Lab-06: Simulation of SD Card Module in Proteus

1. **Objectives: After this lab, the learner will be able to:**
   - Interface SD module with Arduino board
   - Interface
2. **Facilities**
   - Proteus software
   - Arduino IDE
3. **Prerequest**
   - Basic electronic
   - C programming
4. **Introduction**

   In this Lab, we will be going to discuss simple proteus simulation that how to get started with Arduino and SD card (microSD card).

   I will be explaining how to read information about an SD card in the later part of this topic to make you familiar with the concept. In this example, we will learn how to create an image file using winimage software andselecting certain parameters such as volume type, free space, and other information using winimage & the SD library, and sending it over the serial port. Then we will mimic the example in proteus to arrive at best required results that resembles the practical experiment. The SD card library for Arduino is quite excellent, and it makes interacting with SD cards very straightforward. Reading and writing to SD cards is possible with the Arduino SD library. It is based on William Greiman's sdfatlib. On ordinary SD and SDHC cards, the library supports FAT16 and FAT32 file systems. For file names, it utilises short 8.3 names.

   SPI is used to communicate between the microcontroller and the SD card, and it is located on digital pins 11, 12, and 13 (on most Arduino boards) or 50, 51, and 52 (on certain Arduino boards) (Arduino Mega). A second pin is also required to choose the SD card. The hardware SS pin – pin 10 on most Arduino boards or pin 53 on the Mega – or another pin specified in the SD.begin() call can be used.
5. Arduino and SD card interfacing

This topic shows how to get started with Arduino and SD card (microSD cards). It shows how to read information about SD card connected to the Arduino board. The example reports volume type, free space and other information using the SD library, sending it over the serial port. Also Proteus simulation of the Arduino and SD card is available with a small video.

Arduino has a very nice SD card library, with this library the interfacing is very simple. The Arduino SD library allows for reading from and writing to SD cards. It is built on sdfatlib by William Greiman. The library supports FAT16 and FAT32 file systems on standard SD cards and SDHC cards. It uses short 8.3 names for files.

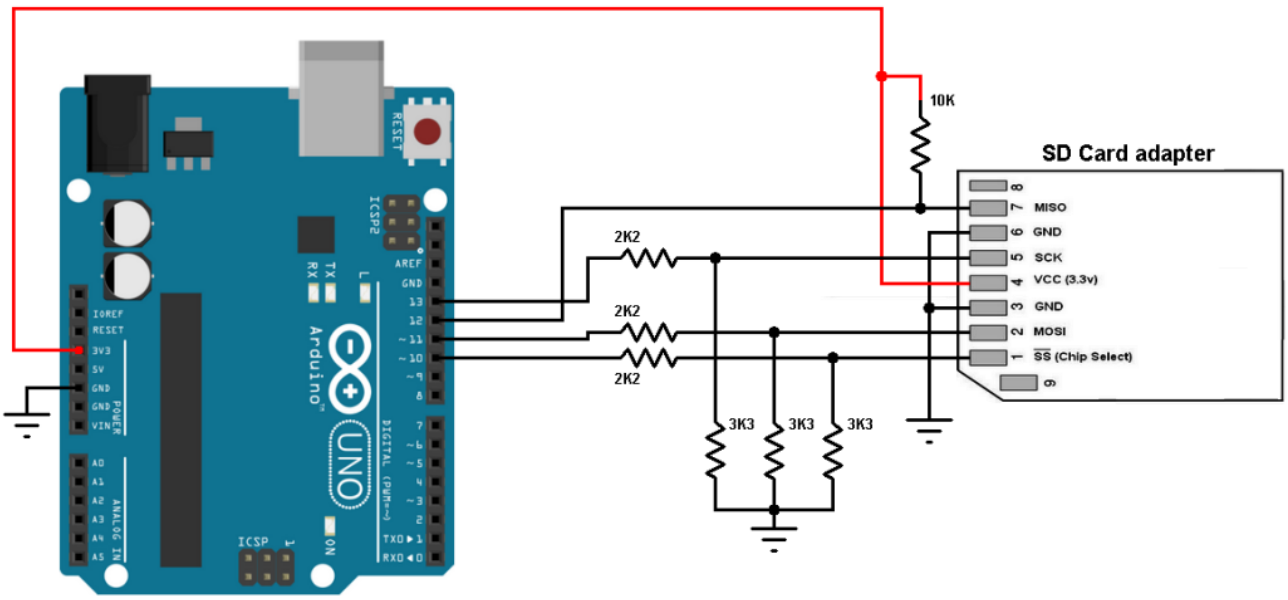
The communication between the microcontroller and the SD card uses SPI, which takes place on digital pins 11, 12, and 13 (on most Arduino boards) or 50, 51, and 52 (Arduino Mega). Additionally, another pin must be used to select the SD card. This can be the hardware SS pin – pin 10 (on most Arduino boards) or pin 53 (on the Mega) – or another pin specified in the call to SD.begin().

Hardware Required:

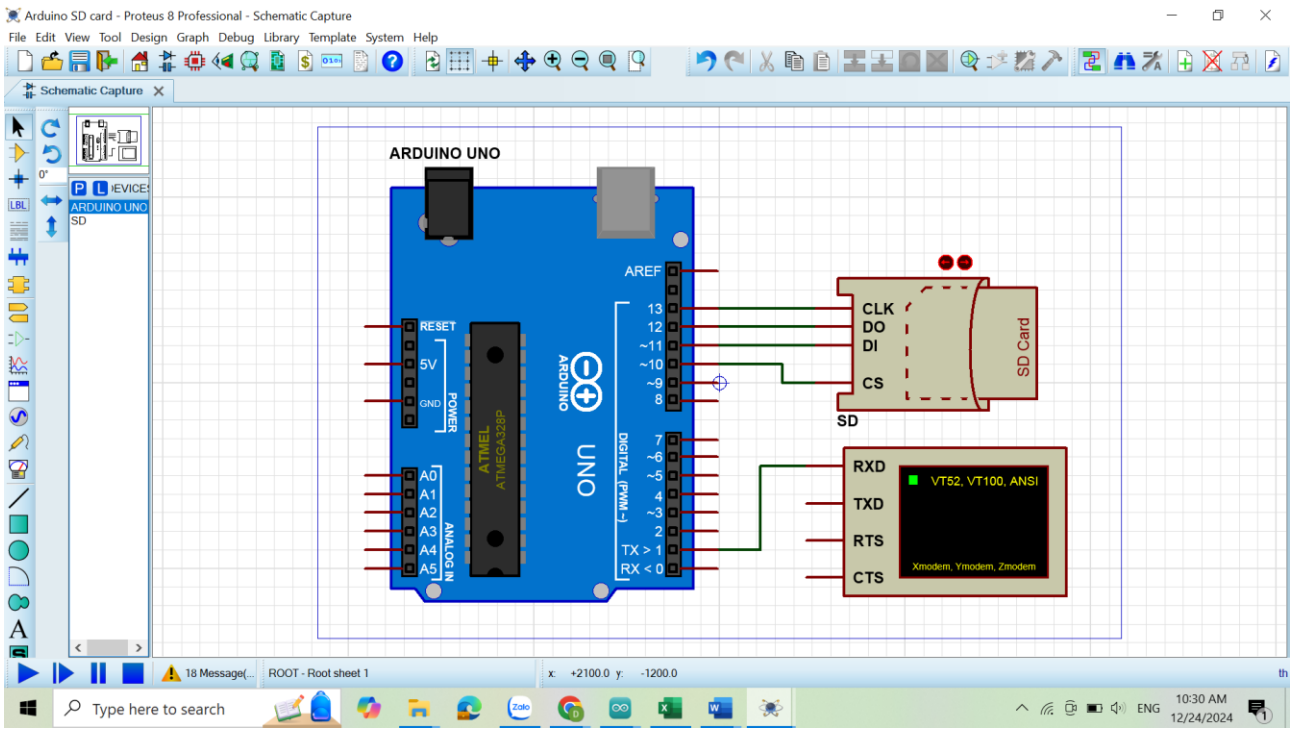
- Arduino board
- SD card with FAT16 or FAT32 file system
- SD card socket (connector)
- 10K ohm resistor
- 3 x 3.3K ohm resistor
- 3 x 2.2K ohm resistor
- Breadboard
- Jumper wires

The circuit:

Example circuit schematic diagram is shown below.



(Grounded terminals are connected together)



The SD card is supplied from the Arduino board with 3.3V.

In the circuit there are 3 voltage dividers, each one consists of 2.2K and 3.3K resistors, they are used to step down 5V that comes from the arduino into 3V which is sufficient for the SD card signals. The voltage dividers are used for SD card signals: SCK (serial clock), MOSI (master out slave in) and SS (chip select). The Arduino sends these signals from pins 13, 11 and 10 respectively. The SD card MISO is connected directly to the arduino because this path is used by the SD card to send data to the arduino (with voltage of 3.3V).
Connecting the SD card directly to the arduino without voltage level converters or voltage dividers may damage it.

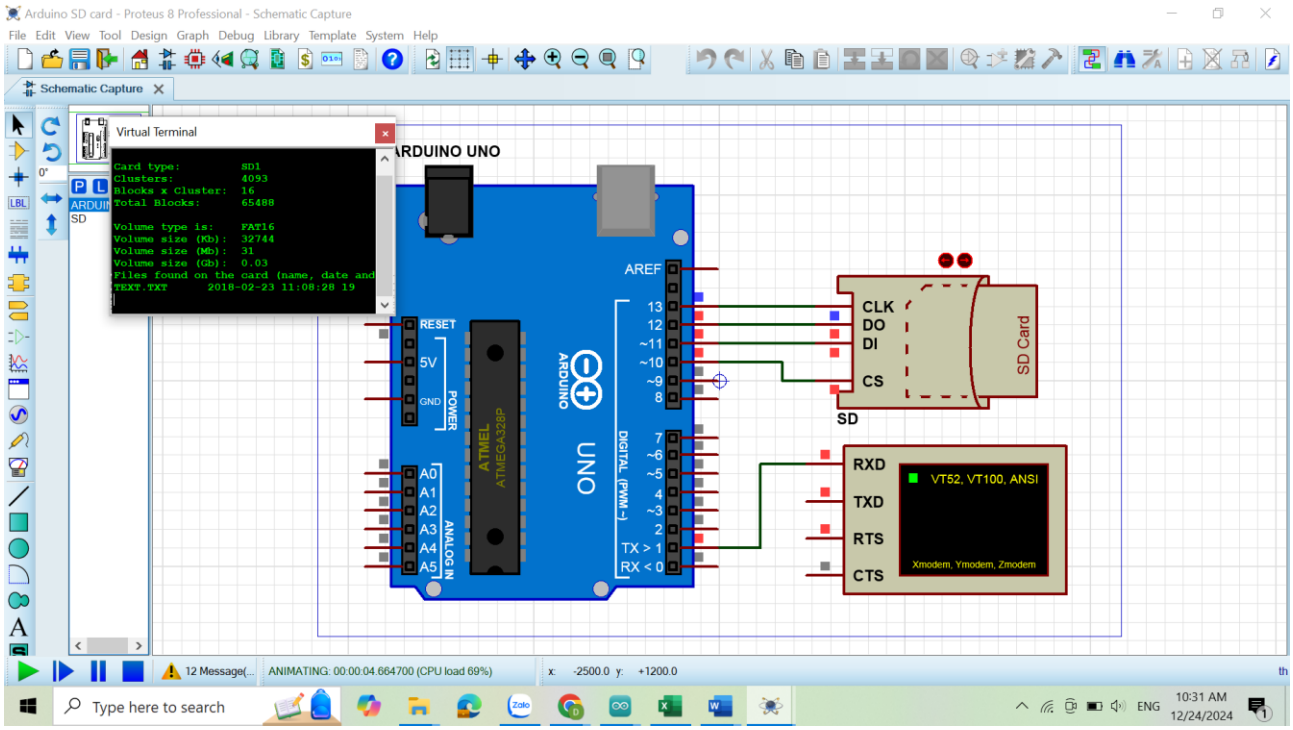
Arduino Code:

I got the code below from arduino examples (with minor modifications), it collects information about the SD card connected to the Arduino and print it to Arduino IDE serial monitor.

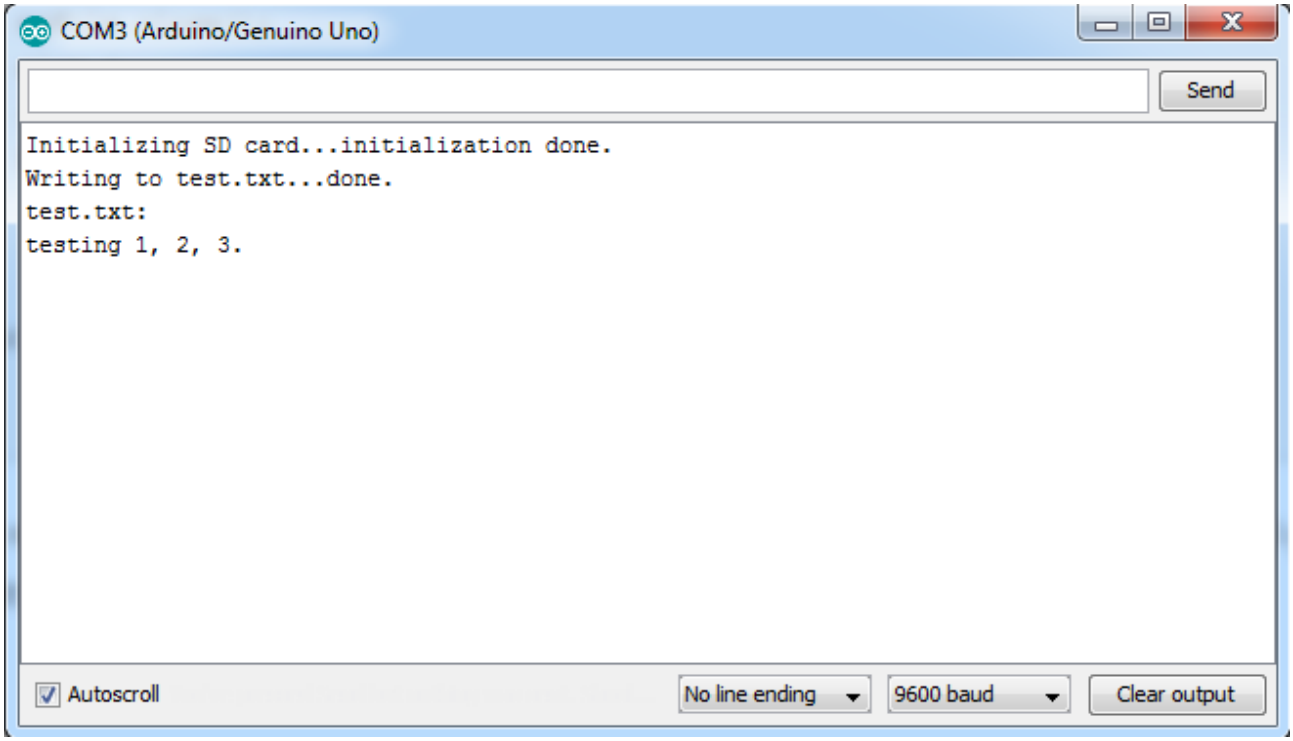
```
// Interfacing Arduino with SD card example (get SD card info)
// http://simple-circuit.com/

// include the SD library:
#include <SPI.h>
#include <SD.h>

// set up variables using the SD utility library functions:
Sd2Card card;
```

```
SdVolume volume;
SdFile root;
void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  Serial.print("\nInitializing SD card...");

  // we'll use the initialization code from the utility libraries
  // since we're just testing if the card is working!
  if (!card.init()) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card inserted?");
    Serial.println("* is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your shield or module?");
    while (1);
  } else {
    Serial.println("Wiring is correct and a card is present.");
  }

  // print the type of card
  Serial.println();
  Serial.print("Card type:         ");
  switch (card.type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Unknown");
  }

  // Now we will try to open the 'volume'/'partition' - it should be FAT16 or FAT32
  if (!volume.init(card)) {
    Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the
card");
    while (1);
  }

  Serial.print("Clusters:          ");
```

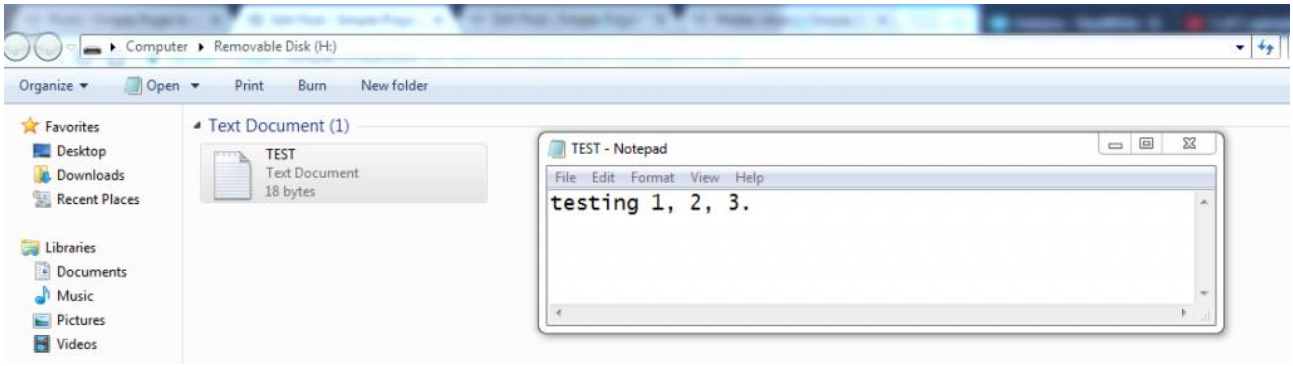
```
  Serial.println(volume.clusterCount());
  Serial.print("Blocks x Cluster:  ");
  Serial.println(volume.blocksPerCluster());

  Serial.print("Total Blocks:      ");
  Serial.println(volume.blocksPerCluster() * volume.clusterCount());
  Serial.println();

  // print the type and size of the first FAT-type volume
  uint32_t volumesize;
  Serial.print("Volume type is:    FAT");
  Serial.println(volume.fatType(), DEC);

  volumesize = volume.blocksPerCluster();    // clusters are collections of blocks
  volumesize *= volume.clusterCount();       // we'll have a lot of clusters
  volumesize /= 2;                  // SD card blocks are always 512 bytes (2 blocks are 1KB)
  Serial.print("Volume size (Kb):  ");
  Serial.println(volumesize);
  Serial.print("Volume size (Mb):  ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.print("Volume size (Gb):  ");
  Serial.println((float)volumesize / 1024.0);

  Serial.println("\nFiles found on the card (name, date and size in bytes): ");
  root.openRoot(volume);

  // list all files in the card with date and size
  root.ls(LS_R | LS_DATE | LS_SIZE);
}

void loop(void) {
}
```



As a result, I placed a samsung 2 GB micro SD card formatted with FAT16 file system and I got the result shown below (Arduino IDE serial monitor):



Opening the file with Microsoft windows:



6. Components Required

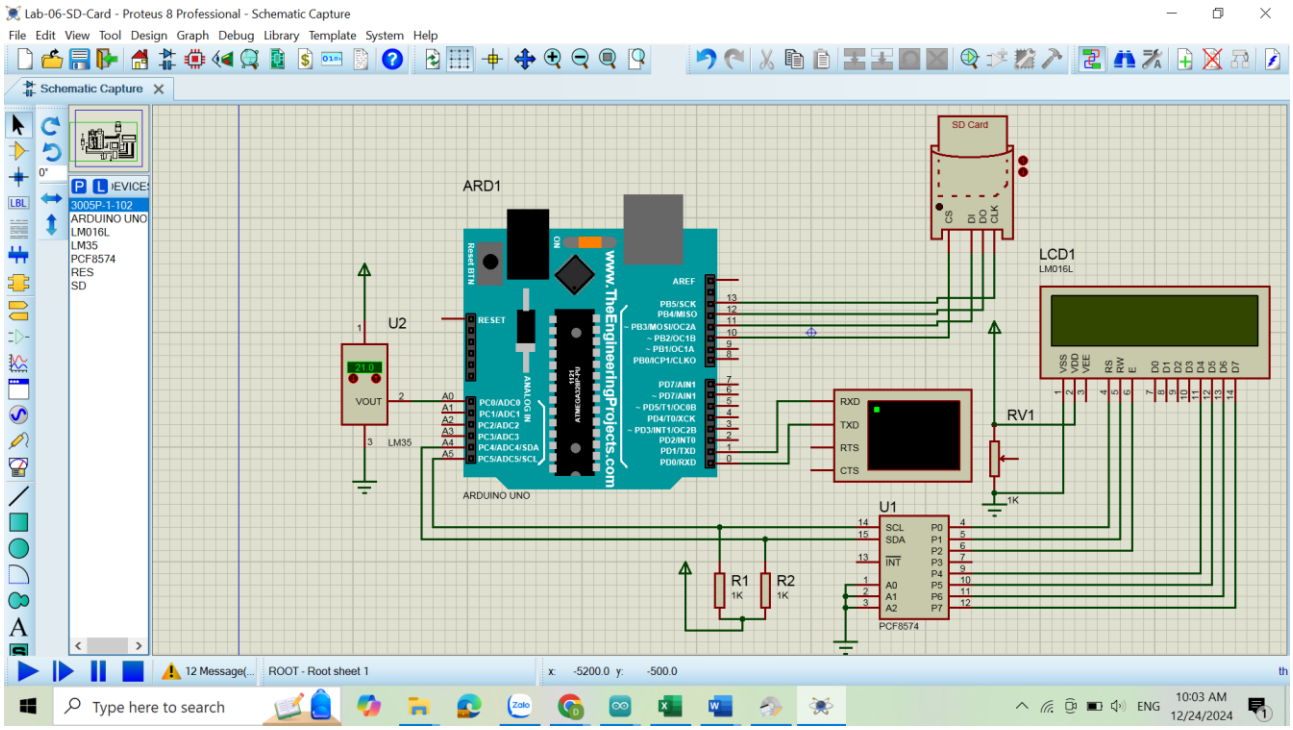
6.1. Hardware

- Arduino board
- SD card with FAT16 or FAT32 file system
- LM35
- Virtual Terminal (as a serial monitor)
- PCF8574 I2C LCD Interface Module
- LM016L (LCD) Display
- R1 & R2 = 10k Ohm

6.2. Software

- Proteus
- Arduino IDE
- Winimage

7. Shematic



8. Implement

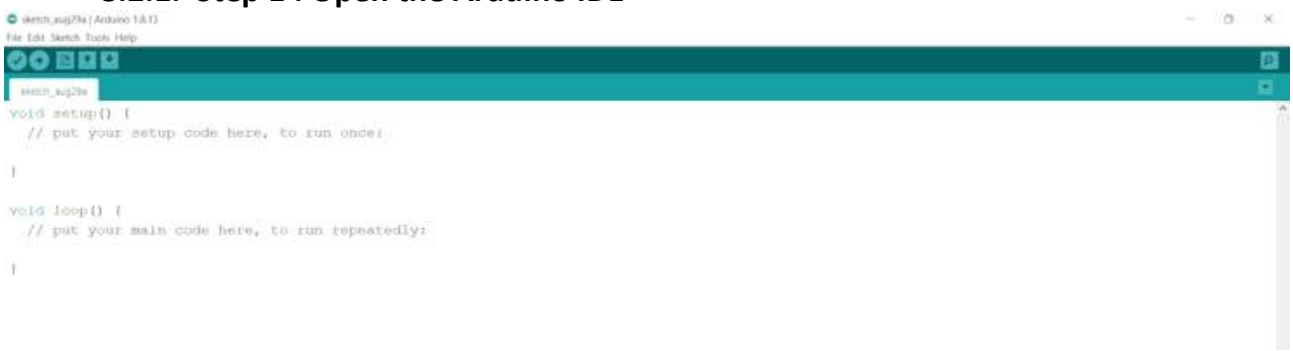
8.1. Prerequisite Exercise

- Before we proceed to the actual part simulation first we will warm up with simple exercise of downloading, installation and setting up libraries for Arduino IDE & Proteus.
- i. Installing Winimage
- Our first step is to download the winimage from internet and set the path of download to be easily accessible for later part. Select the appropriate latest.exe file from the official site as per the system requirement.

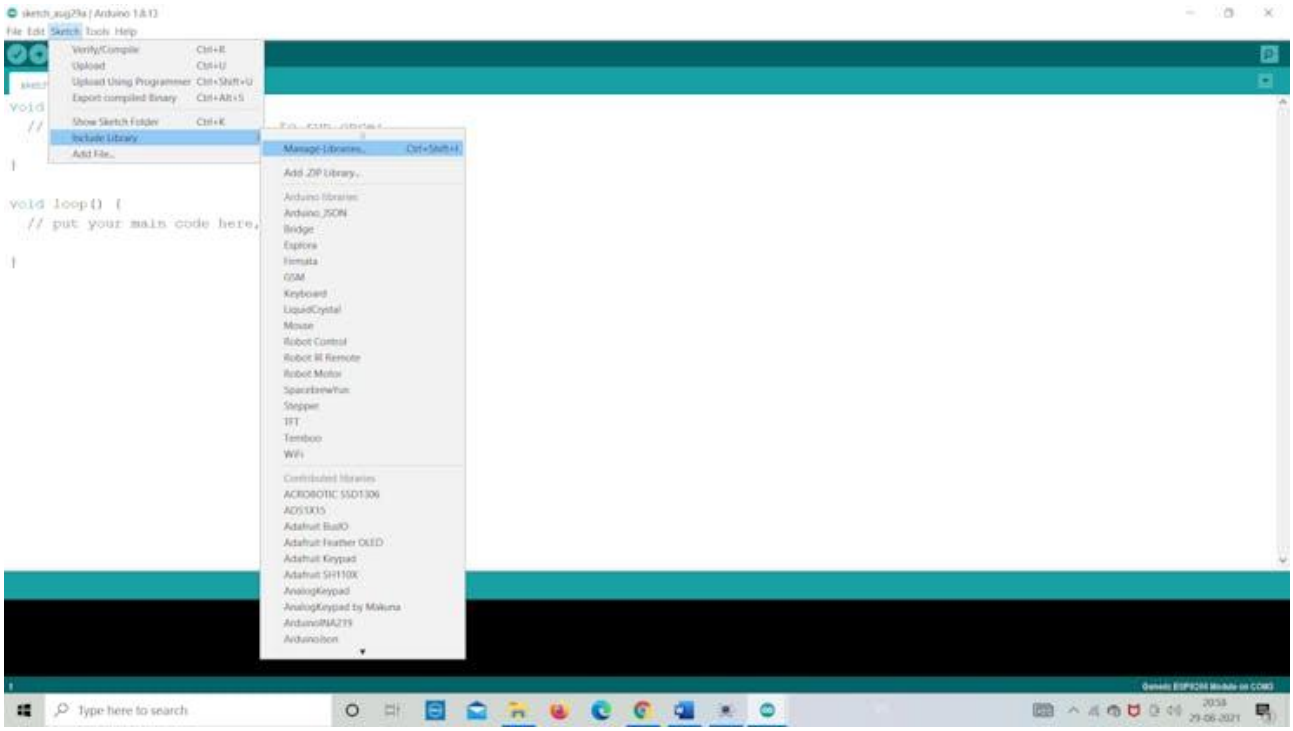
- i. Adding SD card library in IDE
- Now we will download the SD library from IDE. Follow the steps shown in the image below to know more about how to download the SD library for Arduino IDE. This library we will be using for programming the Arduino for SD card module to store and retrieve data.

8.2. Steps

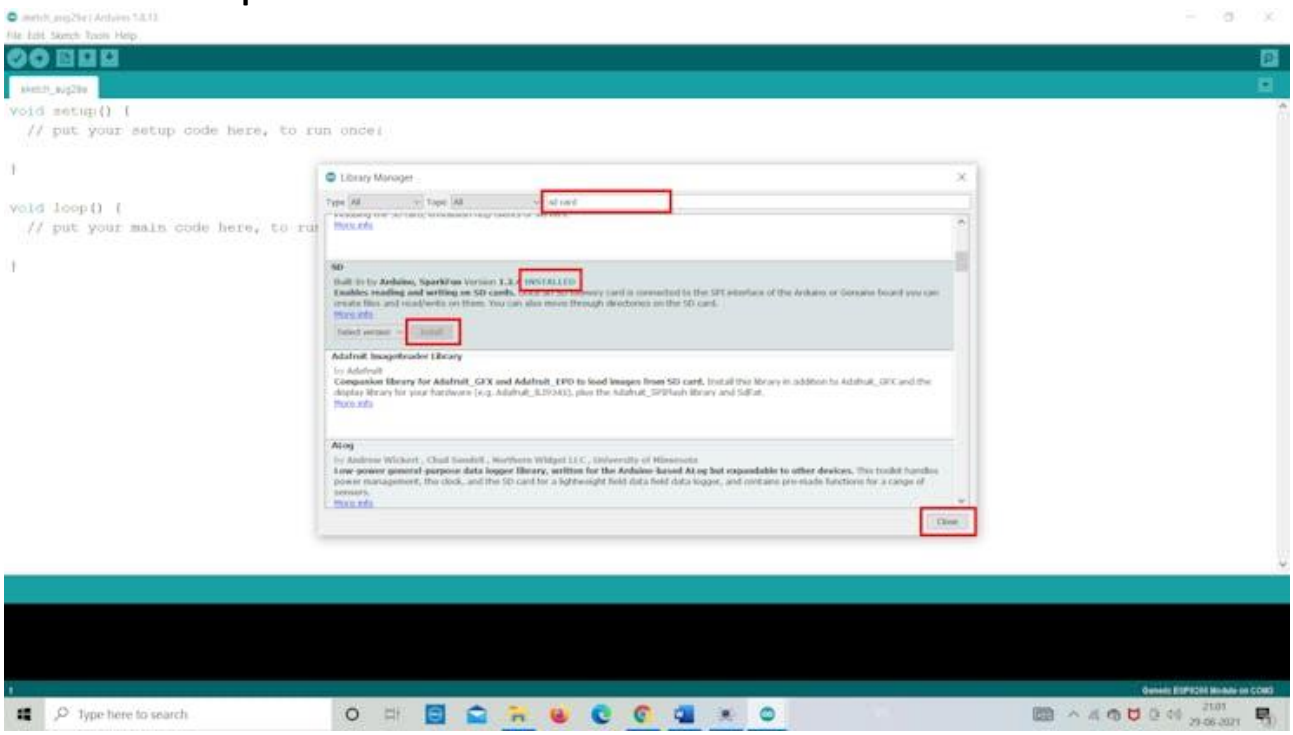
8.2.1. Step 1 : Open the Arduino IDE

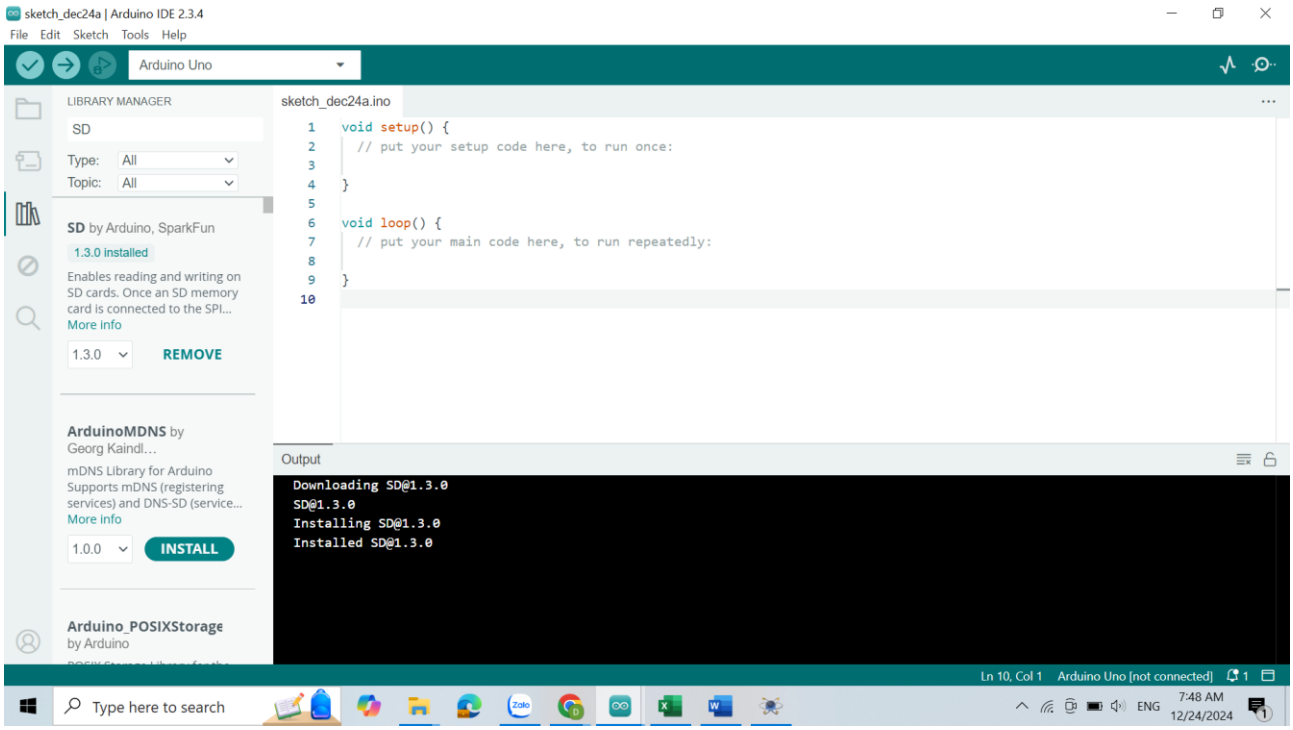


8.2.2. Step 2 : Go to “Sketch” >> “Include Library” >> “Manage Libraries”



8.2.3. Step 3 : Search “SD” in Search bar and “Install” it from there

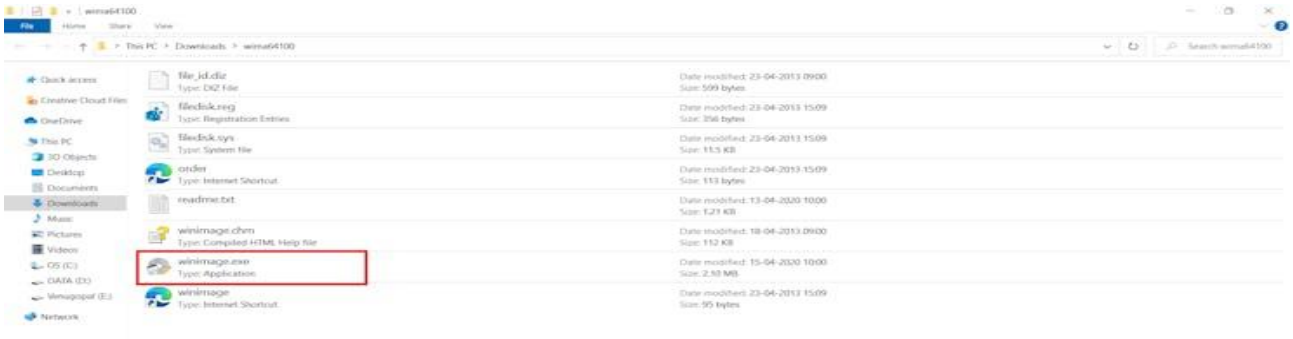




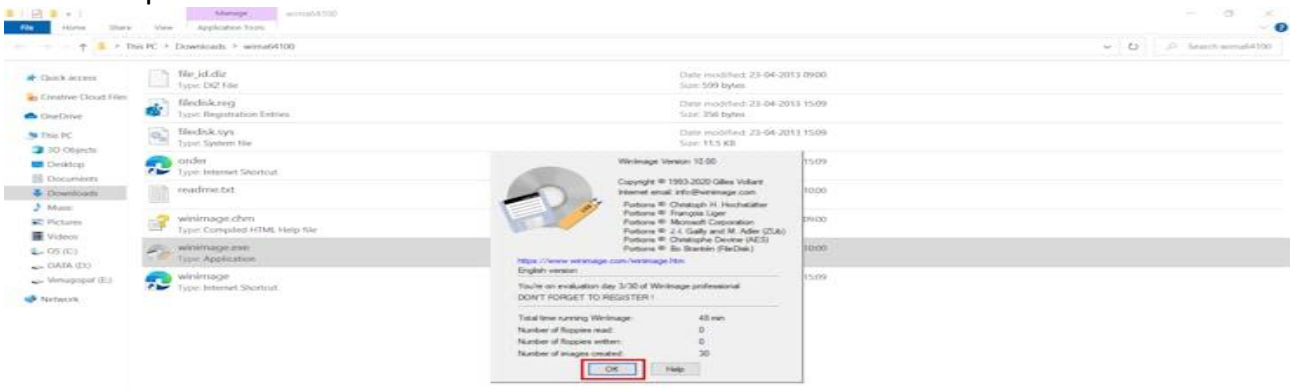
Notes: Add libray

```
#include <SD.h>

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <TimeLib.h>
```

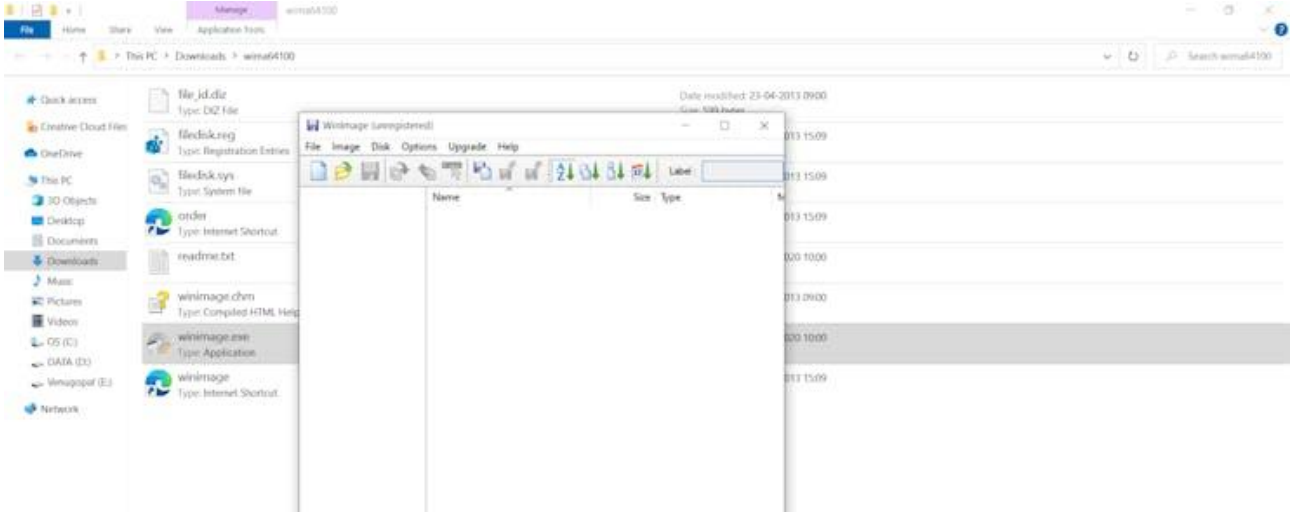
- ii. Creating Winimage image file
- Next step is to create image file using winimge.exe software. Here we will open it from the downloaded folder and will be creating the file of 2 MB size. This file will be using to store the data from the SD card module into.txt form.
- With WinImage in place, you can recreate the disk image on the hard drive or other media, view its content, extract image-based files, add new files and directories, change the format, and defragment the image.
- Follow the Images below to find our how to create the image file.
- Keep the FAT 12/16 and other parameter unaffecting except size of image.
- For this example we will be using 2 MB image file and hence the same is created shown below in the images. We given the name to our image file “Venuuu.IMG”
- Note : Don’t change any extention (*.IMG) while saving files.
- Step 1 : Open the "winimage.exe" file



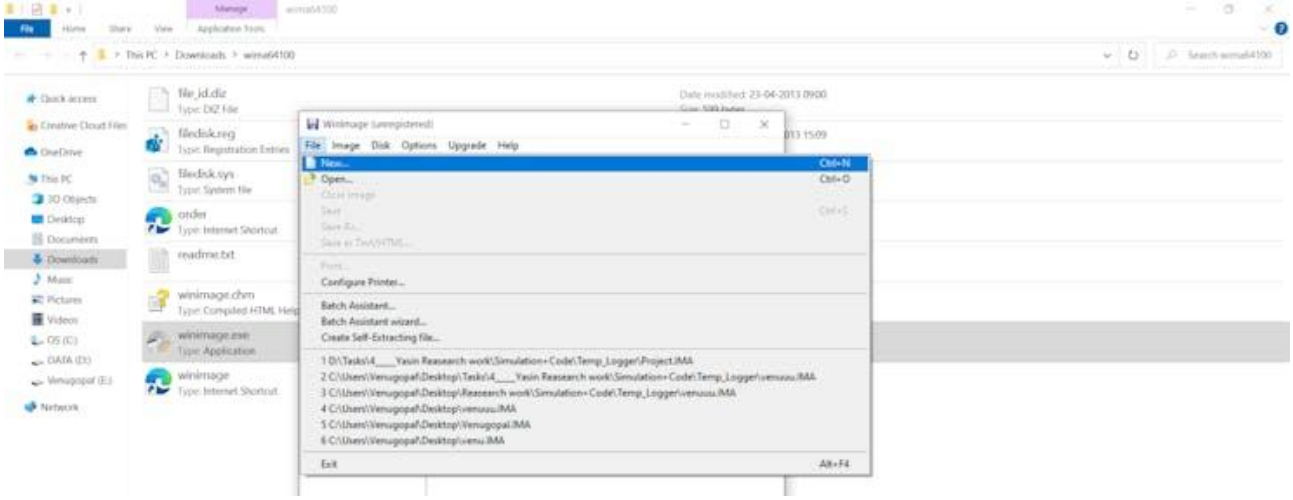
- Step 2 : Click on "Ok " and Proceed



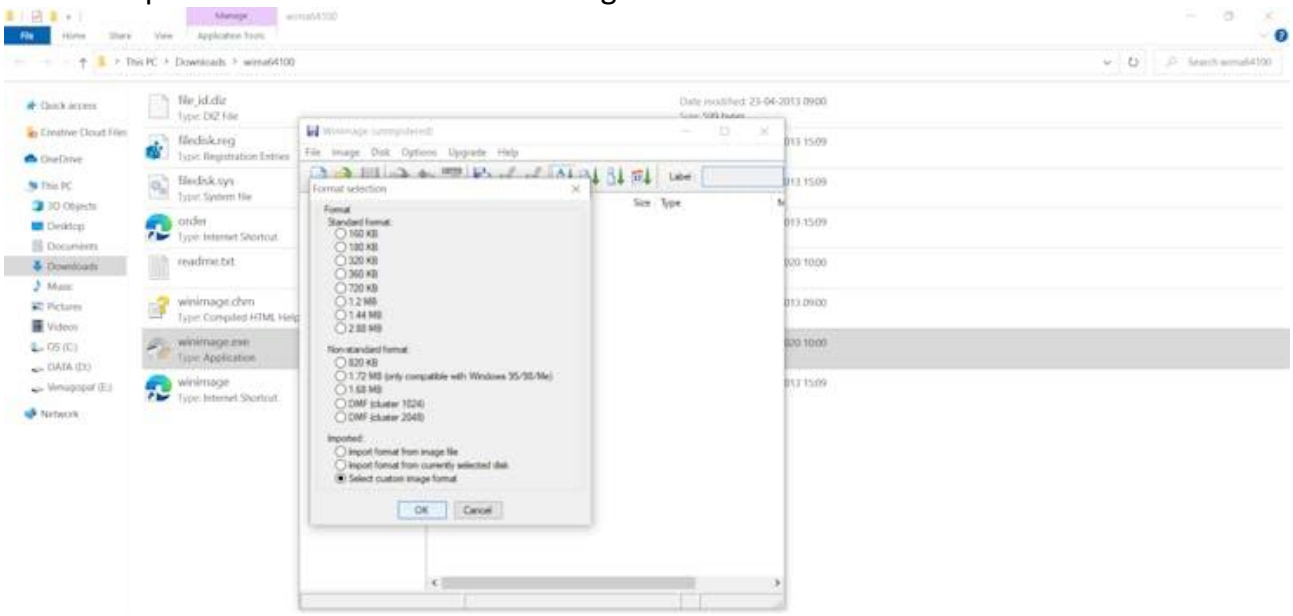
- Step 3 : Go to "File" in the left-hand top corner



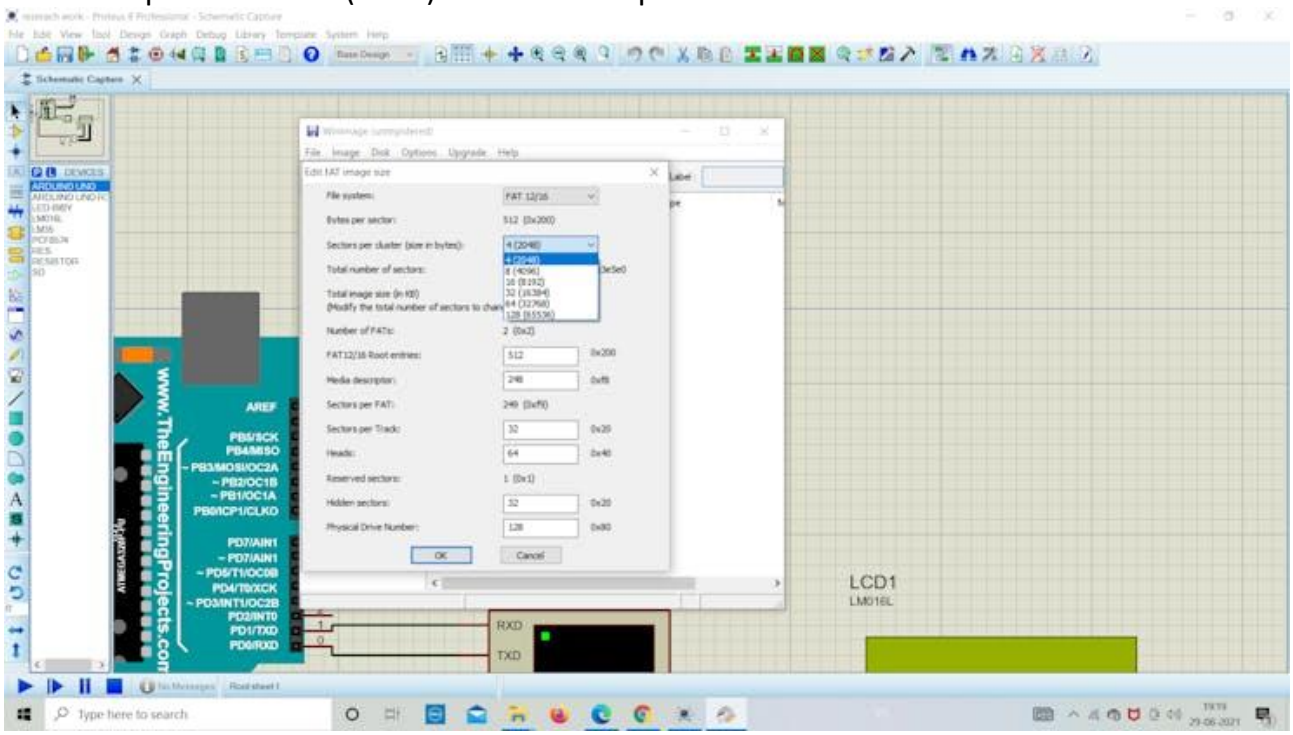
- Step 4 : Click on "New"



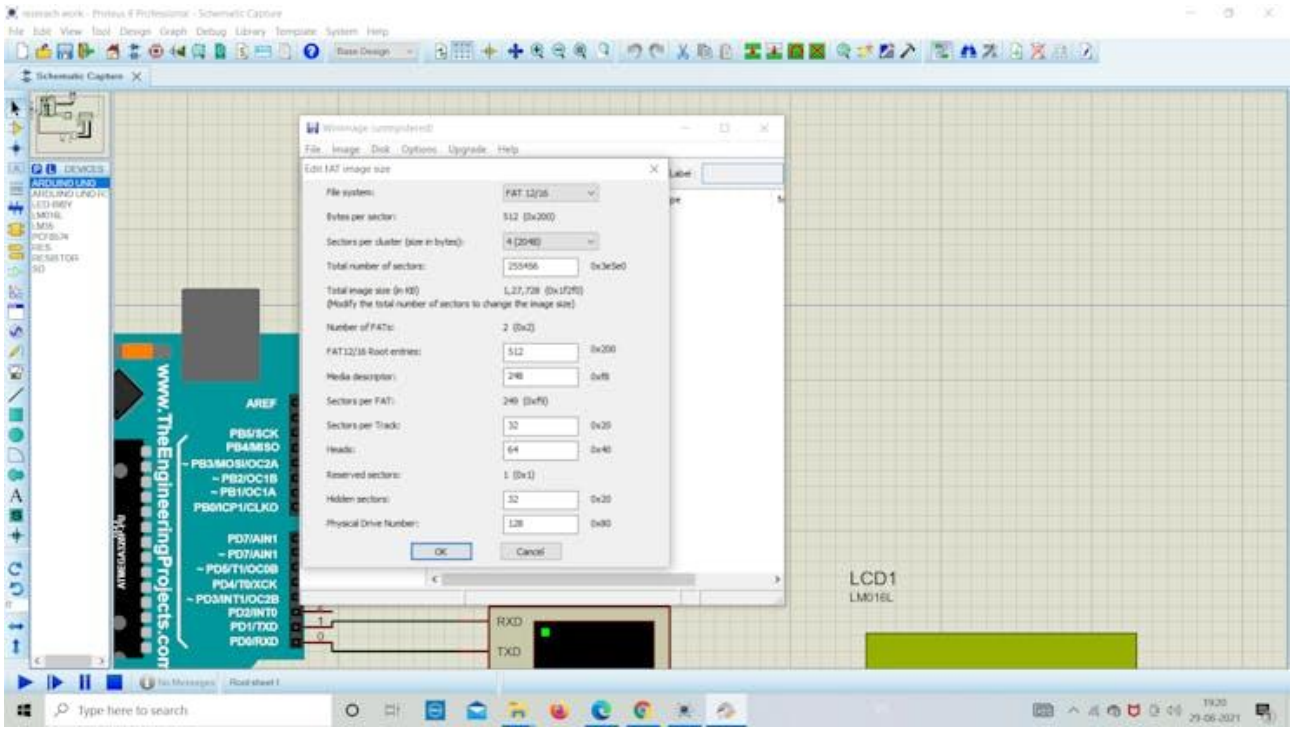
- Step 5 : Click on "Select custom image format"



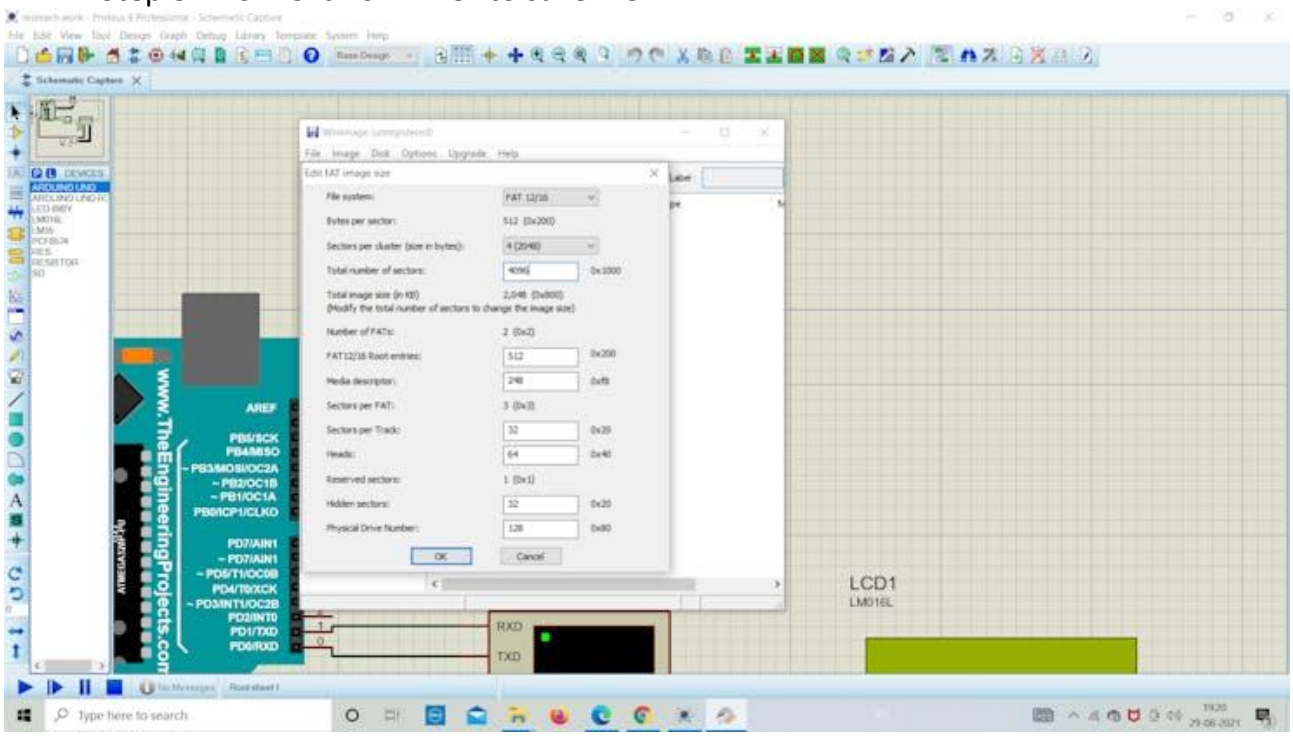
- Step 6 : Select "4 (2048)" from the dropdown list



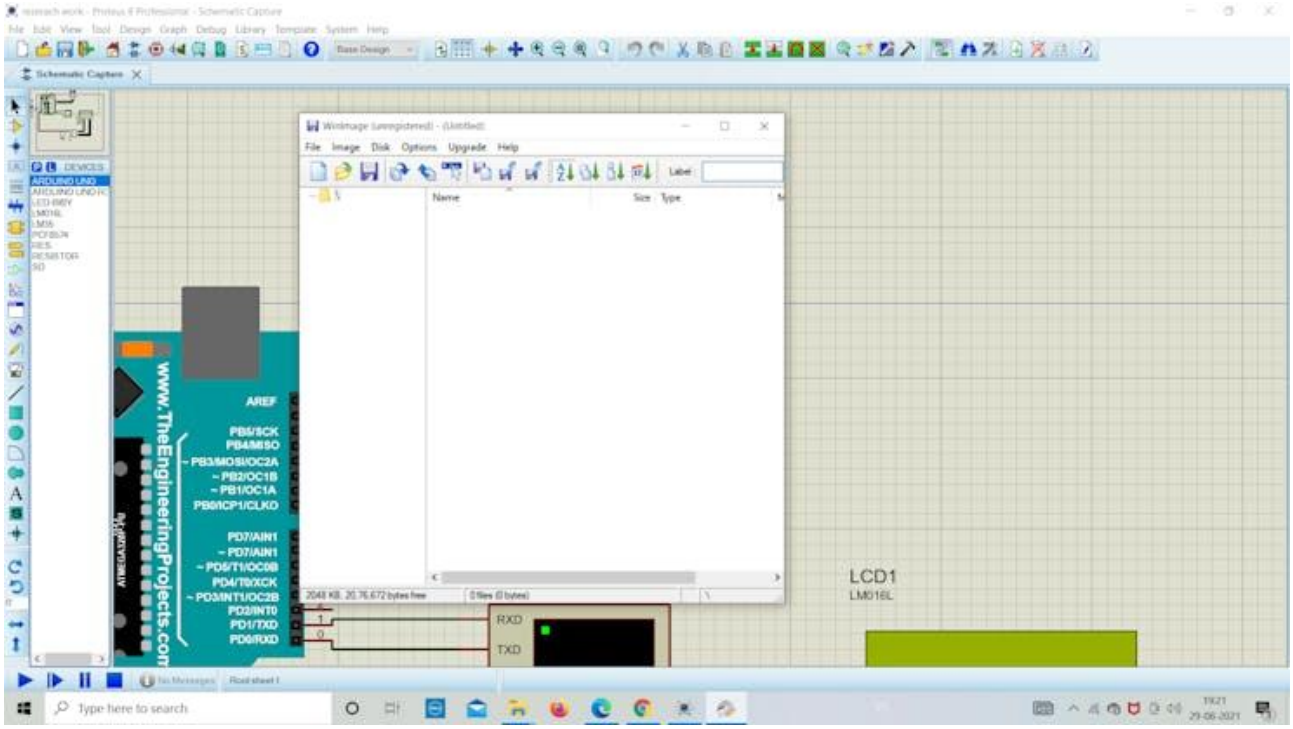
- Step 7 : Keep everything as it is and click on "Ok"

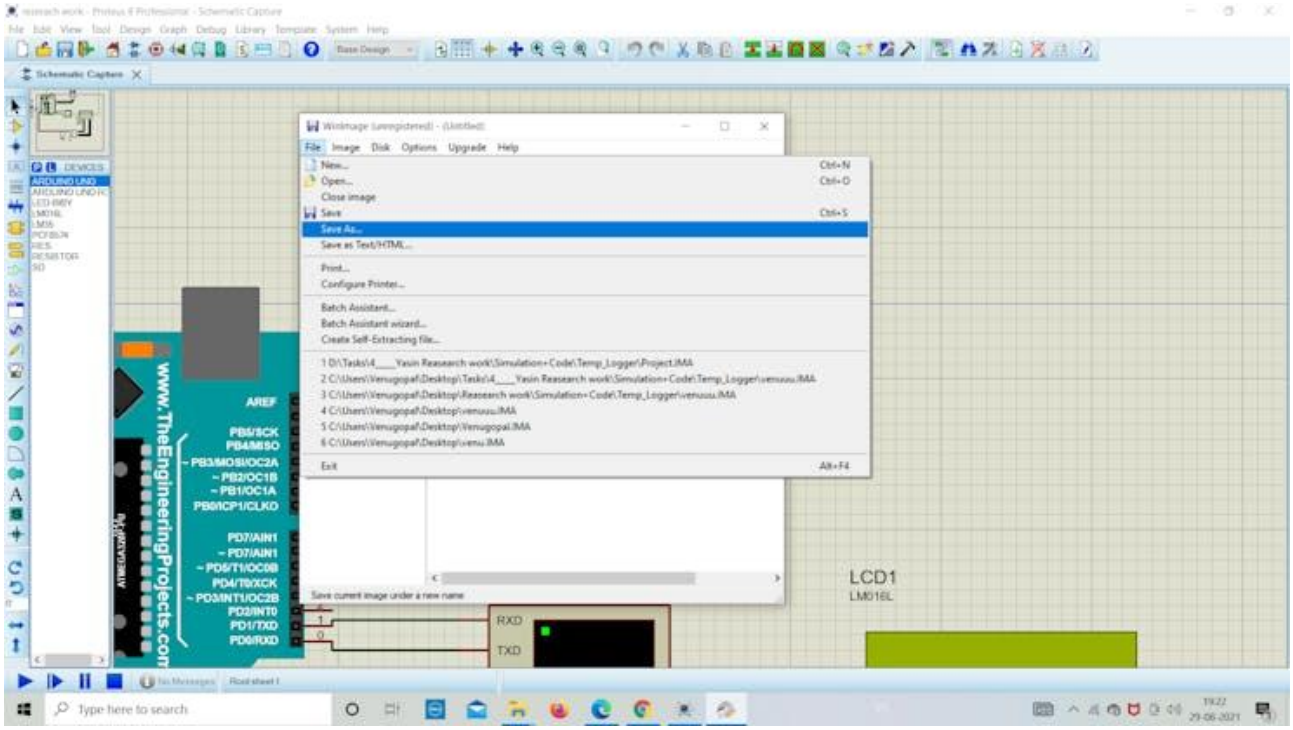


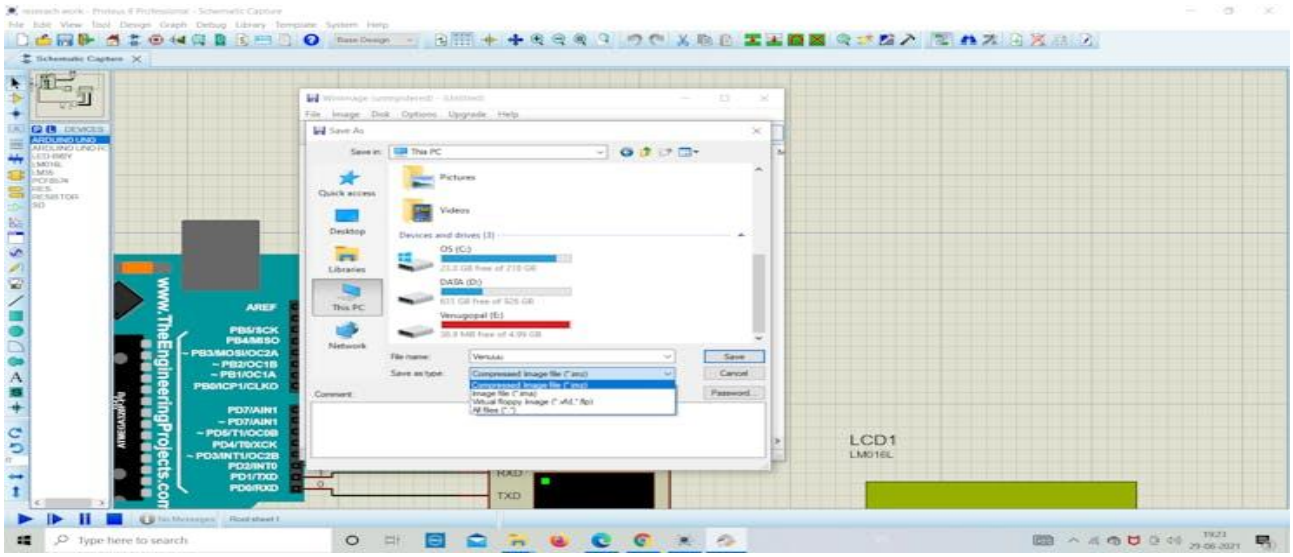
- Step 8 : Now Click on "File" to save file



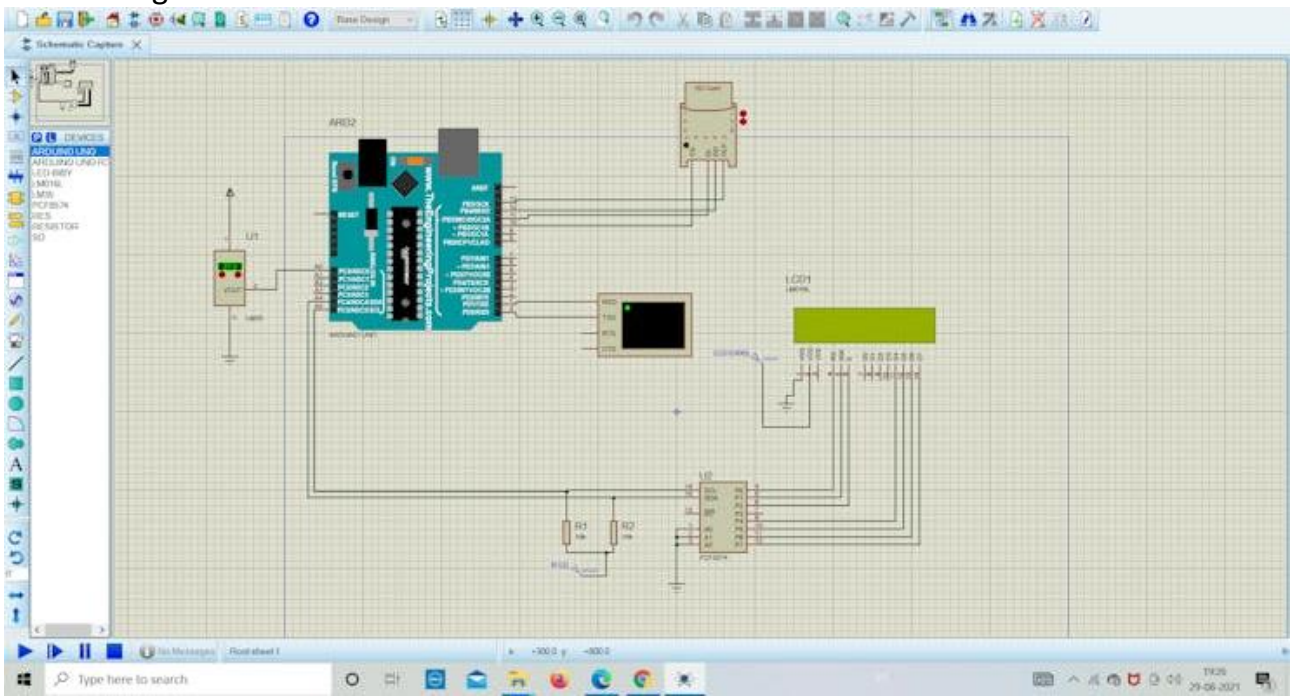
- Step 9 : Now "Save As" the file to the destination folder

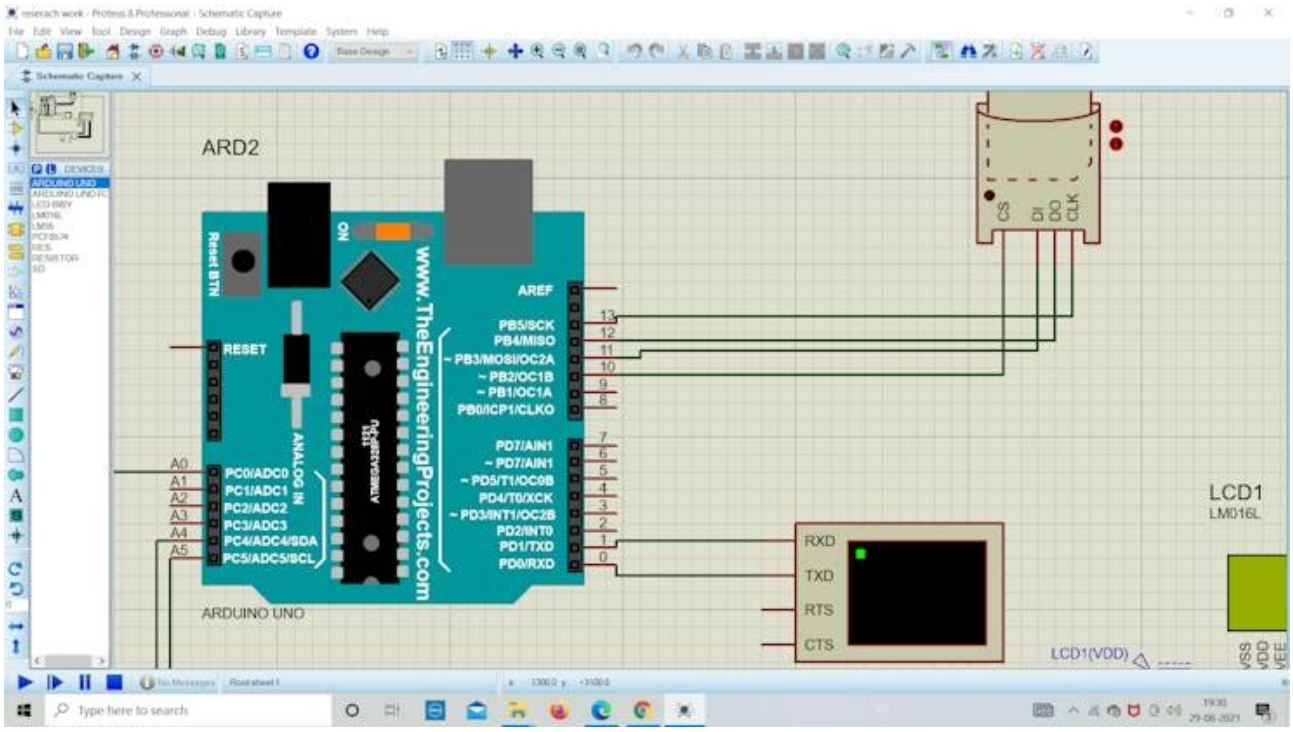

Winimage (unregistered) - (Untitled)
File Image Disk Options Upgrade Help
Name Size Type
www.TheEngineeringProjects.com
AREF
PB5/SCK
PB4/MISO
~ PB3/MOSI/OC2A
~ PB2/OC1B
~ PB1/OC1A
PB0/ICP1/CLKO
PD7/AIN1
~ PD7/AIN1
~ PD5/T1/OC0B
PD4/T0/XCK
~ PD3/INT1/OC2B
PD2/INT0
PD1/TXD
PD0/RXD
RXD
TXD
LCD1
LM016L


Winimage (unregistered) - (Untitled)
File Image Disk Options Upgrade Help
New... Ctrl+N
Open... Ctrl+O
Close image
Save Ctrl+S
Save As...
Save as Text/HTML...
Print...
Configure Printer...
Batch Assistant...
Batch Assistant wizard...
Create Self-Extracting file...
Exit Alt+F4
Save current image under a new name
www.TheEngineeringProjects.com
LCD1
LM016L

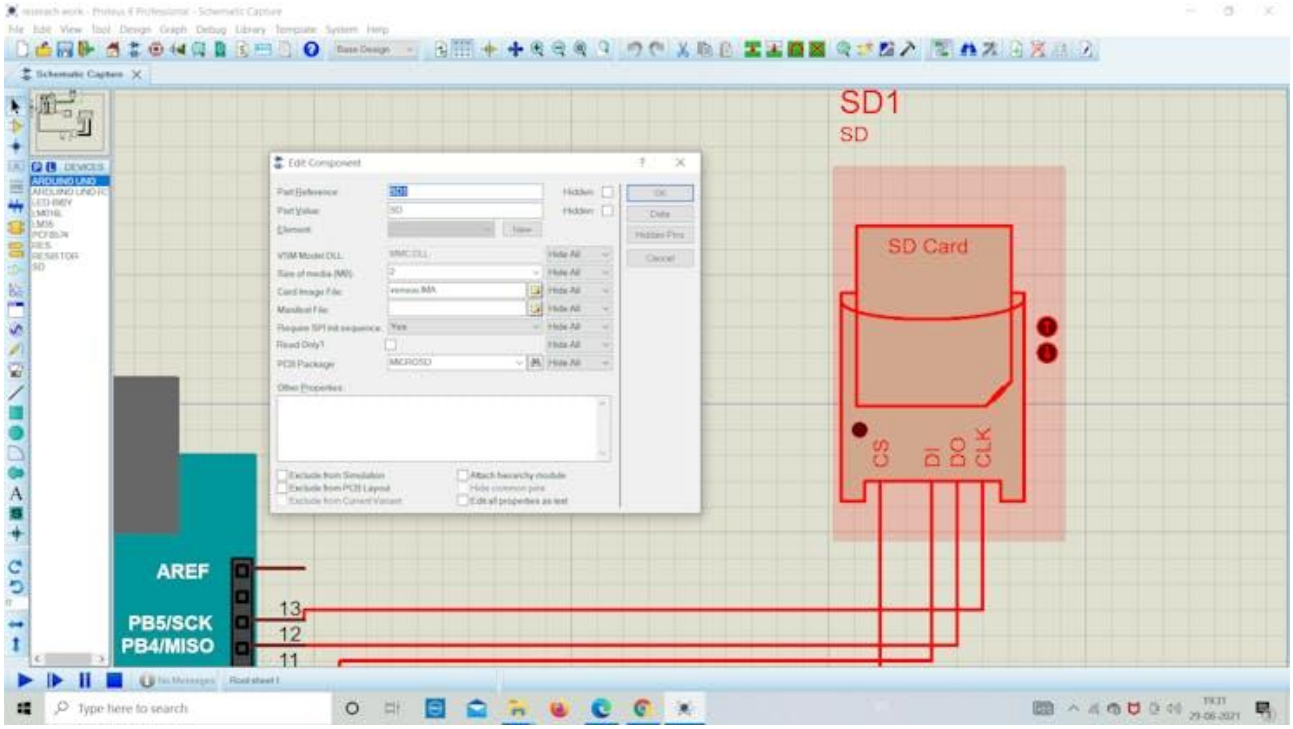


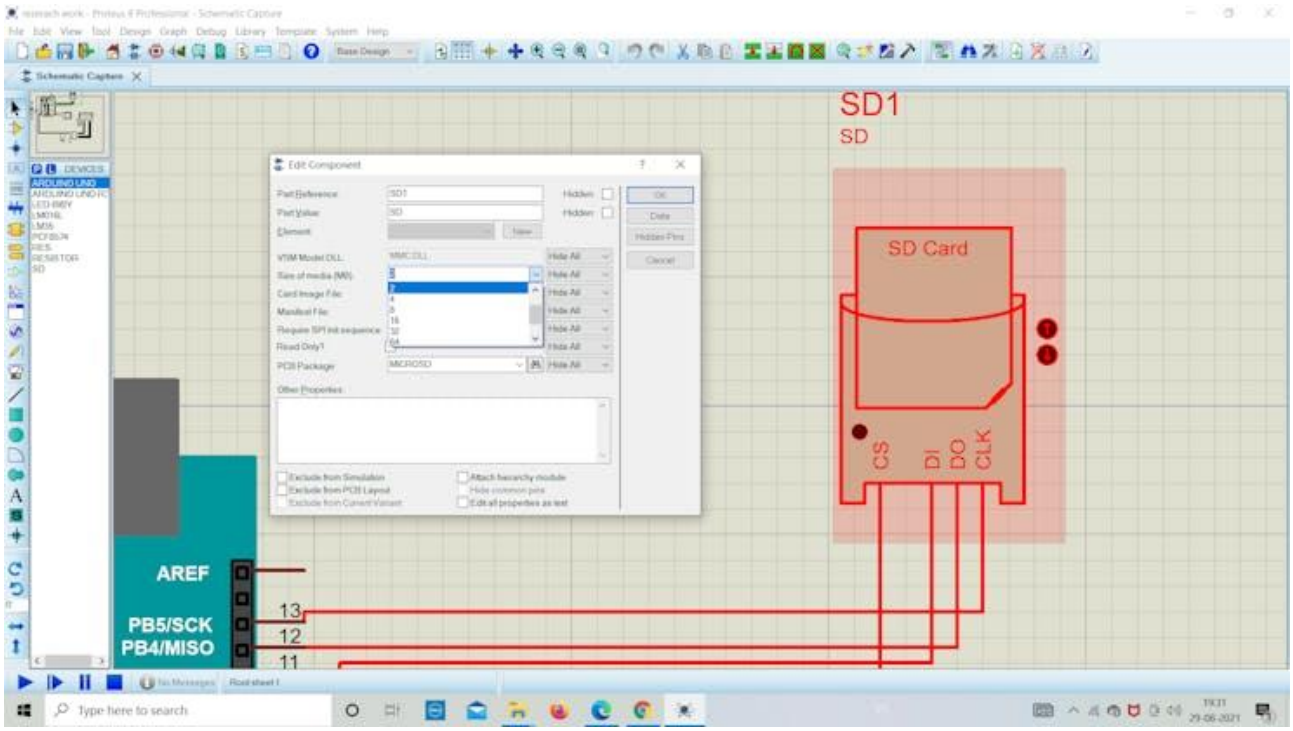
- 1. Simulation & Coding
- After successfully setting up all prerequisites required for simulation, now we will be diving into the actual part of our simulating the SD card module in proteus. Now open the new file in proteus and drag & drop the required components from the list in software (for reference refer image below)as we mentioned names of all components above.
- Here we are using the I2C (PCF8574) for connecting the LCD to Arduino which allows usage of less number of pins on board.
- In the next part assign the file path of image file in SD card module as shown in the image below and select the file size to be 2 MB.

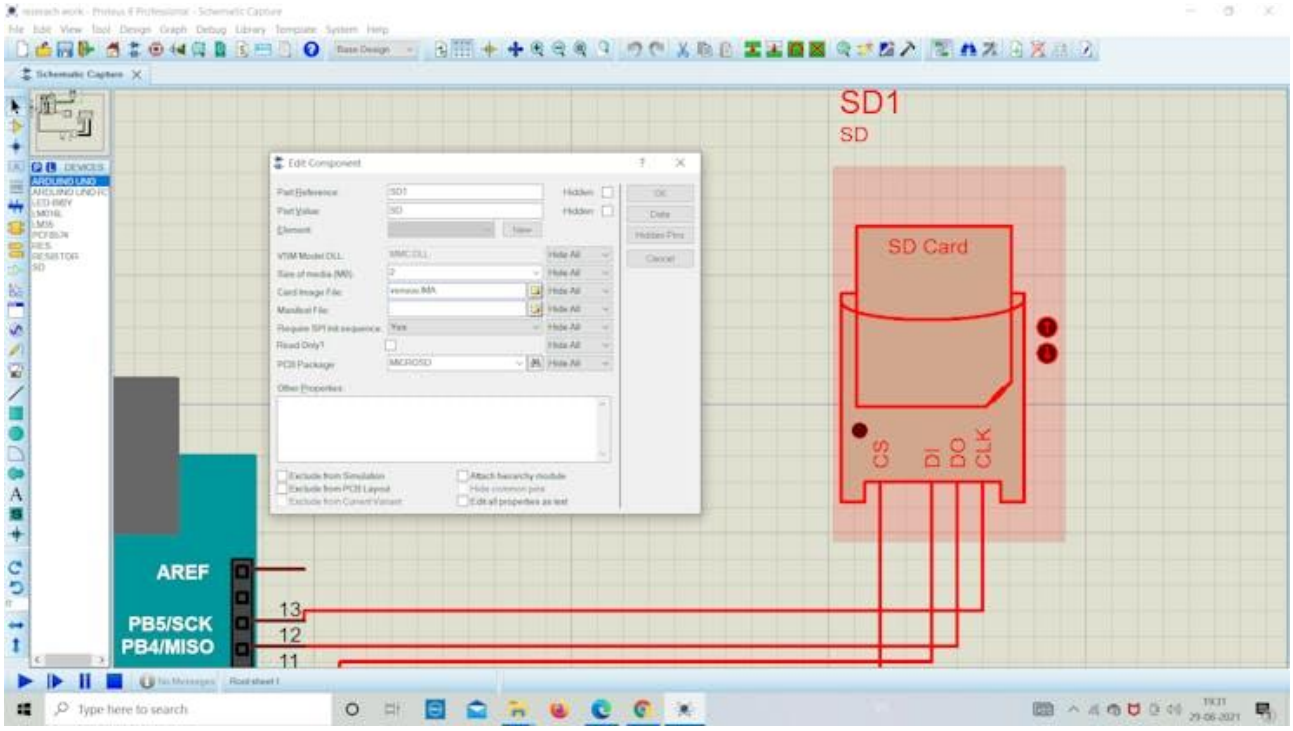




- At this stage, we will set the card image file path in the SD card module in the proteus simulation (same is represented using the below images).
- **Note : Please Select the Size of media in (MB) equal to the image file you created. you can create any size of image file, as per your requirement.**







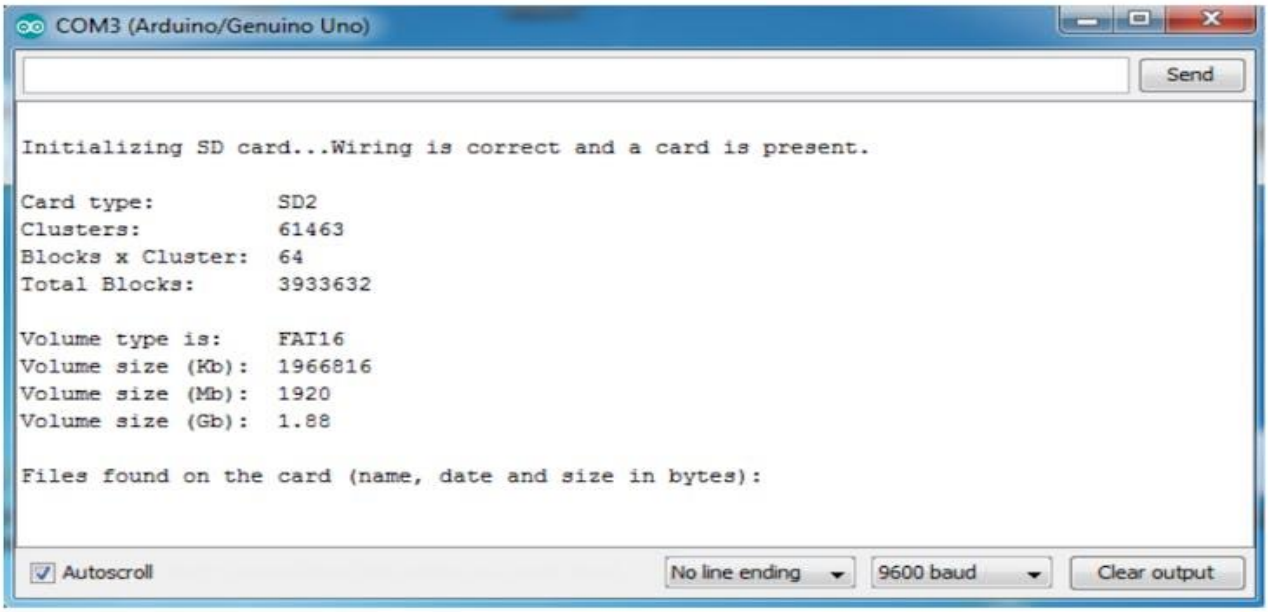
- 2. Checking the Status of SD card
- Now we add the below sketch in the Arduino to know the status of the SD card module.
- The below sketch is used to know that, whether our SD card is correctly connected to Arduino or not. If not find out the reason and debug it.
- Upload the below sketch as a primary step for successfully running the simulation. If everything goes well likewise shown in the below images, then you are on the right track to the end results.

```
// Interfacing Arduino with SD card example (get SD card info)
// include the SD library:
#include <SPI.h>
#include <SD.h>
// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;
void setup() {
// Open serial communications and wait for port to open:
Serial.begin(9600);
while (!Serial) {
; // wait for serial port to connect. Needed for native USB
port only
}
Serial.print("\nInitializing SD card...");
// we'll use the initialization code from the utility
libraries
// since we're just testing if the card is working!
if (!card.init()) {
Serial.println("initialization failed. Things to check:");
Serial.println("* is a card inserted?");
Serial.println("* is your wiring correct?");
Serial.println("* did you change the chipSelect pin to match
your shield or module?");
while (1);
} else {
Serial.println("Wiring is correct and a card is present.");
}
// print the type of card
Serial.println();
Serial.print("Card type:         ");
switch (card.type()) {
case SD_CARD_TYPE_SD1:
Serial.println("SD1");
break;
case SD_CARD_TYPE_SD2:
Serial.println("SD2");
break;
case SD_CARD_TYPE_SDHC:
Serial.println("SDHC");
break;
default:
Serial.println("Unknown");
```

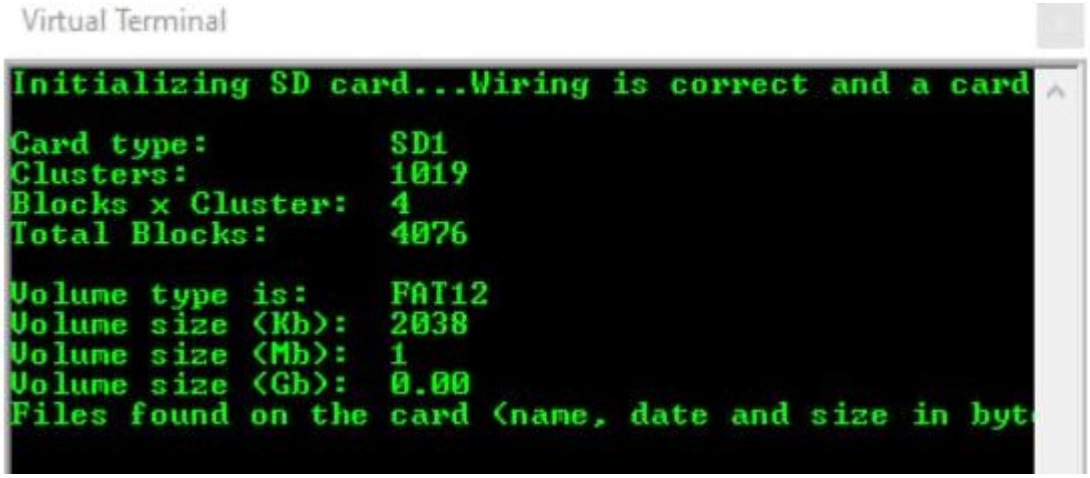
```
}
// Now we will try to open the 'volume'/'partition' - it
should be FAT16 or FAT32
if (!volume.init(card)) {
Serial.println("Could not find FAT16/FAT32 partition.\nMake
sure you've formatted the card");
while (1);
}
Serial.print("Clusters:          ");
Serial.println(volume.clusterCount());
Serial.print("Blocks x Cluster:  ");
Serial.println(volume.blocksPerCluster());
Serial.print("Total Blocks:      ");
Serial.println(volume.blocksPerCluster() *
volume.clusterCount());
Serial.println();
// print the type and size of the first FAT-type volume
uint32_t volumesize;
Serial.print("Volume type is:    FAT");
Serial.println(volume.fatType(), DEC);
volumesize = volume.blocksPerCluster();    // clusters are
collections of blocks
volumesize *= volume.clusterCount();       // we'll have a lot
of clusters
volumesize /= 2;                           // SD card blocks
are always 512 bytes (2 blocks are 1KB)
Serial.print("Volume size (Kb):  ");
Serial.println(volumesize);
Serial.print("Volume size (Mb):  ");
volumesize /= 1024;
Serial.println(volumesize);
Serial.print("Volume size (Gb):  ");
Serial.println((float)volumesize / 1024.0);
Serial.println("\nFiles found on the card (name, date and size
in bytes): ");
root.openRoot(volume);
// list all files in the card with date and size
root.ls(LS_R | LS_DATE | LS_SIZE);
}
void loop(void) {
}
```

- 3. Serial Monitor Output

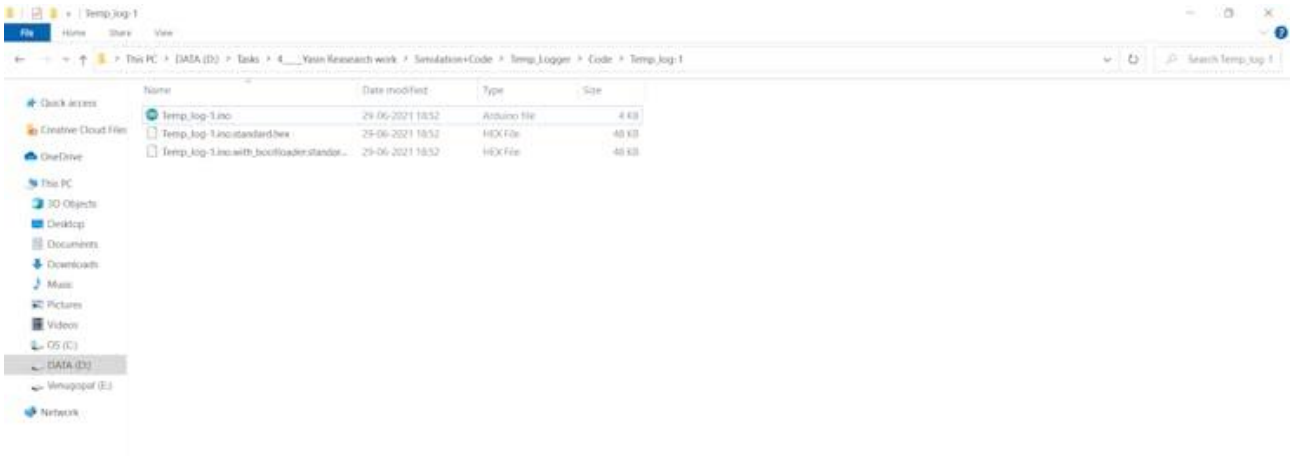
- Once we upload the program in Arduino we will see any of the below status in serial monitor.
- 1. If there is no card or the Arduino can't connect to it, serial monitor will display the message below:



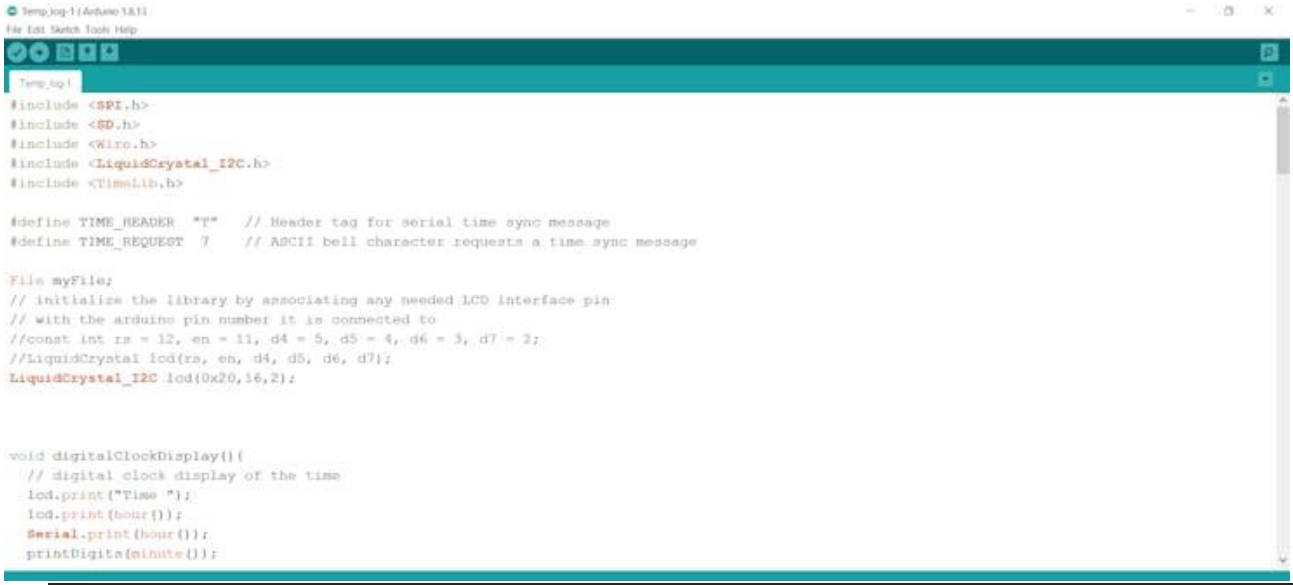
- 2. If we place an image file of 2 MB with FAT16 file system and I got the result shown below:



- 4. Programming, Uploading & Storing
- Now we will generate the code and upload it in the Arduino to get the Temperature values from LM35 sensor. The temperature values with date and time is stored in.txt file using winimage. We are using the LCD here to display the temperature for user without everytime opening the.txt file from image.
- Save the program at desired location and upload. For reference use the code below to know more.
- After uploading run the simulation and wait for some time to store 10-20 readings in SD card.



- Following is the final sketch to be uploaded into the Arduino board. Take the reference of the below code and save it on the destination folder.

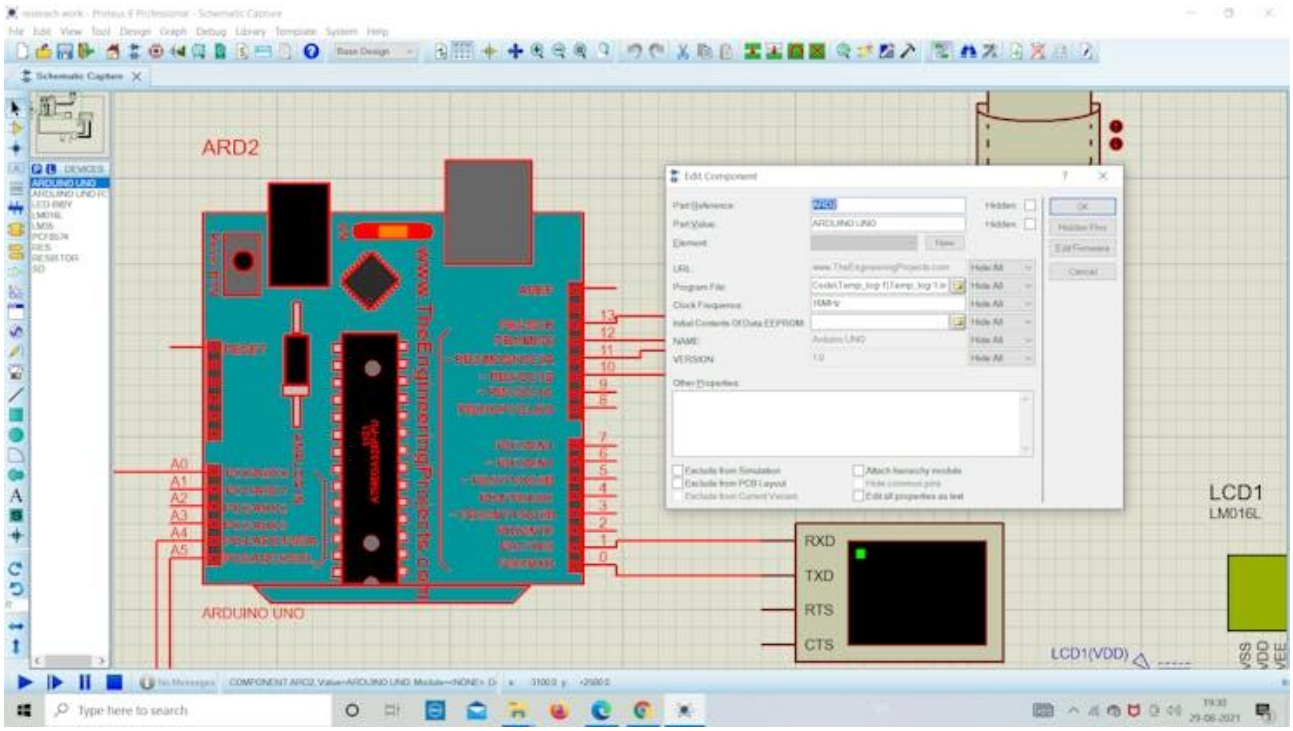


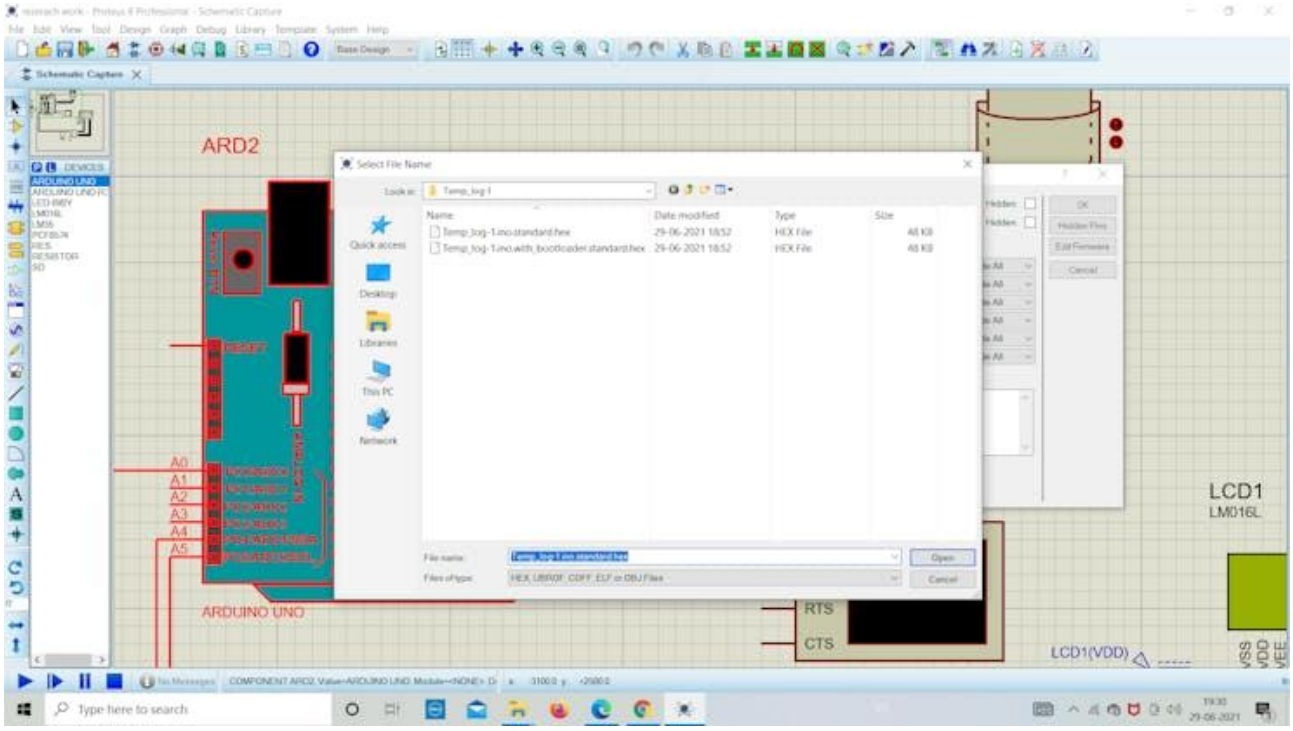
```
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <TimeLib.h>

#define TIME_HEADER  "T"   // Header tag for serial time sync message
#define TIME_REQUEST  7    // ASCII bell character requests a time sync message

File myFile;
// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
//const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
//LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
LiquidCrystal_I2C lcd(0x20,16,2);


void digitalClockDisplay(){
  // digital clock display of the time
  lcd.print("Time ");
  lcd.print(hour());
  Serial.print(hour());
  printDigits(minute());
```

```
lcd.print("Time ");
lcd.print(hour());
Serial.print(hour());
printDigits(minute());
printDigits(second());
lcd.setCursor(0, 1);
lcd.print("Date ");
lcd.print(day());
lcd.print("/");
lcd.print(month());
lcd.print("/");
lcd.print(year());
Serial.print(" ");
Serial.print(day());
Serial.print(" ");
Serial.print(month());
Serial.print(" ");
Serial.print(year());
Serial.println();
}
int printDigits(int digits){
// utility function for digital clock display: prints
preceding colon and leading 0
lcd.print(":");
if(digits < 10)
lcd.print('0');
lcd.print(digits);
return digits;
}
void processSyncMessage() {
unsigned long pctime;
const unsigned long DEFAULT_TIME = 1357041600; // Jan 1 2013
if(Serial.find(TIME_HEADER)) {
pctime = Serial.parseInt();
if( pctime >= DEFAULT_TIME) { // check the integer is a valid
time (greater than Jan 1 2013)
setTime(pctime); // Sync Arduino clock to the time received on
the serial port
}
}
}
time_t requestSync()
{
Serial.write(TIME_REQUEST);
```

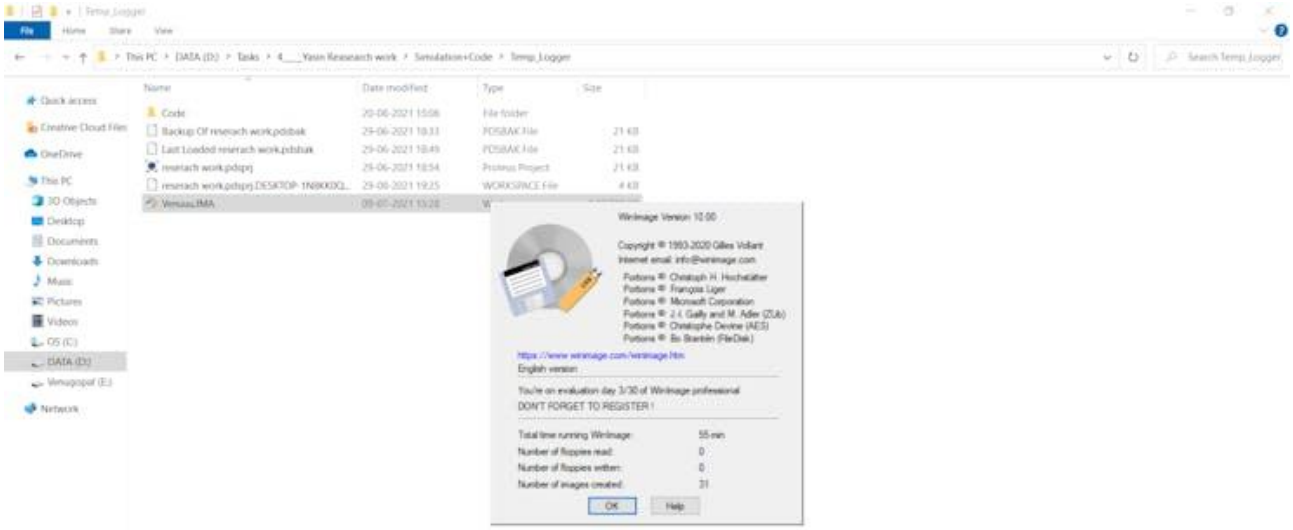
```
return 0; // the time will be sent later in response to serial
mesg
}
void setup() {
lcd.init();
lcd.init();
lcd.backlight();
Serial.begin(9600);
// set up the LCD's number of columns and rows:
if (!SD.begin(10)) {
Serial.println("initialization failed!");
return;
}
else{
Serial.println("initialization Success!");
}
myFile = SD.open("group.txt", FILE_WRITE);
// if the file opened okay, write to it:
if (myFile) {
myFile.print("Date");
myFile.print(",");
myFile.print("Time");
myFile.print(",");
myFile.println("Temperature");
// close the file:
myFile.close();
}
else {
Serial.print("error");
}
while (!Serial) ; // Needed for Leonardo only
pinMode(13, OUTPUT);
setSyncProvider( requestSync); //set function to call when
sync required
Serial.println("Waiting for sync message");
lcd.print("Loading.....");
lcd.clear();
}
void loop() {
if (Serial.available()) {
processSyncMessage();
}
if (timeStatus()!= timeNotSet) {
digitalClockDisplay();
```

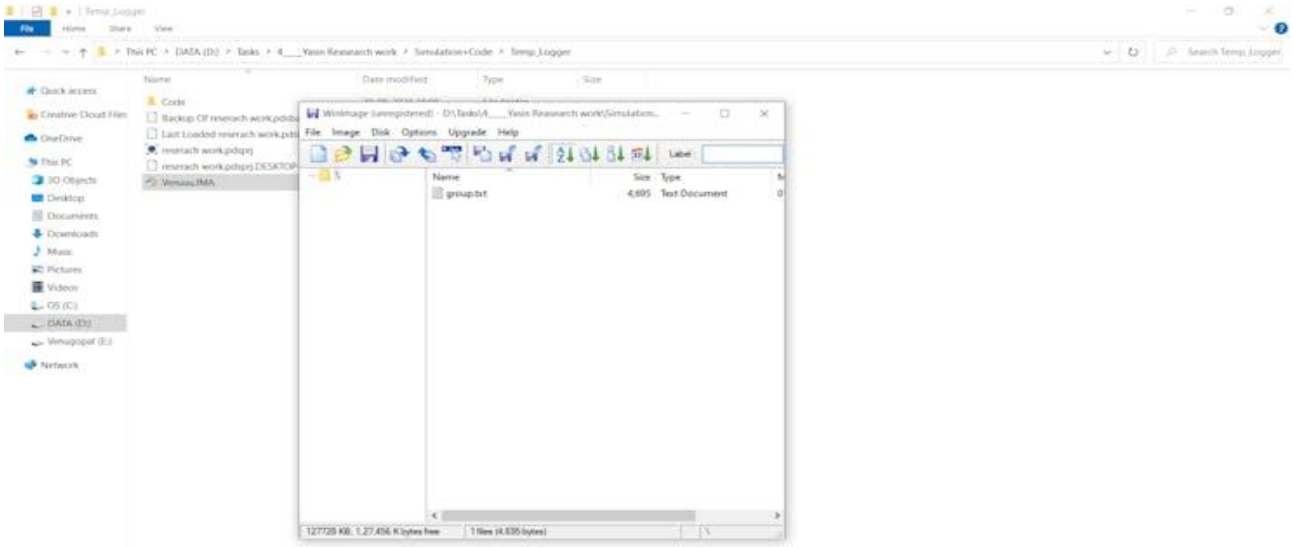
```
}
if (timeStatus() == timeSet) {
digitalWrite(13, HIGH); // LED on if synced
} else {
digitalWrite(13, LOW); // LED off if needs refresh
}
int lm35_pin = A0;
float temperature = analogRead(lm35_pin);
temperature = (temperature*500)/1023;
File data_file = SD.open("group.txt", FILE_WRITE);
if (data_file) {
data_file.print(day());
data_file.print("/");
data_file.print(month());
data_file.print("/");
data_file.print(year());
data_file.print(",");
data_file.print(hour());
data_file.print(":");
// Serial.print(hour());
int p= printDigits(minute());
data_file.print(p);
data_file.print(":");
int m= printDigits(second());
data_file.print(m);
data_file.print(",");
data_file.println(temperature);
data_file.close();
}
else {
Serial.print("error");
}
delay(800);
lcd.setCursor(0, 0);
}
```

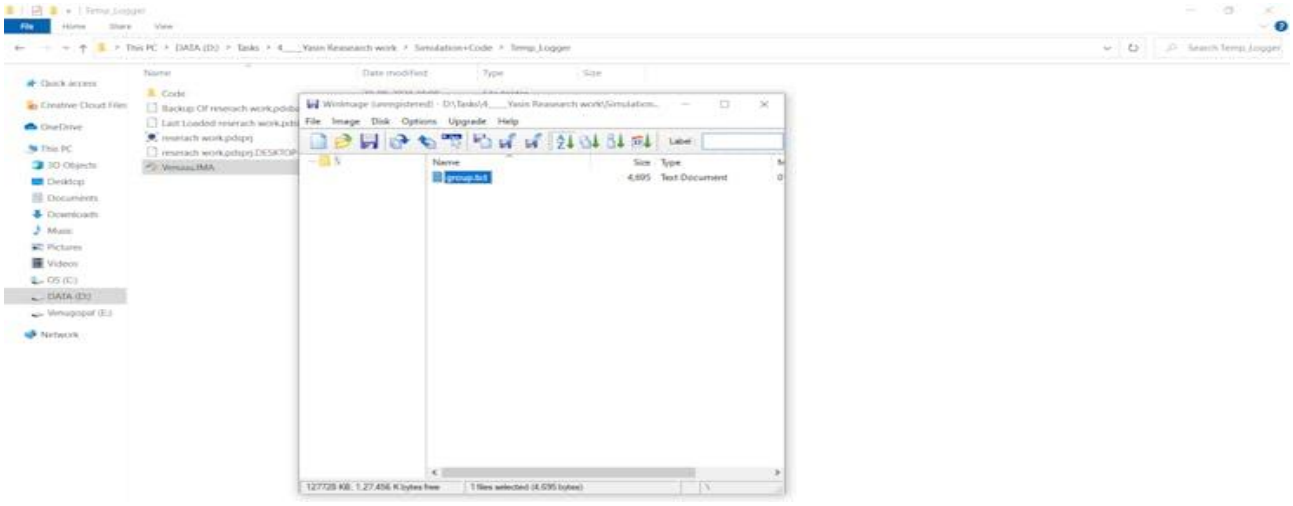


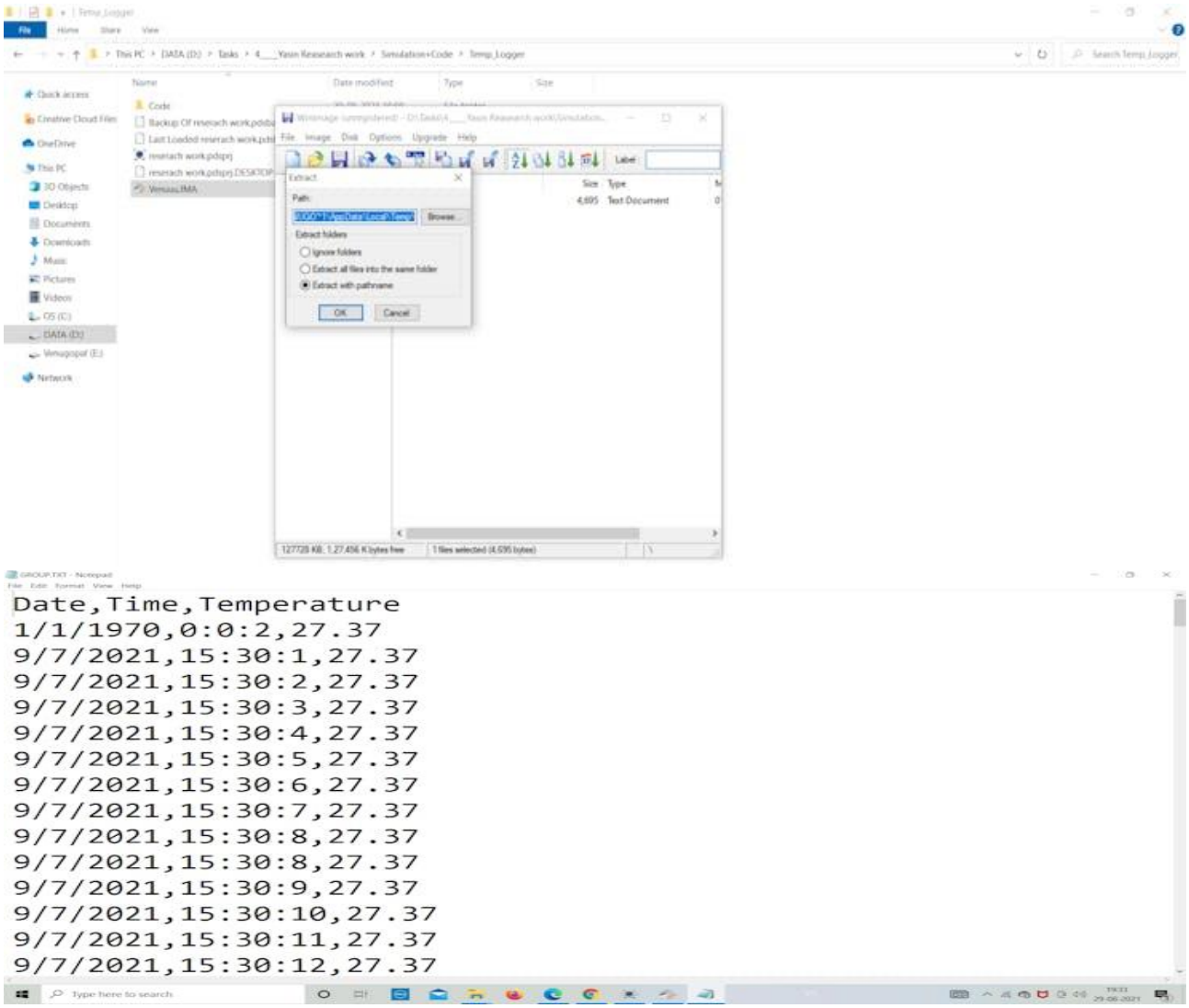


- 5. Exporting SD card Data
- As we did with the simulation part, now let's see whether the data is stored in the image file or not.
- Let's open the "Venuuu.IMG" file to see the content inside the image file.
- As shown in the below images, we can see that.txt file named "group.txt" is created inside the image file. Now we will export the file by double-clicking over it. Then it will automatically open the file and we can see the stored data inside it of temperature.
- Follow the images below to recreate the experiment.

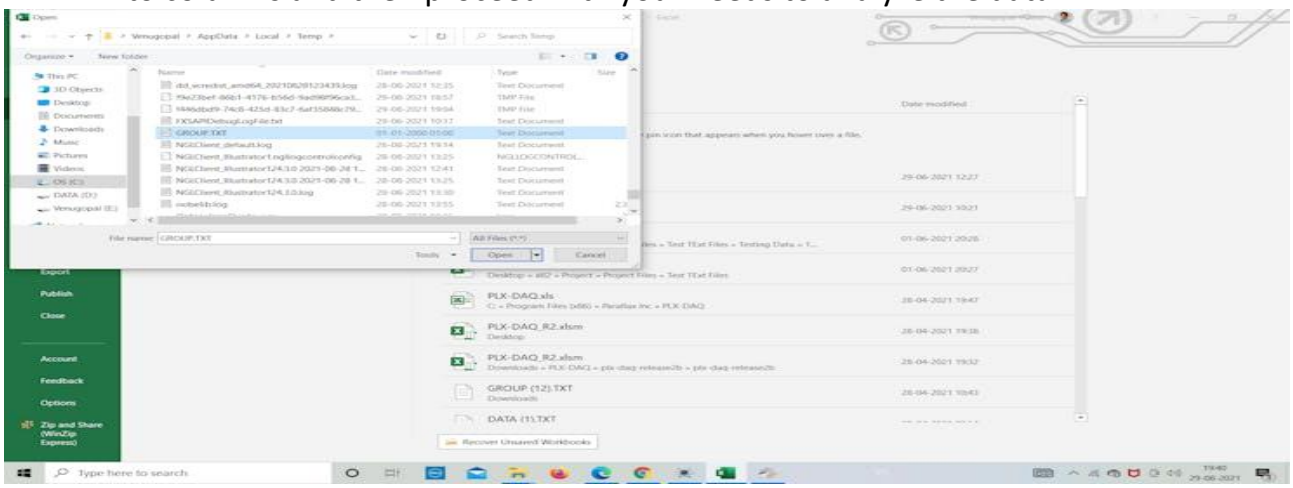


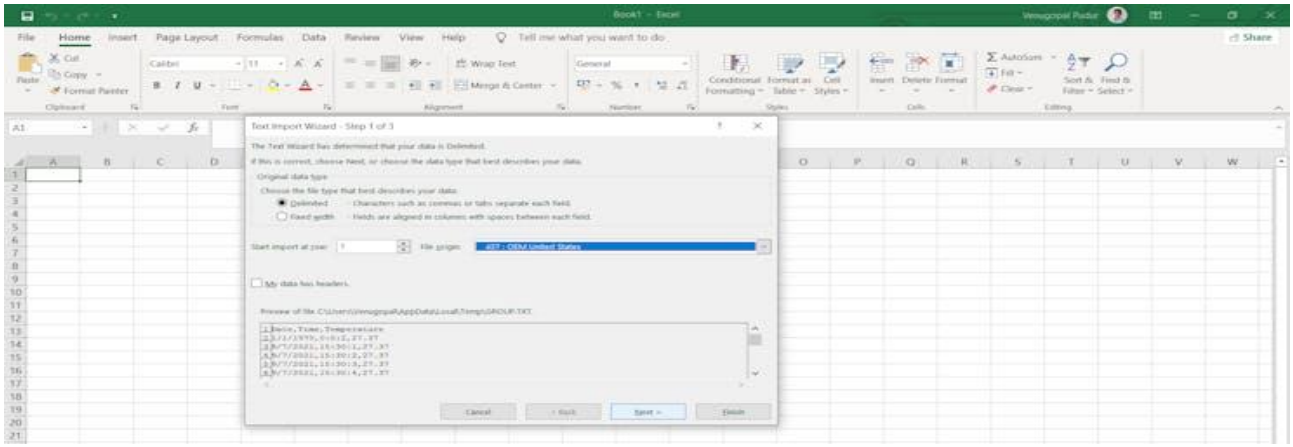


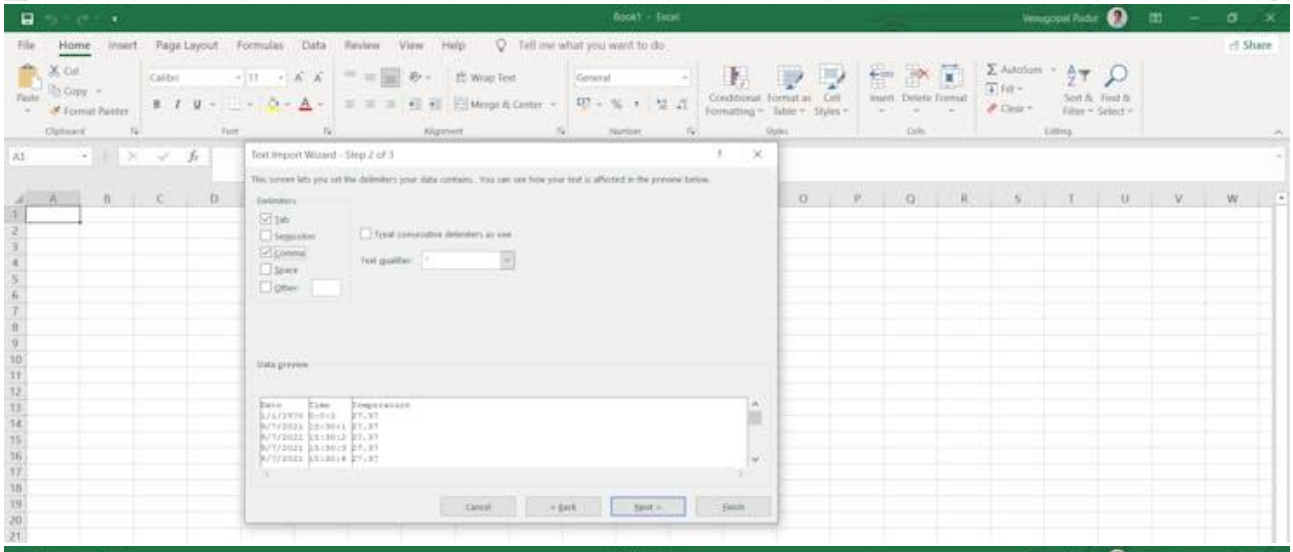


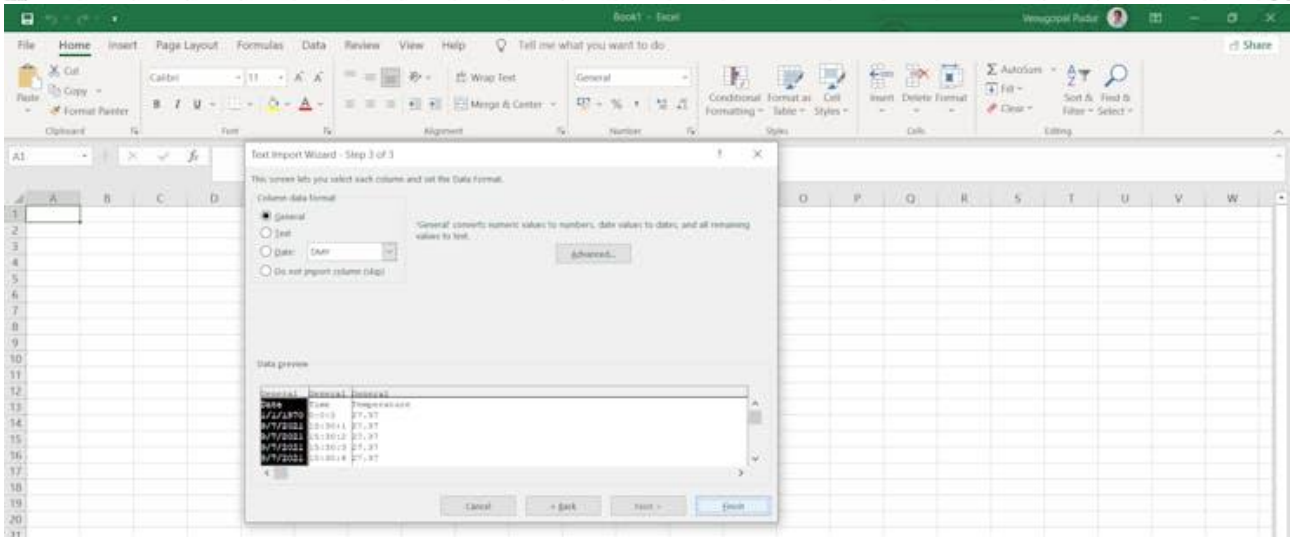


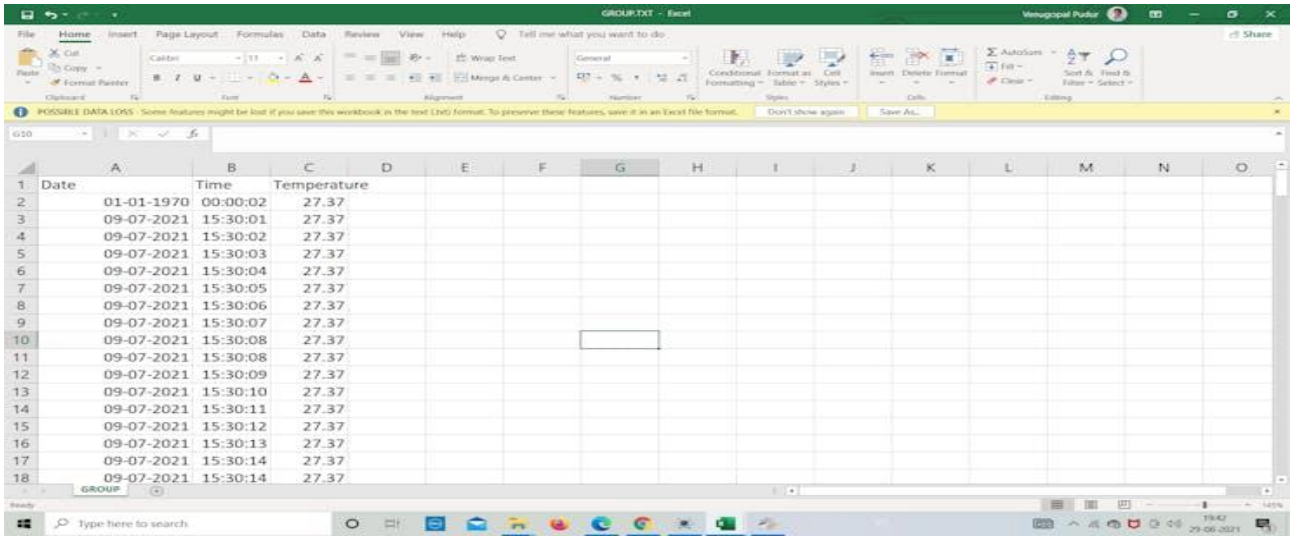
- Now we open the excel to convert the file into excel format for plotting the graphs from data or any other purposes we want.
- Open the.txt file and set the delimiter to separate by acomma for separating data into columns and then proceed with your needs to analyze the data.







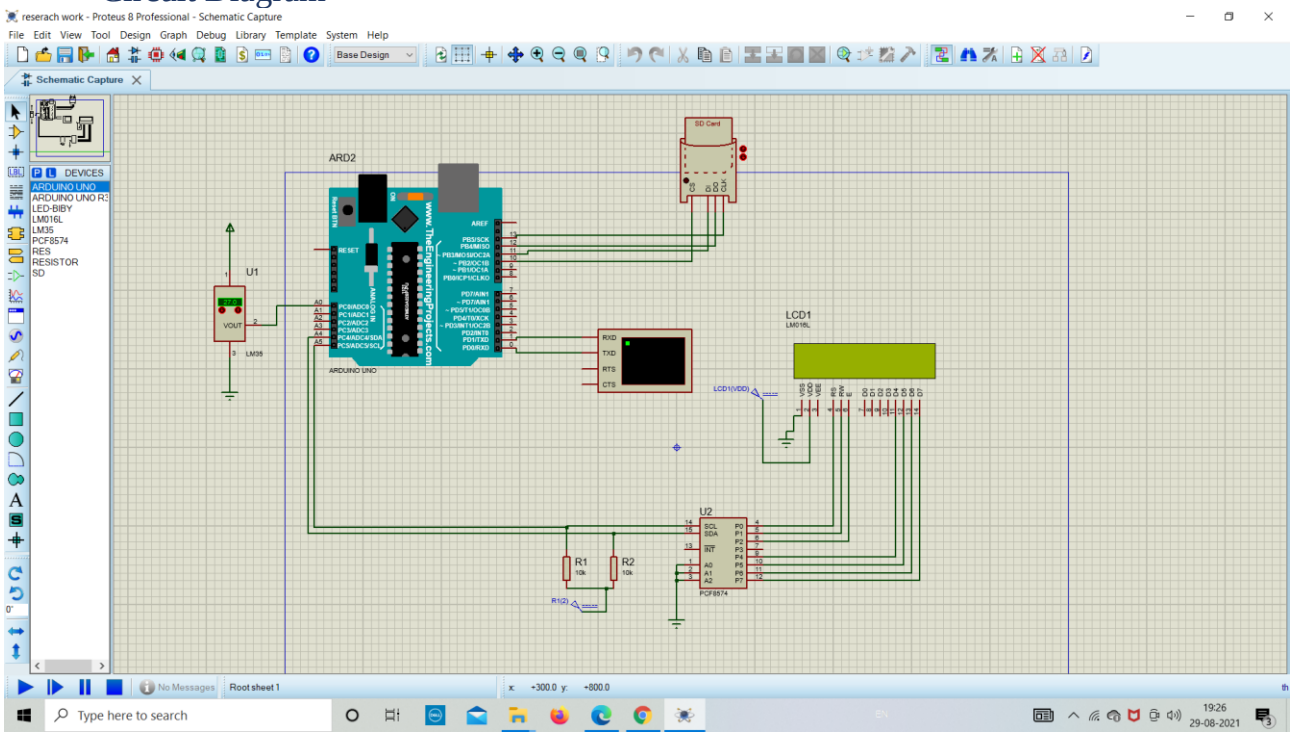




	Date	Time	Temperature
2	01-01-1970	00:00:02	27.37
3	09-07-2021	15:30:01	27.37
4	09-07-2021	15:30:02	27.37
5	09-07-2021	15:30:03	27.37
6	09-07-2021	15:30:04	27.37
7	09-07-2021	15:30:05	27.37
8	09-07-2021	15:30:06	27.37
9	09-07-2021	15:30:07	27.37
10	09-07-2021	15:30:08	27.37
11	09-07-2021	15:30:08	27.37
12	09-07-2021	15:30:09	27.37
13	09-07-2021	15:30:10	27.37
14	09-07-2021	15:30:11	27.37
15	09-07-2021	15:30:12	27.37
16	09-07-2021	15:30:13	27.37
17	09-07-2021	15:30:14	27.37
18	09-07-2021	15:30:14	27.37

- This is all about the simple simulation in Proteus using the SD card module.
- Thank you for reading this article, Have a great day Ahead !

- **Schematics**
- Circuit Diagram



Code

Final Code for Simulation

C/C++

```
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <TimeLib.h>
```

```
#define TIME_HEADER  "T" // Header tag for serial time sync
message
#define TIME_REQUEST  7  // ASCII bell character requests a time
sync message File myFile;
// initialize the library by associating any needed LCD
interface pin
// with the arduino pin number it is connected to
//const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
//LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
LiquidCrystal_I2C lcd(0x20,16,2);
void digitalClockDisplay(){
  // digital clock display of the time
  lcd.print("Time ");
  lcd.print(hour());
  Serial.print(hour());
  printDigits(minute());
  printDigits(second());
  lcd.setCursor(0, 1);
  lcd.print("Date ");
  lcd.print(day());
  lcd.print("/");
  lcd.print(month());
  lcd.print("/");
  lcd.print(year());

  Serial.print(" ");
  Serial.print(day());
  Serial.print(" ");
  Serial.print(month());
  Serial.print(" ");
  Serial.print(year());
  Serial.println();
}

int printDigits(int digits){
  // utility function for digital clock display: prints
preceding colon and leading 0
  lcd.print(":");
  if(digits < 10)
    lcd.print('0');
  lcd.print(digits);
   return digits;
}
void processSyncMessage() {
```

```
  unsigned long pctime;
  const unsigned long DEFAULT_TIME = 1357041600; // Jan 1 2013
  if(Serial.find(TIME_HEADER)) {
     pctime = Serial.parseInt();
     if( pctime >= DEFAULT_TIME) { // check the integer is a
valid time (greater than Jan 1 2013)
       setTime(pctime); // Sync Arduino clock to the time
received on the serial port
     }
  }
}

time_t requestSync()
{
  Serial.write(TIME_REQUEST);
  return 0; // the time will be sent later in response to serial
mesg
}
void setup() {
       lcd.init();
       lcd.init();
       lcd.backlight();
       Serial.begin(9600);
       // set up the LCD's number of columns and rows:
   if (!SD.begin(10)) {
       Serial.println("initialization failed!");
       return;
 }
 else{
  Serial.println("initialization Success!");
  }
  myFile = SD.open("group.txt", FILE_WRITE);
  // if the file opened okay, write to it:
  if (myFile) {
    myFile.print("Time");
    myFile.print(",");
    myFile.println("Temperature");
    // close the file:
    myFile.close();
}
    else {

        Serial.print("error");
    }
```

```
  while (!Serial) ; // Needed for Leonardo only
  pinMode(13, OUTPUT);
  setSyncProvider( requestSync);  //set function to call when
sync required
  Serial.println("Waiting for sync message");
  lcd.print("Loading.....");
}
void loop() {
  if (Serial.available()) {
    processSyncMessage();
  }
  if (timeStatus()!= timeNotSet) {
    digitalClockDisplay();
  }
   int  lm35_pin = A0;
   float temperature = analogRead(lm35_pin);
   temperature = (temperature*500)/1023;
   File data_file = SD.open("group.txt", FILE_WRITE);
   if (data_file) {
        data_file.print(day());
        data_file.print("/");
        data_file.print(month());
        data_file.print("/");
        data_file.print(year());
        data_file.print(",");
        data_file.print(hour());
        data_file.print(":");
        // Serial.print(hour());
        int p= printDigits(minute());
        data_file.print(p);
        data_file.print(":");
        int m= printDigits(second());
        data_file.print(m);
        data_file.print(",");
        data_file.println(temperature);
        data_file.close();
   }
    else {
        Serial.print("error");
    }
  lcd.setCursor(0, 0);
}
```