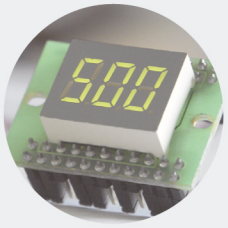
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Electronic product testing in the context of production

RÚBEN ALEXANDRE SEQUEIRA SARMENTO
novembro de 2021

POLITÉCNICO
DO PORTO

isep Instituto Superior de Engenharia do Porto

Electronic product testing in the context of production

Master in Electronics and Computer Engineering

Ruben Alexandre Sequeira Sarmento

Supervision
Manuel Carlos Malheiro de Carvalho Felgueiras

Academic year: 2020-2021

Instituto Superior de Engenharia do Porto
Departamento de Engenharia Eletrotécnica
Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto

Resumo

Teste de eletrónica no contexto de produção é um passo fundamental no ciclo de vida de um circuito antes do mesmo chegar ao seu consumidor, no entanto, facilmente é negligenciado durante a conceção do produto e pode-se tornar um custo adicional após serem detetados erros e até prejudicar a perceção da qualidade dos produtos da empresa se os mesmos forem detetados demasiado tarde.

Tecnimaster FL é uma empresa de produção de eletrónica com uma equipa de engenharia externa, sem equipa de teste dedicada e com um sistema de teste preparado para testes automáticos com uma *interface* de utilizador simples, como tal, um acordo foi realizado para desenvolver um teste para um produto novo não lançado usando os recursos da empresa.

Esta tese irá explorar um exemplo de teste desenvolvido para esse produto não lançado e demonstrar o que é necessário desenvolver, compreender e melhorar para testar a maioria do *hardware* do produto e garantir que um teste funcional e estrutural mínimo será realizado antes do mesmo chegar ao seu cliente. Durante a tese, múltiplos conceitos relacionados com teste e desenvolvimento de *hardware* serão explorados e finalmente uma análise de custo e tempo será realizada como prova do valor que um departamento dedicado a teste pode trazer a qualquer empresa que desenvolva produtos eletrónicos.

Palavras Chave:

Teste, Testabilidade, *Hardware*, Eletronica, Produção, Desenvolvimento, *Software*, ATE, Guia

Abstract

Electronic test in the context of production is a fundamental step in the life cycle of a circuit before it gets to its consumer, however it gets easily neglected during the conception of the product and can become an aggravated cost after errors are detected and even damage the perception of the quality of the companies' products if they are detected too late.

Tecnimaster FL is an electronics production company with an external engineering team, no dedicated test team and a test system ready for automatic tests with a simple user interface, as such an agreement was made to develop a test for a new unreleased product using the companies' resources.

This thesis will explore an example of a test developed for that unreleased product and demonstrate what needs to be developed, understood and improved in order to test most of the products' hardware and ensure that at least a minimal functional and structural test will be performed before the product reaches the client. During the thesis, multiple concepts related to test and hardware development will be explored and finally a cost and time analysis will be made as a proof of the value a department dedicated to test can bring to any company that develops electronic products.

Keywords:

Test, Testability, Hardware, Electronics, Production, Development, Software, ATE, Guide

Contents

List of Figures

List of Tables

Acronyms

ADC Analog to Digital Converter 11
ALS Ambient Light Sensor 71
AOI Automated Optical Inspection 1
ATE Automated Test Equipment 30
AXI Automated X-ray Inspection 29
BCB Bottom Connector Board 76
BoM Bill of Materials 86
BSP Bottom Spacer Board 76
BST Boundary Scan Test 32
CAN Controller Area Network 42
CPHA Clock PHAse 12
CPOL Clock POLarity 12
CPU Central Processing Unit 43
DAC Digital to Analog Converter 41
DC Direct Current 70
GND Ground 14
I2C Inter-Integrated Circuit 13
IR InfraRed 71
IC Integrated Circuit 26
ICT In-Circuit Test 30
IDE Integrated Development Environment 105
INF INternal Fault 49
JTAG Joint Test Action Group 32

LED Light Emitting Diode ix
MEMS Micro-Electro-Mechanical System 26
MISO Master Input Slave Output 12
MOSI Master Output Slave Input 12
OPAMP OPerational AMPlifier 18
PB Positioning Board 76
PCB Printed Circuit Board 30
PIPO Parallel In Parallel Out 24
PISO Parallel In Serial Out 23
PWM Pulse Width Modulation 11
RGB Red, Green and Blue 37
SCL Serial Clock Line 13
SCLK System CLocK 12
SDA Serial DAta Line 13
SIPO Serial In Parallel Out 25
SISO Serial In Serial Out 24
SMD Surface Mount Device 55
SPI Serial Peripheral Interface 12
SPL Sound Pressure Level 25
SS Slave Select 12
STRB Strobe 48
STX Start Transmission 42
SWIO Serial Wire Input and Output 56
TAMP Tamper 49
TAP Test Access Port 33
TCK Test ClocK 33
TDI Test Data In 32
TDL Test Description Language 43
TDO Test Data Out 32
THT Through Hole Technology 55
TMS Test Mode Select 32
TRIG Trigger 49

Acknowledgements

I give my gratitude to *Instituto Superior de Engenharia do Porto (ISEP)* and all of its professors that helped me develop in the academic, intellectual, critical thinking and resourcefulness domains, in particular to professor Carlos Felgueiras, who, with his experience, rigour and discern, helped me learn and grow to new heights in the field of electronic hardware and test development.

To the companies *Efacec, Energia, Máquinas e Equipamentos Elétricos S.A.* and *Tecnimaster FL, Sistemas Electrónicos, Lda.* I offer my thanks for alowing me to put in practise all the concepts I learned and for giving me the opportunity to prove myself and improve my skills along the way. My thanks go in particular to Mr. Alberto Ferreira, Mr. Bruno Marques and Eng. Rosa Castro from Tecnimaster for the continuous support and for helping me get all necessary materials along with helping me overcome some of my technical limitations during the development of the thesis. My gratitude also goes to as Eng. José Moutinho of Efacec for the great opportunity he offered for me to gain true experience in the field.

My greatest gratitude goes to my family, in particular my parents, José and Elizabete, my brother Ricardo, my niece Beatriz and my girlfriend Ana. Thank you from all my heart for all the encouragement and inspiration. You never ceased to believe in me and I am incredibly lucky for having you in my life. To Ana I give my special appreciation, for all the time we couldn't spend together and for your tireless encouragement and guidance through the duration of this project, I love you.

Thank you everyone!

Chapter 1

Introduction

This project was performed as an internship in an electronics production company in order to develop electronic production tests for assembled products before they leave the assembly line. This chapter contextualizes what motivated the company's need for the development of the project, as well as the objectives to be accomplished and the plan to accomplish those objectives. Finally, this chapter relates the structure of the rest of the thesis, as well as a summary for every chapter.

1.1 Context

Tecnimaster FL - Sistemas Electrónicos, Lda is a company that develops and produces electronic products ranging from car and home alarms to home automation and automobile electronic devices. The company is equipped with a production line as automated as possible, using *pick and place* and *wave soldering* machines to automatically assemble the products as well as Automated Optical Inspection (AOI) machines to visually inspect the assembly. The quality assurance and the engineering departments work together developing solutions that minimize the faults in the products as much as possible. However, most of the product and test development is done by outside companies, so internal products sometimes need longer development times to reach the proper level of *testability* and functionality. The test development itself is costly as well and performed by a different company than the one developing, so the process needs continuous feedback and Tecnimaster becomes an intermediary of these communications.

1.2 Objectives

As the costs of test development were higher for the company under the current circumstances, a need arose for an internal test and product developer that could elaborate the test start to finish and perform any necessary change to the original product, so it would be functional and testable. As such, a study will be carried out to implement a test on an unreleased product using the automated test equipment available in the company. This study will encompass the development of everything necessary for the test, along with any changes needed to perform for the product to be tested. In the end, the result will be a cost and time analysis for the development and implementation of a test developed exclusively within the company's engineering team by a non-specialized engineer.

1.3 Plan

An agreement was reached, so the project would be done in person from Monday to Friday for two hours each day and only using the equipment available in the factory. The original plan was set in place before beginning the development of the project and encompassed nine tasks to be performed for the duration of the implementation. However, shortly after the work began, and new needs observed, the plan suffered a change and four tasks were added that would give it a more realistic approach and end date. Table 1.1 lists all the tasks defined and figure 6.18 shows the Gantt chart for all the tasks and respective due dates, with the initial ones in light blue and the four additional ones in dark blue.

Table 1.1: All the initially defined tasks

Name	Duration	Time of Definition
1 - TEST-OK Hardware study	5 days	Initial
2 - TEST-OK Software and programming language study	5 days	Initial
3 - Check how the code of a test is developed	3 days	Initial
4 - Check hardware of a test and how to develop	5 days	Initial
5 - Develop a small application to see how the ATE works	8 days	Initial
6 - Develop the test logic	8 days	Initial
7 - Learn how to use Altium	5 days	Added
8 - Develop the hardware for the full product test	25 days	Initial

Name	Duration	Time of Definition
9 - Correct design and study BoM and PCB prices	10 days	Added
10 - Change XTEC RGB Siren to improve testability	3 days	Added
11 - Develop the software for the full product test	15 days	Initial
12 - Assemble test module hardware	8 days	Added
13 - Validate the final test solution	30 days	Initial
Total time	**130 days**	

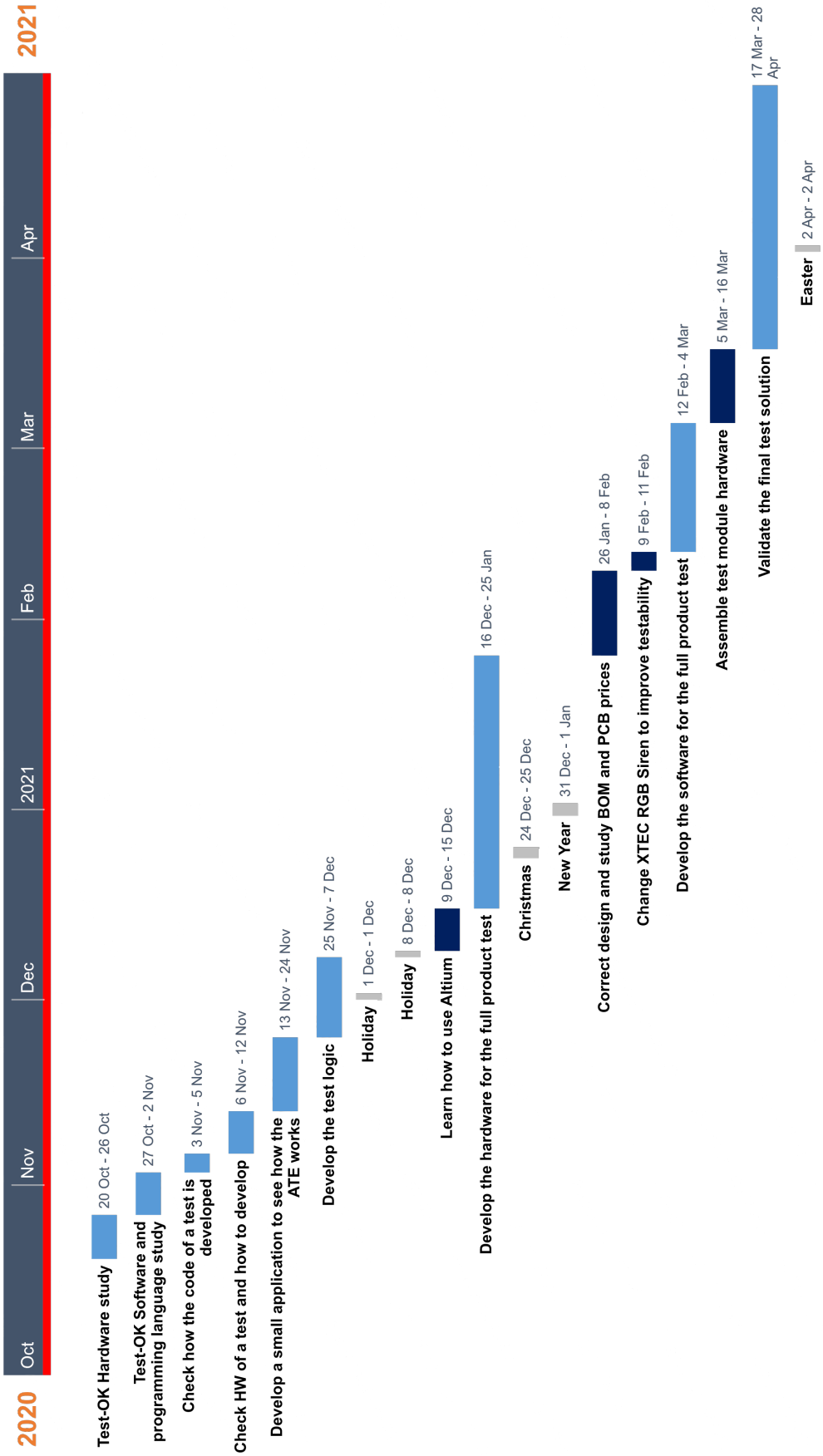


Figure 1.1: Gantt chart for the project

After the main tasks are performed, there will be an analysis made to allow any company to understand if a test development department is valuable or irrelevant to the quality of the product and if the inclusion of that department will, in the end, be a good investment or not.

1.4 Structure

This thesis is organized under the following structure:

Chapter one, *Introduction*, introduces the project and contextualizes its development in the company.

Chapter two, *Fundamental Concepts*, presents concepts associated to the development, production and test of electronic circuits

Chapter three, *State of the Art*, presents the current state of automated test technology.

Chapter four, *Solution Proposal*, presents the technology to use in the development of the tests as well as a summary of the product to test and each functional block to test. Finally, it presents the requirements and a functional diagram of the test to implement.

Chapter five, *Solution Implementation*, reports how the development was made and how each test was implemented on a logical level, as well as on the level of necessary hardware and software.

Chapter six, *Solution Validation*, presents the various scenarios used to validate the solution and certify how well the requirements were met as well as the accomplishment of the proposed plan.

Chapter seven, *Conclusion*, presents the conclusions of the project developed as well as possible directions for future developments.

References, lists the sources of information used throughout the document.

Annexes, compiles all the documents and more lengthy work for consultation.

Chapter 2

Fundamental Concepts

The test is one of the most important stages in the life cycle of an electronic circuit, as it is the one that verifies that the product is working correctly after it has finished its manufacturing stage. In this chapter, the fundamental concepts for the understanding of the basis where the whole project stands upon will be presented, shining a particular spotlight on the subjects related to the test of electronic products and typical circuits used in the development of electronic products.

2.1 Electronic circuit life cycle

The life cycle of a circuit is a path that shows all necessary phases for the conception, production and use of an electronic product. Figure 2.1 shows a diagram of the life cycle as it is.

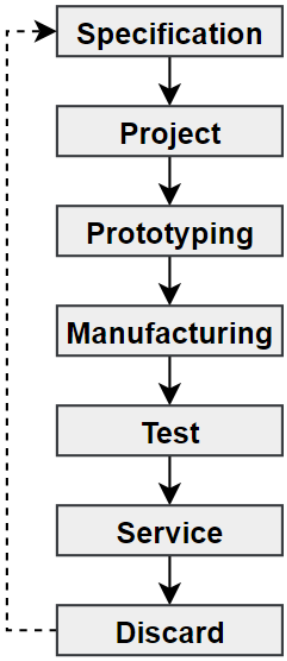


Figure 2.1: Life cycle of an electronic circuit

The first phase, designated *specification*, is where the characteristics and operation of a circuit are defined and for what purpose it is conceived. The second phase, designated *project*, is where the electronic circuit is developed in order to fulfill the objectives defined in the specification phase. This phase involves the simulation and the development of mechanisms to test the circuit. The third phase, designated *prototyping*, is where it is physically implemented all that was defined in the project phase. This phase serves to verify, under real conditions, if the circuit fulfills the operational characteristics defined in the specification phase. The fourth phase, designated *manufacturing*, is where are defined the assembly procedures and performed all the intermediate verifications in order to certify the operability of the final product. The fifth phase, designated *test*, is where it is validated, after the product has passed all manufacturing verifications, if the circuit fulfill all the operational requirements defined in the specification phase. The sixth phase, designated *service*, involves the verification of the proper and continuous operation of the circuit according to what was defined in the specification phase and validated in the test phase. The seventh phase, designated *discard*, involves the process of reutilization or destruction of the circuit in order to not impact the environment negatively [8]. The dotted line on the figure is not yet in practice and is one of the most challenging problems facing electronic production, as the possibility of a recyclable circuit would prove to be advantageous to both the production and cost of a circuit as well as for its sustainability, as most electronic products become garbage after being discarded.

A common mistake given this life cycle is to separate the specification of the project from the specification of the test and as such only worry about the test after prototyping. This is a poor idea, as the preparation for the test is as important as the product specification and should be part of the life cycle of the product from day one. The proper test specification helps in prototyping and in manufacturing and can even help during service to debug the product and repair any possible problem in more expensive or critical electronic applications. The lack of proper specification in the project stage will increase cost and time of production and hinder the time to market of the product, as well as increase the risk of undetected errors reaching the costumer during the service stage. This will also increase the cost to at least double the original value if the product is not properly specified for test, especially if the test is only conceived after manufacturing of the product.

2.2 Test and testability

The test of electronic circuits is a process that determines if the Unit Under Test (UUT) shows acceptable operational deviations and if it accomplishes the functioning requirements before reaching the client.

The test is used to detect flaws on a board. Those flaws can be defined as *defects* when they are physical flaws, as *faults* if they are more abstract or project based flaws or *errors* if they are deviations from the expected behavior of the board.

In an ideal test, all the circuit flaws are detected and repaired before entering the service phase (view section 2.1). However, a real test does not detect all the flaws of a circuit, some flaws, in a very reduced percentage, end up undetected and reach the client. The test involves the stimulation of a circuit or some part of it, with the intention to verify if the stimuli result in responses that correspond with the correct functioning of the circuit.

The ease of implementation of a test in a circuit will depend on the testability of its nodes. *Testability* of a node in a circuit is a joined measure determined by the controllability and observability of that node. *Controllability* of a node is defined by the ease with which one can change the signal of that node during the test through the primary inputs. *Observability* of a node is defined by the ease with which one can observe the signal of that node during the test through the primary outputs. The primary inputs and outputs of a circuit are the external accesses to the nodes of that circuit (ex: connectors on a board). A factor that may difficult controllability or observability of a circuit is the *impact in the operation*, that is characterized by the impact in normal operation caused by the implementation of means to improve the controllability or observability of a node, so

care should be taken to not hinder the functionality of a circuit in favor of improved testability [8]. Figure 2.2 shows a simple Venn diagram that illustrates how testability depends on controllabiity and observability.

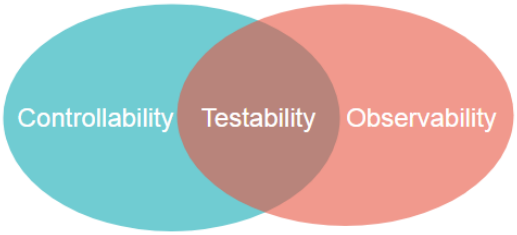


Figure 2.2: Testability Venn diagram

2.2.1 Types of test

When a test is developed for an electronic product, it is necessary to evaluate the ability to detect faults against the complexity necessary for the test to detect those faults. A truly extensive test, which would validate all circuit operations with all possible combinations, will be very time-consuming and virtually impossible to implement. As such, a test developer must, with his resources in mind, develop a test that validates the majority of behavioral requirements of a circuit in the least amount of time possible. We can divide tests in the following categories:

- **Functional Test** is any test that only verifies the functions of the circuit, that is, the expected behaviors of the UUT. It is the most complex form of testing, also being the most expensive and the more difficult to implement. It uses the primary inputs and outputs of the board and validates the expected operations of the circuit, finding faults at the rate that any output deviates from the expected behavior. An example of a functional test is a test to an AND logic gate, in which the whole truth table is verified by controlling the state of its inputs and observing the state of the output.

- **Structural Test** is a test that validates the structure connections of the circuit. An example of how the test works is by stimulating a logic state in each node and observing the propagation of that signal in the remaining nodes of the circuit. It is an extensive test with great ability to detect defects as it validates all the connections between the components of the circuit. An example of a structural test is a short circuit test between two nodes that must be electrically isolated from each other.

- **Parametric Test** validates behavioral and time parameters of the components in the circuit. It is more complex to develop and implement at the level of instrumentation, being that complexity dependent on the need of

precision of the circuit to test. An example of a parametric test is a test to the value of a resistor assembled between two nodes of a circuit. [8]

2.2.2 Types of circuits

Electronic circuits can be classified at the level of the signal they process as such:

- **Digital Circuit** – circuits that only process signals that can assume a finite number of values (ex: circuits that process binary signals);
- **Analogical Circuit** – circuits that only process signals that can assume an infinite number of values within a certain range (ex: circuits that process sinusoidal signals);
- **Quasi-Digital Circuit** – circuits that use signals with a finite number of values to process or transmit data that can assume an infinite number of values (ex: a circuit that uses Pulse Width Modulation (PWM) to transmit information);
- **Mixed-Signal Circuit** – circuits that process digital, analogical or quasi-digital signals that interact with one another (ex: Analog to Digital Converter (ADC)).

They can also be classified at the level of the relationship between inputs and outputs as such:

- **Combinational Circuit** – circuits for which the present output values only depend on the present input values;
- **Memory Circuit** – circuits for which the present output values depend on the past input values and, possibly, also present input values; [8]

2.2.3 Types of circuit access

In order to test a circuit, access to it is necessary in some way, so a measuring or stimulating equipment can be used to either propagate a signal or directly input it to the UUT. As such, access methodologies can be classified on weather they directly stimulate the circuit or use propagation for the same effect and also how that stimulation is performed as such:

- **Direct/physical access**, accesses the primary inputs and outputs or the test points of the UUT using probes or other physical connections through some kind of hardware interface;

- **Direct/electronic access**, accesses the nodes of a circuit using dedicated access circuits within the UUT;

- **Indirect access**, occurs when the way to interact with a given node is only by propagating a signal through the circuit.[9]

2.3 Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a communication protocol that uses only four lines to communicate with multiple systems. SPI is a hierarchical protocol with one *Master* and a number of *Slaves* equal to the number of Slave Select (SS) lines. This is a *Full Duplex* protocol, that is, the sending and receiving of information are performed at the same time. As such, to receive the response from a SPI system after sending data, it is necessary to communicate once more with the system in order to receive the most recent response. As figure 2.3 illustrates, three of the four lines (Master Output Slave Input (MOSI), Master Input Slave Output (MISO) and System CLocK (SCLK)) are shared by all the systems with which it communicates, while SS is connected one to each *Slave* in the system.

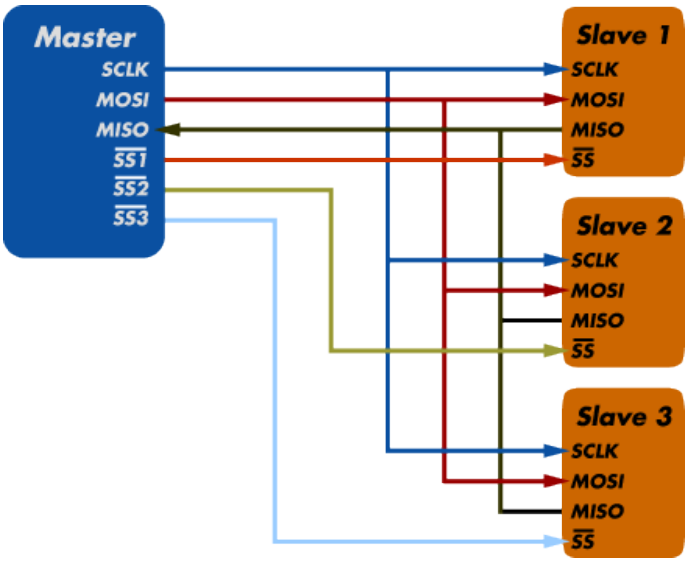


Figure 2.3: Example of a connection of four systems using SPI [1]

The bits sent through pins SCLK, MOSI and MISO must be synchronized in order to send information correctly. The line SS does not need to be synchronized in most cases, only needing a change in the logic state to identify the *Slave*. SPI modes are defined by the configurations of the clock's polarity [Clock POLarity (CPOL)] and phase [Clock PHAse (CPHA)]. These configurations determine the clocks' logic state when there is no communication and if

the communications are performed in the rising edge (CPHA=0) or falling edge of the clock (CPHA=1). Figure 2.4 exemplifies a SPI communication in mode 0.

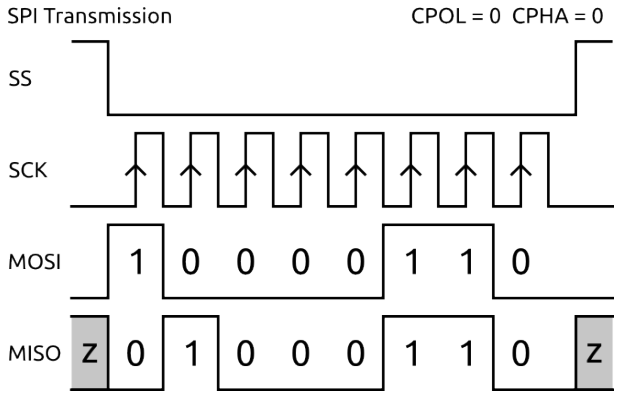


Figure 2.4: Example of communication using SPI in mode 0 [2]

2.4 Inter-Integrated Circuit (I2C)

Inter-Integrated Circuit (I2C) is a serial communication protocol with two wires, one designated Serial DAta Line (SDA) and the other Serial Clock Line (SCL). The protocol is based in a master-slave hierarchy, with support of multiple slaves for each master and the possibility to use more than one master is a given communication circuit. It is a *Half Duplex* protocol, that is, masters cannot send and receive data at the same time. Figure 2.5 shows a typical application circuit of I2C.

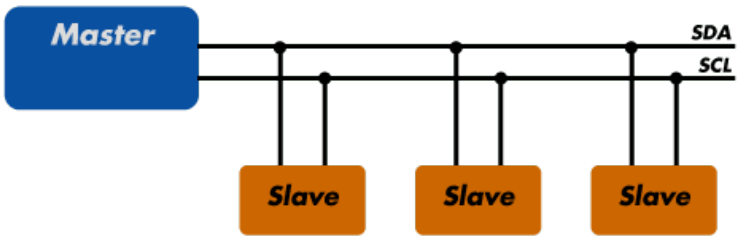


Figure 2.5: Example of a connection of four systems using I2C [3]

The circuit is built-up by connecting all devices in parallel to the SDA and SCL lines, as such all devices receive data at the same time. Due to this, each device has a specific address to which communications are sent to by the master. In order to communicate with a certain device, the master sends a start bit, followed by the address of the slave device, followed by a bit that indicates if the operation is to read or write followed by the data address to read or write. The slave answers each byte with an acknowledge and can also send back feedback bytes before each acknowledge in case of a read operation. The communication

thread is ended by a Stop bit, which must be preceded by a not acknowledge from the master in case the slave has sent any feedback. Figure 2.6 shows a simplified representation of a communication thread between the master and the slave.

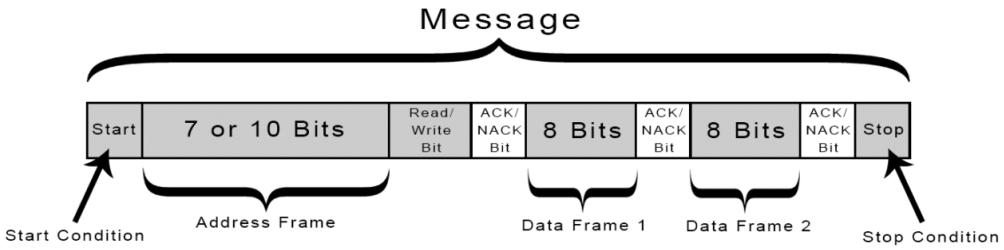


Figure 2.6: Example of a communication thread using I2C [4]

I2C has four speed modes known as Standard mode (100 kHz), Fast mode (400 kHz), Fast mode Plus (1 MHz) and High Speed mode (3.4 MHz) for bidirectional data transfers and an additional Ultra Fast mode (5 MHz) for unidirectional data transfers. The total amount of devices connected in a I2C circuit depend not only on the bit size of the address but also of the distance between the master and the slaves. Typical devices will have either 7 bit or 10 bit addresses, so the amount of different slaves can be either 127 or 1024. [10]

2.5 Typical circuit configurations

In this section, some common circuit configurations will be explained and illustrated, along with their typical applications. This section becomes necessary as these circuit configurations will be mentioned during the rest of the dissertation and a simple guide will help to understand how each works.

2.5.1 Types of digital outputs

Digital circuits can have multiple types of outputs that have differing properties and advantages on the interface with the rest of the circuit and with other external electronics as well. The type of output will be relevant to understand how a circuit and its internal logic work.

- **Open source or open emitter** is a type of output that powers the load using the internal logic voltage for the circuit that interfaces the load but can't connect that load to Ground (GND). Usually this type of output has a low current output and can't really power other circuits without an interface circuit in between to ensure no hardware is damaged. Figure 2.7 shows an example of an open emitter output.

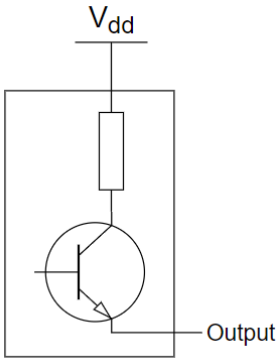


Figure 2.7: An open emitter output example

- **Open drain or open collector** is a type of output that instead of powering a load that connects to it when the logic value is high, it connects that load to GND and cannot power it using the internal voltage. Due to how it works, it is usually assembled in a way that can handle much higher current values, so it is useful for applications that require the load to be controlled without more complex interfacing circuits. Figure 2.8 shows an example of an open collector output.

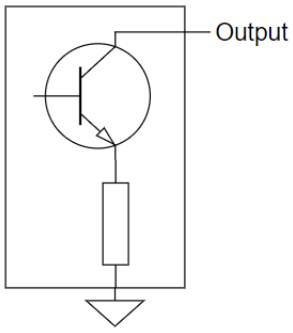


Figure 2.8: An open collector output example

- **Totem pole and Push and pull** are similar types of output that can both power the load when the logic value is high and connect it to GND when the logic value is low. There is a slight difference between both types of assembly as totem pole drives lower currents and is usually limited to voltages up to 5 V, while the push and pull assembly can drive higher currents and much higher voltages. Figure 2.9 shows examples of both configurations.

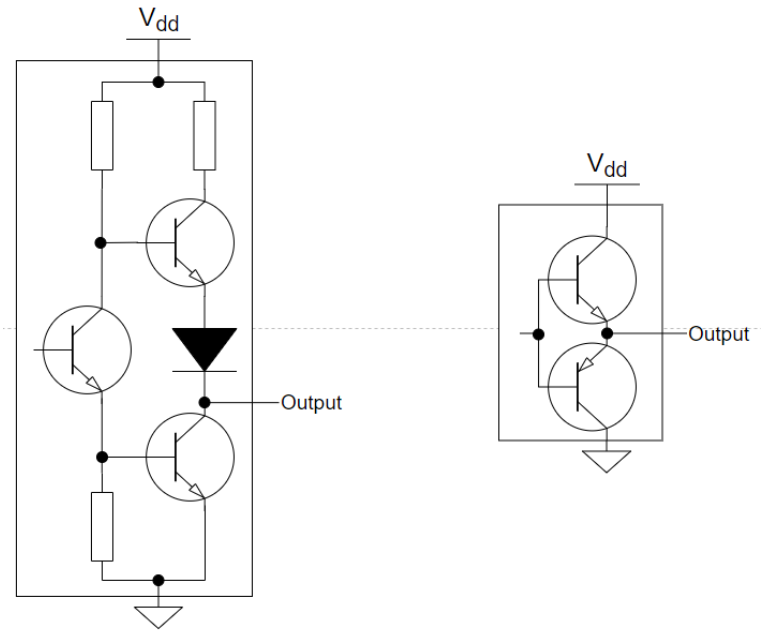


Figure 2.9: A totem pole example on the left and a push and pull example on the right

2.5.2 Pull-up and pull-down resistors

Pull-up and *pull-down* resistors are resistors that connect a branch to the positive power source in the case of a *pull-up*, or to either GND or a negative power source in the case of a *pull-down*. They are used to keep the potential in a branch at a certain voltage level in a circuit. They perform an operation called *wired and* in case of a *pull-up* and *wired or* in case of a *pull-down*. As such, in case of a *pull-up*, the output will be logic high unless the input is logic low and in case of a *pull-down* the output will be logic low unless the input is logic high. Figure 2.10 shows a *pull-up* on the left and a *pull-down* on the right.

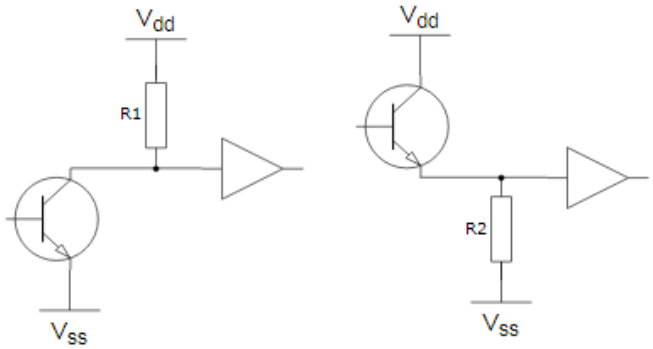


Figure 2.10: A pull-up resistor on the left and a pull-down resistor on the right

2.5.3 High-side and Low-side switches

High-side and *Low-side* switches are definitions given to switches that connect a branch of a circuit to the positive power source or to the negative (or GND) power source respectively. Transistors are regularly used to switch circuits on and off, as such two simple configurations can be used to different effects for this purpose. A *Low-side* switch connects the circuit to GND when the transistor is saturated, having the load of the circuit connected between the power and the collector or the drain of the transistor. A *High-side* switch connects the power to the circuit when the transistor is saturated, having the load connected between the collector or the drain of the transistor and the GND. *High-side* configuration is less used, as it has the limitation of needing the signal and load voltages to be the same in order to function without a driver and is more useful for powering entire circuits rather than to switch actuators or circuits on and off. Figure 2.11 shows the two configurations side by side. [11]

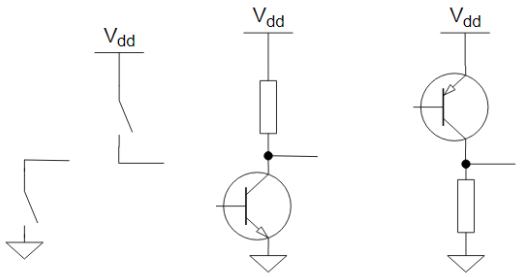


Figure 2.11: Simple representation of a Low-side and High-side switches on the left and practical implementation of the same thing using transistors on the right. On the left of each pair is a Low-side switch and on the right of each pair is a High-side switch

2.5.4 H-Bridge

An H-Bridge is a type of electronic circuit that uses at least four transistors to drive current in different directions on a component with two contacts. It works by arranging a parallel of two pairs of transistors in series. It works by activating two opposite transistors at a time (Q1 and Q4 to power in one direction or Q2 and Q3 to reverse it) in order to have different directions on the component at the center of the circuit. Generally, it is used to drive motors and control the direction of their rotation. Figure 2.12 shows how the circuit is assembled.

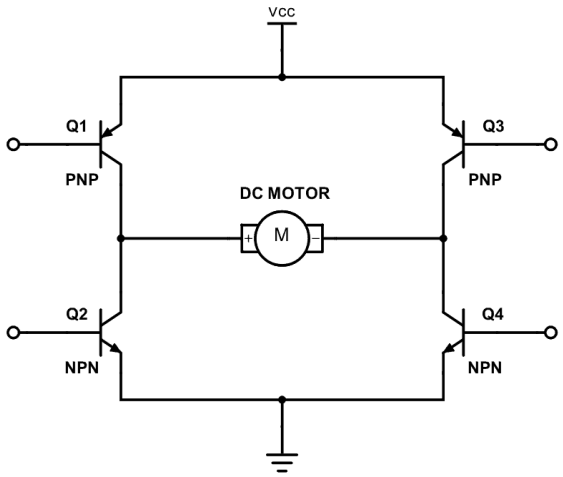


Figure 2.12: An H-Bridge circuit

2.5.5 Operational amplifier circuits

OPerational AMPlifier (OPAMP) are very versatile linear components usually used in analog circuits that require signal processing or a mathematical operation to be performed. In the early history of their use, OPAMPs were used mainly as comparators or in order to perform mathematical operations such as adding, subtracting, differentials or integration. The component itself consists of at least three functional pins and two power source pins. The functional pins are the inverting input (marked with a minus sign), the non-inverting input (marked with a plus sign) and the output. The two inputs have high impedance, and the output can both sink and source a signal, as it is the difference between both inputs. The OPAMP only responds to the difference between its inputs and not to their potential, as such if the same voltage is applied to both inputs, the output will be zero. A few key characteristics can be said about the ideal OPAMP that make their study and use a little easier to understand:

- **Open loop gain is infinite** - if the circuit has no positive or negative feedback, the output will be amplified to the maximum value;
- **Input impedance is infinite** - no current flows into the inverting and non-inverting inputs;
- **Output impedance is zero** - it functions as a perfect voltage source, providing as much current as necessary;
- **Bandwidth is infinite** - it can amplify any input signal regardless of its frequency;

- **Offset voltage is zero** - the output voltage is zero when the difference between the inverting and non-inverting inputs is zero, the same or when they are grounded.

These characteristics are ideal, as the actual component doesn't have such high or low characteristics, however, the reality is still very close to these approximations and these rules still apply to physical circuits. The OPAMP is designed to be assembled with resistors or capacitors between its input and output terminals in such a way that these feedback components result in a functional operation of the amplifier, the following list details some of the most common operational assemblies used with OPAMPs.[5]

- **Comparator** is a circuit where the inputs of the OPAMP are connected to two different voltage signals and the OPAMP is powered by two different voltage values, V_{DD} and GND for example. It is used to compare the input voltage (V_{in}) to a reference voltage (V_{ref}) and switch the output (V_{out}) if the input voltage is above the reference. The state of V_{out} depends on the input pins where V_{in} and V_{ref} are connected. If V_{in} is connected to the non-inverting pin and it is higher than V_{ref}, V_{out} changes state to V_{DD} and if it is below, it goes to GND. If V_{in} is connected to the inverting pin, when the input is higher, V_{out} goes to GND and when it is lower, V_{out} goes to V_{DD}.[5] Figure 2.13 illustrates the circuit and the variables mentioned.

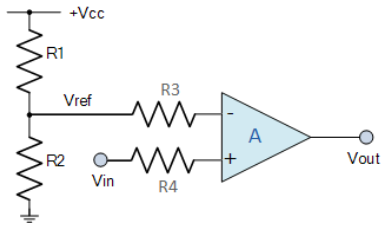


Figure 2.13: Comparator circuit [5]

- **Inverting Amplifier** is a circuit conceived to facilitate the use of the OPAMP as an amplifier without having to deal with the infinite gain it has in open loop. In order to do this in this configuration, a feedback resistor (R_F) is assembled between the output and the inverting input, reducing the gain of the amplifier. This effect, called Negative Feedback, forces the differential input voltage to be 0 V, creating as such a closed loop circuit with a "summing point" at the inverting input. As a closed loop circuit, the gain can be controlled more easily by changing the value of R_F, however, since the inverting input has a signal resulting from the sum of the input signal with the output signal, in order to preserve the input signal, an additional R_{in} resistor is necessary between the signal input and the inverting input.

Also resulting from this closed loop configuration, the non-inverting and inverting inputs of the OPAMP come to the same voltage potential, which in the example below makes for a "Virtual earth", as the non-inverting input is at the potential of the GND. This "Virtual earth" also causes the input resistance of the amplifier to be equal to R_{in}, which means the gain of the amplifier is set by the ratio of the two resistors. The gain is calculated with the following expression: $Gain = -\frac{R_F}{R_{in}}$. [5] Figure 2.14 illustrates the circuit and the variables mentioned.

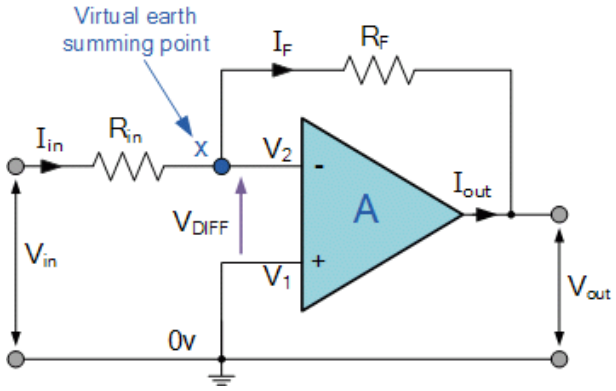


Figure 2.14: Inverting Amplifier [5]

- **Non-Inverting Amplifier** is also a circuit conceived to use the OPAMP as an amplifier. In this configuration, the input signal is applied directly to the non-inverting input of the OPAMP, which makes the output gain positive and the output signal in phase with the input signal. The closed loop feedback on this amplifier circuit is achieved by feeding back the output through a R_F and R_2 voltage divider connected to the inverting input of the OPAMP. This configuration makes a circuit with an infinite input impedance and zero output impedance. In this configuration, the same rules of the inverting amplifier apply, as no current flows to the inputs of the OPAMP and the voltage in the inverting input is the same as on the non-inverting one, as such, the V_1 junction has the same value as the input signal and the R_F and R_2 voltage divider is the only determinant for the output gain with the following expression: $Gain = 1 + \frac{R_F}{R_2}$[5] Figure 2.15 illustrates the circuit and the variables mentioned.

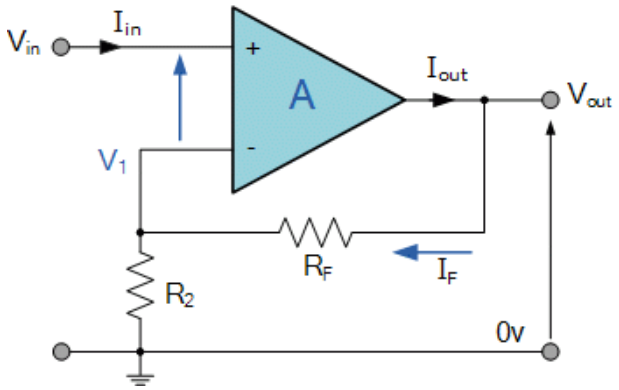


Figure 2.15: Non-Inverting Amplifier [5]

- **Voltage Buffer** is a similar circuit to the non-inverting amplifier, using the same rules to achieve a unitary gain amplifier with the same impedance properties as the non-inverting one. This is done by making R_F of the previous circuit zero and R_2 infinite, feeding back all the output to the inverting input of the amplifier. Due to its infinite input impedance and zero output impedance, this circuit is used as an ideal constant voltage source or voltage regulator.[5] Figure 2.16 illustrates the circuit and the variables mentioned.

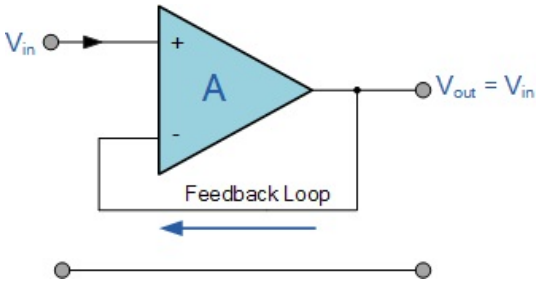


Figure 2.16: Voltage buffer [5]

- **Differential Amplifier** is an amplifying circuit that uses both inputs on the OPAMP to achieve a differential amplification. This means that by connecting two different signals, one to each input on the OPAMP, the output will be proportional to the difference between inputs, making it a subtracting amplifier. In order to make its implementation more simple, the resistor values can be as follows: $R_1 = R_2$ and $R_3 = R_4$ and as such the gain is given by the following expression: $Gain = -\frac{R_3}{R_1}(V_2 - V_1)$. If all resistors have the same value, this circuit becomes a unity gain subtractor.[5] Figure 2.17 illustrates the circuit and the variables mentioned.

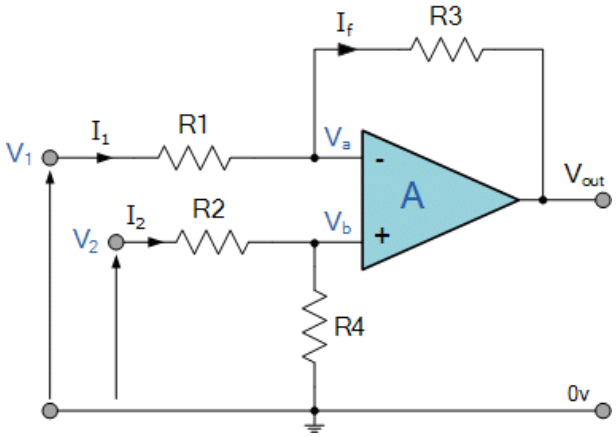


Figure 2.17: Differential Amplifier [5]

- **Instrumentation Amplifier** is a circuit commonly used to amplify very small signals from sensors. Derived from the differential amplifier, it has a very high gain, high input impedance and a single ended output. Unlike other amplifier circuits which have their closed loop gain determined by feedback between the output and one of the inputs, this amplifier has internal feedback resistors that are isolated from its signal inputs. The circuit is made of a differential amplifier with two buffers with gain $1 + 2\frac{R_2}{R_1}$ on each of its inputs. Since both buffers are closed loop negative feedback amplifiers, the values of V_a and V_b will be the same as V_1 and V_2 respectively, as OPAMPs have no current entering their inputs, the current on the mesh will be the same for R_1 and both R_2 which makes the voltage drop in R_1 the same as the difference between V_1 and V_2, as such the differential gain can be changed by changing the value of R_1. With the differential amplifier acting as a subtractor, the output will be the difference between the input signals V_1 and V_2 amplified by the gain, which can be unitary. The general expression for the gain in this circuit is the following: $Gain = (V_2 - V_1)[1 + 2\frac{R_2}{R_1}](\frac{R_4}{R_3})$.[5] Figure 2.18 illustrates the circuit and the variables mentioned.

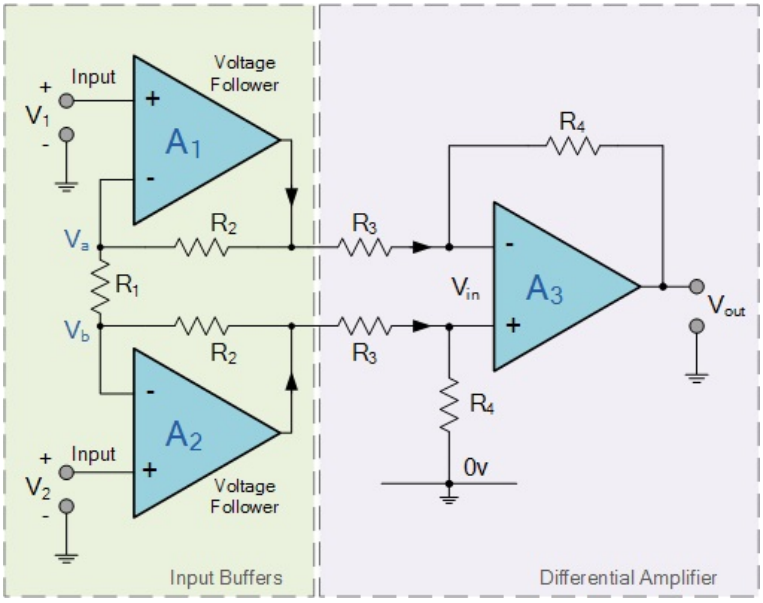


Figure 2.18: Instrumentation amplifier circuit [5]

2.5.6 Shift registers

These are sequential devices that load the data present in their inputs and shifts it to their outputs on every clock cycle. These devices are made using several latches, one for each bit and usually eight, with the output of the first latch connected to the input of the second and so on for every latch in the arrangement. All the latches on the register use the same clock signal and usually the device has a reset pin. Shift Registers are commonly used for data storage or for the movement of data and data can be shifted either right to left, left to right or bidirectional. All the examples in this subsection will eb given as left to right. As latches and flip-flops are similar sequential circuits, a simple shift register can be made using one flip-flop for each data bit, arranged as the latches would be, the difference will be the data will be dependent on the clock input and should, therefore, be synchronized. The following are the four types of shift register arrangement and how each one works using flip-flop examples:

- **Parallel In Serial Out (PISO)** shift registers function by having all the data enter the inputs in parallel and simultaneously, the data is then read sequentially to the output using multiplexers to time each bit properly with the clock, which is required only for the serial data. It can be used to convert multiple lines of data into a single serial stream. Figure 2.19 shows a simple PISO circuit;

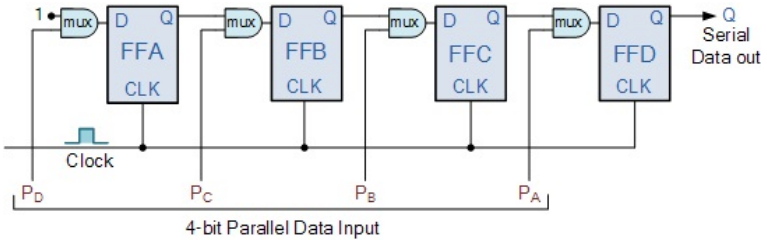


Figure 2.19: PISO example circuit [5]

- **Parallel In Parallel Out (PIPO)** shift registers function as a temporary storage for bits that enter and leave simultaneously through the use of a clock pulse. It is the simplest arrangement of the four as it only needs the input, output and clock pins and no need for proper synchronization. It can be used as a temporary data storage or memory. Figure 2.20 shows a simple PIPO circuit;

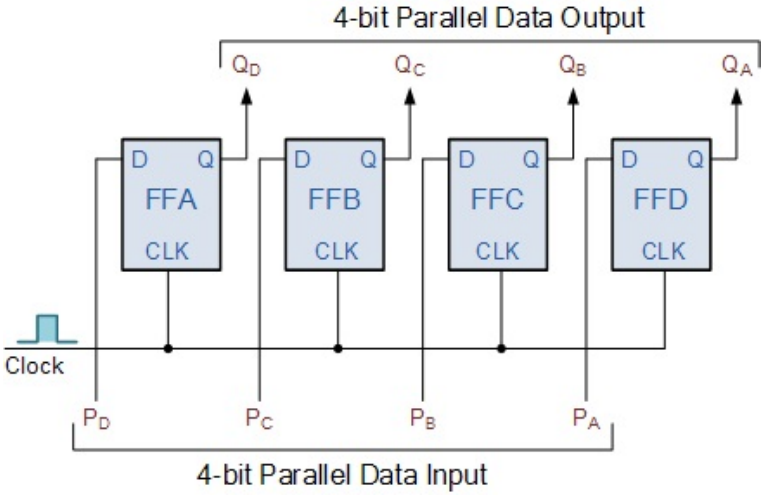


Figure 2.20: PIPO example circuit [5]

- **Serial In Serial Out (SISO)** shift registers function as a delay for data to go through sequentially. The data enters though the shift register and flows trough each flip-flop one clock cycle at a time. It is used also as a temporary data storage and as a delay for the serial data that goes through it. Figure **??** shows a simple SISO circuit;

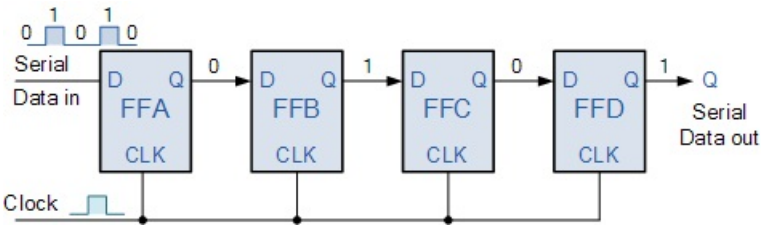


Figure 2.21: SISO example circuit [5]

- **Serial In Parallel Out (SIPO)** shift registers function by having all the data enter trough a serial input and the outputs change their logic value depending on the serial value they are pointing to. As such, if on the first clock cycle a input high is present and the input is low on the rest of the cycles, the first output will be high on the first clock cycle and the remaining will be low, the second output will be high on the second clock cycle and the remaining will be low, including the first one, and so forth for the rest of the outputs. It is normally used to convert serial data into input values. [5] Figure 2.22 shows a simple SIPO circuit.

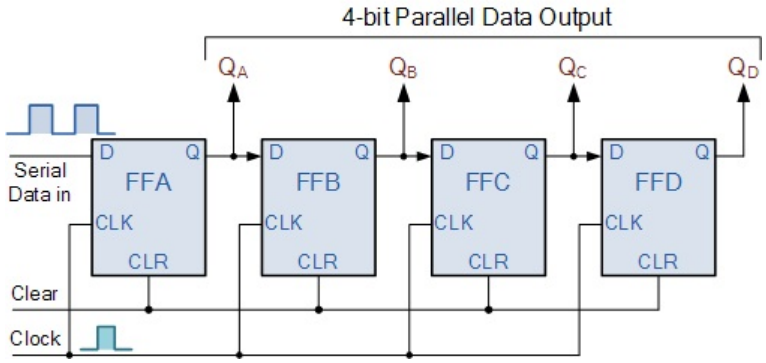


Figure 2.22: SIPO example circuit [5]

2.6 Microphones

Microphones are devices that translate an acoustic signal into an electrical one. Composed by a diaphragm and some type of receiving technology, they are used to convert Sound Pressure Level (SPL) into voltage. There are five types of microphone receivers: piezoresistive, piezoelectric, electromagnetic, electrodynamic and electrostatic.

- **Piezoresistive microphones** are one of the oldest types to be produced. They are assembled with a container filled with grains of pure carbon that is subjected to an electrical current. These microphones need to be powered by a battery, and the electrical signal is generated through the differing resistance that results from the compression and decompression of the grains of carbon;

- **Piezoelectric microphones** are assembled with a piezoelectric crystal, which generates electricity through pressure, allowing to detect SPL and translate them to electric signals;

- **Electromagnetic microphones** are assembled with a fixed wire coil and a conductive magnetic armor connected to the diaphragm. As SPL changes,

the diaphragm moves the armor and generates an electrical current on the coil due to the moving magnetic field;

- **Electrodynamic microphones** are assembled in a reverse fashion in comparison with the electomagnetic variety, having the magnet fixed and the coil connected to the diaphragm. The principal of operation is the same, as the coil moves due to changes in SPL, it generates a current due to the moving magnetic field;
- **Electrostatic microphones** are assembled by using a conductive diaphragm which is placed close to another electrically charged conductor, forming a capacitor sensitive to SPL. [12]

In order to know what to expect from a conversion of SPL to voltage on a microphone, the sensitivity must be analyzed. Proper conversions must be made in order to integrate the microphone in an amplifying circuit, as the maximum expected SPL input for the microphone must be known and the microphone chosen accordingly. Microphone sensitivity is determined using a 1 kHz sine wave at 94 dB SPL, which is equivalent to 1 pascal (Pa) of pressure. The amplitude of the output signal under these conditions is a measure of its sensitivity, represented in logarithmic units as dBV or dBV/Pa. [13]

2.7 Micro-Electro-Mechanical System

Micro-Electro-Mechanical System (MEMS) can be defined as miniature mechanical and electromechanical devices, with the main characteristic being some sort of mechanical functionality with or without moving elements. The types of devices can vary from simple structures with no moving elements to highly complex electromechanical systems with multiple moving elements controlled by integrated electronics.

The most interesting functional elements of MEMS are micro sensors and micro actuators, which convert mechanical signals into electrical ones and vice versa. There are a multitude of sensors with applications in sensing temperature, pressure, inertial forces, chemical species, magnetic fields and radiation, with better performance and lower costs than larger non MEMS devices for the same purpose. There are also a multitude of actuators including: valves for control of gas and liquid flow, optical switches and mirrors to redirect or modulate light beams, independently controlled mirror arrays, resonators for a number of different applications, pumps to develop positive fluid pressures and flaps to modulate air streams on airfoils.

MEMS are manufactured using batch fabrication techniques similar to Integrated Circuit (IC), giving them high levels of functionality, reliability, and sophistica-

tion on a small silicon chip at a relatively low cost. MEMS technology is extremely diverse and fertile, both in its expected application areas, as in how the devices are designed and manufactured.[14]

Chapter 3

State of the Art

The exponential growth of technology has forced the development of industrial methods to verify that electronic products work as best as possible. These methods have been developed alongside with the needs of the products, also being adapted to the new design and conception techniques for electronic solutions. In this chapter that evolution will be explored and the state of the art will be recorded as a future reference.

3.1 Automated Optical Inspection

Automated Optical Inspection (AOI) is a type of manufacturing test performed on assembled boards after they have gone trough the assembly process. AOI is meant to detect visible faults in the product after assembly, which can range from absent or misplaced components to poorly soldered components or reversed polarities. The AOI machine works by comparing images of a previously assembled board and automatically screening a newly assembled one to find differences in their images. Usually the process takes time to perfect as the machine may detect acceptable deviations as unacceptable, so care needs to be taken during the process to improve the inspection to more realistic standards.

Another type of AOI is Automated X-ray Inspection (AXI) which follows the same principles of operation but instead of comparing normal images of the assembly, it compares X-Ray images. This type of inspection is much more precise in detecting soldering and assembly problems, however it is less adequate for polarity or value inspection. This type of inspection is fundamental to detect soldering faults in components which have soldering points beneath them as is the case of Ball Grid Array components. Figure 3.1 shows an example of an X-Ray inspection and how much of the assembly is revealed in comparison to a normal

optical inspection, especially in the case of the Ball Grid Array component in the lower right corner.

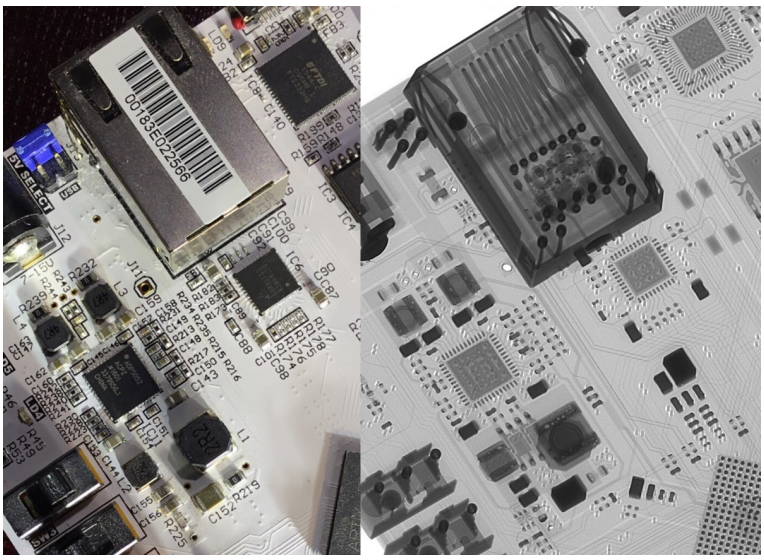


Figure 3.1: Comparison between AOI and AXI [6]

3.2 Automated Test Equipment

The Automated Test Equipment (ATE) is used in the test phase during the production of electronic circuits. The ATE allows for the test of any electronic product to be performed much quicker than it would be if it was performed manually [2]. This ability is obtained with the automation of the test process by the ATE. The time it takes to test is a fundamental aspect as the growing demand of the market makes the production time of a product, for the most part, the condition to its success [15].

3.3 In Circuit Test

In-Circuit Test (ICT) is a physical node access technique used in ATE. Very popular, it is an effective way to access test points for the test of Printed Circuit Board (PCB)s. As a node access method, ICT has proven to be very efficient, allowing to perform structural, functional or parametric testing of the circuit in part or as a whole according to what is necessary. This access technique is not only used to analyse short circuits and open connections (structural test), but also check component values (parametric test) and the operation of ICs during the test (functional test). In the last few years, however, ICT has faced growing restrictions due to new design approaches and circuit integration. Due to the increased miniaturization of IC with increasingly smaller packages and more

and more transistors, the use of ICT as become less and less viable. The figure 3.2 shows an application of ICT to perform a test on a PCB using a bed of nails interface.

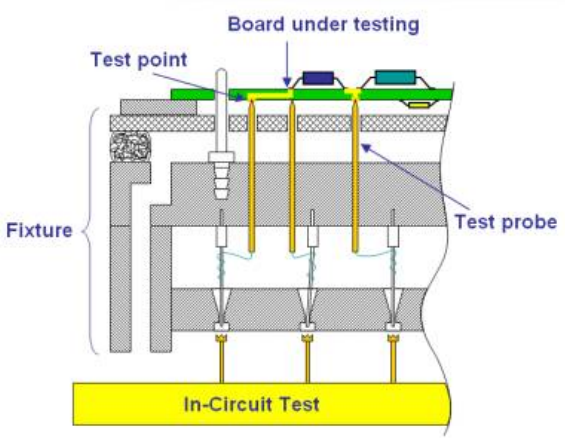


Figure 3.2: Example of a bed of nails structure for ICT

3.4 Flying Probe Test

Flying Probe test is a type of physical access test, performed using a specific ATE with at least two moving test probes. The tests performed with this technology can only be structural or parametric, but typically theses types of ATE also have optical inspection capabilities which make it possible to detect faults in components that may be more difficult to access physically. The test itself works by measuring various points around the UUT and checking their continuity or their values. The number of points that can be tested at the same time depends on the number of probes available in the ATE, however, in comparison with ICT, this type of test will always be slower, as the limitations of the probes make it impossible to have instant access to all the relevant nodes on the circuit. The applicability of this type of ATE is more restrained to simple circuits and lower volumes as the time limitations can prove to be more expensive otherwise. Figure 3.3 shows an example of a circuit being tested with flying probes.

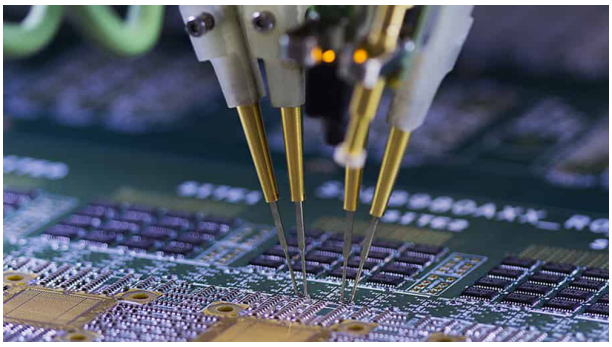


Figure 3.3: A circuit being tested using Flying Probe

3.5 Boundary Scan Test

Developed by Joint Test Action Group (JTAG), with the IEEE 1149.1 and IEEE 1149.4 standards, Boundary Scan Test (BST) offers significant advantages in comparison with more traditional test methods. The reason for the development of BST was to overcome the problems stemming from the lack of physical access to the nodes of a circuit during test which have caused the decline of the ICT method. It is a circuit test method that does not need physical access to the nodes through external hardware. It works using registers organized by cells connected in a series to one of the pins in each IC. The cells can be configured to separate the internal logic of the IC from the rest of the circuit of the PCB. It is also possible to configure whether the value loaded to the cell stimulates the internal logic of the IC or the node where the IC pin connects on the board circuit. Figure 3.4 shows the connection between two IC with the ability to perform BST.

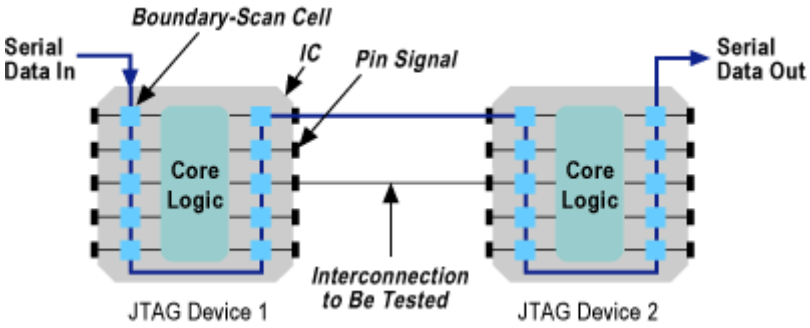


Figure 3.4: Two interconnected boundary scan cells

The test is performed by loading a serial value chain into the cell register, through Test Data In (TDI), that runs through each cell one after the other and checking if the serial values in Test Data Out (TDO) match with the expected results of the stimulation. There are also two parallel pins common to all BST ICs, one to select the mode to use the ICs on, named Test Mode Select (TMS), and one to supply the clock that is necessary for the serial chain to flow trough,

named Test ClocK (TCK). The test itself needs to be though out during board design as the circuit ICs need to be compatible with the standards in order for the test to be possible and a Test Access Port (TAP) with the four test pins needs to be designed into the board before hand. One great advantage of this method is the versatility to either be used for debugging or testing purposes, making the same test development useful for both scenarios, something that doesn't apply so easily in other methods.

3.6 ATE manufacturing companies

In order to implement an automatic test, a company has to either develop a specific in-house ATE or buy a manufactured solution from a manufacturing company. The market is full of solutions and companies with proper experience and test development history also come to fill the possible gaps that a less experienced company may have. The following companies develop test related solution and were added as a reference.

3.6.1 Advantest

Advantest is a company from Japan, founded in 1954. It has a vast number of test systems ranging from IC wafer test to fully assembled product test systems. Their marketed solutions focus on component test, as their system on chip, memory and solid state drive test dedicated solutions are unique to the market. Their solutions are more specific and are conceived as ready to use solutions for the test of products, PCBs or components.

3.6.2 Teradyne

Teradyne is a company from the United States of America, founded in 1960. It has a wide range of test systems and manufacturing automation solutions. The test systems are modular and scalable allowing easy configuration and added functionality with one of the lowest test times on the market. The test systems are configurable to use ICT and BST to perform parametric, structural or functional tests, with multiple sizes and automation options and a common test architecture with consistent hardware and software options across different models.

3.6.3 National Instruments

National Instruments is a company from the United States of America, founded in 1972. It is the most well known and successful of all the test related companies. It developed numerous test and instrumentation standards and their LabVIEW software is still widely used by companies interested in developing test

solutions. Their product range is more focused in supplying proper modular hardware and software system solutions for the assembly of ATE rather than ready to use equipment. However their systems are highly modular and have been used as the basis of multiple other systems both in manufacturing and test development companies.

3.6.4 Chroma ATE

Chroma ATE is a company from the United States of America, founded in 1984. With a wide range of ready to use test solutions, this company sells complete ATE solutions for tests covering batteries, power conversion, electric vehicles, solar power inverters, electric motors, led drivers and electrical safety. With complete solutions fully developed by the company, it is one of the best references on the market for ATE solutions.

3.6.5 Roos Technologies

Roos Technologies is a company from the United States of America, founded in 1989. It specializes in modular and scalable ATE systems designed to be easily configured and capable of testing high frequency signals along with functional, parametric and structural tests that can be as complex as the modules that the ATE has installed while still fully compatible with ICT technology.

3.6.6 Controlar

Controlar is a company from Portugal, founded in 1995. It provides industrial testing equipment as well as full test development with complete implementation of the hardware and software. The solutions are based on cabinets and are developed to perform either ICT functional, parametric or structural test, stress and vibration tests and also optical and visual tests, having a very wide product range for different types of test.

3.6.7 Averna

Averna is a company from Canada, founded in 1999. It has multiple test solutions more oriented toward signal and radio frequency testing, also providing modular test systems compatible with LabVIEW. The solutions offered by this company are oriented toward defect and fault detection and are marketed as more debug oriented than production test oriented, however this solutions are still as easily implemented in a manufacturing environment and not only in research and development.

3.6.8 STAr Technologies

STAr Technologies is a company from Taiwan, founded in 2000. It provides a wide range of test system solutions from functional and parametric tests to reliability and probe tests with the capacity to also perform structural tests to both PCBs and IC wafers.

3.6.9 TEST-OK

TEST-OK is a company from the Netherlands, founded in 2008. It provides simple and low cost ATE solutions that allow for their clients to develop personalized functional ICT tests. The company provides standard material and software in order to give the client enough tools to develop a complete test using the least amount of resources possible.

Chapter 4

Solution Proposal

This chapter will detail the requirements of the project, the solution proposal and also explore the ATE which will be used in the test as well as the product and its features. This serves as an initial approach to the main purpose of this thesis, being a necessary chapter to understand the implementation of the solution.

4.1 Requirements and the Proposed Solution

The company wanted to develop a quick and thorough test using the technology available in its factory. The test needed to use the ICT ATE from TEST-OK and be capable of testing all of the hardware blocks from the XTEC Red, Green and Blue (RGB) Siren, including the sound of the buzzer and the colors of the LEDs. As such a PCB would need to be designed in order to include the necessary hardware to interface the UUT to the ATE and a software would need to be written to interface the ATE to the operator of the test in a way that the test would be fast and clear for the common user to know what is being tested and where the product fault is. As such, the following list of requirements was delivered for the proper conclusion of the project:

- Analyse the hardware of the circuit to test;
- Separate the hardware into functional blocks with all necessary outputs and inputs;
- Delineate a test strategy that includes all the functional blocks of the circuit;
- Develop hardware to test each functional block;

- Develop a test module with all the hardware necessary to interface the UUT to the TEST-OK ATE and the proper sensors to test non electric values;
- Develop a software script that can be run using the TEST TRACK software from TEST-OK;
- Validate the solution and ensure it tests all of the product hardware blocks;
- Develop a validation strategy to ensure the test system is functional after a certain period of time has passed.

Figure 4.1 shows a diagram of the evolution of the project requirements as the project advances.

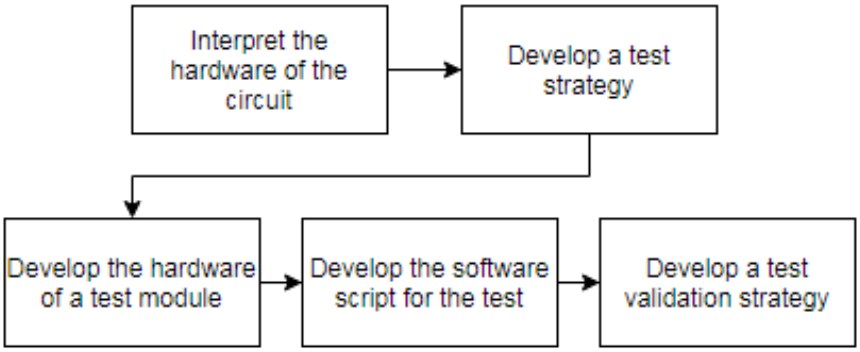


Figure 4.1: Project requirements diagram

4.2 ICT ATE from TEST-OK

The test will be developed with an ATE from TEST-OK (model TEST-OK 4000) and the respective input and output expansion board TCC 1800-UE. To develop a test using this ATE it is necessary to develop an ICT test module that serves as an interface between the ATE and the UUT as well as a test script that performs all the steps of the test. The script is composed by sequential commands that execute test steps and evaluate the success of each step in order to confirm that the UUT carries out the product requirements. Figure 4.2 shows the TEST-OK ATE used.

Figure 4.2: The TEST-OK 4000 ATE

4.2.1 ATE hardware - TCC1800-UE module interfaces

For each UUT to test using the system there is a single ICT hardware module with bed of nails to perform the tests. This interface is called a test module and is necessary to perform any produt test using this ATE. For that purpose the ATE has 4 connectors available with different functionalities on different pins that may or may not be used by the designer on the UUT test module. The figure 4.3 shows all of the hardware blocks that can be used in the test modules as well as the support for hardware expansion. The grey blocks on the figure are logical blocks seen by the software and the remaining blocks are physical blocks that are present on the connector that interfaces with the test module.

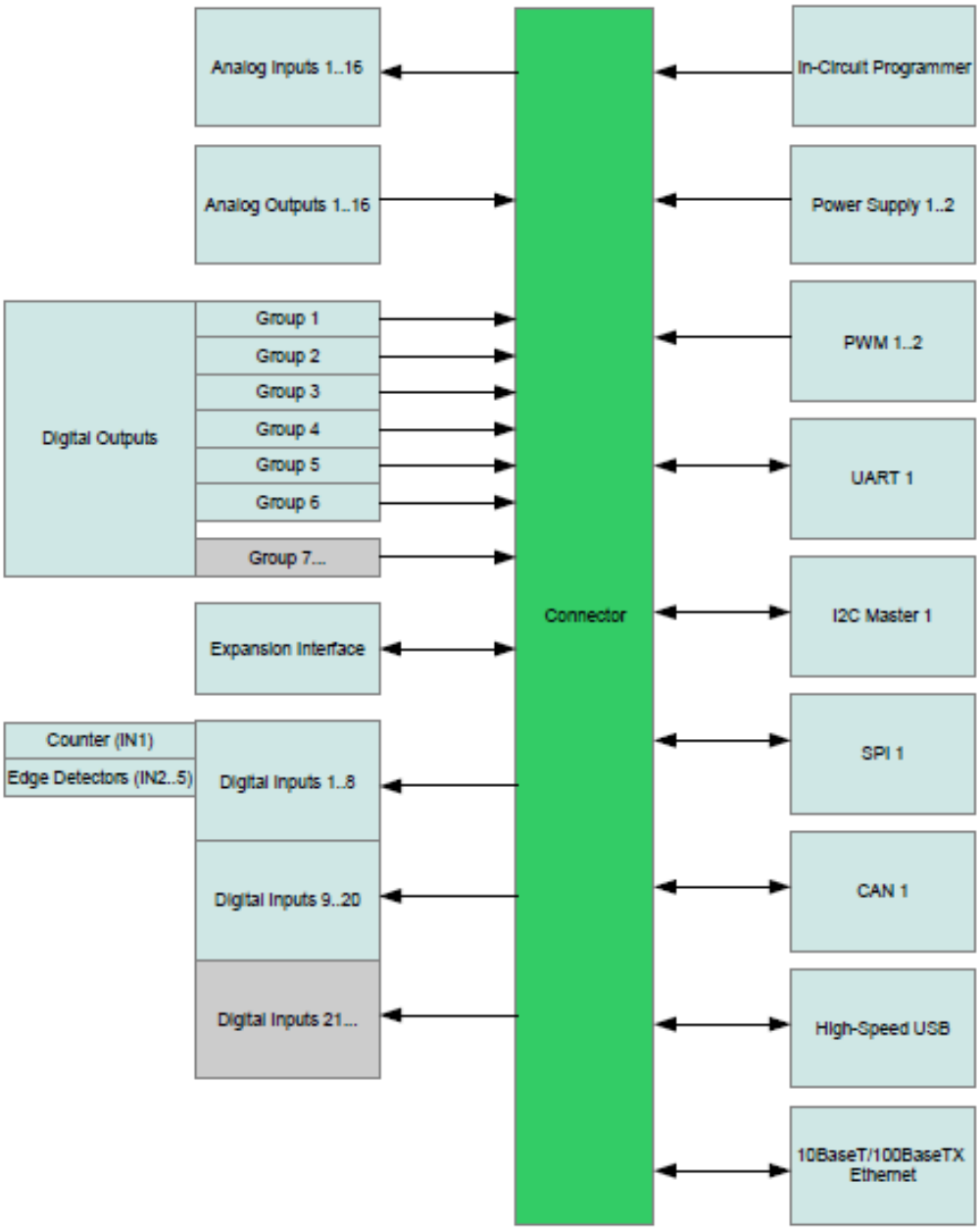


Figure 4.3: Hardware block diagram from the TCC1800-UE expansion board [7]

The ATE is composed by several hardware blocks with the following characteristics:

- **16 analog inputs** with 12 bit ADC and a voltage range between 0 V and 24 V with a precision of $\pm$ 6 mV and an absolute maximum value of 44 V on each input. The sampling rate is 25 Hz and the last 8 inputs are measured 10 ms after the first.

- **16 analog outputs** with 12 bit Digital to Analog Converter (DAC) and a voltage range between 0 V and 24 V with a precision of $\pm$ 6 mV except for values close to 0 V that have a precision of $\pm$ 30 mV. The outputs are protected against input voltages under 0 V and over 26 V, being the maximum current as a source of 40 mA and as a drain of 20 mA. Due to the current limitations on these outputs it isn't recommended to use them to power loads directly.

- **20 digital inputs** with input impedance of 1 MΩ, nominal input voltage of 24 V and maximum input voltage of 31 V. The value after which the logical high is detected is 1.5 V and the value below which the logical low is detected is 0.75 V. The inputs 1 to 5 are multi functional, input 1 is also a pulse and frequency counter and inputs 2 to 5 are also edge detectors and can be used to measure pulse width. Inputs 1 to 8 are internally connected to the outputs of Group 6 making them bidirectional pins. Input 9 is also used in the SPI implementation as a MISO line.

- **52 digital outputs organized in 6 groups**, as such that in groups 1 to 4 the output state corresponds to the configured logic value and in the groups 5 and 6 the output are open drain, that is, configuring an high logic value, the output connects to GND and with a low logic value the output connects to high impedance. For the groups 1 to 3 each output dissipates a maximum of 100 mW, having a global 500 mW limit for each group. It is recommended to not force the outputs to zero in these groups and rather use an auxiliary circuit in order not to run the risk of causing damage due to improper power dissipation. Group 4 is composed by outputs with high-side switches limited to voltages between 5.5 V and 24 V and a maximum output current of 500 mA. Groups 5 and 6 dissipate a maximum of 100 mA for each output. Groups 1, 2, 3 and 6 are composed by 8 outputs each, group 4 has 2 outputs and group 5 has 16 outputs. The outputs in group 6 are internally connected to the inputs 1 to 8 making them bidirectional pins. Pins 1 and 2 of group 1 are used in the SPI implementation as SCLK and MOSI respectively.

- **2 PWM outputs** with independent configuration for frequency and duty cycle and 5 V voltage when in the logic high level. The frequency is configured with one of 1020 different values between 612 Hz and 10 MHz. The duty cycle can be configured between 0 % and 100 % with a resolution dependent on the used frequency value.

- **Serial interface** developed to work with 5 V signals, with a 20 V over voltage protection on the input pin and short circuit protection on the output pin. The output current is limited and it is recommended to use a buffer

or driver in the test module in order not to risk the integrity of the ATE. Baudrate and other parameters are configurable through software.

It has two buffers for transmission and reception. The reception buffer has three working modes: *polling*, where buffer information is requested by the host; *controlled interrupt*, where information is received after a number of bytes are added to the buffer and *timeout interrupt*, where data is received after a timeout is reached with no communication as long as the buffer has more than one byte.

The serial interface also has a Start Transmission (STX) mode where transmission packets are created and delimited with characters at the start and at the end in order to have better control of what is sent in each communication.

- ***Master* I2C interface** with an internal pull up to 5 V on the SCL pin and a bidirectional SDA pin that may need an external pull up in order to work. The communication frequency is limited to 100 kHz.

- **SPI interface** reuses input and output pins and has a logic high voltage equal to the value configured for the Group 1 digital outputs. The communication frequency is limited to three frequencies: 78 kHz, 156.25 kHz and 312.50 kHz.

- **Controller Area Network (CAN) interface** implements the 2 wire standard or the 1 wire method even though there is no guarantee that it will be compatible with all the systems.

- **Universal circuit programmer** is a firmware programmer that is included in the ATE hardware with the purpose of easily programming compatible Microchip or Atmel micro controllers on the UUT.

- **Two independent and configurable power sources** with an output voltage range between 1.24 V and 24 V in increments of 6 mV. Limited to a maximum of 3 A or 45 W of power on the output, with a minimum programmable voltage of 1.24 V. The output current can be requested or configured to have a maximum limit. In case the configured or the absolute maximum output current is exceeded the power source is turned off and is kept off until there is a system power cycle. The measurement resolution for the current is $\pm$ 0.375 mA, considering that the current values only stabilize 480 ms after the circuit is powered on.

- **Two independent and fixed power sources**, one of 28 V with current limited to 0.75 A via fuse (powers all the hardware on the TCC1800-UE board) and another of 5 V limited to 0.25 A via fuse.

- **UUT test module identifier** serves as a way to configure wich test application to run in order to test a certain UUT by using an identifier present on each test module. The O identifier is a 7 or 14 bit shift register that identifies the test module with one of 126 identifier for 7 bit or one of 4096 for 14 bit, as the 0 identifier is reserved. This identifier is used by the test software to detect which test program should run and to identify that the test can be started as well as the presence of a **uet!** (**uet!**) on the module.
- **Two independent Universal Serial Bus (USB) interfaces** available to connect any necessary hardware for the test. It is not possible to connect hubs to these ports.
- **Ethernet interface** present on the expansion module and on the UUT test module as well.
- **Expansion possibilities** for inputs and output using specific hardware for the test modules. It is possible to add up to 256 additional inputs or outputs using more hardware on the module. Is is also possible to add two additional power sources and a cover sensor to detect if the ATE cover is open or closed. [7]

4.2.2 ATE software - TEST-TRACK and TDL

The ATE uses a software named TEST-TRACK which beyond being an environment to develop and validate the test applications, is also the main environment for the user to run the test on the UUT. The software also has capabilities related to traceability, test and fault logs as well as repairs for the tested products. The test applications are composed by individual files that test each hardware block of the UUT as well as a configuration file, named preamble, that performs the necessary configurations before the rest of the test code is performed. The logic of separating the hardware blocks into different code files allows the user to test only one part of the UUT if necessary.

The programming language to use has the designation of Test Description Language (TDL) and was developed by TEST-OK as a specific tool for the development of test applications on the ATE. [16]

4.3 Study Case - XTEC RGB Siren

The XTEC RGB Siren is a product developed by Tecnimaster which will be the only test to be implemented in the TEST-OK ATE during this thesis. The siren is composed by four different hardware blocks which interact, directly or indirectly, with the block that contains the Central Processing Unit (CPU). Each block will be explained in the following subsection.

The siren hardware is split into two PCB, one main board with most of the hardware and a second board with only eight RGB LEDs, that is connected to the main using a strip cable. The purpose of the product is to serve as a domestic siren to be implemented either to signal home intrusions or to signal home fires. The siren has additional installation options with regards to the colours of the LED strips that blink during normal functioning, hence the name referring to the different colours, and subsequent combinations, that can be configured for the strips using the colours red, green and blue. Figure 4.4 shows the PCBs fully assembled.

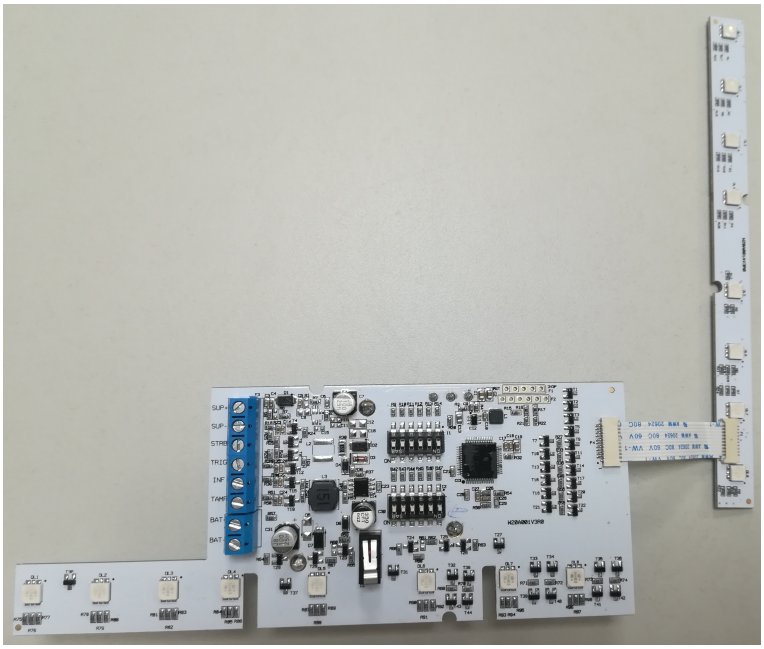


Figure 4.4: The XTEC RGB Siren assembled PCBs

4.3.1 Hardware blocks

The siren is composed of five different hardware blocks that are interconnected and can interact with each other. The blocks are named as such: CPU Block, Power Block, Buzzer Block, Control Signal Block and LED Block. Each block will have to have a different test, but since some of the blocks are not isolated from others the test made to one block can also test another indirectly. The block that most benefits from indirect testing is the CPU as all other blocks interact with it either directly or indirectly. In this subsection the functioning of each block and what hardware it contains will be specified and a brief description will be made for each one. Figure 4.5 shows all of the blocks interconnected.

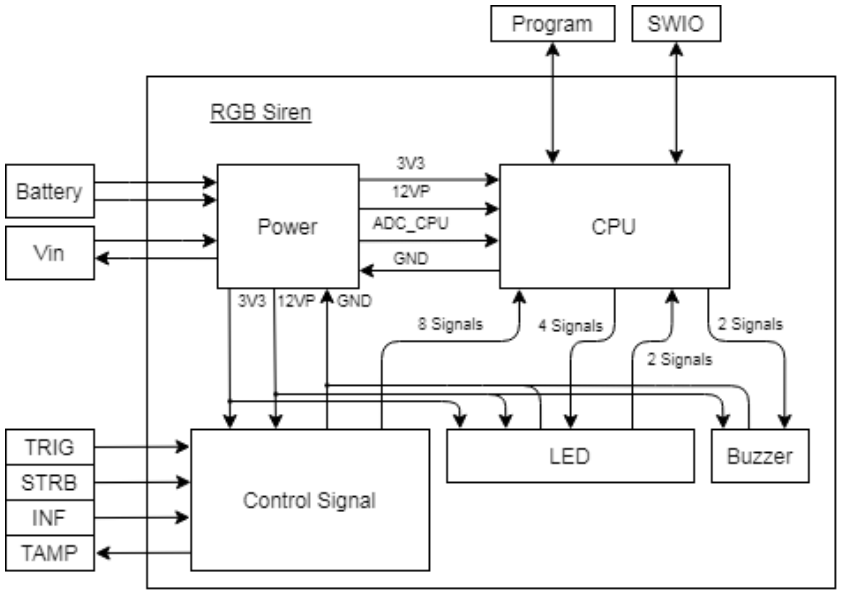


Figure 4.5: RGB siren block diagram

4.3.1.1 CPU Block

The CPU block has ten primary inputs and two primary outputs being one of them also an input. The block is composed by two connectors used in programming and firmware debug as well as the CPU of the board and every other block connects to this one directly. The firmware is loaded to the board through the SWIO interface during production. Figure 4.6 shows the CPU block diagram.

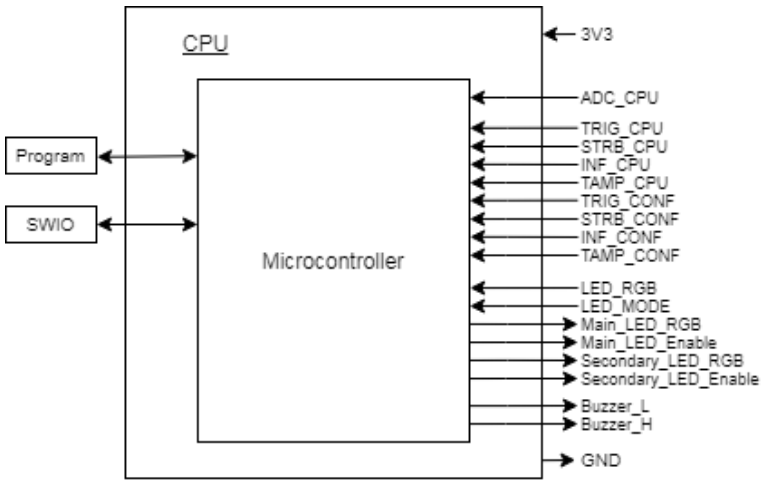


Figure 4.6: CPU block diagram

4.3.1.2 Power Block

The power block has four primary inputs, two of them also serving as primary outputs. The implemented hardware serves to power the board and also

charge a battery that is external to the PCB. The circuit has no protection against over-voltage or over-current. There are four assembly options for the hardware, each depending on the main input voltage and the battery voltage. Table 4.1 shows each possible assembly option for the board.

Table 4.1: Hardware assembly options for the RGB Siren

Option	Input Voltage	Battery Voltage
1	12 V	7.2 V
2	12 V	12 V
3	24 V	7.2 V
4	24 V	12 V

The power block is comprised of three DC-DC converters, a current directing mesh using diodes and a voltage divider that connects to the ADC on the CPU. Of the three converters two of them are buck converters and one of them is a boost converter. The power block has two output voltages that are internally used to power different hardware blocks. The first output voltage is 12 V and is used in every block except CPU as the main power source. The second output is 3.3 V and is used to power the CPU block and all the signals that come to it from other blocks.

The current directing mesh is used as a way to charge the external battery of the product while protecting the battery against over voltage and the external power supply against reverse voltage. The mesh is assembled differently depending on the board option. Figure 4.7 shows that part of the circuit along with the indication of what components are assembled for each assembly option.

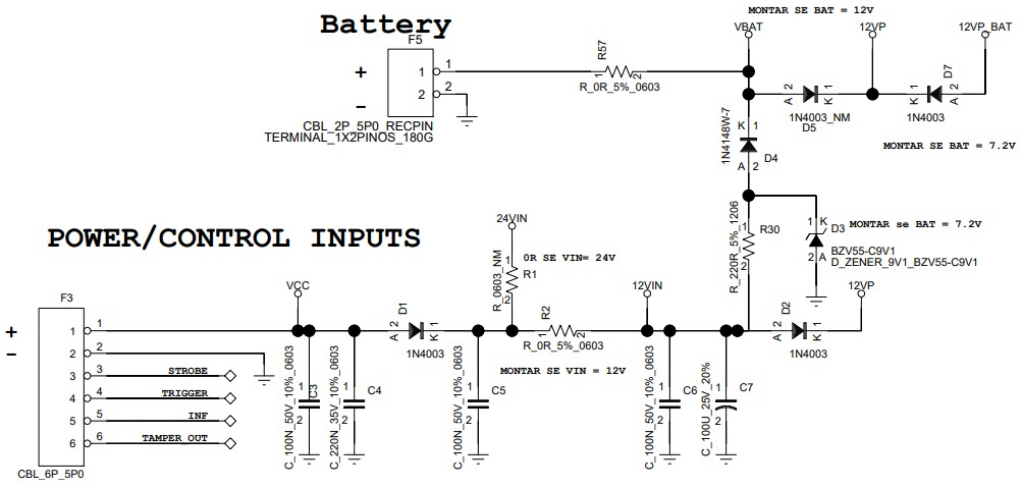


Figure 4.7: Current directing mesh part of the circuit

The first buck converter turns the same 12 V of the first output of the block into the 3.3 V of the second output. The second buck converter is used in the assembly options 3 and 4 to convert the 24 V input voltage into a 12 V input voltage to be used in the circuit. The boost converter is used in the assembly options 1 and 3 to convert the 7.2 V battery voltage to a 12 V input voltage to be used in the circuit. Figure 4.7 shows all the converters and the voltage divider used for the ADC along with the indication of what components are assembled for each assembly option.

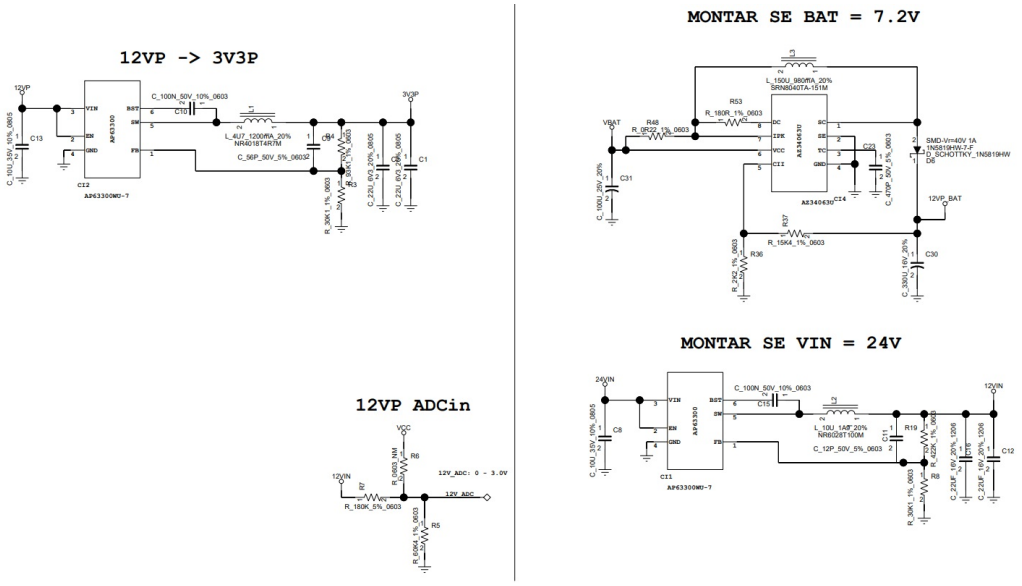


Figure 4.8: DC-DC converter part of the circuit

After analyzing the circuit, the minimum input voltage is limited by the buck DC-DC converters. The minimum operating voltage is 10 V for either of them, so this value is valid for both input voltages. For battery voltage 7.2 V, the minimum value is limited by the boost converter to 4 V and for the 12 V input voltage it is limited to the same 10 V. The maximum operating voltages are limited to 35 V by the converters used in the 24 V and 7.2 V inputs. For the 12 V input, the limitation comes from the micro controller itself, as the ADC_CPU signal cannot be higher than 4 V without damaging the ADC input on the micro controller. As such, the calculations show that the maximum value for the 12 V inputs is 16 V. Table 4.2 shows the minimum voltages and maximum current that the power block can handle and figure 4.9 shows the power block diagram.

Table 4.2: Power Control hardware block pins

Option	$V_{SUP_{min}}$	$V_{BAT_{min}}$	I_{max}
1	10 V	4 V	0.45 A
2	10 V	10 V	0.45 A
3	10 V	4 V	0.45 A
4	10 V	10 V	0.45 A

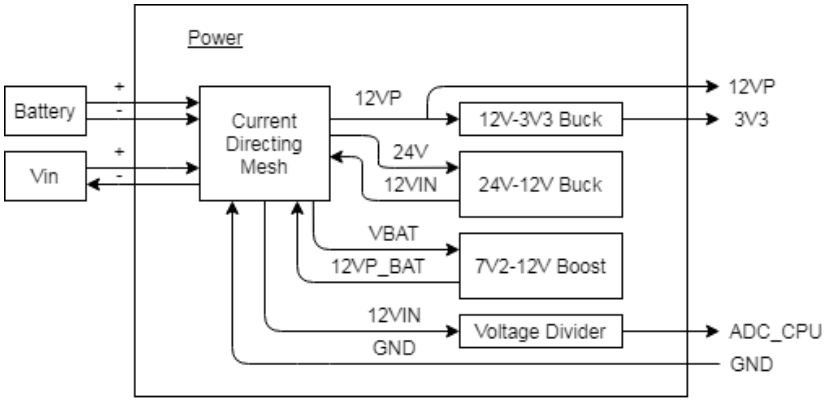


Figure 4.9: Power block diagram

4.3.1.3 Buzzer Block

The Buzzer block has no primary inputs or outputs being solely controlled by the CPU. This block is the simplest, being only composed by an H-Bridge and the Buzzer. The Buzzer is activated with a square wave with 12 V amplitude, 50 % duty-cycle and a frequency of about 2.5 kHz, such that it outputs a sound wave with 110 dB SPL. Figure 4.10 shows the buzzer block diagram.

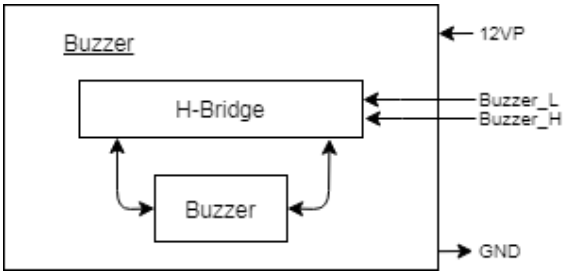


Figure 4.10: Buzzer block diagram

4.3.1.4 Control Signal Block

The control signal block has three primary inputs and one primary output. This block serves as the interface between the inputs and the CPU and it also includes the configuration DIP switches that change the accepted input values for the control signals. The control signals are defined as Strobe (STRB),

Trigger (TRIG), INternal Fault (INF) and Tamper (TAMP). STRB, TRIG and INF are inputs and can be configured as either active high or active low in the DIP switch. TAMP is an output and can be configured to turn on the buzzer or to not have any effect on the board.

The control signal inputs are composed of a simple low side switch for each signal that activate the input in the CPU and a branch before the switch that can set the node on the base to be either active high or active low depending on the state of the CONF bits. Figure 4.11 shows a simplified circuit of the logic used for each input. If Signal_CONF is high, Signal works as active high and Signal_CPU is read as active low on the CPU, if Signal_CONF is low, Signal works as active low and Signal_CPU is read as active high on the CPU.

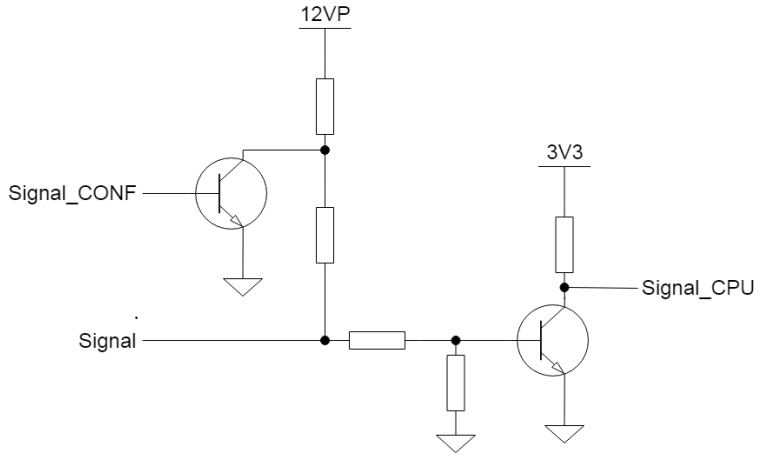


Figure 4.11: High side switch control for signal inputs

The state of TAMP is dependent on two micro switches on either side of the PCB, if both of them are closed, TAMP is at 0 V, if one of them is open, it is at 12 V. This signal also has a high side switch circuit to interface the signal with the CPU. Figure 4.12 shows the control signal block diagram

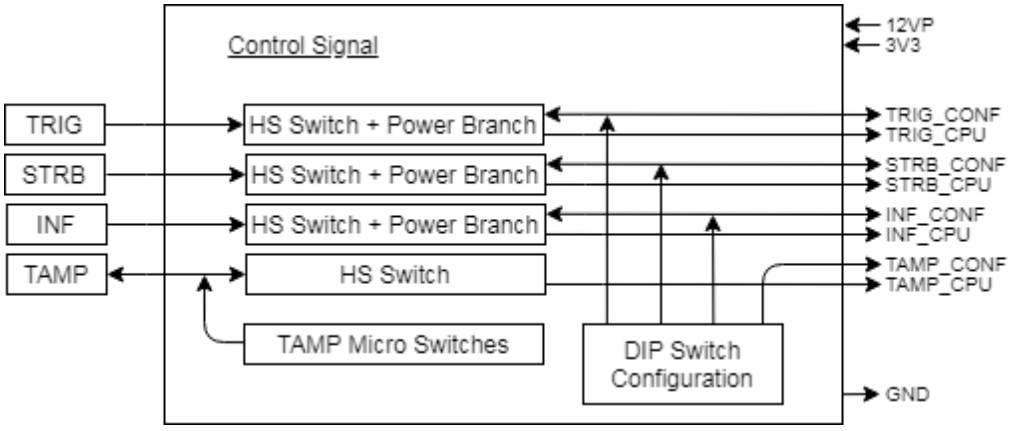


Figure 4.12: Control signal block diagram

4.3.1.5 RGB LED Block

The RGB LED block has no primary inputs or outputs being solely controlled by the CPU and the behaviour and colour configured by the DIP switches that also connect to the CPU. The LEDs are distributed between the main board and a secondary board connected to the main using a contact strip. Each board has 8 LEDs and each group of 8 can have a different colour configuration set in the DIP switches. The DIP switches also change the behaviour of the LEDs in the normal functioning of the product. Two switches are reserved for mode and six are reserved for colour, one for each RGB color component in each of the two groups.

The control of the LEDs is done by the CPU using the information configured in the DIP switches. Table 4.3 shows the way signals are organized to control the LEDs on the board.

Table 4.3: LED control signals

Control	LED	Colour
EN1	1	RGB1
	2	RGB2
EN2	3	RGB1
	4	RGB2
EN3	5	RGB1
	6	RGB2
EN4	7	RGB1
	8	RGB2

The hardware is designed so the colour is controlled independently for even and odd LEDs and the enable of the LEDs is common between consecutive pairs of LEDs. It is designed in a way that the LED only lights up if a enable and a colour are set, as such, the control can be done for each LED individually. The modes of operation of the LEDs are configured using two bits. Table 4.4 shows every operation mode that can be configured and figure 4.13 shows the RGB LED block diagram.

Table 4.4: LED modes

Bit0	Bit1	Mode Name	Behaviour
0	0	Blink	Blinks all LEDs of the main board and all LEDs of the secondary board in alternation
0	1	Fade	Fades the light of every LED out and in
1	0	Strobe Fast	Lights up every LED sequentially and fast
1	1	Strobe Slow	Lights up every LED sequentially and slowly

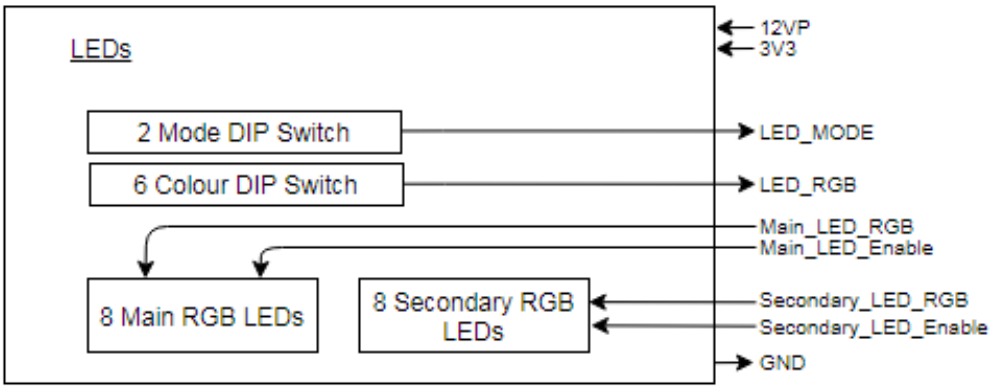


Figure 4.13: LED block diagram

4.3.2 Final product

The final product where the boards will be assembled is the XTEC RGB Siren alarm box which also includes the battery used. The boards are fixed and the TAMP signal is configured so the alarm rings in case either the box is taken from the wall or is opened. This functionality works by using the two buttons present on the bottom and top of the board, which are pressed when the product is fixed to the wall and the box is closed but are released otherwise. The INF, STRB and TRIG signals are connected to the alarm central inside the building and V_{IN} also comes from the same alarm central to the RGB Siren. Once powered on the siren stays with the LEDs turned on, performing one of the behavioral patterns detailed in table 4.4 with the colours that were set for each of the LED strips. In case no color is set for the LEDs on any of the boards, the behaviour pattern is ignored and the LEDs on the board light up in an alternating strobe of red and blue. This will happen to the boards independently, that is, if the colors of the secondary board are set, that board will follow the set behaviour and color, but the main board will keep the red and blue strobe. Table 4.5 shows the behaviour of the board when it is stimulated by any of the different external signals.

Table 4.5: Signal behaviour

Signal	Behaviour
STRB	When active it turns the LEDs on with the configured LED color, if no color is configured it flashes all LEDs white
TRIG	When active it turns on the buzzer
INF	When active it turns all the LEDs on for 15 seconds with the color red and if deactivated after, turns the LEDs on for the same time with the color green
TAMP	When active it activates the buzzer and the LEDs to the set color, as if one of the buttons was released

4.3.3 Test blocks

Considering the previous information in this section, a test block structure can be done in order to sketch out how a complete test can be made and what should be tested. This structure will be completely fulfilled during the development of this test, however, the blocks themselves are simplified and could be much more detailed at the cost of time and practical implementation of the test. This structure is a simple quantification of operations that need to be realized for a bare minimum test to be performed and will be used to compare the new test with the one that was previously performed in the company. Table 4.6 shows the test block structure.

Table 4.6: XTEC RGB test blocks

Main Block	Test
CPU	- Program the firmware without powering the board using Vin or Battery;
Power	- Power the board using Vin and measure +12 V; - Power the board using Vin and measure +3.3 V; - Power the board using Vin and see voltage in the Battery input; - Power the board using Battery and measure +12 V; - Power the board using Battery and measure +3.3 V; - Power the board using Battery and measure 0 V in the Vin input;
Buzzer	- Activate the buzzer and see if sound comes out; - Activate the buzzer and measure the frequency of the signal;
Control Signal	- Activate the Strobe signal and see if the LEDs behave as expected; - Activate the Trigger signal and see if the buzzer behaves as expected; - Activate the INF signal and see if the LEDs behave as expected; - Activate the Tamper signal and see if the buzzer behaves as expected; - Click and release the Tamper buttons and see if the buzzer behaves as expected;
RGB LED	- Set the color red on both boards and see if all the LEDs are red; - Set the color green on both boards and see if all the LEDs are green; - Set the color blue on both boards and see if

Main Block	Test
	all the LEDs are blue; - Set the blink configuration and see if LEDs behave accordingly; - Set the fade configuration and see if LEDs behave accordingly; - Set the strobe fast configuration and see if LEDs behave accordingly; - Set the strobe slow configuration and see if LEDs behave accordingly.

Chapter 5

Solution Implementation

This chapter relates how the solution was implemented both conceptually as well as practically. The chapter shines light on every development performed during the project, both hardware or software as well as all the steps performed to get the final solution implemented before validation.

5.1 RGB Siren Test

In order to implement a production test for the product, it's necessary to think over what kind of test is going to be performed and how thorough that test will be. The selection of the test is dependent on the volume of production and how production is performed. The production of this board is done in a nearly completely automatic component assembly, also having automatic optical inspection for every Surface Mount Device (SMD) component assembled. The only manual assembly is for the few Through Hole Technology (THT) components the board has. The THT components assembled on the board are the buzzer, the two micro switches and the edge screw connectors that interface to the alarm installation wires. Given that information, it's mostly safe to assume that once production has kicked off, all products will have the exact same assembly and the margin for error is very low, so full structural tests are a bad option for validating this product, as they can be lengthy and the production methods don't justify the time spent. The proper test to run will be a functional test that validates the full functionality of each of the boards' hardware blocks individually in order to certify the boards correct expected behaviour.

5.1.1 Test logic for each hardware block

Each hardware block is tested independently using a test module compatible with the TEST-OK ICT ATE. The module to use in the test is designed in such a way that the PCB is set with the top layer facing down and the bottom layer facing up. The reason for this set up is due to the position of the buzzer which is soldered to the PCB bottom, as the area it takes and the complexity of the circuit connections require for the test points to be set at the top layer instead. This upside down configuration is also more convenient to test the LEDs of the board, as they can be pointed to RGB light sensors without needing to design a top interface for the test module. This subsection will detail every test performed on the board and the justification for each choice made in the development.

5.1.1.1 Test 1 - CPU Block Test

The only direct operation performed to the CPU block will be to download the firmware to the micro controller. As such no proper direct test will be performed, all the connections will be tested indirectly through the rest of the hardware tests. The firmware will be downloaded to the micro controller using a ST-LINK V2 USB programmer connected to the Serial Wire Input and Output (SWIO) connector on the board using test probes. The SWIO connector is not assembled during production, so the test points will make direct contact with the pads. This will be the first test performed on the board and it will be done with the UUT powered off, instead only the micro controller will be powered, through the SWIO connector.

5.1.1.2 Test 2 - Power Block Test

In order to test the Power Block, the board will be powered at the minimum values for each assembly option. The maximum values are not validated because the circuit has no over-voltage protection and there is a risk the test may be destructive to the assembly. After being powered, the LEDs will be turned on with the white color configured for maximum current consumption, the buzzer could also be activated, but the noise will be too much for the test to be like that. After that the 3.3 V output will be measured and the test will only be successful if the measured voltage is within $\pm$ 5 % of the nominal output value. The ADC_CPU output will also be measured and a voltage of 2.51 V $\pm$ 5 % is to be expected. Finally the current will be measured and only accepted if it is lower than the expected I_{max} value. The battery connector will also be tested by measurement when the UUT is being powered by the main input. After that the UUT will be powered by the battery input and the output values will be checked again. The power will be set, using one of the controllable power supplies from the ATE, to each of the test voltages to be used and the outputs will be measured

using 3 of the ADC inputs on the ATE. Table 4.2 previously showed the minimum values for each supply along with the maximum current expected under normal conditions and figure 5.1 shows the flowchart of the test logic used for the test.

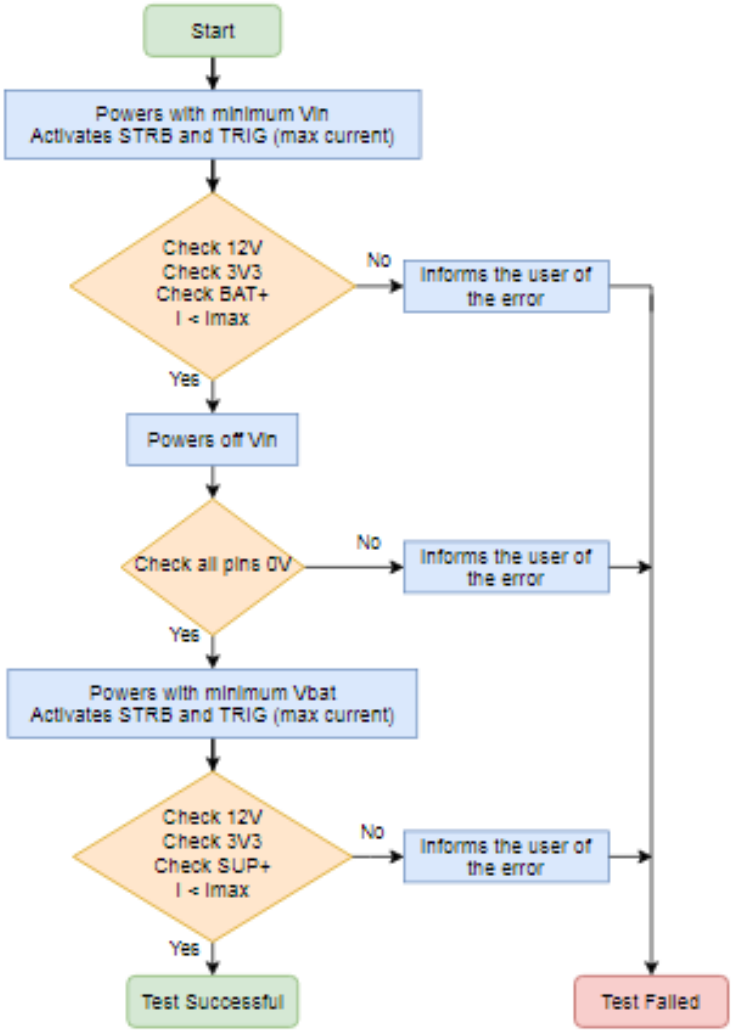


Figure 5.1: Power Block test flowchart

5.1.1.3 Test 3 - Buzzer Block Test

Much like the CPU Block test, this test is not performed directly, rather it is performed at the same time as the Power Block test as it will need to ring the buzzer in order to force the most current consumption from the power source. The test to perform will be to check the frequency of the signal on the buzzer and the intensity of the sound produced by it. The frequency should be at least 2.5 kHz and the intensity of the sound at least 110 dB SPL. To test the intensity of the sound a MEMS microphone will be used along with an amplifying

circuit that connects to one of the ADC inputs on the ATE and to check the frequency a counter input from the ATE will be used in frequency mode and read directly from the UUT pin. The value expected from the microphone will be obtained experimentally, as the positioning of the microphone is unconventional and usually SPL measurements are performed at a distance of 3 m from the source. Figure 5.2 shows the flowchart of the test logic used for the test.

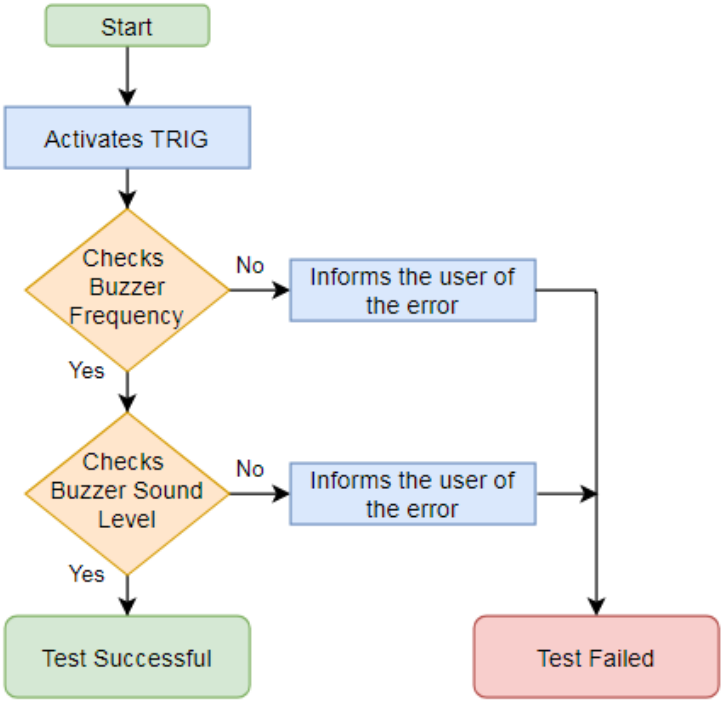


Figure 5.2: Buzzer Block test flowchart

5.1.1.4 Test 4 - LED Block Test

The LED block test will be validated using RGB light sensors that communicate with the ATE using I2C. The test will be performed by changing the DIP switch configurations and checking that every LED turns on with every colour and every behaviour that can be set. The configuration of the DIP switches will be performed automatically through test points connected directly to group 5 output pins on the ATE, setting all the different LED colours and modes. Figure 5.3 shows the flowchart of the test logic used for the test.

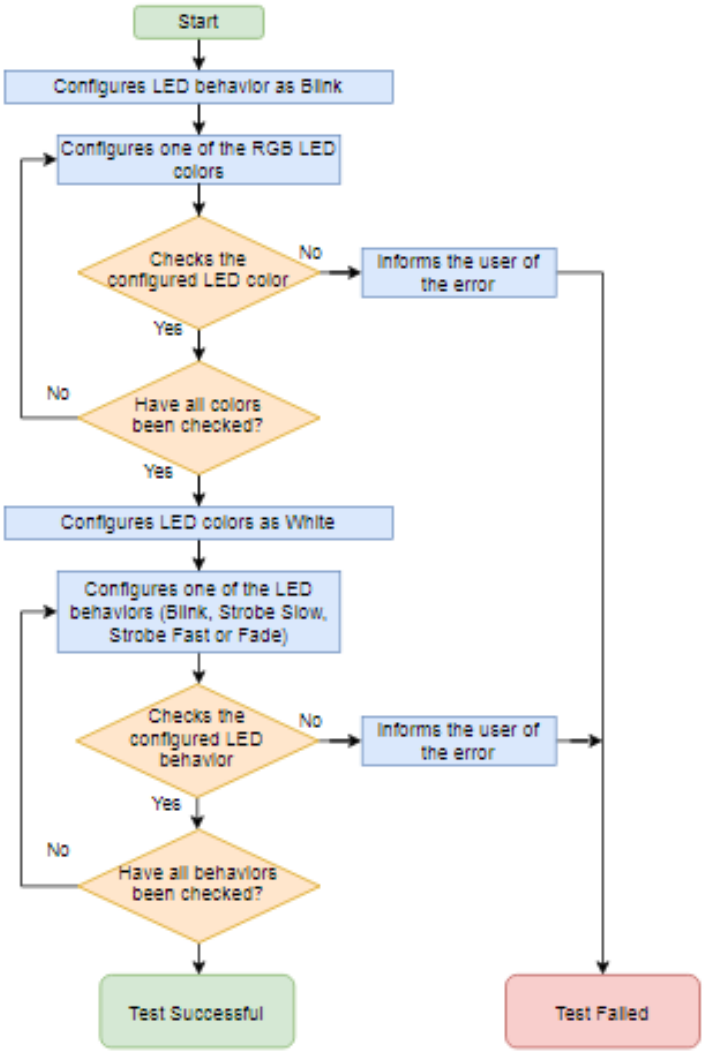


Figure 5.3: LED Block test flowchart

5.1.1.5 Test 5 - Control Signal Block Test

In order to test the Control Signal Block, the board will be powered and every signal will be stimulated or observed at a time. After each signal is stimulated the behaviour of the board will be observed for each signal configuration based on table 4.5, validating each branch of hardware along with the board firmware. The configuration of the DIP switches will be performed automatically through test points connected directly to group 5 output pins on the ATE, setting both modes for each signal. The TAMP signal will be observed and a user operation will be necessary to click on the micro switch that is on the other side of the board in order to validate that the signal changes. The signals are controlled with the open drain output pins from group 5 of the ATE connected using a pull up to 12 V in order to test in both board input configurations. The TAMP output

connects to one of the bidirectional pins from group 6 without any interfacing circuit. Figure 5.4 shows the flowchart of the test logic used for the test.

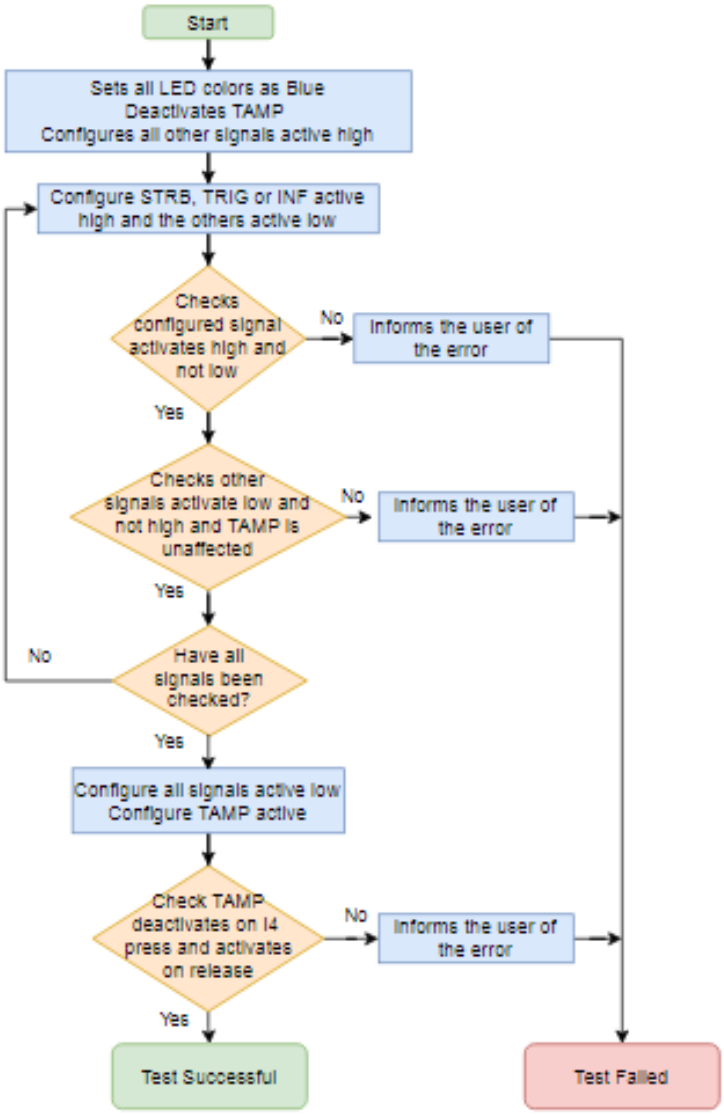


Figure 5.4: Control Signal Block test flowchart

5.1.2 Hardware blocks for each test logic

In order to perform each test, hardware needed to be developed into a TEST-OK Module with test probes and interfaces to the ATE pins with their respective functions. The test module needs to have footprints with the ATE interface connector, a identification PISO shift register for the Test Module Identification as well as a UUT Board Detect pin. The hardware was developed with the test functions in mind and the ICs were chosen depending not only on their func-

tionality but also their availability in the stock of the company. This sub section will detail each hardware block used in the test module.

5.1.2.1 Power Control hardware block (a.)

The Power Control hardware block includes the UUT power control relays as well as the modules' power control relay along with a 12 V to 3.3 V Buck DC-DC converter used to power some of the other hardware blocks. The control relays AQY212EHA were chosen for their availability and their load voltage (maximum of 60 V) and current (maximum of 0.55 A). The Buck DC-DC converter circuit was based upon the AP63300WU which was also chosen for its availability within the company's stock. The circuit follows the directions detailed in the datasheet of the component and uses the proper passive components to reach a 3.3 V output. The BOARD_DETECT pin is also connected to BAT- on the UUT through this circuit. Table 5.1 lists all of the hardware pins of the block and figure 5.5 shows the hardware block circuit.

Table 5.1: Power Control hardware block pins

Signal	Direction	Type	Interface
+12	Out	Power	Module
+3V3	Out	Power	Module
SUPPLY_1	In	Power	ATE
SUPPLY_2	In	Power	ATE
DIG_OUT_1.3	In	Signal	ATE
DIG_OUT_1.4	In	Signal	ATE
DIG_OUT_1.5	In	Signal	ATE
BOARD_DETECT	Out	Signal	ATE
ANA_IN_3	Out	Signal	ATE
ANA_IN_5	Out	Signal	ATE
SUP+	Out	Power	UUT
BAT+	Out	Power	UUT
SUP-	In	Power	UUT
BAT-	In	Power	UUT

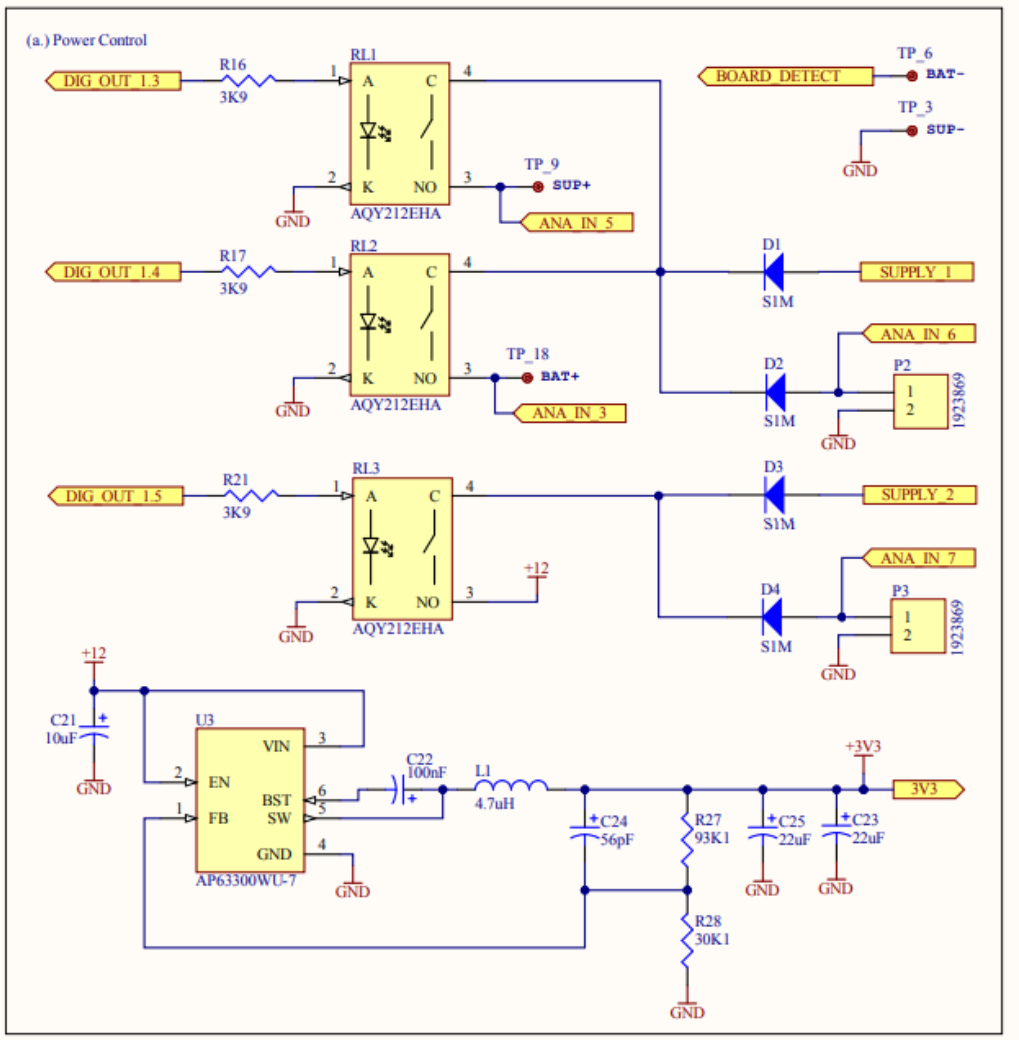


Figure 5.5: Power Control hardware block schematic

5.1.2.2 Signal Control hardware block (b.)

The Signal Control hardware block uses simple direct connections from the ATE to the UUT, three of these with *pull-up* resistors. The three *pull-up* resistor connections are done for the three signal pins of the UUT to test both cases of being active high or active low with the open drain outputs of the ATE. Table 5.2 lists all of the hardware pins of the block and figure 5.6 shows the hardware block circuit.

Table 5.2: Signal Control hardware block pins

Signal	Direction	Type	Interface
ANA_IN_2	Out	Signal	ATE
DIG_OUT_5.1	In	Signal	ATE
DIG_OUT_5.2	In	Signal	ATE

Signal	Direction	Type	Interface
DIG_OUT_5.3	In	Signal	ATE
DIG_OUT_5.4	In	Signal	ATE
DIG_OUT_5.5	In	Signal	ATE
DIG_OUT_5.6	In	Signal	ATE
DIG_OUT_5.7	In	Signal	ATE
DIG_OUT_5.8	In	Signal	ATE
DIG_OUT_5.9	In	Signal	ATE
DIG_OUT_5.10	In	Signal	ATE
DIG_OUT_5.11	In	Signal	ATE
DIG_OUT_5.12	In	Signal	ATE
DIG_OUT_5.13	In	Signal	ATE
DIG_OUT_5.14	In	Signal	ATE
DIG_OUT_5.15	In	Signal	ATE
OUT_6.2/DIG_IN_2	In/Out	Signal	ATE
CONF_STROBE	Out	Signal	ATE
CONF_TRIGGER	Out	Signal	ATE
CONF_INF	Out	Signal	UUT
CONF_TAMPER	Out	Signal	UUT
CONF_LED_MODE0	Out	Signal	UUT
CONF_LED_MODE1	Out	Signal	UUT
CONF_LED1_COLOR0	Out	Signal	UUT
CONF_LED1_COLOR1	Out	Signal	UUT
CONF_LED1_COLOR2	Out	Signal	UUT
CONF_LED2_COLOR0	Out	Signal	UUT
CONF_LED2_COLOR1	Out	Signal	UUT
CONF_LED2_COLOR2	Out	Signal	UUT
12V_ADC	In	Power	UUT
TAMP	Out/In	Signal	UUT
STRB	Out	Signal	UUT
TRIG	Out	Signal	UUT
INF	Out	Signal	UUT

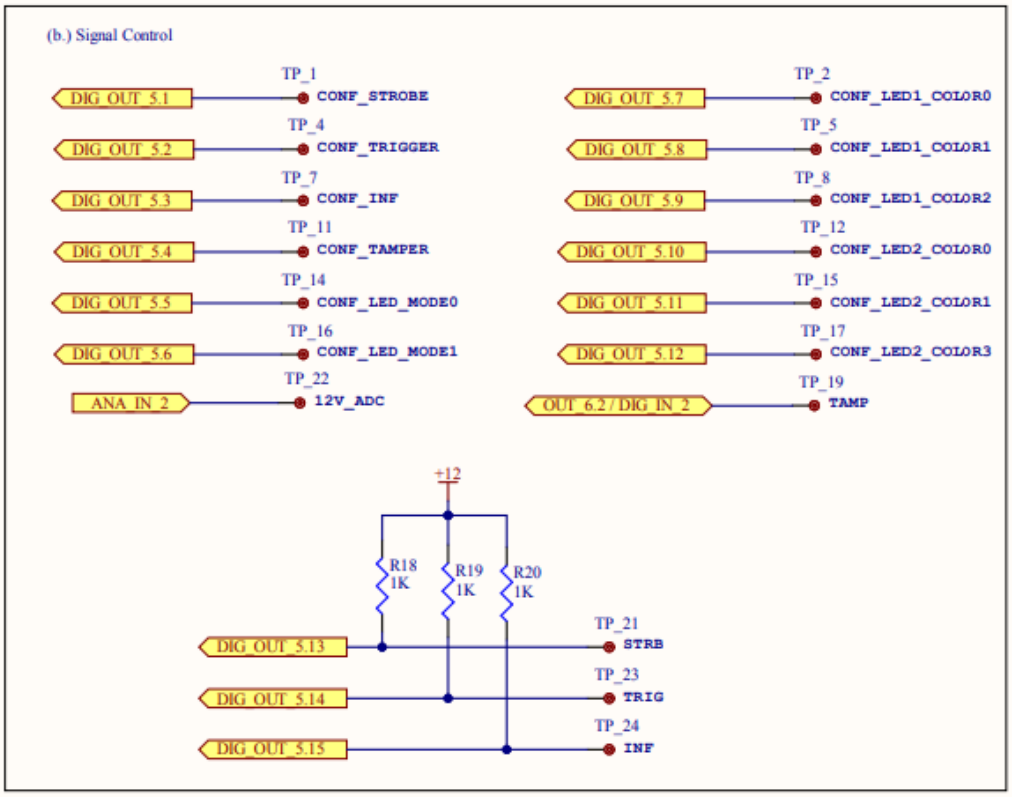


Figure 5.6: Signal Control hardware block schematic

5.1.2.3 Programmer interface hardware block (c.)

The Programmer interface hardware block serves as a way to integrate the ST-Link V2 programmer into the test module without having to design hardware that performs the same function. As such, the module has a USB connector and a 2x5 flat cable connector that are meant to fit the programmer and connect it to the UUT in order to program its firmware. The advantage of this approach is that the existing company programmers can be used and the cost of production of the module can be lowered. Figure 5.7 shows an example of a programmer to connect to this hardware block.

Figure 5.7: ST-Link V2 Programmer

The circuit also implements a AQY212EHA relay to cut the power coming from the programmer to the UUT after the firmware is programmed and the test is being performed. Most signals on the block are direct connections between the programmer and the UUT except for the analog input connection to the ATE. Table 5.3 lists all of the hardware pins of the block and figure 5.8 shows the hardware block circuit.

Table 5.3: Programmer interface block pins

Signal	Direction	Type	Interface
USB_VBUS	In	Power	ATE
USB_D+	In/Out	Signal	ATE
USB_D-	In/Out	Signal	ATE
USB_GND	Out	Power	ATE
DIG_OUT_1.6	In	Signal	ATE
ANA_IN_1	Out	Signal	ATE
3V3	In/Out	Power	UUT
ST_RESET	Out	Signal	UUT
SWCLK	Out	Signal	UUT
SWDIO	In/Out	Signal	UUT
SWD_GND	In	Power	UUT

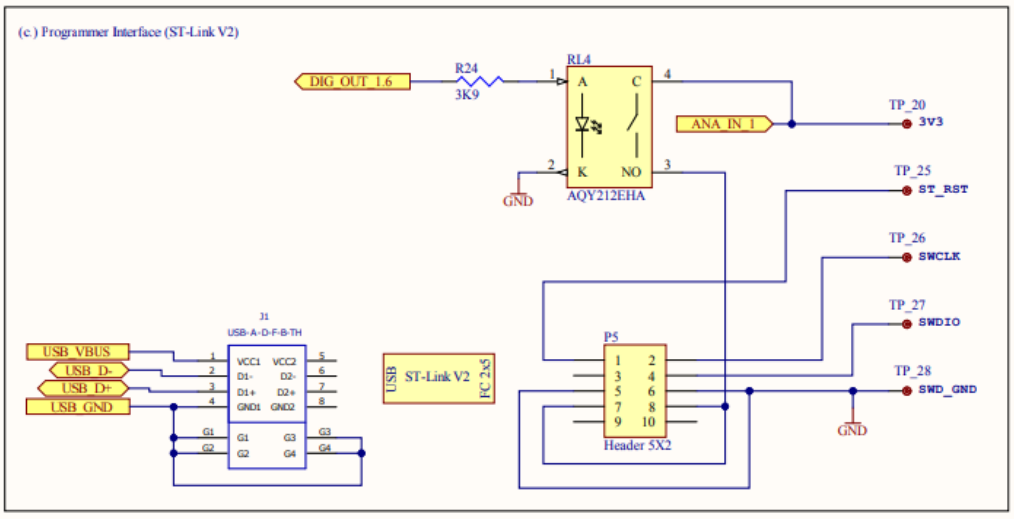


Figure 5.8: Programmer hardware block schematic

5.1.2.4 Buzzer Test hardware block (d.)

The Buzzer test hardware block is built as a method to properly use the SPU0410LR5H MEMS microphone, by implementing a signal amplifier based upon an inverting operational amplifier circuit with an LM358. The microphone and OPAMP components were chosen based on cost and availability at the suppliers, as no compatible components existed on the companies' stock. Accord-

ing to the datasheet (see section 5.1.3.1), the microphone overload point is sufficiently high for the maximum 110 dB SPL output of the UUT buzzer. Using the sensitivity and the acoustic overload point values along with the DC output we can see that the maximum voltage output of the microphone is 0.95 V.

With that information, the amplifier circuit can be assembled, taking care so the output does not surpass the 12 V supply voltage of the OPAMP, while using common resistor values. The V_{REF} voltage on on pin AIN+ of the OPAMP was determined as 0.72 V, as the DC Output of the microphone is 0.73 V and as such the amplifier output has less probability of hitting the 0 V saturation point. Table 5.4 lists all of the hardware pins of the block and figure 5.9 shows the hardware block circuit.

Table 5.4: Buzzer test hardware block pins

Signal	Direction	Type	Interface
+12	In	Power	Module
+3V3	In	Power	Module
DIG_IN_1	Out	Signal	ATE
Buzzer	In	Signal	UUT
ANA_IN_4	Out	Signal	ATE

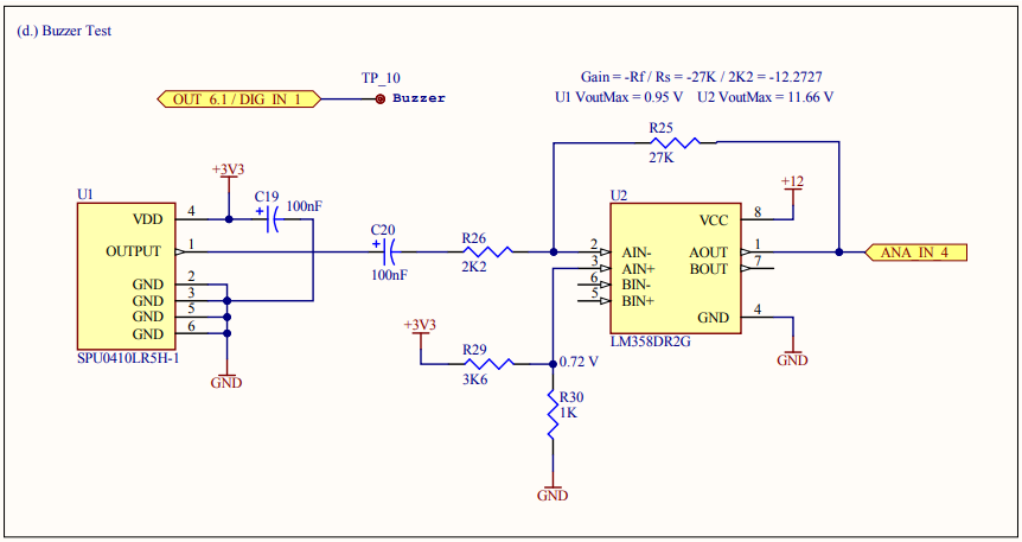


Figure 5.9: Buzzer Test hardware block schematic

5.1.2.5 LED RGB Sensors hardware block (e.)

The LED RGB sensors hardware block is assembled as the interface between the ATE and the APDS-9253-001 RGB light sensors. The circuit includes sixteen sensors, one for each LED to be tested on the UUT, all connected in a way that uses the I2C communication protocol. As each sensor has the same I2C address,

it is necessary to have a cheap strategy to communicate with only one at a time, as such two PCA9548A I2C switches, set to different addresses, are used as a way to only communicate with a selected RGB sensor at a time. The remaining hardware used in the block is to keep the I2C lines active high as well as the interrupt outputs of the RGB sensors. The interrupt lines are to be configured and used in the test to quickly detect if light is present in any of the LEDs instead of having the need to communicate directly with the sensors every single time. Table 5.5 lists all of the hardware pins of the block and figure 5.10 shows the unique hardware from the block circuit.

Table 5.5: LED RGB Sensors hardware block pins

Signal	Direction	Type	Interface
+3V3	In	Power	Module
I2C_SCL	In	Signal	ATE
I2C_SDA	In/Out	Signal	ATE
DIG_OUT_5.16	In	Signal	ATE
DIG_IN_3	Out	Signal	ATE
DIG_IN_4	Out	Signal	ATE
DIG_IN_5	Out	Signal	ATE
DIG_IN_6	Out	Signal	ATE
DIG_IN_7	Out	Signal	ATE
DIG_IN_8	Out	Signal	ATE
DIG_IN_9	Out	Signal	ATE
DIG_IN_10	Out	Signal	ATE
DIG_IN_11	Out	Signal	ATE
DIG_IN_12	Out	Signal	ATE
DIG_IN_13	Out	Signal	ATE
DIG_IN_14	Out	Signal	ATE
DIG_IN_15	Out	Signal	ATE
DIG_IN_16	Out	Signal	ATE
DIG_IN_17	Out	Signal	ATE
DIG_IN_18	Out	Signal	ATE

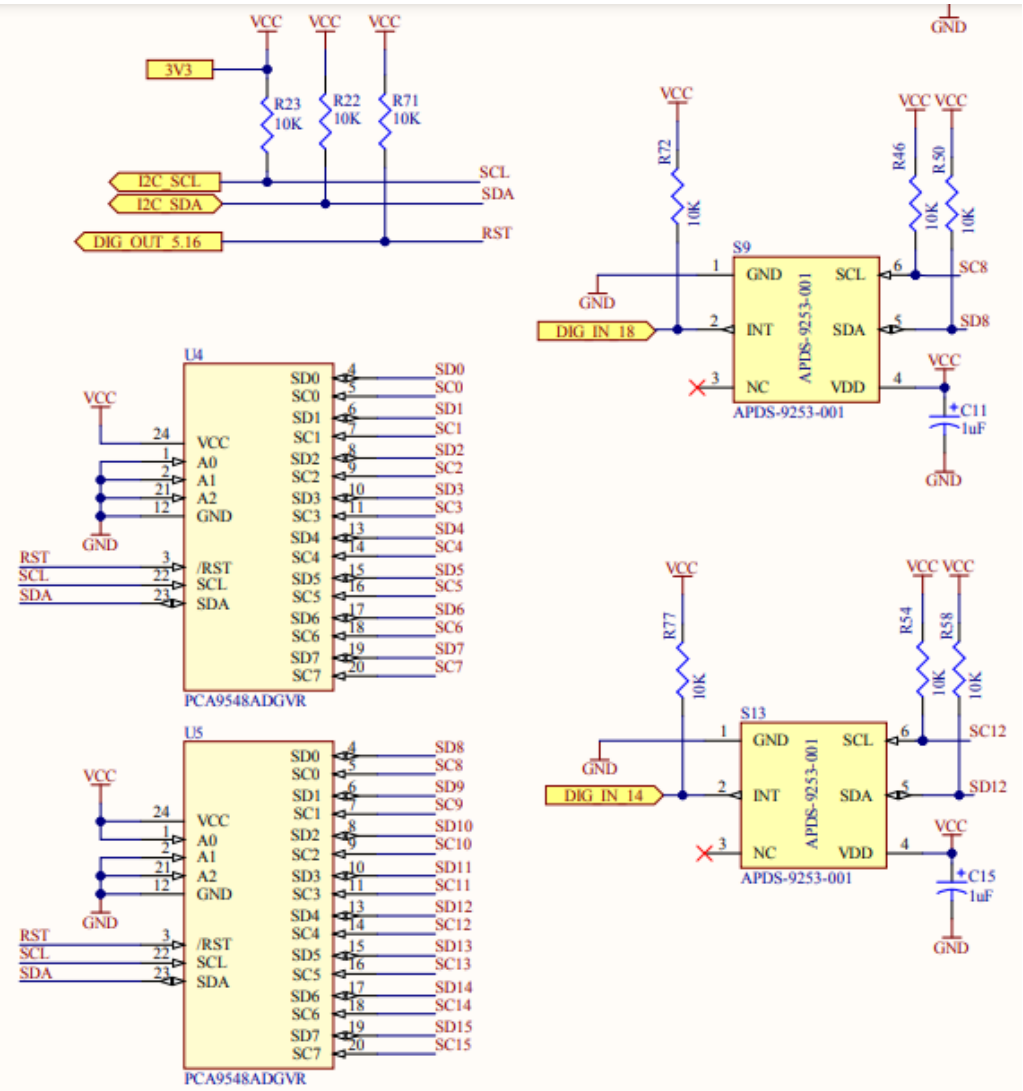


Figure 5.10: LED RGB Sensors hardware block schematic (the remaining sensors are not shown)

5.1.2.6 Other necessary hardware

In order for the module to work properly with the TEST-OK TEST-TRACK software, it is necessary for it to have a configured address using a PISO shift register. The one used is the SN74HC165D, recommended in the official documentation from TEST-OK. The configured address is the hexadecimal 0x09, as there are only eight other programs developed for the TEST-TRACK. Table 5.6 lists all of the hardware pins of the block and figure 5.11 shows the unique hardware from the block circuit.

Table 5.6: Identification hardware block pins

Signal	Direction	Type	Interface
SUPPLY_5V	In	Power	ATE
EXT_CLK	In	Signal	ATE
EXT_LOAD	In	Signal	ATE
EXT_IN	Out	Signal	ATE

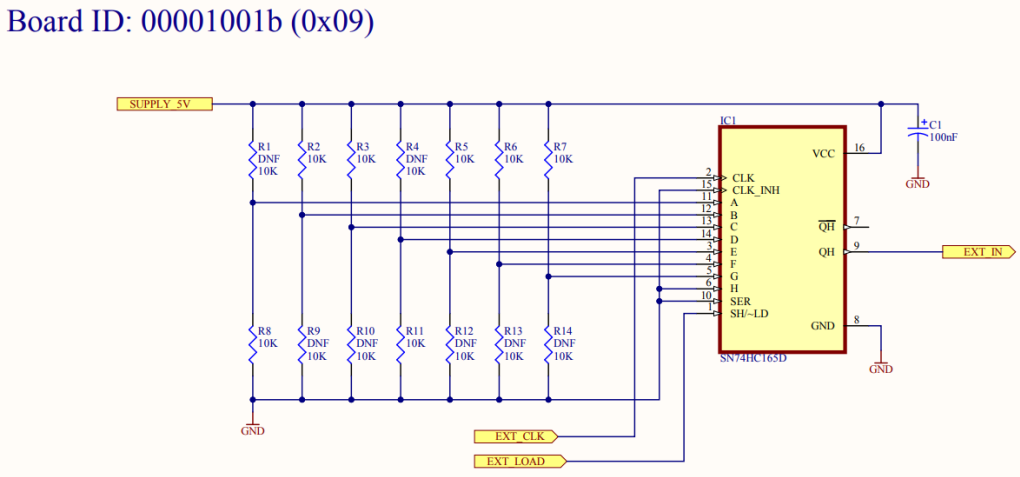


Figure 5.11: Identification hardware block schematic

5.1.3 Hardware components used

Most components used in the hardware project are straight forward in their functionality, however, four of them are more specialized components that need a more thorough explanation of how they were used in the project and, if applicable, what hardware requirements they have documented on their data sheets.

5.1.3.1 SPU0410LR5H - MEMS microphone

The SPU0410LR5H from SiSonic is a miniature, omnidirectional and low power bottom port microphone. Using MEMS technology, it consists of an acoustic sensor, a low noise input buffer and an output amplifier. These devices are suitable for applications where wide band audio performance and radio frequency immunity are required. Figure 5.12 shows the pin out of the component and table 5.7 shows the functionality associated to each pin.

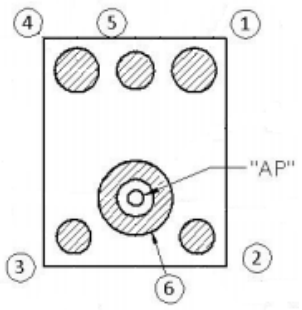


Figure 5.12: SPU0410LR5H pin out diagram

Table 5.7: SPU0410LR5H pin functions

Name	Pin	Type	Description
OUT	1	Out	Output signal
GND	2	Power	Power supply ground
GND	3	Power	Power supply ground
VDD	4	Power	Power supply voltage
GND	5	Power	Power supply ground
GND	6	Power	Power supply ground

The data sheet recommends using an inverting amplifier circuit as an interface to the microphone, with Direct Current (DC) filtering capacitors at the output of the component. In order to properly assemble the circuit for the output to be visible on the ATE ADC input, the data sheet specifications were consulted and the relevant data on table 5.8 was extracted, see section 5.1.2.4 for more information. Figure 5.13 shows the circuit suggested in the data sheet.

Table 5.8: SPU0410LR5H data sheet relevant data

Parameter	Typical	Units
Sensitivity	-38	dBV/Pa
Acoustic Overload Point	118	dB SPL
DC Output	0.73	V

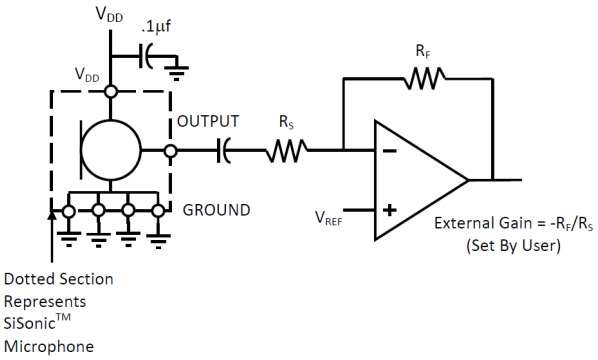


Figure 5.13: SPU0410LR5H circuit example

5.1.3.2 APDS-9253-001 - RGB light sensor

The APDS-9253-001 from Broadcom is a miniature I2C digital RGB sensor with four individual channels of red, green, blue and InfraRed (IR) light detectors. The component has good angular response and accurate RGB spectral response with a high accuracy for light intensity measures over various light sources even through dark glass. The component can be configured, using I2C, as a Ambient Light Sensor (ALS) or as a RGB+IR light sensor. It has programmable interrupt functions using upper and lower thresholds and also persistence functions that can be configured to allow for more immediate sensing capabilities. Figure 5.14 shows the functional diagram of the component and table 5.9 shows the functionality associated to each pin.

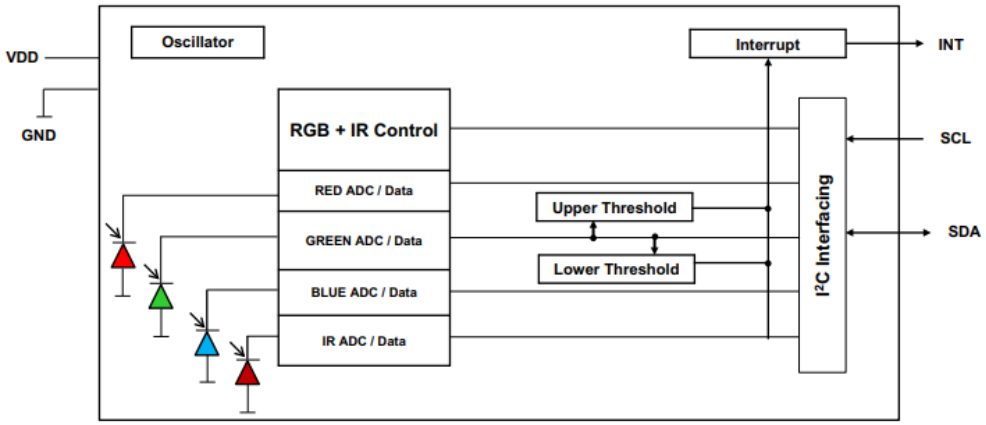


Figure 5.14: APDS-9253-001 functional diagram

Table 5.9: APDS-9253-001 pin functions

Name	Pin	Type	Description
GND	1	Power	Power supply ground
INT	2	Out	Open drain interrupt

Name	Pin	Type	Description
NC	3		Not Connected
VDD	4	Power	Power supply voltage
SDA	5	In/Out	I2C serial data
SCL	6	In	I2C clock input

In the context of this project, the sensor will be used to detect RGB light and the upper threshold interruptions will be configured in order to detect the intensity of the light during the various tests performed. The upper threshold works as a minimum value of light intensity, after which the interruption is triggered. In order to use the sensor for the various functions it is going to perform during the test, the data sheet was consulted and the I2C addresses on table 5.10 were retrieved.

Table 5.10: Addresses used in the I2C communications

Address Name	Address	B7	B6	B5	B4	B3	B2	B1	B0
RD_G0	0x0D	Green data 0							
RD_G1	0x0E	Green data 1							
RD_G2	0x0F	0	0	0	0	Green data 2			
RD_B0	0x10	Blue data 0							
RD_B1	0x11	Blue data 1							
RD_B2	0x12	0	0	0	0	Blue data 2			
RD_R0	0x13	Red data 0							
RD_R1	0x14	Red data 1							
RD_R2	0x15	0	0	0	0	Red data 2			
INT_THR_UP0	0x21	Upper Threshold 0							
INT_THR_UP1	0x22	Upper Threshold 1							
INT_THR_UP2	0x23	0	0	0	0	Upper Threshold 2			
SET_INT	0x19	0	0	SEL		VAR	EN	0	0

RD_Rx, RD_Gx and RD_Bx are the registers corresponding to the Red, Green and Blue sensors on the device. They store the color component read by the sensor until it receives a I2C command that reads that value. These registers are divided in three as the value detected is converted to a 20 bit length word that is read using I2C: two bytes and one 4 bit word. Likewise, the INT_THR_UPx registers also hold a 20 bit word corresponding to the interrupt upper threshold configuration to use by the device. Finally, SET_INT holds the interruption configuration, which is detailed on table 5.11.[17]

Table 5.11: Interrupt configurations for SET_INT

Name	Bit	Description
SEL	B5:B4	00: IR detector
		01: ALS detector/Green detector (default)
		10: Red detector
		11: Blue detector
VAR	B3	0: Threshold interrupt mode (default)
		1: Variation interrupt mode
EN	B2	0: Interrupt disabled (default)
		1: Interrupt enabled

5.1.3.3 PCA9548A - I2C 8 channel switch

The PCA9548A from Texas Instruments is a bidirectional eight channel switch to be used and controlled using I2C. The input pair of SCL/SDA signals can be branched out to eight individual devices or a combination of them, all depending on the control by the user. This device can be used to solve conflicts between different devices with the same I2C slave address. The switch can be reset by the master at any time using the reset input which deselects all the channels and initializes the state machine. The pins allow the use of any voltage up to 5 V, which can be useful in level translation applications and also up to 400 kHz signal frequency, allowing the use of the Standard and Fast I2C modes. The address of the device can also be set using the three hardware address pins, enabling the use of a maximum of eight switches at the same time. Figure 5.15 shows the pin out diagram of the component and table 5.12 shows the functionality associated to each pin.

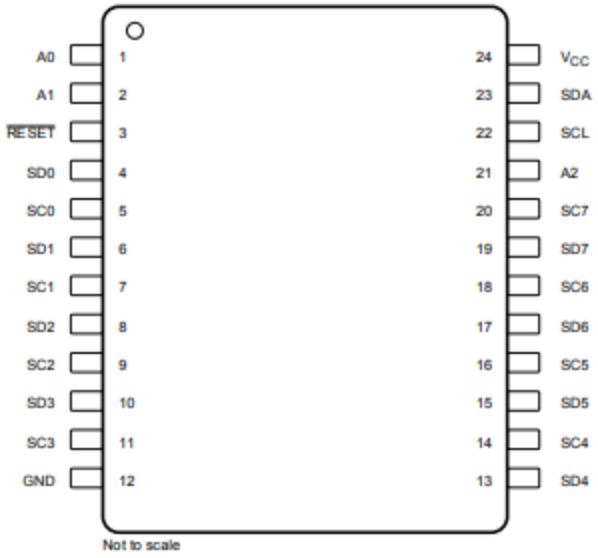


Figure 5.15: PCA9548A pin out diagram

Table 5.12: PCA9548A pin functions

Name	Pin	Type	Description
A0	1	In	Address input 0
A1	2	In	Address input 1
$\overline{RESET}$	3	In	Active-low reset
SD0	4	In/Out	Serial data channel 0
SC0	5	Out	Serial clock channel 0
SD1	6	In/Out	Serial data channel 1
SC1	7	Out	Serial clock channel 1
SD2	8	In/Out	Serial data channel 2
SC2	9	Out	Serial clock channel 2
SD3	10	In/Out	Serial data channel 3
SC3	11	Out	Serial clock channel 3
GND	12	Power	Power supply ground
SD4	13	In/Out	Serial data channel 4
SC4	14	Out	Serial clock channel 4
SD5	15	In/Out	Serial data channel 5
SC5	16	Out	Serial clock channel 5
SD6	17	In/Out	Serial data channel 6
SC6	18	Out	Serial clock channel 6
SD7	19	In/Out	Serial data channel 7
SC7	20	Out	Serial clock channel 7
A2	21	In	Address input 2
SCL	22	In	I2C clock input
SDA	23	In/Out	I2C serial data

Name	Pin	Type	Description
VCC	24	Power	Power supply voltage

In the context of this project, two devices will be used to connect to each one of the 16 I2C RGB sensors individually as all of them come with the same I2C factory address. In order to use the devices properly, the data sheet and figure 5.16 were consulted and the I2C addresses on table 5.13 were retrieved.

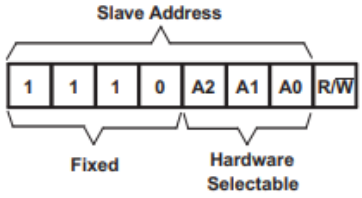


Figure 5.16: Address byte organization

Table 5.13: I2C commands used in the switch

Address Name	Address	Description
FIRST_GROUP_W	0xE0	Address 0x70 + Write bit 0
FIRST_GROUP_R	0xE1	Address 0x70 + Read bit 1
SECOND_GROUP_W	0xE2	Address 0x71 + Write bit 0
SECOND_GROUP_R	0xE3	Address 0x71 + Read bit 1
ADD_S1	0x01	Connects switch to the first device
ADD_S2	0x02	Connects switch to the second device
ADD_S3	0x04	Connects switch to the third device
ADD_S4	0x08	Connects switch to the fourth device
ADD_S5	0x10	Connects switch to the fifth device
ADD_S6	0x20	Connects switch to the sixth device
ADD_S7	0x40	Connects switch to the seventh device
ADD_S8	0x80	Connects switch to the eighth device

5.1.4 TEST-OK Test Module development

The test module was designed using Altium and a template from TEST-OK. The template has modules designed with the identification hardware, the connectors and the special mechanical components to use in the assembly of the module as well as the board test. Of the mechanical components provided by TEST-OK to use in the assembly of test modules, the most noteworthy are the Fast Lock blocks and the positioning pins. These mechanical components are used together to position and hold the UUT in place so the test probes can come in contact with the board test points safely. Fast locks are dynamic components

that hold the UUT in place when the test probes are pushed against the UUT but leave it loose when the probes are away. Figure 5.17 shows an example of a board held by Fast Locks and positioning pins.

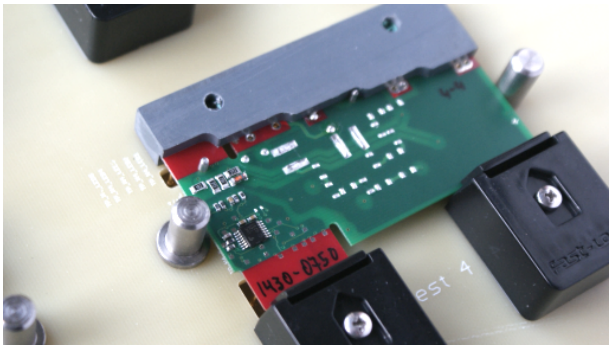


Figure 5.17: An example of a UUT held by Fast Locks and positioning pins

The test module may be comprised of either three or five PCBs depending on whether the test interacts with only the bottom of the UUT or the top and the bottom. In this case only three PCBs were designed as only one of the surfaces of the UUT will be interacted with. As such the design will involve a complete PCB with copper layers named Bottom Connector Board (BCB) and two boards which will not have any copper or finish named Bottom Spacer Board (BSP) and Positioning Board (PB). The BCB will have all of the test module hardware blocks, the remaining two only serve as mechanical supports, BSP for the test probes and PB to hold the UUT. Figure 5.18 shows the example of how two systems are assembled during the test, the top module uses three PCBs and the bottom module uses five PCBs.

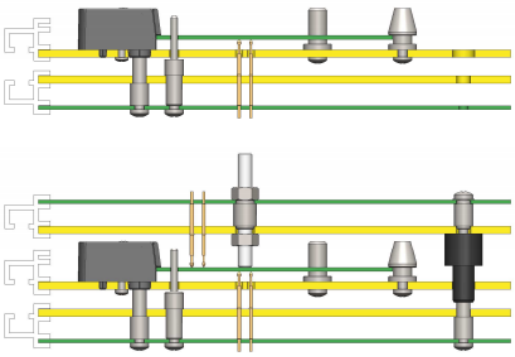


Figure 5.18: A tree PCB module (top) and a five PCB module (bottom)

The design of the hardware module was done by drawing the schematic in Altium and experimenting with the PCB layout in order to achieve the final design. The first step was to define the best position for the UUT to perform the test. It was chosen that the UUT shall be tested flipped in order to observe the

LEDs using RGB sensors on the test module, as well as to facilitate the use of test points as explained in section 5.1.1. The UUT position was drawn on the BCB and all the components were set accordingly, with the test probes in their respective places and the RGB sensors directly underneath the LEDs. Figure 5.19 shows the 3D view of the first design to be made.

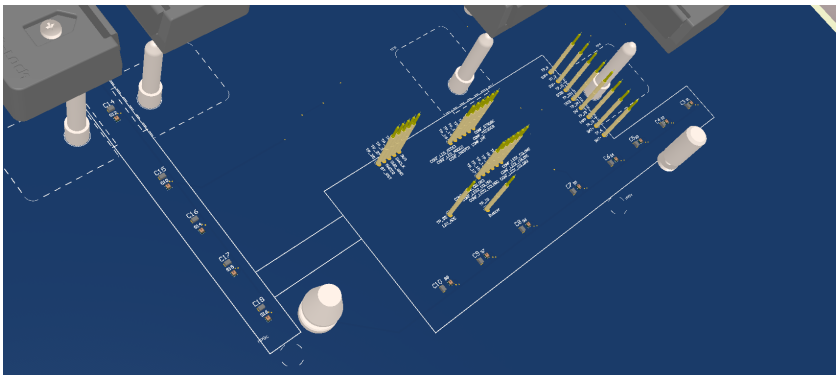


Figure 5.19: BCB first design in 3D view

Since the board needs to be tested upside down, cuts need to be made in the PB so the UUT can be set and safely secured using the TEST-OK mechanical components without damaging the board or the UUT components. In order to more easily see where the cuts should be preformed, a cardboard cutout with the TEST-OK mechanical components was made and cuts were performed one at a time until the UUT could be safely set with no obstruction and no components in contact with the PB. Figure 5.20 shows a cardboard cutout prototype of how the PB cuts will be.

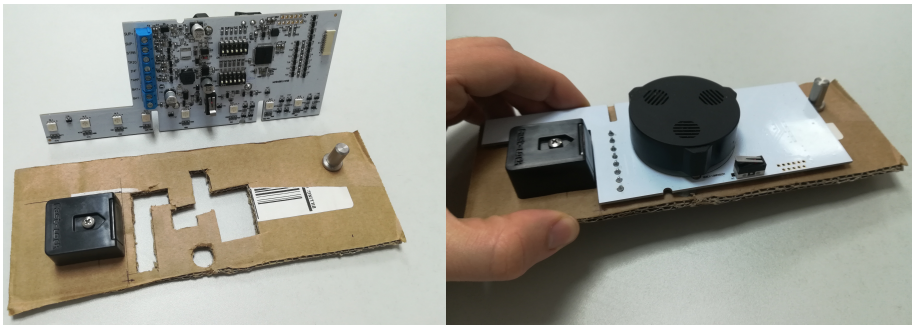


Figure 5.20: Cardboard prototype with holes cutout

The cuts were drawn on the PB after exporting the UUT PCB silk screen and pads to Altium, these served as guides to align all of the cuts and preview how the components would be set based upon the study performed with the cardboard. Figure 5.21 shows how the cuts will be made in the final module.

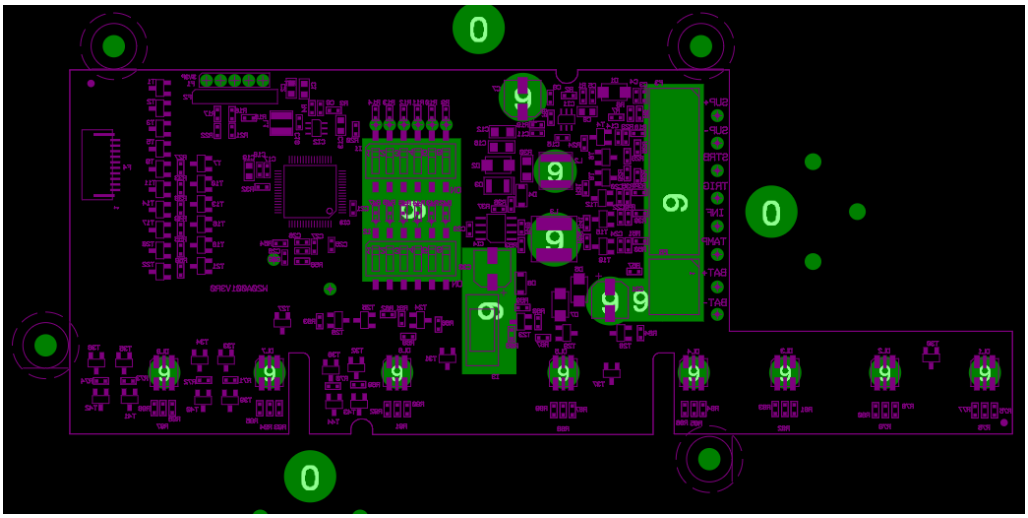


Figure 5.21: Cuts as displayed in Altium

Next the routing was made manually, with special care on the USB D+ and D- lines as they are a differential pair and need to be drawn next to each other, over a GND plane and the track needs to have the exact same length in order to work properly. The first version of the board used all four layers available in the template, but the second version used only two layers in order to save costs in production, as the four layers were not necessary. Figure 5.22 shows the first version of the BCB board with the four layers and figure 5.23 shows the two layer version corrected for the same circuit.

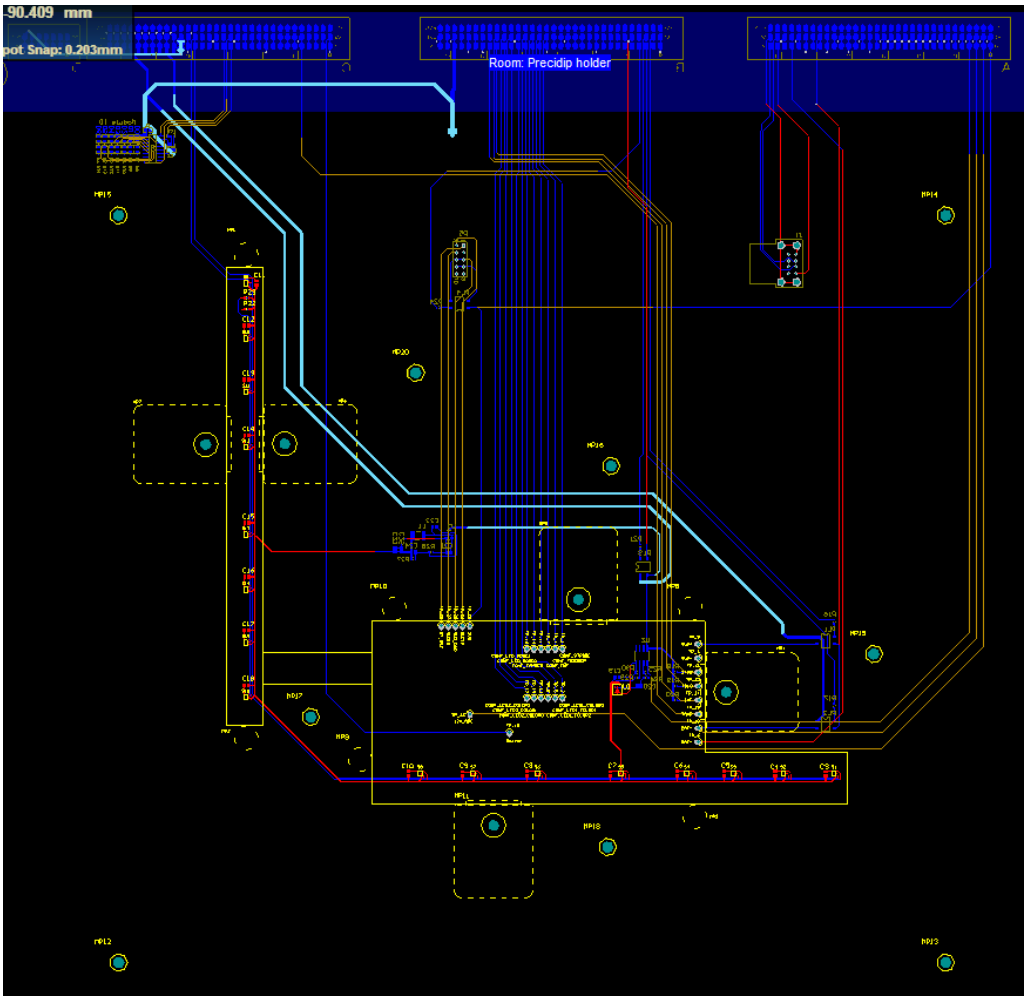


Figure 5.22: First version of the routed BCB

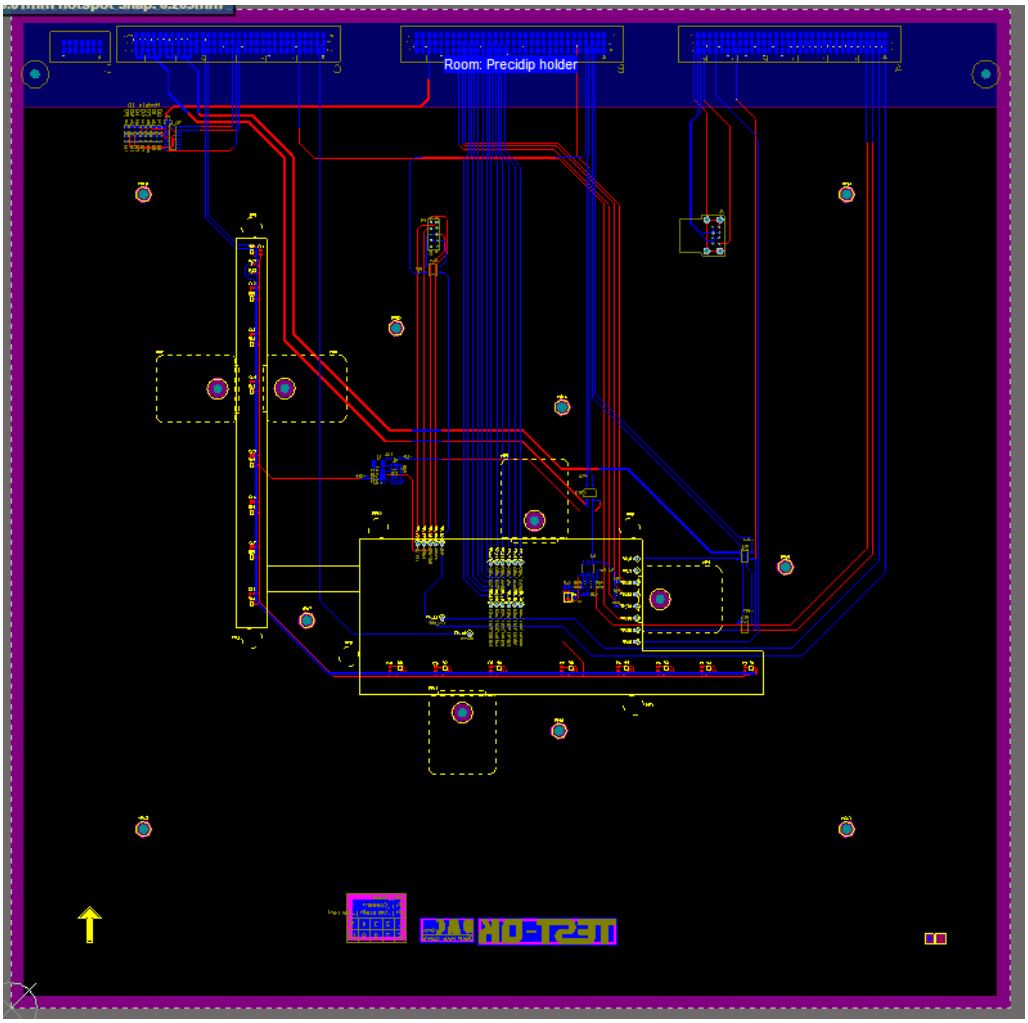


Figure 5.23: Second version of the routed BCB

After the routing was made, several iterations followed to minimize the amount of vias used in each track in order to keep the connections as sturdy as possible, especially the ones that carried signals from or to the UUT. Figure 5.24 shows the third version which was changed to add more vias to the GND plane in order to make them more electrically sturdy as well.

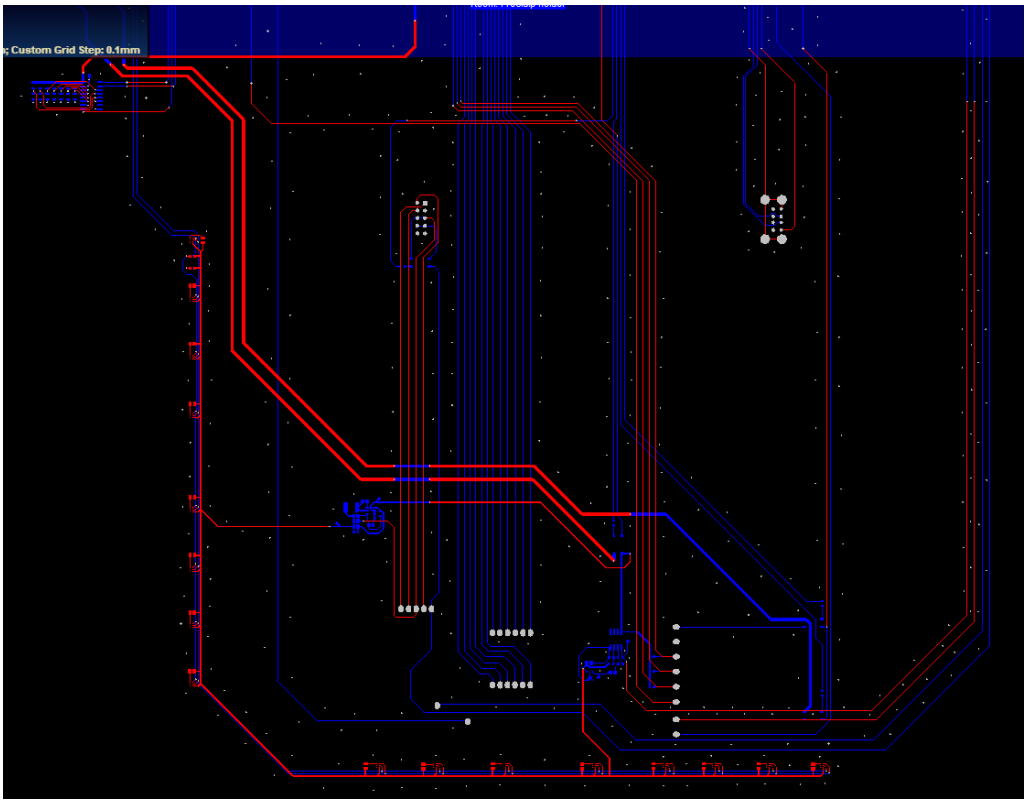


Figure 5.24: Third version of the routed BCB

The fourth version of the boards had the Tecnimaster logo added to all three boards along with their descriptions and also added more hardware to be used to address each RGB sensor properly. This came as a correction to a mistake that was identified when the software execution was being thought out, as each RGB sensor has the same factory I2C address, hardware was necessary to help perform the communication to each one individually. Figure 5.25 shows the fourth version of the design.

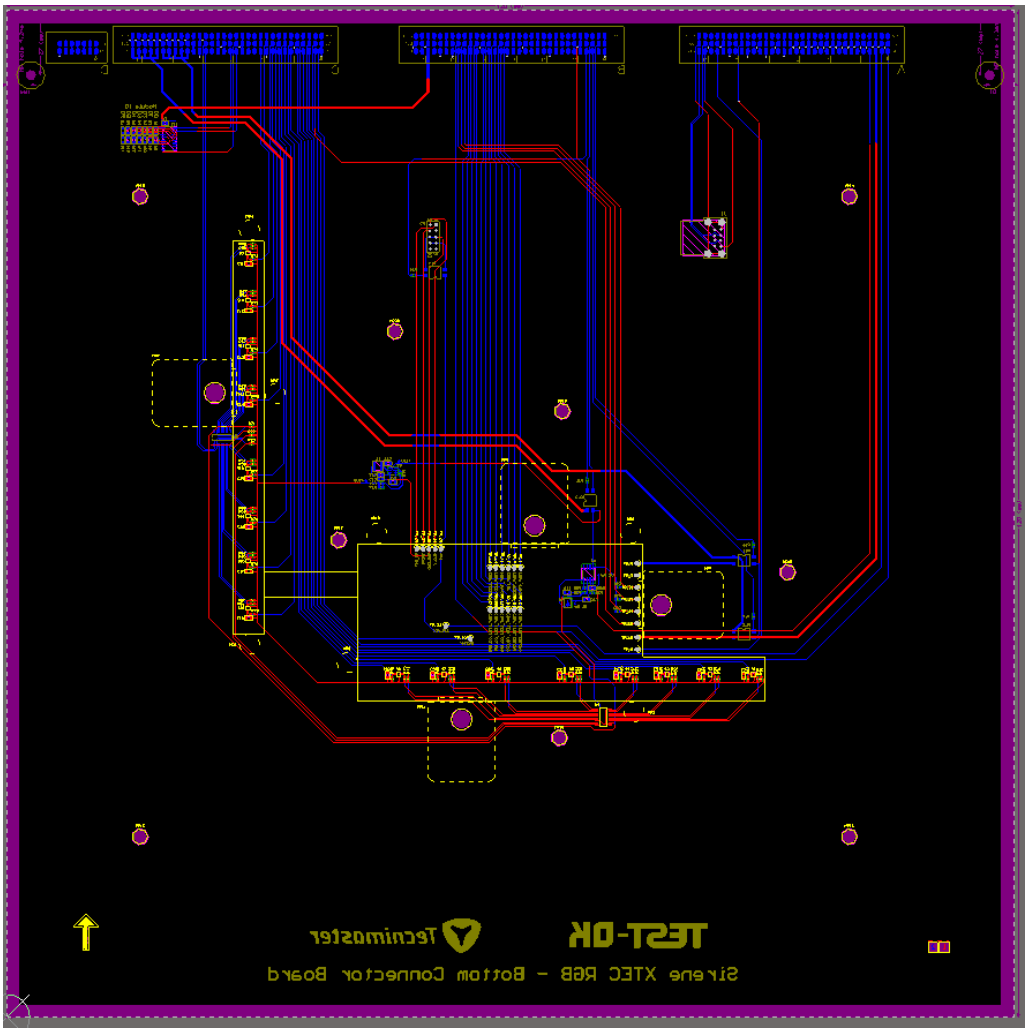


Figure 5.25: Fourth version of the routed BCB

The final version added an internally requested option to power the UUT and the test module externally and came as a suggestion after the hardware and test were reviewed by the technicians at the company. Another suggestion was made to include processing options within the module in order to account for possible errors in the ATE. However, time constraints became an obstacle and that change would require further delays to the conclusion of the project. Figure 5.26 shows the final version of the design.

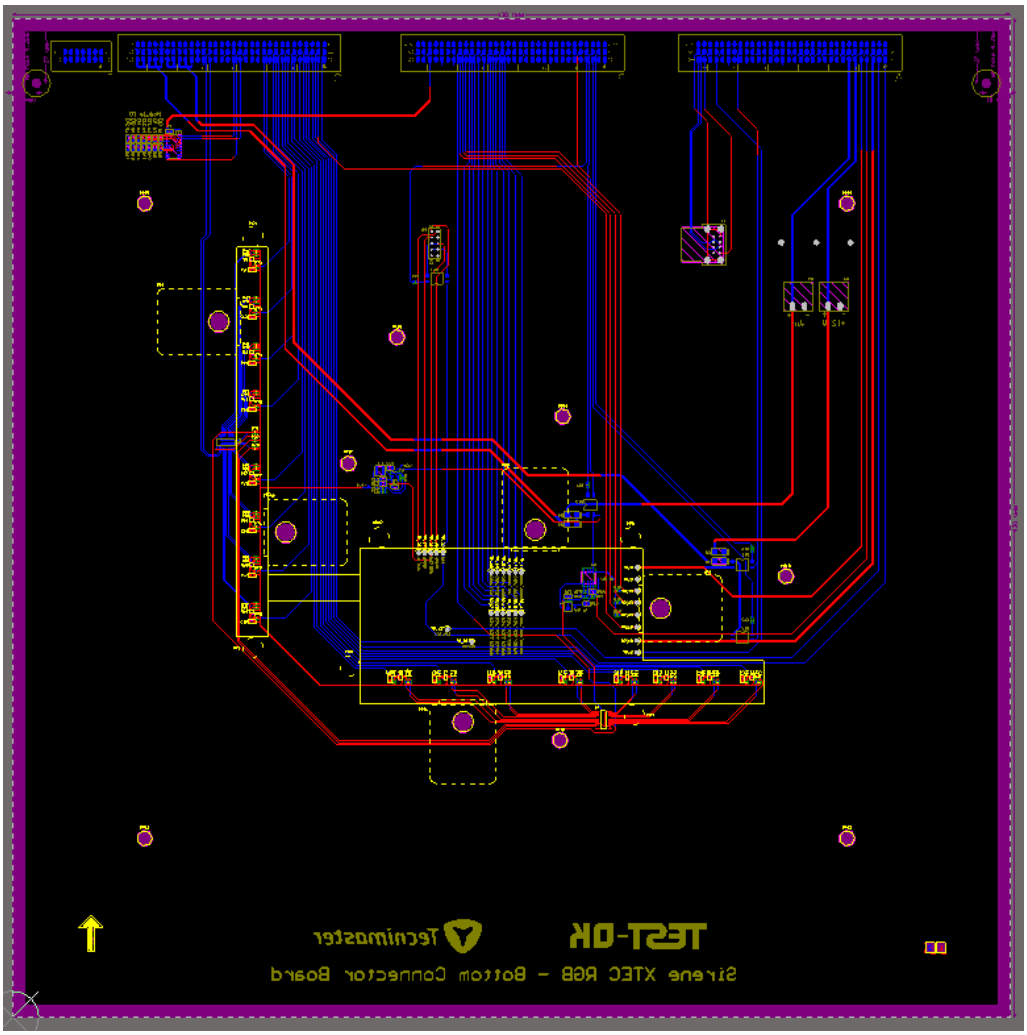


Figure 5.26: Final version of the routed BCB

After all the modifications were performed, a GND plane was drawn to connect all the GND signals to the same potential and further inspection of the design was performed using the 3D view on Altium and verifications of the exported Gerber files. Everything was corrected as the design was being developed and the final images are shown below. Figure 5.27 displays the 3D view of the top layer of the PCB in Altium, figure 5.28 displays the 3D view of the bottom layer of the PCB in Altium, figure 5.29 shows the 3D view of the top area where the UUT will be placed during the test and figure 5.30 shows the 3D view of the bottom area where the UUT will be placed during test.

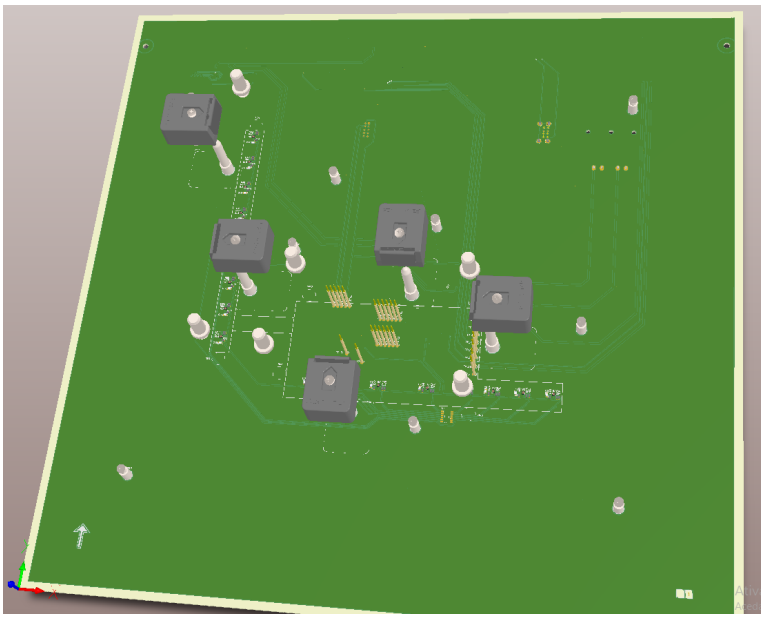


Figure 5.27: Top view of the BCB in Altium

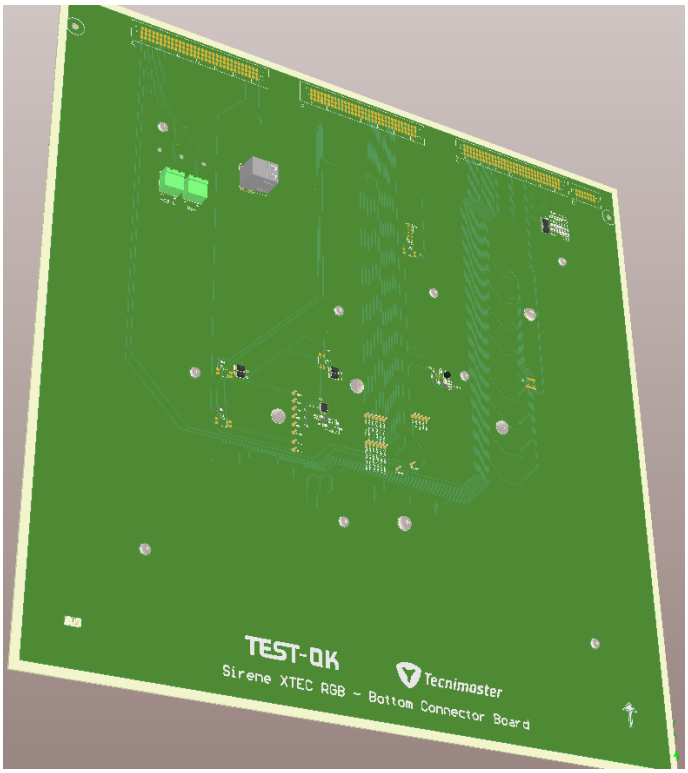


Figure 5.28: Bottom view of the BCB in Altium

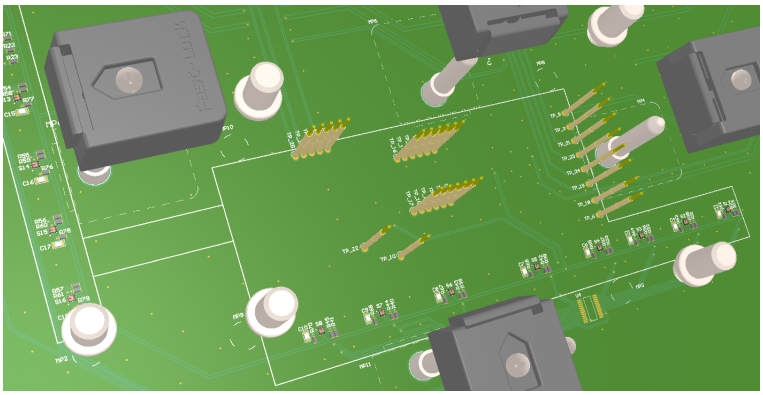


Figure 5.29: Top view of the UUT spot on the BCB in Altium

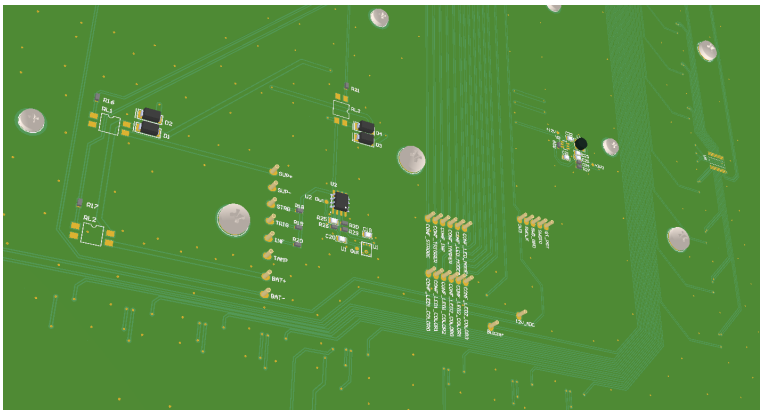


Figure 5.30: Bottom view of the UUT spot on the BCB in Altium

5.1.5 Producing the TEST-OK Test module

The final files to produce the boards were compiled and the documentation from TEST-OK was consulted so the thickness and remaining properties of the boards could be defined for production. Table 5.14 shows the properties requested for the tree boards.

Table 5.14: PCB properties requested

Board	Thickness	Material	Other properties
BCB	1.6 mm	FR4	Two copper layers with 34.7 µm thickness Green solder mask White silk screen
BSP	3.2 mm	FR4	No copper or finish
PB	3.2 mm	FR4	No copper or finish

The process to produce the test module began by requesting quotes to multiple suppliers with a preliminary version of the PCB in order to choose which

one would produce the module based on price and the time it would take to deliver. The final supplier was chosen by Tecnimaster after the initial quotes were requested and the price negotiated. Table 5.15 shows the results of the requests and highlights the two most viable solutions (names were kept confidential on purpose and the location was kept as reference).

Table 5.15: All the quotes requested (all prices are in EUR)

Supplier	BCB price	BSP price	PB price	Shipping	Total	Time to Arrive
1 (China)	Did not answer					
2 (China)	191.73	273.07	273.07	165.17	903.04	NA
3 (China)	107.07	NA	NA	NA	107.07	NA
4 (China)	**166.00**	**66.40**	**66.40**	**124.50**	**423.30**	**7 days**
5 (China)	185.09	374.33	374.33	178.45	1 112.20	6 days
6 (Portugal)	530.00	310.00	310.00	0.00	1 150.00	4 days
7 (Portugal)	**162.43**	**145.29**	**145.29**	**0.00**	**453.01**	**14 days**
8 (Belgium)	Did not answer					
9 (France)	Did not answer					
10 (Spain)	Did not produce small quantities					
11 (Germany)	250.00	190.00	190.00	0.00	630.00	40 days
Final offer from 7	**87.19**	**103.99**	**103.99**	**0.00**	**295.17**	**13 days**

Next came the materials, after a Bill of Materials (BoM) was created, various iterations of the list were made so the maximum amount of components from Tecnimaster would be used. The rest would be bought externally both to part suppliers and TEST-OK for the mechanical and test specific components. Table 5.16 displays the unique quantity, the total quantity and the cost of the components from each of the three suppliers used: Internal, Mouser and TEST-OK.

Table 5.16: Component suppliers

Supplier	Unique comp qty	Comp qty	Price
Internal	19	109	8.61 €
Mouser	7	23	25.49 €
TEST-OK	6	49	547.15 €
Total	**32**	**181**	**581.25 €**

After the production of the PCBs was started, the stencils were ordered and the output for a *pick and place* assembly program was requested to allow the production of the assembled board to be quicker and more precise.

When the boards arrived, a flaw was detected that made it impossible for the production to be completely automatic, as the fiducial markers were not put

on the BCB. This flaw forced the board to go through a near manual assembly, as the solder paste was passed to the pads using the stencils, the components were manually set in their proper places and finally the board was heated in the oven, so the components could be soldered. Figure 5.31 shows the placement of the components and the result after the oven.

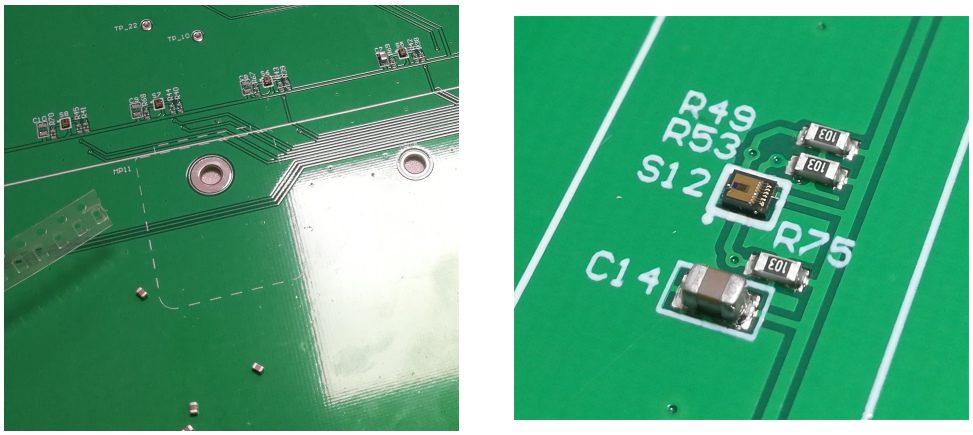


Figure 5.31: Components being placed on the left and components soldered after going to the oven on the right

During the placement of the components a new flaw came to light as one of the bought components had the wrong package. The footprint on BCB was designed for the component PCA9548A**PWR**, but the component that was bought was the PCA9548A**DGVR**. The error occurred because the original component had no stock in any of the suppliers and the alternative package dimensions were not checked for compatibility. To solve this problem, a third alternative was found, PCA9548A**DBR**, with a more compatible package and the same pitch between pins. Figure 5.32 shows the incompatibility between the footprint and the initially bought components.

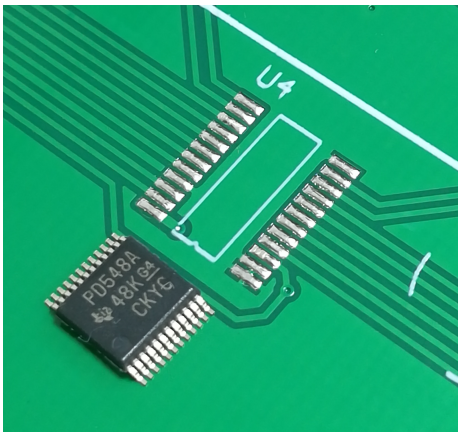


Figure 5.32: Comparison between the wrong IC and the footprint

After the assembly of the BCB, the mechanical components were assembled and a third flaw was found, as the thickness of the PB and BSP boards was too much to fit on the metallic frame from TEST-OK. As such the board receptacle on the frame was slightly enlarged to accommodate a 3.2 mm thick board, when it is designed for 3.0 mm thickness. This mistake occurred because the documentation lists the acceptable thickness as 3.0 +0.2/-0, but as it was it was impossible for the board to be connected to the frame.

The following figures 5.33 to 5.35 follow the assembly of the BCB, BSP and PB.

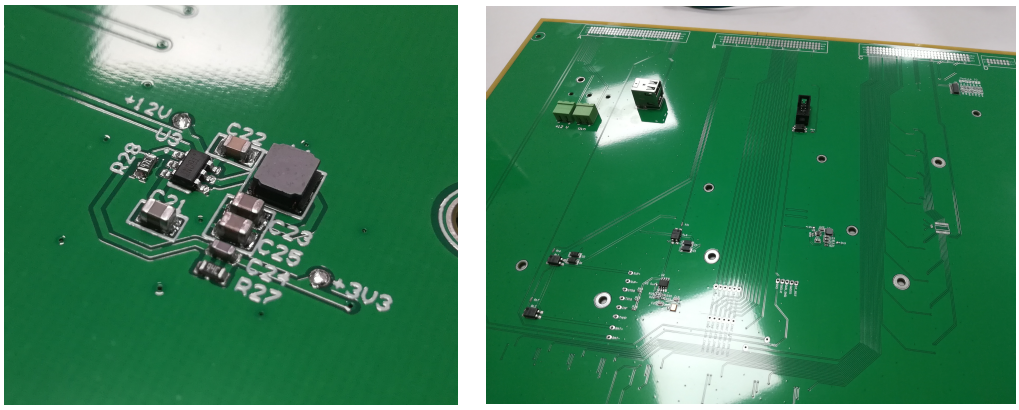


Figure 5.33: SMD components assembled on the left and THT components assembled on the right

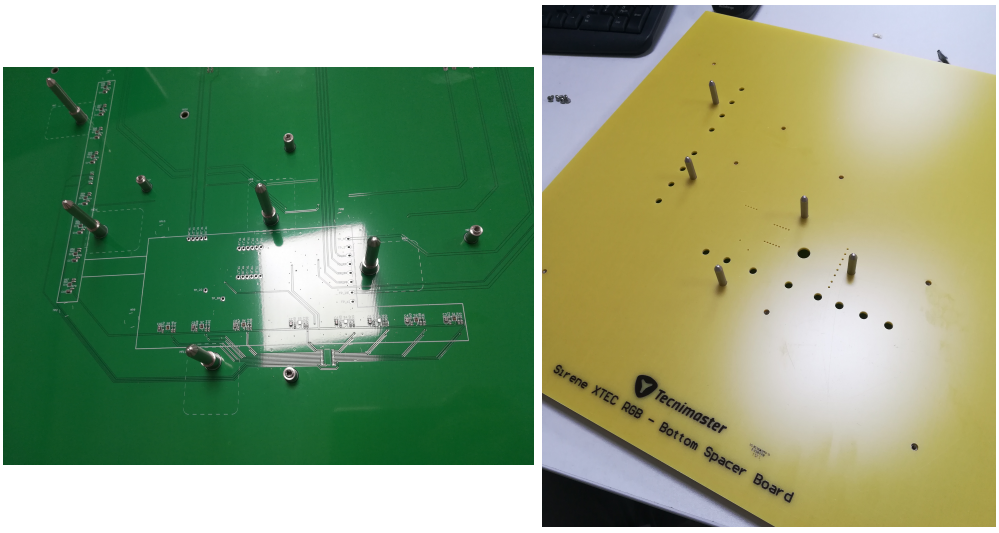


Figure 5.34: Mechanical components assembled on the left and BCB and BSP assembled on the right

Figure 5.35: All boards assembled before the test probes were soldered

After the assembly and validation of the BCB, the click mechanism for the button was done as a bit of foam that was positioned in a way that when the test module is in use, the switch on the UUT is pressed. The test probes were also connected into each of the receptacles and the fast lock components were adjusted on the PB. All the hardware corrections performed during validation are mentioned in section 6.1.1. Figure 5.36 shows the BCB and BSP test modules assembled, figure 5.37 shows the PB assembled to the rest of the test module along with how it looked after being adjusted to the UUT and figure 5.38 shows the TEST-OK system at work with everything assembled.

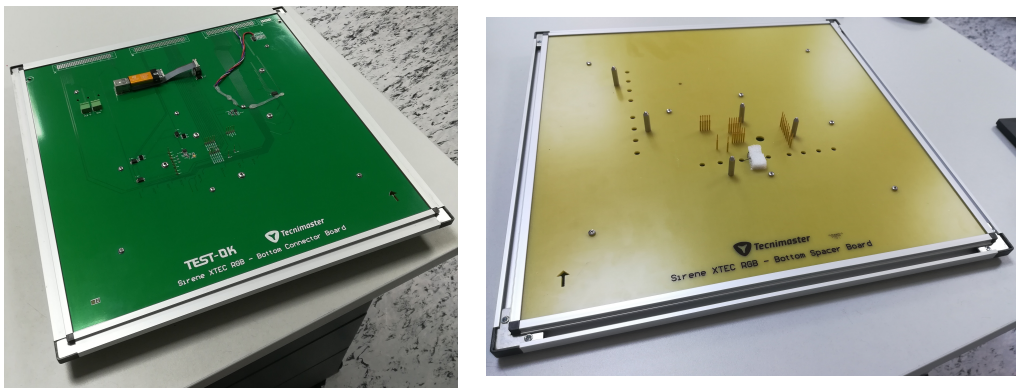


Figure 5.36: Bottom of the test module on the left and top of the test module on the right

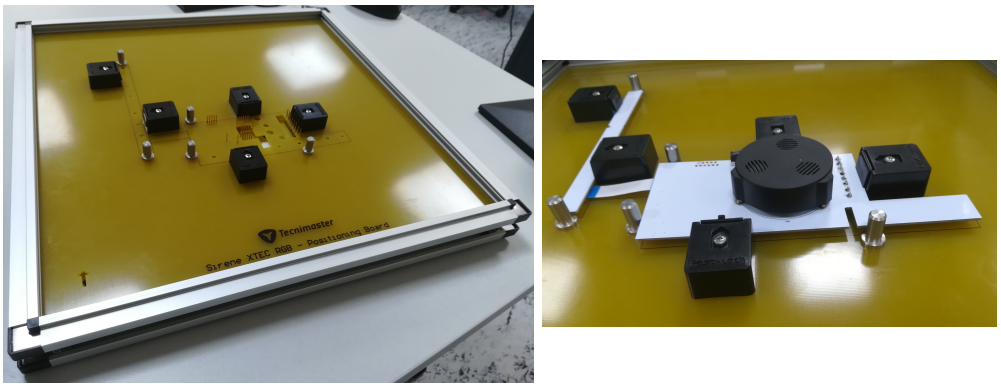


Figure 5.37: The full test module assembled on the left and the PB adjusted to the UUT on the right

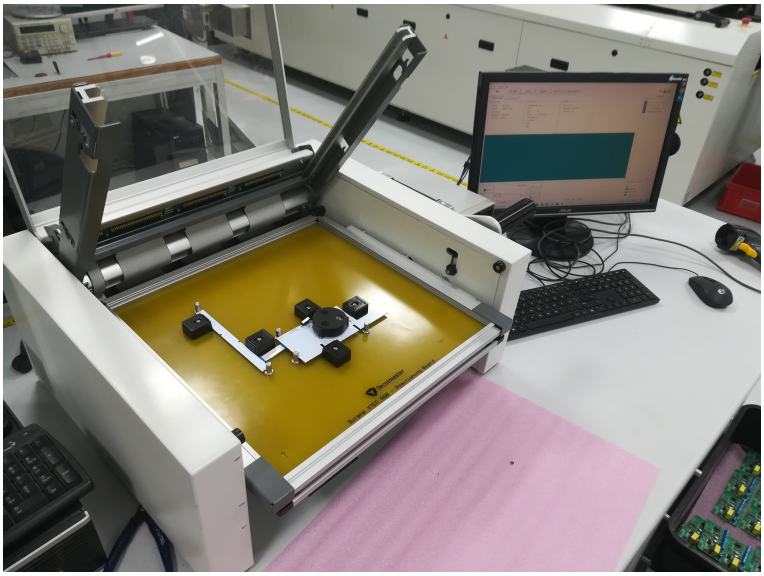


Figure 5.38: TEST-OK system with everything assembled

5.1.6 RGB Siren PCB modifications

The original board had no test points to use the test probes in, thus it was necessary to change the design of the PCB so the board would be compatible with the developed test system. As such, Altium was used to import the Gerber files of the main board and reverse engineer all the tracks and pads for every component. After that, the test points were drawn and the necessary components moved, so the test points would have enough space to be used safely. Another change was the inclusion of a transient voltage suppressor SMBJ28A connecting in parallel to the power input branch. This change was requested by the production engineering at Tecnimaster after some manual tests revealed a fault in the hardware. Figure 5.39 shows the original assembled PCB for the RGB Siren, figure 5.40 shows the preview of the modified PCB, figure 5.41 shows the changes

made as a prototype of the board and figure 5.42 shows the final product after the changes.

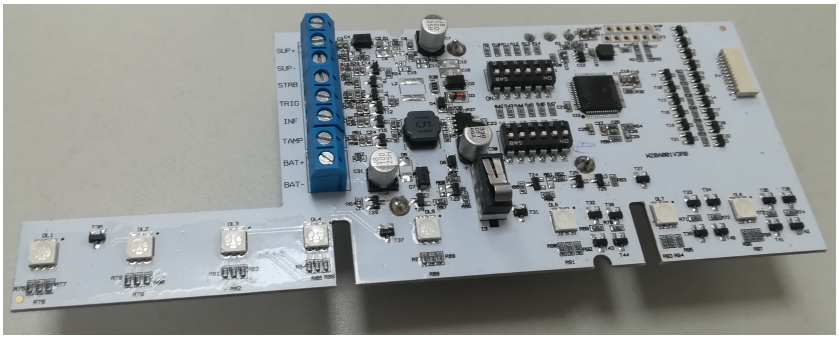


Figure 5.39: Original design of the XTEC RGB Siren main board

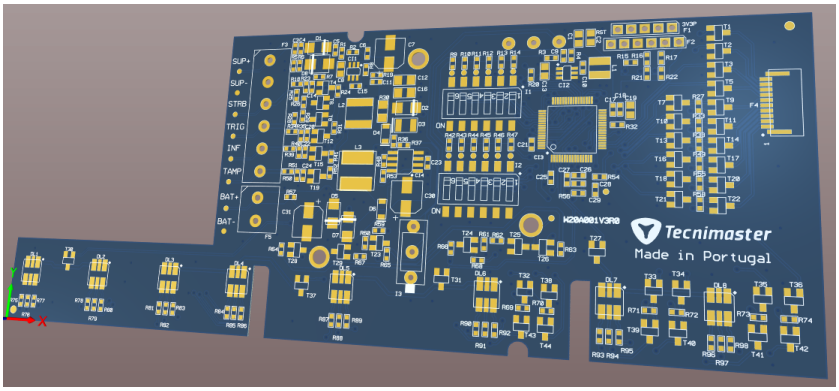


Figure 5.40: Modified design preview for the XTEC RGB Siren

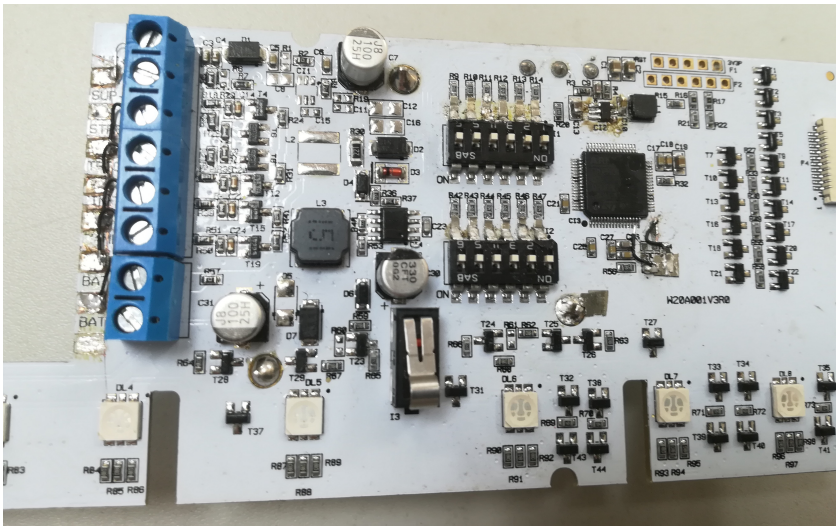


Figure 5.41: Prototype design for the XTEC RGB Siren test points

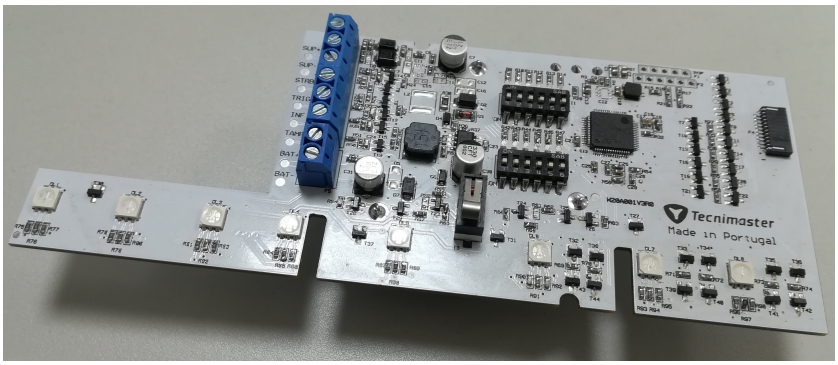


Figure 5.42: Final version of the modified design of the XTEC RGB Siren

5.1.7 TEST-OK TEST-TRACK software development

The software development began by first configuring the TEST-TRACK software to use the test program with the identification 0x09 and create the product in the internal database of the software. Next, six files were created, one for each hardware block to test and an extra one that works as a preamble to the test with pin and variable configurations. The five hardware block tests are coded in different files to allow for the test to be separated if the user desires to debug a specific fault in a block.

5.1.7.1 Preamble code

The preamble file gave proper, more intuitive names to the pins in order to be easier to use them in the test code. Table 5.17 shows the names given to the pins and with what hardware block each of them interacts with.

Table 5.17: All the named signals in preamble

Given Name	Pin	HW Block
CONTROL_SUP	DIG_OUT_1.3	a.
CONTROL_BAT	DIG_OUT_1.4	a.
CONTROL_EN	DIG_OUT_1.5	a.
CHECK_BAT	ANA_IN_3	a.
CHECK_SUP	ANA_IN_5	a.
CONTROL_TAMP_IN	DIG_IN_2	b.
CONF_STRB	DIG_OUT_5.1	b.
CONF_TRIG	DIG_OUT_5.2	b.
CONF_INF	DIG_OUT_5.3	b.
CONF_TAMP	DIG_OUT_5.4	b.
CONF_LED_MODE0	DIG_OUT_5.5	b.
CONF_LED_MODE1	DIG_OUT_5.6	b.
CONF_LED1_COLOR0	DIG_OUT_5.7	b.

Given Name	Pin	HW Block
CONF_LED1_COLOR1	DIG_OUT_5.8	b.
CONF_LED1_COLOR2	DIG_OUT_5.9	b.
CONF_LED2_COLOR0	DIG_OUT_5.10	b.
CONF_LED2_COLOR1	DIG_OUT_5.11	b.
CONF_LED2_COLOR3	DIG_OUT_5.12	b.
CONTROL_STRB	DIG_OUT_5.13	b.
CONTROL_TRIG	DIG_OUT_5.14	b.
CONTROL_INF	DIG_OUT_5.15	b.
CONTROL_TAMP_OUT	DIG_OUT_6.2	b.
3V3	ANA_IN_1	b.
12V_ADC	ANA_IN_2	b.
PROG_EN	DIG_OUT_1.6	c.
FREQ_BUZZ	DIG_IN_1	d.
MICROPHONE	ANA_IN_4	d.
S1_INT	DIG_IN_3	e.
S2_INT	DIG_IN_4	e.
S3_INT	DIG_IN_5	e.
S4_INT	DIG_IN_6	e.
S5_INT	DIG_IN_7	e.
S6_INT	DIG_IN_8	e.
S7_INT	DIG_IN_9	e.
S8_INT	DIG_IN_10	e.
S9_INT	DIG_IN_18	e.
S10_INT	DIG_IN_17	e.
S11_INT	DIG_IN_16	e.
S12_INT	DIG_IN_15	e.
S13_INT	DIG_IN_14	e.
S14_INT	DIG_IN_13	e.
S15_INT	DIG_IN_12	e.
S16_INT	DIG_IN_11	e.
RESET_I2C	DIG_OUT_5.16	e.

5.1.7.2 CPU test code

After the tool to program was requested to the original developers and the necessary arguments were researched in the official documentation, the CPU block test was developed. The tool itself is from ST Microelectronics and works as a command line programmer, accepting various arguments related to the configuration of the interface, the microcontroller to program and the .hex file to program on the device.

The CPU block is tested using the following operations:

- Set both supplies on with 12 V and 0.5 A;
- Set the programmer enable on and enable the adapter power;
- Run the STM application to program the UUT with the correct .hex file;
- End.

5.1.7.3 Buzzer test code

The Buzzer Test code was based upon the flux chart in figure 5.2. The test itself is short, only verifying the frequency of the signal that is reaching the buzzer and the output voltage of the OPAMP that is connected to the microphone, which allow the test to know the SPL of the buzzer when it is active. Table 5.18 shows the limits used during the Buzzer tests AND TABLE .

Table 5.18: Limits used in the Buzzer Test

Signal Name	Min	Max
FREQ	2300 Hz	2700 Hz
SPL	0.75 V	0.9 V

Table 5.19: Timeouts used in the Buzzer Test

Name	Value
TIMEOUT_READ	10000 ms

The Buzzer block is tested using the following operations:

- Set both supplies on with 12 V and 0.5 A;
- Set counter to frequency mode;
- Set the configuration of the signals to only activate high;
- Set the supply on and enable the adapter power;
- Set TRIG on;
- Check if the frequency is within the acceptable values;
- Check if the SPL is within the acceptable values;
- End.

5.1.7.4 Power test code

The Power test code was based upon the flux chart in figure 5.1 with the extra of also testing the Buzzer. The Buzzer test is performed right after the supply is powered under maximum load. Maximum load is reached by activating the buzzer using the Tamper signal and all the LEDs using the Strobe signal at the

same time. The test is exactly the same as the individual Buzzer test and uses the same limits as the ones in table 5.18.

After testing the Buzzer and under near maximum load, the board is powered through the Supply input with the minimum nominal voltage and the 12 V ADC, 3.3 V outputs and the Battery input are tested along with the current output from the ATE power supply. Next the ATE supply is turned off and the values of the outputs are checked to ensure everything is powered off before the following test. The final test powers the board through the Battery input with the minimum nominal voltage and the 12 V ADC and 3.3 V outputs are tested again along with the current output supply, the Supply input is tested to check there is no voltage and the circuit is working correctly. Table 5.20 shows all the limits for the measurements performed during the test.

Table 5.20: Limits used in the Power Test

Signal Name	Min	Max
12V_ADC	1.920 V	2.123 V
3V3	3.135 V	3.465 V
VBAT	7.98 V	8.82 V
VIN_CURR	0.2 A	0.45 A
VBAT_CURR	0.2 A	0.45 A
FREQ	2300 Hz	2700 Hz
SPL	0.75 V	0.9 V
0V	0 V	0.25 V

Table 5.21: Timeouts used in the Power Test

Name	Value
TIMEOUT_READ	10000 ms
TIMEOUT_BAT	10000 ms

The Power block is tested using the following operations:

- Set supply 2 on with 12 V and 0.5 A;
- Set counter to frequency mode;
- Set the configuration of the signals to only activate high;
- Set supply 1 with minimum VIN voltage and 0.5 A;
- Set the VIN supply on and enable the adapter power;
- Perform the buzzer test code;
- Set STRB on;
- Wait for the UUT to power on;

- Check if the 12V ADC voltage is within the acceptable values;
- Check if the 3V3 voltage is within the acceptable values;
- Check if the VIN supply current is within the acceptable values;
- Check if the VBAT voltage is within the acceptable values;
- Turn STRB and the supply off;
- Check if the 12V ADC voltage is at 0V within the acceptable values;
- Check if the 3V3 voltage is at 0V within the acceptable values;
- Set STRB on;
- Set supply 1 with minimum VBAT voltage and 0.5 A;
- Set the VBAT supply on;
- Wait for the UUT to power on;
- Check if the 12V ADC voltage is within the acceptable values;
- Check if the 3V3 voltage is within the acceptable values;
- Check if the VBAT supply current is within the acceptable values;
- Check if the VIN voltage is at 0V within the acceptable values;
- End.

5.1.7.5 LED test code

The LED Test code was based upon the flux chart in figure 5.3. The code is divided into two blocks, one that tests the color configurations and another that tests the LED mode configurations. In order to test all the color configurations, all the sensor interrupts will be enabled with the same threshold value using I2C, so the light can be detected immediately for each LED. The colors are tested by using two nested cycles, one that sets each of the main colors and one that checks the color values detected for each LED on the board.

After the colors are tested, each of the modes are tested. First the blink mode is tested by waiting for the interruption on the first sensor of the main board to activate. After the interruption activates, the interruption on the eighth sensor should also be active, while the interruptions on any of the sensors of the second board should be off. Next the same behaviour is observed on the second board, with no interruptions coming from the main board. The second test is to the strobe slow mode, which is done by detecting the interruption of each sensor and checking if the time it took is acceptable for the mode. The third test is to the strobe fast mode, which is similar to the one performed on the strobe slow mode, but with faster expected times and timeouts. Finally it tests the fade mode by retrieving the light intensity values for at least two cycles of fade

in and fade out and validates that all retrieved values are different and have different values within the accepted interval. Table 5.22 shows all the limits for the measurements performed during the test. During all mode tests, timeouts are defined for the time the code is waiting for an interruption to occur, these timeouts are shown on table 5.23 and table 5.27 defines the interrupt values used for each of the different tests.

Table 5.22: Limits used in the LED Test

Signal Name	Min	Max
TEMPO_STRB_F	0 ms	850 ms
TEMPO_STRB_S	0 ms	1500 ms
TEMPO_FADE	NA	4990 ms
TEMPO_BLINK	NA	1990 ms
BLINK	30	3200
FADE	NA	990
FADE_TEST	NA	4 s
B_OFF	0	1000
B_ON	1200	2500
R_OFF	0	250
R_ON	700	2500
G_OFF	0	850
G_ON	850	2500

Table 5.23: Timeouts used in the LED Test

Name	Value
TIMEOUT_STRB_F	900 ms
TIMEOUT_STRB_S	2000 ms
TIMEOUT_BLINK	2000 ms
TIMEOUT_FADE	5000 ms
MAX_GONE	2 s

Table 5.24: Interrupt values used in the LED Test

Name	Value
BLINK_INT_VAL1	1200
BLINK_INT_VAL2	800
STRB_INT_VAL	200
FADE_INT_VAL	4000

The LED block is tested using the following operations:

- Set both supplies on with 12 V and 0.5 A;
- Set the configuration of the signals to only activate high;

- Set the supply on and enable the adapter power;
- Initialize all sensors using I2C;
- Set interrupt upper threshold of BLINK_INT_VAL2 for all sensors using I2C;
- Set Blink mode;
- Set color red for all LEDs;
- Retrieve color values from all LEDs and check if they are within the acceptable values for red;
- Set interrupt upper threshold of BLINK_INT_VAL1 for all sensors using I2C;
- Set color green for all LEDs;
- Retrieve color values from all LEDs and check if they are within the acceptable values for green;
- Set color blue for all LEDs;
- Retrieve color values from all LEDs and check if they are within the acceptable values for blue;
- Restart the UUT and set all LED colors to white;
- Wait for LEDs on the main board to turn on and check if at that moment there are no other LEDs turned on on the second board;
- Wait for LEDs on the second board to turn on and check if at that moment there are no other LEDs turned on on the main board;
- Wait for LEDs on the main board to turn on and check if at that moment there are no other LEDs turned on on the second board;
- Set Strobe Slow mode;
- Set interrupt upper threshold of STRB_INT_VAL for all sensors using I2C;
- Check if every led turns on within the acceptable amount of time for a slow strobe;
- Set Strobe Fast mode;
- Check if every led turns on within the acceptable amount of time for a fast strobe;
- Set Fade mode;
- Set interrupt upper threshold of FADE_INT_VAL for sensor 1 of main board using I2C;
- Wait for the LEDs to be on at maximum intensity;

- Retrieve luminosity value samples from the LEDs until MAX_GONE timeout has been reached;
- Compare every sample and check if the difference between sample N and N+1 is within the acceptable values;
- End.

5.1.7.6 Signal test code

The Signal Test code was based upon the flux chart in figure 5.4. The test is performed using a cycle with four iterations, with only one of the signals configured on each. The code is the same for all of the iterations, and it validates that the circuit changes behaviour depending only on the configuration. Each signal input is activated at a time and the expected behaviour is checked, after that they are deactivated one at a time and the behaviour is checked once more, failing the test if any of the verifications fail. The only signal that needs special validation is TAMP which requires the user to click on the micro switch present on the board to validate that the switch is working. The expected behaviour of the circuit is the one detailed in chapter 4.3.1.4. Table 5.25 shows all the limits for the measurements performed during the test.

Table 5.25: Limits used in the Power Test

Signal Name	Min	Max
FREQ	2300 Hz	2700 Hz
0	0	100
B_OFF	0	1000
B_ON	1200	2500
R_OFF	0	250
R_ON	700	2500
G_OFF	0	850
G_ON	850	2500
TIME	NA	4999

Table 5.26: Timeouts used in the Buzzer Test

Name	Value
TIMEOUT	5000 ms

Table 5.27: Interrupt values used in the Signal Test

Name	Value
INF_G_VAL	1200
INF_R_VAL	800
STRB_INT_VAL	1500

The Signal block is tested using the following operations:

- Set both supplies on with 12 V and 0.5 A;
- Set the configuration of the signals to only activate high;
- Set all LEDs to turn on with the color blue;
- Set the supply on and enable the adapter power;
- Initialize all sensors using I2C;
- Set interrupt upper threshold of STRB_INT_VAL for sensor 2 of the main board using I2C;
- Set interrupt upper threshold of INF_R_INT_VAL for sensor 6 of second board using I2C;
- Set interrupt upper threshold of INF_G_INT_VAL for sensor 3 of second board using I2C;
- Starts the cycle where each of the signals is configured to be active high is this order: STRB, TRIG, INF and TAMP;
- For the first three signals, sets the signal state so only one is active at a time;
- If STRB is supposed to be on, waits for S2_INT to go off and checks if the color of the LEDs is blue;
- If STRB is supposed to be off, checks if the color of the LEDs is different from blue;
- If TRIG is supposed to be on, measures that the frequency on the counter is within the acceptable values;
- If TRIG is supposed to be off, measures that the frequency on the counter is at 0 within the acceptable values;
- If INF is supposed to be on, waits for S14_INT to go off and checks if the color of the LEDs is red and turns green if INF is turned off afterwards;
- If INF is supposed to be off, checks if the color of the LEDs is different from red;
- Checks that with any value on TAMP control, the frequency on the counter is at 0 within the acceptable values;
- After the first three signals are tested, TAMP is set as active;
- Asks the user to keep button I4 pressed;
- Measures that the frequency on the counter is at 0 within the acceptable values;
- Asks the user to release the button;

- Measures that the frequency on the counter is within the acceptable values;
- End.

Chapter 6

Solution Validation

This chapter is where the solution will be validated as a proper response to the problem discussed in chapter 1. This chapter will discuss how the solution was verified, which corrections were seen as necessary and which ones were corrected, as well as talk about the advantages and disadvantages of having a test development department in an electronic development and production company as well as how well the original plan went and what could be accomplished.

6.1 Verifying Test Module Functionality

In order to test the functionality of the Test Module as the new version of the RGB Siren using test points didn't arrive, a multicolored cable with 23 wires was soldered between the Test Module and the previous version of the UUT in order to perform the same electronic connections as the test probes on the new version. Figure 6.1 shows the finalized cable used during the validations and figure 6.2 shows the connections between the and the UUT and the BCB using the cable.

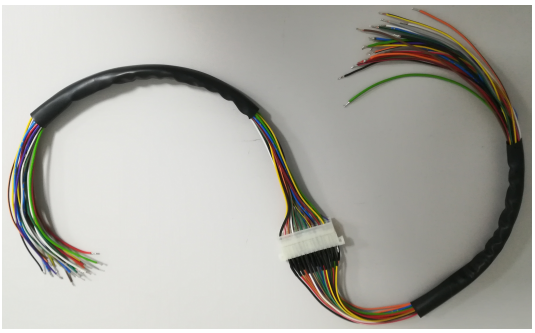


Figure 6.1: Cable used to perform the test probe connections

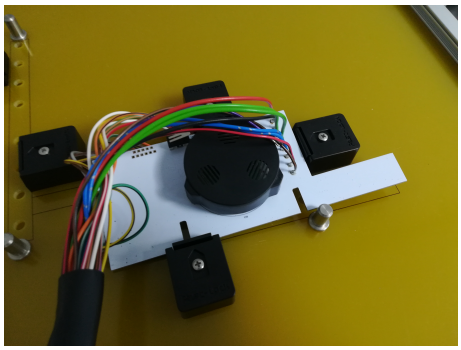
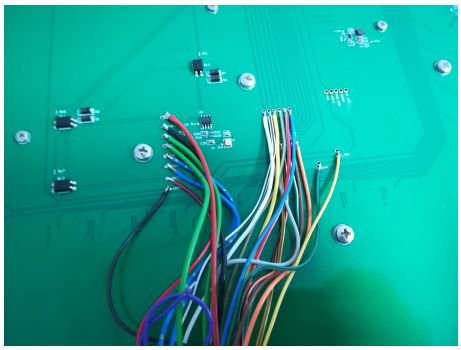



Figure 6.2: Connections between the UUT and Test Module

After that, manual validations were performed using the TEST-TRACK software to check if everything was working properly on the Test Module. This was done using the manual control option present in the software to control the power sources and the outputs to validate each of the hardware functionalities developed. Figure 6.3 shows the manual control window from TEST-TRACK.

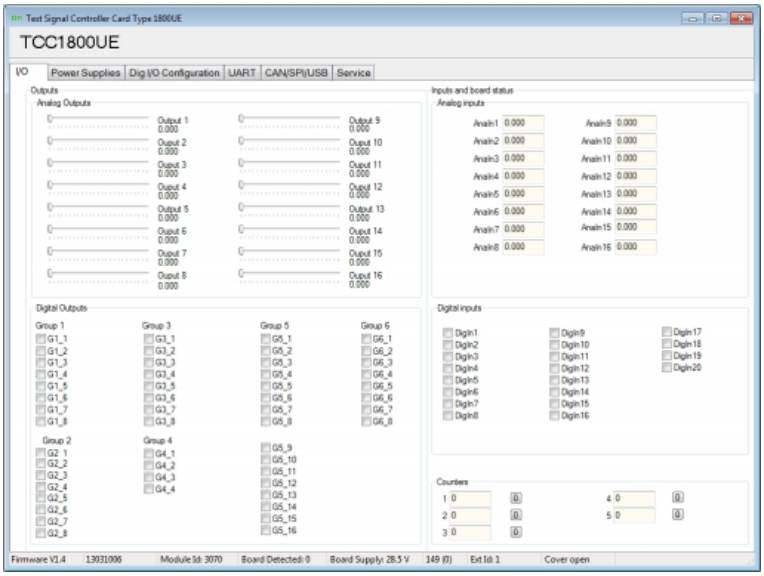


Figure 6.3: TEST-TRACK control window

Since the manual control window has no option to use I2C manually, a simple code was developed to run on the TEST-TRACK software to retrieve the values read from the individual sensors. Mainly based on the LED test code, the operations used are the following:

- Set both supplies on with 12 V and 0.5 A;
- Set the configuration of the signals to only activate high;
- Set the supply on and enable the adapter power;

- Initialize all sensors using I2C;
- Set experimental interrupt upper threshold value for all sensors using I2C;
- Retrieves color values from all LEDs;
- Asks if the user wants to repeat the retrieval of the LED colors;
- Ends if the user says no.

During the full process of verification, the following problems were identified:

1. **Problem:** The resistors that set the address of the BCB were inverted.

   Solution: *The resistors where soldered correctly;*

2. **Problem:** The 12V_ADC signal was not being read.

   Solution: *The wire solder broke from the PCB and was properly soldered and sealed with glue;*

3. **Problem:** The frequency was not being read.

   Solution: *The wire was incorrectly soldered, after correcting the connection to the cable the problem was solved;*

4. **Problem:** The I2C sensors were not responding.

   Solution: *The I2C communication from the ATE is malfunctioning from origin. This conclusion was confirmed by the engineer who previously developed the tests used in the other products from Tecnimaster. The solution to this problem is discussed in subsection 6.1.1;*

6.1.1 Solving the I2C problem

Since the ATE doesn't have functioning I2C, a new solution needed to be implemented on the test module to keep most of the BCB functionality intact and to not jeopardize the hardware that was already developed. As such, the chosen solution was to add to the module a small microcontroller kit with I2C capabilities that would use the least amount of pins from the ATE as possible, in order to keep most of the hardware developed unchanged. The microcontroller kit was programmed to output I2C communications using serial communication inputs from the ATE. The chosen microcontroller kit is the Seeeduino XIAO, a small and Arduino compatible microcontroller kit with a cost of 5.75 €that can easily be programmed and debugged using the Arduino Integrated Development Environment (IDE). Figure 6.4 shows the Seeeduino XIAO board along with the signal it has on each available pin.

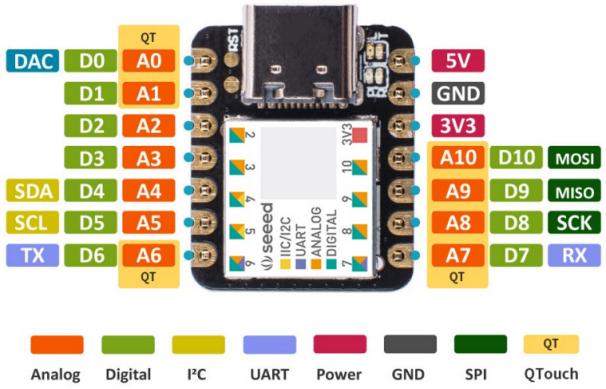


Figure 6.4: Seeeduino XIAO with each pin function

The way it connected to the rest of the circuit needed to be invasive, as some solder mask on the BCB tracks needed to be scratched off so the power and signal wires could be soldered. The Seeeduino needed to be connected to the existing SCL and SDA lines, the 3V3 and GND lines and the serial TX and RX pads on the BCB. Figure 6.5 shows the Seeeduino fixed to the BCB and figure 6.6 shows the schematic of the connections.

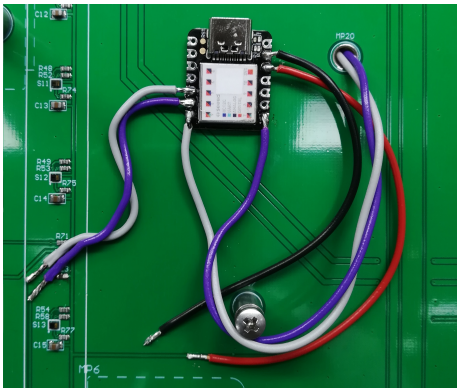


Figure 6.5: Seeeduino soldered to the BCB

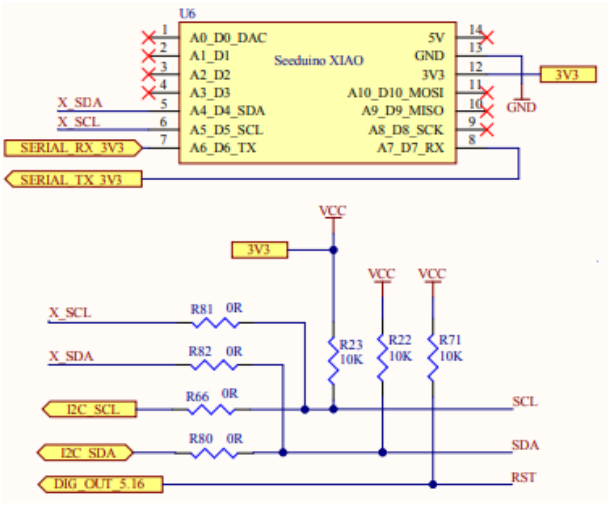


Figure 6.6: Seeeduino schematic on the BCB

After the board was connected, programming and validations were performed using Arduino IDE and code to communicate with the I2C switches and the RGB sensors. As the development was made, a serious problem was detected with the design of the BCB, caused by a bad interpretation of the RGB sensor data sheet: the PCB footprint was flipped. The data sheet indicated a mechanical design with the pinout of the component, but the recommended PCB pinout didn't indicate the pinout, the mistake was caused due to interpreting the mechanical design as equivalent to the footprint pinout. Figure 6.7 shows the data sheet drawings for the APDS-9253-001 sensor.

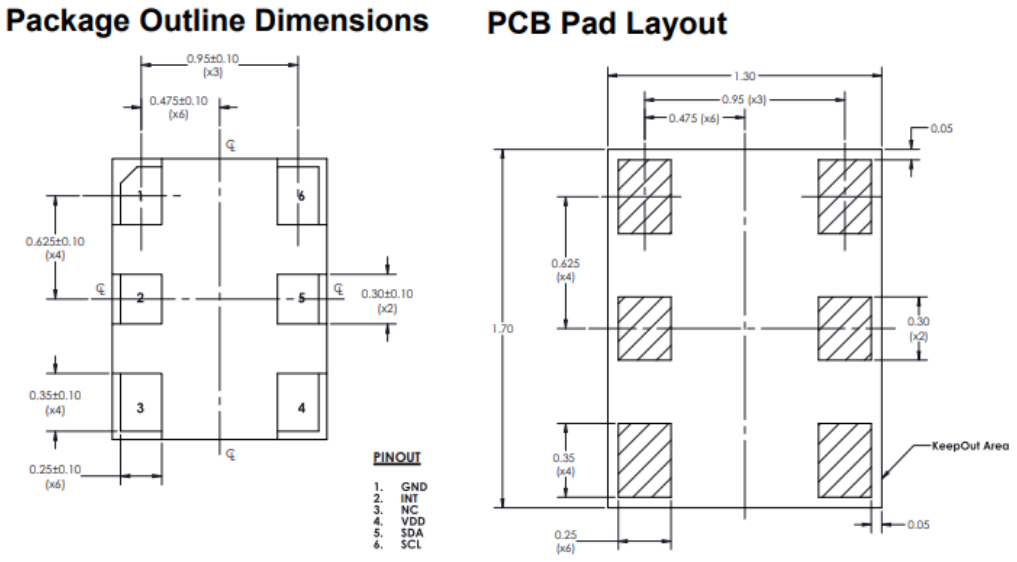


Figure 6.7: APDS-9253-001 data sheet mechanical drawing on the left and PCB footprint on the right

Along with this problem, another accident occurred as an employee of Tecnimaster accidentally pulled the Seeeduino board and partially destroyed both I2C signal lines, including one via. The repairs were quick, but the damage was another delay to the debug of the I2C functionality and to prevent further problems, the Seeeduino was glued to the board. Figures 6.8 and 6.9 show how the BCB was before and after the repair.

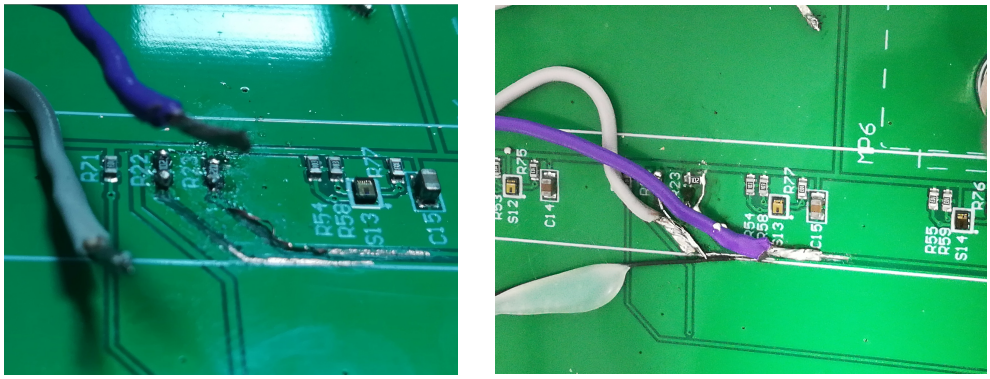


Figure 6.8: BCB damages on the left and the resulting repair on the right

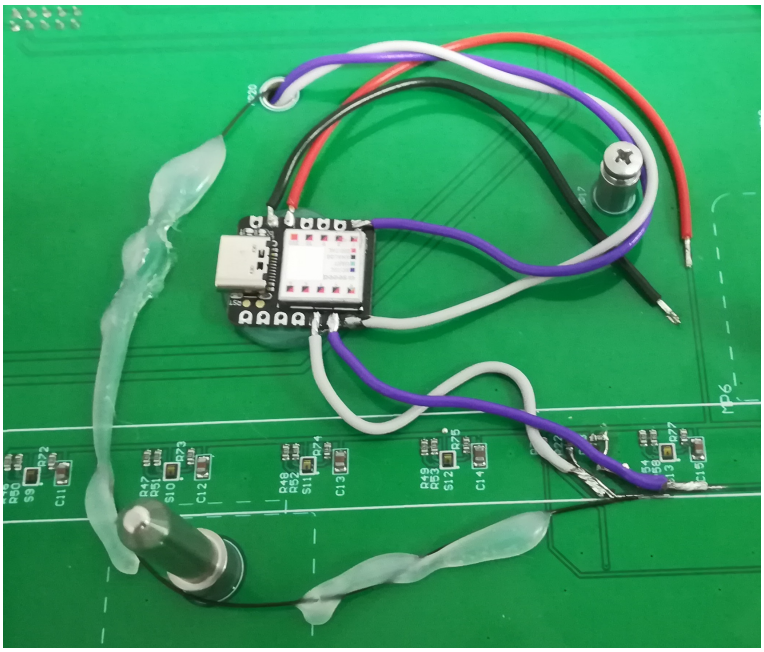


Figure 6.9: BCB after the repair

In order to solve the flipped component problem without ordering a new BCB, which would either require a delay of two weeks at a cost of roughly 100 €, or a delay of two days at a cost of 500 €, the decision taken was to dessolder the RGB sensors and wire the proper connections to them. However, this solution proved to be impossible, as the pads of the component were too small to solder

any wire securely. As such, different sensors were ordered, with the same core functionality (TCS34727). These took two days to arrive, at a cost 42.48 €, so proved to be the better solution. The soldering of the wires took three days to complete, with proper visual verification of each connection to ensure no short circuits. Figure 6.10 shows the sensors being soldered.

Figure 6.10: The new sensors being soldered

As the serial communication lines from the ATE use 5 V and the Seeeduino uses 3.3 V, a simple level translator needed to be assembled between the devices. This was conceived to use easily available materials and was done with high impedance inputs on both sides, as the ATE has current limitations on its transmission output. Figure 6.11 shows the circuit used.

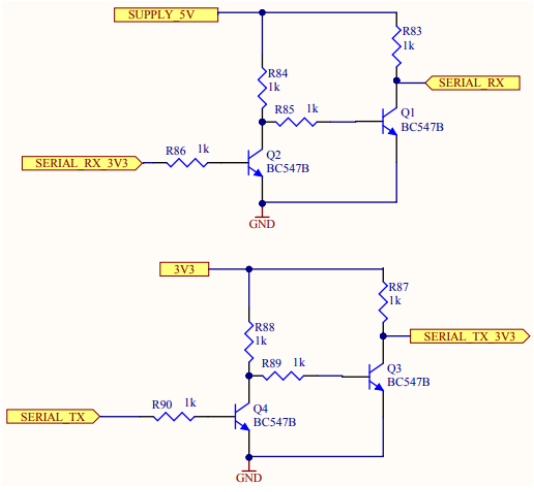


Figure 6.11: The level translator circuit

The circuit was assembled using only materials available at the factory and was conceived to be simple and bidirectional, assembled with a prototyping PCB and solder made tracks. The final result was validated and implemented on the BCB circuit. Figure 6.12 shows the BCB with all the new hardware implemented.

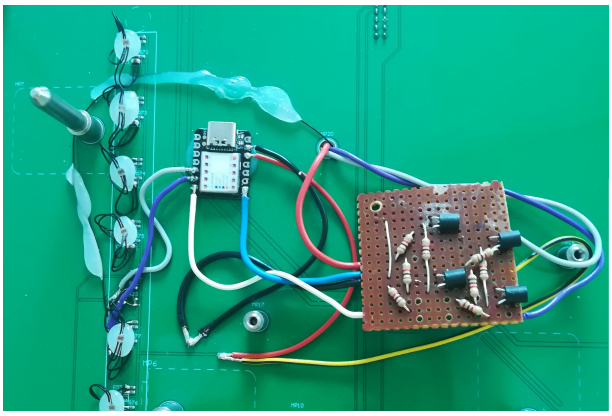


Figure 6.12: The BCB with all the added hardware

6.1.1.1 Firmware development

The code developed was done as a specific application for the hardware of the BCB in order to read data from each of the 16 individual TCS34727 as well as set the upper threshold interrupt values for every one of them. The code was tested and rules were set for it to be used properly by sending commands through the serial port using the characteristics on table 6.1. Figure 6.13 shows the flow chart of the program and the command format ans response is shonw on table 6.2.

Table 6.1: Seeeduino XIAO Serial to I2C control characteristics

Name	Value
Baudrate	9600
Parity	None
Data	8 bit
Stop Bits	1 bit
Flow Control	None

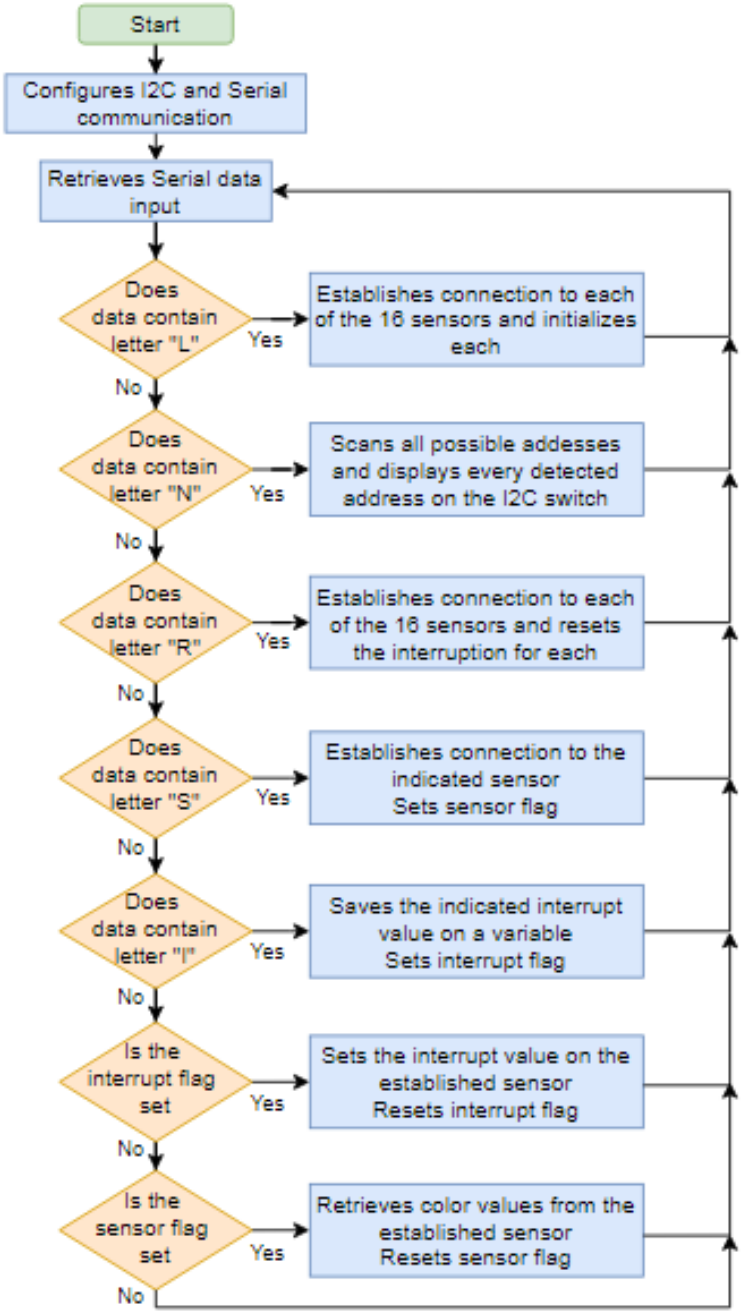


Figure 6.13: Flow chart of the Seeeduino program

Table 6.2: Seeeduino XIAO Serial to I2C control commands

Command	Function
L	Activates all the sensors;
N	Scans all the addresses for each switch and displays the results found;

Command	Function
R	Resets all the sensor interrupt pins (has no feedback);
S:<Sensor>	Reads values from sensor <Sensor>: <Sensor> has the format of two integer numbers from 0 to 8 representing the sensor position as such: XY; The X integer corresponds to the U4 I2C switch sensor and the Y integer to U5; If the integer is 0, no sensor is active and if it is 1 to 8 one of the eight sensors connected to the given switch is active;
S:<Sensor>;I:<Value>	Sets the upper interrupt threshold <Value> on <Sensor>: <Sensor> has the format of two integer numbers from 0 to 8 representing the sensor position, as in the previous command; <Value> is the hexadecimal value of the upper threshold to set from 0x0000 (0) to 0xFFFF (65535) and must always have four digits.
Example:	**S:05** - Activates and reads the values from sensor 5 of U5; **S:30;I:10E9** - Sets the upper interrupt threshold of 0x10E9 (4329) on sensor 3 of U4;

The firmware operations are the following:

- Retrieves serial data input;
- If data includes letter "L", all the sensors are initialized;
- If data includes letter "S", color and temperature data will be retrieved from the sensor indicated by the following 3 characters;
- If data includes letter "R", all sensor interrupts are reset;
- If data includes letter "N", a scanning process will be run, retrieving all detected I2C addresses connected to the I2C switches;
- If data includes letter "I", the upper interrupt value for the active sensor will be set, indicated by the following 5 characters;
- Back to start.

Initially tested using the virtual serial port from the Seeeduino, the code needed to be changed so the main commands would only output a single line of text and be as quick as possible while keeping most of the functionality for any future requirements to the test. The need for only one line of text came as the serial receive command from the ATE is expecting a carriage return and line feed

as the end of line for the information that gets processed. Figure 6.14 shows the feedback from the program during debug and figure 6.15 shows the feedback for the same commands on the final version of the code.

```
19:29:43.386 -> Started analysis
19:29:43.386 -> L
19:29:43.386 -> Activating every sensor...
19:29:43.386 ->
19:29:43.577 -> 1122334455667788
19:29:45.939 -> All set!
19:29:49.032 -> Started analysis
19:29:49.032 ->
19:29:49.077 -> N
19:29:49.077 -> Running scan mode...
19:29:49.077 ->
19:29:49.077 -> TCA Port #1 - Found I2C 0x29
19:29:49.077 -> TCA Port #2 - Found I2C 0x29
19:29:49.077 -> TCA Port #3 - Found I2C 0x29
19:29:49.125 -> TCA Port #4 - Found I2C 0x29
19:29:49.125 -> TCA Port #5 - Found I2C 0x29
19:29:49.125 -> TCA Port #6 - Found I2C 0x29
19:29:49.173 -> TCA Port #7 - Found I2C 0x29
19:29:49.173 -> TCA Port #8 - Found I2C 0x29
19:29:49.173 ->
19:29:49.173 -> TCB Port #1 - Found I2C 0x29
19:29:49.221 -> TCB Port #2 - Found I2C 0x29
19:29:49.221 -> TCB Port #3 - Found I2C 0x29
19:29:49.221 -> TCB Port #4 - Found I2C 0x29
19:29:49.221 -> TCB Port #5 - Found I2C 0x29
19:29:49.269 -> TCB Port #6 - Found I2C 0x29
19:29:49.269 -> TCB Port #7 - Found I2C 0x29
19:29:49.269 -> TCB Port #8 - Found I2C 0x29
19:29:49.316 ->
19:29:49.316 -> Done!
19:29:54.016 -> Started analysis
19:29:54.016 ->
19:29:54.016 -> S:10
19:29:54.016 -> Reading sensor value...
19:29:54.016 ->
19:29:54.016 -> U4:1 U5:0 Color Temp: 6692 K - Lux: 226 - R: 212; G: 324; B: 295; C: 866;
19:29:54.063 -> OK
19:30:09.736 -> Started analysis
19:30:09.782 ->
19:30:09.782 -> S:03;I:00FE
19:30:09.782 -> Reading sensor value...
19:30:09.782 ->
19:30:09.782 -> U4:0 U5:3 Setting sensor interrupt upper threshold...
19:30:09.782 ->
19:30:09.782 ->  I:254
19:30:09.829 -> OK
```

Figure 6.14: The program feedback during debug

```
17:26:13.965 -> 1122334455667788
17:26:36.097 -> Running scan mode...
17:26:36.097 ->
17:26:36.097 -> TCA Port #1 - Found I2C 0x29
17:26:36.144 -> TCA Port #2 - Found I2C 0x29
17:26:36.144 -> TCA Port #3 - Found I2C 0x29
17:26:36.144 -> TCA Port #4 - Found I2C 0x29
17:26:36.191 -> TCA Port #5 - Found I2C 0x29
17:26:36.191 -> TCA Port #6 - Found I2C 0x29
17:26:36.191 -> TCA Port #7 - Found I2C 0x29
17:26:36.239 -> TCA Port #8 - Found I2C 0x29
17:26:36.239 ->
17:26:36.239 -> TCB Port #1 - Found I2C 0x29
17:26:36.239 -> TCB Port #2 - Found I2C 0x29
17:26:36.239 -> TCB Port #3 - Found I2C 0x29
17:26:36.286 -> TCB Port #4 - Found I2C 0x29
17:26:36.286 -> TCB Port #5 - Found I2C 0x29
17:26:36.286 -> TCB Port #6 - Found I2C 0x29
17:26:36.334 -> TCB Port #7 - Found I2C 0x29
17:26:36.334 -> TCB Port #8 - Found I2C 0x29
17:26:36.334 ->
17:26:36.334 -> Done!
17:26:46.157 -> U4:1 U5:0 Color Temp: 6707 K - Lux: 47 - R: 43; G: 67; B: 60; C: 177;
17:27:08.801 -> U4:0 U5:3 OK
```

Figure 6.15: The program feedback as the final version

6.1.1.2 TCS34727 - RGB light sensor

The TCS3472 from AMS is a miniature I2C digital RGB sensor with four individual channels of red, green, blue and clear (unfiltered) light detectors. The component has a IR filter to minimize the influence of that spectral component on the measurements performed. After being powered on, the component starts up in a low power state that needs to be disabled using the proper I2C configuration. It has programmable interrupt functions using upper and lower thresholds and also persistence functions that can be configured to allow for more immediate sensing capabilities. Figure 6.16 shows the functional diagram of the component and table 6.3 shows the functionality associated to each pin.

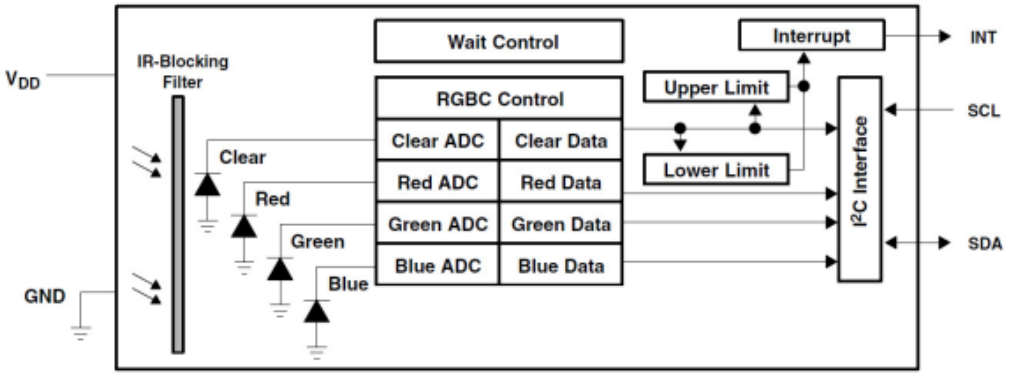


Figure 6.16: TCS3472 functional diagram

Table 6.3: TCS3472 pin functions

Name	Pin	Type	Description
VDD	1	Power	Power supply voltage
SCL	2	In	I2C clock input
GND	3	Power	Power supply ground
NC	4		Not Connected
INT	5	Out	Open drain interrupt
SDA	6	In/Out	I2C serial data

As the sensor will be used to detect RGB light, the upper threshold interruptions will be configured in order to detect the intensity of the light during the various tests. The upper threshold works as the APDS-9253-001: by setting a minimum value of light intensity, which triggers the interruption. In order to use the sensor for the various functions it is going to perform during the test, the data sheet was consulted and the I2C addresses on table 6.4 were retrieved.

Table 6.4: Addresses used in the I2C communications

Name	Address	B7	B6	B5	B4	B3	B2	B1	B0
ENABLE	0x00	0	0	0	AIEN	WEN	0	AEN	PON
ATIME	0x01	Integration time value							
AIHTL	0x06	Upper threshold lower byte							
AIHTH	0x07	Upper threshold higher byte							
PERS	0x0C	0	0	0	0	Interrupt persistence			
STATUS	0x13	0	0	0	AINT	0	0	0	AVALID
CDATAL	0x14	Clear data lower byte							
CDATAH	0x15	Clear data higher byte							
RDATAL	0x16	Red data lower byte							
RDATAH	0x17	Red data higher byte							
GDATAL	0x18	Green data lower byte							

Name	Address	B7 B6 B5 B4 B3 B2 B1 B0
GDATAH	0x19	Green data higher byte
BDATAL	0x1A	Blue data lower byte
BDATAH	0x1B	Blue data higher byte

CDATAx, RDATAGx, GDATAx and BDATAx are the registers corresponding to the Clear, Red, Green and Blue sensors on the device. They store the color component read by the sensor until it receives a I2C command that reads that value. These registers are divided in two as the value detected is converted to two bytes that are read using I2C. Likewise, the AIHTx registers also hold two bytes corresponding to the interrupt upper threshold configuration to use by the device. ATIME holds a byte that sets the integration time of the sensor, that is, the time it takes to retrieve values from the sensor ADCs, the smaller the value, the faster the sensor reads, but the less precise it becomes. Each integration cycle is 2.4 ms and reads a maximum value of 1024 from the ADC, so the value set on the register must be calculated to read the best possible value in the least amount of time, since the maximum precision of the sensor is 65535. PERS holds a 4 bit word that sets how many times the clear sensor value needs to be beyond the interrupt threshold for an interrupt to be generated by the sensor. ENABLE holds the sensor functionality configuration, which is detailed in table 6.5 and STATUS holds sensor status information, which is detailed in table 6.6.[18]

Table 6.5: ENABLE register bits

Name	Bit	Description
AIEN	B4	Sensor interrupt enable
WEN	B3	Wait enable
AEN	B1	RGB + Clear sensor ADCs enable
PON	B0	Power on. Takes the sensor out of low power mode.

Table 6.6: STATUS register bits

Name	Bit	Description
AINT	B4	Indicates the presence of an interrupt
AVALID	B0	Indicates a completed integration cycle

6.2 Full Test Validation

The full test validation was performed with the connecting cable still soldered between the Test Module and the UUT and by running the initial versions of the code multiple times and solving every problem that became apparent as the test went on.

During the validation, the following problems were identified on the code and corrected:

1. 12V_ADC had different values from the initially calculated ones;
2. The CHECK_BAT signal needs a time before it has the correct output value because the capacitors are connected in parallel with the V_{IN} supply and need a time to charge completely before the battery output is valid and this time is never reached if the board is powered at less than +12 V;
3. The light intensity, color values and interrupt threshold values needed to be obtained experimentally as no previous values were available to base the code upon;
4. A hierarchy of signals became apparent, where in the case all three signals are active, the priority goes as follows: INF, STRB and TRIG. So in order to test TRIG, for instance, the other two need to be deactivated;
5. The frequency counter keeps its value after each measure;
6. The order of the tests needs to be changed in the LED and Control Signal blocks in order to optimize the tests.

After the developments and the corrections were performed, the following list of changes was enacted:

1. The values were retrieved experimentally and added to the code;
2. The test at that point was changed to power the board using +12 V and to have a timeout after which the test fails if no voltage is measured;
3. The values were retrieved experimentally and added to the code;
4. When the lower hierarchy signals need to be tested, the code activates only that signal and keeps the other two inactive;
5. A command to reset the counter exists but doesn't work in the compiler, so every time a new count operation need to be made, the counter is set again and a small delay is performed before the test;
6. The tests were changed to become as optimized as possible.

The test was validated against the main test blocks from table 4.6, so after the performance of each test was confirmed and all test functions were verified as accurate, the test was approved to be shown and demonstrated to the engineering team at Tecnimaster. After the test was shown to the companies' engineering team and all the test blocks were demonstrated and the test configuration values were explained, the test was approved for use in production.

6.3 Proof of the capacity of test

As a test needs to be capable of verifying the proper function of a UUT, a validation that ensures that capacity needs to be performed on a regular fashion as the test system is being used. A good time interval between verifications is one year for a regular product, however it is up to the companies quality and engineering department to decide how much time should pass for a verification to be done to the test. In this particular case, a verification needs to check each of the test steps detailed in table 4.6. This verification is done by altering the assembled PCB and purposefully create defects on the assembly of the components, by either taking them out of the circuit or creating short circuits in strategic places that should be detected by the test. This final step of the developed project was not possible to fulfill due to time constraints as the samples were limited to two and the invasive nature of this process would be time consuming, so the process will be simplified and the company will use a standard assembled board to perform those validations.

6.4 Cost and Time Analysis

The initial test for the product was a manual one, which programmed the firmware and validated only a few of the total functionalities, using a single configuration for the signal behavior, LED behavior and color and a small cable with three contacts that allowed to power the board using SUP and stimulated TRIG using a button to check if the buzzer rang. This test is faster, but requires the operator to personally check the behavior, and the production had no control over the results of the test. A manual test of such simplicity also has nearly no coverage (about 1/3 of the new test coverage) and serves as no way to check the proper assembly of all the board components. The little time and cost investment from this test may result in greater expenses in the quality of the product, so a more extensive and automatic test becomes a necessary investment if the product is to be widely available and functional for the general public as is the companies' intention. In order to prove that, the following tables were compiled with relevant data. Table 6.7 shows the relevant costs related to the normal functioning of the company, including the production of the XTEC RGB and the cost of labor from both the production worker and the engineering department. Table 6.8 shows the costs related to the development of the new test for the product, including all the development and the cost of the TEST-OK ATE. Finally table 6.9 compares both tests side by side to understand what changed between both of them.

Table 6.7: Company related costs

Description	Value
XTEC RGB production cost	11.85 €
XTEC RGB manual production time	10 h
Production work per hour	4.45 €
Engineering work per hour	9.94 €

Table 6.8: Full new test costs

Description	Value
TEST-OK ATE	25 000.00 €
TEST-OK Test Module PCB	295.17 €
TEST-OK Test Module Materials	581.25 €
TEST-OK Test Module I2C Solution	48.24 €
Time spent in development	430 h
Development cost	4 274.20 €
Total Costs	30 198.86 €

Table 6.9: Old vs new test

Metric	Old	New
Firmware download time	1 min	1 min
Functional test time	30 s	2 min
First Set-up time	10 min	1 min
Following Set-up time	30 s	20 s
Total time per test	2 min s	3 min 20 s
Cost per test	0.15 €	0.25 €
Percentage of coverage	33 %	100 %

Considering the old test only covers one third of the total hardware, the remaining two thirds will be in an unknowable state, which may result in the manual rework of the board. In the worst case scenario, two thirds of the board may be faulty and may need manual rework, in that case it will take 6 h 30 min for that repair to be made. Considering the value of production work per hour, this would result in a 28.925 €cost for each board. Given this data and the total cost of a full test, if more than 1045 boards are produced the test pays itself. If the ATE is already bought, 180 boards are enough to justify a proper test. This scenario is a direct proof on the advantage that comes from developing a proper test for the board.

6.5 Impact of the test in production

The test of the board with the ATE proved to be an improvement over the previous strategy, as a more complete solution and a more appropriate method

of detecting assembly errors. It became the more usable solution for its simplicity of assembly and test set-up, which is a great advantage of the TEST-OK ATE model. The time and cost of the solution is the greater downside to it, however it comes also as a consequence of a lack of proper planning for production. In this study, some issues could be solved if the test was planned as the hardware of the board was developed, such as the test point placement and the problems related to the absent ATE functionalities could have been detected earlier before implementation. To do so, a proper test engineering team should be formed with the necessary knowledge to not only develop proper Test Modules for the ATE but also retain the knowledge related to the limitations of the test system and validate it from time to time, ensuring its proper function. A test engineering team would also allow for more production oriented development of hardware, as the functional blocks could be designed with the test in mind instead of only functionality.

6.6 Software Used

Diverse programs were used in the development of this work, both in the development of the documentation and the development of the solution.

The document itself was developed using LaTeX and online resources to complement and illustrate the work: **overleaf.com** was used to write, code and format the document, **draw.io** was used to draw the diagrams, circuits and flow charts and **officetimeline.com** was used to build the Gantt charts.

Other software used in the development of the solution itself were:

- **Altium**, to draw the Test Module PCB and schematic, any additional component that was necessary to include in the project and also export the BoM;
- **falstad.com/circuit/**, to simulate simple circuits and concepts as well as confirm calculations before assembly;
- **snapeda.com**, to retrieve component libraries, footprints and 3D models;
- **Office Excel**, to organize the BoM and create quick reference tables with the connections of the Test Module;
- **PSPad** and **TEST-TRACK**, to write the code, validate, run it and facilitate its development and edition;
- **Arduino IDE**, to develop, debug and flash the Seeeduino firmware;
- **Ditto**, to help with the edition of the code and other tasks related to copying text;

- **GIMP** and **Paint**, to edit all images and make them more clear to use in the document;
- **trello.com**, to list tasks related to the project and to the document them in an easy-to-access way;

Table 6.10 displays the estimated time spent on each of the listed software used on the project.

Table 6.10: Estimated hours spent using each software

Program Name	Hours Spent
Ditto	Undetermined
overleaf.com	208
Altium	166
Arduino IDE	60
PSPad	56
TEST-TRACK	31
draw.io	30
Office Excel	20
falstad.com/circuit/	15
snapeda.com	10
Paint	10
trello.com	8
GIMP	4
officetimeline.com	4

6.7 Plan

The plan defined at the beginning of the project was delayed greatly. The reasons for the delays are related mainly to the development of the test module hardware and the solution to the lack of proper I2C communication on the ATE. From the original tasks, two were not possible to be done, as there were no conditions to perform them. The task number 4 was not performed because access was denied to see the schematics of the test modules used in other tests by the company that developed them. The absence of this task didn't impact much, as the test module template from TEST-OK was detailed enough to help understand how a system should be designed. The task number 5 was not performed because the company didn't possess a validation or debugging test module to use in a simple application. The absence of this task impacted in understanding how the system worked and what functionalities were actually working properly. If task number 5 was possible to do as was planned, the time spent on developing an alternative solution to I2C communication would be accounted

for and the BCB would have been changed beforehand, saving time on development and debugging.

Task 10.5 was added during the development of the project as a hardware error was found by the company during validation of the product and since the PCB was already being changed to improve testability, that correction was not passed to the company that designed the original PCB. Task 13.5 was also added during development to solve the ATE I2C compatibility problem. Table 6.11 shows all the tasks performed during the development of the project, along with the time each took and whether it was present on the initial plan or not. Figure shows the Gantt chart for the tasks performed, and figure shows the initial tasks Gantt chart in comparison with the final tasks Gantt chart.

Table 6.11: All the tasks performed

Name	Real Duration	Comparison to planned
1 - TEST-OK Hardware study	3 days	Less time
2 - TEST-OK Software and programming language study	5 days	As planned
3 - Check how the code of a test is developed	2 days	Less time
4 - Check HW of a test and how to develop	0 days	Not possible to do
5 - Develop a small application to see how the ATE works	0 days	Not possible to do
6 - Develop the test logic	8 days	As planned
7 - Learn how to use Altium	**27 days**	**More time**
8 - Develop the hardware for the full product test	**34 days**	**More time**
9 - Correct design and study BOM and PCB prices	**13 days**	**More time**
10 - Change XTEC RGB Siren to improve testability	3 days	As planned
10.5 - Change original design of XTEC RGB to correct design mistake	***6 days***	***Added afterward***
11 - Develop the software for the full product test	**18 days**	**More time**
12 - Assemble test module hardware	7 days	Less time
13 - Validate the final test solution	**34 days**	**More time**
13.5 - Research, development and assembly of I2C solution	***39 days***	***Added afterward***
Total time	**199 days**	

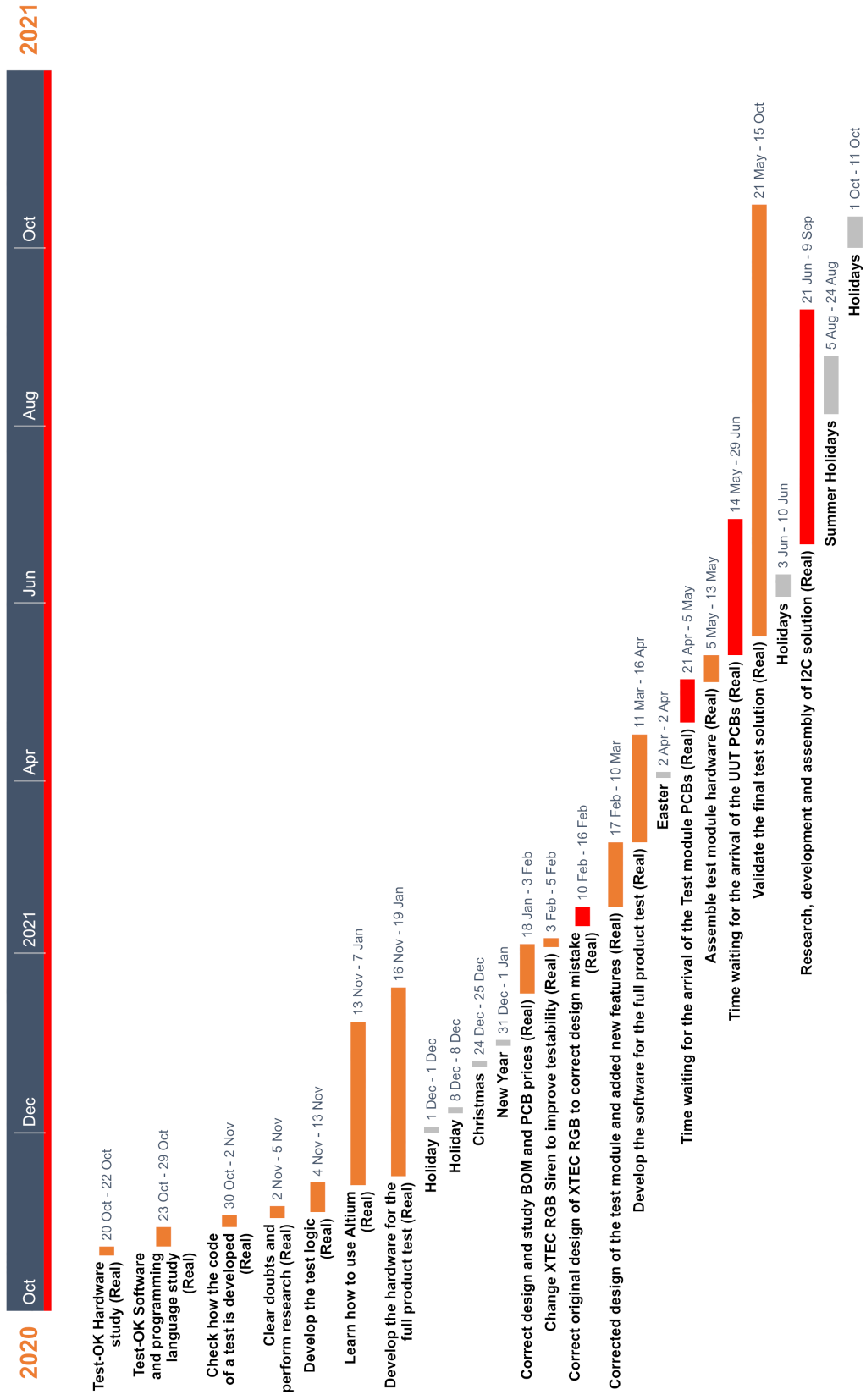


Figure 6.17: Real task duration Gantt chart

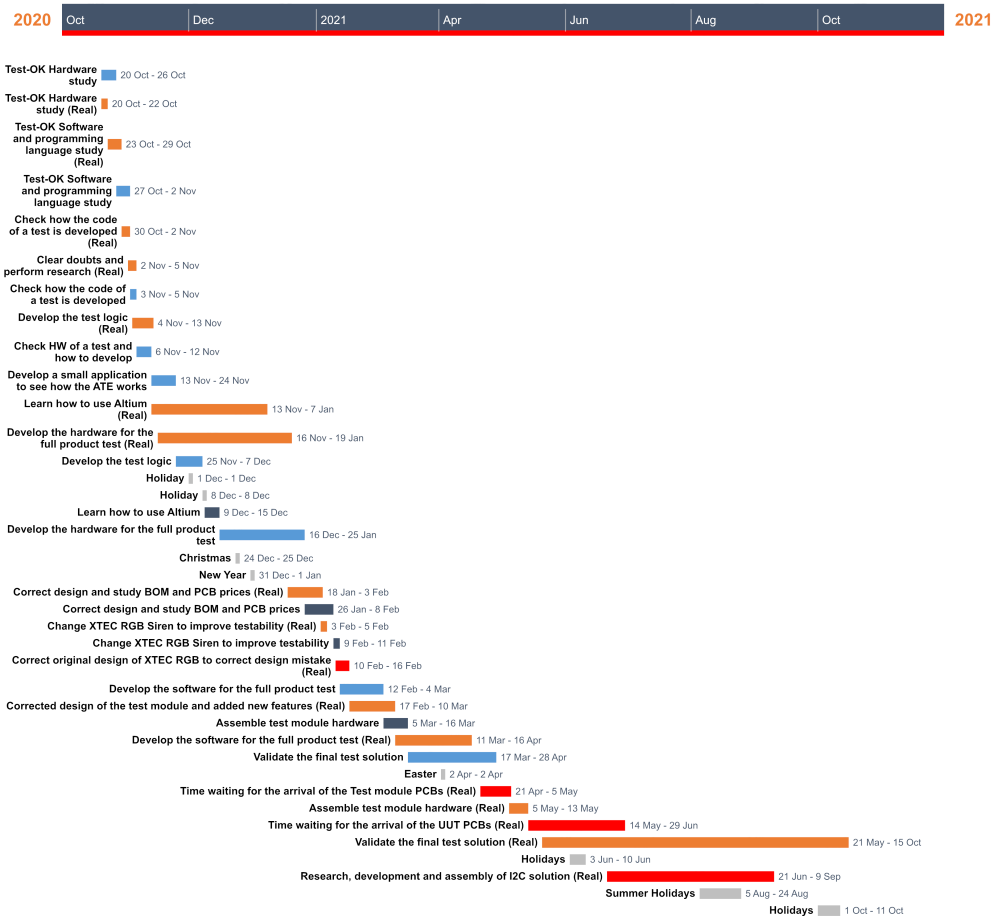


Figure 6.18: Real and original Gantt charts superimposed

Chapter 7

Conclusion

In this final chapter we will discuss the conclusions of the project and mention the possible future work that can be done based upon this project.

7.1 Conclusions

The project proves the real need for an electronics test department in a hardware developing company by taking the example of a company that didn't have such a thing implemented and developing the full test to verify how advantageous it is. The company *Tecnimaster FL - Sistemas Electrónicos, Lda* agreed to have one of their products analyzed and a full automatic test conceived for it. The test involved the development of hardware and software interfaces with a TEST-OK ATE and also strategies to test light color and intensity, analogical electrical values and the sound produced by the PCBs of a siren. It took about 430 hours to conclude with about 1500 lines of code written and multiple hardware and software versions developed before the final products were delivered. The requirements and the plan set at the start were mostly accomplished, and the solution was approved for use in production after being shown to the engineering department. Lastly, the analysis made to the cost of development along with the worst-case scenario of every board having errors in their unchecked hardware, shows how advantageous a test really is and paints the clear picture that investment in test will have a clear return in profit for the company that develops it.

The project taught me valuable lessons in hardware and test development and how challenging a product development cycle can be especially when the integration with an existing equipment is mandatory, as the most time-consuming solution came about due to the TEST-OK I2C limitation. It taught me more about

test development and technologies and gave me a further taste of the complexities and excitement of hardware development, with a further understanding that an electronics project cannot come about without planning the test along with the development of the hardware itself.

7.2 Future Work

Future work involves developing a test module that can be used to validate all the ATE functionalities as well as implementing a proper database for the tests of the XTEC RGB along with every other test that may not have those mechanisms properly implemented. The development of a test module to validate the ATE will be useful for any test developer as well as to test if all the functionalities of the ATE are functioning properly and what to expect facing certain test cases. The database functionality was not implemented due to time constraints and became the responsibility of the company to take care of that part of the project, as it has to do with its internal logistics.

References

[1] "http://www.labdegaragem.com/", Oct. 2017. [Quoted on p. v, 12]

[2] "http://www.electronics-notes.com/", Oct. 2017. [Quoted on p. v, 13, 30]

[3] "https://www.circuitbasics.com/", Dec. 2020. [Quoted on p. v, 13]

[4] "http://www.autocorerobotica.blog.br/", Dec. 2020. [Quoted on p. v, 14]

[5] "https://www.electronics-tutorial.net/", Feb. 2021. [Quoted on p. v, 19, 20, 21, 22, 23, 24, 25]

[6] "https://www.tempoautomation.com/", July 2021. [Quoted on p. v, 30]

[7] TEST-OK, "Tcc1800-ue hardware reference manual," 2016. [Quoted on p. vi, 40, 43]

[8] M. Felgueiras, "Apoio à depuração e teste de circuitos mistos compatíveis com a norma ieee1149.4," 2008. [Quoted on p. 8, 10, 11]

[9] M. Felgueiras, G. Alves, and J. M. Ferreira, "Debugging mixed-signal circuits via the ieee1149.4 std. – analysis of limitations and requirements," 2006. [Quoted on p. 12]

[10] N. Semiconductors, "I2c specification and user manual, rev 6," 2014. [Quoted on p. 14]

[11] "https://www.baldengineer.com/", Aug. 2021. [Quoted on p. 17]

[12] A. L. Spada, "Attack brasil - microfones, parte 1," 2010. [Quoted on p. 26]

[13] J. Lewis, "Understanding microphone sensitivity," 2012. [Quoted on p. 26]

[14] "https://www.mems-exchange.org/", June 2021. [Quoted on p. 27]

[15] J. Chen, R. Reilly, and G. Lynn, "The impacts of speed-to-market on new product success," *Engineering Management*, 2005. [Quoted on p. 30]

[16] TEST-OK, "Test language description," 2016. [Quoted on p. 43]

[17] Broadcom, "Apds-9253-001, digital rgb sensor, data sheet," 2019. [Quoted on p. 72]

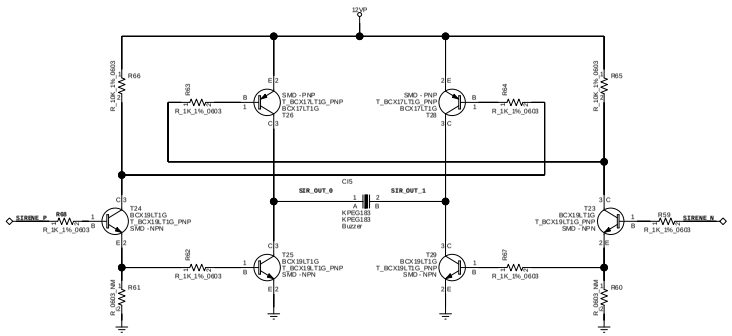
[18] AMS, "Tcs3472 - general description," 2020. [Quoted on p. 116]

Appendix A

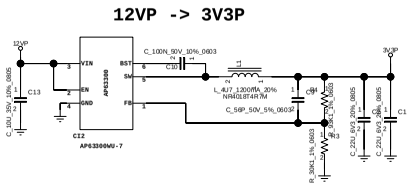
Appendix - Schematics

A.1 XTEC RGB Schematic

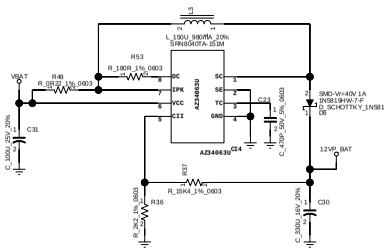
SIRENE



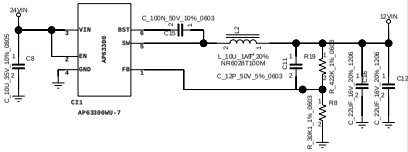
12VP -> 3V3P



MONTAR SE BAT = 7.2V

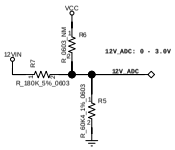


MONTAR SE VIN = 24V

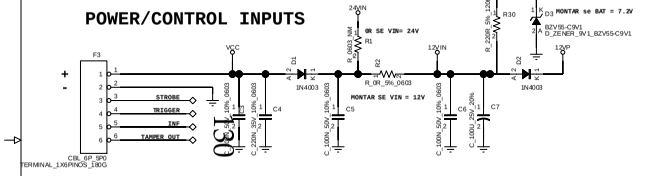


Battery

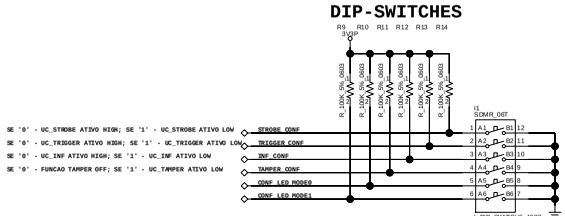
12VP ADCin

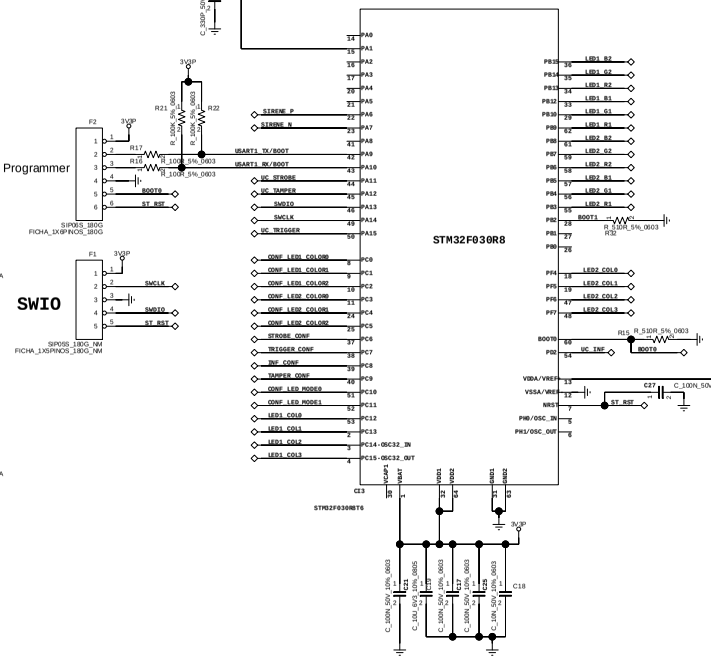


POWER/CONTROL INPUTS

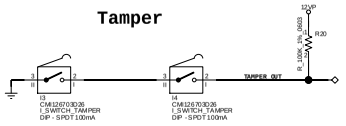


DIP-SWITCHES





Tamper



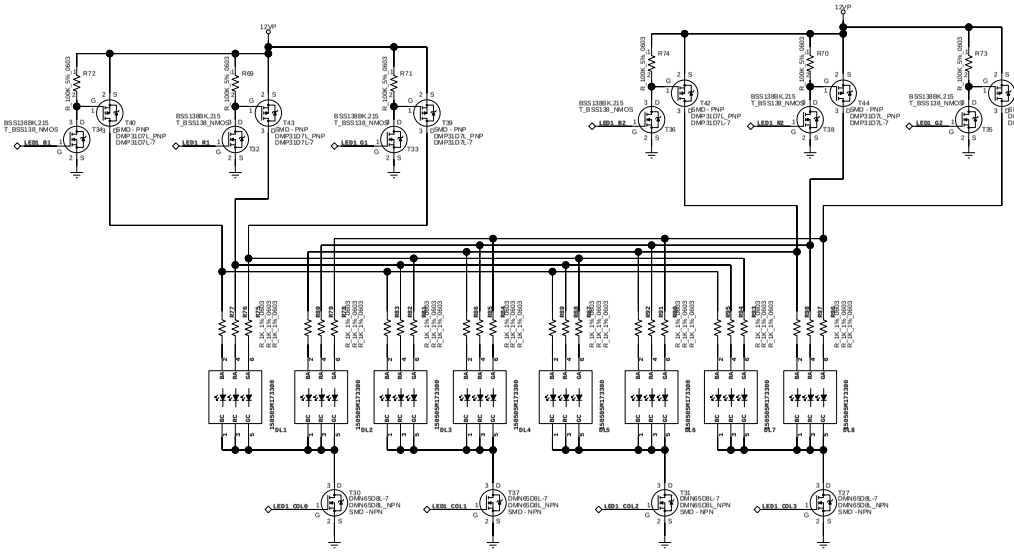
withUS telecommunication developments

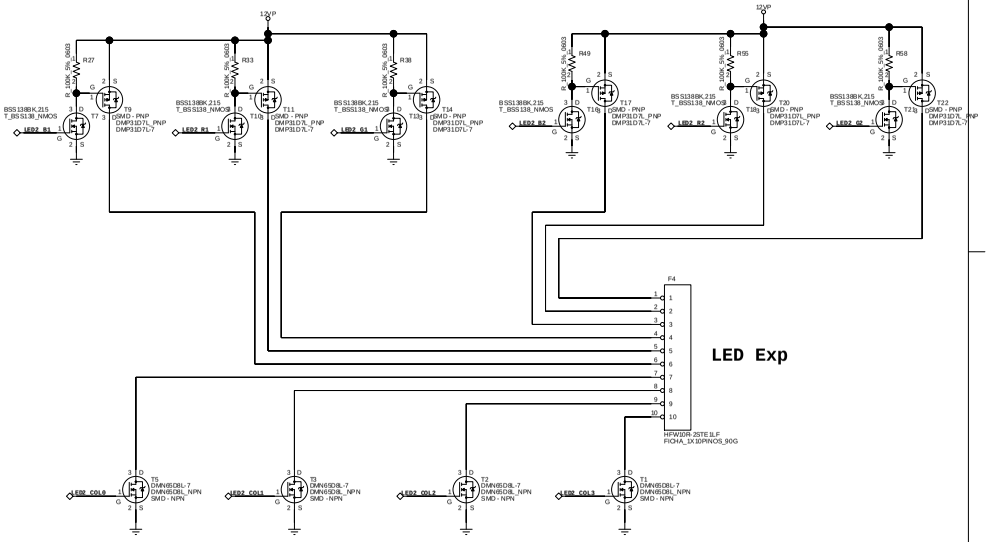
W20A001V3R0-Sirene

PROJ N: W20A001V3R0-Sirene	REV: v2	RESP: Vitor Silva
DATE: 2020/05	PAGE: 1 OF: 2	

H=8ND; V=4LA

8x RGB LEDs





131

DL1

Red ON: LED1_R1='1' + LED1_COL0='1'
Green ON: LED1_G1='1' + LED1_COL0='1'
Blue ON: LED1_B1='1' + LED1_COL0='1'

DL2

Red ON: LED1_R2='1' + LED1_COL0='1'
Green ON: LED1_G2='1' + LED1_COL0='1'
Blue ON: LED1_B2='1' + LED1_COL0='1'

DL3

Red ON: LED1_R1='1' + LED1_COL1='1'
Green ON: LED1_G1='1' + LED1_COL1='1'
Blue ON: LED1_B1='1' + LED1_COL1='1'

DL4

Red ON: LED1_R2='1' + LED1_COL1='1'
Green ON: LED1_G2='1' + LED1_COL1='1'
Blue ON: LED1_B2='1' + LED1_COL1='1'

DL5

Red ON: LED1_R1='1' + LED1_COL2='1'
Green ON: LED1_G1='1' + LED1_COL2='1'
Blue ON: LED1_B1='1' + LED1_COL2='1'

DL6

Red ON: LED1_R2='1' + LED1_COL2='1'
Green ON: LED1_G2='1' + LED1_COL2='1'
Blue ON: LED1_B2='1' + LED1_COL2='1'

DL7

Red ON: LED1_R1='1' + LED1_COL3='1'
Green ON: LED1_G1='1' + LED1_COL3='1'
Blue ON: LED1_B1='1' + LED1_COL3='1'

DL8

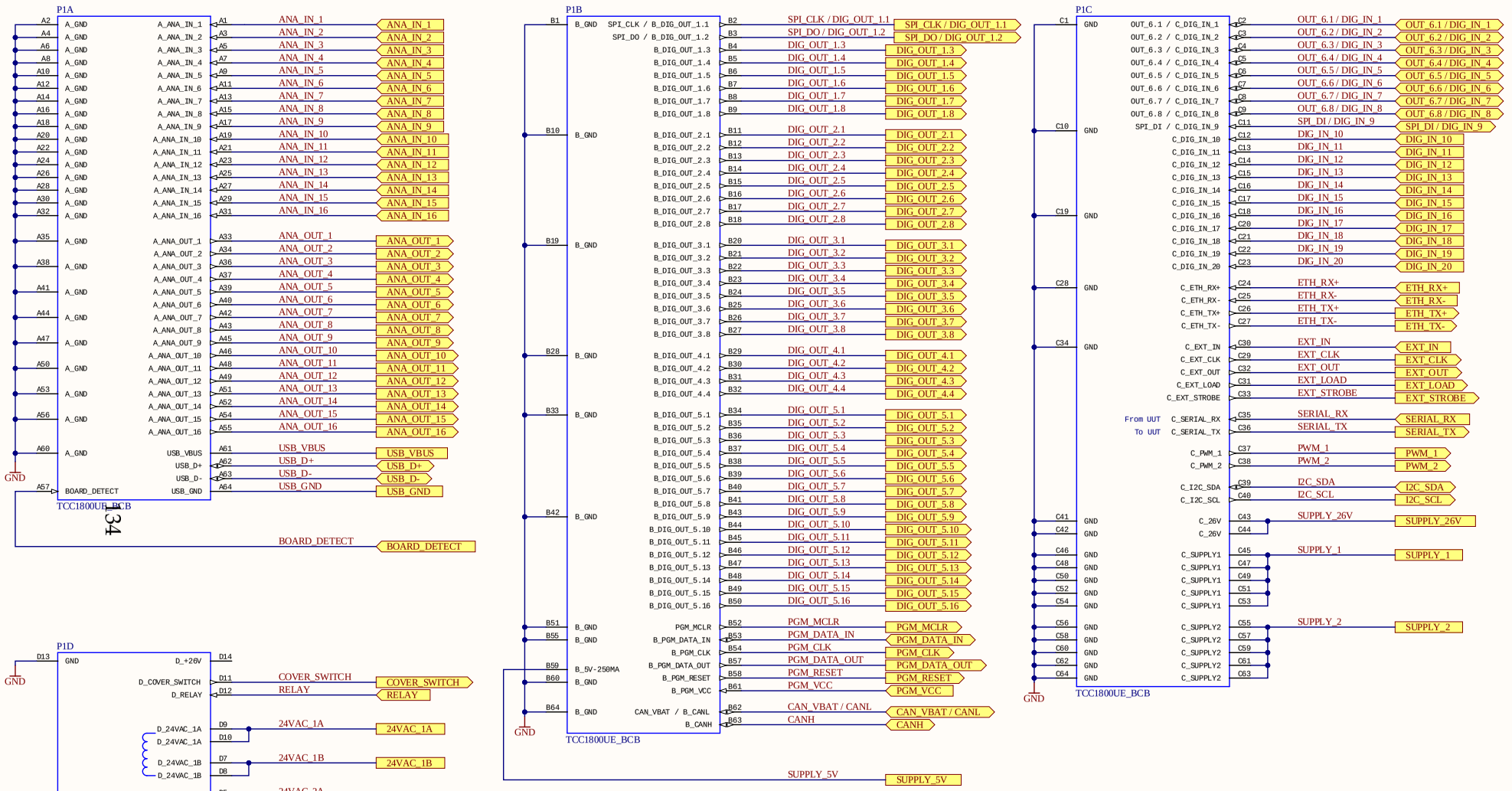
Red ON: LED1_R2='1' + LED1_COL3='1'
Green ON: LED1_G2='1' + LED1_COL3='1'
Blue ON: LED1_B2='1' + LED1_COL3='1'

withus telecommunication developments

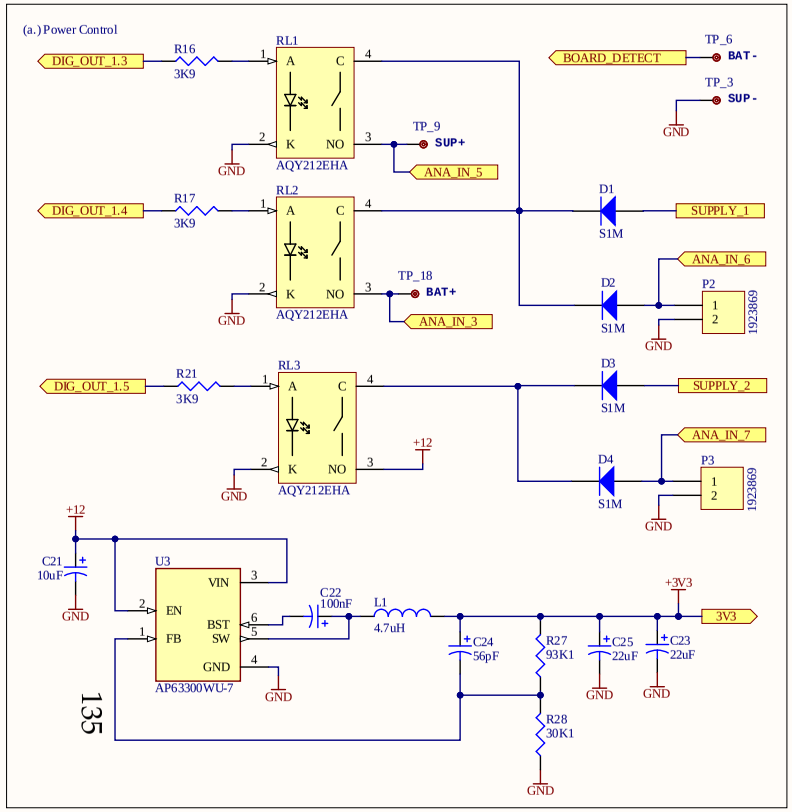
W20A001V3R0-Sirene

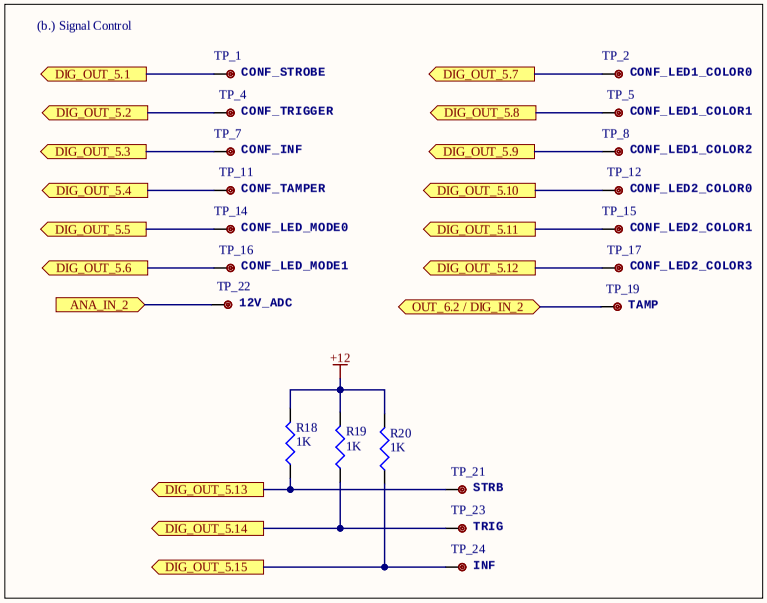
PROJ N: W20A001V3R0-Sirene	REV: v2	RESP: Vitor Silva
DATE: 2020/05	PAGE: 2 OF: 2	

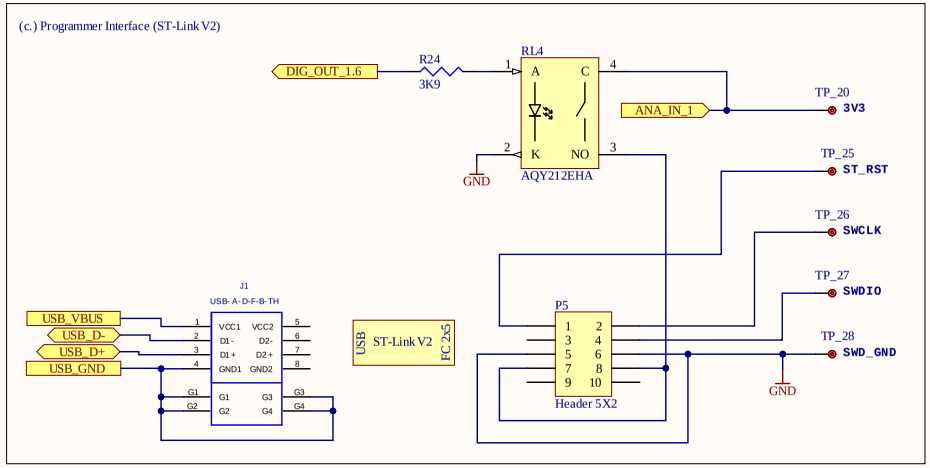
A.2 Test Module Schematic

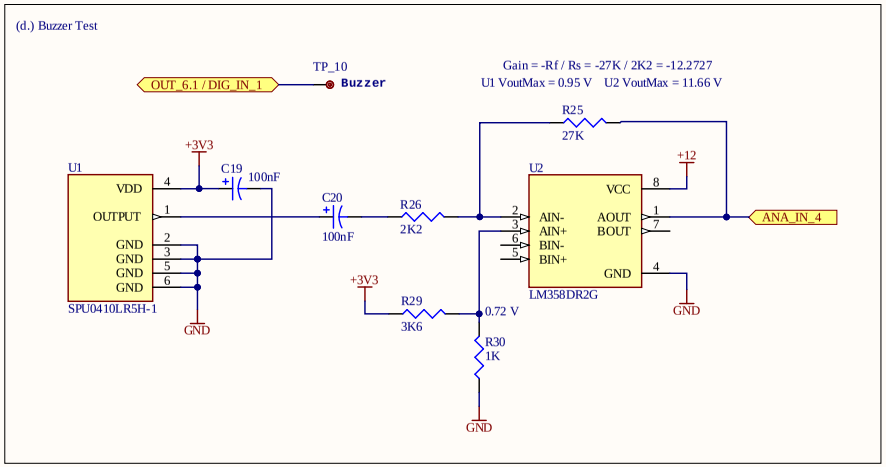


Title	Test-OK Connectors	
Size: A3	Number	Revision: V1.0
Date: 9-18-2021	Sheet 6f 3	
File: C:\Users\..\TestOK_Connectors.SchDoc	Drawn By:	



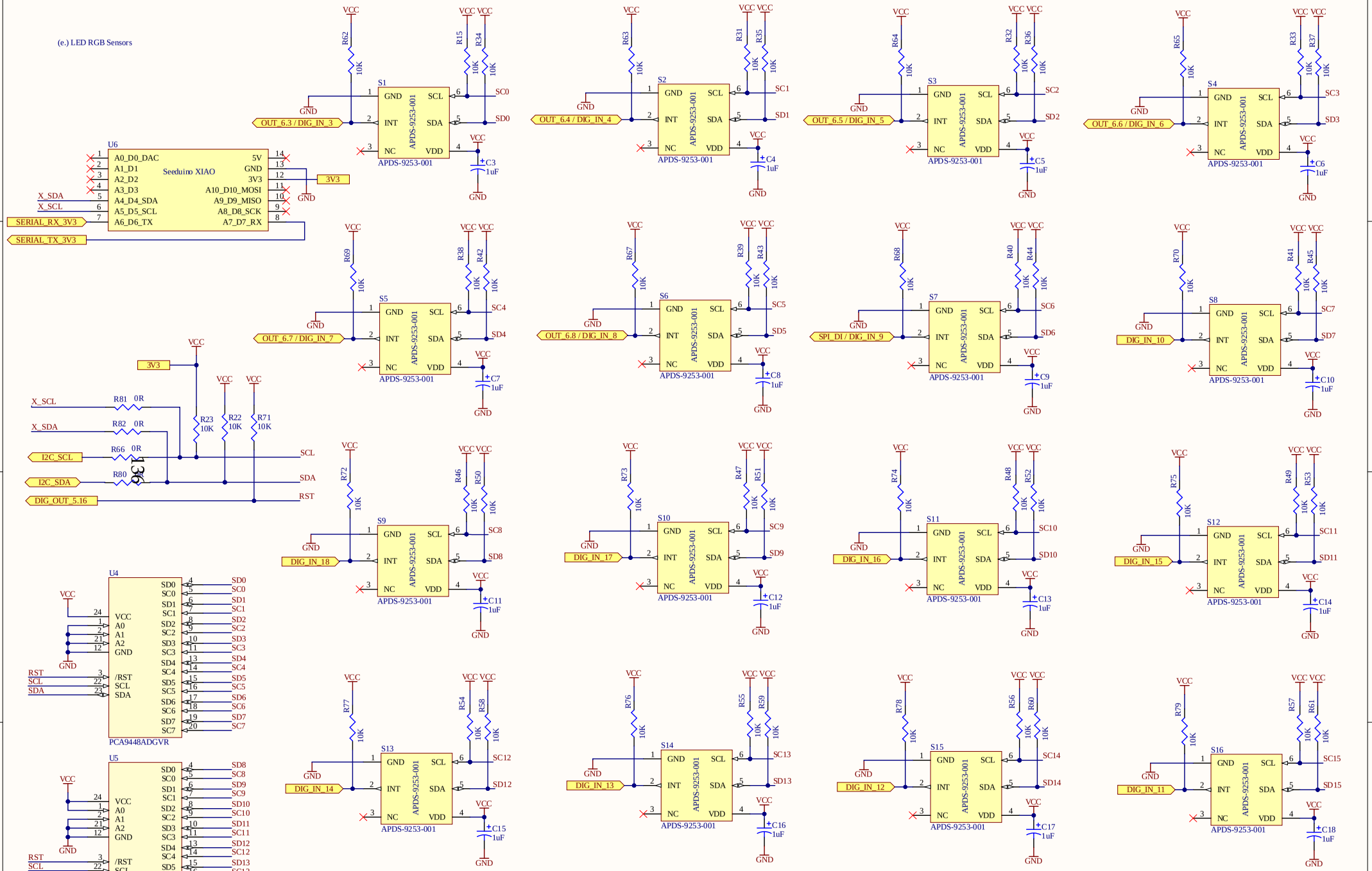






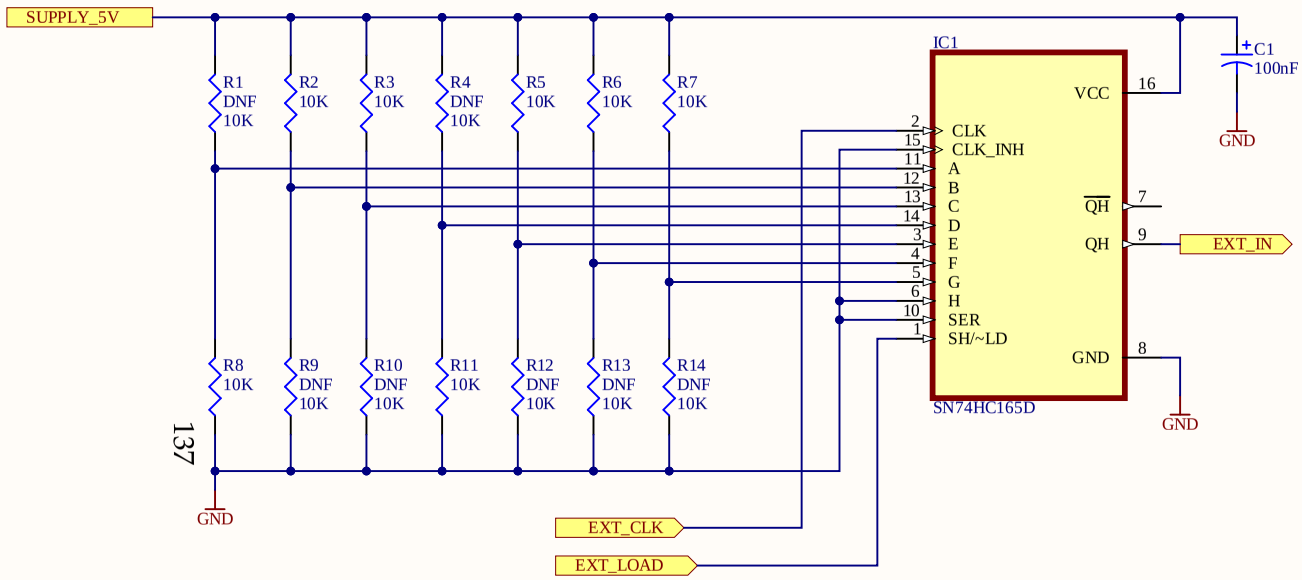
Title: Power Test Module		
Size: A3	Number:	Revision: V1.0
Date: 9-18-2021	Sheet 3 of 3	
File: C:\Users\..\TestPower.SchDoc	Drawn By:	

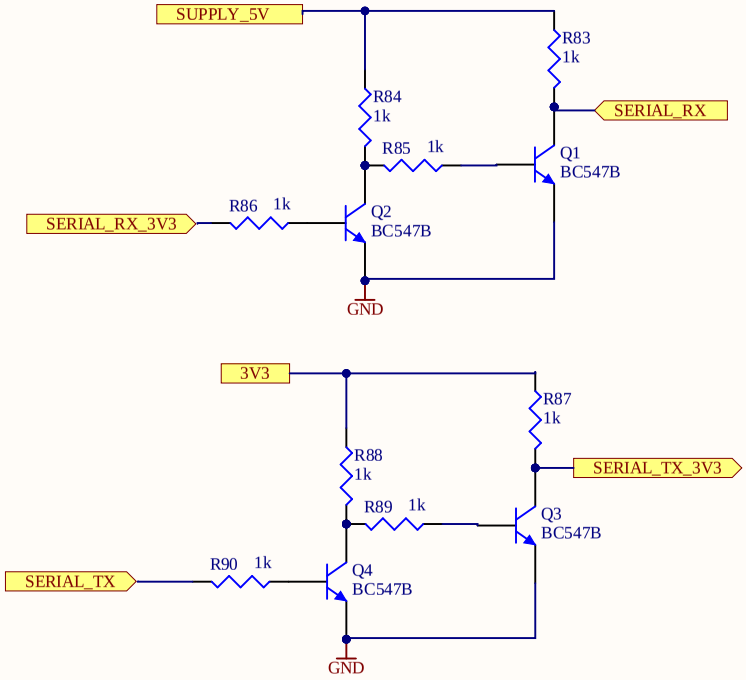
(e.) LED RGB Sensors

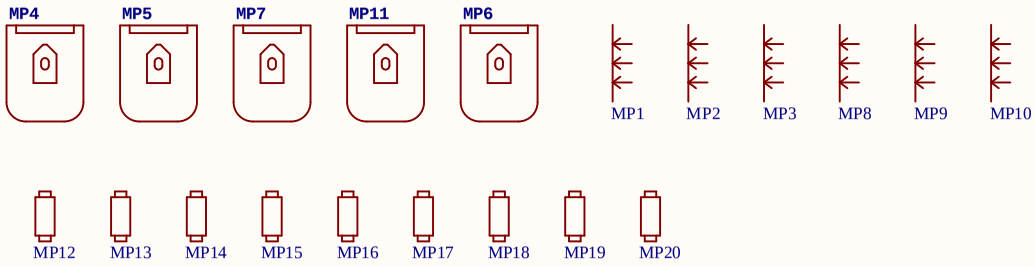


Title		
Size A3	Number	Revision
Date: 9-18-2021		Sheet of
File: C:\Users\..\TestLED.SchDoc		Drawn By:

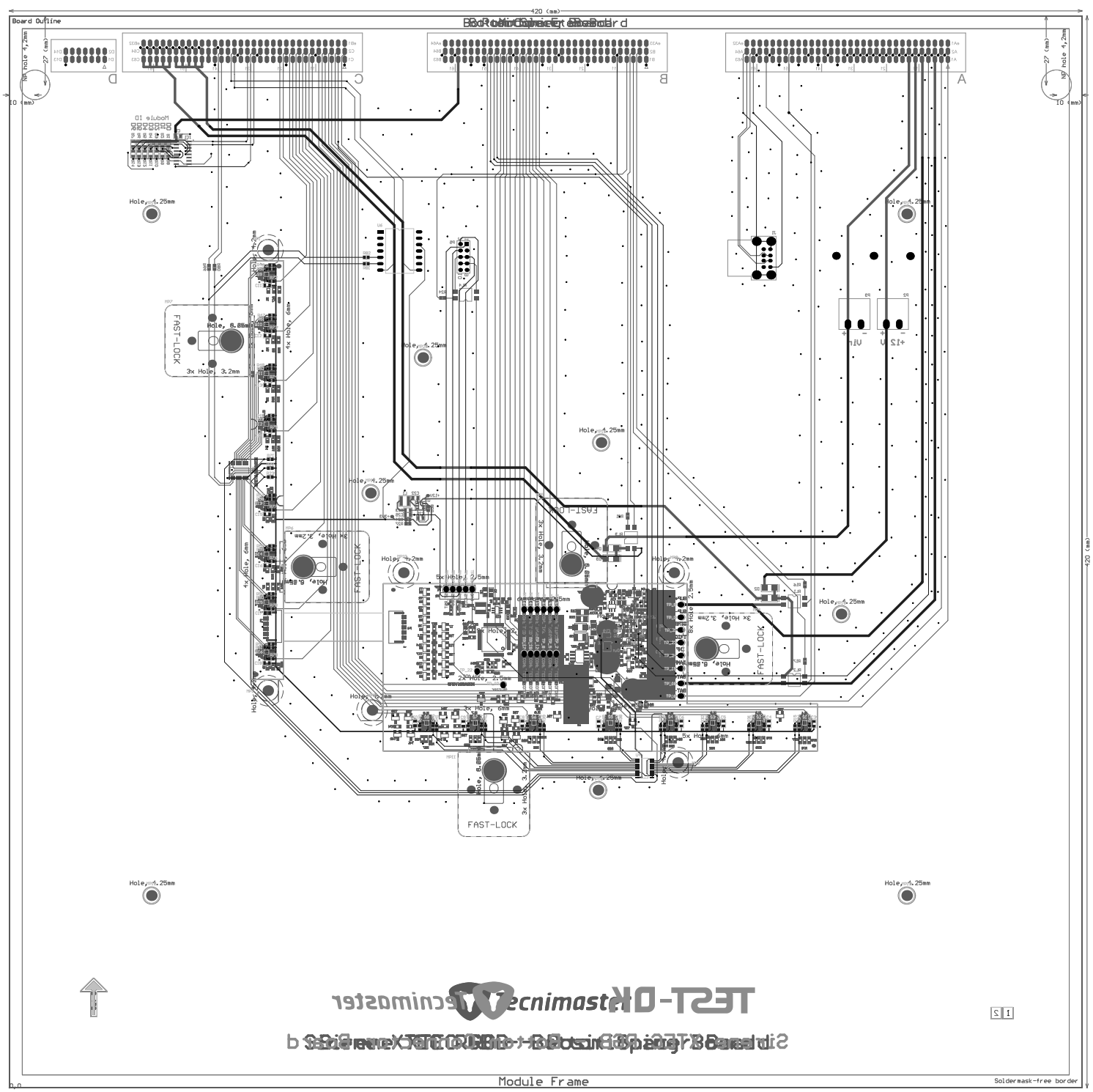
Board ID: 00001001b (0x09)







Title		
Size A4	Number	Revision
Date: 9-18-2021		Sheet of
File: C:\Users\..\TestOK_ID.SchDoc		Drawn By:


Board Outline
Module Frame
Soldermask-free border
FAST-LOCK
Hole, 4.25mm
Hole, 6.85mm
3x Hole, 3.2mm
4x Hole, 6mm
3x Hole, 3.2mm
Hole, 6.85mm
2x Hole, 2.5mm
2x Hole, 2.0mm
3x Hole, 6mm
+12 V
Module ID

A.3 Test Module Signal Table

Conector A					
Pin	**Signal**	**Module**	**Pin**	**Signal**	**Module**
1	ANA_IN_1	3V3 (a.)	2	GND	
3	ANA_IN_2	12V_ADC (a.)	4	GND	
5	ANA_IN_3	Check BAT+ (a.)	6	GND	
7	ANA_IN_4	Microphone (d.)	8	GND	
9	ANA_IN_5	Check SUP+ (a.)	10	GND	
11	ANA_IN_6	Check EXT_SUP1 (a.)	12	GND	
13	ANA_IN_7	Check EXT_SUP2 (a.)	14	GND	
15	ANA_IN_8		16	GND	
17	ANA_IN_9		18	GND	
19	ANA_IN_10		20	GND	
21	ANA_IN_11		22	GND	
23	ANA_IN_12		24	GND	
25	ANA_IN_13		26	GND	
27	ANA_IN_14		28	GND	
29	ANA_IN_15		30	GND	
31	ANA_IN_16		32	GND	
33	ANA_OUT_1		34	ANA_OUT_2	
35	GND		36	ANA_OUT_3	
37	ANA_OUT_4		38	GND	
39	ANA_OUT_5		40	ANA_OUT_6	
41	GND		42	ANA_OUT_7	
43	ANA_OUT_8		44	GND	
45	ANA_OUT_9		46	ANA_OUT_10	
47	GND		48	ANA_OUT_11	
49	ANA_OUT_12		50	GND	
51	ANA_OUT_13		52	ANA_OUT_14	
53	GND		54	ANA_OUT_15	
55	ANA_OUT_16		56	GND	
57	BOARD_DETECT	BAT- (a.)	58	Reserved	
59	Reserved		60	GND	
61	USB_VBUS	Programmer (c.)	62	USB_D+	Programmer (c.)
63	USB_D-	Programmer (c.)	64	USB_GND	Programmer (c.)

Conector B					
Pin	**Signal**	**Module**	**Pin**	**Signal**	**Module**
1	GND		2	DIG_OUT_1.1	
3	DIG_OUT_1.2		4	DIG_OUT_1.3	Control SUP+ (a.)
5	DIG_OUT_1.4	Control BAT+ (a.)	6	DIG_OUT_1.5	Enable Power for Control (b.)
7	DIG_OUT_1.6	Enable Power for Programmer (c.)	8	DIG_OUT_1.7	
9	DIG_OUT_1.8		10	GND	
11	DIG_OUT_2.1		12	DIG_OUT_2.2	
13	DIG_OUT_2.3		14	DIG_OUT_2.4	
15	DIG_OUT_2.5		16	DIG_OUT_2.6	
17	DIG_OUT_2.7		18	DIG_OUT_2.8	
19	GND		20	DIG_OUT_3.1	
21	DIG_OUT_3.2		22	DIG_OUT_3.3	
23	DIG_OUT_3.4		24	DIG_OUT_3.5	
25	DIG_OUT_3.6		26	DIG_OUT_3.7	
27	DIG_OUT_3.8		28	GND	
29	DIG_OUT_4.1		30	DIG_OUT_4.2	
31	DIG_OUT_4.3		32	DIG_OUT_4.4	
33	GND		34	DIG_OUT_5.1	Conf STROBE I1.1 (b.)
35	DIG_OUT_5.2	Conf TRIGGER I1.2 (b.)	36	DIG_OUT_5.3	Conf INF I1.3 (b.)
37	DIG_OUT_5.4	Conf TAMPER I1.4 (b.)	38	DIG_OUT_5.5	LED_MODE0 I1.5 (b.)
39	DIG_OUT_5.6	LED_MODE1 I1.6 (b.)	40	DIG_OUT_5.7	LED1_COR0 I2.1 (b.)
41	DIG_OUT_5.8	LED1_COR1 I2.2 (b.)	42	GND	
43	DIG_OUT_5.9	LED1_COR2 I2.3 (b.)	44	DIG_OUT_5.10	LED2_COR0 I2.4 (b.)
45	DIG_OUT_5.11	LED2_COR1 I2.5 (b.)	46	DIG_OUT_5.12	LED2_COR3 I2.6 (b.)
47	DIG_OUT_5.13	Control STRB F3.3 (b.)	48	DIG_OUT_5.14	Control TRIG F3.4 (b.)
49	DIG_OUT_5.15	Control INF F3.5 (b.)	50	DIG_OUT_5.16	Reset I2C (e.)
51	GND		52	PGM_MCLR	
53	PGM_DATA_IN		54	PGM_CLK	
55	GND		56	Reserved	
57	PGM_DATA_OUT		58	PGM_RESET	
59	+5V/250mA		60	GND	
61	PGM_VCC		62	CANL	
63	CANH		64	GND	

Conector C					
Pin	**Signal**	**Module**	**Pin**	**Signal**	**Module**
1	GND		2	DIG_IN_1	Buzzer Freq (d.)
3	DIG_OUT_6.2	Control TAMP F3.6 (b.)	4	DIG_IN_3	S1 Int (e.)
5	DIG_IN_4	S2 Int (e.)	6	DIG_IN_5	S3 Int (e.)
7	DIG_IN_6	S4 Int (e.)	8	DIG_IN_7	S5 Int (e.)
9	DIG_IN_8	S6 Int (e.)	10	GND	
11	DIG_IN_9	S7 Int (e.)	12	DIG_IN_10	S8 Int (e.)
13	DIG_IN_11	S9 Int (e.)	14	DIG_IN_12	S10 Int (e.)
15	DIG_IN_13	S16 Int (e.)	16	DIG_IN_14	S15 Int (e.)
17	DIG_IN_15	S14 Int (e.)	18	DIG_IN_16	S13 Int (e.)
19	GND		20	DIG_IN_17	S12 Int (e.)
21	DIG_IN_18	S11 Int (e.)	22	DIG_IN_19	
23	DIG_IN_20		24	ETHERNET_RX+	
25	ETHERNET_RX-		26	ETHERNET_TX+	
27	ETHERNET_TX-		28	GND	
29	EXT_CLK		30	EXT_IN	
31	EXT_LOAD		32	EXT_OUT	
33	EXT_STROBE		34		
35	SERIAL_RX1		36	SERIAL_TX2	
37	PWM_1		38	PWM_2	
39	I2C_SDA	LED Sensors (e.)	40	I2C_SCL	LED Sensors (e.)
41	GND		42	GND	
43	+26V		44	+26V	
45	V_SUPPLY_1	UUT Power (a.)	46	GND	
47	V_SUPPLY_1	UUT Power (a.)	48	GND	
49	V_SUPPLY_1	UUT Power (a.)	50	GND	
51	V_SUPPLY_1	UUT Power (a.)	52	GND	
53	V_SUPPLY_1	UUT Power (a.)	54	GND	
55	V_SUPPLY_2	Module Power	56	GND	
57	V_SUPPLY_2	Module Power	58	GND	
59	V_SUPPLY_2	Module Power	60	GND	
61	V_SUPPLY_2	Module Power	62	GND	
63	V_SUPPLY_2	Module Power	64	GND	

Conector D					
Pin	**Signal**	**Module**	**Pin**	**Signal**	**Module**
1	Reserved		2	Reserved	
3	24V AC 2b		4	24V AC 2b	
5	24V AC 2a		6	24V AC 2a	
7	24V AC 1b		8	24V AC 1b	
9	24V AC 1a		10	24V AC 1a	
11	Cover Switch		12	Relay	
13	GND		14	+26V	

Appendix B

Appendix - Code

B.1 Test Track Program

B.1.1 Preamble

```
// VAR
VAR #UUT_PANEL_ID_1 = 1;
VAR #UUT_PANEL_ID_2 = 2;
VAR #UUT_PANEL_ID_3 = 3;
VAR #UUT_PANEL_ID_4 = 4;
VAR #UUT_PANEL_ID_5 = 5;
VAR #UUT_PANEL_ID_6 = 6;
VAR #UUT_PANEL_ID_7 = 7;
VAR #UUT_PANEL_ID_8 = 8;
VAR #UUT_ID_BOARD = 9;

//MAPS IN
MAP $FREQ_BUZZ ON DIGITAL IN BIT 1;
MAP $CONTROL_TAMP_IN ON DIGITAL IN BIT 2;
MAP $S1_INT ON DIGITAL IN BIT 3;
MAP $S2_INT ON DIGITAL IN BIT 4;
MAP $S3_INT ON DIGITAL IN BIT 5;
MAP $S4_INT ON DIGITAL IN BIT 6;
MAP $S5_INT ON DIGITAL IN BIT 7;
MAP $S6_INT ON DIGITAL IN BIT 8;
MAP $S7_INT ON DIGITAL IN BIT 9;
MAP $S8_INT ON DIGITAL IN BIT 10;
MAP $S9_INT ON DIGITAL IN BIT 18;
MAP $S10_INT ON DIGITAL IN BIT 17;
MAP $S11_INT ON DIGITAL IN BIT 16;
MAP $S12_INT ON DIGITAL IN BIT 15;
MAP $S13_INT ON DIGITAL IN BIT 14;
MAP $S14_INT ON DIGITAL IN BIT 13;
MAP $S15_INT ON DIGITAL IN BIT 12;
MAP $S16_INT ON DIGITAL IN BIT 11;

MAP $S1_8_INT ON DIGITAL IN BIT 3..10;
MAP $S16_9_INT ON DIGITAL IN BIT 11..18;

//
// "MAPS OUT"
MAP $CONTROL_SUP ON DIGITAL OUT GROUP 1, BIT 3;
MAP $CONTROL_BAT ON DIGITAL OUT GROUP 1, BIT 4;
MAP $CONTROL_EN ON DIGITAL OUT GROUP 1, BIT 5;
MAP $PROG_EN ON DIGITAL OUT GROUP 1, BIT 6;
MAP $CONF_STRB ON DIGITAL OUT GROUP 5, BIT 1;
MAP $CONF_TRIG ON DIGITAL OUT GROUP 5, BIT 2;
MAP $CONF_INF ON DIGITAL OUT GROUP 5, BIT 3;
MAP $CONF_TAMP ON DIGITAL OUT GROUP 5, BIT 4;
MAP $CONF_LED_MODE0 ON DIGITAL OUT GROUP 5, BIT 5;
MAP $CONF_LED_MODE1 ON DIGITAL OUT GROUP 5, BIT 6;
MAP $CONF_LED1_COLOR0 ON DIGITAL OUT GROUP 5, BIT 7;
MAP $CONF_LED1_COLOR1 ON DIGITAL OUT GROUP 5, BIT 8;
MAP $CONF_LED1_COLOR2 ON DIGITAL OUT GROUP 5, BIT 9;
MAP $CONF_LED2_COLOR0 ON DIGITAL OUT GROUP 5, BIT 10;
MAP $CONF_LED2_COLOR1 ON DIGITAL OUT GROUP 5, BIT 11;
MAP $CONF_LED2_COLOR2 ON DIGITAL OUT GROUP 5, BIT 12;
MAP $CONTROL_STRB ON DIGITAL OUT GROUP 5, BIT 13;
MAP $CONTROL_TRIG ON DIGITAL OUT GROUP 5, BIT 14;
MAP $CONTROL_INF ON DIGITAL OUT GROUP 5, BIT 15;
MAP $RESET_I2C ON DIGITAL OUT GROUP 5, BIT 16;
MAP $CONTROL_TAMP_OUT ON DIGITAL OUT GROUP 6, BIT 2;

//MAPS ANALOG IN
MAP $3V3 ON ANALOG IN 1;
MAP $12V_ADC ON ANALOG IN 2;
MAP $CHECK_BAT ON ANALOG IN 3;
MAP $MICROPHONE ON ANALOG IN 4;
MAP $CHECK_SUP ON ANALOG IN 5;
```

```
65:
66: //
67: // ##########     INITIALIZATIONS
68: // OUTPUTS
69: SET_DIGITAL [ GROUP 1, BIT 1 .. 8 ] = OFF;
70: SET_DIGITAL [ GROUP 2, BIT 1 .. 8 ] = OFF;
71: SET_DIGITAL [ GROUP 3, BIT 1 .. 8 ] = OFF;
72: SET_DIGITAL [ GROUP 5, BIT 1 .. 15 ] = OFF;
73: SET_DIGITAL [ GROUP 6, BIT 1 .. 8 ] = OFF;
74: CONFIG_DIGITAL_GROUP [ 1 ] = 12.000;
75: SET_DIGITAL [ GROUP 1, BIT 1 .. 8 ] = OFF;
76: SET_DIGITAL [ GROUP 2, BIT 1 .. 8 ] = OFF;
77: SET_DIGITAL [ GROUP 3, BIT 1 .. 8 ] = OFF;
78: SET_DIGITAL [ GROUP 5, BIT 1 .. 15 ] = OFF;
79: SET_DIGITAL [ GROUP 6, BIT 1 .. 8 ] = OFF;
80: //
81: // COUNTER CONFIG
82: CONFIG_SERIAL [ 1 ] BAUDRATE = 115200, PARITY = NONE, TXMODE = MANUAL, RXMODE =
    MANUAL, RXTHRESHOLD = 128, RXTIMEOUT = 200, STXMODE = OFF, TERMINALMODE = CRLF;
83: CONFIG_COUNTER [ 1 ] MODE = FREQUENCY;
84: //
85:
86:
```

B.1.2 CPU Test

```
 1: VAR #APP_PATH = "\"C:\\Program Files
    (x86)\\STMicroelectronics\\st_toolset\\stvp\\STVP_CmdLine.exe\"";
 2: VAR #HEX_PATH = "\"C:\\Users\\TECNIMASTER
    TESTE\\AppData\\Local\\VirtualStore\\Program Files\\TestOK\\009.Xtec
    RGB\\Firmware\\sirene_i_v3.0.0.7_20201107.hex\"";
 3:
 4: CONFIG_SUPPLY [ 2 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
 5: SET_SUPPLY [ 2 ] = ON;
 6: SET_DIGITAL [ $CONTROL_EN ] = ON;
 7: SET_DIGITAL [ $PROG_EN ] = ON;
 8: SET_USB [ 1 ] = ON;
 9: WAITMS 1000;
10:
11: RUN "#APP_PATH# -BoardName=ST-LINK -Port=USB -ProgMode=SWD -Device=STM32F030x8 -
    FileProg=#HEX_PATH#", HIDE = ON;
12:
13: SET_DIGITAL [ $PROG_EN ] = OFF;
14: SET_USB [ 1 ] = OFF;
15:
16:
```

B.1.3 Buzzer Test

```
 1: VAR #DEF_FREQ_MIN = 2100;
 2: VAR #DEF_FREQ_MAX = 3300;
 3: VAR #DEF_SPL_MIN = 750;
 4: VAR #DEF_SPL_MAX = 900;
 5: VAR #DEF_0_MIN = 0;
 6: VAR #DEF_0_MAX = 50;
 7: VAR #TIMEOUT_READ=10000;
 8:
 9: CONFIG_LIMITS["DEF_FREQ"] LOW=#DEF_FREQ_MIN, HIGH=#DEF_FREQ_MAX;
10: CONFIG_LIMITS["DEF_SPL"] LOW=#DEF_SPL_MIN, HIGH=#DEF_SPL_MAX;
11:
12: LOG "  ------------------- A INICIAR TESTE DE BUZZER -------------------";
13:
14: //------------------- INITIAL CONDITIONS -------------------
15: CONFIG_SUPPLY [ 2 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
16: SET_SUPPLY [ 2 ] = ON;
17: CONFIG_SUPPLY [ 1 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
18: SET_SUPPLY [ 1 ] = ON;
19: WAITMS 50;
20: CONFIG_COUNTER [ $FREQ_BUZZ ] MODE = FREQUENCY;
21: SET_DIGITAL [ GROUP 5, BIT 1 .. 3 ] = ON;
22: SET_DIGITAL [ GROUP 5, BIT 13 .. 15 ] = OFF;
23: SET_DIGITAL [ $CONTROL_SUP ] = ON;
24: SET_DIGITAL [ $CONTROL_EN ] = ON;
25:
26: //------------------- TEST CODE -------------------
27: LOG "  ------------------- A INICIAR TESTE DE BUZZER -------------------";
28: SET_DIGITAL [ $CONTROL_TRIG ] = ON;
29: WAITWHILE ( #TIMEOUT_READ )
30:     TEST_COUNTER [ 1 ] EXPECT (< #DEF_FREQ_MIN);
31: TEST_LIMITS ["DEF_FREQ"] ON (#_IN_) ELSE ABORT_ALL, "  BUZZER NAO TEM A
    FREQUENCIA ADEQUADA DE SINAL   :FALHOU (#_IN_# Hz) [MIN = #DEF_FREQ_MIN#, MAX
    =#DEF_FREQ_MAX# ]", KVP = "BUZZ_FREQ", UNIT = "Hz";
32: LOG "  FREQUENCIA BUZZER CORRECTA >>> PASSOU (#_IN_# Hz)";
33:
34: WAITWHILE ( #TIMEOUT_READ )
35:     TEST_ANALOG [ $MICROPHONE ] EXPECT (< #DEF_SPL_MIN);
36: SET_DIGITAL [ $CONTROL_TRIG ] = OFF;
37: TEST_LIMITS["DEF_SPL"] ON (#_IN_) ELSE ABORT_ALL, "  BUZZER NAO FAZ BARULHO
    SUFICIENTE   : FALHOU (#_IN_# mV) [MIN = #DEF_SPL_MIN#, MAX = #DEF_SPL_MAX# ]",
    KVP = "BUZZ_SPL", UNIT = "mV";
38: LOG "  BARULHO BUZZER CORRECTO >>> PASSOU (#_IN_# mV)";
39: LOG "  ------------------- TESTE DE BUZZER BEM SUCEDIDO -------------------";
40:
41:
```

B.1.4 Power Test

```
 1: VAR #DEF_12V_ADC_MIN = 1920;     //2.022 V - 5 % = 1.9209
 2: VAR #DEF_12V_ADC_MAX = 2123;     //2.022 V + 5 % = 2.1231
 3: VAR #DEF_3V3_MIN = 3135;
 4: VAR #DEF_3V3_MAX = 3465;
 5: VAR #DEF_0V_MIN = 0;
 6: VAR #DEF_0V_MAX = 250;
 7: VAR #DEF_VBAT_MIN = 7980;        //8.4 V - 5 % = 7.98 V
 8: VAR #DEF_VBAT_MAX = 8820;       //8.4 V + 5 % = 8.82 V
 9: VAR #DEF_VIN_CURR_MIN = 200;
10: VAR #DEF_VIN_CURR_MAX = 450;
11: VAR #DEF_VBAT_CURR_MIN = 200;
12: VAR #DEF_VBAT_CURR_MAX = 450;
13: VAR #DEF_FREQ_MIN = 2300;
14: VAR #DEF_FREQ_MAX = 2700;
15: VAR #DEF_SPL_MIN = 750;
16: VAR #DEF_SPL_MAX = 900;
17:
18: CONFIG_LIMITS["DEF_12V_ADC"] LOW=#DEF_12V_ADC_MIN, HIGH=#DEF_12V_ADC_MAX;
19: CONFIG_LIMITS["DEF_3V3"] LOW=#DEF_3V3_MIN, HIGH=#DEF_3V3_MAX;
20: CONFIG_LIMITS["DEF_0V"] LOW=#DEF_0V_MIN, HIGH=#DEF_0V_MAX;
21: CONFIG_LIMITS["DEF_VBAT"] LOW=#DEF_VBAT_MIN, HIGH=#DEF_VBAT_MAX;
22: CONFIG_LIMITS["DEF_VIN_CURR"] LOW=#DEF_VIN_CURR_MIN, HIGH=#DEF_VIN_CURR_MAX;
23: CONFIG_LIMITS["DEF_VBAT_CURR"] LOW=#DEF_VBAT_CURR_MIN, HIGH=#DEF_VBAT_CURR_MAX;
24: CONFIG_LIMITS["DEF_FREQ"] LOW=#DEF_FREQ_MIN, HIGH=#DEF_FREQ_MAX;
25: CONFIG_LIMITS["DEF_SPL"] LOW=#DEF_SPL_MIN, HIGH=#DEF_SPL_MAX;
26:
27: //VAR #TEMP_READ=0;
28: VAR #TIMEOUT_READ=10000;
29: VAR #TIMEOUT_BAT=10000;
30:
31: LOG "  ------------------- A INICIAR TESTE DE FONTE -------------------";
32:
33: //-------------------- INITIAL CONDITIONS --------------------
34: CONFIG_SUPPLY [ 2 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
35: CONFIG_COUNTER [ 1 ] MODE = FREQUENCY;
36: SET_SUPPLY [ 2 ] = ON;
37: SET_DIGITAL [ GROUP 5, BIT 1 .. 3 ] = ON;
38: SET_DIGITAL [ GROUP 5, BIT 13 .. 15 ] = OFF;
39: SET_DIGITAL [ $CONTROL_EN ] = ON;
40:
41: //-------------------- TEST CODE --------------------
42: //-------------------- SUP TEST WITH MINUMUM VOLTAGE --------------------
43:
44: CONFIG_SUPPLY [ 1 ] VOLTAGE = 10.000, CURRENTLIMIT = 0.500;
45: SET_SUPPLY [ 1 ] = ON;
46: LOG "";
47: SET_DIGITAL [ $CONTROL_SUP ] = ON;
48: LOG "  SUP+ ON com +10.00 V com max 0.50 A";
49: WAITMS 1000;
50:
51:
52: LOG "  ------------------- A INICIAR TESTE DE BUZZER -------------------";
53: SET_DIGITAL [ $CONTROL_TRIG ] = ON;
54: WAITWHILE ( #TIMEOUT_READ )
55:     TEST_COUNTER [ 1 ] EXPECT (< #DEF_FREQ_MIN);
56: TEST_LIMITS ["DEF_FREQ"] ON (#_IN_) ELSE ABORT_ALL, "  BUZZER NAO TEM A
    FREQUENCIA ADEQUADA DE SINAL   :FALHOU (#_IN_# Hz) [MIN = #DEF_FREQ_MIN#, MAX
    =#DEF_FREQ_MAX# ]", KVP = "BUZZ_FREQ", UNIT = "Hz";
57: LOG "  FREQUENCIA BUZZER CORRECTA >>> PASSOU (#_IN_# Hz)";
58:
59: WAITWHILE ( #TIMEOUT_READ )
60:     TEST_ANALOG [ $MICROPHONE ] EXPECT (< #DEF_SPL_MIN);
61: SET_DIGITAL [ $CONTROL_TRIG ] = OFF;
62: TEST_LIMITS["DEF_SPL"] ON (#_IN_) ELSE ABORT_ALL, "  BUZZER NAO FAZ BARULHO
    SUFICIENTE   : FALHOU (#_IN_# mV) [MIN = #DEF_SPL_MIN#, MAX = #DEF_SPL_MAX# ]",
    KVP = "BUZZ_SPL", UNIT = "mV";
```

```
 63: LOG "  BARULHO BUZZER CORRECTO >>> PASSOU (#_IN_# mV)";
 64: LOG "  ------------------- TESTE DE BUZZER BEM SUCEDIDO -------------------";
 65:
 66:
 67:
 68: LOG " -- A ESPERAR ARRANQUE INICIAL --";
 69: SET_DIGITAL [ $CONTROL_STRB ] = ON;
 70: WAITWHILE ( #TIMEOUT_READ )
 71:     TEST_ANALOG [ $12V_ADC ] EXPECT (< #DEF_12V_ADC_MIN);
 72: TEST_LIMITS["DEF_12V_ADC"] ON (#_IN_) ELSE ABORT_ALL, "  SUP+ ON > +12V ADC   :
     FALHOU (#_IN_# mV) [MIN = #DEF_12V_ADC_MIN#, MAX =#DEF_12V_ADC_MAX#]", KVP =
     "12V_ADC_VIN_ON", UNIT = "mV";
 73: SET_DIGITAL [ $CONTROL_TRIG ] = OFF;
 74: LOG "  SUP+ ON > +12V ADC >>> PASSOU (#_IN_# mV)";
 75:
 76: TEST_ANALOG [ $3V3 ];
 77: TEST_LIMITS["DEF_3V3"] ON (#_IN_) ELSE ABORT_ALL, "  SUP+ ON > +3V3   : FALHOU
     (#_IN_# mV) [MIN = #DEF_3V3_MIN#, MAX =#DEF_3V3_MAX#]", KVP = "3V3_VIN_ON",
     UNIT = "mV";
 78: LOG "  SUP+ ON > +3V3 >>> PASSOU (#_IN_# mV)";
 79:
 80: //SET_DIGITAL [ $CONTROL_STRB ] = ON;
 81: WAITWHILE ( #TIMEOUT_READ )
 82:     TEST_SUPPLYCURRENT [ 1 ] EXPECT (< #DEF_VIN_CURR_MIN);
 83: TEST_LIMITS["DEF_VIN_CURR"] ON (#_IN_) ELSE ABORT_ALL, "  SUP+ ON > Current   :
     FALHOU (#_IN_# mA) [MIN = #DEF_VIN_CURR_MIN#, MAX =#DEF_VIN_CURR_MAX#]", KVP =
     "Isup", UNIT = "mA";
 84: SET_DIGITAL [ $CONTROL_STRB ] = OFF;
 85: LOG "  SUP+ ON > Current >>> PASSOU (#_IN_# mA)";
 86:
 87: LOG " -- A ESPERAR SEGUNDO ARRANQUE --";
 88: CONFIG_SUPPLY [ 1 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
 89: SET_SUPPLY [ 1 ] = ON;
 90: WAITWHILE ( #TIMEOUT_BAT )
 91:     TEST_ANALOG [ $CHECK_BAT ] EXPECT (< #DEF_VBAT_MIN);
 92: TEST_LIMITS["DEF_VBAT"] ON (#_IN_) ELSE CONTINUE, "  SUP+ ON > +Vbat   : FALHOU
     (#_IN_# mV) [MIN = #DEF_VBAT_MIN#, MAX =#DEF_VBAT_MAX#]", KVP = "VBAT_VAL",
     UNIT = "mV";
 93: LOG "  SUP+ ON > +Vbat >>> PASSOU (#_IN_# mV)";
 94:
 95:
 96:
 97: //------------------- SUP TEST WITH POWER OFF -------------------
 98: SET_DIGITAL [ $CONTROL_STRB ] = OFF;
 99: SET_DIGITAL [ $CONTROL_SUP ] = OFF;
100: LOG "";
101: LOG "  SUP+ OFF";
102: WAITMS 5000;
103:
104: TEST_ANALOG [ $12V_ADC ];
105: TEST_LIMITS["DEF_0V"] ON (#_IN_) ELSE ABORT_ALL, "  SUP+ OFF > +12V ADC   :
     FALHOU (#_IN_# mV) [MIN = #DEF_0V_MIN#, MAX =#DEF_0V_MAX#]", KVP =
     "12V_ADC_VAL_OFF", UNIT = "mV";
106: LOG "  SUP+ OFF > +12V ADC >>> PASSOU (#_IN_# mV)";
107:
108: TEST_ANALOG [ $3V3 ];
109: TEST_LIMITS["DEF_0V"] ON (#_IN_) ELSE ABORT_ALL, "  SUP+ OFF > +3V3   : FALHOU
     (#_IN_# mV) [MIN = #DEF_0V_MIN#, MAX =#DEF_0V_MAX#]", KVP = "3V3_VAL_OFF", UNIT
     = "mV";
110: LOG "  SUP+ OFF > +3V3 >>> PASSOU (#_IN_# mV)";
111:
112:
113: //------------------- BAT TEST WITH MINUMUM VOLTAGE -------------------
114: SET_DIGITAL [ $CONTROL_STRB ] = ON;
```

```
115: CONFIG_SUPPLY [ 1 ] VOLTAGE = 4.000, CURRENTLIMIT = 0.500;
116: SET_SUPPLY [ 1 ] = ON;
117: LOG "";
118: LOG "  BAT+ ON com +4.00 V with max 0.50 A";
119: SET_DIGITAL [ $CONTROL_BAT ] = ON;
120:
121: LOG " -- A ESPERAR ARRANQUE INICIAL PELA BATERIA --";
122: TEST_ANALOG [ $12V_ADC ];// EXPECT (< #DEF_12V_BAT_ADC_MIN);
123: TEST_LIMITS["DEF_0V"] ON (#_IN_) ELSE ABORT_ALL, "  BAT+ ON > +12V ADC   :
     FALHOU (#_IN_# mV) [MIN = #DEF_0V__MIN#, MAX =#DEF_0V_MAX#]", KVP =
     "12V_ADC_VBAT", UNIT = "mV";
124: LOG "  BAT+ ON > +12V ADC >>> PASSOU (#_IN_# mV)";
125:
126: WAITWHILE ( #TIMEOUT_READ )
127:     TEST_ANALOG [ $3V3 ] EXPECT (< #DEF_3V3_MIN);
128: TEST_LIMITS["DEF_3V3"] ON (#_IN_) ELSE ABORT_ALL, "  BAT+ ON > +3V3   : FALHOU
     (#_IN_# mV) [MIN = #DEF_3V3_MIN#, MAX =#DEF_3V3_MAX#]", KVP = "3V3_VBAT", UNIT
     = "mV";
129: LOG "  BAT+ ON > +3V3 >>> PASSOU (#_IN_# mV)";
130:
131: LOG " -- A ESPERAR SEGUNDO ARRANQUE PELA BATERIA --";
132: //SET_DIGITAL [ $CONTROL_STRB ] = ON;
133: WAITWHILE ( #TIMEOUT_READ )
134:     TEST_SUPPLYCURRENT [ 1 ] EXPECT (< #DEF_VBAT_CURR_MIN);
135: TEST_LIMITS["DEF_VBAT_CURR"] ON (#_IN_) ELSE ABORT_ALL, "  BAT+ ON > Current
     : FALHOU (#_IN_# mA) [MIN = #DEF_VBAT_CURR_MIN#, MAX =#DEF_VBAT_CURR_MAX#]",
     KVP = "Ibat", UNIT = "mA";
136: LOG "  BAT+ ON > Current >>> PASSOU (#_IN_# mA)";
137:
138: TEST_ANALOG [ $CHECK_SUP ];
139: TEST_LIMITS["DEF_0V"] ON (#_IN_) ELSE ABORT_ALL, "  BAT+ ON > +Vsup   : FALHOU
     (#_IN_# mV) [MIN = #DEF_0V_MIN#, MAX =#DEF_0V_MAX#]", KVP = "VIN_VAL", UNIT =
     "mV";
140: LOG "  BAT+ ON > +Vsup >>> PASSOU (#_IN_# mV)";
141:
142: LOG "  ------------------- TESTE DE FONTE TERMINADO -------------------";
143:
144:
```

B.1.5 LED Test

```
 1: VAR #TEMPO_MAX_STRB_F=850;
 2: VAR #TEMPO_MIN_STRB_F=0;
 3: VAR #TIMEOUT_STRB_F=900;
 4:
 5: VAR #TEMPO_MAX_STRB_S=1500;
 6: VAR #TEMPO_MIN_STRB_S=0;
 7: VAR #TIMEOUT_STRB_S=2000;
 8:
 9: VAR #TIMEOUT_FADE=5000;
10: VAR #TEMPO_MAX_FADE=4990;
11: VAR #FADE_TEST_MAX=4;
12: VAR #MAX_GONE=2;
13: VAR #MAX_FADE=3200;
14: VAR #MIN_FADE=30;
15:
16: VAR #TIMEOUT_BLINK=2000;
17: VAR #MAX_BLINK=1990;
18:
19: VAR #BLINK_INT_VAL1=1200;
20: VAR #BLINK_INT_VAL2=800;
21: VAR #STRB_INT_VAL=200;
22: VAR #FADE_INT_VAL=4000;
23:
24: VAR #TEMP_B;
25: VAR #TEMP_G;
26: VAR #TEMP_R;
27: VAR #TEMP_HEAT;
28: VAR #TEMP_LUX;
29: VAR #TEMP_ALL;
30:
31: VAR #B_MIN_OFF=0;
32: VAR #B_MAX_OFF=1000;
33: VAR #B_MIN_ON=1200;
34: VAR #B_MAX_ON=2500;
35:
36: VAR #R_MIN_OFF=0;
37: VAR #R_MAX_OFF=250;
38: VAR #R_MIN_ON=700;
39: VAR #R_MAX_ON=2500;
40:
41: VAR #G_MIN_OFF=0;
42: VAR #G_MAX_OFF=850;
43: VAR #G_MIN_ON=850;
44: VAR #G_MAX_ON=2500;
45:
46: VAR #TEMP_VAL;
47:
48: LOG "  ------------------- A INICIAR TESTE DE LEDS -------------------";
49:
50: //------------------- INITIAL CONDITIONS --------------------
51: CONFIG_SUPPLY [ 2 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
52: SET_SUPPLY [ 2 ] = ON;
53: CONFIG_SUPPLY [ 1 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
54: SET_SUPPLY [ 1 ] = ON;
55: SET_DIGITAL [ GROUP 5, BIT 1 .. 3 ] = ON;
56: SET_DIGITAL [ GROUP 5, BIT 13 .. 15 ] = OFF;
57: SET_DIGITAL [ $CONTROL_SUP ] = ON;
58: SET_DIGITAL [ $CONTROL_EN ] = ON;
59: WAITMS 5000;
60:
61: //CONFIGURA O LIMITE MÁXIMO DE LUMINOSIDADE PARA A INTERRUPÇÃO DE CADA SENSOR
62: LOG "------------------- A ENVIAR CONFIGURACOES -------------------";
63: TRANSMIT_SERIAL[1] "L", LOG=ON;
```

```
 64: RECEIVE_SERIAL[1] "1122334455667788", TIMEOUT = 5000 ELSE ABORT, LOG=ON;

 65: LOG "------------------- A CONFIGURAR INTERRUPCOES -------------------";
 66: FOR #SenseIndex { 1..8 }
 67:     TRANSMIT_SERIAL[1] "S:#SenseIndex#0;I:#BLINK_INT_VAL2:4x#";//, LOG=ON;
 68:     RECEIVE_SERIAL[1] "U4:? U5:0 I:* OK", TIMEOUT = 1000 ELSE CONTINUE;//,
     LOG=ON;
 69:     IF (#SenseIndex < 8)
 70:         TRANSMIT_SERIAL[1] "S:0#SenseIndex#;I:#BLINK_INT_VAL2:4x#";//, LOG=ON;
 71:         RECEIVE_SERIAL[1] "U4:0 U5:? I:* OK", TIMEOUT = 1000 ELSE CONTINUE;//,
     LOG=ON;
 72:     ENDIF;
 73:     //LOG " INT DL#SenseIndex# OK";
 74: ENDFOR;
 75:
 76: //------------------- TEST CODE ----------------------------------------------
     ------------------------------------------------------------------------------
     ----------
 77:
 78: LOG "------------------- A INICIAR TESTE DE CORES -------------------";
 79: SET_DIGITAL [ $CONF_LED_MODE0 ] = OFF;
 80: SET_DIGITAL [ $CONF_LED_MODE1 ] = OFF;
 81:
 82: FOR #COR_RGB { 1..3 }
 83:     LOG "";
 84:     IF ( #COR_RGB == 3 )
 85:         LOG "------ A TESTAR COR AZUL NOS LEDS ------";
 86:         SET_DIGITAL [ $CONF_LED1_COLOR2 ] = ON;
 87:         SET_DIGITAL [ $CONF_LED2_COLOR2 ] = ON;
 88:     ELSE
 89:         SET_DIGITAL [ $CONF_LED1_COLOR2 ] = OFF;
 90:         SET_DIGITAL [ $CONF_LED2_COLOR2 ] = OFF;
 91:     ENDIF;
 92:     IF ( #COR_RGB == 2 )
 93:         LOG "------ A TESTAR COR VERDE NOS LEDS ------";
 94:
 95:     LOG "--- A CONFIGURAR INTERRUPCOES PARA COR VERDE ---";
 96:     FOR #SenseIndex { 1..8 }
 97:         TRANSMIT_SERIAL[1] "S:#SenseIndex#0;I:#BLINK_INT_VAL1:4x#";//, LOG=ON;
 98:         RECEIVE_SERIAL[1] "U4:? U5:0 I:* OK", TIMEOUT = 1000 ELSE CONTINUE;//,
     LOG=ON;
 99:         IF (#SenseIndex < 8)
100:         TRANSMIT_SERIAL[1] "S:0#SenseIndex#;I:#BLINK_INT_VAL1:4x#";//, LOG=ON;
101:         RECEIVE_SERIAL[1] "U4:0 U5:? I:* OK", TIMEOUT = 1000 ELSE CONTINUE;//,
     LOG=ON;
102:         ENDIF;
103:     ENDFOR;
104:         SET_DIGITAL [ $CONF_LED1_COLOR1 ] = ON;
105:         SET_DIGITAL [ $CONF_LED2_COLOR1 ] = ON;
106:     ELSE
107:         SET_DIGITAL [ $CONF_LED1_COLOR1 ] = OFF;
108:         SET_DIGITAL [ $CONF_LED2_COLOR1 ] = OFF;
109:     ENDIF;
110:     IF ( #COR_RGB == 1 )
111:         LOG "------ A TESTAR COR VERMELHA NOS LEDS ------";
112:         SET_DIGITAL [ $CONF_LED1_COLOR0 ] = ON;
113:         SET_DIGITAL [ $CONF_LED2_COLOR0 ] = ON;
114:     ELSE
115:         SET_DIGITAL [ $CONF_LED1_COLOR0 ] = OFF;
116:         SET_DIGITAL [ $CONF_LED2_COLOR0 ] = OFF;
117:     ENDIF;
118:
119:     LOG "--- A TESTAR LEDS PLACA PRINCIPAL ---";
120:         FOR #SenseIndex { 1..8 }
```

```
121:         IF (#SenseIndex==1)
122:           TRANSMIT_SERIAL[1] "R";
123:         WAITMS 50;
124:           WAITWHILE ( #TIMEOUT_BLINK )
125:             TEST_DIGITAL[ $S1_INT] EXPECT == ON;
126:         ELIF (#SenseIndex==4)
127:           TRANSMIT_SERIAL[1] "R";
128: WAITMS 50;
129:           WAITWHILE ( #TIMEOUT_BLINK )
130:             TEST_DIGITAL[ $S4_INT] EXPECT == ON;
131:        ELIF (#SenseIndex==7)
132:           TRANSMIT_SERIAL[1] "R";
133: WAITMS 50;
134:           WAITWHILE ( #TIMEOUT_BLINK )
135:             TEST_DIGITAL[ $S7_INT] EXPECT == ON;
136:        ENDIF;
137:
138:         IF (#_WAITED_ > #MAX_BLINK)
139:                 FAIL " LED DL#SenseIndex# FALHOU NO MODO DE PISCAR
    (#_WAITED_#)", ABORT_ALL;
140:         ENDIF;
141:
142:         TRANSMIT_SERIAL[1] "S:#SenseIndex#0";//, LOG=ON;
143:         RECEIVE_SERIAL[1] "U4:? U5:0 Color Temp: #TEMP_HEAT# K - Lux:
    #TEMP_LUX# - R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",
    TIMEOUT=1000;//, LOG=ON;
144:
145:         LOG " DL#SenseIndex# PRINCIPAL - LUZ R: #TEMP_R# G: #TEMP_G# B:
    #TEMP_B# ";
146:
147:         IF ( #COR_RGB == 3 )
148:             IF ( (#TEMP_B < #B_MIN_ON) OR (#TEMP_B > #B_MAX_ON) )
149:                 FAIL " LED DL#SenseIndex# PRINCIPAL NAO MOSTROU LUZ AZUL ",
    CONTINUE;
150:             ENDIF;
151:         ELSE
152:             IF ( (#TEMP_B < #B_MIN_OFF) OR (#TEMP_B > #B_MAX_OFF) )
153:                 FAIL " LED DL#SenseIndex# PRINCIPAL MOSTROU LUZ AZUL QUANDO NAO
    DEVIA ", CONTINUE;
154:             ENDIF;
155:         ENDIF;
156:
157:         IF ( #COR_RGB == 2 )
158:             IF ( (#TEMP_G < #G_MIN_ON) OR (#TEMP_G > #G_MAX_ON) )
159:                 FAIL " LED DL#SenseIndex# PRINCIPAL NAO MOSTROU LUZ VERDE ",
    CONTINUE;
160:             ENDIF;
161:         ELSE
162:             IF ( (#TEMP_G < #G_MIN_OFF) OR (#TEMP_G > #G_MAX_OFF) )
163:                 FAIL " LED DL#SenseIndex# PRINCIPAL MOSTROU LUZ VERDE QUANDO
    NAO DEVIA ", CONTINUE;
164:             ENDIF;
165:         ENDIF;
166:
167:         IF ( #COR_RGB == 1 )
168:         IF ( (#TEMP_R < #R_MIN_ON) OR (#TEMP_R > #R_MAX_ON) )
169:                 FAIL " LED DL#SenseIndex# PRINCIPAL NAO MOSTROU LUZ VERMELHA ",
    CONTINUE;
170:             ENDIF;
171:         ELSE
172:             IF ( (#TEMP_R < #R_MIN_OFF) OR (#TEMP_R > #R_MAX_OFF) )
173:                 FAIL " LED DL#SenseIndex# PRINCIPAL MOSTROU LUZ VERMELHA QUANDO
    NAO DEVIA ", CONTINUE;
174:             ENDIF;
```

```
175:         ENDIF;
176:
177:     ENDFOR;
178:
179:
180:     LOG "--- A TESTAR LEDS DA LED STRIP ---";
181:     FOR #SenseIndex { 1..8 }
182:        IF (#SenseIndex==1)
183:          TRANSMIT_SERIAL[1] "R";
184:     WAITMS 50;
185:          WAITWHILE ( #TIMEOUT_BLINK )
186:            TEST_DIGITAL[ $S9_INT] EXPECT == ON;
187:        ELIF (#SenseIndex==4)
188:          TRANSMIT_SERIAL[1] "R";
189: WAITMS 50;
190:          WAITWHILE ( #TIMEOUT_BLINK )
191:            TEST_DIGITAL[ $S12_INT] EXPECT == ON;
192:        ELIF (#SenseIndex==7)
193:          TRANSMIT_SERIAL[1] "R";
194: WAITMS 50;
195:          WAITWHILE ( #TIMEOUT_BLINK )
196:            TEST_DIGITAL[ $S15_INT] EXPECT == ON;
197:        ENDIF;
198:
199:        IF (#_WAITED_ > #MAX_BLINK)
200:           FAIL " LED DL#SenseIndex# FALHOU NO MODO DE PISCAR (#_WAITED_#)",
    ABORT_ALL;
201:        ENDIF;
202:
203:        TRANSMIT_SERIAL[1] "S:0#SenseIndex#";//, LOG=ON;
204:        RECEIVE_SERIAL[1] "U4:0 U5:? Color Temp: #TEMP_HEAT# K - Lux: #TEMP_LUX#
    - R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",TIMEOUT=1000;//,
    LOG=ON;
205:
206:        LOG " DL#SenseIndex# SECUNDARIA - LUZ R: #TEMP_R# G: #TEMP_G# B:
    #TEMP_B# ";
207:
208:        IF ( #COR_RGB == 3 )
209:             IF ( (#TEMP_B < #B_MIN_ON) OR (#TEMP_B > #B_MAX_ON) )
210:                 FAIL " LED DL#SenseIndex# LED STRIP NAO MOSTROU LUZ AZUL ",
    CONTINUE;
211:             ENDIF;
212:         ELSE
213:             IF ( (#TEMP_B < #B_MIN_OFF) OR (#TEMP_B > #B_MAX_OFF) )
214:                 FAIL " LED DL#SenseIndex# LED STRIP MOSTROU LUZ AZUL QUANDO NAO
    DEVIA ", CONTINUE;
215:             ENDIF;
216:         ENDIF;
217:
218:         IF ( #COR_RGB == 2 )
219:             IF ( (#TEMP_G < #G_MIN_ON) OR (#TEMP_G > #G_MAX_ON) )
220:                 FAIL " LED DL#SenseIndex# LED STRIP NAO MOSTROU LUZ VERDE ",
    CONTINUE;
221:             ENDIF;
222:         ELSE
223:             IF ( (#TEMP_G < #G_MIN_OFF) OR (#TEMP_G > #G_MAX_OFF) )
224:                 FAIL " LED DL#SenseIndex# LED STRIP MOSTROU LUZ VERDE QUANDO
    NAO DEVIA ", CONTINUE;
225:             ENDIF;
226:         ENDIF;
227:
228:         IF ( #COR_RGB == 1 )
229:         IF ( (#TEMP_R < #R_MIN_ON) OR (#TEMP_R > #R_MAX_ON) )
230:                 FAIL " LED DL#SenseIndex# LED STRIP NAO MOSTROU LUZ VERMELHA ",
    CONTINUE;
```

```
231:             ENDIF;
232:         ELSE
233:             IF ( (#TEMP_R < #R_MIN_OFF) OR (#TEMP_R > #R_MAX_OFF) )
234:                 FAIL " LED DL#SenseIndex# LED STRIP MOSTROU LUZ VERMELHA QUANDO
    NAO DEVIA ", CONTINUE;
235:             ENDIF;
236:         ENDIF;
237:     ENDFOR;
238:     LOG ">>> PASSOU";
239: ENDFOR;
240:
241: LOG "------------------- TESTE DE CORES TERMINADO -------------------";
242: LOG "";
243: LOG "-------------- A INICIAR TESTE DE COMPORTAMENTOS ---------------";
244: SET_DIGITAL [ $CONTROL_SUP ] = OFF;
245: WAITMS 1000;
246: SET_DIGITAL [ $CONTROL_SUP ] = ON;
247: SET_DIGITAL [ $CONF_LED1_COLOR0 ] = ON;
248: SET_DIGITAL [ $CONF_LED1_COLOR1 ] = ON;
249: SET_DIGITAL [ $CONF_LED1_COLOR2 ] = ON;
250: SET_DIGITAL [ $CONF_LED2_COLOR0 ] = ON;
251: SET_DIGITAL [ $CONF_LED2_COLOR1 ] = ON;
252: SET_DIGITAL [ $CONF_LED2_COLOR2 ] = ON;
253:
254: //VAR #Reply=YES;
255: //WHILE (#Reply == YES)
256: LOG "------ A TESTAR BARRAS PISCAM UMA DE CADA VEZ -------";
    //=============================================================================
257: SET_DIGITAL [ $CONF_LED_MODE0 ] = OFF;
258: SET_DIGITAL [ $CONF_LED_MODE1 ] = OFF;
259:
260: LOG " A TESTAR LUZ NA PLACA PRINCIPAL ";
261:
262: TRANSMIT_SERIAL[1] "R";
263: WAITMS 100;
264: WAITWHILE ( #TIMEOUT_BLINK )
265:     TEST_DIGITAL[ $S9_INT] EXPECT == ON;
266: TRANSMIT_SERIAL[1] "R";
267: WAITMS 100;
268: WAITWHILE ( #TIMEOUT_BLINK )
269:     TEST_DIGITAL[ $S1_INT] EXPECT == ON;
270: TRANSMIT_SERIAL[1] "R";
271: WAITMS 200;
272:
273: IF (#_IN_ != OFF)
274:     FAIL " LED DL1 NA PLACA PRINCIPAL NAO LIGOU NO MODO DE PISCAR ", CONTINUE;

275: ENDIF;
276:
277: IF (#_WAITED_ > #MAX_BLINK)
278:     FAIL " TIMEOUT NO MODO DE PISCAR (#_WAITED_#)", CONTINUE;
279: ENDIF;
280:
281: TEST_DIGITAL[ $S9_INT] EXPECT == ON ELSE CONTINUE, " LED DL1 NA LED STRIP LIGOU
    NO MODO DE PISCAR QUANDO NAO DEVIA";
282: TEST_DIGITAL[ $S15_INT] EXPECT == ON ELSE CONTINUE, " LED DL7 NA LED STRIP
    LIGOU NO MODO DE PISCAR QUANDO NAO DEVIA";
283: TEST_DIGITAL[ $S8_INT] EXPECT == OFF ELSE CONTINUE, " LED DL8 NA PLACA
    PRINCIPAL NAO LIGOU NO MODO DE PISCAR ";
284:
285: LOG " A TESTAR LUZ NA LED STRIP ";//-----------------------------------------
    ---------------------------------------
286: //WAITMS 1000;
287: TRANSMIT_SERIAL[1] "R";
```

```
288: WAITMS 100;
289: WAITWHILE ( #TIMEOUT_BLINK )
290:     TEST_DIGITAL[ $S1_INT] EXPECT == ON;
291: TRANSMIT_SERIAL[1] "R";
292: WAITMS 100;
293: WAITWHILE ( #TIMEOUT_BLINK )
294:     TEST_DIGITAL[ $S9_INT] EXPECT == ON;
295: TRANSMIT_SERIAL[1] "R";
296: WAITMS 200;
297:
298: IF (#_IN_ != OFF)
299:     FAIL " LED DL1 NA LED STRIP NAO LIGOU NO MODO DE PISCAR ", CONTINUE;

300: ENDIF;
301:
302: IF (#_WAITED_ > #MAX_BLINK)
303:     FAIL " TIMEOUT NO MODO DE PISCAR (#_WAITED_#)", CONTINUE;
304: ENDIF;
305:
306: TEST_DIGITAL[ $S1_INT] EXPECT == ON ELSE CONTINUE, " LED DL1 NA PLACA PRINCIPAL
     LIGOU NO MODO DE PISCAR QUANDO NAO DEVIA";
307: TEST_DIGITAL[ $S8_INT] EXPECT == ON ELSE CONTINUE, " LED DL8 NA PLACA PRINCIPAL
     LIGOU NO MODO DE PISCAR QUANDO NAO DEVIA";
308: TEST_DIGITAL[ $S15_INT] EXPECT == OFF ELSE CONTINUE, " LED DL7 NA LED STRIP NAO
     LIGOU NO MODO DE PISCAR ";
309:
310: LOG " A TESTAR LUZ NA PLACA PRINCIPAL ";//-----------------------------------
     -------------------------------------------
311: //WAITMS 1000;
312: TRANSMIT_SERIAL[1] "R";
313: WAITMS 100;
314: WAITWHILE ( #TIMEOUT_BLINK )
315:     TEST_DIGITAL[ $S9_INT] EXPECT == ON;
316: TRANSMIT_SERIAL[1] "R";
317: WAITMS 100;
318: WAITWHILE ( #TIMEOUT_BLINK )
319:     TEST_DIGITAL[ $S1_INT] EXPECT == ON;
320: TRANSMIT_SERIAL[1] "R";
321: WAITMS 200;
322:
323: IF (#_IN_ != OFF)
324:     FAIL " LED DL1 NA PLACA PRINCIPAL NAO LIGOU NO MODO DE PISCAR ", CONTINUE;

325: ENDIF;
326:
327: IF (#_WAITED_ > #MAX_BLINK)
328:     FAIL " TIMEOUT NO MODO DE PISCAR (#_WAITED_#)", CONTINUE;
329: ENDIF;
330:
331: TEST_DIGITAL[ $S9_INT] EXPECT == ON ELSE CONTINUE, " LED DL1 NA LED STRIP LIGOU
     NO MODO DE PISCAR QUANDO NAO DEVIA";
332: TEST_DIGITAL[ $S15_INT] EXPECT == ON ELSE CONTINUE, " LED DL7 NA LED STRIP
     LIGOU NO MODO DE PISCAR QUANDO NAO DEVIA";
333: TEST_DIGITAL[ $S8_INT] EXPECT == OFF ELSE CONTINUE, " LED DL8 NA PLACA
     PRINCIPAL NAO LIGOU NO MODO DE PISCAR ";
334:
335: //    #Reply = ASK "REPETIR RECOLHA DE CORES DOS SENSORES?", TYPE = YESNO;

336: //ENDWHILE;
337: LOG ">>> PASSOU";
338:
339: LOG "------ A TESTAR STROBE LENTO ------";
     //=============================================================================
340: SET_DIGITAL [ $CONF_LED_MODE0 ] = ON;
```

```
341: SET_DIGITAL [ $CONF_LED_MODE1 ] = ON;
342:
343: FOR #SenseIndex { 1..8 }
344:     TRANSMIT_SERIAL[1] "S:#SenseIndex#0;I:#STRB_INT_VAL:4x#";//, LOG=ON;
345:     RECEIVE_SERIAL[1] "U4:? U5:0 I:* OK", TIMEOUT = 1000 ELSE CONTINUE;//,
     LOG=ON;
346: ENDFOR;
347:
348: FOR #SenseIndex { 1..8 }
349:     TRANSMIT_SERIAL[1] "R";
350:     WAITMS 50;
351:     IF (#SenseIndex==1)
352:         WAITWHILE ( #TIMEOUT_STRB_S )
353:             TEST_DIGITAL[ $S1_INT] EXPECT == ON;
354:     ELIF (#SenseIndex==2)
355:        WAITWHILE ( #TIMEOUT_STRB_S )
356:             TEST_DIGITAL[ $S2_INT] EXPECT == ON;
357:     ELIF (#SenseIndex==3)
358:         WAITWHILE ( #TIMEOUT_STRB_S )
359:             TEST_DIGITAL[ $S3_INT] EXPECT == ON;
360:     ELIF (#SenseIndex==4)
361:         WAITWHILE ( #TIMEOUT_STRB_S )
362:             TEST_DIGITAL[ $S4_INT] EXPECT == ON;
363:     ELIF (#SenseIndex==5)
364:         WAITWHILE ( #TIMEOUT_STRB_S )
365:             TEST_DIGITAL[ $S5_INT] EXPECT == ON;
366:     ELIF (#SenseIndex==6)
367:         WAITWHILE ( #TIMEOUT_STRB_S )
368:             TEST_DIGITAL[ $S6_INT] EXPECT == ON;
369:     ELIF (#SenseIndex==7)
370:         WAITWHILE ( #TIMEOUT_STRB_S )
371:             TEST_DIGITAL[ $S7_INT] EXPECT == ON;
372:     ELIF (#SenseIndex==8)
373:         WAITWHILE ( #TIMEOUT_STRB_S )
374:             TEST_DIGITAL[ $S8_INT] EXPECT == ON;
375:     ENDIF;
376:
377:     IF ((#_WAITED_ < #TEMPO_MIN_STRB_S) OR (#_WAITED_ > #TEMPO_MAX_STRB_S))
378:         FAIL " TIMEOUT NO MODO DE STROBE LENTO (#_WAITED_#)", ABORT_ALL;
379:     ENDIF;
380: ENDFOR;
381: LOG ">>> PASSOU";
382:
383: LOG "------ A TESTAR STROBE RAPIDO ------";
     //=============================================================================
384: SET_DIGITAL [ $CONF_LED_MODE0 ] = ON;
385: SET_DIGITAL [ $CONF_LED_MODE1 ] = OFF;
386:
387: FOR #SenseIndex { 1..8 }
388:     TRANSMIT_SERIAL[1] "R";
389:     WAITMS 50;
390:     IF (#SenseIndex==1)
391:         WAITWHILE ( #TIMEOUT_STRB_F )
392:             TEST_DIGITAL[ $S1_INT] EXPECT == ON;
393:     ELIF (#SenseIndex==2)
394:         WAITWHILE ( #TIMEOUT_STRB_F )
395:             TEST_DIGITAL[ $S2_INT] EXPECT == ON;
396:     ELIF (#SenseIndex==3)
397:         WAITWHILE ( #TIMEOUT_STRB_F )
398:             TEST_DIGITAL[ $S3_INT] EXPECT == ON;
399:     ELIF (#SenseIndex==4)
400:         WAITWHILE ( #TIMEOUT_STRB_F )
401:             TEST_DIGITAL[ $S4_INT] EXPECT == ON;
402:     ELIF (#SenseIndex==5)
```

```
403:         WAITWHILE ( #TIMEOUT_STRB_F )
404:             TEST_DIGITAL[ $S5_INT] EXPECT == ON;
405:     ELIF (#SenseIndex==6)
406:        WAITWHILE ( #TIMEOUT_STRB_F )
407:             TEST_DIGITAL[ $S6_INT] EXPECT == ON;
408:     ELIF (#SenseIndex==7)
409:        WAITWHILE ( #TIMEOUT_STRB_F )
410:             TEST_DIGITAL[ $S7_INT] EXPECT == ON;
411:     ELIF (#SenseIndex==8)
412:        WAITWHILE ( #TIMEOUT_STRB_F )
413:             TEST_DIGITAL[ $S8_INT] EXPECT == ON;
414:     ENDIF;
415:
416:     IF ((#_WAITED_ < #TEMPO_MIN_STRB_F) OR (#_WAITED_ > #TEMPO_MAX_STRB_F))
417:         FAIL " TIMEOUT NO MODO DE STROBE RAPIDO (#_WAITED_#)", ABORT_ALL;
418:     ENDIF;
419: ENDFOR;
420: LOG ">>> PASSOU";
421:
422: LOG "------ A TESTAR FADE IN E FADE OUT ------";
     //=============================================================================
423: SET_DIGITAL [ $CONF_LED_MODE0 ] = OFF;
424: SET_DIGITAL [ $CONF_LED_MODE1 ] = ON;
425:
426: TRANSMIT_SERIAL[1] "S:10;I:#FADE_INT_VAL:4x#";//, LOG=ON;
427: RECEIVE_SERIAL[1] "U4:1 U5:0 I:* OK", TIMEOUT = 1000 ELSE CONTINUE;//, LOG=ON;
428: TRANSMIT_SERIAL[1] "R";
429: WAITMS 10;
430:
431: WAITWHILE ( #TIMEOUT_FADE )
432:    TEST_DIGITAL[ $S1_INT] EXPECT == ON;
433: IF (#_WAITED_ > #TEMPO_MAX_FADE)
434:    FAIL " TIMEOUT DL1 (#_WAITED_#)", ABORT_ALL;
435: ENDIF;
436:
437: VAR #OLD_TIME=#_SECOND_;
438: VAR #TIME_GONE;
439: VAR #LIGHT_VALUES[];
440: VAR #VAL_COUNT=0;
441: LOG " A RECOLHER VALORES DE LUMINOSIDADE ";//-----------------------------------
     ----------------------------------------------
442: WHILE (#TEMP_VAL == OFF)
443:    WAITMS 10;
444:    TRANSMIT_SERIAL[1] "S:10";//, LOG=ON;
445:    RECEIVE_SERIAL[1] "U4:1 U5:0 Color Temp: #TEMP_HEAT# K - Lux: #TEMP_LUX# -
     R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",TIMEOUT=1000;//, LOG=ON;
446:
447:    IF(#_SECOND_< #OLD_TIME)
448:       #TIME_GONE=60+#_SECOND_ - #OLD_TIME;
449:    ELSE
450:       #TIME_GONE=#_SECOND_ - #OLD_TIME;
451:    ENDIF;
452: //   LOG "#TIME_GONE#";
453:    IF (#TIME_GONE > #MAX_GONE)
454:       #TEMP_VAL=TEST_DIGITAL[ $S1_INT];
455:    ELIF (#TIME_GONE > #FADE_TEST_MAX)
456:       FAIL " TESTE DE FADE IN FADE OUT FALHOU",ABORT_ALL;
457:    ENDIF;
458:    #LIGHT_VALUES[#VAL_COUNT]=#TEMP_ALL;
459:    #VAL_COUNT=#VAL_COUNT+1;
460: ENDWHILE;
461:
462: VAR #NEXT_VAL;
463: VAR #RESULT;
```

```
464: #VAL_COUNT=#VAL_COUNT-2;
465: LOG " A VALIDAR VALORES RECOLHIDOS";//-----------------------------------------
     ---------------------------------------
466: FOR #INDEX {0..#VAL_COUNT}
467:    #NEXT_VAL=#INDEX+1;
468:    LOG " #INDEX# / #VAL_COUNT# #LIGHT_VALUES[#INDEX]# VS
     #LIGHT_VALUES[#NEXT_VAL]#";
469:    IF (#LIGHT_VALUES[#INDEX] < #LIGHT_VALUES[#NEXT_VAL])
470:        #RESULT=#LIGHT_VALUES[#NEXT_VAL] - #LIGHT_VALUES[#INDEX];
471:    ELIF (#LIGHT_VALUES[#INDEX] > #LIGHT_VALUES[#NEXT_VAL])
472:        #RESULT=#LIGHT_VALUES[#INDEX] - #LIGHT_VALUES[#NEXT_VAL];
473:    ELSE
474:        FAIL " VALORES NÃO PODEM SER IGUAIS", ABORT_ALL;
475:    ENDIF;
476:
477:    IF ( (#RESULT < #MIN_FADE) OR (#RESULT > #MAX_FADE) )
478:       FAIL " VALORES NAO ESTAO DENTRO DO ESPERADO (#RESULT#)", ABORT_ALL;
479:    ENDIF;
480: ENDFOR;
481: LOG ">>> PASSOU";
482:
483:
484:
485: LOG "  ------------------- TESTE DE COMPORTAMENTOS TERMINADO -------------------
     ";
```

B.1.6 Signal Test

```
 1: VAR #DEF_FREQ_MIN = 2000;
 2: VAR #DEF_FREQ_MAX = 2700;
 3: VAR #DEF_0_MIN = 0;
 4: VAR #DEF_0_MAX = 100;
 5: VAR #STEST=OFF;
 6: VAR #TTEST=OFF;
 7: VAR #ITEST=OFF;
 8: VAR #TIMEOUT=5000;
 9: VAR #MAX_TIME=4999;
10: VAR #TSTATE=OFF;
11:
12: VAR #STRB_INT_VAL=1500;
13: VAR #INF_R_INT_VAL=800;
14: VAR #INF_G_INT_VAL=1200;
15:
16: VAR #TEMP_B;
17: VAR #TEMP_G;
18: VAR #TEMP_R;
19: VAR #TEMP_HEAT;
20: VAR #TEMP_LUX;
21: VAR #TEMP_ALL;
22:
23: VAR #B_MIN_OFF=0;
24: VAR #B_MAX_OFF=900;
25: VAR #B_MIN_ON=1000;
26: VAR #B_MAX_ON=2500;
27:
28: VAR #R_MIN_OFF=0;
29: VAR #R_MAX_OFF=500;
30: VAR #R_MIN_ON=700;
31: VAR #R_MAX_ON=2500;
32:
33: VAR #G_MIN_OFF=0;
34: VAR #G_MAX_OFF=850;
35: VAR #G_MIN_ON=850;
36: VAR #G_MAX_ON=2500;
37: VAR #VAL;
38:
39: LOG "  ------------------- A INICIAR TESTE DE SINAIS -------------------";
40:
41: //------------------- INITIAL CONDITIONS -------------------
42: CONFIG_SUPPLY [ 2 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
43: SET_SUPPLY [ 2 ] = ON;
44: CONFIG_SUPPLY [ 1 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
45: SET_SUPPLY [ 1 ] = ON;
46: //SET_DIGITAL [ $CONF_LED1_COLOR2 ] = ON;
47: //SET_DIGITAL [ $CONF_LED2_COLOR2 ] = ON;
48: SET_DIGITAL [ GROUP 5, BIT 1 .. 3 ] = ON;
49: SET_DIGITAL [ GROUP 5, BIT 13 .. 15 ] = OFF;
50: SET_DIGITAL [ $CONTROL_SUP ] = ON;
51: SET_DIGITAL [ $CONTROL_EN ] = ON;
52: WAITMS 5000;
53:
54: TRANSMIT_SERIAL[1] "L", LOG=ON;
55: RECEIVE_SERIAL[1] "1122334455667788", TIMEOUT = 5000 ELSE ABORT, LOG=ON;

56: LOG "------------------- A CONFIGURAR INTERRUPCOES -------------------";
57: TRANSMIT_SERIAL[1] "S:20;I:#STRB_INT_VAL:4x#";//, LOG=ON;
58: RECEIVE_SERIAL[1] "U4:2 U5:0 I:* OK", TIMEOUT = 1000 ELSE CONTINUE;//, LOG=ON;
59: TRANSMIT_SERIAL[1] "S:06;I:#INF_R_INT_VAL:4x#";//, LOG=ON;
60: RECEIVE_SERIAL[1] "U4:0 U5:6 I:* OK", TIMEOUT = 1000 ELSE CONTINUE;//, LOG=ON;
61: TRANSMIT_SERIAL[1] "S:03;I:#INF_G_INT_VAL:4x#";//, LOG=ON;
62: RECEIVE_SERIAL[1] "U4:0 U5:3 I:* OK", TIMEOUT = 1000 ELSE CONTINUE;//, LOG=ON;
63:
```

```
 64:
 65:
 66: //------------------- TEST CODE ------------------------------------------------
     ------------------------------------------------------------------------------
     ----------
 67: FOR #TestIndex { 1..4 }
 68:     LOG "";
 69:     IF ( #TestIndex == 0 )
 70:         LOG " --- CONTROL_STRB OFF; CONTROL_TRIGGER OFF; CONTROL_INF OFF;
     CONTROL_TAMPER OFF ---";
 71:     ELIF ( #TestIndex == 1 )
 72:         LOG " --- CONTROL_STRB ON; CONTROL_TRIGGER OFF; CONTROL_INF OFF;
     CONTROL_TAMPER OFF ---";
 73:     ELIF ( #TestIndex == 2 )
 74:         LOG " --- CONTROL_STRB OFF; CONTROL_TRIGGER ON; CONTROL_INF OFF;
     CONTROL_TAMPER OFF ---";
 75:     ELIF ( #TestIndex == 3 )
 76:         LOG " --- CONTROL_STRB OFF; CONTROL_TRIGGER OFF; CONTROL_INF ON;
     CONTROL_TAMPER OFF ---";
 77:     ELIF ( #TestIndex == 4 )
 78:         LOG " --- CONTROL_STRB OFF; CONTROL_TRIGGER OFF; CONTROL_INF OFF;
     CONTROL_TAMPER ON ---";
 79: SET_DIGITAL [ GROUP 5, BIT 13 .. 15 ] = ON;
 80:     ENDIF;
 81:
 82:     //------------------------------------------------------------------------
     ------------------------------------------------------------
 83:
 84:     IF ( #TestIndex == 1 )  //------------------- VALIDATES STRB (STRB ON) -----
     --------------
 85:         SET_DIGITAL [ $CONF_STRB ] = ON;
 86:     ELSE
 87:         SET_DIGITAL [ $CONF_STRB ] = OFF;
 88:     ENDIF;
 89:     IF ( #TestIndex == 2 )  //------------------- VALIDATES TRIG (TRIG ON) -----
     --------------
 90:         SET_DIGITAL [ $CONF_TRIGGER ] = ON;
 91:     ELSE
 92:         SET_DIGITAL [ $CONF_TRIGGER ] = OFF;
 93:     ENDIF;
 94:     IF ( #TestIndex == 3 )  //------------------- VALIDATES INF (INF ON) -------
     -----------
 95:         SET_DIGITAL [ $CONF_INF ] = ON;
 96:     ELSE
 97:         SET_DIGITAL [ $CONF_INF ] = OFF;
 98:     ENDIF;
 99:     IF ( #TestIndex == 4 )  //------------------- VALIDATES TAMP (TAMP ON) -----
     --------------
100:         SET_DIGITAL [ $CONF_TAMP] = ON;
101:     ELSE
102:         SET_DIGITAL [ $CONF_TAMP] = OFF;
103:     ENDIF;
104:
105:     //------------------------------------------------------------------------
     ------------------------------------------------------------
106:     IF ( #TestIndex == 4 )
107:     SET_DIGITAL [ $CONTROL_SUP ] = OFF;
108:     WAITMS 100;
109:     SET_DIGITAL [ $CONTROL_SUP ] = ON;
110:     WAITMS 500;
111:
112:     ASK "MANTENHA PRESSIONADO O BOTAO I4 NO PCB", TYPE=OK;
113:         TEST_DIGITAL[ $CONTROL_TAMP_IN ] EXPECT (== OFF) ELSE ABORT_ALL, "
     BOTAO TAMP OU NAO FOI CLICADO OU NAO FOI DETECTADO      :FALHOU";
```

```
114:         LOG "  BOTAO TAMP DETECTADO >>> PASSOU";
115:     CONFIG_COUNTER [ 1 ] MODE = FREQUENCY;
116:     WAITMS 1000;
117:         #VAL = TEST_COUNTER [ 1] ;
118:         IF ((#VAL < #DEF_0_MIN) OR (#VAL > #DEF_0_MAX))
119:             FAIL "  TAMP RESPONDEU A ON QUANDO NAO DEVIA RESPONDER
     :FALHOU (#VAL# Hz) [MIN = #DEF_FREQ_MIN#, MAX =#DEF_FREQ_MAX#]", ABORT_ALL;
120:         ENDIF;
121:         LOG "  TAMP ON RESPONDE >>> PASSOU";
122:
123:         ASK "LARGUE O BOTAO I4"         , TYPE=OK;
124:         TEST_DIGITAL[ $CONTROL_TAMP_IN ] EXPECT (== ON) ELSE ABORT_ALL, "
     BOTAO TAMP OU NAO FOI LARGADO OU TEM CURTO              :FALHOU";
125:     WAITMS 1000;
126:     CONFIG_COUNTER [ 1 ] MODE = FREQUENCY;
127:     WAITMS 1000;
128:         #VAL=TEST_COUNTER [ 1] ;
129:         IF ((#VAL < #DEF_FREQ_MIN) OR (#VAL > #DEF_FREQ_MAX))
130:             FAIL"  TAMP NAO RESPONDEU A ON QUANDO DEVIA RESPONDER
     :FALHOU (#VAL# Hz) [MIN = #DEF_0_MIN#, MAX =#DEF_0_MAX#]", ABORT_ALL;
131:         ENDIF;
132:         LOG "  TAMP OFF SEM RESPOSTA  >>> PASSOU";
133:     ELSE
134:     FOR #TestVal { 1..3 }
135:         IF ( #TestVal == 1 )
136:         IF ( #TestIndex == 1 )
137:                 #STEST=ON;#TTEST=OFF;#ITEST=OFF;
138:             ELIF ( #TestIndex == 2 )
139:                 #STEST=OFF;#TTEST=OFF;#ITEST=ON;
140:             ELIF ( #TestIndex == 3 )
141:             #STEST=OFF;#TTEST=ON;#ITEST=OFF;
142:             #TSTATE=ON;
143:             ENDIF;
144:         IF ( #TestVal == #TestIndex  )
145:             LOG "  SINAIS STRB=ON, TRIG=ON, INF=ON";
146:         ELSE
147:             LOG "  SINAIS STRB=ON, TRIG=OFF, INF=OFF";
148:         ENDIF;
149:                 SET_DIGITAL [ $CONTROL_STRB ] = ON;
150:         SET_DIGITAL [ $CONTROL_TRIG ] = OFF;
151:         SET_DIGITAL [ $CONTROL_INF ] = OFF;
152:     ELIF ( #TestVal == 2 )
153:         IF ( #TestIndex == 1 )
154:             #STEST=OFF;#TTEST=OFF;#ITEST=ON;
155:         ELIF ( #TestIndex == 2 )
156:             #STEST=OFF;#TTEST=ON;#ITEST=OFF;
157:             #TSTATE=OFF;
158:         ELIF ( #TestIndex == 3 )
159:             #STEST=ON;#TTEST=OFF;#ITEST=OFF;
160:         ENDIF;
161:         IF ( #TestVal == #TestIndex  )
162:             LOG "  SINAIS STRB=ON, TRIG=ON, INF=ON";
163:         ELSE
164:             LOG "  SINAIS STRB=OFF, TRIG=ON, INF=OFF";
165:         ENDIF;
166:         SET_DIGITAL [ $CONTROL_STRB ] = OFF;
167:         SET_DIGITAL [ $CONTROL_TRIG ] = ON;
168:         SET_DIGITAL [ $CONTROL_INF ] = OFF;
169:     ELIF ( #TestVal == 3 )
170:         IF ( #TestIndex == 1 )
171:             #STEST=OFF;#TTEST=ON;#ITEST=OFF;
172:             #TSTATE=ON;
173:         ELIF ( #TestIndex == 2 )
174:             #STEST=ON;#TTEST=OFF;#ITEST=OFF;
```

```
175:         ELIF ( #TestIndex == 3 )
176:             #STEST=OFF;#TTEST=OFF;#ITEST=ON;
177:         ENDIF;
178:         IF ( #TestVal == #TestIndex  )
179:             LOG "  SINAIS STRB=ON, TRIG=ON, INF=ON";
180:         ELSE
181:             LOG "  SINAIS STRB=OFF, TRIG=OFF, INF=ON";
182:         ENDIF;
183:         SET_DIGITAL [ $CONTROL_STRB ] = OFF;
184:         SET_DIGITAL [ $CONTROL_TRIG ] = OFF;
185:         SET_DIGITAL [ $CONTROL_INF ] = ON;
186:     ENDIF;
187:     IF ( #TestVal == #TestIndex  )
188:             SET_DIGITAL [ $CONTROL_STRB ] = ON;
189:         SET_DIGITAL [ $CONTROL_TRIG ] = ON;
190:         SET_DIGITAL [ $CONTROL_INF ] = ON;
191:     ENDIF;
192:     SET_DIGITAL [ $CONTROL_SUP ] = OFF;
193:     WAITMS 100;
194:     SET_DIGITAL [ $CONTROL_SUP ] = ON;
195:     WAITMS 500;
196:
197:     TRANSMIT_SERIAL[1] "R";
198:     WAITMS 50;
199:     IF ( #STEST == ON )//-----------------------------------------------------
    ------------------------------------------------------------------------------
    ------------------------------
200:             //TEST_DIGITAL [ $S2_INT ] EXPECT (== OFF) ELSE ABORT_ALL, "  STRB
    NAO RESPONDEU QUANDO DEVIA RESPONDER   :FALHOU";
201:         WAITWHILE ( #TIMEOUT )
202:                 TEST_DIGITAL [ $S2_INT ] EXPECT == ON;
203:             IF (#_WAITED_ > #MAX_TIME)
204:                 FAIL " STRB NAO RESPONDEU QUANDO DEVIA RESPONDER   :FALHOU",
    ABORT_ALL;
205:         ENDIF;
206:         TRANSMIT_SERIAL[1] "S:20";//, LOG=ON;
207:         RECEIVE_SERIAL[1] "U4:2 U5:0 Color Temp: #TEMP_HEAT# K - Lux:
    #TEMP_LUX# - R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",
    TIMEOUT=1000;//, LOG=ON;
208:         IF ((#TEMP_R > #R_MAX_OFF) AND (#TEMP_B < #B_MIN_ON))
209:             FAIL " STRB NAO LIGOU AZUL QUANDO DEVIA (#TEMP_B# #TEMP_R#)
    :FALHOU", CONTINUE;
210:         ENDIF;
211:             LOG "   -- STRB RESPONDE >>> PASSOU";
212:     ELSE
213:         TRANSMIT_SERIAL[1] "S:20";//, LOG=ON;
214:         RECEIVE_SERIAL[1] "U4:2 U5:0 Color Temp: #TEMP_HEAT# K - Lux:
    #TEMP_LUX# - R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",
    TIMEOUT=1000;//, LOG=ON;
215:         IF (#TEMP_B > #B_MIN_ON)
216:                 FAIL " STRB RESPONDEU        :FALHOU", CONTINUE;
217:         ENDIF;
218:             //TEST_DIGITAL [ $S2_INT ] EXPECT (== ON) ELSE ABORT_ALL, "  STRB
    RESPONDEU  :FALHOU";
219:             LOG "   -- STRB SEM RESPOSTA >>> PASSOU";
220:     ENDIF;
221:
222:     CONFIG_COUNTER [ 1 ] MODE = FREQUENCY;
223:     WAITMS 1000;
224:     #VAL=TEST_COUNTER [ 1 ];
225:     SET_DIGITAL [ $CONTROL_TRIG ] = #TSTATE;
226:     IF ( #TTEST == ON )//-----------------------------------------------------
    ------------------------------------------------------------------------------
    ------------------------------
```

```
227:                 IF ((#VAL < #DEF_FREQ_MIN) OR (#VAL > #DEF_FREQ_MAX))
228:                 FAIL "  TRIG NAO RESPONDEU   :FALHOU (#VAL# Hz) [MIN =
    #DEF_FREQ_MIN#, MAX =#DEF_FREQ_MAX#]", ABORT_ALL;
229:             ENDIF;
230:             LOG "   -- TRIG RESPONDE >>> PASSOU";
231:     ELSE
232:                 IF ((#VAL < #DEF_0_MIN) OR (#VAL > #DEF_0_MAX))
233:                 FAIL "  TRIG RESPONDEU   :FALHOU (#VAL# Hz) [MIN = #DEF_0_MIN#,
    MAX =#DEF_0_MAX#]", ABORT_ALL;
234:             ENDIF;
235:             LOG "   -- TRIG SEM RESPOSTA >>> PASSOU";
236:     ENDIF;
237:
238:     TRANSMIT_SERIAL[1] "R";
239:     WAITMS 50;
240:     IF ( #ITEST == ON )//-------------------------------------------------------
    ------------------------------------------------------------------------------
    ------------------------------
241:         WAITWHILE ( #TIMEOUT )
242:                 TEST_DIGITAL [ $S14_INT ] EXPECT == ON;
243:             IF (#_WAITED_ > #MAX_TIME)
244:                 FAIL " INF NAO RESPONDEU QUANDO DEVIA         :FALHOU",
    ABORT_ALL;
245:         ENDIF;
246:             //TEST_DIGITAL [ $S14_INT ] EXPECT (== OFF) ELSE ABORT_ALL, "  INF
    NAO RESPONDEU QUANDO DEVIA RESPONDER          :FALHOU";
247:         TRANSMIT_SERIAL[1] "S:06";//, LOG=ON;
248:         RECEIVE_SERIAL[1] "U4:0 U5:6 Color Temp: #TEMP_HEAT# K - Lux:
    #TEMP_LUX# - R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",
    TIMEOUT=1000;//, LOG=ON;
249:
250:             IF ((#TEMP_R < #R_MIN_ON) AND (#TEMP_G > #G_MAX_OFF))
251:                     FAIL " INF NAO LIGOU VERMELHO QUANDO DEVIA (#TEMP_R#
    #TEMP_G#)         :FALHOU", CONTINUE;
252:         ENDIF;
253:
254:         IF ( #TestIndex == 3 )
255:             SET_DIGITAL [ $CONTROL_INF ] = OFF;
256:             WAITMS 500;
257:                 TRANSMIT_SERIAL[1] "R";
258:                 WAITMS 50;
259:             WAITWHILE ( #TIMEOUT )
260:                     TEST_DIGITAL [ $S11_INT ] EXPECT == ON;
261:                 IF (#_WAITED_ > #MAX_TIME)
262:                     FAIL " INF NAO RESPONDEU QUANDO DEVIA         :FALHOU",
    ABORT_ALL;
263:             ENDIF;
264:                 //TEST_DIGITAL [ $S11_INT ] EXPECT (== OFF) ELSE ABORT_ALL, "
    INF NAO RESPONDEU QUANDO DEVIA RESPONDER          :FALHOU";
265:             //VERIFICAR QUE FICA VERDE
266:             TRANSMIT_SERIAL[1] "S:03";//, LOG=ON;
267:             RECEIVE_SERIAL[1] "U4:0 U5:3 Color Temp: #TEMP_HEAT# K - Lux:
    #TEMP_LUX# - R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",
    TIMEOUT=1000;//, LOG=ON;
268:
269:             IF ((#TEMP_R > #R_MAX_OFF) AND (#TEMP_G < #G_MIN_ON))
270:                 FAIL " INF NAO LIGOU VERDE QUANDO DEVIA (#TEMP_G# #TEMP_R#)
       :FALHOU", CONTINUE;
271:             ENDIF;
272:         ENDIF;
273:             LOG "   -- INF RESPONDEU >>> PASSOU";
274:         SET_DIGITAL [ $CONTROL_SUP ] = OFF;
275:         WAITMS 100;
276:         SET_DIGITAL [ $CONTROL_SUP ] = ON;
```

```
277:         WAITMS 100;
278:         ELSE
279:         TRANSMIT_SERIAL[1] "S:06";//, LOG=ON;
280:         RECEIVE_SERIAL[1] "U4:0 U5:6 Color Temp: #TEMP_HEAT# K - Lux:
     #TEMP_LUX# - R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",
     TIMEOUT=1000;//, LOG=ON;
281:
282:             IF (#TEMP_R > #R_MIN_ON)
283:                     FAIL " INF RESPONDEU QUANDO NAO DEVIA         :FALHOU
     (#TEMP_R#)", CONTINUE;
284:         ENDIF;
285:             LOG "   -- INF SEM RESPOSTA >>> PASSOU";
286:     ENDIF;
287:     ENDFOR;
288:
289:         SET_DIGITAL [ $CONTROL_TAMP_OUT ] = ON;//-----------------------------
     ------------------------------------------------------------------------
     ------------------------------------------------------
290:     CONFIG_COUNTER [ 1 ] MODE = FREQUENCY;
291:     WAITMS 1000;
292:         #VAL=TEST_COUNTER [ 1 ] ;
293:         IF ((#VAL < #DEF_0_MIN) OR (#VAL > #DEF_0_MAX))
294:             FAIL "  TAMP RESPONDEU A ON QUANDO NAO DEVIA RESPONDER
     :FALHOU (#VAL# Hz) [MIN = #DEF_0_MIN#, MAX =#DEF_0_MAX#]", ABORT_ALL;
295:         ENDIF;
296:         LOG "   -- TAMP ON SEM RESPOSTA >>> PASSOU";
297:
298:         SET_DIGITAL [ $CONTROL_TAMP_OUT ] = OFF;//----------------------------
     ------------------------------------------------------------------------
     ------------------------------------------------------
299:     CONFIG_COUNTER [ 1 ] MODE = FREQUENCY;
300:     WAITMS 1000;
301:         #VAL=TEST_COUNTER [ 1 ] ;
302:         IF ((#VAL < #DEF_0_MIN) OR (#VAL > #DEF_0_MAX))
303:             FAIL"  TAMP RESPONDEU A OFF QUANDO NAO DEVIA RESPONDER
     :FALHOU (#VAL# Hz) [MIN = #DEF_0_MIN#, MAX =#DEF_0_MAX#]", ABORT_ALL;
304:         ENDIF;
305:         LOG "   -- TAMP OFF SEM RESPOSTA >>> PASSOU";
306: ENDIF;
307: ENDFOR;
308:
309:
310: SET_DIGITAL [ $CONF_TAMP] = OFF;
311: LOG "  ------------------- TESTE DE SINAIS BEM SUCEDIDO -------------------";
312:
313:
```

B.1.7 I2C Validation Test

```
 1: VAR #RESULT;
 2: VAR #INT_THR_VAL=0x0FFF;
 3: VAR #TEMP_VAL;
 4:
 5: LOG "  ------------------- A INICIAR TESTE DE SENSORES -------------------";
 6:
 7: //------------------- INITIAL CONDITIONS -------------------
 8: CONFIG_SERIAL [ 1 ] BAUDRATE = 115200, PARITY = NONE, TXMODE = MANUAL, RXMODE =
    MANUAL, RXTHRESHOLD = 128, RXTIMEOUT = 200, STXMODE = OFF, TERMINALMODE = CRLF;
 9: CONFIG_SUPPLY [ 2 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
10: SET_SUPPLY [ 2 ] = ON;
11: CONFIG_SUPPLY [ 1 ] VOLTAGE = 12.000, CURRENTLIMIT = 0.500;
12: SET_SUPPLY [ 1 ] = ON;
13: SET_DIGITAL [ $CONTROL_SUP ] = ON;
14: SET_DIGITAL [ $CONTROL_EN ] = ON;
15: WAITMS 5000;
16:
17: //CONFIGURA O LIMITE MÁXIMO DE LUMINOSIDADE PARA A INTERRUPÇÃO DE CADA SENSOR
18: LOG "  ------------------- A INICIAR ENVIO DE CONFIGURACOES -------------------";
19: TRANSMIT_SERIAL[1] "L", LOG=ON;
20: RECEIVE_SERIAL[1] "1122334455667788", TIMEOUT = 5000 ELSE ABORT, LOG=ON;
21: FOR #SenseIndex { 1..7 }
22:     LOG "  ------------------- A ENVIAR CONFIGURACOES PARA DL#SenseIndex# -------
    ------------";
23:     TRANSMIT_SERIAL[1] "S:#SenseIndex#0;I:#INT_THR_VAL:4x#", LOG=ON;
24:     RECEIVE_SERIAL[1] "U4:? U5:0 I:* OK", TIMEOUT = 1000 ELSE CONTINUE, LOG=ON;
25:     TRANSMIT_SERIAL[1] "S:0#SenseIndex#;I:#INT_THR_VAL:4x#", LOG=ON;
26:     RECEIVE_SERIAL[1] "U4:0 U5:? I:* OK", TIMEOUT = 1000 ELSE CONTINUE, LOG=ON;
27: ENDFOR;
28:
29: //------------------- TEST CODE ----------------------------------------------
    ------------------------------------------------------------------------------
    --------
30:
31: VAR #DL_NAME;
32: VAR #TEMP_B;
33: VAR #TEMP_G;
34: VAR #TEMP_R;
35: VAR #TEMP_HEAT;
36: VAR #TEMP_LUX;
37: VAR #TEMP_ALL;
38: VAR #Reply = YES;
39: WHILE (#Reply == YES)
40:     LOG "";
41:     LOG "  ------------------- A INICIAR TESTE DE CORES -------------------";
42:
43:     FOR #SenseIndex { 1..8 }
44:     //LOG "";
45:     //LOG "  ------------------- TESTE DO SENSOR PRIN #SenseIndex# --------------
    -----";
46:
47:
48:         TRANSMIT_SERIAL[1] "S:#SenseIndex#0";//, LOG=ON;
49:         RECEIVE_SERIAL[1] "U4:? U5:0 Color Temp: #TEMP_HEAT# K - Lux: #TEMP_LUX#
    - R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",TIMEOUT=1000;//, LOG=ON;
50:
51:         LOG " DL#SenseIndex# PRINCIPAL - LUZ R: #TEMP_R# G: #TEMP_G# B: #TEMP_B#
    ";
52:
53:     //LOG "";
54:     //LOG "  ------------------- TESTE DO SENSOR SEC #SenseIndex# ---------------
    ----";
55:
56:         TRANSMIT_SERIAL[1] "S:0#SenseIndex#";//, LOG=ON;
```

```
57:         RECEIVE_SERIAL[1] "U4:0 U5:? Color Temp: #TEMP_HEAT# K - Lux: #TEMP_LUX#
    - R: #TEMP_R#; G: #TEMP_G#; B: #TEMP_B#; C: #TEMP_ALL#;",TIMEOUT=1000;//, LOG=ON;
58:
59:         LOG " DL#SenseIndex# SECUNDARIA - LUZ R: #TEMP_R# G: #TEMP_G# B:
    #TEMP_B# ";
60:     ENDFOR;
61:     TRANSMIT_SERIAL[1] "S:00";
62:     LOG "  ------------------- FIM TESTE DE CORES --------------------";
63:     LOG "";
64:
65:     #Reply = ASK "REPETIR RECOLHA DE CORES DOS SENSORES?", TYPE = YESNO;

66: ENDWHILE;
67: LOG "  ------------------- TESTE DE I2C TERMINADO --------------------";
68:
```

B.2 Seeeduino XIAO Firmware

```
 1: #include "Adafruit_TCS34725.h"
 2: #include <Wire.h>
 3: #define LF 0x0D
 4: #define TCAADDR 0x70
 5: #define TCBADDR 0x71
 6:
 7: char incomingByte[100];
 8: int RecCnt = 0;
 9: int fl = 0;
10:
11: Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_154MS,
    TCS34725_GAIN_1X);
12:
13: void tcaselect(uint8_t i) {
14:   if (i > 8) return;
15:
16:   Wire.beginTransmission(TCAADDR);
17:   if (i == 0)
18:     Wire.write(0);
19:   else
20:     Wire.write(1 << (i - 1));
21:   Wire.endTransmission();
22: }
23:
24: void tcbselect(uint8_t i) {
25:   if (i > 8) return;
26:
27:   Wire.beginTransmission(TCBADDR);
28:   if (i == 0)
29:     Wire.write(0);
30:   else
31:     Wire.write(1 << (i - 1));
32:   Wire.endTransmission();
33: }
34:
35: void getRawData_noDelay(uint16_t *r, uint16_t *g, uint16_t *b, uint16_t *c)
36: {
37:   *c = tcs.read16(TCS34725_CDATAL);
38:   *r = tcs.read16(TCS34725_RDATAL);
39:   *g = tcs.read16(TCS34725_GDATAL);
40:   *b = tcs.read16(TCS34725_BDATAL);
41: }
42:
43: void setup() {  // put your setup code here, to run once:
44:   while (!Serial1);
45:   delay(1000);
46:   Wire.begin();
47:   Serial1.begin(115200);
48: }
49:
50: void loop() {
51:   int i = 0;
52:   uint8_t rInt[2];
53:   char rChar[4];
54:   uint16_t intVal = 0;
55:   int8_t intFl = 0;
56:   uint16_t r, g, b, c, colorTemp, lux;
57:
58:   digitalWrite(LED_BUILTIN, HIGH);
59:   if (Serial1.available() > 0) {
60:     incomingByte[RecCnt] = Serial1.read();
61:
62:     if (incomingByte[RecCnt] == LF)
63:       fl = 1;
```

```
64:     RecCnt++;
65:   }
66:   else {
67:     return;
68:   }
69:
70:
71:   if (fl == 1)  {
72:     digitalWrite(LED_BUILTIN, LOW);
73:     fl = 0;
74:     /*Serial1.println("Started analysis");
75:       for (i = 0; i <= RecCnt; i++)
76:       Serial1.print(incomingByte[i]);
77:       Serial1.println();*/
78:     for (i = 0; i <= RecCnt; i++) {
79:       if (incomingByte[i] == 'S') {//-----------------------------------------
    ------------------------------------------------------------------------------
80:         /*Serial1.println("Reading sensor value...");
81:           Serial1.println();*/
82:         if (i + 3 <= RecCnt) {
83:           rInt[0] = (int)incomingByte[i + 2] - 48; //converts the char to int
    and subtracts the ascii value for 0 (as char 0 = int 48)
84:           rInt[1] = (int)incomingByte[i + 3] - 48;
85:
86:           tcaselect(rInt[0]);
87:           tcbselect(rInt[1]);
88:           tcs.clearInterrupt();
89:           Serial1.print("U4:");
90:           Serial1.print(rInt[0]);//, DEC);
91:           Serial1.print(" U5:");
92:           Serial1.print(rInt[1]);//, DEC);
93:         }
94:       }
95:       else if (incomingByte[i] == 'I') {//-------------------------------------
    ------------------------------------------------------------------------------
    ----
96:         /*Serial1.println(" Setting sensor interrupt upper threshold...");
97:           Serial1.println();*/
98:         if (i + 6 <= RecCnt) {
99:           rChar[0] = incomingByte[i + 2];
100:          rChar[1] = incomingByte[i + 3];
101:          rChar[2] = incomingByte[i + 4];
102:          rChar[3] = incomingByte[i + 5];
103:
104:          intVal = (int)strtol(rChar, NULL, 16);
105:          intFl = 1;
106:          Serial1.print(" I:");
107:          Serial1.print(intVal);//, DEC);*/
108:        }
109:      }
110:      else if (incomingByte[i] == 'L') {//-------------------------------------
    ------------------------------------------------------------------------------
    ----
111:        /*Serial1.println("Activating every sensor...");
112:          Serial1.println();*/
113:        for (int j = 1; j <= 8; j++) {
114:          tcaselect(j);
115:          if (tcs.begin())
116:            Serial1.print(j);
117:          tcaselect(0);
118:          delay(0.1);
119:          tcbselect(j);
120:          if (tcs.begin())
121:            Serial1.print(j);
```

```
122:           tcbselect(0);
123:           delay(0.1);
124:         }
125:
126:         memset(incomingByte, 0, sizeof(incomingByte));//resets all the
     incomingByte array values to 0
127:         /*Serial1.println();
128:           Serial1.println("All set!");*/
129:         RecCnt = 0;
130:
131:         Serial1.print("\r\n");
132:         return;
133:       }
134:       else if (incomingByte[i] == 'R') {//-----------------------------------
     ------------------------------------------------------------------------
     ----
135:         for (int j = 1; j <= 8; j++) {
136:           tcaselect(j);
137:           tcs.clearInterrupt();
138:           tcaselect(0);
139:           delay(0.1);
140:           tcbselect(j);
141:           tcs.clearInterrupt();
142:           tcbselect(0);
143:           delay(0.1);
144:         }
145:
146:         memset(incomingByte, 0, sizeof(incomingByte));//resets all the
     incomingByte array values to 0
147:         RecCnt = 0;
148:
149:         Serial1.print("\r\n");
150:         return;
151:       }
152:       else if (incomingByte[i] == 'N') {//-----------------------------------
     ------------------------------------------------------------------------
     ----
153:         Serial1.println("Running scan mode...");
154:         Serial1.println();
155:         for (uint8_t t = 1; t <= 8; t++) {
156:           tcaselect(t);
157:           Serial1.print("TCA Port #"); Serial1.print(t);
158:
159:           for (uint8_t addr = 0; addr <= 127; addr++) {
160:             if (addr == TCAADDR || addr == TCBADDR) continue;
161:
162:             Wire.beginTransmission(addr);
163:             if (!Wire.endTransmission()) {
164:               Serial1.print(" - Found I2C 0x");  Serial1.println(addr, HEX);
165:             }
166:           }
167:         }
168:         Wire.beginTransmission(TCAADDR);
169:         Wire.write(0);
170:         Wire.endTransmission();
171:         Serial1.println("");
172:
173:         for (uint8_t t = 1; t <= 8; t++) {
174:           tcbselect(t);
175:           Serial1.print("TCB Port #"); Serial1.print(t);
176:
177:           for (uint8_t addr = 0; addr <= 127; addr++) {
178:             if (addr == TCAADDR || addr == TCBADDR) continue;
179:
```

```
180:             Wire.beginTransmission(addr);
181:             if (!Wire.endTransmission()) {
182:               Serial1.print(" - Found I2C 0x");  Serial1.println(addr, HEX);
183:             }
184:           }
185:         }
186:         Wire.beginTransmission(TCBADDR);
187:         Wire.write(0);
188:         Wire.endTransmission();
189:
190:         memset(incomingByte, 0, sizeof(incomingByte));//resets all the
     incomingByte array values to 0
191:         Serial1.println();
192:         Serial1.println("Done!");
193:         RecCnt = 0;
194:         return;
195:       }
196:     }
197:
198:     memset(incomingByte, 0, sizeof(incomingByte));//resets all the incomingByte
     array values to 0
199:     RecCnt = 0;
200:
201:     if (rInt[0] == 0 && rInt[1] == 0)       return;
202:     if (intFl == 1)  {
203:       intFl = 0;
204:       tcs.write8(TCS34725_PERS, TCS34725_PERS_1_CYCLE);
205:       tcs.setIntLimits(0, intVal);
206:       tcs.setInterrupt(true);
207:       Serial1.print(" OK\r\n");
208:       return;
209:     }
210:
211:     //tcs.getRawData(&r, &g, &b, &c);
212:     getRawData_noDelay(&r, &g, &b, &c);
213:     colorTemp = tcs.calculateColorTemperature_dn40(r, g, b, c);
214:     lux = tcs.calculateLux(r, g, b);
215:
216:     Serial1.print(" Color Temp: "); Serial1.print(colorTemp, DEC);
     Serial1.print(" K - ");
217:     Serial1.print("Lux: "); Serial1.print(lux, DEC); Serial1.print(" - ");
218:     Serial1.print("R: "); Serial1.print(r, DEC); Serial1.print("; ");
219:     Serial1.print("G: "); Serial1.print(g, DEC); Serial1.print("; ");
220:     Serial1.print("B: "); Serial1.print(b, DEC); Serial1.print("; ");
221:     Serial1.print("C: "); Serial1.print(c, DEC); Serial1.print(";");
222:     Serial1.print("\r\n");
223:     //Serial1.println("OK");
224:   }
225: }
226:
```