Received: August 2024, Accepted: November 2024, Published: December 2024
Digital Object Identifier: https://doi.org/10.34302/CJEE/PNLK5679

PULSE, BLOOD OXYGEN AND BODY TEMPERATURE MEASUREMENT SYSTEM WITH INTERNET DATA MONITORING. CONSTRUCTION, INSTALLATION, CONFIGURATION SYSTEM PROGRAMMING

Ioana Laura **ALEXANDRESCU**, Ioan Marius **ALEXANDRESCU**
Technical University of Cluj-Napoca, Romania
ioana.laura.alexandrescu@gmail.com, Ioan.alexandrescu@imtech.utcluj.ro

Keywords: **pulse, blood oxygen level, microcontroller, body temperature, pulse, programming, system installation**

Abstract: *The vision of creating this device was to ease the work of the people who would be responsible for the physical recording of the temperature of the people in an enclosure or a specific space, such as a medical office, for example, the pulse and the amount of oxygen in the blood by replacing the human resource with a device that takes all three with the help of sensors and not only that, it sends the data taken by them to some tables in the related database, after which it creates statistical graphs with them. Pulse, blood oxygen and body temperature measurement system with internet data monitoring has in the component sensors MLX90614 for remote body temperature recording, sensor that was connected to ESP32 microcontroller, using I2C communication protocol and MAX30100, which we used to measure pulse and blood oxygen level (SpO2), being connected to another microcontroller, Arduino Uno, also via I2C. What has been measured is displayed on an LCD2004, and the data is transmitted wireless to the local server, in the database created in MySQL.*

1. INTRODUCTION

The solution to help restore day-to-day peace or confidence that we are safe is more thorough monitoring of the health of vulnerable people and more, with constant access to recorded data via the Internet. To put the solution into practice, we created a system to monitor

vital functions, such as pulse and blood oxygen level (SpO_2), but also body temperature, with the help of two sensors and other components presented in the paper.

The components needed to make this Pulse, blood oxygen and body temperature measurement system with internet data monitoring are: ESP32, Arduino Uno, Pulse and SpO_2 Sensor, Temperature Sensor, LCD Display 2004, Voltage Level Regulator, Arduino IDE.

2. COMPONENTS USED, CONSTRUCTION AND OPERATION

We used two microcontrollers (ESP32 and Arduino Uno), 3 sensors (MLX90614 for temperature, GY MAX30100 and RCWL 0530 for pulse and oximeter), an LCD display, an I2C interface module and a logic level converter, as well as software, through which the programming of the components, the connections both serial and wireless, the database and tables in it were made.

In the block diagram below, you can see the block diagram of the system (*Fig.1*).

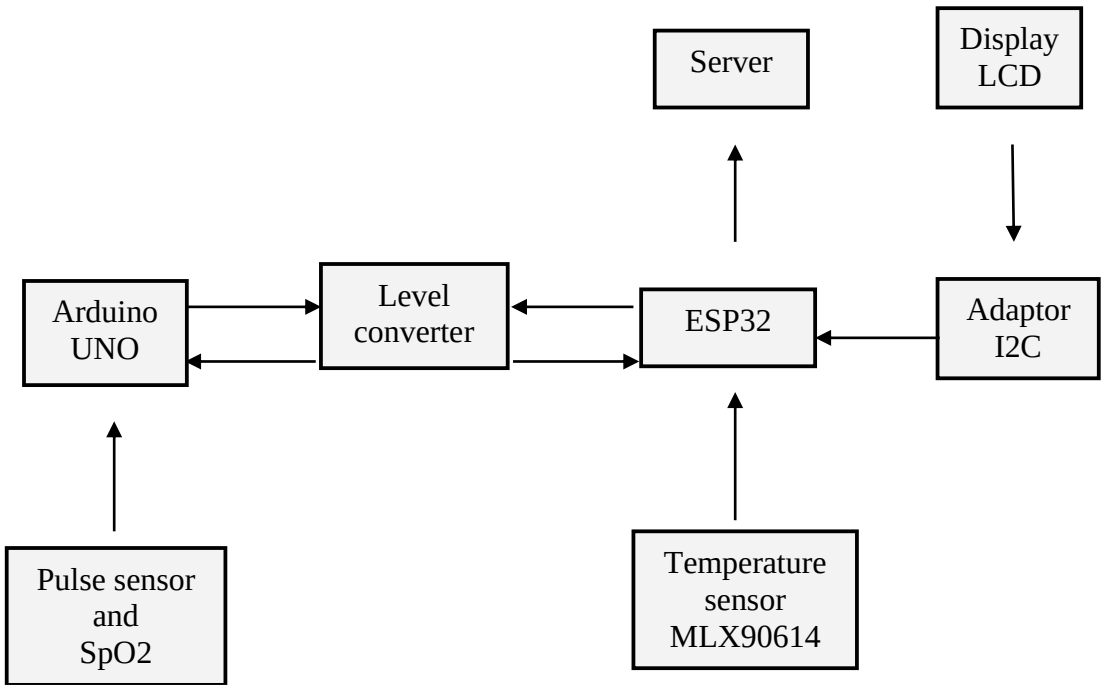


Fig.1 The block diagram

2.1. The ESP32 development board

The ESP32 development board is a microcontroller, system-on-a-chip (SoC) manufactured by Espressif Systems, with an Xtensa LX6 architecture (*Fig. 2*). It is increasingly used in various systems and its popularity has increased in recent years due to its low price, generous capacity, small size, integrating Wi-Fi and Bluetooth (higher versions) wireless

communication interfaces. I would call this microcontroller the "brain" of the project because it manages the data received by the sensors, as well as sending them to the database tables, also controlled by the ESP32 board.

2.2. Arduino Uno development board

The Arduino Uno is a microcontroller that uses the ATmega328P microchip and was developed by AVR Arduino.cc. The year of appearance was 2010. The board is equipped with sets of input/output (I/O) pins, digital and analog channels, which can be interfaced with various expansion boards (shields) and other circuits (*Fig.3*).

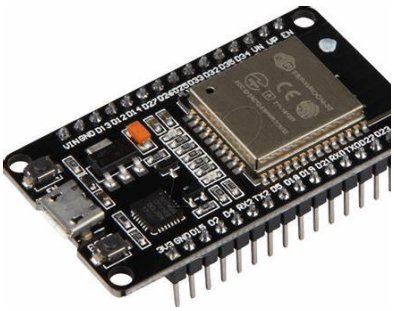


Fig.2 ESP2 microcontroller

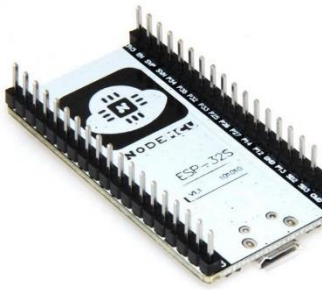


Fig. 3 Arduino Uno

2.3. The MLX90614 sensor

The MLX90614 is an infrared thermometer module used to measure temperature without direct contact with the skin. The temperature resolution is approximately 0.02°C (Fig.4). It is configured with 10-bit PWM, which will transmit a temperature rate continuously in the range of -20 - +120°C with increased resolution. The angle from which the measurement perspective is most accurate is 90° at a distance of 1cm.

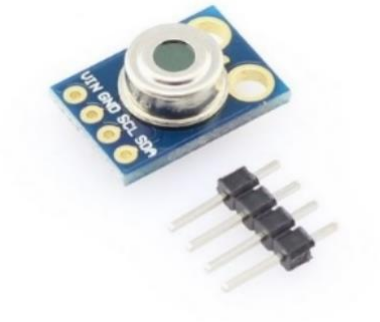
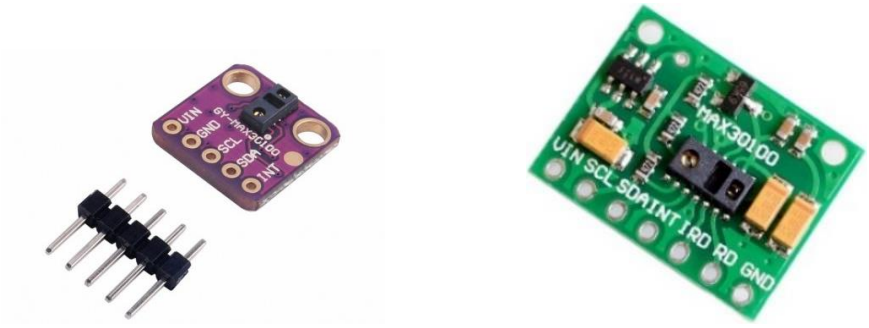


Fig. 4 MLX90614 Infrared Temperature Sensor

2.4. TheMAX30100 sensor

The MAX30100 optical sensor is intended to monitor heart rate and record blood oxygen (SpO2). Its composition includes: the photodetector, LEDs and optical elements. In order not to be affected by ambient light, the sensor features low-noise electronics (*Fig. 5*).



(a) *(b)*

Fig. 5 MAX30100: a-GY MAX30100;b-RCWL 0530

2.5. Display LCD 20-04

This alphanumeric LCD display is used to display symbols, letters and numbers in a total of 80 characters, which can be divided into 4 lines (20 characters/line) (*Fig.6*). It features an HD44780 chip and has a working voltage of 5V. The command can be done in parallel on 4 or 8 bits or through I2C communication, for which I also used an I2C adapter module (for 16x2 or 20x4 LCDs). Also, the display can be placed in less lit areas, the screen having an adjustable brightness by means of a potentiometer also present on the I2C adapter.

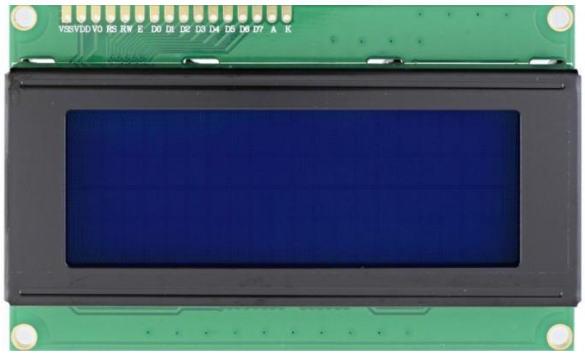


Fig. 6 LCD display

2.6. I2C adapter for LCD

We used this I2C adapter module to reduce the number of pins used by the 20x4 (originally parallel) LCD display on the ESP32, the development board used for the project.

The module attaches directly to the screen, and after that, the only wires we need are SCL and SDA – the clock and data signals, specific to I2C communication. The adapter also features a potentiometer that can be used to adjust the backlight intensity and contrast on the LCD screen (*Fig.7*).

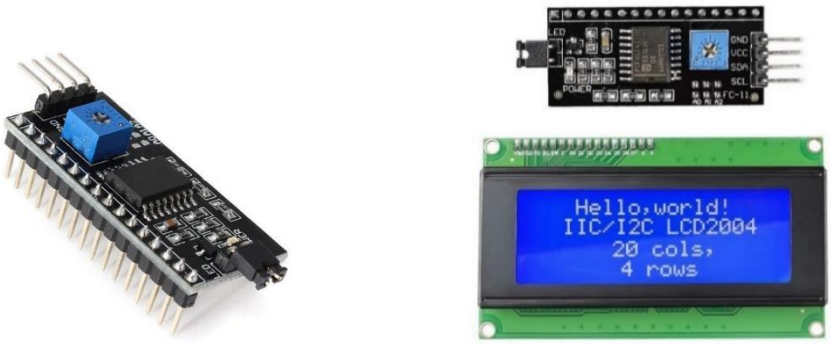


Fig. 7 I2C adapter module

2.7. Logic level converter (level shifter)

The logic level translator module connects components or devices that use different voltages, such as 1.8V, 2.5V, 3.3V or 5V, adjusting the voltage and bringing it to the same level so that they can work together. If we try to make a system with two or more devices that communicate on other voltage levels, such as between development boards, sensors, microcontrollers, other modules, for example Wi-Fi or Bluetooth, this level shifter solves the voltage differences that arise (*Fig.8*).

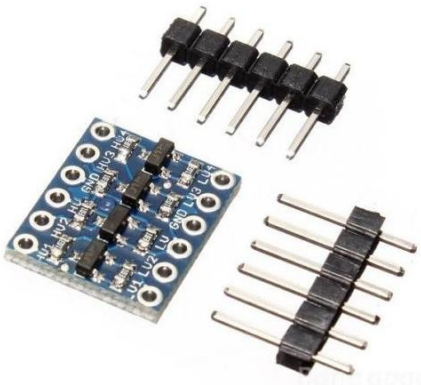


Fig. 8 Logic level converter

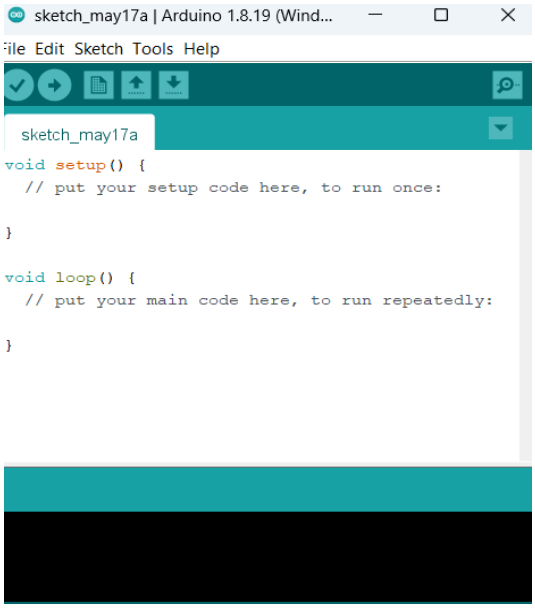


Fig. 9 New Arduino Sketch

2.8. Arduino IDE

Arduino Integrated Development Environment - or Arduino Software (IDE) - is the environment in which the programming of the components and the communication between them was carried out. Uploading programs to the Arduino and ESP32 boards was done via appropriate USB cables. Arduino uses a variant of the C++ programming language. Other methods and special functions from other files have been added to it to make the program easier to use for more users.

2.9. XAMPP, MySQL and phpMyAdmin

XAMPP is a free package and an easily accessible and modifiable web server platform developed by Apache Friends, which mainly contains the Apache HTTP server, the MariaDB database, and readers for script files written in the PHP and Perl programming languages. To run any PHP program, I needed Apache or MYSQL databases, both of which are supported by XAMPP. MySQL is a database management system. I also needed the free phpMyAdmin software tool, also described in the PHP language, intended to deal with the administration of the database created in MySQL online.

2.10 Measurement of blood oxygen level and pulse measurement

When a person touches a pulse oximeter, light from the device passes through the blood in the fingers. The amount of oxygen is calculated analogically according to changes in light absorption from both oxygenated, inspired blood and deoxygenated, expired blood (*Fig.10*). The MAX30100 sensor consists of two LEDs (red and IR) and a photodiode. Both LEDs are used to measure SpO_2. They emit light at different wavelengths, ~640nm for the red LED and ~940nm for the IR LED. At these wavelengths, oxygenated and deoxygenated hemoglobin have very different absorption properties. Oxygenated hemoglobin absorbs more infrared light and reflects red light, while deoxygenated hemoglobin absorbs more red light and reflects infrared light. The reflected light is measured by the photodetector. The MAX30100 sensor reads these different absorption levels to find the blood oxygen concentration (SPO_2). The ratio of IR to red light received by the photodetector gives us the oxygen concentration in the blood. Only the IR LED is needed to measure the heart rate. The heart rate is the ratio of time between two consecutive beats. The altered capillary tissue volume affects the sensor's infrared light, which transmits particles after each heartbeat. In other words, when a finger is placed in front of this sensor, the reflection of the infrared light is changed according to the volume change of the blood inside the capillary vessels.

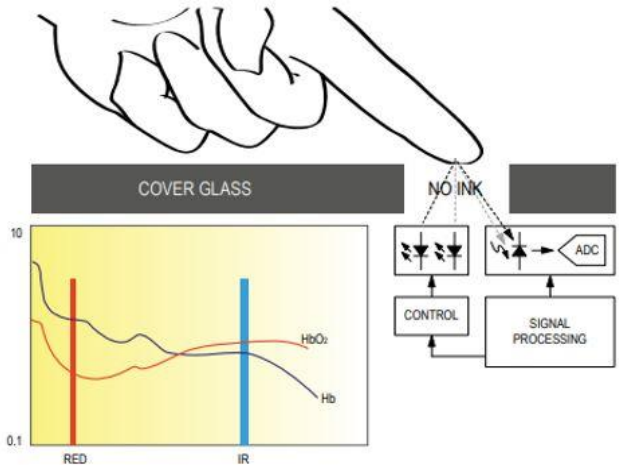


Fig.10 SpO_2 measurement

3. INSTALLATION OF THE SYSTEM

3.1. Installing ESP board package in Arduino IDE

To start testing the functionality of the components one by one, and then them programming, we first needed to be able to choose the ESP32 microcontroller from the top left menu bar of the Arduino IDE interface, from the "Tools" - "Board" button.

Step 1: File> Preferences

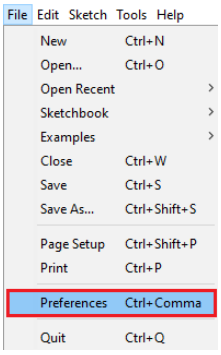


Fig. 11 Installing ESP32 in Arduino IDE step 1

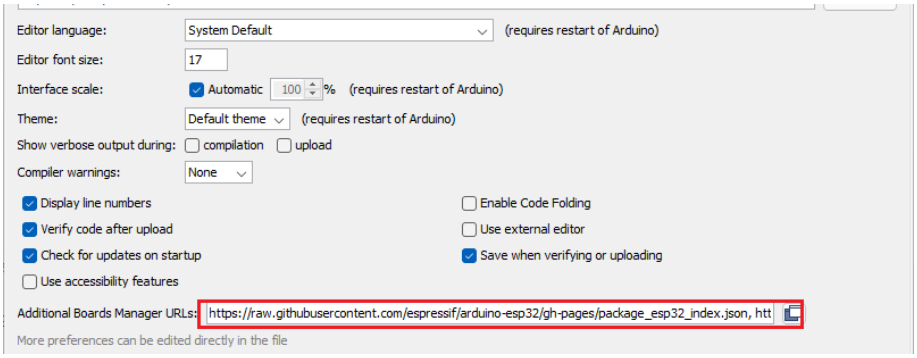


Fig. 12 Installing ESP32 in Arduino IDE step 2

Step 2: We inserted the ESP package link from github in the field shown in *Fig. 12*

Step 3: Tools> Board> Boards Manager

Step 4: We typed in the corresponding field "ESP32", and after the search engine found the package "esp32 by Espressif Systems", we pressed "Install", as can also be observed in *Fig. 14*

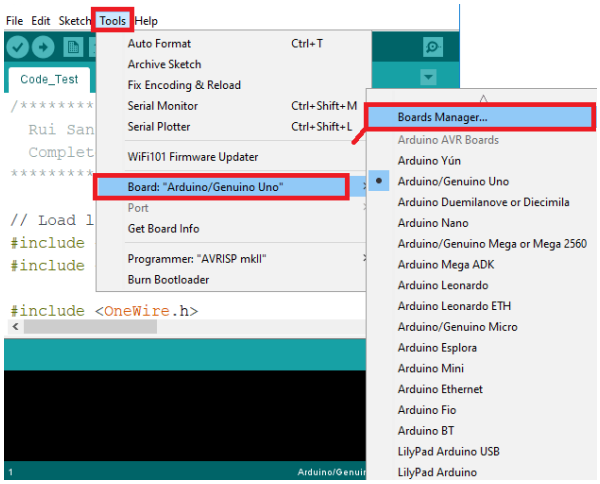


Fig. 13 Installing ESP32 in Arduino IDE step 3

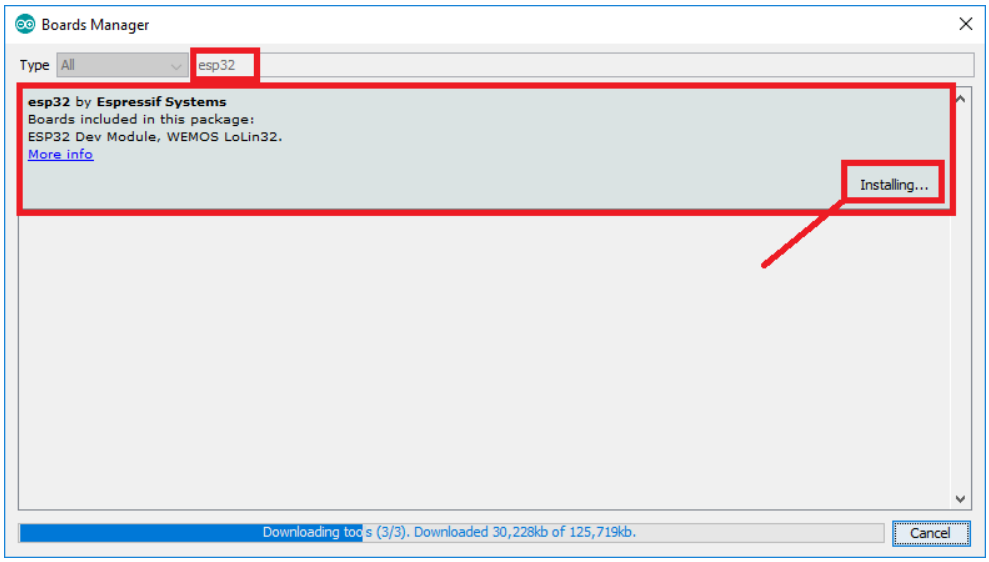


Fig. 14 Installing ESP32 in Arduino IDE step 4

Step 5: Choose the right board and start programming

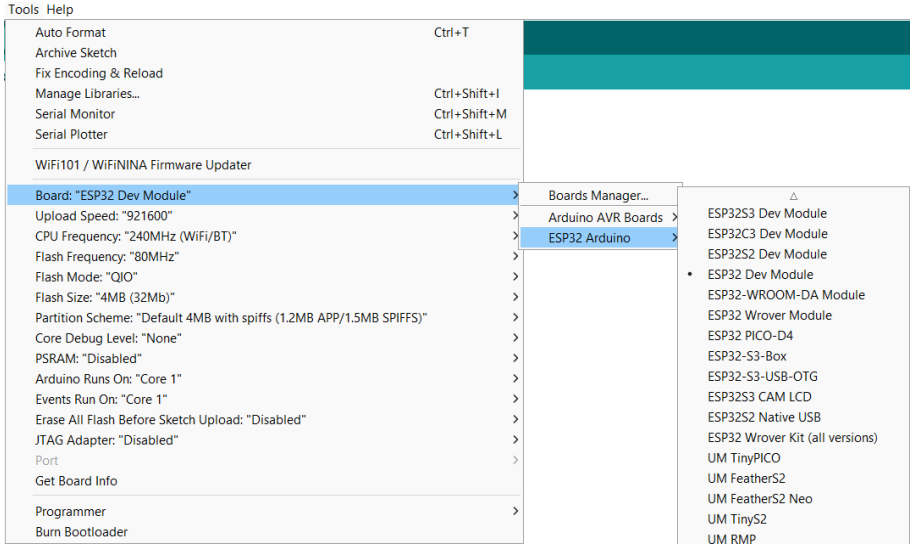


Fig. 15 Installing ESP32 in Arduino IDE step 5

3.2. ESP32 and Arduino Uno Programming

3.2.1 LCD + Adaptor I2C with ESP32

To facilitate the use of the display, which initially uses 16 pins, we opted for the purchase of a special I2C adapter module for this type of display. The LCD component used in the project was the one with 20 characters arranged on 4 lines. Using the module, we reduce the number of pins to 4 and now the display will use I2C communication with the ESP32 board, through the data and clock pins, SDA and SCL, plus the power, Vcc and GND pins. The adapter was soldered to the LCD by tinning in the faculty lab as seen in *Fig. 16*.

Fig. 16 Tinning the I2C adapter to the LCD display

Next came testing them by compiling and deploying a simple program on the ESP32 (*Fig.17*).

Fig. 17 Testing the display with ESP32 via I2C

3.2.2 MLX90614 temperature sensor with ESP32

Next, we connected the MLX90614 temperature sensor to the ESP32. The first time to be able to compile the program, we needed the library belonging to the sensor where the

libraries, special functions, definition of registers, pins, and other predefined elements are located (*Fig.18, Fig.19*).

Fig. 18 Installing the MLX90614 Sensor Libraries

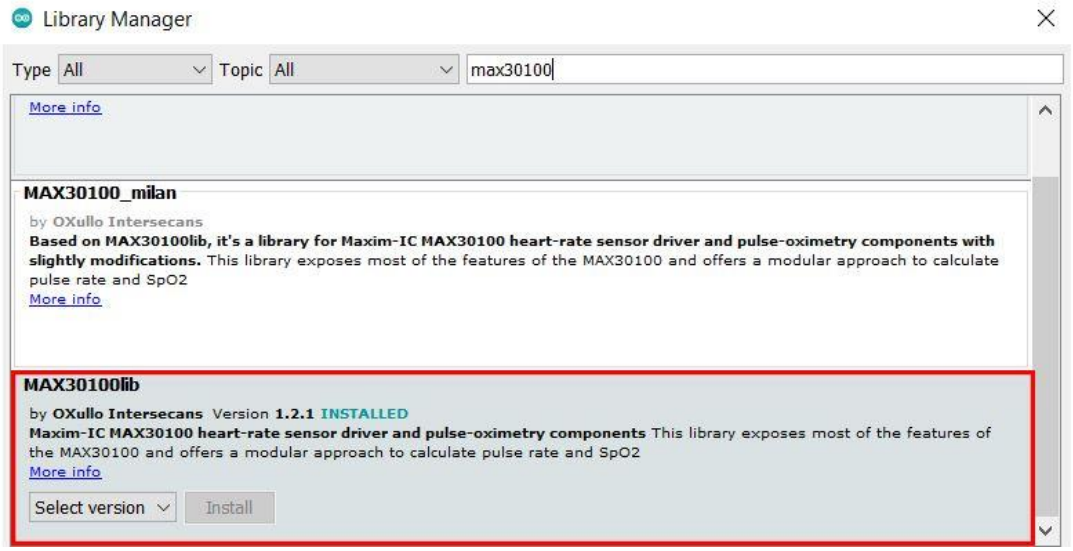


Fig. 19 Installing the MAX30100 Sensor Libraries

4. SYSTEM PROGRAMMING

The serial baud rate was set to 9600 in the setup() function. Then, also in this function, we initialized the temperature (mlx) measured by the sensor through the begin() method, describing the sequence according to the serial connection: as long as it does not exist and if the sensor is not properly connected, the serial monitor will notify us with an error message, and as long as the statement is true the program will wait.

```
while (!Serial);

if (!mlx.begin()) {
    Serial.println("Eroare la conectarea senzorului MLX. Verificați firele.");
    while (1);
};
```

In the loop() function we started the description of the temperature measurement logic. Thus, we defined a variable of type double to save the data retrieved by the sensor through the function readObjectTempC() in degrees Celsius. If the body temperature is greater than 36°C and less than 37.4°C, with a margin of error of about 0.2°C, then the temperature is optimal, a message displayed on both the serial monitor and the LCD screen.

```
double currTemp = mlx.readObjectTempC();
if (currTemp>36) {
    Serial.print("Temperatura = ");
    Serial.print(currTemp);
    Serial.println("°C");
    if (currTemp>36 && currTemp<37.4) {
        Serial.println("Temp optima");
         delay(2000);
        lcd.setCursor(1,0);
        lcd.print(currTemp);
        lcd.setCursor(1,1);
        lcd.print("Temp optima");
```

If the temperature is higher than 37.4°C and lower than 42-43°C, then the displayed message will be the one corresponding to the increased temperature.

```
    } else if (currTemp>37.4 && currTemp<43) {
        Serial.println("Temp crescuta");
         delay(2000);
        lcd.setCursor(1,0);
        lcd.print(currTemp);
        lcd.setCursor(1,1);
        lcd.print("Temp crescuta");
    }
```

If the retrieved temperature is abnormal, lower than 36°C and higher than 43°C, then the displayed message will be an error message and will instruct the person to take another measurement.

```
else if (currTemp<36 || currTemp>43) {
  Serial.println ("Temparatura incorecta. Reincercati.");
```

In the code described for the MAX30100 sensor we did the same for the temperature sensor. For the first time, we have included in the program the libraries where the functions, filters, registers and methods used in measuring pulse and blood oxygen level are defined.

```
#include <MAX30100.h>
#include <MAX30100_BeatDetector.h>
#include <MAX30100_Filters.h>
#include <MAX30100_PulseOximeter.h>
#include <MAX30100_Registers.h>
#include <MAX30100_SpO2Calculator.h>
```

We have defined the object that will call both the pulse function and the SpO2 function. Next, we used a void routine that will be automatically called by the program every time the sensor senses a pulse, and will show us on the serial monitor that it has sensed activity.

```
PulseOximeter pox;
void onBeatDetected() {
    Serial.println("♥ Beat!");
}
```

In the setup() function we specified as with the previous sensor, if the call to the MAX30100 sensor does not occur properly or does not occur at all, the message displayed by the serial monitor will be an error message and will not proceed further. Otherwise, it will tell us that the sensor initialization was successful.

```
if (!pox.begin()) {
    Serial.println("FAILED");
    for(;;);
} else {
    Serial.println("SUCCESS");
}
```

In the loop() function we specify if the time set since the last measurement has passed and we will start taking the information from the sensor one by one, displaying it on the serial monitor. The initial "pulseOk" and "spo2Ok" state variables will be initialized to 0.

```
if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
    int pulseOk = 0;
    int spo2Ok = 0;
    Serial.print("Puls:");
    Serial.print(pox.getHeartRate());
    Serial.println("bpm");
    Serial.print("SpO2:");
    Serial.print(pox.getSpO2());
    Serial.println("%");
```

For the pulse, we compared the recorded values with the normal range of the human pulse (50, 130). We have divided this range into 3 parts to interpret it: (50, 60), (60, 100) and (100, 130). If the pulse value is between 50 and 60 bpm, the pulse will be low. If it is between 60 and 100, the pulse will be normal. If it is between 100 and 130 the pulse will be high. Finally, after the pulse has been properly measured, the variable "pulseOk" is updated to 1.

```
if (pox.getHeartRate() > 50 && pox.getHeartRate() < 130) {
    if (pox.getHeartRate() < 60) {
        Serial.println("Puls scazut");
    } else if (pox.getHeartRate() < 100) {
        Serial.println("Puls normal");
    } else if (pox.getHeartRate() < 130) {
        Serial.println("Puls ridicat");
    }

    pulseOk = 1;
```

Likewise, for the oxygen level. The recording range was set between the percentages 78% and 101%. If the oxygen level is less than 90%, the SpO_2 will be low. Otherwise, if it is in the range (90, 101), SpO_2 is normal/good. Finally, the variable "spo2Ok" is updated to 1.

```
if (pox.getSpO2() > 75 && pox.getSpO2() < 101) {
    if (pox.getSpO2() < 90) {
        Serial.println("SpO2 scazut");

    } else {
        Serial.println("SpO2 normal");
    }

    spo2Ok = 1;
}
```

After recording the two quantities, the time since the last data acquisition by the sensor is updated with the time since the program started running to make the loop again wait until the next measurement.

```
tsLastReport = millis();
```

If what the sensor recorded is not within the normal measurement parameters, the serial monitor will show us a message specifying the information and suggesting that we try another measurement.

```
} else {
  Serial.println ("SpO2 sau Puls incorect. Reincercati.");
  Serial.println("----------------------------------------
}
```

The pulse sensor ended up being connected to the Arduino Uno as we said above, and the code and data interpretation logic remained the same, depending on the normal parameters set. For the code run by the ESP32 I needed some additional logic in addition to the temperature sensor, as the board also takes in the data recorded by the pulse and oxygen sensors on the Arduino.

As the Arduino and the ESP communicate with each other serially via UART, in the software communication chapter we needed a condition that would change every time the data

was successfully retrieved by the ESP from the Arduino. The first time I determined with two variables whether the pulse and the blood oxygen level were correctly recorded in the code of the Arduino board, and if so: I printed on the serial monitor a string of characters: "hrd_", with another character between the two measurements "_", which marks the difference between the two retrieved data, one for pulse and one for oxygen. The following sequence appears on the serial monitor: "hrd_puls_oxygen". After an appropriate measurement, the condition changes to signal this.

```
if (pulseOk && spo2Ok) {
    //Serial.println("Heart read done");
    Serial.print("hrd_");
    Serial.print(pox.getHeartRate());
    Serial.print("_");
    Serial.println(pox.getSpO2());
    cond = 0;
```

For the code on the ESP32, which is also the microcontroller that controls all the data recorded by the sensors, I needed a String variable to read the information from the serial monitor of the Arduino Uno microcontroller (Serial2). After reading what is written on Serial2, to later operate with them, I had to differentiate between the two measurements: pulse and oxygen, because they came on a single line in the form "hrd_puls_oxygen". Through two integer variables "_index" and "_index2", we recorded the position of the first character "_" in the string and the position of the second character "_", respectively. Why do we care about this? Because what is written after the first "_" is the measurement for pulse, which we are interested in, and what is after the second "_" is the measurement for SpO2.

Thus we started to read and record in two distinct variables of type String x and y, the values of the divided string: in x - what is between the two "_" in the big string (measurements corresponding to the pulse) and in y - what is between the the second ”_” and the end of the whole string (measurements corresponding to blood oxygen). After the proper recording of pulse and SpO2 data by the ESP32 has occurred, the condition changes to signal the success of the operations and to move on in the code to the temperature measurement by the sensor connected to the ESP32 itself.

```
while (!cond) {
    String s2ln = Serial2.readStringUntil('\n');
    int _index=s2ln.indexOf('_');
    if(_index==-1)
    {
      Serial.println(s2ln);
    } else
    {
      int _index2=s2ln.indexOf('_', _index+1);
      String x = s2ln.substring(_index + 1, _index2);
      String y = s2ln.substring(_index2 + 1, s2ln.length() - 1);
      cond = 1;
```

The imposed condition works like a switch. While a measurement is being recorded on one of the boards, the other sensor waits until the current one records a correct measurement, then tells it when it's done and can record the second sensor, and vice versa. The display on the LCD of the measurements for temperature, pulse and oxygen takes place after all the data has been received by the ESP, and will present the information to the users with the strict interpretation:

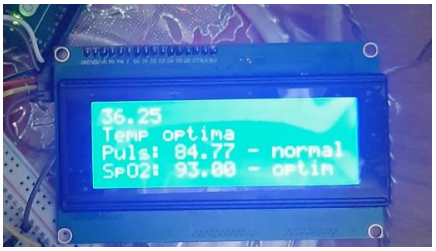


Fig. 20 LCD display of measurements

5. CONCLUSIONS

- The system was designed as a device that measures vital functions and records the data in a table in a database, it works, it uses simple, relatively cheap components, it corresponds to the original idea, the system can be easily used by anyone;
- The system can be used at home by people who want to record data about body temperature, heart rate and blood oxygen concentration, information needed by people at risk, sick or simply curious;
- The system can be used in hospitals, offices or any institutions that want to monitor people entering the premises, to be able to make statistics or to take measures to keep the area safe;
- It has applicability in checking athletes, such as swimmers, people who practice athletics or contact sports, to determine whether they are fit to start training, both in adults and children;
- You can sort the data recorded in the database table by size, or by type, to create advanced statistics in specialized applications.
- For a more accurate measurement and a low margin of error, high quality or more sensitive sensors can be used;
- An application can be created for the phone to be able to access the data from the sensors at any time;
- Power sources can be replaced by batteries or other individual sources.

REFERENCES

[1] S. Oniga, *Microprocesoare şi microcontrolere, note curs*, http://ece.ubm.ro/cursuri/.

[2] M. Margolis, B. Jepson, N. R. Weldin, *Arduino Cookbook*, 3rd Edition, O`Reilly, 2020.

[3] https://microcontrollerslab.com/mlx90614-non-contact-infrared-temperature-sensor-esp32/

[4] https://microcontrollerslab.com/max30100-pulse-oximeter-heart-rate-sensor-esp32/

[5] A. James, *Arduino: The complete guide to Arduino for beginners, including projects, tips, tricks, and programming!*, Ingram Publishing Ltd., 2019.

[6] http://www.esp32learning.com/code/esp32-and-max30100-heart-rate-monitor-sensor.php

[7] E. Lupu, Annamaria Mesaros, A. Suciu, *Microprocessors - Architectures and Applications*, Risoprint Cluj-Napoca, 2003.