Software-defined testbed for next generation navigation transponders

Speretta, Stefano; Oei, Hong Yang; Verhoeven, Chris; Dirkx, Dominic; Karunanithi, Visweswaran; Bentum, Mark; Miraglia, Antonio; Rotteveel, Jeroen; Alvarez, Marco; More Authors

DOI
10.1109/TTC.2019.8895459

Publication date
2019

Document Version
Final published version

Published in
TTC 2019 - 8th ESA International Workshop on Tracking, Telemetry and Command Systems for Space Applications

Citation (APA)
Speretta, S., Oei, H. Y., Verhoeven, C., Dirkx, D., Karunanithi, V., Bentum, M., Miraglia, A., Rotteveel, J., Alvarez, M., & More Authors (2019). Software-defined testbed for next generation navigation transponders. In *TTC 2019 - 8th ESA International Workshop on Tracking, Telemetry and Command Systems for Space Applications* Article 8895459 (TTC 2019 - 8th ESA International Workshop on Tracking, Telemetry and Command Systems for Space Applications). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/TTC.2019.8895459

Software-defined testbed for next generation navigation transponders

Stefano Speretta
Delft University of Technology
Delft, The Netherlands
S.Speretta@tudelft.nl

Chris Verhoeven
Delft University of Technology
Delft, The Netherlands
C.J.M.Verhoeven@tudelft.nl

Dominic Dirkx
Delft University of Technology
Delft, The Netherlands
D.Dirkx@tudelft.nl

Visweswaran Karunanithi
Delft University of Technology
Delft, The Netherlands
V.Karunanithi-1@tudelft.nl

Mark Bentum
Eindhoven University of Technology
Eindhoven, The Netherlands
m.j.bentum@tue.nl

Odysseas Votsis
Eindhoven University of Technology
Eindhoven, The Netherlands
o.votsis@student.tue.nl

Antonio Miraglia
Innovative Solutions in Space B.V.
Delft, The Netherlands
a.miraglia@isispace.nl

Jeroen Rotteveel
Innovative Solutions in Space B.V.
Delft, The Netherlands
j.rotteveel@isispace.nl

Marco Alvarez
Innovative Solutions in Space B.V.
Delft, The Netherlands
m.alvarez@isispace.nl

Hong Yang Oei
Innovative Solutions in Space B.V.
Delft, The Netherlands
h.oei@isispace.nl

Alberto Busso
European Space Agency
Noordwijk, The Netherlands
alberto.busso@esa.int

***Abstract*—This paper presents a software-defined testbed to perform hardware-in-the-loop test of miniaturized coherent transponders. Such a setup has been designed to minimize the access threshold for future users, heavily relying on available open source applications and commercial hardware, targeting future coherent transponders for interplanetary CubeSats. The paper presents the overall architecture of the testbed, a trade-off to select the most suited development framework and the detailed design of the different blocks. Upcoming interplanetary CubeSat missions that would require a coherent transponder are also presented to highlight the need sof such a system. Software qualification, given the use of third-party software with multiple developers, was also addressed to guarantee performances can be consistent and reliable.**

***Index Terms*—SDR, interplanetary CubeSat, testbed, coherent transponder**

I. Introduction

With small satellites moving past Low-Earth Orbit (LEO) and targeting interplanetary missions, a whole set of new components is becoming part of the standard platform hardware: coherent transponders. Traditionally small LEO spacecraft rely on different means to estimate their position (like radar tracking or GPS measurements) but these mechanisms are not available in deep space, where satellites must rely on the traditional radiometric tracking. Transponders implementing ranging and coherent downlink are required to best estimate the spacecraft position and relative velocity. This represents an evolution of the standard transceiver used in CubeSats that so far had limited or no need at all for coherent ranging. In this paper we focus on the efforts to develop a fully software-defined testbed for testing transponders and ground modems compatible with ECSS [1] standards to enable a fully European interplanetary CubeSat mission. As part of the testbed, a reference software-defined transponder is also presented, showing the benefits of the selected framework in the development of Software-Defined-Radio (SDR) applications. We will present the lessons learnt on using tools like GNU Radio and other open source applications for space applications, dealing with the typical testing, qualification and quality assurance issues. Due to the complexity of the testing and qualification of TT&C transponders, a fully software-defined testbed has been designed, capable of emulating the complete radio chain starting from the ground modem to the interplanetary RF channel till the satellite transponder. This solution allows to quickly develop the system and evaluate its performances before starting the real hardware development. Using existing SDR platforms, real hardware can be easily added in the loop. This approach allows using actual units (existing ground modems or satellite transponders, for example) in the simulation testbed to speed-up design and characterize

This work was supported by the European Space Agency under contract ESA AO/2-1660/17/NL/FE.

978-1-7281-3700-1/19/$31.00 ©2019 IEEE

performances. Such a testbed can perform automated tests on the complete system, leading to a very quick unit acceptance process, as needed especially for manufacturing transponders on a large scale. The testbed has been designed using GNU Radio, because it allows reusing a large library of existing functions. A full library implementing ECSS-specific functions and relevant standard terminology has been implemented. Dedicated wrapper components re-using existing GNU Radio functionalities with customized interfaces or new blocks for missing functionalities have also been developed.

A consistent part of the project has been dedicated to quality assurance issues arising from a public open source project with multiple developers and a very fast update pace to be used in an aerospace environment. A complete set of unit and integration tests has been developed to ensure also core functionalities are validated against known data sets to prevent bugs introduced in future versions. An automated test reporting system has also been developed to prevent manual errors from entering the quality assurance chain, and a full file checksum system has been created to guarantee integrity of the files with respect to the qualified version.

This paper presents a general overview of miniaturized space missions (Section II), showing the need for coherent transponders in the future CubeSat missions. Section III describes the overall system architecture while Section IV focuses on the selection for the software framework. System implementation is presented in Section V and software qualification is presented in Section VI.

II. Miniaturized Space Missions

CubeSats have been traditionally always used for missions targeted to LEO: this reduced heavily the environmental requirements on the mission (thermal, radiation, available power, navigation, attitude, etc.) and allowed for a lighter, simpler and ultimately cheaper mission. This in turn lead to less interest in the development of on-board sub-systems not strictly required in LEO, such as navigation ones. The availability of Two-Line Elements (TLE) allowed to compute approximated satellite positions and this has been accurate enough for most missions (despite the limited accuracy achievable [2]). GNSS receivers have also been used recently when a higher accuracy was required while radiometric tracking is not used in CubeSats. Unfortunately this is still the most common navigation technique for interplanetary missions, limiting CubeSat exploitation for interplanetary missions.

Coherent transponders are still uncommon for small satellites and their development is a recent trend: the first example of a CubeSat used on an interplanetary mission are the MARCO [3] CubeSats, used as data relay during the entry, descent and landing phase on Insight on Mars. The CubeSats where not critical for the success of the missions but were used to rely landing telemetry during the descent, allowing to closely monitor the process and, eventually, have much more information about problems in that phase. To also reach the desired reliability, two satellites were used (MARCO-A and -B), also thanks to their relatively low mass and cost.

This initial mission using small satellites far from Earth pushed the development of similar concepts, where the small satellite is also hosting important (but not critical) scientific instruments: this is the case, for example, of the COPINS mission [4] that is tightly coupled to HERA [5]. The CubeSats are flying together with the main mission and can provide complementary science but the main science objective is achieved with the main mission.

Pushing further the concept of interplanetary CubeSats, several missions have been proposed that rely on the CubeSat for the main scientific mission: this is due to the very high cost a bigger satellite would have in the same mission or even due to the impossibility to carry out the scientific task with a bigger satellite. Examples of this last trend are M-Argo [6], targeting a near-Earth object, or LUMIO [7], designed to perform long-term observations of the far-side of the Moon to characterize meteoroid impacts.

Deep-space constellations have been proposed in rare cases, due to the high cost and complexity: OLFAR [8] is an example where multiple CubeSats could be used around the Moon for radio interferometry, something that only small satellites could make possible at the proposed scale.

All the missions just presented are a clear example of the need for navigation transponders for small satellites and related equipment. The aim of this paper is providing an easy entry-level equipment for real hardware-in-the-loop simulation in early mission phases or during final mission testing.

III. Architecture

The general idea behind this project is building a testbed to allow testing the coherent transponder full-chain in a simple and flexible way, starting from the ground modem to the transponder itself. An important implementation constraint was to be capable of supporting a completely simulated system or to introduce hardware components in the loop. Such an approach would allow to first simulate a new design for a transponder or a ground modem and, at a later stage, test the hardware in the exact same situation, actually lowering development cost and time. As transponders for small satellites and CubeSats are the last entrants in the market, this system has been targeted at them but it should not be strictly limited to such hardware.

To achieve the above goals, we divided the overall system in 4 main blocks:

- A modem, used to simulate the ground components of a TT&C chain;
- A channel / dynamics simulator, used to generate real operating conditions to test the transponder and modem blocks;

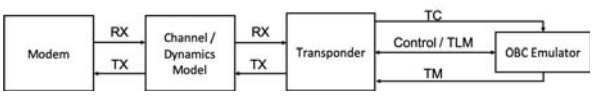

Fig. 1. General system architecture

- A transponder block that mimics a real hardware TT&C transponder;
- An On-Board Computer (OBC) emulator that is used to control the transponder as closely as possible to the real operational conditions.

The test setup is depicted in Fig. 1, where also the connections between the blocks are shown. The selection of the different blocks has been done to ensure the overall system is as similar as possible to the real operational case: a transponder is meant to be controlled by the spacecraft OBC and communicate with the ground modem. Channel modelling has been added but currently only includes additive white Gaussian noise. Orbital dynamics (and so Doppler effect) has been included in the channel model for simplicity. The different blocks will be described in more details in the following sections.

It is important to note that such division of blocks has been selected to also allow for a very modular design, splitting all functionalities in clearly identified containers. Future versions of this testbed can provide alternative implementations of each block, aiming at a full hardware-in-the-loop test, where certain components are running on a computer, as part of a simulation package, and the others are real hardware components.

A. Modem

This block is responsible of providing the RF companion to the transponder, simulating the behaviour of a ground modem (see Fig. 2 for further details). Because this block can be very complex, at this stage, it was decided to limit it to the minimum required to test a coherent ranging transponder and make the architecture flexible enough to be extended in the future.

This block thus requires a recording (IQ-file) taken from a modem while the modem is operating (for example, executing a Physical Layer Operation Procedure [9] to allow the transponder to lock). This has to be recorded while a reference modem is operated in ”open-loop” mode: such an input will be then provided to the transponder and its output will also be recorded. Such recording can be used at a later time for testing the functionality of the transponder without requiring the actual reference modem. The selection of an IQ-file was done to simplify the simulation of the baseband systems, but it clearly brings some limitations: sampling is assumed ideal (thus discarding eventual effects due to the jitter in the master clock that have to be added to the baseband signal, if needed) and limiting the flexibility of the setup at this stage. It is planned to solve this limitation in future versions where a full implementation of the modem side will be performed and a better non-ideality modelling will be included. IQ-files also limit coherent operations: some additional blocks have been added to be able to compute the turn-around ratio and the propagation time through the transponder to perform the basic operations a ground modem would perform.

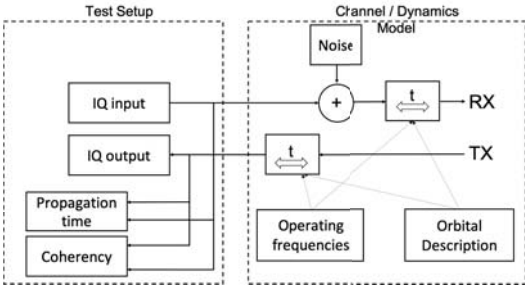

Fig. 2. Test setup and channel / dynamics model block diagram

B. Channel / Dynamics Model

Channel modelling in this first version has been limited to additive white Gaussian noise to account for a simple channel. Due to the modular nature of the system, further effects can be added to better match the interplanetary channel (frequency-selective effects of, for example, effects related to specific locations in the solar system like planetary atmospheres). Leaving this possibility open for future revisions is the reason why propagation and orbital effects have been linked together in one single block, despite their very different nature. More details are shown in Fig. 2.

Orbital dynamics for selected test cases, defined by a combination of ground station / spacecraft locations and a time duration, is performed using TUDat [10], an open-source software suite designed for astrodynamics research. This allows to create a generic orbital description file using:

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

Based on these settings, the orbit of the spacecraft is numerically propagated. This allows to use realistic geometries (even from existing missions) and simulate the conditions the transponder would go through. Doppler is simulated by interpolating the signals (uplink and downlink) actually performing time compression and expansion. This allows to feed the transponder software model (or its hardware equivalent) with a digital IQ signal at a nominal sample rate, despite the signal being a re-sampled copy at a slightly different sample rate (the effect is equivalent to applying the Doppler effect to the sampling clock). 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 re-transmitted signal from spacecraft to the ground station will again be simulated using the approach outlined above, to obtain the downlink Doppler shift. This allows us to quantitatively analyze the impact that our system has on tracking data quality. Moreover, by adding environment noise to both the ideal and hardware-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.

C. 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:

- Tele-Command (TC) bitstream;
- Telemetry (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. This approach allows to push decoding and de-framing to the OBC, which is also the common implementation. In the current version, data on this interfaces will be logged / loaded to / from file for simplicity.

D. Transponder

The transponder has been implemented as a SDR with a zero-IF radio architecture: this implies that the digital system will have input and output signals close to 0 Hz and the full RF to zero-IF conversion is handled elsewhere (typically in hardware, with the assumption that the up/downlink frequencies keep an integer ratio between them) and it is not considered in this project. This block has been designed to perform the following functions:

- **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 relative Doppler (so the relative velocity). It involves demodulating the uplink signal and re-modulating it back (with a non-coherent or coherent frequency).

Fig. 3 shows a simplified block diagram, highlighting the most important blocks.

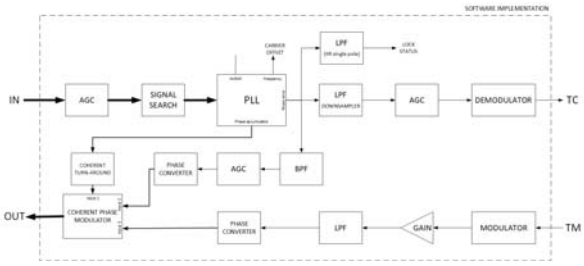

Fig. 3. Transponder block diagram

IV. Framework Selection

The primary requirement for testbed presented in this article is being simple to use and accessible to potentially everyone. Accessibility is a fundamental requirement when addressing the CubeSat community which is mostly made by universities and small entities, usually working with limited budgets and short development cycles. This forced us to think about which type of tools we wanted to use and look for their general availability. Open-Source tools were selected as the best choice, as they can be downloaded freely and used with very limited restrictions. Keeping the same philosophy, this testbed has also been released as Open-Source [11] to ensure the maximum distribution among the community.

To further benefit also from the wide diffusion of SDR equipment [12] [13] and the capability of processing and generating arbitrary RF signals at a very limited cost, we will rely on commercial SDR equipment to connect a hardware device to to our simulation environment. This allows to create a very simple and capable testbed that can be used to simulate fully in software a transponder chain or replace certain building blocks with the real device.

The first step towards the implementation has been the selection of a general framework to implement the project. Several frameworks are currently available, as shown in Table I, with different features, advantages and disadvantages. For our trade-off we gave importance to several criteria, in particular:

- **project maturity**, including also the estimated size of the community behind each framework. The latter point is critical as it drives the amount of available online information, eventual support from other users and the number of maintainers focused on improving the framework. Preference has been given to frameworks with large user basis and vast supporting communities;
- **license**: this in principle can limit the possible usage of such a framework or of future upgrades to it. Preference has been given to less restrictive licenses, in principle also allowing commercial services;
- **hardware support**: it is important for future upgrades of our testbed to support hardware-in-the-loop testing with real transponders or modems. Preference has been given to frameworks with the wide list of supported interfaces;
- **graphical environment**: this is considered important as it makes the software easier to use and quicker to learn.

Based on the previously mentionend requirements, we directly excluded Redhawk SDR since there is no hardware support. PySDR has a rich sized community, but it has a very limited hardware support (very few commercial devices supported) and most importantly the lack of a GUI makes PySDR unsuitable.

MATLAB and Simulink, despite having an extremely wide set of features, MATLAB and Simulink were also discarded due to the commercial license required and the very limited hardware support. Lastly MATLAB is widely used for off-

TABLE I
PRELIMINARY FRAMEWORK TRADE-OFF

Framework	Criteria			
	HW support	***License***	***GUI***	***Maturity***
GNU Radio [14]	HIGH	GPLv3	YES	HIGH
LuaRadio [15]	HIGH	MIT	YES	LOW
PothosSDR [16]	HIGH	MIT	YES	MEDIUM
RedHawk SDR [17]	NO	LGPL	YES	MEDIUM
PySDR [18]	MEDIUM	GPLv3	NO	HIGH
Matlab [19]	MEDIUM	Commercial	YES	HIGH
Simulink [20]	MEDIUM	Commercial	YES	HIGH
GNU Octave [21]	MEDIUM	GPLv3	YES	MEDIUM

line analysis, while other frameworks thrive also in real-time analysis.

GNU Octave is the least competitive candidate due to its limited hardware support despite the community seems quite big but has unfortunately generated few real applications in this specific field.

LuaRadio has a very small footprint (few tens of MB), has an input/output signature system that is very flexible and it supports automatic propagation of sample rate between all blocks, which makes its usage very simple. It also supports a vast selection of hardware and has also a growing community, 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 potential users would have to learn it first. Secondly, it appears to have worse performances than GNU Radio, as seen on this test benchmarks. 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 suitable one for this project. Both frameworks have a very solid hardware support, covering the vast majority of the available SDR platforms. Both frameworks have a very similar GUI and comparable performances. Community size difference, though, is very big: PothosSDR is a relatively recent development and can rely on few users while GNU Radio has been a very popular software for more than decade. Based on the previous conclusions, we selected GNU Radio for this implementation, despite several critical points that emerged, in particular:

- 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.

The latter point require attention, especially in view of quality and/or reliability requirements on the software which might be difficult to trace with a rapid development cycle, as it is very common on many Open-Source applications.

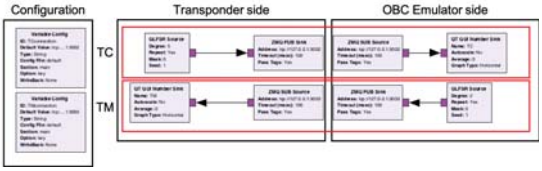

Fig. 4. Transponder - OBC emulator interface

V. IMPLEMENTATION

The implementation is based on GNU Radio, which provides a runtime environment and an easy user interface to describe dataflow algorithms. The framework already provides many of the software building blocks required for this project and it allows to add custom libraries to implement the missing functions. Since GNU Radio is typically used for research and commercial applications, not all the functionalities related to space standards are already implemented: modulation and demodulation blocks, for example, do not support a partially suppressed carrier and this needs to be added.

To systematically implement all required components to support ECSS-compatible transponders, a dedicated components library (gr-ecss [11]) was developed to group all ECSS-specific [1] functions. A second library has also been developed to support more general functions, needed for the testbed. Several components, also specific for the compatibility to the ECSS standards, have also been already developed by several authors but no clear and formal qualification and verification process was present, leading to the extra work to ensure the compatibility to the standard.

As described in Section III, the whole testbed is divided in several blocks and communication between the blocks can be complex. GNU Radio provides several solutions based on IP sockets, like shown in Fig. 4, that allow to create a clean separation in between the different functional blocks. In principle this separation can also happen over a network, allowing to share the load on several machines. This solution has been adopted to provide even more flexibility to the system: in principle, any application capable of communicating over an IP socket could be used. This becomes very convenient, for

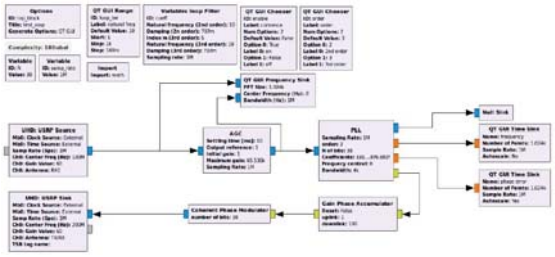

Fig. 5. Transponder simplified implementation

example, for emulating the OBC: the full code could be run on a computer, allowing a complete simulation of the interaction of the OBC, the transponder and the ground modem. A real OBC could be also used in the set-up by adding a serial interface to the computer for the TC / TM streams: a software driver could translate the hardware serial interface into a IP socket connection and route the data, allowing to add even more hardware to the full simulation.

The transponder model has also been implemented in GNU Radio (see Fig. 5 for a simplified version), together with a set of blocks, implementing functionality not present in the current GNU Radio release. An example of this is a PLL / modulator that could be used to recover an incoming carrier and generate a coherent uplink. The default GNU Radio blocks were limited to a very simple implementation (for example without the possibility of configuring the loop filter except for the second order bandwidth). Hardware-related effects, like the effect of the limited size of the phase accumulator are also difficult to simulate and so dedicated blocks were create as well.

VI. Testing and qualification

Software testing and qualification is more complex than in usual cases as we heavily rely on third-party software, developed by multiple authors which are loosely coordinated. This creates a high risk for bugs to enter the system but has the advantage of a very quick development cycle and eventually also a quick bug solution. Verification is very time consuming and, in our particular case, requires to be re-run with every new version of the software (4 releases per year occurred in 2018 and 2019). This creates a big overhead but it allows to benefit from the latest updates and improvements (performances improvements or operating system compatibility): on a typical small satellites program it is difficult to lock to a specific framework version so being able to support the latest available one is important. To address this problem, we looked in a complete suite for automatic testing: part of it was already present as part of GNU Radio but it had to be improved to address standard-specific requirements (attack time for the automatic gain control block, for example, as specified in [1]). This required a more complex framework to be developed to allow for a more complex test per each block, rather than the simplified system that was already available.

Furthermore, a proper documentation of the test results was needed and not already present in GNU Radio. An automatic report generation system has been added to visualize in a more friendly way the test results, allowing also for an easy off-line review of them. Visual review of the results is also possible by adding detailed graphs of the critical test cases, allowing to both check the overall test status (pass / fail) or verify in details the functionality.

Dedicated tests have also been performed on the system with the aid of a reference modem (provided by ESA) which was used to generate recordings (IQ files) to be loaded in the transponder and compare with the expected results. Such

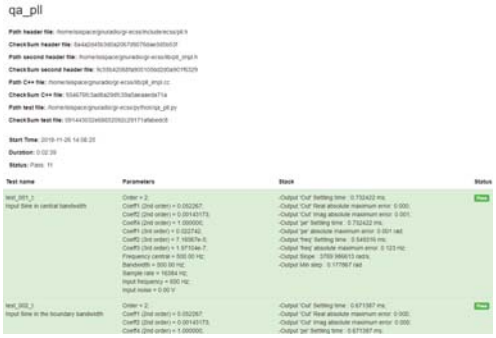

Fig. 6. QA test report example

recordings are integral part of the test system and can be used also for later versions of the software.

A. Quality Assurance Tests

Quality Assurance (QA) tests have been created by extending the existing GNU Radio framework to better test the developed blocks (and potentially GNU Radio internal blocks, to raise the overall quality level of the framework). In fact, being GNU Radio open source, anyone will be able to develop their own blocks in order to freely modify the architecture and still rely upon certified tests.

All the tests have been added to the compilation process and are run automatically after a new version is compiled. This allows for a continuous monitoring of the functionality of the software during development. In order to guarantee the maximum traceability of the tests, it was decided to modify, through the creation of a new library, the generation of reports files. Whenever each block is compiled, the tested is run and a HTML header is generated containing a list of all required files (headers, other files, etc.), their current version (taken from the version control tool) and a checksum per each file. This is outputted as a PDF file to be added to the software documentation: Fig. 6 shows an example of the output report per each test. Moreover, to allow for a deep review of the

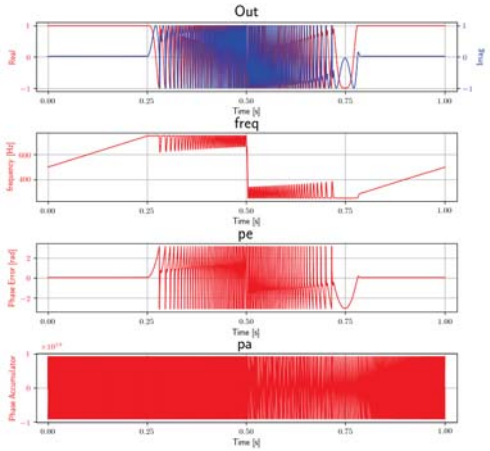

Fig. 7. Visual QA test result for a PLL

functionality being tested, graphs are added (as shown in Fig. 7) to independently verify the functionality of the test.

This system generates automatically the complete report (results and graphs) per each compile cycle, allowing to keep track during the development of remaining bugs in the code. The report can directly be attached to the documentation, simplifying also the overall process.

B. Validation tests

Two tests have been performed to validate the system: reception / transmission using a reference modem and the measurement of the coherent ratio and the propagation delay through the transponder.

Reception and transmission has been done using an existing modem: this test required a recording done with the modem output 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 and can also be repeated at a later stage (for example when qualifying a later version of the framework) by relying on the recordings.

Ranging / tracking tests cannot be performed using a modem as they cannot be performed in real-time (as the library under development does not operate yet in real-time with a hardware interface). In this test, a reference signal is generated (plain carrier with a frequency sweep, sequence of tones) and recorded. Later this is provided to the input of the transponder and the output is recorded and 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) is used on both the input and output to reconstruct the signal phase and then the correlation will be computed on the signal phase.

Further validation tests will have to be performed when the library will be upgraded to support real-time operation with an hardware interface. At that stage, a full validation can be performed with a hardware reference modem and compared with the measurements taken by the transponder.

VII. Conclusions

With this paper we aim at showing that the use of open source applications can be very useful for space systems development, especially as a way to get quicker development time by relying on a big developers team. This is also a way to reduce the costs for space systems, especially with small satellite / CubeSat missions which are typically budget-limited. In this paper we presented a testbed for testing coherent transponders that relies heavily on open source frameworks, like GNU Radio and TUDat, and commercial hardware. Part of the development was also dedicated to build a reference model for a coherent transponder that would serve as a baseline for future developments of European CubeSat deep-space transponders, required by several upcoming missions. Such testbed was designed to perform a full simulation of the transponder / modem design in real conditions (noise and satellite dynamics) and also include real hardware in the simulation chain to validate the final implementation.

We also presented an improved framework to perform automated software testing and validation by relying of reference hardware and creating software models that could be used at a later stage. This is particularly important when working with open source applications with a quick release cycle, where bug-fixes and new and important features can greatly bring benefits to the project but would also require re-validating the software often.

References

[1] "Ranging and Doppler tracking," European Space Agency, Noordwijk, NL, Tech. Rep., Jul. 2008.

[2] S. Speretta, P. Sundaramoorthy, and E. Gill, "Long-term performance analysis of norad two-line elements for cubesats and pocketqubes," in *Proceedings Small Satellites for Earth Observation*. DLR, 2017, pp. 1–6.

[3] S. W. Asmar and S. Matousek, *Mars Cube One (MarCO) Shifting the Paradigm in Relay Deep Space Operation*. AIAA, 2016. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2016-2483

[4] R. Walker, D. Binns, I. Carnelli, M. Kueppers, and A. Galvez, "Cubesat opportunity payload inter-satellite network sensors (copins) on the esa asteroid impact mission (aim)," in *Interplantary CubeSat Workshop, Oxford, UK*, 2016.

[5] D. Sears, C. Allen, D. Britt *et al.*, "The hera mission: multiple near-earth asteroid sample return," *Advances in Space Research*, vol. 34, no. 11, pp. 2270 – 2275, 2004, scientific Exploration, Planetary Protection, Active Experiments and Dusty Plasmas. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0273117704005198

[6] R. Walker, D. Koschny, C. Bramanti, and I. Carnelli, "Miniaturised asteroid remote geophysical observer (m-argo): a standalone deep space cubesat system for low-cost science and exploration missions," in *Interplantary CubeSat Workshop, Paris, France*, 2018.

[7] S. Speretta, A. Cervone, P. Sundaramoorthy *et al.*, *LUMIO: An Autonomous CubeSat for Lunar Exploration*. Cham: Springer International Publishing, 2019, pp. 103–134.

[8] R. T. Rajan, S. Engelen, M. Bentum, and C. Verhoeven, "Orbiting low frequency array for radio astronomy," in *2011 Aerospace Conference*, March 2011, pp. 1–11.

[9] "Telecommand protocols synchronization and channel coding," European Space Agency, Noordwijk, NL, Tech. Rep., Jul. 2008.

[10] K. Kumar, Y. Abdulkadir, P. van Barneveld *et al.*, "Tudat: a modular and robust astrodynamics toolbox," 2012.

[11] FlaReSS, Flexible Radio Science System. [Online]. Available: https://github.com/FlaReSS

[12] Ettus Research. [Online]. Available: http://ettus.com/

[13] LimeSDR. [Online]. Available: https://limemicro.com/products/boards/limesdr/

[14] GNU Radio. [Online]. Available: https://www.gnuradio.org/

[15] LuaRadio. [Online]. Available: http://luaradio.io/

[16] PothosSDR. [Online]. Available: https://github.com/pothosware/PothosCore/wiki

[17] RedHawk SDR. [Online]. Available: https://redhawksdr.github.io/Documentation/

[18] PySDR. [Online]. Available: https://github.com/pysdr/pysdr

[19] Matlab. [Online]. Available: https://www.mathworks.com/products/matlab.html

[20] Simulink. [Online]. Available: https://www.mathworks.com/products/simulink.html

[21] GNU Octave. [Online]. Available: https://www.gnu.org/software/octave/