Faculty of Engineering and Technology

Computer Engineering Department

ENCS4380, INTERFACING TECHNIQUES

I2C protocol implementation without (wire.h) library… using signaling on the bits

Prepared by: -

Student name & number: Jubran Khoury - 1201264

Ins: Dr. Wasel Ghanem

Section: 2

Date: 29 May 2024

Contents

Introduction

I2C (Inter-Integrated Circuit) protocol is synchronous, half duplex protocol

- Synchronous: has a clock line (SCL)
- Half duplex: has only one data line (SDA), at a time only send or receive (not both at same time).

I2C is a multi-master, multi-slave protocol.

Each slave defined by a unique address.

I2C protocol needs a pull up resistors in order to rise the bus to VDD when the bus is idle (without the pull up resistors the bus will be floating, which is bad), these pull up resistors are built in the Arduino microcontroller chip (so no need to connect them).

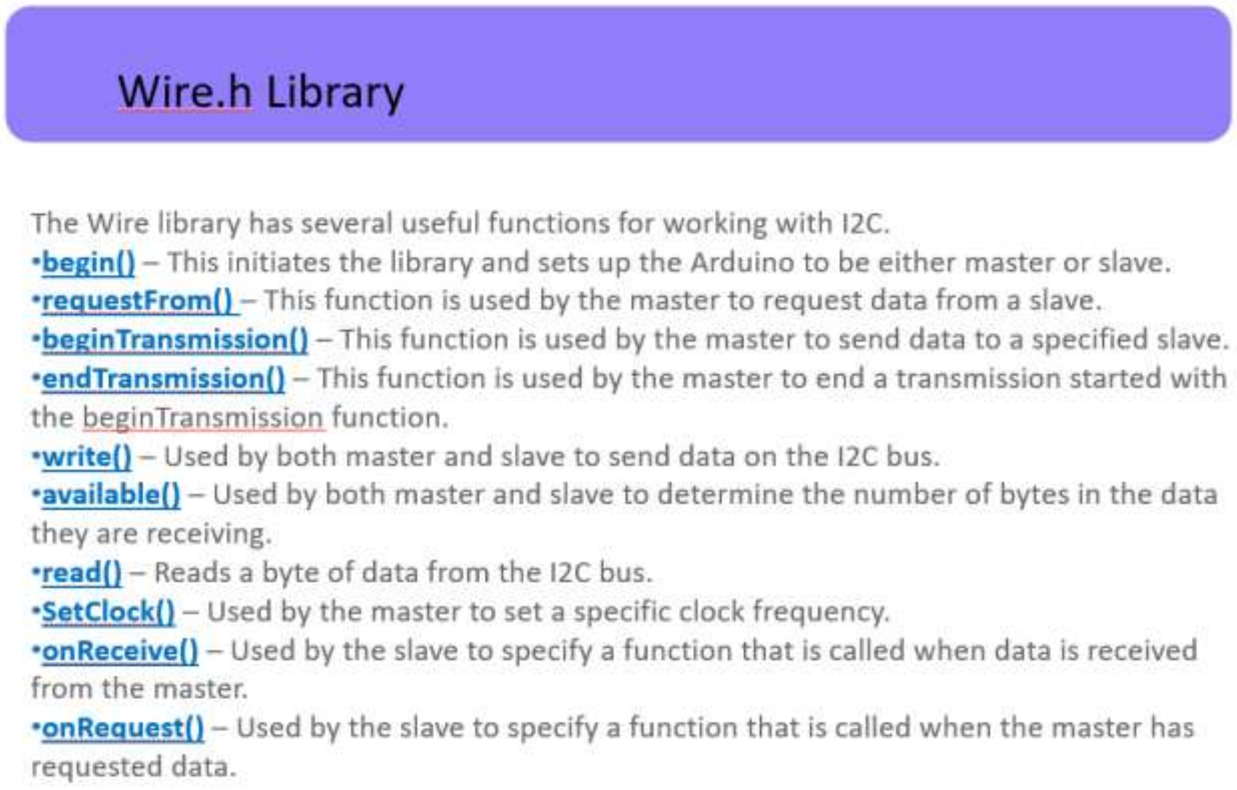
Reference (your lecture).

Procedure

With library

At first to go into the protocol and understand it, I tried at first to build the protocol using the library (wire.h).

The wire.h library functions are



Wire.h Library

The Wire library has several useful functions for working with I2C.
•**begin()** – This initiates the library and sets up the Arduino to be either master or slave.
•**requestFrom()** – This function is used by the master to request data from a slave.
•**beginTransmission()** – This function is used by the master to send data to a specified slave.
•**endTransmission()** – This function is used by the master to end a transmission started with the beginTransmission function.
•**write()** – Used by both master and slave to send data on the I2C bus.
•**available()** – Used by both master and slave to determine the number of bytes in the data they are receiving.
•**read()** – Reads a byte of data from the I2C bus.
•**SetClock()** – Used by the master to set a specific clock frequency.
•**onReceive()** – Used by the slave to specify a function that is called when data is received from the master.
•**onRequest()** – Used by the slave to specify a function that is called when the master has requested data.

Figure 1(wire.h library) [ref. your slides p.37)

The circuit and its connection in addition to the code are shown below,

Circuit

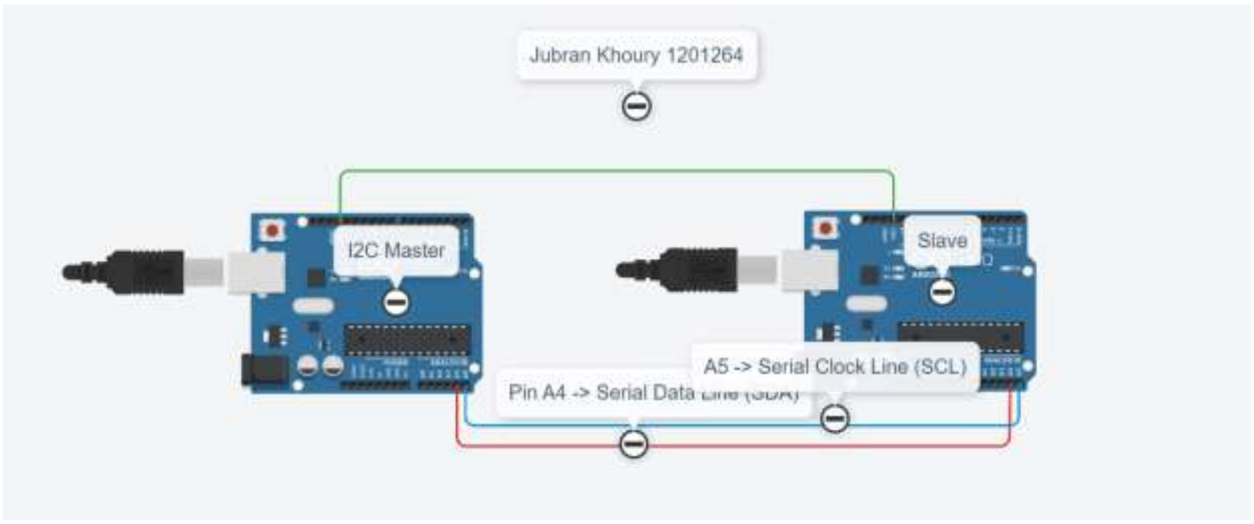

Jubran Khoury 1201264
I2C Master
Slave
A5 -> Serial Clock Line (SCL)
Pin A4 -> Serial Data Line (SDA)

Figure 2(The circuit)

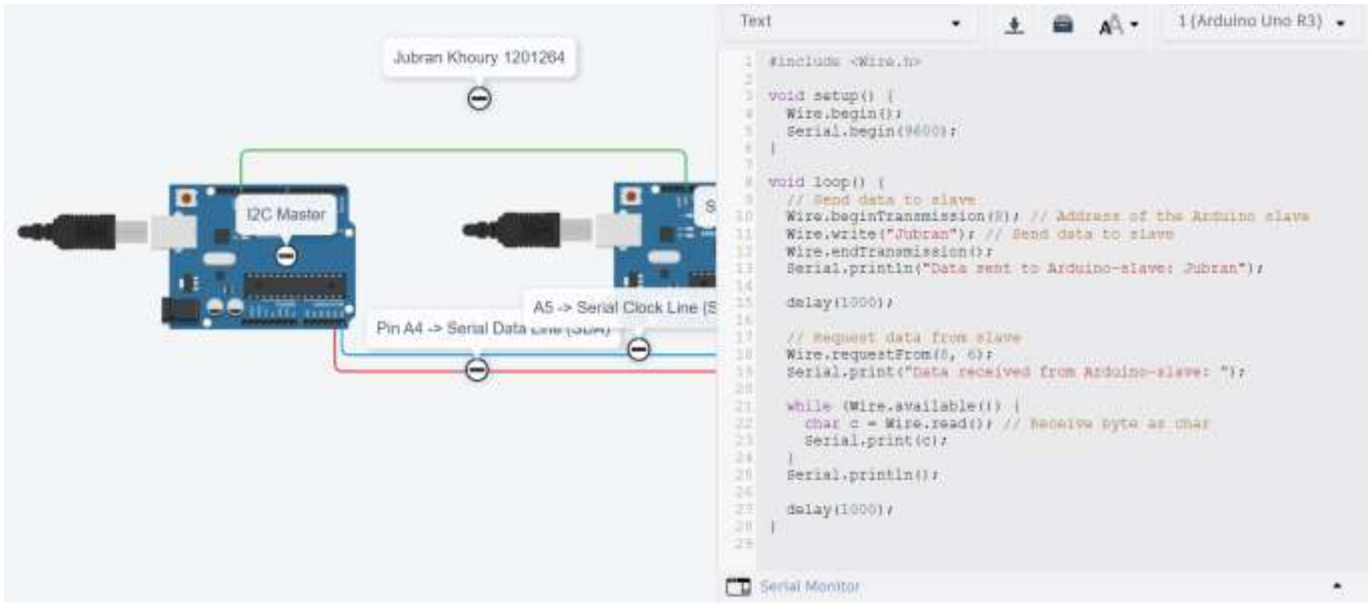
Both connected with common ground (very necessary point).

Master Pin A4(SDA) to Slave Pin A4(SDA).

Master Pin A5(SCL) master to Slave Pin A5(SCL).

Codes

Master code



```
#include <Wire.h>

void setup() {
  Wire.begin();
  Serial.begin(9600);
}

void loop() {
  // Send data to slave
  Wire.beginTransmission(8); // Address of the Arduino slave
  Wire.write("Jubran"); // Send data to slave
  Wire.endTransmission();
  Serial.println("Data sent to Arduino-slave: Jubran");

  delay(1000);

  // Request data from slave
  Wire.requestFrom(8, 6);
  Serial.print("Data received from Arduino-slave: ");

  while (Wire.available()) {
    char c = Wire.read(); // Receive byte as char
    Serial.print(c);
  }
  Serial.println();

  delay(1000);
}
```

Figure 3(master code)

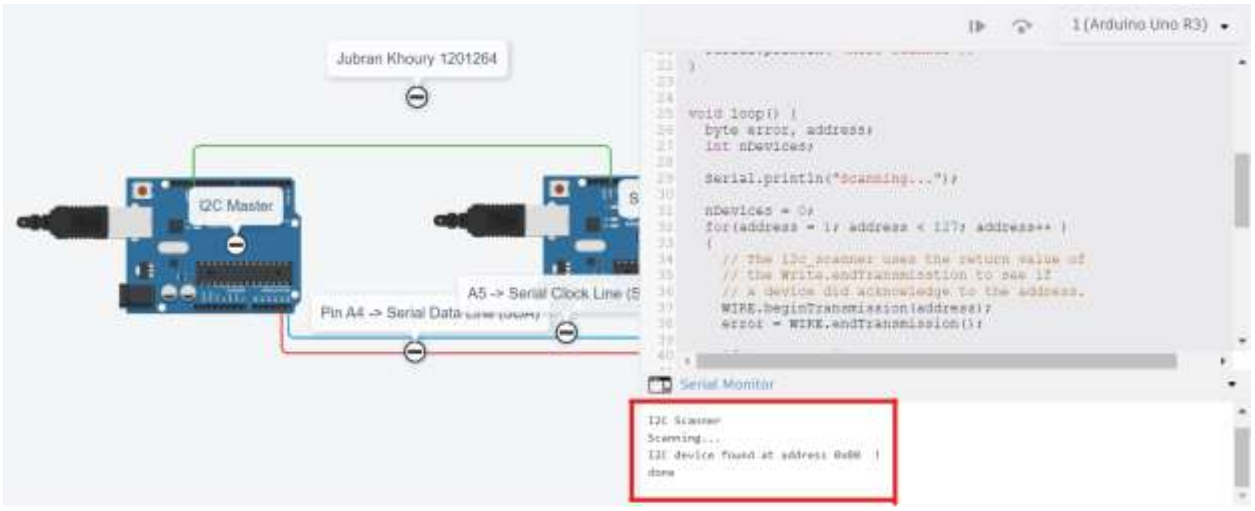
In the setup, the wire.begin ,

•**begin()** – This initiates the library and sets up the Arduino to be either master or slave.

Ref (your slides p.37)

In the loop,

Wire.beginTransmission this function is used by the master to send data to a specific slave defined by its address. In this simulation the slave is at address **0x08**, founded by the following method (ref: https://learn.adafruit.com/scanning-i2c-addresses/arduino)



```
}


void loop() {
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    WIRE.beginTransmission(address);
    error = WIRE.endTransmission();
```

```
I2C Scanner
Scanning...
I2C device found at address 0x08  !
done
```

Then, I send my name "Jubran" to the slave by **Wire.write("Jubran")**.

And **Wire.endTransmission()** to end the transmission.

Delay of 1 second between sending and receiving **delay(1000)**.

Now to receive data from the slave….

The function **requestFrom()**

•**requestFrom()** – This function is used by the master to request data from a slave.

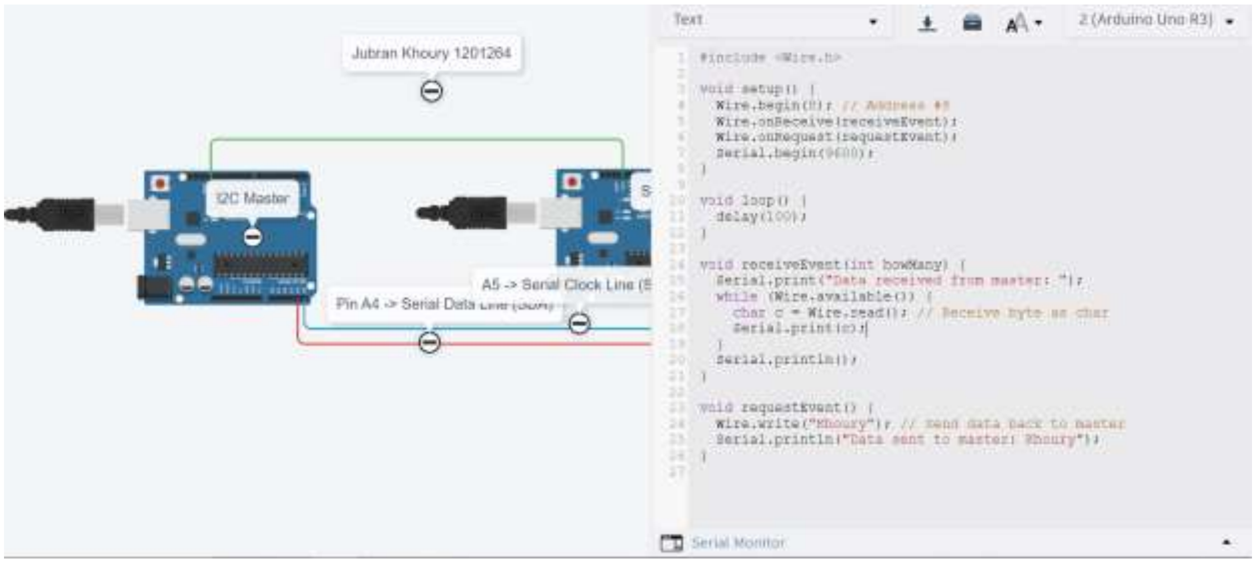
It takes two parameters the first is of course the address of the slave that wants to receive data from. And the second parameter is the number of bytes to be received, since it will receive my family name "Khoury" which it 6 char, then the master will receive each byte as char, as shown

```
// Request data from slave
Wire.requestFrom(8, 6);
Serial.print("Data received from Arduino-slave: ");

while (Wire.available()) {
  char c = Wire.read(); // Receive byte as char
  Serial.print(c);
}
Serial.println();
```

Then in a while loop…. as long as there are available data on the data line then read it as char and print it on the serial monitor.

Finally, a delay of 1 second is applied before the master send the data again to the slave and this process continue, because it's in the loop().

Slave code

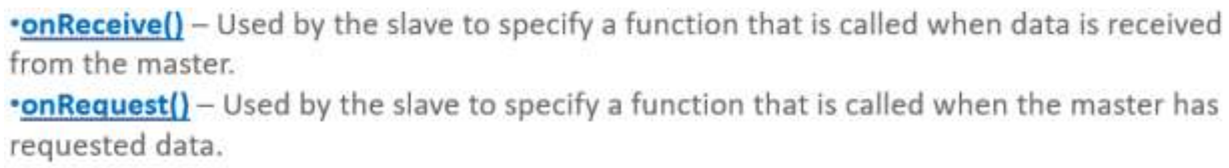


```
#include <Wire.h>

void setup() {
  Wire.begin(8); // Address #8
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
  Serial.begin(9600);
}

void loop() {
  delay(100);
}

void receiveEvent(int howMany) {
  Serial.print("Data received from master: ");
  while (Wire.available()) {
    char c = Wire.read(); // Receive byte as char
    Serial.print(c);
  }
  Serial.println();
}

void requestEvent() {
  Wire.write("Khoury"); // Send data back to master
  Serial.println("Data sent to master: Khoury");
}
```

Figure 4((slave code)

In the setup,

Wire.begin() with address of the slave.



•onReceive() – Used by the slave to specify a function that is called when data is received from the master.
•onRequest() – Used by the slave to specify a function that is called when the master has requested data.

Since both **onReceive()** and **onRequest()** need to call functions,

I have declared two functions.

First function is **receiveEvent()**, this function to print each received byte from the master on the serial monitor (each byte is received as char).

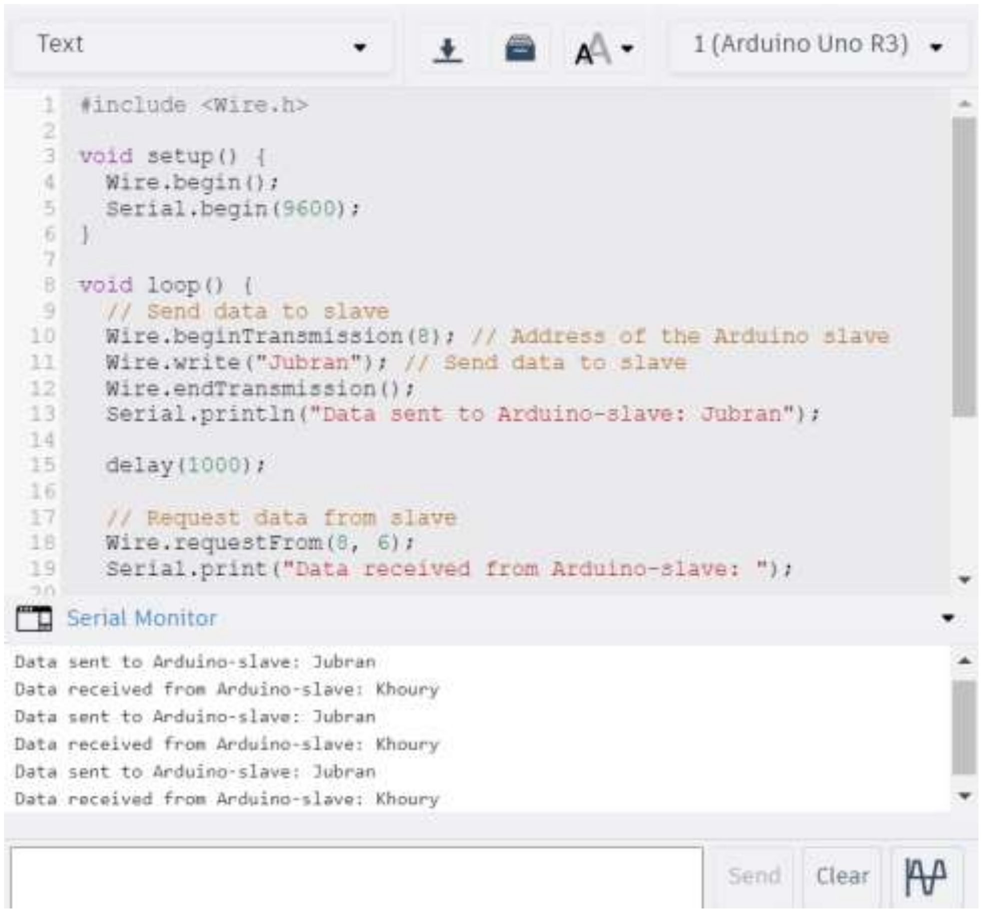
Second function is **requestEvent()**, this function sends data back to the master using **wire.write()**.

So, when the slave receives data from the master → print it on the serial monitor, because this is how **onReceive()** works, its calling the **receiveEvent()** only when the slave receive data from the master, and once the **receiveEvent()** is called, then it prints the data on the serial monitor.

And when the master request data from the slave then **onRequest()** is used, and this function call **requestEvent()**, and once this function is called, then it sends data back to the master.

Simulation result

Serial monitor of the master



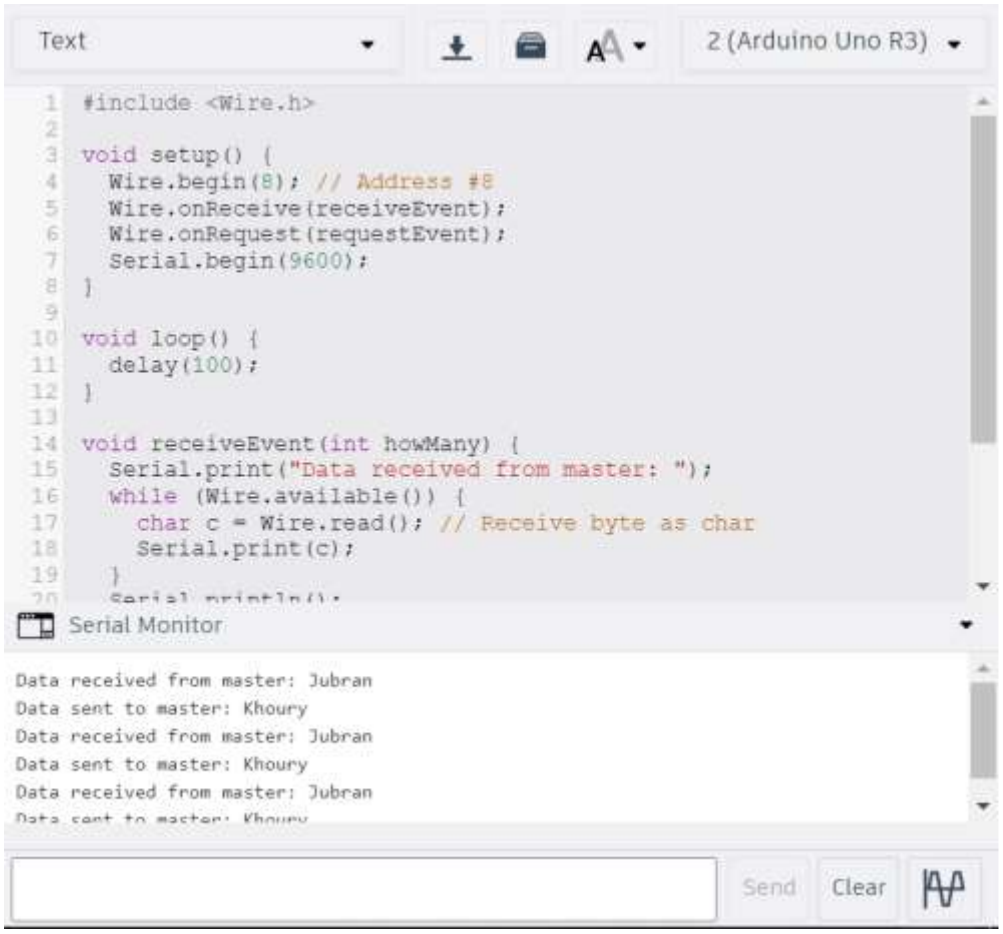
```
#include <Wire.h>

void setup() {
  Wire.begin();
  Serial.begin(9600);
}

void loop() {
  // Send data to slave
  Wire.beginTransmission(8); // Address of the Arduino slave
  Wire.write("Jubran"); // Send data to slave
  Wire.endTransmission();
  Serial.println("Data sent to Arduino-slave: Jubran");

  delay(1000);

  // Request data from slave
  Wire.requestFrom(8, 6);
  Serial.print("Data received from Arduino-slave: ");
```

```
Data sent to Arduino-slave: Jubran
Data received from Arduino-slave: Khoury
Data sent to Arduino-slave: Jubran
Data received from Arduino-slave: Khoury
Data sent to Arduino-slave: Jubran
Data received from Arduino-slave: Khoury
```

Figure 5(serial monitor of the master)

As shown above, the master sending my name "Jubran" to the slave and receive back my family name " Khoury".

Slave serial monitor



```
#include <Wire.h>

void setup() {
  Wire.begin(8); // Address #8
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
  Serial.begin(9600);
}

void loop() {
  delay(100);
}

void receiveEvent(int howMany) {
  Serial.print("Data received from master: ");
  while (Wire.available()) {
    char c = Wire.read(); // Receive byte as char
    Serial.print(c);
  }
  Serial.println();
```

```
Data received from master: Jubran
Data sent to master: Khoury
Data received from master: Jubran
Data sent to master: Khoury
Data received from master: Jubran
Data sent to master: Khoury
```

Figure 6(Slave serial monitor)

As shown above, the slave receiving my name "Jubran" from the master and sending back my family name " Khoury".

Implementing the communication with library helped me to better understanding how the I2C protocol works and to make a communication between the master and the slave using the I2C protocol.

Now to implement it without wire.h library ….

I2C protocol communication without using the library (By signaling on the bits)

How I thinks or what were my references?

Initially, my primary references were the contents of your lecture. During the lecture, you provided a concise overview of the communication mechanisms within this protocol. During the lecture I have written all the notes needed in order to build the I2C protocol communication.

Also, I utilized this slide (ref: your slides p.25) as a reference for better understanding and real example of the communication.

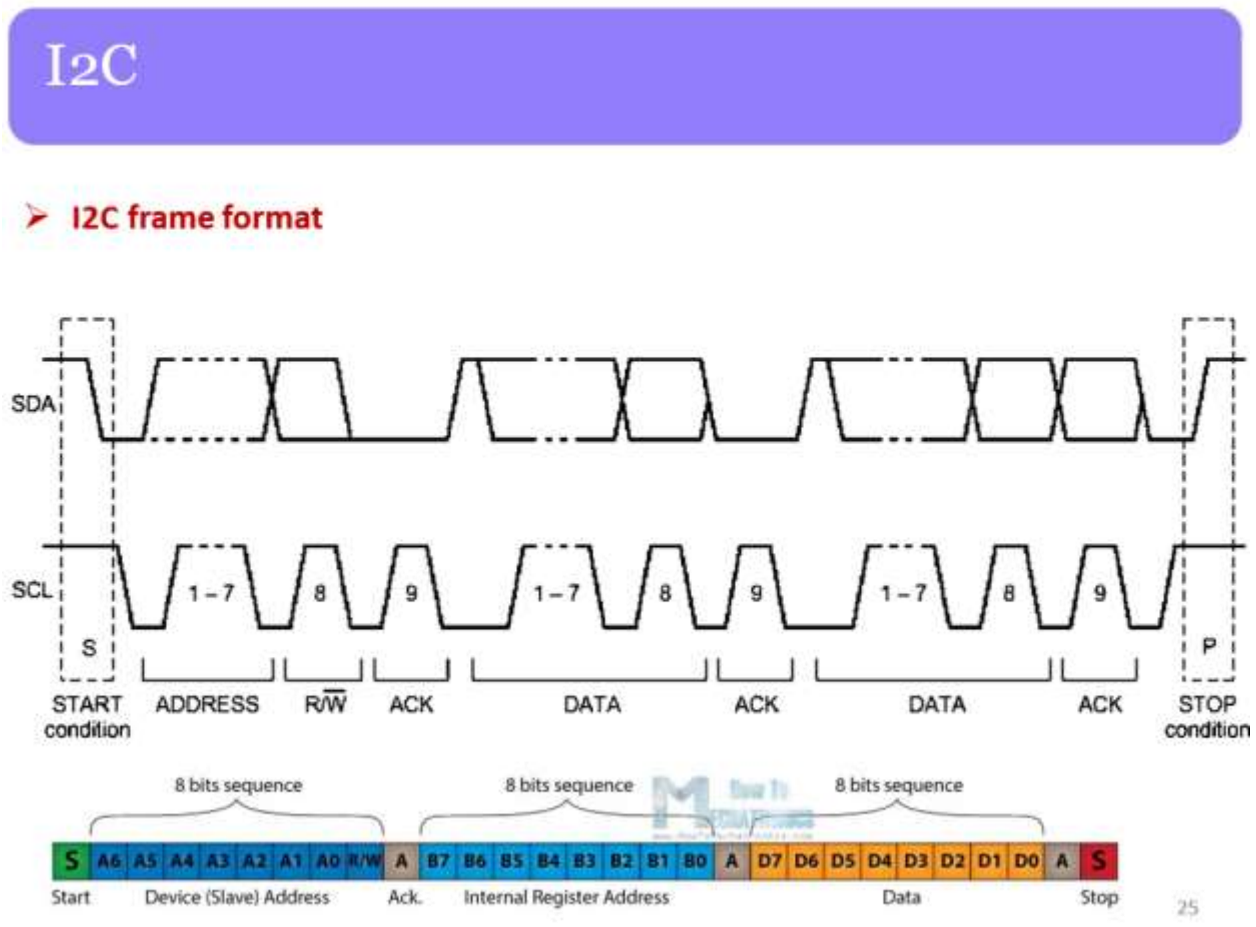
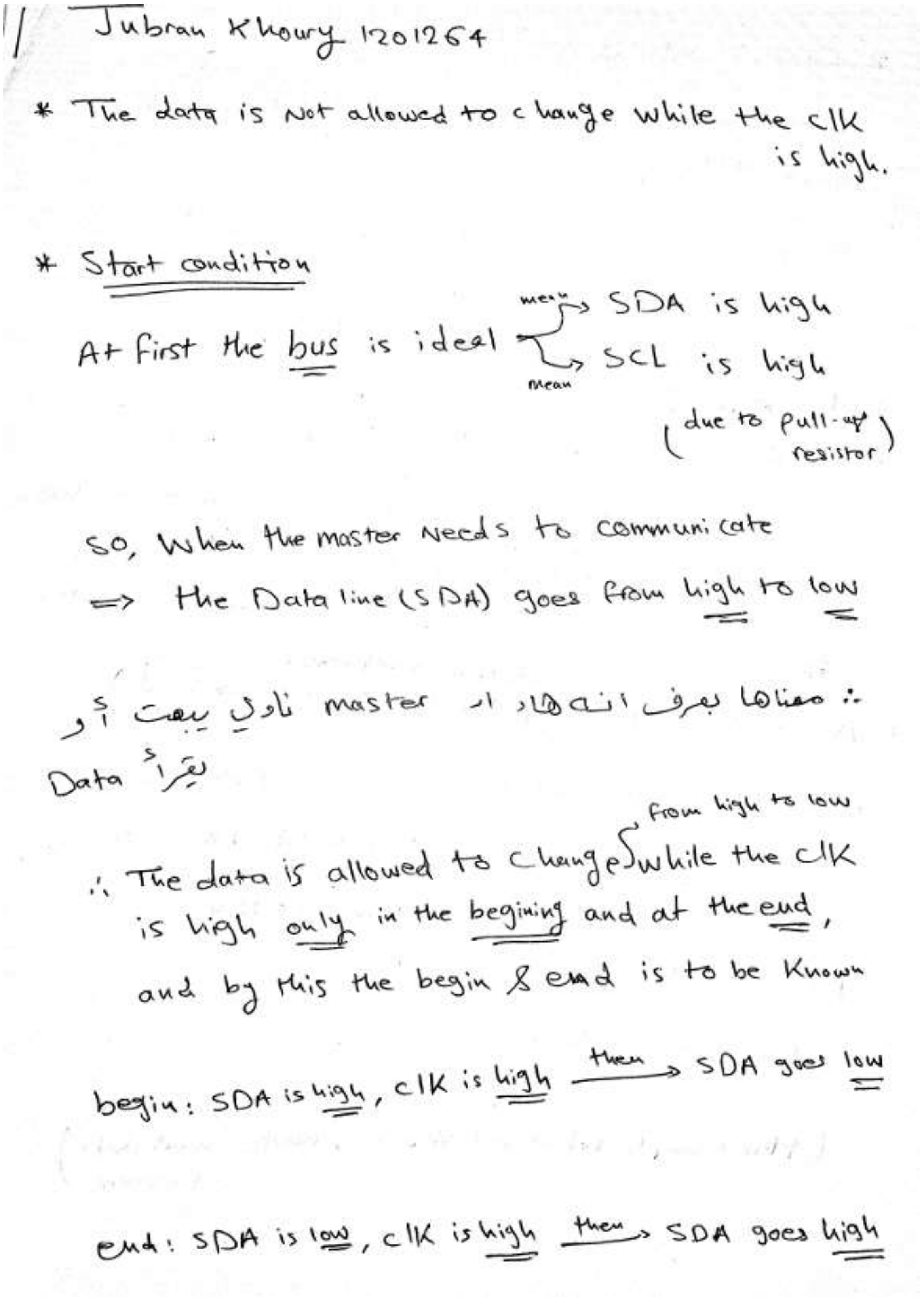


Figure 7(I2C frame format)

My notes



Jubran Khoury 1201264

* The data is Not allowed to change while the clK is high.

* Start condition

At first the bus is ideal $\xrightarrow{\text{mean}}$ SDA is high

$\xrightarrow{\text{mean}}$ SCL is high

(due to pull-up resistor)

So, When the master Needs to communicate

$\Rightarrow$ the Data line (SDA) goes from high to low

∴ معناها بعرف انه هاد ال master ناوي يبعث أو يقرأ Data

∴ The data is allowed to change (from high to low) while the clK is high only in the begining and at the end, and by this the begin & end is to be Known

begin: SDA is high, clK is high $\xrightarrow{\text{then}}$ SDA goes low

end: SDA is low, clK is high $\xrightarrow{\text{then}}$ SDA goes high

* First 7-bits is for Slave addresses.

As we talked early, each slave must have a unique address,

The available addresses are from [0 - 127], since 7-bits gives $2^7 = 128$ address.

* Bit # 8 ⟶ Read / write bit

if 1, then the master needs to read from the slave

if 0, then the master wants to send data to the slave.

* AcKnowledment

Now, the slave which has this address (the address at which the master sent the data to) must make an AcK, So on the bit # 9 goes low.

By this the slave say " i'am here".

(the AcK is always by low).

* After AcK → the master sends the data,
or Recieve data

In case the master wants to recieve data,
then the master must do the ACK.

So, the most important question
When should the master / slave stop from
AcK ??

Ans:- when they stop recieving data

→ end of the communication.

* Same thing when writing but the opposite.

* always, who recieve → make AcK

* of course the first AcK is from the slave.

Circuit

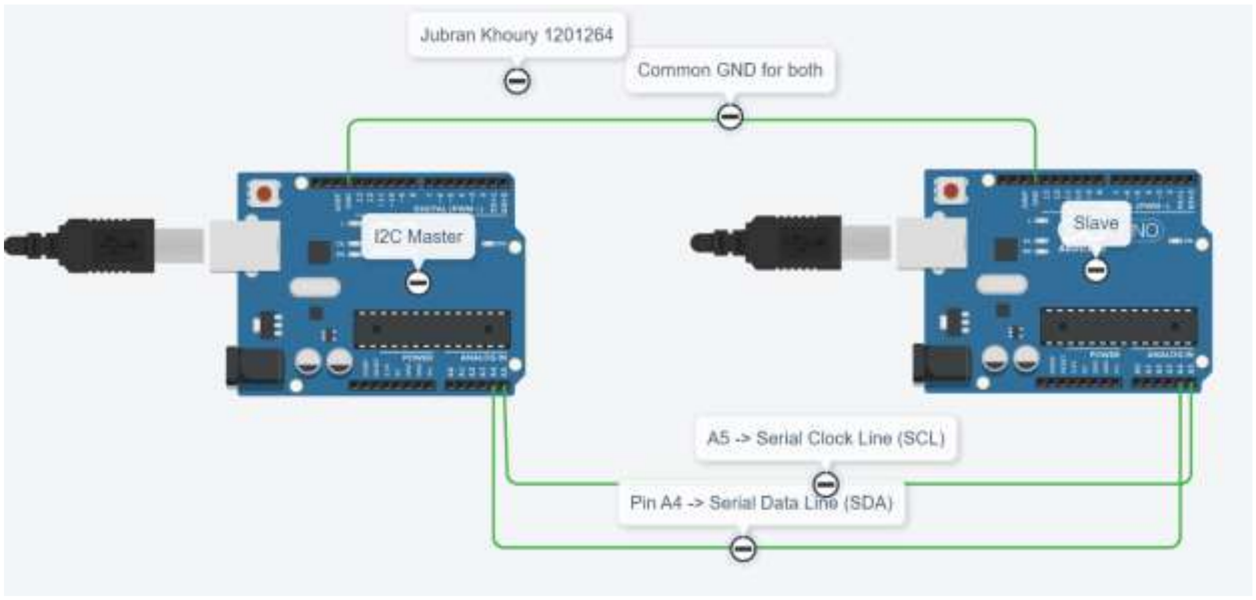

Jubran Khoury 1201264
Common GND for both
I2C Master
Slave
A5 -> Serial Clock Line (SCL)
Pin A4 -> Serial Data Line (SDA)

Figure 8(circuit)

Codes & Explanation

Master

```
#define SDA_PIN  A4
#define SCL_PIN  A5

// Slave address
#define SLAVE_ADDRESS 0x04

void setup() {
  Serial.begin(9600);
  pinMode(SDA_PIN, OUTPUT);
  pinMode(SCL_PIN, OUTPUT);

  // At first both are high
  digitalWrite(SDA_PIN, HIGH);
  digitalWrite(SCL_PIN, HIGH);
}
```

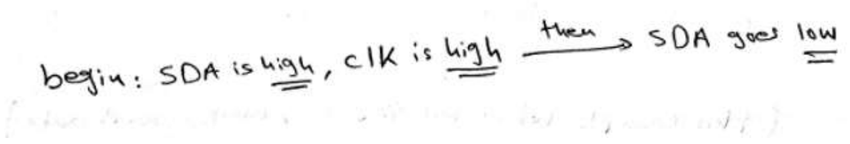
At first, I have given both A4 and A5 the corresponding names (just easy to deal with their names instead of the pins number).

Then, the slave address is at address 0x04 (the same way was used to search for the slave address).

And sets both high to indicate the I2C bus is idle.

```
// Start condition
void Start() {
  digitalWrite(SDA_PIN, LOW);
  delayMicroseconds(5);
  digitalWrite(SCL_PIN, LOW);
}
```

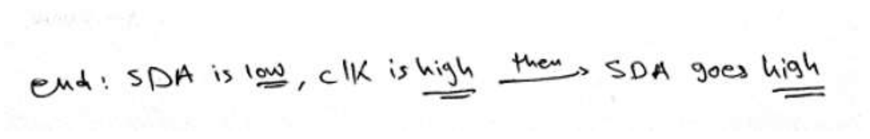
The start condition as described above



begin: SDA is high, clK is high $\xrightarrow{\text{then}}$ SDA goes low

```
// Stop condition
void Stop() {
  digitalWrite(SDA_PIN, LOW);
  delayMicroseconds(4);
  digitalWrite(SCL_PIN, HIGH);
  delayMicroseconds(4);
  digitalWrite(SDA_PIN, HIGH);
}
```

The stop condition as described above

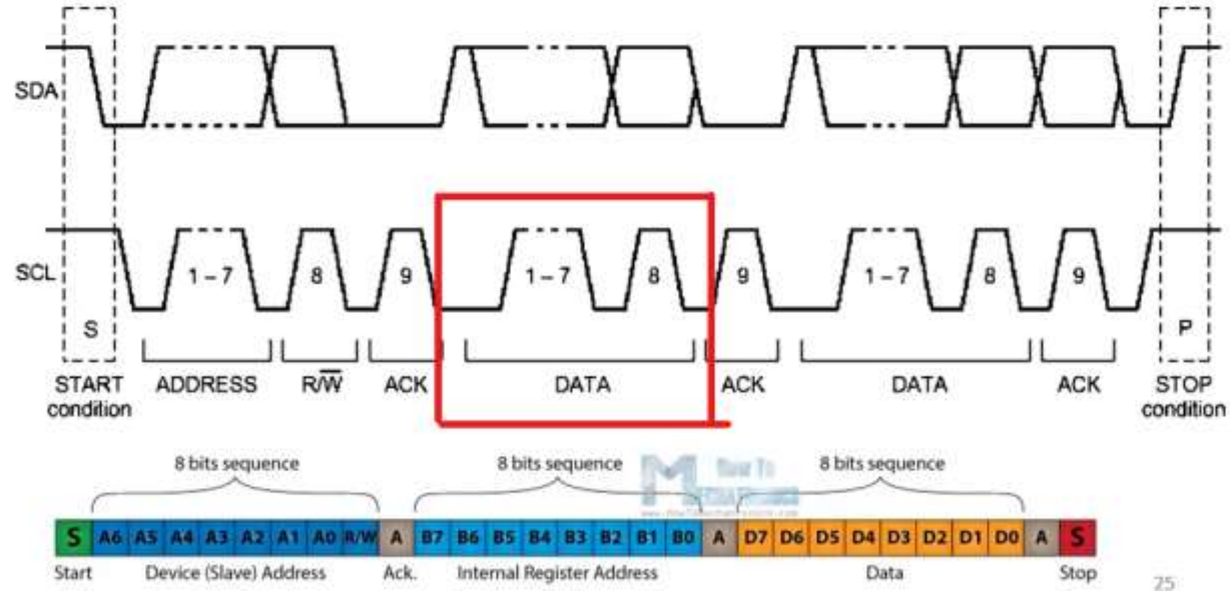


end: SDA is low, clK is high $\xrightarrow{\text{then}}$ SDA goes high

The write function,

```
void Write(byte data) {
  for (byte i = 0; i < 8; i++) {
    digitalWrite(SDA_PIN, (data & 0x80) != 0);
    data <<= 1;
    digitalWrite(SCL_PIN, HIGH);
    delayMicroseconds(4);
    digitalWrite(SCL_PIN, LOW);
    delayMicroseconds(4);
  }

  // Ack bit
  pinMode(SDA_PIN, INPUT);
  digitalWrite(SDA_PIN, HIGH);
  digitalWrite(SCL_PIN, HIGH);
  delayMicroseconds(4);
  digitalWrite(SCL_PIN, LOW);
  pinMode(SDA_PIN, OUTPUT);
}
```

For loop (8 loops), because we send 8-bits (1 byte).



The data is sent from most significant bit to lowest (the bitwise AND operation and then shifts the data left by one-bit concept is used).

For each bit to be sent, the SCL line goes from high to low, after sending the 8 bits… reads the ACK bit from the SDA line by pulse the clk line (high to low).

The read function,

```
char Read() {
  char data = 0;
  pinMode(SDA_PIN, INPUT);
  for (byte i = 0; i < 8; i++) {
    digitalWrite(SCL_PIN, HIGH);
    delayMicroseconds(4);
    data <<= 1;
    if (digitalRead(SDA_PIN)) {
      data |= 1;
    }
    digitalWrite(SCL_PIN, LOW);
    delayMicroseconds(4);
  }

  // Send ACK
  pinMode(SDA_PIN, OUTPUT);
  delayMicroseconds(4);
  digitalWrite(SDA_PIN, LOW);
  digitalWrite(SCL_PIN, HIGH);
  delayMicroseconds(4);
  digitalWrite(SCL_PIN, LOW);
  pinMode(SDA_PIN, INPUT);

  return data;
}
```

Same, the data is received bit by bit starting from the most significant (8-bits).

For each bit to be received, read it from the SDA line while the SCL line goes from high to low.

Finally, after reading the byte… sends ACK to the slave.

So the above functions to be used in the loop for communication

```
void loop() {
  Start();
  Write(SLAVE_ADDRESS << 1);
  Write('j'); // Send char 'j'
  Stop(); // stop
  Serial.println("Master: 'j' sent."); // now... wait for ACK from the slave

  delay(2000);

  Start();
  Write((SLAVE_ADDRESS << 1) | 1); // Send slave address with read bit
  char data = Read();
  Stop();
  Serial.print("Master received: ");
  Serial.println(data);

  delay(5000);
}
```

Send j and wait for 2 second to receive k from the master and this process to be repeated every 5s.

Slave

The slave code is clear and the concept is as described above in the write-handed papers, but I wish to explain only the read and write functions of the slave.

Read

```
byte Read() {
  byte data = 0;
  pinMode(SDA_PIN, INPUT);
  for (byte i = 0; i < 8; i++) {
    while (digitalRead(SCL_PIN) == LOW);
    data <<= 1;
    if (digitalRead(SDA_PIN)) {
      data |= 1;
    }
    while (digitalRead(SCL_PIN) == HIGH);
  }
  return data;
}
```

In the for loop (8-loops) [each loop for each bit (8 bit in total)]

➔ Waits for the clock line to go high (while the clock line is LOW), then shifts the current data left by one bit (data <<= 1). It reads the bit from the SDA line and sets it in the data if the bit is high.

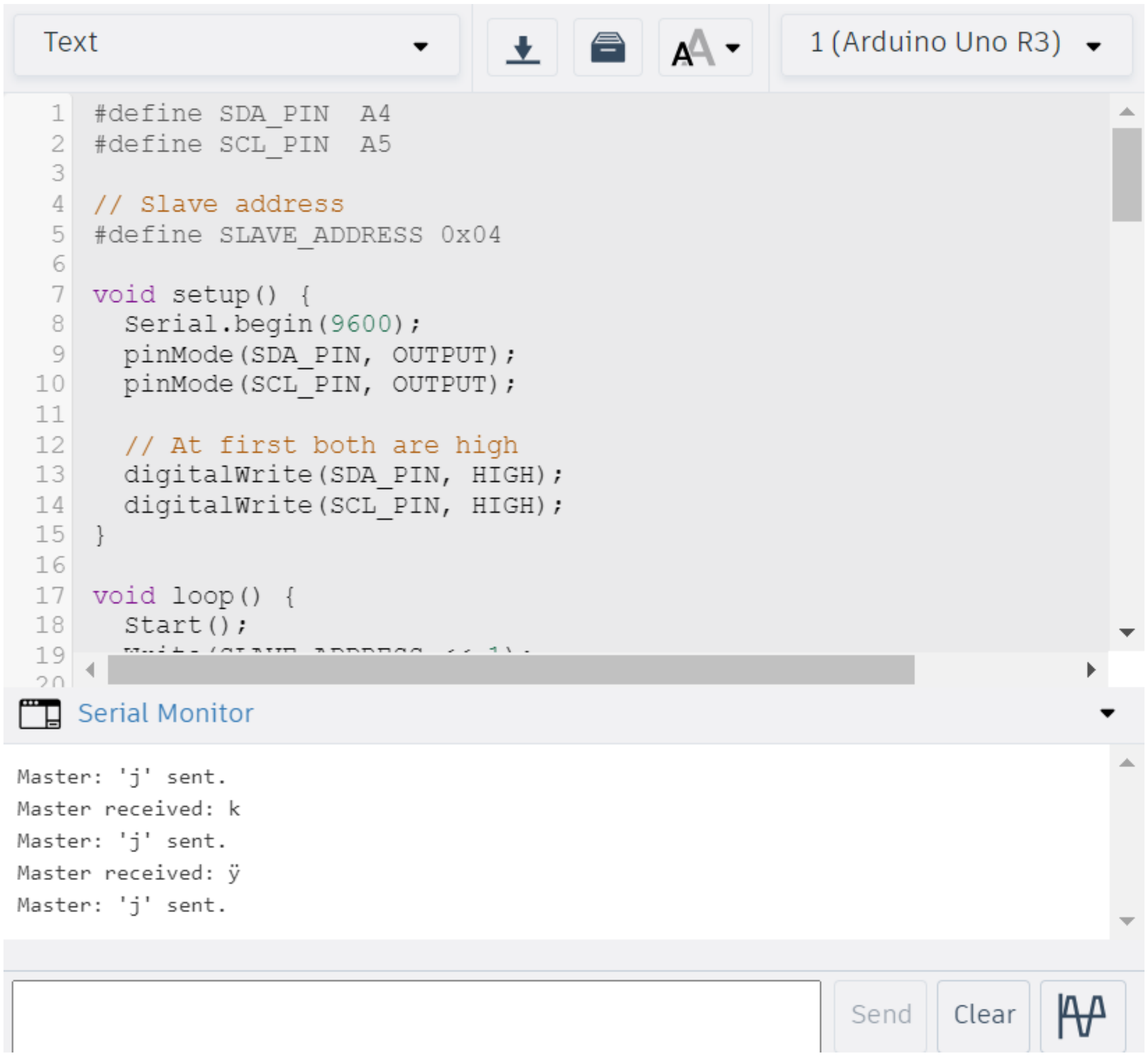
Finally, it waits for the clock line to go low.

Note that the Ack here is defined as a separate function, then to be called in the loop().

The **write** function is the same as the write function in the master.

Simulation result

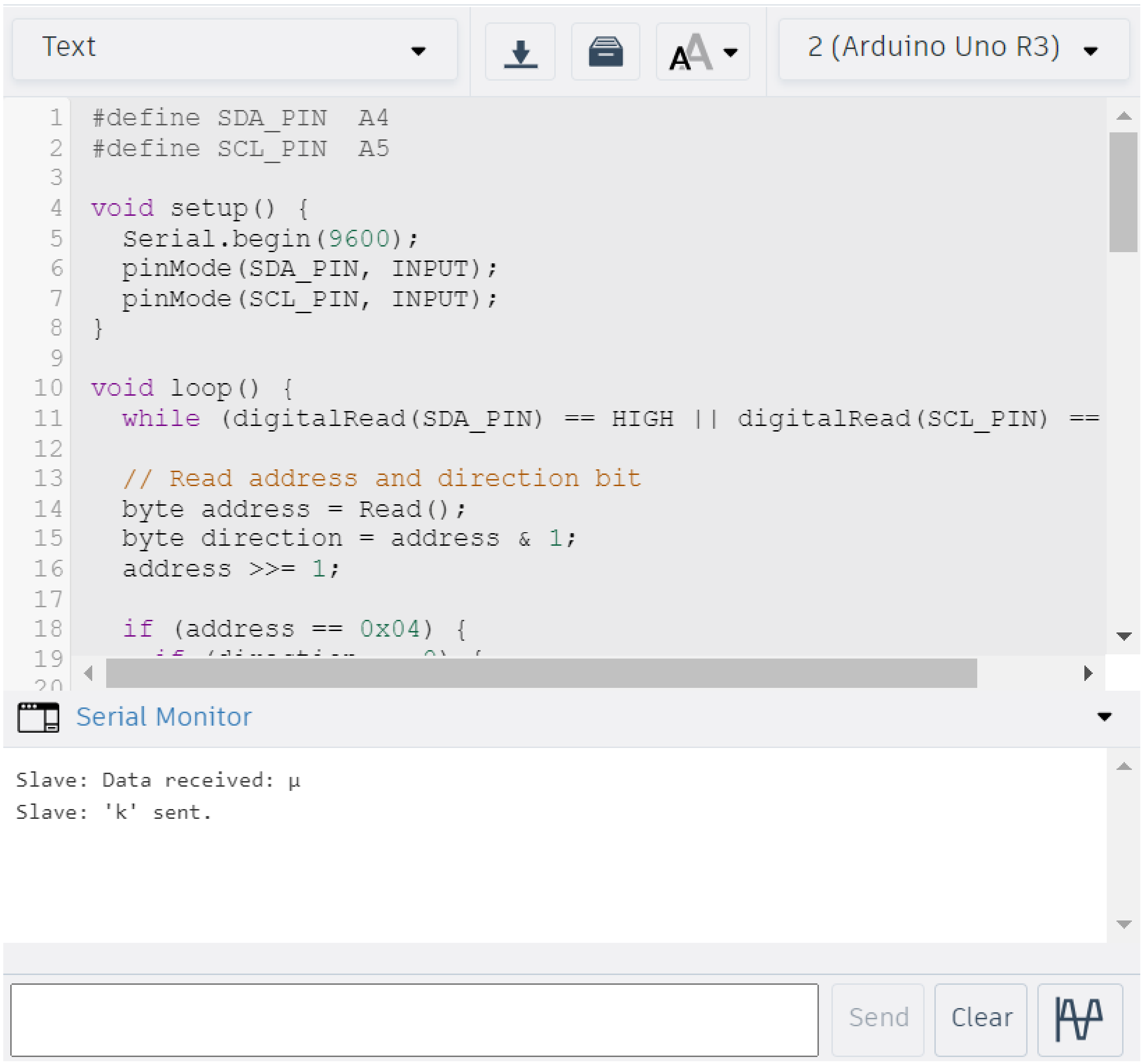
Master



```
#define SDA_PIN  A4
#define SCL_PIN  A5

// Slave address
#define SLAVE_ADDRESS 0x04

void setup() {
  Serial.begin(9600);
  pinMode(SDA_PIN, OUTPUT);
  pinMode(SCL_PIN, OUTPUT);

  // At first both are high
  digitalWrite(SDA_PIN, HIGH);
  digitalWrite(SCL_PIN, HIGH);
}

void loop() {
  Start();
```

```
Master: 'j' sent.
Master received: k
Master: 'j' sent.
Master received: ÿ
Master: 'j' sent.
```

Slave



```
#define SDA_PIN  A4
#define SCL_PIN  A5

void setup() {
  Serial.begin(9600);
  pinMode(SDA_PIN, INPUT);
  pinMode(SCL_PIN, INPUT);
}

void loop() {
  while (digitalRead(SDA_PIN) == HIGH || digitalRead(SCL_PIN) ==

  // Read address and direction bit
  byte address = Read();
  byte direction = address & 1;
  address >>= 1;

  if (address == 0x04) {
```

```
Slave: Data received: µ
Slave: 'k' sent.
```

The first process is 100% correct, but after the first process the result is not correct, I think the problem is timing problem, I tried to solve the problem by using the delay to be in **micro seconds** instead of the regular delay which in mille seconds, and also I have reduced the time between the send and the receive and also the time between one process and another, but unfortunately the same.

Conclusion

In conclusion, this activity has given me a much better understanding of the I2C protocol, especially how communication works both with and without using a library. This was my first time trying to build a library from scratch, and I'm proud to have successfully done it. Although I encountered some timing issues, I consider this normal for a beginner. I'm confident this experience is just the beginning of my journey in creating libraries in the future. After all, if computer engineers don't create libraries, who will? Certainly, it's the role of computer engineers, who dive deep into both coding and hardware.

Something nice I have noticed when I studied this protocol and specially after done this activity.

We can increase the resolution of the Arduino by connecting for example my own 16-bit ADC and make it communicate with Arduino by I2C protocol … instead of the 10-bit Arduino ADC and jump from 1023 to 65,536.

References

[1]: Your I2C protocol slides.

My Tinkercards

With library: https://www.tinkercad.com/things/6Thm84ZqKxb-i2c-communication-with-library?sharecode=tE5uXih3aaPR4k33gWQZswExNnmxZN4cdcXYcva3QwA

Without library: https://www.tinkercad.com/things/89AloFjncDL-i2c-communication-without-library?sharecode=YFRiuF5tUpZW9CO3vCENlQWCqGOc_mGB8cjdMOcBn-g