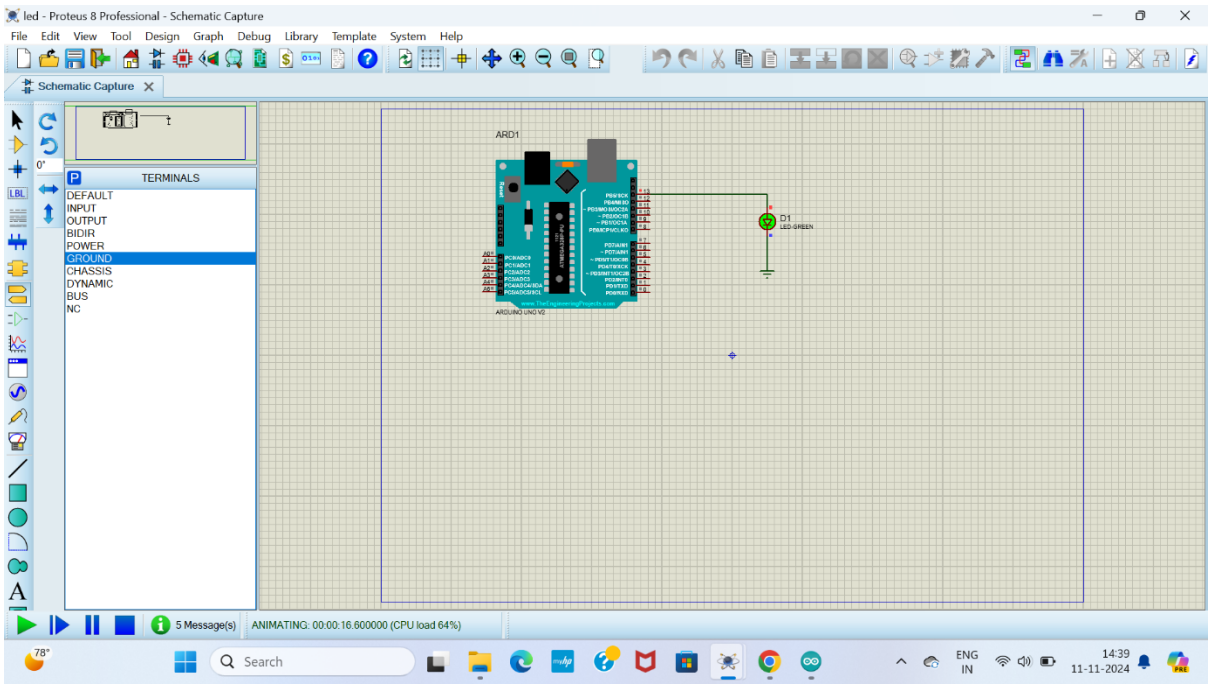
LED Blink

```
void setup()
{
  pinMode(13,OUTPUT);// put your setup code here, to run once:


}

void loop()
{
  digitalWrite(13,HIGH);
  delay(1000);
  digitalWrite(13,LOW);
  delay(1000);
    // put your main code here, to run repeatedly:
}
```

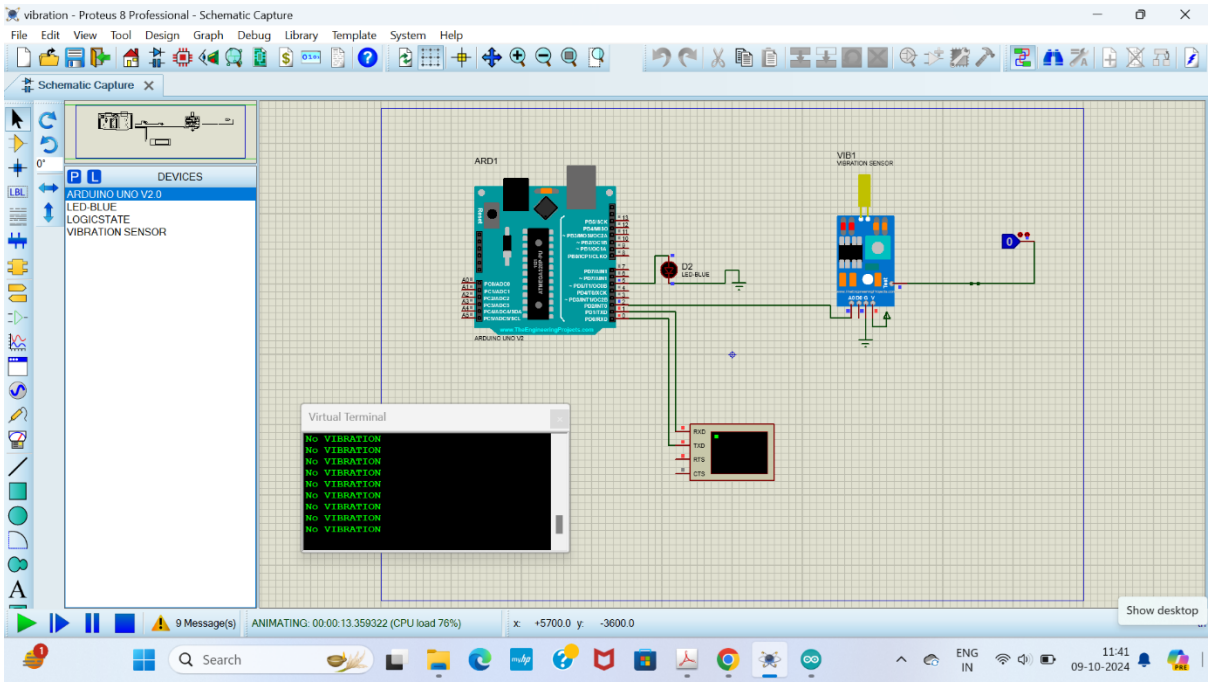


Vibration sensor

```
// include the library code:
int b1 = 2;
int d1 = 5;


int cnt=0,cnt2;
int timer=0;
            // a maximum of eight servo objects can be created
int pos = 0;    // variable to store the servo position
void setup() {
  Serial.begin(9600);      //initialize serial
pinMode(b1, INPUT_PULLUP);
pinMode(d1, OUTPUT);
digitalWrite(d1, HIGH);
digitalWrite(d1,LOW);
  delay(300);              // wait for a second
 cnt=0;
}


void loop() {


  if(digitalRead(b1) == HIGH){
Serial.println("VIBRATION ALERT");


  digitalWrite(d1, HIGH);
  delay(300);              // wait for a second
              // wait for a second



  digitalWrite(d1, LOW);
  delay(300);
```

```
  digitalWrite(d1, HIGH);
  delay(300);              // wait for a second
digitalWrite(d1, LOW);
 delay(300);              // wait for a second

digitalWrite(d1, HIGH);
  delay(300);              // wait for a second
  digitalWrite(d1, LOW);
  delay(300);              // wait for a second

  }
}
```

Vibration sensor

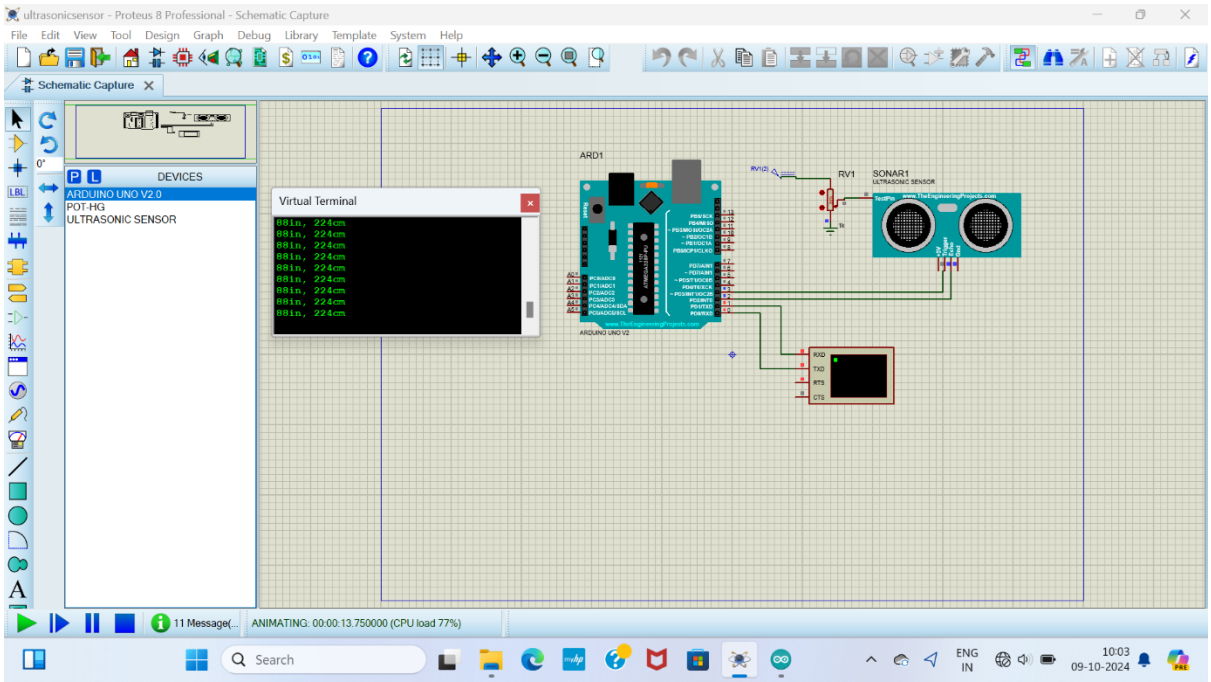


Ultrasonic Sensor

```
const int echoPin = 2; // Echo Pin of Ultrasonic Sensor
const int pingPin = 3; // Trigger Pin of Ultrasonic Sensor

void setup()
{
  Serial.begin(9600); // Starting Serial Communication
  pinMode(pingPin, OUTPUT); // initialising pin 3 as output
  pinMode(echoPin, INPUT); // initialising pin 2 as input
}
void loop()
{
  long duration, inches, cm;

  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);

  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);

  digitalWrite(pingPin, LOW);

  duration = pulseIn(echoPin, HIGH); // using pulsin function to determine total time
  inches = microsecondsToInches(duration); // calling method
  cm = microsecondsToCentimeters(duration); // calling method

  Serial.print(inches);
```

```
 Serial.print("in, ");
 Serial.print(cm);
 Serial.print("cm");
 Serial.println();

 delay(100);
}

long microsecondsToInches(long microseconds) // method to covert microsec to inches
{
 return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds) // method to covert microsec to centimeters
{
  return microseconds / 29 / 2;
}
```

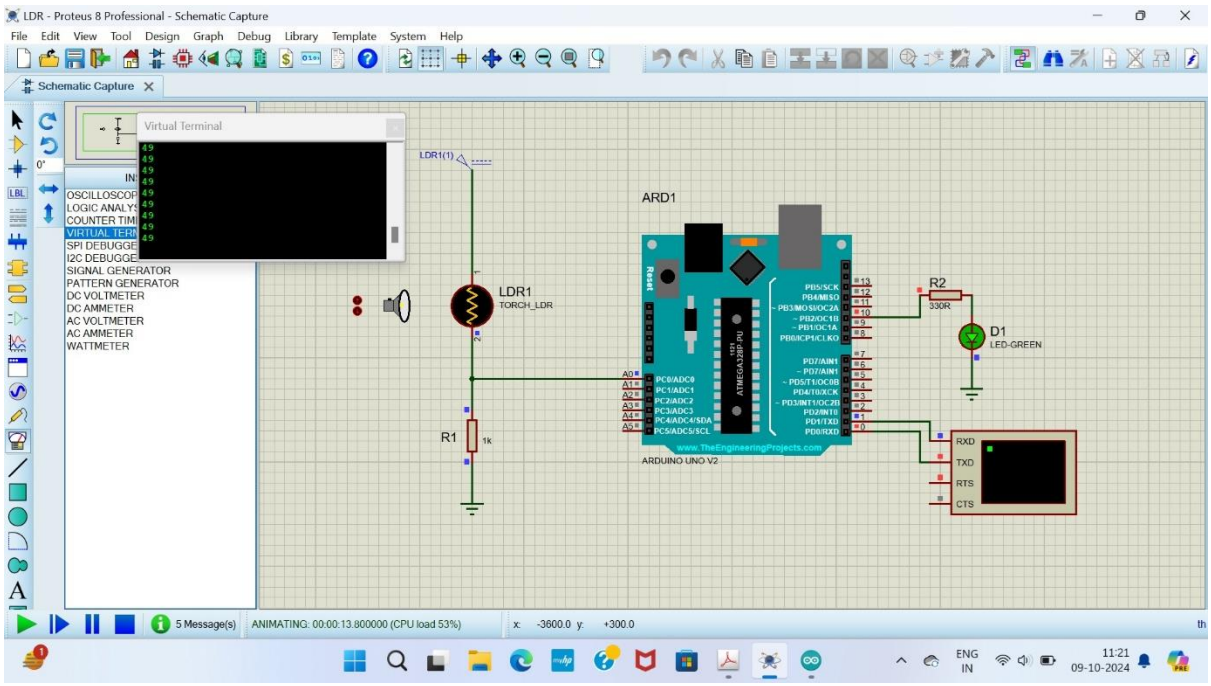
Ultrasonic Sensor



LDR

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  pinMode(10,OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  if(sensorValue < 50)
  {
    digitalWrite(10,HIGH);

    }
    else
    {
      digitalWrite(10,LOW);
      }
  delay(1);        // delay in between reads for stability
}
```

LDR


LDR - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Virtual Terminal
LDR1
TORCH_LDR
R1
1k
ARD1
ARDUINO UNO V2
R2
330R
D1
LED-GREEN
RXD
TXD
RTS
CTS
5 Message(s)
ANIMATING: 00:00:13.800000 (CPU load 53%)
ENG
IN
11:21
09-10-2024

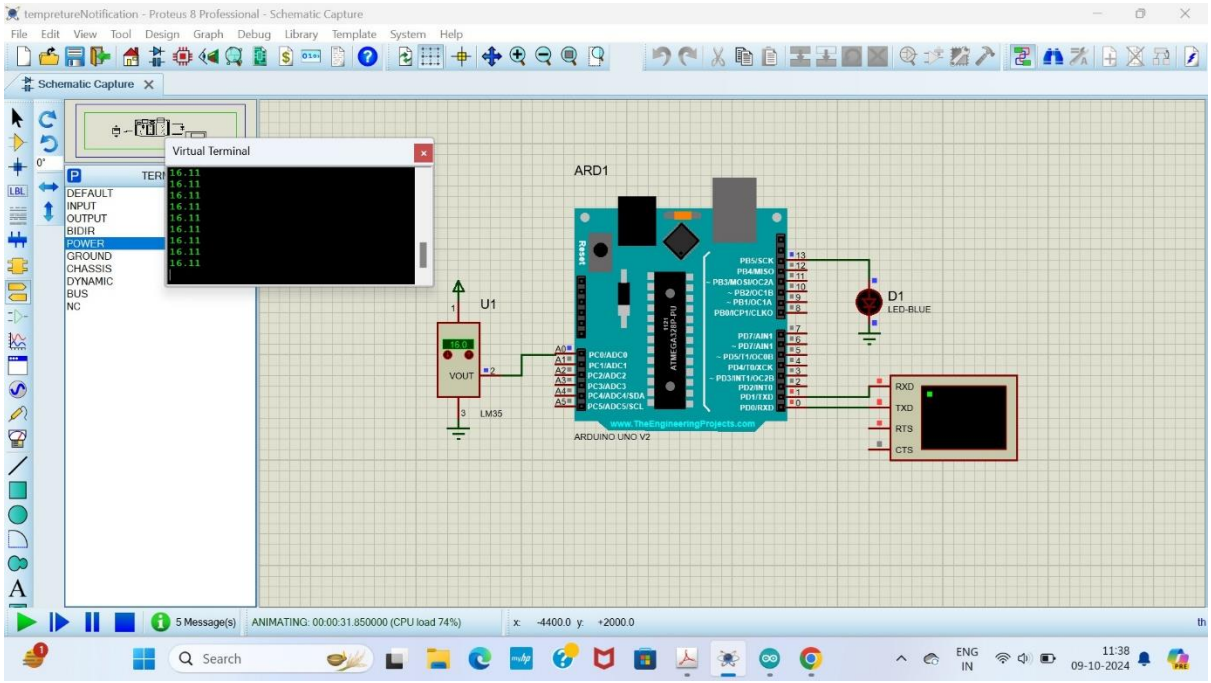
Temperature Notification

```
float temp;

void setup() {

pinMode (13, OUTPUT);

Serial.begin (9600);

}

void loop() {

temp= analogRead (A0);

temp= (temp*500)/1024;

Serial.println (temp);

if (temp>30)

digitalWrite (13, HIGH);

else

digitalWrite (13, LOW);
```

delay (1000);

}

Temperature Notification



program to sense the available networks using Arduino

```
#include <SPI.h>
#include <WiFiNINA.h>

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    // don't continue
    while (true);
  }

  String fv = WiFi.firmwareVersion();
  if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
    Serial.println("Please upgrade the firmware");
  }

  // print your MAC address:
  byte mac[6];
  WiFi.macAddress(mac);
  Serial.print("MAC: ");
  printMacAddress(mac);
}
```

```
void loop() {
  // scan for existing networks:
  Serial.println("Scanning available networks...");
  listNetworks();
  delay(10000);
}

void listNetworks() {
  // scan for nearby networks:
  Serial.println("** Scan Networks **");
  int numSsid = WiFi.scanNetworks();
  if (numSsid == -1) {
    Serial.println("Couldn't get a wifi connection");
    while (true);
  }

  // print the list of networks seen:
  Serial.print("number of available networks:");
  Serial.println(numSsid);

  // print the network number and name for each network found:
  for (int thisNet = 0; thisNet < numSsid; thisNet++) {
    Serial.print(thisNet);
    Serial.print(") ");
    Serial.print(WiFi.SSID(thisNet));
    Serial.print("\tSignal: ");
    Serial.print(WiFi.RSSI(thisNet));
    Serial.print(" dBm");
    Serial.print("\tEncryption: ");
    printEncryptionType(WiFi.encryptionType(thisNet));
  }
```

```
}

void printEncryptionType(int thisType) {
  // read the encryption type and print out the title:
  switch (thisType) {
    case ENC_TYPE_WEP:
      Serial.println("WEP");
      break;
    case ENC_TYPE_TKIP:
      Serial.println("WPA");
      break;
    case ENC_TYPE_CCMP:
      Serial.println("WPA2");
      break;
    case ENC_TYPE_NONE:
      Serial.println("None");
      break;
    case ENC_TYPE_AUTO:
      Serial.println("Auto");
      break;
    case ENC_TYPE_UNKNOWN:
    default:
      Serial.println("Unknown");
      break;
  }
}

void printMacAddress(byte mac[]) {
  for (int i = 5; i >= 0; i--) {
    if (mac[i] < 16) {
      Serial.print("0");
```

```
    }
    Serial.print(mac[i], HEX);
    if (i > 0) {
      Serial.print(":");
    }
  }
  Serial.println();
}
```