Minor Project

Heart Rate Monitor

By Shibjyoti Das,
DEIE, 5th Semester
(D222302316)

ABSTRACT

This project demonstrates the application of Arduino technology to create a functional and user-friendly heart rate monitor. By integrating an Arduino Uno, an OLED display, and a pulse sensor, the device provides real-time heart rate measurements and offers basic health advisories.

The OLED display is connected to the Arduino via the I2C protocol, while the pulse sensor is interfaced with the analog input pin A0. The sensor detects blood volume changes in the fingertip and produces an analog signal that the Arduino processes to calculate the heart rate. The OLED display then presents the BPM and a visually engaging heart icon.

The project's code incorporates several key functionalities. These include initializing the display and sensor, continuously monitoring sensor readings, applying a smoothing algorithm to reduce noise, calculating BPM, and providing health advisories based on the heart rate. A threshold and debounce mechanism are implemented to ensure accurate heart rate measurements.

The heart rate monitor offers basic health advice based on the measured BPM. For example, if the heart rate is below 60 BPM, it recommends seeing a doctor as this may indicate a medical condition; if the heart rate falls within the normal resting range of 60 to 100 BPM, it indicates a healthy heart rate; if the heart rate is between 100 and 140 BPM, it suggests an elevated heart rate that could be due to exercise, stress, or other factors; and if the heart rate is above 140 BPM, it warns that a very high heart rate may be a sign of a medical problem and recommends seeking medical attention.

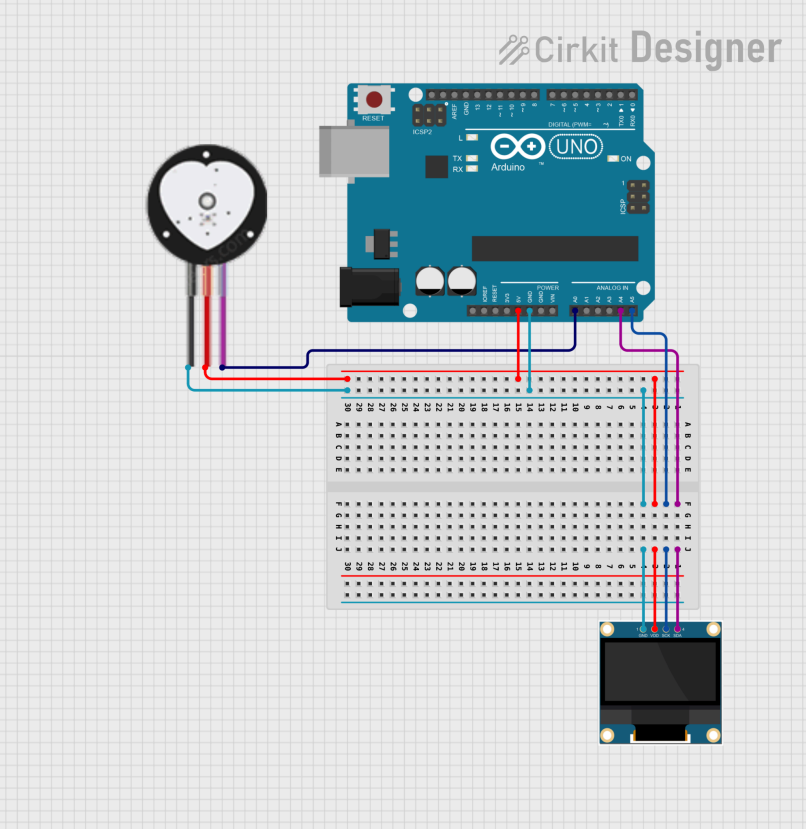
THEORY

The pulse sensor detects changes in light transmission through the skin as blood flow pulses, producing an analog signal. This signal corresponds to the intensity of blood flow, peaking with each heartbeat. The sensor outputs a voltage, which is sampled and smoothed using a moving average algorithm to reduce noise. A threshold-based algorithm detects a beat by identifying a rise and fall in the signal amplitude. The time between consecutive beats (inter-beat interval) is used to calculate the heart rate in BPM. The OLED display shows the current BPM along with a health advisory based on standard heart rate ranges.

APPARATUS REQUIRED

- Arduino UNO
- 128×64 OLED display
- Pulse sensor
- Breadboard
- Jumper wires

CIRCUIT DIAGRAM



OPERATION

1. **Signal Acquisition**: The pulse sensor outputs an analog voltage signal proportional to blood flow.
2. **Smoothing**: The raw sensor data is averaged over a window of 4 readings to reduce noise and provide a smoother signal.
3. **Beat Detection**: A beat is detected when the smoothed signal crosses a threshold (set at 550) and follows a rising and falling pattern.
4. **BPM Calculation**: The time between consecutive beats is measured in milliseconds, and the BPM is calculated as:

$$BPM = \frac{60000}{Time\ between\ two\ beats\ (in\ ms)}$$

5. **Display**: The BPM and a corresponding health advisory are displayed on the OLED. A beating heart icon provides visual feedback for each detected beat.

bpm_monitor.ino File:

(View on GitHub)[1]

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SMOOTHING_WINDOW_SIZE 4
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

const int PULSE_SENSOR_PIN = A0;
const int LED_PIN = 13;

const int SAMPLE_INTERVAL = 100; // Time between samples in milliseconds const int
BEAT_THRESHOLD = 550; // Adjust based on your sensor's characteristics const int
NO_OBJECT_THRESHOLD = 400;

const unsigned long MIN_BEAT_INTERVAL = 600; // Minimum time between beats (200
BPM)
const unsigned long MAX_BEAT_INTERVAL = 1500; // Maximum time between beats (40
BPM)

unsigned long lastBeatTime = 0;
```

[1] https://github.com/shibjyoti555/arduino/blob/main/bpm_monitor.ino

```
int lastReading = 0;
bool rising = false;
float beatsPerMinute = 0;
bool objectPresent = false;

int readings[SMOOTHING_WINDOW_SIZE] = {0}; // Array for smoothing
int readIndex = 0;
int total = 0;

int getSmoothedReading(int reading) {
  total = total - readings[readIndex]; // Subtract the oldest reading
  readings[readIndex] = reading; // Store new reading total = total +
  readings[readIndex]; // Add the new reading readIndex = (readIndex + 1) %
  SMOOTHING_WINDOW_SIZE;
  return total / SMOOTHING_WINDOW_SIZE; // Return the average }


// Heart icon (7x7 pixels)
const unsigned char PROGMEM heart_bmp[] = {
  B00100100,
  B01111110,
  B11111111,
  B11111111,
  B01111110,
  B00111100,
  B00011000
};

// Smaller heart for "beating" effect
const unsigned char PROGMEM heart_small_bmp[] = {
  B00000000,
  B00100100,
  B01111110,
  B01111110,
  B00111100,
  B00011000,
  B00000000
};

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

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }
```

```
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.println("Heart Rate Monitor");
  display.setTextSize(1);
  display.println("\n~~shibjyoti");
  display.display();
  delay(2000);
  display.clearDisplay();
}

void loop() {
  display.clearDisplay();
  static unsigned long lastSampleTime = 0;
  unsigned long currentTime = millis();
  int reading = getSmoothedReading(analogRead(PULSE_SENSOR_PIN)); // Smoothed
reading

  // Serial.println(SAMPLE_INTERVAL);
  if (currentTime - lastSampleTime >= SAMPLE_INTERVAL) {
    lastSampleTime = currentTime;
    int reading = analogRead(PULSE_SENSOR_PIN);

    if (reading < NO_OBJECT_THRESHOLD) {
      if (objectPresent) {
        objectPresent = false;
        display.clearDisplay();
        // Serial.println("No object detected");
        display.setTextSize(2);
        display.setCursor(0, 0);
        display.println("No object");
        display.println("Place finger on sensor");
        display.display();
       }
    } else {
      objectPresent = true;
      bool beatDetected = detectBeat(reading, currentTime);

      if (beatDetected) {
        unsigned long beatInterval = currentTime - lastBeatTime;
        if (beatInterval >= MIN_BEAT_INTERVAL && beatInterval <= MAX_BEAT_INTERVAL) {
          beatsPerMinute = 60000.0 / beatInterval;
          Serial.println(beatInterval);
          display.clearDisplay();
          displayHeartRate(beatsPerMinute, true);
          getHealthAdvisory(beatsPerMinute);
          digitalWrite(LED_PIN, HIGH);
        }
```

```
        lastBeatTime = currentTime;
      } else {
        displayHeartRate(beatsPerMinute, false);
        digitalWrite(LED_PIN, LOW);
      }
    }
  }
  delay(100);
}

bool detectBeat(int reading, unsigned long currentTime) {
  static unsigned long lastDebounceTime = 0;
  bool beat = false;

  if (reading > BEAT_THRESHOLD && reading > lastReading && !rising) {
    rising = true;
  } else if (reading < lastReading && rising) {
    unsigned long debounceDelay = 200;
    if (currentTime - lastDebounceTime > debounceDelay) {
      rising = false;
      beat = true;
      lastDebounceTime = currentTime;
    }
  }

  lastReading = reading;
  return beat;
}


void displayHeartRate(float bpm, bool beating) {
  display.setTextSize(2);
  display.setCursor(0, 16);
  display.print(bpm, 1); // Print BPM value without clearing the whole display
  display.println(" BPM");
  display.setTextSize(1);
  display.println(getHealthAdvisory(bpm));

  // Display the heart icon (update only if beating status changes)
  if (beating) {
    display.drawBitmap(110, 0, heart_bmp, 8, 7, SSD1306_WHITE);
  } else {
    display.drawBitmap(110, 0, heart_small_bmp, 8, 7, SSD1306_WHITE); }

  display.display();
}


const char* getHealthAdvisory(float bpm) {
```

```
  if (bpm < 60) {
    return "Low. Consult doctor if symptomatic";
  } else if (bpm >= 60 && bpm <= 100) {
    return "Normal resting adult heart rate.";
  } else if (bpm > 100 && bpm <= 140) {
    return "Elevated. Normal during exercise or nervousness";
  } else {
    return "Very high. Seek medical attention if persistent";
  }
}
```

APPLICATIONS

- Monitoring heart rate during exercise or rest.
- Health monitoring devices for fitness tracking.
- Biomedical signal processing experiments.

ADVANTAGES

- **Real-time Heart Rate Monitoring**: The system provides continuous, real-time heart rate feedback, making it useful for fitness tracking or health monitoring.
- **Noise Reduction via Smoothing**: The system employs a smoothing algorithm to filter out noise from the pulse sensor readings, ensuring more accurate heart rate detection.
- **Customizable Parameters**: The beat threshold, sampling rate, and other parameters can be easily adjusted to accommodate different sensors or environments, providing flexibility.
- **Simple and Low-Cost Hardware**: The system uses affordable components, making it accessible for hobbyists, students, or developers working on biomedical applications.
- **Health Advisory Feedback**: It not only measures BPM but also provides instant health advice based on predefined heart rate ranges, offering basic health monitoring.
- **Energy Efficient**: The OLED display and Arduino setup are energy efficient, making the system suitable for battery-powered applications.

ERROR

Ideal Time between Beats (T_o): This is calculated using the actual heart rate.

$$T_0 = \frac{60000}{True\ BPM}$$

Measured Time (T_m): This is the time interval measured by the system.
Percentage Error in BPM:

$$Error\ (\%) = \frac{|T_0 - T_m|}{T_0} \times 100$$

Error Calculation:

Beat intervals measured by the system(T_m):

862
710
1166
918
710
862
763 [2]

Average $T_m = \frac{862+710+1166+918+710+862+763}{7} ms$

$= \frac{5991}{7} ms$

$= 855.857\ ms$

$T_0 = 630\ ms$ [3]

$$Error\ (\%) = \frac{|T_0 - T_m|}{T_0} \times 100\%$$

$$= \frac{|630 - 855.857|}{855.857} \times 100\%$$

$$= 0.26389 \times 100\%$$

$$= 26.389\%$$

[2] From Serial Monitor of Arduino IDE
[3] Measured in a commercially available calibrated apparatus

CONCLUSION

In conclusion, this project provides a simple, low-cost solution for real-time heart rate monitoring using an Arduino, pulse sensor, and OLED display. It offers reliable performance with noise reduction, visual feedback, and basic health advisories, making it ideal for personal fitness tracking and health-related applications. While the project is relatively simple, it demonstrates the potential of Arduino and related technologies to create practical and meaningful applications in the field of healthcare.

BIBLIOGRAPHY

1. Adafruit SSD1306 library by Adafruit
   https://github.com/adafruit/Adafruit_SSD1306
2. Adafruit GFX Library by Adafruit
   https://github.com/adafruit/Adafruit-GFX-Library
3. What is a normal pulse rate? - British Heart Foundation
   https://www.bhf.org.uk/informationsupport/heart-matters-magazine/medical/ask-the-experts/pulse-rate
4. What's a normal resting heart rate? - Mayo Clinic
   https://www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979

1

2

3

4