KHANH DUY NGUYEN - knguye71 - knguye71@uic.edu

TIM HUYNH - khuynh22 - khuynh22@uic.edu

CS 362

Professor Diana Diaz Herrera

12/01/2021

Prototype of Affordable Portable SpO_2 Tracking Device and Fast Reaction System for COVID-19 patients treating at home

Outline for the Project

1. Abstract Project Ideas

Due to the COVID-19 pandemic, there are some patients who decide to treat themselves at home. As a result, there is a renewed focus on developing low-cost devices that can assist individuals in tracking their symptoms and seeking quick medical assistance if they experience any emergency warning signs. In order to provide prompt attention to the patients, we decided to create a gadget combining two Arduino boards with a Wi-Fi connection and a SpO_2 device to track the current SpO_2 level and heartbeat.

Keywords: portable, inexpensive, SpO_2, COVID-19, Arduino, fast attention

2. Detailed Project Ideas

2.1. Overall Description of Project Idea

The COVID-19, commonly known as COVID-19 and the coronavirus, is an infectious illness caused by coronavirus 2, which causes severe acute respiratory syndrome (SARS-CoV-2). In December of 2019, the first case (F0) was discovered in Wuhan, China. Since then, the disease has spread globally, resulting in a pandemic that is still underway. We have observed similar events in the great majority of underdeveloped and developing countries when COVID-19 spread generated severe hospital overcrowding and a high lack of healthcare resources and professional burden in a matter of weeks.

As a consequence, many patients had to treat their symptoms of COVID-19 at home. However, Gonzalo Mena of the Harvard T.H. Chan School of Public Health in Boston, Massachusetts, and his colleagues conducted a study on death rates in several neighborhoods of Santiago, Chile, particularly among persons under the age of 80. (G. E. Mena et al. Science https://doi.org/f9b4; 2021). They found out in lower-income neighborhoods, 90 percent of COVID-19 deaths occurred outside of health-care institutions, compared to 55 percent in a more wealthy section of the city. Hence, this raises the attention to develop inexpensive equipment that could help the patients to track their symptoms and give immediate medical attention when they have any emergency warning signs.

After conducting many pieces of research on detecting the COVID-19 symptoms, our group figured that the pulse oximeter is one of the most effective tools for detecting shortness of breath. When you breathe, oxygen enters your lungs, travels through thin membranes, and enters your bloodstream, where it is taken up by hemoglobin and transported to various organs throughout the body. A pulse oximeter is a little gadget that fits over your fingertip or clips into your earlobe to detect how effectively oxygen binds to the hemoglobin via infrared light refraction. Blood oxygen levels are reported by oximeters using an oxygen saturation measurement known as peripheral capillary oxygen saturation, or SpO_2.

Hence, we - Khanh Duy Nguyen and Tim Huynh - decided to design a prototype of a portable device that can track the patient's SpO_2 using the Pulse Oximeter & Heart Rate Sensor then communicate the current status of the patient to the nearest caregivers.

2.2. Project Design

The design will focus on three crucial aspects of the device: functionality, portability, and affordability.

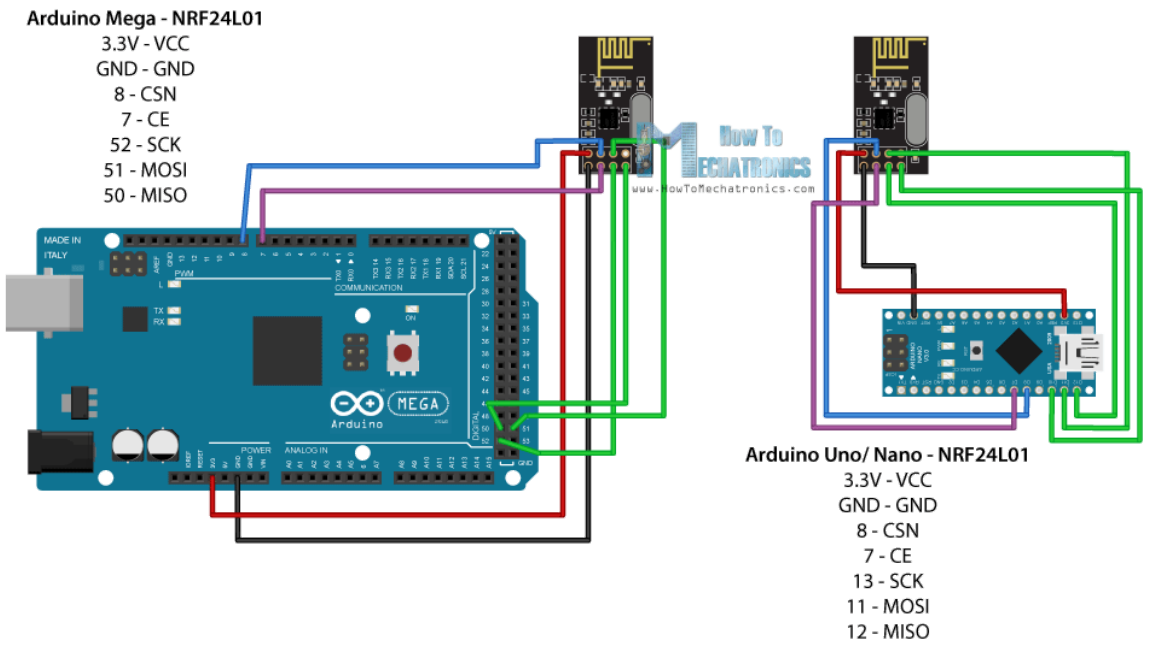
By functionality, the prototype needs to work properly and the results should give the right number of heartbeat and SpO_2 within the margin of error. By portability, the design needs to be small, lightweight but still keep the flexibility and comfortability for the user. By affordability, we aim to design an affordable device that could help the poor to easily buy it. The design consists of two key elements: The server focuses on gathering SpO_2 data and sending it to the peripheral; the peripheral focuses on determining the appropriate action to take based on the data. As a result, we chose to develop only two components for the server device, while the peripheral will be more complex, which is in charge of processing data ends and illustrating the appropriate output behavior.

The server consists of a MAX 30102-based sensor (SpO_2 and Heartbeat sensor) connected to a single Arduino board. The sensor will read the sensor's analog signal and relay all of the information to the Arduino board. The Arduino will then use the Wi-Fi module to send SpO_2 data to another Arduino board. Furthermore, the sensor will be designed to be a system that can be attached to the finger. We highly suggest using the Arduino Mini to minimize the size of the Peripheral device. However, with the prototype, we use the Arduino Uno to show the full functionality and its behavior.

One Arduino board is attached to a 16x2 LCD screen that displays the current heartbeat and SpO_2 level, while the other processor will have an S.O.S LED and a buzzer. The Arduino board will collect data from the server devices and determine what action is required and needed. The Arduino just feeds the data to the LCD screen for normal SpO_2 levels. However, if the SpO_2 level falls below 75%, we will begin to activate the buzzer and switch on the LED light. Furthermore, we will use the breadboard to organize the outputs.

2.3. Communication Method

The main method of communication between two Arduino boards is the Wi-Fi module NF24L01. We have considered using Bluetooth as our communication method before, but when testing it, the range of the Bluetooth module is not as far as the Wi-Fi module (up to 80m), and the connection is unstable. For that reason, the NF24L01 Wi-Fi module is a better option to serve as a communication method. The schematic below shows how the Wi-Fi module is installed.



The communication methods between outputs and the Arduino board are digital and analog signals. We prefer the digital output for the LED due to the consistency in voltage. Furthermore, the buzzer will use the analog signal to output the correct pitch for the sound.

2.4. The Originality of the Project

As there are many devices to measure the level of oxygen and heartbeat of the patient, we pay higher attention to the device's affordability and low cost because of the uniqueness of our concept. In particular, after careful measurement, our prototype only costs about \$70 - \$80. Furthermore, modern SpO_2 devices do not measure in real-time; instead, they only measure when the user requests it. As a result, people pay less attention to the decline in SpO_2 when the oxygen content in the blood drops at night. As a result, patients are unable to get to the hospital in a timely manner, putting them at risk of catastrophic harm. Additionally, caregivers will have more time to sleep at night and will not have to be as consistent in the care of their loved ones.

2.5. Weekly Lab Report

The weekly Lab Report will consist of two important aspects: the progress of the project and analysis of the performance of the device.

For week one, our goal was to get the output components working in the single Arduino. We simulated the input with the global variable of SpO_2 to be rising from 0% -100% then going down in 1 minute. The output gave the LCD printing out the correct SpO2 level through time, the alarm should rise when SpO_2 is lower than 75%, and the LED turned on when the alarm is ringing.

For week two, we had the same aim as the first week but focused more on the User Interface and the new complex hardware such as the NRF24L01 Wi-Fi component and SpO_2

sensor. For the Wi-Fi module, we do a small analysis of the data rate and the latency of data. The latency must be less than 100 ms to keep track of the patient's SpO_2 to be more effective.

For the third week, we focused more on the demos and the presentation. As the project is fully built, we focus on how to make it good-looking and easy to use. Also, we tried to test all the possible cases when the level of heartbeat and/or SpO_2 went wrong. So far, the prototype worked well as expected. We also completed our report, contract, and user guide for the product since this is a potential idea that might help for future development or purposes.

2.6. User Guide

The user guide consists of two parts: the server part and the peripheral part.

Server User Guide:

The 16x2 LCD screen will display the instructions on how to use the sensor. Firstly, press the yellow button to start measuring the heartbeat and the SpO_2. When the LCD displays: "Please put your finger on the sensor", put your index finger on the glass part of the sensor (the part that looks like a camera). The sensor will take about 20s to measure your level of oxygen and 40s to measure your heart rate. After measuring, the first line of the LCD will display your heart rate per minute and the second line will display your percentage of the level of oxygen. When you no longer want to use it, press the red button to stop running the sensor.

Peripheral User Guide:

When the LED lights up and the buzzer buzzes, please come to the patient and check the condition. If the level of oxygen or the heartbeat is too far from the margin of acceptance, immediately contact the nearest hospital and bring the patients there for a deeper treatment. After carefully handling everything, you can turn off the LED and the buzzer by pressing the push button.

3. Supporting Materials

3.1. Timeline of the development

The table below illustrates our specific work and time for this project. It demonstrates the starting at the first week where we started to think about the project to this final week where we completed the project.

Note that Week 1 in the project is not the first week of the semester but is the first week that we started to work on the project.

Week	Description	Date
Week 1	Brainstormed and decided on the final project idea after analyzing the pros and cons.	09/10/2021
Week 2	Analyzed all the components that were needed for the project including the inputs/outputs, the communication method, and all the other components. Researched all the other similar work to determine our original work.	09/24/2021
Week 3	Written the Project Contract, stated our pseudo code and the anticipated schematic.	10/08/2021
Week 4	Updated our Project Contract, bought the needed materials (Wi-Fi Modules, Arduino Mega Board, SpO_2 sensor, etc.) Updated the diagram and the pseudo-code.	11/05/2021
Week 5	Learned how to use the sensor and the Wi-Fi module along with other materials. Also created the slides and the voiced over the presentation of the project	11/12/2021
Week 6	Finished installing the server board and the peripheral part of the project. Analyzed what was left to do and what had not been done yet. Presented to other teams and the TAs and Professor. Completed the Individual Evaluation.	11/24/2021
Week 7	Tested the project again to make sure there are all perfect. Completed the final report and the commitment.	12/03/2021

3.2. Final List of Materials Needed

As spending our time working and building the prototype for our project, we have carefully selected the materials that we will be needed for our project:

One Arduino Mega Board and One Arduino Uno Board
One Interfacing MAX30102 Pulse Oximeter Sensor with Arduino
One RGB 16x2 LCD
Two Wi-Fi Module NRF24L01 (one transmitter and one receiver)
One Yellow Button to start running the sensor and One Red Button to stop running the sensor
One Buzzer
LEDs
Three Pushbuttons
Two Breadboards
Resistors, Potentiometer & Jumper wires

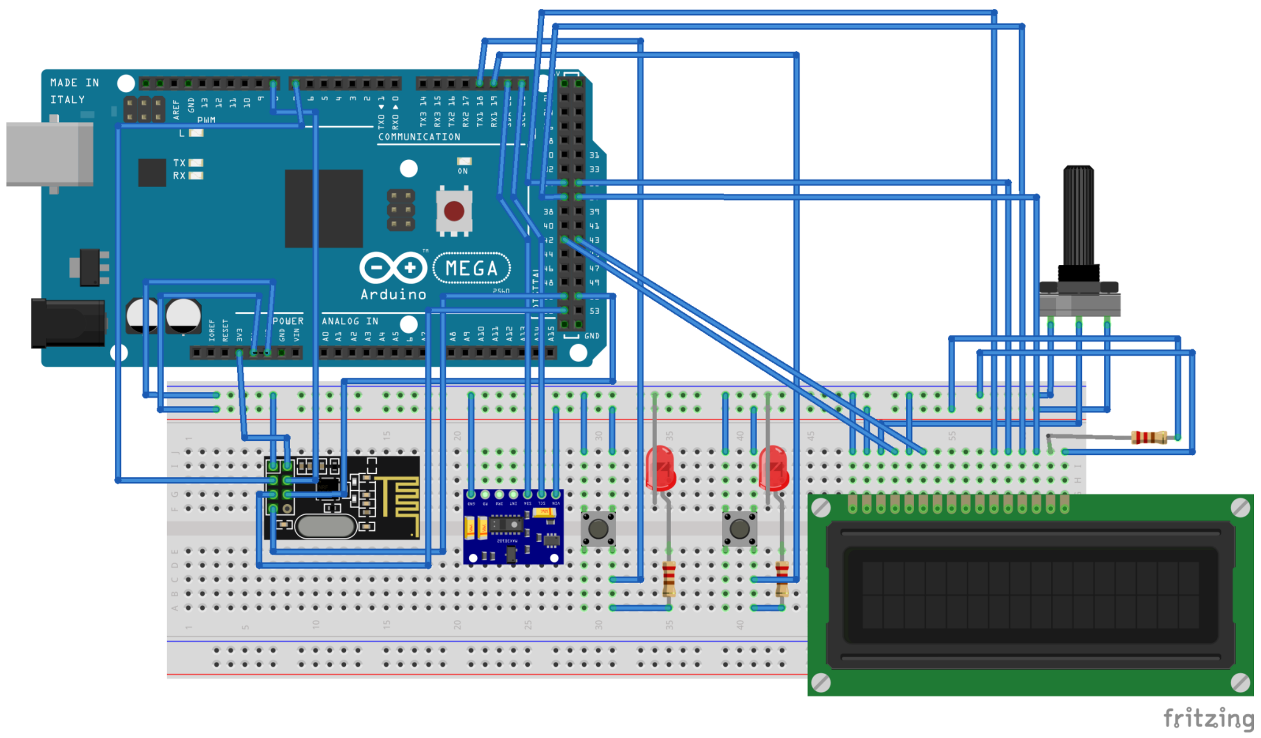
Note that for all the materials that are in the list above, we have calculated to have enough for usage and we would not buy more than needed so that we will not waste the materials.

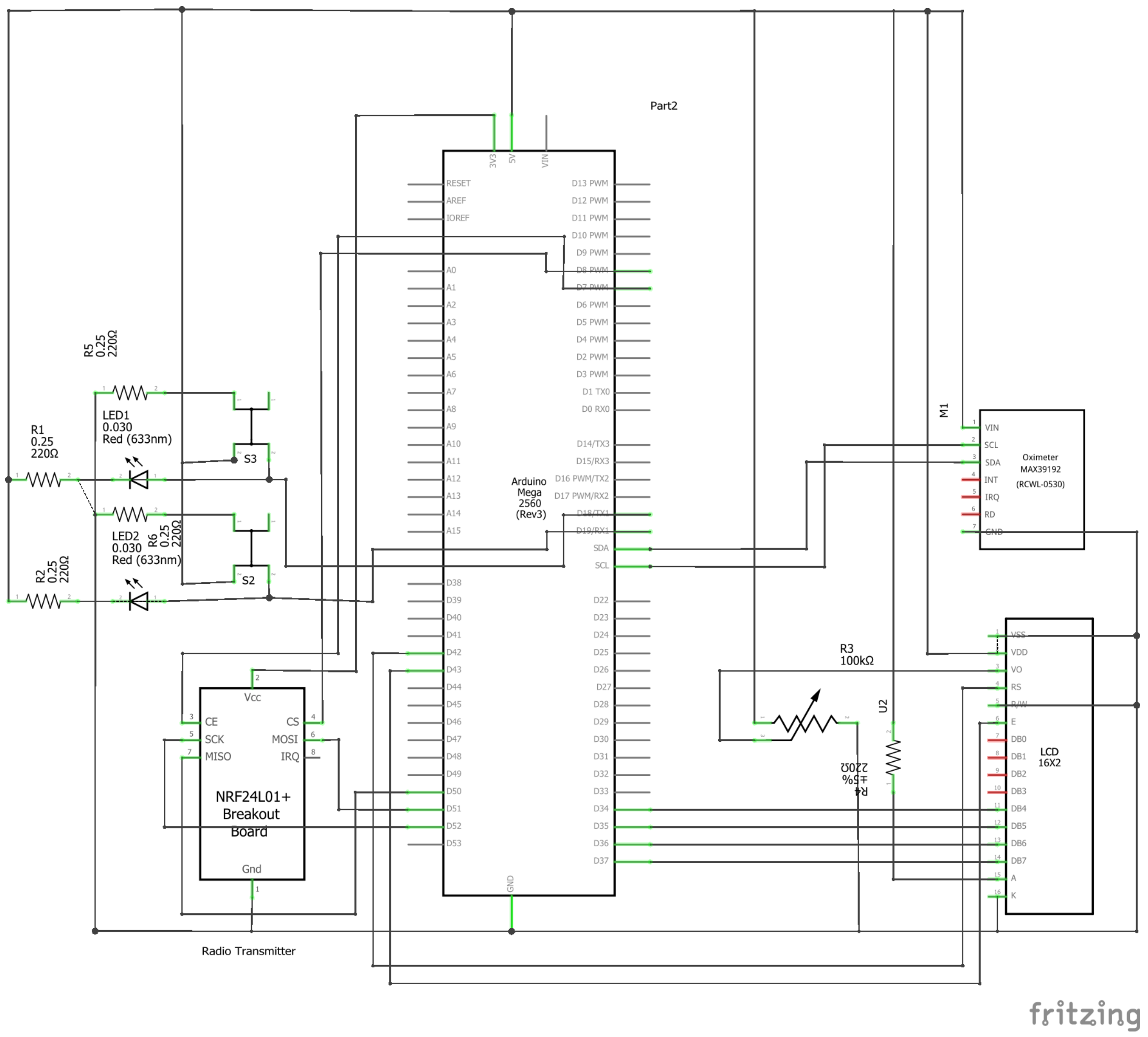
3.3. Hardware Used Diagram

In this project, as was mentioned, we used two microprocessors and two boards, one for the server and one for the peripheral. The diagram below shows the sketch of our project in terms of hardware for both of the boards.

The Server:

For the server, after many considerations, our group decided to use the Arduino MEGA processor. The MEGA board provides more pins (both analog and digital), which is necessary since we are using a lot of hardware pieces for the server. Also, for testing purposes, we need to use a MEGA board for the server and the UNO board for the peripheral since when we try to upload the code to both of the boards at the same time, the different types of the board help us to upload the right code to the right board. From the diagram, the LCD will display the instruction for the patient to know how to set it up and measure their heartbeat and SpO_2. When measuring, the screen will display the value of heartbeat and SpO_2. To start measuring these values, the patient can press the yellow button, and to stop measuring, the patient can press the red button. Some LEDs will turn on whenever these buttons are pressed. We put the Interfacing MAX30102 Pulse Oximeter Sensor in the middle of the board so that the patient can easily use it without colliding with other wires. Last but not least, we can see from the diagram that there is an NRF24L01 Wi-Fi Module that serves as a transmitter to send all the data to the receiver.





The Peripheral:

The schematic of the receiver is fairly simple compared to the server since all it has to do is to notify the caregivers when the data of the heartbeat or the level of oxygen of the patient exceed the margin of acceptance. According to the diagram, the receiver contains one LED and one Buzzer. These two output devices are there to make sure that the caregivers will be notified visually and audibly when things went wrong. The pushbutton is designed to turn off the output devices after the caregivers have already been notified and handled the situation. And again, there will be another NRF24L01 Wi-Fi module here in the receiver to receive the data sent by the transmitter. We decided to use the UNO processor for the receiver as we are only using one LED and one Buzzer. Also, the UNO processor is much cheaper than the MEGA processor, which met one of our important criteria of the project: affordability.

3.4. Final Code Sketches for the project

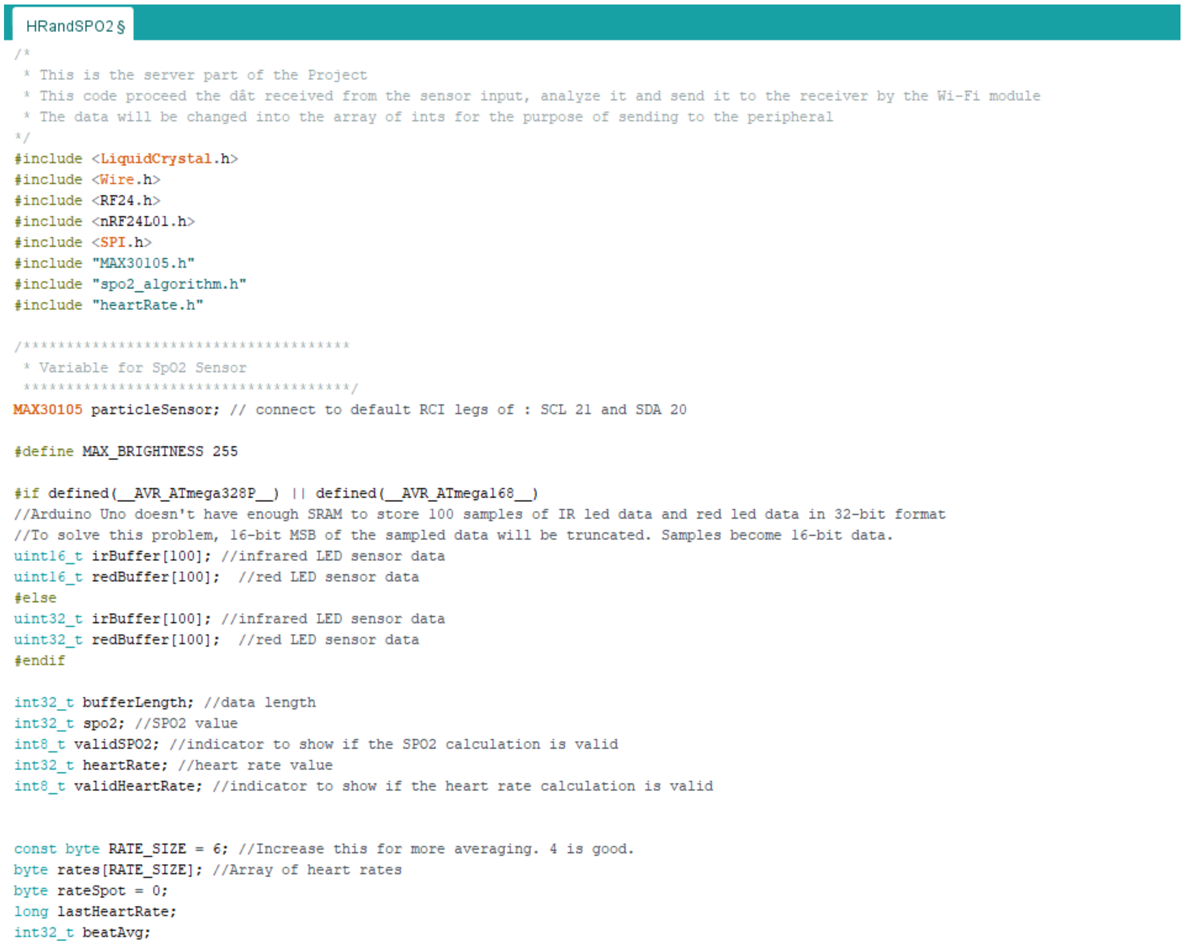
There are two files of .ino code, one for the server and one for the peripheral code.

As the two files are very long and complicated, we decided to put the link to them here:

https://github.com/duynguyen2001/SpO2Project

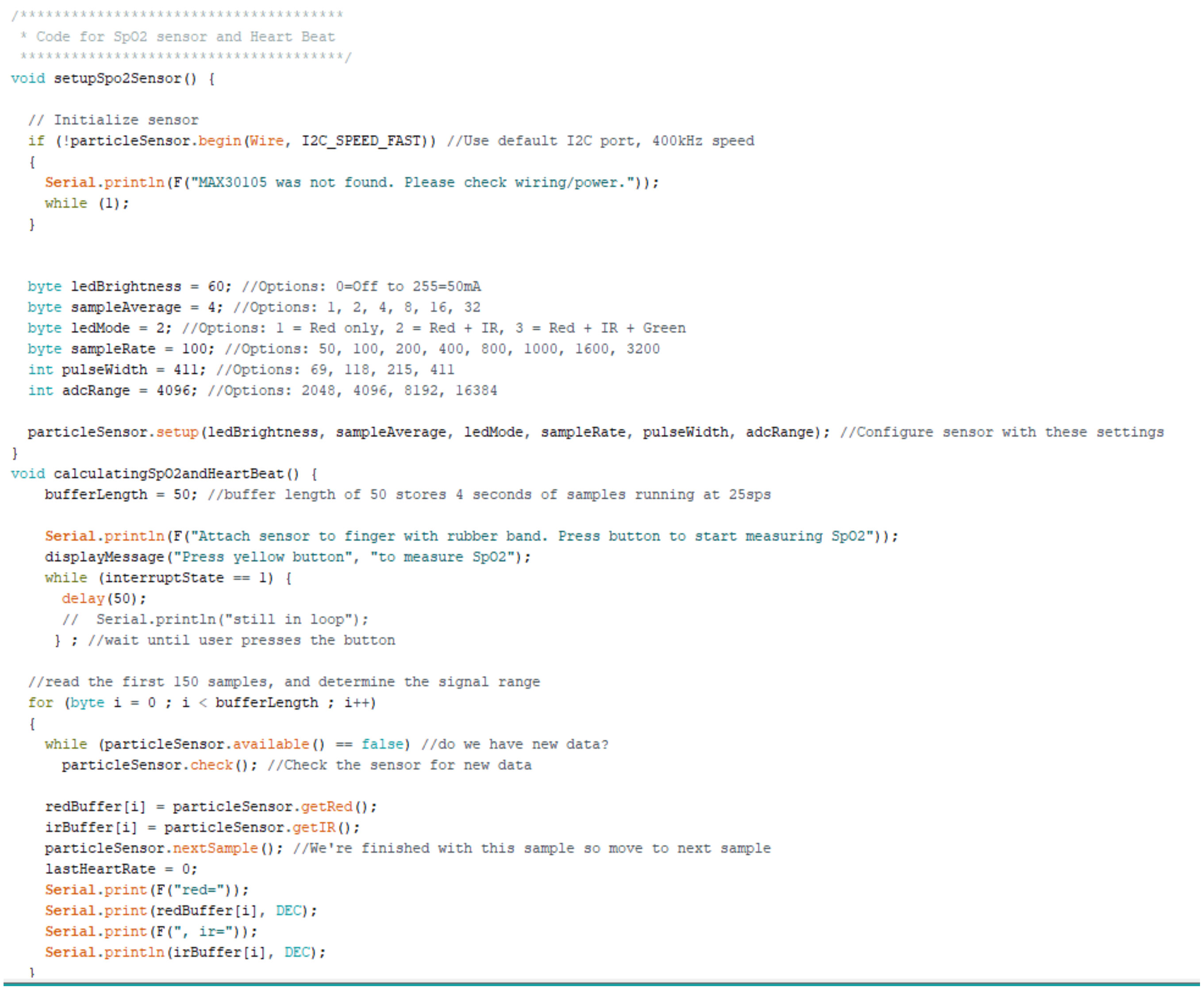
However, the following pictures are the captures of the code.

The server code:

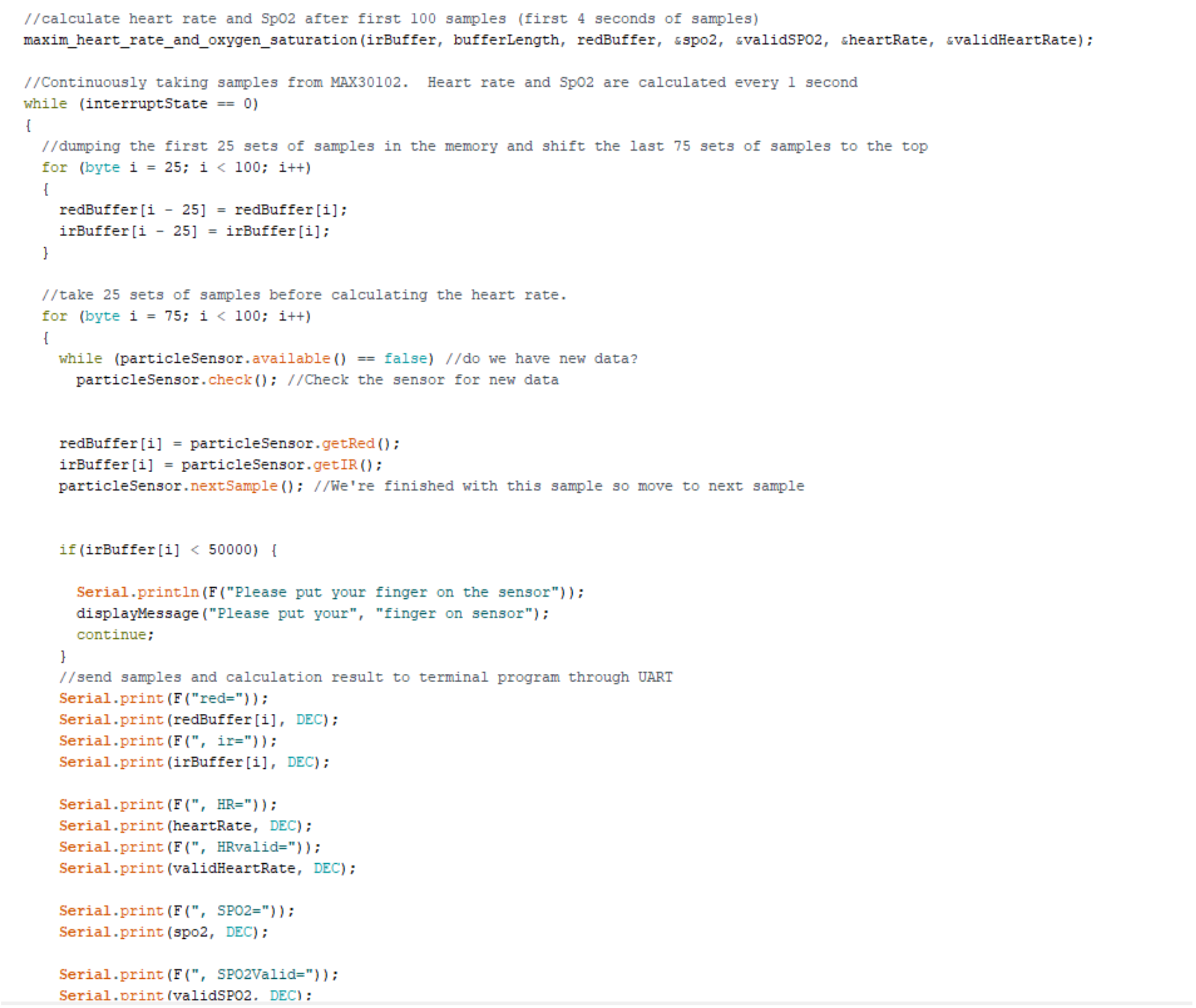


```
/*
 * This is the server part of the Project
 * This code proceed the dât received from the sensor input, analyze it and send it to the receiver by the Wi-Fi module
 * The data will be changed into the array of ints for the purpose of sending to the peripheral
*/
#include <LiquidCrystal.h>
#include <Wire.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <SPI.h>
#include "MAX30105.h"
#include "spo2_algorithm.h"
#include "heartRate.h"

/**************************************
 * Variable for Sp02 Sensor
 **************************************/
MAX30105 particleSensor; // connect to default RCI legs of : SCL 21 and SDA 20

#define MAX_BRIGHTNESS 255

#if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
//Arduino Uno doesn't have enough SRAM to store 100 samples of IR led data and red led data in 32-bit format
//To solve this problem, 16-bit MSB of the sampled data will be truncated. Samples become 16-bit data.
uint16_t irBuffer[100]; //infrared LED sensor data
uint16_t redBuffer[100];  //red LED sensor data
#else
uint32_t irBuffer[100]; //infrared LED sensor data
uint32_t redBuffer[100];  //red LED sensor data
#endif

int32_t bufferLength; //data length
int32_t spo2; //SPO2 value
int8_t validSPO2; //indicator to show if the SPO2 calculation is valid
int32_t heartRate; //heart rate value
int8_t validHeartRate; //indicator to show if the heart rate calculation is valid


const byte RATE_SIZE = 6; //Increase this for more averaging. 4 is good.
byte rates[RATE_SIZE]; //Array of heart rates
byte rateSpot = 0;
long lastHeartRate;
int32_t beatAvg;
```

```
void displayData(){
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("BPM=");
  lcd.print(msg[0]);
  lcd.setCursor(0, 1);
  lcd.print("SPO2= ");
  lcd.print(msg[1]);
}
void displayMessage(String s1, String s2){
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(s1);
  lcd.setCursor(0, 1);
  lcd.print(s2);
  lcd.print(msg[1]);

}
/***************************************
 * Code for data transmitting by Wifi
 **************************************/
void setupWifi() {
  // code to set up and process the transmitter part of the wifi module
  radio.begin();
  radio.setAutoAck(1);
  radio.setRetries(15,15);
  radio.setDataRate(RF24_2MBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setChannel(10);
  radio.openWritingPipe(address);
  radio.stopListening();
}
void transmitData() {
  if(beatAvg >= 50)
    msg[0]= (int) beatAvg;
  if(validSPO2 == 1)
    msg[1]= (int) spo2;
  radio.write(&msg,sizeof(msg));
}




/***************************************
 * Variable for interrupt
 **************************************/
bool interruptState = true;
int interruptLegON = 19;
int interruptLegOFF = 18;


/***************************************
 * Variable for displaying data for display 16x2
 **************************************/

// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const int rs = 42, en = 43, d4 = 34, d5 = 35, d6 = 36, d7 = 37;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

/***************************************
 * Main start and loop
 **************************************/
void setup()
{
   setupDisplay();
   setupInterrupt();
   setupWifi();
   Serial.begin(9600); // initialize serial communication at 9600 bits per second:
   setupSpo2Sensor();
}

void loop()
{
  calculatingSpO2andHeartBeat();
}

/***************************************
 * Code for displaying data to 16x2 display
 **************************************/
void setupDisplay() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
}
```



```
/****************************************
 * Code for SpO2 sensor and Heart Beat
 ****************************************/
void setupSpo2Sensor() {

  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) //Use default I2C port, 400kHz speed
  {
    Serial.println(F("MAX30105 was not found. Please check wiring/power."));
    while (1);
  }


  byte ledBrightness = 60; //Options: 0=Off to 255=50mA
  byte sampleAverage = 4; //Options: 1, 2, 4, 8, 16, 32
  byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green
  byte sampleRate = 100; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200
  int pulseWidth = 411; //Options: 69, 118, 215, 411
  int adcRange = 4096; //Options: 2048, 4096, 8192, 16384

  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adcRange); //Configure sensor with these settings
}
void calculatingSpO2andHeartBeat() {
    bufferLength = 50; //buffer length of 50 stores 4 seconds of samples running at 25sps

    Serial.println(F("Attach sensor to finger with rubber band. Press button to start measuring SpO2"));
    displayMessage("Press yellow button", "to measure SpO2");
    while (interruptState == 1) {
      delay(50);
      //  Serial.println("still in loop");
     } ; //wait until user presses the button

  //read the first 150 samples, and determine the signal range
  for (byte i = 0 ; i < bufferLength ; i++)
  {
    while (particleSensor.available() == false) //do we have new data?
      particleSensor.check(); //Check the sensor for new data

    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample(); //We're finished with this sample so move to next sample
    lastHeartRate = 0;
    Serial.print(F("red="));
    Serial.print(redBuffer[i], DEC);
    Serial.print(F(", ir="));
    Serial.println(irBuffer[i], DEC);
  }
```

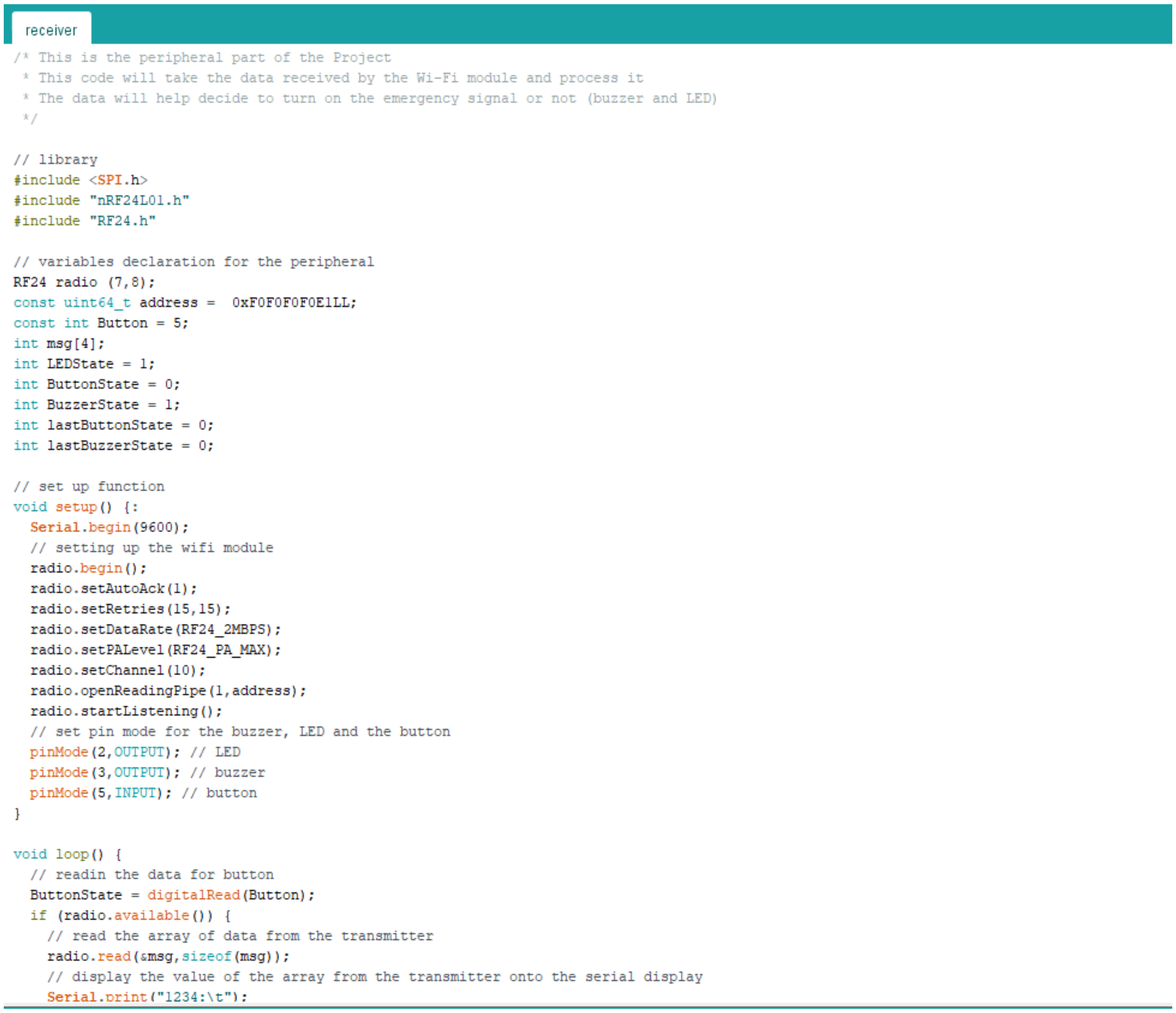


```
//calculate heart rate and SpO2 after first 100 samples (first 4 seconds of samples)
maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);

//Continuously taking samples from MAX30102.  Heart rate and SpO2 are calculated every 1 second
while (interruptState == 0)
{
  //dumping the first 25 sets of samples in the memory and shift the last 75 sets of samples to the top
  for (byte i = 25; i < 100; i++)
  {
    redBuffer[i - 25] = redBuffer[i];
    irBuffer[i - 25] = irBuffer[i];
  }

  //take 25 sets of samples before calculating the heart rate.
  for (byte i = 75; i < 100; i++)
  {
    while (particleSensor.available() == false) //do we have new data?
      particleSensor.check(); //Check the sensor for new data


    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample(); //We're finished with this sample so move to next sample


    if(irBuffer[i] < 50000) {

      Serial.println(F("Please put your finger on the sensor"));
      displayMessage("Please put your", "finger on sensor");
      continue;
    }
    //send samples and calculation result to terminal program through UART
    Serial.print(F("red="));
    Serial.print(redBuffer[i], DEC);
    Serial.print(F(", ir="));
    Serial.print(irBuffer[i], DEC);

    Serial.print(F(", HR="));
    Serial.print(heartRate, DEC);
    Serial.print(F(", HRvalid="));
    Serial.print(validHeartRate, DEC);

    Serial.print(F(", SPO2="));
    Serial.print(spo2, DEC);

    Serial.print(F(", SPO2Valid="));
    Serial.print(validSPO2, DEC);
```

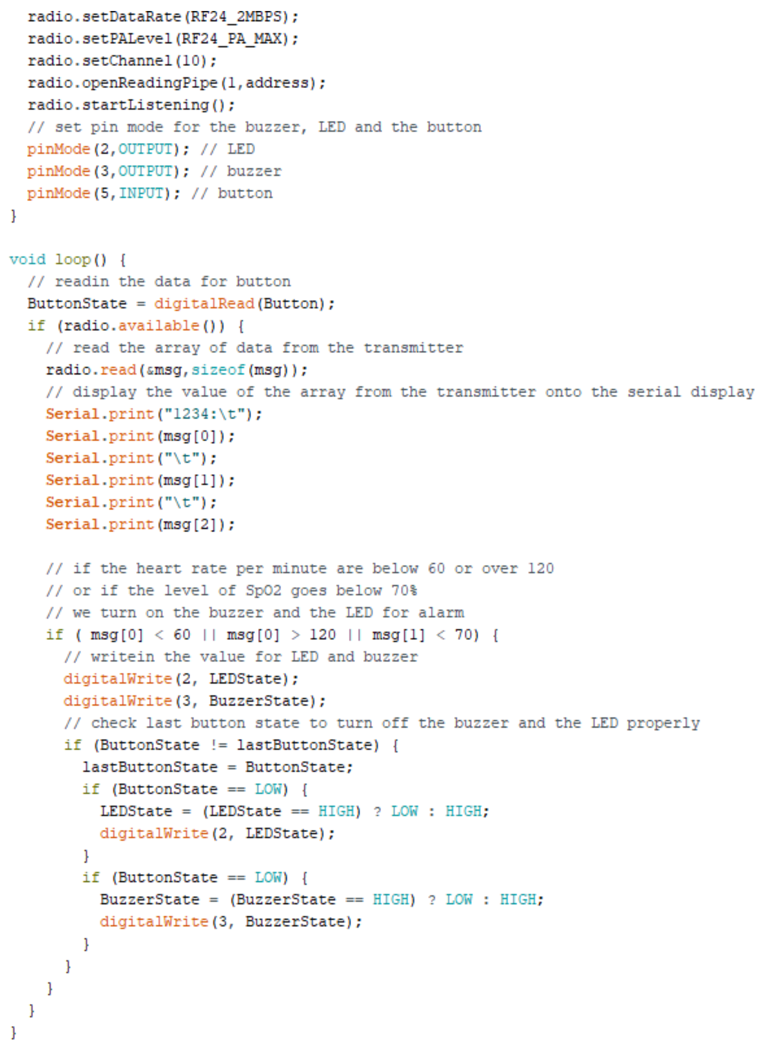
```
      Serial.print(F(", HRvalid="));
      Serial.print(validHeartRate, DEC);

      Serial.print(F(", SPO2="));
      Serial.print(spo2, DEC);

      Serial.print(F(", SPO2Valid="));
      Serial.print(validSPO2, DEC);

      Serial.print(", Avg BPM=");
      Serial.print(beatAvg);


      Serial.print(", last BPM=");
      Serial.println(lastHeartRate);

      calculateAvgHeartBeat(heartRate, validHeartRate);
      transmitData();
      displayData();
    }

    //After gathering 25 new samples recalculate HR and SP02
    maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);
  }

}

void calculateAvgHeartBeat(uint32_t heartRate, int8_t validHeartRate) {

    Serial.print(F("Go inside"));
    if (validHeartRate <= 0 || heartRate == lastHeartRate) {
      return;
    }

    if (heartRate < 255 && heartRate > 20)
    {
      rates[rateSpot++] = (byte)heartRate; //Store this reading in the array
      rateSpot %= RATE_SIZE; //Wrap variable

      lastHeartRate = heartRate;
      //Take average of readings
      beatAvg = 0;
      for (byte x = 0 ; x < RATE_SIZE ; x++)
        beatAvg += rates[x];
      beatAvg /= RATE_SIZE * 1.9;
    }
}
```

```
void calculateAvgHeartBeat(uint32_t heartRate, int8_t validHeartRate) {

    Serial.print(F("Go inside"));
    if (validHeartRate <= 0 || heartRate == lastHeartRate) {
      return;
    }

    if (heartRate < 255 && heartRate > 20)
    {
      rates[rateSpot++] = (byte)heartRate; //Store this reading in the array
      rateSpot %= RATE_SIZE; //Wrap variable

      lastHeartRate = heartRate;
      //Take average of readings
      beatAvg = 0;
      for (byte x = 0 ; x < RATE_SIZE ; x++)
        beatAvg += rates[x];
      beatAvg /= RATE_SIZE * 1.9;
    }
}

/**************************************
 * Code for interrupt component
 **************************************/
void setupInterrupt(){
  pinMode(interruptLegON, INPUT_PULLUP);
  pinMode(interruptLegOFF, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptLegON), interruptHandler1, CHANGE);
  attachInterrupt(digitalPinToInterrupt(interruptLegOFF), interruptHandler2, CHANGE);
}
// Change interrupt State to false
void interruptHandler1() {
  interruptState = false;
  Serial.println("Interrupt set");
  Serial.println(interruptState);
}
// Change interrupt State to true
void interruptHandler2() {
  interruptState = true;
  Serial.println("Interrupt set");
  Serial.println(interruptState);
  displayMessage("Stop measuring", "Press yellow to start");
  delay(1000);
}
```

The Peripheral Code:



```
/* This is the peripheral part of the Project
 * This code will take the data received by the Wi-Fi module and process it
 * The data will help decide to turn on the emergency signal or not (buzzer and LED)
 */

// library
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

// variables declaration for the peripheral
RF24 radio (7,8);
const uint64_t address =  0xF0F0F0F0E1LL;
const int Button = 5;
int msg[4];
int LEDState = 1;
int ButtonState = 0;
int BuzzerState = 1;
int lastButtonState = 0;
int lastBuzzerState = 0;

// set up function
void setup() {:
  Serial.begin(9600);
  // setting up the wifi module
  radio.begin();
  radio.setAutoAck(1);
  radio.setRetries(15,15);
  radio.setDataRate(RF24_2MBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setChannel(10);
  radio.openReadingPipe(1,address);
  radio.startListening();
  // set pin mode for the buzzer, LED and the button
  pinMode(2,OUTPUT); // LED
  pinMode(3,OUTPUT); // buzzer
  pinMode(5,INPUT); // button
}

void loop() {
  // readin the data for button
  ButtonState = digitalRead(Button);
  if (radio.available()) {
    // read the array of data from the transmitter
    radio.read(&msg,sizeof(msg));
    // display the value of the array from the transmitter onto the serial display
    Serial.print("1234:\t");
```



```
  radio.setDataRate(RF24_2MBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setChannel(10);
  radio.openReadingPipe(1,address);
  radio.startListening();
  // set pin mode for the buzzer, LED and the button
  pinMode(2,OUTPUT); // LED
  pinMode(3,OUTPUT); // buzzer
  pinMode(5,INPUT); // button
}

void loop() {
  // readin the data for button
  ButtonState = digitalRead(Button);
  if (radio.available()) {
    // read the array of data from the transmitter
    radio.read(&msg,sizeof(msg));
    // display the value of the array from the transmitter onto the serial display
    Serial.print("1234:\t");
    Serial.print(msg[0]);
    Serial.print("\t");
    Serial.print(msg[1]);
    Serial.print("\t");
    Serial.print(msg[2]);

    // if the heart rate per minute are below 60 or over 120
    // or if the level of SpO2 goes below 70%
    // we turn on the buzzer and the LED for alarm
    if ( msg[0] < 60 || msg[0] > 120 || msg[1] < 70) {
      // writein the value for LED and buzzer
      digitalWrite(2, LEDState);
      digitalWrite(3, BuzzerState);
      // check last button state to turn off the buzzer and the LED properly
      if (ButtonState != lastButtonState) {
        lastButtonState = ButtonState;
        if (ButtonState == LOW) {
          LEDState = (LEDState == HIGH) ? LOW : HIGH;
          digitalWrite(2, LEDState);
        }
        if (ButtonState == LOW) {
          BuzzerState = (BuzzerState == HIGH) ? LOW : HIGH;
          digitalWrite(3, BuzzerState);
        }
      }
    }
  }
}
```

4. References

“Arduino Bluetooth Basic Tutorial.” *Arduino Project Hub*, https://create.arduino.cc/projecthub/mayooghgirish/arduino-bluetooth-basic-tutorial-d8b737. Accessed 30 Nov. 2021.

“Coronavirus Disease 2019 (COVID-19) – Symptoms.” *Centers for Disease Control and Prevention*, 22 Feb. 2021, www.cdc.gov/coronavirus/2019-ncov/symptoms-testing/symptoms.html. Accessed 30 Nov. 2021.

Covid-19: How to Monitor Your Oxygen Level. https://www1.nyc.gov/assets/doh/downloads/pdf/covid/providers/covid-19-monitor-oxygen-patient-handout.pdf. Accessed 30 Nov. 2021.

Fahad, Engr. “Max30100 Pulse Oximeter Arduino Code, Circuit, and Programming.” *Electronic Clinic*, 14 Aug. 2021, www.electroniclinic.com/max3. Accessed 30 Nov. 2021.

Last Minute Engineers. “In-Depth: How NRF24L01 Wireless Module Works & Interface with Arduino.” *Last Minute Engineers*, Last Minute Engineers, 18 Dec. 2020, https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/. Accessed 30 Nov. 2021.

“Measure spo2, Heart Rate, and BPT Using Arduino.” *Arduino Project Hub*, 6 Jan. 2021, https://create.arduino.cc/projecthub/iasonas-christoulakis/measure-spo2-heart-rate-and-bpt-using-arduino-68724d?ref=part&ref_id=10308&offset=9. Accessed 30 Nov. 2021.

Mena, Gonzalo E., et al. “Socioeconomic Status Determines COVID-19 Incidence and Related Mortality in Santiago, Chile.” *Science*, vol. 372, no. 6545, 2021, p. eabg5298. *Crossref*, doi:10.1126/science.abg5298.0100-pulse-oximeter-Arduino-code-circuit-and-programming. Accessed 30 Nov. 2021.