Documentation for I2C communication via Arduino-Uno

Documented by :

Ranojoy Barua (CST 2nd Year - 510515025)

Sourav Das (CST 2nd Year - 510515012)

Institution :

IIEST Shibpur , Kolkata, INDIA

Documentation for I2C communication.

I^2C was originally developed in 1982 by Philips for various Philips chips.

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. It is only intended for short distance communications within a single device.

I^2C requires a mere two wires, like asynchronous serial, but those two wires can support up to 128 devices in 7-bit communication and in 10-bit communication 1008 slave devices.

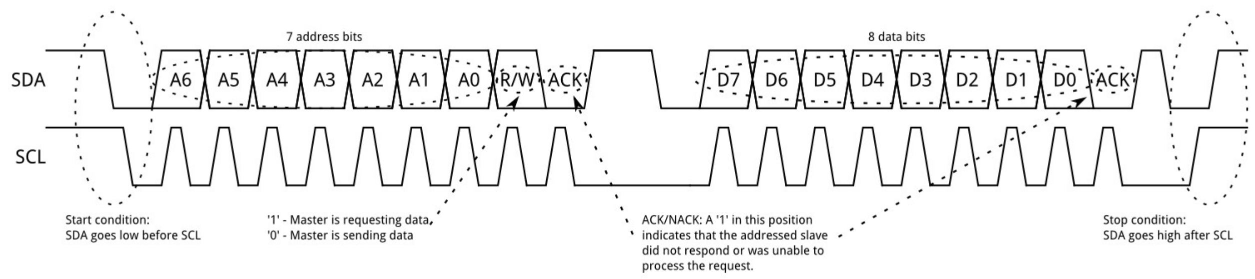
I^2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus (although the master devices can’t talk to each other over the bus and must take turns using the bus lines).

Most I^2C devices can communicate at 100 kHz or 400 kHz.

Protocol

Communication via I^{2}C is more complex than with a UART or SPI solution. The signaling must adhere to a certain protocol for the devices on the bus to recognize it as valid I^{2}C communications. Fortunately, most devices take care of all the fiddly details for you, allowing you to concentrate on the data you wish to exchange.

Basics



7 address bits
8 data bits
SDA
SCL
A6
A5
A4
A3
A2
A1
A0
R/W
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
Start condition: SDA goes low before SCL
'1' - Master is requesting data, '0' - Master is sending data
ACK/NACK: A '1' in this position indicates that the addressed slave did not respond or was unable to process the request.
Stop condition: SDA goes high after SCL

Messages are broken up into two types of frame: an address frame, where the master indicates the slave to which the message is being sent, and one or more data frames, which are 8-bit data messages passed from master to slave or vice versa. Data is placed on the SDA line after SCL goes low, and is sampled after the SCL line goes high. The time between clock edge and data read/write is defined by the devices on the bus and will vary from chip to chip.

Start Condition

To initiate the address frame, the master device leaves SCL high and pulls SDA low. This puts all slave devices on notice that a transmission is about to start. If two master devices wish to take ownership of the bus at one time, whichever device pulls SDA low first wins the race and gains control of the bus. It is possible to issue repeated starts, initiating a new communication sequence without relinquishing control of the bus to other masters; we'll talk about that later.

Address Frame

The address frame is always first in any new communication sequence. For a 7-bit address, the address is clocked out most significant bit (MSB) first, followed by a R/W bit indicating whether this is a read (1) or write (0) operation.

The 9th bit of the frame is the NACK/ACK bit. This is the case for all frames (data or address). Once the first 8 bits of the frame are sent, the receiving device is given control over SDA. If the receiving device does not pull the SDA line low before the 9th clock pulse, it can be inferred that the receiving device either did not receive the data or did not know how to parse the message. In that case, the exchange halts, and it’s up to the master of the system to decide how to proceed.

Data Frames

After the address frame has been sent, data can begin being transmitted. The master will simply continue generating clock pulses at a regular interval, and the data will be placed on SDA by either the master or the slave, depending on whether the R/W bit indicated a read or write operation. The number of data frames is arbitrary, and most slave devices will auto-increment the internal register, meaning that subsequent reads or writes will come from the next register in line.

Stop condition

Once all the data frames have been sent, the master will generate a stop condition. Stop conditions are defined by a 0->1 (low to high) transition on SDA *after* a 0->1 transition on SCL, with SCL remaining high. During normal data writing operation, the value on SDA should **not** change when SCL is high, to avoid false stop conditions.

Applying I2C communication in Arduino

Requirements: 1) Data Line

2) Clock Line

3) Common Power Line

4) Common Ground Line

5) Pull-up registers (2 pcs. Of 4.7k Ohm)

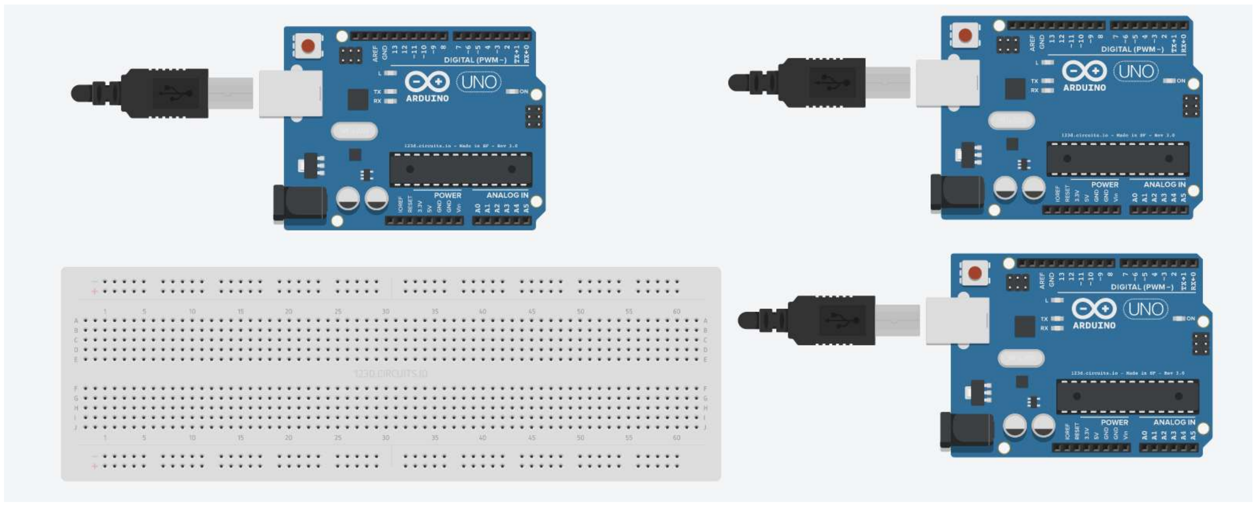
We used Arduino-Uno in this development board (A4 is the Data Line) and (A5 is the Clock Line).

Connection

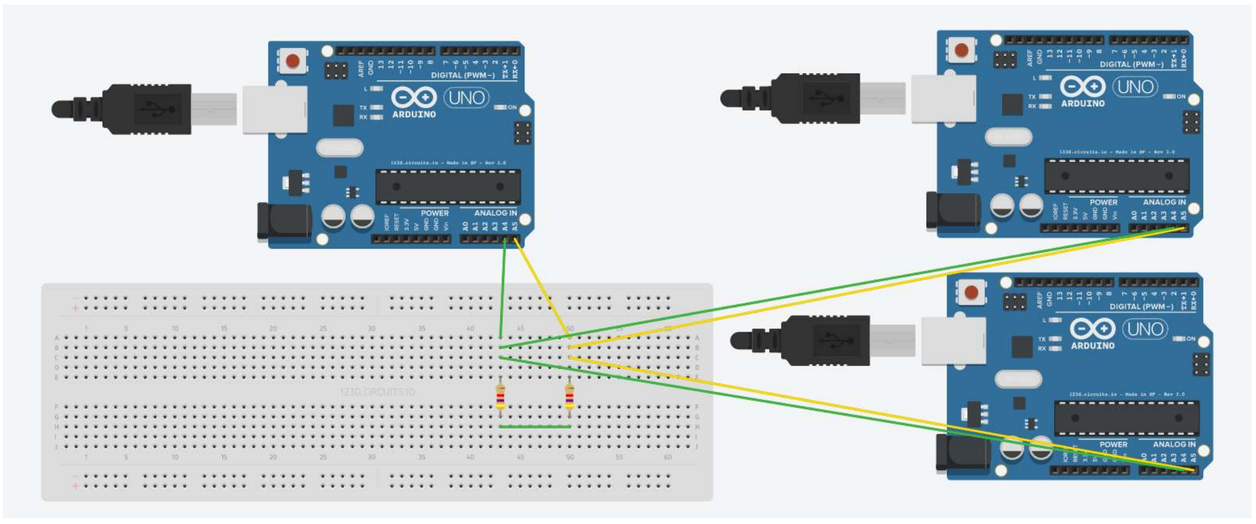
Components : 1) 3-Arduino Uno (1 Master , 2 Slave)

2) Bread Board

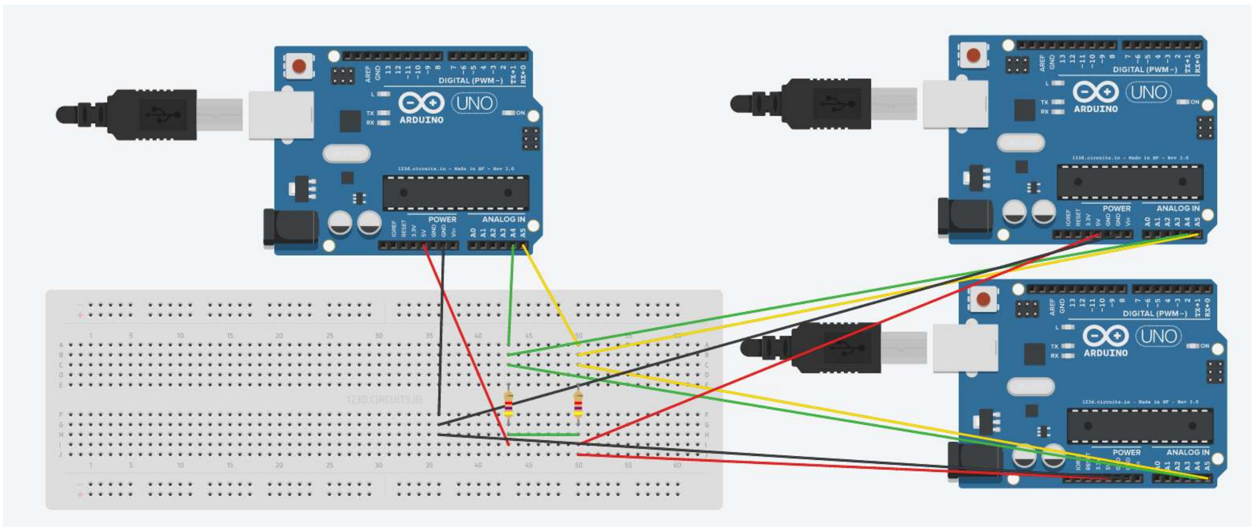
3) 2, 4.7k Ohm registers



Make two common lines for Data and Clock and connect it to all the Arduino (A4 is Data Line and A5 is Clock Line) and connect one end of the 4.7k Ohm registers to each of the Data and Clock line and make a common line for the other ends.



Connect the 5V of the Arduino in the line where the resistors are connected and make a common line for the ground and connect all the grounds of all the Arduino there.



Your connection is ready.

Arduino Code for Slave Side.

```
#include<Wire.h>

int trigPin = 8;  // to be set as required as per sensor

int echoPin = 7;  // to be set as required as per sensor

void setup()

{ /* Using wire library the communication is started when the request is for device numbered 1 from master Using
Wire.begin(x)  we denote the slave number where x holds the slave number. In this case it is one. */

  Wire.begin(1);

/* Setting up pins for sensor in this case it is Ultra-Sound. */

  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

/* The communication in this protocol is event based so an event function is created name requestEvent and invoked     as
required.*/

  Wire.onRequest(requestEvent); }



void loop()  {     delay(500);       }



/* The event function. */

void requestEvent()

{ /* Declsreing the variables. */

 int i;   unsigned  long duration, distance;   char buf[3];

/* Reading data from the sensor. */

  digitalWrite(trigPin,HIGH);

  delayMicroseconds(1000);

  digitalWrite(trigPin, LOW);

  duration=pulseIn(echoPin, HIGH); // only a single device i.e Sonar was connected.

  distance =(duration/2)/29.1;

/* Converting the numeric value to string for transmission through the Data-Bus. */

  sprintf(buf,"%lu",distance);

/* Sending data through the bus. */

  Wire.write(buf);

  delay(1000);   }
```

Arduino Code for Master Side.

```
#include<Wire.h>

void setup()

{ /* Starting wire and serial communication for communicating through I2C data bus and for Serial Communication */

 Wire.begin();

  Serial.begin(9600);   }



void loop()

 {  int i;

/* Running the loop for number of times equal to the total number of slave devices. */

  for(i=1;i<=2;i++)

{ /* Wire.requestFrom(x,y) is used to request data from a specific Slave with a unique number. X denotes the slave number
(it should be equal to the number mentioned in the slave side) and y denotes the number of bits requested. In out case we
requested only 3 bits. */

 Wire.requestFrom(i,3);

/* Reading data from the data bus and printing it in serial monitor. */

  while(Wire.available()){

    char c = Wire.read();

    Serial.print(c);     }



  Serial.println("\n");

  delay(3000);  }
```