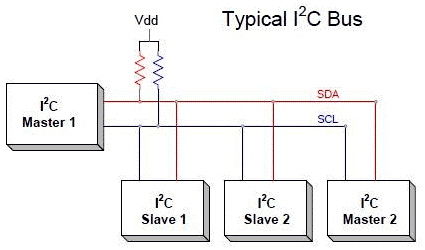
**I2C in Arduino**

Comparing I2C with SPI, I2C has only two wires while SPI uses four and I2C can have Multiple Master and Slave, while SPI can have only one master and multiple slaves. So there are more than one microcontroller in a project that need to be masters then I2C is used. I2C communication is generally used to communicate with Gyroscope, accelerometer, barometric pressure sensors, LED displays etc.

**What is I2C Communication Protocol?**

The term IIC stands for “**Inter Integrated Circuits**”. It is normally denoted as I2C or I squared C or even as 2-wire interface protocol (TWI) at some places but it all means the same. I2C is a synchronous communication protocol meaning, both the devices that are sharing the information must share a common clock signal. It has only two wires to share information out of which one is used for the cock signal and the other is used for sending and receiving data.



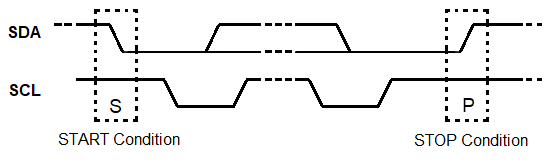
The complete communication takes place through these two wires namely, Serial Clock (SCL) and Serial Data (SDA).

**Serial Clock (SCL):**Shares the clock signal generated by the master with the slave

**Serial Data (SDA):** Sends the data to and from between the Master and slave.

The **voltage levels of I2C are not predefined**. I2C communication is flexible, means the device which is powered by 5v volt, can use 5v for I2C and the 3.3v devices can use 3v for I2C communication. But what if two devices which are running on different voltages, need to communicate using I2C? A**5V I2C bus can’t be connected with 3.3V device**. In this case voltage shifters are used to match the voltage levels between two I2C buses.

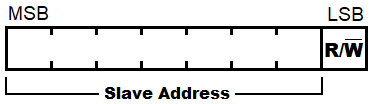
There are some set of conditions which frame a transaction. Initialization of transmission begins with a falling edge of SDA, which is defined as ‘START’ condition in below diagram where master leaves SCL high while setting SDA low.



As shown in the above diagram below,

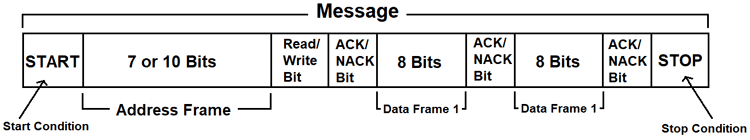
The falling edge of SDA is the hardware trigger for the START condition. After this all devices on the same bus go into listening mode.

In the same manner, rising edge of SDA stops the transmission which is shown as ‘STOP’ condition in above diagram, where the master leaves SCL high and also releases SDA to go HIGH. So rising edge of SDA stops the transmission.



R/W bit indicates the direction of transmission of following bytes, if it is HIGH means the slave will transmit and if it is low means the master will transmit.

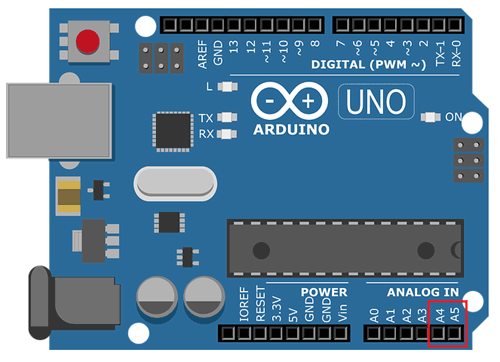
Each bit is transmitted on each clock cycle, so it takes 8 clock cycles to transmit a byte. After each byte either sent or received, ninth clock cycle is held for the ACK/NACK (acknowledged/not acknowledged). This ACK bit is generated by either slave or master depending upon the situation. For ACK bit, SDA is set to low by master or slave at 9th clock cycle. So it is low it considered as ACK otherwise NACK.



* I2C communication is used only for **short-distance communication**.
* For long-range communication, you should try **RS232.**
* For more reliable communication you should try the **SPI protocol.**

**I2C in Arduino**

**The image below shows the I2C pins present in Arduino UNO.**

****

|  |  |
| --- | --- |
| **I2C Line** | **Pin in Arduino** |
| **SDA** | **A4** |
| **SCL** | **A5** |

**Wire library**

The **library <Wire.h>** is included in the program for using the following functions for I2C communication.

**1. Wire.begin(address):**

**Use:**  This library is used for making communication with I2C devices. This Initiate the Wire library and join the I2C bus as a master or slave.

Address: The 7-bit slave address is optional and if the address is not specified, it joins the bus as a master like this [Wire.begin()].

**2. Wire.read():**

**Use:** This function is used to read a byte that was received from master or slave device, either that was transmitted from a slave device to a master device after a call to *requestFrom()* or was transmitted from a master to a slave.

**3. Wire.write():**

**Use:** This function is used to write data to a slave or master device.

**Slave to Master:**Slave writes data to a master when *Wire.RequestFrom()* is used in master.

**Master to Slave:**For transmission from a master to slave device *Wire.write()* is used in-between calls to *Wire.beginTransmission()* and *Wire.endTransmission().*

**Wire.write()** can be written as:

* Wire.write(value)

value: a value to send as a single byte.

* Wire.write(string) :

string: a string to send as a series of bytes.

* Wire.write(data, length):

data: an array of data to send as bytes

length: the number of bytes to transmit.

**4. Wire.beginTransmission(address):**

**Use:**This function is used to begin a transmission to the I2C device with the given slave address. Subsequently, build queue of bytes for transmission with the *write()* function and then transmit them by calling *endTransmission()* function. 7-bit address of the device is transmitted.

**5. Wire.endTransmission();**

**Use:**This function is used to end a transmission to a slave device that was begun by *beginTransmission()* and transmits the bytes that were queued by *Wire.write().*

**6. Wire.onRequest();**

**Use:**This function gets called when a master requests data using *Wire.requestFrom()* from the slave device. Here we can include *Wire.write()* function to send data to the master.

**7. Wire.onReceive();**

**Use:**This function gets called when a slave device receives a data from a master. Here we can include *Wire.read();* function to read the data sent from master.

**8. Wire.requestFrom(address,quantity);**

**Use:**This function is used in the master to request bytes from a slave device. The function *Wire.read()* is used to read the data sent from the slave device.

address: the 7-bit address of the device to request bytes from

quantity: the number of bytes to request