Development of a microcontroller based system for measuring environmental conditions

Oleksandra Sidlovska

Master Thesis Dissertation

Supervisor: Prof. Dr. Joaquim Gabriel Magalhães Mendes
Co–supervisor: CEng. Pedro Bastardo

Master in Mechanical Engineering

Specialization of Automation

June 2018

The work presented in this dissertation was performed at the
Laboratory of Automation, Instrumentation and Control,
Department of Mechanical Engineering
Faculty of Engineering
University of Porto
Porto, Portugal
and at
Bosch Security Systems
Ovar, Portugal.

Oleksandra Sidlovska
E-mail: up201303807@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Mecânica
Rua Dr. Roberto Frias s/n
4200-465 Porto
Portugal

Abstract

The detection of fire in a building uses a fire detectors network that may include several sensors (thermal, optical, chemical) whose signals are constantly analysed by algorithms to reduce the occurrence of false alarms.

This dissertation was performed as a part of collaboration between Bosch Security Systems (ST) – Ovar, a company of German origin specialized in video surveillance, communication and fire detection systems and the Faculty of Engineering of the University of Porto.

The aim of this project is to use the fire detector infrastructure to integrate an environmental monitoring system (temperature, humidity, pressure and VOCs – Volatile Organic Compounds).

It was developed a system based on a microcontroller (ESP8266) connected to a multi–parameter sensor (BME680) using SPI communication and to the fire detector FCP–320 through the base detector MS 400 terminals.

The information acquired by ESP8266 is made available in a website, using an open channel on ThingSpeak platform. When air quality is poor or a fire situation occurs, an extra measurement is performed and a SMS alert is sent to the user.

Finally, a functional prototype was built which included a PCB and a new part manufactured by 3D printing which is attached to the fire detector. Thus in a simple way, the fire network sensors can be upgraded to include environment monitoring.

Keywords: Fire Alarm, Environmental Sensor, ESP8266, BME680

Resumo

A deteção de um incêndio num edifício usa uma rede de detetores de incêndio que devem incluir vários sensores (térmico; ótico; químico) cujos sinais são constantemente analisados por algoritmos para diminuir a ocorrência de falsos alarmes.

Esta dissertação foi realizada no âmbito de uma colaboração entre a Bosch Security Systems (ST) – Ovar, uma empresa de origem alemã especializada em sistemas de videovigilância, comunicação e deteção de incêndio e a Faculdade de Engenharia da Universidade do Porto.

O objetivo deste projeto é usar a infraestrutura do detetor de incêndio para integrar um sistema de monitorização ambiental (temperatura; humidade; pressão e COVs – Compostos Orgânicos Voláteis).

Foi desenvolvido um sistema baseado em microcontroaldor (ESP8266) conetado à um sensor multiparamétrico (BME680) usando a comunicação SPI e ao detetor de incêndio FCP–320 através dos terminais da base do detetor de incêndio MS 400.

A informação adquirida pelo ESP8266 é disponibilizada no *website*, usando um canal aberto na plataforma *ThingSpeak*. Quando a qualidade de ar é má ou quando ocorre uma situação de incêndio, uma medição extra é realizada e o utilizador é notificado através de uma SMS.

Finalmente, foi construído um protótipo funcional que incluíu um PCB e foi fabricada uma peça por impressão 3D que por sua vez foi ligada ao detetor de incêndio. Assim, de uma forma simples, os sensores da rede de incêndio podem ser atualizados para incluir a monitorização do ambiente.

Keywords: Alarme de Incêndio, Sensor Ambiental, ESP8266, BME680

To my Parents and my Sister

“Не піддаватись.”

–Сідловський В. Ф.

Acknowledgements

I would like to express my sincere gratitude to all the people who guided me during this dissertation.

From the Faculty of Engineering of the University of Porto:

To my supervisor, Professor Joaquim Gabriel Magalhães Mendes, for the support, motivation, patience, and knowledge which helped me throughout this entire work.

To all the teachers who shared with me their knowledge.

To all the friends that became part of my life and with whom my time at FEUP was enjoyable.

From Bosch Security Systems Ovar:

To my co–supervisor, Engineer Pedro Bastardo, whose guidance, ideas, knowledge and support have helped to improve this task.

To Engineer Andrea Cannizzaro, for all the help in electronics which was essential to perform this assignment.

To Engineer Carlos Ribeiro, for explaining the operation of the fire detector and its manufacture.

To Engineer Álvaro Matos, for sharing his knowledge of LSN and microcontrollers and for all the support.

To Engineer Joaquim Gomes, for sharing his knowledge of PCB layout and for helping with the PCB construction.

To Engineer Carlos Azevedo, for helping me to understand and implement the C–Point on detector base.

To Engineer Francisco Moreira, for welcoming me to the ENI section from Bosch Security Systems Ovar.

Last but not least:

To my grandfather Volodymyr, my grandmothers Lyudmila and Tamara, my uncle Sergiy, my aunt Oksana and my cousine Rostislav for being always present even with more than 4000 km of distance.

To my friend Jéssica, for the unique friendship and for all the support.

To my sister Margarita, for being my happiness and for cheering me daily.

To my boyfriend Bruno, for giving me his love, support and patience.

To my parents: Larysa and Yevhen, for the support and effort in guaranteeing me the best personal and academic formation.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ABS	Acrylonitrile Vutadiene Styrene
ADC	Analog to Digital Converter
AFDS	Automatic Fire Detection and Alarm Systems
API	Application Programming Interface
APP	Application Software
BIS	Building Integration System
BSEC	Bosch Software Environmental Cluster
BT	Building Technologies
CAN	Controller Area Network
CPU	Central Processor Unit
DAC	Digital to Analog Converter
EEPROM	Electrically–Erasable Programmable Read-Only Memory
FAP	Fire Addressable Photoelectric
FEUP	Faculty of Engineering of the University of Porto
GPIO	General Purpose Input/Output
HTTP	Hypertext Transfer Protocol
I^2C	Inter–Integrated Circuit
IAQ	Indoor Air Quality
IC	Integrated Circuit
IDE	Integrated Development Environment
IIR	Infinite Impulse Response
IoT	Internet of Things
IP	Internet Protocol

ISO	International Organization for Standardization
ISP	Intelligent Signal Processing
JST	Japan Solderless Terminal
JTAG	Joint Test Action Group
LED	Light Emitting Diode
LGA	Land Grid Array
LSN	Local Security Network
MCU	Microcontroller Unit
MMS	Multimedia Messaging Service
NEMA	National Electrical Manufacturers Association
OPC	Open Platform Communications
PC	Personal Computer
PCB	Printed Circuit Board
RH	Relative Humidity
RTC	Real–Time Clock
SMS	Short Message Service
SPI	Serial Peripheral Interface
ST	Security Systems
TCP	Transmission Control Protocol
ULP	Ultra Low Power
URL	Uniform Resource Locator
USB	Universal Serial Bus
VOC	Volatile Organic Compound

Chapter 1

Introduction

This study results from the collaboration between an academic and industrial environment that involves three interested parties: the student, Mechanical Engineering Graduate, the teaching institution, Faculty of Engineering of the University of Porto (FEUP), and the company, Bosch Security Systems (ST) Ovar.

It depicts different tasks performed at FEUP and at Bosch so as to share scientific and industrial information.

1.1 The Bosch Group

The Bosch Group is an important global supplier of technology and services which employs approximately 402,000 associates worldwide with business divided into four sectors:

- Mobility Solutions;
- Industrial Technology;
- Consumer Goods;
- Energy and Building Technology [1].

The Bosch Group came to the Portuguese market in 1911 and has currently five locations: two in Lisbon, Braga, Ovar and Aveiro. Bosch Building Technologies – Ovar belongs to the Building Division which produces video cameras, monitors, digital recorders and accessories for security systems.

Figure 1.1. Bosch Security Systems S.A., in Ovar [2]

Seeing that Bosch is a leading Internet of Things (IoT) company, it consequently offers innovative solutions for smart homes, smart cities with connected mobility and manufacturing.

Bosch Building Technologies (BT) supplies video surveillance, intrusion, fire–alarm and voice–alarm systems, access control systems and management software [3].

1.2 Project Background

This work is included in an innovative project under a consortium created by Bosch and the University of Porto. It is a project focused on creating solutions to Safe Cities and in Industry 4.0.

1.3 Aim and Objectives

The main objective of this work is to implement an environmental sensor inside the fire detector, in order to monitor the air quality and detect the fire simultaneously. The environmental sensor indicates when the air quality is poor and consequently prevents health problems and help increase people productivity. To achieve this objective, several intermediate tasks were defined:

- Study fire detectors and their interfaces;
- Selection and purchase of the components for the project – microcontroller and environmental sensor;
- Programming the microcontroller and design a Printed Circuit Board (PCB);
- Communication with a website;
- Design of a mechanical part to contain the system;
- Final assembly of the prototype and test.

1.4 Thesis Outline

Chapter 2: *State-of-the-art* according to Smart Cities from Bosch which contain Smart Buildings, describes the existing fire alarm systems and the most important standards.

Chapter 3: *Architecture of the Proposed System* presents the used architecture, from the proposed solution to the final solution, justifying the relevant options that were taken. All the used components and the connections between them are covered.

Chapter 4: *Software Design* presents the implementation of the programming code and the construction of the different functionalities. Moreover, sending information to a web page and alert messages are defined.

Chapter 5: *Prototype Development* The main stages of prototype construction are described: creating a PCB and a new part where it can be placed.

Chapter 6: *Conclusions and Future Works* presents the conclusions of this thesis as well as suggested future works.

Chapter 2

State-of-the-art

This State-of-the-art reviews the main issues concerning the automatic detection of fire in Smart Buildings. Moreover, it considers the most significant standards referring to automatic fire detectors. Finally, some products related to automatic fire detection are mentioned to offer a view of current market offers.

2.1 Smart Buildings

The Connected Building Solution joins building equipment, like fire, intrusion alarm and access control systems which are inside commercial buildings. Bosch provides different solutions for buildings in order to guarantee internal networking connection, security and safety.

Fire Alarm System assures safety of people and their property. Nowadays, buildings have a lot of technical infrastructures which increase electromagnetic pollution and its impact on a fire detector is often unknown. The AVENAR 4000 has an eSMOG (electromagnetic emission) feature which provides robustness against electromagnetic pollution and prevents false fire alarms. The causes of electromagnetic interference can be: defects in installed devices like lamps and loudspeakers; incorrect installation of electrical devices; unsuitable cable route (especially in older buildings) and electromagnetic radiation in frequencies which have not been tested. Consequently AVENAR's hardware and software contains measures to avoid these electromagnetic interferences [4].

Bosch has high–quality Fire Alarm System which includes panels and peripherals to provide flexibility and early fire detection such as intelligent detectors, interface modules, manual call points and sounders.

FPA–5000 Modular Fire Panel has modular configuration and allows easy extension, thus it is possible to be connected to 32 Panel Controllers, Remote Keypads and OPC (Open Platform Communications) servers, also connected to the Building Integration System (BIS).

The 5000 Series is certified according to current European standards including the latest updated version of EN54–2 and EN54–4. EN54 stipulates the requirements, methods of testing and performance criteria for every component of fire detection and fire alarm systems installed inside and outside buildings. Standard EN54–2 relates to the controlling and indicating equipment while EN54–4 concerns the power supply equipment.

By using external CAN (Controller Area Network) and Ethernet interfaces, interconnecting numerous Panel Controllers and Remote Keypads is possible. The combination of CAN and Ethernet enables the use of more flexible network topologies with greater

number of panels. Furthermore, the Ethernet interface makes possible the connection to a Building Management System like BIS, via an OPC server.

The Modular Fire Panel 5000 Series can be equipped with multiple interfaces for an extended range of applications which can be interfaced to voice evacuation and fire monitoring systems and Ethernet/OPC can be integrated in building management including third party systems.

The Local Security Network (LSN) bus system by Bosch, joins fire and intrusion alarm systems which enables the network be set up as specified: loop, tee–off or a mix of both configurations. Therefore, the digital transmission is bidirectional, allowing sensors and control panels to exchange data among them. Information flows between the control panel and the LSN elements via a single two–wire connection which also supplies power to the detectors. All devices are initialized, controlled and identified from the control panel which displays precise and easy to read information on each detector [5].

2.2 Standards

The subject of fire in buildings, particularly in high buildings and public locations such as hotels, shopping centres, schools and theatres, or even industrial areas, has given engineers significant motivation to develop fire detectors. Currently, standards concerning fire detection and fire alarm systems have been developed to guarantee the correct installation, maintenance and inspection of this essential equipment.

This section mentions the most important standards regarding fire detectors, namely, European Standard EN54 by CEN – European Committee for Standardization and portuguese Technical Note N°12 by Autoridade Nacional de Proteção Civil.

The EN54 whose title is "Fire Detection and Fire Alarm Systems" is divided in following parts:

1	Introduction.
2	Control and indicating equipment.
3	Fire alarm devices – Sounders.
4	Power supply equipment.
5	Heat detectors – Point detectors.
7	Smoke detectors – Point detectors using scattered light, transmitted light or ionization.
10	Flame detectors – Point detectors.
11	Manual call points.
12	Smoke detectors – Line detectors using an optical light beam.
13	Compatibility assessment of system components.
14	Guidelines for planning, design, installation, commissioning, use and maintenance.
15	Point detectors using a combination of detected phenomena.
16	Voice alarm control and indicating equipment.

17 Short-circuit isolators.

18 Input/output devices.

19 Aspirating smoke detectors.

21 Alarm transmission and fault warning routine equipment.

22 Line-type heat detectors.

23 Fire alarm devices – Visual alarms.

24 Components of voice alarm systems – Loudspeakers.

25 Components using radio links.

26 Point fire detectors using carbon monoxide sensors.

27 Duct smoke detectors.

28 Non-resettable line-type heat detectors.

29 Multi-sensor fire detectors – Point detectors using a combination of smoke and heat sensors.

30 Multi-sensor fire detectors – Point detectors using a combination of carbon monoxide and heat sensors.

31 Multi-sensor fire detectors – Point detectors using a combination of smoke, carbon monoxide and optionally heat sensors [6].

The Techinical Note N°12 defines the Automatic Fire Detection and Alarm Systems (AFDS) which are a technical installation that registers the beginning of the fire without human intervention and then transmits the information to the signalling and controlling centre. The AFDS ensures the safety of the people who are in the building and it has the components presented in **Fig. 2.1** [7].

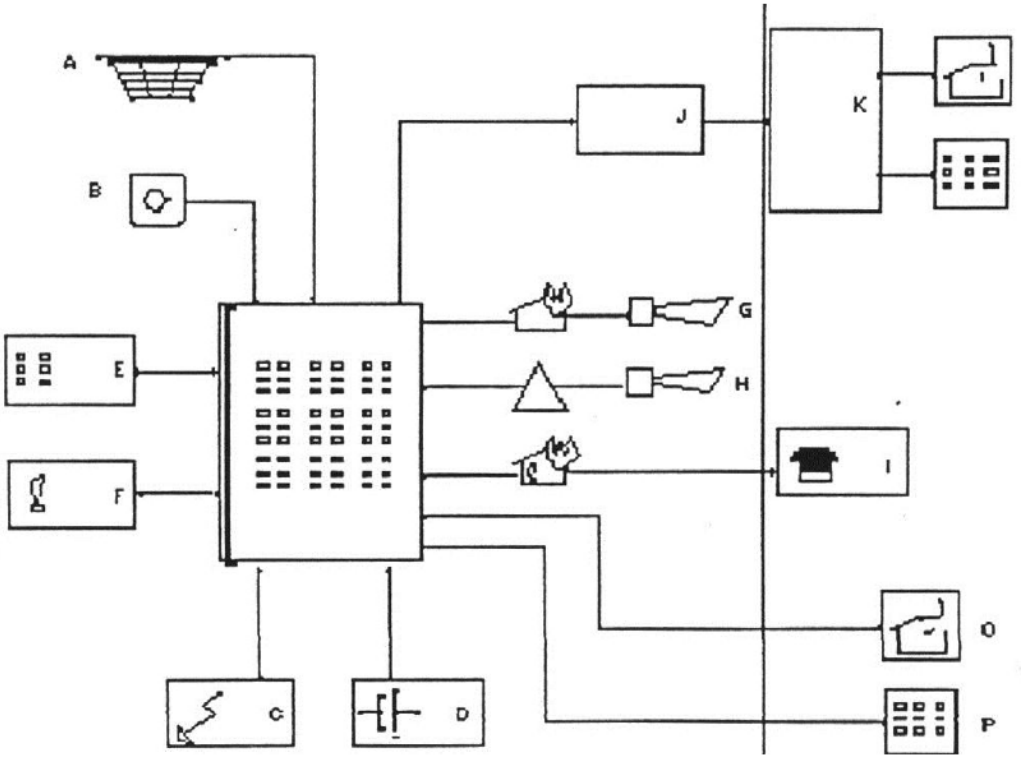


Figure 2.1. Automatic Fire Detection and Alarm Systems [7]

A	Automatic Detector
B	Manual Detector
C	Primary Power Supply
D	Secondary Power Supply
E	Panel
F	Alarm Organisation
G	Intern Alarm
H	Internal Signalling – Failure
I	External Alarm – Fault Signal
J	Interconnection
K	Building Automation
L	Commands in case of fire
M	Remote Signalling
O	Commands in case of fire
P	Remote Signalling (controlled directly by the detection Centre)

2.3 Fire Detectors

This section mentions three Bosch fire detectors available in the market: Twinguard by Bosch Smart Home which is interesting because of its Wi–Fi, AVENAR and FCP–320 which are used in industrial applications.

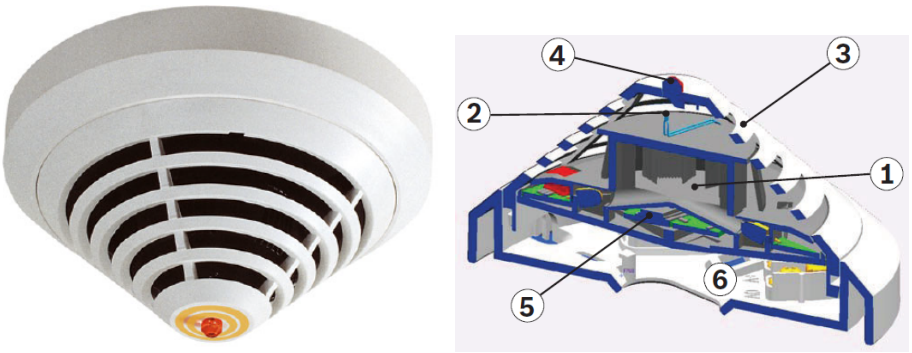
2.3.1 Twinguard by Bosch Smart Home

Bosch Smart Home provides Twinguard which is a smoke detector with an air quality sensor and Wi–Fi connection to be used at home (**Fig. 2.2**). It provides warnings in the event of smoke or impure air and has a connection to the smartphone by the Twinguard app (application software). The air quality sensor BME680 measures temperature, relative humidity and the concentration of Volatile Organic Compounds (VOC) indoors. It is powered by alkaline batteries that normally last for two years [8].

Figure 2.2. Twinguard [8]

2.3.2 AVENAR 4000

FAP–425 (Fire Addressable Photoelectric) or AVENAR 4000 features an excellent accuracy and swiftness in fire detection (**Fig. 2.3**). Its strength against electromagnetic pollution and information about dangerous environmental conditions allow the system to recognize and resolve this situation much quicker thus saving time and money [5].



(a) FAP–425–DOTC–R **(b)** AVENAR 4000 Components

Figure 2.3. AVENAR 4000 [9]

The represented components are:
1 – Smoke measurement chamber with optical sensor.
2 – Thermal sensor.

3 – Chemical sensor.
4 – Individual display with two colour LEDs (Light Emitting Diode) (red and green).
5 – PC (Personal Computer) board with evaluation electronics.
6 – MS 400 / MS 400 B Detector Base.
This fire detector is powered up by the LSN bus and the communication to the fire panel is also made by LSN.

2.3.3 FCP–320

This automatic fire detector has optical, thermal and chemical sensors and an intelligent evaluation electronics which continuously analyses all sensor signals (**Fig. 2.4**). It prevents false alarms because of good speed and accuracy of detection such as AVENAR. The biggest difference between these two detectors is that the FCP-320 does not use the LSN system and is not programmed, it is powered up with 24 V DC, whose operating voltage is 8.5 V DC – 30 V DC. There are two variant with 820 Ω alarm resistor and 470 Ω alarm resistor.

Figure 2.4. FCP-320 [10]

The last two fire detectors are the most interesting for industrial applications, specially AVENAR 4000 because of its technology, therefore it will be described in more detail in next chapter.

2.4 Summary

This chapter presented some of Bosch fire alarm system which detects fire efficiently, its components and how it works.

The following **Chapter 3** describes the used fire detector as well as the other components used in this work and their hardware and software connections.

Chapter 3

Architecture of the Proposed System

This chapter explores the architecture used in this project. The solution is based on the use of a microcontroller (μC) that communicates with an environmental sensor, indicates the state of the fire detector and sends the information to a website. The following sections describe the components and define the connections among them.

3.1 AVENAR 4000

This section describes in detail the AVENAR fire detector, its components and functionalities. It includes three different sensors:

- **Optical sensor** (smoke sensor) uses the scattered–light method, that is, a LED transmits light to the measuring chamber. If there is a fire, smoke enters the measuring chamber and the ligt is scattered by the smoke particles and hits the photo diodes which convert the quantity of light into a proportional electrical signal [9]. The Dual–Ray technology consists of using two optical sensors with different wavelengths – one infrared and one blue (**Fig. 3.1**). This technology allows to distinguish real smoke from disturbances such as light smoke, steam or dust, by comparing the intensity of scattered light emitted by various particles, resulting in less false alarms because of better size differentiation [5].

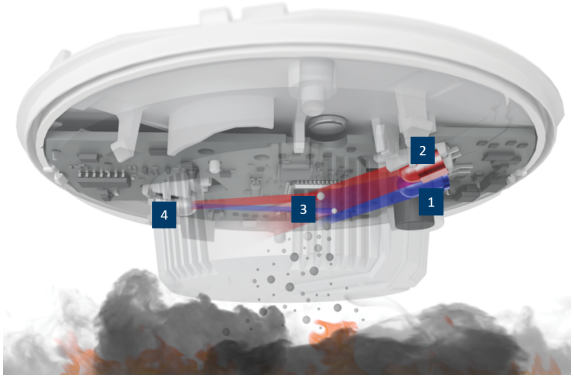


Figure 3.1. Dual Ray Technology [5]

The represented components are:

1 – Blue LED.
2 – Infrared LED.
3 – Scattered light.
4 – Photo Diode.

- **Thermal sensor** (temperature sensor) is a thermistor. An alarm is triggered when the temperature is higher than 69 °C (thermal maximum) or if the temperature reaches a defined value within a specified time (thermal differential).
- **Chemical sensor** (CO gas sensor) detects the carbon monoxide (CO) which results from fire, hydrogen (H) and nitrous monoxide (NO). The sensor signal value and the concentrarion of gas are proportional [9].

The sensors are self–monitoring and the fire panels indicate the following errors:

- Fault indication in the event of the failure of the detector electronics.
- Continuous display of the contamination level during service.
- Fault indication if heavy contamination is detected (in the place of false alarm).

The integrated dividing elements maintain the functional security of the LSN loop in the event of wire interruption or short–circuit. In the event of an alarm, the fire panel receives the individual detector identification. The detector alarm indication is a red flashing LED which is easily 360 ° visible [5].

Intelligent Signal Processing (ISP) contributes to faster recognition of fires and minimises the occurrence of false alarms because all sensor signals are processed continually by smart algorithms using neuronal network (**Fig. 3.2**) [4].

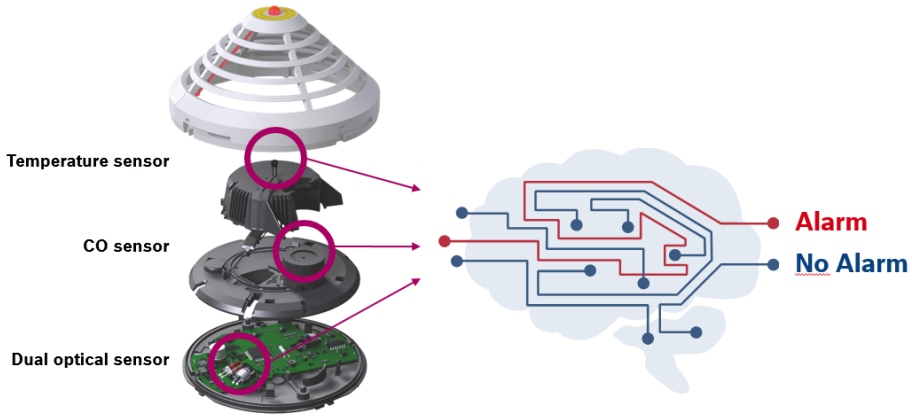


Figure 3.2. Intelligent Signal Processing [4]

Fig. 3.3 shows the block diagram of the fire detector which has a LSN chip, microcontroller, sensors, LED, C–Point and power supply.

The LSN chip functions are:

- Power supply;
- Handling of rotaries which address the detectors;

- Communicate with the microcontroller;
- Communicate with the Panel;
- Handling of indicator LEDs [11].

The versions for automatic and manual address setting have three rotary switches on the bottom of the detector. These rotaries are used to select automatic and manual address allocation with or without auto–detection. The versions without rotaries are automatically addressable only [9].

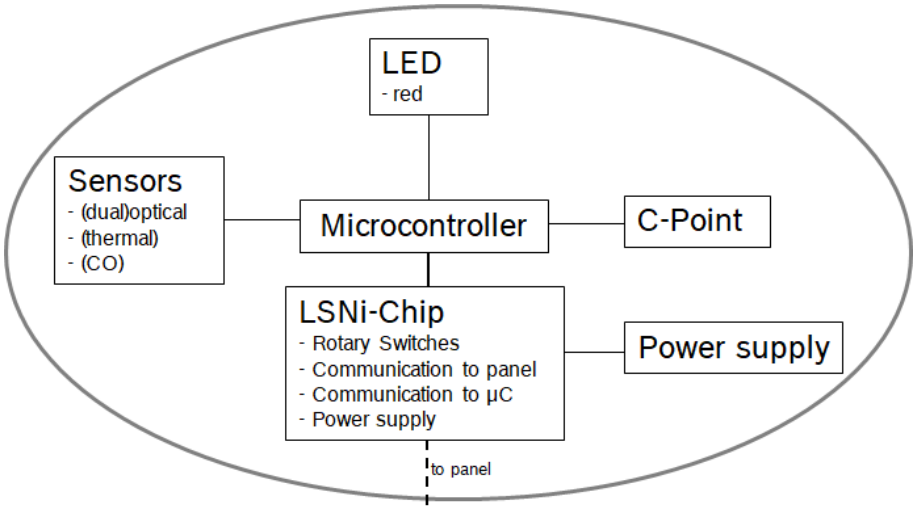


Figure 3.3. Block Diagram [11]

3.1.1 Microcontroller MSP430

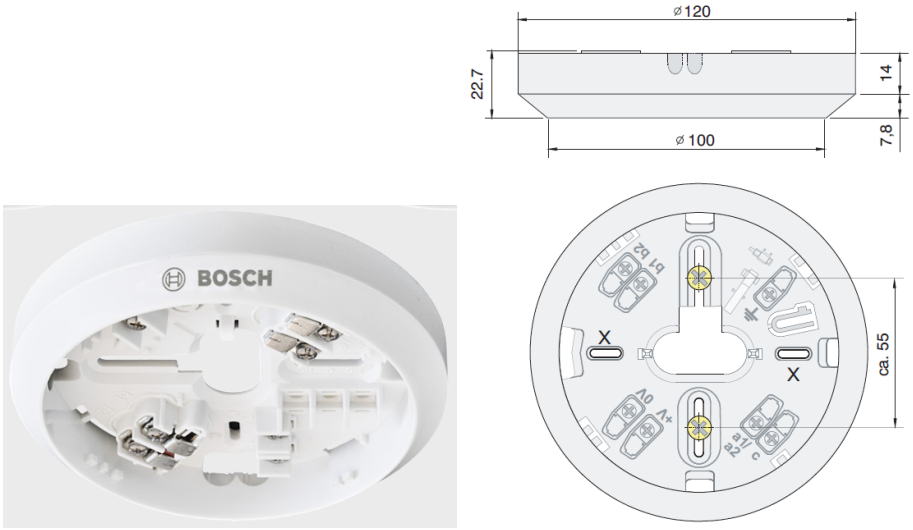
The microcontoller inside the AVENAR is a MSP430F2272 from Texas Instruments. It is an ultra–low power microcontroller which was introduced in the late 90s. Its typical application is on fire and smoke detectors because of it ultra–low power aspects, cost optimization and small size [12].

This microcontroller is programmed using a 38–pin Target Development Board: MSP–TS430DA38. This Development Board uploads the programme through the Joint Test Action Group (JTAG) interface or the Spy Bi–Wire (2–wire JTAG) protocol [13].

3.1.2 Detector Base MS 400

The fire detector head is installed in the Detector Base MS 400 which is used in surface–mounted and flush–mounted cable feed (**Fig. 3.4**).

This base has seven terminal screws which allow the connection to the fire detector: $a1/a2$, $b1$, $b2$, c, 0V, +V and ground. The power supply connections are 0V and +V, the LSN contacts are $a1/a2$, $b1$ and $b2$. Additionally, there is the C–Point connection which consists of measure the voltage between terminals $b1$ or $b2$ and c [14].



(a) MS 400 **(b)** Technical Drawing of MS 400

Figure 3.4. MS 400 [14]

3.2 Architecture change

The initial idea was to use the AVENAR microcontroller to communicate to the environmental sensor BME680 and send its data to a website, including the data from its sensors.

AVENAR is programmed via a PC connected to the fire panel through FSP–5000–RPS (Remote Programming System) and powered via LSN–bus which is in the fire panel. This system makes it possible to programme the sensors individually, send commands and execute different functions. Unfortunately, the Modular Fire Panel and the programming system were not available to perform this task.

Another problem concerns the communication protocol LSN, that is Bosch proprietary, that makes difficult the connection with the environmental sensor.

To overcome these problems, it was decided to use the conventional fire detector FCP–320 and ESP8266 microcontroller which already contains a Wi–Fi module. This conventional detector is powered by 24 V DC source. To supervise the FCP–320 state, the so called C–Point in the detector base is used. The voltage between $b1$ or $b2$ and c is 0V in normal use and it increases to 24V in case of an alarm. The microcontroller communicates to the environmental sensor and reads the fire detector state using C–Point. **Fig. 3.5** shows the defined architecture.

The following sections describe the components: the conventional fire detector, the microcontroller and the environmental sensor.

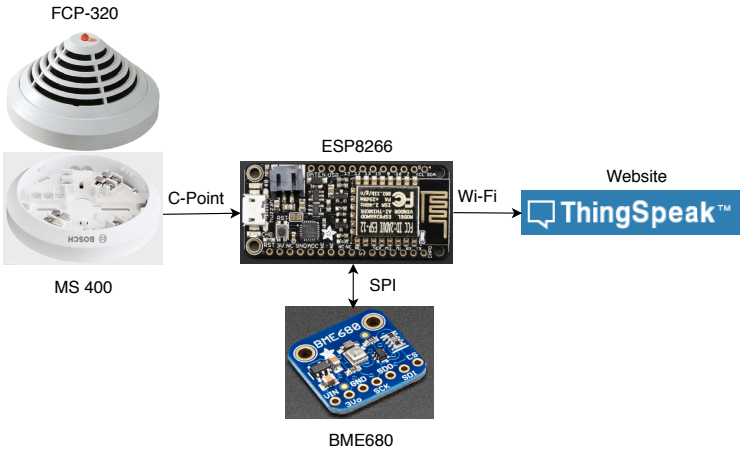


Figure 3.5. Proposed solution

3.3 FCP–320 / FCH–320 Conventional Automatic Fire Detector

As previously mentioned, FCP–320 is a conventional fire detector which combines three different sensors. An alarm is automatically triggered if a sensor signal combination corresponds to a fire condition. These detectors work perfectly in areas with light smoke, dust or steam. The FCP–320 series has three sensors (optical, thermal and chemical) equal to AVENAR sensors.

Detectors FCP–OC–320, FCP–O–320, FCP–OT–320 and FCH–T320 have both models with 470 Ω and 820 Ω [10].

3.4 Adafruit Feather HUZZAH ESP8266

The Adafruit Feather HUZZAH ESP8266 is an Arduino compatible Wi–Fi development board and it has 32 MBit of memory (**Fig. 3.6**). It is clocked at 80 MHz and the power supply is 3.3V. The ESP8266 can be used to control devices with two types of communication: Inter–Integrated Circuit (I^2C) and Serial Peripheral Interface (SPI). The Adafruit Feather HUZZAH ESP8266 will be called as ESP8266 hereforth.

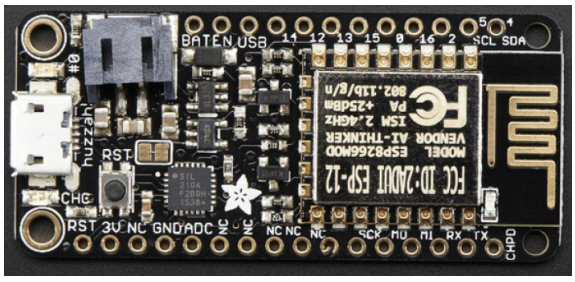


Figure 3.6. ESP8266 [15]

There are two different ways to power this feature: micro USB (Universal Serial Bus) cable or 4.2/3.7 Lithium Polymer (Lipo/Lipoly) or Lithium Ion (Lilon) battery to the Japan

Solderless Terminal (JST) jack. The board converts the 5V USB down to 3.3V. When the ESP8266 is powered by the USB and the battery is connected, it automatically switches to USB power and charges the battery at 100mA. **Appendix A.1** shows the ESP8266 pin–out configuration and its schematic.

The ESP8266 is programmed with Arduino IDE (Integrated Development Environment). There are some rules to properly install the ESP8266 Board Package into Arduino IDE software so it is essential to define some configurations such as Central Processor Unit (CPU) Frequency at 80MHz, Upload Speed at 115200 and install the right USB driver for the CP2104 USB–to–Serial chip [15].

3.5 BME680 – Environmental Sensor

The BME680 is a digital 4 in 1 sensor which measures gas, humidity, pressure and temperature (**Fig. 3.7**). The sensor module is inside a compact metal–lid Land Grid Array (LGA) package with the dimensions 3.0 mm x 3.0 mm x 0.93 mm. The main supply voltage range is between 1.71 and 3.6V. Some of its applications are indoor air quality, home automation and control, IoT and weather forecast [16].

Figure 3.7. BME680 [17]

It has three different power modes: sleep, normal and forced. In normal mode, the sensor changes automatically between a measurement and a standby period. In forced mode, the sensor executes a single measurement on request and then goes back to sleep mode. This mode is used in applications that need a low sampling rate. **Tab. 3.1** shows its current consumption. The existing digital interfaces are I^2C (up to 3.4 MHz) and SPI (3 and 4 wire, up to 10 MHz) [16].

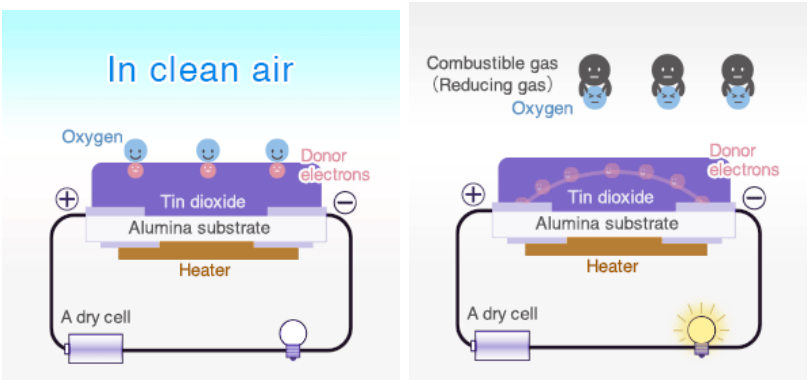
Table 3.1. Current consumption of BME680 [16]

Current consumption	Conditions
2.1 μA	at 1 Hz humidity and temperature
3.1 μA	at 1 Hz pressure and temperature
3.7 μA	at 1 Hz humidity, pressure and temperature
0.09–12 mA	for p/h/T/gas depending on operation mode
0.15 μA	in sleep mode

The measurement period includes temperature, humidity and pressure measurement with selected oversampling. Additionally, it contains a heating phase for the gas sensor hot plate to a target temperature, typically between 200 °C and 400 °C, then it is maintained for a certain duration of time to measure the gas sensor resistance. The measurements of pressure and temperature pass through the Infinite Impulse Response (IIR) which filters a short–term disturbances caused by external factors such as blowing into the sensor or slamming a door. In case of humidity and gas sensors, this filter is not needed because these values do not oscillate quickly.

The gas sensor is a metal oxide–based which detects VOCs by adsorption and then oxidation/reduction on its sensitive layer. It detects the VOCs from paints, lacquers, paint strippers, cleaning supplies, furnishings, office equipment, glues, adhesives and alcohol. The output signal is a resistance value which changes with the variation of VOC concentration: the higher the concentration of reducing VOCs, the lower the resistance and vice versa. It means that if air quality is bad, the concentration of VOCs is high so the resistance will decrease. The raw signal is transformed to an Indoor Air Quality (IAQ) index by smart algorithms inside Bosch Software Environmental Cluster (BSEC) [16].

When air quality is good, oxygen is absorbed by the surface of the metal oxide and attracts free electrons within this surface, forming a potential barrier. This barrier prevents the flow of electrons and increases resistance (**Fig. 3.8a**). In the presence of VOCs, oxygen reacts with the reducing gases and the potential barrier is not formed. Thus, there is free flow of electrons and resistance decreases (**Fig. 3.8b**).



(a) Clean air – high resistance **(b)** Bad air – low resistance

Figure 3.8. Resistance of metal oxide sensor [18]

BSEC software is engineered to work with the four integrated sensors inside the BME680. It is based on an intelligent algorithm which provides an IAQ output. This output has

values between 0 (clean air) and 500 (heavily polluted air) with a resolution of 1 to indicate and quantify the quality of the air available nearby (**Fig. 3.9**).

IAQ Index	Air Quality
0 – 50	good
51 – 100	average
101 – 150	little bad
151 – 200	bad
201 – 300	worse
301 – 500	very bad

Figure 3.9. IAQ classification and colour–coding [16]

Gas sensor specification

Response time ($\tau_{33-63\%}$) is less than 1 s and power consumption is less than 0.1 mA in ultra–low power mode. Its parametres are inferred by lab measurements under controlled environmental conditions compliant to the International Organization for Standardization (ISO) 16000–29 standard "Test methods for VOC detectors".

Humidity sensor specification

Response time ($\tau_{0\text{-}63\%}$) is approximately 8 s, accuracy tolerance is ± 3% Relative Humidity (RH), hysteresis is ± 1.5% RH, operating range is 0 – 100% RH and the resolution is 0.008 RH.

Pressure sensor specification

Operating range is 300 – 1100 hPa and resolution of output data is 0.18 Pa. As previously mentioned, it is used an IIR filter to decrease the impact of external disturbances.

Temperature sensor specification

Operating range is from -40 to +85 °C and output resolution is 0.01 °C. Temperature data is also filtered by the IIR filter [16].

3.5.1 Adafruit BME680 Development Board

It was used the Adafruit BME680 Environmental Sensor Development Board that includes the Bosch's sensor placed on a PCB with a 3.3 V regulator and level shifting (**Fig. 3.10**). Adafruit BME680 Development Board will be called as BME680 hereafter. This board can be used with 3.3 V or 5 V microcontrollers so it is perfectly compatible with ESP8266. The dimensions are 16.0 mm x 11.0 mm x 2.8 mm [19]. **Appendix A.2** shows the BME680 schematic.

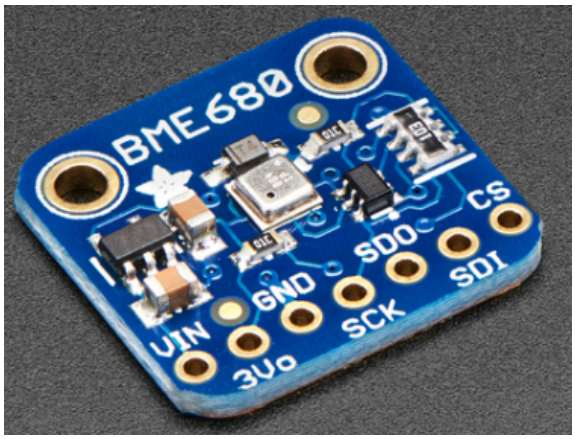


Figure 3.10. Adafruit BME860 [19]

3.5.2 First Calibration

There is an IAQ Accuracy indicator which varies between 1 and 3 (**Fig. 3.11**). This indicator shows the state of the calibration of the sensor which is executed automatically in the background after any reset.

Virtual sensor	Value	Accuracy description
IAQ	0	The sensor is not yet stablized or in a run-in status
	1	Calibration required
	2	Calibration on-going
	3	Calibration is done, now IAQ estimate achieves best perfomance

Figure 3.11. IAQ Accuracy indicator [20]

The first sensor calibration can take several days and it takes several minutes to reach accuracy of 1 for the first time. It is possible to accelerate the calibration process (only for testing purpose) after reaching 1 by manually introducing "bad air" to the sensor for a few seconds. It can be put in a box with an open permanent marker or perfume which contains alcohol. A box with an open permanent marker was used to accelerate the calibration (**Fig. 3.12**).

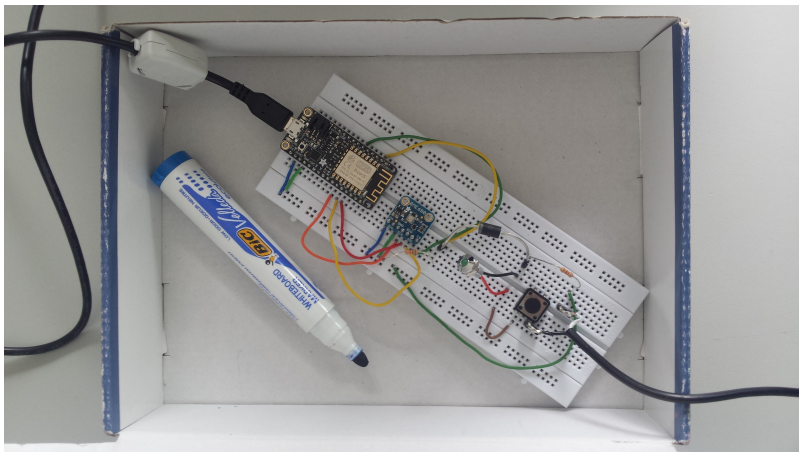


Figure 3.12. First calibration

This method can only be used for evaluation purposes since it is common for the sensor to take a long time to calibrate. Once the sensor has been calibrated for the first time, it takes about 20 minutes to achieve IAQ Accuracy 3 when used after.

3.6 Communication and connection

3.6.1 **ESP**8266 – **BME**680

Both ESP8266 and BME680 have two types of communications: I^2C and SPI. The first one is used to interconnect ICs (Integrated Circuits) on a PCB. It was created by Philips Semicondutors, it is cheap, simple and widely used and provides four data rates: 100, 400 kb/s, 1, 3.4 Mb/s.

The second one is used in chip–to–chip connection in a PCB or between PCBs. Additionally, SPI establishes the communication between microcontrollers and peripheral devices like memory chips, Analog to Digital Converter (ADC), Digital to Analog Converter (DAC) and sensors. It was created by Motorola and the data rate is unspecified, normally 20 Mb/s up to about 100 Mb/s [21].

When it was thought to use AVENAR, the used communication protocol between LSN chip and MSP430F2272 was analysed which is SPI, so it was decided to keep the same communication.

In order to establish SPI communication, four connections are necessary (**Fig. 3.13**):

- MISO – Master In Slave Out – the Slave line for sending data to the Master.
- MOSI – Master Out Slave In – the Master line for sending data to the peripherals.
- SCLK – Serial Clock.
- SS – the pin on each device that the Master can use to enable and disable specific devices.

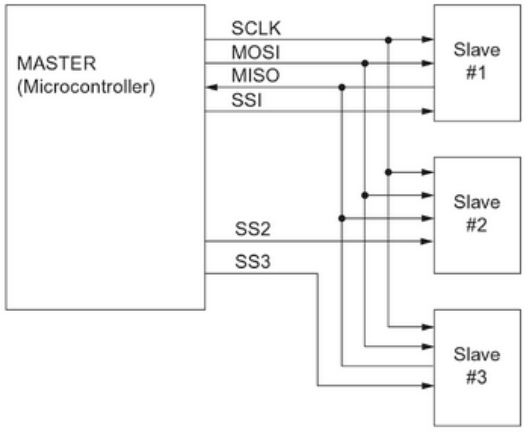


Figure 3.13. SPI connections [21]

It is also necessary to power the BME680 with the ESP8266 energy. **Fig. 3.14** shows the wire connections which have energy and communication lines.

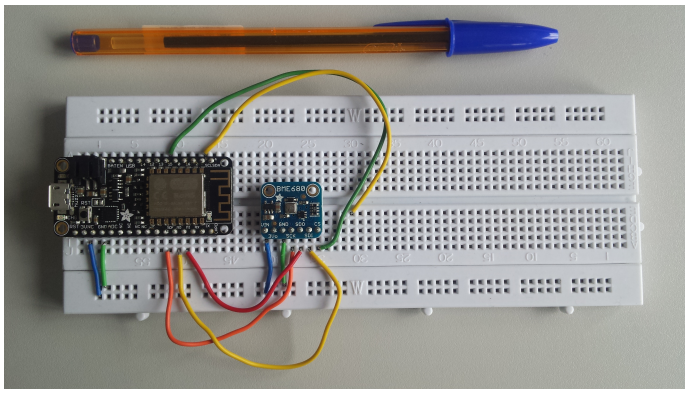


Figure 3.14. Wire connections among ESP8266 and BME680

3.6.2 ESP8266 – FCP–320

It has already been mentioned in **Section 3.2** that the ESP8266 reads the fire detector state using the C–Point. It consists of measuring the voltage between $b1$ and c on the Detector Base MS 400, in normal use the voltage is 0 V and in case of an alarm, it is 24 V.

In an initial phase, it was used one external LED signal and a push button to simulate the fire detector behaviour (**Fig. 3.15**). When the button was pressed, the led turned on and this meant an alarm situation, so ESP8266 indicated 1.

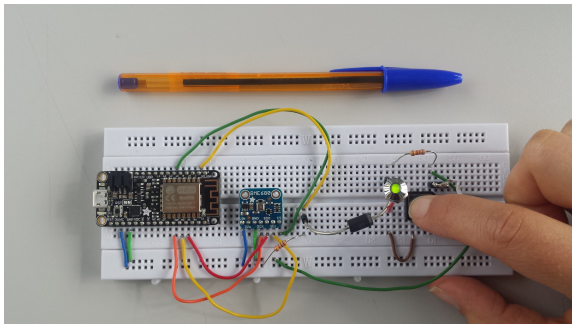


Figure 3.15. Use of an external LED

Then the fire detector was connected to the ESP8266 using the C–Point. **Appendix B** shows the wire connection on the detector base. The voltage at $b1$ is always 24 V and $a1$ is ground. Voltage at c changes according to the situation of alarm or non–alarm:

- Voltage at c is 24 V – normal situation.
- Voltage at c is 0 V – alarm situation.

In order to read the voltage at c, a PC123 photocoupler was used to adjust the voltage levels while 510 Ω and 10 kΩ resistors were used to convert 24 V to 3.3 V (**Fig. 3.16**). In this case, the output voltage of the photocoupler is 3 V in a normal state and 0 V in a

fire situation thus ESP8266 reads 1 in a normal situation and 0 in a fire situation.

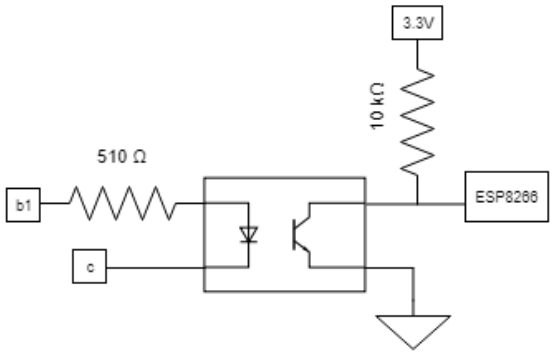


Figure 3.16. Use of a photocoupler

3.6.3 ESP8266 – Website

To send the information about the environmental sensor and the fire detector to a website, the chip that has the Wi–Fi antenna included in the ESP8266 is used. Wi–Fi is a wireless interface used in the Internet of Things connectivity. ESP8266 has the following Wi–Fi standards: 802.11 b/g/n (**Tab. 3.2**).

Table 3.2. Wi–Fi stadards [21]

Standard	Frequency /GHz	Maximum Data Rate /Mb·s^{-1}	Range /m
802.11 b	2.4	11	100
802.11 g	2.4	54	100
802.11 n	2.4/5	600	100

The communication protocol to send data from ESP8266 to the website is Hypertext Transfer Protocol (HTTP). It defines rules which allow information transfer on the World Wide Web and runs on top of the Transmission Control Protocol/Internet Protocol (TCP/IP) [22].

An HTTP session consists of a sequence of network request–response transactions where a client starts a request by establishing a TCP connection to a specific port, usually port 80. A server listener on that port waits for a client request message and when is received, a status line and a message with a requested resource is sent nack [23]. The latest version of HTTP is 1.1 which accesses web pages more quickly and decreases web traffic [24].

3.7 Summary

This chapter explained, in detail, the components used when carrying out the proposed task and the connections among them. The reasons to not use AVENAR have been mentioned and an alternative solution which consists of using the conventional fire detector

FCP–320 and ESP8266 microcontroller were presented. The environmental sensor information and the state of the fire detector are sent through the HTTP protocol to a web page.

The programming code in Arduino IDE, the definition of sending data to a web page and the configuration of an alert message are covered in the next **Chapter 4**.

Chapter 4

Software Design

This chapter describes software design, including the programming code, sending information to a website and sending Short Message Service (SMS) to a phone number in case of an alarm and detection of poor air quality.

4.1 Arduino IDE

Chapter 3 has already mentioned that ESP8266 is programmed using the Arduino IDE. The extension of the file is .ino, the programming language is C++ and the structure is divided in setup() and loop(). The setup() function is called when a sketch starts but it only runs once, after each power up or reset of the board. Inside setup() variables and pin modes are initiated as well initial values set. While the loop() function is looping consecutively, allowing the programme to change and respond.

BSEC has a library to use in Arduino IDE which allows higher–level signal processing for the BME680 [25]. There is a BME680 Application Programming Interface (API) and this driver package includes two header files (.h) and one source file (.c) (**Tab. 4.1**) [26].

Table 4.1. Files' content [26]

File	Content
BME680_defs.h	Constants, macros and datatype declaration.
BME680.c	Declaration of the sensor driver APIs.
BME680.h	Definition of the sensor driver APIs.

BSEC library is supported in 32, 16 and 8 bit Microcontroller Unit (MCU) platforms and it is compatible with ESP8266 using xtensa–1x106–elf–gcc compiler.

In this task, one example of BSEC Software Library with additional functions is used: read the BME680 values, read the status of the FCP–OT320 and send the information to a website.

The used example from BSEC Software is basic_config_state_ulp_plus.ino which includes three libraries: EEPROM.h, bsec.h and bsec_serialized_configurations_iaq.h.

The EEPROM.h (Electrically-Erasable Programmable Read–Only Memory) library allows read and write to the non–volatile memory of ESP8266. BSEC state is saved when IAQ accuracy reaches 3 for the first time. Next time the programme is started, it will be read the EEPROM state of ESP8266.

Header file bsec.h has the definitions for the library and it includes a list of everything that is inside such as other necessary libraries.

bsec_serialized_configurations_iaq.h configures solutions to specific needs. There are three different parametres which can be chosen: supply voltage of BME680, maximum time between values reading and time to background calibration. **Fig. 4.1** shows the possible configurations of BME680.

Configuration	Supply voltage of BME680	Maximum time between bsec_sensor_control() calls	Time considered for background calibration
generic_33v_300s_28d	3.3V	300s	28 days
generic_33v_300s_4d	3.3V	300s	4 days
generic_33v_3s_28d	3.3V	3s	28 days
generic_33v_3s_4d	3.3V	3s	4 days
generic_18v_300s_28d	1.8V	300s	28 days
generic_18v_300s_4d	1.8V	300s	4 days
generic_18v_3s_28d	1.8V	3s	28 days
generic_18v_3s_4d	1.8V	3s	4 days

Figure 4.1. Available BSEC configuration[20]

Supply voltage can be 1.8 V or 3.3 V and can influence the self–heating of the sensor. The maximum time between two measurements is 3 s in normal mode and 300 s in Ultra Low Power mode (ULP), in this mode the system sleeps for 300 s (5 min) to minimize power consumption. The history of BSEC is considered for the automatic background calibration of the IAQ in days. So, if there are changes in this time period, it will influence the IAQ value. BSEC can consider the last 4 or 28 days of operation for automatic background calibration.

File bsec_serialized_configurations_iaq.cpp contains the chosen BSEC configuration. The selected configuration is generic_33v_300s_4d because the supply voltage from ESP8266 to the sensor is 3.3 V, the time between two measurements is 300 s and it was chosen to consider the last 4 days of operation for the automatic background calibration.

General Purpose Input/Output (GPIO) 15 of ESP8266 is the slave select so the function to define the SPI communication between BME680 and ESP8266 is: iaqSensor.begin(15, SPI). To read the state of FCP–320, it is necessary to choose a pin which will be the input, in this case, GPIO 5 of ESP8266 was chosen. As previously mentioned, ESP8266 reads 1 in a normal situation and 0 in a fire situation thus in programming code the *DigitalRead* function which reads the state of pin 5 is inverted using *!* symbol. When the state of this pin changes from 1 to 0, it runs the interrupt which performs an extra measurement and a SMS alert is sent.

The mains steps of the code are below:

- Include the necessary libraries;
- Define the function declarations;
- Create an object of the class Bsec;
- Define the apiKey from the ThingSpeak;
- **void setup**:
  - Connects to Wi–Fi;
  - Setup the pin which reads the fire detector state;
  - SPI configuration;

- **void loop**:
  - Digital read of the pin which reads the fire detector state;
  - Print the output data;
  - Send the output data to website;
- Define the function declarations;

The programming code, bsec_serialized_configurations_iaq.h and bsec_serialized_configurations_iaq.cpp files are attached in **Appendices C.1**, **C.2** and **C.3**, respectively.

4.1.1 Interrupt

As previously mentioned, when the pin which reads the state from the fire detector, changes from 1 to 0 (falling), the interrupt routine runs once and then goes back to the main code (**Fig. 4.2**). In other words, ESP8266 is working normally and when the input pin 5 triggers from high to low, the normal routine is interrupted, then the interruption routine is executed and returns to normal routine.

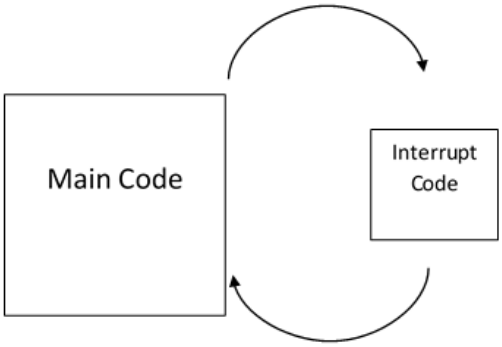


Figure 4.2. Interrupt routine

The interrupt routine is defined as follows:

```
1  void setup() {
2    /* Setup pin that reads fire detector state and executes the interrupt ...
         */
3    pinMode(5, INPUT);
4    attachInterrupt(digitalPinToInterrupt(5), alarm_situation, FALLING);
5  }
6
7  void loop() {
8    int alarmState = !digitalRead(5); // Reads the value from input pin 5
9  }
10
11 /* Interruption that occurs when there is an alarm situation */
12 void alarm_situation()
13 {
14   int alarmState = !digitalRead(5);
15   Serial.println("Alarme");
16   bsec_virtual_sensor_t sensorList[1] = {
17     BSEC_OUTPUT_IAQ_ESTIMATE,
18   };
19
```

```
20    iaqSensor.updateSubscription(sensorList, 1, ...
          BSEC_SAMPLE_RATE_ULP_MEASUREMENT_ON_DEMAND);
21    checkIaqSensorStatus();
22  }
```

4.1.2 Sleep mode

ESP8266 and BME680 have sleep modes with lower energy consumption than in a normal operation mode. The equipment executes the routine and then falls asleep for a certain amount of time. After that, it wakes up to run the code again and goes back to sleep.

The BME680 current consumption was presented in **Tab. 3.1**. It was implemented the ultra–low power mode which features an update rate of 300 seconds (5 minutes) and has an average current consumption of 0.1 mA.

ESP8266 has three different sleep modes which allow to disable, or not, the following functions:

- Wi–Fi connection;
- System clock which transmits a steady high–frequency signal to synchronize all the internal components;
- Real–Time Clock (RTC) which keeps track of the current time;
- CPU which performs the programming code.

Tab. 4.2 defines the existing ESP8266 sleep modes.

Table 4.2. Sleep modes of ESP8266 [27]

Item	Modem–sleep	Light–sleep	Deep–sleep
Wi–Fi	OFF	OFF	OFF
System clock	ON	OFF	OFF
RTC	ON	ON	ON
CPU	ON	Pending	OFF
Current	15 mA	0.4 mA	20 μA

The deep–sleep mode was implemented and noticed that the programme ran once, then the ESP8266 fell asleep for a defined time, restarted to run the code and feel asleep again. This way, the environmental sensor could not achieve IAQ Accuracy 3 because there was not enough time for a stable reading (20 minutes) so ESP8266 was set to work at full power.

Current measurement

Current measurements were made in two operating states: normal and alarm. The average values were calculated and are presented in **Appendix D**.

FCP–320 without the additional electronics consumes 99.56 µA in normal mode and 29.24 mA in alarm situation. The red LED turns on when a fire situation is detected thus the current increases. FCP–320 with the additional electronics consumes 26.41 mA in normal mode and 61.29 mA in alarm situation.

4.2 ThingSpeak

To send the environmental sensor data and the state of FCP–320, it is used an IoT platform named ThingSpeak. This platform enables to collect, store, analyse and visualize data from sensors. It works with different hardware including Arduino, so it was possible to use ESP8266.

The main element of ThingSpeak activity is the channel which contains *data fields*, *locations fields* and a *status field*. The data is sent to the channel and then processed. The data can be visualized using a MATLAB code as well as generate tweets and alerts based on the processed information [28].

The microcontroller starts collecting the data from the environmental sensor BME680 and from the fire detector which indicates if there is or not a fire situation. The second step is to analyse the data, for example, calculate the average temperature or the maximum humidity. Finally, it is possible to act when the fire starts by sending a SMS alert to a defined phone number.

ThingSpeak has Analytics and Actions Apps which transform and visualize data or trigger an action. In Analytics there are MATLAB analysis to explore and transform data, MATLAB Visualization to visualize data in MATLAB plots and Plugins that displays data in gauges, charts or custom plugins. In Actions there is ThingTweet to connect a device to Twitter and send alerts, TweetControl to listen to the Twitterverse and react in real time and TimeControl which automatically executes actions at predetermined times with ThingSpeak Apps. React when channel data meets certain conditions, TalkBack to queue up commands for the device and ThingHTTP to simplify device communication with web services and APIs [29].

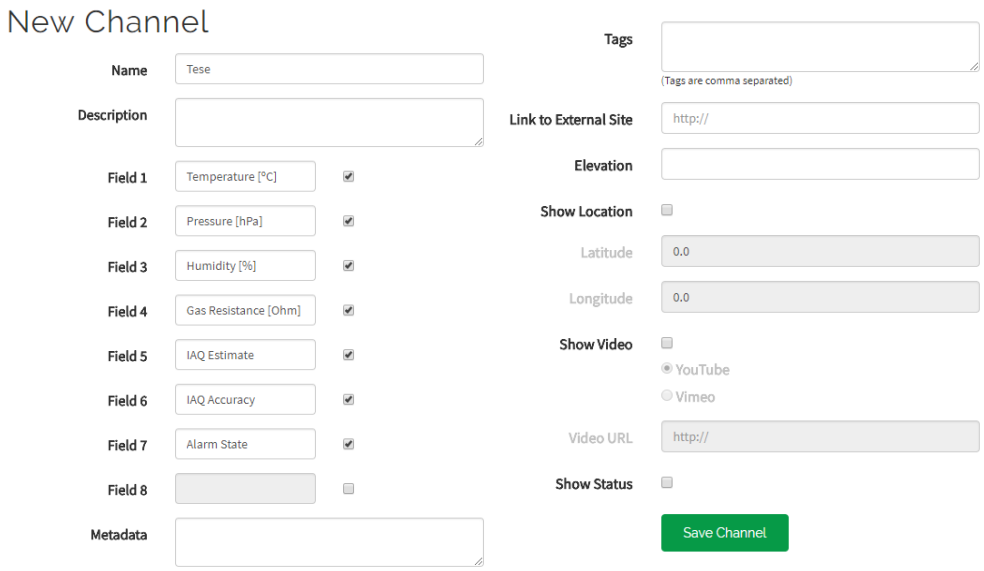
To create a channel in ThingSpeak, it is necessary to indicate some information:

- Channel Name – a unique name for the ThingSpeak channel.
- Description – a description of the ThingSpeak channel.
- Field # – check box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- Metadata – information about channel data, including JSON, XML, or CSV data.
- Tags – keywords that identify the channel.
- Latitude and Longitude – the position of the sensor or thing that collects data in decimal degrees.
- Elevation – the position of the sensor or thing which collects data in meters.
- Link to External Site – Uniform Resource Locator (URL) of a website which contains information about the ThingSpeak channel.
- Video URL – the full path of the YouTube™ or Vimeo® video URL which displays the channel information.

Not all fields are required, **Fig. 4.3** shows the filled fields. Only the name of the channel and the name of the fields were indicated. There are seven fields which corresponds to microcontroller readings: information from BME680 (Temperature, Pressure, Humidity, Gas Resistance, IAQ Estimate and IAQ Accuracy) and the fire detector state.

The created channel has Write API Key and Read API Key. In this case, it is used

the Write API Key in .ino file to write the information to ThingSpeak channel. The Read API Key is used to allow other people to view the private channel feeds and charts.



(a) Fields to fill **(b)** Fields to fill (cont)

Figure 4.3. Filled fields to create a channel on ThingSpeak [30]

4.2.1 Code to send data

The programming code includes the routine of the Wi–Fi connection and the routine to send the information to the ThingSpeak channel. The code responsible for these two functions is as follows:

```
1   /* Replace with your channels thingspeak API key*/
2  String apiKey = "DK50Q5TQUJYK7HIZ";
3  const char* ssid = "Optimus4G_2510";
4  const char* password = "8BF5E79AE68";
5  const char* server = "api.thingspeak.com";
6
7  void setup() {
8    /* Wi-Fi connection */
9    WiFi.begin(ssid, password);
10
11   Serial.println();
12   Serial.println();
13   Serial.print("Connecting to ");
14   Serial.println(ssid);
15
16   WiFi.begin(ssid, password);
17
18   while (WiFi.status() != WL_CONNECTED)
19   {
20     delay(500);
21     Serial.print(".");
22   }
23   Serial.println("");
```

```
24    Serial.println("WiFi connected");
25  }
26
27  void loop() {
28    /* Send data to ThingSpeak */
29      float t = iaqSensor.rawTemperature;
30      float p = iaqSensor.pressure;
31      float h = iaqSensor.rawHumidity;
32      float gr = iaqSensor.gasResistance;
33      float iaqe = iaqSensor.iaqEstimate;
34      float iaqa = iaqSensor.iaqAccuracy;
35      float fire = alarmState;
36
37      if (client.connect(server,80))  // "184.106.153.149" or api.thingspeak...
            .com
38      {
39          String postStr = apiKey;
40             postStr +="&field1=";
41             postStr += String(t);
42             postStr +="&field2=";
43             postStr += String(p);
44             postStr +="&field3=";
45             postStr += String(h);
46             postStr +="&field4=";
47             postStr += String(gr);
48             postStr +="&field5=";
49             postStr += String(iaqe);
50             postStr +="&field6=";
51             postStr += String(iaqa);
52             postStr +="&field7=";
53             postStr += String(fire);
54          client.print("POST /update HTTP/1.1\n");
55          client.print("Host: api.thingspeak.com\n");
56          client.print("Connection: close\n");
57          client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
58          client.print("Content-Type: application/x-www-form-urlencoded\n");
59          client.print("Content-Length: ");
60          client.print(postStr.length());
61          client.print("\n\n");
62          client.print(postStr);
63      }
64      client.stop();
65    } else {
66      checkIaqSensorStatus();
67    }
68  }
```

4.3 Twilio

When the fire starts, the alarm state changes from 0 to 1 and a SMS is sent using the cloud communication platform Twilio which allows making and receiving phone calls as well as sending and receiving SMS.

There are several steps to be performed in order to create the service to send SMS when the fire starts. Firstly, it was created a new project with programmable SMS service. When the project is created, Account SID and Auth Token are automatically generated and this information is used to create a ThingHTTP App (**Fig. 4.4**) [31].

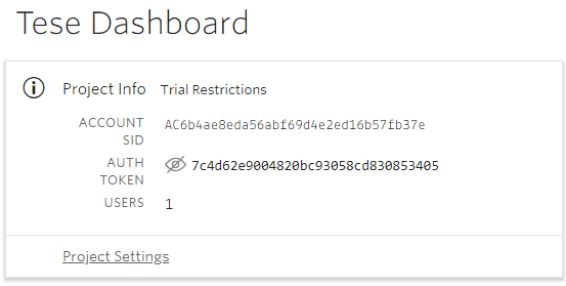


Figure 4.4. Twilio Information [31]

It is also possible to observe a chart with the incoming and outgoing information (**Fig. 4.5**).

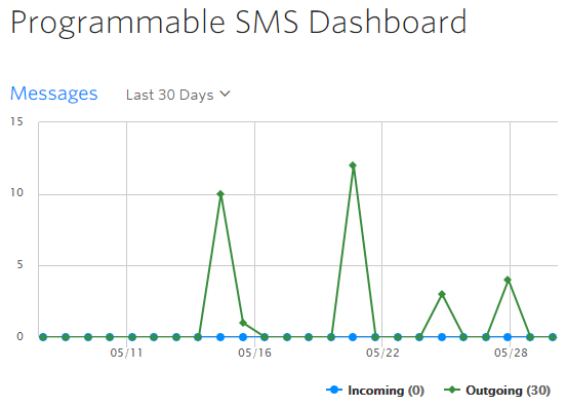


Figure 4.5. SMS usage [31]

4.4 Configuration of SMS sending

In order to send a SMS message it is necessary to use two ThingSpeak Actions Apps: ThingHTTP and React. ThingHTTP allows communications between devices, websites and web services without implementing the protocol on the device level. The actions are specified in ThingHTTP and the triggering is defined on TweetControl, TimeControl or React Apps. React can be used with ThingHTTP, ThingTweet and MATLAB Analysis to achieve actions when channel data reaches a certain condition.

To create a ThingHTTP App, it is necessary to fill the following fields although not all are required (**Fig. 4.6**).

- Name – a unique name for ThingHTTP request.
- API Key – auto generated API key for the ThingHTTP request.
- URL – the address of the website which is requesting data from or writing data to starting with either http:// or https://.
- HTTP Auth Username – if the URL requires authentication, enter a username for authentication to access private channels or websites.

- HTTP Auth Password – if the URL requires authentication, enter a password for authentication to access private channels or websites.
- Method – the HTTP method required to access the URL.
- Content Type – the MIME or form type of the request content.
- HTTP Version – the version of HTTP on the server.
- Host – if the ThingHTTP request requires a host address, enter a domain name.
- Headers – if the ThingHTTP request requires custom headers, enter the information. The name of the headers and a value must be specified.
- Body – message which is required to be included in the request.
- Parse String – the exact string to look for in the response data, if it is necessary to parse the response.

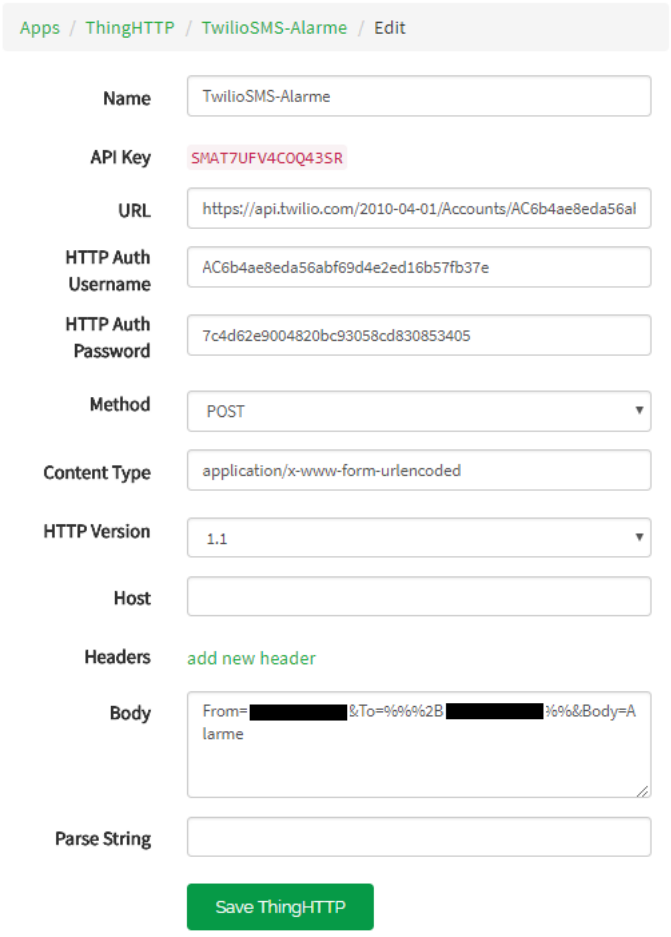


Figure 4.6. ThingHTTP App – Alarm

It was indicated the name and the URL which contains the Account SID from Twilio, the HTTP Auth Username corresponds to the Account SID from Twilio and the HTTP Auth Password is the Auth Token from Twilio (**Fig. 4.4**). The used method is post because the information from ESP8266 is processed and then posted on ThingSpeak channel. The

field body contains the number which sends the SMS, the number which receives it and the sent text: “Alarme”.

Finally, to create a React App, it is necessary to fill the following fields (**Fig. 4.7**).

- React Name – a unique name for React App.
- Condition Type – a condition type corresponding to the data. A channel can hold numeric sensor data, text, strings, status updates, or geographic location information.
- Test Frequency – whenever the condition is tested, every time data enters the channel or on a periodic basis.
- Condition – select a channel, a field and the condition for the React.
- Action – select ThingTweet, ThingHTTP, or MATLAB Analysis to run when the condition is met.
- Options – when the React runs, run action only the first time the condition is met or run action each time condition is met.

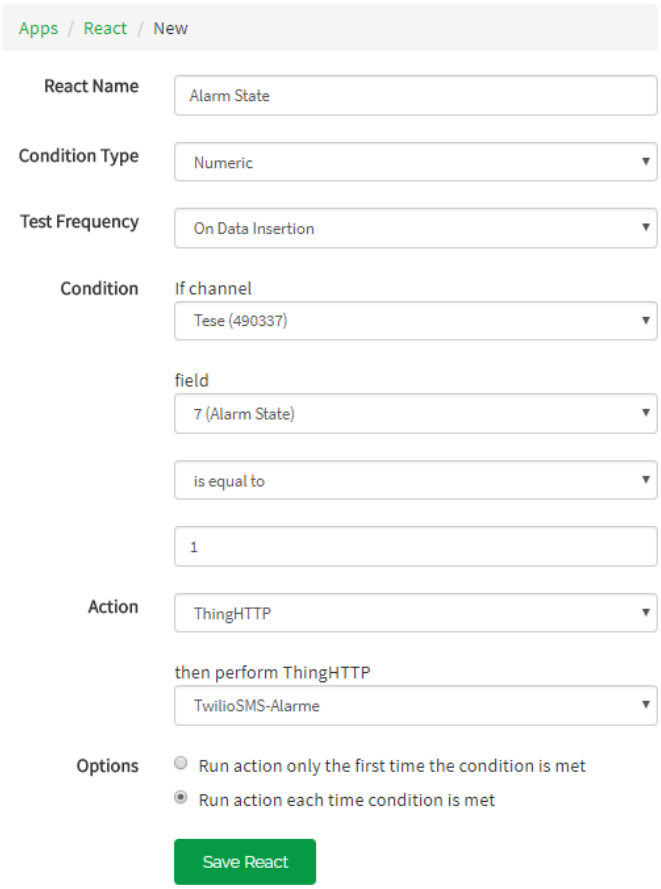


Figure 4.7. React App – Alarm

It was indicated the name of the React App, the condition type is numeric because the alarm state is 0 or 1, the frequency test is on every data insertion because it is necessary to

analyse every measure. The defined condition is on Channel "Tese", whenever the number on the field with Alarm State is 1, an action has to be taken. This action is the defined ThingHTTP which sends the SMS.

Additionally, it was defined other SMS when the IAQ Estimate is greater than 200 which means that the air quality is worse according to **Fig. 3.9**. It was created another ThingHTTP and React Apps to send this SMS.

4.5 MATLAB Analysis

MATLAB Analysis calculates average of data contained on the fields from a channel. It was created another channel which receives the average measurement from the first channel.

This channel contains the average values of Temperature, Pressure, Humidity, Gas Resistance and IAQ Estimate. MATLAB Analysis App contains a MATLAB Code which reads the values from channel "Tese", calculates the averages and writes these values on the respective fields on a new channel. This MATLAB Code is presented in **Appendix C.4**. Moreover, it was added the Time Control App which schedules when MATLAB Analysis App should be executed (**Fig. 4.8**).

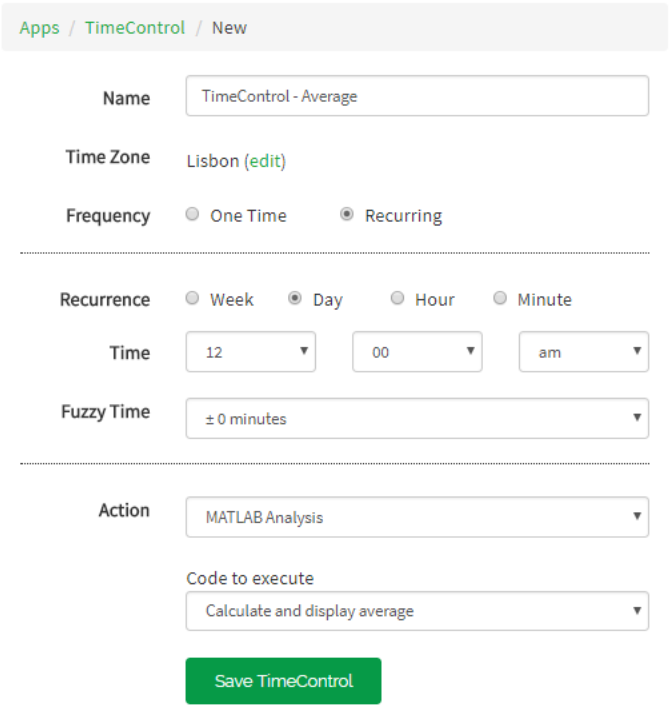


Figure 4.8. TimeControl App

It is necessary to fill the following fields:

- Name – a unique name for TimeControl.
- Time Zone – according to location.
- Frequency – when TimeControl should run.
- Recurrence – specification of week, day, hour or minute.

- Time – time at which TimeControl should run.
- Fuzzy Time – minutes around a scheduled time to run TimeConrol.
- Action – to be trigger when specified parametres are met. It can be MATLAB Analysis, ThingHTTP, ThingTweet or TalkBack.

It was defined to execute the MATLAB Analysis "Calculate and display average" daily at midnight.

This channel can be accessed through the link: https://thingspeak.com/channels/512784.

4.6 Results

This section shows the obtained results: the channel view on the website and on the ThingView app. Additionally, the received SMS in case of an alarm detection and in poor air quality are presented.

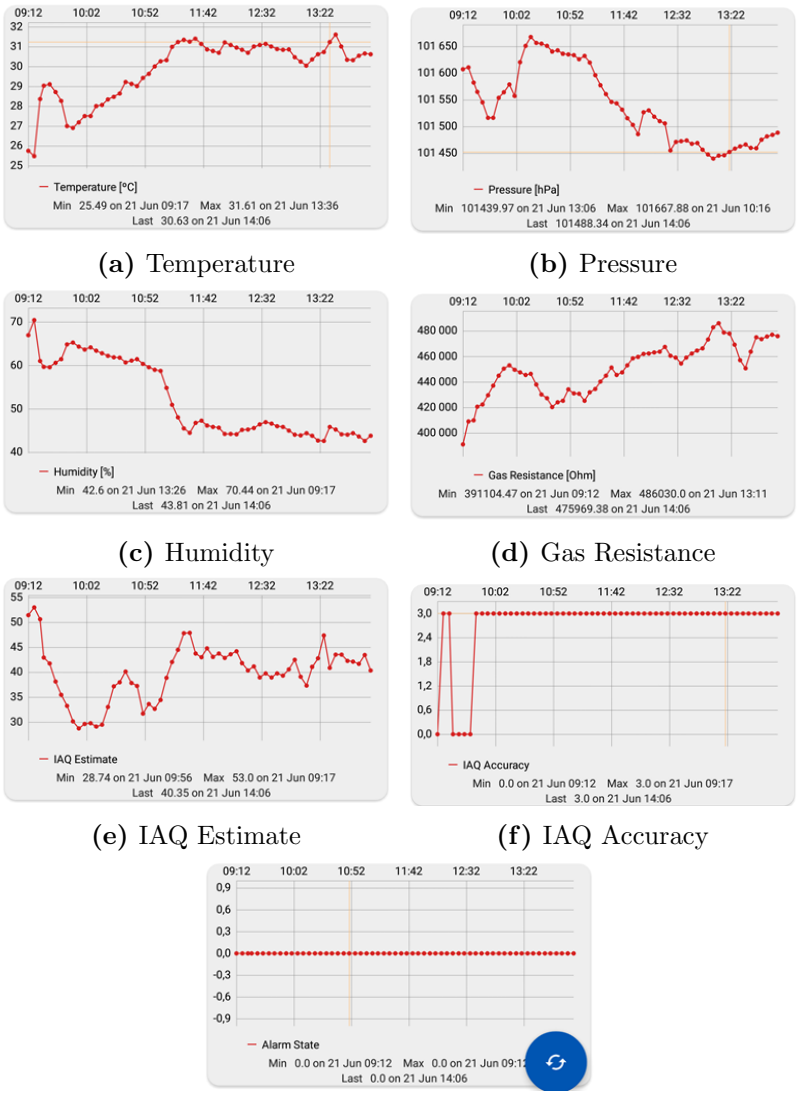
4.6.1 ThingSpeak channel

The created ThingSpeak channel is presented in **Fig. 4.9** and it can be accessed through the link: https://thingspeak.com/channels/490337.

Figure 4.9. Channel view

4.6.2 ThingView – ThingSpeak viewer

ThingView is a free app for mobile phones which makes it possible to observe the information contained in a ThingSpeak channel, it is only necessary to introduce the channel ID (**Fig. 4.10**).



(a) Temperature **(b)** Pressure

(c) Humidity **(d)** Gas Resistance

(e) IAQ Estimate **(f)** IAQ Accuracy

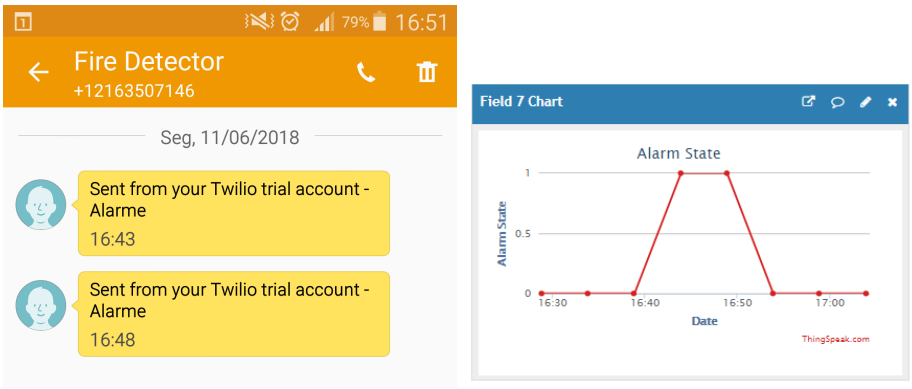
(g) Alarm State

Figure 4.10. Charts of the measured parametres on smartphone

4.6.3 Alarm detection

As previously stated in **Section 3.6.2**, it was used one external LED to simulate the signal coming from the fire detector. To turn on the LED, it is necessary to press the button. When the button is pressed, the LED turns on and an extra measurement occurs, sending the data to ThingSpeak and a SMS to the mobile phone indicating the alarm situation.

Fig. 4.11a shows the received SMS alert and **Fig. 4.11b** shows the field of Alarm State on ThingSpeak channel.



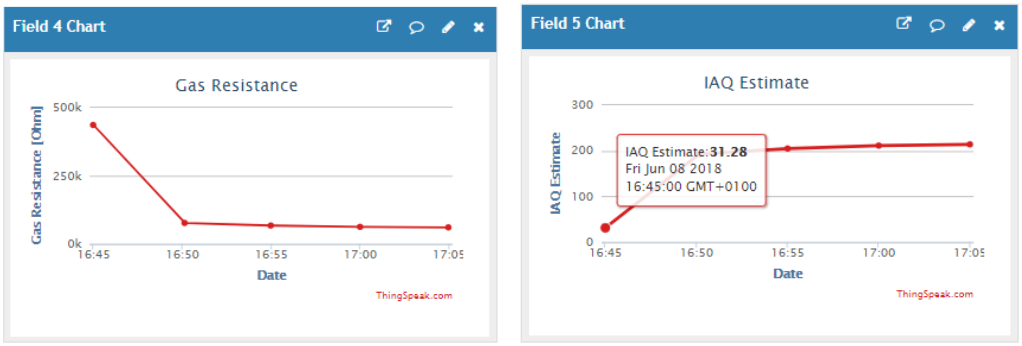
(a) Received SMS – Fire Alarm **(b)** Alarm State

Figure 4.11. Alarm detection

4.6.4 Poor air quality detection

To simulate poor air quality IAQ higher than 200, the sensor was placed in a box with an open permanent marker which contains alcohol, similar conditions which were used in the first sensor calibration (**Fig. 3.12**).

Fig. 4.12 shows the fields of Gas Resistance and IAQ Accuracy on ThingSpeak, note that when IAQ Accuracy is high, it means that the quality of the air is not good so the value of the gas resistance is low. **Fig. 4.13** shows the received SMS alert.



(a) Gas Resistance **(b)** IAQ Accuracy

Figure 4.12. Fields results

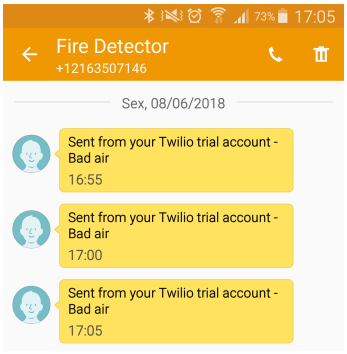


Figure 4.13. Received SMS – IAQ

4.7 Summary

This chapter covered the programming code which includes the routine to connect to Wi–Fi and send data to ThingSpeak. It was explained the creation of ThingSpeak channel and the configuration of the use of Apps and MATLAB Analysis. Moreover, it was presented the received SMS in case of an alarm and in poor air conditions.

The next step is to create a PCB which includes electronic components and to place it into the fire detector. These are presented in the next **Chapter 5**.

Chapter 5

Prototype Development

This chapter explains the construction of a PCB which contains ESP8266 and BME680. Additionally, it describes the new part design where these electronic components fit.

5.1 PCB development

A PCB board was produced where ESP8266 and BME680 were placed. This PCB contains tracks which connect the different pins in order to establish communication between the μC and the environmental sensor. It also has the track which connects the FCP–320 signal to the μC to read the fire detector state.

ESP8266 is powered up by FCP–320, so it is necessary to convert the 24 V DC to 3.3 V DC which is done by a DC/DC converter.

5.1.1 Development steps

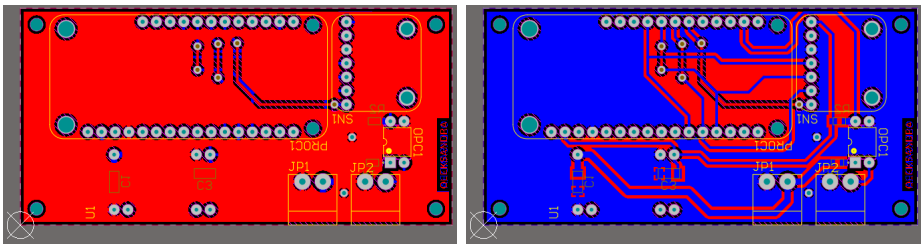
The software used to create a PCB board was Altium Designer. Firstly, the components and their pins were defined and added to the library. Then, two connectors , one photocoupler and two capacitors were added. The next step was to design a PCB layout which connects these components and the tracks (**Fig. 5.1a**, **Fig. 5.1b**).

Afterwards, another software called CircuitCAM was used which permitted the creation of gerber files with the description of bottom and top layers of the PCB and the contained holes. Additionally, gap zones to perform a better cut were added, the zone where the machine has to cut the PCB board and the areas where the copper should be removed around the tracks were defined.

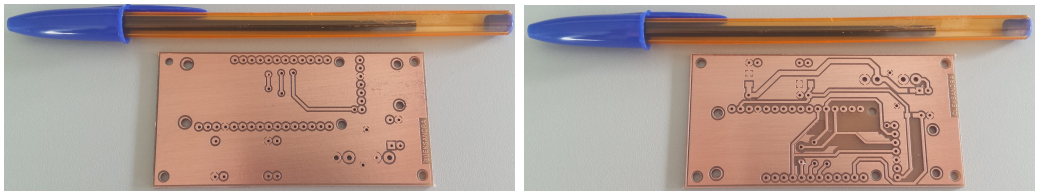
The CircuitCAM files were converted by Board Master ProtoMat C30 software which communicates to the printing machine in order to manufacture the PCB board.

The PCB was wiped with the steel wool to become brighter and an adequate lacquer was used to prevent PCB oxidation (**Fig. 5.1c**, **Fig. 5.1d**).

The copper alloy rivets were manually placed on the PCB with the help of tweezers, automated punch with the stamp and anvil. The rivets are used to connect the circuits of both sides of the board. The produced PCB is double–sided which has wiring patterns on both sides of the insulating material, this is, the circuit pattern is available on the components side and the solder side [32].



(a) Top layer in Altium **(b)** Bottom layer in Altium



(c) Top layer **(d)** Bottom layer

Figure 5.1. PCB

The PCB material is FR–4 which is a National Electrical Manufacturers Association (NEMA) name for glass-reinforced epoxy laminates. It is an electrical insulator, a flame retardant, and has a good strength–for–weight ratio [33]. The PCB layers are copper, FR–4 and copper.

The final step was to solder all the components to the PCB (**Fig. 5.2**).

(a) Top layer **(b)** Bottom layer

Figure 5.2. Final PCB

Appendix A.3 shows the schematic of the connections between all the components presented on the PCB.

5.1.2 Components

This section identifies and characterizes the components used in the PCB construction. As previously mentioned, it were used a DC/DC converter, photocoupler, capacitors and connectors.

DC/DC converter

The used DC/DC converter is THL 3 WI from Traco Power. The input voltage is between 9 and 36 V DC, being 24 V the nominal voltage while the output voltage is 3.3V and the maximum output current is 600 mA [34].

There are two capacitors of 47 μF in the input and in the output of the DC/DC converter (**Fig. 5.3**). The capacitors filter the signal and guarantee the constant voltage,

this is, they absorb the eventual voltage peaks and ripple.

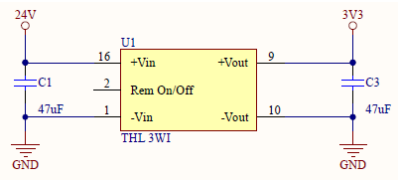
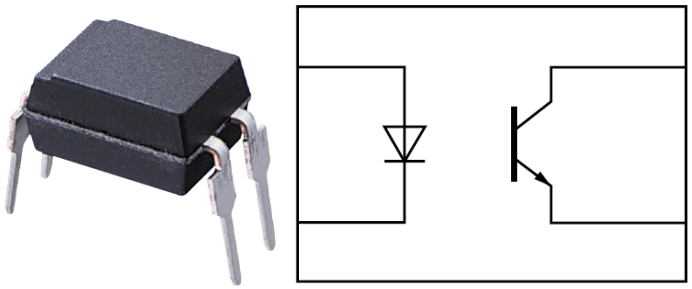


Figure 5.3. Power circuit

Photocoupler

As previously mentioned in section **Section 3.6.2**, it was used a photocoupler to read the C–Point voltage (**Fig. 5.4**). This component adjusts the voltage levels of 24 V circuit from FCP–320 and 3.3 V from ESP8266.



(a) Photocoupler **(b)** Internal Connections Diagram

Figure 5.4. PC123 Photocoupler [35]

Connector

It was used two connectors each with two positions, one to provide the power supply to the microcontroller and other to provide the signal which contains the fire detector state, both having the ground line (**Fig. 5.5**).

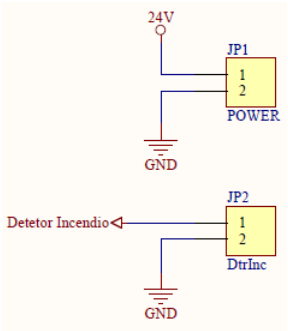


Figure 5.5. Schematic of the connectors

5.2 New part design

The produced PCB should be included in the fire detector. There is no available space inside the FCP–320 and MS 400, so it is necessary to create a new part with enough space to include the PCB.

There are three different possibilities:

- Create a part between FCP–320 and MS 400.
- Modify MS 400 to include the PCB.
- Create a part which is assembled to MS 400 and includes the PCB.

The first possibility implies producing a new part which geometry has to be compatible with FCP–320 and MS 400. In other words, it must have the feature to fix to FCP–320 and to MS 400 simultaneously. It is difficult because of the way that the FCP–320 fits on MS 400, the fire detector has to rotate to engage the base. It also is necessary to design a new part with the same terminals to fix the fire detector and the equivalent terminals to fix the base. This solution is very complex because it has to simultaneously satisfy the connection to FCP–320 and to MS 400.

The second option implies increasing the internal size of the base to fit the PCB. It is necessary to keep the base terminals to connect the fire detector and redesign the base to fix to the fire detector. This redesign appears to be complicated because it is necessary to modify an existing base.

The third option has a single requirement which is to mount to base. It is easier and more flexible in terms of design, so this solution was selected.

5.2.1 Construction

This section is dedicated to explain the steps in the construction of the new part. It was decided to construct the new part with similar geometry of the detector base. It has a truncated cone shape with free space inside to assemble the PCB. The dimensions are 160 $\times$ 23 $\times$ 160 mm.

It has two holes for the cables to pass similar to the detector base, two holes to fix it with M6 $\times$ 13 screws to the surface, two big plastic bosses to fix it to the detector base with M4 $\times$ 14 screws and four little plastic bosses to fix the PCB with M2 $\times$ 4.5 screws. The screws that are placed on the plastic bosses are self–tapping. **Fig. 5.6** shows the rules to design the plastic bosses and **Tab. 5.1** shows the recommended dimensions and the chosen dimensions for the plastic bosses.

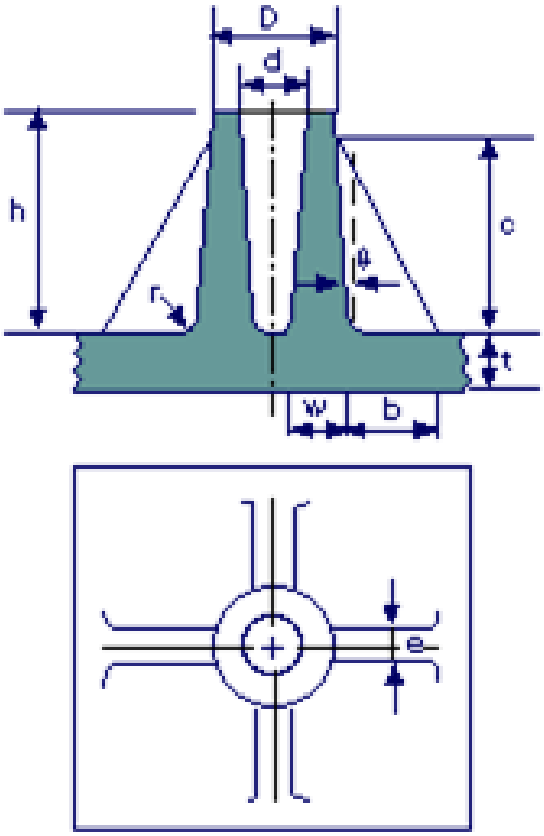


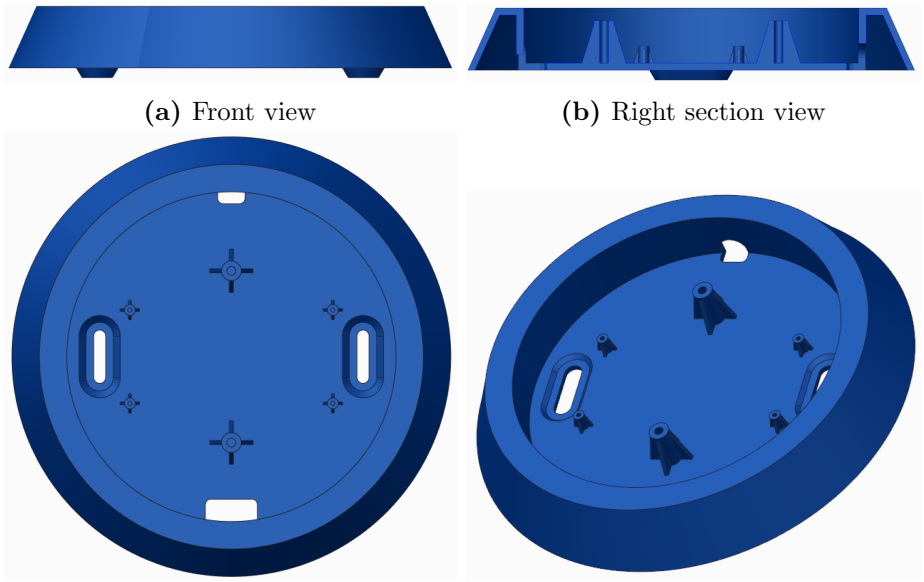
Figure 5.6. Rules to design the plastic bosses [36]

Table 5.1. Dimensions for plastic bosses with gussets [36]

Recommended dimensions	Big Plastic Boss	Little Plastic Boss
$w = 0.5t - 0.8t$	$w = 1.25$ mm	$w = 1.25$ mm
$h = 2.0D - 2.5D$	$h = 15$ mm	$h = 6.0$ mm
$\theta = 0.5\,^{\circ} - 2.0\,^{\circ}$	$\theta = 0.5\,^{\circ}$	$\theta = 0.5\,^{\circ}$
$r = 0.13$ mm – 0.20 mm	$r = 0.20$ mm	$r = 0.20$ mm
$D = 2.5d - 3.0d$	$D = 7.2$ mm	$D = 4.0$ mm
$c \leq 0.95h$	$c = 13.7$ mm	$c = 5.0$ mm
$b = 0.3c - 1.0c$	$b = 4.3$ mm	$b = 1.5$ mm

The surface thickness t is 2.5 mm and with this dimension is possible to stipulate the other ones. The dimension d is according to the used self–tapping screw. The dimension D in the bigger plastic boss is lower than what is advisable to decrease the thickness of the wall, because of this the dimension w will not be the recommendable. In the other plastic boss, the dimension h is smaller to fit the height of the PCB and not to touch the base detector.

The software used to design the part is by Creo Patametric. **Fig. 5.7** shows the designed part and **Appendix E** shows its technical drawing.



(a) Front view **(b)** Right section view

(c) Top view **(d)** ISO view

Figure 5.7. New part

This part was made by 3D additive manufacturing, PolyJet technology, which produces prototypes with a precision of 16 μm and allows the production of complex geometries [37]. The material of this new part is Acrylonitrile Butadiene Styrene (ABS) and the resolution is 30 μm.
In 3D additive manufacturing, the dimensions θ and r are not important so they were not designed. If the manufacturing process was by plastic injection molding, it would be necessary to have several design consideration. The following section shows the main consideration and the implemented changes to satisfy the injection requirements.

5.2.2 Plastic injection molding

Plastic injection molding consists of injecting molten plastic material into a mold under high pressure. Then, the material is cooled, solidified and released by opening the two halves of the mold.

There are several design rules:

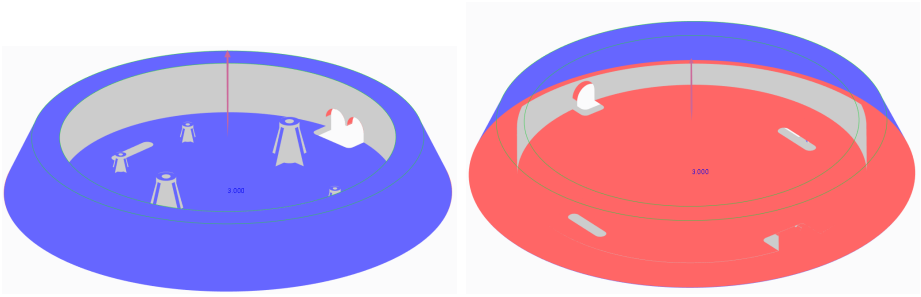
- Wall thickness rule – compliance with similar wall thickness, typical for plastic.
- Curvature rule – curvature of corners and edges.
- Draft rule – all surfaces that lie in the demolding direction are provided with a draft.
- Undercut rule – avoid undercuts.

The wall thickness is 2.5 mm and it is similar in all the part. The created part does not have undercuts so the first and the last rules were satisfied. The curvature rule was also

satisfied by rounding the corners and the edges. It was used the radius of 0.50 mm on the internal geometries and 1.0 mm on external walls.

Creo Parametric has the functionality to analyse the draft situation which was used to understand which surfaces did not satisfy the draft rule. It has a colour scale that indicates the existence or not of drafts.

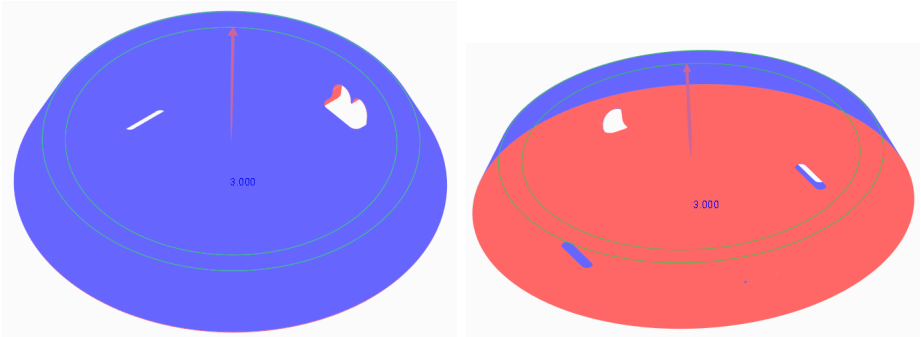
The blue colour means that the surface which is lying in the demolding direction has a draft, the pink colour means that the surface which is lying in the opposite demolding direction has a draft and the grey colour means that the surface does not have drafts (**Fig. 5.8**). In order to fulfil this rule, 3° drafts have been added to the grey surfaces according to their demolding direction.



(a) View 1 **(b)** View 2

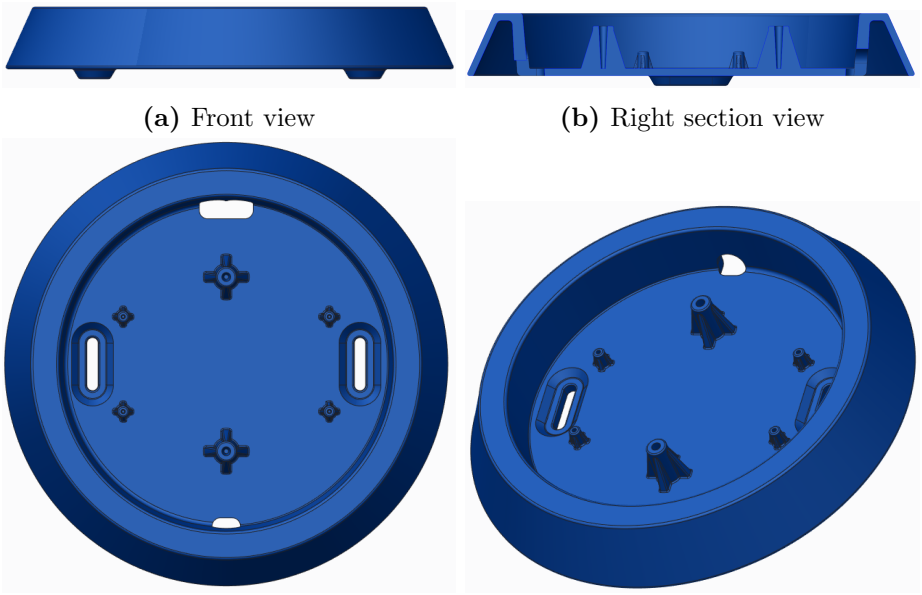
Figure 5.8. First draft analysis

With all the drafts added, the final result of draft analysis is presented in **Fig. 5.9**. The 3D model of the new part satisfies the plastic injection molding rules is presented in **Fig. 5.10**.



(a) View 1 **(b)** View 2

Figure 5.9. Final draft analysis



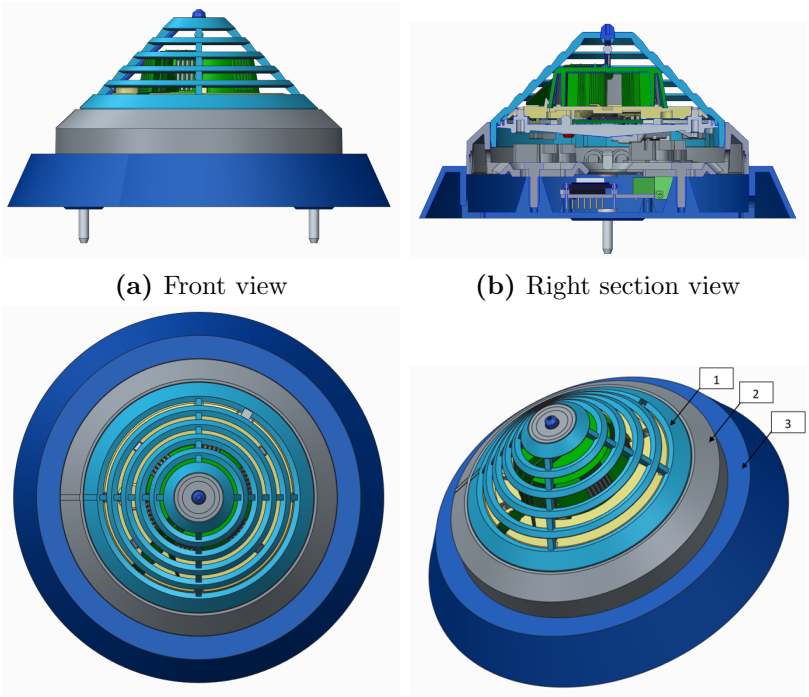
(a) Front view **(b)** Right section view

(c) Top view **(d)** ISO view

Figure 5.10. New part with drafts and rounds

5.3 Final Prototype

This section showed the obtained prototype, firstly it is presented the assembly of the final 3D model (**Fig. 5.11**) and the 3D model of the PCB in Creo (**Fig. 5.12**).



(a) Front view **(b)** Right section view

(c) Top view **(d)** ISO view

Figure 5.11. Final prototype - 3D model

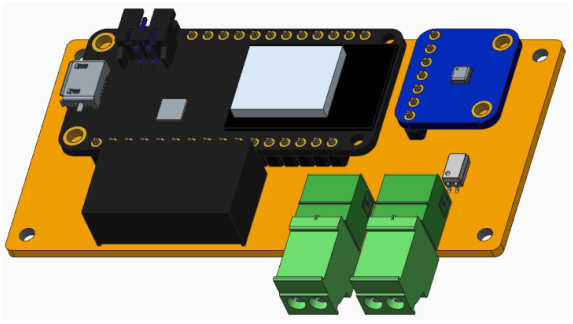
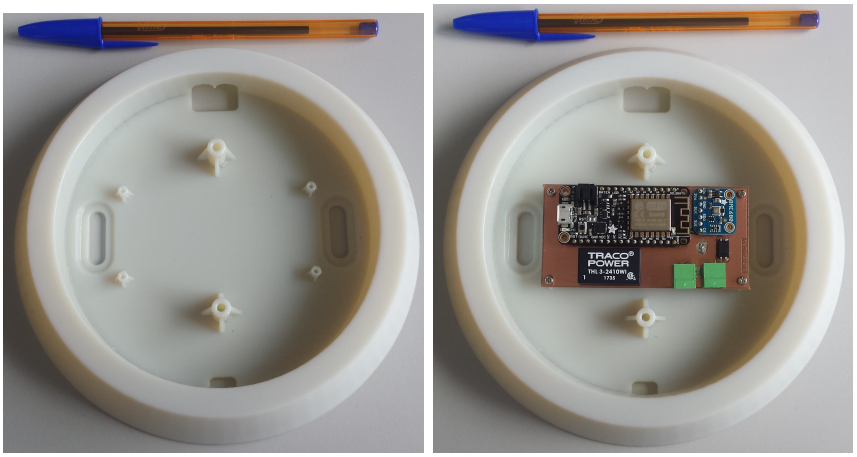


Figure 5.12. PCB 3D model

The represented components in **Fig. 5.11d** are:
1 – FCP–320.
2 – MS 400.
3 – New part.
The obtained new part by 3D additive manufacturing is presented in **Fig. 5.13** and the final prototype is presented in **Fig. 5.14**.



(a) New part **(b)** New part with PCB

Figure 5.13. New part obtained by 3D additive manufacturing

Figure 5.14. Final prototype

5.3.1 Project Costs

This project had associated costs because it was necessary to buy some equipment. The FCP–320 and the MS 400 were not purchased because they are Bosch products. Moreover, PCB construction was performed at the Bosch Ovar and the used capacitors, resistors and photocoupler exist at the plant. So, it was purchased the MSP–TS430DA38 Development Board at the beginning when the decision to work with AVENAR microcontroller was considered. The BME680 Shuttle Board from Bosch was bought which was only used at the beginning while waiting for the arrival of the Adafruit BME680 Development Board. **Tab. 5.2** summarizes the purchased components used in the PCB and associated costs.

Table 5.2. Price of purchased components

Component	Quantity	Price
Adafruit Feather HUZZAH ESP8266	1	10.34 €
Adafruit BME680 Development Board	1	18.45 €
DC/DC Converter	1	13.52 €
3D printing of the new part	1	205.80 €
Total	4	248.11 €

5.4 Summary

This chapter focused on the main stages of PCB development which includes ESP8266, BME680, one photocoupler, two capacitors, two connectors and one DC/DC converter. The construction of a new part and the rules used to design it are stated. Additionally, it was explained the necessary modifications to be taken into account if the new part was made by plastic injection molding. Finally, the prototype and its operations were presented.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

The aim of this project was to implement an environmental sensor inside a fire detector to construct a 2 in 1 product which detects fire and simultaneously measures the air quality.

The state of the art explained the concept of Smart Buildings, mentioned the most important standards concerning to fire detection systems and presented some fire detectors available in the market.

After designing the architecture of the proposed system, it was noted that instead of using the AVENAR fire detector, it would be used a conventional fire detector FCP–320 because it was not possible to establish communication between AVENAR microcontroller and the environmental sensor. Additionally, it was decided to use ESP8266 microcontroller to read the values from environmental sensor and the fire detector state.

It was implemented SPI communication between ESP8266 and BME680 and C–Point was used to read the fire detector state. Programming was done on Arduino IDE software. Moreover, it was configured to send this information to a website and sent a SMS alert in case of an alarm or poor air quality detection.

The PCB development board was designed using Altium and CircuitCAM software which contains all the necessary components and connections. Additionally, a new part was constructed to fit this PCB. It was designed with Creo software and produced by 3D printing. In conclusion, the prototype was successfully obtained and it is fully functional.

6.2 Future Works

As future projects, it would be interesting to perform the following tasks:

- Implement the environmental sensor inside AVENAR as initially thought, in order to use LSN communication. Programme AVENAR microcontroller which consumes less current than ESP8266 microcontroller. Additionally, redesign AVENAR to include the environmental sensor without having to use the new part.
- Measure the electromagnetic interference caused by the Wi–Fi antenna on the fire detector.
- Use this study in the innovative project that should be signed between Bosch and the University of Porto.

References

[1] Bosch, "Bosch annual report," Dec. 2017, last accessed on 22.05.2018. [Online]. Available: https://assets.bosch.com/media/global/bosch_group/our_figures/pdf/bosch-annual-report-2017.pdf

[2] ——, "Bosch global," last accessed on 22.05.2018. [Online]. Available: https://www.bosch.com/

[3] ——, "Bosch security systems to become bosch building technologies," 2018, last accessed on 22.05.2018. [Online]. Available: https://www.boschsecurity.com/xc/en/news/press-room/2018/bosch-building-technologies/

[4] B. S. Systems, "Avenar detector 4000 - maximize your detection performance," 2017.

[5] B. I. Docupedia, "Smart office buildings," Sep. 2017.

[6] *EN 54*, European Committee for Standardization Std.

[7] *NOTA TÉCNICA Nº 12 - Sistemas automáticos de detecção de incêndio*, Autoridade Nacional de Proteção Civil Std., Dec. 2013. [Online]. Available: http://www.prociv.pt/bk/SEGCINCENDEDIF/Normas%20Tecnicas/12_NT-SCIE-SISTEMAS%20AUTOM%C3%81TICOS%20DE%20DETE%C3%87%C3%83O%20DE%20INC%C3%8ANDIO.pdf

[8] B. S. Home, "Twinguard starter set," last accessed on 05.06.2018. [Online]. Available: https://www.bosch-smarthome.com/uk/en/products/smart-single-solutions/twinguard-starter-set

[9] Bosch, "Avenar detector 4000 - operation manual," 2018.

[10] ——, "Conventional automatic detectors fcp-320 | fch-320," 2017.

[11] B. S. Systems, "Bosch fire detectors - hardware," 2011.

[12] T. Instruments, "Msp430f2272," last accessed on 05.06.2018. [Online]. Available: http://www.ti.com/product/MSP430F2272

[13] ——, "Msp-ts430da38 - 38-pin target development board," last accessed on 05.06.2018. [Online]. Available: http://www.ti.com/tool/MSP-TS430DA38

[14] B. S. Systems, "Ms 400 detector bases," Mar. 2012, last accessed on 05.06.2018. [Online]. Available: http://resource.boschsecurity.com/documents/MS_400_Data_sheet_enUS_1260707595.pdf

[15] Adafruit, "Adafruit feather huzzah esp8266," Sep. 2016.

[16] B. Sensortec, "Bme680 - low power gas, pressure, temperature & humidity sensor," Jul. 2017. [Online]. Available: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME680-DS001-00.pdf

[17] ——, "Bme 680," last accessed on 05.06.2018. [Online]. Available: https://www.bosch-sensortec.com/bst/products/all_products/bme680

[18] Figaro, "Operating principle," last accessed on 30.05.2018. [Online]. Available: http://www.figaro.co.jp/en/technicalinfo/principle/mos-type.html

[19] Adafruit, "Adafruit bme680 environmental sensor development board," last accessed on 22.05.2018. [Online]. Available: https://pt.mouser.com/pdfdocs/adafruit-bme680-humidity-temperature-barometic-pressure-voc-gas.pdf

[20] B. Sensortec, "Integration guide bosch software environmental cluster (bsec)," Nov. 2017.

[21] L. Frenzel, *Handbook of Serial Communications Interfaces: A Comprehensive Compendium of Serial Digital Input/Output (I/O) Standards*. Elsevier Science, 2015. [Online]. Available: https://books.google.pt/books?id=wnGDBAAAQBAJ

[22] "Http (hypertext transfer protocol)," last accessed on 09.06.2018. [Online]. Available: https://searchwindevelopment.techtarget.com/definition/HTTP

[23] "Hypertext transfer protocol - wikipedia," last accessed on 09.06.2018. [Online]. Available: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

[24] "Http 1.1," last accessed on 09.06.2018. [Online]. Available: https://searchmicroservices.techtarget.com/definition/HTTP-11

[25] Bosch, "Bosch sensortec environmental cluster (bsec) software," last accessed on 23.05.2018. [Online]. Available: https://www.bosch-sensortec.com/bst/products/all_products/bsec

[26] B. Sensortec, "Bme680 sensor driver," last accessed on 23.05.2018. [Online]. Available: https://github.com/BoschSensortec/BME680_driver

[27] Espressif, "Esp8266 low power solutions," 2016, last accessed on 05.06.2018. [Online]. Available: https://www.espressif.com/sites/default/files/9b-esp8266-low_power_solutions_en_0.pdf

[28] Mathworks, "Thingspeak support from desktop matlab," last accessed on 05.06.2018. [Online]. Available: https://www.mathworks.com/hardware-support/thingspeak.html

[29] ThingSpeak, "Apps," last accessed on 05.06.2018. [Online]. Available: https://thingspeak.com/apps

[30] ——, "New channel," last accessed on 28.05.2018. [Online]. Available: https://thingspeak.com/channels/new

[31] Twilio, "Dashboard," last accessed on 05.06.2018. [Online]. Available: https://www.twilio.com/console

[32] R. Khandpur, *Printed Circuit Boards: Design, Fabrication, Assembly and Testing*. Tata McGraw-Hill, 2005, last accessed on 08.06.2018. [Online]. Available: https://books.google.pt/books?id=VY8iBAAAQBAJ

[33] S. Circuits, "Fr-4 pcb materials," last accessed on 11.06.2018. [Online]. Available: http://www.sunstone.com/pcb-manufacturing-capabilities/detailed-capabilities/pcb-materials/fr-4-material

[34] T. Power, "Dc/dc converter," Jun. 2016, last accessed on 08.06.2018. [Online]. Available: http://www.farnell.com/datasheets/2547150.pdf?_ga=2.137648114.315590080.1529396976-902483060.1516205962&_gac=1.227876463.1528102654.Cj0KCQjwxtPYBRD6ARIsAKs1XJ7NzS4NvDlby_LpgI5HaAryfLX40vniylJPCI6prBM78T48-Bv3Mi8aAhrCEALw_wcB

[35] Sharp, "Pc123xnnsz0f series," Sep. 2006, last accessed on 08.06.2018. [Online]. Available: http://www.sharp-world.com/products/device/lineup/data/pdf/datasheet/pc123xnnsz_e.pdf

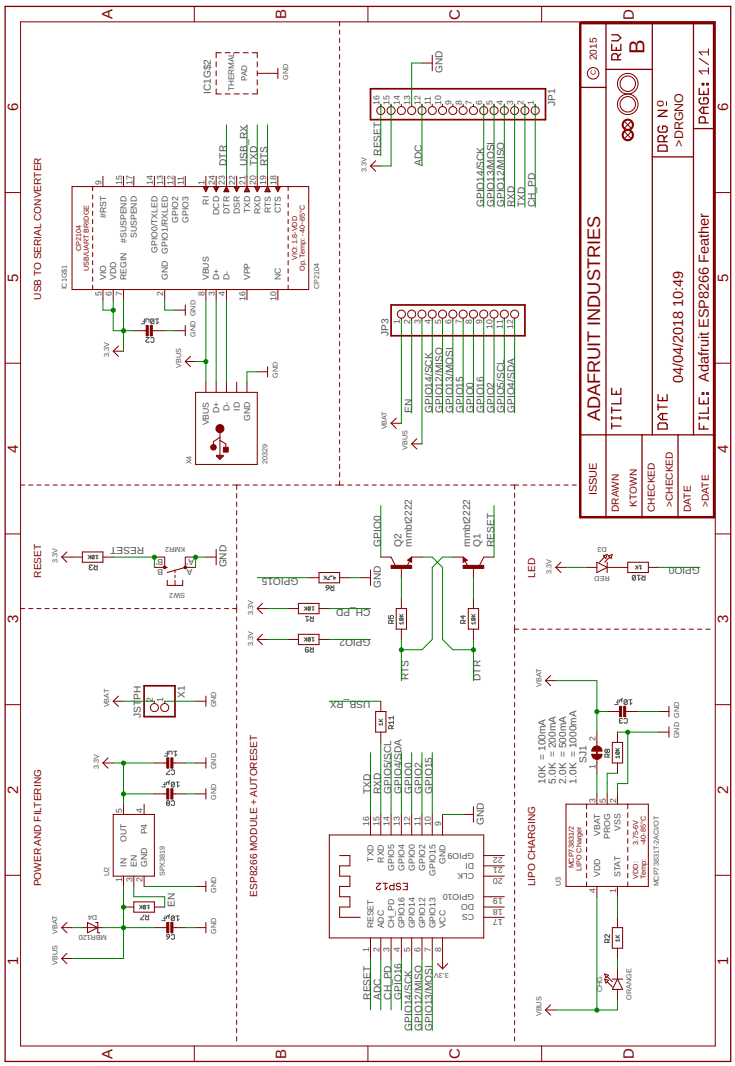
[36] S. C. U. E. D. Center, "Boosting structural integrity with ribs," last accessed on 08.06.2018. [Online]. Available: http://www.dc.engr.scu.edu/cmdoc/dg_doc/develop/design/part/33000003.htm

[37] CODI, "Addictive manufacturing," last accessed on 08.06.2018. [Online]. Available: https://www.codi.pt/en/addictive-manufacturing/

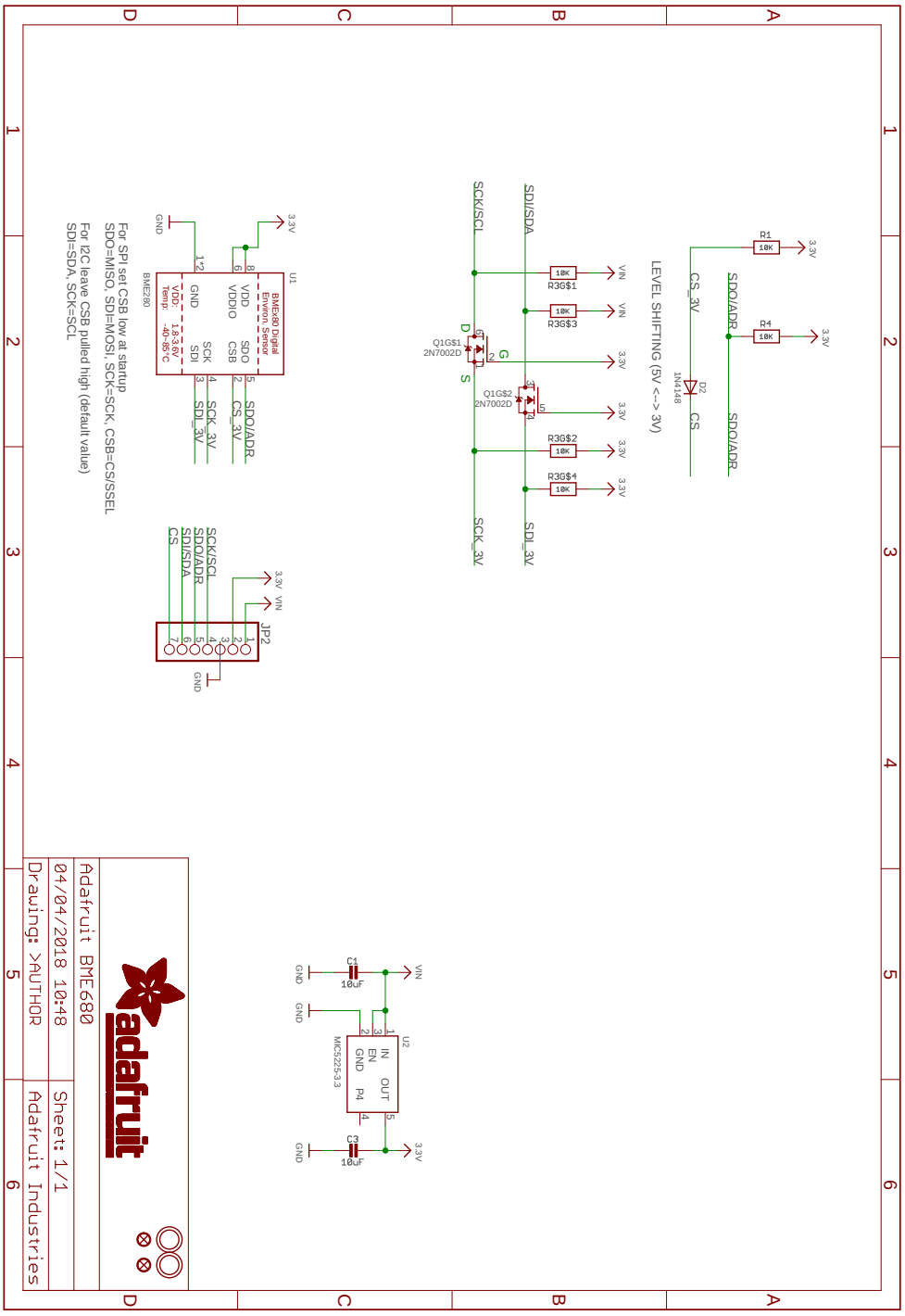
Appendix A

Schematics

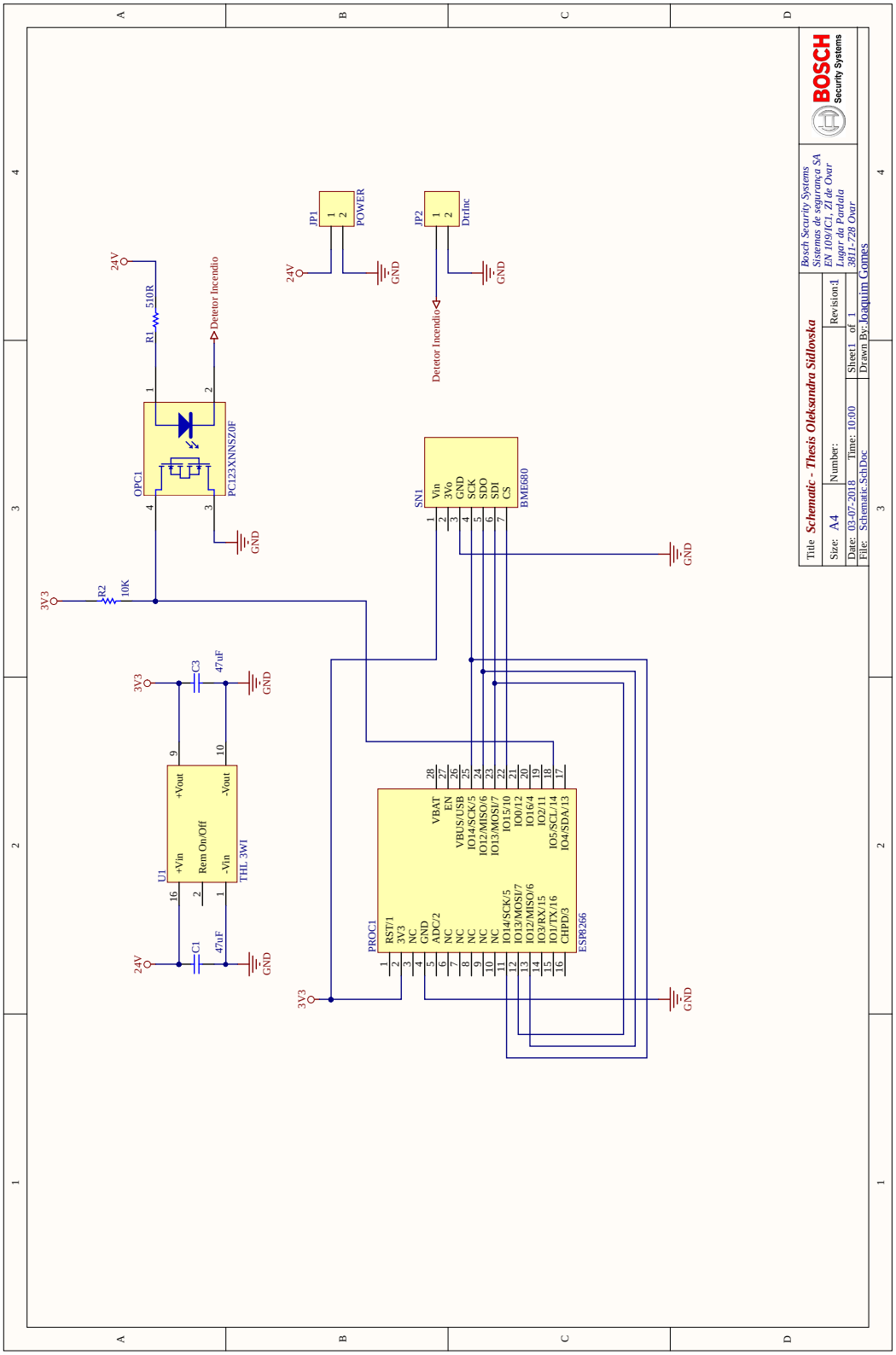
A.1 ESP8266 Schematic



A.2 BME680 Schematic



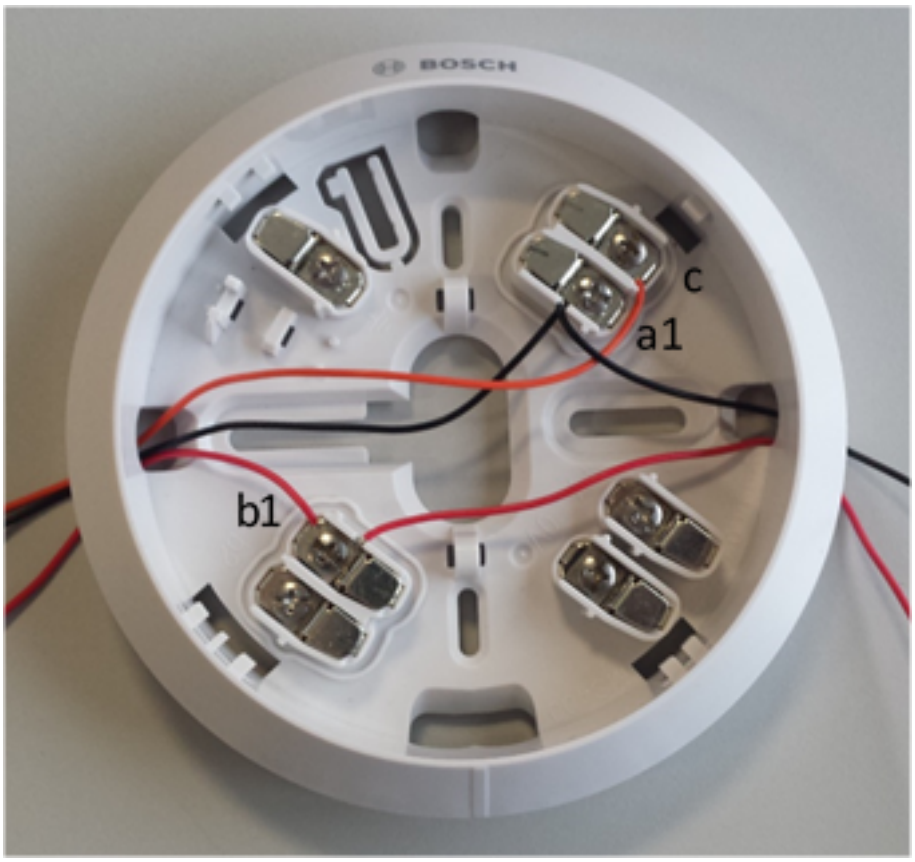
A.3 Schematic of the designed PCB



Appendix B

MS 400

B.1 MS 400 wire connection



Appendix C

Arduino Code

C.1 Programming code

```
/*...
    ********************************************************************************...

  BME680        ESP8266
   1 (VIN)      3v3
   2 (GND)      GND
   3 (SCK)      SCK
   4 (SDO)      MISO
   5 (SDI)      MOSI
   6 (CS)       15
 ********************************************************************************...
    */

#include <EEPROM.h>
#include "bsec.h"
#include "bsec_serialized_configurations_iaq.h"
#include <ESP8266WiFi.h>

#define STATE_SAVE_PERIOD  UINT32_C(360 * 60 * 1000) // 360 minutes - 4 ...
    times a day

// Helper functions declarations
void checkIaqSensorStatus(void);
void errLeds(void);
void loadState(void);
void updateState(void);

// Create an object of the class Bsec
Bsec iaqSensor;
uint8_t bsecState[BSEC_MAX_STATE_BLOB_SIZE] = {0};
uint16_t stateUpdateCounter = 0;
uint32_t millisOverflowCounter = 0;
uint32_t lastTime = 0;

String output;

// Channels ThingSpeak API key and Wi-Fi parameters
String apiKey = "DK50Q5TQUJYK7HIZ";
const char* ssid = "Optimus4G_2510";
const char* password = "8BF5E79AE68";
const char* server = "api.thingspeak.com";
```

```
38 WiFiClient client;
39
40 // Entry point for the example
41 void setup(void)
42 {
43   EEPROM.begin(BSEC_MAX_STATE_BLOB_SIZE + 1); // 1st address for the ...
       length
44   Serial.begin(115200);
45
46   WiFi.begin(ssid, password);
47
48   Serial.println();
49   Serial.println();
50   Serial.print("Connecting to ");
51   Serial.println(ssid);
52
53   WiFi.begin(ssid, password);
54
55   while (WiFi.status() != WL_CONNECTED)
56   {
57     delay(500);
58     Serial.print(".");
59   }
60   Serial.println("");
61   Serial.println("WiFi connected");
62
63   /* Setup button interrupt to trigger ULP plus */
64   pinMode(5, INPUT);
65   attachInterrupt(digitalPinToInterrupt(5), alarm_situation, FALLING);
66
67   /* Initialize the LED_BUILTIN pin as an output */
68   pinMode(LED_BUILTIN, OUTPUT);
69
70   iaqSensor.begin(15, SPI); /*Digital pin used as chip select*/
71   output = "\nBSEC library version " + String(iaqSensor.version.major) + "...
       ." + String(iaqSensor.version.minor) + "." + String(iaqSensor....
       version.major_bugfix) + "." + String(iaqSensor.version.minor_bugfix)...
       ;
72   Serial.println(output);
73   checkIaqSensorStatus();
74
75   iaqSensor.setConfig(bsec_config_iaq);
76   checkIaqSensorStatus();
77
78   loadState();
79
80   bsec_virtual_sensor_t sensorList[7] = {
81     BSEC_OUTPUT_RAW_TEMPERATURE,
82     BSEC_OUTPUT_RAW_PRESSURE,
83     BSEC_OUTPUT_RAW_HUMIDITY,
84     BSEC_OUTPUT_RAW_GAS,
85     BSEC_OUTPUT_IAQ_ESTIMATE,
86     BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE,
87     BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY,
88   };
89
90   iaqSensor.updateSubscription(sensorList, 7, BSEC_SAMPLE_RATE_ULP);
91   checkIaqSensorStatus();
92
93   // Print the header
```

```
94    output = "Timestamp [ms], raw temperature [ C ], pressure [hPa], raw ...
          relative humidity [%], gas [Ohm], IAQ, IAQ accuracy, temperature [...
          C ], relative humidity [%], Alarm State";
95    Serial.println(output);
96  }
97
98  // Function that is looped forever
99  void loop(void)
100 {
101   int alarmState = !digitalRead(5); //pin 5 reads the Fire Detector State
102   if (iaqSensor.run()) { // If new data is available
103     output = String(millis()) + " [ms]";
104     output += ", " + String(iaqSensor.rawTemperature) + " [ C ]";
105     output += ", " + String(iaqSensor.pressure) + " [hPa]";
106     output += ", " + String(iaqSensor.rawHumidity) + " [%]";
107     output += ", " + String(iaqSensor.gasResistance) + " [Ohm]";
108     output += ", " + String(iaqSensor.iaqEstimate) + " IAQ Index";
109     output += ", " + String(iaqSensor.iaqAccuracy);
110     output += ", " + String(iaqSensor.temperature) + " [ C ]";
111     output += ", " + String(iaqSensor.humidity) + " [%]";
112     output += ", " + String(alarmState);
113     Serial.println(output);
114     updateState();
115   digitalWrite(LED_BUILTIN, LOW);
116   delay(1000);
117   digitalWrite(LED_BUILTIN, HIGH);
118   delay(1000);
119     //send data to ThingSpeak
120     float t = iaqSensor.rawTemperature;
121     float p = iaqSensor.pressure;
122     float h = iaqSensor.rawHumidity;
123     float gr = iaqSensor.gasResistance;
124     float iaqe = iaqSensor.iaqEstimate;
125     float iaqa = iaqSensor.iaqAccuracy;
126     float fire = alarmState;
127
128     if (client.connect(server,80))  // "184.106.153.149" or api.thingspeak...
          .com
129     {
130         String postStr = apiKey;
131            postStr +="&field1=";
132            postStr += String(t);
133            postStr +="&field2=";
134            postStr += String(p);
135            postStr +="&field3=";
136            postStr += String(h);
137            postStr +="&field4=";
138            postStr += String(gr);
139            postStr +="&field5=";
140            postStr += String(iaqe);
141            postStr +="&field6=";
142            postStr += String(iaqa);
143            postStr +="&field7=";
144            postStr += String(fire);
145         client.print("POST /update HTTP/1.1\n");
146         client.print("Host: api.thingspeak.com\n");
147         client.print("Connection: close\n");
148         client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
149         client.print("Content-Type: application/x-www-form-urlencoded\n");
150         client.print("Content-Length: ");
151         client.print(postStr.length());
```

```
152         client.print("\n\n");
153         client.print(postStr);
154       }
155       client.stop();
156     } else {
157       checkIaqSensorStatus();
158     }
159 }
160
161 // Helper function definitions
162 void checkIaqSensorStatus(void)
163 {
164   if (iaqSensor.status != BSEC_OK) {
165     if (iaqSensor.status < BSEC_OK) {
166       output = "BSEC error code : " + String(iaqSensor.status);
167       Serial.println(output);
168       for (;;)
169         errLeds(); /* Halt in case of failure */
170     } else {
171       output = "BSEC warning code : " + String(iaqSensor.status);
172       Serial.println(output);
173     }
174   }
175
176   if (iaqSensor.bme680Status != BME680_OK) {
177     if (iaqSensor.bme680Status < BME680_OK) {
178       output = "BME680 error code : " + String(iaqSensor.bme680Status);
179       Serial.println(output);
180       for (;;)
181         errLeds(); /* Halt in case of failure */
182     } else {
183       output = "BME680 warning code : " + String(iaqSensor.bme680Status);
184       Serial.println(output);
185     }
186   }
187 }
188
189 void errLeds(void)
190 {
191   pinMode(LED_BUILTIN, OUTPUT);
192   digitalWrite(LED_BUILTIN, HIGH);
193   delay(100);
194   digitalWrite(LED_BUILTIN, LOW);
195   delay(100);
196 }
197
198 void loadState(void)
199 {
200   if (EEPROM.read(0) == BSEC_MAX_STATE_BLOB_SIZE) {
201     // Existing state in EEPROM
202     Serial.println("Reading state from EEPROM");
203
204     for (uint8_t i = 0; i < BSEC_MAX_STATE_BLOB_SIZE; i++) {
205       bsecState[i] = EEPROM.read(i + 1);
206       Serial.println(bsecState[i], HEX);
207     }
208
209     iaqSensor.setState(bsecState);
210     checkIaqSensorStatus();
211   } else {
212     // Erase the EEPROM with zeroes
```

```
213       Serial.println("Erasing EEPROM");
214
215       for (uint8_t i = 0; i < BSEC_MAX_STATE_BLOB_SIZE + 1; i++)
216         EEPROM.write(i, 0);
217
218       EEPROM.commit();
219     }
220   }
221
222   void updateState(void)
223   {
224     bool update = false;
225     if (stateUpdateCounter == 0) {
226       /* First state update when IAQ accuracy is >= 1 */
227       if (iaqSensor.iaqAccuracy >= 3) {
228         update = true;
229         stateUpdateCounter++;
230       }
231     } else {
232       /* Update every STATE_SAVE_PERIOD minutes */
233       if ((stateUpdateCounter * STATE_SAVE_PERIOD) < millis()) {
234         update = true;
235         stateUpdateCounter++;
236       }
237     }
238
239     if (update) {
240       iaqSensor.getState(bsecState);
241       checkIaqSensorStatus();
242
243       Serial.println("Writing state to EEPROM");
244
245       for (uint8_t i = 0; i < BSEC_MAX_STATE_BLOB_SIZE ; i++) {
246         EEPROM.write(i + 1, bsecState[i]);
247         Serial.println(bsecState[i], HEX);
248       }
249
250       EEPROM.write(0, BSEC_MAX_STATE_BLOB_SIZE);
251       EEPROM.commit();
252     }
253   }
254
255   /*!
256       @brief       Interrupt handler for press of a ULP plus button
257
258       @return      none
259   */
260   void alarm_situation()
261   {
262     /* We call bsec_update_subscription() in order to instruct BSEC to ...
          perform an extra measurement at the next
263       possible time slot
264     */
265     int alarmState = !digitalRead(5);
266     Serial.println("Alarme");
267     bsec_virtual_sensor_t sensorList[1] = {
268       BSEC_OUTPUT_IAQ_ESTIMATE,
269     };
270
271     iaqSensor.updateSubscription(sensorList, 1, ...
          BSEC_SAMPLE_RATE_ULP_MEASUREMENT_ON_DEMAND);
```

```
272     checkIaqSensorStatus();
273 }
```

C.2 bsec_serialized_configurations_iaq.h

```
#include "bsec_serialized_configurations_iaq.h"

const uint8_t bsec_config_iaq[304] =
    {0,6,4,1,61,0,0,0,0,0,0,0,24,1,0,0,40,0,1,0,137,65,
  0,63,0,0,64,63,205,204,76,62,0,0,225,68,0,192,168,
  71,0,0,0,0,0,80,10,90,0,0,0,0,0,0,0,0,21,0,2,0,0,244,
  1,225,0,25,10,144,1,0,0,112,65,0,0,0,63,16,0,3,0,10,
  215,163,60,10,215,35,59,10,215,35,59,9,0,5,0,0,0,0,0,
  1,51,0,9,0,10,215,163,59,205,204,204,61,225,122,148,
  62,41,92,15,61,0,0,0,63,0,0,0,63,154,153,89,63,154,153,
  25,62,1,1,0,0,128,63,6,236,81,184,61,51,51,131,64,12,0,
  10,0,0,0,0,0,0,0,0,0,131,0,254,0,2,1,5,48,117,100,0,44,
  1,151,7,132,3,197,0,144,1,64,1,64,1,48,117,48,117,48,
  117,48,117,100,0,100,0,100,0,48,117,48,117,48,117,100,
  0,100,0,48,117,100,0,100,0,100,0,100,0,48,117,48,117,48,
  117,100,0,100,0,100,0,48,117,48,117,100,0,44,1,44,1,44,1,
  44,1,44,1,44,1,44,1,44,1,44,1,44,1,44,1,44,1,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255
  ,255,255,255,255,255,255,255,255,255,48,117,0,0,0,0,240,156,0,0};
```

C.3 bsec_serialized_configurations_iaq.cpp

```
1 #include <stdint.h>
2
3 extern const uint8_t bsec_config_iaq[304];
```

C.4 MATLAB Analysis

Average calculation of Temperature, Pressure, Humidity, Gas Resistance and IAQ Estimate.

```
% Read temperature, pressure, humidity, gas resistance and IAQ
%estimate over the past 24 hours from a ThingSpeak channel and write
% the average to another ThingSpeak channel.

% Channel 490337 contains data from the BME680 sensor.
%The data is collected once every five minutes.
%Fields: 1 - temperature; 2 - pressure; 3 - himidity; 4 - gas resistance; ...
    5 - IAQ estimate.

% Channel ID to read data from
readChannelID = 490337;
% temperature Field ID
temperatureFieldID = 1;
% pressure Field ID
pressureFieldID = 2;
% humidity Field ID
humidityFieldID = 3;
% gas Field ID
gasFieldID = 4;
% IAQ Field ID
IAQFieldID = 5;

% Channel Read API Key
% If your channel is private, then enter the read API Key between the '' ...
    below:
readAPIKey = '';

% Get data for the last day from the channel.

temperature = thingSpeakRead(readChannelID,'Fields',temperatureFieldID,'...
    NumMinutes',1440,'ReadKey',readAPIKey);
pressure = thingSpeakRead(readChannelID,'Fields',pressureFieldID,'...
    NumMinutes',1440,'ReadKey',readAPIKey);
humidity = thingSpeakRead(readChannelID,'Fields',humidityFieldID,'...
    NumMinutes',1440,'ReadKey',readAPIKey);
gas = thingSpeakRead(readChannelID,'Fields',gasFieldID,'NumMinutes',1440,'...
    ReadKey',readAPIKey);
IAQ = thingSpeakRead(readChannelID,'Fields',IAQFieldID,'NumMinutes',1440,'...
    ReadKey',readAPIKey);

% Calculate the average temperature
avgtemperature = mean(temperature);
display(avgtemperature,'Average temperature');

% Calculate the average pressure
avgpressure = mean(pressure);
display(avgpressure,'Average pressure');

% Calculate the average humidity
avghumidity = mean(humidity);
display(avghumidity,'Average humidity');

% Calculate the average gas
avggas = mean(gas);
```

```
48 display(avggas,'Average gas');
49
50 % Calculate the average IAQ
51 avgIAQ = mean(IAQ);
52 display(avgIAQ,'Average IAQ');
53
54 fprintf(['Note: To write data to another channel, assign the write channel...
       ID \n',...
55     'and API Key to ''writeChannelID'' and ''writeAPIKey'' variables. ...
          Also \n',...
56     'uncomment the line of code containing ''thingSpeakWrite'' \n',...
57     '(remove ''%%'' sign at the beginning of the line.)']);
58
59 % To store the calculated average temperature, write it to a channel other
60 % than the one used for reading data. To write to a channel, assign the
61 % write channel ID to the 'writeChannelID' variable, and the write API Key
62 % to the 'writeAPIKey' variable below. Find the write API Key in the right
63 % side pane of this page.
64
65 % Replace the [] with channel ID to write data to:
66 writeChannelID = 512784;
67 % Enter the Write API Key between the '' below:
68 writeAPIKey = '8QQV6JTMRO33U2ZE';
69
70 % Learn more about the THINGSPEAKWRITE function by going to the ...
       Documentation tab on
71 % the right side pane of this page.
72
73 thingSpeakWrite(writeChannelID,[avgtemperature,avgpressure,avghumidity,...
       avggas,avgIAQ],'writekey',writeAPIKey);
```

Appendix D

Current calculation

D.1 Average current calculation

Normal Mode

FCP-320

Current [µA]
99,5451
99,6358
99,5487
99,6012
99,5418
99,5206
99,6225
99,5049
99,4985
99,5554
Average: 99,55745

FCP-320 and PCB

Current [mA]
26,3788
26,4008
26,4063
26,3702
26,4956
26,3804
26,4156
26,5596
26,3948
26,3255
Average: 26,41276

Fire Situation

FCP-320

Current [mA]
29,9986
29,0078
29,0406
29,043
29,0447
29,0482
29,0546
29,0567
29,0568
30,058
Average: 29,2409

FCP-320 and PCB

Current [mA]
61,1601
61,384
61,1955
61,5036
61,1678
61,2789
61,1531
61,3758
61,1805
61,548
Average: 61,29473

Appendix E

Technical draw

E.1 Technical draw of the new mechanical part