

# A Software-Defined Spaceborne GNSS Receiver

C. Fernández-Prades

*Communication Systems Division  
CTTC/CERCA*

Castelldefels, Spain

carles.fernandez@cttc.cat

J. Arribas

*Communication Systems Division  
CTTC/CERCA*

Castelldefels, Spain

javier.arribas@cttc.cat

M. Majoral

*Communication Systems Division  
CTTC/CERCA*

Castelldefels, Spain

marc.majoral@cttc.cat

A. Ramos

*Institut Municipal d'Informàtica (IMI) Communication Systems Division European Space Research and Technology Centre  
Formerly at CTTC/CERCA*

Barcelona, Spain

aramosd@bcn.cat

J. Vilà-Valls

*Communication Systems Division  
CTTC/CERCA*  
Castelldefels, Spain  
jvila@cttc.cat

P. Giordano

*European Space Research and Technology Centre  
European Space Agency (ESA)  
Noordwijk, The Netherlands  
Pietro.Giordano@esa.int*

**Abstract**—This paper reports the design, proof-of-concept implementation and preliminary performance assessment of a low-cost, software-defined spaceborne GNSS receiver. The presented approach takes advantage of the flexibility of software-defined radio technology and the forthcoming availability of radiation-hardened, space-certified Systems-on-Module to implement a fully customizable receiver with the capability to process GNSS signals in real-time and to deliver GNSS products in standard formats. The core GNSS engine is based on a free and open source software implementation of a multi-band, multi-system GNSS receiver released under the General Public License v3.0 and available in a public source code repository.

**Index Terms**—Aircraft navigation, Software radio, Embedded software, Signal processing, Global navigation satellite system.

## I. INTRODUCTION

Spaceborne GNSS receivers are currently critical elements of multiple space missions, ranging from science to fully commercial applications, including new navigation concepts and algorithms for space users (*e.g.*, on-board precise orbit determination, radio occultation, GNSS reflectometry, space weather).

However, spaceborne GNSS receivers need to operate in scenarios that are quite different from those of ground-based receivers, in particular the higher (albeit predictable) dynamics conditions. In addition, space is a harsh environment for semiconductor devices. Charged particles and gamma rays create ionization, which can alter device parameters. In addition to permanently damaging complementary metal-oxide semiconductor (CMOS) integrated circuits, radiation may cause single-event effects, which are caused by ionizing radiation strikes that discharge the charge in storage elements, such as configuration memory cells, user memory and registers. When those effects happen, the system is usually recoverable with a power reset or a memory rewrite, but they also may destroy the device [1].

Until recently, radiation-hardened solutions were limited to application-specific integrated circuits (ASICs) and one-time-

programmable solutions. A relevant example of spaceborne GNSS receiver is the AGGA-4 (Advanced Galileo and GPS ASIC, see [2], [3]). However, lately there has been an increase in the availability of space-grade FPGAs and memory devices. As examples, we can mention Xilinx's Virtex-5QV, Microsemi's RTG4 and Atmel's ATF80 FPGA processors. Those devices potentially allow the implementation of space-qualified GNSS receivers fully defined by software. Finally radiation tolerant solutions are becoming more and more popular in low cost missions, such as GOMspace's GOMX-3 commercial software-defined radio platforms.

Software-defined receivers bring interesting features for space, such as reprogrammability (or upgradeability) and self-healing (or auto-remediation) capabilities. Examples could be the possibility to upload algorithms yet-to-be-invented at the receiver's launch time, or the ability to recover from a single-event effect by remotely rewriting damaged functionalities, reducing the need of onboard redundancy.

The objective of the work presented in this paper is to demonstrate the feasibility of a software-defined spaceborne GNSS receiver. The proposed design constitutes a new approach to GNSS spaceborne receivers. While current approaches are ASIC-based, the design presented in this paper takes advantage of the flexibility of software-defined radio technology and the forthcoming availability of radiation-hardened, space-certified Systems-on-Module (SoMs) with the capability to receive GNSS signals and to process them in real-time, all in a customized UNIX-like environment providing standard communication protocol interfaces for telemetry and telecontrol.

In a further innovation step, the core GNSS engine is based on a free and open source software implementation of a multi-band, multi-system GNSS receiver released under the General Public License v3.0 and available in a public source code repository<sup>1</sup>. This scheme brings some benefits to the development, such as public scrutiny of how the baseband

This work has been partially funded by the European Space Agency through contract AO/2-1647/17/NL/CRS.

<sup>1</sup>Upstream repository available at <https://github.com/gnss-sdr/gnss-sdr>

processing is performed and the possibility to provide an auditable and automatically reproducible path for scientific experiments involving a GNSS receiver (see [4]).

The remainder of the paper is organized as follows: Section II summarizes the targeted main system requirements, Section III describes the proposed system architecture and provides implementation details, Section IV describes the developed prototype, Section V discusses performance assessment, and finally Section VI wraps up some conclusions.

## II. SYSTEM REQUIREMENTS

The targeted system features a dual-band (centered at 1176.45 and 1575.42 MHz) GNSS receiver with the ability to process GPS L1 C/A, Galileo E1b/c, GPS L5 and Galileo E5a signals, with operation modes selectable from single and dual frequencies, GPS-only, Galileo-only or multi-constellation, in any of the possible combinations of GNSS signals and with 12 channels per signal (so up to 48 channels for a dual frequency E1/L1 and L5/E5a, GPS and Galileo configuration). For the presented proof-of-concept, the targeted sampling rates were 12.5 MSps for L1/E1 and 12.5 MSps for L5/E5a signals, with 4 bits per sample.

For each visible satellite, the system must provide time-tagged measurements of pseudorange (in m), carrier phase (in cycles) or phase range (in m), Doppler shift (in Hz) or pseudorange rate (in m/s), received signal strength (in dB-Hz), dilution of precision parameters, raw navigation data, correlators' output, position and velocity, GPS Time, Galileo Time, UTC time and uncertainties, channel status (including acquisition/tracking stage, PRN number and  $C/N_0$ ), almanac, ephemeris, and annotation of observables and navigation data in RINEX 3.03 file format, as well as in other output formats for real-time streaming of GNSS products such as RTCM v3.2 and NMEA-0183 messages.

The receiver must allow for independent configuration of frequency, constellation, PVT and starting mode, including cold and warm starts, independent channel configuration, user-configurable tracking of pilot and data components, configurable integration time, and configurable rate for observables (from 1 kHz to the maximum coherent integration time) and position annotation (up to 10 Hz).

The form factor of the whole receiver must fit into a CubeSat, a type of miniaturized satellites for space research that are made up of multiples of  $10 \times 10 \times 10$  cm cubic units. CubeSats have generally a mass of no more than 1.33 kg per unit, and often use commercial off-the-shelf (COTS) components for their electronics and structure.

The receiver must also implement mechanisms to turn off channels and other functionalities not used in a particular configuration in order to reduce the power consumption. The targeted overall consumption is 5 W.

## III. SYSTEM DESIGN

The requirements on size, weight and power consumption demand the use of embedded processors. As discussed in [5], current ARM-based processing platforms are not fast enough

for a multiple-constellation, multiple-band GNSS baseband processing working in real time at the targeted sample rates. Hence, the most computational demanding operations must be off-loaded to FPGA processors.

The proposed architecture design, shown in Figure 1, is based on the ADRV9361-Z7035 board [6], which integrates an Analog Devices' AD9361 RF Agile Transceiver [7] and a Xilinx's Zynq XC7Z035-L2 FBG676I AP SoC [8] housing a Dual ARM Cortex-A9 MPCore running at 800 MHz and a Kintex-7 FPGA with 275 k logic cells and 900 DSP48 slices. This platform, shown in Figure 2 allows the implementation of a dual-band GNSS receiver providing end-to-end receiver functionalities from the RF output of an antenna (or signal generator) up to generation of navigation products in real time on a single board, compact and transportable, and fully upgradeable.

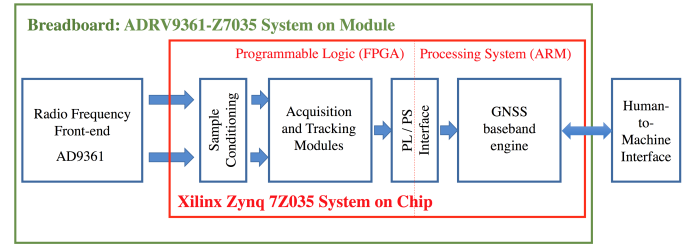


Fig. 1. Block diagram of the proposed system architecture.

The ADRV9361-Z7035 is a System-on-Module (SoM) composed of a dual-channel 70 MHz to 6 GHz front-end AD9361 IC with integrated 12-bits ADCs attached to a Xilinx Zynq Z-7035 System-on-Chip (SoC), which is divided internally in a Programmable Logic (PL) and a Processing System (PS) as shown in Figure 3.

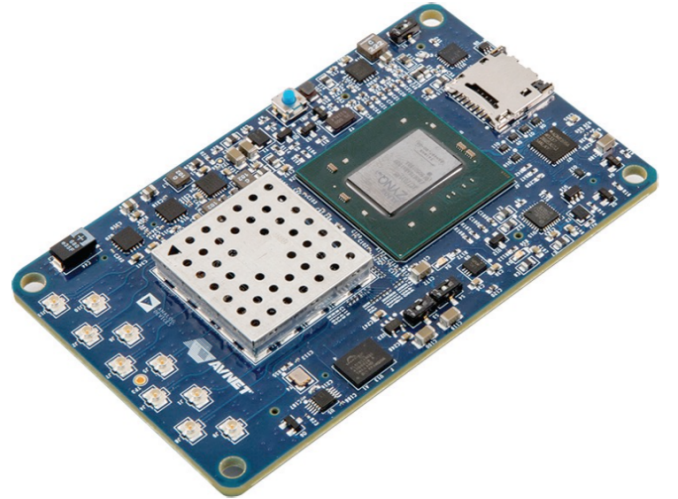


Fig. 2. Picture of the ADRV9361-Z7035 System on Module (from [6]).

The PS is composed of a dual-core ARM Cortex-A9 MPCore clocked up to 800 MHz. The on-board PS memory consists of a 1 GB DDR 3 and 256 Mb QSPI memory to store

the FPGA firmware if required. The operating system memory is implemented using a Micro SD card plugged into the on-board Micro SD card reader for prototyping purposes, although a space-qualified storage device should be used in further development stages. PS connectivity to implement the human-to-machine or machine-to-machine interfaces is guaranteed through the Gigabit Ethernet bus, the USB OTG PHY and with custom serial interfaces implemented using the SDIO/GPIO expansion bus. All the PS/PL and power supply connections are compacted in JX Micro Headers.

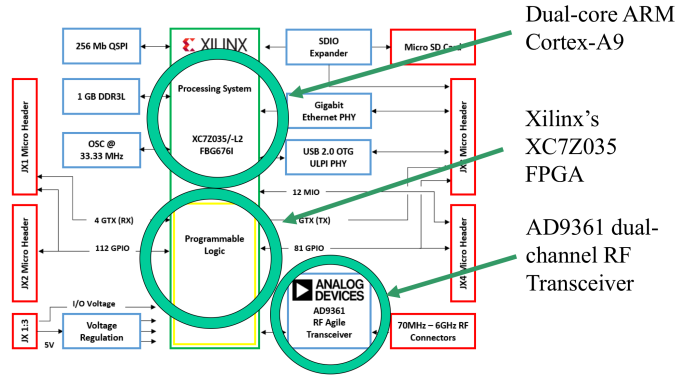


Fig. 3. ADRV9361-Z7035 System on Module diagram (from [6]).

The PL contains the FPGA fabric tightly coupled with the ARM processor. The AD9361 RF front-end is connected directly to the PL, thus assuring a direct flow of baseband samples from the ADCs to the hardware accelerators implemented in VHDL without consuming resources from the ARM processor. The front-end PLLs clock reference and the ADCs sample clock can be provided either by the local oscillator or by an external source.

A basic test using the on-board clock shipped with the ADRV9361-Z7035 revealed that its poor stability performance (25 ppm) prevented from successful tracking of GNSS signals even with an integration time of 1 ms. In order to solve that, the default oscillator was replaced with a pin-per-pin compatible high-precision temperature-compensated crystal oscillator by TXC Crystal, model TXC 7Q series for GPS specific applications, clocked at 40 MHz with a frequency stability of 2 ppm. Other options are available for improved clock stability.

#### A. Radio Frequency front-end

Analog Device's AD9361 is a highly integrated dual-channel RF transceiver originally designed for use in 3G and 4G base station applications. The AD9361's local oscillator can operate from 70 MHz to 6.0 GHz, and channel bandwidths from less than 200 kHz to 56 MHz are supported. The two independent direct conversion receivers have state-of-the-art noise figure and linearity. Each receive subsystem includes independent automatic gain control (AGC), direct current offset correction, quadrature correction, and digital filtering, thereby eliminating the need for these functions in the digital baseband processing. Two high dynamic range analog-to-digital converters (ADCs) per channel digitize the received I

and Q signals and pass them through configurable decimation filters and 128-tap finite impulse response filters to produce a 12-bit output signal at the appropriate sample rate.

Due to the applications for which this integrated circuit was designed for (that is, multi-antenna communication systems), the AD9361 features two receiving channels but *it does not allow tuning each channel at a different frequency band*. This means that, as the ADRV9361-Z7035 SoM is shipped, it cannot be used as a stand-alone front-end for a dual-band GNSS receiver.

The proposed solution consists of setting one of the AD9361's transmission channels to generate a CW signal tuned at the difference between L1/E1 and L5/E5a center frequencies (that is, 398.97 MHz), and then use that signal to shift the L1/E1 band down to the L5/E5a center frequency. Hence, both AD9361 transmission channels can work at the same center frequency (that is,  $f_{L5} = 1176.45$  MHz), while the whole system is still acting as a dual-band RF front-end. As shown in Figure 4, only two additional external components are required. Note that mixers tend to have a high noise figure (over 10 dB), so a low noise amplifier (LNA) is required at the input of the mixer in order to keep the overall noise figure low. This supports the choice of bringing L1 down to L5 (instead of the other way around), since it is much easier and cheaper to find LNAs for L1 than for L5 in the market. A Linear Technology's LTC5510 mixer and a Maxim's MAX2670 LNA were used in the prototype.

The CW signal generated by the AD9361 in one of its transmission channels is: *i*) phase controlled with respect to the receiver chain, with granularity of 1 degree; *ii*) amplitude controlled, up to 0 dBm; *iii*) generated by a 16-bit digital-to-analog converter (DAC), providing excellent spurious-free dynamic range.

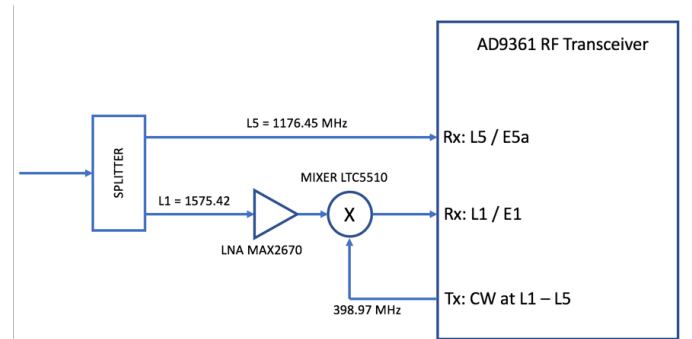


Fig. 4. Making the AD9361 a dual-band receiver.

Hence, the Radio Frequency Plan consists of a direct conversion into a baseband signal in the L5/E5a band, and an additional frequency translation step external to the IC and then direct conversion in the L1/E1 band, as summarized in Table I.

#### B. Interface to the Programmable Logic

The ADRV9361-Z7035 connects the AD9361 RF Agile Transceiver directly to the Xilinx Zynq Z7035 SoC through

TABLE I  
RADIO FREQUENCY PLAN (UNITS IN MHZ)

Band	RF	LO <sub>1</sub>	IF <sub>1</sub>	LO <sub>2</sub>	IF <sub>2</sub>
L5/E5a	1176.45	—	1176.45	1176.45	0
L1/E1	1575.42	398.97	1176.45	1176.45	0

dedicated high bandwidth data ports and clocks, an SPI control interface, and other control and framing signals. The AD9361 digital interface is comprised of two parallel data ports and several clock, synchronization, and control signals to transfer samples between the AD9361 and the Zynq SoC.

These signals can be configured as single-ended CMOS signals or as Low Voltage Differential Signal (LVDS, ANSI-644 compatible) for systems that require high speed, low noise data transfer. In LVDS mode, the interface is operated in double-data rate (DDR) mode. Therefore, 12-bit samples to/from the AD9361 are sent across two 6-bit lanes on differential pairs. The maximum rate across the Zynq-AD9361 data interface is limited by the AD9361 maximum data rate (122.88 MSps).

### C. Sample conditioning

This module handles the interface between the RF front-end's analog-to-digital converters and the processing engines in the Programmable Logic.

The first functional block in the receiver chain, labeled as "Downsampling" in Figure 5, adapts the sample rate with configurable parameters. Then a first-in first-out (FIFO) memory buffer to deliver samples to the acquisition and tracking engines. Prior to the downsampling block, the sample bit depth is adapted and the different clock domains are synchronized.

### D. Acquisition acceleration engine

This module implements the circular correlation operation, defined as:

$$R_{xd}(\tilde{\tau}, \tilde{f}_D) = \mathcal{F}^{-1} \{ \mathcal{F}\{\mathbf{x}_{IN}\} \cdot \mathcal{F}\{\mathbf{d}(\tilde{\tau}, \tilde{f}_D)\} \} \quad (1)$$

where  $\mathbf{x}_{IN}$  is a batch of  $N$  input samples taking at least one full spreading code,  $\mathbf{d}$  is a sequence of the same length containing a local replica of the spreading code, parameterized by the code delay  $\tilde{\tau}$  and Doppler shift  $\tilde{f}_D$  check parameters, and  $\mathcal{F}\{\cdot\}$  and  $\mathcal{F}^{-1}\{\cdot\}$  are the direct and inverse Fourier transforms, respectively. This operation provides the result of a linear correlation between the  $\mathbf{x}$  and  $\mathbf{d}$  sequences if and only if those sequences are periodic, so precautions are taken to properly zero pad the input sequences in order to obtain the desired output.

This approach provides some computational advantages with respect to linear correlation, since it is well known that the computational complexity of discrete Fourier transforms grows as  $\mathcal{O}(N \log N)$ , whereas the direct computation of the linear correlation has a complexity that grows as  $\mathcal{O}(N^2)$ .

The implementation in VHDL of this functionality was packaged in form of IP core.

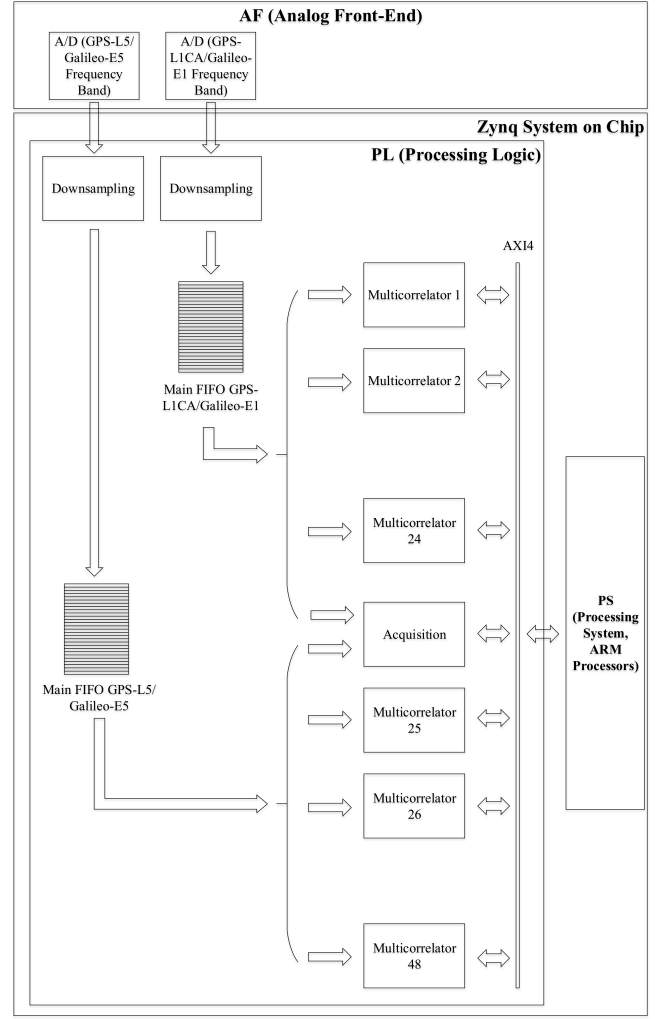


Fig. 5. The signal sample ingestion mechanism is managed by memory buffers.

### E. Tracking acceleration engines

The basic and most time-consuming functional component in a signal tracking block is the bank of correlators that correlate the incoming signal with a local replica, with one correlator centered at  $\tau = \hat{\tau}$  and others with different delays  $\epsilon$  shorter than a chip period. Those outputs are commonly known as Early, Prompt and Late correlators.

Each of those correlators multiply the input sequences for a given integration time, and then they integrate and dump the result. This operation can be described as:

$$R_{xd}[m](\hat{\tau}, \hat{f}_D, \hat{\theta}, \epsilon) = \frac{1}{K} \sum_{k=0}^{K-1} x_{IN}[k] d[k](\hat{\tau}, \hat{f}_D, \hat{\theta}, \epsilon) \quad (2)$$

where  $K$  is the number of samples within an integration time, and  $\hat{\tau}$ ,  $\hat{f}_D$  and  $\hat{\theta}$  are the previous estimations of code delay, Doppler shift and carrier phase, respectively.

The implementation in VHDL of this functionality were packaged in different versions of IP cores:



- Multicorrelator v3.0: This multicorrelator consists of three correlators (Early, Prompt and Late) and it is designed to be used with GPS L1 signals.
- Multicorrelator v3.1: This multicorrelator consists of three correlators for the pilot signal (Early, Prompt and Late), and one extra correlator for the data signal. The correlators used with the pilot component keep the receiver synchronized, and the correlator used for the data component (driven by the pilot's Prompt correlator) is used to demodulate the navigation message. There is also the option of not using the pilot signal and using three correlators in the data component. This multicorrelator is designed to be used with GPS L5 and Galileo E5a signals.
- Multicorrelator v5.1: This multicorrelator consists of five correlators for the pilot component and one correlator for the data component, with the possibility to track pilot or data components. This multicorrelator is designed to be used with Galileo E1 open service signals.

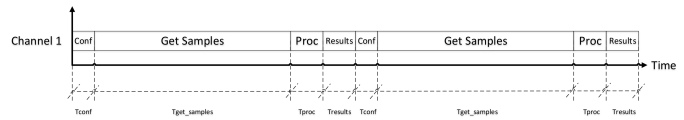


Fig. 6. Timing diagram of the Tracking process.

The GNSS baseband processing engine is based on GNSS-SDR<sup>2</sup>, a free and open source software-defined receiver written in C++. GNSS-SDR implements the baseband receiving chain for all the targeted signals, from raw signal samples up to the computation of PVT solution and the generation of GNSS products in standard formats, in a modular design that allows the easy addition of different algorithms for each of the processing blocks (namely: signal source, signal conditioner, acquisition, tracking, demodulation of the navigation message, formation of GNSS observables and PVT computation), making them interchangeable and easily configured. The source code can be built using popular and freely available compilers (such as GCC or Clang) and in a wide range of GNU/Linux and other UNIX-based operating systems, ensuring its portability to the vast majority of personal computers, including those based on i386, x86\_64 and ARM processor architectures.

### F. Interface to the Processing System

This middleware software component is in charge of provide an efficient mechanism to enable, disable, configure, and exchange information within the receiver IP cores implemented in the FPGA fabric (PL) and the receiver channels. Each FPGA-based accelerator exposes a set of read/write registers using an AXI slave bus connection with the Zynq PS. In order to reduce the computational load, a dedicated interrupt signal is routed from each FPFA module to the Zynq Generic Interrupt Controller (GIC). Since the GIC is limited to 16 interrupt signals but more than 16 satellite channels are required, an extension to the GIC controller was implemented using a Xilinx Interrupt controller IP core [9]. This allows to assign more than 60 interrupt lines, covering the interruptions of all required Acquisition and Tracking blocks. The interrupts trigger the acquisition and tracking channels callbacks in order to read the results and to reload the acquisition and tracking parameters for processing the next batch of samples. The dual-core ARM processor is clocked at 800 MHz, thus it has plenty of processor cycles to assist the interrupts from the FPGA-based accelerators, even at the shortest coherent integration time (1 ms for GPS L1/L5 and Galileo E5a).

Figure 6 shows a timing diagram of the tracking process. The tracking process cycles through the following steps:

- 1) A software executing in the ARM processor configures the Multicorrelator engines with updated parameters: number of samples, Doppler correction, etc. ( $T_{\text{conf}}$  time).
- 2) The Multicorrelator engine captures a new batch of samples from one of the main FIFOs shown in Figure 5 and processes them on the fly ( $T_{\text{get\_samples}}$ ).
- 3) The Multicorrelator engine finishes processing the received samples ( $T_{\text{proc}}$ ).

Fig. 7. Generic flow graph implemented by GNSS-SDR when executing in a regular personal computer.

All software radios can be formally modeled as acyclic flow graphs, where the nodes in the flow graph represent the processing blocks, and the arrows between them represent communication mechanisms. Then, an underlying process scheduler is in charge of transporting data from “sources” to “sinks”, ensuring a scalable spread of processes (or threads)

<sup>2</sup>See <https://gnss-sdr.org>

among the available processing cores and avoiding processing bottlenecks as much as possible [5]. In GNSS-SDR, this duty is left to GNU Radio<sup>3</sup>, a well-established open source framework for software radios.

The typical usage of GNSS-SDR in real time mode consists of a set up with a RF front-end connected to a computer via USB or Ethernet. Then, the software takes the incoming stream of signal samples as its data source and performs all the signal processing up to the generation of GNSS products. However, the Zynq device architecture imposes some changes in the GNSS-SDR receiver flow graph with respect to its execution in a personal computer. The fact that signal samples coming from the RF front-end enter directly into a memory buffer in the FPGA device breaks the regular “Signal Source → Signal Conditioning → Channels processing” chain (see Figure 7), since the signal source is no longer managed in the Processing System (*i.e.*, in the ARM processors) but in the Programmable Logic (*i.e.*, in the FPGA processor). This issue requires a redesign of the receiver’s flow graph. The proposed new design is an extension of the existing flow graph, in which:

- The Signal Source and Signal Conditioning blocks are eliminated, transferring their functionalities to the Programmable Logic;
- Acquisition blocks become non-GNU Radio blocks, but controllers of the acquisition acceleration engine inside the FPGA; they implement signal acquisition with the aid of that engine, and
- Tracking blocks are turned into signal sources that deliver a stream of data with the results of the tracking loops, which are computed with the aid of the tracking acceleration engines.

These new blocks are then chained to the rest of the GNSS-SDR processing blocks (extraction of navigation message, computations of observables and PVT solution), thus directly reusing the existing implementations in the Processing System. Figure 8 shows a diagram of the proposed solution.

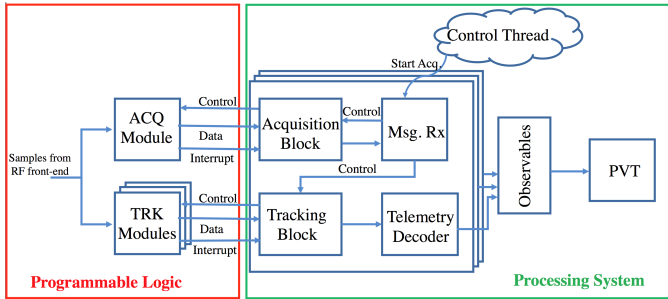


Fig. 8. Interplay within the GNSS-SDR flow graph and the IP Cores running in the FPGA.

This solution requires very few changes in the original GNSS-SDR software architecture. An extension of the existing flow graph, and specific new implementations for the Acquisition and Tracking blocks, are the only required additions

in order to accommodate the FPGA off-loading mechanism. The solution reuses most of GNSS-SDR’s existing software architecture and processing blocks, thus benefiting from an open source implementation with a growing base of users and developers. The fact that FPGA acceleration engines implement basic operations used by most of the related algorithms makes them reusable in a wide range of possible acquisition and tracking implementations.

#### H. Operating system and development environment

The operating system and cross-compilation environment are based on OpenEmbedded, a building framework for embedded GNU/Linux. OpenEmbedded offers a cross-compiling environment, allowing to create a completely customized GNU/Linux distribution for embedded systems, where all packages are specified by their release version, git commit or tag. It also allows the creation of a Software Development Kit (SDK) for cross-compilation<sup>4</sup>.

#### IV. PROOF-OF-CONCEPT PROTOTYPE IMPLEMENTATION

In order to facilitate the development, debugging and testing of the GNSS receiver, the prototype makes use of a ADRV1CRR-BOB breakout carrier board (see Figure 9), which implements suitable standard connectors for power supply, Ethernet communications, USB and JTAG programming interfaces, among other debugging features. In addition, the carrier provides multiple access points to supply external I/O bank voltages and measure power consumption of the SoM.

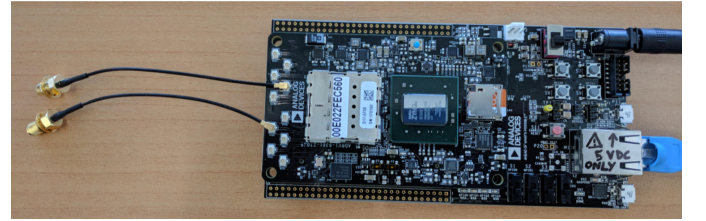


Fig. 9. Picture of the ADRV9361-Z7035 System-on-Module plugged onto the ADRV1CRR-BOB breakout board.

All the GNSS receiver components, including an input antenna RF connector, a bias-T for active antenna feeding, an L-band passive splitter/combiner, the LNA and the mixer described in Section III-A, and power connectors and switches, were packaged together in a plastic box, as shown in Figure 10 and Figure 11, allowing easier handling and transportation of the prototype. The box was manufactured using a custom design and a 3D printer.

#### V. PERFORMANCE ASSESSMENT METHODOLOGY

The proposed receiver must undergo the same types of test to those applied to IC-based receivers, including typical performance measurements such as the Time To First Fix (TTFF), acquisition and tracking sensitivity, robustness of decision thresholds in the presence of CW interferences, pseudorange and carrier phase error measurements, and PVT

<sup>3</sup>See <https://www.gnuradio.org>

<sup>4</sup>SDK available at <https://gnss-sdr.org/docs/tutorials/cross-compiling/>

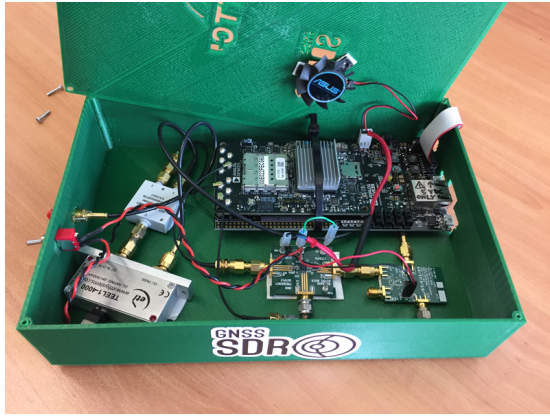


Fig. 10. Inside view of the laboratory prototype.



Fig. 11. Picture of the laboratory prototype.

solution accuracy, all in space-like scenarios, and to provide results in standard metrics. However, as discussed in [10], the nature of software-defined radio technology admits a broader quality assessment and testing approach. A GNSS receiver in which the baseband processing chain is implemented in software and executed by a general-purpose processor in a computer system has other additional features and metrics not captured by *traditional* GNSS testing procedures. Embracing software industry testing procedures and best practices, the automation of unit, component, integration and system testing, and the use of Continuous Integration systems help to ensure system's reliability upon the addition of changes in the source code.

For instance, the receiver is able to act as a sample recording system. Hence, all kind of realistic RF scenarios for LEO, MEO, HEO and GEO dynamics can be generated using for instance a Spirent's GSS9000 Multi-Frequency, Multi-GNSS RF Constellation Simulator [11], captured in data files by the receiver, and then replayed and post-processed using exactly the same processing chain than the one used in real-time mode. This allow fine tuning of parameters under fair conditions in post-processing mode, and the experimentation with new algorithms or navigation concepts in an easy and cost-effective way.

The automation of testing procedures provides important practical benefits. Metrics such as acquisition and tracking sensitivity greatly depend on the specific RF front-end, so its automation makes the evaluation of other RF front-ends for GNSS receiving purposes easier. It also allows the rapid impact evaluation of changes in the source code in system-level metrics.

Some of the implemented testing procedures are:

- The TTFF indicator provides a measurement of the time required by the receiver to provide a valid position fix after it is started. A valid position fix is required to have a 3D accuracy below a given threshold. The value includes the time to recover ephemeris data from all satellites used in the navigation estimation process. After defining a cold or warm start, each TTFF sample is computed as the time interval starting with the invocation of the receiver's executable and ending with the first valid navigation data point derived from live or simulated satellite signals. The start times of the test samples should not be synchronized to any UTC time boundary and should be randomly spread within the 24 hour UTC day and within the GNSS data collection interval (e.g., 30 s for GPS L1 C/A signals). For a given sample size, the mean, standard deviation, maximum and minimum values of the measured TTFFs are reported.
- Acquisition sensitivity determines the minimum signal power threshold that allows the receiver to successfully perform a cold start TTFF within a specified time frame. The generation of testing inputs is as follows: fixing the number of visible satellites to one, the power level of the received signal is set such that the GNSS software receiver under test can detect the single GNSS satellite signal within a given probability of detection. The power level of the GPS satellite signal is then decreased until the GNSS receiver is not able to acquire that satellite signal. This power level and the corresponding reported carrier-to-noise density ratio ( $C/N_0$ ) are collected as data.
- Tracking sensitivity refers to the minimum signal level that allows the receiver to maintain a location fix within some specified degree of accuracy. The generation of testing inputs is as follows: fixing the number of visible satellites to eight, the power level of the received signals is set such that the GNSS software receiver under test can acquire, track and demodulate the navigation message of four or more signal and obtain position fixes. The power level of the GNSS satellite signals is then decreased by 1 dB at 60 s intervals until the position fix is lost. The reported  $C/N_0$  and the corresponding signal power level are collected as data.
- Other tests include pseudorange, Doppler shift and carrier phase estimation error measurements, automatic measurement of tracking loops' pull-in range, phase and code inter-channel bias, computation of position and velocity error metrics, etc.

GNSS-SDR's quality assessment code is based on Google's



C++ Testing Framework (usually referred to as “Google Test”), which allows to rapidly implement tests that are automatically run without requiring human intervention, deterministic, independent, repeatable and portable. The software repository contains more than 200 unit tests and several system tests can be easily re-executed each time there is a change in the source code, allowing the continuous improvement of the software-receiver in a safe and controlled way.

As a example, results in this section show the performance of the Observables block, which is in charge of obtaining pseudorange (in m), carrier phase (in cycles) and Doppler shift (in Hz) measurements for each of the tracked signals. The testing setup is shown in Figure 12.

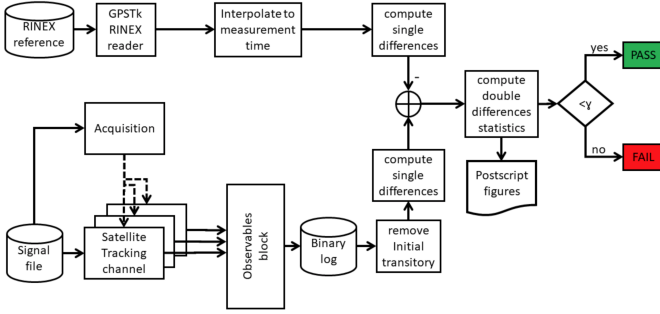


Fig. 12. Block diagram of the Observables test.

From left to right in Figure 12, the required input dataset is composed of a baseband signal binary capture file and a reference RINEX file that contains the observables to be used as the expected output of the Observables block. Once the visible satellites have been acquired, the signal file reader is restarted and every acquired satellite is assigned to a new tracking channel. Each tracking channel is composed of a Tracking block and a Telemetry decoder block. The output of each Telemetry decoder block is connected to the Observables block in parallel. The Observables block was configured to enable the binary log option, thus producing a binary log file with the observables to be read after the signal processing step. Finally, the GNU Radio flow graph is started and the unit test waits for the complete process of the number of seconds of signal specified by a test command line parameter.

After the signal processing step, the test reads the produced observables from the binary log file. The reference RINEX observables are read by using the GPSTk<sup>5</sup> library. Reference observables are then aligned in time with the measured epochs by using a linear interpolation.

Then, the double difference algorithm is applied. This is a well-known method to compare the observables of two different receivers. Considering a simplified pseudorange model for the  $m$ -satellite path:

$$\rho_m = r_m + c\Delta t_{\text{atm}}^m + c(\Delta t_u - \Delta t_s^m) \quad (3)$$

where  $r$  is the true range,  $c$  is the speed of light,  $t_{\text{atm}}^m$  is the atmospheric delay for the  $m$ -th satellite,  $\Delta t_u$  is the receiver

clock offset, and  $\Delta t_s^m$  is the  $m$ -th satellite clock offset. By differencing the pseudoranges of two satellites  $m$  and  $n$ , received by a single receiver, one obtains

$$\Delta\rho_{m-n} = r_m - r_n + c(\Delta t_{\text{atm}}^m - \Delta t_{\text{atm}}^n) + c(\Delta t_s^m - \Delta t_s^n) \quad (4)$$

where the user clock offset has been cancelled. If two receivers  $A$  and  $B$  are used, sharing the same antenna, it is possible to compute the double difference  $\Delta\Delta\rho_{m-n}$  as follows:

$$\Delta\Delta\rho_{m-n} = \Delta\rho_{m-n}^A - \Delta\rho_{m-n}^B = 0 \quad (5)$$

where the expected output should be equal to zero because the baseline is zero. Then, only thermal noise should appear in that measure, since all other common sources of error have been cancelled out.

The double difference technique is also valid for the accumulated carrier phase and for the carrier Doppler observable with the same expected result. In the case of the accumulated carrier phase is necessary to remove the carrier phase offset at the starting point of both the reference carrier phase and the measured one, because the start of operation of each receiver is not synchronized. Notice that the double difference of the carrier Doppler observable also removes the front-end carrier frequency offset between receivers.

Results shown below were obtained from signals generated for a Low Earth Orbit scenario, sampled at 12.5 MSps and 8 bits per complex sample and captured in files. This results were obtained from signals with a duration of 100 s. In addition to the RF signals, the GSS9000 also provides reference files in RINEX format which are used to obtain the *true* values of the estimated parameters, and hence compute the error in such estimations. This direct comparison is not possible in the L5 band, since data in RINEX files generated by the GSS9000 are referred to the whole E5 band (centered at 1191.795 MHz) instead of to the L5/E5a band (centered at 1176.45 MHz). Figures 13 to 15 show the histograms of the obtained measurements.

TABLE II  
RMSE OBTAINED IN OBSERVABLES FOR A LEO SCENARIO

Signal	RMSE [m] Pseudorange	RMSE [cycles] Carrier Phase	RMSE [Hz] Doppler shift
GPS L1	2.27	2.15	8.731
Galileo E1 OS	1.14	0.80	2.01
GPS L5	0.50	N/A	0.60
Galileo E5a	0.34	N/A	N/A

Results presented here are obtained from a Continuous Integration system each time a new change is uploaded to the upstream source code repository, and hence relevant metrics are continuously evaluated, bugs are early detected and improvements can be objectively quantified and integrated in the source code.

<sup>5</sup>University of Texas at Austin. See <https://github.com/SGL-UT/GPSTk>



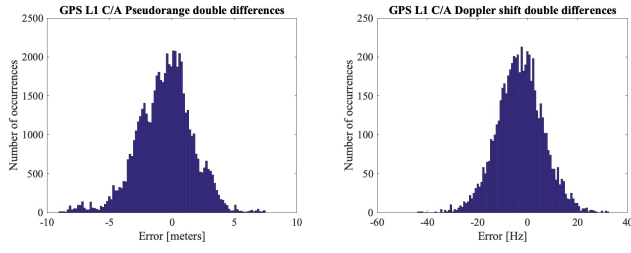


Fig. 13. Double difference errors for a GPS L1 C/A signal.

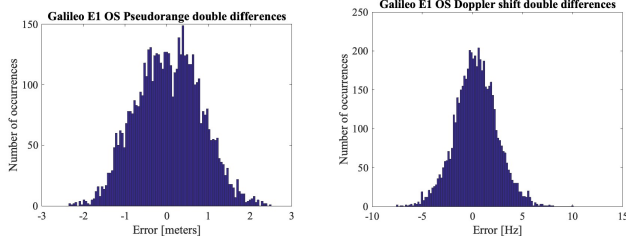


Fig. 14. Double difference errors for a Galileo E1 OS signal.

## VI. CONCLUSIONS

This paper reported the design, proof-of-concept implementation and preliminary performance assessment methodology of a low-cost, software-defined spaceborne GNSS receiver, providing practical details of a working prototype. The presented approach takes advantage of the flexibility of software-defined radio technology and the forthcoming availability of radiation-hardened, space-certified Systems-on-Module to implement a fully customizable receiver with the capability to process GNSS signals in real-time under realistic space-like scenarios and to deliver GNSS products in standard formats. The baseband processing engine is publicly available and released under a free and open source license.

## REFERENCES

- [1] J. T. Curran, C. Fernández-Prades, A. Morrison, and M. Bavaro, "Innovation: The continued evolution of the GNSS software-defined radio," *GPS World*, vol. 29, no. 1, pp. 43–49, Jan. 2018.
- [2] J. Roselló, P. Silvestrin, G. López Risueño, R. Weigand, J. V. Perelló, J. Heim, and I. Tejerina, "AGGA-4: Core device for GNSS space receivers of this decade," in *Proc. of the 5th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, ESA/ESTEC, Noordwijk, The Netherlands, Dec. 2010, pp. 1–8, doi: 10.1109/NAVITEC.2010.5708068.

- [3] J. Roselló, P. Silvestrin, R. Weigand, S. d'Addio, A. García Rodríguez, and G. López Risueño, "Next generation of ES's GNSS receivers for Earth Observation satellites," in *Proc. of the 6th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, ESA/ESTEC, Noordwijk, The Netherlands, Dec. 2012, pp. 1–8, doi: 10.1109/NAVITEC.2012.6423104.
- [4] C. Fernández-Prades, J. Vilà-Valls, J. Arribas, and A. Ramos, "Continuous reproducibility in GNSS signal processing," *IEEE Access*, vol. 6, no. 1, pp. 20 451–20 463, Apr. 2018, doi: 10.1109/ACCESS.2018.2822835.
- [5] C. Fernández-Prades, J. Arribas, and P. Closas, "Accelerating GNSS software receivers," in *Proc. 29th Int. Tech. Meeting Sat. Div. Inst. Navig.*, Portland, OR, Sep. 2016, pp. 44–61, ISSN: 2331-5911.
- [6] Avnet, "PicoZed SDR 2x2 System-on-Module User Guide," Sep. 2016, Version 1.7.
- [7] Analog Devices Inc., "AD9361 Reference Manual," Norwood, MA, Oct. 2015, UG-570. Rev. A.
- [8] Xilinx, "Zynq-7000 All Programmable SoC Technical Reference Manual," Oct. 2017, UG585 (v1.12).
- [9] —, "AXI Interrupt Controller (INTC) v4.1 LogiCORE IP Product Guide," Oct. 2017, Vivado Design Suite PG099.
- [10] C. Fernández-Prades, J. Arribas, and P. Closas, "Assessment of software-defined GNSS receivers," in *Proc. of the 8th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, ESA/ESTEC, Noordwijk, The Netherlands, Dec. 2016, pp. 1–9, doi: 10.1109/NAVITEC.2016.7931740.
- [11] Spirent Communications PLC., "Spirent GSS9000 Constellation Simulator," Datasheet, Jul. 2015, available online at [https://www.spirent.com/-/media/Datasheets/Positioning/GSS9000\\_Specifications.pdf](https://www.spirent.com/-/media/Datasheets/Positioning/GSS9000_Specifications.pdf), Accessed: December 10, 2018.

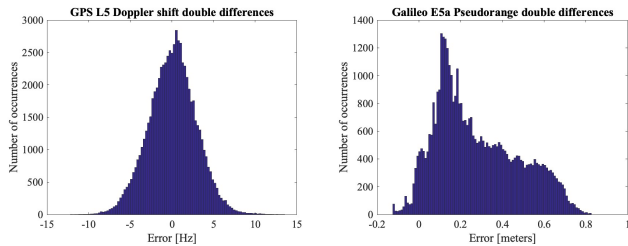


Fig. 15. Double difference errors for a Galileo E5a and GPS L5 signals.