Implementation of GNSS Receiver Hardware Accelerators in All-Programmable System-On-Chip Platforms

Marc Majoral, *Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA)*, Spain.

Carles Fernández-Prades, *Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA)*, Spain.

Javier Arribas, *Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA)*, Spain.

BIOGRAPHY

Marc Majoral is a Researcher at the Advanced Signal and Information Processing Department (ASIP), Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), Barcelona, Spain. He received the M.S. degree in Electrical Engineering from the Polytechnic 3University of Catalonia (UPC) in 1998. In 1999 he worked in the Signal Theory and Communications department in the Polytechnic University of Catalonia, where he was involved in DSP programming for mobile communications. From 1999 to 2006 he worked in the Cochlear Technology Centre in Belgium, where he implemented signal processing strategies for cochlear implants on DSPs. Since 2007 he is working at CTTC, implementing signal processing strategies on DSPs and FPGAs for telecommunication applications.

Dr. Carles Fernández–Prades holds a position of Senior Researcher and currently serves as Head of the Statistical Inference for Communications and Positioning (SICP) Department at CTTC. He received a PhD degree in Electrical Engineering from Universitat Politècnica de Catalunya (UPC) in 2006, Barcelona, Spain. His primary areas of interest include statistical and multi-sensor signal processing, estimation and detection theory, and Bayesian filtering, with applications to communication systems, GNSS and software-defined radio technology.

Dr. Javier Arribas holds a position of Senior Researcher at the SICP Department at CTTC. He received the MSc degree in Telecommunication Engineering from La Salle University in 2004, and the PhD from UPC in 2012. His primary areas of interest include statistical signal processing, GNSS synchronization, detection and estimation theory, software defined receivers, FPGA prototyping and the design of RF front-ends. He is the recipient of the 2015 EURASIP Best PhD Thesis Award.

ABSTRACT

This paper reports the design and proof-of-concept implementation of hardware accelerator modules for a low power consumption and small form factor software-defined GNSS receiver using an all-programmable System-On-Chip (SoC) platform. An all-programmable SoC is a device that integrates the software reprogrammability of a CPU with the hardware reprogrammability of an FPGA. The presented approach takes advantage of the flexibility of software-defined radio technology, and the power efficiency and small form factor of a SoC, to implement a portable fully customizable GNSS receiver with the capability to process GNSS signals in real-time and to deliver GNSS products in standard formats.

The SoC runs 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. However, the most computationally demanding tasks are offloaded to the FPGA and implemented as hardware acceleration modules. The hardware acceleration modules can take advantage of the inherent parallelism in the GNSS receiver signal processing functions.

A review of the GNSS receiver architecture is presented, together with an overview of the software and a design description of the hardware accelerators in the SoC. A dual-band proof of concept GNSS receiver is exposed, together with some results.

1. INTRODUCTION

The advent of a number of new GNSS (Galileo, BeiDou) and the modernization of existing ones (GPS and GLONASS), and the deployment of augmentation systems depict an unprecedented landscape for receiver designers. Unfortunately there is a lack of devices that are both cheap and flexible enough to be widely used for scientific research on new GNSS algorithms and processing methods.

In order to address the lack of flexibility of existing GNSS receivers, we created GNSS-SDR [1], an open source, software-defined GNSS receiver. The availability of hardware and software whose design and source code are published under open licenses opens new opportunities and challenges for science [2]. GNSS-SDR takes care of all the digital signal processing, performing digital acquisition and tracking of the available satellite signals, decoding the navigation message and computing the observables needed by positioning algorithms, which ultimately compute the navigation solution. The software is designed to facilitate the inclusion of new signal processing techniques, offering an easy way to measure their impact in the overall receiver performance. GNSS-SDR can be used on most types of general purpose computers. If the computer executing the software receiver is powerful enough, GNSS-SDR can be executed in real-time. When GNSS-SDR is executed in real-time, it can be used together with several popular RF front-ends such as USRP boards [3] using live GNSS signals received from satellites. Therefore, testing of all the processes is conducted either using real-time signals received by a radio-frequency front-end or a file containing those raw signal samples.

The GNSS-SDR platform implements a very low cost, highly-flexible GNSS receiver equipment. However, this equipment is not portable because it requires a general purpose computer to run the signal processing algorithms required by the GNSS signals. In this work, the authors targeted the development of a portable GNSS-SDR-based platform for use in many applications, for instance high-precision agriculture [4, 5, 6, 7] and spaceborne GNSS receivers [8, 9]. A portable GNSS-SDR-based platform embodies a low cost real-time flexible GNSS receiver that can be used in the field for researching and testing novel GNSS algorithms and signal processing methods.

An adequate solution to make a small form factor GNSS-SDR-based platform portable and efficient in terms of energy consumption is to use an all-programmable System-On-Chip (SoC). The SoC runs the most computationally intensive operations in an FPGA [10]. The SoC also runs the remaining signal processing functions in a general purpose CPU. In this project, we used the Xilinx Zynq 7000 SoC [11], which combines a dual-core ARM Cortex-A9 processor and an FPGA in a single chip. The ARM processor runs the GNSS-SDR on a GNU/Linux operating system, in the same way the GNSS-SDR runs on a desktop computer, but the most computationally demanding tasks are offloaded to the FPGA and implemented as hardware acceleration modules. The hardware acceleration modules can take advantage of the inherent parallelism in the GNSS-SDR receiver signal processing functions [12]. This paper presents the implementation of hardware accelerator modules for the GNSS-SDR software platform with the objective of developing a low-cost, small form factor, low power consumption and real-time flexible GNSS receiver. A GNSS receiver with these characteristics fills the gap between the market segments of professional GNSS receivers and mass-market GNSS receivers existing today.

The hardware accelerator modules are implemented in the form of FPGA IP cores [13]. The importance of FPGA IPs has grown over the years. They ease the reusability and the integration of the VHDL modules and they are a common way of distributing VHDL modules. The hardware accelerators are implemented as IPs that use the IP-XACT specification [14], which is widely used for packaging, integrating and reusing IP within design tool flows. The use of hardware accelerators are a way of reducing the overall complexity and the power consumption of the GNSS-SDR platform while keeping the real-time functionality. By using the numerous FPGA resources simultaneously, we can deliver real-time performance while keeping the clock frequency of the digital part low, thus reducing power consumption when compared to a laptop computer.

The remainder of this paper is organized as follows: Section 2 summarizes the main objectives of this work. Section 3 describes the system architecture and provides a description of the software and hardware components of the GNSS receiver with a focus on the GNSS hardware accelerators. Section 4 describes the developed prototype, Section 5 presents some results, and finally Section 6 wraps up some conclusions.

2. OBJECTIVES

The main motivation for this work is to implement a small form factor portable GNSS receiver that fulfills the list of features shown below:

- Possibility to either use synthetically generated or real-life GNSS signals.
- Possibility to process signals either in real time or in post-processing time.
- Possibility to use interchangeable RF front-ends.
- Possibility to define custom receiver architectures.
- Possibility to easily define / interchange implementations and parameters for each processing block.
- Availability of operation modes, as combinations of single/multiple frequency bands, single/multiple constellations.
- Testeability: the ability to observe the states, outputs, resource usage and other side effects of the software under test and the ability to apply inputs to the software under test or place it in specified states.
- Maintainability: The GNSS HW Accelerators shall be easy to maintain in order to isolate and correct bugs or their cause.
- Reliability: ability of a system or component to function under stated conditions for a specified period of time. Reliability refers to the consistency of the results provided by a system; internal and external reliability are, respectively, the ability to detect gross errors and the effect of an undetected blunder on the solution.
- Small form factor.
- Low power consumption.

Table 3 in section 5 shows how these requirements are met by the GNSS receiver implemented on a SoC platform.

3. SYSTEM DESIGN

This section explains how the GNSS receiver is designed using an all-programmable System-On-Chip (SoC).

3.1. System Architecture

A simplified block diagram of a Programmable System on Chip (SoC) is shown in Figure 1. The SoC consists of two main parts: a Processing System (PS) and a Processing Logic (PL). The Processing system contains a CPU and several peripherals (watchdogs, interrupt controllers, DMA, UART, and many others). The Processing Logic contains FPGA logic, which can be programmed to implement hardware accelerators for the CPU. The SoC also contains a number of I/O pins which can be configured to implement various interfaces (access to external DDR memory, GPIO, and many others).

The implementation of the GNSS receiver consists of the GNSS-SDR software running in the PS, where the most computationally demanding signal processing blocks of the GNSS-SDR software are replaced by hardware accelerators in the PL.

A system block diagram of the GNSS receiver is shown in Figure 2. The Radio Frequency Front-End captures and digitizes the GNSS signal. The RF Front-end is external to the SoC. The SAMPLE_SW, which is implemented in the PL, acts as an interface between the RF Front-End and the GNSS HW accelerators running in the PL. Using another RF Front-End only requires changes in the SAMPLE_SW. The FPGA_SW contains the GNSS hardware accelerator modules implemented in the PL. The MID_SW is the interface between the hardware accelerator modules and the microprocessor in the PS. The uP_SW Microprocessor in the PS running the GNSS-SDR SW. Finally, the results are shown to the user using a Human Machine Interface (HMI).

3.2. Software Design

The GNSS-SDR software is a software-defined GNSS Receiver [1], which is implemented using GNU Radio libraries [15]. The GNSS-SDR contains various instances of a channel function, where each instance is assigned to a satellite that is detected by in the signal coming from the RF Front-End. Each channel contains three software modules: Acquisition, Tracking and Telemetry. The Acquisition module is responsible for detecting the GNSS satellites. When a satellite is detected by an acquisition function then the corresponding channel is assigned to it. The results of all the channel processing modules are forwarded to the Observables module, and finally a Position, Velocity and Time (PVT) result is computed and delivered to the user.

There is an underlying scheduler that moves data from sources to sinks.

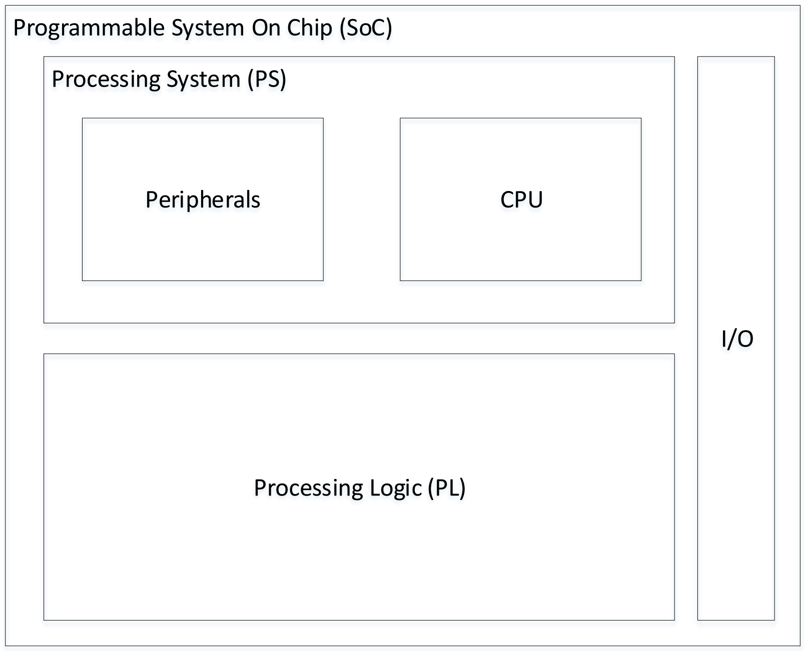

Fig. 1. All-Programmable System-On-Chip.

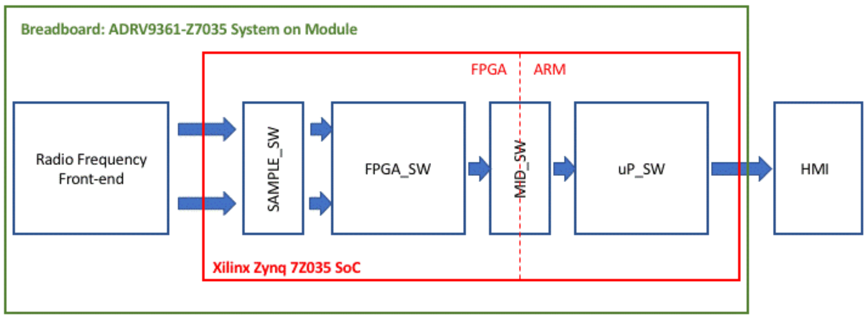
Breadboard: ADRV9361-Z7035 System on Module
FPGA
ARM
Radio Frequency Front-end
SAMPLE_SW
FPGA_SW
MID_SW
uP_SW
HMI
Xilinx Zynq 7Z035 SoC

Fig. 2. Architecture of the GNSS Receiver.

In order for the GNSS-SDR software to drive the HW Accelerators, the Signal Source and Signal Conditioning blocks of the GNSS-SDR software are eliminated, transferring their functionalities to the PL. The acquisition and the tracking blocks are replaced by HW accelerators in the PL. The GNSS-SDR acquisition blocks become non-GNU Radio Blocks, but controllers on the Acquisition HW accelerator Modules inside the PL. The HW Accelerator Tracking blocks become signal sources from the point of view of the SW. These signal sources deliver a stream of data extracted from the signal acquisition and tracking process.

The HW Accelerator blocks are 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. This is shown in Figure 4.

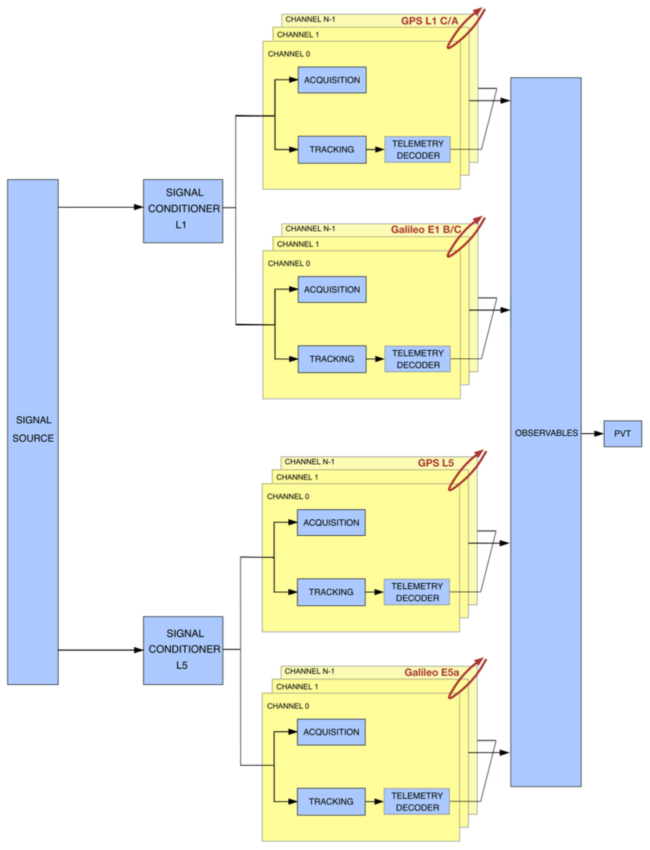

Fig. 3. Architecture of the GNSS-SDR software.

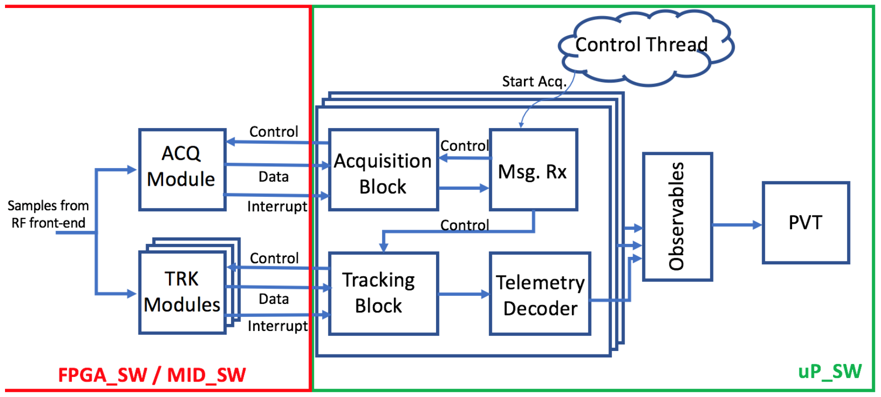

Fig. 4. Interface between the HW Accelerators and the GNSS-SDR Software.

There are two types of HW accelerators in our implementation:

- The acquisition HW accelerator: the acquisition HW accelerator performs a parallel code phase search acquisition [16]
- The multicorrelator HW accelerator (also referred to as tracking HW accelerators): the tracking HW accelerators run a Doppler wipe-off and a multi-correlator between the received GNSS signal and a local copy of the GNSS codes [16]. They demodulate the GNSS signal and transfer the demodulated symbols to the GNSS-SDR software running in the PS in order to decode the navigation messages.

The acquisition HW accelerator and the tracking HW accelerators replace the acquisition functions and most of the functionality of the tracking functions in the GNSS-SDR SW shown in Figure 4.

3.3. Hardware Design

Figure 5 shows the architecture of a dual band GNSS receiver in the PL. In our work one frequency band is used to capture and track the GPS L1 C/A and the Galileo E1 signals. The other frequency band is used to capture the GPS L5 and Galileo E5 signals (see section 4). The Analog Front-End has two A/D converters, one for each frequency band. The signal that is captured by the A/D converters is temporarily stored in a buffer. There is one buffer for each frequency band. The buffers deliver the captured signals to the various HW accelerators.

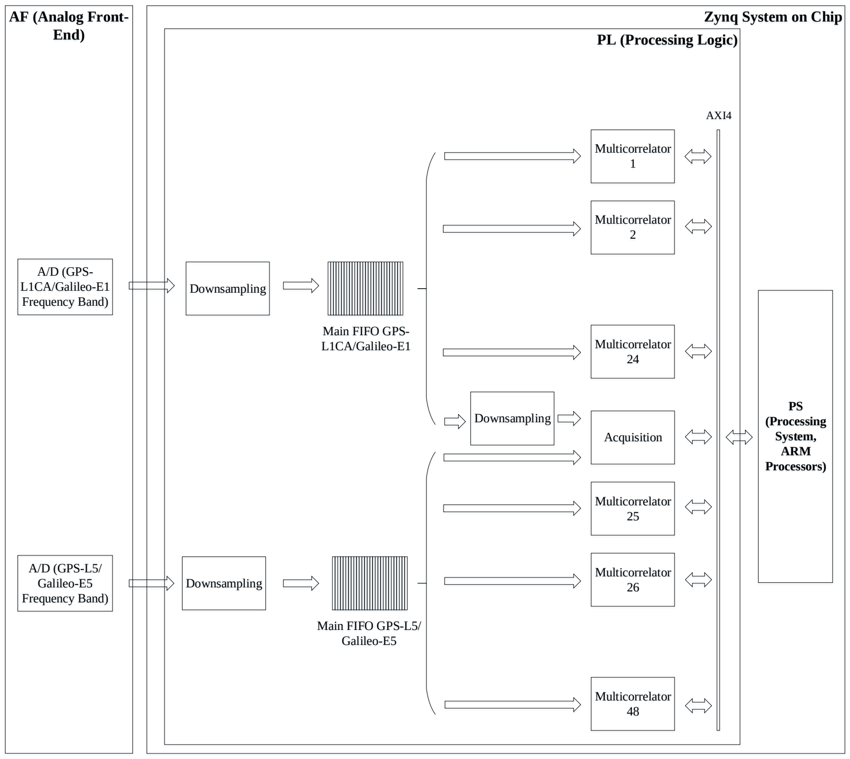

Fig. 5. Block diagram of the HW accelerator blocks in the System-On-Chip's Programmable Logic.

The receiver contains one acquisition HW accelerator and a large number of tracking HW accelerators. The acquisition HW accelerator can be used for all the types of GPS and Galileo signals under consideration (GPS L1 C/A, Galileo E1, GPS L5 and Galileo E5). The GNSS receiver searches for the various GNSS satellites sequentially, using the same acquisition HW acceleration module.

The receiver uses one tracking HW accelerator module for each detected GNSS satellite. In this work, 12 tracking HW accelerator modules for GPS L1 C/A, 12 for Galileo E1, 12 for GPS L5 and 12 for Galileo E5 (see section 4) were implemented.

The HW accelerator modules were connected to the PS using a high speed and low latency communication bus. In this way the microprocessor can control the hardware accelerators in real time.

3.4. Acquisition HW Accelerator

A block diagram of the Acquisition HW Accelerator is shown in figure 6. The Acquisition HW Accelerator performs a Doppler wipe–off, and a correlation between the PRN of a GNSS satellite and the received GNSS signal. This correlation is done in the frequency domain. In order to save space in the PS, a reversible FFT module is used. The reversible FFT module performs both the FFT and the IFFT algorithm. The signal in the frequency domain is routed back to the input of the reversible FFT module using switches.

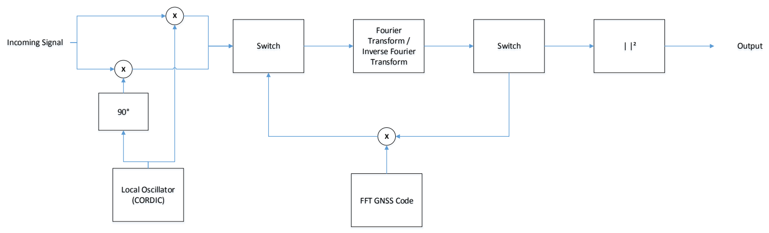
Incoming Signal
x
x
90°
Local Oscillator (CORDIC)
Switch
Fourier Transform / Inverse Fourier Transform
Switch
| |²
Output
x
FFT GNSS Code

Fig. 6. Block diagram of the Acquisition HW accelerator.

To run a correlation in the frequency domain, the Acquisition HW accelerator computes the Fourier transform of the received signal and the Fourier transform of the GNSS code. The GNSS code is previously resampled to match the sampling frequency of the GNSS signal. The size of the Fourier transform depends on the sampling frequency and the time duration of the PRN code. In order for the FFT size to be a power of two, the vector of input samples and the PRN code are zero-padded before running the FFTs. Zero padding consists in adding zeros to the end of a time-domain signal to increase its length. The Fourier transform is computed as shown in the equations below, where $x(n)$ is the local PRN code and $y(n)$ is the received GNSS signal after performing zero padding [16]:

$$X(K) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \tag{1}$$

$$Y(K) = \sum_{n=0}^{N-1} y(n) e^{-j2\pi kn/N} \tag{2}$$

As a second step, the correlation in the frequency domain is computed by multiplying the two FFTs:

$$Z(k) = X(k)^* Y(k) \tag{3}$$

Finally, the circular cross-correlation sequence between the zero-padded received signal and the zero-padded GNSS code is obtained by computing the inverse Fourier transform [16]:

$$z(n) = \frac{1}{N} \sum_{n=0}^{N-1} x^*(m) y(m+n) \tag{4}$$

The Acquisition HW Accelerator also performs a Doppler sweep without any SW intervention. The result of the acquisition algorithm is an assessment whether the satellite signal is present in the received signal. This result is based on the cross-correlation sequences obtained for each Doppler value.

3.5. Tracking HW Accelerator

A block diagram of the Tracking HW Accelerator is shown in figure 7. The tracking HW Accelerator is a multicorrelator that consists of a Dopler wipe–off unit and a number of correlators working in parallel. Each correlator performs a correlation between the received GNSS signal and the PRN code of the GNSS satellite that is being tracked. This correlation is done in the time domain [16]. Each correlator also contains a resampler that changes the sampling frequency of the local copy of the PRN code on the fly, to match the sampling frequency of the received GNSS signal. In order to save space in the FPGA, the GNSS receiver uses a different type of Tracking HW Accelerator for each GNSS signal under consideration (GPS L1 C/A, Galileo E1, GPS L5 and Galileo E5). The only difference between these tracking HW accelerators is the number of correlators that are used:

- The GPS L1 C/A multicorrelator consists of three correlators: Early, Prompt and Late (E, P and L).
- The Galileo E1 multicorrelator consists of six correlators: Five of them are used to obtain the Very Early, Early, Prompt and Late results (VE, E, P, L and VL). The sixth correlator is used when the pilot signal and the data signal are tracked simultaneously.

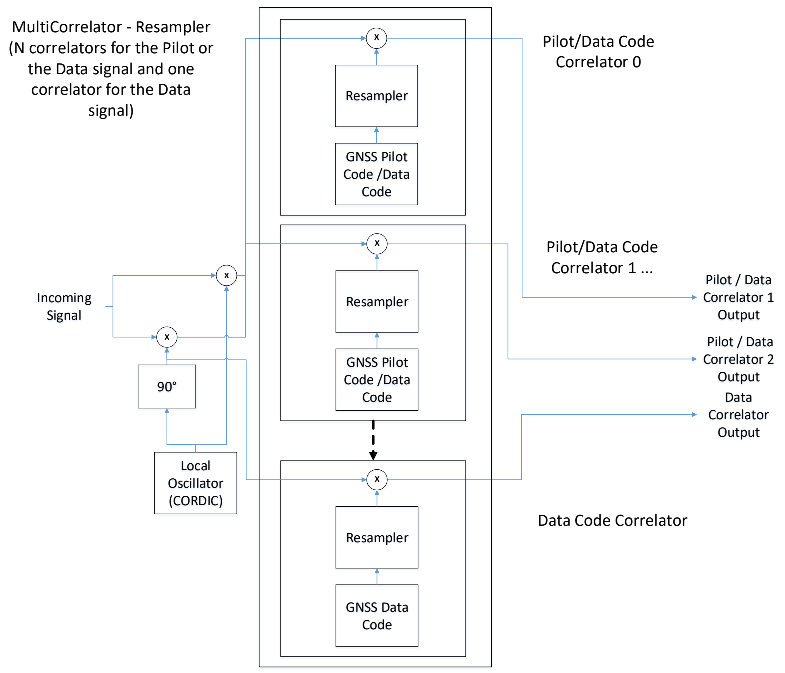
MultiCorrelator - Resampler (N correlators for the Pilot or the Data signal and one correlator for the Data signal)
Pilot/Data Code Correlator 0
Resampler
GNSS Pilot Code /Data Code
Pilot/Data Code Correlator 1 ...
Incoming Signal
90°
Local Oscillator (CORDIC)
Pilot / Data Correlator 1 Output
Pilot / Data Correlator 2 Output
Data Correlator Output
Data Code Correlator
GNSS Data Code

Fig. 7. Block diagram of the Tracking HW accelerator.

- The GPS L5 C/A multicorrelator consists of four correlators: Three of them are used to obtain the Early, Prompt and Late (E, P and L). The fourth correlator is used when the pilot signal and the data signal are tracked simultaneously.
- The Galileo E5 multicorrelator consists of four correlators: Three of them are used to obtain the Early, Prompt and Late (E, P and L). The fourth correlator is used when the pilot signal and the data signal are tracked simultaneously.

The HW accelerator modules are compatible with the IP-XACT specification [14]. Therefore the HW accelerator modules can easily be reused or replaced by other IP-XACT compliant modules.

4. PROOF-OF-CONCEPT PROTOTYPE IMPLEMENTATION

4.1. Main Features

In order to facilitate the development, debugging and testing of the GNSS receiver, a prototype has been implmented with the following characteristics:

- Possibility to demodulate and decode combinations of the following signals: GPS L1 C/A, Galileo E1, GPS L5 and Galileo E5.
- 48 Multicorrelators: 12 multicorrelators for GPS L1 C/A, 12 multicorrelators for Galileo E1, 12 multicorrelators for GPS L5 and 12 multicorrelators for Galileo E5.
- Sampling Frequency: 12.5 Msps.
- 8 bits per complex sample.

The prototype is based on the Xilinx Zynq-7000 Z7035 SoC, which has the characteristics shown in Table 1. The resource occupation in the PL is shown in Table 2. The most used resource is the Block RAM memory. The Block RAM memory is mainly needed to implement the temporary input sample buffers and to store the PRN code and the FFT of the PRN code in the PL.

The prototype makes use of an ADRV 361-Z7035 System-on-Module (SoM) composed of a dual-channel 70 MHz to 6 GHz front-end AD 9361 IC with integrated 12-bit ADCs attached to a Xilinx Zynq-7000 Z7035 SoC.

Resource	Feature
CPU	Dual-core ARM Cortex-A9
Logic Cells	275K
Block RAM	17.6 Mb
DSP Slices	900
Maximum IO Pins	362

Table 1. Xilinx Zynq-7000 Z7035 SoC

Resource	Utilization	Available	Utilization %
LUT	103762	171900	60.36
LUTRAM	2580	70400	3.66
FF	116786	343800	33.97
BRAM	484.5	500	96.90
DSP	759	900	84.33

Table 2. Resource Utilization

In order to facilitate the development, debugging and testing of the GNSS receiver, we use the ADRV1CRR-BOB breakout carrier board, which implements suitable standard connectors for power supply, Ethernet communications, USB and JTAG programming interfaces, among other debugging features. Figure 8 shows the SoM mounted on the ADRV1CRR-BOB breakout board. The PS runs on PetaLinux [17]. The PetaLinux Software Development Kit (SDK) is a Xilinx development tool that contains everything necessary to build, develop, test and deploy Embedded GNU Linux systems.

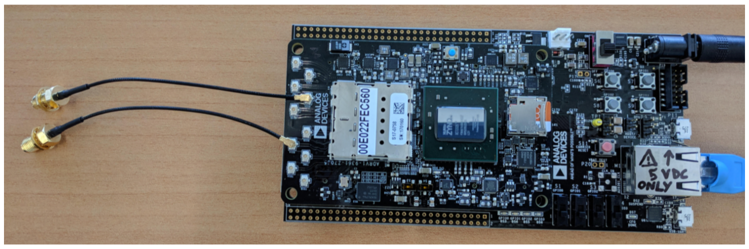

Fig. 8. Picture of the ADRV9361-Z7035 System-on-Module plugged onto the ADRV1CRR-BOB breakout board.

All the GNSS receiver components were packaged in a 24cm x 16 cm x 5 cm plastic box, as shown in Figure 9, allowing easier handling and transportation of the prototype. The box was manufactured using a custom design and a 3D printer.

The power consumption of the fully programmed SoC is about 7 W, much less than the power consumption of laptop computers.

4.2. Sampling Frequency

A timing diagram of the HW accelerators is shown in figure 10. The Acquisition and Tracking HW Accelerators work in a loop where the following actions are carried out: The baseband software engine in the PL first configures a HW accelerator (Tconf), then the software activates the HW accelerator, then the HW accelerator captures a number of samples coming from the A/D converters (Tget_samples) and performs some calculations (Tproc). When the HW accelerator finishes its calculations, the HW accelerator interrupts the CPU. Finally, the software reads and processes the results of the HW accelerator (Tresults) and the loop starts again. In the actual implementation, the tracking HW accelerators get the samples coming from the ADC and process them simultaneously.

The Acquisition HW accelerator does not block the flow of samples coming from the A/D converter. However, the Tracking HW accelerator does block the flow of samples coming from the A/D converter whenever it is waiting for the software to read the results, process the results and reconfigure the hardware accelerator. The reason for that is that the samples coming from the

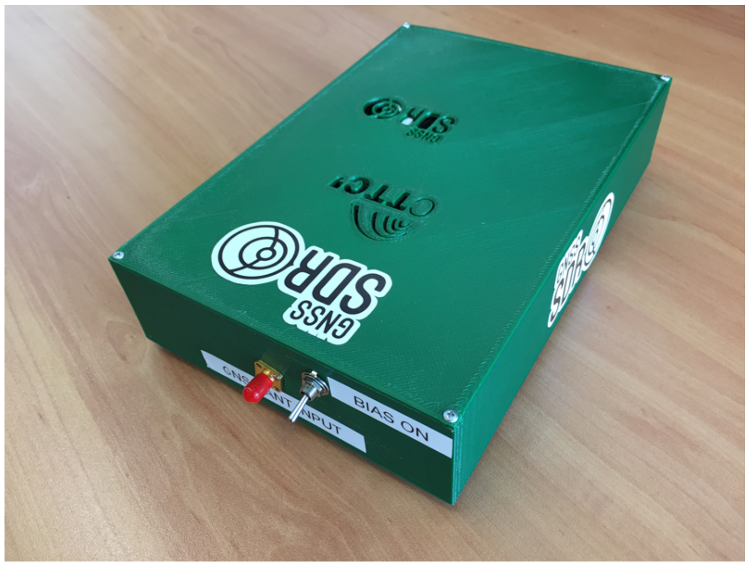

Fig. 9. Laboratory prototype.

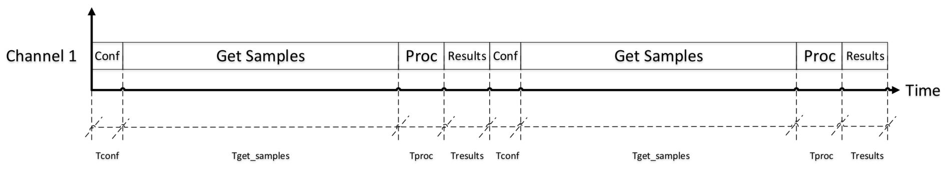

Fig. 10. Timing diagram of the HW accelerators.

A/D converter can not be lost during tracking, otherwise the receiver would loose the synchronization. Therefore the tracking HW accelerators exert back pressure to the buffer that contains the samples coming from the A/D converter.

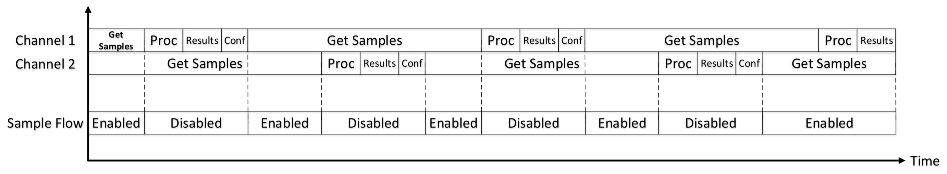

Fig. 11. Timing diagram of two tracking HW accelerators working simultaneously.

Figure 11 shows the timing diagram of two or more tracking HW accelerators running simultaneously. Whenever one of the tracking HW accelerators is exerting back pressure to the buffer that contains the samples coming from the A/D converter, none of the other tracking HW accelerators can get any samples either. If we increase the number of tracking HW accelerators running simultaneously then the percentage of time the HW accelerators can get samples from the A/D converter decreases. Thus, the overall throughput of the system decreases when the number of tracking HW accelerators increases. To avoid this problem there is a buffer at the input of every tracking HW accelerator (these buffers are not shown in figure 2). These buffers allow the HW tracking accelerators to continue to process samples whenever any of the other HW tracking accelerators is exerting back pressure to its input buffer. In this way the overall throughput of the GNSS tracking HW accelerators is not decreased when we increase the number of HW tracking accelerators working in parallel. In this situation the flow of samples is enabled most of the time, provided that the input buffers are sufficiently large (see figure 12).

The key for the system to work on real time is that the average throughput of the whole system (Hardware and Software) has to be higher than the sampling frequency itself.

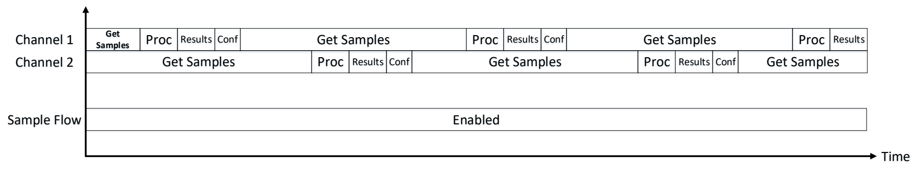

Fig. 12. Timing diagram of two tracking HW accelerators working simultaneously using input buffers.

On the hardware side, the HW accelerators work using a clock frequency of 150 MHz. The maximum average throughput of the tracking HW accelerators is one sample per 1.2 clocks when they are getting and processing samples (during Tget + Tproc in figure 10) . Therefore the GNSS receiver is designed in a way that the Tracking HW accelerators can be in a waiting state (i.e. not processing samples) 90% of time for the target sampling frequency of 12.5 Msps to be reached. This gives sufficient time to the SW to react to the interrupts coming from the HW accelerators.

As shown in Figure 5, apart from the tracking HW accelerator input buffers (not shown in 5), the GNSS receiver contains large sample buffer for each frequency band. The sample buffers are used to store the received samples temporarily. These buffers compensate unexpected temporary delays that might be caused by the SW and do not occur on a regular basis, such as delays caused by the GNU Linux OS running in the PL.

4.3. Communication between the HW Accelerator modules and the PS

The HW accelerator modules and the PS communicate by means of a high speed AXI4 bus [18]. The HW Accelerator modules are controlled using read/write registers that are accessible to the PS. These registers are mapped to a physical address memory space within the address space of the AXI4 bus. The PS uses these registers to configure and read the results of the HW accelerators. On top of that the HW Accelerator modules use interrupts to notify the PS when their tasks are completed. In this way the PS can control the whole process in real time. There is also a DMA engine (not shown in Figure 2) which is used to test the receiver in post-processing mode using GNSS signals stored in files.

5. RESULTS

The GNSS-SDR software has a number of flags that allows the user to monitor and log the status of the GNSS receiver at various points in the signal path. In this way the behavior of the HW accelerator modules can be analyzed.

Figure 13 shows the results obtained when using the Tracking HW accelerators to track the GPS L1 C/A signal. Figure 13 shows the status of various variables at the GNSS receiver for one of the channels. At approximately 1.5 s the GNSS receiver transitions from acquisition to tracking and the channel shown in figure 13 starts tracking the detected satellite. Figure 13 shows the following information:

- The decoded bits of the navigation message
- A polar scatter plot of the I and Q signals. This plot is useful to check that the Doppler wipe–off is done correctly.
- The results of the Early, Prompt and Late correlators. We can see that the results of the P correlator have the highest values, which means that the receiver is correctly synchronized.
- Various plots showing the behavior of the PLL and the DLL, the estimated Doppler frequency, the carrier to noise density ratio and the estimated code frequency of the received signal.

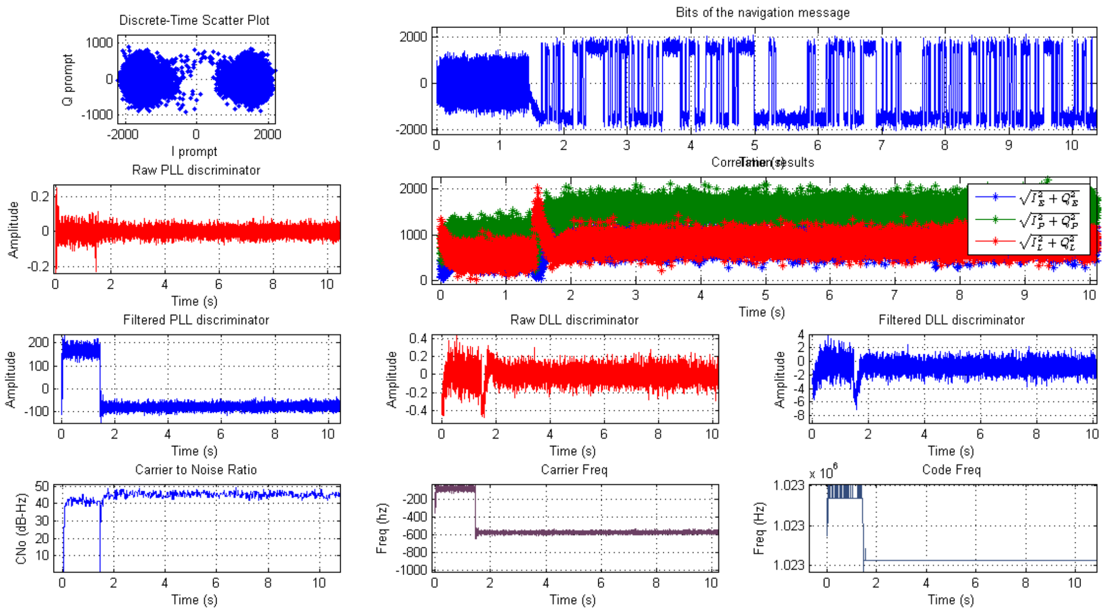

Fig. 13. Test results of the GNSS receiver using the GPS L1 signal.

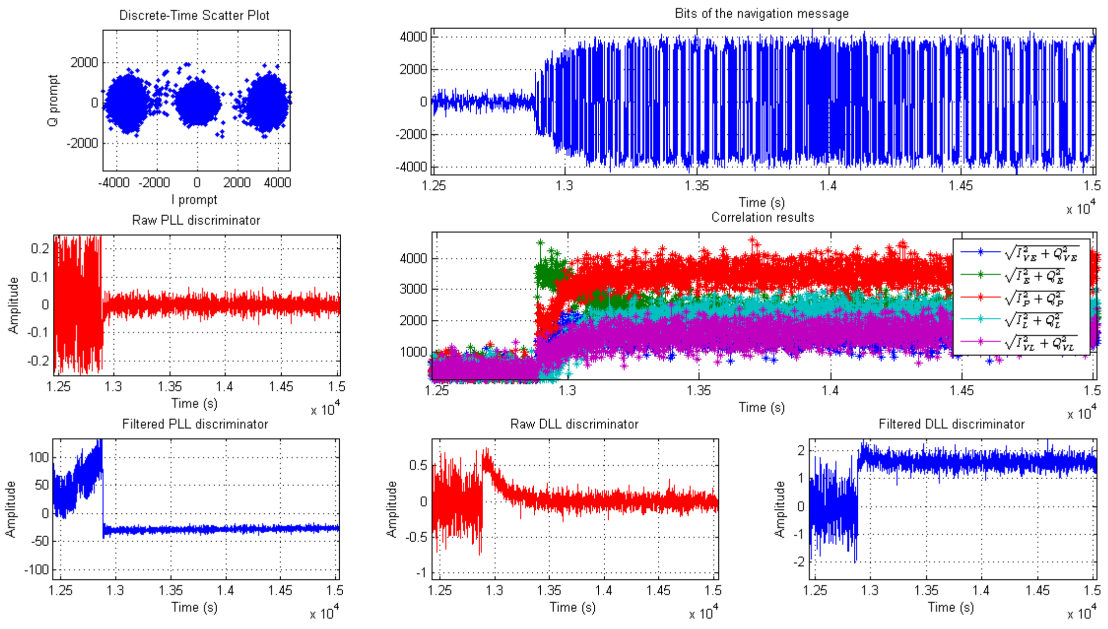

Fig. 14. Test results of the GNSS receiver using the Galileo E1 signal.

Figure 14 shows the results obtained when using the Tracking HW accelerators to track the Galileo E1 signal. Figure 14 shows the status of various variables at the GNSS receiver for one of the channels. At approximately 1.3s the GNSS receiver transitions from acquisition to tracking and the channel shown in figure 14 starts tracking the detected satellite. Figure 14 shows the following information:

- The decoded bits of the navigation message
- A polar scatter plot of the I and Q signals. This plot is useful to check that the Doppler wipe–off is done correctly. The points centered around the origin are caused by a false alarm where the channel started tracking a non-existent satellite.
- The results of the Very Early, Early, Prompt, Late and Very Late correlators. We can see that the results of the P correlator have the highest values, which means that the receiver is correctly synchronized.
- Various plots showing the behavior of the PLL and the DLL.

Table 3 shows how the objectives listed in section 2 are fulfilled.

6. CONCLUSIONS

This work shows that it is feasible to implement a flexible, portable low-cost and low power consumption GNSS-SDR-based receiver using a programmable System-on-Chip. The inherent parallelism of the PL allows the GNSS receiver to track a large number of GNSS satellites simultaneously and the internal status of the PL can be monitored for testing and research. By offloading the most computationally demanding tasks from the PL to the PS, the power consumption used for the digital signal processing tasks is reduced to 7 W. On top of that the implemented prototype makes use of the ADRV 361-Z7035 System-On-Module, which can be packaged in a portable-sized box.

In the proposed implementation, the number of GNSS satellites that can be tracked simultaneously is limited by the workload in the PS. A way to decrease the workload in the Processing System is to transfer more signal processing functions to the PL and to reduce the communication between the PS and the PL. The communication can be reduced by implementing extended tracking integration time in the PL. The GNSS-SDR-based receiver implemented in an all-programmable System-on-Chip also provides a way to asses the performance of the RF front-end, as the RF front-end can be replaced by modifying the interface between the RF front-end and the HW accelerators in the PL.

The proposed receiver takes advantage of the GNSS-SDR software features while keeping the flexibility and reprogrammability of a software based design. The System-on-Chip implementation of the GNSS receiver fulfills the requirements for a real-time portable software-defined GNSS receiver.

ACKNOWLEDGEMENTS

This work was supported by the Spanish Ministry of Economy and Competitiveness through project TEC2015-69868-C2-2-R (ADVENTURE).

Requirement	**Comments**
Possibility to either use synthetically generated or real-life GNSS signals	The GNSS receiver has a connector that can be used with an antenna, a signal generator, or raw samples stored in files.
Possibility to process signals either in real time or in post-processing time	The PL contains a DMA and a switch that can be used to either process signals coming from the antenna or function generator, or signals stored as files in a SD card connected to the GNSS receiver.
Possibility to use interchangeable RF front-ends	Recompiling the GNSS receiver for another RF Front-End only requires changes in the SAMPLE_SW (see Figure 2). Minor changes in the SW running in the PL for the RF Front-End configuration might also be required.
Possibility to define custom receiver architectures	The whole GNSS receiver is reprogrammable. The PL is reprogrammable and the PS is reprogrammable too. Therefore the signal processing algorithms can easily be changed using SW.
Possibility to easily define / interchange implementations and parameters for each processing block	The HW accelerator modules are compatible with the IP-XACT specification [14]. Therefore the HW accelerator modules can easily be reused or replaced by other IP-XACT compliant modules.
Availability of operation modes, as combinations of single/multiple frequency bands, single/multiple constellations	The HW Accelerators can decode the followings GNSS signals: GPS L1 C/A, Galileo E1, GPS L5 and Galileo E5. Any combination of two GNSS signals can be tracked simultaneously. Figure 13 and figure 14 show the test results when the receiver is decoding the GPS L1 C/A and the Galileo E1 signals.
Testeability	The states and signals of the HW Accelerator modules can be monitored using flags in the GNSS-SDR SW [19]. As an example, figure 13 and figure 14 show various internal states of the GNSS receiver when tracking GPS L1 C/A and Galileo E1 signals
Maintainability	The GNSS HW Accelerators are programmed in VHDL [20], using the standard Xilinx programmable tools [19]
Reliability	The HW Accelerators are tested using VHDL simulation and the whole GNSS receiver is tested to ensure proper operation. On top of that the HW Accelerators are automatically verified against design rule checkings (DRC) by the design tools. The DRC determines whether the physical layout of the HW Accelerator modules satisfy a series of recommended parameters
Small Form Factor	The GNSS Receiver is packaged in a 24 cm x 16 cm x 5 cm box, which makes it suitable for testing in the field
Low Power Consumption	The power consumption of the SOC is about 7 W, which makes it suitable for testing in the field.

Table 3. Design Requirements.

REFERENCES

[1] C. Fernández-Prades et al., "GNSS-SDR - An open source Global Navigation Satellite Systems software-defined receiver," Available online at `https://gnss-sdr.org/`, Accessed: November 15, 2018.

[2] C. Fernández-Prades, J. Arribas, M. Majoral, J. Vilà-Valls, and A. García-Rigo, "An open path from the antenna to scientific-grade GNSS products," in *Proc. of the 6th ESA Intl. Colloquium on Scientific and Fundamental Aspects of GNSS / Galileo*, Valencia, Spain, Oct. 2017, pp. 1–8, doi: 10.5281/zenodo.1144104.

[3] Ettus Research, "The leader in Software Defined Radio (SDR)," Website at `https://www.ettus.com/`, Accessed: November 15, 2018.

[4] European Global Navigation Satellite Systems Agency, "AUDITOR: Advanced Multi-Constellation EGNSS Augmentation and Monitoring," [Online]. Available: `https://auditor-project.eu/index.html`, Accessed: November 15, 2018.

[5] M. Pérez-Ruiz and S. K. Upadhyaya, *GNSS in Precision Agricultural Operations*, chapter New approach of indoor and outdoor localization systems, pp. 3–26, InTech, London, UK, 2012, doi: 10.5772/50448.

[6] A. Ruckelshausen, P. Biber, M. Dorna, H. Gremmes, R. Klose, A. Linz, R. Rahe, R. Resch, M. Thiel, D. Trautz, and U. Weiss, "BoniRob: An autonomous field robot platform for individual plant phenotyping," *Precision Agriculture*, vol. 9, pp. 841–847, Jan. 2009.

[7] P. J. Zarco-Tejada, J. A. J. Berni, L. Suárez, and E. Fereres, "A new era in remote sensing of crops with unmanned robots," *SPIE Newsroom*, pp. 1–3, Jan. 2008, doi: 10.1117/2.1200812.1438.

[8] J. Roselló, P. Silvestrin, R. Weigand, S. d'Addio, A. García Rodríguez, and G. López Risueño, "Next generation of ESA's GNSS receivers for Earth Observation satellites," in *6th ESA Workshop on Satellite Navigation Technologies (Navitec 2012) & European Workshop on GNSS Signals and Signal Processing*, Noordwijk, Netherlands, Dec. 2012, pp. 1–8, doi: 10.1109/NAVITEC.2012.6423104.

[9] Y. Fazliani, M. Unwin, and P. Jales, "Satellite-borne remote sensing using space GPS/GNSS receiver for reflectometry," in *6th ESA Workshop on Satellite Navigation Technologies (Navitec 2012) & European Workshop on GNSS Signals and Signal Processing*, Noordwijk, Netherlands, Dec. 2012, pp. 1–4, doi: 10.1109/NAVITEC.2012.6423102.

[10] A. Soghoyan, A. Suleiman, and D. Akopian, "A development and testing instrumentation for GPS software defined radio with fast FPGA prototyping support," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 8, pp. 2001–2012, August 2014.

[11] Xilinx, Inc., "SoCs, MPSoCs & RFSoCs," Available online at `https://www.xilinx.com/products/silicon-devices/soc.html`, Accessed: November 15, 2018.

[12] 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.

[13] Xilinx, Inc., "Intellectual property," Available online at `https://www.xilinx.com/products/intellectual-property.html`, Accessed: November 15, 2018.

[14] "IEEE standard for IP-XACT, Standard Structure for Packaging, Integrating and Reusing IP within tool flows," *IEEE Std. 1685-2014*, pp. 1–510, 2014, doi: 10.1109/IEEESTD.2014.6898803.

[15] GNU Radio, "The Free and Open Software Radio Ecosystem," Website at `https://www.gnuradio.org/`, Accessed: November 15, 2018.

[16] K. Borre, D.M. Akos, N. Bertelsen, P. Rinder, and S.H. Jensen, *A Software-Defined GPS and Galileo Receiver*, Birkhäuser, Boston, MA, 2012.

[17] Xilinx Wiki, “Petalinux OS,” Website at `http://www.wiki.xilinx.com/PetaLinux`, Accessed: November 15, 2018.

[18] Arm Limited, “AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite Documentation,” Available online at `https://developer.arm.com/docs/ihi0022/d`, Accessed: November 15, 2018.

[19] Xilinx, Inc., “Hardware Development,” Available online at `https://www.xilinx.com/products/design-tools/hardware-zone.html`, Accessed: November 15, 2018.

[20] “IEEE Standard VHDL Language Reference Manual,” *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pp. c1–626, Jan 2009, doi: 10.1109/IEEESTD.2009.4772740.