



AUDITOR – GA 687367

Advanced Multi-Constellation EGNSS Augmentation and Monitoring Network and its Application in Precision Agriculture

D3.3 Version 1.1

GNSS receiver module validation

Contractual Date of Delivery:	<i>M21</i>
Actual Date of Delivery:	M30 (16-07-2018)
Editor:	Microsoft Office User
Author(s):	Microsoft Office User, Javier Arribas, Esther López, Jacobo Dominguez, Alberto García Rigo, Manuel Hernández-Pajares.
Work package:	<i>WP3 – GNSS receiver module</i>
Security:	PU
Nature:	R
Version:	1.1
Total number of pages:	40

Abstract:

This document reports the GNSS receiver module validation performed in WP3 of Project AUDITOR.



Document Control

Version	Details of Change	Author	Approved	Date
1.0	Document finalized	CF	EL	16/07/18
1.1	Minor format issues corrected	JD	EL	17/07/18

Executive Summary

AUDITOR's targeted receiver module features a dual-band (one centered at 1176.45 or 1227.60 MHz and the other at 1575.42 MHz) GNSS receiver with the ability to process GPS L1 C/A, Galileo E1b/c, GPS L2C 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 8 channels per signal. For the presented proof-of-concept, the targeted sampling rate was 8 MSps.

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/N₀), 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).

In order to facilitate the development, debugging and testing of the GNSS receiver, the prototype was packaged in a convenient plastic box, implementing suitable standard connectors for power supply, Ethernet communications, USB and JTAG programming interfaces, among other debugging features. All the GNSS receiver components, including an input antenna RF connector, a bias-T for active antenna feeding, an L-band passive splitter/combiner, LNAs, and power connectors and switches, are available in a portable device, allowing easier handling and transportation of the prototype. The box was manufactured using a custom design and a 3D printer (see Figure 1).

This document reports the validation procedures applied to the developed prototype.



Figure 3.1 Picture of AUDITOR's prototype

Authors

Partner	Name	e-mail
CTTC	Carles Fernández-Prades	cfernandez@cttc.cat
	Javier Arribas	jarribas@cttc.cat
ACORDE	Esther López	esther.lopez@acorde.com
	Jacobo Domínguez	jacobo.dominguez@acorde.com
UPC	Alberto García Rigo	alberto.garcia.rigo@upc.edu
	Manuel Hernández-Pajares	manuel.hernandez@upc.edu

Table of Contents

Document Control	2
Executive Summary	3
Authors	5
Table of Contents	6
List of Tables	6
List of Figures.....	7
List of Acronyms and Abbreviations.....	8
1. Analysis of indicators	9
2. Validation of GNSS RF Front-End Module.....	22
3. Validation of the software-defined GNSS receiver	23
3.1 Signal Tracking	23
3.2 Tracking pull-in test.....	23
3.2.1 Test results	23
3.3 Tracking steady-state test.....	27
3.3.1 CRB for the Code Delay	27
3.3.2 Test results	29
4. Analysis of main Key Performance Indicators	33
5. Conclusion.....	36

List of Tables

Table 1.1: Indicators analysis	9
Table 3.1: Tracking pull-in test flags.....	23
Table 3.2: Tracking steady-state test	29
Table 4.1: KPI 1 - Accuracy	33
Table 4.2: KPI 2 - Availability	33
Table 4.3: KPI 3 - Efficiency.....	33
Table 4.4: KPI 4 - Interoperability.....	34
Table 4.5: KPI 5 - Repeatability.....	34

List of Figures

Figure 3.1: Tracking pull-in results for $CN0= 45$ dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.	25
Figure 3.2: Tracking pull-in results for $CN0= 42$ dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.	25
Figure 3.3: Tracking pull-in results for $CN0= 39$ dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.	26
Figure 3.4: Tracking pull-in results for $CN0= 36$ dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.	26
Figure 3.5: Tracking pull-in results for $CN0= 33$ dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.	27
Figure 3.6: Tracking Doppler estimation error and variance for PLL/DLL bandwidths of 5 Hz and 0.25 Hz respectively.	30
Figure 3.7: Accumulated carrier phase estimation error and variance for PLL/DLL bandwidths of 5 Hz and 0.25 Hz respectively.....	31
Figure 3.8: Code Phase estimation error and variance for PLL/DLL bandwidths of 5 Hz and 0.25 Hz respectively.	31
Figure 3.9: Measured RMSE and corresponding Cramér-Rao Bound for the Code Delay estimation. The PLL filter was set to 1 Hz for $T_{int}=20$ ms and 10 Hz for $T_{int}=1$ ms..	32

List of Acronyms and Abbreviations

Term	Description
ADC	Analog to Digital Converter
CEP	Circular Error Probability
TTFF	Time To First Fix
DAC	Digital to Analog Converter
DRMS	Distance Root Mean Square
FE	Front End
FGPA	Field Programmable Gate Array
FMC	FPGA Mezzanine Card
GIS	Geographic Information System
GNSS	Global Navigation Satellite Systems
IF	Intermediate Frequency
I&Q	Inphase & Quadrature
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
LNA	Low Noise Amplifier
MRSE	Mean Radial Spherical Error
NMEA	National Marine Electronics Association
PGA	Programmable Gain Amplifier
RINEX	Receiver Independent Exchange Format
RTCM	Radio Technical Commission for Maritime Services
SEP	Spherical Error Probable
SPI	Serial Peripheral Interface
TTFF	Time To First Fix
VCO	Voltage Controlled Oscillator
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
XML	eXtensible Markup Language
WGS84	World Geodetic System 1984





1. Analysis of indicators

This section summarizes the status of the GNSS receiver's quality indicators identified in Deliverable 1.3.

Table 1.1: Indicators analysis

Indicator	Requirements	Current status	Pass / Fail
Definition of project's geographical coordinate frame and geodetic datum.	Position must be expressed in an adequate coordinate reference system and geodetic datum.	The World Geodetic System (WGS84) is a standard for use in cartography, geodesy, and navigation (including GPS and Galileo). All computations inside the GNSS receiver device are referred to WGS84. Differential systems can perform transformations to express position in other datums.	✓
Definition of accuracy metrics and procedures.	Definition of metrics for 2D and 3D positioning. Detailed definition of measurement procedures.	AUDITOR accuracy metrics: <ul style="list-style-type: none"> for 2D positioning: Distance Root Mean Square (DRMS) and the Circular Error Probability (CEP), in meters, and, for 3D positioning: the Mean Radial Spherical Error (MRSE) and the Spherical Error Probable (SEP), in meters. 	✓
Meet AUDITOR's static positioning accuracy requirements.	Decimeter-level static accuracy	Testing software implemented.	✗
Meet AUDITOR's dynamic positioning accuracy requirements.	Decimeter-level dynamic accuracy	Testing software implemented.	✗
24/7 service	24/7 service	The receiver delivers GNSS products continuously for more than a week (observed, no theoretical limit).	✓
Typical duration of usage session	12 hours	The receiver delivers GNSS products continuously for more than a week (observed, no theoretical limit).	✓

Acquisition sensibility	Comparable to that of professional receivers	Measurement procedures defined in https://gnss-sdr.org/design-forces/availability/	✓
Tracking sensibility	Comparable to that of professional receivers	Measurement procedures defined in https://gnss-sdr.org/design-forces/availability/ .	✓
Time to First Fix	Definition of detailed TFF measurement procedures in different receiver's status.	Cold start TFF measurement procedures defined in https://gnss-sdr.org/design-forces/availability/ Cold start testing software has been implemented (see https://gnss-sdr.org/docs/tutorials/testing-software-receiver-2/)	✓
		Warm start TFF measurement procedures defined in https://gnss-sdr.org/design-forces/availability/ Warm start testing software has been implemented (see https://gnss-sdr.org/docs/tutorials/testing-software-receiver-2/)	✓
		Hot start TFF measurement procedures defined TFF measurement procedures defined in https://gnss-sdr.org/design-forces/availability/ Hot start testing software has been implemented (see https://gnss-sdr.org/docs/tutorials/testing-software-receiver-2/)	✓
Reacquisition	Implemented	Controlled by lock detector parameters. See https://gnss-sdr.org/docs/sp-blocks/tracking/	✓
Number of parallel channels that the software receiver can sustain in real time, given the targeted GNSS signal of each channel and the computational resources available for signal processing.	8 channels per signal	8 channels per signal	✓

Power consumption for a given host computer and computational load in terms of number of signals and channels to be processed.	Power consumption < 10 W	Not measured.	
Availability of profiling tools for identifying processing bottlenecks and measuring efficiency.	Allow for statistical profiling tools. Allow for CPU profiling tools. Allow for instrumenting profiler tools. Statistical execution time measurement tool available for the supported processing platforms.	GNSS-SDR allows for the usage of several software profiling tools, see http://gnss-sdr.org/documentation/how-profile-code for some examples. The suggested approach consists of using a set of freely available profiling tools that use different techniques, in the hope of taking advantage of their complementary nature and obtain a better insight about how the code is performing.	
Possibility to either use synthetically generated or real-life GNSS signals.	-	Implemented	
Possibility to process signals either in real time or in post-processing time (only limited by the computational capacity of the host machine).	-	AUDITOR's software receiver architecture makes use of a thread-per-block strategy and GNU Radio's task scheduler, which manages a flow graph of nodes. Each node represents a signal processing block, whereas links between nodes represents a flow of data. Under this scheme, software-defined signal processing blocks read the available samples in their input memory buffer(s), process them as fast as they can, and place the result in the corresponding output memory buffer(s). This strategy results in a software receiver that always process signal at the maximum processing capacity, regardless of its input data rate. Achieving real-time is <i>only</i> a matter of executing the full receiver's processing chain in a processing system powerful enough to sustain the required processing load, but it does not prevent from executing exactly the same processing at a slower pace, for example by reading	

		samples from a file, in a less powerful platform.	
Possibility to use different radio frequency front-ends.	-	AUDITOR's software receiver configuration system allows the selection of UHD and OsmoSDR compatible RF front-ends, in addition to AUDITOR's front-end.	✓
Possibility to define custom receiver architectures.	-	Implemented	✓
Possibility to easily define / interchange implementations and parameters for each processing block.	-	AUDITOR's software receiver configuration system allows for an unlimited number of block implementations and number of parameters for each particular implementation.	✓
Possibility to deploy different receiver architectures and components.	-	Implemented	✓
Possibility to allow for unit/component/integration/system testing.	-	Implemented	✓
Possibility to be executed in different processing platforms (mainframes, personal computers, embedded systems, etc).	-	AUDITOR's software receiver building system, based on CMake, currently supports i386, x86_64/amd64, armhf and arm64 processor architectures.	✓
Flexible configuration mechanism.	<p>The software defined receiver must be fully configured in a single file.</p> <p>Configuration system must be arbitrarily extendable and easy to use.</p> <p>Allow possibility of overriding parameters via commandline flags.</p> <p>Required tools/dependencies released under open source license.</p>	<p>AUDITOR's GNSS software defined receiver follows a single-file configuration strategy. Properties are passed around within the program using a configuration interface class. Classes that need to read configuration parameters will receive instances of such interface class from where they will fetch the values. The name of these parameters can be anything but one reserved word: implementation. This parameter indicates in its value the name of the class that has to be instantiated by the processing block factory for that role. Since the configuration is just a set of property names and values without any meaning or syntax, the</p>	✓

		<p>system is very versatile and arbitrarily extendable. Adding new properties to the system only implies modifications in the classes that will make use of these properties. In addition, the configuration files are not checked against any strict syntax so it is always in a correct status (as long as it contains pairs of property names and values in the INI format, see https://en.wikipedia.org/wiki/INI_file)</p> <p>For commandline flags management, AUDITOR's GNSS software defined receiver relies on gflags (formerly Google Commandline Flags), see https://github.com/gflags/gflags.</p> <p>Gflags is the commandline flags library used within Google, and differs from other libraries in that flag definitions can be scattered around the source code, and not just listed in one place such as main(). In practice, this means that a single source-code file will define and use flags that are meaningful to that file. Any application that links in that file will get the flags, and the gflags library will automatically handle that flag appropriately. There is significant gain in flexibility, and ease of code reuse, due to this technique. Gflags is released under the BSD 3-clauses license.</p>	
"Operation modes"	Single / dual band. Second band selectable from L2 and L5.	Implemented	✓
GNSS Signals	Signal acquisition, tracking, decoding of the navigation message and generation of code and phase observables.	GPS L1 C/A	✓
		GPS L2C(M)	✓
		GPS L5	✓
		Galileo E1b/c	✓

		Galileo E5a	✓
		Galileo E5b still not implemented. Band not covered by the RF front-end.	✗
	Signal acquisition, tracking, and decoding of navigation message	EGNOS still not fully implemented.	✗
RF frontend drivers	The GNSS software receiver must receive data at an adequate bandwidth and sampling frequency.	AUDITOR's RF front-end	✓
		USRP family	✓
		OsmoSDR-compatible	✓
Input data types for raw samples	AUDITOR's GNSS software defined receiver must allow for most usual input raw sample data types (i.e. bit length, integer and floating point encodings) delivered by available GNSS radio-frequency front-ends' analog to digital converters and associated software drivers.	AUDITOR's raw sample data type	✓
		Reading samples represented by 2 bits (sign and magnitude).	✓
		Reading real (IF) samples represented by 1 byte (8-bit signed integer)	✓
		Reading real (IF) samples represented by 1 <i>short</i> (16-bit signed integer)	✓
		Reading real (IF) samples represented by 1 <i>float</i> (32-bit floating point)	✓
		Reading I&Q (IF or baseband) interleaved samples represented by 1 byte (8-bit signed integer)	✓
		Reading I&Q (IF or baseband) interleaved samples represented by 1 <i>short</i> (16-bit signed integer)	✓
		Reading complex (baseband) samples represented by 1 byte (8-bit signed integer) each component.	✓
		Reading complex (baseband) samples represented by 1 <i>short</i> (16-bit signed integer) each component.	✓

		Reading complex (baseband) samples represented by 1 <i>float</i> (32-bit floating point) each component.	✓
Output formats	The “output products” of the GNSS software defined receiver (position, observables, navigation data) must be delivered in (preferably open) standards in order to maximize interoperability with other software packages.	RINEX v2.11 and 3.02 file generation (obs and nav). RINEX (Receiver Independent Exchange Format) is an interchange format for raw satellite navigation system data, covering observables and the information contained in the navigation message broadcast by GNSS satellites. This allows the user to post-process the received data to produce a more accurate result (usually with other data unknown to the original receiver, such as better models of the atmospheric conditions at time of measurement).	✓
		RTCM SC-104 provides standards that define the data structure for differential GNSS correction information for a variety of differential correction applications. Developed by the Radio Technical Commission for Maritime Services (RTCM), they have become an industry standard for communication of correction information. GNSS-SDR implements RTCM version 3.2, defined in [RTCM 10403.2].	✓
		NMEA 0183 is a combined electrical and data specification for communication between marine electronics such as echo sounder, sonars, anemometer, gyrocompass, autopilot, GPS receivers and many other types of instruments. It has been defined by, and is controlled by, the U.S. <u>National Marine Electronics Association</u> . At the application layer, the standard also defines the contents of each sentence (message) type, so that all listeners can parse messages accurately. Those messages can be sent through the serial port (that could be for instance a Bluetooth link) and be used/displayed by a number of software applications such as <u>gpsd</u> , <u>JOSM</u> , <u>OpenCPN</u> , and many others	✓

		(and maybe running on other devices).	
		GeoJSON is a geospatial data interchange format based on JavaScript Object Notation (JSON) supported by numerous mapping and GIS software packages, including OpenLayers , Leaflet , MapServer , GeoServer , GeoDjango , GDAL , and CartoDB . It is also possible to use GeoJSON with PostGIS and Mapnik , both of which handle the format via the GDAL OGR conversion library. The Google Maps Javascript API v3 directly supports the integration of GeoJSON data layers , and GitHub also supports GeoJSON rendering . Format specification freely available at http://geojson.org/geojson-spec.html	✓
		KML (Keyhole Markup Language) is an XML grammar used to encode and transport representations of geographic data for display in an earth browser. KML is an open standard officially named the OpenGIS KML Encoding Standard (OGC KML), and it is maintained by the Open Geospatial Consortium, Inc. (OGC). KML files can be displayed in geobrowsers such as Google Earth , Marble , osgEarth , or used with the NASA World Wind SDK for Java . Open standard freely available at http://www.opengeospatial.org/standards/kml	✓
UNIX-friendly	The binary file that executes the software receiver must be system-wide installable. Configuration files must be in plain text format and human-readable. Executable must accept commandline flags.	Implemented	✓
Source code under a version control system.	Available under open source license. Free public access to the source code repository.	AUDITOR uses Git as a distributed version control system and GitHub as the Git server hosting service. See https://github.com/gnss-sdr/gnss-sdr	✓

		Git is available under the GNU General Public License v2.	
Automated documentation system.	Usable in C++, Python and VHDL source code. Available under open source license, and in all supported environments. Easily maintainable.	AUDITOR uses Doxygen (http://www.stack.nl/~dimitri/doxygen/) the <i>de facto</i> standard tool for generating documentation from annotated C++ sources, but it also supports other programming languages such as C, Python, and VHDL. The documentation is written within code and is thus relatively easy to keep up to date. Doxygen can cross reference documentation and code, so that the reader of a document can easily refer to the actual code. Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically. Doxygen is highly portable and can generate documentation in HTML and PDF formats. Available under GNU General Public License v2.	✓
Automated build environments.		Provided by GitLab (https://gitlab.com/gnss-sdr/gnss-sdr)	✓
Availability of “debugging modes” and tools.		Provided via CMake.	✓
Static code analysis		Provided by Coverity Scan (https://scan.coverity.com/)	✓
Dynamic code analysis		Provided by Valgrind (http://valgrind.org/)	✓
Definition of a source tree structure		Described in the README file at the root of the source code tree.	✓
Availability of a coding style guide.		Coding guidelines and conventions can be found at: https://gnss-sdr.org/documentation/coding-style	✓

Supported processor architectures		AUDITOR's processor architecture	✓
		i386 processor architecture supported by GNSS-SDR	✓
		x86_64 / amd64 processor architecture supported by GNSS-SDR.	✓
		armhf processor architecture supported by GNSS-SDR.	✓
		arm64 processor architecture supported by GNSS-SDR.	✓
Usage of a well-established building tool	Available for all the supported processor architectures, and under an open source license.	AUDITOR uses CMake (https://cmake.org) as a building tool for its GNSS software defined receiver. Available under BSD 3-Clause license.	✓
Availability of software package	AUDITOR's software defined receiver should be easily built and installed, ideally requiring one single line in the user terminal.	GNSS-SDR now forms part of Debian and Ubuntu releases. In addition, the software receiver is also available as a Docker container. See https://github.com/carlesfernandez/docker-gnssdr	✓
Freely available, industry-proven software compilers.	Usable in the processor architectures and Operating Systems described above. Available under open source license.	AUDITOR's software defined receiver can use GCC (available under the GNU General Public License v3) and LLVM (available under University of Illinois/NCSA Open Source License)	✓
Available for most popular (Unix-based) operating system distributions.	AUDITOR's software defined receiver must be compilable and executable (including the availability of all required dependencies in a given environment).	Debian 8 and above (32 and 64 bits)	✓
		Ubuntu 14.04 LTS and above (32 and 64 bits)	✓
		Linaro 15.03 and above	✓
		Apple's Mac OS X 10.9 and above	✓
Source code released under an open license.		GNSS-SDR is released under the GNU General Public License v3, as specified in the COPYING file at the root of the source code tree (as it is standard practice in the discipline). It can be	✓

		checked online at https://github.com/gnss-sdr/gnss-sdr	
Unique identifier for source code snapshots.	Every single change in the source tree (either on the reference development branch or in any other) must be uniquely identified and retrievable, keeping an annotated history of source code evolution.	Git assigns a 40 character-long identifier to every revision (i.e, specific snapshot of the status of every single file present in the repository), which is the output of the SHA-1 algorithm applied to a set of information required to recreate the full source tree. Hence, every single bit change in the source code is registered by Git, with the added benefit of <i>integrity</i> over the source code identification.	✓
Availability of a Digital Object Identifier for GNSS-SDR releases	Automated and persistent DOI assignment to GNSS-SDR stable releases.	Every new release of GNSS-SDR will obtain a new DOI. Automated DOI assignment already set-up, service freely provided by ZENODO and GitHub.	✓
Quasi-linear acceleration with the number of cores/processors	-	Achieved. See: C. Fernández-Prades, J. Arribas and P. Closas, "Accelerating GNSS Software Receivers", in Proceedings of the 29th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2016), Portland, OR, Sept. 2016, pp. 44-61.	✓
Arbitrarily scalable receiver's software architecture.	-	Achieved. See: C. Fernández-Prades, J. Arribas and P. Closas, "Accelerating GNSS Software Receivers", in Proceedings of the 29th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2016), Portland, OR, Sept. 2016, pp. 44-61.	✓
Arbitrarily scalable configuration system.	-	See indicator "Flexible configuration mechanism."	✓
Logging system	Possibility to set up severity levels and verbose modes for messages.	AUDITOR uses Google Glog , Google's C++ logging system (see https://github.com/google/glog). It provides simple yet powerful APIs to various log events in the program.	✓

	<p>Possibility to set up conditional / occasional logging. Available under open source license and for all supported platforms</p>	<p>Messages can be logged by severity level, and the users can control logging behaviour from the command line, log based on conditionals, abort the program with stack trace when expected conditions are not met, and introduce their own verbose logging levels. Available under BSD 3-clause license.</p>	
Unit test software framework	<p>Test should be easy to write for programmers, letting test writers to focus on the test <i>content</i>. Test should be easy to read for programmers. Test should be order independent. Test should be deterministic. Test should be versionable. Test should be automatic. Tests should be <i>independent</i> and <i>repeatable</i>. Tests should be well <i>organized</i> and reflect the structure of the tested code. Tests should be <i>portable</i> and <i>reusable</i>. When tests fail, they should provide as much <i>information</i> about the problem as possible. Tests should be <i>fast</i> (less than 5-10 minutes) to execute. Testing framework available under open source license, and in all the supported environments.</p>	<p>AUDITOR uses Google Test, Google's C++ test framework (https://github.com/google/googletest), which meets all the required features. Available under BSD 3-clause license.</p>	✓
Public source code repository	<ul style="list-style-type: none"> ● Freely accessible. ● Robust hosting service. ● Management tools. ● Bug tracking system. 	<p>Available at https://github.com/gnss-sdr/gnss-sdr.git</p>	✓

	<ul style="list-style-type: none"> • Allowing automated building and other hooks on new code changes. 		
Communication channels	Public mailing list	Public mailing list at https://sourceforge.net/projects/gnss-sdr/lists/gnss-sdr-developers	✓
Documentation for users	Howtos, tutorials. Documentation is a never-ending process that continually gets feedback from users.	Available at https://gnss-sdr.org	✓
Documentation for developers	Identification of a source-code based automated documentation tool. Documentation is a never-ending process that continually gets feedback from users.	AUDITOR uses Doxygen as the automated source code documentation tool. Doxygen is free software, released under the terms of the GNU General Public License. In-code documentation, presented in an ordered manner and generated automatically, helps to keep an updated version of the source code manual.	✓

2. Validation of GNSS RF Front-End Module

In D3.1 the design and implementation of the FE architecture defined in D2.2 was introduced. The FE architecture was based on a dual channel configuration, with one fixed band (L1/E1) and one configurable band (L2 or E5a/L5).

The initial architecture implemented in the first prototype, FE v1.0, was upgraded to a second implementation, FE v2.0, which mitigated several issues related to EMI and in-band spurious.

In D6.2 different integration/validation tests were executed using FE v2.0, with a set of representative inputs signals (simple tones, modulated signals, GPS outdoor signals), in different environments (laboratory, outdoor). These tests identified several issues related to microcontroller noise interferences in the RF signal, clock stability, and minor in-band spurious. In order to mitigate these issues three different setups were defined:

- Setup #1: original FE architecture already presented in D3.1/D6.1.
- Setup #2: setup #1 downgraded from the external 8-bit DAC to the internal 2-bit per I/Q signal to reduce noise in the digitalization process.
- Setup #3: setup #2 with external clock reference to test different high stability clocks.

Several outdoor captures were carried out with Setup #3 which were successfully processed using the GNSS-SDR offline implementation as shown in D6.2.

An additional setup was implemented based on new transceiver which offered improved performance and higher flexibility for multi-band GNSS data acquisition.

3. Validation of the software-defined GNSS receiver

3.1 Signal Tracking

The performance of the signal tracking module is validated by using specific unit tests for the selected modules. The unit test framework uses an **open-source** software-defined GPS L1 CA signal generator that produces a complete constellation of GPS satellites plus a controlled Gaussian noise. The Carrier-to-Noise ratio (CNO) can be adjusted programmatically.

In addition, the signal generator produces also a metadata file containing the true synchronization parameters for the generated signal and its evolution along time. The unit test uses this file to compute KPIs for all the tracking products.

The signal tracking process implies two steps:

- Tracking pull-in: is the signal tracking initial transitory that absorbs the Doppler and Code Delay estimation errors from the signal acquisition module [ref Kaplan]. The tracking capture margin is defined as the Doppler error window expressed in Hertz and the Code Delay error window, expressed in Chips, that converges to a stable tracking and not produces a loss-of-lock. Usually depends on the PLL and DLL bandwidths respectively.
- Steady-state: the tracking errors have been stabilized to its nominal values and its performance for the selected PLL/DLL bandwidths depend only on the CNO and the satellite and receiver dynamics.

3.2 Tracking pull-in test

It is coded using the Google Test framework and included in the main *run_tests* executable with the test name *GpsL1CADIIPIITrackingPullInTest*. The GNSS-SDR module under test is the generic *dll_pll_veml_tracking* engine, configured to track GPS L1 CA signals. It can make use of the software-defined signal generator to produce GPS L1 CA signals at different CNO and obtain the true synchronization parameters. The test performs a two-dimensional sweep of Doppler errors and Code Delay errors for each CNO to emulate an imperfect signal acquisition in the pull-in tracking step. The test output is a 2D grid plot showing those combinations of Doppler and Code delay errors that produced a valid tracking (green dots) and those that produced a loss of lock (black dots).

3.2.1 Test results

This test accepts the following flags:

Table 3.1: Tracking pull-in test flags

Flag	Default value	Description
fs_gen_sps	2600000	Sampling rate, in Samples/s
enable_external_signal_file	false	Use an external signal file capture instead of the software-defined signal generator. NOTICE: when external file is selected, the test will try to perform a high sensitivity acquisition with an enhanced Doppler estimation to estimate the

		true signal synchronization parameters for all the satellites present in the signal
signal_file	signal_out.bin	Path of the external signal capture file, must be in int8_t format. If set, the signal generator will not be used and no CNO sweep will be done.
disable_generator	false	Disable the signal generator (the pre-generated signal file set must be available for the test, i.e. by running the test without disabling the generator previously).
duration	100	Duration of the experiment [in seconds, max = 300]. For this test the recommended signal duration is 4 seconds.
test_satellite_PRN	1	PRN of the satellite under test (must be visible during the observation time).
acq_Doppler_error_hz_start	1000	Acquisition Doppler error start sweep value [Hz]
acq_Doppler_error_hz_stop	-1000	Acquisition Doppler error stop sweep value [Hz]
acq_Doppler_error_hz_step	-50	Acquisition Doppler error sweep step value [Hz]
acq_Delay_error_chips_start	2.0	Acquisition Code Delay error start sweep value [Chips]
acq_Delay_error_chips_stop	-2.0	Acquisition Code Delay error stop sweep value [Chips]
acq_Delay_error_chips_step	-0.1	Acquisition Code Delay error sweep step value [Chips]
PLL_bw_hz_start	40.0	PLL Wide configuration value [Hz]
DLL_bw_hz_start	1.5	DLL Wide configuration value [Hz]
extend_correlation_symbols	1	Set the tracking coherent correlation to N symbols (up to 20 for GPS L1 C/A)
PLL_narrow_bw_hz	5.0	PLL Narrow configuration value [Hz]
DLL_narrow_bw_hz	0.75	DLL Narrow configuration value [Hz]
CNO_dBHz_start	(noise disabled)	Enable noise generator and set the CNO start sweep value [dB-Hz]
CNO_dBHz_stop	(noise disabled)	Enable noise generator and set the CNO stop sweep value [dB-Hz]
CNO_dB_step	3.0	Noise generator CNO sweep step value [dB]
plot_detail_level	0	Specify the desired plot detail (0,1,2): 0 - Minimum plots (default) 2 - Plot all tracking parameters.

show_plots	true	Shows plots on screen. Set it to false for non-interactive testing.
------------	------	---

Example of test execution command line:

```
$ ../install/run_tests --gtest_filter=GpsL1CADLLPLLTrackingPuLLInTest* --
plot_detail_level=0 --duration=5 --CN0_dBHz_start=45 --CN0_dBHz_stop=30 --
PLL_bw_hz_start=35 --DLL_bw_hz_start=1.5 --disable_generator=0 --
enable_external_signal_file=0 --acq_Doppler_error_hz_start=1000 --
acq_Doppler_error_hz_stop=-1000 --acq_Delay_error_chips_start=2 --show_plots=true --
acq_Delay_error_chips_stop=-2 --test_satellite_PRN=1
```

Test output:

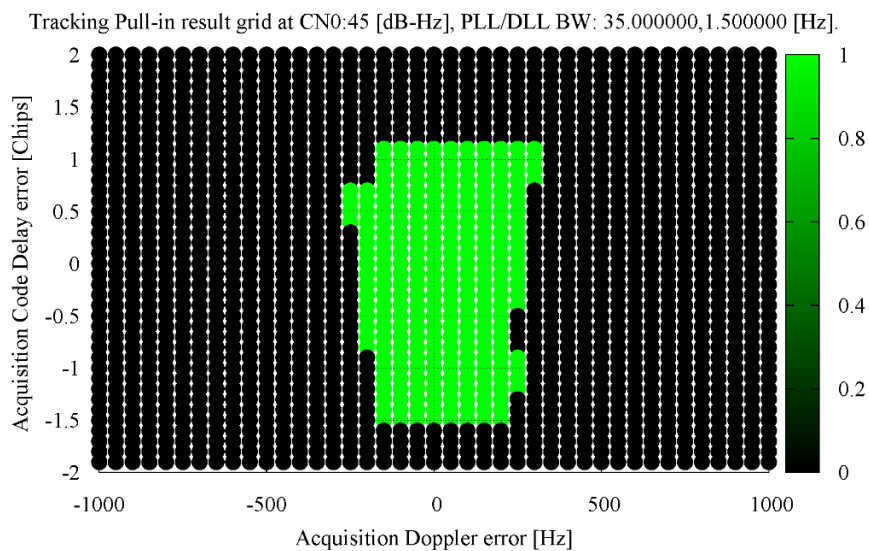


Figure 3.1: Tracking pull-in results for CN0= 45 dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.

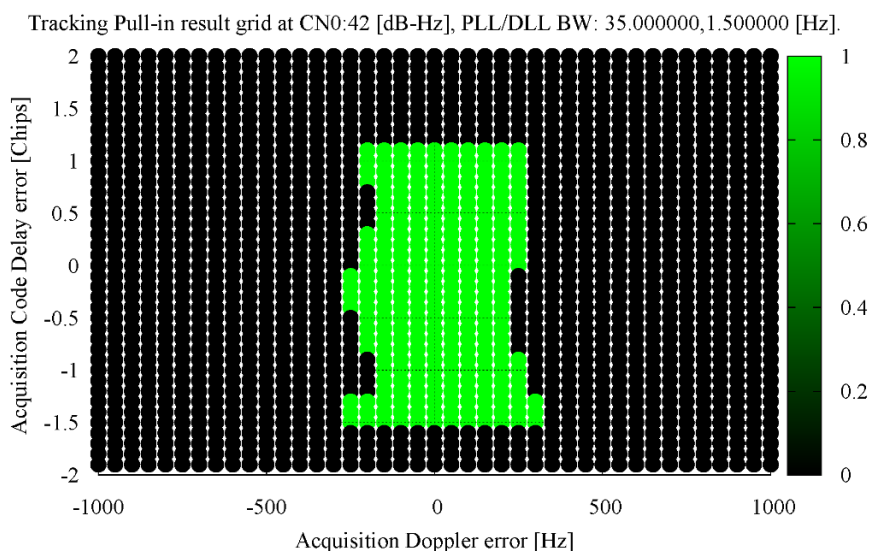


Figure 3.2: Tracking pull-in results for CN0= 42 dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.

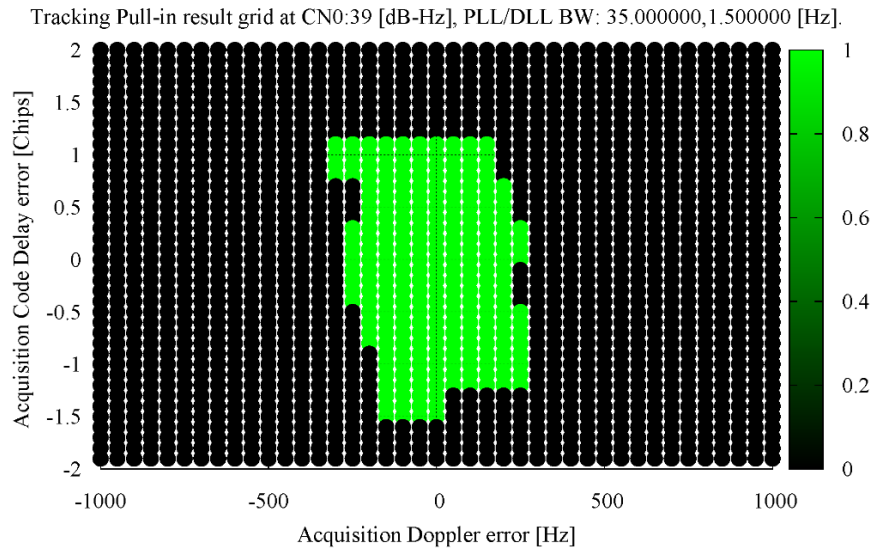


Figure 3.3: Tracking pull-in results for CN0= 39 dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.

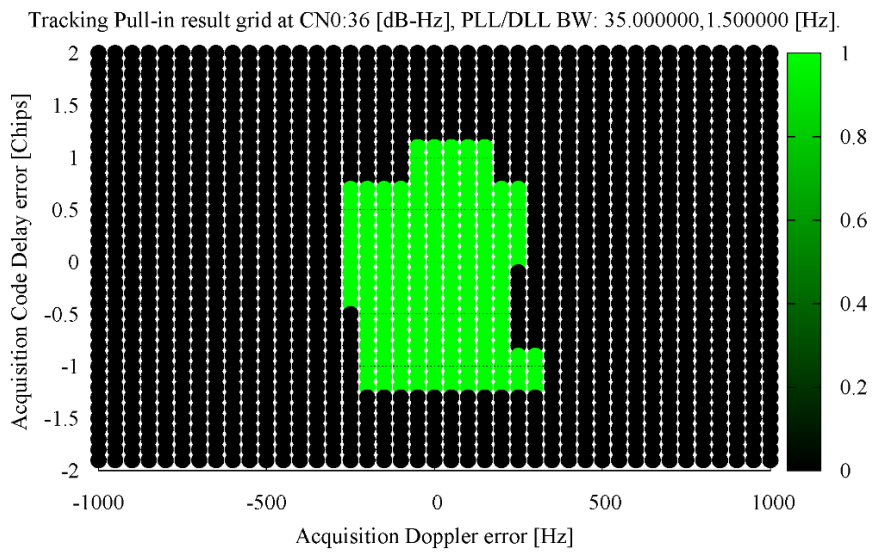


Figure 3.4: Tracking pull-in results for CN0= 36 dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.

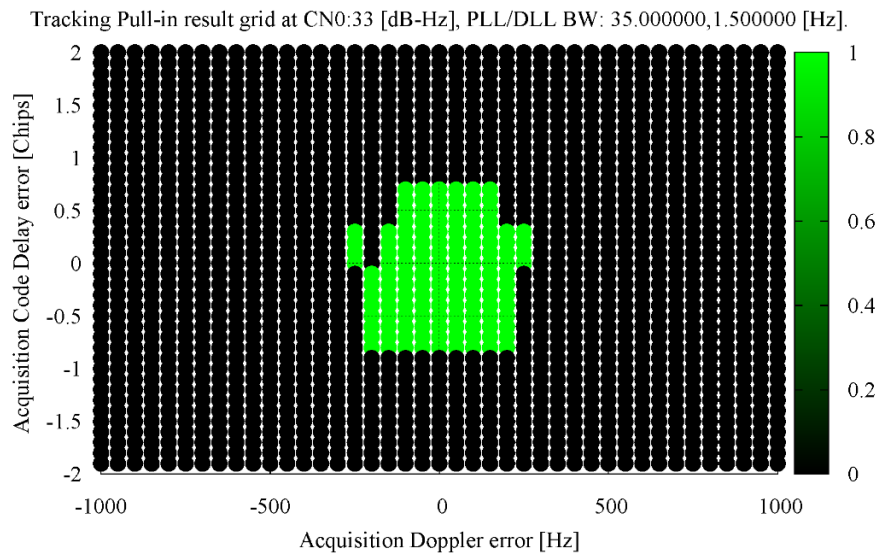


Figure 3.5: Tracking pull-in results for CN0= 33 dB-Hz using PLL/DLL bandwidths of 35 Hz and 1.5 Hz respectively.

The tracking pull in test results show that the GPS L1 CA tracking pull-in tolerates a Doppler error of [-250 to 250] Hz in a signal CN0 range of [45 to 33] dB-Hz, thus, the acquisition Doppler grid granularity of 500 Hz or narrower should work.

The Code Delay error tolerance on the other hand is reduced from [-1.5 to 1.0] Chips at 45 dB-Hz to [-0.85, 0.85] Chips in at 33 dB-Hz. The acquisition engine uses a parallel code search algorithm that provides an estimation of the signal Code Delay with suitable precision to enable the signal tracking at 33 dB-Hz, as shown in previous section.

3.3 Tracking steady-state test

The tracking steady-state test uses the signal generator metadata file to provide a “perfect” signal acquisition’s synchronization parameters to the tracking module. This procedure eliminates the pull-in transitory effect, thus it obtains the tracking errors and its performance in the estimation of the signal synchronization parameters for different tracking loops bandwidths and signal CN0.

The performance bounds of a PLL/DLL algorithm are defined hereafter.

3.3.1 CRB for the Code Delay

One satellite received complex baseband signal model

$$x(t)=as(t-\tau)\exp\{j2\pi f_d t\}+n(t) \quad (1)$$

with $s(t)$ the transmitted complex baseband low-rate navigation signal spread by the pseudorandom code, a its complex amplitude, τ the time-delay, f_d the Doppler shift and $n(t)$ is zero-mean additive Gaussian noise with variance σ^2 .

The CRB states that for any unbiased estimate of a generic real-valued parameter vector ξ , the covariance matrix of the estimates $\mathbf{C}(\xi)$ is bounded as

$$\mathbf{C}(\hat{\boldsymbol{\xi}}) \geq \mathbf{J}^{-1}(\boldsymbol{\xi}), \quad (2)$$

where $\mathbf{J}(\boldsymbol{\xi})$ is the Fisher information matrix (FIM). The elements of the fim are defined as

$$[\mathbf{J}(\boldsymbol{\xi})]_{u,v} = -\mathbb{E} \left\{ \frac{\partial^2 \Delta_x(\boldsymbol{\xi})}{\partial \xi_u \partial \xi_v} \right\}. \quad (3)$$

In our case, the parameter to be estimated is $\xi=\tau$, and the FIM is computed as

$$[\mathbf{J}(\tau)]_{\tau,\tau} = 2 \left\{ \frac{\partial d(t)^H}{\partial \tau} \frac{\|a\|^2}{\sigma_n^2} \frac{\partial d(t)}{\partial \tau} \right\}, \quad (4)$$

with $d(t)=s(t-\tau)\exp\{j2\pi f_d t\}$. The partial derivative of $d(t)$ is

$$\frac{\partial d(t)}{\partial \tau} = -\dot{s}(t - \tau) \exp\{j2\pi f_d t\}, \quad (5)$$

where $\dot{s}(t)$ is the derivative of time of the waveform $s(t)$. Then

$$[\mathbf{J}(\tau)]_{\tau,\tau} = 2C/N_0 T_d (2\pi)^2 \int f^2 G(f) df, \quad (6)$$

because $\frac{\|a\|^2}{\sigma_n^2} = \frac{C}{N_0} T_d$ with T_d the integration time, and the derivative of time results in a product on f in frequency. $G(f)$ is the power spectral density of the waveform, which depends on the GNSS signal. Finally, we have that

$$\text{CRB}_\tau = \frac{1}{8\pi^2 C/N_0 T_d \int f^2 G(f) df}. \quad (7)$$

We have that (T_c the chip period)

$$\begin{aligned} G(f)_{BPSK} &= T_c \text{sinc}^2(\pi f T_c) = T_c \left[\frac{\sin(\pi f T_c)}{\pi f T_c} \right]^2 \\ G(f)_{BOC(1,1)} &= \frac{1}{T_c} \left[\frac{\sin(\pi f T_c/2) \sin(\pi f T_c)}{\pi f \cos(\pi f T_c/2)} \right]^2 \end{aligned} \quad (8)$$

Taking into account that we use a coherent early-late time-delay detector (discriminator), the lower bound (LB) is given by

$$\sigma_{LB}^2 = \frac{B_L(1 - 0.5B_L T_d)}{4\pi^2 C/N_0 \int_{-\beta/2}^{\beta/2} f^2 G(f) df}, \quad (9)$$

with β the precorrelation bandwidth and B_L the DLL bandwidth.

3.3.2 Test results

This test accepts the following flags:

Table 3.2: Tracking steady-state test

Flag	Default value	Description
fs_gen_sps	2600000	Sampling rate, in Samples/s
enable_external_signal_file	false	Use an external signal file capture instead of the software-defined signal generator. NOTICE: when external file is selected, the test will try to perform a high sensitivity acquisition with an enhanced Doppler estimation to estimate the true signal synchronization parameters for all the satellites present in the signal
signal_file	signal_out.bin	Path of the external signal capture file, must be in int8_t format. If set, the signal generator will not be used and no CNO sweep will be done
disable_generator	false	Disable the signal generator (the pre-generated signal file set must be available for the test, i.e. by running the test without disabling the generator previously).
duration	100	Duration of the experiment [in seconds, max = 300]. For this test the recommended signal duration is 4 seconds.
test_satellite_PRN	1	PRN of the satellite under test (must be visible during the observation time).
PLL_bw_hz_start	40.0	PLL Wide configuration start sweep value [Hz]
PLL_bw_hz_stop	40.0	PLL Wide configuration stop sweep value [Hz]
PLL_bw_hz_step	5.0	PLL Wide configuration sweep step value [Hz]
DLL_bw_hz_start	1.5	DLL Wide configuration start sweep value [Hz]
DLL_bw_hz_stop	1.5	DLL Wide configuration stop sweep value [Hz]
DLL_bw_hz_step	0.25	DLL Wide configuration sweep step value [Hz]
extend_correlation_symbols	1	Set the tracking coherent correlation to N symbols (up to 20 for GPS L1 C/A)
PLL_narrow_bw_hz	5.0	PLL Narrow configuration value [Hz]
DLL_narrow_bw_hz	0.75	DLL Narrow configuration value [Hz]
CNO_dBHz_start	(noise disabled)	Enable noise generator and set the CNO start sweep value [dB-Hz]

CNO_dBHz_stop	(noise disabled)	Enable noise generator and set the CNO stop sweep value [dB-Hz]
CNO_dB_step	3.0	Noise generator CNO sweep step value [dB]
skip_trk_transitory_s	1.0	Skip the initial tracking output observables to avoid transitory results [s]. It is especially important when the extend_correlation_symbols is set to >1, in order to skip the symbol synchronization transitory.
plot_detail_level	0	Specify the desired plot detail (0,1,2): 0 - Minimum plots (default) 2 - Plot all tracking parameters.
show_plots	true	Shows plots on screen. Set it to false for non-interactive testing.

Example of test execution command line:

```
./run_tests --gtest_filter=GpsL1CADLLPLLTrackingTest* --duration=200 --CNO_dBHz_start=50 --CNO_dBHz_stop=25 --PLL_bw_hz_start=5 --PLL_bw_hz_stop=5 --DLL_bw_hz_start=0.25 --DLL_bw_hz_stop=0.25 --DLL_bw_hz_step=0.25 --disable_generator=1 --plot_detail_Level=0 --plot_gps_l1_tracking_test=1 --fs_gen_sps=4000000 --skip_trk_transitory_s=135 --PLL_narrow_bw_hz=1.0 --DLL_narrow_bw_hz=0.1 --extend_correlation_symbols=1
```

Test output:

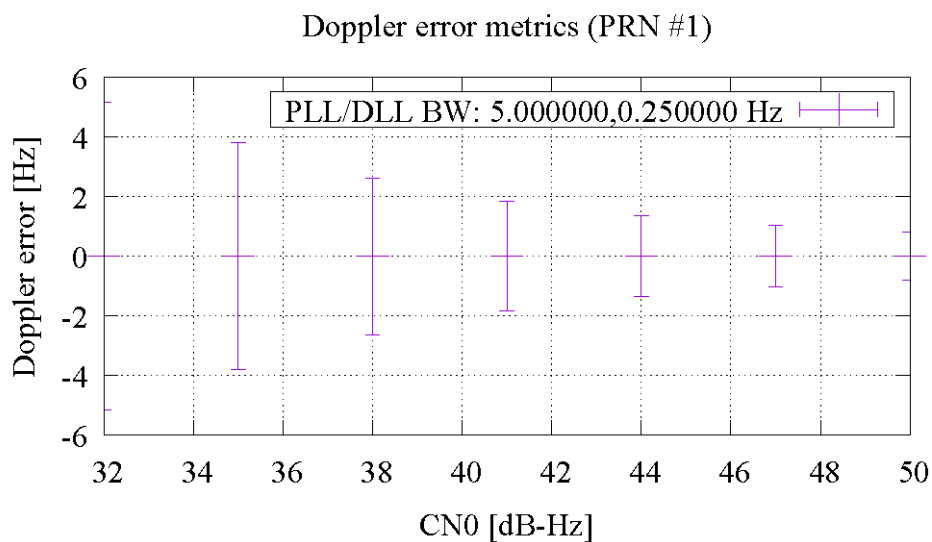


Figure 3.6: Tracking Doppler estimation error and variance for PLL/DLL bandwidths of 5 Hz and 0.25 Hz respectively.

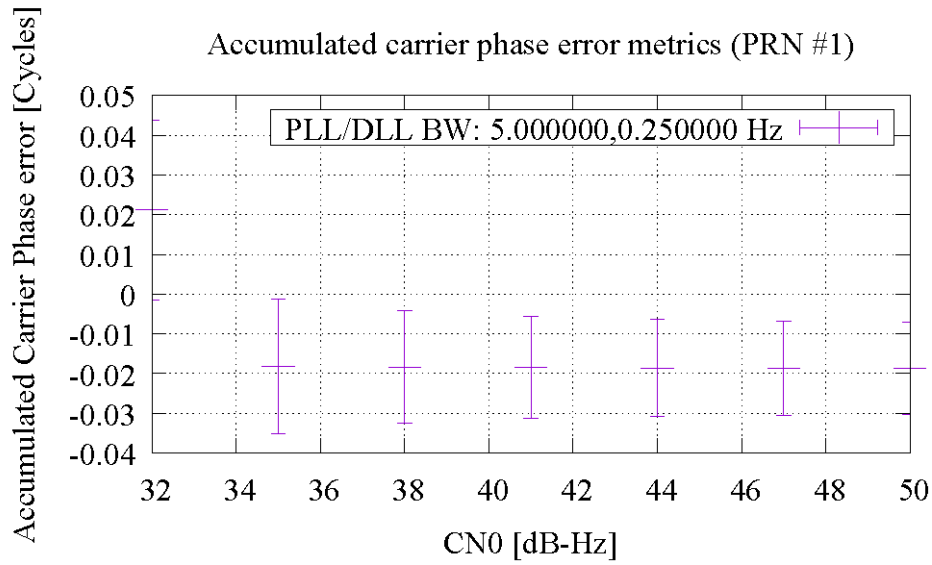


Figure 3.7: Accumulated carrier phase estimation error and variance for PLL/DLL bandwidths of 5 Hz and 0.25 Hz respectively.

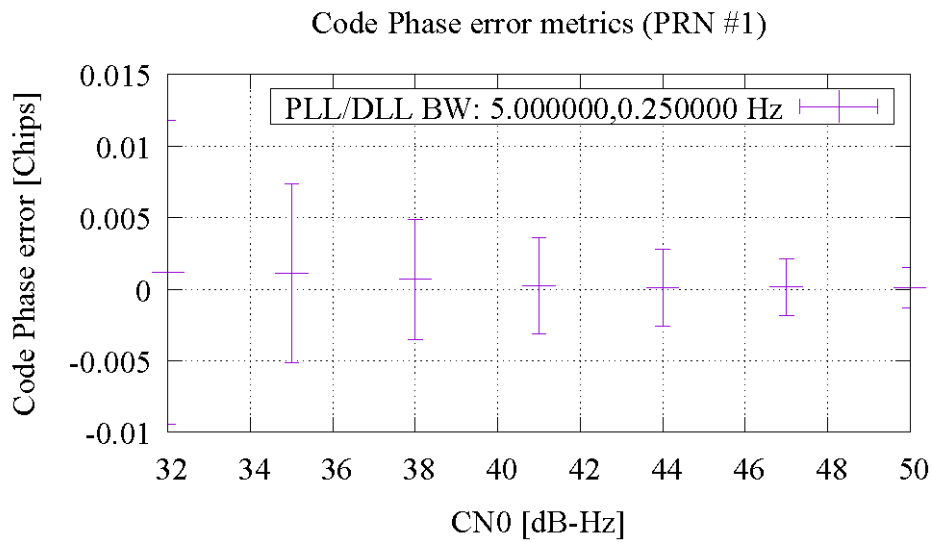


Figure 3.8: Code Phase estimation error and variance for PLL/DLL bandwidths of 5 Hz and 0.25 Hz respectively.

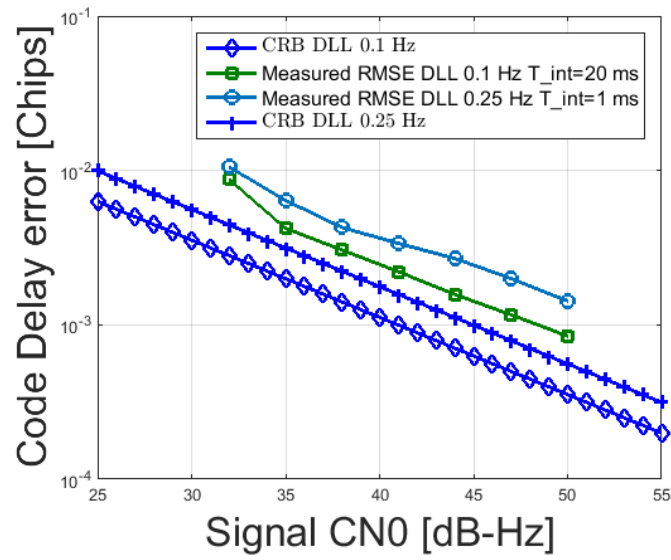


Figure 3.9: Measured RMSE and corresponding Cramér-Rao Bound for the Code Delay estimation.
The PLL filter was set to 1 Hz for $T_{int}=20$ ms and 10 Hz for $T_{int}=1$ ms.

4. Analysis of main Key Performance Indicators

This Section summarizes the main achievements of the AUDITOR Project in the KPIs identified in Deliverable 1.3.

Table 4.1: KPI 1 - Accuracy

KPI	KPI 1 - Position accuracy
Definition of KPI	Degree of correctness of the position fixes provided by the GNSS receiver.
Metrics	Position accuracy results are given in meters of error with respect to a reference (fiducial) point previously measured in a geodetic survey. Two of the most commonly used confidence measurements for positioning are the Distance Root Mean Square (DRMS) and the Circular Error Probability (CEP), for 2D positioning; and the Mean Radial Spherical Error (MRSE) and the Spherical Error Probable (SEP) when measures are expressed in 3D. Measurement procedures defined in https://gnss-sdr.org/design-forces/accuracy/
AUDITOR's achievements	<p>Test programs computing the metrics mentioned above have been implemented (see https://gnss-sdr.org/docs/tutorials/testing-software-receiver-2/)</p> <p>The accuracy of the standalone GNSS receiver has improved from a rough 100-m positioning at the starting of the Project to the order of 1 meter SEP in good satellite visibility conditions. The user can choose among Single-point and PPP-based PVT solutions.</p>

Table 4.2: KPI 2 - Availability

KPI	KPI 2 – Availability
Definition of KPI	The availability of a navigation system is the percentage of time that the services of the system are usable by the navigator.
Metrics	Several specific tests for measuring receiver's availability were proposed in https://gnss-sdr.org/design-forces/availability/
AUDITOR's achievements	The availability of the standalone GNSS receiver has improved from a six hours at the starting of the Project to a continued service (delivering PVT fixes continuously, measured for more than one week).

Table 4.3: KPI 3 - Efficiency

KPI	KPI 3 - Efficiency
------------	---------------------------

Definition of KPI	Number of concurrent channels (understood as the number of different GNSS satellites that can be tracked per GNSS signal, including GPS L1 C/A, GPS L2C, GPS L5, Galileo E1 OS, Galileo E5a, Galileo E5b, and EGNOS) can be sustained in real time for a given processing platform.
Metrics	In the software-defined GNSS receiver, the number of channels per signal is a configuration parameter. This number can be increased progressively until reaching a processing load that the executing platform is unable to deliver in real time. Measurement procedures defined in https://gnss-sdr.org/design-forces/efficiency/
AUDITOR's achievements	Relevant advances in execution time optimization (see reference above), both in personal computers and in FPGA-based devices. The AUDITOR receiver attains real time with 8 channels per GNSS signal (GPS L1, Galileo E1 OS, GPS L2C / GPS L5 + Galileo E5a).

Table 4.4: KPI 4 - Interoperability

KPI	KPI 4 - Interoperability
Definition of KPI	Number of GNSS signals that AUDITOR's software receiver is able to process, and the number of standard output formats for the products of GNSS signal processing.
Metrics	Specific interoperability issues are discussed in https://gnss-sdr.org/design-forces/interoperability/
AUDITOR's achievements	The software receiver is now able to generate GNSS products from GPS L1 C/A, Galileo E1 OS, GPS L2C, GPS L5 and Galileo E5a in standard formats: RTCM 3.2 and RINEX 3.01 for observables and navigation data, and KML, GPX and NMEA-0183 for GIS tools. Intermediate signals are available in binary .mat format, readable from Matlab, Octave and Python.

Table 4.5: KPI 5 - Repeatability

KPI	KPI 5 - Repeatability
Definition of KPI	Consistency of the position fixes delivered by the GNSS software receiver.
Metrics	Position precision results can be statistically characterized by averaging the results of repeated experiments performed on the same position or trajectory. Two of the most commonly used confidence measurements for positioning are the Distance Root Mean Square (DRMS) and the Circular Error Probability (CEP), for 2D positioning; and the Mean Radial Spherical Error (MRSE) and the Spherical Error Probable (SEP) when measures are expressed in 3D. Measurement procedures discussed in https://gnss-sdr.org/design-forces/repeatability/

AUDITOR's achievements	<p>Test programs computing the metrics mentioned above have been implemented (see https://gnss-sdr.org/docs/tutorials/testing-software-receiver-2/)</p> <p>The precision of the standalone GNSS receiver has improved from 2 m SEP positioning at the starting of the Project to the order of 20 cm SEP in good satellite visibility conditions.</p>
-------------------------------	---

5. Conclusion

The AUDITOR Project has achieved to deliver:

- 1) A proof-of-concept of the AUDITOR concept for high-precision accuracy for the agriculture.
- 2) A free and open source software-defined GNSS receiver available to the general public.

The main improvements in the software receiver developed during AUDITOR can be summarized as follows:

Improvements in Accuracy:

- Part of the RTKLIB core library has been integrated into GNSS-SDR. There is now a single PVT block implementation which makes use of RTKLIB to deliver PVT solutions, including Single and PPP navigation modes.
- Fixed CNO estimation for other correlation times than 1 ms.
- Improved computation of tracking parameters and GNSS observables.
- Major rewriting in the generation of pseudoranges.

Improvements in Availability:

- Internal Finite State Machines rewritten for improved continuity in delivering position fixes. This fixes a bug that was stalling the receiver after about six hours of continuous operation.
- Redesign of the time counter for enhanced continuity.
- Improved flow graph in multisystem configurations: the receiver does not get stalled anymore if no signal is found from the first system.
- Improved acquisition and tracking sensitivity.

Improvements in Efficiency:

- Added the possibility of non-blocking acquisition, which works well when using real-time data from an RF front-end.
- Improved flow graph in multiband configurations: satellites acquired in one band are immediately searched in others.
- Complex local codes have been replaced by real codes, alleviating the computational burden.
- Improvements in processing speed: The VOLK_GNSSDR library has been rewritten, following current VOLK standards and adding a number of new kernels. This approach addresses both efficiency and portability. Now the library provides the key kernels for GNSS signal processing in 16ic and 32fc versions, including SSE2, SSE3, SSE4.1, AVX, AV2 and NEON implementations.
- Improvements in C++ usage: Use of const container calls when result is immediately converted to a const iterator. Using these members removes an implicit conversion from iterator to const_iterator.

Improvements in Flexibility:

- A number of new parameters have been exposed to the configuration system.
- Possibility to choose Pilot or Data component for tracking of GPS L5 and Galileo E5a signals.
- Enabled extended coherent integration times.

- Some configuration parameters can now be overridden by commandline flags for easier use in scripts.

Improvements in Interoperability:

- Added the GPS L5 receiver chain.
- Added the GLONASS L1 SP receiver chain.
- Added the GLONASS L2 SP receiver chain.
- Improvements in the Galileo E5a and GPS L2C receiver chains.
- Updated list of available GNSS satellites.
- Added five more signal sources: "Fmcomms2_Signal_Source" (requires gr-iio), "Plutosdr_Signal_Source" (requires gr-iio), "Spir_GSS6450_File_Signal_Source", "Labsat_Signal_Source" and "Custom_UDP_Signal_Source" (requires libpcap). Documented in <https://gnss-sdr.org/docs/sp-blocks/signal-source/>
- Improved support for BladeRF, HackRF and RTL-SDR front-ends.
- Added tools for the interaction with front-ends based on the AD9361 chipset.
- Intermediate results are now saved in .mat binary format, readable from Matlab/Octave and from Python via h5py.
- Added a RTCM printer and TCP server in PVT blocks. The receiver is now able to stream data in real time, serving RTCM 3.2 messages to multiple clients. For instance, it can act as a Ntrip Source feeding a Ntrip Server, or to be used as data input in RTKLIB, obtaining Precise Point Positioning fixes in real-time. The TCP port, Station ID, and rate of MT1019/MT1045 and MSM can be configured. GPS_L1_CA_PVT serves MT1019 (GPS Ephemeris) and MSM7 (MT1077, full GPS pseudoranges, phase ranges, phase range rates and CNR - high resolution) messages, while GALILEO_E1_PVT serves MT1045 (Galileo ephemeris) and MSM7 (MT1097, full Galileo pseudoranges, phase ranges, phase range rates and CNR - high resolution).
- Added a GeoJSON printer. Basic (least-squares) position fixes can be now also stored in this format, in addition to KML.
- Added the GPX output format.
- Fixed a bug in the format of NMEA sentences when latitude or longitude minutes were >10.
- Improvements in the correctness of generated RINEX files.

Improvements in Maintainability:

- Setup of a Continuous Integration system that checks building and runs QA code in a wide range of GNU/Linux distributions (ArchLinux, CentOS, Debian, Fedora, OpenSUSE, Ubuntu) and releases. See <https://gitlab.com/gnss-sdr/gnss-sdr>
- Creation of multi-system processing blocks, drastically reducing code duplication and maintainability time.
- Automated code formatting with clang-format. This tool is widely available and easy to integrate into many code editors, and it also can be used from the command line. It cuts time spent on adhering to the project's code formatting style.
- Improvement in C++ usage: C-style casts have been replaced by C++ casts. C-style casts are difficult to search for. C++ casts provide compile time checking ability and express programmers' intent better, so they are safer and clearer.
- Improvement in C++ usage: The override special identifier is now used when overriding a virtual function. This helps the compiler to check for type changes in the base class, making the detection of errors easier.
- Improvement in C++ usage: A number of unused includes have been removed. Order of includes set to: local (in-source) headers, then library headers, then system headers. This helps to detect missing includes.

- Improvement in C++ usage: Enhanced const correctness. Misuses of those variables are detected by the compiler.

Improvements in Marketability:

- Reduced time from a commit to deployment (see virtualization mechanisms in Portability).

Improvements in Portability:

- Now GNSS-SDR can be run in virtual environments through snap packages (see <https://github.com/carlesfernandez/snapcraft-sandbox>) and docker images (see <https://github.com/carlesfernandez/docker-gnssdr>).
- Now GNSS-SDR is adapted to cross-compiling environments for embedded devices (see <https://gnss-sdr.org/docs/tutorials/configuration-options-building-time/> and <https://github.com/carlesfernandez/oe-gnss-sdr-manifest>).
- Several CMake scripts improvements, more verbose outputs in case of errors. Building configuration has been documented in <https://gnss-sdr.org/docs/tutorials/configuration-options-building-time/>
- Improved SDK for cross-compilation in embedded devices. Documented in <https://gnss-sdr.org/docs/tutorials/cross-compiling/>
- Improved control over minimum required versions for core dependencies.
- The software builds with C++11, C++14 and C++17 standards.
- The software can now be built using GCC >= 4.7.2 or LLVM/Clang >= 3.4.0 compilers on GNU/Linux, and with Clang/AppleClang on MacOS.
- The Ninja build system can be used in replacement of make.
- The volk_gnssdr library can be built using Python 2.7 or Python 3.6.
- The volk_gnssdr library is now ready for AArch64 NEON instructions.
- Ready for GNU Radio 3.8 C++ API (as per current next branch of GNU Radio upstream repository).
- Improved detection of required and optional dependencies in many GNU/Linux distributions and processor architectures.
- Improvement in C++ usage: The <ctime> library has been replaced by the more modern and portable <chrono>.
- Improvement in C++ usage: The <stdio.h> library has been replaced by the more modern and portable <fstream> for file handling.
- Improvement in C++ usage: C++ libraries preferred over C libraries (e.g., <cctype> instead of <ctype.h>, <cmath> instead of <math.h>).
- Fixes required by Debian packaging.
- Fixes required by Macports packaging.

Improvements in Reliability:

- Introduced 3 new Input Filter implementations for pulsed and narrowband interference mitigation: `Pulse_Blanking_Filter`, `Notch_Filter` and `Notch_Filter_Lite`. Documented in <https://gnss-sdr.org/docs/sp-blocks/input-filter/>
- Improved flow graph stability.
- Introduction of high-integrity C++ practices into the source code and included in the coding style guide. See <https://gnss-sdr.org/coding-style/>
- Fixed a number of defects detected by Coverity Scan.
- Improvement in C++ usage: rand() function replaced by <random> library.

- Improvement in C++ usage: strlen and strncpy have been replaced by safer C++ counterparts.
- Improvement in C++ usage: Some destructors have been fixed, avoiding segmentation faults when exiting the program.
- Website switched from http to https. Links in the source tree switched when available.

Improvements in Reproducibility:

- Setup of a Continuous Reproducibility system at GitLab for the automatic reproduction of experiments. The concept was introduced in <https://ieeexplore.ieee.org/document/8331069/> Example added in the src/utis/reproducibility/ieee-access18/ folder.
- Fixes of Lintian warnings related to build reproducibility.

Improvements in Scalability:

- Improvements in receiver design: Internal block communication has been redesigned to accommodate the addition of new signals, and now upstream and downstream communication within blocks is implemented through the GNU Radio block's asynchronous message passing system, leading to a more scalable, more robust and cleaner design.
- Improvements in multi-system, multi-band receiver configurations. The receiver now accepts any number of channels and systems in the three available bands.
- All possible combinations of signals and integration times are now accepted by the Observables block.

Improvements in Testability:

- Major QA source code refactoring: they have been split into src/tests/unit-tests and src/tests/system-tests folders. They are optionally built with the ENABLE_UNIT_TESTING=ON (unit testing QA code), ENABLE_UNIT_TESTING_EXTRA=ON (unit tests that require extra files downloaded at configure time), ENABLE_SYSTEM_TESTING=ON (system tests, such as measurement of Time-To-First-Fix) and ENABLE_SYSTEM_TESTING_EXTRA=ON (extra system test requiring external tools, automatically downloaded and built at building time) configuration flags. The EXTRA options also download and build a custom software-defined signal generator and version 2.9 of GPSTk, if not already found on the system. Download and local link of version 2.9 can be forced by ENABLE_OWN_GPSTK=ON building configuration flag. Only ENABLE_UNIT_TESTING is set to ON by default.
- Several Unit Tests added. Documentation of testing concepts and available tests at <https://gnss-sdr.org/docs/tutorials/testing-software-receiver/>
- Receiver channels can now be fixed to a given satellite.
- Improved CTest support in volk_gnssdr.

Improvements in Usability:

- All Observables block implementations have been merged into a single implementation for all kinds of GNSS signals, making it easier to configure.
- All PVT block implementations have been merged into a single implementation for all kinds of GNSS signals, making it easier to configure.
- Misleading parameter name GNSS-SDR.internal_fs_hz has been replaced by GNSS-SDR.internal_fs_sps. The old parameter name is still read. If found, a warning is provided to the user.

- Updated and improved documentation of processing blocks at <https://gnss-sdr.org/docs/sp-blocks/>
- Improved documentation of required dependency packages in several GNU/Linux distributions.
- Parameter names with the same role have been harmonized within different block implementations.
- Added a changelog, a code of conduct, a contributing guide and a pull-request template in the source tree.
- Added colors to the commandline user interface.
- Updated manfiles.

See the definitions of concepts and metrics at <https://gnss-sdr.org/design-forces/>

The source code is publicly available at <https://github.com/gnss-sdr/gnss-sdr>