Politecnico di Torino
College of Engineering

Master thesis

Implementation of a Software Defined Radio TT&C Transponder for CubeSats

Master's Degree in Electronic Engineering

Supervisor:
Leonardo Reyneri

Student:
Antonio Miraglia

December 2, 2018

Contents

Chapter 1

Introduction

The aim of this project is the development of a software-model of a TT&C Transponder to be fitted in a future test bench, to validate new hardware and improve the knowledge and confidence in existing transponder systems by simulating real-life situations. This TT&C Transponder has been designed to be used within the new generations of CubeSats contributing to increase the flexibility and performance of the telecommunications part. Furthermore, it will also be possible to reduce costs not only by using cheaper hardware, but also by speeding up the design time. Finally, thanks to the choice of a suitable framework and the software models developed, it will be possible to create both a test and development environment for future updates and upgrades for the developed system.

1.1 Main technical objectives

A software-defined transponder, compared to a hardware-defined one, presents several advantages from the operational point of view, with flexibility being the major one. The usual development and qualification cycle for space-hardware takes even 10 years to the point that qualified hardware can sometimes be considered obsolete, as compared to contemporary systems at the beginning of the qualification cycle. Space-qualification is a lengthy process which is fundamental for mission success, but at the same time causes sometimes a huge delay between technology introduction and final use. Software-defined systems try to bridge this gap by relying on a qualified set of hardware and by implementing the relevant functions (in this case demodulation, on-board processing a modulation) in software. The software qualification cycle is usually shorter and can heavily rely on the already existing hardware platform. Based on the previous considerations, it can be seen why a software-defined transponder would be interesting: the hardware could be designed once (or at least be more readily available and standard) while the software can be customized at a later stage or even re-programmed while the system is already in operation (actually shortening even more the product development cycle) or allowing to optimize the system functionalities over the mission lifetime.
Approaches described in "Pseudo-Noise Ranging for Future Transparent and Regenerative Channels"[1], where a more modern standard is evaluated also on equipment already designed for previous standards, show clearly this issue. Moreover the advantages of system flexibility are not limited to the space segment, but could also be seen on the ground side, where the deployment of a new technology could be lengthy due to the time required to deploy new hardware world-wide.

1.2 Definition of TT&C Transponder

Telemetry, Tracking and Command (TT&C) are fundamental functions for a space vehicle. The transmitter and receiver (on the spacecraft) together with their complementary versions inside the Ground Control Station (GCS) allow data communication between the terrain and the spacecraft. This is a list of the fundamental functions of a TT&C system:

- **Command**: used to send commands to perform any reconfiguration, to send or request data.

- **Telemetry**: used to send data to the ground station. Telemetry contains information on the status of the satellite, useful for its control, status, configuration or any emergency messages.
- **Tracking** and **ranging**: used to measure the round trip time (so the instantaneous distance) and the Doppler effect (so the relative velocity). It involves demodulating the uplink signal from the Ground Station and remodulating it back (with a non-coherent or coherent frequency). Furthermore, a system implementing such function is called transponder.

In CubeSats, as in many other technological sectors, in addition to cost reduction, it is essential to reduce size, weight and power consumption. In order to achieve these goals, there has been a constant shift from analogue to digital technology. Progress have been made in miniaturization (FPGA, MMIC, ASIC, DSP) and in the digitization of modulation, demodulation and frequency generation functions[2]. The advantages of using digital technology are many: it offers greater reliability and performance as well as reduced costs and dimensions. Furthermore, the digitization of the signal processing part also simplifies the interface with the microprocessors on board.
As mentioned previously, in the space environment the components are subjected to different types of disturbances. It therefore becomes important to increase the reliability of the system. Redundant systems are often used, which can be applied to hybrid implementation based on a digital signal processor (DSP) and Field Programmable Gate Array (FPGA), which is considered the ideal solution for the hardware implementation of TT&C Designed transponder.

1.3 FlaReSS

The work done in this project was part of the FlaReSS project carried out for the European Space Agency (ESA) called "Implementation of a Software defined Radio TT&C Transponder". As it can be read in the proposal, this project was carried out through a collaboration between the Delft University of Technology (TUD), the Eindhoven University of Technology (TUe) and finally ISIS - Innovative Solutions in Space, based in Delft, The Netherlands. The author of this report has therefore participated in the realization of this project by doing an internship at ISIS company and uses its right to use part of the documents produced during the project.

In Figure 1.1 it is possible to see the division of work among the various members. Although involved in the preceding phases, the author dealt mainly with the part of "architecture definition", "building block definition" and "ECSS block analysis". That is, in the development and testing of all the blocks that make up the complete architecture of the TT&C Transponder. In fact, the whole project can be divided into blocks, where each block corresponds to a software library that has been designed to perform a specific function within the complete architecture, for example the PLL.

The thesis work is organized as follows:

- In the CubeSats chapter a brief overview is presented to the CubeSats way, paying attention to both their history, their current and future use, and how they are generally designed.
- In the System design and justification chapter are included all the design choices, the compromises, and the motivations that led to the determination of both the final architecture, but also the development and testing environment.
- In the TT&C Transponder design chapter is presented the final designed architecture. By explaining for all the various architecture's blocks their function and their realization within the chosen development framework.
- In the Extra Blocks chapter are presented all the various extra blocks that are not directly useful for the construction of the architecture, but which are necessary for the use of it within the chosen framework.
- In the QA Tests chapter are presented all the various quality assurance tests performed for each block realized in order to guarantee their correct behaviour.

- In the Validation Tests chapter are presented all the tests to certify the correct behaviour of the complete architecture for the main functions of the designed transponder. In addition, both hardware and files certified by ESA have been used in order to guarantee maximum reliability.

Finally, to simplify re-distribution from ESA to other parties, the source code is already present on a public website at the following link: `https://github.com/FlaReSS`.

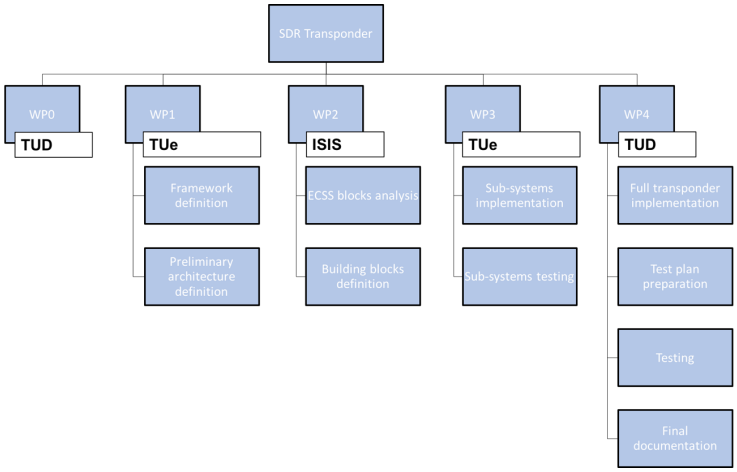

Figure 1.1: Work Breakdown Structure (WBS)

Chapter 2

CubeSats

2.1 Introduction

The story of CubeSats began as a collaboration between the California Polytechnic State University (Cal Poly) and the Stanford University's Space Systems Development Laboratory (SSDL). The aim of the project was to provide affordable access to space for the university and science community, and that is exactly what happened. Nowadays, thanks to CubeSats, plenty country of the world have a space program also thanks to universities. Not just big ones, also small universities and high schools have also been able to start their own CubeSats program.
Some nations have obtained their first space program thanks to CubeSats launched by universities, states or private companies. as shown in Figure 2.1, many countries have CubeSats programs.

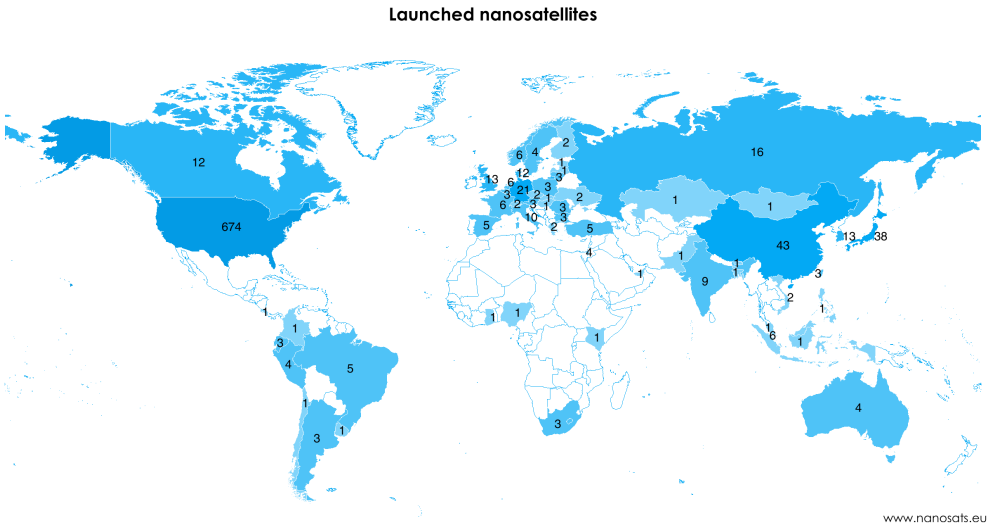

Figure 2.1: NanoSat and CubeSats world map

Furthermore, private companies and government agencies are investing in CubeSats, as shown in Figure 2.2.

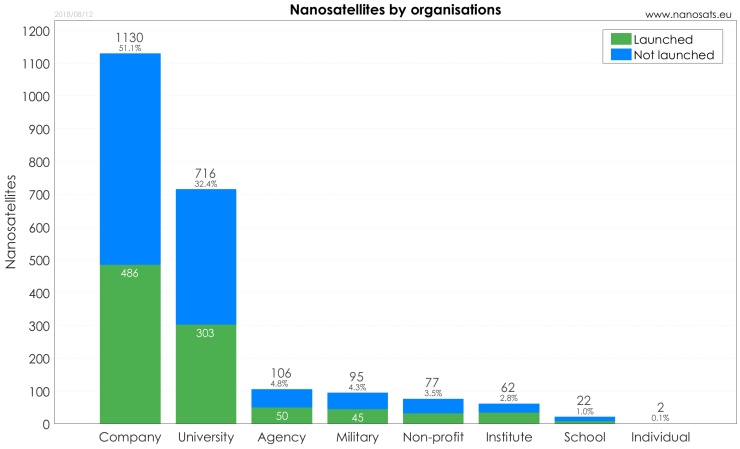

Figure 2.2: NanoSat and CubeSats organization type

as shown in Figure 2.3, several companies are creating their own constellation of CubeSats. There are many purposes, for example Earth-imaging (Planet[3]), Cloud Data & Analytics (Spire[4]), communications networks (Sky and Space Global[5]), climate monitoring (GeoOptics[6]) and so on.

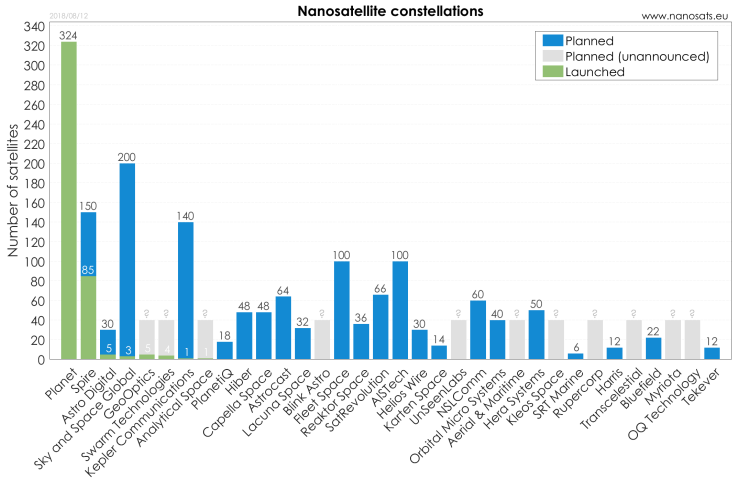

Figure 2.3: NanoSat and CubeSats constellation

The cost reduction is driven by the standardized form-factor of CubeSats, that, in turn, reduced research and development, production and launch costs. This has led to an exponential growth in the popularity of CubeSats from the beginning, totally revolutionizing the space sector[7].

The CubeSats is a miniaturized satellite with a modular structure. Its base unit is $(10 \times 10 \times 10)$ cm with maximum 1.33 kg of weight, and it is equivalent to one unit (1U). Thus, i.e. a 6-Unit CubeSats has dimensions of $(10 \times 20 \times 30)$ cm and weighs about 7 kg. as shown in Figure 2.5, the 3U CubeSats are the most common. in Figure 2.4 an example of the different sizes is shown. In reality, the dimensions defined up to now are only theoretical, the practical dimensions differ slightly. For example a 1U is $(10 \times 10 \times 11.4)$ cm, a 2U is $(10 \times 10 \times 22.8)$ cm, a 6U is $(10 \times 24 \times 34.2)$ cm

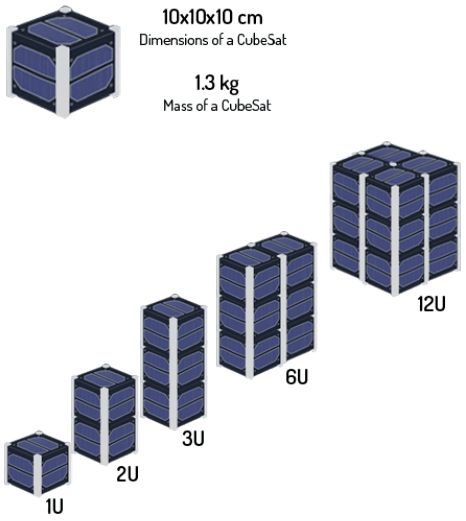

Figure 2.4: NanoSats and CubeSats form factor

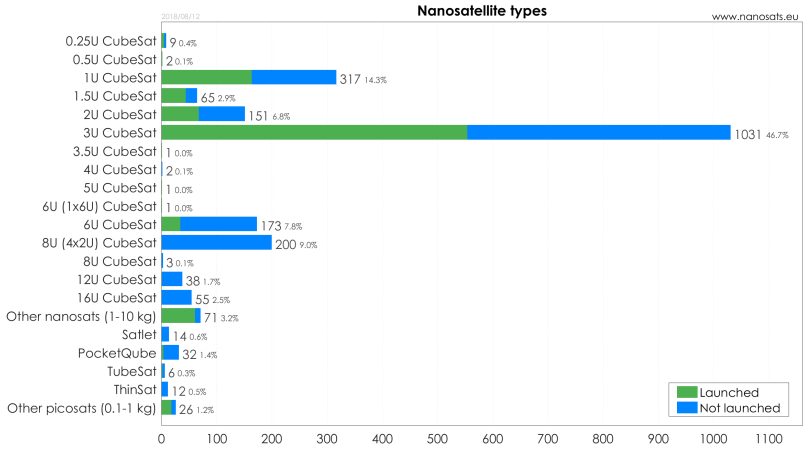

Figure 2.5: NanoSats and CubeSats types

The standardization of satellite size allows CubeSats to be available as "ready for commercial use" products, now produced by different companies and which can easily be adapted to the needs of mission, mainly changing the payload. Generally they are made by commercial off-the-shelf (COTS) components.
In addition, the compact and standard dimensions allow CubeSats to exploit the vacant space of other launches, ie as secondary loads. This simplifies accommodation on the launcher and minimizes flight safety issues, increasing the number of launch opportunities and keeping launch costs low. Finally, they can be set up for flight on a much faster basis than traditional satellite programs, usually within one to two years[8].
To date, over 875 CubeSats and 958 NanoSats have been launched. as shown in Figure 2.6, many have already been planned with a real positive trend[9].

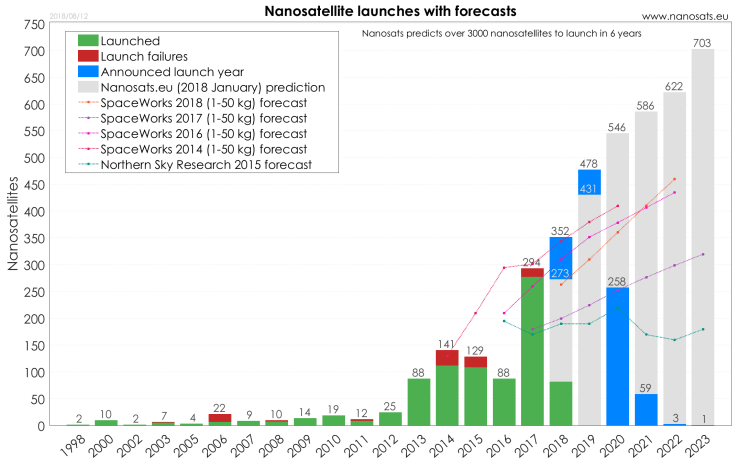

Figure 2.6: NanoSats and CubeSats launches with forecasts

The uses of CubeSats are many and generally involve scientific experiments that can be miniaturized or for purposes such as Earth observation or amateur radio. CubeSats are often used to demonstrate new spacecraft technologies or that have a questionable feasibility, or experiments whose theory is not proven, it is unlikely to justify the cost of a larger satellite. Several missions on the Moon and on Mars plan to use CubeSats[10].

2.2 Design

In the design of CubeSats, for communication protocols or design specifications no specific form factor are required. COTS components are often used, selected to ensure that the devices can tolerate the radiation present by the specific conditions of use. A particular case are the very low earth orbits (LEO), in which the flight time is a few days or weeks, the radiation can even be ignored. This is because the probability of a single event upset (SEU) is very low. For longer space missions, a design that takes into account radiation and high vacuum operation is required. In fact, effects of out-gassing, sublimation, and metal whiskers can occur in space. Therefore, it is necessary to test all the hardware components in irradiated environments, under vacuum, and in the different temperature ranges. In critical missions, radiation-hardened devices must be used.
Finally, the designed electronics generally conform to the PC / 104 form factor, which has a (90×96) mm profile and occupies most of the volume of the CubeSats. Stackthrough connectors are often used to allow simple assembly.[10]
Figure 2.7 shows the most common orbits.

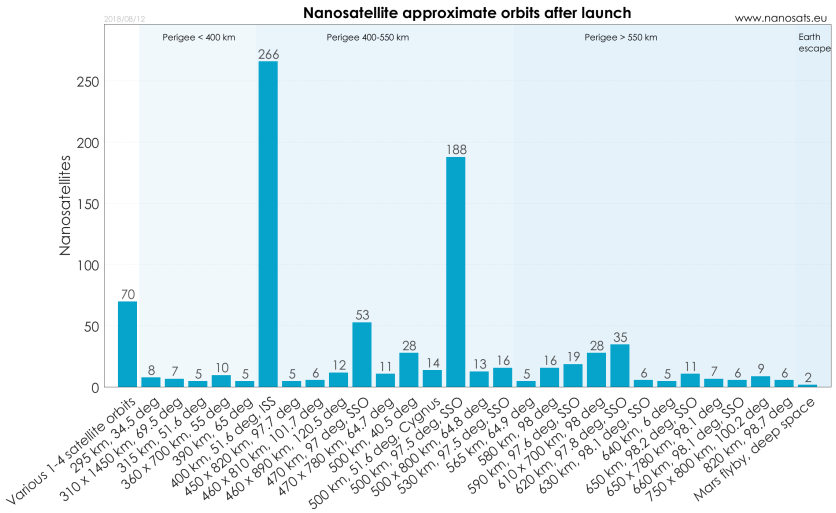

Figure 2.7: NanoSats and CubeSats orbits

2.2.1 Computing

CubeSats often have multiple computers which handle in parallel different tasks. Typically, power management, the attitude control (orientation), payload operation, and primary computer.
Payloads must interface with the main computer, often requiring the use of another computer, so as not to limit it too much. The use of a second unit prevents the main computer from having to handle raw data, considerably reducing the workload of the payload management, such as image processing, analysis and/or data compression. This allows the main computer to take care of its primary tasks, such as the communication of the other subsystems, without too many interruptions.
As mentioned previously, CubeSats are often subject to radiation and the components used are generally sensitive to radiation as COTS. To keep this problem under control, special design solutions are often adopted such as the use of redundancy through the use of multiple primary computers, and the use of Error-Correcting Code (ECC) RAM. This is done especially on more important missions[10].

2.2.2 Power

The CubeSats are mainly powered by rechargeable lithium-ion batteries. The latter, are recharged by solar cells that, by converting sunlight into electricity, allow energy to be stored which will be used during periods of darkness and during periods of maximum load.
The CubeSats have a limited surface area of the external walls, moreover, it is often occupied by other parts such as antennas, access doors, possible propulsion systems and particular payloads. Therefore, the area useful for solar panels is very reduced, this is why other solutions are often used, like deployable solar arrays that allow a wider area to be covered by solar cells. Also, where possible, attitude control is also used to orient the satellite so that the solar panels have an ideal exposure. But they have a very limited operational temperature range, which requires several extra components (like protections and heaters).
Not all types of batteries can be used in space: Lithium-ion batteries are very common, as they have excellent energy-mass ratios, essential parameters for devices with mass limits.
The charge and discharge of the batteries are generally managed by the dedicated power supply system (EPS). In addition, special heaters are often needed to prevent batteries from reaching too low temperatures that could cause failures[10].

2.2.3 Antennas

In CubeSats, the power available is limited for its communication antennas. For example, the ISIS S-Band transmitter has a transmit power between 27 and 33 dBm[11]. as shown in Figure 2.8, it is possible to use almost each radio band. The most frequent are: VHF, UHF, S-band and X-band.

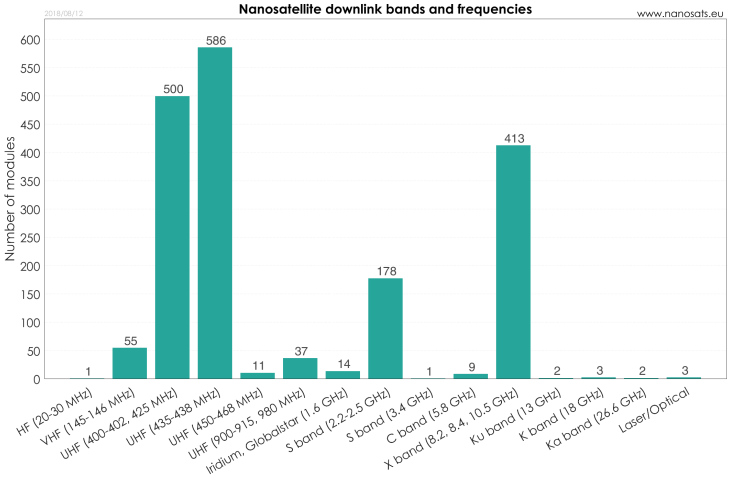

Figure 2.8: NanoSats and CubeSats radio bands

For UHF / VHF transmissions monopolies, dipoles or turnstiles antennas are generally used. They are folded and automatically opened in flight. In addition, an omnidirectional monopole or a dipole antenna constructed with a commercial measuring tape is often used. For more complex requirements, high-gain antennas are on the market, but require significantly more complex pointing systems.
Traditionally, small satellites that operate in low Earth orbit, use VHF, UHF and S-band. While, for more distant missions, larger antennas compatible with the Deep Space Network are needed, S-band, X-band and Ka-band are used because it is possible to have a higher gain with a smaller antenna if the frequency is higher[10].

Chapter 3

System design and justification

The implementation is based on GNU Radio, a framework used to develop digital signal processing applications. GNU Radio provides a runtime environment and an easy user interface to describe dataflow algorithms and it will be used to implement the transponder software. Many of the software building blocks required are already present in libraries compatible with GNU Radio. To properly operate and validate the developed software, two additional applications need to be developed which will act as interface to GNU Radio to simplify the development of external modules interfacing with the transponder.
This project will also serve as a starting point to implement a wide library of components for digital-signal processing compatible with the ECSS suite of standards. This is will allow future development of digital-signal processing system in a quick and efficient way.

3.1 Software Framework

The first thing to define in this project is the implementation technology and/or framework. This may seem trivial, but it could make the whole effort much simpler or more complex depending on the different choices. Software-defined systems have become so widespread on the market also thanks to the existence of frameworks: they are already matched in many cases with RF generation/acquisition hardware or with existing functions to record/read recordings. These frameworks can also include some higher level functions (like frequency mixers, filters, etc.) but are usually limited to data-flow blocks (blocks that perform data processing without any internal state, in some aspects called Linear-Time-Invariant systems so processing the inputs always in the same way, like a frequency mixer or a low-pass filter). The advantages of relying on a standard framework are also that, in most cases, several functions for visualization are already present: a graphical user interface will simplify the configuration of the system and will also allow a simpler understanding of the underlying logic: for example, a system providing pre-define and custom blocks and allowing them to be interconnected would allow a user to "reverse-engineer" the system quickly and to master its functionalities and possibilities. A further advantage of this type of framework, is that documentation can be tied to the single blocks (and eventually hierarchical blocks), simplifying the complete documentation creation and understanding process.

One last important point to address is the licensing of the product that will be provided. It is important to provide the complete software source code together with the eventual development framework selected. Great attention in the final framework selection should be paid to this, as license agreements vary vastly in case different frameworks are selected.

3.1.1 Framework selection

The correct selection of framework to efficiently perform the work is essential for the first step of the implementation phase. Choosing the right software framework is critical for the rest of the project

tasks, so a thorough research for the optimal framework has been conducted.
Nowadays, there is a bloom in frameworks for software-radio development each with its own advantages and disadvantages. To select the optimal framework, a number of selection criteria are set:

- Maturity of the software environment: are excluded solutions that do not rely on multiple (and interfacing) frameworks or that require the developers to fully program all the functions (including save / restore functions, graphical APIs to customize the transponder or display results). This is a criterion that is justified by the limited time available for the implementation. This also means that an existing GUI is also beneficial, since it could allow to change most parameters on the fly, have results displayed and generally make it more user-friendly.
- Support of hardware board:; this is not strictly required as part of the project, but it is important to consider the criterion, to avoid developing the system on one of the frameworks and being forced to switch later on. Hardware boards will be used for the implementation of the transponder and having a rich support will be an advantage. Furthermore, system cost should be kept to a minimum to inspire small industries and universities to use this development for small satellites and CubeSats. Having the same mindset, it is leaned towards an open source license rather than a commercial one. This will simplify other teams to get involved and push them to contribute their developments back to the main developers.
- Simplicity: the selected framework should be easily accessible, friendly to the user and most importantly, have a very lively community support. Availability of already made software/packages (such as GitHub repositories), number of relevant publications, forums/StackOverflow/Google groups and lastly easy to find examples and tutorials were some factors that were taken into account.
- Level of confidence and experience our team has in the frameworks: confidence in using the framework also makes sure that the selected technical solution can be implemented in an effective way and that limited time is also spent in getting the framework to work (installation and configuration time).

In this trade-off, few possible frameworks are been shown. These are listed in Table 3.1:

Table 3.1: Preliminary framework trade-off

Framework	**HW support**	**license**	**Confidence**	**GUI**	**Maturity**
GNU Radio[12]	high	GNU GPL v3	high	yes	high
LuaRadio[13]	high	MIT	low	yes	low
PothosSDR[14]	high	MIT	low	yes	medium
RedHawk SDR[15]	none	GNU LGPL	low	yes	medium
PySDR[16]	medium	GNU GPL v3	medium	no	high
Matlab[17]	medium	Commercial	medium	yes	high
Simulink[18]	medium	Commercial	medium	yes	high
GNU Octave[19]	medium	GNU GPL v3	medium	yes	medium

Redhawk SDR is excluded, since there is no hardware support, although it has a medium-sized community and is the only one of the existing frameworks capable of coding in Java. PySDR is a framework that our team had had some experience using and there is a rich sized community, but the fact that it lacks hardware support (only USRP is supported by PySDR) and most importantly the lack of GUIs makes PySDR unsuitable.
Keeping in mind the fact that cost should be minimized and license management should be simplified (also to inspire universities and small industries to use this software), MATLAB and Simulink are discarded. The use of MATLAB along with Simulink is already costly and, on top of that, a user must also purchase certain toolboxes to fully explore the capabilities of the software for space applications. This makes this option very costly. Another factor of excluding MATLAB and Simulink is the very

limited hardware support, including only USRP, RTL-SDR Radio, ADALM-PLUTO Radio and Zynq SDR. Lastly Matlab is widely used for off-line analysis, while other frameworks thrive also in real-time analysis.
GNU Octave is the least competitive candidate due to its limited hardware support (only LimeSDR and possibly USRP, following a GitHub repository project) and the fact that although the community seems quite big there are limited publications about applications.
LuaRadio has a very small footprint (few tens of MB), has an input/output signature system that is very flexible and also a nice feature seems to be the automatic propagation of sample rate between all blocks. It supports also a vast selection of hardware, including USRP, RTL-SDR, SoapySDR, Airspy, HackRF, SDRplay RSP. The community seems to be growing, although it is still very limited. However, there are certain drawbacks. Firstly it only operates using the Lua programming language, which is not very popular, meaning also that our team is not very confident with it. Secondly, it appears to have worse performance than GNU Radio, as seen on this test benchmarks[13]. Due to the previous drawbacks, it was decided to exclude LuaRadio.
The last two frameworks (PothosSDR and GNU Radio) have been evaluated and tested in more details to better understand which of the two would be the most feasible one for this project. Both frameworks have a very solid hardware support. PothosSDR supports USRP, UmTRX, OsmoSDR, RTL-SDR, Blade RF, Hack RF, LimeSDR, Miri SDR while GNU Radio supports, USRP, UmTRX, SoapySDR, LimeSDR Funcube Dongle, Novena RF, Blade RF, Hack RF, Microtelecom Perseus. Both frameworks have a very similar GUI, although GNU Radio has a more complete initial block library than PothosSDR. Potentially PothosSDR can be compiled on all operating systems, but currently it is only supported under Linux. GNU Radio runs instead on all operating systems, but performances are more consistent under Linux (this last statement is rather qualitative and on a statistical base, which comes from the team experience in trying it under different operating systems).
The difference in the size of the community is tremendous. While PothosSDR has a dedicated Google Group for help, it seems to rely only on that. GNU Radio instead has a special Reddit page, strong support in StackOverflow and ten times more repositories in Github. In GNU Radio commercial support and commercial training can be provided. Moreover, GNU Radio is a framework that our team is more confident with. PothosSDR seems to be very promising, but still judged immature.
Finally, GNU Radio is chosen, principally thanks to the richer community, pre-existing knowledge/projects and much higher team confidence on the framework.
It should anyway be noted (about GNU Radio) that:

- Changes between versions need to be tracked;
- Usually the GNU Radio scheduler is well checked (as it is one of the core parts of the framework), but other blocks might not be;
- Actual functionality of the blocks needs to be checked to ensure it complies with documentation and/or name;
- Implementation mistakes might be present in the code;
- Deliver version information for all provided files and hash code;
- Evaluate packaging options (ex. Docker) for quick deployment.

3.1.2 Additional remarks

Furthermore, it is important to notice that usual SDR hardware (like USRPs) communicate with the computer using standard interfaces (like USB, Ethernet or PCIe) that are equipped with small buffers to handle the communication. It is then very important for the computer to react in time to empty a buffer before it overloads. It can be seen in practice that sometimes SDRs are not limited by the total data throughput the computer can handle, but by the latency in emptying these buffer. Thread switching is thus required to be very quick and not be interrupted for a long time (in general few tens on milliseconds). Considering the experience of the team with operating systems and general-purpose computers, it was noted that Linux (in general, with some variations between different distribution)

provides more consistent and predictable performances. Indeed, its kernel allows a higher level of configurability (without looking into real time extensions) and consistency (in terms of the maximum loss of priority of critical threads) much better than the Windows Kernel. For the time being, it is not expected to require specific kernel real-time extensions but a regular computer and operating system are expected.
It is recommended to run an SDR application on a computer running Linux. This has shown, based on the experience of the team, a more consistent behaviour. This does not exclude high performances implementations under other operating systems (such as Microsoft Sora[20]).

3.1.3 Implementation

The selected framework (GNU Radio) is organised in modules and blocks[21]. Thus, are been developed dedicated GNU Radio module (to be considered as a library of blocks to drag and drop) which can be used to implement ECSS-compatible systems. Following the usual GNU Radio naming, a module called "gr-ecss" is implemented and populated with blocks that implement the required functions. Transponder parameters will be divided in two categories: high level parameters which are generally configurable in a real transponder (such as data rates, modulation, etc.) and DSP parameters which are fixed in real transponder, but that need to be adjustable during algorithm optimization.
The goal of this project is the development of a TT&C transponder. In real-life applications, this transponder is coupled with an On-Board Computer (OBC) to control the different modes and functionalities. In our case, this can be achieved with an external application connected to the TT&C transponder via a socket (already available in GNU Radio) to allow an external user to implement a fully capable software model of an OBC. Real-time TC decoding and TM generation will not be implemented, but the application skeleton will be realised and will allow extracting data (via a network connection or files).
As the application is be based on GNU Radio, are been used the GNU Radio conventions for coding, commenting and formats.

3.1.4 Software license

The selected framework GNU Radio is licensed under the terms of the GNU GPLv3 license[22]. The terms are many and in general quite complex, but for simplicity are reported here few extracts to clarify the selection.

The license grants unlimited permissions to run the provided program:
"All rights granted under this license are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This license explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this license only if the output, given its content, constitutes a covered work. This license acknowledges your rights of fair use or other equivalent, as provided by copyright law."

The license also mandates the distribution of the source code of the application together with the binary application:
"You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this license and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this license along with the Program. You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee."

If modifications are made to the application source code and the application is re-distributed, the (modified) source code shall be distributed as well:
"You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

1. *The work must carry prominent notices stating that you modified it, and giving a relevant date.*

2. *The work must carry prominent notices stating that it is released under this license and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".*

3. *You must license the entire work, as a whole, under this license to anyone who comes into possession of a copy. This license will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This license gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.*

4. *If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so."*

These conditions will apply to the development team that, as stated by the license, is distributing the software to ESA, and it so bound to distribute the source code as well. No requirement for fully public distribution is mentioned: the modified source code shall be distributed in the same way as the binary code. This practically means, in case of re-distribution of the code by ESA, that the modified source code should be attached as well (or at least reachable, for example via email or via a website) to the party receiving the binary code.
To simplify re-distribution from ESA to other parties, the source code on a public website (`https://github.com/FlaReSS`) and make mentions to it in the source code will be provided. This will satisfy the source code redistribution. This might not satisfy the license conditions in case of modifications to our code by ESA. Those will have to be provided by ESA to the interested parties.

3.1.5 Hardware

One of the important objectives of this project is the use of general purpose computers to reduce the cost and complexity of the overall system and to allow also small institutions to perform this type of research. The last years saw the explosion of digital signal processing, especially for Software-defined radios thanks to the wide availability of cheap hardware (and sometimes as cheap as 20 $) that could be turned into complex receivers by means of software running on a regular notebook. An important aspect of the required work is also to try and be compatible with this trend and allow the future the inclusion of hardware interfaces to use real-life signals (generated with an ad-hoc modem or even coming directly from an antenna).

3.1.6 Communication interfaces

In real life, the transponder is expected to operate in conjunction with an ECSS-compatible ground station modem in real-time: this would allow, for example, to try different modems to assess the performances using the same transponder software implementation or trying different transponder settings and/or implementations. By looking at the project in a broader view, the transponder will be integrated in a more complex test-bed that could, in principle, allow to remove the software transponder and replace it with a real (hardware-based) one to compare the performances. It could even allow to compare the software model of the transponder with the EM/FM implementation. From all these possible scenarios, it follows that the software model should be able to run in real-time on a regular computer (like a Intel Core I3). As "To Tender" also states, it will not be required to demonstrate the interfacing with hardware equipment to sample/generate real-life signals but it will be required to input/output recordings to test the real implementation. An important requirement for the system to be developed arises from this last statement: correlation between input and output is required as the time offset between the 2 streams should be consistent and repeatable otherwise all the results will be affected.
The handling of the connection handshake requires thus that the transponder also contain a link-management block (to handle the required communication states), beside the digital signal processing

parts. This has to be taken into account from the beginning since, as already stated, most existing frameworks are designed to simulate only the signal processing part and do not handle state diagrams and transitions (like the ones needed when handling data transfer or in this case carrier synchronization). Again, proper selection of the framework should solve this issue, as some of the existing frameworks allow the implementation of fully custom blocks where state transitions can be handled.

3.2 Stream synchronization

In GNU Radio, all blocks require interconnections that can be of different type, depending on the data that needs to be transported:

- The interfaces labelled RX and TX are used to transport samples (I + Q) and could be seen as the equivalent of an analog connection (RF or IF);
- The interfaces labelled TC and TM are used to transport demodulated bits and bits to be modulated and can be seen as an equivalent of a digital bit-oriented connection;
- The interface labelled Control and TLM are used to transport information based on frames (like commands or telemetry frames).

Due to the different nature of the previously mentioned interfaces, implementation might be complex and lengthy. GNU Radio provides already 3 types of interfaces that can be used directly:

- XML RPC;
- Remote Control;
- ZeroMQ.

The first one (XML RPC) is based on a remote procedure call framework based on XML. This method is very convenient as it can be used to set / get variables directly. This interface allows to group control operations all on one socket, in a similar way as control of the transponder is done in a real satellite via the main control bus. The Remote Control interface is marked as well as deprecated in GNU Radio 3.7 and will also not be considered further. ZeroMQ is part of the latest changes introduced in GNU Radio and is based on an open-source framework (called ZeroMQ) which is used to transport generic types of data. This allows, for example, to rely on the already existing ZeroMQ blocks on the GNU Radio side and to use the ZeroMQ framework in a fully custom application: this approach conjugates the ease of use of GNU Radio and the flexibility of using a stand-alone application. This will simplify the development as the two types of software (GNU Radio and external applications) will run in separate applications. This will allow, for example, to decouple certain parts of the software from GNU Radio, like the OBC simulator and the orbital simulator.
A connector block on the GNU Radio side should not be required: the ZeroMQ blocks only connect to each other when the simulation is started (either manually by pressing on the start button in the GNU Radio GUI or when the GNU Radio flowgraph is started from command line) and this can be used to detect the start of the simulation.

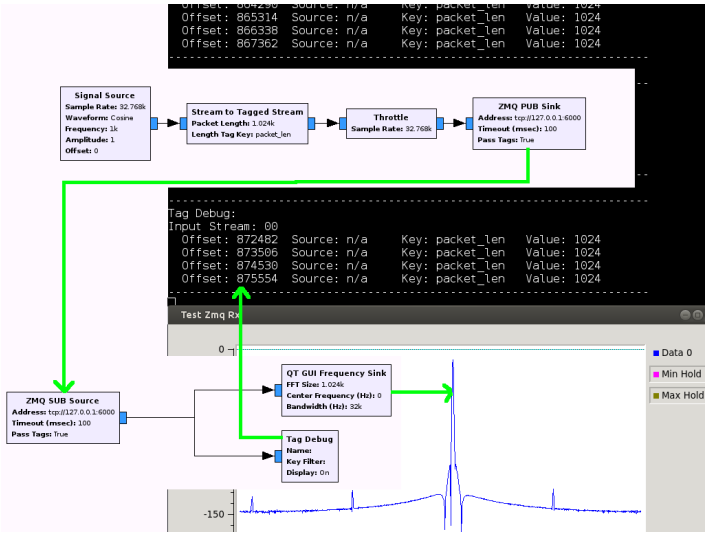

Figure 3.1: ZeroMQ blocks[23]

Once the simulation is done, the ZeroMQ block can be used to send an "finished" message to the "system" message port of the next GNU Radio block. This will cause all subsequent blocks to finish processing and terminate. The previous strategy is an attempt to set a termination time to the simulation (which otherwise might run forever) after all the tests have been performed. This is because GNU Radio does not implement a way to terminate automatically the simulation.
This strategy could also be employed in case of batch execution of tests on a transponder. A sequence of tests could be run by simply executing the compiled flowgraphs from command line and each test will terminate after the simulation is completed, terminating the GNU Radio blocks to allow the next test to be executed.

3.2.1 Interconnections implementation

As it can be seen from Figure 3.11, several interconnections are present between the different blocks. Standardization of such an interconnection will help reuse the code and simplify the design. The selected method to interconnect the different blocks is ZeroMQ, that allows to interconnect easily GNU Radio applications with external application. ZeroMQ supports the transmission of IQ samples and digital data over network socket connections and it abstracts the transport logic to provide a simple and portable way of interconnecting applications.
The telemetry and control interface will be implemented using a different mechanism, based on XML RPC. This solution offers several advantages as it is targeted towards commands (to send and get values) and it can be easily configured based on XML queries sent from clients that could be used to read and set variables. This interface operates also on sockets so it can control the transponder from a different application / different machine.
Taking Figure 3.11 as a reference:

- The TX and RX interface will use a ZeroMQ subscriber block and publisher block, as shown in Figure 3.2. The data type used in these interfaces in complex (32-bit float for I and 32-bit float for Q). The Test setup and Transponder parts do not need to be in the same flowgraph and on the same machine, as long as the connection settings (shown in the left side of the figure) are consistent;

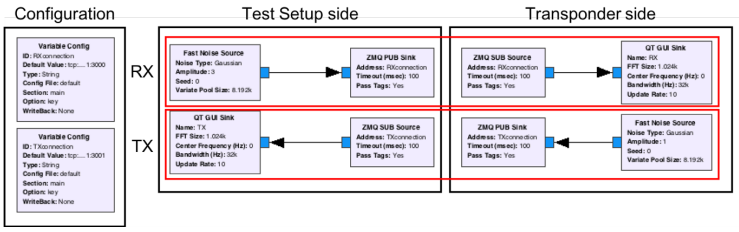

Figure 3.2: RX and TX interfaces

- The TC and TM interface will use a ZeroMQ subscriber block and publisher block, as shown in Figure 3.3. The data type used in these interfaces is byte (8-bits encoding one single bit demodulated/to be demodulated). The Test setup and Transponder parts do not need to be in the same flowgraph and on the same machine, as long as the connection settings (shown in the left side of the figure) are consistent;

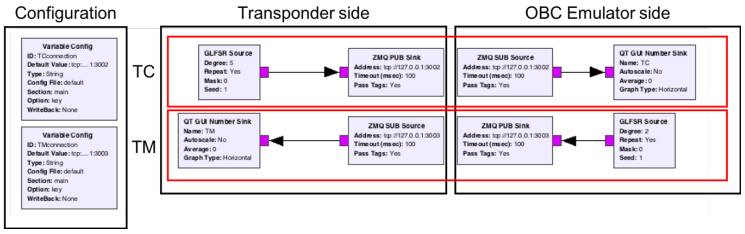

Figure 3.3: TC and TM interfaces

- The Control / TLM interface will use instead the XML RPC blocks to get / set the value of variables. Figure 3.4 shows how the interface will be implemented. Operations are performed over a TCP socket and can be controlled using python or by sending XML code[24].

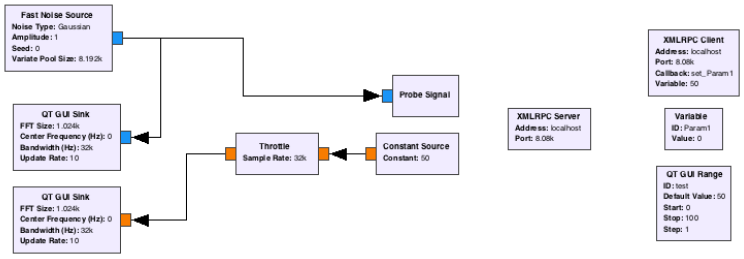

Figure 3.4: Control / TLM interface

It is important to note that with the introduction of the XML RPC control port, it will be possible to set any runtime parameter from OBC emulator. Due to this reason there is no distinction anymore between parameters settable via the GUI and parameters settable by the OBC emulator.

3.2.2 Interconnections flow control

Since all interconnections will be implemented using network sockets, there will not be any concept of data/symbol rate on these connections. Network connections operate at the maximum possible speed (supported by the computer and the network devices). This might be a problem when interconnecting two different applications which might run not perfectly synchronous (due to a different CPU / core load or different computer performances). Since this project does not require any hardware interface,

precise rate control is not needed but samples/bits will be processed as soon as they become available (and assuming the correct rate). It is important to note that, in case these socket connections are not present when the script starts (which is when the socket connections are automatically established), the simulation will produce unexpected results. This is especially true for the connections to the test setup and for the OBC bit-oriented connections. This behaviour is in line with the behaviour of real TT&C transponders that have an unspecified behaviour when the OBC connection is not present. Overflow of receiver / sender buffer is another potential issue when the two systems operate at different speeds. ZeroMQ provides the High-Water Mark mechanism to provide back-pressure and avoid buffer saturation. The last considerations do not apply to the command interface, as this interface is used to probe / modify internal parameters only when needed. If the command connection is not available, no probing / modification to the internal variables can happen.

3.3 TT&C transponder architecture

Most of the readily available RF generation/acquisition systems are based on narrow band designs (even if the definition of narrow varies widely from system to system) with a direct conversion from RF to baseband (or at least this is what is visible to the end-user, since many systems perform the conversion in more steps and handle the complete conversion automatically): this simplifies the usage of such radios but sometimes also creates issues in particular cases (as when non-idealities of the radio start to play a role, like with LO leakage or DC offset). The zero-IF architecture is mostly used because it allows great flexibility in the design and performances improved dramatically in the last years thanks to the miniaturization of complex (IQ) mixers. In principle, the RF signal (at any frequency that the input mixer supports) can be translated to DC in one step and this is performed by mixing the input signal with a complex oscillator (with in-phase and quadrature components). These mixers present no ideal side effect, but in practice are limited by the real performances for several parameters like carrier suppression or sideband suppression. Because of the underlying implementation in the RF generation/acquisition equipment, most of the software frameworks available use IQ data as input and output (and many also allow direct storage/acquisition to/from file).

3.4 State of art

Figure 3.5 shows a simplified model of a commercial SDR, which uses a single local oscillator to control both the receiver and the transmitter branch. A mixer (usually complex) is used to down-convert the RF input signal before acquisition. Usually, due to the limitations of the analog front-end, the first mixer does not convert directly to baseband but to a low IF. The final conversion stage to quasi-baseband is performed digitally to better reject eventual mixer non-idealities (like oscillator feed-through or limited sideband suppression). The ADC and DAC are assumed to be referenced to the same local oscillator but it is not shown in figure for clarity.

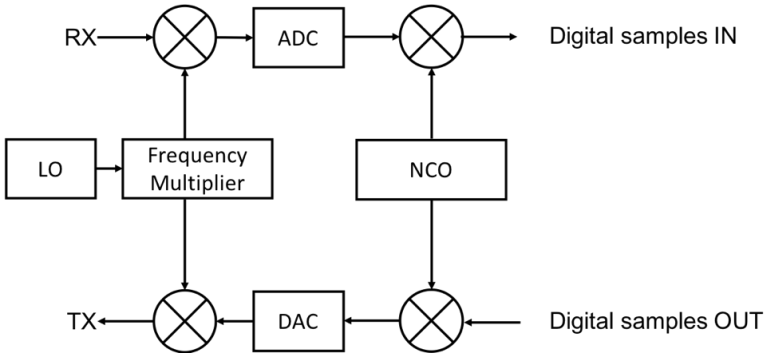

Figure 3.5: SDR simplified model

The software model has to be designed keeping in mind a possible implementation in hardware using commercial SDR hardware (National Instruments USRP devices, for example). These radios are used for general purposes test and development, so they do not contain any specific hardware that might be required for a satellite transponder. This limits the possible architectures that can be selected, as the one shown in Figure 3.7. This architecture is used for the IRIS transponder from JPL[25]: this system used a programmable local oscillator to lock on the incoming RF signal from the ground station. This architecture might be supported by this project by accepting the need for a second SDR. This can be used to generate the local oscillator signal and controlled via a digital control loop (including the first SDR) as seen in Figure 3.6. The overall bandwidth of the system would need to be carefully evaluated as it depends on the delay between the acquisition of a sample via one SDR to the generation of a correction with the second one.

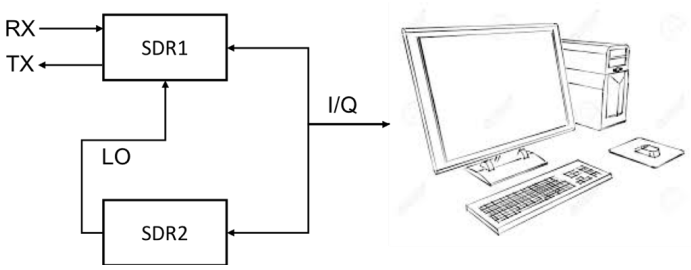

Figure 3.6: Dual SDR to implement a fully tunable transponder

This architecture presents an interesting advantage: the stream coming from the receiver (when lock on the incoming RF signal is achieved) is exactly centred around baseband (or the intermediate frequency) due to the tuning in the local oscillator. This guarantees also that the signal to be transmitted will also be centred around baseband (of intermediate frequency). This is a great simplification of the design and hardware and it could also allow a very simple extension of the transponder from a single downlink frequency to a dual downlink frequency by simply adding a second RF stage but requiring no extra digital processing.

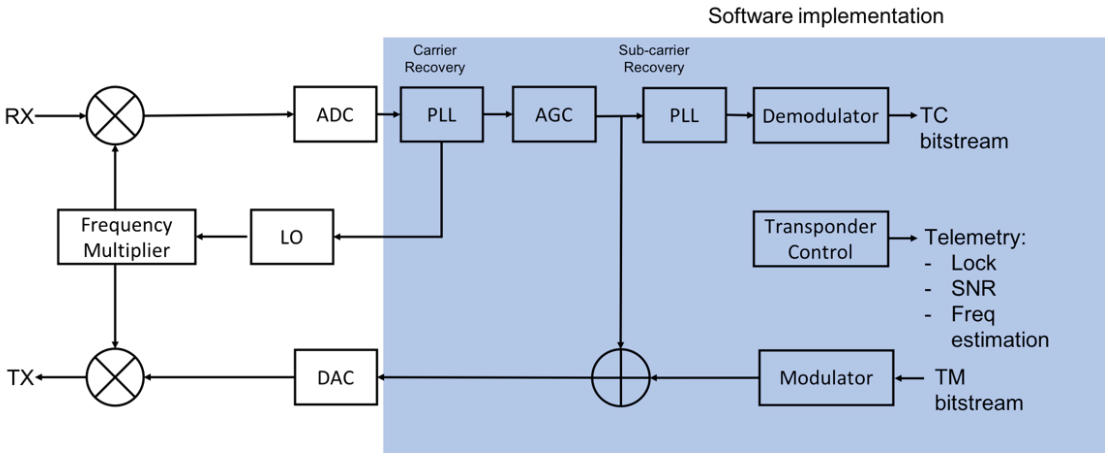

Figure 3.7: Simplified JPL IRIS architecture

3.4.1 Potential hardware issues

Figure 3.8 shows a simplified model of a commercial SDR, which uses a single local oscillator to control both the receiver and the transmitter branch. A mixer (usually complex) is used to down-convert the RF input signal before acquisition. Usually, due to the limitations of the analog front-end, the first mixer does not convert directly to baseband but to a low IF. The final conversion stage

to quasi-baseband is performed digitally to better reject eventual mixer non-idealities (like oscillator feed-through or limited sideband suppression). The ADC and DAC are assumed to be referenced to the same local oscillator but it is not shown in figure for clarity.

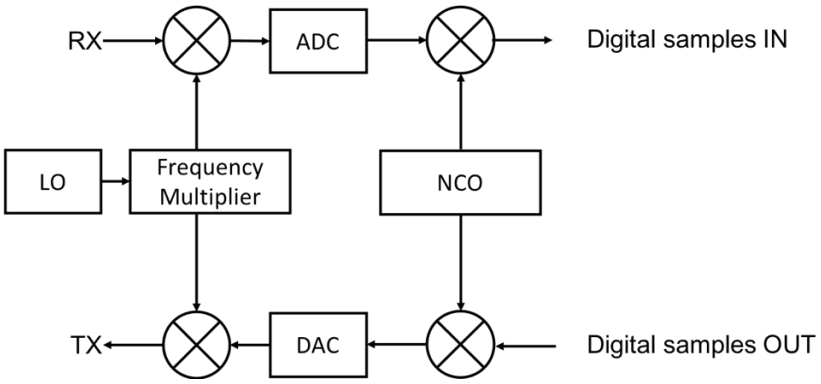

Figure 3.8: SDR simplified front-end

Both oscillators used in the system (the local oscillator and the digital NCO) could have small errors (in terms of absolute accuracy and resolution) and the analog mixer is usually a fractional mixer and both components could impact the coherent frequency ratio between transmitter and receiver. The coherence ratio is very important and any error directly translates into a position error. Typical errors are in the order of the milli-Hertz or lower depending on RF frequency. To overcome these issues, one further digital frequency conversion stage needs to be implemented and special attention has to be paid to the frequency resolution achievable to guarantee eventual errors introduced by the previous two mixers could be corrected. This stage might not be needed depending on the particular SDR that will be selected for the eventual future tests, but for now it is assumed to be necessary also for the sake of general hardware support. This unit might also be included in the carrier recovery loop to reduce the number of mixers in the system. Propagation delays (inside the GNU Radio block diagram) will be closely monitored as it is critical to the final implementation. Feedback loops are not allowed within GNU Radio but can anyway be implemented within single blocks.

3.4.2 TT&C transponder features

Taking into account the previous considerations, the transponder model has been updated adding further features:

- A frequency correction stage used to compensate eventual errors in the analog part of the radio to minimize the error in the turn-around ratio of the transponder. In case the transponder is not locked on the uplink carrier, this stage performs no correction;
- Two filters have been added after the automatic gain control stage to limit the echo of the uplink command into the transponder downlink and to limit the influence on the ranging tones on the TC stream. Careful design of the filters is important to guarantee the performances. According to CCSDS's document "Pseudo-Noise (PN) Ranging System"[26], the filter selection plays a major role in establishing the performances. As discussed during the negotiation meeting, a maximum symbol rate of $1 - 3\ Mchip/s$ is assumed for the ranging code;
- A switch to enable (in case the receiver lock on the uplink carrier) the coherent downlink.

3.5 ECSS

European cooperation for space standardization (ECSS) is an initiative that deals with standardizing all aspects of European space missions. These standards are often created by contractors working

with the European Space Agency (ESA). There is also a second Standard Consultative Committee for Space Data Systems (CCSDS) that deals with standardizing the space missions of all states.
Being a work for ESA, the designed transponder must meet the requirements indicated in the various ECSS standards. In particular, for the purposes of this project, the following documents were used:

- Space data links - Telemetry synchronization and channel coding[27];
- Space data links - Ranging and Doppler tracking[28];
- Space data links - Telemetry transfer frame protocol[29];
- Space data links - Telecommand protocols synchronization and channel coding[30];
- Space data links - Radio frequency and modulation[31].

According to them and ESA, the designed TT&C Transponder must satisfy the requirements to be classified as category B, that is, "spacecraft having an altitude above the Earth of less than $2 \cdot 10^6$ km"[31]. It has been used for both the uplink and downlink modulation only the Binary Phase-Shift Keying modulation (BPSK) as requested by ESA. According to ECSS, only BPSK, QPSK, GMSK and 8PSK can be used[31]. It is widely used in deep space probes, since it is the best modulation in terms of BER performances, spectral efficiency is so critical and it is relatively simple to implement. Its theoretical bandwidth efficiency is limited to 1 $(bit/s)/Hz$, while the real one is limited to 0.7 $(bit/s)/Hz$. Finally, Bit shaping like raised can be used, since it is the most common because it reduces the bandwidth required.

In addition, the operating bands considered for TT&C Transponder will be S-Band, X-Band or Ka-Band. In reality, for the purposes of the project, the choice of a band instead of the other does not particularly influence, since, as will be explained later, the designed transponder will work in Zero-IF. Therefore, the whole front-end part of down-shifting and up-shifting will not be considered. What will be mainly conditioned will be the Turn-Around Ratio and the sub-carrier frequencies and their modulations.

So, for the sake of simplicity, we chose to consider mainly the S-band. According to ECSS document "Radio frequency and modulation", the uplink can have a sub-carrier of 8/16 kHz, which will be a sinewave modulated in NRZ-L, NRZ-M with a symbol rate of maximum 4 $ksps$. Values of higher symbol rates can also be achieved by modulating the carrier directly with SP-L modulation.
In addition, during the ranging phase, the signal will consist of a tone between 100 kHz and 1.5 MHz[28]. While for the tracking phase there should be only the carrier.
Therefore, the uplink signal will be composed of the residual carrier type with the sub-carrier modulated in phase at a distance of at least 8 kHz. This optimizes the power used, but makes the signal easier to modulate. In fact, having a central residual carrier, it will be possible to use a simple PLL to lock the signal accurately. Indeed, by setting a PLL with a rather narrow band, it will only see a carrier making the lock simple. Finally, through a Costas loop it will be possible to intercept the various sub-carriers.

Finally, in downlink the same type of signal shall be generated. For telemetry it is possible to use a sub-carrier between 2 and 300 kHz, with an NRZ-L modulation with symbol rates up to 60 $ksps$. While it is possible to have a symbol rates up to 1 $Msps$ modulating directly the carrier in SP-L[31].

3.5.1 Transponder model

The transponder model to be developed will have to be in line with standard ECSS-compatible transponders to be included in the already described test-bed. This compatibility should be ensured at the physical level modulation mainly, as the RF and operational band requirements would have to apply mainly to the hardware (acquisition and generation part): some exceptions anyway exist since the RF front-end will mainly take care only of down/up converting the signal from/to the desired

RF band to the desired intermediate frequency (being this quasi-baseband or a higher intermediate frequency). Doppler (and especially the range), which may vary if the system is simulating an S-Band link at 2.1/2.2 GHz, an X-Band link at 7.4/8.4 GHz or a Ka-Band link at 25 GHz. The transponder will have to account for the real frequency of operations of the real link (both uplink and downlink) to be able to correctly generate a coherent carrier. It should be also noted that the Doppler range (considering for example a maximum relative velocity of approximately 8 km/s for Earth Observation satellites or maybe up to 10 km/s to allow the use for interplanetary missions) has an influence on the required bandwidth of the total system. To ensure that the transponder model is capable of operating up to 27 GHz (Ka-Band), the system needs to be able to acquire the signal with the maximum and minimum Doppler, possibly forcing the system into using a higher sample rate that required to process only the signal itself. This is required because the hardware part of the full transponder (considering a generic RF interface equipment like a National Instruments USRP[32] device, considered not part of this project) cannot perform coherent frequency tracking[25][33][34], but this has to be performed in software. Several aspects related to the ECSS compatibility should also be considered as proper modulation schemes should be selected and high level functions (like telemetry generation) or configurability (frequency band, modulation, ranging tone/code rates and eventually intermediate frequency).

3.5.2 Turn-around ratio accuracy

According to the ECSS's document for "Space engineering Ranging and Doppler tracking"[28] clause 4.5.2.b, the maximum Doppler bias error allowed is 0.1 mm/s (averaged over 1 s): this error applies ultimately, given the different Doppler rates as a function of the different orbits, to the maximum resolution (and accuracy) in measuring the incoming uplink carrier and in synthesizing the downlink carrier.
Taking into account an Earth Observation satellite, the satellite speed can be as high as 8.5 km/s while the maximum speed relative to ground would amount to approximately 7.7 km/s, as shown in Figure 3.9. Thus, considering that the transponder would be used for interplanetary missions, the relative speed could increase up to approximately 10 km/s.

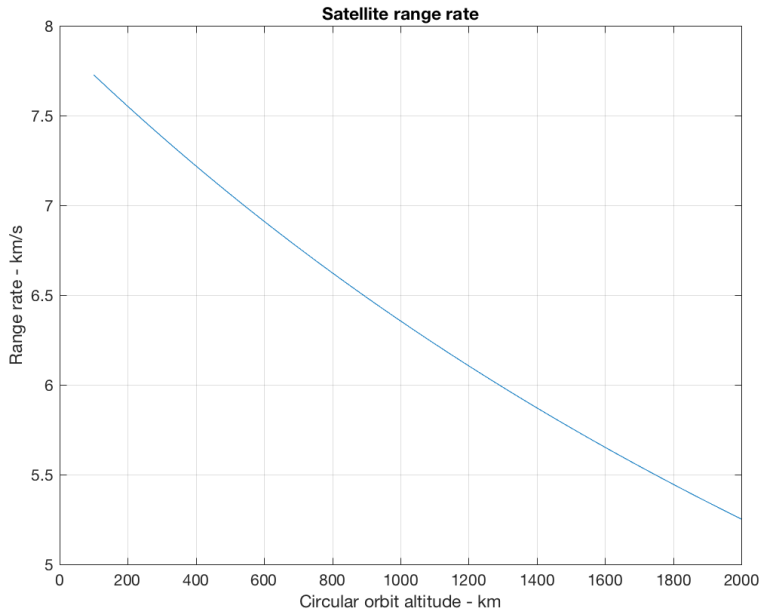

Figure 3.9: Range rate of an Earth Observation satellite as a function of the orbit altitude

From the relative speed, taking into account the minimum and maximum operational frequencies

(in downlink and uplink[31], clause 5.1.2), the maximum and minimum Doppler frequencies can be calculated and from there the influence of the 0.1 mm/s maximum error can be related to a frequency error.
The available frequencies are shown in the table shown in Figure 3.10.

	Earth-space (MHz)	Space-Earth (MHz)	Turnaround ratio (f_{up}/f_{down})
Cat. A	2 025,833 333 – 2 108,708 333	2 200 – 2 290	221/240
	2 025 - 2110	25 500 – 27 000	221/2772 221/2850
	7 192,102 273 – 7 234,659 091 7 190 – 7 235	8 450 – 8 500 25 500 – 27 000	749/880 749/2652 749/2662 749/2678 749/2688 749/2704 749/2720 749/2736 749/2754 749/2772 749/2784 749/2800

Figure 3.10: Frequency and turnaround ration according to ECSS's document for "Space engineering Radio frequency and modulation"

Taking into account the maximum and minimum frequencies in the previous table, and the Doppler shift Equation 3.1 (with v being range rate, c being the speed of light and f the signal frequency), the effect of a 0.1 mm/s error can be calculated. It should be noted, though, that the equation is a first order approximation as secondary effects can appear but for the sake of this analysis the will have limited influence.

$$DopplerShift = \frac{v \cdot f}{c} \tag{3.1}$$

Table 3.2: Frequency error as a function of carrier frequency

	Frequency $[MHz]$	**Error** $[Hz]$
Minimum uplink frequency	2025	0.0007
Maximum uplink frequency	7235	0.0024
Minimum downlink frequency	2200	0.0007
Maximum downlink frequency	27000	0.0090

3.5.3 Digital implementation

As shown in Table 3.2, in case the transponder would be used in S-Band (2.2 GHz), the frequency resolution required to guarantee a Doppler measurement error lower than 0.1 mm/s needs to be better than 0.7 mHz. This requirement is relaxed at higher frequencies. In order to take a reasonable margin over the minimum required accuracy (and resolution), we select to have a resolution 10 times better than the minimum accuracy required.
When digital frequency synthesis is employed (both a CORDIC sinewave generator or a lookup table could be used), the resolution is directly linked to size (in bits) of the digital phase accumulator and to the sample rate, as in the Equation 3.2:

$$Res = \frac{S_r}{2^{bits}} \tag{3.2}$$

with S_r being the sample rate and bits being the number of bits of the phase accumulator. Considering a maximum sample rate of 10 $Msps$ and a 32-bit phase accumulator would lead to a 4.6 mHz resolution, not sufficient for operations in S-band. Given a frequency resolution of 0.07 mHz, we require that the numerical oscillator can achieve such a resolution without averaging bigger phase steps (that would thus introduce an artificial noise in the generated signal). This requires at least a 38-bits phase accumulator (which would lead to a frequency resolution of 0.036 mHz). Given that a 38-bits number is not a standard data type, the closest data type required would be a double precision floating point (52-bits mantissa).
Numerical accuracy in the generation of the trigonometric functions should also be considered[35], the hardware implementation in Intel CPUs has a limited accuracy when the argument of the trigonometric function is close to π (and as a consequence, the error would be accumulating every time the argument of the trigonometric function is re-normalised between $-\pi$ and $+\pi$. The software will be implemented in C++ and the compiler automatically should select a software implementation (which solves the hardware errors), but this needs to be verified during the implementation phase.
Due to the risk of cancellation error in the continuous re-normalisation of the argument of the sine and cosine functions (which is performed on the phase accumulator of NCOs) it was decided to directly couple the NCO of the carrier recovery block with the one of the downlink mixer (actually using only one NCO in coherent mode) to avoid the risk the two phase accumulators could accumulate different errors due to numerical cancellation (and actually being non synchronous anymore).

3.6 System architecture

This section will focus on the overall implementation of the full software library, providing an in-dept description of all the different blocks shown in Figure 3.11. The general architecture is shown in the figure below and consists of three different blocks:

- A test setup, used to generate real operating conditions to test the transponder block;
- An OBC emulator that is used to control the transponder as closely as possible to the real operational conditions;
- A transponder block that mimics a real hardware TT&C transponder.

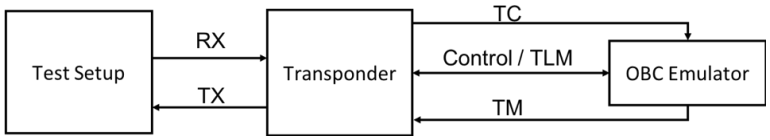

Figure 3.11: General system architecture

The whole architecture is implemented in software, taking into account possible provisions to include hardware-software interfaces (software defined radio peripherals, for example). But no hardware interface will be considered for this project. For the sake of simplicity, the software will be designed considering a zero-IF radio architecture: this implies that the digital system will have input and output signals close to 0Hz and the full RF to zero-IF conversion is handled with a hardware circuit that is not considered in this project.
The implementation is based on GNU Radio as framework for digital signal processing development. This framework will be required to run the application as the developed software depends on library provided as part of GNU Radio. This limits the amount of software than needs to be developed and allows to re-use a wide range of building blocks already built and tested for GNU Radio. As it will be described in the next sections, GNU Radio has a central role as it will be used to run the code implementing the transponder. The external applications (OBC emulator and test setup) will not be implemented in GNU Radio but will interface to it: this limits the required effort and simplifies the integration of eternal applications and peripherals to the transponder.

GNU Radio organises the building blocks under libraries and two libraries, implementing the basic building blocks required for this project, will be implemented. The first one (called gr_ecss to maintain the GNU Radio naming policy) will contain all the blocks relevant to the ECSS standards) while other building blocks (of more general nature and not strictly correlated with ECSS) will be implemented under a second library (gr_flaress).

3.6.1 Configurable parameters

It should be noted that:

- All parameters that are selectable at start-up will have to be selected either on the GNU Radio flowgraph (for the transponder) or on the dedicated application (test setup or OBC emulator). These parameters require no code/logic modification, but simply constitute parameters whose value can only be altered before starting the simulation as they have a big impact on the whole execution.
- All parameters that are selectable at runtime will be accessible either via the OBC emulator: these parameters represent the parameters that normally can be changed in a real space mission plus some extra ones that allow customisations to be performed on the system that would not be doable at start-up. It should be noted also that when a runtime parameter is changed, the time between the command is changed and the transponder implements the change might vary. This depends on the network traffic (since the command is changed through a network connection) and the exact scheduling details (which depend on current CPU load). It is expected from experience that the parameter change will take effect well within one second but an exact execution time is impossible to predict.

3.6.2 Test setup

The Test setup block is responsible of performing the simulation of the environment (Doppler, delay) to ensure the transponder is put in realistic conditions. This block will accept at the input an IQ file containing a Physical Layer Operation Procedure (PLOP-x) to allow the transponder to lock. Since in real life the PLOP foresees a real time feedback from the transponder, the sequence will be hardcoded (to be in "open loop" mode): a clean carrier will be sent for a predetermined time, followed by acquisition sequence, followed by TC, without real-time feedback from the transponder. This recording is not affected by Doppler or Doppler rate, so this will have to be applied based on simulated conditions.

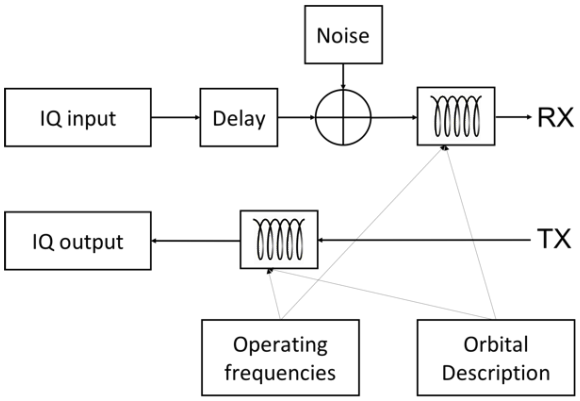

Figure 3.12: Test setup block diagram

The simulation of the Doppler effect for selected test cases, defined by a combination of ground station/spacecraft will be performed using the open-source TUDat software suite (`http://tudat.`

tudelft.nl). This software will be provided with general interfaces, allowing any type of orbit to be used. JSON files will be used to define the orbit by:

- Initial orbital elements (Keplerian, Cartesian, ...);
- Dynamical model settings;
- Simulation time;
- Ground station(s);
- Arcs during which radio data is to be simulated.

Based on these settings, the orbit of the spacecraft is numerically propagated. This orbit will be stored as e.g., Spice kernels, SP3 files. Alternatively, the tool could allow a pre-defined orbit to be loaded from the same types of files.
From the orbital position r and velocity v, and transmitted frequency f, the Doppler shift of the signal Δf is calculated as:

$$\frac{f_2}{f_1} = \frac{\partial \tau_1}{\partial \tau_2} = \left(\frac{\partial \tau}{\partial t}\right)_1 \cdot \left(\frac{\partial t_1}{\partial t_2}\right) \cdot \left(\frac{\partial t}{\partial \tau}\right)_2 \tag{3.3}$$

where τ denotes the proper time, t the coordinate time, and the 1 and 2 subscripts denote transmitter and receiver respectively[36]. Terms in the above equation will be expanded to 1st post-Newtonian order. The Doppler shift will be applied as a time compression / expansion: this process requires a rational re-sampling of the signals in order to simulate the reception of a signal coming from a moving source / observer. The signal will be re-sampled (compressing / expanding) it but it will be processed by all the subsequent blocks using the same sample rate. This procedure simulates the physical effect a hypothetical ADC would see when acquiring the signal. Nominally, numerical performance of this model will nominally be limited by 64-bit floating point precision, although 80-bit precision can be used if needed. Doppler will be calculated based on the operational frequencies but applied on the baseband / intermediate frequency signal. This allows to simulate the effect of Doppler on the RF signal at the output of a direct conversion receiver.
Re-sampling will be performed using a polynomial interpolation to limit the CPU usage. One important criterion when re-sampling a phase-modulated signal is that the polynomial approximations should not introduce discontinuities. Particular care will be taken to ensure the signal is not distorted.
Additive noise can be included in the signal sent to the transponder. Noise distribution and spectral density will be selectable. A delay on the processing chain has also been added to simulate the period before reception of the recorded IQ file. In this period, only noise will be present to test the behaviour of the transponder (random walk of the receiver, time to lock, etc...).
Using this setup, the software transponder will be placed in-the-loop and receive radio data from the simulation (with a given frequency transmitter by the ground station). The retransmitted signal from spacecraft to the ground station will again be simulated using the approach outlined above, to obtain the downlink Doppler shift. These simulations will provide:
Ideal noise-free one- and two-way Doppler observations Doppler observations, as generated with our system in-the-loop
This allows us to quantitatively analyze the impact that our system has on Doppler data quality. Moreover, by adding environment/ground station noise to both the ideal and system-in-the-loop Doppler data, we will be able to determine whether we introduce a dominant noise source and, if so, to what degree this degrades the orbit determination. In doing so we derive, for arbitrary mission geometry/planning, the quality of the data products obtained from the radio data.
At the end of the simulation, data transmitted from the transponder will be recorded again to IQ files to verify the functionality of the complete system Due to the way GNU Radio processes the samples, the first sample on the input port will also be the first sample on the out port (ideally there is no latency due to the hardware) but there is a latency due to the phase shift introduced by all the blocks (computational delay) and this will be simpler to measure.

3.6.3 OBC emulator

The OBC emulator is responsible to emulate a standard On-Board Computer to control the transponder and it needs to provide the following interfaces:

- TC bitstream;
- TM bitstream;
- Transponder telemetry interface;
- Transponder control interface.

The TC bitstream comes from the demodulator and contains only the demodulated bits: this is a "software" replica of the wired connection going from the TT&C transponder to the OBC. The TM bitstream is generated by the OBC and it is sent to the transponder to be modulated: this is the "software" replica of the hardware connection between transponder and OBC. Both bitstream are considered to be required in "real-time" from the OBC so they will be implemented as a continuous stream of data.
Inside the OBC simulator, the TC and TM stream will be mapped to two separate binary files (A file recorded by the OBC emulator for TC, a file played by the OBC simulator for TM).
Telemetry from the transponder can have a dual use:

- it could be used to debug and improve the software model;
- it is needed for the operations of a real OBC.

Due to these two uses, the telemetry connection between the transponder and the OBC will be implemented in both ways, so to provide a good toll for debugging, and to provide a realistic interface to the OBC.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- TC file to record. If no file is selected data will not be recorded to file;
- TM file to play back. If no file is selected, no data will be played back and transmitted by the transponder.

3.6.3.1 GNU Radio implementation

ZeroMQ blocks is used to transmit this information. As GNU Radio will output bits and sockets are based on bytes, the bits will be "packed" into bytes and transmitted (MSB first or LSB first to be decided). The transponder side of the communication channel will conclude with the ZeroMQ block and the OBC simulator will be implemented in a separate flowgraph / application.
ZeroMQ allows also to create a multi-socket connection with one source and multiple receivers (publisher / subscriber structure) that will be used for the OBC emulator. This will allow multiple applications (by means of ZeroMQ as transport layer) to connect to the transponder. This will allow, for example, to create an external application that connects to the TC stream on one side and connects to a real serial port on the other side. Due to the publisher / subscriber model implemented by ZeroMQ, multiple applications could connect to the transponder, giving great flexibility to the setup. The TM stream will be implemented in the same way, allowing external applications to also connect to the transponder. The possibility to have external applications connect to the TT&C transponder will decouple the software and simplify the implementation and debugging of external applications, that do not have to depend on GNU radio code, but only comply with ZeroMQ. The flow control in ZeroMQ (HWM) will be used to ensure buffer overrun are not happening during the simulation.

3.7 Validation tests

Two types of test will be performed to validate the system:

- Reception and transmission using an existing modem: this test requires a recording to be made / played with the modem and using the modem to prove the functionality of the system being developed. These tests are useful to validate the receiver and transmitter implementation against a reference device;
- Ranging / tracking test: these tests cannot be performed using a modem as they cannot be performed in real-time (as the library under development does not require to operate in real-time with a hardware interface). In this test a reference signal will be generated (plain carrier, sequence of tones) and provided t the input of the transponder. The output of the transponder will be recorded and will be compared to the input signal. Two major parameters of interest will be measured: the coherency ratio accuracy (when in coherent mode) by comparing the receiver and transmitter frequencies (all done in software by the test setup) and the transponder delay (considered as phase shift introduced in the baseband signal by the transponder while going from the receiver to the transmitter port). The latter measurement is implemented with a cross-correlation between the input and output signal from the transponder. Since the two signals might be on slightly different frequencies (due to the turnaround ratio), a direct correlation is not possible. A phase demodulator (implemented as a complex inverse-tangent function) will be used on both the input and output to reconstruct the signal phase and then the correlation will be computed on the signal phase. The signal phase is an accurate method till the shift is less than a $2\ \pi$. Longer delays can be still measured by introducing multiple tones and measuring the time-difference between input and output (by simply performing ranging with a zero distance between transmitter and receiver).

The measurement of the coherency ratio and processing delay will be performed fully in the test setup. A modem might be required to generate a realistic input signal (recorded using an SDR to use the IQ files as reference).

Relevant parameters changeable at runtime:

- Signal to noise ratio.

Relevant parameters changeable at simulation start-up:

- Sampling rate;
- Orbital parameters;
- Simulation duration;
- Nominal uplink frequency;
- Nominal downlink frequency;
- Intermediate frequency;
- Input IQ file (when no file is selected, only input noise will be simulated);
- Output IQ file;
- Delay.

3.7.1 Implementation

GNU Radio already provides most of the blocks required to implement this sub-system. In particular:

- File source: to read an IQ file and process it using GNU Radio;
- File sink: to write a GNU Radio stream into a file;
- Multiply: to implement a complex mixer;
- Add: to implement a complex adder;
- Fast Noise Source / Noise Source / Phase Noise Source: different noise sources that could be used in the simulation.

The correlation between the input and output signal (with respect to the transponder) needs to be fully implemented (including a phase demodulator and a correlator capable of estimating the time/phase difference between the signals).

3.7.2 On Board Computer and system control

A final test shall have to be carried out taking a recording (provided by ESA) from a reference ground modem performing ranging/tracking and producing another recording. In such recording, the modem will execute a Physical Layer Operation Procedure (PLOP-x) to allow the transponder to lock. Since in real life the PLOP foresees a real time feedback from transponder, the sequence will be hardcoded (to be in "open loop" mode): a clean carrier will send clean-carrier for a predetermined time, followed by acquisition sequence, followed by TC, without knowing what really happened to the unit. Doppler, Doppler rate and delay will be generated by the software in both the transmission and reception branch. The control of the transponder (selecting if an unmodulated carrier is transmitted, or if the sub-carrier is turned on, etc...) is done by the OBC emulator that connects to the developed transponder. This external application will also allow to record the received bit stream (for testing purposes) and inject a bit stream to test the modulator section.

3.7.3 Target orbits

It is important to develop the required ECSS blocks together with unit test scripts to assess the correctness of the produced code. It is planned to implement a simulator block to apply Doppler, Doppler rate and delay to the transmission and reception path and use a text-based orbital description (for example, SPICE kernels or equivalent systems) to allow to quickly customize the orbital simulation part. To verify this approach, we would like to select a discrete set of orbits (used to generate the Doppler profiles) to verify our implementation. The test setup will also take as input the required transmission and reception frequency to allow the actual Doppler and Doppler rate calculation.

The supported orbits are:

- Circular LEO 300 km altitude
- GEO
- Lunar orbit (TBD)

Chapter 4

TT&C Transponder design

This section describes the designed architecture of TT&C Transponder. In Figure 4.1 it is shown the block diagram. They are also described individually all the blocks compose it analyzing not only its operation but also its implementation on GNU Radio.

It is assumed that the input signal is acquired by the ADC undistorted due to saturation. It is thus assumed that a very coarse AGC is present before the ADC to guarantee the signal can be properly acquired. This AGC is expected to react slowly to signal changes in order not to influence the response of the other blocks to be implemented.

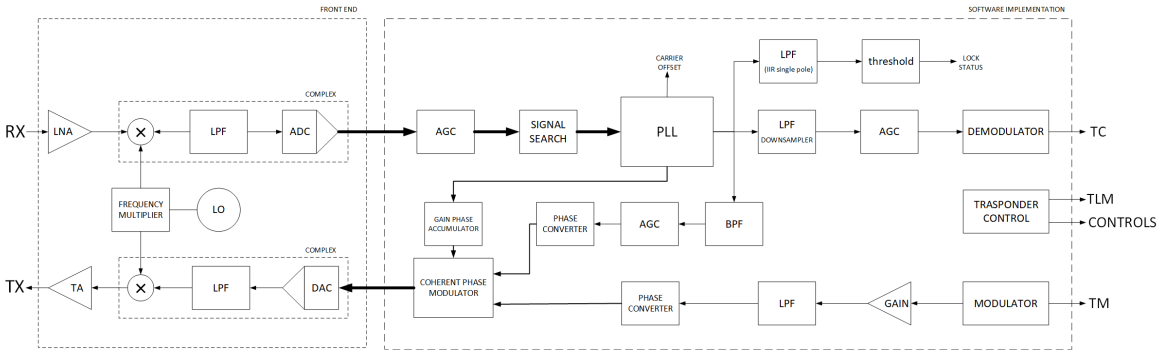

Figure 4.1: Software transponder block diagram

All the real signals were drawn with thin lines, while the complex signals were drawn with thick lines. Furthermore, real-type signals are indicated with medium-thickness lines.
The different sampling rates are also shown in figure: since the ranging channel is considered as the critical one in terms of the required sampling rate, all blocks that are used as part of the ranging channel operate at the same sampling rate. Modulators and demodulators instead operate at a lower rate and reducing the sampling rate will not influence the overall performances but it will reduce the computational load. It should be noted that in the TM channel, upsampling is performed after the modulator (and eventual bit shaping) to limit the computational load on the shaping filter (which scales with the oversampling rate).
The following sections will describe more into details the different blocks, providing insights in the implementation strategy and on the configurable parameters.

4.1 Automatic Gain Control

The AGC block is responsible to adapt the amplitude of the input signal to comply with the required output value. According to the ECSS document for "Space engineering Ranging and Doppler tracking"[28],

clause 4.5.8.a, the AGC shall operate on the RMS value of the input signal and keep it constant.
Three different AGCs are present in the transponder. They are used for different purposes and require different settings. The purpose of the different AGCs is as follows:

- AGC 1: is needed to guarantee that the input signal has a constant amplitude to guarantee optimal operations of the carrier recovery PLL;
- AGC 2: is needed to guarantee that the TC signal has a constant amplitude to guarantee optimal performances of the demodulator. When the ranging tones and the TC signal are both present, the amplitude may vary and this can vary the loop bandwidth of the demodulator;
- AGC 3: is needed to guarantee that the ranging tones, after being demodulated, have a constant amplitude not to vary the modulation index of the signal being re-modulated. The AGC output value can also be set to vary the modulation coefficient of the ranging tones.

Detailed specifications for AGC 2 were not found in ECSS documents "Space engineering Radio frequency and modulation"[31] and CCSDS document "Pseudo-Noise (PN) Ranging Systems"[26] while a detailed specification for the attack time were found for AGC 3 in ECSS document "Space engineering Ranging and Doppler tracking"[28] (clause 4.5.8.b): the attack time should be between 10 ms and 30 ms. According to the latter document, clause 4.5.8.a, the AGC shall guarantee a constant NanoSats value at the output.
All AGCs are implemented using the same block but with different parameters. Since AGC 3 is the most constrained one (in terms of attack time, selected to be 20 ms), AGC 1 will use a slower attack time (100 ms, not to distort the signal) and AGC 3 will use the same attack time (20 ms).

4.1.1 Design

The automatic gain control (AGC) is a closed loop feedback control circuit, whose purpose is to maintain an amplitude of the output signal at a fixed value, despite the variation in the amplitude of the input signal. The RMS output signal level is used to dynamically adjust the gain of the amplifiers[37].
In addition, it is chosen to use an logarithmic since it makes the settling time of the AGC insensitive to the input signal amplitude[38]. Thus, the constant time depend only on the internal gain (rate coefficient).
Furthermore, in order to make simpler the use of that block, it is used as input the expected attack time. Thus, it is internally got the internal gain value. The attack time of the AGCs will be controlled via variables, so it will be easy to set to different values to optimize for performances.

The AGC will be implemented using a simple first order filter in the gain control loop, whose time constant is equal to:

$$\tau = \frac{t_{attack}}{2.95} \simeq \frac{1}{\alpha} \tag{4.1}$$

from which it is obtained:

$$\alpha = \frac{2.95}{f_s \cdot t_{attack}} \tag{4.2}$$

The Equation 4.1 is derived from the approximated attack time for a first order filter, considered as the time required for the response to rise from 5% to 95% of its final value. Thus, the internal gain α is given by the Equation 4.2, where, f_s is the sampling rate. The basic block diagram on the AGC block is shown in Figure 4.2.

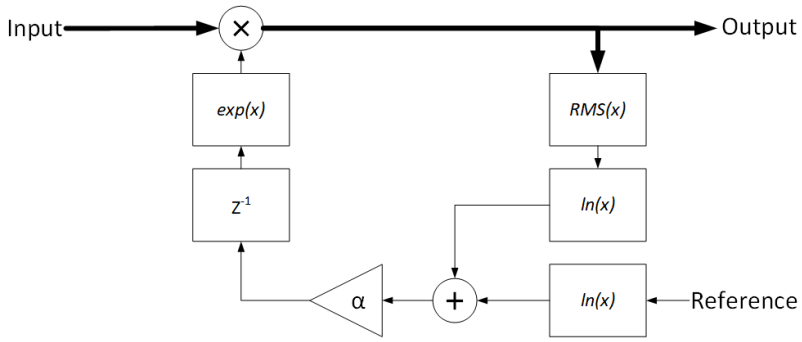

Figure 4.2: AGC block diagram

4.1.2 GNU Radio implementation

GNU Radio already has a block performing the AGC function: the block has been tested and it implements the required function. The input parameters of the GNU Radio block are the reference level desired (RMS value), minimum and maximum gain and a rate coefficient that specifies the attack rate. This is equivalent to an integrator with an adjustable loop gain (realizing a first order filter). Due relative complexity of specifying the desired attack rate in milliseconds (which requires a formula to be manually typed at every new instance of the existing GNU Radio block), it was decided to realize a dedicated block, adding the required formula to simplify the block usage. In addition, a logarithmic and exponential block is added to implement a constant attack time, according to[38]. Indeed, in this way, the AGC's constant time will depend on the rate coefficient only, without depending on the input amplitude.

In addition, in a real case behaviour zero values can occur since a logarithmic approach has been used. This would mean an explosion of numeric value of gain in the logarithmic conversion. Thus, it is important to make settable a maximum gain that the AGC can have. In a real use, this will be proportional to the output stability. More precisely, the maximum gain will be also the value of gain when a zero value input is received. Since the AGC has a first order response with settable time constant, with a too high maximum gain, also the following items will be multiplied by a high gain. By setting a lower maximum gain, those cases can be kept under control, obviously it will be necessary to set the maximum gain on the basis of input amplitude.

Relevant parameters changeable at runtime:

- Attack time: it is the expected settling time value, expressed in seconds;
- Reference value: it is expected output RMS value.

Relevant parameters changeable at simulation start-up:

- Sample rate: it is the sampling rate, expressed in samples per second.
- Initial gain: it is the initial gain of the block;
- Maximum gain: it is the maximum gain of the block;

4.2 Filters

4.2.1 Band-Pass Filter

The band-pass filter is used to filter the ranging tones and eliminate the telecommand eco in the downlink.

4.2.2 GNU Radio implementation

This blocks can be implemented with a GNU Radio Band Pass Filter blocks. These blocks implement FIR filters which are designed according to the parameters provided in the configuration panel. Bandwidth and attenuation can be set in Hertz. These blocks cannot be completely updated in real-time but only when the flow graph is started. This is an inherent limitation of the system and would prevent changing the filtering options at runtime: this will require the user to configure the bandwidth of the filter (based on the selected symbol rate and ranging tone frequency) at startup.

Relevant parameters changeable at runtime:

- Gain: this is the filer's gain;
- Low cut-off frequency: this is the low cut-off frequency of the band pass filter;
- High cut-off frequency: this is the high cut-off frequency of the band pass filter;
- Transition width: this is transition width of the band pass filter for both the edges;
- Window: this is the windowing type of band pass filter (Hamming, Hann, Blackman, Rectangular, Kaiser);
- Beta: this is the beta parameter only applied to the Kaiser window.

Relevant parameters changeable at simulation start-up:

- FIR type: this is the FIR filter type, it can be an either interpolator or decimator and it can have either real or complex taps;
- Decimation: this is the downsampling rate;
- Sample rate: this is the sampling rate and it is expressed in samples per second.

4.2.3 Low-Pass Filter Downsampler

The low-pass filter is used to limit the interferences the ranging tones might have on the TC. Additionally,it is also used to perform downsampling as part of the filtering process. This reduces considerably the computational load.
This filter should to be a polyphase FIR filter, since it is very effective when downsampling is required.

4.2.4 GNU Radio implementation

These blocks can be implemented with a GNU Radio Low Pass Filter blocks. These blocks implement FIR filters which are designed according to the parameters provided in the configuration panel. Unfortunately, it will be not implemented as a polyphase filter.

Thus, it is possible to use the GNU Radio Polyphase Decimator Block. This block creates a polyphase filter with the purpose of decimating the input signal. Into this filter is possible to set the option to use an FFT filter instead of a FIR. Actually, this block requires as a parameter the taps to be used as a filter to be converted into polyphase, and if the FFT option is selected, automatic conversion to an FFT filter will be performed. In order to make simpler the uses of this block, can be used the GNU Radio Low-pass Filter Taps block to generate the filter taps. It generates automatically the taps of the designed FIR filter as variable that shall be put as parameter of Polyphase Decimator Block. The parameters indicated below are to be considered valid for the combination of both blocks proposed.
FFT filters perform better for larger numbers of taps but is architecture-specific. Generally, for more than 30 taps, the FFT implementation is faster than others[39]. Since this block is part of the channelizers family, it also offers the possibility of FFT rotation. In case of a high number of channels, the FFT method will perform better. Generally, this value of channel is small (~ 5), but it is architecture-specific.

Relevant parameters changeable at runtime:

- Gain: this is the filer's gain;
- Cut-off frequency: this is the cut-off frequency of the low pass filter;
- Transition width: this is transition width of the low pass filter;
- Window: this is the windowing type of band pass filter (Hamming, Hann, Blackman, Rectangular, Kaiser);
- Beta: this is the beta parameter only applied to the Kaiser window;

Relevant parameters changeable at simulation start-up:

- FIR type: this is the FIR filter type, it can be an either interpolator or decimator and it can have either real or complex taps;
- Decimation: this is the decimation rate;
- FFT filters: use FFT filters (fast convolution) instead of FIR filters;
- FFT rotator: rotate channels using FFT method instead of e^{π};
- Stop-band attenuation: this is the attenuation in the stop band;
- Output channel: selects the channel to return;
- Sample delay: this is delay of samples;
- Sample rate: this is the the sampling rate and it is expressed in samples per second.

4.3 Coherent Phase Modulator

The phase modulator is used as modulator and mixer to translate the composite signal (ranging and TM) to the correct frequency. This modulator accepts one or several signals as input, and uses all of them to drive a phase modulator. The phase inputs expect a phase scaled between $-\pi$ and $+\pi$ the block produces a complex signal (I and Q) generated by adding all the input phases and translating them to a complex signal.

4.3.1 Design

The phase modulation is performed using sine and cosine functions driven by the input phases. In particular, to perform a coherent phase modulation, the transponder has been designed in order to be able to reuse the phase accumulator value of PLL block, properly adjusted taking into account the Turn Around Ratio by Gain Phase Accumulator block. To simulate a real hardware implementation, where a fixed-point representation will be adopted, the phase input is represented by a finite number of bits (up to 52 bits, since it is the significant precision of double-precision floating-point format[40]), which corresponds to a phase normalized between $-\pi$ and $(\pi - 1LSB)$ signed fixed-point integer).
The outputs, more precisely the sine and cosine values, are expressed as floating-point numbers and no other quantization is applied in this block. This solution will reproduce all the artefacts present in modulators with a limited bit width. This solution is similar to the phase accumulator used in the PLL block.

The Coherent Phase Modulator block diagram is shown in Figure 4.3.

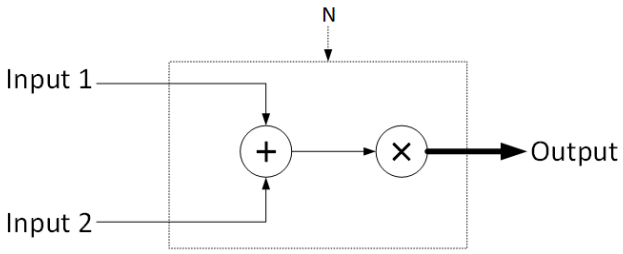

Figure 4.3: Phase modulator block diagram

4.3.2 GNU Radio implementation

This block has been implemented completely in GNU Radio. Moreover, a no traditional approach has been used in order to emulate properly a fixed-point mathematics to emulate fixed point mathematics, 64-bits integer mathematics was needed.
The use of 64-bits numbers, theoretically supported by GNU Radio, but practically not used by any existing block, involved the creation of new blocks that could handle these types of numbers. Since the the phase accumulator output of the PLL is "Int64", at least one of the input of the Coherent Phase Modulator must be "Int64". Since the number of inputs is user-selectable, it was decided, for the sake of simplicity, that all the inputs have to be of the same type. Otherwise, between the PLL and the Coherent Phase Modulator, a de-normalization and normalization would have been necessary. By introducing unnecessary mathematical errors due to conversions. Consequently, an external block to normalize real numbers has been developed.
The implementation of this block is identical to the NCO used in the PLL to ensure a coherent signal can be generated with no artifacts.

This block does not require the sampling rate to be set as it is assumed that all sources will produce samples at the same rate and that an output sample will be produced per each set of samples at the inputs. However, the number of bits is required in input tp properly denormalize the phase step obtained by denormalizing all the inputs.

Relevant parameters changeable at runtime

- None.

Relevant parameters changeable at simulation start-up

- Ninputs: number of input signals.
- N: number of bits used for the fixed-point representation.

4.4 Demodulator

According to the ECSS document "Space engineering Radio frequency and modulation"[31], the supported modulations for TC are:

- 4 $kbits/s$ and lesser: PCM/PSK/PM with NRZ-L/M;
- For data rates higher than 8 $kbits/s$ and lower than 256 $kbits/s$: PCM/PSK/PM with SP-L.

The BPSK modulation on the sub-carrier will be demodulated using a Costa's loop operating on the sub-carrier frequency (16 kHz nominal, user-settable). SP-L on the carrier will require no extra demodulation step but only clock recovery. As stated in the ECSS document "Space engineering Radio frequency and modulation", Table 6-2, the sub-carrier waveform for TC is sine.

Due to the different demodulators, which are both present at the same time in the flow graph but only one of them operates at a time, selector blocks are necessary to decide which branch of the flow graph is used. These selectors are controlled, with variables, by the central control block in the transponder. The input signal coming into the demodulator is a complex stream coming from the first demodulation stage and after an AGC stage to guarantee minimal variations to the signal amplitude (which might cause problems to the PLL and Costa's loop). The output signal from the modulator is formed by demodulated bits (non-packed into bytes but transmitted as 1 bit per byte).

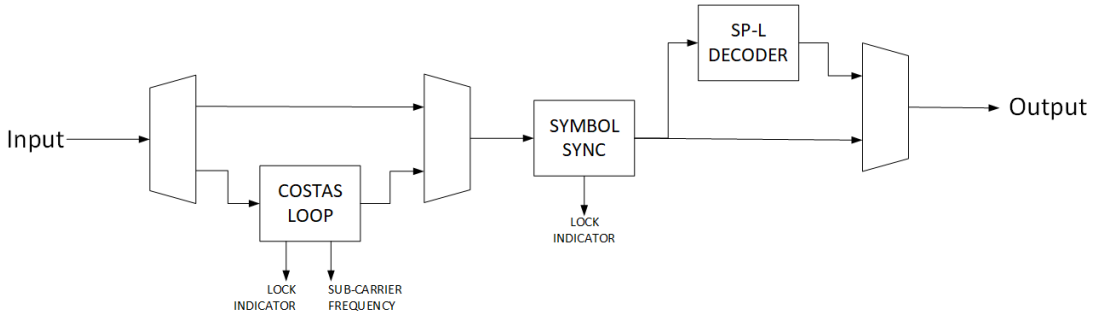

Figure 4.4: Demodulator block diagram

Relevant parameters changeable at runtime

- None.

Relevant parameters changeable at simulation start-up

- sub-carrier frequency: select the sub-carrier frequency;
- Sample rate: expressed in samples per second;
- Data rate: selectable parameter, also selects the modulation.

4.4.1 Costa's loop

The Costa's loop is used to demodulate fully suppressed carrier modulations and, in this case, it will be applied to demodulate the sub-carrier. Costa's loop can operate in a similar way as a PLL, recovering the incoming carrier and then demodulating it.

4.4.1.1 GNU Radio implementation

This block is already present in GNU Radio Band and it can be fully reused. GNU Radio uses a second order loop filter with selectable bandwidth and critical dumping.

Relevant parameters changeable at runtime

- Lock frequency: frequency range in which the PLL is allowed to lock
- sub-carrier frequency: selectable, depending on data rate

Relevant parameters changeable at simulation start-up

- Sample rate: expressed in samples per second.

4.4.2 Clock recovery

The clock recovery block is used to acquire the data clock and, based on that, sample the demodulated stream. Based on the trade-off, the block to be implemented shall support at least the following algorithms:

- DTTL (Data Transition Tracking Loop)[41];
- Early-late;
- Gardner.

4.4.3 Clock recovery algorithm

Several clock recovery algorithms are used in digital radios but the most common ones are:

- DTTL[41];
- Early-late;
- Gardner;
- Mueller&Mueller.

DTTL is the most common algorithm used in space applications and it has a very long heritage[42]. It is in principle very similar to a PLL in implementation and shows good performances[43] with low SNR. Several modified versions exist that improve slightly the performances, especially at low SNR: in principle these implementations would be beneficial for this project but, in typical situations, the SNR will always be approximately 10 dB (due to the absence of coding on the TC channel). DTTL is in general performing better with modulations that do not present shaping.
Gardner is a very common algorithm in digital receivers as well and it is similar in performances and complexity with respect to DTTL[43]. Gardner can operate with as few as 2 samples per bit, which is an advantage over DTTL which is usually used at higher numbers of samples per bit. Gardner operates generally very well in case of shaped modulations (like raised cosine filtering with PSK modulations) but works also in the more generic case of unshaped modulations.
Early-late is probably the simplest algorithm in use for clock recovery and it could also operate with as few as 2 samples per bit. Its complexity is the least and performances found in literature are also similar. Early-late operates generally very well in case of shaped modulations (like raised cosine filtering with PSK modulations) but works also in the more generic case of un-shaped modulations.
The last of the algorithms commonly used is Mueller&Mueller, which has a considerable complexity due to its rational re-sampler block used to calculate the optimal sampling point.
Considering the already available implementations, GNU Radio features only Mueller&Mueller but this has been reported by several users to require a very high CPU load.

4.4.3.1 GNU Radio implementation

In recent literature survey, it was found that a new clock recovery block was included to GNU radio called "Symbol Sync" which is configurable with various types of Timing Error Detectors (TED) methods and interpolating re-sampler[44]. This block already implements:

- Mueller and Muller;
- Modified Mueller and Muller;
- Zero Crossing;
- Gardner;
- Early-Late;
- D'Andrea and Mengali;
- Maximum Likelyhood.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- Clock recovery algorithm: select which algorithm to use for lock recovery;
- Data rate: selectable parameter, also depending on the modulation type;
- Sample rate: expressed in samples per second.

4.4.4 Modulator

The supported modulations for TM are:

- BPSK/PM on a sub-carrier up to 300 kHz for rates below 60 $kbps$;
- SPL or BPSK (on the carrier) for rates > 60 $kbps$.

The modulator, shown in Figure 4.5, operates on a similar principle, taking bits to be transmitted as input (non-packed into bytes but transmitted as 1 bit per byte) and providing as output the phase information for the following phase-modulation process. The modulator block will support 3 types of modulating signals (SP-L, NRZ-L on the main carrier and NRZ-L on a square wave sub-carrier) with an optional convolutional encoder and Root-Raised Cosine filtering (optional).
The modulator produces a signal on the output with unitary amplitude and, together with a gain block on the TM channel (used to select the modulation index, specified in the ECSS document "Space engineering Ranging and Doppler tracking"[28] to be between $0,1$ rad to $0,7$ rad) and the phase modulator, allows to generate the complete TM modulation.

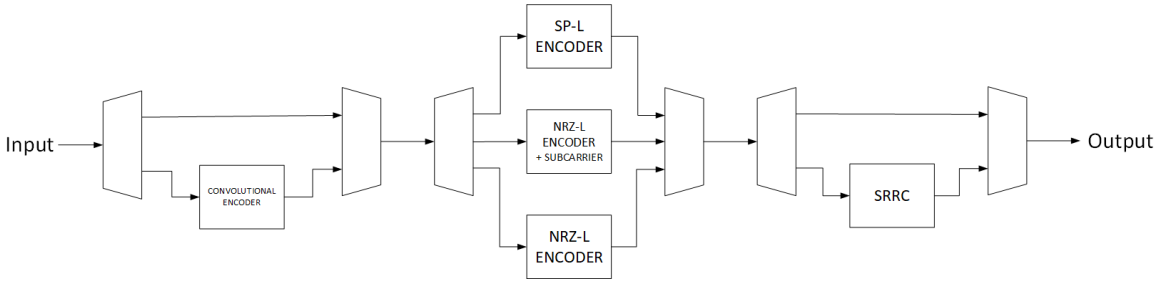

Figure 4.5: Modulator block diagram

4.4.4.1 GNU Radio implementation

The block will be implemented with a hierarchical block and several sub-blocks, in particular:

- Convolutional encoder (already present in GNU Radio);
- SRRC filter (already present in GNU Radio);
- Selector;
- SP-L encoder;
- NRZ-L encoder (used, together with the gain block on the TM channel and the phase modulator to produce BPSK);
- sub-carrier generator: signal source locked to the bit clock used (together with to generate a BPSK modulated sub-carrier.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- Convolutional encoder enable: enable the convolutional encoder;
- SRRC filter enable enable the pulse shaping;
- SRRC rolloff-factor: set the roll-off factor for the SRRC filter;
- sub-carrier frequency: select the sub-carrier frequency;
- Sample rate: expressed in samples per second;
- Data rate: selectable parameter, also selects the modulation.

4.4.5 SP-L encoder

SP-L encoding is performed using a dedicated blocks. SP-L (Or also Manchester encoding) encodes one bit as a transition, being it from Level A to Level B or the opposite way. Figure 4.6 shows how the encoding is performed. This block will transform every input value (one bit of information) into two values (considering no oversampling).

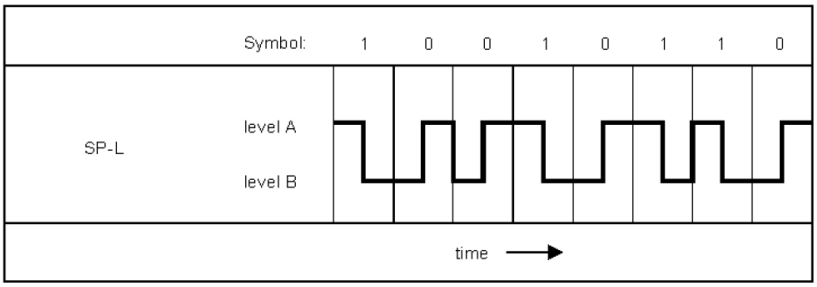

Figure 4.6: SP-L encoding[31]

4.4.5.1 GNU Radio implementation

The block is fully implemented. It considers each input value higher than '0' as '1' and equal or lower than '0' as '0'. And it converts them into two samples ('+1' and '-1' or '-1' and '+1', level A and B in Figure 4.6).
In order to improve the efficiency, in the same block is performed the repetition of the samples in order to satisfy the set bit rate. Indeed, into that GNU Radio block, it is generated once at start-up a couple vector of length as ratio between sample rate and bit rate. Indeed, one of these vectors is simply the first half filled by '-1' and the other half by '+1', and the second the opposite. By depending of input value, one of them is simply copied as output.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- Bit rate: it is the bit rate value of encoder, expressed in bit per second;
- Sampling rate: it is the sampling rate of encoder, expressed in sample per second.

4.4.6 NRZ-L encoder

This block is used as an input to the phase modulator to generate the corresponding PSK. As stated in the ECSS document "Space engineering Radio frequency and modulation"[31], '1' is represented by level-A which corresponds to '+1' and '0' is represented by level-B which corresponds to '-1'. In the case of BPSK modulation, this input is used to produce the $\pm 180°$ phase shift. The full modulation (mapping the output of the NRZ-L encoder to a phase) will be done using a gain block in the TM chain that sets the modulation index.

4.4.6.1 GNU Radio implementation

A similar block is already provided by GNU (Constellation Modulator). Unfortunately, it generates already the full modulation (from symbols to a phase modulated complex number) while the process in this case has to be performed in two steps: digital value to phase representation and phase modulation. This particular block will map the incoming bits to the required phase while the modulation will be performed by a different block. This block will thus only perform NRZ-L encoding.

Similarly to the SPL encoder, this block generates two vectors of length as the ratio between sampling rate and bit rate at start-up. In this encoder, the all the values in the vector are similar and they are '+1' or '-1'. Indeed,

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- Bit rate: it is the bit rate value of encoder, expressed in bit per second;
- Sampling rate: it is the sampling rate of encoder, expressed in sample per second.

4.4.7 NRZ-L encoder with sub-carrier generator

According to the ECSS document "Space engineering Radio frequency and modulation"[31] clause 6.1.4.1.1, the sub-carrier generator block is an oscillator whose frequency is an exact multiple of the data rate. The output of this block will be phase modulated by an NRZ-L encoded signal. The sub-carrier to be generated can be either a square wave or a sinewave.
to enable the oscillator only when a sub-carrier is needed (when a data rate is lower than 60 *kbps*[31];

4.4.8 GNU Radio implementation

This block does not exist in GNU Radio and it will be fully developed. It will be made by a digital oscillator (sine or square wave) with its phase controlled by the input bit stream (to align phase jumps to bit transitions).

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- Bit rate: it is the bit rate value of encoder, expressed in bit per second;
- Sampling rate: it is the sampling rate of encoder, expressed in sample per second.
- Frequency sub-carrier: it is the frequency of the sub-carrier, expressed in Hz;
- Wave type: it sets the wave type of sub-carrier, it can be a square or sine wave.

4.4.9 Convolutional Encoder

The convolutional encoder block is placed at the input of the modulator and it constitutes the first block that receives the TM bits from the OBC. This block presents also an extra input signal to enable / disable it. This is needed since the convolutional encoder is normally used but it could also be disabled by the OBC. This extra input is used to replace the encoder with a transparent connection. Bypassing the convolutional encoder also impacts the sample rate at the modulator, as the sample rate is not doubled by the encoder.

4.4.9.1 GNU Radio implementation

A convolutional encoder is already present in GNU Radio, as shown in Figure 4.7. The block operates on bits at the input (provided as a full byte) and produces bits at the output (as bytes, only 1 bit out of 8 is used). The encoder can be configured using the Encoder definition block that can update the parameters.

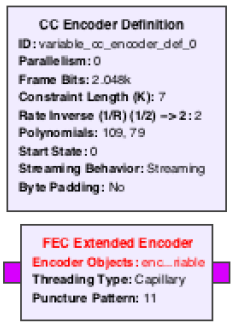

Figure 4.7: GNU Radio convolutional encoder block

Currently the GNU Radio implementation presents a bug: the polynomials are swapped. This can be seen in Figure 4.7 as well, where the first polynomial should have been 79 (octal) and the second 109 (octal). This implementation mistake can be corrected easily by enclosing the standard GNU Radio implementation in a hierarchical block and swapping the polynomials in the configuration.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- Enable / disable: enable / disable the encoder. When disabled, the output of the encoder is identical to the input;
- Polynomials: select different polynomials than the standard ones;
- Puncturing: select different puncturing matrixes than the standard one.

4.5 Gain

The gain block is in general used to adapt the value of a signal to the required amplitude. In the transponder it is used to select the desired modulation index of the TM modulated stream, before it is summed and phase-modulated. The gain will be set using a GNU Radio variable: this makes the implementation very simple and also allows the variable to be set using the XML RPC command interface or a graphical user interface. The phase modulator expects an input in phase (limited

between 0 and 2π) while the modulators generate a signal between -1 and $+1$. The gain block can be used, for example to translate the modulated signal (-1 to $+1$) into a phase between π and -$-\pi$ setting a gain equal to π.

4.5.1 GNU Radio implementation

The block can be implemented with a GNU Radio Multiply Const block and requires no modification.

Relevant parameters changeable at runtime:

- TM Modulation index: specifies the gain of the gain block.

Relevant parameters changeable at simulation start-up:

- None.

4.6 Gain phase accumulator

The Gain phase accumulator block is used to generate the signal to phase-modulate the TM signal. This block is responsible for the modality of transmission of the transponder. Indeed, this block can decide if the downlink is coherent with the uplink, or not. Additionally, it can generate a signal that, through the Coherent Phase Modulator, can shift the downlink frequency in order to adjust / correct it as correction of the downshifting in base-band of the uplink. Moreover, in case of coherent mode, this block will be responsible to take in account the turn around ratio, so, the fixed ratio between the uplink and downlink.

4.6.1 Design

This block outputs the phase of the carrier that shall be either constant (0 Hz, in case of non-coherent operations and without any frequency correction) or that can come from the phase accumulator of the PLL (in case of coherent operations). In case of non-coherent applications, a null constant value of phase is used to force the Coherent Phase Modulator block afterwards to generate a fixed frequency signal.

4.6.2 GNU Radio implementation

A block diagram is shown in Figure 4.8. The block shall allow two settings: the turnaround ratio (implemented as a gain block on the PLL phase accumulator to guarantee TC and TM are phase-locked, but on different frequencies) and the coherency mode (ON or OFF, depending on the transponder settings).

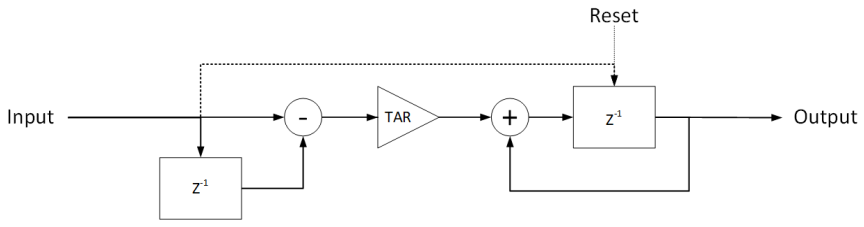

Figure 4.8: Coherent frequency generation block diagram

Since is no correct to directly multiply the PLL phase accumulator output with the turn around ratio, it is needed to recover the phase step of the PLL phase accumulator. To evaluate it, a derivation has to be performed, so, the step phase has to be multiplied by the turn around ratio, and finally,

integrated. In addition, in the coherent mode, it is mandatory to have a phase sync between the uplink and downlink. In order to satisfy this recruitment, in the exact moment when the coherent mode is turned on, the PLL phase accumulator value is copied in the phase accumulator of this block. In this way, the starting point is the same for both and the synchrony is satisfied.

Since the z^{-1} block is directly linked to the output, and it is practically a register, so the reset of this register is used to perform the non-coherent mode. Indeed, when the reset is on, the output of register, so of the block, will be zero.

Moreover, since both input and output are Int64, in order to perform the calculus with the maximum accuracy, only integer operation are performed inside the block. Thus, the phase step (in integer) is divided by the uplink value, so, the result is multiplied by the downlink, by exploiting the self-wrapping of the ALU with a possibly overflow.
Finally, the drawback is that in the derivative evaluation, a delay of one step occurs. That is not a problem since it is a fixed delay.

Relevant parameters changeable at runtime:

- Reset: select if the transponder is operating in coherent mode (ON/OFF);
- Uplink: this parameter define the turn around ratio, that is the correction factor used when rescaling the TC PLL phase accumulator. This is the value that divide the phase step.
- Downlink: this parameter define the turn around ratio, that is the correction factor used when rescaling the TC PLL phase accumulator. This is the value that multiply the phase step.

Relevant parameters changeable at simulation start-up:

- None.

4.7 Lock detector

The lock detector is usually an integral part of a PLL, being linked to the phase error signal of the PLL. In out implementation, we selected to use an external lock detector (external as not being integral part of a PLL) to allow for a greater flexibility.
The lock detector will be implemented using already present GNU Radio blocks.
Figure 4.9 shows a block diagram on the lock detector that will be used.

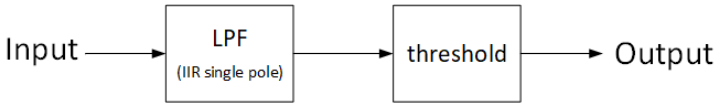

Figure 4.9: Lock detector block diagram

4.7.1 GNU Radio implementation

The block will be implemented using already existing blocks, in particular a low-pass filter and a threshold indicator. They are both part of the standard GNU Radio library. The low-pass filter has been selected as being a first-order filter for the implementation, but this will be left customizable for the user.

Relevant parameters changeable at runtime:

- Lock threshold: threshold of the comparator used to report the lock condition.

Relevant parameters changeable at simulation start-up:

- Low-pass filter cutoff frequency: used to adjust the response of the lock detector;
- Low-pass filter order: used to adjust the response of the lock detector;
- Sample rate: expressed in samples per second.

4.8 Phase-Locked Loop

The PLL designed has to perform several functions, first of all it has to shift the input signal in baseband. Additionally, it has to output both the frequency value of the PLL (the one that is multiplied with the input signal in order to shift it), the phase accumulator value (needed by the frequency correction block, to produce coherent downlink) and the phase error output (useful for the phase demodulation and to evaluate externally the lock detection signal). Furthermore, it is used a flexible architecture for the PLL. First of all, it is designed a Loop Filter that can be set to second or third order. Moreover, in the second order modality it is allowed an imperfect integration in order to reduce the free-running noise[45]. On the contrary, in the third order modality, the imperfect integration is not allowed since it is used only when the PLL is not locked yet.

The PLL block is responsible for performing the carrier tracking/recovery of the incoming signal. Given that a residual carrier is employed in the uplink signal, a phase locked loop is used. This block will receive as input the digitized RF stream (I/Q) in quasi-baseband (considered as a frequency close to 0 Hz, but not exactly 0 Hz) and it will output 4 signals: the signal multiplied by the recovered carrier (baseband), the recovered carrier frequency (used as telemetry value and to control the uplink carrier in coherent mode), the phase error output (used in the phase demodulation step) and the phase accumulator of its numerically controlled oscillator.
This block requires the input to be normalized to prevent variations of the loop gain and so it requires an AGC at its input.
This block is used to recover the TC carrier and, based on its frequency and phase once in coherent mode, to regenerate the TM carrier. Due to this, the synchronization between the TC and TM carrier is extremely important: for this reason, it was decided to directly use the phase accumulator on the internal NCO to drive the TM mixer. This solution was selected to prevent possible numerical errors that can introduce a drift between the TC and TM frequencies.

4.8.1 Design

The simplified closed-loop transfer function H(s) Software Phase-Locked Loop is:

$$H(s) = \frac{\alpha F(s)}{s + \alpha F(s)} \tag{4.3}$$

Where $F(s)$ is the transfer function of the loop filter. Whereas α is the product of gain K_0 of the phase accumulator and the gain K_d of the phase detector. The latter is a constant, and is fixed to one for the sake of simplicity. Furthermore, also K_d is equal to one in this design, being the arctangent of the product of the input with the signal locked by the PLL. Indeed, the arctangent for small value of phase error, can be approximated to one considering that both the signal have an unitary amplitude, thanks to the AGC in input. Thus, the order of loop will be always an order more than the filter's loop.
According to the sampling theorem, the whole algorithm of the Software-PLL must be executed at least twice for each cycle of the reference signal[46]. According to ECSS, the sampling rate should to be around 10 $Msps$ in order to cover properly all the sub-tones.
At so high sampling rate, the input noise power density spectra (B_i) is going to be quite big. This means, that compared with the power density spectra of the signal, which is essentially a carrier in

this case, will yield on a negative $SNR_{i_{dB}}$. Since is intrinsically related to the sampling frequency bandwidth. One of the main advantage of the PLL is is its ability to filter noise. Indeed, ony the noise in equivalent noise bandwidth of the PLL (B_L) is able to corrupt the PLL performances[46], and it is defined by the loop transfer function of the PLL.

By looking the Figure 4.1, can be found out that actually there are not filters before the PLL stage. Thus, the signal acquiring performances of the transponder are directly related to the PLL's one. In fact, Signal to Noise Ratio (that is this case can be considered as the Carrier to Noise Ratio) to which the transponder can intercept the incoming carrier is the equivalent one defined by the PLL (SNR_L):

$$SNR_L = \frac{P_s}{Pn} \cdot \frac{B_i}{2B_L} = SNR_i \cdot \frac{B_i}{2B_L} \tag{4.4}$$

$$SNR_{L_{dB}} = SNR_{i_{dB}} + 10log_{10}\left(\frac{B_i}{2B_L}\right) \tag{4.5}$$

$$\overline{\theta_n'^2} = \frac{1}{2 \cdot SNR_L} \tag{4.6}$$

where $\overline{\theta_n'^2}$ is the phase jitter at the output of PLL.
A block diagram of the PLL to be implemented can be seen in Figure 4.10.

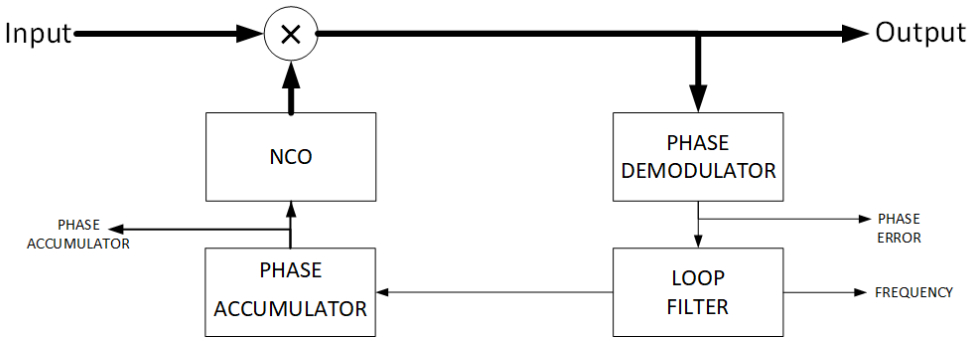

Figure 4.10: PLL block diagram

4.8.1.1 Loop Filter

The loop filter function is to filter the phase detector's output in order to get mainly the DC component, since it is roughly proportional to the phase error (θ_e). As explained hitherto, the Loop filter defines the equivalent noise bandwidth of the PLL, which is approximately equal to the double of -3 dB bandwidth of the PLL.
Loop filter is generally a PI filter, since his particularly easy to implement in software. Moreover, even if the filter transfer function has a pole in the origin, the loop transfer function will have a pole not in the origin, as well as a normal low-pass filter.

A second order loop has a good step response and it is stable. This makes it suitable for most situations. Unfortunately, the gain rolls off of the first order filter is only -20 dB/dec, the spectral purity of the PLL can have spurious side-bands which can establish errors. For more critical application third order loops have been used, since allow a better noise suppression and steady stare error (in particular with respect to a frequency ramp). However, the loop stability becomes an issue.
In this project, an hybrid PLL has been developed to allow both second and third order loops, and an imperfect integration. Finally, it uses an integer phase accumulator in order to emulate a fixed point mathematics in order to exploit its self-wrapping feature. In fact, in the wrapping, due to the floating-point mathematics nature, some calculation errors can occur.

Second order loop The Equation 4.7 shows the transfer function of a second order loop PLL with a PI filter.

$$H(s) = \frac{2s\zeta\omega_n + \omega_n^2}{s^2 + 2s\zeta\omega_n + \omega_n^2} \tag{4.7}$$

where, ω_n is the natural frequency and ζ is the damping.

$$B_L = \frac{\omega_n}{2}\left(\zeta + \frac{1}{4\zeta}\right) \tag{4.8}$$

Where B_L is the equivalent noise bandwidth. In addiction, by assuming an high-gain loops, we get:

$$\Delta\omega_n \approx 2\zeta\omega_n, \quad T_L \approx \frac{2\pi}{\omega_n} \tag{4.9}$$

where $\Delta\omega_n$ is the lock range and T_L lock-in time (also referred as settling time).

Perfect Integrator The transfer function for a perfect integrator, that in this case is equal to the PI filer's one, is expressed in the Equation 4.10.

$$F(s) = \frac{1 + \tau_2 s}{\tau_1 s} \tag{4.10}$$

it has a pole in the origin and a zero in correspondence of $1/(2\pi\tau_2)$.
An digital version of the PI filter is shown in the Figure 4.11. Whose transfer function in the Z-domain is expressed in the Equation 4.11.

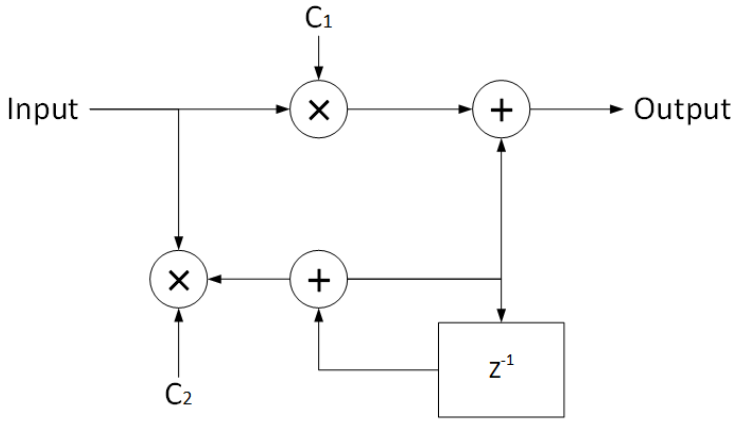

Figure 4.11: Perfect integrator implementation in Z-domain

$$F(z) = C_1 + \frac{C_2}{1 - z^{-1}} \tag{4.11}$$

In oder to explain the transfer function in the Z-domain through the Laplace domain, applying the Tustin's transformation shown in Equation 4.12. Through which it is possible to obtain the sequential equalities:

$$\frac{1}{s} \rightarrow \frac{T_s}{2}\frac{1 + z^{-1}}{1 - z^{-1}} \tag{4.12}$$

where T_s is the sampling period. Finally, it is possible to express the the damping and the natural frequency as function of the gains in the Z-domain[47]:

$$C_1 = \frac{8\zeta\omega_n T_s}{\omega_n^2 T_s^2 + 4\zeta\omega_n T_s + 4} \tag{4.13}$$

$$C_2 = \frac{4\omega_n^2 T_s^2}{\omega_n^2 T_s^2 + 4\zeta\omega_n T_s + 4} \tag{4.14}$$

where, $K_0 K_d = 1$ in the expressed solution.

Imperfect Integrator The transfer function for a imperfect integrator, is expressed in the Equation 4.15.

$$F(s) = K \cdot \frac{1 + \tau_4 s}{1 + \tau_3 s} \tag{4.15}$$

it has a pole in correspondence of $1/(2\pi\tau_3)$ and a zero in correspondence of $1/(2\pi\tau_4)$.
An digital version of the PI filter with imperfect integration is shown in the Figure 4.12. Whose transfer function in the Z-domain is expressed in the Equation 4.16.

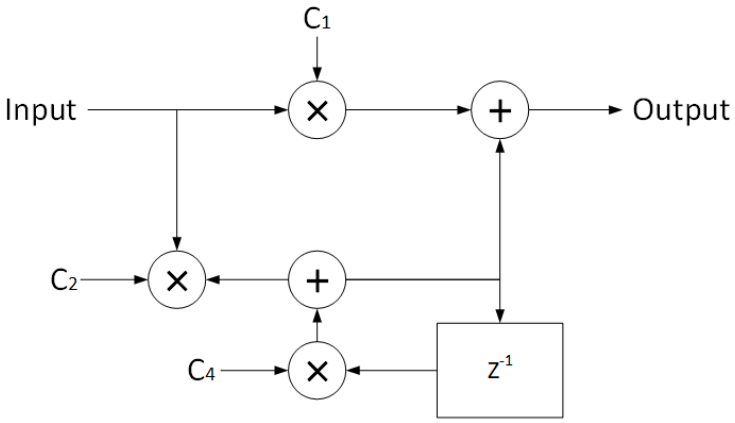

Figure 4.12: Imperfect integrator implementation in Z-domain

$$F(z) = C_1 + \frac{C_2}{1 - C_3 z^{-1}} \tag{4.16}$$

Alternatively, the transfer function can be expressed as:

$$F(z) = \frac{\tau_3(C_1 + C_2 - C_3) + \tau_3(C_1 + C_2)(z-1)}{\tau_3(1 - C_3) + \tau_3(z-1)} \tag{4.17}$$

In order to explain the transfer function in the Z-domain through the Laplace domain, applying a transformation already shown in Equation4.12. Thus, it is possible to express the the damping and the natural frequency as function of the gains in the Z-domain:

$$C_1 = \frac{-\epsilon\omega_n^2 T_s^2 + 4\zeta(2-\epsilon)\zeta\omega_n T_s - 4\epsilon}{(1-\epsilon)\omega_n^2 T_s^2 + 4(1-\epsilon)\zeta\omega_n T_s + 4(1-\epsilon)} \tag{4.18}$$

$$C_2 = \frac{(\epsilon^2 - 4\epsilon + 4)\omega_n^2 T_s^2 + 4\zeta(\epsilon^2 - 2\epsilon)\omega_n T_s + 4\epsilon}{(1-\epsilon)\omega_n^2 T_s^2 + 4(1-\epsilon)\zeta\omega_n T_s + 4(1-\epsilon)} \tag{4.19}$$

$$C_3 = 1 - \epsilon \tag{4.20}$$

where the following equality $T_s/\tau_3 = \epsilon$ can be used. Indeed, ϵ need to be a precise value, through it is possible to define the best-lock frequency offset versus time since loss of lock. Its value should be $[3.750 \cdot 10^{-9};\ 1.333 \cdot 10^{-8}]$, and its precision should to be $\pm 1\%$ in order to get almost the same precision in B_L[45].

Third order loop The Equation 4.25 shows the transfer function of a second order PI filter. Whit a third order PLL the parameters have to be chosen taking in account the stability of the system. Indeed, the open-loop transfer function, shown in the Equation 4.22, must be evaluated. It is possible to define the transition frequency $\omega_T = 2\pi f_T$ as the point on the magnitude bode diagram of $G(s)$ when the magnitude curve crosses the 0 dB. Since the phase of $G(\omega_T)$ have to be more positive than -180, with a phase margin $\phi(\omega_T)$ between about 30 and 60, it is possible to design the filter in order to have the zero of $G(s)$ coincident to ω_T. Thus, the phase margin is approximately 45.

$$F(s) = K\frac{1+\tau_2 s}{s(1+\tau_1 s)} \tag{4.21}$$

$$G(s) = \frac{\alpha F(s)}{s} \tag{4.22}$$

In a third order system, the natural frequency is not strictly defined, so the ω_{3dB} (that can be considered equal to B_L) should be used. The relation between the transition frequency and the $-3dB$ bandwidth is shown in the Equation 4.23 and can be considered fixed. For a critical damper system ($\zeta = 0.707$) $\omega_{3dB} \approx 2.06\omega_n$ and $\omega_T \approx 1.55\omega_n$.

$$\omega_T \approx \frac{\omega_{3dB}}{1.33} \tag{4.23}$$

Thus, in order to respect the stability constrains, the time constant of the PI filter should to be set as following:

$$K = \frac{\omega_T^2}{K_0 K_d} = \frac{1.33\omega_n^2}{K_0 K_d}, \quad \tau_2 = \frac{1}{\omega_T} = \frac{1.33}{\omega_n}, \quad \tau_1 = \frac{1}{5\omega_T} = \frac{0.266}{\omega_n} \tag{4.24}$$

where the time constant τ_1 is set in order to have an open-gain loop equal to 1 and τ_3 is set 5 times greater that τ_2 in order to have a good loop stability[46].

The Equation 4.25 shows the transfer function of a third order loop PLL.

$$H(s) = \frac{(m+2)\zeta\omega_n s^2 + (1+2m\zeta^2)\omega_n^2 s + m\zeta\omega_n^3}{s^3 + (m+2)\zeta\omega_n s^2 + (1+2m\zeta^2)\omega_n^2 s + m\zeta\omega_n^3} \tag{4.25}$$

An digital version of the second order PI filter is shown in the Figure 4.13. Whose transfer function in the Z-domain is expressed in the Equation 4.26.

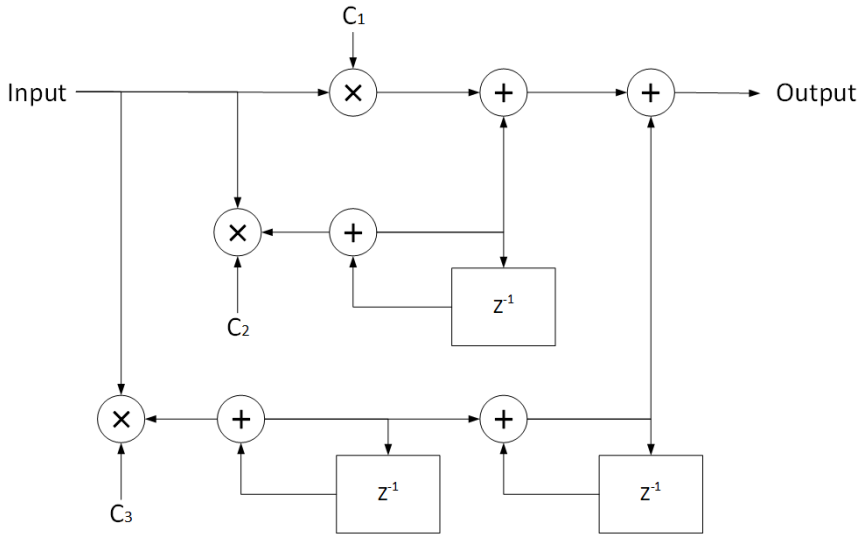

Figure 4.13: Second order filter implementation in Z-domain

$$F(z) = C_1 + \frac{C_2}{1 - z^{-1}} + \frac{C_3}{(1 - z^{-1})^2} \tag{4.26}$$

In oder to explain the transfer function in the Z-domain through the Laplace domain, applying a transformation 4.12. Thus, it is possible to express the the damping and the natural frequency as function of the gains in the Z-domain:

$$C_1 = \frac{8(1 + 2m\zeta^2)\omega_n T_s + 2m\zeta\omega_n^3 T_s^3}{8 + 4(1 + 2m\zeta^2)\omega_n T_s + 2(m + 2)\zeta\omega_n^2 T_s^2 + m\zeta\omega_n^3 T_s^3} \tag{4.27}$$

$$C_2 = \frac{8(m + 2)\zeta\omega_n^2 T_s^2 - 4m\zeta\omega_n^3 T_s^3}{8 + 4(1 + 2m\zeta^2)\omega_n T_s + 2(m + 2)\zeta\omega_n^2 T_s^2 + m\zeta\omega_n^3 T_s^3} \tag{4.28}$$

$$C_3 = \frac{8m\zeta\omega_n^3 T_s^3}{8 + 4(1 + 2m\zeta^2)\omega_n T_s + 2(m + 2)\zeta\omega_n^2 T_s^2 + m\zeta\omega_n^3 T_s^3} \tag{4.29}$$

4.8.2 GNU Radio implementation

Several PLL blocks are present in GNU Radio:

- PLL Carrier Tracking: a PLL mixing the input signal with the recovered carrier to output a baseband signal;
- PLL Ref Out: a PLL recovering and outputting the carrier from the incoming signal;
- PLL Freq Det: a PLL outputting the frequency of the incoming carrier.

Unfortunately, neither of the 3 available blocks fulfil the required functions described above, requiring a dedicated PLL block to be designed with the combined output. This can easily be done since GNU Radio already has the required functions implemented in the C++ code, but this is not accessible from the flow graph.
The PLL employs a complex mixer to down-convert the input signal to baseband, using a reconstructed carrier generated with an NCO. The resolution of such an NCO is fundamental for the overall frequency accuracy of the transponder, requiring at least 38-bit precision in the phase accumulator. In order to simulate properly a real attitude, it is inserted an integer accumulator up to 64 bits. Indeed, the input signal in the accumulator, is truncated and treated as integer number. In this way, the minimum angle step is fixed and without errors due to wrapping. Obviously, the NCO have to consider this mathematical transformation.
Actually, the phase demodulator is obtained exploiting the complex mixer (useful to shift the input signal in baseband) and converting its output with an inverse tangent operator, in order to calculate the phase error between the input signal and the reference oscillator. The phase demodulator output is made available as output of the PLL. This because, it is useful for the phase demodulation and for the lock signal generation. Indeed, the lock signal have to be evaluated outside this block, in order to allow more design freedom. After the inverse tangent operator, a low-pass filter is used to smooth eventual peaks.
The loop filters coefficients are externally selectable (also the order is selectable by setting the last coefficient to zero) to allow changing the PLL operational mode. To improve performances, also a leaky integrator has been added but only used for the second order PLL. The reason for this choice is the complexity of the calculations needed to obtain the transfer function from the gains when the loop filter is of third order. Since the leaky integrator improves performances in acquisition mode and the PLL is expected to be operated in acquisition mode with a second order loop filter, the choice was natural. Furthermore, the leakage has minimal influence on the loop transfer function if the filter is of second order[45].
The phase error output can be used directly as it provides the demodulated phase of the incoming signal: this output will be used in the project while the baseband complex output will not be used. The phase error can also be used to drive an external lock detector.
the PLL support tag.

Relevant parameters changeable at runtime:

- Loop filter order: it is the Loop filter order;
- Loop filter coefficients: they are the coefficients of the internal gains of the Loop filter;
- Frequency: central: it is the central frequency of the PLL when it is reset, so it is initial value of frequency, and the central frequency in the lockable range of frequency;
- Bandwidth: it sets the range of lockable frequencies.

Relevant parameters changeable at simulation start-up:

- N: it is the number of bits used for the accumulator. It sets the accuracy of the PLL;
- Sample rate: it is the sampling frequency expressed in samples per second.

4.9 Signal Search

The signal acquisition block is required to be able to acquire an incoming TC signal outside of the nominal acquisition bandwidth of the carrier tracking PLL. Usually PLLs have a very narrow band to limit noise coupling into the system and this limits heavily the acquisition range.
Two implementations are provided, a simpler one able to detect the presence of a signal within a certain band (the PLL acquisition band) and a more complex one, able to detect the presence of a signal is a wider band, also detecting the signal frequency.
Both implementations of the block will not operate on the signal constantly but only a limited amount of times per second (user selectable) to limit the computational load and ensure a timely signal detection.
Such a block would be used to steer the center frequency of the carrier recovery block. Once the Signal Search block locked onto the incoming signal, this quick acquisition block is not needed anymore. Due to the high signal to noise ratio expected for successful communication (approximately 10 dB, user selectable).

4.9.1 Carrier acquisition

Usual transponders operate with a very narrow carrier recovery loop bandwidth, leading toward a narrow pull-in range of the PLL. This problem leads to the need to precisely track the expected central frequency from ground otherwise the transponder might not operate at all. This turns impractical in certain applications, leading to the need for a wide frequency acquisition range[48]. This can be implemented in several ways, the simplest one of them could be by simply increasing the PLL loop bandwidth. This would lead in general to a degradation of tracking performances to widen the acquisition range. Three different approaches to solve this issue[48]:

- Steering the PLL lock frequency digitally from the OBC, using knowledge of the possible frequency the signal would be expected to use;
- Implement a search algorithm scanning a wide band in several small steps;
- Implementing a signal acquisition routine based on a wideband FFT or equivalent algorithm.

The first two approaches suggested would be possible with the current system being designed without no modification: it would only be needed to steer the receiver central frequency. But it would come at the penalty that the system (satellite maybe flying on an interplanetary trajectory) would require quite an extensive knowledge about its position with respect to Ground to perform a proper frequency prediction.
This approach would be the most flexible solution, even if it would entail also the highest complexity. Considering the high SNR required for the TC receiver to operate (approximately 10 dB), it would

be reasonably simple to scan a wide band around the receiver center frequency (even several hundreds kHz) to locate a narrow-band signal with a moderate to high SNR (probably more than 5 dB). The FFT should be sized such that every frequency bin covers slightly less than the locking range of the PLL, to guarantee a quick locking. Once such a signal is found in any of the FFT bin, the frequency acquisition block might control directly the carrier acquisition PLL to steer towards the incoming signal. This carrier tracking system should be enabled only when the PLL is currently not locked onto any signal and it should be disabled once the PLL locked. This way spurious locks and random walks could be avoided.

In a real environment, as explained hitherto, the high noise can make errors into the evaluation. In fact, can happen that, also with a proper average, the evaluation of input will provide a value under the threshold also after that the carrier is correctly locked. In a real implementation, this case can produce errors, or, in the worst case, to lose the lock status. To avoid that, the carrier acquisition must to be disabled after the lock. Since, it is used principally to speed up the locking of the carrier, and after that, it is not needed anymore, by saving computational power as well.

4.9.2 Pass-Band Filters

Ideally the Signal search block can be made by using only 3 band-pass filters, like shown in Figure 4.14. Indeed, by using a band-pass filter, and integrating its output by a simple IIR single pole filter, it is possible to get the magnitude squared in that band. Performing the same evaluation in the lateral bands, it is possible to take in account the amount of noise as well. Thus, if the power in the central band is greater of both the lateral bands, this means that a carrier is present in the central one.

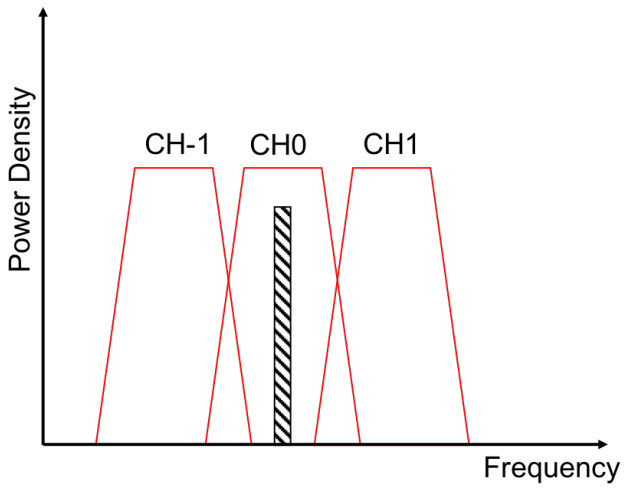

Figure 4.14: Simple signal search description

4.9.2.1 GNU Radio

Unfortunately, that filter should to be very sharp and, considering a sampling rate of 10 $Msps$, it should mean an huge amount of taps required for each filter. This algorithm so, would require thousands of taps for each filter that means malfunction of the whole block. Indeed, for many taps, numerical cancellation occurs for the calculation of the coefficients, which can also lead to instability.Even the complexity of the filter becomes heavy.

4.9.3 FFT

Thanks to fast Fourier transform (FFT) algorithm, it is possible to get all the frequency component of the whole spectrum of a signal. The FFT algorithm is an optimized version of the discrete Fourier

transform (DFT). The complexity of computing is $O(Nlog(N))$, instead of $O(N^2)$ of the DFT, where N is the FFT size. In fact, through the FFT (as for the DFT) is possible to get only N spectral components. The whole spectrum is defined by the sampling rate, and the FFT produces the spectrum of a signal from $-f_s/2$ to $f_s/2$. Thus, it is possible to see both positive and negative frequency, and it is possible to set the number of bins to see.

4.9.3.1 Design

The use of a FFT algorithm is useful to know if there is a carrier/signal into the wanted band, and in addition is ideally possible to know also the exact frequency of the carrier. Considering that the band to scan is of few kHz, the FFT size should to be large enough to have tens of bins in the wanted band, in order to distinguish its frequency and magnitude properly. The transponder designed is supposed to work with 10 $Msps$, so without a down-sampling the needed FFT size should has been too large. Thus, a down-sampling is needed.
However, the FFT size has to be a power of 2, and keeping under control the FFT size, an quite aggressive down-sampling can be needed. For example, if the sampling frequency is 10 $Msps$ and the wanted bandwidth is 2 kHz, considering only 20 bins in bandwidth, so a resolution of almost 100 Hz. Considering an FFT size of 1024, so a down-sampling of almost 96 is needed.
As described above, a Low-Pass polyphase filter is implemented. The transponder has to scan in order to find the carrier. Actually, the input signal is more complex with several possible sub-bands, so it is important to use an anti-aliasing filter. Fortunately, being interested only to the carrier, that is at least 16 kHz far from the first sub-band, the anti-aliasing filter can be no so sharp. Indeed, the number of taps for a FIR is strongly related to the transition width. Thus, it is possible to design the filter in order to have al least the passband up to $(16\ kHz) + f_{central} + Bw/2$, in order to be sure that the carrier is not filtered, and the stop band in correspondence of the frequency relative to the new sampling frequency (after downsampling). In this way, the transition width is the maximum possible.
At this point, by applying the FFT algorithm is necessary to evaluate the proper Carrier to Noise Ratio. The same algorithm of the SNR Estimator block has been used. In order to evaluate properly the CNR in the scanned band, it is important to set the FFT size in order to have tens of bins in that band, since 7 bins are used to evaluate the Carrier power, and at least 7 (it is a too small value) to evaluate the power noise density.
The algorithm for the CNR estimation start by searching the highest bin in the bandwidth. So, it sum up its value with the values of the 3 bins on the left and on the right (they are value of magnitude squared, not expressed in d) in order to consider that the spread is wider than the main lobes of the windows[49]. After that, all the values of bins into the band less the ones used for the power carrier evaluation, are sum up each others. Furthermore, in order to consider the noise under the carrier, the mean of the values of bins summed up for the noise has been evaluated. Thus, it is multiplied by 7 and summed up to the previous partial sum for the noise power density.
Finally, evaluating the estimated CNR as expressed in the Equation 4.30, if the CNR of the input signal is equal or greater to the set one, the input signal shall be outputted.

$$CNR_{dB} = 10log_{10}\left(\frac{\sum_{i=j-3}^{j+3} Power(j)}{\sum_{i=i_{-Bw}}^{j-4} Power(i) + \sum_{i=j+4}^{i_{bw}} Power(i) + 7 \cdot mean(Power(w))}\right) \tag{4.30}$$

where i_{-Bw} and i_{Bw} are the bins correspondent to lower and upper limits of the bandwidth and $w = [i_{-Bw}, i_{Bw}] \cap [j-3, j+3]$.

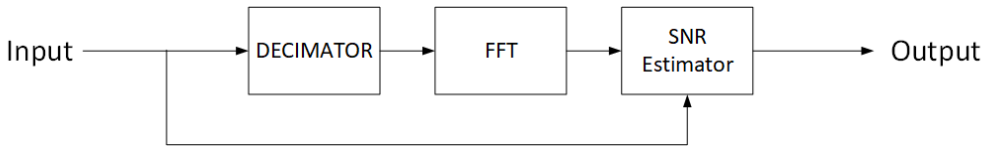

Figure 4.15: Signal Search block with FFT algorithm

4.9.3.2 GNU Radio implementation

This block presents several complex functions, like down-sampling, FFT, scanning through the SNR estimation algorithm and evaluation of results. In addition, it is fundamental to output the input data evaluated and without any decomposition. Due to these reasons, a new block has to be full implemented to guarantee the required data correctness.
The FFT algorithm needs in input a number of items equal to the FFT size chosen. This means that the scheduler shall allocate in input of block a number of items as the size of FFT. Furthermore, a decimation is needed, so, the number of items in input of the block is $decimation \cdot size$. Considering a sampling rate of 10 $Msps$, this number of items shall be greater than GNU Radio's scheduler limit. The limit of items for the scheduler is 8192, but, experimentally it has been noticed that, it can not exceed 4096 items (the tests were carried out on a laptop with an Intel i3 processor with a 64-bit architecture). On GNU Radio is easily possible to overtake this limit by using a vectorization of data, this mean to insert the n items into vectors of dimension n through "Stream to Vector" block. In this way the scheduler will consider the whole vector as on item.
Moreover, it is important to output exactly the input data if they satisfy the CNR constrain. Indeed, this algorithm works with large blocks of data, and to lose one, by outputting the next block if the actual one satisfy the constrains, it would mean to lose thousands of items and to have not guarantee that the next ones satisfy the constrain. Considering that an down-sampling is applied to the input data, the only way to be sure to outputting exactly the input data is to do all the processing inside the same block. In this way, it is possible to check a block of items, and if they satisfy the constrain, the GNU Radio block shall output directly them.
Thus, the output of Signal Search can have discontinuity, since the items are evaluated by blocks. To let the others blocks to know it, first of all the PLL, a reset Tag (completely supported by GNU Radio) has been added to the first items of each block that satisfy the constrain after at least one that had not satisfied that. In this way, is possible to have also a fixed delay, since each block needs the same amount of time to be evaluated. This features is extremely important in a transponder.
On the other hand, considering that blocks performs FFT and down-sampling are already available on GNU Radio, these should to be included into the designed one. Unfortunately, to allow that, a block should to be designed in order to have public functions that make the evaluation, different by the standard "work" or "general_work". The FFT block already available on GNU Radio allows that, and the only block with a polyphase re-sampling with that feature is "polyphase arbitrary resampler", so, it has been included. The its only drawback is that the an arbitrary resampler evaluates automatically the internal values of down-sampling and up-sampling in order to reach the set ratio (that can be fractional as well), and this is often not so effective, since the internal algorithm tries to use both, also when a normal down-sampling is requested. In addition, it was noticed that this block works better with an internal parallelism (number of filter banks) of 32. Those drawbacks are considered acceptable since only one simulation should be performed.
Finally, for the sake of simplicity, also an hierarchical block has been developed in order to handle the data vectorization. Indeed, since the blocks "Stream to Vector" and "Vector to Stream", with a length fixed by FFT size and decimation, are needed. Thus, the hierarchical block is made with the two latter and the signal search block, in order to have in input and output a normal complex type.

Relevant parameters changeable at runtime:

- Enable: it defines if the input evaluation is performed or not. If the block is not enable, the input items are directly outputted;

- Average: it defines if average (with IIR Filters) at the output of bins is applied or not,
- Frequency central: it is the centre of the bandwidth where signal is searched, it must be an integer multiple of the sampling frequency (sampling rate);
- Bandwidth: it is the value of bandwidth where the signal is searched;
- Frequency cut-off: it is the value of cut-off frequency of the internal IIR used to filter the output of the internal bandwidth;
- Threshold: it is the minimum difference that must to be between the central band and the lateral ones in order to discriminate if there is it.

Relevant parameters changeable at simulation start-up:

- Decimation: it is the decimation factor of the input signal;
- FFT size: it is the number of FFT bins;
- Sampling rate: it is the sampling rate value used, expressed in samples per second;
- Window type: it is the windowing method type used for FFT evaluation.

4.9.4 Goertzel

An interesting solution is to exploit the Goertzel algorithm, in order to perform a similar evaluation. Essentially, it is able to perform the DFT calculation for only one frequency component. It is made by two main steps, the one expressed in the Equation 4.31 that must to be iterated N times, and the second one in the Equation 4.32 that should to be performed once at the end.

$$y'[n] = x[n] + 2\cos(\omega_0)y'[n-1] - y'[n-2] \tag{4.31}$$

$$y[n] = y'[n] - e^{-j\omega_0}y'[n-1] \tag{4.32}$$

Goertzel block size N corresponds to the size of a DFT or FFT, and it defines the frequency resolution (also called bin width). The frequency resolution is given by the ratio of the sampling rate with the block size N, moreover this value should ideally be integer in order to have the frequencies centered in their respective bins (exactly as for the FFT or DFT). Furthermore, unlike the FFT the block size must not be a power of two[50].
This algorithm is particularly suitable to an hardware and software implementation. Indeed, the coefficient used can be pre-computed, only two registers and N iterations are needed.
This algorithm should to be applied separately to the real and imaginary part. For each of them, a complex number is gotten, thus, a complex sum has to be performed. Thus, it is now possible to perform the magnitude. The response of algorithm for a single bin is shown in Figure 4.16.

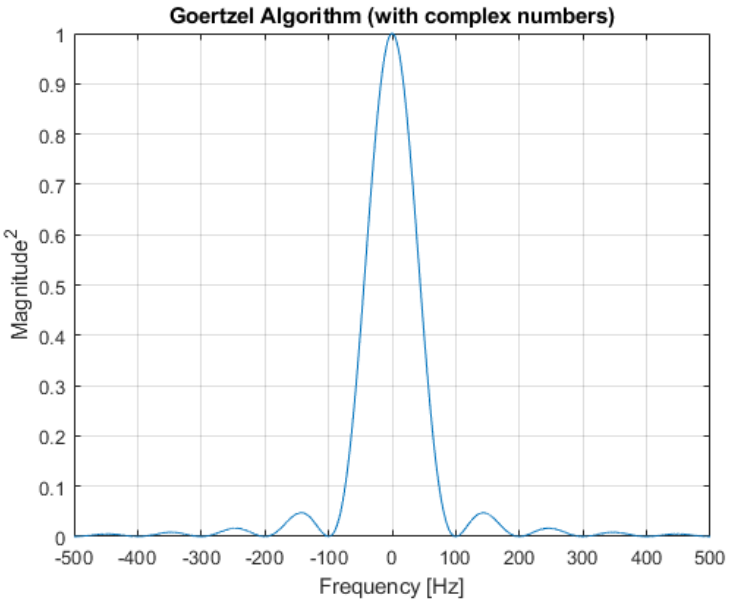

Figure 4.16: Goertzel algorithm response

4.9.4.1 Design

The key idea of this solution, it to design the Goertzel algorithm in order to have a resolution as $2Bw$, and to centre the bin with respect to the set central frequency. Additionally, perform the same algorithm with a two more bins centred respectively in $f_0 - Bw$ and $f_0 + Bw$. In this way, the two lateral bins will intersect the main one in $+Bw/2$ and $Bw/2$ in order to get the set bandwidth resolution. In Figure 4.17 is shown an example with $f_0 = 0\ Hz$.

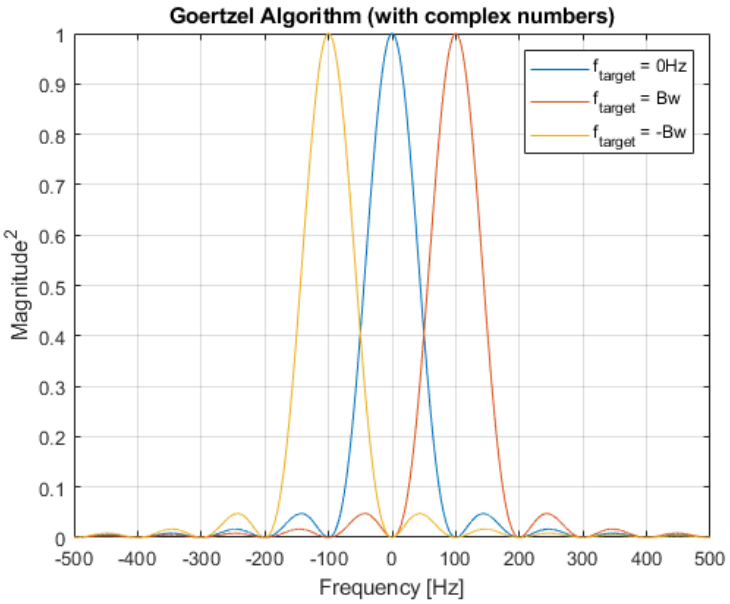

Figure 4.17: Goertzel algorithm for three different bins

Since the Goertzel algorithm should to be performed for target frequency that are integer multiple of sampling rate, so zeros of the outputs will be in the same points for all the three bins. This mean

to do a ratio between zero and zero, and mathematical errors occurs. Indeed, that solution will have fake positive for each integer multiple of the bandwidth.

The designed transponder has already an AGC in front of Signal search, thus the RMS value of the input can be considered constant and equal to 1. This can allow an important advantage in the design of he signal search. Indeed, it is already known that the ideal maximum amplitude of each bins is 1. According to that, it is possible to apply an truncation to the algorithm's response in order to avoid the presence of side-lobes and zeros. According to the Figure 4.17, the insertion point of the curves is around 0.405, thus, it is possible to design the algorithm in order that if the bin's output is under 0.405, the output will be truncated and set to a value equal to ϵ. Where ϵ is a value small enough to avoid the mathematical error zero to zero. The new algorithm is shown in the Figure 4.18.

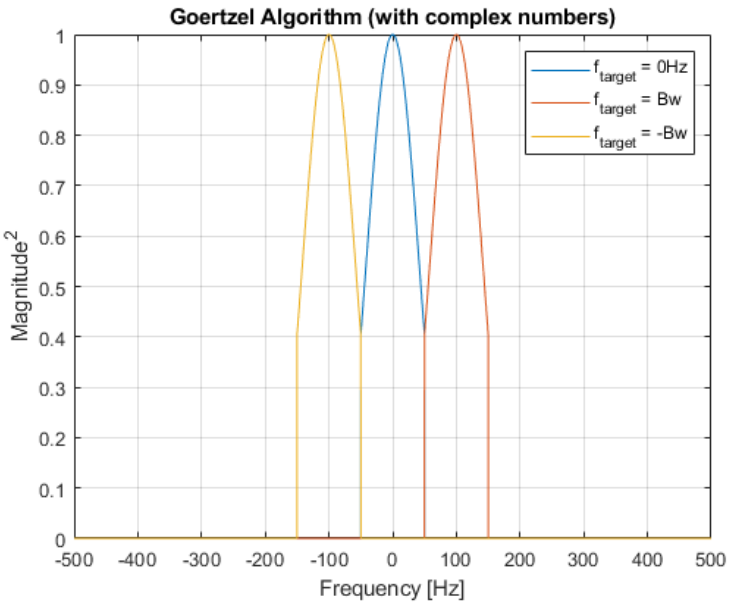

Figure 4.18: Truncated Goertzel algorithm for three different bins

In Figure 4.19 is shown the ratio of the main lobe with the lateral ones. Thus, if both the ratio are greater than the set value, the input value shall be outputted.

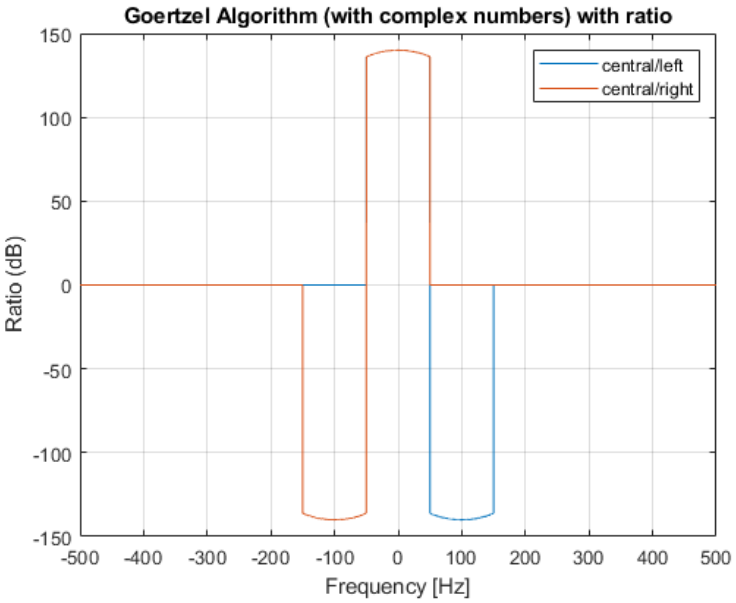

Figure 4.19: Ration between bins of truncated Goertzel algorithm

Finally, the detection probability of the algorithm in noise conditions was calculated.

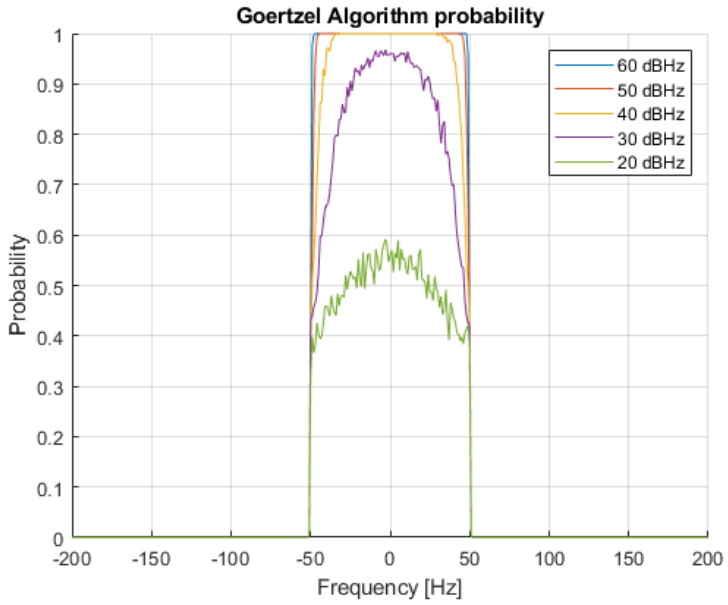

Figure 4.20: Probability of correct detection of the input signal

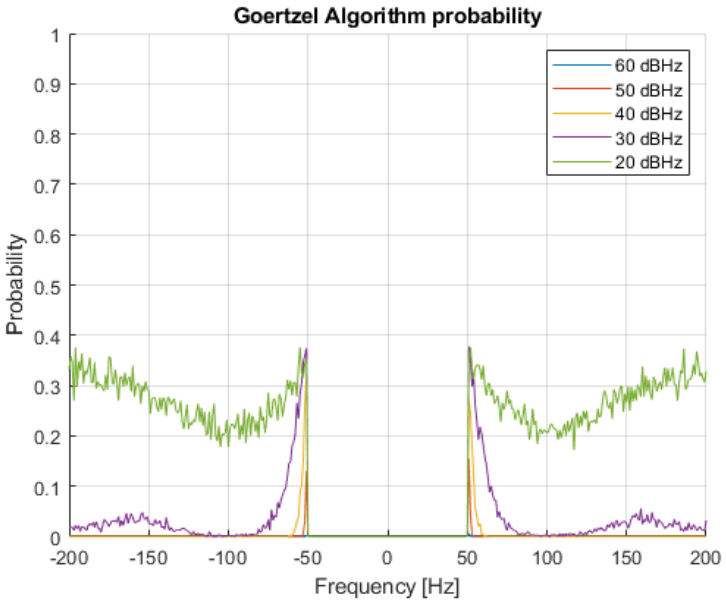

Figure 4.21: Probability of false detection of the input signal

Therefore, from the simulations, the algorithm designed is not able to work correctly with a signal having an SNR of less than $\sim 40\ dBHz$.

4.9.4.2 GNU Radio implementation

A block that evaluates the Goertzel algorithm is already present on GNU Radio, and is called Goertzel Block. Unfortunately it does not perform the optimized algorithm previously presented, so a new block has been implemented.
This block applies the Goertzel algorithm on a number of items equal to the size of the algorithm. As previously mentioned, the size is automatically calculated from the ratio of the sampling rate with the set bandwidth. These items are internally saved and if the result of the algorithm on these items is over the set threshold they are outputted, otherwise they are eliminated. Therefore, the algorithm is applied on a number of samples equal to the size at a time.
Finally, the output of Signal Search can have discontinuity, since the items are evaluated by blocks. To let the others blocks to know it, first of all the PLL, a reset Tag (completely supported by GNU Radio) has been added to the first items of each block that satisfy the constrain after at least one that had not satisfied that. In this way, is possible to have also a fixed delay, since each block needs the same amount of time to be evaluated. This features is extremely important in a transponder.

Relevant parameters changeable at runtime:

- Enable: it defines if the input evaluation is performed or not. If the block is not enable, the input items are directly outputted;
- Average: it defines if average (with IIR Filters) at the output of bins is applied or not;
- Frequency central: it is the centre of the bandwidth where signal is searched, it must be an integer multiple of the sampling frequency (sampling rate);
- Bandwidth: it is the value of bandwidth where the signal is searched;
- Frequency cut-off: it is the value of cut-off frequency of the internal IIR used to filter the output of the internal bandwidth;

- Threshold: it is the minimum difference that must to be between the central band and the lateral ones in order to discriminate if there is

Relevant parameters changeable at simulation start-up:

- Sampling rate: expressed in samples per second.

Chapter 5

Extra blocks

This section describes few extra blocks that will be implemented to help with the simulation. They are not strictly related to a particular sub-part (transponder, test setup, etc.) but can be used to simulate non-ideal behaviours or to use different numeric types. They can be inserted in the transponder to simulate particular effects, like the finite precision in case of a limited number of bits in the processing path. GNU Radio is designed to simplify the design of software-defined radios by providing pre-designed blocks that show an almost ideal behaviour. This simplifies the design of an algorithm by removing the secondary effects but it makes it complex to simulate a real system (with non-ideal effects as in real life). In order to make the system closer to a real implementation, non-ideal behaviour is added.

5.1 Debug Function Probe

Debug Function Probe is a block specifically created to deal with a problem encountered during the execution of the QA Tests. In fact, as happens during the execution of a flow graph on the GNU radio, it is necessary to use a specific (and specifically created) callback set function to update a parameter while the block is running. To do this automatically, the Function Probe block must to be used. The latter is not a real block, but it is a sort of script that writes a specific function within the flow graph. This function in turn creates a thread, which with the use of timing functions (as wait function) goes to call a specific callback of a specific block at regular intervals. Therefore, this is the only way to change the parameters of a block in execution.

Unfortunately, it is not possible to know exactly when this parameter has actually been set in a qa test. In fact, it is only possible at the end of the simulation to check that the parameter actually has the new value, but it is not possible to check when. Moreover, it has to been considered that GNU Radio allows callbacks to be called only before or after the work function (or general_work), not in the meantime. Therefore, even when planning an appropriate test, one can never be certain that the parameter will be set at a certain moment because it depends on the synchronization of the various threads, which can not be set in a precise way.

5.1.1 GNU Radio implementation

This block has a simple callback function. Recalling it, the absolute offset of the item to which the function has been called is internally saved. In fact, by connecting the block in the flow graph in order to have exactly the same input as the block to be tested, the items that will process the Debug Function Probe block will be the same as the block to be tested. Furthermore, by calling the Debug Function Probe block callback immediately after (or immediately before) the block to be tested, ideally, they will get the call to the set function in the same break between the various blocks

of items. Therefore, through the absolute offset of the last processed item, it is possible to know at what moment of the situation the call to the function was received for both blocks.
So, at the end of the simulation, through the call of a second function, it will be possible to obtain a vector with the history in which the various spare parts to the functions have been registered.

Obviously, it can happen that the two functions can be recalled in two different breaks. Therefore, the accuracy of the measurement will depend on the scheduler and the size of the blocks of items that each block must process at a time. The latter can not be predicted on GNU Radio (only its maximum value is known, or 4096 items per block), but during a QA Test can be forced. Therefore, it is possible to know the accuracy of the measurement.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- vlen: vector length of data streams;
- Type: input type.

5.2 Debug Sine

The sine debug block is a block specifically designed to identify any phase jumps in a sinusoidal signal. In reality the latter simply makes the derivation of the phase of the input signal, and in the case of the sinusoid it is equivalent at the phase step. This value must be constant.

5.2.1 GNU Radio implementation

A very simple approach to solving the problem is to calculate the input signal argument and then make the derivative simply by making the difference between the value in an instant and its value in the previous instant.

Unfortunately, this approach requires checks when the signal phase completes one complete revolution. A more robust evaluation method, which is also the used one, is as follows:

$$\Phi_{step} = arctg\left(\frac{\mathbb{I}m\left(X[i]\cdot\overline{X[i-1]}\right)}{\mathbb{R}e\left(X[i]\cdot\overline{X[i-1]}\right)}\right) \tag{5.1}$$

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- None.

5.3 Fixed-point math emulator

Fixed-point mathematics is very common in hardware implementations of digital signal processing chains. GNU Radio also supports integer (and complex integer) types but unfortunately most of the blocks are only implemented in float (and complex) arithmetic. To avoid re-implementing all the existing blocks with integer math, a rounding block can be used to simulate this effect: the precision

of the floating-point number shall be reduced in case the rounding is performed with a number of bits lower than a floating-point number (mantissa of 23 bits). This block shall process the integer and fractional part of the input value and round them to a maximum number of bits. The maximum number of bits is 64, including both the integer and fractional part. Saturation will be implemented in a selectable way, considering saturation and wrap-around.

5.3.1 GNU Radio implementation

This block is implemented in C++ to have all the necessary freedom in implementation. The implementation for the integer part will be based on a simple rounding (floor rounding) and the fractional part will be obtained by multiplication of the fractional part for the amplitude of the least significant bit, rounding and division. Thus, the maximum values that can be represented are:

$$Max(X) = 2^{N_{integer}-1} - 2^{N_{fractional}}; \quad Min(X) = -2^{N_{integer}-1} \tag{5.2}$$

Finally, different types are supported, such as complex, double and float.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- vlen: vector length of data streams;
- type: data type of data streams;
- Nint: number of bits for the integer part;
- Nfrac: number of bits for the fractional part.

5.4 Integer math emulator

Integer mathematics is an hardware implementations of digital signal processing chains. GNU Radio also supports integer (and complex integer) types but unfortunately can not be set the number of bits. This block shall process the integer of the input value and round it to a maximum number of bits. The maximum number of bits is 64 for long and 32 for int types.

5.4.1 GNU Radio implementation

This block is implemented in C++ to have all the necessary freedom in implementation. The implementation for the integer part will be based on a simple saturation of minimum and maximum values. Thus, the maximum values that can be represented are:

$$Max(X) = 2^{N_{integer}-1} - 1; \quad Min(X) = -2^{N_{integer}-1} \tag{5.3}$$

Finally, different types are supported, such as int and Int64.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- vlen: vector length of data streams;
- type: data type of data streams;
- Nint: number of bits for the integer part.

5.5 IQ impairment correction

IQ impairment is the simplest impairment and it relates to a non-perfect phase shift of 90° and balanced amplitude between the I and Q channel. Offsets in the two channels can also introduce a non-ideal behaviour. Considering a zero-IF architecture, as in our case, the IQ impairment can translate into limited local-oscillator suppression or limited side-band suppression, introducing unwanted signals in-band. The generic formula to implement an amplitude imbalance, a phase imbalance and an offset on both I and Q is shown here:

$$\begin{bmatrix} I_{out} \\ Q_{out} \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I_{in} \\ Q_{in} \end{bmatrix} + \begin{bmatrix} I_{off} \\ Q_{off} \end{bmatrix} \tag{5.4}$$

The A and D terms are used to impose an amplitude imbalance when not identical, while the terms B and C mix I and Q generating a phase error. The matrix can be rewritten as:

$$\begin{bmatrix} \sqrt{K_D} cos\left(\frac{X_D}{2}\right) & \sqrt{K_D} sin\left(\frac{X_D}{2}\right) \\ \frac{1}{\sqrt{K_D}} sin\left(\frac{X_D}{2}\right) & \frac{1}{\sqrt{K_D}} cos\left(\frac{X_D}{2}\right) \end{bmatrix} \tag{5.5}$$

With kD and Xd being the gains imbalance between and the phase difference between the I and Q channels[51]. This formula divides the introduced phase shift and amplitude imbalance in equal parts and applies it to both channels not to introduce a phase offset or an amplitude mismatch that might alter the transponder behaviour.

5.5.1 GNU Radio implementation

This block implements the single branch IQ imbalance transmitter and receiver models. According to "In-Phase and Quadrature Imbalance: Modeling, Estimation, and Compensation" [52], the following model are developed.
For the TX impairment:

$$Y = \left[Re(X)10^{\frac{mag}{20}} + cos\left(\frac{\pi \cdot phase}{180}\right)\right] + j\left[Re(X)10^{\frac{mag}{20}} + sin\left(\frac{\pi \cdot phase}{180}\right) + Im(X)\right] \tag{5.6}$$

For the RX impairment:

$$Y = Re(X)10^{\frac{mag}{20}}\left[sin\left(\frac{\pi \cdot phase}{180}\right) + cos\left(\frac{\pi \cdot phase}{180}\right)\right] + j\left[Im(X)\right] \tag{5.7}$$

Relevant parameters changeable at runtime:

- Magnitude: magnitude correction;
- Phase: phase correction.

Relevant parameters changeable at simulation start-up:

- mode: TX or RX.

5.6 Phase Converter

This block converts the phase input in a Int64 normalized output. Furthermore, this block allows to reduce the accuracy (setting the number of bits N) of the mathematics in order to simulate properly a real behaviour. Finally, this block is designed for work together with the coherent phase modulator OOT of the ecss module.

5.6.1 GNU Radio implementation

Pay attention that the input is considered as radiant (suggested being in the range $[-\pi; \pi]$, but an wrapping function is done internally). The output is the input value, normalized by pi, rounded and multiplied by 2^{64-N} in order to be suitable as input for an "integer accumulator".

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- N: this is the number of bits used for the fixed-point representation.

5.7 Selector

The selector block is required to select one out of several connections to enable/disable certain processing blocks. The selector can be used am multiplexer and demultiplexer, to select between multiple inputs for an output or to select multiple outputs for an input. In addition, by setting only one input and one input, it can be a switcher.

5.7.1 GNU Radio implementation

This block is already existing in GNU Radio but it is currently (as of version 3.7.8) marked as deprecated. The block relies on a connection being established/removed at runtime between two blocks and this mechanism will not be supported in the future. The block has been re-implemented using a memory copy operation that is driven by the input variable that selects the branches to connect. Thus, the number of input and output are settable, but the number of inputs or the number of outputs, or both, must be equal to one. In addition, the select can be also set as '-1', in this condition the block becomes a switcher. Finally, the tags can pass through only the selected path, the others tags on the others ports will be consumed and deleted.

Relevant parameters changeable at runtime:

- Select: used to select which port is operational (input or output).

Relevant parameters changeable at simulation start-up:

- vlen: vector length of data streams;
- type: input/output type;
- Num Input: Number of input ports;
- Num Output: Number of output ports;

5.8 Signal to Noise Ratio Estimator

This block uses exactly the same algorithm of the signal search with FFT expounded into the Section 4.9.3 to output an Carrier to Noise estimation of the input signal. In fact, it can evaluate the input signal in different way. Indeed, it is possible to get an estimation of the CNR with a self-detection of the carrier. Additionally, it is also possible to set the frequency to consider as carrier. As for the Signal Search Block, it possible to set the bandwidth to consider to the noise evaluation. In that case, it considered the bandwidth around the carrier found (or set). Furthermore, it is also possible to set to evaluate the noise for the whole spectrum. It is also possible to get the Signal to Noise estimation.

In the latter, it has to be set the signal bandwidth of the signal. As for the noise bandwidth, it will be considered around the carrier found (or set).
Finally, it is possible to set the window type for the FFT evaluation, the FFT size and the average.

5.8.1 GNU Radio implementation

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- Auto carrier: if the carrier has to be auto detected or used the set one;
- SNR/CNR: if the CNR has to be evaluated or the SNR;
- All spectrum: if the whole spectrum has to be consideres as noise bandwidth or only the set one;
- Average: it defines if average (with IIR Filters) at the output of bins is applied or not;
- Frequency central: it is the centre of the bandwidth where signal is searched if the auto carrier function is not set;
- Signal Bandwidth: it is the value of bandwidth of the signal used for the SNR evaluation;
- Noise Bandwidth: it is the value of bandwidth of the noise used for the SNR or the CNR evaluation if the all spectrum is not set;
- Frequency cut-off: it is the value of cut-off frequency of the internal IIR used to filter the output of the internal bandwidth;
- Threshold: it is the minimum difference that must to be between the central band and the lateral ones in order to discriminate if there is it;
- FFT size: it is the number of FFT bins;
- Sampling rate: it is the sampling rate value used, expressed in samples per second;
- Window type: it is the windowing method type used for FFT evaluation;
- FFT output: if the FFT performed has to be outputted or not.

5.9 Sinks and Sources

5.9.1 Vector Sink

This block was created because the current version of GNU Radio did not support double and Int64 types. So both in the qa test and for use on the GNU Radio interface are necessary to manage the blocks with an output or input of these two types.

5.9.1.1 GNU Radio implementation

This block is the GNU Radio one upgraded in order to support also the double and Int64 types.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- vlen: vector length of data streams;
- reserve items: reserve space in the internal storage.

5.9.2 Null Sink

This block was created because the current version of GNU Radio did not support double and Int64 types. So both in the qa test and for use on the GNU Radio interface are necessary to manage the blocks with an output or input of these two types.

5.9.2.1 GNU Radio implementation

This block is the GNU Radio one upgraded in order to support also the double and Int64 types.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- vlen: vector length of data streams;
- type: data type of data streams.

5.9.3 Vector Source

This block was created because the current version of GNU Radio did not support double and Int64 types. So both in the qa test and for use on the GNU Radio interface are necessary to manage the blocks with an output or input of these two types.

5.9.3.1 GNU Radio implementation

This block is the GNU Radio one upgraded in order to support also the double and Int64 types.

Relevant parameters changeable at runtime:

- Repeat: if the data set shall be repeated or not;
- Data: data set to output.

Relevant parameters changeable at simulation start-up:

- vlen: vector length of data streams;
- tags: tag set to output.

5.9.4 Null Source

This block was created because the current version of GNU Radio did not support double and Int64 types. So both in the qa test and for use on the GNU Radio interface are necessary to manage the blocks with an output or input of these two types.

5.9.4.1 GNU Radio implementation

This block is the GNU Radio one upgraded in order to support also the double and Int64 types.

Relevant parameters changeable at runtime:

- vlen: vector length of data streams;
- type: data type of data streams.

Relevant parameters changeable at simulation start-up:

- None.

5.10 Type Casting

5.10.1 Float to Double

This block was created because the current version of GNU Radio did not support double and Int64 types. So both in the qa test and for use on the GNU Radio interface are necessary to manage the blocks with an output or input of these two types.

5.10.1.1 GNU Radio implementation

This block is the GNU Radio one upgraded in order to support also the double type. It performs a simple casting.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- None.

5.10.2 Float to Int64

This block was created because the current version of GNU Radio did not support double and Int64 types. So both in the qa test and for use on the GNU Radio interface are necessary to manage the blocks with an output or input of these two types.

5.10.2.1 GNU Radio implementation

This block is the GNU Radio one upgraded in order to support also the Int64 type. It performs a simple casting.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- None.

5.10.3 Int to Int64

This block was created because the current version of GNU Radio did not support double and Int64 types. So both in the qa test and for use on the GNU Radio interface are necessary to manage the blocks with an output or input of these two types.

5.10.3.1 GNU Radio implementation

This block is the GNU Radio one upgraded in order to support also the Int64 type. It performs a simple casting.

Relevant parameters changeable at runtime:

- None.

Relevant parameters changeable at simulation start-up:

- None.

Chapter 6

QA Tests

The following are the Quality Assurance (QA) tests created ad hoc for each GNU Radio block implemented. QA tests are test files supported by GNU Radio that are based on the Python unit testing framework[53]. They are generally written in Python and allow to be launched at the time of the Out-Of-Tree (OOT)[54] installation in order to certify the correct functioning of each single block. In order to guarantee the maximum traceability of the tests for ESA, it has been decided to modify, through the creation of a new library, the generation of reports files. It has been decided to automatically generate for each block an HTML file that contains not only the information whether the test was positive or negative, but also all the parameters set for each block test, the results, the moment in which the test was generated and, above all, the check sum of each file involved in the test. For the latter, an MD5 encoding was used that uniquely defined each file, including the headers for C ++, in order to certify the test in digital format. Finally, at the end of each test, two PDF files are generated, one containing each of the performed tests, and the other containing the set of graphs generated for each test.

6.1 Automatic Gain Control

As explained in Section 4.2, the AGC is required to have a constant settling time between 10 ms and 30 ms. In order to test if the designed AGC can satisfy that specification, the minimum settling time is set for all the cases. Additionally, according to the designed transponder, the AGC is needed in order to guarantee an output value of one. Thus, all the cases evaluated will have the reference value set to one. Finally, the initial gain is not relevant in the tests, it should to be set only if the ratio between output and input is already known. Thus, is set to one, since is a neutral value.

For each test case, it is checked if the settling time is equal or greater that 10 ms and if it is equal or lower than 30 ms. In addition, it is checked if the value of output before the step is stable, so if the reference value has been reached. Finally, the mean error percentage of the output before and after the step have been checked. The latter, must be not greater than a defined value.

To evaluate all the characteristic of the transient, a proper function is used. It starts from the end of the transient, and evaluates the mean output error from the end until the maximum allowed settling time. In addition, it check last value of transient is into the expected error range. Furthermore, it consider as the last value outside the expected error range (considered of 5%) as the moment to consider for the settling time evaluation. The start of the step is already known, and is exactly in the middle of simulation. That is done in order to guarantee enough time to the AGC to reach the stable output before the step. Finally, as for the error evaluation after the step, the error evaluation before the start is evaluated between the start of the step and after 30 ms of the start of simulation.

qa_pll

Path header file: /home/isispace/gnuradio/gr-ecss/include/ecss/pll.h

CheckSum header file: 8a4a2d45b3d0a2067d9076dae3d5b53f

Path second header file: /home/isispace/gnuradio/gr-ecss/lib/pll_impl.h

CheckSum second header file: 9c55b42068fa905105bd2d0a901f6329

Path C++ file: /home/isispace/gnuradio/gr-ecss/lib/pll_impl.cc

CheckSum C++ file: 934676fc3ad8a29dfc39a5aeaaeda71a

Path test file: /home/isispace/gnuradio/gr-ecss/python/qa_pll.py

CheckSum test file: 091443032e68652092c29171afabedc8

Start Time: 2018-11-26 14:08:25

Duration: 0:02:39

Status: Pass: 11

Test name	Parameters	Stack	Status
test_001_t: Input Sine in central bandwidth	Order = 2; Coeff1 (2nd order) = 0.052267; Coeff2 (2nd order) = 0.00143173; Coeff4 (2nd order) = 1.000000; Coeff1 (3rd order) = 0.022742; Coeff2 (3rd order) = 7.18567e-5; Coeff3 (3rd order) = 1.97104e-7; Frequency central = 500.00 Hz; Bandwidth = 500.00 Hz; Sample rate = 16384 Hz; Input frequency = 600 Hz; Input noise = 0.00 V	-Output 'Out' Settling time : 0.732422 ms; -Output 'Out' Real absolute maximum error: 0.000; -Output 'Out' Imag absolute maximum error: 0.001; -Output 'pe' Settling time : 0.732422 ms; -Output 'pe' absolute maximum error: 0.001 rad; -Output 'freq' Settling time : 0.549316 ms; -Output 'freq' absolute maximum error: 0.123 Hz; -Output Slope : 3769.986613 rad/s; -Output Min step : 0.177867 rad.	Pass
test_002_t: Input Sine in the boundary bandwidth	Order = 2; Coeff1 (2nd order) = 0.052267; Coeff2 (2nd order) = 0.00143173; Coeff4 (2nd order) = 1.000000; Coeff1 (3rd order) = 0.022742; Coeff2 (3rd order) = 7.18567e-5; Coeff3 (3rd order) = 0.000000; Frequency central = 500.00 Hz; Bandwidth = 500.00 Hz; Sample rate = 16384 Hz; Input frequency = 749 Hz; Input noise = 0.00 V	-Output 'Out' Settling time : 0.671387 ms; -Output 'Out' Real absolute maximum error: 0.000; -Output 'Out' Imag absolute maximum error: 0.000; -Output 'pe' Settling time : 0.671387 ms; -Output 'pe' absolute maximum error: 0.000 rad; -Output 'freq' Settling time : 0.549316 ms; -Output 'freq' absolute maximum error: 0.001 Hz; -Output Slope : 4706.103030 rad/s; -Output Min step : 0.176081 rad.	Pass

Figure 6.1: Example of QA test report

6.1.1 Step of 60 dB

This test case is used to prove that the constant settling time requirement is satisfied with a step of 60 dB.

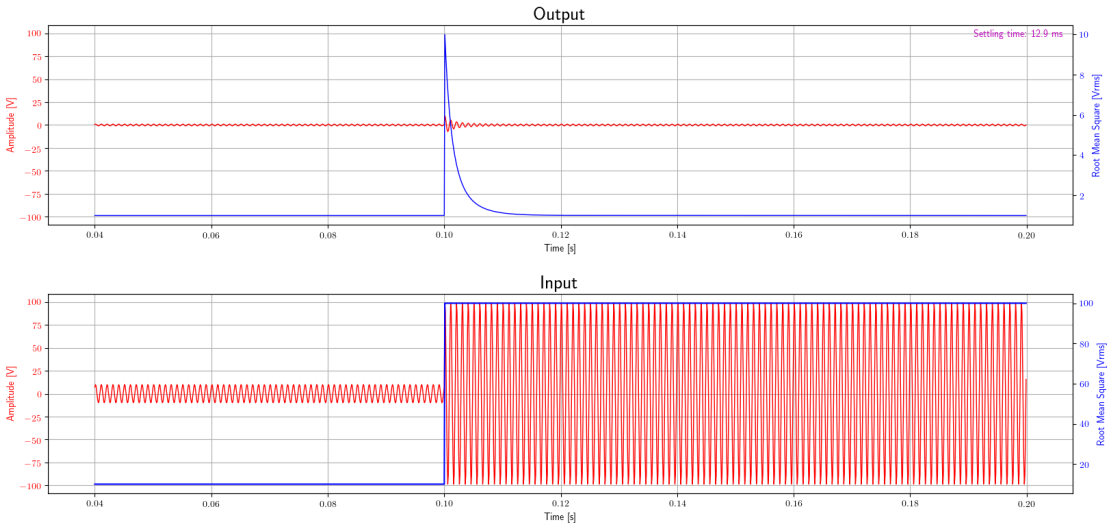

Figure 6.2: AGC test with step of 60 dB

6.1.2 Step of 40 dB

This test case is used to prove that the constant settling time requirement is satisfied with a step of 40 dB.

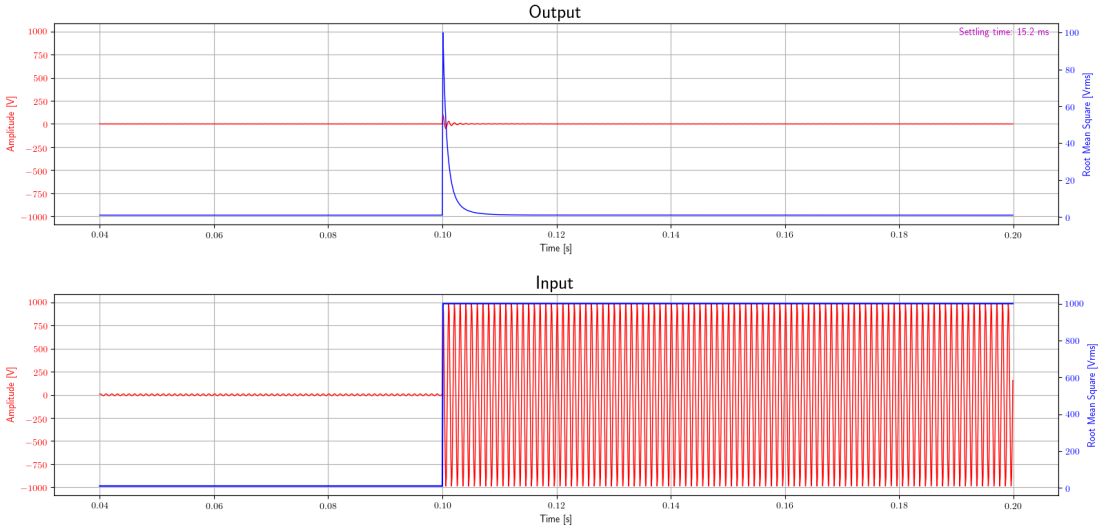

Figure 6.3: AGC test with step of 40 dB

6.1.3 Step of 20 dB

This test case is used to prove that the constant settling time requirement is satisfied with a positive of 20 dB.

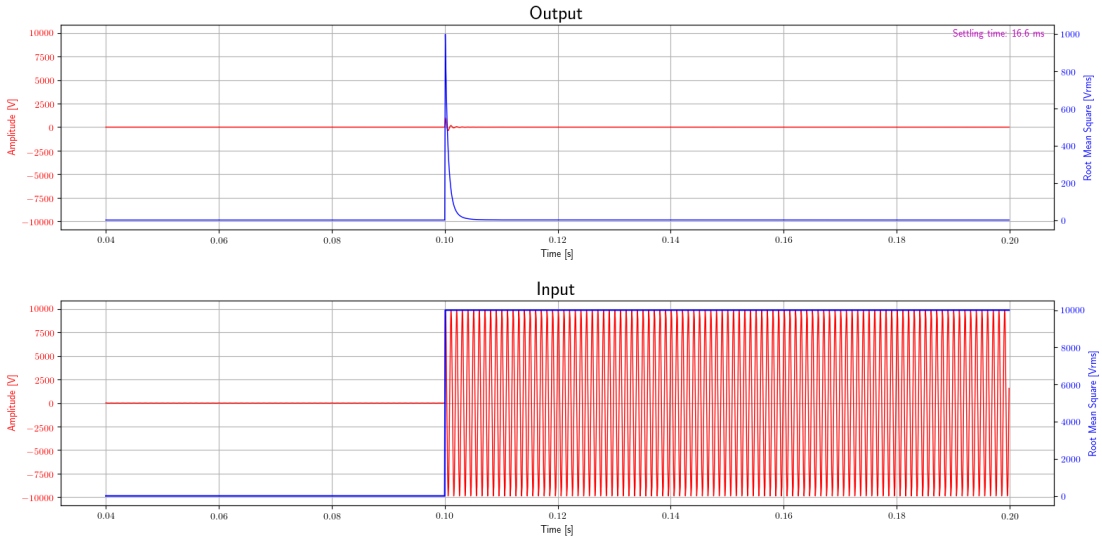

Figure 6.4: AGC test with step of 20 dB

6.1.4 Small step

This test case is used to prove that the constant settling time requirement is satisfied with a small step. In this specific case, the settling time should not be considered, since with a small step, the

AGC should reach the expected output value shorter than the settling time. Thus, it is checked only the maximum settling time requirement.

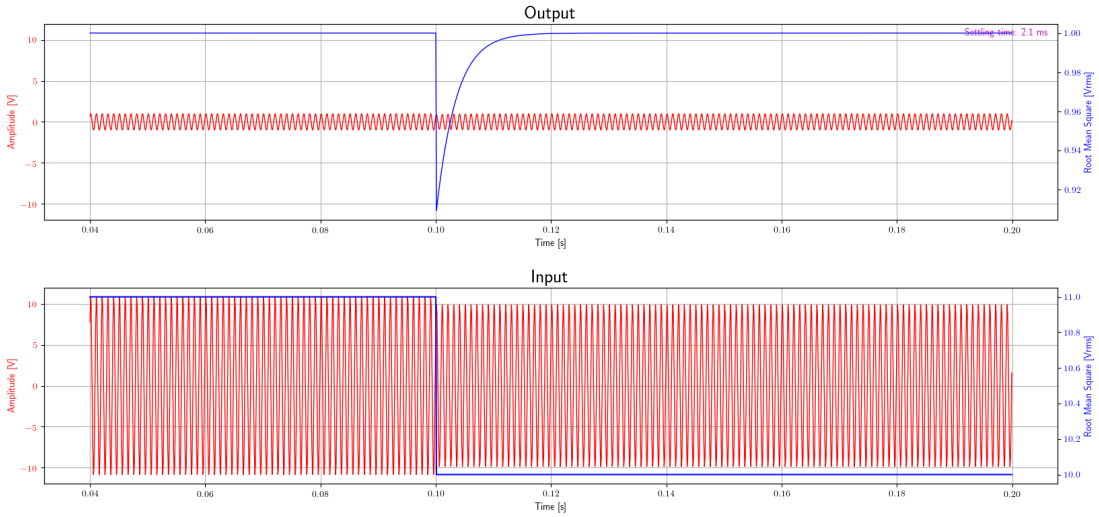

Figure 6.5: AGC test a small step

6.1.5 Step of -20 dB

This test case is used to prove that the constant settling time requirement is satisfied with a step of $-20\ dB$. In this case the time evaluated is called decay time.

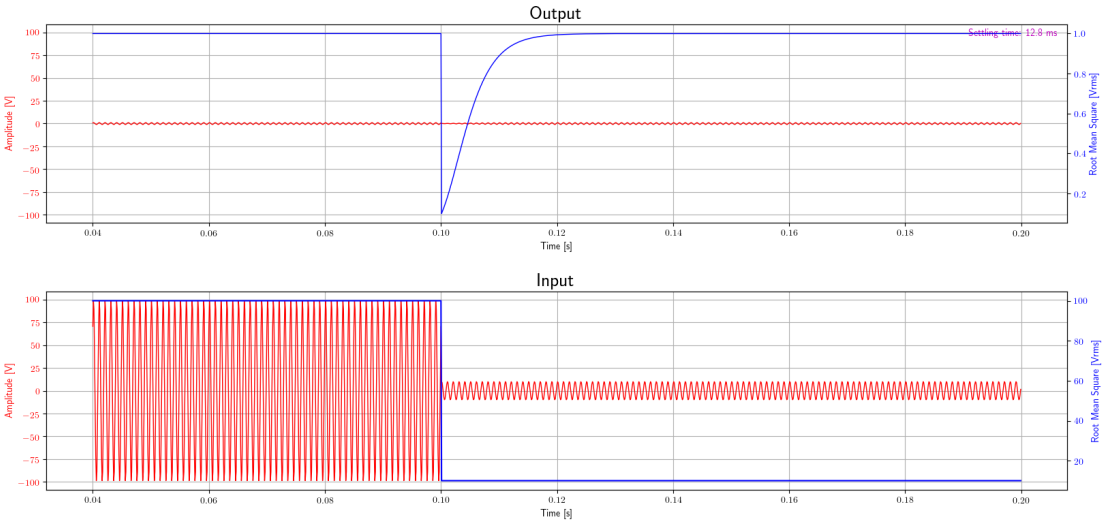

Figure 6.6: AGC test with step of $-20\ dB$

6.1.6 Step of -40 dB

This test case is used to prove that the constant settling time requirement is satisfied with a positive of $-40\ dB$. In this case the time evaluated is called decay time.

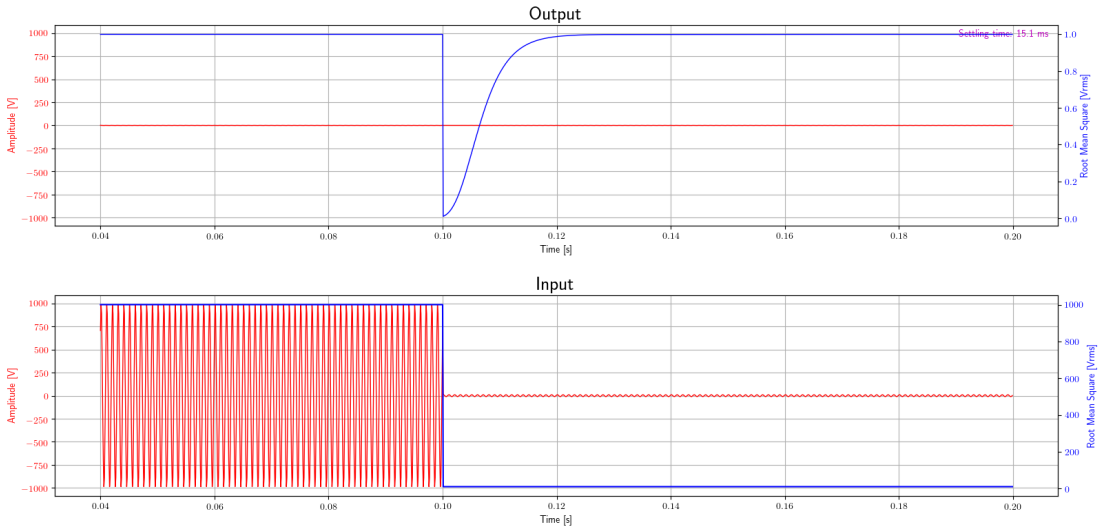

Figure 6.7: AGC test with step of $-40\ dB$

6.1.7 Step of -60 dB

This test case is used to prove that the constant settling time requirement is satisfied with a step of $-60\ dB$. In this case the time evaluated is called decay time.

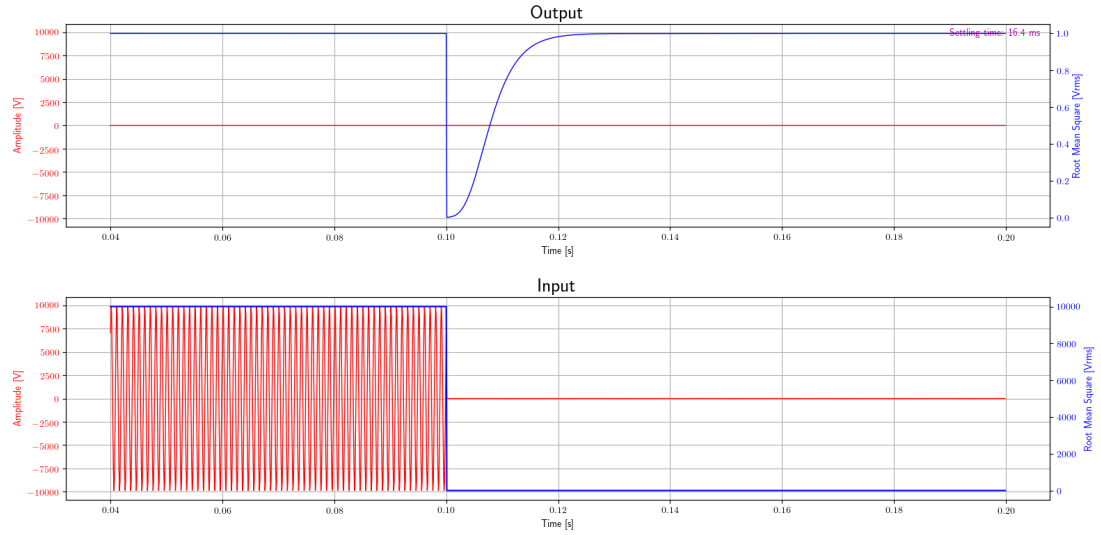

Figure 6.8: AGC test with step of $-60\ dB$

6.1.8 Step of 60 dB with noise

This test case is used to prove that the constant settling time requirement is satisfied with a step of $60\ dB$ with noise.

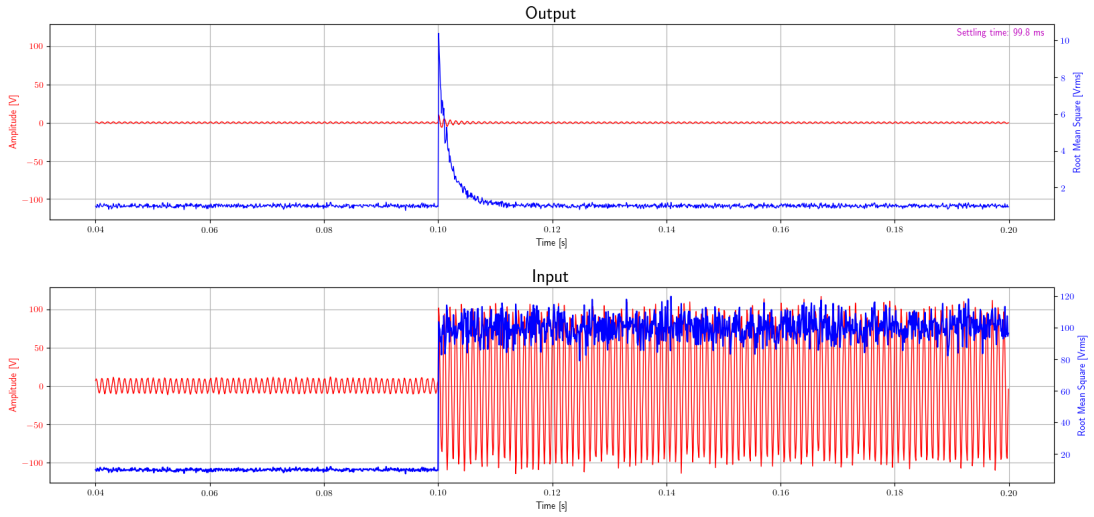

Figure 6.9: AGC test with step of 60 dB with noise

6.1.9 Step of 40 dB with noise

This test case is used to prove that the constant settling time requirement is satisfied with a step of 40 dB with noise.

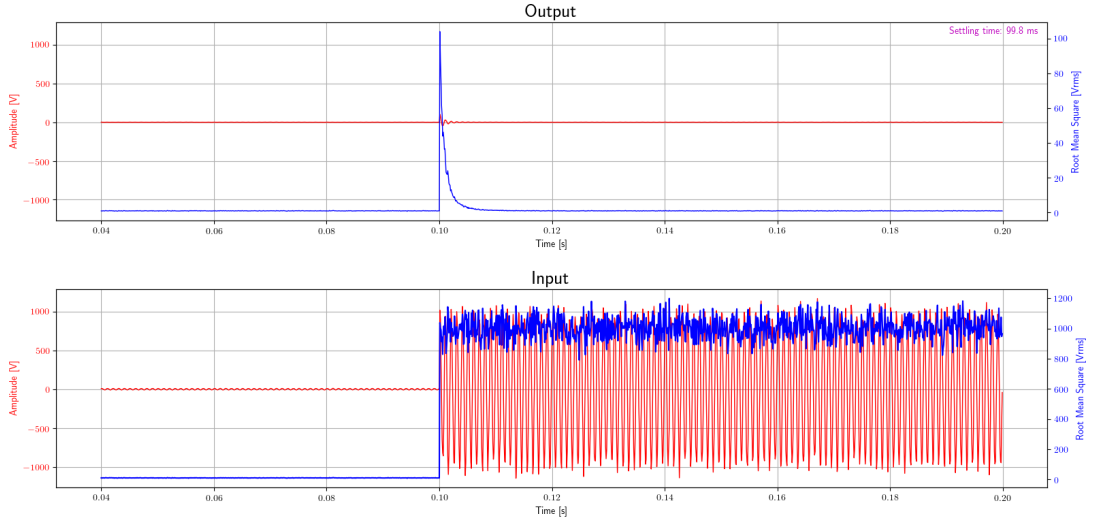

Figure 6.10: AGC test with step of 40 dB with noise

6.1.10 Step of 20 dB with noise

This test case is used to prove that the constant settling time requirement is satisfied with a step of 20 dB with noise.

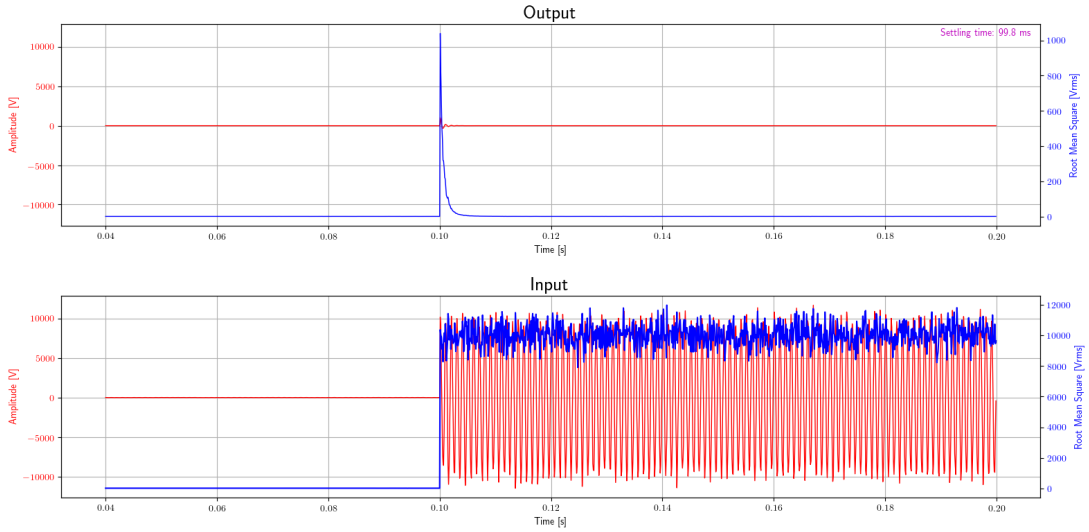

Figure 6.11: AGC test with step of 20 dB with noise

6.1.11 Small step with noise

This test case is used to prove that the constant settling time requirement is satisfied with a small step with noise. In this specific case, the settling time should not be considered, since with a small step, the AGC should to reach the expected output value shorter than the settling time. Thus, it is checked only the maximum settling time requirement.

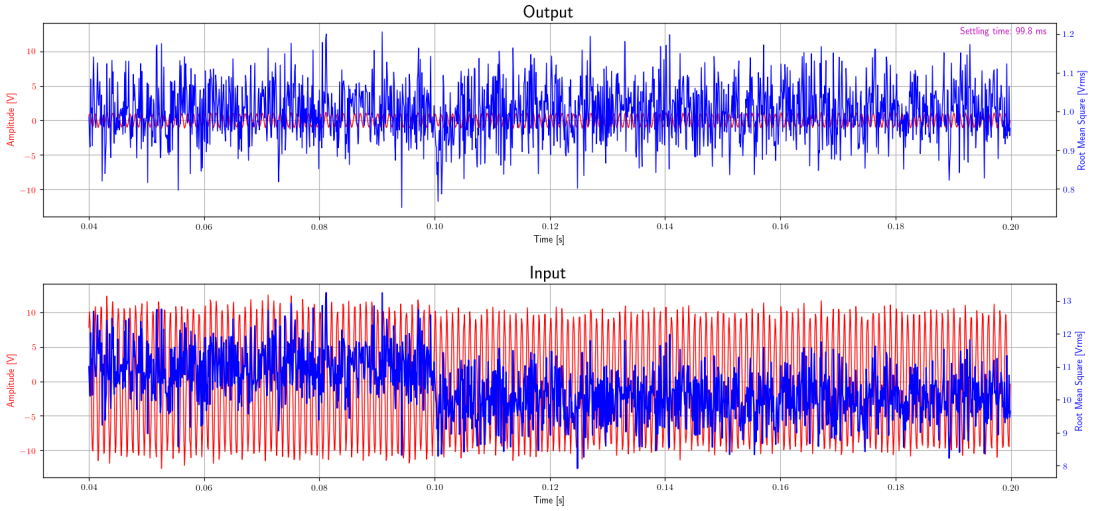

Figure 6.12: AGC test with small step with noise

6.1.12 Step of -20 dB with noise

This test case is used to prove that the constant settling time requirement is satisfied with a positive step of -20 dB with noise. In this case the time evaluated is called decay time.

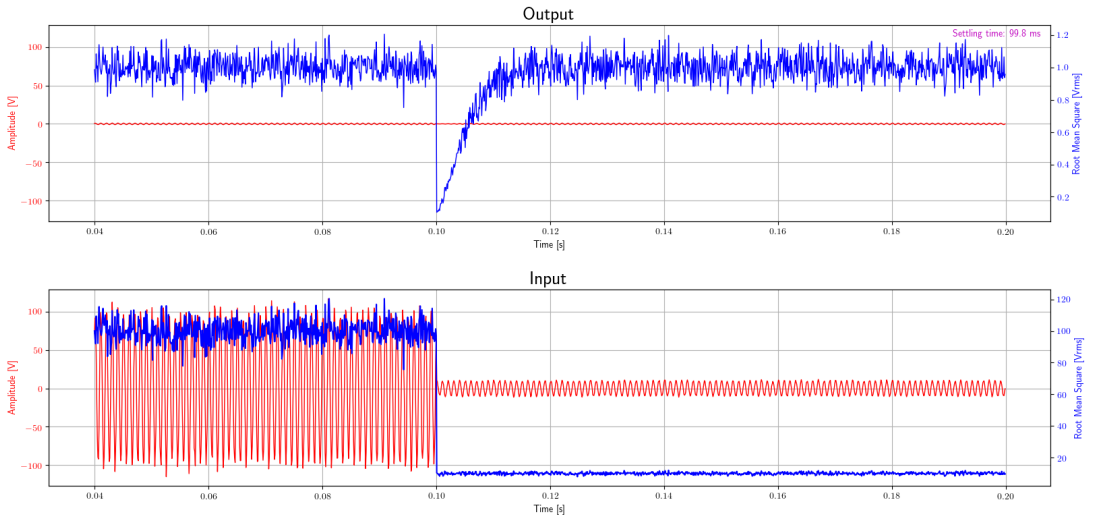

Figure 6.13: AGC test with step of -20 dB with noise

6.1.13 Step of -40 dB with noise

This test case is used to prove that the constant settling time requirement is satisfied with a step of -40 dB with noise. In this case the time evaluated is called decay time.

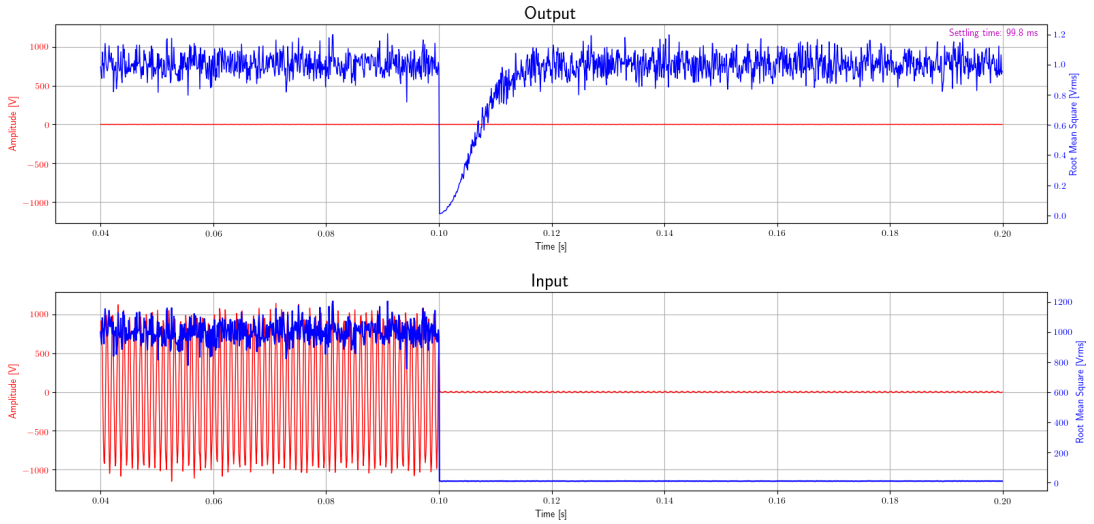

Figure 6.14: AGC test with step of -40 dB with noise

6.1.14 Step of -60 dB with noise

This test case is used to prove that the constant settling time requirement is satisfied with a step of -60 dB with noise. In this case the time evaluated is called decay time.

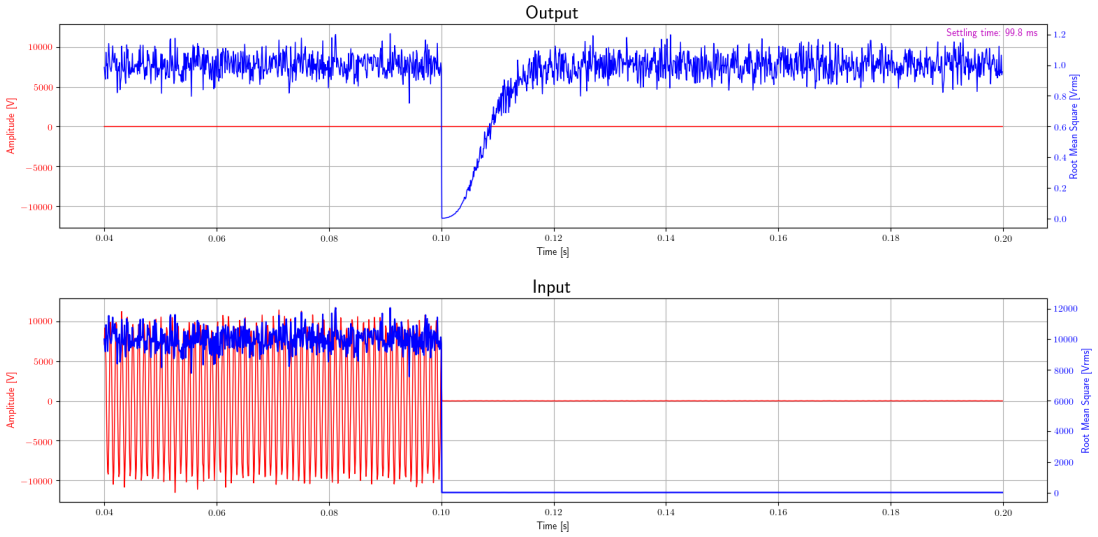

Figure 6.15: AGC test with step of $-60\ dB$ with noise

6.2 Coherent Phase Modulation

The Coherent Phase Modulation block can have several inputs, each of them is Int64 type and it is designed in order to work with the phase accumulator of the PLL.

The Coherent phase modulator generates a complex signal modulated in phase by the sum of inputs. Since the inputs of this block are Int64, to generate a proper input signal on GNU Radio it is needed the phase converter block, present into the ECSS OOT. In order to make easy the test, a basic flow graph is repeated several times. The basic flow graph has an signal generator block that generate a sine wave, connected to it, there is a block that evaluates the argument of that signal for each complex value. In this way is possible to generate an accurate signal that is the phase of the sine wave. Thus, it is converted as a proper Int64 value and sent to the Coherent Phase Modulation. In this flow graph, the output of the Coherent Phase Modulation shall be a sine wave with exactly the same frequency of the generated one.

Several checks are done on the output, an FFT is performed in order to print the output spectrum, in addition, the phase of signal is evaluated using another complex to argument block. The output phase is checked with a specific function, that uses the same algorithm of the sine debug block. The latter, performs the derivative of the phase signal. The average of this value is also the phase step, that says the output signal frequency with the following relation:

$$f = \frac{1}{2\pi \cdot \theta_{step}} \tag{6.1}$$

where θ_{step} is the phase step. Finally, the variance of the phase signal is performed. Finally, the variance of the signal phase is calculated: this is very similar to the Root Mean Square but the variance is already an available function in Python. The variance has to be zero, otherwise phase jumps have occurred.

6.2.1 Relative low frequency

This test case is used to prove that the block works with a relative low frequency (with respect to the sampling rate) signal generated.

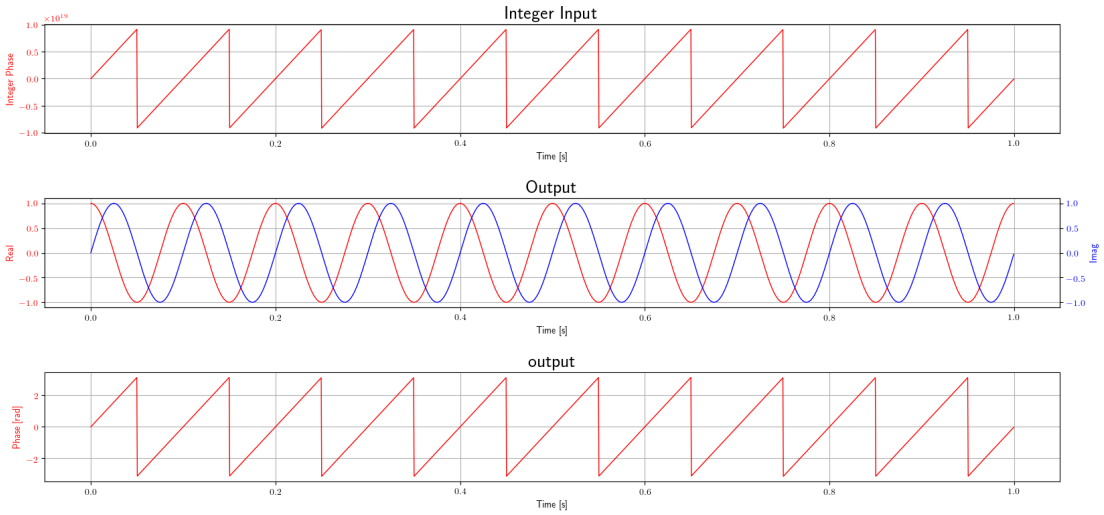

Figure 6.16: Coherent phase accumulator test

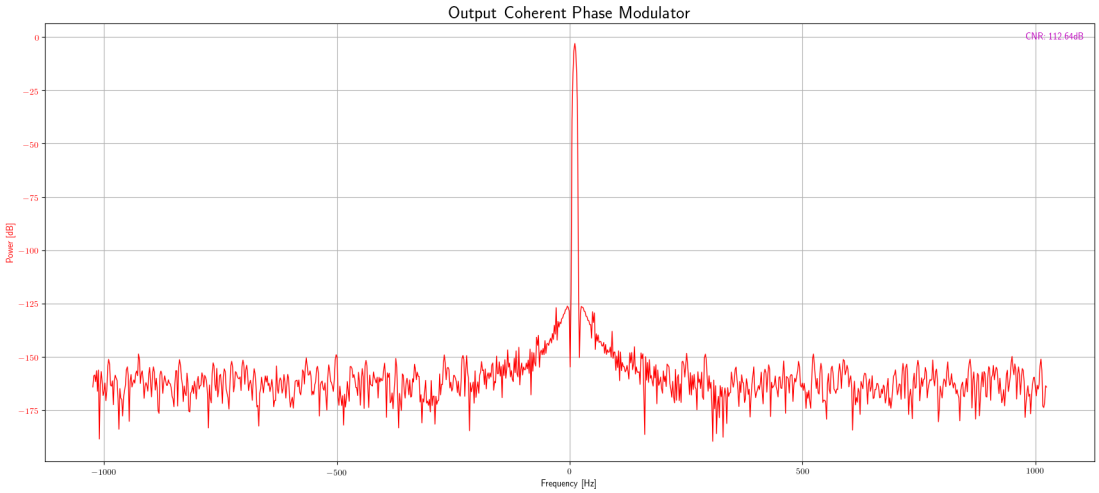

Figure 6.17: Coherent phase accumulator test FFT

Therefore, the test is considered as passed only if the average of the phase step of the output corresponds to the desired one and the variance of the values of phase step is equal to zero.

6.2.2 Relative higher frequency

This test case is used to prove that the block works with a relative high frequency (with respect to the sampling rate) signal generated.

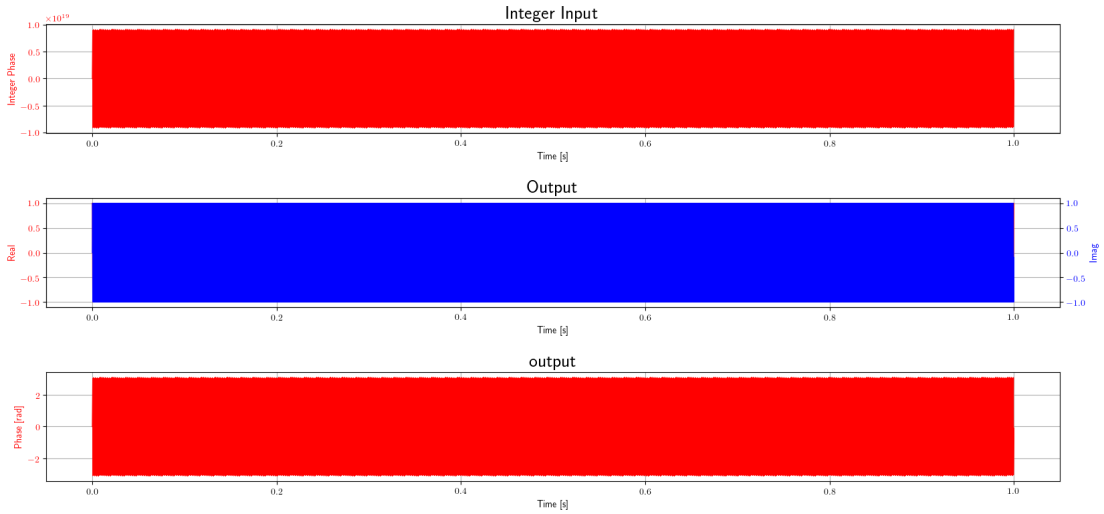

Figure 6.18: Coherent phase accumulator test

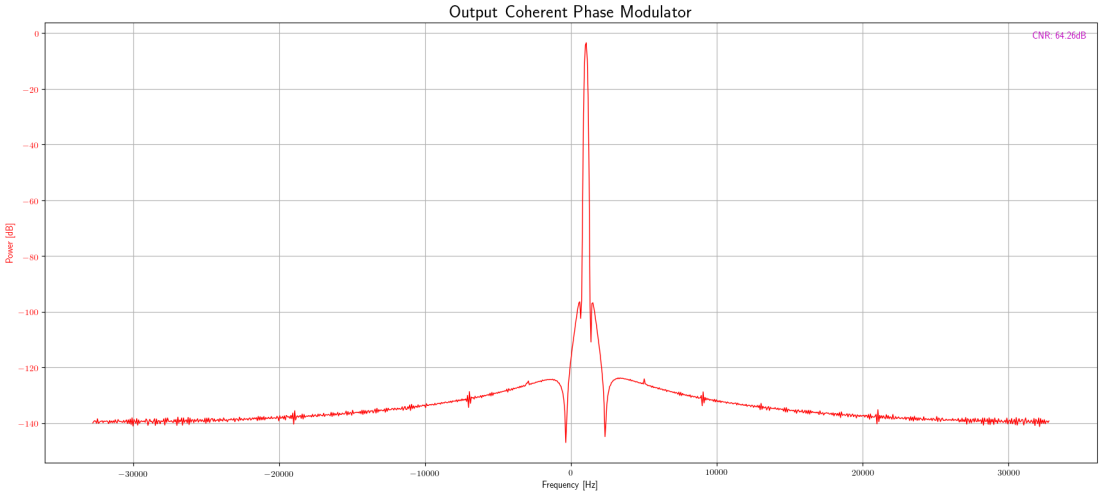

Figure 6.19: Coherent phase accumulator test FFT

Therefore, the test is considered as passed only if the average of the phase step of the output corresponds to the desired one and the variance of the values of phase step is equal to zero.

6.2.3 Gain Phase Accumulator

This test case is used to prove that the block works with the gain phase accumulator.

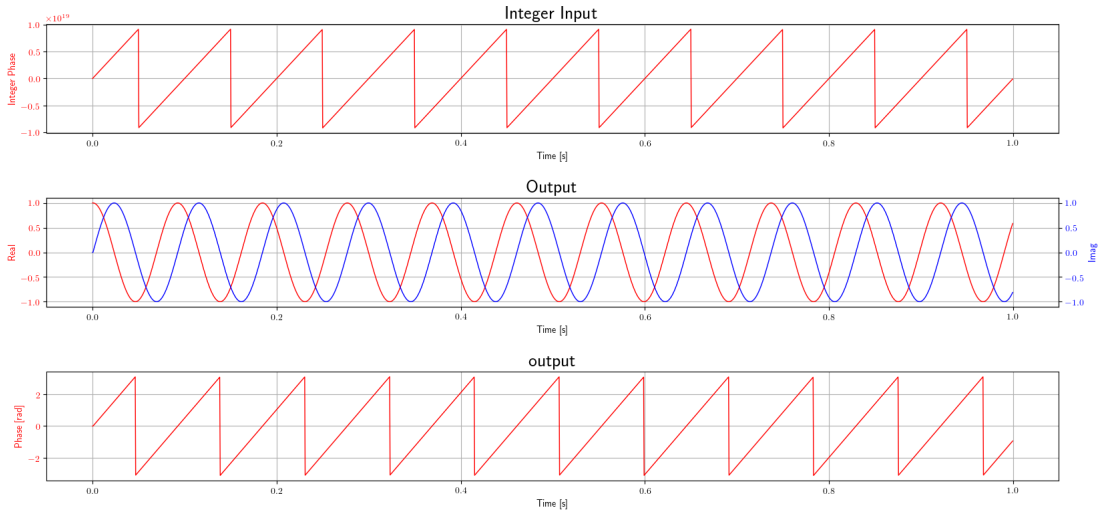

Figure 6.20: Coherent phase accumulator test

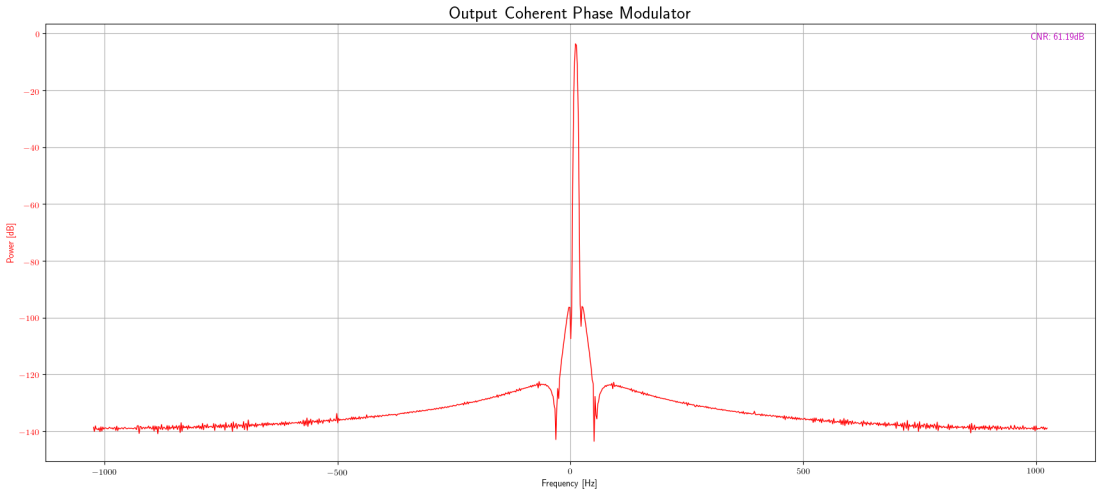

Figure 6.21: Coherent phase accumulator test FFT

Therefore, the test is considered as passed only if the average of the phase step of the output corresponds to the desired one and the variance of the values of phase step is equal to zero.

6.3 Debug Function Probe

As explained above, the purpose of this block is to save the instant in which the callback was called. So, a very simple test has been created, where the callback function is called and if it is saved correctly. Finally, check that the number of saved callbacks is correct and that it happened in a correct instant. Unfortunately, as already mentioned, it is not possible to accurately predict this instant when it depends on two different threads that can not be perfectly synchronized. Therefore, by setting the size of each block of items equal to the sampling rate, and providing to call back the callback at a certain moment, the instant saved by the Debug Function Probe block will be considered correct if it falls within a range of more or less a number of samples equals to sampling rate.

6.3.1 Single set function

In this test only one callback was called.

6.3.2 Double set function

In this test two callbacks were called.

6.4 Demodulator

6.5 Fixed-point Math Emulator

This block converts input data from floating-point to fixed point. It is possible to set both the number of bits of the integer part and those of the fractional part.

All tests were performed simply by creating a vector with input data and one with expected data. The test was considered as passed only if the data in output are the same as those expected.

6.5.1 Q1.4

In this test the block has been set so as to have 1 bit as an integer part and 4 as a fractional part.

6.5.2 Q1.10

In this test the block has been set so as to have 1 bit as an integer part and 10 as a fractional part.

6.5.3 Q10.10

In this test the block has been set so as to have 10 bits as an integer part and 10 as a fractional part.

6.6 Gain Phase Accumulator

The function of the Gain Phase Accumulator block is to modify the slope of the Phase Accumulator output of the PLL in order to consider the Turn-Around Ratio. Therefore, it is necessary to first test the two modes, Coherent and Non-Coherent. Furthermore, it is necessary to test that the block output respects the expected behaviour. In fact, it was checked for each test that the slope of the output form is the same as the correct input signal of the TAR. Moreover, since the block has been designed to work with an integer Phase Accumulator, it is also necessary to check that the minimum phase step is not smaller than the minimum input at the correct TAR level.
So, for each test a simple ramp signal is generated, after, it is converted through the Phase Converter block into an Int64 phase value perfectly similar to the one output from the PLL. Finally, this signal is used as input to the Gain Phase Accumulator block.

6.6.1 Wrapping test

In this test, according to ECSS [31], the $TAR = 221/240$ is used.

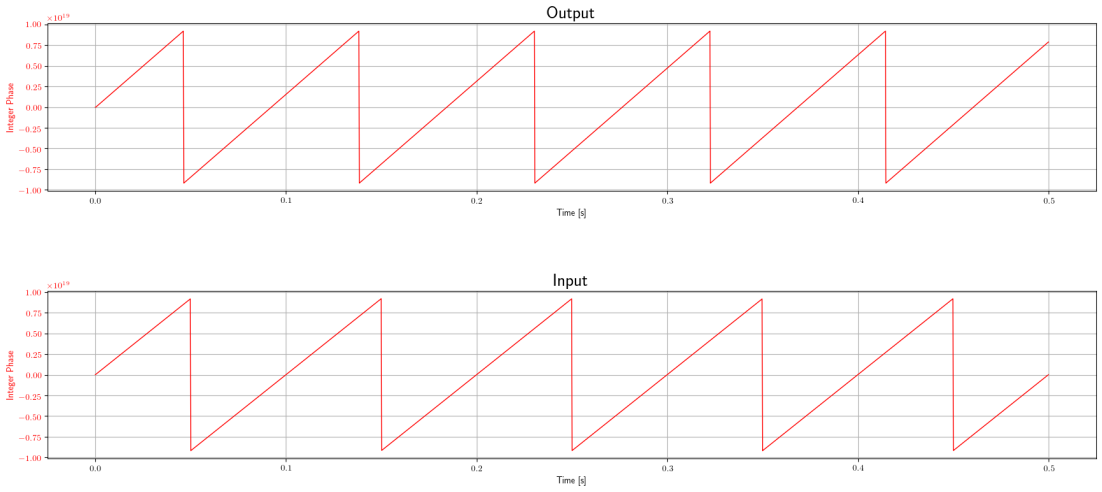

Figure 6.22: Gain phase accumulator test

6.6.2 Wrapping test with high ratio

In this test, according to ECSS [31], the highest $TAR = 221/2850$ is used.

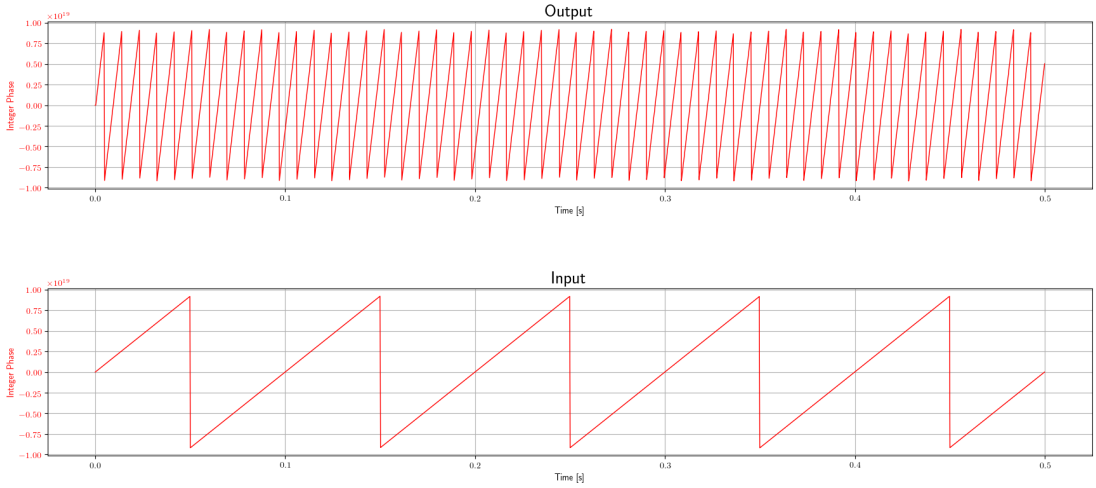

Figure 6.23: Gain phase accumulator test

6.6.3 Wrapping test with higher slope

In this test a higher slope, so frequency, is used.

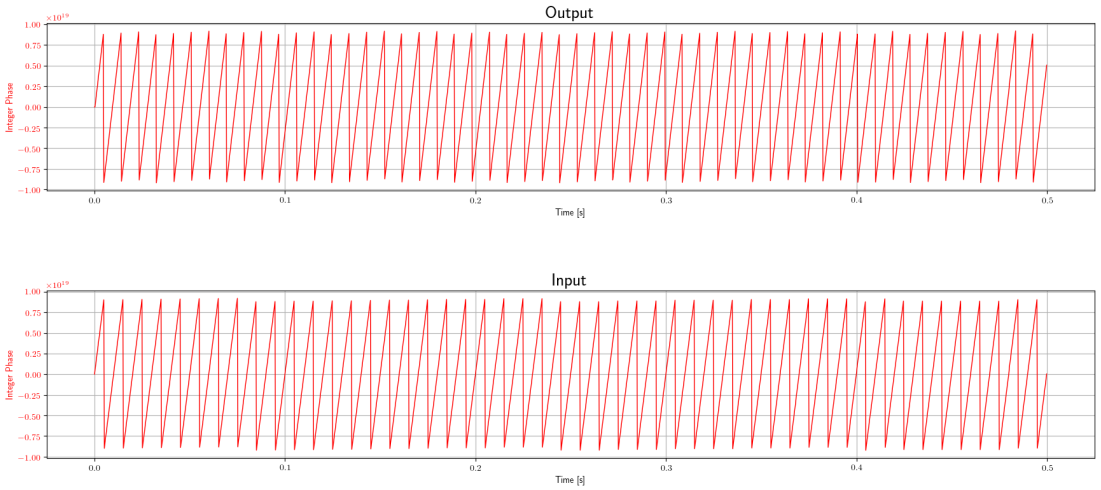

Figure 6.24: Gain phase accumulator test

6.6.4 Precision test

In this test the precision of the input signal is set, through the Phase Converter block, on a number of bits equal to 4.

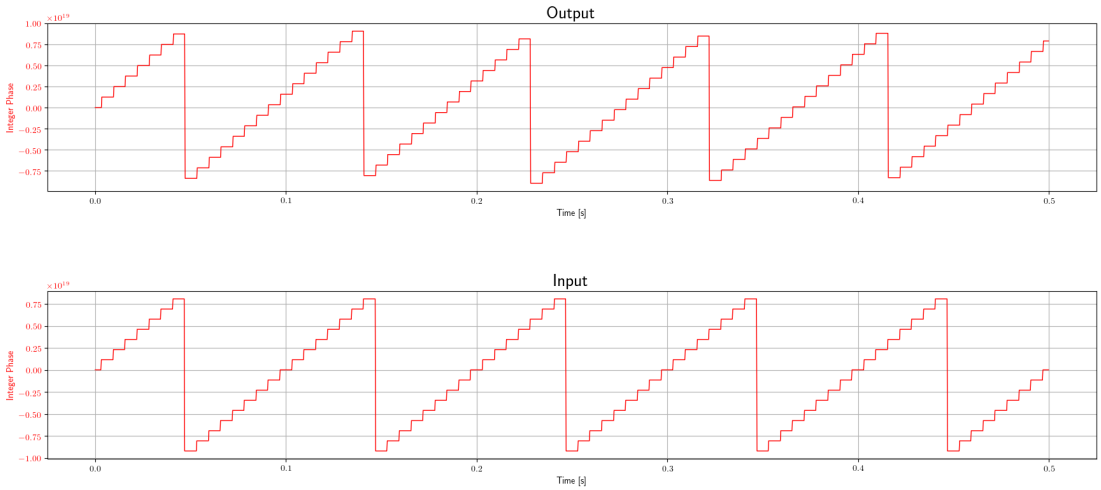

Figure 6.25: Gain phase accumulator test

6.6.5 Reset in the middle of the simulation

In this text the change between coherent and non-coherent modes is tested. This is done by calling a function in the middle of the simulation. What is important to check, is that at the exact moment of the switching, the value of the input and output phase is the same. Moreover, before this event, the exit must be zero.

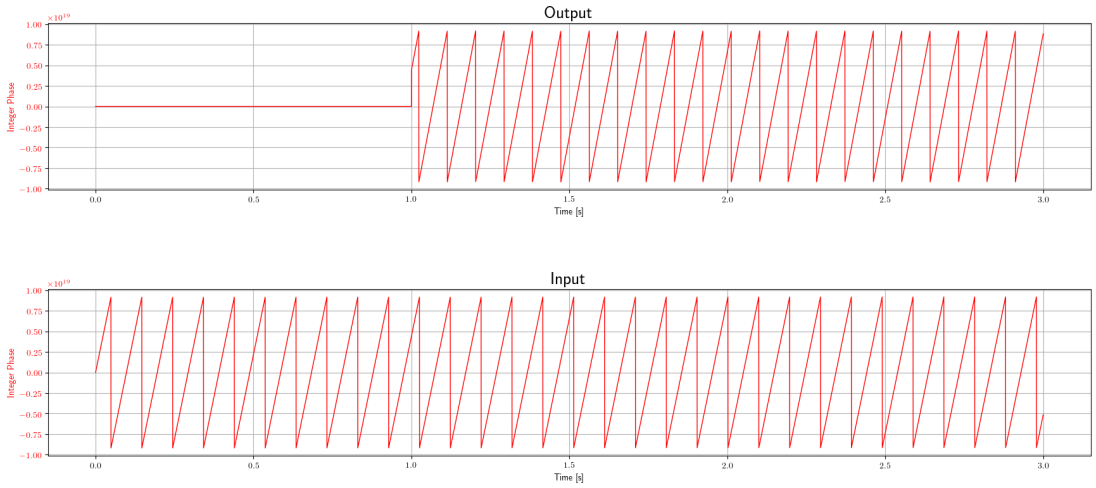

Figure 6.26: Gain phase accumulator test

6.7 Modulator

Since this block is hierarchical, its test has been carried out separately for each block from which it is composed. Furthermore, the Convolutional Encoder block has been tested despite being already present on GNU Radio. In fact, it is necessary to check that the parameters that are set correctly reflect those required by the relevant ECSS document, as mentioned earlier in the section 4.4.9. In this regard, ISIS has provided two files to test this block. One of the two files contains data to be inserted into the Convolutional Encoder block, while the other contains the expected data. This data was generated by one of the encoders currently used by ESA.

6.7.1 NRZL encoder

For this block it was decided to create a simple test. Two data vectors were generated, one to be used as input and one to control the output. Therefore, if the output data are equal to the expected ones, the test can be defined as passed.

6.7.1.1 Sample per symbol of 2

In this test has been set a ratio between sampling rate and bit rate equal to 2. It is also equivalent to the number of sampling for each bit.

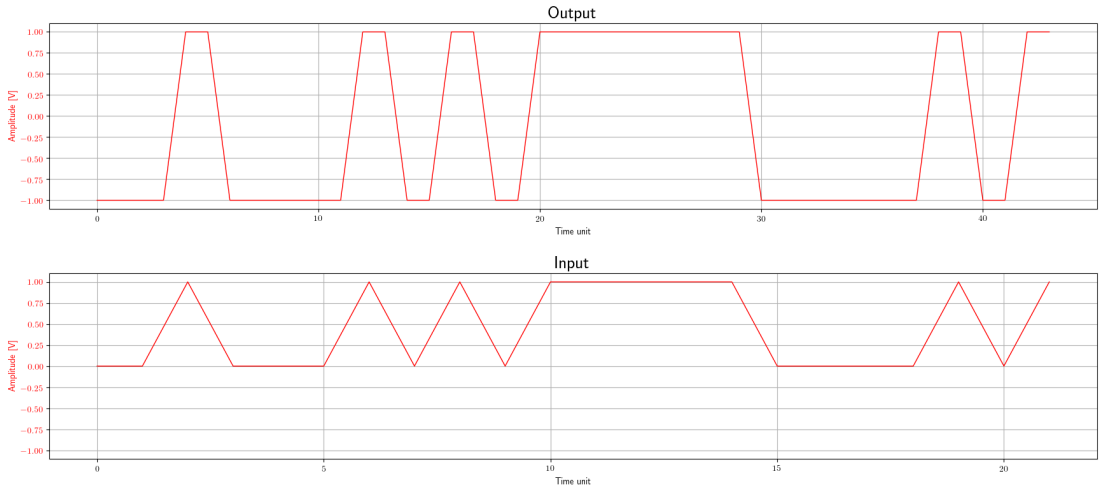

Figure 6.27: NRZL encoder test

6.7.1.2 Sample per symbol of 4

In this test has been set a ratio between sampling rate and bit rate equal to 4. It is also equivalent to the number of sampling for each bit.

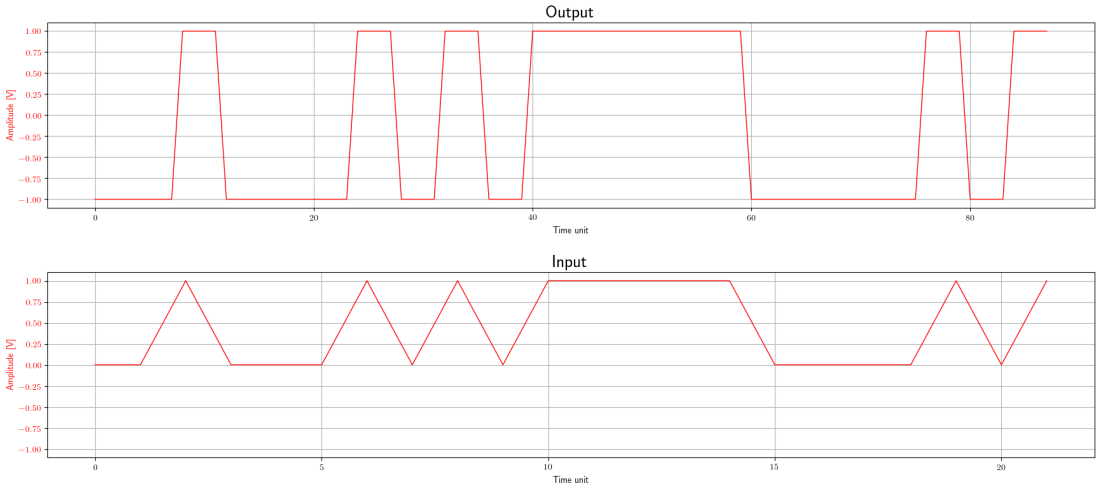

Figure 6.28: NRZL encoder test

6.7.2 NRZL with sub-carrier encoder

For this block it is necessary to test if the output sub-carrier is correctly modulated, but also if it produces the correct output waveform.

6.7.2.1 Sine wave

In this test we want to check if a sine wave is actually generated at the correct frequency and at the correct sampling rate when an input composed of '1' is present.

The test consists in generating a sine wave through a signal generator block at the same frequency set on the NRZL with subcarrier encoder block and at the same sampling rate. In fact, a bit rate equal to

one has been set, so the sampling rate of the signal coming out of the NRZL with subcarrier encoder block will be equal to the set one. Finally, it has been checked that the two signals are similar.

6.7.2.2 Cosine wave

In this test we want to check if a cosine wave is actually generated at the correct frequency and at the correct sampling rate when an input composed of '0' is present.

The test consists in generating a cosine wave through a signal generator block at the same frequency set on the NRZL with subcarrier encoder block and at the same sampling rate. In fact, a bit rate equal to one has been set, so the sampling rate of the signal coming out of the NRZL with subcarrier encoder block will be equal to the set one. Finally, it has been checked that the two signals are similar.

6.7.2.3 Square wave

In this test we want to check if a square wave is actually generated at the correct frequency and at the correct sampling rate when an input composed of '1' is present.

The test consists in generating a square wave through a signal generator block at the same frequency set on the NRZL with subcarrier encoder block and at the same sampling rate. In fact, a bit rate equal to one has been set, so the sampling rate of the signal coming out of the NRZL with subcarrier encoder block will be equal to the set one. Finally, it has been checked that the two signals are similar.

6.7.2.4 Inverse square wave

In this test we want to check if a inverse square wave is actually generated at the correct frequency and at the correct sampling rate when an input composed of '0' is present.

The test consists in generating a square wave, but with a negative amplitude, through a signal generator block at the same frequency set on the NRZL with subcarrier encoder block and at the same sampling rate. In fact, a bit rate equal to one has been set, so the sampling rate of the signal coming out of the NRZL with subcarrier encoder block will be equal to the set one. Finally, it has been checked that the two signals are similar.

6.7.2.5 Sine wave with data

In this test two data vectors were generated, one to be used as input and one to control the output. Therefore, if the output data are equal to the expected ones, the test can be defined as passed.

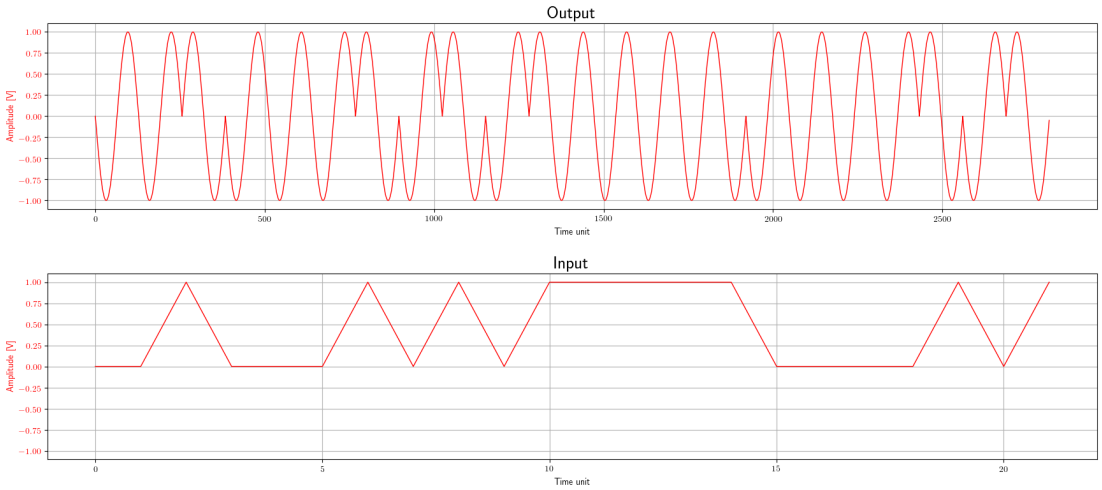

Figure 6.29: NRZL with subcarrier encoder test

6.7.2.6 Square wave with data

In this test two data vectors were generated, one to be used as input and one to control the output. Therefore, if the output data are equal to the expected ones, the test can be defined as passed.

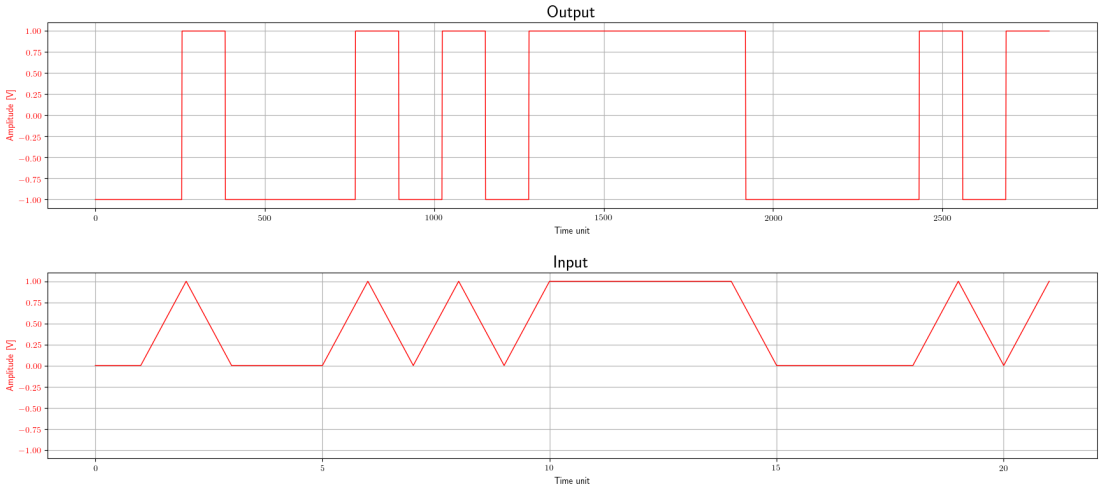

Figure 6.30: NRZL with subcarrier encoder test

6.7.3 SPL encoder

For this block it was decided to create a simple test. Two data vectors were generated, one to be used as input and one to control the output. Therefore, if the output data are equal to the expected ones, the test can be defined as passed.

6.7.3.1 Sample per symbol of 1

In this test has been set a ratio between sampling rate and bit rate equal to 1. It is also equivalent to the number of sampling for each bit.

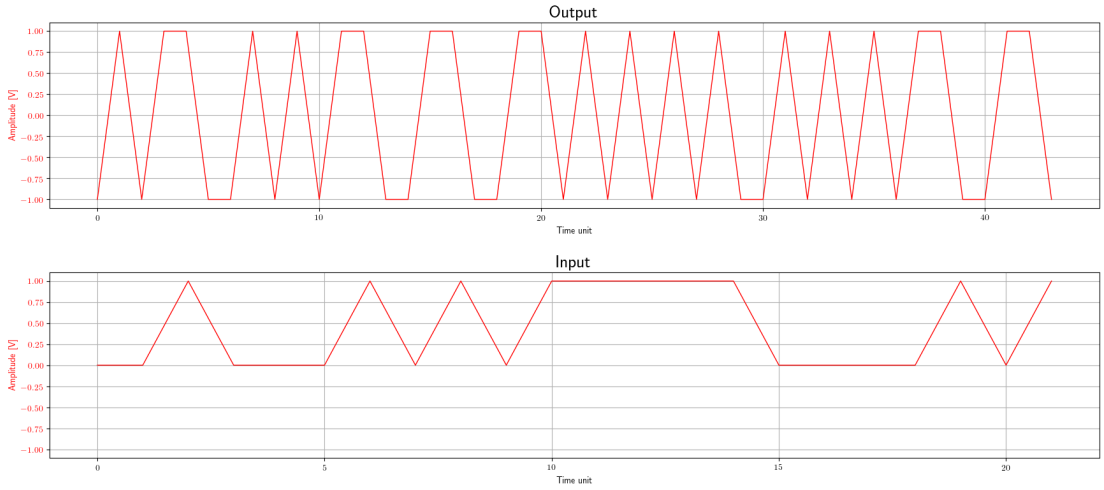

Figure 6.31: SPL encoder test

6.7.3.2 Sample per symbol of 2

In this test has been set a ratio between sampling rate and bit rate equal to 2. It is also equivalent to the number of sampling for each bit.

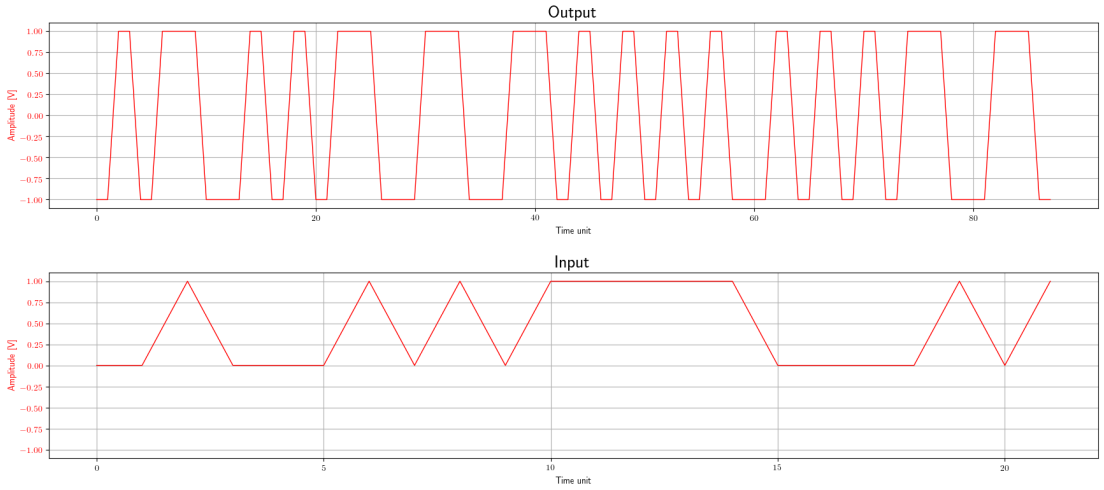

Figure 6.32: SPL encoder test

6.8 Integer Math Emulator

This block saturates an integer at most obtainable with the number of bits set.
All tests were performed simply by creating a vector with input data and one with expected data. The test was considered as passed only if the data in output are the same as those expected.

6.8.1 N = 4

In this test the block has been set so as to have 4 bit as an integer part.

6.8.2 N = 8

In this test the block has been set so as to have 8 bit as an integer part.

6.8.3 N = 16

In this test the block has been set so as to have 16 bits as an integer part.

6.9 IQ imbalance Correction

This block is already present on GNU Radio. However, it was decided to test its correctness, in particular it was wanted to create if it applies a delay between the input and the output in addition to unbalancing in phase and amplitude.

6.9.1 Cross-correlation

In this test the cross-correlation between the input and output data was applied.

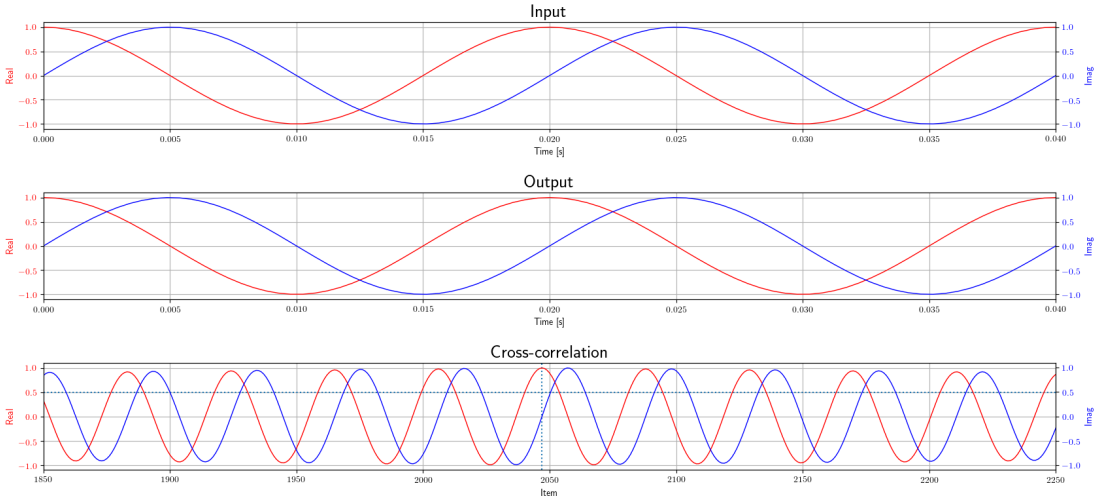

Figure 6.33: Cross-correlation without IQ imbalance

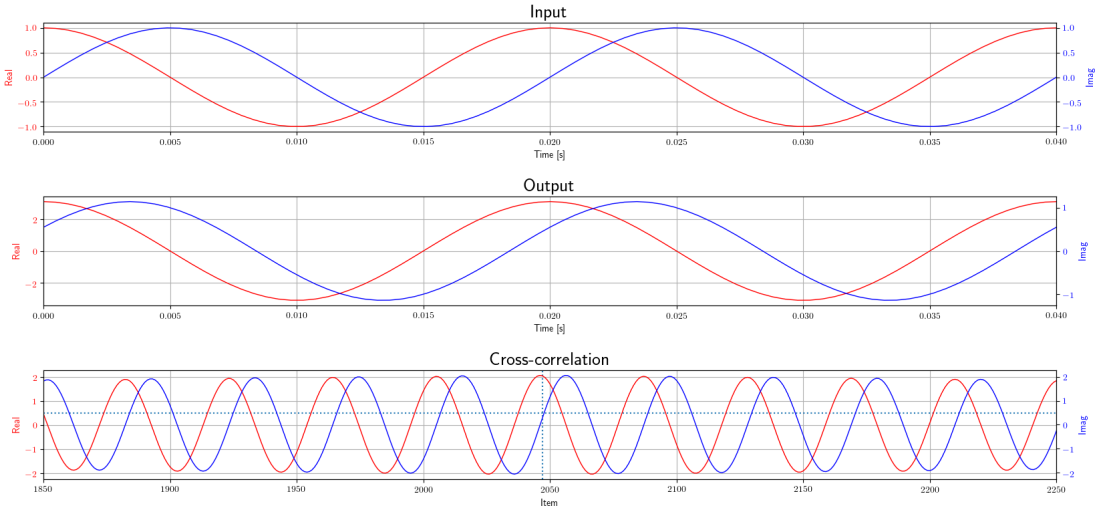

Figure 6.34: Cross-correlation with IQ imbalance

Through the following test it is possible to state the correct behaviour of the block. In particular, that does not apply any time shift, but only an imbalance of the components.

6.10 Null Sink and Source

This tests are performed as GNU Radio does with the Null Sink and Source blocks. The tests consist to connect the Null Source to the Null Sink directly, and run it. The same flow graph is used for the following types:

- Float;
- Double;
- Short;
- Byte;
- Int64;
- Double;
- Complex.

6.11 Phase Converter

This block converts a phase data from float to Int64 type. It has been chosen to perform a test in which aa ramp of phase values that exceeded -2π and 2π. tTherefore, the block must be able to convert the value and to wrap the input phase.

6.11.1 Wrapping test

In this test we will check if the phase wrap has actually occurred, and if the slope of the output signal is the same as that of the input signal. It is also checked that the minimum step is greater than or equal to the minimum possible for the set bit value.

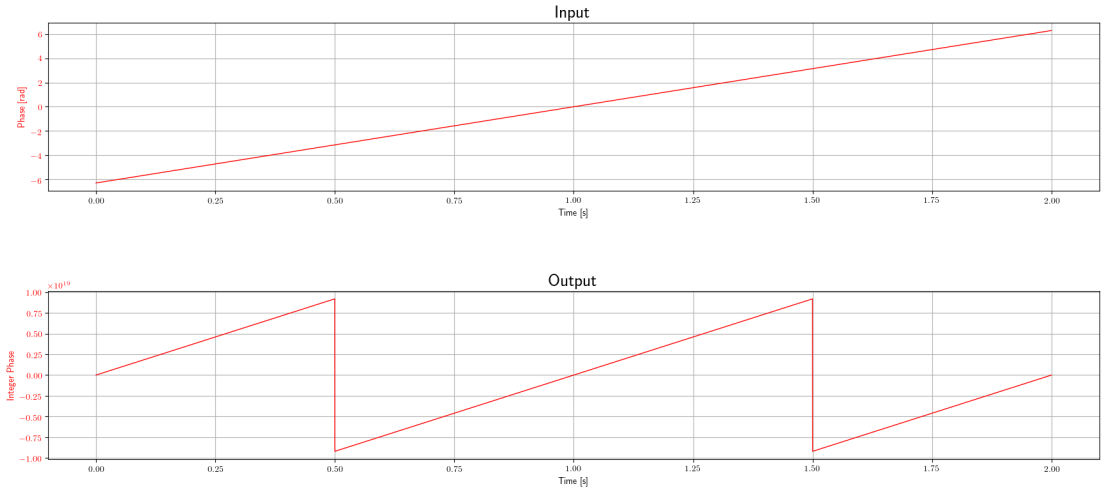

Figure 6.35: Phase converter wrapping test

6.11.2 Precision test

In this test the same checks are made of the previous test, but setting a bit value of 4, in order to have a much larger minimum step.

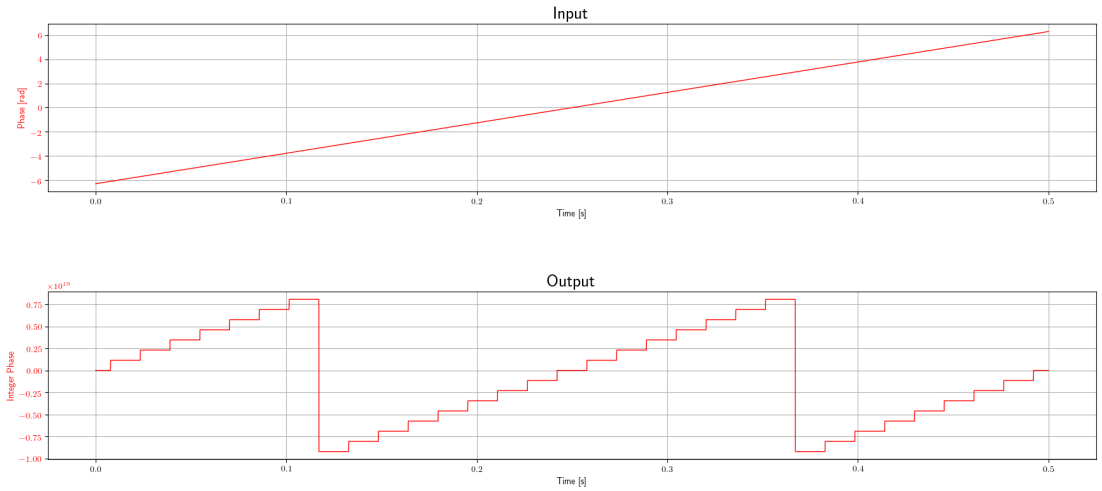

Figure 6.36: Phase converter precision test

As can be see from Figure 6.36, setting a sufficiently small number of bits, the minimum phase step is very evident.

6.12 Phase-Locked Loop

As explained in Section 4.10, the PLL has 4 different outputs, each of them with different aims and types. In order to test if the designed PLL can satisfy all the functions for which it has been designed, several test cases have been used. In many of these, the test bench is similar, so distinct functions have been developer and reused in almost all the tests.

The basic test is called *test_ sine* and consists in a signal source set to output a sine wave with settable frequency (the amplitude is set to one, since in the designed transponder, an AGC is present in front of PLL. Moreover, the offset is not relevant and always set to zero). In addition, a noise source with settable noise amplitude is added to the signal source, in order to test input with noise cases. Furthermore, a throttle and a head block is added for a proper test. Finally, each PLL output is stored in different sinks. To take the Phase Accumulator output, a Int64 type vector sink has been developed, since no already available on GNU Radio.

In order to plot a spectral representation of the input and output of PLL, a similar test function has been developed, but with the designed SNR estimator in input and output of PLL. the others outputs of PLL, have been linked to a null source. Also in this case, an Int64 type null sink has been developed, since no already available on GNU Radio.

The PLL output *out* is the only complex type, it is the input signal multiplied with the reference signal generated by PLL. Thus, it shall be ideally the input signal shifted to 0 Hz. So, the ideal value of Real part will be 1 and the Imag part will be 0 when the PLL is locked.
One generic function *check_ complex* has been developed in order to test mainly out. It checks if the last Real and Imag values at the end of simulation are effectively the expected ones (it is supposed that the simulation is enough long to allow PLL to lock) plus or minus an error, both settable. Moreover, if the preliminary check is passed, is evaluated the settling time as the moment into Real or Imag value overtakes the defined error for at least five items, in order to take into account glitch due to noise.

For all the tests it has been set up for the second order loop mode $\zeta = 0.7$. This because the PLL has the best transient response at the value of damping, since the function of $B_L(\zeta)$ is quite flat close to $\zeta = 0.5$ and $B_L = 0.53\omega_n$[46]. Thus, for all the tests the following parameters were used, with a sampling rate of 16 kHz.

For the second order PLL:

- Natural frequency = 10 Hz;
- Damping = 0.7;
- Epsilon = 1.

For the second order PLL:

- Natural frequency = 2 Hz;
- Damping = 0.707;
- Index m = 5.

Obtaining the following internal coefficients:

- Coefficient 1 (2nd order): 0.052267
- Coefficient 2 (2nd order): 0.00143173
- Coefficient 4 (2nd order): 1
- Coefficient 1 (3rd order): 0.022742
- Coefficient 2 (3rd order): $7.18567 \cdot 10^{-05}$
- Coefficient 3 (3rd order):$1.97104 \cdot 10^{-07}$

6.12.1 Input Sine in central bandwidth

In this test an input signal was used within the PLL bandwidth. Thus, it has been checked that the PLL has been able to lock properly.

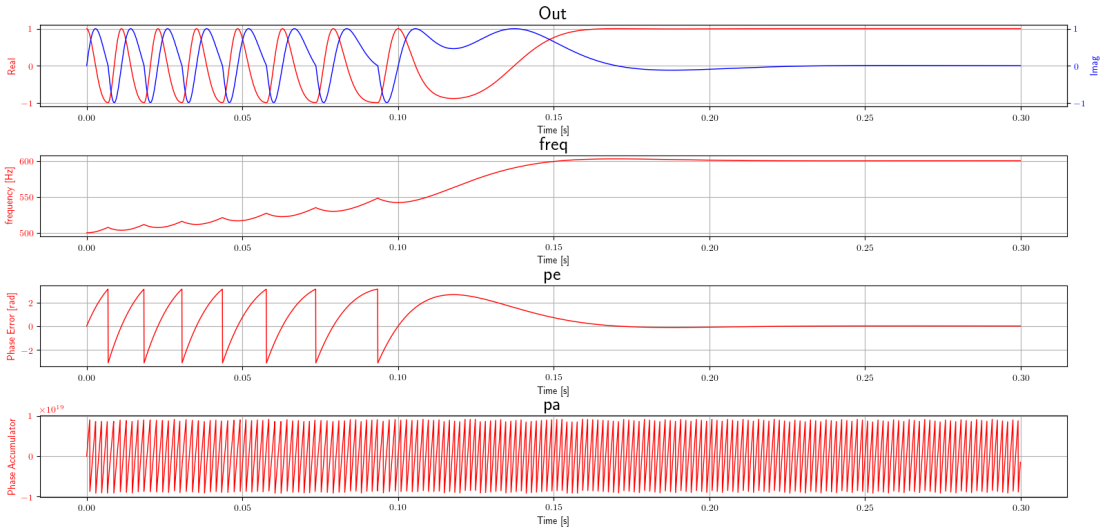

Figure 6.37: PLL outputs

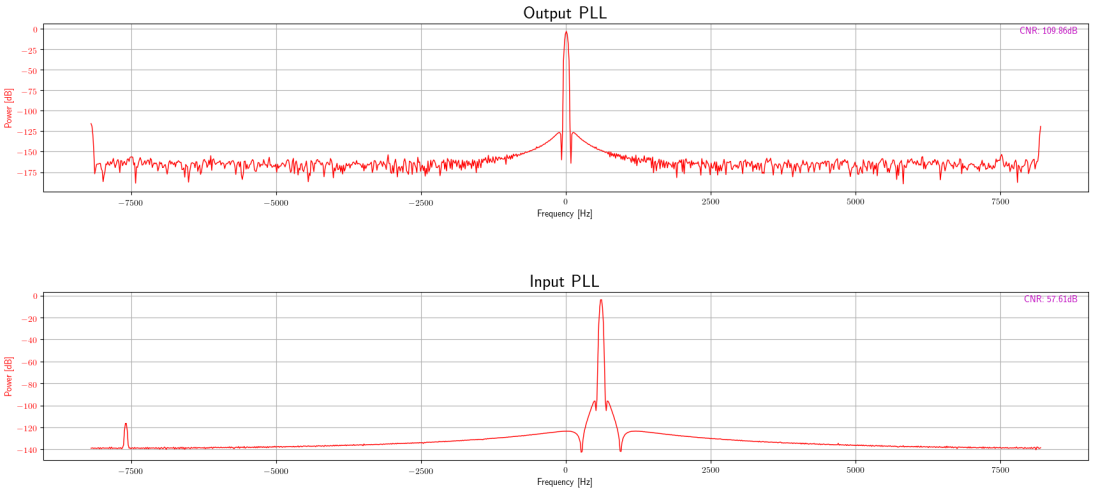

Figure 6.38: FFT input and output of PLL

As to see from the Figure 6.37, in the final part of the graphs the PLL lock occurs. This can be mainly detected by the value of Phase Error (PE) which becomes constant and equal to zero. Moreover, since the output of the PLL corresponds to the input signal translated in base-band, it will be the equivalent of a 0 Hz signal, so the imaginary and real part will be constant and equal to 0 and 1 respectively. The PLL Frequency output becomes constant at the value equal to the frequency of the input signal. Finally, the Phase Accumulator (PA) output will have a constant slope.

6.12.2 Input Sine in the boundary bandwidth

In this test an input signal was used in the border of the PLL bandwidth. Thus, it has been checked that the PLL has been able to lock properly.

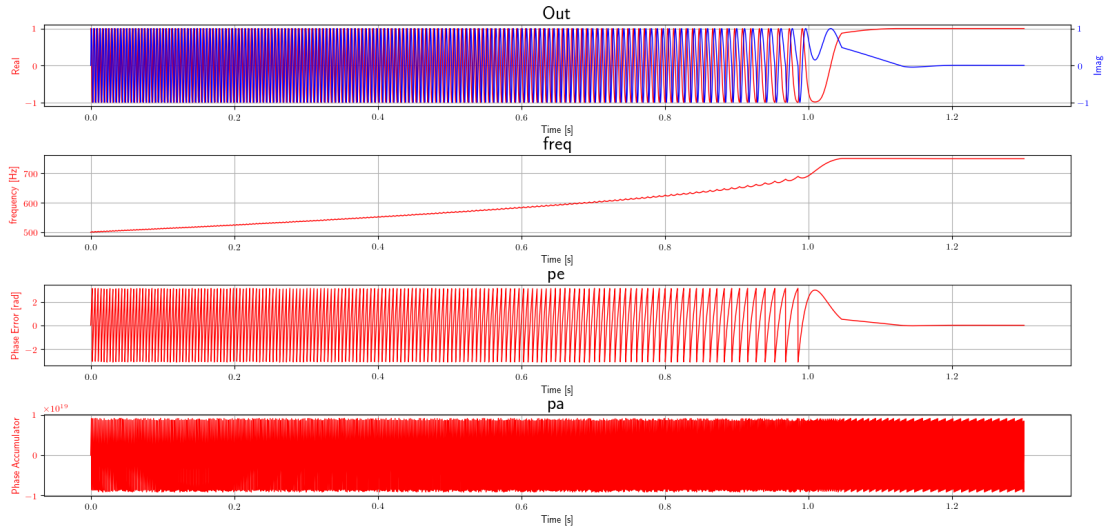

Figure 6.39: PLL outputs

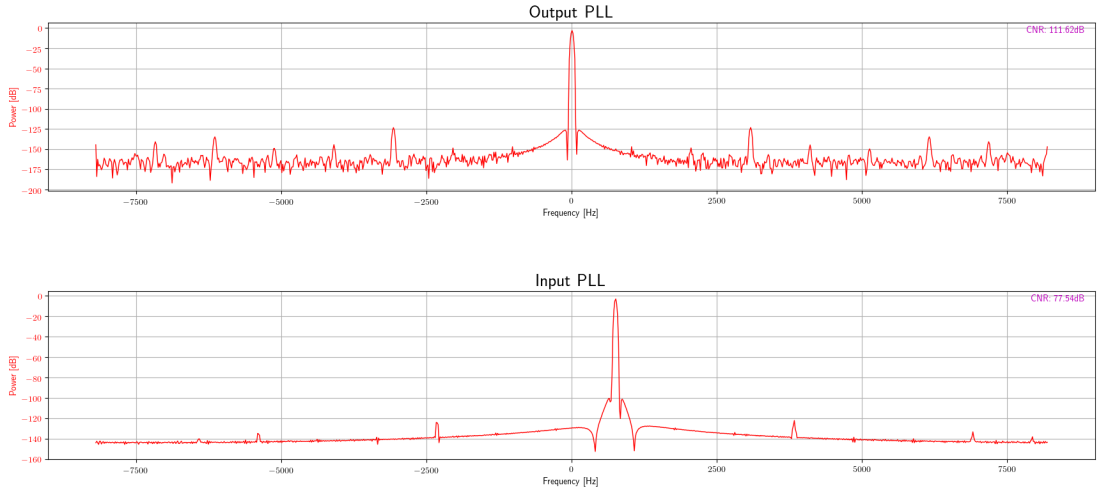

Figure 6.40: FFT input and output of PLL

As it can be seen in Figure 6.39, at the end of the test the phase error goes to zero showing a correct lock of the PLL.

6.12.3 Input Sine out of bandwidth

In this test an input signal was used outside the PLL bandwidth. Thus, it has been checked that the PLL has not been able to lock properly.

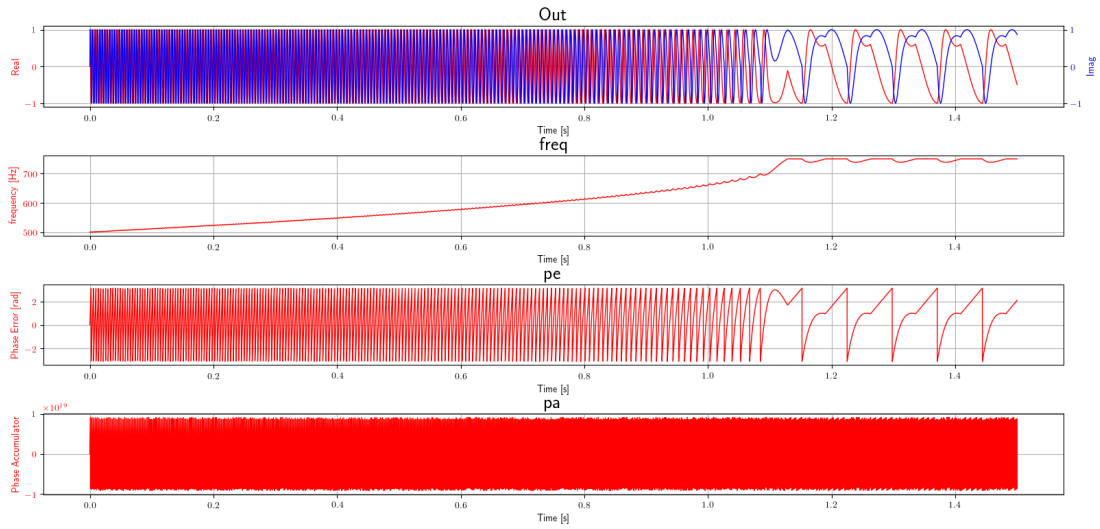

Figure 6.41: PLL outputs

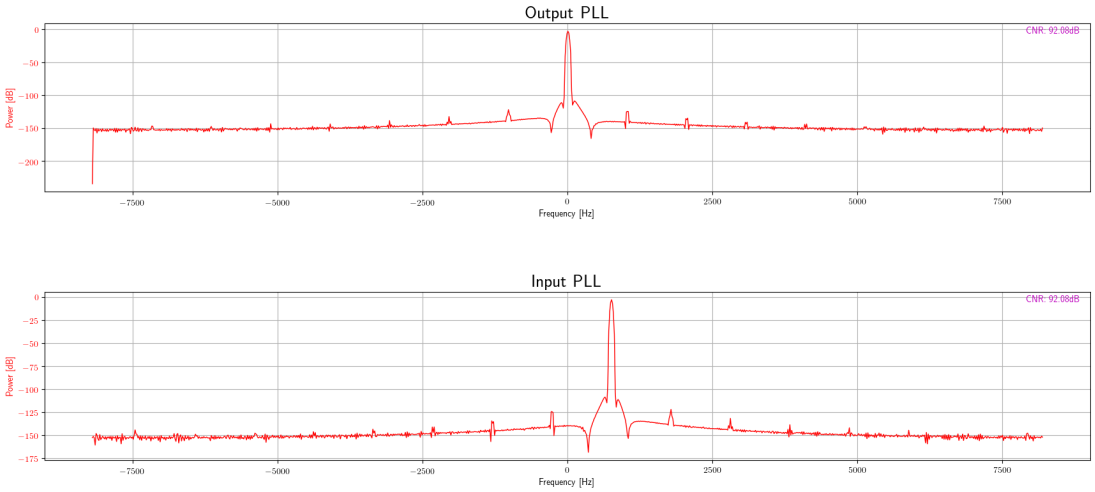

Figure 6.42: FFT input and output of PLL

As it can be seen in Figure 6.41, at the end of the test the phase error does not go to 0, showing the PLL is not locked since the input signal was outside the lock band of the PLL.

6.12.4 Reset Tag

In this test it is checked that after a reset tag, the PLL is correctly reset.

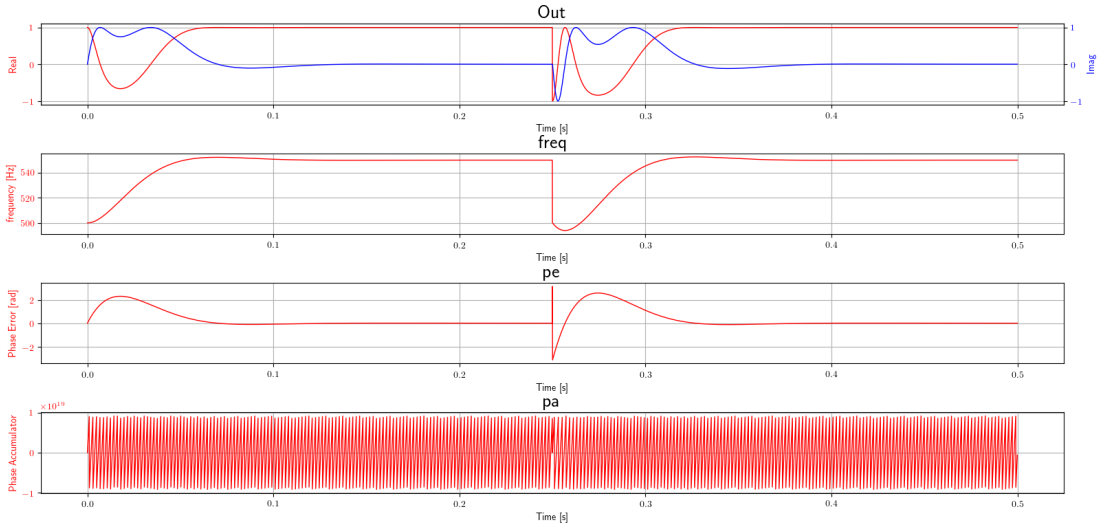

Figure 6.43: PLL outputs

As it can be seen in Figure 6.43, in the middle of the test the reset signal is sent. Therefore, at the next instant the PLL is reset and starts again from the central frequency. In addition, it is possible to see that after the reset signal, the PLL is perfectly able to re-perform the lock.

6.12.5 Switch from 2nd to 3rd order

In this test it is checked that the PLL is able to change from the second to third order and that it is able to lock again.

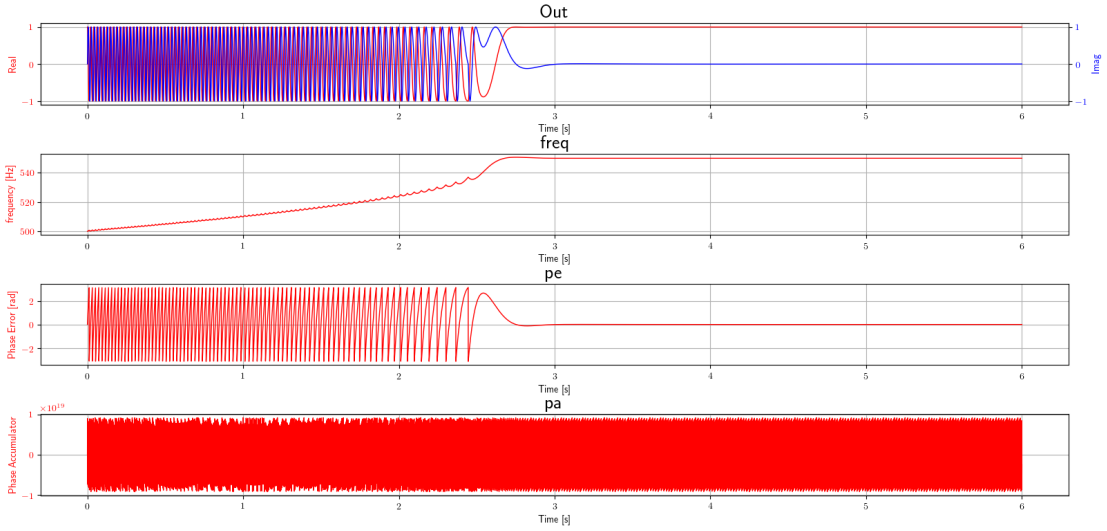

Figure 6.44: PLL outputs

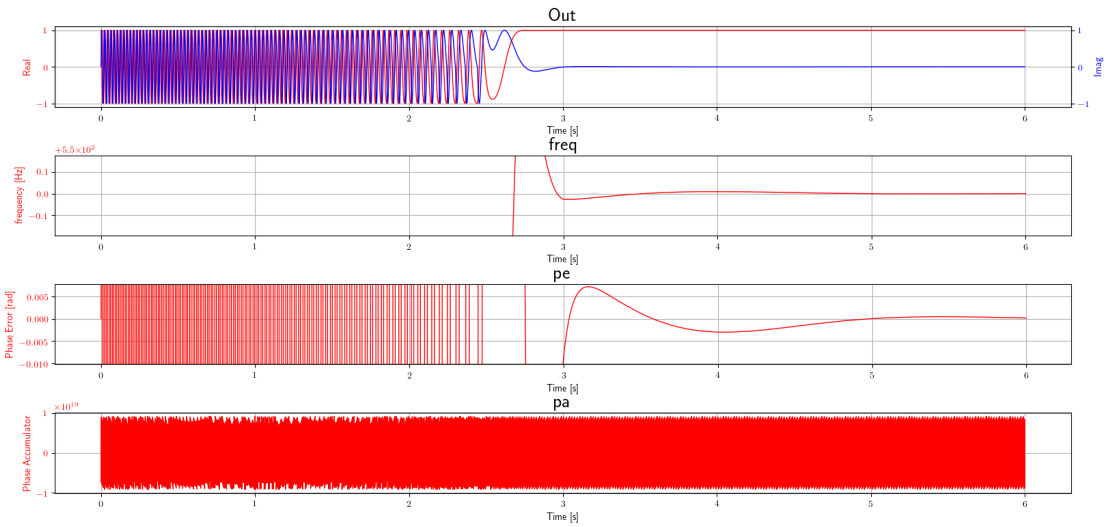

Figure 6.45: PLL outputs zoomed

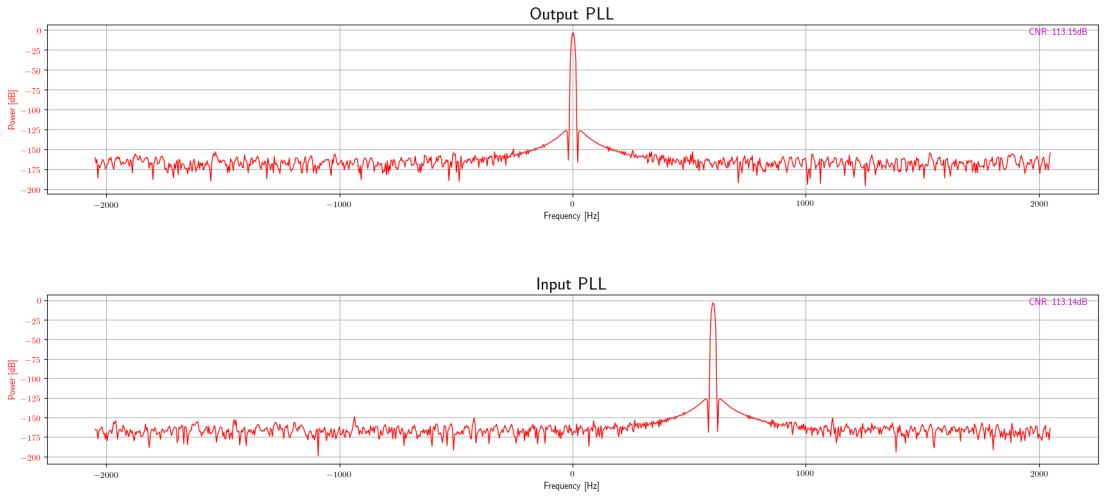

Figure 6.46: FFT output and input PLL

6.12.6 Frequency sweep

In this test it is checked that the PLL is lock when the input signal is sweeping.

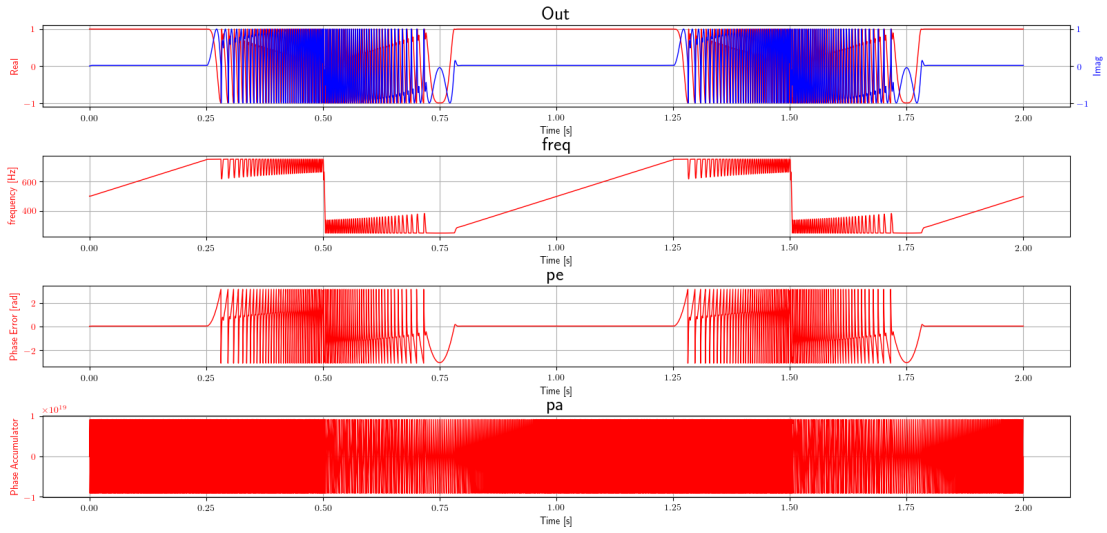

Figure 6.47: PLL outputs

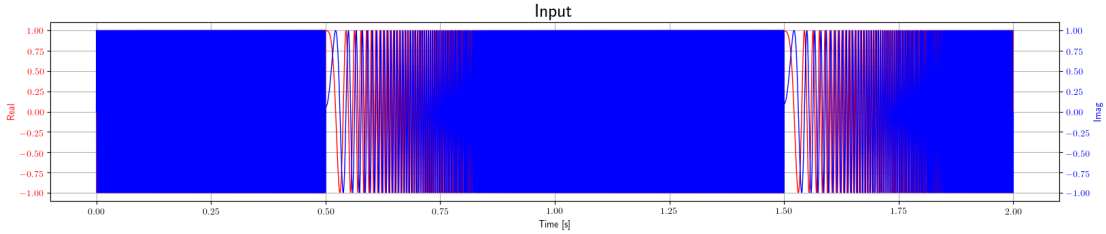

Figure 6.48: PLL input

6.12.7 Input Sine in central bandwidth with noise

In this test an input signal with noise was used within the PLL bandwidth. Thus, it has been checked that the PLL has been able to lock properly.

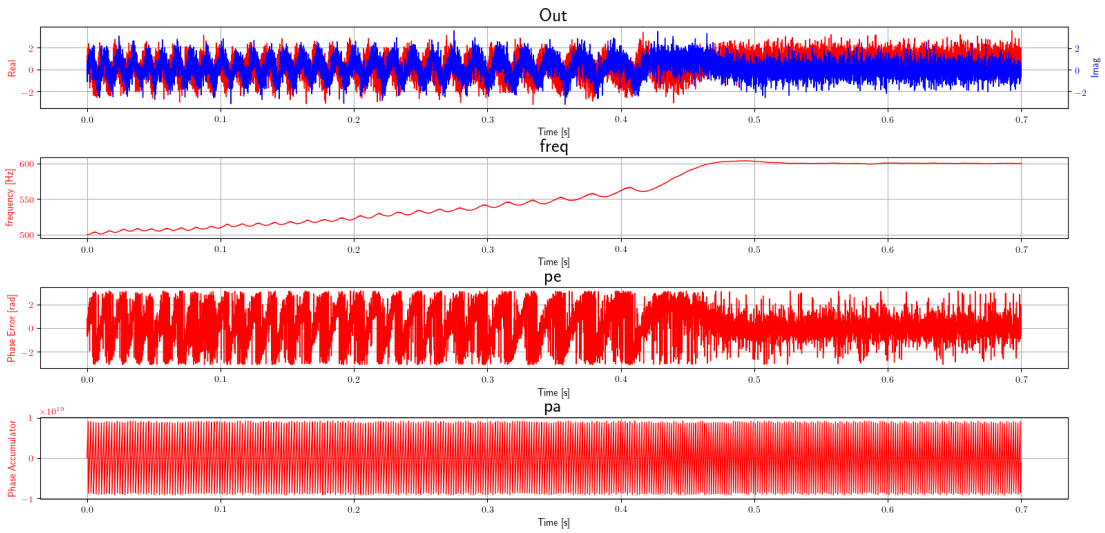

Figure 6.49: PLL outputs

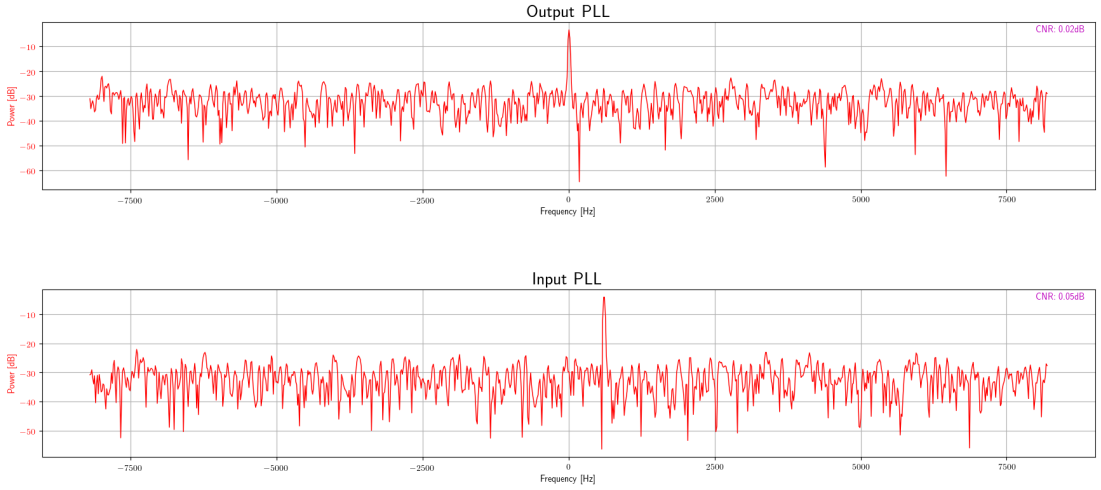

Figure 6.50: FFT output and input PLL

6.12.8 Input Sine in the boundary bandwidth with noise

In this test an input signal with noise was used in the border of the PLL bandwidth. Thus, it has been checked that the PLL has been able to lock properly.

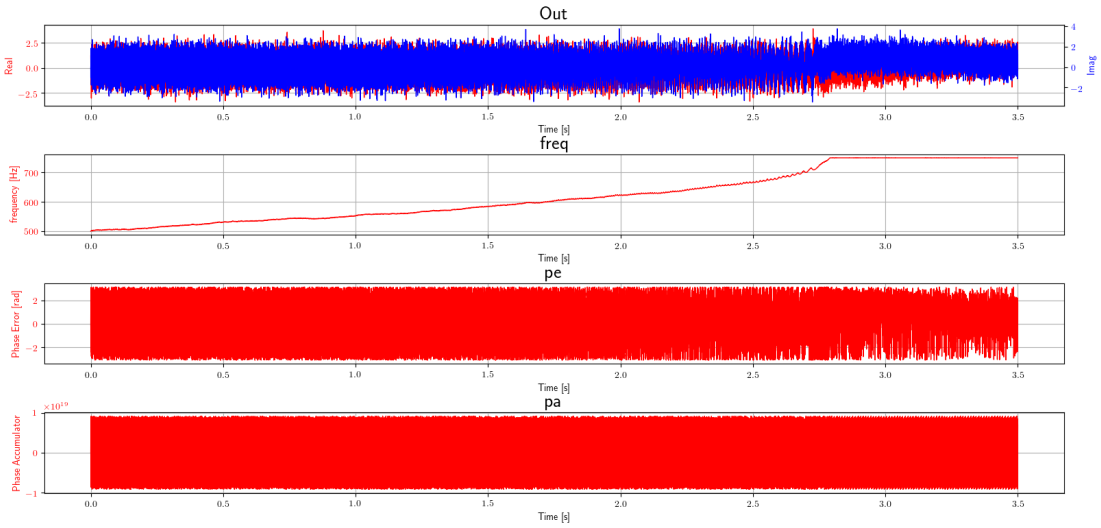

Figure 6.51: PLL outputs

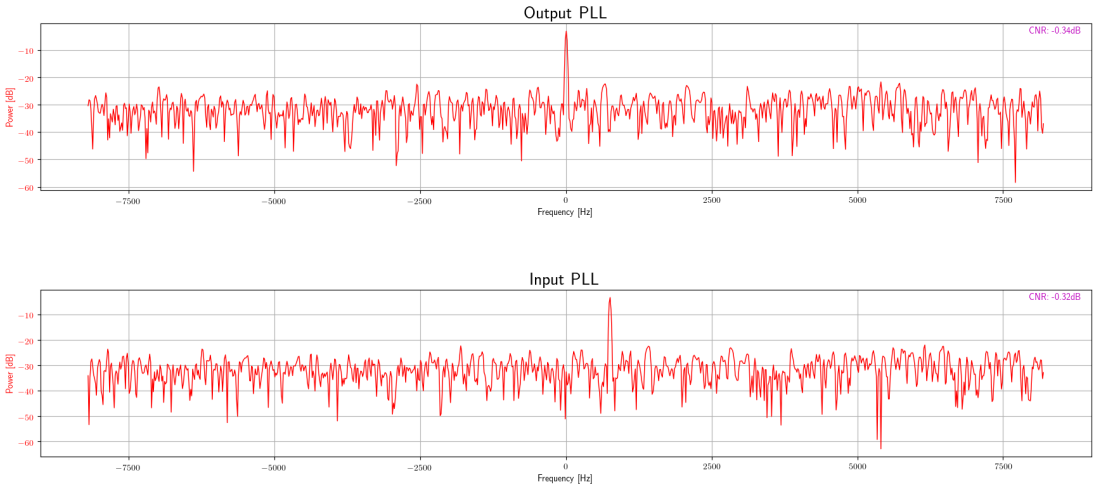

Figure 6.52: FFT output and input PLL

6.12.9 Input Sine out of bandwidth with noise

In this test an input signal with noise was used outside the PLL bandwidth. Thus, it has been checked that the PLL has not been able to lock properly.

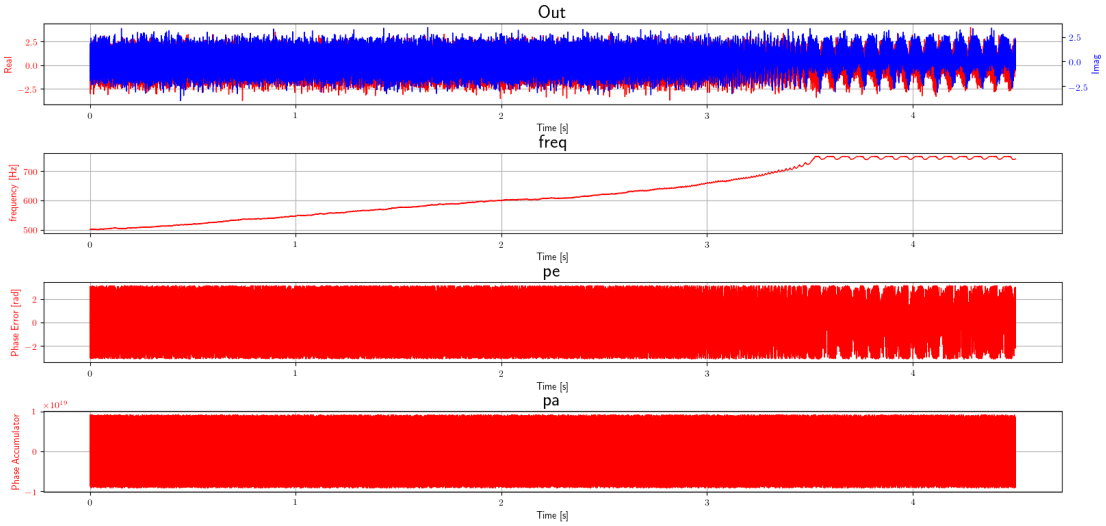

Figure 6.53: PLL outputs

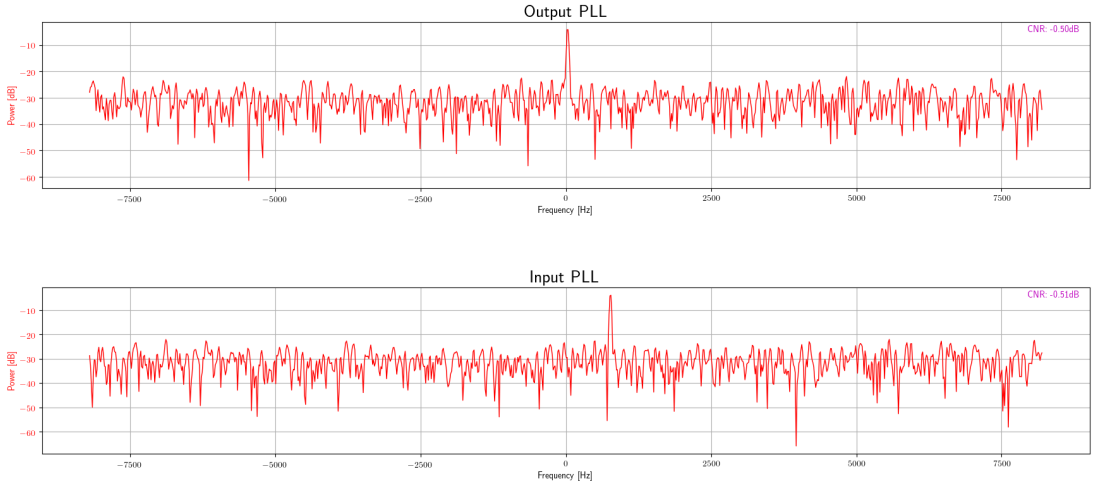

Figure 6.54: FFT output and input PLL

6.12.10 Switch from 2nd to 3rd order with noise

In this test it is checked that the PLL is able to change from the second to third order and that it is able to lock again with a noisy signal.

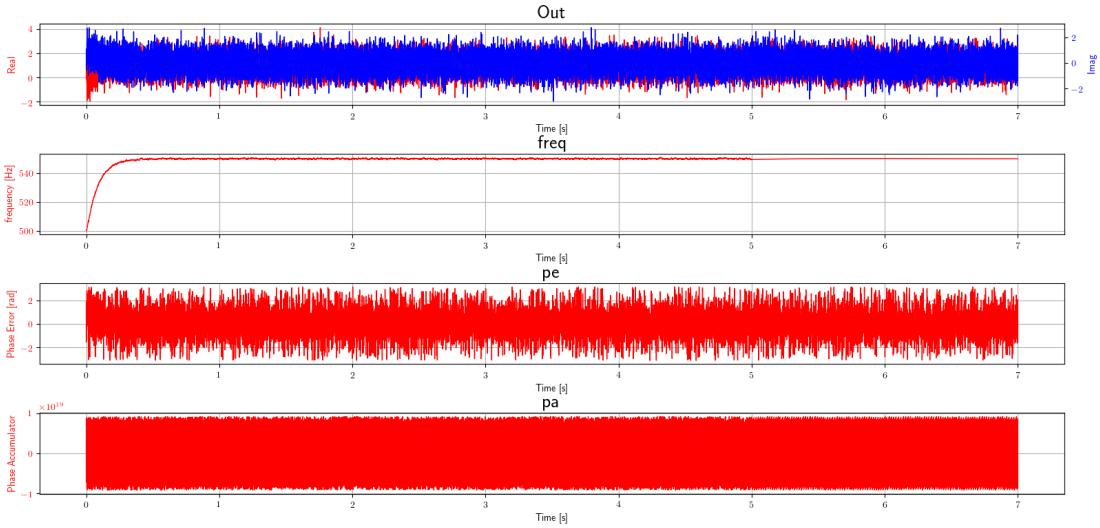

Figure 6.55: PLL outputs

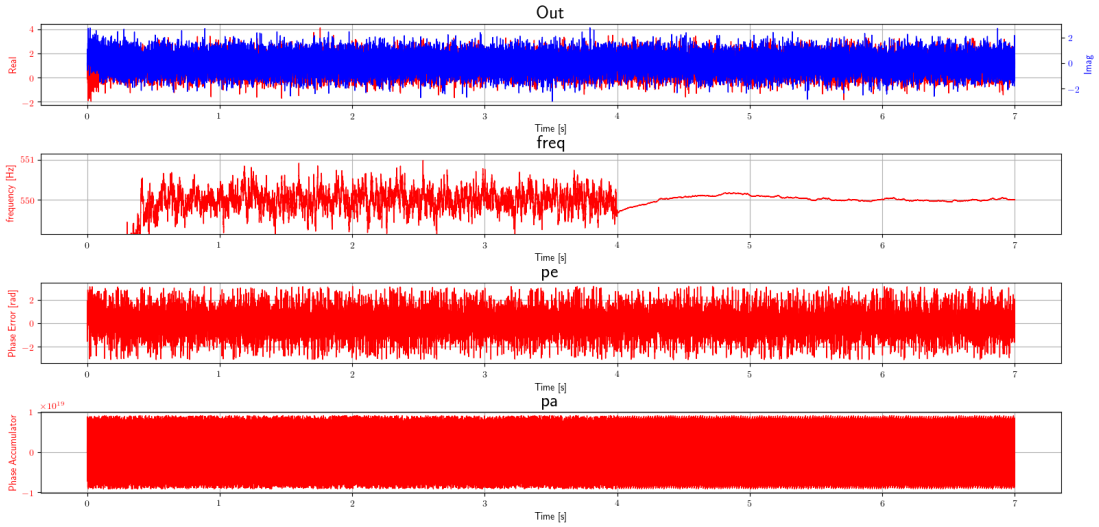

Figure 6.56: PLL outputs zoomed

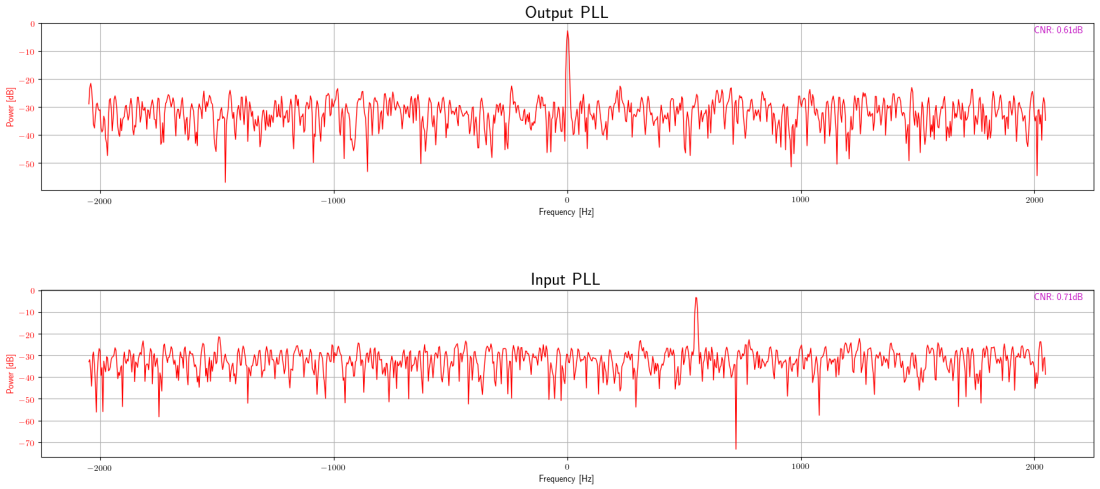

Figure 6.57: FFT output and input PLL

6.12.11 Frequency sweep with noise

In this test it is checked that the PLL is lock when the input noisy signal is sweeping.

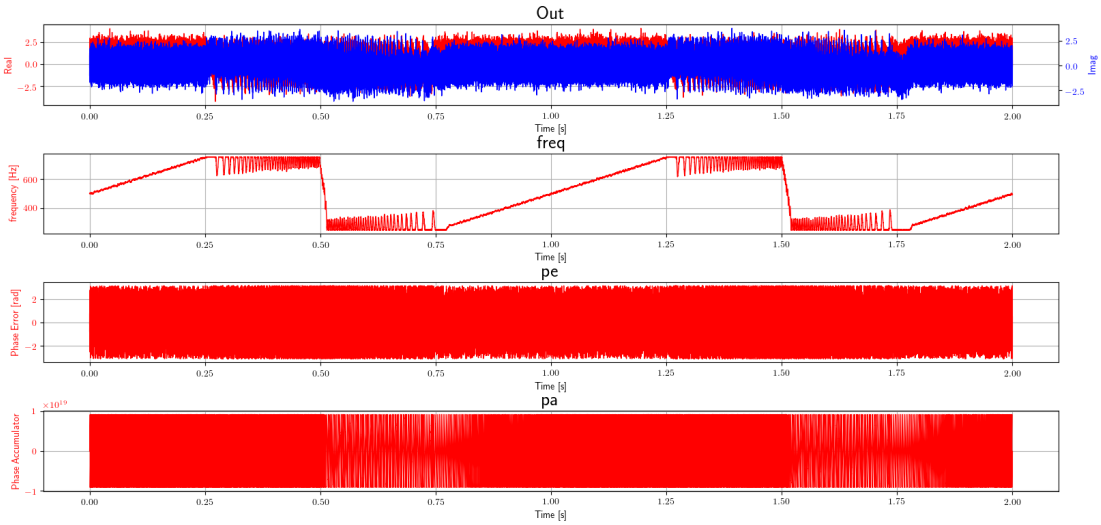

Figure 6.58: PLL outputs

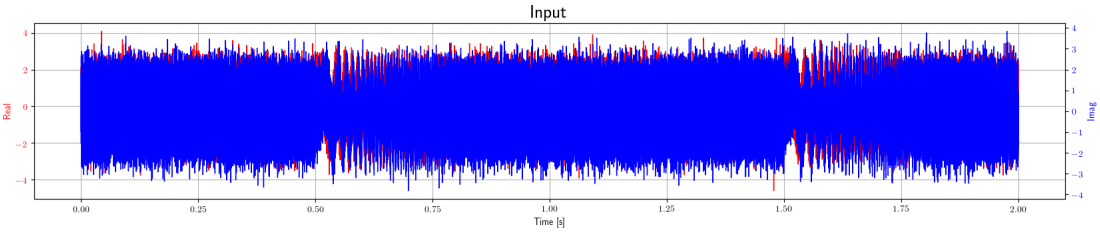

Figure 6.59: PLL input

6.13 Selector

Since the Selector block can support as input and output each type, each test should to be repeated for each type supported. Practically, that is performed only for the first test, to avoid a too long simulation.

6.13.1 Multiplexer with 2 inputs

In this test, the selector is set in order to have a behaviour of a multiplexer with 2 inputs.

For this test two signal sources have been used for both the inputs of Selector block. The two signal sources are both set to output a ramp of values, but the two ramp are different and not overlapped each others. Thus, it this way it is possible to have a unique value for each instant of the simulation for both the inputs, in order to recognize at output. Additionally, the selector is switched in the middle of simulation, and, as for the equivalent test of PLL, the Debug Switch block has been used.
Finally, it is checked if the switching occurred, and if the output of Selector is made with both the data inputs. Thus, it is checked that no one item is lost.

6.13.2 Multiplexer with 3 inputs

This test is performed as the first one, but with 3 inputs.

6.13.3 De-multiplexer with 2 outputs

This test is performed as the first one, but set to be a demodulator with 2 outputs.

6.13.4 De-multiplexer with 3 outputs

This test is performed as the first one, but set to be a demodulator with 3 outputs.

6.13.5 Switching

This test is performed as the first one, but set to be a switcher, so with only one input and one output. Thus, that effectively there is a loosing of items when set as off.

6.13.6 Tagged streams

In this the behavior with tagged streams has to be checked. Two signal source with two set of tags are connected as input of the Selector. The Selector is set in order to use only the first input. Thus, is checked that at the output are present only the tag sent on the first input.

6.14 Signal to Noise Estimator

This block is able to estimate the signal to noise or carrier to noise ratio. The test was created by generating a signal with a Signal Generator block and adding noise with a Noise Generator block. The signal with noise has been inserted as input to the SNR Estimator block. Furthermore, the signal was filtered by a band pass filter, as well as the noise generated. So, the SNR was calculated by making the ratio of the variances of the filtered signal and the filtered noise. Unfortunately, for both filters can not set a null transition band, so the measurement of Signal to Noise Estimator block is considered correct if there is a small difference with the calculated one.

6.14.1 CNR with fixed noise bandwidth

In this case only the noise was filtered with the same band set on the Signal to Noise Estimator block. The Signal to Noise Estimator block has been set up to calculate the CNR with the self carrier search.

6.14.2 CNR with all spectrum

In this case both the noise and the signal were not filtered. The Signal to Noise Estimator block has been set up to calculate the CNR with the self carrier search and all spectrum.

6.14.3 SNR with fixed noise bandwidth

In this case both the noise and the signal were filtered with the same bands set on the Signal to Noise Estimator block. The Signal to Noise Estimator block has been set up to calculate the SNR with the self carrier search.

6.15 Signal Search FFT

For this Signal Search FFT block, it is necessary to check that when a signal is present at the input with a signal to noise ratio greater than the one set, he actually copied the same data present at the input to the output. Since this block does not emit any data if the input signal does not satisfy the requirements, the number of output samples must also be checked. Finally, the block also takes care of creating an output tag every time it finds a signal that satisfies the requirements.
All tests were created using a simple input signal generator at Signal Search FFT block.

6.15.1 Input sine without noise in the Bandwidth

In this test a signal was generated at the center of the band to be searched. It was therefore checked that the output signal is exactly the same as the input signal both as values and as the number of samples. Furthermore, it has been checked that only a reset tag has been generated and that it is actually the expected tag.

6.15.2 Input sine without noise on border of Bandwidth

In this test a signal was generated at the border of the band to be searched. It was therefore checked that the output signal is exactly the same as the input signal both as values and as the number of samples. Furthermore, it has been checked that only a reset tag has been generated and that it is actually the expected tag.

6.15.3 Input sine without noise outside the Bandwidth

In this test a signal was generated outside of the band to be searched. It was therefore checked that there are no output data and that no tags have been generated.

6.15.4 Input sine with noise in the Bandwidth

In this test a signal with noise was generated at the center of the band to be searched. It was therefore checked that the output signal is exactly the same as the input signal both as values and as the number of samples. Furthermore, it has been checked that only a reset tag has been generated and that it is actually the expected tag.

6.16 Signal Search Goertzel

For this Signal Search Goertzel block, it is necessary to check that when a signal is present at the input with a signal to noise ratio greater than the one set, he actually copied the same data present at the input to the output. Since this block does not emit any data if the input signal does not satisfy the requirements, the number of output samples must also be checked. Finally, the block also takes care of creating an output tag every time it finds a signal that satisfies the requirements.
All tests were created using a simple input signal generator at Signal Search Goertzel block.

6.16.1 Input sine without noise in the Bandwidth

In this test a signal was generated at the centre of the band to be searched. It was therefore checked that the output signal is exactly the same as the input signal both as values and as the number of samples. Furthermore, it has been checked that only a reset tag has been generated and that it is actually the expected tag.

6.16.2 Input sine without noise on border of Bandwidth

In this test a signal was generated at the border of the band to be searched. It was therefore checked that the output signal is exactly the same as the input signal both as values and as the number of samples. Furthermore, it has been checked that only a reset tag has been generated and that it is actually the expected tag.

6.16.3 Input sine without noise outside the Bandwidth

In this test a signal was generated outside of the band to be searched. It was therefore checked that there are no output data and that no tags have been generated.

6.16.4 Input sine with noise in the Bandwidth

In this test a signal with noise was generated at the center of the band to be searched. It was therefore checked that the output signal is exactly the same as the input signal both as values and as the number of samples. Furthermore, it has been checked that only a reset tag has been generated and that it is actually the expected tag.

6.17 Type Casting

As said in Section 5.10, three blocks that perform a type cast are made:

- Float To Double;
- Float To Int64;
- Int To Int64.

for all of them, the tests are the same, but, obviously, with the proper type. The tests performed are the same performed by GNU Radio for its blocks for type casting.

6.17.1 Data set source

A simple data set of values is used as input, generated as values in a fixed range. The output, shall to be the same. In the "Float To Int64", the expected result are generated in the same way, but as integer and not as float as for the inputs.
Finally, the are checked if equals or not.

6.17.2 Vectors

This test is performed as the previous one, but with the only difference that the data are managed as vectors.

6.17.3 Tagged streams

This test is performed as the first one, but tags are added at the data set.
Finally, it is checked that the same data sent are present after the data casting.

6.17.4 Set_data

This test is performed as the first one, but it is checked that the set_data function of source works properly.

6.17.5 Set_repeat

This test is performed as the first one, but the repeat of source is enabled.

6.18 Vector Sink and Source

This tests are performed as GNU Radio does with the Null Sink and Source blocks. The tests consist of connect the Vector Source to the Vector Sink directly, and check if the data are the same. The same flow graph is used for the following types:

- Int64;
- Double.

6.18.1 Data set source

A simple data set of values is used as input, generated as values in a fixed range.

6.18.2 Vectors

This test is performed as the previous one, but with the only difference that the data are managed as vectors.

6.18.3 Tagged streams

This test is performed as the first one, but tags are added at the data set.

6.18.4 Repeat works with tagged streams

This test is performed as the first one, but tags are added at the data set and repeat is enabled.

6.18.5 Set_data

This test is performed as the first one, but it is checked that the set_data function of source works properly.

6.18.6 Set_repeat

This test is performed as the first one, but the repeat of source is enabled.

Chapter 7

Validation Tests

In this chapter some validation tests have been carried out in order to verify the correct functioning of the complete system. The three main functions of the TT&C Transponder have been tested separately to state its correct behaviour.

7.1 Turn-around accuracy

As previously described, one of the main design challenges is the accuracy of the turn around ratio. As a preliminary test, it was decided to use a USRP connected via USB to a PC on which GNU Radio was present. Through a specific flow graph, shown in Figure 7.1. In fact, although a hardware experimentation is not required for this project, the USRP is an excellent development and testing environment for Software Defined Radio implementation.

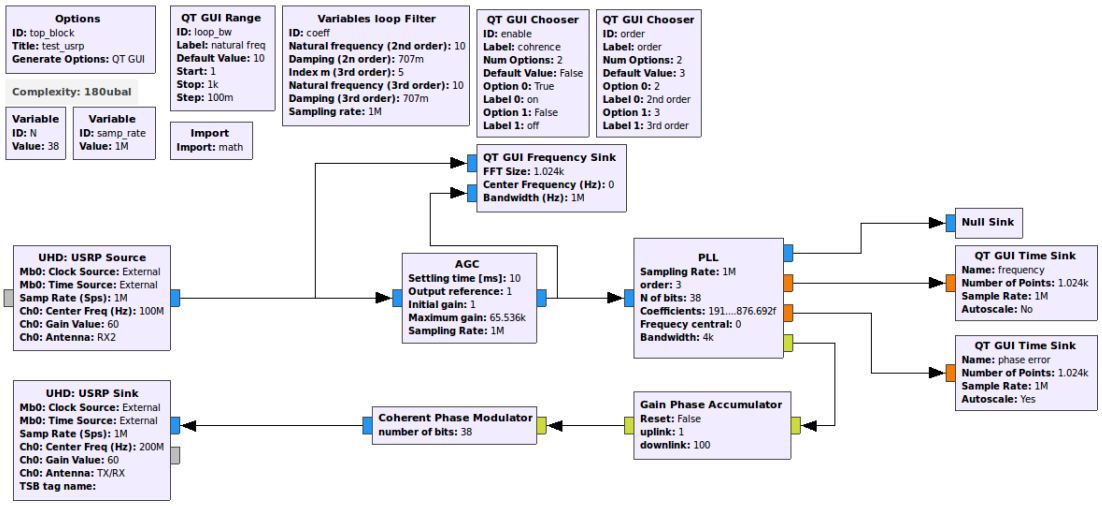

Figure 7.1: Turn around ratio preliminary test flow graph

In this test a USRP B200 was used, which has two ports. one used as TX and one as RX. As you can see from the Figure 7.2 a signal generator was used to generate a carrier at a precise frequency. In addition, a spectrum analyzer was used to analyse the output of the USRP. In order to increase measurement accuracy, a 10 MHz clock was shared on all three devices.

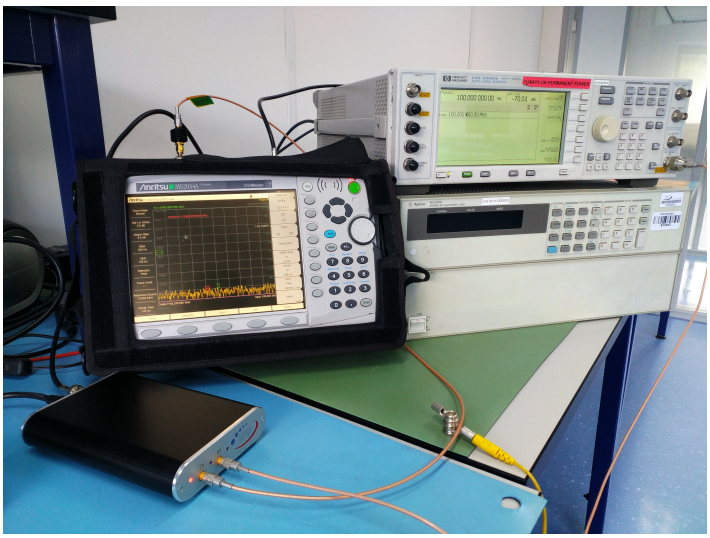

Figure 7.2: Turn around ratio preliminary test setup

To this end, a sufficiently high TAR was set up to allow the frequency translation to be appreciated. A TAR equal to 100 has been chosen. Through the signal generator several carriers have been generated at different frequency values. While the measurements were made through the spectrum analyzer. It is important to underline how the measurements carried out by the latter instrument are affected by poor accuracy, but still sufficient to give an idea of how it works.

7.2 Tracking

For this validation test, the European Space Agency provided three files containing three sample signal recordings that are normally used. These files are composed of two phases, the first is composed of a frequency sweep of 150 kHz at three speeds for the three different files: 10 kHz/s, 20 kHz/s and 32 kHz/s. The second corresponds to a modulated signal. In this test only the first part is important.

This frequency sweep is used for a specific reason. In fact, between the ground station and the satellite there will be a certain relative speed. This difference translates into a frequency offset between the frequency of the signal sent from the ground station and that received from the satellite. In fact, one of the objectives of this transponder, as already mentioned, is to allow the tracking of the satellite, precisely through the accurate measurement of Doppler and its variation over time. Moreover, in order for the PLL to be able to lock the input signal, it is necessary to have a very narrow noise bandwidth in order to keep the CNR under control. Therefore, a carrier frequency sweep is required in order to be sure to send, at least for a short period, a carrier that is within the PLL band. Obviously, this means that the faster the sweep, the less time it will be in the PLL band. This means that the transponder must be able to identify if a signal is present at the input (through the signal search) and to lock the signal in a time in a time less than the transition time of the signal in the useful band.

7.2.1 Signal detection

The main thing to test is if the signal search designed is able to detect the signal when it is into the set bandwidth. To evaluate that, the flow graph shown in Figure 7.3 has been made.

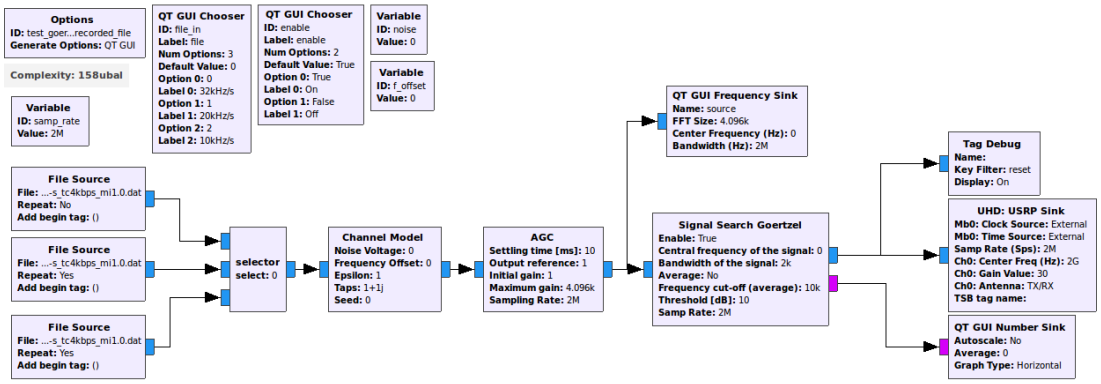

Figure 7.3: Goertzel Signal Search output with the source file with a frequency sweep flow graph

With the same setup shown in Figure 7.2, the output of the USRP has been checked on the spectrum analyzer. Actually, the signal search is as a filter of the input, so only if the input satisfy the set parameters, it will be outputted. Thus, by exploiting the "max-hold" of a spectrum analyzer, the following Figures have been made. The Signal Search was set with a threshold of 10 dB and a bandwidth of 2 kHz.

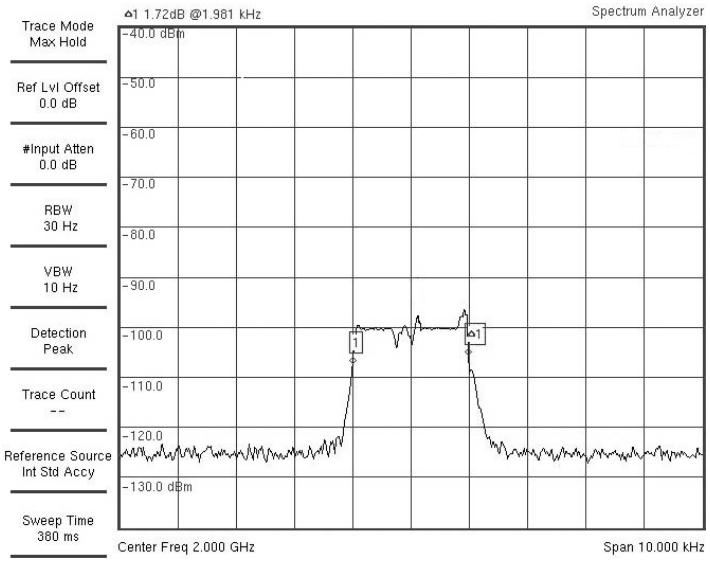

Figure 7.4: Goertzel Signal Search output with the source file with a frequency sweep at 10 kHz/s

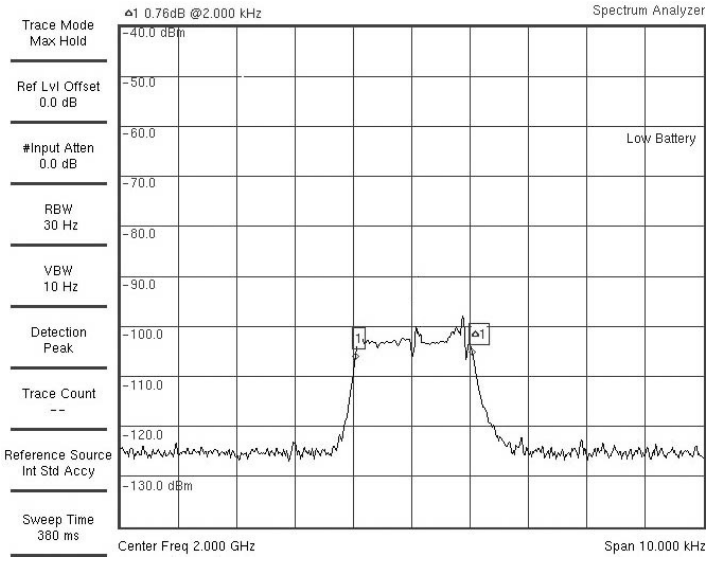

Figure 7.5: Goertzel Signal Search output with the source file with a frequency sweep at 20 kHz/s

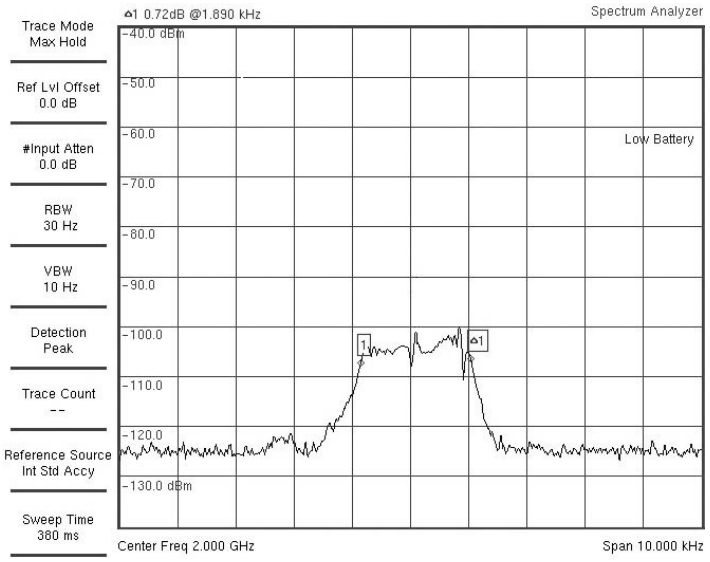

Figure 7.6: Goertzel Signal Search output with the source file with a frequency sweep at 32 kHz/s

As it is possible to see from the previous figures, the range of frequencies acquired by the spectrum analyzer corresponds to the bandwidth set on the Signal Search block.

7.2.1.1 Signal detection with noise and Doppler

The same flow graph shown in Figure 7.3, but with the Channel Model block set in order to get a CNR of 52.8 $dBHz$ and a frequency offset of -2 kHz.

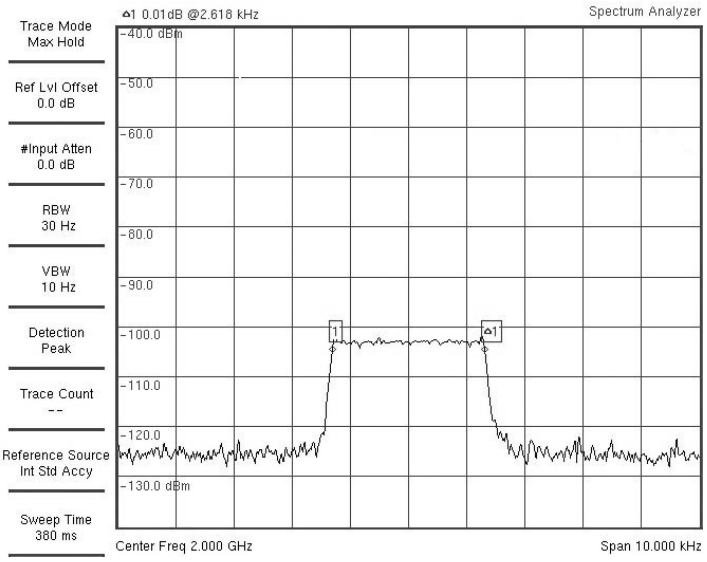

Figure 7.7: Goertzel Signal Search output with the source file with a frequency sweep at 10 kHz/s with noise and Doppler

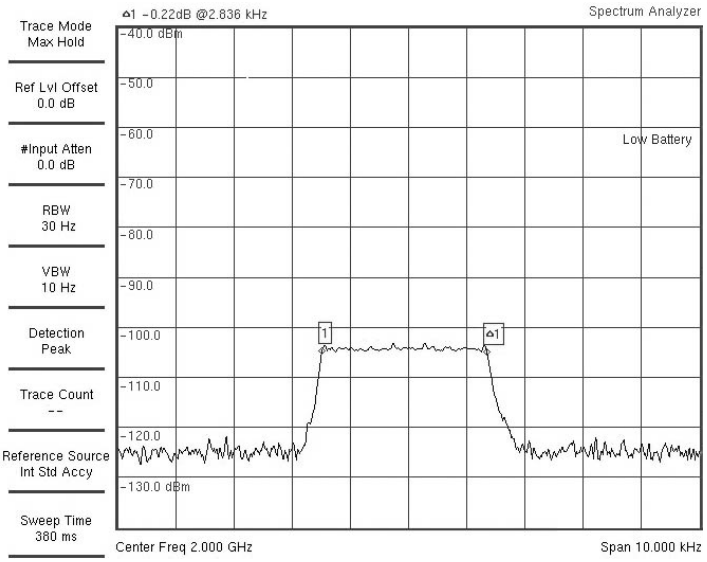

Figure 7.8: Goertzel Signal Search output with the source file with a frequency sweep at 20 kHz/s with noise and Doppler

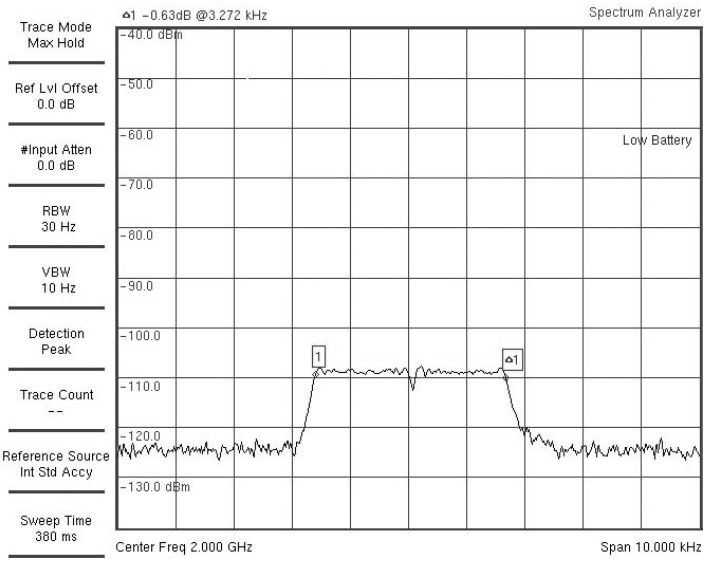

Figure 7.9: Goertzel Signal Search output with the source file with a frequency sweep at 32 kHz/s with noise and Doppler

As it is possible to see from the previous figures, the frequency range acquired by the spectrum analyzer corresponds roughly to the bandwidth set in the signal search block. Obviously, the added noise worsens the accuracy of the block. Unfortunately, the Signal Search with the Goertzel Algorithm has difficult to handle properly an higher value of noise. On the other hand, the Signal Search with the FFT Algorithm does not have that problem. Moreover, it is important to set the parameters in order to get a less delay of the block due to complex mathematical calculations. In this case, it was set with a bandwidth of 2 kHz, a threshold of 10 dB, decimation of 6 and the FFT size of 2048. with the FFT Algorithm, a CNR of 43 $dBHz$ has been used.

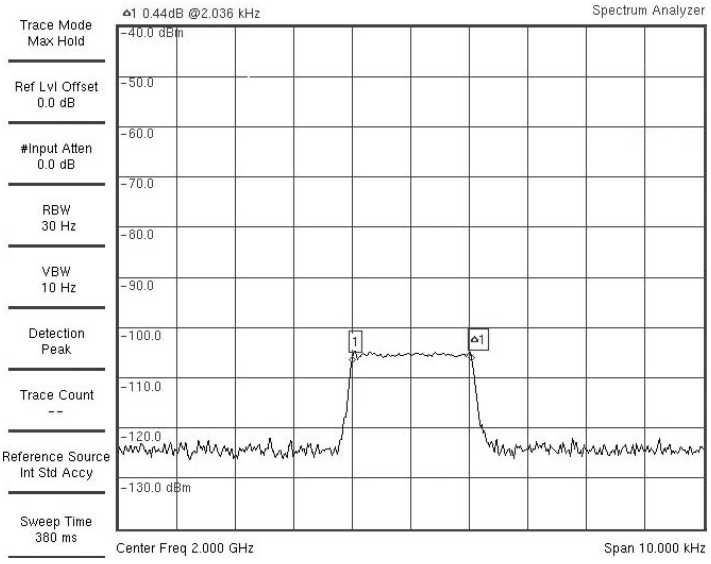

Figure 7.10: FFT Signal Search output with the source file with a frequency sweep at 10 kHz/s with CNR of 50.2 $dBHz$ and Doppler

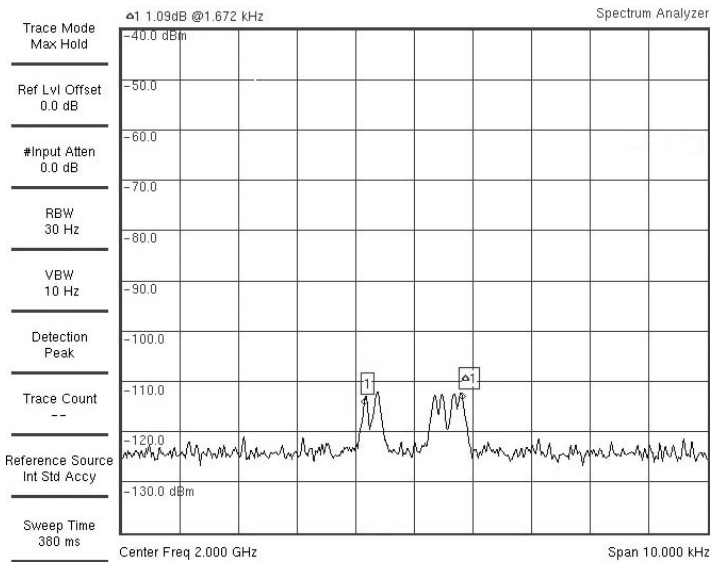

Figure 7.11: FFT Signal Search output with the source file with a frequency sweep at 10 kHz/s with CNR of 43 $dBHz$ and Doppler

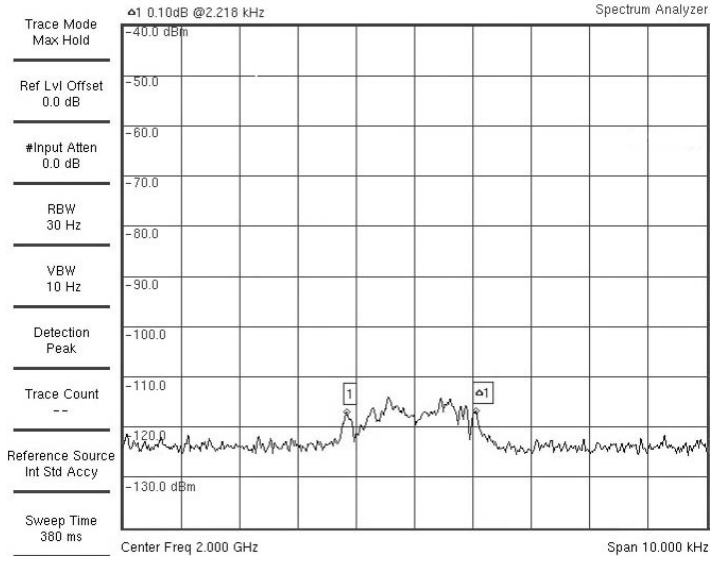

Figure 7.12: FFT Signal Search output with the source file with a frequency sweep at 32 kHz/s with CNR of 43 $dBHz$ and Doppler

The range of frequencies acquired by the spectrum analyzer corresponds to the bandwidth set on the Signal Search block only with a frequency sweep of 10 kHz/s, as it is possible to see in Figure 7.10. Moreover, with a higher noisy frequency sweep, the frequency range is more distorted, as it is possible to see in Figures 7.11 and 7.12. It is however possible to state that the system is able to correctly recognize the desired signal, albeit less effectively.

7.2.2 Transponder in coherent mode

The main thing to test is if the PLL is able to lock after the Signal Search block. To evaluate that, the flow graph shown in Figure 7.13 has been made.

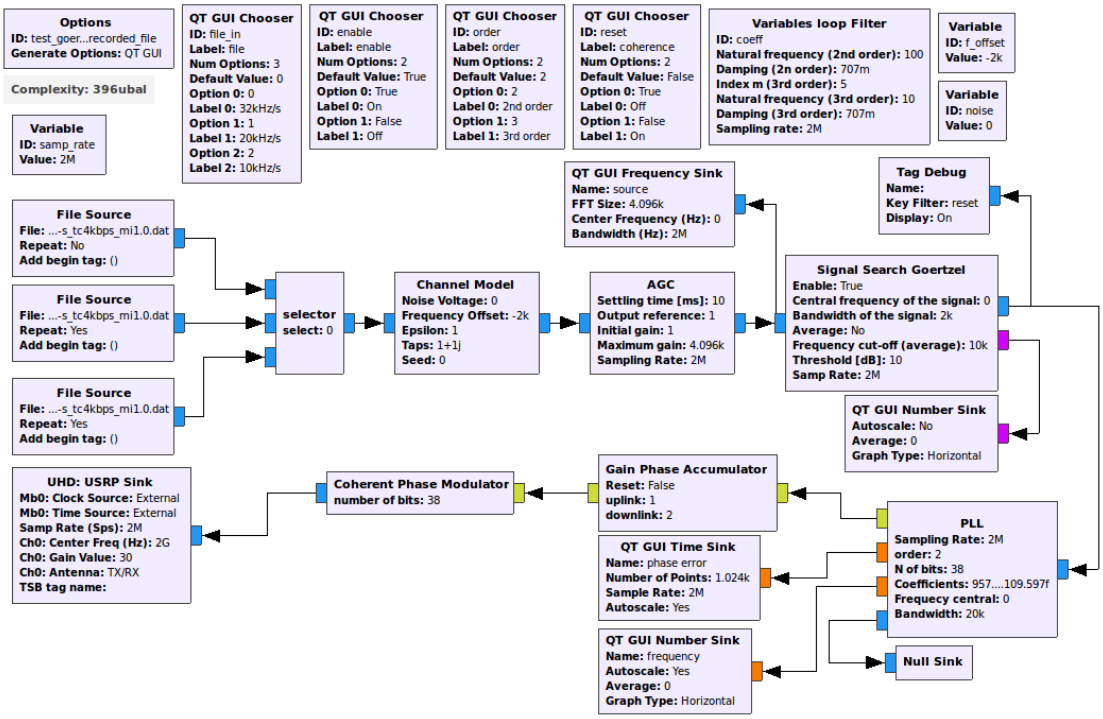

Figure 7.13: Turn around ratio with the source file with a frequency sweep test setup

With the same setup shown in Figure 7.2, the output of the USRP has been checked on the spectrum analyzer. Actually, the signal search is as a filter of the input, so only if the input satisfy the set parameters, it will be outputted. Thus, by exploiting the "hold maximums" of the spectrum analyzer, the following Figures have been made. The Signal Search was set with a threshold of 10 dB and a bandwidth of 2 kHz. Finally, a Doppler of 2 kHz has been set in order to simulate properly the PLL, since at start, the file, is for some seconds at 0 Hz, so the PLL will lock it immediately.

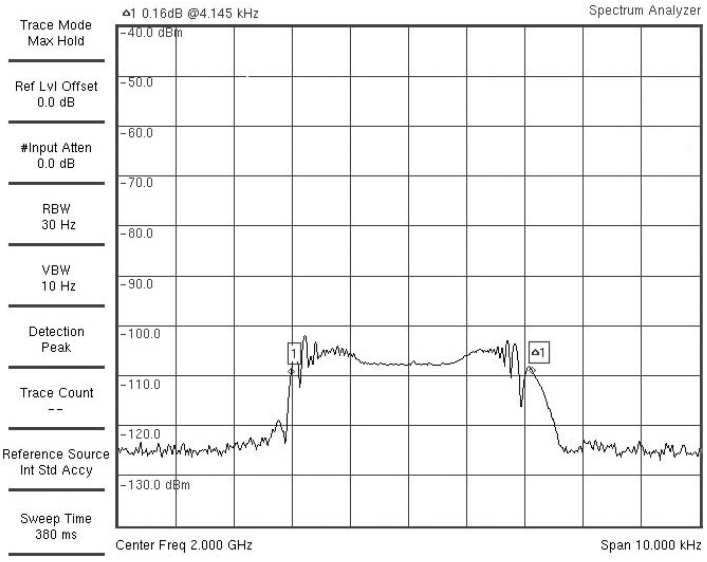

Figure 7.14: Transponder in coherent mode output with the source file with a frequency sweep at 10 kHz/s

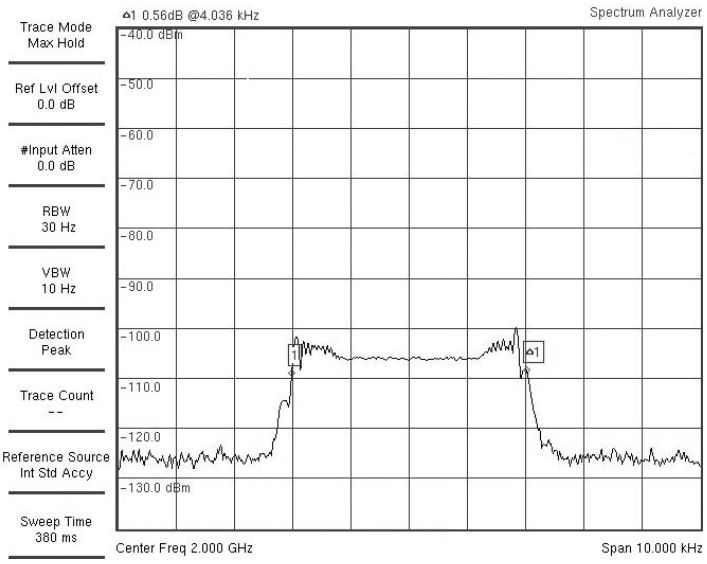

Figure 7.15: Transponder in coherent mode output with the source file with a frequency sweep at 20 kHz/s

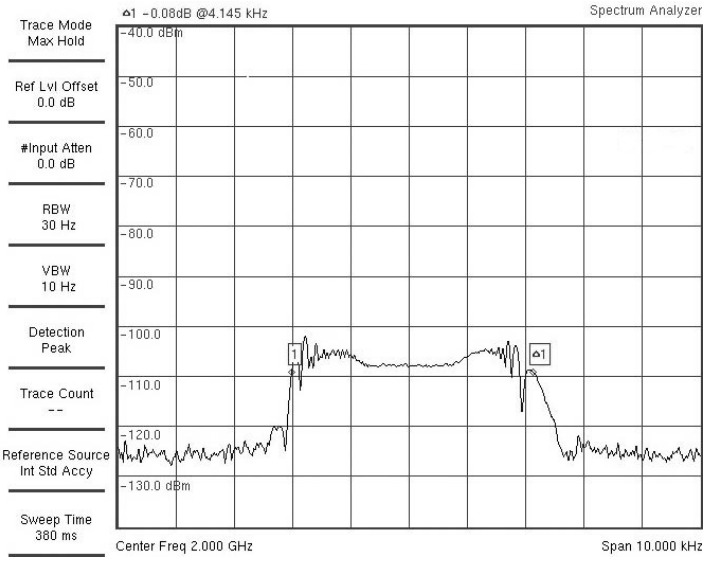

Figure 7.16: Transponder in coherent mode output with the source file with a frequency sweep at 32 kHz/s

It is important to notice that the Turn-Around Ratio set is 2, so the range of frequencies acquired by the spectrum analyzer is approximately double since the PLL has been locked correctly. As it is possible to see from the previous figures, the range of frequencies acquired by the spectrum analyzer corresponds to the double of bandwidth set on the Signal Search block.

7.2.3 Transponder in coherent mode with noise and Doppler

The same flow graph shown in Figure 7.13, but with the Channel Model block set in order to perform add a CNR of 57.1 $dBHz$.

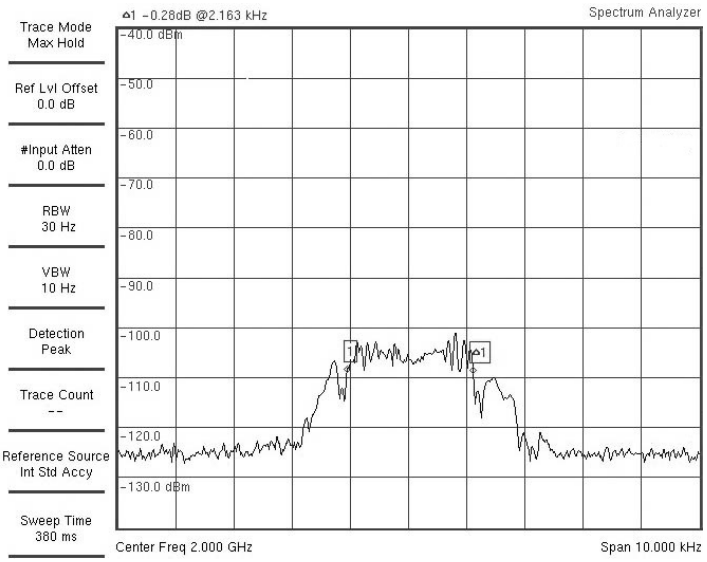

Figure 7.17: Transponder with Goertzel in coherent mode output with the source file with a frequency sweep at 10 kHz/s with CNR of 57.1 $dBHz$

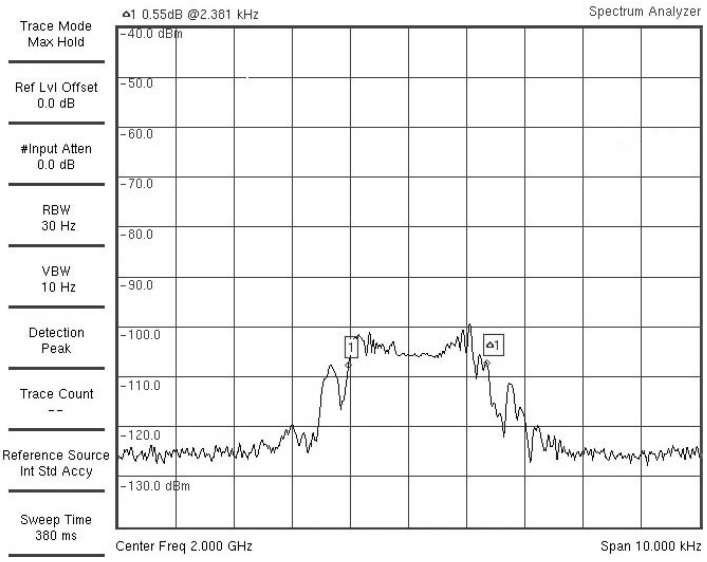

Figure 7.18: Transponder with Goertzel in coherent mode output with the source file with a frequency sweep at 20 kHz/s with CNR of 57.1 $dBHz$

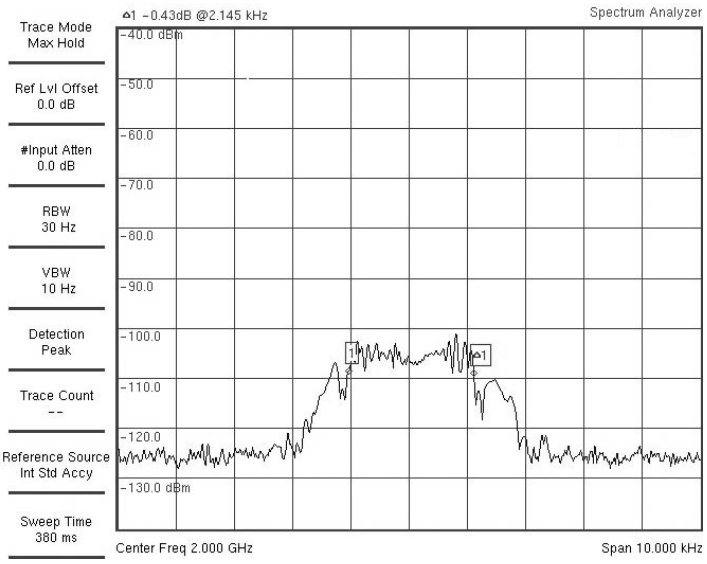

Figure 7.19: Transponder with Goertzel in coherent mode output with the source file with a frequency sweep at 32 kHz/s with CNR of 57.1 $dBHz$

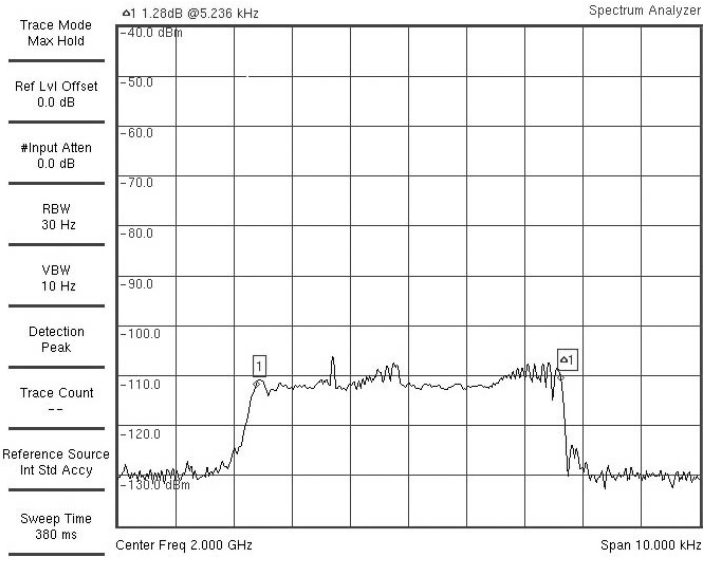

Figure 7.20: Transponder with FFT in coherent mode output with the source file with a frequency sweep at 10 kHz/s with CNR of 41 $dBHz$

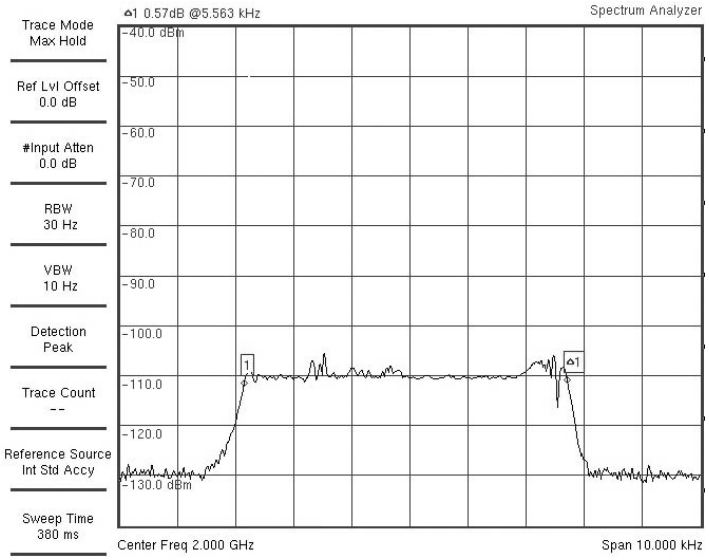

Figure 7.21: Transponder with FFT in coherent mode output with the source file with a frequency sweep at 20 kHz/s with CNR of 41 $dBHz$

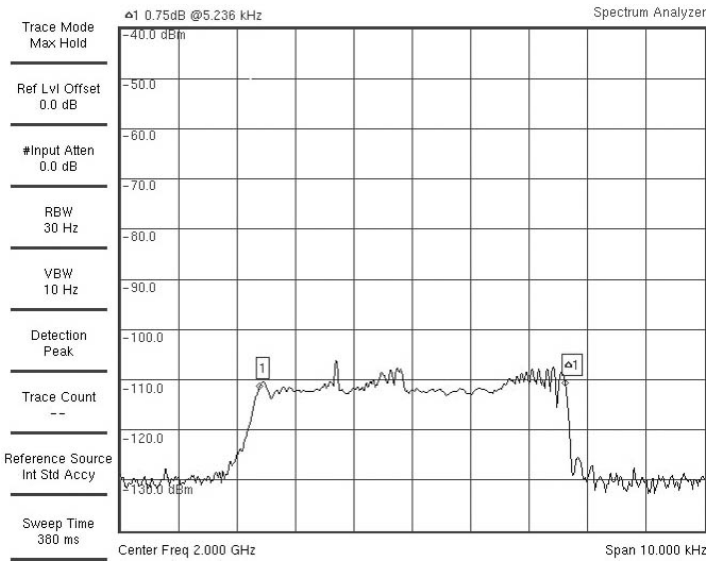

Figure 7.22: Transponder with FFT in coherent mode output with the source file with a frequency sweep at 32 kHz/s with CNR of 41 $dBHz$

It is important to notice that the Turn-Around Ratio set is 2, so the range of frequencies acquired by the spectrum analyzer is approximately double since the PLL has been locked correctly. As it is possible to see from the previous figures, the range of frequencies acquired by the spectrum analyzer corresponds to the double of bandwidth set on the Signal Search block.
Moreover, it can be noticed by the previous figures, that the solution with the FFT seems to allow a more effective lock in a noisier environment than the Goertzel solution. Unfortunately, these tests are not sufficient to discriminate one solution over the other. Also because the shapes of the acquired signals are artefacts created by the spectrum analyzer through mathematical averages, that depend on the speed of the signal and of the instrument. So, for example, the side part in Figure 7.17 not steep can be justified by the fact that the acquired signal spends less time in that area than the central

one. While, the peaks present in the previous figures as in Figure 7.6, are due to the reason exactly opposite to the preceding.

7.3 Telemetry

One of the functions that the designed transponder must have is to send the telemetry in downlink. Telemetry consists in sending data through downlink modulation of the carrier. To test this function the USRP B200 has been used again, this time connected with two modules that are generally used in ground stations by ISIS and ESA. The latter are a programmable mixer that down-shiftes to 70 MHz the incoming signal, which in turn is connected to a modem. The modem takes care of demodulating a signal in intermediate frequencies, ie around 70 MHz. They are shown in Figure 7.23.

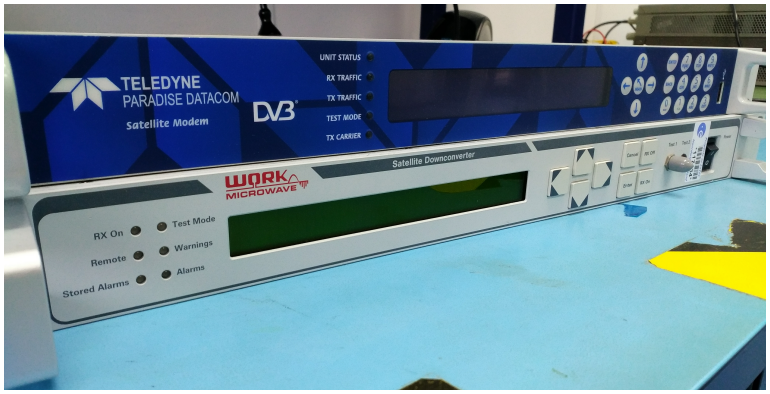

Figure 7.23: Programmable mixer and modem

In Figure 7.24 the flow graph used for the test is shown. The sent data have been the same used in the Modulator block test in Section 6.7. Finally, a modulation index equal to one has been chosen.

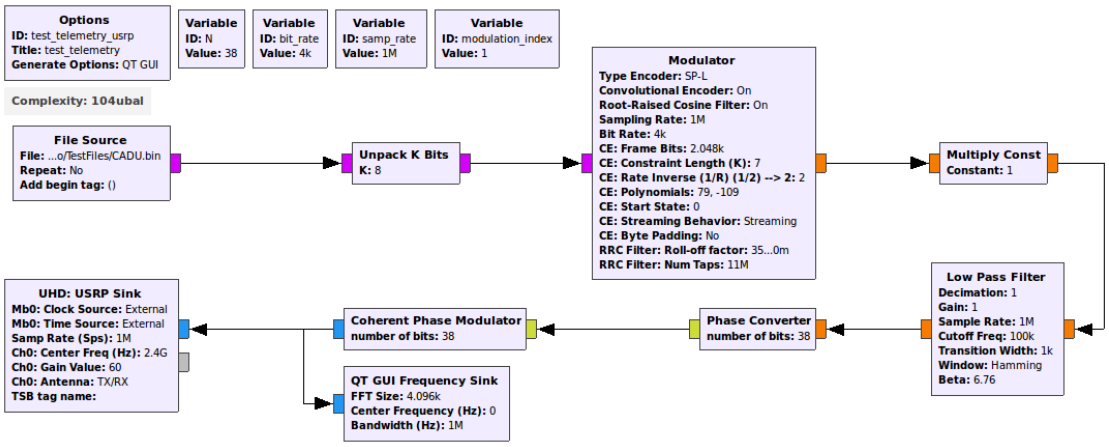

Figure 7.24: Telemetry test flow graph

Several types of modulations have been tested:

- SP-L modulation directly on the carrier;
- NRZ-L modulation directly on the carrier;
- NRZ-L modulation with an 16 kHz sinewave sub-carrier;

- NRZ-L modulation with a 300 kHz sinewave sub-carrier;

The bit rate used for all tests is 60 *ksps*. both the Convolutional Encoder and the SRRC filter were used.

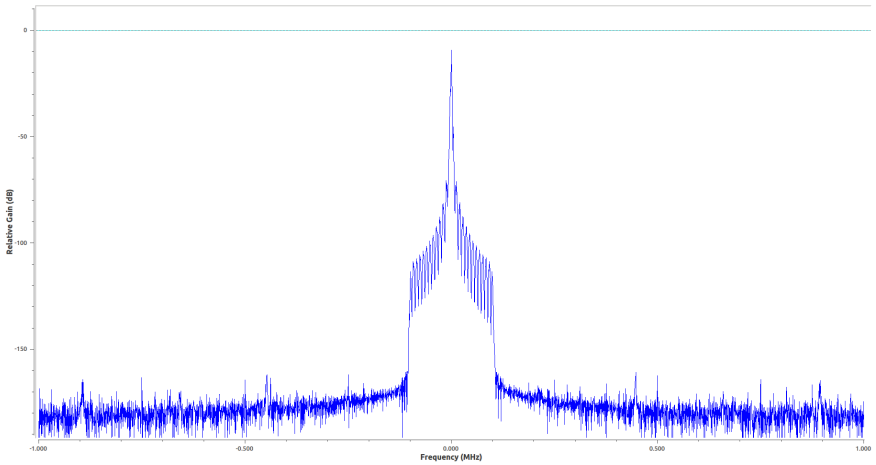

Figure 7.25: Spectrum of generated signal with SP-L modulation

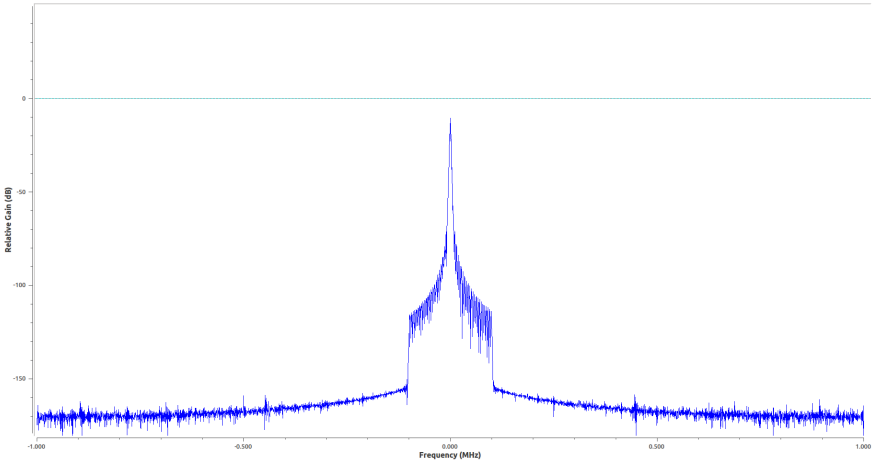

Figure 7.26: Spectrum of generated signal with NZR-L modulation

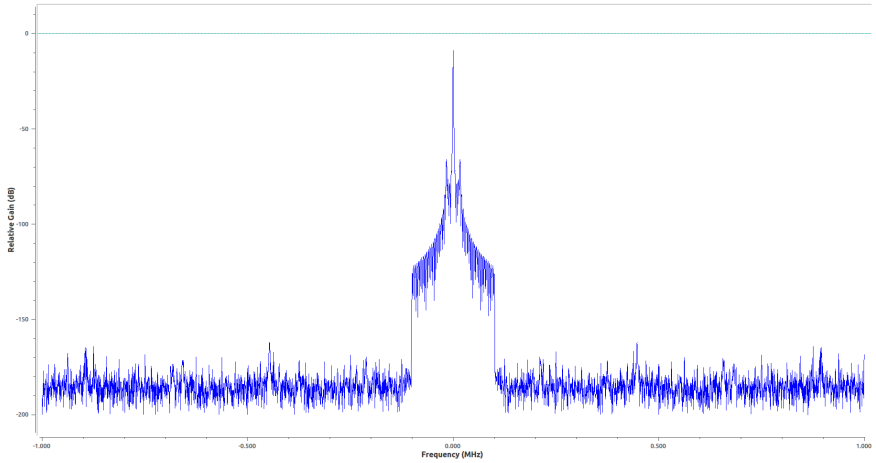

Figure 7.27: Spectrum of generated signal with NZR-L modulation with a sub-carrier at 16 kHz

The extrapolated data was saved on a file and compared to the expected data. Since they are equal the test can be declared passed.

7.4 Commands

In the first part of the test the part used was used to send the telemetry to modulate the data which will then be demodulated properly with the part of the demodulation of the commands. The flow graph used for this test is shown in Figure 7.28 During the simulation, the Modulator block has been sin order to not use either the SRRC and Convolutional Encoder, as they are not essential in this configuration. It was possible to directly connect the output of Coherent Phase Modulator block with the input of AGC block because the TT&C Transponder, and therefore all its blocks, works in Zero-IF. However, in order to emulate real conditions, a small noise and offset of frequency are been added, but in order to still have the carrier of signal in the PLL bandwidth. The Demodulator block has been connected directly to the Phase Error output of the PLL since in this way it is possible to directly obtain the signal modulating the input carrier. Finally, the acquired data were saved to a file. By not using the Convolutional Encoder, the demodulated data must be the same as the input data with an uncertainty of 180 degrees. The sent data have been the same used in the Modulator block test in Section 6.7.

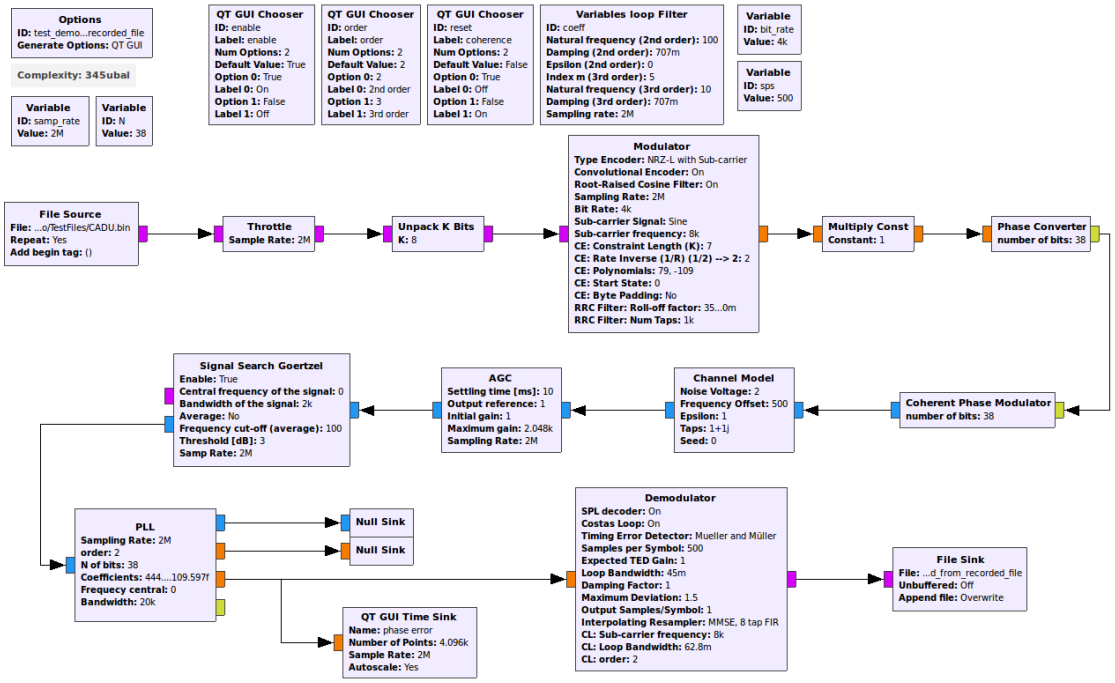

Figure 7.28: Commands and telemetry test flow graph

Several types of modulations have been tested:

- SPL modulation directly on the carrier;
- NRZ-L modulation with an 8 kHz sinewave sub-carrier;
- NRZ-L modulation with a 16 kHz sinewave sub-carrier;

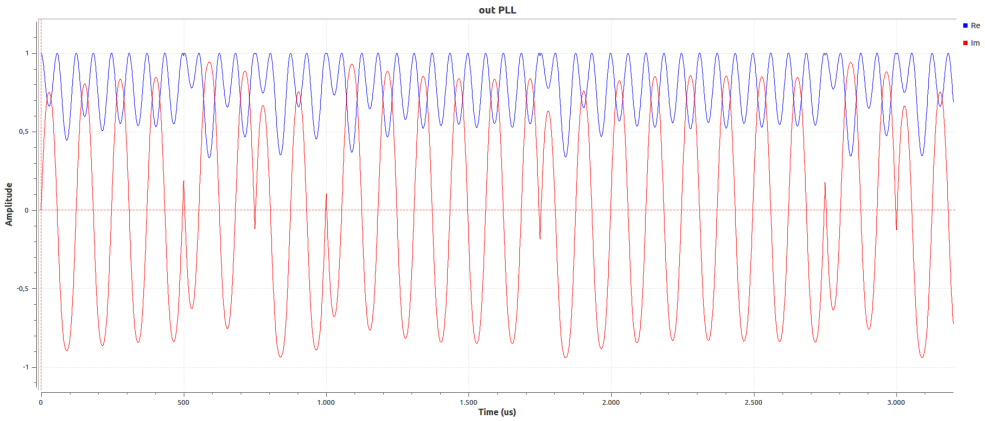

Figure 7.29: Output of PLL with a modulated signal in NRZ-L with a sub-carrier of 8 kHz

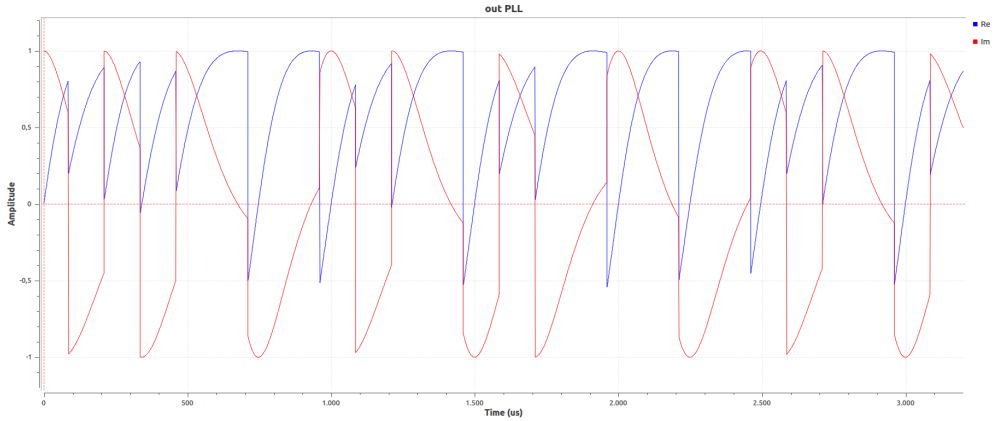

Figure 7.30: Output of PLL with a modulated signal in SP-L

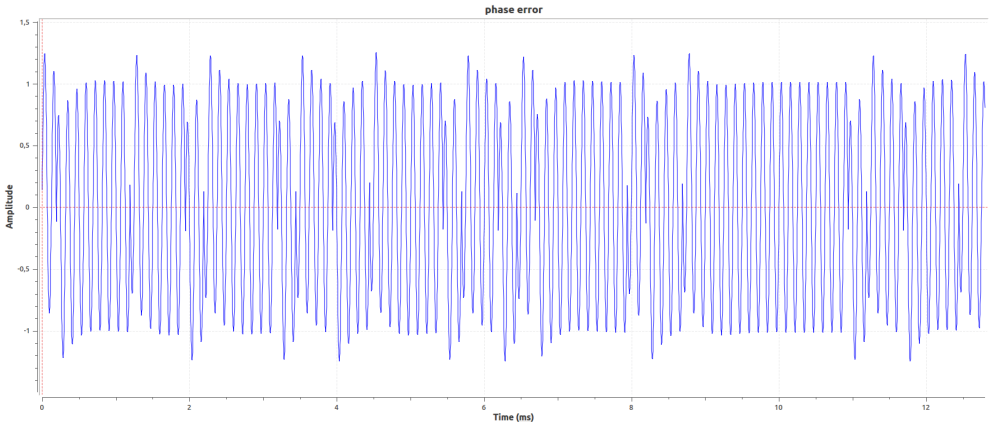

Figure 7.31: Phase error of PLL with a modulated signal in NRZ-L with a sub-carrier of 8 kHz

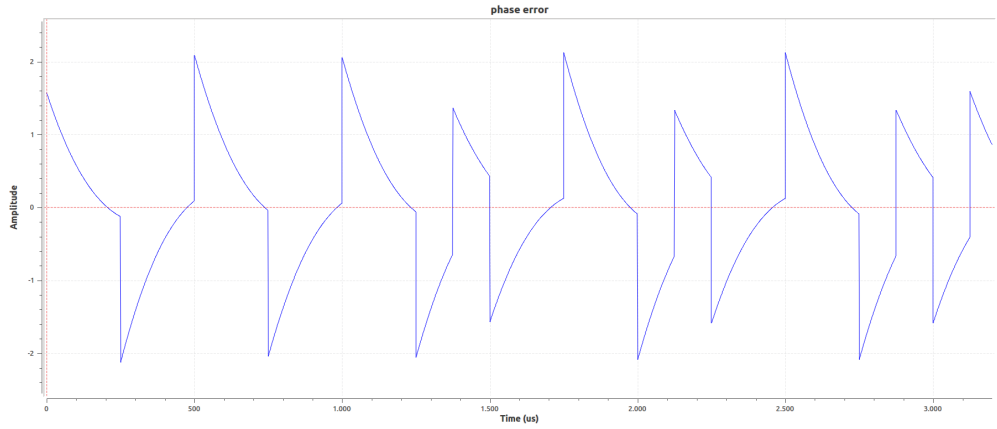

Figure 7.32: Phase error of PLL with a modulated signal in SP-L

Finally, all texts have been correctly executed.

7.4.1 ESA recorded file

As previously described in Section 7.2, the file provided by ESA contains modulated data in the second part. The latter are generally commands sent from the ground station, which will then be processed by the OBC. In this validation test it is important to verify the correctness of demodulation of the sent signal. In Figure 7.33 the flow graph used for the test is shown. The Channel Model block was used in order to emulate a possible Doppler effect and noise.

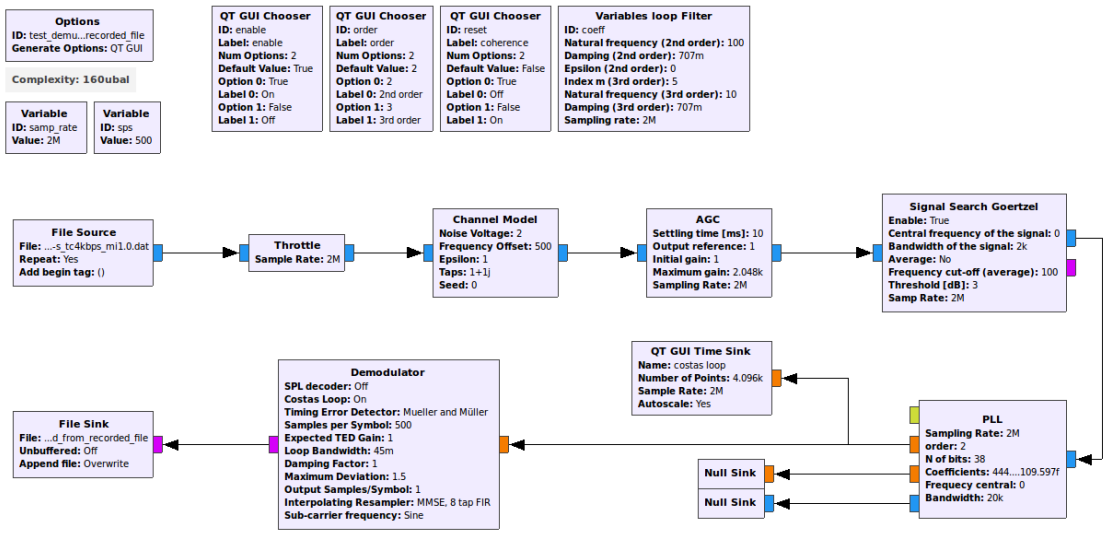

Figure 7.33: Commands test flow graph

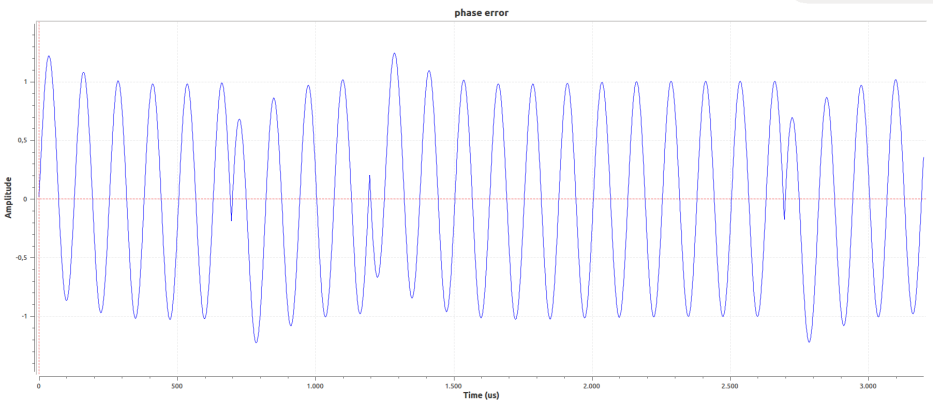

Figure 7.34: Phase error of PLL with the ESA recorded file

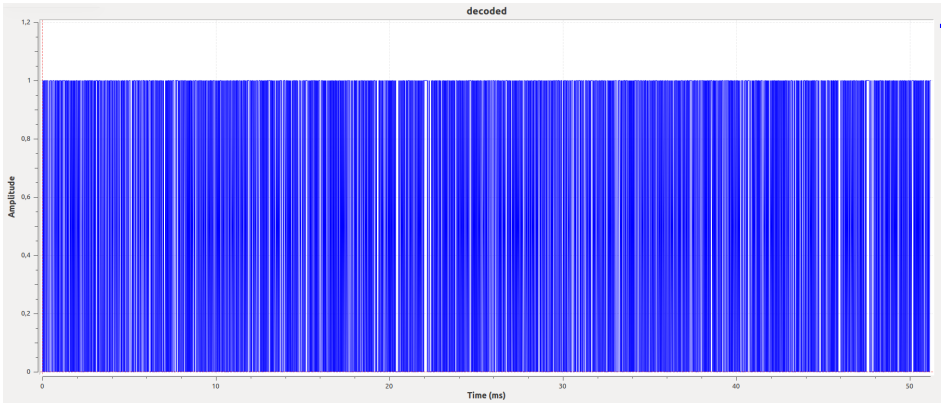

Figure 7.35: Data extrapolated from the ESA recorded file

The extrapolated data was saved on a File and compared to the expected data. Since they are equal the test can be declared passed.

Chapter 8

Conclusion

The designed software model of a TT&C Transponder has been implemented in order to be a basic version that can be used for each future space mission as well as with the Ground Stations already present. Furthermore, the combination of the chosen framework and the system implemented on it also has the function of validating new hardware and improving the knowledge and confidence in existing transponder systems by simulating real-life situations.
Since this is a project developed for ESA, several meetings have been held with ESA in which the European Space Agency has already had the opportunity to make design choices and to approve the project presented. Furthermore, the work made has been carried out mainly at the ISIS headquarters and at TU Delft.
The implemented transponder was mainly carried out in the following phases:

- In the first phase it was necessary to research and understand the state of the art of Software Designed Radio and their applications in satellite transponders. In fact, to date only NASA has developed a Software Defined Transponder that can be used in CubeSats. Furthermore, it was necessary to carry out a research activity on the ECSS standards to define the design specifications of the Transponder so as to be able to satisfy these requirements, and therefore those of ESA. Finally, based on this information, a preliminary version of the transponder was defined.

- In the second phase the ideal framework for the implementation of the project was searched and chosen. This was chosen in such a way as to allow the implementation of new software libraries, but also in order to allow future updates of the solution created (for example new types of modulations). Finally, a software called GNU Radio was chosen, which is an open source software and has considerable potential in the development of SDR.

- In the third phase the software libraries were implemented within the chosen framework. GNU Radio is organized in blocks that together conveniently form a flow graph. Therefore, it was possible to implement individually the various blocks that constitute the complete architecture of the TT&C Transponder in order to simplify both design and future improvements.

- In the fourth phase it was necessary to develop dedicated tests for each single block created in order to certify their behaviour. These tests are called QA Tests and have been developed exploiting the unittest of Python, but implementing a new report generation system in order to digitally guarantee the result of the test. In fact, each report presents check sum that certify the tests for each software library. This reporting system was necessary to guarantee the reliability of the tests produced to ESA.

- In the last phase it was decided to test the whole architecture in its overall functioning. The individual blocks were assembled to create a complete transponder and the whole system validation tests were developed. These tests have been validated with the use of certified instrumentation, such as a modem provided by ESA and used to certify systems for space use. In addition,

recorded files provided by ESA containing recordings of signals sent from Ground Stations are used to effect Tracking & Ranging and send Commands.

The validation tests carried out here are to be considered quantitative, as the system requires parts that will have to be subsequently developed. Moreover, even if the USRP is a very valid tool to test the implementation, it remains a quantitative evaluation as the transponder designed will require dedicated hardware optimized in order to meet the required accuracy.
On the other hand, the work carried out has allowed to acquire fundamental notions concerning the software implementations of a TT&C Transponder and concerning the chosen framework. In fact, GNU Radio had a determinant role in the realization of the project as it has been binding in many design choices and it has been needed to solve problems due to the scarcity of documentation and software bugs. This is due to the fact that GNU Radio is open source software, but it remains a very valid framework even considering these problems.
Finally, we can state that up to this point, the project reflects the expected behaviour, but the further verification steps for the validation of the project remain necessary.

Acknowledgements

Prima di tutto sento il dovere morale di ringraziare la mia fidanzata Francesca, che è stata una delle poche persone a credere in me fin dall'inizio, in un periodo in cui probabilmente nemmeno io sarei stato in grado di farlo. Senza di lei non solo tutto questo non sarebbe potuto accadare, ma neppur essere immaginato.

Devo inoltre ringraziare il professor Stefano Speretta che, con la sua impressionante conoscenza, professionalità e passione, è stato così gentile da aiutarmi, da vero mentore, in tutto il progetto di tesi.

Doveroso ringraziare la mia famiglia che, attraverso numerosi sacrifici, mi ha permesso di completare questo percorso.

Vorrei anche ringraziare i miei numerosissimi amici e colleghi che mi hanno sempre sostenuto ed hanno creduto in me.

Inoltre, sento il dovere di ringraziare il mio relatore Leonardo Reyneri e tutta l'azienda ISIS per avermi permesso di fare questa spendida esperienza.

Infine, non posso esimermi dal ringraziare tutti i professori che ho incontrato nel mio intero percorso accademico, che sono stati in grado di riconoscere le mie reali potenzialità e che mi hanno trasmesso la loro passione per la cultura.

Vorrei infine esprimere un mio personale pensiero. Sempre più frequentemente nella società moderna siamo abituati a sentirci obbligati ad intraprendere e completare un percorso universitario. Quest'ultimo è doveroso che sia svolto nel minor tempo possibile e con i massimi voti. Come spesso accade però, ai primi inadempimenti tendiamo a sentirci frustrati nonostante siamo tutti soggetti alla medesima tipologia di valutazione, che per ovvie ragioni, non potrà mai risultare equa nei confronti di tutti. Credo che spesso tendiamo a dimenticare l'importanza dei momenti che stiamo perdendo per adepiere, spesso senza nemmeno aver compreso appieno il motivo, a quelli che percepiamo essere i nostri doveri. Dunque, credo che noi tutti dovremmo imparare a riconoscere ed incentivare le reali capacità degli individui che incontriamo, smettendola di cercare di uniformare e di catalogare tutti nel medesimo modo. Dico questo perchè nella mia carriera scolastica ho incontrato molte difficoltà, prima su tutte il pregiudizio e, solamente grazie alle giuste persone che hanno saputo credere in me, professori compresi, sono stato in grado di superarle. Ovviamente, nel mio caso, ho avuto la fortuna di avere delle capacità che si sposassero bene con un percorso accademico di tipo ingegneristico, ma dovrebbe essere considerata come una mera coicidenza. Ognuno meriterebbe di essere scoperto e valorizzato per le sue personalissime capacità.

Bibliography

[1] P. Holsters, G. Boscagli, and E. Vassallo, "Pseudo-Noise Ranging for Future Transparent and Regenerative Channels", presented at the SpaceOps 2008. DOI: 10.2514/6.2008-3277.

[2] V. Piskorzh, V. Volokhov, M. Davydov, and M. A. Matar, "TT&C transponder for small satellites", *IEEE Xplore*, Dec. 30, 2003. DOI: 10.1109/MWSCAS.2003.1562354.

[3] (2018). Planet, [Online]. Available: https://www.planet.com/.

[4] (2018). Spire: Space to Cloud Data & Analytics, [Online]. Available: https://www.spire.com/.

[5] (2018). Sky and Space Global, [Online]. Available: https://www.skyandspace.global/.

[6] (2018). GeoOptics, [Online]. Available: www.geooptics.com/.

[7] NASA, *CubeSat101, Basic Concepts and Processes for First-Time CubeSat Developers.* Oct. 2017.

[8] ESA. (Nov. 15, 2017). Technology CubeSats, [Online]. Available: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Technology_CubeSats (visited on 07/22/2018).

[9] Nanosats. (2018). Nanosatellite database, [Online]. Available: https://www.nanosats.eu (visited on 09/12/2018).

[10] Wikipedia. (2018). CubeSat, [Online]. Available: https://en.wikipedia.org/wiki/CubeSat (visited on 07/22/2018).

[11] (2018). ISIS High Data Rate S-Band Transmitter, [Online]. Available: https://www.isispace.nl/product/isis-txs-s-band-transmitter/.

[12] (2018). GNU Radio, [Online]. Available: https://www.gnuradio.org/ (visited on 06/05/2018).

[13] (2018). LuaRadio benchmarks, [Online]. Available: http://luaradio.io/benchmarks.html#intel-i5-desktop (visited on 06/05/2018).

[14] (2018). PothosSDR, [Online]. Available: https://github.com/pothosware/PothosCore/wiki (visited on 06/05/2018).

[15] (2018). RedHawk SDR, [Online]. Available: https://redhawksdr.github.io/Documentation/ (visited on 06/05/2018).

[16] (2018). PySDR, [Online]. Available: https://github.com/pysdr/pysdr (visited on 06/05/2018).

[17] (2018). Matlab, [Online]. Available: https://www.mathworks.com/products/matlab.html (visited on 06/05/2018).

[18] (2018). Simulink, [Online]. Available: https://www.mathworks.com/products/simulink.html (visited on 06/05/2018).

[19] (2018). GNU Octave, [Online]. Available: https://www.gnu.org/software/octave/ (visited on 06/05/2018).

[20] (2018). Microsoft Sora, [Online]. Available: https://github.com/Microsoft/Sora (visited on 06/05/2018).

[21] (2018). GNU Radio external modules, [Online]. Available: https://wiki.gnuradio.org/index.php/OutOfTreeModules (visited on 06/05/2018).

[22] (2018). GNU GPLv3 license, [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html (visited on 06/05/2018).

[23] (2018). GNU Radio and ZeroMQ, [Online]. Available: https://oshearesearch.com/index.php/2014/12/30/gnu-radio-zeromq-block-enhancements/ (visited on 06/05/2018).

[24] (2018). Signal probe, [Online]. Available: https://wiki.gnuradio.org/index.php/Guided_Tutorial_GRC#2.4.1._Examining_the_Probe_Signal_Block (visited on 07/06/2018).

[25] M. M. Michael Kobayashi, "Iris Deep-Space Transponder for SLS EM-1 CubeSat Missions, 31st Annual AIAA/USU Conference on Small Satellites", presented at the 31st AIAA/USU Conference on Small Satellites.

[26] CCSDS, Ed. (Feb. 2014). CCSDS 414.1-B-2, Pseudo-Noise (PN) Ranging Systems, [Online]. Available: https://public.ccsds.org/Pubs/414x1b2.pdf.

[27] ECSS, Ed. (Jul. 31, 2008). ECSS-E-ST-50-01C, Space data links – Telemetry synchronization and channel coding, [Online]. Available: http://ecss.nl/standard/ecss-e-st-50-01c-space-data-links-telemetry-synchronization-and-channel-coding/.

[28] ECSS, Ed. (Jul. 31, 2008). ECSS-E-ST-50-02C, Space data links – Space engineering Ranging and Doppler tracking, [Online]. Available: http://ecss.nl/standard/ecss-e-st-50-02c-ranging-and-doppler-tracking/.

[29] ECSS, Ed. (Jul. 31, 2008). ECSS-E-ST-50-03C, Space data links - Telecommand protocols, synchronization and channel coding, [Online]. Available: http://ecss.nl/standard/ecss-e-st-50-04c-space-data-links-telecommand-protocols-synchronization-and-channel-coding/.

[30] ECSS, Ed. (Jul. 31, 2008). ECSS-E-ST-50-04C, Space data links – Telemetry transfer frame protocol, [Online]. Available: http://ecss.nl/standard/ecss-e-st-50-03c-space-data-links-telemetry-transfer-frame-protocol/.

[31] ECSS, Ed. (Oct. 24, 2011). ECSS-E-ST-50-05C, Space engineering Radio frequency and modulation, [Online]. Available: http://ecss.nl/standard/ecss-e-st-50-05c-rev2-radio-frequency-andmodulation-4-october-2011/.

[32] (2018). Universal Software Radio Peripheral, [Online]. Available: https://en.wikipedia.org/wiki/Universal_Software_Radio_Peripheral.

[33] C. Duncan, A. Smith, and F. Aguirre, "Iris Deep-Space Transponder for SLS EM-1 CubeSat Missions, 31st Annual AIAA/USU Conference on Small Satellites", presented at the Conference on Small Satellites, 2018.

[34] B. Cook, M. Dennis, S. Kayalar, J. Lux, and N. Mysoor, "Development of the Advanced Deep Space Transponder", JPL, Progress Report, Feb. 15, 2004.

[35] (2018). Numerical cancellation in trigonometric functions, [Online]. Available: https://randomascii.wordpress.com/2014/10/09/intel-underestimates-error-bounds-by-1-3-quintillion/ (visited on 06/05/2018).

[36] D. Müller, D. Dirkx, G. Kopeikin S M andLion, I. Panet, G. Petit, and P. N. A. M. Visser, "High Performance Clocks and Gravity Field Determination", Nov. 30, 2017.

[37] (2018). Automatic gain control, [Online]. Available: https://en.wikipedia.org/wiki/Automatic_gain_control (visited on 09/01/2018).

[38] J. P. Alegre Pérez, *Automatic Gain Control, Analog Circuits and Signal Processing*, S. S. Media, Ed. 2011. DOI: 10.1007/978-1-4614-0167-4_2.

[39] K. Banks. (Feb. 27, 2014). To Use or Not to Use FFT Filters, [Online]. Available: http://www.trondeau.com/blog/2014/2/27/to-use-or-not-to-use-fft-filters.html (visited on 08/15/2002).

[40] (2018). Double-precision floating-point format, [Online]. Available: https://en.wikipedia.org/wiki/Double-precision_floating-point_format.

[41] M. K. Simon, "A Tracking Performance Comparison of the Conventional Data Transition Tracking Loop (DTTL) with the Linear Data Transition Tracking Loop (LDTTL)", JPL, IPN Progress Report 42-162, Aug. 15, 2005.

[42] A. Mileant, S. Million, S. Hinedi, and U. Cheng, "The performance of the all-digital data transition tracking loop using nonlinear analysis", *IEEE Transactions on Communications*, vol. 43, 2/3/4 1995. [Online]. Available: https://ieeexplore.ieee.org/document/380153.

[43] M. Srinivasan, A. Tkacenko, M. Lyubarev, and P. Estabrook, "Effects of Transmitter Symbol Clock Jitter Upon Ground Receiver Performance", JPL, Progress Report, May 15, 2010.

[44] (2018). GNU Radio Symbol Synch block, [Online]. Available: https://www.gnuradio.org/wp-content/uploads/2017/12/Andy-Walls-Samples-to-Digital-Symbols.pdf (visited on 07/06/2018).

[45] J. B. Berner, J. M. Layland, and P. W. Kinman, "Flexible Carrier Loop Design for the Spacecraft Transponding Modem (STM)", JPL, Progress Report, Nov. 15, 1998.

[46] R. E. Best, *Phase-Locked Loops, Design, Simulation, and Applications*, ISBN: 978-0-07-149375-8.

[47] M. Rice, *Digital Communications: A Discrete-Time Approach*. 2009.

[48] D. Morabito and D. S. Abraham, "Multiple Uplinks Per Antenna (MUPA) Signal Acquisition Schemes", presented at the 2018 SpaceOps Conference. DOI: 10.2514/6.2018-2610.

[49] N. I. Inc. (). Tutorial on Measurement of Power Spectra, [Online]. Available: http://123.physics.ucdavis.edu/week_2_files/tutorial_on_measurement_of_power_spectra.pdf (visited on 10/10/2018).

[50] (2018). The Goertzel Algorithm, [Online]. Available: https://www.embedded.com/design/configurable-systems/4024443/The-Goertzel-Algorithm (visited on 10/21/2018).

[51] J. K. Cavers and M. W. Liao, "Adaptive compensation for imbalance and offset losses in direct conversion transceivers", *IEEE Transactions on Vehicular Technology*, 1993.

[52] Li and Yabo, *In-Phase and Quadrature Imbalance*. 2014, ISBN: 978-1-4614-8618-3.

[53] (2018). Unit testing framework, [Online]. Available: https://docs.python.org/2/library/unittest.html.

[54] (2018). Out-Of-Tree modules, [Online]. Available: https://wiki.gnuradio.org/index.php/OutOfTreeModules (visited on 10/21/2018).