Politechnika Wrocławska

PULSEMETER SENSOR

Optoelectronics-project

Laboratory assistant:mgr inż. Michał Szubert

Wrocław 2024

Name, index no., department, group	Marcin Kochalski 275514 Patryk Krawczyk 275515 Group nr. E43 WEFIM
Report submission date	16.01.2025

Table of contents

1.Introduction……………………..…………………...………………………………..……3
2.Technical introduction…………………..……………………...……………..…………4
3.Assumptions……………………..……………………..……………………………………5
4.Description of the hardware part…………….…….……………….....………………………5
5.Description of the software………………..…………………..……………..……………8
6.Start-up, calibration……………………..……………………….…..………………………..10
7.Test measurements……………………..………………………………….…………………..10
8.User manual……………………………..…………………………………………….……..……10
9.Summary………………………..………………………………………………………………11
10.Bibliography………...…………………..……………………………………………..…..11
11.Appendix………………...……………………………………………….………………...……12

1.Introduction

Presented report introduces the reader into the main points of creating the pulse sensor by us. It contains all the key processes we used in the creation.

- Theoretical introduction - Explains the technical side of our project and the phenomena that make it possible.
- Assumptions - Shows our expectation for this project
- Description of the hardware - Description of all parts used in our project
- Description of the software - Shows and explains the code we use
- Start-up and calibration - Explain how to start-up the project
- Test measurements - Test that shows how performance of our device
- User manual - How-to-use instruction
- Summary - Sums up our project
- Bibliography
- Appendix

We decided to choose this project considering that we just started working with hardware this semester. It combines simplicity with major principles of optoelectronics, making it a valuable experience. Also, main elements of the setup, like the sensor, were to be found in laboratory class, making it easier for us to work.

2. Theoretical introduction

Photoplethysmography(PPG) was invented in 1937 by an American Physiologist named Alrick B. Hertzman. Oximeters work because hemoglobin in the blood absorbs red and infra-red light differently depending on how much attached oxygen it has. Our PulseSensor is **not** a Pulse Oximeter, it measures heart rate only. The method that is used in this project is the LED and sensor are next to each other, and it measures changes in light reflection due to pulsatile blood flow. Then, sample the sensor output at a regular rate and graph the pulse waveform. The best place on the body to measure pulse are fingertips and earlobes because the changes in light are the most visible here.Accurate results depend on sampling the sensor output at a consistent and rapid rate. We use a sampling frequency of 500 Hz (2 ms interval) based on validated scientific recommendations. This rate provides sufficient resolution for precise beat-to-beat timing, improving heart rate accuracy and reliability. The wavelength of light impacts PPG performance. Red and infrared lights are common in pulse oximeters due to their sensitivity to oxygen levels, but green light often offers a better signal for heart rate monitoring. Green light, less affected by deeper tissue movement, improves signal clarity. That's why in our project we used green light, because it works the best with measuring pulse.When calibrated correctly, the output from the PulseSensor saturates at the analog read range limits, either 0 or 1023, when there is no physical contact with the sensor. This occurs because the absence of tissue contact disrupts the normal light transmission or reflection path, causing the signal to reach extreme values. Once contact is reestablished and the sensor has time to stabilize, the output gradually returns to a steady value around the midpoint of the analog range, typically 512. Proper calibration is essential to ensure consistent and accurate signal detection, as it prevents signal drift and maintains reliable readings for heart rate analysis.

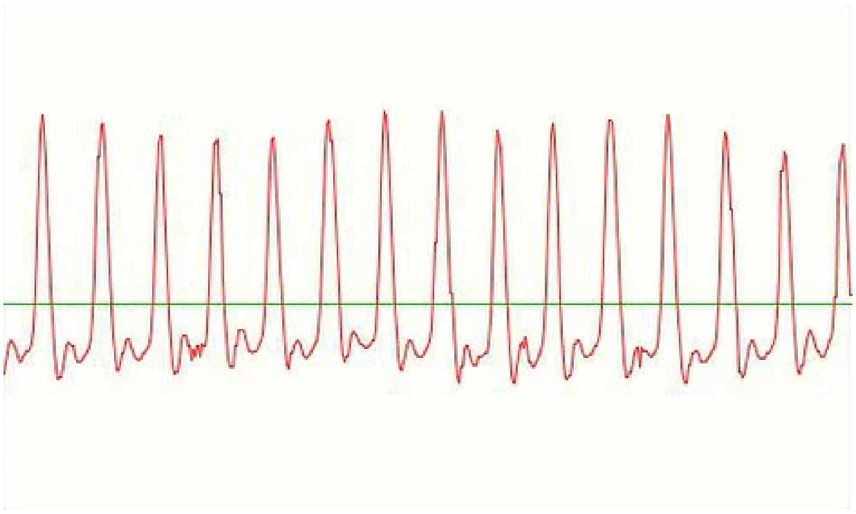

Fig1. Heartbeat graph

Formula to calculate BPM

$$BPM = \frac{60000}{IBI(ms)} \qquad (1)$$

3. Assumptions

3.1. Functional assumptions

- The device correctly measures the BPM with 90% accuracy
- The device correctly displays the heartbeat graph in Arduino grapher and displays the BPM on LCD screen
- The device is stable in measurements
- The device is not susceptible to overheating, noise and voltage drops
- The device is portable

3.2. Design assumptions

- - For the MCU part, we chose the Arduino Uno R3 with Atmega328p
- - For the sensor part, we chose PulseSensor from World Famous Electronics llc
- - For screen we are using LCD 1602 with I2C module
- - The circuit is supposed to be portable, so we assumed batteries as power supply

4. Description of the hardware part

4.1 Diagram + description

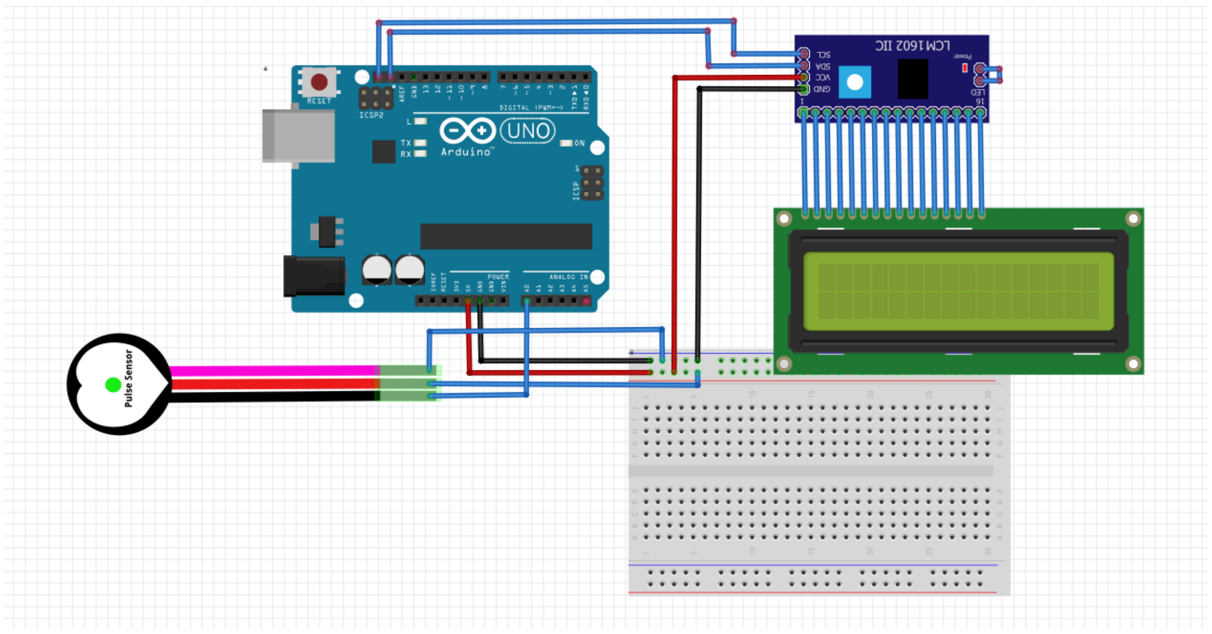

Fig2. Project diagram

To acquire data, we use the PulseSensor module. Main principle of its task is measuring the intensity of light reflected from the fingertip. It is connected to Arduino Uno for data processing and also it is powered by its 5V port and grounded. After processing the data, the result is displayed on the 16x2 LCD. To simplify the connection, we used I2C module for data transmission, powering and grounding the LCD. The Arduino board can be powered either by battery or USB, which then powers the whole system.

4.2 Photo of the actual system with reference to the diagram

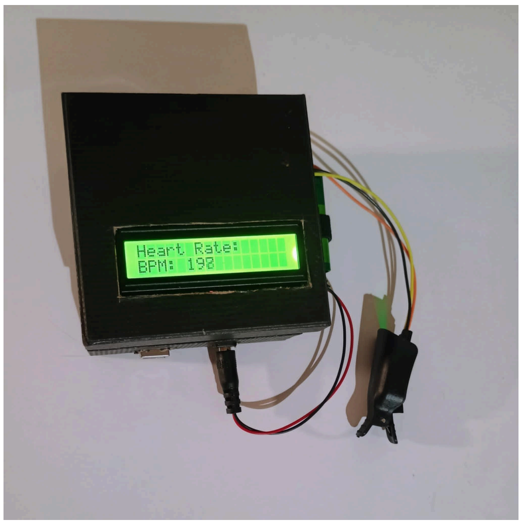

Fig3. Photo of the project

In the photo we can see only the LCD screen, battery, measuring clip and connecting cables. The rest of the system is inside the case - Arduino R3 microcontroller and I2C module to the LCD screen. The pulsemeter sensor is inside of the measuring clip, because it doesn't work outside of it.

4.3 Key elements - description

- PulseSensor module - acquires the data in analog mode. Measures the intensity of light reflected from the fingertip.

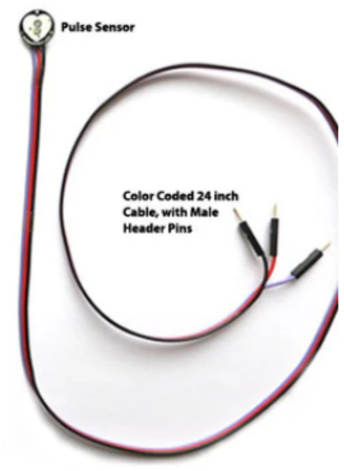

Fig4. Pulsesensor

- Arduino Uno R3 - Processes the analog signal into digital signal (ADC), analyzes the signal with software written by us and proceeds to output the results on LCD. Uses Atmega328p MCU.

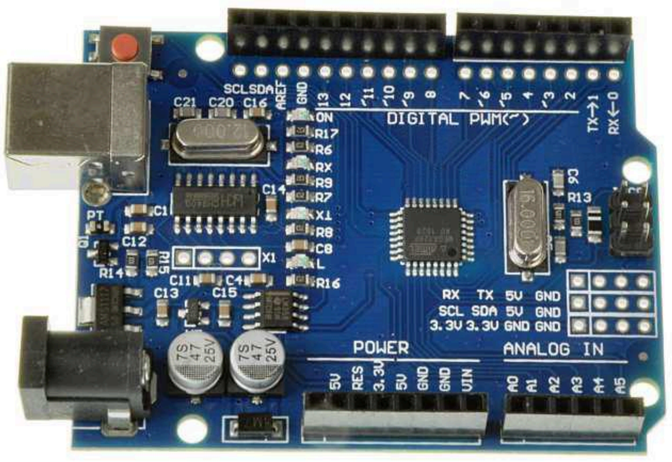

Fig5. Arduino Uno

- Lcm1602 - I2C converter which simplifies the connection between Arduino and LCD.
- 16x2 LCD display - 16 characters in one row, double row display. Displays the results of the measurements.

Fig6. LCD screen with I2C module

- Case - Case is made of black carton box and top and bottom part are connected using plastic brackets. The case makes sure that the device is safe from external damage. Everything inside the case is glued together to make sure it will be stable. Fig. A1 Pulsemeter case top view , Fig. A3. Pulsemeter case bottom view in the Appendix.

5. Description of the software part

5.1 Main algorithm - diagram + description

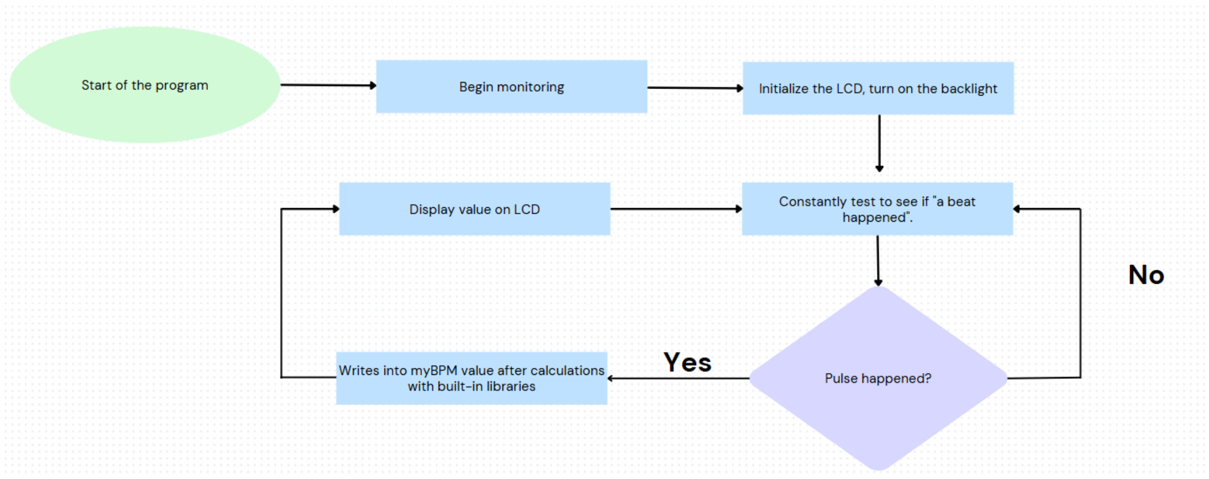
Start of the program
Begin monitoring
Initialize the LCD, turn on the backlight
Display value on LCD
Constantly test to see if "a beat happened".
No
Writes into myBPM value after calculations with built-in libraries
Yes
Pulse happened?

Fig7. Algorithm block diagram

Full code can be found in the Appendix Fig. A4. Pulsemeter code.

5.2 Description of key functions

Most of the key functions are predefined by built-in libraries. After defining the variables and ports, initialization begins.

```
  Serial.begin(115200); - For monitoring in Arduino software

  // Initialize LCD
  lcd.init();            - Initialization of LCD
  lcd.backlight();       - Enabling the backlight
  lcd.setCursor(0, 0); - Setting the first position for the display
  lcd.print("Initializing..."); - Testing the LCD
```

Later, we created the PulseSensor related objects.

```
pulseSensor.analogInput(PulseWire); - PulseSensor object
pulseSensor.blinkOnPulse(LED); - PCB blinks on heartbeat appearance
pulseSensor.setThreshold(Threshold); - Threshold for filtering out the
fake heartbeats.
```

After that, we inform the user if the initialization was succesful:

```
  if (pulseSensor.begin()) {
    Serial.println("PulseSensor initialized!");
    lcd.setCursor(0, 0);
    lcd.print("Pulse Sensor Ready");
  } else {
    Serial.println("PulseSensor initialization failed!");
    lcd.setCursor(0, 0);
    lcd.print("Sensor Error!");
  }

  delay(2000);
  lcd.clear();
}
void loop() {
  if (pulseSensor.sawStartOfBeat()) - If the threshold was passed, the
device recognizes a heartbeat
{
    int myBPM = pulseSensor.getBeatsPerMinute(); - Stores beat into
myBPM integer
```

5.3 Transmission protocols, key variables and parameters
The communication between Arduino Uno and LCD is made easier by using I2C module. It reduces the cabling complexity needed to connect the LCD. For the PulseSensor module, it communicates with Arduino Uno using single wire setup. The data is later processed after using ADC.

```
int Threshold = 512; - Due to the noise, we decided the value of 512
would filter out any noise that would imitate a heartbeat. If the
signal passes this value, it counts as a heartbeat.

const int PulseWire = 0; - PulseSensor purple wire (data) connected to
pin0 on Arduino Uno.

const int LED = LED_BUILTIN; - Definition of the LED that blinks on
heartbeat happening.

lcd.print(myBPM); - myBPM is calculated with built-in library. The
beats per minute value is stored there and printed with LCD.
```

Also, the LCD parameters are 2 rows and 16 columns for signs writing. For the SerialMonitor, we used 115200 baud rate for accuracy.

6. Start-up, calibration

6.1 First Start-up

- Connect 9V batteries through the included cable. Another option is to connect the device through USB cable, this will allow us to use the device without batteries.
- If the device is not working after connecting make sure that the program is inserted to the microcontroller. To do that, get the program from the included pendrive and insert it using a USB cable.
- To measure your pulse, insert your finger into the measuring clip, and wait for a reading, the device needs a moment to measure it correctly.
- To ensure that the measurement is correct make sure you put your finger in correctly.

6.2 Calibration:

Pulsemetr doesn't need any extra configuration, it always has the same lighting conditions because it is inside the measuring clip which makes it independent of the conditions outside. Because of that it's ready to use right after connecting the power.

7. Test measurements

7.1 Measurement Conditions

- Big indoor room with natural light and with ceiling light
- Sensor placement near the windows

7.2 Results

Tab1 Testing result

Results from our pulsemeter	Result from normal pulsemeter
95	100
85	80
80	83

As we can observe, the results from our pulsemeter are really close to the normal store bought pulsemeter, which proves that our device works correctly.

8. User manual

8.1 Package contents

- 1x Pulsemeter
- 1x USB cable
- 1x 9V battery cable
- 1x 9V battery
- Pendrive with backup program

8.2 Instruction

- Before using make sure that battery is charged for optimal performance, and place the batteries in the designated place to avoid damage to the device.
- To use the pulsemeter connect the device with the 9V battery through the included cable, or through the USB cable
- After connecting the battery the pulsemeter is ready to use
- To measure your pulse, insert your finger into the measuring clip, and wait for a reading, the device needs a moment to measure it correctly.

8.3 Safety

- Make sure that battery is in the designated place to avoid damage
- Do not expose device to heat, it can cause sensor to being damaged

8.4 Troubleshooting

- Inaccurate Reading: Make sure that your finger is clean and make sure that you are placing it correctly
- Device is not turning on: Check the battery level, and make sure that battery connection is right. If this doesn't help make sure that the program is inserted to the microcontroller. To do that, get the program from the included pendrive and insert it using a USB cable.

9. Summary

The pulse meter project demonstrated the application of optoelectronics principles by using a reflective PPG sensor to measure heart rate. We achieved a working system capable of displaying BPM on an LCD screen with an accuracy of approximately 90%. The project helped us understand analog-to-digital conversion, signal filtering, and I2C communication. We face a lot of problems building this pulsemeter. Firstly we burn the first sensor during soldering, because its sensitive to high temperatures. Later it was showing random values instead of actual pulse, because we were using the sensor without any case, and it was too light for the sensor to successfully read the changes in the light intensity. Future improvements could involve refining signal processing algorithms for increased noise resistance and exploring more ergonomic designs for broader usability.

10. Bibliography

[1] https://pulsesensor.com
[2] https://pulsesensor.com/pages/datasheet
[3] https://pulsesensor.com/pages/getting-advanced

11. Appendix

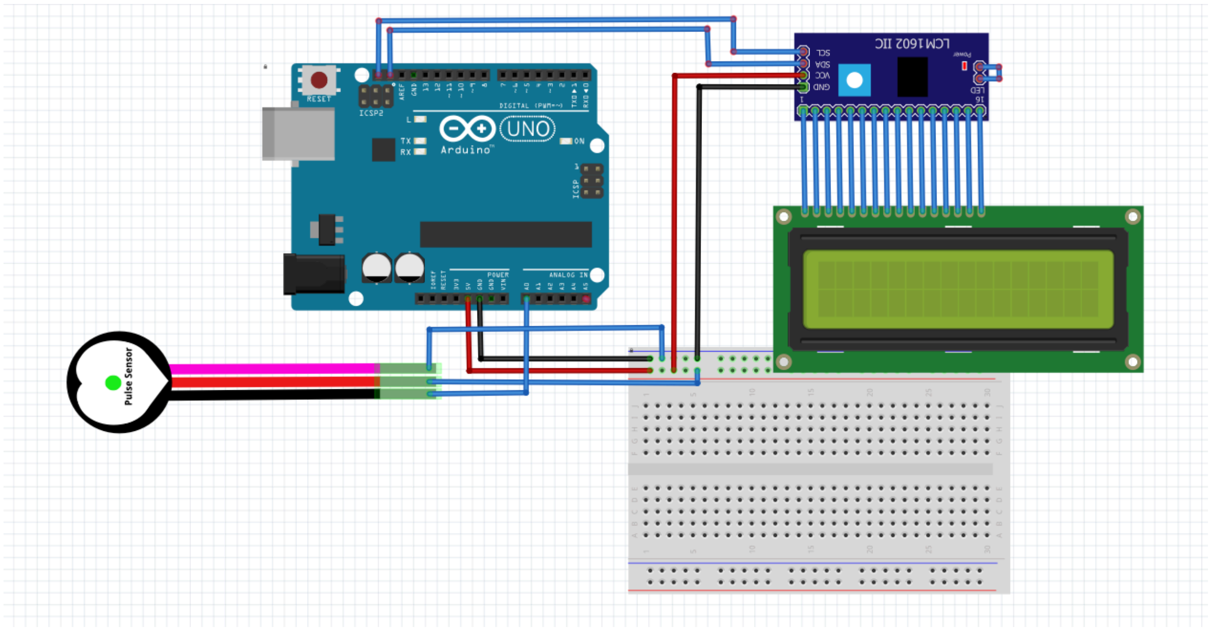

Fig. A1. Project diagram

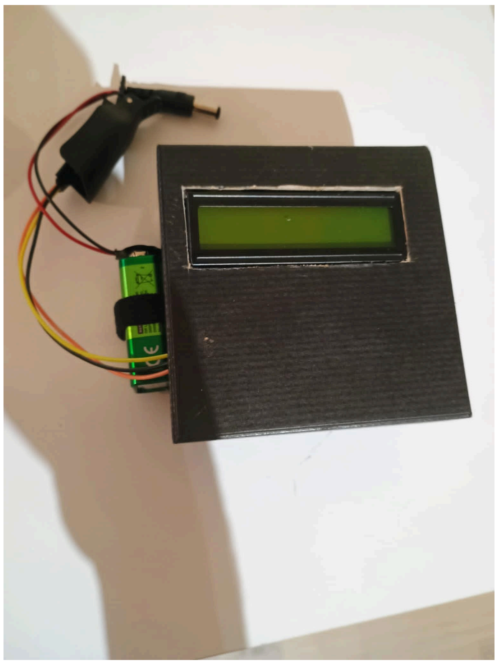

Fig. A2. Pulsemeter case top view

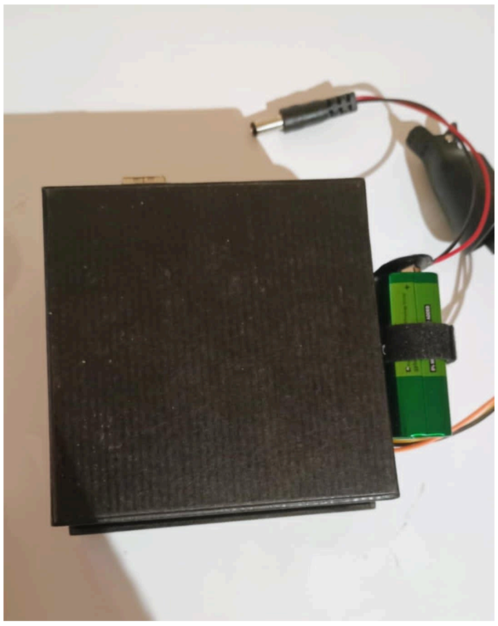

Fig. A3. Pulsemeter case bottom view

```
#include <PulseSensorPlayground.h>     // Includes the
PulseSensorPlayground Library.
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Pulse Sensor Variables
const int PulseWire = 0;       // PulseSensor PURPLE WIRE connected to
ANALOG PIN 0
const int LED = LED_BUILTIN;   // The on-board Arduino LED, close to
PIN 13.
int Threshold = 512;           // Determine which Signal to "count as a
beat".
                               // Use the "Getting Started Project" to
fine-tune Threshold Value beyond default setting.
                               // Otherwise, leave the default "550"
value.

PulseSensorPlayground pulseSensor;  // Creates an instance of the
PulseSensorPlayground object called "pulseSensor".

// LCD Variables
LiquidCrystal_I2C lcd(0x27, 16, 2); // Set the LCD address to 0x27 for
a 16 chars and 2-line display
```

```
void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);

  // Initialize LCD
  lcd.init();                       // Initialize the LCD
  lcd.backlight();                  // Turn on the backlight
  lcd.setCursor(0, 0);
  lcd.print("Initializing...");

  // Configure the PulseSensor object
  pulseSensor.analogInput(PulseWire);
  pulseSensor.blinkOnPulse(LED);        // Auto-magically blink
Arduino's LED with heartbeat.
  pulseSensor.setThreshold(Threshold);

  // Double-check the "pulseSensor" object was created and "began"
seeing a signal.
  if (pulseSensor.begin()) {
    Serial.println("PulseSensor initialized!");
    lcd.setCursor(0, 0);
    lcd.print("Pulse Sensor Ready");
  } else {
    Serial.println("PulseSensor initialization failed!");
    lcd.setCursor(0, 0);
    lcd.print("Sensor Error!");
  }

  delay(2000); // Display message for 2 seconds
  lcd.clear();
}

void loop() {
  if (pulseSensor.sawStartOfBeat()) {             // Constantly test to
see if "a beat happened".
    int myBPM = pulseSensor.getBeatsPerMinute(); // Calls function on
our pulseSensor object that returns BPM as an "int".
                                                 // "myBPM" holds this
BPM value now.


    // Display BPM on the LCD
    lcd.setCursor(0, 0);
    lcd.print("Heart Rate:      ");  // Clear previous data with spaces
    lcd.setCursor(0, 1);
    lcd.print("BPM: ");
    lcd.print(myBPM);
    lcd.print("   "); // Clear trailing digits
  }

  delay(20);                          // Considered best practice in a simple
sketch.
}
```

Fig. A4. Pulsemeter code