Communication: I2C

INTER INTEGRATED CIRCUITS INTERFACE (I2C)

I2C Basics

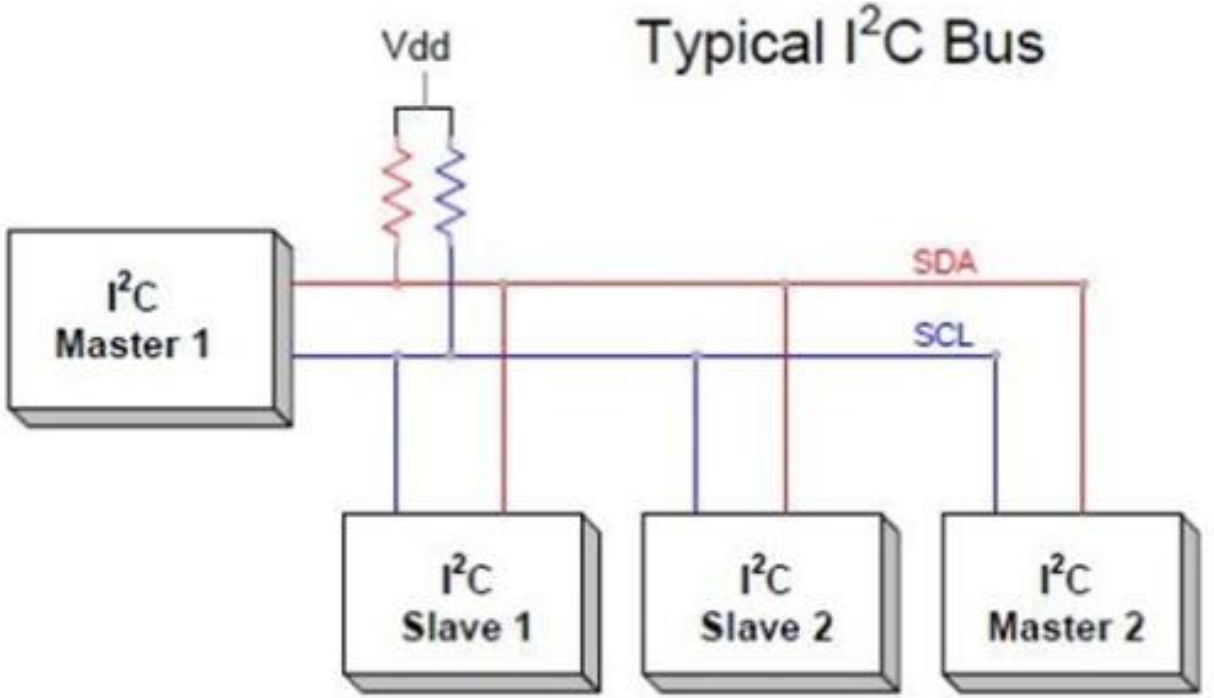
- I2C is a multi-master, multi-slave, serial bus invented by Philips Semiconductor.
- Developed by Philips in late 1980s; Version 1.0 – 1992 (100 kbps and 400 kbps modes); Version 2.0 – 1998 (3.4 Mbps)
- Devices are classified as Master and Slave for communication
- Allows multiple Masters / Multiple Slaves on same bus
- It requires two lines per bus; one clock and one data for all the connected devices.
- The lines are High when floating (not selected) and Low when driven (accessed by device)
- SDA (serial data line) and SCL (serial clock line) pins need external pull up resistors

I2C Basics

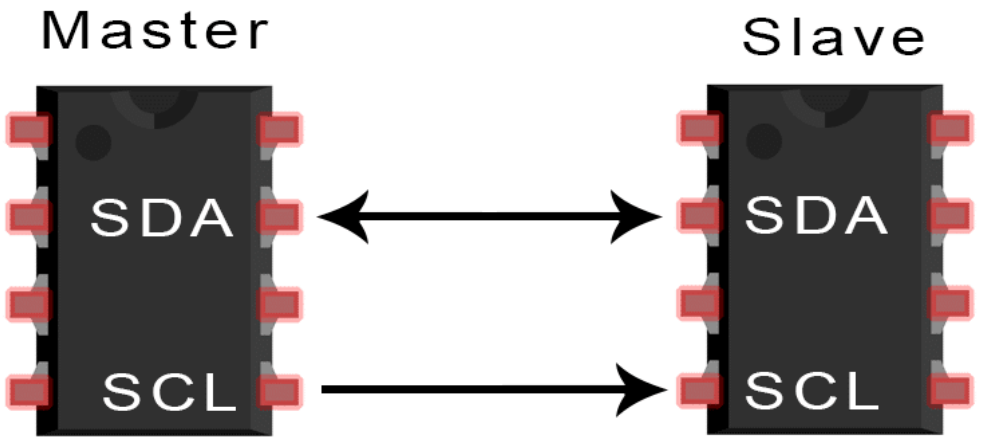
- A slower communication speed is the problem of I2C compared to other communication methods.
- Lines can be pulled low by any device for communication, and are high when all devices release them.
- No “slave-select” lines – the devices have “addresses” that are sent as a part of the transmission protocol.
- Four max speeds (100 kbS (standard), 400 kbS (fast), 1 MbS (fast plus), and 3.4 MbS (high-speed)
- Maximum number of Slaves is 1008

What is I2C

- Bidirectional 2-wire bus
- Serial Interface
- Half Duplex
- Synchronous



Control of Master



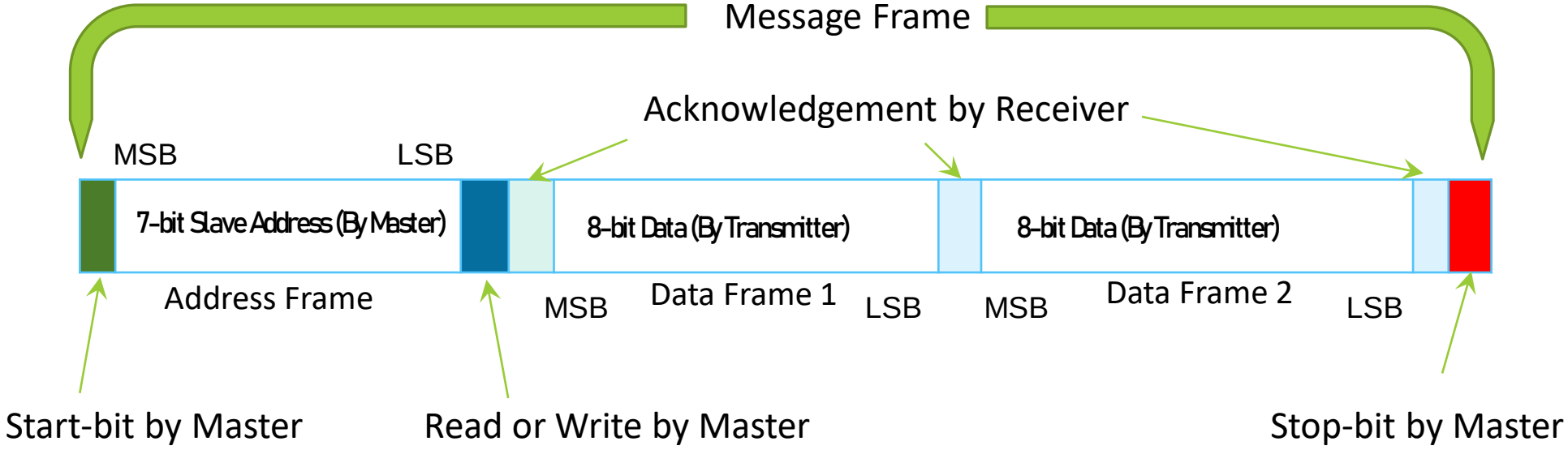
SDA (Serial Data) – The line for the master and slave to send and receive data.
SCL (Serial Clock) – The line that carries the clock signal.

I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line).

I2C Bus

- Two bi-directional lines – SDA (Serial Data line) and SCL (Serial Clock line)
- Multi-master bus – Any device on bus can initiate data transfer
- Device initiating communication is termed as Master and the device being addressed by Master is termed as Slave
- Clock is generated by Master
- It has a 7-bit address space with 16 reserved addresses, maximum 112 number of nodes can be made

Master to Slave Communication



Data is transferred in *messages.* Messages are broken up into *frames* of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame

Start Condition: The SDA line switches from a high voltage level to a low voltage level *before* the SCL line switches from high to low.

Stop Condition: The SDA line switches from a low voltage level to a high voltage level *after* the SCL line switches from low to high.

Address Frame: A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.

Read/Write Bit: A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

ACK/NACK Bit: Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

Master to Slave Communication

1. When the I2C bus has no activity (SDA and SCL – both High) Master acts
2. To inform other devices that is has acted, Master issues a start condition. Then Slave devices listen on SA for instructions / data.
3. Clock signal is provided by Master on SCL line
4. Unique binary address is sent by Master to select the required device. (at a time only one device can be selected for communication)
5. One bit message is put by Master on bus to indicate whether it wants to SEND data to Slave or RECEIVE data from Slave
6. Selected Slave device responds with acknowledgement
7. Master transmits or receives 8-bit data to or from the receiver and receiver sends 1-bit acknowledgement
8. After completion of the communication, Master issues stop condition and frees the bus

Merits of I2C

- Only 2 wires are required
- Suitable for boards with multiple devices connected on a bus
- Cost and complexity gets reduced
- Supports multiple masters and multiple slaves
- ACK/NACK bit gives confirmation that each frame is transferred successfully
- Well known and widely used protocol

Demerits of I2C

- Addressing and acknowledgement is to be done neatly since only 2 wires are present
- In simple configuration it may not be useful
- Slower data transfer rate than SPI
- The size of the data frame is limited to 8 bits

I2C in Arduino

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.

I2C in Arduino

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.

I2C in Arduino

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.

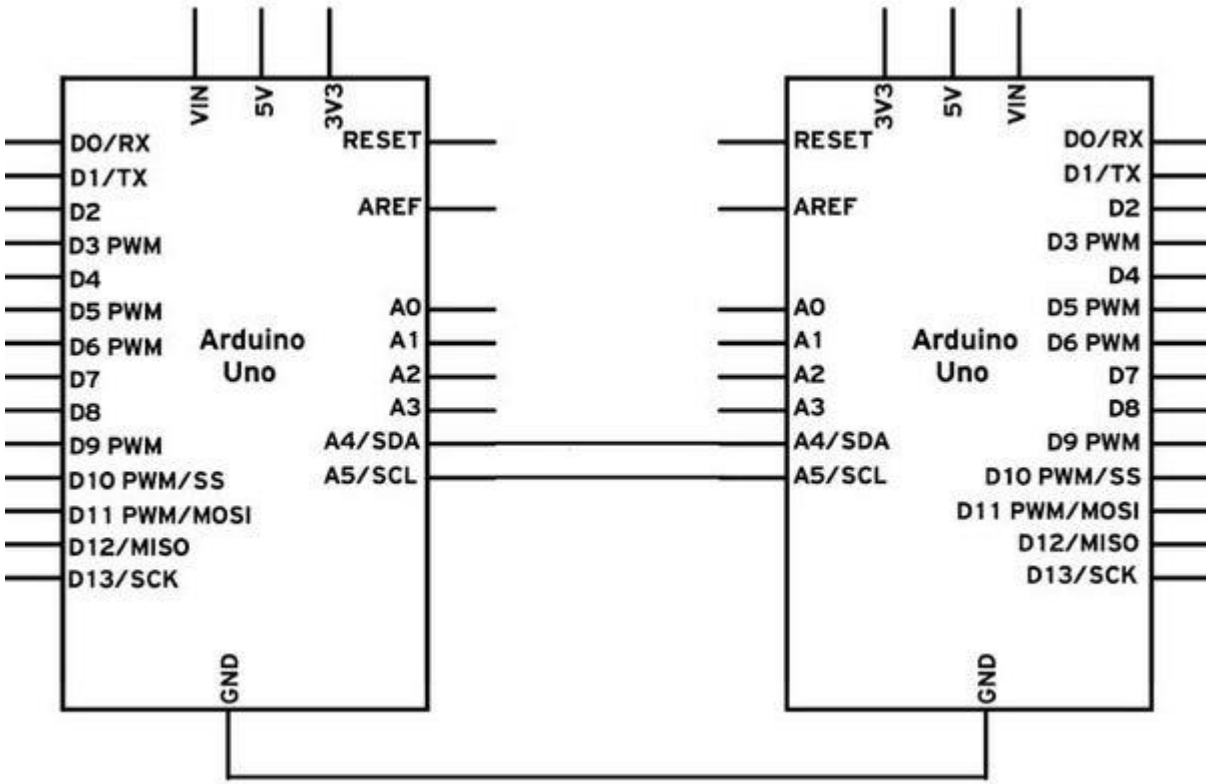
address: the 7-bit address of the device to request bytes from

quantity: the number of bytes to request

Arduino – Arduino Interface

Connect SDA and SCL of both

Connect ground of both together



Arduino Uno : Pins A4 for SDA (data) and A5 for SCL (clock):

Arduino Mega, SDA is pin 20 and SCL is 21,

DIP ATmega328-PU microcontroller, pins 27 for SDA and 28 for SCL.

Master Code

```
// Include the required Wire library for I2C<br>#include
int x = 0;
void setup() {
  // Start the I2C Bus as Master
  Wire.begin();
}
void loop() {
  Wire.beginTransmission(9); // transmit to device #9
  Wire.write(x);             // sends x
  Wire.endTransmission();    // stop transmitting
  x++; // Increment x
  if (x > 5) x = 0; // `reset x once it gets 6
  delay(500);
}
```

Slave Code

```
// Include the required Wire library for I2C<br>#include <Wire.h>
int LED = 13;
int x = 0;
void setup() {
  // Define the LED pin as Output
  pinMode (LED, OUTPUT);
  // Start the I2C Bus as Slave on address 9
  Wire.begin(9);
  // Attach a function to trigger when something is received.
  Wire.onReceive(receiveEvent);
}
void receiveEvent(int bytes) {
  x = Wire.read();    // read one character from the I2C
}
void loop() {
  //If value received is 0 blink LED for 200 ms
  if (x == '0') {
    digitalWrite(LED, HIGH);
    delay(200);
    digitalWrite(LED, LOW);
    delay(200);
  }
  //If value received is 3 blink LED for 400 ms
  if (x == '3') {
    digitalWrite(LED, HIGH);
    delay(400);
    digitalWrite(LED, LOW);
    delay(400);
  }
}
```

Applications

- Real-time clocks,
- Digital potentiometers,
- Temperature sensors,
- Digital compasses,
- Memory chips,
- FM radio circuits,
- I/O expanders,
- LCD controllers,
- Amplifiers,