pin, duty);//отдельно изменить скважность
Timer1.disablePwm(pin);//остановить ШИМ
```

Библиотека GyverTimers от Алекса Гайвера.

Подробное описание здесь. Скачать можно здесь.

Эта библиотека более универсальна. Она работает со всеми таймерами на Uno (3 таймера) и на Mega (6 таймеров). Про Leonardo на странице автора информации нет, но скорее всего, работать будет. Про таймеры на Leonardo есть немного здесь.

Особености библиотеки:

Можно выбрать любой таймер.

На каждом таймере можно использовать по-отдельности два канала (A и B), а на Mega даже три (ещё C). Это значит, что можно делать по 2-3 прерывания на каждом таймере.

Однако, период прерываний на каналах A и B одного таймера будет один и тот же. Отличаться может только фаза. То есть, например, оба прерывания вызываются раз в секунду, но в разные моменты: скажем, прерывание на канале B отстаёт от канала A на 0,4 секунды. Фаза измеряется в градусах. Это означает, что фаза 180 соответствует половине периода, а фаза 90 — одной четверти периода.

Не нужно давать команду attachinterrupt и самому придумывать названия функций для обработки прерываний. Всё заранее задано. Например, функция для канала A на таймере 1 будет выглядеть:

ISR(TIMER1_A) {
// ваш код
}

Другие аналогично: только подставить номер таймера и букву канала.

Возможности библиотеки с примерами подробно разобраны на странице автора. Обрати внимание на ограничения по частотам. Для 8-разрядных таймеров минимальная частота 31 герц.

Здесь приведём небольшой пример:

```
#include <GyverTimers.h>// подключаем библиотеку
void setup() {
  Timer1.setFrequency(4);// устанавливаем частоту 4 герца
  //Timer1.setPeriod(250000);// то же самое! частота 4 Гц это период
  250000 микросекунд, то есть, 0,25 секунды (частота равна 1 делить
  на период: 4=1/0,25)
  Timer1.enableISR();// запускаем прерывание (по умолчанию канал A)
  pinMode(13, OUTPUT);// будем мигать
}

void loop() {
  //пока тут пусто
  }

// функция для прерывания на канале A таймера 1
ISR(TIMER1_A) {
  digitalWrite(13, !digitalRead(13));//переключаем светодиод
}
```

Библиотека timer-api.h

Почитать про эту библиотеку и посмотреть примеры можно здесь и здесь (тут можно и скачать). Она позволяет использовать разные таймеры, но не все. На Uno

— два, на Leonardo — один, на Mega — четыре. Особенность: заранее заданный набор частот, из которых можно выбирать.

Библиотека ticker для esp8266 и esp32

В комплекте расширений к Arduino IDE для процессоров esp есть библиотека ticker, позволяющая создавать прерывания по таймеру. Отдельно её устанавливать не нужно, она доступна, если ты установил расширение esp8266 или esp32. Некоторые пояснения и примеры есть здесь. А вот видео. Библиотека проста в базовом использовании, и позволяет создавать несколько объектов-прерываний.

Вот простейший пример.

```
#include <Ticker.h> //подключаем библиотеку
Ticker blinker;//создаём объект

void setup() {
  pinMode(D4, OUTPUT);//будем мигать светодиодом на порте D4
  digitalWrite(D4, LOW);//на всякий случай пока выключим

  blinker.attach(0.3, blink);//создаём прерывание для нашего объекта
  blinker, будем вызывать функцию blink, период вызова 0,3 секунды
}

void loop() {
// тут пока пусто
}

void blink() {// функция обработки прерывания
  digitalWrite(D4,!digitalRead(D4)); //переворачиваем
}
```

В примере использован вариант задания периода в секундах: attach(). Но можно те же 0,3 секунды задать в миллисекундах:

```
 blinker.attach_ms(300, blink);
```

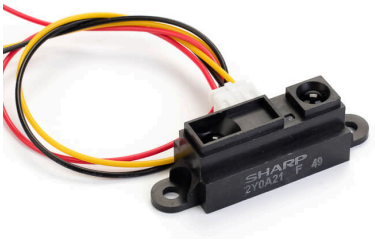
Ещё необычно, что можно функции-обработчику передавать параметр:

```
blinker1.attach(0.3, setpin,1);//передадим функции setpin единицу
blinker2.attach(0.3, setpin,0);//передадим функции setpin ноль
void setpin(bool state) {
  digitalWrite(D4,state); //включаем или выключаем — смотря что в
  пришло в state
}
```

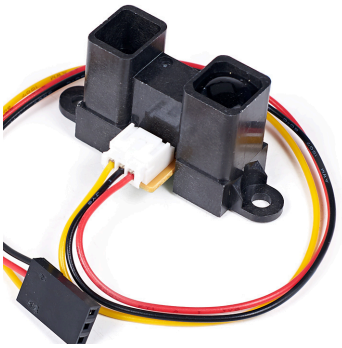
Некоторые дополнительные возможности можно посмотреть в встроенных примерах, хотя там не всё можно сходу понять.

Инфракрасные датчики отражения

Это довольно многообразный класс датчиков: дальномеры, датчики оттенка, датчики препятствия. Они могут быть как цифровые, так и аналоговые. Для использования достаточно просто подключить выход такого датчика к цифровому или аналоговому порту, в зависимости от типа датчика. На некоторых датчиках присутствуют одновременно и цифровой и аналоговый выводы.



Принцип работы таких датчиков заключается в том, что инфракрасный светодиод светит, а отражённый от препятствия свет воспринимается инфракрасным светочувствительным элементом (обычно это фотодиод). Понятно, что отражение тем сильнее, чем ближе препятствие и чем оно светлее.

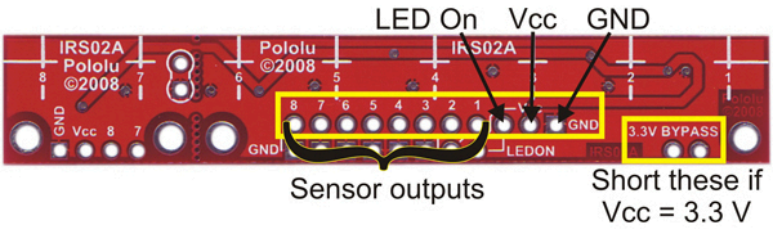


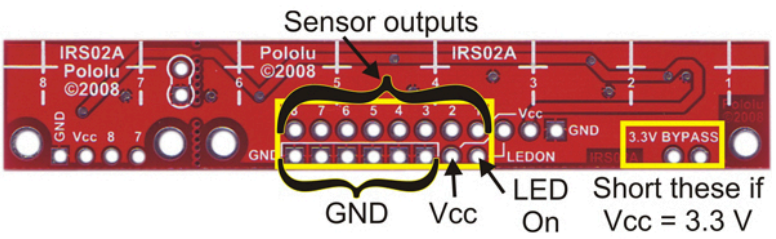
На картинках можно увидеть примеры инфракрасных датчиков. Справа два дальномера, они дают аналоговый сигнал — чем меньше расстояние, тем больше значение. Под ними цифровой датчик препятствия. Он не измеряет расстояние точно, но даёт HIGH, если препятствие близко.

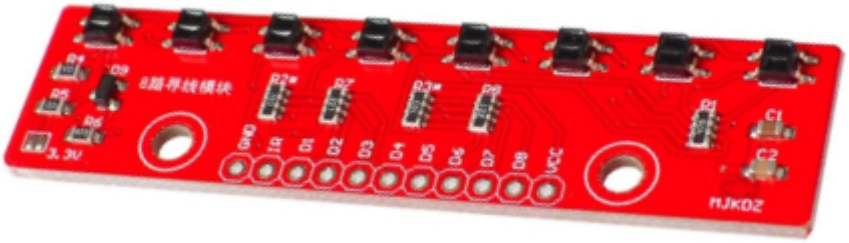


Ниже три датчика оттенков, которые используются обычно как датчики чёрной линии для движения роботов по линии. Слева цифровой датчик (отличает "чёрное" от "белого"), в середине универсальный датчик с цифровым и аналоговым выводами, и справа линейка из 8 датчиков, для продвинутых моделей. Такие линейки выпускаются как с цифровыми, так и с аналоговыми выводами. Из можно рассматривать как отдельные датчики, собранные на одной плате: у каждого датчика свой вывод.









Ультразвуковые дальномеры

Ультразвуковые дальномеры дешевле инфракрасных и используются чаще. Они работают по принципу эхолокации, то есть, излучают ультразвуковой сигнал и измеряют время, за которое приходит "эхо", то есть отражение звука. Зная скорость звука, можно легко рассчитать расстояние. На картинке справа показан самый популярный дальномер HC-SR04.



Вот как он подключается:

VCC — питание (5 вольт).

GND — "земля".

Trig — цифровой вход. Для запуска измерения необходимо подать на этот вход HIGH на 10 микросекунд. Следующее измерение рекомендуется выполнять не ранее чем через 20-30 миллисекунд (хотя, лучше проверить это значение на практике).

Echo — цифровой выход. На этот выход дальномер подаст HIGH и будет держать его, пока не "услышит" эхо. Измерив продолжительность сигнала HIGH, мы узнаем сколько времени звук шёл туда и обратно, а значит сможем вычислить расстояние. На самом деле нужно просто время в микросекундах поделить примерно на 56 и получим расстояние в сантиметрах.

Тестовый скетч для проверки дальномера:

```
void setup() {
pinMode(7,OUTPUT); // 7-й пин в режим вывода (для Trig).
pinMode(8,INPUT);// 8-й пин в режим ввода (для Echo).
Serial.begin(9600); - Готовим последовательный порт.
}

void loop() {
int distance; //объявляем переменную, в которой будем хранить
              // расстояние
digitalWrite(7,LOW);//на всякий случай отправляем LOW на порт
                    // Trig, чтобы избежать помех.
delayMicroseconds(2);//Ждём (на всякий случай) 2 микросекунды.
digitalWrite(7,HIGH);// посылаем сигнал HIGH на Trig дальномера
delayMicroseconds(10);// ждём 10 микросекунд
digitalWrite(7,LOW);// выключаем сигнал
distance=pulseIn(8,HIGH,15000);// замеряем продолжительность (в
                               // микросекундах) ответного
                               // сигнала на контакте Echo и
```

```
                                        // записываем его в переменную
                                        // distance.
  Serial.println(distance/57);// выводим в последовательный порт
значение distance, поделённое на 57, чтобы получить сантиметры и
посмотреть их в мониторе порта)
  delay(500);// ждём полсекунды до следующего измерения.
  }
```

Про функции pulseIn и delayMicroseconds подробнее можно прочитать здесь: https://docs.google.com/document/d/1N3f_otRakcnZCbZYmuM7Kbo6yMNrK2dlprArx0AIptU/edit#bookmark=id.sjtf6y344vi3

Для ленивых на сайте iarduino.ru выложены две функции, которые всё вышесказанное делают за вас. Названия функций iarduino_HC_SR04 и iarduino_HC_SR04_int. Страницы этих библиотек: http://iarduino.ru/file/283.html и http://iarduino.ru/file/17.html. Пример программы:

```
#include <iarduino_HC_SR04.h> //подключаем библиотеку
iarduino_HC_SR04 dalnomer(4,9); //создаём объект дальномер
void setup() {
Serial.begin(9600); //готовим последовательный порт
}

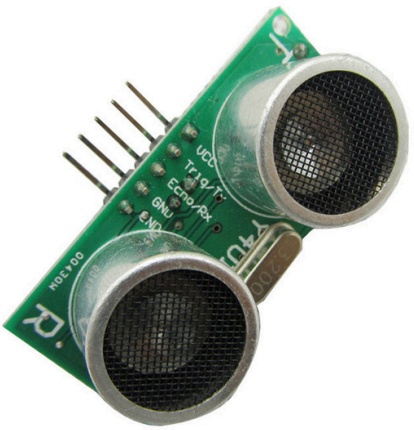
void loop() {
Serial.println(dalnomer.distance());//выводим результат, который
                                    //библиотечная функция
                                    // .distance даёт нам сразу в
                                    //сантиметрах
delay(500);
}
```

Ещё одна библиотека: https://github.com/ErickSimoes/Ultrasonic/releases.

Вот здесь примеры её использования: https://smartmodules.ru/ultrasonic-h

Есть и другие модели ультразвуковых дальномеров. Одни работают по такой же схеме (Trig-Echo), другие по-другому.

Например, модель US-016 выводит обычный аналоговый сигнал (от 0 до 1023) и ещё может работать в двух диапазонах: 1 метр и 3 метра, которые переключаются специальным контактом range (для включения метрового диапазона надо подать на него HIGH).



А модель US-100 (справа) кроме Trig-Echo может подключаться ещё и через последовательный порт, что вообще-то редкий случай для датчиков. Правда, подключается он странно: RX нужно подключить к RX, а TX

к TX. Это китайцы, похоже, напутали, обычно бывает наоборот.

Вот пример программы для US-100.

```
   //для начала включаем библиотеку SoftwareSerial и создаём
// дополнительный последовательный порт. Это не обязательно, но на
// UNO без этого вывод с датчика будет мешать соединению с
// компьютером.
   #include <SoftwareSerial.h>
   SoftwareSerial SerialD(10, 9); // создаём новый объект Serial и
                                  // назначаем ему пины: 10-RX, 9-TX

   // дальше надо учесть, что результат придёт в двух байтах
   unsigned int HighLen = 0;//переменная для старшего байта
                            // результата измерения
   unsigned int LowLen  = 0;//переменная для младшего байта
                            // результата измерения
   unsigned int Len_mm  = 0;//переменная для окончательного
                            // результата
   void setup() {
   // готовим к работе оба последовательных порта
     Serial.begin(9600);
     SerialD.begin(9600);
   }

   void loop() {
     SerialD.flush();      // очищаем буфер порта
     SerialD.write(0X55); // отправляем в дальномер специальный
                          // символ, который значит "начать
                          // измерение" (как сигнал Trig)
     delay(50);           // ждём результат
     if (SerialD.available() >= 2) //проверяем, готовы ли два байта
     {
       HighLen = SerialD.read();  //читаем старший байт
       LowLen  = SerialD.read();  //читаем младший байт
       Len_mm  = HighLen*256+LowLen; //объединяем два байта в
                                     // одно число и записываем
                                     // результат
       {
   // выводим результат красиво в последовательный порт
         Serial.print("Present Length is: ");
         Serial.print(Len_mm, DEC);
         Serial.println("mm");
       }
     }
     delay(100);    //делаем задержку перед следующим измерением
   }
```

Датчик освещённости, цвета, расстояния и жестов APDS-9960

ВНИМАНИЕ!!! Этот датчик питается от 3,3 В (не от 5В !!!). Будьте внимательны при подключении!

Есть две разновидности этого датчика: с 5 контактами и с 6. В том, что с шестью, есть светодиод для подсветки, он и питается от 6-го контакта.

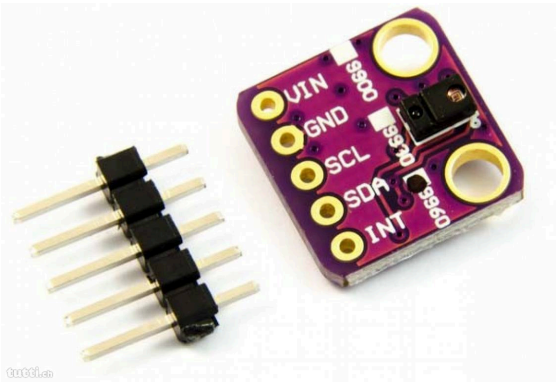
Пять основных контактов:

VIN — питание 3,3 В;

GND — "земля";

SCL и SDA — порты I2C;

INT — порт для сигнала прерывания[1], подключается к одному из цифровых портов, поддерживающих прерывания.



Для использования датчика нужно установить и подключить специальную библиотеку <SparkFun_APDS9960.h>.

Пример распознавания жестов (вверх, вниз, вправо, влево, удаление, приближение)

```
//подключаем библиотеку Wire
#include <Wire.h>
//подключаем библиотеку для этого сенсора
#include <SparkFun_APDS9960.h>
//объявляем переменную apds специального типа
SparkFun_APDS9960 apds = SparkFun_APDS9960();

void setup() {
//готовим последовательный порт для вывода результата
  Serial.begin(9600);
//инициализируем датчик в целом, а потом функцию жестов без
//прерываний (значение false)
  apds.init();
  apds.enableGestureSensor(false);
}
void loop() {
//если информация с датчика жестов доступна...
  if (apds.isGestureAvailable()) {
//выводим результат в последовательный порт
  Serial.println(apds.readGesture());
 }
}
```

[1] Прерывание — изменение хода выполнения программы по сигналу, с помощью которого датчик может "отвлечь" процессор от текущей работы для более срочного дела (обработки какого-то события).

Результат в этом примере будет выведен в виде числа от 0 до 6. В библиотеке также определены константы, которые равны этим числам.

Направление	**Число**	**Константа**
Не распознано	0	DIR_NONE
Влево	1	DIR_LEFT
Вправо	2	DIR_RIGHT
Вверх	3	DIR_UP
Вниз	4	DIR_DOWN
Приближение	5	DIR_NEAR
Удаление	6	DIR_FAR

В приведённом выше примере определение жестов — это единственное, что делает программа. На практике эта функция используется для управления чем-то, поэтому она обычно вызывается в нужный момент через прерывание.

Пример с прерыванием (поясняются только новые строчки)**:**

```
#include <Wire.h>
#include <SparkFun_APDS9960.h>
//объявляем вспомогательную переменную и присваиваем ей 0
int flag = 0;
SparkFun_APDS9960 apds = SparkFun_APDS9960();

void setup() {
// готовим цифровой порт 2 для работы в режиме ввода
 pinMode(2, INPUT);
 Serial.begin(9600);
//назначаем прерывание на порте 2, условие срабатывания – падение
//сигнала (переход от HIGH к LOW), для обработки вызываем функцию
//whattodo
 attachInterrupt(digitalPinToInterrupt(2), whattodo, FALLING);
 apds.init();
//включаем функцию жестов с использованием прерываний (true)
 apds.enableGestureSensor(true);
}
void loop() {
  //если переменная flag равна 1 (смотри функцию whattodo)...
if( flag == 1 ) {
```

```
//временно отключаем прерывание, чтобы оно не мешало, пока мы
//обрабатываем один жест
    detachInterrupt(0);
    if ( apds.isGestureAvailable() ) {
    Serial.println(apds.readGesture());
//возвращаем переменной flag значение 1
    flag = 0;
// снова включаем прерывание
    attachInterrupt(digitalPinToInterrupt(2), whattodo, FALLING);
}
}
}
//функция, которая вызывается прерыванием: она просто
//присваивает переменной flag значение 1
void whattodo(){
 flag = 1;
}
```

Пример определения уровня освещения

```
#include <Wire.h>
#include <SparkFun_APDS9960.h>
SparkFun_APDS9960 apds = SparkFun_APDS9960();
// объявляем переменные для записи уровней освещения: полного и
//по цветовым составляющим, unit16_t это такой тип данных.
uint16_t ambient_light = 0;
uint16_t red_light = 0;
uint16_t green_light = 0;
uint16_t blue_light = 0;

void setup() {
  Serial.begin(9600);
  apds.init();
// Включаем датчик света (false значит не включать режим
// прерываний)
 apds.enableLightSensor(false)
// рекомендуется задержка, чтобы датчик успел подготовиться
  delay(500);
}

void loop() {
// Читаем значения освещённости и записываем в переменные
 apds.readAmbientLight(ambient_light);
 apds.readRedLight(red_light);
 apds.readGreenLight(green_light);
 apds.readBlueLight(blue_light);
```

```
//Выводим значения в последовательный порт
    Serial.print("Ambient: ");
    Serial.print(ambient_light);
    Serial.print(" Red: ");
    Serial.print(red_light);
    Serial.print(" Green: ");
    Serial.print(green_light);
    Serial.print(" Blue: ");
    Serial.println(blue_light);
// Ждём 1 секунду перед новым выводом
  delay(1000);
}
```

Датчик также может выдавать сигнал прерывания (типа FALLING), когда общая яркость света больше или меньше установленных значений. Для использования этой возможности пригодятся такие функции:

`apds.enableLightSensor(true)` — включаем датчик света в режиме прерываний

`apds.setLightIntLowThreshold(значение)` — установить значение нижнего порога срабатывания прерывания (в условных единицах, которые даёт датчик);

`apds.setLightIntHighThreshold(значение)` — установить значение верхнего порога;

`apds.setAmbientLightIntEnable(1)` — включение режима срабатывания прерываний по уровню света;

`apds.clearAmbientLightInt()` — очистить прерывание (нужно это обязательно делать в функции, обрабатывающей прерывание, чтобы потом могло сработать следующее прерывание).

Полный пример можно посмотреть в примерах из библиотеки.

Датчик также умеет измерять расстояние, правда, в небольших пределах, примерно от 5 до 25 см. Результат выражается в условных единицах от 0 (дальнее расстояние) до 255 (ближнее).

Пример работы сенсора расстояния (приближения)

```
#include <Wire.h>
#include <SparkFun_APDS9960.h>
SparkFun_APDS9960 apds = SparkFun_APDS9960();
//готовим переменную для записи расстояния (именно типа unit8_t)
uint8_t proximity_data = 0;

void setup() {
  Serial.begin(9600);
  apds.init();
```

```
  // Включаем сенсор расстояния (false значит не включать режим
  // прерываний)
  apds.enableProximitySensor(false);
    }
void loop() {
// Читаем значение расстояния и записываем в переменную
  apds.readProximity(proximity_data);
//Выводим результат в последовательный порт
  Serial.print("Proximity: ");
  Serial.println(proximity_data);
// Делаем паузу перед следующим измерением
  delay(250);
}
```

Датчик расстояний также может выдавать сигнал прерывания (типа FALLING) при выходе значения за пределы установленных границ. Для использования этого режима нужны следующие функции:

`apds.enableProximitySensor(true)` — включаем датчик расстояния в режиме прерываний;

`apds.setProximityIntLowThreshold(значение)` — установить значение нижнего порога срабатывания прерывания (в условных единицах, которые даёт датчик);

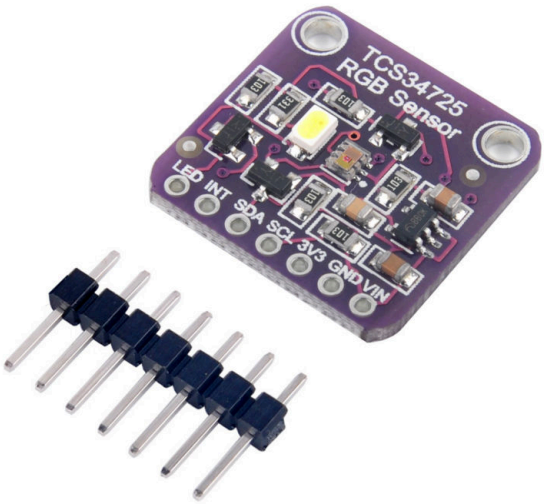
`apds.setProximityIntHighThreshold(значение)` — установить значение верхнего порога;

`apds.clearProximityInt()` — очистить прерывание (нужно это обязательно делать в функции, обрабатывающей прерывание, чтобы потом могло сработать следующее прерывание).

Полный пример можно посмотреть в примерах из библиотеки.

Датчик цвета TCS34725

Этот датчик похож по принципу работы на TCS230, но работает через I2C интерфейс и светодиод подсветки у него один.



Назначение контактов:

VIN — питание 5 В;

GND — "земля";

3V3 — питание 3,3 В;

SCL и SDA — порты I2C;

INT — порт для сигнала прерывания[2];

LED — "выключатель" подсветки: по умолчанию светодиод светится, но если подать на этот контакт LOW, то он погаснет.

Для работы с модулем можно установить библиотеку "Adafruit TCS34725".

В библиотеке, кроме прочего, есть пример программы, заставляющей RGB светодиод светиться вычисленным цветом.

Пример программы:

```
#include <Wire.h>
#include "Adafruit_TCS34725.h"
// Ниже для примера строка для инициализации с параметрами по
// умолчанию: время измерения 2.4 мс, усиление 1x (то есть, без
// усиления).

//Adafruit_TCS34725 tcs = Adafruit_TCS34725();
// Инициализация модуля с указанными параметрами. Здесь указано
// время измерения 700 мс и усиление 1x. Другие возможные значения
// времени: 24, 50, 101, 154 мс, усиления: 4X, 16X, 60X.
Adafruit_TCS34725 tcs =
Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_700MS, TCS34725_GAIN_1X);

void setup() {
// Готовим последовательный порт для вывода результатов.
  Serial.begin(9600);
// Включаем датчик.
tcs.begin();
}
```

[2] Прерывание — изменение хода выполнения программы по сигналу, с помощью которого датчик может "отвлечь" процессор от текущей работы для более срочного дела (обработки какого-то события).

```
void loop() {
// Готовим переменные типа unit16_t для записи разных значений.
  uint16_t r, g, b, c, colorTemp, lux;

// Получаем с датчика 4 значения: красный, зелёный, синий и общий
// (без светофильтра) и записываем их в соответствующие переменные.
  tcs.getRawData(&r, &g, &b, &c);
// Специальной функцией можно вычислить цветовую температуру.
  colorTemp = tcs.calculateColorTemperature(r, g, b);
// Ещё можно вычислить освещённость, она измеряется в люксах, но
// здесь, конечно, значения условные.
  lux = tcs.calculateLux(r, g, b);

// Выводим все значения в последовательный порт для просмотра на
// экране компьютера.
  Serial.print("Color Temp: "); Serial.print(colorTemp);
  Serial.print(" K - ");
  Serial.print("Lux: "); Serial.print(lux); Serial.print(" - ");
  Serial.print("R: "); Serial.print(r); Serial.print(" ");
  Serial.print("G: "); Serial.print(g); Serial.print(" ");
  Serial.print("B: "); Serial.print(b); Serial.print(" ");
  Serial.print("C: "); Serial.print(c); Serial.print(" ");
  Serial.println(" ");
}
```

Цифровой "компас" GY-273 на основе микросхемы HMC5883L

Это трёхмерный датчик магнитного поля, который обычно используют в роли компаса. Он измеряет магнитное поле по трём осям (x, y, z). Ноль по оси X соответствует направлению на север. Для работы с ним можно использовать готовую библиотеку HMC5883L, однако, их существует несколько версий. В примере ниже используется версия отсюда:



https://github.com/jarzebski/Arduino-HMC5883L.

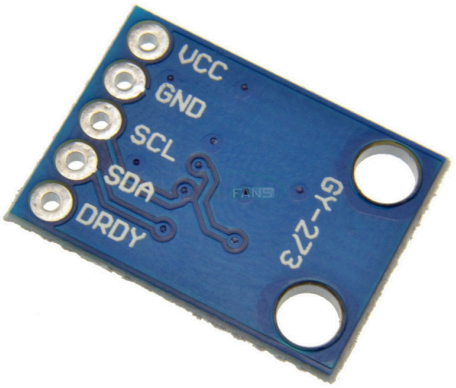
Назначения контактов:

VCC — питание 5 В;

GND — "земля";

SCL и SDA — выводы I2C;

DRDY — вывод сигнала готовности к работе (HIGH означает готовность к измерению), использовать этот выход не обязательно.



Пример программы-компаса

```
#include <Wire.h>
#include <HMC5883L.h>
// Создаём объект-компас типа  HMC5883L
HMC5883L compass;

void setup()
{
  Serial.begin(9600);
// Инициализируем объект HMC5883L
compass.begin();
// Устанавливаем диапазон измерений магнитного поля (варианты этой и
// других настроек можно посмотреть в примере simple из библиотеки)
  compass.setRange(HMC5883L_RANGE_1_3GA);
// Устанавливаем режим измерения (непрерывный)
  compass.setMeasurementMode(HMC5883L_CONTINOUS);
// Устанавливаем скорость (частоту)
  compass.setDataRate(HMC5883L_DATARATE_75HZ);
// Устанавливаем число измерений
  compass.setSamples(HMC5883L_SAMPLES_8);
// Устанавливаем поправку, которую можно определить, запустив пример
// HMC5883L_calibration из библиотеки.
  compass.setOffset(121, -54);
}
```

```
void loop()
{
// Считываем с датчика показатели в контейнер типа vector (там будут
// данные по трём осям)
  Vector norm = compass.readNormalize();
// Вычисляем азимут (угол от направления на север), используя
// функцию арктангенс. Результат пока в радианах.
  float heading = atan2(norm.YAxis, norm.XAxis);
// Вносим поправку на магнитное склонение, его можно узнать для
// своей местности на разных сайтах. Сначала переводим его в
// радианы, потом добавляем к азимуту. Для Екатеринбурга склонение
// примерно 15 градусов. Формула: (град+(минуты/60))/(180/M_PI);
  float declinationAngle = 15 / (180 / M_PI);
  heading += declinationAngle;
// Проверяем, чтобы угол был от 0 до 360 градусов и если он больше
// или меньше — добавляем или вычитаем 360 градусов (или 2пи
// радиан).
  if (heading < 0)
  {
    heading += 2 * PI;
  }
  if (heading > 2 * PI)
  {
    heading -= 2 * PI;
  }
// Переводим радианы в градусы.
  float headingDegrees = heading * 180/M_PI;
// Вносим хитрую поправку, нужную из-за неточности датчика.
// Результат в переменной fixedHeadingDegrees.
  float fixedHeadingDegrees;
  if (headingDegrees >= 1 && headingDegrees < 240)
  {
    fixedHeadingDegrees = map(headingDegrees, 0, 239, 0, 179);
  } else
  if (headingDegrees >= 240)
  {
    fixedHeadingDegrees = map(headingDegrees, 240, 360, 180, 360);
  }
// Округляем результат в новую переменную, чтобы не было дробей.
  int smoothHeadingDegrees = round(fixedHeadingDegrees);
// Выводим результат
  Serial.print("Heading: ");
  Serial.print(smoothHeadingDegrees);
  Serial.println();
```

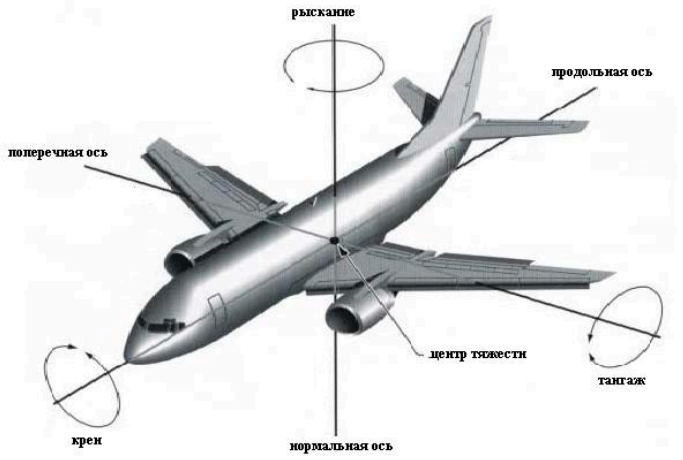
Гироскоп и акселерометр GY-521 на микросхеме MPU6050

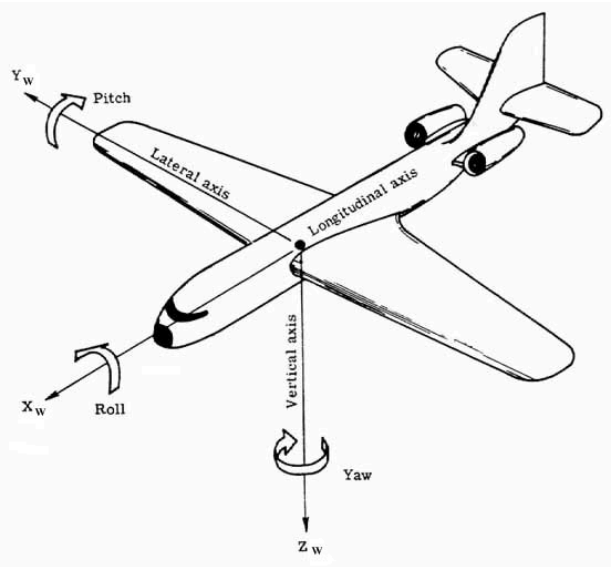
Гироскоп это прибор для определения ориентации объекта в пространстве, для отслеживания изменений ориентации. Акселерометр — прибор для измерения ускорения или, другими словами, действующей на объект силы. Когда объект разгоняется или замедляется — прибор измерит ускорение, но когда объект неподвижен, на него тоже действует сила — сила тяжести. Мы знаем, что она направлена всегда вниз, а значит, по её направленности по отношению к объекту тоже можно установить его ориентацию в пространстве.



На этих принципах и основана работа датчика MPU6050: гироскоп и акселерометр в нём дополняют друг друга и могут работать как вместе, так и по отдельности. Кроме того, в датчик зачем-то встроен термометр. :)

С этого датчика можно получать "сырые" данные об ускорении и изменении ориентации по трём осям, но использовать такие данные трудно. Однако, есть библиотеки, превращающие эти данные в готовые углы ориентации объекта в пространстве. Это гораздо удобнее. Традиционно эти углы называются: крен (roll), тангаж (pitch) и рыскание (yaw):





Такие библиотеки бывают двух видов.

В первых пересчёт сырых данных производится в самой программе, то есть, силами процессора, установленного на Ардуино. Пример — библиотека Kalman: https://github.com/TKJElectronics/KalmanFilter. Или очень удобная библиотека MPU6050 того же польского автора, чью библиотеку мы использовали для компаса: https://github.com/jarzebski/Arduino-MPU6050. В других библиотеках используется способность микросхемы MPU6050 самостоятельно обрабатывать данные встроенным процессором DMP (Digital Motion Processor). В этом случае специальная программа для DMP загружается в датчик, и он возвращает уже

готовые вычисленные значения. Это разгружает центральный процессор. Пример — библиотека MPU6050 из набора I2Cdev: https://github.com/jrowberg/i2cdevlib. Проблема в том, что большинство библиотек для этого датчика имеют одинаковое название, а значит, их нельзя установить одновременно.

Пример работы датчика по технологии DMP (загружаемая программа)

```
//Подключаем библиотеки. Вторая — это программа для DMP. Библиотека
// Wire загрузится автоматически при необходимости
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
// Создаём объект типа MPU6050
MPU6050 mpu;
// Назначаем порт для прерывания.
#define INTERRUPT_PIN 2
// Объявляем нужные переменные.
bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; //return status after each device operation (0 =
success, !0 = error)
uint16_t packetSize;//expected DMP packet size (default is 42 bytes)
uint16_t fifoCount; // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer
// Переменные для информации об ориентации в пространстве.
Quaternion q;        // [w, x, y, z]         quaternion container
VectorFloat gravity;// [x, y, z]            gravity vector
float ypr[3]; // [yaw, pitch, roll]   yaw/pitch/roll container and
gravity vector
// Специальная переменная, сигнализирующая о срабатывании прерывания
volatile bool mpuInterrupt = false;
// Функция обработки прерывания (просто меняется значение сигнальной
//  переменной)
void dmpDataReady() {
    mpuInterrupt = true;
}

void setup() {
// Настройка нужных параметров в зависимости от используемой
// библиотеки.
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
   Wire.begin();
   Wire.setClock(400000); // 400kHz I2C clock. Comment this line if
having compilation difficulties
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
#endif
```

```
// Включаем последовательный порт (скорость можно поставить другую)
    Serial.begin(115200);
// Инициализируем датчик.
    mpu.initialize();
// Готовим порт для прерывания.
    pinMode(INTERRUPT_PIN, INPUT);
// Инициализируем DMP и если всё хорошо, то записывается 0 в
devStatus.
    devStatus = mpu.dmpInitialize();

// Здесь можно вводить поправки, но как их получить это вопрос.
   mpu.setXGyroOffset(220);
   mpu.setYGyroOffset(76);
   mpu.setZGyroOffset(-85);
   mpu.setZAccelOffset(1788);//1688 factory default for my test chip

// Проверяем, что DMP готов к работе.
    if (devStatus == 0) {
// Включаем DMP.
        mpu.setDMPEnabled(true);
// Настраиваем прерывание типа RISING.
attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady,
RISING);
// Записываем состояние прерывания в переменную
        mpuIntStatus = mpu.getIntStatus();
// Показываем через переменную, что DMP готов
        dmpReady = true;
// Определяем размер пакета данных, чтобы потом проверять его
целостность.
        packetSize = mpu.dmpGetFIFOPacketSize();
  }
}

void loop() {
// Проверяем готовность DMP
    if (!dmpReady) return;
// Ничего не делаем, пока нет сигнала прерывания и размер информации
// в буфере меньше размера пакета (то есть, информация не готова).
    while (!mpuInterrupt && fifoCount < packetSize) {
    }
// Прерывание появилось, данные готовы. Начинаем обрабатывать. Для
// начала сбрасываем обратно метку прерывания.
    mpuInterrupt = false;
// Записываем состояние прерывания в переменную
    mpuIntStatus = mpu.getIntStatus();
```

```
// Читаем пакет из буфера.
     mpu.getFIFOBytes(fifoBuffer, packetSize);
//Получаем нужные значения и выводим рыскание, тангаж и крен в порт.
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
}
```

OLED дисплей 0,96" 128x64 с интерфейсом I2C на микросхеме SSD1306



Этот дисплей хорош маленьким размером и простотой подключения. На него можно выводить любой текст и графику. У представленного здесь варианта всего 4 контакта: 2 стандартных контакта I2C, плюс земля и питание. Есть разновидности этого дисплея и на других интерфейсах.

Для работы с дисплеем существует несколько библиотек, мы рассмотрим две. В оригинале эти библиотеки не поддерживают русские буквы, но есть доработанные версии с русскими шрифтами.

Первая библиотека — OLED_I2C. Мы используем модификацию с добавленным русским и украинским шрифтом:

http://arduino-project.net/russkie-ukrainskie-shrifty-oled-i2c/

К этой библиотеке есть хорошая инструкция:

https://drive.google.com/file/d/0B743SQ4GE4ggQk4tX2hIaEdHVU0/view?usp=sharing.

Для вывода русских букв их приходится заменять на латинские с соответствующих клавиш. Буквы ё нет, вместо неё пишется "э".

Пример программы с использованием этой библиотеки:

```
#include <OLED_I2C.h>
// Создаём объект типа OLED, указываем порты I2C интерфейса: на
// Arduino UNO и NANO порт SDA соответствует A4, SCL - A5; Для
// стандартных портов можно просто написать SDA, SCL.
OLED  myOLED(SDA, SCL);
//Подключаем русский и латинский шрифты
extern uint8_t RusFont[];
extern uint8_t SmallFont[];

void setup() {
//Инициализируем дисплей.
  myOLED.begin();
}
void loop() {
  myOLED.clrScr(); // Очищаем экран.
  myOLED.setFont(RusFont); // Включаем русский шрифт.
// Выводим надпись "Русский язык" на 0 пикселей сверху и по центру
// по горизонтали.
  myOLED.print("Heccrbq ihban", CENTER, 0);
 // Выводим надпись "А Б В Г Д Е Ж З И Й" на 12 пикселей от верха.
```

```
  myOLED.print("F < D U L T : P B Q", CENTER, 12);
// Выводим ту же надпись маленькими буквами на 22 пикселя от верха.
    myOLED.print("f , d u l t ; p b q", CENTER, 22);
// Выводим надпись "Э Ю Я" на 32 пикселя от верха.
  myOLED.print("~ > Z", CENTER, 32);
// То же самое маленькими буквами на 42 пикселя от верха.
    myOLED.print("` . z", CENTER, 42);
  myOLED.setFont(SmallFont); // Включаем латинский шрифт.
  myOLED.print("Arduino", CENTER, 52); // Выводим надпись "Arduino".
// Обновляем информацию на дисплее (до этого все действия
// производились с буфером экрана, и только сейчас мы отправляем всё
// на экран).
  myOLED.update();
  delay(3000); // Пауза 3 секунды.
// Очищаем экран
  myOLED.clrScr();
// рисуем прямоугольник со скруглёнными углами с координатой левого
// верхнего угла (5,5), а нижнего правого (100,50).
  myOLED.drawRoundRect(5, 5, 100, 50);
  myOLED.update();
  delay(3000);
}
```

Вторая библиотека — Adafruit_SSD1306. Она используется в паре с библиотекой Adafruit-GFX. Эта библиотека несколько сложнее в управлении, у неё больше возможностей. К ней есть модификация с русскими буквами:

http://arduino.ru/forum/programmirovanie/rusifikatsiya-biblioteki-adafruit-gfx-i-vyvod-russkikh-bukv-na-displei-v-kodi.

Эта модификация позволяет писать русские буквы прямо в программе, присутствует в том числе буква "ё". Однако, выводить их надо через специальную функцию utf8rus, которую нужно добавить к скетчу или положить в папку скетча в отдельном файле. (Ссылка на исправленную версию библиотеки.)

Чтобы программа правильно работала с нашим дисплеем (128x64 пикселей), нужно проверить, чтобы в файле Adafruit_SSD1306.h (в папке библиотеки) была раскомментирована нужная строка (одна из трёх, соответствующих разным типам дисплеев, две ненужные должны быть закомментированы):

#define SSD1306_128_64

Кроме того, нужно чтобы был указан правильный I2C адрес дисплея. Нам нужен 0x3C (не 0x3D!), поэтому желательно, чтобы в в том же файле была строка:

#define SSD1306_I2C_ADDRESS   0x3C

Инструкция по работе с библиотекой Adafruit-GFX есть здесь:

https://drive.google.com/file/d/0B743SQ4GE4ggS2ZkdzR4NV9lTjQ/view?usp=sharing.

А здесь есть русский перевод:

http://robocontroller.ru/news/opisanie_biblioteki_adafruit_gfx_dlja_arduino/2015-04-26-56.

Пример программы с использованием библиотек Adafruit:

```
// Подключаем нужные библиотеки.
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
// Создаём объект дисплей.
Adafruit_SSD1306 display;

void setup(){
// Инициализируем дисплей, адрес 0x3C указывать не обязательно, если
// он правильно прописан в библиотеке.
  display.begin(SSD1306_SWITCHCAPVCC,  0x3C);
// Эта строка нужна чтобы правильно выводились русские буквы.
  display.cp437(true);
}

void loop() {
  display.clearDisplay(); // Очистка экрана.
// Выбор цвета: 1 светится, 0 не светится. Можно через запятую
// указать цвет фона.
  display.setTextColor(1);
  display.setCursor(0,0); // Устанавливаем курсор в позицию (0,0).
// Устанавливаем размер шрифта. 1 это 5x7 пикселей, 2, 3 и т.д.
// показывают во сколько раз будет увеличена буква.
  display.setTextSize(1);
//Выводим русский текст через функцию utf8rus.
  display.println(utf8rus("Размер шрифта 1"));
  display.setTextSize(2); // Размер 2
  display.println(utf8rus("Размер 2")); // Выводим текст.
  display.setTextSize(1); // Размер 1
  display.println("English text"); // Выводим английский текст.
  display.display();// Выводим на экран то что приготовили.
  delay(5000); // Ждём 5 секунд.
  display.clearDisplay();
// Вывод прямоугольника со скруглёнными углами: координаты левого
// верхнего угла, ширина, высота, радиус скругления, цвет.
  display.drawRoundRect(5,5,100,50,8,1);
  display.display();
```

```
  delay(5000);
 }

// Функция для перекодировки русских букв
#define maxString 21
char target[maxString + 1] = "";

char *utf8rus(char *source)
{
  int i,j,k;
  unsigned char n;
  char m[2] = { '0', '\0' };
  strcpy(target, ""); k = strlen(source); i = j = 0;
  while (i < k) {
    n = source[i]; i++;
    if (n >= 0xC0) {
      switch (n) {
        case 0xD0: {
          n = source[i]; i++;
          if (n == 0x81) { n = 0xA8; break; }
          if (n >= 0x90 && n <= 0xBF) n = n + 0x30;
          break;
        }
        case 0xD1: {
          n = source[i]; i++;
          if (n == 0x91) { n = 0xB8; break; }
          if (n >= 0x80 && n <= 0x8F) n = n + 0x70;
          break;
        }
      }
    }
    m[0] = n; strcat(target, m);
    j++; if (j >= maxString) break;
  }
  return target;
}
```

Обе библиотеки позволяют выводить картинки, закодировав их в текстовом виде в виде массива с кодами пикселей и поместив в виде переменной (константы) в скетч или в отдельный файл в той же папке, что и скетч. Примеры присутствуют в библиотеках. Для перевода изображения (монохромный bmp или png) в массив есть конвертеры — отдельные программы или онлайн. К сожалению, дисплей не всегда показывает картинки правильно. Для библиотеки OLED_I2C подошёл онлайн-конвертер на сайте создателя этой библиотеки:

http://www.rinkydinkelectronics.com/t_imageconverter_mono.php.

Для библиотеки Adafruit подходят две программы: Img2Code на Java (https://github.com/ehubin/Adafruit-GFX-Library/tree/master/Img2Code) и lcd-image-converter (https://sourceforge.net/projects/lcd-image-converter/). Во второй программе нужно загружать файл bmp.

Пример показа картинки с библиотекой OLED_I2C:

```
#include <OLED_I2C.h>
#include <avr/pgmspace.h> // Эту строчку нужно не забыть!
// Ниже идёт описание константы, в которой закодирована картинка.
const uint8_t test[] PROGMEM={
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0xFE, 0xFE, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80,
0xC0, 0xE0, 0xF0, 0xF0, 0xF8, 0xF8, 0xFC, 0x7C, 0x7E, 0x3E, 0x3E, 0x3E, 0x3F, 0x1F, 0x1F, 0x1F,
0xFF, 0xFF, 0xFF, 0x1F, 0x1F, 0x1F, 0x3E, 0x3E, 0x3E, 0x7E, 0x7C, 0xFC, 0xF8, 0xF8, 0xF0, 0xF0,
0xE0, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0xC0,
0xF8, 0xFE, 0xFF, 0xFF, 0xBF, 0x8F, 0x87, 0x83, 0x81, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80, 0x80, 0x80, 0x80, 0xFF, 0xFF, 0xFF, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80, 0x81, 0x83, 0x87, 0x8F, 0xBF, 0xFF, 0xFF, 0xFE, 0xF8, 0xE0, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x03, 0x1F, 0x7F, 0xFF, 0xFF, 0xFF, 0xF3, 0xC3, 0x83, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0xFF, 0xFF, 0xFF, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x83, 0xC3, 0xF3, 0xFF, 0xFF, 0xFF, 0x7F, 0x1F, 0x03,
0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x07, 0x0F, 0x1F, 0x3F, 0x3F,
0x7E, 0xFC, 0xFC, 0xF8, 0xF8, 0xF0, 0xF0, 0xF0, 0xE0, 0xE0, 0xE0, 0xE0, 0xFF, 0xFF, 0xFF, 0xE0,
0xE0, 0xE0, 0xF0, 0xF0, 0xF0, 0xF8, 0xF8, 0xFC, 0xFC, 0x7E, 0x3F, 0x3F, 0x1F, 0x0F, 0x07, 0x03,
0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0x03, 0x03, 0x03, 0x03,
0x7F, 0x7F, 0x7F, 0x03, 0x03, 0x03, 0x01, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC,
0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC,
0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC,
0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC,
0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC,
0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC,
};
OLED  myOLED(SDA, SCL); // Создаём объект дисплей.
void setup() {
  myOLED.begin(); // Включаем дисплей.
```

```
}
void loop() {
  myOLED.clrScr();
// Функция вывода изображения. В скобках координаты левой верхней
// точки, название константы с изображением, ширина и высота.
  myOLED.drawBitmap(10, 20, test, 100, 50);
  myOLED.update();
  delay(4000);
}
```

Пример показа картинки с библиотекой Adafruit:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display;

void setup(){
  display.begin(SSD1306_SWITCHCAPVCC);
}

void loop() {
static const unsigned char test [] PROGMEM = {
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0x0,0x3f,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xf0,0x0,0x3,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xc0,0x0,0x0,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0x0,0x0,0x0,0x3f,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xfc,0x0,0x0,0x0,0xf,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xf8,0x0,0x71,0xc0,0x7,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xf0,0x7,0xf1,0xf8,0x3,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xe0,0x1f,0xf1,0xfe,0x1,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xc0,0x7f,0xf1,0xff,0x80,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0x80,0xff,0xf1,0xff,0xc0,0x7f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0x81,0xff,0xf1,0xff,0xe0,0x7f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0x3,0xff,0xf1,0xff,0xf0,0x3f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0x7,0xff,0xf1,0xff,0xf8,0x3f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0x7,0xff,0xf1,0xff,0xf8,0x1f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xfe,0xf,0xff,0xf1,0xff,0xfc,0x1f,0xff,0xff,0xff,0xf0,
0xff,0xf0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x1,0xff,0xff,0xf0,
0xff,0xf0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x1,0xff,0xff,0xf0,
0xff,0xf0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x1,0xff,0xff,0xf0,
0xff,0xff,0xfe,0xf,0xff,0xf1,0xff,0xfc,0x1f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xfe,0xf,0xff,0xf1,0xff,0xfc,0x1f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xfe,0x7,0xff,0xf1,0xff,0xf8,0x1f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0x7,0xff,0xf1,0xff,0xf8,0x3f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0x3,0xff,0xf1,0xff,0xf0,0x3f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0x81,0xff,0xf1,0xff,0xe0,0x7f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0x80,0xff,0xf1,0xff,0xc0,0x7f,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xc0,0x7f,0xf1,0xff,0x80,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xe0,0x1f,0xf1,0xfe,0x1,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xf0,0x7,0xf1,0xf8,0x3,0xff,0xff,0xff,0xff,0xf0,
```

```
0xff,0xff,0xff,0xf8,0x0,0xf1,0xc0,0x7,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xfc,0x0,0x0,0x0,0xf,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0x0,0x0,0x0,0x3f,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0x80,0x0,0x0,0x7f,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xf0,0x0,0x3,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0x0,0x3f,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf0
};

  display.clearDisplay();
// Как видно из примера, в этой библиотеке можно указывать также и
// цвет картинки: последние два параметра в скобках это цвет
// изображения и цвет фона. Это удобно, если нужно инвертировать
// картинку (и только её, а не весь экран).
  display.drawBitmap(0, 0,  test, 100, 50, 1, 0);
  display.display();
  delay(5000);
 }
```

Дополнение: Новые библиотеки для дисплеев I2C.

В 2018 году появились две хороши библиотеки от сайта iarduino. Они называются iarduino_OLED и iarduino_OLED_txt и отличаются тем, что в первой можно выводить и текст и графику, а во второй только текст. Они немного уступают по количеству возможностей библиотекам Adafruit, но проще в использовании и изначально поддерживают русский язык без всяких ухищрений.

Официальные страницы библиотек:

iarduino_OLED_txt: http://iarduino.ru/file/341.html

iarduino_OLED: http://iarduino.ru/file/340.html

Подробные инструкции:

https://wiki.iarduino.ru/page/OLED_trema/

Дополнительно:

https://wiki.iarduino.ru/page/i2c_connection/

Вся нужная информация собрана в отдельном документе.

Полезная фишка библиотек iArduino: **перед** их подключением (именно перед!) можно переопределить пины для SDA и SCL через #define. Это не обязательно, но бывает полезно. Например, вы хотите на Leonardo назначить те же порты, что и на UNO (вместо родных леонардовских 3 и 2):

```
#define pin_SW_SDA A4
#define pin_SW_SCL A5
```

или просто решили использовать случайные пины:

```
#define pin_SW_SDA 8
#define pin_SW_SCL 9
```

Вот один небольшой пример программы (тут переопределения нет):

```
#include <iarduino_OLED_txt.h>// Подключаем библиотеку
                              // iarduino_OLED_txt.
iarduino_OLED_txt myOLED(0x3C);// Объявляем объект myOLED, указывая
                               // адрес дисплея на шине I2C: 0x3C
                               //или 0x3D.
extern uint8_t MediumFont[];   // Подключаем шрифт MediumFont.

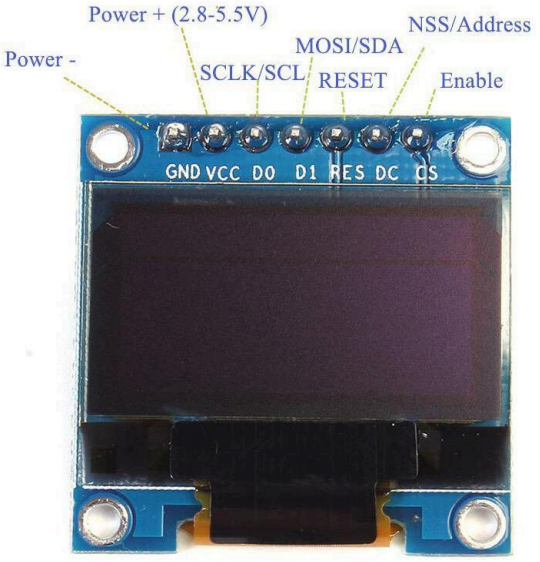
void setup() {
 myOLED.begin(); // Инициируем работу с дисплеем.
 myOLED.setFont(MediumFont);// Указываем шрифт который требуется
                           //использовать для вывода цифр и текста.
}

void loop() {
 myOLED.print("Hello!", 20,3);// выводим текст на 20 пикселей от
                               //левого края экрана в третьей строке
                               //(из семи)
```

```
 myOLED.print(analogRead(A0), 5, 6);// выводим показания с
                                    // аналогового порта на 5
                                    // пикселей слева в 6-й строке.
}
```

Варианты OLED дисплеев 0,96" 128x64 на микросхеме SSD1306 с интерфейсом SPI

Интерфейс SPI использует несколько цифровых портов и подключение по нему несколько сложнее, чем по I2C. Но в каких-то случаях он тоже может пригодиться.



На картинках показаны примеры таких экранов.

Значения контактов.

VCC или VDD это, разумеется, питание (3,3 или 5 В), GND - земля.

Контакты протокола SPI могут обозначаться по-разному, и к тому же часто они одновременно являются контактами I2C (дисплей можно переключать из одного режима в другой), и от этого появляется путаница.

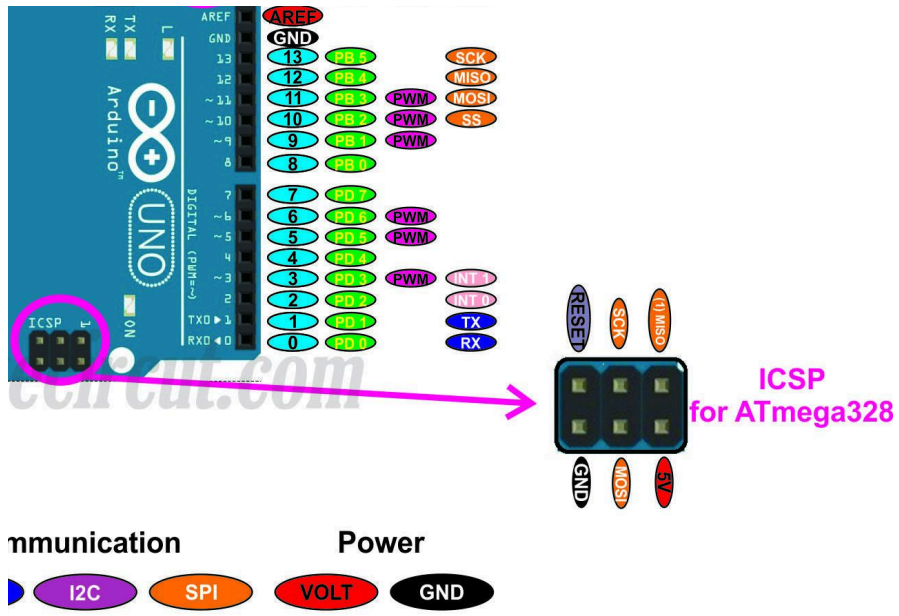
Картина примерно такая (посмотрите таблицу и сравните с картинками):



Название	Официальное обозначение	Варианты
Clock	CLK	SCL, SCK, D0
Data Input	MOSI	SDA, D1, DATA
Chip select signal	CS	SS
Data/Command change	D/C	DC, A0
Reset	RST	RES

Все эти контакты в принципе можно подключать к любым цифровым портам Arduino, указав их номера в программе (ниже будут примеры). Однако, есть

некоторые стандартные порты, которые зависят от модели Arduino. Например, на UNO 13-й порт это Clock, а 11-й - MOSI. На Leonardo эти контакты вообще не связаны с цифровыми портами и выведены на специальный разъём ICSP, что очень хорошо, потому что экономятся цифровые порты. На UNO такой разъём тоже есть, и туда можно подключаться, но там просто повторяются те же цифровые порты 11 и 13, так что всё равно они у вас окажутся заняты. На картинке ниже показан разъём ICSP и его контакты.



Для работы с экраном можно использовать ту же самую библиотеку Adafruit_SSD1306 в паре с библиотекой Adafruit-GFX. О ней подробно написано выше. Это удобно - использовать одну библиотеку для обоих типов дисплеев: I2C и SPI. отличия будут только в начале программы.

Пример программы.

Это тот же пример, что приводился для I2C, но уже в варианте для SPI. Строки, в которых есть отличия, выделены красным:

```
// Подключаем нужные библиотеки, в том числе SPI.
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
// Создаём объект дисплей, настраивая подключение SPI. Можно указать
// все порты:
/*  #define OLED_MOSI  11 //порт для MOSI
  #define OLED_CLK   13 //порт для Clock
  #define OLED_DC    9 //порт для DC
  #define OLED_CS    10 //порт для CS
  #define OLED_RESET 8 //порт для RESET
```

```
  Adafruit_SSD1306 display(OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET,
OLED_CS); */

// А можно указать только три порта, если Clock и MOSI вы
// подключаете к стандартным разъёмам для данного вида Arduino:
#define OLED_DC     9
#define OLED_CS     10
#define OLED_RESET  8
Adafruit_SSD1306 display(OLED_DC, OLED_RESET, OLED_CS);
// Внимание! В принципе можно сэкономить порт, подключив Reset к
// стандартному контакту Reset на Arduino. А в качестве OLED_RESET
// указать какой-то несуществующий порт, например, 55. Но иногда это
// может приводить к глюкам.

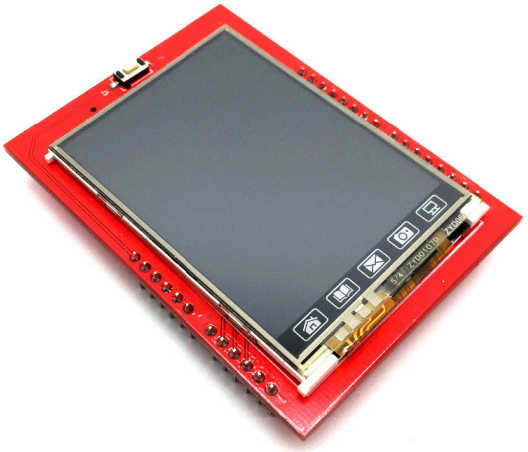
void setup(){
// Инициализируем дисплей.
  display.begin(SSD1306_SWITCHCAPVCC);
// Эта строка нужна чтобы правильно выводились русские буквы.
  display.cp437(true);
}

void loop() {
  display.clearDisplay(); // Очистка экрана.
// Выбор цвета: 1 светится, 0 не светится. Можно через запятую
// указать цвет фона.
  display.setTextColor(1);
  display.setCursor(0,0); // Устанавливаем курсор в позицию (0,0).
// Устанавливаем размер шрифта. 1 это 5x7 пикселей, 2, 3 и т.д.
// показывают во сколько раз будет увеличена буква.
  display.setTextSize(1);
//Выводим русский текст через функцию utf8rus.
  display.println(utf8rus("Размер шрифта 1"));
  display.setTextSize(2); // Размер 2
  display.println(utf8rus("Размер 2")); // Выводим текст.
  display.setTextSize(1); // Размер 1
  display.println("English text"); // Выводим английский текст.
  display.display();// Выводим на экран то что приготовили.
  delay(5000); // Ждём 5 секунд.
  display.clearDisplay();
// Вывод прямоугольника со скруглёнными углами: координаты левого
// верхнего угла, ширина, высота, радиус скругления, цвет.
  display.drawRoundRect(5,5,100,50,8,1);
  display.display();
  delay(5000);
 }
```

```
// Функция для перекодировки русских букв
#define maxString 21
char target[maxString + 1] = "";

char *utf8rus(char *source)
{
  int i,j,k;
  unsigned char n;
  char m[2] = { '0', '\0' };
  strcpy(target, ""); k = strlen(source); i = j = 0;
  while (i < k) {
    n = source[i]; i++;
    if (n >= 0xC0) {
      switch (n) {
        case 0xD0: {
          n = source[i]; i++;
          if (n == 0x81) { n = 0xA8; break; }
          if (n >= 0x90 && n <= 0xBF) n = n + 0x30;
          break;
        }
        case 0xD1: {
          n = source[i]; i++;
          if (n == 0x91) { n = 0xB8; break; }
          if (n >= 0x80 && n <= 0x8F) n = n + 0x70;
          break;
        }
      }
    }
    m[0] = n; strcat(target, m);
    j++; if (j >= maxString) break;
  }
  return target;
}
```

2,4 дюймовый сенсорный экран на микросхеме ILI9341



В продаже есть много разновидностей сенсорных экранов, они могут быть сделаны на разных микросхемах-драйверах. Наш вариант сделан на ILI9341 и имеет разрешение 240x320.

Этот экран имеет резистивный сенсор, что означает возможность воздействовать на него любым предметом. Лучше всего взять пластиковую или деревянную палочку.

Для работы понадобится несколько библиотек:

- Уже упомянутая выше универсальная библиотека Adafruit-GFX. Это значит, и всё что говорилось про её русификацию — будет работать и здесь. Функции рисования тоже.
- Adafruit_TFTLCD для управления экраном: https://github.com/adafruit/TFTLCD-Library.
- Adafruit Touchscreen для управления сенсором: https://github.com/adafruit/Touch-Screen-Library.

В нашем случае понадобятся некоторые настройки, чтобы программа могла правильно работать с экраном.

Во-первых, в файле Adafruit_TFTLCD.cpp нужно проверить, чтобы были активны строки, соответствующие нашему разрешению:

#define TFTWIDTH 240
#define TFTHEIGHT 320

Аналогичные строки для разрешения 320x480 нужно закомментировать.

Во-вторых, в примерах библиотеки TFTLCD предполагается автоматическое определение типа драйвера, но с ILI9341 это не срабатывает, нужно указать вручную. Это будет видно в примере.

Пример программы-рисовалки:

```
// Подключаем три библиотеки.
#include <Adafruit_GFX.h>
#include <Adafruit_TFTLCD.h>
#include <TouchScreen.h>
// Определяем порты (обратить внимание: в примерах из библиотеки они
// могут отличаться!)
#define YP A1
#define XM A2
#define YM 7
#define XP 6
```

```
// Эти числа будут использоваться для пересчёта координат нажатия.
#define TS_MINX 150
#define TS_MINY 120
#define TS_MAXX 920
#define TS_MAXY 940

// Создаём объект-дисплей. Последний параметр это сопротивление
// между слоями, от правильности этого числа зависит определение
// силы нажатия.
TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);
// Назначаем ещё несколько портов.
#define LCD_CS A3
#define LCD_CD A2
#define LCD_WR A1
#define LCD_RD A0
#define LCD_RESET A4
// Даём названия цветам для удобства.
#define   BLACK   0x0000
#define   BLUE    0x001F
#define   RED     0xF800
#define   GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF
// Инициализируем дисплей.
Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
// Задаём размер квадратика с цветом на палитре и диаметр точки,
// которую будет ставить "карандаш".
#define BOXSIZE 40
#define PENRADIUS 3
// Переменные для цветов — старого и текущего.
int oldcolor, currentcolor;

void setup() {
  tft.reset(); // Перезагружаем экран.
// Под комментарием — оригинальная строка из примера, в которой тип
// драйвера определяется автоматически. Ниже — явно прописанное
// указание на нужный нам драйвер.
  //uint16_t identifier = tft.readID();
  uint16_t identifier =0x9341;

tft.begin(identifier); //Запускаем экран.

tft.fillScreen(BLACK); // Заливаем его чёрным.
```

```
//Рисуем на краю экрана палитру из шести или семи квадратов, залитых
// разными цветами.
  tft.fillRect(0, 0, BOXSIZE, BOXSIZE, RED);
  tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, YELLOW);
  tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, GREEN);
  tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, CYAN);
  tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, BLUE);
  tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, MAGENTA);
  // tft.fillRect(BOXSIZE*6, 0, BOXSIZE, BOXSIZE, WHITE);
// Делаем белую рамку вокруг первого (красного) квадрата и
// записываем красный в текущий цвет.
  tft.drawRect(0, 0, BOXSIZE, BOXSIZE, WHITE);
  currentcolor = RED;
 }
// Устанавливаем минимальный и максимальный нажим.
#define MINPRESSURE 10
#define MAXPRESSURE 1000

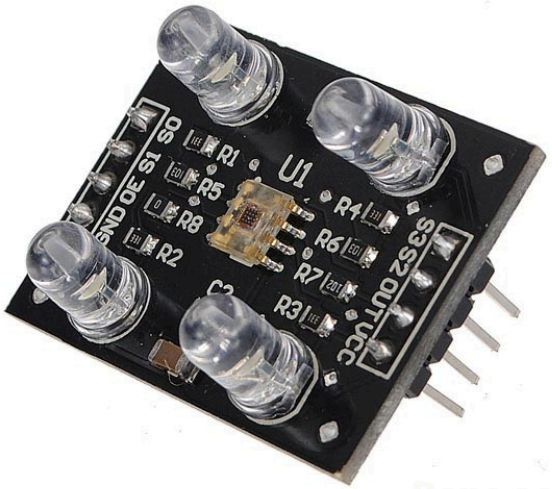
void loop()
{
   TSPoint p = ts.getPoint(); // Получаем координаты точки нажатия.

// Устанавливаем режимы портов.
  pinMode(XP, OUTPUT);
  pinMode(XM, OUTPUT);
  pinMode(YP, OUTPUT);
  pinMode(YM, OUTPUT);
// Проверяем, что давление в норме.
  if (p.z > MINPRESSURE && p.z < MAXPRESSURE) {
// Если нажатие на краю экрана (где "кнопки") — то "очищаем" экран,
рисуя чёрный прямоугольник на зоне рисования.
  if (p.y < (TS_MINY-5)) {
    tft.fillRect(0, BOXSIZE, tft.width(), tft.height()-BOXSIZE,
BLACK);
    }
// Пересчитываем координаты из 0-1023 в реальные пиксели.
    p.x = map(p.x, TS_MINX, TS_MAXX, tft.width(), 0);
    p.y = map(p.y, TS_MINY, TS_MAXY, tft.height(), 0);
// Проверяем, что нажатие было в зоне палитры и обновляем переменную
// предыдущего цвета (текущий становится предыдущим).
    if (p.y < BOXSIZE) {
       oldcolor = currentcolor;
// Определяем в каком квадрате произошло нажатие, устанавливаем
// соответствующий цвет, рисуем белую рамку на нужном квадратике.
       if (p.x < BOXSIZE) {
```

```
          currentcolor = RED;
          tft.drawRect(0, 0, BOXSIZE, BOXSIZE, WHITE);
        } else if (p.x < BOXSIZE*2) {
          currentcolor = YELLOW;
          tft.drawRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, WHITE);
        } else if (p.x < BOXSIZE*3) {
          currentcolor = GREEN;
          tft.drawRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, WHITE);
        } else if (p.x < BOXSIZE*4) {
          currentcolor = CYAN;
          tft.drawRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, WHITE);
        } else if (p.x < BOXSIZE*5) {
          currentcolor = BLUE;
          tft.drawRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, WHITE);
        } else if (p.x < BOXSIZE*6) {
          currentcolor = MAGENTA;
          tft.drawRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, WHITE);
        }
// Перерисовываем квадратик предыдущего цвета, чтобы убрать там
// белую рамку.
   if (oldcolor != currentcolor) {
    if (oldcolor == RED) tft.fillRect(0, 0, BOXSIZE, BOXSIZE, RED);
    if (oldcolor == YELLOW) tft.fillRect(BOXSIZE, 0, BOXSIZE,
BOXSIZE, YELLOW);
    if (oldcolor == GREEN) tft.fillRect(BOXSIZE*2, 0, BOXSIZE,
BOXSIZE, GREEN);
    if (oldcolor == CYAN) tft.fillRect(BOXSIZE*3, 0, BOXSIZE,
BOXSIZE, CYAN);
    if (oldcolor == BLUE) tft.fillRect(BOXSIZE*4, 0, BOXSIZE,
BOXSIZE, BLUE);
    if (oldcolor == MAGENTA) tft.fillRect(BOXSIZE*5, 0, BOXSIZE,
BOXSIZE, MAGENTA);
        }
    } // Это скобочка от "if (p.y < BOXSIZE)".
// Если нажатие за пределами палитры — ставим в этом месте кружок
// ("точку") заданного размера (то есть, рисуем).
  if (((p.y-PENRADIUS) > BOXSIZE) && ((p.y+PENRADIUS) <
tft.height())) {
      tft.fillCircle(p.x, p.y, PENRADIUS, currentcolor);
    }
  } //Это скобочка от "if (p.z > MINPRESSURE && p.z < MAXPRESSURE)".
}
```

Датчик цвета TCS230

Эти датчики бывают в разных формах, на фотографии представлен прямоугольный вариант с 8 выводами.



В центре расположен световой сенсор (блок фотодиодов), состоящий из отдельных маленьких элементов с цветными светофильтрами. Включая разные элементы, можно получить информацию о красном, синем и зелёном компонентах света. По углам стоят 4 белых светодиода, которые подсвечивают объект.

Назначение выводов:

S0 и S1 — выбор условной шкалы измерения, другими словами, насколько большие числа он будет выдавать. Для Ардуино рекомендуется подавать на S0 HIGH, а на S1 LOW.

S2 и S3 — выбор цвета (то есть, с какого фотодиода считывать информацию):

S2	**S3**	**Цвет**
LOW	LOW	Красный
LOW	HIGH	Синий
HIGH	LOW	Белый (без фильтра) — обычно это смысла не имеет.
HIGH	HIGH	Зелёный

OE — включение и отключение датчика, но лучше его не использовать, по умолчанию датчик включен.

GND — "земля".

VCC — питание 5 В.

OUT — порт вывода.

Данные выводятся в виде сигнала, частота которого зависит от силы света. Для считывания можно использовать функцию pulseIn, она даёт длительность одиночного сигнала в микросекундах. Чем больше сила света, тем меньше эта длительность. Для удобства в программе можно переводить эти числа в более удобные значения функцией map, а также вносить поправки для разных цветов, добиваясь, чтобы белый или серый цвета давали равные значения красного, синего и зелёного. Точность определения цвета этим датчиком не высока.

Подробнее:

http://howtomechatronics.com/tutorials/arduino/arduino-color-sensing-tutorial-tcs230-tcs3200-color-sensor/

https://arduino-kit.ru/catalog/id/datchik-tsveta-tcs230

Пример программы для определения цвета.

```
//Задаём условные обозначения для номеров портов (макроопределения).
#define S0 4
#define S1 5
#define S2 6
#define S3 7
#define sensorOut 8
//Готовим переменную для записи значений.
int frequency = 0;

void setup() {
//готовим порты к работе (управляющие на вывод, порт данных на
ввод).
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(sensorOut, INPUT);
// Устанавливаем шкалу измерения
  digitalWrite(S0,HIGH);
  digitalWrite(S1,LOW);
//Включаем последовательный порт для показа данных на экране.
  Serial.begin(9600);
}

void loop() {
// Выбираем красный цвет
  digitalWrite(S2,LOW);
  digitalWrite(S3,LOW);
// Читаем данные с порта, сразу внося подобранную поправку для
// красного (умножение на 1,1) и переводя значения от 30 до 300 в
// "классические" 0-255, попутно "переворачивая" шкалу, чтобы было
// "больше света — больше значение". Результат записываем в
// переменную frequency.
  frequency = map(pulseIn(sensorOut, HIGH)*1.1, 30, 300, 255, 0);
// Выводим результат по красному цвету в последовательный порт.
  Serial.print("R= ");
  Serial.print(frequency);
  Serial.print("  ");
  delay(100);
// Выбираем зелёный цвет.
  digitalWrite(S2,HIGH);
  digitalWrite(S3,HIGH);
// Читаем данные с переводом в удобные значения.
  frequency = map(pulseIn(sensorOut, HIGH), 30, 300, 255, 0);
```

```
// Выводим результат.
  Serial.print("G= ");//printing name
  Serial.print(frequency);//printing RED color frequency
  Serial.print("  ");
  delay(100);
// Выбираем синий
  digitalWrite(S2,LOW);
  digitalWrite(S3,HIGH);
// Читаем с поправкой и переводом.
  frequency = map(pulseIn(sensorOut, HIGH)*1.41, 30, 300, 255, 0);
// Выводим.
  Serial.print("B= ");//printing name
  Serial.print(frequency);//printing RED color frequency
  Serial.println("  ");
  delay(100);
}
```

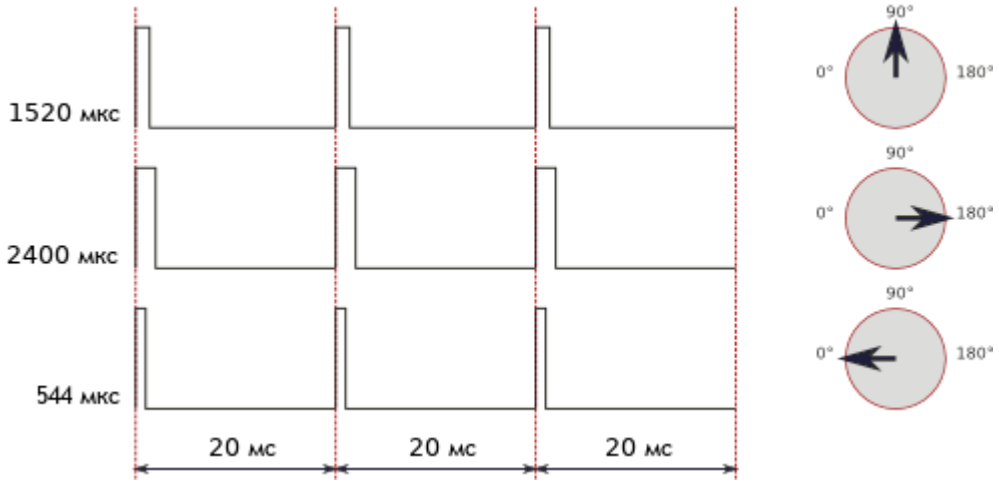
Работа с сервомоторами (сервоприводами)

Сервомотором называется такой мотор, который может поворачиваться на заданный угол (обычный сервомотор) или двигаться с заданной скоростью (сервомотор постоянного вращения). Собственно, это два разных типа моторов, но управляются они по одному принципу: сигналом, похожим на ШИМ. Подробнее про ШИМ читайте тут.

Первый тип моторов чаще всего может поворачиваться на 180 градусов, хотя, встречаются модели на 90 и на 360. Однако, чаще всего, когда указывают что сервомотор вращается на 360 градусов, имеют в виду мотор постоянного вращения (второй тип).

Обычно сервомоторами управляют с помощью специальных библиотек, в том числе стандартной библиотеки Servo. Но желательно понимать принцип управления. Вот объяснение с сайта "Амперка":

"Чтобы указать сервоприводу желаемое положение, по предназначенному для этого проводу необходимо посылать управляющий сигнал. Управляющий сигнал — импульсы постоянной частоты и переменной ширины.



Чаще всего в хобби-сервах импульсы производятся с частотой 50 Гц. Это значит, что импульс испускается и принимается раз в 20 мс. Обычно при этом длительность импульса в 1520 мкс означает, что сервопривод должен занять среднее положение. Увеличение или уменьшение длины импульса заставит сервопривод повернуться по часовой или против часовой стрелки соответственно. При этом существуют верхняя и нижняя границы длительности импульса. В библиотеке Servo для Arduino по умолчанию выставлены следующие значения длин импульса: 544 мкс — для 0° и 2400 мкс — для 180°.

На что ещё стоит обратить внимание, так это на путаницу в терминологии. Часто способ управления сервоприводами называют PWM/ШИМ (Pulse Width Modulation) или PPM (Pulse Position Modulation). Это не так, и

использование этих способов может даже повредить привод. Корректный термин — PDM (Pulse Duration Modulation). В нём крайне важна длина импульсов и не так важна частота их появления. 50 Гц — это норма, но сервопривод будет работать корректно и при 40, и при 60 Гц. Единственное, что нужно при этом иметь в виду — это то, что при сильном уменьшении частоты он может работать рывками и на пониженной мощности, а при сильном завышении частоты (например, 100 Гц) может перегреться и выйти из строя. "

Остаётся только добавить, что мотор **постоянного вращения** будет в ответ на импульсы разной длины вращаться с разной скоростью.



Справа показан самый дешёвый и популярный моторчик SG90, он работает от 5 вольт, вращается на 180 градусов и подходит там, где не нужно большой мощности. Поскольку он очень маломощный, то один такой мотор для экспериментов можно подключить прямо к разъёму питания 5В на Ардуино. Но обычно для любых моторов нужно отдельное питание.

Подключение проводов:

Коричневый или чёрный — земля.

Красный – питание.

Оранжевый – управление (к цифровому пину).

Можно подключать к любому пину, но есть особенность: независимо от того, куда вы подключили сервомотор, он мешает нормальной работе функции analogWrite на 9 и 10 пинах.

Использование библиотеки Servo.

#include <Servo.h> — подключить библиотеку

Servo [название] — создать объект типа серво

Например:

Servo servo1;

Методы объекта типа Servo:

servo1.attach(номер пина) — подключение мотора к пину.

servo1.attach(номер пина, min, max) — то же самое, но у установкой длин импульсов в микросекундах, соответствующих 0 градусов (min) и 180 градусов (max). По умолчанию это 544 и 2400 мкс соответственно.

servo1.write(угол) — установка положения (угла) поворота.

servo1.writeMicroseconds(время в мкс) — установка положения через прямое задание длины импульса в микросекундах.

servo1.read() — читает текущее значение угла, в котором находится сервопривод. Возвращается целое значение от 0 до 180.

servo1.attached(номер пина) — возвращает истину если к данному пину присоединён сервомотор и ложь если нет (то есть, это просто проверка).

servo1.detach(номер пина) — отключение сервомотора от пина.

Пример программы с двумя сервоприводами.

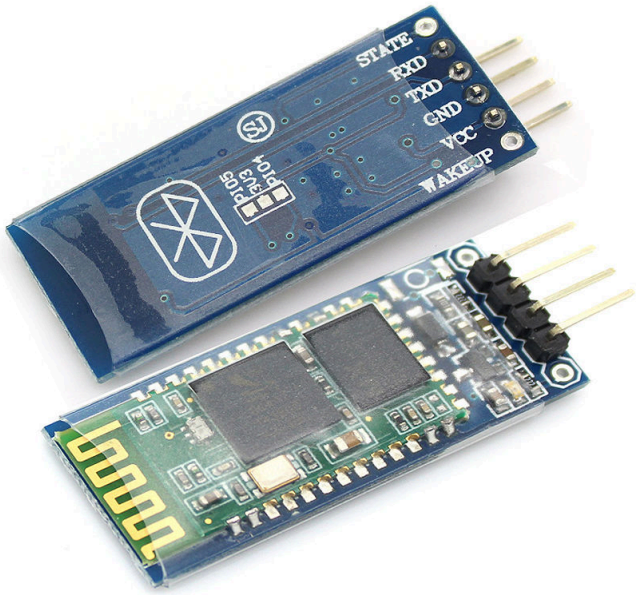
```
// подключаем библиотеку для работы с сервоприводами
#include <Servo.h>
// создаём объекты для управления сервоприводами
Servo myservo1;
Servo myservo2;

void setup()
{
  // подключаем сервоприводы к 11 и 12 пину
  myservo1.attach(11);
  myservo2.attach(12);
}

void loop()
{
// устанавливаем сервопривод в серединное положение (90 градусов)
  myservo1.write(90);
  myservo2.write(90);
  delay(500);
// устанавливаем сервопривод в крайнее левое положение (0 градусов)
  myservo1.write(0);
  myservo2.write(0);
  delay(500);
// устанавливаем в крайнее правое положение (180 градусов)
  myservo1.write(180);
  myservo2.write(180);
  delay(500);
}
```

Использование контроллера Bluetooth

Контроллер Bluetooth это самый простой способ дистанционного управления устройством на Arduino. Подключается он очень просто, управляется с любого телефона на андроиде или iOs.



Самый популярный модуль Bluetooth для Ардуино называется HC-06 и выглядит примерно так как на картинке справа.

Модуль работает через обычный последовательный порт (Serial). Подробно о работе с последовательным портом можно прочитать здесь.

На модуле 4 вывода: земля, питание и два вывода для подключения к последовательному порту (TX и RX).

На большинстве плат Arduino стандартные TX и RX находятся на цифровых пинах 1 и 0 соответственно. На Mega есть дополнительные два порта. Используя библиотеку SoftwareSerial вы можете назначить на роль TX и RX любые пины.

Итак, подключаем Bluetooth модуль к микроконтроллеру Arduino (используем для примера стандартные TX и RX):

Arduino	Bluetooth
Pin 1 (TX)	RX
Pin 0 (RX)	TX
GND	GND
5V	VCC

Будьте внимательны: подключать подключать нужно TX к RX и RX к TX. TX это передача, а RX — приём, так что логично, что передающий контакт модуля нужно подключать к принимающему контакту Arduino и наоборот.

Во время загрузки скетча в UNO или Nano необходимо чтобы Bluetooth модуль был отключен от микроконтроллера arduino. В противном случае скетч не запишется, потому что связь с Bluetooth модулем происходит по одному и тому же порту RX и TX, что и связь с компьютером через USB. На Leonardo и Micro такой проблемы нет (а также её, конечно, нет, если вы используете SoftwareSerial на других пинах).

Соединение по Bluetooth как бы заменяет провод, а компьютер мы заменяем телефоном. Для управления мы просто передаём с телефона на Arduino любые символы (обычно цифры) и прописываем в программе — что делать, если пришёл тот или этот символ.

Пример скетча для включения и выключения светодиода по командам через Bluetooth.

```
byte val;// создаём переменную для записи команды, используем тип
         // byte так как данные передаются в виде байтов (символов)
void setup(){
  Serial.begin(9600);//готовим порт
  pinMode(13, OUTPUT);// включаем 13-й пин в режим вывода
                      // (это порт со встроенным светодиодом)
}
void loop()
{
  if (Serial.available())//если в порт поступили данные...
  {
    val = Serial.read(); // записываем их в переменную val.

    // Если пришёл символ "1" включаем светодиод
    if (val == '1')
    {
      digitalWrite(13, HIGH);
    }

    // Если пришёл символ "0" выключаем светодиод
    if ( val == '0')
    {
      digitalWrite(13, LOW);
    }
  }
}
```

Так можно управлять чем угодно. Например, для управления движущимся роботом можно настроить разные действия (прямо, стоп, повороты...) при поступлении различных команд (разных букв или цифр).

Можно также передавать данные и в обратном направлении, то есть, получать на телефон показания с датчиков или какие-то сообщения. Для реализации этого просто используйте функции работы с последовательным портом.

Как подключить Bluetooth модуль HC-06 к телефону:

1. Включаем Bluetooth на телефоне и ищем новые устройства
2. Находим в списке расстройств "HC-06" и подключаемся к нему.
3. Телефон спросит пин-код. необходимо ввести "1234" или "0000"
4. Ура. Устройство подключено.

Теперь нужно скачать программу для управления на ваш телефон. Их много, можно выбирать по вкусу. Например ArduinoRC (Arduino Bluetooth Controller). Для нашего примера со светодиодом нужен просто терминал (окошко, где можно писать знаки для передачи на Arduino или читать то, что пришло оттуда). После того как мы установили программу, запускаем её выбираем наш bluetooth модуль HC-06, ждём подключения и запускаем терминал. Пишем в терминале цифру "1" и отправляем. Светодиод, который находится на плате arduino, должен загореться. Теперь отправим через терминал цифру "0" и светодиод должен погаснуть.

Кроме терминала в таких программах бывает экранный джойстик: кнопки, которым можно присвоить текстовые значения для отправки на Arduino. В некоторых программах возможно также управление наклоном телефона.

Общая информация о шаговых моторах

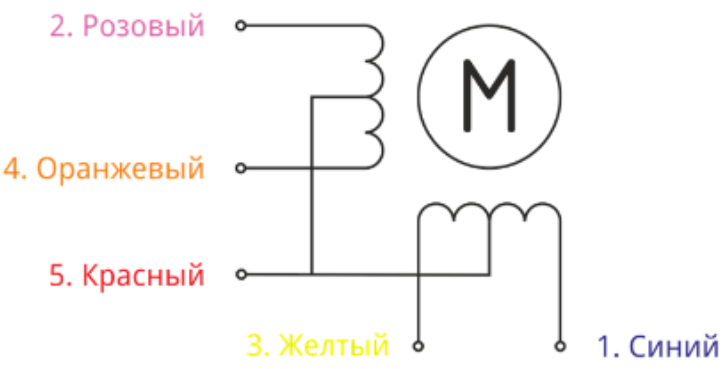
Шаговый мотор поворачивает вал понемногу, небольшими "шагами". Каждый шаг это 90 градусов или 45 или ещё меньше (а с учётом замедляющего редуктора шаги превращаются в доли градуса). Этими шагами можно точно управлять: количеством шагов, скоростью, направлением. По точности шаговые моторы намного превосходят серво. Поэтому их используют там, где нужны точно отмеренные движения (принтеры, автоматические станки, механические "руки" и т.п.).



К недостаткам шаговых моторов относится их высокая цена, требовательность к мощности питания и довольно сложное управление.

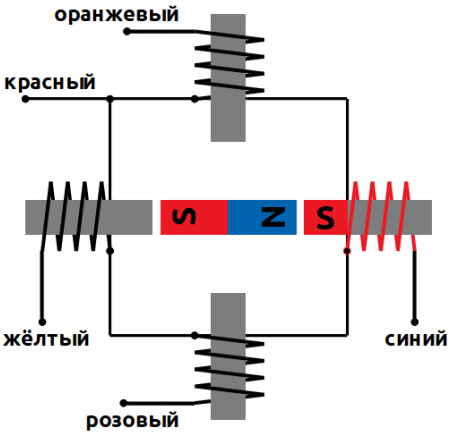
Принцип работы такого мотора заключается в последовательном включении нескольких обмоток (это электромагниты), которые притягивают магнитный ротор сложной формы так, чтобы он поворачивался вслед за переключением обмоток.

Справа вверху показан самый дешёвый шаговый мотор, который обычно используют для пробы начинающие "ардуинщики" — это 28BYJ-48 с небольшой платой-драйвером. Он относится к типу **униполярных** шаговых моторов, имеет пять выводов. Справа его схема. Обратите внимание: красный провод является общим для четырёх разных обмоток.

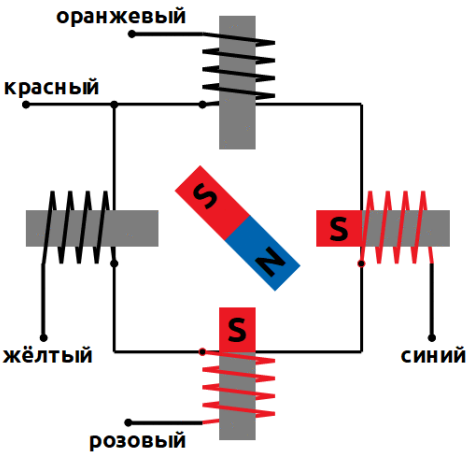


Посмотрим анимацию разных схем работы такого **униполярного мотора**. Тут видно как вращающийся магнит (ротор) притягивается по очереди к разным обмоткам.

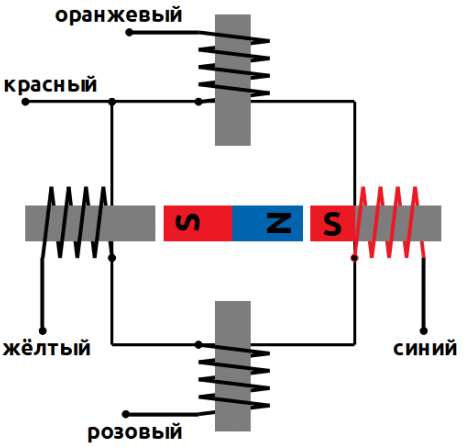
В первом случае (справа) включается одновременно только одна обмотка, это *полношаговое управление **одной** обмоткой (волновой режим)*.



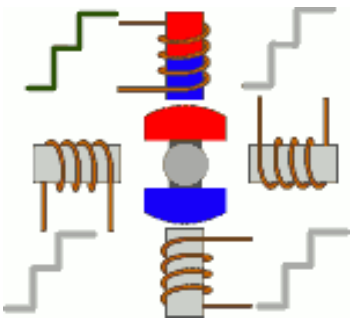
Теперь *полношаговое управление **двумя** обмотками*. Так мотор развивает больший крутящий момент. Такой режим ещё называют четырёхшаговым.



Для более точных движений можно использовать *полушаг*: когда работают то одна то две обмотки. Такой режим ещё называют восьмишаговым.

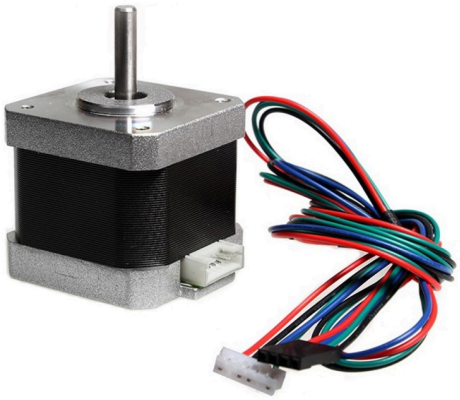


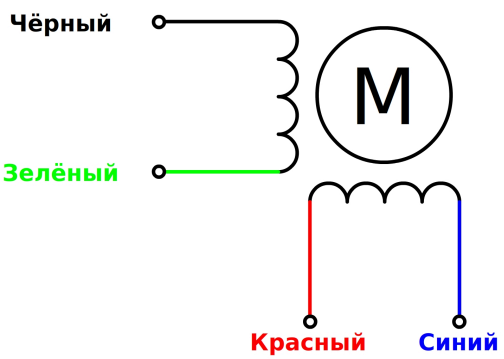
Микрошаг. Сила тока в обмотках меняется ступенчато. То есть, они могут работать вполсилы, в четверть силы и т.д.. Количество микрошагов на оборот может доходить до сотен. Этот вариант широко используется сейчас.



Обратите внимание: во всех этих примерах обмотки-электромагниты всегда создают в сторону ротора только магнитный полюс одного типа (в первых трёх примерах красный — "южный", в последнем синий — "северный"), поэтому мотор и называется униполярным (однополярным).

Теперь посмотрим на другой тип шагового мотора — **биполярный**.
У него две независимые обмотки и четыре вывода.

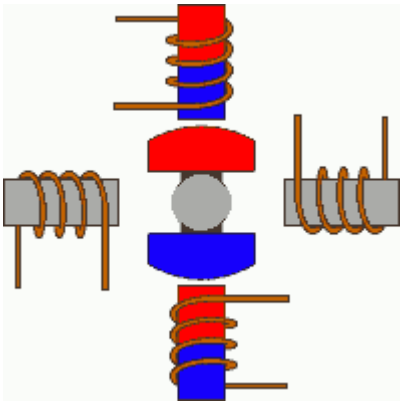


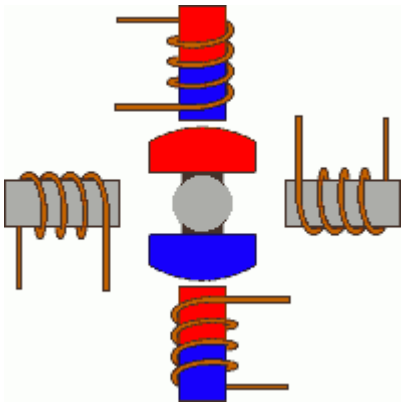


Обмотки не только включаются поочерёдно, но и **меняют полярность**. То есть, на их контактах постоянно меняется "плюс" с "минусом", а это приводит к изменению полярности магнитного поля. Отсюда и название "**биполярный**".

Вот примеры режимов такого мотора. Две обмотки здесь превращены в четыре: просто каждая обмотка разделена на две катушки — напротив друг друга. Видно, что противоположные катушки работают или не работают всегда одновременно.

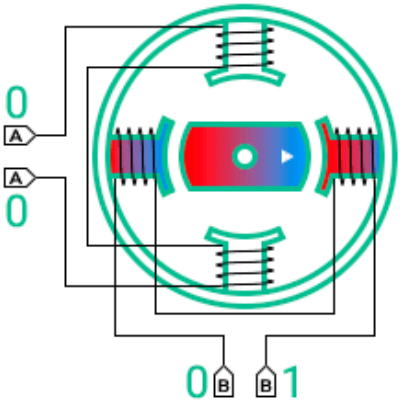
Справа полношаговый режим. Обратите внимание: обмотки "умеют" включать в сторону ротора то синий то красный полюс. Поэтому мотор и называется **биполярным**.



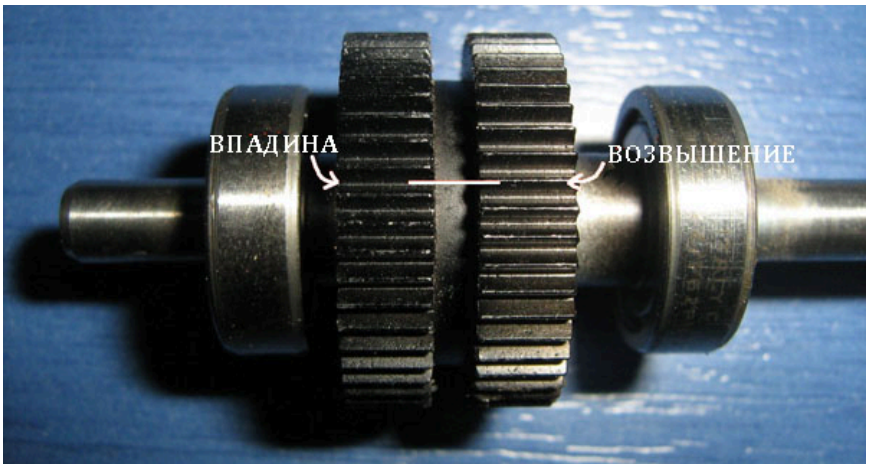


Слева полушаговый режим. Видно, что в отличие от униполярного мотора тут работают все катушки сразу

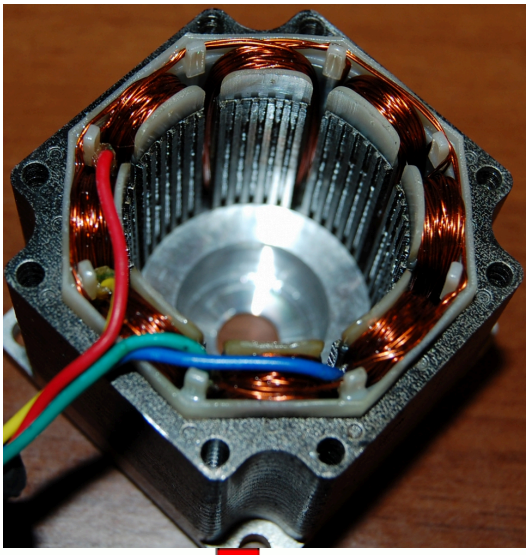
Справа микрошаг:



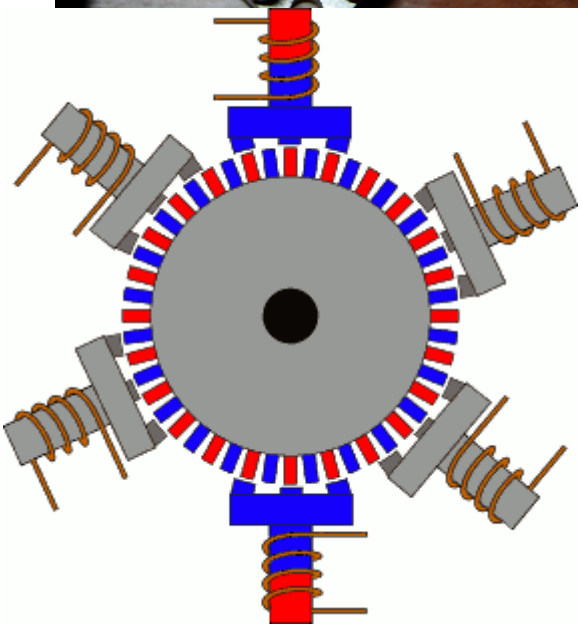
Надо понимать, что это упрощённые схемы, на самом деле ротор, конечно, имеет более хитрую форму, у него может быть много чередующихся магнитных полюсов. И обмоток может быть больше, (то есть, их всё равно как бы две или четыре, но каждая их них может быть разделена на несколько катушек, которые включаются одновременно). Ниже пример конструкции, где на статоре (неподвижной части мотора) 8 обмоток и ротор с множеством полюсов.



Каждый зубчик одной "шестерёнки" — северный полюс, а другой "шестерёнки" — южный полюс, причём, идут они со сдвигом. Если смотреть с торца — получится чередование полюсов Ниже есть анимация, где это видно).



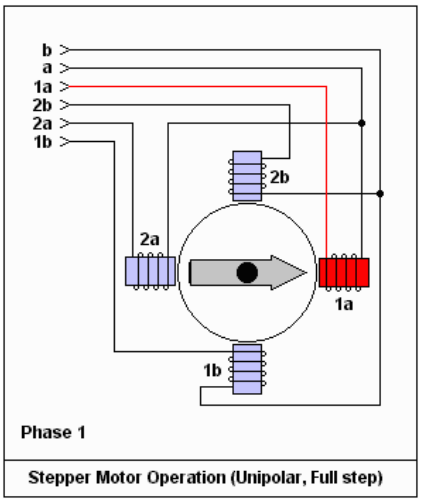
А обмотки тоже непросты, их магнитный полюс не плоский: из их сердечников выступают рёбра. Таким образом магнитные рёбра ротора притягиваются к рёбрам статора. И расположены они с такими отступами, чтобы заставлять рёбра ротора каждый шаг - при переключении катушек — чуть-чуть "подтягиваться" к следующему ребру очередной работающей обмотки.



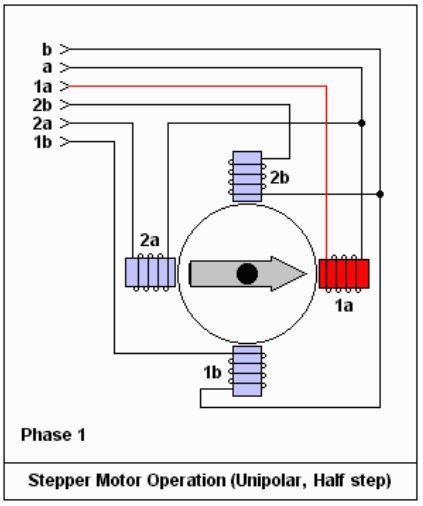
Вот как примерно это работает (тут, правда, показано 6 катушек). Синие и красные рёбра на роторе относятся к разным шестерёнкам. Они расположены друг за другом. Но когда мы смотрим с торца — они сливаются в одно колесо.

Вот прекрасное видео, где всё наглядно показано.

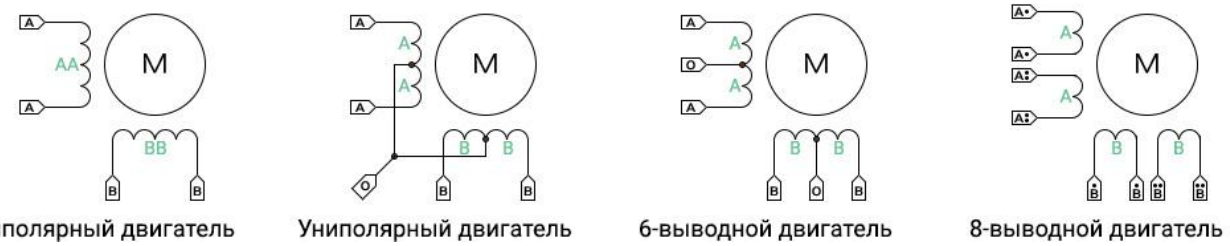
Для полноты обзора ещё несколько примеров.



Здесь показаны полношаговый и полушаговый режимы для 6-контактного униполярного шагового мотора. В 5-контактном был один общий (красный) провод на все четыре обмотки. А в этом случае он как бы разделён на два: a и b. Каждый из них является общим на две обмотки.

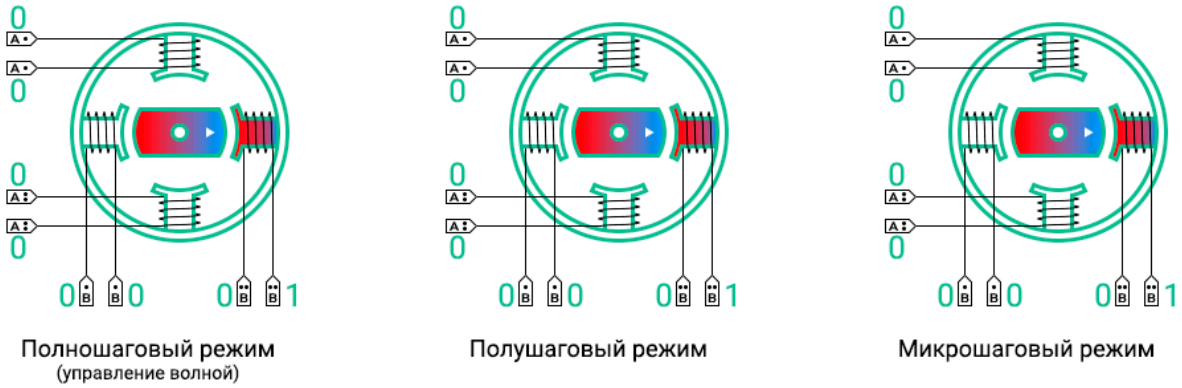


Вот все используемые типы шаговых двигателей по схеме подключения.

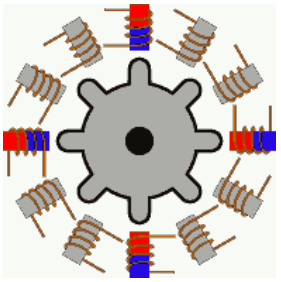


8-контактный мотор можно подключать и в униполярном режиме и в биполярном. Впрочем, любители переделывают и 5 и 6-контактные униполярные двигатели в биполярные. В 5-контактном нужно только перерезать внутри провод, соединяющий две обмотки, а в 6-контактном просто не использовать два общих контакта. Подробнее про схемы подключения можно прочитать здесь.

8-контактный мотор можно также использовать в режиме пониженного энергопотребления, когда используются только половина обмоток.



И, наконец, есть шаговые двигатели с ротором, который не является магнитом (но сделан из магнитного металла). Его зубчатый ротор притягивается зубцами к выступающим полюсам обмоток.



Подключение шаговых моторов к драйверам

Шаговый мотор не имеет специального управляющего провода (как сервомотор). У него только выводы обмоток, которые нужно переключать в нужном порядке с большой скоростью. Причём, они обычно потребляют довольно большой ток, поэтому напрямую к Arduino их подключать нельзя.

Для подключения используются специальные платы с микросхемами-драйверами. Драйверы бывают двух типов по способу управления:

1. С четырьмя контактами (на микросхемах типа Н-мост) — для прямого управления четырьмя обмотками. В этом случае программа должна сама включать обмотки в нужном порядке.

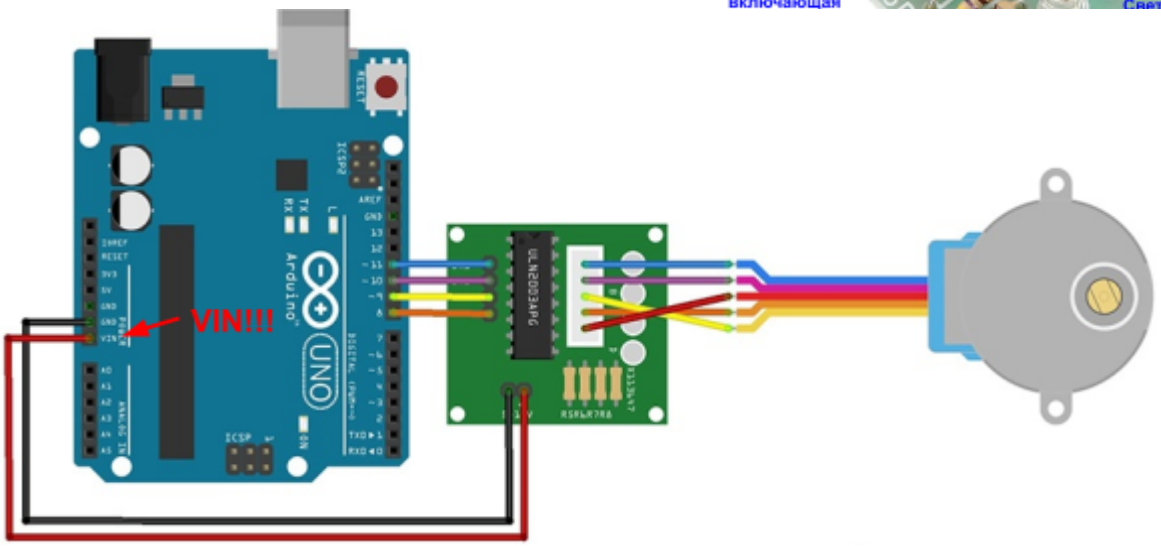
2. С двумя контактами Step и Dir — "шаг" и "направление". На первый нужно просто подавать HIGH, чтобы мотор сделал шаг, а на второй HIGH или LOW, чтобы установить направление вращения. Бывают также отдельные выводы для управления размером шага (полный, половинный, ¼, ⅛ и т.д.).

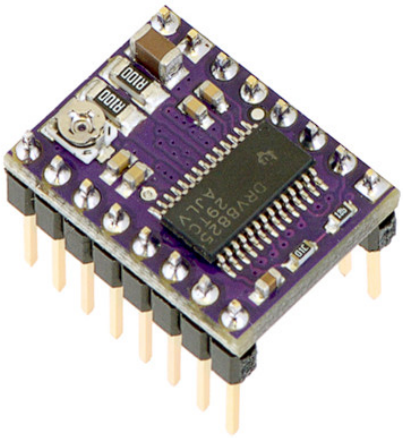
Конечно, использовать драйвер второго типа проще, и он не так сильно нагружает процессор Arduino. Хотя, в любом случае для управления мотором обычно используются специальные библиотеки. Посмотрим примеры подключения.

Подключение 28BYJ-48.

Вот часто используемая плата на микросхеме **ULN2003** для управления дешёвым мотором 28BYJ-48. Её можно питать (5-12 В) от контакта VIN Arduino (а от контакта 5V нежелательно). А лучше вообще использовать отдельный источник питания.

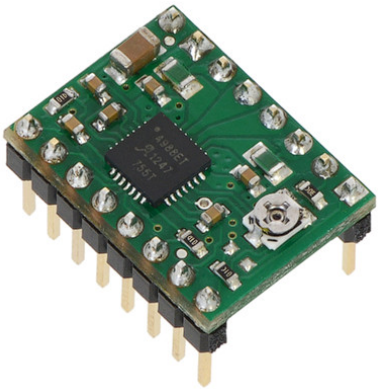




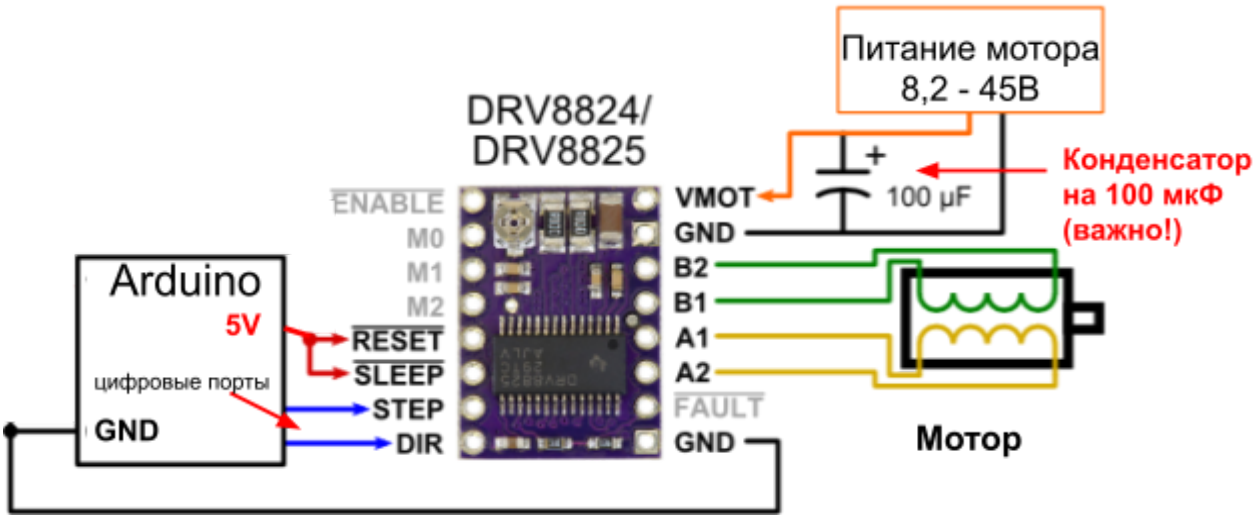


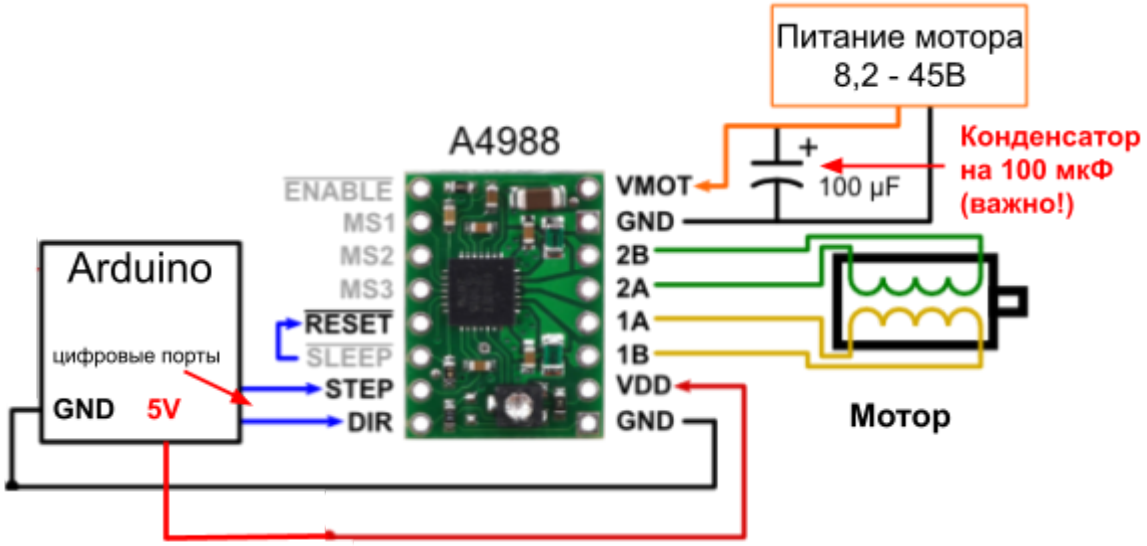
Подключение биполярных моторов к драйверам с выводами Step и Dir.

Драйверы для шагового мотора на микросхемах **A4988**, **DRV8825** и некоторых других выпускаются в виде модулей стандартного размера (смотри фотографии слева). Изначально это дизайн американской фирмы Pololu, но примерно такие же делаются китайцами и продаются гораздо дешевле. Они часто используются в 3d принтерах и в других случаях. Их можно использовать один вместо другого.

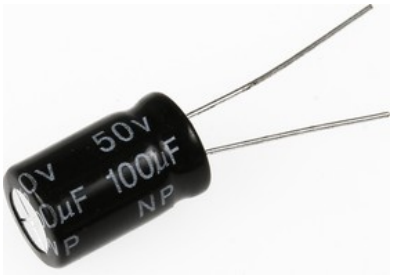


Эти модули имеют выводы Step (шаг) и Dir (направление), а также три контакта для установки шага: полный, ½, ¼, ⅛, 1/16, а на DRV8825 также 1/32. Ниже показаны схемы подключения этих модулей для полного шага. **Внимание! Не подключайте мотор, пока не выполните настройку тока, описанную через одну страницу и вообще не дочитаете главу до конца!!!**





Пояснения к схемам.



1. Не забыть подключить конденсатор! Он должен иметь ёмкость 100 микрофарад и быть рассчитан на напряжение 50 В. Это довольно большой конденсатор. Для электролитического конденсатора важна полярность, его "плюсовой" контакт обычно удлинён.

2. На модулях есть три "загадочных" контакта, смысл которых (если не вдаваться в подробности) — просто отключать моторы. Это ENABLE, RESET и SLEEP. Внутри платы DRV8825 они подтянуты на "землю". Чтобы мотор крутился нужно на ENABLE держать LOW (поэтому он на схеме не подключен никуда), а на двух других держать HIGH (поэтому они подключены к 5В). На A4988 SLEEP сразу подтянут на 5В, а RESET никуда не подтянут, поэтому на схеме они соединены.

На DRV8825 есть ещё FAULT, это выход, который сигнализирует о проблемах, можно на него не обращать внимания.

3. Если вы не вникли в пункт 2 — не беда. Просто подключайте как нарисовано на схемах. Но если всё поняли, то можно, например, SLEEP подключить к цифровому порту и, подавая LOW переводить драйвер в спящий режим, когда мотор должен остановиться. Это уменьшит нагрев.

4. DRV8825 можно подключать как A4988, потому что FAULT связан с SLEEP через резистор и получится нужная подтяжка SLEEP к 5В. Это сделано специально, чтобы можно было в 3d принтере или другом готовом устройстве без проблем заменить A4988 на более новый DRV8825.

5. Управлять размером шага можно через три специальных контакта. На DRV8825 они называются M0, M1, M2, а на A4988 — MS1, MS2, MS3. В модуле они подтянуты к земле, то есть на них изначально LOW (полный шаг).

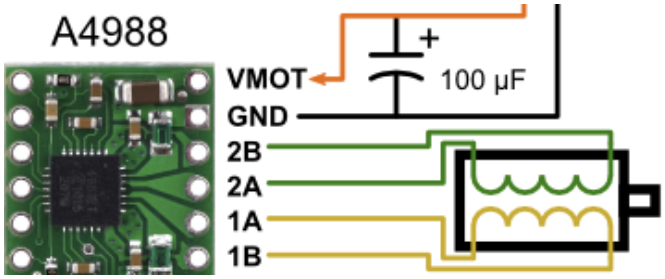
M0/MS1	M1/MS2	M2/MS3	DRV8825	A4988
Low	Low	Low	Полный шаг	Полный шаг
High	Low	Low	Полушаг	Полушаг
Low	High	Low	1/4 step	1/4 step
High	High	Low	1/8 step	1/8 step
Low	Low	High	1/16 step	—
High	Low	High	1/32 step	—
Low	High	High	1/32 step	—
High	High	High	1/32 step	1/16 step

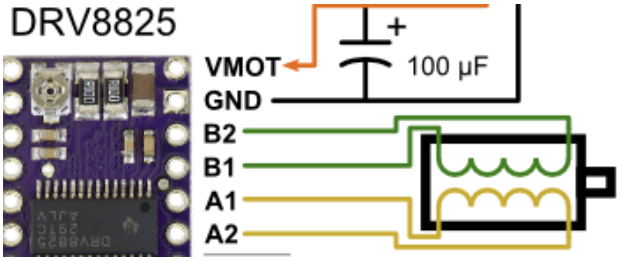
Подключение мотора.

Сначала предупреждение: **никогда не отключайте и не подключайте мотор к драйверу, если на драйвер подано напряжение!**

Биполярный мотор имеет 4 вывода. Это провода разных цветов. Два из них связаны с одной обмоткой, два со второй.

Их нужно подключать к драйверу так, как выше было показано на схемах:





Как можно заметить: обозначение контактов на разных драйверах разное.

На картинке слева:

первая обмотка называется "1", а её выводы — "1A" и "1B",

вторая обмотка — "2" и выводы "2A" и "2B".

На картинке справа:

первая обмотка называется "A" и её выводы "A1" и "A2",

вторая обмотка — "B" и выводы её "B1" и "B2".

На китайских копиях буквы и цифры могут быть переставлены, например:

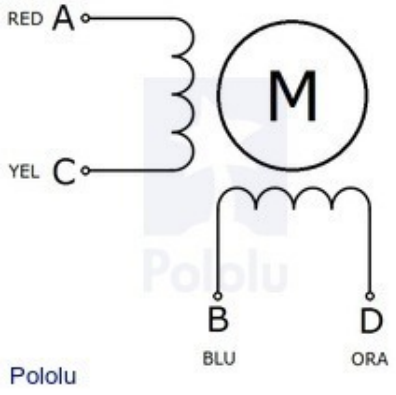
первая "1A" и "2A",

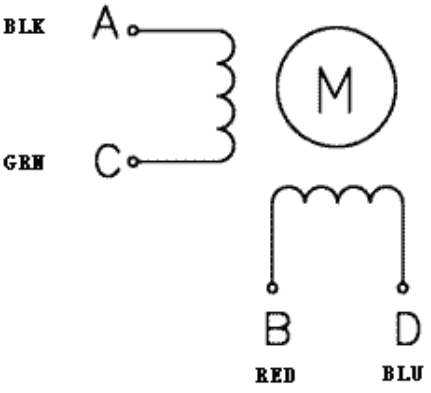
вторая "1B" и "2B".

Как бы они не обозначались — суть не меняется: первую обмотку нужно подключить к нижней паре контактов, вторую — к верхней.

Полярность подключения не критична, но влияет на направление вращения. Если мотор крутится не в ту сторону, что вам нужно — можно подключение **одной** из обмоток поменять местами. Если поменять местами верхнюю и нижнюю обмотки то направление также поменяется.

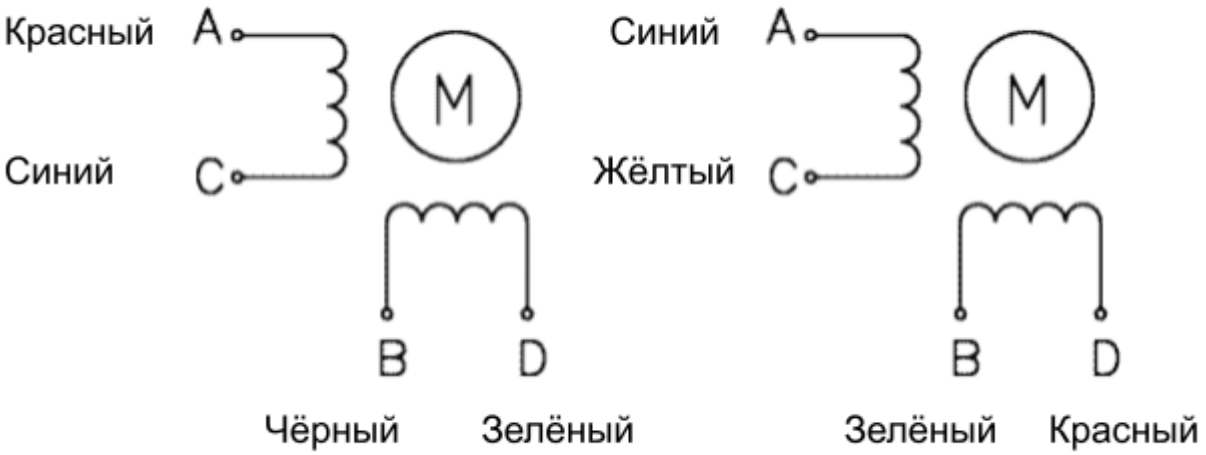
Теперь разберёмся с проводами от мотора. На моторе обычно нет обозначений, только разноцветные провода. Обычно на сайте, где продаётся мотор, есть схема, где показано как цвета соотносятся с обмотками. Вот на картинках примеры с сайта Pololu.



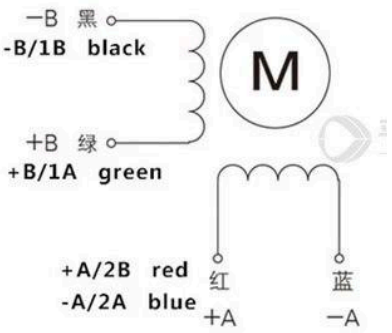


С обозначениями проводов тоже нет единой системы. Выводы могут обозначаться A,C,B,D как

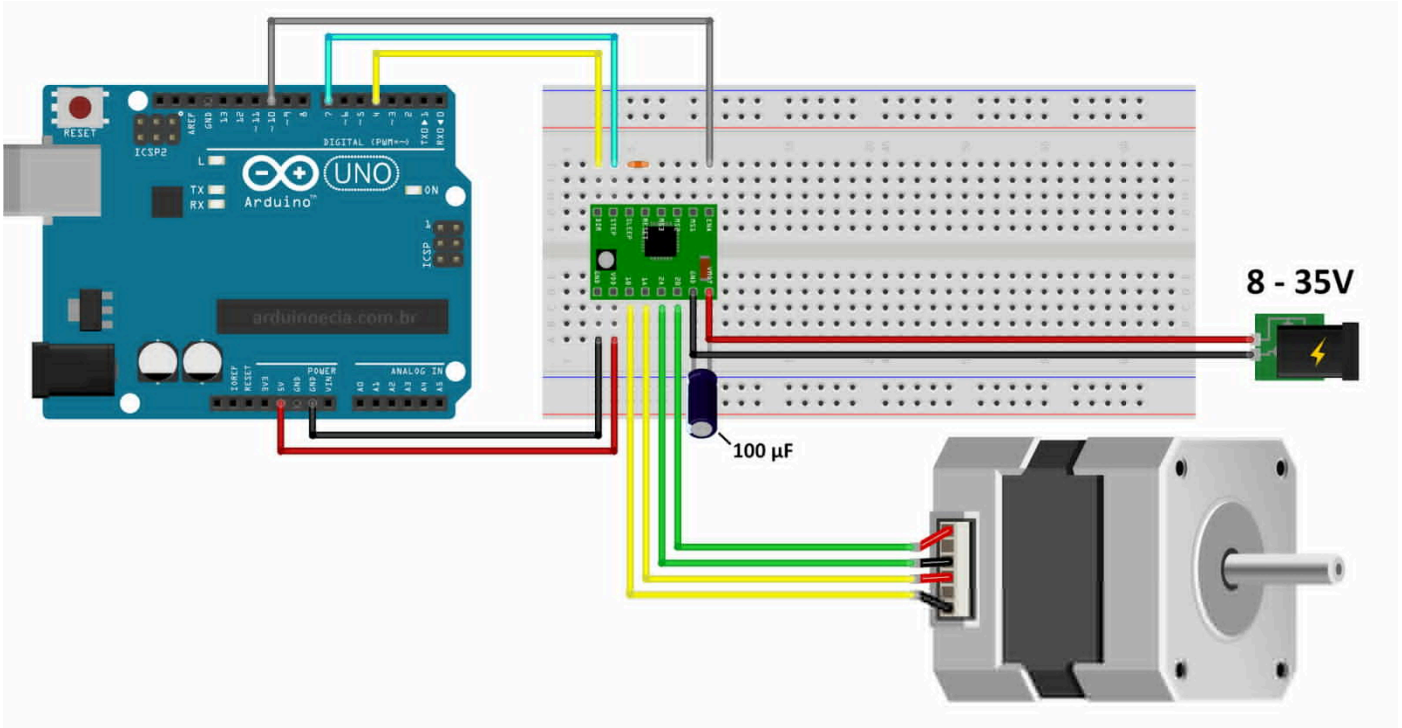
на этих картинках, а могут A+, A-, B+, B-. На AliExpress цветовые схемы чаще всего такие:



Может быть и подробнее: на картинке справа подписаны обозначения контактов для драйвера A4988.



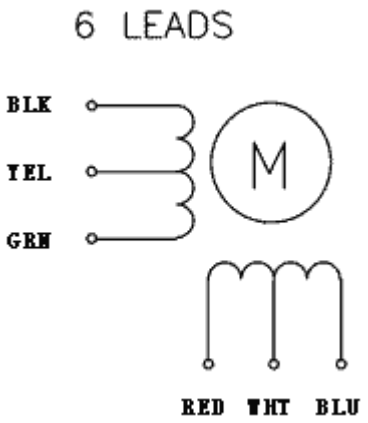
Если на моторе жёсткий разъём с выводами, то обычно они идут в правильном порядке: сначала два вывода одной обмотки, потом два вывода другой. Так что можно подключать провода в порядке их выхода из мотора. Но лучше сначала убедиться по сайту продавца, что это правильно. Ниже пример подключения.



Если информации о моторе нет и непонятно где у него что — есть простые способы узнать какие выводы представляют одну обмотку.

1. Измерить сопротивление между разными контактами: Контакты одной обмотки дадут почти нулевое сопротивление.
2. Никуда не подключая мотор, попробуйте замкнуть разные пары его контактов и повернуть мотор рукой. Когда вы замкнёте контакты одной обмотки — мотор будет прокручиваться с некоторым трудом.

Бывают также моторы с 6 выводами, которые можно использовать по желанию как униполярные или биполярные. Если используем его как биполярный — просто не подключаем средние провода (на картинке справа жёлтый (YEL) и белый (WHT)).



Настройка тока.

В характеристиках любого шагового мотора обязательно указывается рекомендуемая сила тока, который должен проходить через обмотки. Например, 1А, 0,6А, 1,7А и т.д.. Если вы превысите это значение — мотор будет сильно греться, а может и совсем перегорит.

Драйвер также имеет ограничения:

DRV8825: долговременно может давать до 1,5А, а кратковременно или с хорошим охлаждением — до 2,2А;

A4988: 1А и 2А соответственно.

Даже при нормальных токах микросхема сильно греется. Осторожно, можно обжечься! Для охлаждения на неё приклеивают маленький радиатор и часто сверху ставят вентилятор.

Очень важно во-первых, не превышать эти границы, во-вторых, настроить драйвер так, чтобы сила тока подходила для вашего мотора (рекомендуется ставить примерно 70% от номинала). Для этого на драйверах есть специальный потенциометр ("крутилка"). "Крутилку" можно крутить тонкой отвёрткой, но очень осторожно! В крайних положениях она упирается, но если приложить силу — она сломается. Всего она крутится меньше чем на один оборот. На драйвере также расположены так называемые токочувствительные резисторы номиналом в доли ома. (Смотри рисунок.) На них написана буква R и сопротивление в миллиомах: R100 это 0,1 Ом, R50 это 0,05 Ом, R200 это 0,2 Ом.



Приступаем к измерению и настройке. Вначале хорошо бы выкрутить потенциометр на минимум, но поскольку направление вращения для китайских модулей непредсказуемо, то лучше поставить в среднее положение. Подключаем всё как нужно, подаём питание, но пока не подключаем мотор. Измеряем напряжение между землёй и корпусом потенциометра. Это у нас Vref. Теперь по формуле считаем ток:

Для A4988: I=Vref:(8*Rs) или Vref==I*(8*Rs)

Для DRV8825: I=Vref:(5*Rs) или Vref==I*(5*Rs)

I это сила тока, Rs — сопротивление токочувствительных резисторов (в омах).

Например, для DRV8825 Rs обычно равно 0,1 Ом. Значит:

I=Vref:0,5 или I=Vref*2 или Vref=I:2

То есть, намерили напряжение 1В значит сила тока 2А.

Нужно крутить потенциометр (по чуть-чуть!) до тех пор, пока не получим силу тока, соответствующую мотору. Например, нам нужно 1,6А. Значит (для DRV8825), нам нужно напряжение:

Vref=1,6/2=0,8 — крутим, пока не получим 0,8В.

После приблизительного выставления нужного значения можно подключить мотор, запустить тестовый скетч, чтобы мотор вращался, и установить силу тока поточнее, лучше меньше номинала, главное чтобы мотор ещё крутился.

Вот два видео, где авторы показывают этот процесс:

https://www.youtube.com/watch?v=rFElvcnMShU

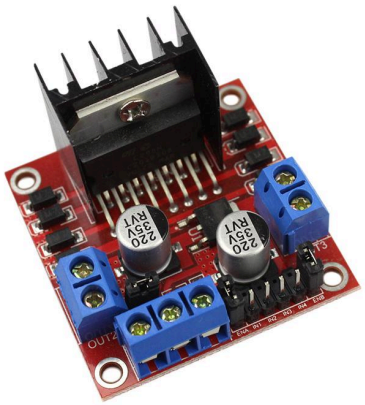
https://www.youtube.com/watch?v=FQepebH2CPY

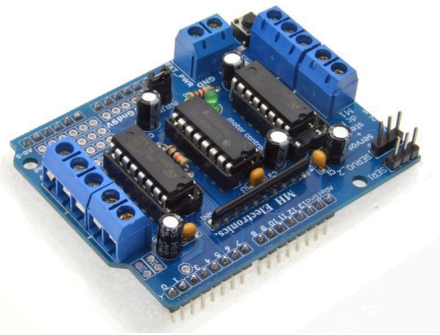
Примечание 1: на плате имеется дырочка-контакт, специально для измерения напряжения (смотри рисунок выше). Но на китайских драйверах она может давать неправильные значения. Поэтому лучше сразу мерить на корпусе потенциометра. К "земле" можно подключиться в любом удобном месте (контакт на самом драйвере, контакт на Arduino, корпус разъёма USB на Arduino...).

Примечание 2: Можно также напрямую подключить амперметр к одной из обмоток мотора, включить его и померить ток непосредственно. При условии, что вы умеете мерить ток, и знаете, что амперметр нужно подключать последовательно с мотором, а иначе будет короткое замыкание.

Использование других драйверов.

Поскольку управление шаговым мотором сводится к попеременному включению обмоток, то их можно подключать и к драйверам, предназначенным для обычных (не шаговых) моторов. Только вместо двух обычных подключается один шаговый. Вот к такому шилду на L293D (справа) можно подключить два шаговых мотора вместо четырёх обычных. Подробнее здесь. А к такой плате на L298N (слева) можно подключить один шаговик. 4 управляющих входа этой платы позволят напрямую управлять обмотками. Нужно только учитывать предельный ток, который способна выдать выбранная плата. Его должно хватать для работы мотора. Причём, регулировать ток эти платы не умеют. Кроме того, каждому мотору нужно своё напряжение, а эти платы подадут на моторы ровно то, что вы подали на их вход питания. Так что прежде нужно внимательно прочитать описания плат-драйверов и моторов и хорошо проверить (а где-то и просчитать по закону Ома) соответствие всех параметров.





Вот здесь описывается пример подключения биполярного мотора к L298N. А здесь — расшифровка схемы.

Существуют также множество других драйверов для шаговых моторов — в виде отдельных модулей или в виде "шилдов", устанавливающихся на Arduino или в виде отдельных "коробочек".

Полезные ссылки.

Объяснения принципов работы шаговых двигателей:
http://ckep.umi.ru/lekcii/lekciya_5_-_kak_rabotayut_shagovye_dvigateli/

https://wiki.iarduino.ru/page/shagovye-dvigateli/

Драйверы на сайте Pololu:

https://b.pololu-files.com/product/2133

https://b.pololu-files.com/product/1182

Перевод статьи с Pololu на русский язык:

http://www.robototehnika.ru/e-store/catalog/203/1252/

Ещё толковая статья про эти драйверы:

https://reprap.org/wiki/A4988_vs_DRV8825_Chinese_Stepper_Driver_Boards/ru

Настройка тока:

https://howtomechatronics.com/tutorials/arduino/how-to-control-stepper-motor-with-a4988-driver-and-arduino/

Неплохие рассказы про мотор 28BYJ-48, но есть неточности насчёт перемычки:

http://blog.rchip.ru/obzor-28byj-48-s-drayverom-uln2003/

http://robotosha.ru/arduino/stepper-motor-28byj-uln2003-arduino.html

Управление шаговыми моторами

Прямое управление (без библиотек).

Драйверы Step/Dir.

Проще всего управлять шаговым мотором через умный драйвер с входами STEP и DIR типа тех же **A4988**, **DRV8825** и им подобных. Мы просто посылаем им команду сделать шаг, а они сами делают остальную работу по правильному переключению обмоток. Вот простой пример скетча, выполняющего поворот на 200 шагов сначала в одну сторону, а потом в другую с помощью такого драйвера. 200 полных шагов это для многих современных шаговых двигателей полный оборот (1 шаг у них 1,8 градуса).

```
void setup() {
pinMode(8,OUTPUT);//порт для DIR
pinMode(9,OUTPUT);//порт для STEP
}

void loop() {
digitalWrite(8,LOW);//устанавливаем первое направление
for(int i=1; i<=200;i++){//проходим цикл 200 раз,
                         //то есть делаем 200 шагов
  digitalWrite(9,HIGH); //подаём сигнал на STEP, чтобы сделать шаг
  delay(2);             //маленькая пауза
  digitalWrite(9,LOW);  //отключаем сигнал на STEP
  delay(2);             //ещё маленькая пауза
}
digitalWrite(8,HIGH);//меняем направление и далее всё так же
for(int i=1; i<=200;i++){
  digitalWrite(9,HIGH);
  delay(2);
  digitalWrite(9,LOW);
  delay(2);
}
}
```

Скорость можно регулировать, меняя время задержки между шагами или используя более мелкие шаги, как было написано выше.

Драйверы типа H-мост (четырёхконтактные).

Существенно сложнее управлять через драйвер другого типа — с 4 управляющими контактами, связанными с обмотками. В этом случае мы должны сами в программе переключать обмотки в нужном порядке.

Ниже пример для мотора 28BYJ-48 с его "родным" драйвером. Он униполярный, поэтому у него как бы 4 обмотки.

Сделаем программу полного шага. Таблица показывает какие обмотки (по цветам проводов) в каком порядке включать (1 — включено, 0 — выключено).

ПОЛНОШАГОВЫЙ РЕЖИМ				
ШАГ	СИНИЙ (А)	РОЗОВЫЙ (В)	ЖЁЛТЫЙ (А')	ОРАНЖЕВЫЙ (В')
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

```
void setup() {
    pinMode(2, OUTPUT);//синий провод
    pinMode(3, OUTPUT);//розовый провод
    pinMode(4, OUTPUT);//жёлтый провод
    pinMode(5, OUTPUT);//оранжевый провод
}
void loop() {
//первый шаг:
    digitalWrite( 2, HIGH );
    digitalWrite( 3, HIGH );
    digitalWrite( 4, LOW );
    digitalWrite( 5, LOW );
    delay(5);
//второй шаг:
    digitalWrite( 2, LOW );
    digitalWrite( 3, HIGH );
    digitalWrite( 4, HIGH );
    digitalWrite( 5, LOW );
    delay(5);
//третий шаг:
    digitalWrite( 2, LOW );
    digitalWrite( 3, LOW );
    digitalWrite( 4, HIGH );
    digitalWrite( 5, HIGH );
    delay(5);
//четвёртый шаг:
    digitalWrite( 2, HIGH );
    digitalWrite( 3, LOW );
    digitalWrite( 4, LOW );
    digitalWrite( 5, HIGH );
    delay(5);
}
```

Аналогично можно легко сделать программу для полушагового режима, руководствуясь следующей таблицей:

ПОЛУШАГОВЫЙ РЕЖИМ (HALF-STEP MODE)				
ШАГ	СИНИЙ (A)	РОЗОВЫЙ (B)	ЖЁЛТЫЙ (A′)	ОРАНЖЕВЫЙ (B′)
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Какой шаг выбрать?

Полный шаг используется в случаях, когда нужна большая скорость и не важна точность. При малых скоростях мотор на полном шаге работает шумно, сильно вибрирует. Чаще используют ½ или ¼ шага (если драйвер поддерживает). Если нужно плавное движение на небольшой скорости — подбирайте размер шага путём экспериментов. Микрошаг (1/16, 1/32) — наиболее плавный и тихий, но сильно разогнать мотор не получится: драйвер с мотором просто не могут выдавать более чем несколько тысяч шагов в секунду, а для микрошага это немного. Зато микрошаг даёт высокую точность и применяется в принтерах, дисководах и т.п..

Управление через библиотеку AccelStepper.

AccelStepper — одна из самых популярных библиотек для управления шаговыми моторами. Она может управлять мотором через драйвер любого типа. Библиотека работает по своеобразной логике, когда сначала задаются параметры движения (позиция, скорость), а потом уже выполняется движение, причём, команду движения нужно повторять непрерывно. Эти сбивает с толку новичков. Впрочем, есть и команды моментального исполнения.

Эта библиотека не умеет управлять размером шага,

Подключение библиотеки:

#include <AccelStepper.h>

Создание объекта для мотора:

AccelStepper mystepper(1,пин Step,пин Dir); — для драйверов Step/Dir

AccelStepper mystepper(4, pinA1, pinA2, pinB1, pinB2); — для драйверов типа Н-мост с четырьмя управляющими контактами.

Во втором случае можно просто написать:

AccelStepper mystepper; — по умолчанию пины 2, 3, 4, 5.

Есть ещё вариант для двухконтактного подключения биполярного мотора к Н-мосту:

AccelStepper mystepper(2, pinA, pinB);

но для этого подключения нужна особая схема, мы её использовать не будем.

Настройка параметров скорости:

Эти параметры можно установить один раз (в setup), они работают потом до тех пор, пока не будут переопределены.

mystepper.setMaxSpeed(шагов в секунду); — максимальная скорость. Если она установлена — мотор стремится по-возможности её достичь (если успевает).

mystepper.setAcceleration(шагов в секунду в секунду); — ускорение, с которым мотор будет пытаться достичь максимальной скорости, а перед остановкой будет с тем же ускорением замедляться. Если не указано, то будет разгоняться и замедляться очень медленно.

Возможность ускорения считается одной из фишек этой библиотеки, но не всегда это нужно. Если вы хотите чтобы мотор сразу раскручивался до нужной скорости — просто поставьте достаточно большое ускорение (например, 5000).

Управление через позицию (по количеству шагов):

Абсолютная позиция это позиция (в шагах) относительно начального положения мотора (в момент запуска программы). Может быть как + так и -.

Команды движения до заданной позиции исполняются с переменной скоростью, которая зависит от *setMaxSpeed* и *setAcceleration*.

mystepper.moveTo(абсолютная позиция); — вращать до указанной позиции (в шагах). Для исполнения нужно многократно повторять функцию *run()* или *runSpeedToPosition()*.

mystepper.move(расстояние в шагах); — перемещение на указанное количество шагов, считая от текущей позиции (может быть + и -). Для исполнения нужно многократно повторять функцию *run()* или *runSpeedToPosition()*.

mystepper.run(); — выполнить движение до заданной позиции (абсолютной или относительной). Команду нужно повторять в цикле до полного исполнения. Используется, когда параллельно с движением нужно ещё что-то делать (в том числе двигать второй мотор).

mystepper.runToPosition(); — выполнить заданное движение за один раз. То есть, эту функцию повторять не нужно. Используется, когда ничего больше делать не нужно и можно просто подождать окончания движения.

stepper.runToNewPosition(абсолютная позиция в шагах); — команда мгновенного исполнения: мотор сразу двигается к заданной позиции.

stepper.setCurrentPosition(число шагов); — единственная команда, изменяющая "точку отсчёта". Она устанавливает значение текущей позиции равным указанному числу. Если указать 0, то эта позиция будет установлена как новая точка отсчёта.

Управление через скорость (шагов в секунду):

mystepper.setSpeed(шагов в секунду); — установить скорость для будущего движения (команда *runSpeed()*). Не работает, если *setMaxSpeed* меньше (тогда будет выбрана меньшая скорость) или *setMaxSpeed* вообще не задана (тогда будет двигаться еле-еле). Не работает, если после setSpeed есть команда управления через позицию.

mystepper.runSpeed(); — исполнить движение с заданной скоростью. Команду нужно повторять в цикле. Движение не ограничено по позиции.

Управление по скорости и позиции:

stepper.runSpeedToPosition(); — исполнить движение с заданной скоростью до заданной позиции. Команду нужно повторять в цикле. Скорость должна быть задана ПОСЛЕ задания позиции.

Другие команды:

stepper.stop(); — остановка.

mystepper.currentPosition(); — возвращает текущую абсолютную позицию.

mystepper.distanceToGo(); — возвращает количество шагов, которое осталось до заданной ранее цели.

Примеры.

Рассматриваем управление мотором с поворотом 1,8 градуса на шаг, то есть, 200 шагов на полный оборот (в полношаговом режиме).

Во всех примерах в начале должны стоять строки подключения библиотеки и мотора. Например, если мы пользуемся драйвером типа Step/Dir на портах 8 и 9:

```
#include <AccelStepper.h>   //подключение библиотеки
AccelStepper stepper(1,9,8); //создание объекта stepper, Step
                             // подключен к 9-му пину, Dir — к 8-му.
```

Повторять их в каждом примере не будем.

Пример 1: движение с постоянной скоростью.

```
void setup()
{
   stepper.setMaxSpeed(1000);//максимальная скорость 1000 шагов/с
   stepper.setSpeed(300);    //скорость 300 шагов/с
}
void loop()
{
   stepper.runSpeed(); //движение — бесконечно, так как в цикле loop
}
```

Пример 2: крутим один раз 10 оборотов (2000 шагов) с ускорением 200 и максимальной скоростью 800 шагов в секунду. Скорость сначала растёт от 0 до 800, потом уменьшается обратно до нуля. Поскольку 2000 шагов это много — мотор успевает пройти полный цикл по скорости.

```
void setup()
{
  stepper.setMaxSpeed(800);//устанавливаем максимальную скорость
  stepper.setAcceleration(200);//устанавливаем ускорение
  stepper.moveTo(2000);//устанавливаем цель: абсолютная позиция 2000
}
void loop()
{
   stepper.run();// повторяем run в цикле loop, движение само
                 // прекращается по достижении позиции 2000
}
```

Пример 3: Крутим непрерывно по 5 оборотов (1000 шагов) с промежутком 1 секунда. Используем на этот раз runToPosition(). Ускорение сделаем большое, чтобы мотор разгонятся и останавливался мгновенно.

```
void setup()
{
  stepper.setMaxSpeed(800);       //задаём максимальную скорость
```

```
  stepper.setAcceleration(10000);// с таким ускорением мотор сразу
                                 // разгонится до максимальной скорости
  }
void loop()
{
   stepper.move(1000);// задаём цель перемещение на 1000 шагов
                      // относительно текущей позиции
   stepper.runToPosition();// запускаем движение, эта функция не
                           // требует повторов, она исполняется до
                           // достижения цели
   delay(1000);// пауза 1 секунда
}
```

Пример 4: использование цикла while с функцией distanceToGo().

```
void setup()
{
  stepper.setMaxSpeed(800);// задаём максимальную скорость
  stepper.setAcceleration(400);// задаём ускорение
 }

void loop()
{
  stepper.move(2000); // относительное перемещение на 2000 шагов
   while(stepper.distanceToGo()!=0){//пока остаток шагов до цели
   stepper.run();                  // не равен нулю — крутим мотор
   }
   delay(1000);                    // пауза 1 секунда
}
```

Пример 5: использование runToNewPosition для перемещения в одну команду. Мотор делает 2 оборота вперёд, потом оборот назад, потом ещё оборот назад (возвращаясь на ноль).

```
void setup()
 {
  stepper.setMaxSpeed(400);//устанавливаем максимальную скорость
  stepper.setAcceleration(200);//устанавливаем ускорение
 }

void loop()
{
  stepper.runToNewPosition(0);// перемещаемся на нулевую позицию
  delay(1000);                // пауза 1 секунда
```

```
  stepper.runToNewPosition(400);//перемещаемся на позицию 400
                                // (2 оборота)
  delay(1000);                 // пауза 1 секунда
  stepper.runToNewPosition(200);// перемещаемся на позицию 200
                                // (фактически возвращаемся на 1
                                // оборот назад)
  delay(1000);                  // пауза 1 секунда
}
```

Пример 6: смена точки отсчёта функцией setCurrentPosition. Первое смещение будет на +200 (один оборот вперёд), но после того как эта позиция превратится в 400, повторное смещение в позицию 200 уже приведёт к повороту назад (логично: мы стоим на 400, а нам нужно 200!). Таким образом, все дальнейшие повороты (кроме первого) будут назад на 1 оборот.

```
void setup()
{
  stepper.setMaxSpeed(200.0);     // скорость
  stepper.setAcceleration(100.0);// ускорение
  }
void loop()
{
  stepper.runToNewPosition(200);// перемещаемся на позицию 200
   delay(1000);
  stepper.setCurrentPosition(400);// переобозначаем эту позицию:
                                  // теперь это не "200", а "400"
   delay(1000);
}
```

Пример 7: движение к позиции с заданной скоростью (повтор runSpeedToPosition в цикле).

```
void setup()
{
   stepper.setMaxSpeed(1000); // задаём максимальную скорость
}
void loop()
{
 stepper.move(200); // задание на перемещение на 200 шагов вперёд
 stepper.setSpeed(800); // устанавливаем скорость (не больше
                        // максимальной)
 while(stepper.distanceToGo()!=0){ //пока не дошли до нужной позиции
  stepper.runSpeedToPosition(); // двигаемся с заданной скоростью
 }
 delay(1000);
}
```

Пример 8: одновременное управление тремя моторами. Программа заставит первый мотор вращаться туда-сюда по 24 шага, а остальные два мотора будут просто непрерывно вращаться вперёд.

```
AccelStepper stepper1(1,9,8); //подключаем три мотора
AccelStepper stepper2(1,7,6);
AccelStepper stepper3(1,5,4);
void setup()
{
    stepper1.setMaxSpeed(200);// скорость первого мотора
    stepper1.setAcceleration(100);// ускорение первого мотора
    stepper1.moveTo(24); // движение на абсолютную позицию 24
    stepper2.setMaxSpeed(300);// скорость второго мотора
    stepper2.setAcceleration(100);// ускорение второго мотора
    stepper2.moveTo(1000000);// движение на абсолютную позицию
                             // 1000000 (фактически бесконечно)
    stepper3.setMaxSpeed(300);    // всё то же самое для 3-го мотора
    stepper3.setAcceleration(100);// как для второго
    stepper3.moveTo(1000000);
}
void loop()
{
    if (stepper1.distanceToGo() == 0)//если первый мотор дошёл до
                                     // конца (все свои 24 шага)
     stepper1.moveTo(-stepper1.currentPosition());//меняем ему
                                                  //задание на обратное
    stepper1.run(); // движение всех трёх моторов
    stepper2.run(); //(каждый выполняет своё задание)
    stepper3.run();
}
```

Управление через библиотеку StepperDriver.

Библиотека StepperDriver работает только для биполярных моторов с драйверами типа Step/Dir. https://github.com/laurb9/StepperDriver. В отличие от AccelStepper она умеет сама управлять размером шага (только для нескольких конкретных драйверов: A4988, DRV8825, DRV8834, DRV8880). В простом режиме (без управления размером шага) она работает с любыми драйверами Step/Dir. Размер шага при этом вы должны задать другими способами (в зависимости от конкретного драйвера), а библиотеке только сообщить этот размер. В состав библиотеки входит несколько модулей: универсальный и отдельный для каждого из 4-х поддерживаемых драйверов.

Примеры.

Пример 1: простая программа поворота на 1 оборот вперёд и сразу назад

```
//подключаем модуль простого универсального драйвера
#include "BasicStepperDriver.h"

// Создаём объект-мотор с указанием числа шагов на оборот (первое
// число), номера пина DIR (второе число) и номера пина STEP (третье
// число). В нашем примере мотор на 200 шагов на оборот, DIR
// подключен к цифровому порту 8, STEP к 9.
BasicStepperDriver stepper(200, 8, 9);

//А можно ещё указать четвёртое число — порт ENABLE:
//BasicStepperDriver stepper(MOTOR_STEPS, DIR, STEP, ENABLE);

void setup() {
// включаем мотор: первое число — скорость в шагах в секунду, второе
// число — размер шага: 1 полный шаг, 2 - полушаг, 4 - 1/4 шага
// и т.д. Размер шага мы устанавливаем сами, а здесь просто сообщаем
// его библиотеке, чтобы она правильно считала скорость. В нашем
// примере мы используем полный шаг и скорость 100 шагов в секунду
  stepper.begin(100, 1);
}

void loop() {
  stepper.rotate(360);// поворот на 360 градусов
  stepper.move(-200); // поворот в обратную сторону на 200 шагов
  delay(2000);
}
```

Пример 2. Переделаем ту же программу для шага 1/8. Допустим. мы используем драйвер DRV8825 и его контакты, управляющие размером шага (M0, M1, M2) подключены, соответственно, к пинам 10, 11, 12.

```
#include "BasicStepperDriver.h"

BasicStepperDriver stepper(200, 8, 9);

void setup() {
  stepper.begin(100, 8);// 8 обозначает шаг 1/8

// Теперь готовим к работе порты для управления размером шага и
// устанавливаем на них комбинацию для шага 1/8 (смотри таблицу)
  pinMode(10, OUTPUT);
```

```
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  digitalWrite(10, HIGH);
  digitalWrite(11, HIGH);
  digitalWrite(12, LOW);
}

void loop() {
  stepper.rotate(360);// поворот на 360 градусов
  stepper.move(-200*8); // поворот в обратную сторону на 200 шагов,
                        // но не забываем умножить на 8, потому что
                        // шаги у нас сейчас 1/8.
  delay(2000);
}
```

Пример 3. Управление размером шага через библиотеку.

Теперь используем модуль для драйвера DRV8825. Теперь библиотека сама может задавать размер шага.

```
#include "DRV8825.h"//подключаем библиотеку (модуль DRV8825)

// Создаём объект-мотор с указанием параметров: число шагов на
// оборот, пин DIR, пин STEP, пин ENABLE, пины M0, M1, M2
DRV8825 stepper(200, 8, 9, 13, 10, 11, 12);
void setup() {
  stepper.begin(100, 8);// скорость 100, шаг 1/8
}

void loop() {
  stepper.rotate(360);
  stepper.move(-200*8);
  delay(2000);
}
```

Другие команды библиотеки.

Команда begin может не указывать размер шага, а содержать только скорость:
stepper.begin(шагов в секунду);

Размер шага можно указывать отдельной командой, например:
stepper.setMicrostep(1);
stepper.setMicrostep(4);
stepper.setMicrostep(16);

Команды включения и выключения питания мотора. По умолчанию сразу после begin питание включается и мотор удерживает позицию (его трудно повернуть рукой). Но если вы подключили к Arduino порт ENABLE и определили его в строке создания объекта, то можно этими командами включать и отключать питание.

stepper.enable(); — включение

stepper.disable(); — отключение (теперь мотор можно вращать рукой)

Библиотека также имеет функции, назначающие цель движения в градусах или шагах (как в AccelStepper, но только относительно текущего положения):

stepper.startMove(число шагов); — сколько пройти в шагах

stepper.startRotate(градусов); — сколько пройти в градусах

Задав цель, нужно (как и в AccdelStepper) повторять в цикле команду действия. Мотор будет двигаться, пока цель не достигнута. Здесь команда действия такая:

stepper.nextAction();

Эта функция также возвращает время в микросекундах до следующего шага. Если время стало равно нулю значит мотор уже закончил движение. В примере к библиотеке предлагается использовать это так:

```
// Записываем в переменную время до следующего шага в микросекундах
// (при этом команда действия срабатывает и сама по себе и мотор
// движется
unsigned wait_time_micros = stepper.nextAction();

if (wait_time_micros <= 0) {//если 0 — значит мотор уже остановился
[здесь пишем что делать если мотор закончил двжение]
    }
if (wait_time_micros > 100){// если времени >100 мкс
[здесь пишем что делать если мотор крутится, и до следующего шага
ещё "долго"]
    }
```

Время до следующего шага зависит от размера шага и от скорости.

В процессе движения к цели мотор можно остановить:

stepper.stop();

Получить значение текущей скорости (шагов в секунду):

stepper.getCurrentRPM()

Настройка движения с ускорением. разгон происходит до скорости, указанной в строке begin. Указываются значения ускорения в шагах в секунду в секунду отдельно для разгона и для замедления:

stepper.setSpeedProfile(stepper.LINEAR_SPEED,разгон, замедление);

Например:

stepper.setSpeedProfile(stepper.LINEAR_SPEED,500, 200);

После этого можно запускать вращение любыми функциями и оно будет с ускорением и замедлением.

Начать торможение с заданным ускорением:

stepper.startBrake();

Использование сдвигового регистра 74HC595 для расширения портов

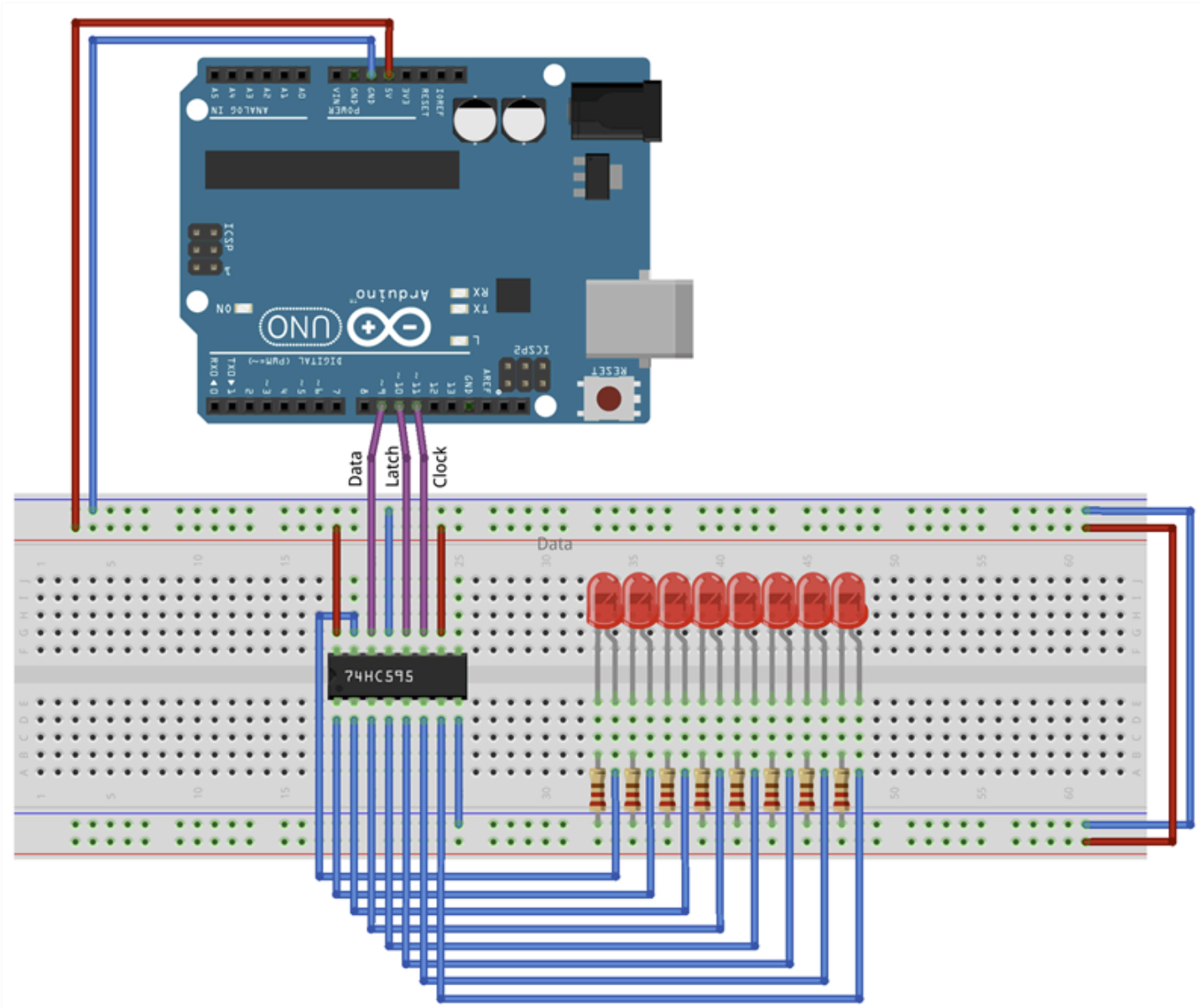
Портов на Arduino не так уж много, если делать серьёзные проекты. Есть разные способы увеличить их количество. Один из самых простых — использование дешёвой микросхемы 74HC595 — сдвигового регистра. Эта штука позволяет сделать из трёх цифровых портов - восемь!

Есть пара ограничений:

1. Работает только на вывод.

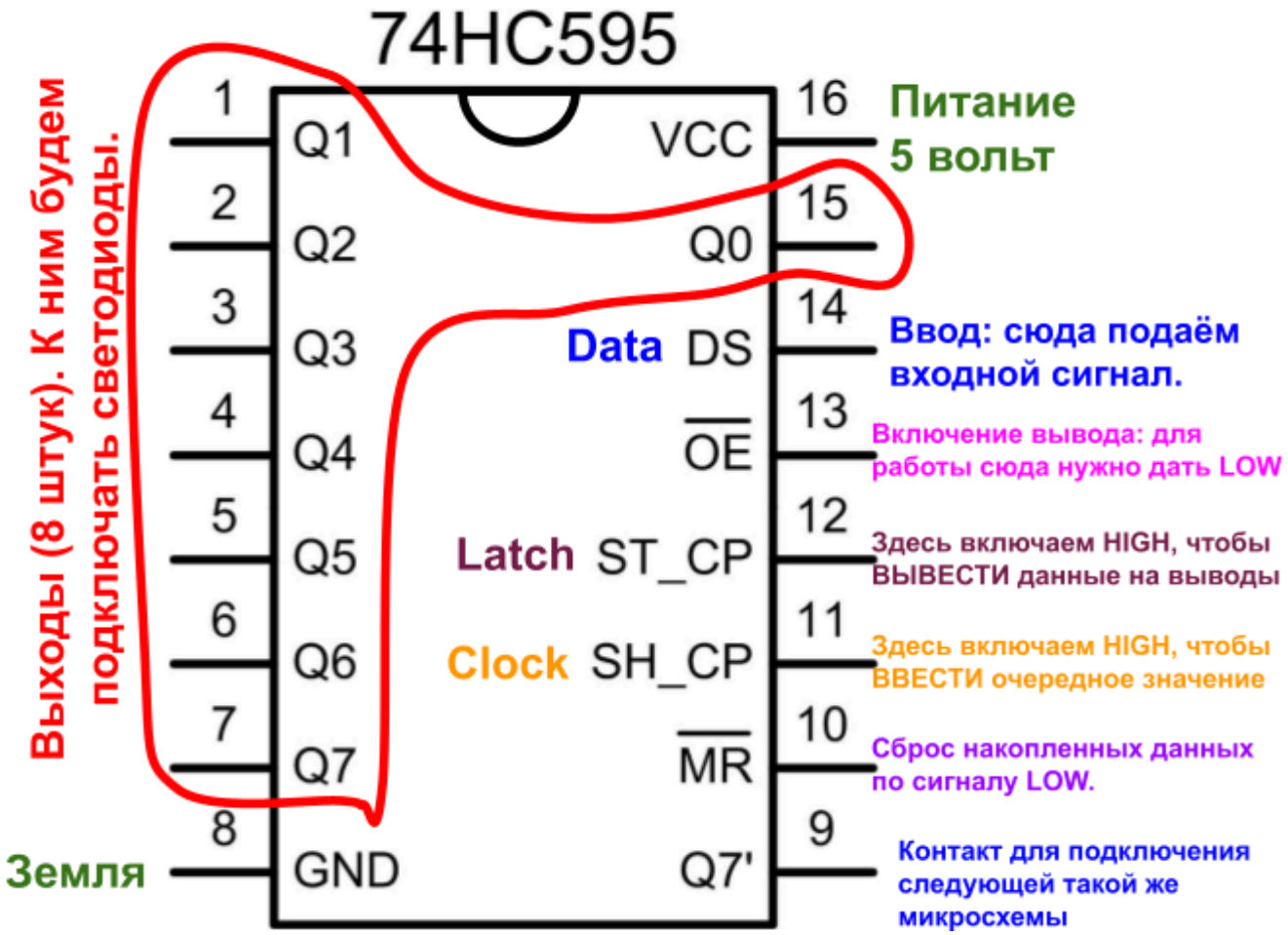
2. Сигналы для портов придётся передавать последовательно. Это занимает время. То есть, такой вывод немного медленнее, чем обычный. Но для большинства задач скорости хватает.

Схема подключения на примере подключения 8 светодиодов:



Для устранения помех рекомендуется также соединить latch и clock с землёй через небольшие конденсаторы.

Выводы и их назначение (сравни со схемой подключения вверху):



Как работает эта микросхема:

В микросхеме есть два хранилища (регистра) по 8 ячеек каждый. Внутренний (сдвиговый) и внешний (хранения). В первый приходят данные с входа (на схеме контакт №14) — единички и нули (HIGH и LOW). Ячейки второго уже напрямую подключены к выводам **Q0-Q7**.

Чтобы микросхема "знала" когда пора читать очередное значение, существует контакт **SH_CP** (№11) — **SH**ift register **C**lock **P**in или просто "**clock**" — контакт синхронизации ввода. Подавая сюда HIGH, мы приказываем микросхеме прочитать сигнал с входа (№14) и записать его в первую ячейку внутреннего (сдвигового) регистра (сдвинув все остальные дальше). Такие сигналы называются ещё "тактовые импульсы".

Когда в сдвиговом регистре накопились нужные нам значения (обычно заполняют все 8 ячеек, но это не обязательно), мы подаём HIGH на контакт **ST_CP** (№12) — **ST**orage register **C**lock **P**in — контакт синхронизации вывода. Подавая сюда HIGH, мы приказываем микросхеме передать накопленные значения внутреннего сдвигового регистра во внешний (регистр хранения), который подключен к выводам Q0-Q7. То есть это команда на вывод. Этот сигнал ещё называется "**latch**", что значит "защёлка". По щелчку этой "защёлки" накопленные данные освобождаются и вываливаются на выход.

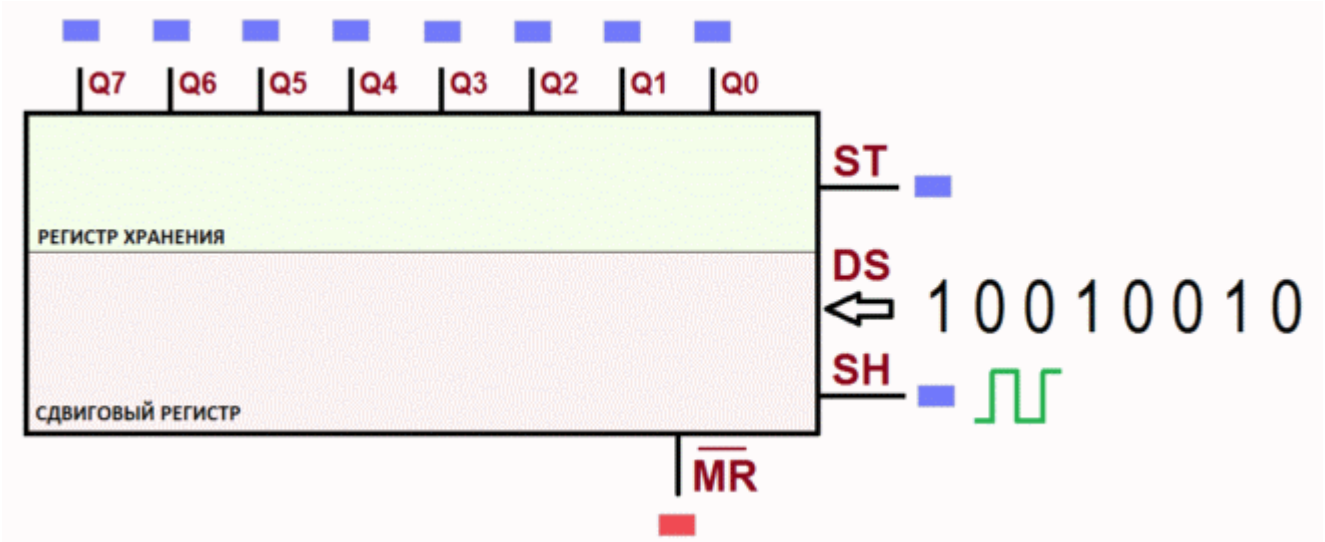
В итоге светодиоды, которым придёт 1 (HIGH) — загорятся.

MR (10) — контакт для сброса. Обычно здесь нужно держать HIGH, но если мы хотим обнулить ячейки сдвигового регистра, то нужно подать LOW. Обычно этого не требуется и этот контакт можно просто подключить к питанию (5 вольт).

OE (13) — контакт отключения вывода. Для работы нужно дать сюда LOW. Если подать сюда HIGH — вывод должен отключиться. Информация в выходном регистре сохранится. Правда, у меня этот контакт не работал. Можно просто подключить его к gnd.

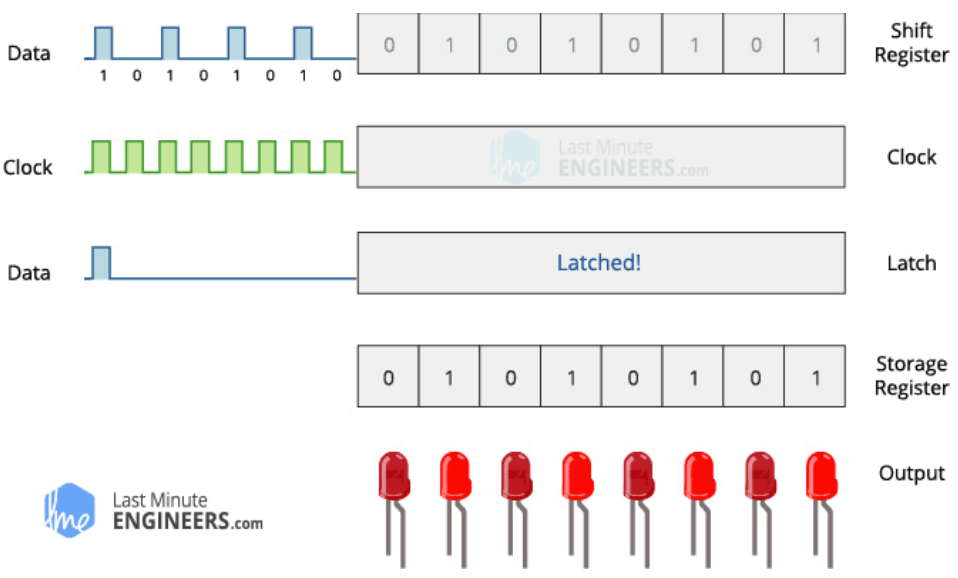
Q7' (9) — Обрати внимание: к выводу 9 можно подключить вход (14) второй микросхемы, ко второй — третью и так далее. Данные будут проходить "насквозь" через все регистры и мы сможем вместо 8 выходов получить 16, 24, 32 и так далее. Но нам пока столько светодиодов не нужно.

Анимация, иллюстрирующая работу сдвигового регистра:



Проследи внимательно за работой схемы. У каждого вывода есть цветной прямоугольничек, он горит красным, когда на этом контакте HIGH и синим, когда LOW. Сначала данные "заходят" в сдвиговый регистр через вход DS по сигналам SH (clock). Потом по команде HIGH на ST (lutch) они переходят вверх в регистр хранения. На выходах вверху появляется сигнал. Потом данные в нижнем регистре сбрасываются по сигналу LOW на контакт MR. Потом следует снова сигнал на ST и верхний регистр тоже обнуляется. И цикл начинается снова.

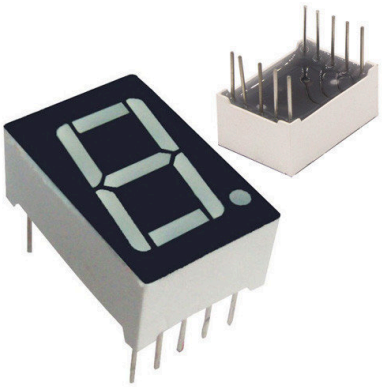
Обнулять значения совсем не обязательно. Можно просто вводить в сдвиговый регистр новые значения, они будут "выдавливать" старые. А старый сигнал на выходах будет сохраняться пока мы не "нажмём" ST, чтобы его обновить.



Справа ещё одна анимация. В верхней части идёт поток

данных, ниже — тактовые импульсы, ещё ниже — latch сигналы по которым цифры с верхнего регистра переходят в нижний.

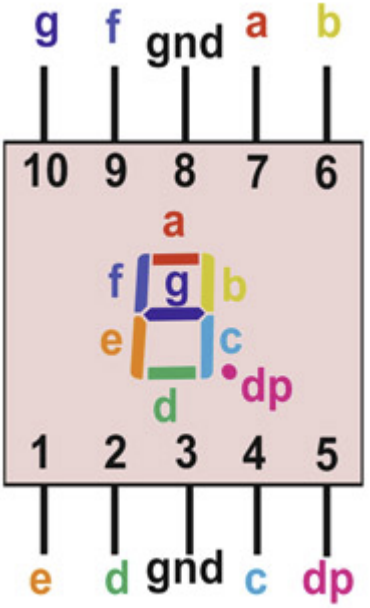
7-сегментный индикатор.



7-сегментный индикатор служит для вывода цифр. В принципе это просто 8 светодиодов, собранных так, чтобы они могли образовывать цифры. Цифры строятся из 7 сегментов, а восьмой это как бы десятичная точка.

У каждого светодиода свой отдельный вывод (обычно это "плюс"). А "земля" общая на всех. Такой вариант называется "с общим катодом[3]". Он используется чаще. В нём сегменты включаются подачей HIGH. Но бывает и версия с общим анодом — когда "плюс" общий, а земля у каждого своя. В этом случае сегменты включаются подачей LOW.

Мы будем рассматривать вариант с общим катодом.

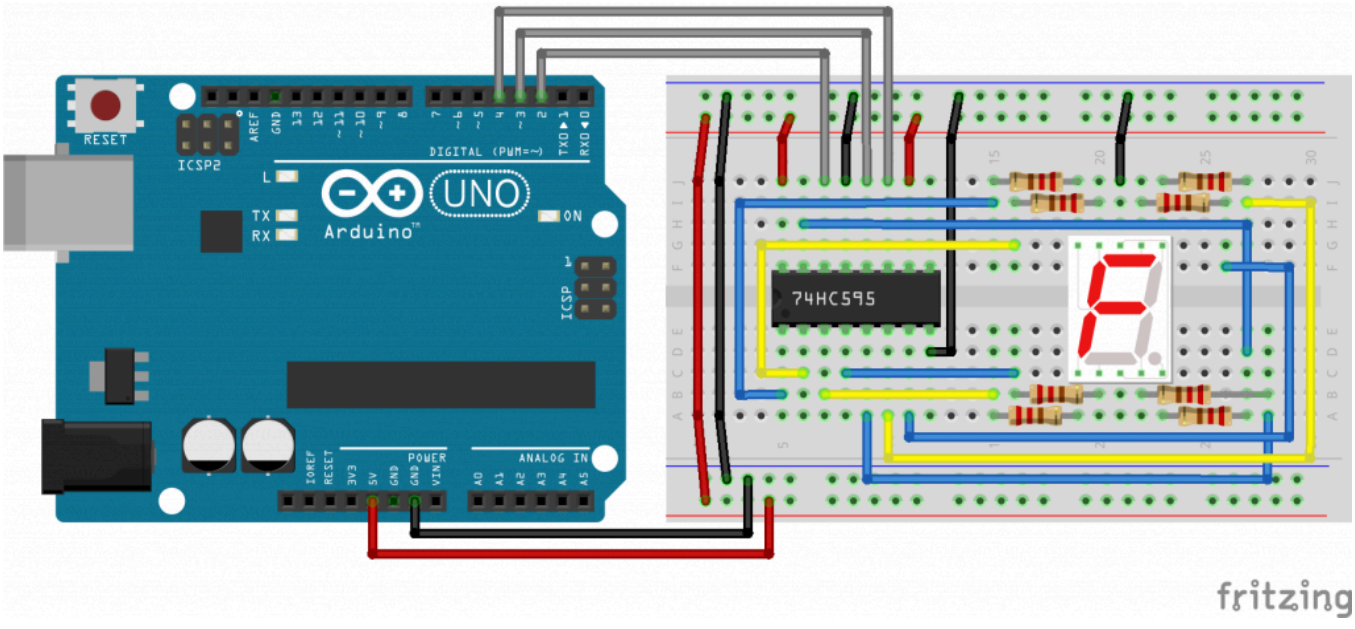


На индикаторе 10 контактов, они нумеруются по кругу. Два из них "земля". Сегменты названы буквами. Справа схема соответствия сегментов контактам.

Как видно, порядок нумерации ножек не совпадает с порядком буквенных обозначений сегментов. Как же подключать? В порядке ножек или в порядке сегментов? В принципе как угодно. Главное, не запутаться.

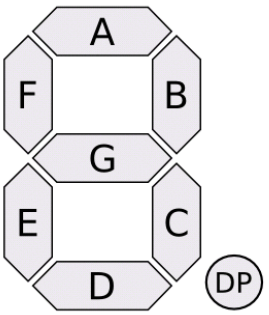
Просто подключаем все контакты сегментов в качестве светодиодов. Не забываем резисторы. Контакты 3 и 8 подключаются к земле, но достаточно только одного из них.

Чтобы сэкономить порты, воспользуемся сдвиговым регистром. Должно получиться что-то похожее на это (здесь подключение в порядке букв сегментов):



[3] Катод это отрицательный электрод. Анод — положительный.

Сдвиговый регистр удобен тем, что на него можно выводить готовое 8-значное двоичное число через функцию shiftOut(). Только сначала нужно подобрать такие двоичные числа, чтобы единички в них соответствовали нужным сегментам для каждой цифры. Например, тройка получится если включить A, B, C, D и G (смотри справа).



Составим таблицу включения сегментов для получения цифр. Единицы обозначают горящие сегменты, нули — не горящие. В правом столбце двоичные числа уже переведены в десятичные, чтобы было удобнее (для функции всё равно в каком виде записано число, а нам удобнее писать десятичные числа). В программе просто выведи код нужного числа в двоичном или десятичном виде через shiftOut.

Значения для shiftOut при подключении по порядку букв сегментов:

Контакт сдвигового регистра	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	**получившееся двоичное число (код)**	**соответствующее десятичное число**
Нога индикатора	7	6	4	2	1	9	10	5		
Сегмент	a	b	c	d	e	f	g	dp		
0	1	1	1	1	1	1	0	0	B11111100	**252**
1	0	1	1	0	0	0	0	0	B01100000	**96**
2	1	1	0	1	1	0	1	0	B11011010	**218**
3	1	1	1	1	0	0	1	0	B11110010	**242**
4	0	1	1	0	0	1	1	0	B01100110	**102**
5	1	0	1	1	0	1	1	0	B10110110	**182**
6	1	0	1	1	1	1	1	0	B10111110	**190**
7	1	1	1	0	0	0	0	0	B11100000	**224**
8	1	1	1	1	1	1	1	0	B11111110	**254**
9	1	1	1	1	0	1	1	0	B11110110	**246**

В этом варианте для включения десятичной точки нужно к любому числу добавить 1, так как точка у нас включается крайним правым разрядом.

Значения для shiftOut при подключении по порядку номеров ног индикатора:

Контакт сдвигового регистра	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	**получившееся двоичное число**	**соответствующее десятичное число**
Нога индикатора	1	2	4	5	6	7	9	10		
Сегмент	e	d	c	dp	b	a	f	g		
0	1	1	1	0	1	1	1	0	B11101110	**238**
1	0	0	1	0	1	0	0	0	B00101000	**40**
2	1	1	0	0	1	1	0	1	B11001101	**205**
3	0	1	1	0	1	1	0	1	B01101101	**109**
4	0	0	1	0	1	0	1	1	B00101011	**43**
5	0	1	1	0	0	1	1	1	B01100111	**103**
6	1	1	1	0	0	1	1	1	B11100111	**231**
7	0	0	1	0	1	1	0	0	B00101100	**44**
8	1	1	1	0	1	1	1	1	B11101111	**239**
9	0	1	1	0	1	1	1	1	B01101111	**111**

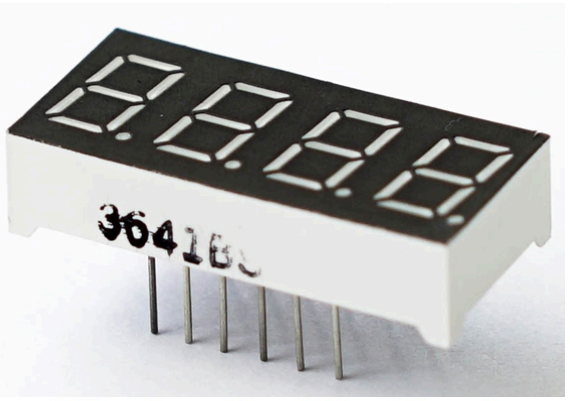
В этом варианте для включения десятичной точки нужно к любому числу прибавить 16, так как точка включается пятым разрядом с конца, который означает 16 (2^4=16).

Для понимания как переводятся двоичные числа в десятичные:



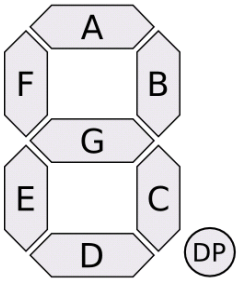
Подробнее о двоичной системе можно прочитать здесь.

4-х-разрядный 7-сегментный индикатор.



Кроме одиночных продаются и модули с несколькими индикаторами в одном корпусе. Рассмотрим такой модуль из 4-х индикаторов с общим катодом. У него два ряда по 6 контактов. На картинке внизу показано как они подключены.

A, B, C, D, E, F, G и десятичная точка (decimal или dp) — это всё знакомые нам сегменты. На них подаём напряжение ("плюс"). А вот контакты D1, D2, D3 и D4 это общие "минусы" разных разрядов. Они определяют, какой именно разряд загорится. Если мы подключим GND на D1 — будет светиться первая цифра, если на D2 — вторая и так далее.



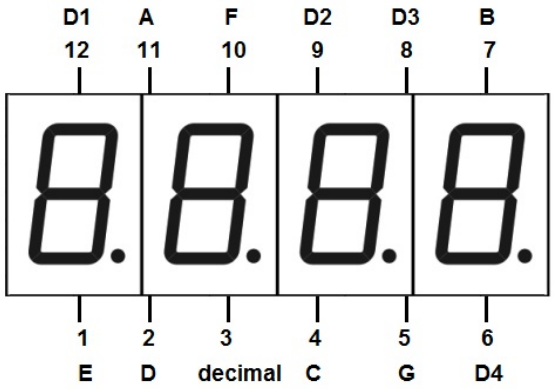


Схема с общим анодом такая же, но на общие контакты D1, D2, D3, D4 нужно подавать "плюс", а на сегменты — "минус" (GND).

Вот полные схемы двух вариантов:

Схема с общим катодом

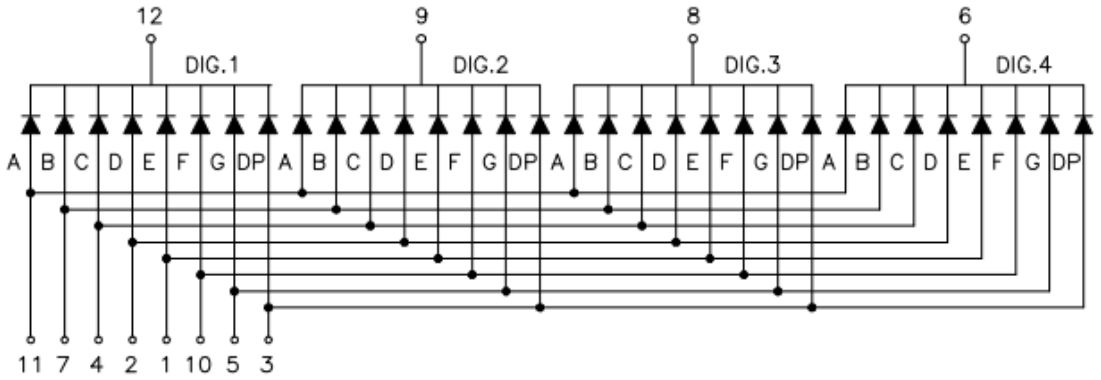
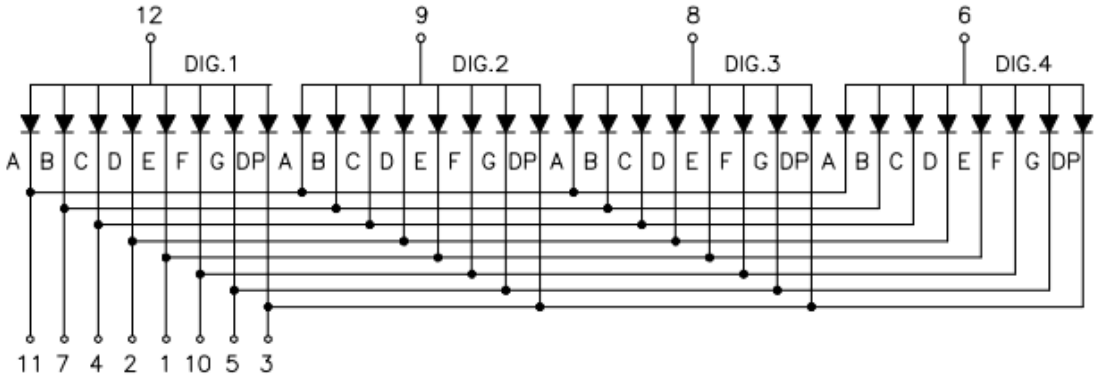


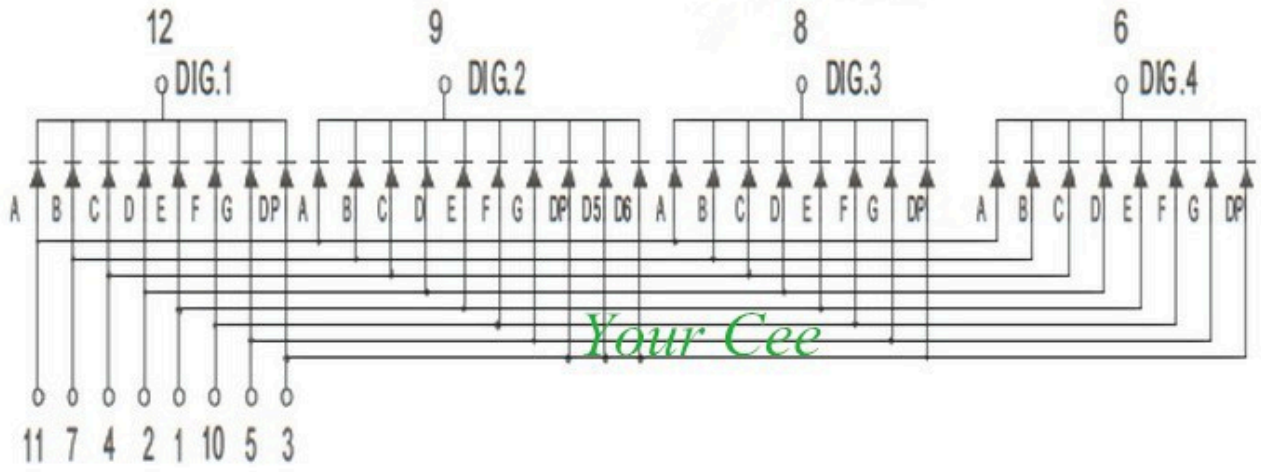
Схема с общим анодом



Бывают также модули, где посередине есть две точки, которые позволяют сделать как бы часы с двоеточием между часами и минутами или минутами и секундами.

Иногда для этих точек есть отдельный вывод, но чаще они совмещены с нижней точкой и либо горят сразу все три, либо нижняя точка в этом разряде отключена. Может оказаться, что и в других разрядах нижние точки не работают. В общем, нужно проверять.





Как заставить работать?

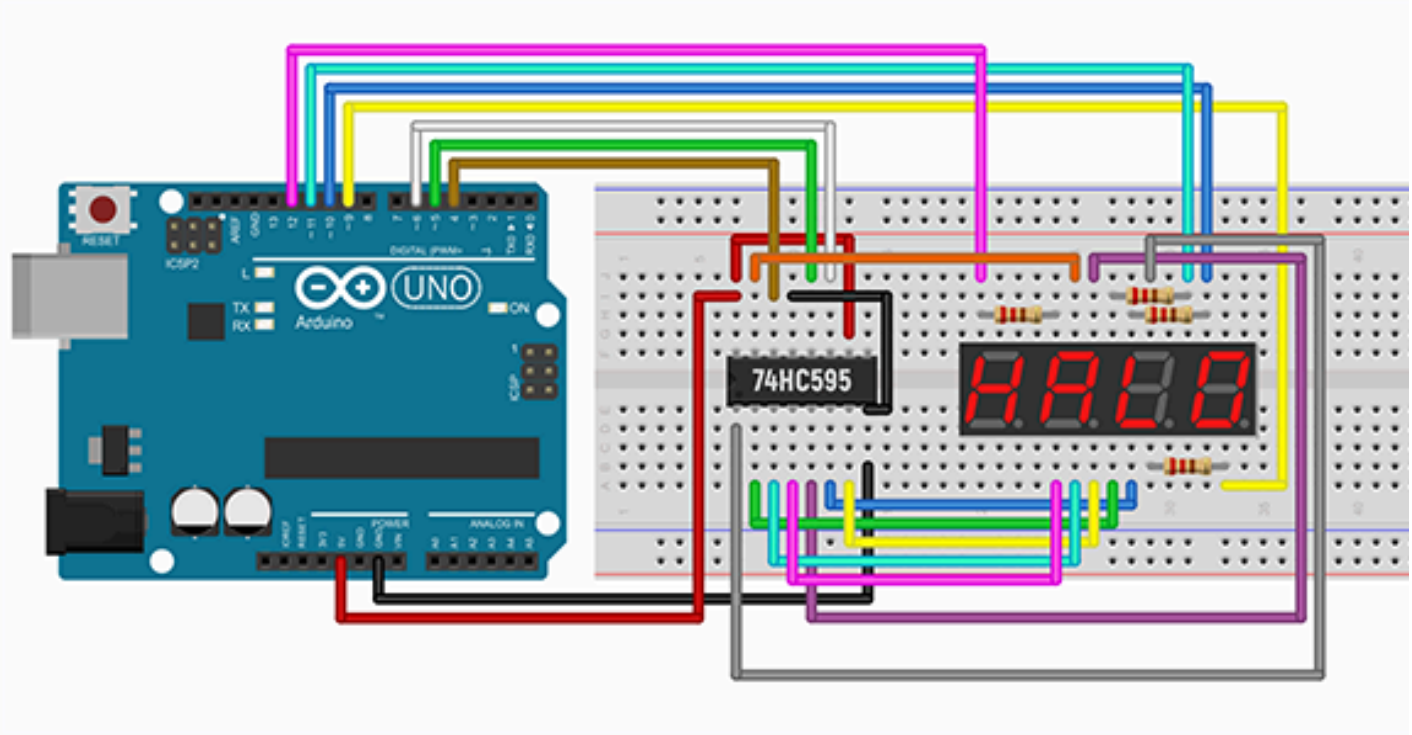
Как же сделать, чтобы одновременно в разных разрядах стояли разные цифры? А никак. Если мы включим одновременно больше одного разряда — там будет светиться одна и та же цифра. Выход в том, чтобы переключать разряды очень быстро, подставляя на каждый свою цифру. Для глаза будет казаться, что все светятся одновременно, и на каждом своя цифра.

Можно, конечно, и одновременно включить несколько разрядов, если нам действительно нужно на них одно и тоже, но это нежелательно, так как повышает нагрузку на Arduino, да и программа будет сложнее.

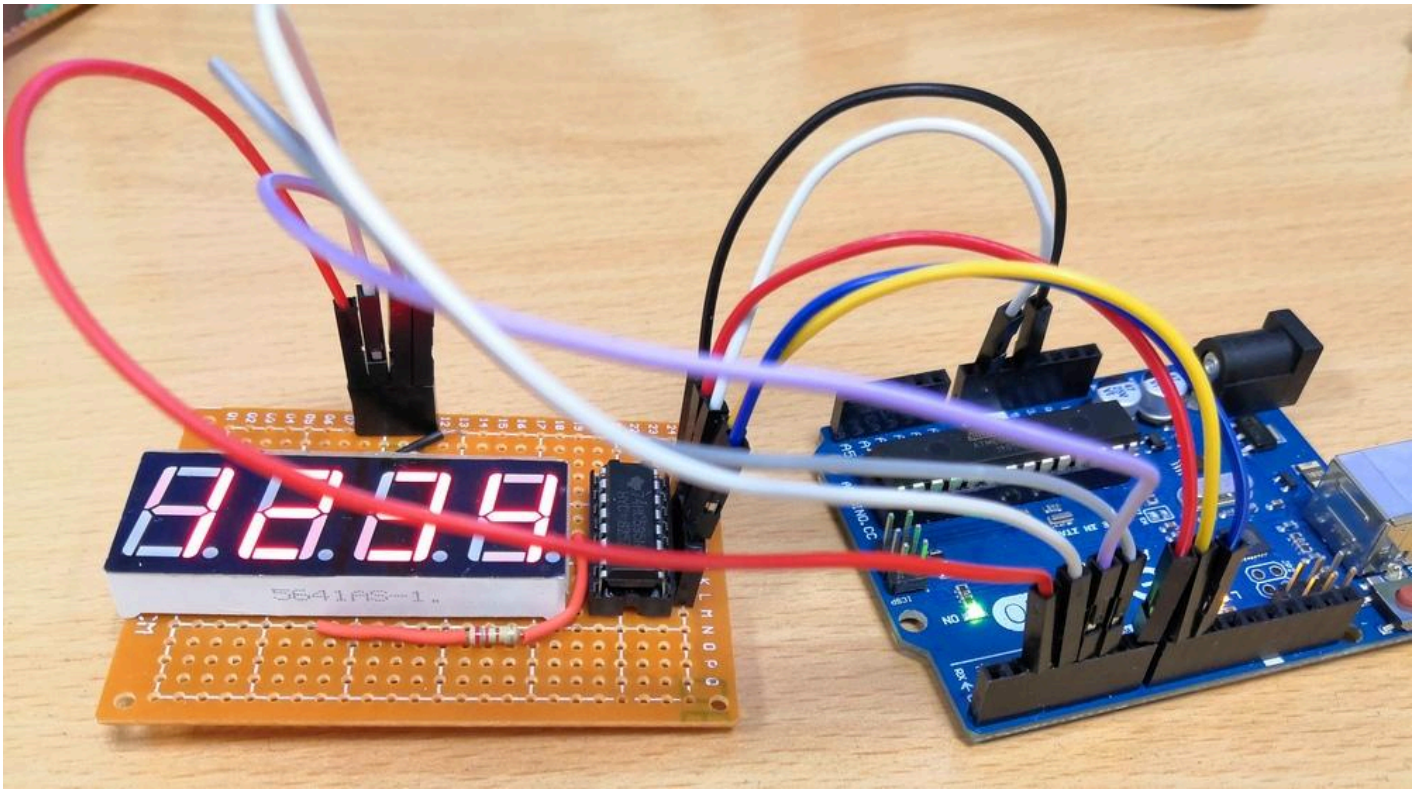
Сделаем пример программы, которая выводит произвольное 4-х значное целое число.

Используем сдвиговый регистр. Его подключаем точно так же, как в примерах для одиночного 7-сегментного индикатора: контакты Q0-Q7 от микросхемы — к сегментам от "a" до "dp" (контакты 11, 7, 4, 2, 1, 10, 5, 3). Добавляются только 4 цифровых порта для управления разрядами (D1, D2, D3, D4 — номера контактов 12, 9, 8 и 6 соответственно).

На картинке внизу показан пример такого подключения.



Только давайте мы для совместимости с предыдущими примерами используем для data, latch и clock те же порты, что использовали раньше (7, 8 и 9). Разряды подключим к портам 2, 3, 4 и 5. Такое подключение показано на фото:



Ниже пример программы для вывода 4-значного числа.

Пример программы.

```
#define data_pin 7 //как всегда задаём порты управления в
  макроопределениях
#define clock_pin 9
#define latch_pin 8
int digits[10] = {252, 96, 218, 242, 102, 182, 190, 224, 254, 246};
  //массив кодов цифр от 0 до 9
int digpin[4] = {2, 3, 4, 5};//Создаём массив с номерами портов,
  которые управляют разрядами. Так будет проще выбирать из них
  нужный.

void setup() {
  for (int i = 0; i < 4; i++) { //Готовим порты в цикле, чтобы не
  писать кучу строк.
    pinMode(digpin[i], OUTPUT);
  digitalWrite(digpin[i],HIGH);//На всякий случай сразу на них HIGH,
  чтобы светодиоды точно не светились (для свечения нужно сюда
  пустить LOW, это же у нас общие катоды).
  }
  pinMode(data_pin, OUTPUT);
  pinMode(clock_pin, OUTPUT);
  pinMode(latch_pin, OUTPUT);
  digitalWrite(latch_pin, LOW);
  Serial.begin(9600);
}

void loop() {
//Нам нужно вывести, к примеру, число 1234. Чтобы разложить его на
  отдельные цифры, преобразуем его в текстовый объект типа String.
  String num = String(1234); //Преобразовали и записали в массив num
  for (int i = 0; i < 4; i++) { //Перебираем все 4 разряда
    shiftOut(data_pin, clock_pin, LSBFIRST, digits[num[i]-48]);//
  Зачем вычитаем 48? Читай пояснение ниже.
    digitalWrite(latch_pin, HIGH);
    digitalWrite(latch_pin, LOW);
    digitalWrite(digpin[i],LOW);//Включаем очередной разряд
    delay(1);//Если не дать разряду погореть хотя бы одну
  миллисекунду, цифры будут почти не видны.
    digitalWrite(digpin[i],HIGH);//отключаем разряд, чтобы перейти к
  новому
  }
 }
```

Пояснение про разбор числа на цифры. В массиве "num" у нас не числа, а текстовые знаки типа char. А нам нужны числа. Например, там есть знак "2", а нам нужно сделать из него число 2. Это не так-то просто. Если просто попробовать перевести знак в цифру (это можно), то получим код этого знака. У текстового символа "2", например код 50. Можно воспользоваться тем, что в кодовой таблице все коды идут по порядку:

'0' — код 48,
'1' — код 49,
.....
'5' — код 53,
.....
'9' — код 57.

Значит, чтобы узнать число — можно из его кода вычесть 48 (код нуля).
Например, 53-48=5.

Есть другой вариант. Можно не переводить число в текст, а разобрать его на цифры другим способом:

1-й вариант: делением на 10 с остатком (оператор %). Остаток от деления на 10 как раз и будет последней цифрой числа.

5318%10=8
(5318-8)/10=531
531%10=1
и т.д..

2 вариант: целочисленное деление на 1000 даст первую цифру, потом можно вычесть найденное число тысяч и новое трёхзначное число делить на 100. И так далее.

5318/1000=5 (остаток автоматически отбрасывается, если целые)
5318-(5*1000)=318
318/100=3
и т.д..

То есть, можно несколькими делениями получить все цифры, сделать из них уже цифровой массив и потом выводить.

Если понадобится выводить **дробные числа**, использовать десятичную точку, то надо подумать как сделать удобнее, чтобы определить и целую часть и дробную и цифры в них.

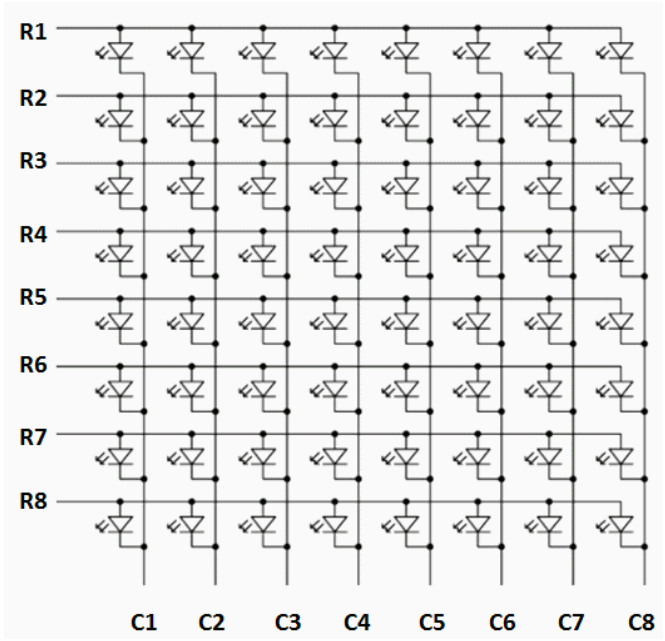
Так же придётся подумать как выводить числа **менее чем в 4 знака**. Наш пример этого не умеет, нужно его совершенствовать.

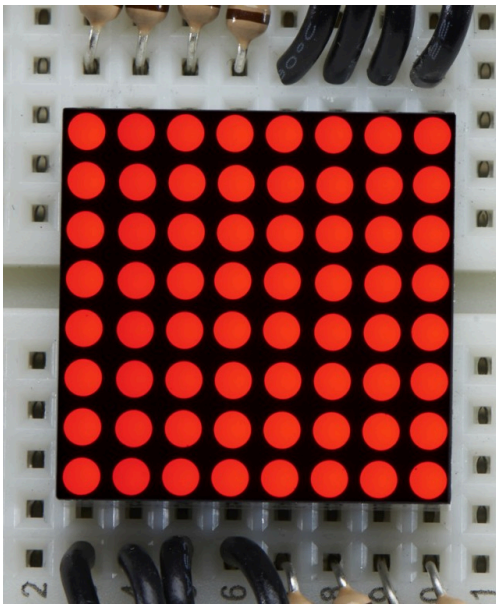
Ссылки с дополнительной информацией и примерами:

Несколько примеров с выводом разной информации.

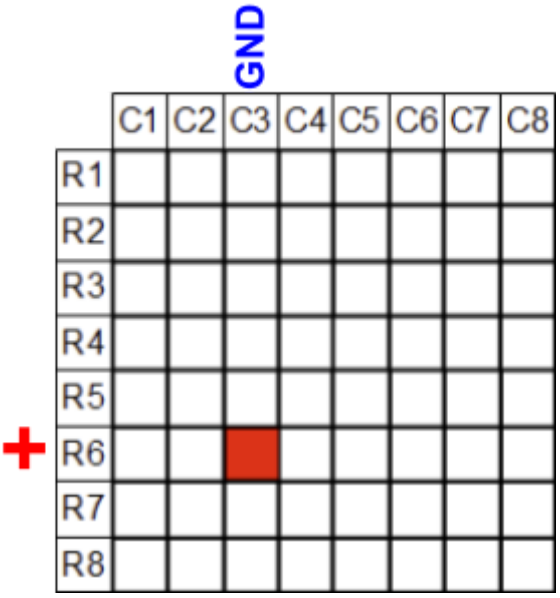
Светодиодная матрица 8x8.

Светодиодная матрица это 64 светодиода, расположенные "таблицей" в 8 строк и 8 столбцов. Они бывают разного размера. На схеме строки обозначены буквой R (ряды), а столбцы буквой C (колонки).





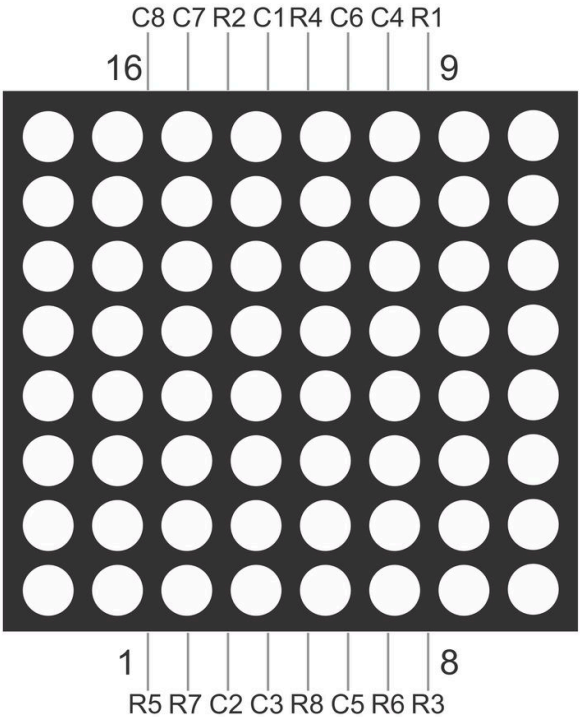
Как видно по схеме, в этой версии к строкам подключаются "плюсы", к столбцам "минусы". На алиэкспрессе это называется "общий анод".



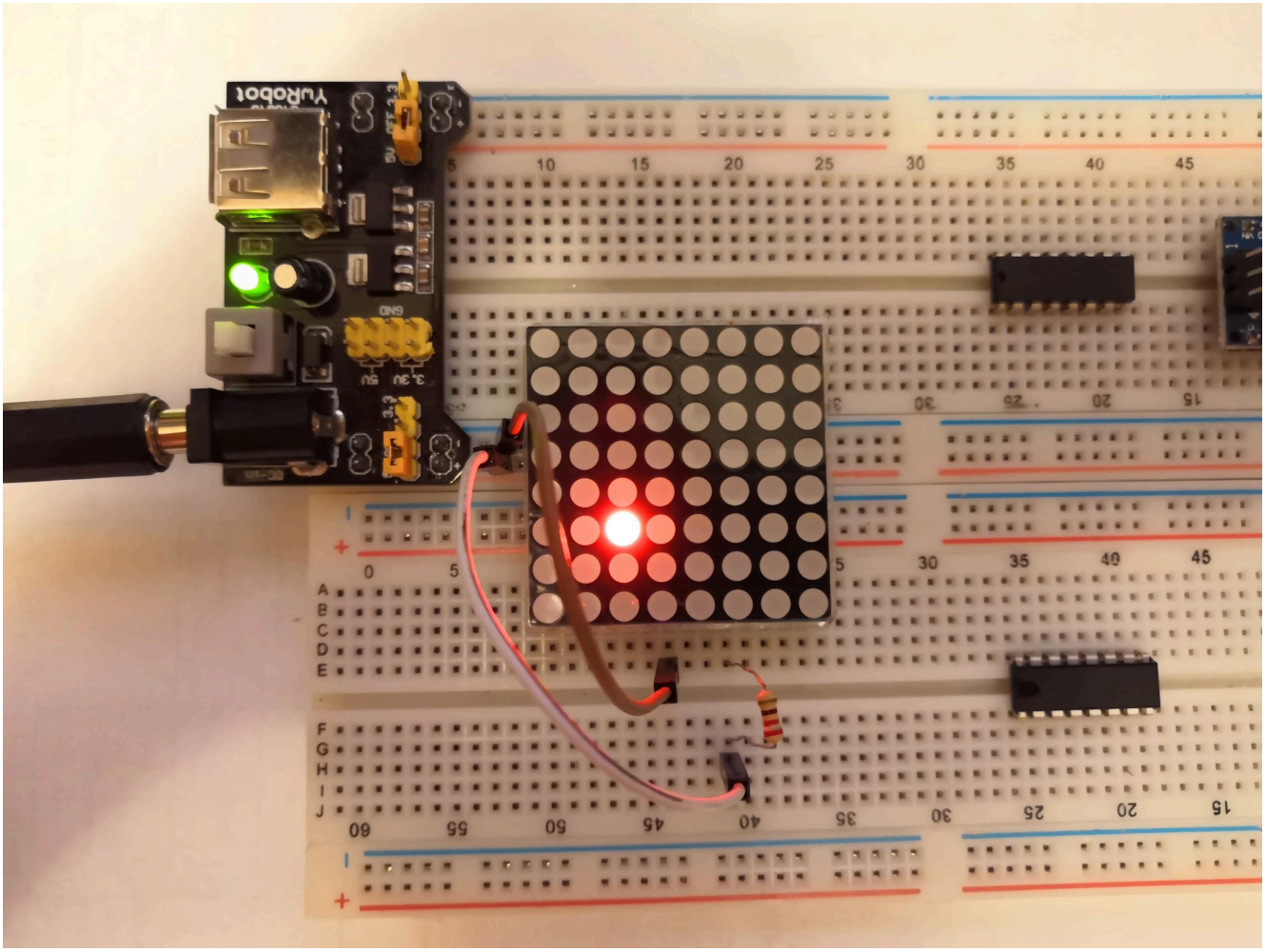
Светодиод загорится, если одновременно подать HIGH (+) на его строку и LOW (GND) на его столбец. Например, на рисунке слева это R6 и C8.

К сожалению, ножки идут не по порядку. Чтобы всё подключить правильно, сверяйся со схемой внизу. Матрицу нужно повернуть вниз той стороной, где нанесены надписи.



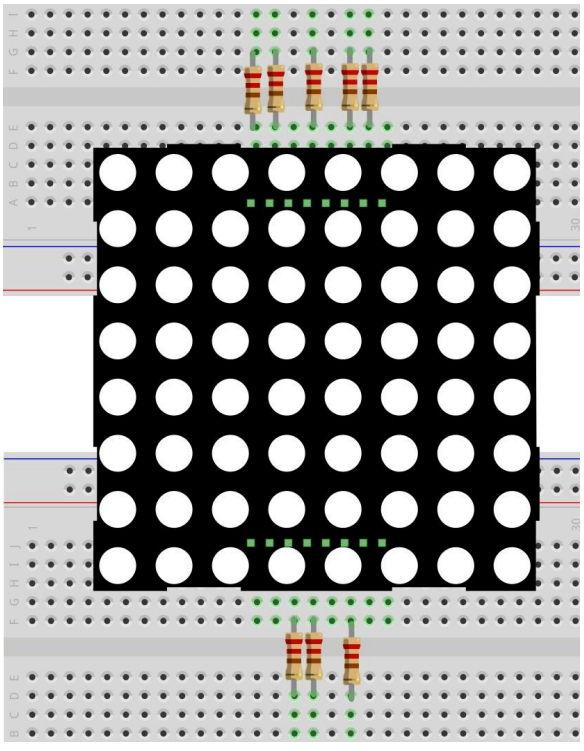


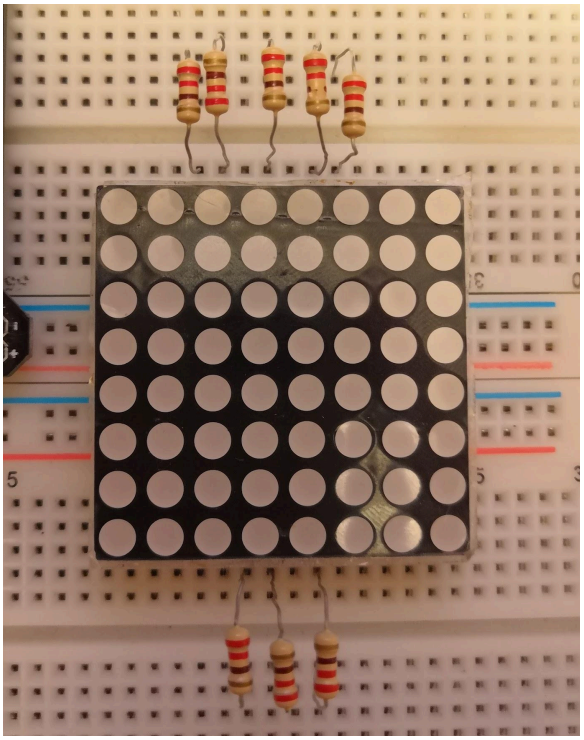
Работает! К R6 подключен "плюс", к C3 — "минус":



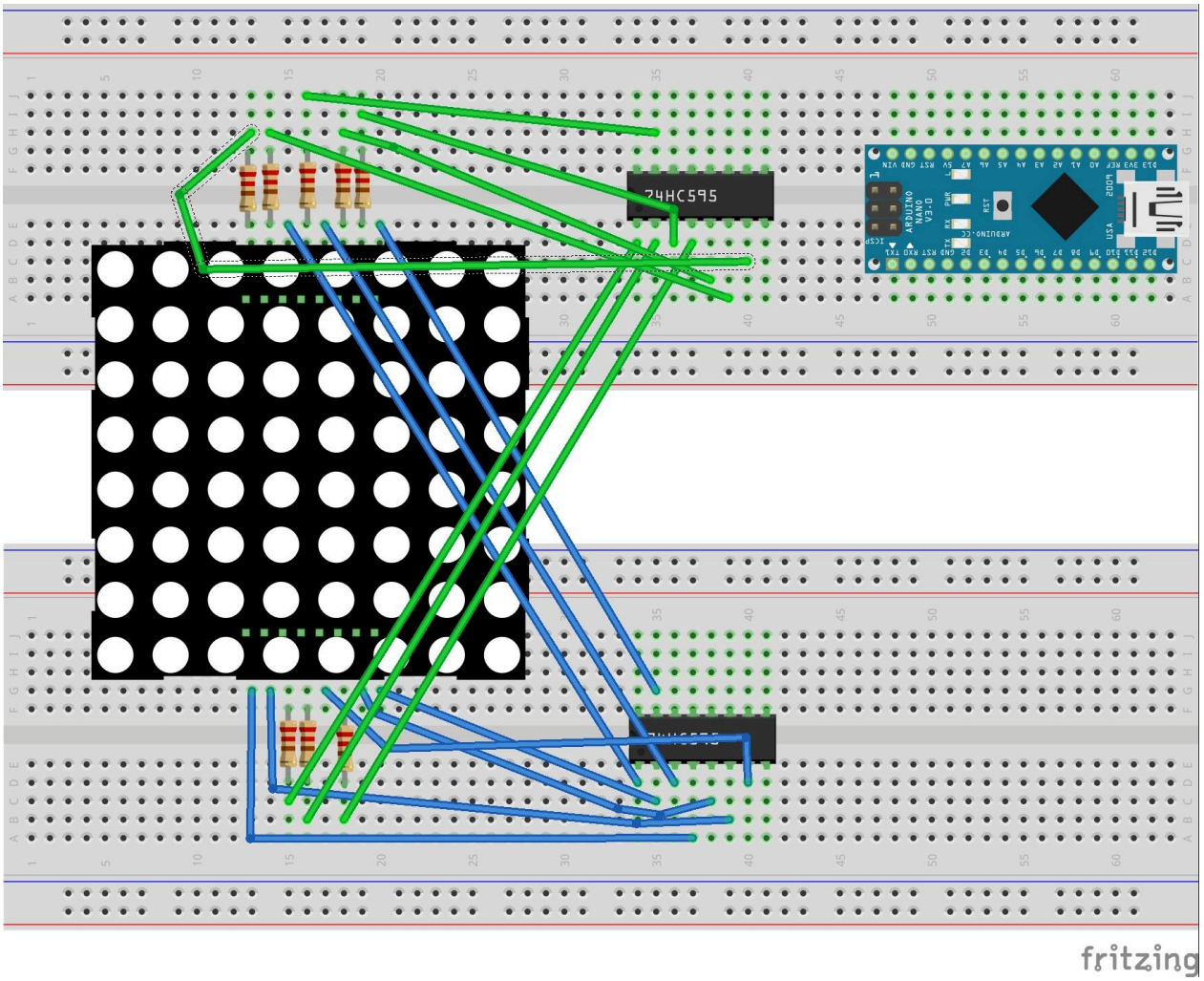
Подключение.

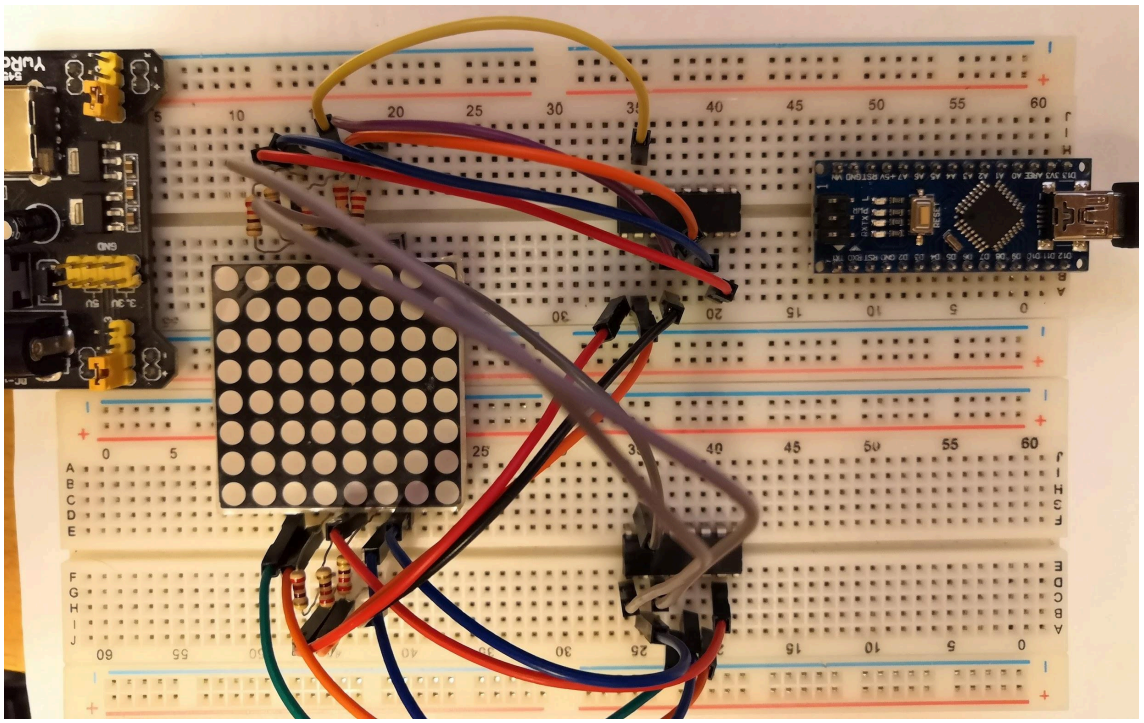
Чтобы получить осмысленный рисунок из пикселей, нужно рисовать его последовательно по строкам. Значит, подключим резисторы к столбцам, чтобы каждый раз резистор работал только с одним светодиодом:





Используем два сдвиговых регистра: один для выбора строки (его подключим к R1-R8 напрямую), другой для зажигания нужных точек на строке (его подключим к столбцам C1-C8 через резисторы 220 Ом. Соединять всё будем по порядку номеров. Это самый сложный момент — легко запутаться в этих проводах. Но если делать не спеша, то всё возможно. ;)





Потом не забыть подключить как надо все остальные контакты микросхем.

Пишем программу.

Для управления первым регистром используем порты 7, 8 и 9, для второго — 4, 5 и 6. Готовим порты управления:

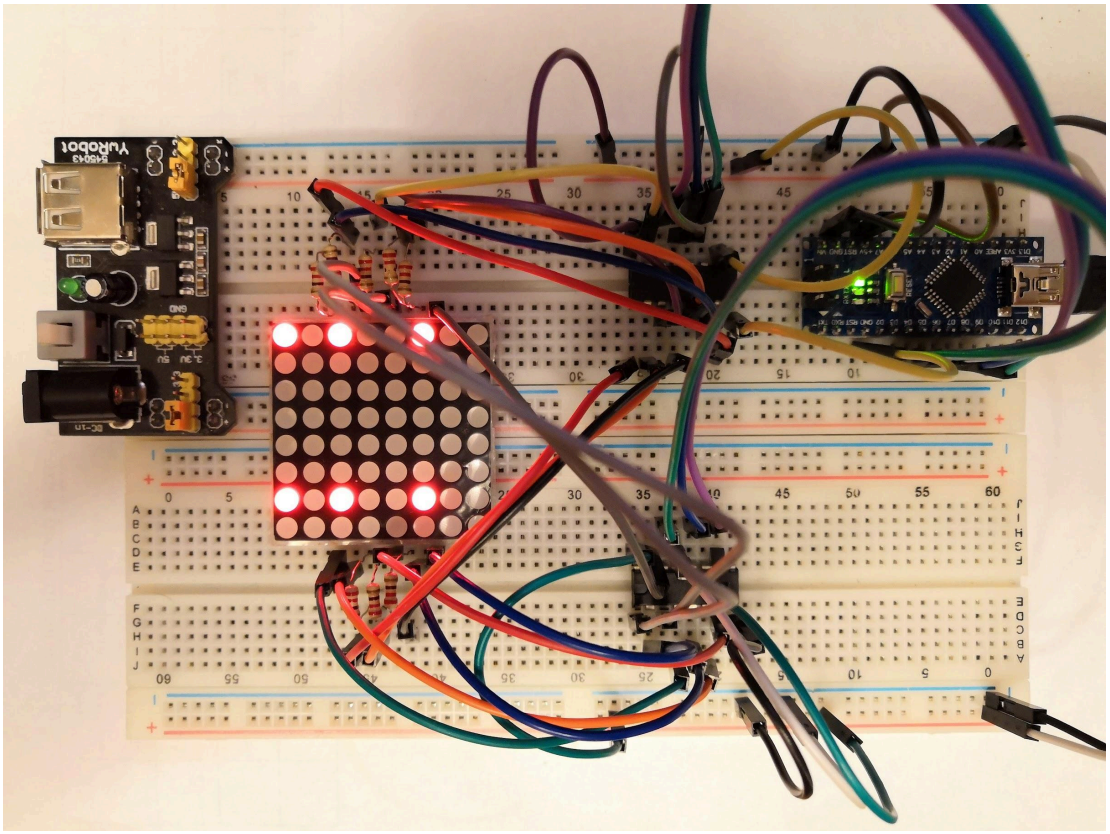
```
#define data1_pin 7 //порт данных первого регистра
#define latch1_pin 8 //порт latch первого регистра
#define clock1_pin 9 //порт clock первого регистра
#define data2_pin 4 //порт данных второго регистра
#define latch2_pin 5 //порт latch второго регистра
#define clock2_pin 6 //порт clock второго регистра
void setup() {
//всё готовим к работе
  pinMode(data1_pin, OUTPUT);
  pinMode(clock1_pin, OUTPUT);
  pinMode(latch1_pin, OUTPUT);
  pinMode(data2_pin, OUTPUT);
  pinMode(clock2_pin, OUTPUT);
  pinMode(latch2_pin, OUTPUT);
// на latch посылаем LOW
  digitalWrite(latch1_pin, LOW);
  digitalWrite(latch2_pin, LOW);
}
```

Выводить будем двумя функциями shiftOut. Но тут вспоминаем, что на столбцы нужно подавать не HIGH, а LOW!!! То есть, для первого регистра, который подключен к С1-С8 нужно все нолики поменять на единицы, а единицы на нолики (сделать инверсию или отрицание). Например, если мы хотим включить 1-й, 3-й и 7-й столбец мы могли бы написать: B10100010. Но нам нужно инвертировать и написать B01011101. Это удобно сделать автоматически оператором побитового отрицания "~". Он сам "перевернёт" все биты, а мы не запутаемся. То есть, мы будем писать: ~B10100010 и получим то что нужно.

Сделаем для примера программу, которая зажжёт первый, третий и седьмой светодиод в первой и седьмой строках:

```
void loop() {
shiftOut(data1_pin, clock1_pin, LSBFIRST, ~B10100100);//для столбцов
                                // включаем первый, третий и шестой
                                // бит, но делаем отрицание!
  shiftOut(data2_pin, clock2_pin, LSBFIRST, B10000010);//для строк
                                   //включаем первый и шестой бит
  digitalWrite(latch1_pin, HIGH);//как всегда latch для вывода
  digitalWrite(latch2_pin, HIGH);
  digitalWrite(latch1_pin, LOW);
  digitalWrite(latch2_pin, LOW);
}
```

Работает:



Это не единственный вариант подключения.. Мы можем сэкономить три порта на Ардуино, если подключим сдвиговые регистры друг за другом через контакт Q7'. Тогда три управляющих контакта (data, latch, clock) мы делаем общими для обоих микросхем, они будут работать синхронно. Мы будем "задвигать" в регистры 16 бит подряд через одну (первую) микросхему. Наш "паровозик" пройдёт сквозь оба регистра и заполнит их. Однако, тогда нужно будет "склеить" оба наших байта в одно 16-разрядное число и передавать его через shiftOut в два приёма, используя оператор сдвига. И вместо записи типа B01001110 нужно будет использовать 0b01001110, чтобы не было ограничения на 8 бит.

Мы научились управлять матрицей. Но тут есть одна проблема: мы не можем выбрать в одной строке один столбец, а в другой строке другой столбец! Одни и те же точки будут светиться во всех включенных строках! Чтобы обойти это ограничение нам придётся включать строки по очереди, но очень быстро, чтобы глаз не замечал переключений и зрителю казалось бы, что всё светится одновременно.

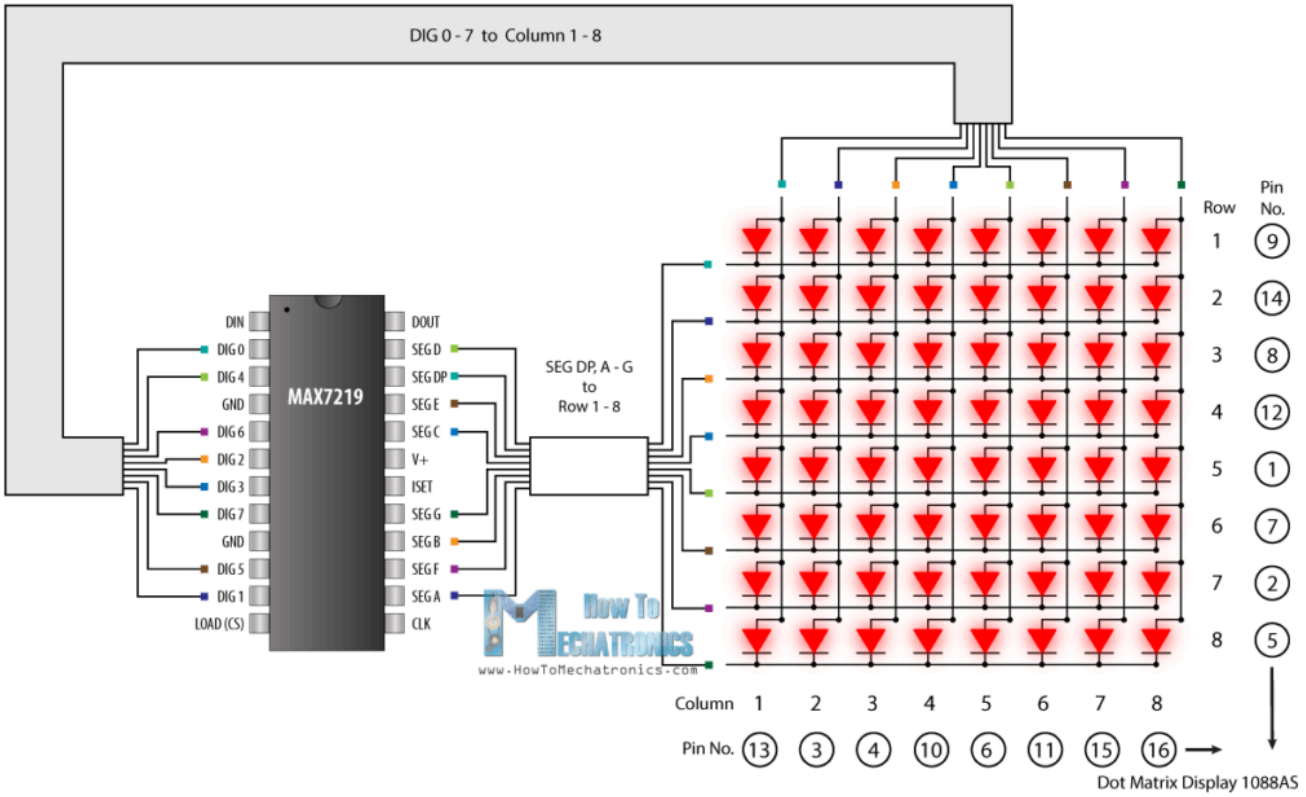
Это будем делать уже в задании.

Обрати внимание, что мы рассмотрели версию с "положительными" строками и "отрицательными" столбцами. Если нужно использовать вариант, в котором наоборот "плюсы" на столбцах и "минусы" на строках ("общий катод"), то по идее всё должно работать так же, но побитовое отрицание (~) нужно ставить не в первом shiftOut, а во втором.

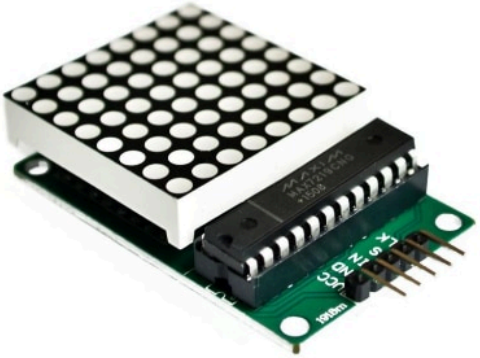
Сборки светодиодных матриц на микросхеме MAX7219

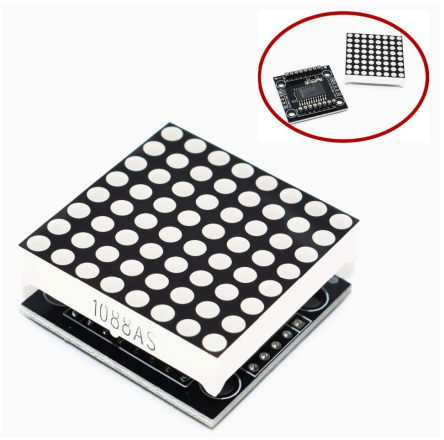
Драйвер MAX7219 и сборки на его основе.

Ручное подключение и управление даже одной светодиодной матрицей или 7-сегментным индикатором через сдвиговые регистры не очень удобно. А их часто нужно использовать по несколько штук. Управление облегчают специализированные микросхемы-драйверы. Один из популярных драйверов — MAX7219. Впрочем, в этой микросхеме тоже содержится сдвиговый регистр, 16-битный. Внизу на схеме показана схема подключения.

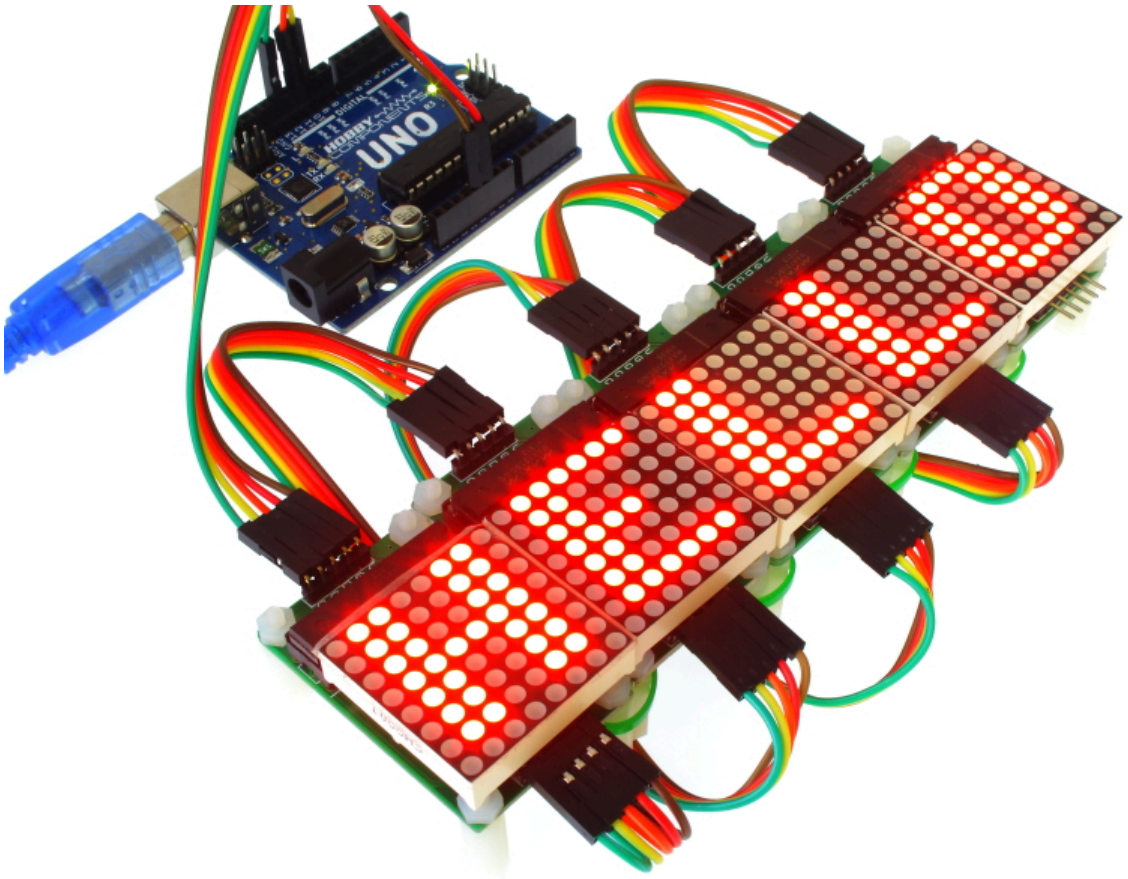


Но нам не придётся подключать его самостоятельно. Продаются готовые модули на основе этого драйвера. Для управления таким модулем нужно только три цифровых порта:

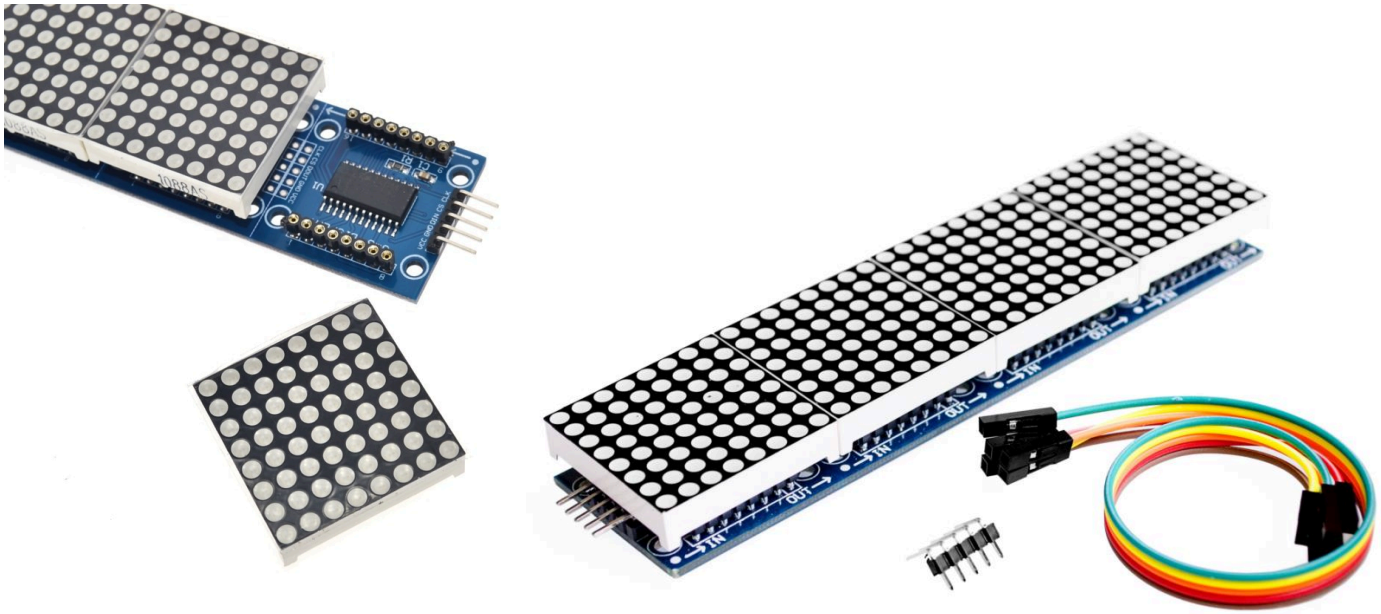




Эти модули можно собирать в линейные и прямоугольные сборки. Правда, тот вариант что на левой картинке, годится только для линейной сборки, и для соединения придётся использовать кучу проводов (смотри ниже):

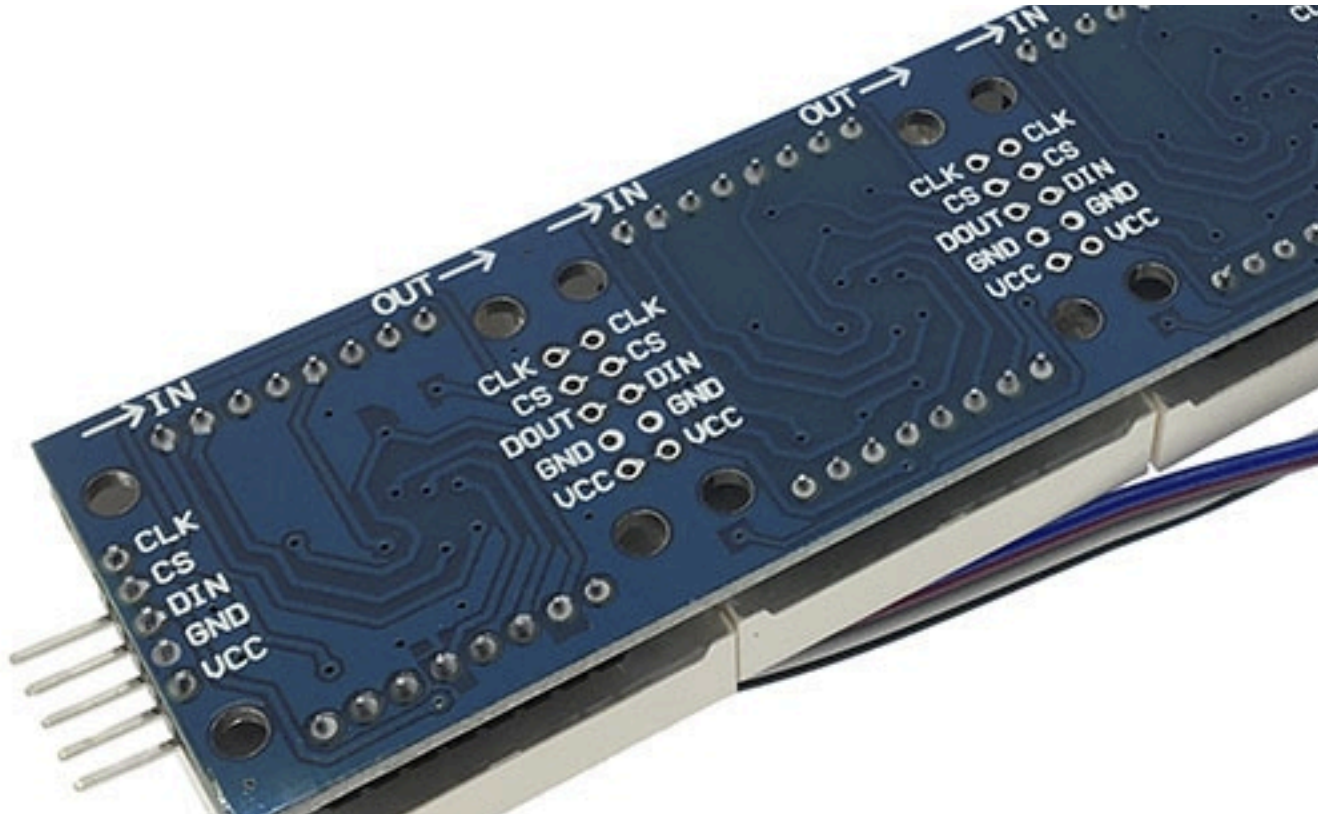


Второй вариант удобнее. Эти модули можно соединять в любом количестве по горизонтали и вертикали. Это можно делать самостоятельно или купить готовую сборку. Пока я видел только модули для дисплеев с общим катодом (строки минус, столбцы плюс).



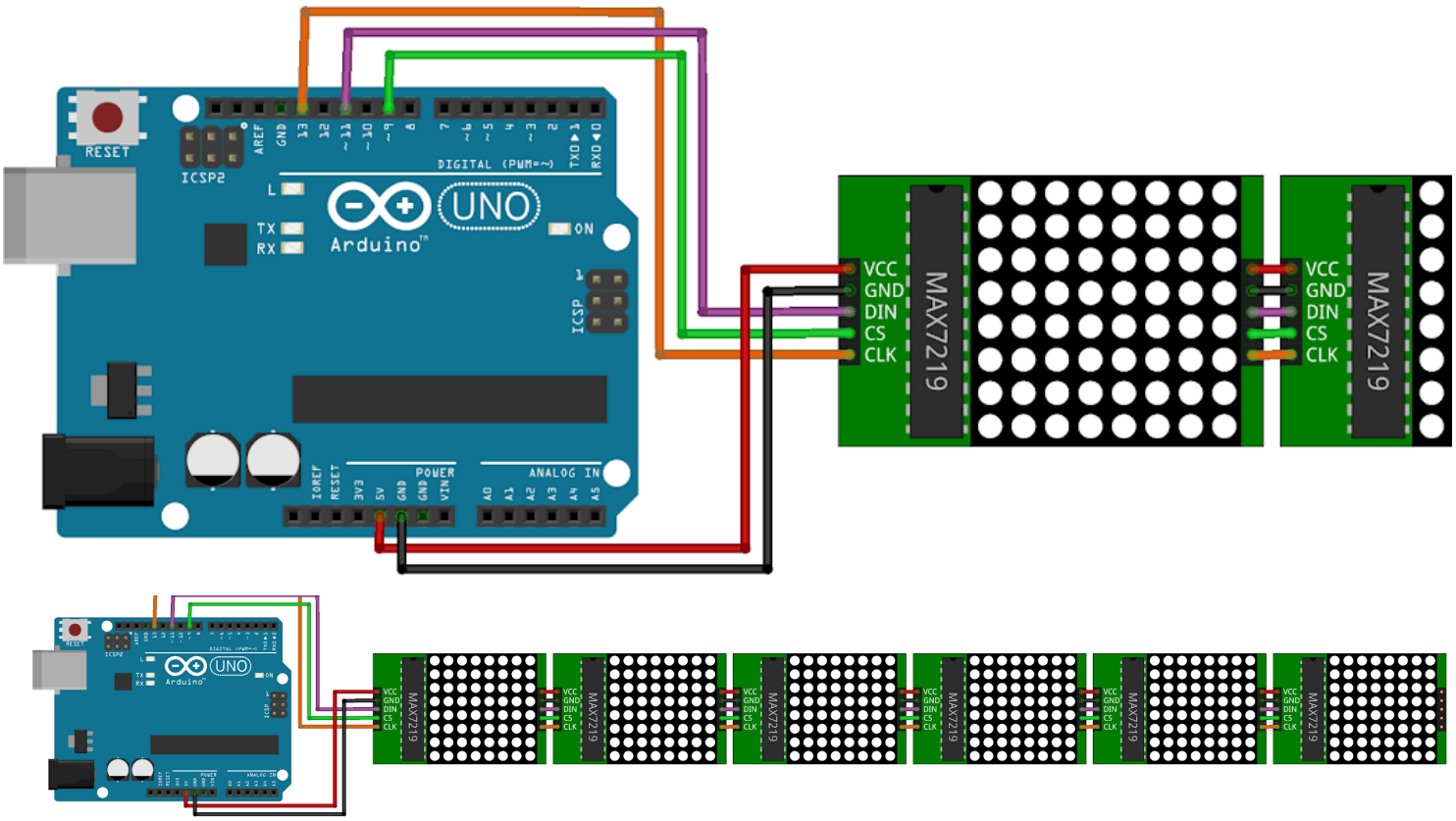


Модули соединяются перемычками или пайкой, а при желании и проводами:

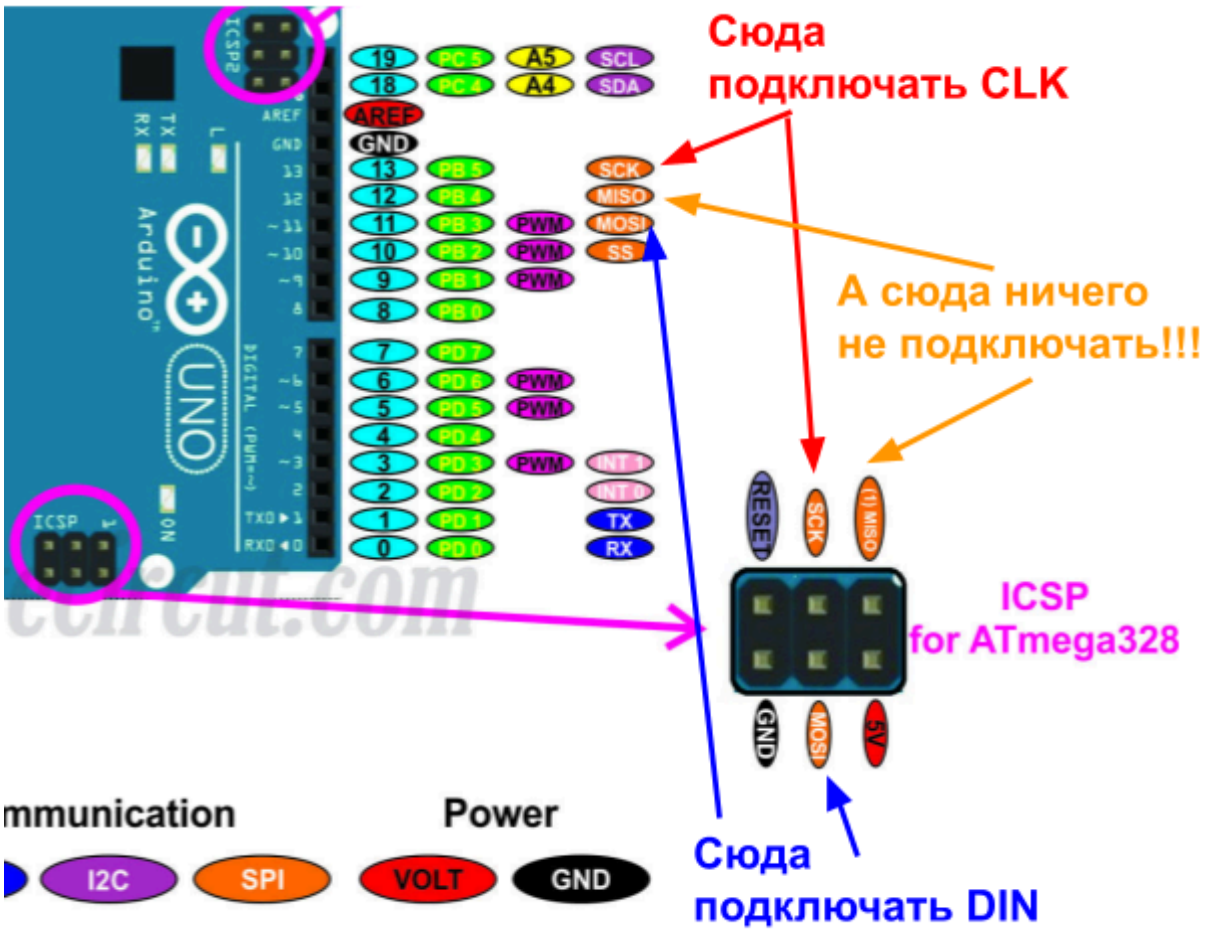


Подключение к Ардуино.

К Ардуино подключается только первый модуль, и связь идёт по всем модулям по цепочке (смотри ниже):



Подключать первый модуль можно, в принципе, к любым портам Ардуино, но в некоторых библиотеках предполагается, что вы будете использовать два пина стандартного интерфейса SPI. Это 11-й пин для входа данных DIN и 13-й для тактового сигнала CLK. Эти пины выведены также на отдельную контактную группу, что хорошо видно на [подробной схеме Ардуино](#).



Третий управляющий контакт — CS — подключай к любому порту, например к 9-му. Как показано на схеме, порт 12, он же "MISO" использовать нельзя. Некоторые авторы также советуют не использовать 10-й.

Питание и землю можно подключать к Ардуино как обычно, но если ты используешь не один-два модуля, а большую сборку, то нужно предусмотреть отдельное питание на 5 вольт. Иначе Arduino может не выдержать.

Библиотеки.

Для работы с драйвером MAX7219 есть множество библиотек. Посмотрим для примера три из них — что попроще в использовании.

1. LEDMatrixDriver. Библиотека достаточно проста. Работает с линейными сборками второго типа (из описанных выше). В программе задаётся количество матриц. Буквы и любые другие элементы можно вставлять в программу самому в виде двоичных или шестнадцатеричных кодов. Эти коды можно получать в онлайн-редакторах. Например, https://gurgleapps.com/tools/matrix или http://xantorohara.github.io/led-matrix-editor. Вот пример вывода бегущего текста:

```
#include <LEDMatrixDriver.hpp>
// Other pins are Arduino specific SPI pins (MOSI=DIN, SCK=CLK)
const uint8_t LEDMATRIX_CS_PIN = 10;//пин для CS
const int LEDMATRIX_SEGMENTS = 4; //число матриц в ряду
const int LEDMATRIX_WIDTH    = LEDMATRIX_SEGMENTS * 8;

LEDMatrixDriver lmd(LEDMATRIX_SEGMENTS, LEDMATRIX_CS_PIN);

// Пример текста. Коды знаков заданы ниже.
char text[] = "** LED MATRIX DEMO! ** (1234567890) ++
\"ABCDEFGHIJKLMNOPQRSTUVWXYZ\" ++ <$%/=?'.@,> --";

const int ANIM_DELAY = 30; //задержка при сдвиге (мс)

void setup() {
  lmd.setEnabled(true);//включаем дисплей
  lmd.setIntensity(2);// Задаём яркость от 0 до 10
}

int x=0, y=0;   // Координаты начала
//Ниже определения символов в виде двумерного массива. Автор просто
//взял часть кодовой таблицы ASCII от кода 32 до 90 и поэтому выбор
//нужного символа из массива происходит по его коду ASCII.
byte font[95][8] = { {0,0,0,0,0,0,0,0}, // SPACE
          {0x10,0x18,0x18,0x18,0x18,0x00,0x18,0x18}, // EXCL
          {0x28,0x28,0x08,0x00,0x00,0x00,0x00,0x00}, // QUOT
```

```
{0x00,0x0a,0x7f,0x14,0x28,0xfe,0x50,0x00}, // #
{0x10,0x38,0x54,0x70,0x1c,0x54,0x38,0x10}, // $
{0x00,0x60,0x66,0x08,0x10,0x66,0x06,0x00}, // %
{0,0,0,0,0,0,0,0}, // &
{0x00,0x10,0x18,0x18,0x08,0x00,0x00,0x00}, // '
{0x02,0x04,0x08,0x08,0x08,0x08,0x08,0x04}, // (
{0x40,0x20,0x10,0x10,0x10,0x10,0x10,0x20}, // )
{0x00,0x10,0x54,0x38,0x10,0x38,0x54,0x10}, // *
{0x00,0x08,0x08,0x08,0x7f,0x08,0x08,0x08}, // +
{0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x08}, // COMMA
{0x00,0x00,0x00,0x00,0x7e,0x00,0x00,0x00}, // -
{0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x06}, // DOT
{0x00,0x04,0x04,0x08,0x10,0x20,0x40,0x40}, // /
{0x00,0x38,0x44,0x4c,0x54,0x64,0x44,0x38}, // 0
{0x04,0x0c,0x14,0x24,0x04,0x04,0x04,0x04}, // 1
{0x00,0x30,0x48,0x04,0x04,0x38,0x40,0x7c}, // 2
{0x00,0x38,0x04,0x04,0x18,0x04,0x44,0x38}, // 3
{0x00,0x04,0x0c,0x14,0x24,0x7e,0x04,0x04}, // 4
{0x00,0x7c,0x40,0x40,0x78,0x04,0x04,0x38}, // 5
{0x00,0x38,0x40,0x40,0x78,0x44,0x44,0x38}, // 6
{0x00,0x7c,0x04,0x04,0x08,0x08,0x10,0x10}, // 7
{0x00,0x3c,0x44,0x44,0x38,0x44,0x44,0x78}, // 8
{0x00,0x38,0x44,0x44,0x3c,0x04,0x04,0x78}, // 9
{0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x00}, // :
{0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x08}, // ;
{0x00,0x10,0x20,0x40,0x80,0x40,0x20,0x10}, // <
{0x00,0x00,0x7e,0x00,0x00,0xfc,0x00,0x00}, // =
{0x00,0x08,0x04,0x02,0x01,0x02,0x04,0x08}, // >
{0x00,0x38,0x44,0x04,0x08,0x10,0x00,0x10}, // ?
{0x00,0x30,0x48,0xba,0xba,0x84,0x78,0x00}, // @
{0x00,0x1c,0x22,0x42,0x42,0x7e,0x42,0x42}, // A
{0x00,0x78,0x44,0x44,0x78,0x44,0x44,0x7c}, // B
{0x00,0x3c,0x44,0x40,0x40,0x40,0x44,0x7c}, // C
{0x00,0x7c,0x42,0x42,0x42,0x42,0x44,0x78}, // D
{0x00,0x78,0x40,0x40,0x70,0x40,0x40,0x7c}, // E
{0x00,0x7c,0x40,0x40,0x78,0x40,0x40,0x40}, // F
{0x00,0x3c,0x40,0x40,0x5c,0x44,0x44,0x78}, // G
{0x00,0x42,0x42,0x42,0x7e,0x42,0x42,0x42}, // H
{0x00,0x7c,0x10,0x10,0x10,0x10,0x10,0x7e}, // I
{0x00,0x7e,0x02,0x02,0x02,0x02,0x04,0x38}, // J
{0x00,0x44,0x48,0x50,0x60,0x50,0x48,0x44}, // K
{0x00,0x40,0x40,0x40,0x40,0x40,0x40,0x7c}, // L
{0x00,0x82,0xc6,0xaa,0x92,0x82,0x82,0x82}, // M
{0x00,0x42,0x42,0x62,0x52,0x4a,0x46,0x42}, // N
{0x00,0x3c,0x42,0x42,0x42,0x42,0x44,0x38}, // O
```

```
            {0x00,0x78,0x44,0x44,0x48,0x70,0x40,0x40}, // P
            {0x00,0x3c,0x42,0x42,0x52,0x4a,0x44,0x3a}, // Q
            {0x00,0x78,0x44,0x44,0x78,0x50,0x48,0x44}, // R
            {0x00,0x38,0x40,0x40,0x38,0x04,0x04,0x78}, // S
            {0x00,0x7e,0x90,0x10,0x10,0x10,0x10,0x10}, // T
            {0x00,0x42,0x42,0x42,0x42,0x42,0x42,0x3e}, // U
            {0x00,0x42,0x42,0x42,0x42,0x44,0x28,0x10}, // V
            {0x80,0x82,0x82,0x92,0x92,0x92,0x94,0x78}, // W
            {0x00,0x42,0x42,0x24,0x18,0x24,0x42,0x42}, // X
            {0x00,0x44,0x44,0x28,0x10,0x10,0x10,0x10}, // Y
            {0x00,0x7c,0x04,0x08,0x7c,0x20,0x40,0xfe}, // Z
                        };    // {}, //
void loop()
{
      // Рисуем текст на текущей позиции
      int len = strlen(text);
      drawString(text, len, x, 0);
// Если вы не понимаете, почему мы не используем lmd.clear() в
//каждом цикле: потому что шрифт имеет прозрачный (тёмный) фон...

      // Выводим изображение
      lmd.display();
      // Ждём, чтобы человек смог увидеть
      delay(ANIM_DELAY);
      // переходи на новую позицию
      if( --x < len * -8 ) {
            x = LEDMATRIX_WIDTH;
      }
}

//Эта функция рисует текстовую строку заданной длины с заданной
//позиции.
void drawString(char* text, int len, int x, int y )
{
  for( int idx = 0; idx < len; idx ++ )
  {
    int c = text[idx] - 32;
    // Остановиться, если буква за пределами видимой области
    if( x + idx * 8  > LEDMATRIX_WIDTH )
      return;
    // рисуем, если буква видима
    if( 8 + x + idx * 8 > 0 )
      drawSprite( font[c], x + idx * 8, y, 8, 8 );
  }
}
```

```
//функция рисует "спрайт" (заданную фигуру) на заданной позиции с
//заданной шириной и высотой.
void drawSprite( byte* sprite, int x, int y, int width, int height )
{
  // маска испольуется чтобы получить бит столбца из строки спрайта
  byte mask = B10000000;

  for( int iy = 0; iy < height; iy++ )
  {
    for( int ix = 0; ix < width; ix++ )
    {
      lmd.setPixel(x + ix, y + iy, (bool)(sprite[iy] & mask ));
      // сдвигаем маску на один пиксель вправо
      mask = mask >> 1;
    }
    // сбрасываем маску
    mask = B10000000;
  }
}
```

В библиотеке, как всегда, есть и другие примеры.

2. Библиотека Max72xxPanel. Хороша тем, что позволяет произвольно задать число матриц по горизонтали и вертикали, а также место и ориентацию каждой матрицы в этой сборке. Это очень полезная фишка, поскольку часто бывает, что изображение выводится не так, как нам нужно. Эта библиотека работает в паре с универсальной библиотекой Adafruit_GFX, которую мы уже использовали. Поскольку в библиотеке Adafruit GFX есть и готовые шрифты и функции для рисования графики, то возможности этой связки библиотек практически не ограничены. Про русификацию тоже уже было написано выше. Нужна исправленная библиотека, в которой заменён один файлик: glcdfont.c, и специальная функция перекодировки для обработки русских букв.

Ниже пример бегущего текста с использованием русификации.

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Max72xxPanel.h>

int pinCS = 9; // Attach CS to this pin, DIN to MOSI and CLK to SCK
int numberOfHorizontalDisplays = 4;//число дисплеев по горизонтали
int numberOfVerticalDisplays = 1;//число дисплеев по вертикали
```

```
Max72xxPanel matrix = Max72xxPanel(pinCS,
numberOfHorizontalDisplays, numberOfVerticalDisplays);//объявление
                                                    // устройства
String tape;
int wait = 100; // In milliseconds
int spacer = 1;//ширина пробела
int width = 5 + spacer; // ширина шрифта с пробелом

void setup() {
tape = utf8rus("Arduino-тест");//Чтобы русский текст показывался
                //правильно, "оборачиваем" его в функцию utf8rus()
  matrix.setIntensity(3); // Яркость от 0 до 15
//Настройка позиций матриц в сборке
  matrix.setPosition(0, 3, 0); //Первая матрица имеет координаты 3,0
  matrix.setPosition(1, 2, 0); //Вторая матрица имеет координаты 2,0
  matrix.setPosition(2, 1, 0); //Третья матрица имеет координаты 1,0
  matrix.setPosition(3, 0, 0); //4-я матрица имеет координаты 0,0
// Настройка ориентации. Указано число поворотов на 90 градусов
  matrix.setRotation(0, 3);//Все матрицы имеют ориентацию 3.
  matrix.setRotation(1, 3);
  matrix.setRotation(2, 3);
  matrix.setRotation(3, 3);
}

void loop() {
  for ( int i = 0 ; i < width * tape.length() + matrix.width() - 1 -
spacer; i++ ) {
    matrix.fillScreen(LOW);

    int letter = i / width;
    int x = (matrix.width() - 1) - i % width;
    int y = (matrix.height() - 8) / 2; //выравниваем текст по
                                       // вертикали
    while ( x + width - spacer >= 0 && letter >= 0 ) {
      if ( letter < tape.length() ) {
        matrix.drawChar(x, y, tape[letter], HIGH, LOW, 1);
      }
      letter--;
      x -= width;
    }

    matrix.write(); // Отправляем картинку на дисплей
    delay(wait);
//ниже закомментированный пример рисования линии
    //matrix.drawLine(1,1,18,7,1);
```

```
    //matrix.write(); // Send bitmap to display
    //delay(1000);
  }
}
// та самая функция для перекодировки русского языка
// через COM порт с клавиатуры и так русский работает
String utf8rus(String source)
{
  int i, k;
  String target;
  unsigned char n;
  char m[2] = { '0', '\0' };

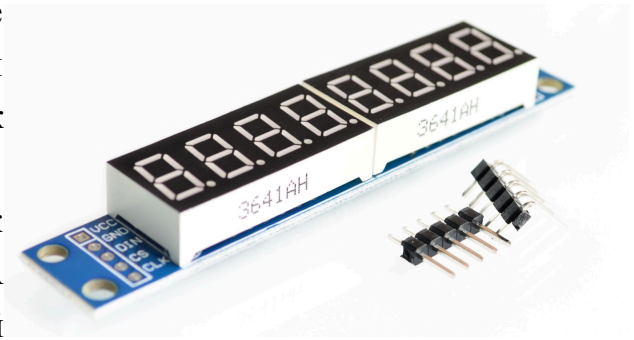
  k = source.length(); i = 0;
  while (i < k) {
    n = source[i]; i++;
    if (n >= 0xC0) {
      switch (n) {
        case 0xD0: {
            n = source[i]; i++;
            if (n == 0x81) {
              n = 0xA8;
              break;
            }
            if (n >= 0x90 && n <= 0xBF) n = n + 0x2F;
            break;
          }
        case 0xD1: {
            n = source[i]; i++;
            if (n == 0x91) {
              n = 0xB7;
              break;
            }
            if (n >= 0x80 && n <= 0x8F) n = n + 0x6F;
            break;
          }
      }
    }
    m[0] = n; target = target + String(m);
  }
  return target;
}
```

Вот тут неплохая статья о работе с этой библиотекой и практическими заданиями: http://robotclass.ru/tutorials/arduino-matrix-8x8-max7219/.

3. Две библиотеки, которые предлагается использовать в комплекте: MD_Parola и MD_MAX72XX. Возможностей очень много, есть разные текстовые эффекты, деление панели на зоны... Но в использовании эти библиотеки посложнее. Вот небольшая статья со сравнением библиотек: http://zlitos.com/publ/ehlektronika/arduino_i_t_p/arduino_i_svetodiodnye_matricy_8kh8_max7219/12-1-0-35. И вот ещё — поподробнее: http://кушеев.рф/archives/1133.

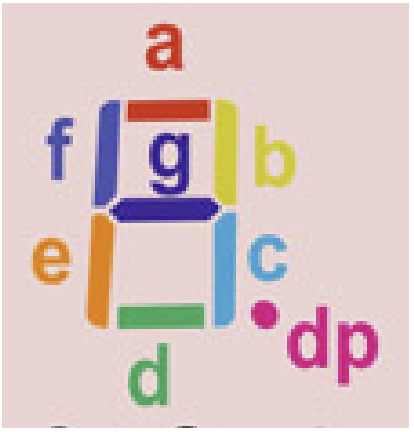
Кто захочет — разберётся. ;)

Сборки 7-сегментных индикаторов на MAX7219



Микросхема MAX7219 может также управлять восемью 7-сегментными индикаторами. Модуль может выглядеть как на картинке справа.

Подключаются они точно также как модули светодиодных матриц. А управляются ещё проще. Для этих модулей есть специальные библиотеки, но некоторые библиотеки могут работать как с матрицами, так и с 7-сегментными индикаторами.



Сначала вспомним как обозначаются сегменты.

Это нужно для понимания записи байтов, соответствующих выводу нужных цифр и некоторых букв. Обозначим горящий сегмент как "1", не горящий как "0". Посмотрим как вывести разные символы. Это похоже на таблицу, с которой мы уже работали, но в библиотеках обычно за десятичную точку отвечает старший бит, и ещё полезно сразу перевести в 16-ричную систему.

Коды знаков.

Знак	Сегменты								получившееся двоичное число	Шестнадца-теричный вариант	соответствующее десятичное число
	dp	a	b	c	d	e	f	g			
0	0	1	1	1	1	1	1	0	B01111110	0x7e	**126**
1	0	0	1	1	0	0	0	0	B00110000	0x30	**48**
2	0	1	1	0	1	1	0	1	B01101101	0x6d	**109**
3	0	1	1	1	1	0	0	1	B01111001	0x79	**121**

4	0	0	1	1	0	0	1	1	B00110011	0x33	**51**
5	0	1	0	1	1	0	1	1	B01011011	0x5b	**91**
6	0	1	0	1	1	1	1	1	B01011111	0x5f	**95**
7	0	1	1	1	0	0	0	0	B01110000	0x70	**112**
8	0	1	1	1	1	1	1	1	B01111111	0x7f	**127**
9	0	1	1	1	1	0	1	1	B01111011	0x7b	**123**

Библиотеки принимают любой вид числа — двоичный, шестнадцатеричный, десятичный. Чтобы вывести десятичную точку, нужно, соответственно, прибавить 128, так как это старший разряд.

Аналогично можно вывести некоторые буквы, например, A, L, H, C, E и другие, в том числе некоторые русские.

Библиотеки.

1. Уже упоминавшаяся раньше библиотека LEDMatrixDriver. Пример:

```
#include <LEDMatrixDriver.hpp>
const uint8_t LEDMATRIX_CS_PIN = 9;//пин для CS, остальные как и
                                   // раньше: 11-й пин (MOSI) для входа
                                   // данных DIN и 13-й (SCK) для
                                   // тактового сигнала CLK.

const int NO_OF_DRIVERS = 1; // Количество модулей. Предполагается,
                             //что каждый модуль содержит 8 индикаторов.

LEDMatrixDriver lmd(NO_OF_DRIVERS, LEDMATRIX_CS_PIN);//объявляем

void setup() {
  lmd.setEnabled(true);//включаем
  lmd.setIntensity(2);  // яркость от 0 до 15
  lmd.setScanLimit(7);  // номер последнего индикатора; 7 означает,
                         // что работают все 8 (от 0 до 7)
  lmd.setDecode(0xFF);  // Влючается режим "BCD Type B". Какие ещё
                       // есть режимы — неизвестно, пусть этот будет.
}

void loop() {
  // Показываем время работы в секундах в течение 1 секунды.
  unsigned long now = millis();
  byte digit;
```

```
  do {
    unsigned long rest = millis();
    for (digit = 0; digit < 8; digit++) {//цикл на 8 проходов
      lmd.setDigit(digit, rest % 10, digit == 3);//записывается
     //значение в каждый разряд, оно вычисляется как остаток от
// деления на 10 третий параметр это просто true или false для
// вывода десятичной точки, в примере точка выводится, когда
// digit==3, то есть делается три десятичных знака.
      rest /= 10; //разряд, который уже выведен — отбрасывается
    }
    lmd.display();//вывод картинки
    delay(10);
  }
  while (millis() - now < 1000);

  // Показать "- HELP -" на одну секунду. В библиотеке предусмотрен
//вывод нескольких дополнительных символов: - (чёрточка) это 0xA или
//просто 10, буква E это 0xB или просто 11, H это 0xC или просто 12,
// L это 0xD или просто 13, P это 0xE или просто 14, пробел это 0xF
//или просто 15. В этом примере символы выводятся сложным способом
  lmd.setDigit(7, LEDMatrixDriver::BCD_DASH);
  lmd.setDigit(6, LEDMatrixDriver::BCD_BLANK);
  lmd.setDigit(5, LEDMatrixDriver::BCD_H);
  lmd.setDigit(4, LEDMatrixDriver::BCD_E);
  lmd.setDigit(3, LEDMatrixDriver::BCD_L);
  lmd.setDigit(2, LEDMatrixDriver::BCD_P);
  lmd.setDigit(1, LEDMatrixDriver::BCD_BLANK);
  lmd.setDigit(0, LEDMatrixDriver::BCD_DASH);
  lmd.display();
  delay(1000);
  lmd.clear();//очистить (заполнить всё нулями)
  lmd.setDigit(5, 0xF, false);//а тут простым способом: поставить
                              // пробел на 5-ю позицию
  lmd.display();//показать
  delay(1000);
}
```

2. Библиотека LedControl. Она работает и с матрицами, но особенно удобна для 7-сегментных сборок. Пример:

```
#include "LedControl.h"
LedControl lc=LedControl(11,13,9,1);//объявление устройства с
// указанием портов: 11 для DataIn (DIN), 13 для CLK, 9 для CS
// (LOAD), последняя цифра — число модулей, у нас один.

unsigned long delaytime=250;
```

```
void setup() {
  lc.shutdown(0,false);//разбудить первый модуль
  lc.setIntensity(0,8);// яркость
  lc.clearDisplay(0);// очистить
}
//эта функция выводит слово "Arduino" с 6-й позиции до 0-й. Цифры
// (включая a,b,c,d,e,f, которые служат цифрами в 16-ричной системе)
// выводятся функцией setChar, остальное можно вывести напрямую
// кодами через функцию setRow.
void writeArduinoOn7Segment() {
  lc.setChar(0,6,'a',false);//вывод "цифры" A на 6 позиции, false
                            // значит, что десятичная точка не нужна
  lc.setRow(0,5,0x05); //выводим символ с 16-ричным кодом 5, что
                //соответствует букве "r" (включаются сегменты e и g)
  lc.setChar(0,4,'d',false);
  lc.setRow(0,3,0x1c);
  lc.setRow(0,2,B00010000);
  lc.setRow(0,1,0x15);
  lc.setRow(0,0,0x1D);
  delay(delaytime*10);
  lc.clearDisplay(0);
  delay(delaytime);
}

//Эта функция зачем-то скроллит все цифры по 4 индикаторам
void scrollDigits() {
  for(int i=0;i<13;i++) {
    lc.setDigit(0,3,i,false);
    lc.setDigit(0,2,i+1,false);
    lc.setDigit(0,1,i+2,false);
    lc.setDigit(0,0,i+3,false);
    delay(delaytime);
  }
  lc.clearDisplay(0);
  delay(delaytime);
}
void loop() {
  writeArduinoOn7Segment();
  scrollDigits();
}
```

3. Библиотека DigitLedDisplay. Библиотека маленькая, входит в комплект среды Arduino IDE.

```
#include "DigitLedDisplay.h"
DigitLedDisplay ld = DigitLedDisplay(11, 9, 13);//объявление с
                // указанием пинов: 11 для DIN, 9 для CS, 13 для CLK.
```

```
void setup() {
  ld.setBright(10);//яркость от 0 до 15
  ld.setDigitLimit(8);//число разрядов
}

void loop() {
  ld.printDigit(12345678);//просто выводим числа
  delay(500);
  ld.clear();//очищаем
  ld.printDigit(22222222);
  delay(500);
  ld.clear();
  ld.printDigit(44444444);
  delay(500);
  ld.clear();

  for (int i = 0; i < 100; i++) {//отсчёт от 0 до 99
    ld.printDigit(i);//выводим числа
    ld.printDigit(i, 4);//ещё раз выводим, начиная с разряда №4.
    delay(50);
  }
  for (int i = 0; i <= 10; i++) {
    ld.off();//выключаем дисплей
    delay(150);
    ld.on();//включаем снова (помигали)
    delay(150);
  }

  ld.clear();
  delay(500);

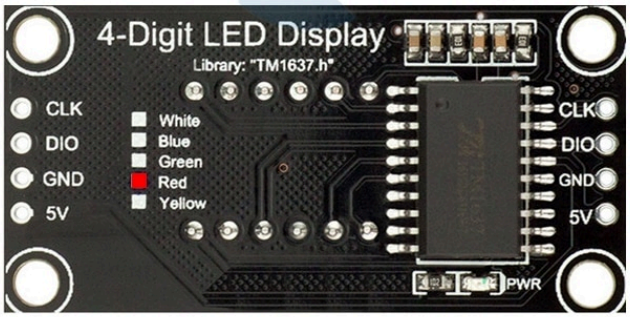
  for (long i = 0; i < 100; i++) {
    ld.printDigit(i);
    delay(25);
  }

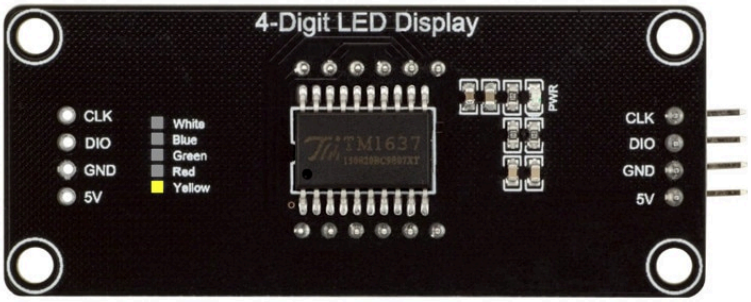
  for (int i = 0; i <= 20; i++) {
    ld.write(5, B01100011);//рисуем символ по коду — это кружочек
    delay(200);
    ld.write(5, B00011101);// это тоже кружочек, но сверху
    delay(200);
  }
  ld.clear();
  delay(500);
}
```

Сборки 7-сегментных индикаторов на микросхеме TM1637









Модули на основе драйвера TM1637 обычно содержат 4 семисегментных разряда. Бывают модули с индикаторами разных размеров. Ещё они отличаются точками. На левой картинке показан дисплей с нормальными десятичными точками, их можно зажигать. на правой картинке десятичные точки хотя и есть, но они не включаются, вместо них есть "двоеточие", которое подходит для отображения времени. Бывают также варианты, где десятичных точек вообще не видно. Это нужно учитывать.

Подключение похоже на MAX7219, только нет контакта CS, так как эти модули используются по одному. Контакты DIO и CLK можно подключать к любым пинам Ардуино.

Библиотеки.

1) TM1637 — простая библиотека. Может выводить целые, дробные, отрицательные числа, отдельную цифру в указанную позицию, а также массив.

```
#include "TM1637.h"
#define CLK 3 //назначаем пины для подключения
#define DIO 2
TM1637 disp(CLK,DIO);
int8_t chislo[] = {2, 14, 10, 0};//объявляем массив с числами, тип
                                 // данных должен быть int8_t
void setup()
{
  disp.set(BRIGHT_TYPICAL);// Яркость от о до 7, можно использовать
       //слова: BRIGHT_TYPICAL = 2,BRIGHT_DARKEST = 0,BRIGHTEST = 7;
```

```
  disp.init(D4056A);//инициализируем дисплей. D4056A это тип
//дисплея. Этот дисплей поддерживает десятичные точки. Его нужно
//выбирать, если мы работаем с дробными числами. Второй тип D4036B
//он выбирается по умолчанию и не поддерживает десятичные точки. Он
//поддерживает двоеточие для показа времени.
  }
void loop()
{
  disp.display(123);//Просто выводим число (прижато вправо)
  delay(1000);
  disp.display(-17);//отрицательное число
  delay(1000);
  disp.display(1.67);//дробное число
  delay(1000);
  disp.display(-91.32);//отрицательное дробное число
  delay(1000);
  disp.display(chislo);//выводим массив, объявленный в начале
  delay(1000);
  disp.display(2,7);//выводим 7 в позицию 2 (позиции нумеруются
                    // слева направо: 0, 1, 2, 3)
  delay(1000);

}
```

Ещё возможные методы:

clearDisplay(); — очищает дисплей

disp.point(true или false); — включает или выключает двоеточие на "часах"

writeByte(любой байт); — выводит на крайний левый индикатор произвольный байт (единички соответствуют светящимся сегментам, нули — несветящимся) по принципу, описанному ранее.

В методе display() в вариантах вывода числа в заданную позицию и вывода массива можно в принципе вывести что угодно. В оригинальной библиотеке предусмотрены только числа, включая 16-ричные a,b,c,d,e,f, а также "минус" и пробел. Желающие добавляют в неё новые символы. Вот исправленный вариант со множеством "букв". Номера этих букв перечислены в таблице на следующей странице (плюс 39 — пробел). Выводим, например, слово "LEGO":

```
int8_t Lego[] = {24, 14, 16, 0};
disp.display(Lego);
```

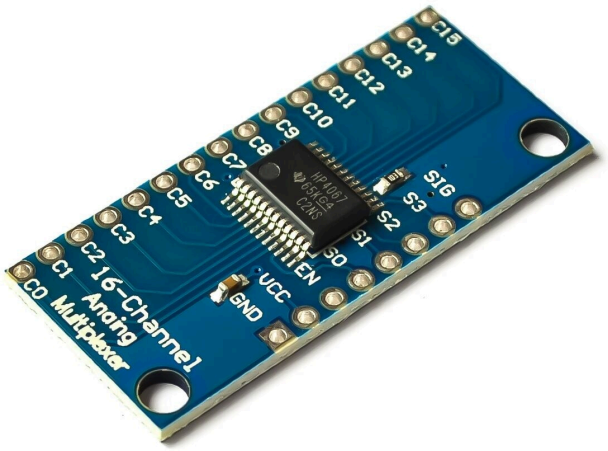
2) Сильно переделанная библиотека от Алекса Гайвера. По ссылке есть подробные пояснения. Библиотека всем хороша, кроме того, что не поддерживает дробные числа и вообще десятичные точки. Число с точкой можно вывести вручную через метод displayByte, добавляя 128 к коду любой цифры.

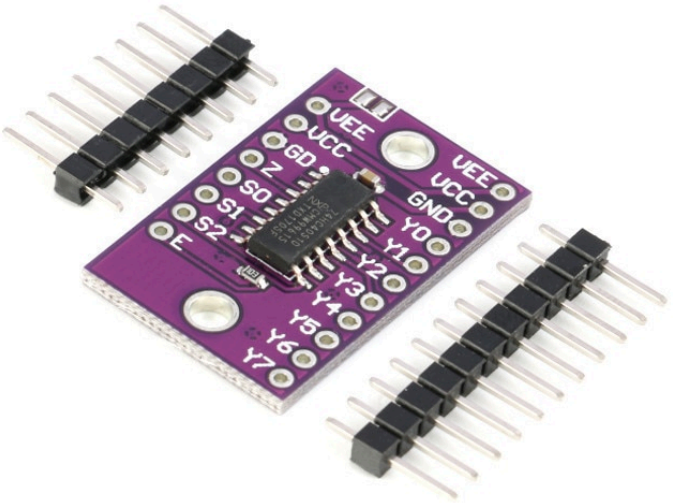
Индикация	Символ	Индекс
	0	0
	1	1
	2	2
	3	3
	4	4
	5	5
	6	6
	7	7
	8	8
	9	9
	A	10
	B	11
	C	12
	D	13
	E	14
	F	15

Индикация	Символ	Индекс
	G	16
	H	17
	h	18
	l	19
	i	20
	J	21
	j	22
	K	23
	L	24
	M	25
	n	26
	*градус	27
	o	28
	P	29
	Q	30
	r	31
	t	32
	U	33
	u	34
	Y	35
		36
		37
		38

Расширители портов (мультиплексоры)

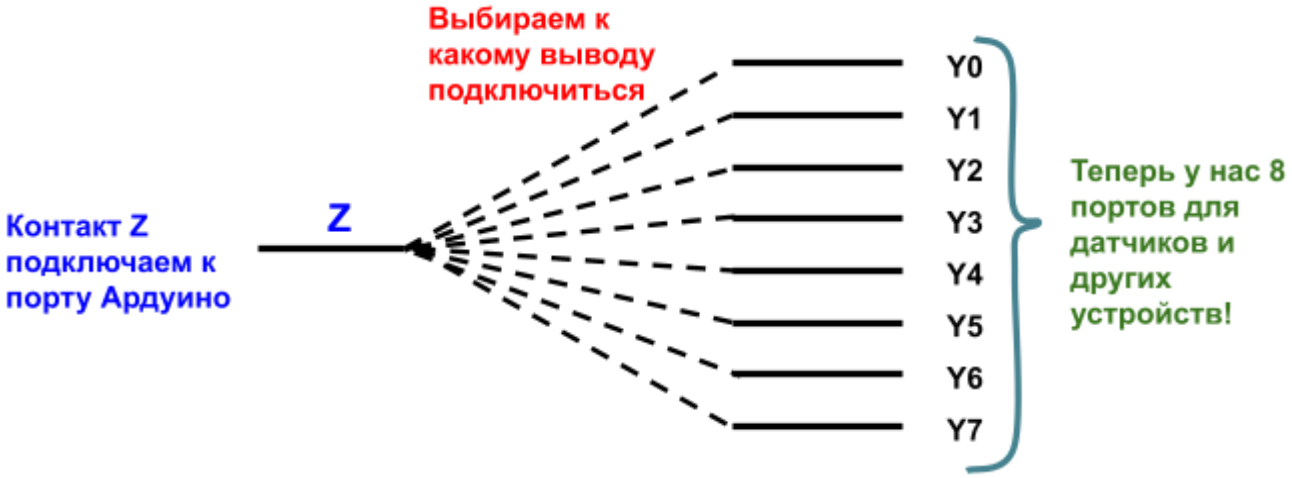
Кроме сдвигового регистра для увеличения количества портов можно использовать модули на основе микросхем-мультиплексоров. Справа пример 8-канального мультиплексора на микросхеме 74HC4051. Слева — 16-канальный на CD74HC4067. Дальше будем





рассматривать восьмиканальный

Мультиплексор работает как переключатель. Он подключается к одному порту Arduino через контакт Z, и в своей программе мы его можем соединить с любым из 8 выводов мультиплексора (Y0-Y7). Вот так:



То есть, вместо одного порта получаем восемь. Но обращаться к ним мы можем только по очереди. Мы должны указать микросхеме, к какому именно выводу мы хотим подключиться. Для такого выбора используется ещё три порта Ардуино. Они подключаются к контактам S0, S1, S2. Через них модулю передаётся в двоичном виде номер его порта, с которым мы хотим соединиться.

S2	S1	S0	
0	0	0	— 0
0	0	1	— 1
0	1	0	— 2
0	1	1	— 3
1	0	0	— 4
1	0	1	— 5
1	1	0	— 6
1	1	1	— 7

Например, мы захотели считать сигнал с датчика, который подключен к выводу Y6. Значит, на S2 и S1 нужно подать HIGH, а

на S0 — LOW. Сделаем маленький скетч. Допустим, мы подключили пины так:
S0 к цифровому пину 3,
S1 к пину 4,
S2 к пину 5,
Z к аналоговому порту A1.

Результат хотим вывести в последовательный порт.

```
void setup() {
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  Serial.begin(9600);
}
void loop() {
//устанавливаем адрес вывода Y6:
digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, HIGH);
Serial.println(analogRead(A1)); //читаем и выводим
}
```

Однако, такой способ не удобен для случаев, когда нужно опросить произвольный порт по его номеру или все порты, например в цикле. Удобнее просто "доставать" нужные биты из номера порта и отправлять их на S0,S1,S2. Для этого есть удобная функция bitRead, она читает нужный бит из заданного числа. Например, нам по-прежнему нужно прочитать порт Y6. Шестёрка это в двоичной форме 110. Биты (двоичные разряды) нумеруются с конца: нулевой (младший) бит здесь =0, первый =1 и второй =1. Вот и читаем их и сразу отправляем по назначению:

```
digitalWrite(3, bitRead(6,0));//читаем нулевой бит из шестёрки
digitalWrite(4, bitRead(6,1));//читаем первый бит из шестёрки
digitalWrite(5, bitRead(6,2));//читаем второй бит из шестёрки
```

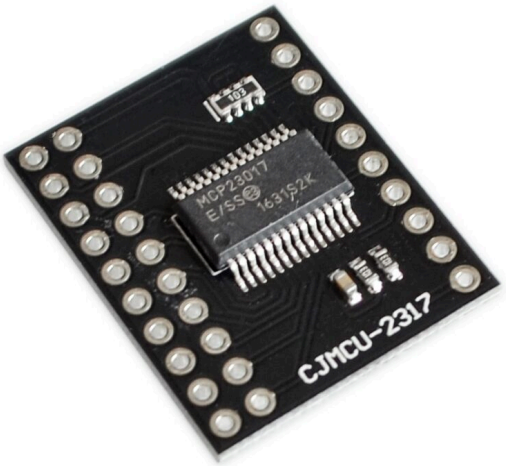
(и сразу всё отправили в S0, S1 и S2).

Естественно, вместо шестёрки может быть переменная, например, счётчик из цикла, тогда мы удобно пройдёмся по всем портам.

Большой плюс мультиплексора в том, что он работает как на ввод, так и на вывод, как с цифровым, так и с аналоговым сигналом. Ему всё равно что там куда передаётся — он просто соединяет порты.

Пояснения к подключению: контакты E и VEE можно подключать к земле. При желании можно использовать E для отключения модуля, подав на него HIGH.

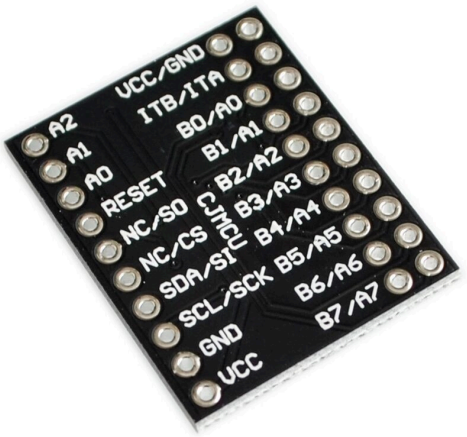
Расширитель портов с интерфейсом I2C на MCP23017



Другой вариант увеличения количества портов — микросхема MCP23017 и модули на её основе. Она подключается по шине I2C и даёт 16 дополнительных цифровых портов. К сожалению, без возможности использовать ШИМ. Зато, если использовать библиотеку от Adafruit, то мы можем работать с этими портами практически как с обычными (digitalRead и digitalWrite).

Подключение:

Выводы I2C: SDA и SCL подключаем к соответствующим выводам Arduino.



GND и VCC - соответственно, земля и питание (5 вольт).

Выходы A0-A7 и B0-B7 это 16 новых портов — подключаем их к чему нужно.

С той же стороны модуля есть ещё пара выводов VCC и GND — их можно подключать к устройствам на выходе (если нужно).

Если мы используем библиотеку Adafruit, то порты имеют сплошную нумерацию: от A0 до A7 это просто 0-7, а от B0 до B7 имеют номера 8-15.

Пример мигалки на порту B3 (он же 11):

```
#include <Wire.h>
#include "Adafruit_MCP23017.h"

Adafruit_MCP23017 mcp;//объявление устройства (объекта) с именем mcp

void setup() {
  mcp.begin();      // начинаем работу

  mcp.pinMode(11, OUTPUT); //обычный pinMode, только для объекта mcp
}

void loop() {
//тут обычная мигалка только все функции применяются к объекту mcp
  mcp.digitalWrite(11, HIGH);
  delay(1000);
  mcp.digitalWrite(11, LOW);
  delay(1000);
}
```

Есть отдельная команда для подтяжки нужного порта к питанию:
mcp.pullUp(3, HIGH); — включить подтяжку на третьем порте,
mcp.pullUp(3, LOW); — выключить подтяжку на третьем порте.

Дополнительные возможности при подключении:

Вывод Reset служит для сброса при подаче на него LOW. На всякий случай рекомендуется подключать его к 5 вольтам.

Выводы A0, A1 и A2 служат для изменения адреса устройства на шине I2C. Это может понадобиться если нужно подключить к Ардуино одновременно более одного такого модуля или если адрес по умолчанию совпадает с адресом какого-то другого подключенного устройства. В стандартном варианте эти выводы можно подключить к земле или никуда не подключать.

Выводы ITA и ITB служат для передачи на Ардуино сигналов прерываний. Их можно подключать к портам, поддерживающим прерывания, а модуль может выдавать туда сигналы прерывания от своих портов (если это настроить). В библиотеке есть пример с обработкой прерываний.

Дополнительные подробности можно прочитать здесь:
http://www.bizkit.ru/2019/02/20/12466/
https://mysku.ru/blog/aliexpress/59294.html
http://ww1.microchip.com/downloads/en/DeviceDoc/20001952C.pdf

Знаковый LCD дисплей (двухцветный)

Это очень распространённый тип дисплеев. LCD означает "жидкокристаллический дисплей". На них рядами расположены позиции для текстовых символов, каждая позиция это матрица 5x8 пикселей, из них под знаки используется 5x7, а нижний ряд зарезервирован для мигающего курсора.



Обычно показываются только текстовые знаки, так как они зашиты в памяти дисплея и их удобно выводить через готовые библиотеки. Хотя, в принципе, на каждой знаковой позиции можно изобразить любой пиксельный рисунок. В стандартном варианте русские буквы не поддерживаются, но продаются и специальные версии с прошитыми русскими буквами.

Чаще всего встречаются дисплеи на 2 строки по 16 символов, их обозначают как "1602", но бывают и 4 строки по 20 символов ("2004") и некоторые другие.

Различаются они также и по цвету символов и фоновой подсветки.

На картинке сверху показаны три дисплея типа "2004" разных цветов.

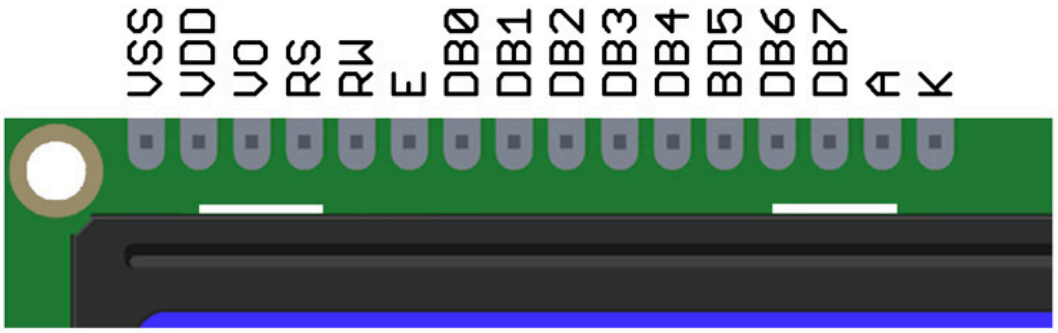
Такие дисплеи не очень удобны для практических проектов, так как они довольно большие и тяжёлые. Но они являются фактическим стандартом в мире Arduino, их часто используют в задачах, примерах. Поэтому нужно уметь с ними работать.

Мы попробуем в деле дисплей 1602 в двух вариантах: в обычном и с дополнительным контроллером I2C. Второй вариант удобнее, но классику надо знать, поэтому начнём с первого.

Подключение без I2C.

На дисплее 16 контактов.

Вот контакты крупным планом и пояснения к ним:



1 (VSS) - Питание электроники (Gnd)

2 (VDD) - Питание электроники (+5В)

3 (VO) - Контраст: чем меньше напряжение — тем ярче буквы (рекомендуется 0,8-1,4 В). Сюда рекомендуется подключить потенциометр на 10 кОм и им регулировать контраст. Но можно просто подключить сюда Gnd — будет максимальный контраст.

4 (RS) - Управляющий контакт. Выбор режима "инструкции" или "данные".

5 (R/W) - Управляющий контакт. Переключение чтение/запись. Обычно сюда подключают просто Gnd для работы в режиме записи.

6 (E) - Enable (используется как строб-сигнал, то есть, заставляет дисплей передать или получить подготовленную информацию, в сдвиговом регистре это ещё называлось clock).

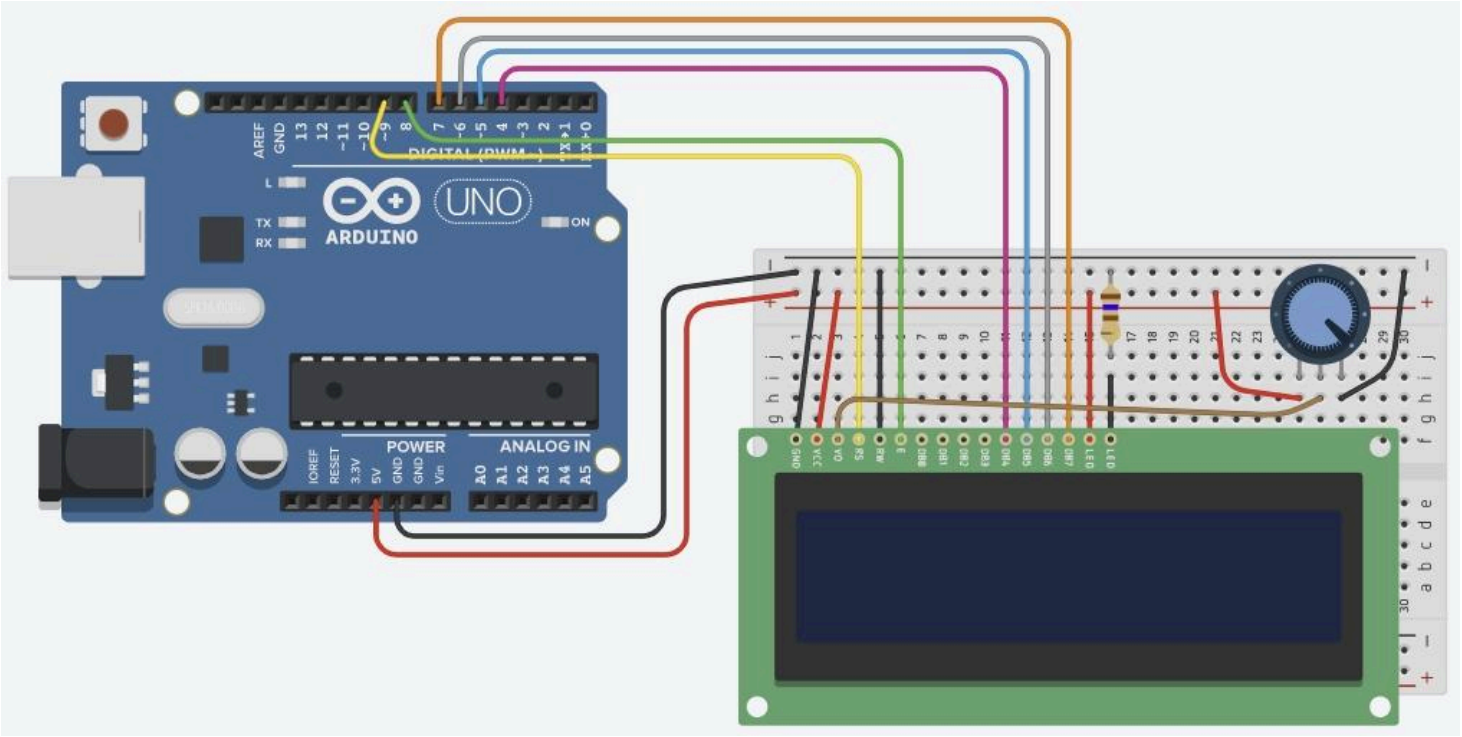
7-10 (DB0-DB3) - Младшие биты 8-битного интерфейса (используются только в 8-битном режиме, мы их не будем использовать).

11-14 (DB4-DB7) - Старшие биты интерфейса (их будем использовать в 4-х битном режиме).

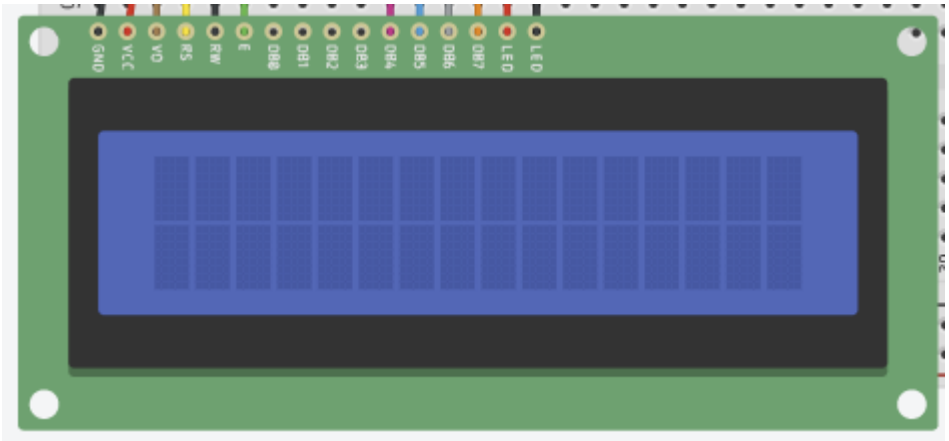
15 (A) - Питание подсветки (+5В). На Tinkercad предлагается поставить сюда резистор на 220 Ом, как для обычного светодиода. Но во многих примерах просто подключают 5 вольт. Видимо, это зависит от разновидности дисплея, в каких-то резистор может быть встроен. Можно подключить сюда потенциометр и настроить яркость по вкусу.

16 (K) - Питание подсветки (Gnd).

Для подключения используются обычно 6 цифровых портов: для контактов R, E, DB4-DB7. Контакт RW подключают к "земле". Ниже пример подключения. Используем цифровые порты с 4-го по 9-й, а также резистор защиты подсветки (для перестраховки).



Для проверки можно включить питание без всякой программы и проверить, засветится ли фон и будут ли видны пиксельные матрицы:



Если не видны — можно подрегулировать контраст потенциометром.

В программе будет всё просто. В Arduino IDE сразу есть в комплекте библиотека для нашего дисплея: LiquidCrystal.h.

```
#include <LiquidCrystal.h>//Подключаем библиотеку
LiquidCrystal lcd(9,8,4,5,6,7);//Создаём объект-дисплей и указываем
  порты в правильном порядке (RS,E,DB4,DB5,DB6,DB7)
void setup(){
  lcd.begin(16, 2);// Задаем размер экрана: символов, строк
  lcd.print("Hello, world!");// Выводим текст. Поскольку мы не
  указали позицию курсора, то он будет выведен от начальной позиции.
 }
void loop(){
  lcd.setCursor(0, 1);// Устанавливаем курсор в начало 2 строки (она
  имеет номер "1", так как нумерация строк идёт с нуля).
```

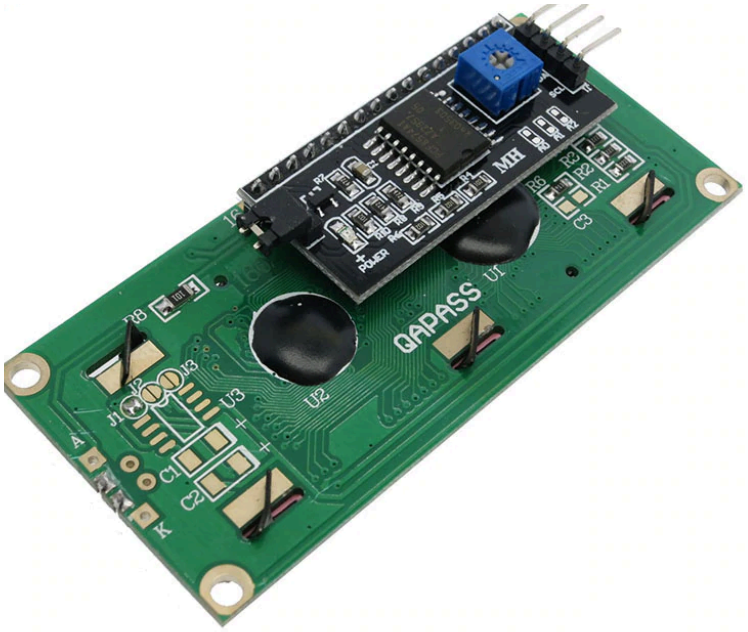
```
  lcd.print("lotsman.ru");// Выводим текст
  lcd.setCursor(11, 1);// Устанавливаем курсор в позицию 11 второй
  строки
  int summa=2+2;// считаем 2+2 и записываем в переменную
  lcd.print(summa);// Выводим значение переменной
  lcd.print(" ");// Выводим пробел
  lcd.print(millis()/1000);// Выводим число секунд с момента старта
}
```

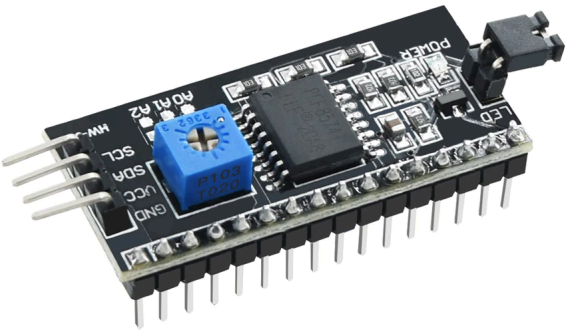
Справочник по основным командам библиотеки:

- **begin(cols,rows);** – Инициализация дисплея с указанием количества столбцов и строк.
- **setCursor(col,row);** – Установка курсора в позицию указанную номером колонки и строки.
- **print(text);** – Вывод текста, символов или цифр на экран дисплея. Синтаксис схож с одноимённой функцией класса Serial.
- **clear();** – Очистка дисплея с установкой курсора в положение 0,0 (Занимает много времени!).
- **home();** – Установка курсора в положение 0,0 (Занимает много времени!).
- **display();** – Быстрое включение дисплея (без изменения данных в памяти).
- **noDisplay();** – Быстрое выключение дисплея (без изменения данных в памяти).
- **blink();** – Включение мигающего курсора (мигает вся позиция 5x8 примерно 1 раз в секунду.
- **noBlink();** – Выключение мигающего курсора.
- **cursor();** – Включение обычного курсора (чёрточка внизу строки).
- **noCursor();** – Выключение обычного курсора.
- **scrollDisplayLeft();** – Прокрутка дисплея влево. Сдвиг координат дисплея на один столбец влево (без изменения памяти). Так можно сделать бегущую строку. Координаты сдвигаются как для имеющейся на дисплее информации, так и для той, которая будет выведена после. Если вызвать функцию 20 раз подряд, то координата вернётся в изначальную точку.
- **scrollDisplayRight();** – То же самое, но вправо.
- **autoscroll();** – Указывает в дальнейшем выравнивать текст по правому краю от позиции курсора. Пояснение: если установить курсор в позицию 10 и вывести текст, то в данной позиции будет находиться курсор. (Координаты дисплея будут сдвинуты влево, как будто Вы вызвали функцию scrollDisplayLeft столько раз, сколько букв в выведенном тексте).
- **noAutoscroll();** – Указывает в дальнейшем выравнивать текст по левому краю от позиции курсора (как обычно).

Подключение по I2C.

Широко распространены также дополнительные модули-контроллеры для подключения таких дисплеев по протоколу I2C. Эти контроллеры делаются на микросхеме pcf8574, это сдвиговый регистр с I2C интерфейсом.



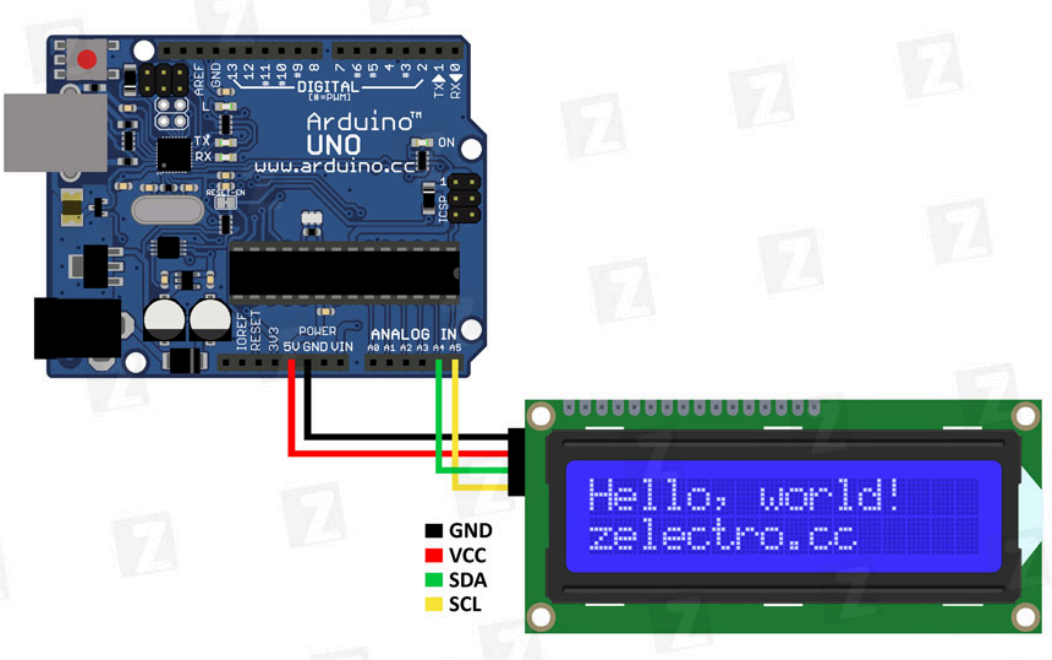


На картинках показан общий вид модуля контроллера (справа) и дисплея с установленным модулем. **Внимание! Потенциометром можно регулировать контраст.** Иногда он выкручен так, что дисплей не работает.

Дисплей с таким контроллером подключать гораздо проще. I2C это всего два управляющих контакта плюс питание и земля.

Управление почти не отличается, только нужно использовать библиотеку LiquidCrystal_I2C. Библиотек с таким названием есть несколько, рекомендую вот эту: https://github.com/johnrickman/LiquidCrystal_I2C. Автор Frank de Brabander. Она есть в менеджере библиотек Arduino IDE, но там не последняя версия. Можно скачать также тут.

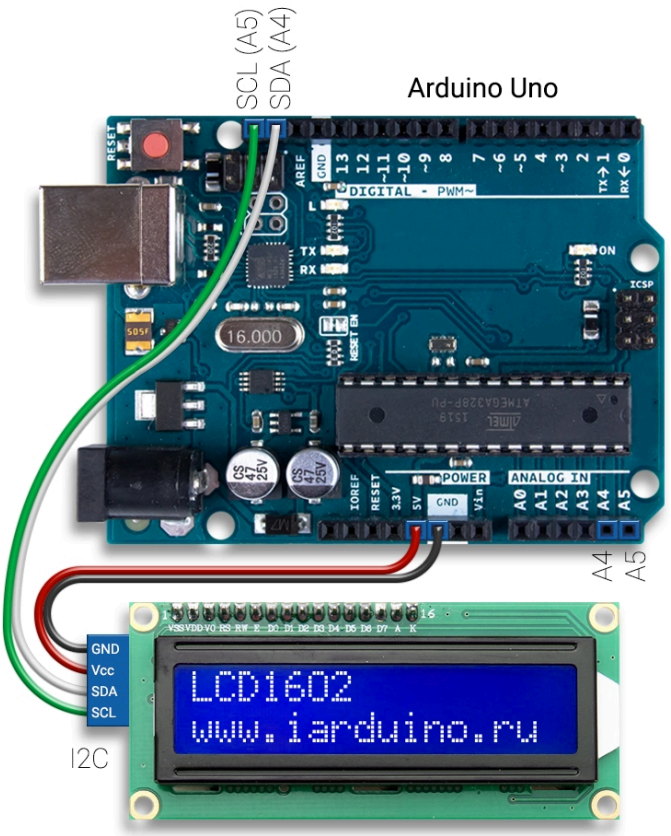
Контроллер подключается к ардуино следующим образом



На платах формата Uno контакты SDA и SCL дублируются также с другой стороны (картинка справа).

Таблица соответствия контактов I2C на разных платах:

Arduino Mega	Arduino Leonardo	Arduino Uno	Контроллер
GND	GND	GND	GND
5V	5V	5V	VCC
20 (SDA)	2	A4	SDA
21(SCL)	3	A5	SCL



Дисплей LCD1602 I2C

Пример скетча для дисплея с библиотекой для I2C

```
#include <Wire.h>  // подключаем библиотеку для I2C.
#include <LiquidCrystal_I2C.h> // подключаем библиотеку для
  контроллера дисплея.
LiquidCrystal_I2C lcd(0x27,16,2); // Создаём объект с указанием
  параметров (16 символов в строке, 2 строки, 0x27 - это адрес
  устройства на шине I2C, скорее всего менять не придётся, если не
  работает - попробуй 0x3f)
void setup(){
  lcd.init(); // включаем дисплей.
  lcd.backlight(); // Включаем подсветку дисплея
  lcd.print("lotsman.ru"); // выводим надпись в первую строку
  lcd.setCursor(8, 1); // устанавливаем курсор на вторую строку (она
    имеет номер 1, потому что первая — нулевая) в позицию 8.
  lcd.print("LCD 1602"); // выводим с этого места надпись.
}
void loop(){
  lcd.setCursor(0, 1); // Устанавливаем курсор на вторую строку и
    нулевой символ.
  lcd.print(millis()/1000); // Выводим на экран количество секунд с
    момента запуска ардуино (миллисекунды делим на 1000)
}
```

Как видишь, отличия только в командах начальной настройки и включения.

Есть несколько функций, которые используются только в библиотеке LiquidCrystal_I2C:

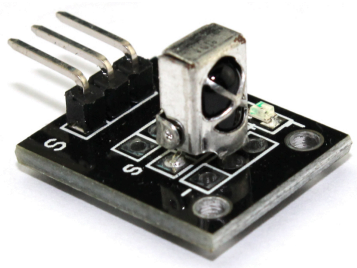
- **init();** – Инициализация дисплея. Должна быть первой командой библиотеки LiquidCrystal_I2C после создания объекта. На самом деле данная функция есть и в библиотеке LiquidCrystal, но в той библиотеке она вызывается автоматически (по умолчанию) при создании объекта.
- **backlight();** – Включение подсветки дисплея.
- **noBacklight();** – Выключение подсветки дисплея.
- **setBacklight(**flag**);** – Управление подсветкой (true - включить / false - выключить), используется вместо функций noBacklight и backlight.

Дополнительная информация:
http://zelectro.cc/LcdI2Cmodule
http://zelectro.cc/LCD1602
В этих статьях в том числе показано как создать собственные символы. Впрочем, в библиотеке есть пример про это.

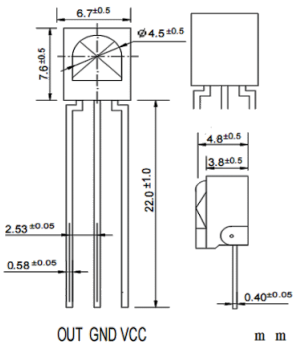
Инфракрасный пульт дистанционного управления



ИК пульты широко используются с телевизорами и прочей бытовой электроникой. Можно их использовать и для управления устройствами на Arduino. Для этого понадобится пульт и ИК приёмник. Пульты могут быть разные, примерный вид на картинке справа. Приёмники могут быть отдельные или припаянные на маленькой платке. Примеры на картинках внизу. Обрати внимание, что у приёмника три контакта: выход (сигнал), питание и земля. Выход подключается к цифровому порту.



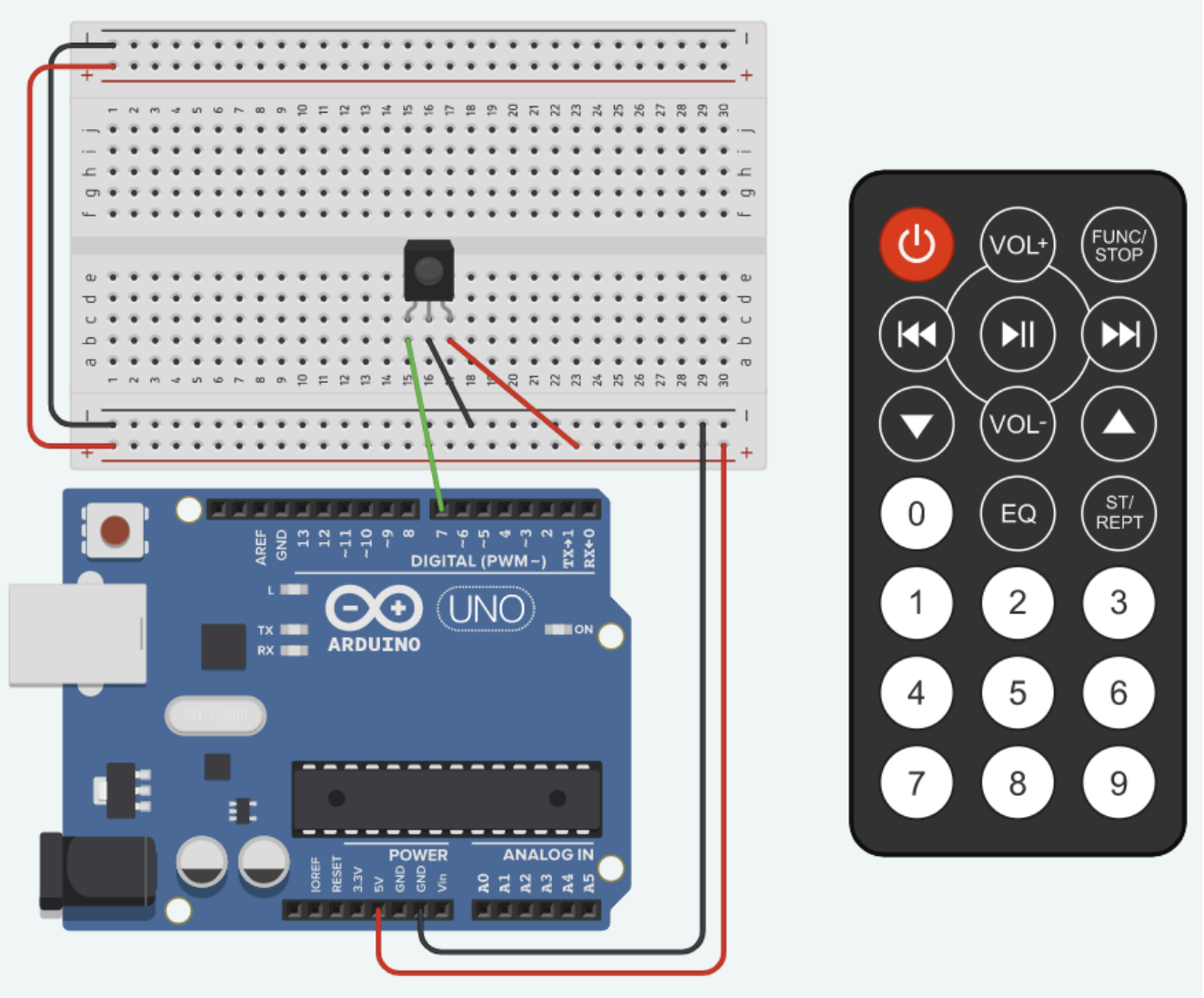




Для работы понадобится библиотека IRremote или какая-то её разновидность. Самая стандартная библиотека находится здесь. Или можно скачать здесь (версия 2.5.0). Какие-то версии этой библиотеки можно найти в менеджере библиотек Arduino IDE, но лучше установить последнюю версию.

Да, пульт вообще можно взять любой — хоть от телевизора или старого видеомагнитофона. Скорее всего, приёмник сможет увидеть его сигнал.

Библиотека может не только принимать сигналы, но и посылать их через инфракрасный светодиод. То есть, ты можешь сделать пульт управления из Ардуино или связать две платы Ардуино между собой. Но практически это довольно сложно и не очень полезно, когда есть WiFi и другие более продвинутые способы связи. Так что ограничимся здесь только приёмом сигналов от пульта. Не забудь только убедиться, что в пульте есть батарейка! :)



Принцип работы этой системы простой. Пульт посылает серии инфракрасных импульсов определённой длины, приёмник их принимает, а Ардуино с помощью библиотеки расшифровывает последовательности импульсов в виде чисел (кодов). Каждая кнопка на пульте даёт свою последовательность импульсов, а значит, свой код.

Для начала нужно узнать коды кнопок для твоего пульта (они на разных пультах разные). Используем простейшую программу, чтобы узнать коды:

```
#include <IRremote.h> // подключаем библиотеку
IRrecv irrecv(6); // создаём объект приёмника и указываем цифровой
  порт, к которому подключен приёмник (у нас это 6)
decode_results results; // создаём объект results специального типа
  (этот тип определён в библиотеке), в котором будет храниться
  полученный код.

void setup() {
irrecv.enableIRIn(); // включаем приёмник
Serial.begin(9600);
}

void loop() {
if (irrecv.decode(&results)) // В этой строке совмещено считывание
  сигнала и проверка, что сигнал пришёл: метод decode даст "ложь",
  если ничего нет.
{
 Serial.println(results.value); //Если сигнал пришёл — выводим его
  код.
 irrecv.resume(); // Закончить обработку сигнала, чтобы быть готовым
  принять следующий.
}
}
```

Теперь нажимаем по очереди разные кнопки на пульте и смотрим в мониторе порта их коды. Это будут довольно большие числа — по 3 байта или больше. Например, что-то типа 16625743, 16609423, 16601263, 16609423. Многие авторы примеров предпочитают выражать их в 16-ричной системе. Тогда нужно написать вывод так:

```
 Serial.println(results.value, HEX);
```

И значения будут похожи на такие: FD48B7, FD6897, FD9867...

Запиши коды тех кнопок, которые планируешь использовать. Лучше сделать таблицу сразу для всех кнопок.

Теперь можно перейти к управлению чем-нибудь. Потренируемся на светодиодах.

Допустим, по нажатию кнопки "1" я хочу включать светодиод на 11-м порту, а по нажатию "2" — на 12 порту. По нажатию "4" и "5" я их буду, соответственно, выключать. А по нажатию красной кнопки выключения я буду выключать сразу оба. Делаем простую программу с if.

```
#include <IRremote.h>
IRrecv irrecv(6);
decode_results results;

void setup(){
irrecv.enableIRIn();
pinMode(11, OUTPUT);
pinMode(12, OUTPUT);
Serial.begin(9600);
}

void loop() {
  if (irrecv.decode(&results)){
    if (results.value == 16582903){//код кнопки 1
      digitalWrite(11, HIGH);
    }
    if (results.value == 16591063){//код кнопки 4
       digitalWrite(11, LOW);
    }
    if (results.value == 16615543){//код кнопки 2
      digitalWrite(12, HIGH);
    }
    if (results.value == 16623703){//код кнопки 5
      digitalWrite(12, LOW);
    }
    if (results.value == 16580863){//код красной кнопки
      digitalWrite(11, LOW);
      digitalWrite(12, LOW);
    }
    irrecv.resume();
  }
}
```

Только имей в виду: если хочешь использовать 16-ричную запись кодов, то нужно ставить перед ней "0x". Например: 0xFD48B7, 0xFD6897, 0xFD9867...

Более удобно использовать в таких программах оператор switch:

```
if (irrecv.decode(&results)){
  switch(results.value){
    case 16582903:
      digitalWrite(11, HIGH);
      break;
    case 16591063:
      digitalWrite(11, LOW);
```

```
      break;
    case 16615543:
      digitalWrite(12, HIGH);
      break;
    case 16623703:
      digitalWrite(12, LOW);
      break;
    case 16580863:
      digitalWrite(11, LOW);
    digitalWrite(12, LOW);
      break;
  }
  irrecv.resume();
}
```

Дополнительно:
https://robotclass.ru/tutorials/arduino-ir-remote-control/