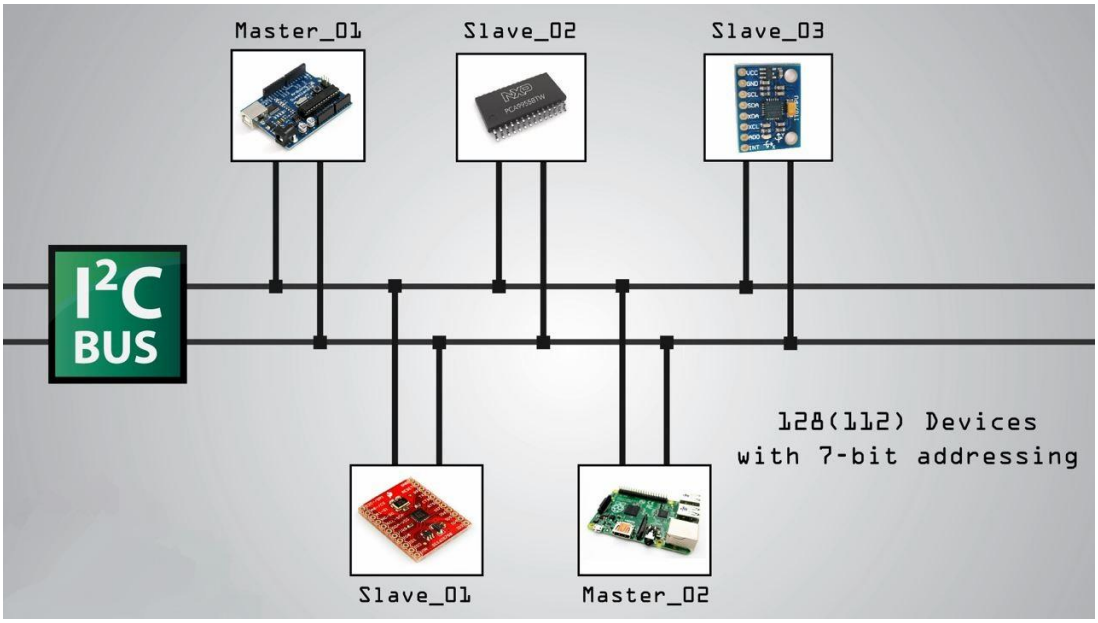
I2C Communication and how to use it with Arduino

The Inter-integrated Circuit (I^2C) Protocol is a protocol intended to allow multiple “slave” digital integrated circuits (“chips”) to **communicate** with one or more “master” chips.



Construction:

The I2C bus consists of two main lines:

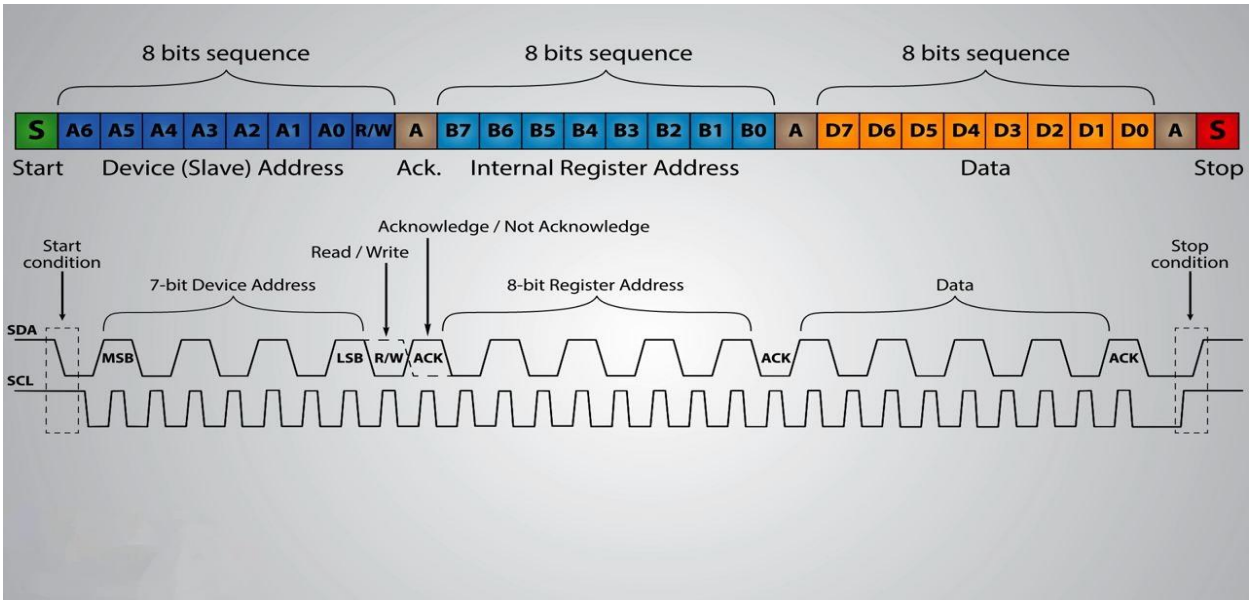
1. SDA (Serial Data line).
2. SCL (Serial Clock line).

A Common Ground Wire and Two resistors (The value of the resistors is not critical.) are also needed.

Working:

The Clock line is used to synchronize all the data transfer between the devices. Only the master devices can initiate a transfer. The data line is used to transfer data using a specified protocol. Each device that can communicate through I2C protocol comes with a pre-defined ID which can be used to identify it from the many devices connected in parallel to the bus. With 7-bit addressing, you can connect as many as 128 devices, however it is better to just stop at 112. If you want to connect more devices then it’s better to use 10-bit addressing, which can handle numbers up to 1024, hence able to connect to that many devices at one.

We need the pull-up resistors on both SDA and SCL because the devices are Active-Low and all the devices must have a common ground in order to communicate properly.



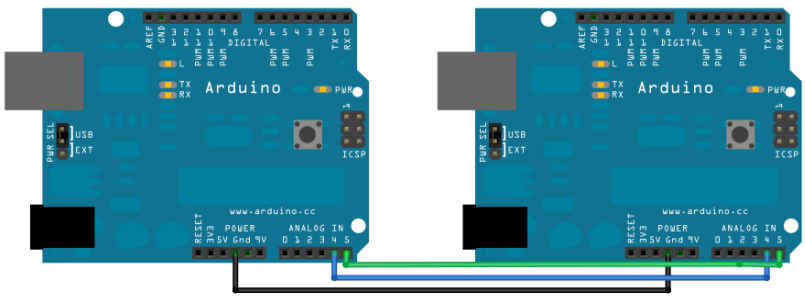
Protocol:

The first thing that will happen is that the master will send out a start sequence. This will alert all the slave devices on the bus that a transaction is starting and they should listen in incase it is for them. Next the master will send out the device address. The slave that matches this address will continue with the transaction, any others will ignore the rest of this transaction and wait for the next. Having addressed the slave device the master must now send out the internal location or register number inside the slave that it wishes to write to or read from. The master will now be able to send or receive data from the required location in sequences of 8 bits which is followed by an acknowledge pin which tells whether the data is received/sent properly or not. This is followed by an end transmission condition.

Arduino Master/Slave configuration:

An Arduino can easily utilize this protocol with the help of **Wire.h** library. Make sure that the required library is installed on your Arduino Software and make the following connections because on Arduino Uno

- SDA pin is AnalogPin-4 (A4)
- SCL pin is AnalogPin-5 (A5)



Code:

The following code for the **master Arduino** requests a device with the ID number 8 to send data consisting of 6 bytes which it will read and display on the Serial Monitor.

```
#include <Wire.h>
void setup() {
  Wire.begin();       // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
}
void loop() {
  Wire.requestFrom(8, 6);    // request 6 bytes from slave device #8
  while (Wire.available()) { // slave may send less than requested
    char c = Wire.read(); // receive a byte as character
    Serial.print(c);        // print the character
  }
```

The following code for the **slave Arduino** gives this device the ID number 8. Then it initializes the request event function such that it will send “hello “ (6-characters) on the data line.

```
#include <Wire.h>
void setup() {
  Wire.begin(8);              // join i2c bus with address #8
  Wire.onRequest(requestEvent); // register event
}
void loop() {
  delay(100);
}
// this function executes whenever data is requested by master, it is registered as an event, see setup()
void requestEvent() {
  Wire.write("hello "); // respond with message of 6 bytes as expected by master
}
```

For Further Studies:

http://howtomechatronics.com/tutorials , http://www.robot-electronics.co.uk & https://www.arduino.cc/en/Tutorial