design your future

DESIGN AND DEVELOPMENT OF AN AUTOMATED TEST BENCH FOR SONAR ELECTRONICS

VIVES Technology
Bachelor in electronics-ICT

Campus Kortrijk

Luján María

Vynckier Ilias

Academic year 2021-2022

vives katholieke hogeschool
associatie KU Leuven

DESIGN AND DEVELOPMENT OF AN AUTOMATED TEST BENCH FOR SONAR ELECTRONICS

VIVES Technology
Bachelor in electronics-ICT

Campus Kortrijk

Luján María

Vynckier Ilias

Academic year 2021-2022

Preface

This thesis named 'Design and Development of an Automated Test Bench for Sonar Electronics' has been written as the last milestone of our four months internship being part of the ECA Robotics engineering team.

As ECA Robotics Belgium is still a newly created branch of ECA Group, we have been able to watch a lot of the steps that had been and will be taken to make ECA Robotics a leading company among underwater military drone suppliers in Belgium. We were part of the engineering team, developing the first design of this test bench for Sonar Electronics. The remarkable part was that we were working there as if we were full-fledged employees. We can proudly reveal that the final design based on our work will be included in the production line in the near future.

The project was appealing for us from the beginning. The opportunity to work in a field related to the manufacturing and design of something as notable as underwater robots was one of the things that made us choose the company.

The thesis proposal was also attractive to us. We had quite some experience in microcontroller programming and software design on Linux, which are the main components of this project. The advantages of working for a yet not-so-big company, more employee-oriented, was something we had in mind too when deciding the environment where our bachelor thesis was going to be developed.

We want to remind the reader that, due to working for a private company and furthermore for one that works in cooperation with military institutions, some parts or details of the software and technology used in this project cannot be discussed with the general public. All information contained in this document has been reviewed and approved by the company in order to be published without constraints.

We both are very grateful to ECA Robotics for letting us be here these past months. The team was very welcoming, and we want to especially thank them for guiding us so well and sharing all the knowledge they could with us. Also, we want to strongly thank them for make us feel as an active part of the company and the team.

We would like to take this opportunity to express our gratitude to some people in our personal name.

I, Ilias, would like to thank my internship supervisors at ECA Robotics Belgium. Thank you Bastien Loisy, Bjorn Declercq, Guillaume Villee, Dominique Aernout and Jeroen Debonnet for the guidance during the fascinating internship period. Furthermore, I would like to thank my Vives mentor Tom Termote for his helpful feedback and guidance during the writing of our bachelor thesis. Thirdly, I would like to thank my fellow student, Maria. Thanks for the good teamwork, the pleasant internship period and the critical view of our internship project. Without you, our bachelor thesis would not have become what it is now! Furthermore, I would like to thank my sister, Esther, for the tips she gave me during the process of writing my bachelor thesis. I also wish to acknowledge all

the people who have helped me over the years. Finally, I would like to thank my parents for giving me the opportunity to follow this fascinating study programme.

I, María, in first place want to thank Ilias for being an amazing bachelor thesis partner, and to Tom Termote for all the help and for the faith he put on us and our work. Also, as said before, to all the ECA team that guided us so well through the development of this project.

As this is one of the final steps to get my diploma, I want also to acknowledge the people that have been with me all these years back in Spain:

I want to first thank my parents. I am who and where I am because of them and the infinite patience, care, and attention they have put in the education of their children. Not only academically, but in every other aspect of our lives.

Also, the rest of my family, for always supporting me and being the best example one can have. Specially my grandparents because a grandchild couldn't ask for better role models.

Thanks too to my high school friends that believed in me since day one, and never doubted I would finish this thing.

And finally, thanks to my lil-chuchoshot for being the ray of sunshine of my bachelor's degree and my life in general for the past five years.

Ilias & María

Abstract

This thesis aims to be a first approach for the development of a test bench capable of performing automatic tests on an electronic board and reporting back the results to the user.

The general project set sights on implementing this test bench in the production line of a sonar developed by ECA Robotics, that will be mounted into the underwater drone toolbox that they, in collaboration with Naval Group, will provide to the Belgian and Dutch navies for Mine Warfare Countermeasure purposes.

Specifically, the main targets of this thesis are to evaluate the feasibility of the idea and to test the concept of it. For that, a PC application and the software for a microcontroller was designed and developed, as well as the communication between them using a customized protocol.

After implementing this design - using a prototype - a functional automated testbench was built, which verified the concept that had to be attested.

Table of contents

List of figures

List of tables

Symbols and abbreviations used

AUV. Autonomous Underwater Vehicle

DMO. Defence Material Organization

FPGA. Field Programable Gate Array

GPS. Global Positioning System

HS. High Speed

HSE. High-speed External Oscillator

IDR. Input Data Register

InSAS. Interferometric Synthetic Aperture Sonar

LED. Light-emitting diode

MCM. Mine Countermeasures

MCWP. Marine Corps Warfighting Publication

MIW. Mine Warfare

MODER. Port Mode Register

NATO. North Atlantic Treaty Organization

NOAA. National Oceanic and Atmospheric Administration

NVIC. Nested Vector Interrupt Controller

NWP. Naval Warfare Publication

ODR. Output Data Register

OTG. On The Go

PCB. Printed Circuit Board

PHY. Physical Interface

RCC. Reset and Clock Control

RMS. Royal Mail Ship

ROV. Remotely Operated Vehicle

SAS. Synthetic Aperture Sonar

SONAR. Sound navigation and ranging

SSS. Side-scan sonar

TIM. Timer

UAV. Unmanned Aerial Vehicle

UMIS. Unmanned MCM Integrated System

UUT. Unit Under Test

UUV. Unmanned Underwater Vehicle

1 Introduction

The prevalence of naval mines in almost all seas poses a major threat to both commercial and military vessels. Nowadays, demining operations are carried out by specialised military teams, most notably those of the Belgian and Dutch navies.

Today, this dangerous task is aided by drones, which avoid endangering human lives, and go deep into minefields to identify and dispose the explosive devices safely.

The partnership formed by ECA Robotics Belgium and Naval Group aims to provide a next generation drone system for mine countermeasure missions for the Belgian and Dutch navies. This bachelor thesis is part of the development of one of these drones, specifically focused on its sonar and the electronics that compose it.

Motivation

As the project develops, more units of drones and sonars mounted on them are being produced after the prototypes' performance met the expectations. The number of units to be produced make efficient testing a crucial part of the production chain. Due to this, the company has decided to implement an automatic test bench for some of the electronics that are going to be used in the sonar, that are difficult to evaluate and to access once they are assembled.

An automatic test bench has two major advantages: first, the tests are done as efficient as they can be, without having to measure and check things by hand. Second, it avoids human errors by taking the operator out of the loop, relegating her or him to manage a user interface used to manage the tests and receive the results.

Objectives

The aim of this project is to program the microprocessor of the test bench to allow testing of all interfaces of the RX32 electronics card. The test bench will be used at the supplier's side of the electronics card and in the factory at Ostend. The test bench will be able to send back voltage information, send signals and "loop through" the GPIO's (Vynckier, 2022, p. 5).

More precisely, the specific targets of this thesis due to the proximity of deadlines, complexity of the final product and the unavailability of some crucial pieces (like the electronics to test) are reduced to do previous research about the feasibility of the idea, lay the foundations for the development of the test bench and mainly to **test the concept** of the project.

This thesis is therefore meant to be the groundwork for the development of the functional automated test bench, that will be manufactured with the collaboration of a subcontractor in the near future.

Development timeline

The project development (including this thesis) lasted a total of 17 weeks. The internship at ECA Robotics, where this project was created, covered a period of 14 of those weeks. The thesis and defence preparation took up the other 3 weeks, as it was established by VIVES University.

We can structure the internship period into three main parts:

- Weeks 1 to 5: All previous research, familiarization with the components, software and specifications, and initial documentation (scope and vision of the project, UML diagrams…) were made.
    - Week 5: A restructuration of the project was necessary. The maximum priority was moved from making a completely functional test bench to test the concept for further development.
- Weeks 6 to 13: Development phase. All software for the test bench was created relaying on the previous research and agreed documentation.
    - Week 9: The company organized a travel to the headquarters of ECA in Toulon. There, we had a meeting with the subcontractor who is going to develop part of the project, as we will explain later. This meant a little rescope of the project.
- Weeks 13 and 14: Test phase and construction of the prototype. The final product was tested using this prototype. All documentation and manuals for further keeping up with the project by the next developers was finished and explained to the team.

The timeline in the Annexes (pg. 70) provides more details about the steps taken to finish this project.

Structure of the document

In the next chapters we will discuss what an autonomous underwater vehicle, a sonar, an RX32, a test bench and a Nucleo-144 are. After that, we will explain the context of the project, go into the technical background, and give an overview of the time course of this project within our internship. At the end of the document, a clear conclusion is provided, along with the bibliography and annexes.

Overview by chapter:

- **Background**: in this chapter crucial terms for understanding the project will be discussed. The terminology that will be used, the detailed explanation of technologies that will be mentioned along this document and some historical contexts are the main points included in this part.
- **Context of the project**: this chapter aims to inform the reader about the scope of the project, locating it from the general context -naval mines countermeasures technology- to the specific target this thesis is focused on: the test bench for sonar electronics.

- **Technical explanation**: as the technical part of the project can be easily divided into a computer part (PC part) and an electronics-controlled part (Nucleo part), we will also divide this chapter in the exact same way. The chapter starts with a general overview and then goes through the software description following the data flow (PC→Communication→Nucleo). It includes the explanation of the development of the hardware too.
- **Timeline**: the purpose of this chapter is to locate the project in time. The reader will have a clear idea of how much time was dedicated to every part of the development and the sequence followed to build not only the software and hardware of the project, but all the documentation attached to it too.

2 Background

Autonomous Underwater Vehicle

Terminology

The Cambridge University Press Dictionary defines *drone* as “an aircraft that does not have a pilot but is controlled by someone on the ground, used especially for dropping bombs or for surveillance”.

According to this definition an underwater robot could never be called a drone, as it is not an aerial vehicle or aircraft. However, the term *drone* is often widely used in all fields (civilian *and* military) for designating an autonomous driverless vehicle, regardless of the physical environment it was designed for. As a matter of fact, the leading companies in this project use the term *underwater drone* in their press releases. Taking this into consideration, the term *drone* will be used for any unmanned vehicle reference from now on.

Along with that, we must be careful when naming the subtypes of this kind of vehicles. A wrong use of the terminology can lead to confusion with acronyms and definitions.

To avoid that, the specific nomenclature listed below will be always followed in this document, and the differences and classification of these terms will be explained in detail further in this paper

- UUV: Unmanned Underwater Vehicle (U.S. Department of Defence, 2016).
- AUV: Autonomous Underwater Vehicle. Not to be mistaken with Unmanned Aerial Vehicle or UAV.
- ROV: Remotely Operated Vehicle.

Definitions and history

To talk about autonomous underwater vehicles, we must first define what an unmanned underwater vehicle (UUV) is.

A UUV is a pilotless robot capable of operating underwater.

According to Siciliano & Khatib (2016, Chapter 25. Underwater Robots), nowadays most of commercially available UUVs are remotely operated through a cable coming from a mother vessel. These remotely operated vehicles, or ROVs, use this thread for power, control and sending information back from the sensors to the land or mothership operators.

On the other hand, the term autonomous underwater vehicle, or its more used acronym AUV, stands for submarine drones that can operate unthreaded without an on-board human operator.

In 1960, the US Navy founded an early ROV technology development. Since then, they have been used for a wide range of applications: from military operations to scientific missions, ocean mining and gas and oil industry. However, during the last two decades an increasing demand of a next

generation of ROVs appeared. Autonomous under water vehicles started to be developed in response to that demand (Siciliano & Khatib, 2016, Chapter 25. Underwater Robots).

Sonar

Definitions and terminology

Water is one of the most challenging environments for sensors. The light does not usually penetrate far in the water column, making optical sensing dependent on the turbidity level and therefore of limited use. Also, the electromagnetic wave propagation is extremely low range underwater, turning position sensing into a very difficult task too, as the GPS is not available (Siciliano & Khatib, 2016, Chapter 25.4 Underwater Actuators and Sensors).

Contrastingly, acoustic waves can travel long distances underwater without attenuating excessively. They can reach every depth in the oceans, from the deepest points in the Marianas Trench to the shallowest waters of the North Sea coast, just a few centimetres beneath the surface. This allows acoustic sensors to explore a range of more than 11 km deep, according to Blondel's Handbook of Sidescan Sonar (2009).

Acoustic sensors can collect three basic types of information: seafloor shape, seafloor reflectivity, and impedance changes in the sub-seafloor. All these types of acoustic data are provided by the sensors installed on platforms that, in combination, are referred to as sonar systems (Carton et al., 2017).

SONAR or Sound Navigation And Ranging hence is a technique for detecting and determining the distance and direction of objects located below the water surface by acoustic means (The Editors of Encyclopædia Britannica, 2019).

Despite being an acronym, *sonar* has become a word in its own right, and it is considered grammatically correct to write it lowercased.

Additional sonar-related definitions useful for understanding the following chapters are included below:

Hydrophone: underwater device that detects and records underwater sounds from all directions. Most of them are based on a special property of ceramic materials that produces a small electrical current when the device is exposed to water pressure changes (NOAA, 2021).

Beam: line of energy, radiation or particles sent in a particular direction. Also used as *ping*: a short high-pitched resonant sound, as of a bullet striking metal or a sonar echo (HarperCollins Publishers, n.d).

Transceiver: both transmitter and receiver device.

Bathymetry: acoustic data that represents seafloor shape (Carton et al., 2017, p. 92).

History

Some animal species have the capability of locating objects using sound waves. Well known examples of these animals are bats and dolphins. They do so by interpreting the echoes of the ultrasonic sounds they produce, using this skill to locate food and objects in their habitat. These kinds of animals have inspired humans to use similar technics.

According to Ainslie (2016, Chapter 1.1 WHAT IS SONAR?), the first electronic systems enabling detection and localisation were invented in the 20th century. The creation of these systems was motivated by the sinking of the RMS Titanic in 1912 and the First World War, incidents that drew attention to the lack of underwater object-localisation systems. The Second World War and the Cold War stimulated the further development of these systems into the high-performance sonars we have today.

Physical principles

Acoustic waves, propagation, and scattering

In his book, Blondel (2009) defines acoustic waves as the propagation of a pressure wave through an environment, characterized by its intensity, that is often measured in a logarithmic scale relative to the source (decibels or dB), its frequency and the length and type of the pulse.

The intensity is related to the changes in pressure and the frequency (f) is related to the wavelength(λ) by means of the velocity of the sound in the medium (c).

$$f = \frac{c}{\lambda}$$

In water, c ranges between 1,450m/s and 1,550m/s in seawater, and it is reliant on the salinity, pressure, and temperature of the water.

When acoustic waves are transmitted through a water column, they experience an attenuation, which depends on the travelled distance and the dissipation of the acoustic energy due to viscosity and chemical reactions in the medium. This decrease in intensity is much more important when acoustic waves encounter the seabed. This is used in the sonar imagery: more range means weaker acoustic returns.

Signal emission and reception

The beams transmitted by the sonar are projected both sides of the sonar. These beams are narrow along-track to acquire high resolution and wide across-track to improve the range of coverage.

Most of the energy will be reflected in the specular direction of the beam and only a small portion of it will be received back by the receivers of the sonar (backscatter). The information that can be extracted from this backscatter is the time offset between the transmission (ping) and the reception, that its directly proportional to the slant range. From the sonar height the ground range to the echo can be deduced, and phase shifts are used to measure the arrival angle of the beam.

Comparing the phase shift from two contiguous transducers some systems can also extract bathymetry. Also, the amplitude can give information about the nature of the imaged point (Blondel, 2009).

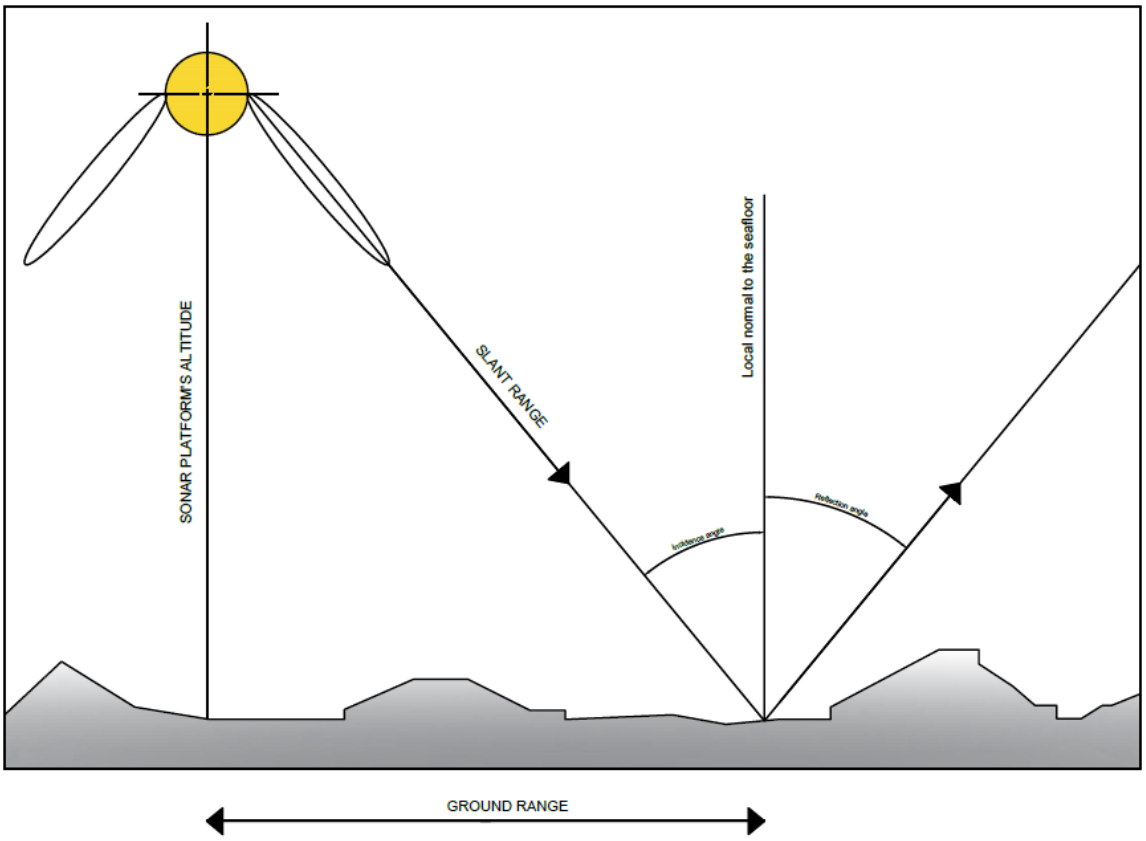


Figure 1: Adapted from Blondel, P. (2009a). Definitions of some parameters [Ilustration]. In The Handbook of Sidescan Sonar (p. 15).

Classification

A first distinction is made between three different systems: active sonar, passive sonar, and acoustic communication systems.

Active sonar generates pulses that are reflected by the object. The reflected pulses are then picked up and analysed to determine the distance, bearing and relative movement of the target (Sonar | Definition, Acronym, Uses, & Facts, 1998).

Passive sonar analyses and identifies these features as well, interpreting both direction and distance of the sound it has listened through its receiver. This sound may come from vessels, marine life, submarines…

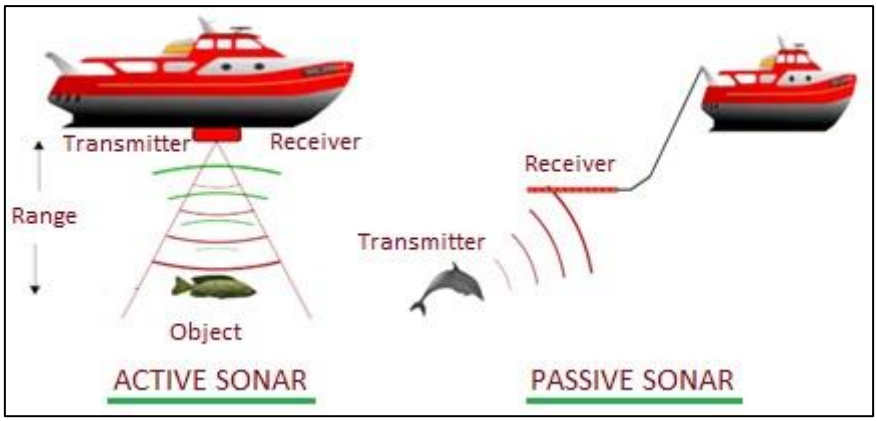


Figure 2 [Active SONAR vs Passive SONAR]. (n.d.). https://www.rfwireless-world.com/images/Active-SONAR-vs-Passive-SONAR.jpg

The third system, acoustic communication, requires a sound emitter and receiver on both sides of the link. This can be used to set up the communication of a submarine with a surface vessel.

Carton et al. (2017) make a second classification among the sonars that measure the seafloor reflectivity. This kind of measurement provides information about the texture, allowing to find objects on or right underneath the seabed as long as they have high acoustic contrast in relation to the surrounding environment.

Side-scan sonar (SSS)

These sonars project fan-shaped sound beams that reach an area that covers from beneath the projector to a point close to the horizon, called *nadir.* The projector transmits a ping, and the hydrophones will record back first the reflection from the nearest point, that will be the nadir in a flat bottom. The rest of the reflections will come from the closest to the furthest points and the strength of the signal will vary if an object with a different sound reflection, resulting into a strip of pixels. When the projector moves forward, it adds more strips creating a bidimensional image of the seafloor.

Synthetic aperture sonar (SAS)

Synthetic aperture sonars are an emerging technology that aim to provide a high-resolution swath imagery of the seabed. These sonars follow the same principle as the SSS, emitting sound beams to both sides of the platform (wider pulses in SAS), but they combine successive acoustic pings along the known track to increase the apparent resolution along this track (azimuth resolution). The resulting image has a resolution of cm in an over hundreds of meters range.

SAS faces a lot of challenges, like unexpected changes of heigh and trajectory of the platform that must be compensated by a precise control of the platform where the sonar is mounted. Powerful algorithms and signal processing is necessary for a good performance of the sonar, that is never carried by the surface vessel, but in a towed platform behind a survey vehicle or an AUV to minimize the motion variations and better control the host vehicle attitude.

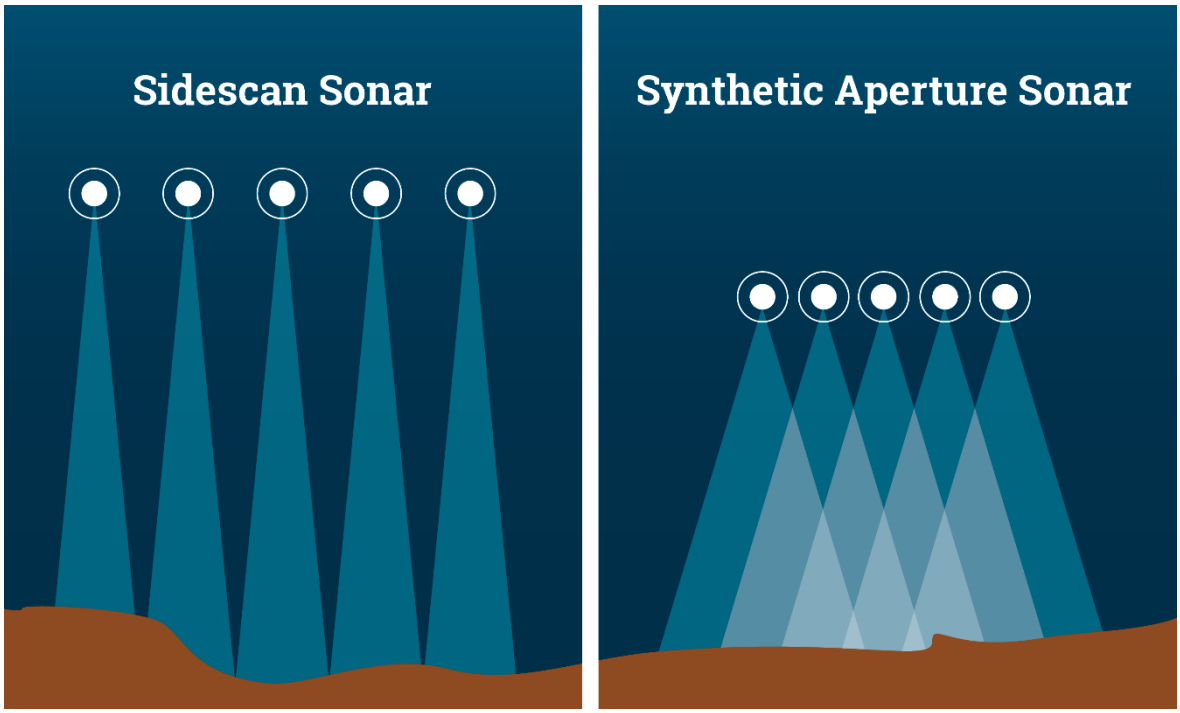


Figure 3: National Oceanic and Atmospheric Administration [NOAA]. (n.d.). *Side-scan sonar vs synthetic aperture sonar pulses* [Illustration]. National Oceanic and Atmospheric Administration (NOAA) Ocean Exploration. https://oceanexplorer.noaa.gov/technology/sonar/sas.html

The reason SAS in gaining popularity is mainly due to its versability. In the images acquired with these sonars the difference between soft and hard textures in the bottom of the seas can be observed. When these images are combined with depth measurement, an accurate bathymetry is obtained (interferometric synthetic aperture sonar). However, it has some disadvantages too. SAS is expensive to operate and requires very complicated processing of the data received. The trajectory the device follows must be pre-fixed to satisfy the assumptions for the calculations (National Oceanic and Atmospheric Administration [NOAA], n.d.-a).

RX32

The RX32 is a small board, designed and made by a subcontractor. This board contains an FPGA, some LEDs and connectors, among other electronics. Several of these boards are used in the pulser and receiver of the sonar unit of the UUV, where they represent a critical part of the project. Once these boards are installed in the sonar unit, it is a time-consuming process to replace them. It is therefore essential to test and validate this part before installing it.
In this project, the RX32 is plugged into the test bench using an interface card.

Test Bench

Terminology

A test bench or test bed is a term used by various disciplines to describe a platform or environment that is used for testing components, design, or models. A test bench is used to perform tests that produce transparent and reproducible results ("Testbed," 2022). There are two types of test benches, namely development test benches and end-of-line test benches. The test bench used in this thesis is an end-of-line test bench.

The overall purpose of this type of test bench is to check the function of newly developed test pieces. ECA Robotics Belgium chooses not only to check samples, for example a newly developed RX32, but to test every RX32. This can be seen as a general quality control ("Prüfstand," 2006).

Nucleo-144

The Nucleo-144 is a development board manufactured by STMicroelectronics. This board is part of the STM32 Nucleo family and with its 144 pins is the largest of the family. For this project, the Nucleo-H7A3Z1-Q, based on an Arm® Cortex®-M7 core was chosen. It was selected because this board integrates a USB-Serial converter and a programmer/debugger for the microcontroller. For this project, it was suggested to plug the Nucleo-H7A3Z1-Q directly into the test bench, to avoid extra manipulation of the board and wires.

About the manufacturer

STMicroelectronics is the result of a merger between the two semiconductor companies of the French and Italian governments. The merger took place in 1987 between the Italian company SGS Microelettronica and the French company Thomson Semiconducteurs (STMicroelectronics, 2022).

Features

All STM32 Nucleo-144 boards share the following common features:

- STM32 microcontroller in LQFP144 package
- 3 user LEDs
- 2 user and reset push-buttons
- 32.768 kHz crystal oscillator
- Board connectors: SWDST Zio expansion connector including ARDUINO® Uno V3ST morpho expansion connector
- Flexible power-supply options: ST-LINK, USB V_{BUS}, or external sources

- On-board ST-LINK debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port, and debug port
- Comprehensive free software libraries and examples available with the STM32Cube MCU Package
- Support of a wide choice of Integrated Development Environments (IDEs) including IAR Embedded Workbench®, MDK-ARM, and STM32CubeIDE

The following features are specific to this board:

- External or internal SMPS to generate V_{core} logic supply
- Ethernet compliant with IEEE-802.3-2002
- USB OTG full speed or device only
- Board connectors:USB with Micro-AB or USB Type-C®Ethernet RJ45
- Arm® Mbed Enabled™ compliant

(*NUCLEO-H723ZG*, n.d)

3 Context of the project

What is a naval mine?

According to the Dictionary of Military and Associated terms of the U.S. Department of Defence (2016), a mine in a naval warfare context is an explosive device laid in the water with the intention of damaging or sinking ships or of deterring shipping from entering an area. The Official NATO Terminology Database adds to this definition that the term does not include devices attached to the bottoms of ships or to harbour installations by personnel operating underwater, neither devices that explode immediately on expiration of a predetermined time after laying (NATO, n.d.-a).

Nowadays, the two principal types of naval mines present in seas are moored and bottom mines:

A moored mine is a "contact or influence[1]-operated mine of positive buoyancy held below the surface by a mooring attached to a sinker or anchor on the bottom" as is defined in the NATO Terminology Database. In the other hand, a bottom mine is a "mine with negative buoyancy which remains on the seabed" (NATO, n.d.-a).

Figure 4: ONI-37 & Naval History and Heritage Command. (n.d.). Floating Mine in the North Sea 123952 [Photograph]. Naval History and Heritage Command. https://www.history.navy.mil/

The history of naval mine warfare

The Naval History and Heritage Command (2021) dates the first attempt of use of naval mine warfare back to the American Revolution, in 1777, when part of the British fleet was stationed at the Delaware River off Philadelphia. David Bushnell discovered a gunpowder capable of exploding underwater and was authorized by General George Washington to use it to destroy some of the enemy ships. The device failed but the attempt brought attention from all over the world to this underwater exploding device.

However, in the International Security Department Workshop about International Law Applicable to Naval Mines (Chatham House & The Royal Institute of International Affairs, 2014) the early

[1] An influence mine is actuated by the effect of a target on some physical condition in the vicinity of the mine or on radiations emanating from the mine (NATO, n.d.-a).

precursors to sea mines are tracked back to the Ming dynasty, in the 16th century, where they were used to target pirates operating off the coasts of China.

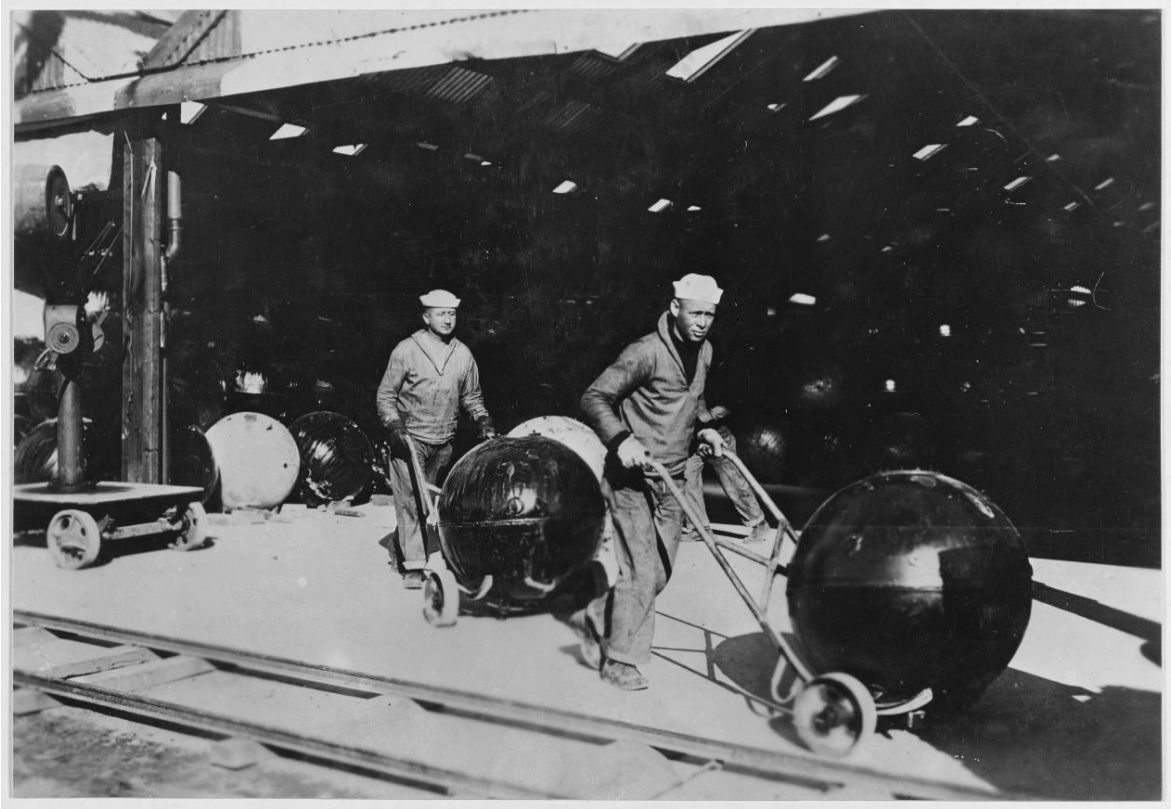


Figure 5: Naval History and Heritage Command & Signal Corps 43589. (n.d.). Handling Mine cases. NH 123955 [Photograph]. Naval History and Heritage Command. https://www.history.navy.mil/

Naval mines were used in the upcoming wars, from the U.S Civil War to both the World Wars. These devices were used to sink vessels, destroy submarines and to restrict fleets movement across the sea (Naval History and Heritage Command, 2021), but it was not until the Russo-Japanese War (1904-1905) that they were used extensively (Chatham House & The Royal Institute of International Affairs, 2014). Because of this conflict, those neutral countries with important surface fleets were deeply concerned about the potential destructive power of naval mine warfare on their own ships. Concurrently, mines were blocking merchant shipping free navigation, causing disruptions that made the commercial trading interests to have them banned extensively (Chatham House & The Royal Institute of International Affairs, 2014).

In 1899, through the personal initiative of Tsar Nicholas II of Russia, a conference was convened in The Hague with the main purpose of limiting armaments. The posterior Hague Conference of 1907 was much more focused on naval affairs after both civil and military concerns. As it was said in the International Security Department Workshop, summarized by Chatham House & The Royal Institute of International Affairs in 2014, the commercial trading interests wanted an outright ban of naval mines. However, the Royal Navy saw naval interests in them and only wanted them restricted. Therefore, the British opinion favoured tight restrictions on mining activity instead of a total ban.

Applicable regulations on naval mine warfare

According to the mine warfare publication of the U.S. Marine Corps and U.S. Navy (1996, pp. 1–7), “The Hague Convention (VIII) of 1907 probably had more of a legal impact on MIW than any other forum”. The document also collects the specific provisions laid down by this convention:

1. Armed, unanchored mines must have a maximum life of 1 hour.
2. Armed, anchored mines must become unarmed if they break free from their moorings.
3. Mines must be designed to become harmless should they miss their target.
4. It is illegal to mine solely against commercial shipping.
5. Neutral nations are not to be interfered with, and the safe transit of neutral shipping must be ensured.
6. Mines must be removed by the planting force at the conclusion of hostilities.

Furthermore, the Convention states that the mine-laying states are required to notify neutral states as to the dangers that exist to innocent shipping unless the mines laid are controlled mines that are used only to target military objectives. But, since the obligation to notify must be made as soon as military exigencies permit, the failure to notify immediately does not automatically constitute a violation of international law (Haines, 2014, "IV. E. Article 4" section).

In addition, even though the specific international laws set forth during the Hague Convention remain in effect today, as it is also said in the in the above-mentioned document that “they have not always been adhered to by all nations, and world events have seen major deviations from these principles”.

Reality of current underwater mine warfare threats

These days, naval mines still remain a danger for civilian and military ships. The lack of knowledge of the exact location and number of these mines make the countermeasure missions difficult and newly placed mines that have drifted away by the sea currents can become a danger outside the countries in conflict.

Sadly, there are plenty of examples of this problem today. The newspaper CE Noticias Financieras reports that “The Bulgarian government has warned about the presence of explosive mines drifting in the Black Sea that were allegedly washed up from the Ukrainian city of Odessa endangering maritime traffic in the area”. According to the Bulgarian Executive, they “were swept away by a storm and are floating in the Black Sea posing a threat to all vessels” (Content Engine LLC, 2022).

And this is not the only case across the Black Sea. The Ghana News Agency (GNA) also reports the detection of another naval mine off the coast of Kefken, in Istanbul, following the discovery of two other mines at the end of March (SyndiGate Media Inc, 2022).

That is why, even for countries that are not currently participants of an armed conflict, mine countermeasures (or MCM) have crucial importance among the naval operations conducted by military forces.

Nowadays Mine Countermeasures (MCM)

MCM can be offensive (proactive) or defensive (enabling). This project's scope lays on the defensive mine countermeasures, that can be also classified into active and passive. Passive defensive MCM includes all measures that reduce the effectiveness of the mine without physically removing it. Active measures remove the mine, by destroying it or neutralizing it (U.S. Marine Corps & U.S. Navy, 1996).

The two main subsets of active MCM are mine hunting and mine sweeping. According to the naval warfare publication of the U.S. Marine Corps and U.S. Navy (1996), mine hunting actions individually locate the mines for countermeasures to be taken, while in mine sweeping, all the mines in the sweep path are addressed at the same time.

In that document is also stated that the mine hunting process is done by detecting the mines for them to be later classified, located, identified using video camera and sonar, and neutralized. The mine disposal is done by using an explosive charge to cause the detonation of the mine.

For many European countries MCM has been a high priority, mainly because of the mines left over from the Cold War and the World Wars. Specifically, the Netherlands and Belgium constitute one of the strongest NATO allies, and it has been stated by the U.S. Marine Corps and the U.S. Navy (even from 1996) that they are perhaps "the best MCM school in the world".

Future of MCM: Belgium Naval & Robotics

In 2021, it was announced that the Ministry of Defence of the Netherlands contract from the Defence Material Organization (DMO), was awarded to the Naval Group and ECA Group consortium (Called Belgium Naval & Robotics). The project is meant for the supply of a drone system for mine hunting capable of replacing the Dutch Navy (Belgium Naval & Robotics, 2021).

In a 2018 press release from ECA Group, it is stated that the partnership created in 2018 "will bring together the skills of Naval, European leader in naval defence, responsible for designing, supplying, and maintaining military vessels carrying drones, with those of ECA Group, a robotics specialized Group with expertise in mine warfare developed over several decades, responsible for designing and providing resources that may be launched into minefields (drones, sonars, etc.)".

The collaboration, born 60 years ago, is extended to integrate UMIS^{TM} Toolbox into the Naval Group vessels for developing the next generation of MCM.

The mine-hunter drone system for BE/NL[2] Navy

The project aims to preform safe MCM operations from a distant vessel using unmanned systems. The mission starts from a Mine Countermeasures warship, that deploys a USV (Figure 6) into the mine zone, keeping human operators out of danger. The USV can launch an AUV (Figure 7) that tracks the mine locations using a powerful Synthetic Aperture Sonar (Figure 8) or a self-powered towed sonar (For further information refer to page 21), that uses the same technology to detect in real time, classify and map seabed and moored mines. For further identification, a ROV provided with a high-quality camera (Figure 9) gets closer to the mines, and after confirmation, another AUV (Figure 10) completes the neutralisation by exploding the mine (Belgium Naval & Robotics, 2020). A UAV is also included, mostly for communication purposes.

Figure 6: ECA Group. (n.d.). INSPECTOR 125 [Photograph]. ECA Group. https://www.ecagroup.com/en/solutions

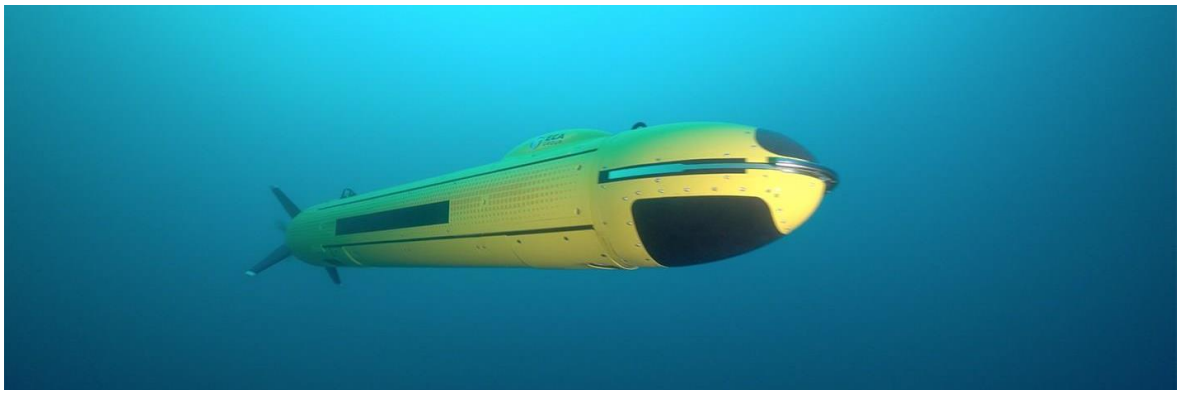


Figure 7: ECA Group. (n.d.). A18-M [Photograph]. ECA Group. https://www.ecagroup.com/en/solutions

[2] Belgium and Netherlands

Figure 8: ECA Group. (n.d.). Image obtained with SAS [Photograph]. ECA Group. https://www.ecagroup.com/en/solutions

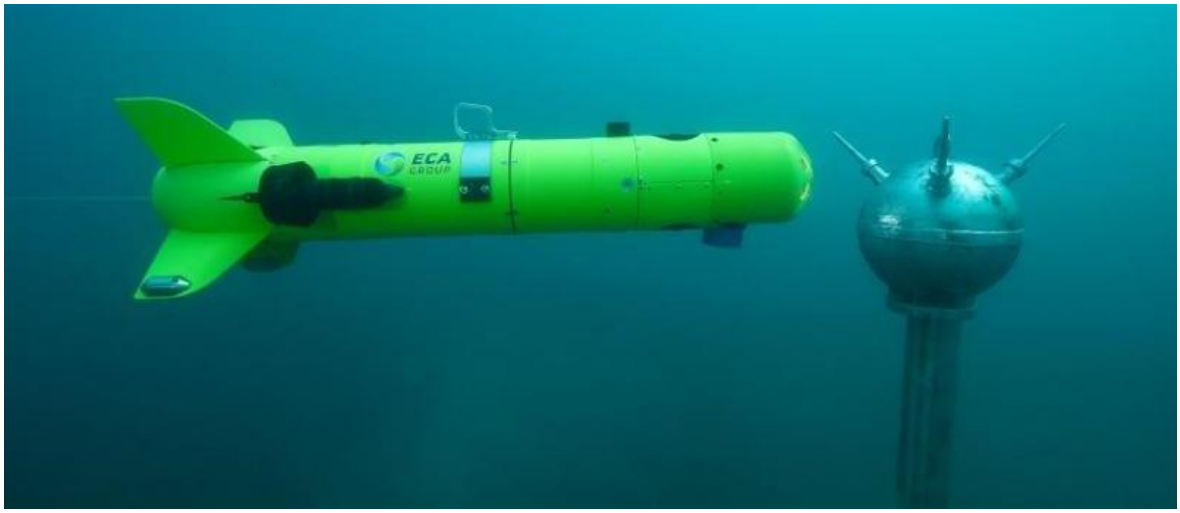


Figure 9: ECA Group. (n.d.). SEASCAN MK2 [Photograph]. ECA Group. https://www.ecagroup.com/en/solutions

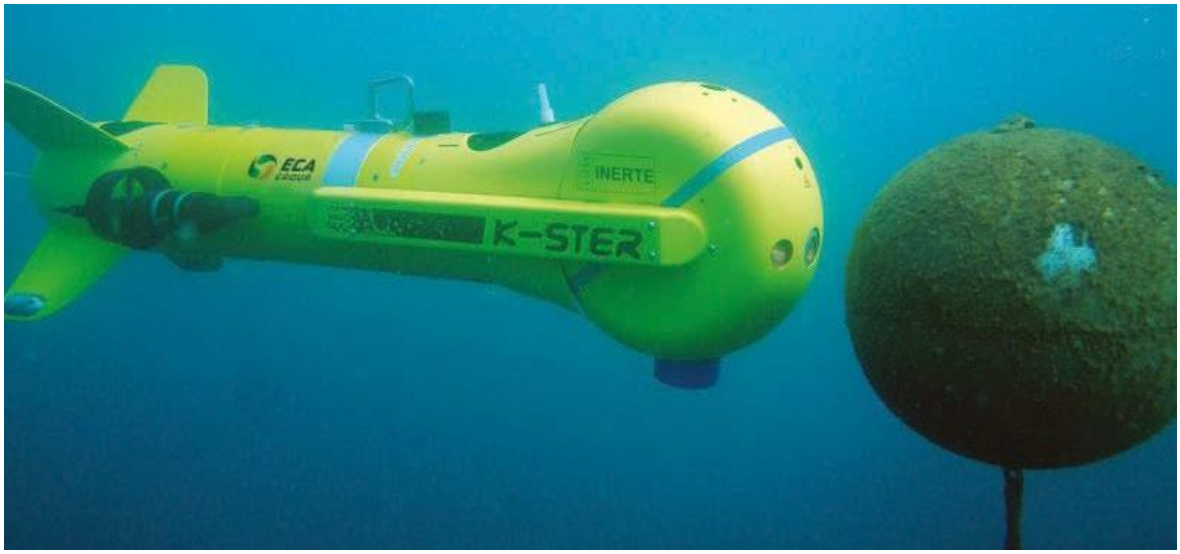


Figure 10: ECA Group. (n.d.). K-STER [Photograph]. ECA Group. https://www.ecagroup.com/en/solutions

Figure 11: ECA Group. (n.d.). UMISAS Toolbox [Ilustration]. ECA Group. https://www.ecagroup.com/en/solutions

UMIS™ sonar: UMISAS

The SAS system mounted on the AUV in charge of the localization of the mines (the A18-M and the T18-M) is called UMISAS, and uses a technique protected by an ECA group patent.

The UMISAS sonars are interferometric synthetic aperture sonars (InSAS[3]) that aim to obtain a spatial resolution of 3cm x 3cm in order to better classify small and irregular objects, as it is said in the ECA group webpage describing the A18-M features.

There, it is also highlighted that the UMISAS sonar does not use the navigation sensors dedicated to the sonar present in the market, since they are less efficient, do not allow geo-referencing of the images formed and are, above all, less reliable. UMISAS uses the drone itself inertial unit to preform SAS processing.

[3] For further explanation refer to page 14.

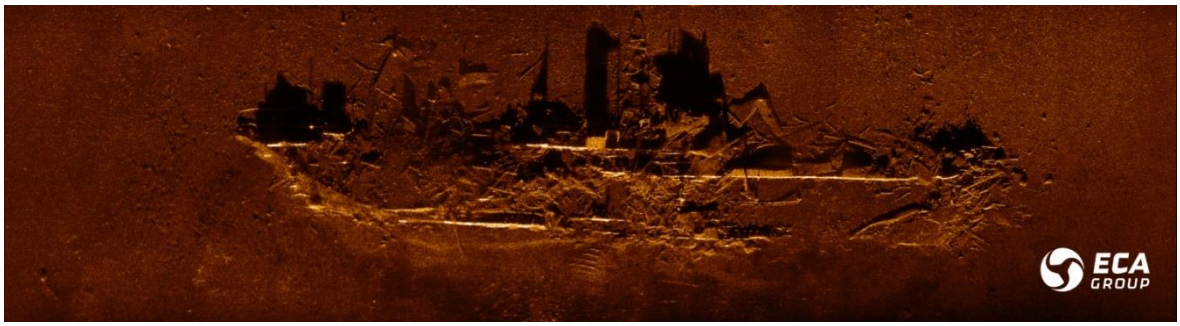


Figure 12: ECA Group. (2021, March 31). UMISAS 120 VHF sonar image of S.S. Ferrando collected off the coast of Hyères, France. [Photograph]. Belgium Naval & Robotics. https://www.belgium-naval-and-robotics.be/sucessful-sea-trials-eca-group-umisas-sonar/

The electronics of SAS and the approach of their testing

Each antenna of the SAS has sets of transmitters that emit the sound pulse and sets of receivers that listen back. Part of the electronics of these antennas is made up of RX32 cards.

For the first stage of this project, the predicted quantity of RX32 cards to be mounted on the SAS of the drones to be delivered to the BE/NL Navy amounts up to several hundreds of units. Each unit of RX32 card must perform under demanding environments without flaws, and furthermore, it must not fail once installed on the antenna.

The testing of electronic components before assembling them together is always a crucial part of the production of any device. In this scenario, the RX32 are mounted in places that are difficult to access for the operator. If a card is installed and the fault is only detected after testing the whole SAS, it must be dismantled in a high time-consuming process, not knowing if the error comes from that card or not.

Moreover, the number of units of these cards in each antenna makes the process of removing damaged units and replacing them an even more tedious and slow procedure, that could delay the production of the drones unnecessarily.

Due to these constraints, it is deeply important that all RX32 are flawlessly tested before their final integration into the system. Furthermore, because of the high number of units to be tested, it is also important that the tests are executed as fast and with as less interaction from the operator as possible to avoid human mistakes.

As a result of all of this, the solution reached was to build a test bench capable of automatically perform all necessary tests by just plugging the card onto it and interacting with the system through an intuitive computer interface where the operator can select the tests and see the results in a simple and fast way.

Hence, the scope of this thesis focuses on the automation of such test bench, and specifically on the software and hardware developed for that purpose and how it was designed to being able to incorporate the parts created by other teams, both inside and outside the company.

4 Technical explanation

General view

The following set-up is used for the development and automatization of the test bench.

The set-up consists of two major parts, namely a computer on which a GUI (general user interface) was programmed and a test bench consisting of a circuit board on which an RX32 is plugged. The RX32 is the component to be tested in this set-up. The communication between the computer and the test bench goes through three USB ports and one RJ45 port.

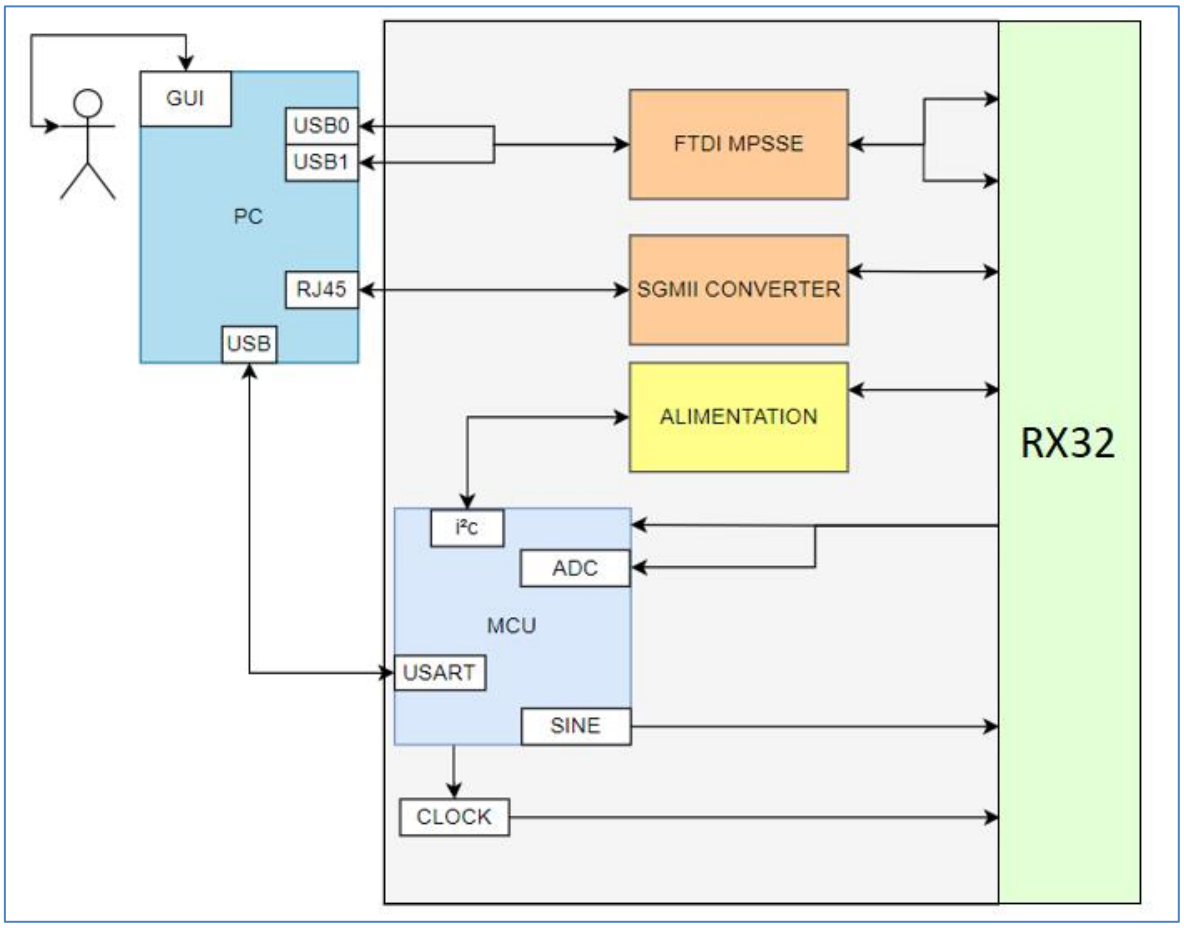


Figure 13: Software overview diagram

The operation of the set-up takes the following steps. A command is given to the test bench via the GUI. After the test bench received the command, the command is processed by the test bench. The result of the processing is sent back to the computer and can be read in the GUI. Based on the result, it can be checked whether the RX32 reacted as expected or not.

Project vision and scope

As agreed with the company, the product will be used to automatically preform the test of the RX32 card. It will have a simple interface for the user to select the test and visualize the information. It will produce a report in PDF form with the test results.

1) **The system must automatically preform the required tests for checking the correct performance of the UUT[4].**
   - User interaction to select the test.
   - Execute the test commands in correct order.
   - Run the provided scripts.
   - Send/receive digital and analog signals to/from UUT and peripherals.
2) **The system must check if the UUT passed the test or not.**
   - Compare the obtained results with the expected ones.
3) **The system must draw up a report containing the test results.**
   - Read the SN of the UUT.
   - Elaborate a report file in PDF displaying test and UUT information.
   - Print/export the report file.
   - Show test results on screen.

Use cases

Actor and target identification

Extern event	Actor	Target
Select the test to be preformed	User	Check the correct performance of the UUT system's elements
Generate power	DC/DC converter	Generate the correct voltages for the UUT
Generate a clock signal	Extern clock generator	Send a specific clock signal for some tests

Table 1: Actor and target identification

[4] Unit Under Test.

Use cases diagram

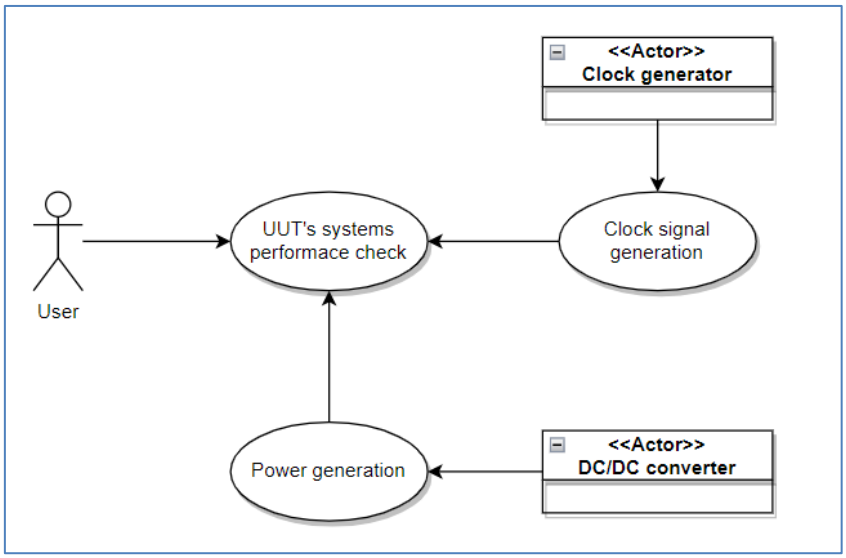


Figure 14: Use cases diagram

UUT's systems performance check

Actor:	User.
Description:	Check if UUT is working properly and ready to be used.
Preconditions:	The UUT is properly connected to all specified devices. The MCU is connected and responding.
Postconditions:	Test results report.
Normal course:	1.0 The user connects the system and the UUT and starts the PC program. 2.0 The system sends an ECHO message to the MCU and waits for it to respond ok. 3.0 Opens the interface and selects the test/tests to be performed on the unit. 4.0 The system preforms the selected test/s. 5.0 The user waits till the test is finished. 6.0 The test ends, the report is produced, and results are showed in screen.
Alternative course:	3.1 Interaction between user and system. The user wants to send a command to the Nucleo.
Exceptions:	2.0.E.1 The MCU is not responding (no/wrong ECHO message). [Stop system] 3.0.E.1 The scripts couldn't be run or returned error. [Test No OK] 3.0.E.2 There is an error on any peripherals. [Stop system]
Includes:	Power generation. Clock generation.

Priority:	Maximum. System core.
Notes and documents:	Test results report.

Table 2: Use case. UUT's systems performance check

Power generation

Actor:	DC/DC converter
Description:	Generate the different necessary power voltages to supply the UUT.
Preconditions:	A test has been selected.
Postconditions:	Power supply is generated and delivered to the UUT.
Normal course:	1.0 The user selects a test. 2.0 The system generates power when requested. 3.0 The system sends a feedback current. 4.0 The system stops generating power when requested.
Alternative course:	-
Exceptions:	3.0.E.1 The system can't send a feedback current. An error message is sent. 3.0.E.2 The system can't power some pin/s (short-circuit, no power…etc). An error message is sent specifying witch one is in fault state.
Includes:	-
Priority:	Medium. System peripheral.
Notes and documents:	-

Table 3: Use cae. Power generation

Clock signal generation

Actor:	Extern clock generator
Description:	Generate a clock signal of a certain frequency and send it to the UUT.
Preconditions:	Test where is used is selected.
Postconditions:	A clock signal is generated and delivered to the UUT.
Normal course:	1.0 The user selects one of the specified tests. 2.0 The system generates clock signal when requested.

	3.0 The system stops generating the clock signal when requested.
Alternative course:	-
Exceptions:	-
Includes:	-
Priority:	Medium. System peripheral.
Notes and documents:	-

Table 4: Use case. Clock signal generation

UML diagrams

See annexes (pg. 71).

Complementary specifications

1. Programming languages: C for MCU and for the PC application.
2. Platforms: Linux (Ubuntu 16.04).
3. Data storage: Database.
4. Report template must contain the date and name of the user performing the test, the SN of the UUT and the test results for each test (OK, no OK and N/A).

Sharing of the design process between the parties

As mentioned before in this document, as the project was not meant to be finished before our internship finished, the company came to an agreement with a subcontractor to develop part of the low-level software for the Nucleo board and the main part of the hardware of the testbench.

All PC parts were designed by us, and all drivers for the Nucleo too, except for the Input Capture, Watchdog and I2C drivers, that will be provided by this subcontractor.

Communication

The communication between the computer on which the GUI is programmed and the test bench on which the RX32 is plugged in runs through 4 ports. There are 3 USB ports and 1 RJ45 port.

Within this project the focus of communication lays on the connection between the computer and the NUCLEO. The computer and NUCLEO are connected via USB port.

Initially, the communication between the computer and NUCLEO was done via USART (universal synchronous/ asynchronous receiver/ transmitter). The disadvantage of USART is that the ring buffer is not present by default but had to be programmed. An attempt was made to program a ring buffer, but this did not succeed. During some tests, it was noticed that the communication between the computer and NUCLEO via USART was not optimal cause the computer did not receive

all the data, and to be able to properly develop and automate the test bench, it is of great importance that all data is received properly. For this reason, a switch was made to an USB OTG (USB on-the-go). The advantage of this device is that a ring buffer is built in. After adding the USB OTG, the communication between the computer and the Nucleo was much better. The computer received all the results without any problems.

The election of the communication technology was carefully studied and reported to the company in a document called “PC-MCU communication comparison” that is included in the Annexes (pg.76).

Protocol

To standardise the communication between the PC and the Nucleo, it was decided to develop a protocol of our own. The protocol contains all the necessary commands that will be needed to carry out the tests. These commands are made up of different symbols, follow strict rules and must be formatted correctly.

The rules to which the command must adhere are as follows:

- Command is case sensitive.
- All commands have a response.

All consists of 3 capital letters. Each command consists of a header, command, parameters, and an end character with a delimiter between each block. At the very end, a new line character is required before the command can be properly interpreted. If these blocks are correctly assembled, a command will look like this:

<HEADER>:<CMD>:<PARAM>:<INFO>\n

- Header: Contains one single character that indicates the direction of the message: Nucleo to PC or PC to Nucleo.
- CMD: All commands must be 3 letters.
- PARAM: All the parameters needed for the command separated by commas: <p1,p2,p3,…>
- INFO: ‘!’ or ‘?’. Execute an action or ask for information.

The following symbols are defined in the protocol:

- **C** → Header of command for messages coming from the PC.
- N → Header of command for messages coming from the Nucleo.
- **:** → Separator.
- **,** → Separator of parameters.
- **?** → Query for response.
- **!** → Query for action.
- \n → End of message (managed by the USART OTG in this application).

PC part

Coding language

As mentioned earlier, the commands to the test bench were given through a GUI programmed on the computer. The programming of the GUI was done by the developers themselves, using the coding language C. In addition, the GTK library was used to render the GUI. the choice of coding language depended on the operating system that would be used. The ECA company and developers preferred Linux as the operating system. This was because it was expected that the scripts coming from the external company might have been written in several other coding languages. The developers expected that there would be fewer problems in executing these scripts on Linux than on Windows. Within Linux, the developers had experience with the C coding language and Javascript. Since ECA has less experience with Javascript than with C, C was chosen as the coding language for this project

Code

The code is composed of several parts. In what follows, the parts are discussed step by step. The discussion of the parts follows the process by which the different parts are covered. The whole thing will be illustrated using an example so that the operation of the code becomes clear.

The main function is the most important function of all. This function is the start and end of the programme. A control system always calls on the main when the programmer executes his code.

The second part of the code to be discussed is the GUI. The GUI is the platform through which developers can send commands to the test bench. The developers can issue commands as follows. First, the developer must select the NUCLEO. This way, the computer knows which NUCLEO to pass the command to. Next, the developer must enter the serial number of the RX32. The purpose of entering the serial number is to find out which tests were performed on which RX32. Third, the developer needs to select the test folder. Here it is important that the developer places the correct test in the test folder before running the test. To run the test, the developer needs to press play. While running the test, the GUI displays the received data. If necessary, the user can enter commands manually. He can do this by entering a command in the right pane and after which he should click ‘send’. After the user has done this, the command will be executed. Even then, the GUI will display the received data.

The third part of the code is the USB. Via USB, a communication channel is established between the GUI and chardev (Nucleo). Through this channel, the GUI can send commands to the Nucleo and the GUI receives responses from the Nucleo

The fourth part of the code is the test sequencer. The test sequencer plays a very central role within the entire code. The test sequencer comes into action after the user has entered all the information in the GUI (including which tests they want to run) and the user runs the tests.

Once the user has done this, the test sequencer goes through 3 steps.

- In step 1, the test sequencer looks at which tests to run.
- In step 2, the test sequencer reads and runs the tests one at a time. The sequencer checks what the name and number of the test is. In addition, it examines what command should be given to the Nucleo and indicates what response it expects from the Nucleo. Once the data is collected, the test sequencer executes the test. In doing so, it sends the commands of the test in question to the Nucleo. Then the test sequencer compares the responses received from the Nucleo with the responses the test sequencer expected in advance. If the responses received are equal to the expected responses, the test sequencer moves on to the next test and the steps described above are repeated on the next test. If the responses received do not equal the expected responses, the test sequencer repeats the commands. If the test sequencer has sent out the commands three times, but the responses received still differ from the expected responses, the test sequencer stops sending out the commands of this particular test and moves on to the next test. Once all tests have been performed, the test sequencer moves on to step 3.
- In step 3, the results obtained are reported. These results are stored by the test sequencer in a struct.

The fifth part of the code is pdf. After the test has been run, the results of these tests can be accessed in a pdf document. This pdf document is automatically generated after the tests are run. The title of the pdf document is always the serial number of the RX32. In this way the results of the tests can always be linked to the correct RX32.

The document provides the following information: serial number of the RX32, which tests were run, whether the tests were successful or not, date, time.

As previously indicated, an error message 'Nok' will be given if a test generates responses other than the expected responses after three attempts. If a test takes less than three attempts to generate the same responses as the expected response, the test will be rated as 'okay'.

The results of the tests performed can not only be found in the pdf documents. In addition to the pdf documents, a database is also created. This sixth part of the code collects the results of the tests performed per RX32. The difference with the pdf documents is in the display of the results. The results are displayed per test performed in a very clear and visual way in the pdf documents. In the database, the results are also displayed in an orderly fashion, but in a slightly less visually friendly way. Nevertheless, the database has a great advantage in certain situations. Suppose a test is run several times at the same RX32, the database allows the results of the different runs to be displayed in a table. The pdf documents cannot give this info. In fact, a new pdf document will be created for each run. The results of the different tests, will not be able to be displayed at a glance.

'General' is the last subdivision of the code. In this section, a number of functions are grouped together. A first function is 'timestamp'. This function displays the date and time. A second function is 'regex'. The function 'regex' is used to check whether the entered data is qualitative. More specifically, it checks whether the serial number of the RX32 can be correct. The last function 'listport' lists all serial devices. This is important to be able to find NUCLEO.

Serial connection

The Nucleo behaves like a serial device that is connected, via USB On-The-Go, to the computer.

In Linux everything is regarded as a file or document so that the same functions can be used as with a normal text document. However, within Linux, the Nucleo is recognised as a character device. Because of this, extra configuration is needed to open the file that the Nucleo represents. The configuration uses the POSIX terminal control or thermios.h header file to set up serial communication. After the configuration is complete, the PC can access the information transmitted by the Nucleo without any problems. When processing the information, the following process is always repeated:

1. opening the filepointer
2. configuring the filepointer
3. reading the filepointer
4. closing the filepointer

In a normal text file, the file pointer remains open until all necessary actions are processed. In the current setup, the decision was made to close the file pointer after each action and then open it again. This is because otherwise a large part of the CPU capacity is taken up.

Test sequencer

The test sequencer is a state machine that reads the test file and sends the commands to the Nucleo.

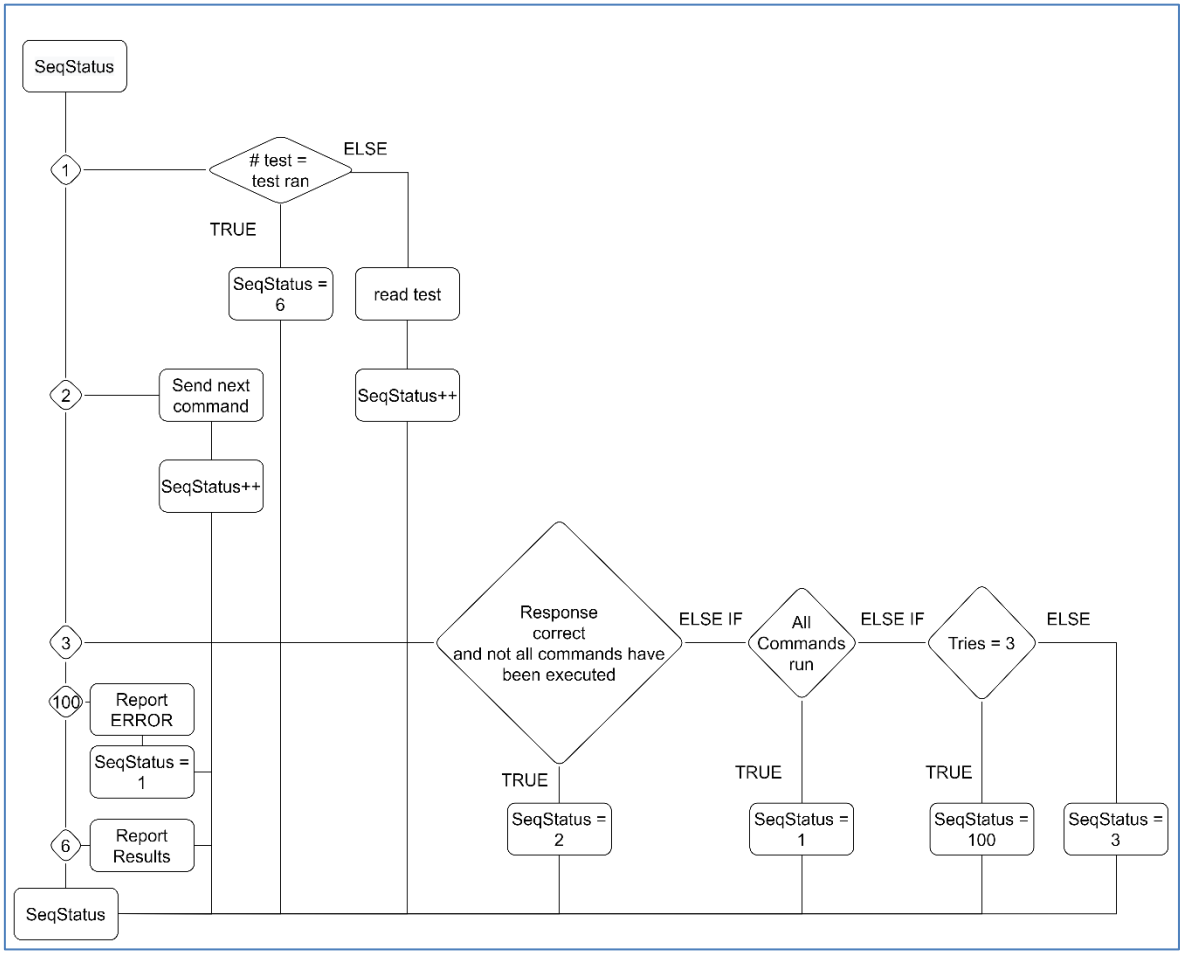


Figure 15: Flowchart of Test sequencer

When the correct test folder is chosen in the GUI and play is pressed, you will arrive at status 1 within the test sequencer.

Status 1

In status 1 the number of tests is compared to the number of tests that were run. If no tests have been run yet, the test sequencer reads the first test and increases the status by 1. After 500 milliseconds (see also GTK mainloop and single threating) the sequencer switches to status 2.

Status 2

In state 2 the test sequencer sends the command and the state is increased again by 1. After that status 3 is reached.

Status 3

In state 3 a comparison is made between the response received and the response expected. In addition, it is also checked whether not all commands were executed. If this is the case, the test sequencer goes to state 2 and state 2 is repeated.

If not, the test sequencer checks whether all commands have been completed. If all commands are completed, it goes to status 1 and repeats status 1.

If not, status 3 is run through again. Status 3 can be run through a maximum of three times. If status 3 is not successfully completed after 3 times, the test sequencer switches to status 100.

Status 100

In status 100 an error is reported. After that the sequencer switches to status 1 where the next test is read.

Status 6

When all tests have been completed, the test sequencer changes to state 6. In status 6 the data are saved in the database and the pdf is generated.

After the report, the test sequencer is reset and is ready to run through the next test sequence.

GTK main loop and single Threading

It would be ideal if one could receive data from the Nucleo at any time, send data, iterate the test sequencer and perform other events like clicking buttons at the same time. Using the GTK API, one can do this without introducing too much effort or vulnerabilities into the code.

GTK provides Glib. Glib is a library that offers, among other things, main loop abstraction. From this library two functions are used.

- idle_add
- timeout_add

```
while (!done)
{

    if (serialport > 0)
    {

        if (flag)
        {
            g_idle_add((GSourceFunc)reading, &serialport);
            g_idle_add((GSourceFunc)readUpdate, DATA_BACKptr);

            // om de 500ms volgende functie oproepen:
            g_timeout_add(500, (GSourceFunc)testsequencer, NULL);
            flag = 0;
        }
    }
```

Figure 16: GLib functions in main loop GUI

idle_add

This function makes the given function an event which will be called when there is no event with a higher priority (“GLib.Idle_add,” 2022). This function is used twice in the code. Once for the "continuous" reading or receiving of data sent by the Nucleo and a second time for updating the buffer which is used to visualise the received data in the GUI.

timeout_add

With timeout_add one can set an interval when the specified function should be called (“GLib.Timeout_add,” 2022). In the code this function is used to iterate the test sequencer every 500 ms.

Test

The tests used in the test sequencer are constructed according to the following rules:

Line one and two are reserved for the name and number of the test.

After that there is an empty line. The following lines are for the commands and the expected response to that command.

The first command of a test must always be the ECHO command. This is done to ensure that the PC and the Nucleo are connected and can communicate with each other.

```
1   # POWER TEST
2   # Number:3
3
4   C:ECH: :!
5   N:ECH:OK:!
6   C:PWR:1:!
7   N:PWR:OK:!
8   C:CLK:1:!
9   N:CLK:OK:!
10  C:SEL:1:!
11  N:SEL:OK:!
12  C:LOL: :?
13  N:LOL:1:!
14  C:PWR:0:!
15  N:PWR:OK:!
16  C:CLK:0:!
17  N:CLK:OK:!
18  C:SEL:0:!
19  N:SEL:OK:!
```

Figure 17: Power test

Database

The results are stored in a relational database. In this case, MariaDB is being utilised. This is an open-source relational database management system or RDBMS that emerged from a fork of the MySQL source code because of concerns over the acquisition of MySQL by the Oracle corporation.

The database is composed of the two tables card and test. These tables are linked via SerialNumber_ID. Using the serial number of each RX32 card, they will receive their unique SerialNumber_ID. Which value is then used to associate the completed tests to the card.

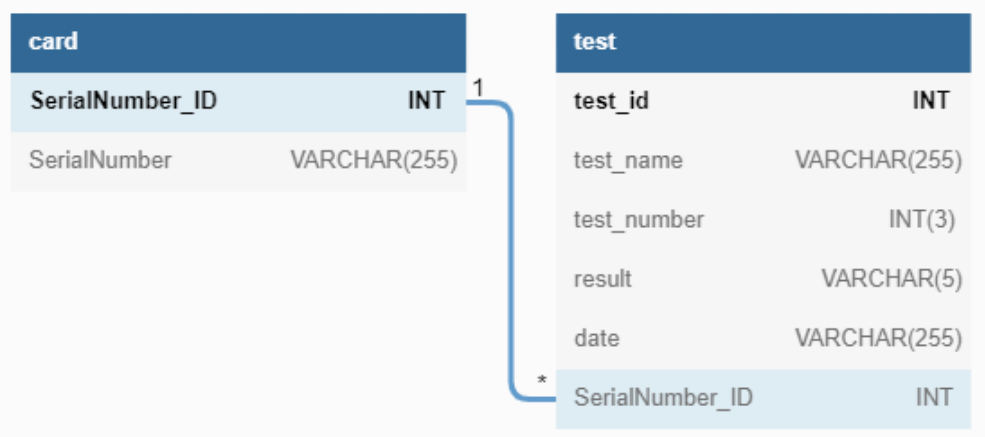


Figure 18: Tables of the Database

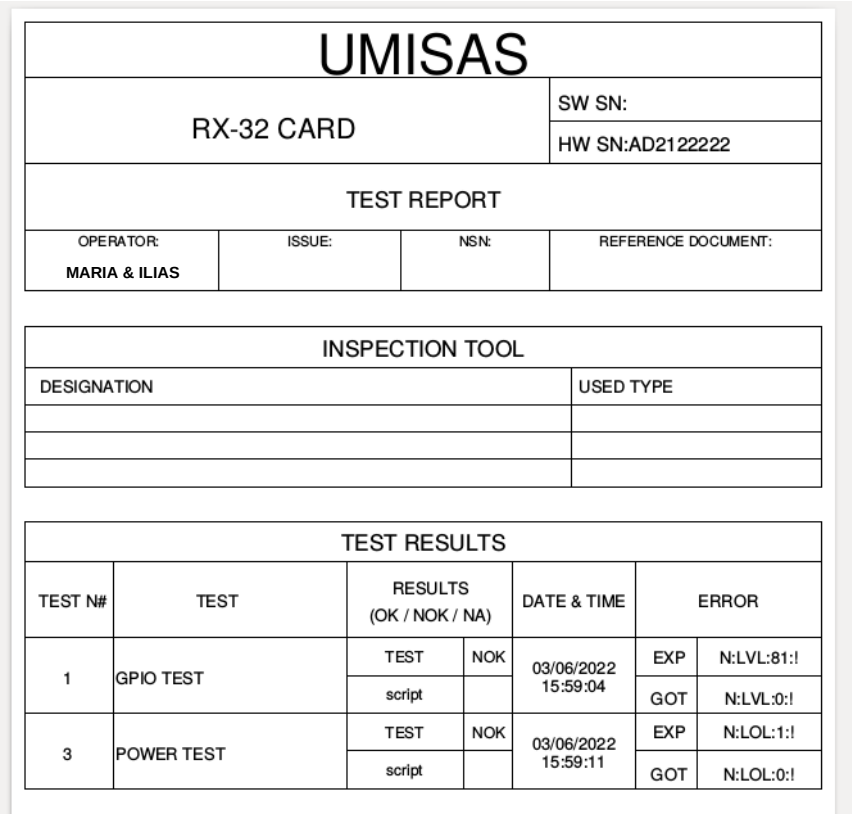
The relationship between the two tables is a "one-to-many" relationship. This means that a card can have multiple test_id's but a test cannot have more than one serialNumber_id. So, in short, 1 card may have several tests but a test may not have several cards as "owners".

Figure 15 contains a visual representation of the table. As you can see, the test table opens up various values including the name, number, date and the result of the test.

PDF

The generation of the PDF reports is done with the help of Haru Free PDF Library. This is an open-source library that is cross-platform and provided free of charge. This library can be used to generate quite complex documents. The following are some of the features supported by this library:

- Outline, text annotation, link annotation.
- Generating PDF files with lines, text, images.
- Embedding PNG, Jpeg images.



UMISAS

RX-32 CARD	SW SN:
	HW SN:AD2122222

TEST REPORT

OPERATOR:	ISSUE:	NSN:	REFERENCE DOCUMENT:
MARIA & ILIAS			

INSPECTION TOOL

DESIGNATION	USED TYPE

TEST RESULTS

TEST N#	TEST	RESULTS (OK / NOK / NA)		DATE & TIME	ERROR	
1	GPIO TEST	TEST	NOK	03/06/2022 15:59:04	EXP	N:LVL:81:!
		script			GOT	N:LVL:0:!
3	POWER TEST	TEST	NOK	03/06/2022 15:59:11	EXP	N:LOL:1:!
		script			GOT	N:LOL:0:!

Figure 19: example of PFD-rapport

The PDF reports that are created after all the tests are carried out contain the following information: serial number of the RX32 card, name of the operator, tests that were carried out, test results of the tests that were carried out.

Nucleo part

The Nucleo software is based on a central software call Engine, that manages all the functionality of the microcontroller. The control of the peripherals is divided in drivers that are called by the Engine when necessary. The communication with the PC is managed by the Nucleo itself and administrated by the management unit. In the main program, only this central software is called in an Update() function that keeps the Nucleo actualized and synchronized with the PC[5].

STM32CubeIDE

This tool provided by ST is a generic Eclipse-based platform that allows the user to configure the microcontroller and write code, upload it to the MCU and debug it all in one intuitive platform.

All written code is located in the Core\Src and Core\Inc folders.

MCU configuration

The .ioc file in STM32CubeIDE provides a graphic interface for the configuration of the MCU resources. After configuring the parameters, the necessary lines of code are generated in the main.c and the hardware initialisation file[6].

The following resources are configured:

- System Core
  - DMA
  - NVIC
  - RCC
- Analog
  - ADC
  - DAC
- Timers
  - TIM
- Connectivity
  - USB_OTG_HS
- Clock

[5] See UML diagram in Annexes, page 59.

[6] Only resources for active code are included in this document. For the I2C, Watchdog and Input Capture drivers, configuration will be done according to the providers of those drivers.

DMA

Direct memory access (DMA) is used to provide high-speed data transfer between peripherals and memory and between memory and memory. Data can be quickly moved by DMA without any CPU action. This keeps CPU resources free for other operations (Reference Manual, 2022).

In this driver DMA will be used to periodically generate a wave without blocking the CPU. For that, the DAC channel's DMA request is enabled and configured in circular mode and word data width on both peripheral and memory.

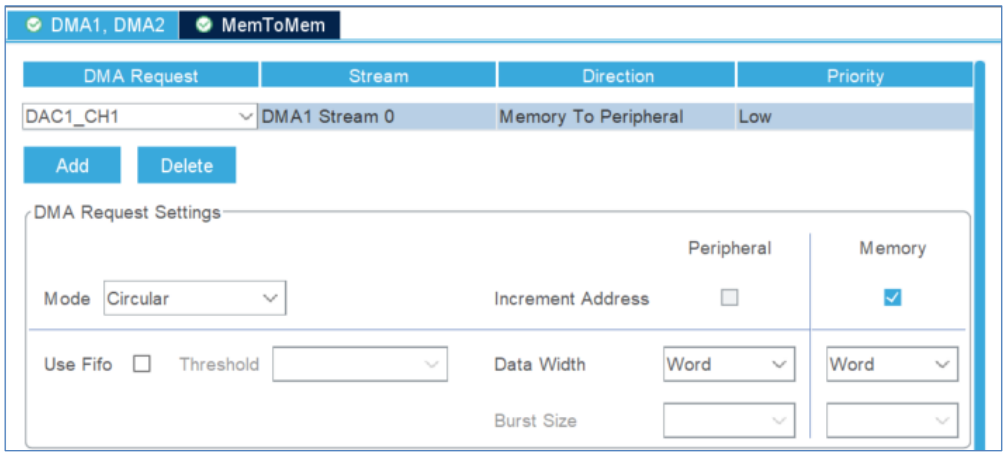


Figure 20: DAC DMA configuration.

NVIC

The Nested Vector Interrupt Controller is the software that manages the interruptions. To prevent the DMA from interfering with the to be implemented DAC control, the interrupt generated by the DMA must be lower than the used for controlling the DAC.

RCC

The RCC manages the clock and reset generation for the whole microcontroller. The STM32H7A3 has 2 external clock oscillators and 4 internal ones. For the optimal performance of high frequency peripherals (DAC and USB_OTG_HS) the high-speed external oscillator (HSE) must be selected as crystal/ceramic resonator, capable of supporting a wide range of frequencies (4-50MHz).

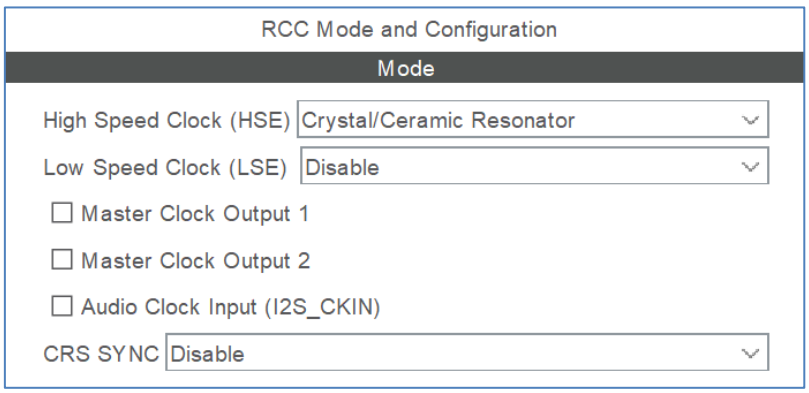


Figure 21: RCC Mode and Configuration.

ADC

Each ADC consists of a 16-bit successive approximation analog-to-digital converter and has up to 20 multiplexed channels. A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored in a left-aligned or right-aligned 32-bit data register and can be configured as 16, 14, 12, 10 or 8-bit resolution (Reference Manual, 2022).

As 4 channels are needed for test purposes, all of them are selected from only one of the two available ADC. This allows a better performance of the ADC as the two of them are tightly coupled and can interfere with each other.

DAC

The DAC module is a 12-bit, voltage output digital-to-analog converter. The DAC can be configured in 8- or 12-bit mode and may be used in conjunction with the DMA controller. In 12-bit mode, the data could be left- or right-aligned (Reference Manual, 2022). The output must be configured as "connected to only external pin". The trigger will be a timer event, and the DMA request is enabled.

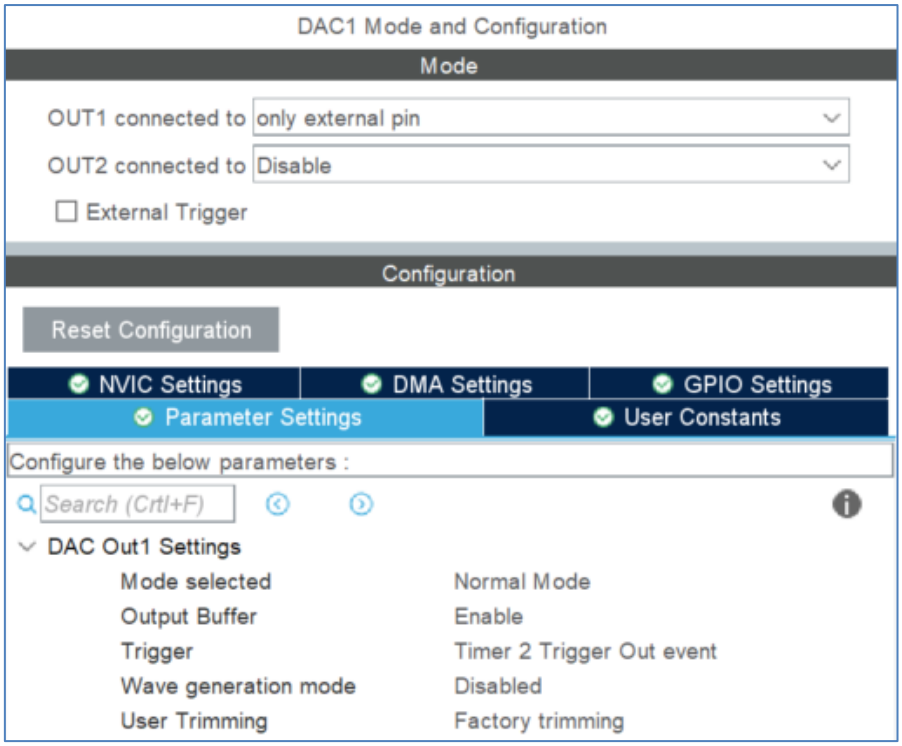


Figure 22: DAC1 Mode and Configuration.

TIM

The general-purpose timer is used in this driver to trigger the ADC-DMA conversion. It will periodically produce an event, which is acknowledged by the DMA, who generates a request to produce a determined voltage. The repetition of this action generates the sine wave form.

The system peripheral clock is divided, generating and event at the desired frequency. This will allow the MCU to produce the correct signal frequency using a number of samples.

USB_OTG_HS

The USB OTG is a dual-role device (DRD) controller that supports both device and host functions and is fully compliant with the On-The-Go Supplement to the USB 2.0 Specification. For this software it will be configured with a device-only internal physical interface.

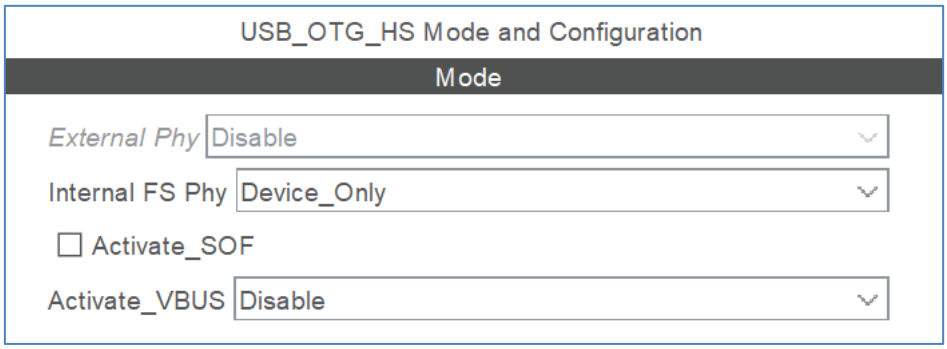


Figure 23: USB_OTG_HS Mode and Configuration.

These settings allow the following features to be implemented:

- 1 bidirectional control endpoint0.
- 8 IN endpoints (EPs) configurable to support bulk, interrupt or isochronous transfers.
- 8 OUT endpoints configurable to support bulk, interrupt, or isochronous transfers.
- Management of a shared Rx FIFO and a Tx-OUT FIFO for efficient usage of the USB data RAM.
- Management of up to 9 dedicated Tx-IN FIFOs (one for each active IN EP) to put less load on the application.
- Support for the soft disconnect feature.

The serial communication will be then completely managed by this interface, and the only task the software must do to receive and send messages is to read the provided buffer and use the CDC_Transmit_HS() function to send the transmitting buffer through USART.

Clock configuration

To make the DAC timer work properly the clock frequency of the peripheral bus connected to it must be 100MHz. This value must be the same for the whole project. If it's changed, a further reconfiguration of the timer prescaler and period will be necessary to maintain the correct wave frequency.

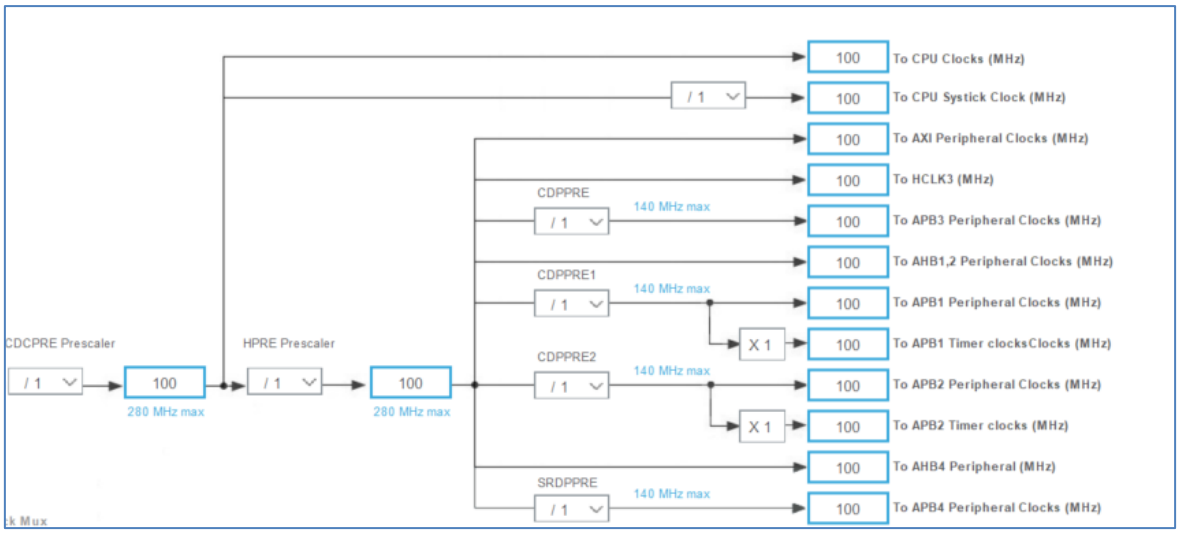


Figure 24: Clock Configuration.

Software description

The test bench software for the Nucleo is divided in three main parts:

First, the files generated by the STMCubeIDE platform. These files contain all the HAL and CMSIS libraries, and drivers for low-level control of the peripherals. It includes the main.c and main.h files, that are the only ones that had been modified.

In second place, the peripheral drivers. They can use the HAL library and include all the functions necessary to execute all required actions in a simpler way than only using the provided libraries.

Last, the unit that manages and controls the peripheral's actions and receives and sends information from/to the PC. This unit is placed into the Engine.c and Engine.h files and incorporates a message formatter too (CMD.c and CMD.h).

Each unit and driver will be further described in detail in the next chapters.

Peripheral drivers

GPIO Driver

The GPIO modules in the STM32H7 family are divided into 11 ports named from A to K. Each port has 16 I/O[7] pins that can be individually configured and controlled using registers.

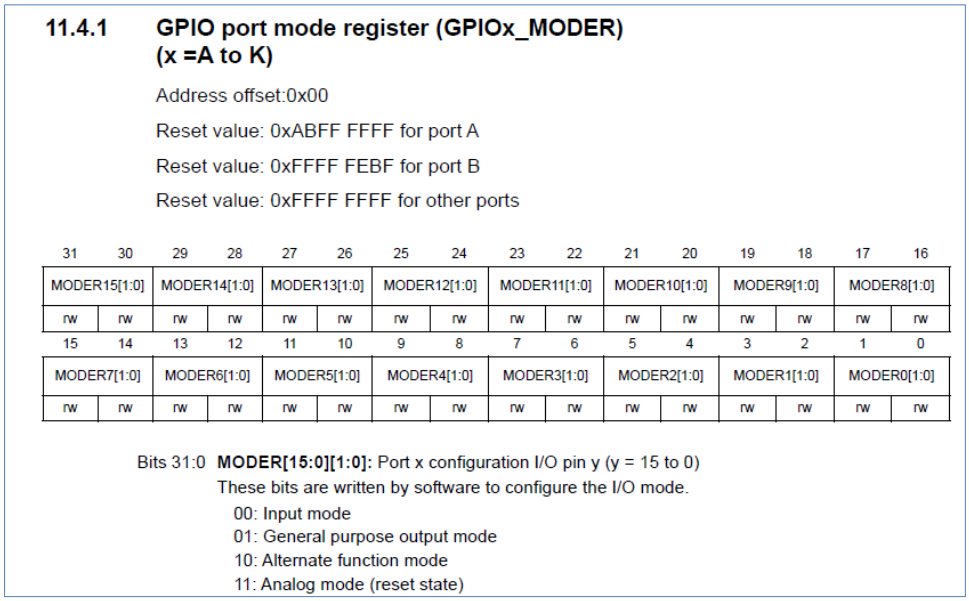
The purpose of this driver is to modify the configuration of this I/O pins without using the HAL library, by writing and reading those registers. The GPIO module has many of them, but the ones used in this driver are the following ones:

- **I/O port control registers**

[7] Input/Output

Each GPIO port has four 32-bit memory-mapped control registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR) to configure up to 16 I/Os.

From this set of registers, the only one that will be written on will be the GPIOx_MODER register, which configures the pin's mode of each port.



11.4.1 GPIO port mode register (GPIOx_MODER) (x =A to K)

Address offset:0x00

Reset value: 0xABFF FFFF for port A

Reset value: 0xFFFF FEBF for port B

Reset value: 0xFFFF FFFF for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

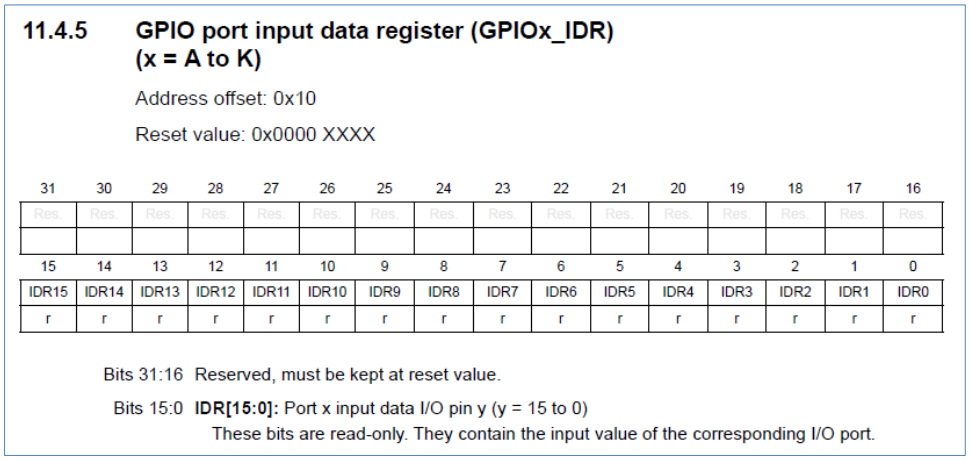
Bits 31:0 **MODER[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

Figure 25: STMicroelectronics. (2022, January). GPIO port mode register (GPIOx_MODER) [Screenshot]. In Reference Manual. STM32H7A3/7B3 and STM32H7B0 Value line advanced Arm®-based 32-bit MCUs (RM0455 rev 8 ed., p. 517).

- **I/O port data registers**

Each GPIO has two 16-bit memory-mapped data registers: input and output (GPIOx_IDR and GPIOx_ODR).

The data input through I/O is stored in the GPIOx_IDR read-only register.



11.4.5 GPIO port input data register (GPIOx_IDR) (x = A to K)

Address offset: 0x10

Reset value: 0x0000 XXXX

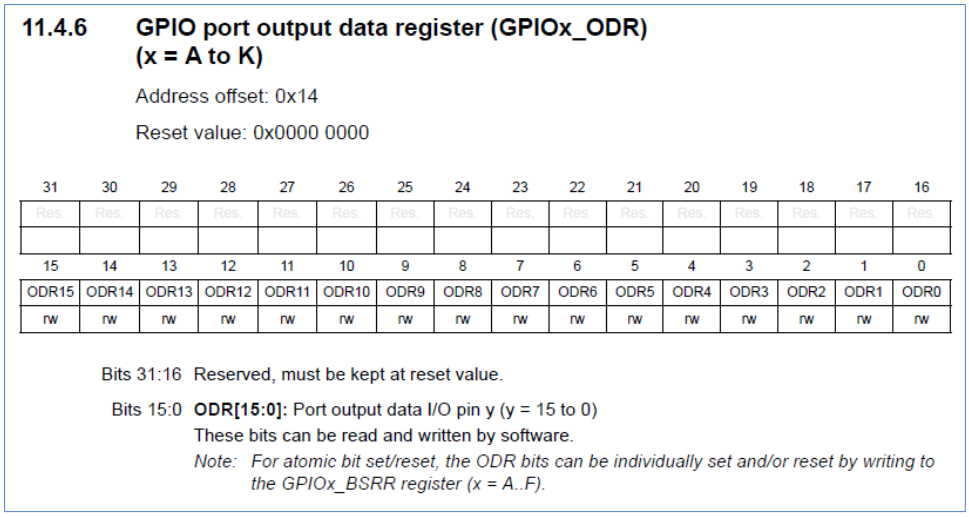
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDR[15:0]:** Port x input data I/O pin y (y = 15 to 0)
These bits are read-only. They contain the input value of the corresponding I/O port.

Figure 26: STMicroelectronics. (2022, January). GPIO port input data register (GPIOx_IDR) [Screenshot]. In Reference Manual. STM32H7A3/7B3 and STM32H7B0 Value line advanced Arm®-based 32-bit MCUs (RM0455 rev 8 ed., p. 517).

GPIOx_ODR is a read/write register that stores the data to be output.



11.4.6 GPIO port output data register (GPIOx_ODR) (x = A to K)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODR[15:0]:** Port output data I/O pin y (y = 15 to 0)
These bits can be read and written by software.
Note: For atomic bit set/reset, the ODR bits can be individually set and/or reset by writing to the GPIOx_BSRR register (x = A..F).

Figure 27:STMicroelectronics. (2022, January). GPIO port output data register (GPIOx_ODR) [Screenshot]. In Reference Manual. STM32H7A3/7B3 and STM32H7B0 Value line advanced Arm®-based 32-bit MCUs (RM0455 rev 8 ed., p. 517).

The pin information will be provided in 16-bit variables, where each pin has a position assigned from 0 (LSB) to 15 (MSB).

The driver compares that variable with pin and port masks and determines the register that should be read or written on.

For the initialisation of the pins, the driver acknowledges which port clocks (apart from the ones already used) must be accessed and configured.

After that, for addressing each pin, the driver shifts the correct digit to the provided position of the register and configures and controls that pin.

The driver consists in a source file and a header file called GPIO_Driver.c and GPIO_Driver.h. The elements contained in these files are described below.

HEADER FILE

The header contains the function definitions, the definition of the number of ports (11) and pins in each port (16), the struct containing the status of the GPIO and the GPIO configuration array.

As we explained, GPIO will be configured and controlled using bit masking, so the source file also contains the mask of each GPIO port.

```
typedef struct
 {      GPIO_StatusTypeDef status;
        uint16_t GPIO[NUMBER_OF_PORTS];
 } GPIO_Params;
```

SOURCE FILE

The source file contains all the functions and variables that allows the driver to work independently of the main.c where it is used. The source also contains the initialisation of parameters structure for the driver, called UGPIO, and all the arrays for the pin and port masks.

```
GPIO_Params UGPIO = {GPIO_NOT_READY, {0}};
```

The functions can be divided in two classes: GPIO initialisation and GPIO control.

For initialisation there are functions to enable the clock, set the pin default pin mode and then change it to input or output.

For controlling the GPIO, there are functions that allow to read/write an individual pin by writing on registers, and also for controlling all the pins of a port at once, depending on which pins of the provided array are 1 or 0 (see Annexes, page 72).

ADC Driver

The ADC of the Nucleo board can read analog values in polling mode, using interruptions, or using DMA.

For multiple channels the most common option is reading the values using Direct Memory Access and interruptions. Using this configuration, the DMA sends continuous requests to the ADC and stores the value in an array with the same number of values as the number of channels that we are reading. This option is usually better because the CPU does not block while the conversion is being made. But in this case, to maintain a high-speed conversion and a good resolution, the number of cycles at which the ADC works must be kept high enough. The problem with this is that the DMA generates too many requests, and the CPU is always at the call-back function. The while(1) loop is not executed and the program does not work properly.

To solve this, the ADC will be read in polling mode. Taking into consideration that the ADC will only be called at specific times, the CPU will not be busy enough with that for it to be a problem.

Instead of relying on the DMA for the multi-channel configuration, we will modify it and change the configuration every time a channel must be read. For that, the configuration in the .ioc file will be 4 channels in independent mode. But then, in the main.c the Init() function will be replaced for other that only makes one conversion at a time.

This way the main channel will be reconfigured for each conversion, selecting the input and starting/stopping the ADC.

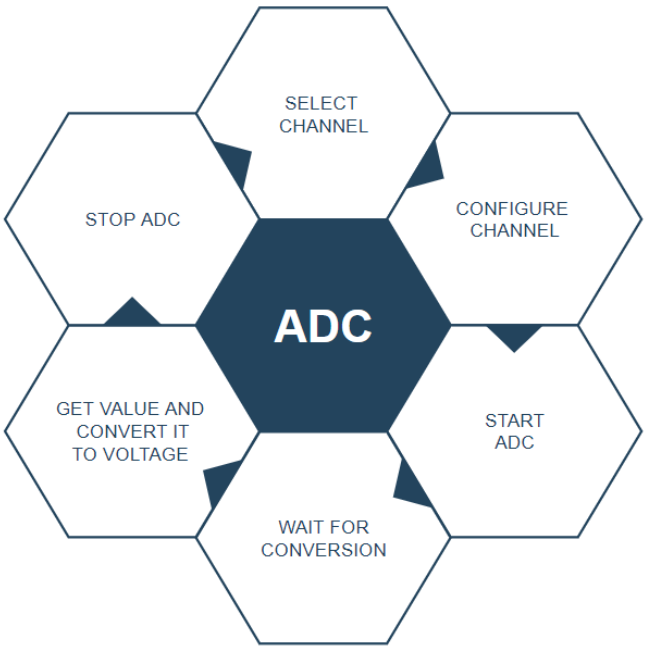


Figure 28: ADC sequence diagram

The ADC gets a digital value between 1 and 1023 (as we are using a 10-bit resolution) that represents the measured voltage. The conversion of that value to volts is made by calibrating the ADC.

For achieving that one of the channels was connected to GND and that value set as the offset of the ADC. The conversion is made using the following formula:

$$Voltage\ (V) = \frac{(v - offset) \cdot Vref}{full\_scale}$$

Where *v* is the digital value, *Vref* is the maximum value read (3.3V according to the datasheet) and *full_scale* is the maximum digital value: $2^{10} = 1024$ (STM32 Nucleo − 144 Boards Data Brief, 2021).

HEADER FILE

The header contains the function prototypes, and the parameters struct containing the status of the DAC and the float array for storing the read values from ADC. It includes the HAL header file, the definition of the number of channels and other data necessary for the driver too.

The resources used from the Nucleo are included as extern variables from the main where the driver is used.

```
typedef struct
{       ADC_StatusTypeDef status;
        float voltage[ADC_CHANNELS];
} ADC_Params;
```

SOURCE FILE

The source file contains all the functions and variables that allow the driver to work independently of the main.c where it is used.

The source also contains the initialisation of parameters structure for the driver, called ADC.

```
ADC_Params ADC = {ADC_OFF, {0}};
```

Apart from the modified Init() function, there are functions for selecting each channel and reading the selected channel, as well as for converting digital values to voltage and handling ADC errors.

DAC Driver

The DAC of the Nucleo board only generates triangular and noise waves. For generating a sine wave the DMA and a timer are needed.

According to the information provided by ST (Application Note: Audio and Waveform Generation Using the DAC in STM32 Products, 2020), to generate a sine wave the following formula is used, where n is the number of samples and 0xFFF corresponds to the 3.3V voltage range of this board.

$$y_{SineDigital}(x) = \left(\sin\left(x \cdot \frac{2\pi}{n_s}\right) + 1\right)\left(\frac{(0xFFF + 1)}{2}\right)$$

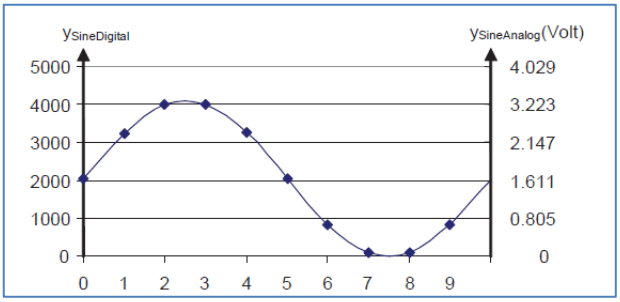


Figure 29: STMicroelectronics. (2020, June). Sine wave model samples [Figure]. In Application note: Audio and waveform generation using the DAC in STM32 products.

To set the frequency we use this other formula:

$$f_{Sinewave} = \frac{f_{TimerTRGO}}{n_s}$$

Knowing the frequency of the timer clock is 100MHz and that is divided by the prescaler and the period, the final frequency is obtained setting the number of samples to what is needed.

The driver consists in a source file and a header file called DAC_Driver.c and DAC_Driver.h. The elements contained in these files are described below.

HEADER FILE

The header contains the function definitions, the definition of the number of samples (n = 50) and the struct containing the status and the output voltage peak to peak. It includes the HAL header file too.

The resources used from the Nucleo are included as extern variables from the main where the driver is used.

```
typedef struct
{       DAC_StatusTypeDef status;
        float vpp_out;
} DAC_Params;
```

SOURCE FILE

The source file contains all the functions and variables that allows the driver to work independently of the main.c where it is used. The resources used from the Nucleo are included as external variables from the main where the driver is used.

The source also contains the initialisation of parameters structure for the driver, called DAC.

```
DAC_Params DAC = {DAC_OFF, 0};
```

There are functions for initialising the DAC and the DMA independently, and to set the desired output voltage. Also, there is a function that generates the sine wave form (see Annexes, page 73) and others that start and stop this sine wave generation. The driver includes an error handler too.

Input Capture, I2C and Watchdog drivers

The Input Capture driver must configure and set a timer capable of reading a pin and comparing the input it receives of it (ON or OFF) for determining the blinking frequency of a LED. This driver is meant to be developed by a subcontractor, and as it is not crucial for the development of the test it has not been included into the Engine management.

The same thing for the I2C driver. The purpose of this driver was to communicate with the EEPROM of the DC/DC converter peripheral to control it and read a feedback current. In this project it has been replaced by a simple command that powers a pin connected to the DC/DC converter on and off to trigger the RUN0 and RUN1 pins, that according to the datasheet of the device, allows the user to control it without I2C.

The Watchdog driver will be developed by the subcontractor too. It has been conceived as a timer that triggers and reads a pin after some selected time. This driver has been replaced by a command in the Engine, that executes a set of actions that allow the user to do the exact same thing but without using one of the timers of the Nucleo.

The principle is simple: the Nucleo has an internal timer called System Tick, that counts the time that has passed since it was powered. By saving that time at some point and comparing it with the actual System Tick-value we can know if a specific amount of time has passed. This function is included in the Annexes (pg. Watchdog driver substitute function73).

Functionality that does not need a driver

For some of the test that will be performed on the RX32 there is a set of fixed pins that must be read or written and that are not managed by the GPIO driver, but by commands. This prevents the

Nucleo pins that will always be connected to the same RX32 pins from being modified by the operator executing the tests by mistake.

The same for the selector that activates the multiplexor that chooses the way of connecting the RX32 to the PC or to itself. There is a specific command that powers the fixed pin connected to it on or off.

There is another external peripheral controlled by the Nucleo: the clock generator.

This external clock generator has a pin that must be turned on and off to enable or disable the clock generation for each test. The Engine oversees this operation, which is triggered by a command and that does not need a driver either.

How the Engine works and manages these commands will be described in detail in the next chapter.

Management unit

The management unit combines two different modules: one for executing actions and other to format output and input messages.

The unit receives messages from the PC according to a fixed protocol, interprets these messages and actualize the parameters of the driver that must execute the actions. After that, the output parameters are formatted and sent back to the PC following the same protocol.

The protocol establishes that every message must have a one-character header, followed by a three-character command, the parameters separated by commas, and finished by a '!' or '?' character. Every message part is separated by ':' and ended with a '\n' character added automatically by the PC.

<header>:CMD:<parameter, parameter>:!

Engine module

This module gets the message, formats it, checks if it is a correct message and an existing command, and stores it into a structure for the actions' functions to access them. The function Update() executes the desired actions and returns the results that are formatted back into one only message that will be sent back to the PC.

HEADER FILE

The header contains all the function prototypes and the definition for the number and length of the parameters. It includes all the headers of the drivers described above and the USB_OTG_HS.

It also contains the structures to store a message divided in parts, and others to store all the incoming and outgoing parameters in arrays of strings, in addition of the system status and the last character info. Splitting and storing the data travelling through the USART in message structures has two major advantages. First, the program synchronises the information that comes and goes with the execution of the Update() function, which takes all the parameters it needs from an incoming structure and stores back the parameters it generates in an outgoing structure. Second,

it allows the program to evaluate the incoming message and check every part is correctly formatted, and enhances the flexibility when sending the outgoing message, as every section of it can be modified separately.

```
typedef struct
{       char *header, *cmd, *params, *info;
} message;
```

```
typedef struct
{       /*Arrays of strings for formatted parameters*/
        char iparam[MAX_PARAMS][MAX_LENGTH];
        char oparam[MAX_PARAMS][MAX_LENGTH];
        /*Last character of message is a '?' (0) or a '<'(1)*/
        int lchar;
        SYS_StatusTypeDef status;
} System_Params;
```

SOURCE FILE

The source file contains all the functions and variables that allows the module to work independently of the main.c where it is used. It includes the header file of the message formatter (CMD).

The source also contains the initialisation of parameters structure and for incoming and outcoming messages.

```
message oMSG;
message iMSG;
System_Params ENG = {{{0}},{{0}},-1,SYS_OK};
```

The Update() function gets the message, and if it is correct (status == SYS_READY) executes the actions and sends back the information to the PC. If there is no incoming message (status == SYS_OK) the Update() function gets ignored. If the message is incorrect or there is a problem with any of the drivers, the system sends back an error message to the PC.

CMD module

This module formats the message by dividing it into parts and then merge them again to send back information to the PC. It also includes buffer management functions.

HEADER FILE

The header contains all the function prototypes and the incoming and outcoming message structures as external variables from the Engine module. It also includes the system parameters structure from the Engine module too.

SOURCE FILE

The source file contains all the functions and variables that allow the module to work independently of the main.c where it is used.

The functions in this module divide the incoming message after checking if the format is correct (see Annexes, page 74), also to divide the parameters by the commas, store the last character into the Engine structure and then to merge back the parameters and the output message. There is a function for clearing the output message structure as well.

STM32CubeIDE files

The only changes to be made in the generated files are located in the main.c.

We must change the initialisation function from ADC like is described above and call the Update() function in the while(1) loop.

For testing purposes, there are 4 variables defined for commented and uncommented parts of code that allows the user to test the drivers one by one using the debugging tools from STM32CubeIDE.

For testing only one driver TestMode must be 1. For using the Engine TestMode must be 0 and all driver tests 0.

Hardware

To manufacture the hardware of the test bench, including the electronics dedicated to signal processing and alimentation of the Nucleo and all peripherals, ECA hired a subcontractor. This company has been working alongside with us in the development of the test bench for the last two months of the internship.

Due to the reduced development time we finally had during the internship, this subcontractor is also going to design some of the low-level software for the Nucleo, as mentioned above in this document.

One of the challenges of this project was to make the test bench as flexible as possible to later be able to incorporate all this software and hardware. Although we have not fully participated in the development of this part, we provided our vision and recommendations for its design to the subcontractor.

Some examples of this are the filter for the sine wave generator in hardware, or the prototypes for the functions in the drivers to be programmed by them in software.

Filter for the Nucleo DAC

As the DAC is an integrated peripheral that depends on the chosen sampling rate, and the output signal is very low voltage, it comes with a noticeable amount of noise.

To avoid that as much as possible, a low pass filter is added between the Nucleo output and the tension follower. The selected cut-off frequency is 1MHz so the filter doesn't interfere with the desired signal.

Knowing the transference function of this RC filters is the following one:

$$F(S) = A_0 \frac{1}{1+\tau s} = A_0 \frac{1}{1+\frac{s}{\omega_c}}$$

And that $1/\omega c = fc = RC$, to cancel all the frequencies over 1MHz a R=100Ω and a C=10nF have been selected.

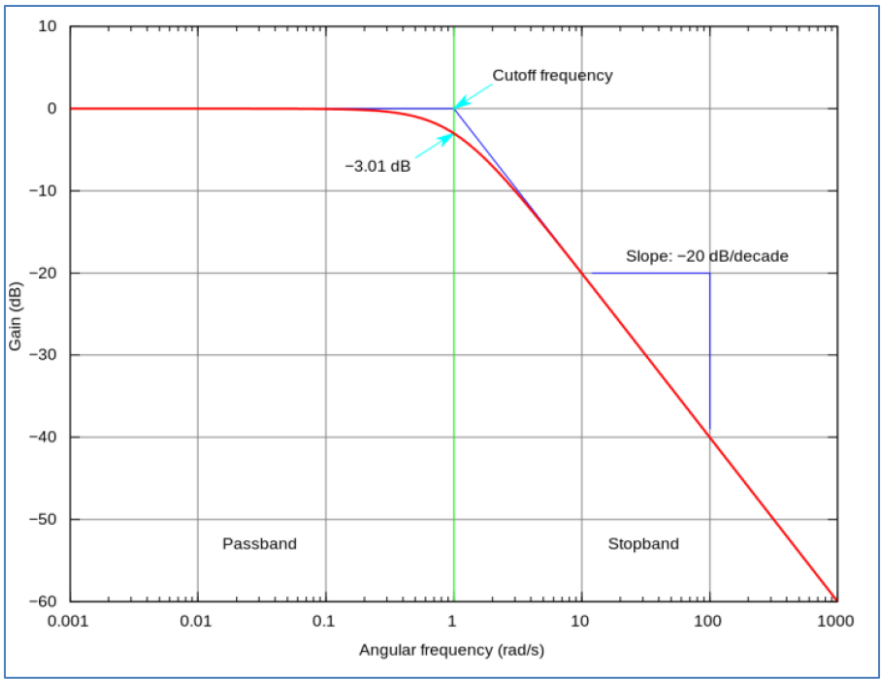


Figure 30: Low pass filter Bode diagram

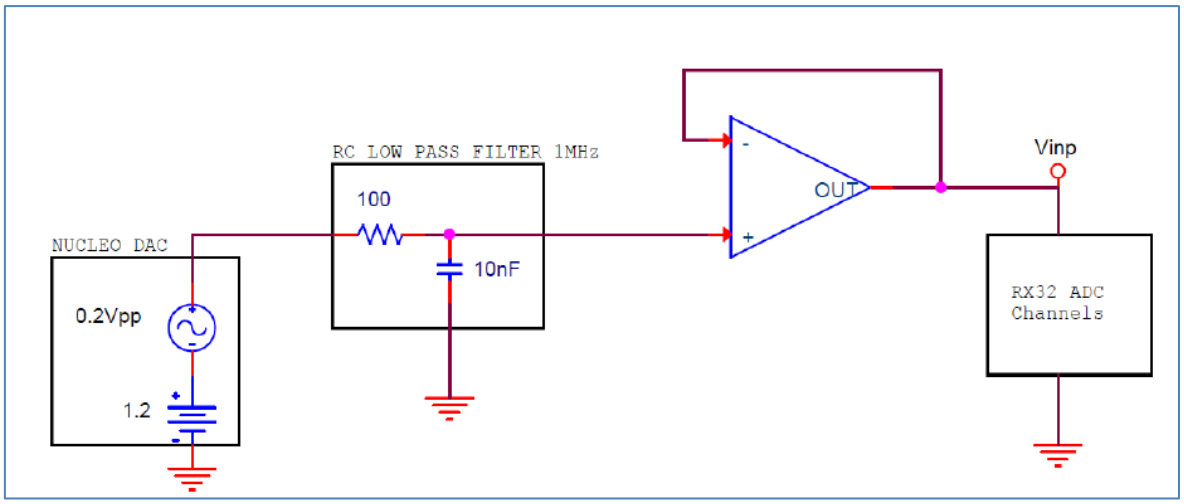


Figure 31: Circuit schematic for the DAC sine wave generation

In the board developed for the thesis, the circuit above is included till before the low pass filter. The rest of the components will be provided by the subcontractor.

The improvement of the signal is quite noticeable. Pictures of it are included in the annexes (pg. 74).

Tension divisor for DC/DC converter peripheral

As the I2C driver will be developed after our internship is finished, an alternative was proposed to control the power peripheral. As we explained before, the DC/DC converter can be controlled using the RUN0 and RUN1 pins, that withstand voltage between 1.1V and 1.35V.

The Nucleo board output pin voltage is 3.3V, so a tension divisor must be installed between the Nucleo power pin and the DC/DC RUN pins.

The tension out of a tension divisor can be calculated as follows:

$$V_o = V_i \frac{R_2}{R_1 + R_2}$$

Where V_0 and V_i are the output and input voltages, 1.2V and 3.3V respectively. Selecting R_1=180Ω and R_2=100Ω, V_0=2.12V, that belongs to the desired range.

The divisor is implemented as shown below and soldered to the test bench prototype board.

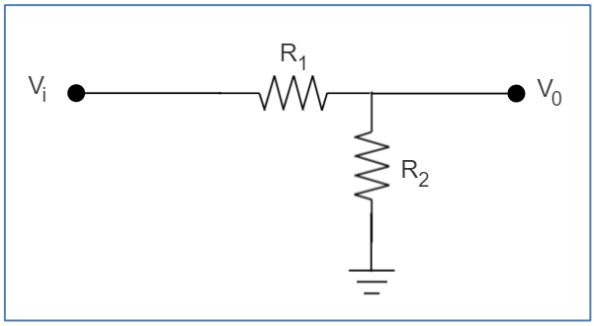


Figure 32: Tension divisor electrical schematic

Test bench prototype

As the final hardware will be manufactured with other parts, a prototype was developed for testing and presentation purposes.

This prototype consists of a test PCB that allows to connect the Nucleo and connect all the peripherals to it, using as little cable as possible. The Nucleo pins are soldered to the board, that includes the sine wave generator filter, the tension follower, and two LEDs for the GPIO test. The power and communication are directly connected to the Nucleo through its own connectors.

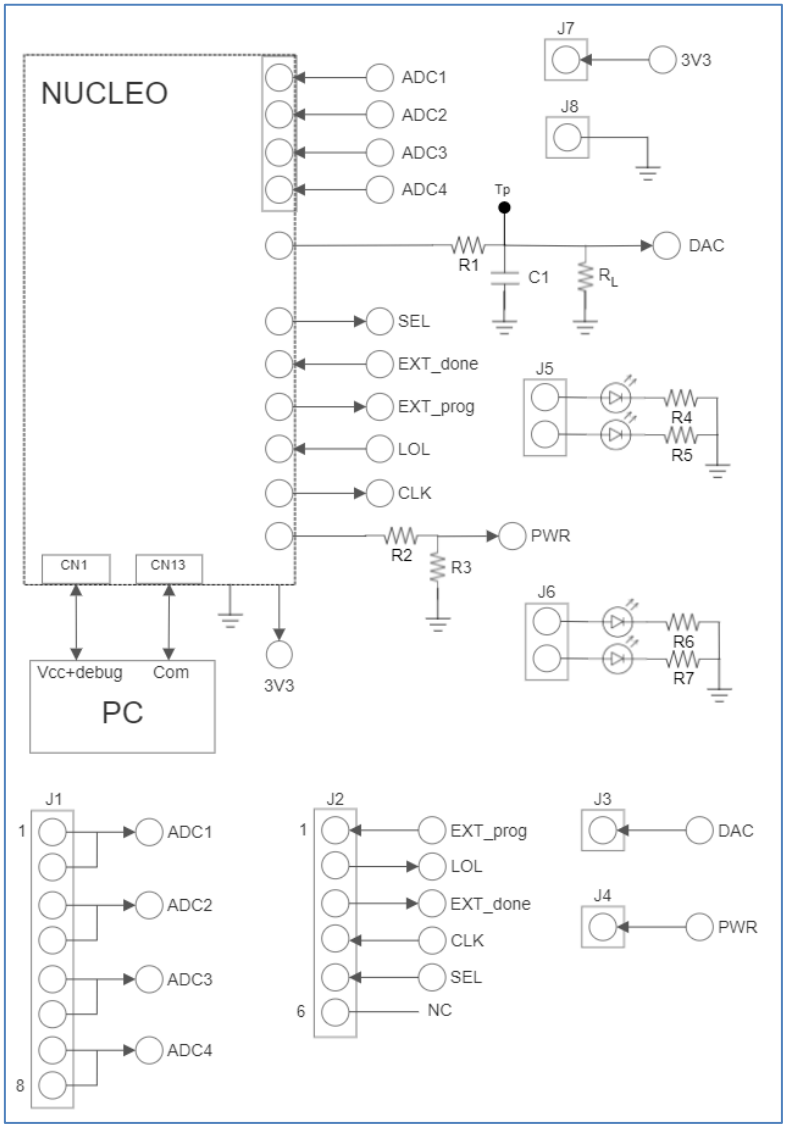


Figure 33: Test bench prototype electric diagram

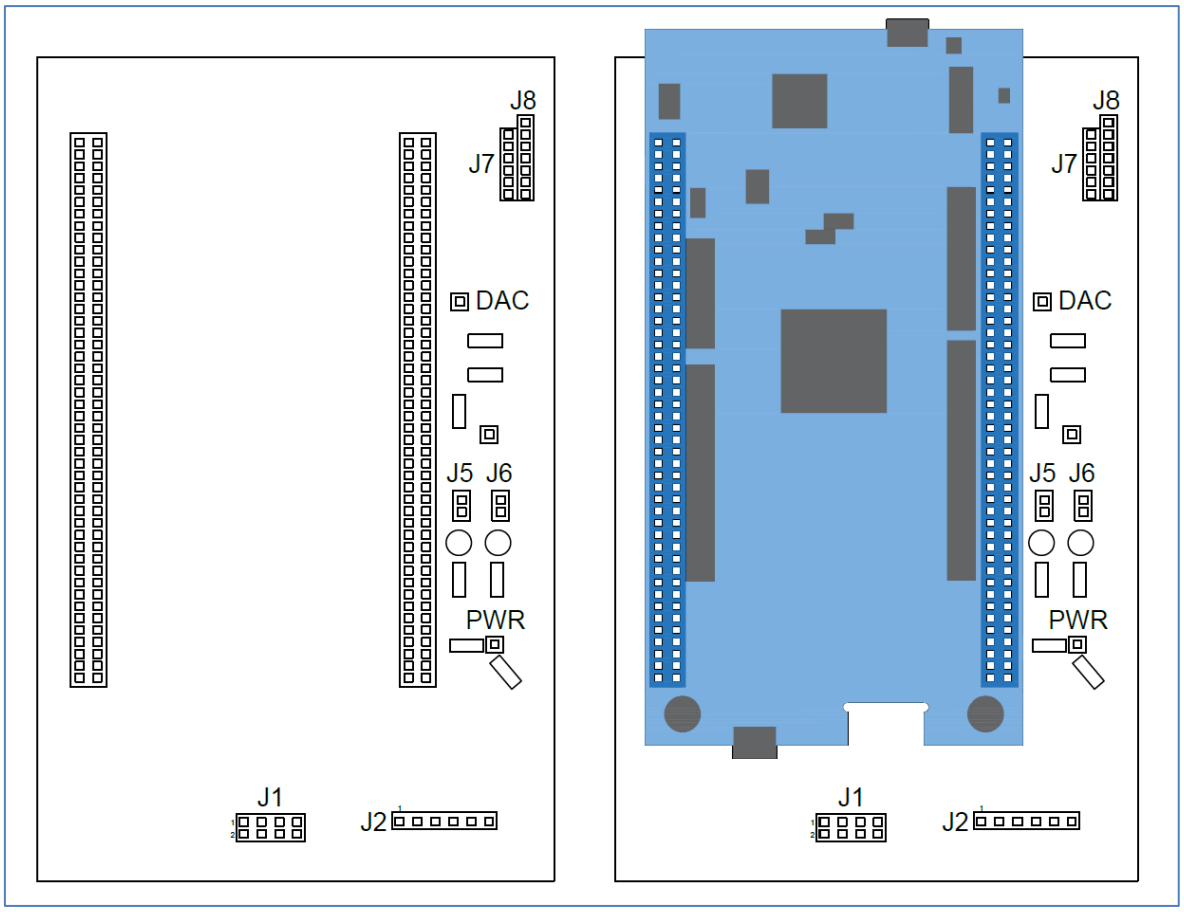


Figure 34: Test bench prototype PCB schematic with and without the Nucleo plugged in.

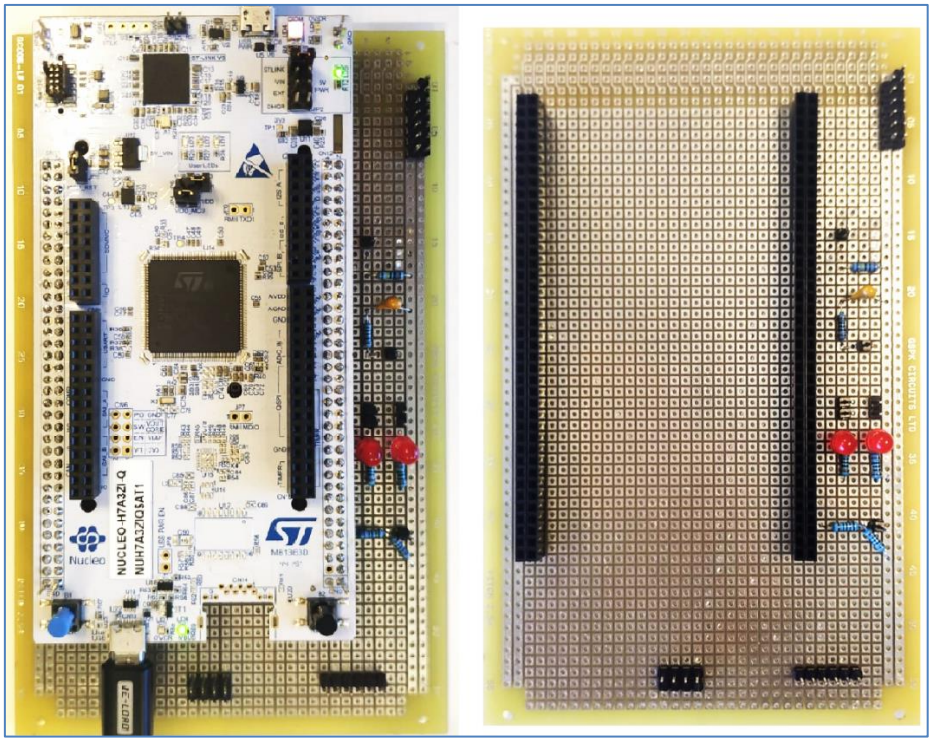


Figure 35: [Photograph] Test bench prototype PCB with and without the Nucleo plugged in.

5 General conclusion

Technical conclusion

After some delays and re-scopes during the design and elaboration phases, the deadlines were finally met, and the project was finished and closed before the end of our internship at ECA Robotics.

We have achieved the main focus of our project, that was to test the concept and asses the feasibility of the idea. The development of an automated test bench is not only **feasible** but can be **easily adapted** to other units that have to be tested. The final product has proven to be **user friendly** and has correctly executed the two most important tasks: preform tests **automatically** and **report back** the results to the operator.

In addition, other functionalities such as a working database to store the results was implemented, making the test bench prototype a more complete groundwork for further development.

The process and final product were complemented by a vast amount of documentation, including manuals, schematics and diagrams that will help the developers who will take over this project to understand the work we have done.

Consequently, we can assure that the targets of the project were accomplished, and that the final product will be used as a part of the production process of the company.

Personal conclusion

Working at ECA Robotics for the past 14 weeks allowed us not only to learn more about software and tools, but also about how to integrate software parts together and deepen in aptitudes like interpreting software documentation. We also acquired fluency in writing efficient and useful documentation following the company templates.

We are now familiarized with how a technology company works, the process and protocols followed and what being part of an engineering team is like, as we have been treated as full-fledged engineers during our internship.

In a more personal way, this project made us face obstacles in our way of working. We learned to deal with frustration, deadlines and problems that don’t have an easy solution. Throughout the internship we had to be flexible and adapt to the circumstances, as we had to re-scope the project and reshape it with the new input we received from the team and the subcontractor. We acknowledge we had to be more communicative with our team and it is one of the main things we have personally learned.

Overall, we think of this experience as a crucial part of our development as engineers and future employees, and furthermore as a great opportunity to become more aware of our strengths and weaknesses

6 Bibliography

Ainslie, M. (2016). *Principles of Sonar Performance Modelling (Springer Praxis Books)* (Softcover reprint of the original 1st ed. 2010 ed.). Springer. https://doi.org/10.1007/978-3-540-87662-5

Belgium Naval & Robotics. (2020, October 16). *The Stand-off MCM Capability* [Video]. YouTube. https://www.youtube.com/watch?time_continue=12&v=BwlmEvILabk&feature=emb_logo&ab_channel=BelgiumNaval%26Robotics

Belgium Naval & Robotics. (2021, March 31). *The Netherlands Ministry of Defence placed an order for a drone system within the Mine Countermeasures capability replacement of the Royal Netherlands Navy* [Press release]. https://www.belgium-naval-and-robotics.be/the-netherlands-ministry-of-defence-placed-an-order-for-a-drone-system-within-the-mine-countermeasures-capability-replacement-of-the-royal-netherlands-navy/

Blondel, P. (2009b). *The Handbook of Sidescan Sonar* (2009th ed.). Springer.

Cambridge University. (n.d.). drone. In *Cambridge Advanced Learner's Dictionary & Thesaurus*. Cambridge University Press. Retrieved 25 April 2022, from https://dictionary.cambridge.org

Carton, G., DuVal, C., Trembanis, A., Edwards, M., Rognstad, M., Briggs, C., & Shjegstad, S. (2017, July). *Munitions and Explosives of Concern Survey Methodology and In-field Testing for Wind Energy Areas on the Atlantic Outer Continental Shelf*. US Department of Interior, Bureau of Ocean Energy Management. https://doi.org/10.13140/RG.2.2.25765.60645

Chatham House & The Royal Institute of International Affairs. (2014, October). International Law Applicable to Naval Mines. In L. Arimatsu & British Academy (Eds.), *Summary*. Chatham House.

Content Engine LLC (translated by). (2022, March 21). Bulgaria warns of drifting naval mines in the Black Sea. *CE Noticias Financieras, English Ed.; Miami*. https://www.proquest.com/wire-

feeds/bulgaria-warns-drifting-naval-mines-black-sea/docview/2641828238/se-2?accountid=14712

ECA Group. (2018, October 9). *Naval Group and ECA Group offer an innovative mine hunting solution to Belgium and the Netherlands* [Press release]. https://www.ecagroup.com/en/financial/naval-group-and-eca-group-offer-an-innovative-mine-hunting-solution-to-belgium-and-the-netherlands

Fitzgerald, C. (2022, January 21). *The Evolution of Sea Mines and Their Impact on War*. WAR HISTORY ONLINE. Retrieved 4 May 2022, from https://www.warhistoryonline.com/instant-articles/sea-mine-evolution.html

GLib.idle_add. (2022, May 26). In *GTK*. https://docs.gtk.org/glib/func.idle_add.html

GLib.timeout_add. (2022, May 26). In *GTK*. https://docs.gtk.org/glib/func.timeout_add.html

Haines, S. (2014). *1907 Hague Convention VIII Relative to the Laying of Automatic Submarine Contact Mines* (No. 90). Stockton Center for the Study of International Law. https://digital-commons.usnwc.edu/ils/vol90/iss1/7/

HarperCollins Publishers. (n.d.-a). beam. In *Collins English Dictionary*. Retrieved 24 May 2022, from https://www.collinsdictionary.com/dictionary/english/beam

HarperCollins Publishers. (n.d.-b). ping. In *Collins English Dictionary*. Retrieved 24 May 2022, from https://www.collinsdictionary.com/dictionary/english/ping

National Oceanic and Atmospheric Administration [NOAA]. (2021, February 26). *What is a hydrophone?* National Ocean Service. Retrieved 24 May 2022, from https://oceanservice.noaa.gov/facts/hydrophone.html#:%7E:text=A%20hydrophone%20is%20an%20underwater,reproduction%2C%20and%20to%20seek%20prey.

National Oceanic and Atmospheric Administration [NOAA]. (n.d.-a). *Exploration Tools: Synthetic Aperture Sonar: NOAA Office of Ocean Exploration and Research*. National Oceanic and Atmospheric Administration (NOAA) Ocean Exploration. Retrieved 28 May 2022, from https://oceanexplorer.noaa.gov/technology/sonar/sas.html

NATO. (n.d.-a). bottom mine. In *NATO Term. The Official NATO Terminology Database*. Retrieved 9 May 2022, from https://nso.nato.int/natoterm/Web.mvc

NATO. (n.d.-b). influence mine. In *NATO Term. The Official NATO Terminology Database*. Retrieved 9 May 2022, from https://nso.nato.int/natoterm/Web.mvc

NATO. (n.d.-c). mine. In *NATO Term. The Official NATO Terminology Database*. Retrieved 4 May 2022, from https://nso.nato.int/natoterm/Web.mvc

NATO. (n.d.-d). moored mine. In *NATO Term. The Official NATO Terminology Database*. Retrieved 6 May 2022, from https://nso.nato.int/natoterm/Web.mvc

Naval History and Heritage Command. (2021). *Naval Mine Warfare*. Retrieved 4 May 2022, from https://www.history.navy.mil/browse-by-topic/exploration-and-innovation/naval-mine-warfare.html

Siciliano, B., & Khatib, O. (2016). *Springer Handbook of Robotics*. Springer Publishing.

SyndiGate Media Inc. (2022, May 6). Turkey spots naval mine off Black Sea after similar discoveries. *Ghana News Agency (GNA); Accra*. https://www.proquest.com/wire-feeds/turkey-spots-naval-mine-off-black-sea-after/docview/2647395835/se-2?accountid=14712

The Editors of Encyclopædia Britannica. (2019). sonar. In *Encyclopædia Britannica*. Encyclopædia Britannica Inc. https://www.britannica.com/technology/sonar

U.S. Department of Defense. (2016). *Department of Defense Dictionary of Military and Associated Terms*. U.S. Government. https://www.defense.gov/

U.S. Marine Corps & U.S. Navy. (1996, August). *NWP 3–15/ MCWP 3–3.1.2 . Mine Warfare* (PCN 143 000008 00). Naval Warfare Publications.

Vynckier, I. (2022). Kennismakingsverslag ECA robotics. CONFIDENTIAL

7 Annexes

Timeline

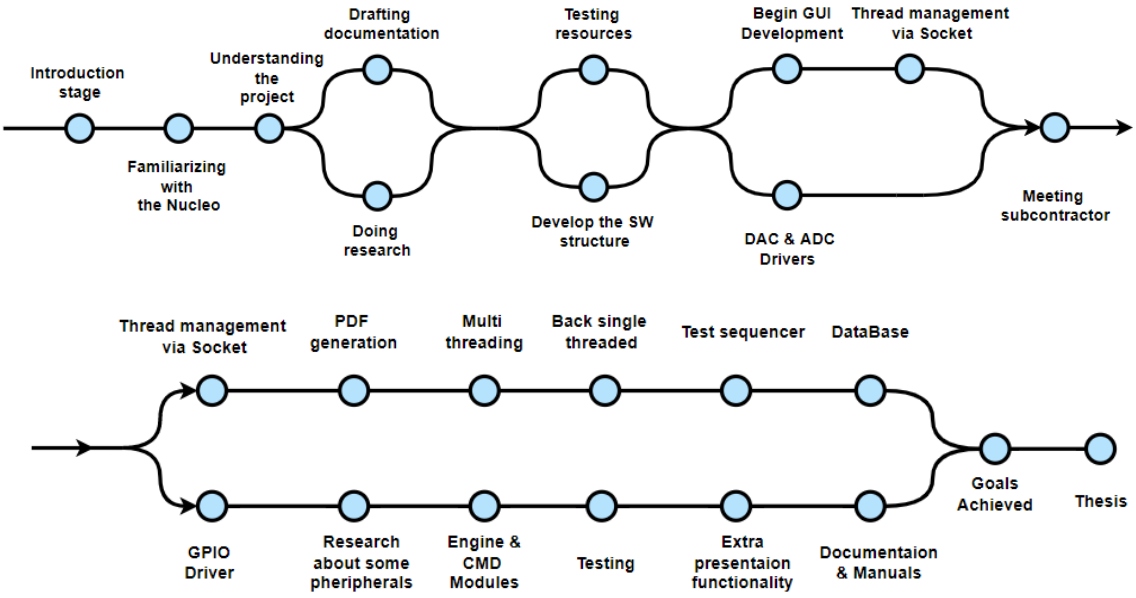


Figure 36: Project timeline

UML diagrams

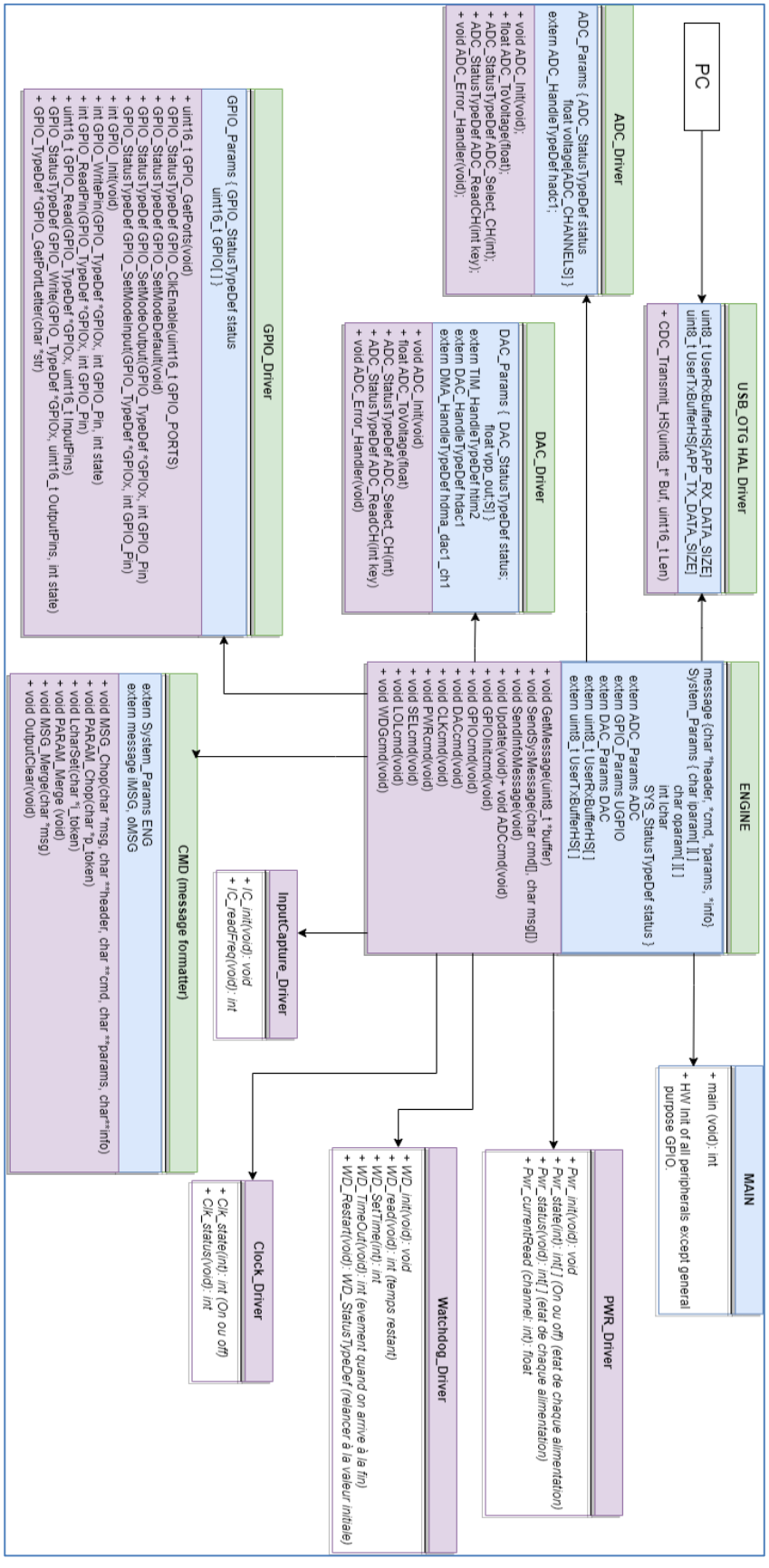


Figure 37: Nucleo UML diagram

GPIO read function

```
/**
 * @brief Read all set pins. Set all selected pins as inputs + read all IDR registers.
 * @param GPIO_TypeDef pointer to GPIO port.
 * @param uint16_t pins to read.
 * @retval Stored pin state: 0 for OFF 1 for ON.
 */
uint16_t GPIO_Read(GPIO_TypeDef *GPIOx, uint16_t InputPins)
{
	uint16_t pin_result = 0;
	for (int i=0; i<NUMBER_OF_PINS; i++){
		if ((InputPins & PINS_MASK[i]) == PINS_MASK[i])
		{
			if (GPIO_ReadPin(GPIOx, i))
				pin_result |= (1U << i); //Puts a 1 in the pin position if it is on
			else
				pin_result &= ~(1U << i);//Puts a 0 in the pin position if its off
		}
		else{}
		UGPIO.status = GPIO_NOT_READY;
	}
	UGPIO.status = GPIO_READY;
	return pin_result;
}
```

Sine wave form function

```
/**
 * @brief Generate a sine wave form with an offset of 1.2V. Add 12 to the sin and multiply it
 * by the digital desired voltage/2.
 * @param None.
 * @retval None.
 */
void DAC_SineWave(void)
{
	for (int i=0; i<DAC_SAMPLES;i++)
	{
		sine_samp[i] = ((sin(i*2*M_PI/DAC_SAMPLES) + 12)*(DAC_ToDig(DAC.vpp_out)/2));
	}
}
```

Watchdog driver substitute function

```
/**
 * @brief Set/reset and read all the pins in the selected port.
 * @param None.
 * @retval None.
 */
void WDGcmd(void)
{
	/*WARNING: This driver will be provided. For DEMO purposes only*/
	/*'!' Sets output pin ON and check the input pin right after, and after a delay*/
	if (ENG.lchar){
		int delay = atoi(ENG.iparam[0]);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET); //Set pin
		int time = HAL_GetTick();//The actual time its saved
		sprintf (ENG.oparam[0], "%d,", HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_14)); //First read
		while (HAL_GetTick() - time < delay){} //Waiting time
		sprintf (ENG.oparam[1], "%d,", HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_14));//Read after delay
		SendInfoMessage();
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET); //Reset pin
	}
```

Incoming message divider function

```
/**
 * @brief Divide the incoming message into tokens and check if it has the correct format.
 * @param Buffer from USART.
 * @param Header token.
 * @param Cmd token.
 * @param Params token
 * @param Info token.
 * @retval None.
 */
void MSG_Chop(char *msg, char **header, char **cmd, char **params, char**info)
{
	*header = strtok(msg, ":");
	if (!strcmp(*header, "C")){
		*cmd = strtok(NULL, ":");
		if (sizeof(*cmd) == 4){
			*params = strtok(NULL, ":");
			*info = strtok(NULL, ":");
		}
		else
			ENG.status = SYS_InvalidMessage;
	}
	else
		ENG.status = SYS_InvalidMessage;
}
```

DAC signal after low pass filter

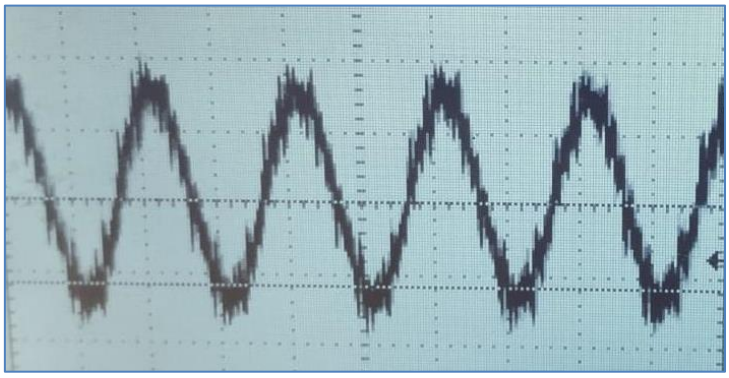


Figure 38: Output signal with no filter

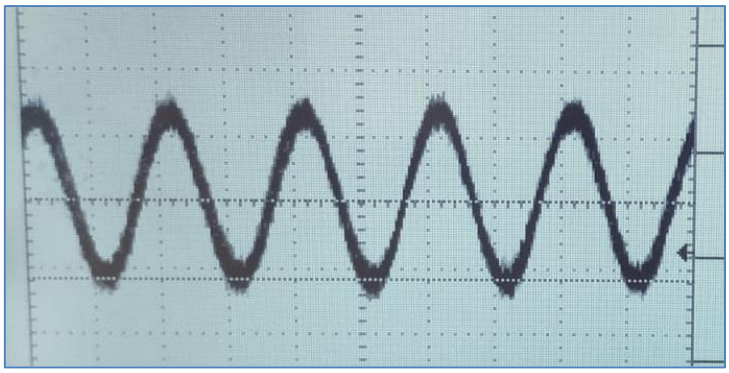


Figure 39: Output signal with RC low pass filter

MCU-PC Communication Comparison

Published with ECA Robotics authorization.

UMISAS

RX32 TEST BENCH

MCU-PC COMMUNICATION COMPARISON: SPI/I2C/UART/USART ANALISYS

ORIGIN	DOCUMENT N°	ISSUE
MOU	262-1-XX-XXX	$04

NB. OF PAGES	WRITTEN BY: Luján María & Vynckier Ilias	COPY
18	DATE: 24/02/22	N° 1 / 1

Version	Date	Autor	Modification
$00	14/02/2022	I. V & M. L	Creation
$01	16/02/2022	I.V & M.L	Updating after feedback
$02	18/02/2022	I.V	Porting original document to template
$03	21/02/2022	M.L	Updating and extracting to PDF
$04	24/02/2022	I.V	Updating after Feedback

	Title	Date and Signature
Written by :	*Engineer* *Ilias Vynckier* *María Luján*	24/02/2022
Checked by :	*System engineer* *Bastien Loisy*	25/02/2022
Approved by :		

Internal distribution:
External distribution:

TABLE OF CONTENTS

List of figures

1. INTRODUCTION

This document describes the principal communication protocols and peripherals that can be used in the RX32 test bench project for communication between MCU and the PC. The analysis is only for the communication protocols allowed by the FTDI converter and the MCU itself.

1.1. REFERENCE DOCUMENTS

	Reference	Version	Description
[DA01]	262-1-93-171	00	SPECIFICATIONTECHNIQUEOUTILLAGE DE TEST RX32

**last applicable version*

1.2. ACRONYMS

Acronym	Definition	Commentaries
ADC	Analog Digital Converter	
DAC	Digital Analog Convertor	
FTDI	Future Technology Devices International	
I2C	Inter-Integrated Circuit	
JTAG	Joint Test Action Group	
MCU	Microcontroller Unit	
SPI	Serial Peripheral Interface	
UART	Universal Asynchronous Receiver-Transmitter	
USART	Universal Synchronous Asynchronous Receiver-Transmitter	
USB	Universal Serial Bus	

2. SPI

2.1. DEFINITION

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication. SPI is a full-duplex with a slave-master architecture with a single master. Device selection is done by sending a '0' with the slave select line to the desired device (Wikipedia contributors, 2021).

The SPI uses four logic signals:

- SCLK: Serial Clock
- MOSI: Master Out Slave In
- MISO: Master In Slave Out
- SS: Slave-Select or Chip-Select

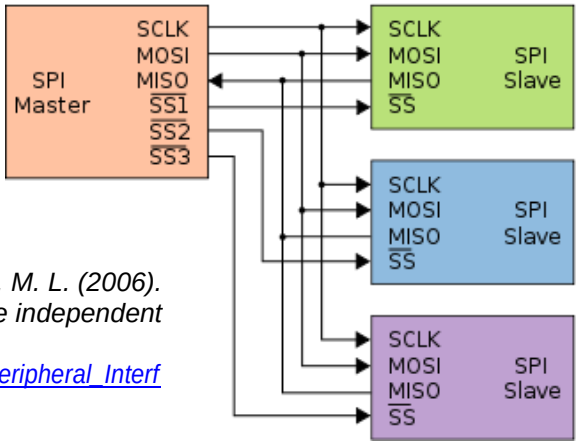


Figure 1: Wikipedia, & Burnett, C. M. L. (2006). Typical SPI bus: Master and three independent slaves [Figure]. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

2.2. DATA TRANSMISSION

- The master configures the clock and the polarity.
- The master selects the device with the SS line, sending a falling edge.
- During each SPI clock cycle, full-duplex transmission occurs.

(Wikipedia contributors, 2021 & Exostivlabs, n.d.).

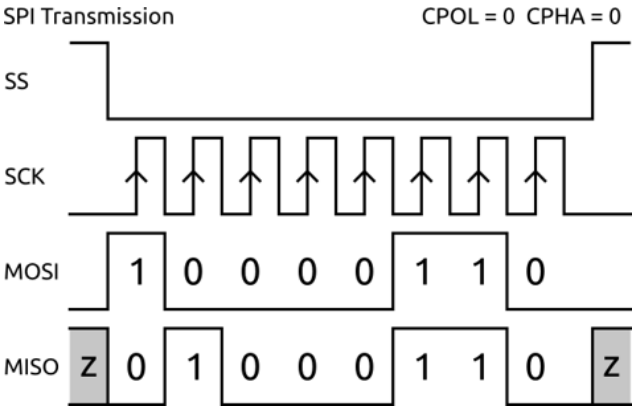


Figure 2: MathWorks. (2021). SPI transmission [Figure].
https://ww2.mathworks.cn/help/supportpkg/raspberrypi/ug/support-spi-communication.html

2.3. PROS AND CONS

Pros :

- Full-Duplex.
- No maximum clock speed, high-speed communication.
- Complete protocol flexibility.
- No transceivers needed.
- No need for device addresses.
- No data size limit.

Cons :

- Only one master.
- Four cables are needed.
- No flow control or error check.
- The interrupt operation can only be performed through additional signal lines.
- No acknowledgement mechanism confirms whether the data are

received.(SPI - Introduction to Serial Peripheral Interface, 2021)

3. I²C

3.1. DEFINITION

Is a synchronous, multicontroller/multitarget (controller/target), packet switched, single-ended, serial communication bus. (Wikipedia contributors, 2022)

3.2. BUS TOPOLOGY

I²C only needs two lines to communicate SDA (serial data line) and SCL (serial clock line).
There is a controller/target (master/slave) relation between two devices.

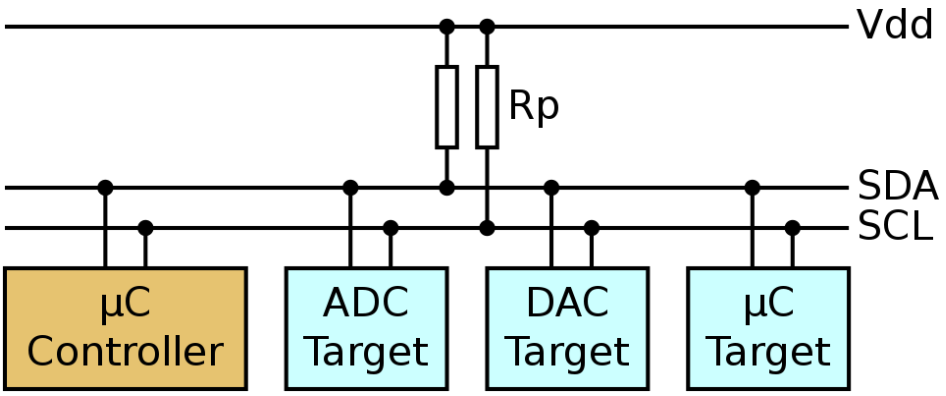


Figure 3Mathias, T. M. (2021, October 28). Bus topology [Schematic].
https://en.wikipedia.org/wiki/I%C2%B2C#/media/File:I2C_controller-target.svg

3.3. SPEED

- Three modes:
  - standard: 100kbps
  - fast: 400kbps
  - high speed: 3.4Mbps
  - extra (only for some I²c variants):
    - low speed mode: 10 kbps
    - fast mode +: 1 Mbps

3.4. RESOURCES

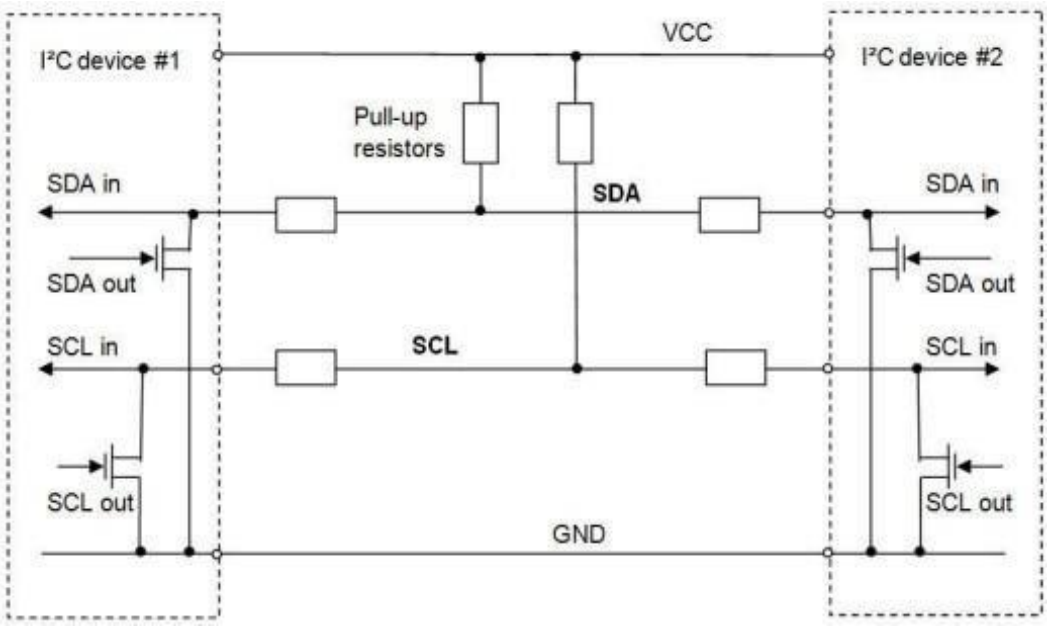


Figure 4 I2C bus schematic with two devices connected. (n.d.). [Schematic]. *https://www.exostivlabs.com/files/documents/Introduction-to-I2C-and-SPI-Protocols.pdf?/article/aa-00255/22/introduction-to-spi-and-ic-protocols.htm* [1]

I2c does not need many extra components, just some pull-up resistors for 'clearer' communication.

3.5. CONNECTIONS

As said in the bus topology I²C only needs 2 lines to work. The number of devices that you can connect is limited only by the number of different addresses you can find.

3.6. DATA SIZE

Data on the I²C bus are transferred in 8-bit packets. There is no limitation on how many bytes one can send. Each byte just has to be followed by an acknowledge bit.
The master can initiate communication with a START condition, ends with a STOP condition, and selects the target device with its unique address.

[1] SDA and SCL are connected to VCC trough pull-up resistors. Each device controls de bus lines outputs with open drain buffers.

START	Slave address	Rd/nWr	ACK	Data	ACK	Data	ACK	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 1: writing 2 byte to a slave. The data put on the bus by the master are shaded.

START	Slave address	0	0	Data	0	Data	0	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 2: reading 2 bytes from a slave. The data put on the bus by the master are shaded.

START	Slave address	1	0	Data	0	Data	1	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Figure 5 I2C data example. (n.d.). [Illustration]. https://www.exostivlabs.com/files/documents/Introduction-to-I2C-and-SPI-Protocols.pdf?/article/aa-00255/22/introduction-to-spi-and-ic-protocols.html

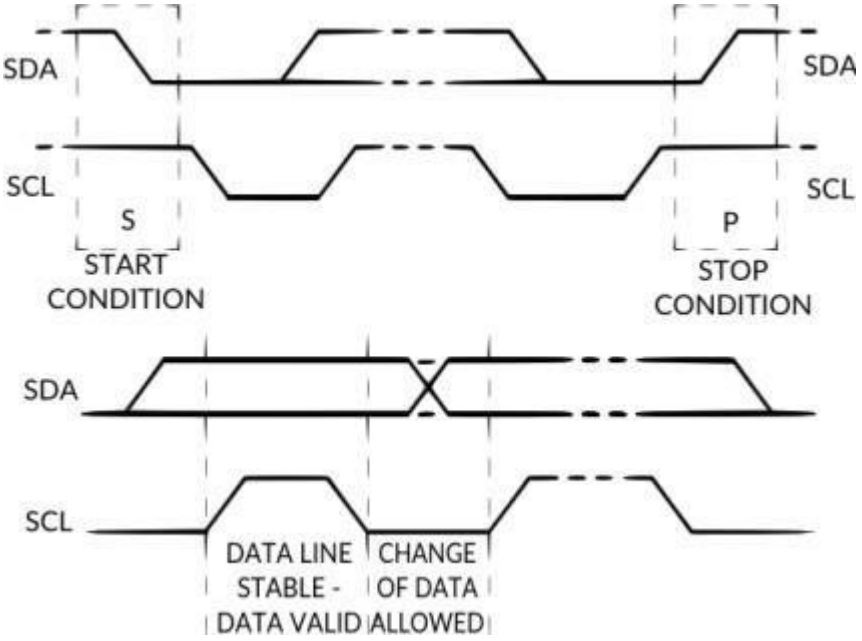


Figure 6 START STOP. (n.d.). [Illustration]. https://www.exostivlabs.com/files/documents/Introduction-to-I2C-and-SPI-Protocols.pdf?/article/aa-00255/22/introduction-to-spi-and-ic-protocols.html

3.7. COST

I²C is very inexpensive to implement and there are many devices on the market.

3.8. PROS AND CONS

Pros :

- 1 bus can connect many slaves.
- Just 2 lines.
- Low cost.
- Industry used.
- Synchronous.
- Robust.

Cons :

- Limited addresses
- Synchronous
- Meant for 'inside the box' communication
- Does not make sense for communication between a PC and microcontroller

4. USART

4.1. DEFINITION

This hardware communication protocol uses asynchronous or synchronous serial communication with configurable speed.
It allows for the connection between only two devices using two cables for data transmission/reception (TX and RX) and one additional cable for the clock if it is synchronous communication.

4.2. BUS TOPOLOGY

The TX pin of the transmitting device is connected to the RX pin of the receiving device, and the other way around for the other two pins:

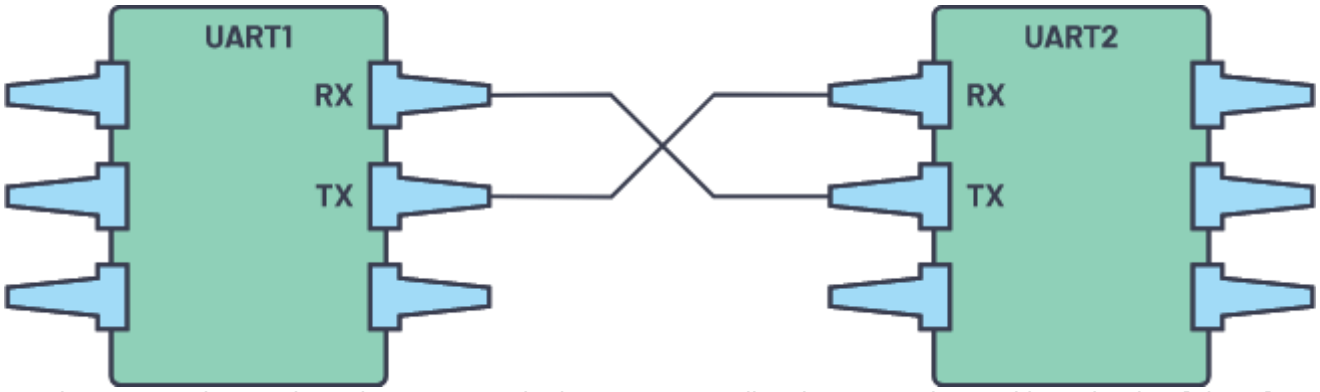


Figure 2: Analog Devices. (2020, December). Two UARTs directly communicate with each other [Figure]. Analog Dialogues. https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html. ©2020 Analog Devices, Inc. All rights reserved.

The transmitting device converts parallel data to serial data and transmits it to the receiving device, which will convert the received data back into parallel form.

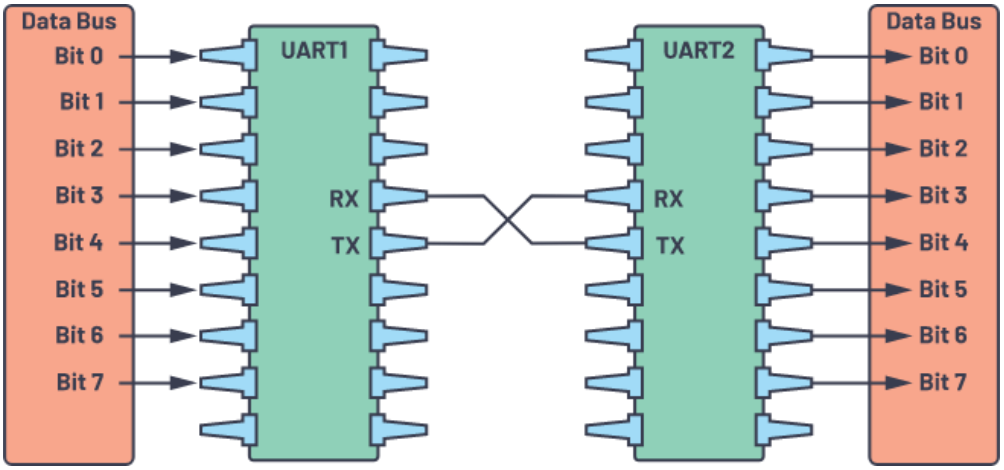


Figure 3: Analog Devices. (2020, December). UART with data bus [Figure]. UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter. https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html ©2020 Analog Devices, Inc. All rights reserved.

For using asynchronous communication, a clock signal must also be connected.

4.3. DIFFERENCES BETWEEN UART AND USART

The USART protocol can use an additional line for the clock that synchronizes both connected devices, while the UART protocol does not have that option. UART can only transmit the data asynchronously, generating a bitstream based on its own clock signal, and the receiver does the same with its internal clock signal. To work properly, both devices must be configured with the same or very similar baud rate (more than a 90%) baud rate (Legaspi & Peña, 2020).
Apart of the synchronous possibility (which can make USART communication much faster than the asynchronous one) the other major difference between them is the number of protocols the peripheral can support. The USART peripheral can generate data in many more standard protocols than UART. (Beningo, 2015).

4.4. USART VS. UART

The USART peripheral has also the same asynchronous capabilities as UART, so is the most complete and complex option for serial communication. In STM32 microcontrollers are usually meant for high speed and 'high' energy consumption during short periods of heavy data transmission. UART, on the other hand, is usually used in low-power modes with low-speed communication.

4.5. DATA TRANSMISSION IN ASYNCHRONOUS MODE

Data transmission mode is a serial packet usually made of a start bit, a data frame, a parity bit, and a stop bit (Legaspi & Peña, 2020).

Figure 4:Analog Devices. (2020, December). UART packet [Figure]. UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter. https://www.analog.com/en/analog-dialogue/articles/uart-a-hard ©2020 Analog Devices, Inc. All rights reserved.

- Start bit: the transmission line is usually held at high voltage. To let the receiving device know that there are data available to read, the transmitting device sends one falling edge to the other device.
- Data frame: the real data that are sent to the receiver. The data frame length is usually between 5 and 8 bits. It can be 9 when there is no parity bit. Generally, the LSB of the data is transmitted first.
- Parity bit: This bit allows the receiver to ensure that the data received is correct. It can only detect errors in 1,3,5,7 or 9 bits (Brunete, 2021)
- Stop bit: 2-bit length, but usually only one is used. To stop the transmission, the UART keeps the data line on high voltage.

(Agarwal, 2020)

4.5.1. Speed

The transmission speed is also called the Baud rate.
The baud rate is the rate at which information is transferred in a communication channel. In the context of serial ports, “9600 baud” means that the serial port is capable of transferring a maximum of 9600 bits per second (Wilcox, 2020).
The baud rate can be chosen and can be unique to the application, but there are 'standard' baud rates that are expressed in Baud or Bits/s. These are:

- 1200
- 2400
- 4800
- 9600
- 19200
- 38400
- 57600
- 115200

4.5.2. Pros and Cons

Pros :

- Only two wires are needed (in asynchronous communication).
- Allows error check (parity bit).
- Asynchronous/synchronous communication can be chosen, and the baud rate for the asynchronous one can be configured.
- Compatible with most devices.
- USB-UART interface needed for communication included in most devices.
- Freedom in message forms (plain text, hex, Unicode, …).

Cons :

- Baud rates for both devices must be 90% similar for asynchronous communication.
- Noise-sensitive signal and susceptible to degradation (can be solved using higher voltages and differential signals) (Brunete, 2021).
- Point-to-point topology does not support a large number of nodes (solved by using a multi-drop or multi-pronged approach) (Brunete, 2021).

4.6. ASYNCHRONOUS / SYNCHRONOUS ON NUCLEO BOARD

ASYNCHRONOUS	SYNCHRONOUS
Stop, start, and parity bit available, data length configurable to 7, 8 or 9 bits.	No clock pulses on stop and start bits. Software option to send a clock pulse on the last data bit.
Fixed data rate	Data rate not fixed
Usually transmitted in blocks	One byte at a time
Lower speed	Higher speed
Full-duplex	Half-duplex

Table 1: Comparison between synchronous and asynchronous UART communication in the STM32F7A3ZI MCU. (STMicroelectronics, 2022)

5. Summary

	SPI	I2C	UART/USART
SPEED	High (no max clock limit)	Mid	Low
N° OF CONNECTIONS	4	2	2/3 (depending on Sync/Async)
FLOW CONTROL/ ERROR CHECK	no	yes	Yes (UART)
MASTER-SLAVE	1 master many slaves	many masters / many slaves	direct connection (two devices only)
DATA LENGTH	no size limit	8-bit packets	12 (includes start, stop, data and parity bits) Data configurable between 7, 8 and 9 bits
COST OF ADAPTERS	Low	Low	None (already in the Nucleo board)
N° OF DEVICES	depends on SS lines	∞ (limited by the available unique addresses)	2
PROTOCOL FLEXIBILITY	Loose	strict (start, address, R/W, ack, data, stop)	Less strict (start, data, parity, stop in UART)

Table 2: Summary of characteristics of the main serial communication protocol available.

6. Recommendation

Why use asynchronous communication/USART:

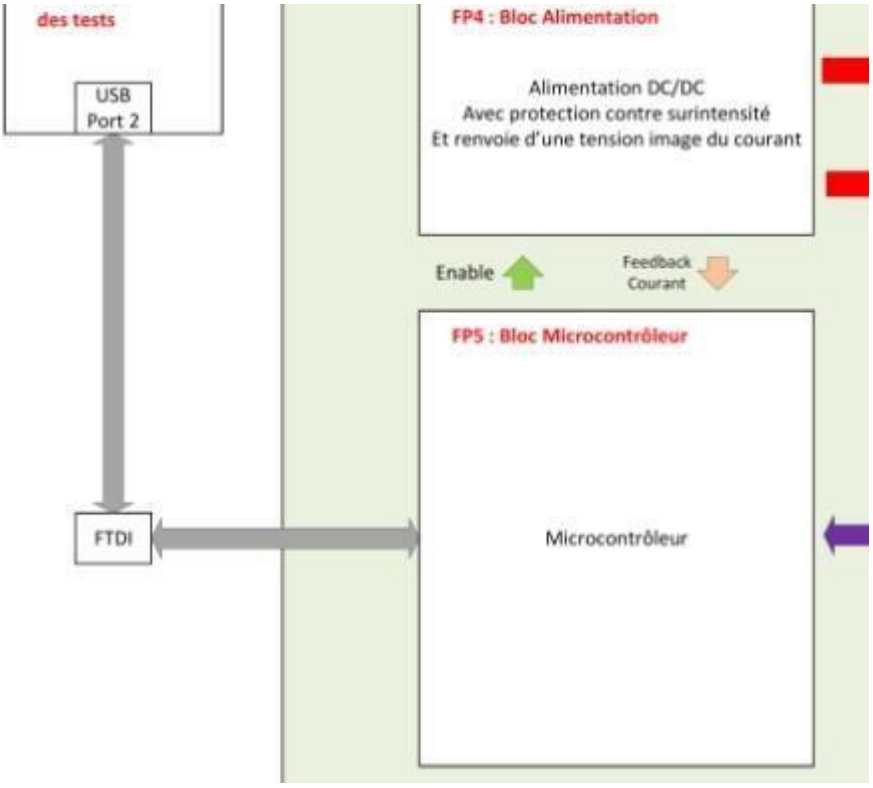


Figure 13: Part of the test bench synoptic

As you can see in the image above, we believe that the proposed option was to use an FTDI adapter/converter from USB to I2C/JTAG/… to make communication with the microcontroller possible.
We think that there is no need to use such a device to communicate between the PC and Microcontroller because there are already technologies available on the Microcontroller that allow us to do that asynchronously instead of synchronously.
We were baffled by this proposal, because it is like communicating by scanning a handwritten letter and sending it by e-mail instead of just an e-mail. You can definitely do that and it will work, but why?
The SPI protocol lacks flow control/error check features, and as there is only one connected device at a time, the no-address characteristic will not be useful. The same with I2C: there is plenty of functionality for multiple master-slave mode that is not necessary, and high-speed applications are not part of this project (as we know).

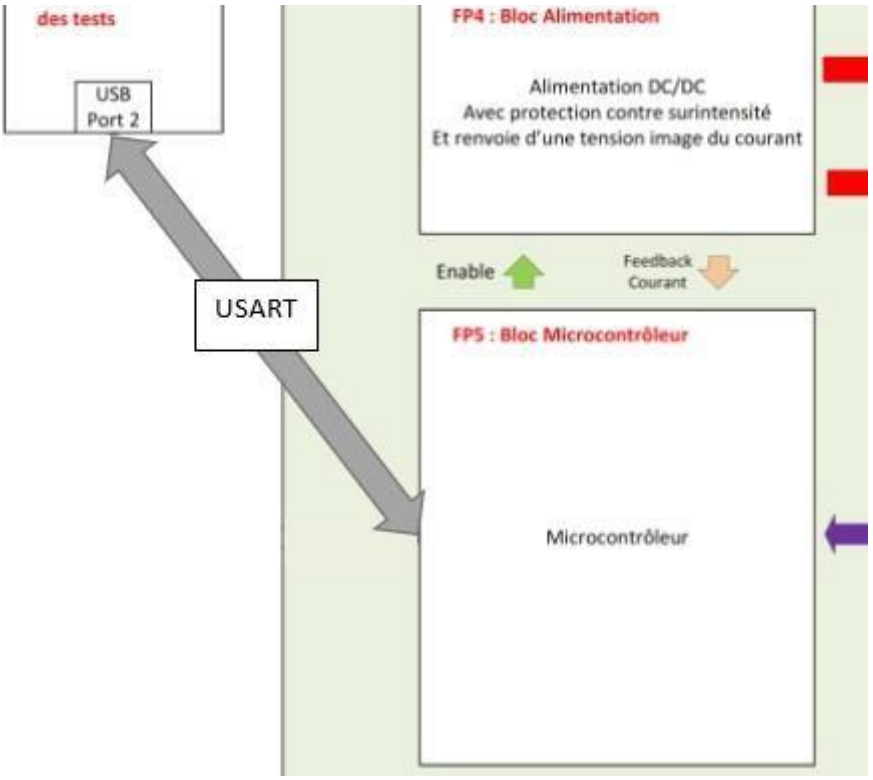


Figure 14 Improved part of the test bench synoptic

In the matter of asynchronous or synchronous communication, we prefer to use asynchronous communication as it is simpler to use to send isolated data to the computer and vice-versa. This choice will be better tested when we start programming so using the USART lets us change the synchronisation of the communication in the last minute if necessary.
In addition, we believe USART is the best option for communication between PC and MCU based on the following arguments:

- No need for an extra adapter.
- Simplifies programming.
- There are no time-critical communications happening over it.
- It is asynchronous, and we can change easily to synchronous if necessary.
- It would be illogical not to use the components we already have (unless there is information we don't know/have access to yet).
- We are more used to work with this peripheral.

7. References

Agarwal, T. (2020, July 1). *Basics of UART Explained - Communication Protocol and Its Applications*. ElProCus - Electronic Projects for Engineering Students. Retrieved 14 February 2022, from https://www.elprocus.com/basics-of-uart-communication-block-diagram-applications/

Beningo, J. (2015, September 21). *USART vs UART: Know the difference*. EDN. Retrieved 14 February 2022, from https://www.edn.com/usart-vs-uart-know-the-difference/

Brunete, A. (2021). *Sistemas Electrónicos Digitales. Tema 4.8: Comunicaciones Serie* [Slides]. MoodleUPM. https://moodle.upm.es/

Introduction to I2C and SPI protocols. (n.d.). Exostivlabs. https://www.exostivlabs.com/files/documents/Introduction-to-I2C-and-SPI-Protocols.pdf?/article/aa-00255/22/introduction-to-spi-and-ic-protocols.html

Legaspi, M. G., & Peña, E. (2020). UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter. *Analog Dialogue*, *54*(4). https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

Wikipedia contributors. (2021a, June 17). *Universal synchronous and asynchronous receiver-transmitter*. Wikipedia. Retrieved 11 February 2022, from https://en.wikipedia.org/wiki/Universal_synchronous_and_asynchronous_receiver-transmitter

Wikipedia contributors. (2021b, June 17). *Universal synchronous and asynchronous receiver-transmitter*. Wikipedia. https://en.wikipedia.org/wiki/Universal_synchronous_and_asynchronous_receiver-transmitter

Wikipedia contributors. (2021c, November 22). *Serial Peripheral Interface*. Wikipedia. Retrieved 11 February 2022, from https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

Wikipedia contributors. (2022, February 8). *I2C*. Wikipedia. https://en.wikipedia.org/wiki/I%C2%B2C

Wilcox, M. (2020, March 17). *Is baud rate same as frequency?* Colors NewYork. Retrieved 14 February 2022, from https://colors-newyork.com/is-baud-rate-same-as-frequency/

Y. (2021, January 15). *SPI - Introduction to Serial Peripheral Interface*. Latest Open Tech From Seeed. Retrieved 11 February 2022, from https://www.seeedstudio.com/blog/2019/11/22/spi-introduction-to-serial-peripheral-interface/

Campus Brugge
Xaverianenstraat 10
8200 Brugge
T 050 30 51 00

Campus Brugge station
Spoorwegstraat 12
8200 Brugge
T 050 40 59 00

Campus Kortrijk
Doorniksesteenweg 145
8500 Kortrijk
T 056 26 41 60

Campus Oostende station
Lijndraaiersstraat 60
8400 Oostende
T 059 56 90 00

Campus Oostende VLOC
Nieuwpoortsesteenweg 945C
8400 Oostende
T 059 30 81 50

Campus Roeselare
Wilgenstraat 32
8800 Roeselare
T 051 23 23 30

Campus Torhout
Sint Jozefstraat 1
8820 Torhout
T 050 23 10 30

www.vives.be

vives
hogeschool