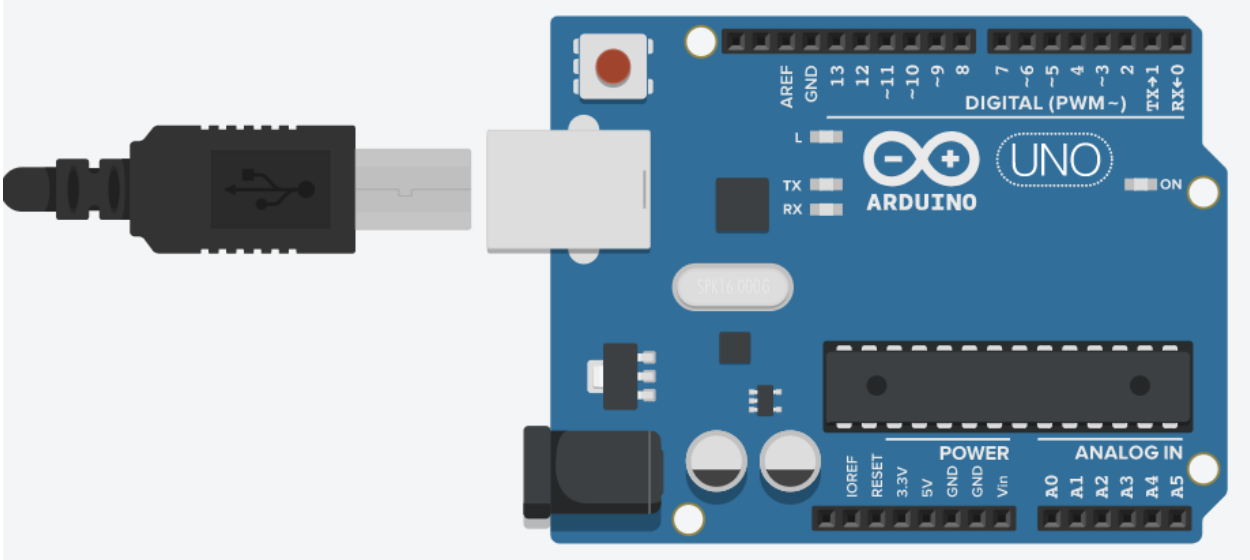
EXPERIMENT 7

Hemish Shah

J056

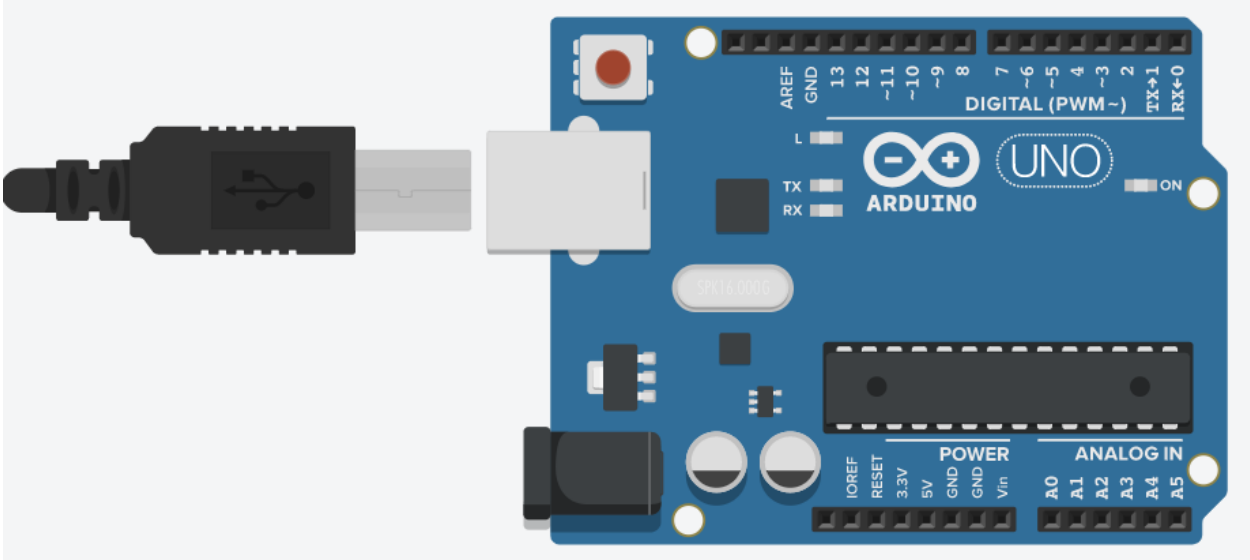
1)EEPROM separate programs using read(), write(), get(), put(), update()

(i)read()



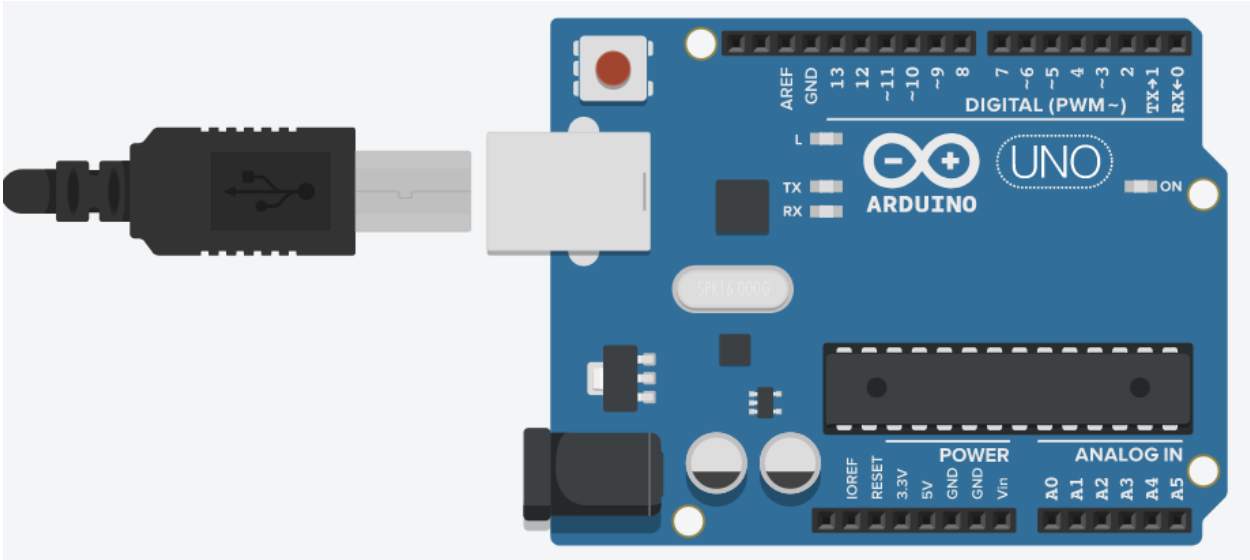

```
#include < EEPROM.h>
int i = 0;
int value;

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

void loop()
{
  value = EEPROM.read(i);
  Serial.print(i);
  Serial.print("\t");
  Serial.print(value);
  Serial.println();
  i = i + 1;
  if (i == 512)
  {
    i = 0;
  }
  delay(1000);
}
```

(ii)write()



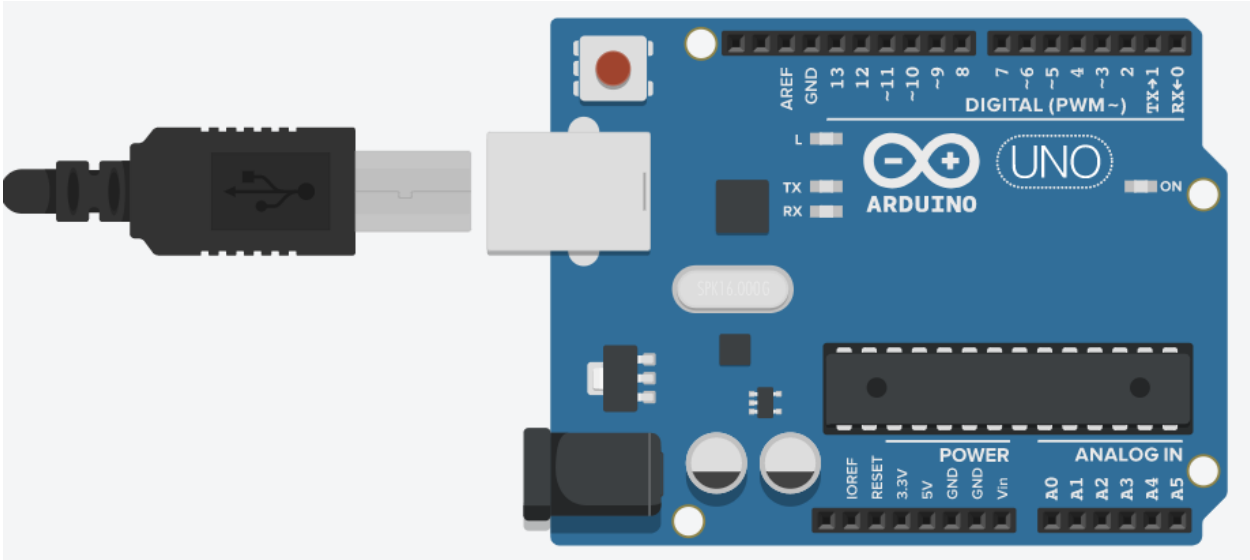

```
#include <EEPROM.h>

void setup()
{
  for (int i = 0; i < 255; i++)
  {
    EEPROM.write(i, i);
  }
}

void loop(){}
```

(iii)get()



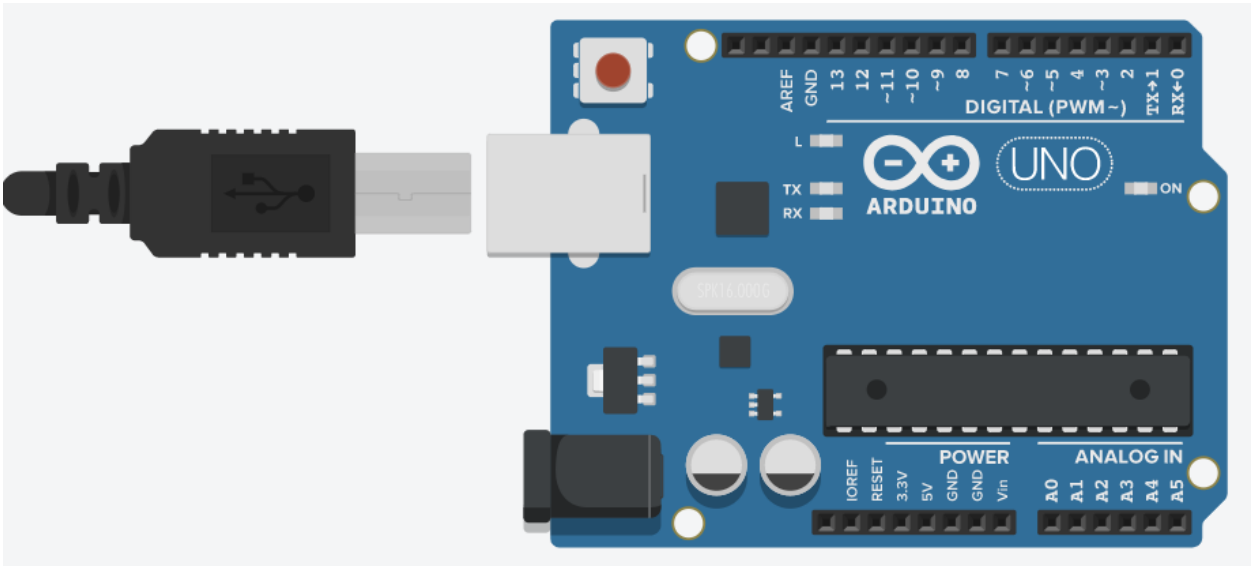
```ino
#include <EEPROM.h>
struct MyObject
{
  float field1;
  byte field2;
  char name[10];
};

void setup()
{
  float f = 0.00f;
  int eeAddress = 0;
  Serial.begin( 9600 );
  while (!Serial)
  {
    ;
  }
  Serial.print( "Read float from EEPROM: " );
  EEPROM.get( eeAddress, f );
  Serial.println( f, 3 );

  eeAddress = sizeof(float);
  MyObject customVar;
  EEPROM.get( eeAddress, customVar );

  Serial.println("Read custom object from EEPROM: " );
  Serial.println( customVar.field1 );
  Serial.println( customVar.field2 );
  Serial.println( customVar.name );
}

void loop(){}
```

(iv)put()



```
#include <EEPROM.h>
struct MyObject
{
  float field1;
  byte field2;
  char name[10];
};

void setup()
{
  Serial.begin(9600);
  while (!Serial)
  {
    ;
  }
  float f = 123.456f;
  int eeAddress = 0;
  EEPROM.put(eeAddress, f);
  Serial.println("Written float data type!");
  MyObject customVar =
  {
    3.14f,
    65,
    "Working!"
  };
  eeAddress += sizeof(float);
  EEPROM.put(eeAddress, customVar);
  Serial.print("Written custom data type! \n\nView the example sk
}

void loop(){}
```

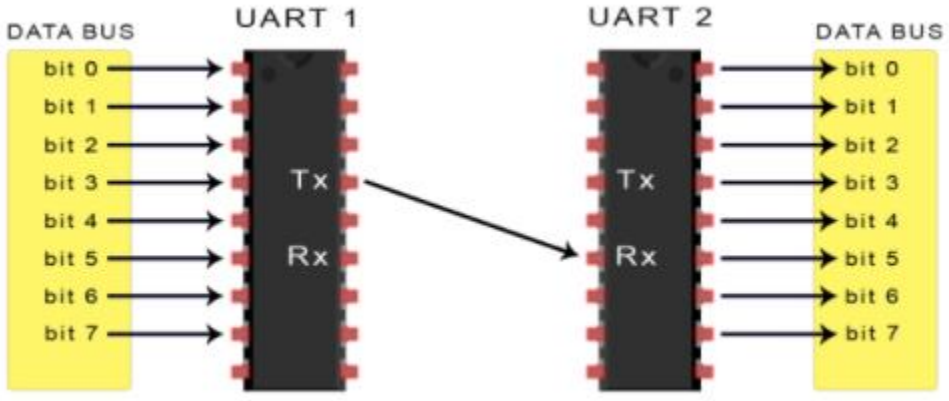
(v)update()



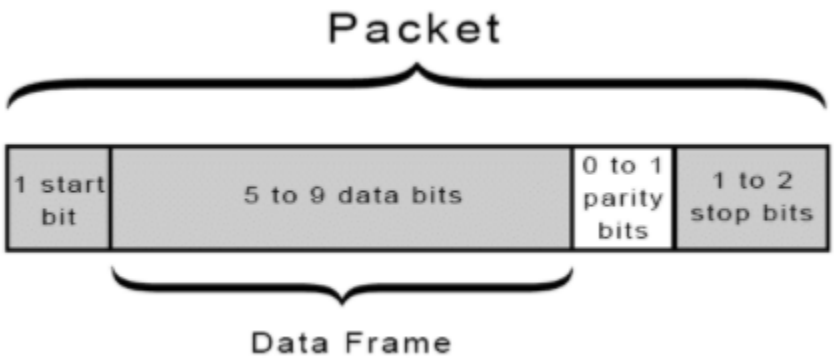
```
#include <EEPROM.h>

void setup()
{
  for (int i = 0; i < 255; i++)
  {
    EEPROM.update(i, i);
  }
  for (int i = 0; i < 255; i++)
  {
    EEPROM.update(3, 12);
  }
}

void loop(){}
```

2) UART - sending and receiving messages Or UART study the protocol and list inbuilt functions for sending and receiving data

UART stands for Universal Asynchronous Receiver/Transmitter. It's not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART's main purpose is to transmit and receive serial data. The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller. Data is transferred from the data bus to the transmitting UART in parallel form. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet. Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end:



UART transmitted data is organized into *packets*. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional *parity* bit, and 1 or 2 stop bits:



To use UART serial in the Arduino IDE, we will have to initialize the serial module.

Serial.begin(9600); // Start the serial module with a baud rate of 9600 bps and the default configuration

To write using the UART serial we use the write command. It sends only one byte of data at a time. A single value of 0-255.

Serial.write(70); // transmits the value of 70

To print the strings line after line we can use print command.

char array1[5] = {4, 8, 16, 23, 42};

Serial.print(array1); // Will transmit the values 4, 8, 16, 23, and 42

To get first byte of the serial buffer

int receivedByte;

receivedByte = Serial.read(); // Returns the first byte from the serial buffer

To get return the number of bytes that are currently available in the serial buffer.

```
if (Serial.available() > 0)
{
  receivedByte = Serial.read();
}
```

To clear all bytes from the Serial buffer

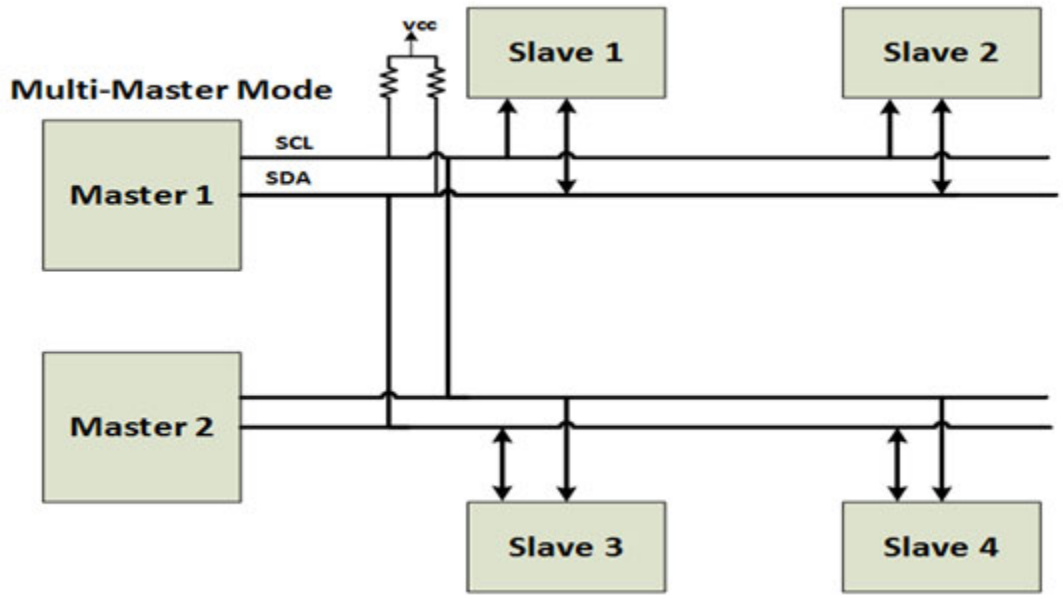
Serial.flush();

To return the first byte in the serial buffer but does not remove it.

Serial.peak();

3) I2C - sending and receiving messages Or I2C study the protocol and list inbuilt functions for sending and receiving data

The inter-integrated circuit or I2C Protocol is a way of serial communication between different devices to exchange their data with each other. It is a half-duplex bi-directional two-wire bus system for transmitting and receiving data between masters (M) and slaves (S). These two wires are Serial clock line or SCL and Serial data line or SDA. Masters and Slaves play important role in I2C communication. Master is the one which initiates a communication, generates a clock and terminates the communication and Slave is the one which is handled by master and acts according to the master command. It can also be possible that multiple masters can communicate with multiple slaves. In Figure 2, we can see two Masters are communicating with multiple slaves over I2C Bus.



There are mainly four modes which define the data rate of the I2C communication system:

Types of Mode	Communication Speed
Standard Mode	100Kbit/s
Fast Mode	400 Kbit/s
Fast Mode plus	1 Mbit/s
High-Speed Mode	3.4Mbit/s

It is used to write (transmit) data to the master or slave device.

Wire.write('hello'); //sends the string to the slave device

This function is used by a master or slave to check the requested data is available or not. It returns the no. of bytes available

Wire.available();

It is used to read the requested data by master from slave or read the data transmitted from a master to a slave.

Wire.read();

It initiates the Wire library and joins the bus as a master.

Wire.begin();

This function begins a transmission with the I2C slave device having specified slave address.

Wire.beginTransmission (slave address);

It initiates the Wire library and joins the I2C bus as a slave with specified address

Wire.begin(address);

The handler functions to be called when a slave device receives a transmitted data from a master.

Wire.onReceive(handler);