Title:	PulseOximeter

Teaching Cycle **2024**	Course:	**R2004**	Group:	**5**

Members	Last Name First Names	File	Individual qualification	Date
Costarelli, Facundo		1762916		
Lagache, Ezequiel		2086669		

Group rating:		Date:	12/12/2024

Teacher:	Ing. Prieto Canalejo, Mariana Andrea
Teaching Assistant/s:	School Engineer, Jorge Ignacio

Observations first delivery	
Observations second installment	

INDEX

INTRODUCTION.....3
GOALS.....3
BLOCK DIAGRAMS.....4
DEVELOPMENT.....5
HARDWARE AND SOFTWARE DESCRIPTION.....5
Block Diagrams.....5
Tools and equipment.....6
Project Specifications and Limitations.....6
Circuit and PCB.....7
Main program and drivers.....8
QT app.....13
Data sheets.....15
PROBLEMS FOUND IN THE DEVELOPMENT OF THE PROJECT.....16
BENEFITS FOUND IN THE DEVELOPMENT OF THE PROJECT.....18
CONCLUSION.....19
ANNEX.....20

INTRODUCTION

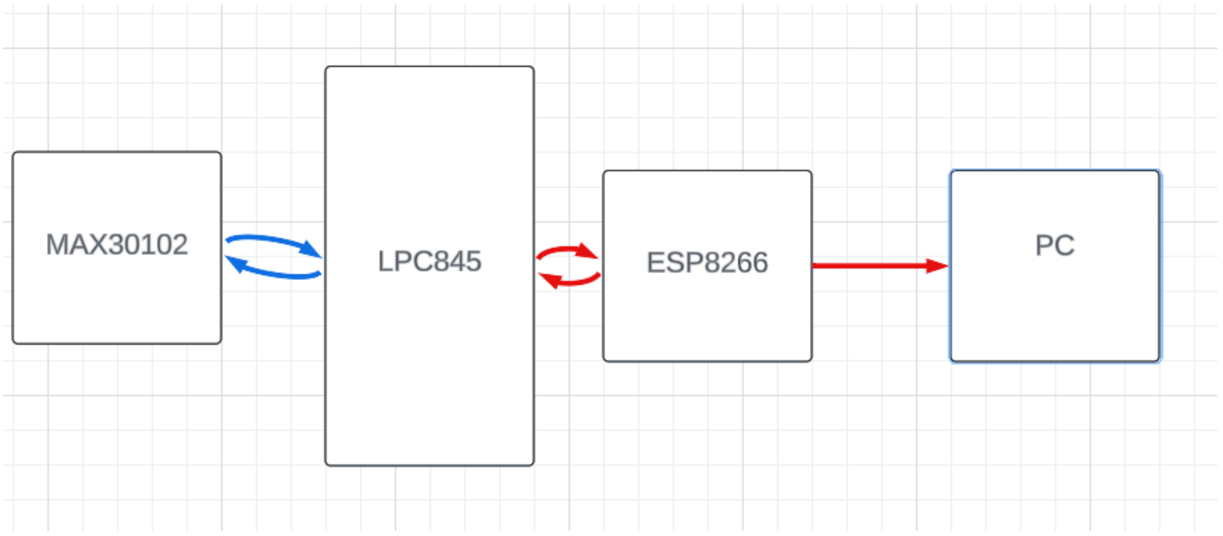
This project seeks to develop a prototype pulse oximeter device in terms of software and basic and simple hardware to apply the software. For the first, an application will be created with its drivers, so that through a graphical interface the measured values of blood oxygen saturation (SPO2) and heart rate (Heart Rate) can be displayed in real time, obtained through a MAX 30102 sensor module. Communication between the sensor and the interface is carried out through an ESP8266 module, using the WiFi TCP protocol to transmit the data to a portable PC. The control of the modules is directed by the LCP 845 microcontroller. For the second, it is intended to combine programmable modules on a single PCB.

This project integrates multiple concepts of object-oriented programming, graphical interface design, wireless communication, management of intermodular communication protocols such as I2C, schematic development with PCB and connection of modules and necessary hardware components. The project is aimed at accessible and efficient biomedical applications.

GOALS

- Study and application of knowledge of the LPC845 stick.
- Incorporate and strengthen C++ theory, OOP concepts, Embedded Systems and Signal and Systems Analysis.
- Unite application concepts with driver concepts, both developed for the microcontroller and modules.
- Establish communication between programmable modules and the microcontroller with I2C protocol, as well as serial communication.
- Establish a TCP/IP wireless connection between a QT interface application and the microcontroller application along with its drivers.
- Incorporate Internet Of Things concepts.
- Implement hardware development concepts in a basic way using KiCad.
- Introduce and study basic bioengineering concepts such as SPO2 measurement and HeartBeat.

BLOCK DIAGRAMS

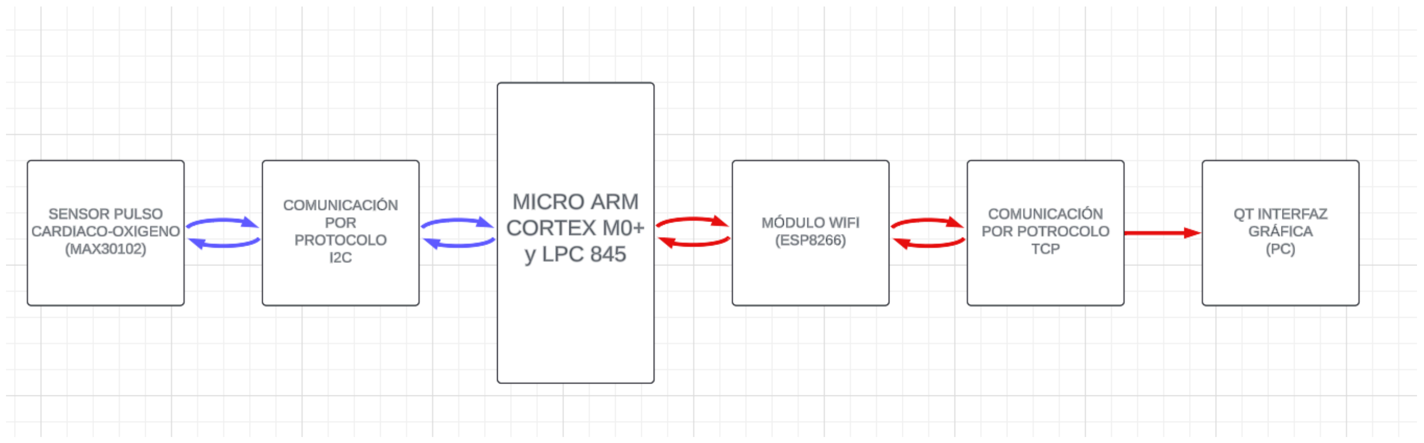


With this simple diagram we can see that the LPC845 microcontroller communicates with the MAX 30102 sensor to request the censused SPO2 and Heart Rate data. This data is processed with an algorithm and subsequently sent to the ESP8266, which is responsible for transmitting the information via WiFi to the PC. Said device receives them and processes them again in such a way as to be able to display them in a graphical interface with simple animations.

DEVELOPMENT

HARDWARE AND SOFTWARE DESCRIPTION

Block Diagrams


SENSOR PULSO CARDIACO-OXIGENO (MAX30102)
COMUNICACIÓN POR PROTOCOLO I2C
MICRO ARM CORTEX M0+ y LPC 845
MÓDULO WIFI (ESP8266)
COMUNICACIÓN POR POTROCOLO TCP
QT INTERFAZ GRÁFICA (PC)

The MAX 30102 sensor emits electromagnetic waves with wavelengths such that one of them is visible red while the other is not visible and is infrared. These waves collide with the molecules present in the blood of a finger, reflecting in such a way that they are captured by a photodiode. The changes produced in the emitted waves with respect to the reflected ones depend on the heart rate and the oxygenation present in the blood. To obtain the SPO2 and Heart Rate values, calculations, signal filtering processes and ADC analog-digital conversion are carried out, where everything is carried out by the sensor module's own hardware, allowing us to read a train of pulses at its output.

Communication with said sensor is via I2C protocol such that the pulse train corresponds to this coding. Communication is established between the microcontroller in Master mode while the sensor functions as Slave. An initial configuration of the sensor is carried out from the microcontroller by modifying its registers to achieve the aforementioned functionalities. Sensor data reading methods are also implemented.

For its part, the raw SPO2 and Heart Rate data are processed by the LPC845 microcontroller through an algorithm suggested by the manufacturer of the Maxim sensor. With the data obtained, a second filtering is carried out using a weighted average, in order to achieve greater reliability in the measurement. The processed data is then sent to the ESP8266 WiFi module.

The ESP8266 module is initially configured through AT commands without strictly modifying its registers. Then communication is established with the PC via TCP protocol using AT commands, through the WiFi zone emitted by the mobile device. On the other hand, the PC acts as the server, and the ESP8266 module as the client. With communication established, the SPO2 and Heart Rate data are sent to the PC, which receives and processes them with a graphical interface previously created in QT.

Tools and equipment

- MAX 30102 heart rate and oxygen saturation sensor.
- ESP8266 WiFi module.
- LPC845 microcontroller.
- DC-DC Step Down Module Mini-360 Converter Source 3A 0.8V - 20V
- PCB board 10cm long and 3cm wide, 1 1n4148 diode, 1 100uF 50V electrolytic capacitor, female pin strip and 12x7x5.5 cm cabinet.
- 9V DC battery power supply.
- PC or Notebook for QT graphical interface.

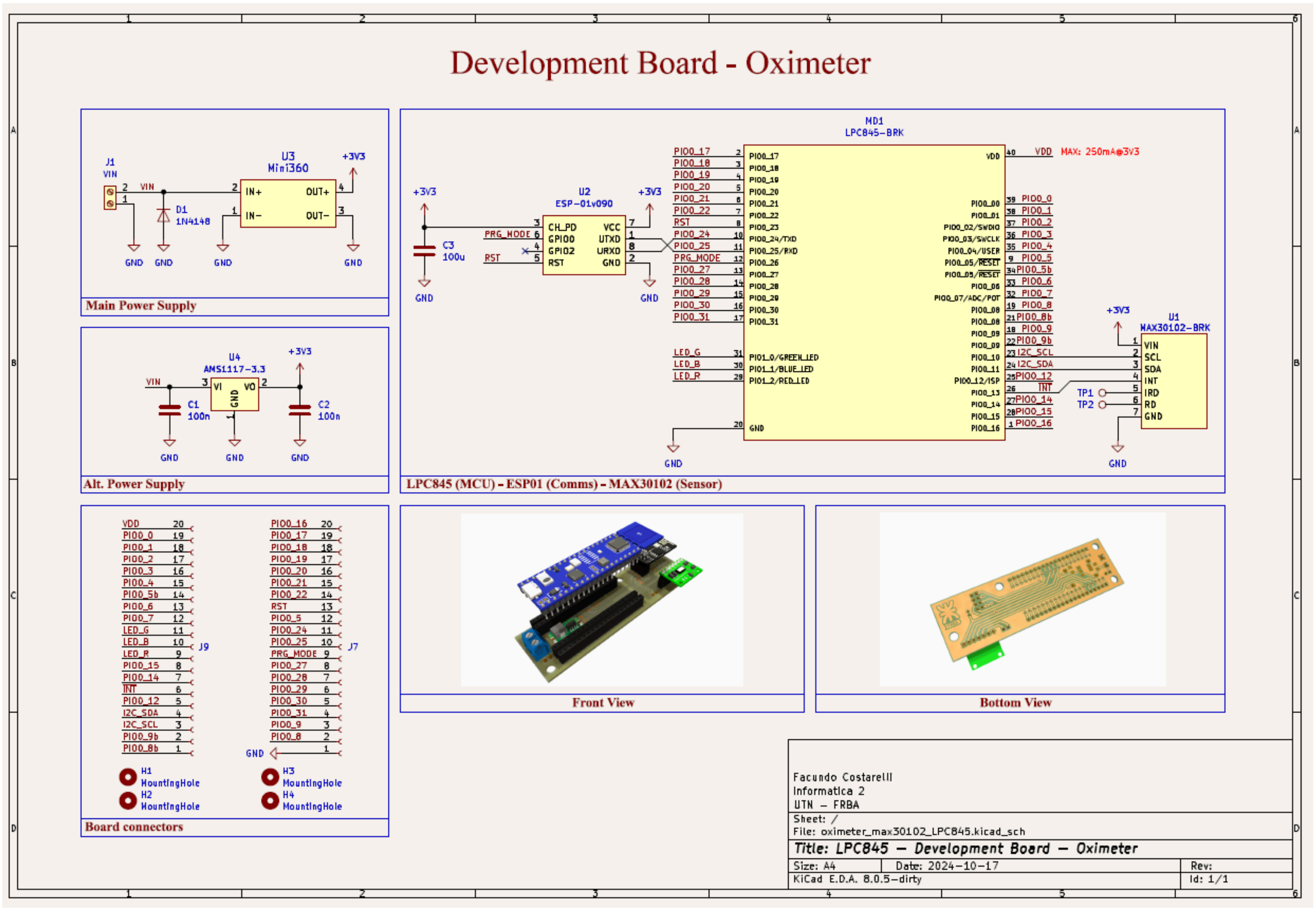
Project Specifications and Limitations

- Input power of all modules: 3.3V.
- Typical current consumption of the project: 150-200mA.
- DC-DC regulator efficiency is ~75%
- Estimated maximum duration with 9V battery: ~2-3 hours under typical load.
- Separate power supply for modules with regulator and for the LPC845 via USB port.
- Modular design with connection between MAX 30102, ESP8266 and LPC845.
- Main power based on AMS1117 regulator and Mini360.
- The appearance of combined current spikes from the ESP8266 and the sensor can increase the temperature in the regulator and the microcontroller. You cannot connect the VCC pin of the LPC845 to the VCC pin of the ESP8266 since the latter typically consumes more than 200 mA, exceeding the current availability that the microcontroller can offer. Feedings must be separate.
- I2C communication protocol: Instability and noise compared to wiring for communication. It is necessary to design short copper traces and/or adequate shielding to minimize interference. Maximum data transfer speed of 400 KHz. Robust software is required for the correct handling of everything.
- UART communication protocol: Recommended Baud rate 9600 or 115200 bps for reliable communication. Using the DC-DC regulator can introduce noise that can interfere with RX/TX. The buffers are small so they need to be emptied or refreshed quickly. Because it is a serial protocol, it has

an intrinsic latency due to the sequential transmission of bits. It only allows direct communication between two nodes. For multiple devices, additional configurations such as UART-to-Multiplexers or converters are needed. Robust software is required for the correct handling of everything.

Circuit and PCB

From the following image we see the schematic circuit plan of the device as well as its 3D modeling and its PCB board with the modules:



For the development of the device we used the KiCad software, with the intention of eliminating the problems of false contact in connection pins that we had on a breadboard, and reducing the use of unnecessary cables. In this way, all modules are fixed and placed on pins, offering the possibility of replacing them with identical spare modules in case one breaks. Furthermore, the tracks are as short as possible in width and length, which mainly allows the noise introduced in I2C, TCP and UART communication to be significantly reduced.

The power input is given by the blue terminal block, allowing a DC input voltage of up to 20V. Since this is connected to a DC-DC Step Down regulator that is located just below the LPC845, we can have hardware portability, allowing the input to be food with batteries of different voltages, such as 9V. The regulator output is adjusted with a screw located on the regulator plate. We leave it configured to have 3.3 V output. Its output current can be up to 3A depending on the demand of the devices.

It is important to note that in a Step Down device, the input must have a higher voltage than that expected at the output. So, in this case, we could not enter with a voltage lower than 3.3V. The tests were carried out with 9V input batteries and laboratory sources at 5V input where positive results were evaluated.

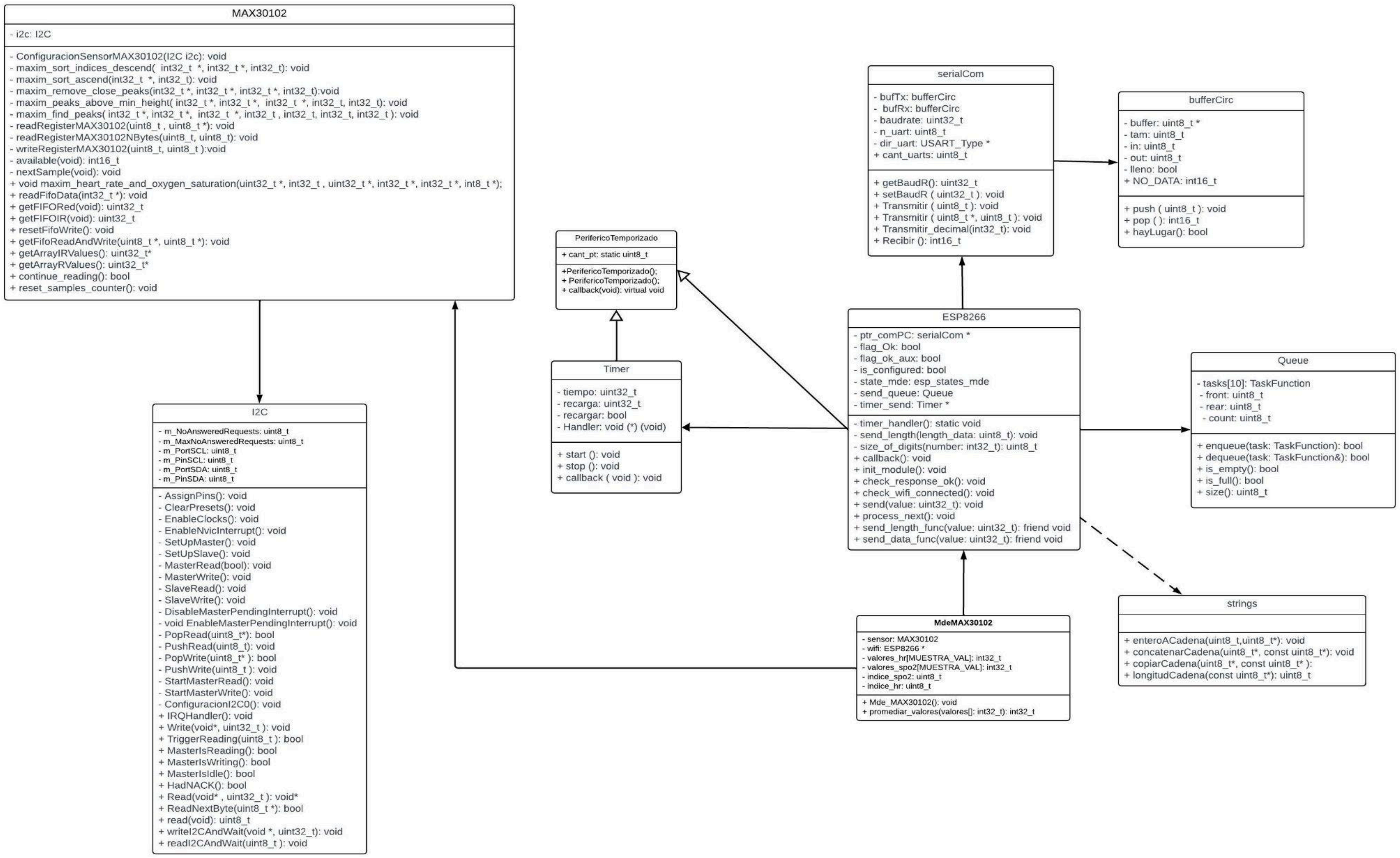
On the other hand, we highlight that in this design the output voltage of the regulator affects only the WiFi module and the sensor, while to power the microcontroller module it is necessary to connect it to a PC with a mini USB-USB C cable. The need to have separate power supplies is due to the fact that if all the modules are connected to each other through their VCC pins at 3.3V, these pins being linked in turn to the Vout of the regulator, it happens that the WiFi module demands a lot of current during its operation, demanding more and more current from the microcontroller that can exceed the maximum 200 mA that the microcontroller can offer. This results in overheating of the WiFi module and the microcontroller module until it breaks. To detect this situation we have placed, post-design and post-construction of the board, a pair of pins as a “jumper” that unifies the VCCs with each other with the Vout pin of the regulator. Possible improvements in this regard are mentioned in the Annex section.

Finally, we can see a fast diode 1N4148 placed at the input with the terminal block just before reaching the regulator. This is to protect against short circuit caused by a possible reversal of the power cables at the terminal block. Also, given that the ESP8266 WiFi module produces current peaks during its operation, we have placed a 100uF 50V electrolytic capacitor as protection to absorb these currents and avoid problems in the rest of the circuit.

Main program and drivers

The main program is made up of a group of lines of code that execute the initialization of the module LPC845, UART module serialCom, ESP8266 module and sensor module; and a while(1) that repeatedly executes the State Machine (MDE) of the application. This involves the interaction of all modules. The code can be seen in the project.

Below is a class diagram that relates all the module drivers as well as the utilities implemented to achieve control of all the modules. The aim is to observe without too much detail what the general structure of our system is. The link is attached in the Annex section.


MAX30102
- i2c: I2C
- ConfiguracionSensorMAX30102(I2C i2c): void
- maxim_sort_indices_descend(int32_t *, int32_t *, int32_t): void
- maxim_sort_ascend(int32_t *, int32_t): void
- maxim_remove_close_peaks(int32_t *, int32_t *, int32_t *, int32_t):void
- maxim_peaks_above_min_height(int32_t *, int32_t *, int32_t *, int32_t, int32_t): void
- maxim_find_peaks(int32_t *, int32_t *, int32_t *, int32_t , int32_t, int32_t, int32_t): void
- readRegisterMAX30102(uint8_t , uint8_t *): void
- readRegisterMAX30102NBytes(uint8_t, uint8_t): void
- writeRegisterMAX30102(uint8_t, uint8_t):void
- available(void): int16_t
- nextSample(void): void
+ void maxim_heart_rate_and_oxygen_saturation(uint32_t *, int32_t , uint32_t *, int32_t *, int32_t *, int32_t *, int8_t *);
+ readFifoData(int32_t *): void
+ getFIFORed(void): uint32_t
+ getFIFOIR(void): uint32_t
+ resetFifoWrite(): void
+ getFifoReadAndWrite(uint8_t *, uint8_t *): void
+ getArrayIRValues(): uint32_t*
+ getArrayRValues(): uint32_t*
+ continue_reading(): bool
+ reset_samples_counter(): void
I2C
- m_NoAnsweredRequests: uint8_t
- m_MaxNoAnsweredRequests: uint8_t
- m_PortSCL: uint8_t
- m_PinSCL: uint8_t
- m_PortSDA: uint8_t
- m_PinSDA: uint8_t
- AssignPins(): void
- ClearPresets(): void
- EnableClocks(): void
- EnableNvicInterrupt(): void
- SetUpMaster(): void
- SetUpSlave(): void
- MasterRead(bool): void
- MasterWrite(): void
- SlaveRead(): void
- SlaveWrite(): void
- DisableMasterPendingInterrupt(): void
- void EnableMasterPendingInterrupt(): void
- PopRead(uint8_t*): bool
- PushRead(uint8_t): void
- PopWrite(uint8_t*): bool
- PushWrite(uint8_t): void
- StartMasterRead(): void
- StartMasterWrite(): void
- ConfiguracionI2C0(): void
+ IRQHandler(): void
+ Write(void*, uint32_t): void
+ TriggerReading(uint8_t): bool
+ MasterIsReading(): bool
+ MasterIsWriting(): bool
+ MasterIsIdle(): bool
+ HadNACK(): bool
+ Read(void* , uint32_t): void*
+ ReadNextByte(uint8_t *): bool
+ read(void): uint8_t
+ writeI2CAndWait(void *, uint32_t): void
+ readI2CAndWait(uint8_t): void
PerifericoTemporizado
+ cant_pt: static uint8_t
+PerifericoTemporizado();
+ PerifericoTemporizado();
+ callback(void): virtual void
Timer
- tiempo: uint32_t
- recarga: uint32_t
- recargar: bool
- Handler: void (*) (void)
+ start (): void
+ stop (): void
+ callback (void): void
serialCom
- bufTx: bufferCirc
- bufRx: bufferCirc
- baudrate: uint32_t
- n_uart: uint8_t
- dir_uart: USART_Type *
+ cant_uarts: uint8_t
+ getBaudR(): uint32_t
+ setBaudR (uint32_t): void
+ Transmitir (uint8_t): void
+ Transmitir (uint8_t *, uint8_t): void
+ Transmitir_decimal(int32_t): void
+ Recibir (): int16_t
bufferCirc
- buffer: uint8_t *
- tam: uint8_t
- in: uint8_t
- out: uint8_t
- lleno: bool
+ NO_DATA: int16_t
+ push (uint8_t): void
+ pop ( ): int16_t
+ hayLugar(): bool
ESP8266
- ptr_comPC: serialCom *
- flag_Ok: bool
- flag_ok_aux: bool
- is_configured: bool
- state_mde: esp_states_mde
- send_queue: Queue
- timer_send: Timer *
- timer_handler(): static void
- send_length(length_data: uint8_t): void
- size_of_digits(number: int32_t): uint8_t
+ callback(): void
+ init_module(): void
+ check_response_ok(): void
+ check_wifi_connected(): void
+ send(value: uint32_t): void
+ process_next(): void
+ send_length_func(value: uint32_t): friend void
+ send_data_func(value: uint32_t): friend void
Queue
- tasks[10]: TaskFunction
- front: uint8_t
- rear: uint8_t
- count: uint8_t
+ enqueue(task: TaskFunction): bool
+ dequeue(task: TaskFunction&): bool
+ is_empty(): bool
+ is_full(): bool
+ size(): uint8_t
MdeMAX30102
- sensor: MAX30102
- wifi: ESP8266 *
- valores_hr[MUESTRA_VAL]: int32_t
- valores_spo2[MUESTRA_VAL]: int32_t
- indice_spo2: uint8_t
- indice_hr: uint8_t
+ Mde_MAX30102(): void
+ promediar_valores(valores[]: int32_t): int32_t
strings
+ enteroACadena(uint8_t,uint8_t*): void
+ concatenarCadena(uint8_t*, const uint8_t*): void
+ copiarCadena(uint8_t*, const uint8_t*):
+ longitudCadena(const uint8_t*): uint8_t

We explain the operation of the system, the class diagram and the communication protocols used: I2C, TCP, UART.

The system is designed in such a way that initialization of the microcontroller module is first performed using the function InitializePLL(), then instance and initializes a type object serialCom and one type ESP8266 for the WiFi module. Subsequently, the configuration of the WiFi module is carried out using a while loop. This while is blocking but does not negatively affect the project since it is part of the system initialization block. Immediately after instance and initializes the sensor module and finally a while(1) is executed with a state machine that controls the application. It is interesting to study in detail how the sensor and WiFi modules are configured and used in the application.

On the one hand, communication is established between the LPC845 microcontroller module and the ESP8266 WiFi module by using AT commands. Specifically, to configure the WiFi, a small state machine is executed within a while loop, given as while(!wifi.check_wifi_connected()){ wifi.init_module();}. Once the connection is established, a group of methods is used to achieve the transmission of the data obtained by the sensor to a PC, through a FIFO type queuing system. We do not study the reception of data from the PC in this project since it is not necessary. It is important to note that the communication is via the TCP protocol, therefore, we must first send the size of the message or data that we want to transmit expressed in number of bytes, and then we must send the message itself. For each command sent to the WiFi module, it responds accordingly with different strings that have the keyword “OK\r\n”. It is important to be able to receive this “OK\r\n” for each AT command executed since it is the way to know what stage the WiFi module is in and if everything is working correctly. To achieve correct reception, we use a state machine that uses serial port and UART protocol, which will be explained later.

On the other hand, communication is established between the LPC845 microcontroller module and the MAX 30102 sensor module using the I2C protocol. In particular, to configure the sensor, commands must be sent consisting of the address of an internal register to be modified and the data to be loaded into that register, both in hexadecimal format. First the address must be sent and then the data thought of as a group of bits that must be placed in that register with bit-level operators. In this mode, the microcontroller is in Master mode, while the sensor is in Slave mode. This configuration defines the resolution and sampling frequency, what data we are going to sense and, based on that, what LEDs to turn on, among other things. Once the configuration is done, we can see the activation of a red and infrared LED light to measure the level of oxygenation in the blood (SPO2) and heart rate (Heart Beat). Analog data is captured by the sensor by photopleigtism, filtered and converted with ADC, and saved as samples in memory spaces managed by pointers called FIFO write and FIFO read. All of the latter mentioned is done by the sensor module's own hardware. The raw data processing algorithm is the one used by the manufacturer Maxim, it is not as precise and exact but it provides interesting measurements.

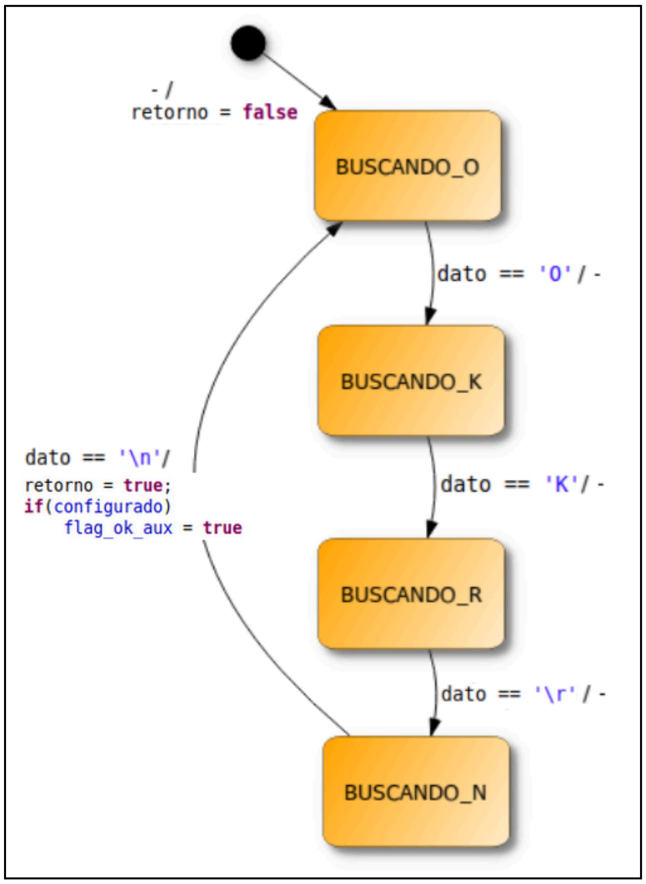
Finally, the application itself is controlled by a state machine, based on the sensor data sheets where two transitions are established. In the first, the sensor is asked to indicate the number of available samples according to the direction where the FIFO write and read pointers point; and in the second, the samples are read and a data processing algorithm is applied to obtain a SPO2 and Heart Rate value in order to transmit them via WiFi, be received on the PC and processed again by the QT graphical interface program.

We now analyze the different state machines of our project.

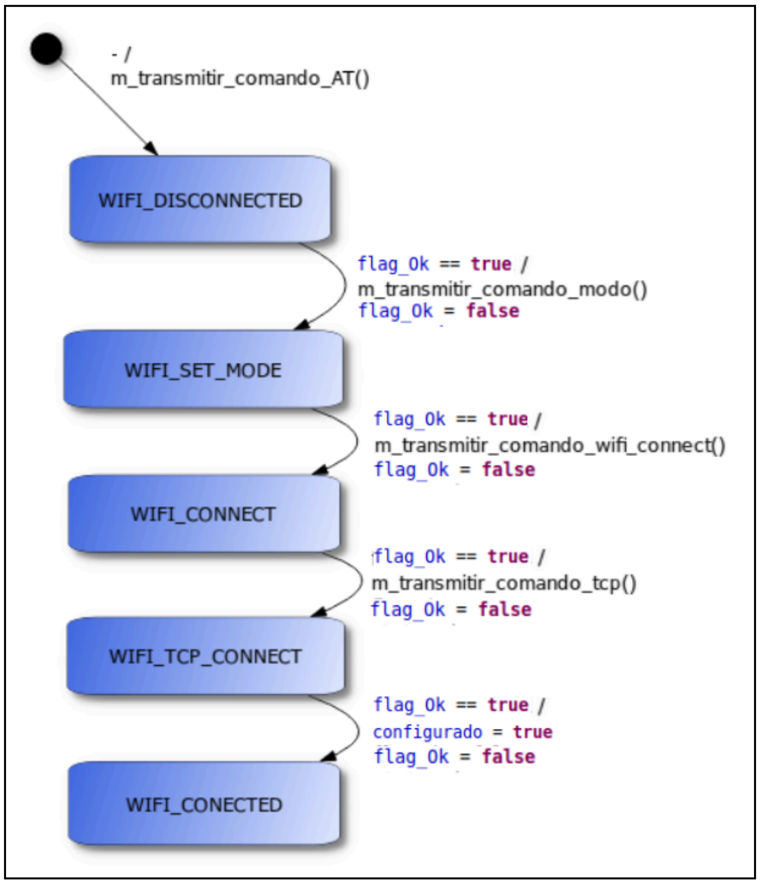
In the first state machine, in orange, we can observe the logic to control the arrival of the string "OK\r\n" that is sent by the ESP8266 WiFi module in each of the operating stages. This is both in the configuration and in the transmission and reception. The state machine receives a piece of data and asks for its value in ASCII code. When the received data is equal to 'O', 'K', or '\r', it goes to the next state. If, on the other hand, the data received is equal to '\n', which means that all the characters arrived, it returns a boolean true, sets an auxiliary flag to true that we will use stops the transmission and resets to its initial state, that is, to the state where the 'O' waits. Otherwise false is returned.

In the second state machine, in blue, we observe the logic that allows the configuration of the WiFi module and its connection with the PC, which, as mentioned above, will act as a server. It consists of five states and, when changing from one to another, the module transmits the different commands necessary to establish the connection. The condition to go from one state to another is precisely that the boolean that indicates whether the string "OK\r\n" was received is true. When you get to the state WIFI_CONNECTED the connection is established, so you won't have to go back to any other state.

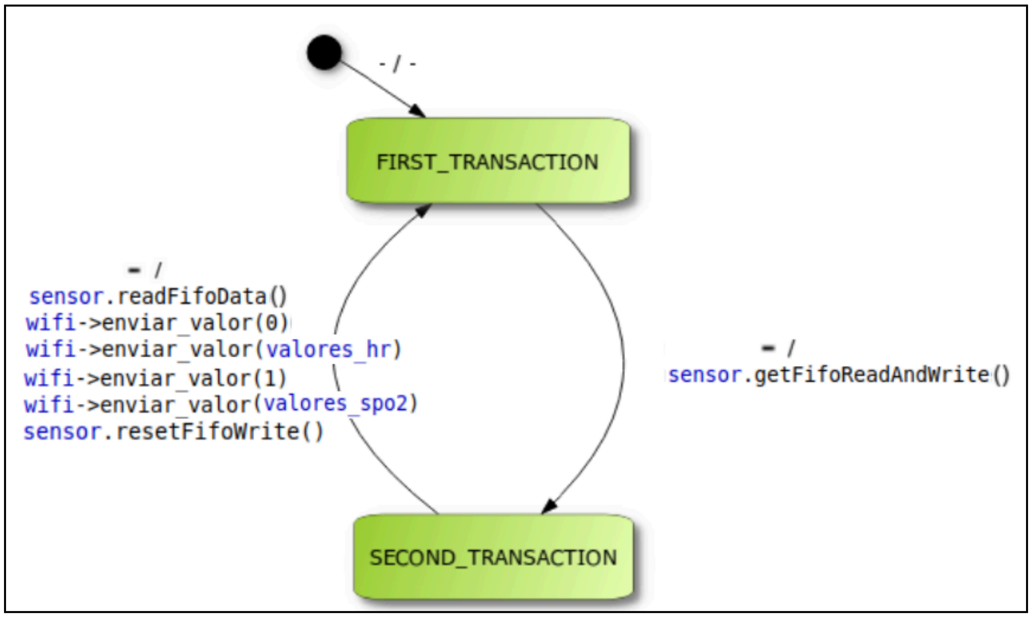
The third state machine of the project, in green, is the state machine that controls the application. It consists of two states, the first called FIRST_TRANSACTION, where the different sensor measurements are processed and stored, and the second, SECOND_TRANSACTION, which reads the collected information and sends it to the PC. Before sending the information, perform a weighted average, so that it is more reliable and is not so affected by garbage or errors in any measurement. To send the information, first send a header (0 or 1) so that the Qt interface knows what measurement it is (whether it is heart rate or blood oxygenation level), and then send the measurement value. After sending the two measurements, it cleans the sensor and leaves it ready to collect measurements again in the state FIRST_TRANSACTION.



State Machine I



State Machine II



State Machine III

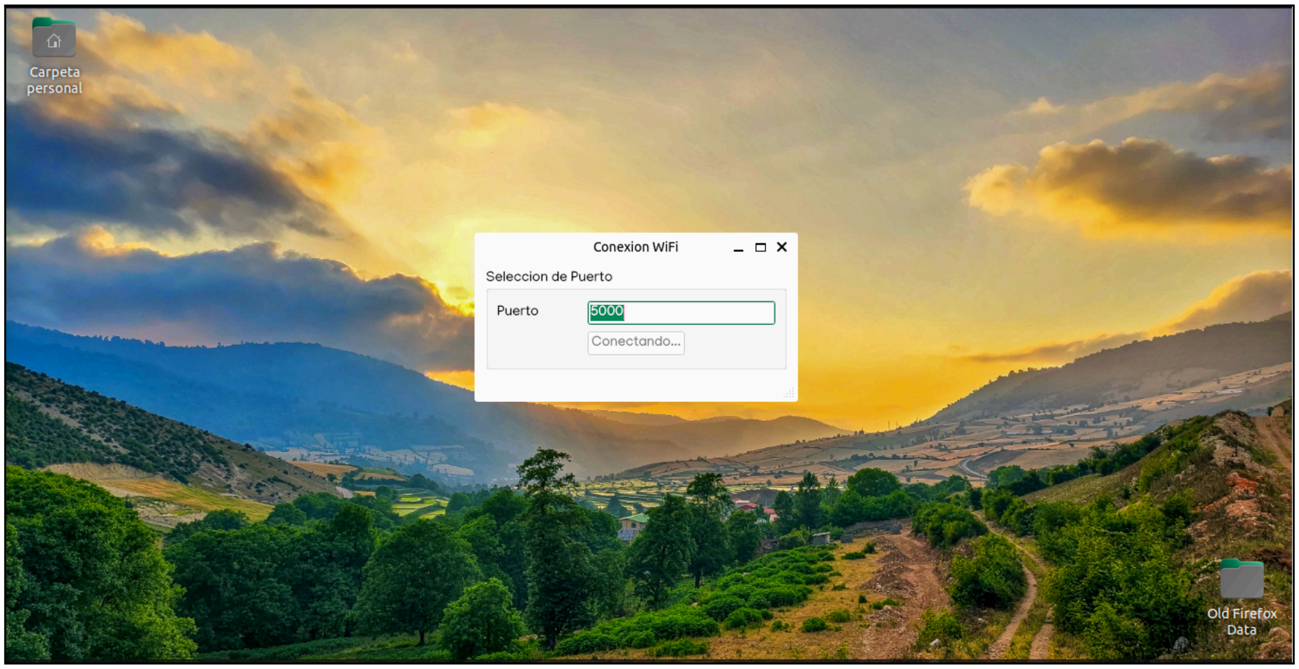
QT app

To present the information sent by the sensor, we developed an interface using the Qt development environment. It consists of four interactive windows, which allow you to do different things according to the user's needs.

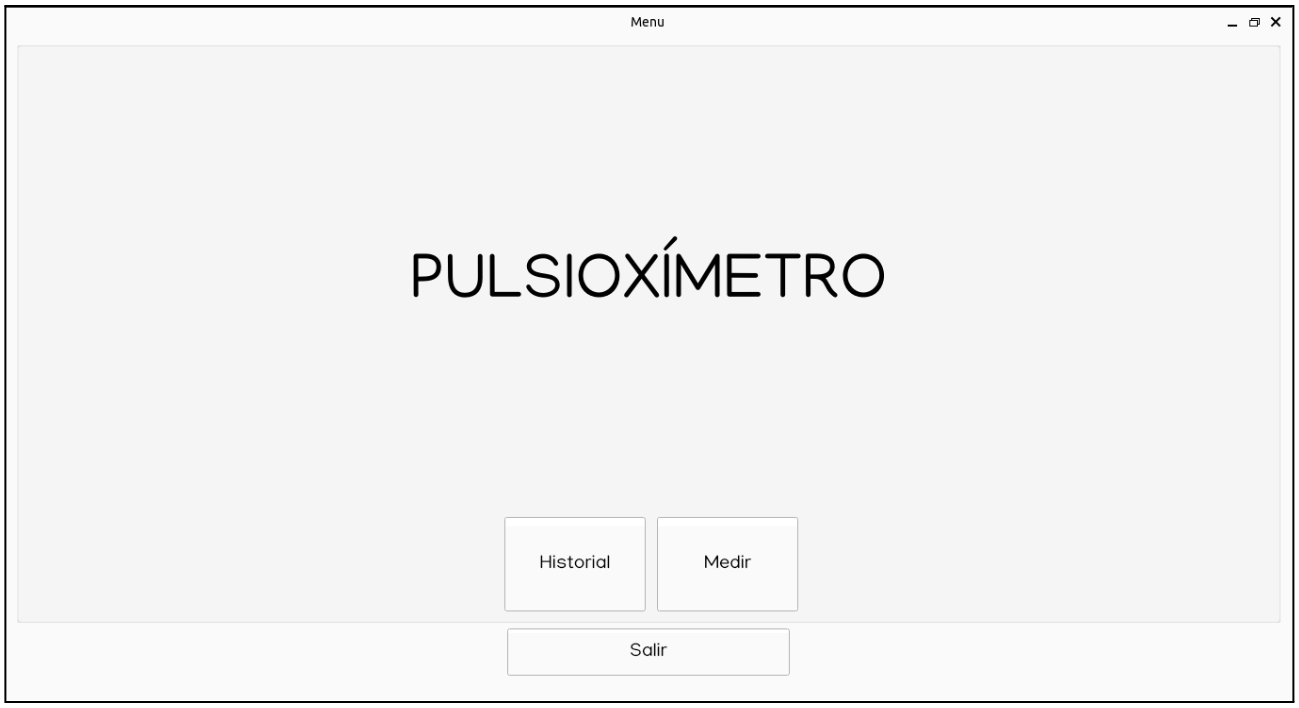
As soon as you start, the WiFi connection window is displayed, which allows you to select the port on which we are going to listen to the client connections. Once the WiFi module is connected, the menu is displayed, which offers us two options: we can display a new window that will show us the measurements sent by the WiFi module, or we can open a history with all the measurements made and that were previously saved.

If the user chooses to display the measurements window, in addition to being able to see the measured data, they will have the possibility of saving them in a file, by pressing the “Save” button. When you do so, a dialog is displayed that will allow you to enter your information, such as name, surname or ID, among others.

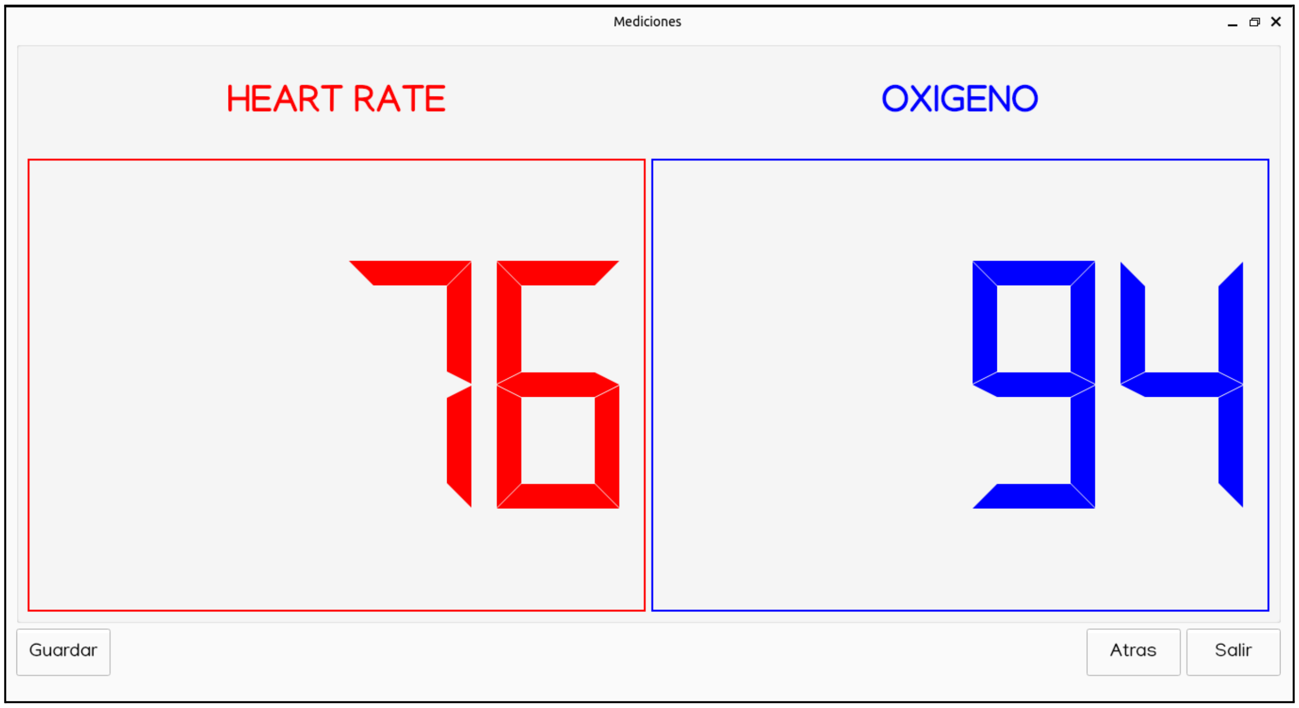
If, on the other hand, the user decides to open the history, the file with all the saved information will be opened and displayed (if it exists). If such a file does not exist, a warning will appear. In addition, you are offered the possibility of editing the history, or deleting it, by pressing the corresponding buttons.



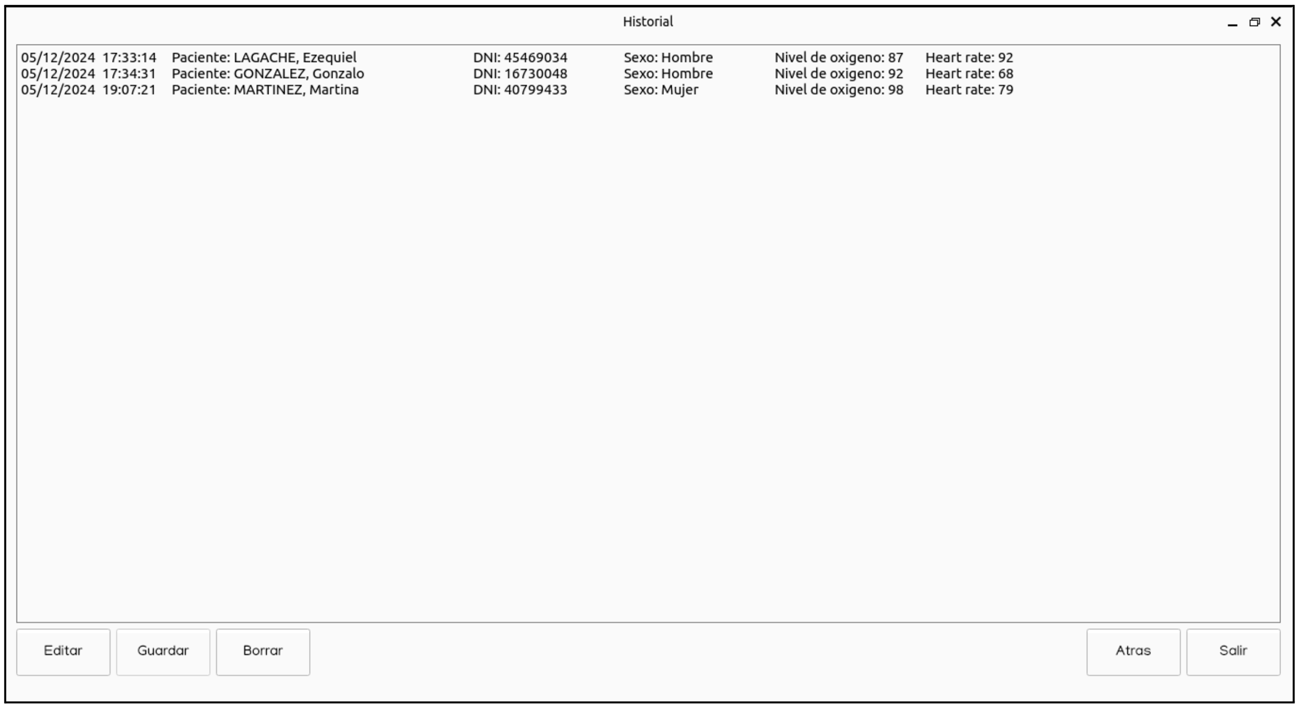
WiFi connection window



Menu



Measurement window



```
05/12/2024  17:33:14   Paciente: LAGACHE, Ezequiel      DNI: 45469034   Sexo: Hombre   Nivel de oxigeno: 87   Heart rate: 92
05/12/2024  17:34:31   Paciente: GONZALEZ, Gonzalo      DNI: 16730048   Sexo: Hombre   Nivel de oxigeno: 92   Heart rate: 68
05/12/2024  19:07:21   Paciente: MARTINEZ, Martina      DNI: 40799433   Sexo: Mujer    Nivel de oxigeno: 98   Heart rate: 79
```

Record

Data sheets

- MAX 30102 sensor module
  - MAX30102 Datasheet and Product Info | Analog Devices
  - MAX30102--High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health
- ESP8266 WiFi Module and AT Commands
  - ESP8266EX
  - ESP8266 Technical Reference
  - ESP8266 AT Instruction Set
  - Basic AT Commands — ESP-AT User Guide documentation
- LPC845 microcontroller module
  - LPC845 Breakout Board for LPC84x family MCUs | NXP Semiconductors
  - LPC84x Data Sheet
  - UM10601 LPC800 User manual
- DC-DC Regulator Module Mini360
  - MP2307
- Diodo 1N4148
  - 1N4148 Datasheet(PDF) - NXP Semiconductors

PROBLEMS FOUND IN THE DEVELOPMENT OF THE PROJECT

During the implementation of the project we encountered various problems both from the research stage and in the hardware and software development stages. We mention the most relevant problems as well as the solutions we found.

- During the research and core idea stage of the project, we first encountered problems obtaining devices that were cheap and easy to program, as well as that had enough information on the internet about it, discussions in forums, support and so on. In this sense, with the participation of the university's GBIO in the project we were able to easily access these devices by providing us with the MAX 30102 sensor and the ESP8266 WiFi module. Both used by other student members of the GBIO laboratory previously and with experience.
- When making a simple and simple hardware prototype, we focus on using a breadboard, where we connect the sensor and the microcontroller. However, we encounter noise in communication, produced by long breadboard cables and false contact; long Tx and Rx pin distances between sensor and module; and noise introduced by laboratory DC sources. This is solved making a PCB board by soldering strips of pins so that the modules are inserted there, remaining fixed and without false contact with short tracks between Tx and Rx pins. A battery was introduced as a power source instead of a laboratory source.
- When obtaining the PBC board, we are faced with the dilemma of whether or not to unify the supply voltages between all the modules. To do this, we place a jumper between the VCC of the microcontroller module, the remaining modules and the Vout of the DC-DC regulator. However, this introduced an additional problem which led us to eliminate the use of the jumper. This problem consisted of high current demands from the WiFi module to the microcontroller and not to the regulator, resulting in overheating and destruction of the WiFi module or the microcontroller. We then opted not to use the jumper, which is why the power supplies are separated, leaving the output of the regulator that powers the MAX 30102 sensor and the ESP8266 WiFi module while a USB mini-USB C cable powers the microcontroller.
- When wanting to configure the WiFi module, we found different alternatives, where one of them was quite common and consisted of using a TTL serial adapter module named FTDI232, which in theory facilitates the visualization of the commands transmitted and received between the module and the LPC845. This module did not work correctly and we did not fully understand how to use it. As a solution, we chose to use the IDE console of the MCUXpresso through the UART protocol and specifically using UART0 of the LPC845, since when using another UART peripheral, it was not possible to display the AT commands on the console.
- During the development of the sensor driver software, we encountered I2C management problems, where we could not correctly send in a timely manner the address of the register to be modified as well as the bits to be placed in said registers. We deduced through multiple tests and forums that it should be sent with a certain delay between one message and another, first

the address of the register and then the hexadecimal data associated with the bits to be placed in that register. We also had conceptual problems with the protocol. As a solution we turned again to NXP forums, websites and GBIO example projects to guide us. Additionally, we used a logic analyzer from Saleae Logic Analyzer that allowed us to graphically observe what will happen during communication based on pulse trains and hexadecimal codes associated with the pulses.

- Also, while developing the sensor software, the datasheet used was not very clear regarding the order of steps to be followed in the configuration and also problems with the fifo write and fifo read pointers as it was not mentioned how to use them properly. So we had to do some testing where we sent certain configurations and then read the registers to verify that it was configured correctly until we found the correct order of register configuration. The photos and codes from more advanced GBIO students were very helpful.
- Regarding the I2C protocol, we observed a lack of synchronization in the communication when sending the address and the register data to the sensor and conceptual problems of the protocol. As a solution we turned again to NXP forums, websites and GBIO example projects to guide us.
- When we wanted to use the WiFi module we found that it produced current peaks that exceeded 200 mA so, as a solution, the demanded current had to be offered by a voltage regulator whose output was much higher than the evaluated current peaks. Also placing a 100uF capacitor at 50V was an effective solution to dampen the spikes.
- When programming the WiFi module we encountered problems understanding the communication pattern with the WiFi module, the handling and order of AT commands, the use of blocking functions for receiving the “OK\r\n” and for transmission, as well as problems establishing adequate delays between one message and another. To solve this we use a state machine to handle the reception of the “OK\r\n” and a queuing system or Queue to transmit the size of the message and the message itself.

BENEFITS FOUND IN THE DEVELOPMENT OF THE PROJECT

As benefits obtained from the project we can identify:

- Learning and incorporation of programming knowledge in C++ and OOP in embedded systems to develop a prototype device and be able to visualize the effects of our programming.
- Understanding how to create our own applications and drivers or code libraries.

- Integration of interdisciplinary content from different subjects such as ASyS, INFO1, INFO2 and Representation Systems, Embedded Systems, Hardware Design and Signal Analysis.
- Academic and technical growth.
- Understand the importance of optimizing hardware and software resources in biomedical devices.
- Solving complex problems.
- Development of skills to face and solve problems in real time, both in hardware and software.
- Improve group work, communication and task organization skills.
- Understand the practical challenges of biomedical systems, which broadens the vision towards real applications in IoT and bioengineering.

CONCLUSION

We can establish the following conclusions from the project:

- The project met the objectives of developing a functional pulse oximeter prototype.
- Successful integration of hardware, software and wireless communication for real-time visualization of biomedical parameters.
- Overcoming challenges related to I2C communication and current spikes in the ESP8266 module.

- Implementation of robust solutions, such as state machines and PCB design optimization.
- This project lays the foundation for future developments in accessible and modular biomedical devices.
- Potential for continued work in areas such as more advanced interfaces, improved measurement accuracy and optimization of energy consumption.

ANNEX

- Lucidchart
- NXP Forum

https://community.nxp.com/t5/forums/searchpage/tab/message?advanced=false&advanced=false&allow_punctuation=true&allow_punctuation=true&q=LPC845&q=LPC845

- INFO2 virtual class

https://aulasvirtuales.frba.utn.edu.ar/course/view.php?id=15301

- I2C Guide

How to write a microcontroller driver for an I2C device? (MSP430 and VL6180X)

A Basic Guide to I2C
Understanding the I2C Bus
I2C Communication Protocol - GeeksforGeeks
I2C-bus specification and user manual
Understanding I2C
I2C For Beginners: Device Addresses from Data Sheets

- ESP8266 and TCP Guide

AT Command Set — ESP-AT User Guide documentation
ESP8266 AT Instruction Set
TCP protocol: definition and operation
What is TCP (Transmission Control Protocol)? - GeeksforGeeks
https://www.ionos.com/es-us/digitalguide/servidores/know-how/que-es-tcp-transport-control-protocol/?srsltid=AfmBOor_nT8KNcOoOwQFfQeYSDmm9i1DfOYVijV4Dj-I1e1-Apa7k2Ne