Software-Defined Radio: Bridging the Analog–Digital Divide

This paper describes the early emergence and evolution of software radio.

By Raquel G. Machado, Student Member IEEE and
Alexander M. Wyglinski, Senior Member IEEE

ABSTRACT | In this paper, we present the evolution of software-defined radio (SDR) technology and show how it is currently at the forefront of numerous advances within the wireless sector, enabling new applications considered unrealizable only a decade ago. Specifically, this paper focuses on SDR from a discrete-time sampling perspective and discusses the efforts that are currently being pursued in order to further bridge the gap between these discrete-time samples, the hardware used to generate this information from continuous-time over-the-air signaling waveforms, and the software and digital logic used to process these samples into digital data via baseband processing. Given the extensive deployment of SDR technology across a growing number of applications, such as national defense, public safety, connected vehicles, education, and scientific research and development activities, it is vitally important that the wireless community understands the features, advantages, and limitations of this technology. With its versatility, cost, and functionality continuously improving, SDR has become a viable solution for prototyping wireless transceivers and networks that are much more tailored to specific applications and performance requirements relative to available off-the-shelf wireless solutions. To highlight the advantages and current issues with SDR technology, this paper presents several examples using a recently released, commercially available SDR platform.

KEYWORDS | Analog–digital divide; digital domain; FMCOMMS; GNU radio; software-defined radio (SDR); software-defined radio platform; software domain; software interface; software radio

I. INTRODUCTION

Advances and innovation within the wireless sector have always been closely coupled to corresponding improvements in digital technology, including computing devices. Until the 1950s [1]–[3], wireless systems operated exclusively in the analog domain, where communications functions such as modulation and filtering were performed using analog circuits and components. With the rapid evolution of digital technology, especially analog-to-digital and digital-to-analog converters, it became possible to perform these same baseband communication functions partially or entirely within the digital domain, thereby greatly reducing cost, enabling mass production of these transceivers, and providing a greater flexibility and system functionality. This capability of communication systems to transcend the analog-digital divide is a defining moment of the Information Age that has enabled the ubiquitous wireless data access to which today’s society is accustomed.

The first wireless devices that employed digital technology were based on nonprogrammable, static designs that used application-specific integrated circuits (ASICs)
began experimenting with wireless transceivers that pos-
modulation, and other baseband operations. It was during
the wireless system, such as filtering, data compression,
rammed with a static set of operations to be performed by
systems. However, these computing devices were prog-
into the baseband digital implementation of these wireless
processors (DSPs) [5], they were increasingly incorporated
into the baseband digital implementation of these wireless
systems. However, these computing devices were pro-
grammed with a static set of operations to be performed by
the wireless system, such as filtering, data compression,
 modulation, and other baseband operations. It was during
the 1970s that wireless communication systems engineers
began experimenting with wireless transceivers that pos-
sessed programmable, or software-defined, attributes. In the
early 1980s, digital baseband radios with programmable
features were starting to be prototyped [6], and in the early
1990s the first large-scale SDR platforms, SpeakEASY 1 and
SpeakEASY 2 [7], were developed. These platforms used an
assortment of computing technology, including multiple
DSP platforms and field programmable gate array (FPGA)
technology [8], [9]. Since then, there has been substantial
research and development efforts by industry, government,
and academia with respect to creating more versatile,
powerful, functional SDR platforms [10], [11].

At the time of the writing of this paper, the latest SDR
platforms available boast several significant capabilities,
including supported large data bandwidths, small form
factor, and high computational horsepower. Despite these
advances as well as the availability of numerous SDR solu-
tions to choose from, there are several difficulties that
prevent this technology from being accessible or usable by
many people within the wireless sector.

1) **Interdisciplinary nature of SDR systems:** SDR
platforms are complex systems, and designers
must either possess a breadth of knowledge span-
ing several disciplines or work with a team of
individuals who each possess expertise in one or
more of these disciplines [12]. As a result, a signif-
ificant time investment is needed to understand
how these systems work and how designs can be
applied to them in order to prototype wireless
systems.

2) **Steep learning curves with design software:**
The software tools needed to design, implement,
and test the baseband functionality of the SDR
platform potentially can be overwhelming [13],
[14]. A wide range of software solutions that can be
used to “build” wireless systems using SDR are
available, but they are often either limited in func-
tionality to make the software suite accessible to
the designer, or they allow for a high degree of
functionality but are overly complex and can only
be used by software experts.

3) **Software design portability to new SDR plat-
forms:** Given the amount of time, resources, and
effort needed to design various modules of an SDR
platform, it is advantageous if the same code can
be ported to other SDR systems, as well as next-
generation SDR implementations that use newer
computing technology [15]. Without code porta-
bility, the process of designing SDR systems is
inefficient, as each successive design must rein-
vent the wheel to fit the new platform.

4) **Software and driver limitations:** Although cur-
rent software tools designed to manipulate and
process actual real-world information are very ca-
ble, there still exist issues with respect to the
handling of large amounts of data in real time.
While analog-to-digital converter (ADC) and
digital-to-analog converter (DAC) technologies
are capable of handling giga-samples-per-second
of sampling data at reasonable cost, and computing
technology is capable of performing operations in
the gigahertz range, the current “bottleneck” for
SDR technology is the interface between samples
on the SDR hardware and the software tools on
computing device.

In this paper, we will discuss some of the current issues
impacting SDR technology and research and investigate
how they are being resolved. Specifically, we will examine
how the sampled data between the SDR platform and the
computing device is treated in order to realize a wireless
communication system, as well as study issues that affect
the performance of these systems. It is the expectation of
the authors that by exploring how the SDR platform
handles information between the information source/sink
and the radio frequency front-end (RF), greater insights
can be provided to the entire wireless community on how
these systems actually function.

The rest of this paper is organized as follows: In
Section II, a brief history of the SDR technology is provided
along with a summary of developed SDR platforms in
academia and industry. In Section III, a detailed
description of the anatomy of a typical SDR platform is presented,
from the antenna all the way to the binary information.
Section IV describes the current issues with SDR technol-
ogy in terms of sufficient software support for designing the
baseband functionality of these systems and having them
operate in real time. In Section V, we present a case study
with respect to a commercially available SDR platform that
was recently released, emphasizing the technical chal-
lenges and solutions needed in order to implement and
prototype viable wireless transceiver designs. Finally, fu-
ture opportunities for research and development, as well as
an overview of current open challenges, are presented in
Section VI.

II. BRIEF HISTORY
The term software radio was introduced by Joseph Mitola
in 1992 [16]. However, an SDR prototype had already been
presented in 1988 by Hoehler and Lang [17]. The establishment of SDR as a technology came with the first publicly funded SDR development initiative, called SpeakEasy I/II by the U.S. military [7]. The first generation of the SpeakEasy system initially used a Texas Instruments TMS320C40 processor (40 MHz), while the SpeakEasy II platform was the first SDR platform to involve a field programmable gate array (FPGA). Later, the U.S. Navy developed the digital modulator radio (DMR), a platform with many waveforms and modes that could be remotely controlled with an Ethernet interface.

Fig. 1 represents a timeline of the evolutions of both processing technology and SDR technology and shows how the development of components with higher computational power and flexibility enabled better SDR platforms. It was only after the year 2000, with powerful FPGAs and DSPs, that most of the existing SDR platforms were developed. More recently, the ARM Cortex A9 opened the possibility of accessible on-board processing, discarding the necessity of a host computer for system development.

The widespread use of wireless mobile devices has presented several significant technical challenges with respect to the myriad of different wireless standards, services, and applications that all need to be supported, often within the same operating environment. In many cases, several layers of wireless infrastructure are needed in order to support multiple concurrent networks within the same region, which requires a substantial investment in terms of equipment and other resources. Furthermore, supporting multiple wireless networks simultaneously may result in a degradation in network performance due to issues such as interference. Consequently, reconfigurable radio technologies provides an opportunity to reduce infrastructure costs, provide more efficient wireless connectivity across various networks, and offer a new dimension of operational flexibility that can ultimately enhance network performance.

In this sense, in the late 1990s, SDR spread from the military domain to the commercial sector, with cellular networks being considered the natural area of application [18]. Several companies including Vanu [19], Airspan [20] and Etherstack [21] started to develop SDR products for cellular base stations. In 2005, Vanu released the first SDR product approved by the software radio regulation: the Anywave GSM base station. The BTS (base transceiver station), the BSC (base station controller), and TRAU (transcoder and rate adaptation unit) modules of the BSS (base station subsystem) were implemented in software in the Anywave base station. Although this successful implementation brought a lot of attention to SDR technology—and it was thought at the time that SDR base stations would be key for 3G networks—the reality of a commercial usage of SDR technology still remains distant today.

However, given its versatility and cost, SDR technology has emerged as a solution for prototyping wireless transceivers and networks by taking advantage of its customizable features to accelerate development time. As a result, a growing number of applications, such as national defense [22], public safety [23], connected vehicles [24], and education [25], can benefit from SDR technology. Furthermore, it is important to combine versatile hardware components with ergonomic development software environments in order to achieve efficient prototyping and development.

One of the most commonly used SDR hardware platforms is the Universal Software Radio Peripheral (USRP) [26]. Developed by Ettus Research LLC, the USRP is a device that turns general purpose computers into flexible SDR platforms. The core of the USRP is a motherboard with four high-speed ADCs and DACs and a FPGA. The ADCs/DACs are connected to the radio front-ends (called daughterboards), while the FPGA is connected to a general purpose computer. In the Universal Software Radio Peripheral-Version 1 (USRP1) this connection is performed by a USB port, while the USRP2 series (USRP N200 and USRP N200) includes a Gigabit ethernet interface. The main principle behind the USRP is that the digital radio tasks are divided between the internal FPGA and the external host CPU. The high speed general purpose processing, like down and up conversion, decimation, and interpolation are performed in the FPGA, while waveform-specific processing, such as modulation and demodulation, are performed at the host CPU. Specifically, the USRP N210 includes a Xilinx Spartan 3A-DSP 3400 FPGA, a 100 MS/s dual ADC and a 400 MS/s dual DAC for sampling/reconstruction and high speed general purpose processing. The USRP platform can be used with both

<table>
<thead>
<tr>
<th>Year</th>
<th>Technology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990s</td>
<td>SDR Spread</td>
<td>SDR spread from military to commercial sector</td>
</tr>
<tr>
<td>2000</td>
<td>USRP Release</td>
<td>Vanu released the first SDR product approved by software radio regulation</td>
</tr>
<tr>
<td>2010</td>
<td>Improved USRP</td>
<td>USRP2 series includes Gigabit ethernet interface</td>
</tr>
</tbody>
</table>

**Fig. 1.** Processing technology versus SDR technology. Timeline of the past decades shows how the evolution of different types of processors had a great impact in the development of SDR platforms.
GNU radio and MATLAB software development environments. More recently, Ettus Research released the new X Series, a platform that contains more powerful daughter-board slots, 6 GHz with up to 120 MHz of baseband bandwidth, and a large user-programmable Kintex-7 FPGA.

Another SDR hardware platform is the Kansas University Agile Radio (KUAR) [27]. The KUAR platform was designed to be a low-cost experimental platform targeted at the frequency range 5.25 to 5.85 GHz and a tunable bandwidth of 30 MHz. The platform contains a Xilinx Virtex-II Pro FPGA board and a PCI Express 1.4 GHz Pentium-M microprocessor. These features enable almost all processes to be implemented in the platform, instead of in the host computer, which minimizes the host-interface requirements. In addition, the KUAR utilizes a modified form of the GNU Radio software framework to complete the hardware platform.

With respect to compact SDR platforms, the Maynooth Adaptable Radio System (MARS) [28] was designed to be connected to a personal computer that handles all of the signal processing algorithms. Another objective of the platform was to deliver a performance equivalent to a base station and the wireless communication standards in the frequency range 1700 to 2450 MHz. The software framework selected for initial development was the IRiS framework (Implementing Radio in Software).

Some other SDR platforms based on Virtex-II include:

1) Berkeley BEE2 [29]: has five Xilinx Virtex-II Pro FPGAs on a custom-built emulation board.
2) Japanese National Institute of Information and Communications Technology (NICT) SDR Platform [30]: contains two embedded processors, four Xilinx Virtex-II FPGAs, and RF modules that could support 1.9 to 2.4 and 5.0 to 5.3 GHz.
3) Rice University Wireless Open Access Research Platform (WARP) [31]: radios include a Xilinx Virtex-II Pro FPGA board and a MAX2829 transceiver.

While these SDR platforms are mainly used for research and experimentation, the ultimate goal of developing SDR systems capable of implementing modern communication protocols remains far from reality. The development of new hardware platforms and accompanying software interfaces is key to advances in the area and to better understanding the capabilities and limitations of the SDR technology when used in real-life communications systems.

In 2006, Lyrtech, Inc. launched the Small Form Factor (SFF) Software-defined Radio (SDR) Development Platform, developed in collaboration with Texas Instruments, Xilinx, Objective Interface Systems, Green Hills Software, The MathWorks and the Communications Research Centre Canada (CRC). This collaboration focused on the need for a portable SDR solution capable of supporting applications from the military, public safety and other commercial sectors. The SFF-SDR platform includes the TMS320DM6446 system-on-chip (SoC) from Texas Instruments and Virtex-4 SX35 FPGA from Xilinx, a 125-MSPS 14-bit dual-channel ADC, a 500-MSPS 16-bit ADC and an RF module that operates between 360 and 960 MHz.

The Reconfigurable Hardware Interface for Computing and Radio (RHINO) project was developed by the University of Cape Town, South Africa. Being open source, the project aims to provide both hardware and software support for SDR teaching and research. The platform includes a Spartan6 FPGA and a Texas Instrument Sitara AM3517ZCN ARM Cortex A8 processor. Targeting satellite-terrestrial mobile communication, Infineon developed in 2009 the XMM SDR 200 Software Defined Radio platform. This platform is based on the X-Gold SDR 20 and a SmartRF transceiver, being able to support multiple standards using the same baseband part.

Lately, a few companies have also been developing different SDR solutions for use in academia and industry. For example, Nutaq [32] developed two main SDR products, the ZeptoSDR and the PicoSDR, both of which support RF frequencies between 300 MHz and 3.8 GHz and bandwidth of 1.5 to 28 MHz. The ZeptoSDR uses a Xilinx Zynq-7 and an embedded ARM Cortex-A9, and the PicoSDR uses a Xilinx Virtex-6 and an embedded Quad-Core i7.

Within this context, Epix Solutions [33] also developed two compact SDR products, the Sidekiq and the Matchstiq, as well as another platform called Maveriq. The Sidekiq is a miniature SDR platform that relies on a Xilinx Spartan-6 LX45T FPGA with ×1 PCIe interface. It provides an RF range from 70 MHz to 6 GHz and 50 MHz RF bandwidth per channel. The Matchstiq is a compact software defined radio solution with a broadband RF transceiver. It provides two options for the integrated CPU/FPGA processing: the TI DM3730 @ 1 GHz (ARMv7/Cortex A8 + TI C64× DSP) and the Xilinx Zynq-7020 @ 800 MHz (ARMv7/Cortex A9 Dual Core). Its RF ranges from 300 MHz to 3.8 GHz and supports channel bandwidth up to 28 MHz.

III. ANATOMY OF A SOFTWARE-DEFINED RADIO

In traditional radios, all radio functionalities are performed by specialized components that execute specific functions, such as modulators/demodulators, coding/decoding, etc. In this case, all signal processing happens within the specialized hardware. The software-defined radio technology replaces some of the traditional radio components with components implemented in software.

A software-defined radio transceiver is divided into two main parts: an analog front-end, which performs the narrowband frequency downconversion followed by an analog-to-digital conversion (ADC), and the digital signal processing components, which are responsible for the remaining signal processing flow [34]. Thus, operations such as (de)modulation, filtering, and channel (de)coding are
performed in the digital domain. Fig. 2 shows the typical data flow in a software-defined radio system. In this case, nearly the entire baseband signal processing on both the transmission and receiving ends is performed in the software domain.

Ideally, if the analog-to-digital/digital-to-analog conversion can be pushed further into the RF front end, the programmability could be extended to the RF front end and an ideal software radio could be implemented [18]. The advantage of having components implemented in software is flexibility, as different frequency bands, air interface protocols, and functionalities could be upgraded through a software download instead of having to completely replace the hardware.

Thus, the ultimate goal for software-defined radio is to move the AD/DA conversion as close as possible to the antenna so that all signal processing can be done digitally. However, some technical limitations make it currently infeasible to perform the AD/DA conversion at the antenna.

A. Digital Domain

Fig. 2 shows the functional blocks that can be implemented in the digital domain of a communication system. These blocks include modulation and demodulation blocks, which perform mapping between bits and electromagnetic waveform characteristics; coding/decoding blocks, which help mitigate impairments in the wireless channel; source encoding and decoding blocks, which remove redundant information from the binary data; and channel encoding and decoding blocks, which introduce redundant information to protect transmissions from potential errors.

In an SDR platform, all of these components are implemented in software and can run in different processing venues including field programmable gate arrays (FPGAs), graphics processing units (GPUs), digital signal processors (DSP), general purpose processors (GPP), or a combination thereof. While FPGAs are computationally powerful, they are relatively power efficient but inflexible, and it is difficult to implement new modules in them. Similarly, GPUs are very computationally powerful but are difficult to use, especially when trying to implement new modules. DSPs are processors that perform specialized mathematical computations. While users can implement new modules into them with relative ease, and they are relatively power efficient, they are not well suited for computationally intensive processes and can quickly lose speed. Finally, GPPs are a popular solution for SDR implementations and prototypes due to their high level of flexibility with respect to reconfigurability. However, since GPPs are not specialized for mathematical computations, they can be very power inefficient.

Returning to the problem of where the AD/DA conversion should happen, there are a number of challenges presented by the transition from hardware radio to software radio. First, the transition from hardware to software processing results in a substantial increase in computation, which leads to increased power consumption and reduced battery life. This large power consumption is one of the key reasons why software-defined radios have not been deployed in end-user devices but are instead used in base stations and access points, which can take advantage of external power resources. Another example is the use of SDR technology in satellite communications, where several space applications would greatly benefit from features such as remote software updates and reconfigurable functionality for the formation of space-based networks. However, satellite-based SDR systems are very resource constrained resulting in few real-world implementations [35], [36]. Second, the question of where the AD/DA conversion can be performed determines what radio functions can be done in software, and hence how reconfigurable a radio can be.
B. Analog Domain

Fig. 3 shows a typical RF front-end responsible for processing the analog portion of the digital transmission [37], [38]. In the transmission signal path, the digital samples are converted into analog signal by the DAC to be input to the RF front-end; the analog signal is later mixed with high frequency carriers and modulated to a determined RF frequency and transmitted over the air. In the receiving signal path, the RF signal is captured by the antenna and brought back to base band to be processed by ADC. The RF mixing and modulation is driven by the local oscillator (LO), which generates the RF signal, which is mixed with the incoming signal. Another very important component used in radio transmission is the low-noise amplifier (LNA), which is usually located close to the antenna and is used to amplify weak signals without significantly increasing noise level.

C. Sampling and A/D—D/A Conversion

Digital transmissions are all about sampling. A continuous-time signal can be converted to a discrete-time signal using sampling, and a discrete-time signal can also be converted to a continuous-time signal using reconstruction [39], [40]. To sample a signal, instantaneous measurements are taken every $T_s$ seconds; in this sense, $T_s$ is the sampling period, and $f_s = 1/T_s$ is the sampling frequency. To reconstruct the original signal from the sampled signal, it is necessary to apply a low-pass filter on the sampled signal. However, by the Nyquist theorem, for the reconstruction to be successful, $f_s$ needs to be higher than two times the analog signal’s bandwidth, which is called the Nyquist frequency. The components responsible for sampling and reconstructing the signal are the DAC and the ADC.

As previously mentioned, moving the analog-to-digital and digital-to-analog conversions closer to the antennas is the ultimate goal of software-defined radio technology. In order to do so, the major challenge lies in the DAC and ADC’s sampling capabilities. To digitize an RF signal it is necessary to sample it at least at the Nyquist frequency, and the higher the data rate of the signal, the higher the resolution required to capture the information. For example, an 802.11n Wi-Fi channel is 40 MHz wide [41], which means the ADC has to digitize 80 MHz of signal bandwidth, resulting in a sample rate of at least 160 million samples per second (Msps).

For these reasons, the development of SDR platforms is closely related to the development of more powerful ADCs and DACs. In Table 1, we provide a list of SDR platforms and their corresponding sampling capabilities. It is possible to note that the advent of ADCs and DACs with higher maximum sampling values contributed to the rapid development of several SDR platforms in sequence. From the SPEAKeasy platform in the 1990s to the FMCOMMS 2 and USRP-X Series, the possible usable bandwidth increased 3000 times. This allows the implementation of modern wireless transmission standards in SDR platforms and pushes the edge of the software defined radio technology applications.

D. GNU Radio

GNU Radio is a free software toolkit licensed under the GPL for implementing software-defined radios. Initially, it was mainly used by amateur radio enthusiasts, but it later gained significant interest from wireless researchers, and today it has a large community of users and contributors. It supports Linux natively, and packages are precompiled for the major Linux distributions.

GNU Radio provides means for performing the digital signal processing portion of a communication system design. Several software algorithms include filters, channel codes, synchronization elements, equalizers, demodulators, decoders, and many other elements. It is possible to use these components as building blocks of a communication system; GNU radio not only provides these blocks but also a method of connecting them together. Communications systems can be implemented by using the already available blocks or by developing new components for the software platform.

The data management is performed by the software environment and is transparent to the user. In this sense, both the input data type for receivers and output data type for transmitters are complex baseband samples. In general, GNU Radio applications are primarily written using the

---

**Table 1 Sampling Capabilities of Different SDR Platforms**

<table>
<thead>
<tr>
<th>SDR Platform</th>
<th>Sampling Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEAKeasy</td>
<td>200 Kbps</td>
</tr>
<tr>
<td>USRP 1</td>
<td>64 Ms/s dual ADC and 128 Ms/s dual DAC</td>
</tr>
<tr>
<td>KUAR</td>
<td>105 Msps ADC and 100 Msps DAC</td>
</tr>
<tr>
<td>USRP 2</td>
<td>100 Ms/s dual ADC and 400 Ms/s dual DAC</td>
</tr>
<tr>
<td>FMCOMMS 1</td>
<td>250 Msps ADC and 1 Gbps DAC</td>
</tr>
<tr>
<td>FMCOMMS 2</td>
<td>640 Msps ADC and 320 Msps</td>
</tr>
<tr>
<td>USRP-X Series</td>
<td>640 Msps ADC and 320 Msps</td>
</tr>
</tbody>
</table>
IV. CURRENT SDR CHALLENGES

As previously mentioned, the sampling capabilities of ADs and DAs converters are a key ingredient for the implementation of SDR systems and prototypes. The ability to digitize high frequencies is fundamental for bridging the analog domain with the digital domain and for leveraging the flexibility made available by the digital hardware and software. On the other hand, the software and digital logic implementation imposes a computation burden on the platform that leads to an increase in power consumption. This tradeoff means that designers must choose either flexibility or energy efficiency, and many designers must therefore use different SDR solutions for different applications.

A second consideration concerns interface to the RF portion of the digital transceiver [42]. It is challenging to design antennas over a wide range of frequencies since the antennas propagate signals differently for different frequencies transmitted. In addition, the electronic circuits that connect the antennas to the rest of the circuit, called baluns, are optimized for different antennas and must be matched for optimal power performance. This requirement complicates the radio design and prevents the implementation of systems with very different frequency ranges on the same SDR platform. Usually it is necessary to choose between baluns that have an excellent linear response on a narrow frequency band or baluns that have reasonable response on a broad frequency range.

Another issue related to sampling rates is the timing and synchronization required within the radio [42]. In order to avoid confusion and mismatch of sampling rates, it is important that the rates of the processing devices (Microprocessors, FPGAs, DSPs) that are running the digital signal processing blocks are synchronized with the clock of the hardware components in the analog domain of the digital transmission. The hardware and software clocks should be equivalent and translatable to avoid confusion and mismatch in the sampling rates used. This synchronization is major concern to software environment designers but should be transparent to the users.

In addition to technical issues, a major challenge related to SDR implementations involves the software environments used for the system design. Two commonly used software tools for the design and prototyping of SDR implementations are Matlab [43] and GNU Radio [44]. Matlab is a versatile software tool with substantial user support and many different modules and processing blocks already available for usage. However, Matlab is proprietary and requires an initial investment in the designing process. As previously mentioned, GNU Radio is a free software toolkit that provides a flexible environment for the design of communication systems. While it does not provide as many blocks as Matlab, users can change pre-existing blocks and develop new ones. However, this increased flexibility can also be problematic since the learning curve to utilize GNU Radio and its capabilities is steep, as it requires considerable programming experience.

V. CASE STUDY: FMCOMMS RF FRONT-END WITH THE ZEDBOARD

A. FMCOMMS SDR Platform

Several SDR architectures have been developed and are commercially available, to enhance the prototyping of new wireless technologies and advance the current state-of-the-art in wireless and networking communications systems. While there exist several well-known SDR platforms that are commercially available [26]–[28], [31], [45], many of these systems are designed primarily for conducting fundamental research and experimentation and not for development of actual commercial products and prototypes. The creation of new SDR prototyping platforms and their associated software interfaces is the key for enabling continued advances in the wireless sector and for gaining a better understanding of both the capabilities and limitations of SDR technology when used in real-world applications.

Analog Devices, Inc. (ADI) has been in the SDR market since the 1990s and currently serves a wide range of customers. In 2012, ADI designed the FMCOMMS1 high-speed analog module [46] to demonstrate the latest generation of high-speed data converters. The FMCOMMS1 module displays sampling-level processing capabilities of 1 GS/s and enables radio frequency (RF) applications across a wide frequency range. In addition, it is customizable across software, and without any hardware changes it is possible to use different configurations that can be applied in many different communications applications. Following the FMCOMMS1, ADI developed another RF front-end: the AD-FMCOMMS2-EBZ [47]. The FMCOMMS2 is a high-speed analog module incorporating the AD9361, a high performance, integrated, RF agile transceiver. It is intended for use in applications, such as 3G and 4G base stations. The device integrates an RF front-end portion with a mixed-signal baseband section and frequency synthesizers.

Although ADI’s FMCOMMS modules are currently available to the community, there is no substantial software support for these products in terms of a software environment for communications system design and prototyping, e.g., GNU Radio [44], that can enable the community to use these platforms for over-the-air experimentation.

B. FMCOMMS 1 RF Front End

The FMCOMMS1 board (Fig. 4) consists of two main functional partitions, the transmit path and the receive path [46]. Table 2 shows the main components of the board in both paths, as well as their specifications.
In the transmit direction, the system converts complex in-phase (I) and quadrature (Q) signals into a corresponding RF signal. In the first stage, the digital-analog converter (DAC) interpolates the complex samples and translates the signal to a baseband signal. From the DAC output, the signal is sent to a quadrature modulator and again translated to the specified RF output frequency. The analog signal also passes through an image rejection filter and an amplifier for $+20$ dB gain. Finally, the RF output power can be controlled by adjusting the baseband data.

In the receive direction, the RF signal is demodulated by the direct-conversion quadrature and brought back to baseband frequency. The resulting I and Q baseband signals are filtered and amplified to obtain $4.5$ to $20.25$ dB of gain. Before the ADC, there is an anti-alias filter, which removes harmonics and other out-of-band signals. Finally, the samples are converted to the digital domain by the ADC and the complex samples are processed by the direct memory access (DMA) interface.

### C. FMCOMMS 2 RF Front End

Since the AD9361 chip operates in the 70 MHz to 6 GHz range, it covers most of the frequency bands used for most radio standards. As mentioned previously, by changing the sample rate, digital filters, and decimation and interpolation factors inside the AD9361, the system is capable of supporting channel bandwidths up to 56 MHz. The data path consists of a low noise amplifier (LNA), a demodulator, a low-pass filter (LPF), an ADC and digital filters in the receiver portion, and digital filters, a DAC, and modulators in the transmitter portion. The key features of receive and transmit paths are shown in Table 3.

The AD9361 transmit signal path receives 12-bit 2’s complement data in I-Q format from the AD9361 digital interface, and each channel passes this data through four digital interpolating filters to a 12-bit DAC. In order to obtain different data rates, each of the four interpolating filters can be bypassed. Fig. 6(a) shows a block diagram for the AD9361 TX signal path. The TX FIR filter is a programmable poly-phase FIR filter. Its number of taps is

### Table 2 Listing of FMCOMMS1 Module Hardware Components Present in Both Transmit and Receive Paths, With Respective Specifications. We Show the Part Numbers for Each Component and the Descriptions and Specifications. Source: [46]

<table>
<thead>
<tr>
<th>Components</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD9122</td>
<td>Dual, 16-Bit, 1200 MSPS, Digital-to-Analog Converter</td>
</tr>
<tr>
<td>ADL5375</td>
<td>400 MHz to 6 GHz Broadband Quadrature Modulator</td>
</tr>
<tr>
<td>ADF4351</td>
<td>Wideband Synthesizer with Integrated VCO (35MHz to 4400MHz)</td>
</tr>
<tr>
<td>ADL5602</td>
<td>50 MHz to 4.0 GHz RF/IF Gain (20dB) Block</td>
</tr>
<tr>
<td>ADL5380</td>
<td>400 to 6000 MHz Quadrature Demodulator, 500MHz bandwidth</td>
</tr>
<tr>
<td>AD8366</td>
<td>DC to 600 MHz, Dual-Digital Variable Gain (4.5dB to 20.5dB) Amplifiers</td>
</tr>
<tr>
<td>AD9643</td>
<td>14-Bit, 250 MSPS, Dual Analog-to-Digital Converter (ADC)</td>
</tr>
<tr>
<td>ADF4351</td>
<td>Wideband Synthesizer with Integrated VCO (35MHz to 4400MHz)</td>
</tr>
</tbody>
</table>

**Fig. 4. FMCOMMS1 module is an RF front-end (RF FE) board. Here we have the board and markers highlighting important components of the platform, such as the ADC, DAC, (de)modulators and amplifiers. Source: [46]. (a) Top view; (b) bottom view.**
configurable for a minimum of 16 and a maximum of 128 taps, and its gains can be set to values between $-6$ dB and 0 dB. The filters HB1 and HB2 are fixed-coefficient half-band interpolating filters, while HB3/INT3 provides two different choices of fixed-coefficient interpolating filters; it can interpolate by a factor of 2 or 3. After the DAC, there are two analog filters: the BB (Base-Band) LPF, a third-order Butterworth low-pass filter, and the secondary LPF, a single-pole low-pass filter. Both analog filters possess a programmable 3 dB corner frequency and are responsible for reducing spurious outputs and providing general low pass filtering prior to up-conversion. Note that both the I and the Q paths are schematically identical to each other.

The AD9361 RX signal path passes downconverted signals (I and Q) to the baseband receiver section. The baseband RX signal path is composed of two programmable analog low-pass filters, a 12-bit ADC, and four stages of digital decimating filters. Each of the four decimating filters can be bypassed. Fig. 6(b) shows a block diagram for the AD9361 RX signal path. The RX LPF is a single-pole low-pass filter, and the BB LPF is a third-order Butterworth low-pass filter. As in the TX signal path, they are responsible for reducing spurious signal levels and for providing low pass filtering prior to up-conversion. The digital filters HB3/DEC3, HB2, HB1, and RX FIR are equivalent to the digital filters present in the TX signal path.

**Table 3** Listing of FMCOMMS2 Key Features [47]

<table>
<thead>
<tr>
<th>Receive Path</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF</td>
<td>2.5dB @1GHz</td>
</tr>
<tr>
<td>ADC</td>
<td>Continuous time sigma-delta, 640MSPS</td>
</tr>
<tr>
<td>Digital Filters</td>
<td>128 complex taps, decimation between 2 and 48</td>
</tr>
<tr>
<td>Gain</td>
<td>1dB step size, 80dB analog range, 30dB digital range (post ADC scaling)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transmit Path</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC</td>
<td>320MSPS</td>
</tr>
<tr>
<td>Digital Filters</td>
<td>128 complex taps, decimation between 2 and 48</td>
</tr>
<tr>
<td>Gain</td>
<td>0.25dB step size, 86dB range</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Clocking &amp; Power</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>40MHz crystal</td>
</tr>
<tr>
<td>ADP1755</td>
<td>Low dropout, linear regulator, 1.2A, 1.6 to 3.6V</td>
</tr>
<tr>
<td>ADP7104</td>
<td>High accuracy, 500mA LDO</td>
</tr>
<tr>
<td>ADP190</td>
<td>High side power switch, 1.1 V to 3.6V input range</td>
</tr>
</tbody>
</table>
path. Note that both the I and Q paths are schematically identical to each other.

As already mentioned, the four blocks leading up to the DAC in Fig. 6(a) form the digital filtering section of the transmit path, while the four blocks following the ADC in Fig. 6(b) comprise the digital filtering for the receive path. These programmable filters provide the bandwidth limiting required prior to conversion from digital to analog in the transmitter section and bandwidth limiting and out of band noise and spurious signal reduction after digitalization in the receiver section. They also provide interpolation/decimation to generate the correct data rates. In each filter, interpolation/decimation is performed first, followed by the filter transfer function.

D. ZedBoard and the SDR Platform

As already mentioned, the FMCOMMS modules are analog front-end hardware platforms [34] that are responsible for dealing with the RF portion of a wireless transmission. In addition to the front-end, a digital communication system also requires a radio back-end, which is responsible for the remaining signal processing operations in a receiving or transmitting chain [48]. It is in the back-end that operations such as (de)modulation, filtering, and channel (de)coding are performed, already in the digital domain. The main idea of an SDR system is to implement the digital radio back-end in software to provide some degree of reconfigurability.

We can observe in Figs. 7 and 8 that the FMCOMMS board is attached to the fabric portion of the SDR platform. This fabric can be represented by various FPGA development boards, where the software is loaded to perform the digital signal processing methods to implement the communications algorithms. Originally, the FPGA boards combined with FMCOMMS boards can use different microprocessors: ML605, KC705, ZC702, ZC706 and the ZED (Zync). For each one of these devices, ADI provides Linux support by providing drivers for the different parts
on the FMCOMMS board. In this work, the software portion of the SDR platform is implemented through the usage of Xilinx’s ZedBoard [49]. The Zedboard is a development kit based on the Zync-7000 SoC (System on a Chip) and can be combined with the FMCOMMS boards to form a complete SDR platform.

The connection between the two boards is performed through a FMC-LPC (FPGA Mezzanine Card—Low Pin Count) connector, which delivers and receives the complex samples to the DAC and to the ADC at a signaling speed supported up to 10 GB/s. Note that the complex samples in this platform can be generated from an internal direct digital synthesizer (DDS) or an external memory. The internal DDS is formed by four independent signal generators. These four signal generators are combined to create two tones (the I and Q signals) that are delivered to the DAC. Additionally, the Zedboard includes a Gigabit ethernet interface that allows remote access to the onboard system. It also provides a 4 GB SD Card that can be used to boot a Linux environment. With this feature in mind, Analog Devices provides Linux images built for the FMCOMMS modules that complete and enable the development environment.

Fig. 8. Schematic of the software-defined radio hardware. Left, FPGA development board, and right, FMCOMMS2 RF front-end. Figure illustrates the transmit and receive chain [47].

The industrial I/O subsystem [50] in the Linux kernel provides a unified framework for drivers for many different types of converters, sensors, RF devices, etc., using a number of different physical interfaces (i2c, spi, parallel, high speed serial, etc.). It does this by providing a common user space API for these types of devices. Originally targeted at analog to digital (ADCs) and digital to analog converters (DACs), it has expanded to include accelerometers, digitally controlled amplifiers (DVGAs), capacitance to digital converters, direct digital synthesis, frequency synthesizers/phase-locked loops, gyroscopes, impedance converters and network analyzers, and inertial measurement units, which are natively supported by Linux.

Consequently, the forced abstraction between hardware devices and userspace algorithms ensures that hardware can be swapped out and that algorithm development stays exactly the same. Changing hardware no longer entails the daunting task of poring through semiconductor vendor datasheets to write a new driver, but it is instead a simple matter of recompiling the Linux kernel. The IIO subsystem was accepted into the mainline Linux kernel (in ./drivers/iio/) as of April 2012, and over the last few years it has gone through numerous improvements,
and iterations, similarly to the others pieces of the Linux kernel.

The IIO Command Server [51] is a library developed to enable the communication between external devices and the FMCOMM’s components using the internet. It runs on the embedded target and translates a set of simple human readable commands into more complex sysfs and device node interactions. This allows network clients (Matlab, GNURadio, Labview, etc) to access data from the real hardware and provides a low cost, low overhead networked-based data acquisition platform. The IIO Command Server is also referenced as IIO lib 1.0. Its later version, IIO lib 2.0, was released in 2014 and it unifies the network and local accesses.

E. Interface With GNU Radio

Once the libraries and drivers that directly interface with the FMCOMMS components are installed on the ZedBoard, communications with the RF-front end can be performed by the user. To assist the user, a software development environment such as GNU Radio can be used to implement communications systems that utilize the FMCOMMS modules for real wireless transmission. However, to access and communicate with the components on the SDR platform, it is necessary to access the IIO-lib from inside GNU Radio.

As previously mentioned, GNU Radio works by using flow graphs. This means that each digital signal processing (DSP) component is represented by a block, and systems can be built by connecting the blocks according to a determined signal flow. For the GNU Radio user, it is important that the communication with the hardware portion of the radio transmission be transparent since the user is mainly interested in the digital portion of the signal processing. Consequently, the objective is to deliver or collect the complex samples to be processed or generated by the DSP blocks in a flow graph from within a GNU Radio application.

To address this problem, we developed both sink and source blocks in GNU Radio to collect and deliver the I/Q samples to other blocks in a flow graph. When necessary, it is also possible to control certain parameters of the components in FMCOMMS modules, such as transmission power, center frequency, and bandwidth. The sink block functions as an information sink and absorbs the samples from the output of flow graph and manages the transmitting process in the hardware. On the other hand, the source block manages the receiving process in the RF-front end and delivers the complex samples to the other blocks in the flow graph. In a lower level, the blocks communicate directly to the DAC and ADC buffers. Using the drivers and libraries (IIO lib) made available by ADI, it is possible to write and read directly to and from the buffers of the hardware components, taking into consideration the used data types.

In this work, we access and drive the components in the RF front-end utilizing the network through a remote host running GNU Radio and communicating via the Ethernet connection. Fig. 9 illustrates this process.

Fig. 9 shows the signal flow of the transmitting/receiving processes when the application implemented in GNU Radio is running in a remote host computer. In this case, the interactions between the GNU Radio blocks and the IIO Lib occur on a remote host computer instead of the ZedBoard. To communicate with the SDR platform a network interface is used to access the IIO-Server running on the Zynq processor. The IIO-Server program is then used to communicate with the IIO drivers and subsequently with the FMCOMMS boards.

To characterize the provided libraries and drivers and understand the mechanisms through which the data flows in this architecture, it is important to stress the software/hardware interface and verify the limits in which it starts to deteriorate. In this sense, we tested different sampling frequencies allowed by the hardware components (ADC and DAC) and verified the percentage of lost samples. The test consisted of writing I and Q samples directly to the DAC buffer, feeding them back to the receiver chain, without passing through the RF portion of the FMCOMMS board, and collecting them in the GNU Radio environment. As such, we seek to provide an upper bound for the sampling frequencies that can be supported by the use of this software interface with the FMCOMMS SDR platform. Table 4 shows the tested sampling frequencies and the respective percentages of samples lost during the GNU Radio processing. For each frequency tested, $10^7$ samples were analyzed.

It is easy to note that the interface starts to deteriorate when the sampling frequency of 245.76 is reached.
sampling frequency, the ADC buffer fills up too quickly for the GNU radio block to handle. For future works, the implementation of a buffer in the GNU Radio blocks should help accommodate higher sampling frequency values. However, in this case, the tradeoff between throughput and delay also needs to be analyzed.

F. Complete Interface Experimentation

To provide a complete test for the interface with GNU Radio, we again use a loop connecting the entire transmitting and receiving chains, as well as a flow graph in GNU Radio. A signal is generated using digital signal processing blocks in GNU Radio Companion tool and its I and Q samples are written to the DAC buffer and passed through the transmit chain. The signal is then fed back to the receiving chain and sampled from the ADC. After sampling the ADC, the samples are delivered to the GNU Radio environment to be processed and analyzed. The ADC buffer is sampled and the samples are processed inside the IIO Source to generate the I/Q information, which is required as an input for GNU Radio.

We implemented an experiment where DBPSK symbols are transmitted using the developed interface. Fig. 10 shows the digital blocks used in GNU Radio to generate, transmit, receive, and the analyze the digital symbols. The transmitter and receiver flow graphs appear together in this picture for illustration purposes only. During the experiment each portion is executed by a different instance of the GNU Radio Companion tool. The “Random Source” block generates random bits that are encapsulated in blocks of two by the “Packet Encoder” block. The information is then modulated and pulse-shaped using the “DPSK Mod” block. For this experiment, we used two samples per symbol and a root-raised cosine filter with an excess bandwidth of 0.35. The “Throttle” block used in both the transmitter and receiver serves as rate-limiting device for resource management in GNU Radio. The I and Q samples are then delivered to the “Fmcomms sink” block, which interfaces with the DAC component in the RF front-end. In the receiver portion, the samples are collected from the ADC into the GNU Radio environment using the “Fmcomms Source” block.

Fig. 11 shows the DBPSK symbols at different stages during the transmission. The transmitted symbols can be visualized in the “QT GUI Constellation Sink” block in the transmitter portion of GNU Radio flow graph. It is possible to see the DBPSK symbols spread over the in-phase axis given the pulse-shaping operation. The received symbols can be visualized in two manners: using the ADI IIO Oscilloscope tool running in the Linux from the ZedBoard, or using the other “QT GUI Constellation Sink” block in the receiver portion of the GNU Radio flow-graph. It can be noticed that the received symbols present a phase rotation in comparison to the transmitted symbols. This rotation is introduced by the RF cable used to connect the RF in and RF out of the FMCOMMS1 board. Even more importantly, it is possible to compare the received constellations outside and inside the GNU Radio environment and guarantee a perfect match, which shows the correct functioning of the developed interface for GNU Radio.

Table 4 Percentage of Samples Lost During the Transmission of $10^7$ Samples

<table>
<thead>
<tr>
<th>Frequencies (MHz)</th>
<th>Lost samples (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.31</td>
<td>0.07</td>
</tr>
<tr>
<td>30.72</td>
<td>0.16</td>
</tr>
<tr>
<td>61.44</td>
<td>0.28</td>
</tr>
<tr>
<td>81.30</td>
<td>0.57</td>
</tr>
<tr>
<td>98.22</td>
<td>1.34</td>
</tr>
<tr>
<td>122.88</td>
<td>2.62</td>
</tr>
<tr>
<td>163.54</td>
<td>2.73</td>
</tr>
<tr>
<td>245.76</td>
<td>14.29</td>
</tr>
</tbody>
</table>

Fig. 10. Transmitter and Receiver configurations in GNU Radio. Signal is generated and modulated in DBPSK symbols before being sent to the “Fmcomms sink”. Signal goes through the transmitting and receiving paths in the hardware platform and the received I/Q are delivered back to GNU Radio through the “Fmcomms source” block.
VI. CONCLUSION AND FUTURE DIRECTIONS

In this paper we discussed the evolution of SDR technology over the past few decades and elaborated on some of the current challenges that prevent the widespread usage of software-defined radio in commercial applications. As processing technologies and sampling components become more powerful, higher data rates can be achieved, and the implementation of modern communications standards for different applications can become feasible. In addition, we provided a case study for the characterization and development of a software interface for the FMCOMMS platform.

The FMCOMMS1 provides an RF range of 400 MHz to 4 GHz and 125 MHz channel bandwidth, and the FMCOMMS2 utilizes the AD9631, a high performance, highly integrated RF agile transceiver on a single chip that functions on a 70 MHz to 6 GHz range and provides a maximum bandwidth of 56 MHz. With such specifications, the FMCOMMS boards constitute the most powerful hardware RF analog front-ends currently available. However, the platform does not currently have substantial software support for design and prototyping in environments such as GNU Radio. In this sense, we presented an interface architecture that enables software connectivity and support for the FMCOMMS boards and the GNU Radio development environment. We also provided experiments using GNU Radio Companion and the FMCOMMS hardware platform that attest to the correct functionality of the proposed interface.

As the performed stress test suggests, the bottleneck in terms of real-time processing of broadband signals is not on the available hardware (DACs and ADCs), but on the bridge between the component’s buffers and the development software environment, i.e., Matlab and GNU Radio. To approach this issue and optimize this process, it is necessary to modify core functions and data pipes in both software environments and hardware drivers. Moreover, as the SDR platforms offer processing venues other than x86-based host computers, it is important to customize these software tools and make them available for different processing platforms, such as the ARM Cortex A9 that was in this paper.

REFERENCES


Fig. 11. Transmitted and received DBPSK symbols sent through the platform. (a) Transmitted symbols are generated and modulated in GNU Radio and sent through the transmit chain. (b) Symbols are then fed back to the receive chain and can be visualized using the ADI IIO oscilloscope tool in the ZedBoard and (c) inside the GNU Radio environment.
ABOUT THE AUTHORS

Raquel G. Machado (S’12) received the B.S. degree in computer engineering and the M.S. degree in electrical engineering from the State University of Campinas (Unicamp), Brazil, in 2008 and 2010, respectively, and the Ph.D. degree in electrical and computer engineering from Worcester Polytechnic Institute (WPI), Worcester, MA, USA, in 2014.

She is currently a Research Engineer at the Mathematical and Algorithmic Sciences Lab, France Research Center, Huawei Technologies Company, Ltd. Her research interests include the estimation and dissemination of information, software defined radio architectures, radio communication theory and vehicular communications.

Dr. Wyglinski is a member of Sigma Xi, Eta Kappa Nu, and the American Society for Engineering Education. He is a Distinguished Lecturer and Vice-President of Membership for the IEEE Vehicular Technology Society. He serves on the editorial boards of the IEEE Communications Magazine, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and the IEEE TRANSACTIONS ON COMMUNICATIONS, in addition to serving on the organizing committees of numerous international conferences.