Microprocessors Lab

7- I2C (TWI)

Instructor: Dr. Mohsen Raji

Graders:
- Mohammad Hossein Hashemi
- Reza Hesami
- Negin Shirvani

Shiraz University – Fall 2022

Based on the slides by:
- Hossein Dehghanipour
- Mohammad Hossein Allah Akbari

What is I2C?

- The I2C protocol involves using two lines to *send* and *receive* data: a serial clock pin (SCL) that the Arduino 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. (reference)
- As you may have realized by now, the **only** line that the data is sent/received by is SDA. This means that SDA is a Half-duplex wire.
- I2C is Serial (sends data bit by bit) and Synchronous (needs a clock for synchronization) protocol, therefore, SCL is a line used for sync clock.

More About I2C

- Stands for **Inter-integrated Circuit (IIC or I2C)**

- It is a serial communications protocol similarly to UART but not used for PC-device communication but are used with modules and sensors.

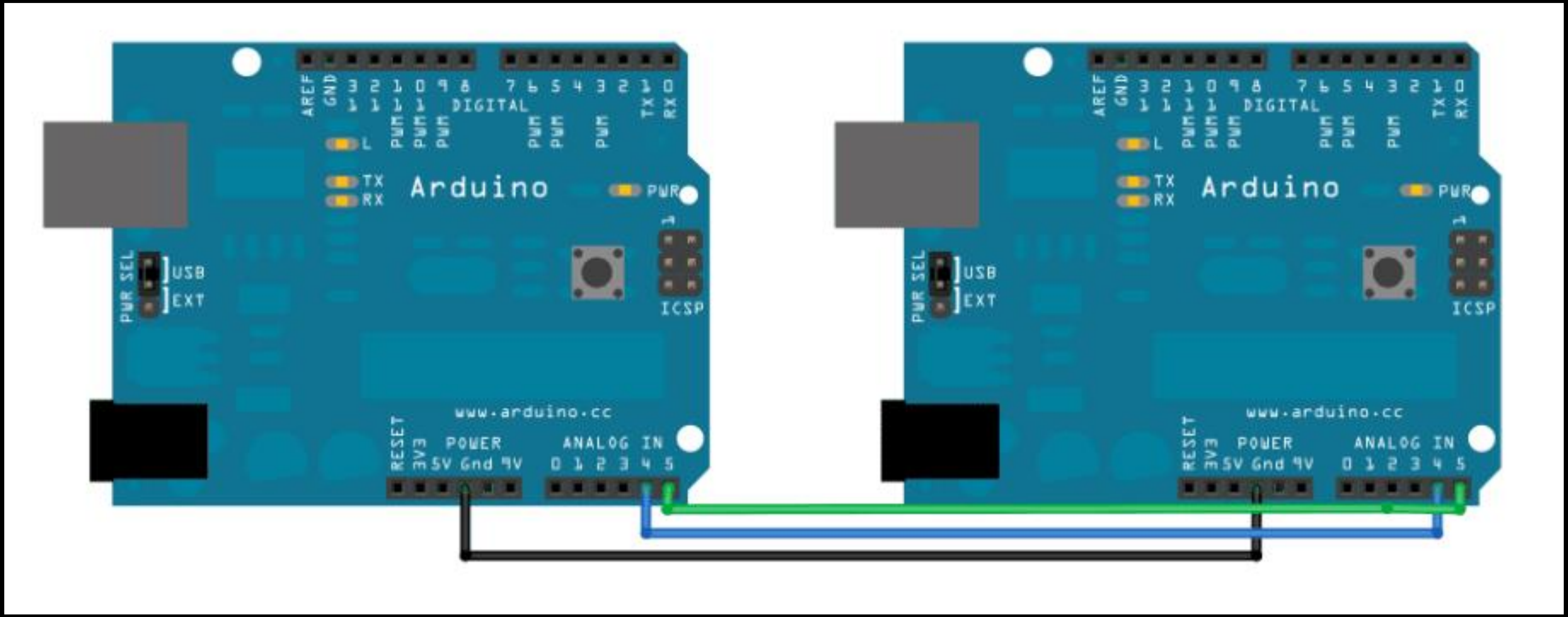
- It is a simple, bidirectional two-wire synchronous serial bus and requires only two wires to transmit information between devices connected to the bus.

- They are useful for projects that require many different parts (eg. sensors, pin, expansions and drivers) working together as they can connect up to 128 devices (7 bits of addressing) to the mainboard while maintaining a clear communication pathway!

- It is as I2C uses an address system and a shared bus = many different devices can be connected using the same wires and all data are transmitted on a single wire and have a low pin count. However, the tradeoff for this simplified wiring is that it is slower than SPI.
- Speed of I2C is also dependent by data speed, wire quality and external noise.

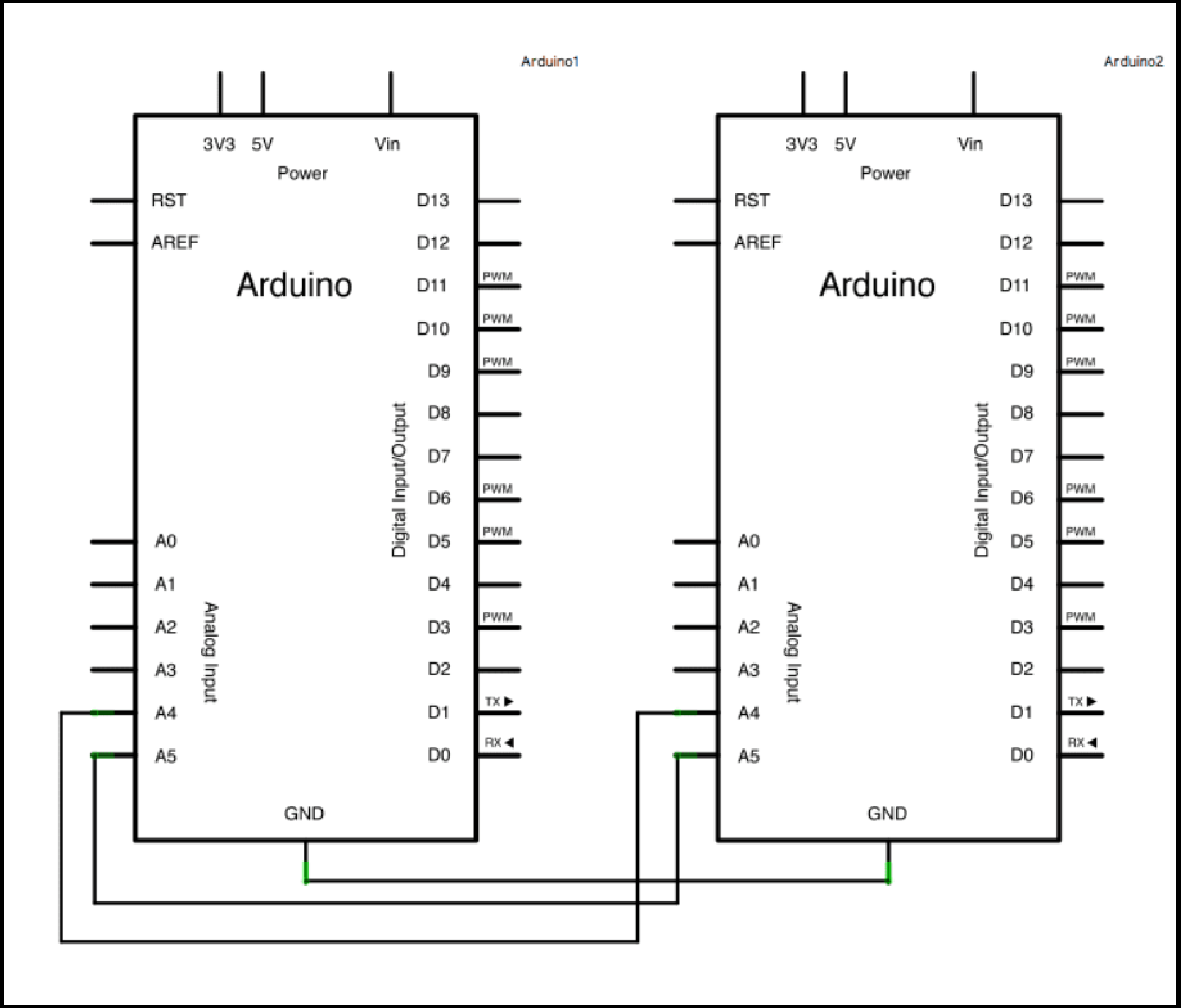
- The I2C protocol is also used for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems

reference



- Connect:
  - Master(A4) to Slave(A4)
  - Master(A5) to Slave(A5)
  - Master(Gnd) to Slave(Gnd)

reference

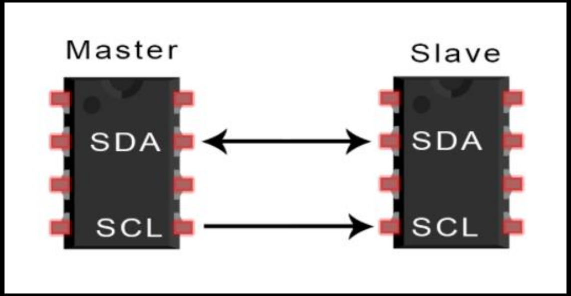

Arduino1
Arduino2
Arduino
3V3
5V
Vin
Power
RST
AREF
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
PWM
TX ▶
RX ◀
Digital Input/Output
A0
A1
A2
A3
A4
A5
Analog Input
GND

[reference]

Pros and Cons

Pros

Like UART communication, I2C only uses two wires to transmit data between devices



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

Has a low pin/signal count even with numerous devices on the bus.

Flexible, as it supports multi-master and multi slave communication.

Simple as it only uses 2 bidirectional wires to establish communication among multiple devices.

Adaptable as it can adapt to the needs of various slave devices.

Support multiple masters.

reference

Cons

•Slower speed as it requires pull-up resistors rather than push-pull resistors used by SPI. It also has an open-drain design = limited speed.

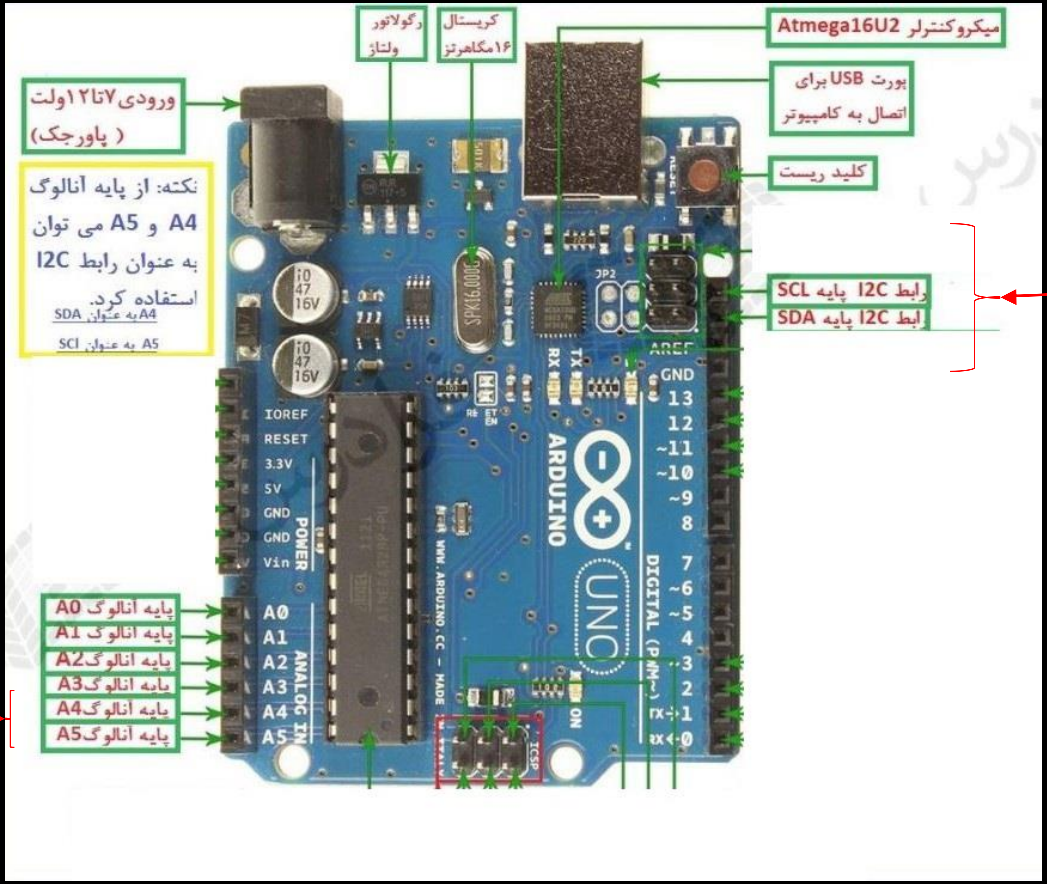
•Requires more space as the resistors consume valuable PCB real estate.

•May become complex as the number of devices increases.

•Only used for short distances (in scale of meters).

reference

- There are two ways to use I2C protocol:
  - Software Mode: Using A4 & A5 Analog pins
  - Hardware Mode (TWI): Using two special pins on the Arduino
  - *: Both of these pins are connected to a single reference.



How it Works

- It has 2 lines which are SCL (serial clock line) and SDA (serial data line acceptance port)
- SCL is the clock line for synchronizing transmission. SDA is the data line through which bits of data are sent or received.
- The master device initiates the bus transfer of data and generates a clock to open the transferred device and any addressed device is considered a slave device.
- The relationship between master and slave devices, transmitting and receiving on the bus is not constant. It depends on the direction of data transfer at the time.
- If the master wants to send data to the slave, the master must first address the slave before sending any data.
- The master will then terminate the data transfer. If the master wants to receive data from the slave, the master must again address the slave first.
- The host then receives the data sent by the slave and finally, the receiver terminates the data trabsfer. The host is also responsible for generating the timing clock and terminating the data transfer/receiving process. .
- It is also necessary to connect the power supply through a *pull-up resistor*. When the bus is idle, both lines on a high power level.
- The capacitance in the line will affect the bus transmission speed. As the current power on the bus is small, when the capacitance is too large, it may cause transmission errors. Thus, its load capacity must be 400pF, so the allowable length of the bus and the number of connected devices can be estimated.

reference

Data Transfer Method

- The master sends the transmitting signal to every connected slave by switching the SDA line from a high voltage level to a low voltage level and SCL line from high to low after switching the SDA line.
- The master sends each slave the 7 or 10-bit address of the slave and a read/write bit to the slave it wants to communicate with.
- The slave will then compare the address with its own. If the address matches, the slave returns an ACK bit which switches the SDA line low for one bit. If the address does not match its address, the slave leaves the SDA line high
- The master will then send or receive the data frame. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful transmission.
- To stop the data transmission, the master sends a stop signal to the slave by switching SCL high before switching SDA high

Reference

Wire Library

(further reading)

This library allows you to communicate with I2C / TWI devices. On the Arduino boards with the R3 layout (1.0 pinout), the SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin. The Arduino Due has two I2C / TWI interfaces SDA1 and SCL1 are near to the AREF pin and the additional one is on pins 20 and 21.
As a reference the table below shows where TWI pins are located on various Arduino boards.

```
#include <Wire.h>
```

Board	I2C / TWI pins
Uno, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

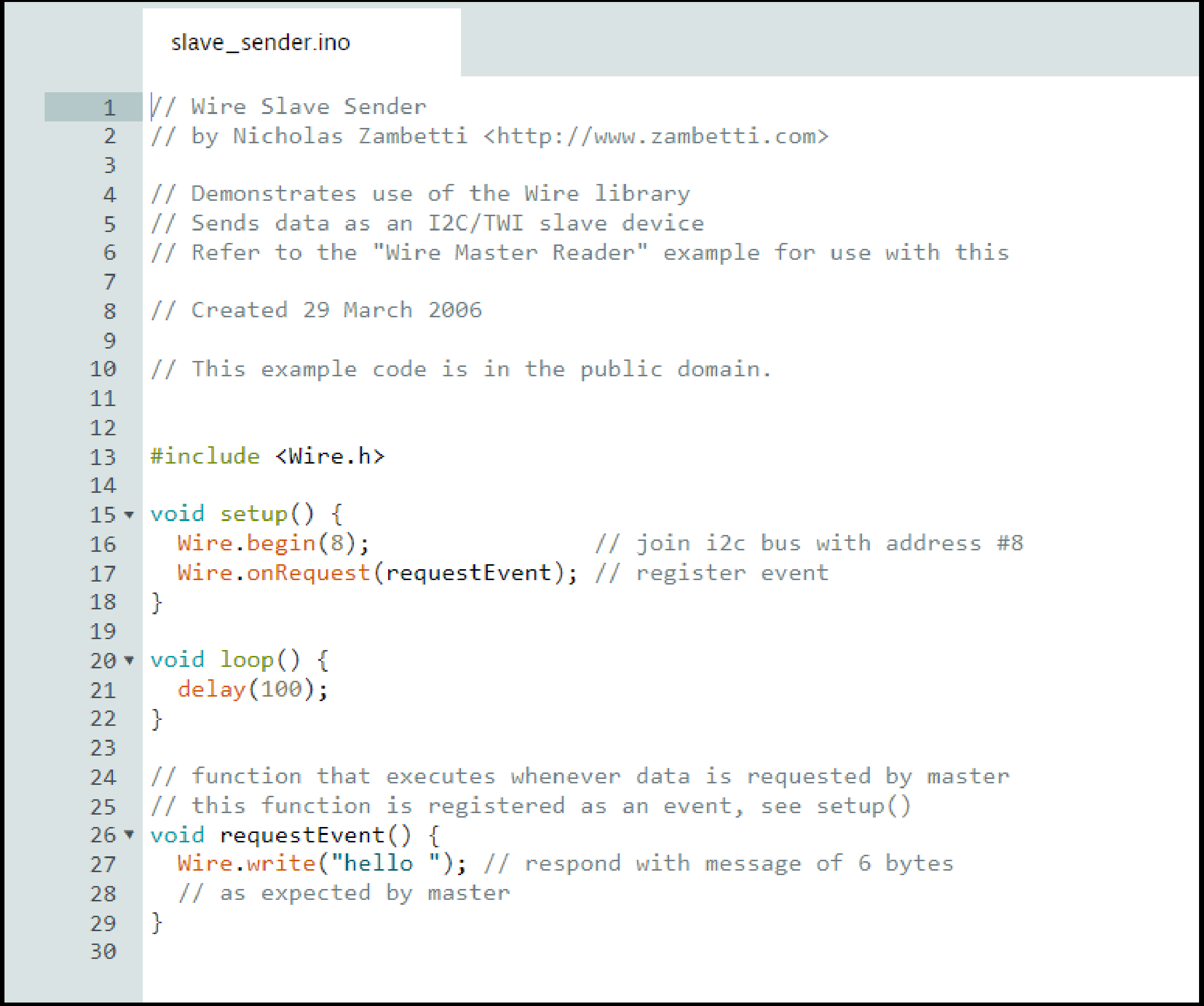
Functions

For detailed information, please check the given links. We refrain writing them here in order to avoid redundancy.

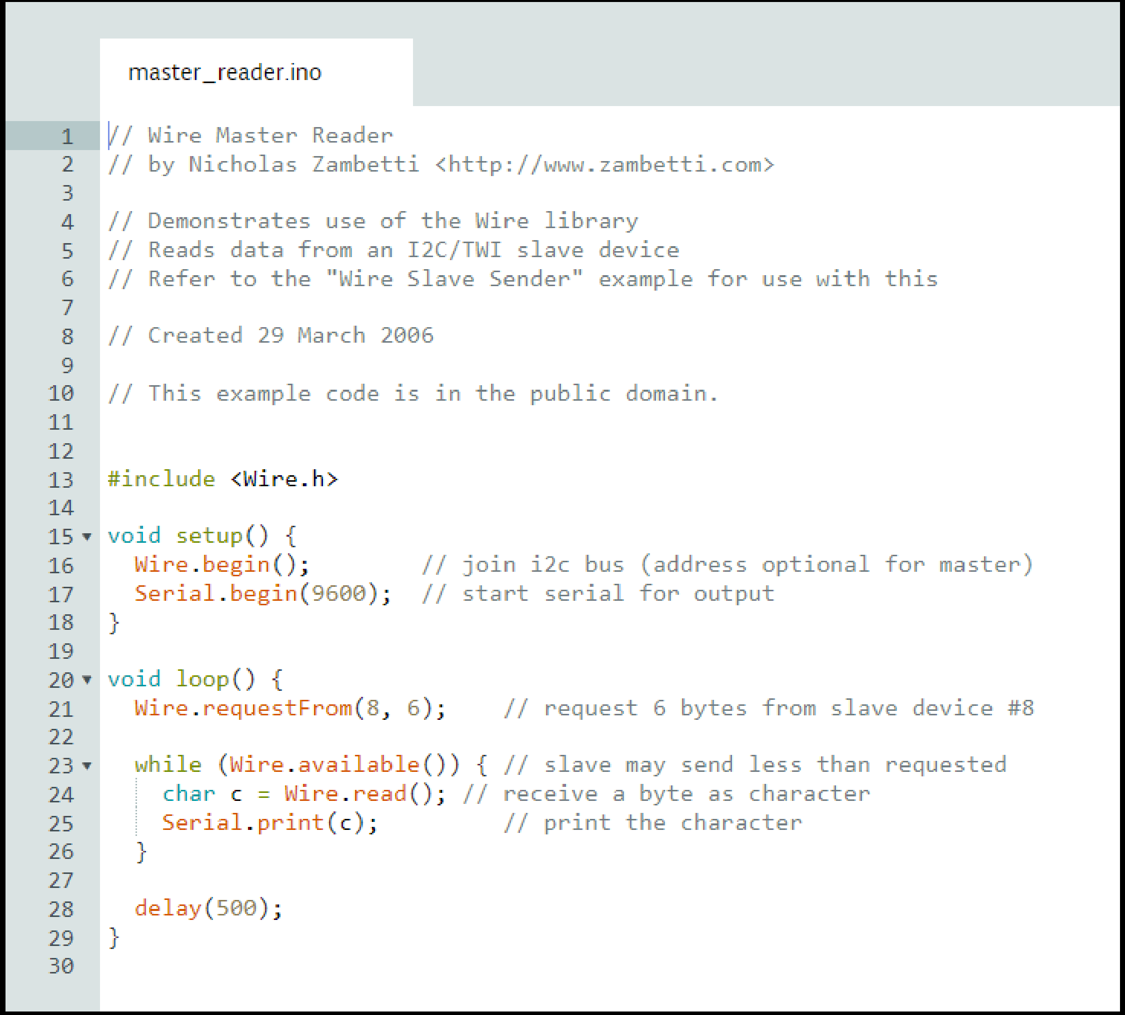
- begin() : Begins the connection
- requestFrom() : Requests for Data
- beginTransmission(): begins the Data Transmission
- endTransmission() : Terminates Data Transmission
- write() : Write the Data on the Bus.
- available() : Checks whether any bytes of data is being received.
- read() : Returns the received data (only one byte)
- SetClock() : Modifies the clock frequency for I2C communication.
- onReceive(): Sets a Handler to be called while receiving data.
- onRequest() : Sets a Handler to be called while the master is requesting data from its slave.

Sample

Slave sending Data to Master


slave_sender.ino

```
// 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
}

```


master_reader.ino

```
// 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);
}

```

Reference: https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterReader

Classwork:

In a hypothetical communication system, we have one master device and multiple slave devices. Each slave is meant to show a specific message (on a character LCD or Serial interface). The master sends different messages to all of these slave devices periodically.

In this classwork, you will write one sketch for slave and one for master device. Assume that there is only one slave in the system right now, and you just send that specific slave a hardcoded string message every 5 seconds and the slave prints that message on the Serial interface.

There is no need to mention that we will use I2C protocol for this purpose (why?) and you're not allowed to use any libraries except "wire.h".

After all this time, if you still have any questions and you don’t know what to do, go ahead; ask your TAs.

Good Luck ☺

Further Reading

1. https://www.seeedstudio.com/blog/2019/09/25/uart-vs-i2c-vs-spi-communication-protocols-and-uses/
2. https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterReader
3. https://www.arduino.cc/en/Reference/Wire