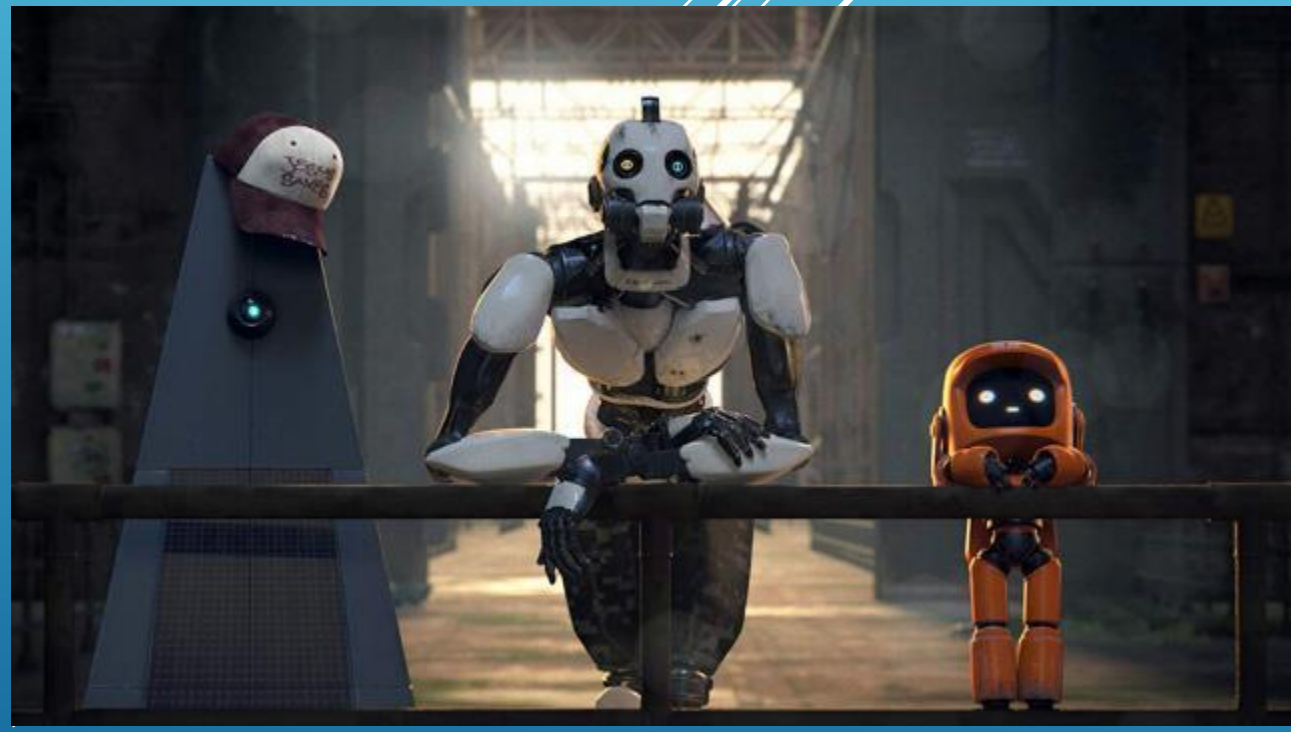
# ОБРАЗОВАТЕЛЬНАЯ РОБОТОТЕХНИКА

**Лекция 3. Подключение внешних устройств к Arduino. Протоколы передачи данных.**

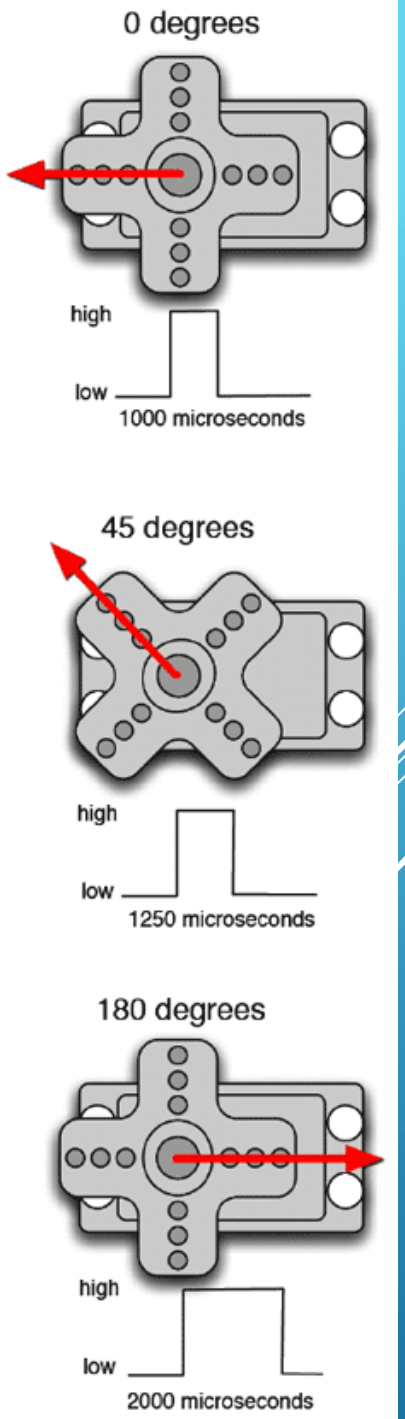


***Пикулев В. Б.***

*http://sandbox.scilink.ru*

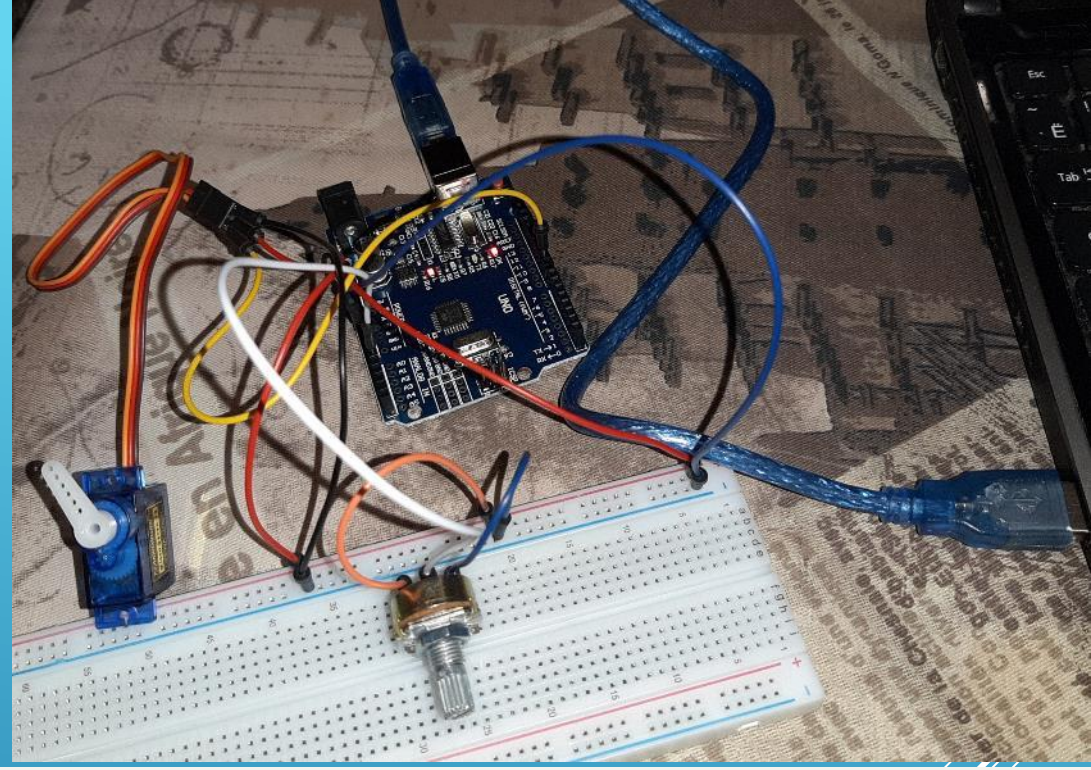
# СЕРВОПРИВОД

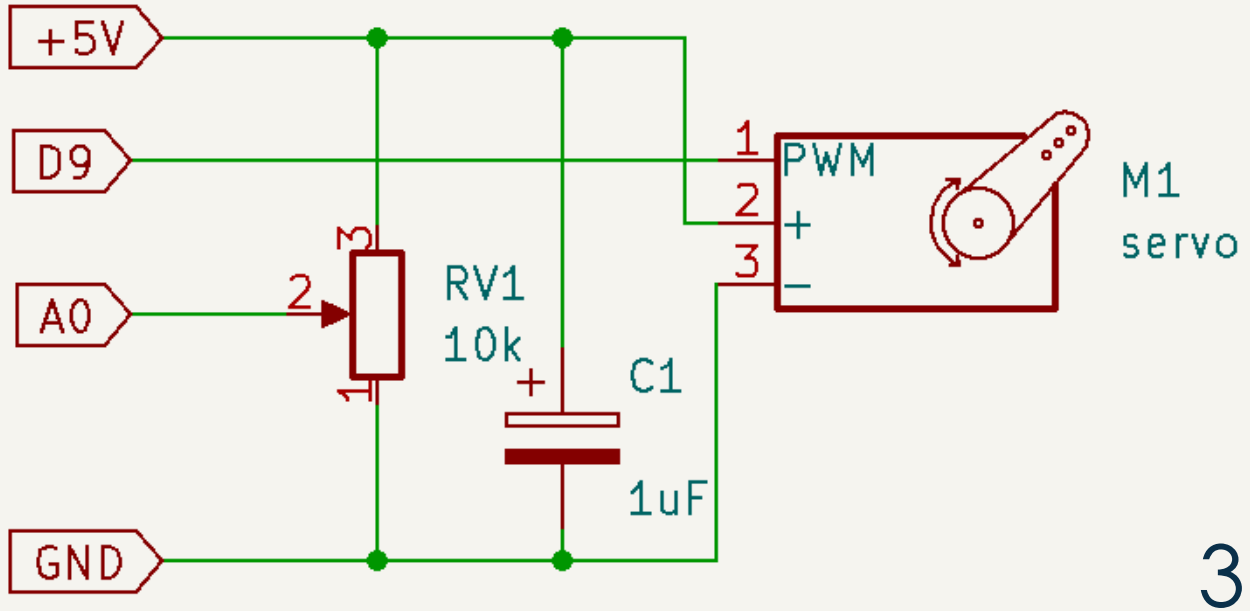
- **Сервопривод** — электромотор с датчиком поворота вала. Сервомотору можно точно в градусах задать положение, в которое встанет вал. Сервоприводы используются для реализации различных механических движений роботов.
- Технически сервопривод состоит из:
  - электромотора постоянного тока,
  - редуктора для увеличения мощности и повышения точности позиционирования,
  - энкодера (обычно в виде потенциометра), связанного с валом редуктора,
  - интерфейсной электроники для управления позиционированием мотора.
- Серводвигатели имеют три контактных провода: питание (красный), земля (чёрный) и сигнальный вход (белый). Управление сервоприводом с цифровым интерфейсом происходит подачей прямоугольного импульса на сигнальный вход. Длительность импульса определяет угол поворота.
- В зависимости от мощности двигателя питание на него может подаваться, минуя плату Arduino (контур с дополнительным питанием).
- Сервоприводы обычно имеют ограниченный угол вращения – 180 градусов. Для устранения скачков питания при старте двигателя рекомендуется параллельно контактам питания мотора ставить конденсатор ~ 500 мкФ.



# УПРАВЛЕНИЕ СЕРВОПРИВОДОМ

- В Arduino IDE предусмотрена библиотека Servo для упрощения управления серво-приводами. В библиотеке по умолчанию выставлены следующие значения длин импульса: 544 мкс — для 0° и 2400 мкс — для 180°.
  - **servo.attach(pin)** - присоединяет "объект servo" к конкретному выводу платы,
  - **servo.write(angle)** – передаёт угол в градусах, на который должен повернуться сервопривод,
  - **servo.read()** – читает угол, на который повёрнут сервопривод (от 0 до 180°),
  - **servo.detach()** – отсоединяет объект от пина.
- Самый простой способ проверить функционирование сервопривода — управление позицией вала с помощью потенциометра. Значение 0 потенциометра соответствует повороту сервопривода на 0°, значение 1023 — повороту на 180°.



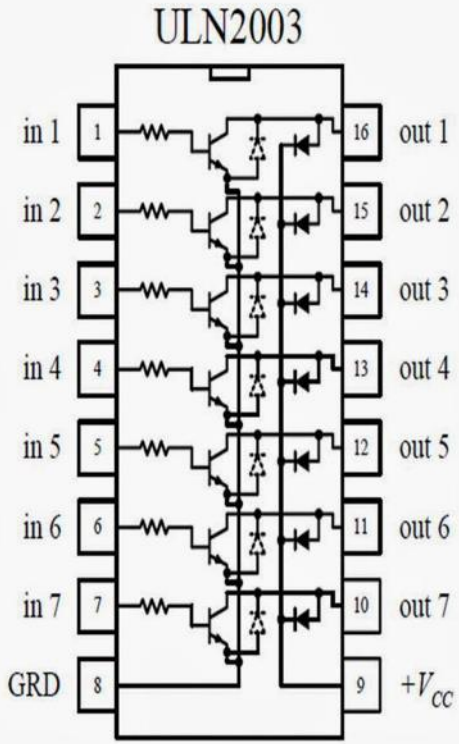


# СКЕТЧ РАБОТЫ С СЕРВОПРИВОДОМ

```
#include <Servo.h>

const int SERVO = 9; // Вывод 9 для подключения сигнального провода сервопривода
const int POT = 0; // Подключение потенциометра к аналоговому входу A0
Servo myServo;
int val = 0; // Переменная для чтения показаний потенциометра

void setup()
{
  myServo.attach(SERVO);
}

void loop ()
{
  val = analogRead(POT); // Чтение данных потенциометра
  val = map(val, 0, 1023, 0, 179); // Преобразование к нужному диапазону
  myServo.write(val); // Установить положение сервопривода
delay(15);
}
```

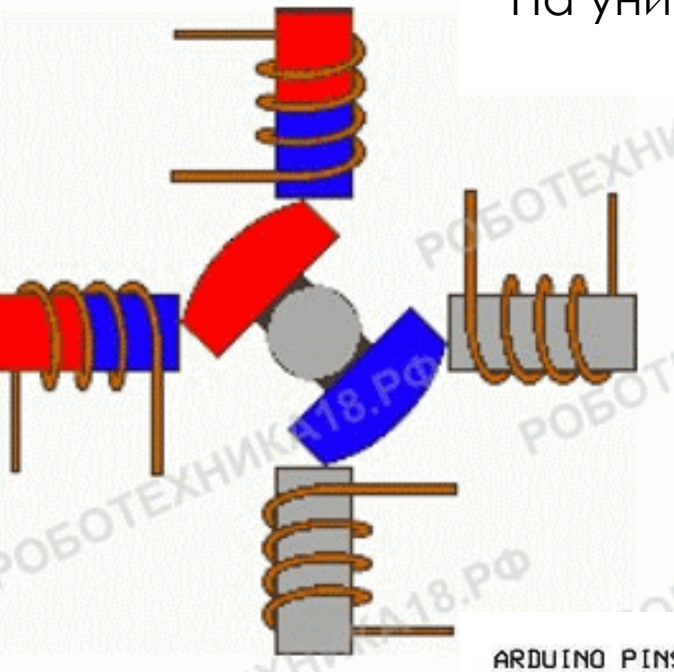
# РАБОТА С ШАГОВЫМ ДВИГАТЕЛЕМ

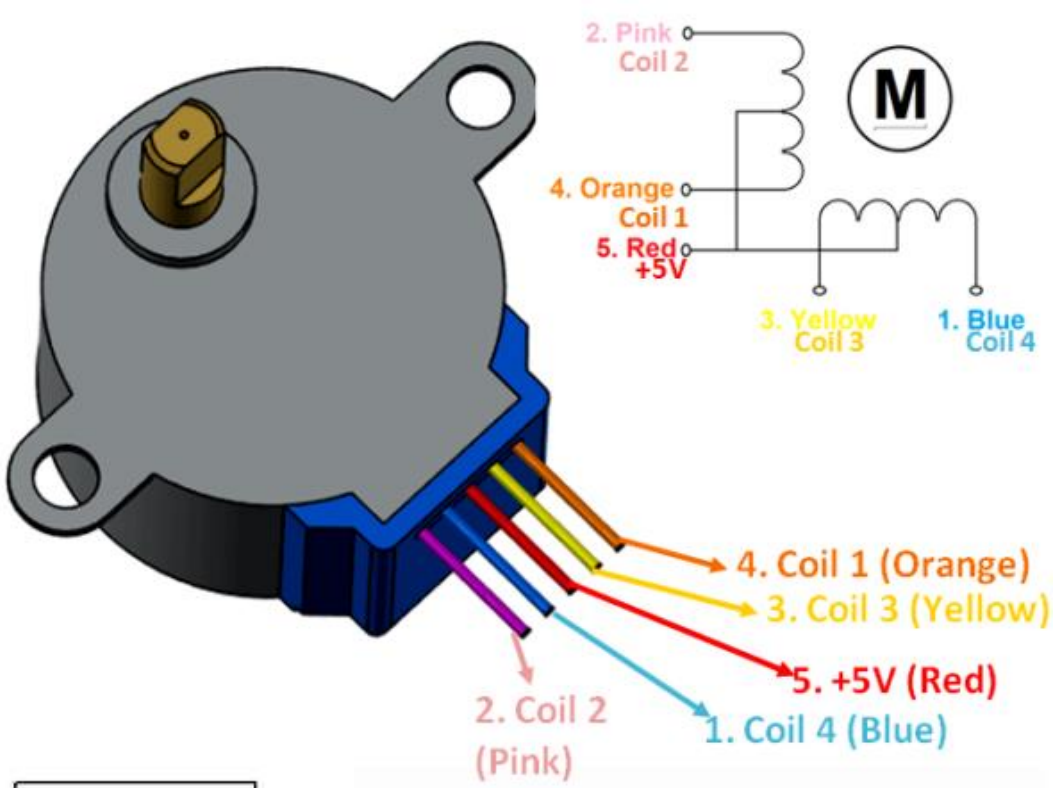
- Шаговый бесколлекторный двигатель позволяет повернуть вал на точно определённый угол, соответствующий одному шагу. В отличие от сервопривода не имеет обратной связи по углу поворота, но зато может вращаться без ограничений, и при этом легко сосчитать, какое количество шагов он сделал.
- Обычно на вал шагового двигателя также устанавливается понижающий редуктор, так что при 64 шагах на оборот двигателя, входящего в комплект Arduino Start, получается 4096 шагов на оборот выходного вала.
- Управление шаговым двигателем через Arduino производится путем подачи импульсов на обмотки мотора в определенной последовательности. Подключение двигателя к четырём цифровым выводам платы осуществляется через буферную микросхему (ULN2003AN – коммутатор мощных нагрузок на основе транзисторов Дарлингтона) либо с использованием специализированной микросхемы-драйвера с отдельным питанием. В последнем случае используется всего два вывода – шаг и направление движения.

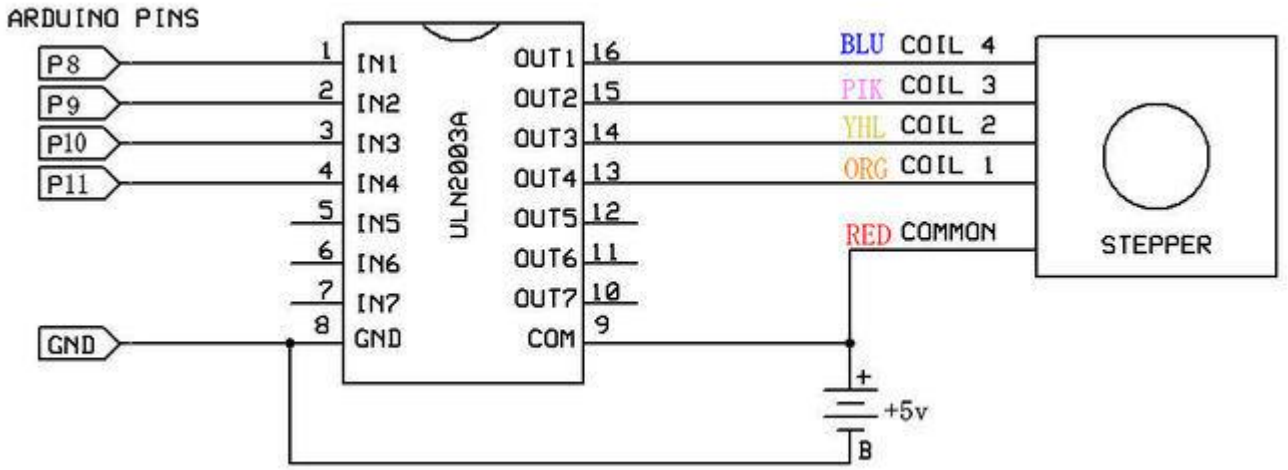


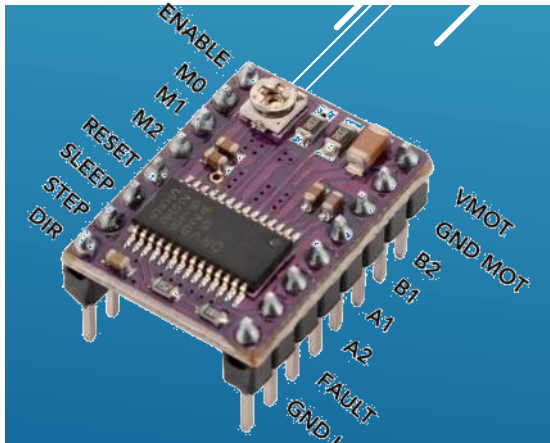
# УПРАВЛЕНИЕ ШАГОВЫМ ДВИСГАТЕЛЕМ

На униполярный двигатель подаётся +5В



# СКЕТЧ ДЛЯ УПРАВЛЕНИЯ ШД

```
#include <Stepper.h>
const int stepsPerRevolution = 2038;  // число шагов на 1 оборот
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);

void setup() {
myStepper.setSpeed(100); // скорость в об./мин
Serial.begin(9600);
}

void loop() {
  // оборот по часовой стрелке
  Serial.println("clockwise");
  myStepper.step(stepsPerRevolution);
  delay(500);

  // оборот против часовой стрелки
  Serial.println("counterclockwise");
  myStepper.step(-stepsPerRevolution);
  delay(500);
}
```

```
void loop() {
  // одиночный шаг
  myStepper.step(1);
  Serial.print("steps:");
  Serial.println(stepCount);
  stepCount++;
  delay(500);
}
```

Библиотека AccelStepper – позволяет вращать мотор с плавным ускорением и торможением.

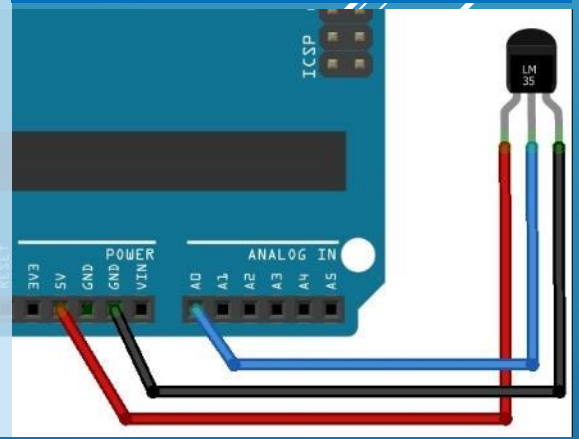
В случае сложных проектов легче написать собственный алгоритм для вращения двигателя.

# АНАЛОГОВЫЙ ДАТЧИК ТЕМПЕРАТУРЫ

- Устройства LM35 – это прецизионные интегральные датчики температуры, у которых выходное напряжение пропорционально температуре по шкале Цельсия. Датчик обеспечивает измерение температуры с точностью ±0.25 °C в комнатных условиях и с точностью ±0.75 °C в полном диапазоне рабочих температур -55 … +150 °C, без внешней калибровки или подгонки выходного напряжения.
- Датчик работает в широком диапазоне напряжения питания 4 – 30 В. Изменение напряжения 10.0 mV на 1°C, то есть если температура датчика 25°C – на выходе (при идеальной калибровке) будет 250mV.
- Диапазон входного напряжения для Arduino от 0 до 5 В. То есть, Arduino не сможет брать с этого датчика отрицательные значения напряжения, если не использовать специальную калибровку и двуполярное питание датчика.

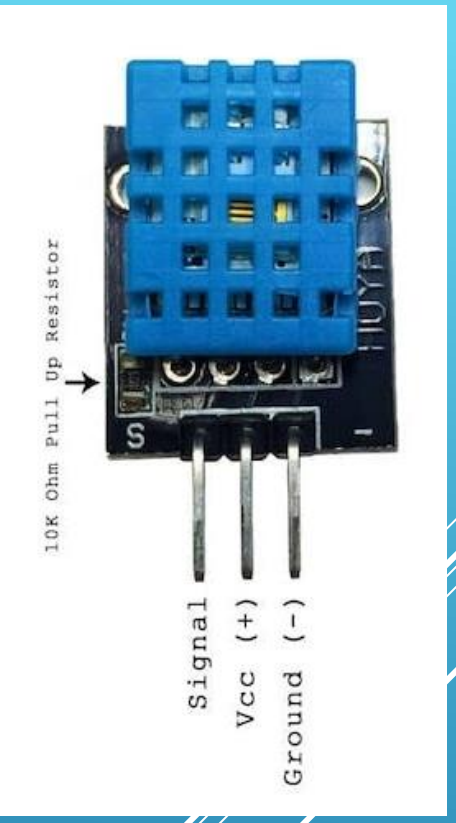

```
float tempC;
int reading;

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

void loop()
{
  reading = analogRead(A0);
  tempC = reading / 9.31;
  Serial.print(tempC);
  Serial.println(" C");
  delay(1000);
}
```

Для правильного чтения информации с аналогового входа необходимо включить внутренний источник опорного напряжения **1.1 В**



# ЦИФРОВОЙ ДАТЧИК ТЕМПЕРАТУРЫ И ВЛАЖНОСТИ

- Датчик DHT-11 состоит из ёмкостного датчика влажности и термосопротивления. Находящийся внутри корпуса чип выполняет аналого-цифровые преобразования и выдаёт по запросу цифровой код. Датчик измеряет влажность в диапазоне от 20% до 95% с погрешностью 5%. Измеряет температуру в интервале от 0 до 50 градусов с точностью 2°. Частота опроса – не более 1 раза в секунду. Напряжение питания: 3.5 – 5 В.
- Количество передаваемой от датчика информации— 40 битов (5 байтов). Первый и второй байты содержат целую и дробную часть информации о влажности, третий и четвертый байты - целую и дробную часть информации о температуре, пятый байт - контрольная сумма, которая представляет собой последние 8 битов от сложения предыдущих 4 байтов.
- Модуль с датчиком DHT-11 оборудован трехпиновым разъемом: G — подключается к земле, V — подключается к +5V, S — подключается к выбранному цифровому выводу.
- Протокол общения с датчиком нестандартный, но полностью поддерживается внешней библиотекой DHT. Данные передаются побитно. Перед передачей каждого нового бита датчик выставляет 0 на линии на 50 мкс, после чего возвращает линию к 1. Затем, если длительность временного промежутка до следующего выставления нуля 26-28 мкс, то передан 0, если 70 мкс — передана 1.




распиновка модулей у разных производителей может отличаться.

# СКЕТЧ ДЛЯ РАБОТЫ С DHT

```
#include <DHT.h>     // подключаем библиотеку для датчика
DHT dht(2, DHT11);  // указываем порт и модель датчика

void setup() {
   dht.begin();           // инициализируем датчик DHT11
   Serial.begin(9600);    // подключаем монитор порта
}

void loop() {
   // считываем температуру и влажность
   float h = dht.readHumidity();
   float t = dht.readTemperature();

   Serial.print("Humidity: ");
   Serial.println(h);
   Serial.print("Temperature: ");
   Serial.println(t);

   delay(1500);
}
```

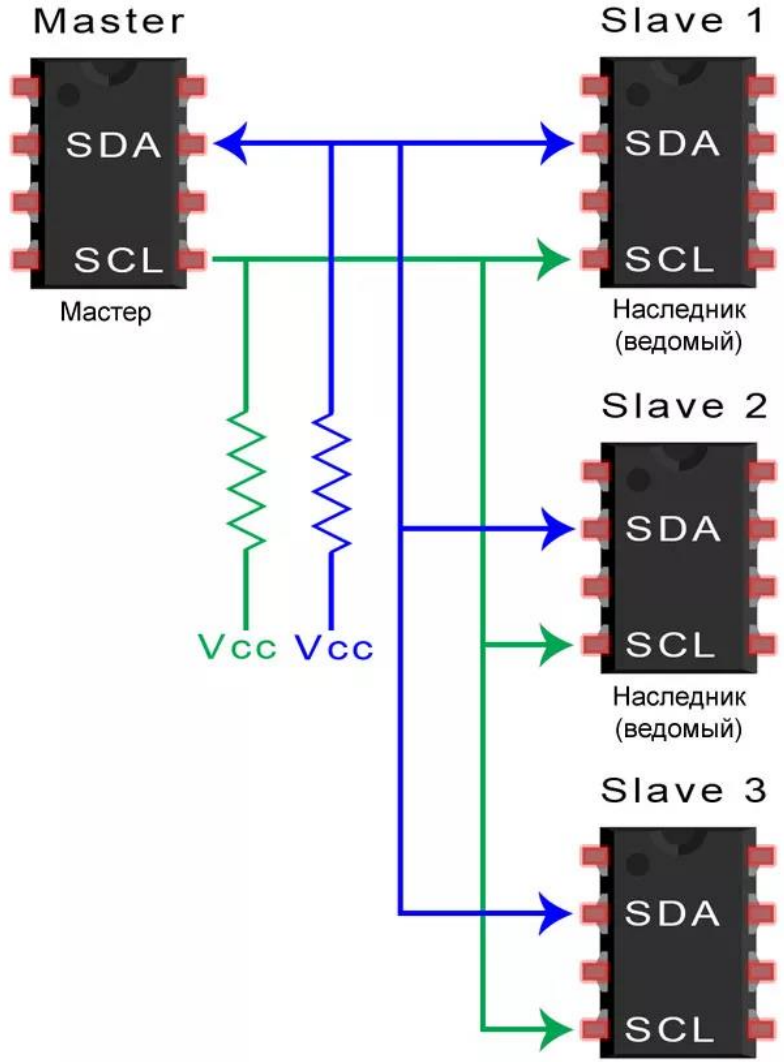
**Характеристики DHT-22:**

- Питание – от 3 до 5 В;
- Максимальный ток – 2,5 мА;
- Способен измерять влажность в интервале от 0% до 100% с точностью от 2% до 5%;
- Минимальная измеряемая температура – минус 40, максимальная – 125 градусов по Цельсию (точность измерений – 0,5°);
- Быстродействие: одно измерение за 2 секунды.

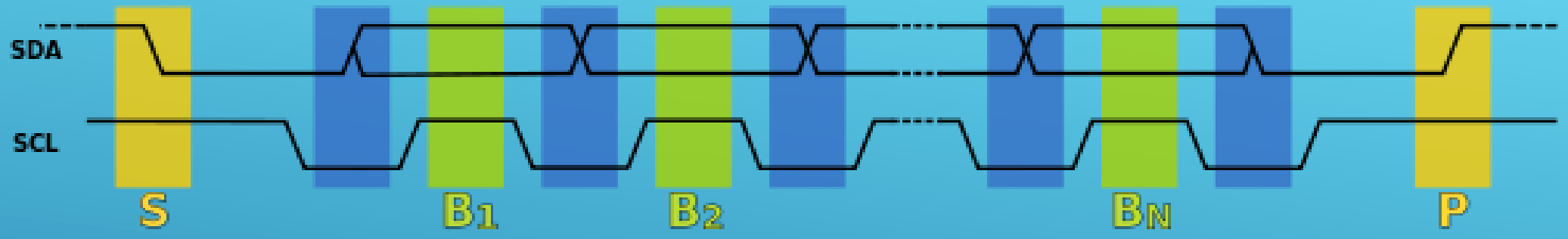
Подключение и работа с Arduino точно такие же, как и для DHT-11.

# ПРОТОКОЛ I²C

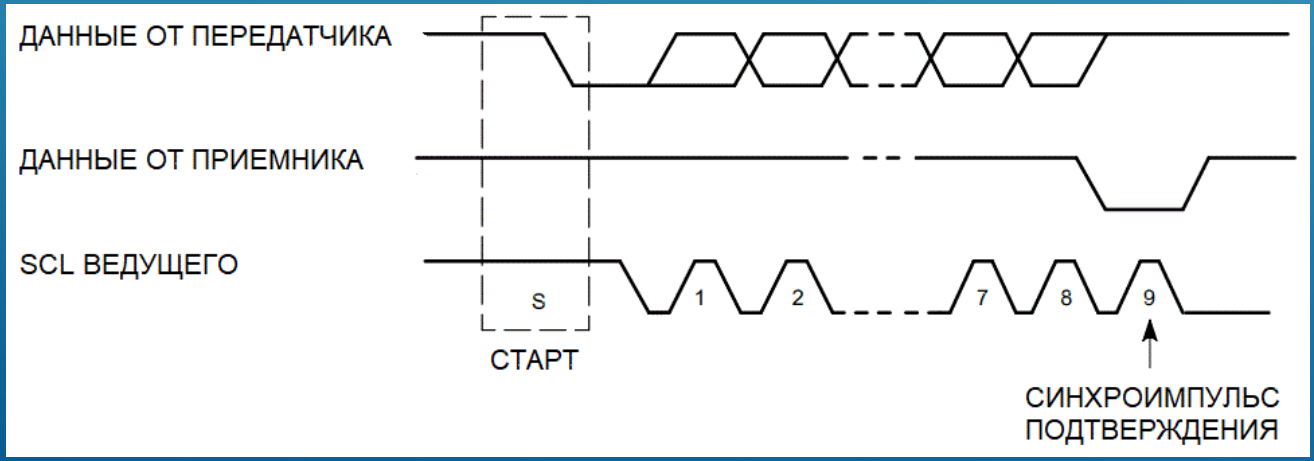
- Низкоскоростной последовательный протокол обмена данными I²C (Philips, Inter-Integrated Circuit) использует для передачи данных две двунаправленные линии связи (шину):
  - **SDA** (Serial Data) — линия последовательных данных;
  - **SCL** (Serial Clock) — линия тактирования (синхронизации).
- Нужны также линия питания и линия земли. Линии SDA и SCL подтягиваются к линии питания через резисторы.
- К одной шине I²C может быть подключено до **127** устройств. В сети должно присутствовать хотя бы одно ведущее устройство (master), которое инициализирует передачу данных и генерирует сигналы синхронизации, остальные устройства являются ведомыми (slave) и передают данные по запросу ведущего. Все ведомые устройства имеют адрес, который используется для идентификации устройства в сети. Одновременно только одно устройство может занимать шину для передачи данных.
- Arduino использует для работы по интерфейсу I²C два контакта (для Uno это 4 и 5, для Mega – 20 и 21). Для обмена данными с устройствами по шине I²C в Arduino есть стандартная библиотека Wire (установлена по умолчанию).
- Две платы Arduino можно соединить по I²C и обмениваться данными между ними.

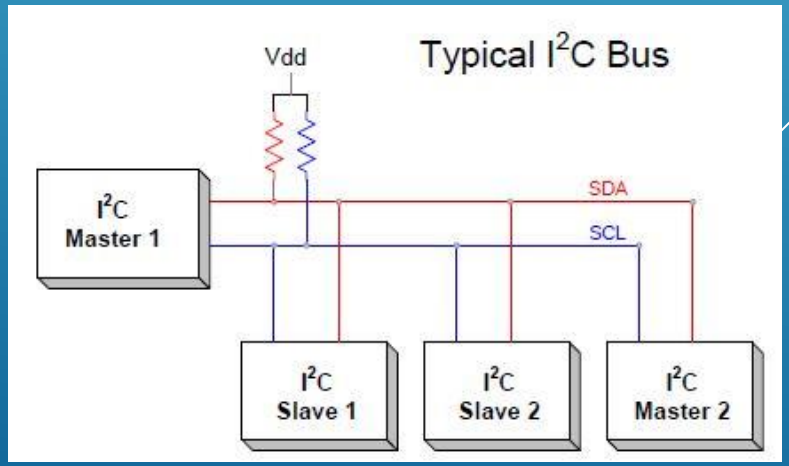


# ПРОТОКОЛ I²C



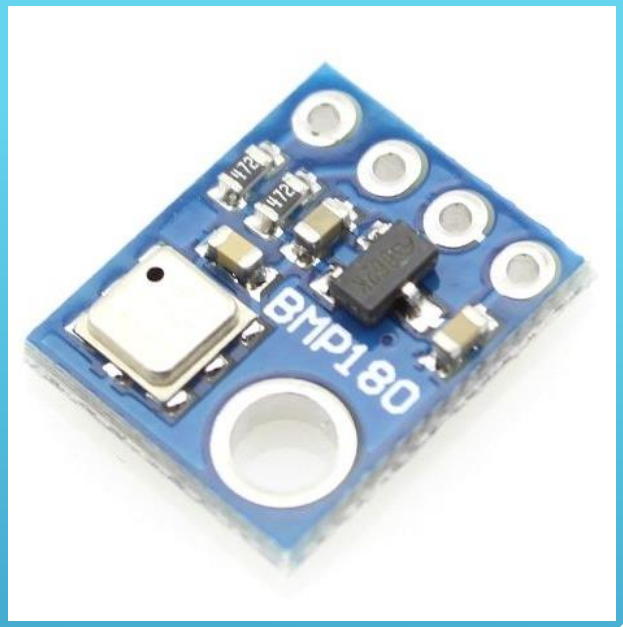




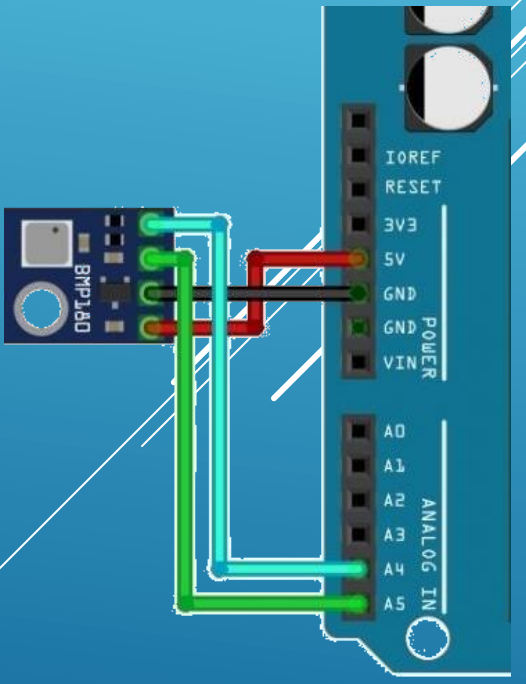



# ПОДКЛЮЧЕНИЕ ДАТЧИКА BMP180 ЧЕРЕЗ I²C

- Датчик BMP180 – это барометр, созданный по МЭМС-технологии, в котором используется эффект изменения сопротивления материала под действием внешнего атмосферного давления. Не входит в стартовый набор Arduino.
- Характеристики BMP180:
  - диапазон измеряемых значений: от 300 гПа до 1100 гПа (от -500м от +9000м над уровнем моря);
  - напряжение питания: от 3.3 до 5 Вольт;
  - скорость опроса – 1 Гц.
- Датчик возвращает величину давления в гектопаскалях (гПа), при этом показания автоматически корректируются на температуру датчика. По изменению давления можно определить изменение высоты, что определяет основное предназначение этих датчиков как альтимертов для дронов.
- Для удобной работы с датчиком, помимо библиотеки Wire, используется одна из библиотек для обслуживания этого датчика (например, SFE_BMP180).




```
#include <SFE_BMP180.h>
#include <Wire.h>

SFE_BMP180 pressure;

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

void loop(){
    double P;
    P = getPressure();
    Serial.println(P, 4);
    delay(100);
}
```



# ПРОГРАММИРОВАНИЕ ДЛЯ ДАТЧИКА BMP180

```
double getPressure(){
    char status;
    double T,P;
    status = pressure.startTemperature();
    if (status != 0){
        // ожидание замера температуры
        delay(status);
        status = pressure.getTemperature(T);
        if (status != 0){
            status = pressure.startPressure(3);
            if (status != 0){
                // ожидание замера давления
                delay(status);
                status = pressure.getPressure(P,T);
                if (status != 0){
                    return(P);
                }
            }
        }
    }
}
```

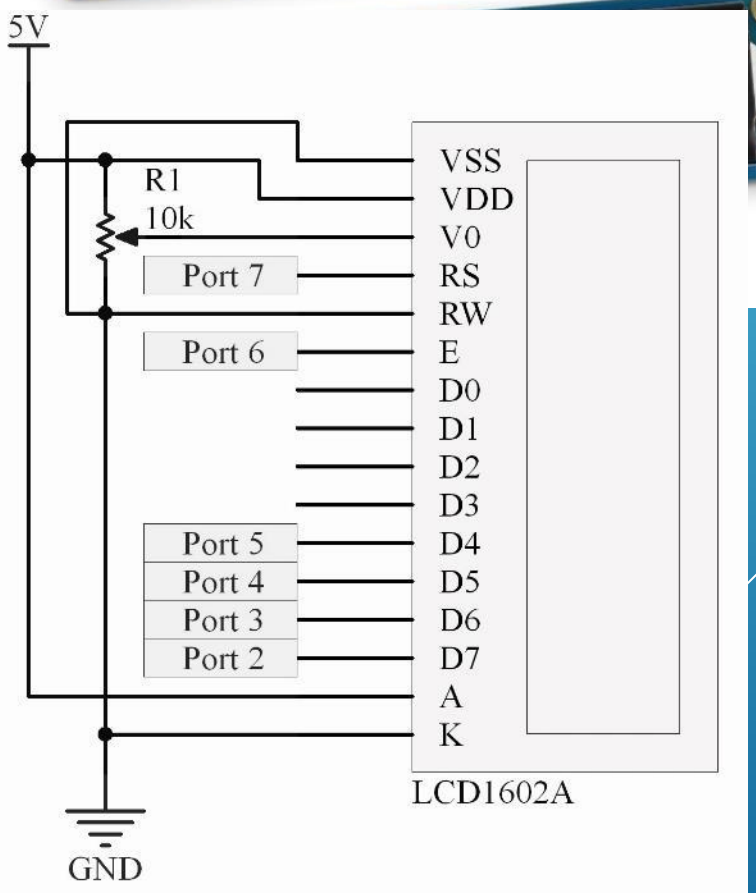
Если все хорошо, функция **pressure.startTemperature()** вернет status с количеством милисикунд, которые нужно подождать. Если какая-то проблема, то функция вернет 0.

Параметр в **pressure.startPressure(3)** указывает расширение, от 0 до 3. Чем больше расширение, тем больше точность, тем дольше ждать результата.

Подключаются устройства с интерфейсом $I^2C$ к пинам A4 (SDA) и A5 (SCL) платы Arduino Uno. Несколько устройств с разными адресами подключаются параллельно.

# ДИСПЛЕЙ LCD1602

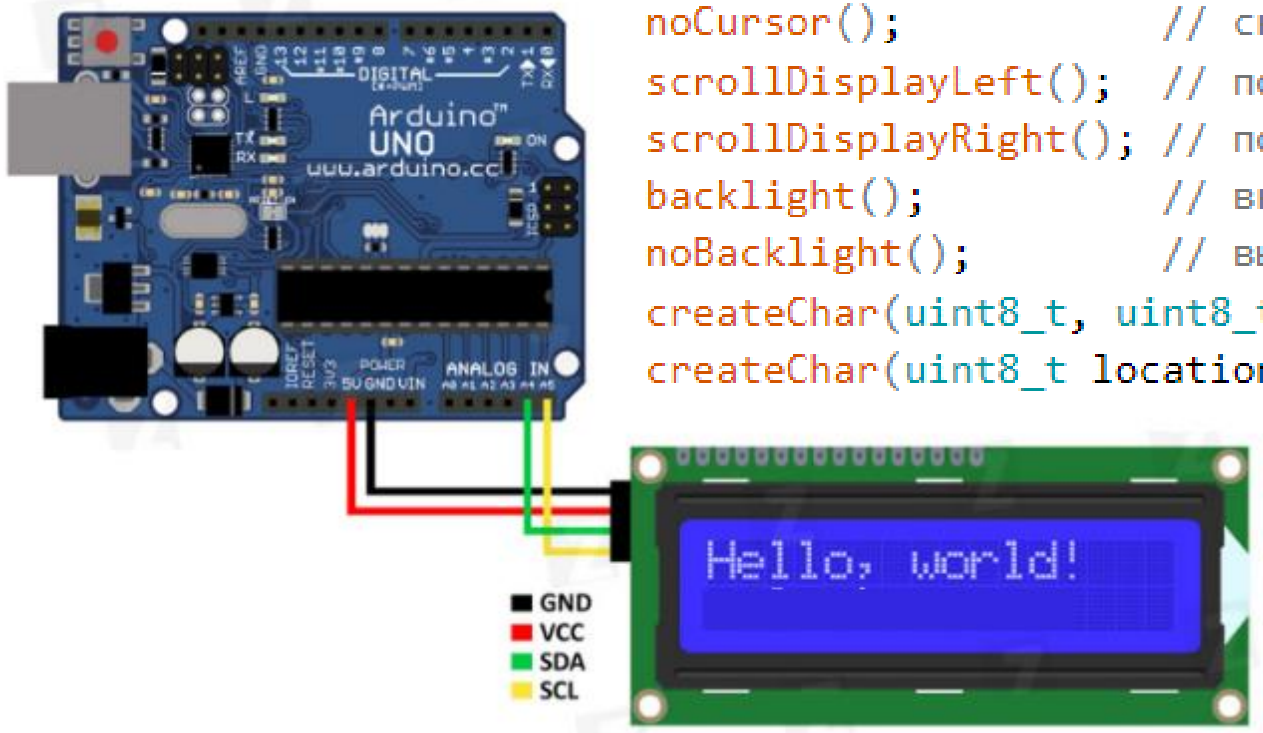
- Жидкокристаллические текстовые дисплеи часто используются в проектах Arduino. Обычно это модуль, состоящий из микроконтроллера HD44780 и непосредственно самого дисплея. Микроконтроллер принимает команды и отрисовывает на дисплее соответствующие символы. Однако использовать большое количество выводов для подключения микроконтроллера неэкономно, поэтому в модуль дисплея дополнительно вводят I²C-конвертер, часто в виде отдельной платы.
- Подключается дисплей четырьмя контактами: +5В, GND, SDA, SCL. Обычно текстовые дисплеи имеют знакогенератор, уже прошитый в памяти контроллера, перепрограммируемых символов очень мало. Обычно русские буквы либо отсутствуют, либо сложно отображаются.
- Для работы с дисплеем удобно использовать библиотеку LiquidCrystal_I2C, которую можно скачать из репозитория Arduino. Подсветкой дисплея можно также управлять с помощью соответствующих функций библиотеки.

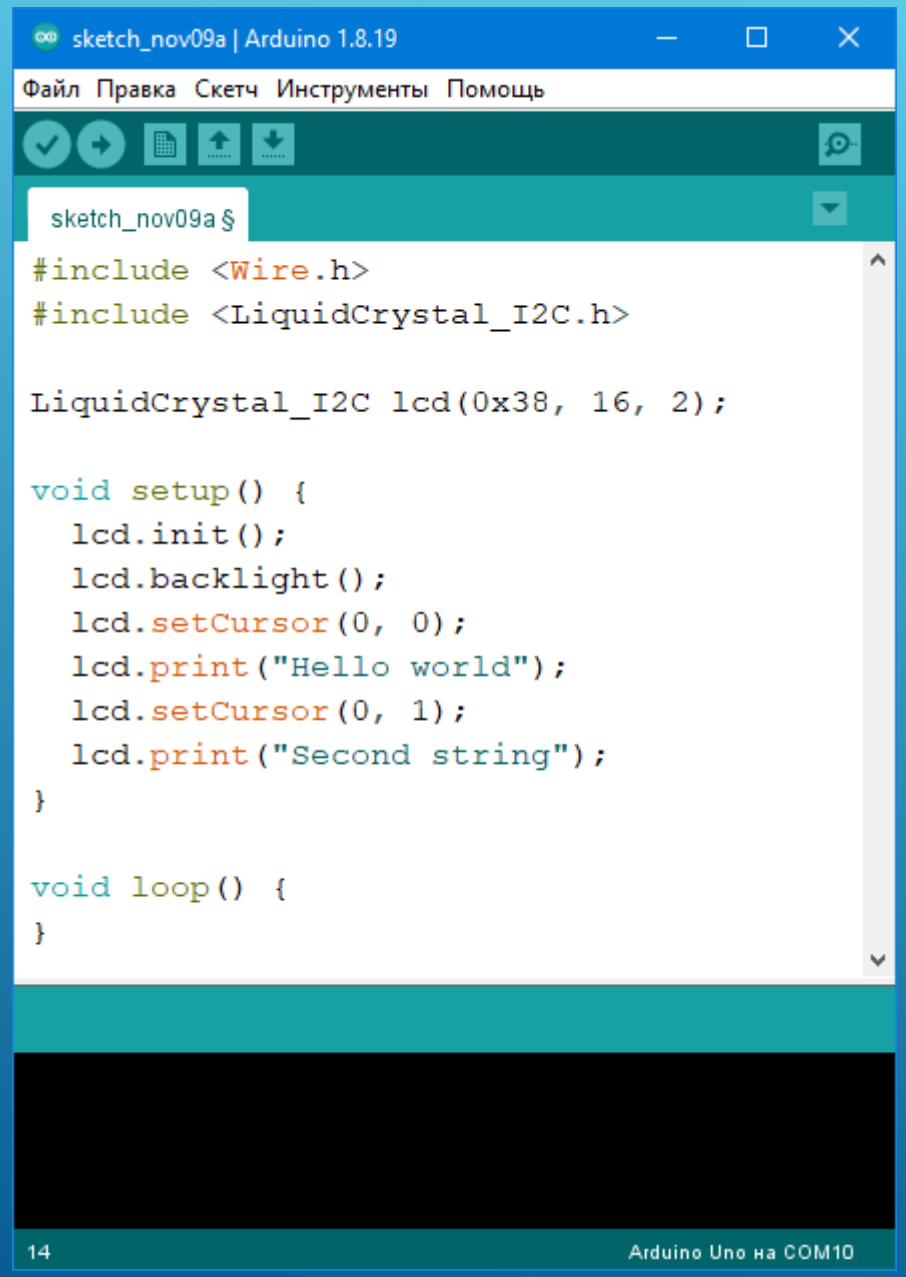
# ПОДКЛЮЧЕНИЕ LC-ДИСПЛЕЯ ЧЕРЕЗ I²C

Возможности библиотеки LiquidCrystal_I2C:

```
print(data);          // вывести (любой тип данных)
setCursor(x, y);      // курсор на (столбец, строка)
clear();              // очистить дисплей
home();               // аналогично setCursor(0, 0)
noDisplay();          // отключить отображение
display();            // включить отображение
blink();              // мигать курсором на его текущей позиции
noBlink();            // не мигать
cursor();             // отобразить курсор
noCursor();           // скрыть курсор
scrollDisplayLeft();  // подвинуть экран влево на 1 столбец
scrollDisplayRight(); // подвинуть экран вправо на 1 столбец
backlight();          // включить подсветку
noBacklight();        // выключить подсветку
createChar(uint8_t, uint8_t[]);                    // создать символ
createChar(uint8_t location, const char *charmap);  // создать символ
```



# ПРОГРАММИРОВАНИЕ РАБОТЫ С LC-ДИСПЛЕЕМ



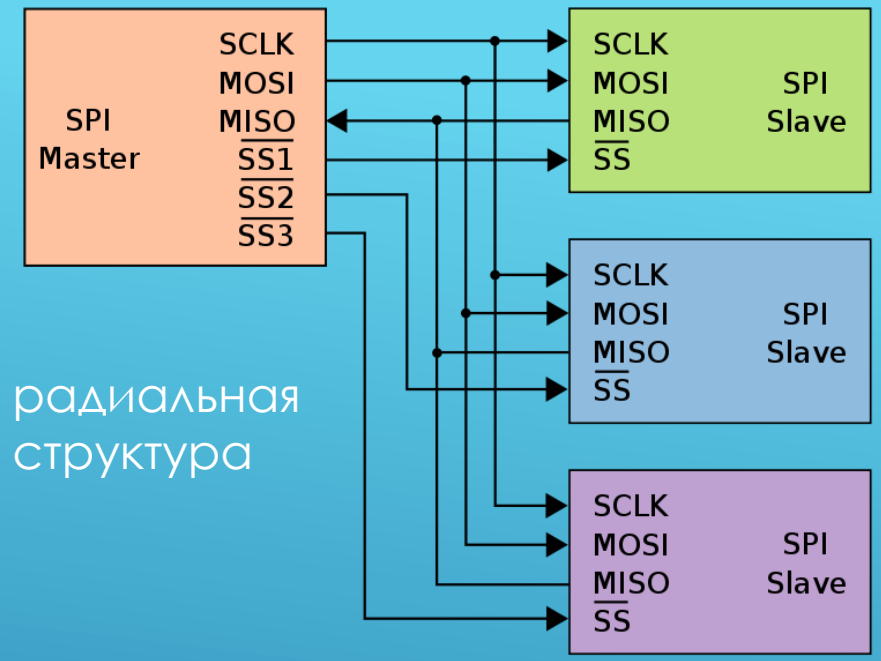

Фрагмент более сложного кода

```
void Change_screen() {
  for (int i=0; i<16; i++) {
    if (sd) { lcd.scrollDisplayLeft();
    } else { lcd.scrollDisplayRight(); }
  }
  if (sd) {
      lcd.setCursor(16, 0);
      lcd.print(Get_meteo());
      lcd.setCursor(16, 1);
      lcd.print("SLS " + Get_time(true));
  }
  sd = !sd;
  delay(150);
}
```

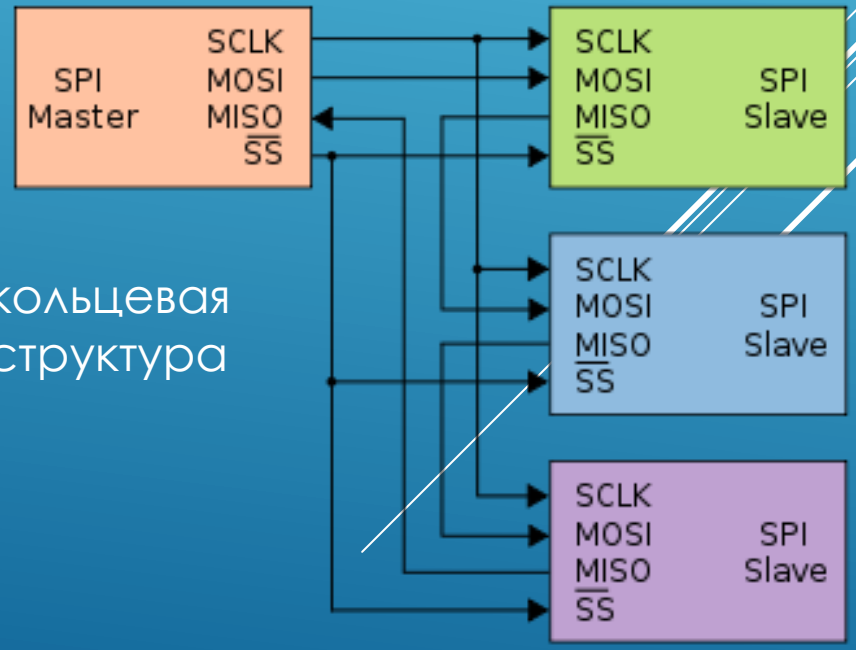
Единственная проблема – знать адреса устройств на шине. Адрес устанавливается при изготовлении устройства, и узнать его порой непросто. Для LCD это обычно 0x38 или 0x27.

# ИНТЕРФЕЙС SPI

- SPI – Motorola Serial Peripheral Interface, шина последовательного периферийного интерфейса, последовательный синхронный стандарт передачи данных, предназначенный для обеспечения простого высокоскоростного сопряжения микроконтроллеров и периферии.
- SPI, в отличие от рассмотренных ранее интерфейсов, является синхронным, т.е. в нём любая передача синхронизирована с общим тактовым сигналом, генерируемым ведущим устройством. Шина SPI имеет отдельные линии для отправки и получения данных, а также дополнительную линию для выбора ведомого устройства. Всего линий четыре (если устройство одно – тогда линий три):
  - *MOSI* — выход ведущего, вход ведомого (Master Out Slave In). Служит для передачи данных от ведущего устройства к ведомому.
  - *MISO* — вход ведущего, выход ведомого (Master In Slave Out). Служит для передачи данных от ведомого устройства ведущему.
  - *SCLK* — тактовый сигнал (Serial Clock). Служит для передачи тактового сигнала для ведомых устройств.
  - $\overline{SS}$ — выбор микросхемы, выбор ведомого (Slave Select).

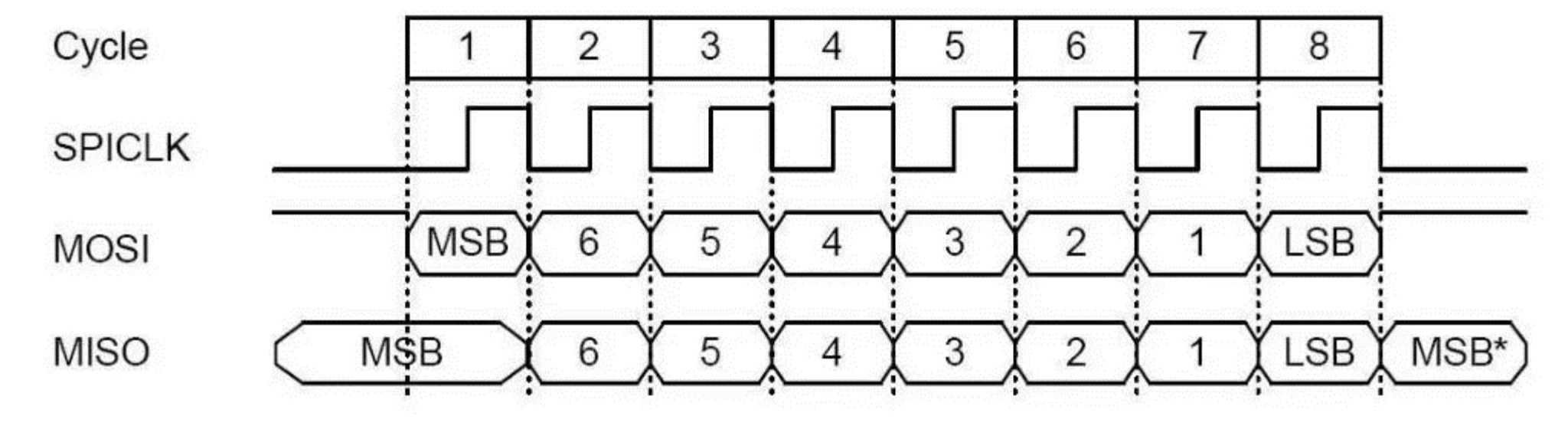



радиальная структура




кольцевая структура

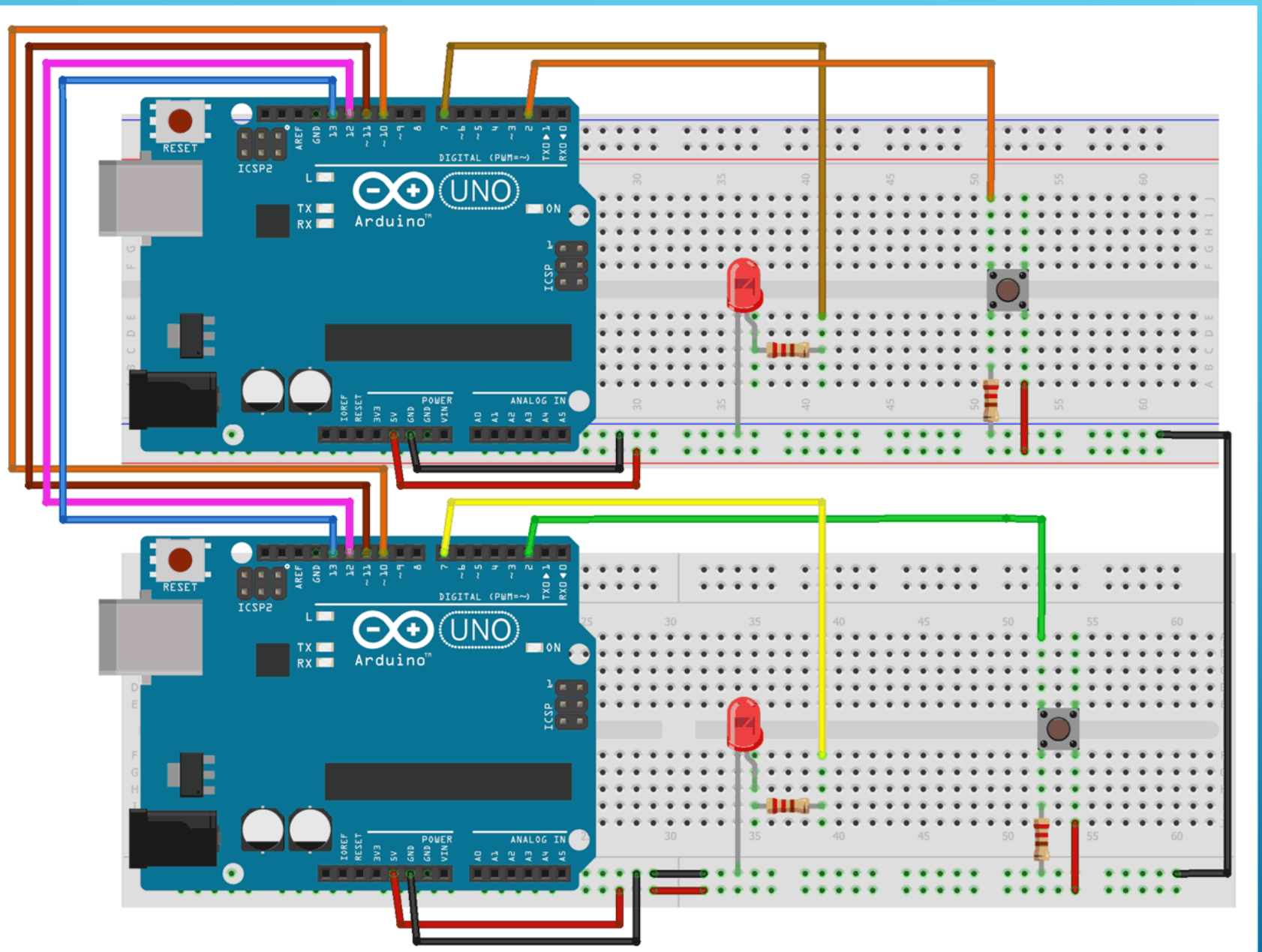
# ИСПОЛЬЗОВАНИЕ ПРОТОКОЛА SPI




- Поскольку SPI не является универсальным стандартом, существуют различные способы обмена информацией с периферийными устройствами разных типов.
- Все команды, отправляемые мастером, проявляются на входах MOSI, MISO, SCLK всех ведомых устройств. Состояние контакта SS сообщает устройству, игнорировать эти данные или принимать. На каждом такте данные должны быть отправлены (или получены).

Пусть наше устройство работает в режиме 0 (один из четырёх вариантов синхронизации, наиболее распространённый). В таком случае ведомое устройство будет считывать бит данных со входа MOSI по переднему нарастающему фронту синхронизирующего сигнала, ведущее устройство будет считывать данные от ведомого на входе MISO также по переднему нарастающему фронту.

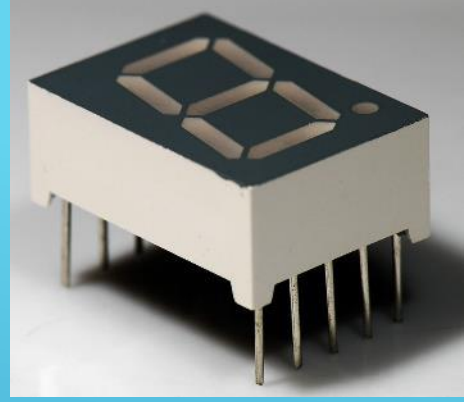
# СВЯЗЬ МЕЖДУ ПЛАТАМИ ARDUINO ПО SPI



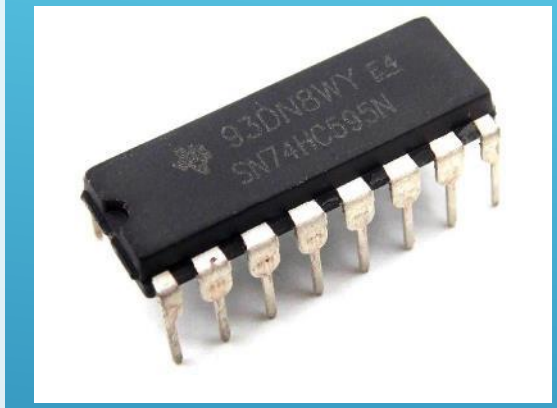

Когда нажимается кнопка на верхней плате, загорается светодиод на нижней. Когда нажимается кнопка на нижней плате, загорается светодиод на верхней.

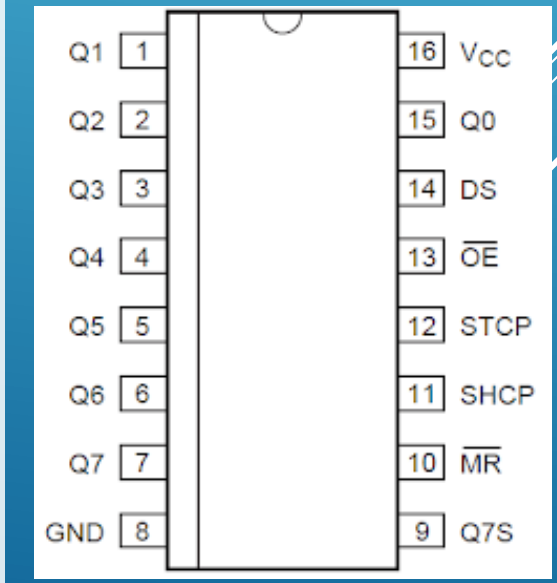
Коммуникация между платами запускается тогда, когда состояние кнопки изменилось. Для ведомой платы реализуется режим работы программы с прерываниями.

# ПОДКЛЮЧЕНИЕ 7-СЕГМЕНТНОГО ИНДИКАТОРА

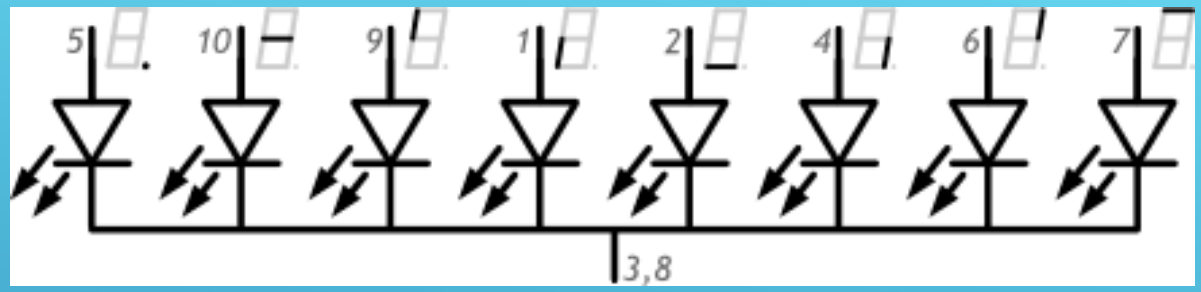


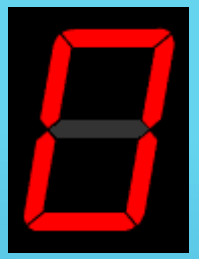
- Предлагается использовать в качестве ведомого устройства микросхему - восьмибитный сдвиговый регистр-защёлку **74HC595**, к которому будет подключён 7-сегментный индикатор. Принцип действия такого регистра заключается в том, что он последовательно получает биты на контакт DS, а при выставлении низкого уровня на контакте STCP «защелкивает» полученные биты так, что на его параллельных выводах устанавливается весь полученный байт. Используемые контакты:
  - **$V_{CC}$** – питание +5В;
  - **GND** – земля;
  - **DS** – последовательный ввод. Должен подключаться к выходу MOSI шины SPI;
  - **SHCP** – тактовый вход. Должен подключаться к выходу SCLK шины SPI;
  - **STCP** – защелкивание выводов. Должен подключаться к выходу CS шины SPI;
  - **Q0:Q7** – параллельные выходы, подключаются к дисплею;
  - **~OE** – разрешение работы выходов. При низком уровне выходы работают.



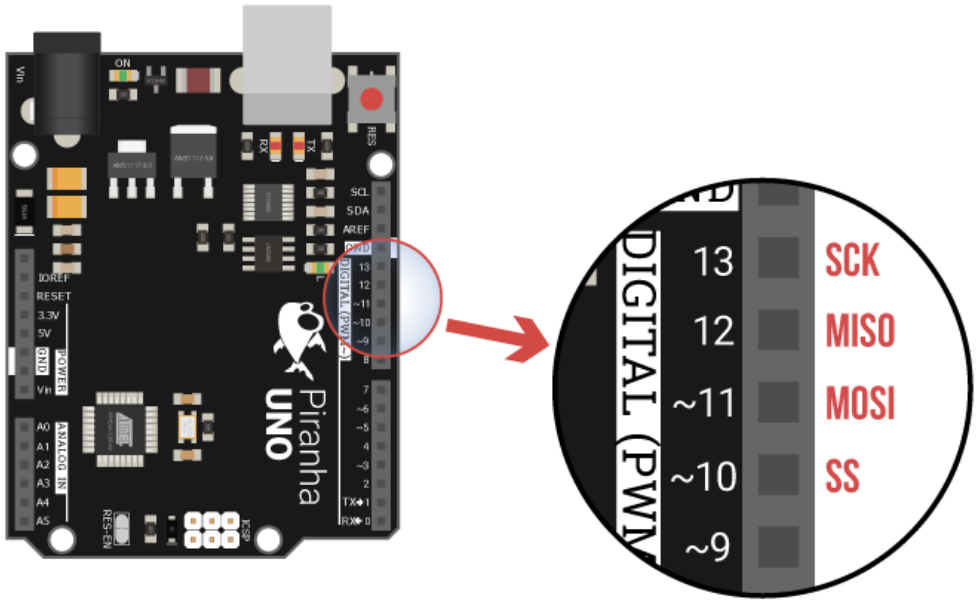



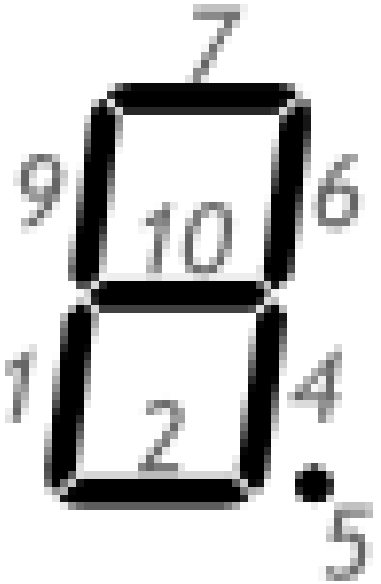
# ПОДКЛЮЧЕНИЕ 7-СЕГМЕНТНОГО ИНДИКАТОРА

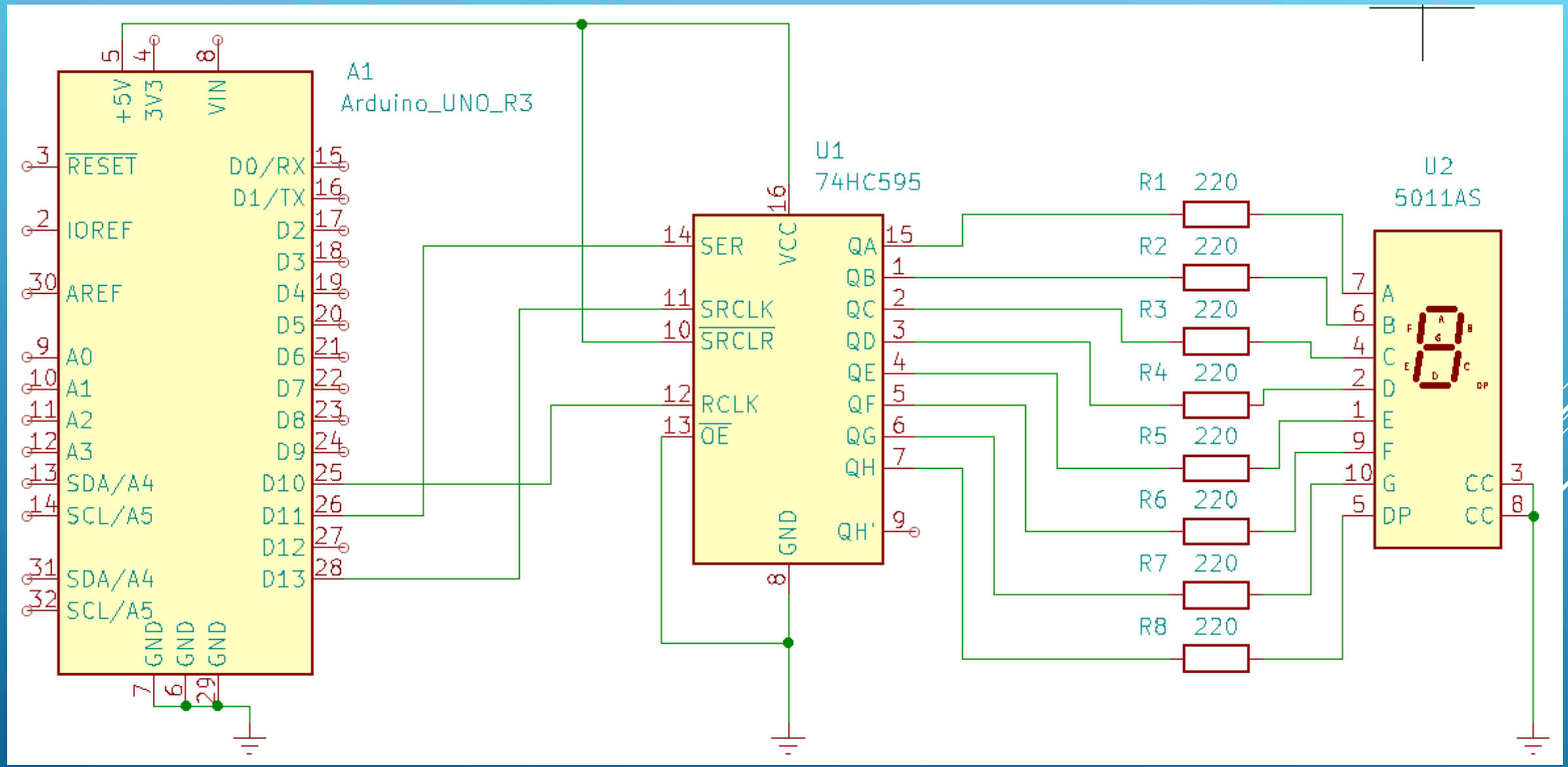
Индикаторы могут быть с общим анодом или с общим катодом (контакты 3 и 8). В наборах Arduino start используются индикаторы 5011AS с общим катодом (подключаются к земле).




| Цифры индикатора | 5 10 9 1 2 4 6 7 |
|---|---|
| 0 | 00111111 |
| 1 | 00000110 |
| 2 | 01011011 |
| 3 | 01001111 |
| 4 | 01100110 |
| 5 | 01101101 |
| 6 | 01111101 |
| 7 | 00000111 |
| 8 | 01111111 |
| 9 | 01101111 |

# ПОДКЛЮЧЕНИЕ 7-СЕГМЕНТНОГО ИНДИКАТОРА



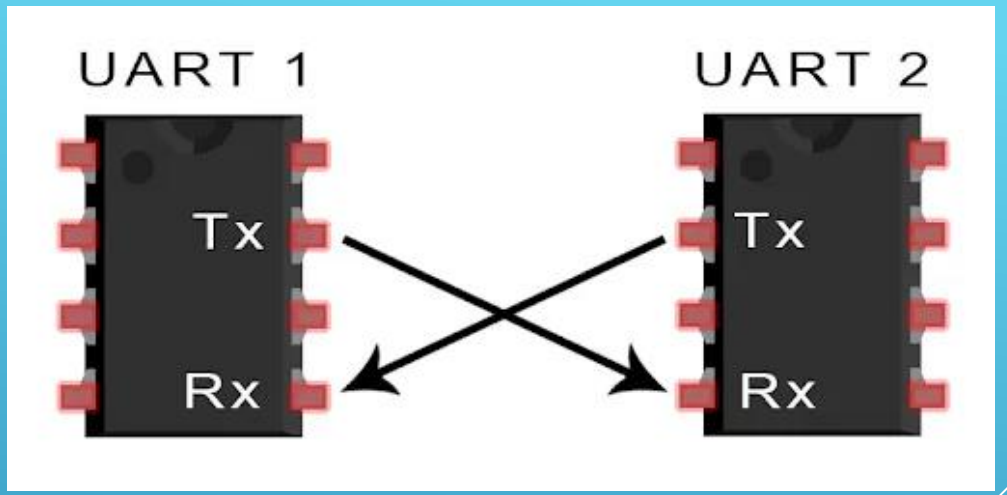
# 7-СЕГМЕНТНЫЙ ИНДИКАТОР НА SPI

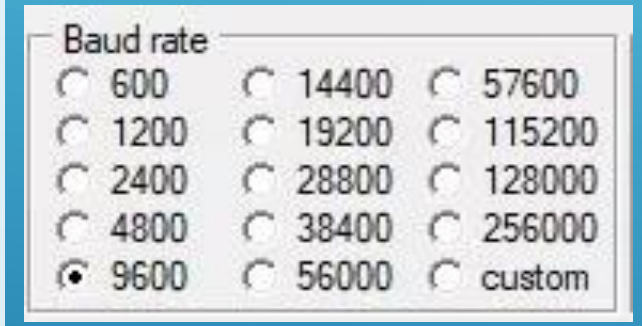
```
#include <SPI.h>
enum { reg = 10 }; // CS на 10 пине Arduino
static uint8_t digit[16] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,
0x88,0x83,0xC6,0xA1,0x86,0x8E};

void setup()
{
  SPI.begin();
  pinMode(reg, OUTPUT);
}
```

```
void loop()
{
  // выводим цифры по одной
  for (int i=0; i<16; i++){
    digitalWrite(reg, LOW); // начало передачи
    SPI.transfer(digit[i]);
    digitalWrite(reg, HIGH); // конец передачи
    delay(1000);
  }
  //очистка дисплея
  digitalWrite(reg, LOW);
  SPI.transfer(0);
  digitalWrite(reg, HIGH);
  delay(1000);
}
```

# ИНТЕРФЕЙС UART

- **UART** (Univsersal Asynchronos Reciever-Transmitter) – асинхронный протокол обмена через последовательный порт. Базируется на спецификациях RS-232, RS-485, COM и др. Предполагает двунаправленный обмен данными между **двумя** устройствами. При этом оба устройства являются равноправными — явно выраженного деления устройств на главное и подчиненное не предполагается. Для организации обмена данными используются всего две линии: по одной данные передаются от первого устройства ко второму, а по другой — в обратном направлении. На микроконтроллере при этом задействуются два контакта (пина): **TX** — передающий и **RX** — принимающий. Линии передачи соединяют TX первого устройства с RX второго и RX первого устройства с TX второго. Дополнительно необходимо объединить референсные уровни двух устройств (GND-GND).
- Arduino Uno имеет один *аппаратный* последовательный порт, подключенный к порту USB и, "параллельно", к выводам **0** (RX) и **1** (TX). Arduino Mega имеет три дополнительных последовательных порта.

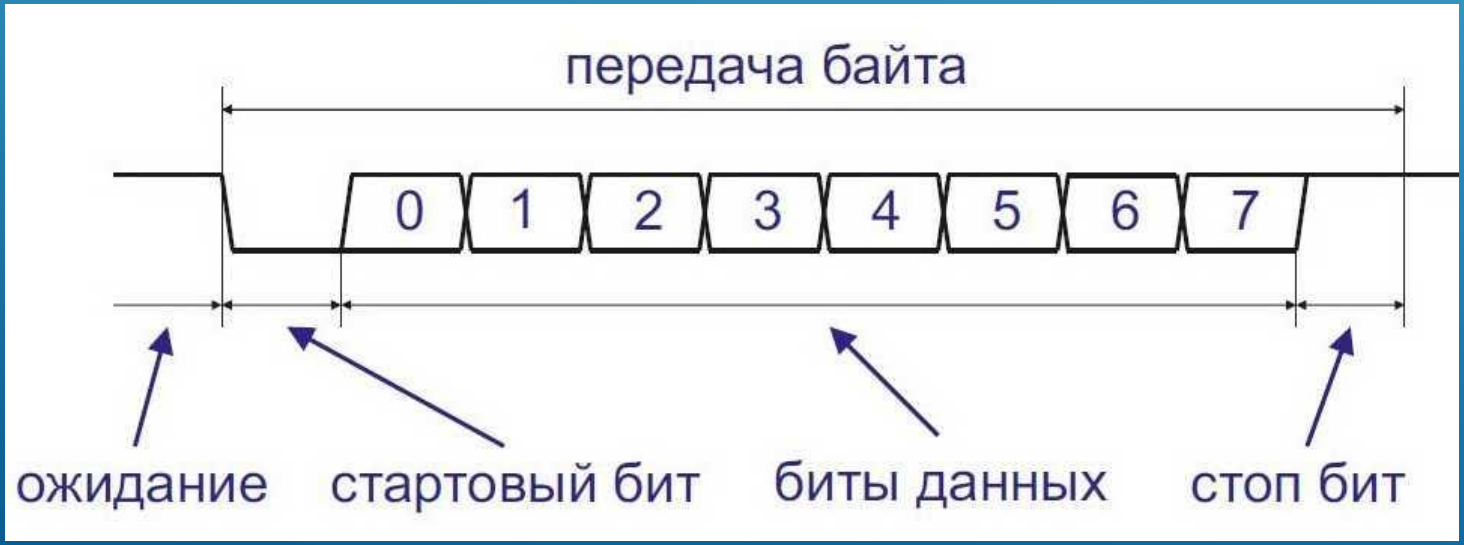


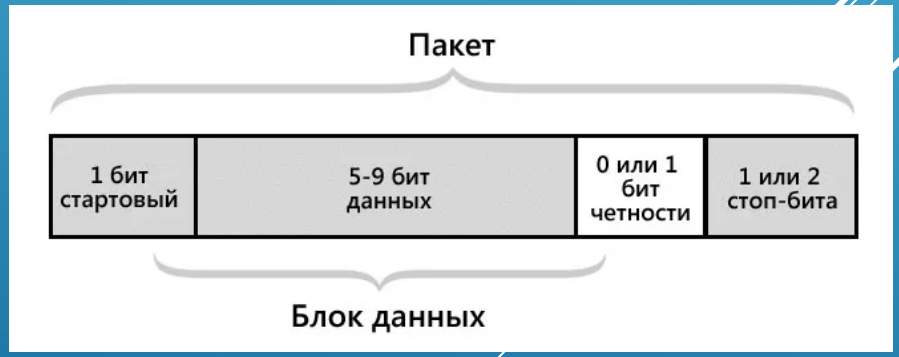



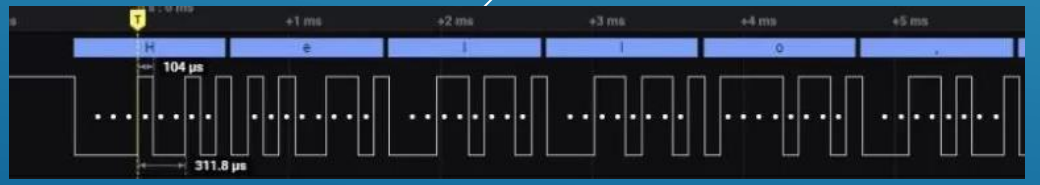

Скорости передающего и принимающего устройств должны быть одинаковы.

# ПРОТОКОЛ ОБМЕНА ДАННЫМИ ДЛЯ UART

- При отсутствии передачи линия удерживается в состоянии логической единицы (в случае Arduino это 5 вольт, т.е. $V_{CC}$). Как только передающее устройство притягивает линию к 0 (GND или 0 вольт в случае Arduino, старт-бит), это сигнализирует принимающему устройству о том что сейчас будет передача данных.
- При появлении старт-бита на линии принимающее устройство начинает отсчитывать время в соответствии с установленной скоростью и считывать состояния линии через определённые промежутки времени в соответствии с предварительно установленным количеством бит данных в кадре. Каждый бит передается за равные промежутки времени. Время передачи одного бита определяется скоростью передачи.
- По завершении передачи данных принимающее устройство ожидает стоп-бит, который должен быть на уровне логической единицы. Если по завершении кадра удерживается логический ноль – значит, данные неверны.



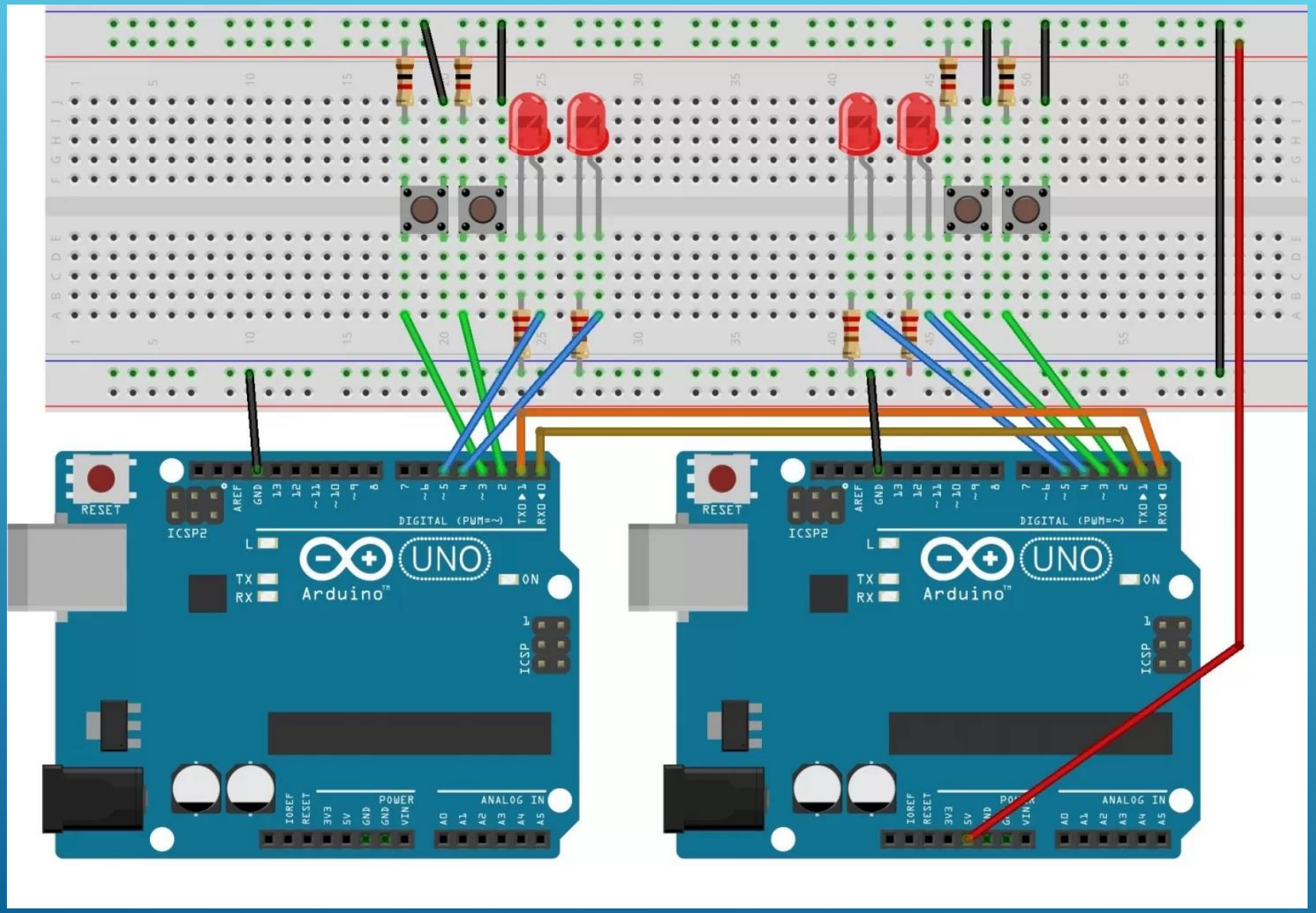



# БИБЛИОТЕКА SERIAL

- Для работы с аппаратными UART-контроллерами в Arduino существует встроенный класс serial. Он предназначен для управления обменом данными через порт UART.
- **Serial.begin(long)** - Запускает работу порта с заданной в параметре скоростью в бодах.
- **Serial.end()** - Останавливает работу порта, если он был ранее запущен. На практике используется редко, но бывают случаи, когда необходимо на время освободить пины 0 и 1.
- **Serial.available()** - Возвращает в виде числа int количество принятых в буфер порта байт. Если возвращает 0, информации не поступало. Обычно используется как триггер для приема информации.
- **Serial.read()** - Возвращает один байт из буфера приема. Следующий вызов возвращает следующий байт и так далее. Если буфер опустел, возвращает 0xFFFF.
- **Serial.print(xxx)** - Различные варианты передачи данных в порт, от байта до строки символов и числа с плавающей точкой. **Serial.println(xxx)** отличается отправкой двух служебных символов переноса строки после информации из параметра. Следующее сообщение начнётся с новой строки. *Удобны для отладки программ*.
- **Serial.write(xxx)** - Передает двоичные данные в порт. Возвращает число переданных байтов.
- **Serial.read()** - Возвращает принятые через порт двоичные данные.

# ПРИМЕР СОЕДИНЕНИЯ ДВУХ ПЛАТ ARDUINO ЧЕРЕЗ UART




Кнопки, подключенные к одной плате, будут управлять светодиодами, подключенными к другой. И наоборот. Для этого каждая плата должна передавать информацию о том, что происходит на ее кнопках другой плате, одновременно принимая от нее такие же данные и управляя, согласно принятым данным, своими светодиодами. Сборки симметричны, функции тоже, значит и программы на обеих платах будут одинаковые.

# ПРОГРАММА УПРАВЛЕНИЯ



```
#define LED_1   4  // светодиод 1
#define LED_2   5  // светодиод 2
#define BUT_1   2  // кнопка 1
#define BUT_2   3  // кнопка 2

byte state;
// отслеживание состояния кнопок

void setup() {
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(BUT_1, INPUT);
  pinMode(BUT_2, INPUT);
  Serial.begin(9600);
}
```

```
void loop() {
  if (get_state()) { // с кнопками что-то было?
    Serial.write(state); // отправляем в порт
  }
  if (Serial.available() > 0) { // что-то пришло
    byte incom = Serial.read(); // считываем данные
    digitalWrite(LED_1, !(incom % 2)); // светодиод 1
    digitalWrite(LED_2, !(incom / 2)); // светодиод 2
  }
}

byte get_state() {
  byte oldstate = state;
  state = digitalRead(BUT_1) + 2 * digitalRead(BUT_2);
  // данные с обеих кнопок закодированы в одну переменную
  if (state != oldstate) return true;
  return false;
}
```

# СПАСИБО ЗА ВНИМАНИЕ!