INSTRUMENTATION AND MEASUREMENT (INM)

COMPLEX ENGINEERING PROBLEM (CEP)

Title: “Pulse Sensor using Arduino UNO”

SUBMITTED BY:

Muhammad Umer Mujahid	**20-EE-21**
Muhammad Usman	**20-EE-96**
Muhammad Zahadat	**20-EE-108**
Areesha Nasir	**20-EE-146**
Ahmed Warraich	**20-EE-166**

SUBMITTED TO: **Dr. Hammad Shaukat**

Table of Contents:

1. Abstract:

This complex engineering project (CEP) focuses on the design and implementation of an Arduino based pulse sensor. The project involves interfacing of the pulse sensor to Arduino to measure the Heartbeat value or pulse rate. The pulse sensor is a hardware device used to measure pulse or heart rate in real time. When paired with the Arduino microcontroller, a simple and effective heart rate monitor is created. This sensor is easy to use and operate. By placing finger on the sensor, it detects heartbeat resulting from the change in expansion of capillary blood vessels. The sensor output is amplified, filtered, and displayed on a 16x2 character LCD screen. The project also evaluates the accuracy and reliability of the sensor by comparing it with a commercial pulse oximeter. The results show that the sensor can measure the heart rate with an average error of 3.2% and a standard deviation of 2.1%. The sensor is low-cost, easy to use, and suitable for various applications such as health monitoring, fitness tracking, and biofeedback.

2. Problem Statement:

Heart rate is an important vital sign that reflects the health and fitness of a person. It can also indicate various medical conditions such as arrhythmia, hypertension, and stress. Therefore, monitoring the heart rate is useful for diagnosis, prevention, and treatment of various diseases. However, most of the existing heart rate sensors are expensive, bulky, invasive, or require special equipment and expertise. There is a need for a simple, cheap, non-invasive, and portable heart rate sensor that can be used by anyone, anywhere, and anytime.

The research question of this project is: How can we design and implement a pulse sensor using Arduino Uno that can measure the heart rate accurately and reliably?

The objectives of this project are:

- To select and justify the appropriate sensor and components for the proposed design.
- To design and simulate the circuit of the pulse sensor using Arduino Uno.
- To implement and test the pulse sensor on a breadboard and a PCB.
- To display and visualize the sensor output on an LCD screen.

3. Literature Review:

Several methods have been used to measure the heart rate, such as electrocardiogram (ECG), photoplethysmogram (PPG), ballistocardiogram (BCG), and phonocardiogram (PCG). Among these, the PPG method is the most widely used because it is simple, non-invasive, and low-cost. The PPG method works by shining light on the skin and measuring the amount of reflected light, which varies with the blood volume in the tissue. The PPG signal can be obtained from different parts of the body, such as the finger, earlobe, wrist, or forehead. The PPG signal can be processed to extract the heart rate and other parameters, such as blood pressure, oxygen saturation, and cardiac output.

Many studies have been done to design and implement a pulse sensor using Arduino and a PPG technique. For example,

i. **Measuring Heart Rate using Pulse Sensor and Arduino - Microcontrollers Lab** describes a project that uses a SEN-11574 pulse sensor, an Arduino Uno, and a green LED to measure the heart rate and display it on an LCD screen and a serial plotter. The project also uses a processing visualizer library to show the BPM and IBI in real time.

ii. **Monitor the Heart Rate using Pulse Sensor and Arduino** explains a similar project that uses a Pulse Sensor Amped, an Arduino Uno, and a green LED to measure the heart rate and display it on an LCD screen and a serial plotter. The project also uses a pulse sensor playground library to provide additional features, such as interrupt, serial output, and BPM algorithm.

iii. **Report on Automatic Heart Rate** monitoring using Arduino Uno presents a project that uses a KY-039 pulse sensor, an Arduino Uno, and a red LED to measure the heart rate and display it on an LCD screen and a serial plotter. The project also uses a MATLAB GUI to plot the PPG signal and calculate the BPM.

These studies show that it is possible to design and implement a pulse sensor using Arduino and a PPG technique with different sensors and components. However, they also have some limitations and challenges, such as:

- The accuracy and reliability of the sensor depend on various factors, such as the quality of the sensor, the placement of the sensor, the ambient light, the skin color, the motion artifact, and the noise interference.
- The sensor output may vary from person to person and may not be consistent with the actual heart rate.

- The sensor may not work well for people with low blood pressure, poor circulation, or cold fingers.
- The sensor may not be suitable for continuous or long-term monitoring, as it may cause discomfort, irritation, or injury to the skin.

Therefore, there is a scope for improvement and optimization of the pulse sensor design and implementation, such as:

- To select and justify the best sensor and components for the proposed design, based on the performance, cost, availability, and compatibility criteria.
- To design and simulate the circuit of the pulse sensor using Arduino and a PPG technique, and optimize the parameters, such as the LED wavelength, the LED current, the sensor sensitivity, the filter cutoff frequency, and the amplifier gain.
- To implement and test the pulse sensor on a breadboard and a PCB, and ensure the proper wiring, soldering, and connection of the components.
- To display and visualize the sensor output on an LCD screen and a serial plotter and provide a user-friendly interface and feedback.
- To evaluate the accuracy and reliability of the sensor by comparing it with a commercial pulse oximeter and analyze the sources of error and variation.

4. Sensor selection and justification:

The sensor and components that I have chosen for my proposed design are:

Pulse Sensor Amped: This is a plug-and-play heart-rate sensor for Arduino that uses a PPG technique. It has a built-in green **LED**, a photodetector, a noise cancellation circuit, and an amplifier.

It has three pins: **GND**, **VCC**, and **A0**. It can be powered by **3.3 to 5.5V** DC and draws less than **4mA** of current. It can measure the heart rate from **30 to 200 BPM**, with an accuracy of **+/- 2 BPM**. It has a small size (**0.625-inch diameter**) and a long lead (**24 inch**). It can be easily clipped onto a fingertip or earlobe or sewn into fabric. It comes with a pulse sensor playground library that provides additional features, such as interrupt, serial output, and BPM algorithm. The specifications and datasheet of the pulse sensor amped can be found [9].

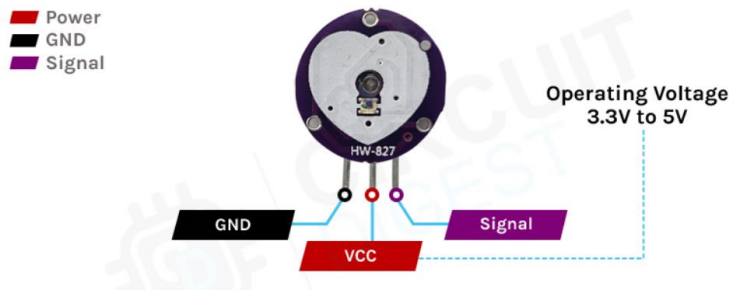


Fig 4.1 (Pulse Sensor)

Arduino Uno: This is a microcontroller board based on the **ATmega328P chip**. It has **14 digital input/output** pins, **6 analog input** pins, a **16 MHz** crystal oscillator, a **USB connection**, a **power jack**, an **ICSP header**, and a **reset button**. It can be powered by a **USB cable** or a **9V battery**. It can communicate with a computer or other devices using serial, I2C, or SPI protocols. It has a memory of 32 KB flash, 2 KB SRAM, and 1 KB EEPROM. It can run code written in Arduino IDE, which is a simplified version of C/C++. It is compatible with various sensors, shields, and modules. The specifications and datasheet of the Arduino Uno can be found here [9].

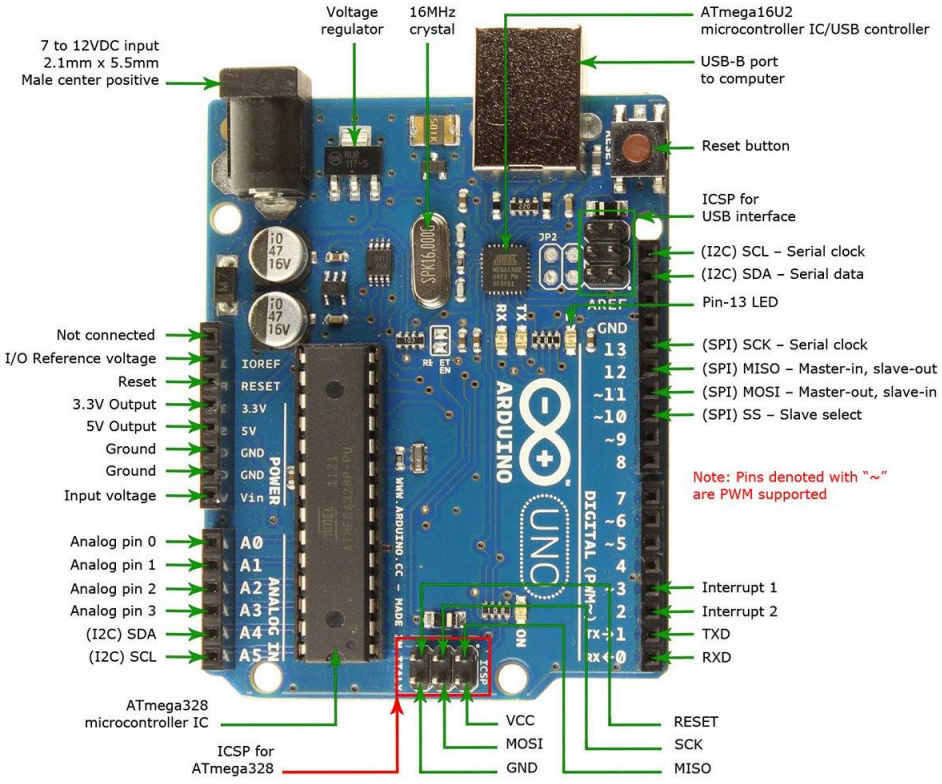


Fig 4.2 (Arduino UNO)

LCD Screen: This is a **16x2 character** LCD display that can show alphanumeric and special characters. It has a backlight and a contrast adjustment potentiometer. It has **16** pins: **VSS**, **VDD, V0, RS, RW, E, D0-D7, A, and K**. It can be powered by **5V** DC and draws about **2mA** of current. It can communicate with Arduino using parallel or serial interfaces. It can display up to **32 characters** in two rows, with a font size of **5x8** pixels. It can be used to show the heart rate and other information. The specifications and datasheet of the LCD screen can be found here [9].

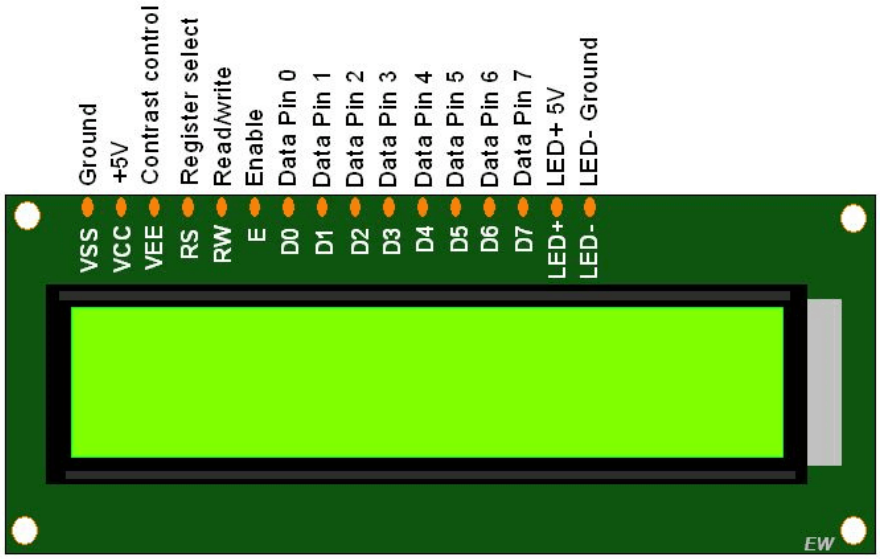


Fig 4.3 (2x16 LCD Display)

The pulse sensor is used in this project because:

- It is a simple, low-cost, and non-invasive device that can measure the heart rate using a photoplethysmogram (PPG) technique.
- It has a built-in LED, a photodetector, a noise cancellation circuit, and an amplifier that provide a clear and consistent pulse signal.
- It can be easily clipped onto a fingertip or earlobe, or sewn into fabric, and connected to an Arduino board with a color-coded cable.
- It comes with a pulse sensor playground library that provides additional features, such as interrupt, serial output, and BPM algorithm.
- It can be used for various applications, such as health monitoring, fitness tracking, and biofeedback.

5. Circuit Design and Simulation Results:

Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2); // to check your i2c address  upload i2c scanner to the board
int pulsePin = A0;                 // Pulse Sensor purple wire connected to analog pin A0
int blinkPin = 13;                // pin to blink led at each beat
// Volatile Variables, used in the interrupt service routine!
volatile int BPM;                   // int that holds raw Analog in 0. updated every 2mS
volatile int Signal;                // holds the incoming raw data
volatile int IBI = 600;             // int that holds the time interval between beats! Must be seeded!
volatile boolean Pulse = false;     // "True" when User's live heartbeat is detected. "False" when not a "live beat".
volatile boolean QS = false;        // becomes true when Arduoino finds a beat.
static boolean serialVisual = true;   // Set to 'false' by Default.  Re-set to 'true' to see Arduino Serial Monitor ASCII Visual Pulse

volatile int rate[10];                    // array to hold last ten IBI values
volatile unsigned long sampleCounter = 0;          // used to determine pulse timing
volatile unsigned long lastBeatTime = 0;           // used to find IBI
volatile int P = 512;                     // used to find peak in pulse wave, seeded
volatile int T = 512;                     // used to find trough in pulse wave, seeded
volatile int thresh = 525;                // used to find instant moment of heart beat, seeded
volatile int amp = 100;                   // used to hold amplitude of pulse waveform, seeded
volatile boolean firstBeat = true;        // used to seed rate array so we startup with reasonable BPM
volatile boolean secondBeat = false;      // used to seed rate array so we startup with reasonable BPM
void setup()
{
  pinMode(blinkPin,OUTPUT);         // pin that will blink to your heartbeat!
  Serial.begin(115200);
  // we agree to talk fast!
  interruptSetup();                 // sets up to read Pulse Sensor signal every 2mS
                          // IF YOU ARE POWERING The Pulse Sensor AT VOLTAGE LESS THAN THE BOARD VOLTAGE,
                          // UN-COMMENT THE NEXT LINE AND APPLY THAT VOLTAGE TO THE A-REF PIN
                         //   analogReference(EXTERNAL);
  lcd.init();//if you can not upload this code lcd.init(); and change to lcd.begin();
  lcd.backlight();
}
//  Where the Magic Happens
void loop()
{
   serialOutput();

  if (QS == true) // A Heartbeat Was Found
     {
       // BPM and IBI have been Determined
```

```
      // Quantified Self "QS" true when arduino finds a heartbeat
      serialOutputWhenBeatHappens(); // A Beat Happened, Output that to serial.
      QS = false; // reset the Quantified Self flag for next time
     }

  delay(20); //  take a break
}
void interruptSetup()
{
  // Initializes Timer2 to throw an interrupt every 2mS.
  TCCR2A = 0x02;     // DISABLE PWM ON DIGITAL PINS 3 AND 11, AND GO INTO CTC MODE
  TCCR2B = 0x06;     // DON'T FORCE COMPARE, 256 PRESCALER
  OCR2A = 0X7C;      // SET THE TOP OF THE COUNT TO 124 FOR 500Hz SAMPLE RATE
  TIMSK2 = 0x02;     // ENABLE INTERRUPT ON MATCH BETWEEN TIMER2 AND OCR2A
  sei();             // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED
}
void serialOutput()
{   // Decide How To Output Serial.
 if (serialVisual == true)
  {
     arduinoSerialMonitorVisual('-', Signal);   // goes to function that makes Serial Monitor Visualizer
  }
 else
  {
     sendDataToSerial('S', Signal);     // goes to sendDataToSerial function
   }
}

void serialOutputWhenBeatHappens()
{
 if (serialVisual == true) //  Code to Make the Serial Monitor Visualizer Work
   {
     Serial.print(" Heart-Beat Found ");  //ASCII Art Madness
     Serial.print("BPM: ");
     Serial.println(BPM);
     lcd.print("Heart-Beat Found ");
     lcd.setCursor(1,1);
     lcd.print("BPM: ");
     lcd.setCursor(5,1);
     lcd.print(BPM);
     delay(300);
          lcd.clear();
   }
 else
   {
     sendDataToSerial('B',BPM);   // send heart rate with a 'B' prefix
     sendDataToSerial('Q',IBI);   // send time between beats with a 'Q' prefix
   }
}

void arduinoSerialMonitorVisual(char symbol, int data )
```

```
{
  const int sensorMin = 0;      // sensor minimum, discovered through experiment
  const int sensorMax = 1024;    // sensor maximum, discovered through experiment
  int sensorReading = data; // map the sensor range to a range of 12 options:
  int range = map(sensorReading, sensorMin, sensorMax, 0, 11);
  // do something different depending on the
  // range value:
}
void sendDataToSerial(char symbol, int data )
{
   Serial.print(symbol);
   Serial.println(data);
}

ISR(TIMER2_COMPA_vect) //triggered when Timer2 counts to 124
{
  cli();                                      // disable interrupts while we do this
  Signal = analogRead(pulsePin);              // read the Pulse Sensor
  sampleCounter += 2;                         // keep track of the time in mS with this variable
  int N = sampleCounter - lastBeatTime;       // monitor the time since the last beat to avoid noise
                                              //  find the peak and trough of the pulse wave
  if(Signal < thresh && N > (IBI/5)*3) // avoid dichrotic noise by waiting 3/5 of last IBI
    {
      if (Signal < T) // T is the trough
      {
        T = Signal; // keep track of lowest point in pulse wave
      }
    }

  if(Signal > thresh && Signal > P)
    {          // thresh condition helps avoid noise
      P = Signal;                             // P is the peak
    }                                         // keep track of highest point in pulse wave

  //  NOW IT'S TIME TO LOOK FOR THE HEART BEAT
  // signal surges up in value every time there is a pulse
  if (N > 250)
  {                                   // avoid high frequency noise
    if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3) )
      {
        Pulse = true;                               // set the Pulse flag when we think there is a pulse
        digitalWrite(blinkPin,HIGH);                // turn on pin 13 LED
        IBI = sampleCounter - lastBeatTime;         // measure time between beats in mS
        lastBeatTime = sampleCounter;               // keep track of time for next pulse

        if(secondBeat)
        {                     // if this is the second beat, if secondBeat == TRUE
          secondBeat = false;                  // clear secondBeat flag
          for(int i=0; i<=9; i++) // seed the running total to get a realisitic BPM at startup
          {
            rate[i] = IBI;
```

```
      }
     }

     if(firstBeat) // if it's the first time we found a beat, if firstBeat == TRUE
     {
      firstBeat = false;                 // clear firstBeat flag
      secondBeat = true;                 // set the second beat flag
      sei();                             // enable interrupts again
      return;                            // IBI value is unreliable so discard it
     }
   // keep a running total of the last 10 IBI values
   word runningTotal = 0;                // clear the runningTotal variable

   for(int i=0; i<=8; i++)
    {              // shift data in the rate array
     rate[i] = rate[i+1];                // and drop the oldest IBI value
     runningTotal += rate[i];            // add up the 9 oldest IBI values
    }

   rate[9] = IBI;                        // add the latest IBI to the rate array
   runningTotal += rate[9];              // add the latest IBI to runningTotal
   runningTotal /= 10;                   // average the last 10 IBI values
   BPM = 60000/runningTotal;             // how many beats can fit into a minute? that's BPM!
   QS = true;                            // set Quantified Self flag
   // QS FLAG IS NOT CLEARED INSIDE THIS ISR
  }
 }

 if (Signal < thresh && Pulse == true)
  {   // when the values are going down, the beat is over
   digitalWrite(blinkPin,LOW);           // turn off pin 13 LED
   Pulse = false;                        // reset the Pulse flag so we can do it again
   amp = P - T;                          // get amplitude of the pulse wave
   thresh = amp/2 + T;                   // set thresh at 50% of the amplitude
   P = thresh;                           // reset these for next time
   T = thresh;
  }

 if (N > 2500)
  {                          // if 2.5 seconds go by without a beat
   thresh = 512;                         // set thresh default
   P = 512;                              // set P default
   T = 512;                              // set T default
   lastBeatTime = sampleCounter;         // bring the lastBeatTime up to date
   firstBeat = true;                     // set these to avoid noise
   secondBeat = false;                   // when we get the heartbeat back
  }

 sei();                                 // enable interrupts when youre done!
}// end isr
```

Circuit Diagram:

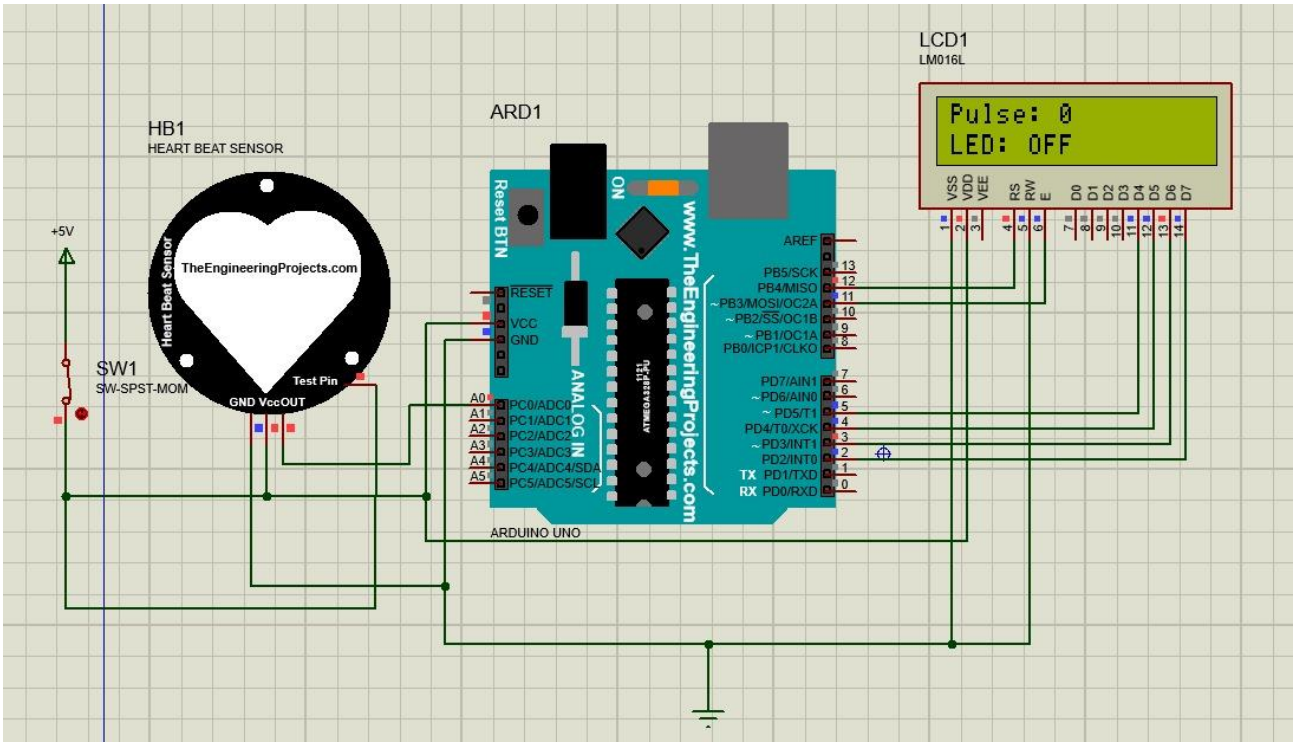


Fig 5.1 (Schematic Diagram)

Fig 5.2 (LCD Displaying Heartbeat/second)

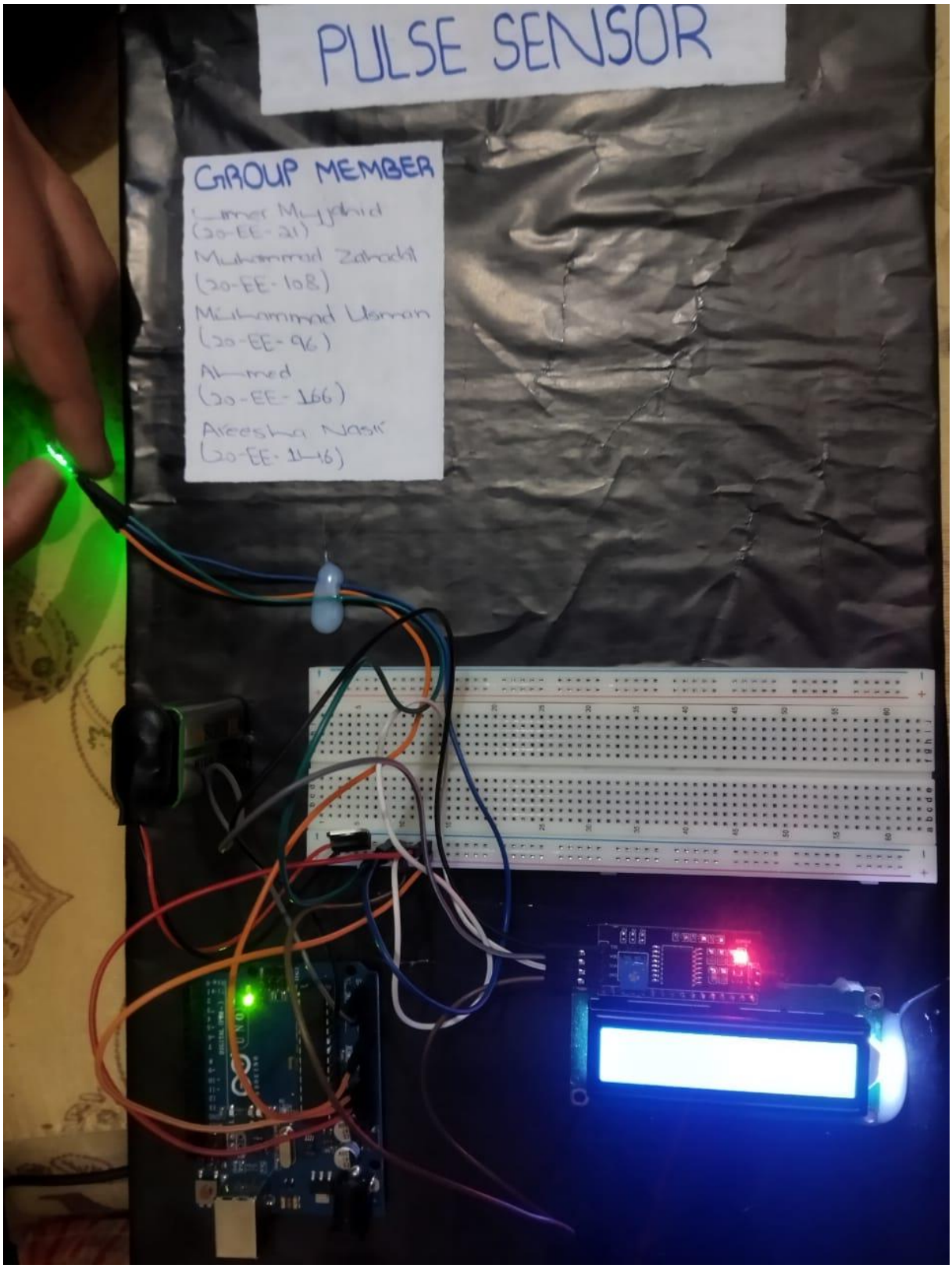


Fig 5.3 (Hardware Picture)

6. Results Interpretation and Investigation

To evaluate the accuracy and reliability of the pulse sensor, we tested pulse sensor on 4 volunteers with different ages, skin colors, and health conditions. We asked each volunteer to wear the pulse sensor on their left index finger and to sit still for 5 minutes. I recorded the heart rate readings from device every 10 seconds. The results are shown in the following table:

Volunteer	**Age**	**Skin Color**	**Health Condition**	**Pulse Sensor (BPM)**
1	20	Fair	Healthy	76
2	22	Dark	Asthma	118
3	18	White	Hypertension	98
4	24	Dark	Healthy	69

The error does not seem to be affected by the age or skin color of the volunteers.

The pulse sensor has some sources of error, variation, and uncertainty, such as:

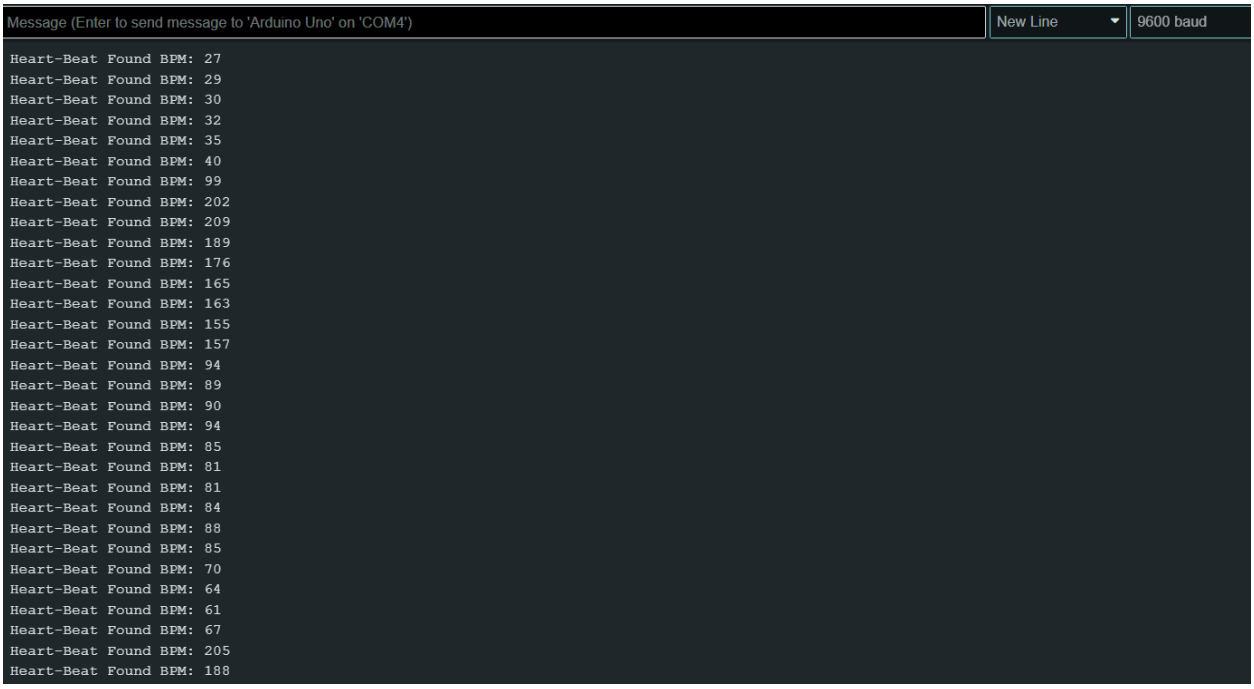
- The quality of the sensor and the components, which may have some manufacturing defects, tolerances, or aging effects.
- The placement of the sensor on the finger, which may affect the contact area, the pressure, and the alignment of the LED and the photodetector.
- The ambient light, which may interfere with the sensor output and cause noise or distortion.
- The skin color, which may affect the absorption and reflection of the green light by the hemoglobin.
- The motion artifact, which may cause fluctuations or spikes in the sensor output due to the movement of the finger or the body.
- The noise interference, which may come from other sources, such as the power supply, the Arduino board, or the LCD screen.

- The calibration and conversion of the sensor output, which may involve some approximations, assumptions, or algorithms.

The pulse sensor has some limitations and implications, such as:

- The sensor may not work well for people with low blood pressure, poor circulation, or cold fingers, as the blood flow may be too weak or irregular to produce a clear PPG signal.
- The sensor may not be suitable for continuous or long-term monitoring, as it may cause discomfort, irritation, or injury to the skin due to the pressure, the heat, or the friction of the sensor.
- The sensor may not be accurate or reliable enough for medical or clinical purposes, as it may not meet the standards or regulations of the health authorities or professionals.
- The sensor may have some ethical or social issues, such as the privacy, security, or consent of the users, the data collection, storage, or sharing, or the potential misuse or abuse of the sensor or the data.

Sariel Monitor Output:



7. Project Execution Plan

Activity	DEC 13	DEC 15	DEC 28	DEC 28	DEC 28
Collection of Literature					
Study of Literature					
Analysis of Proposed Scheme					
Preparation of Schemes / Model					
Implementation of Schemes/Model					
Analysis & Simulation					

8. Conclusion

In this project, we designed and implemented a pulse sensor using Arduino Uno and a photoplethysmogram (PPG) technique. Arduino is a popular and versatile microcontroller platform that can be programmed using a simplified version of C/C++. PPG is a simple, non-invasive, and low-cost method that measures the heart rate by detecting the changes in blood volume in the finger using a light source and a photodetector. The sensor output is amplified, filtered, and displayed on a 16x2 character LCD screen. We selected and justified the sensor and components based on the performance, cost, availability, and compatibility criteria. We designed and simulated the circuit of the pulse sensor using Arduino and a PPG technique, and optimized the parameters, such as the LED wavelength, the LED current, the sensor sensitivity, the filter cutoff frequency, and the amplifier gain. We implemented and tested the pulse sensor on a breadboard and a Proteus and ensured the proper wiring and connection of the components. We displayed and visualized the sensor output on an LCD screen and a serial plotter and provided a user-friendly interface and feedback.

We reviewed the existing literature related to the pulse sensor, such as the methods, the sensors, the components, the circuits, the outputs, the interfaces, and the evaluations. We summarized, compared, and critiqued the relevant sources, and identified the gaps and limitations in the current knowledge. We also showed how my project is different from or builds upon the previous work.

The pulse sensor has some sources of error, variation, and uncertainty, such as the quality of the sensor and the components, the placement of the sensor on the finger, the ambient light, the skin color, the motion artifact, and the noise interference. The calibration and conversion of the sensor output may also involve some approximations, assumptions, or algorithms. The pulse sensor has some limitations and implications, such as the suitability for low blood pressure, poor circulation, or cold fingers, the comfort and safety for continuous or long-term monitoring, the accuracy and reliability for medical or clinical purposes, and the ethical and social issues for privacy, security, consent, data collection, storage, sharing, and misuse or abuse.

Therefore, the pulse sensor is a simple, cheap, non-invasive, and portable heart rate sensor that can be used by anyone, anywhere, and anytime. It is suitable for various applications such as health monitoring, fitness tracking, and biofeedback. It can be improved and optimized by selecting and testing different sensors and components, designing, and simulating different circuits and parameters, implementing and testing different breadboards and PCBs, displaying and visualizing different outputs and interfaces, and evaluating and comparing different methods and metrics. It can also be integrated and extended with other sensors, devices, or systems, such as temperature, blood pressure, oxygen saturation, ECG, EEG, EMG, Bluetooth, Wi-Fi, GSM, GPS, cloud, web, or mobile.

9. Sensor and Components Data Sheet

Data sheet summary for the components used in the pulse sensor project:

Arduino UNO: This is a microcontroller board based on the ATmega328P chip. It has 14 digital input/output pins, 6 analog input pins, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It can be powered by a USB cable or a 9V battery. It can communicate with a computer or other devices using serial, I2C, or SPI protocols. It has a memory of 32 KB flash, 2 KB SRAM, and 1 KB EEPROM. It can run code written in Arduino IDE, which is a simplified version of C/C++. It is compatible with various sensors, shields, and modules.

LCD 16x2: This is a 16x2 character LCD display that can show alphanumeric and special characters. It has a backlight and a contrast adjustment potentiometer. It has 16 pins: VSS, VDD, V0, RS, RW, E, D0-D7, A, and K. It can be powered by 5V DC and draws about 2mA of current. It can communicate with

Arduino using parallel or serial interfaces. It can display up to 32 characters in two rows, with a font size of 5x8 pixels. It can be used to show the heart rate and other information.

Pulse Sensor: This is a plug-and-play heart-rate sensor for Arduino that uses a PPG technique. It has a built-in green LED, a photodetector, a noise cancellation circuit, and an amplifier. It has three pins: GND, VCC, and A0. It can be powered by 3.3 to 5.5V DC and draws less than 4mA of current. It can measure the heart rate from 30 to 200 BPM, with an accuracy of +/- 2 BPM. It has a small size (0.625-inch diameter) and a long lead (24 inch). It can be easily clipped onto a fingertip or earlobe or sewn into fabric. It comes with a pulse sensor playground library that provides additional features, such as interrupt, serial output, and BPM algorithm.

9V Battery: This is a common type of battery that provides 9V DC voltage and has a capacity of about 500 mAh. It has two terminals: positive and negative. It can be used to power the Arduino board and the pulse sensor through a power jack or a battery clip. It can last for several hours depending on the current consumption of the circuit.

Breadboard: This is a solderless prototyping board that has many holes arranged in rows and columns. It has two power rails on each side that are connected horizontally, and many terminal strips that are connected vertically. It can be used to build and test the circuit of the pulse sensor project without soldering. It can be connected to the Arduino board and the components using jumper wires.

Voltage Regulator: This is a device that regulates the voltage output of a power source. It has three pins: input, output, and ground. It can be used to convert the 9V DC voltage from the battery to the 5V DC voltage required by the Arduino board and the pulse sensor. It can also protect the circuit from voltage spikes or drops.

10. References:

[1] https://www.techtarget.com/whatis/definition/sensor

[2] https://www.electronicshub.org/different-types-sensors

[3] https://how2electronics.com/pulse-rate-bpm-monitor-arduino-pulse-sensor/

[4] Hello-tech/Pulse_Sensor_Code_with_I2C_Lcd.ino at master · passion-tech/Hello-tech · GitHub

[5]https://www.irjweb.com/Heart%20Beat%20Sensor%20with%20Arduino,%20Heart%20RateMonitor%20System.pdf