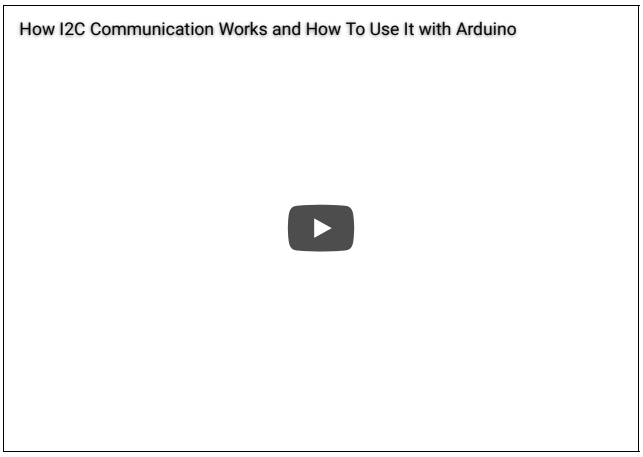
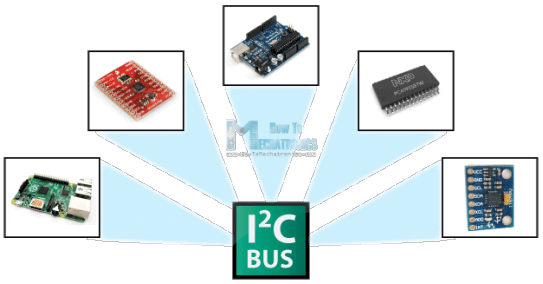
How I2C Communication Works and How To Use It with Arduino

In this tutorial we will learn how the I2C communication protocol works and also we will make a practical example of it with the Arduino Board and a sensor which uses this protocol. You can watch the following video or read the written tutorial below.



Overview

The I2C communication bus is very popular and broadly used by many electronic devices because it can be easily implemented in many electronic designs which require communication between a master and multiple slave devices or even multiple master devices. The easy implementations comes with the fact that only two wires are required for communication between up to almost 128 (112) devices when using 7 bits addressing and up to almost 1024 (1008) devices when using 10 bits addressing.

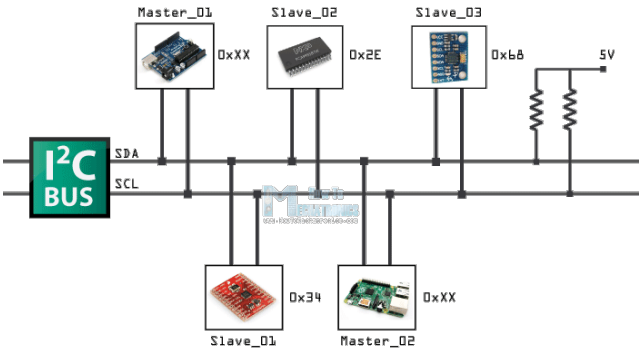


How It Works

How is it possible, a communication between so many devices with just to wires? Well each device has a preset ID or a unique device address so the master can choose with which devices will be communicating.

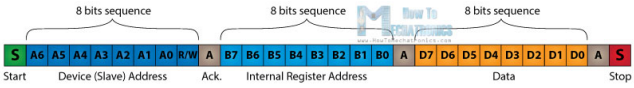
The two wires, or lines are called Serial Clock (or SCL) and Serial Data (or SDA). The SCL line is the clock signal which synchronize the data transfer between the devices on the I2C bus and it's generated by the master device. The other line is the SDA line which carries the data.

The two lines are "open-drain" which means that pull up resistors needs to be attached to them so that the lines are high because the devices on the I2C bus are active low. Commonly used values for the resistors are from 2K for higher speeds at about 400 kbps, to 10K for lower speed at about 100 kbps.



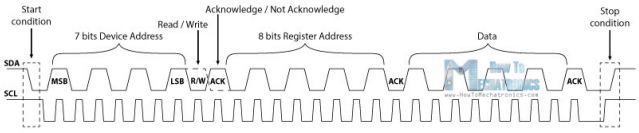
Protocol

The data signal is transferred in sequences of 8 bits. So after a special start condition occurs comes the first 8 bits sequence which indicates the address of the slave to which the data is being sent. After each 8 bits sequence follows a bit called Acknowledge. After the first Acknowledge bit in most cases comes another addressing sequence but this time for the internal registers of the slave device. After the addressing sequences follows the data sequences as many until the data is completely sent and it ends with a special stop condition.

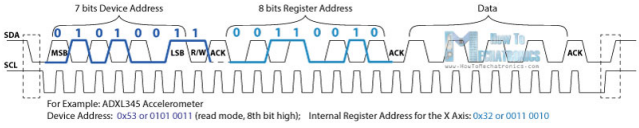


Let's take even closer look at these events. The start condition occurs when data line drops low while the clock line is still high. After this the clock starts and each data bit is transferred during each clock pulse.

The device addressing sequence stars with the most significant bit (MSB) first and ends with the least significant bit (LSB) and it's actually composed of 7 bits because the 8th bit is used for indicating whether the master will write to the slave (logic low) or read from it (logic high).



The next bit AKC/ NACK is used by the slave device to indicate whether it has successfully received the previous sequence of bits. So at this time the master device hands the control of the SDA line over to the slave device and if the slave device has successfully received the previous sequence it will pull the SDA line down to the condition called Acknowledge. If the slave does not pull the SDA line down, the condition is called Not Acknowledge, and means that it didn't successfully received the previous sequence which can be caused by several reasons. For example, the slave might be busy, might not understand the received data or command, cannot receive any more data and so on. In such a case the master device decides how it will proceed.



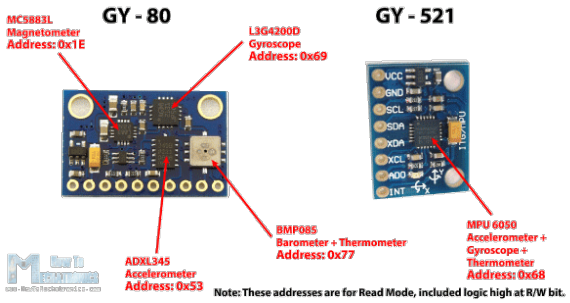
Next is the internal registers addressing. The internal registers are locations in the slave's memory containing various information or data. For example the ADX345 Accelerometer has a unique device address and addition internal registers addresses for the X, Y and Z axis. So if we want to read the data of the X-axis, first we need to send the device address and then the particular internal register address for the X-axis. These addresses can be found from

datasheet of the sensor.

After the addressing, the data transfer sequences begin either from the master or the slave depending of the selected mode at the R/W bit. After the data is completely sent, the transfer will end with a stop condition which occurs when the SDA line goes from low to high while the SCL line is high.

I2C and Arduino

As an example I will use the GY-80 breakout board which consists 5 different sensors and the GY-521 breakout board which consists 3 different sensors. So we can get data from 8 different sensors with just two wires with the I2C bus.

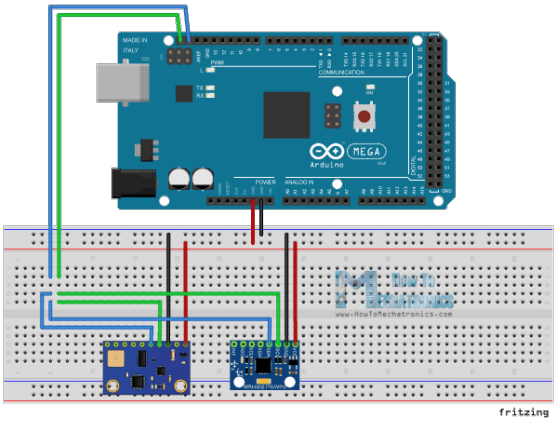


You can purchase these components from ICStation, a recommended retailer by HowToMechatronics:

- ADXL345 3-Axis Accelerator..... Amazon / Aliexpress
- L3G4200D 3-Axis Gyroscope..... Amazon / Aliexpress
- HMC5883L 3-Axis Electronic Compass..... Amazon / Aliexpress
- 3 in 1: GY-80 9-Axis BMP085 3-Axis Magnetic Field Acceleration Gyroscope..... Amazon / Aliexpress
- 2 in 1: MPU6050 6-Axis Gyroscope & Accelerometer ..... Amazon / Aliexpress

*Please note: These are affiliate links. I may make a commission if you buy the components through these links.
I would appreciate your support in this way!*

Here's how we will connect the boards. The Serial Clock pin of the Arduino Board will be connected to the Serial Clock pins of the two breakout boards, the same goes for the Serial Data pins and we will power the boards with the Gnd and the 5V pin from the Arduino Board. Note here we are not using pull-up resistors because the breakout boards already have.



Now in order to communicate with these chips or sensors we need to know their unique addresses. We can find them from the datasheets of the sensors. For the GY-80 breakout board we have the following 4 addresses: a hexadecimal 0x53 for the 3 Axis Accelerometer sensor, a hexadecimal 0x69 for the 3 Axis Gyro, a hexadecimal 0x1E for the 3 Axis Magnetometer and a hexadecimal 0x77 for the Barometer and Thermometer sensor. For the GY-521 breakout board we have only one address and that's a hexadecimal 0x68. We can also get or check the addresses using the I2C Scanner sketch which can be found from the Arduino official website. So here if we upload and run that sketch, we will get the addresses of the connected devices on the I2C bus.

Sensor	Part Number	I2C Address
3 Axis Accelerometer Datasheet	Analog Devices ADXL345	0x53
3 Axis GyroST Datasheet	Microelectronics L3G4200D	0x69
3 Axis Magnetometer Datasheet	Honeywell MC5883L	0x1E
Barometer + Thermometer Datasheet	Bosch BMP085	0x77

After we have found the addresses of the devices we also need to find the addresses of their internal registers in order to read the data from them. For example if we want to read the data for the X axis from the 3 Axis Accelerometer sensor of the GY-80 breakout board, we need to find the internal register address where the data of the X axis is stored. From the datasheet of the sensor, we can see that data for the X axis is actually stored in two registers, DATAX0 with a hexadecimal address 0x32 and DATAX1 with a hexadecimal address 0x33.

Source Code

Now let's make the code that will get the data for the X axis. So we will use the Arduino Wire Library which has to be include in the sketch. Here first we have to define the sensor address and the two internal registers addresses that we previously found. The *Wire.begin()* function will initiate the Wire library and also we need to initiate the serial communication because we will use the Serial Monitor to show the data from the sensor.

In the *loop()* we will start with the *Wire.beginTransmission()* function which will begin the transmission to the particular sensor, the 3 Axis Accelerometer in our case. Then with the *Wire.write()* function we will ask for the particular data from the two registers of the X axis. The *Wire.endTransmission()* will end the transmission and transmit the data from the registers. Now with the *Wire.requestFrom()* function we will request the transmitted data or the two bytes from the two registers. The *Wire.available()* function will return the number of bytes available for retrieval and if that number match with our requested bytes, in our case 2 bytes, using the *Wire.read()* function we will read the bytes from the two registers of the X axis. At the end we will print the data into the serial monitor. Here's that data but keep in mind that this is raw data and some math is needed to be done in order to get the right values of the X axis. You can find more details for that in my next tutorial for using accelerometers with the Arduino Board because I don't want to overload this tutorial because its main goal was to explain how the I2C communication works.

Master Reader/Slave Sender

In some situations, it can be helpful to set up two (or more!) Arduino and Genuino boards to share information with each other. In this example, two boards are programmed to communicate with one another in a Master Reader/Slave Sender configuration via the I2C synchronous serial protocol (http://en.wikipedia.org/wiki/I2C). Several functions of Arduino's **Wire Library** (//www.arduino.cc/en/Reference/Wire) are used to accomplish this. Arduino 1, the Master, is programmed to request, and then read, 6 bytes of data sent from the uniquely addressed Slave Arduino. Once that message is received, it can then be viewed in the Arduino Software (IDE) serial monitor window.

The I2C protocol involves using two lines to send and receive data: a serial clock pin (SCL) that the Arduino or Genuino Master board pulses at a regular interval, and a serial data pin (SDA) over which data is sent between the two devices. As the clock line changes from low to high (known as the rising edge of the clock pulse), a single bit of information - that will form in sequence the address of a specific device and a a command or data - is transferred from the board to the I2C device over the SDA line. When this information is sent - bit after bit -, the called upon device executes the request and transmits it's data back - if required - to the board over the same line using the clock signal still generated by the Master on SCL as timing.

Because the I2C protocol allows for each enabled device to have it's own unique address, and as both master and slave devices to take turns communicating over a single line, it is possible for your Arduino or Genuino board to communicate (in turn) with many devices, or other boards, while using just two pins of your microcontroller.

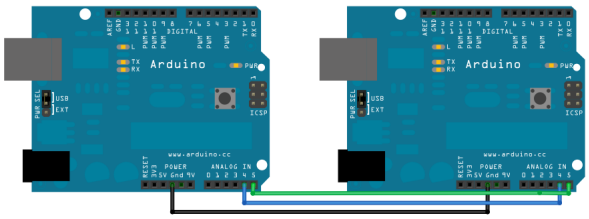
Hardware Required

- 2 Arduino or Genuino Boards
- hook-up wires

Circuit

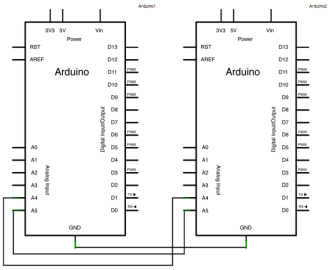
Connect pin 4 (the data, or SDA, pin) and pin 5 (the clock, or SCL, pin) on the master board to their counterparts on the slave board. Make sure that both boards share a common ground. In order to enable serial communication, the master board must be connected to your computer via USB.

If powering the boards independently is an issue, connect the 5V output of the Master to the VIN pin on the slave.



(//www.arduino.cc/en/uploads/Tutorial/Master_Sender_bb.png)
image developed using Fritzing (http://www.fritzing.org). For more circuit examples, see the Fritzing project page (http://fritzing.org/projects/)

Schematic



(//www.arduino.cc/en/uploads/Tutorial/Master_Sender_sch.png)

Code

Code for Master Reader - Program for Arduino 1

```
// Wire Master Reader
// by Nicholas Zambetti <http://www.zambetti.com>

// Demonstrates use of the Wire library
// Reads data from an I2C/TWI slave device
// Refer to the "Wire Slave Sender" example for use with this

// Created 29 March 2006

// This example code is in the public domain.


#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
  }

  delay(500);
}
```

[Get Code] (//www.arduino.cc/en/Tutorial/MasterReader?action=sourceblock&num=1)

Code for Slave Sender - Program for Arduino 2

```
// Wire Slave Sender
// by Nicholas Zambetti <http://www.zambetti.com>

// Demonstrates use of the Wire library
// Sends data as an I2C/TWI slave device
// Refer to the "Wire Master Reader" example for use with this

// Created 29 March 2006

// This example code is in the public domain.

#include <Wire.h>

void setup() {
  Wire.begin(8);                // join i2c bus with address #8
  Wire.onRequest(requestEvent); // register event
}

void loop() {
  delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent() {
  Wire.write("hello "); // respond with message of 6 bytes
  // as expected by master
}
```

[Get Code] (//www.arduino.cc/en/Tutorial/MasterReader?action=sourceblock&num=2)

See Also

- Wire.begin (//www.arduino.cc/en/Reference/WireBegin)()
- Wire.RequestFrom (//www.arduino.cc/en/Reference/WireRequestFrom)()
- Wire.receive (//www.arduino.cc/en/Reference/WireReceive)()
- Wire.send (//www.arduino.cc/en/Reference/WireSend)()
- Wire.onRequest (//www.arduino.cc/en/Reference/WireOnRequest)()
- Wire Library (//www.arduino.cc/en/Reference/Wire) – Your reference for the Wire Library.
- DigitalPotentiometer (//www.arduino.cc/en/Tutorial/DigitalPotentiometer) - How to control an Analog Devices AD5171 Digital Potentiometer.
- SFRRanger_reader (//www.arduino.cc/en/Tutorial/SFRRangerReader) - How to read an ultra-sonic range finder interfaced via the I2C.
- Master Writer/Slave receiver (//www.arduino.cc/en/Tutorial/MasterWriter) - Two Arduino are programmed to communicate in a Master Writer/Slave Receiver configuration via the I2C.

Last revision 2018/05/17 by SM

Master Writer/Slave Receiver

Sometimes, the folks in charge just don't know when to shut up! In some situations, it can be helpful to set up two (or more!) Arduino or Genuino boards to share information with each other. In this example, two boards are programmed to communicate with one another in a Master Writer/Slave Receiver configuration via the I2C synchronous serial protocol (http://en.wikipedia.org/wiki/I2C). Several functions of Arduino's **Wire Library** (//www.arduino.cc/en/Reference/Wire) are used to accomplish this. Arduino 1, the Master, is programmed to send 6 bytes of data every half second to a uniquely addressed Slave. Once that message is received, it can then be viewed in the Slave board's serial monitor window opened on the USB connected computer running the Arduino Software (IDE).

The I2C protocol involves using two lines to send and receive data: a serial clock pin (SCL) that the Arduino or Genuino Master board pulses at a regular interval, and a serial data pin (SDA) over which data is sent between the two devices. As the clock line changes from low to high (known as the rising edge of the clock pulse), a single bit of information - that will form in sequence the address of a specific device and a a command or data - is transferred from the board to the I2C device over the SDA line. When this information is sent - bit after bit -, the called upon device executes the request and transmits it's data back - if required - to the board over the same line using the clock signal still generated by the Master on SCL as timing. The initial eight bits (i.e. eight clock pulses) from the Master to Slaves contain the address of the device the Master wants data from. The bits after contain the memory address on the Slave that the Master wants to read data from or write data to, and the data to be written, if any.

Each Slave device has to have its own unique address and both master and slave devices need to take turns communicating over a the same data line line. In this way, it's possible for your Arduino or Genuino boards to communicate with many device or other boards using just two pins of your microcontroller, using each device's unique address.

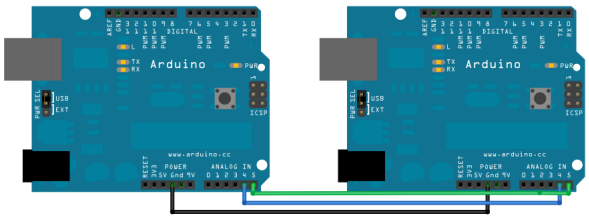
Hardware Required

- 2 Arduino or Genuino Boards
- hook-up wires

Circuit

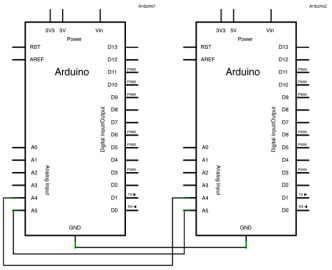
Connect pin 5 (the clock, or SCL, pin) and pin 4 (the data, or SDA, pin) on the master Arduino to their counterparts on the slave board. Make sure that both boards share a common ground. In order to enable serial communication, the slave Arduino must be connected to your computer via USB.

If powering the boards independently is an issue, connect the 5V output of the Master to the VIN pin on the slave.



(//www.arduino.cc/en/uploads/Tutorial/Master_Sender_bb.png)
image developed using Fritzing (http://www.fritzing.org). For more circuit examples, see the Fritzing project page (http://fritzing.org/projects/)

Schematic



(//www.arduino.cc/en/uploads/Tutorial/Master_Sender_sch.png)

Code

Master Writer Code - Program for Arduino 1

```
// Wire Master Writer
// by Nicholas Zambetti <http://www.zambetti.com>

// Demonstrates use of the Wire library
// Writes data to an I2C/TWI slave device
// Refer to the "Wire Slave Receiver" example for use with this

// Created 29 March 2006

// This example code is in the public domain.


#include <Wire.h>

void setup() {
  Wire.begin(); // join i2c bus (address optional for master)
}

byte x = 0;

void loop() {
  Wire.beginTransmission(8); // transmit to device #8
  Wire.write("x is ");        // sends five bytes
  Wire.write(x);              // sends one byte
  Wire.endTransmission();    // stop transmitting

  x++;
  delay(500);
}
```

[Get Code] (//www.arduino.cc/en/Tutorial/MasterWriter?action=sourceblock&num=1)

Slave Receiver Code - Program for Arduino 2

```
// Wire Slave Receiver
// by Nicholas Zambetti <http://www.zambetti.com>

// Demonstrates use of the Wire library
// Receives data as an I2C/TWI slave device
// Refer to the "Wire Master Writer" example for use with this

// Created 29 March 2006

// This example code is in the public domain.

#include <Wire.h>

void setup() {
  Wire.begin(8);                // join i2c bus with address #8
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600);           // start serial for output
}

void loop() {
  delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany) {
  while (1 < Wire.available()) { // loop through all but the last
    char c = Wire.read(); // receive byte as a character
    Serial.print(c);         // print the character
  }
  int x = Wire.read();    // receive byte as an integer
  Serial.println(x);         // print the integer
}
```

[Get Code] (//www.arduino.cc/en/Tutorial/MasterWriter?action=sourceblock&num=2)

See Also

- Wire.begin (//www.arduino.cc/en/Reference/WireBegin)()
- Wire.beginTransmission (//www.arduino.cc/en/Reference/WireBeginTransmission)()
- Wire.endTransmission (//www.arduino.cc/en/Reference/WireEndTransmission)()
- Wire.send (//www.arduino.cc/en/Reference/WireSend)()
- Wire.OnReceive (//www.arduino.cc/en/Reference/WireOnReceive)()
- Wire.available (//www.arduino.cc/en/Reference/WireAvailable)()
- Wire Library (//www.arduino.cc/en/Reference/Wire) – Your reference for the Wire Library.
- DigitalPotentiometer (//www.arduino.cc/en/Tutorial/DigitalPotentiometer) - How to control an Analog Devices AD5171 Digital Potentiometer.
- SFRRanger_reader (//www.arduino.cc/en/Tutorial/SFRRangerReader) - how to read an ultra-sonic range finder interfaced via the I2C.
- Master Reader/Slave Writer (//www.arduino.cc/en/Tutorial/MasterReader) - Two Arduino are programmed to communicate with one another in a Master Reader/Slave Sender configuration