Project Report[1]

Smart Sensor for temperature and oxygenation level measurements of a patient.

Authors	**Ceglia Salvatore** **Ferrara Luigina** **Gargiulo Anna** **Kárason Halldór**
Group Number	03
Report Delivery Date	June 23, 2022
Document Version	1.0

[1] This work is licensed under a Creative Common Attribution 4.0 International License

Summary

Project Work

Project Description

The aim of the project is to realize a smart sensor that allows to measure the temperature and the blood oxygenation levels of a patient: the blood oxygenation and the temperature should be shown on a display that allows the patient to read the values. The sensor should also send information to a portable computer where information about the timestamp of when the measurements are taken and the measurements themselves are visible. If the values are considered not normal, an alarm has to be shown on the display.

Requirements

The information about the blood oxygenation levels and the temperature of the patient needs to be aggregated: the samples collected in a minute need to be averaged and shown on the display.

If the values measured for the blood oxygenation levels are lower than a configured threshold and the temperature of the patient is higher than a configured threshold, then an alarm should be shown on the display.

Periodically, the measures need to be sent through USB to the Portable Computer, with a timestamp of when the measurement was taken.

Standards

The thresholds for the blood oxygenation levels and the temperature of the patient must be configured according to Covid19 regulations:

- The temperature of the patient should be below 37,5 °C;
- The blood oxygenation values are considered normal if they're in the range 94-100% . If the values are below 94%, medical care should be sought; if the values are below 90% the Covid-19 infection is considered severe[2].

[2] Source: Living guidance for clinical management of Covid- 19, World Health Organization

Devices

To complete the project different devices are necessary:

- To measure the blood oxygenation levels and the temperature of the patient two different sensors are used:
    1. SparkFun Pulse Oximeter and Heart Rate Monitor;
    2. NTC MF52 3950 Thermistor on the KY-028 Temperature Sensor Module;
- To display the date and time of the measurement the DS1307 Real Time Clock is used;
- The information is shown on a SSD1306 Oled Display.

SparkFun Pulse Oximeter and Heart Rate Monitor

The SparkFun Pulse Oximeter and Heart Rate Monitor is an I²C based biometric sensor. Utilizing two chips from Maxim Integrated, the SparkFun Pulse Oximeter and Heart Rate Monitor has both the MAX30101 biometric sensor and MAX32664 biometric hub. While the former does all the sensing, the latter is a Cortex M4 processor that handles all of the algorithmic calculations, digital filtering, pressure/position compensation, advanced R-wave detection and automatic gain control.

Figure 1: SparkFun Pulse Oximeter and Heart Rate Monitor

The **MAX30101** gets the heart rate (BPM) and blood oxygen levels (SpO2) through the process of photoplethysmography, which is the process of obtaining the aforementioned biometric data with light. The SparkFun Pulse Oximeter works by placing the finger of the patient gently on the sensor, in which it shines red, infrared, and sometimes green light through the skin. The capillaries filled with blood under your skin will absorb this light, or not, and the MAX30101 sensor will read which light comes back. This light data will then be sent back to the **MAX32664** Biometric Sensor Hub which handles all the calculations to determine heart rate and blood oxygen levels. Figure 2 shows how the two are connected: the MAX30101 communicates through I²C with the MAX32664 processor, which communicates with a host through another I²C connection.

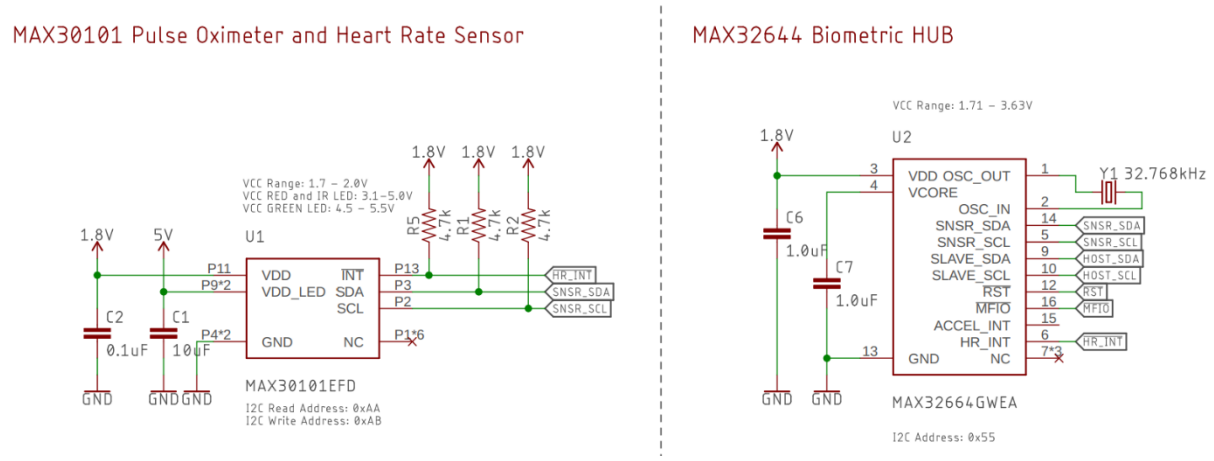

Figure 2: SparkFun Pulse Oximeter and Heart Rate Monitor Schematic

To be able to work with the sensor, it is necessary to use the MAX32664 interface, since the MAX30101 interface is not accessible from the outside. This kind of processor will handle all the calculations if two specific algorithms are enabled on the sensor:

- **Automatic Gain Control (AGC) algorithm:** this algorithm allows to determine automatically the LED currents and pulse with of the MAX30101 sensor;
- **MaximFast algorithm:** this algorithm determines automatically the percentage of the blood oxygenation values without doing any manual computation. The algorithm also allows to measure the number of heart beats per minute.

NTC MF52 3950 Thermistor on the KY 028 Temperature Sensor Module

The sensor measures temperature changes based on thermistor resistance. A thermistor is a type of resistor whose resistance is dependent on temperature, more so than in standard resistors. Thermistors are widely used as inrush current limiter, temperature sensors (NTC type typically), self-resetting overcurrent protectors, and self-regulating heating elements.

The temperature sensor also contains a potentiometer to adjust the detection threshold on the digital interface. If the potentiometer is rotated clockwise the detection threshold is increased; if it is turned counterclockwise the threshold is decreased.

The analog interface returns a numeric value that depends on the temperature and the potentiometer's position.

The module has both an analog signal output and an digital signal output, which is different from analog temperature sensor and other temperature sensor modules.

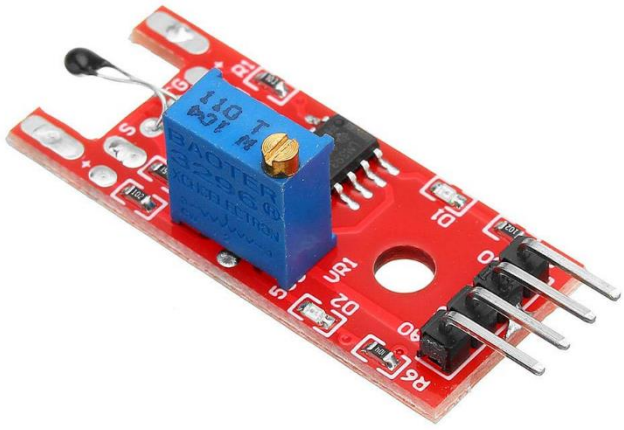

Figure 3: KY028 Temperature Sensor

DS1307 Real Time Clock

The DS1307 serial real-time clock (RTC) is a low power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of battery-backed NV SRAM. Address and data are transferred serially through an I2C, bidirectional bus.

The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator.

The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

An open drain output signal is provided to carry out a programmable clock at 1Hz, 4.096Khz, 8.192Khz, 32.768Khz; this feature is very useful to synchronize your application with a “one second” reference or one of its multiples.

This output can also be fixed at high or low logic level via internal register, allowing you to use this signal for debugging or test routines.

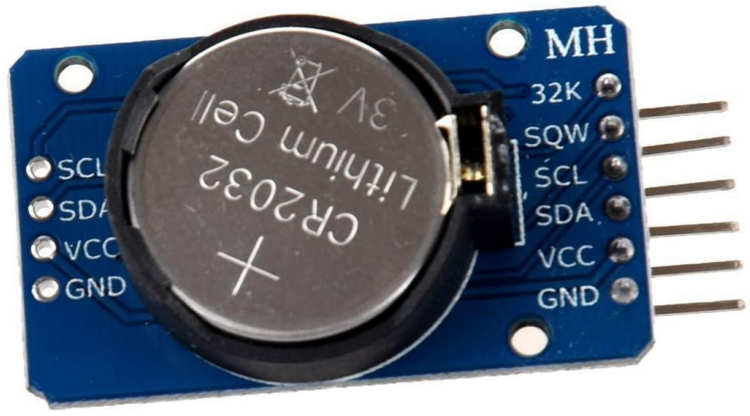

Figure 4: DS1307 Real Time Clock

SSD1306 Oled Display

SSD1306 is a single-chip CMOS OLED/PLED driver with a controller for organic / polymer light emitting diode dot-matrix graphic display system.

It consists of 128 segments and 32 commons. The SSD1306 embeds with contrast control, display RAM and oscillator, which reduces the number of external components and power consumption. It has 256-step brightness control. Data/Commands are sent from general MCU through the hardware selectable 6800/8000 series compatible Parallel Interface, I²C interface or Serial Peripheral Interface. It is suitable for many compact portable applications, such as mobile phone sub-display, MP3 player and calculator, and so on.

The SSD1306 Display is used through its I²C interface. To do so, it's necessary to identify the I²C dedicated pins on the display:

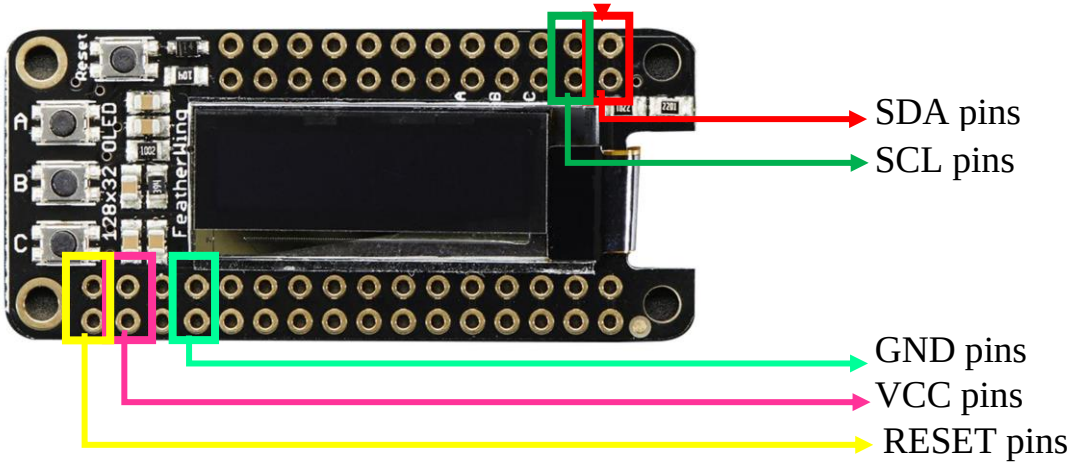

Figure 5: SSD1306 128x32 Oled Display

Solution

Hardware Architecture

The project relies on the STMicroelectronics NUCLEO-64 ARM – CORTEX M4 microcontroller, which communicates with the peripherals described in the *Devices* section to achieve the requirements of the project.

This microcontroller is integrated with a General Purpose I/O that allows to define different modalities for its pins: for the aim of the project the pins are configured to be used as Analog/Digital converters, Output pins and I²C pins.

Since all the devices used communicate through I²C (apart from the temperature sensor), the hardware has been designed to use a single I²C interface located on the microcontroller, specifically the second I²C interface, I²C 2. This is possible since the I²C protocol is designed as an open collector bus. The pins used for the Clock Transmission and Data Transmission are the PB10 and PB3 pins, respectively.

The different devices are equipped with their own specific pins apart from the SCL and SDA pins, so other pins need to be used:

- for the SSD1306 Oled Display, the Reset pin has been connected to the PC13 pin on the board, configured as an Output pin;
- for the MAX32664 pulse oximeter and heart rate monitor, the pins PC2 and PC15 have been connected as Output pins to the Reset pin and MFIO pin of the sensor, respectively;
- the temperature sensor sends data to the microcontroller through the PA0 pin, defined as an ADC1 pin, on channel 0.

The last requirement of the project, i.e. the system has to send data through the USB to the Portable Computer, is fulfilled by enabling the PA2 and PA3 pins of the MCU to be used as UART transmit pin and UART receive pin.

All the devices are connected to a shared ground, and the same stands for their source voltage. The only exception is for the temperature sensor, which needs a higher source voltage compared to the other devices.

After enabling all the necessary pins, the configuration of the device is as it follows:

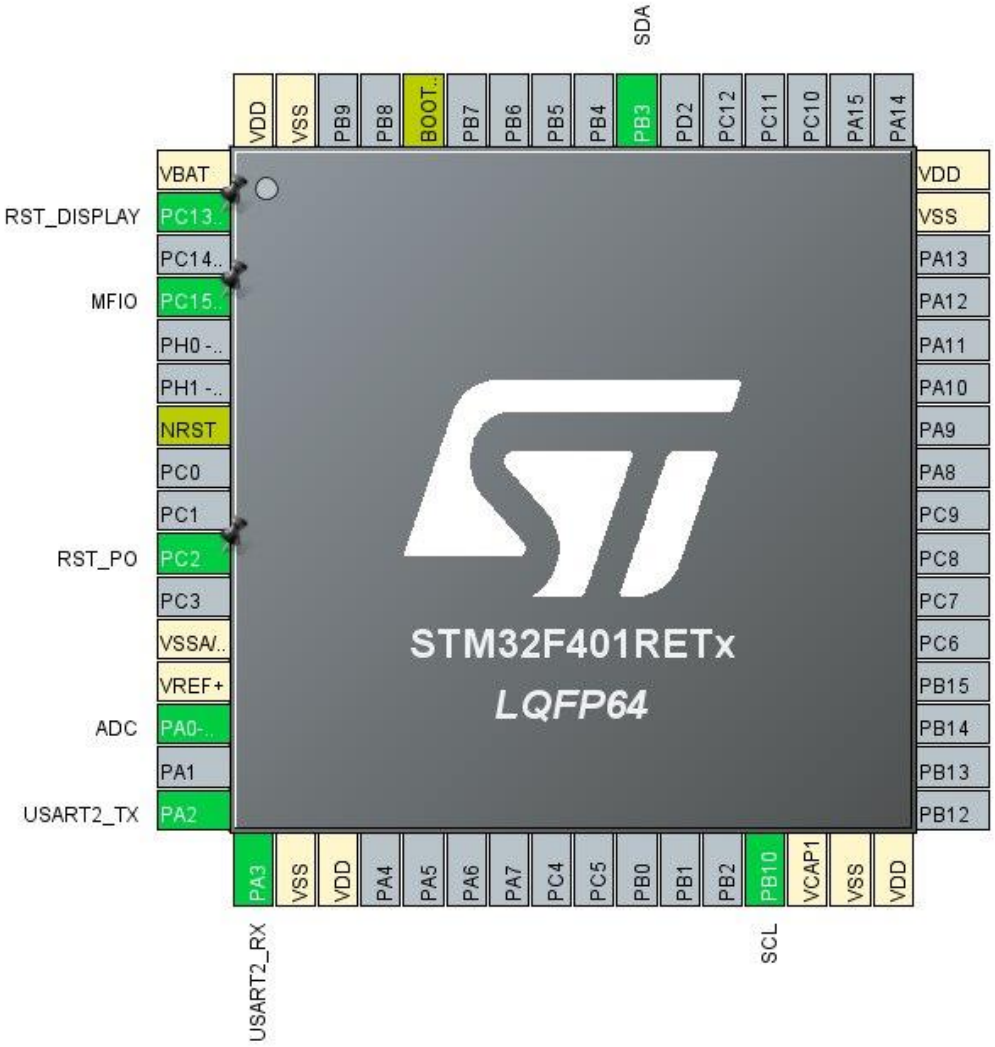

Figure 6: MCU pinout configuration of the project

The devices have been connected using a breadboard, in the following configuration:

Color	Signal
White	5 V
Red	3.3 V
Black	GND
Yellow	I2C SCL
Orange	I2C SDA
Green	RST Oled
Blue	AO Temp
Grey	MFIO PO
Brown	RST PO

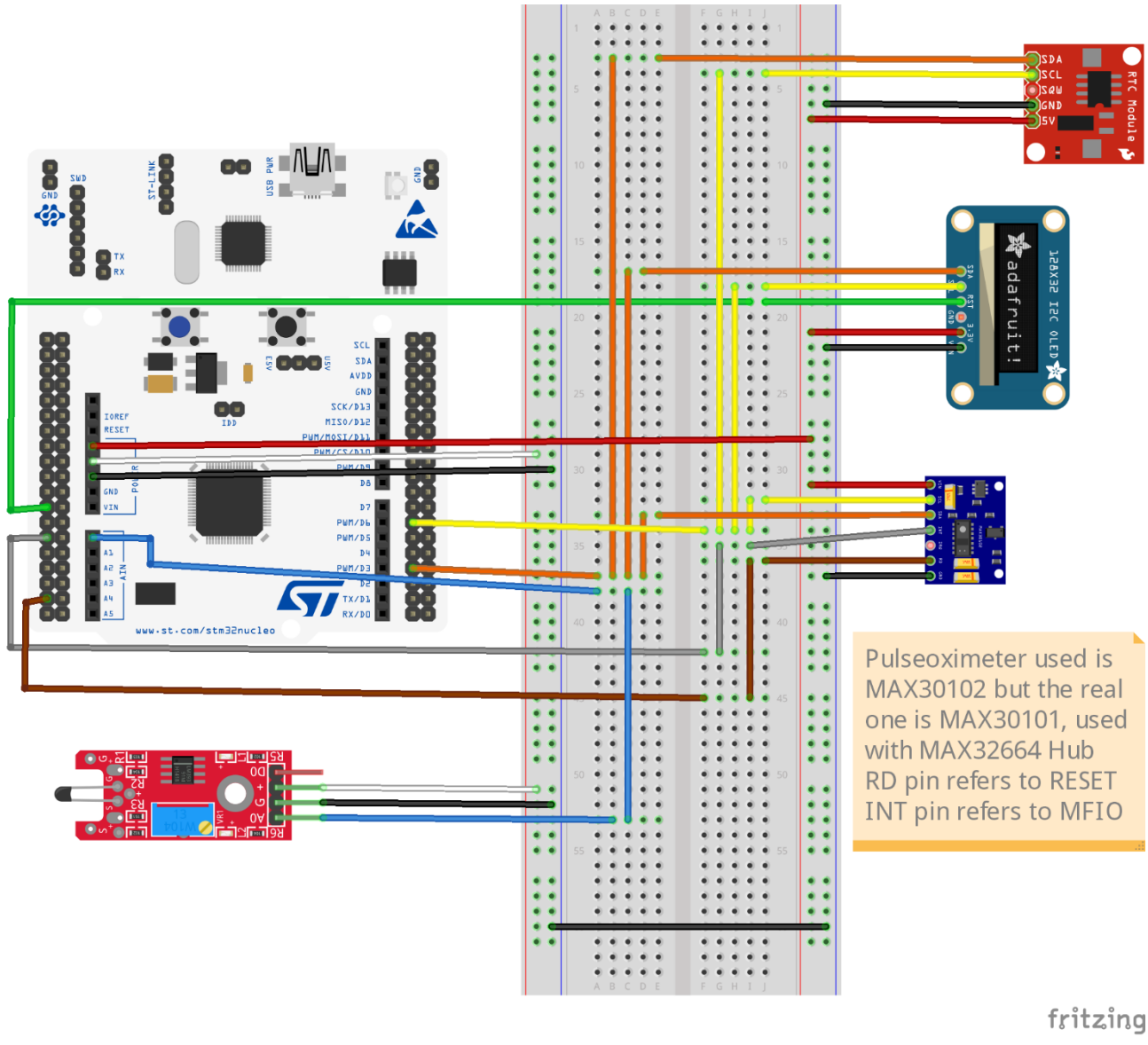

Figure 7: Fritzing schematic of the hardware connections of the project

Software Architecture

As for the software, the programming language used is the C language.

To create modularity in the firmware, the design choices led to the creation of different libraries for the different sensors and actuators to use, as to reproduce the idea of an object-oriented language.

The focus of the project software was to create the drivers that would allow the intended use of the devices, and then interface the different libraries in the core of the software.

The different libraries are interfaced with the main loop, where the different components of the project are initialized, and a timer is defined. The aim of the timer is to print out the information that the patient has to read.

The initialization of the components is arranged in this way:

- Reset of the state of display & initialization of the actuator, which displays a "Loading…" string;
- Initialization of the real time clock, specifying the current date and time;
- Initialization of the temperature sensor;
- Initialization of the pulse oximeter sensor in application mode and configuration of the sensor;

The timer is then started.

Every time the timer elapses, its callback function is executed to handle the interrupt.

The main operations of the application are executed in said callback as the handling of the timer interrupt, so the architecture of the embedded software is *Interrupt Driven*.

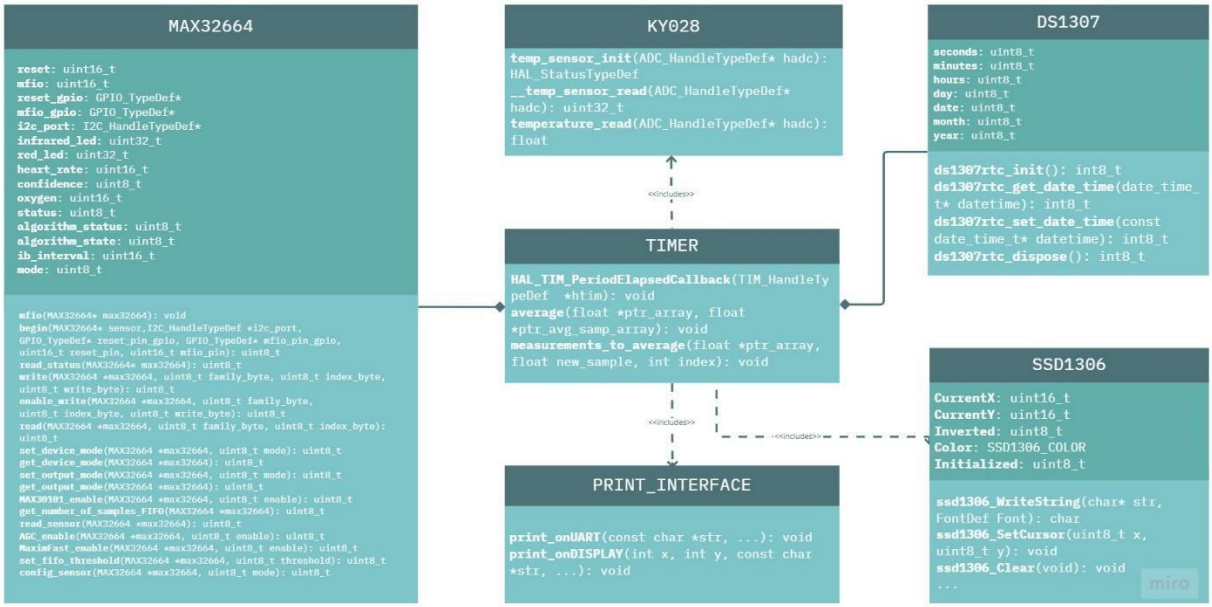

Figure 8: UML Scheme of the Software Architecture

The *Timer* block refers to the Interrupt block of the application, where the main functions are located.

The workflow of the application is shown in the figure at right.

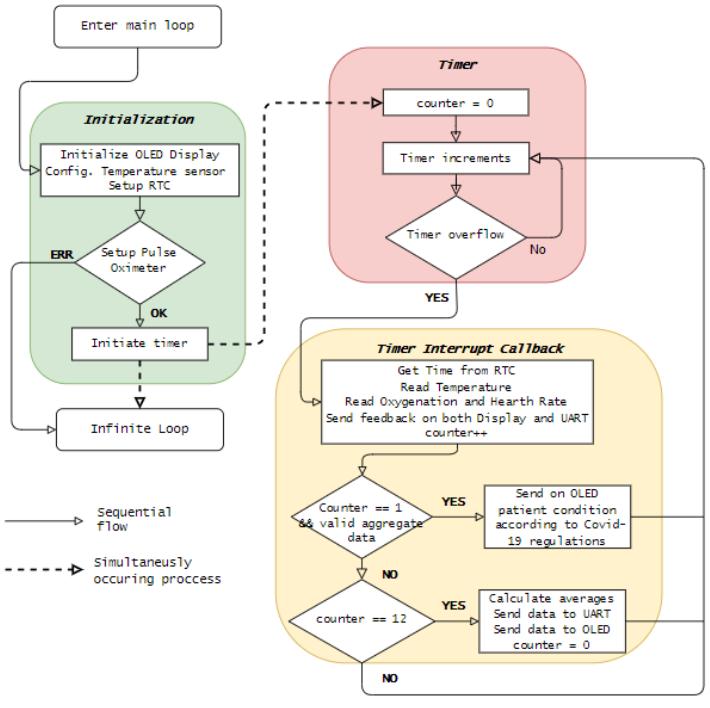

Figure 9: Workflow

Software Drivers

The drivers for the different sensors and actuators of the project were developed starting from the datasheet and user guide of the specific device.

SparkFun Pulse Oximeter and Heart Rate Monitor Driver

The driver for this specific sensor was developed following the MAX32664 User Guide. Since the MAX32664 sensor allows reading information about different kinds of biometric data, it was chosen to create a structure that would allow to store all of this information.

```
struct MAX32664_sensor{
    uint16_t reset;
    uint16_t mfio;
    GPIO_TypeDef* reset_gpio;
    GPIO_TypeDef* mfio_gpio;
    I2C_HandleTypeDef *i2c_port;
    uint32_t infrared_led;
    uint32_t red_led;
    uint16_t heart_rate;
    uint8_t confidence;
    uint16_t oxygen;
    uint8_t status;
    uint8_t algorithm_status;
    uint8_t algorithm_state;
    uint16_t ib_interval;
    uint8_t mode;
};

typedef struct MAX32664_sensor MAX32664;
```

Figure 10: MAX32664 structure definition

For the specific purposes of the project, the sensor had to read blood oxygenation levels. As a plus, the device also reports the heart beats per minute. To do so, apart from the initialization of the device, the focus was to configure the device and read its samples.

To configure the device, it was chosen to:

- Set the device output mode to read both the raw MAX30101 sensor data and the clean algorithm data coming from the MAX32664 processor;
- Set a threshold of 1 free space in the FIFO of the MAX30101 before firing an interrupt;
- Enable the MAX30101 sensor to read the biometric data;
- Enable the AGC algorithm;
- Enable the MaximFast algorithm.

To read from the device, the FIFO was read following the order specified in the User Guide, and all of the information was stored in the MAX32664 structure.

The main functions implemented for the sensor are:

```
/* This function allows to change the mode of the MFIO pin of the MAX32664 from an output pin to an input pin */
void mfio(MAX32664* max32664);

/* The begin function allows to initialize the fields of the MAX32664 structure to non acceptable values.
 * Then, the device is started and initialized to application mode.
 * The function returns OK when all the operations are executed correctly,
 * instead it returns ERROR when the operations fail. */
uint8_t begin(MAX32664* sensor,I2C_HandleTypeDef *i2c_port, GPIO_TypeDef* reset_pin_gpio,GPIO_TypeDef* mfio_pin_gpio,
              uint16_t reset_pin, uint16_t mfio_pin);

/* The read_status functions allows to read the state of the MAX32664.
 * If the operations performed before the call to this functions were executed correctly, the status of the Pulse Oximeter
 * should be SUCCESS.
 * If the operations performed before the call to this functions have failed, then the status field will reflect the error
 * occurred. */
uint8_t read_status(MAX32664* max32664);

/* The write function allows to write in the specific memory location, the necessary command.
 * After the operation is executed, it is necessary to read the status of the MAX32664 to check if the command
 * was executed correctly. */
uint8_t write(MAX32664 *max32664, uint8_t family_byte, uint8_t index_byte, uint8_t write_byte);

/* The write function allows to write in the specific memory location, the necessary command.
 * This function is needed to enable the MAX30101 sensor and to enable the algorithm that compute the necessary values
 * to read the blood oxygenation nad heart beat of the patient.
 * After the operation is executed, it is necessary to read the status of the MAX32664 to check if the command
 * was executed correctly. */
uint8_t enable_write(MAX32664 *max32664, uint8_t family_byte, uint8_t index_byte, uint8_t write_byte);

/* The read function allows to read the specific memory location.
 * After the operation is executed, it is necessary to read the status of the MAX32664 to check if the command
 * was executed correctly. */
uint8_t read(MAX32664 *max32664, uint8_t family_byte, uint8_t index_byte);

/* This function allows to set the device mode of the MAX32664. If the mode parameter is not an acceptable modality
 * then the function returns ERROR. */
uint8_t set_device_mode(MAX32664 *max32664, uint8_t mode);
```

Figure 11: Main Functions of the MAX32664 Driver

```c
/* This function allows to read the device mode of the MAX32664. */
uint8_t get_device_mode(MAX32664 *max32664);

/* This function allows to set the output mode of the MAX32664. If the mode parameter is not an acceptable modality
 * then the function returns ERROR, otherwise it writes the command to the specified location. */
uint8_t set_output_mode(MAX32664 *max32664, uint8_t mode);

/* This function allows to read the output mode of the MAX32664. */
uint8_t get_output_mode(MAX32664 *max32664);

/* This function allows to enable and disable the MAX30101 pulse oximeter and heart rate monitor sensor. */
uint8_t MAX30101_enable(MAX32664 *max32664, uint8_t enable);

/* This function allows to read the number of samples from the MAX30101 sensor. */
uint8_t get_number_of_samples_FIFO(MAX32664 *max32664);

/* This function is used to read the samples from the MAX30101 sensor.*/
uint8_t read_sensor(MAX32664 *max32664);

/* This function allows to enable and disable the Automatic Gain Control algorithm. */
uint8_t AGC_enable(MAX32664 *max32664, uint8_t enable);

/* This function allows to enable and disable the MaximFast algorithm. */
uint8_t MaximFast_enable(MAX32664 *max32664, uint8_t enable);

/* This function allows to specify the number of remaining free slots in the MAX30101 FIFO that are used as a
 * threshold to launch an interrupt from the MFIO pin. */
uint8_t set_fifo_threshold(MAX32664 *max32664, uint8_t threshold);

/* This function is necessary to configure the MAX32664 sensor to read the heart beat and blood oxygenation of the patient.
 * The SENSOR_ALGORITHM_DATA output mode allows to read from the MAX30101 FIFO both the raw values and the values computed
 * from the algorithms enabled on the MAX32664. */
uint8_t config_sensor(MAX32664 *max32664, uint8_t mode);
```

Figure 12: Main functions of the MAX32664 Driver

The driver for the sensor can be found here.

KY 028 Temperature Sensor Driver

For the temperature sensor the driver was written following the ELEGOO Guide for the sensor.

After initializing the sensor, it reads the raw values of the changes in the thermistor resistance. The values are read using the ADC interface of the microcontroller and then translated into temperature values.

The temperature evaluation is done by using the β parameter equation, characterized by the value of the resistance at temperature T = 25 °C, the resistance of the thermistor at the current temperature, and a B parameter (in this case, the value for this parameter is 3950 K).

To adjust the temperature value, the resistance of the potentiometer was changed as well.

The main functions of the driver are:

```
/*The function initializes the digital temperature sensor*/
HAL_StatusTypeDef temp_sensor_init(ADC_HandleTypeDef* hadc);

/*The function reads and returns the raw measure from the ADC*/
uint32_t __temp_sensor_read(ADC_HandleTypeDef* hadc);

/* The function returns the temperature in °C computed from the ADC raw value.
 * The formula that is used to the conversion is the B equation. */
float temperature_read(ADC_HandleTypeDef* hadc);
```

Figure 13: Main functions of the KY028 Driver

The driver for the sensor can be found here.

DS1307 Real Time Clock & SSD1306 Oled Display Drivers

For these devices, the drivers were not directly implemented by the team. For the DS1307 Real Time Clock driver, the material was provided by the Professor of the course; for the SSD1306 Oled Display driver the material used was a library provided by Roberto Benjami on github, that allows to use the display with DMA.

Timer

The main operations of the embedded system are performed every 5 seconds, when the timer elapses.

Every time the timer callback is executed, the samples are read from the real time clock, the temperature sensor, and the pulse oximeter.

The values read are then added to the array of values to average every minute, and additional checks are performed on the values read by the pulse oximeter: if the values read have a confidence below 90% then the samples are discarded, and an additional check is performed to confirm the presence of the finger on the sensor.

Furthermore, the samples are then compared with the standards chosen for the project, and different levels of alarms are fired:

- If the average oxygen levels are below 90%, a critical oxygenation message is displayed;
- If the average temperature is higher than 37.5 °C, a critical temperature message is displayed;
- If the average oxygenation is between 90% and 94% a warning oxygenation message is displayed;
- If the average temperature is in between 37 and 37.5 °C a warning message is shown on the display;

These alarms are shown for 5 seconds if in the previous minute of measurements one of the conditions was true.

Once every 5 seconds, the single values read are shown on the portable computer.

Once every minute, the averaged values are shown on the display (and on the portable computer) for the user to read. To get a complete information, the number of samples used to perform the average is printed on the UART as well. To simulate the minute passing in software, this operation is performed when the timer callback is called 12 times.

The 16-bit timer of the STM32 can count to 65535 before rolling over. For our application the CPU clock was set to 16 MHz, and a prescaler of 16000 was used to set the timer clock to 1kHz.

Software Protocols

The protocols used in the project are:

I²C PROTOCOL

The I²C protocol has been used for the communication between the STM32 microcontroller and the devices used in the project. The microcontroller acts as the master of the communication, and asks the devices to either receive or send data.

UART

For the communication between the STM32 and the portable computer the UART protocol was used, having the STM32 microcontroller transmit the samples read from

the sensors to the portable computer. The baud rate selected for this communication is 9600.

Power Consumption

The specifications for the sensors and actuators used in the project:

SparkFun Pulse Oximeter and Heart Rate Monitor

	Value	Unit of Measurement
Supply Voltage	3.3	V
Supply Current	100	mA
Operating Temperature Range	-40 to 105	°C

Table 1: Specifications of the MAX32664 pulse oximeter and heart rate sensor

NTC MF52 3950 Thermistor on the KY 028 Temperature Sensor Module

	Value	Unit of Measurement
Supply Voltage	3.3 – 5.5	V
Temperature Measurement Range	-55 to 125	°C

Table 2: Specifications of the KY028 temperature sensor

DS1307 Real Time Clock

	Value	Unit of Measurement
Supply Voltage	4.5 – 5.5	V
Supply Current	1.5	mA

Operating Temperature Range	+0 to +70	°C

Table 3: Specifications of the DS1307 real time clock

SSD1306 Oled Display

	Value	**Unit of Measurement**
Supply Voltage	1.65 to 3.3	V
Supply Current	10	mA
Operating Temperature Range	-40 to 85	°C

Table 4: Specifications of the SSD1306 oled display

Some expedients were taken to reduce power consumptions:

the logic of the firmware is embedded in a timer: the team chose to read samples once every 5 seconds, to try and reduce the power consumption without giving up the accuracy of the average shown to the patient.

Index of Figures

Index of Tables