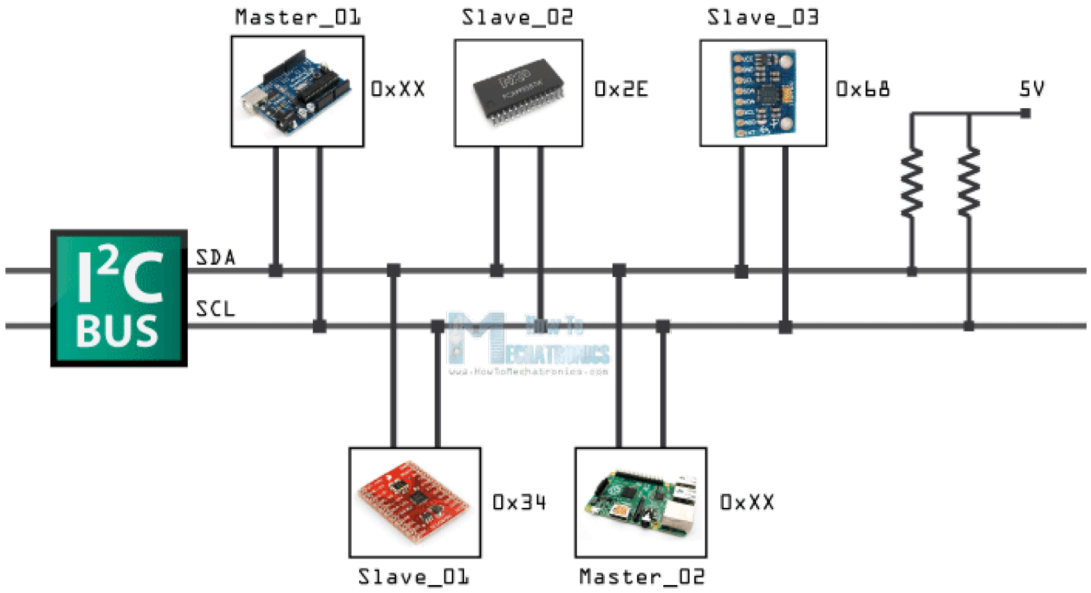
Embedded Systems

Master 's Degree in Informatics Engineering

I2C BUS

Inter-Integrated Circuit



What is I2C?

- The name stands for “Inter - Integrated Circuit Bus”
- A Small Area Network connecting ICs and other electronic systems
- Originally intended for operation on one single board / PCB
  - Synchronous Serial Signal
  - Two wires carry information between a number of devices
  - One wire use for the data
  - One wire used for the clock
- Today, a variety of devices are available with I^2C Interfaces
  - Microcontroller, EEPROM, Real-Timer, interface chips, LCD driver, A/D converter

What is I²C used for?

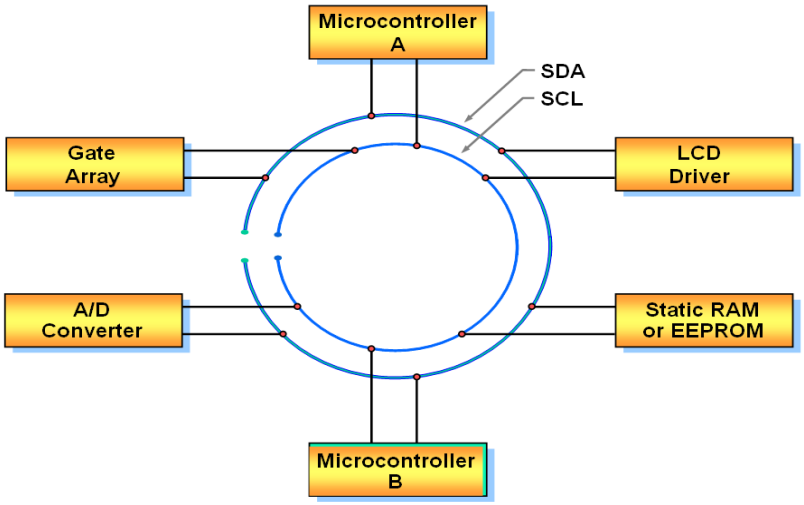
- **Data transfer between ICs and systems at relatively low rates**
  - "Classic" I²C is rated to 100K bits/second
  - "Fast Mode" devices support up to 400K bits/second
  - A "High Speed Mode" is defined for operation up to 3.4M bits/second
- **Reduces Board Space and Cost By:**
  - Allowing use of ICs with fewer pins and smaller packages
  - Greatly reducing interconnect complexity
  - Allowing digitally controlled components to be located close to their point of use

I2C Characteristics

- Electrical, timing specifications and an associated bus protocol
- Two wire serial data & control bus implemented with the serial data (SDA) and clock (SCL) lines (common ground is required).
- Unique start and stop condition
- True multi-master capability
- Slave selection protocol uses a 7-Bit slave address
- Bi-directional data transfer
- Acknowledgement after each transferred byte
- No fixed length of transfer

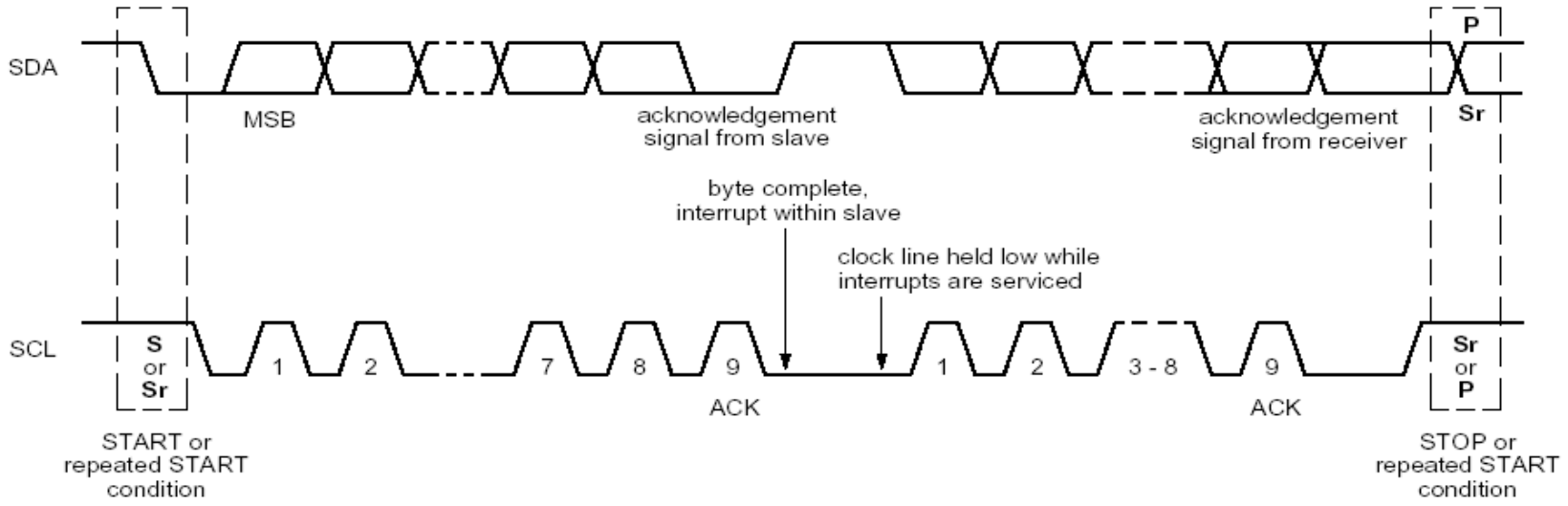
Definitions

- **Master:**
  - Initiates a transfer by generating start and stop conditions
  - Generates the clock
  - Transmits the slave address
  - Determines data transfer direction
- **Slave:**
  - Responds only when addressed
  - Timing is controlled by the clock line



Communication Protocol

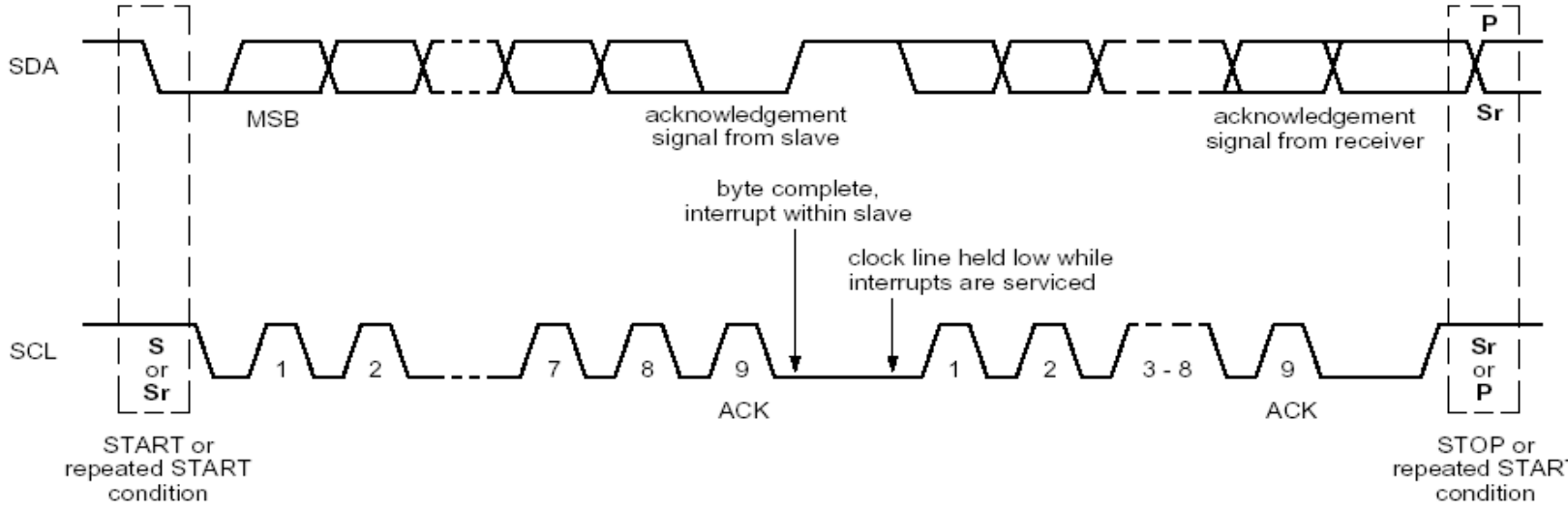
- Start & Stop condition
- Slave identification
- Transfer data



Communication Protocol

Data Transfer - SCL

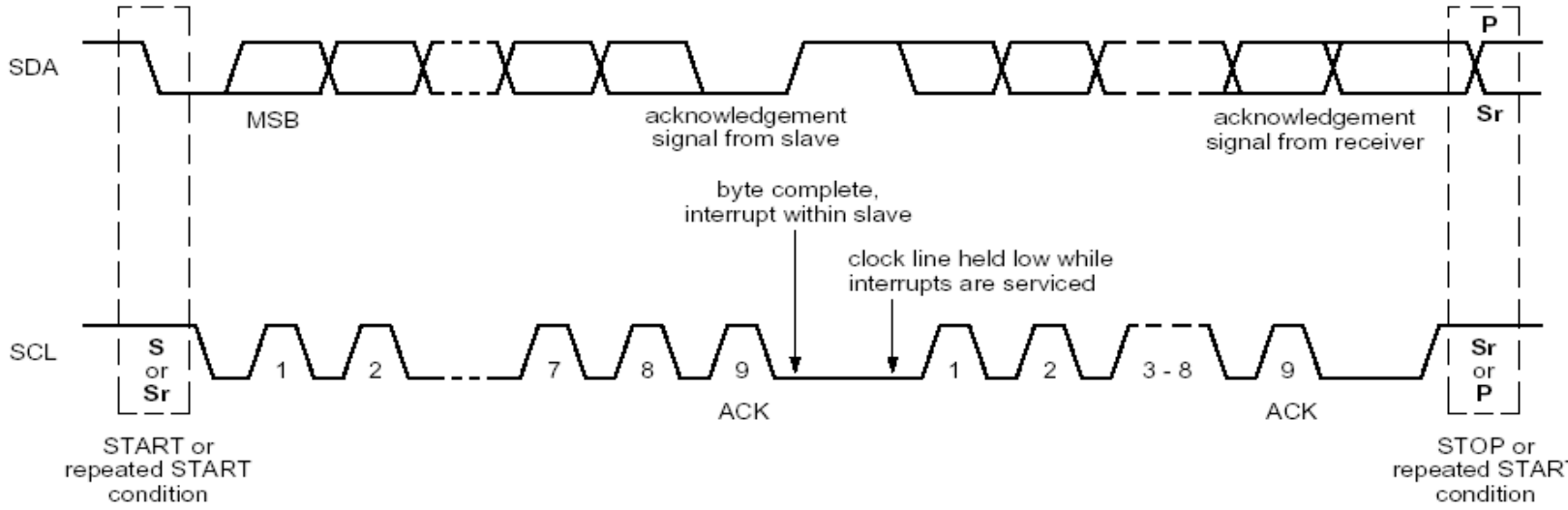
- master sets SCL = 0 and generates pulse for each data bit
- 8 pulses for data bits are followed by one pulse for ack. Bit after ack.
  - master tries to generate next byte's first pulse
  - slave can hold SCL low → master switches to wait state



Communication Protocol

Data Transfer – SDA

- Data bits are generated by transmitter as SCL pulses
- 9-th pulse:
  - transmitter releases SDA
  - receiver must hold SDA low in order to ack. received data
  - slave must release SDA after ack. bit (allows master to end frame)



Communication Protocol

Addressing Slaves – 7 bits

- The first byte transmitted by master:
  - 7 bits: address
  - 1 bit: direction (R/W)

    0 … master writes data (W), becomes transmitter

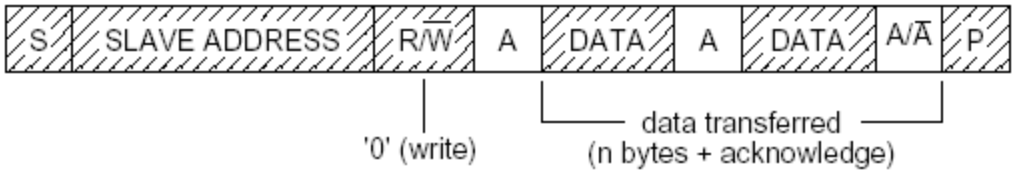
    1 … master reads data (R), becomes receiver
- Data transfer terminated by stop condition
- Master may generate repeated start and address another device
- Each device listens to address
  - address matches its own → device switches state according to R/W bit

- Address = fixed part + programmable part
  - fixed part assigned by I^2C committee

Communication Protocol

Frame formats

Master-transmitter



from master to slave

from slave to master

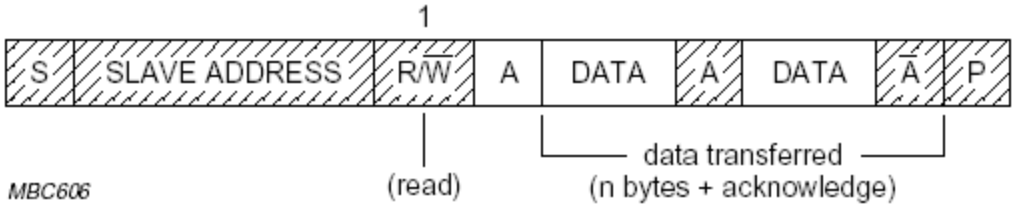
A = acknowledge (SDA LOW)

Ā = not acknowledge (SDA HIGH)

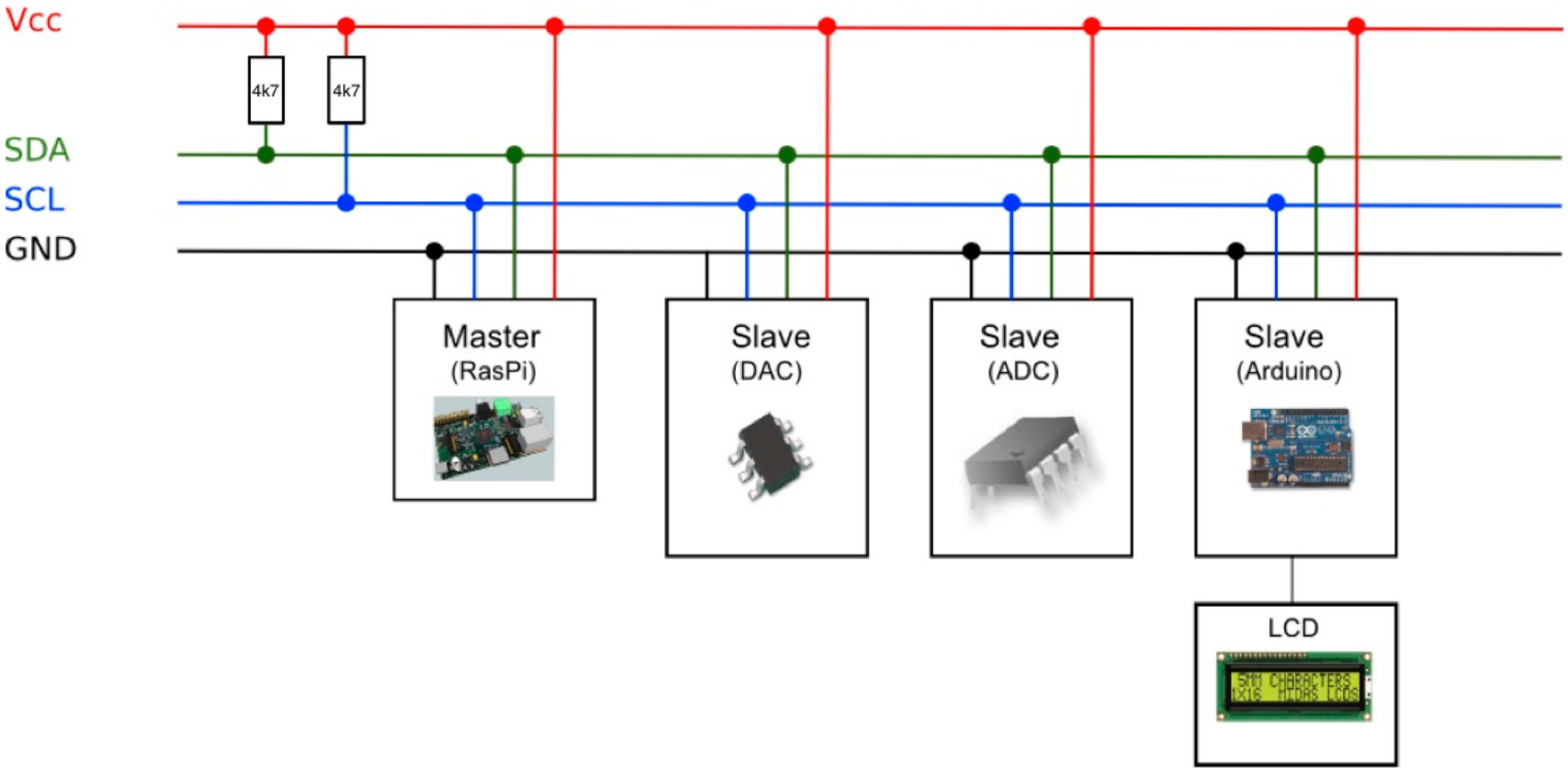
S = START condition

P = STOP condition

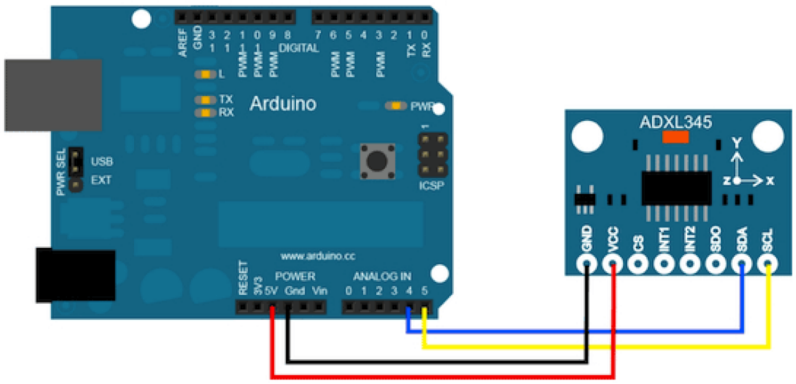
Master-receiver (since second byte)



Typical I2C configuration



Arduino I2C interconnection



I2C lines are mapped to the A4 (SCA) and A5 (SCL) analog inputs.

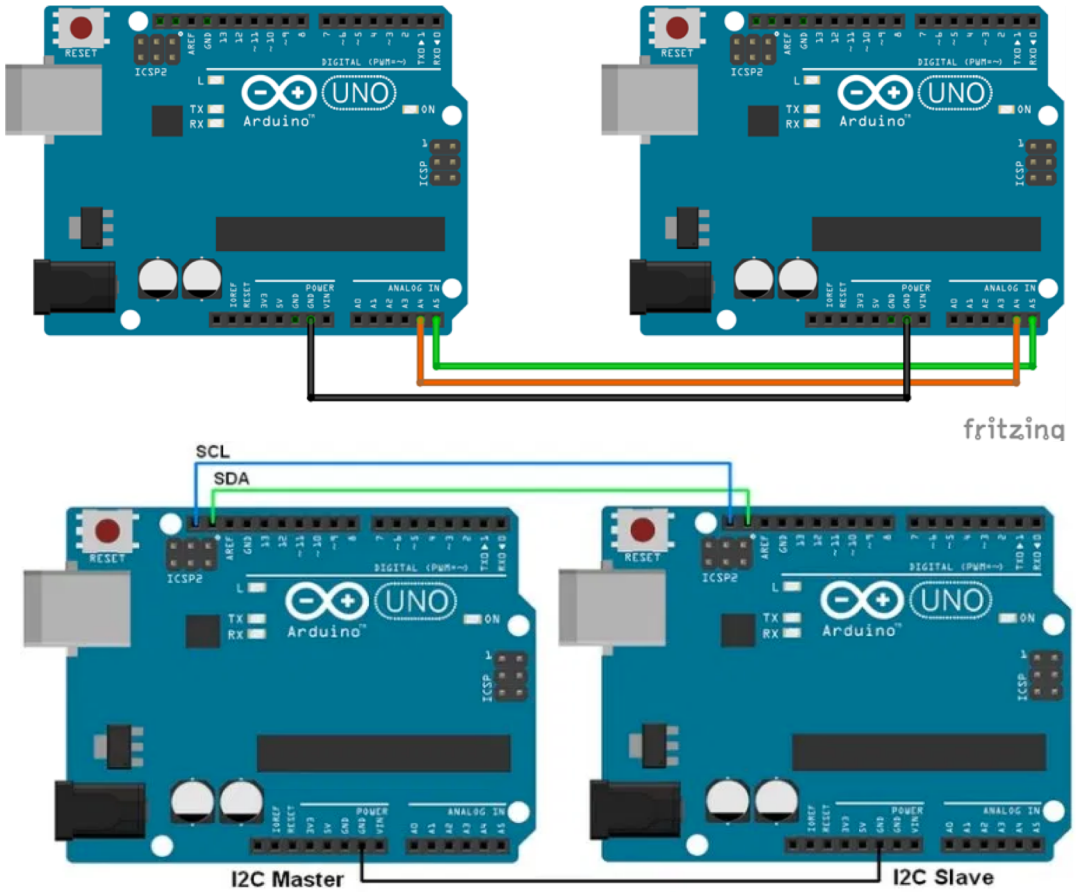
To work with the I2C BUS the Wire Library is required.

https://www.arduino.cc/en/Reference/Wire

Functions

- `begin()`
- `requestFrom()`
- `beginTransmission()`
- `endTransmission()`
- `write()`
- `available()`
- `read()`
- `SetClock()`
- `onReceive()`
- `onRequest()`

Arduino to Arduino interconnection



One must be configured as Master while the other will act as Slave.

Be really careful with sharing the Ground level

I2C – Arduino as Master

```
//Configuring the I2C Bus as Master – Sending data

#include <Wire.h>
#define I2C_SLAVE 0x04

unit8_t data=0x20; // Unsigned 8 bits variable

void setup(){
  Wire.begin(); // Join as the Master
}

void loop (){
  ...
  Wire.beginTransmission(I2C_SLAVE); // Start transmission - device 0x04
  Wire.write("Message");             // sends 7 ASCII bytes
  Wire.write(data);                  // sends one unsigned byte
  Wire.endTransmission();            // End of transmission
  ...
}
```

I2C – Arduino as Master

```
//Configuring the I2C Bus as Master – Requiring data

#include <Wire.h>
#define I2C_SLAVE 0x04

unit8_t data; // Unsigned 8 bits variable

void setup(){
  Wire.begin(); // Join as the Master
}

void loop (){
 ...
 Wire.beginTransmission(I2C_SLAVE); // Start transmission - device 0x04
 Wire.write(SLAVE_REGISTER);        // Determine the accessed register
 Wire.requestFrom(I2C_SLAVE, 6);    // request 6 bytes from device
 while (Wire.available()) {
    data = Wire.read();             // receive bytes
    Serial.print(c);
 }
 Wire.endTransmission();            // End of transmission
 ...
}
```

When requiring specific register data from any slave, it must be previously identificated.

I2C – Arduino as slave

```
//Configuring the I2C Bus as Slave

#include <Wire.h>
#define I2C_ADDR 0x04
unit8_t data; // Unsigned 8 bits variable

void setup(){
  Wire.begin(I2C_ADDR);
  // Callbacks functions for I2C
  // communication
  Wire.onReceive(receiveData_handler);
  Wire.onRequest(sendData_handler);
}

void loop(){
 ...
 ...
}
```

```
// Callback for received data
void receiveData_handler (int byteCount){
 ...
  while(Wire.available()) {
     data=Wire.read()
      ...
  }
 ...
}

// Callback for sending data
void sendData_handler (){
 ...
 Wire.write(data);
 ...
}
```

There are examples in the Arduino's IDE